

# Probabilistic Approaches to Stochastic Optimization

## **Dissertation**

der Mathematisch-Naturwissenschaftlichen Fakultät  
der Eberhard Karls Universität Tübingen  
zur Erlangung des Grades eines  
Doktors der Naturwissenschaften  
(Dr. rer. nat.)

vorgelegt von  
Dipl.-Phys. Maren Mahsereci  
aus Stuttgart

Tübingen  
2018

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:

23.07.2018

Dekan

Prof. Dr. Wolfgang Rosenstiel

1. Berichterstatter:

Prof. Dr. Philipp Hennig

2. Berichterstatter:

Prof. Dr. Ulrike von Luxburg

---

# Probabilistic Approaches to Stochastic Optimization

---

Maren Mahsereci  
2018

*Probabilistic Approaches to Stochastic Optimization*

© 2018 Maren Mahsereci



*In memory of Ruth Marie Jenner née Kleinheinz*

*Nothing in life is to be feared, it is only to be understood.  
Now is the time to understand more, so that we may fear less. — Marie Curie*

## Abstract

---

Optimization is a cardinal concept in the sciences, and viable algorithms of utmost importance as tools for finding the solution to an optimization problem. Empirical risk minimization is a major workhorse, in particular in machine learning applications, where an input-target relation is learned in a supervised manner. Empirical risks with high-dimensional inputs are mostly optimized by greedy, gradient-based, and possibly stochastic optimization routines, such as stochastic gradient descent.

Though popular, and practically successful, this setup has major downsides which often makes it finicky to work with, or at least the bottleneck in a larger chain of learning procedures. For instance, typical issues are:

- Overfitting of a parametrized model to the data. This generally leads to poor *generalization performance* on unseen data.
- Tuning of algorithmic parameters, such as *learning rates*, is tedious, inefficient, and costly.
- Stochastic losses and gradients occur due to sub-sampling of a large dataset. They only yield incomplete, or *corrupted information* about the empirical risk, and are thus difficult to handle from a decision making point of view.

This thesis consist of four conceptual parts.

In the first one, we argue that conditional distributions of local full and mini-batch evaluations of losses and gradients can be well approximated by Gaussian distributions, since the losses themselves are sums of independently and identically distributed random variables. We then provide a way of estimating the corresponding sufficient statistics, i. e., variances and means, with low computational overhead. This yields an analytic likelihood for the loss and gradient at every point of the inputs space, which subsequently can be incorporated into active decision making at run-time of the optimizer.

The second part focuses on estimating generalization performance, not by monitoring a validation loss, but by assessing if stochastic gradients can be fully explained by noise that occurs due to the finiteness of the training dataset, and not due to an informative gradient direction of the expected loss (risk). This yields a criterion for *early-stopping* where no validation set is needed, and the full dataset can be used for training.

The third part is concerned with *fully automated learning rate adaption* for stochastic gradient descent (SGD). Global learning rates are arguably the most exposed manual tuning parameters of stochastic optimization routines. We propose a cheap and self-contained sub-routine, called a ‘probabilistic line search’ that automatically adapts the learning rate in every step, based on a local probability of descent. The result is an entirely parameter-free, stochastic optimizer that reaches comparable or better generalization performances than SGD with a carefully hand-tuned learning rate on the tested problems.

The last part deals with noise-robust *search directions*. Inspired by classic first- and second-order methods, we model the unknown dynamics of the gradient or Hessian-function on the optimization path. The approach has strong connections to classic filtering frameworks and can incorporate noise-corrupted evaluations of the gradient at successive locations. The benefits are twofold. Firstly, we gain valuable insight on less accessible or ad-hoc design choices of classic optimizer as special cases. Secondly, we provide the basis for a flexible, self-contained, and easy-to-use class of stochastic optimizers that exhibit a higher degree of robustness and automation.

## Zusammenfassung

---

Optimierung ist ein grundlegendes Prinzip in den Wissenschaften, und Algorithmen zu deren Lösung von großer praktischer Bedeutung. Empirische Risikominimierung ist ein gängiges Modell, vor allem in Anwendungen des Maschinellen Lernens, in denen eine Eingabe-Ausgabe Relation überwacht gelernt wird. Empirische Risiken mit hoch-dimensionalen Eingaben werden meist durch gierige, gradientenbasierte, und möglicherweise stochastische Routinen optimiert, so wie beispielsweise der stochastische Gradientenabstieg.

Obwohl dieses Konzept populär als auch erfolgreich in der Praxis ist, hat es doch beträchtliche Nachteile, die es entweder aufwendig machen damit zu arbeiten, oder verlangsamen, sodass es den Engpass in einer größeren Kette von Lernprozessen darstellen kann. Typische Verhalten sind zum Beispiel:

- Überanpassung eines parametrischen Modells an die Daten. Dies führt oft zu schlechterer *Generalisierungsleistung* auf ungesehenen Daten.
- Die manuelle Anpassung von algorithmischen Parametern, wie zum Beispiel *Lernraten* ist oft mühsam, ineffizient und kostspielig.
- Stochastische Verluste und Gradienten treten auf, wenn Zufallsstichproben anstelle eines ganzen großen Datensatzes für deren Berechnung benutzt wird. Erstere stellen nur inkomplette, oder *korrupte Information* über das empirische Risiko dar und sind deshalb schwieriger zu handhaben, wenn ein Algorithmus Entscheidungen treffen soll.

Diese Arbeit enthält vier konzeptionelle Teile.

Im ersten Teil argumentieren wir, dass bedingte Verteilungen von lokalen Voll- und Mini-Batch Verlusten und deren Gradienten gut mit Gaußverteilungen approximiert werden können, da die Verluste selbst Summen aus unabhängig und identisch verteilten Zufallsvariablen sind. Wir stellen daraufhin dar, wie man die suffizienten Statistiken, also Varianzen und Mittelwerte, mit geringem zusätzlichen Rechenaufwand schätzen kann. Dies führt zu analytischen Likelihood-Funktionen für Verlust und Gradient an jedem Eingabepunkt, die daraufhin in aktive Entscheidungen des Optimierer zur Laufzeit einbezogen werden können.

Der zweite Teil konzentriert sich auf die Schätzung der Generalisierungsleistung nicht indem der Verlust eines Validierungsdatensatzes überwacht wird, sondern indem beurteilt wird, ob stochastische Gradienten vollständig durch Rauschen aufgrund der Endlichkeit des Trainingsdatensatzes und nicht durch eine informative Gradientenrichtung des erwarteten Verlusts (des Risikos), erklärt werden können. Daraus wird ein *Early-Stopping* Kriterium abgeleitet, das keinen Validierungsdatensatz benötigt, sodass der komplette Datensatz für das Training verwendet werden kann.

Der dritte Teil betrifft die *vollständige Automatisierung der Adaptierung von Lernraten* für den stochastischen Gradientenabstieg (SGD). Globale Lernraten sind wohl die prominentesten Parameter von stochastischen Optimierungsroutinen, die manuell angepasst werden müssen. Wir stellen eine günstige und eigenständige Subroutine vor, genannt 'Probabilistic Line Search', die automatisch die Lernrate in jedem Schritt, basierend auf einer lokalen Abstiegswahrscheinlichkeit, anpasst. Das Ergebnis ist ein

vollständig parameterfreier stochastischer Optimierer, der vergleichbare oder bessere Generalisierungsleistung wie SGD mit sorgfältig von Hand eingestellten Lernraten erbringt.

Der letzte Teil beschäftigt sich mit *Suchrichtungen*, die robust gegenüber Rauschen sind. Inspiriert von klassischen Optimierern erster und zweiter Ordnung, modellieren wir die Dynamik der Gradienten- oder Hesse-Funktion auf dem Optimierungspfad. Dieser Ansatz ist stark verwandt mit klassischen Filter-Modellen, die aufeinanderfolgende verrauschte Gradienten berücksichtigen können. Die Vorteile sind zweifältig. Zunächst gewinnen wir wertvolle Einsichten in weniger zugängliche oder ad hoc gewählte Designs klassischer Optimierer als Spezialfälle. Zweitens bereiten wir die Basis für flexible, eigenständige und nutzerfreundliche stochastische Optimierer mit einem erhöhten Grad an Robustheit und Automatisierung.

## Acknowledgments

---

I am sincerely and heartily grateful to my advisor Philipp Hennig for the fantastic support and thoughtful professional and personal guidance he granted me throughout the past years. Philipp is an impressive source of thoughts that reach beyond established concepts. I feel privileged to have worked in his group.

I am grateful to Ulrike von Luxburg, who supported my PhD, especially towards the end, as supervisor, as well as with guidance and valuable insight on career paths and opportunities.

I want to thank Bernhard Schölkopf for accepting me to the institute and for providing this fantastic research environment. The dynamic environment, the skilled people, and the access to science are amazing and I am thankful for the opportunity to be a part of it.

I am also thankful to the the IT-crew of the MPI, and especially to Sebastian Stark, who run everything so smoothly, including the cluster.

Research is seldom the works of a single person, and the effort is worth little without joy and laughter. I thank my present and past colleagues of the Max Planck Institute for Intelligent Systems, and especially of the Probabilistic Numerics group: Edgar Klenske, Michael Schober, Simon Bartels, Hans Kersting, Lukas Balles, Alexandra Gessner, Filip DeRoos, Motonobu Kanagawa, Frank Schneider, and Matthias Werner. I am thankful for the scientific discussions, but also for the quality time we spent together.

Finally, I would like to thank my parents Karin and Thomas for their faith in me, and Steffen, my brother, for always having my back.

*Maren Mahsereci*  
Cambridge & Tübingen, July 2018



*Uncertainty is an uncomfortable position.  
But certainty is an absurd one. — Voltaire*

# Contents

---

Prologue	1
0 Introduction	3
<b>I Preliminaries</b>	<b>15</b>
1 Gaussian Process Regression	17
1.1 Probability Calculus	17
1.2 Gaussian Distributions	18
1.3 Continuous Indexing—Gaussian Processes	20
1.4 Wiener Processes & Kalman Filters	22
2 Empirical Risk Minimization	29
2.1 Risk and Empirical Risk	29
2.2 Artificial Neural Networks	30
2.3 Iterative Optimization Routines	36
2.4 Uncertain Gradients	48
2.5 Line Searches	52
3 Quadratic Problems & Probabilistic Linear Solvers	57
3.1 Gaussian Inference on Positive Definite Matrices	58
3.2 Kronecker Algebra	59
4 Miscellaneous	63
4.1 Bayesian Optimization	63
4.2 Central Limit Theorem	65
<b>II Overfitting, Generalization &amp; Early-Stopping</b>	<b>67</b>
5 Local Distributions of Losses and Gradients	69
5.1 Likelihood for Losses and Gradients	69
5.2 Variance-Estimation from Mini-Batches	70
6 Early-Stopping Without a Validation Set	77
6.1 Overfitting, Regularization and Early-Stopping	77
6.2 When to Stop?—A Criterion Based on Gradient Statistics	79
6.3 Experiments	83
6.4 Comparison to <code>RMSPROP</code>	90
6.5 Conclusion and Outlook	94
<b>III Automated Step Size Adaptation</b>	<b>97</b>
7 Probabilistic Line Searches	99
7.1 Motivation	99
7.2 From Classic to Probabilistic Line Searches	100
7.3 Lightweight BayesOpt for Candidate Selection	104
7.4 Probabilistic Wolfe Conditions for Termination	105



7.5	Experiments	111
7.6	Conclusion and Outlook	123
IV	Kalman Filtering for Stochastic Optimization	125
8	First-Order Filter for Gradients	127
8.1	A Model for Once-Differentiable Functions	127
8.2	Diagonal Approximations	130
8.3	Experiments	133
8.4	Conclusion and Outlook	142
9	Second-Order Filter for Hessian Elements	145
9.1	A Model for Twice-Differentiable Functions	145
9.2	Recovering Classic Quasi-Newton Methods	154
9.3	Towards a Fast Solver: Low-Rank Approximations	161
9.4	Experiments	163
9.5	Conclusion and Outlook	165
	Epilogue	167
10	Conclusions and Outlook	169
	Appendix	173
A	Kronecker Algebra	175
A.1	Kronecker Products	175
A.2	Symmetric Kronecker Products	181
A.3	Anti-Symmetric Kronecker Products	186
B	Derivation of Filtering Equations for Optimization	189
B.1	Hyper-Parameter Adaptation for First-Order Filter	189
B.2	Second-Order Filter (Non-symmetric)	191
B.3	Second-Order Filter (Symmetric)	192
B.4	Low-Rank Approximation (Second-Order Filter, Symmetric)	194
C	Additional Experimental Results for <code>PROBLS</code>	207
C.1	Noise Sensitivity	207
C.2	Hyper-parameter Sensitivity	210
C.3	Noise Estimation	216
D	Detailed Pseudocode of <code>PROBLS</code>	219
E	Bibliography	231

## List of Figures

---

1	Intro: Hexagons in a Honeycomb.	3	
2	Intro: Surface-area to volume ratio in 3D.	6	
3	Intro: Orbit of a planet & Kepler's equation.	6	
4	Prelim: Cumulative distribution function.	17	
5	Prelim: Marginalization and conditioning.	18	
6	Prelim: Visualization of correlated 2D Gaussian.	18	
7	Prelim: Two-dimensional Gaussian inference.	19	
8	Prelim: Illustration of Gaussian process priors.	20	
9	Prelim: Illustration of Gaussian process posteriors.	21	
10	Prelim: Maximum marginal likelihood estimation.	25	
11	Prelim: Chart that illustrates predicting, filtering, and smoothing.	25	
12	Prelim: Wiener process, filtering and smoothing.	26	
13	Prelim: Integrated Wiener process.	28	
14	Prelim: Conceptual illustration of an empirical distribution.	29	
15	Prelim: Sigmoidal and ReLU-activation function.	33	
16	Prelim: Unit-circle of p-norms.	34	
17	Prelim: Gradient descent on 2D-quadratic functions with different condition numbers.	40	
18	Prelim: GD, GD+MOMENTUM, and BFGS on Rosenbrock.	45	
19	Prelim: Orthogonality of high dimensional Gaussian random vectors.	48	
20	Prelim: Two-dimensional Gaussian random vectors.	49	
21	Prelim: Sketch of classic line searches.	53	
22	Prelim: Bayesian optimization acquisition functions.	64	
23	Prelim: Illustration of Central Limit Theorem.	66	
24	EB-crit Empirical distribution of mini-batch gradients for $ \mathcal{B}  = 100$ .	73	
25	EB-crit Empirical distribution of mini-batch gradients for $ \mathcal{B}  = 10$ .	74	
26	EB-crit Empirical distribution of mini-batch gradients for $ \mathcal{B}  = 10$ .	75	
27	EB-crit: Sketch of early-stopping criterion.	79	
28	EB-crit: Linear least-squares toy problem.	84	
29	EB-crit: Synthetic quadratic problem.	85	
30	EB-crit: Illustration of buffer-region induced by the EB-criterion.	86	
31	EB-crit: Synthetic quadratic problem with sub-optimal initialization.	87	
32	EB-crit: Logistic regression on Wisconsin Breast Cancer dataset.	87	
33	EB-crit: Multi-layer perceptron on MNIST trained with GD.	88	
34	EB-crit: Multi-layer perceptron on MNIST trained with SGD.	89	
35	EB-crit: Shallow net and logistic regressor on SECTOR.	90	
36	EB-crit: Greedy element-wise stopping on MNIST.	91	
37	EB-crit: Sketch of RMSPROP-damping relative to an SGD-step.	92	
38	EB-crit: Comparison of non-greedy element-wise stopping to RMSPROP-damping.	95	
39	PROBLS: Comparison of classic interpolator and IWP-mean.	103	

40	PROBLS: Sketch of candidate selection.	105
41	PROBLS: Sketch of acceptance procedure.	105
42	PROBLS: Curated snapshots on MNIST.	106
43	PROBLS: Sketch of PROBLS-algorithm.	107
44	PROBLS: Noise sensitivity: N-II on MNIST.	115
45	PROBLS: Sensitivity to hyper-parameters $c_2$ , and $c_W$ .	117
46	PROBLS: Sensitivity to hyper-parameters $c_2$ , and $\alpha_{\text{ext}}$ .	119
47	PROBLS: Sensitivity to hyper-parameter $\theta_{\text{reset}}$ .	120
48	PROBLS: Different choices of acquisition function.	121
49	PROBLS: Accepted logarithmic learning rate traces.	122
50	PROBLS: Logarithmic noise levels $\sigma_f$ and $\sigma_{f'}$ .	122
51	KFGRAD: In-model toy example..	134
52	KFGRAD: Out-of-model toy example.	136
53	KFGRAD: SGD and KFGRAD on Rosenbrock.	138
54	KFGRAD: SGD, SGD+MOMENTUM, and KFGRAD training an MLP on MNIST.	140
55	KFGRAD: Distribution of Kalman gains.	140
56	KFGRAD: Hyper-parameter traces of KFGRAD_SCALAR.	141
57	KFGRAD: Hyper-parameter traces of KFGRAD_DIAG.	142
58	KFHES: Dennis-class hyper-parameters.	159
59	KFHES: BFGS, GD, and L-KFHES training and MLP on MNIST.	164
60	PROBLS: Noise sensitivity: N-I on MNIST.	207
61	PROBLS: Noise sensitivity: N-II on CIFAR-10.	207
62	PROBLS: Noise sensitivity: N-I on CIFAR-10.	208
63	PROBLS: Noise sensitivity: N-III on GISETTE.	208
64	PROBLS: Noise sensitivity: N-III on WDBC.	209
65	PROBLS: Noise sensitivity: N-III on EPSILON.	209
66	PROBLS: Hyper-parameter sensitivity: $c_2$ - $c_W$ -space, $\alpha_{\text{ext}} = 1.4$ .	210
67	PROBLS: Hyper-parameter sensitivity: $c_2$ - $c_W$ -space, $\alpha_{\text{ext}} = 1.3$ .	210
68	PROBLS: Hyper-parameter sensitivity: $c_2$ - $c_W$ -space, $\alpha_{\text{ext}} = 1.2$ .	211
69	PROBLS: Hyper-parameter sensitivity: $c_2$ - $c_W$ -space, $\alpha_{\text{ext}} = 1.1$ .	211
70	PROBLS: Hyper-parameter sensitivity: $c_2$ - $c_W$ -space, $\alpha_{\text{ext}} = 1.0$ .	212
71	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.01$ .	212
72	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.10$ .	212
73	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.20$ .	213
74	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.30$ .	213
75	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.40$ .	213
76	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.50$ .	214
77	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.60$ .	214
78	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.70$ .	214
79	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.80$ .	215
80	PROBLS: Hyper-parameter sensitivity: $c_2$ - $\alpha_{\text{ext}}$ -space, $c_W = 0.90$ .	215

## List of Algorithms

---

1	Prelim: Sketch of an iterative optimizer.	36
2	Prelim: Sketch of a classic line search.	54
3	Prelim: Sketch of Bayesian optimization.	65
4	Sketch of a probabilistic line search.	101
5	Sketch of diagonal KF <sub>GRAD</sub> .	137
6	Sketch of scalar KF <sub>GRAD</sub> .	138
7	Sketch of noise-free KF <sub>HESS</sub> .	164

## Acronyms

---

### *Gradient Based Optimizers*

GD	Gradient descent
SGD	Stochastic gradient descent
ADAGRAD	The Adagrad optimizer
ADAM	The Adam optimizer
RMSPROP	The RMSprop optimizer
ADADELTA	The Adadelata optimizer
MOMENTUM	Indicates that a momentum term is added, e. g., SGD+MOMENTUM.
NESTEROV	Nesterov accelerated gradients
BFGS	The BFGS optimizer (Broyden-Fletcher-Goldfarb-Shanno)
DFP	The DFP optimizer (Davidon-Fletcher-Powell)
KFGRAD	The KFgrad optimizer (Kalman filter on gradients)
KFHES	The KFhess optimizer (Kalman filter on Hessian elements)

## *Processes, Models and Probability*

pdf	Probability density function
cdf	Cumulative distribution function
GP	Gaussian process
WP	Wiener process
IWP	Integrated Wiener process
BO	Bayesian optimization
CLT	Central Limit Theorem
MLP	Multi-layer perceptron
CNN	Convolutional neural network

## Notation

---

### *Risk, Gradients and Distributions (all Chapters)*

$w$	Inputs to an objective function (Usually $\mathcal{L}$ or $L_{\mathcal{D}}$ ); parameters of a model
$N$	Size of $w$ , i. e., $w \in \mathbb{R}^N$
$d$	Single datapoint from a data distribution
$\ell(w, d)$	Loss, goodness of fit on datapoint $d$ for parameters $w$
$\mathcal{D}$	A finite and fixed dataset (usually the training dataset); elements are i. i. d. samples from the data distribution
$\mathcal{B}$	A mini-batch; $\mathcal{B}$ is a subset of $\mathcal{D}$ with elements sampled i. i. d. from $\mathcal{D}$ with or without replacement
$\mathcal{S}$	Placeholder for arbitrary dataset, can e. g., be $\mathcal{D}, \mathcal{B}$
$ \mathcal{S} ,  \mathcal{D} ,  \mathcal{B} $	Dataset sizes; number of elements in datasets $\mathcal{S}, \mathcal{D}$ , and $\mathcal{B}$
$\mathcal{L}(w), \nabla \mathcal{L}(w)$	Risk and its gradient at $w$
$L_{\mathcal{D}}(w), \nabla L_{\mathcal{D}}(w)$	Empirical risk defined by full dataset $\mathcal{D}$ and its gradient
$L_{\mathcal{B}}(w), \nabla L_{\mathcal{B}}(w)$	A mini-batch loss defined by mini-batch $\mathcal{B}$ and its gradient
$L_{\mathcal{S}}(w), \nabla L_{\mathcal{S}}(w)$	A mean-loss defined by arbitrary dataset $\mathcal{S}$ and its gradient; can be e. g., an empirical risk, or a mini-batch loss

$\Lambda(w)$	Population variance of $\ell(w)$
$\Sigma(w)$	Population covariance of $\nabla\ell(w)$
$\hat{\Lambda}(w)$	An Estimator for $\Lambda(w)$
$\hat{\Sigma}(w)$	An Estimator for $\text{diag}[\Sigma(w)]$

### *Probabilistic Line Searches (Chapter III)*

$\alpha_t$	Learning rate; step size
$p_t$	Search direction (not normalized), defining a one-dimensional search space
$t$	One-dimensional input to the objective along search direction, in units of $\alpha_{\text{ext}}\alpha_t$ ; overloaded with iteration index $t$
$f(t), f'(t)$	Function value and gradient of one-dimensional search space
$y(t), y'(t)$	Noisy evaluations of $f(t)$ and $f'(t)$
$\sigma_y^2, \sigma_{y'}^2$	Variances of $y$ and $y'$
$p^{\text{Wolfe}}$	Wolfe probability
$c_1$	Parameter of Armijo condition
$c_2$	Parameter of sufficient decrease condition
$c_W$	Parameter for Wolfe threshold
$\alpha_{\text{ext}}$	Extrapolation parameter
$\theta$	Scale of Wiener process kernel

### *Filtering Framework (Chapters I, IV)*

$x_t$	State of a probabilistic state space model
$D$	Size of state $x_t \in \mathbb{R}^D$
$F_t$	Drift matrix
$A_t$	Transition matrix
$H_t$	Measurement matrix
$y_t$	Noisy observation of $H_t x_t$
$R_t$	Measurement covariance

$G_t$	Innovation covariance/ Gram-matrix
$g_t$	Kalman gain
$L_t$	Diffusion/ dispersion matrix
$Q_t$	Diffusion covariance
$q$	Intensity matrix
$m_{t+1-}$	Mean estimator of predictive state $x_{t+1}$ given $y_{1,\dots,t}$
$P_{t+1-}$	Covariance of predictive state $x_{t+1}$ given $y_{1,\dots,t}$
$m_t$	Mean estimator of updated state $x_t$ given $y_{1,\dots,t}$
$P_t$	Covariance of updated state $x_t$ given $y_{1,\dots,t}$
$m_t^s$	Mean estimator of smoothed state $x_t$ given $y_{1,\dots,T}$
$P_t^s$	Covariance of smoothed state $x_t$ given $y_{1,\dots,T}$





## Prologue



## Introduction

---

OPTIMIZATION problems, or rather the *solutions* thereof manifest in the very basic laws of nature: Light travels the path of minimal time (Fermat's principle), closed physical systems settle in states of lowest energy (2<sup>nd</sup> law of thermodynamics), or the dynamics of a system arise from minimizing the action functional (Hamilton's principle). Vaguer versions appear in biology, when scarce iron in red blood cells is recycled by an organism, or in the structure of honeycombs (Figure 1). They are artificially created in human-defined objectives, for example a concrete engineering task like the fuel efficiency of an internal combustion engine, or the growth of sales figures and profit. Even in our daily lives, we might seek to minimize the time to get to work by choosing an appropriate travel path.

The concept of optimization is thus often related to a fundamental law, a limited and/or valuable resource, a subjective loss or gain. Optimization itself is a powerful principle and mathematical tool that prevailed over time, and hence has been extensively studied, albeit with varying focus in different fields, motivations and applications. Solvers find the solutions to optimization problems, and are thus of utmost practical relevance.

### *Definition and Task*

Nowadays, automated computers are used to solve optimization problems. While the design of such a solver can be guided by intuition and experimental feedback, further constrained by hardware requirement and finite computational budgets, mathematics provides a formal definition of the problem setting: Formally, optimization is the task of finding the extremal value of a function, also called the *objective*,  $f : \mathbb{R}^N \rightarrow \mathbb{R}$ ,  $w \mapsto f(w)$ ; without loss of generality, hereafter always phrased as *minimization*:

$$\min_w f(w). \tag{1}$$

In practice though, we are often interested in the the point  $w^*$  where the objective attains its extremal value, i.e.,  $w^* = \arg \min_w f(w)$ , rather than the minimal value of  $f$  alone. We will call the point  $w^*$  the *solution*. It determines our course of action, an optimal algorithm, or the specific manifestation of a production line. Formally,  $w^*$  is the *global* minimizer of  $f(w)$ , that is a point  $w^* \in \mathbb{R}^N$  where  $f(w^*) \leq f(w)$  for all  $w \in \mathbb{R}^N$ . There might be multiple points ful-

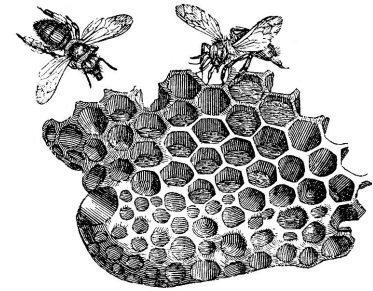


Figure 1: The honeycomb is composed of many hexagons. The shape is beneficial to minimize building cost for the bees. [Image taken from S. G. Goodrich Animal Kingdom Illustrated Vol 2 (New York, NY: Derby & Jackson, 1859), public domain]

filling this requirement, in which case any of them is a solution. For our purposes, we will further assume that  $f$  is bounded from below, and that  $w^*$  is attained in  $\mathbb{R}^N$ . We also assume that  $f$  is at least once-differentiable everywhere, such that the multi-output gradient function  $\nabla f : \mathbb{R}^N \rightarrow \mathbb{R}^N, w \mapsto \nabla f(w)$  exists. Therefore  $w^*$  equals a point of vanishing slope, i. e.,  $\nabla f(w^*) = 0$ .<sup>1</sup>

If the input-dimension  $N$  is large, we often simplify the task further, and assume that any *local* minimizer, that is a point  $w^*$  that fulfills  $f(w^*) \leq f(w)$ , for all  $w \in \Omega$ , with  $\Omega \subset \mathbb{R}^N$  a neighborhood of  $w^*$ , is an acceptable solution as well. The latter statement enables the use of greedy, gradient-based optimizers that do not explore the whole domain of  $f(w)$ , but rather focus on exploiting promising areas towards a local minimum. This is algorithmically and structurally more appealing and easier to handle, but comes at the expense of potentially not finding the best solution possible. Still, local minimizers often perform well in the task they are applied to later, even though they might not be optimal by definition. The goodness of an algorithm that performs optimization tasks, also simply called an *optimizer*, can be evaluated in the light of mathematical convergence results, or alternatively (but not mutually exclusive) by an extended empirical evaluation. The former arguably provides more rigor, but possibly only for a restricted class of functions whose assumptions are hard to check in practice, or are possibly not fulfilled by definition of  $f$ . The latter argues in the light of experimental evidence, which is less rigorous, but sometimes the only accessible tool at hand. In practice, both methods often go hand in hand and provide a more complete picture about an optimization algorithm. In this thesis we will focus on experimental evidence, not as a matter of principal, but rather for reasons of personal taste, and since the aim of this dissertation is to design novel optimizers of immediate practical and applied relevance. The immediate feedback from toy-examples with a controlled environment as well as real-world applications is valuable in that respect.

### *Optimization in Machine Learning & General Outline*

Optimization problems occur in many different disciplines. This dissertation specifically deals with optimization tasks arising from machine learning applications, where computers learn a task without being explicitly programmed for it.<sup>2</sup> Since computers are deterministic calculators, this means that these programs need to select or continuously change a model that subsequently identifies their current strategy and hence the algorithm, rather than directly encoding it. This is done according to a finite amount of *data* that they see sequentially, or alternatively at one point in time. If this selection process is non-trivial,

<sup>1</sup> Other objectives that will not be discussed here might include functions with discrete input and/or output, non-differentiable functions, or constrained solution domains.

<sup>2</sup> This definition is commonly attributed to Arthur Lee Samuel.

it is typically performed by *optimizers* which find a good model under some measure among a parametrized class of potentially infinitely many models. This might be the hyperparameters of a hierarchical probabilistic model under the model's evidence, the indicators and group parameters of a clustering model under some average distance relation, or the weights of a neural network under an empirical risk. A particular instance of this is high-dimensional, stochastic empirical risk minimization, which currently is a major workhorse for solving supervised regression and classification tasks, and has been attracting attention especially in combination with artificial neural network models. This dissertation develops novel concepts for optimization and the latter will provide the main application for testing. All relevant concepts briefly mentioned here will be introduced in greater detail in later chapters.

Intriguingly though, even numerical methods like optimizers, which are applied to a variety of different tasks are not at all static methods and, they, too, need to adapt and change according to the 'data' they collect from the CPU, usually in the form of possibly stochastic function evaluations and gradients, albeit in a much more lightweight, efficient, and hence often approximate way. Thus, a part of this thesis will be concerned with elaborating on the connection to probabilistic inference, and the interpretation of optimizers as being learning machines themselves that exhibit additional requirements on tractability, constraints on hardware, memory and computational cost. This will be done by tackling three sub-challenges, present in contemporary empirical risk minimization problems. These are: i) Improvement of generalization performance, ii) Gains in automation by removing tuning parameters, and, iii) Design of search directions that are robust to stochastic gradient evaluations.

The next section provides a brief, targeted historic overview, followed by a section that summarized the main tasks and contributions of this thesis in greater detail.

### *A Short Historic Introduction*

The history of optimization is vast and old. The following paragraphs thus mention some selected historic steps in the development of *defining*, *solving* and *analyzing* non-trivial optimization problems, with the aim to position this thesis in today's knowledge and challenges. Towards the end, contextual emphasis will be placed on topics relevant to this dissertation; as mentioned above, these are stochastic high-dimensional optimization problems, in particularly empirical risk minimization.

- **FIRST GEOMETRIC OPTIMIZATION PROBLEMS:** The first optimization problems that were analyzed in a structured way, were probably of

geometric nature. Especially the works of Euclid of Alexandria<sup>3</sup>, *The Elements—Books I–XIII*, ~ 300 B.C. [38], is an extensive collection and study of geometric and trigonometric shapes and their relations, some of which are the solutions to questions of optimality: The minimal distance of a point to a plane, the square as the rectangle of largest area for a given circumference, or the sphere as the 3D-object with largest volume for a given surface area (Figure 2). Most of these problems have an analytic representation of  $f$ , as well as an analytic solution  $w^*$ . Also the honeycomb-conjecture mentioned above and illustrated in Figure 1, proven to be correct just recently in 1999, 2001 by T. C. Hales, was already phrased back then, as the division of a 2D-surface into equal areas that minimizes the perimeter (optimal packing).<sup>4</sup>

- **FROM ANALYTIC SOLUTIONS TO ITERATIVE TECHNIQUES:** It became apparent soon that some equations would not have analytic solutions, such as finding the eccentric anomaly  $E$  of a star's orbit in Kepler's equation  $M = E - \epsilon \sin(E)$ , given the eccentricity  $\epsilon$  of the ellipse and mean anomaly  $M$  (Figure 3). Issac Newton famously solved it by first starting at a good guess close to  $E^*$ , and then adding terms to it, found by an algebraic expansion, with the goal to approach the true solution ever closer. With the advent of calculus, and thus *gradients* (fluxions), by Newton and Leibniz (~ 1670), and later Euler and Lagrange (calculus of variations, ~ 1750), similar solvers appeared, such as steepest descent by Cauchy [21] that lowered a function's value by walking cautiously but ad infinitum into a descent direction. The first *iterative* solvers were born, breaking down a difficult problem into multiple, successive easier ones.
- **COMPUTERS AND NEW MATHEMATICAL CONCEPTS:** In the meantime, new mathematical concepts appeared (~ 1850 – 1950), and existing ones were formulated more rigorously, such as continuity (Weierstraß), convexity (Jensen), or Lipschitz continuity of functions, which made it possible to analyze iterative solver in the light of convergence and convergence rates. The questions “Can the true minimizer be returned if we just iterate long enough, and, if yes, how fast will we get there?” that had only been hypothesized before, e. g., by Cauchy, became more prominent and of practical concern since it was now possible to consider more than a handful of iterates. Automated computers finally leveraged the full potential of iterative techniques that showed very good behavior just beyond the first few iterations, notably quasi-Newton methods (~ 1950), or iterative solvers for constrained linear problems (linear programming). Computational complexity and memory requirement of algorithms became primary issues, due to the increasing demand for also solving multi-dimensional optimization problems.

[21] Cauchy, “Méthode générale pour la résolution des systèmes d'équations simultanées,” 1847

[38] Euclid and Fitzpatrick, *Euclid's Elements*, 2009

<sup>3</sup> He also invented the phrase ‘*Quod erat demonstrandum*’, still used to date to end mathematical proofs.

<sup>4</sup> Pappus of Alexandria provided a proof for divisions by the polygons triangle, square and hexagon in *Collection Book V* (~ 300 B.C.).

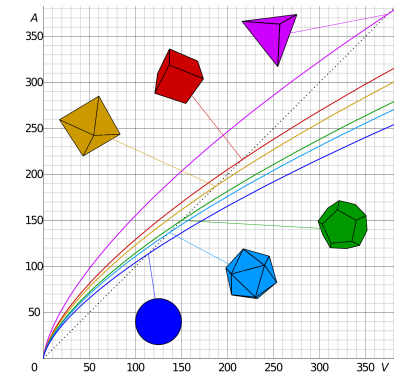


Figure 2: For a given surface area  $A$ , the sphere has the largest volume  $V$  of all 3D-objects.

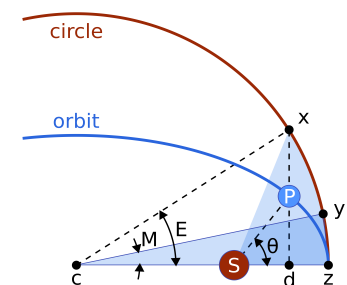


Figure 3: Orbit of a planet (P) around the sun (S). Solving Kepler's equation (Johannes Kepler, 1609 in *Astronomia Nova* Chapter 60) corresponds to minimizing the 1D-function

$$f(E) = 0.5E^2 - ME + \epsilon \cos(E),$$

where  $\epsilon \in (0, 1)$  is the eccentricity of the elliptical orbit.

Source Fig. 2: Wikipedia CC BY-SA 3.0, 'Tetrahedron.svg', by Cmglee <https://commons.wikimedia.org/w/index.php?curid=26667051>

Source Fig. 3: Wikipedia CC BY-SA 4.0, 'Mean Anomaly.svg', by CheCheDaWaff <https://commons.wikimedia.org/w/index.php?curid=48381371>

- **FASTER COMPUTERS, GPUS AND LARGE DATASETS:** Towards the end of the 20<sup>th</sup> century, the possibility to collect and store large amounts of data became ubiquitous. Processing it, though, still remained slow, but was considerably sped up by graphics processing units (GPUs) which enabled parallel processing of a limited amount of datapoints, as long as algebraic operations could be expressed as independent arithmetic operations, e.g., matrix-vector products. In an attempt to lower the cost further, *stochastic* optimizers, such as stochastic gradient descent (SGD), were invented, groundbreakingly proven to converge under certain conditions by Robbins and Monro [113]. These methods only process a small fraction of data at a time, sub-sampled from a much larger, finite dataset, or an online stream of data that is infinite theoretically, but finite at every point in time. This arguably led to a shift in the optimization community, towards solvers needing an excessive amount of, but therefore very cheap iterations. Especially stochastic empirical risk minimization became a major workhorse, where unbiased gradient estimators are easily accessible by sub-sampling a large dataset. At the same time, this also opened up new questions, as how to trade-off potential progress per step, and the cost involved in doing so, which were not as present before and still are mostly unsolved to date.
- **TODAY'S CHALLENGES AND NEW QUESTIONS:** The success of certain practical machine learning applications foremost in industry (the second advent of AI), revealed that optimization is still often a bottleneck in a larger chain of learning procedures. Thus, the need for better (faster, data-efficient) optimizers is again at a high today and already led to a pool of variants of SGD that are easier to use and more cost efficient.

A notable difference, though, in today's challenges, is that optimizers increasingly are not only concerned about minimizing the objective function, but become intertwined with the whole learning problem itself. In empirical risk minimization, for instance, where the objective  $f(w)$  of Eq. 1 is the empirical risk, 'good' optimizers nowadays, are expected to return solutions that also generalize well to unseen data, reduce overfitting, or react robustly to the noise, that originates from sub-sampling the dataset. These solutions are not necessarily equal to the solution of  $\arg \min_w f(w)$ , and are subject to questions of information theory and statistics. This connection is intriguing, but comes with the drawback that the desired objective can often not be written down analytically anymore, or is subject to disputable approximations and assumptions. Consequently, it is up to controversy what the optimizer's objective should be after

[113] Robbins and Monro, "A stochastic approximation method," 1951

all, if not  $f(w)$ ; and, throwing Eq. 1 out of the window, the notion of ‘what a good optimizer is’, is even debated today.

### Detailed Outline

Optimization is and has always been a combined effort of disciplines such as math, physics, and later computer science, driven by applications and practical problems to solve. Standing on the shoulders of giants, in this thesis we will attempt to tackle some of the current open questions of the optimization community, and contribute to the knowledge and progress of scientific research. In particular, we focus on stochastic, high-dimensional optimization, with an emphasis on empirical risk minimization for supervised classification and regression tasks. As outlined above, current open questions include:

- Improving generalization performance that can not directly be measured by  $f(w)$ .
- Gaining automation by removing tuning parameters.
- Designing search directions that are robust to stochastic gradient evaluations.

Partially based on the publications listed below, this thesis contains the following parts that are summarized here briefly. Several concepts that are mentioned here without further note, will be introduced in more detail in later chapters.

- **EARLY-STOPPING WITHOUT A VALIDATION SET:** Consider empirical risk minimization, where  $f(w)$  is the empirical risk, defined by a finite dataset  $\mathcal{D}$  that contains pairs of an unknown input-target relation. A very expressive model  $\mathcal{M}_w$ , parametrized by  $w$ , that learns this relation, might overfit to the dataset and perform poorly on unseen data from the same (true) input-target relation. This is unsurprising, since the encoding of  $\mathcal{M}_w$ , and thus the objective  $f(w)$ , has knowingly ignored all other possibly occurring input-target-pairs not contained in  $\mathcal{D}$ .

Besides regularization techniques that change the model  $\mathcal{M}_w$  and thus  $f(w)$ , and some others, arguably the widely accepted paradigm to prevent overfitting is *early-stopping*; a procedure that simply halts the optimizer before it reaches the minimizer  $w^*$  of the empirical risk. The rationale behind it is that the sequence of models  $\mathcal{M}_{w_t}$ ,  $t = 0, \dots, T$  that the optimizer proposes at iterations  $t$ , generally lead from underfitting to overfitting, and generalize well somewhere along the path. A good stopping point is then identified by monitoring the loss of a validation set; that is, the dataset  $\mathcal{D}$  is partitioned into two or three *smaller* disjoint datasets: One for training/defining  $f(w)$ , a validation set for monitoring generalization



that also induces stopping, and possibly another for a final evaluation, since the stopping decision is biased towards the validation set, too. This is a very reliable technique when the dataset  $\mathcal{D}$  is large enough, such that splitting it up does not deprive each partition, but especially the optimizer's objective, of valuable information about the input-target relation it has to learn; but for smaller, medium sized or highly diverse datasets, splitting can notably affect the generalization performance of  $\mathcal{M}_w$  that can possibly be reached.

In Chapter 6, we will thus formulate a weaker, novel early-stopping criterion which allows to *fold-in* the validation set into the training procedure. The optimizer is halted when gradients can be fully explained by sample-noise arising from the finite size of the dataset, and not by an informative gradient direction, even if the full dataset is used per iteration. We show on several test cases that the additional data accessible to the training procedure helps to improve generalization on a withheld test-set, and also empirically induces well calibrated stopping decisions, especially on small to mid-sized datasets, where the stopping decision induced by a validation set can be corrupted or heavily biased.

- **GAUSSIAN LIKELIHOODS FOR LOSSES AND GRADIENTS:** We will make a general point on the form of the distribution of local (for a given  $w$ ) losses and gradients of an empirical risk function. We will argue that, by the central limit theorem, the former can be approximated by Gaussians whose parameters can be *estimated* at run-time with little computational overhead. We will also support this claim with heuristic evidence. This yields an analytic likelihood of the unknown true loss and gradient at every input-location  $w$ , which subsequently can be incorporated into an inference scheme. This is in line with a re-emerging field called *probabilistic numerics* [58], based on previous works by Diaconis [31] and O'Hagan [101] (for univariate integration, which led to Bayesian quadrature), Skilling [126] (for ODEs), and more philosophically Poincaré [103]. Specific recent works on optimization include Hennig and Kiefel [57] on probabilistic quasi-Newton methods, as well as Hennig [56] for works on the solution of linear systems which is closely related to optimization of quadratic problems. In all these, 'data', such as the evaluations of mini-batch gradients, is of computational nature, mined by a brain or a CPU at the cost of time or energy consumption; and the objects of inferential interest are solutions to non-trivial numerical, non-physical problems.

For optimization in particular, this means that having incorporated more knowledge about the distribution of possibly occurring, but yet unseen gradients, the optimizer has an increased potential to learn and adapt its internal model parameters such as smoothing

[126] Skilling, "Bayesian solution of ordinary differential equations," 1991

[103] Poincaré, *Calcul des probabilités*, 1896

[57] Hennig and Kiefel, "Quasi-Newton methods – a new direction," 2012

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

[58] Hennig, Osborne, and Girolami, "Probabilistic numerics and uncertainty in computations," 2015

[31] Diaconis, "Bayesian numerical analysis," 1988

[101] O'Hagan, "Some Bayesian Numerical Analysis," 1992

constants, learning rates et cetera, that would need to be hand-crafted otherwise. In other words, the capability of the optimizer to encode crucial information *about* the objects it collects from a CPU, increases its ability to foresee, weigh, and react to yet unseen evaluations thereof. We will see that, adopting an empirical Bayes approach, the specific parameters of these distributions, in particular the variances, can often be estimated with little overhead that justifies the overall performance gains. The possible results are optimizers with a higher degree of automation as well as robustness; two examples thereof will be discussed next.

- **AUTOMATED STEP SIZE ADAPTATION FOR STOCHASTIC GRADIENT DESCENT:** We will choose to look at optimizers as models, themselves parametrized by tunable quantities like smoothing constants, decay factors, learning rates or parameters of learning rate schedules. Some are more prominent than others, but they all identify a particular algorithm uniquely. In general, parameters can either be set to roughly insensitive design choices, or, if that is not possible, they need to be hand-tuned by an expert, according to a performance measure, such as generalization loss after a finite budget of CPU- or wall-clock-time. Given a function handle to  $f(w)$ , the single parameter of stochastic gradient descent (SGD) is the step size, or *learning rate*  $\alpha$ , which is often tedious to tune or to hand-craft and, even then, is virtually never optimal per step.

In Chapter 7, we will introduce a novel subroutine, called a *probabilistic line search*, of constant, small cost that adapts the learning rate of SGD at each step by checking local probabilities of descent. The algorithm is based on the idea of a line search subroutine (e. g., [100, § 3]), classically designed for deterministic gradient and loss evaluations. We take design parts of the algorithm one-by-one and translate them to their probabilistic equivalent where possible, or generalize them where necessary. We show on several classification tasks that the novel line-search-adapted SGD exhibits superior or same generalization performance in comparison to SGD with carefully hand-tuned learning rates; but with the advantage of not having to do *any* explorative experiments or expert-user interaction. It shows that previous tuning-parameters can be removed entirely, by explicitly encoding and estimating not only loss and gradient estimators, but also the structure of their distribution.

- **A FILTERING FRAMEWORK FOR STOCHASTIC OPTIMIZATION:** If one compares the computer-codes of stochastic gradient descent (SGD) and full-batch gradient descent (GD), given a function handle to  $f$ , they are indistinguishable. They also share the same set of tunable parameters: scalar learning rates  $\alpha_t$ . This means that SGD in fact does not understand that it is exposed to corrupted gradients; all we

[100] Nocedal and Wright, *Numerical Optimization*, 1999

know, as users, is that it does not matter to some extent because we were able to prove, that it converges anyway (with constraints on its own model parameters  $\alpha_t$  [113]). Thus, the question arises if also the search direction, and not just a small subroutine like line searches, might benefit from an explicit encoding of gradient distributions. After all, it has been shown before [57] [56] that some update rules of *deterministic* optimizers are special cases of Gaussian inference on Hessians, involving Gaussian likelihood functions that collapse on delta-peaks for exact gradient evaluations, i. e., vanishing variances. Now that we know that Gaussian likelihoods are quite accurate assumption even, what would happen for finite variances? Would inference still be tractable? How would we use the posterior distributions in a concrete iterative scheme? And would an optimizer even benefit from the potentially more involved iterations?

There can be many more questions asked, though Chapters 8 and 9 can be understood as a cautious approach (similar to [57] who did not solve tractability) to motivate and derive tractable updates for noise-informed first- and second-order search directions for stochastic optimization. We will draw connections to current popular methods and show direct support of exponential smoothing of gradients in first-order methods, as done rather ad-hoc in rules such as momentum-SGD [105] [116] or the denominator of the ADAM-update [74]. We will also see that objects, such as sample variances of stochastic gradients, appear implicitly and much less structured in many popular updates (see also the works of Balles and Hennig [5]) which supports the use of possibly better, since unbiased, *local* variance estimates. Based on these findings, we propose KF-GRAD, a momentum-like update, derived from a Gauss-Markov model for the true gradient function defined on the optimization path, which can learn its own natural smoothing factors (also called ‘gains’ in filtering/signal processing), and is richer in structure than momentum-SGD.

In a similar manner, we analyze a novel class of probabilistic quasi-Newton methods, arising from a tractable estimator based on a Gauss-Markov model for Hessian elements. We draw connections to classic quasi-Newton methods (like the members of the Dennis family [29], including its famous member BFGS [19] [40] [44] [123]) for the limit of deterministic gradient evaluations. We are then able to analyze the implicit model assumptions made by these classic methods and argue that similar strategies or choices of hyperparameters, in stochastic settings, might in fact be less preferable, and should not blindly be transferred. We also show first steps towards a scalable probabilistic quasi-Newton optimizer that might be worth investigating in the future.

[113] Robbins and Monro, “A stochastic approximation method,” 1951

[57] Hennig and Kiefel, “Quasi-Newton methods – a new direction,” 2012

[56] Hennig, “Probabilistic Interpretation of Linear Solvers,” 2015

[105] Polyak, “Some methods of speeding up the convergence of iteration methods,” 1964

[116] Rumelhart, Hinton, and Williams, “Learning representations by back-propagating errors,” 1986

[74] Kingma and Ba, “Adam: A Method for Stochastic Optimization,” 2014

[5] Balles and Hennig, “Follow the Signs for Robust Stochastic Optimization,” 2017

[123] Shanno, “Conditioning of quasi-Newton methods for function minimization,” 1970

[29] Dennis, “On some methods based on Broyden’s secant approximations,” 1971

[19] Broyden, “A new double-rank minimization algorithm,” 1969

[40] Fletcher, “A new approach to variable metric algorithms,” 1970

[44] Goldfarb, “A family of variable metric updates derived by variational means,” 1970

We hope that the above points, and the corresponding detailed texts below, contribute to progress in optimization research, and that they will be found useful by the community. Optimization, as a problem-setting or task, is located at an intriguing mid-ground somewhere in between a clear inference task with a clear goodness-measure, such as approximating the Newton direction, and the task of the user that is evaluated in *another* measure, such as the performance of returned  $\mathcal{M}_w$  after a finite computational budget. This trade-off is still poorly understood and hard to analyze. We hope that this work contributes towards disentangling both sides, such that interpretability, user-friendliness, and automation of optimization machines can be improved further. Towards that end, we present first practical algorithms in three sub-tasks: Search directions, learning rates, and generalization; all based on probabilistic models that use and rely on statistics of stochastic gradients and losses, and either analyze model assumptions, ease the usage, or improve the performance of stochastic optimizers today.

### Content

PART I introduces concepts and notation that will be used throughout the text, as well as relevant related literature. These are: **Chapter 1** *General concepts of reasoning and probabilistic inference*: Basic probability calculus, Gaussian distributions, Gaussian processes, probabilistic state space models, Wiener processes, and Kalman filters. **Chapter 2** *Optimization and applications*: Empirical risk minimization, artificial neural networks, gradient-based deterministic and stochastic optimizers, regularization, early-stopping, and deterministic line searches. **Chapter 3** *Closely related literature*: Quadratic problems, Kronecker algebra, and the probabilistic linear solver of Hennig [56]. **Chapter 4** *Miscellanea*: Bayesian optimization, and the central limit theorem.

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

PARTS II, III & IV contain the main contributions of this dissertation. These are: **Chapter 5** Explicit description of conditional distributions of stochastic gradients and losses. **Chapter 6** A novel early-stopping criterion for better generalization. **Chapter 7** A probabilistic line search sub-routine for fully automated learning rate tuning of SGD. **Chapter 8** A novel filtering framework for stochastic gradients and first-order optimization. **Chapter 9** A novel filtering framework for stochastic Hessians and second-order optimization.

The Appendix contains additional results, derivations, plots, and pseudo-codes of Chapters 5-9.

### Publications

Parts of the work in this thesis were done in collaboration with colleagues and are based on the following publications:

Chapter 7 is based on the following peer-reviewed journal publication:

M. Mahsereci and P. Hennig. “Probabilistic Line Searches for Stochastic Optimization.” In: *Journal of Machine Learning Research* 18.119 (2017), pp. 1–59. URL: <http://jmlr.org/papers/v18/17-049.html>

A peer-reviewed conference version, selected for a full oral presentation was published in :

M. Mahsereci and P. Hennig. “Probabilistic Line Searches for Stochastic Optimization.” In: *Advances in Neural Information Processing Systems (NIPS)*. vol. 28. 2015, pp. 181–189

Chapter 6 is based on the on the following pre-print (in preparation for submission to JMLR):

M. Mahsereci, L. Balles, C. Lassner, and P. Hennig. *Early Stopping without a Validation Set*. 2017. eprint: [arXiv:1703.09580](https://arxiv.org/abs/1703.09580)

Chapters 8 and 9 are unpublished at the time of writing and in preparation for submission to JMLR.

I also contributed major parts to the following peer-reviewed work:

L. Balles, M. Mahsereci, and P. Hennig. “Automating Stochastic Optimization with Gradient Variance Estimates.” In: *ICML AutoML Workshop* (2017)

Additionally, the Preliminaries (Part I), the Appendix, as well as Chapter 5 are partially taken from the publications mentioned above, or are collections thereof, as indicated in the running text.



Part I

Preliminaries





## Gaussian Process Regression

**U**NCERTAINTY is fundamentally inherent to smart decision making. In other words, if there is no uncertainty, there is always a clear answer. A mathematical notion of uncertainty is probability theory which can be derived e. g., from the Cox axioms [25]; these are basic notions of reasoning in agreement with ‘comparability’, ‘common sense’, and ‘consistency’.<sup>1</sup>

### 1.1 Probability Calculus

The symbol  $p(A)$  denotes the probability that proposition  $A$  is true, and  $p(A \cap B)$  the probability that both  $A$  and  $B$  are true. Given that  $B$  is true,  $p(A|B)$  is the *conditional* probability that  $A$  is also true. The two basic rules of probability calculus are the *product rule* and the *sum rule*:

$$p(A \cap B) = p(A|B)p(B) \quad \text{product rule} \quad (2a)$$

$$p(A) = \sum_B p(A \cap B) = \sum_B p(A|B)p(B) \quad \text{sum rule} \quad (2b)$$

The sum rule computes  $p(A)$  by adding up all conditional probabilities  $p(A|B)$  weighted with the probability that  $B$  is true, for all possible propositions  $B$ . All probabilities are normalized between zero and one, i. e.,  $p(A) \in [0, 1]$ .

#### Bayes’ Theorem and Probabilistic Reasoning

A direct consequence of the product rule of Eq. 2 is *Bayes’ theorem* [9] [80]

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}, \quad \text{for } p(B) \neq 0. \quad (3)$$

Bayes’ theorem states that given the conditional probability  $p(B|A)$  and the probabilities  $p(A)$  and  $p(B)$ , it is possible to compute the *inverse* conditional probability  $p(A|B)$ .

Instead of general propositions  $A$  and  $B$ , we are particularly interested in the probability that a variable  $X$  is equal to a value  $x$  from a discrete non-empty set  $\mathcal{X}$ . We then say that  $p(X = x)$  is the probability that  $X$  takes the value  $x$ , and likewise for another variable  $Y$ , with values  $y$  from  $\mathcal{Y}$ . Then,  $p(X = x, Y = y)$  is called a *joint* probability, and e. g.,  $p(X = x)$  a *marginal* probability. Additionally, it is common

[75] Kolmogorov, “Grundbegriffe der Wahrscheinlichkeitsrechnung,” 1933

[25] Cox, “Probability, frequency and reasonable expectation,” 1946

<sup>1</sup>Another commonly used axiomatic system is due to Kolmogorov [75]. Both systems yield the same probability calculus. The difference is merely and mostly in the interpretation of the word ‘probability’, as a statement of plausibility of a hypothesis (Cox), or a frequency of an event from an experiment with random outcomes (Kolmogorov). In this thesis we will also assign probabilities to values of non or not obviously random quantities, without making a rigorous point that this is the only way probabilities should be looked at. We will also use wording and notation of either approach depending on the task and the related literature.

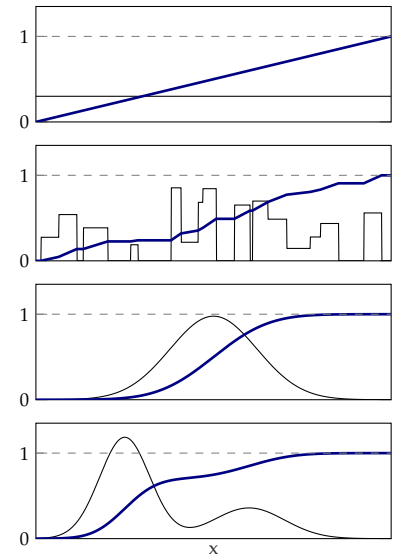


Figure 4: Probability density (—) and cumulative distribution function (—). Top: uniform. Second: steps. Third: single Gaussian. Bottom: mixture of two Gaussians. The cdfs are computed over the whole real line for pdfs that are non-zero outside of the shown interval.

[9] Bayes, “On a problem in the doctrine of chances,” 1763

[80] Laplace, “Mémoire sur la probabilité des causes par les évènements,” 1774

to use the short-hand notation  $p(x)$  instead of  $p(X = x)$  et cetera, and to use  $x$  as the symbol for both, the variable  $X$  as well as the value  $x$ . In applications, realizations of  $y$  are often associated with *observations* or *data*, and  $x$  with an *unknown quantity*. The probability  $p(y|x)$  is then called the *likelihood*,  $p(x)$  the *prior*,  $p(y)$  the *evidence*, and  $p(x|y)$  the *posterior*, all of the unknown  $x$ . This means that given a joint or *generative*<sup>2</sup> model of observed and unobserved variables  $y$  and  $x$ , we can reason about the unobserved ones, given some examples of  $y$ , even if they do not identify  $x$  completely.

### Continuous Variables and Densities

If the variables  $X$  and  $Y$  are not discrete but real-valued, then  $p$  might also denote a probability *density* function (pdf) and the sum in Eq. 2b turns into an integral. Another relevant function is the *cumulative distribution function* (cdf):

$$\text{cdf}_X(x) := \int_{-\infty}^x p(a) da \quad (4)$$

which is the probability  $p(X \leq x)$  that  $X$  has a value smaller than  $x$ . Figure 4 shows examples of four different pdfs (uniform, steps, Gaussian, mixture of two Gaussians) and their corresponding cdf. Some cdfs can be computed analytically, e.g., the cdfs shown in Figure 4, others involve solving the integral in Eq. 4 numerically. The probability  $p(x_1 < X \leq x_2)$  that  $X$  lies in the half-open interval  $(x_1, x_2]$  can be obtained from twice evaluating the cdf:  $\text{cdf}_X(x_2) - \text{cdf}_X(x_1)$ . Figure 5 visualizes ‘marginalization’ and ‘conditioning’ of a multi-modal probability density. Pictorially speaking, conditioning defines a normalized *slice* through the the joint pdf, while marginalization over a single variable defines a *weighted average* of all of those possible slices.

## 1.2 Gaussian Distributions

The density of a one-dimensional Gaussian or *normal* distribution is of the form:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right], \quad (5)$$

also denoted as calligraphic  $\mathcal{N}(x; \mu, \sigma^2)$ . It has two parameters  $\mu \in \mathbb{R}$  and  $\sigma^2 \in \mathbb{R}_+$  which identify the distribution.<sup>3</sup> If  $p$  is unknown but known to be Gaussian, it is thus enough to collect statistics about  $\mu$  and  $\sigma^2$  from independent and identically distributed (i. i. d.) samples. These statistics are called *sufficient statistics*, meaning that, roughly speaking, it suffices to keep around these two numbers instead of the collected samples. The parameters  $\mu$  and  $\sigma^2$  have a geometric

<sup>2</sup> The term ‘generative’ is used non-uniquely in the literature. We will use it to denote a procedure, that can generate  $(x, y)$ -pairs from their joint probability distribution.

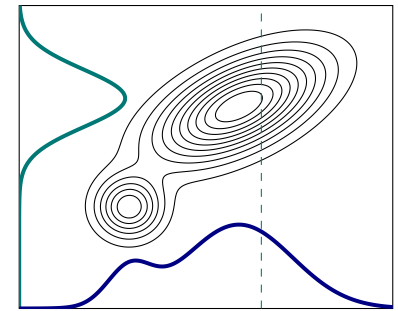


Figure 5: Marginalization and conditioning. Contours of two-dimensional probability density (—), marginal density of abscissa (—), conditional density of ordinate (—), location of conditioning on abscissa (---).

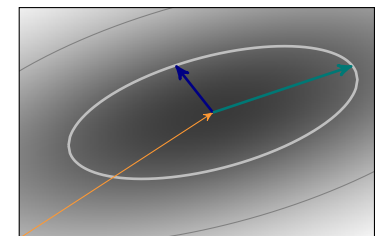


Figure 6: Visualization of mean and covariance. 2D-Gauss with mean vector  $\mu$  (—), eigenvectors of  $\Sigma$  scaled with the square root of their eigenvalues (—/—), contour of  $p(x) = (2\pi)^{-1}(|\Sigma|e)^{-\frac{1}{2}}$  (—).

<sup>3</sup> The parameters can also be represented in other bases: A common one is  $\frac{\mu}{\sigma^2}$  and  $-\frac{1}{2\sigma^2}$  which are also called *natural parameters*.

interpretation:  $\mu$  is the *mean* (first moment, or expected value) of  $x$ , and  $\sigma^2$  is the *variance* (second central moment). For Gaussians the mean is also simultaneously the location of the maximum of  $p(x)$ .

The Gaussian distribution can be generalized to multi dimensions:  $x \sim \mathcal{N}(\mu, \Sigma)$ , where  $x \in \mathbb{R}^D$  is now a vector,  $\mu \in \mathbb{R}^D$  is the mean *vector*, and  $\Sigma \in \mathbb{R}^{D \times D}$  is the positive semi-definite covariance *matrix*. If  $\Sigma$  is positive definite (thus non-singular), the pdf can be written as:

$$p(x) = (2\pi)^{-\frac{D}{2}} |\Sigma|^{-\frac{1}{2}} \exp \left[ -\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu) \right], \quad (6)$$

where  $|\Sigma| > 0$  is the determinant of  $\Sigma$  (illustration of 2D-Gauss in Figure 6).

### Gaussian Inference

Marginalization and conditioning—the elementary operations of probabilistic inference—are *analytic* for Gaussians. This means that given the joint pdf

$$\mathcal{N} \left( \begin{bmatrix} x \\ y \end{bmatrix}; \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix} \right), \quad (7)$$

the marginals are  $\mathcal{N}(x; \mu_x, \Sigma_{xx})$  and  $\mathcal{N}(y; \mu_y, \Sigma_{yy})$ , and the conditional for observed  $y$  is  $\mathcal{N}(x|y; \mu_{x|y}, \Sigma_{x|y})$  with:

$$\mu_{x|y} = \mu_x + \Sigma_{xy} \Sigma_{yy}^{-1} (y - \mu_y), \quad \Sigma_{x|y} = \Sigma_{xx} - \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{yx}. \quad (8)$$

$\Sigma_{x|y}$  is called the *Schur complement* of  $\Sigma_{yy}$ . Pictorially it shrinks the covariance  $\Sigma_{xx}$  by subtracting parts that contain information about  $x$ , observed through  $y$ . In other words if the vector  $[v_x, v_y]^\top$  is drawn according to Eq. 7, then  $v_y$  and the vector  $v_{x|y} := v_x - \Sigma_{xy} \Sigma_{yy}^{-1} v_y$  are independent.

If additionally  $y$  is a noise corrupted linear map of  $x$ , i. e.,  $y = Hx + \epsilon$ , with  $\epsilon \sim \mathcal{N}(0, R)$  ( $H$  and  $R$  are matrices of appropriate size and properties), then Eq. 8 turns into:

$$\begin{aligned} \mu_{x|y} &= \mu_x + \Sigma_{xx} H^\top (H \Sigma_{xx} H^\top + R)^{-1} (y - H \mu_x) \\ \Sigma_{x|y} &= \Sigma_{xx} - \Sigma_{xx} H^\top (H \Sigma_{xx} H^\top + R)^{-1} H \Sigma_{xx}. \end{aligned} \quad (9)$$

Figure 7 shows inference on a 2D-Gaussian variable for differing noisy observations i) top: both dimensions are observed ( $H = I$ ), ii) middle: only one dimension (abscissa) is observed ( $H = [1, 0]$ ), iii) bottom: a one-dimensional arbitrary subspace is observed ( $H = [1, 0] \Theta$ , where  $\Theta$  is a 2D-rotation). For observed parts, the posterior variances are always smaller than both the prior and the likelihood variances; for unobserved parts they are identical to the prior variance. This can be seen from the middle plot where the variance of the unobserved ordi-

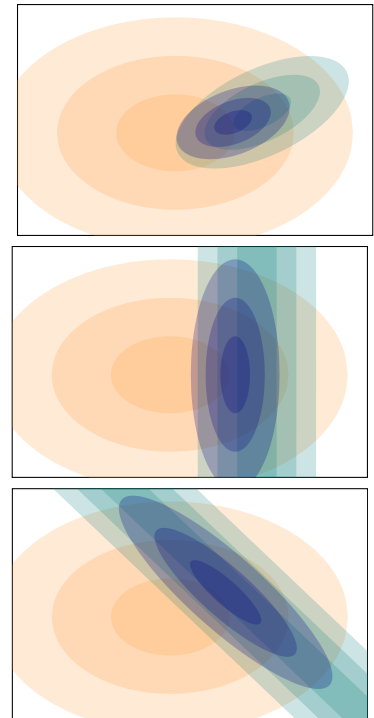


Figure 7: Two-dimensional Gaussian inference . Probability densities shaded for 1, 2, and 3 standard deviations. Prior (orange), likelihood (blue) and posterior (purple). Top: noisy observation. Middle: noisy observation of axis-aligned 1D-subspace. Bottom: noisy observation of arbitrary 1D-subspace.

nate component does not change. The posterior mean  $\mu_{x|y}$  of Eq. 9 is always an elementwise linear combination of the projected prior mean  $H\mu_x$  and the observation  $y$  weighted by  $g := \Sigma_{xx}H^\top(H\Sigma_{xx}H^\top + R)^{-1}$ ; in other words  $\mu_{x|y} = (I - gH)\mu_x + gy$  and  $\Sigma_{x|y} = (I - gH)\Sigma_{xx}$ .<sup>4</sup>

Exact inference in fully Gaussian systems thus only involves operations of linear algebra such as matrix-matrix products, matrix-vector products or matrix-inverses. Gaussian fits to arbitrary probability distributions can thus be seen as a tool to approximate probabilistic inference to first order (linear) computations. This insight is tremendously helpful for the design of probabilistic numerical methods, as considered in this thesis, since numerical linear algebra are operations that a computer can perform to high precision in finite time.

### 1.3 Continuous Indexing—Gaussian Processes

The concept of a multi-dimensional Gaussian distribution can be generalized to continuous index spaces  $\mathcal{Z}$ , e. g., the real line. Therefore, an infinite dimensional object like a *function* instead of a finite dimensional random variable can be modeled with it. A *Gaussian process* (GP) is a collection of random variables  $\{x_z\}_{z \in \mathcal{Z}}$  indexed by  $z$ . The random variables  $\{x_{z_i}\}_{i=1, \dots, T}$  associated with any finite subset  $\{z_i\}_{i=1, \dots, T} \subset \mathcal{Z}$  of these indices are jointly Gaussian distributed [109], [73].<sup>5</sup> Similar to finite-dimensional Gaussian distributions, the Gaussian process is fully specified by a *mean function*  $\mu(z)$ ,  $\mu : \mathcal{Z} \rightarrow \mathbb{R}$  and a *covariance* or *kernel function*  $k(z, z')$ ,  $k : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathbb{R}$ . For the distribution we also write calligraphic  $x \sim \mathcal{GP}(\mu, k)$ .

Any  $T$ -dimensional finite subset of the GP can be computed for an index set  $z_{1, \dots, T}$  by evaluating  $\mu(z_i)$  and  $k(z_i, z_j)$  for all  $i, j = 1, \dots, T$ . This yields a  $T$ -dimensional mean *vector* and a  $T \times T$ -dimensional positive definite covariance, or kernel-Gram *matrix*.

Figure 8 illustrates this idea: Row 1 shows two covariance matrices (5- and 9-dimensional). Rows 2 and 3 show illustrations of the corresponding 5- and 9-dimensional Gaussians: Samples (---) and mean  $\pm 1$  standard deviation (—) versus the discrete index sets  $z_{1, \dots, 5} = \{1, 3, 5, 7, 9\}$  and  $z_{1, \dots, 9} = \{1, 2, 3, \dots, 9\}$  (the drawback of this visualization is that off-diagonal elements of the covariance matrix can not be represented explicitly). Row 4 shows two 200-dimensional kernel-Gram matrices for real-valued, equally spaced and sorted indices on the shown interval of  $z$ .<sup>6</sup> Rows 5 and 6 again show corresponding samples (---), mean (—) and  $\pm 1$  standard deviation (▭) versus  $z$ . The dashed and solid interpolations *between* indices were rather associative in rows 2-3, representative for the covariance structure, but no distribution over  $x$  was actually defined there. For the kernel-induced covariances of the GP (rows 4-6), in principle  $z_{1, \dots, T}$

<sup>4</sup> If  $H$  is a selector map, i. e., a projection onto a subspace, like e. g., in Figure 7, instead of an arbitrary linear map, then it is a *convex* combination in the observed parts.

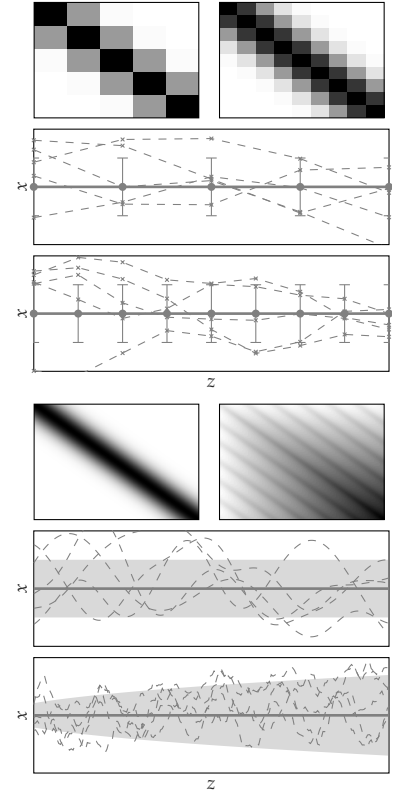


Figure 8: Illustration of continuous indexing. Rows 1-3: 5- and 9-dimensional finite Gaussians. Rows 4-6: 200-dimensional representation of two Gaussian processes.

[109] Rasmussen and Williams, *Gaussian Processes for Machine Learning*, 2006

[73] Karatzas and Shreve, *Brownian Motion and Stochastic Calculus*, 1991

<sup>5</sup> We will mostly skip the curly brackets which indicate *sets*, i. e.,  $\{x_{z_i}\}_{i=1, \dots, T}$  will be denoted as  $x_{z_{1, \dots, T}}$ . We will later also overload this notation and represent *vectors* and *matrices* for double indices with it.

<sup>6</sup> The left covariance matrix is induced by the well-known stationary squared exponential kernel  $k(z, z') = \exp(-\frac{(z-z')^2}{2})$ . The right one  $k(z, z') = \exp(-2 \sin^2(0.5\pi|z-z'|)) + \min(z, z')$  is a combination of a periodic kernel and a Wiener kernel and is non-stationary.

could be chosen arbitrarily dense in the shown interval (or in  $\mathcal{Z}$ ), such that the random vectors approach infinite-dimensional functions.

### Gaussian Observations

Similar to the finite-dimensional case, observations  $y$  of  $x$  with additive Gaussian noise can be easily integrated into GPS:

$$y_{z'_i} = x_{z'_i} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, r), \quad i \in \{1, \dots, M\} \quad (10)$$

where  $z'_{1, \dots, M}$  denotes the set of  $M$  observed locations. Since any finite subset of outputs  $x$  is Gaussian distributed with covariance given by the kernel-Gram matrix, it is possible to define a finite-dimensional Gaussian joint over the outputs  $x_{z_{1, \dots, T}}$  corresponding to predictive locations  $z_{1, \dots, T}$ , and observations  $y_{z'_{1, \dots, M}}$  at locations  $z'_{1, \dots, M}$ . The posterior mean and covariance is given by Eq. 9:

$$\begin{aligned} \mu_{x_{z_{1, \dots, T}} | y_{z'_{1, \dots, M}}} &= \mu_{x_{z_{1, \dots, T}}} + \Sigma_{xy} (\Sigma_{yy} + rI)^{-1} (y_{z'_{1, \dots, M}} - \mu_{x_{z'_{1, \dots, M}}}) \\ \Sigma_{x_{z_{1, \dots, T}} | y_{z'_{1, \dots, M}}} &= \Sigma_{xx} - \Sigma_{xy} (\Sigma_{yy} + rI)^{-1} \Sigma_{yx} \end{aligned} \quad (11)$$

where e. g.,  $\Sigma_{xy}$  is shorthand for the off-diagonal covariance block of the joint with the  $ij^{\text{th}}$  element  $\Sigma_{xy}^{ij} := k(z_i, z'_j)$ ,  $i \in \{1, \dots, T\}$ ,  $j \in \{1, \dots, M\}$  computed by the kernel function; and similarly mean elements, e. g.,  $\mu_{x_{z_{1, \dots, T}}}^i := \mu(z_i)$ ,  $i \in \{1, \dots, T\}$ , computed by the mean function. Since the predictive locations  $z_{1, \dots, T}$  are arbitrary on the continuous index domain,  $\mu_{x_{(\cdot)} | y_{z'_{1, \dots, M}}}$  can be interpreted as posterior mean function and  $\Sigma_{x_{(\cdot)} | y_{z'_{1, \dots, M}}}$  as posterior kernel function which identify the posterior GP.

Figure 9 illustrates the posterior of a GP. It uses the same kernels as in Figure 8: Squared exponential kernel (rows 1-3), and periodic plus Wiener kernel (rows 4-6). Rows 1 and 4 show a 200-dimensional covariance matrix of the prior  $\Sigma_{xx}$  and posterior  $\Sigma_{x|y}$ , left and right plot respectively, for equally spaced and sorted  $z$  on the shown domain. Rows 2 and 5 show the prior, as in rows 5 and 6 of Figure 8; rows 3 and 6 show the posterior. Both kernel choices give rise to quite different posterior structures, i. e., patterns in the posterior covariance matrix and probable regressors, although they were conditioned on the *same* partly noisy observations  $y_{z_{1,2,3}}(\bullet\text{---})$ .<sup>7</sup> Thus kernel design and also fitting of the kernel parameters is a crucial element in GP-regression tasks.<sup>8</sup>

The largest computational cost of GP-inference usually stems from inverting the Gram-matrix  $\Sigma_{yy} + rI \in \mathbb{R}^{M \times M}$  which has cubic cost in its size  $\mathcal{O}(M^3)$ . The next section explores structures of  $\Sigma_{yy}$  which reduce the complexity to linear cost for certain choices of kernels and ‘time’-ordered inputs  $z$ . In these settings *exact* GP-inference can be

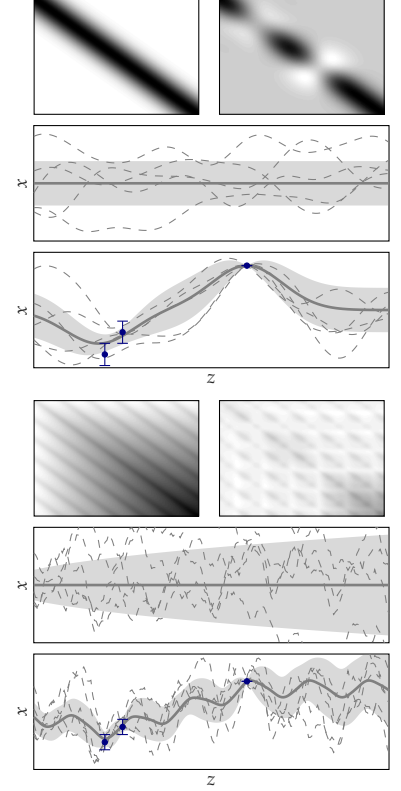


Figure 9: Posterior GP and covariances. Rows 1-3: squared exponential kernel. Rows 4-6: periodic with additive Wiener kernel; colors and both kernels as in rows 4-6 of Figure 8.

<sup>7</sup> The patterns are visible because the indices  $i$  of  $z_i$  are chosen such that the  $z$  are sorted and kernels usually encode some kind of distance or larger-smaller-comparison. A random permutation of  $i$  would render this pattern unrecognizable for the eye, but since the kernel function  $k$  defines how an arbitrary pair of  $(z_i, z_j)$  covaries, the GP is invariant under the permutation as long as the link between input  $z_i$  and output  $x_i$  can be identified.

<sup>8</sup> GPs can also be derived as an infinite limit of parametric linear feature regression (e. g., [109]). Since the infinitely many weights corresponding to these features are the ‘actual’ parameters of the GP-model, it is common to call the kernel parameters ‘hyper-parameters’.

cast as the solution of a linear filtering and smoothing problem of a probabilistic state space model which, from a practical point of view, leverages fast and analytic computations.

Section 1.4 introduces notation for probabilistic state space models, Kalman filters and smoothers, and draws connections in particular to the Wiener process and the related integrated Wiener process.

## 1.4 Wiener Processes & Kalman Filters

An excellent overview over discrete-time probabilistic state space models and Kalman filtering can be found in [117], which this section closely follows.

[117] Särkkä, *Bayesian filtering and smoothing*, 2013

### State Space Models

A *probabilistic state space model* is a Markov-model defined by the initial state distribution  $p(x_0)$  and a sequence of conditional probabilities:

$$x_t \sim p(x_t|x_{t-1}) \quad \text{dynamic model} \quad (12a)$$

$$y_t \sim p(y_t|x_t) \quad \text{measurement model} \quad (12b)$$

The vector  $x_t \in \mathbb{R}^D$  is the hidden state of the system at iteration  $t$ , and  $y_t \in \mathbb{R}^{D_y}$  is a possibly noisy measurement related to this state. Since the model is Markov<sup>9</sup>, Eq. 12 together with an initial distribution  $p(x_0)$  fully describes the joint probability distribution over all  $x_t, t = 0, \dots, T$ , and  $y_t, t = 1, \dots, T$ . The discrete-time *filtering equations* are recursively given by the *predictive* marginal distributions  $p(x_{t+1}|y_{1,\dots,t})$  and *updated* marginal distributions  $p(x_t|y_{1,\dots,t})$ . The discrete-time *smoothing equations* are given by the marginal distributions  $p(x_t|y_{1,\dots,T})$  conditioned on *all* observations up to time step  $T$ .

<sup>9</sup>This means that the distribution of  $x_t$  is independent of past  $x_i, i = 0, \dots, t-2$ , given the preceding one, i. e.,  $p(x_t|x_{0,\dots,t-1}) = p(x_t|x_{t-1})$ .

### 1.4.1 Kalman Filter

Given a state space model as in Eq. 12, the *Kalman filter* is the closed form solution of the discrete-time filtering equations for *linear* dynamics and *linear* measurements with additive Gaussian disturbances, and Gaussian initial condition:

$$x_0 \sim \mathcal{N}(m_0, P_0) \quad (13a)$$

$$x_{t+1} = A_t x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, Q_t) \quad \text{dynamic model} \quad (13b)$$

$$y_t = H_t x_t + \nu_t, \quad \nu_t \sim \mathcal{N}(0, R_t) \quad \text{measurement model} \quad (13c)$$

$A_t \in \mathbb{R}^{D \times D}$  is called the *transition matrix*, since it transitions the old state  $x_t$  into the deterministic part of the new one  $x_{t+1}$ , and  $H_t \in \mathbb{R}^{D_y \times D}$  is called the *measurement matrix*, since it maps the state  $x_t$  onto



the space where the measurement  $y_t$  happens.  $Q_t$  and  $R_t$  are the covariances of the diffusion and the measurement noise respectively. Under these assumptions the probabilities of Eq.12 as well as the filtering equations can be rephrased as:

$$\begin{aligned}
 p(x_0) &= \mathcal{N}(x_0; m_0, P_0) \\
 p(x_{t+1}|x_t) &= \mathcal{N}(x_{t+1}; A_t x_t, Q_t) \\
 p(x_{t+1}|y_1, \dots, y_t) &= \int p(x_{t+1}|x_t) p(x_t|y_1, \dots, y_t) dx_t = \mathcal{N}(x_{t+1}; m_{t+1-}, P_{t+1-}) \\
 p(y_{t+1}|x_{t+1}) &= \mathcal{N}(y_{t+1}; H_{t+1} x_{t+1}, R_{t+1}) \\
 p(x_{t+1}|y_1, \dots, y_{t+1}) &\propto p(y_{t+1}|x_{t+1}) p(x_{t+1}|y_1, \dots, y_t) = \mathcal{N}(x_{t+1}; m_{t+1}, P_{t+1}).
 \end{aligned} \tag{14}$$

All expressions in Eq. 14 are analytic for Gaussian distributions. The explicit forms for the means and covariances divide into two *predictive* equations for the mean vector  $m_{t+1-} \in \mathbb{R}^D$  and covariance matrix  $P_{t+1-} \in \mathbb{R}^{D \times D}$ , and two *update* equations for  $m_{t+1} \in \mathbb{R}^D$  and  $P_{t+1} \in \mathbb{R}^{D \times D}$ . They have the general forms:<sup>10</sup>

$$\begin{aligned}
 m_{t+1-} &= A_t m_t \\
 P_{t+1-} &= A_t P_t A_t^\top + Q_t \\
 m_{t+1} &= m_{t+1-} + P_{t+1-} H_{t+1}^\top [H_{t+1} P_{t+1-} H_{t+1}^\top + R_{t+1}]^{-1} [y_{t+1} - H_{t+1} m_{t+1-}] \\
 P_{t+1} &= P_{t+1-} - P_{t+1-} H_{t+1}^\top [H_{t+1} P_{t+1-} H_{t+1}^\top + R_{t+1}]^{-1} H_{t+1} P_{t+1-}.
 \end{aligned} \tag{15}$$

Equations 15 are commonly known as the *Kalman filter equations* [72]. Often, the last two rows of Eq. 15 are presented in a slightly different way:

$$\begin{aligned}
 G_{t+1} &= H_{t+1} P_{t+1-} H_{t+1}^\top + R_{t+1} \\
 g_{t+1} &= P_{t+1-} H_{t+1}^\top G_{t+1}^{-1} \\
 m_{t+1} &= [I - g_{t+1} H_{t+1}] m_{t+1-} + g_{t+1} y_{t+1} \\
 P_{t+1} &= [I - g_{t+1} H_{t+1}] P_{t+1-}
 \end{aligned} \tag{16}$$

where  $G_{t+1}$  is the *innovation covariance* and  $g_{t+1}$  is the *Kalman gain*. The gain encodes how much the current predictive estimator  $H_{t+1} m_{t+1-}$  is trusted in comparison to the noisy measurement  $y_{t+1}$ . If observations  $y_{t+1}$  are noise free ( $R_{t+1} = 0$ ), then the observed part of the state  $x_{t+1}$  collapses on  $H_{t+1} m_{t+1} = y_{t+1}$  with  $H_{t+1} P_{t+1} H_{t+1}^\top = 0$ .

### Connection to Stochastic Differential Equations

A linear, time-invariant stochastic differential equation (SDE) is of the form:

$$dx = (Fx)dz + Ld\beta. \tag{17}$$

The first term  $(Fx)dz$  is called the *drift* and describes the deterministic dynamics of the SDE;  $Ld\beta$  is called *diffusion* and encodes the stochastic

<sup>10</sup> A more common way to denote the mean and covariance of the predictive state is  $m_{t+1}^-$  and  $P_{t+1}^-$ , instead of  $m_{t+1-}$  and  $P_{t+1-}$ . So, this slightly awkward sub-script is introduced, because in later chapters we will also heavily use superscripts in addition.

[72] Kalman, "A New Approach to Linear Filtering and Prediction Problems," 1960

contribution;  $\beta$  is a Wiener process with *intensity*  $q$  that will be defined in Section 1.4.2. The solution of this SDE with initial distribution  $x_{z_0} \sim \mathcal{N}(m_{z_0}, P_{z_0})$  is a Gaussian process (e. g., [73, § 2.9 and 5.6] or [118, Theorem 2.7]) with mean  $m(z)$  and covariance  $P(z, z_0)$ , for  $z \geq z_0$ :

$$m(z) = \exp(F\Delta z) m_{z_0} \quad (18a)$$

$$P(z, z_0) = \exp(F\Delta z) P_{z_0} \exp^\top(F\Delta z) + \int_{z_0}^z \exp(F(z - \kappa)) LqL^\top \exp^\top(F(z - \kappa)) d\kappa. \quad (18b)$$

The difference  $\Delta z = z - z_0$  is the traveled path, often also called ‘time’. The forms of  $F$ ,  $L$  and  $q$  thus define the forms of  $Q$  and  $A$  of Eq. 13; in particular

$$\begin{aligned} x_{t+1} &= A_t x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, Q_t) \\ A_t &= \exp(F\Delta z) = \sum_{k=0}^{\infty} \frac{(F\Delta z)^k}{k!} \\ Q_t &= \int_{z_0}^z \exp(F(z - \kappa)) LqL^\top \exp^\top(F(z - \kappa)) d\kappa, \end{aligned} \quad (19)$$

where  $x_t := x_{z_0}$  is the current state and  $x_{t+1} := x_z$  the predicted state. The dynamic model of the Kalman filter is thus the exact *discretized* solution of the time-invariant linear SDE of Eq. 17, even if the underlying dynamics are continuous. The Kalman filter predictive equations compute the marginal means and the covariances of the exact solution.

### Marginal Likelihood

Maximum marginal likelihood estimation is a way of fitting a parameterized model to data. Suppose that observations occur in a finite interval of time steps  $1, \dots, T$ . Eqs. 13, 17 and 19 define a class of probabilistic models which is parameterized by  $\theta := \{F, L, q, R_{1, \dots, T}\}$ . The marginal<sup>11</sup> likelihood or *evidence*  $p_\theta(y_{1, \dots, T})$  for one of these models is the integral

$$p_\theta(y_{1, \dots, T}) = \int p(y_1, \dots, y_T | x_0, \dots, x_T) p(x_0, \dots, x_T) dx_0 \dots dx_T. \quad (20)$$

This is the probability of observing  $y_{1, \dots, T}$ , given that the filtering model, indexed by  $\theta$ , is the true data-generating process. The subscript  $\theta$ , omitted in other places, emphasizes this dependence. Usually Eq. 20 involves solving a non-trivial integral, but in the Kalman filtering framework it is analytic and cheap since the model is Markov and Gaussian:

$$\begin{aligned} p_\theta(y_{1, \dots, T}) &= \int \prod_{t=1}^T p(y_t | x_t) p(x_t | x_{t-1}) p(x_0) dx_t dx_0 \\ &= \int \prod_{t=1}^T \mathcal{N}(y_t; H_t x_t, R_t) \mathcal{N}(x_t; A_t x_{t-1}, Q_t) \mathcal{N}(x_0; m_0, P_0) dx_t dx_0 = \prod_{t=1}^T \mathcal{N}(y_t; H_t m_{t-}, H_t P_{t-} H_t^\top + R_t). \end{aligned} \quad (21)$$

[73] Karatzas and Shreve, *Brownian Motion and Stochastic Calculus*, 1991

[118] Särkkä, “Recursive Bayesian Inference on Stochastic Differential Equations,” 2006

<sup>11</sup> The term ‘marginal’ is used to distinguish from the likelihood  $p_\theta(y_{1, \dots, y_T} | x_0, \dots, x_T)$ ,  $x \sim \mathcal{GP}_\theta$ . Note that both  $p$  as well as  $\mathcal{GP}$  can depend on (parts of)  $\theta$ .



Thus the *logarithmic* marginal likelihood becomes:<sup>12</sup>

$$\log p_\theta(y_{1,\dots,T}) = -\frac{1}{2} \sum_{t=1}^T \left[ (y_t - H_t m_{t-})^\top G_t^{-1} (y_t - H_t m_{t-}) + \log |G_t| \right] + \text{const.} \quad (22)$$

Eq. 22 mostly contains terms which are being computed already in the prediction and update step of the Kalman filter (Eq. 15, 16), thus tracking  $p_\theta(y_{1,\dots,T})$  during filtering only adds minimal computational overhead ( $\log |G_t|$  is cheap since  $G_t^{-1}$  is computed anyway). There is an intuitive explanation to both terms in Eq. 22: The first one, quadratic in  $(y_t - H_t m_{t-})$ , roughly speaking tries to fit the mean prediction  $H_t m_{t-}$  of the model as close as possible to the observation  $y_t$  (bias term); the second term  $\log |G_t|$  is related to the differential entropy of the filtering distribution and enforces low variance in the prediction.

Figure 10 shows a simple example of a one-dimensional Gaussian random walk ( $F = 0, L, H = 1, R_{1,2}$  given) with two noisy observations  $y_{1,2}$ . The top plot shows  $\log p_\theta(y_{1,2})$  versus the only free parameter  $q \in \mathbb{R}_+$ . The bottom plot shows the observations  $y_t \pm R_t^{1/2}$  ( $\bullet$ ) as well as filtering distributions for three different  $q$ : The model with a large  $q$  ( $\square$ ) fits the observations well, but is under-confident, while the model with a small  $q$  ( $\square$ ) does not fit the observations well and is over-confident at the same time. The max-marginal likelihood estimator for  $q$  yields a model ( $\square$ ) which is a trade-off between fitting the datapoints well without sacrificing too much predictive power.

The maximum marginal likelihood estimator  $\hat{\theta}$  is the global maximizer of Eq. 22. In some (rare) cases  $\hat{\theta}$  can even be found analytically. Alternatively, a gradient based optimizer can be used, if the gradient with respect to  $\theta$  is available. This is usually straightforward and quite user-friendly, but it requires multiple runs of the filter (one after each update of the outer optimization loop) and also only finds local maxima of  $\log p_\theta(y_{1,\dots,T})$ . If the dimensionality of  $\theta$  is not too large, other approaches include Bayesian optimization (BO) which strategically searches for parameters with a large marginal likelihood (§ 4.1). All three approaches are not applicable if the filter needs to adapt its parameters  $\theta$  *online*, i. e., while it is running, which is the case that is mostly considered in this thesis.

### Smoother

A filter computes the solution of the discrete-time filtering equations. These are the marginal probabilities of the state  $x_t$  conditioned only on the history as well as the current observation  $y_{1,\dots,t}$ , meaning that possible future observations  $y_{t+1,\dots,T}$  do not affect older states. This is especially useful when only the distribution of latest state is needed, e. g., in online applications, since it depends on all available data up to that point in time. The *smoother* computes the solution of the discrete-

<sup>12</sup> The logarithm is a monotonic function and does not alter the location of optima. It is usually used since it is numerically more stable than  $p_\theta$  itself.

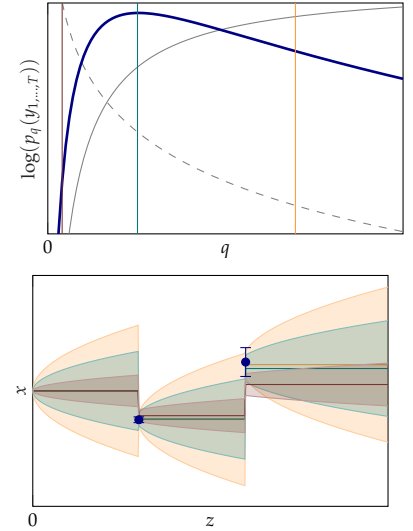


Figure 10: Maximum marginal likelihood. Top: log marginal likelihood versus different values of intensity  $q$  (—), composed of a variance part (---), and a bias part (—). Bottom: filters (mean solid,  $\pm 1$  standard deviation shaded) for the best fit ( $\square$ ), and too small/too large  $q$  fits ( $\square$ / $\square$ ). Corresponding  $q$  indicated in the top plot as vertical lines; observations  $y$  with error bars  $\pm R^{1/2}$  ( $\bullet$ ).

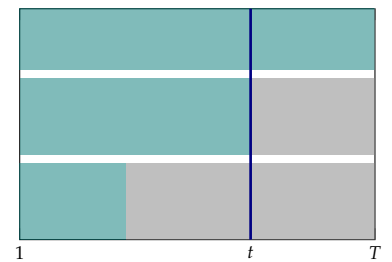


Figure 11: Illustration of predicting, filtering and smoothing. State estimate (—), observations (■). Top: smoothing. Middle: filtering. Bottom: prediction. Figure inspired by Figure 1.7 in [117].

time smoothing equations. It modifies the filtered solution such that the state  $x_t$  is conditioned on *all*, also future observations (illustration in Figure 11). This is useful if the marginal probabilities of each  $x_t$  conditioned on all observations  $y_{1,\dots,T}$  is needed, e. g., in regression tasks, but it also requires additional computations. They are about the same cost as the filtering run with complexity linear in  $T$ . In particular, given the filtered predictive and updated probabilities with means and covariances as in Eq. 15 we get the backwards recursion for the smoothed marginal probabilities  $p(x_t|y_{1,\dots,T}) = \mathcal{N}(x_t; m_t^s, P_t^s)$ :

$$\begin{aligned}\tilde{A}_t &= P_t A_t^\top P_{t+1}^{-1} \\ m_t^s &= m_t + \tilde{A}_t (m_{t+1}^s - m_{t+1}) \\ P_t^s &= P_t + \tilde{A}_t (P_{t+1}^s - P_{t+1}) \tilde{A}_t^\top.\end{aligned}\quad (23)$$

The recursion is initialized with  $m_T^s = m_T$  and  $P_T^s = P_T$  since the last distribution  $p(x_T|y_{1,\dots,T})$  of the filtering state is already conditioned on all data and is thus equivalent to its smoothed distribution. Eq. 23 is commonly known as the *Rauch–Tung–Striebel smoother* [110]. Its derivation is based on the Markov property that  $p(x_t|x_{t+1}, y_{1,\dots,T}) = p(x_t|x_{t+1}, y_{1,\dots,t})$ .

#### 1.4.2 Wiener Process

A Wiener process (WP) is a stochastic process  $x$  with continuous sample paths,  $x(0) = 0$ , and independent increments  $x(z + \delta z) - x(z)$  for every  $\delta z$  that are Gaussian distributed with zero mean and variance  $\delta z$ . A proper mathematical definition can be found e. g., in [73, § 2.1]. The Wiener process  $\beta(z)$  is a special Gaussian process [73, § 2.9]. It is also the solution to the arguably most basic SDE of Eq. 17: A one-dimensional state  $x \in \mathbb{R}$ , no dynamic component ( $F = 0$ ), unit dispersion ( $L = 1$ ) and initial condition  $x(0) = 0$ ; thus  $x(z) = \beta(z)$ . In fact, this is how the stochastic part of the SDE was defined in the first place. Let us for now denote this simple one-dimensional Wiener process with  $\beta_1(z)$ . A  $D$ -dimensional *correlated* Wiener process with positive definite intensity matrix  $q \in \mathbb{R}^{D \times D}$  is of the form  $x(z) = q^{\frac{1}{2}} \beta(z)$ , where  $\beta(z) \in \mathbb{R}^D$  is a vector of  $D$  independent one-dimensional Wiener processes  $\beta_1(z)$ , and  $q^{\frac{1}{2}}$  is the square root of the matrix  $q$ . For simplicity, we will use the word ‘Wiener process’ also for the  $D$ -dimensional correlated Wiener process. Especially in physics the Wiener process is also known as *Brownian motion*.<sup>13</sup>

The corresponding kernel function  $k(z, z') = \min(z, z')$  can be found by computing the covariance between two arbitrary outputs  $x(z)$  and  $x(z')$ . This means that the Wiener process can be represented either as a state space model, or by a GP with kernel function  $k$ . The solutions for the prior and posterior induced by the vanilla ‘kernel’-

[110] Rauch, Striebel, and Tung, “Maximum likelihood estimates of linear dynamic systems,” 1965

[36] Einstein, “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen,” 1905

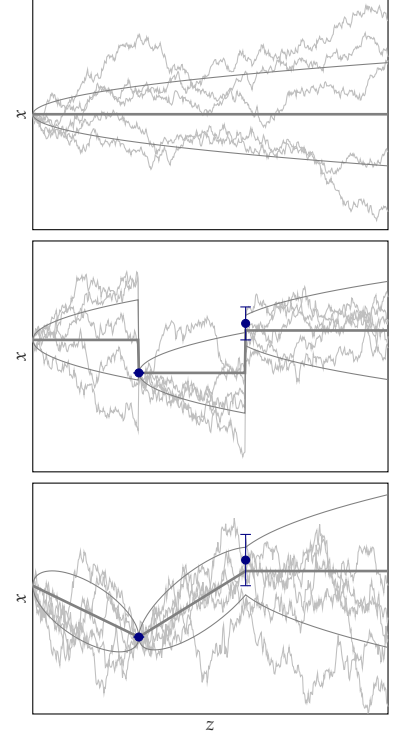


Figure 12: Wiener process/Filtering and smoothing. Mean (—),  $\pm 1$  standard deviation (---), 5 samples (—), observations  $y$  with error bars  $\pm R^{1/2}$  (—•). Top: prior. Middle: filter. Bottom: posterior/smoother.

[73] Karatzas and Shreve, *Brownian Motion and Stochastic Calculus*, 1991

<sup>13</sup>Named after the botanist Robert Brown. One of the first mathematical descriptions include one of Albert Einstein’s famous 1905-papers “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen” [36], where he mentions the ‘Brownian molecular movement’ but could not decide if his description of molecular kinetics matches those of the, at that time, heuristic notion of Brownian motion; even earlier mentions were done e. g., by the Danish astronomer Thorvald N. Thiele in 1880.

view and the state space view are identical for initial condition  $x_0 = 0$ , but the algorithmic complexity in the number of datapoints  $M$  of the latter is drastically reduced: Linear cost in  $M$  instead of cubic cost in  $M$ . This is due to the fact that, although the kernel-Gram matrix  $\Sigma_{xx}$  is dense, its *inverse* is block-tridiagonal and is represented explicitly in the state space view. The filtering model thus never constructs  $\Sigma_{xx}$  and  $\Sigma_{yy}$  explicitly but works directly with the block-tridiagonal inverses [8], [50], [51]. Vanilla GP-inference on the other hand simply does not exploit this structure. Practical approaches and work on mapping Gaussian processes to state space models, exact or with approximations, include e. g., [54], [8], [50], [129], and [51].

Figure 12 illustrates the Wiener process: The top and bottom plot show the prior and posterior respectively, again for equally spaced and sorted  $z$  on the shown interval (symbols and colors as in Figure 9). The middle plot shows the filtering distribution where successive  $x_t$  are only conditioned on observations  $y$  up to this point in time  $z_t$ . The sampled functions are continuous but non-differentiable (by construction), but the mean function is *constant* for the prior and *piece-wise linear* (between observations) for the posterior. This is intuitive, since the drift is zero ( $F = 0$ ), thus the transition matrix  $A = \exp(0) = 1$  is the unity operator and the independent Gaussian increments are zero in expectation. The standard deviation (std) of the prior grows with the *square root* of  $z$  because the diffusion variance grows linearly with  $z$ :  $Q = \int_{z_0}^z \exp(0)^2 d\kappa = z - z_0$ .

### Integrated Wiener Process

The *integrated* Wiener process (IWP) is the time-integral of the Wiener process. Since integration is a linear operator, the IWP is also a Gaussian process. Its kernel function is the kernel function of the Wiener process integrated in both arguments:

$$k(z, z') = \frac{1}{3} \min^3(z, z') + \frac{1}{2} |z - z'| \min^2(z, z'). \quad (24)$$

The corresponding two-dimensional state  $x = [x^f, x^{f'}]^\top$  of the state space model represents a function  $x^f$  and its time-derivative  $x^{f'}$  which are correlated through the dynamic model. The corresponding SDE is:

$$d \begin{bmatrix} x^f \\ x^{f'} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x^f \\ x^{f'} \end{bmatrix} dz + \begin{bmatrix} 0 \\ 1 \end{bmatrix} d\beta = \begin{bmatrix} x^{f'} dz \\ d\beta \end{bmatrix}. \quad (25)$$

The gradient  $x^{f'}$  still evolves like a Wiener process and the function value  $x^f$  is deterministically connected to it through the integral ( $x^{f'} dz$ ). From Eq. 25 we can also read off  $F = [[0, 1]^\top [0, 0]^\top]$  and

[8] Barfoot, Tong, and Särkkä, "Batch Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression," 2014

[50] Grigorievskiy and Karhunen, "Gaussian Process Kernels for Popular State-Space Time Series Models," 2016

[51] Grigorievskiy, Lawrence, and Särkkä, "Parallelizable sparse inverse formulation Gaussian processes (Sp-InGP)," 2016

[54] Hartikainen and Särkkä, "Kalman filtering and smoothing solutions to temporal Gaussian process regression models," 2010

[129] Solin and Särkkä, "Explicit link between periodic covariance functions and state space models," 2014

$L = [0, 1]^T$ . The drift matrix  $F$  is nilpotent of order two ( $F^2 = 0$ ), thus the matrix exponential in  $A_t$  and  $Q_t$  is analytic:

$$A_t = \exp(F\Delta z) = \sum_{k=0}^{\infty} \frac{(F\Delta z)^k}{k!} = I + F\Delta z = \begin{bmatrix} 1 & \Delta z \\ 0 & 1 \end{bmatrix}. \tag{26}$$

$$Q_t = \int_{z_0}^z \begin{bmatrix} (z - \kappa) \\ 1 \end{bmatrix} q \begin{bmatrix} (z - \kappa) & 1 \end{bmatrix} d\kappa = q \begin{bmatrix} \frac{1}{3}\Delta z^3 & \frac{1}{2}\Delta z^2 \\ \frac{1}{2}\Delta z^2 & \Delta z \end{bmatrix}, \quad \text{with } \Delta z = z - z_0 = z_{t+1} - z_t.$$

The marginal of the derivative block  $x^{f'}$  again evolves like a Wiener process with constant mean,  $\text{std} \propto z^{\frac{1}{2}}$ . The marginal of  $x^f$  evolves like an integrated Wiener process with standard deviation proportional to  $z^{\frac{3}{2}}$  and a mean function that is *linear* for the prior and *piece-wise cubic* (between observations) for the posterior. Again this is intuitive since the drift term is linear thus the transitions matrix  $A_t$  is the identity operator plus a linear contribution proportional to the derivative.

Figure 13 illustrates the integrated Wiener process: The top and bottom plots show the prior and posterior respectively (colors as in Figure 12). The samples  $[x^f, x^{f'}]$  are jointly drawn from the iwr and represent once-differentiable functions: The differentiated parts of the samples  $x^{f'}$  are again continuous but non-differentiable, the function value parts  $x^f$  are smooth as can be seen from the figure.

In the state space notation it is also straightforward to include partly observed states and differing observation noise. In particular in Figure 13 the first two datapoints only observe the derivative:  $H_1 = H_2 = [0, 1]^T$  with noise free ( $R_1 = 0$ )  $y_1 \in \mathbb{R}$ , and noisy ( $R_2 > 0$ )  $y_2 \in \mathbb{R}$ , the third datapoint observes the whole state:  $H_3 = I, y_3 \in \mathbb{R}^2$ , but noise corrupted  $R_3 = \text{diag}([R_3^f, R_3^{f'}]) \in \mathbb{R}^{2 \times 2}$ .

The filtering formulation of probabilistic state space models and the connection to Gaussian processes will be used heavily in Chapters 8 and 9.

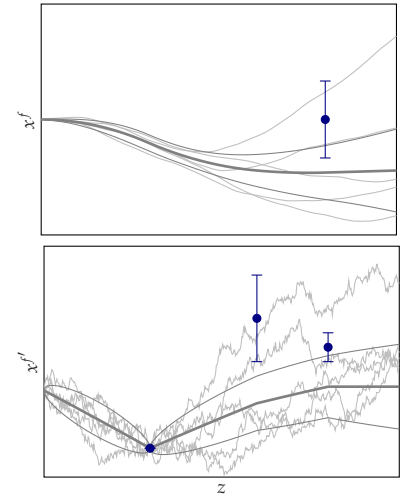


Figure 13: Integrated Wiener process. Mean (—),  $\pm 1$  standard deviation (—), 5 samples (—), observations  $y$  with error bars  $\pm R^{1/2}$  (—•—). Top: posterior marginal on function value. Bottom: posterior marginal on derivative.

## Empirical Risk Minimization

**G**IVEN some dataset  $\mathcal{D}$  containing pairs of input  $x \in \mathcal{X}$  and target  $y \in \mathcal{Y}$ , the task of *classification* is to learn some general, predictive law between the input and categorical target-space of the data-distribution. The same holds for *regression*, where the targets are real-valued. A common approach is to choose a class of functional relationships  $f_w(x)$ , indexed by parameters  $w$ , as a surrogate for this map, as well as a loss  $\ell_f(y)$  which penalizes discrepancies between the predictor  $f_w(x)$  and the targets  $y$ . Then, one function  $f_{w^*}(x)$  is selected by minimizing the *empirical risk* (Section 2.1), which is the expected loss over the empirical distribution defined by the dataset  $\mathcal{D}$ . A powerful as well as algebraically appealing class of functional relationships  $f_w(x)$ , are artificial neural networks (Section 2.2) which exhibit a particular structure in their gradients  $\nabla f_w(x)$  that enables their efficient computation, and thus sped-up optimization of the parameters  $w$ . Sections 2.3 and 2.4 introduce some of the most common iterative solvers for this optimization problem, first for exact, and then for noise corrupted evaluations of the empirical risk. Some of the text is based on the introductory chapters of the publications listed at the end of Chapter 0.

### 2.1 Risk and Empirical Risk

Many contemporary machine learning problems involve minimizing an expected loss (risk) of the form

$$\mathcal{L}(w) = \mathbf{E}_{d \sim Q} [\ell(f_w(x), y)], \quad \text{risk} \quad (27)$$

where  $w \in \mathbb{R}^N$  is the parameter vector, and  $\ell(f_w(x), d)$  quantifies how well  $w$  performs on datapoint  $d = (x, y) \in \mathcal{X} \times \mathcal{Y}$ .<sup>1</sup> In practice, though it is often not possible to compute the expectation in 27 precisely, and instead the empirical risk, define by a dataset  $\mathcal{D}$  of size  $|\mathcal{D}|$ , is optimized

$$L_{\mathcal{D}}(w) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \ell(f_w(x), y). \quad \text{empirical risk} \quad (28)$$

The datapoints contained in  $\mathcal{D}$  are assumed to be independent samples from the (usually unknown) data-distribution  $Q$  (Figure 14). Likewise

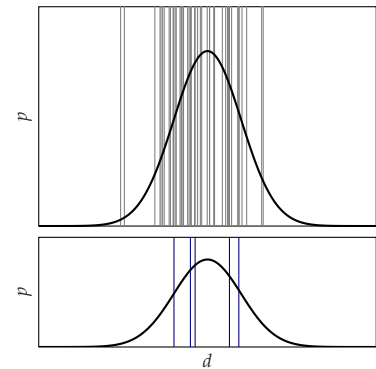


Figure 14: Illustration of empirical distribution  $\hat{Q} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \delta(d)$ . Top: The point masses  $\delta(d)$  are indicated as vertical bars (—).  $\hat{Q}$  is used to approximate the true data-distribution  $Q$  (—, shown as pdf in this sketch). Especially areas of large ‘gaps’ between point masses (but non-trivial density  $q$ ) are not modeled well by  $\hat{Q}$ . Bottom: Same as top plot but for a mini-batch  $\mathcal{B} \subset \mathcal{D}$  with  $|\mathcal{B}| \approx 0.1|\mathcal{D}|$ . The approximation is even coarser, especially rare samples and low density regions are not represented well.

<sup>1</sup> Depending on which dependence we want to emphasize we will use short-hands like  $\ell_f(y)$ ,  $\ell_w$ ,  $\ell_f$ , or  $\ell(f)$  for the loss as well. The same holds for the function  $f$ .

the gradient of the risk  $\nabla \mathcal{L}(w) = \mathbf{E}_{d \sim Q} [\nabla \ell(f_w(x), y)]$  with respect to  $w$  can be approximated by

$$\nabla L_{\mathcal{D}}(w) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \nabla \ell(f_w(x), y). \quad (29)$$

When the size  $|\mathcal{D}|$  of the dataset is large, even computing  $L_{\mathcal{D}}$  might pose challenging, and  $\nabla \mathcal{L}$  is approximated with an even coarser estimator  $\nabla L_{\mathcal{B}}$  by sub-sampling a *mini-batch*  $\mathcal{B} \subset \mathcal{D}$

$$\nabla L_{\mathcal{B}}(w) = \frac{1}{|\mathcal{B}|} \sum_{d \in \mathcal{B}} \nabla \ell(f_w(x), y). \quad (30)$$

The number of elements  $|\mathcal{B}|$  in  $\mathcal{B}$  is referred to as the mini-batch size. We see here already that there is no fundamental difference between choosing  $\nabla L_{\mathcal{D}}$  or  $\nabla L_{\mathcal{B}}$  as local (i. e., for a given  $w$ ) estimator for  $\nabla \mathcal{L}$ . Both of them are noisy, unbiased estimators of the same quantity using a dataset which is sampled i. i. d. from the true data-distribution. The former though is much more precise, meaning that its variance is reduced by a factor of  $(|\mathcal{D}|/|\mathcal{B}|)^{-1/2}$ . This will be further discussed in Chapter 5.

## 2.2 Artificial Neural Networks

An artificial neural network<sup>2</sup> is a function  $f_w(x)$  parametrized by  $w \in \mathbb{R}^N$  which maps from some input space  $x \in \mathcal{X} \subseteq \mathbb{R}^{n_0}$  to a target space  $y \in \mathcal{Y} \subseteq \mathbb{R}^{n_L}$ . In addition,  $f_w(x)$  is a concatenation of alternating affine maps with linear transformations  $W_l$  (weights), translations  $b_l$  (biases), and non-linear maps  $\sigma_l$ ; e. g., for one linear, and one non-linear map:

$$f_w(x) = \sigma_1(W_1x + b_1), \quad w = \{W_1, b_1\}. \quad (31)$$

There is a variety of different types of neural networks, which differ in the structure of how linear and non-linear maps are arranged. These include rather classic ones like perceptrons or multi-layer perceptrons [114] [65], and more recent developments like convolutional neural networks (CNN) [83], or recurrent neural networks [68] [66]. Also logistic regression is a simple instance of a neural network<sup>3</sup> that even has a convex optimization surface (in general, neural networks are non-convex functions and yield non-convex risks). A recent overview on networks and developments can e. g., be found in [48]. The experimental part of this thesis mostly considers logistic regressors and multi-layer perceptrons (MLPs). The next section will thus focus simply on the computations involved in finding  $f_w(x)$  and  $\nabla f_w(x)$  of fully-connected nets. These restrictions do not mean, however, that the results obtained in Parts II, III, and IV can not in the future be

<sup>2</sup> The name ‘neural network’ was motivated by the structure of  $f_w(x)$  which was originally designed after the brain, where neurons (nodes) are connected to each other through neural pathways/weights of varying strength, that ideally can be learned by observing examples. The literature is vast, starting approximately in the 1970s in the machine learning community, so the citations below only capture some of the contributions.

<sup>3</sup> Precisely, that is for  $f_w(x)$  as in Eq. 31,  $\sigma_1$  is the sigmoidal function,  $\ell_f(y)$  the cross-entropy loss, and  $y \in \{0, 1\}$ .

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

[114] Rosenblatt, “The Perceptron—A Perceiving and Recognizing Automaton,” 1957

[65] Hinton and Sejnowski, “Optimal Perceptual Inference,” 1983

[83] LeCun et al., “Object Recognition with Gradient-Based Learning,” 1999

[68] Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” 1982

[66] Hochreiter and Schmidhuber, “Long Short-Term Memory,” 1997



extended to CNNs or the like, and we will refer to (ongoing) related works towards that direction throughout the text.

### 2.2.1 Multi-Layer Perceptrons

#### Forward Pass

In general, for  $L$  linear and  $L$  non-linear maps,  $f_w(x)$  can be computed recursively for mini-batch-inputs  $X \in \mathbb{R}^{n_0 \times |\mathcal{B}|}$  as:

$$a_0 = X \quad \text{first activation is equal to input} \quad (32a)$$

$$z_l = W_l^\top a_{l-1} + b_l \quad \text{linear layer} \quad (32b)$$

$$a_l = \sigma_l(z_l) \quad \text{non-linear layer, activations} \quad (32c)$$

$$f_w = a_L = \sigma_L(z_L) \quad \text{last activation is equal to output} \quad (32d)$$

$$\ell_f = \ell(f_w, Y) \quad \text{individual losses} \quad (32e)$$

$$L_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \ell_f^\top \mathbf{1}_{|\mathcal{B}| \times 1} \quad \text{mean loss of mini-batch} \quad (32f)$$

The parameters  $w$  are the set of all weights and biases  $w = \{W_1, b_1, \dots, W_L, b_L\}$ . Computing one or multiple function values  $f_w(X)$  as in Eq. 32 is called a *forward pass*. A collection of variables and maps of a single index  $l$  that only involve affine transformations ( $W_l, b_l, z_l$ ) is loosely called a *linear layer*, while a collection involving a non-linear transform ( $\sigma_l, a_l$ ) is called a *non-linear layer*. All variables which have no direct connection to the input  $x$  or the output  $f_w$  are called *hidden layers*, the remaining two layers are called *input* and *output* layer. Depending on the number of hidden layers, a network is vaguely called *deep* (many) or *shallow* (none or little). The sizes of all matrices of Eq. 32, except input and output, are fixed by the mini-batch size  $|\mathcal{B}|$  as well as the user-defined *architecture* of the network; they will be denoted as  $n_l$  with  $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$  and  $b_l, z_l, a_l \in \mathbb{R}^{n_l \times |\mathcal{B}|}$ .<sup>4</sup>

#### Backpropagation

The recursive structure of Eq. 32 immensely simplifies the computation of  $f_w(X)$  but more importantly of the gradient  $\nabla f_w(x)$  with respect to  $w$  because (analogously to the forward pass in Eq. 32) the

[11] Bergstra et al., “Algorithms for hyper-parameter optimization,” 2011

[69] Hutter, Hoos, and Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” 2011

[127] Snoek, Larochelle, and Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” 2012

[84] Li et al., “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization,” 2016

<sup>4</sup>Finding the right architecture is another area of research, which will not feature here. Popular recent approaches include systematic automated hyperparameter searches e. g., by Bayesian optimization [11] [69] [127] [84].

chain rule for computing  $\nabla f_w(x)$  decomposes into successive low-dimensional matrix vector products:

$$\delta_L = \frac{\partial \ell_f}{\partial f_w} \quad \text{output residual} \quad (33a)$$

$$\nabla \ell_L = a_{L-1} \delta_L^\top \quad \text{output gradient} \quad (33b)$$

$$\delta_l = (W_{l+1} \delta_{l+1}) \odot \frac{\partial a_l(z_l)}{\partial z_l} \quad \text{residual of layer } l \quad (33c)$$

$$\nabla \ell_l = a_{l-1} \delta_l^\top \quad \text{gradient of layer } l \quad (33d)$$

where  $\delta_l \in \mathbb{R}^{n_l \times |\mathcal{B}|}$  and  $\nabla \ell_l \in \mathbb{R}^{n_{l-1} \times n_l}$ . For biases  $b_l$ , the gradient per layer is  $\nabla \ell_l = \mathbb{1}_{1 \times |\mathcal{B}|} \delta_l^\top$ . Eq. 33 is also called *backpropagation* [116] [81]. As in Eq. 32 the matrix-vector products are of the size of the mini-batch as well as the weights, biases and activations of the individual layers, and not the whole net, and can be computed efficiently and in parallel on CPUs and GPUs.

[116] Rumelhart, Hinton, and Williams, "Learning representations by back-propagating errors," 1986

[81] LeCun et al., "Efficient BackProp," 1998

### Losses

The loss  $\ell_f(y) = \ell(f_w(x), y)$  encodes how much a misfit of  $f_w(x)$  to the corresponding target  $y$  is penalized, or, in other words, how well  $f_w(x)$  performs in hitting the target  $y$ . The choice of the loss is usually up to the user, a typical one for regression problems is the *squared* loss  $\ell_f(y) = \frac{1}{2}(f_w(x) - y)^2$  which quadratically penalizes misfits of  $f_w(w)$  on  $y$ . A usual choice for binary classification tasks is the *cross-entropy* loss  $\ell_f(y) = -y \log(f_w(x)) - (1 - y) \log(1 - f_w(x))$ , for targets  $y \in \{0, 1\}$ . Both can be interpreted as negative log-likelihoods,  $-\log p(y|f_w(x))$  of the parameters  $w$ , or as conditional distributions of  $y$  given  $f_w(x)$ : a Gaussian with mean  $f_w(x)$  and arbitrary finite variance for the squared loss, and Bernoulli with parameter  $f_w(x)$  for the cross-entropy loss. Thus training an MLP can sometimes be seen as finding the maximum likelihood estimator of the parameters of a probabilistic discriminative model. Not all losses though have this interpretability (only if the integral of  $\exp[-\ell_f(y)]$  over  $y$  is finite).

Multi-class classification tasks either need non-binary targets  $y$  (if the target classes exhibit a natural order), or binary *vectors*  $y \in \mathbb{R}^{n_L}$  which contain 1 only for the true class label and 0 otherwise. In addition, it is sometimes useful to move the last non-linear layer of the net, and regard it as part of the loss-function instead since it might benefit certain optimizer (Section 2.3).



### Activation Functions

Activation functions are the non-linear maps  $\sigma_l$  between two linear layers. Examples include the tangens hyperbolicus  $\sigma(z) = \tanh(z) \in (-1, 1)$ , or the sigmoidal function

$$\sigma(z) = \frac{1}{1 + \exp(-z)} = \frac{1}{2} + \frac{1}{2} \tanh\left(\frac{z}{2}\right) \in (0, 1). \quad (34)$$

Both of them saturate for large absolute values of  $z$ . This is beneficial on the one hand, since it can be ensured that the input to each linear layer ranges between certain digits. On the other hand, once saturated, it is hard for the optimizer to ‘move away’ from the saturated areas again, since gradients become very small, too. A somewhat compromise is the rectifier (ReLU-activation, for rectified linear unit) where  $\sigma(z) = z\mathbb{I}[z]$ , with  $\mathbb{I}[z]$  the indicator function.

An activation which is often used in the output layer for multi-class classification is the softmax-function  $\sigma_L^i(z) = \exp(-z_L^i) / \sum_{i=1}^n \exp(-z_L^i) \in (0, 1)$ . In contrast to the others, it couples the activations of the layer and can thus be interpreted as categorical distribution of the classes. Figure 15 shows the sigmoidal and the ReLU-activation function (—) as well as their derivatives  $\partial\sigma(z)/\partial z$  (---), which are needed for computing Eq. 33c.

#### 2.2.2 Overfitting, Regularization, and Generalization

Very flexible, expressive models tend to overfit, and in extreme cases only memorize the exact input-target mapping of the finite dataset they were trained on. They then lose the ability to generalize. This is usually circumvented by a variety of regularization techniques which involve model design, data selection and augmentation, but also the optimizer itself. There is an up- and a down-side to the latter: Entangling the optimizer formally and algorithmically with model design, can lead to less clear answers to the question, what exactly contributed to better or worse performance, and why. On the other hand, it enables exploration of the unknown error surfaces on the fly, and the optimizer can attenuate possible design flaws.

If the loss function can be interpreted as negative log-likelihood, optimizing for the weights can be seen as finding a maximum likelihood estimator which is known to potentially overfit for a (virtually always) finite amount of data [13] § 1.1. Since the topic of over-fitting will be picked up later again in Chapter 6, we will introduce some of the most prominent regularization techniques here, which can be roughly grouped into i) changes about the model, i.e., in  $f_w$  or  $\ell_f$ , ii) changes in the empirical distribution  $\hat{Q}$ , iii) changes in the optimization routine. A recent overview can be found in [48] § 7. The

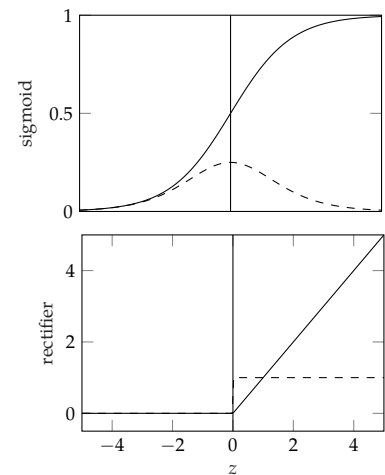


Figure 15: Sigmoidal and ReLU-activation function (top and bottom respectively). Function values (—) and gradients (---). The derivative of the sigmoidal is  $\sigma(z)' = \sigma(z)(1 - \sigma(z))$  and the one of ReLU is  $\sigma(z)' = \mathbb{I}[z]$ .

[13] Bishop, *Pattern Recognition and Machine Learning*, 2006

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

following subsections mostly focus on regularization in classification tasks, the major application area of Chapters II, III, and IV.<sup>5</sup>

### Changing the Model

Additive terms to the loss  $\ell$  change how weights<sup>6</sup> are penalized. A term proportional to the squared Euclidean norm of the weights  $\|w\|^2$  (also sometimes called ‘weight decay’ [78] and more sloppily ‘Tikhonov regularization’, or ‘ridge regression’) pulls weights towards zero, while the 1-norm  $\sum_i |w_i|$  also enforces sparsity (the latter is also called the LASSO-regularizer [134]). The idea behind the 2-norm is that large weights tend to overfit more because they represent more flexible function, while the 1-norm reduces complexity by reducing the number of effective parameters. Additionally, both can be interpreted as adding a prior probability on the weights: a zero-mean Gaussian for the Euclidean norm and Laplace for the 1-norm. Figure 16 shows the unit-circle of p-norms for  $p = 0.5, 1, 2$ , and 10 from ‘inner’ to ‘outer’. Unit-norm-vectors for  $p = 0.5, 1$  tend to lie close to an axis in comparison to  $p = 2, 10$ , thus sparsity is enforced.

Another way to regularize, is to once in a while drop (non-output) activations or weights by setting them to zero. The former is called ‘Dropout’ [63] [131] and the latter ‘DropConnect’ [139]. Effectively both procedures reduce the number of active connections, and hence the model complexity. At each iteration, a different random sub-model is updated with a random subset of the data, thus it can be seen as training all possible sub-models (weighted by the probability that they occur) as done in bagging, but all of them sharing the same weights.

A more involved change in the model is ‘batchNorm’ [70], which adds two additional learnable parameters per activation to the existing weights  $w$ , as to minimize the shift in input distribution that occurs when the weights of the preceding layer change. In contrast to a vanilla neural network, these additional batchNorm-transformations tie together the elements in a mini-batch. BatchNorm was initially designed to help avoid saturated gradients, but it also does seem to have a regularizing effect. Although less well understood, a possible explanation is that, similar to Dropout, more activations lie close to zero, which lead to less extreme values in the weights  $w$ .

### Changing the Dataset

Data augmentation techniques approach the problem from the opposite point of view: If there is too little data to match the model-complexity, increase it. This is only possible if something is *known* about the gaps that need to be filled in the data-distribution, for example if there are transformations of the input that should leave the output invariant. Prominent examples for image-inputs are rotations,

<sup>5</sup> Another issue which will not be featured here is under-fitting, when greedy optimizers converge to a sub-optimal local minimum or get stuck on a stationary saddle point. Currently it is believed that saddle points are the bigger practical concern for very deep networks, and that local minima with a high loss rather occur in shallow architecture. For a discussion see e.g. [48] § 8.2 or [26].

[26] Dauphin et al., “Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization,” 2014

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

<sup>6</sup> We will use the term ‘weights’ also to denote all parameters  $w$ , and not only the elements of the weight matrices  $W_l$ . The distinction is not essential for the made arguments.

[78] Krogh and Hertz, “A simple weight decay can improve generalization,” 1991

[134] Tibshirani, “Regression shrinkage and selection via the lasso,” 1996

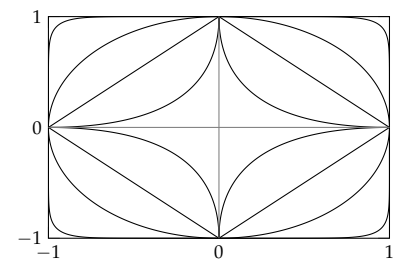


Figure 16: Unit-circle of the p-norm for a two-dimensional vector. From inner to outer:  $p = 0.5, 1, 2$ , and 10.

[63] Hinton et al., “Improving neural networks by preventing co-adaptation of feature detectors,” 2012

[131] Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” 2014

[139] Wan et al., “Regularization of Neural Networks using DropConnect,” 2013

[70] Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 2015

translations, stretching or scaling of the inputs as to represent more possible variations of the same example.

An opposite approach is to reduce the differences between individual datapoints (not classes), for example by injecting noise of the same scale that ‘blurs out’ irregularities, applied to the inputs but also the hidden units or the weights [124] [137]. A structured approach in case of RGB-images, is contrast and colorspace augmentation which mimics different external conditions, such as light exposure, for pictures of the same content [77]. All of the above (besides choosing the dataset itself) bias toward the artificially created data, but the effect might be affordable if a better generalization can be achieved thereby.

A third approach is importance sampling, or weighing of datapoints. This locally changes the empirical distribution of the data, or weighs individual datapoints non-uniformly [145] [23]. The assumption here is that the collected dataset does not represent the data-distribution well enough, or that certain classes or examples are harder to learn than others and thus need to be shown more often to the optimization machine. Though not originally motivated as a regularizer it helps to learn easier and harder input-target relations at the same speed, which then yields better generalization. In contrast to a pre-processing step, this approach is formally and algorithmically dependent on the optimizer.

### *Adapting the Optimizer*

The arguably most straightforward means of regularization, as part of the optimization procedure, is to reduce its resolution by injecting noise into the search direction. The (possibly unsatisfying) pictorial idea is that certain indents, local minima, or even the precise location of a minimum are artifacts of a finite dataset. If these are, roughly speaking, on a smaller scale in  $w$  and  $\ell$ -space than the structure of the true risk, then softening the resolution of the optimizer will lead to better generalization. This is usually done by *mini-batching* since then, gradients are still unbiased estimators of the true gradients.<sup>7</sup>

More recent works include optimizers that encode the goodness of an optimum in terms of probable generalization performance and bias their update towards these regions [24] [87]. A measure for this in [87] is the entropy of the distribution of weights at convergence.

A last, but not incompatible, approach is to simply halt the optimization process when it starts to overfit. This is also called *early-stopping*, [94] [111] [107]. The point of stopping is usually found by withholding some data (a *validation set*) from the training procedure and monitoring generalization performance on it. This estimator is unbiased but only precise if the validation set is large enough. In addition, it reduces the effective size of the training data that defines the empirical risk

[124] Sietsma and Dow, “Creating artificial neural networks that generalize,” 1991

[137] Vincent et al., “Extracting and composing robust features with denoising autoencoders,” 2008

[77] Krizhevsky, Sutskever, and Hinton, “Imagenet classification with deep convolutional neural networks,” 2012

[145] Zhao and Zhang, “Stochastic Optimization with Importance Sampling,” 2014

[23] Chang, Learned-Miller, and McCallum, “Active Bias: Training More Accurate Neural Networks by Emphasizing High Variance Samples,” 2017

[24] Chaudhari et al., “Entropy-SGD: Biasing Gradient Descent Into Wide Valleys,” 2016

[87] Maclaurin, Duvenaud, and Adams, “Early Stopping is Nonparametric Variational Inference,” 2015

[94] Morgan and Bourlard, “Generalization and parameter estimation in feed-forward nets: Some experiments,” 1989

[111] Reed, “Pruning algorithms-a survey,” 1993

[107] Prechelt, “Early Stopping — But When?” 2012

<sup>7</sup> There are further benefits of mini-batching: Stochastic gradients might overcome flat plateau-like areas faster than deterministic ones, but more importantly it also allows for trading of precision of calculation with its cost.

and hence that the optimizer can access. For large enough datasets, though, it is arguably the gold standard and a reliable method to induce early-stopping (further discussion in Chapter 6).

### 2.3 Iterative Optimization Routines

The empirical risk minimization problem

$$w^* = \arg \min_w L_{\mathcal{D}}(w) \quad (35)$$

( $L_{\mathcal{D}}(w)$  defined in Eq. 28) is usually solved by *greedy, iterative* optimization routines, all of which will only find *local* minimizers of  $L_{\mathcal{D}}$  (we will assume that this is satisfactory). There are many good textbooks for an overview on (non-)convex optimization e. g., [100] [16], some specifically target neural network optimization [81] [48, § 8].

Most iterative optimizers loop over the following subroutines: i) They initialize  $w$  randomly as a current best guess for  $w^*$ . ii) They approximate  $L_{\mathcal{D}}(w)$  around  $w_t$ , usually with a first- or second-order model. iii) They define a direction of descent  $p_t \in \mathbb{R}^N$ , also called *search direction*, based on this model. iv) They set a scalar step size  $\alpha_t \in \mathbb{R}_+$  (also called *learning rate* in the neural network community) *conditioned* on this direction, and, finally, v) They make a step into the scaled descent direction:

$$w_{t+1} = w_t + \alpha_t p_t. \quad (36)$$

A pseudocode is shown in Algorithm 1. The following subsections

```

1: function ITERATIVEOPTIMIZER_SKETCH( $L_{\mathcal{D}}(\cdot), w_0$ )
2:    $w_t \leftarrow w_0$  // initial guess for weights
3:   while budget not used do
4:     [ $L_{\mathcal{D},t}, \nabla L_{\mathcal{D},t}$ ]  $\leftarrow L_{\mathcal{D}}(w_t)$  // evaluate objective
5:      $p_t \leftarrow \text{COMPUTE\_SEARCH\_DIRECTION}(\nabla L_{\mathcal{D},t}, \dots)$ 
6:      $\alpha_t \leftarrow \text{COMPUTE\_STEP\_SIZE}(L_{\mathcal{D}}, \dots)$ 
7:      $w_t \leftarrow w_t + \alpha_t p_t$  // update best guess
8:   end while
9:   return  $w_t$ 
10: end function

```

discuss instances of Algorithm 1, where it is assumed that  $L_{\mathcal{D}}(w_t)$  and  $\nabla L_{\mathcal{D}}(w_t)$  can be computed exactly; this is usually called *deterministic* optimization. Section 2.4 gives an overview over *stochastic* optimization, where gradient evaluations are inexact, since they are computed on mini-batches only. Before we proceed, we motivate common concepts in optimization, and introduce some terms and notation that become relevant in the later chapters.

[100] Nocedal and Wright, *Numerical Optimization*, 1999

[16] Boyd and Vandenberghe, *Convex Optimization*, 2004

[81] LeCun et al., "Efficient BackProp," 1998

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

Algorithm 1: Sketch of iterative optimizer. The subroutines that compute the search direction  $p_t \in \mathbb{R}^N$  and the step size  $\alpha_t \in \mathbb{R}_+$  might take additional arguments e. g., a history of evaluated gradients, the previous search direction, or the current position  $w_t$ . COMPUTE\_STEP\_SIZE could for example be a line search subroutine (§ 2.5).

### Lipschitz Continuity

The gradient function  $\nabla L_{\mathcal{D}}(w)$  is called *Lipschitz continuous* if there exists a constant  $\mathcal{L} > 0$  such that  $\|\nabla L_{\mathcal{D}}(w) - \nabla L_{\mathcal{D}}(w')\| \leq \mathcal{L}\|w - w'\|$  for all  $w$  and  $w'$ . If this holds, the function  $L_{\mathcal{D}}(w)$  is also called  $\mathcal{L}$ -smooth. In practice, not all loss-functions  $L_{\mathcal{D}}$  are Lipschitz continuous or  $\mathcal{L}$ -smooth, but the assumption that  $L_{\mathcal{D}}(w)$  does not change ‘all too much’ from one iteration to the next is inherent to most practical implementations of iterative optimizers. This manifests for, example, when learning rates, Hessian estimates, or summary statistics of gradient-magnitudes are propagated from one iteration to the next. This is only viable if the function, for all practical purposes, does not change unexpectedly from one step to another; and a way of approximately capturing this sloppy notion in mathematical terms, and without being too restrictive, is Lipschitz continuity on gradient, the loss, or the Hessian function.

### Global Convergence

An optimizer of the form of Eq. 36 is *globally convergent* if

$$\lim_{t \rightarrow \infty} \|\nabla L_{\mathcal{D}}(w_t)\| = 0 \quad (37)$$

for arbitrary  $w_0$ , meaning that the gradient vanishes if the optimizer runs ‘long enough’.<sup>8</sup> Assume that  $p_t$  is a descent direction, and that the angle  $\theta_t$  between  $p_t$  and  $\nabla L_{\mathcal{D}}(w_t)$  is bounded away from  $\pm 90^\circ$ , i. e., there exists a  $\delta_\theta$  such that  $\cos \theta_t \geq \delta_\theta > 0$  for all iterations  $t$ . For functions  $L_{\mathcal{D}}(w)$  which are bounded below,  $\mathcal{L}$ -smooth, and for iterates  $L_{\mathcal{D}}(w_t)$  and  $p_t^\top \nabla L_{\mathcal{D}}(w_t)$  that fulfill the weak Wolfe conditions (decay constraints on  $L_{\mathcal{D}}(w_t)$  and  $p_t^\top \nabla L_{\mathcal{D}}(w_t)$ , definitions in Section 2.5 Eq. 68 below), it is

$$\sum_{t=0}^{\infty} \cos^2 \theta_t \|\nabla L_{\mathcal{D}}(w_t)\|^2 < \infty. \quad (38)$$

This is the Zoutendijk theorem and Eq. 38 is often referred to as *Zoutendijk’s condition* (e. g., [100, §3.2]). It implies that  $\lim_{t \rightarrow \infty} \|\nabla L_{\mathcal{D}}(w_t)\| = 0$ , and thus global convergence. In practice, imposing the weak Wolfe conditions is fairly easy, since they can simply be checked at runtime and controlled by  $\alpha_t$ , but finding search directions  $p_t$  such that  $\cos \theta_t$  is bounded away from zero is often harder.

### Convergence Rates

Optimizers that are globally convergent might still have very slow convergence *rates*; or a fast convergent optimizers might not necessarily be globally convergent. The latter is intuitive, since some search

<sup>8</sup> To avoid confusion, the word ‘global’ only implies that the optimizer converges to a stationary point from any globally initialized point, and not that it converges to a global minimum.

[100] Nocedal and Wright, *Numerical Optimization*, 1999

directions, e. g., the Newton direction, might be nearly orthogonal to the gradient, but, if initialized well, still convergence rapidly.

Suppose the sequence  $x_t$  converges to  $x^*$ , then the *rate of Q-convergence* is defined as:

$$\frac{\|x_{t+1} - x^*\|}{\|x_t - x^*\|} \leq r_t, \quad r_t \rightarrow 1, \quad \text{sub-linear} \quad (39a)$$

$$\frac{\|x_{t+1} - x^*\|}{\|x_t - x^*\|} \leq r, \quad r \in (0, 1), \quad \text{linear} \quad (39b)$$

$$\frac{\|x_{t+1} - x^*\|}{\|x_t - x^*\|} \leq r_t, \quad r_t \rightarrow 0, \quad \text{super-linear} \quad (39c)$$

$$\frac{\|x_{t+1} - x^*\|}{\|x_t - x^*\|^2} \leq r, \quad 0 < r < \infty, \quad \text{quadratic} \quad (39d)$$

each for  $t$  sufficiently large. From top to bottom, the sequence  $x_t$  converges faster to  $x^*$ .<sup>9</sup> The reduction factors  $r$  and  $r_t$  define the local or global slope of reduction. They depend on the choice of the optimizer, in particular  $p_t$  and  $\alpha_t$ , as well as on the specific problem  $L_{\mathcal{D}}$ . An optimizer with linear rate but with a less preferable slope  $r$  close to one, will reduce the distance to  $x^*$  very little even for many steps (e. g., if  $r = 0.999$ , after 100 steps the total reduction will be just  $0.999^{100} \approx 0.9$ ). In practice, the performance of an optimizer will not be measured in the relative reduction of the loss per step, but the *total* reduction after a computational budget is used up. This means that a good optimizer need to trade off rates, by computing more involved  $p_t$  and  $\alpha_t$ , with the cost in doing so.

### Scale Invariance

Scale-invariant updates  $p_t$  possess the *unit* of the weight space, such that they re-scale exactly right if the loss or the weights are re-parametrized. This is especially helpful since parameters of the optimizer (such as the step size  $\alpha_t$ ) do not have to be re-learned or re-tuned when applied to a different problem. One might even say that tuned parameters in some way *encode* the scales unique to one optimization problem in case the update is not scale-invariant. Consider for example re-scaling the loss by a scalar  $c > 0$ , such that  $L_{\mathcal{D}}^{\text{scaled}}(w) := cL_{\mathcal{D}}(w)$ . In this example, the update  $p_t$  should ideally not change, since the parameterization of the weights did not change. Newton's method, which is scale-invariant, produces  $p_t^{\text{scaled}} = -(c\Delta L_{\mathcal{D}})^{-1}(c\nabla L_{\mathcal{D}}) = -\Delta L_{\mathcal{D}}^{-1}\nabla L_{\mathcal{D}} = p_t$ , which is exactly what is required; the gradient decent update, on the other hand, is *not* scale-invariant and produces  $p_t^{\text{scaled}} = -c\nabla L_{\mathcal{D}} = cp_t$ . Thus the learning rate  $\alpha_t$  would have to be changed accordingly, in this example exactly by  $\alpha_t^{\text{scaled}} := c^{-1}\alpha_t$ , to retain the same optimizer.<sup>10</sup> Non-scale-invariant optimizers are often used in practice, since learning the right scale is more difficult than just learning a direction. In order to mini-

<sup>9</sup> Examples for  $\gamma \in (0, 1)$  are:

$(t+1)^{-1}$	sub-linear
$\gamma^t$	linear
$\gamma^{t^2}$	super-linear
$\gamma^{2^t}$	quadratic.

<sup>10</sup> Scale invariance can be checked by computing the unit of  $p_t$ . Let  $[w]$  denote the unit of the weights and  $[L]$  the unit of the loss, then Newton's update has the units  $([L]/[w]^2)^{-1}[L]/[w] = [w]$ . The update of GD has units  $[L]/[w] \neq [w]$ . The parameters of the optimizer have to capture this scale, e. g.,  $\alpha_t^{\text{GD}}$  will need to encode the *unit*  $[w]^2/[L]$  which can be thought of as a (scalar) inverse Hessian or an inverse Lipschitz constant for  $\mathcal{L}$ -smooth functions.



mize re-tuning cost, usually care is taken not to confront the optimizer with all too different optimization problems. For example neural networks usually consist of similar puzzle pieces, and one might argue that these also yield losses of similar scales; these puzzle pieces might be similar choices of loss functions, normalizing the data  $X$ , similar choices of activations, batchNorm, and so on, such that re-tuning e. g., of learning rates is milder. In general though, if used in different contexts, non-scale-invariant updates can require re-tuning of parameters of arbitrary and unknown scale.

### 2.3.1 Gradient Descent

The arguably most basic algorithm that follows Eq. 36 is *gradient descent*<sup>11</sup> (GD) where the search direction  $p_t = -\nabla L_{\mathcal{D}}(w_t)$  locally points into the direction of the steepest path downhill in Euclidean norm, meaning that for  $\delta \in \mathbb{R}^N$ ,  $\epsilon > 0$ :

$$\lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{\delta: \|\delta\| \leq \epsilon} L_{\mathcal{D}}(w_t + \delta) = \frac{-\nabla L_{\mathcal{D}}(w_t)}{\|\nabla L_{\mathcal{D}}(w_t)\|} =: \frac{p_t}{\|p_t\|}, \quad (40)$$

which yields the update:

$$w_{t+1} = w_t - \alpha_t \nabla L_{\mathcal{D}}(w_t). \quad (41)$$

The underlying model can be thought of as a first-order Taylor expansion  $L_{\mathcal{D}}(w) \approx L_{\mathcal{D}}(w_t) + (w - w_t)^{\top} \nabla L_{\mathcal{D}}(w_t)$  around the current location  $w_t$ , i. e., the tangent plane to  $L_{\mathcal{D}}(w)$  at  $w_t$ . This also means that the local approximation is unbounded below, and does not provide a meaningful estimate for the step size  $\alpha_t$ .<sup>12</sup> The assumptions on  $L_{\mathcal{D}}(w)$  and  $\alpha_t$  which lead to Zoutendijk's condition are typically met (apart from  $\mathcal{L}$ -smoothness), since  $L_{\mathcal{D}}(w)$  is bounded below by definition and  $\cos \theta_k = 1$  can be bounded below by any fixed number  $\delta_{\theta} \in (0, 1)$ . Thus GD is globally convergent. It does converge rather slow, especially in cases where the condition numbers of the local Hessian is large: For example for a quadratic function with positive definite Hessian and conditions number  $\kappa$  and *exact* line searches (meaning that  $\alpha_t$  minimizes the 1D-function  $L_{\mathcal{D}}(w_t + \alpha_t p_t)$ ), GD is linearly convergent in the loss with rate  $r = (\kappa-1)^2/(\kappa+1)^2$ . For large condition numbers  $\kappa \gg 1$ ,  $r$  is close to one, thus the reduction is very little.<sup>13</sup> This is illustrated in Figure 17; Top: positive definite Hessian with low condition number  $\kappa = 4$  ( $\dashrightarrow$ ,  $r \approx 0.36$ ) and middle: positive definite Hessian with a larger condition number  $\kappa = 20$  ( $\dashrightarrow$ ,  $r \approx 0.82$ ). Both optimization runs start at the same distance from the minimizer  $w^* = [0, 0]^{\top}$  at  $w_0 = [-0.1, -0.98]^{\top}$ , and line searches were exact at each iteration (closed form exists for quadratic functions). The zig-zagging of GD is clearly visible, especially for the ill-conditioned problem; also the em-

<sup>11</sup> It is likely that gradient descent was invented multiple times. A very clearly documented instance of this, however, is by Augustin-Louis Cauchy in his 1847 paper "Méthode générale pour la résolution des systèmes d'équations simultanées" [21]. He explicitly states the importance of choosing the step size: "It is easy to conclude from it that the value  $\Theta$  of  $u$  [ $u$  is objective,  $\Theta$  is updated value], determined [...] will become inferior to  $u$ , if  $\theta$  [step size] is sufficiently small.". He also emphasizes the iterative nature of his algorithm: "If the new value of  $u$  is not a minimum, one will be able to deduce, by operating always in the same manner, a third still smaller value; and, by continuing thus, one will find successively some values of  $u$  more and more small, which will converge toward a minimum value of  $u$ ". (Translation by Richard J. Pulskamp.)

[21] Cauchy, "Méthode générale pour la résolution des systèmes d'équations simultanées," 1847

<sup>12</sup> In principle  $\alpha_t$  could be *any* positive number, since the search direction  $p_t = -\nabla L_{\mathcal{D}}$  can be arbitrarily scaled by multiplying  $L_{\mathcal{D}}$  with a positive constant. As mentioned above, this is known as *non-scale-invariance* of the gradient descent search direction.

<sup>13</sup> For isotropic quadratic problems though ( $\kappa = 1$ ),  $r$  becomes zero and GD converges in only one step.

irical relative reduction of the loss  $L_{\mathcal{D}}$  (bottom plot) is close to the theoretical upper bound  $r$ . For *inexact* line searches, the rates are not expected to improve, but they also do not necessarily worsen e. g., with minor restrictions on the step size  $\alpha_t$ , such as upper bounding it with the Lipschitz constant of the gradient  $\alpha_t \leq \mathcal{L}^{-1}$ . Nevertheless, GD is computationally inexpensive such that many iterations can be performed; it is quite robust meaning that it is applicable to many problems as well as numerically stable, and also incredibly easy to implement (all you need is the gradient). These characteristics make GD still one of the most widely used iterative optimization routines, since more than 150 years.

### Momentum Methods

Momentum methods are inspired by physical systems: the motion of a massive particle in a potential that experiences friction. Due to the inertia of the particle, the solution to the dynamics equations is smoother, and, even when discretized, less saw-toothed than a gradient descent path. For instance, for a differential equation of the form  $m\dot{w} = -\dot{w}\zeta - \nabla L_{\mathcal{D}}(w)$  and a linear approximation of  $L_{\mathcal{D}}(w)$  around  $w_0$  (just like in GD), the solution for the *velocity*  $v(\tau)$  is:

$$v(\tau) := \dot{w}(\tau) = -\alpha \nabla L_{\mathcal{D}}(w(\tau_0)) + \gamma v(\tau_0) \quad (42)$$

where the scalars  $\alpha$  and  $\gamma$  depend on the mass  $m \geq 0$ , the friction  $\zeta \geq 0$ , and the time interval  $\tau - \tau_0$ .<sup>14</sup> In optimization, the learning rate  $\alpha_t$  and the *momentum parameter*  $\gamma_t \in (0, 1)$  do not have an obvious interpretation and are set heuristically, usually  $\gamma_t = 0.9$ . Eq. 42, in fact, leads to the famous GD+MOMENTUM algorithm [105] [116]:

$$\begin{aligned} v_t &= -\alpha_t \nabla L_{\mathcal{D}}(w_t) + \gamma_t v_{t-1} \\ w_{t+1} &= w_t + v_t. \end{aligned} \quad (43)$$

The velocity term has an intuitive behavior: If similar gradients are seen in succession,  $v$  increases and larger steps are taken; if gradients rather point in different directions,  $v$  decreases which yields more cautious steps. Pictorially, this allows the optimizer to traverse ravines more smoothly as well as to speed up on plateaus. The approximation of the potential in the differential equation is linear; although their flexible behavior suggests otherwise, many momentum algorithms are thus first-order methods. Another momentum method is NESTEROV accelerated gradients [97], where  $v_t = \alpha_t \nabla L_{\mathcal{D}}(w_t + \alpha_t v_t) + \gamma_t v_{t-1}$ , which encodes a ‘look-ahead’ by evaluating  $\nabla L_{\mathcal{D}}$  not at current position  $w_t$  but at the position which would be reached if the search direction was not altered. GD+MOMENTUM is illustrated in Figure 18 together with the unaltered GD-direction and

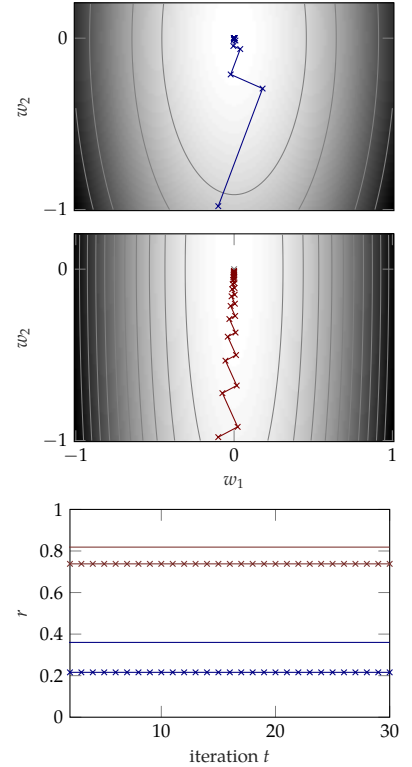


Figure 17: Gradient descent on 2D-quadratic functions with different condition numbers  $\kappa$ . Top:  $\kappa = 4$ , middle:  $\kappa = 20$ . Path of optimizers ( $\text{---}\ast\text{---}$  /  $\text{---}\ast\text{---}$ ). Bottom: theoretical upper bound on slopes  $r \approx 0.36, 0.82$  ( $\text{---}$  /  $\text{---}$ ,  $\kappa = 4, 20$  respectively), and the heuristic values in the same colors as the paths above.

<sup>14</sup> The constants are  $\gamma = e^{-\frac{\zeta}{m}(\tau-\tau_0)} \in (0, 1)$  and  $\alpha = \frac{1}{\zeta} \left(1 - e^{-\frac{\zeta}{m}(\tau-\tau_0)}\right) > 0$ . The time  $\tau$  has arbitrary scale in the optimization setting.

[105] Polyak, “Some methods of speeding up the convergence of iteration methods,” 1964

[116] Rumelhart, Hinton, and Williams, “Learning representations by back-propagating errors,” 1986

[97] Nesterov, “A method of solving a convex programming problem with convergence rate  $O(1/\text{sqr}(k))$ ,” 1983



the `BFGS`-optimizer (§ 2.3.4) on the 2D Rosenbrock polynomial. It is clearly visible that `GD+MOMENTUM` exhibits a smoother optimization path than `GD`.

### 2.3.2 Second-Order Methods

Second-order methods model the curvature or *Hessian*  $\Delta L_{\mathcal{D}}(w_t)$  of the loss, or a *related quantity*. The local approximation around  $w_t$  is quadratic:

$$L_{\mathcal{D}}(w) \approx L_{\mathcal{D}}(w_t) + (w - w_t)^\top \nabla L_{\mathcal{D}}(w_t) + \frac{1}{2}(w - w_t)^\top B_t (w - w_t), \quad (44)$$

where  $B_t \in \mathbb{R}^{N \times N}$  is the second derivative of the quadratic approximation, i. e.,  $B_t = \Delta L_{\mathcal{D}}(w_t)$  would yield a second-order Taylor expansion. Often, but not necessarily, the corresponding search directions  $p_t$  are scale-invariant, e. g., the Newton direction, such that they also provide a natural length-scale for each step, defined by the distance to the optimum of the local approximation. If this is not the case, they usually capture some kind of *relative* scaling of the input-axis which have been proven to be very helpful in practice.

The search direction is defined as the vector to the unique root of the gradient of the right hand side of Eq. 44:  $\nabla L_{\mathcal{D}}(w_t) + B_t(w - w_t) \stackrel{!}{=} 0$ , therefore  $p_t = -B_t^{-1} \nabla L_{\mathcal{D}}(w_t)$ ,<sup>15</sup> or, for positive definite  $B_t$ , alternatively through the weighted norm minimization:

$$\lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{\delta: \|\delta\|_{B_t^{1/2}} \leq \epsilon} L_{\mathcal{D}}(w_t + \delta) = \frac{-B_t^{-1} \nabla L_{\mathcal{D}}(w_t)}{\|B_t^{-1} \nabla L_{\mathcal{D}}(w_t)\|_{B_t^{1/2}}} := \frac{p_t}{\|p_t\|_{B_t^{1/2}}}, \quad (45)$$

for  $\delta \in \mathbb{R}^N$ ,  $\epsilon > 0$ , and  $\|w\|_{A^{1/2}}^2 := w^\top A w = \|A^{1/2} w\|^2$  with  $A \in \mathbb{R}^{N \times N}$  positive definite. Thus the update of Eq. 36 is:

$$w_{t+1} = w_t - \alpha_t B_t^{-1} \nabla L_{\mathcal{D}}(w_t). \quad (46)$$

This defines a descent direction only if  $B_t$  is positive definite, since then,  $-\nabla L_{\mathcal{D}}(w_t)^\top p_t = \nabla L_{\mathcal{D}}(w_t)^\top B_t^{-1} \nabla L_{\mathcal{D}}(w_t) > 0$  (for  $\nabla L_{\mathcal{D}} \neq 0$ ). There are optimizers which enforce this property when designing  $B_t$  (e. g., `BFGS`, `DFFP`, § 2.3.4), but for example Newton's method does not fulfill this property for non-convex functions.

The cost of computing  $p_t$  depends on the cost of inverting  $B_t$ . This can range from cubic cost in  $N$  for methods that invert a dense  $B_t$ , to methods that are quadratic or only linear in cost. The latter two usually exploit low-rank structures of successive updates to  $B_t$ . Still, the additional cost for computing  $p_t$ , in comparison to a simple update like `GD`, has to be outweighed by faster descent per step in order to yield a superior optimizer.

<sup>15</sup> Unless noted otherwise, it will always be assumed that  $B_t$  is invertible.

An additional benefit of second-order methods, is that they often expose less to none manual tuning parameters. This is a big advantage in the context of automation and user-friendliness, over algorithms that do expose free parameters. Three direct consequences are: i) Increased applicability, especially in build-in black-box subroutines, ii) decreased overall training time, since additional runs for parameter search are not needed, and iii) usually better performance for restricted budgets. One type of arguably parameter free second-order methods are quasi-Newton optimizers (§ 2.3.4) in combination with a line search (§ 2.5).

### Newton's Method

Newton's method<sup>16</sup> is a true classic and follows from Eq. 44 if the right hand side is the second-order Taylor expansion of  $L_{\mathcal{D}}(w)$  around  $w_t$ , i. e.,  $B_t = \Delta L_{\mathcal{D}}(w_t)$ . Then:

$$w_{t+1} = w_t - \alpha_t \Delta L_{\mathcal{D}}(w_t)^{-1} \nabla L_{\mathcal{D}}(w_t). \quad (47)$$

This only defines a descent direction if  $\Delta L_{\mathcal{D}}(w_t)$  is positive definite, which holds for convex functions or in a neighborhood around a true minimum. Thus Newton's methods only converges (with  $\alpha_t = 1$  and for a Lipschitz-continuous Hessian functions) if started 'close enough' to a minimizer. Then, the rate is quadratic for the iterates  $w_t$  as well as the gradient norms  $\|\nabla L_{\mathcal{D}}(w_t)\|$ . In practice, sometimes a mild version of a line search, e. g., a backtracking line search that always tests  $\alpha_t = 1$  first, is used for robustness. Newton's method is computationally quite expensive (cubic in  $N$ ), since the Hessian  $\Delta L_{\mathcal{D}}(w_t)$  has to be computed as well as inverted at every iteration.

### Generalized Gauss-Newton

The *generalized Gauss-Newton matrix* at point  $w$  is defined as:

$$G(w) := \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} J_f(w) \Delta_f \ell(f) J_f(w)^{\top}, \quad (48)$$

where the operators  $\nabla_f$  and  $\Delta_f$  are the first and second partial derivative with respect to  $f$ , and  $J_f(w) \in \mathbb{R}^{N \times n_L}$  is the Jacobian of  $f(w) \in \mathbb{R}^{n_L}$  with respect to  $w$ . The loss  $\ell(f)$  is written in such a way to emphasize the dependence on  $f$ . The matrix  $G(w)$  is an approximation to the Hessian  $\Delta L_{\mathcal{D}}(w)$  of the loss. To see this write:

<sup>16</sup> The Newton-Raphson method also denotes a more generic root-finding algorithm. In optimization this refers to finding the root of the gradient of a function, thus Newton's method for optimization, described here, is a specific instance of the more general Newton-Raphson algorithm.

The Newton-Raphson method is named after Sir Isaac Newton and Joseph Raphson. Like gradient descent, it was most likely invented multiple times (traces go back to the 12<sup>th</sup> century, an Iranian algebraist called Sharaf al-Din al-Tusi, who applied it to cubic equations). Newton applies his method successfully to the non-polynomial problem  $x - \epsilon \sin(x) = M$  (Kepler's equation) in the 2<sup>nd</sup> and 3<sup>rd</sup> edition of his *Philosophiae naturalis principia mathematica* [98] (around 1713 and 1726, 1<sup>st</sup> appeared in 1687) which states: "But since the description of this curve is difficult, a solution by approximation will be preferable. [...] diminished by the cosine of the angle  $[x_i]$  [...] And so we may proceed in infinitum. [...] But since the series [...] converges so very fast that it will be scarcely ever needful to proceed beyond second term [iteration]." (translated from Latin by Andrew Motte).

[98] Newton, *Philosophiae naturalis principia mathematica*, 1726

$$\begin{aligned}
\Delta L_{\mathcal{D}}(w) &= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \Delta \ell(f_w(x), y) \\
&= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \left[ \sum_{k=1}^{n_L} \nabla_f \ell_k(f) \Delta f_k(w) + \sum_{k,l=1}^{n_L} \Delta_f \ell_{kl}(f) \nabla f_k(w) \nabla f_l(w)^\top \right] \\
&= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \left[ \sum_{k=1}^{n_L} \nabla_f \ell_k(f) \Delta f_k(w) + J_f(w) \Delta_f \ell(f) J_f(w)^\top \right],
\end{aligned} \tag{49}$$

where, from the second row on, we dropped the  $y$ s and we used  $f(w)$  shorthand for  $f_w(x)$  to highlight the dependence on  $w$ .

The last row of Eq. 49 is equivalent to Eq. 48 for either a root of  $\nabla_f \ell_k(f)$ , or a vanishing Hessian  $\Delta f_k(w)$ . This means, for instance, that the generalized Gauss-Newton matrix is identical or very similar to the Hessian at minimizers of the loss where  $\nabla_f \ell$  is supposedly small, too. The geometric interpretation is that the Gauss-Newton matrix only models the Hessian of the most outer of the nested functions: the loss  $\ell(f)$ , but not the Hessian of  $f(w)$ . Thus it is a lightweight Newton step, which only captures parts of a full second-order approximation. The benefit of the Gauss-Newton matrix is that it is always positive definite if the loss  $\ell(f)$ , as a function of  $f$ , is convex (convexity of  $f(w)$  is not needed; can be seen from Eq. 48 for pos. def.  $\Delta_f \ell$ ). It thus always yields descent directions  $p_t = -G_t^{-1} \nabla L_{\mathcal{D}}$ . There are also connections to the Fisher information matrix discussed below. When the Gauss-Newton direction is used, sometimes  $f_w$  and  $\ell_f$  are re-defined, and the last non-linearity of  $f_w$  is assigned to  $\ell$ , such that more of the curvature can be captured in the update. For  $n_L < N$  the pseudo-inverse of one of the summands is  $(J_f^\top J_f)^{-1} J_f$  which can be computed at cost  $\mathcal{O}(n_L^3 + N n_L)$ .

### 2.3.3 Natural Gradient

The natural gradient direction [2] ([92] for overview) is based on the idea to measure progress not in the loss-space, but in the *distribution* space of the learned conditional distribution  $P_w(y|x)$ , parametrized by  $w$ , between input- and target-space.<sup>17</sup> A measure for the difference of two distributions  $P$  and  $Q$  is the Kullback-Leibler divergence  $\text{KL}(P\|Q)$ .<sup>18</sup> Now suppose we move in weight space by  $\delta \in \mathbb{R}^N$ , analogously to above, then the difference between the old and new distribution is given by:

$$\begin{aligned}
\text{KL}(P_{w+\delta}(x,y) \| P_w(x,y)) &= \int p_{w+\delta}(x,y) \log \frac{p_{w+\delta}(x,y)}{p_w(x,y)} dx dy \\
&= \mathbf{E}_{Q(x)} [\text{KL}(P_{w+\delta}(y|x) \| P_w(y|x))].
\end{aligned} \tag{50}$$

<sup>18</sup> The KL-divergence is not a distance, since it is not generally symmetric:  $\text{KL}(P\|Q) \neq \text{KL}(Q\|P)$ ; but it is always positive and only zero iff  $Q = P$ . For densities  $p(x)$  and  $q(x)$  it is defined as  $\text{KL}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)} dx$  and also known as the relative entropy. Pictorially it measures how well  $p(x)$  is approximated by  $q(x)$  weighted by  $p(x)$ , and then averaged over all weighted differences. This also explains why KL is not symmetric, since discrepancies between  $p$  and  $q$  might matter more or less depending if  $p$  or  $q$  has nontrivial mass there.

[2] Amari, "Natural Gradient Works Efficiently in Learning," 1998

[92] Martens, "New insights and perspectives on the natural gradient method," 2014

<sup>17</sup> This interpretation only holds if the loss  $L_{\mathcal{D}}$  can be interpreted as negative log likelihood as described in 2.2.1.

where  $Q(x, y)$  is the unknown joint data distribution with density  $q(x, y)$ ,  $Q(x)$  the marginal distribution, and  $P_w(x, y)$  has density  $p_w(x, y)$ .<sup>19</sup> Since the KL-divergence is non-negative, the steepest descent direction in distribution space can be defined as:

$$\lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \arg \min_{\delta: \text{KL}(P_{w+\delta}(x, y) \| P_w(x, y)) \leq \epsilon^2} L_{\mathcal{D}}(w_t + \delta) = \frac{-\mathcal{F}_t^{-1} \nabla L_{\mathcal{D}}(w_t)}{\left\| \mathcal{F}_t^{-1} \nabla L_{\mathcal{D}}(w_t) \right\|_{\mathcal{F}_t^{-1/2}}} =: \frac{p_t}{\|p_t\|_{\mathcal{F}_t^{-1/2}}}. \quad (51)$$

The limit of Eq. 51 includes a weighted norm with positive definite matrix  $\mathcal{F}_t$  defined as:

$$\begin{aligned} \mathcal{F}_t &= -\mathbf{E}_{P_{w_t}(x, y)} [\Delta \log p_{w_t}(x, y)] = -\mathbf{E}_{Q(x)} \left[ \mathbf{E}_{P_{w_t}(y|x)} [\Delta \log p_{w_t}(y|x)] \right] \\ &= \mathbf{E}_{P_{w_t}(x, y)} [\nabla \log p_{w_t}(x, y) \nabla \log p_{w_t}(x, y)^\top] = \mathbf{E}_{Q(x)} \left[ \mathbf{E}_{P_{w_t}(y|x)} [\nabla \log p_{w_t}(y|x) \nabla \log p_{w_t}(y|x)^\top] \right]. \end{aligned} \quad (52)$$

$\mathcal{F}_t$  is known as the *Fisher information matrix*, and (for  $\ell(f_w(x), y) = -\log p_w(y|x)$ ) is the expected Hessian of  $\ell$  under  $P_w(x, y)$  (not  $Q$ ). An intuition for this result can be obtained by developing Eq. 50 around  $\delta = 0_N$ , which by some calculus yields:

$$\text{KL}(P_{w_t+\delta}(x, y) \| P_{w_t}(x, y)) = \frac{1}{2} \delta^\top \mathcal{F}_t \delta + \mathcal{O}(\delta^3). \quad (53)$$

This is an explicit approximation of the KL-divergence in weight space, and, since  $\mathcal{F}_t$  is positive definite (can be seen from Eq. 52), it also defines a distance.<sup>20</sup> The update of the optimizer is scale-invariant and given by

$$w_{t+1} = w_t - \alpha_t \mathcal{F}_t^{-1} \nabla L_{\mathcal{D}}(w_t). \quad (54)$$

Assuming  $\mathcal{F}_t$  can be computed, the update is of cost  $\mathcal{O}(N^3)$  which is the same as the cost of a Newton update.

### Connections

The Fisher information matrix  $\mathcal{F}$  is the expected Hessian of the loss  $\ell$  under the learned distribution  $P_w(x, y)$ . The Hessian  $\Delta L_{\mathcal{D}}$ , though, is *not* the Fisher information matrix for the empirical distribution  $\hat{P}(x, y)$ , but rather the expected Hessian of the loss over the sample-distribution  $\hat{Q}(x, y)$  (the expected Hessian of the loss over  $Q(x, y)$  is  $\Delta \mathcal{L}$ ). The Gauss-Newton matrix  $G_t$  is identical to the Fisher if we approximate the marginal  $Q(x)$  (not the joint) with the empirical distribution  $\hat{Q}(x)$  while computing  $\mathcal{F}$  and if  $P_w(y|f_w)$  is in the exponential family (this include e.g., the squared and the cross-entropy loss). Since the Gauss-Newton matrix is a good approximation to the Hessian close to a minimum, thus also  $\mathcal{F}$  is, under these constraints. This does not hold further away from minima.<sup>21</sup>

<sup>19</sup> The last line of Eq. 50 holds since only the conditional probability between inputs and targets is modeled and  $p_w(x, y) = p_w(y|x)q(x)$ .

<sup>20</sup> The metric tensor  $\mathcal{F}(w)$  (locally given by Eq. 52), under some assumptions yields a Riemannian manifold. Since optimization steps are discrete, the metric defined by  $\mathcal{F}_t = \mathcal{F}(w_t)$  is used in a vicinity around  $w_t$ .

<sup>21</sup> All of this can be checked by rearranging terms in the definitions of  $\mathcal{F}$ ,  $G$ ,  $\Delta \mathcal{L}$  and  $\Delta L_{\mathcal{D}}$  and using the empirical distributions where applicable.

### Empirical Fisher

Often, it is not practical or even impossible to compute the integrals in Eq. 52 since  $Q(x)$  is unknown and  $P_w(y|x)$  complicated. The Fisher information matrix is occasionally approximated by the *empirical Fisher information matrix*  $\hat{\mathcal{F}}$  (short: empirical Fisher) which replaces  $Q(x)$  and  $P_w(y|w)$  by their empirical distributions:

$$\begin{aligned}\hat{\mathcal{F}}_t &= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \nabla \log p_{w_t}(y|x) \nabla \log p_{w_t}(y|x)^\top \\ &= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \nabla \ell(f_{w_t}(x), y) \nabla \ell(f_{w_t}(x), y)^\top.\end{aligned}\quad (55)$$

The update  $p_t = -\hat{\mathcal{F}}_t^{-1} \nabla L_{\mathcal{D}}(w_t)$  arising from the empirical Fisher is *not* scale-invariant anymore, and  $\hat{\mathcal{F}}$ , in contrast to  $\mathcal{F}$ , is also in general not a meaningful approximation to the Hessian or the generalized Gauss-Newton matrix; and even not to the Fisher itself. The benefits, though, are that it is easy to compute since the gradients  $\nabla \ell(f_{w_t}(x), y)$  are readily available; it conserves the property of positive (semi-)definiteness for, and it still provides some relative scaling of the weights (§ 2.4.2). Some of the currently most popular stochastic optimization algorithms (RMSPROP, ADAM, ...) are based on a diagonal approximation to  $\hat{\mathcal{F}}$ .

### 2.3.4 Quasi-Newton Methods

As the name suggests, quasi-Newton methods ([30], [100, § 6] for an overview) perform Newton-like updates by approximating the Hessian  $\Delta L_{\mathcal{D}}(w_t)$ . They are cheaper ( $\mathcal{O}(N^2)$  or  $\mathcal{O}(N)$ ) than a Newton update and build their approximation entirely by using previously collected gradient differences  $\Delta y_t := \nabla L_{\mathcal{D}}(w_{t+1}) - \nabla L_{\mathcal{D}}(w_t)$  and path segments  $s_t := w_{t+1} - w_t$ . The basic model assumption for all quasi-Newton methods is that the estimator  $B_{t+1}$  for the Hessian must fulfill the *secant equation*:

$$B_{t+1} s_t = \Delta y_t \quad \text{for all } t, \quad (56)$$

such that  $B_t$  can be interpreted as being the mean Hessian on the line between points  $w_t$  and  $w_{t+1}$ . The matrix  $B_t \in \mathbb{R}^{N \times N}$  is not fully identified by the  $N$  constraints of Eq. 56. Thus, for obtaining the next approximation  $B_{t+1}$ , one imposes the condition that  $B_{t+1}$  must be close to the previous one  $B_t$  in some weighted norm. The complete optimization problem to identify  $B_{t+1}$  can be phrased as:

$$B_{t+1} = \arg \min_B \|B - B_t\|_{W,F} \quad \text{s.t. } B s_t = \Delta y_t, \quad (57)$$

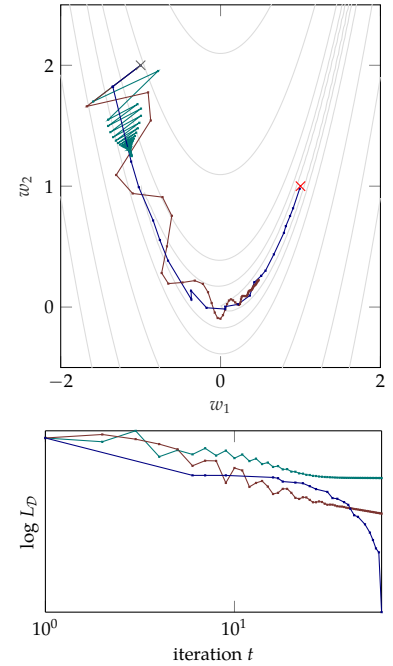


Figure 18: GD, GD+MOMENTUM, and BFGS on Rosenbrock. Top: paths of the optimizers (—/—/—, for GD, GD+MOMENTUM, and BFGS respectively); start value (x), minimizer at  $w^* = [1, 1]^\top$  (x). Bottom: Corresponding function values per iteration (double logarithmic). GD+MOMENTUM performs better than GD, although BFGS is the only optimizer of the three that reaches the minimum. The 2D-Rosenbrock polynomial is  $L(w) = 100(w_2 - w_1^2)^2 + (1 - w_1)^2$ .

[30] Dennis and Moré, “Quasi-Newton methods, motivation and theory,” 1977

[100] Nocedal and Wright, *Numerical Optimization*, 1999

where  $\|A\|_{W,F} = \|W^{1/2}AW^{1/2}\|_F$  is the Frobenius norm weighted by a positive definite matrix  $W \in \mathbb{R}^{N \times N}$ .<sup>22</sup> Different choices of  $W$  lead to different quasi-Newton methods, but the optimization update is always:

$$w_{t+1} = w_t - \alpha_t B_t^{-1} \nabla L_{\mathcal{D}}(w_t). \quad (58)$$

The Hessian estimate  $B_t$  is usually initialize with a multiple of the identity  $B_0 = bI$ ,  $b \in \mathbb{R}_+$ , the update to  $B_t$  at each step is of rank one or two, and its inverse thus analytic by the Sherman–Morrison formula.<sup>23</sup> In general, this leads to an update with cost quadratic in  $N$  due to the matrix vector product  $B^{-1} \nabla L_{\mathcal{D}}$ . Even more efficient linear updates, in cost and memory, can be achieved by only keeping the last  $M \ll N$  gradient differences  $\{\Delta y_{t-i+1}\}_{i=1,\dots,M}$  and path segments  $\{s_{t-i+1}\}_{i=1,\dots,M}$  in storage. These *limited memory* versions of quasi-Newton optimizers, e. g., L-BFGS, are highly successful, although they only model a handful of off-diagonal curvature contributions.<sup>24</sup> For non-trivial Hessians, they usually perform superior to GD in overall performance. Figure 18 shows an example of a quasi-Newton method (BFGS) in comparison to GD, and GD+MOMENTUM on a toy problem; especially the bottom plot shows how much more rapidly (super-linear in  $w_t$  for mild assumptions, plot shows log-log of  $L_{\mathcal{D}}$  vs.  $t$ ) BFGS converges than GD.

### Broyden's Method

Broyden's method [18] is the update corresponding to the solution of Eq. 57 for  $W = I$ , i. e., for the standard Frobenius norm:

$$B_{t+1} = B_t + \frac{(\Delta y_t - B_t s_t) s_t^{\top}}{s_t^{\top} s_t}. \quad (59)$$

It is in spirit much closer to root-finding methods, which approximate the Jacobian of a multi-output function. This can be seen from the estimator  $B_t$ , which is not generally symmetric, although Hessians are, because Eq. 57 did not encode this. Broyden's method is listed here for historic completeness, and also since we will use it later to draw connections to novel probabilistic second-order methods. The general solution to Eq. 57 for any positive definite  $W$  is given by

$$B_{t+1} = B_t + \frac{(\Delta y_t - B_t s_t) c_t^{\top}}{c_t^{\top} s_t}, \quad (60)$$

where  $c_t := W s_t$  is a vector in  $\mathbb{R}^N$  (e. g., [30, Thm. 7.3 and below]). Thus for the special case of choosing  $W$  (non-uniquely) such that  $c_t = \Delta y_t - B_t s_t$ , the estimator  $B_t$  is indeed symmetric and known as *symmetric rank-one* (sr1) update, probably first introduced by Davidon [112].

<sup>22</sup> The standard Frobenius norm  $\|A\|_F$  is defined as  $\|A\|_F^2 := \|A\|_{I,F}^2 = \sum_{ij} A_{ij}^2 = \text{tr}[A^{\top}A] = \|\vec{A}\|^2$  and can be obtained by setting  $W = I$ . It is not to be confused with the weighted vector-norm above used to derive steepest descent directions (overloaded notation).

$\vec{A} \in \mathbb{R}^{N^2 \times 1}$  is a vectorized version of  $A$ , constructed by stacking the elements of the matrix  $A$  row-by-row, and not column-by-column. This is just for notational convention at this point, but we will use this notation repeatedly in later chapters.

<sup>23</sup> Let  $B$  be a matrix with known inverse, and  $v, w$  vectors, then  $(B + vw^{\top})^{-1} = B^{-1} + (B^{-1}vw^{\top}B^{-1})/(1 + w^{\top}B^{-1}v)$ .

<sup>24</sup> This might even suggest, that the Hessians of typical losses  $L_{\mathcal{D}}$  are structured and can locally be captured by a simple model like a scalar-plus-low-rank matrix. Alternatively one might argue, that 'old' observations which identify a large part of the matrix  $B_t$  in the infinite memory version are outdated to some degree, since the Hessian changes with  $w$  for non-quadratic functions, and rather hamper the optimizer.

[18] Broyden, "A class of methods for solving nonlinear simultaneous equations," 1965

[30] Dennis and Moré, "Quasi-Newton methods, motivation and theory," 1977

[112] Davidon, *Variable metric method for minimization*, 1959



### Dennis Family

The Dennis family of quasi-Newton methods [29] is obtained by solving Eq. 57 with an additional symmetry constraint on the estimator  $B_{t+1}$ , i. e., :

$$B_{t+1} = \arg \min_B \|B - B_t\|_{W,F} \quad \text{s.t.} \quad B s_t = \Delta y_t \wedge B = B^\top. \quad (61)$$

The solution to Eq. 61 is given by:

$$B_{t+1} = B_t + \frac{(\Delta y_t - B_t s_t) c_t^\top + c_t (\Delta y_t - B_t s_t)^\top}{c_t^\top s_t} - \frac{c_t s_t^\top (\Delta y_t - B_t s_t) c_t^\top}{(c_t^\top s_t)^2}, \quad (62)$$

again with  $c_t = W s_t \in \mathbb{R}^N$ . The estimator  $B_{t+1}$  is symmetric by construction and the additive terms to  $B_t$  are of rank two, or, for special choice of  $c_t$ , of rank one. In general, different choices of  $c_t$  define different members of the Dennis family of quasi-Newton updates, some of which are listed below:

$$\text{sr1} \quad c_t = z(\Delta y_t - B_t s_t) \quad (63a)$$

$$\text{PSB} \quad c_t = s_t \quad (63b)$$

$$\text{GREENSTADT} \quad c_t = z B_t s_t \quad (63c)$$

$$\text{DFP} \quad c_t = z \Delta y_t \quad (63d)$$

$$\text{BFGS} \quad c_t = z \left( \Delta y_t + \sqrt{\frac{s_t^\top \Delta y_t}{s_t^\top B_t s_t}} B_t s_t \right). \quad (63e)$$

The scalar  $z > 0$  is an arbitrary positive constant. Eq. 63a again recovers the sr1 update, the others are *Powell-symmetric-Broyden* (PSB, Eq. 63b [106]), *Greenstadt* (Eq. 63c [49]), *Davidon-Fletcher-Powell* (DFP, Eq. 63d [112], [41]), and the infamous *Broyden-Fletcher-Goldfarb-Shanno* (BFGS, Eq. 63e [19] [40] [44] [123]). Again all inverses of  $B_t$  can be computed analytically by the Sherman–Morrison formula. For DFP and BFGS, positive definiteness of  $B_t$ , and thus a descent direction  $p_t$ , can be ensured by appropriate line search routines (§ 2.5), which gives them a huge advantage over their siblings. Empirically, BFGS has proven to perform very well, and, in deterministic optimization, is arguably the state-of-the-art of quasi-Newton method to date.<sup>25</sup>

### 2.3.5 (Non-)Linear Conjugate Gradients

The *linear* conjugate gradient method (CG) [62] solves for  $w^* \in \mathbb{R}^N$  in the systems  $A w^* = b$ , with  $b \in \mathbb{R}^N$  given, and  $A \in \mathbb{R}^{N \times N}$  symmetric positive definite (spd).<sup>26</sup> The method is iterative and starts with a random guess  $w_0$  for  $w^*$ . Then, it constructs successive  $w_{t+1} = w_t + \alpha_t p_t$ , with exact line searches  $\alpha_t = -\frac{r_t^\top p_t}{p_t^\top A p_t}$  and *residuals*  $r_t = A w_t - b$ , such that search directions  $p_t$  are *A-conjugate*, meaning

[29] Dennis, “On some methods based on Broyden’s secant approximations,” 1971

[106] Powell, “A new algorithm for unconstrained optimization,” 1970

[49] Greenstadt, “Variations on variable-metric methods,” 1970

[112] Davidon, *Variable metric method for minimization*, 1959

[41] Fletcher and Powell, “A rapidly convergent descent method for minimization,” 1963

[19] Broyden, “A new double-rank minimization algorithm,” 1969

[40] Fletcher, “A new approach to variable metric algorithms,” 1970

[44] Goldfarb, “A family of variable metric updates derived by variational means,” 1970

[123] Shanno, “Conditioning of quasi-Newton methods for function minimization,” 1970

<sup>25</sup> It is not entirely clear why this is so. [100, § 6.1] mention ‘self-correcting properties’ if the Hessian estimate is off, in combination with Wolfe-governed line searches; or [30, § 7.3] state that there are results for the DFP-estimator to approximate the Hessian well, which also hold for the BFGS-estimator and the *inverse* Hessian (which seems to be more useful, since  $B_t^{-1}$  is ultimately used for computing  $p_t$ ). We will follow up on this discussion in Chapter 9.

[62] Hestenes and Stiefel, “Methods of conjugate gradients for solving linear systems,” 1952

<sup>26</sup> Such a system can be interpreted as finding the optimum of a quadratic function of the form  $L(w) = \frac{1}{2} w^\top A w - b^\top w$ , with derivative  $\nabla L(w) = A w - b$ .

that  $p_j^\top A p_i \propto \delta_{ij}$ . It can be shown that the residuals  $\{r_i\}_{i=0}^t$  as well as the search directions  $\{p_i\}_{i=0}^t$  span the Krylov subspace of degree  $t$ , that is  $\mathcal{K}(r_0, t) = \text{span}\{r_0, A r_0, \dots, A^t r_0\}$ , and therefore approximate the leading vectors of the eigen-basis of  $A$ .

The *non*-linear conjugate gradient method (N-CG, first due to [42]) are variants of CG, tailored for non-linear problems, i. e., where the Hessian of a function changes with  $w$ . The residuals  $r_t$  are replaced e. g., by the gradient  $\nabla L(w_t)$  of the function and a line search is used since the exact root along  $p_t$  can not be found analytic anymore. For quadratic losses (constant Hessians  $A$ ), the iterates  $w_t$  produced by the BFGS-algorithm for exact line searches are *identical* to those of CG. Thus both algorithms coincide on linear problems [96]; but they split into separate algorithms for non-linear objectives. While, by the secant equation, it is quite clear that BFGS still infers a mean-Hessian along  $p_t$ , the inferred second-order object of N-CG is a bit more opaque; nevertheless both methods are successfully used in practice and especially the mathematical properties of linear CG are very well studied.

## 2.4 Uncertain Gradients

When the dataset  $\mathcal{D}$  is very large, it can be impractical or too costly to compute the gradient  $\nabla L_{\mathcal{D}}(w_t)$ . Eq. 30 defines a *mini-batch gradient*  $\nabla L_{\mathcal{B}}$  which is only computed on a random i. i. d. subset  $\mathcal{B} \subset \mathcal{D}$  of the full dataset. It is thus an inexact but unbiased estimator of  $\nabla L_{\mathcal{D}}$ ; and its quality depends on the data distribution as well as on the size  $|\mathcal{B}|$  of the mini-batch. Iterative optimizers which use  $\nabla L_{\mathcal{B}}$  instead of, or additionally to,  $\nabla L_{\mathcal{D}}$  are called *stochastic*, in contrast to deterministic. The preceding as well as the following subsections are to a certain degree arbitrary, since e. g., the diagonal preconditioners (§ 2.4.2) could also be used with full batch gradients; they are grouped here since they are commonly used with stochastic gradients in practice.

Before we motivate stochastic algorithms, we first need to get an intuition about the behavior of random vectors in high dimensions.

### Random Vectors in High Dimensions

Vectors in high dimensions act differently than in lower dimensions when their elements are corrupted by noise as it is the case for mini-batch gradients. As a toy experiment, suppose that  $z \sim \mathcal{N}(\mu, \sigma^2 I)$  with  $z, \mu \in \mathbb{R}^N$  and  $\sigma \geq 0$  (Gaussian assumptions for mini-batch gradients will be discussed in Chapter 5). Then, what are the mean and variance of the angle  $\theta$  between  $z$  and  $\mu$ , i. e.,  $\mathbf{E}[\theta] = \mathbf{E}[\arccos(\mu^\top z / \|\mu\| \|z\|)]$ , and  $\mathbf{var}[\theta] = \mathbf{E}[\theta^2] - \mathbf{E}[\theta]^2$ ? We can estimate these quantities by sampling independent  $\{z_i\}_{i=1}^M$  and computing  $\mathbf{E}[\theta] \approx \hat{\theta} = 1/M \sum_i \theta_i$  and

[42] Fletcher and Reeves, "Function minimization by conjugate gradients," 1964

[96] Nazareth, "A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms," 1979

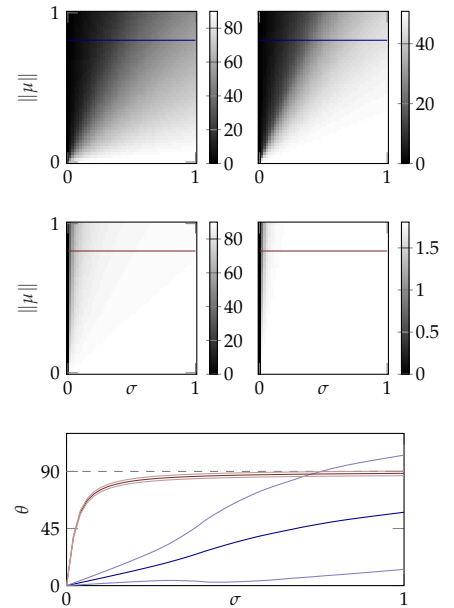


Figure 19: Orthogonality of high dimensional Gaussian random vectors. Top and middle row: mean angle  $\hat{\theta}$  (left), and corresponding standard deviation sfd (right). The axis are  $\|\mu\|$  and  $\sigma$  for an equally spaced grid between 0 and 1. Top row shows dimensionality  $N = 2$  and middle plot  $N = 1000$ . Bottom:  $\hat{\theta} \pm 1$  sfd for a fixed  $\|\mu\|$  versus  $\sigma$  (the slice is indicated in the top and middle row.)



$\mathbf{var}[\theta] \approx \hat{\mathbf{var}} = 1/M \sum_i \theta_i^2 - \hat{\theta}^2$  for  $\theta_i := \arccos(\mu^\top z_i / \|\mu\| \|z_i\|)$ . Figure 19 shows results for  $\|\mu\|$ - $\sigma$  combinations, using  $M = 10^3$  samples each. The top and middle row show the mean angle  $\hat{\theta}$  (left) and its sample standard deviation  $\hat{\text{std}}$  (right) for an equally spaced grid of  $\|\mu\|$  and  $\sigma$  between  $[0, 1]$ . The top and middle row show low and high dimensionality,  $N = 2, 1000$  respectively. The bottom row depicts  $\hat{\theta} \pm 1 \hat{\text{std}}$  for a fixed  $\|\mu\|$  versus  $\sigma$ ; the corresponding slices are indicated in the upper plots in the same colors (—/— for  $N = 2, 1000$ ). The curve for  $N = 1000$  concentrates much faster and tighter around  $\theta \approx 90^\circ$  than the one for  $N = 2$ , even for very small ratios  $\sigma/\|\mu\|$ .

The same can be observed for the whole  $\|\mu\|$ - $\sigma$ -grid (middle row) where all vectors (except the ones for vanishing noise  $\sigma$ ) stand roughly orthogonal to their mean, and, at the same time, exhibit a low variance in doing so (colorbars for scale, and Figure 20 for a 2D illustration for two instances of  $\mu$ - $\sigma$  combination). For an optimizer for instance, where  $N$  as well as  $\sigma/\|\mu\|$  might be even larger, this means that the collected stochastic gradients  $\{\nabla L_{\mathcal{B}}(w_i)\}_{i=1}^t$  span a very different subspace than the noise free ones  $\{\nabla L_{\mathcal{D}}(w_i)\}_{i=1}^t$  would, even if evaluated at the same locations  $w_i$ . This has major implications on the robustness of methods which inherently rely on inner or outer products involving e. g.,  $\Delta y_t := \nabla L_{\mathcal{D}}(w_{t+1}) - \nabla L_{\mathcal{D}}(w_t)$  such as all, but especially limited memory versions, of quasi-Newton methods. This can be observed empirically as well, where these methods either perform poorly, or fail when used with noise corrupted gradients.

### 2.4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) [113] is the exact same algorithm as GD with the exception that all gradients  $\nabla L_{\mathcal{D}}(w_t)$  are replaced by mini-batch gradients<sup>27</sup>  $\nabla L_{\mathcal{B}}(w_t)$ , thus

$$w_{t+1} = w_t - \alpha_t \nabla L_{\mathcal{B}}(w_t). \quad (64)$$

This also means that  $p_t = -\nabla L_{\mathcal{B}}(w_t)$  does not necessarily define a descent direction anymore (in fact, by the previous section, it won't in up to 50% of the cases), and it is also not possible to know this for certain unless the full  $\nabla L_{\mathcal{D}}$  was computed (which we will assume is no, or an un-desirable option). SGD does not seem to bother about this; in fact, given a function handle to some gradient—might it be noisy or not—the computer codes for GD and SGD are identical; it is oblivious to the choice of the gradient estimator or even to the size of  $|\mathcal{B}|$ . Robbins and Monro [113] showed ground-breakingly<sup>28</sup> that the sequence  $w_t$  defined by Eq. 64 converges to a minimizer  $w^*$  in expected squared error, for a diminishing learning rate schedule that fulfills  $\sum_{i=1}^{\infty} \alpha_i = \infty$  and  $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$ .<sup>29</sup> Compared to GD, the convergence rate is worse,

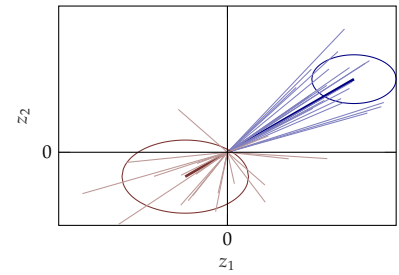


Figure 20: Distributions of angle  $\theta$  depending on  $\mu$  and  $\sigma$  in two dimensions. Shown are two Gaussian distributions with means (—/—), 1 std (—/—), and 20 random samples each (—/—).  $\theta$  is the angle between a sample and its corresponding mean. For 2D (as shown here), the distribution of  $\theta$  is quite dependent on  $\|\mu\|$  and  $\sigma$ , which is less the case in high dimensions.

<sup>27</sup> This includes mini-batches of size one. If the samples originate from a continuous data-stream rather than from a fixed, finite dataset, this is also sometimes called ‘online-learning’.

<sup>28</sup> The exposition in [113] covers the case for one-dimensional functions only, but it has since been extended to multi-dimensional settings, too.

<sup>29</sup> Convergence in expected squared error is defined as

$$\lim_{t \rightarrow \infty} \mathbf{E}[\|w_t - w^*\|^2] \rightarrow 0.$$

If this holds, then  $w_t$  converges to  $w^*$  in probability and the method is also called *consistent* for a given distribution  $p(\nabla \ell(w)|w)$ . For SGD, this is fulfilled e. g., for a learning rate decay of type  $\alpha_t = \frac{\alpha_0}{t}$  and additional (typical) assumptions on  $p(\nabla \ell|w)$  as well as the loss  $L_{\mathcal{D}}(w)$  (strong convexity and bounded gradients).

[113] Robbins and Monro, “A stochastic approximation method,” 1951

but at the same time a lot more iterates can be performed for the same budget. This is because one SGD-step is much cheaper to compute, by a fraction of  $|\mathcal{B}|/|\mathcal{D}|$ , than one GD-step. So, roughly, if the progress per  $|\mathcal{D}|/|\mathcal{B}|$  steps outweighs a single one of GD, then SGD is more efficient.<sup>30</sup>

To get an intuition for the need of a diminishing learning rate, note that the probability for a stochastic gradient to vanish might in fact be zero, even at a true root of  $\nabla L_{\mathcal{D}}$ ; if  $\mathcal{B}$  is sampled without replacement from  $\mathcal{D}$ , the variance of  $\nabla L_{\mathcal{B}}$  is proportional to  $(|\mathcal{D}|-|\mathcal{B}|)/(|\mathcal{D}||\mathcal{B}|-|\mathcal{B}|) \approx 1/|\mathcal{B}|$  for  $|\mathcal{B}| \ll |\mathcal{D}|$ . Thus, since  $\nabla L_{\mathcal{B}}(w_t)$  does not decay as much as  $\nabla L_{\mathcal{D}}(w_t)$  would for large  $t$ , the learning rate must.

Similar to GD, SGD is easy to implement, cheap, and quite robust (applicable to many problems), but the manual tuning of  $\alpha_t$  is often quite intricate. The idea of SGD though—reducing cost per step by sampling random gradients—carries over to more involved stochastic updates, some of which are discussed below. Most of them collect statistics, which in fact have some relation to  $p(\nabla \ell|w)$  and reduce the required effort for tuning their class parameters.

#### 2.4.2 Diagonal Preconditioners

Instead of constructing a matrix  $B_t$  which is scalar-plus-low-rank, as quasi-Newton methods do, *diagonal preconditioners*<sup>31</sup> learn a diagonal matrix. Each dimension is thus treated independent of all others, and separate summary statistics over many iterations are collected for each entry of  $\text{diag}[B_t]$ , which mostly includes estimators for the first and second moment of gradients. This averaging, i. e., collecting many noisy numbers to estimate one scalar, yields a more robust estimator  $B_t$  under noise, in contrast to quasi-Newton updates. The downside is that correlations between dimensions, i. e., off-diagonal elements of  $B_t$ , are neglected. Corresponding algorithms often go by the name of ‘element-wise learning rate tuning’ in the literature; we will avoid this terminology here, since it virtually covers all possible search directions, and seldom these methods actually fit an absolute scale which is the purpose of a learning rate. The diagonal matrix  $B_t$  is usually related to the Hessian or the Fisher.

Define exponential running averages over collected gradients, gradient squares, and path segments as

$$m_t = \beta m_{t-1} + (1 - \beta) \nabla L_{\mathcal{B}}(w_t) \quad (65a)$$

$$v_t = \gamma v_{t-1} + (1 - \gamma) \nabla L_{\mathcal{B}}^{\odot 2}(w_t) \quad (65b)$$

$$h_t = \zeta h_{t-1} + (1 - \zeta) (w_t - w_{t-1})^{\odot 2}, \quad (65c)$$

each with some decay factor  $\gamma, \beta, \zeta \in (0, 1)$ . The symbol ‘ $\odot$ ’ denotes the Hadamard-product (elementwise multiplication), and ‘ $\oslash$ ’ elementwise division. Additionally, define the sum (not mean) of collected

<sup>30</sup> One might argue, that it should not matter, if the information contained in a dataset is evaluated at once, or per mini-batch, hence there should not be a difference in performance. The difference, however, lies in the fact that SGD-mini-batches can be evaluated at different locations in weight space, while the full batch evaluation is only locally performed for fixed weights; additionally mini-batches allow the optimizer to explore redundant information in the dataset, as well as use hardware optimally.

In practice, there is usually an intermediate sized mini-batch size  $|\mathcal{B}|$  that performs best for a given computational budget, but ideally one would like to also adapt  $|\mathcal{B}|$  during the optimization run, since the distribution  $p(\nabla \ell|w)$  changes with  $w$ .

<sup>31</sup> We adopt the term ‘preconditioners’ from other works, e. g., [26]. ‘Preconditioner’ often only describes a theoretically desired, or a priori imposed preconditioner-matrix, and not empirical estimates thereof. Here we will call both as such.

gradient squares as  $\bar{v}_t = \sum_{i=1}^t \nabla L_{\mathcal{B}}^{\odot 2}(w_i)$ . Ignoring the bias from moving in  $w$ -space, with slight abuse of notation, it is

$$\begin{aligned} m(w) &\approx \mathbf{E}_{\hat{Q}}[\nabla L_{\mathcal{B}}(w)] = \nabla L_{\mathcal{D}}(w), \\ v(w) &\approx \mathbf{E}_{\hat{Q}}[\nabla L_{\mathcal{B}}(w)^{\odot 2}] = \nabla L_{\mathcal{D}}(w)^{\odot 2} + \mathbf{var}_{\hat{Q}}[\nabla L_{\mathcal{B}}(w)] \\ &= \text{diag}[\hat{\mathcal{F}}_t] - \mathbf{var}_{\hat{Q}}[\nabla \ell(f(w))](1 - |\mathcal{B}|^{-1}). \end{aligned} \quad (66)$$

The last line of Eq. 66 shows that for  $|\mathcal{B}| = 1$ ,  $v_t$  can also be viewed as an estimator for the diagonal of the empirical Fisher information matrix  $\hat{\mathcal{F}}_t$ ; and for  $|\mathcal{B}| > 1$ ,  $v_t$  is something like a noise-reduced  $\text{diag}[\hat{\mathcal{F}}_t]$ , or a  $\text{diag}[\hat{\mathcal{F}}_t]$  where the empirical distribution  $\hat{Q}$  is defined for mini-batches, not individual  $(x, y)$  data-pairs. The goodness of the estimators  $v_t$  and  $m_t$  depends on the number of samples they are averaged over. For exponential running averages this means, how much samples significantly contribute to them, which directly depends on the smoothing factors  $\gamma$  and  $\beta$ . If they are close to one, the estimators are less noisy, but therefore biased from evaluating samples at different locations in weight-space. Smoothing factors thus need to be tuned when used in an actual algorithm in order to find a good trade-off. Default choices for smoothing factor usually range between 0.9 – 0.999. Some of the most famous diagonal preconditioners are composed of the estimators of Eq. 65 as well as  $\bar{v}_t$ :

$$\text{ADAGRAD} \quad p_t = -\nabla L_{\mathcal{B}}(w_t) \odot (\bar{v}_t^{\odot 1/2} + \epsilon) \quad (67a)$$

$$\text{RMSPROP} \quad p_t = -\nabla L_{\mathcal{B}}(w_t) \odot (v_t^{\odot 1/2} + \epsilon) \quad (67b)$$

$$\text{ADAM} \quad p_t = -m_t \odot (v_t^{\odot 1/2} + \epsilon) \quad (67c)$$

$$\text{ADADELTA} \quad p_t = -\nabla L_{\mathcal{B}}(w_t) \odot ((h_t + \epsilon) \odot (v_t + \epsilon))^{\odot 1/2}, \quad (67d)$$

where  $\epsilon = \epsilon_0 \mathbb{I} \in \mathbb{R}^N$ ,  $\epsilon_0 \approx 10^{-8}$  are small positive perturbations for numerical stability. The list includes ADAGRAD [34], RMSPROP [135], ADADELTA [143], and the ADAM-optimizer [74].<sup>32</sup> The only scale-invariant update among the four is ADADELTA, which can be observed empirically as well ( $\alpha_t = 1$  usually works best). RMSPROP and ADAM scale the search direction inversely proportional to the square root of  $v_t$ , thus damping the numerator if  $|\mathcal{B}|^{-1} \mathbf{var}_{\hat{Q}}[\nabla \ell(f_w(x, y))]$  or  $\nabla L_{\mathcal{D}}^{\odot 2}$  is large. ADAM additionally uses the biased but therefore noise reduced estimator  $m_t$  for the gradient  $\nabla L_{\mathcal{D}}(w_t)$ .

Although none of the above methods model correlations between elements of  $\nabla \ell$ , they are highly successful and widely used in empirical risk minimization tasks and neural network training. They do, however, expose free parameters ( $\beta$ ,  $\gamma$ ,  $\zeta$ , as well as the global learning rate  $\alpha_t$ ) that can be more or less fiddly to tune.

<sup>32</sup> Additionally, ADAM performs a mild bias correction for  $m_t$  and  $v_t$ , not for the reason that samples originate from different locations  $w$ , but that  $v_t$  is initialized with  $v_0 = (1 - \gamma) \nabla L_{\mathcal{B}}(w_0)$ , and similarly for  $m_0$ .

[34] Duchi, Hazan, and Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” 2011

[135] Tieleman and Hinton, *RMSprop Gradient Optimization*, 2015

[143] Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” 2012

[74] Kingma and Ba, “Adam: A Method for Stochastic Optimization,” 2014

### 2.4.3 Second-Order Methods

Robust second-order methods that also estimate the off-diagonal of the Hessian are rather rare in stochastic optimization (for a recent overview see e. g., [15]). Some notable papers include [20], [95], and [146] which all, in one way or another, reduce the variance on gradients by either evaluating on more than one mini-batch per iteration or averaging. A hybrid of deterministic and stochastic second-order optimization is *Hessian-free optimization* (HF) or *truncated Newton* [100, § 7] which was tailored further for the purpose of training deep MLPs by [91]. In its original deterministic form, HF runs a few steps of the linear conjugate gradient algorithm (CG) on the problem  $Bp_t := (\Delta L_{\mathcal{D}}(w_t) + \lambda I)p_t = \nabla L_{\mathcal{D}}(w_t)$ , where  $\lambda \geq 0$  is a scalar ensuring that the matrix  $B$  is positive definite, or additionally that the approximate Newton steps are shortened and stay inside of a trusted region of the local quadratic approximation. The hybrid proposed by [91] also computes  $\nabla L_{\mathcal{D}}(w_t)$  on the full training dataset but, in contrast to the original HF, it runs CG on a single local mini-batch only.<sup>33</sup>

Nevertheless, especially in neural network applications involving large datasets, where even computing  $\nabla L_{\mathcal{D}}(w_t)$  is prohibitively expensive, fully fletched second-order methods do not yet exist, or are massively outperformed by SGD, momentum methods, or the diagonal preconditioners mentioned above.

## 2.5 Line Searches

Line searches control the step size  $\alpha_t$  of the optimization routine. Their main goal is to stabilize the optimizer (avoid divergence by reducing  $\alpha_t$ , but also push progress by increasing  $\alpha_t$ ), though sometimes they have auxiliary functionality in the construction of the search direction (in case of the quasi-Newton optimizers BFGS and DFP they only accept points which ensure positive definiteness of the Hessian estimate  $B_t$ ). From a practical perspective they also automate one of the most sensitive hyper-parameters of Algorithm 1, which is extremely tedious if it needed to be set by hand. Hence line searches might seem like small/non-essential subroutines at first glance but often they are at the heart of the optimizer and the most intricate problems to solve efficiently as well as to implement them robustly. The text below is partly based on sections taken from [89] and [90], listed at the end of Chapter 0. We will assume again for now that the loss  $L_{\mathcal{D}}(w_t)$  and its gradients  $\nabla L_{\mathcal{D}}(w_t)$  can be computed exactly (no mini-batching).

There is a host of existing line search variants [100, § 3]. In essence, though, these methods explore a univariate domain ‘to the right’ of a starting point, until an ‘acceptable’ point is reached. More pre-

[91] Martens, “Deep learning via Hessian-free optimization,” 2010

[15] Bottou, Curtis, and Nocedal, “Optimization Methods for Large-Scale Machine Learning,” 2016

[20] Byrd et al., “A Stochastic Quasi-Newton Method for Large-Scale Optimization,” 2014

[95] Moritz, Nishihara, and Jordan, “A Linearly-Convergent Stochastic L-BFGS Algorithm,” 2015

[146] Zhao, Haskell, and Tan, “Stochastic L-BFGS: Improved Convergence Rates and Practical Acceleration Strategies,” 2017

<sup>33</sup> This means that the Hessian-vector products needed for CG are computed on a mini-batch Hessian only, and in essence a linear problem of the form  $(\Delta L_{\mathcal{B}}(w_t) + \lambda I)p_t = \nabla L_{\mathcal{D}}$  (with  $\mathcal{D}$  for gradient and  $\mathcal{B}$  for Hessian) is solved instead. For neural networks, it is possible to compute a Hessian product with an arbitrary vector without explicitly computing and storing the Hessian matrix itself (see e. g., [102] [120]).

[100] Nocedal and Wright, *Numerical Optimization*, 1999

[120] Schraudolph, “Fast curvature matrix-vector products for second-order gradient descent,” 2002

[102] Pearlmutter, “Fast exact multiplication by the Hessian,” 1994

[89] Mahsereci and Hennig, “Probabilistic Line Searches for Stochastic Optimization,” 2015

[90] Mahsereci and Hennig, “Probabilistic Line Searches for Stochastic Optimization,” 2017

cisely, consider the problem of minimizing  $L_{\mathcal{D}}(w)$  as in Eqs. 28, 35, with access to  $\nabla L_{\mathcal{D}}(w)$ . At iteration  $t$ , some ‘outer loop’ chooses, at location  $w_t$ , a search direction  $p_t \in \mathbb{R}^N$  as in Algorithm 1. The line search operates along the univariate domain  $w(\alpha) = w_t + \alpha p_t$  for  $\alpha \in \mathbb{R}_+$ . Along this direction it collects scalar function values and projected gradients that will be denoted  $f(\alpha) = L_{\mathcal{D}}(w(\alpha))$  and  $f'(\alpha) = p_t^\top \nabla L_{\mathcal{D}}(w(\alpha)) \in \mathbb{R}$ . Most line searches involve an initial extrapolation phase to find a point  $\alpha_r$  with  $f'(\alpha_r) > 0$  (point ③ in Figure 21).

This is followed by a search in  $[0, \alpha_r]$  or  $[\alpha_{r-1}, \alpha_r]$ , by interval nesting or by interpolation of the collected function and gradient values, e.g. with cubic splines (Figure 21 and Algorithm 2, the pseudocode denotes the evaluations of  $f(\alpha)$  and  $f'(\alpha)$  as  $y$  and  $y'$  respectively).<sup>34</sup>

### 2.5.1 Wolfe Conditions for Termination

As the line search is only an auxiliary step within a larger iteration, it need not find an exact root of  $f'$ ; it suffices to find a point ‘sufficiently’ close to a minimum. The *Wolfe conditions* [141] are a widely accepted formalization of this notion; they consider  $\alpha$  acceptable if it fulfills

$$f(\alpha) \leq f(0) + \alpha c_1 f'(0) \quad (\text{W-I}) \quad \text{and} \quad (68a)$$

$$f'(\alpha) \geq c_2 f'(0) \quad (\text{W-II}), \quad (68b)$$

using two constants  $0 \leq c_1 < c_2 < 1$  chosen by the designer of the line search, not the user.<sup>35</sup> W-I is the *Armijo* or *sufficient decrease* condition [3]. It encodes that acceptable functions values should lie below a linear extrapolation line of slope  $c_1 f'(0)$ . W-II is the *curvature condition*, demanding a decrease in slope. The choice  $c_1 = 0$  accepts any value below  $f(0)$ , while  $c_1 = 1$  rejects all points for convex functions. For the curvature condition,  $c_2 = 0$  only accepts points with  $f'(\alpha) \geq 0$ ; while  $c_2 = 1$  accepts any point of greater slope than  $f'(0)$ . W-I and W-II are known as the *weak* form of the Wolfe conditions. The *strong* form replaces W-II with  $|f'(\alpha)| \leq c_2 |f'(0)|$  (W-IIa). This guards against accepting points of low function value but large positive gradient. Figure 21 and Algorithm 2 shows a conceptual sketch illustrating the typical process of a line search, and the weak and strong Wolfe conditions.

#### Ensuring Positive Definiteness of Hessian Estimates

As shown in Section 2.3.4, quasi-Newton optimizers (such as BFGS or DFP) do not explicitly encode positive definiteness of the Hessian estimate  $B_t$ . Still, this property is desirable since then, the quasi-Newton direction  $p_t = -B_t^{-1} \nabla L_{\mathcal{D}}(w_t)$  always yields a descent direction. It can

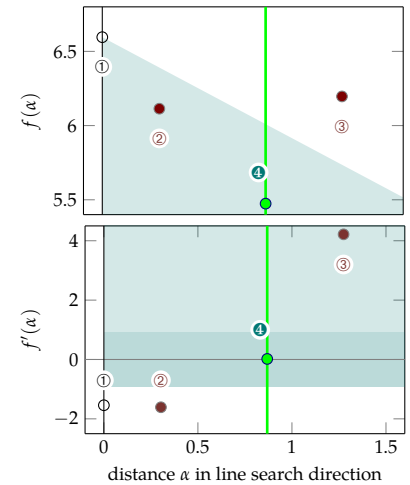


Figure 21: Sketch of line searches. The task is to tune  $\alpha$  along a univariate search direction. The search starts at the end-point ① of the previous line search, at  $\alpha = 0$ . Top: Function values (○/●/●) numbered in the order of their evaluation. Armijo acceptable region W-I (shaded) Bottom: Corresponding gradients (○/●/●). Acceptable region for W-II/W-IIa (shaded / shaded, weak, strong respectively) A sequence of extrapolation steps ②,③ finds a point of positive gradient at ③. It is followed by interpolation steps until an acceptable point ④ is found (●, —).

<sup>34</sup> This is the strategy in `minimize.m` by C. Rasmussen. It also provided a model for the probabilistic line search of Chapter 7, and it is thus explained here in more detail. At the time of writing, `minimize.m` can be found at <http://learning.eng.cam.ac.uk/carll/code/minimize/minimize.m>

[141] Wolfe, “Convergence conditions for ascent methods,” 1969

<sup>35</sup> The constraints on  $c_1$  and  $c_2$  ensure that there exists a Wolfe point for  $\alpha \in \mathbb{R}_+$  for functions  $f(\alpha)$  which are bounded below (e.g., [100, Lemma 3.1]).

[3] Armijo, “Minimization of functions having Lipschitz continuous first partial derivatives,” 1966

[100] Nocedal and Wright, *Numerical Optimization*, 1999

be shown that the estimates  $B_t^{\text{BFGS}}$  and  $B_t^{\text{DFP}}$  as in Eqs. 62 and 63 stay positive definite if they are constructed with gradients that fulfill the Wolfe conditions at each iteration. To see this write (with  $s(\alpha) = \alpha p_t$ ):

$$\begin{aligned} s(\alpha)^\top \Delta y_t &= \alpha p_t^\top \nabla L_{\mathcal{D}}(w(\alpha)) - \alpha p_t^\top \nabla L_{\mathcal{D}}(w(0)) \\ &= \alpha f'(\alpha) - \alpha f'(0) \stackrel{!}{>} 0 \\ &\rightarrow f'(\alpha) \stackrel{!}{>} f'(0) \end{aligned} \quad (69)$$

which is fulfilled by imposing Eq. 68b (note that  $f'(0)$  is negative). Theorems 7.7 and 7.8 in Dennis and Moré [30] show that  $B_t^{\text{BFGS}}$  and  $B_t^{\text{DFP}}$  are positive definite iff  $s_t^\top \Delta y_t > 0$ , thus when W-II is fulfilled.

[30] Dennis and Moré, “Quasi-Newton methods, motivation and theory,” 1977

```

1: function LINESEARCHSKETCH( $f, y_0, y'_0, \alpha_0$ )
2:    $T, Y, Y' \leftarrow \text{INITSTORAGE}(0, y_0, y'_0)$  // for scalar observation
3:    $\alpha \leftarrow \alpha_0$  // position of initial candidate
4:
5:   while budget not used and no Wolfe-point found do
6:      $[y, y'] \leftarrow f(\alpha)$  // evaluate objective
7:      $T, Y, Y' \leftarrow \text{UPDATESTORAGE}(\alpha, y, y')$ 
8:      $p^{\text{Wolfe}} \leftarrow \text{CHECKWOLFECONDITIONS}(y_0, y'_0, y, y')$ 
9:     if  $p^{\text{Wolfe}}$  true then
10:      return Wolfe-point
11:     else
12:        $\alpha \leftarrow \text{COMPUTENEXTCANDIDATE}(\alpha, y, y', T, Y, Y')$ 
13:     end if
14:   end while
15:
16:   // no Wolfe point found in budget
17:   return observed location in  $T$  with lowest value  $y$ 
18: end function
19: function COMPUTENEXTCANDIDATE( $\alpha, y, y', T, Y, Y'$ )
20:   if  $y' < 0$  and extrapolation done then // still negative slope
21:      $\alpha \leftarrow 2\alpha$  // double step size
22:   else
23:      $\alpha \leftarrow \text{CUBICMINIMUM}(y, y', T, Y, Y')$  // of previous cell
24:   end if
25:   return  $\alpha$ 
26: end function

```

Algorithm 2: Sketch of line searches. The main algorithm consists of a loop which alternates between evaluating the objective  $L_{\mathcal{D}}$ , then checking the point for acceptance (by Wolfe conditions), and finding a new candidate for evaluation by collapsing the search space  $\alpha \in \mathbb{R}_+$  efficiently. The latter is done by the subroutine COMPUTENEXTCANDIDATE which extrapolates with exponentially increasing steps until a gradient of positive slope is found, then interpolates between points of the last promising cell (with a cubic polynomial). Note that *all* relevant quantities ( $y, y', \alpha$ , etc.) are *scalar*; this means that besides evaluating  $f$ , line searches virtually add no overhead to the optimization routine. Another important characteristic of a line search is the ability to ‘immediately-accept’ after the very first function evaluation (lines 8,10), such that well scaled initial trials  $\alpha_0$  add no overhead.

### Limitation

The classic concept of a line search is based on (at least) two hard assumptions: First, that the Wolfe conditions can be checked exactly (needs descent direction  $p_t$ ), and second that the search space of  $\alpha$



can be collapsed efficiently. This can be ensured if and only if  $L_{\mathcal{D}}$  and  $\nabla L_{\mathcal{D}}$  can be evaluated exactly, but not anymore if the evaluated gradients and losses are noisy. In other words, if one is *uncertain* about the true values of  $f(\alpha)$  and  $f'(\alpha)$  it is unclear what it means to evaluate W-I, W-II, and unclear how to find suitable locations to evaluate the objective. For this reason line search subroutines can not be used in stochastic optimization settings increasingly prominent in machine learning application. Consequently the step size  $\alpha_t$  is a free parameters again that requires tuning and attention. This is done manually or (semi-)automated in an ‘outer-outer loop’, either by a parameter search algorithm, grid search, or an expert user, burning through a lot of CPU and GPU time on the way. But even leaving the cost aside, these approaches do not adapt steps sizes *locally* like line searches do, and are thus arguably always less optimal.

In an effort to gain the same automation and user-friendliness as met by classic line searches, Chapter 7 extends this concept to a *probabilistic* line search which can take in noisy evaluations of  $f$  and  $f'$  and return an analytic probability measure on the Wolfe conditions. Instead of collapsing the search space of  $\alpha$ , it will propose locations in  $\mathbb{R}_+$  which are likely to fulfill the Wolfe conditions and bring about descent. Importantly, the algorithm will be of the same cost as a classic line searches (small, constant, and independent of  $N$ ), and create little to no overhead to the outer optimization routine whose cost is dominated by the evaluation of the mini-batch gradient  $\nabla L_{\mathcal{B}}(w_t)$ .





## Quadratic Problems & Probabilistic Linear Solvers

---

THIS chapter provides basic notation and algebra for Gaussian inference on matrices for linear systems. This includes Kronecker products and Kronecker algebra (Section 3.2). It is introduced here since we will use similar models, notions, and notation in Chapter 9, where we solve a *sequence* of correlated linear systems instead of only one. Linear systems  $Ax = b$  (solve for vector  $x$ ) with symmetric positive definite (spd) matrices  $A$  can be phrased as quadratic optimization problems of the form  $\min_x x^\top Ax - x^\top b$ . Thus Section 3.1 also bridges the gap to classic quasi-Newton methods for optimization that were discussed in Section 2.3.4. In linear systems, the quantity of interest  $A^{-1}b$  is a purely mathematical object, but since it is unknown, finding it by evaluating related mathematical objects, usually  $Ax_t$  for some vectors  $x_t$ , can be phrased as inference task. The general notion of probabilistic inference on computational objects will be shortly discussed next.

### *Inference on Computational Objects*

Numerical methods estimate hard to compute or intractable mathematical objects, such as Hessians in optimization, or an integral of a function. This is done by computing well-posed<sup>1</sup> mathematical quantities which are related to them, e.g., gradients, or the integrand at various input locations. This can be seen as *inferring* an unknown (hidden) quantity from related known (observed) ones. In contrast to classic inference tasks, observation or *data* is of computational nature, in contrast to physical, mined by a CPU, by investing resources that might be measured in CPU-time or energy consumption. The general discussion is related to the field of *probabilistic numerics* [58], which phrases non-trivial numerical tasks, i.e., where the error of an estimator to the true solution can not be known easily, as probabilistic inference problems. In probabilistic numerics, the probability distributions which are used to describe the gradual commitment to the solution of a numerical problem, also provide a way to capture the lack of knowledge about the latter; and thus, if scaled well, also provide an error estimate.

Often, general mathematical properties of the object of interest are accessible by definition or by proof and can be incorporated into prior distributions, such as symmetry of the Hessian matrix, or smoothness of an integrand. A concrete algorithm then needs to trade off the

<sup>1</sup> ‘Well-posed’ is supposed to mean that the task is clearly defined as solvable by a computer in finite time, and apart from numerical errors, the precision of the estimator to the true solution is known exactly, and can be imposed a priori. These well-posed problems might be a character string encoding an analytic gradient or an integrand value, both given the input.

[58] Hennig, Osborne, and Girolami, “Probabilistic numerics and uncertainty in computations,” 2015

following points: i) encoding both, the mathematically true or known properties of the hidden quantities, and also the mathematical relation to the observables, and ii) keeping the computation tractable and cheap. This means that the designer of a numerical method needs to weigh i) and ii), such that the resulting algorithm performs best for finite computational budgets.

The next section will introduce works about solving linear systems with symmetric positive definite (spd) matrices from a probabilistic perspective. The quantity of interest here is (the inverse of) this spd matrix, and the data are path segments and gradient differences of the corresponding quadratic objective.

### 3.1 Gaussian Inference on Positive Definite Matrices

We restate the results of Hennig [56] about the solution of linear systems of the form  $\Delta L w = b$  (solve for  $w \in \mathbb{R}^N$  for given  $b \in \mathbb{R}^N$  and  $\Delta L \in \mathbb{R}^{N \times N}$  spd). In the context of second-order optimization, this implies a constant Hessian, i.e.  $\Delta L_{\mathcal{D}}(w_t) = \Delta L = \text{const}$  and a quadratic objective  $L(w) = w^\top \Delta L w - w^\top b$  with gradient  $\nabla L(w) = \Delta L w - b$ . Let  $B_t$  be the current Dennis-estimator for the Hessian, then the Dennis family of quasi-Newton methods is defined by the update rule (repeated from Eq. 62):

$$B_{t+1} = B_t + \frac{(\Delta y_t - B_t s_t) c_t^\top + c_t (\Delta y_t - B_t s_t)^\top}{c_t^\top s_t} - \frac{c_t s_t^\top (\Delta y_t - B_t s_t) c_t^\top}{(c_t^\top s_t)^2}, \quad (70)$$

where  $s_t := w_{t+1} - w_t$ ,  $\Delta y_t := y_{t+1} - y_t = \nabla L(w_{t+1}) - \nabla L(w_t)$  and  $c_t = W_t s_t$  as in Section 2.3.4.<sup>2</sup> The identical estimator for  $\Delta L$  for one step  $t \rightarrow t+1$  can be derived by phrasing the minimization problem of Eq. 61 as a multi-dimensional Gaussian inference problem on the matrix  $\Delta L$  with prior  $p(\Delta L) = \mathcal{N}(B_t, W_t \otimes W_t)$  and likelihood  $p(\Delta y_t | \Delta L) = \lim_{\beta \rightarrow 0} \mathcal{N}(\Delta L s_t, \beta \Lambda) = \delta(\Delta y_t - \Delta L s_t)$ . The posterior is  $p(\Delta L | \Delta y_t) = \mathcal{N}(B_{t+1}, W_{t+1} \otimes W_{t+1})$  with mean and covariance

$$\begin{aligned} B_{t+1} &= B_t + \frac{(\Delta y_t - B_t s_t)(W_t s_t)^\top + c_t (\Delta y_t - B_t s_t)^\top}{s_t^\top W_t s_t} - \frac{(W_t s_t) s_t^\top (\Delta y_t - B_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)^2} \\ W_{t+1} &= W_t - \frac{(W_t s_t)(W_t s_t)^\top}{s_t^\top W_t s_t}. \end{aligned} \quad (71)$$

Thus, the different sub-algorithms of the Dennis family differ in their choice of  $c_t$  (or  $W_t$ ) according to:

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

<sup>2</sup> Short reminder:  $W \in \mathbb{R}^{N \times N}$  spd was used to define the minimization problem  $B_{t+1} = \arg \min_B \|B - B_t\|_{W,F}^2$  s.t.  $B s_t = \Delta y_t \wedge B = B^\top$ . This also means, that a constant  $\Delta L$ , and not just its estimator, always fulfills the secant equation  $\Delta L s_t = \Delta y_t$ , too.

$$\text{SR1} \quad c_t = \zeta(\Delta y_t - B_t s_t) \quad W_t = \zeta(\Delta L - B_t) \quad (72a)$$

$$\text{PSB} \quad c_t = \zeta s_t \quad W_t = \zeta I \quad (72b)$$

$$\text{GREENSTADT} \quad c_t = \zeta B_t s_t \quad W_t = \zeta B_t \quad (72c)$$

$$\text{DFP} \quad c_t = \zeta \Delta y_t \quad W_t = \zeta \Delta L \quad (72d)$$

$$\text{BFGS} \quad c_t = \zeta \left( \Delta y_t + \sqrt{\frac{s_t^\top \Delta y_t}{s_t^\top B_t s_t}} B_t s_t \right) \quad W_t = \zeta \left( \Delta L + \sqrt{\frac{s_t^\top \Delta y_t}{s_t^\top B_t s_t}} B_t \right) \quad (72e)$$

for some arbitrary positive scalar  $\zeta$ . The relation  $c_t \leftrightarrow W_t$  is not uniquely defined for a single  $s_t$ , but for all possibly occurring  $s_t \in \mathbb{R}^N$ . The equivalence holds in general for one-step Gaussian regression on symmetric matrices (Theorem 2.3, Corollary 2.4 and Corollary 3.1 in [56]), and in particular for `BFGS` and `DFP` also for multi-step Gaussian regression, when exact line searches are available (Lemma 3.2 and Lemma 3.3 in [56]).

Similarly Broyden's methods can be phrased as the non-symmetric update:

$$B_{t+1} = B_t + \frac{(\Delta y_t - B_t s_t) s_t^\top}{s_t^\top s_t}, \quad (73)$$

which is equivalent to the posterior mean after one-step of multi-dimensional Gaussian regression on a  $\Delta L$  with prior  $p(\Delta L) = \mathcal{N}(B_t, W_t \otimes W_t)$  that does not encode the symmetry of  $\Delta L$  (Kronecker instead of symmetric Kronecker covariance, see § 3.2), and likelihood as above. Eq. 73 results for the parameter choice  $W_t = \zeta I$  (Lemma 2.1 and thereafter in [56]).

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

The general update for *multiple* observations  $\Delta Y = [\Delta y_1, \dots, \Delta y_T] \in \mathbb{R}^{N \times T}$  and  $S = [s_1, \dots, s_T] \in \mathbb{R}^{N \times T}$ , with likelihood  $p(\Delta Y | \Delta L) = \delta(\Delta Y - \Delta L S)$  is given by:

$$\begin{aligned} B_T &= B_0 + (\Delta Y - B_0 S)[S^\top W_0 S]^{-1}(W_0 S)^\top + (W_0 S)[S^\top W_0 S]^{-1}(\Delta Y - B_0 S)^\top \\ &\quad - (W_0 S)[S^\top W_0 S]^{-1}(S^\top(\Delta Y - B_0 S))[S^\top W_0 S]^{-1}(W_0 S)^\top \\ W_T &= W_0 - (W_0 S)[S^\top W_0 S]^{-1}(W_0 S)^\top \end{aligned} \quad (74)$$

for the symmetric update (Theorem 2.3 in [56]). Eq. 71 is recovered for diagonal  $S^\top W_0 S$  ( $W_0$ -conjugate search directions  $S$ ), and  $W_t = W_0$  for all  $t$ .

### 3.2 Kronecker Algebra

This section recaps the most important algebraic properties of the Kronecker product for better reference, since we will heavily use it, especially throughout Chapter 9 (see e. g., [85] for an overview). Details and most proofs can be found in Appendix A. The Kronecker product,

[85] Loan, "The ubiquitous Kronecker product," 2000

denoted by ' $\otimes$ ', between two matrices  $A \in \mathbb{R}^{N_1 \times N_2}$  and  $B \in \mathbb{R}^{K_1 \times K_2}$  is defined as:

$$(A \otimes B)_{(ij),(kl)} = A_{ik}B_{jl}, \quad i = 1 \dots N_1, k = 1 \dots N_2, \\ j = 1 \dots K_1, l = 1 \dots K_2. \quad (75)$$

The resulting matrix is of size  $N_1K_1 \times N_2K_2$  and its elements are denoted by two double-indices  $(ij), (kl)$ . A consequence of Eq. 75 is that, a Kronecker product applied to a *vectorized* matrix  $\vec{X} \in \mathbb{R}^{N_2K_2}$ , is the vectorized version of two lower-dimensional, cheaper matrix-matrix multiplications:<sup>3</sup>

$$(A \otimes B)\vec{X} = \overrightarrow{AXB^T}. \quad (76)$$

The Kronecker product is a generalization of the outer products of two vectors, and it also exhibits similar algebraic structures: For matrices  $A$  and  $B$  of appropriate size and properties, it is

$$\text{TRANSPOSE} \quad (A \otimes B)^T = A^T \otimes B^T \quad (77a)$$

$$\text{INVERSE} \quad (A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \quad (77b)$$

$$\text{FACTORIZING} \quad (A \otimes B)(C \otimes D) = AC \otimes BD \quad (77c)$$

$$\text{DISTRIBUTIVE LEFT} \quad (A \otimes B) + (A \otimes C) = A \otimes (B + C) \quad (77d)$$

$$\text{DISTRIBUTIVE RIGHT} \quad (A \otimes B) + (C \otimes B) = (A + C) \otimes B \quad (77e)$$

$$\text{ASSOCIATIVE} \quad (A \otimes B) \otimes C = A \otimes (B \otimes C) \quad (77f)$$

$$\text{TRACE} \quad \text{tr}[(A \otimes B)] = \text{tr}[A] \text{tr}[B]. \quad (77g)$$

Eqs. 77 especially hold if one or both,  $A$  and  $B$ , are scalars or vectors (where applicable). All formulas of Eq. 77 exploit the factorizing structure of Eq. 75, such that often one side of the equalities is much cheaper to compute in practice than the other. For example the left hand side of Eq. 77b, i. e., the inverse of  $A \otimes B$  (for square and invertible  $A$  and  $B$ ) naively is of cost  $\mathcal{O}(N^3K^3)$ , but the right hand side only of cost  $\mathcal{O}(N^3 + K^3)$ . Importantly, the Kronecker product of two full rank matrices  $A$  and  $B$  is also of full rank, or more precisely  $\text{rk}[A \otimes B] = \text{rk}[A] \text{rk}[B]$ . Thus the assumption of Kronecker structure on a large matrix does not restrict the space it is operating on when applied to a vector ( $A \otimes B$  is still bijective if  $A$  and  $B$  are). It merely restricts the type of full-rank matrices to ones whose elements factor as in Eq. 75.<sup>4</sup> For a better understanding, we take a look at the closest Kronecker approximation to an arbitrary matrix  $C \in \mathbb{R}^{N_1K_1 \times N_2K_2}$  under the Frobenius norm:

$$A^*, B^* = \arg \min_{A, B} \|C - A \otimes B\|_F^2. \quad (78)$$

<sup>3</sup> The operation  $\vec{\phantom{x}}$  stacks the rows of a matrix  $X \in \mathbb{R}^{N_2 \times K_2}$  into a column vector  $\vec{X} \in \mathbb{R}^{N_2K_2 \times 1}$ .

<sup>4</sup> This property will become important later again, where Kronecker structure is imposed on the covariance matrix of a multi-dimensional Gaussian distribution. Restricting the rank would restrict the hypothesis class; restricting its structure only shifts mass in the space.

There exists a fixed, known permutation  $\mathcal{R}$  such that the Kronecker product can be written as an outer product of the vectorized matrices  $A$  and  $B$ :  $\mathcal{R}(A \otimes B) = \vec{A}\vec{B}^\top$  (details in Appendix A) and thus Eq. 78 can be re-phrased as a *rank-one* approximation problem in a  $N_1N_2 \times K_1K_2$  dimensional space

$$\vec{A}^*, \vec{B}^* = \arg \min_{A, B} \left\| \mathcal{R}(C) - \vec{A}\vec{B}^\top \right\|_F^2. \quad (79)$$

So, the Kronecker product is a rank-one matrix in a permuted space, and—under the Frobenius norm—is the closest rank-one approximation to an arbitrary matrix in this space. With this, also the *weighted* Frobenius norm can be re-written as an outer product

$$\|A\|_{W, F}^2 = \vec{A}^\top (W \otimes W) \vec{A}, \quad (80)$$

for  $W$  symmetric positive definite.

### 3.2.1 Symmetric Kronecker Product

The *symmetric* Kronecker product (overview in Appendix A.2), denoted by ' $\otimes$ ' is the symmetrized version of the Kronecker product, defined by  $(A \otimes B) := \Gamma(A \otimes B)\Gamma^\top$  where  $\Gamma$  is the symmetrization operator  $\Gamma C = \frac{1}{2}(C + C^\top)$  for some square matrix  $C$ . Thus with  $A$  and  $B$  square and of same size:

$$\begin{aligned} (A \otimes B)_{(i,j),(k,l)} &= \frac{1}{4}(A_{ik}B_{jl} + A_{il}B_{jk} + A_{jk}B_{il} + A_{jl}B_{ik}) \\ (A \otimes B)\vec{X} &= \frac{1}{4}(\overrightarrow{AXB^\top + AX^\top B^\top + BX^\top A^\top + BXA^\top}). \end{aligned} \quad (81)$$

If  $A = B$  then Eq. 81 simplifies to

$$\begin{aligned} (A \otimes A)_{(i,j),(k,l)} &= \frac{1}{2}(A_{ik}A_{jl} + A_{jk}A_{il}) \\ (A \otimes A)\vec{X} &= \frac{1}{2}(\overrightarrow{AXA^\top + AX^\top A^\top}). \end{aligned} \quad (82)$$

Similar to the Kronecker product, the symmetric version has some nice algebraic properties, although not all of them are carried over:

$$\text{TRANSPOSE} \quad (A \otimes B)^\top = A^\top \otimes B^\top \quad (83a)$$

$$\text{FACTORIZING} \quad (A \otimes A)(C \otimes C) = AC \otimes AC \quad \text{but} \quad (A \otimes B)(C \otimes D) \neq (AC \otimes BD) \quad (83b)$$

$$(A \otimes B)(C \otimes D) = \frac{1}{2}[AC \otimes BD + AD \otimes BC] \quad (83c)$$

$$\text{INVERSE} \quad (A \otimes A)^{-1} = (A^{-1} \otimes A^{-1}) \quad \text{but} \quad (A \otimes B)^{-1} \neq (A^{-1} \otimes B^{-1}) \quad (83d)$$

$$\text{COMMUTATIVE} \quad A \otimes B = B \otimes A \quad \text{but} \quad A \otimes B \neq B \otimes A \quad (83e)$$

$$\text{TRACE} \quad \text{tr}[A \otimes B] = \frac{1}{2}(\text{tr}[A] \text{tr}[B] + \text{tr}[AB]). \quad (83f)$$

Also,  $A \otimes B$  does not have the distributive property anymore. If  $A$  and  $B$  are both of full rank  $N$ , then  $A \otimes B$  is only of rank  $\frac{1}{2}N(N + 1)$ , as can be also seen from the definition via  $\Gamma$ .

## Miscellaneous

---

**T**HIS chapter introduces frameworks and notation of topics that do not fit well into the previous chapters but are still important for the subsequent chapters. Section 4.1 introduces Bayesian optimization, a fully probabilistic concept for global optimization. Section 4.2 recaps the central limit theorem, which provides an argument that the distribution of sums of i. i. d. random variables can be approximated by a Gaussian.

### 4.1 Bayesian Optimization

The task of Bayesian optimization [79] [93] is to find the global minimizer  $x^*$  of a function  $f(x)$ , that is:

$$x^* = \arg \min_x f(x) \quad (84)$$

with  $x^*, x \in \Omega \subset \mathbb{R}^D$  and  $f : \Omega \rightarrow \mathbb{R}$ . The only access to  $f$  is a restricted number of possibly noisy function evaluations  $y$ , as well as sometimes the corresponding gradients. Often, there is not even an analytic description of  $f$  and its evaluation is time-consuming and/or expensive. Besides modeling  $f$  adequately, the main challenge is to choose the locations of these evaluations such that good estimators for the minimizer  $x^*$  and its function value  $f(x^*)$  are obtained. Bayesian optimization has three main ingredients: i) A surrogate for the unknown objective  $f$ , ii) an acquisition function that depends on the surrogate and  $y$ s, which encodes a strategy for finding good evaluation points. Maximizing the acquisition function defines an ‘inner’ non-convex optimization problem after each evaluation of  $f$  that can be solved with standard greedy optimization procedures, and iii) possibly a stopping criterion. A pseudocode is shown in Algorithm 3.

#### *Gaussian Process Surrogate*

The function  $f$  is usually modeled with a Gaussian process (GP) with kernel  $k(x, x')$  and mean function  $\mu(x)$ , such that  $f \sim \mathcal{GP}(\mu, k)$ .<sup>1</sup> The kernel encodes general assumptions about  $f$  such as smoothness, length-scales of variability or even periodicity. It is usually dependent on its own kernel-parameters, called hyper-parameters, that need to be set, learned, or marginalized out.

[79] Kushner, “A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise,” 1964

[93] Moćkus, “On Bayesian Methods for Seeking the Extremum,” 1975

[121] Shah, Wilson, and Ghahramani, “Student-t Processes as Alternatives to Gaussian Processes,” 2014

<sup>1</sup> A Gaussian process is arguably the most widely used surrogate in BO, but there are also other ones e. g., Student-t processes [121].

### Acquisition Function

The acquisition function  $u(x)$  characterizes the *active* part of the optimizer. It is based on a utility that encodes a desired strategy for finding  $x^*$ . Examples are the *probability of improvement* (POI) [79], the *upper/lower-confidence bound* (UCB) [4] [130], the *expected improvement* (EI) [71] or *entropy search* (ES) [59] [61]:

$$u_{\text{POI}}(x) = p(f(x) < \eta) = \text{cdf}_{\mathcal{N}(0,1)}(\gamma(x)) \quad (85a)$$

$$u_{\text{UCB}}(x) = -\mu_{f|y}(x) + \beta^{\frac{1}{2}} s_{f|y}(x), \quad \beta \in \mathbb{R}_+ \quad (85b)$$

$$u_{\text{EI}}(x) = \mathbf{E}_{f(x)}[\min\{0, \eta - f(x)\}] \quad (85c)$$

$$= s_{f|y}(x) \left[ \gamma(x) \text{cdf}_{\mathcal{N}(0,1)}(\gamma(x)) + \text{pdf}_{\mathcal{N}(0,1)}(\gamma(x)) \right]$$

$$u_{\text{ES}}(x) = \mathbf{E}_{f(x)}[H[p(x^*|f(x), x)]], \quad (85d)$$

where  $s_{f|y}(x) = k_{f|y}^{1/2}(x, x)$ ,  $\gamma(x) = (\eta - \mu_{f|y}(x))/s_{f|y}(x)$ , and  $\eta$  is a current best guess for  $f(x^*)$ . All of them are heuristics which trade off the exploration of the domain of possible minimizers, and further exploitation of promising regions of low values of  $f$ . Figure 22 shows two toy examples of the same GPs as in Section 1.3, Figure 9, but this time conditioned on two noise-free observations  $y$  ( $\bullet \rightarrow$ ). The top row shows the corresponding posterior Gram-matrices. Left: squared exponential kernel, right: periodic plus Wiener kernel; the second and fourth row show the posterior GPs. Rows three and five show three different acquisition functions: EI, UCB, and POI (—/---/----). The vertical bars (—/---/----) indicate the location of the maximum of each acquisition function, which is the point that is chosen for the next evaluation. Note that both GPs are conditioned on identical observations. It is apparent that the strategy of the optimizer depends crucially on the choice of the GP (the surrogate for  $f$ ) as well as on the search heuristic (acquisition function).

### Variants

The literature on BO is vast; an overview can be found e. g., in Shahriari et al. [122]. Bayesian optimization as described in Algorithm 3 is *serial* in the function evaluations, meaning that the acquisition function is optimal for one subsequent observation. There are also acquisition functions which are less myopic and encode optimality for multi-step look-ahead [47]. Related to this, there are variants of BO, called ‘parallel-BO’ or ‘batch-BO’, where the acquisition function is modified to return more than one point of interest, such that speedups from parallel evaluations of  $f$  at multiple locations are accessible [46]. Additionally, recent research also tries to include gradient information of  $f$  into the GP, as well as into the acquisition function [142]. Although this increases the cost, both of updating the GP and optimizing the

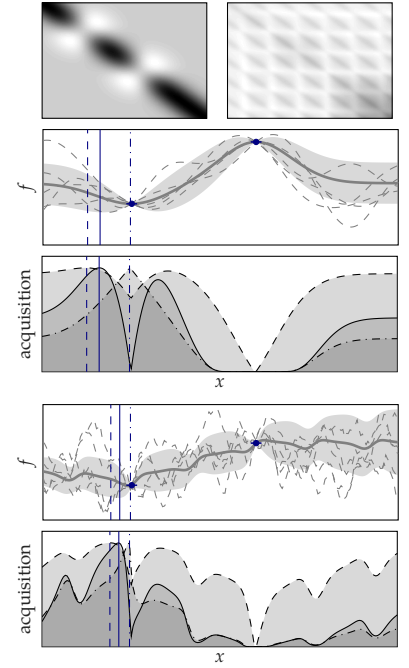


Figure 22: Bayesian optimization acquisition functions. *Row 1*: Gram matrices of the posterior GPs of rows 2 and 4 (squared exponential and periodic plus Wiener respectively). *Rows 3 and 5*: The acquisition functions EI, UCB, and POI.

[4] Auer, “Using confidence bounds for exploitation-exploration trade-offs,” 2003

[130] Srinivas et al., “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design,” 2010

[71] Jones, Schonlau, and Welch, “Efficient global optimization of expensive black-box functions,” 1998

[59] Hennig and Schuler, “Entropy Search for Information-Efficient Global Optimization,” 2012

[61] Hernández-Lobato, Hoffman, and Ghahramani, “Predictive Entropy Search for Efficient Global Optimization of Black-box Functions,” 2014

[122] Shahriari et al., “Taking the Human Out of the Loop: A Review of Bayesian Optimization,” 2016

[47] González, Osborne, and Lawrence, “GLASSES: Relieving The Myopia Of Bayesian Optimisation,” 2015

[46] González et al., “Batch Bayesian Optimization via Local Penalization,” 2015

[142] Wu et al., “Bayesian Optimization with Gradients,” 2017



acquisition function, the sample efficiency can possibly be increased. Finally, there are variants of `bo` which use a different class of surrogates for  $f$ , such as Student-t processes [121] or even neural networks [128].

```

1: function BAYESOPTSKETCH( $f, \mathcal{GP}$ )
2:   while budget not used do
3:      $u(x) \leftarrow$  DEFINEACQUISITIONFUNCTION( $\mathcal{GP}$ )
4:      $x_t \leftarrow$  OPTIMIZEACQUISITIONFUNCTION( $u(x)$ )
5:      $y_t \leftarrow f(x_t)$  // evaluate function
6:      $\mathcal{GP} \leftarrow$  UPDATEGP( $\mathcal{GP}, x_t, y_t$ )
7:   end while
8:   return  $x_t$  with minimal value  $y_t$ 
9: end function

```

## 4.2 Central Limit Theorem

The central limit theorem (CLT) probably goes back to Abraham De Moivre in 1738, the ‘Doctrine of Chances’ [27, p. 243] who considered the special case of coin tosses, 39 years before Carl Friedrich Gauss was born. The name was popularized by Pólya in 1920 [104] who called it ‘central’ because it plays such a ‘central role’ in probability calculus, and names Laplace as the proper inventor. The definition below closely follows Feller [39] § 8.4.

Let  $\mathcal{S} = \{x_1, \dots, x_{|\mathcal{S}|}\}$  be a set of i. i. d. random variables with finite mean  $\mu$  and invertible covariance  $\Sigma$ . As the set size  $|\mathcal{S}| \rightarrow \infty$ , the distribution of the sum  $\bar{X}_{|\mathcal{S}|} := |\mathcal{S}|^{-\frac{1}{2}} \sum_{i=1}^{|\mathcal{S}|} x_i$ , which itself is a random variable, tends to a normal distribution  $\mathcal{N}(\mu, \Sigma)$ .

This means that under mild assumptions, normalized sums of i. i. d. random variables will eventually follow a Gaussian distribution. Nevertheless the CLT is a limit-statement, meaning that for finite sums of random variables  $|\mathcal{S}| < \infty$  it is generally not clear if a Gaussian approximation to the probability distribution of  $\bar{X}_{|\mathcal{S}|}$  is sensible. In practice though finite- $|\mathcal{S}|$  approximations are already quite accurate for a variety of base-distributions (the distributions that the  $x_i$  follow).<sup>2</sup>

Figure 23 illustrates Gaussian approximations to the distributions of  $\bar{X}_{|\mathcal{S}|}$  for finite sums. Different rows show different (quite small) set sizes  $|\mathcal{S}| = 2, 10$ , and 100 which are roughly on the lower end of mini-batch sizes used in stochastic optimization routines. The columns from left to right show different base-distribution (■): uniform, Gamma, and a mixture of three Gaussians. They showcase different properties of distributions of random variables that occur in optimization, e. g., strictly positive (individual losses), multi-modal (individual gradients and losses), and clipped (individual gradients). The his-

[121] Shah, Wilson, and Ghahramani, “Student-t Processes as Alternatives to Gaussian Processes,” 2014

Algorithm 3: Sketch of the Bayesian optimization algorithm. The basic structure is a loop which alternates between finding a location for evaluation by optimizing the acquisition function, evaluating the point, and updating the surrogate on  $f$ .

[128] Snoek et al., “Scalable Bayesian Optimization Using Deep Neural Networks,” 2015

[27] DeMoivre, *The Doctrine of Chances*, 1738

[104] Pólya, “Über den zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung und das Momentenproblem,” 1922

[39] Feller, *An Introduction to Probability Theory and Its Applications*, 1971

[12] Berry, “The accuracy of the Gaussian approximation to the sum of independent variates,” 1941

[37] Esseen, “On the Liapounoff limit of error in the theory of probability,” 1942

<sup>2</sup> Quantitative convergence results exist (with known upper bounds on the constant) for slightly more restrictive assumptions on the distribution of the  $x_i$ , e. g., the Berry-Esseen Theorem [12] [37] assumes that  $\mathbb{E}[|x|^3] < \infty$  and shows convergence in Kolmogorov-Smirnov distance.

togram ( $\blacksquare$ ) in each plot is composed of  $10^4$  draws of  $\bar{X}_{|\mathcal{S}|}$  for each combination of base-distribution and mini-batch size  $|\mathcal{S}|$  (a total of  $10^4 \cdot |\mathcal{S}|$  draws of  $x_i$  from the base-distribution). The Gaussian curve ( $\text{---}$ ) is defined by the sample mean and variance of the  $10^4$  samples  $\bar{X}_{|\mathcal{S}|}$ ; if it is indistinguishable from the histogram, then the Gaussian fit is very good. Already a set size of  $|\mathcal{S}| = 10$  (second row) produces very good Gaussian approximations, and even the one for the smallest set size possible ( $|\mathcal{S}| = 2$ ) is passable at least for the uniform and Gamma base-distributions. Additionally the  $|\mathcal{S}|^{-\frac{1}{2}}$ -dependence of the standard deviation of the fitted Gaussians to the mini-batch size can be observed since the Gaussian fits each get more narrow/ more certain (from top row to bottom row) with increasing  $|\mathcal{S}|$ .

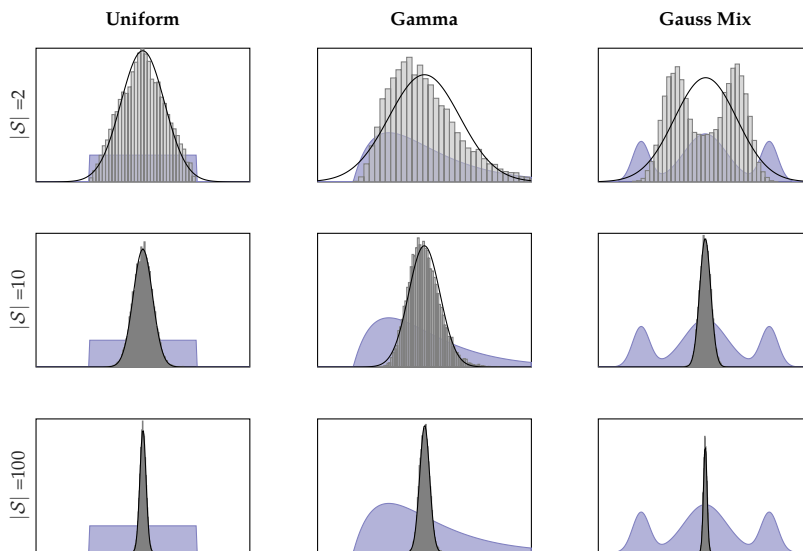


Figure 23: Illustration of Gauss approximation for finite set sizes. All plots: Base distribution ( $\blacksquare$ ). Histogram of samples  $\bar{X}_{|\mathcal{S}|}$  ( $\blacksquare$ ) each for  $10^4$  samples. Gaussian fit using the sample mean and sample variance of the mini-batch samples  $\bar{X}_{|\mathcal{S}|}$  ( $\text{---}$ ). Columns from left to right: Uniform, Gamma, Gaussian mixture as base distribution. Rows from top to bottom: increasing set size  $|\mathcal{S}| = 2, 10$  and  $100$  respectively. Scale of ordinate of the Gauss-fit and the normalized histogram are identical but arbitrary for the base-distribution.

We will use the central-limit-argument later in this thesis, especially in Chapter 5, to justify Gaussian approximations to mini-batch gradients as well as, to a lesser degree, to mini-batch losses. Heuristics in a similar style to Figure 23 on a real world problem will back-up the claim.

Part II

## Overfitting, Generalization & Early-Stopping



## Local Distributions of Losses and Gradients

---

**I**N stochastic empirical risk minimization as introduced in Chapter 2, the mini-batch size  $|\mathcal{B}|$  trades off computational speed with the precision of gradient and loss estimators. We will see that this relation locally can be described approximately with the variances of these quantities which drop linearly with  $|\mathcal{B}|$ . This chapter lays the foundations for the empirical, local variance estimator of stochastic gradients and losses, which will be used by all following chapters. The text is partly based on the publication [90].

[90] Mahsereci and Hennig, “Probabilistic Line Searches for Stochastic Optimization,” 2017

### 5.1 Likelihood for Losses and Gradients

The estimators  $\nabla L_{\mathcal{D}}(w)$  and  $\nabla L_{\mathcal{B}}(w)$  of Eq. 29 and 30 are both sums of independent random variables drawn from a common distribution. More precisely the datapoints  $d$  in  $\mathcal{D}$  or  $\mathcal{B}$  are drawn i. i. d. from the data-distribution  $Q$  and are then transformed by the deterministic network and loss function  $\ell(f_w(x), y)$  with  $d = (x, y)$ .<sup>1</sup> Thus the central limit theorem (CLT, Chapter 4 § 4.2) applies and the estimators  $\nabla L_{\mathcal{D}}(w)$ , and  $\nabla L_{\mathcal{B}}(w)$  locally (i. e., for given parameters  $w$ ) are Gaussian distributed around the gradient of the risk  $\nabla \mathcal{L}(w)$  in the limit  $|\mathcal{B}|, |\mathcal{D}| \rightarrow \infty$ . Strictly speaking, the CLT does not make a statement about the distribution for finite  $|\mathcal{B}|, |\mathcal{D}|$ , but often Gaussian approximations are already sufficiently accurate even for low values of  $|\mathcal{B}|$  (some tens or hundreds). The same holds for the estimators  $L_{\mathcal{D}}(w)$  and  $L_{\mathcal{B}}(w)$ , although the Gaussian approximation might be a bit poorer for  $w$  close to a minimizer of the empirical risk, since  $L_{\mathcal{D}}(w)$ ,  $L_{\mathcal{B}}(w)$ , as well as  $\mathcal{L}(w)$  are bounded below. For a general dataset  $\mathcal{S}$  of size  $|\mathcal{S}|$  with elements drawn independently from the data-distribution  $Q$ , we thus approximately get:

<sup>1</sup> For ease of notation, and since the map  $f$  is not important for the argument below, we will write  $\ell(w, d)$  instead of  $\ell(f_w(x), y)$ .

$$L_{\mathcal{S}}(w) \sim \mathcal{N} \left( \mathcal{L}(w), \frac{\Lambda(w)}{|\mathcal{S}|} \right) \quad (86a)$$

$$\nabla L_{\mathcal{S}}(w) \sim \mathcal{N} \left( \nabla \mathcal{L}(w), \frac{\Sigma(w)}{|\mathcal{S}|} \right) \quad (86b)$$

with population (co-)variances  $\Lambda(w)$  and  $\Sigma(w)$  of function value and gradients respectively:

$$\Lambda(w) = \mathbf{var}_{d \sim Q}[\ell(w, d)] \in \mathbb{R}_+ \quad (87a)$$

$$\Sigma(w) = \mathbf{cov}_{d \sim Q}[\nabla \ell(w, d)] \in \mathbb{R}^{N \times N}. \quad (87b)$$

The (co-)variances of  $L_{\mathcal{S}}$  and  $\nabla L_{\mathcal{S}}$  scale inversely proportional to the set size  $|\mathcal{S}|$ . Efficient ways of locally estimating the diagonal of  $\Sigma(w)$  (the variances only) and  $\Lambda(w)$  are discussed in Section 5.2.

The statements in Eq. 86 and 87 especially hold for each isolated mini-batch  $\mathcal{B} = \mathcal{S}$  or the full dataset  $\mathcal{D} = \mathcal{S}$ . Estimators of different iterates (e. g., from  $w_t$  to  $w_{t+1}$ ) are dependent, since, in practice, there is only one *finite* dataset available with which either consecutive  $\nabla L_{\mathcal{D}}$  are computed (each time using the *same*  $\mathcal{D}$  with  $d \sim Q$ ), or consecutive  $\nabla L_{\mathcal{B}}$  where  $\mathcal{B} \subset \mathcal{D}$  is sub-sampled from a *fixed* set  $\mathcal{D}$ . The objective that can be optimized is thus always limited to the empirical risk  $L_{\mathcal{D}}$  defined by the whole dataset. When  $\mathcal{B} \subset \mathcal{D}$  is locally sampled with replacement, a similar statement as in Eq. 86 and 87 can be made which regards the finite empirical distribution  $\hat{Q}$  as ground truth. Then:

$$L_{\mathcal{B}}(w) \sim \mathcal{N}\left(L_{\mathcal{D}}(w), \frac{\Lambda_{\mathcal{D}}(w)}{|\mathcal{B}|}\right) \quad (88a)$$

$$\nabla L_{\mathcal{B}}(w) \sim \mathcal{N}\left(\nabla L_{\mathcal{D}}(w), \frac{\Sigma_{\mathcal{D}}(w)}{|\mathcal{B}|}\right) \quad (88b)$$

with the (co-)variances:

$$\Lambda_{\mathcal{D}}(w) = \mathbf{var}_{d \sim \hat{Q}}[\ell(w, d)] = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (\ell(w, d_i) - L_{\mathcal{D}}(w))^2 \quad (89a)$$

$$\Sigma_{\mathcal{D}}(w) = \mathbf{cov}_{d \sim \hat{Q}}[\nabla \ell(w, d)] = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (\nabla \ell(w, d_i) - \nabla L_{\mathcal{D}}(w))(\nabla \ell(w, d_i) - \nabla L_{\mathcal{D}}(w))^{\top}. \quad (89b)$$

If  $\mathcal{B} \subset \mathcal{D}$  is sampled without replacement, the factor in front the population (co-)variances becomes  $(|\mathcal{D}|-|\mathcal{B}|)/(|\mathcal{D}||\mathcal{B}|-|\mathcal{B}|)$  instead of  $1/|\mathcal{B}|$ ; both factors are very similar when  $|\mathcal{B}| \ll |\mathcal{D}|$ .

## 5.2 Variance-Estimation from Mini-Batches

An unbiased estimator for the population *variances* (diagonal of matrix  $\Sigma(w)$ ) are the sample variances  $\hat{\Sigma}(w) \in \mathbb{R}^N$  of the individual gradient elements.<sup>2</sup> Again for an arbitrary i. i. d. dataset  $\mathcal{S}$  we get:

$$\begin{aligned} \hat{\Sigma}(w) &= \frac{1}{|\mathcal{S}|-1} \sum_{i=1}^{|\mathcal{S}|} \nabla \ell(w, d_i)^{\odot 2} - \frac{|\mathcal{S}|}{|\mathcal{S}|-1} \nabla L_{\mathcal{S}}(w)^{\odot 2} \\ &\approx \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \nabla \ell(w, d_i)^{\odot 2} - \nabla L_{\mathcal{S}}(w)^{\odot 2}, \end{aligned} \quad (90)$$

where  $\odot^2$  denotes the elementwise square. The estimator in the last row is not unbiased anymore, but for large  $|\mathcal{S}|$  both estimators are

<sup>2</sup> Note that  $\Sigma$  is a covariance *matrix*, and  $\hat{\Sigma}$  just a *vector* of the variances. This might cause confusion, but is also convenient as not to introduce yet another symbol.

nearly identical. Similarly, an unbiased estimator for  $\Lambda(w)$  is the sample variance  $\hat{\Lambda}(w)$ :

$$\begin{aligned}\hat{\Lambda}(w) &= \frac{1}{|\mathcal{S}|-1} \sum_{i=1}^{|\mathcal{S}|} \ell(w, d_i)^2 - \frac{|\mathcal{S}|}{|\mathcal{S}|-1} L_{\mathcal{S}}(w)^2 \\ &\approx \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \ell(w, d_i)^2 - L_{\mathcal{S}}(w)^2.\end{aligned}\tag{91}$$

Both estimators  $\hat{\Sigma}(w)$  and  $\hat{\Lambda}(w)$  require the sample *means* (first moments)  $L_{\mathcal{S}}(w)$  and  $\nabla L_{\mathcal{S}}(w)$ , as well as the statistics  $\sum_i \nabla \ell(w, d_i)^{\odot 2}$  and  $\sum_i \ell(w, d_i)^2$ .

### Implementation

Parts of this subsection are taken from [6] where the author is second-author (of three). The derivations of  $\hat{\Lambda}$  and  $\hat{\Sigma}$  for multi-layer perceptrons were originally done by the author, the ones for convolution filter of CNNs by L. Balles.

The sample means  $L_{\mathcal{S}}(w)$  and  $\nabla L_{\mathcal{S}}(w)$  are readily available during the optimization run. In common auto-differentiation frameworks, as the ones mentioned in Chapter 2, usually individual losses  $\ell(w, d_i)$  are accessible, such that  $\hat{\Lambda}(w)$  is straightforward to compute. An efficient implementation (in terms of speed and memory) of the sum of elementwise squares of the individual gradients  $\sum_i \nabla \ell(w, d_i)^{\odot 2}$  is a bit more tricky. The reason is that individual  $\nabla \ell(w, d_i)$  are usually not accessible in auto-differentiation frameworks because the sum over the mini-batch is performed implicitly via some matrix-matrix computations. Even if individual  $\nabla \ell(w, d_i)$  were accessible, summing them explicitly would not be a desirable approach since holding them all in memory is usually too expensive.

Consider a fully connected multi-layer perceptron (MLP) with weight matrices  $W_l$  of layer  $l$  and activations  $a_l$  of the preceding layer as in Section 2.2.<sup>3</sup> The gradients of the weights  $\nabla \ell_l$  can be computed recursively by backpropagation (Chapter 2, Eq. 33) as  $\nabla \ell_l = a_{l-1} \delta_l^{\top}$  (with  $\delta_l = (W_{l+1} \delta_{l+1}) \odot \partial_{a_l(z_l)}/\partial z_l$ ). The matrix-matrix multiplication  $a_{l-1} \delta_l^{\top}$  implicitly performs the sum over the mini-batch. The elementwise *squares* of the gradients can thus be computed as  $\nabla \ell_l^{\odot 2} = (a_{l-1})^{\odot 2} (\delta_l^{\top})^{\odot 2}$ , where again the summation over the mini-batch is performed *implicitly* by the matrix-matrix product. To see this, let us decompose the sum as:

$$\begin{aligned}[\nabla \ell_l^{\odot 2}]^{\alpha\beta} &= [(a_{l-1} \delta_l^{\top})^{\odot 2}]^{\alpha\beta} = \sum_i a_{l-1}^{\alpha i} a_{l-1}^{\alpha i} \delta_l^{\beta i} \delta_l^{\beta i} \\ &= \sum_i (a_{l-1}^{\alpha i})^2 (\delta_l^{\beta i})^2 = [(a_{l-1})^{\odot 2} (\delta_l^{\top})^{\odot 2}]^{\alpha\beta}\end{aligned}\tag{92}$$

[6] Balles, Mahseerci, and Hennig, "Automating Stochastic Optimization with Gradient Variance Estimates," 2017

<sup>3</sup> Reminder of sizes:  $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ , and  $b_l, z_l, a_l \in \mathbb{R}^{n_l \times |\mathcal{B}|}$ .

Backpropagation would thus include only one more line:

$$\delta_l = (W_{l+1}\delta_{l+1}) \odot \frac{\partial a_l(z_l)}{\partial z_l} \quad \text{residual} \quad (93a)$$

$$\nabla \ell_l = a_{l-1}\delta_l^\top \quad \text{gradient} \quad (93b)$$

$$\nabla \ell_l^{\odot 2} = (a_{l-1})^{\odot 2}(\delta_l^\top)^{\odot 2}. \quad \text{squared gradient} \quad (93c)$$

For biases, the squared gradient is  $\nabla \ell_l^{\odot 2} = \mathbb{1}_{1 \times |\mathcal{B}|}(\delta_l^\top)^{\odot 2}$ . The objects  $a_{l-1}$  and  $\delta_l$  can be re-used from the gradient computation. Roughly speaking, computing  $\hat{\Sigma}(w)$  during backpropagation adds one more matrix-matrix computation per layer of complexity  $n_{l-1}n_l|\mathcal{B}|$  to existing three of same cost (one from the forward pass, two from the backwards pass); a factor of 1.3. The actual cost is lower because of non-linearities or auxiliary actions like fetching data or updating the variables. An efficient way of implicitly computing the second moments for convolution filters of a CNN can be found in [6].

### Connections

The non-central second moments  $\mathbf{E}_{d \sim Q}[\nabla \ell(w, d)^{\odot 2}] = \nabla \mathcal{L}(w)^{\odot 2} + \text{diag}[\Sigma(w)]$  are quantities that appear in diagonal preconditioners like ADAM or RMSPROP (Chapter 2 § 2.4). There, the elements of the search direction are scaled inversely proportional to  $v_t^{\odot 1/2}$  (Eq. 65), where  $v_t$  is an exponential running average over the squared stochastic gradients  $\nabla \ell(w_t, d)^{\odot 2}$ . Neglecting the bias which occurs from moving the optimizer in  $w$ -space, one can say that  $v_t \approx \mathbf{E}_{d \sim Q}[\nabla \ell(w_t, d)^{\odot 2}]$  for  $|\mathcal{S}| = 1$ .<sup>4</sup> In principle  $v_t$  could thus be used to estimate  $\text{diag}[\Sigma]$  and incorporate it in gradient likelihoods as in Eq. 86. For general  $|\mathcal{S}| \geq 1$ , this would amount to a variance estimator of the form  $|\mathcal{S}|^{-1} \text{diag}[\Sigma(w_t)] \approx v_t - m_t^{\odot 2}$ , where  $m_t$ , as in Eq. 65, is a running average over mini-batch gradients  $\nabla L_{\mathcal{S}}(w_t)$ . The major difference between  $v_t$  and  $\hat{\Sigma}$  of Eq. 90 is the *local* versus *non-local* computation. Averaging over different locations  $w_t$  introduces a ‘memory’ of roughly a few hundred to thousand previous locations depending on the smoothing factor ( $\gamma \approx 0.999$  for ADAM decays to  $\approx 5\%$  after 3000 steps). This has two considerable effects: i) Averaging in  $w$ -space introduces a non-trivial bias, and ii) locally large variances get reduced and locally small variances get enlarged.

This has minor or major implications, depending on the task, and the benefits of either of the estimation methods, i. e., running averages or local estimators, or perhaps a combination of the two, can for sure be exploited. For instance, rough damping of SGD-steps might require less accurate variance estimates, but if decisions depend crucially on the gradient likelihood as in Eq. 86, locally unbiased estimators of  $\Sigma(w)$  might be inevitable.

[6] Balles, Mahseerci, and Hennig, “Automating Stochastic Optimization with Gradient Variance Estimates,” 2017

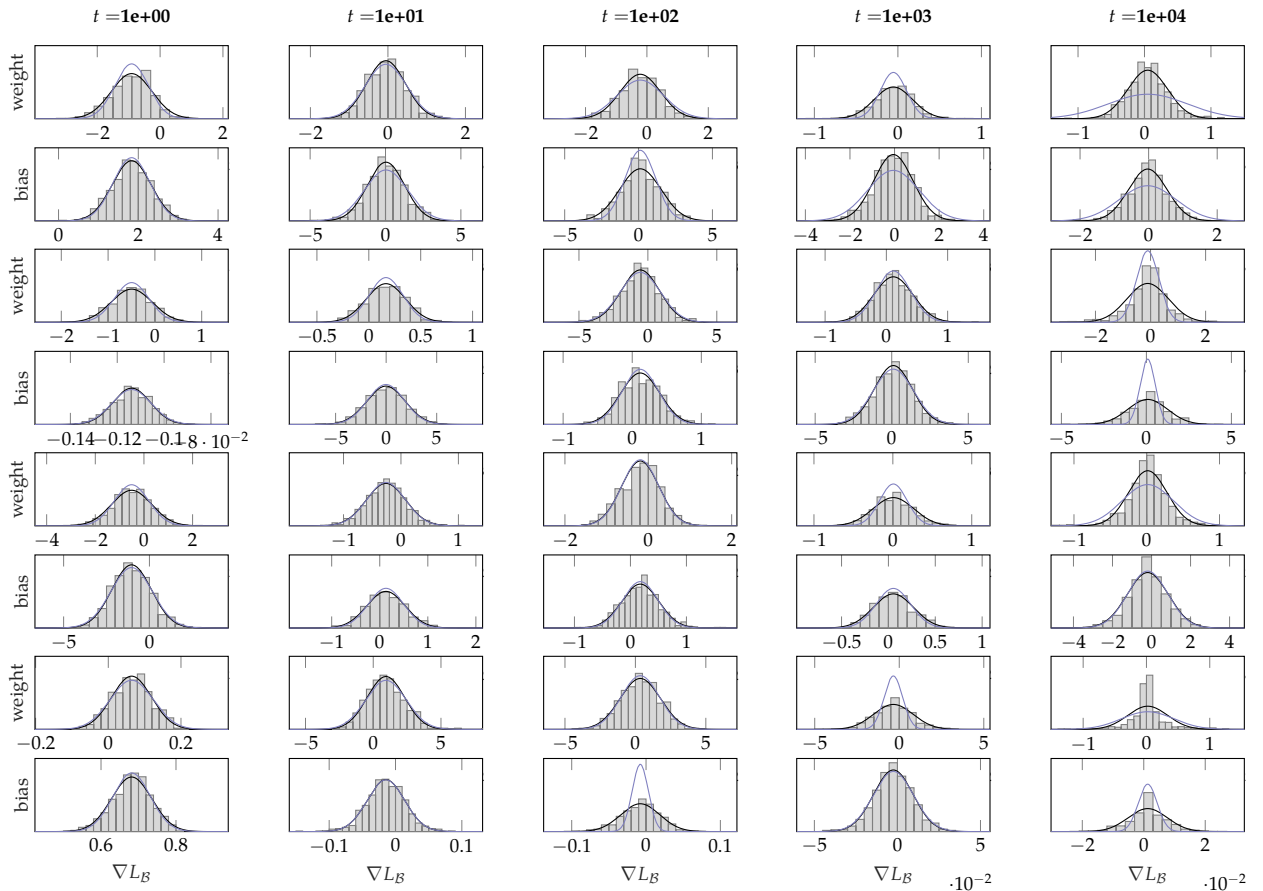
<sup>4</sup> For  $|\mathcal{S}| > 1$  the same holds, but for the expectation of mini-batch gradients over the distribution of mini-batches.



The following chapters (especially 6 and 7) show examples of algorithms which heavily rely on the local estimators  $\hat{\Lambda}$  and  $\hat{\Sigma}$  and do not perform well with running averages like  $v_t$ , possibly for the mentioned reasons.

### Empirical Study

Similar to Chapter 4 § 4.2 we conduct experiments on the empirical distribution of elements of mini-batch gradients and losses, and compare them to Gaussian fits. Figure 24 shows empirical distributions





() of gradient weights and biases (one per layer, layers from top to bottom) for mini-batch size  $|\mathcal{B}| = 100$ , in different stages of the optimization process. Columns from left to right: iteration number  $t = 1$  (initialization),  $10^1$ ,  $10^2$ ,  $10^3$ , and  $10^4$  (converged). In each box, two Gaussian fits are plotted: one using the sample mean and sample variance of all mini-batch gradients (—); and the other using the variance estimation  $\hat{\Sigma}$  of a *single* mini-batch (---), as it would be used by an algorithm. The network has 4-layers, is fully connected and trained to convergence with sGD on MNIST<sup>5</sup> and a well performing step size. Rows 1-2 show the input layer (random weights in row 1

Figure 24: Empirical distribution of mini-batch *gradients* for  $|\mathcal{B}| = 100$ . All plots: empirical distribution (). Gaussian fit using the sample mean and sample variance of all mini-batch gradients (—); same for the variance estimation  $\hat{\Sigma}$  of a *single* mini-batch (---) (further details in text).

<sup>5</sup> MNIST is a dataset for handwritten digit classification containing 60k training datapoints. It is introduced in more detail in Chapter 7 § 7.5.

and random biases in row 2); and the same for rows 3-4, 5-6, 7-8 but for hidden layers 1 and 2 and the output layer respectively. The Gaussian form holds astonishingly well over the whole optimization process, even for this (rather small) mini-batch size; and there are only minor differences in the variance estimate computed by all mini-batches and the one using only a single mini-batch.<sup>6</sup>

Figure 25 is structured like Figure 24 but shows histograms and Gauss fits for a *very* small mini-batch size  $|\mathcal{B}| = 10$ . The distribution of the initial gradients (leftmost column,  $t = 1$ ) is still well approximated by a Gaussian, but the fits get poorer the more the optimizer progresses. Additionally, the sample-variances  $\hat{\Sigma}$  often underestimate the true variances  $\text{diag}[\Sigma]$ . This is because gradients of the tail of the base-distribution (not the plotted histogram, but the distribution of individual gradients) are less likely represented in small mini-batches of only size 10. The effects can also be observed empirically (Chapter 7 § 7.5) where an algorithm (the probabilistic line search) that relies on the Gaussian assumption of Eq. 88 as well as on  $\hat{\Lambda}$  and  $\hat{\Sigma}$ , performs less well for  $|\mathcal{B}| = 10$  but performs well and reliably for larger  $|\mathcal{B}| > 10$ .

<sup>6</sup> The variance of the variance estimator  $\hat{\Sigma}$  also grows with decreasing mini-batch size.

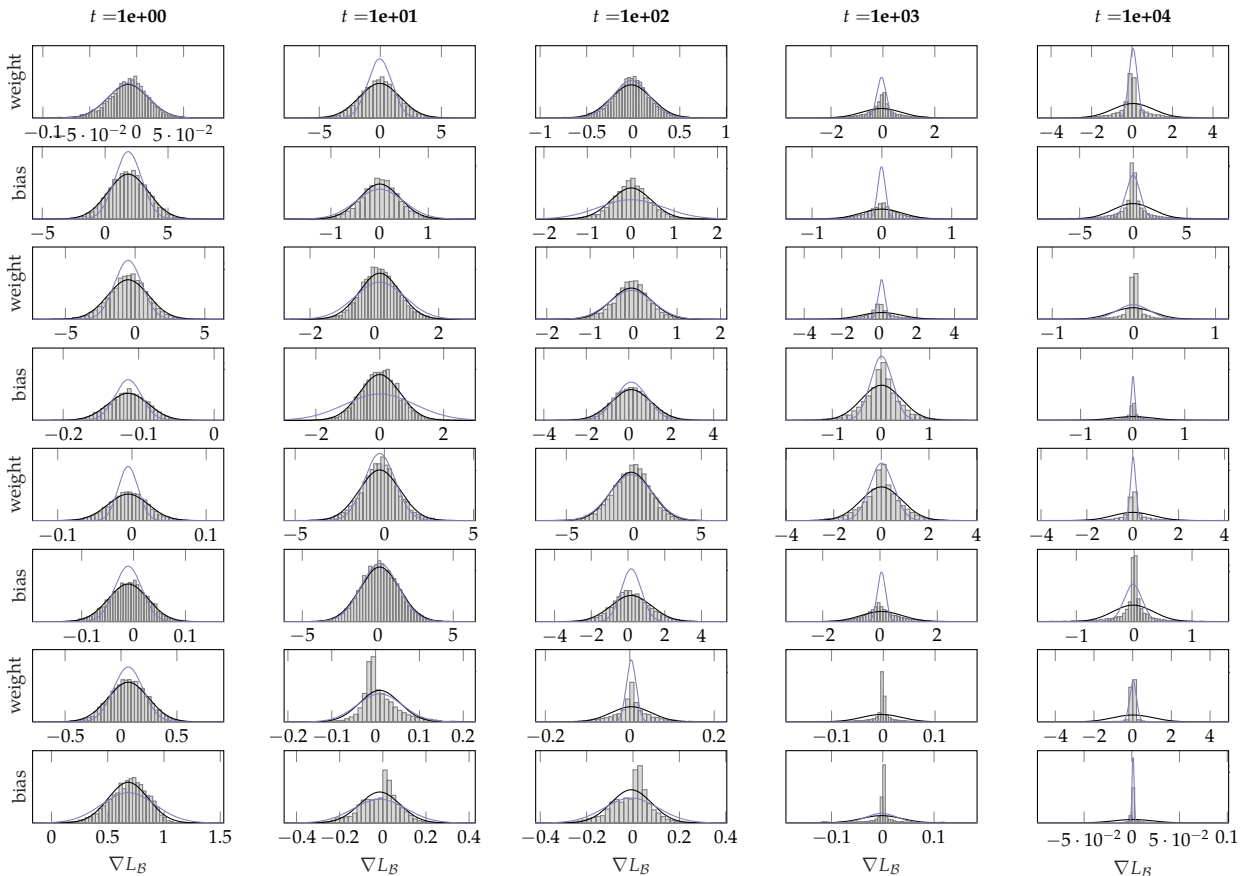


Figure 25: Same as Figure 24 but for a smaller mini-batch size  $|\mathcal{B}| = 10$ . The histograms are less well approximated, especially towards the end of the optimization process.

For completeness, Figure 26 shows the same histograms and plots for the *losses*  $L_B$  instead of gradients. The main difference is that the base-distribution (not shown), and thus the histogram, is lower bounded by zero, such that the Gauss distribution (which has support on the whole real line) tends to be a poorer fit if much mass is accumulated close to this boundary. Again, for a ‘large enough’ mini-batch size  $|\mathcal{B}| = 100$  (top row) the Gauss fits are satisfactory and the variance estimate  $\hat{\Lambda}$  (—) approximates the true variance (—) well enough; histograms for the smaller  $|\mathcal{B}| = 10$  (bottom row) are less Gaussian-like and  $\hat{\Lambda}$  (similar to  $\hat{\Sigma}$ ) has a higher variance itself, as well the tendency to underestimate  $\Lambda$ .

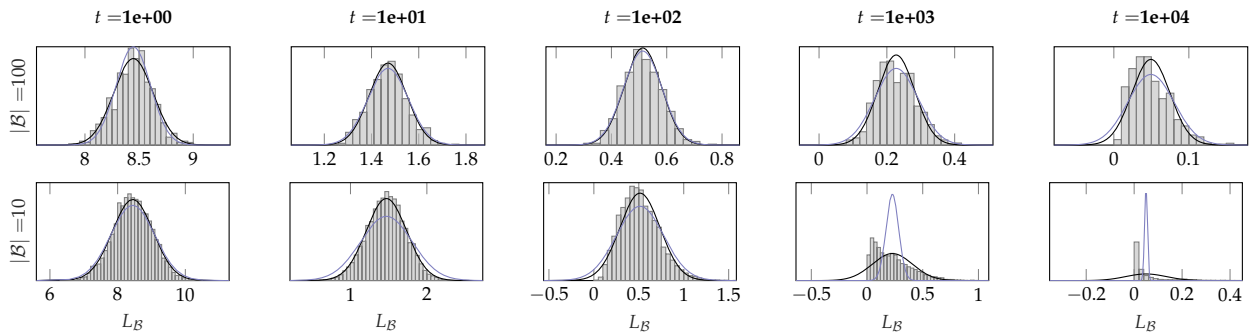


Figure 26: Same as Figure 24 but for *losses*  $L_B$ . Top row: mini-batch size  $|\mathcal{B}| = 100$ ; bottom row: mini-batch size  $|\mathcal{B}| = 10$ . Again the Gaussian fits are better for the larger mini-batch size.

All following sections will make use of the Gaussian assumption of Eq. 88 as well as the statistical estimators  $\hat{\Sigma}$  and (to a lesser degree)  $\hat{\Lambda}$  (Eqs. 90 and 91). It will be assumed that they are ‘good enough’ for our purposes, in the sense outlined above. In the future it will be helpful to incorporate additional checks, for example in the form of lightweight occasional statistical tests during or prior to the optimization run, if the Gaussian assumptions, as well as the quality of the estimators are indeed still fulfilled to a well enough degree (which depends on the given task). If this should not be the case, the mini-batch size could be increased, the noise estimation could be altered, the optimization method could be switched, or the optimizer could at least return a flag. This thesis will not feature these future works.



## Early-Stopping Without a Validation Set

---

**E**ARLY stopping is a widely used technique to prevent poor generalization performance when training an over-expressive model by means of gradient-based optimization. To find a good point to halt the optimizer, a common practice is to split the dataset into a training and a smaller *validation* set to obtain an ongoing estimate of the generalization performance. This chapter introduces an early-stopping criterion based on fast-to-compute local statistics of the computed gradients and removes the need for a held-out validation set. The experiments in Section 6.3 show that this is a viable approach, especially for smaller to mid-sized datasets, in the setting of least-squares and logistic regression, as well as neural networks. The text is mostly based on the article [88]. Section 6.1 motivates the problem, Section 6.2 introduces notation, model assumptions, and motivate the idea of the stopping criterion. Section 6.2.1 covers the more intuitive case of gradient descent and Section 6.2.2 extends to stochastic settings. Finally Section 6.3 shows experimental result on convex problems (quadratic, least-squares, binary logistic regression) as well as non-convex problems (multi-layer perceptrons, multinomial logistic regression).

[88] Mahsereci et al., *Early Stopping without a Validation Set*, 2017

### 6.1 Overfitting, Regularization and Early-Stopping

Since the *risk*  $\mathcal{L}(w)$  i. e., the expectation of the loss over the unknown data distribution as defined in Eq. 27, is virtually always unknown, a key question arising when minimizing the *empirical risk*  $L_{\mathcal{D}}(w)$  (Eq. 28), is how the performance of a model trained on a finite dataset  $\mathcal{D}$  generalizes to unseen data. Performance can be measured by the loss itself or other quantities, e.g., the mean accuracy in classification problems. Typically, to measure the generalization performance a finite *test set* is entirely withheld from the training procedure and the performance of the final model is evaluated on it. This test loss, however, is also only an estimator for  $\mathcal{L}$  with a finite stochastic error whose variance drops linearly with the test set size. If the used model is overly expressive, minimizing the empirical risk  $L_{\mathcal{D}}(w)$  exactly—or close to exactly—will usually result in poor test performance, since the model overfits to the training data. There is a range of measures that can be taken to mitigate this effect, some of which were already discussed in Chapter 2 § 2.2.2; textbooks like [13] give an overview over general concepts, chapter 7 of [48] gives a comprehensive summary targeted

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

[13] Bishop, *Pattern Recognition and Machine Learning*, 2006

at deep learning. Some widely used concepts are briefly repeated in the following paragraphs.

*Model selection* techniques choose a model among a hypothesis class which, under some measure, has the closest level of complexity to the given dataset. They alter the form of the loss function  $\ell_f$  (Eq. 28) or the mapping  $f_w$  over an outer optimization loop (first find a good  $\ell_f, f_w$ , then optimize  $L_{\mathcal{D}}$ ), such that the final optimization on  $L_{\mathcal{D}}$  is conducted on an adequately expressive model. This can—but does not need to—constrain the number of parameters of the model. In the case of deep neural networks the number of parameters can even significantly exceed the number of training examples [77] [125] [133] [55].

If the dataset is not sufficiently representative of the data distribution, an opposite, although not incompatible, approach is to artificially enrich it to match a complex model. *Data augmentation* artificially enlarges the training set by adding transformations/perturbations of the training data. This can range from injecting noise [124] [137] to carefully tuned contrast and colorspace augmentation [77].

Finally, a widely-used provision against overfitting is to add regularization terms to the objective function that penalize the parameter vector  $w$ , typically measured by the  $l_1$  or  $l_2$  norm [78]. These terms constrain the magnitude of  $w$ . They tend to drive individual parameters toward zero or, in the  $l^1$  case, enforce sparsity [13] [48]. In linear regression, these concepts are known as least-squares and LASSO regularization [134], respectively.

Despite these countermeasures, high-capacity models will often overfit in the course of the optimization process. While the loss on the training set decreases throughout the optimization procedure, the test loss saturates at some point and starts to increase again. This undesirable effect is usually countered by *early-stopping* the optimization process, meaning that for a *given* model, the optimizer is halted if a user-designed early-stopping criterion is met. This is complementary to the model and data design techniques mentioned above and does not undo eventual poor design choices of  $\ell$ . It merely ensures that the empirical risk  $L_{\mathcal{D}}$  of a given model is not minimized beyond the point of best generalization. In practice it is often even more accessible to ‘early-stop’ a high-capacity model for algorithmic purposes or because of restrictions to a specific model class, and thus preferred or even enforced by the model designer.

Arguably the gold-standard of early-stopping is to monitor the loss of a *validation set* [94] [111] [107]. For this, a portion of the training data is split off and its loss is used as an estimate of the generalization loss  $\mathcal{L}$ , leaving less effective training data to define the training loss  $L_{\mathcal{D}}$ . An ongoing estimate of this generalization performance is then tracked and the optimizer is halted when the generalization perfor-

[77] Krizhevsky, Sutskever, and Hinton, “Imagenet classification with deep convolutional neural networks,” 2012

[125] Simonyan and Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” 2014

[133] Szegedy et al., “Going deeper with convolutions,” 2015

[55] He et al., “Deep residual learning for image recognition,” 2016

[124] Sietsma and Dow, “Creating artificial neural networks that generalize,” 1991

[137] Vincent et al., “Extracting and composing robust features with denoising autoencoders,” 2008

[78] Krogh and Hertz, “A simple weight decay can improve generalization,” 1991

[13] Bishop, *Pattern Recognition and Machine Learning*, 2006

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

[134] Tibshirani, “Regression shrinkage and selection via the lasso,” 1996

[107] Prechelt, “Early Stopping — But When?” 2012

[94] Morgan and Bourlard, “Generalization and parameter estimation in feed-forward nets: Some experiments,” 1989

[111] Reed, “Pruning algorithms—a survey,” 1993

mance drops again. This procedure has many advantages, especially for very large datasets where splitting off a part has minor or no effect on the generalization performance of the learned model. Nevertheless, there are a few obvious drawbacks. Evaluating the model on the validation set in regular intervals can lead to computational overhead. More importantly, the choice of the *size* of the validation set poses a trade-off: A small validation set produces an estimator of  $\mathcal{L}$  with a large stochastic error, which can lead to a misguided stopping decision. Enlarging the validation set yields a more reliable estimate of generalization, but reduces the remaining amount of training data, depriving the model of potentially valuable information. This trade-off is not easily resolved, since it is influenced by properties of the data distribution (the variance  $\Lambda$  of Eq. 86a) and subject to practical considerations, e.g., redundancy in the dataset.

Recently [86] introduced an interpretation of (stochastic) gradient descent in the framework of variational inference. As a side effect, this motivated an early-stopping criterion based on the estimation of the marginal likelihood, which is done by tracking the change in entropy of the posterior distribution of  $w$ , induced by each optimization step. Since the method requires estimation of the Hessian diagonals, it comes with computational overhead comparable to the cost of a few additional gradient evaluations per step.

The following section motivates and derives a cheap and scalable early-stopping criterion which is solely based on local statistics of the computed gradients. In particular, it does not require a held-out validation set, thus enabling the optimizer to use *all* available training data.

## 6.2 When to Stop?—A Criterion Based on Gradient Statistics

The perhaps obvious but crucial observation at the heart of the criterion proposed below is that even the full, but finite, data-set is just a finite-variance sample from a population: As argued in Chapter 5 and by Eq. 86, the estimators  $L_{\mathcal{D}}$  and  $\nabla L_{\mathcal{D}}$  are approximately Gaussian samples around their expectations  $\mathcal{L}$  and  $\nabla \mathcal{L}$ , respectively. Figure 27 provides an illustrative, one-dimensional sketch. The top plot shows the marginal distribution of function values (Eq. 86a). The true, but usually unknown, optimization objective  $\mathcal{L}$ , is the mean of this distribution (—). The objective  $L_{\mathcal{D}}$ , which is optimized in practice and is fixed by the training set  $\mathcal{D}$ , defines *one* realization out of this distribution (---). Middle: same as top plot but for corresponding gradients defined by Eq. 86b. Mean  $\nabla \mathcal{L}$  (—)  $\pm 1 \Sigma^{\frac{1}{2}}$  (—), realization  $\nabla L_{\mathcal{D}}$  (---). Bottom: Same as middle plot; desired stopping regions (□).

In general, the minimizers of  $\mathcal{L}$  and  $L_{\mathcal{D}}$  need not be the same. Often, for a finite but large number of parameters  $w \in \mathbb{R}^N$ , the loss  $L_{\mathcal{D}}$  can be optimized to be very small. When this is the case the

[86] Maclaurin, Duvenaud, and Adams, *Early Stopping is Nonparametric Variational Inference*, 2015

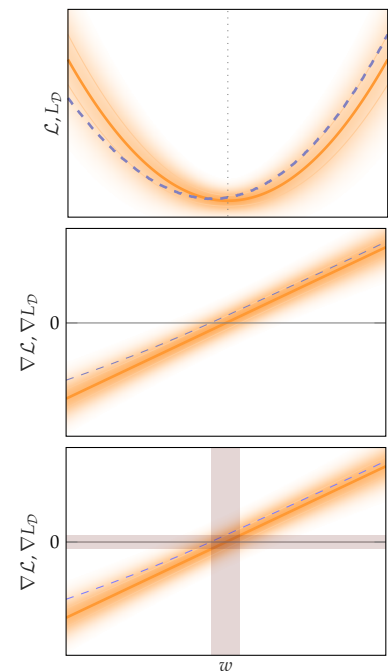


Figure 27: Sketch of early-stopping criterion. Top: marginal distribution of function values defined by Eq. 86a. Mean  $\mathcal{L}$  (—)  $\pm 1 \Lambda^{\frac{1}{2}}$  (—), pdf shaded. The full dataset defines *one* realization of this distribution ( $L_{\mathcal{D}}$  of Eq. 28/---). Middle: same as top plot but for corresponding gradients defined by Eq. 86b. Mean  $\nabla \mathcal{L}$  (—)  $\pm 1 \Sigma^{\frac{1}{2}}$  (—), realization  $\nabla L_{\mathcal{D}}$  (---). Bottom: Same as middle plot; desired stopping regions (□).

model tends to overfit to the training data and thus performs poorly on newly generated (test) data  $\mathcal{T} \sim Q$  with  $\mathcal{T} \cap \mathcal{D} = \emptyset$ . To prevent this overfitting-effect, the optimization process is stopped early. The idea behind this is that variations between training examples of the same class are mostly learned at the very end of the optimization process where the weights  $w$  are fine-tuned. In practice, the true minimum of  $\mathcal{L}$  is unknown, however, the approximate errors of the estimators  $L_{\mathcal{D}}$  and  $\nabla L_{\mathcal{D}}$  are accessible at every position  $w$ . Local estimators for the diagonal of  $\Sigma(w)$  can be computed efficiently even for very high dimensional optimization problems (Chapter 5, § 5.2). The variance estimator of the gradient distribution will be again, consistent with Chapter 5, denoted as  $\hat{\Sigma}(w) \approx \mathbf{var}_{d \sim Q} [\nabla \ell(w, d)]$  with  $\hat{\Sigma}(w) = 1/(|\mathcal{S}|-1) \sum_{d \in \mathcal{S}} (\nabla \ell(w, d) - \nabla L_{\mathcal{S}}(w))^{\odot 2}$ , where  $\odot^2$  denotes the element-wise square and  $\mathcal{S}$  is either the full dataset  $\mathcal{D}$  or a mini-batch  $\mathcal{B}$ .

Since the minimizers of  $\mathcal{L}$  and  $L_{\mathcal{D}}$  are not generally identical, also their gradients will cross zero at different locations  $w$ . The middle plot of Figure 27 illustrates this behavior. Similar to the left plot, it shows a marginal distribution, but this time over gradients (Eq. 86b). The true gradient  $\nabla \mathcal{L}$  is the mean of this distribution (—). The *one* realization  $\nabla L_{\mathcal{D}}$  (---) defined by the dataset  $\mathcal{D}$  corresponds to  $L_{\mathcal{D}}$ . Ideally, the optimizer should stop in an area in  $w$ -space where possible minima are likely to occur if different datasets of same size were samples from  $Q$ . In the sketch, this is encoded as the *vertical* red shaded area in the bottom plot (▭). It is the area around the minimizer of  $\mathcal{L}$  where  $\nabla \mathcal{L} \pm 1$  standard deviation encloses zero.

Since  $\nabla \mathcal{L}$  is unknown, however, this criterion is hard to use in practice, and must be turned into a statement about  $\nabla L_{\mathcal{D}}$ . A similar criterion that captures this desiderata in essence is to stop when the collected gradients  $\nabla L_{\mathcal{D}}$  are becoming consistently very small in comparison to the error  $\Sigma/|\mathcal{D}|$ . This is shown as *horizontal* red shaded area (▭). Close enough to the minima of  $L_{\mathcal{D}}$  and  $\mathcal{L}$ , the two criteria roughly coincide (intersection of red vertical and horizontal shaded areas). A measure for this is the probability:

$$p(\nabla L_{\mathcal{D}}(w) | \nabla \mathcal{L}(w) = 0) = \mathcal{N} \left( \nabla L_{\mathcal{D}}(w); 0, \frac{\Sigma(w)}{|\mathcal{D}|} \right), \quad (94)$$

of observing  $\nabla L_{\mathcal{D}}(w)$  with covariance  $\Sigma(w)$ , were it generated by a true zero gradient  $\nabla \mathcal{L}(w) = 0$ .<sup>1</sup> If gradients  $\nabla L_{\mathcal{D}}$  are becoming too small (stepping into the horizontal red shaded area), the magnitude of the gradients can mostly be attributed to noise that represents the finiteness of the dataset, and not an informative gradient direction; then the optimizer should stop. Using these assumptions, the next section derives a stopping criterion for the gradient decent algorithm which then can be extended to stochastic gradient descent as well.

<sup>1</sup>This can be seen as the evidence of a model for  $\nabla \mathcal{L}(w)$  and  $\nabla L_{\mathcal{D}}(w)$ , described by, both, a prior  $p(\nabla \mathcal{L}(w)) = \delta(\nabla \mathcal{L}(w))$ , and likelihood as in Eq. 94. Then  $p(\nabla L_{\mathcal{D}}(w)) = \int p(\nabla L_{\mathcal{D}}(w) | \nabla \mathcal{L}(w)) p(\nabla \mathcal{L}(w)) d\nabla \mathcal{L}(w)$ . We will thus also use the term ‘evidence-based’ stopping. In principal more general models can be formulated, which lead to a richer class of stopping criteria.



### 6.2.1 Early-Stopping Criterion for Gradient Descent

When using gradient descent, the whole dataset is used to compute the gradient  $\nabla L_{\mathcal{D}}$  in each iteration. Still this gradient estimator has an error in comparison to the true gradient  $\nabla \mathcal{L}$ , which is encoded in the covariance matrix  $\Sigma$ . In practice  $\Sigma$  is unknown, the variance estimator  $\hat{\Sigma}$  described in Chapter 5 however is always accessible. Thus, at every position  $w$  an approximation to  $p(\nabla L_{\mathcal{D}}(w) | \nabla \mathcal{L}(w) = 0)$  of Eq. 94 is (using the ‘evidence’-notation  $p(\nabla L_{\mathcal{D}}(w))$  for notational convenience):

$$p(\nabla L_{\mathcal{D}}(w)) \approx \prod_{n=1}^N \mathcal{N} \left( \nabla L_{\mathcal{D}}^n(w); 0, \frac{\hat{\Sigma}_n(w)}{|\mathcal{D}|} \right). \quad (95)$$

Though being a simplification, this allows for fast and scalable computations since dimensions are treated independent of each other. To derive an early-stopping criterion based only on  $\nabla L_{\mathcal{D}}$  we borrow the idea of the previous section that the optimizer should halt when gradients relatively become so small that they are indistinguishable from noise. Specifically: stop when

$$\log p(\nabla L_{\mathcal{D}}(w)) - \mathbf{E}_{\nabla L_{\mathcal{D}}(w) \sim p} [\log p(\nabla L_{\mathcal{D}}(w))] > 0. \quad (96)$$

Here  $\mathbf{E}[\cdot]$  is the expectation operator. According to Eq. 96, the optimizer stops when the logarithmic evidence is larger than its expected value, roughly meaning that more/enough gradient samples  $\nabla L_{\mathcal{D}}$  lie inside of the expected range.<sup>2</sup> In particular, combining Eq. 95 with Eq. 96 and normalizing with the dimension  $N$  of the objective, yields

$$\frac{2}{N} \left[ \log p(\nabla L_{\mathcal{D}}(w)) - \mathbf{E}_{\nabla L_{\mathcal{D}}(w) \sim p} [\log p(\nabla L_{\mathcal{D}}(w))] \right] = 1 - \frac{|\mathcal{D}|}{N} \sum_{n=1}^N \left[ \frac{(\nabla L_{\mathcal{D}}^n(w))^2}{\hat{\Sigma}_n(w)} \right] > 0. \quad (97)$$

This criterion, hereafter called **EB-criterion**, for ‘evidence-based’, is intuitive: If all gradient elements lay at exactly one standard deviation distance to zero, then  $\sum_n (\nabla L_{\mathcal{D}}^n)^2 / \hat{\Sigma}_n = \sum_n \hat{\Sigma}_n / |\mathcal{D}| \cdot \hat{\Sigma}_n = N/|\mathcal{D}|$ ; thus the left-hand side of Eq. 97 would become zero and the optimizer would stop.

We note on the side that Eq. 97 defines a mean criterion over all elements of the parameter vector  $w$ . This is sensible if all dimensions converge in roughly the same time scale such that weighing the fractions  $f_n := |\mathcal{D}| \cdot (\nabla L_{\mathcal{D}}^n)^2 / \hat{\Sigma}_n$  equally is justified. In other words, overfitting is a global phenomenon on the weights, not a local one. If optimization problems deal with parameters that converge at different speeds, like for example different layers of neural networks, or biases and weights inside one layer, it might be appropriate to compute one stopping criterion per subset of parameters which are roughly having similar timescales. In Section 6.3.4 we will use this slight variation

<sup>2</sup> Besides the model-evidence interpretation, Eq. 96 can also be found in Hotelling’s one-sample t-square test, with the hypothesis that the mean of the Gaussian is zero and the covariance is given. In practice we do not have access to the covariance thus we estimate it. Also note that correspondences to hypothesis tests remain rather vague here in the same sense as monitoring a validation loss is not a fully fledged hypothesis test either.

of Eq. 97 for experiments on logistic regressors as well as multi-layer perceptron.

### 6.2.2 Stochastic Gradients and Mini-batching

It is straightforward to extend the stopping criterion of Eq. 97 to stochastic gradient descent (sgd), where the estimator for  $\nabla L_{\mathcal{D}}$  is replaced with an even more uncertain  $\nabla L_{\mathcal{B}}$  by sub-sampling the training dataset at each iteration. The local gradient generation is

$$\nabla L_{\mathcal{B}}(w) = \nabla L_{\mathcal{D}(w)} + \eta = \nabla \mathcal{L}(w) + v \quad \text{with} \quad \eta \sim \mathcal{N}(0, \Sigma_{\text{obs}}(w)), v \sim \mathcal{N}\left(0, \frac{\Sigma(w)}{|\mathcal{D}|} + \Sigma_{\text{obs}}(w)\right). \quad (98)$$

Combining this with Eq. 86b yields  $\Sigma/|\mathcal{D}| + \Sigma_{\text{obs}} = \Sigma/|\mathcal{B}|$ . Thus  $\Sigma_{\text{obs}} = \frac{|\mathcal{D}|-|\mathcal{B}|}{|\mathcal{B}||\mathcal{D}|}\Sigma$ . Equivalently to Eq. 94, 95 and 97, this results in an early-stopping criterion for stochastic gradient descent:

$$\frac{2}{N} \left[ \log p(\nabla L_{\mathcal{B}}(w)) - \mathbf{E}_{\nabla L_{\mathcal{B}}(w) \sim p} [\log p(\nabla L_{\mathcal{B}}(w))] \right] = 1 - \frac{|\mathcal{B}|}{N} \sum_{n=1}^N \left[ \frac{(\nabla L_{\mathcal{B}}^n(w))^2}{\hat{\Sigma}_n(w)} \right] > 0. \quad (99)$$

#### Implementation

Computing the stopping criterion is straight-forward, given that the variance estimate  $\hat{\Sigma}(w)$  introduced in Chapter 5 is available. In this case, it amounts to an element-wise division of the squared gradient by the variance, followed by an aggregation over all dimensions, i. e.,  $\text{sum} [\nabla L_{\mathcal{B}}(w)^{\odot 2} \oslash \hat{\Sigma}(w)]$ .

#### Smoothing the Criterion

Each iteration  $t$  has access to only *one* gradient sample (from dataset or mini-batch) to compute the  $\text{EB}$ -criterion locally, i. e., for a given  $w_t$ . Additionally the estimator  $\hat{\Sigma}(w_t)$  for the gradient variances is inexact, since it, too, is computed from a finite amount of examples only. Hence this local  $\text{EB}$ -criterion itself is noise corrupted as well. An efficient and practical way to reduce this noise is to average the criterion over many iterations. Hence, many successive iterations need to ‘agree’ on a meaningful stopping point. Note that there is a difference in between averaging the gradient squares and variances separately before computing their fraction, or to average their fraction. Apart from reducing the variance on the criterion, this is also beneficial if the optimizer is passing regions of low gradient/ unstable fix-points like stationary points of possibly non-convex loss function. With this in mind, the smoothing factor for and exponential running average should be as large as possible but still small enough not to bias the stopping point all too much. All real-world experiment below used a smoothing factor of  $\gamma = 0.999$  which is fully decayed between 1-3k

steps. This number is very small in comparison to the total number of optimization steps in the conducted experiments, and does not move the stopping decision visibly to the eye. Only the artificial toy problems (Section 6.3.1 and 6.3.2) perform a very small number of steps ( $\sim 100$ -1000) and consequently used no smoothing.

### 6.3 Experiments

Strictly speaking, the EB-criterion tests for stationary points only and not just on minimizers of  $\mathcal{L}$ . Thus, for proof of concept experiments, we test the EB-criterion first on *convex* problems where the only stationary point is also the minimizer (quadratics, linear least squares, logistic regression), and later on *non-convex* problems (multi-layer perceptrons, multinomial logistic regression) where saddle points might occur [26], on a number of standard classification and regression datasets. For illustration and controlled environments, Sections 6.3.1 and 6.3.2 show a least-squares toy problem and ill-conditioned large synthetic quadratic problems. Sections 6.3.3 and 6.3.4 deal with the more realistic setting of logistic regression on the Wisconsin Breast Cancer Dataset (WDBC) [140] and a multi-layer perceptron on the handwritten digits dataset MNIST [82]. Section 6.3.5 contains experiments for multinomial logistic regression, as well as for a shallow neural network on the SECTOR dataset [22]; the SECTOR dataset complements MNIST and WDBC, in the sense that it has a much less favorable feature-to-datapoint ratio ( $\sim 9$ ); increasing the gains on the generalization performance, when all available training data can be used. In general, we did not use overly large datasets where overfitting is a less prominent issue and also splitting off of a validation set has minor effects on the absolute value of the best test loss. We will also see that models trained on mid-sized but redundant datasets like MNIST do gain little when folding the validation set into the training, while models trained on datasets that represent the data distribution  $Q$  to a lesser degree, like e.g. WDBC, benefit a lot.

#### 6.3.1 Linear Least-Squares as Toy Problem

An illustrative example is a toy regression problem on artificial data, generated from a one-dimensional linear function  $\tilde{f}(x)$ ,  $x \in \mathbb{R}$  with additive uniform Gaussian noise, i. e., datapoints  $y$  are random according to  $y_i = \tilde{f}(x_i) + \epsilon$ , with  $\epsilon \sim \mathcal{N}$ . This simple setup allows illustration of the model fitted at various stages of the optimization process and also provides the *true* generalization performance, since we can generate large amounts of test data. We use a largely over-parametrized 50-dimensional linear regression model  $f_w(x) =$

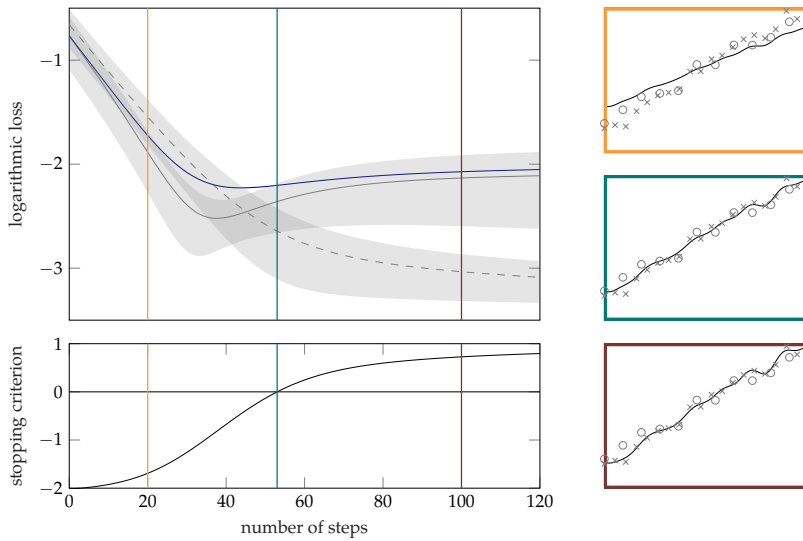
[26] Dauphin et al., “Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization,” 2014

[140] Wolberg, Street, and Mangasarian, *UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set*, 2011

[82] LeCun et al., “Gradient-based learning applied to document recognition,” 1998

[22] Chang and Lin, *LIBSVM: A library for support vector machines*, 2011

$w^T\phi(x)$  which contain the features of the ground truth (bias and linear) and additional periodic features with varying frequency: The features  $\phi(x) = [1, x, \sin(a_1x), \cos(a_1x), \dots, \sin(a_px), \cos(a_px)]^T$  with  $p = 24$ , and  $a_p$  positive integers, by construction define a massively over-parametrized model for the true function and is thus prone to overfitting. The model is fit by minimizing the squared error, i.e.  $\ell(f_w(x), y) = \frac{1}{2}(y - f_w(x))^2$ . 20 samples are used for training with full batch gradient descent (no subsampling of datapoints) and about 10 samples for validation. The results are shown in Figure 28; both, validation loss, and the EB-criterion find an acceptable point to stop the optimization procedure, thus preventing overfitting.<sup>3</sup>



<sup>3</sup>In this illustrative example, the EB-criterion was also evaluated on the training set which does *not* include the validation set. This has two diagnostic advantages: i) both criteria, validation loss and EB-criterion, are evaluated on the same optimization path and ergo on the same fits  $f_w$ , which would not be the case if the validation set was folded in, and ii) the top left plot is less cluttered since we would need two more lines (a second train and test loss curve) the former with shaded areas. Of course a main advantage of the EB-criterion is that one can fold in the validation set which will also be done with the real world datasets.

Figure 28: Linear least-squares toy problem. Top left: logarithmic losses vs. number of optimization steps: test/train/validation (—/---/—); shaded areas indicate two standard deviations  $\pm 2\sqrt{\hat{\Lambda}/|S|}$  of the loss estimates computed during the optimization (Eq. 91). Bottom left: evolution of the EB-criterion (—). Induced stopping point vertical (—). For the steps marked with color-coded vertical bars, the model fit  $f_w$  (—) is illustrated on the right column: training/validation data ( $\times/\circ$ ). Sub-optimal fit (—), to the training data; fit, when the EB-criterion indicates stopping (—);  $f_w$  has already overfitted to the training data (—).

### 6.3.2 Synthetic Large-Scale Quadratic Problem

Synthetic quadratic optimization problems offer a controlled environment to test on functions with ill-conditioned Hessians (ravines, long valleys, gradient elements spanning magnitudes et cetera) as they occur in real world applications [24]. Construct  $\mathcal{L}(w) = \frac{1}{2}(w - w^*)^T B(w - w^*)$ , where  $B \in \mathbb{R}^{N \times N}$  is a positive definite matrix and  $w^* \in \mathbb{R}^N$  is the global minimizer of  $\mathcal{L}(w)$ ; the gradient is  $\nabla \mathcal{L} = B(w - w^*)$ . In this controlled environment it is possible to design the curvature of  $\mathcal{L}$  explicitly and the EB-criterion can be tested on different configurations of eigen-spectra, for example uniform, exponential, or structured (a few large, many small eigenvalues). The matrix  $B$  is build by defining a diagonal matrix  $\Gamma \in \mathbb{R}^{N \times N}$  which contains the eigenvalues on its diagonal, and a random rotation  $R \in \mathbb{R}^{N \times N}$  that is drawn from the Haar-measure on the  $N$ -dimensional uni-sphere [32]; then  $B := R\Gamma R^T$ . The ‘empirical’ loss  $L_{\mathcal{D}}(w)$  can be artificially defined

[24] Chaudhari et al., “Entropy-SGD: Biasing Gradient Descent Into Wide Valleys,” 2016

[32] Diaconis and Shahshahani, “The subgroup algorithm for generating uniform random variables,” 1987

by moving the true minimizer  $w^*$  by a Gaussian random variable  $\zeta_{\mathcal{D}}$ , such that  $L_{\mathcal{D}}(w) = \frac{1}{2}(w - w^* + \zeta_{\mathcal{D}})^{\top} B(w - w^* + \zeta_{\mathcal{D}})$  with  $\zeta_{\mathcal{D}} \sim \mathcal{N}(0, \Lambda)$ . Thus  $\nabla L_{\mathcal{D}} = \nabla \mathcal{L} + B\zeta_{\mathcal{D}}$  is distributed according to  $B\zeta_{\mathcal{D}} \sim \mathcal{N}(0, B\Lambda B^{\top})$ , and we define  $\hat{\Sigma}/|\mathcal{D}| := \text{diag}(B\Lambda B^{\top})$ . For experiments we chose  $N = 10^3$  as input dimension and zero ( $w^* = 0$ ) as the true minimizer of  $\mathcal{L}$ . Figure 29 shows results for three different types of eigen-spectra. The  $\text{EB}$ -criterion performs well across the different

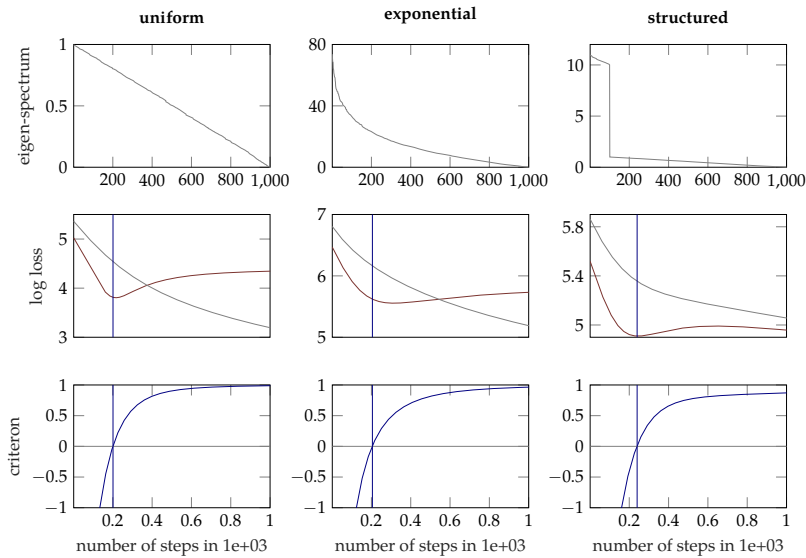


Figure 29: Synthetic quadratic problem for three different structures of eigen-spectra. Columns from left to right: uniform, exponential, structured. Middle row: logarithmic (exact) test loss (—) and train loss (—). Bottom row: evolution of the  $\text{EB}$ -criterion, inducing a stopping decision indicated by the vertical bar (—).

types of partially ill-conditioned problems and induces meaningful stopping decisions; this worked well for different noise levels  $\Lambda$  (Figure 29 shows  $\Lambda = 10 \cdot I$  as well as random seeds; note that the noise covariance  $B\Lambda B^{\top}$  of the gradient is dense, but the algorithm just has access to its diagonal as it is the case in real world applications).

### The $\text{EB}$ -criterion as Buffer-Region

The experiments give additional insight on a geometrical interpretation of the  $\text{EB}$ -criterion, which might also explain the slightly conservative stopping decision for the logistic regressor on WDBC (Figure 32 in subsequent section) and full batch  $\text{GD}$  on MNIST (Figure 34, column 1). Eq. 96 has no directional information to where the true minimizer  $w^*$  of the risk lies as seen from the current position of the optimizer. This is of course fundamentally hard to know, and early-stopping would be obsolete in these cases anyway. Thus Eq. 96 is based on variances only (in the form of  $\hat{\Sigma}$ ), defining a buffer-region/ a ‘no-go’-area around the minimizer  $w_{\mathcal{D}}^*$  of the *empirical* risk, where statistics of gradients tell us we should be careful to trust them. Usually, early-stopping is based on the assumption that a continuation of training that continuously decreases the training loss, will lead away again from weights that

generalize well. For the  $\epsilon_B$ -criterion this means that the optimizer, with a deterministic path, for a given mini-batch-sequence and starting point, will enter the buffer-region at a point which is generalizing better than subsequent locations of the optimizer (and in particular of  $w_{\mathcal{D}}^*$ ) inside this buffer-region. This is usually well justified, primarily because otherwise early-stopping would not be a viable concept in the first place; and second because overfitting is usually associated with ‘too large’ weights (weights are initialized small; and regularizers that pull weights to zero are often a good idea). On the way from small weights (underfitting) to too large weights (overfitting), optimizers usually pass a better point with weights of intermediate size.

If the point where the optimizer enters the buffer region has a worse generalization performance than subsequent iterates inside the buffer-region, again especially as  $w_{\mathcal{D}}^*$ , the  $\epsilon_B$ -criterion will still stop and potentially underfit slightly. We can artificially construct this setup by initializing the optimizer with weights that lead to an optimization path that does not lead to *any* overfitting; this is depicted in Figure 31. The setup is identical to the one in Figure 29 ( $B, w^*$  as well as  $\zeta_{\mathcal{D}}$  and  $w_{\mathcal{D}}^*$  are identical); the *only* difference is the initialization of the weights  $w_0$  for the optimization process. Since—with this initialization—the lowest point of  $\mathcal{L}$  that can be reached by minimizing  $L_{\mathcal{D}}$  is  $w_{\mathcal{D}}^*$ , *any* early-stopping decision will lead to underfitting. In Figure 31 the exact test loss flattens out and does not increase again for all three configurations. Figure 30 illustrates these two scenarios in a 2D-sketch: Contours of the optimizer’s objective  $L_{\mathcal{D}}(w)$  (—); contours of the true loss  $\mathcal{L}(w)$  not shown. Their minimizers  $w^*$  and  $w_{\mathcal{D}}^*$  are marked as crosses (×). Also shown are two optimization paths: i) An optimizer that passes by weights of better generalization performance than  $w_{\mathcal{D}}^*$  (—+—), and ii) an optimizer that than can not overfit (—+—), since weights were initialized such that, out of all weights  $w_t$  produced by the optimizer,  $w_{\mathcal{D}}^*$  yields best generalization performance. Both optimizers are stopped by the  $\epsilon_B$ -criterion when they enter the buffer-region (▭) around the minimizer  $w_{\mathcal{D}}^*$  of the empirical loss  $L_{\mathcal{D}}$ , resulting in a good and a too conservative stopping decision respectively for i) and ii).

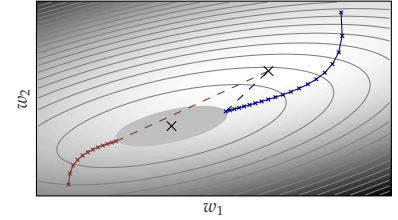


Figure 30: Illustration of buffer-region induced by the  $\epsilon_B$ -criterion. Contours of the optimizer’s objective  $L_{\mathcal{D}}(w)$  (—). Contours of the true loss  $\mathcal{L}(w)$  not shown. Their minimizers  $w^*$  and  $w_{\mathcal{D}}^*$  are marked as crosses (×). Buffer-region of the  $\epsilon_B$ -criterion (▭). Path of optimizer i) (—+—) and ii) (—+—).

### 6.3.3 Logistic Regression on WDBC

Next, we apply the  $\epsilon_B$ -criterion to logistic regression on the Wisconsin Breast Cancer dataset. The task is to classify cell nuclei, described by features such as radius, area, symmetry, et cetera, as either malignant or benign. We conduct a second-order polynomial expansion of the original 30 features, i. e., features of the form  $x_i x_j$ , resulting in 496 effective features. Of the 569 instances in the dataset, we withhold 369, a relatively large share, for testing purposes in order to get a

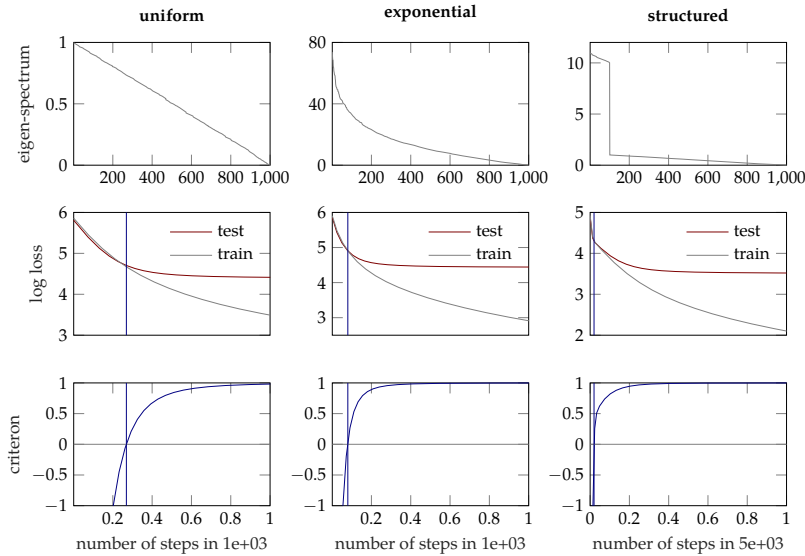


Figure 31: Synthetic quadratic problem for three different structures of eigen-spectra. Plot and experimental setup as in Figure 29. Weights are initialized such that the path of the optimizer can not yield parameters  $w_t$  which lead to a model  $f_w$  that overfits. The exact test loss (—) flattens out, but does not increase again; the EB-criterion still induces a (in this case conservative) stopping decision.

reliable estimate of the generalization performance. The remaining 200 instances are available for training the classifier. We perform two training runs: one with early-stopping based on a validation set of 60 instances (reducing the training set to 140 instances) and one using the full training set and early-stopping with the EB-criterion derived in Section 6.2.1.

If parameters converge at different speeds during the optimization, as indicated in Section 6.2.1, we compute the criterion separately for different subgroups of parameters. Generally, if we split the parameters into  $K$  disjoint subgroups  $S_k \subset \{1, \dots, N\}$ , and denote  $N_k = |S_k|$ , the criterion reads

$$\frac{1}{K} \sum_{k=1}^K \left( 1 - \frac{|D|}{N_k} \sum_{n \in S_k} \left[ \frac{(\nabla L_D^n)^2}{\hat{\Sigma}_n} \right] \right) > 0. \quad (100)$$

This means that every subgroup has one weighted ‘vote’ for stopping. For logistic regression, we treat the weight vector and the bias parameter of the logistic regressor as separate subgroups, since they tend to have different magnitudes. Figure 32 shows results. The effect of the additional training data is clearly visible, resulting in lower test losses throughout the optimization process. In this scarce data setting, the validation loss (top plot, —), computed on a set of only 60 instances, is clearly misleading. It decreases throughout the optimization process and thus fails to find a suitable stopping point. The bottom plot shows the evolution of the EB-criterion (—) which does induce a (possibly slightly conservative) stopping decision with a lower or equal test loss than any test loss attainable when withholding a validation set.

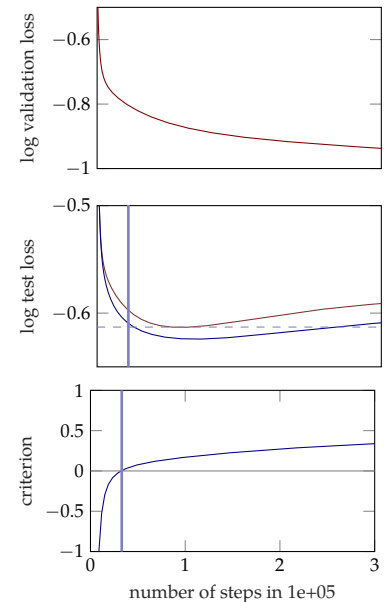


Figure 32: Logistic regression on the Wisconsin Breast Cancer dataset. Results for the two variants are color-coded: Validation set-based early-stopping (—), EB-criterion (—). Middle: test loss versus the number of optimization steps for both methods. Top: Validation loss. Bottom: Evolution of the EB-criterion; induced stopping decision as vertical bar.



## 6.3.4 Multi-Layer Perceptron on MNIST

For a non-convex optimization problem, where also saddle points might occur, we train a multi-layer perceptron (MLP) on the well-studied problem of hand-written digit classification with the MNIST dataset ( $28 \times 28$  gray-scale images). We use an MLP with five hidden layers with 2500, 2000, 1500, 1000 and 500 units, respectively, ReLU activation, and a standard cross-entropy loss for the 10 outputs with soft-max activation ( $\sim 12$  million trainable parameters). Analogously to Section 6.3.3, each weight matrix and each bias vector of the network is treated as a separate subgroup. The MNIST dataset contains 60k training images, which we split into 40k-10k-10k for train, test and validation set.

The results for full-batch gradient descent are shown in Figure 33, and sgd runs with mini-batch size 128 and three different learning rates in Figure 34. The loss of the relatively large validation set (10k images) in combination with a quite homogeneous dataset yields accurate estimates of the generalization performance. Consequently, the stopping points are very close to the points of minimal test loss. The folded training set and validation set leads to only slightly lower test losses since the data of the validation set does not contain much additional information which is not already present in the training set. Since the strength of the EB-criterion is to use the additional training data as well as the fact that also validation losses are only inexact guesses of the generalization loss, both of these points thus favor the validation set based stopping criterion. Still, for all three sgd-runs in Figure 34, the EB-criterion performs as good as or better than the validation set induced method. For gradient descent (full training set in each iteration, Figure 33), the EB-criterion performs reasonably well, however, and very similarly to the gradient descent runs on the logistic regression on WDBC in Figure 32, chooses to stop a bit too early. The difference in loss is not very much (test loss red:  $10^{-1.04}$ , blue  $10^{-0.92}$ ), but it again shows that validation loss induces stopping can be preferred in settings where data count is not the main concern.

## 6.3.5 Logistic Regression and Shallow-Net on SECTOR

Finally, we train a multinomial logistic regressor and a shallow fully-connected neural network on the SECTOR dataset [22]. It contains 6412 training and 3207 test datapoints with 55 197 features each, thus having a less favorable feature-to-datapoint ratio than for example MNIST (784 features vs. 60 000 datapoints). The features are extracted from web-pages of companies and the classes describe 105 different industry sectors. The shallow network has one hidden layer with 200

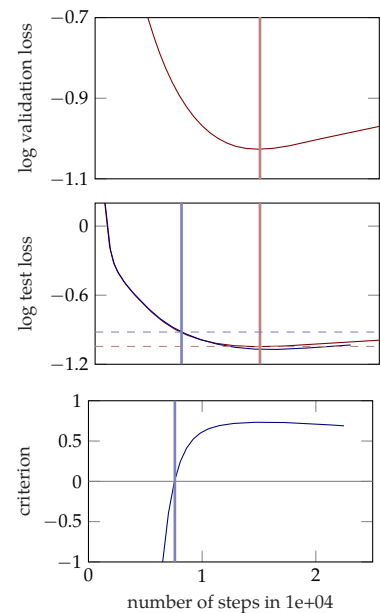


Figure 33: Multi-layer perceptron on MNIST trained with full batch gradient descent. Plot and colors as in Figure 32.

[22] Chang and Lin, *LIBSVM: A library for support vector machines*, 2011



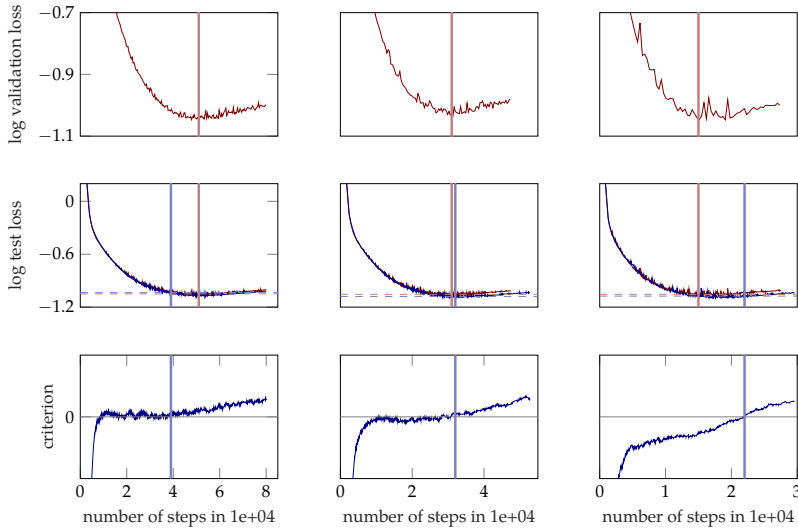


Figure 34: Multi-layer perceptron on MNIST trained with `sgd` and a mini-batch size of 128. Columns from left to right: learning rates 0.003, 0.005 and 0.01, respectively. Plot and colors as in Figure 33.

hidden units; the logistic regressor thus contains  $\sim 5.8$  million, and the shallow net  $\sim 11.1$  million trainable parameters. Experiments are set up in the same style as the ones in Section 3.3 and 3.4. We use 20% of the training data for the validation set which yields 1282 validation examples and a reduced number of 5130 training examples. Figure 35 shows results: Columns 1-2 for the logistic regressor and columns 3-4 for the shallow net. Since the dataset is quite small for this task, the gap between test losses is quite large (middle row). Both architectures do not overfit properly, the test loss rather flattens out, although we trained both architectures for very long ( $2.5 \cdot 10^5$  steps) and initialized weights close to zero. The `EB`-criterion is again a bit too cautious and induces stopping when the test loss starts to flatten out; but since it has access to an enlarged training set, it beats the validation set on both architectures.

### 6.3.6 Greedy Element-wise Stopping

The `EB`-criterion computes the quantities  $f_n := |\mathcal{B}|(\nabla L_B^n)^2 / \hat{\Sigma}_n \in \mathbb{R}$  for each gradient element  $n$  at each iteration. This quantity can be understood as a ‘signal-to-noise ratio’, and the `EB`-criterion is the average over the individual  $f_n$ . As a side experiment, we employ the same idea in an element-wise fashion: we stop the training for an individual parameter  $w_n \in \mathbb{R}$  (not to be confused with the full parameter vector  $w_t \in \mathbb{R}^N$  at iteration  $t$ ) as soon as the scalar  $f_n$  falls below one. Similarly to the `EB`-criterion before, we smooth each element-wise criterion with an exponential running average for noise reduction. Importantly, this is *not* a sparsification of the parameter vector  $w$  (since  $w_n$  is not set to zero when being switched off but merely fixed at its current

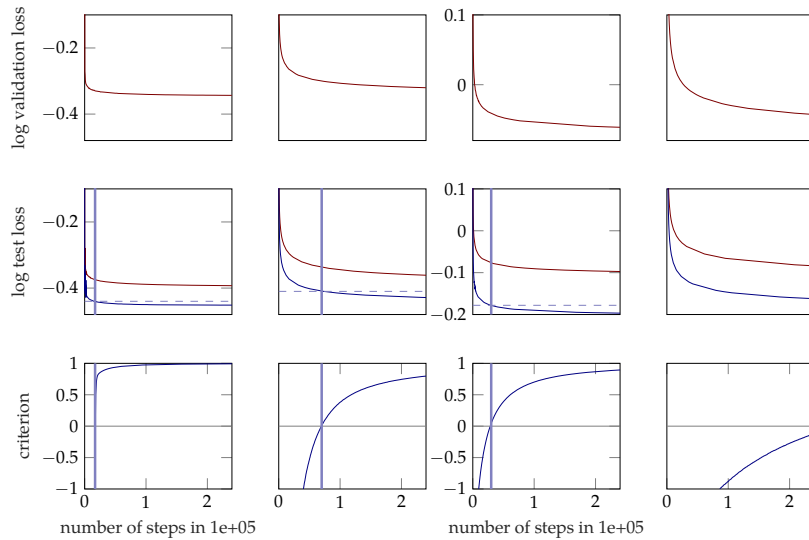


Figure 35: Shallow net and logistic regressor on SECTOR trained with SGD and mini-batch size 128. Columns 1-2: Logistic regression with learning rates 0.03 and 0.003 respectively. Columns 3-4: Shallow net with learning rates 0.03 and 0.003 respectively. Rows and colors as in Figure 33.

value), but rather a sparsification of the update. The smoothed averages are initialized at high values, resulting in a warm-up phase where all weights are ‘active’. Figure 36 presents results. Intriguingly, immediately after the warm-up phase the training of a considerable fraction of all weights (10 percent or more, depending on the training configuration) is being stopped. This fraction increases further as training progresses. Especially towards the end where overfitting sets in, a clear signal can be seen: the fraction of weights where learning has been stopped suddenly increases at a higher rate. Despite this reduction in effective model complexity, the network reaches test losses comparable to our training runs without greedy element-wise stopping (test losses in Figure 34). The fraction of switched-off parameters towards the end of the optimization process reaches up to 80 percent in a single layer and around 50 percent for the whole net.

#### 6.4 Comparison to RMSPROP

This section explores the differences and similarities of SGD+EB-criterion and RMSPROP. This is rather meant as a means for gaining a better intuition, and not for comparing them as competitors since both methods were derived for different purposes and could be combined in principle.

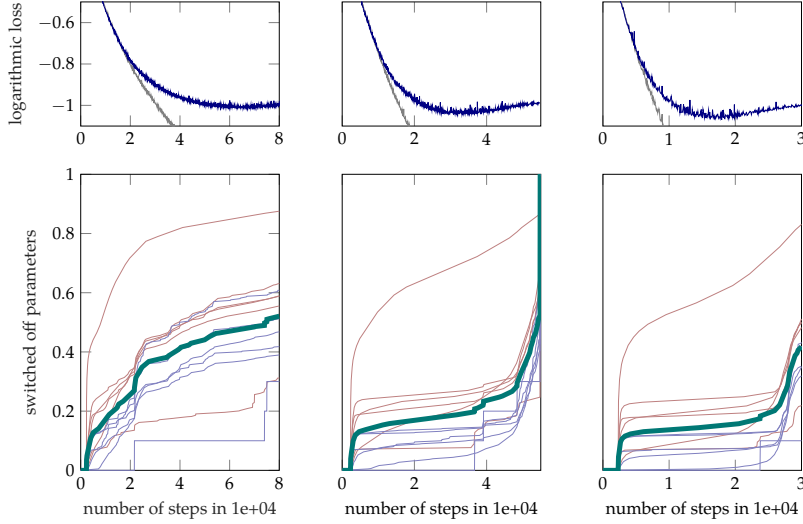


Figure 36: Greedy element-wise stopping for a multi-layer perceptron on MNIST trained with `sgd` and batch size 128. Columns: learning rates 0.003, 0.005 and 0.01, respectively. Top row: logarithmic training (—) and test loss (---). Bottom row: fraction of weights where learning has been shut off by the greedy element-wise stopping for each weight matrix (—), each bias vector (—) and the full net (—).

#### 6.4.1 Non-Greedy Element-wise EB-Criterion

The *non-greedy* element-wise EB-criterion can be formulated as

$$\begin{aligned} c_t &= \beta c_{t-1} + (1 - \beta) (1 - f_t^{\text{EB-crit}}) \\ w_{t+1} &= w_t - \alpha \cdot \mathbb{I}[c_t \leq 0] \odot \nabla L_{\mathcal{B}}(w_t) \end{aligned} \quad (101)$$

for some conservative smoothing constant  $\beta \in (0, 1)$ , usually  $\beta \approx 0.999$ , or 0.99, learning rate  $\alpha$ , and the fraction  $f_t^{\text{EB-crit}} := |\mathcal{B}|[\nabla L_{\mathcal{B}}(w_t)]^{\odot 2} \odot \hat{\Sigma}(w_t)$ . The symbol  $\odot$  denotes element-wise division and  $\mathbb{I}[\cdot]$  is the element-wise indicator function. In contrast to the greedy implementation of Section 6.3.6, where switched-off learning rates stayed switched off, Eq. 101 allows learning to be switched on again.

#### 6.4.2 Learning Rate Damping in RMSPROP

RMSPROP [135] is a well known optimization algorithm that scales learning rates element-wise by an exponential running average of gradient magnitudes (Chapter 2 § 2.4); specifically:

$$\begin{aligned} v_t &= \gamma v_{t-1} + (1 - \gamma) \nabla L_{\mathcal{B}}(w_t)^{\odot 2} \\ w_{t+1} &= w_t - \alpha \nabla L_{\mathcal{B}}(w_t) \odot v_t^{\odot 1/2}, \end{aligned} \quad (102)$$

again for some smoothing constant  $\gamma \in (0, 1)$ , usually  $\gamma \approx 0.95$ , and learning rate  $\alpha$ . Let  $a_t^{\max}$  be the largest element of the vector  $a_t := v_t^{\odot -1/2}$ , then the second line of Eq. 102 can be rewritten as

$$w_{t+1} = w_t - \alpha a_t^{\max} \left( \frac{a_t}{a_t^{\max}} \right) \odot \nabla L_{\mathcal{B}}(w_t). \quad (103)$$

[135] Tieleman and Hinton, *RMSprop Gradient Optimization*, 2015

The fraction  $f_t^{\text{RMSPROP}} := (a_t/a_t^{\max}) \in (0, 1]$  describes the scaling of learning rates relative to the largest one: if the  $i^{\text{th}}$  element of  $f_t^{\text{RMSPROP}}$  is very small, the learning of the corresponding parameter is damped heavily relative to a full step of size  $\alpha a_t^{\max}$ . This can be interpreted as ‘switching-off’ the learning of these parameters, similarly to the element-wise EB-criterion.

### 6.4.3 Connections and Differences

The following table gives a rough overview over the possible set of learning rates for each method.

method	domain	maximal $\alpha_t$	minimal $\alpha_t$
SGD	$\{\alpha\}$	$\alpha$	$\alpha$
SGD+EB-crit	$\{0, \alpha\}$	$\alpha$	0 (when converged)
RMSPROP	$(0, \alpha a_t^{\max}]$	$\alpha a_t^{\max}$	$> 0$

The table shows that SGD+EB-criterion is a very minor variation of SGD, in the sense that it can also set the learning rate to zero, but only for *converged* parameters. It does not change SGD during effective training; specifically, it does not explicitly encode curvature, or other geometric properties of the loss.

In contrast to this, RMSPROP also adapts the absolute value of the largest possible step at every iteration by a varying factor  $a_t^{\max}$ , and scales the other steps relative to it. Heuristically, gradient elements with a larger running average  $v_t \approx \mathbf{E}_{d \sim Q}[\nabla L_B(w_t)^{\odot 2}] = \nabla \mathcal{L}(w_t)^{\odot 2} + |\mathcal{B}|^{-1} \text{diag}[\Sigma(w_t)]$  get down-scaled relative to the ones with a smaller  $v_t$ -element [5]. Figure 37 sketches this behavior. The top plot shows contour lines (gray) of one element of the damping term  $(\nabla \mathcal{L}^{\odot 2} + |\mathcal{B}|^{-1} \text{diag}[\Sigma])^{-\frac{1}{2}}$  for a single dimension. The bottom plot shows color-coded slices through the top plot for different constant  $\nabla \mathcal{L}$ , and one for the diagonal slice (—) where  $\nabla \mathcal{L}^{\odot 2} = |\mathcal{B}|^{-1} \text{diag}[\Sigma]$ . Elements with large true gradients  $\nabla \mathcal{L}$  and/or noise variances  $|\mathcal{B}|^{-1} \text{diag}[\Sigma]$  get damped *relative* to ones with low  $\nabla \mathcal{L}$  and low  $|\mathcal{B}|^{-1} \text{diag}[\Sigma]$ . This does not mean, though, that the absolute step size of RMSPROP drops as the noise level  $\Sigma$  increases, since the expected absolute value of the numerator is non-trivially dependent on  $\Sigma$  as well. So, roughly speaking, the EB-criterion defines a strict threshold when learning should be terminated. RMSPROP defines a vaguer version, in the sense that the optimizer should move somewhat ‘less’ into directions of uncertain gradients. For example in situations where the EB-criterion triggers ( $\nabla \mathcal{L} = 0, \Sigma > 0$ ), RMSPROP just weighs the step proportional to  $\sqrt{|\mathcal{B}|^{-1} \Sigma}$  which depends on the absolute value of  $\Sigma$ , and not on a signal-to-noise ratio. Of course, the argumentation above that  $v_t \approx \mathbf{E}_{d \sim Q}[\nabla L_B(w_t)^{\odot 2}]$  also ignores the smoothing over

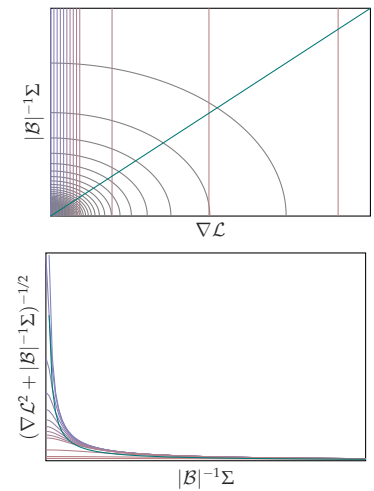


Figure 37: Sketch of RMSPROP-damping relative to an SGD-step. Top: contour lines of the damping term  $(\nabla \mathcal{L}^2 + |\mathcal{B}|^{-1} \Sigma)^{-\frac{1}{2}}$  Bottom: color-coded slices through the top plot for constant  $\nabla \mathcal{L}$ , and one (—) for the diagonal slice where  $\nabla \mathcal{L}^2 = |\mathcal{B}|^{-1} \Sigma$ .

[5] Balles and Hennig, “Follow the Signs for Robust Stochastic Optimization,” 2017

iterations. In contrast to this, the fractions  $f_t^{\text{EB-crit}}$  are computed locally, and only the local criteria are smoothed. This is essential for a stopping (not damping) decision if the magnitude of the gradient as well as its variance are dependent on  $w$ .<sup>4</sup>

#### 6.4.4 Empirical Comparison

For an empirical comparison, we run `RMSPROP`, `SGD` with element-wise `EB-criterion` (as in Eq. 101), and an instance of vanilla `SGD` on a multi-layer-perception on MNIST similar to the setup in Section 6.3.4. For the `SGD` instance that uses the `EB-criterion`, the fraction of switched-off parameters is defined as

$$P_t^{\text{EB-crit}} := \frac{1}{N} \sum_{n=1}^N \mathbb{I}[c_{n,t} > 0] \quad (104)$$

where  $c_{n,t}$  is the  $n^{\text{th}}$  element of  $c_t$  as in Eq. 101. The percentage of ‘switched-off’ parameters for `RMSPROP` can be roughly described as the fraction  $P_t^{\text{RMSPROP}}$  of parameters, whose  $f_t^{\text{RMSPROP}}$  (defined in Section 6.4.2) lie below a threshold  $f_{\text{cut}} \in (0, 1)$

$$P_t^{\text{RMSPROP}} := \frac{1}{N} \sum_{n=1}^N \mathbb{I}[f_{n,t}^{\text{RMSPROP}} < f_{\text{cut}}]. \quad (105)$$

The same smoothing factor  $\gamma = \beta = 0.99$  was used for both methods, for a meaningful comparison. Figure 38 depicts results: The first row shows training losses (light colors) and test losses (corresponding dark colors) of all three methods (`SGD`, `SGD+EB-criterion` and `RMSPROP`). Row 2 shows the evolution of  $P_t^{\text{EB-crit}}$ , and rows 3-7 of  $P_t^{\text{RMSPROP}}$  for five choices of  $f_{\text{cut}} = [10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$  respectively. As mentioned above, in contrast to the ‘greedy’ implementation of Section 6.3.6 (switched-off learning rates, stayed switched-off), and for a more natural comparison to `RMSPROP`, learning rates are allowed to be switched *on* again as well. The results for  $P_t^{\text{RMSPROP}}$  and  $P_t^{\text{EB-crit}}$  are color-coded: full net (—), weight matrices (—) and biases (—), the latter two each per layer.

The test losses of vanilla `SGD` and `SGD+EB-criterion` are almost identical, while the training loss of `SGD+EB-criterion` is a bit more conservative than the one of vanilla `SGD`. This is expected, since the `EB-criterion` ideally should not impair generalization performance, but might lead to larger training losses at convergence, due to the overfitting prevention. Already at the beginning of the training, `SGD+EB-criterion` switches off about 10-20% of all learning rates, after that, the fraction increases to about 50% (green line, second row). The curve is quite monotonic, exhibiting not significant jumps.

<sup>4</sup> We, in fact tried to use a running average for variance estimates instead of  $\hat{\Sigma}$ , since they are usually easier to access, but this corrupted the stopping decision heavily.

`RMSPROP` converges a bit faster, as it is expected. Also the plots for  $P_t^{\text{RMSPROP}}$  are richer in structure. Especially one layer seems to have significantly smaller learning rates for both, biases and weights, than the other layers. Overall the difference between the largest learning rate and all others tends to roughly increase over the optimization process (especially for  $f_{\text{cut}} = 10^{-1}$ , green line, last row). There are also significant jumps in all the curves in contrast to the rather monotonic increasing line of `SGD+EB-criterion`. This indicates nontrivial scaling of the absolute, as well as relative sizes of learning rates throughout the optimization process; also, no learning rate is smaller than  $10^{-5}$  times the largest one at each iteration (third row, green line at exactly zero).

## 6.5 Conclusion and Outlook

This chapter presented the `EB-criterion`, a novel approach to the problem of determining a good point for early-stopping in gradient-based optimization. In contrast to existing methods it does not rely on a held-out validation set and enables the optimizer to make use of all available training data. We exploit fast-to-compute statistics of the observed gradient to assess when it represents noise originating from the finiteness of the training set, instead of an informative gradient direction. The presented method so far is applicable in gradient descent as well as stochastic gradient descent settings and adds little overhead in computation, time, and memory consumption. Experimental results were presented for linear least-squares fitting, logistic regression and a multi-layer perceptron, proving the general concept to be viable. Furthermore, preliminary findings on greedy as well as non-greedy element-wise early-stopping open up the possibility to monitor and control model fitting with a higher level of detail.

Desirable future research directions are:

- Theoretical analysis of the criterion for convex functions. Under which assumptions can we be certain to stop in a ‘desirable region’?
- Thorough experimental tests: This chapter provided promising proof-of concept experiments. The method needs to be tested further, especially on non-convex functions which are in a sense ‘out-of model’, and theoretical guarantees are less accessible .
- Combining of the (element-wise or global) `EB-criterion` with further search directions like e. g., `RMSPROP`. This would be a powerful tool to encode both—learning rate scaling as well as overfitting prevention—in one solver.
- Clarification of the vague relation to hypothesis testing. Related to this: possible informative limits which arise due to the uncertainty

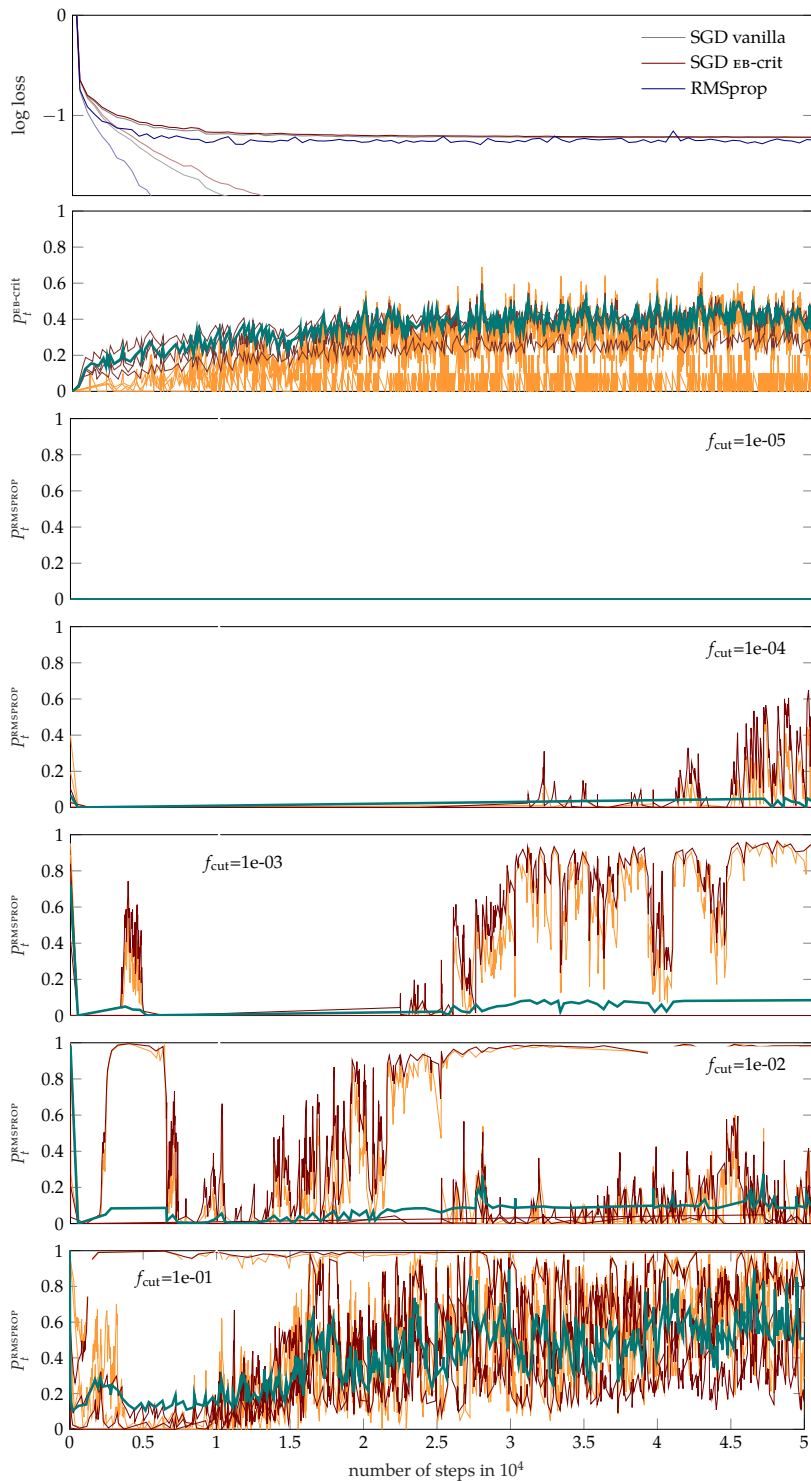


Figure 38: Comparison of `RMSPROP` and `SGD+EB-criterion` on a multi-layer perceptron on MNIST, both trained with batch size is 120. Top row: logarithmic training loss (light colors) and test loss (corresponding dark colors) for vanilla `SGD`, `SGD+EB-criterion` and `RMSPROP` (colors in legend). Row 2: fraction of weights  $D_t^{\text{EB-crit}}$  where learning has been shut off by the element-wise stopping: each weight matrix (—), each bias vector (—) and full net (—). Row 3-7: same as row 2, but for  $D_t^{\text{RMSPROP}}$  for different choices of threshold  $f_{\text{cut}}$  which are indicated in the legends.

of the  $\text{EB}$ -criterion itself i. e., constrains on the mini-batch size  $|\mathcal{B}|$  which controls the gradient noise as well as the goodness of the variances estimator. Also, when is a validation loss too uncertain or reliable enough to induce a stopping decision, and when the  $\text{EB}$ -criterion? This might help to decide which stopping criterion to use for which datasets and experimental setups.



Part III

## Automated Step Size Adaptation



## Probabilistic Line Searches

---

**I**N deterministic optimization, line searches are a standard tool ensuring stability and efficiency. Where only stochastic gradients are available, no direct equivalent has so far been formulated, because uncertain gradients do not allow for a strict sequence of decisions collapsing the search space. This chapter constructs a probabilistic line search by combining the structure of existing deterministic methods with notions from Bayesian optimization. The method retains a Gaussian process surrogate of the univariate optimization objective, and uses a probabilistic belief over the Wolfe conditions to monitor the descent. The algorithm has low computational cost, and no user-controlled parameters. Experiments show that it effectively removes the need to define a learning rate for stochastic gradient descent. The chapter is mostly based on the publications [89] and [90].

### 7.1 Motivation

Stochastic gradient descent (Chapter 2 § 2.3) is widely used in machine learning for the optimization of highly multivariate functions if their gradient is corrupted by noise. This includes the online or mini-batch training of neural networks, logistic regression [144] [14] and variational models e. g., [67] [60] [17]. In all these cases, noisy gradients arise because an empirical risk  $L_{\mathcal{D}}(w)$  of the optimization parameters  $w \in \mathbb{R}^N$ , across a large dataset  $\mathcal{D}$  of size  $|\mathcal{D}|$  is evaluated only on an i. i. d. sub-sampled set  $\mathcal{B} \subset \mathcal{D}$  (notation as in Chapter 2). As argued in Chapter 5, we will use here that the error  $L_{\mathcal{B}}(w) - L_{\mathcal{D}}(w)$  as well as the error  $\nabla L_{\mathcal{B}}(w) - \nabla L_{\mathcal{D}}(w)$  are unbiased and approximately normal distributed with (co-)variances  $|\mathcal{B}|^{-1}\Lambda_{\mathcal{D}}(w)$  and  $|\mathcal{B}|^{-1}\Sigma_{\mathcal{D}}(w)$  respectively. Despite its popularity and its low cost per step, SGD has well-known deficiencies that can make it inefficient, or at least tedious to use in practice. Two main issues are that, first, the gradient itself, even without noise, is not the optimal search *direction*; and second, SGD requires a *step size* (learning rate) that has a large effect on the algorithm's efficiency, is often difficult to choose well, and virtually never optimal for each individual descent step. The former issue, adapting the search direction, has been addressed by many authors (e. g., [43] for an overview). Existing approaches were discussed in Chapter 2 § 2.3; they range from lightweight diagonal preconditioning approaches like ADAM and ADAGRAD, to empirical estimates for the

[89] Mahsereci and Hennig, "Probabilistic Line Searches for Stochastic Optimization," 2015

[90] Mahsereci and Hennig, "Probabilistic Line Searches for Stochastic Optimization," 2017

[144] Zhang, "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms," 2004

[14] Bottou, "Large-scale machine learning with stochastic gradient descent," 2010

[67] Hoffman et al., "Stochastic variational inference," 2013

[60] Hensman, Rattray, and Lawrence, "Fast variational inference in the conjugate exponential family," 2012

[17] Broderick et al., "Streaming Variational Bayes," 2013

[43] George and Powell, "Adaptive step-sizes for recursive estimation with applications in approximate dynamic programming," 2006

natural gradient or the Newton direction [115], to problem-specific algorithms [108]. Most of these algorithms also include an auxiliary adaptive effect on the learning rate. Schaul, Zhang, and LeCun [119] provided an estimation method to adapt the learning rate from one gradient descent step to another using statistics over iterates. Reinforcement learning and ‘learning-to-learn’ approaches yield more specialized solvers that are supposed to work well on the same or similar problems they were trained on, but they also come with higher training cost and lower generality.

None of the mentioned algorithms change the size of the *current* descent step. Accumulating statistics across steps in this fashion requires some conservatism: If the step size is initially too large, or grows too fast, SGD can become unstable and ‘explode’, because individual steps are not checked for robustness at the time they are taken.

As discussed in Chapter 2 § 2.5, essentially the same problem exists in deterministic (noise-free) optimization problems. There, providing stability is one of several tasks of the *line search* subroutine. It is a standard constituent of algorithms like the classic nonlinear conjugate gradient [42] and quasi-Newton methods like BFGS [100, § 3].<sup>1</sup> In the noise-free case, line searches are considered a solved problem [100, § 3]. But the methods used in deterministic optimization are not stable to noise. They are easily fooled by even small disturbances, either becoming overly conservative or failing altogether. The reason for this brittleness is that existing line searches take a sequence of hard decisions to shrink or shift the search space. This yields efficiency, but breaks hard in the presence of noise. Section 7.2 constructs a probabilistic line search for noisy objectives, stabilizing the stochastic gradient decent algorithm.

## 7.2 From Classic to Probabilistic Line Searches

Chapter 2.5 introduced the concept of a *classic* lines search and the Wolfe terminations conditions (§ 2.5.1). This chapter will construct a *probabilistic* line search which operates in a similar scheme as its classic sibling, but reacts robustly to noise corrupted observations of the univariate function  $f(\alpha)$  and its gradient  $f'(\alpha)$  (same notation as in § 2.5). The exposition in § 7.4 will initially focus on the weak Wolfe conditions, which can be precisely modeled probabilistically. Section 7.4 then adds an approximate treatment of the strong form.

Consider minimizing  $f(\alpha) = L_{\mathcal{D}}(w(\alpha))$  where  $w(\alpha) = w_t + \alpha p_t$  and  $\alpha \in \mathbb{R}_+$ . That is, the algorithm can access only noisy function values and gradients  $y(\alpha), y'(\alpha)$  at location  $\alpha$ , with Gaussian likelihood

[115] Roux and Fitzgibbon, “A fast natural Newton method,” 2010

[108] Rajesh et al., “An Adaptive Learning Rate for Stochastic Variational Inference,” 2013

[119] Schaul, Zhang, and LeCun, “No more pesky learning rates,” 2013

[42] Fletcher and Reeves, “Function minimization by conjugate gradients,” 1964

[100] Nocedal and Wright, *Numerical Optimization*, 1999

<sup>1</sup> As mentioned in § 2.5, in these algorithms, another task of the line search is to guarantee certain properties of the surrounding estimation rule. In BFGS, e.g., it ensures positive definiteness of the Hessian estimate. This aspect will not feature here.

[100] Nocedal and Wright, *Numerical Optimization*, 1999

$$p(y(\alpha), y'(\alpha) | f(\alpha)) = \mathcal{N} \left( \begin{bmatrix} y(\alpha) \\ y'(\alpha) \end{bmatrix}; \begin{bmatrix} f(\alpha) \\ f'(\alpha) \end{bmatrix}, \begin{bmatrix} \sigma_f^2(\alpha) & 0 \\ 0 & \sigma_{f'}^2(\alpha) \end{bmatrix} \right). \quad (106)$$

The Gaussian form is supported by the Central Limit argument of Chapter 5. The function value  $y(\alpha)$  and the gradient  $y'(\alpha)$  are assumed independent for simplicity.<sup>2</sup> Each evaluation of  $f(\alpha)$  uses a newly drawn mini-batch. We will also use the notation  $t := \alpha/\alpha_0$  which expresses the input domain of  $f$  in units of  $\alpha_0$ , for convenience. With this:  $w(t) = w_t + t\alpha_0 p_t$ ,  $t \in \mathbb{R}_+$  and  $f(t) := f(\alpha(t))$  et cetera.<sup>3</sup>

The probabilistic line search is modeled after the classic line search routine `minimize.m`<sup>4</sup> and translates each of its ingredients one-by-one to the language of probability. The three main ingredients are: A robust yet lightweight Gaussian process surrogate on  $f(t)$  facilitating analytic optimization (§ 7.2.1); a simple Bayesian optimization objective for exploration (§ 7.3); and a probabilistic formulation of the Wolfe conditions as a termination criterion (§ 7.4). Appendix D contains a detailed pseudocode while Algorithm 4 sketches the structure of the probabilistic line search and highlights its essential ingredients.

```

1: function PROBLINESearchSketch( $f, y_0, y'_0, \sigma_{f_0}, \sigma_{f'_0}$ )
2:    $\mathcal{GP} \leftarrow \text{INITGP}(y_0, y'_0, \sigma_{f_0}, \sigma_{f'_0})$ 
3:    $T, Y, Y' \leftarrow \text{INITSTORAGE}(0, y_0, y'_0)$  // for observed points
4:    $t \leftarrow 1$  // location of initial candidate
5:
6:   while budget not used and no Wolfe-point found do
7:      $[y, y'] \leftarrow f(t)$  // evaluate objective
8:      $T, Y, Y' \leftarrow \text{UPDATESTORAGE}(t, y, y')$ 
9:      $\mathcal{GP} \leftarrow \text{UPDATEGP}(t, y, y')$ 
10:     $P^{\text{Wolfe}} \leftarrow \text{PROBWOLFE}(T, \mathcal{GP})$  // Wolfe probabilities at  $T$ 
11:
12:    if any  $P^{\text{Wolfe}}$  above Wolfe threshold  $c_W$  then
13:      return Wolfe-point
14:    else
15:       $T_{\text{cand}} \leftarrow \text{COMPUTECANDIDATES}(\mathcal{GP})$ 
16:       $EI \leftarrow \text{EXPECTEDIMPROVEMENT}(T_{\text{cand}}, \mathcal{GP})$ 
17:       $PW \leftarrow \text{PROBWOLFE}(T_{\text{cand}}, \mathcal{GP})$ 
18:       $t \leftarrow \text{where } (PW \odot EI) \text{ is maximal}$  // best candidate
19:    end if
20:  end while
21:
22:  // no Wolfe point found during budget
23:  return observed location in  $T$  with lowest GP mean
24: end function

```

<sup>2</sup> See § 7.4.1 and Appendix C.3 regarding estimation of the variances  $\sigma_f^2, \sigma_{f'}^2$ .

<sup>3</sup> The notation for the scaled step size  $t$  is now also overloaded with the index of the  $t^{\text{th}}$  iteration, usually denoted as subscript rather than input to a function, e.g., as in  $w_t$ .

<sup>4</sup> `minimize.m` was introduced in § 2.5. At the time of writing, it can be found here: <http://learning.eng.cam.ac.uk/carl/code/minimize/minimize.m>.

Algorithm 4: Sketch of probabilistic line search. Like Algorithm 2 (classic line search), the main algorithm consists of a loop which alternates between evaluating  $L_B$  and  $\nabla L_B$ , then checking all evaluated points for acceptance (by probabilistic Wolfe conditions), and finding new candidates for evaluation by lightweight Bayesian optimization. The latter is done by the subroutine `COMPUTENEXTCANDIDATES` which uses the univariate GP as model and returns up to  $|T|$  locations (at most one per cell and one extrapolation point). Then a promising point is selected among the candidates which maximizes the product of the expected improvement and the Wolfe probability, and the loop repeats. Note that again all relevant quantities ( $y, y', t$ , etc.) are scalar, which means that besides evaluating  $f$ , the probabilistic line search virtually adds no overhead to the optimization routine. Also the ability of ‘immediately-accept’ after the very first function evaluation (lines 12,13) exhibited by classic line searches is retained, such that well scaled initial trials  $\alpha_0$  ( $t = 1$ ) add no overhead.

## 7.2.1 Integrated Wiener Process Surrogate

We model information about the objective in a probability measure  $p(f)$ . There are two requirements on such a measure: First, it must be robust to irregularity (low and high variability) of the objective. And second, it must allow analytic computation of discrete candidate points for evaluation, because a line search should not call yet another optimization subroutine itself. Both requirements are fulfilled by a once-integrated Wiener process, i. e., a zero-mean Gaussian process prior  $p(f) = \mathcal{GP}(f; 0, k)$  with covariance function

$$k(t, t') = \theta^2 \left[ \frac{1}{3} \min^3(\tilde{t}, \tilde{t}') + \frac{1}{2} |t - t'| \min^2(\tilde{t}, \tilde{t}') \right]. \quad (107)$$

Here  $\tilde{t} := t + \tau$  and  $\tilde{t}' := t' + \tau$  denote a shift by a constant  $\tau > 0$ . This ensures this kernel is positive semi-definite, the precise value  $\tau$  is irrelevant as the algorithm only considers positive values of  $t$  (the implementation uses  $\tau = 10$ ). See § 7.4.1 regarding the scale  $\theta^2$ . With the likelihood of Eq. 106, this prior gives rise to a GP posterior whose mean function is a cubic spline [138].<sup>5,6</sup>

Because Gaussian distributions are closed under linear maps, Eq. 107 implies a Wiener process (linear spline) model on  $f'$ :

$$p(f; f') = \mathcal{GP} \left( \begin{bmatrix} f \\ f' \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k & k^\partial \\ \partial k & \partial k^\partial \end{bmatrix} \right), \quad (108)$$

with (using the indicator function  $\mathbb{I}(x) = 1$  if  $x$ , else 0)

$$\begin{aligned} k_{tt'}^\partial &:= \frac{\partial k(t, t')}{\partial t'} = \theta^2 \left[ \mathbb{I}(t < t') \frac{\tilde{t}^2}{2} + \mathbb{I}(t \geq t') \left( \tilde{t}\tilde{t}' - \frac{\tilde{t}'^2}{2} \right) \right] \\ \partial k_{tt'} &:= \frac{\partial k(t, t')}{\partial t} = \theta^2 \left[ \mathbb{I}(t' < t) \frac{\tilde{t}'^2}{2} + \mathbb{I}(t' \geq t) \left( \tilde{t}\tilde{t}' - \frac{\tilde{t}^2}{2} \right) \right] \\ \partial k_{tt'}^\partial &:= \frac{\partial^2 k(t, t')}{\partial t' \partial t} = \theta^2 \min(\tilde{t}, \tilde{t}'). \end{aligned} \quad (109)$$

Given a set of evaluations  $(T, Y, Y')$  (vectors, with elements  $t_i, y_{t_i}, y'_{t_i}$ ,  $i = 1, \dots, M$ ) with independent likelihood 106, the posterior  $p(f | Y, Y')$  is a GP with posterior mean function  $\mu$  and covariance function  $\tilde{k}$  as follows:

$$\begin{aligned} \begin{bmatrix} \mu(t) \\ \mu'(t) \end{bmatrix} &= \underbrace{\begin{bmatrix} k_{tT} & k_{tT}^\partial \\ \partial k_{tT} & \partial k_{tT}^\partial \end{bmatrix} \begin{bmatrix} k_{TT} + \sigma_f^2 I & k_{TT}^\partial \\ \partial k_{TT} & \partial k_{TT}^\partial + \sigma_{f'}^2 I \end{bmatrix}^{-1}}_{=: \mathbf{g}^\top(t)} \begin{bmatrix} Y \\ Y' \end{bmatrix}, \\ \begin{bmatrix} \tilde{k}(t, t') & \tilde{k}^\partial(t, t') \\ \tilde{\partial k}(t', t) & \tilde{\partial k}^\partial(t', t) \end{bmatrix} &= \begin{bmatrix} k_{tt'} & k_{tt'}^\partial \\ \partial k_{tt'} & \partial k_{tt'}^\partial \end{bmatrix} - \mathbf{g}^\top(t) \begin{bmatrix} k_{Tt'} & k_{Tt'}^\partial \\ \partial k_{Tt'} & \partial k_{Tt'}^\partial \end{bmatrix}. \end{aligned} \quad (110)$$

[109] Rasmussen and Williams, *Gaussian Processes for Machine Learning*, 2006

[138] Wahba, *Spline models for observational data*, 1990

<sup>5</sup> Eq. 107 can be generalized to the ‘natural spline’, removing the need for the constant  $\tau$  [109, § 6.3.1]. However, this notion is ill-defined in the case of a single observation, as in the line search.

<sup>6</sup> As stated in Chapter 1.4, regression on  $f$  and  $f'$  from  $M$  observations of pairs  $(y, y')$  can be formulated as a filter and thus performed in  $\mathcal{O}(M)$  time. However, since a line search typically collects  $< 10$  data points, generic GP inference, using a Gram matrix, has virtually the same, low cost.

The posterior marginal variance will be denoted by  $\mathbb{V}(t) = \bar{k}(t, t)$ . To see that  $\mu$  is indeed piecewise cubic (i. e., a cubic spline), we note that it has at most three non-vanishing derivatives<sup>7</sup>, because

$$\begin{aligned} \partial^2 k_{tt'} &:= \frac{\partial^2 k(t, t')}{\partial t^2} = \theta^2 \mathbb{I}(t \leq t') & \partial^3 k_{tt'} &:= \frac{\partial^3 k(t, t')}{\partial t^3} = \theta^2 \mathbb{I}(t \leq t')(t' - t) \\ \partial^2 k_{t't} &:= \frac{\partial^2 k(t, t')}{\partial t'^2} = -\theta^2 \mathbb{I}(t \leq t') & \partial^3 k_{t't} &:= \frac{\partial^3 k(t, t')}{\partial t'^3} = 0. \end{aligned} \quad (111)$$

This piecewise cubic form of  $\mu$  is beneficial for our purposes: having collected  $M$  values of  $f$  and  $f'$ , respectively, all local minima of  $\mu$  can be found analytically in  $\mathcal{O}(M)$  time in a single sweep through the ‘cells’  $t_{i-1} < t < t_i$ ,  $i = 1, \dots, M$  (here  $t_0 = 0$  denotes the start location, where  $(y_0, y'_0)$  are ‘inherited’ from the preceding line search. For typical line searches  $M < 10$ , c.f. § 7.5. In each cell,  $\mu(t)$  is a cubic polynomial with at most one minimum in the cell, found by an inexpensive quadratic computation from the three scalars  $\mu'(t_i), \mu''(t_i), \mu'''(t_i)$ . This is in contrast to other GP regression models—for example the one arising from a squared exponential kernel—which give more involved posterior means whose local minima can be found only approximately.

Another advantage of the cubic spline interpolant is that it does not encode the existence of higher derivatives (in contrast to the Gaussian kernel, for example), and thus reacts robustly to irregularities in the objective. In our algorithm, after each evaluation of  $(y_M, y'_M)$ , we use this property to compute a short list of *candidates* for the next evaluation, consisting of the  $< M$  local minimizers of  $\mu(t)$  and one additional *extrapolation* node at  $t_{\max} + t_{\text{ext}}$ , where  $t_{\max}$  is the currently largest evaluated  $t$ , and  $t_{\text{ext}}$  is an extrapolation step size starting at  $t_{\text{ext}} = 1$  and doubled after each extrapolation step.

A conceptual rather than algorithmic motivation for using the *iWP* as surrogate, are classic line searches. There, the 1D-objective is modeled by piecewise cubic interpolations between neighboring datapoints. In a sense, this is a non-parametric approach, since a new spline is defined, when a datapoint is added. Classic line searches though always only deal with one spline at a time, since they are able to collapse/rule out all other parts of the search space. Indeed, for noise-free observations, the mean of the posterior *iWP* is identical to the classic cubic interpolations, and thus candidate locations are identical as well (Figure 39 for illustration). The non-parametric approach also prevents issues of over-constrained surrogates for more than two datapoints. For example, unless the objective is a perfect cubic function, it is impossible to fit a parametric third order polynomial to it, for more than two noise-free observations. All other variability in the objective would need to be explained away by artificially introducing noise on the observations. An integrated Wiener process very naturally extends its complexity with each newly added datapoint without

<sup>7</sup> There is no well-defined probabilistic belief over  $f''$  and higher derivatives—sample paths of the Wiener process are almost surely non-differentiable almost everywhere [1, § 2.2]. But  $\mu(t)$  is always a member of the reproducing kernel Hilbert space induced by  $k$ , thus piecewise cubic [109, § 6.1].

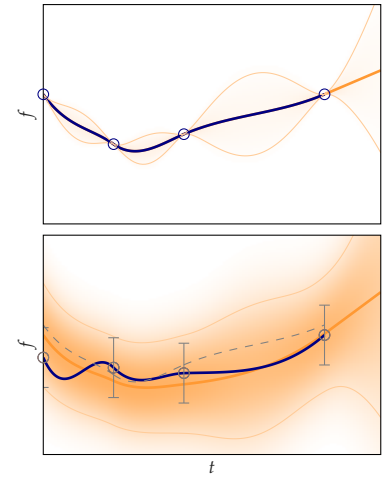


Figure 39: Integrated Wiener process. GP marginal posterior of function values. posterior mean  $\mu$  (—), and  $\pm 2\sqrt{\mathbb{V}}$  (—), local pdf marginal shaded; function value observations (—○—) (corresponding gradients not shown). Classic interpolation by piecewise cubic spline (—). Top: Exact observations ( $\sigma_f = 0$ ); the mean of the GP and the cubic spline interpolator of a classic line search coincide. Bottom: Same observations with additive Gaussian noise (error-bars indicate  $\pm\sigma_f$ ); noise-free interpolator of top plot (---) for comparison. The classic interpolator (—), which exactly matches the observations, becomes unreliable, the GP reacts robustly to noisy observations while the GP-mean still consists of piecewise cubic splines.

[1] Adler, *The Geometry of Random Fields*, 1981

[109] Rasmussen and Williams, *Gaussian Processes for Machine Learning*, 2006

being overly assertive – the encoded assumption is, that the objective has at least one derivative which is also observed in this case.

**REMARK:** *The current design of the probabilistic line search considers the GP-mean but not the variance for candidate selection. For the integrated Wiener process (IWP) and heteroscedastic noise, the variance  $\mathbb{V}(t)$  always attains its maximum exactly at the mid-point between two evaluations. Including the variance into the candidate proposal biases the existing candidates towards the center. Additional candidates might occur between evaluations without local minimizer, even for noise-free observations/classic line searches. We did not explore this further since the algorithm showed very good sample efficiency already with the adopted scheme, and has the possibly desirable property of reverting to the classic candidate proposal for noise-free observations.*

### 7.3 Lightweight BayesOpt for Candidate Selection

The previous section described the construction of  $\leq M$  discrete candidate points for the next evaluation. To decide at which of the candidate points to actually call  $f$  and  $f'$ , we make use of a popular acquisition function from Bayesian optimization. *Expected improvement* ([71], introduced in Chapter 4 § 4.1 Eq. 85) is the expected amount, under the GP surrogate, by which the function  $f(t)$  might be smaller than a ‘current best’ value  $\eta$  (we set  $\eta = \min_{i=0,\dots,M}\{\mu(t_i)\}$ , where  $t_i$  are observed locations),

$$u_{\text{EI}}(t) = \frac{\eta - \mu(t)}{2} \left( 1 + \operatorname{erf} \frac{\eta - \mu(t)}{\sqrt{2\mathbb{V}(t)}} \right) + \sqrt{\frac{\mathbb{V}(t)}{2\pi}} \exp \left( -\frac{(\eta - \mu(t))^2}{2\mathbb{V}(t)} \right). \quad (112)$$

The next evaluation point is chosen as the candidate maximizing the product of Eq. 112 and Wolfe probability  $p^{\text{Wolfe}}$ , which is derived in the following section. The intuition is that  $p^{\text{Wolfe}}$  precisely encodes properties of desired points, but has poor exploration properties;  $u_{\text{EI}}$  has better exploration properties, but lacks the information that we are seeking a point with low curvature;  $u_{\text{EI}}$  thus puts weight on, by W-II, clearly ruled out points. An illustration of the candidate proposal and selection is shown in Figure 40.

**REMARK:** *In principle other acquisition functions, e. g., the upper-confidence bound, GP-UCB [130], Eq. 85, are possible, which might have a stronger explorative behavior; we opted for  $u_{\text{EI}}$  since exploration is less crucial for line searches than for general BO and some, e. g., GP-UCB, had one additional parameter to tune. We tracked the sample efficiency of  $u_{\text{EI}}$  instead and it was very good (low), and comparable to the one of classic line searches. The experimental Subsection 7.5.3 contains further comments and experiments on the alternative choices of  $u_{\text{EI}}$  and  $p^{\text{Wolfe}}$  as standalone acquisition functions; they*

[71] Jones, Schonlau, and Welch, “Efficient global optimization of expensive black-box functions,” 1998

[130] Srinivas et al., “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design,” 2010



performed equally well in terms of loss and sample efficiency to their product on the tested setups.

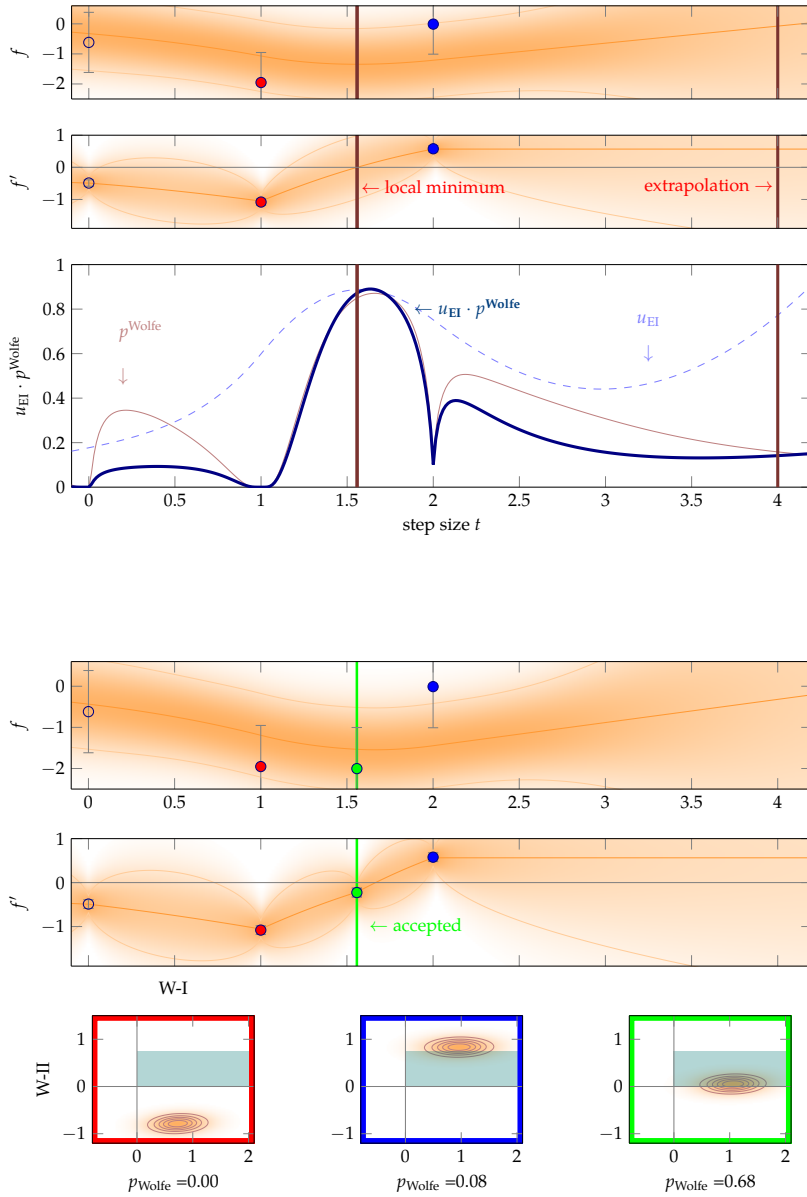


Figure 40: Candidate selection by Bayesian optimization. Top and bottom: GP marginal posterior of function values and gradients respectively (colors as in Figure 39); newly collected evaluations (●/●). Locations of the *two* candidates as vertical lines (—). The left one at about  $t_1^{\text{cand}} \approx 1.54$  is a local minimum of the mean  $\mu$  (the gradient mean  $\mu'$  crosses through zero here); the right one at  $t_2^{\text{cand}} = 4$  is a candidate for extrapolation. Bottom: Decision criterion in arbitrary scale: Expected improvement  $u_{\text{EI}}$  (---, Eq. 112), the Wolfe probability  $p^{\text{Wolfe}}$  (—, Eqs. 115, 117), their decisive product (—). For illustrative purposes all criteria are plotted for the whole  $t$ -space. In practice solely the values at  $t_1^{\text{cand}}$  and  $t_2^{\text{cand}}$  are computed, compared, and the candidate with the higher value of  $u_{\text{EI}} \cdot p^{\text{Wolfe}}$  is chosen for evaluation. In this example this would be the candidate at  $t_1^{\text{cand}}$ .

Figure 41: Acceptance procedure. Top and middle: plot and colors as in Figure 40 with an additional observation (●). Bottom: Implied bivariate Gaussian belief over the validity of the Wolfe conditions (Eq. 114a) as contours at the red, blue and green point respectively. Points are considered acceptable if their Wolfe probability  $p_t^{\text{Wolfe}}$  is above a threshold  $c_W = 0.3$ ; this means that at least 30% of the orange 2D Gauss density must cover the shaded area (■). Only the point ● fulfills this condition and is therefore accepted.

## 7.4 Probabilistic Wolfe Conditions for Termination

The key observation for a probabilistic extension of the Wolfe conditions W-I and W-II is that they are positivity constraints on two

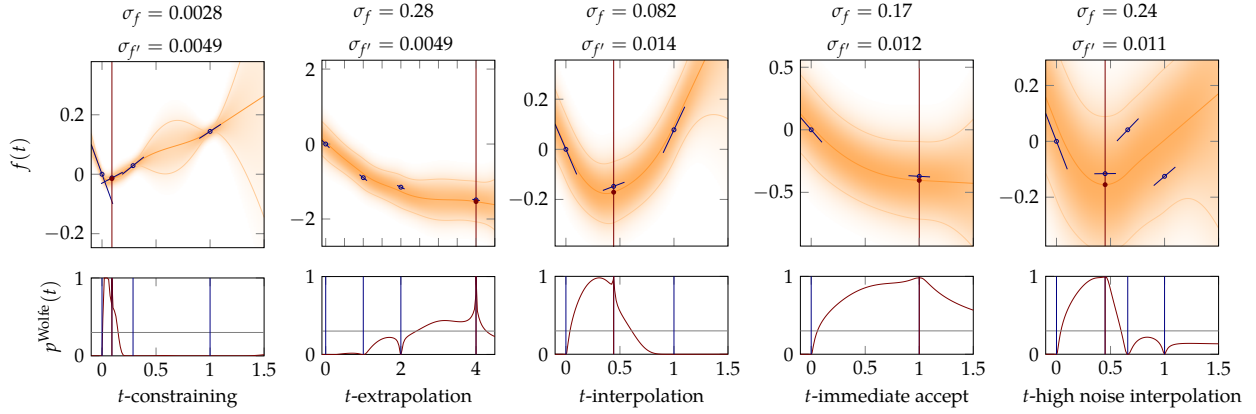


Figure 42: Curated snapshots of line searches (from N-I on MNIST), showing variability of the objective's shape and the decision process. Top row: GP marginal posterior of function values and evaluations, bottom row: approximate  $p^{\text{Wolfe}}$  over strong Wolfe conditions. Accepted point marked red.

variables  $a_t, b_t$  that are both linear projections of the jointly Gaussian variables  $f$  and  $f'$ :

$$\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(0) \\ f'(0) \\ f(t) \\ f'(t) \end{bmatrix} \geq 0. \quad (113)$$

The GP of Eq. 108 on  $f$  thus implies, at each value of  $t$ , a bivariate Gaussian distribution

$$p(a(t), b(t)) = \mathcal{N} \left( \begin{bmatrix} a(t) \\ b(t) \end{bmatrix}; \begin{bmatrix} m^a(t) \\ m^b(t) \end{bmatrix}, \begin{bmatrix} C^{aa}(t) & C^{ab}(t) \\ C^{ba}(t) & C^{bb}(t) \end{bmatrix} \right), \quad (114a)$$

$$m^a(t) = \mu(0) - \mu(t) + c_1 t \mu'(0)$$

$$m^b(t) = \mu'(t) - c_2 \mu'(0) \quad (114b)$$

$$C^{aa}(t) = \tilde{k}_{00} + (c_1 t)^2 \partial \tilde{k}_{00}^\partial + \tilde{k}_{tt} + 2[c_1 t (\tilde{k}_{00}^\partial - \partial \tilde{k}_{0t}^\partial) - \tilde{k}_{0t}^\partial]$$

$$C^{bb}(t) = c_2^2 \partial \tilde{k}_{00}^\partial - 2c_2 \partial \tilde{k}_{0t}^\partial + \partial \tilde{k}_{tt}^\partial \quad (114c)$$

$$C^{ab}(t) = C^{ba}(t) = -c_2 (\tilde{k}_{00}^\partial + c_1 t \partial \tilde{k}_{00}^\partial) + c_2 \partial \tilde{k}_{0t}^\partial + \partial \tilde{k}_{t0}^\partial + c_1 t \partial \tilde{k}_{0t}^\partial - \tilde{k}_{tt}^\partial.$$

The quadrant probability  $p^{\text{Wolfe}}(t) = p(a(t) > 0 \wedge b(t) > 0)$  for the Wolfe conditions to hold, is an integral over a bivariate normal probability,

$$p^{\text{Wolfe}}(t) = \int_{-\frac{m^a(t)}{\sqrt{C^{aa}(t)}}}^{\infty} \int_{-\frac{m^b(t)}{\sqrt{C^{bb}(t)}}}^{\infty} \mathcal{N} \left( \begin{bmatrix} a(t) \\ b(t) \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho(t) \\ \rho(t) & 1 \end{bmatrix} \right) da(t) db(t), \quad (115)$$

with correlation coefficient  $\rho(t) = C^{ab}(t) / \sqrt{C^{aa}(t)C^{bb}(t)}$ . It can be computed efficiently [33], using readily available code.<sup>8</sup> The line search computes this probability for all evaluation nodes, after each evaluation. If any of the nodes fulfills the Wolfe conditions with

<sup>8</sup> e.g., <http://www.math.wsu.edu/faculty/genz/software/matlab/bvn.m>

[33] Drezner and Wesolowsky, "On the computation of the bivariate normal integral," 1990

$p^{\text{Wolfe}}(t) > c_W$ , greater than some threshold  $0 < c_W < 1$ , it is accepted and returned. If several nodes simultaneously fulfill this requirement, the most recently evaluated node is returned.<sup>9</sup> Section 7.4.1 below motivates fixing  $c_W = 0.3$ . The acceptance procedure is illustrated in Figure 41.

*Approximation for Strong Conditions:*

As noted in Chapter 2 § 2.5.1, deterministic optimizers tend to use the strong Wolfe conditions, which use  $|f'(0)|$  and  $|f'(t)|$ . A precise extension of these conditions to the probabilistic setting is numerically taxing, because the distribution over  $|f'|$  is a non-central  $\chi$ -distribution, requiring customized computations. However, a straightforward variation to Eq. 115 captures the spirit of the strong Wolfe conditions, that large positive derivatives should not be accepted: Assuming  $f'(0) < 0$ , i. e., that the search direction is a descent direction, the strong second Wolfe condition can be written exactly as

$$0 \leq b(t) = f'(t) - c_2 f'(0) \leq -2c_2 f'(0). \quad (116)$$

The value  $-2c_2 f'(0)$  is bounded to 95% confidence by

$$-2c_2 f'(0) \lesssim 2c_2 \left( |\mu'(0)| + 2\sqrt{\mathbb{V}'(0)} \right) =: \bar{b}. \quad (117)$$

Hence, an approximation to the strong Wolfe conditions can be reached by replacing the infinite upper integration limit on  $b$  in Eq. 115 with  $(\bar{b} - m^b(t)) / \sqrt{C^{bb}(t)}$ . The effect of this adaptation, which adds no overhead to the computation, is shown in Figure 43 as a dashed line.

#### 7.4.1 Eliminating Hyper-parameters

As a black-box inner loop, the line search should not require any tuning by the user. The preceding section introduced six so-far undefined parameters:  $c_1, c_2, c_W, \theta, \sigma_f, \sigma_{f'}$ . We will now show that  $c_1, c_2, c_W$  can be fixed by hard design decisions:  $\theta$  can be eliminated by standardizing the optimization objective within the line search; and the noise levels can be estimated at runtime with low overhead for finite-sum objectives of the form in Eq. 28. The result is a parameter-free algorithm that effectively *removes* the one most problematic parameter from SGD—the learning rate.

*Design Parameters  $c_1, c_2, c_W$*

The algorithm inherits the Wolfe thresholds  $c_1$  and  $c_2$  from its deterministic ancestors. We set  $c_1 = 0.05$  and  $c_2 = 0.5$ . This is a standard setting that yields a ‘lenient’ line search, i. e., one that accepts most

<sup>9</sup> There are additional safeguards for cases where e.g. no Wolfe-point can be found, which can be deduced from the pseudo-code in Appendix D; they are similar to standard safeguards of classic line search routines, e.g., returning the node of lowest mean.

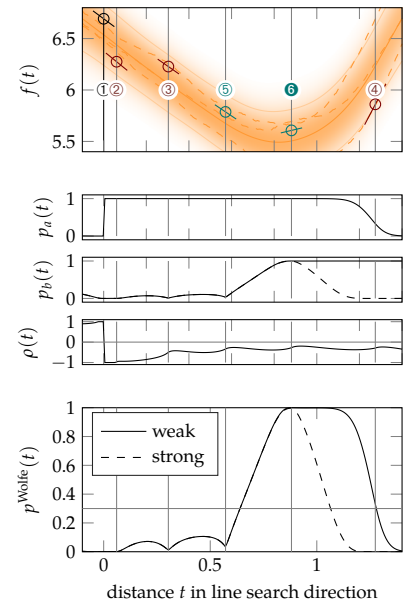


Figure 43: Sketch of a probabilistic line search. As in Figure 21, the algorithm performs extrapolation (②,③,④) and interpolation (⑤,⑥), but receives unreliable, noisy function and gradient values. These are used to construct a GP posterior (top: colors as in Figure 40, three dashed sample paths). This implies a bivariate Gaussian belief (§ 7.4) over the validity of the weak Wolfe conditions (middle three plots.  $p_a(t)$  is the marginal for W-I,  $p_b(t)$  for W-II,  $\rho(t)$  their correlation). Points are considered acceptable if their joint probability  $p^{\text{Wolfe}}(t)$  (bottom) is above a threshold (gray horizontal). An approximation to the strong Wolfe conditions is shown dashed.

descent points. The rational is that line searches shall not waste expensive evaluations, since more precise, or strict line searches usually do not yield a better performance of the outer optimizer.

The acceptance threshold  $c_W$  is a new design parameter arising only in the probabilistic setting. We fix it to  $c_W = 0.3$ . To motivate this value, first note that in the noise-free limit, all values  $0 < c_W < 1$  are equivalent, because  $p^{\text{Wolfe}}$  then switches discretely between 0 and 1 upon observation of the function. A back-of-the-envelope computation, assuming only two evaluations at  $t = 0$  and  $t = t_1$  and the same fixed noise level on  $f$  and  $f'$  (which then cancels out), shows that function values barely fulfilling the conditions, i. e.,  $a(t_1) = b(t_1) = 0$ , can have  $p^{\text{Wolfe}} \sim 0.2$  while function values at  $a(t_1) = b(t_1) = -\epsilon$  for  $\epsilon \rightarrow 0$  with ‘unlucky’ evaluations (both function and gradient values one standard-deviation from true value) can achieve  $p^{\text{Wolfe}} \sim 0.4$ . The choice  $c_W = 0.3$  balances the two competing desiderata for precision and recall. Additionally, for a fixed mean  $[m^a, m^b]^\top$  and large covariance (roughly  $\det[\text{cov}[a, b]] \rightarrow \infty$ ), it is that  $p^{\text{Wolfe}} \rightarrow 0.25$ . Since the line search should not accept completely uninformative points, the threshold  $c_W$  should lie above this value. Empirically (Figure 42), we rarely observed values of  $p^{\text{Wolfe}}$  close to this threshold. Even at high evaluation noise, a function evaluation typically either clearly rules out the Wolfe conditions, or lifts  $p^{\text{Wolfe}}$  well above the threshold.

### Scale $\theta$

The parameter  $\theta$  of Eq. 107 simply scales the prior covariance. It can be eliminated by scaling the optimization objective: We set  $\theta = 1$  and scale  $y_i \leftarrow (y_i - y_0)/|y'_0|$ ,  $y'_i \leftarrow y'_i/|y'_0|$  within the code of the line search. This gives  $y(0) = 0$  and  $y'(0) = -1$ , and typically ensures the objective ranges in the single digits across  $0 < t < 10$ , where most line searches take place. The division by  $|y'_0|$  causes a non-Gaussian disturbance, but this does not seem to have notable empirical effect.

**REMARK:** *The assumption behind this scaling is that the initial step size provides a rough estimate of where we expect to see a Wolfe point. This is especially true if the learning rate is well scaled from previous line searches. For the Wiener kernel, scaling  $f'$  with  $\theta^2$  is equivalent to scaling the input  $t$  with  $\theta^2$ . We would like  $\mu(t) + \sqrt{\mathbb{V}'(t)}$  to cross zero at about  $t = \alpha/\alpha_0 = 1$ , i. e.,  $\theta \approx |y'_0|$ ; this is approximated by scaling  $y_0$  and  $y'_0$  as mentioned above.*

### Noise Scales $\sigma_f, \sigma_{f'}$

The likelihood 106 requires standard deviations for the noise on both function values ( $\sigma_f$ ) and gradients ( $\sigma_{f'}$ ). One could attempt to learn these across several line searches. However, in empirical risk models, as captured by Eq. 28, the variance of the loss and its gradient can be estimated directly for the mini-batch, at low computational overhead

As described in Chapter 5, we collect the empirical statistics  $\hat{\Sigma}(w)$  and  $\hat{\Lambda}(w)$  (Eqs. 90 and 91 respectively) during each evaluation of  $L_B$  and  $\nabla L_B$  and set

$$\sigma_f^2 = \hat{\Lambda}(w(0)) \quad \text{and} \quad \sigma_{f'}^2 = (p_t^{\odot 2})^\top \hat{\Sigma}(w(0)) \quad (118)$$

at the beginning of a line search. This amounts to the assumption that noise on the gradient is independent. We finally scale the two empirical estimates:  $\sigma_f \leftarrow \sigma_f / |y'(0)|$ , and ditto for  $\sigma_{f'}$ , in the same style as  $y$  and  $y'$ . The overhead of this estimation is rather small if the computation of  $\ell(x, y_j)$  itself is more expensive than the summation over  $j$ . For neural networks the factor is upper bounded by 1.3 (Chapter 5.2).

#### *Propagating Step Sizes Between Line Searches*

As will be demonstrated in §7.5, the line search can find good step sizes even if the length of the direction  $s_i$  is mis-scaled. Since such scale issues typically persist over time, it would be wasteful to have the algorithm re-fit a good scale in each line search. Instead, we propagate step lengths from one iteration of the search to another: We set the initial search direction to  $p_0 = -\alpha_0 \nabla L_B(w_0)$  with some initial learning rate  $\alpha_0$ . Then, after each line search ending at  $w_{t+1} = w_t + t_t^* p_t$ , the next search direction is set to  $p_t = -\alpha_{\text{ext}} \cdot t_t^* \alpha_0 \nabla L_B(w_t)$  (with  $\alpha_{\text{ext}} = 1.3$ ). Thus, the next line search starts its extrapolation at 1.3 times the step size of its predecessor (Section 7.5.2 discusses details).

#### 7.4.2 Relation to Bayesian Optimization and Noise-Free Limit

The probabilistic line search algorithm is closely related to Bayesian optimization (BO) discussed in Chapter 4 § 4.1 since it approximately minimizes a (1D-)objective under potentially noisy function evaluations. Similarities to BO mostly lie in the model structure, e. g., a GP-surrogate for the objective, and an acquisition function to discriminate locations for the next evaluation of the loss; but there are some differences to BO concerning the aim, requirements on computational efficiency, and termination condition, which are shortly discussed here: (i) *Performance measure*: The final performance in BO is usually measured by the lowest found value of the objective function. Line searches are subroutines inside of a greedy, iterative optimization machine, which often performs several thousand steps (and line searches); many, very approximate steps often performs better overall than taking less, but preciser steps. (ii) *Termination*: The line search, in contrast to BO, has access to a clear measure when an evaluated point is ‘good enough’. This stopping decision is encoded in the Wolfe termination conditions and is imposed from the outside. Stricter Wolfe conditions do not usually improve the performance of the outer op-

timizer, thus, no matter if a better (lower) minimum could be found, any Wolfe-point is acceptable at all times. This especially enables the line search to ‘immediately-accept’ the first evaluated node, a key feature to an efficient line search routine. (iii) *Sample efficiency*: Since the last evaluation from the previous line search can be re-used in the current line search, only one additional value and gradient evaluation is enough to terminate the procedure. This ‘immediate-accept’ is, in contrast to  $\text{BO}$ , the *desired behavior* if the learning rate is currently well calibrated. (iv) *Locations for evaluation*:  $\text{BO}$ , usually calls an optimizer to maximize the acquisition function, and the preciseness of this optimization is crucial for performance. Line searches just need to find a Wolfe-acceptable point. Classic line searches suggest, that it is enough to look at plausible locations, like minimizer of a local interpolator, or some rough extrapolation point; this inexpensive heuristic usually works rather well. (v) *Exploration*:  $\text{BO}$  needs to solve an intricate trade-off problem in between exploring enough of the parameters space for possible locations of minima, and exploiting locations around them further. Since line searches are only concerned with finding a Wolfe-point, they do not need to explore the parameter space of possible step sizes to that extend; crucial features are rather the possibility to explore somewhat larger steps than previous ones (which is done by extrapolation-candidates), and likewise to shorted steps (which is done by interpolation-candidates).

#### *Noise-Free Limit*

In the limit of noise-free observed gradients and function values ( $\sigma_f = \sigma_{f'} = 0$ ) the probabilistic line search reverts to its classic parent: The GP-mean is identical to the classic interpolator, all candidate locations are thus identical, too. The Wolfe probability becomes binary and is identical to the classic Wolfe conditions (1 for fulfilled, and 0 otherwise). The only slight difference is that the probabilistic line search always also proposes one extrapolation candidate/ a second option, since it does not collapse the search space strictly. Thus, in rare cases, another candidate might be chosen for evaluation (which is subject to the same Wolfe conditions).

#### 7.4.3 Implementation

The line search routine itself has little memory and time overhead, comparable to a classic line search; most importantly it is independent of the dimensionality  $N$  of the optimization problem.

### Computational Time Overhead

After every call of the objective function, the  $\text{GP}$  needs to be updated, which at most is at the cost of inverting a  $2M \times 2M$ -matrix, where  $M$  usually is equal to 1, 2, or 3 but never  $> 10$ . In addition, the bivariate normal integral  $p^{\text{Wolfe}}$  of Eq. 115 needs to be computed at most  $M$  times. On a laptop, one evaluation of  $p^{\text{Wolfe}}$  costs about 100 microseconds. For the choice among proposed candidates (§ 7.3), again at most  $M$ , for each, we need to evaluate  $p^{\text{Wolfe}}$  and  $u_{\text{EI}}$  (Eq. 112) where the latter comes at the expense of evaluating two error functions. Since all of these computations have a fixed cost (in total some milliseconds on a laptop), the relative overhead becomes less the more expensive the evaluation of  $\nabla L_{\mathcal{B}}(w)$ .

The largest overhead is due to the estimation of  $\sigma_{f'}$  and lies outside of the actual line search routine. The computation of the sample variance  $\hat{\Sigma}$ , increase the cost of evaluating  $\nabla L_{\mathcal{B}}$  of a constant factor that, for neural networks, can be upper bounded by  $< 1.3$ , but is usually smaller in practice (Chapter 5). At the same time though, *all* exploratory experiments which very considerably increase the time spend when using  $\text{SGD}$  with a hand tuned learning rate schedule need not be performed anymore. In Section 7.5.1 we will also see that  $\text{SGD}$  using the probabilistic line search often needs less function evaluations to converge, which might lead to overall faster convergence than classic  $\text{SGD}$  in a single run.

### Memory Requirement

Vanilla  $\text{SGD}$ , at all times, keeps around the current optimization parameters  $w \in \mathbb{R}^N$  and the gradient vector  $\nabla L_{\mathcal{B}}(w) \in \mathbb{R}^N$ . In addition to this, the probabilistic line search needs to store the estimated gradient variances  $\hat{\Sigma}(w)$  (Eq. 118) of same size. The memory requirement of  $\text{SGD}+\text{PROBLS}$  is thus comparable to  $\text{ADAGRAD}$  or  $\text{ADAM}$ . If combined with a search direction other than  $\text{SGD}$ , always one additional vector of size  $N$  needs to be stored.

## 7.5 Experiments

This section reports on an extensive set of experiments to characterise and test the line search. The overall evidence from these tests is that the line search performs well and is relatively insensitive to the choice of its internal hyper-parameters as well as the mini-batch size. We performed experiments on two multi-layer perceptrons N-I and N-II; both were trained on two well known datasets MNIST and CIFAR-10.

- N-I: fully connected net with 1 hidden layer and 800 hidden units + biases, and 10 output units, sigmoidal activation functions and



a cross entropy loss. Structure without biases: 784-800-10. Many authors used similar nets and reported performances.<sup>10</sup>

- N-II: fully connected net with 3 hidden layers and 10 output units, tanh-activation functions and a squared loss. Structure without biases: 784-1000-500-250-10. Similar nets were also used for example in [91] and [132].
- MNIST [82]: multi-class classification task with 10 classes: hand-written digits in gray-scale of size  $28 \times 28$  (numbers '0' to '9'); training set size 60 000, test set size 10 000.
- CIFAR-10 [76]: multi-class classification task with 10 classes: color images of natural objects (horse, dog, frog, . . .) of size  $32 \times 32$ ; training set size 50 000, test set size 10 000; like other authors, we only used the "batch 1" sub-set of CIFAR-10 containing 10 000 training examples.

In addition, we train logistic regressors with sigmoidal output (N-III) on the following binary classification tasks:

- Wisconsin Breast Cancer Dataset (WDBC) [140]: binary classification of tumors as either 'malignant' or 'benign'. The set consist of 569 examples of which we used 169 to monitor generalization performing; thus 400 remain for the training set; 30 features describe for example radius, area, symmetry, et cetera. In comparison to the other datasets and networks, this yields a very low dimensional optimization problem with only 30 (+1 bias) input parameters as well as just a small number of datapoints.
- GISETTE [52]: binary classification of the handwritten digits '4' and '9'. The original  $28 \times 28$  images are taken from the MNIST dataset; then the feature set was expanded and consists of the original normalized pixels, plus a randomly selected subset of products of pairs of features, which are slightly biased towards the upper part of the image; in total there are 5000 features, instead of 784 as in the original MNIST. The size of the training set and test set is 6000 and 1000 respectively.
- EPSILON: synthetic dataset from the PASCAL Challenge 2008 for binary classification. It consists of 400 000 training set datapoint and 100 000 test set datapoints, each having 2000 features.

In the text and figures, SGD using the probabilistic line search will occasionally be denoted as SGD+PROBLS. Section 7.5.1 contains experiments on the sensitivity to varying gradient noise levels (mini-batch sizes  $|\beta|$ ) performed on both multi-layer perceptrons N-I and N-II, as well as on the logistic regressor N-III. Section 7.5.2 discusses sensitivity to the hyper-parameters choices introduced in Section 7.4.1

<sup>10</sup><http://yann.lecun.com/exdb/mnist/>

[91] Martens, "Deep learning via Hessian-free optimization," 2010

[132] Sutskever et al., "On the importance of initialization and momentum in deep learning," 2013

[82] LeCun et al., "Gradient-based learning applied to document recognition," 1998

[76] Krizhevsky and Hinton, "Learning multiple layers of features from tiny images," 2009

[140] Wolberg, Street, and Mangasarian, *UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set*, 2011

[52] Guyon et al., "Result Analysis of the NIPS 2003 Feature Selection Challenge," 2005



and Section 7.5.3 contains additional diagnostics on step size statistics. Each single experiment was performed 10 times with different random seeds that determined the starting weights and the mini-batch selection and seeds were shared across all experiments. We report all results of the 10 instances as well as means and standard deviations.

### 7.5.1 Varying Mini-batch Sizes

The noise level of the gradient estimate  $\nabla L_{\mathcal{B}}(w)$  and the loss  $L_{\mathcal{B}}(w)$  is determined by the mini-batch size  $|\mathcal{B}|$  and ultimately there should exist an optimal  $|\mathcal{B}|$  that maximizes the optimizer’s performance in CPU-time. In practice of course the performance is not necessarily linear in  $|\mathcal{B}|$  since it is upper bounded by the memory capacity of the hardware used. We suppose here, that the mini-batch size is chosen by the user; thus we test the line search with the default hyper-parameter setting ( $c_1 = 0.05$ ,  $c_2 = 0.5$ ,  $c_W = 0.3$ ,  $\alpha_{\text{ext}} = 1.3$ ) on four different mini-batch sizes:

- $|\mathcal{B}| = 10, 100, 200$  and  $1000$  (MNIST, CIFAR-10, and EPSILON)
- $|\mathcal{B}| = 10, 50, 100$ , and  $400$  (WDBC and GISETTE)

which correspond to increasing signal-to-noise ratios. Since the training set of WDBC only consists of 400 datapoints, the run with the largest mini-batch size of 400 in fact runs full-batch gradient descent on WDBC; this is not a problem, since—as discussed above—the probabilistic line search can also handle noise-free observations.<sup>11</sup> We compare to SGD-runs using a fixed step size which is typical for these architectures, and an annealed step size with annealing schedule  $\alpha_t = \alpha_0/t$ . Because annealed step sizes performed much worse than SGD+fixed step sizes, we will only report on the latter results in the plots.<sup>12</sup> Since classic SGD without the line search needs a hand crafted learning rate, we search on exhaustive logarithmic grids of

$$\begin{aligned}\alpha_{\text{SGD}}^{\text{N-I}} &= [10^{-5}, 5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}, 10^{-1}, 5 \cdot 10^{-1}] \\ \alpha_{\text{SGD}}^{\text{N-II}} &= [\alpha_{\text{SGD}}^{\text{N-I}}, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0] \\ \alpha_{\text{SGD}}^{\text{N-III}} &= [10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2].\end{aligned}$$

We run 10 different initialization for each learning rate, each mini-batch size, each net, and each dataset combination ( $10 \cdot 4 \cdot (2 \cdot 10 + 2 \cdot 17 + 3 \cdot 11) = 3480$  runs in total) for a large enough budget to reach convergence and report all numbers. Then, we perform the same experiments using the same seeds and setups with SGD using the probabilistic line search and compare the results. For SGD+PROBLS,  $\alpha_{\text{SGD}}$  is the initial learning rate  $\alpha_0$  which is used in the very first step. After that, the line search automatically adapts the learning rate, and shows no significant sensitivity to its initialization.

<sup>11</sup> Since the dataset size  $|\mathcal{D}|$  of WDBC is very small, we used the factor  $(|\mathcal{D}|-|\mathcal{B}|)/(|\mathcal{B}||\mathcal{D}|)$  instead of  $1/|\mathcal{B}|$  to scale the variances of Eq. 5.1. The former encodes sampling mini-batches  $\mathcal{B}$  with replacement, the latter without replacement; for  $|\mathcal{B}| \ll |\mathcal{D}|$  both factors are nearly identical.

<sup>12</sup> An example of annealed step size performance can be found in [89].

[89] Mahseeci and Hennig, “Probabilistic Line Searches for Stochastic Optimization,” 2015

Results of N-I and N-II on both, MNIST and CIFAR-10 are shown in Figures 44, 60, 61, and 62; results of N-III on GISETTE, WDBC and EPSILON are shown in Figures 63, 64, and 65 respectively. All instances (SGD and SGD+PROBLS) get the same computational budget (number of mini-batch evaluations) and not the same number of optimization steps. The latter would favour the probabilistic line search since, on average, a bit more than one mini-batch is evaluated per step. Likewise, all plots show performance measure versus the number of mini-batch evaluations, which is proportional to the computational cost.

All plots show similar results: While classic SGD is sensitive to the learning rate choice, the line search-controlled SGD performs as good, close to, or sometimes even better than the (in practice unknown) optimal classic SGD instance. In Figure 44, for example, SGD+PROBLS converges faster to a good test set error than the best classic SGD instance. In all experiments, across a reasonable range of mini-batch sizes  $|\mathcal{B}|$  and of initial  $\alpha_{\text{SGD}}$  values, the line search quickly identified good step sizes  $\alpha_t$ , stabilized the training, and progressed efficiently, reaching test set errors similar to those reported in the literature for tuned versions of these kind of architectures and datasets. The probabilistic line search thus effectively removes the need for exploratory experiments and learning-rate tuning.

REMARK: *The training error of SGD+PROBLS often plateaus earlier than the one of vanilla SGD, especially for smaller mini-batch sizes. This does not seem to impair the performance of the optimizer on the test set. We did not investigate this further, since it seemed like a nice natural annealing effect, but the exact causes are unclear for now. One explanation might be that the line search does indeed improve overfitting, since it tries to measure descent (by Wolfe conditions which rely on the noise-informed GP). This means that if—close to a minimum—successive acceptance decisions can not identify a descent direction anymore, diffusion might set in.*

## 7.5.2 Sensitivity to Design Parameters

Most, if not all, numerical methods make implicit or explicit choices about their hyper-parameters. Most of these are never seen by the user since they are either estimated at run time, or set by design to a fixed, approximately insensitive value. Well known examples are the step size in ordinary differential equation solvers [53, § 2.4], or the Wolfe parameters  $c_1$  and  $c_2$  of classic line searches. The probabilistic line search inherits the Wolfe parameters  $c_1$  and  $c_2$  from its classical counterpart as well as introducing two more: The Wolfe threshold  $c_W$  and the extrapolation factor  $\alpha_{\text{ext}}$ .  $c_W$  does not appear in the classical formulation since the objective function can be evaluated exactly and

[53] Hairer, Nørsett, and Wanner, *Solving Ordinary Differential Equations I – Nonstiff Problems*, 1987

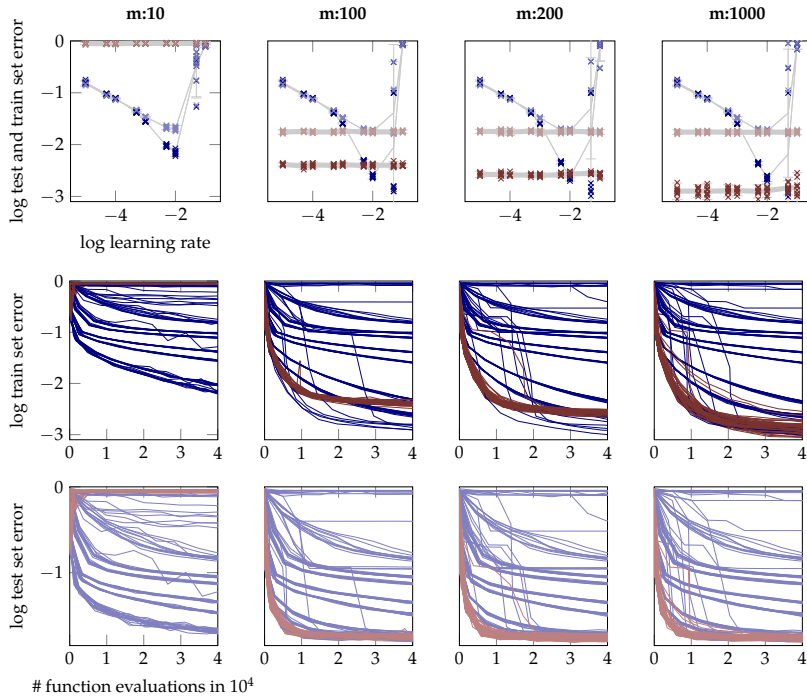


Figure 44: Performance of N-II on MNIST for varying mini-batch sizes. Top: final logarithmic test set and train set error after 40 000 function evaluations of training versus a large range of learning rates each for 10 different initializations. SGD-runs with fixed learning rates ( $\times$ , test/train); SGD+PROBLS-runs ( $\circ$ , test/train). Means and two standard deviations for each of the 10 runs in gray. Columns from left to right: different mini-batch sizes  $|\mathcal{B}| = 10, 100, 200$  and 1000 which correspond to decreasing noise in the gradient observations. Not surprisingly, the performance of SGD-runs with a fixed step size are very sensitive to the choice of this step size. SGD+PROBLS adapts initially mis-scaled step sizes and performs well across the whole range of initial learning rates. Middle and bottom: Evolution of the logarithmic error rates for all SGD-runs (—/—, test/train) and SGD+PROBLS-runs (---/---, test/train) versus # function evaluations. For mini-batch sizes of  $m = 100, 200$  and 1000 *all* instances of SGD using the probabilistic line search reach the same best test set error. Vanilla SGD occasionally reaches smaller train errors but this advantage does not seem to translate to a better test set error. For very small mini-batch sizes ( $m = 10$  and first column in the plot) the line search performs poorly on this architecture, most likely because of the variance estimation becoming too inaccurate (see Chapter 5.2).

the Wolfe probability is binary (either fulfilled or not). While  $c_W$  is thus a natural consequence of allowing the line search to model noise explicitly, the extrapolation factor  $\alpha_{\text{ext}}$  is the result of the line search favoring shorter steps, which we will discuss below in more detail, but most prominently because of bias in the line search’s first gradient observation.

In the following sections we will give an intuition about the task of the most influential design parameters  $c_2$ ,  $c_W$ , and  $\alpha_{\text{ext}}$ , discuss how they affect the probabilistic line search, and validate good design choices through exploring the parameter space and showing insensitivity to most of them. All experiments on hyper-parameter sensitivity were performed training N-II on MNIST with mini-batch size  $|\mathcal{B}| = 200$ . For a full grid-search of the parameter space  $c_W$ - $c_2$ - $\alpha_{\text{ext}}$  we performed 4950 runs in total with 495 different parameter combinations. All results are reported.

#### Wolfe-II Parameter $c_2$ and Wolfe Threshold $c_W$

As described in Section 7.4.1,  $c_2$  encodes the strictness of the curvature condition W-II. Pictorially speaking, a larger  $c_2$  extends the range of acceptable gradients (■-area in the lower part of Figure 41) and leads to a lenient line search while a smaller value of  $c_2$  shrinks this area, leading to a stricter line search.  $c_W$  controls how certain we want to be, that the Wolfe conditions are actually fulfilled, i. e., how much of the mass of  $p(a, b)$  need to lie in ■. An overly strict line

search (e. g.,  $c_W = 0.99$  and/ or  $c_2 = 0.1$ ), will still be able to optimize the objective function well, but will waste evaluations at the expense of efficiency. Figure 45 explores the  $c_2$ - $c_W$  parameter space, while keeping  $\alpha_{\text{ext}}$  fixed at 1.3. The left column shows final test and train set error, the right column the average number of function evaluations per line search, both versus different choices of Wolfe parameter  $c_2$ . The left column thus shows the overall performance of the optimizer, while the right column is representative for the efficiency of the line search. Intuitively, a line search which is minimally invasive (only corrects the learning rate, when it is really necessary) is preferred. Rows in Figure 45 show the same plot for different choices of the Wolfe threshold  $c_W$ .

The effect of strict  $c_2$  can be observed clearly in Figure 45 where for smaller values of  $c_2 < \approx 0.2$  the average number of function evaluations spend in one line search goes up slightly in comparison to looser choices of  $c_2$ , while still a very good performance is reached in terms of train and test set error. Likewise, the last row of Figure 45 for the extreme value of  $c_W = 0.99$  (demanding 99% certainty about the validity if the Wolfe conditions), shows significant loss in computational efficiency having an average number of 7 function evaluations per line search. Besides losing efficiency, it is still optimizing the objective well. Lowering this threshold a bit to 90% increases the computational efficiency of the line search to be nearly optimal again. Ideally, we want to trade off the desiderata of being strict enough to reject too small and too large steps that prevent the optimizer to converge, but being lenient enough to allow all other reasonable steps, thus increasing computational efficiency. The values  $c_W = 0.3$  and  $c_2 = 0.5$ , which are adopted in our current implementation are marked as dark red vertical lines in Figure 45.

#### *Extrapolation Factor $\alpha_{\text{ext}}$*

The extrapolation parameter  $\alpha_{\text{ext}}$ , introduced in Section 7.4.1, pushes the line search to try a larger learning rate first, than the one which was accepted in the previous step. Figure 46 is structured like Figure 45, but this time explores the line search sensitivity in the  $c_2$ - $\alpha_{\text{ext}}$  parameter space (abscissa and rows respectively) while keeping  $c_W$  fixed at 0.3. Unless we choose  $\alpha_{\text{ext}} = 1.0$  (no step size increase between steps) in combination with a lenient choice of  $c_2$  the line search performs well. For now we adopt  $\alpha_{\text{ext}} = 1.3$  as default value which again is shown as dark red vertical line in Figure 46.

The introduction of  $\alpha_{\text{ext}}$  might seem arbitrary at first, but is a necessity and well-working fix because of a few shortcomings of the current design. First, the curvature condition W-II is the single condition that prevents too small steps and pushes optimization progress. On the

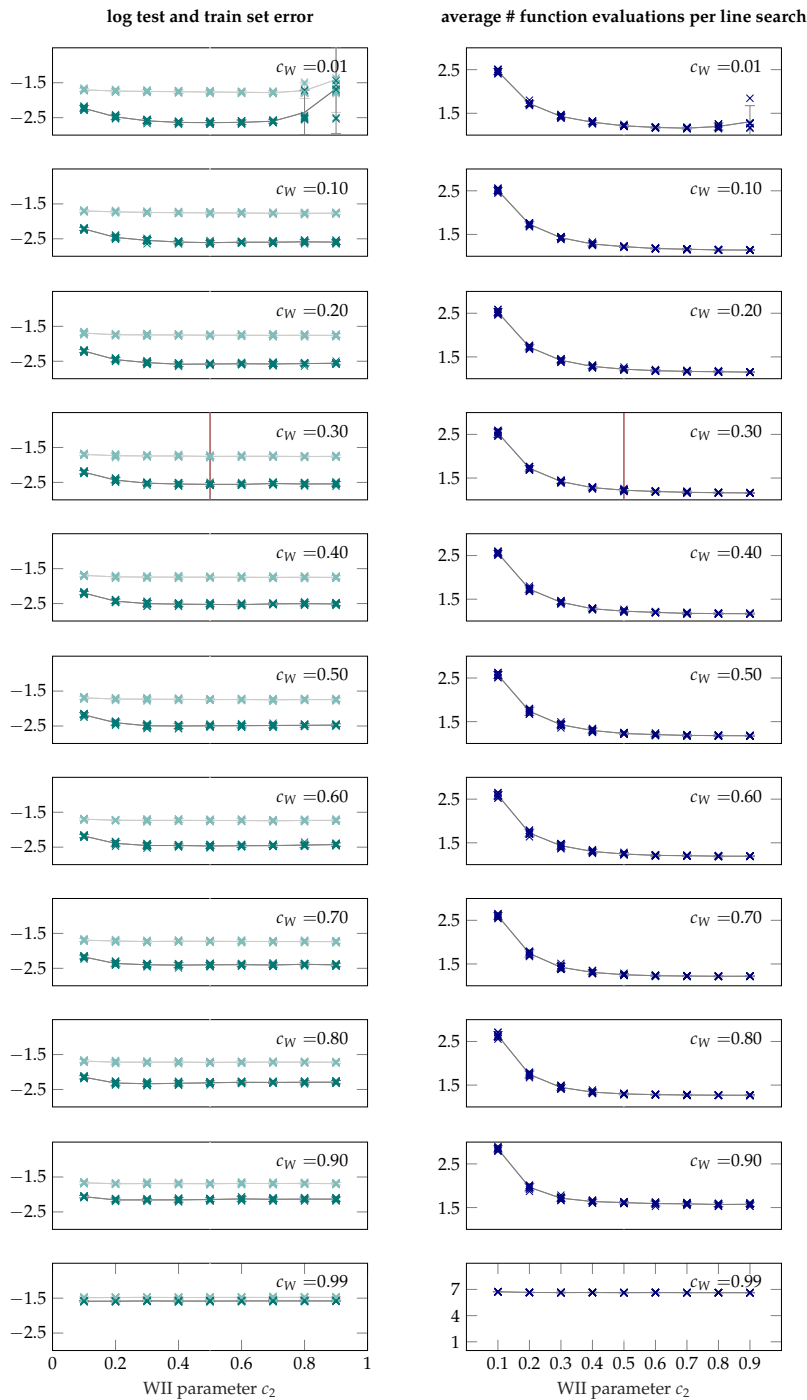


Figure 45: Sensitivity to varying hyper-parameters  $c_2$ , and  $c_W$ . Runs were performed training N-II on MNIST with mini-batch size  $|\mathcal{B}| = 200$ . For each parameter setting 10 runs with different initializations were performed. Left column: logarithmic test and train set error ( $\times/\times$ ) after 40 000 function evaluations; mean and  $\pm$  two standard deviations of the 10 runs in gray. Right Column: average number of function evaluations per line search ( $\times$ ). A low number indicates an efficient line search procedure (perfect efficiency at 1). For most parameter combinations this lies around  $\approx 1.3 - 1.5$ . Only at extreme parameter values for example  $c_W = 0.99$ , which amounts to imposing nearly absolute certainty about the Wolfe conditions, or  $c_2 < 0.2$  which demands and unnecessarily large decrease in gradient magnitude, the line search becomes less efficient. Adopted parameters as vertical line ( $\text{---}$ ) at  $c_W = 0.3$  and  $c_2 = 0.5$

other hand both W-I and W-II simultaneously penalize too large steps (Figure 21 for a sketch). This is not a problem in case of deterministic observation ( $\sigma_f, \sigma_{f'} \rightarrow 0$ ), where W-II undoubtedly decides if a gradient is still too negative. Unless W-II is chosen very tightly (small  $c_2$ ) or  $c_W$  unnecessarily large (both choices, as discussed above, are undesirable), in the presence of noise,  $p^{\text{Wolfe}}$  will thus be more reliable in preventing overshooting than pushing progress. The first row of Figure 46 illustrates this behavior, where the performance drops somewhat if no extrapolation is done ( $\alpha_{\text{ext}} = 1.0$ ) in combination with a looser version of W-II (larger  $c_2$ ).

Another factor that contributes towards accepting small rather than larger learning rates is a bias introduced in the first observation of the line search at  $t = 0$ . Observations  $y'(t)$  that the GP gets to see are projections of the gradient sample  $\nabla L_B(t)$  onto the search direction  $p = -\nabla L_B(0)$ . Since the first observations  $y'(0)$  is computed from the same mini-batch as the search direction (not doing this would double the optimizer's computational cost) a bias is introduced. Since the scale parameter  $\theta$  of the Wiener process is implicitly set by  $y'(0)$  (§ 7.4.1), the GP becomes more uncertain at unobserved points than it needs to be; or alternatively expects the 1D-gradient to cross zero at smaller steps, and thus underestimates a potential learning rate. The posterior at observed positions is little affected. The over-estimation of  $\theta$  rather pushes the posterior towards the likelihood (since there is less model to trust) and thus still gives a reliable measure for  $f(t)$  and  $f'(t)$ . The effect on the Wolfe conditions is similar. With  $y'(0)$  biased towards larger values, the Wolfe conditions, which measure the drop in projected gradient magnitude, are thus prone to accept larger gradients combined with smaller function values, which again is met by making small steps. Ultimately though, since candidate points at  $t^{\text{cand}} > 0$  that are currently queried for acceptance, are always observed and unbiased, this can be controlled by an appropriate design of the Wolfe factor  $c_2$  (§ 7.4.1 and § 7.5.2) and of course  $\alpha_{\text{ext}}$ .

#### *Full Hyper-Parameter Search: $c_W$ - $c_2$ - $\alpha_{\text{ext}}$*

An exhaustive performance evaluation on the whole  $c_W$ - $c_2$ - $\alpha_{\text{ext}}$ -grid is shown in Appendix C.2 in Figures 70-66 and Figures 71-81. As discussed above, it shows the necessity of introducing the extrapolation parameter  $\alpha_{\text{ext}}$  and shows slightly less efficient performance for obviously undesirable parameter combinations. In a large volume of the parameter space, and most importantly in the vicinity of the chosen design parameters, the performance of the line search is stable and comparable to carefully hand tuned learning rates.

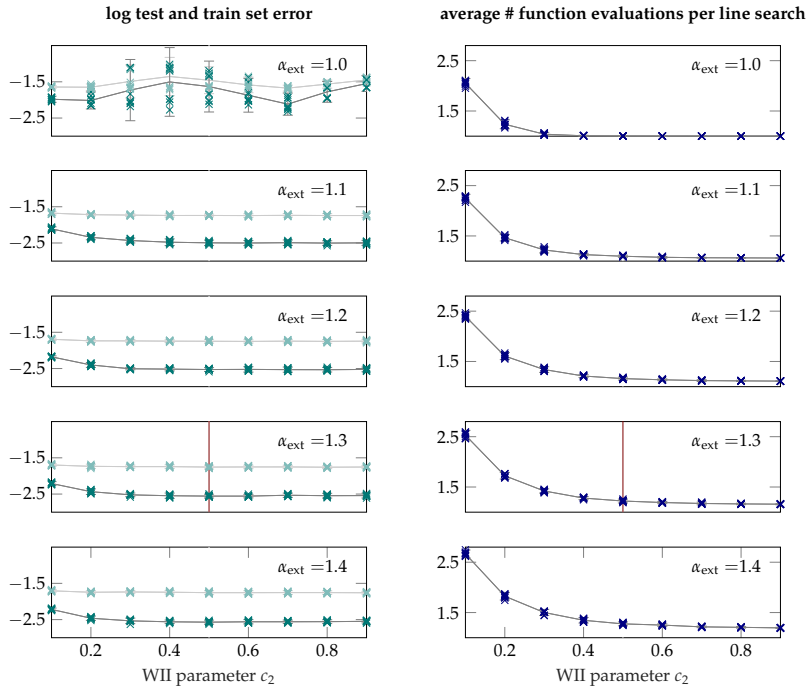


Figure 46: Sensitivity to varying hyper-parameters  $c_2$ , and  $\alpha_{\text{ext}}$ . Plot and colors as in Figure 45 but this time for varying  $\alpha_{\text{ext}}$  instead of  $c_W$ . Right Column: Again a low number indicates an efficient line search procedure (perfect efficiency at 1). For most parameter combinations this lies around  $\approx 1.3 - 1.5$ . Only at extreme parameter values, for example  $\alpha_{\text{ext}} = 1.0$ , which amounts to no extrapolation at all in between successive line searches, the line search performs poorer. The hyper-parameters adopted in the line search implementation are indicated as vertical line (—) at  $\alpha_{\text{ext}} = 1.3$  and  $c_2 = 0.5$ .

#### Safeguarding against Mis-scaled GPs: $\theta_{\text{reset}}$

For completeness, we performed an additional experiment on the threshold parameter that is denoted by  $\theta_{\text{reset}}$  in the pseudo-code in Appendix D and safeguards against GP mis-scaling. Because the line search models observation noise, it also needs to model the expected variability of the 1D-objective along the search direction, which is described by the kernel scale parameter  $\theta$ . Setting this hyper-parameter is implicitly done by scaling the observation input, by assuming a similar scale than in the previous line search (§ 7.4.1), and thus information of a well-scaled learning rate is carried over from one iteration to the next. If, for some reason, the previous line search accepted an unexpectedly large or small step (what this means is encoded in  $\theta_{\text{reset}}$ ) the GP scale  $\theta$  for the next line search is reset to an exponential running average of previous scales, represented by  $\alpha_{\text{stats}}$  in the pseudo-code. This occurs very rarely (for the default value  $\theta_{\text{reset}} = 100$  the reset occurred in 0.02% of all line searches), but it is necessary to safeguard against extremely mis-scaled GPs.  $\theta_{\text{reset}}$  therefore is not part of the probabilistic line search model as such, but prevents mis-scaled GPs due to some unlucky observation or sudden extreme change in the learning rate. Figure 47 shows performance of the line search for  $\theta_{\text{reset}} = 10, 100, 1000$  and 10 000 showing no significant performance change. We adopted  $\theta_{\text{reset}} = 100$  in our implementation since this is the expected and desired multiplicative (inverse) factor to maximally vary the learning rate in one single step.



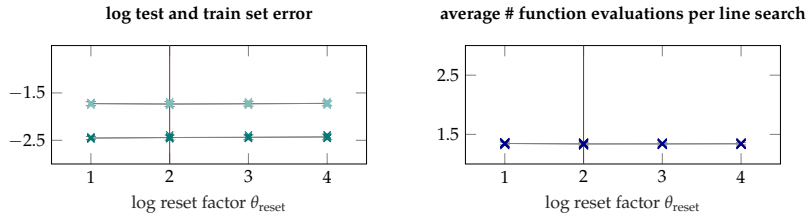


Figure 47: Sensitivity to hyper-parameter  $\theta_{\text{reset}}$ . Plot and colors as in Figure 46. Adopted parameter as vertical line (—) at  $\theta_{\text{reset}} = 100$ . Resetting the GP scale occurs very rarely. For example for  $\theta_{\text{reset}} = 100$  the reset occurred in 0.02% of all line searches.

### 7.5.3 Candidate Selection and Learning Rate Traces

The probabilistic line search, chooses among candidates by computing values of the acquisition function  $u_{\text{EI}}(t_i^{\text{cand}}) \cdot p^{\text{Wolfe}}(t_i^{\text{cand}})$  at every candidate point  $t_i^{\text{cand}}$ ; then it selects the one with the highest value and evaluated the objective  $\mathcal{L}_{\mathcal{B}}, \nabla L_{\mathcal{B}}$  there (§ 7.3). The Wolfe probability  $p^{\text{Wolfe}}$  actually encodes precisely what kind of point we want to find and incorporates both conditions (W-I and W-II) about the function value *and* the gradient (§ 7.4). However,  $p^{\text{Wolfe}}$  does not have very desirable exploration properties. Since the uncertainty of the GP grows to ‘the right’ of the last observation, the Wolfe probability quickly drops to a low, approximately constant (non-zero) value there (Figure 40). Also  $p^{\text{Wolfe}}$  is partially allowing for undesirably short steps (§ 7.5.2). The expected improvement  $u_{\text{EI}}$ , on the other hand, is a well studied acquisition function of Bayesian optimization trading off exploration and exploitation. It aims to globally find a point with a function value lower than a current best guess. Though this is a desirable property also for the probabilistic line search, it is lacking the information that we are seeking a point that also fulfills the W-II curvature condition. This is evident in Figure 40 where  $p^{\text{Wolfe}}$  significantly drops at points where the objective function is already evaluated but  $u_{\text{EI}}$  does not. In addition, we do not need to explore the positive  $t$  space to an extend, the expected improvement suggests, since the aim of a line search is just to find a good, acceptable point at positive  $t$  and not the globally best one. The product of both acquisition function  $u_{\text{EI}} \cdot p^{\text{Wolfe}}$  is thus a trade-off between exploring enough, but still preventing too much exploitation in obviously undesirable regions. In practice though, we found that all three choices ((i)  $u_{\text{EI}} \cdot p^{\text{Wolfe}}$ , (ii)  $u_{\text{EI}}$  only, (iii)  $p^{\text{Wolfe}}$  only) perform comparable. The following experiments were all performed training N-II on MNIST; only the mini-batch size might vary as indicated.

Figure 48 compares all three choices for mini-batch size  $|\mathcal{B}| = 200$  and default design parameters. The top plot shows the evolution of the logarithmic test and train set error (for plot and color description see Figure caption). All test and train set error curves respectively bundle up (only lastly plotted clearly visible). The choice of acquisition



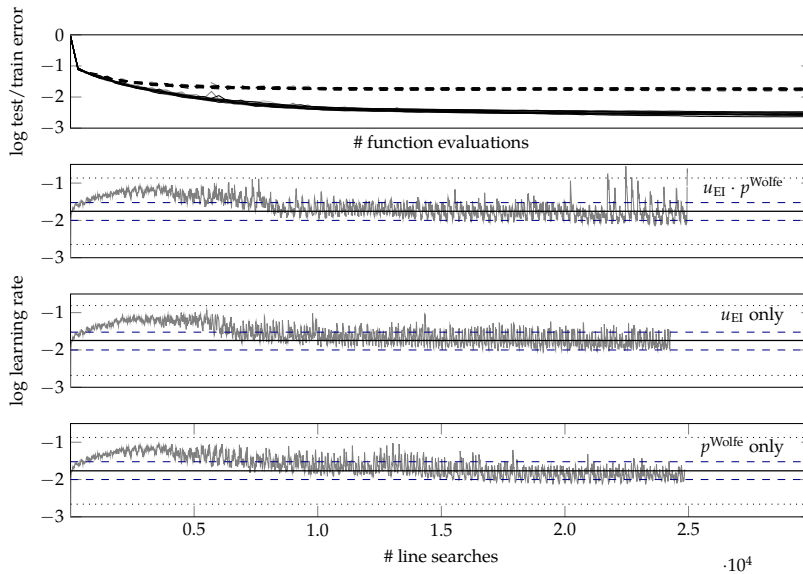


Figure 48: Different choices of acquisition functions. Top: evolution of the logarithmic test (---/---/---) and train set error (—/—/—) for  $u_{EI} \cdot p^{Wolfe} / u_{EI} / p^{Wolfe}$  respectively (only lastly plotted clearly visible since the curves are very similar). Different lines of the same color correspond to different seeds. Rows 2-4: Learning rate traces of a single seed (—, acquisition function in legend). For plotting purposes the curves were smoothed and thinned out. Mean of the raw, non-smoothed values of accepted learning rates across the whole optimization process (—);  $\pm$  two standard deviations (.....); a range of well performing constant learning rates (---).

function thus does not change the performance here. Rows 2-4 of Figure 48 show learning rate traces of a single seed. All three curves show very similar global behavior. First, the learning rate grows, then drops again, and finally settles around the best found constant learning rate. This is intriguing since on average a larger learning rate seems to be better at the beginning of the optimization process, which then later drops again to a smaller one. This might also explain why `SGD+PROBLS` in the first part of the optimization progress outperforms vanilla `SGD` (Figure 44). Runs, that use just slightly larger constant learning rates than the best performing constant one (above --- in Figure 48) were failing after a few steps. This shows that there is some non-trivial adaptation going on, not just globally, but locally at every step.

Figure 49 shows traces of accepted learning rates for different mini-batch sizes  $|\mathcal{B}| = 100, 200, 1000$ . Again, the global behavior is qualitatively similar for all three mini-batch sizes. For the largest mini-batch size  $|\mathcal{B}| = 1000$  (last row of Figure 49) the probabilistic line search accepts a larger learning rate (on average and in absolute value) than for the smaller mini-batch sizes  $|\mathcal{B}| = 100$  and  $200$ , which is in agreement with practical experience and theoretical findings ([64, § 4 and 7], [48, § 9.1.3], [7]).

Figure 50 shows traces of the scaled noise levels  $\sigma_f$  and  $\sigma_{f'}$  and the average number of function evaluations per line search for different noise levels (—/—/— for  $|\mathcal{B}| = 1000, 200$ , and  $100$  respectively); same colors show the same setup but different seeds. The average number of function evaluations rises very slightly to  $\approx 1.5 - 2$  for mini-batch size  $|\mathcal{B}| = 1000$  towards the end of the optimization pro-

[64] Hinton, “A Practical Guide to Training Restricted Boltzmann Machines,” 2012

[48] Goodfellow, Bengio, and Courville, *Deep Learning*, 2016

[7] Balles, Romero, and Hennig, “Coupling Adaptive Batch Sizes with Learning Rates,” 2016

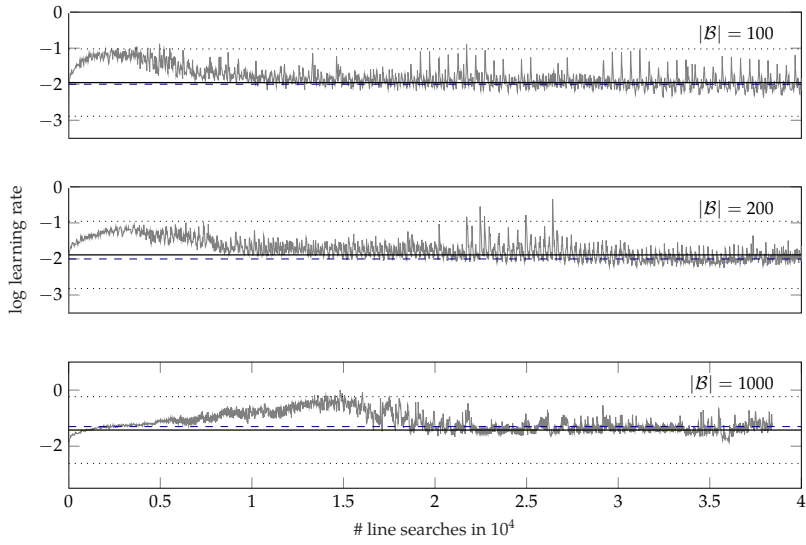


Figure 49: Traces of accepted logarithmic learning rates. All runs are performed with default design parameters. Different rows show the same plot for different mini-batch sizes of  $|\mathcal{B}| = 100, 200$  and  $1000$ . Plots and smoothing as in rows 2-4 of Figure 48 (details in text).

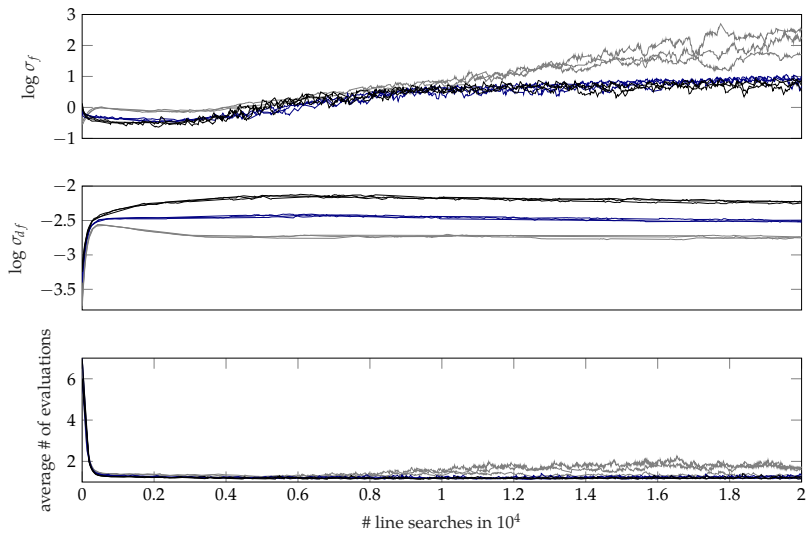


Figure 50: Traces of logarithmic noise levels  $\sigma_f$  (top),  $\sigma_{f,l}$  (middle) and average number of function evaluations per line search (bottom). Setup and smoothing as in Figure 49. Different colors correspond to different mini-batch sizes (—/—/— for  $|\mathcal{B}| = 1000, 200$ , and  $100$  respectively). Curves of the same color correspond to different seeds (3 shown).

cess, in comparison to  $\approx 1.5$  for  $|\mathcal{B}| = 100,200$ . This seems counter intuitive in a way, but since larger mini-batch sizes also observe smaller values and gradients, especially towards the end of the optimization process, the relative noise levels might actually be larger. (Although the curves for varying  $|\mathcal{B}|$  are shown versus the same abscissa, the corresponding optimizers might be in different regions of the loss surface, especially  $|\mathcal{B}| = 1000$  probably reaches regions of smaller absolute gradients). At the start of the optimization the average number of function evaluations is high, because the initial default learning rate is small ( $10^{-4}$ ) and the line search extends each step multiple times.

## 7.6 Conclusion and Outlook

The line search paradigm widely accepted in deterministic optimization can be extended to noisy settings. Our design combines existing principles from the noise-free case with ideas from Bayesian optimization, adapted for efficiency. We arrived at a lightweight “black-box” algorithm that exposes no parameters to the user. Empirical evaluations so far show compatibility with the SGD search direction and viability for logistic regression and multi-layer perceptrons. The line search effectively frees users from worries about the choice of a learning rate: Any reasonable initial choice will be quickly adapted and lead to close to optimal performance. Our matlab implementation can be found at <http://tinyurl.com/probLineSearch>.

In the future, it would be interesting to:

- Test the line search also on different neural network architectures (ReLU-activations, CNNs, deeper architectures et cetera), as well as larger datasets (CIFAR-100[76], ImageNet[28], ...). A main challenge will be the efficient implementation of the sample variances  $\hat{\Sigma}(w)$  into common auto-differentiation frameworks to make it accessible for wider use. Works towards this goal include [6] and [5].
- Test the line search on problems other than neural networks which also use empirical risks as loss functions as in Eq. 28, such as those arising in stochastic variational inference [67].
- Combine the line search with search directions other than SGD. Though in principle this should be possible, care must be taken about potentially ill-posed Wolfe conditions in case that  $\nabla L_{\mathcal{B}}(0)^{\top} p > 0$ .
- Examine and analyze how the line search interacts with typical regularization strategies as discussed in Chapter 2.2.2, such as Dropout or batchNorm.

[76] Krizhevsky and Hinton, “Learning multiple layers of features from tiny images,” 2009

[28] Deng et al., “ImageNet: A Large-Scale Hierarchical Image Database,” 2009

[6] Balles, Mahseerci, and Hennig, “Automating Stochastic Optimization with Gradient Variance Estimates,” 2017

[5] Balles and Hennig, “Follow the Signs for Robust Stochastic Optimization,” 2017

[67] Hoffman et al., “Stochastic variational inference,” 2013

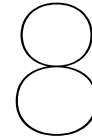
- Couple the line search to methods that adapt the mini-batch size  $|\mathcal{B}|$  during the optimization run, such as e. g., [7].
- Improve the line search, possibly by defining a different set of Wolfe-like conditions, which do not require the extrapolation parameter  $\alpha_{\text{ext}}$  anymore, and are more suited to the noisy setting. Additionally, define conditions which are not ill-defined for positive initial GP-means of the gradient.
- Extend the concept to situations where only gradients, but no losses are robustly available.
- Combine the lines search with methods that provide a Gaussian posterior over gradients and function values, rather than a likelihood, such as those discussed in the next two chapter.

[7] Balles, Romero, and Hennig, "Coupling Adaptive Batch Sizes with Learning Rates," 2016

Part IV

## Kalman Filtering for Stochastic Optimization





## First-Order Filter for Gradients

---

**T**HIS Chapter develops and derives formulas for Kalman filtering on gradients. The main goal is to design a general probabilistic framework for first-order optimization with stochastic gradients of arbitrary noise levels; and then to draw connection to existing, classic stochastic and deterministic optimizers like GD, SGD, and MOMENTUM-methods. This provides further interpretation of these methods, especially of the smoothing constants of the latter. We will represent the dynamic of the true but unknown gradient function  $\nabla L_{\mathcal{D}}(w)$  of the empirical risk with a Gauss-Markov process, defined only on the optimization path. It will turn out that transitions between Kalman states actually precisely encode the exponential smoothing done manually in classic methods and thus—adopting this interpretation—directly support their use. The conducted experiments include proof-of-concepts for two instances of a novel class of first-order methods that we call KFGRAD (for ‘Kalman filtering on gradients’); they perform similar to existing methods, but with a higher degree of automation, flexibility and extensibility. Additionally, diagnostic results on the magnitude of learned Kalman-gains support the hypothesis that smoothing in momentum-SGD can entirely or to a large portion be explained to benefit stochastic noise reduction due to mini-batching instead of geometrical smoothing, the latter being often advocated as the main benefit of these methods.

### 8.1 A Model for Once-Differentiable Functions

Classic first-order optimization algorithms (e. g., GD or SGD, Sections 2.3 and 2.4) assume only *one* derivative of the objective function  $L_{\mathcal{D}}(w)$  and locally perform *linear* approximations. Higher derivatives are discarded and the optimizer has no knowledge about them. In reality  $L_{\mathcal{D}}(w)$  might have more than one derivative but it is often more efficient to conduct crude and cheap, but therefore many iterations. This section constructs a probabilistic equivalent to classic first-order methods by modeling  $\nabla L_{\mathcal{D}}(w)$  with a stochastic process whose samples are only once-differentiable with probability one, as well as Markov, similar to the update rules of most iterative optimizers. We will specify the process by a stochastic differential equation (SDE) and an initial condition. The stochastic contribution of the former serves as a vehicle to express the unknown change in the gradient function  $\nabla L_{\mathcal{D}}(w)$ .

We then derive formulas which describe the discretized solution of this SDE that will lead to concrete, deterministic update rules of a new class of viable optimizers. This general but concise framework also allows for straightforward interpretation of parameters of the process, and consequently of the optimization routine, which eases the on-line estimation of these parameters by established and well-known techniques. We first start by defining the process.

Section 1.4.1 introduced the Kalman filter equations and their underlying continuous, stochastic model, but did not make a statement about the specific forms of the drift matrix  $F$ , the diffusion matrix  $L$ , nor what the state  $x$  encodes. In the same sense as classic first-order methods, we will assume here that  $L_{\mathcal{D}}(w)$  has only *one* derivative we know of, thus encoding a very simple, minimally-assertive structure on  $L_{\mathcal{D}}$ . The state  $x \in \mathbb{R}^N$  represents the unknown gradient  $\nabla L_{\mathcal{D}}$  of the empirical risk. The most simple SDE that follows Eq. 17 is a Wiener process on  $x$ :

$$dx = 0_{N \times 1} + d\beta. \quad (119)$$

By comparing to Eq. 17, we can read off  $F = 0_{N \times N}$  and  $L = I_{N \times N}$ . Combining this with Eq. 19 yields the diffusion covariance  $Q \in \mathbb{R}^{N \times N}$  and the transition matrix  $A \in \mathbb{R}^{N \times N}$ :

$$\begin{aligned} A_t &= \sum_{k=0}^{\infty} \frac{(0_{N \times N})^k}{k!} = I_{N \times N} \\ Q_t &= \int_{\tau_t}^{\tau_{t+1}} \exp(0_{N \times N}(\tau_{t+1} - \kappa)) I_{N \times N} q I_{N \times N} \exp^{\top}(0_{N \times N}(\tau_{t+1} - \kappa)) d\kappa = \int_{\tau_t}^{\tau_{t+1}} q d\kappa = q \Delta\tau, \end{aligned} \quad (120)$$

where  $q \in \mathbb{R}^{N \times N}$  is the positive definite intensity matrix introduced in Section 1.4.1. The predictive Kalman equations simplify to:

$$\begin{aligned} m_{t+1-} &= m_t \\ P_{t+1-} &= q \Delta\tau + P_t. \end{aligned} \quad (121)$$

Eq. 121 is intuitive: Since the drift on the state is zero ( $F = 0$ ), and successive Gaussian increments ( $t \rightarrow t + 1$ ) are independent of each other and of same intensity  $q$ , the estimated predictive state  $m_{t+1-}$  is the same as the current one  $m_t$ ; but the covariance  $P_{t+1-}$  grows according to a Wiener process proportional to the traveled distance  $\Delta\tau = \|w_{t+1} - w_t\|$ .

The predictive Kalman filter equations in 121 represent a multi-output distribution over gradient elements  $\nabla L_{\mathcal{D}}(w)$ , only on the one-dimensional optimization path, and not on the whole weight space of  $ws$ . The optimization path, in this context, are all  $ws$  which lie on the the ordered sequence of successive piece-wise linear interpolations between  $w_i$  and  $w_{i+1}$ , for  $i = 0, \dots, t - 1$ . This means that, between



two nodes  $w_i$  and  $w_{i+1}$ , the path can be parametrized with a scalar  $\tau \in [0, \|w_{i+1} - w_i\|]$ .

### Model of Global Change

Importantly, Eq. 119 does not encode a globally constant derivative of the loss  $L_{\mathcal{D}}(\tau)$  (just the predictive estimator  $m_{t+1-}$  of this particular sDE is), but a derivative *function*  $x$  that may change with location  $w(\tau)$ ; and we become more uncertain about its value the farther we move away from the current location. The magnitude of this growing uncertainty is encoded by  $Q$ . The ‘stochastic’ part of the sDE is hence just a vehicle to express our lack of knowledge on how the function  $\nabla L_{\mathcal{D}}(\tau)$  evolves, no pseudo-random numbers or physical random processes need to occur. Disregarding the Gaussian form for a moment, hence, more sloppily, and rather associative, Eq. 119 can be thought of as a Lipschitz-type notion on gradients, not in terms of an absolute less-or-equal statement, but rather that the relative probability of finding gradient elements  $i$  further away from  $m_{t+1-}^i \pm 2(\Delta\tau q^{ii} + P_t^{ii})^{1/2}$  is very low (although not impossible) *given* an already uncertain estimator for the previous gradient.<sup>1</sup> The possible change of gradients is encoded by the intensity *matrix*  $q$ , which, like  $\mathcal{L}$ , is a property of the loss  $L_{\mathcal{D}}(w)$  and needs to be learned, or adapted while the optimizer is running in case it is unknown (§ 8.2.2).

### Measurements

In mini-batch settings, the state  $x$  is never observed exactly (this would only be possible by evaluating  $\nabla L_{\mathcal{D}}$  on the full dataset), but only noise corrupted versions of it. As motivated in Chapter 5, we use Gaussian distributions on sub-sampled gradients  $y = \nabla L_S$  (e. g., a mini-batch gradient  $\nabla L_B$ ) with noise covariance  $R_t = \Sigma(w_t)/|\mathcal{S}| \in \mathbb{R}^{N \times N}$ ; then the measurement model is straightforward: Since the full state  $x$  is observed, albeit noisy, the measurement matrix  $H = I_{N \times N}$  is the identity. The updated Kalman equations are thus:

$$\begin{aligned} G_{t+1} &= P_{t+1-} + R_{t+1} \\ g_{t+1} &= P_{t+1-} G_{t+1}^{-1} \\ m_{t+1} &= [I - g_{t+1}]m_{t+1-} + g_{t+1} \nabla L_S(w_{t+1}) \\ P_{t+1} &= [I - g_{t+1}]P_{t+1-}. \end{aligned} \tag{122}$$

Eq. 121 together with Eq. 122 provide the base-equations for *filtering on gradients* where the hypothesis class models *one* derivative and no drift. In principle, other processes than the one defined by Eq. 119 can be motivated, which e. g., incorporate knowledge about the behavior of successive gradients chosen by the optimizers’s routine.<sup>2</sup> The algorithm defined by Eqs. 121 and 122 has general cost cubic in  $N$  (due to

<sup>1</sup> In other words, the expected squared distance between two states is:

$$\begin{aligned} \mathbb{E}_{p(x_{t+1}|x_t)} [\|x_{t+1} - x_t\|^2] \\ = \text{tr}[q] \|w_{t+1} - w_t\|. \end{aligned}$$

Compare to a Lipschitz statement of the form  $\|\nabla L_{\mathcal{D}}(w_{t+1}) - \nabla L_{\mathcal{D}}(w_t)\| \leq \mathcal{L} \|w_{t+1} - w_t\|$ .

<sup>2</sup> Assuming a ‘well-working’ optimizer with well-tuned hyper-parameters, gradient elements will tends towards zero rather than arbitrarily diffusing away from zero. Thus a zero-mean reverting process might be a suitable prior choice as well. We will not explore this further here.

the matrix inversion of  $G$ ), assuming that the hyper-parameters  $R$  and  $q$  are given.

## 8.2 Diagonal Approximations

For very high dimensional optimization problems, it is often only feasible to store objects that have memory requirements linear in the dimensionality  $N$ . The covariance matrices  $R$ ,  $q$ ,  $P$ , and  $P_-$  introduced in Section 8.1, though, are of squared size  $N^2$ . One way of compressing the information contained in these matrices, is to only store their diagonals. Though this is a very drastic simplification, it is a common and often well working concept in stochastic optimization.<sup>3</sup> Diagonal covariance matrices yield parallelizable prediction and update rules for each component of the state, allowing for sped-up computations.<sup>4</sup> The Kalman equations for the  $i^{\text{th}}$  component are then scalar:

$$\begin{aligned} m_{t+1-}^i &= m_t^i \\ P_{t+1-}^{ii} &= q^{ii} \Delta\tau + P_t^{ii} \\ g_{t+1}^{ii} &= \frac{P_{t+1-}^{ii}}{P_{t+1-}^{ii} + R_{t+1}^{ii}} \\ m_{t+1}^{ii} &= [1 - g_{t+1}^{ii}] m_{t+1-}^{ii} + g_{t+1}^{ii} \nabla L_{\mathcal{D}}(w_{t+1})^{ii} \\ P_{t+1}^{ii} &= [1 - g_{t+1}^{ii}] P_{t+1-}^{ii} = g_{t+1}^{ii} R_{t+1}^{ii}. \end{aligned} \tag{123}$$

The above equations already define a viable algorithm for filtering on gradients for some given intensity matrix  $q = \text{diag}[q^{ii}]$  and noise variances  $R = \text{diag}[R^{ii}]$ . As discussed in Chapter 5, the variances  $R^{ii}$  can be estimated efficiently within a mini-batch; Section 8.2.2 will introduce a way of also estimating the intensities  $q^{ii}$  with little computational overhead at runtime. The memory requirement is less than what Eq. 123 suggests, since  $g_{t+1}$ ,  $P_{t+1-}$ ,  $m_{t+1-}$  are only intermediate algebraic steps which do not need to be computed or stored explicitly.

### 8.2.1 Connections

Consider an optimizer which updates the weights with the filtered gradients  $m_t$ :

$$w_{t+1} = w_t - \alpha_t m_t. \tag{124}$$

The update rule for  $m_t$  for a *constant* gain  $g = g_t^{ii} = \text{const.} \in (0, 1)$  for all  $i$  and  $t$  resembles three well known update rules, which we will summarize in three Lemmas.

<sup>3</sup> Well known examples are the ADAM-optimizer, RMSPROP, or ADAGRAD. In general, it depends on the structure of the loss  $L_{\mathcal{D}}$ , if taking the diagonal is a meaningful way of approximation.

<sup>4</sup> It is not identical to assuming independent models on elements of  $x$ , since then  $\Delta\tau$  would split into a different  $\Delta\tau_i = |w_{t+1}^i - w_t^i|$  for each dimension  $i = 1, \dots, N$ .

**Lemma 1** *The filtered state  $m_t$  of Eq. 123 is identical to the enumerator of the not bias corrected ADAM-update for  $\beta = (1 - g) \in (0, 1)$ , same initialization  $m_0$ , and the same sequence of observations  $y_1, \dots, y_t$ .*

**Proof** Can be directly seen from Eq. 67c where the enumerator of ADAM is  $m_t = \beta m_{t-1} + (1 - \beta) \nabla L_S(w_t)$ . ■

**Lemma 2** *The filtered state  $m_t$  of Eq. 123 is identical, up to a global multiplicative constant, to the negative velocity  $v_t$  of SGD+momentum (Eq. 43) with constant learning rate, constant momentum factor  $\gamma$ , initialization  $m_0 = -v_0$ , and the same sequence of observations  $y_1, \dots, y_t$ . Additionally, the global constant is given by  $g$ , meaning that both algorithms yield the identical sequence of locations  $w_t$  if  $v_0 = -m_0$ ,  $\gamma = (1 - g)$ , and learning rates  $\alpha_{\text{MOM}} = \alpha_{\text{KF}} g$ .*

**Proof** Let  $v_t$  be the velocity of SGD+MOMENTUM as defined in Eq. 43, and let  $\alpha_{\text{MOM}}$  and  $\alpha_{\text{KF}}$  be the constant learning rates of momentum SGD and the filter respectively. Then,  $v_t = \gamma v_{t-1} - \alpha_{\text{MOM}} \nabla L_S(w_t) = -\alpha_{\text{MOM}} \sum_{i=1}^t \gamma^{t-i} \nabla L_S(w_i)$ , and analogously for the filter with constant gain  $m_t = g \sum_{i=1}^t (1 - g)^{t-i} \nabla L_S(w_i)$ ; the formulas are equivalent for  $g = 1 - \gamma$  up to a constant global scale of size  $\alpha_{\text{MOM}}/g$ , which can be absorbed into the learning rate of Eq. 124. ■

**Lemma 3** *The filtered state  $m_t$  of Eq. 123 for noise-free observed gradients ( $R_t = 0$  for all  $t$ ) recovers gradient descent on  $L_{\mathcal{D}}$  for the same sequence of observations  $y_1, \dots, y_t$ , regardless of the initialization  $m_0$ . Additionally, for same initialization  $m_0 = \nabla L_{\mathcal{D}}$  and same learning rates  $\alpha_{\text{GD}} = \alpha_{\text{KF}}$ , both algorithms yield the identical sequence of locations  $w_t$ .*

**Proof** For noise-free observed gradients ( $R_t = 0$  for all  $t$ ), the gain  $g$  is always one since  $g^{ii} = P^{ii}/(P^{ii}+R) = P^{ii}/(P^{ii}) = 1$ , and thus  $m_t = \nabla L_{\mathcal{D}}(w_t)$ . ■

All lemmata, give insight on possible model assumptions of the mentioned classic algorithm. Nevertheless, although the algebraic similarities are intriguing, the filtering equations purely encode smoothing due to noisy observations (mini-batching). The momentum parameter  $\gamma$  or the smoothing constant  $\beta$  of ADAM which yield the best overall performance of their corresponding iterative procedure might differ from the corresponding optimal gain of the filter, since also geometric smoothing effects can be captured by them (smoothing of zig-zagging in ravines, bending around curves, et cetera), which the filtering model simply does not encode. The filtering formulation is thus a way of disentangling noise effects due to mini-batching and additional desirable smoothing due to the geometry of the loss that might even be beneficial when  $\nabla L_{\mathcal{D}}$  is known precisely. In the experimental Section 8.3, we

will see hints, which support the hypothesis that, in neural network application, most, if not all, of the smoothing of SGD+MOMENTUM can be explained by noise due to mini-batching (i. e.,  $R$ ).

## 8.2.2 Hyper-parameter Adaptation

An efficient estimator for the noise variances  $R_{kk}$  of the measurements was introduced in Chapter 5, Eq. 90. Here, analogously, we can set  $R_t^{kk} = \hat{\Sigma}^{kk}(w_t)/|S|$ .

One way of adapting the intensity matrix  $q$ , is by maximum marginal likelihood estimation (§ 1.4.1). In general, the (logarithmic) marginal likelihood for parameter  $q$ , for the current noisy datapoint  $y_{t+1}$  with noise covariance  $R_{t+1}$  is:

$$\begin{aligned} p(y_{t+1}) &= \int p(y_{t+1}|x_{t+1}, y_1, \dots, y_t) p(x_{t+1}|y_1, \dots, y_t) dx_{t+1} \\ &= \mathcal{N}(y_{t+1}; m_{t+1-}, P_{t+1-} + R_{t+1}) \\ \log p(y_{t+1}) &\propto -\frac{1}{2} \log |P_{t+1-} + R_{t+1}| - \frac{1}{2} (y_{t+1} - m_{t+1-})^\top (P_{t+1-} + R_{t+1})^{-1} (y_{t+1} - m_{t+1-}). \end{aligned} \quad (125)$$

The following derivation uses diagonal forms for all matrices  $R$ ,  $q$ ,  $P$ , et cetera as in Section 8.2 (the full derivation can be found in Appendix B.1). The gradient of  $\log p(y_{t+1})$  with respect to  $q_{kk}$  is, dropping the index  $t$  to declutter:

$$\frac{\partial}{\partial q_{kk}} \log p(y_{t+1}) = -\frac{1}{2} \frac{\partial}{\partial q_{kk}} \log |P^- + R| - \frac{\partial}{\partial q_{kk}} \frac{1}{2} \Delta^\top (P^- + R)^{-1} \Delta. \quad (126)$$

where  $\Delta_{t+1} := y_{t+1} - m_{t+1-} \in \mathbb{R}^N$  is the *residual* (not to be confused with the scalar path segment  $\Delta\tau$ ), and  $P^- = P_{t+1-} = P_t + q\Delta\tau$ . With some algebra, Eq. 126 simplifies to:

$$\frac{\partial}{\partial q_{kk}} \log p(y_{t+1}) = \frac{\Delta\tau}{2} \left( \Delta_k G_{kk}^{-1} \right)^2 - \frac{\Delta\tau}{2} G_{kk}^{-1}, \quad (127)$$

such that the root of the gradient with respect to  $q_{kk}$  is at

$$q_{kk}^* = \frac{1}{\Delta\tau} (\Delta_k \Delta_k - P_{kk} - R_{kk}). \quad (128)$$

Ideally, one would like to incorporate all, or  $M < t$  past observations  $\{y_{t-i+1}\}_{i=1}^M$  into the marginal likelihood estimator. Practically, though, this is not ideal, since computing  $p(y_{t-M+1}, \dots, y_t)$  would require to keep a significant amount of gradients in storage. A practical workaround to this is to compute  $q_{kk}^*$  as in Eq. 128 in every iteration, and then smooth  $q_{kk}^*$  with a slowly decaying exponential running average  $\bar{q}_{t+1} = \gamma \bar{q}_{t+1} + (1 - \gamma) q_{t+1}^*$ .<sup>5</sup> The value of  $\gamma$  should be at least as large as conservative smoothing choices of classic decay factors (something around  $\gamma \approx 0.95$  or larger), since it in fact smoothes the parameters of the *distribution*, and not the quantities of interest (such

<sup>5</sup> Since the elements of the diagonal matrix  $q^*$  can also become negative, but the theoretical  $q$  is positive definite by definition, a practical algorithm will need to use e.g., a ‘clipped’ version, i.e.,  $\text{diag}[q_t] = \max(0, \text{diag}[\bar{q}_t])$  for computing  $P^-$ . A vanishing or negative  $q^*$  just means that the noise  $R_t$  and the state uncertainty  $P_t$  fully explain the discrepancy  $\Delta_{t+1}$  between the predictive estimator and the current stochastic observation.

Additionally, one might note here that we got rid of a smoothing parameter ( $\gamma$  of SGD+MOMENTUM) just to introduce yet a new one to smooth  $\bar{q}$ . The difference is that smoothing occurs one level higher in the parameter hierarchy, and can thus be hoped for to be much less sensitive to choices of that smoothing factor. After all, the gains  $g_t$  still can alter per element and per iteration, depending on a distribution which is parametrized among others by a slower changing  $\bar{q}$ .

as the gradients). If an online-model, such as the filter, would be based on hyper-parameters which change faster with the data than the quantities they estimate, then the inference is as random as the data itself. For the filter, this roughly means that  $1 - \gamma$  should be larger than an average gain.

Instead of diagonal forms, an even coarser simplification are *scalar* forms  $P_t = pI$ ,  $R_t = rI$ , and  $q_t = uI$ , with  $p, r, u > 0$  for all relevant matrices. The maximum marginal likelihood estimator then measures the mean-discrepancy between observation and predictive state which can not be explained by  $r$  and  $p$  already:

$$u^* = \frac{1}{\Delta\tau} \left( \frac{1}{N} \sum_{i=1}^N \Delta_i^2 - p - r \right). \quad (129)$$

Taking the average over parameters increases the amount of numbers available for statistics, and the resulting estimator  $\bar{u}^*$  is thus arguably more robust, but it also reduces the flexibility of the model. The corresponding update  $m_t = (1 - g_t)m_{t-1} + g_t y_t$ , however, much more resemble that of `SGD+MOMENTUM` since then the gain  $g_t \in (0, 1)$  is also scalar.<sup>6</sup>

### 8.3 Experiments

The following experiments provide a proof-of-concept for filtering on gradients. The viability on a broader range of problems, possible benefits or shortcomings over existing (momentum) methods, as well as extensions and fine tuning, will need further testing in the future on a larger number of models  $f_w$ ,  $\ell_f$  and datasets  $\mathcal{D}$ . For now, Section 8.3.1 provides an in-model toy-example of a synthetic 50-dimensional multi-output regression problem mimicking gradients  $\nabla L_{\mathcal{D}}(\tau)$ , to test approximations and the maximum likelihood estimator for  $q$  where ground truth is available. Section 8.3.2 tests the diagonal filter (§ 8.2) with learned  $\bar{q}$  (§ 8.2.2) on an illustrative out-of-model function with fast varying  $q$ . And, finally, Section 8.3.3 applies the filter to a real world problem (an MLP on MNIST), and compares gain heuristics to the smoothing factor  $\gamma$  of `SGD+MOMENTUM`. The last experiments cautiously supports the claim that most, or all of the smoothing in momentum-`SGD` benefits noise reduction only.

#### 8.3.1 In-Model Toy Example

First, we start with an in-model toy example, where the artificially constructed 50-dimensional gradients of the objective  $L_{\mathcal{D}}(\tau)$  are a draw from the generative model, i. e., a multi-output Wiener process with positive definite dense intensity matrix  $q \in \mathbb{R}^{N \times N}$ . Since the

<sup>6</sup> Statistics can also be collected per subgroup of parameters, e. g., per weights in one layer or per biases and weights. Pseudo-codes can be found in Algorithms 5 and 6.

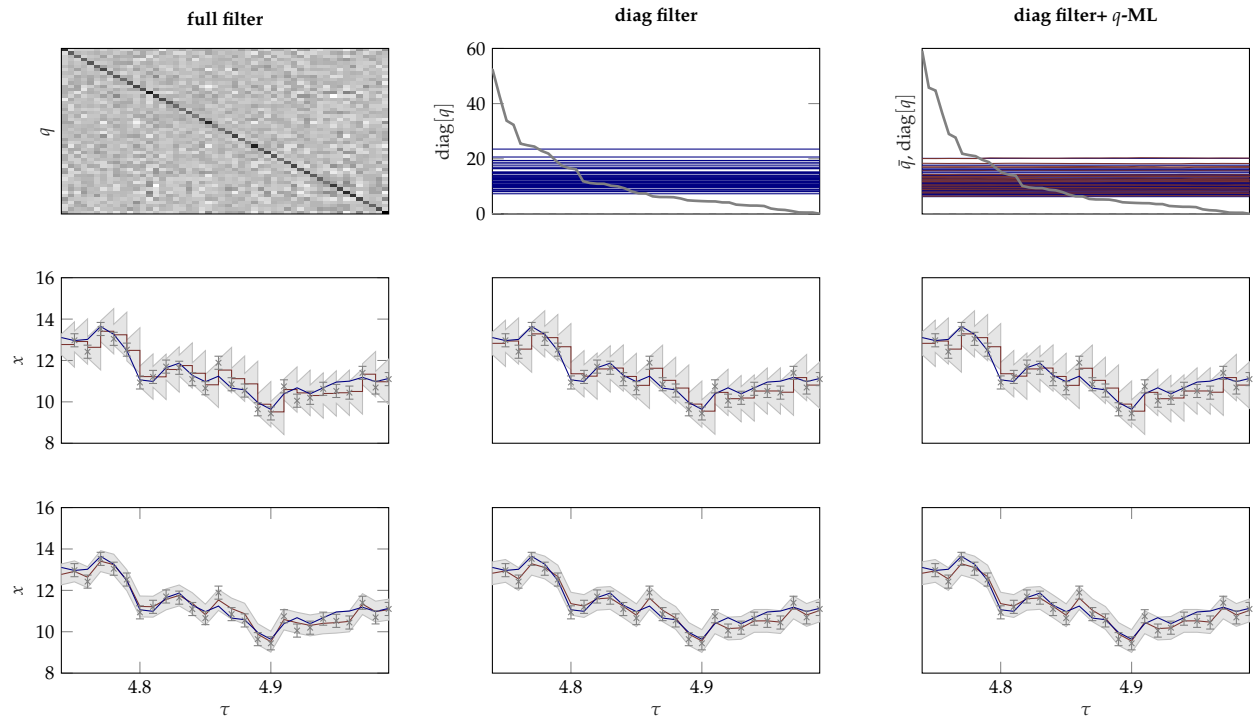


Figure 51: In-model toy example. Columns from left to right: full filter, diagonal filter, diagonal filter with max marginal likelihood estimator for  $\text{diag}[q]$ . Top row: true and inferred intensity  $q$ . Middle row: predictive states. Bottom row: updated states. Means  $\pm 2$  std (—  $\pm 2$   $\square$ ), observations (— $\times$ —) and ground truth (—).

ground truth is known, this controlled setup allows to test the diagonal approximations of Section 8.2 as well as the maximum likelihood estimator for  $\text{diag}[q]$  of Section 8.2.2. If the filter (hereafter generically called KFGRAD, for ‘Kalman filter on gradients’) does perform poorly here, we can not expect it to perform much better on out-of-model functions.

The intensity matrix  $q$  is constructed by defining an eigen-spectrum with exponential structure. Then, similar to Section 6.3.2, we draw a random rotation  $\zeta \in \mathbb{R}^{N \times N}$  uniformly from all possible 50-dimensional rotations, and define  $q := \zeta \Gamma \zeta^T$ , where  $\Gamma \in \mathbb{R}^{N \times N}$  is a diagonal matrix that contains the eigenvalues on its diagonal. The inputs space  $\tau$  is discretized in equal portions  $\Delta\tau$ , such that  $x_{t+1} = x_t + \zeta$  with  $\zeta \sim \mathcal{N}(0, \Delta\tau q)$ . Observations  $y_t$  are constructed by adding isotropic Gaussian noise on the ground truth function:  $y_t = x_t + v$  with  $v \sim \mathcal{N}(0, R)$ , and  $R = \sigma_R^2 I$ .

We conduct three experiments with filters that possess decreasing knowledge or expressiveness about  $\nabla L_{\mathcal{D}}(\tau)$ : i) A full filter as in Eqs. 121 and 122 (inferring dense covariance matrices  $P_t$  and  $P_{t+1}$ ) which has access to the true *dense* intensity matrix  $q$  as well as noise covariance  $R$ . This is an *in-model* problem, meaning that the filter infers a function which is an instance of its own exact generative model, thus providing a ground truth comparison to the following approximate filters. ii) A diagonal filter as in Eq. 123 which has access to

the true *diagonal* of  $q$  (as well as  $R$ ), but not its off-diagonal elements.

iii) A diagonal filter as in ii) which learns the diagonal of  $q$  by maximum marginal likelihood estimation as in Section 8.2.2, Eq. 128. This filter, too, has access to the true measurement noise  $R$ . The latter two filters are to separate effects of contributions from the diagonal approximation and the estimation of  $q$ .

Figure 51 illustrates results of all three runs: columns from left to right are i), ii) and iii) respectively. The middle and bottom row show one (the first by random choice) of the  $N = 50$  dimensions of the inferred state  $x$  for predictive and updated marginal probabilities versus the input  $\tau$  respectively (true function values  $\nabla L_{\mathcal{D}}^1$  (—), observations  $y_t^1 \pm \sigma_R$  (---), means  $m_{t-}^1$  and  $m_t^1$  (—) with  $\pm 2$  standard deviations  $(P_{t-}^{11})^{1/2}$  and  $(P_t^{11})^{1/2}$  (□□)). The top left plot shows the true dense intensity matrix  $q$  as used by i) in arbitrary gray-scale; the top middle plot the (constant) diagonal elements of  $q$  (—) versus  $\tau$  as used by ii), as well as the sorted eigen-spectrum of  $q$  versus an arbitrary abscissa for reference (—). The top right plot additionally shows the learned diagonal of  $q$  ( $\bar{q}$ , —) versus  $\tau$  as used by iii). All three filters look very similar, and also the learned  $\bar{q}$  of iii) seem to match the true  $\text{diag}[q]$  quite well. This is not too surprising, since the model indeed gets to see the whole state  $x$ , although noise corrupted, i. e., the likelihood covers the full state space at every discrete time step. Thus, if  $\bar{q}$  is scaled right, the model can not be arbitrarily wrong.

### 8.3.2 Out-of-Model Toy Example

Usually, the gradients  $\nabla L_{\mathcal{D}}$  of an objective function are not draws from a multi-output Wiener process and the filtering model, with a globally constant  $q$ , might only be approximately correct, or locally around  $w_t$ . In a similar sense, classic optimizers often use simple local approximate models or a Lipschitz-constant that is allowed to change if one moves further away from a point than a local neighborhood. Corresponding estimators thus may evolve over time, which is implicitly encoded by smoothing factors like  $\gamma$ . We test this concept here on a simple two-dimensional function: the Rosenbrock polynomial. The ground truth of the state  $x$ , as well as the step intervals  $\Delta\tau$  are obtained by running gradient descent on Rosenbrock. Then, each obtained gradient  $x_t$  is corrupted by additive isotropic Gaussian noise  $R = \sigma_R^2 I$ ; after that the diagonal filter with on-line  $q$ -adaptation (number (iii) of the previous section) is trained on it.

Figure 52 depicts results. The top left plot shows contour lines of Rosenbrock as well as the optimizer's path which was used for the ground truth collection (start location and minimum as crosses). The path-segment in red corresponds to the interval shown in all the



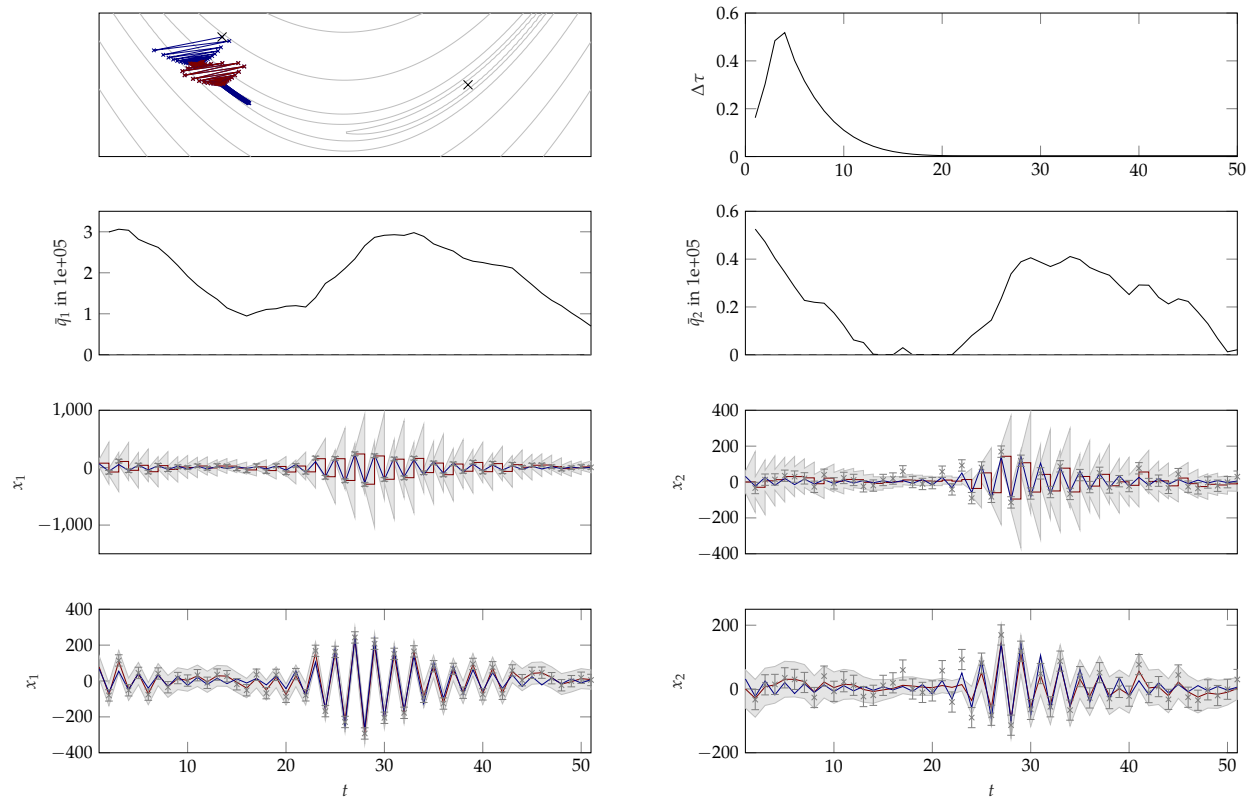


Figure 52: Out-of-model toy example. Left and right column are first and second dimension of state  $x$ . Rows 3-4: Predictive and updates state (colors as in Figure 51). Row 2: estimated intensity  $\bar{q}$ . Row 1: non-uniform path segments  $\Delta\tau$  (right), and ground truth with contours of Rosenbrock (left).

other plots; its starts shortly before the fixed learning rate is set to a larger, also fixed value to encourage changes in  $q$ . The top right plot shows corresponding lengths  $\Delta\tau$  of path segments for reference. All plots of rows 2-4 are plotted versus # iteration  $t$  as opposed to  $\tau$  for better illustration since  $\Delta\tau$  spans a few magnitudes. Rows 3 and 4 again show the predictive and updated marginal probabilities of  $x$ : Left column for first dimension, right column for second dimension, colors same as in Figure 51. Row 2 additionally plots the learned intensity estimators  $\bar{q}_1$  and  $\bar{q}_2$ . The qualitative behavior (drop and rise) of  $\bar{q}$  well matches the changes in observed gradients per path segment: In areas where the change in gradient is nearly fully explained by  $R$  (low signal-to-noise ratio around  $t \approx 10$  to  $20$ ),  $\bar{q}$  drops, and then rises again when gradient changes become larger, as they can not be explained by  $R$  only anymore ( $t \approx 20$  to  $40$ ). Also, the estimators  $m_t$  for the gradient seem to be closer to the corresponding true values than the noisy observations are.

#### *A First Iterative Test on Rosenbrock*

So far, we only tested KFGRAD as a regression type algorithm with previously generated (gradient) data. This helped to compare to the same ground truth as SGD and also factored out the feedback an opti-



```

1: function KFGRAD_DIAG( $L, w_0, \alpha, \gamma \approx 0.95$ )
2:    $w_t \leftarrow w_0$  // initial guess for weights
3:    $[y_t, \hat{\Sigma}_t] \leftarrow L(w_t)$  // initial evaluation
4:    $m_t \leftarrow (1 - \gamma) \cdot y_t$ 
5:    $R_t \leftarrow \hat{\Sigma}_t$ 
6:    $P_t \leftarrow R_t$ 
7:    $\bar{q}_t \leftarrow 10^2 \cdot R_t$ 
8:   while budget not used do
9:      $w_t \leftarrow w_t - \alpha m_t$  // update best guess
10:     $[y_t, \hat{\Sigma}_t] \leftarrow L(w_t)$  // evaluate objective
11:
12:     $R_t \leftarrow \hat{\Sigma}_t$  or  $R_t \leftarrow \gamma R_t + (1 - \gamma) \hat{\Sigma}_t$ 
13:     $\Delta\tau \leftarrow \|\alpha m_t\|$ 
14:     $P_t^- \leftarrow P_t + \Delta\tau \max(0, \bar{q}_t)$ 
15:     $g_t \leftarrow P_t^- \oslash (P_t^- + R_t)$ 
16:     $\bar{q}_t \leftarrow \gamma \bar{q}_t + (1 - \gamma) \Delta\tau^{-1} [(y_t - m_t)^{\odot 2} - P_t - R_t]$ 
17:     $m_t \leftarrow (1 - g_t) m_t + g_t y_t$ 
18:     $P_t \leftarrow (1 - g_t) P_t^-$ 
19:  end while
20:  return  $w_t$ 
21: end function

```

Algorithm 5: Sketch of the KFGRAD\_DIAG algorithm. The intensity  $\bar{q}$  is initialized larger than the noise  $R$  to force an SGD-type start (until  $\bar{q}$  has burned in). It is important to compute  $P_t^-$  before updating  $\bar{q}$ , and updating  $P_t$  after computing  $\bar{q}$ .

mizer gets when it is choosing its own data through  $p_t$  and  $\alpha_t$ . Thus next, we apply KFGRAD and SGD to the same function as above, noisy Rosenbrock, but this time as an iterative scheme (Algorithm 5 for KFGRAD\_DIAG). Both optimizers start from the same initialization  $w_0$  and get the same best working constant learning rate  $\alpha$  for each run. We perform 100 restarts each for three different noise levels ( $\sigma_R^2 = 10, 100, 1000$ ), and report the evolution of the mean and standard deviation of the *true* logarithmic loss.<sup>7</sup> Figure 53 shows results. Rows from top to bottom correspond to increasing noise levels  $\sigma_R^2 = 10, 100, 1000$  respectively; means of logarithmic losses (—/—)  $\pm 1$  standard deviation (■/■) for KFGRAD and SGD respectively. It is evident that KFGRAD outperforms SGD both in lower mean loss as well as lower loss variances, but more so for smaller noise levels. Also, at the start of the optimization process ( $t$  up to  $\approx 50 - 100$ ) KFGRAD and SGD perform very similarly since the noise level is low and the Kalman gains are close to one (In essence for  $R \rightarrow 0$ , KFGRAD and SGD are identical to gradient descent, but they differ more and more for larger  $R$ ). With this motivating toy example we try KFGRAD on a real world problem next.

<sup>7</sup> Since gradients become smaller closer to the minimum, this decreases the signal-to-noise-ratio towards the end of the optimization process.

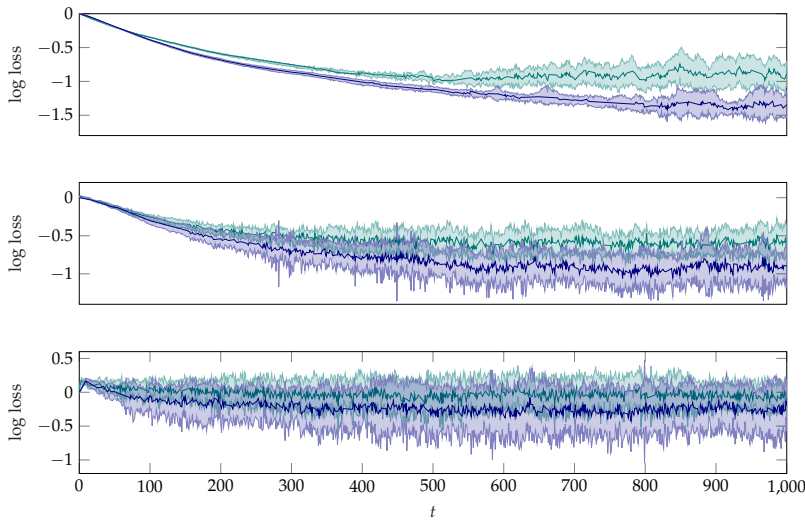


Figure 53: KFGRAD and SGD on Rosenbrock. Rows from top to bottom  $\sigma_R^2 = 10, 100, 1000$ . Means of true log losses (— / —)  $\pm 1$  std ( / /) for KFGRAD and SGD respectively.

```

1: function KFGRAD_SCALAR( $L, w_0, \alpha, \gamma \approx 0.95$ )
2:    $w_t \leftarrow w_0$  // initial guess for weights
3:    $[y_t, \hat{\Sigma}_t] \leftarrow L(w_t)$  // initial evaluation
4:    $m_t \leftarrow (1 - \gamma) \cdot y_t$ 
5:    $r_t \leftarrow \text{mean}[\hat{\Sigma}_t]$  // scalar
6:    $p_t \leftarrow r_t$  // scalar
7:    $\bar{u}_t \leftarrow 10^2 \cdot r_t$  // scalar
8:   while budget not used do
9:      $w_t \leftarrow w_t - \alpha m_t$  // update best guess
10:     $[y_t, \hat{\Sigma}_t] \leftarrow L(w_t)$  // evaluate objective
11:     $r_t \leftarrow \gamma r_t + (1 - \gamma) \cdot \text{mean}[\hat{\Sigma}_t]$  // scalar
12:
13:     $\Delta\tau \leftarrow \|\alpha m_t\|$  // scalar
14:     $p_t^- \leftarrow p_t + \Delta\tau \max(0, \bar{u})$  // scalar
15:     $g_t \leftarrow p_t^- / (p_t^- + r_t)$  // scalar
16:     $u^* \leftarrow \Delta\tau^{-1} (\text{mean}[(y_t - m_t)^{\odot 2}] - p_t - r_t)$  // scalar
17:     $\bar{u}_t \leftarrow \gamma \bar{u}_t + (1 - \gamma) u^*$  // scalar
18:     $m_t \leftarrow (1 - g_t) m_t + g_t y_t$ 
19:     $p_t \leftarrow (1 - g_t) p_t^-$  // scalar
20:  end while
21:  return  $w_t$ 
22: end function

```

Algorithm 6: Sketch of the KFGRAD\_SCALAR algorithm. The pseudo-code is similar to Algorithm 5 but this time for scalar measurement noise  $R$  and diffusion  $q$  (consequently the variances  $P_t, P_t^-$ , and the gain  $g_t$  are also scalar). Thus the algorithm has very little memory requirement; the same as SGD+MOMENTUM.

### 8.3.3 Multi-Layer Perceptron and Comparison to Momentum

As argued in Section 8.2.1, the most obvious relation of  $\text{KFGRAD}$  occurs to  $\text{SGD+MOMENTUM}$ . We can thus compare the gains learned by  $\text{KFGRAD}$  to the smoothing factor  $(1 - \gamma_{\text{MOM}}) = 0.1$  of  $\text{SGD+MOMENTUM}$ . Let  $\alpha_{\text{MOM}}$  denote the constant learning rate for  $\text{SGD+MOMENTUM}$ , then, from Lemma 2, we know that we can recover the identical  $\text{KFGRAD}$ -update by setting the filtering learning rate to  $\alpha_{\text{KF}} = \alpha_{\text{MOM}} / (1 - \gamma_{\text{MOM}})$  and the gain to  $g = 1 - \gamma_{\text{MOM}}$ . For the same network (N-II) as in Chapter 7.5 (fully connected MLP with 5 layers, 3 hidden) on MNIST and mini-batch size  $|\mathcal{B}| = 200$ , we ran four experiments: i)  $\text{SGD+MOMENTUM}$  with  $\gamma_{\text{MOM}}$ , ii)  $\text{KFGRAD\_DIAG}$  as in Algorithm 5, iii)  $\text{KFGRAD\_SCALAR}$  as in Algorithm 6, and iv) vanilla  $\text{SGD}$  for comparison (can be seen as corner case for  $1 - \gamma = g = 1$ ).  $\text{KFGRAD\_DIAG}$  and  $\text{KFGRAD\_SCALAR}$  have different strengths; for instance  $\text{KFGRAD\_SCALAR}$  can collect better statistics for the hyperparameters  $R_t$  and  $q_t$  since it can average over all weights per iteration. Also it is very memory efficient which is relevant in very high dimensional problems: The only vector is the search direction  $m_t$ , all other quantities are scalar.  $\text{KFGRAD\_DIAG}$ , on the other hand, is more expressive and potentially more powerful since it can adjust a different measurement noise, diffusion, and thus gain in each dimension. A more direct comparison can hence be done between  $\text{KFGRAD\_SCALAR}$  and  $\text{SGD+MOMENTUM}$  since both model a scalar gain/smoothing, only.

In the following analysis, we will be primarily interested in the question if we can recover similar performances than momentum  $\text{SGD}$  with these first prototypes of  $\text{KFGRAD}$ . And also, to what extent the smoothing done by  $\text{SGD+MOMENTUM}$  can be explained by noise corrupted gradients only, in contrast to geometrical smoothing, since the latter is often called out as the explanation for the success of  $\text{SGD+MOMENTUM}$ .

For each experiment, we search for the best learning rate on a logarithmic grid of  $\alpha = 10^{-8}, 10^{-7}, \dots, 10^0$ , and then fine tune the search in a promising region in steps of  $3, 5, 7 \cdot 10^{-k}$  for some  $k$ . The best performing learning rate for  $\text{SGD+MOMENTUM}$  was  $\alpha_{\text{MOM}}^* = 7 \cdot 10^{-3}$ . Indeed, and perhaps surprisingly, the best performing learning rate for  $\text{KFGRAD}$  (scalar as well as diagonal version) was ten times the one of  $\text{SGD+MOMENTUM}$   $\alpha_{\text{KF}}^* = \alpha_{\text{MOM}}^* / (1 - \gamma_{\text{MOM}}) = 7 \cdot 10^{-2}$ .

Figure 54 shows the logarithmic train and test error traces for all four setups (colors in caption), each for the best performing learning rate and five different random seeds.  $\text{KFGRAD\_DIAG}$ ,  $\text{KFGRAD\_SCALAR}$ , and  $\text{SGD+MOMENTUM}$  perform very similarly and all three better than vanilla  $\text{SGD}$ , especially in train error decay.

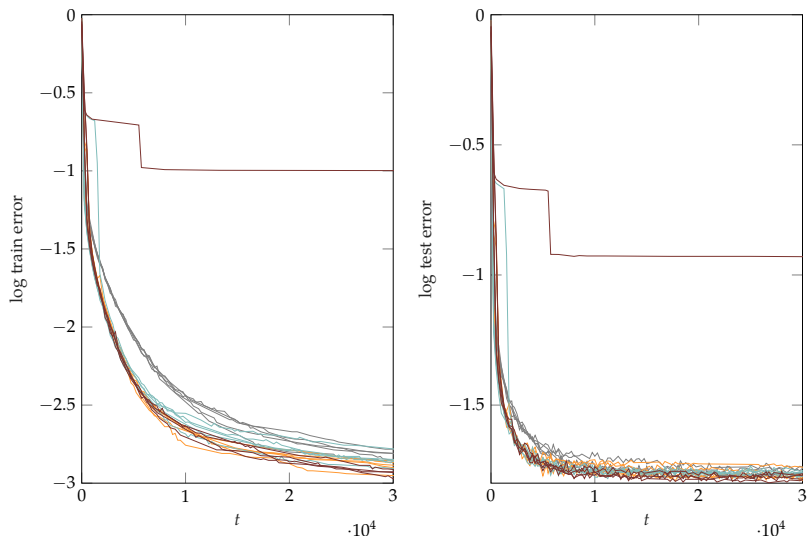


Figure 54: Multi-layer perceptron on MNIST. Left and right columns show log train and test error respectively versus number of mini-batch evaluations. KFGRAD (diagonal/scalar —/—), and SGD+MOMENTUM (—); SGD for comparison (—). Each setup is shown for 5 random seeds (curves of same color) which determine initial weights and mini-batch sub-sampling.

### Diffusion and Gains

It is not straightforward to compare the adaptive gains of the Kalman filter to a single ‘gain’ of SGD+MOMENTUM i. e., the smoothing constant  $1 - \gamma_{\text{MOM}} = 0.1$ , since we compare many numbers to one. Thus we have a look at the *distribution* of gains over the course of all iterations. The top and bottom plot of Figure 55 shows a histogram over all gains  $g_t$  for KFGRAD\_DIAG and KFGRAD\_SCALAR respectively (top plot for a random weight). The mode, median, and mean of the distribution are shown as vertical lines (—/---/----). The values are 0.05/0.13/0.25 for the scalar model (bottom), and 0.05/0.08/0.20 for a single weight of the diagonal model (top) which is very close/similar to the smoothing of SGD+MOMENTUM. Additionally, the shape of the distribution is telling: There is a prominent peak around the mode at small gains, and most gains arguably occur in between  $(0, 0.2)$ , but there is also a heavy tail of larger gains. It is not clear if this tail is a true gradient-signal, i. e., gradients with a good signal-to-noise ratio, or an artifact of the smoothing or averaging of variance estimates  $r_t$  or  $R_t$ : Empirically the noise is somewhat coupled to the gradient magnitude, thus, if the noise estimate is smoothed or even averaged, the gains for larger stochastic gradients or outliers would indeed be biased towards larger gains, since  $R_t$  is estimated too small. This might but does not need to explain the heavy tails.

So, although this analysis can not be completely conclusive, it is a strong indicator that in shallow or deep learning problems, a lot or all of the smoothing done in SGD+MOMENTUM is beneficial for noise reduction on stochastic gradients, and not, as often advocated,

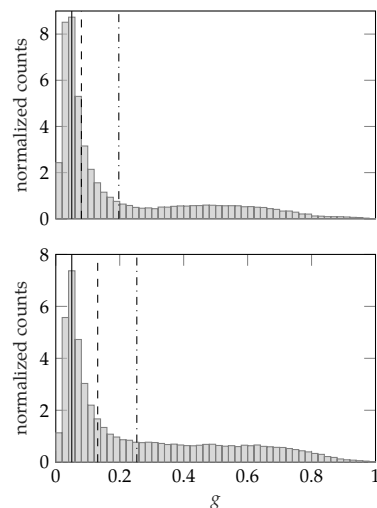


Figure 55: Distribution of Kalman gains of KFGRAD collected over iterations. Bottom: scalar version with single gain per net. Top: diagonal version, shown are gains of one randomly chosen weight. In both plots, the mean (----), mode (—), and median (---) are marked with vertical lines.

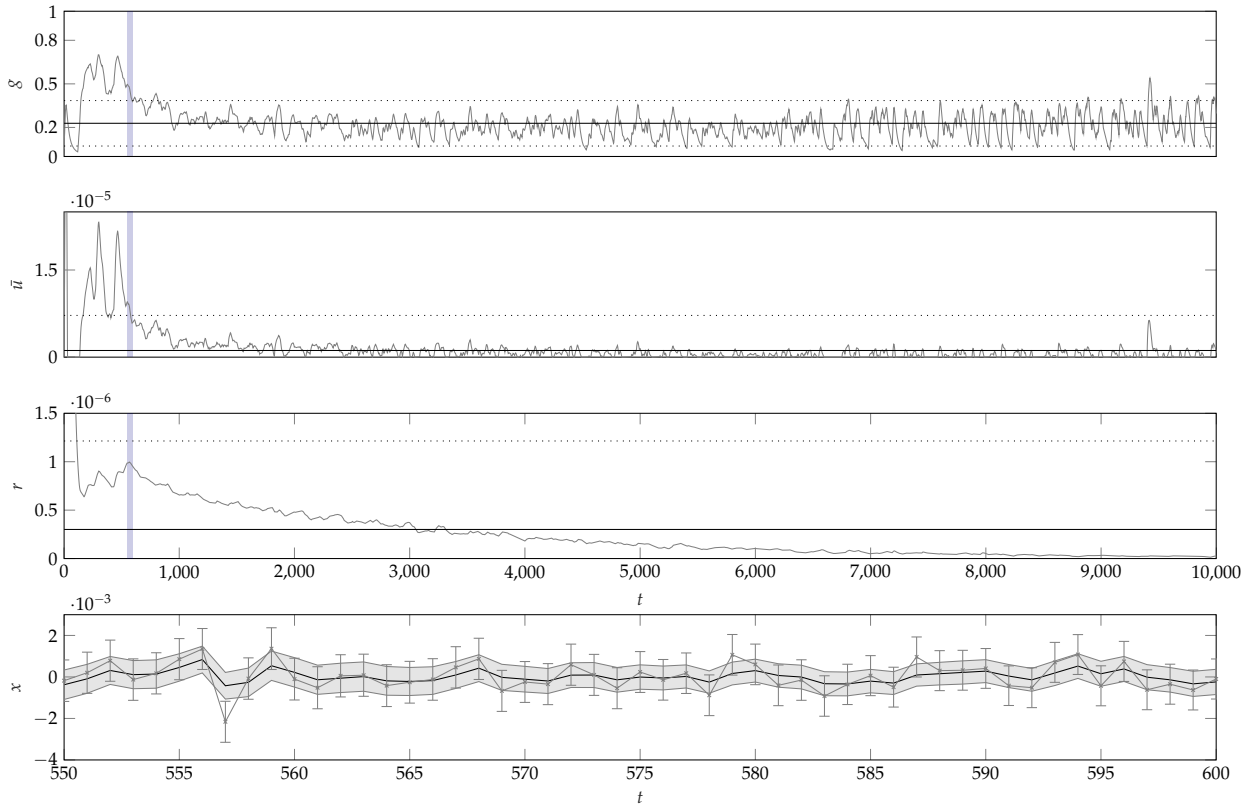


Figure 56: Diagnostics of KFGRAD\_SCALAR, from one run of Figure 54. Rows 1-3: Traces of learned scalar gains, diffusions, and gradient noise respectively, versus number of iterations (—). Mean $\pm$ 1std as horizontal lines (— $\pm$ .....). Row 4: Posterior marginals of  $x_t^i$  for a single random weight  $i$ , mean  $m_t^i$  (—),  $m_t^i \pm \sqrt{P_t^i}$  (□), observations  $y_t^i \pm \sqrt{r_t^i}$  (---). The traces in the top three plots are smoothed for plotting purposes but the mean and std are computed with the raw data. The blue areas (□) indicate the plotting interval for the fourth row, chosen such that it shows an interesting region in the optimization process.

due to geometrical effects. This might of course be very different in other applications, e. g., with lower dimensionality, or where stochastic noise due to mini-batching is not present or not a major concern. As mentioned above, another fact which supports this claim, is that the best performing learning rate of KFGRAD corresponds to  $\alpha_{\text{MOM}}^*/(1-\gamma_{\text{MOM}})$ , which indicates that the average norm of the search direction is similar for all three methods. This can only occur if a similar smoothing is done. Thus, assuming the learned gains are roughly sensible for the given estimated gradient variances  $\hat{\Sigma}$ , nearly all, or all of the smoothing contribution done by SGD+MOMENTUM can be attributed to benefit gradient noise reduction.

Figures 56 and 57, rows 1-3, show additional diagnostics of gain, diffusion, as well as gradient-noise traces for KFGRAD\_SCALAR and KFGRAD\_DIAG, for one of the runs of Figure 54. It shows that all three quantities are initially larger and then decay during the optimization process, though especially the gain seems to settle for some distribution when the optimizer converges. The last row shows the marginal filtering distributions of  $x_t$  conditioned on all  $y_t$  up to that point in time, of a randomly chosen weight. The noisy gradient evaluations  $y_t$  are shown with error bars. The mean estimator  $m_t$  (—) seems to be smaller in magnitude on average than gradient ob-

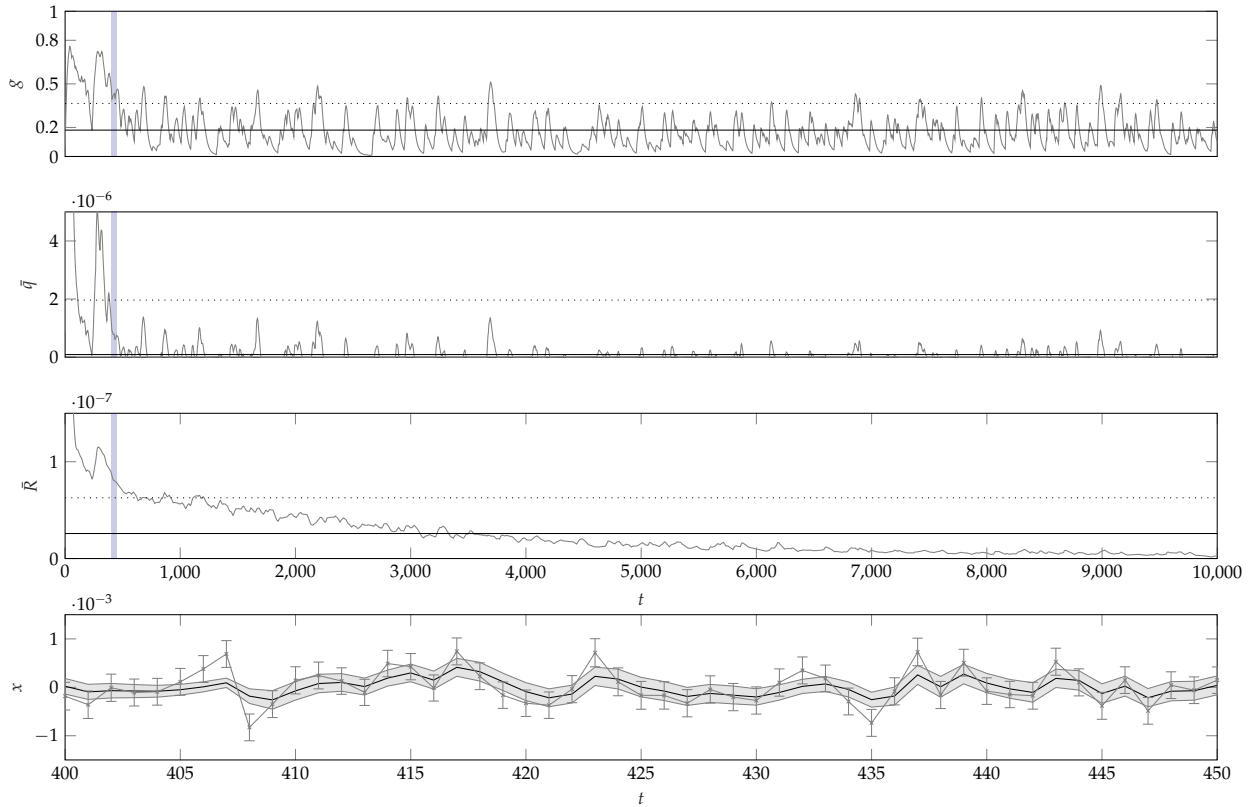


Figure 57: Diagnostics of KFGRAD\_DIAG, from one run of Figure 54. Plots are colored as in Figure 56, but this time the traces are also only shown for a single random weights (identical to the one in the last row).

servations  $y_t$  (—), and also less ‘spiky’ which might explain that it is possible to choose a larger learning rate for KFGRAD in comparison to vanilla SGD. Also, just by eyesight, the posterior standard deviations  $\sqrt{P_t}$  (■) seem well calibrated. A concern might be that the learned diffusion  $\bar{q}$  is quite spiky, too (partly translates to the gain), which is an artifact of ad-hoc smoothing of one-sample maximum likelihood estimators (§ 8.2.2). This might need improvement in the future, e. g., by hierarchically modeling  $q$ , and thus softening the impact of unexpected observations.

## 8.4 Conclusion and Outlook

The following bullet-points summarize the main results of this Chapter:

- We presented a novel filtering framework for stochastic gradients, based on a continuous Gauss-Markov-model on the optimization path, and local Gaussian observations. We derived discrete Kalman filter prediction and update equations for the special case of Brownian motion.

- We showed correspondences of the mean estimator to classic *first-order* methods such as `SGD+MOMENTUM`, or the enumerator of the `ADAM`-optimizer. While exponential smoothing factors are ad-hoc in these methods, they arise naturally in the filtering equations, and thus provides an indirect justification for using them. Additionally, these smoothing factors can now, wholly or partially, be interpreted as Kalman-gains and provide a measure of how much stochastic gradients can be trusted.
- We presented a prototype of a probabilistic first-order optimizer (`KFGRAD`) that uses gradient-mean-estimates of the filtering distributions as search directions, and can learn the smoothing constants. It is expressive and has the potential to be more powerful than hand-tuned smoothing constants since one gain can be learned and locally adapted per dimension. This is promising towards the goal to control, analyze and automate optimization further.
- Additionally, the new framework helps to disentangle noise- and geometric contributions of smoothing constants. Experiments showed evidence that classic smoothing in high-dimensional mini-batch settings indeed only, or mostly, benefits stochastic noise reduction in contrast to geometrical smoothing.

Future research directions:

- Performance improvement of `KFGRAD` might go towards a more robust estimation of the diffusion  $q$ , which is currently done by averaging over single-sample maximum likelihood estimators. This might be approached by hierarchically modeling  $q$  such that outliers have less impact.
- The posterior variances  $P_t$  of the marginal filtering distributions can be used for methods that currently use a sample-variance estimator (a likelihood) instead, such as updates in `ADAM`, `RMSPROP`, or even the probabilistic lines search of Chapter 7.
- It is possible to derive different filtering equation for other discrete or continuous Gauss-Markov models of the gradient. The continuous process presented here is the most basic one (Brownian motion), but other ones, e. g., an Ornstein-Uhlenbeck process might perform well, too. They might come with additional parameters, though which also need to be learned.
- For full automation, `KFGRAD` could be combined with the probabilistic line search of Chapter 7. As mentioned above, this might also include the line search using the mean-estimators, as well as variances of the posterior marginal distribution over gradient elements instead of the likelihood only. Other, milder adaptations

might include step-size damping for uncertain search-directions, as in diagonal preconditioners (§ 2.4.2) according to  $\frac{m_i^{\odot 2}}{P_i}$  instead of  $\frac{y_i^{\odot 2}}{\text{diag}[\Sigma_i]}$ , or on a global scale.



## Second-Order Filter for Hessian Elements

---

**A**NALOGOUSLY to the previous one, this chapter develops and derives formulas for Kalman filtering on *Hessians* instead of gradients. Again, the main goal is to design a general probabilistic framework for second-order optimization for gradient evaluations of arbitrary noise levels, and then to draw connections to existing classic optimizers, like members of the Dennis family of quasi-Newton updates, or Broyden’s method. This provides further interpretation of their implicit hyper-parameters, similar to the one done by Hennig [56] for linear solvers which was discussed in Chapter 3. The difference to Hennig [56] will be that the Hessian is *not* constant over the weight-space of  $w$ s, but rather changing with the traveled path distance  $\Delta\tau$ . Specifically, we will represent the dynamics of the true but unknown Hessian function  $\Delta L_{\mathcal{D}}(w)$  of the empirical risk with a Gauss-Markov process, defined only on the optimization path. It will turn out that we re-discover Broyden’s method, as well as Dennis class updates for special choices of diffusion matrices  $Q$ . We will also argue that hyperparameters of successful classic methods are less suitable for stochastic problems and thus should not be transferred blindly. Instead we argue in favor of an empirical Bayes-type approach similar to the `sr1`-update of the Dennis class. The last section considers approximations to the filtering equations for very high-dimensional problems, such that the resulting quasi-Newton update can be computed in linear time. In this Chapter, we will occasionally use the *Einstein summation convention* [35] for better readability. This means that sum-symbols will be dropped if the index of the sum appears twice per term.

[56] Hennig, “Probabilistic Interpretation of Linear Solvers,” 2015

[35] Einstein, “Die Grundlage der allgemeinen Relativitätstheorie,” 1916

### 9.1 A Model for Twice-Differentiable Functions

Classic quasi-Newton methods model the first *two* derivatives of the loss function  $L_{\mathcal{D}}(w)$ . So, for a filter to encode a second derivative as well, we need to include the Hessian of the loss  $L_{\mathcal{D}}(w)$  into the state  $x$ . That is, the enlarged state  $x$  will consist of two parts stacked on top of each other: One for the gradient  $x^{\nabla} \in \mathbb{R}^N$  and one for the (vectorized) Hessian  $x^B \in \mathbb{R}^{N^2}$ . They are linked to each other since one of them is the derivative of the other. Expanding the state not only changes the Gauss-Markov process on  $L_{\mathcal{D}}(w)$  (twice instead of only once-differentiable sample paths), but also makes inference on  $x$  more

challenging: The state  $x$  is significantly larger now (size  $N + N^2$ ), but still only  $N$  informative numbers at each iteration, in the form of a noisy gradient  $\nabla L_S$ , are observed. Even for exact gradient evaluations this is a heavily under-constraint inference problem which can only be solved by introducing prior assumptions on  $\Delta L_{\mathcal{D}}(w)$ . This is analogous to the classic derivation of quasi-Newton methods (§ 2.3.4) where the secant equation  $B_t s_t = \Delta y_t$  also did not identify the estimator  $B_t$  for the Hessian fully, and thus a ‘closeness’-relation from one iterate to the next, i. e., minimizer of the Frobenius norm  $\|B - B_{t-1}\|_{W,F}$  s.t.  $B s_t = \Delta y_t$ , worked as a regularizing term.

The Kalman filter equations now represent a multi-output distribution jointly over the Hessian and gradient elements of  $L_{\mathcal{D}}(w)$  on the one-dimensional optimization path, and not on the whole weight space of  $w$ s, similar to before. The optimization path, was defined in Section 8.1. This implies, for example, that the marginal covariance between Hessian elements  $x_{(ij)}^B(\tau)$  and  $x_{(kl)}^B(\tau')$  depends only on two double-index pairs  $(ij)$  and  $(kl)$  for the output correlation of the Hessian elements, and the *scalar* path locations  $\tau$  and  $\tau'$  along the search direction  $p_t$ , in contrast to arbitrary weight *vectors*  $w$  and  $w'$ .<sup>1</sup> Thus, the filtering approach can be seen as solving  $t$  successive linear systems (with one observation each) as in Chapter 3 that are connected to each other via a dynamic probabilistic model that encodes how these problems might change on the line defined by successive locations  $w_i$  for  $i = 0, \dots, t$ .

In the following derivations, all relevant vectors and matrices will be split into sub-blocks for gradient-gradient, hessian-hessian and gradient-hessian interaction. They will be denoted with the superscript  $\nabla$  for gradient-related, and  $B$  for Hessian related quantities. For example, as mentioned, the state  $x \in \mathbb{R}^{N+N^2}$  can be split into two sub-vectors  $x^\nabla \in \mathbb{R}^N$  and  $x^B \in \mathbb{R}^{N^2}$  with  $x = [x^\nabla; x^B]$ ; the upper part  $x^\nabla$  of the state represents the true gradient function  $\nabla L_{\mathcal{D}}(w)$  analogously to Chapter 8, whereas the lower part  $x^B$  represents the vectorized true Hessian function  $\Delta L_{\mathcal{D}}(w)$ . The Hessian function is the derivative of the gradient which can be encoded in the integral  $\int_0^\tau \Delta L_{\mathcal{D}}(\tau') \bar{s} d\tau' = \nabla L_{\mathcal{D}}(\tau) - \nabla L_{\mathcal{D}}(0)$  for an arbitrary normalized direction  $\bar{s} \in \mathbb{R}^N$ . This directly leads to the secant equation used in classic methods. The most simple Gauss-Markov process for  $x$  can be expressed with the SDE:

$$\underbrace{d \begin{pmatrix} x^\nabla \\ x^B \end{pmatrix}}_{dx} = \underbrace{\begin{pmatrix} 0_{N \times N} & F^{\nabla B} \\ 0_{N^2 \times N} & 0_{N^2 \times N^2} \end{pmatrix}}_F \underbrace{\begin{pmatrix} x^\nabla \\ x^B \end{pmatrix}}_x d\tau + \underbrace{\begin{pmatrix} 0_{N \times N^2} \\ I_{N^2 \times N^2} \end{pmatrix}}_L d\beta_{N^2 \times N^2} \quad (130)$$

with  $F^{\nabla B}$  defined below. The Hessian elements evolve according to a Wiener process with intensity matrix  $q \in \mathbb{R}^{N^2 \times N^2}$ , analogously to

<sup>1</sup> The latter approach was e. g., used in [57] who used a squared exponential kernel to describe the covariance between two locations  $w$  and  $w'$ . This approach suffers from the usual problem, that it is difficult to define covariance functions in very high dimensional spaces, which in fact live on a much lower dimensional manifolds.

[57] Hennig and Kiefel, “Quasi-Newton methods – a new direction,” 2012

the gradient elements in Chapter 8. The gradient  $x^\nabla$  is linked to the Hessian via the block  $F^{\nabla B} = (I \otimes \bar{s}_t^\top)$ . The Kronecker product, symbolized by ‘ $\otimes$ ’, naturally arises if the Hessian is vectorized in the secant equation: For the  $i^{\text{th}}$  component of the vector  $\Delta L_{\mathcal{D}} \bar{s}$  we can write:  $\Delta L_{\mathcal{D}}^{ij} \bar{s}_j = (\delta_{ik} \bar{s}_j) \Delta L_{\mathcal{D}}^{kj} = (I \otimes \bar{s}^\top)_{(i),(kj)} \Delta L_{\mathcal{D}}^{(kj)}$ .

The explicit forms of the transition matrix  $A_t \in \mathbb{R}^{N+N^2 \times N+N^2}$  and the diffusion covariance  $Q_t \in \mathbb{R}^{N+N^2 \times N+N^2}$  of the Kalman filter again can be derived by inserting  $F$  and  $L$  into Eq. 19. The resulting expressions are again analytic, since  $F$  is nilpotent of order two ( $F^2 = 0$ ) and the sum of the matrix exponential  $\exp(F\Delta\tau)$  is finite:

$$\begin{aligned} A_t &= \exp F_t \Delta\tau = \sum_{k=0}^{\infty} \frac{(F_t \Delta\tau)^k}{k!} = (I + F_t \Delta\tau) = \begin{pmatrix} I_{N \times N} & (I \otimes \bar{s}_t^\top) \Delta\tau \\ 0_{N^2 \times N} & I_{N^2 \times N^2} \end{pmatrix} \\ Q_t &= \int_{\tau_t}^{\tau_{t+1}} \begin{pmatrix} (I \otimes \bar{s}_t^\top)(\tau_{t+1} - \kappa) \\ I_{N^2 \times N^2} \end{pmatrix} q \left( (I \otimes \bar{s}_t)(\tau_{t+1} - \kappa) \quad I_{N^2 \times N^2} \right) d\kappa \\ &= \begin{pmatrix} \frac{1}{3} (I \otimes \bar{s}_t^\top) q (I \otimes \bar{s}_t) \Delta\tau^3 & \frac{1}{2} (I \otimes \bar{s}_t^\top) q \Delta\tau^2 \\ \frac{1}{2} q (I \otimes \bar{s}_t) \Delta\tau^2 & q \Delta\tau \end{pmatrix}. \end{aligned} \quad (131)$$

The length of a path segment is again denoted by  $\Delta\tau = \|w_{t+1} - w_t\|$ . The predictive Kalman equations for each block of  $m_{t+1-}$  and  $P_{t+1-}$  are thus:

$$\begin{aligned} m_{t+1-}^\nabla &= m_t^\nabla + (I \otimes s_t^\top) m_t^B \\ m_{t+1-}^B &= m_t^B \\ P_{t+1-}^{\nabla\nabla} &= Q_t^{\nabla\nabla} + P_t^{\nabla\nabla} + P_t^{\nabla B} (I \otimes s_t) + (I \otimes s_t^\top) P_t^{B\nabla} + (I \otimes s_t^\top) P_t^{BB} (I \otimes s_t) \\ P_{t+1-}^{\nabla B} &= Q_t^{\nabla B} + P_t^{\nabla B} + (I \otimes s_t^\top) P_t^{BB} \\ P_{t+1-}^{BB} &= Q_t^{BB} + P_t^{BB} \end{aligned} \quad (132)$$

with  $s := \bar{s} \Delta\tau$ . Eq. 132 is intuitive: Since the drift matrix  $F^{BB}$  is zero, the expected predictive Hessian-state  $m_{t+1-}^B$  is the same as the current one  $m_t^B$  and its block-covariance  $P_{t+1-}^{BB}$  grows according to a Wiener process proportional to the traveled distance  $\Delta\tau$ . This is the same behavior as the predictive mean and covariance of first-order filtering of the previous chapter, just this time it occurs one derivative higher. This is, because in comparison to Chapter 8, now we chose a Gauss-Markov process for the evolution of the Hessian instead of the gradient. Once the Hessian  $x^B$  is known, the gradient  $x^\nabla$  is deterministically linked to it since no further derivatives are encoded by Eq 130. Thus the predictive mean estimator  $m_{t+1-}^\nabla$  of the gradient changes linearly with the current estimate  $m_{t+1-}^B$  of the Hessian, as expected for a quadratic model. Its block-covariance  $P_{t+1-}^{\nabla\nabla}$  increases with the third power of the traveled path  $\Delta\tau$  according to an integrated Wiener process (this was introduced in § 1.4.2).<sup>2</sup>

<sup>2</sup>The expected distance between two Hessian states is:

$$\begin{aligned} \mathbf{E}_{p(x_{t+1}^B | x_t^B)} [\|x_{t+1}^B - x_t^B\|^2] \\ = \text{tr}[q] \|w_{t+1} - w_t\|. \end{aligned}$$

Again this can sloppily be associated with Lipschitz continuity on Hessians where  $\|\Delta L_{\mathcal{D}}(w_{t+1}) - \Delta L_{\mathcal{D}}(w_t)\|_F \leq \mathcal{L}_B \|w_{t+1} - w_t\|$  e.g., in the Frobenius norm.

### Measurements

Analogously to Chapter 8, we will assume here that we only have access to gradients  $\nabla L_S(w_t)$  that are Gaussian distributed with mean  $\nabla L_{\mathcal{D}}(w_t)$  and noise covariance  $R_t = \Sigma(w_t)/|\mathcal{S}| \in \mathbb{R}^{N \times N}$ . The measurement matrix  $H$  thus needs to select the observed part  $x^\nabla$  of the full state  $x$ , i. e.,  $H \in \mathbb{R}^{N \times N+N^2} = [I_{N \times N}, 0_{N \times N^2}]$ . Then, the updated Kalman equations for each block are:<sup>3</sup>

$$\begin{aligned}
 m_{t+1}^\nabla &= m_{t+1-}^\nabla + P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} \left[ y_{t+1} - m_{t+1-}^\nabla \right] \\
 m_{t+1}^B &= m_{t+1-}^B + P_{t+1-}^{B\nabla} G_{t+1}^{-1} \left[ y_{t+1} - m_{t+1-}^\nabla \right] \\
 P_{t+1}^{\nabla\nabla} &= P_{t+1-}^{\nabla\nabla} - P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla\nabla} \\
 P_{t+1}^{\nabla B} &= P_{t+1-}^{\nabla B} - P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B} \\
 P_{t+1}^{BB} &= P_{t+1-}^{BB} - P_{t+1-}^{B\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B},
 \end{aligned} \tag{133}$$

where  $G_{t+1} = P_{t+1-}^{\nabla\nabla} + R_{t+1} \in \mathbb{R}^{N \times N}$  is the innovation covariance. Eq. 132 together with Eq. 133 provide the base-equations for filtering on Hessian-elements where the Gauss-Markov model encodes *two* derivatives and no drift. The algorithm defined by Eqs. 132 and 133 has general cost quartic in  $N$ , assuming the hyper-parameters  $R$  and  $q$  are given. For noise free observation ( $R_t = 0$ ), the gradient part of the expected state collapses onto the observed exact gradient  $m_{t+1}^\nabla = \nabla L_{\mathcal{D}}(w_{t+1})$ , with  $P_{t+1}^{\nabla\nabla} = 0$  and  $P_{t+1}^{\nabla B} = 0$ , since it is fully identified (can be directly seen from Eq. 133 for  $P^{\nabla\nabla} G^{-1} = P^{\nabla\nabla} P^{\nabla\nabla-1} = I$ ). The Hessian part  $x^B$  of the state is not fully identified by a single gradient observation in the same sense as the secant equation of quasi-Newton methods does not uniquely identify Broyden's method or updates of the Dennis class. Thus, the model falls back on the prior and is still uncertain about the Hessian in most directions of the  $N$ -dimensional space. The corresponding covariance block  $P^{BB}$  collapses only in the observed direction; for  $R > 0$  it never collapses completely.

#### 9.1.1 Non-Symmetric Hessian Estimates

Section 9.1 derived general predictive and updated Kalman equations for filtering on the Hessian of  $L_{\mathcal{D}}(w)$ , for the case of Gaussian gradient measurements. Most formulas in 132 and 133 still contain matrices which are extremely large and are thus not usable in practice (e. g.,  $P^{BB}$  is of size  $N^2 \times N^2$  and thus has memory requirement quartic in  $N$ ). As discussed in Chapter 3, classic quasi-Newton methods solve this issue by smartly choosing the structure and values of their hyper-parameters, in this case  $q$  and  $P_0$ . In order to lower the computational complexity of Eqs. 132 and 133, we will use Kronecker structures and

<sup>3</sup> In principle one could think of evaluating e. g., the diagonal of the Hessian on a mini-batch as well, since this is sometimes feasible, e. g., in neural networks [10]. This could be incorporated into the filter by changing the measurement matrix  $H$  accordingly. In general, any observation which is a linear map of the state  $x$ , corrupted by additive Gaussian noise can be included into the update step. This includes all selector maps. We will not explore this further here.

[10] Becker and LeCun, "Improving the Convergence of Back-Propagation Learning with Second-Order Methods," 1989

similar choices for the diffusion matrix  $q \in \mathbb{R}^{N^2 \times N^2}$  and covariance blocks of the current state  $P_t^{BB}$  and  $P_t^{\nabla B}$ :

$$\begin{aligned} P_t^{\nabla B} &= (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top) \quad U_t \in \mathbb{R}^{N \times N}, \quad v_t \in \mathbb{R}^N \\ P_t^{BB} &= W_t \otimes W_t \quad W_t \in \mathbb{R}^{N \times N}, \quad W_t \text{ positive definite} \\ q &= V \otimes V \quad V \in \mathbb{R}^{N \times N}, \quad V \text{ positive definite.} \end{aligned} \quad (134)$$

The matrices  $q$  and  $P$  still have full rank, hence we did not restrict the support of the Gaussian distribution on  $x_t$  by choosing Kronecker form. The predictive Kalman equations of 132 together with 131 and 134 turn into:

$$\begin{aligned} m_{t+1-}^\nabla &= m_t^\nabla + (I \otimes s_t^\top) m_t^B \\ m_{t+1-}^B &= m_t^B \\ P_{t+1-}^{\nabla \nabla} &= P_t^{\nabla \nabla} + \frac{\Delta \tau}{3} (I \otimes s_t^\top)(V \otimes V)(I \otimes s_t) + (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top)(I \otimes s_t) \\ &\quad + (I \otimes s_t^\top)(U_t \otimes U_t)(I \otimes v_t) + (I \otimes s_t^\top)(W_t \otimes W_t)(I \otimes s_t) \\ P_{t+1-}^{\nabla B} &= \frac{\Delta \tau}{2} (I \otimes s_t^\top)(V \otimes V) + (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top) + (I \otimes s_t^\top)(W_t \otimes W_t) \\ P_{t+1-}^{BB} &= (V \otimes V)\Delta \tau + (W_t \otimes W_t) \end{aligned} \quad (135)$$

All predictive covariance blocks are now sums of terms which exhibit the original structure of this block; for example  $P_{t+1-}^{BB}$  is a sum of Kronecker products, and  $P_{t+1-}^{\nabla B}$  and  $P_{t+1-}^{\nabla \nabla}$  are both sums of linearly transformed Kronecker products, where the linear map has the same algebraic structure across terms.

### Measurements

The updated Kalman equations of Eq. 135 can be simplified further. The derivations can be found in Appendix B.2. By combining Eq. 135 with 133, the predictive covariance  $P_{t+1-}^{\nabla \nabla} \in \mathbb{R}^{N \times N}$  can be written as:

$$P_{t+1-}^{\nabla \nabla} = P_t^{\nabla \nabla} + \Delta \tau / 3 V (s_t^\top V s_t) + U_t^\top (v_t^\top U_t^\top s_t) + U_t (s_t^\top U_t v_t) + W_t (s_t^\top W_t s_t). \quad (136)$$

It is thus a linear combination of the current block covariance  $P_t^{\nabla \nabla}$ , diffusion matrix  $V$ , and contributions from Hessian-gradient correlations  $U_t$  as well as uncertainty on the Hessian  $W_t$ . With this, the updated Kalman equations turn into:

$$\begin{aligned} m_{t+1}^B &= m_t^B + (V G_{t+1}^{-1} \Delta_t)(V \tilde{s}_t)^\top + (U_t G_{t+1}^{-1} \Delta_t)(U_t v_t)^\top + (W_t G_{t+1}^{-1} \Delta_t)(W_t s_t)^\top \\ m_{t+1}^\nabla &= m_t^\nabla + (I \otimes s_t^\top) m_t^B + P_{t+1-}^{\nabla \nabla} G_{t+1}^{-1} \Delta_t \\ P_{t+1}^{\nabla \nabla} &= P_{t+1-}^{\nabla \nabla} - P_{t+1-}^{\nabla \nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla \nabla} \\ P_{t+1}^{\nabla B} &= (V - \tilde{V}_t^\top) \otimes (V \tilde{s}_t)^\top + (U_t^\top - \tilde{U}_t^\top) \otimes (U_t v_t)^\top + (W_t - \tilde{W}_t^\top) \otimes (W_t s_t)^\top \end{aligned} \quad (137)$$

$$\begin{aligned}
P_{t+1}^{BB} &= \Delta\tau(V \otimes V) + (W_t \otimes W_t) \\
&\quad - VG_{t+1}^{-1}V \otimes (V\tilde{s}_t)(V\tilde{s}_t)^\top - U_tG_{t+1}^{-1}V \otimes (U_tv_t)(V\tilde{s}_t)^\top - W_tG_{t+1}^{-1}V \otimes (W_ts_t)(V\tilde{s}_t)^\top \\
&\quad - VG_{t+1}^{-1}U_t^\top \otimes (V\tilde{s}_t)(U_tv_t)^\top - U_tG_{t+1}^{-1}U_t^\top \otimes (U_tv_t)(U_tv_t)^\top - W_tG_{t+1}^{-1}U_t^\top \otimes (W_ts_t)(U_tv_t)^\top \\
&\quad - VG_{t+1}^{-1}W_t \otimes (V\tilde{s}_t)(W_ts_t)^\top - U_tG_{t+1}^{-1}W_t \otimes (U_tv_t)(W_ts_t)^\top - W_tG_{t+1}^{-1}W_t \otimes (W_ts_t)(W_ts_t)^\top.
\end{aligned}$$

Here the predictive Kalman equations of 135 are already inserted into the updated equations, except for  $P_{t+1}^{\nabla\nabla}$ . The innovation covariance is again denoted by  $G_{t+1} := P_{t+1}^{\nabla\nabla} + R_{t+1}$  and the residual, which measures the discrepancy between the expected predictive state  $m_{t+1}^{\nabla}$  and the gradient measurement  $y_{t+1}$ , is denoted by  $\Delta_t := y_{t+1} - m_t^{\nabla} - (I \otimes s_t^\top)m_t^B$ . Further notation e. g.,  $\tilde{V}_t$  is a ‘gain-corrected’ version of  $V$ , and introduced in Eq. 217, Section B.2 for notational convenience. The estimator  $m_t^B$  for the Hessian is *not* symmetric, since symmetry was not encoded in the prior on  $x^B$ . In other words, the correlated Wiener process  $\beta$  of Eq. 130 allowed for arbitrary Gaussian increment with covariance  $q$  of *general* Kronecker structure; i. e., it did not encode that the Gauss increments should correlate such that the resulting matrix is symmetric. Nevertheless, for a quasi-Newton optimizer, it is desirable to encode symmetry of Hessians already in the prior assumptions on  $x^B$ . This will be the subject of Section 9.1.2 below.

### Connections

From Eq. 137 it can already be seen that for noise free gradients ( $R, U = 0$ ), the Hessian estimator  $m^B$  resemble Broyden’s method (§ 2.3.4). We will make this connection more explicit in Sections 9.2.1 and 9.2.2. The previously observed behavior that the mean estimator of Wiener processes stays unchanged unless new gradients/ new evidence is collected, is apparent in Eq. 137: The only parts of  $m^B$  which are updated are spanned by a low-rank matrix that is constructed with vectors depending on the residual  $\Delta_t$ , containing gradient differences  $\Delta y_t$ , as well as path segments  $s_t$ . This non-stationary behavior is exhibited by virtually all classic quasi-Newton methods. Corresponding non-stationary probabilistic models thus possibly have an advantage over stationary probabilistic models, such as a GP with a squared exponential kernel. This was already argued in Chapter 2.4, meaning that it is inefficient if models ‘forget’ what they have learned, when moved in  $w$ -space, especially when this space is high-dimensional. In contrast, we would like to overwrite old information with new one, weighted according to its likelihood, as soon as it is available, and rely on it less the further we travel from its observed point by growing the uncertainty.

### Back-Projection for an Iterative Procedure

Before we proceed to symmetric Hessian beliefs, let us investigate how Eq. 137 could be used as an iterative procedure, meaning that successive blocks of  $m_t$  and  $P_t$  will need to have the same algebraic structure. As of now this is not the case since e. g.,  $P_t^{BB}$  is a Kronecker product and  $P_{t+1}^{BB}$  is a sum of Kronecker products. Luckily there is structure in Eq. 137: The vector  $m^\nabla$  is updated with a vector  $\in \mathbb{R}^N$  of same size, and  $m^B$  with low-rank terms. So as long as  $m_0^B$  is of simple form, e. g., a scalar matrix  $m_0 = \sigma_0^B I$ ,  $m_t$  will be, too.

The covariance contributions  $P^{BB}$  and  $P^{\nabla B}$  are sums of Kronecker products (although of different size) and we would like to project it onto a ‘close’ single Kronecker product of the same shape. It turns out that this can be done rather easily under the Frobenius norm by solving the minimization problems (Appendix A.1.2 for derivation and pseudo-code):

$$W_{t+1} \otimes W_{t+1} := \arg \min_W \left\| W \otimes W - P_{t+1}^{BB} \right\|_F^2 \quad (138a)$$

$$U_{t+1} \otimes U_{t+1} v_{t+1} := \arg \min_{U,v} \left\| U \otimes Uv - P_{t+1}^{B\nabla} \right\|_F^2. \quad (138b)$$

This can be computed to high precision by linear algebra operations available in standard libraries.<sup>4</sup> The results hold for dense  $W_t$ ,  $U_t$ , and  $V$ , but especially also for scalar-plus-low-rank structured matrices. In other words, as long as  $V$ ,  $W_t$ , and  $U_t$  are scalar-plus-low-rank ( $V$  can be chosen as such and we will see that  $W_t$  and  $U_t$  will have that form if  $W_0$  and  $U_0$  have),  $W_{t+1}$  and  $U_{t+1}$ , will be, too. This is because the solution to Eq. 138 are matrices which are linear combinations of the matrices contained in their corresponding Kronecker sums. This will become relevant in Section 9.3.

#### 9.1.2 Encoding Symmetric Hessian Beliefs

The Gauss-Markov process on  $x^B$  in Section 9.1.1 did not encode that the Hessian is *symmetric* by definition, hence the derived estimator  $m^B$  in 137 was not either. This section restricts the process on  $x^B$ , such that it yields symmetric matrices only. It then derives filtering equations in the same style as the previous section. We again use Kronecker structure on all relevant matrices

$$\begin{aligned} P_t^{\nabla B} &= (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top) & U_t &\in \mathbb{R}^{N \times N}, & v_t &\in \mathbb{R}^N \\ P_t^{BB} &= W_t \otimes W_t & W_t &\in \mathbb{R}^{N \times N}, & W_t &\text{ positive definite} \\ q &= V \otimes V & V &\in \mathbb{R}^{N \times N}, & V &\text{ positive definite,} \end{aligned} \quad (139)$$

<sup>4</sup>In short: Under the Frobenius norm as in Eq. 138, it is possible to reshuffle the norm-sum with a fixed, known permutation, such that the minimization problem can be rephrased as an equivalent rank-one approximation problem. This can be readily solved by singular value decompositions of the matrices contained in the summands, which is of complexity  $O(N)$  if they already exhibit scalar-plus-low-rank structure, and of negligible cost if an ortho-normal low rank basis (shared by all matrices) is known (§ 9.3).

[56] Hennig, “Probabilistic Interpretation of Linear Solvers,” 2015



but this time using the *symmetric Kronecker product*, symbolized by ‘ $\otimes$ ’. Its definition and some of its algebraic properties were already discussed on in Section 3.2; further properties can be found in Appendix A.2. The matrices  $q$  and  $P^{BB}$  now have reduced rank of  $\frac{1}{2}N(N+1)$  and the support of the prior distribution on  $x_t^B$  equals the space of all symmetric matrices in contrast to arbitrary squared ones [56, Lemma 2.2]. Intuitively Eq. 139 can be thought of as applying the symmetrization operator  $\Gamma$ , defined by  $\Gamma X = 1/2(X + X^T)$  onto an  $N \times N$  reshaped  $x^B$ . Since  $\Gamma$  is linear,  $x^B$  is still Gaussian distributed.<sup>5</sup> The Gaussian increments of the Wiener process now correlate through  $q$  in such a way, that  $x^B$  is always symmetric. The predictive Kalman equations of 132 together with 139 and 131 turn into:

$$\begin{aligned}
m_{t+1-}^{\nabla} &= m_t^{\nabla} + (I \otimes s_t^T) m_t^B \\
m_{t+1-}^B &= m_t^B \\
P_{t+1-}^{\nabla B} &= \frac{1}{2}(I \otimes s_t^T) q \Delta \tau + (I \otimes v_t^T)(U_t^T \otimes U_t^T) + (I \otimes s_t^T)(W_t \otimes W_t) \\
P_{t+1-}^{BB} &= (V \otimes V) \Delta \tau + (W_t \otimes W_t) \\
P_{t+1-}^{\nabla \nabla} &= P_t^{\nabla \nabla} + \frac{1}{3}(I \otimes s_t^T)(V \otimes V)(I \otimes s_t) \Delta \tau + (I \otimes v_t^T)(U_t^T \otimes U_t^T)(I \otimes s_t) \\
&\quad + (I \otimes s_t^T)(U_t \otimes U_t)(I \otimes v_t) + (I \otimes s_t^T)(W_t \otimes W_t)(I \otimes s_t).
\end{aligned} \tag{140}$$

All predictive covariance blocks are again sums of terms which exhibit the original structure of this block in the same way as in Eq 135. The predictive covariance block for gradient-gradient interaction  $P_{t+1-}^{\nabla \nabla}$  can be simplified further (Appendix B.3):

$$\begin{aligned}
P_{t+1-}^{\nabla \nabla} &= P_t^{\nabla \nabla} + \frac{1}{2}[\Delta \tau / 3 V(s_t^T V s_t) + \Delta \tau / 3 (V s_t)(V s_t)^T + U_t^T(v_t^T U_t^T s_t) + (U_t^T s_t)(U_t v_t)^T \\
&\quad + U_t(s_t^T U_t v_t) + (U_t v_t)(U_t^T s_t)^T + W_t(s_t^T W_t s_t) + (W_t s_t)(W_t s_t)^T].
\end{aligned} \tag{141}$$

In contrast to Eq 136, also rank-one terms (outer vector products) appear in the above equation.

### Measurements

Eq 141 as well as the explicit form of the updated Kalman equations for the symmetrized process on the Hessian is derived in Appendix B.3; here again just the results are reported; the predictive Kalman equations, except for  $P_{t+1-}^{\nabla \nabla}$ , are already inserted into the updates. By combining Eq. 140 with 133 we get:

$$\begin{aligned}
m_{t+1}^B &= m_t^B + \frac{1}{2}[(V G_{t+1}^{-1} \Delta_t)(V \tilde{s}_t)^T + (V \tilde{s}_t)(V G_{t+1}^{-1} \Delta_t)^T + (U_t G_{t+1}^{-1} \Delta_t)(U_t v_t)^T \\
&\quad + (U_t v_t)(U_t G_{t+1}^{-1} \Delta_t)^T + (W_t G_{t+1}^{-1} \Delta_t)(W_t s_t)^T + (W_t s_t)(W_t G_{t+1}^{-1} \Delta_t)^T] \\
m_{t+1}^{\nabla} &= m_t^{\nabla} + (I \otimes s_t^T) m_t^B + P_{t+1-}^{\nabla \nabla} G_{t+1}^{-1} \Delta_t
\end{aligned} \tag{142}$$

<sup>5</sup> For this fact, it is much harder to impose positive definiteness of the Hessian estimate, since this would involve a non-linear transformation of  $x^B$ , or a Wishart prior. Gaussians are always more general, since they have support on whole  $R^D$ . Section 9.3 will introduce a way of efficiently projecting each estimator  $m_t^B$  onto the positive definite cone. The connection of a Wishart prior  $\mathcal{W}(x^B; W, \nu)$  for  $x^B$  reshaped to  $N \times N$  to a Gaussian prior on vectorized  $x^B$  with symmetric Kronecker covariance, is that the covariance of this Wishart is given by  $\nu^{-1}(W \otimes W)$ .



$$\begin{aligned}
 P_{t+1}^{\nabla\nabla} &= P_{t+1-}^{\nabla\nabla} - P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla\nabla} \\
 P_{t+1}^{\nabla B} &= (I \otimes \tilde{s}_t^\top)(V \otimes V) + (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top) + (I \otimes s_t^\top)(W_t \otimes W_t) \\
 &\quad - (I \otimes \tilde{v}_t^\top)(\tilde{V}_t^\top \otimes \tilde{V}_t^\top) - (I \otimes \tilde{u}_t^\top)(\tilde{U}_t^\top \otimes \tilde{U}_t^\top) - (I \otimes \tilde{w}_t^\top)(\tilde{W}_t^\top \otimes \tilde{W}_t^\top) \\
 P_{t+1}^{BB} &= \Delta\tau(V \otimes V) + (W_t \otimes W_t) \\
 &\quad - V G_{t+1}^{-1} V \otimes (V \tilde{s}_t)(V \tilde{s}_t)^\top - U_t G_{t+1}^{-1} V \otimes (U_t v_t)(V \tilde{s}_t)^\top - W_t G_{t+1}^{-1} V \otimes (W_t s_t)(V \tilde{s}_t)^\top \\
 &\quad - V G_{t+1}^{-1} U_t^\top \otimes (V \tilde{s}_t)(U_t v_t)^\top - U_t G_{t+1}^{-1} U_t^\top \otimes (U_t v_t)(U_t v_t)^\top - W_t G_{t+1}^{-1} U_t^\top \otimes (W_t s_t)(U_t v_t)^\top \\
 &\quad - V G_{t+1}^{-1} W_t \otimes (V \tilde{s}_t)(W_t s_t)^\top - U_t G_{t+1}^{-1} W_t \otimes (U_t v_t)(W_t s_t)^\top - W_t G_{t+1}^{-1} W_t \otimes (W_t s_t)(W_t s_t)^\top.
 \end{aligned}$$

where  $G_{t+1} = P_{t+1-}^{\nabla\nabla} + R_{t+1}$  again is the innovation covariance and  $\Delta_t := y_{t+1} - m_{t+1}^\nabla - (I \otimes s_t^\top)m_t^B$  the residual (further notation e.g.  $\tilde{V}_t$  is introduced in Eq. 228). The expected value  $m^B$  of the Hessian is now symmetric as well in contrast to the estimator in Eq. 137. Also the uncertainty on  $x^B$  contracts faster, since more information about  $x^B$  can be deduced from a single gradient under stronger prior assumptions. The update  $m^B$  roughly resemble the ones of Dennis class quasi-Newton methods; connections will be discussed in Sections 9.2.1 and 9.2.2. We will call algorithms that arise from the Kalman filter update as in Eqs. 141 and 142 by the general name of KFHESS for ‘Kalman filtering on Hessians’.

#### Back-Projection for an Iterative Procedure

Similar to the back-projection for the non-symmetric process, the updated estimators  $m^\nabla$ ,  $m^B$  and  $P^{\nabla\nabla}$ , already exhibit their original structure. Also the Hessian covariance  $P_{t+1}^{BB}$  can be projected back with the same algorithm as described below Eq. 143 and in Appendix A.2.2, since the minimization problem

$$W_{t+1} \otimes W_{t+1} := \arg \min_W \left\| W \otimes W - P_{t+1}^{BB} \right\|_F^2 \quad (143)$$

is just the square of a linear transformation of a Kronecker sum as in Eq. 143. Thus its solution is the linear transform of the solution if all symmetric Kronecker products were replaced by Kronecker products. In other words let  $\mathcal{S}$  be the linear operator defined by  $\mathcal{S}[A \otimes B] := A \otimes B$  (Eq. 190 in Appendix A.2.1), then  $W_{t+1}$  can be obtained by solving

$$W_{t+1} \otimes W_{t+1} := \arg \min_W \left\| W \otimes W - \tilde{P}_{t+1}^{BB} \right\|_F^2 \quad (144)$$

where  $\tilde{P}_{t+1}^{BB}$  is the same as  $P_{t+1}^{BB}$  of Eq. 142 but all symmetric Kronecker products are replaced by normal Kronecker products.

Unfortunately, the back-projection of  $P_{t+1}^{\nabla B}$  is not as easy as in the non-symmetric case since  $(I \otimes v^\top)(A \otimes B) \neq (A \otimes v^\top B)$  (the right hand

side is not even defined). It is not clear, how to do this properly algorithmically for now. An intermediate practical workaround is to impose independence between the gradient and Hessian part of the state  $x$  such that  $P_t$  is block diagonal with  $P^{\nabla B} = 0$ .<sup>6</sup> The theoretical analysis below is not affected by this.

## 9.2 Recovering Classic Quasi-Newton Methods

The estimators for the Hessian  $m_t^B$  and gradient  $m_t^\nabla$  can be used to construct noise-informed, probabilistic quasi-Newton methods. With slight abuse of notation ( $m_t^B$  occasionally denotes the reshaped  $N \times N$  matrix, instead of the vectorized version) we can write updates of the form:

$$w_{t+1} = w_t - \alpha_t (m_t^B)^{-1} m_t^\nabla \quad \text{or} \quad w_{t+1} = w_t - \alpha_t (m_t^B)^{-1} \nabla L_S(w_t). \quad (145)$$

The updates to the Hessian estimators  $m_t^B$  are of low-rank and thus  $m_t^B$  can be inverted analytically by the matrix inversion lemma, analogously to Dennis-class estimators. Section 9.2.1 will explore connections of noise- and diffusion-free Kalman updates to Broyden's method and the Dennis family, and Section 9.2.2 likewise for noise-free updates with non-zero diffusion.

### 9.2.1 Diffusion- and Noise-Free Updates

Hennig [56] (recap in Chapter 3) established that Broyden's method and the Dennis family of quasi-Newton methods can be seen as one-step Gaussian regression on *constant* symmetric positive definite (spd) Hessian matrices in  $w$ -space, when gradient evaluations are exact ( $R = 0$ ). In addition, the Dennis members  $\text{BFGS}$  and  $\text{DFP}$  are identical to multi-step Gaussian regression for exact line searches, again for constant Hessians.

For this reason we examine the Kalman update equations, first under the assumption that the Hessian is constant, i. e., diffusion  $q$  is zero,  $\Delta L(w) = \Delta L = \text{const}$ , and in Section 9.2.2 for non-quadratic objectives where the Hessian changes with  $w$ . All lemmas assume that the initial state  $x_0^B$  is Gaussian distributed with mean  $m_0^B = \sigma_0^B I$  and (symmetric) Kronecker covariance  $P_0^{BB}$ .

**Lemma 4** *One-step, diffusion-free Kalman updates ( $q = 0$ ) which have access to exact gradient evaluation ( $R = 0$ ), are identical to one-step Gaussian regression updates on matrices as in [56], for both, the symmetric and non-symmetric hypothesis on  $x_t^B$  and same observation pair  $(\Delta y_t, s_t)$ .*

**Proof** For the non-symmetric update: For  $R = V = 0$ , and  $t > 0$  Eq. 135 simplifies to

<sup>6</sup>Note that the majority of the full covariance  $P \in \mathbb{R}^{(N+N^2) \times (N+N^2)}$ , i. e., the block  $P^{BB} \in \mathbb{R}^{N^2 \times N^2}$  as well as the block  $P^{\nabla \nabla} \in \mathbb{R}^{N \times N}$ , is still dense under this approximation. Thus omitting  $U$  might have minor practical implications even. If not, an algorithmic solution for back-projecting  $P^{\nabla B}$  can be found in the future.

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

$$P_{t+1-}^{\nabla\nabla-1} = [(I \otimes s_t^\top)(W_t \otimes W_t)(I \otimes s_t)]^{-1} = \frac{W_t^{-1}}{s_t^\top W_t s_t} \quad \text{and} \quad W_t P_{t+1-}^{\nabla\nabla-1} = \frac{I_{N \times N}}{s_t^\top W_t s_t}. \quad (146)$$

Thus the Kalman updates of Eq. 137 become:

$$\begin{aligned} m_{t+1}^B &= m_t^B + (W_t P_{t+1-}^{\nabla\nabla-1} \Delta_t)(W_t s_t)^\top = m_t^B + \frac{\Delta_t (W_t s_t)^\top}{s_t^\top W_t s_t} \\ P_{t+1}^{BB} &= (W_t \otimes W_t) - W_t P_{t+1-}^{\nabla\nabla-1} W_t \otimes (W_t s_t)(W_t s_t)^\top = W_t \otimes \left( W_t - \frac{(W_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)} \right). \end{aligned} \quad (147)$$

Eq. 147 is equivalent to Eqs. 2.4 and 2.5 in [56] for  $M = 1$  (single step). Analogously for the symmetric update, and by the the matrix inversion lemma, Eq. 140 becomes:

$$\begin{aligned} P_{t+1-}^{\nabla\nabla-1} &= [(I \otimes s_t^\top)(W_t \otimes W_t)(I \otimes s_t)]^{-1} = 2 \frac{W_t^{-1}}{s_t^\top W_t s_t} - \frac{s_t s_t^\top}{(s_t^\top W_t s_t)^2} \\ W_t P_{t+1-}^{\nabla\nabla-1} &= 2 \frac{I_{N \times N}}{s_t^\top W_t s_t} - \frac{(W_t s_t) s_t^\top}{(s_t^\top W_t s_t)^2}. \end{aligned} \quad (148)$$

Thus the corresponding Kalman updates of Eq. 141 are:

$$\begin{aligned} m_{t+1}^B &= m_t^B + \frac{1}{2} [(W_t P_{t+1-}^{\nabla\nabla-1} \Delta_t)(W_t s_t)^\top + (W_t s_t)(W_t P_{t+1-}^{\nabla\nabla-1} \Delta_t)^\top] \\ &= m_t^B + \left[ \frac{\Delta_t (W_t s_t)^\top + (W_t s_t) \Delta_t^\top}{s_t^\top W_t s_t} - \frac{s_t^\top \Delta_t (W_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)^2} \right] \\ P_{t+1}^{BB} &= (W_t \otimes W_t) - W_t P_{t+1-}^{\nabla\nabla-1} W_t \otimes (W_t s_t)(W_t s_t)^\top \\ &= \left( W_t - \frac{(W_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)} \right) \otimes \left( W_t - \frac{(W_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)} \right) \end{aligned} \quad (149)$$

Eq. 149 is identical to Eqs. 2.7 and 2.8 in [56] for  $M = 1$  (single step). ■

**Lemma 5** For the setup as in Lemma 4, multi-step Kalman updates are equivalent to multi-step, auto-regressive Gaussian updates as in [56].

**Proof** Can be directly seen from Eqs. 147 and 149 for  $W_{t+1} := W_t - \frac{(W_t s_t)(W_t s_t)^\top}{(s_t^\top W_t s_t)}$ , and Eqs. 2.4 and 2.7 of [56] for  $M = 1$  (single observation). ■

**Lemma 6** For setup as in Lemma 4 and symmetric updates, as well as a given sequence of search directions  $p_t$ , the Kalman update equations are equivalent to exact Gaussian inference on matrices as in [56] if the search directions are  $W_0$ -conjugate, i.e.,  $p_t^\top W_0 p_{t'} = p_t^\top W_0 p_t \delta_{tt'}$  for all  $t$  and  $t'$ . If the search directions are defined as  $p_t := -(I \otimes m_t^{\nabla\top})(m_t^B)^{-1}$ , and line searches are exact, this is e.g., fulfilled for  $W_0 = \zeta \Delta L$ ,  $\zeta > 0$ , which also recovers the DFP-algorithm in the sense that the sequence of search

directions  $p_t$  (and consequently  $(w_t, y_t)$ -pairs) are identical. The same holds for  $W_0 = \zeta \left( \Delta L + \sqrt{\frac{p_t^\top \Delta y_t}{p_t^\top m_t^B p_t}} m_t^B \right)$  and the BFGS-update. This also means that for  $m_0 = \vec{I}$ , both algorithms are identical to linear conjugate gradients.

**Proof** First note that  $(I \otimes m_t^{\nabla\top})(m_t^B)^{-1} = (m_t^B)^{-1} m_t^\nabla$ , where the left hand side of the equation uses the original vectorized version of  $m_t^B$ , and the right hand side (with overloaded notation) the non-vectorized version. For the first part of the proof, combine Lemma 3.2 of [56] with Lemma 4 above and set  $W_0 = W_{\text{HENNIG}}$ . The second part of the lemma follows then directly from Lemma 3.3 and Theorem 3.4 of [56]. ■

### 9.2.2 Noise-Free Updates

In the previous section, we saw that it is possible to recover classic one-step quasi-Newton updates with a diffusion-free filter and exact gradient observations. Thus, the question arises if it is also possible to construct equivalent *multi-step* Kalman filter updates, if the filter adapts its intensity  $q$ , or to a less rigorous degree its diffusion  $Q$ , in such a way that classic methods, like the BFGS-rule or Broyden's method get recovered. Differing  $q$  or  $Q$  per iteration in essence imply a different scale for the variability of the Hessian elements as well as different correlations. We will do this here for the symmetric case only, since it is usually more involved, but the results translate directly to the non-symmetric updates as well, by replacing the symmetric Kronecker products with non-symmetric ones. Firstly, the predictive Kalman filter equations for  $q \neq 0$  and  $R = 0$  are:

$$\begin{aligned} P_{t+1-}^{\nabla\nabla} &= \frac{\Delta\tau}{3} (I \otimes s_t^\top) q (I \otimes s_t) + (I \otimes s_t^\top) (W_t \otimes W_t) (I \otimes s_t) \\ P_{t+1-}^{\nabla B} &= \frac{\Delta\tau}{2} (I \otimes s_t^\top) q + (I \otimes s_t^\top) (W_t \otimes W_t) \\ P_{t+1-}^{BB} &= \Delta\tau q + (W_t \otimes W_t) \end{aligned} \quad (150)$$

From the previous section, we know that we recover the Dennis class on  $m^B$  if

$$\begin{aligned} P_{t+1-}^{\nabla\nabla} &\stackrel{!}{=} (I \otimes s_t^\top) (W_{\text{DEN}} \otimes W_{\text{DEN}}) (I \otimes s_t) \\ P_{t+1-}^{\nabla B} &\stackrel{!}{=} (I \otimes s_t^\top) (W_{\text{DEN}} \otimes W_{\text{DEN}}) \\ P_{t+1-}^{BB} &\stackrel{!}{=} (W_{\text{DEN}} \otimes W_{\text{DEN}}), \end{aligned} \quad (151)$$

where  $W_{\text{DEN}}$  is defined up to a positive scalar as in Eq. 72, and the subscript DEN is a placeholder for a member of the Dennis family, e. g., DFP. Thus, for the filter-update  $m^B$  to be identical, we require:

$$\Delta\tau q = Q^{BB} \stackrel{!}{=} (W_{\text{DEN}} \otimes W_{\text{DEN}}) - (W_t \otimes W_t) \quad (152a)$$

$$\frac{1}{2} \Delta\tau (I \otimes s^\top) q = Q^{\nabla B} \stackrel{!}{=} (I \otimes s^\top) [(W_{\text{DEN}} \otimes W_{\text{DEN}}) - (W_t \otimes W_t)] \quad (152b)$$

$$\frac{1}{3} \Delta\tau (I \otimes s^\top) q (I \otimes s) = Q^{\nabla\nabla} \stackrel{!}{=} (I \otimes s^\top) [(W_{\text{DEN}} \otimes W_{\text{DEN}}) - (W_t \otimes W_t)] (I \otimes s). \quad (152c)$$

Thus the Dennis-updates can be recovered for blocks of  $Q$  as in Eq. 152, but we can also see that all three equations can not simultaneously be fulfilled for some common intensity matrix  $q \in \mathbb{R}^{N^2 \times N^2}$ , due to the factors  $1/2$  and  $1/3$  on the left hand side. This leads to the following theorem:

**Theorem 7** *Assume exact gradient evaluations ( $R = 0$ ), and the same sequence of pairs  $(w_t, y_t)$ , i. e.,  $s_t := (I \otimes m_t^{\nabla\top})(m_t^B)^{-1}$  and the same learning rates  $\alpha_t$ . Then, the Kalman filter updates as in Eq. 133 recover the Dennis family of quasi-Newton updates for possibly indefinite diffusion matrices  $Q$  given by Eq. 152. Additionally, the same can not generally be constructed for diffusions arising from an integrated Wiener process with intensity matrix  $q$  as in Eq. 130 and 131.*

The structure is, however, very close to that of a Wiener process, since matrix blocks just differ by a constant relative factor, i. e., if  $q$  is chosen such that  $Q^{BB} \stackrel{!}{=} Q_{\text{DEN}}^{BB}$ , then the other blocks need to be multiplied by constant factors as follows:  $Q^{\nabla B} \rightarrow 2 \cdot Q^{\nabla B}$  and  $Q^{\nabla\nabla} \rightarrow 3 \cdot Q^{\nabla\nabla}$ , in order to recover the Dennis-updates.<sup>7</sup>

We can ask now if it is possible to generally find a Gaussian model, not necessarily only one that is based on a Wiener process, which recovers the Dennis family. For this, we need the matrix  $Q$  to be symmetric positive semi-definite (semi-spd) such that  $p(x_{t+1}|x_t)$  defines a Gaussian distribution. This is only satisfied if the difference  $(W_{\text{DEN}} \otimes W_{\text{DEN}}) - (W_t \otimes W_t)$  is semi-spd. Since  $W_{\text{DEN}}$  and  $W_t$  are both coupled to the identity matrix, the Hessian estimator  $B_t$ , or the mean Hessian  $\Delta\bar{L}_D$ , depending on the Dennis member, this is not satisfied in general.

An easy instance is the Dennis-update  $\text{PSB}$ , where  $W_{\text{DEN}} = I$  and  $W_t = I - \frac{s_{t-1}s_{t-1}^\top}{(s_{t-1}^\top s_{t-1})}$ , where we used  $\zeta = 1$  for simplicity and without loss of generality. If we choose an ortho-normal basis  $P \in \mathbb{R}^{N \times N}$  of  $W_t$  with  $P^\top P = I$ , where the first column is equal to  $s_{t-1}/\|s_{t-1}\|$ , then we can write

$$\begin{aligned} (W_{\text{DEN}} \otimes W_{\text{DEN}}) - (W_t \otimes W_t) &= \Gamma(P \otimes P) [(I \otimes I) - (I - e_1 e_1^\top) \otimes (I - e_1 e_1^\top)] (P \otimes P)^\top \Gamma^\top \\ &= (P \otimes P) [I_{N^2} - K] (P \otimes P)^\top = (P \otimes P) D (P \otimes P)^\top =: Q_t^*, \end{aligned} \quad (153)$$

where  $e_1$  is the first Cartesian vector and  $\Gamma$  is the linear operator defined in Eq. 186 that maps a Kronecker product onto a symmetric Kronecker product, and  $D := I - K$  a diagonal matrix.<sup>8</sup> We would like to find the

<sup>7</sup> Relative factor means that the same correspondence could be achieved by choosing  $q$  such that  $Q^{\nabla\nabla} \stackrel{!}{=} Q_{\text{DEN}}^{\nabla\nabla}$ , then  $Q^{\nabla B} \rightarrow 2/3 \cdot Q^{\nabla B}$  and  $Q^{BB} \rightarrow 1/3 \cdot Q^{BB}$ . Thus blocks have different sizes relative to each other, once an absolute scale is fixed.

<sup>8</sup> We also used that  $(P \otimes P) = \Gamma(P \otimes P) \Gamma^\top = \Gamma(P \otimes P)$ . For general Kronecker products this does not hold, i. e.,  $\Gamma(A \otimes B) \Gamma^\top \neq \Gamma(A \otimes B)$ .

non-zero eigenvalues of the matrix  $Q_t^*$ . These are at most  $\frac{1}{2}N(N+1)$ , since this is the maximal rank of  $Q_t^*$ . Define the matrices  $v_{(ij)} := p_i p_j^\top + p_j p_i^\top$  and  $e_{(ij)} = e_i e_j^\top + e_j e_i^\top$  where  $p_i \in \mathbb{R}^N$  is the  $i^{\text{th}}$  column of the ortho-normal basis  $P$  and  $e_i$  the  $i^{\text{th}}$  Cartesian vector. We can see that  $e_{(ij)}$  form the (not-normalized) standard basis for symmetric  $N \times N$  matrices, which is of rank  $\frac{1}{2}N(N+1)$ . Since the matrices  $v_{(ij)}$  and  $e_{(ij)}$  are congruent via the invertible transformation  $P$ , i. e.,  $v_{(ij)} = P e_{(ij)} P^\top$  with  $P P^\top = I$ , also  $v_{(ij)}$  form a basis. Vectorization does not change the linear independence, thus a corresponding vectors-basis is given by  $\overrightarrow{v_{(ij)}} = \overrightarrow{p_i p_j^\top + p_j p_i^\top} = (P \otimes P) \overrightarrow{e_i e_j^\top + e_j e_i^\top}$ , with  $(P \otimes P)(P \otimes P)^\top = I$ . We can now find all  $\frac{1}{2}N(N+1)$  non-zero eigenvalues of  $Q_t^*$  since

$$Q_t^* \overrightarrow{v_{(ij)}} = (P \otimes P) D (P \otimes P)^\top \overrightarrow{v_{(ij)}} = \left( D_{(ij)(ij)} + D_{(ji)(ji)} \right) \overrightarrow{v_{(ij)}}. \quad (154)$$

From Eq. 154 we see that they are given by all possible combinations  $\lambda_{(ij)} := D_{(ij)(ij)} + D_{(ji)(ji)}$ . From the definition of  $K$  and  $D$  we see that  $D$  is the zero-matrix with a few '1'-entries on the diagonal, in particular  $\lambda_{(ij)} = \delta_{j1} + \delta_{i1} - \delta_{j1} \delta_{i1} \in \{0, 1\}$ . Since the eigenvalues  $\lambda_{(ij)}$  are greater or equal to zero,  $Q^*$  is semi-spd. We can therefore formulate the following theorem.

**Theorem 8** *Assume noise free observations ( $R = 0$ ), a matrix  $Q_t^*$  as in Eq. 153, and a diffusion matrix  $Q_t$  that is constructed from the blocks  $Q_t^{BB} = Q_t^*$ ,  $Q_t^{\nabla B} = (I \otimes s_t^\top) Q_t^*$ , and  $Q_t^{\nabla \nabla} = (I \otimes s_t^\top) Q_t^* (I \otimes s_t)$ . Then, the Kalman filter updates as in Eq. 133 recover the pSB-updat. Additionally, since  $Q_t^*$  is symmetric positive semi-definite, there exists corresponding Gaussian distributions  $x_{t+1} \sim \mathcal{N}(A_t x_t, Q_t)$ , and thus a Gauss-Markov-model for  $x_t$ . The identical algorithm arises for  $p_t = -(m_t^B)^{-1} y_t$ , and same  $(w_t, y_t)$ -pairs, i. e., for identical learning rates.*

Note that Theorem 8 did not impose further restrictions on  $L_{\mathcal{D}}(w)$  and also does not assume exact line searches, as e. g., done in Hennig [56]. For other members of the Dennis family, the same would apply if, exchanging sub- with superscript,

[45] Golub and Van Loan, *Matrix computations*, 1996

[56] Hennig, "Probabilistic Interpretation of Linear Solvers," 2015

$$(W_t^{\text{DEN}} \otimes W_t^{\text{DEN}}) - (W_{t-1}^{\text{DEN}} - \frac{(W_{t-1}^{\text{DEN}} s_{t-1})(W_{t-1}^{\text{DEN}} s_{t-1})^\top}{s_{t-1}^\top W_{t-1}^{\text{DEN}} s_{t-1}}) \otimes (W_{t-1}^{\text{DEN}} - \frac{(W_{t-1}^{\text{DEN}} s_{t-1})(W_{t-1}^{\text{DEN}} s_{t-1})^\top}{s_{t-1}^\top W_{t-1}^{\text{DEN}} s_{t-1}}) \quad (155)$$

is semi-spd. One might ask if conditions can be imposed on the estimator  $B_t$  and on the loss  $L_{\mathcal{D}}(w)$ , such that 155 is always semi-spd. Interesting corner cases might include quadratic losses where  $\Delta L_{\mathcal{D}} = \text{const}$ , or Hessians  $\Delta L_{\mathcal{D}}(w)$  that all share the same eigen-basis. Alternatively one might attempt to find weaker restrictions on  $\Delta L_{\mathcal{D}}$  only on the optimization path. A promising direction might be to decompose Eq. 155 with a generalized eigen-decomposition for  $W^{\text{DEN}}$  and  $W_t$ , and then to impose restrictions on the eigenspectrum. See e. g., [45, § 8.7], Theorems 8.71 and 8.72.

### Scales of Dennis-Covariances

Since classic quasi-Newton methods are succesful in practice, it might be beneficial to get a better intuition about the relative scaling of the Dennis-covariances  $W_{\text{DEN}}$  between Hessian elements only. Consider a Hessian function  $\Delta L_{\mathcal{D}}(w)$  with  $N = 2$ -dimensional input  $w$  and noise free gradient observations ( $R = 0$ ). Figure 58 shows the diagonal of the Hessian estimator  $B_t^{\text{DEN}}$  of five members of the Dennis class on a 2D-toy function for absolute scale  $\zeta = 1$  as in Eq. 72. Top and bottom row show the (1, 1) and (2, 2)-element of  $\Delta L_{\mathcal{D}}(w(\tau))$  respectively (the (2, 2)-element is constant). The true Hessian is shown in blue (—) and the Dennis-estimator  $B^{\text{DEN}}$  in black (—), 2 standard deviations  $2 \text{diag}[W_{\text{DEN}}]$  shaded (■). Note that all standard devia-

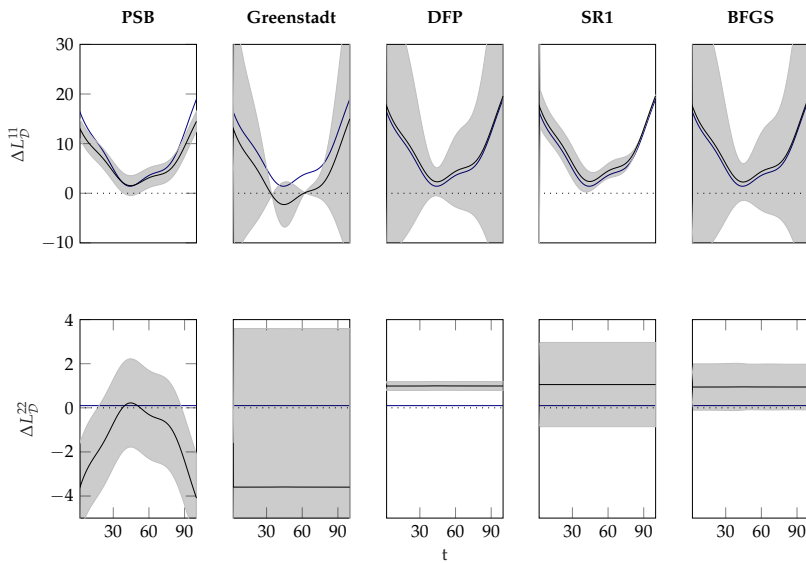


Figure 58: Sketch of Dennis-class hyperparameters. Top and bottom row: (1, 1)-element and (2, 2)-element of the Hessian  $\Delta L_{\mathcal{D}}(w)$  respectively. True Hessian (—), Dennis-estimator (—), and  $2 \text{diag}[W_{\text{DEN}}]$  (■).

tions  $\text{diag}[W_{\text{DEN}}]$  (■) could be scaled up or down by a positive scalar shared across all dimensions; so we are interested in their behavior relative to the evolution of  $\Delta L_{\mathcal{D}}$  and the estimator  $B^{\text{DEN}}$ , as well as how a shared fitted absolute scale would affect the individual dimensions.<sup>9</sup> From the figure, we can deduce the following points:

- **PSB** is equally uncertain, independent of the Hessian estimator  $B_t$  or the true Hessian  $\Delta L_{\mathcal{D},t}$ . This is probably the simplest, yet possibly effective, approach, a global uncertainty scale, assuming  $B$  stays in a reasonable range. If this concept was used for  $\text{KF}_{\text{HES}}$ ,  $q$  would adapted such that the posterior uncertainty would be isotropic and probably quite conservative for most directions.
- **GREENSTADT** scales its uncertainty proportional to the magnitude of the Hessian estimator  $B_t$ . While it might be true that larger parts of the estimator also “need larger variances”, it is still less

<sup>9</sup>Since  $W_{\text{DEN}}$  occasionally depends on the quantities  $B^{\text{DEN}}$  and  $\Delta L_{\mathcal{D}}$ , positive definiteness of  $B^{\text{DEN}}$  and  $\Delta L_{\mathcal{D}}$  will be assumed where necessary for the sake of interpreting  $W_{\text{DEN}} \otimes W_{\text{DEN}}$  as covariance matrix.

avored from an uncertainty point of view, since it vanishes for vanishing estimators ( $t \approx 30$  and  $70$  in the figure). This means that the uncertainty might be scaled down although  $B_t$  might be far off, or likewise, but less concerning, scaled up although the estimator might be performing well ( $t < 20$  and  $t > 80$ ). Most definitely it would be troublesome to find a well working absolute scale for all dimensions simultaneously as can be seen by scaling  $W$  in the bottom row up and down.

- **DFP** is similar to **GREENSTADT** in the sense that it couples the uncertainty to the Hessian, but this time to the true mean value  $\Delta\bar{L}_{\mathcal{D}}$  instead of to the estimator  $B$ . Again this only scales the uncertainty up for large Hessians relative to small Hessians, but does not encode if the current estimator is actually doing well and thus an absolute scale is hard to find.
- **BFGS** is a combination of **GREENSTADT** and **DFP** in the sense that it scales the uncertainty up if either the estimator or the true Hessian is large.<sup>10</sup> This is beneficial since it is virtually never overconfident for a large enough  $\zeta \geq 1$ . But it also means that it is still very underconfident for potentially very good estimators, as can be seen from the figure: The estimator  $B_t$  is quite good nearly everywhere but the uncertainty wildly varies. **BFGS** usually does not bother about this relative under-confidence, since the residual  $\Delta = \Delta y - Bs$  vanishes in that case and the *update* to  $B_t$  vanishes, too. In other words, the relative scaling of  $W_{\text{BFGS}}$  is sensible as long as the residual  $\Delta$  is large as well, but it is not sensible for vanishing residuals  $\Delta$ . For noisy gradient observations, and thus noisy residuals  $\Delta y$ , the latter will virtually never be the case and thus **BFGS**-type scaling might be less favorable, too.
- **sr1** possesses the only covariance, among the five shown, which encodes the difference of the Hessian estimator  $B_t$  to the true mean Hessian  $\Delta\bar{L}_{\mathcal{D},t}$  and has thus, from an uncertainty perspective, the arguably best approach to coupling  $W_{\text{DEN}}$  to these quantities. This can be seen especially at iterations  $t > 70$  where  $m_t$  and  $\Delta L$  are very close and also the uncertainty shrinks. This is a feature that all other methods, including **BFGS**, lack.<sup>11</sup> It also enables the possibility to adjust a reasonable absolute scale  $\zeta$  which yields an uncertainty that is neither too conservative nor specifically over-confident for all Hessian elements.

The points above shed light on how classic quasi-Newton methods tune their relative uncertainty: For achieving good estimators  $B_t$ , some, including the state-of-the-art **BFGS**-optimizer, additionally rely on the residual  $\Delta$  to vanish when the estimator is well calibrated, and trade this assumption for a  $W_{\text{DEN}}$  that is possibly not well scaled in

<sup>10</sup> The hyperparameter  $W_{\text{BFGS}}$  can even be rewritten as a convex combination of  $B_t$  and  $\Delta\bar{L}_{\mathcal{D},t}$  such that:

$$W_{\text{BFGS}} = \zeta(\theta\Delta\bar{L}_{\mathcal{D},t} + (1-\theta)B_t)$$

$$\theta = \left(1 + \sqrt{\frac{s_t^T B_t s_t}{s_t^T \Delta y_t}}\right)^{-1} \in (0,1).$$

If  $B_t = \Delta\bar{L}_{\mathcal{D},t}$  then  $\theta = \frac{1}{2}$  but  $W_{\text{BFGS}} \neq 0$ . For  $s_t^T B_t s_t \gg s_t^T \Delta\bar{L}_{\mathcal{D},t} s_t$ ,  $\theta \rightarrow 0$  and  $W_{\text{BFGS}} \rightarrow \zeta B_t$ , while for  $s_t^T B_t s_t \ll s_t^T \Delta\bar{L}_{\mathcal{D},t} s_t$ ,  $\theta \rightarrow 1$  and  $W_{\text{BFGS}} \rightarrow \zeta \Delta\bar{L}_{\mathcal{D},t}$ .

<sup>11</sup> The factor  $\theta$  of **BFGS** also compares  $B$  to  $\Delta\bar{L}_{\mathcal{D}}$ , but, depending on their magnitudes, just shifts  $W_{\text{DEN}}$  to the larger of the two instead of shrinking it when they are close, or scaling it up when they are different. Thus **BFGS** encodes their *relative magnitude*, while **sr1** encodes their *difference*.



those cases. Ultimately this strategy is justified since only the estimator counts. This means that the  $W$ -adaptation of e. g., `BFGS` is less preferable when  $\Delta$ , which appears in all of the Dennis-updates, is not exact anymore, i. e., when gradients are noise corrupted and  $\Delta$  virtually never vanishes.<sup>12</sup> Thus, when one is concerned about the scale  $W$  not only for non-vanishing  $\Delta$ , then e. g., the adaption of `SR1` should be a favorable choice.<sup>13</sup>

This also means that the best performing classic hyperparameter choices are not necessary the best choice for probabilistic quasi-Newton methods, where  $\Delta$  is a random variable, too, and features like positive definite estimates  $m^B$  might not be needed or can be insured in different ways (see § 9.3). The approach taken by `SR1`, which resembles a local maximum marginal likelihood estimator, or possibly just an isotropic adaptive scale as in `PSB`, might indeed be more appropriate. A first cautious approach to the hyperparameter setting of  $q$  based on maximum marginal likelihood estimators is discussed in Appendix B.4.4 and left out here, since it is not used for the experiments.

### 9.3 Towards a Fast Solver: Low-Rank Approximations

Depending on the input-dimension  $N$ , for practical algorithms, the filtering updates will need to be approximated further. Lower-dimensional noisy problems already might benefit from updates as in Eqs. 141 and 142 which are cubic in cost, like Newton’s method. In this section we will focus on the more technical case of very high dimensional inputs  $N$ . In this scenario, all computations need to be resource and memory efficient, in particular with cost linear in  $N$ . Classic quasi-Newton methods solve this issue by limited memory versions of their full updates: A small number  $M < 10$  of lastly evaluated gradients  $y$  and path segments  $s$  are kept in storage. Pictorially this restarts the estimation of  $\Delta L_{\mathcal{D}}$  with a shifting memory window of size  $M$  and yields a scalar-plus-low-rank estimator  $B_M$  of the Hessian, such that the search direction  $-B_M^{-1}\nabla L_{\mathcal{D}}$  can be computed in linear time. Intuitively this only works well if there is structure in  $\Delta L_{\mathcal{D}}(w)$ , which often seems to be the case in practice; for example the limited memory version `L-BFGS` [99] of the `BFGS` optimizer is widely successful and sometimes even performs better than vanilla `BFGS`.

A first approach to scalable probabilistic quasi-Newton updates is thus to impose a limited memory, too. There is an up and a downside to this. On the one hand, the Kalman filter updates will be of same complexity as classic limited memory versions and can thus be computed very fast, on the other hand, the information contained in only a handful of noisy gradients might not be enough to construct

<sup>12</sup> This is supposed to mean that, if the gradients  $y$  are instances drawn from a distribution, then also  $\Delta$  are, and the ‘true’ (expected) residual is unknown and generally non-vanishing.

<sup>13</sup> One might wonder, why `SR1` does not do better than `BFGS` even in the noise free case, since  $W_{\text{DEN}}$  seems to be scaled well everywhere. In practice though, there are further benefits of `BFGS` which are crucial for a good overall optimization performance, such as being able to ensure positive definite  $B$  (and thus descent directions) while still being able to scale  $W_{\text{DEN}}$  reasonable well for large  $\Delta$ s. In a sense, `BFGS` might be the better trade-off between algorithmic appeal and relative uncertainty estimation under noise free gradients which ensure exact vanishing residuals.

[99] Nocedal, “Updating quasi-Newton matrices with limited storage,” 1980

[26] Dauphin et al., “Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization,” 2014

[20] Byrd et al., “A Stochastic Quasi-Newton Method for Large-Scale Optimization,” 2014

a low-rank basis of  $\Delta L_{\mathcal{D}}$  that is robust to stochastic gradient evaluations. In contrast to this, a fully fletched filter with a dense but analytic  $(m^B)^{-1}$  as in Eq. 142 will automatically average over all past observations, weighted by the gains. This is closely related to the discussion in Section 2.4 on noisy, high dimensional Gaussian random vectors. We will postpone the discussion on robustness for now and start with deriving an approximate filter.<sup>14</sup>

For tractability reasons, we simplify all relevant matrices to scalar-plus-low-rank structure:

$$\begin{aligned}
m_t^B &= \sigma_t^B I + P_t^B \Lambda_t^B P_t^{B\top} & \Lambda_t^B &= \Lambda_t^{B\top} \in \mathbb{R}^{2M \times 2M}, & \sigma_t^B &\in \mathbb{R}_+ \\
W_t &= \sigma_t^W I + P_t^W \Lambda_t^W P_t^{\top} & \Lambda_t^W &= \Lambda_t^{W\top} \in \mathbb{R}^{M \times M}, & \sigma_t^W &\in \mathbb{R}_+ \\
V_t &= \sigma_t^V I + P_t^V \Lambda_t^V P_t^{\top} & \Lambda_t^V &= \Lambda_t^{V\top} \in \mathbb{R}^{M \times M}, & \sigma_t^V &\in \mathbb{R}_+ \\
U_t &= \sigma_t^U I + P_t^U \Lambda_t^U P_t^{\top} & \Lambda_t^U &\neq \Lambda_t^{U\top} \in \mathbb{R}^{M \times M}, & \sigma_t^U &\in \mathbb{R} \\
R_t &= \sigma_t^R I + P_t^R \Lambda_t^R P_t^{R\top} & \Lambda_t^R &= \Lambda_t^{R\top} \in \mathbb{R}^{M \times M}, & \sigma_t^R &\in \mathbb{R}_+ \\
P_t^{\nabla\nabla} &= \sigma_t^{\nabla} I + P_t^{\nabla} \Lambda_t^{\nabla} P_t^{\top} & \Lambda_t^{\nabla} &= \Lambda_t^{\nabla\top} \in \mathbb{R}^{M \times M}, & \sigma_t^{\nabla} &\in \mathbb{R}_+.
\end{aligned} \tag{156}$$

The low rank terms of classic quasi-Newton updates are spanned by the past  $M$  gradient differences  $\Delta y$  that are still in storage as well as their corresponding path segments  $s$ , leading to a basis of size  $2M$ . We will assume the same here and define a low-rank basis  $P^B \in \mathbb{R}^{N \times 2M}$  for the (reshaped) Hessian estimator  $m^B$  that is spanned by the last  $M$  gradient evaluations  $\{y_{t-i+1}\}_{i=1\dots M+1}$  (this yields  $M-1$  past gradient differences) as well as past  $M-1$  path segments  $\{s_{t-i+1}\}_{i=1\dots M}$ . All other objects in Eq. 156 share a reduced basis  $P \in \mathbb{R}^{N \times M}$ , which is only spanned by path segments  $\{s_{t-i+1}\}_{i=1\dots M}$ , such that  $P^B = [P, P^y]$ .<sup>15</sup> This is reasonable since contraction of covariances should mostly happen in dimensions spanned by the observed projections of the Hessian. Additionally, since the basis  $P^B$  can be always ortho-normalized, e. g., by Gram-Schmidt, we will use throughout that  $P^{B\top} P^B = I_{2M \times 2M}$  and  $P^{\top} P = I_{M \times M}$ .<sup>16</sup>

Combining Eq. 156 with Eqs. 140 and 142 yields the predictive equations

$$\begin{aligned}
m_{t+1-}^{\nabla} &= P_-^B \pi_{t+1-}^{m^{\nabla}} \\
P_{t+1-}^{\nabla\nabla} &= \sigma_{t+1-}^{\nabla} I + P_- \Lambda_{t+1-}^{\nabla} P_-^{\top},
\end{aligned} \tag{157}$$

and updated equations

$$\begin{aligned}
G^{-1} &= \sigma_{t+1}^{G^{-1}} I + P_- \Lambda_{t+1}^{G^{-1}} P_-^{\top} \\
m_{t+1}^{\nabla} &= P_{t+1}^B \pi_{t+1}^{m^{\nabla}} \\
m_{t+1}^B &= \sigma_{t+1}^B I + P_{t+1}^B \Lambda_{t+1}^B P_{t+1}^{B\top} \\
P_{t+1}^{\nabla\nabla} &= \sigma_{t+1}^{\nabla} I + P_- \Lambda_{t+1}^{\nabla} P_-^{\top}
\end{aligned} \tag{158}$$

<sup>14</sup> Potential extensions might be to stabilize the basis  $P$  and  $P^B$ , defined below (overloaded notation with the covariance  $P$ ) by averaging it, similar to e. g., [26] who smooth the first eigen-direction of  $\Delta L$  from noisy as well as approximate evaluations thereof; or advocated by [20].

<sup>15</sup> With this definitions, the matrices  $P^y$  contains one column less than  $P^y$ . This is because  $P$  encodes path segments instead of locations, but  $P^y$  encode gradients instead of gradient differences. Other parameterization, i. e., a basis using gradient differences  $\Delta y$ s are also possible, but algorithmically less appealing. In a practical implementation, it is easier to work with matrices of same size, thus  $P$  is initialized with the zero vector such that it contains an additional column of zeros. We will also use this notation for the derivations below and consider  $P$  and  $P^y$  of same size  $N \times M$ .

<sup>16</sup> As mentioned above, precisely speaking, the first of the diagonal elements of  $P^{B\top} P^B$  is zero in a practical implementation, since  $P$  contains a column of zeros. This is not relevant for the derivations below, nor does it need special care during the implementation.

$$\begin{aligned}
P_{t+1}^{BB} &= \sum_{i=1}^{11} \left( \sigma_i^{L, BB} I + P_- \Lambda_i^{L, BB} P_-^\top \right) \otimes \left( \sigma_i^{R, BB} I + P_- \Lambda_i^{R, BB} P_-^\top \right) \\
P_{t+1}^{\nabla B} &= \sum_{j=1}^6 \left( I \otimes \zeta_j^\top \right) \left( \sigma_j^{L, \nabla B} I + P_- \Lambda_j^{L, \nabla B} P_-^\top \right) \otimes \left( \sigma_j^{R, \nabla B} I + P_- \Lambda_j^{R, \nabla B} P_-^\top \right).
\end{aligned}$$

All matrices  $\Lambda$  are of very small size  $M \times M$ , or  $2M \times 2M$ , and all  $\sigma$  are positive scalars. Also, all matrices and vectors can be expressed again only in terms of the low-rank basis  $P_-^B$ , which is the same as  $P^B$  with a newly added observation-pair  $(s_t, y_t)$ . The precise formulas for updating them are reported in Appendix B.4 and left out here for readability. Importantly, all corresponding computations are of small cost and only involve the memory limit  $M$ ; in particular, they are independent of the input-dimensionality  $N$ , once the basis  $P_-^B$  is computed. The latter has to be done once per gradient observation in storage and is of cost linear in  $N$ .

Computing the search direction as in Eq. 145 is linear in  $N$  because the inverse of  $m_t^B$  is analytic by the matrix inversion lemma, and the low-rank part of the inverse is also spanned by the ortho-normal basis  $P_-^B$ . Between iterations, the back-projection of  $P^{BB}$  onto Kronecker form is of negligible cost (Appendix B.4.3), since the inner products needed also only involve  $\Lambda$ s and  $\sigma$ s. It is also possible to manipulate the eigen-values of  $m^B$ , since they can be obtained by computing the eigen-decomposition of  $\Lambda^B \in \mathbb{R}^{2M \times 2M}$  only. Thus, even if the estimators  $m^B$  might exhibit negative eigen-values, the *update* can be forced to use a positive definite  $m^{B, \text{pos}}$  with negligible computation overhead.<sup>17</sup> We will call the corresponding algorithm L-KFHES, for the limited memory version of ‘Kalman filtering on Hessians’.

Comments and formulas on hyperparameters adaptation, especially for  $R_t$  and  $V$  can be found in Appendix B.4.4. In particular, the observation noise  $R_t$  can again be estimated within the mini-batch using  $\hat{\Sigma}$ , and  $V_t$  by maximum marginal likelihood estimation.

## 9.4 Experiments

A first test for L-KFHES are noise-free problems where exact gradients are available, since, if it performs very poorly there, it probably also will on noisy problems. We use the same MLP as in Section 8.3.3 and train it on MNIST for feasibility. For comparison, we also run full-batch gradient descent (GD), as well as the state-of-the-art optimizer L-BFGS and compare the performances. The cost per step is larger for L-BFGS and KFHESS updates in comparison to GD by a constant factor but they all are of same complexity linear in  $N$ . We will be mostly interested in how L-KFHES performs relative to GD, and regard L-

<sup>17</sup> Flipping negative eigen-values is e. g., advocated by [26]. The rational is that the magnitude of the eigen-value still encodes a proper length-scale, just the direction is flipped.

[26] Dauphin et al., “Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization,” 2014

```

1: function L-KFHESSE_NOISEFREE( $L, w_t$ )
2:    $[y_t] \leftarrow L(w_t)$  // initial evaluation
3:    $Y \leftarrow y_t, S \leftarrow 0_N$  // store gradient and path
4:   all  $\Lambda \leftarrow 0$  and  $\sigma_B \leftarrow 1, \sigma_V \leftarrow 1, \sigma_W \leftarrow 1$  // init state
5:   while budget not used do
6:      $p_t \leftarrow -(m^B)^{-1} m^\nabla$ 
7:      $\alpha \leftarrow \text{LINESEARCH}(w, p_t)$ 
8:      $w_t \leftarrow w_t + \alpha p_t$  // update best guess
9:      $[y_t] \leftarrow L(w_t)$  // evaluate objective
10:    store  $Y \leftarrow [Y, y_t]$  // remove oldest if size > M
11:    store  $S \leftarrow [S, \alpha p_t]$  // remove oldest if size > M
12:
13:     $P^B \leftarrow \text{COMPUTE BASIS}(S, Y)$  // Gram-Schmidt,  $\mathcal{O}(N)$ 
14:    [all  $\Lambda^-, \sigma^-] \leftarrow \text{COMPUTE PREDICTIONS}(S, Y)$ 
15:    [all  $\Lambda, \sigma] \leftarrow \text{COMPUTE UPDATES}(\text{all } \Lambda^-, \sigma^-)$ 
16:    [ $\Lambda_W, \sigma_W] \leftarrow \text{PROJECT KRONECKER}(\text{all } \Lambda^-, \sigma^-, \Lambda, \sigma)$ 
17:  end while
18:  return  $w_t$ 
19: end function

```

Algorithm 7: Sketch of the noise-free KFHESS algorithm. The intensity matrix  $V$ , as well as  $W_0$  are set to the unity matrix.

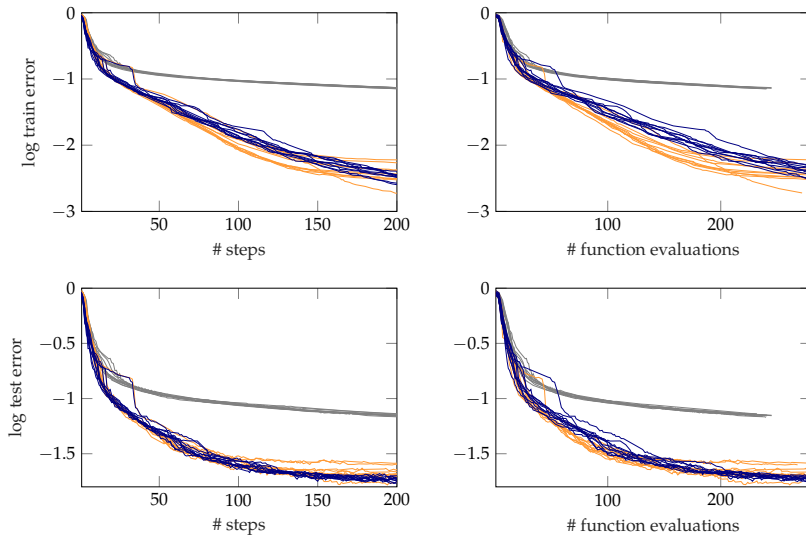


Figure 59: Multi-layer perceptron on MNIST. Top and bottom row show log train and test error respectively. Left and right column shows values versus  $t$  and # evaluations respectively. Colors are GD (—), L-BFGS (—), and L-KFHESSE (—). Same colors show the same setup for different seeds (10 each).

BFGS as ‘best possible’ performance on this problem. If we can gain performance relative to GD here, it is more likely to also perform better in stochastic setting later in comparison to SGD, where L-BFGS is not applicable anymore. The initial covariance  $W_0$  and the intensity matrix  $V$  are set to the identity, such that  $\sigma_V = \sigma_W = 1$ , which is similar to a PSB-style update. L-BFGS and L-KFHES both set the scalar contribution of the Hessian estimate to  $B_0 = m_0^B = (y_t^\top y_t)/(s_t^\top y_t)I$  in each step. A pseudo-code for the noise-free version of L-KFHES can be found in Algorithm 7. The first search direction is initialized with gradient descent, after that,  $m_t^B$  has non-trivial structure. We set the memory limit to the smallest possible value,  $M = 1$ , such that always one past gradient  $y$  and path segment  $s$  are kept in storage.

Figure 59 shows results. The top and bottom row show logarithmic test- and training set error respectively. In the left column, the values are plotted versus the number of iterations  $t$ , and in the right column versus the number of function evaluations (since all optimizers use a line search, the curves need not be equal). L-BFGS (—) and L-KFHES(—) both perform much better than GD (—). L-BFGS seems to be slightly more efficient than L-KFHES since it needs less function evaluations on average during a line search, but therefore L-KFHES reaches good test errors more consistently on this problem. The good performance of L-KFHES suggests that it might be worth looking into its noisy version in the future, a feature which is not natural to classic quasi-Newton methods.

## 9.5 Conclusion and Outlook

The following bullet-points summarize the main results of this chapter:

- We presented a novel probabilistic filtering framework on Hessians based on a continuous Gauss-Markov-model and local observations of Gaussian distributed gradients. We derived discrete Kalman filter prediction and update equations for the special case of Brownian motion. We then provided a way to use the updates in an iterative manner by projecting back onto the original structure of the distribution of the Kalman state. Finally, we provided approximate updates for very high-dimensional optimization problems by imposing a scalar-plus-low-rank structure on all relevant matrices.
- We analyzed correspondence of the mean estimator of the Hessian, arising from the filtering framework, to classic quasi-Newton methods, such as the members of the Dennis family, or Broyden’s method, for the special case of exact gradient evaluations. We showed that correspondence can be exact for the diffusion-free filter, while it is closely related but only approximate otherwise. There

is, however, exact correspondence between Gauss-Markov models that are not based on Wiener processes, for the Dennis member  $\text{rsb}$ , and possibly for others with restrictions on the the function  $L_{\mathcal{D}}(w)$  and the estimator  $m^B$ .

- Based on the findings above, we analyzed the implicit hyperparameters choices of the Dennis class members. We argued that most of them are only sensible in the case of exact gradient observations and should not be transferred blindly to the stochastic setting. We instead argue in favor of an `sr1`-type approach that resembles hyperparameter setting by empirical Bayes or maximum marginal likelihood estimation.
- We showed first experimental evidence that the derived filter can run on very high-dimensional problems, too, and that it might be worth investigating this line of research in the future.

Future research directions:

- Promising applications for `KFHES` might be lower to mid-sized dimensional noisy problems, where all relevant matrices can be stored densely. Since then, all past collected information will potentially be present in the Hessian estimator instead of a cut-off memory only, this setup should be more robust. Still, the question of success can not easily be answered, since it crucially depends on how many noisy gradient differences, i. e.,  $N$  numbers instead of  $\frac{1}{2}N(N+1)$  of interest, will be needed to estimate a sufficiently accurate  $m^B$ , while the optimizer moves in  $w$ -space. This depends on the dimensionality  $N$ , the mini-batch size  $|\mathcal{B}|$ , as well as the loss  $L_{\mathcal{D}}(w)$  and the local noise levels  $\Sigma(w)$ . But there might be classes of applications where this trade-off is in favor of the filter.
- It might be further interesting to ask if the approximate filter of Section 9.3 can be robustified for noisy settings. Currently, due to the limited memory, it loses its ability to naturally smooth over all past iterations. Nevertheless, evidence showed that modeling only a handful of off-diagonal Hessian-contributions is worthwhile in high-dimensional applications. It is not entirely clear how this could be approached but one might attempt to use a different low-rank basis, which does not lose all past information and propagate it to the next iteration.

## Epilogue





## Conclusions and Outlook

---

**T**HIS dissertation mainly focused on three current issues in stochastic, empirical risk minimization for machine learning applications. These are: i) improvement of generalization performance and over-fitting prevention, ii) increase in automation by removing manual tuning parameters, in particular learning rates, and iii) development of concepts for robust, stochastic search directions.

Chapter 6 introduced a novel early-stopping criterion that allows to fold-in the validation set into the training procedure. This enables an increase in generalization performance since all the available data can be used for training. Chapter 7 designed a novel probabilistic line search routine, so far applicable to stochastic gradient descent (SGD), which eliminates the learning rate as tuning parameter entirely, while retaining, or even increasing performance in the test accuracy. The last two Chapters 8 and 9 elaborated on a novel probabilistic framework for noise-robust first- and second-order optimization. By drawing connections, the framework gives valuable insight into existing classic optimizers, and also opens up a class of potentially viable probabilistic optimizers with increased expressiveness, automation, and robustness to noise. First prototypes were presented in the form of KFGRAD and KFHESS.

All of the derived work is based on the argument that conditional distributions of full- as well as stochastic mini-batch gradients and losses can locally be approximated by Gaussians. These distributions have analytic form, and their parameters, in particular variances, can be estimated with justifiable overhead at run-time. They can also be interpreted as likelihoods, such that stochastic evaluations of losses and gradients can readily be incorporated into noise-informed decision making at run-time of the optimizer, by means of well-known inference techniques of machine learning and statistics. In particular, we used GP-regression, Kalman filtering, Bayesian optimization, and others, rendered to the specific needs of an optimizer, such as memory requirements, hardware limits, and constraints on the computational complexity of the desired algorithm. The result are self-contained, fast and flexible numerical methods that—on the lowest level of computation—are learning machines themselves, whose own internal model-parameters are either set by smart design or adapted by cheap/approximate empirical Bayes.

The benefits of a single language to cover all these topics—probability theory—in this regard are mainly: The possibility to express different aspects of the same problem in a single framework. Current stand-alone concepts and algorithms that solve sub-tasks can in the future be chained together and communicate with each other. For instance, as outline in Chapter 5, the posterior marginal distribution of the first-order Kalman filter might be used by the probabilistic line search. Thus, instead of a large toolbox with many standalone implementations that a user has to choose from, a unified framework will potentially be able to automatically pick a sub-algorithm depending on the task, adapt its hyperparameters, and interpolate between, or switch optimization routines during one run. There is still a lot of work to do towards that end, but the works presented here are first steps towards that goal.

The models used throughout this thesis were fully Gaussian, meaning that all relevant variables are affine maps of each other, and observation noise is additive and Gaussian. Inference in fully Gaussian systems is analytic, tractable and only needs access to solvers of classic linear algebra, i. e., solutions to matrix-matrix products and linear systems. We have seen that Gaussian likelihoods for losses and gradients are often approximately correct; alternatively, Gaussian approximations can be seen as casting a difficult inference task onto one, or many successive linearized ones, in the sense that the latter can be solved using only linear operations (see also [58]). Since numerical methods, like optimizers, should ideally not call yet another non-trivial numerical method, it can be argued that Gaussians thus occur somewhat naturally, as a minimalistic model that includes uncertainty of computations, if one imposes linear constraints on the class of allowed computations.

Further benefits of a shared probabilistic framework are increased interpretability of parameters or hyper-parameters of the optimization routine, as well as valuable insight into classic methods. An example of this was presented in Chapter 8 where the ad-hoc exponential smoothing factors of momentum-SGD resemble Kalman-gains of a Gauss-Markov-model. Or Chapter 9, which presented an argument that a BFGS-type update is not what should be aimed for in stochastic optimization.

The outlooks and future research directions were already discussed at the end of each chapter. On a more general, practical note, future research should also go towards: i) The creation of software and toolboxes that are accessible and easy-to-use. ii) Linking standalone algorithms to each other, such that they can share resources and computations. On a more theoretical note, it would be beneficial to iii) Improve the theoretical analysis of the presented algorithms. This is more or less crucial depending on the task. But, for instance, the EB-criterion

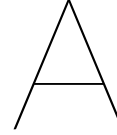
[58] Hennig, Osborne, and Girolami, “Probabilistic numerics and uncertainty in computations,” 2015

for early stopping, presented in Chapter 8 might benefit greatly from it. An interesting research direction that might also lead to algorithmic considerations, is the iv) Trade-off between computational cost per step, and the gains in the overall performance. This is closely linked to the size  $|\mathcal{B}|$  of the mini-batch in stochastic empirical risk minimization, and for sure not easy so solve. Success in this research direction might help the optimizer to automatically allocate its resources, increase or decrease  $|\mathcal{B}|$  as needed, and also choose among different probabilistic sub-routines, depending on, whether it is beneficial to use one or the other. The general notion of using readily available and well-studied techniques of probabilistic inference, for naturally or artificially occurring stochasticity in optimization, is a promising research direction in general. The notion of uncertainty arising from lack of knowledge, rather than physical randomness must be embraced to that end.



## Appendix





## Kronecker Algebra

---

**T**HIS chapter introduces the Kronecker product and its algebra which was used throughout this thesis. A significant part of this section is taken from [85] and [136]; proofs were added, some results (especially on the symmetric Kronecker product) were extended or stated more explicitly (e.g. Eq. 180 is not given in [85]). The *Einstein summation convention* [35] is used throughout for ease of notation; this means sum-symbols are being dropped (unless emphasis is necessary), and sums occur across same indices.

[85] Loan, "The ubiquitous Kronecker product," 2000

[136] Van Loan and Pitsianis, "Approximation with Kronecker Products," 1993

[35] Einstein, "Die Grundlage der allgemeinen Relativitätstheorie," 1916

### A.1 Kronecker Products

Let  $A \in \mathbb{R}^{N_1 \times N_2}$  and  $B \in \mathbb{R}^{K_1 \times K_2}$  be matrices or vectors. Then the Kronecker product of  $A$  and  $B$  is in  $\mathbb{R}^{N_1 K_1 \times N_2 K_2}$  and defined as:

$$(A \otimes B)_{(ij),(kl)} = A_{ik} B_{jl}, \quad i = 1 \dots N_1, k = 1 \dots N_2, \\ j = 1 \dots K_1, l = 1 \dots K_2 \quad (159)$$

Example for  $N_1 = N_2 = 2$  and  $K_1 = 3, K_2 = 2$ :

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \otimes \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix} = \left[ \begin{array}{cc|cc} A_{11}B_{11} & A_{11}B_{12} & A_{12}B_{11} & A_{12}B_{12} \\ A_{11}B_{21} & A_{11}B_{22} & A_{12}B_{21} & A_{12}B_{22} \\ A_{11}B_{31} & A_{11}B_{32} & A_{12}B_{31} & A_{12}B_{32} \\ \hline A_{21}B_{11} & A_{21}B_{12} & A_{22}B_{11} & A_{22}B_{12} \\ A_{21}B_{21} & A_{21}B_{22} & A_{22}B_{21} & A_{22}B_{22} \\ A_{21}B_{31} & A_{21}B_{32} & A_{22}B_{31} & A_{22}B_{32} \end{array} \right] \quad (160)$$

The Kronecker product applied to a vector  $\vec{X}$  yields

$$(A \otimes B) \vec{X} = \overrightarrow{AXB^\top} \quad (161)$$

where the operation  $\vec{\phantom{x}}$  stacks the rows of a matrix  $X \in \mathbb{R}^{N_2 \times K_2}$  into a column vector  $\vec{X} \in \mathbb{R}^{N_2 K_2 \times 1}$ . The operation on the right side of Eq. 161 costs  $\mathcal{O}(N_2 K_1 (K_2 + N_1))$  or  $\mathcal{O}(N_1 K_2 (N_2 + K_1))$  while the left side costs  $\mathcal{O}(N_1 K_1 N_2 K_2)$ . The Kronecker product has some nice algebraic

properties which roughly resemble the ones of rank-one matrices. For matrices  $A$  and  $B$  of appropriate size it is:

$$\text{TRANSPOSE} \quad (A \otimes B)^\top = A^\top \otimes B^\top \quad (162a)$$

$$\text{INVERSE} \quad (A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \quad (162b)$$

$$\text{FACTORIZING} \quad (A \otimes B)(C \otimes D) = AC \otimes BD \quad (162c)$$

$$\text{DISTRIBUTIVE LEFT} \quad (A \otimes B) + (A \otimes C) = A \otimes (B + C) \quad (162d)$$

$$\text{DISTRIBUTIVE RIGHT} \quad (A \otimes B) + (C \otimes B) = (A + C) \otimes B \quad (162e)$$

$$\text{ASSOCIATIVE} \quad (A \otimes B) \otimes C = A \otimes (B \otimes C) \quad (162f)$$

$$\text{TRACE} \quad \text{tr}[(A \otimes B)] = \text{tr}[A] \text{tr}[B] \quad (162g)$$

Also the *Frobenius norm* of a matrix  $A$  can be re-written using the Kronecker product:

$$\|A\|_F^2 = \vec{A}^\top (I \otimes I) \vec{A} \quad (163)$$

**Proof of Eq. 162a:** follows directly from definition (Eq. 159):

$$(A \otimes B)_{(ij)(kl)}^\top = (A \otimes B)_{(kl)(ij)} = A_{ki} B_{lj} = A_{ik}^\top B_{jl}^\top = (A^\top \otimes B^\top)_{(ij)(kl)}. \quad \blacksquare$$

**Proof of Eq. 162c:** Sizes of matrices need to fit the operations, in other words  $A \in \mathbb{R}^{N_1 \times N_2}$ ,  $C \in \mathbb{R}^{N_2 \times N_3}$ ,  $B \in \mathbb{R}^{K_1 \times K_2}$ , and  $D \in \mathbb{R}^{K_2 \times K_3}$ , then

$$\begin{aligned} [(A \otimes B)(C \otimes D)]_{(ij),(kl)} &= (A \otimes B)_{(ij)(mn)} (C \otimes D)_{(mn),(kl)} = A_{im} B_{jn} C_{mk} D_{nl} \\ &= (AC)_{ik} (BD)_{jl} = (AC \otimes BD)_{(ij),(kl)}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 162b:** uses Eq. 162c. It only holds for matrices where an inverse exists. Then

$$(A \otimes B)(A^{-1} \otimes B^{-1}) = AA^{-1} \otimes BB^{-1} = I_{N \times N} \otimes I_{K \times K} = I_{NK \times NK}. \quad \blacksquare$$

**Proof of Eq. 162d and Eq. 162e:** follows from definition and rearranging terms:

$$[(A \otimes B) + (A \otimes C)]_{(ij),(kl)} = A_{ik} B_{jl} + A_{ik} C_{jl} = A_{ik} (B + C)_{jl} = [A \otimes (B + C)]_{(ij),(kl)}. \quad \blacksquare$$

**Proof of Eq. 162f:** follows from definition and rearranging terms:

$$\begin{aligned} [(A \otimes B) \otimes C]_{((ij),m),(kl),n} &= (A \otimes B)_{(ij),(kl)} C_{mn} = A_{ik} B_{jl} C_{mn} \\ &= A_{ik} (B \otimes C)_{(jm),(ln)} = [A \otimes (B \otimes C)]_{(i,(jm)),(k,(ln))}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 162g:** follows from definition and definition of trace:

$$\text{tr}[(A \otimes B)] = (A \otimes B)_{(ij),(ij)} = A_{ii} B_{jj} = \text{tr}[A] \text{tr}[B]. \quad \blacksquare$$

**Proof of Eq. 163:** follows from definition and definition of the Frobenius norm:



$$\vec{A}^\top(I \otimes I)\vec{A} = A_{(ij)}(I \otimes I)_{(ij),(kl)}A_{(kl)} = A_{ij}\delta_{ik}\delta_{jl}A_{kl} = \sum_{i,j} A_{ij}^2 = \|A\|_F^2. \quad \blacksquare$$

### A.1.1 Closest Kronecker Product under Frobenius Norm

Find  $A^* \in \mathbb{R}^{N_1 \times N_2}$  and  $B^* \in \mathbb{R}^{K_1 \times K_2}$  such that the Frobenius norm

$$\|C - A \otimes B\|_F^2 \quad \text{with } C \in \mathbb{R}^{N_1 K_1 \times N_2 K_2} \quad (164)$$

is minimized. Equivalently:

$$A^*, B^* = \arg \min_{A, B} \|C - A \otimes B\|_F^2. \quad (165)$$

There exists a fixed permutation  $\mathcal{R}$  such that the Kronecker product can be written as an outer product of the vectorized matrices  $A$  and  $B$ :

$$\mathcal{R}(A \otimes B) = \vec{A}\vec{B}^\top \quad \text{and} \quad \mathcal{R}(C) = \begin{pmatrix} \text{vec}(c^{11})^\top \\ \vdots \\ \text{vec}(c^{N_1 N_2})^\top \end{pmatrix}. \quad (166)$$

$\mathcal{R}$  vectorizes and then transposes each *block*  $c^{\alpha\beta}$  of the resulting matrix  $C = A \otimes B$  and stacks them row by row (blocks as shown in Eq. 160 by horizontal and vertical lines, and  $\alpha = 1, \dots, N_1$ ,  $\beta = 1, \dots, N_2$ ). The resulting matrix is in  $\mathbb{R}^{N_1 N_2 \times K_1 K_2}$ . The Frobenius norm in Eq. 165 stays the same since elements in the sum are only reordered. Eq. 165 thus reduced to solving a rank-one approximation problem of the form:

$$\vec{A}^*, \vec{B}^* = \arg \min_{A, B} \|\mathcal{R}(C) - \vec{A}\vec{B}^\top\|_F^2. \quad (167)$$

Let  $\mathcal{R}(C) = U\Sigma V^\top$  be the singular value decomposition of  $\mathcal{R}(C)$ . Then this can be solved by computing the leading singular value  $\sigma_1$  and corresponding left and right singular vectors  $u_1$  and  $v_1$  of  $\mathcal{R}(C)$ . Then the solution up to a scalar  $z \neq 0$  is:

$$\vec{A}^* = z\sqrt{\sigma_1}u_1 \quad \text{and} \quad \vec{B}^* = z^{-1}\sqrt{\sigma_1}v_1. \quad (168)$$

For order- $r$  approximations we simply get

$$\vec{A}^* = z \sum_{i=1}^r \sqrt{\sigma_i} u_i \quad \text{and} \quad \vec{B}^* = z^{-1} \sum_{i=1}^r \sqrt{\sigma_i} v_i. \quad (169)$$

## A.1.2 Approximating Sums of Kronecker Products

Find  $A^* \in \mathbb{R}^{N \times N}$  and  $B^* \in \mathbb{R}^{K \times K}$  such that the Frobenius norm

$$\|C - A \otimes B\|_F^2 \quad \text{with} \quad C = \sum_{\alpha=1}^p R^\alpha \otimes S^\alpha \quad (170)$$

is minimized. Equivalently:

$$A^*, B^* = \arg \min_{A, B} \left\| \sum_{\alpha=1}^p (R_\alpha \otimes S_\alpha) - A \otimes B \right\|_F^2. \quad (171)$$

$R^\alpha$  and  $S^\alpha$  have the same shape as  $A$  and  $B$  respectively. This is equivalent to finding the rank-one approximation of:

$$\left\| RS^\top - \vec{A} \vec{B}^\top \right\|_F^2 \quad \text{with} \quad R \in \mathbb{R}^{N^2 \times p}, S \in \mathbb{R}^{K^2 \times p}, \vec{A} \in \mathbb{R}^{N^2 \times 1}, \vec{B} \in \mathbb{R}^{K^2 \times 1}. \quad (172)$$

Denote the singular value decomposition (SVD) and compact SVD of  $C$  as:

$$\begin{aligned} C &= \tilde{U} \Sigma \tilde{V}^\top \quad \text{with} \quad \tilde{U} \in \mathbb{R}^{N^2 \times N^2}, \Sigma \in \mathbb{R}^{N^2 \times K^2}, \tilde{V} \in \mathbb{R}^{K^2 \times K^2} \\ C &= \tilde{U}_+ \Sigma_+ \tilde{V}_+^\top \quad \text{with} \quad \tilde{U}_+ \in \mathbb{R}^{N^2 \times p}, \Sigma_+ \in \mathbb{R}^{p \times p}, \tilde{V}_+ \in \mathbb{R}^{K^2 \times p}. \end{aligned} \quad (173)$$

$\Sigma_+$  is of the form  $\text{diag}(\sigma_1 \dots \sigma_p)$  where  $\sigma_i$  are the singular values of  $C$ . Assuming the  $\sigma_i$  are sorted from largest to smallest the solution to the rank-one approximation problem of Eq. 172 is

$$A = \sqrt{\sigma_1} \tilde{u}_1 \quad \text{and} \quad B = \sqrt{\sigma_1} \tilde{v}_1 \quad (174)$$

where  $\tilde{u}_1$  and  $\tilde{v}_1$  are the corresponding left and right singular vectors to  $\sigma_1$ . They are also the eigenvectors to the following matrices

$$\begin{aligned} [CC^\top] \tilde{u}_i &= [RS^\top SR^\top] \tilde{u}_i = \sigma_i^2 \tilde{u}_i \\ [C^\top C] \tilde{v}_i &= [SR^\top RS^\top] \tilde{v}_i = \sigma_i^2 \tilde{v}_i. \end{aligned} \quad (175)$$

This can be seen by plugging Eq. 173 into Eq. 175 and using the orthonormality of  $\tilde{U}$  and  $\tilde{V}$ . This means that if we find the eigendecomposition of  $C^\top C$  and  $CC^\top$  we can find the  $\sigma_1$ ,  $\tilde{u}_1$ , and  $\tilde{v}_1$ . Define:

$$S^\top S =: U_S D_S U_S^\top \quad \in \mathbb{R}^{p \times p} \quad \text{eigen-decomposition} \quad (176a)$$

$$R^\top R =: U_R D_R U_R^\top \quad \in \mathbb{R}^{p \times p} \quad \text{eigen-decomposition} \quad (176b)$$

$$\Xi := D_S^{1/2} U_S^\top R^\top R U_S D_S^{1/2} \quad (176c)$$

$$\Xi =: U_{SR} D_{SR} U_{SR}^\top \quad \in \mathbb{R}^{p \times p} \quad \text{eigen-decomposition} \quad (176d)$$

$$\tilde{S} := S U_S D_S^{-1/2} \quad \text{with} \quad \tilde{S}^\top \tilde{S} = I_{p \times p} \quad (176e)$$

$$\hat{S} := \tilde{S} U_{SR} \quad \text{with} \quad \hat{S}^\top \hat{S} = I_{p \times p} \quad (176f)$$

$$I_{p \times p} = U_S D_S^{-1/2} D_S^{1/2} U_S^\top. \quad (176g)$$

Note that  $\Xi$  can be written as  $\Xi = \zeta\zeta^\top$ . Plugging Eq. 176 into Eq. 175 yields

$$\begin{aligned}
 C^\top C &= SR^\top RS^\top \\
 &= S(U_S D_S^{-1/2} D_S^{1/2} U_S^\top)(R^\top R)(U_S D_S^{1/2} D_S^{-1/2} U_S^\top) S^\top \\
 &= \tilde{S}(D_S^{1/2} U_S^\top R^\top R U_S D_S^{1/2}) \tilde{S}^\top = \tilde{S}(U_{SR} D_{SR} U_{SR}^\top) \tilde{S}^\top \\
 &= \hat{S} D_{SR} \hat{S}^\top.
 \end{aligned} \tag{177}$$

Analogously:

$$\begin{aligned}
 CC^\top &= RS^\top SR^\top \\
 &= R(U_R D_R^{-1/2} D_R^{1/2} U_R^\top)(S^\top S)(U_R D_R^{1/2} D_R^{-1/2} U_R^\top) R^\top \\
 &= \tilde{R}(D_R^{1/2} U_R^\top S^\top S U_R D_R^{1/2}) \tilde{R}^\top = \tilde{R}(U_{RS} D_{RS} U_{RS}^\top) \tilde{R}^\top \\
 &= \hat{R} D_{RS} \hat{R}^\top.
 \end{aligned} \tag{178}$$

The computation of  $A$  and  $B$ , requires  $\hat{R}$ ,  $\hat{S}$ ,  $D_{SR}$ , and  $D_{RS}$ . The following pseudo-code (matlab notation) illustrates the necessary algorithm:

**Require:**  $S, R$

```

RR ← R⊤R // compute p × p matrices
SS ← S⊤S
US, DS ← eigs(SS, p) // sorted eigen-decomposition of RR.
UR, DR ← eigs(RR, p) // sorted eigen-decomposition of SS.

ΞS ← DS1/2US⊤RRUSDS1/2
USR, DSR ← eigs(ΞS, p) // sorted eigen-decomposition of ΞS.
S̃ ← SUSDS-1/2USR // right singular vectors

ΞR ← DR1/2UR⊤SSURDR1/2
URS, DRS ← eigs(ΞR, p) // sorted eigen-decomposition of ΞR.
R̃ ← RURDR-1/2URS // left singular vectors

σ ← diag(DSR1/2) // singular values
Ri* ← R̃(:, i)σ(i)1/2 for i = 1, …, papprox // reshaped as N × N
Si* ← S̃(:, i)σ(i)1/2 for i = 1, …, papprox // reshaped as K × K
    
```

For the special (symmetric) case of  $R = S$ , some equations simplify:

$$\begin{aligned}
 C^\top C &= SS^\top SS^\top = S(U_S D_S U_S^\top) S^\top \\
 &= S(U_S D_S^{-1/2} D_S^{1/2} U_S^\top)(U_S D_S U_S^\top)(U_S D_S^{1/2} D_S^{-1/2} U_S^\top) S^\top \\
 &= (S U_S D_S^{-1/2})(D_S^{1/2} D_S D_S^{1/2})(D_S^{-1/2} U_S^\top S^\top) \\
 &= \tilde{S} D_S^2 \tilde{S}^\top.
 \end{aligned} \tag{179}$$

In this case only the SVD of  $S^\top S$  is needed. The corresponding simplified pseudo-code (matlab notation) is:

**Require: S**

```

SS ← STS // compute p × p matrix
US, DS ← eigs(SS, p) // sorted eigen-decomposition of SS.
 $\bar{S} \leftarrow SU_S D_S^{-1/2}$  // singular vectors
 $\sigma \leftarrow \text{diag}(D_S)$  // singular values
Si* ←  $\bar{S}(:, i)\sigma(i)^{1/2}$  for i = 1, …, papprox // reshaped as K × K

```

## A.1.3 Coefficients of Sum of Kronecker Approximation

Ideally find a solution for  $A^*$  and  $B^*$  of the form  $A_i^* = \sum_{\alpha=1}^p \gamma_{\alpha} R_{\alpha}$  and  $B_i^* = \sum_{\alpha=1}^p \gamma_{\alpha} S_{\alpha}$  which are linear combinations of the input with weights  $\gamma_{\alpha}, \alpha = 1 \dots p$ . Like this beneficial matrix forms are preserved, for example if  $R_{\alpha}$  and  $S_{\alpha}$  are of the form 'diagonal-plus-low-rank'. Some algebra shows that the  $i^{\text{th}}$  approximation is:

$$\begin{aligned}
A_i^* &= \sum_{\alpha=1}^p \left[ U_R D_R^{-1/2} U_{RS} \right]_{\alpha,i} \sigma_i \cdot R_{\alpha} \\
B_i^* &= \sum_{\alpha=1}^p \left[ U_S D_S^{-1/2} U_{SR} \right]_{\alpha,i} \sigma_i \cdot S_{\alpha} \\
A^* &= \sum_{\alpha=1}^p \left[ U_R D_R^{-1/2} U_{RS} \right]_{\alpha,1} \sigma_1 \cdot R_{\alpha} \quad \text{leading component} \\
B^* &= \sum_{\alpha=1}^p \left[ U_S D_S^{-1/2} U_{SR} \right]_{\alpha,1} \sigma_1 \cdot S_{\alpha}. \quad \text{leading component}
\end{aligned} \tag{180}$$

Again for the symmetric case, where  $R = S$  we get:

$$\begin{aligned}
A_i^* &= \sum_{\alpha=1}^p [U_S]_{\alpha,i} \cdot R_{\alpha} \\
A^* &= \sum_{\alpha=1}^p [U_S]_{\alpha,1} \cdot R_{\alpha}. \quad \text{leading component}
\end{aligned} \tag{181}$$

## A.1.4 Decomposition into Symmetric and Anti-Symmetric Part

Let  $A$  be in  $\mathbb{R}^{N \times N}$ . The Kronecker product  $A \otimes A$  can be decomposed into a symmetric  $A \otimes A$  and a anti-symmetric  $A \oplus A$  part leading to the definitions of the symmetric and anti-symmetric Kronecker product. If  $A \otimes A$  has full rank of  $N^2$ , then the symmetric Kronecker product and anti-symmetric Kronecker product span the  $1/2N(N+1)$  and  $1/2N(N-1)$  dimensional subspaces respectively. They can be obtained by applying the symmetrization and anti-symmetrization operators  $\Gamma$  and  $\Delta$ . Define the operators ' $\otimes$ ' and  $\oplus$  by:

$$A \otimes B := \Gamma(A \otimes B) \Gamma^T \quad \text{and} \quad A \oplus B := \Delta(A \otimes B) \Delta^T \tag{182}$$

for  $B$  also in  $\mathbb{R}^{N \times N}$ . The product can be decomposed into:

$$\begin{aligned}
 A \otimes B &= (\Gamma + \Delta)(A \otimes B)(\Gamma + \Delta)^\top \\
 &= \Gamma(A \otimes B)\Gamma^\top + \Delta(A \otimes B)\Delta^\top + \Delta(A \otimes B)\Gamma^\top + \Gamma(A \otimes B)\Delta^\top \\
 &= A \otimes B + A \otimes B + \Delta(A \otimes B)\Gamma^\top + \Gamma(A \otimes B)\Delta^\top.
 \end{aligned} \tag{183}$$

As mentioned above, if  $A = B$ , then  $\Delta(A \otimes A)\Gamma^\top = \Gamma(A \otimes A)\Delta^\top = 0$  and the product decomposes into symmetric and anti-symmetric part:

$$A \otimes A = A \otimes A + A \otimes A \tag{184}$$

This also implies that:

$$I_{N \times N} \otimes I_{N \times N} = \Gamma(I_{N \times N} \otimes I_{N \times N})\Gamma^\top = \Gamma \tag{185a}$$

$$I_{N \times N} \otimes I_{N \times N} = \Delta(I_{N \times N} \otimes I_{N \times N})\Delta^\top = \Delta \tag{185b}$$

$$I_{N \times N} \otimes I_{N \times N} + I_{N \times N} \otimes I_{N \times N} = (I_{N \times N} \otimes I_{N \times N}) = \Gamma + \Delta = I_{N^2 \times N^2}. \tag{185c}$$

The element-wise definitions of  $\Gamma$ ,  $\Delta$ ,  $'\otimes'$  and  $'\otimes'$  and proof for the decomposition can be found in Appendix A.2 and A.3 respectively.

## A.2 Symmetric Kronecker Products

Let  $A \in \mathbb{R}^{N \times N}$  and  $B \in \mathbb{R}^{N \times N}$  be square matrices of same size. Then the symmetric Kronecker product of  $A$  and  $B$  is in  $\mathbb{R}^{N^2 \times N^2}$  and defined as  $(A \otimes B) := \Gamma(A \otimes B)\Gamma^\top$  with projection operator  $\Gamma$ :

$$\begin{aligned}
 \Gamma A &= \frac{1}{2}(A + A^\top), \quad \Gamma_{(ij),(kl)} = \frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{kj}) \\
 (A \otimes B)_{(ij),(kl)} &= \frac{1}{4}(A_{ik}B_{jl} + A_{il}B_{jk} + A_{jk}B_{il} + A_{jl}B_{ik}) \\
 (A \otimes B)\vec{X} &= \frac{1}{4}(\overline{AXB^\top + AX^\top B^\top + BX^\top A^\top + BXA^\top})
 \end{aligned} \tag{186}$$

where  $\delta_{ij} = 1$  if  $i = j$  and zero otherwise. The symmetric Kronecker product symmetrizes  $X$ . If  $A = B$  then Eq. 186 simplifies to

$$\begin{aligned}
 (A \otimes A)_{(ij),(kl)} &= \frac{1}{2}(A_{ik}A_{jl} + A_{jk}A_{il}) \\
 (A \otimes A)\vec{X} &= \frac{1}{2}(\overline{AXA^\top + AX^\top A^\top}).
 \end{aligned} \tag{187}$$

Similar to the Kronecker product, its symmetric version has some nice algebraic properties:

$$\text{TRANSPOSE} \quad (A \otimes B)^\top = A^\top \otimes B^\top \quad (188a)$$

$$\text{FACTORIZING} \quad (A \otimes A)(C \otimes C) = AC \otimes AC \quad \text{but} \quad (A \otimes B)(C \otimes D) \neq (AC \otimes BD) \quad (188b)$$

$$(A \otimes B)(C \otimes D) = \frac{1}{2} [AC \otimes BD + AD \otimes BC] \quad (188c)$$

$$\text{INVERSE} \quad (A \otimes A)^{-1} = (A^{-1} \otimes A^{-1}) \quad \text{but} \quad (A \otimes B)^{-1} \neq (A^{-1} \otimes B^{-1}) \quad (188d)$$

$$\text{COMMUTATIVE} \quad A \otimes B = B \otimes A \quad \text{but} \quad A \otimes B \neq B \otimes A \quad (188e)$$

$$\text{TRACE} \quad \text{tr} [A \otimes B] = \frac{1}{2} (\text{tr} [A] \text{tr} [B] + \text{tr} [AB]). \quad (188f)$$

**Proof of Eq. 188a:** follows from definition and rearranging terms:

$$\begin{aligned} (A \otimes B)_{(ij),(kl)}^\top &= (A \otimes B)_{(kl),(ij)} = \frac{1}{4} (A_{ki} B_{lj} + A_{kj} B_{li} + A_{li} B_{kj} + A_{lj} B_{ki}) \\ &= \frac{1}{4} (A_{ik}^\top B_{jl}^\top + A_{jk}^\top B_{il}^\top + A_{il}^\top B_{jk}^\top + A_{jl}^\top B_{ik}^\top) = (A^\top \otimes B^\top)_{(ij),(kl)}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 188c:** follows from definition and rearranging terms:

$$\begin{aligned} [(A \otimes B)(C \otimes D)]_{(ij),(kl)} &= (A \otimes B)_{(ij),(mn)} (C \otimes D)_{(mn),(kl)} \\ &= \frac{1}{4} [A_{im} B_{jn} + A_{jm} B_{in} + A_{in} B_{jm} + A_{jn} B_{im}] \frac{1}{4} [C_{mk} D_{nl} + C_{nk} D_{ml} + C_{ml} D_{nk} + C_{nl} D_{mk}] \\ &= 2 \frac{1}{4} \frac{1}{4} [(AC)_{ik} (BD)_{jl} + (AC)_{il} (BD)_{jk} + (AC)_{jk} (BD)_{il} + (AC)_{jl} (BD)_{ik} \\ &\quad + (AD)_{ik} (BC)_{jl} + (AD)_{il} (BC)_{jk} + (AD)_{jk} (BC)_{il} + (AD)_{jl} (BC)_{ik}] \\ &= \frac{1}{2} [AC \otimes BD + AD \otimes BC]_{(ij),(kl)}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 188b:** follows directly from Eq. 188c for  $A = B$  and  $C = D$ .  $\blacksquare$

**Proof of Eq. 188d:** follows directly from Eq. 188c. Note that in general  $(A \otimes B)^{-1} \neq A^{-1} \otimes B^{-1}$ .

$$(A \otimes A)(A^{-1} \otimes A^{-1}) = AA^{-1} \otimes AA^{-1} = I_{N \times N} \otimes I_{N \times N} = \Gamma \quad \blacksquare$$

**Proof of Eq. 188e:** follows directly from definition and reordering.

Note that  $A \otimes B \neq B \otimes A$ :

$$\begin{aligned} (A \otimes B)_{(ij),(kl)} &= \frac{1}{4} (A_{ik} B_{jl} + A_{il} B_{jk} + A_{jk} B_{il} + A_{jl} B_{ik}) \\ &= \frac{1}{4} (B_{ik} A_{jl} + B_{il} A_{jk} + B_{jk} A_{il} + B_{jl} A_{ik}) = (B \otimes A)_{(ij),(kl)}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 188f:** follows directly from definition and the definition of trace:

$$\begin{aligned} \text{tr} [A \otimes B] &= \sum_{i,j} (A \otimes B)_{(ij),(ij)} = \frac{1}{4} \sum_{i,j} (A_{ii} B_{jj} + A_{ij} B_{ji} + A_{ji} B_{ij} + A_{jj} B_{ii}) \\ &= \frac{1}{2} (\text{tr} [A] \text{tr} [B] + \text{tr} [AB]). \quad \blacksquare \end{aligned}$$

### A.2.1 Closest Symmetric Kronecker Product under Frobenius Norm

Find the nearest symmetric Kronecker product to a square matrix  $C$  of appropriate size under the Frobenius norm similar to Eq. 165:

$$A_{\text{sym}}^* B_{\text{sym}}^* = \arg \min_{A,B} \|C - A \otimes B\|_F^2. \quad (189)$$

The symmetric Kronecker product can be obtained from the Kronecker product by applying a linear operator  $\mathcal{T}$  to its vectorized version;  $\mathcal{T}$  can be read off from Eq. 186:

$$\begin{aligned} \mathcal{T}_{(ijkl),(abcd)}(A \otimes B)_{(abcd)} &= \Gamma_{(ij),(ab)}(A \otimes B)_{(ab),(cd)} \Gamma_{(kl),(cd)} = (A \otimes B)_{(ij),(kl)} \\ \mathcal{T}_{(ijkl),(abcd)} &= \Gamma_{(ij),(ab)} \Gamma_{(kl),(cd)} = \frac{1}{2}(\delta_{ia}\delta_{jb} + \delta_{ib}\delta_{aj}) \frac{1}{2}(\delta_{kc}\delta_{ld} + \delta_{kd}\delta_{cl}) \\ &= \frac{1}{4}(\delta_{ia}\delta_{jb}\delta_{kc}\delta_{ld} + \delta_{ib}\delta_{aj}\delta_{kc}\delta_{ld} + \delta_{ia}\delta_{jb}\delta_{kd}\delta_{cl} + \delta_{ib}\delta_{aj}\delta_{kd}\delta_{cl}). \end{aligned} \quad (190)$$

With slight abuse of notation Eq. 189 can be rewritten as:

$$A_{\text{sym}}^* B_{\text{sym}}^* = \arg \min_{A,B} \|\mathcal{T}[C - A \otimes B]\|_F^2. \quad (191)$$

Since Eq. 191 is a quadratic form, the solution of Eq. 191 is the linear projection  $\mathcal{T}$  of the solution of Eq. 170. Therefore we can simply solve the corresponding nearest Kronecker product problem as described in Appendix A.1.1 and project the solution through  $\mathcal{T}$ . In other words the nearest symmetric Kronecker product is:

$$\begin{aligned} A_{\text{sym}}^* &= A^* && \text{with } A^* \text{ from Section A.1, Eq. 168} \\ B_{\text{sym}}^* &= B^* && \text{with } B^* \text{ from Section A.1, Eq. 168} \\ A^* \otimes B^* &= \mathcal{T}[A^* \otimes B^*] = A_{\text{sym}}^* \otimes B_{\text{sym}}^*. \end{aligned} \quad (192)$$

The above is also valid for higher order approximations as described in Appendix A.1.1.

### A.2.2 Approximating Sums of Symmetric Kronecker Products

Find the nearest symmetric Kronecker product to a sum of symmetric Kronecker products under the Frobenius norm similar to Eq. 171:

$$A_{\text{sym}}^* B_{\text{sym}}^* = \arg \min_{A,B} \left\| \sum_{\alpha=1}^p (R_\alpha \otimes S_\alpha) - A \otimes B \right\|_F^2. \quad (193)$$

Using the same operator  $\mathcal{T}$  as defined in Eq. 190, Eq. 193 can be rewritten as:

$$A_{\text{sym}}^* B_{\text{sym}}^* = \arg \min_{A,B} \left\| \mathcal{T} \left[ \sum_{\alpha=1}^p (R_\alpha \otimes S_\alpha) - A \otimes B \right] \right\|_F^2. \quad (194)$$

By the same argument as in Section B.1 the solution to Eq. 193 is:

$$\begin{aligned} A_{\text{sym}}^* &= A^* && \text{with } A^* \text{ from Section A.3, Eq. 180} \\ B_{\text{sym}}^* &= B^* && \text{with } B^* \text{ from Section A.3, Eq. 180} \\ A^* \otimes B^* &= \mathcal{T} [A^* \otimes B^*] = A_{\text{sym}}^* \otimes B_{\text{sym}}^*. \end{aligned} \quad (195)$$

The above is also valid for higher order approximations as described in Appendix A.1.2.

### A.2.3 Approximate 2x2 Symmetric Kronecker Product with reduced 2x2 Symmetric Kronecker Product

Let  $A, B, C$  be in  $\mathbb{R}^{2 \times 2}$ . Find the nearest symmetric Kronecker product  $C \otimes C$  to a general symmetric Kronecker product  $A \otimes B$  under the Frobenius norm:

$$C_{\text{sym}}^* = \arg \min_C \|A \otimes B - C \otimes C\|_F^2. \quad (196)$$

Note that using Eq. 188e we can write

$$C_{\text{sym}}^* = \arg \min_C \left\| \frac{1}{2} A \otimes B + \frac{1}{2} B \otimes A - C \otimes C \right\|_F^2. \quad (197)$$

Using the same notation for  $R$  and  $S$  as in Appendix A.1.2 and A.2.2 yields

$$S = \begin{pmatrix} \overrightarrow{\sqrt{0.5}A} & \overrightarrow{\sqrt{0.5}B} \end{pmatrix}, \quad R = \begin{pmatrix} \overrightarrow{\sqrt{0.5}B} & \overrightarrow{\sqrt{0.5}A} \end{pmatrix}. \quad (198)$$

We want to show that in this setting  $\hat{R} = \hat{S}$ . Define  $a := \overrightarrow{\sqrt{0.5}A}$  and  $b := \overrightarrow{\sqrt{0.5}B}$ , then  $R^\top R$  and  $S^\top S$  become

$$S^\top S = \begin{pmatrix} a^\top a & a^\top b \\ b^\top a & b^\top b \end{pmatrix}, \quad R^\top R = \begin{pmatrix} b^\top b & b^\top a \\ a^\top b & a^\top a \end{pmatrix}. \quad (199)$$



$SS$  and  $RR$  contain the same elements, just permuted. For arbitrary  $2 \times 2$  matrices we can write the eigenvectors and eigenvalues explicitly: ( $\zeta_{11}$ ,  $\zeta_{12}$ ,  $\zeta_{21}$ , and  $\zeta_{22}$  stand for arbitrary elements of a  $2 \times 2$  matrix)

$$\begin{aligned} S^T S x &= \begin{pmatrix} \zeta_{11} & \zeta_{12} \\ \zeta_{21} & \zeta_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \lambda_{1,2} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ \lambda_{1,2} &= \frac{1}{2} (\zeta_{11} + \zeta_{22}) \pm \frac{1}{2} \left[ (\zeta_{11} + \zeta_{22})^2 - 4(\zeta_{11}\zeta_{22} - \zeta_{12}\zeta_{21}) \right]^{\frac{1}{2}} \quad (200) \\ x_1 &= \frac{-\zeta_{12}}{\zeta_{11} - \lambda_{1,2}} x_2, \quad x_2 = \frac{-\zeta_{21}}{\zeta_{22} - \lambda_{1,2}} x_1. \end{aligned}$$

$RR$  has the same trace and determinant as  $SS$ , such that we can write analogously:

$$\begin{aligned} R^T R x &= \begin{pmatrix} \zeta_{22} & \zeta_{21} \\ \zeta_{12} & \zeta_{11} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \lambda_{1,2} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ \lambda_{1,2} &= \frac{1}{2} (\zeta_{11} + \zeta_{22}) \pm \frac{1}{2} \left[ (\zeta_{11} + \zeta_{22})^2 - 4(\zeta_{11}\zeta_{22} - \zeta_{12}\zeta_{21}) \right]^{\frac{1}{2}} \quad (201) \\ x_1 &= \frac{-\zeta_{21}}{\zeta_{22} - \lambda_{1,2}} x_2, \quad x_2 = \frac{-\zeta_{12}}{\zeta_{11} - \lambda_{1,2}} x_1. \end{aligned}$$

For the same eigenvalue  $\lambda_i$ ,  $x_1$  and  $x_2$  exchange place. With this knowledge the eigenvalue decomposition of  $R^T R$  and  $S^T S$  can be written as:

$$\begin{aligned} S^T S &= U_S D_S U_S^T = \begin{pmatrix} v_1 & u_1 \\ v_2 & u_2 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_1 & v_2 \\ u_1 & u_2 \end{pmatrix} \\ R^T R &= U_R D_R U_R^T = \begin{pmatrix} v_2 & u_2 \\ v_1 & u_1 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_2 & v_1 \\ u_2 & u_1 \end{pmatrix}. \end{aligned} \quad (202)$$

Let us show that  $\tilde{S} = \tilde{R}$ . With some algebra we find:

$$\begin{aligned} \tilde{S} &= S U_S D_S^{-1/2} = \begin{pmatrix} \frac{v_1}{\sqrt{\lambda_1}} a + \frac{v_2}{\sqrt{\lambda_1}} b & \frac{u_1}{\sqrt{\lambda_2}} a + \frac{u_2}{\sqrt{\lambda_2}} b \\ \frac{v_2}{\sqrt{\lambda_1}} b + \frac{v_1}{\sqrt{\lambda_1}} a & \frac{u_2}{\sqrt{\lambda_2}} b + \frac{u_1}{\sqrt{\lambda_2}} a \end{pmatrix} \\ \tilde{R} &= R U_R D_R^{-1/2} = \begin{pmatrix} \frac{v_2}{\sqrt{\lambda_1}} b + \frac{v_1}{\sqrt{\lambda_1}} a & \frac{u_2}{\sqrt{\lambda_2}} b + \frac{u_1}{\sqrt{\lambda_2}} a \\ \frac{v_1}{\sqrt{\lambda_1}} a + \frac{v_2}{\sqrt{\lambda_1}} b & \frac{u_1}{\sqrt{\lambda_2}} a + \frac{u_2}{\sqrt{\lambda_2}} b \end{pmatrix}. \end{aligned} \quad (203)$$

To show that  $\hat{R} = \hat{S}$  we need to show that  $U_{RS} = U_{SR}$ . Again with some algebra we find:

$$\begin{aligned} D_S^{1/2} U_S^T R^T R U_S D_S^{1/2} &= D_R^{1/2} U_R^T S^T S U_R D_R^{1/2} = \\ &= \begin{pmatrix} \lambda_1 (v_1^2 b^T b + v_1 v_2 (a^T b + b^T a) + v_2^2 a^T a) & \sqrt{\lambda_1 \lambda_2} (v_1 v_2 b^T b + v_2 u_1 a^T b + v_1 u_2 b^T a + v_2 u_2 a^T a) \\ \sqrt{\lambda_1 \lambda_2} (v_1 v_2 b^T b + v_2 u_1 b^T a + v_1 u_2 a^T b + v_2 u_2 a^T a) & \lambda_2 (u_1^2 b^T b + u_1 u_2 (a^T b + b^T a) + u_2^2 a^T a) \end{pmatrix}. \end{aligned}$$

Since  $\hat{R} = \hat{S}$  the resulting Kronecker product will be of the form  $C \otimes C$ . Also, we only need to compute  $S^T S$  and its eigendecomposition (we get  $R^T R$  for free) and the eigendecomposition of  $D_S^{1/2} U_S^T R^T R U_S D_S^{1/2}$ .

## A.3 Anti-Symmetric Kronecker Products

Let  $A \in \mathbb{R}^{N \times N}$  and  $B \in \mathbb{R}^{N \times N}$  be square matrices of same size. Then the anti-symmetric Kronecker product of  $A$  and  $B$  is in  $\mathbb{R}^{N^2 \times N^2}$  and defined as  $(A \otimes B) := \Delta(A \otimes B)\Delta^\top$  with projection operator  $\Delta$ :

$$\begin{aligned}\Delta A &= \frac{1}{2}(A - A^\top), \quad \Delta_{(ij),(kl)} = \frac{1}{2}(\delta_{ik}\delta_{jl} - \delta_{il}\delta_{kj}) \\ (A \otimes B)_{(ij),(kl)} &= \frac{1}{4}(A_{ik}B_{jl} - A_{il}B_{jk} - A_{jk}B_{il} + A_{jl}B_{ik}) \\ (A \otimes B)\vec{X} &= \frac{1}{4}(\overrightarrow{AXB^\top - AX^\top B^\top - BX^\top A^\top + BXA^\top})\end{aligned}\quad (204)$$

The anti-symmetric Kronecker product anti-symmetrizes  $X$ . If  $A = B$  then:

$$\begin{aligned}(A \otimes A)_{(ij),(kl)} &= \frac{1}{2}(A_{ik}A_{jl} - A_{jk}A_{il}) \\ (A \otimes A)\vec{X} &= \frac{1}{2}(\overrightarrow{AXA^\top - AX^\top A^\top}).\end{aligned}\quad (205)$$

The algebraic properties are similar to the ones of the symmetric Kronecker product::

$$\text{TRANSPOSE} \quad (A \otimes B)^\top = A^\top \otimes B^\top \quad (206a)$$

$$\text{FACTORIZING} \quad (A \otimes A)(C \otimes C) = AC \otimes AC \quad \text{but} \quad (A \otimes B)(C \otimes D) \neq (AC \otimes BD) \quad (206b)$$

$$(A \otimes B)(C \otimes D) = \frac{1}{2}[AC \otimes BD + AD \otimes BC] \quad (206c)$$

$$\text{INVERSE} \quad (A \otimes A)^{-1} = (A^{-1} \otimes A^{-1}) \quad \text{but} \quad (A \otimes B)^{-1} \neq (A^{-1} \otimes B^{-1}) \quad (206d)$$

$$\text{COMMUTATIVE} \quad A \otimes B = B \otimes A \quad \text{but} \quad A \otimes B \neq B \otimes A \quad (206e)$$

$$\text{TRACE} \quad \text{tr}[A \otimes B] = \frac{1}{2}(\text{tr}[A]\text{tr}[B] - \text{tr}[AB]). \quad (206f)$$

**Proof** of Eq. 206a: follows from definition and rearranging terms:

$$\begin{aligned}(A \otimes B)_{(ij),(kl)}^\top &= (A \otimes B)_{(kl),(ij)} = \frac{1}{4}(A_{ki}B_{lj} - A_{kj}B_{li} - A_{li}B_{kj} + A_{lj}B_{ki}) \\ &= \frac{1}{4}(A_{ik}^\top B_{jl}^\top - A_{jk}^\top B_{il}^\top - A_{il}^\top B_{jk}^\top + A_{jl}^\top B_{ik}^\top) = (A^\top \otimes B^\top)_{(ij),(kl)}. \quad \blacksquare\end{aligned}$$

**Proof** of Eq. 206c: follows from definition and rearranging terms:

$$\begin{aligned}[(A \otimes B)(C \otimes D)]_{(ij),(kl)} &= (A \otimes B)_{(ij)(mn)}(C \otimes D)_{(mn),(kl)} \\ &= \frac{1}{4}[A_{im}B_{jn} - A_{jm}B_{in} - A_{in}B_{jm} + A_{jn}B_{im}] \frac{1}{4}[C_{mk}D_{nl} - C_{nk}D_{ml} - C_{ml}D_{nk} + C_{nl}D_{mk}] \\ &= 2 \frac{1}{4} \frac{1}{4} [(AC)_{ik}(BD)_{jl} - (AC)_{jk}(BD)_{il} - (AC)_{il}(BD)_{jk} + (AC)_{jl}(BD)_{ik} \\ &\quad + (AD)_{ik}(BC)_{jl} - (AD)_{jk}(BC)_{il} - (AD)_{il}(BC)_{jk} + (AD)_{jl}(BC)_{ik}] \\ &= \frac{1}{2}[AC \otimes BD + AD \otimes BC]_{(ij),(kl)}. \quad \blacksquare\end{aligned}$$

**Proof of Eq. 206b:** follows directly from Eq. 206c for  $A = B$  and  $C = D$ . ■

**Proof of Eq. 206d:** follows directly from Eq. 206b. Note that in general  $(A \otimes B)^{-1} \neq A^{-1} \otimes B^{-1}$ .

$$(A \otimes A)(A^{-1} \otimes A^{-1}) = AA^{-1} \otimes AA^{-1} = I_{N \times N} \otimes I_{N \times N} = \Gamma. \quad \blacksquare$$

**Proof of Eq. 206e:** follows directly from definition and reordering. Note that  $A \otimes B \neq B \otimes A$ :

$$\begin{aligned} (A \otimes B)_{(i,j),(k,l)} &= \frac{1}{4}(A_{ik}B_{jl} - A_{il}B_{jk} - A_{jk}B_{il} + A_{jl}B_{ik}) \\ &= \frac{1}{4}(B_{ik}A_{jl} - B_{il}A_{jk} - B_{jk}A_{il} + B_{jl}A_{ik}) = (B \otimes A)_{(i,j),(k,l)}. \quad \blacksquare \end{aligned}$$

**Proof of Eq. 206f:** follows directly from definition and the definition of trace:

$$\begin{aligned} \text{tr}[A \otimes B] &= \sum_{i,j} (A \otimes B)_{(ij),(ij)} = \frac{1}{4} \sum_{i,j} (A_{ii}B_{jj} - A_{ij}B_{ji} - A_{ji}B_{ij} + A_{jj}B_{ii}) \\ &= \frac{1}{2}(\text{tr}[A] \text{tr}[B] - \text{tr}[AB]). \quad \blacksquare \end{aligned}$$



# B

## Derivation of Filtering Equations for Optimization

---

This chapter includes derivations of the most essential formulas of Chapter 9. Section B.2 and Section B.3 deal with the non-symmetric and symmetric hypothesis class on matrices respectively. Section B.4 derives low-rank approximations of the symmetric class. Additionally, Section B.1 derives the maximum marginal likelihood solution for the diagonal of the intensity matrix  $q$  for first order filtering.

### B.1 Hyper-Parameter Adaptation for First-Order Filter

The marginal likelihood for parameter  $q$ , for the current noisy data-point  $y_{t+1}$  with noise covariance  $R_{t+1}$  is:

$$\begin{aligned}
 p(y_{t+1}) &= \int p(y_{t+1}|x_{t+1}, y_1, \dots, y_t) p(x_{t+1}|y_1, \dots, y_t) dx_{t+1} \\
 &= \int \mathcal{N}(y_{t+1}; x_{t+1}, R_{t+1}) \mathcal{N}(x_{t+1}; m_{t+1-}, P_{t+1-}) dx_{t+1} \\
 &= \mathcal{N}(y_{t+1}; m_{t+1-}, P_{t+1-} + R_{t+1}) \\
 &= \frac{e^{-\frac{1}{2}(y_{t+1}-m_{t+1-})^\top (P_{t+1-}+R_{t+1})^{-1} (y_{t+1}-m_{t+1-})}}{(2\pi)^{N/2} |P_{t+1-} + R_{t+1}|^{1/2}} \\
 \log p(y_{t+1}) &\propto -\frac{1}{2} \log |P_{t+1-} + R_{t+1}| - \frac{1}{2} (y_{t+1} - m_{t+1-})^\top (P_{t+1-} + R_{t+1})^{-1} (y_{t+1} - m_{t+1-})
 \end{aligned} \tag{207}$$

This derivative assumes that all matrices  $R$ ,  $q$ ,  $P$ , et cetera are diagonal as in Section 8.2. First, we need the gradient with respect to  $q_{kk}$ , the find its root. For notational convenience we write  $\Delta_{t+1} := y_{t+1} - m_{t+1-}$  (residual) and skip all indice  $t$ :

$$\frac{\partial}{\partial q_{kk}} \log p(y_{t+1}) = -\frac{1}{2} \frac{\partial}{\partial q_{kk}} \log |P^- + R| - \frac{\partial}{\partial q_{kk}} \frac{1}{2} \Delta^\top (P^- + R)^{-1} \Delta. \tag{208}$$

From Eq. 121 we know that  $P^- = P + \Delta\tau q$ . Also,  $P^-$  is the only term that includes  $q$ . The second term of the right hand side of of Eq. 208 is:

$$\begin{aligned}
 &-\frac{\partial}{\partial q_{kk}} \frac{1}{2} \Delta^\top (P^- + R)^{-1} \Delta = -\frac{1}{2} \frac{\partial}{\partial q_{kk}} \Delta_i (\Delta\tau q + P + R)_{ij}^{-1} \Delta_j = -\frac{1}{2} \frac{\partial}{\partial q_{kk}} \Delta_i \frac{\delta_{ij}}{\Delta\tau q_{ii} + P_{ii} + R_{ii}} \Delta_j \\
 &= -\frac{1}{2} \Delta_i \frac{\partial}{\partial q_{kk}} (\Delta\tau q_{ii} + P_{ii} + R_{ii})^{-1} \Delta_i = -\frac{1}{2} \Delta_i (-1) (\Delta\tau q_{ii} + P_{ii} + R_{ii})^{-2} \frac{\partial}{\partial q_{kk}} (\Delta\tau q_{ii} + P_{ii} + R_{ii}) \Delta_i \\
 &= \frac{\Delta\tau}{2} \Delta_i (\Delta\tau q_{ii} + P_{ii} + R_{ii})^{-2} \frac{\partial q_{ii}}{\partial q_{kk}} \Delta_i = \frac{\Delta\tau}{2} \Delta_i (\Delta\tau q_{ii} + P_{ii} + R_{ii})^{-2} \Delta_i \delta_{ik} \\
 &= \frac{\Delta\tau}{2} \Delta_k (\Delta\tau q_{kk} + P_{kk} + R_{kk})^{-2} \Delta_k = \frac{\Delta\tau}{2} \left( \Delta_k G_{kk}^{-1} \right)^2
 \end{aligned} \tag{209}$$

For the first part we need the matrix identity  $\partial \log |X(z)| / \partial z = \text{tr}[X^{-1} \partial(z) X / \partial z]$

$$\begin{aligned}
-\frac{1}{2} \frac{\partial}{\partial q_{kk}} \log |P^- + R| &= -\frac{1}{2} \frac{\partial}{\partial q_{kk}} \log |\Delta \tau q + P + R| = -\frac{1}{2} \text{tr} \left[ G^{-1} \frac{\partial(\Delta \tau q + P + R)}{\partial q_{kk}} \right] \\
&= -\frac{1}{2} G_{ij}^{-1} \left( \frac{\partial(\Delta \tau q_{ji} + P_{ji} + R_{ji})}{\partial q_{kk}} \right) = -\frac{1}{2} \Delta \tau G_{ij}^{-1} \frac{\partial q_{ji}}{\partial q_{kk}} = -\frac{1}{2} \Delta \tau G_{ij}^{-1} \delta_{kj} \delta_{ik} \delta_{ij} \\
&= -\frac{1}{2} \Delta \tau G_{kk}^{-1}
\end{aligned} \tag{210}$$

Therefore the gradient from Eq. 208 is:

$$\frac{\partial}{\partial q_{kk}} \log p(y_{t+1}) = \frac{\Delta \tau}{2} (\Delta_k G_{kk}^{-1})^2 - \frac{\Delta \tau}{2} G_{kk}^{-1} \tag{211}$$

The analytic root of the gradient thus is at

$$\begin{aligned}
\Delta \tau (\Delta_k G_{kk}^{-1})^2 &= \Delta \tau G_{kk}^{-1} \\
\Delta \tau \Delta_k \Delta_k \frac{1}{(\Delta \tau q_{kk} + P_{kk} + R_{kk})^2} &= \Delta \tau \frac{1}{(\Delta \tau q_{kk} + P_{kk} + R_{kk})} \\
\Delta_k \Delta_k &= \Delta \tau q_{kk} + P_{kk} + R_{kk} \\
q_{kk} &= \frac{1}{\Delta \tau} (\Delta_k \Delta_k - P_{kk} - R_{kk}).
\end{aligned} \tag{212}$$

If we want to approximate Eq. 207 with a gamma distribution around some point  $q_0$  then we also need the second derivative; so we list it here for completeness. From Eq. 211 we have:

$$\begin{aligned}
\frac{\partial^2}{\partial^2 q_{kk}} \log p(y_{t+1}) &= \frac{\Delta \tau}{2} \Delta_k \Delta_k \frac{\partial}{\partial q_{kk}} (G_{kk}^{-1})^2 - \frac{\Delta \tau}{2} \frac{\partial}{\partial q_{kk}} G_{kk}^{-1} \\
&= \left( \Delta \tau \Delta_k \Delta_k G_{kk}^{-1} - \frac{1}{2} \Delta \tau \right) \frac{\partial}{\partial q_{kk}} (\Delta \tau q_{kk} + P_{kk} + R_{kk})^{-1} \\
&= \left( \Delta \tau \Delta_k \Delta_k G_{kk}^{-1} - \frac{1}{2} \Delta \tau \right) (\Delta \tau q_{kk} + P_{kk} + R_{kk})^{-2} (-1) \Delta \tau \\
&= \left( \frac{1}{2} - \Delta_k \Delta_k G_{kk}^{-1} \right) (G_{kk}^{-1} \Delta \tau)^2.
\end{aligned} \tag{213}$$

For  $P$ ,  $q$ , and  $R$  scalar matrices with  $P = pI$ ,  $q = uI$  and  $R = rI$ , the gradient and its root are:

$$\begin{aligned}
\frac{\partial}{\partial u} \log p(y_{t+1}) &= \frac{\Delta \tau}{2} \sum_{i=1}^N (\Delta_i G_{ii}^{-1})^2 - \frac{\Delta \tau}{2} \text{tr}[G^{-1}] \\
u^* &= \frac{1}{\Delta \tau} \left( \frac{1}{N} \sum_i \Delta_i^2 - p - r \right),
\end{aligned} \tag{214}$$

which measure the how the *mean-discrepancy* between observation and predictive state which can not be explained by  $p$  or  $r$  already.

## B.2 Second-Order Filter (Non-symmetric)

This section derives the explicit forms of the updated Kalman equation of Section 9.1.1. The iteration index  $t$  will be occasionally dropped, in order to de-clutter notation. For computing the updated Kalman equations we need terms of the following structure:

$$\begin{aligned} [(I \otimes v^\top)(A^\top \otimes B^\top)]_{i,(kl)} &= \delta_{ia} v_b A_{ak}^\top B_{bl}^\top = A_{ik}^\top B_{bl}^\top v_b = A_{ik}^\top (Bv)_l = (A^\top \otimes (Bv)^\top)_{i,(kl)} \\ [(A \otimes B)(I \otimes v)]_{(ik),l} &= A_{ia} B_{kb} \delta_{al} v_b = A_{il} B_{kb} v_b = A_{il} (Bv)_k = (A \otimes (Bv))_{(ik),l} \\ [(I \otimes s^\top)(A \otimes A)(I \otimes v)]_{p,q} &= \delta_{pa} s_b A_{ac} A_{bd} \delta_{cq} v_d = s_b A_{pq} A_{bd} v_d = A_{pq} (s^\top Av). \end{aligned} \quad (215)$$

All updated Kalman equations (Eq. 133) need explicit form of the innovation  $G_{t+1} = P_{t+1-}^{\nabla\nabla} + R_{t+1}$  and related quantities which we derive here: inserting 215 in 135 yields a rewritten form of the predictive covariance bloc  $P_{t+1-}^{\nabla\nabla}$ :

$$P_{t+1-}^{\nabla\nabla} = P_t^{\nabla\nabla} + \Delta\tau/3 V_t (s_t^\top V_t s_t) + U_t^\top (v_t^\top U_t^\top s_t) + U_t (s_t^\top U_t v_t) + W_t (s_t^\top W_t s_t). \quad (216)$$

For ease of notation, define  $\tilde{s} := \frac{\Delta\tau}{2} s$  and also:

$$\tilde{V}_t^\top := P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} V_t, \quad \tilde{U}_t^\top := P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} U_t^\top, \quad \tilde{W}_t s^\top := P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} W_t \quad (217)$$

The quantities defined in Eq. 217 can be roughly thought of as a ‘gain-corrected’ versions of the original quantities; if observations are noise free ( $R = 0$ ), then  $\tilde{V} = V$ ,  $\tilde{W} = W$  and  $\tilde{U} = U$ . For the updated covariance block  $P_{t+1}^{\nabla B}$  we need the intermediate quantity:

$$\begin{aligned} [P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B}]_{p,(kl)} &= (P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1})_{p,i} [V_{ik} (V\tilde{s})_l + U_{ik}^\top (Uv)_l + W_{ik} (Ws)_l] \\ &= \tilde{V}_{pk}^\top (V\tilde{s})_l + \tilde{U}_{pk}^\top (Uu)_l + \tilde{W}_{pk}^\top (Ws)_l \\ &= [(\tilde{V}_t^\top \otimes (V\tilde{s}_t)^\top) + (\tilde{U}_t^\top \otimes (U_t u_t)^\top) + (\tilde{W}_t^\top \otimes (W_t s_t)^\top)]_{p,(kl)} \end{aligned} \quad (218)$$

Therefore the updated covariance block  $P_{t+1}^{\nabla B}$  is:

$$\begin{aligned} P_{t+1}^{\nabla B} &= (V - \tilde{V}_t^\top) \otimes (V\tilde{s}_t)^\top + (U_t^\top - \tilde{U}_t^\top) \otimes (U_t v_t)^\top + (W_t - \tilde{W}_t^\top) \otimes (W_t s_t)^\top \\ &= (I \otimes \tilde{s}_t^\top) ([V - \tilde{V}_t^\top] \otimes V) + (I \otimes v_t^\top) ([U_t^\top - \tilde{U}_t^\top] \otimes U_t^\top) + (I \otimes s_t^\top) ([W_t - \tilde{W}_t^\top] \otimes W_t) \\ &= (I \otimes \tilde{s}_t^\top) (V \otimes V) + (I \otimes v_t^\top) (U_t^\top \otimes U_t^\top) + (I \otimes s_t^\top) (W_t \otimes W_t) \\ &\quad - (I \otimes \tilde{s}_t^\top) (\tilde{V}_t^\top \otimes V) - (I \otimes v_t^\top) (\tilde{U}_t^\top \otimes U_t^\top) - (I \otimes s_t^\top) (\tilde{W}_t^\top \otimes W_t) \end{aligned} \quad (219)$$

The updated covariance block  $P_{t+1}^{BB}$  requires the term  $P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B}$ :

$$\begin{aligned} [P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B}]_{(ij),(kl)} &= [V_{ip} (V\tilde{s})_j + U_{ip} (Uv)_j + W_{ip} (Ws)_j] G_{pq}^{-1} \\ &\quad [V_{qk} (V\tilde{s})_l + U_{qk}^\top (Uv)_l + W_{qk} (Ws)_l] \end{aligned} \quad (220)$$

Eq 220 contains only terms of general form:

$$A_{ip} (A\zeta)_j G_{pq}^{-1} B_{qk}^\top (B\eta)_l = [AG^{-1}B^\top]_{ik} [(A\zeta)(B\eta)^\top]_{jl} = [AG^{-1}B^\top \otimes (A\zeta)(B\eta)^\top]_{(ij),(kl)}. \quad (221)$$

This mean that  $P_{t+1}^{BB}$  can be re-written as:

$$\begin{aligned}
P_{t+1}^{BB} &= \Delta\tau(V \otimes V) + (W_t \otimes W_t) \\
&\quad - VG_{t+1}^{-1}V \otimes (V\tilde{s}_t)(V\tilde{s}_t)^\top - U_tG_{t+1}^{-1}V_t \otimes (U_tv_t)(V\tilde{s}_t)^\top - W_tG_{t+1}^{-1}V \otimes (W_t s_t)(V\tilde{s}_t)^\top \\
&\quad - VG_{t+1}^{-1}U_t^\top \otimes (V\tilde{s}_t)(U_tv_t)^\top - U_tG_{t+1}^{-1}U_t^\top \otimes (U_tv_t)(U_tv_t)^\top - W_tG_{t+1}^{-1}U_t^\top \otimes (W_t s_t)(U_tv_t)^\top \\
&\quad - VG_{t+1}^{-1}W_t \otimes (V\tilde{s}_t)(W_t s_t)^\top - U_tG_{t+1}^{-1}W_t \otimes (U_tv_t)(W_t s_t)^\top - W_tG_{t+1}^{-1}W_t \otimes (W_t s_t)(W_t s_t)^\top
\end{aligned} \tag{222}$$

For the Hessian part of the updated mean  $m_{t+1}^B$  we need the term  $P_{t+1}^{B\nabla}G_{t+1}^{-1}$  multiplied with the residual  $\Delta_t := [y_{t+1} - m_t^\nabla - (I \otimes s_t^\top)m_t^B]$ .

Again use  $\tilde{s}_t = \frac{\Delta_t}{2}s_t$ .

$$\begin{aligned}
[P_{t+1}^{B\nabla}G_{t+1}^{-1}\Delta_t]_{(ij)} &= [V_{ip}(V\tilde{s})_j + U_{ip}(Uv)_j + W_{ip}(Ws)_j](G^{-1}\Delta)_p \\
&= (VG^{-1}\Delta)_i(V\tilde{s})_j + (UG^{-1}\Delta)_i(Uv)_j + (WG^{-1}\Delta)_i(Ws)_j
\end{aligned} \tag{223}$$

Therefore  $m_{t+1}^B$  becomes:

$$m_{t+1}^B = m_t^B + (V_tG_{t+1}^{-1}\Delta)(V_t\tilde{s}_t)^\top + (U_tG_{t+1}^{-1}\Delta)(U_tv_t)^\top + (W_tG_{t+1}^{-1}\Delta)(W_t s_t)^\top. \tag{224}$$

The gradient part of the updated mean  $m_{t+1}^\nabla$  requires the term  $P_{t+1}G_{t+1}^{-1}\Delta_t$ .

For appropriate forms of all  $N \times N$  objects we can perform this computation (see also Sections 9.3 and B.4). The same holds for  $(I \otimes s_t^\top)m_t^B$ .

Therefore we can directly compute:

$$m_{t+1}^\nabla = m_t^\nabla + (I \otimes s_t^\top)m_t^B + P_{t+1}^{\nabla\nabla}G_{t+1}^{-1}\Delta_t. \tag{225}$$

### B.3 Second-Order Filter (Symmetric)

This section derives the explicit forms of the updated Kalman equation of Section 9.1.2. The iteration index  $t$  will be occasionally dropped, in order to de-clutter notation. The section is structured similarly to the derivation of the non-symmetric filter in Section B.2. For computing the updated Kalman equations we need terms of the following structure:

$$\begin{aligned}
[(I \otimes v^\top)(A^\top \otimes A^\top)]_{i,(kl)} &= \frac{1}{2} (A_{ik}^\top(Av)_l + A_{il}^\top(Av)_k) \\
[(A \otimes A)(I \otimes v)]_{(kl),i} &= \frac{1}{2} (A_{ki}(Av)_l + A_{li}(Av)_k) \\
[(I \otimes s^\top)(A \otimes A)(I \otimes v)]_{p,q} &= \frac{1}{2} (A_{pq}(s^\top Av) + (Av)_p(A^\top s)_q)
\end{aligned} \tag{226}$$

All updated Kalman equations (Eq. 133) need explicit form of the innovation  $G_{t+1} = P_{t+1}^{\nabla\nabla} + R_{t+1}$  and related quantities which we derive here: inserting 226 in 135 yields a rewritten form of the predictive covariance bloc  $P_{t+1}^{\nabla\nabla}$ , which we will also use in Section 9.3 and B.4.

$$\begin{aligned}
P_{t+1}^{\nabla\nabla} &= P_t^{\nabla\nabla} + \frac{1}{2} [\Delta\tau/3V(s_t^\top V s_t) + \Delta\tau/3(V s_t)(V s_t)^\top + U_t^\top(v_t^\top U_t^\top s_t) + (U_t^\top s_t)(U_tv_t)^\top \\
&\quad + U_t(s_t^\top U_tv_t) + (U_tv_t)(U_t^\top s_t)^\top + W_t(s_t^\top W_t s_t) + (W_t s_t)(W_t s_t)^\top]
\end{aligned} \tag{227}$$



For ease of notation, define  $\tilde{s} := \frac{\Delta\tau}{2}s$  and also:

$$\begin{aligned}\tilde{V}_t^\top &= P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} V & \tilde{v}_t &= \tilde{V}_t^{-1} V s_t \\ \tilde{U}_t^\top &= P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} U_t^\top & \tilde{u}_t &= \tilde{U}_t^{-1} U_t v_t \\ \tilde{W}_t^\top &= P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} W_t & \tilde{w}_t &= \tilde{W}_t^{-1} W_t s_t\end{aligned}\quad (228)$$

The quantities defined in Eq. 228 can be roughly thought of as a ‘gain-corrected’ versions of the original quantities; if observations are noise free ( $R = 0$ ), then  $\tilde{V} = V$ ,  $\tilde{W} = W$ ,  $\tilde{U} = U$ ,  $\tilde{v} = \tilde{s}$ ,  $\tilde{u} = v$ , and  $\tilde{w} = s$ . For the updated covariance block  $P_{t+1}^{\nabla B}$  we need the intermediate quantity:

$$\begin{aligned}\left[ P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B} \right]_{p,(kl)} &= (P_{t+1-}^{\nabla\nabla} G_{t+1}^{-1})_{p,i} \\ &\quad \frac{1}{2} [V_{ik}(V\tilde{s})_l + V_{il}(V\tilde{s})_k + U_{ik}^\top(Uv)_l + U_{il}^\top(Uv)_k + W_{ik}(Ws)_l + W_{il}(Ws)_k] \\ &= \frac{1}{2} [\tilde{V}_{pk}^\top(\tilde{V}\tilde{v})_l + \tilde{V}_{pl}^\top(\tilde{V}\tilde{v})_k + \tilde{U}_{pk}^\top(\tilde{U}\tilde{u})_l + \tilde{U}_{pl}^\top(\tilde{U}\tilde{u})_k + \tilde{W}_{pk}^\top(\tilde{W}\tilde{w})_l + \tilde{W}_{pl}^\top(\tilde{W}\tilde{w})_k] \\ &= [(I \otimes \tilde{v}_t^\top)(\tilde{V}_t^\top \otimes \tilde{V}_t^\top) + (I \otimes \tilde{u}_t^\top)(\tilde{U}_t^\top \otimes \tilde{U}_t^\top) + (I \otimes \tilde{w}_t^\top)(\tilde{W}_t^\top \otimes \tilde{W}_t^\top)]_{p,(kl)}.\end{aligned}\quad (229)$$

Therefore the updated covariance block  $P_{t+1}^{\nabla B}$  is:

$$\begin{aligned}P_{t+1}^{\nabla B} &= (I \otimes \tilde{s}_t^\top)(V \otimes V) + (I \otimes v_t^\top)(U_t^\top \otimes U_t^\top) + (I \otimes s_t^\top)(W_t \otimes W_t) \\ &\quad - (I \otimes \tilde{v}_t^\top)(\tilde{V}_t^\top \otimes \tilde{V}_t^\top) - (I \otimes \tilde{u}_t^\top)(\tilde{U}_t^\top \otimes \tilde{U}_t^\top) - (I \otimes \tilde{w}_t^\top)(\tilde{W}_t^\top \otimes \tilde{W}_t^\top).\end{aligned}\quad (230)$$

The updated covariance block  $P_{t+1}^{BB}$  requires the term  $P_{t+1-}^{B\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B}$ :

$$\begin{aligned}\left[ P_{t+1-}^{B\nabla} G_{t+1}^{-1} P_{t+1-}^{\nabla B} \right]_{(ij),(kl)} &= \frac{1}{4} [V_{ip}(V\tilde{s})_j + V_{jp}(V\tilde{s})_i + U_{ip}(Uv)_j + U_{jp}(Uv)_i \\ &\quad + W_{ip}(Ws)_j + W_{jp}(Ws)_i] G_{pq}^{-1} [V_{qk}(V\tilde{s})_l + V_{ql}(V\tilde{s})_k + U_{qk}^\top(Uv)_l + U_{ql}^\top(Uv)_k \\ &\quad + W_{qk}(Ws)_l + W_{ql}(Ws)_k].\end{aligned}\quad (231)$$

Eq 231 contains only terms of general form:

$$\begin{aligned}&\frac{1}{4} [A_{ip}\zeta_j + A_{jp}\zeta_i] G_{pq}^{-1} [B_{qk}\eta_l + B_{ql}\eta_k] \\ &= \frac{1}{4} [A_{ip}G_{pq}^{-1}B_{qk}\eta_l\zeta_j + A_{jp}G_{pq}^{-1}B_{qk}\eta_l\zeta_i + A_{ip}G_{pq}^{-1}B_{ql}\eta_k\zeta_j + A_{jp}G_{pq}^{-1}B_{ql}\eta_k\zeta_i] \\ &= \frac{1}{4} [(AG^{-1}B)_{ik}(\zeta\eta^\top)_{jl} + (AG^{-1}B)_{jk}(\zeta\eta^\top)_{il} + (AG^{-1}B)_{il}(\zeta\eta^\top)_{jk} + (AG^{-1}B)_{jl}(\zeta\eta^\top)_{ik}] \\ &= [AG^{-1}B \otimes \zeta\eta^\top]_{(ij),(kl)}.\end{aligned}\quad (232)$$

This mean that  $P_{t+1}^{BB}$  can be re-written as:

$$\begin{aligned}P_{t+1}^{BB} &= \Delta\tau(V \otimes V) + (W_t \otimes W_t) \\ &\quad - VG_{t+1}^{-1}V \otimes (V\tilde{s}_t)(V\tilde{s}_t)^\top - U_t G_{t+1}^{-1}U_t \otimes (U_t v_t)(U_t v_t)^\top - W_t G_{t+1}^{-1}W_t \otimes (W_t s_t)(W_t s_t)^\top \\ &\quad - VG_{t+1}^{-1}U_t^\top \otimes (V\tilde{s}_t)(U_t v_t)^\top - U_t G_{t+1}^{-1}U_t^\top \otimes (U_t v_t)(U_t v_t)^\top - W_t G_{t+1}^{-1}U_t^\top \otimes (W_t s_t)(U_t v_t)^\top \\ &\quad - VG_{t+1}^{-1}W_t \otimes (V\tilde{s}_t)(W_t s_t)^\top - U_t G_{t+1}^{-1}W_t \otimes (U_t v_t)(W_t s_t)^\top - W_t G_{t+1}^{-1}W_t \otimes (W_t s_t)(W_t s_t)^\top\end{aligned}\quad (233)$$

For the Hessian part of the updated mean  $m_{t+1}^B$  we need the term  $P_{t+1}^{B\nabla} G_{t+1}^{-1}$  multiplied with the residual  $\Delta_t := [y_{t+1} - m_t^\nabla - (I \otimes s_t^\top) m_t^B]$ . Again use  $\tilde{s}_t = \frac{\Delta_t}{2} s_t$ .

$$\begin{aligned} [P_{t+1}^{B\nabla} G_{t+1}^{-1} \Delta_t]_{(ij)} &= \frac{1}{2} [V_{ip}(V\tilde{s})_j + V_{jp}(V\tilde{s})_i + U_{ip}(Uv)_j + U_{jp}(Uv)_i \\ &\quad + W_{ip}(Ws)_j + W_{jp}(Ws)_i] (G^{-1}\Delta)_p \\ &= \frac{1}{2} [(VG^{-1}\Delta)_i (V\tilde{s})_j + (VG^{-1}\Delta)_j (V\tilde{s})_i + (UG^{-1}\Delta)_i (Uv)_j \\ &\quad + (UG^{-1}\Delta)_j (Uv)_i + (WG^{-1}\Delta)_i (Ws)_j + (WG^{-1}\Delta)_j (Ws)_i] \end{aligned} \quad (234)$$

Therefore  $m_{t+1}^B$  becomes:

$$\begin{aligned} m_{t+1}^B &= m_t^B + \frac{1}{2} [(V_t G_{t+1}^{-1} \Delta) (V_t \tilde{s}_t)^\top + (V_t \tilde{s}_t) (V_t G_{t+1}^{-1} \Delta)^\top + (U_t G_{t+1}^{-1} \Delta) (U_t v_t)^\top \\ &\quad + (U_t v_t) (U_t G_{t+1}^{-1} \Delta)^\top + (W_t G_{t+1}^{-1} \Delta) (W_t s_t)^\top + (W_t s_t) (W_t G_{t+1}^{-1} \Delta)^\top]. \end{aligned} \quad (235)$$

The gradient part of the updated mean  $m_{t+1}^\nabla$  requires the term  $P_{t+1}^\nabla G_{t+1}^{-1} \Delta_t$ .

For appropriate forms of all  $N \times N$  objects we can perform this computation (see also Section 9.3 and B.4). The same holds for  $(I \otimes s_t^\top) m_t^B$ . Therefore we can directly compute:

$$m_{t+1}^\nabla = m_t^\nabla + (I \otimes s_t^\top) m_t^B + P_{t+1}^{\nabla\nabla} G_{t+1}^{-1} \Delta_t. \quad (236)$$

#### B.4 Low-Rank Approximation (Second-Order Filter, Symmetric)

This section states the solution to linear-cost filtering equations on Hessians. All matrices have scalar-plus-low-rank form as in Eq. 156. The derivations and intermediate steps are left out because they are very lengthy (although straightforward by inserting the low-rank assumptions into the Kalman equations). For better readability, terms are color coded in **red** for contributions of path segments  $s_t$ , **orange** for contributions of gradients  $y$ , and **green** for contributions of off-diagonal noise terms  $v_t$ .

**BASIS:** Given an new observation pair  $(s, y)$ , and the updated state from the previous iteration  $\pi_v, \pi_{m^\nabla}, \pi_\nabla, \sigma_V, \sigma_W, \sigma_U, \sigma_B, \sigma_R, \Lambda_V, \Lambda_W, \Lambda_U, \Lambda_B$ , with basis  $P, P^B$ , then we can write:

$$\begin{aligned}
 m^\nabla &= P^B \pi_{m^\nabla} & P_{t+1}^B &= \begin{bmatrix} P & \hat{s}_\perp & \hat{p}^Y & \hat{y}_\perp \end{bmatrix} \in \mathbb{R}^{N \times 2(M+1)} \\
 v &= P \pi_v & P^\top P &= \begin{bmatrix} I_M \\ s_\perp^\top P \\ \frac{s_\perp^\top P}{\|s_\perp\|} \end{bmatrix} = \begin{bmatrix} I_{M \times M} \\ \mathbf{0}_{1 \times M} \end{bmatrix} \in \mathbb{R}^{(M+1) \times M} \\
 s &= P \pi_s + \|s_\perp\| \hat{s}_\perp & p_\Delta^- &= \begin{bmatrix} I_{(M+1) \times (M+1)} & \mathbf{0}_{(M+1) \times (M+1)} \end{bmatrix} \pi_\Delta \\
 y &= P_-^B \pi_y + \|y_\perp\| \hat{y}_\perp & p_\Delta &= \begin{bmatrix} I_{M \times M} & \mathbf{0}_{M \times (M+2)} \end{bmatrix} \pi_\Delta \\
 s^\top s &= \pi_s^\top \pi_s + \|s_\perp\|^2 & \pi_{p^Y} &\in \mathbb{R}^{1 \times M}, n^Y \in \mathbb{R}^{M \times M} \\
 P_{\text{old}}^Y &= \hat{s}_\perp \pi_{p^Y} + \hat{p}^Y n^Y & v^\top s &= \pi_v^\top \pi_s \\
 P_- &= \begin{bmatrix} P & \hat{s}_\perp \end{bmatrix} \in \mathbb{R}^{N \times (M+1)} & & \\
 P_-^B &= \begin{bmatrix} P & \hat{s}_\perp & \hat{p}^Y \end{bmatrix} \in \mathbb{R}^{N \times (2M+1)} & &
 \end{aligned}$$

It can be shown (by inserting low rank approximations into the Kalman updates) that especially  $m^\nabla$ ,  $m^B$ ,  $v$  et cetera keep their respective forms even after the Kalman update. The only forms that have to be projected back again are  $P^{BB}$  and  $P^{\nabla B}$  just like in the original non-low-rank updates.

USEFUL NOTATION:  $s^\top s$  and  $v^\top s$  given above:

$$\begin{aligned}
 a_{\bar{v}} &:= \sigma_{\bar{v}}(s^\top s) + \pi_s^\top \Lambda_{\bar{v}} \pi_s & b_v &:= 1/2\Delta t / 2\sigma_V^2 \Lambda_{G-1} p_\Delta^- & d_v &:= 1/2\Delta t / 2\sigma_V \sigma_{G-1} \pi_s^\top \Lambda_V^\top \\
 a_{U^\top} &:= \sigma_U(v^\top s) + \pi_v^\top \Lambda_U^\top \pi_s & & & & \\
 a_U &:= \sigma_U(s^\top v) + \pi_s^\top \Lambda_U \pi_v & b_U &:= 1/2\sigma_U^2 \Lambda_{G-1} p_\Delta^- & d_U &:= 1/2\sigma_U \sigma_{G-1} \pi_v^\top \Lambda_U^\top \\
 a_W &:= \sigma_W(s^\top s) + \pi_s^\top \Lambda_W \pi_s & b_W &:= 1/2\sigma_W^2 \Lambda_{G-1} p_\Delta^- & d_W &:= 1/2\sigma_W \sigma_{G-1} \pi_s^\top \Lambda_W^\top \\
 \\
 c_U &:= 1/2\sigma_{G-1} \sigma_U \Lambda_U p_\Delta + 1/2\sigma_U \Lambda_U (P^\top P_-) \Lambda_{G-1} p_\Delta^- & e_U &:= 1/2\sigma_U \Lambda_{G-1} p_\Delta^- \pi_v^\top \Lambda_U^\top \\
 c_V &:= 1/2\Delta t / 2\sigma_{G-1} \sigma_V \Lambda_V p_\Delta + 1/2\Delta t / 2\sigma_V \Lambda_V (P^\top P_-) \Lambda_{G-1} p_\Delta^- & e_V &:= 1/2\Delta t / 2\sigma_V \Lambda_{G-1} p_\Delta^- \pi_s^\top \Lambda_V^\top \\
 c_W &:= 1/2\sigma_{G-1} \sigma_W \Lambda_W p_\Delta + 1/2\sigma_W \Lambda_W (P^\top P_-) \Lambda_{G-1} p_\Delta^- & e_W &:= 1/2\sigma_W \Lambda_{G-1} p_\Delta^- \pi_s^\top \Lambda_W^\top \\
 \\
 f_U &:= 1/2\sigma_{G-1} \Lambda_U p_\Delta \pi_v^\top \Lambda_U^\top + 1/2\Lambda_U (P^\top P_-) \Lambda_{G-1} p_\Delta^- \pi_v^\top \Lambda_U^\top & g_U &:= 1/2\sigma_U^2 \sigma_{G-1} \\
 f_V &:= 1/2\Delta t / 2\sigma_{G-1} \Lambda_V p_\Delta \pi_s^\top \Lambda_V^\top / 2 + 1/2\Delta t / 2\Lambda_V (P^\top P_-) \Lambda_{G-1} p_\Delta^- \pi_s^\top \Lambda_V^\top & g_V &:= 1/2\Delta t / 2\sigma_V^2 \sigma_{G-1} \\
 f_W &:= 1/2\sigma_{G-1} \Lambda_W p_\Delta \pi_s^\top \Lambda_W^\top + 1/2\Lambda_W (P^\top P_-) \Lambda_{G-1} p_\Delta^- \pi_s^\top \Lambda_W^\top & g_W &:= 1/2\sigma_W^2 \sigma_{G-1} \\
 \\
 \lambda_a &:= f_U + f_U^\top + f_V + f_V^\top + f_W + f_W^\top + \pi_v c_U^\top + c_U \pi_v^\top + \pi_s c_V^\top + \pi_s c_W^\top + c_V \pi_s^\top + c_W \pi_s^\top \\
 \lambda_b &:= e_V + e_W + e_U + b_U \pi_v^\top + b_V \pi_s^\top + b_W \pi_s^\top \\
 \lambda_c &:= g_U \pi_v + d_V^\top + d_W^\top + d_U^\top + g_V \pi_s + g_W \pi_s \\
 \lambda_d &:= \|s_\perp\| [b_V + b_W] \\
 \lambda_e &:= \|s_\perp\| [c_V + c_W] \\
 \lambda_f &:= \|s_\perp\| [g_V + g_W]
 \end{aligned}$$

## B.4.1 Predictive Equations

PREDICT  $P^{\nabla\nabla}$ :

$$\begin{aligned}
P_{t+1-}^{\nabla\nabla} &= (\sigma_{\nabla} + 1/2 [\sigma_{\bar{v}} a_{\bar{v}} + \sigma_U a_{U^T} + \sigma_U a_U + \sigma_W a_W]) I \\
&+ P \left( \Lambda_{\nabla} + 1/2 [a_{\bar{v}} \Lambda_{\bar{v}} + a_{U^T} \Lambda_U^T + a_U \Lambda_U + a_W \Lambda_W + \Lambda_{\bar{v}} \pi_s \pi_s^T \Lambda_{\bar{v}} \right. \\
&+ \Lambda_U^T \pi_s \pi_v^T \Lambda_U^T + \Lambda_U \pi_v \pi_s^T \Lambda_U + \Lambda_W \pi_s \pi_s^T \Lambda_W + \sigma_{\bar{v}} \Lambda_{\bar{v}} \pi_s \pi_s^T + \sigma_U \Lambda_U \pi_v \pi_s^T \\
&+ \sigma_W \Lambda_W \pi_s \pi_s^T + \sigma_U \Lambda_U^T \pi_s \pi_v^T + \sigma_U \pi_v \pi_s^T \Lambda_U + \sigma_{\bar{v}} \pi_s \pi_s^T \Lambda_{\bar{v}} + \sigma_U \pi_s \pi_v^T \Lambda_U^T \\
&\left. + \sigma_W \pi_s \pi_s^T \Lambda_W + \sigma_U^2 \pi_s \pi_v^T + \sigma_U^2 \pi_v \pi_s^T + \sigma_{\bar{v}}^2 \pi_s \pi_s^T + \sigma_W^2 \pi_s \pi_s^T \right) P^T \\
&+ P \left( 1/2 \|s_{\perp}\| [\sigma_{\bar{v}} \Lambda_{\bar{v}} \pi_s + \sigma_U \Lambda_U \pi_v + \sigma_W \Lambda_W \pi_s + \sigma_U^2 \pi_v + \sigma_{\bar{v}}^2 \pi_s + \sigma_W^2 \pi_s] \right) \hat{s}_{\perp}^T \\
&+ \hat{s}_{\perp} \left( 1/2 \|s_{\perp}\| [\sigma_{\bar{v}} \pi_s^T \Lambda_{\bar{v}} + \sigma_U \pi_v^T \Lambda_U^T + \sigma_W \pi_s^T \Lambda_W + \sigma_U^2 \pi_v^T + \sigma_{\bar{v}}^2 \pi_s^T + \sigma_W^2 \pi_s^T] \right) P^T \\
&+ \hat{s}_{\perp} (1/2 \|s_{\perp}\|^2 [\sigma_{\bar{v}}^2 + \sigma_W^2]) \hat{s}_{\perp}^T = \sigma_{\nabla-} I + P_- \Lambda_{\nabla-} P_-^T
\end{aligned} \tag{237}$$

PREDICT  $m^{\nabla}$ :

$$\begin{aligned}
m_{t+1-}^{\nabla} &= P (\pi_{m^{\nabla}}^P + \sigma_B \pi_s + \Lambda_B^{PP} \pi_s + \Lambda_B^{PY} \pi_{p^Y}^T \|s_{\perp}\|) \\
&+ \hat{s}_{\perp} (\pi_{p^Y} \pi_{m^{\nabla}}^Y + \sigma_B \|s_{\perp}\| + \pi_{p^Y} \Lambda_B^{YP} \pi_s + \pi_{p^Y} \Lambda_B^{YY} \pi_{p^Y}^T \|s_{\perp}\|) \\
&+ \hat{P}^Y (n^Y \pi_{m^{\nabla}}^Y + n^Y \Lambda_B^{YP} \pi_s + n^Y \Lambda_B^{YY} \pi_{p^Y}^T \|s_{\perp}\|) \\
&= \begin{bmatrix} P & \hat{s}_{\perp} & \hat{P}^Y \end{bmatrix} \pi_{m^{\nabla-}} = P_-^B \pi_{m^{\nabla-}}^{t+1}
\end{aligned} \tag{238}$$

## B.4.2 Updated Equations

INNOVATION MATRIX  $G$ :

$$\begin{aligned}
G^{-1} &= \sigma_{G-1} I + P_- \Lambda_{G-1} P_-^T \quad \text{with} \quad \Lambda_{G-1} = -\sigma_{G-1}^2 [\sigma_{G-1} I + \Lambda_{\nabla-}^{-1}]^{-1}, \\
&\text{and} \quad \sigma_{G-1} = (\sigma_{\nabla-} + \sigma_R)^{-1} = \sigma_G^{-1}
\end{aligned} \tag{239}$$

UPDATE STATE:  $m^{\nabla}$ :

$$\begin{aligned}
\Delta &= P_-^B (\pi_y - \pi_{m^{\nabla-}}) + \|y_{\perp}\| \hat{y}_{\perp} = P_{t+1}^B \pi_{\Delta} \\
m_{t+1}^{\nabla} &= P_-^B \pi_{m^{\nabla-}} + P_{t+1}^B (\sigma_{\nabla-} - \sigma_{G-1}) \pi_{\Delta} \\
&+ P_- (\sigma_{\nabla-} \Lambda_{G-1} p_{\Delta}^- + \sigma_{G-1} \Lambda_{\nabla-} p_{\Delta}^- + \Lambda_{\nabla-} \Lambda_{G-1} p_{\Delta}^-) = P_{t+1}^B \pi_{m^{\nabla}}^{t+1}
\end{aligned} \tag{240}$$

UPDATE STATE:  $m^B$ :

$$\begin{aligned}
 m_{t+1}^B &= \sigma_B I \\
 &+ P \left( \Lambda_B^{PP} + \lambda_b^{PP} + \lambda_b^{PP\top} + \lambda_a \right) P^\top \\
 &+ P \left( \lambda_d^{Ps} + \lambda_b^{sP\top} + \Lambda_B^{PY} \pi_{pY}^\top + \lambda_e \right) \hat{s}_\perp^\top \\
 &+ \hat{s}_\perp \left( \lambda_d^{Ps\top} + \lambda_b^{sP} + \pi_{pY} \Lambda_B^{YP} + \lambda_e^\top \right) P^\top \\
 &+ \hat{s}_\perp \left[ \pi_{pY} \Lambda_B^{YY} \pi_{pY}^\top + \lambda_d^{ss} + \lambda_d^{ss} \right] \hat{s}_\perp^\top \\
 &+ P \left[ \Lambda_B^{PY} n_Y^\top \right] \hat{P}^{Y\top} + \hat{P}^Y \left[ n_Y \Lambda_B^{YP} \right] P^\top \\
 &+ \hat{s}_\perp \left[ \pi_{pY} \Lambda_B^{YY} n_Y^\top \right] \hat{P}^{Y\top} + \hat{P}^Y \left[ n_Y \Lambda_B^{YY} \pi_{pY}^\top \right] \hat{s}_\perp^\top \\
 &+ \hat{P}^Y \left[ n_Y \Lambda_B^{YY} n_Y^\top \right] \hat{P}^{Y\top} \\
 &+ P_{t+1}^B \left( \pi_\Delta \lambda_c^\top \right) P^\top + P \left( \lambda_c \pi_\Delta^\top \right) P_{t+1}^{B\top} + P_{t+1}^B \left( \pi_\Delta \lambda_f \right) \hat{s}_\perp^\top + \hat{s}_\perp \left( \lambda_f \pi_\Delta^\top \right) P_{t+1}^{B\top} \\
 &= \sigma_B^{t+1} I + P_{t+1}^B \Lambda_B^{t+1} P_{t+1}^{B\top}
 \end{aligned} \tag{241}$$

UPDATE  $P^{\nabla\nabla}$ :

$$\begin{aligned}
 P_{t+1}^{\nabla\nabla} &= \left( \sigma_{\nabla-} - \sigma_{\nabla-}^2 \sigma_{G-1} \right) I + P_- \left( \Lambda_{\nabla-} - \left[ \sigma_{\nabla-}^2 \Lambda_{G-1} + 2\sigma_{\nabla-} \sigma_{G-1} \Lambda_{\nabla-} \right. \right. \\
 &\quad \left. \left. + \sigma_{\nabla-} \Lambda_{\nabla-} \Lambda_{G-1} + \sigma_{\nabla-} \Lambda_{G-1} \Lambda_{\nabla-} + \sigma_{G-1} \Lambda_{\nabla-} \Lambda_{\nabla-} + \Lambda_{\nabla-} \Lambda_{G-1} \Lambda_{\nabla-} \right] \right) P_-^\top \\
 &= \sigma_{\nabla-}^{t+1} I + P_- \Lambda_{\nabla-}^{t+1} P_-^\top
 \end{aligned} \tag{242}$$

COVARIANCE  $P^{BB}$ : Left hand side (terms without  $U$ ):

$$\begin{aligned}
 \sqrt{\Delta t} V &= \sqrt{\Delta t} \sigma_v I + P \left( \sqrt{\Delta t} \Lambda_V \right) P^\top = \sigma_{L_1} I + P_- \Lambda_{L_1} P_-^\top \\
 W &= \sigma_W I + P \left( \Lambda_W \right) P^\top = \sigma_{L_2} I + P_- \Lambda_{L_2} P_-^\top \\
 VG^{-1}V &= \left( \sigma_V \sigma_V \sigma_{G-1} \right) I + P \left( \sigma_V \sigma_{G-1} \Lambda_V + \sigma_V \sigma_{G-1} \Lambda_V + \sigma_{G-1} \Lambda_V \Lambda_V + \Lambda_V \left( P^\top P_- \right) \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_V \right) P^\top \\
 &\quad + P_- \left( \sigma_V \sigma_V \Lambda_{G-1} \right) P_-^\top + P \left( \sigma_V \Lambda_V \left( P^\top P_- \right) \Lambda_{G-1} \right) P_-^\top + P_- \left( \sigma_V \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_V \right) P^\top \\
 &= \sigma_{L_3} I + P_- \Lambda_{L_3} P_-^\top \\
 WG^{-1}V &= \left( \sigma_V \sigma_W \sigma_{G-1} \right) I + P \left( \sigma_V \sigma_{G-1} \Lambda_W + \sigma_W \sigma_{G-1} \Lambda_V + \sigma_{G-1} \Lambda_W \Lambda_V + \Lambda_W \left( P^\top P_- \right) \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_V \right) P^\top \\
 &\quad + P_- \left( \sigma_V \sigma_W \Lambda_{G-1} \right) P_-^\top + P \left( \sigma_V \Lambda_W \left( P^\top P_- \right) \Lambda_{G-1} \right) P_-^\top + P_- \left( \sigma_W \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_V \right) P^\top \\
 &= \sigma_{L_5} I + P_- \Lambda_{L_5} P_-^\top \\
 VG^{-1}W &= \left( \sigma_W \sigma_V \sigma_{G-1} \right) I + P \left( \sigma_W \sigma_{G-1} \Lambda_V + \sigma_V \sigma_{G-1} \Lambda_W + \sigma_{G-1} \Lambda_V \Lambda_W + \Lambda_V \left( P^\top P_- \right) \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_W \right) P^\top \\
 &\quad + P_- \left( \sigma_W \sigma_V \Lambda_{G-1} \right) P_-^\top + P \left( \sigma_W \Lambda_V \left( P^\top P_- \right) \Lambda_{G-1} \right) P_-^\top + P_- \left( \sigma_V \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_W \right) P^\top \\
 &= \sigma_{L_9} I + P_- \Lambda_{L_9} P_-^\top \\
 WG^{-1}W &= \left( \sigma_W \sigma_W \sigma_{G-1} \right) I + P \left( \sigma_W \sigma_{G-1} \Lambda_W + \sigma_W \sigma_{G-1} \Lambda_W + \sigma_{G-1} \Lambda_W \Lambda_W + \Lambda_W \left( P^\top P_- \right) \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_W \right) P^\top \\
 &\quad + P_- \left( \sigma_W \sigma_W \Lambda_{G-1} \right) P_-^\top + P \left( \sigma_W \Lambda_W \left( P^\top P_- \right) \Lambda_{G-1} \right) P_-^\top + P_- \left( \sigma_W \Lambda_{G-1} \left( P_-^\top P \right) \Lambda_W \right) P^\top \\
 &= \sigma_{L_{11}} I + P_- \Lambda_{L_{11}} P_-^\top
 \end{aligned}$$

Left hand side (terms including  $U$ ):

$$\begin{aligned}
UG^{-1}V &= (\sigma_V\sigma_U\sigma_{G-1})I + P(\sigma_V\sigma_{G-1}\Lambda_U + \sigma_U\sigma_{G-1}\Lambda_V + \sigma_{G-1}\Lambda_U\Lambda_V + \Lambda_U(P^\top P_-)\Lambda_{G-1}(P_-^\top P)\Lambda_V)P^\top \\
&\quad + P_-(\sigma_V\sigma_U\Lambda_{G-1})P_-^\top + P(\sigma_V\Lambda_U(P^\top P_-)\Lambda_{G-1})P_-^\top + P_-(\sigma_U\Lambda_{G-1}(P_-^\top P)\Lambda_V)P^\top \\
&= \sigma_{L_4}I + P_-\Lambda_{L_4}P_-^\top \\
VG^{-1}U^\top &= (\sigma_U\sigma_V\sigma_{G-1})I + P(\sigma_U\sigma_{G-1}\Lambda_V + \sigma_V\sigma_{G-1}\Lambda_U^\top + \sigma_{G-1}\Lambda_V\Lambda_U^\top + \Lambda_V(P^\top P_-)\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&\quad + P_-(\sigma_U\sigma_V\Lambda_{G-1})P_-^\top + P(\sigma_U\Lambda_V(P^\top P_-)\Lambda_{G-1})P_-^\top + P_-(\sigma_V\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&= \sigma_{L_6}I + P_-\Lambda_{L_6}P_-^\top \\
UG^{-1}U^\top &= (\sigma_U\sigma_U\sigma_{G-1})I + P(\sigma_U\sigma_{G-1}\Lambda_U + \sigma_U\sigma_{G-1}\Lambda_U^\top + \sigma_{G-1}\Lambda_U\Lambda_U^\top + \Lambda_U(P^\top P_-)\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&\quad + P_-(\sigma_U\sigma_U\Lambda_{G-1})P_-^\top + P(\sigma_U\Lambda_U(P^\top P_-)\Lambda_{G-1})P_-^\top + P_-(\sigma_U\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&= \sigma_{L_7}I + P_-\Lambda_{L_7}P_-^\top \\
WG^{-1}U^\top &= (\sigma_U\sigma_W\sigma_{G-1})I + P(\sigma_U\sigma_{G-1}\Lambda_W + \sigma_W\sigma_{G-1}\Lambda_U^\top + \sigma_{G-1}\Lambda_W\Lambda_U^\top + \Lambda_W(P^\top P_-)\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&\quad + P_-(\sigma_U\sigma_W\Lambda_{G-1})P_-^\top + P(\sigma_U\Lambda_W(P^\top P_-)\Lambda_{G-1})P_-^\top + P_-(\sigma_W\Lambda_{G-1}(P_-^\top P)\Lambda_U^\top)P^\top \\
&= \sigma_{L_8}I + P_-\Lambda_{L_8}P_-^\top \\
UG^{-1}W &= (\sigma_W\sigma_U\sigma_{G-1})I + P(\sigma_W\sigma_{G-1}\Lambda_U + \sigma_U\sigma_{G-1}\Lambda_W + \sigma_{G-1}\Lambda_U\Lambda_W + \Lambda_U(P^\top P_-)\Lambda_{G-1}(P_-^\top P)\Lambda_W)P^\top \\
&\quad + P_-(\sigma_W\sigma_U\Lambda_{G-1})P_-^\top + P(\sigma_W\Lambda_U(P^\top P_-)\Lambda_{G-1})P_-^\top + P_-(\sigma_U\Lambda_{G-1}(P_-^\top P)\Lambda_W)P^\top \\
&= \sigma_{L_{10}}I + P_-\Lambda_{L_{10}}P_-^\top
\end{aligned}$$

Right hand side (terms without  $U$ ):

$$\begin{aligned}
\sqrt{\Delta t}V &= (\sqrt{\Delta t}\sigma_V)I + P(\sqrt{\Delta t}\Lambda_V)P^\top = \sigma_{R_1}I + P_-\Lambda_{R_1}P_-^\top \\
W &= \sigma_W I + P(\Lambda_W)P^\top = \sigma_{R_2}I + P_-\Lambda_{R_2}P_-^\top \\
-(\Delta t/2Vs)(\Delta t/2Vs)^\top &= P(-(\Delta t/2)^2 [\sigma_V\sigma_V\pi_s\pi_s^\top + \sigma_V\Lambda_V\pi_s\pi_s^\top + \sigma_V\pi_s\pi_s^\top\Lambda_V^\top + \Lambda_V\pi_s\pi_s^\top\Lambda_V^\top])P^\top \\
&\quad + \hat{s}_\perp(-(\Delta t/2)^2 \|s_\perp\| [\sigma_V\sigma_V\pi_s^\top + \sigma_V\pi_s^\top\Lambda_V^\top])P^\top + P(-(\Delta t/2)^2 \|s_\perp\| [\sigma_V\sigma_V\pi_s + \sigma_V\Lambda_V\pi_s])\hat{s}_\perp^\top \\
&\quad + \hat{s}_\perp(-(\Delta t/2)^2 \|s_\perp\| \|s_\perp\| \sigma_V\sigma_V)\hat{s}_\perp^\top = \sigma_{R_3}I + P_-\Lambda_{R_3}P_-^\top \\
-(Ws)(\Delta t/2Vs)^\top &= P(-\Delta t/2 [\sigma_W\sigma_V\pi_s\pi_s^\top + \sigma_V\Lambda_W\pi_s\pi_s^\top + \sigma_W\pi_s\pi_s^\top\Lambda_V^\top + \Lambda_W\pi_s\pi_s^\top\Lambda_V^\top])P^\top \\
&\quad + \hat{s}_\perp(-\Delta t/2 \|s_\perp\| [\sigma_W\sigma_V\pi_s^\top + \sigma_W\pi_s^\top\Lambda_V^\top])P^\top + P(-\Delta t/2 \|s_\perp\| [\sigma_W\sigma_V\pi_s + \sigma_V\Lambda_W\pi_s])\hat{s}_\perp^\top \\
&\quad + \hat{s}_\perp(-\Delta t/2 \|s_\perp\| \|s_\perp\| \sigma_W\sigma_V)\hat{s}_\perp^\top = \sigma_{R_5}I + P_-\Lambda_{R_5}P_-^\top \\
-(\Delta t/2Vs)(Ws)^\top &= P(-\Delta t/2 [\sigma_V\sigma_W\pi_s\pi_s^\top + \sigma_W\Lambda_V\pi_s\pi_s^\top + \sigma_V\pi_s\pi_s^\top\Lambda_W^\top + \Lambda_V\pi_s\pi_s^\top\Lambda_W^\top])P^\top \\
&\quad + \hat{s}_\perp(-\Delta t/2 \|s_\perp\| [\sigma_V\sigma_W\pi_s^\top + \sigma_V\pi_s^\top\Lambda_W^\top])P^\top + P(-\Delta t/2 \|s_\perp\| [\sigma_V\sigma_W\pi_s + \sigma_W\Lambda_V\pi_s])\hat{s}_\perp^\top \\
&\quad + \hat{s}_\perp(-\Delta t/2 \|s_\perp\| \|s_\perp\| \sigma_V\sigma_W)\hat{s}_\perp^\top = \sigma_{R_9}I + P_-\Lambda_{R_9}P_-^\top \\
-(Ws)(Ws)^\top &= P(-[\sigma_W\sigma_W\pi_s\pi_s^\top + \sigma_W\Lambda_W\pi_s\pi_s^\top + \sigma_W\pi_s\pi_s^\top\Lambda_W^\top + \Lambda_W\pi_s\pi_s^\top\Lambda_W^\top])P^\top \\
&\quad + \hat{s}_\perp(-\|s_\perp\| [\sigma_W\sigma_W\pi_s^\top + \sigma_W\pi_s^\top\Lambda_W^\top])P^\top + P(-\|s_\perp\| [\sigma_W\sigma_W\pi_s + \sigma_W\Lambda_W\pi_s])\hat{s}_\perp^\top \\
&\quad + \hat{s}_\perp(-\|s_\perp\| \|s_\perp\| \sigma_W\sigma_W)\hat{s}_\perp^\top = \sigma_{R_{11}}I + P_-\Lambda_{R_{11}}P_-^\top
\end{aligned}$$

Right hand side (terms including  $U$ ):

$$\begin{aligned}
 -(Uv)(\Delta t/2Vs)^\top &= P(-\Delta t/2 [\sigma_U \sigma_V \pi_v \pi_s^\top + \sigma_V \Lambda_U \pi_v \pi_s^\top + \sigma_U \pi_v \pi_s^\top \Lambda_V^\top + \Lambda_U \pi_v \pi_s^\top \Lambda_V^\top]) P^\top \\
 &\quad + P(-\Delta t/2 \|s_\perp\| [\sigma_U \sigma_V \pi_v + \sigma_V \Lambda_U \pi_v]) \hat{s}_\perp^\top = \sigma_{R_4} I + P_- \Lambda_{R_4} P_-^\top \\
 -(\Delta t/2Vs)(Uv)^\top &= P(-\Delta t/2 [\sigma_V \sigma_U \pi_s \pi_v^\top + \sigma_U \Lambda_V \pi_s \pi_v^\top + \sigma_V \pi_s \pi_v^\top \Lambda_U^\top + \Lambda_V \pi_s \pi_v^\top \Lambda_U^\top]) P^\top \\
 &\quad + \hat{s}_\perp (-\Delta t/2 \|s_\perp\| [\sigma_V \sigma_U \pi_v^\top + \sigma_V \pi_v^\top \Lambda_U^\top]) P^\top = \sigma_{R_6} I + P_- \Lambda_{R_6} P_-^\top \\
 -(Uv)(Uv)^\top &= P(- [\sigma_U \sigma_U \pi_v \pi_v^\top + \sigma_U \Lambda_U \pi_v \pi_v^\top + \sigma_U \pi_v \pi_v^\top \Lambda_U^\top + \Lambda_U \pi_v \pi_v^\top \Lambda_U^\top]) P^\top = \sigma_{R_7} I + P_- \Lambda_{R_7} P_-^\top \\
 -(Ws)(Uv)^\top &= P(- [\sigma_W \sigma_U \pi_s \pi_v^\top + \sigma_U \Lambda_W \pi_s \pi_v^\top + \sigma_W \pi_s \pi_v^\top \Lambda_U^\top + \Lambda_W \pi_s \pi_v^\top \Lambda_U^\top]) P^\top \\
 &\quad + \hat{s}_\perp (-\|s_\perp\| [\sigma_W \sigma_U \pi_v^\top + \sigma_W \pi_v^\top \Lambda_U^\top]) P^\top = \sigma_{R_8} I + P_- \Lambda_{R_8} P_-^\top \\
 -(Uv)(Ws)^\top &= P(- [\sigma_U \sigma_W \pi_v \pi_s^\top + \sigma_W \Lambda_U \pi_v \pi_s^\top + \sigma_U \pi_v \pi_s^\top \Lambda_W^\top + \Lambda_U \pi_v \pi_s^\top \Lambda_W^\top]) P^\top \\
 &\quad + P(-\|s_\perp\| [\sigma_U \sigma_W \pi_v + \sigma_W \Lambda_U \pi_v]) \hat{s}_\perp^\top = \sigma_{R_{10}} I + P_- \Lambda_{R_{10}} P_-^\top
 \end{aligned}$$

BACK-PROJECTION OF  $P^{BB}$  ONTO KRONECKER STRUCTURE: Let  $w_{L_1}, \dots, w_{L_{11}}$  be the weights from the Kronecker-approximation of the left hand side as in Eq. 180, and  $w_{R_1}, \dots, w_{R_{11}}$  of the right hand side, then:

$$\begin{aligned}
 A_l &= \left( \sum_{i=1}^{11} w_{L_i} \sigma_{L_i} \right) I + P_- \left( \sum_{i=1}^{11} w_{L_i} \Lambda_{L_i} \right) P_-^\top \\
 A_r &= \left( \sum_{i=1}^{11} w_{R_i} \sigma_{R_i} \right) I + P_- \left( \sum_{i=1}^{11} w_{R_i} \Lambda_{R_i} \right) P_-^\top.
 \end{aligned} \tag{243}$$

The weights  $w_{L_i}$  and  $w_{R_i}$  for  $i = 1, \dots, 11$  are cheap to compute (most notably independent of the dimensionality  $N$ ) since they only involve the smaller matrices  $\Lambda$  and scalars  $\sigma$ . This is because the inner products  $R^\top R$  (here  $R$  is the matrix as in Eq. 172 and not the measurement noise) and  $S^\top S$  (also Eq. 172) are easy to compute when the basis  $P$  is known (see Section B.4.3 for formula). We then find  $W_{t+1}$  such that  $W_{t+1} \otimes W_{t+1} = A_l \otimes A_r$ . This is fast as well since  $A_l$  and  $A_r$  both lie in the span of  $P_-$ . Thus also  $W_{t+1}$  will lie in the span of  $P_-$  and the computation again only involves small  $\Lambda$ s and  $\sigma$ s.

UPDATE  $P^{\nabla B}$ : As mentioned in Section 9.1.2, the back-projection for  $P^{\nabla B}$  is not straightforward in the symmetric hypothesis class, since  $(I \otimes v^\top) \neq (A \otimes v^\top A)$  (right side not even defined). So here we will report the back-projection as if the *non-symmetric(!)* hypothesis class was used. Until this is solved, for the symmetric class,  $P^{\nabla B}$  can be set to zero, such that off diagonal covariance contributions between gradient and Hessian are ignored.<sup>1</sup> The right hand side of Kronecker for non-symmetric Kronecker products is:

<sup>1</sup> This does not mean that the whole covariance  $P_t \in \mathbb{R}^{(N^2+N) \times (N^2+N)}$  is diagonal, since  $P^{\nabla \nabla} \in \mathbb{R}^{N \times N}$  and  $P^{BB} \in \mathbb{R}^{N^2 \times N^2}$  are dense, just that it is block-diagonal.

$$\begin{aligned}
Uv &= \sigma_U Iv + P\Lambda_U P^\top v = P(\sigma_U \pi_v + \Lambda_U \pi_v) \\
&= P_- \pi_3 \\
\Delta t/2Vs &= \Delta t/2\sigma_V Is + P\Delta t/2\Lambda_V P^\top s = P(\Delta t/2\sigma_V \pi_s + \Delta t/2\Lambda_V \pi_s) + \hat{s}_\perp(\Delta t/2\sigma_V \|s_\perp\|) \\
&= P_- \pi_1 \\
Ws &= \sigma_W Is + P\Lambda_W P^\top s = P(\sigma_W \pi_s + \Lambda_W \pi_s) + \hat{s}_\perp(\sigma_W \|s_\perp\|) \\
&= P_- \pi_2
\end{aligned} \tag{244}$$

Left hand side of Kronecker:

$$\begin{aligned}
U - UG^{-1}P_{t+1-}^{\nabla\nabla} &= (\sigma_U - \sigma_U \sigma_{G-1} \sigma_{\nabla-})I \\
&+ P(\Lambda_U - \sigma_{\nabla-} \sigma_{G-1} \Lambda_U)P^\top \\
&+ P_-(-[\sigma_{\nabla-} \sigma_U \Lambda_{G-1} + \sigma_U \sigma_{G-1} \Lambda_{\nabla-} + \sigma_U \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&+ P(-[\sigma_{\nabla-} \Lambda_U (P^\top P_-) \Lambda_{G-1} + \sigma_{G-1} \Lambda_U (P^\top P_-) \Lambda_{\nabla-} + \Lambda_U (P^\top P_-) \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&= \sigma_1 I + P_- \Lambda_1 P_-^\top \\
V - VG^{-1}P_{t+1-}^{\nabla\nabla} &= (\sigma_V - \sigma_V \sigma_{G-1} \sigma_{\nabla-})I \\
&+ P(\Lambda_V - \sigma_{\nabla-} \sigma_{G-1} \Lambda_V)P^\top \\
&+ P_-(-[\sigma_{\nabla-} \sigma_V \Lambda_{G-1} + \sigma_V \sigma_{G-1} \Lambda_{\nabla-} + \sigma_V \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&+ P(-[\sigma_{\nabla-} \Lambda_V (P^\top P_-) \Lambda_{G-1} + \sigma_{G-1} \Lambda_V (P^\top P_-) \Lambda_{\nabla-} + \Lambda_V (P^\top P_-) \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&= \sigma_2 I + P_- \Lambda_2 P_-^\top \\
W - WG^{-1}P_{t+1-}^{\nabla\nabla} &= (\sigma_W - \sigma_W \sigma_{G-1} \sigma_{\nabla-})I \\
&+ P(\Lambda_W - \sigma_{\nabla-} \sigma_{G-1} \Lambda_W)P^\top \\
&+ P_-(-[\sigma_{\nabla-} \sigma_W \Lambda_{G-1} + \sigma_W \sigma_{G-1} \Lambda_{\nabla-} + \sigma_W \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&+ P(-[\sigma_{\nabla-} \Lambda_W (P^\top P_-) \Lambda_{G-1} + \sigma_{G-1} \Lambda_W (P^\top P_-) \Lambda_{\nabla-} + \Lambda_W (P^\top P_-) \Lambda_{G-1} \Lambda_{\nabla-}])P_-^\top \\
&= \sigma_3 I + P_- \Lambda_3 P_-^\top
\end{aligned} \tag{245}$$

BACK-PROJECTION  $P^{\nabla B}$ : If  $w_1$ ,  $w_2$  and  $w_3$  denote the weights found by the Kronecker approximation, we see that  $U_{t+1}$  is spanned by  $P_-$ . Then:

$$\begin{aligned}
U_{t+1} &= (w_1 \sigma_1 + w_2 \sigma_2 + w_3 \sigma_3)I + P_-(w_1 \Lambda_1 + w_2 \Lambda_2 + w_3 \Lambda_3)P_-^\top \\
&= \sigma_U^{t+1} + P_- \Lambda_U^{t+1} P_-^\top \\
U_{t+1}^{-1} &= \sigma_{U_{t+1}}^{-1} + P_-([\sigma_{U_{t+1}}^{-2} (\Lambda_U^{t+1,-1} + \sigma_{U_{t+1}}^{-1})])^{-1} P_-^\top \\
&= \sigma_{U-1}^{t+1} I + P_- \Lambda_{U-1}^{t+1} P_- \\
U_{t+1}^{-1} v_{t+1} &= P(\sigma_V w_V \Delta t/2 \pi_s + w_V \Delta t/2 \Lambda_V \pi_s + w_U \sigma_U \pi_v + w_U \Lambda_U \pi_v + w_W \sigma_W \pi_s \\
&\quad + w_W \Lambda_W \pi_s) + \hat{s}_\perp(\|s_\perp\| [w_V \Delta t/2 \sigma_V + w_W \sigma_W]) \\
&= P_- \tilde{\pi}_v \\
v_{t+1} &= P_-(\sigma_{U-1}^{t+1} \tilde{\pi}_v + \Lambda_{U-1}^{t+1} \tilde{\pi}_v) = P_- \pi_v.
\end{aligned}$$



## B.4.3 Gram-Schmidt &amp; Matrix Algebra:

**LOW-RANK MATRIX ALGEBRA:** Let  $A = \sigma_A I + P\Lambda_A P^\top$  and  $B = \sigma_B I + P\Lambda_B P^\top$  with  $P^\top P = I$ , and let  $\Lambda_A, \Lambda_B \in \mathbb{R}^{M \times M}$  and  $P \in \mathbb{R}^{N \times M}$  with  $M \ll N$  then:

$$A + B = (\sigma_A + \sigma_B)I + P(\Lambda_A + \Lambda_B)P^\top \quad (246a)$$

$$AB = (\sigma_A + \sigma_B)I + P(\sigma_A \Lambda_B + \sigma_B \Lambda_A + \Lambda_A \Lambda_B)P^\top \quad (246b)$$

$$A^{-1} = \sigma_A^{-1}I + P(-\sigma_A^{-2}[\Lambda_A^{-1} + \sigma_A^{-1}I_M]^{-1})P^\top \quad (246c)$$

$$\vec{A}^\top \vec{B} = N\sigma_A \sigma_B + \sigma_A \text{tr}[\Lambda_B] + \sigma_B \text{tr}[\Lambda_A] + \text{tr}[\Lambda_A^\top \Lambda_B] \quad (246d)$$

Especially Eq. 246d reduces the cost for the Kronecker back-projections drastically.

**POSITIVE DEFINITE PROJECTION:** Let  $C = \sigma I + P\Lambda P^\top$  be symmetric ( $\Lambda = \Lambda^\top$ ) with  $M$  non-zero eigenvalues  $\lambda_1, \dots, \lambda_M$  and  $\sigma > 0$ , then flipping the sign of negative eigenvalues involves:

$\tilde{Q}D\tilde{Q}^\top = \Lambda$  eigen-decomposition with elements  $l_i = D_{ii}$  eigen-values, then

$\lambda_i = \sigma + l_i$ . Therefore for *negative*  $\lambda_k$  (not negative  $l_k$ )

$$l_k^{\text{pos}} = -l_k - 2\sigma \quad \text{since} \quad \lambda_k^{\text{pos}} = \sigma + (-l_k - 2\sigma) = -\sigma - l_k = -\lambda_k \quad \text{and thus} \quad (247)$$

$$\Lambda^{\text{pos}} = \tilde{Q}D^{\text{pos}}\tilde{Q}^\top \quad \text{where} \quad l_k \rightarrow l_k^{\text{pos}} \text{ in } D.$$

The resulting matrix is of form  $C^{\text{pos}} = \sigma I + P\Lambda^{\text{pos}}P^\top$  and has eigen-values  $\lambda_i^{\text{pos}} = |\lambda_i|$ .

**GRAM-SCHMIDT:** Some explanation on how to get  $\pi_{p_Y}$ .  $\hat{p}_i^Y$  denotes the  $i^{\text{th}}$  column of the  $N \times M$  matrix  $\hat{P}^Y$ .  $P^{Y+}$  denotes the  $N \times (M+1)$  matrix  $[\hat{s}_\perp, \hat{P}^Y]$

$$\hat{p}_i^Y = \frac{P_{\text{old},i}^Y - \sum_{j=1}^i \frac{(\hat{p}_j^{Y+\top} P_{\text{old},i}^Y)}{\hat{p}_j^{Y+\top} \hat{p}_j^{Y+}} \hat{p}_j^{Y+}}{\|P_{\text{old},i}^Y - \sum_{j=1}^i \frac{(\hat{p}_j^{Y+\top} P_{\text{old},i}^Y)}{\hat{p}_j^{Y+\top} \hat{p}_j^{Y+}} \hat{p}_j^{Y+}\|} \quad (248)$$

Note that the sum in Eq. 248 runs to  $i$  instead of  $i-1$  since we added  $\hat{s}_\perp$  to  $\hat{P}^{Y+}$ . Now use that  $\hat{P}^Y$  are orthonormal  $\hat{p}_j^{Y+\top} \hat{p}_j^{Y+} = 1$  and define  $\tilde{n}_{ji} = \hat{p}_j^{Y+\top} P_{\text{old},i}^Y$ .  $\tilde{n}$  will be an  $(M+1) \times M$  matrix where the first row is the projection on  $\hat{s}_\perp$ .

$$\hat{p}_i^Y = \frac{P_{\text{old},i}^Y - \sum_{j=1}^i \tilde{n}_{ji} \hat{p}_j^{Y+}}{\|P_{\text{old},i}^Y - \sum_{j=1}^i \tilde{n}_{ji} \hat{p}_j^{Y+}\|} \quad (249)$$

Now define  $c_{ii}^Y = \|P_{\text{old},i}^Y - \sum_{j=1}^i \tilde{n}_{ji} \hat{P}_j^{Y+}\|$ .  $c^Y$  will be an  $M \times M$  matrix.

$$\hat{P}_i^Y c_{ii}^Y = P_{\text{old},i}^Y - \sum_{j=1}^i \tilde{n}_{ji} \hat{P}_j^{Y+} \quad (250)$$

If the remaining elements of  $c^Y$  and  $\tilde{n}$  are filled with zeros, then after the complete loop (computation for all  $i$ ) is done we can write:

$$\begin{aligned} \hat{P}^Y c^Y &= P_{\text{old}}^Y - \hat{P}^{Y+} \tilde{n} \\ &= P_{\text{old}}^Y - \hat{P}^Y \tilde{n}_{(2:\text{end},:)} - \hat{s}_{\perp} \tilde{n}_{(1,:)} \\ P_{\text{old}}^Y &= \hat{P}^Y n^Y + \hat{P}^Y \tilde{n}_{(2:\text{end},:)} + \hat{s}_{\perp} \tilde{n}_{(1,:)} \\ &= \hat{P}^Y (c^Y + \tilde{n}_{(2:\text{end},:)}) + \hat{s}_{\perp} \tilde{n}_{(1,:)} \\ &= \hat{P}^Y n^Y + \hat{s}_{\perp} \pi_{p^Y}. \end{aligned} \quad (251)$$

#### B.4.4 Hyper-Parameter Adaptation (Second-Order, Symmetric)

The observation noise  $R_t$  can again be estimated within the mini-batch using  $\hat{\Sigma}$ . The difference now to filtering on gradients, is that  $R_t$  must be of form scalar-plus-low rank instead of being diagonal (the same holds for the hyper-parameters  $P_0^{\nabla\nabla}$  and  $V$ ). For a start, it is the simplest, to drop the low rank term of  $R_t$  and set  $\sigma_t^R = \frac{1}{|S|N} \sum_i \hat{\Sigma}_i$  to the mean of the estimated gradient variances.<sup>2</sup>

We also drop the low rank part of  $V$ , and just adapt a scalar  $\sigma_V$  with  $V = \sigma_V I$ . The maximum marginal likelihood estimator is not analytic anymore, but we can always compute the first (and perhaps second) derivative of  $\log p(y_t)$  and perform a, or multiple gradient descent, or Newton steps on it. The (logarithmic) marginal likelihood of parameter  $\sigma_V$ , given the current datapoint  $y_{t+1}$  is:

$$\begin{aligned} p(y_{t+1}) &= \int p(y_{t+1}|x_{t+1}, y_1, \dots, y_t) p(x_{t+1}|y_1, \dots, y_t) dx_{t+1} \\ &= \int \mathcal{N}(y_{t+1}; x_{t+1}^{\nabla}, R_{t+1}) \mathcal{N}(x_{t+1}; m_{t+1-}, P_{t+1-}) dx_{t+1} \\ &= \mathcal{N}(y_{t+1}; m_{t+1-}^{\nabla}, P_{t+1-}^{\nabla\nabla} + R_{t+1}) \\ &= \frac{e^{-\frac{1}{2}(y_{t+1} - m_{t+1-}^{\nabla})^{\top} (P_{t+1-}^{\nabla\nabla} + R_{t+1})^{-1} (y_{t+1} - m_{t+1-}^{\nabla})}}{(2\pi)^{N/2} |P_{t+1-}^{\nabla\nabla} + R_{t+1}|^{1/2}} \end{aligned} \quad (252)$$

$$2 \log p(y_{t+1}) \propto -\log |P_{t+1-}^{\nabla\nabla} + R_{t+1}| - (y_{t+1} - m_{t+1-}^{\nabla})^{\top} (P_{t+1-}^{\nabla\nabla} + R_{t+1})^{-1} (y_{t+1} - m_{t+1-}^{\nabla})$$

FIRST DERIVATIVE: Some matrix identities which we will need to compute the derivative of Eq. (252) are:

$$\frac{\partial a^{\top} X b}{\partial z} = a^{\top} \frac{\partial X}{\partial z} b, \quad \frac{\partial X^{-1}}{\partial z} = -X^{-1} \frac{\partial X}{\partial z} X^{-1}, \quad \frac{\partial \log |X(z)|}{\partial z} = \text{tr} \left[ X^{-1} \frac{\partial X}{\partial z} \right] \quad (253)$$

<sup>2</sup> Alternatively one could assume a block-diagonal Hessian where correlations between layers are dropped, and then estimate one  $\sigma_t^R$  per layer.

Where  $a$  and  $b$  are vectors and  $X$  is a matrix.  $V$  is hidden in  $P^{\nabla\nabla-}$  only. The derivative of Eq. (252) with respect to the scalar  $\sigma_V$  yields:

$$\begin{aligned} -\frac{\partial}{\partial\sigma_V} 2\log p(y) &= (y - m^{\nabla-})^\top \frac{\partial(P^{\nabla\nabla-} + R)^{-1}}{\partial\sigma_V} (y - m^{\nabla-}) + 2\operatorname{tr} \left[ (P^{\nabla\nabla-} + R)^{-1} \frac{\partial(P^{\nabla\nabla-} + R)}{\partial\sigma_V} \right] \\ &= -(y - m^{\nabla-})^\top (P^{\nabla\nabla-} + R)^{-1} \frac{\partial(P^{\nabla\nabla-} + R)}{\partial\sigma_V} (P^{\nabla\nabla-} + R)^{-1} (y - m^{\nabla-}) + 2\operatorname{tr} \left[ (P^{\nabla\nabla-} + R)^{-1} \frac{\partial(P^{\nabla\nabla-} + R)}{\partial\sigma_V} \right] \end{aligned} \quad (254)$$

Thus, we need  $\partial a_{\bar{V}}/\partial\sigma_V = \sqrt{\Delta t/3}(s^\top s)$  and  $\partial\sigma_{\bar{V}}/\partial\sigma_V = \sqrt{\Delta t/3}$ . We get:

$$\begin{aligned} \frac{\partial(P^{\nabla\nabla-} + R)}{\partial\sigma_V} &= \left( \frac{1}{2} a_{\bar{V}} \frac{\partial\sigma_{\bar{V}}}{\partial\sigma_V} + \frac{1}{2} \sigma_{\bar{V}} \frac{\partial a_{\bar{V}}}{\partial\sigma_V} \right) I + P \left( \frac{1}{2} 2\sigma_{\bar{V}} \frac{\partial\sigma_{\bar{V}}}{\partial\sigma_V} \pi_s \pi_s^\top \right) P^\top \\ &\quad + P \left( \frac{1}{2} \|s_\perp\| 2\sigma_{\bar{V}} \frac{\partial\sigma_{\bar{V}}}{\partial\sigma_V} \pi_s \right) \hat{s}_\perp^\top + \hat{s}_\perp \left( \frac{1}{2} \|s_\perp\| 2\sigma_{\bar{V}} \frac{\partial\sigma_{\bar{V}}}{\partial\sigma_V} \pi_s^\top \right) P^\top + \hat{s}_\perp \left( \frac{1}{2} \|s_\perp\|^2 2\sigma_{\bar{V}} \frac{\partial\sigma_{\bar{V}}}{\partial\sigma_V} \right) \hat{s}_\perp^\top \\ &= \frac{\Delta t}{3} \sigma_V \left[ (\pi_s^\top \pi_s + \|s_\perp\|^2) I + P (\pi_s \pi_s^\top) P^\top + P (\|s_\perp\| \pi_s) \hat{s}_\perp^\top + \hat{s}_\perp (\|s_\perp\| \pi_s^\top) P^\top + \hat{s}_\perp (\|s_\perp\|^2) \hat{s}_\perp^\top \right] \\ &= \sigma_C I + P_- \Lambda_C P_-^\top = C \end{aligned} \quad (255)$$

Combining Eq. (255) with Eq. (254) gives:

$$\frac{\partial}{\partial\sigma_V} \log p(y) = \frac{1}{2} \Delta^\top G^{-1} C G^{-1} \Delta - \operatorname{tr} [G^{-1} C] \quad (256)$$

Compute 2nd term of Eq. 256:

$$\begin{aligned} G^{-1} C &= (\sigma_{G^{-1}} I + P_- \Lambda_{G^{-1}} P_-^\top) (\sigma_C I + P_- \Lambda_C P_-^\top) \\ &= (\sigma_{G^{-1}} \sigma_C) I + P_- (\sigma_C \Lambda_{G^{-1}} + \sigma_{G^{-1}} \Lambda_C + \Lambda_{G^{-1}} \Lambda_C) P_-^\top \\ &=: \sigma_{GC} I + P_- \Lambda_{GC} P_-^\top \end{aligned} \quad (257)$$

$$\operatorname{tr} [G^{-1} C] = N \sigma_{GC} + \operatorname{tr} [\Lambda_{GC}]$$

and 1st term:

$$\begin{aligned} G^{-1} C G^{-1} &= (\sigma_{GC} I + P_- (\Lambda_{GC}) P_-^\top) (\sigma_{G^{-1}} I + P_- \Lambda_{G^{-1}} P_-^\top) \\ &= \frac{1}{2} \sigma_{GC} \sigma_{G^{-1}} \pi_\Delta^\top \pi_\Delta + \frac{1}{2} \pi_\Delta^\top (P_{t+1}^{B\top} P_-) (\sigma_{G^{-1}} \Lambda_{GC} + \sigma_{GC} \Lambda_{G^{-1}} + \Lambda_{GC} \Lambda_{G^{-1}}) (P_{t+1}^{B\top} P_-)^\top \pi_\Delta \end{aligned} \quad (258)$$

Here  $P_{t+1}^{B\top} P_- = [I_{M+1}; 0_{M+1}] \in \mathbb{R}^{2(M+1) \times (M+1)}$ . Thus, the gradient is:

$$\begin{aligned} \frac{\partial}{\partial\sigma_V} \log p(y) &= \frac{1}{2} \sigma_{GC} \sigma_{G^{-1}} \pi_\Delta^\top \pi_\Delta + \frac{1}{2} \pi_\Delta^\top (P_{t+1}^{B\top} P_-) (\sigma_{G^{-1}} \Lambda_{GC} + \sigma_{GC} \Lambda_{G^{-1}} + \Lambda_{GC} \Lambda_{G^{-1}}) (P_{t+1}^{B\top} P_-)^\top \pi_\Delta \\ &\quad - N \sigma_{GC} - \operatorname{tr} [\Lambda_{GC}], \end{aligned} \quad (259)$$

or

$$\begin{aligned} \frac{\partial}{\partial\sigma_V} \log p(y) &= \frac{\Delta t}{3} \sigma_V \left[ \frac{1}{2} (s^\top s) (\pi_\Delta^\top \pi_\Delta) \sigma_{G^{-1}}^2 + \frac{1}{2} \pi_\Delta^\top (P_{t+1}^{B\top} P_-) \tilde{C} (P_{t+1}^{B\top} P_-)^\top \pi_\Delta \right. \\ &\quad \left. - N (s^\top s) \sigma_{G^{-1}} - \operatorname{tr} [(s^\top s) \Lambda_{G^{-1}} + \Lambda_C \sigma_{G^{-1}} + \Lambda_{G^{-1}} \Lambda_C] \right] \\ &=: \frac{\Delta t}{3} \sigma_V \tilde{D} \end{aligned} \quad (260)$$

with

$$\tilde{C} := -2(s^\top s)\sigma_{G-1}\Lambda_{G-1} + (s^\top s)\Lambda_{G-1}\Lambda_{G-1} + \sigma_{G-1}^{-2}\Lambda_{G-1}\Lambda_{\nabla-}^{-1}\Lambda_C\Lambda_{\nabla-}^{-1}\Lambda_{G-1}. \quad (261)$$

We could then update  $\sigma_V$  at each iteration with a, or multiple gradient steps with step size  $\alpha$  according to:

$$\sigma_V = \sigma_V - \alpha \frac{\partial}{\partial \sigma_V} \log p(y), \quad (262)$$

with  $\partial \log p(y) / \partial \sigma_V$  as in Eq. 259 or Eq. 260.

SECOND DERIVATIVE: The second partial derivative of Eq. 260 is:

$$\begin{aligned} \frac{\partial^2}{\partial^2 \sigma_V} \log p(y) &= \frac{1}{2} \frac{\Delta t}{3} (s^\top s) (\pi_\Delta^\top \pi_\Delta) \frac{\partial}{\partial \sigma_V} \left( \sigma_{G-1}^2 \sigma_V \right) + \frac{1}{2} \frac{\Delta t}{3} \pi_\Delta^\top (P_{t+1}^{B\top} P_-) \frac{\partial}{\partial \sigma_V} (\sigma_V \tilde{C}) (P_{t+1}^{B\top} P_-)^\top \pi_\Delta \\ &- N \frac{\Delta t}{3} (s^\top s) \frac{\partial}{\partial \sigma_V} (\sigma_V \sigma_{G-1}) - \frac{\partial}{\partial \sigma_V} \left( \text{tr} \left[ \frac{\Delta t}{3} (s^\top s) \sigma_V \Lambda_{G-1} + \frac{\Delta t}{3} \Lambda_C \sigma_V \sigma_{G-1} + \frac{\Delta t}{3} \sigma_V \Lambda_{G-1} \Lambda_C \right] \right). \end{aligned} \quad (263)$$

We need:

$$\begin{aligned} \frac{\partial}{\partial \sigma_V} \left( \sigma_{G-1}^2 \sigma_V \right) &= \sigma_{G-1}^2 \frac{\partial \sigma_V}{\partial \sigma_V} + 2\sigma_V \sigma_{G-1} \frac{\partial \sigma_{G-1}}{\partial \sigma_V} = \sigma_{G-1}^2 - 2 \frac{\Delta t}{3} \sigma_V^2 \sigma_{G-1}^3 (s^\top s) \\ \frac{\partial}{\partial \sigma_V} (\sigma_{G-1} \sigma_V) &= \sigma_{G-1} \frac{\partial \sigma_V}{\partial \sigma_V} + \sigma_V \frac{\partial \sigma_{G-1}}{\partial \sigma_V} = \sigma_{G-1} - \sigma_V \frac{\Delta t}{3} \sigma_{G-1}^2 \sigma_V (s^\top s). \end{aligned} \quad (264)$$

We also need the derivative of  $\tilde{C}$ . Use  $\partial(XY)/\partial z = X\partial Y/\partial z + \partial X/\partial z Y$ , then:

$$\begin{aligned} \frac{\partial \tilde{C}}{\partial \sigma_V} &= 2(s^\top s) \frac{\partial}{\partial \sigma_V} (\sigma_{G-1} \Lambda_{G-1}) + (s^\top s) \frac{\partial}{\partial \sigma_V} (\Lambda_{G-1} \Lambda_{G-1}) + \frac{\partial}{\partial \sigma_V} \left( \sigma_{G-1}^{-2} \Lambda_{G-1} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1} \right) \\ &= \Lambda_{G-1} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \frac{\partial \Lambda_{G-1}}{\partial \sigma_V} + \Lambda_{G-1} \Lambda_{\nabla-}^{-1} \Lambda_C \frac{\partial \Lambda_{\nabla-}^{-1}}{\partial \sigma_V} \Lambda_{G-1} + \Lambda_{G-1} \frac{\partial \Lambda_{\nabla-}^{-1}}{\partial \sigma_V} \Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1} \\ &+ \frac{\partial \Lambda_{G-1}}{\partial \sigma_V} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1}. \end{aligned} \quad (265)$$

Therefore  $\partial \tilde{C} / \partial \sigma_V$  becomes:

$$\begin{aligned} \frac{\partial \tilde{C}}{\partial \sigma_V} &= -\sigma_V \frac{\Delta t}{3} \left[ 2(s^\top s) \sigma_{G-1} \tilde{C} + 2(s^\top s) (s^\top s) \sigma_{G-1}^2 \Lambda_{G-1} + (s^\top s) \Lambda_{G-1} \tilde{C} + (s^\top s) \tilde{C} \Lambda_{G-1} \right. \\ &+ \sigma_{G-1}^{-2} \Lambda_{G-1} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \left( \tilde{C} + 2\Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1} \right) + \sigma_{G-1}^{-2} \tilde{C} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1} \\ &\left. + 2(s^\top s) \sigma_{G-1}^3 \Lambda_{G-1} \Lambda_{\nabla-}^{-1} \Lambda_C \Lambda_{\nabla-}^{-1} \Lambda_{G-1} \right] \\ &=: -\sigma_V \frac{\Delta t}{3} \tilde{C}^\partial. \end{aligned} \quad (266)$$

For the last term in Eq. (263) we need to compute:

$$\begin{aligned}
 & \frac{\partial}{\partial \sigma_V} \left( \text{tr} \left[ \frac{\Delta t}{3} (s^\top s) \sigma_V \Lambda_{G-1} + \frac{\Delta t}{3} \Lambda_C \sigma_V \sigma_{G-1} + \frac{\Delta t}{3} \sigma_V \Lambda_{G-1} \Lambda_C \right] \right) \\
 &= \text{tr} \left[ \frac{\Delta t}{3} (s^\top s) \frac{\partial}{\partial \sigma_V} (\sigma_V \Lambda_{G-1}) + \frac{\Delta t}{3} \Lambda_C \frac{\partial}{\partial \sigma_V} (\sigma_V \sigma_{G-1}) + \frac{\Delta t}{3} \frac{\partial}{\partial \sigma_V} (\sigma_V \Lambda_{G-1}) \Lambda_C \right] \\
 &= \text{tr} \left[ - \left( \frac{\Delta t}{3} \sigma_V \right)^2 (s^\top s) \tilde{C} + \frac{\Delta t}{3} (s^\top s) \Lambda_{G-1} - \left( \frac{\Delta t}{3} \sigma_V \sigma_{G-1} \right)^2 (s^\top s) \Lambda_C + \frac{\Delta t}{3} \sigma_{G-1} \Lambda_C \right. \\
 &\quad \left. - \left( \frac{\Delta t}{3} \sigma_V \right)^2 \tilde{C} \Lambda_C + \frac{\Delta t}{3} \Lambda_{G-1} \Lambda_C \right] =: \text{tr} [C_{\text{trace}}].
 \end{aligned} \tag{267}$$

Now we can combine all equations to get the second derivative:

$$\begin{aligned}
 \frac{\partial^2}{\partial^2 \sigma_V} \log p(y) &= \frac{1}{2} \frac{\Delta t}{3} (s^\top s) (\pi_\Delta^\top \pi_\Delta) \left( \sigma_{G-1}^2 - 2 \frac{\Delta t}{3} \sigma_V^2 \sigma_{G-1}^3 (s^\top s) \right) \\
 &\quad + \frac{1}{2} \frac{\Delta t}{3} \pi_\Delta^\top (P_{t+1}^{B\top} P_-) \left( -\sigma_V^2 \frac{\Delta t}{3} \tilde{C}^\partial + \tilde{C} \right) (P_{t+1}^{B\top} P_-)^\top \pi_\Delta \\
 &\quad - N \frac{\Delta t}{3} (s^\top s) \left( \sigma_{G-1} - \frac{\Delta t}{3} \sigma_V^2 \sigma_{G-1}^2 (s^\top s) \right) - \text{tr} [C_{\text{trace}}]
 \end{aligned} \tag{268}$$

The update for one Newton step, given the first and second derivative as in Eqs. 260 and 268, is thus:

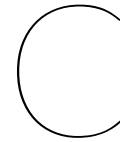
$$\sigma_V = \sigma_V - \alpha \left( \frac{\partial^2}{\partial^2 \sigma_V} \log p(y) \right)^{-1} \frac{\partial}{\partial \sigma_V} \log p(y) \tag{269}$$

with steps size  $\alpha \leq 1, \alpha > 0$ . Thus, at each iteration,  $\sigma_V$  can be updated according to

$$\begin{aligned}
 \sigma_V &= \sigma_V - \tilde{\alpha} \frac{\partial}{\partial \sigma_V} \log p(y) \quad \text{or} \\
 \sigma_V &= \sigma_V - \left( \frac{\partial^2}{\partial^2 \sigma_V} \log p(y) \right)^{-1} \frac{\partial}{\partial \sigma_V} \log p(y)
 \end{aligned} \tag{270}$$

for a gradient descent, or Newton step respectively.





# Additional Experimental Results for PROBLS

## C.1 Noise Sensitivity

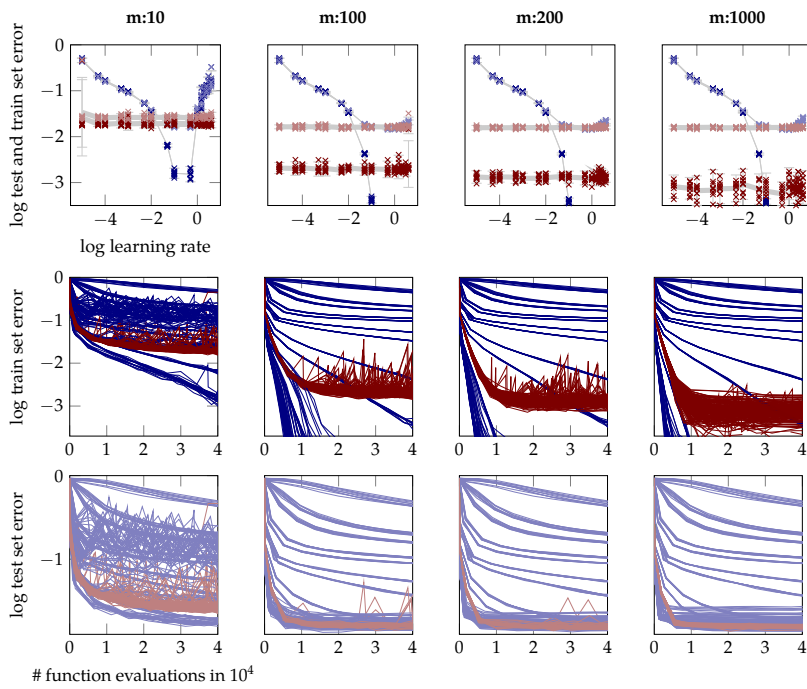


Figure 60: Performance of N-I on MNIST for varying mini-batch sizes: Plots and colors same as in Figure 44 (middle plots cropped for readability).

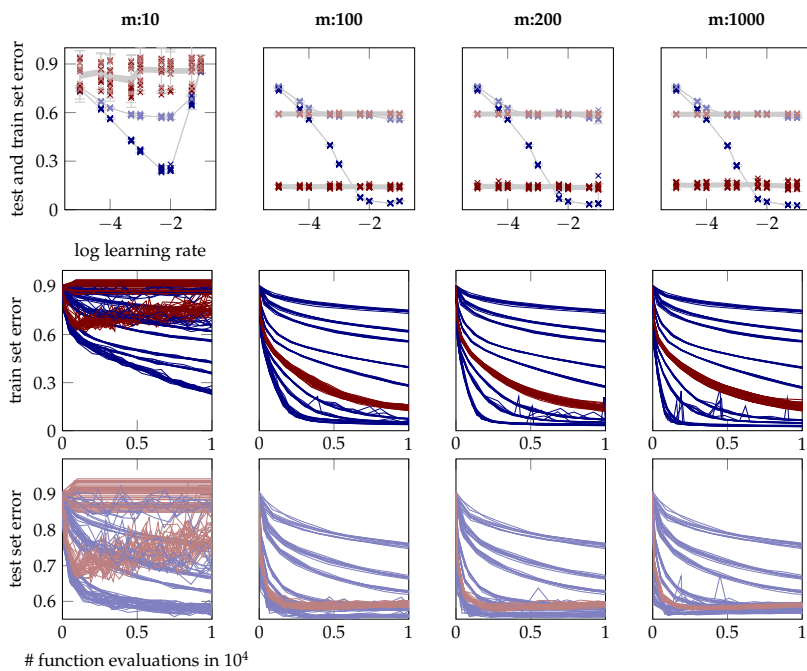


Figure 61: Performance of N-II on CIFAR-10 for varying mini-batch sizes: Plots and colors same as in Figure 44, except the scaling of the y-axis which is not logarithmic here.

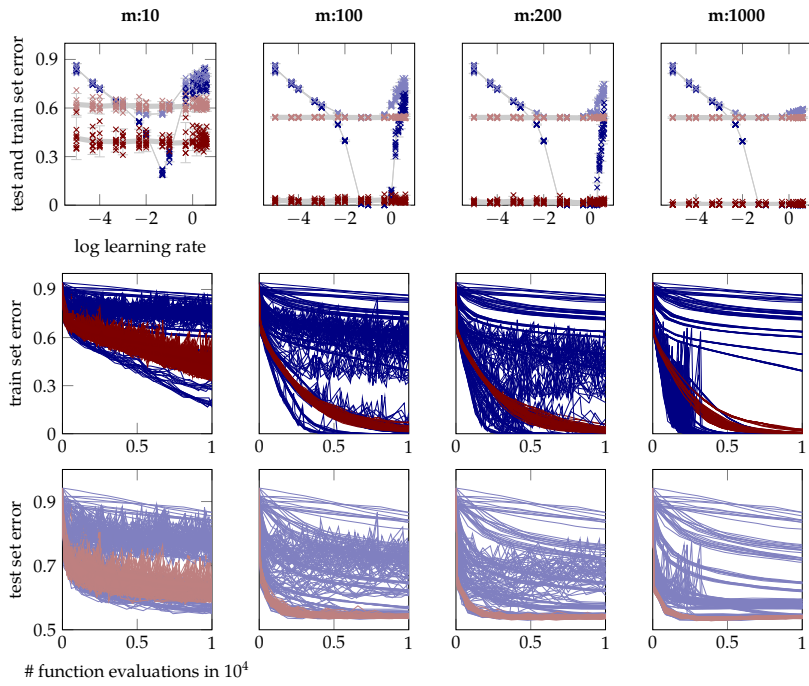


Figure 62: Performance of N-I on CIFAR-10 for varying mini-batch sizes: Plots and colors same as in Figure 61.

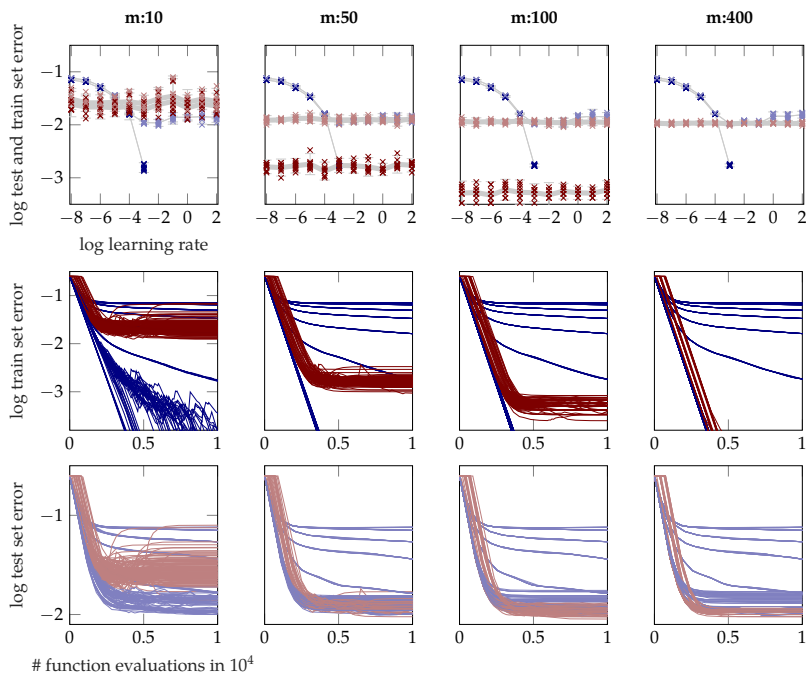


Figure 63: Performance of N-III on GISETTE for varying mini-batch sizes: Plots and colors same as in Figure 61.



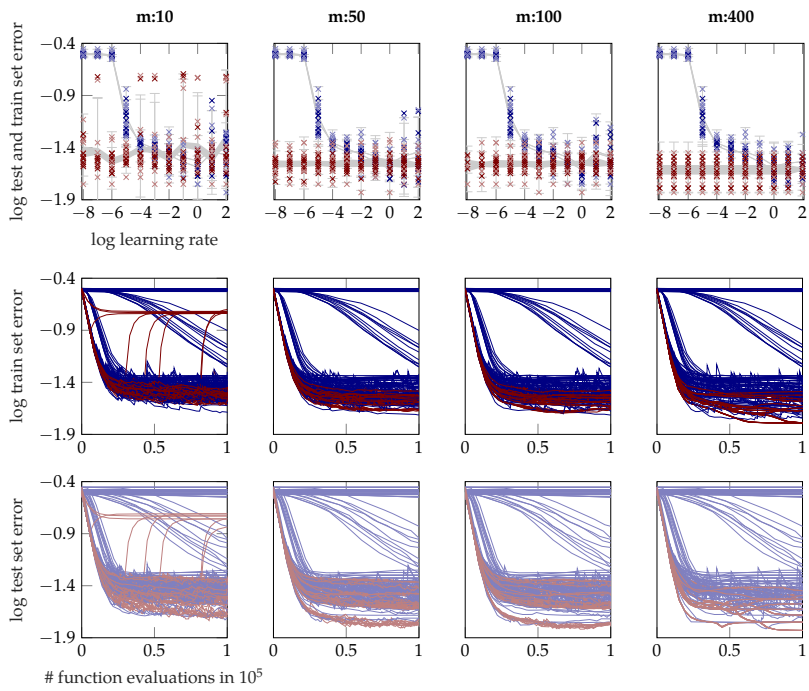


Figure 64: Performance of N-III on WDBC for varying mini-batch sizes: Plots, colors, and description same as in Figure 63. *Remark:* since the training set is of size 400, the most right column ( $m = 400$ ) in fact runs full-batch gradient descent; this is not a problem, since the probabilistic line search can handle noise free observations as well.

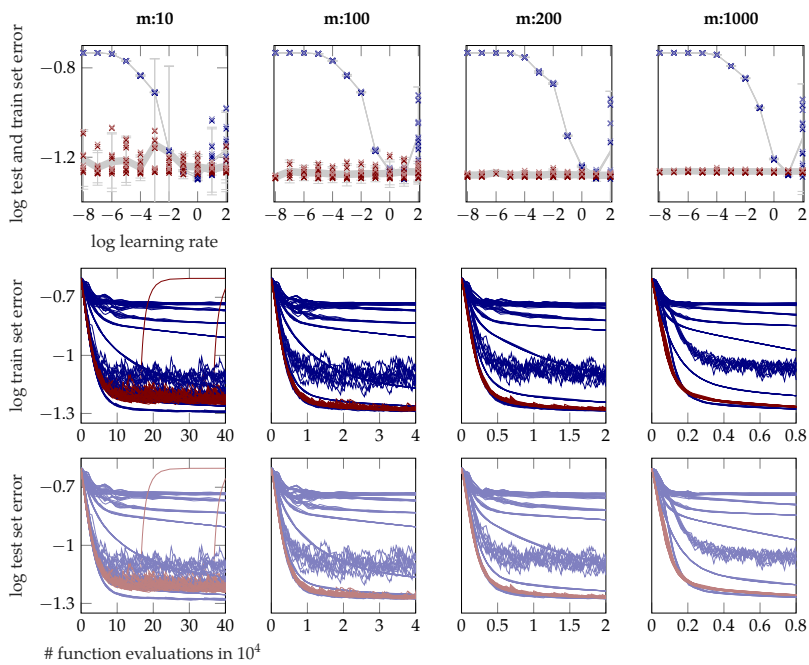


Figure 65: Performance of N-III on EPSILON for varying mini-batch sizes: Plots, colors, and description same as in Figure 63. EPSILON is the largest dataset, that was used in the experiments (400k samples); this did not seem to impair the performance of the line search or variance estimator.

## C.2 Hyper-parameter Sensitivity

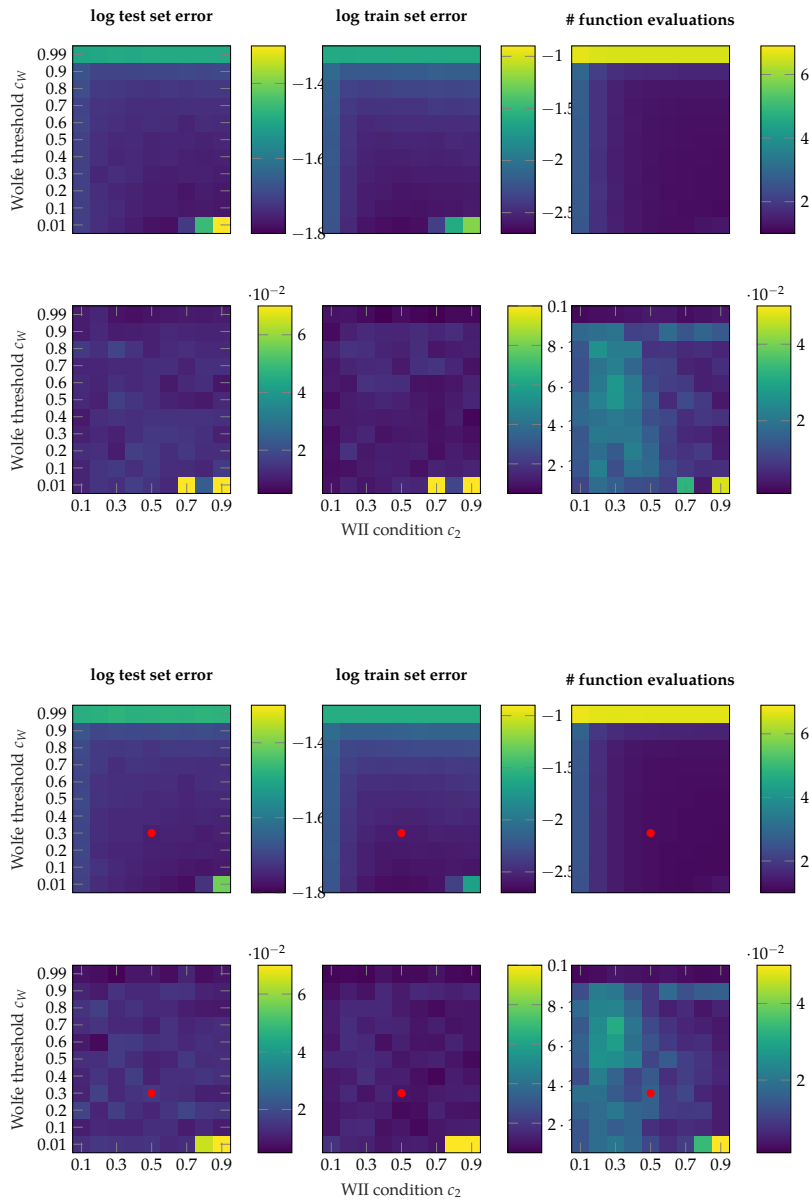


Figure 66: Sensitivity to hyper-parameters  $c_2$ , and  $c_W, \alpha_{\text{ext}} = 1.4$  (§7.4.1). Experimental setup as in Figures 46 and 45. Top row from left to right: logarithmic test set error, train set error, and average number of function evaluations per line search averaged over 10 different initializations. Bottom row: Corresponding relative standard deviations. In all plots darker colors are better. For extrapolation parameters  $\alpha_{\text{ext}} > 1$  (Figures 67, 68, 69, and 70) the different parameter combinations all result in similar good performance. Only at extreme choices (e. g.,  $\alpha_{\text{ext}} = 1.0$ , Figure 70, no extrapolation), the line search performs poorer. At the extreme value of  $c_W = 0.99$  (imposes near absolute certainty about the Wolfe conditions), the line search becomes less efficient. The default values ( $c_W = 0.3$ ,  $c_2 = 0.5$ , and  $\alpha_{\text{ext}} = 1.3$ ) are indicated as red dots in Figure 67.

Figure 67: Same as Figure 66 but for fixed  $\alpha_{\text{ext}} = 1.3$ . The default values adopted in the line search implementation ( $c_W = 0.3$ ,  $c_2 = 0.5$ , and  $\alpha_{\text{ext}} = 1.3$ ) are indicated as red dots.

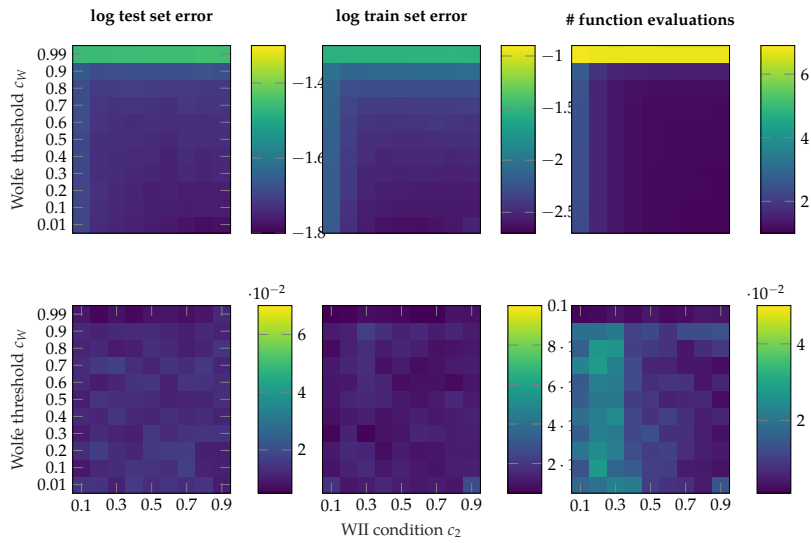


Figure 68: Same as Figure 66 but for fixed  $\alpha_{\text{ext}} = 1.2$

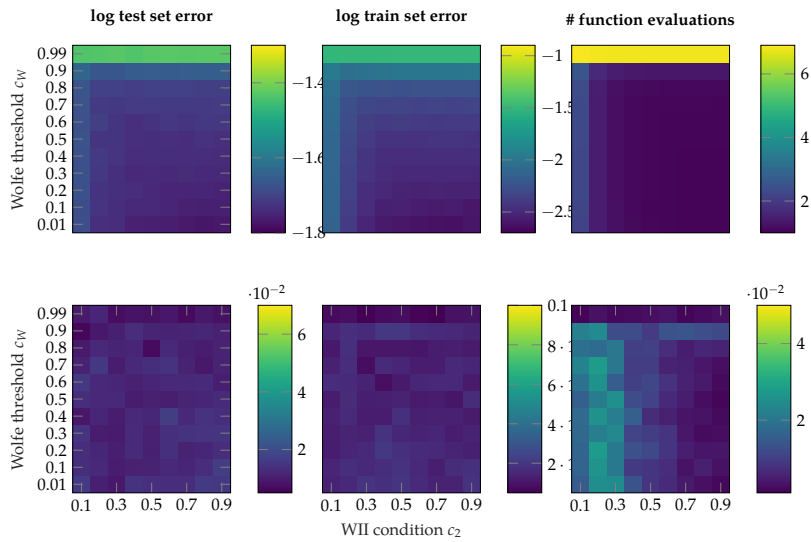


Figure 69: Same as Figure 66 but for fixed  $\alpha_{\text{ext}} = 1.1$

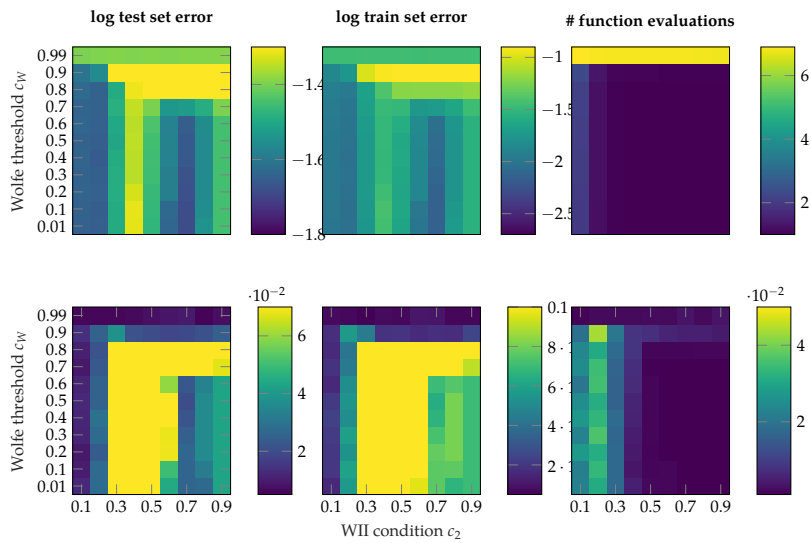


Figure 70: Same as Figure 66 but for fixed  $\alpha_{\text{ext}} = 1.0$

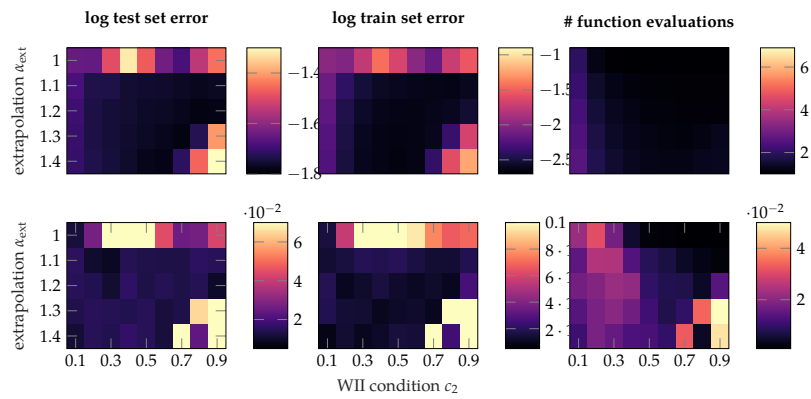


Figure 71: Sensitivity to varying hyperparameters  $c_2$ , and  $\alpha_{\text{ext}}$ ,  $c_W = 0.01$  (§7.4.1). Experimental setup as in Figures 45 and 46, plots like in Figure 66. In all plots darker colors are better. All choices of  $c_W$  result in good performance though very tight choices of  $c_W = 0.99$  (imposes near absolute certainty about the Wolfe conditions, Figure 81) are less efficient. As described in Figure 66, for a dropped extrapolation factor ( $\alpha_{\text{ext}} \rightarrow 1$ ) in combination with a loose curvature condition (large  $c_2$ ) the line searches performs poorer (top row, right half of columns in Figures 71–80). The default values ( $c_W = 0.3$ ,  $c_2 = 0.5$ , and  $\alpha_{\text{ext}} = 1.3$ ) are indicated as red dots in Figure 74.

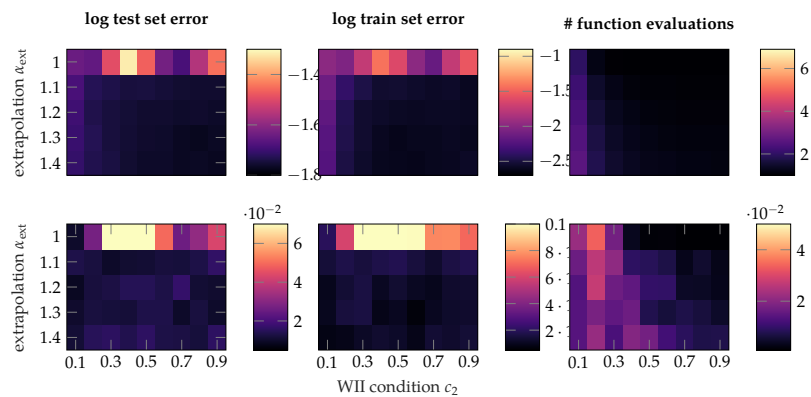


Figure 72: Same as Figure 71 but for fixed  $c_W = 0.10$ .

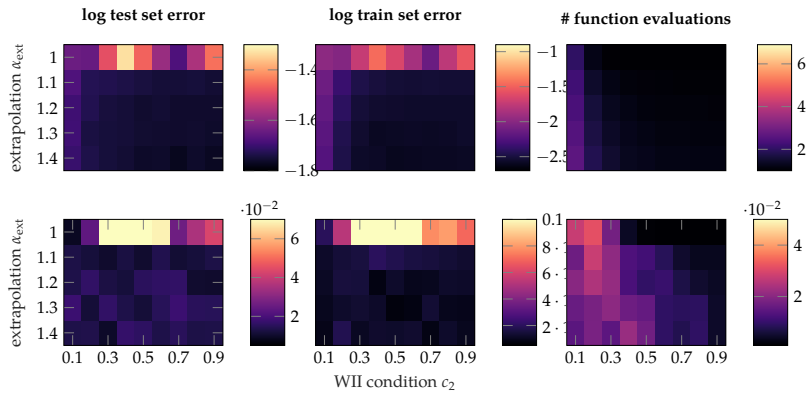


Figure 73: Same as Figure 71 but for fixed  $c_W = 0.20$ .

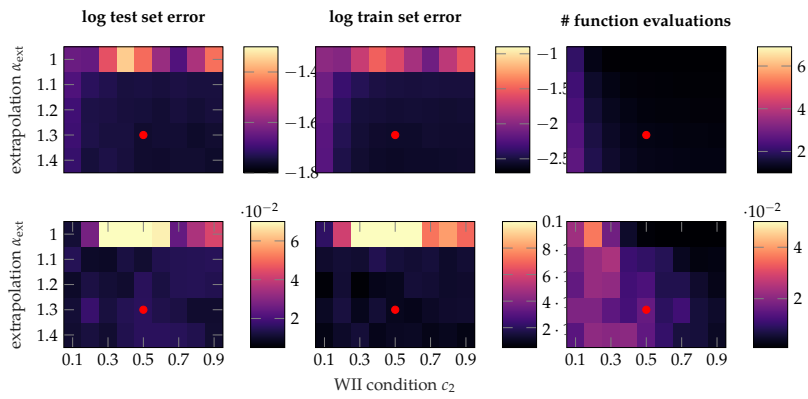


Figure 74: Same as Figure 71 but for fixed  $c_W = 0.30$ . The default values as red dots.

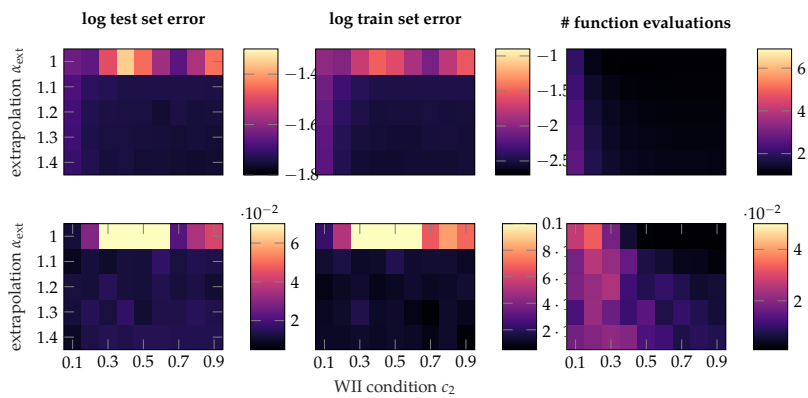


Figure 75: Same as Figure 71 but for fixed  $c_W = 0.40$ .

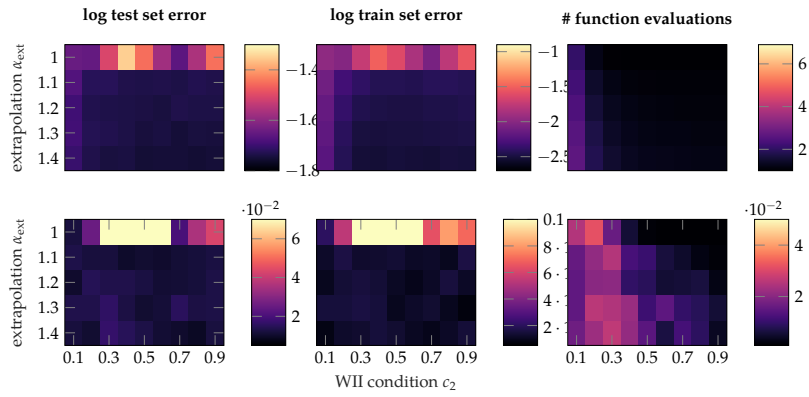


Figure 76: Same as Figure 71 but for fixed  $c_W = 0.50$ .

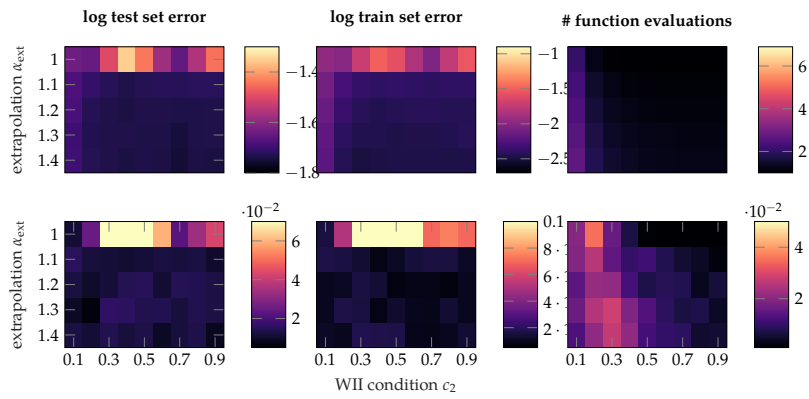


Figure 77: Same as Figure 71 but for fixed  $c_W = 0.60$ .

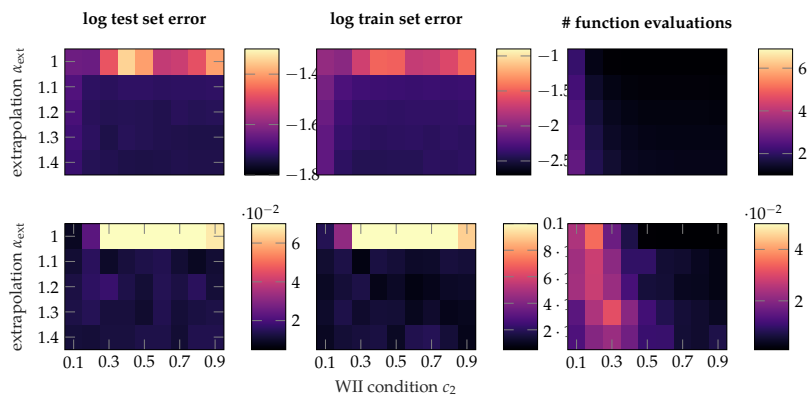


Figure 78: Same as Figure 71 but for fixed  $c_W = 0.70$ .

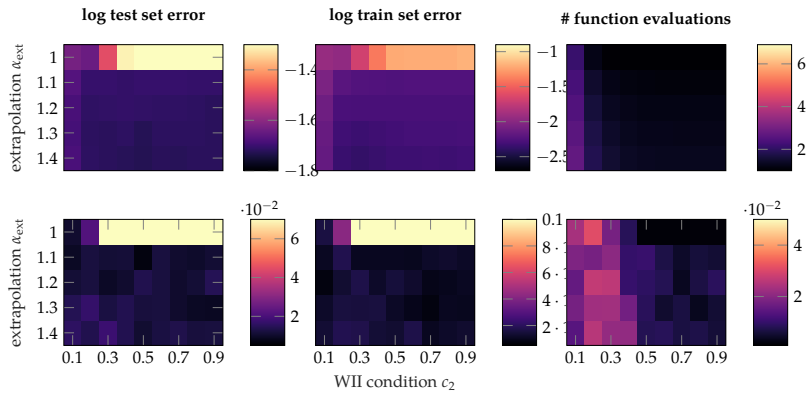


Figure 79: Same as Figure 71 but for fixed  $c_W = 0.80$ .

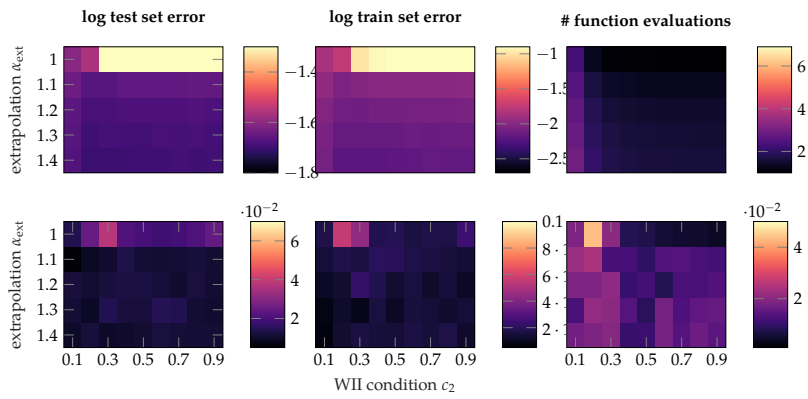


Figure 80: Same as Figure 71 but for fixed  $c_W = 0.90$ .

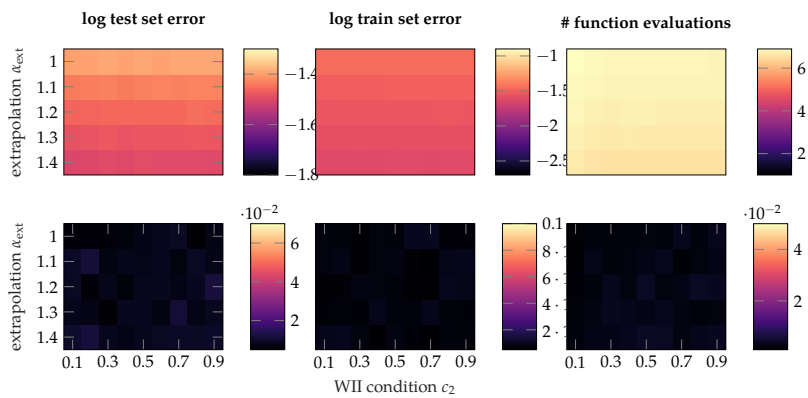


Figure 81: Same as Figure 71 but for fixed  $c_W = 0.99$ .

## C.3 Noise Estimation

Chapter 5 introduced the statistical variance estimators  $\hat{\Lambda}(w)$  and  $\hat{\Sigma}(w)$  of the function and gradient estimate  $L_S(w)$  and  $\nabla L_S(w)$  at position  $x$ . The underlying assumption is that  $L_S(w)$  and  $\nabla L_S(w)$  are distributed according to

$$\begin{bmatrix} L_S(w) \\ \nabla L_S(w) \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} \mathcal{L}(w) \\ \nabla \mathcal{L}(w) \end{bmatrix}, \begin{bmatrix} \Sigma(w) & 0_{D \times 1} \\ 0_{1 \times D} & \text{diag} \Sigma'(w) \end{bmatrix} \right) \quad (271)$$

which implies Eq 106

$$\begin{bmatrix} L_S(w) \\ s(w)' \cdot \nabla L_S(w) \end{bmatrix} = \begin{bmatrix} y(w) \\ y'(w) \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} f(w) \\ f'(w) \end{bmatrix}, \begin{bmatrix} \sigma_f(w) & 0 \\ 0 & \sigma_{f'}(w) \end{bmatrix} \right). \quad (272)$$

where  $s(w)$  is the possibly new search direction at  $w$ . This is an approximation since the true covariance matrix is in general not diagonal. A better estimator for the projected gradient noise would be (dropping  $w$  from the notation)

$$\begin{aligned} \eta_{f'} &= s^\top \left[ \frac{1}{|\mathcal{B}| - 1} \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} (\nabla \ell^k - \nabla L_S) (\nabla \ell^k - \nabla L_S)^\top \right] s \\ &= \sum_{i,j=1}^N s_i s_j \frac{1}{|\mathcal{B}| - 1} \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} (\nabla \ell_i^k - \nabla L_S^i) (\nabla \ell_j^k - \nabla L_S^j) \\ &= \frac{1}{|\mathcal{B}| - 1} \sum_{i,j=1}^N s_i s_j \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla \ell_i^k \nabla \ell_j^k - \nabla L_S^i \nabla L_S^j - \nabla L_S^j \nabla L_S^i + \nabla L_S^i \nabla L_S^j \right) \\ &= \frac{1}{|\mathcal{B}| - 1} \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \sum_{i,j=1}^N s_i \nabla \ell_i^k s_j \nabla \ell_j^k - \sum_{i,j=1}^N s_j \nabla L_S^j s_i \nabla L_S^i \right) \\ &= \frac{1}{|\mathcal{B}| - 1} \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} (s' \cdot \nabla \ell^k)^2 - (s' \cdot \nabla L_S)^2 \right). \end{aligned} \quad (273)$$

Comparing to  $\sigma_{f'}$  yields

$$\begin{aligned} \eta_{f'} &= \frac{1}{|\mathcal{B}| - 1} \sum_{i,j=1}^N s_i s_j \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla \ell_i^k \nabla \ell_j^k - \nabla L_S^j \nabla L_S^i \right) \\ &= \frac{1}{|\mathcal{B}| - 1} \sum_{i=1}^N s_i^2 \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} (\nabla \ell_i^k)^2 - (\nabla L_S^i)^2 \right) + \frac{1}{|\mathcal{B}| - 1} \sum_{i \neq j=1}^N s_i s_j \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla \ell_i^k \nabla \ell_j^k - \nabla L_S^j \nabla L_S^i \right) \\ \eta_{f'} &= \sigma_{f'} + \frac{1}{|\mathcal{B}| - 1} \sum_{i \neq j=1}^N s_i s_j \left( \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla \ell_i^k \nabla \ell_j^k - \nabla L_S^j \nabla L_S^i \right). \end{aligned} \quad (274)$$

From Eq 273 we see that, in order to compute  $\eta_{f'}$ , we need an efficient way of computing the inner product  $(s' \cdot \nabla \ell^k)$  for all  $k$ . In addition, we need to know the search direction  $s(w)$  of the potential next step (if  $w$  was accepted) at the time of computing  $\eta_{f'}$ . This is might be possible



e. g., for the SGD search direction where  $s(w) = -\frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla \ell^k(w)$  but potentially not possible or practical for arbitrary search directions. For all experiments in this thesis we used the approximate variance estimator  $\sigma_{f'}$ .

The following paragraph is concerned with the independence assumption of gradient and function value  $y$  and  $y'$  (in contrast to independence among gradient elements). In general  $y$  and  $y'$  are not independent since the algorithm draws them from the same minibatch; the likelihood including the correlation factor  $\rho$  reads

$$p(y(t), y'(t) | f) = \mathcal{N} \left( \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}; \begin{bmatrix} f(t) \\ f'(t) \end{bmatrix}, \begin{bmatrix} \sigma_f^2 & \rho \\ \rho & \sigma_{f'}^2 \end{bmatrix} \right). \quad (275)$$

The noise covariance matrix enters the GP only in the inverse of the sum containing the kernel matrix of the observations. We can compute it analytically for one datapoint at position  $t$ , since it is only a  $2 \times 2$  matrix. For  $\rho = 0$ , define:

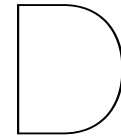
$$\begin{aligned} \det_{\rho=0} &:= [k_{tt} + \sigma_f^2][\partial k_{tt}^{\partial} + \sigma_{f'}^2] - k_{tt}^{\partial} \partial k_{tt} \\ G_{\rho=0}^{-1} &:= \begin{bmatrix} k_{tt} + \sigma_f^2 & k_{tt}^{\partial} \\ \partial k_{tt} & \partial k_{tt}^{\partial} + \sigma_{f'}^2 \end{bmatrix}^{-1} = \frac{1}{\det_{\rho=0}} \begin{bmatrix} \partial k_{tt}^{\partial} + \sigma_{f'}^2 & -k_{tt}^{\partial} \\ -\partial k_{tt} & k_{tt} + \sigma_f^2 \end{bmatrix}. \end{aligned} \quad (276)$$

For  $\rho \neq 0$  we thus get:

$$\begin{aligned} \det_{\rho \neq 0} &:= [k_{tt} + \sigma_f^2][\partial k_{tt}^{\partial} + \sigma_{f'}^2] - [k_{tt}^{\partial} + \rho][\partial k_{tt} + \rho] = \det_{\rho=0} - \rho(\partial k_{tt} + k_{tt}^{\partial}) - \rho^2 \\ G_{\rho \neq 0}^{-1} &:= \begin{bmatrix} k_{tt} + \sigma_f^2 & k_{tt}^{\partial} + \rho \\ \partial k_{tt} + \rho & \partial k_{tt}^{\partial} + \sigma_{f'}^2 \end{bmatrix}^{-1} = \frac{1}{\det_{\rho \neq 0}} \begin{bmatrix} \partial k_{tt}^{\partial} + \sigma_{f'}^2 & -(k_{tt}^{\partial} + \rho) \\ -(\partial k_{tt} + \rho) & k_{tt} + \sigma_f^2 \end{bmatrix} \\ &= \frac{\det_{\rho=0}}{\det_{\rho \neq 0}} G_{\rho=0}^{-1} - \frac{\rho}{\det_{\rho \neq 0}} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \end{aligned} \quad (277)$$

The fraction  $\det_{\rho=0}/\det_{\rho \neq 0}$  in the first term of the last row, is a positive scalar that scales all element of  $G_{\rho=0}^{-1}$  equally (since  $G_{\rho=0}$  and  $G_{\rho \neq 0}$  are positive definite matrices, we know that  $\det_{\rho=0} > 0, \det_{\rho \neq 0} > 0$ ). If  $|\rho|$  is small in comparison to the determinant  $\det_{\rho=0}$ , then  $\det_{\rho \neq 0} \approx \det_{\rho=0}$  and the scaling factor is approximately one. The second term corrects off-diagonal elements in  $G_{\rho \neq 0}$  and is proportional to  $\rho$ ; if  $|\rho| \ll \det_{\rho=0}$  this term is small as well. It might be possible to estimate  $\rho$  as well from the minibatch in a similar style to the estimation of  $\sigma_f$  and  $\sigma_{f'}$ ; it is not clear, however, if the additional computational cost would justify the improvements in the GP-inference.





## Detailed Pseudocode of PROBLETS

---

Algorithm 4 of Chapter 7 § 7.2 roughly sketches the structure of the probabilistic line search and its main ingredients. This section provides a detailed pseudocode which can be used for re-implementation. It is based on the code which was used for the experiments in this thesis. A matlab implementation including a minimal example can be found at <http://tinyurl.com/probLineSearch>. The actual line search routine is called `PROBLINESearch` below and is quite short. Most of the pseudocode is occupied with comments, helper function that define the kernel of the GP, the GP-update or Gauss cdf and pdf. For better readability of the pseudocode we use the following color coding: **Green**: variables of the integrated Wiener process, **Red**: most recently evaluated observation (noisy loss and gradient). If the line search terminates, these will be returned as 'accepted'. **Orange**: inputs from the main solver procedure and unchanged during each line search.

operator or function	definition
$A \odot B$	elementwise multiplication
$A \oslash B$	elementwise division
$A^{\odot b}$	elementwise power of $b$
$A'$	transpose of $A$
$A \cdot B$	scalar-scalar, scalar-matrix or matrix-matrix multiplication
$A/B$	right matrix division, the same as $A \cdot B^{-1}$
$A \setminus B$	left matrix division, the same as $A^{-1} \cdot B$
$\text{sign}(a)$	sign of scalar $a$
$\text{erf}(x)$	error function $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$
$\max(A)$	maximum element in $A$
$\min(A)$	minimum element in $A$
$ a $	absolute value of scalar $a$
$A < B$	elementwise 'less' comparison
$A \leq B$	elementwise 'less-or-equal' comparison
$A > B$	elementwise 'greater' comparison
$A \geq B$	elementwise 'greater-or-equal' comparison
$[a, b, c] \leftarrow f(x)$	function $f$ called at $x$ returns the values $a, b$ and $c$

---

```

1: function SGDSOLVER( $f, x_0$ )
2:   //  $f$  – function handle. Usage:  $[y, dy, \Sigma_f, \Sigma_{df}] \leftarrow f(x)$ .
3:
4:    $x \leftarrow x_0$  // initial weights
5:    $\alpha \leftarrow e.g. \approx 10^{-4}$  // initial step (rather small)
6:    $\alpha_{\text{stats}} \leftarrow \alpha$ 
7:    $[y, dy, \Sigma_f, \Sigma_{df}] \leftarrow f(x)$  // initial function evaluation
8:    $d \leftarrow -dy$  // initial search direction
9:
10:  // loop over line searches
11:  while budget not used do
12:     $[\alpha, \alpha_{\text{stats}}, x, y, dy, \Sigma_f, \Sigma_{df}] \leftarrow \text{PROBLINESEARCH}(x, d, y, dy, \Sigma_f, \Sigma_{df}, \alpha, \alpha_{\text{stats}}, f)$ 
13:     $d \leftarrow -dy$  // new search direction
14:  end while
15:  return  $x$ 
16: end function

```

---

```

1: function PROBLINESEARCH( $x_0, d, f_0, df_0, \Sigma_{f_0}, \Sigma_{df_0}, \alpha_0, \alpha_{\text{stats}}, f$ )
2:   //  $x_0$  – current weights [ $D \times 1$ ]
3:   //  $f$  – function handle to objective.
4:   //  $d$  – search direction [ $D \times 1$ ]
5:   //  $f_0$  – function value at start,  $f_0 = f(x_0)$ 
6:   //  $df_0$  – gradient at start,  $df_0 = \nabla f(x_0)$  [ $D \times 1$ ]
7:   //  $\Sigma_{f_0}$  – sample variance of  $f_0$ 
8:   //  $\Sigma_{df_0}$  – sample variances of  $df_0$ , [ $D \times 1$ ]
9:   //  $\alpha_0$  – initial step size
10:
11:  // scaling and noise level of GP
12:   $\beta \leftarrow |d' \cdot \Sigma_{df_0}|$ 
13:   $\sigma_f \leftarrow \sqrt{\Sigma_{f_0} / (\alpha_0 \cdot \beta)}$  // variance of  $y_0$ 
14:   $\sigma_{df} \leftarrow \sqrt{((d^{\odot 2})' \cdot \Sigma_{df_0}) / \beta}$  // sample variance of  $dy_0$ 
15:
16:  // initialize counter and non-fixed parameters
17:   $L \leftarrow 6$  // maximum # of  $f$  evaluations per line search
18:   $N \leftarrow 1$  // size of GP =  $2 \cdot N$ 
19:   $t_{\text{ext}} \leftarrow 1$  // scaled step size for extrapolation
20:   $tt \leftarrow 1$  // scaled position of first function evaluation
21:
22:  // storage for GP, dynamic arrays of max size [ $L + 1 \times 1$ ]
23:   $T \leftarrow [0]$  // scaled positions along search direction
24:   $Y \leftarrow [0]$  // scaled function values at  $T$ 
25:   $dY \leftarrow [(df_0' \cdot d) / \beta]$  // scaled projected gradients at  $T$ 
26:

```

```

27: // initialize GP with observation at start
28:  $[G, A] \leftarrow \text{UPDATEGP}(T, Y, dY, N, \sigma_f, \sigma_{df})$ 
29:
30: // loop until budget is used or acceptable point is found
31: for  $N$  from 2 to  $L + 1$  do
32:
33:     // evaluate objective function at  $tt$ .
34:      $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N] \leftarrow \text{EVALUATEOBJECTIVE}(tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f)$ 
35:
36:     // update the GP which is now of size  $2 \cdot N$ .
37:      $[G, A] \leftarrow \text{UPDATEGP}(T, Y, dY, N, \sigma_f, \sigma_{df})$ 
38:
39:     // storage candidates, dynamic arrays of max size  $[N \times 1]$ .
40:      $T_{\text{cand}} \leftarrow []$  // scaled position of candidates
41:      $M_{\text{cand}} \leftarrow []$  // GP mean of candidates
42:      $S_{\text{cand}} \leftarrow []$  // GP standard deviation of candidates
43:
44:     // current point above the Wolfe threshold? If yes, return.
45:     if  $\text{PROBWOLFE}(tt, T, A, G)$  then
46:          $output \leftarrow \text{RESCALEOUTPUT}(x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta)$ 
47:         return  $output$ 
48:     end if
49:
50:     // Wolfe conditions not satisfied at this point.
51:     // find suitable candidates for next evaluation.
52:
53:     // GP mean of function values and gradients at points in  $T$ .
54:      $M \leftarrow \text{map function } M(\_, T, A) \text{ over } T$ 
55:      $dM \leftarrow \text{map function } D1M(\_, T, A) \text{ over } T$ 
56:
57:     // candidates 1: local minima of GP mean.
58:      $T_{\text{sorted}} \leftarrow \text{sort } T \text{ in ascending order}$ 
59:      $T_{\text{Wolfes}} \leftarrow []$  // prepare list of acceptable points
60:
61:     // iterate through all  $N - 1$  cells, compute local minima.
62:     for  $n$  from 1 to  $N - 1$  do
63:          $T_n \leftarrow \text{value of } T_{\text{sorted}} \text{ at } n$ 
64:          $T_{n+1} \leftarrow \text{value of } T_{\text{sorted}} \text{ at } n + 1$ 
65:
66:         // add a little offset for numerical stability
67:          $t_{\text{rep}} \leftarrow T_n + 10^{-6} \cdot (T_{n+1} - T_n)$ 
68:
69:         // compute location of cubic minimum in  $n^{\text{th}}$  cell
70:          $t_{\text{cubMin}} \leftarrow \text{CUBICMINIMUM}(t_{\text{rep}}, T, A, N)$ 
71:

```

```

72:     // add point to candidates if minimum lies between  $T_n$  and  $T_{n+1}$ 
73:     if  $t_{\text{cubMin}} > T_n$  and  $t_{\text{cubMin}} < T_{n+1}$  then
74:         if (not isnanOrIsinf( $t_{\text{cubMin}}$ )) and ( $t_{\text{cubMin}} > 0$ ) then
75:              $T_{\text{cand}} \leftarrow \text{append } t_{\text{cubMin}}$ 
76:              $M_{\text{cand}} \leftarrow \text{append } \mathbf{M}(t_{\text{cubMin}}, T, A)$ 
77:              $S_{\text{cand}} \leftarrow \text{append } \mathbf{V}(t_{\text{cubMin}}, T, G)$ 
78:         end if
79:     else
80:
81:         // most likely uphill? If yes, break.
82:         if  $n = 1$  and  $\text{D1M}(0, T, A) > 0$  then
83:              $r \leftarrow 0.01$ 
84:              $tt \leftarrow r \cdot (T_n + T_{n+1})$ 
85:
86:             // evaluate objective function at  $tt$  and return.
87:              $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N] \leftarrow \text{EVALUATEOBJECTIVE}(tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f)$ 
88:
89:              $\text{output} \leftarrow \text{RESCALEOUTPUT}(x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta)$ 
90:             return output
91:         end if
92:     end if
93:
94:     // check for Wolfe point among the old evaluations
95:     if  $n > 1$  and  $\text{PROBWOLFE}(T_n, T, A, G)$  then
96:          $T_{\text{Wolfes}} \leftarrow \text{append } T_n$ 
97:     end if
98: end for
99:
100: // check if acceptable points exists and return
101: if  $T_{\text{Wolfes}}$  is not empty then
102:
103:     // if last evaluated point is among acceptable ones, return it.
104:     if  $tt$  in  $T_{\text{Wolfes}}$  then
105:          $\text{output} \leftarrow \text{RESCALEOUTPUT}(x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta)$ 
106:         return output
107:     end if
108:
109:     // else, choose the one with the lowest GP mean
110:     // and re-evaluate its gradient.
111:      $M_{\text{Wolfes}} \leftarrow \text{map } \mathbf{M}(\_, T, A)$  over  $T_{\text{Wolfes}}$ 
112:      $tt \leftarrow \text{value of } T_{\text{Wolfes}}$  at index of  $\min(M_{\text{Wolfes}})$ 
113:
114:     // evaluate objective function at  $tt$ .
115:      $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N] \leftarrow \text{EVALUATEOBJECTIVE}(tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f)$ 
116:

```

```

117:      $output \leftarrow \text{RESCALEOUTPUT}(x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta)$ 
118:     return  $output$ 
119: end if
120:
121:     // candidates 2: one extrapolation step
122:      $T_{\text{cand}} \leftarrow \text{append } \max(T) + t_{\text{ext}}$ 
123:      $M_{\text{cand}} \leftarrow \text{append } \mathfrak{M}(\max(T) + t_{\text{ext}}, T, A)$ 
124:      $S_{\text{cand}} \leftarrow \text{append } \mathfrak{V}(\max(T) + t_{\text{ext}}, T, G)^{\frac{1}{2}}$ 
125:
126:     // find minimal mean among  $M$ .
127:      $\mu_{\text{EI}} \leftarrow \text{minimal value of } M$ 
128:
129:     // expected improvement and Wolfe probabilities at  $T_{\text{cand}}$ 
130:      $EI_{\text{cand}} \leftarrow \text{EXPECTEDIMPROVEMENT}(M_{\text{cand}}, S_{\text{cand}}, \mu_{\text{EI}})$ 
131:      $PW_{\text{cand}} \leftarrow \text{map } \text{PROBWOLFE}(\_, T, A, G) \text{ over } T_{\text{cand}}$ 
132:
133:     // choose candidate that maximizes  $EI_{\text{cand}} \wedge PW_{\text{cand}}$ 
134:      $i_{\text{bestCand}} \leftarrow \text{index of } \max(EI_{\text{cand}} \odot PW_{\text{cand}})$ 
135:      $tt_{\text{bestCand}} \leftarrow \text{value of } T_{\text{cand}} \text{ at } i_{\text{bestCand}}$ 
136:
137:     // extend extrapolation step if necessary
138:     if  $tt_{\text{bestCand}}$  is equal to  $tt + t_{\text{ext}}$  then
139:          $t_{\text{ext}} \leftarrow 2 \cdot t_{\text{ext}}$ 
140:     end if
141:
142:     // set location for next evaluation
143:      $tt \leftarrow tt_{\text{bestCand}}$ 
144: end for
145:
146:     // limit reached: evaluate a final time
147:     // and return the point with lowest GP mean
148:      $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N] \leftarrow \text{EVALUATEOBJECTIVE}(tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f)$ 
149:
150:     // update the GP which is now of size  $2 \cdot N$ .
151:      $[G, A] \leftarrow \text{UPDATEGP}(T, Y, dY, N, \sigma_f, \sigma_{df})$ 
152:
153:     // check last point for acceptance
154:     if  $\text{PROBWOLFE}(tt, T, A, G)$  then
155:          $output \leftarrow \text{RESCALEOUTPUT}(x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta)$ 
156:         return  $output$ 
157:     end if
158:
159:     // at the end of budget return point with the lowest GP mean
160:      $M \leftarrow \text{map } \mathfrak{M}(\_, T, A) \text{ over } T$ 
161:      $i_{\text{lowest}} \leftarrow \text{index of minimal value in } M$ 

```

```

162:  $t_{\text{lowest}} \leftarrow$  value of  $T$  at  $i_{\text{lowest}}$ 
163:
164: // if  $t_{\text{lowest}}$  is the last evaluated point, return
165: if  $t_{\text{lowest}}$  is equal to  $tt$  then
166:      $output \leftarrow$  RESCALEOUTPUT( $x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta$ )
167:     return  $output$ 
168: end if
169:
170: // else, re-evaluate its gradient and return
171:  $tt \leftarrow$  value of  $t_{\text{lowest}}$ 
172:
173: // evaluate objective function at  $tt$ .
174:  $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N] \leftarrow$  EVALUATEOBJECTIVE( $tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f$ )
175:
176:  $output \leftarrow$  RESCALEOUTPUT( $x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta$ )
177: return  $output$ 
178: end function

```

---

```

1: function RESCALEOUTPUT( $x_0, f_0, \alpha_0, d, tt, y, dy, \Sigma_f, \Sigma_{df}, \beta, \alpha_{\text{stats}}$ )
2:     // design parameters
3:      $\alpha_{\text{ext}} \leftarrow 1.3$  // extrapolation parameter
4:      $\theta_{\text{reset}} \leftarrow 100$  // reset threshold for GP scale
5:
6:      $\alpha_{\text{acc}} \leftarrow tt \cdot \alpha_0$  // rescale accepted step size
7:      $x_{\text{acc}} \leftarrow x_0 + \alpha_{\text{acc}} \cdot d$  // update weights
8:      $f_{\text{acc}} \leftarrow y \cdot (\alpha_0 \cdot \beta) + f_0$  // rescale accepted function value
9:      $df_{\text{acc}} \leftarrow dy$  // accepted gradient
10:     $\Sigma_{f_{\text{acc}}} \leftarrow \Sigma_f$  // sample variance of  $f_{\text{acc}}$ 
11:     $\Sigma_{df_{\text{acc}}} \leftarrow \Sigma_{df}$  // sample variances of  $df_{\text{acc}}$ 
12:     $\gamma \leftarrow 0.95$  // exponential running average of scalings
13:     $\alpha_{\text{stats}} \leftarrow \gamma \cdot \alpha_{\text{stats}} + (1 - \gamma) \cdot \alpha_{\text{acc}}$ 
14:     $\alpha_{\text{next}} \leftarrow \alpha_{\text{acc}} \cdot \alpha_{\text{ext}}$  // next initial step size
15:
16:    // reset it GP scaling if needed.
17:    if ( $\alpha_{\text{next}} < \alpha_{\text{stats}} / \theta_{\text{reset}}$ ) or ( $\alpha_{\text{next}} > \alpha_{\text{stats}} \cdot \theta_{\text{reset}}$ ) then
18:         $\alpha_{\text{next}} \leftarrow \alpha_{\text{stats}}$ 
19:    end if
20:
21:    // compressed output for readability of pseudocode
22:     $output \leftarrow [\alpha_{\text{next}}, \alpha_{\text{stats}}, x_{\text{acc}}, f_{\text{acc}}, df_{\text{acc}}, \Sigma_{f_{\text{acc}}}, \Sigma_{df_{\text{acc}}}]$ 
23:    return  $output$ 
24: end function

```

---



---

```

1: function EVALUATEOBJECTIVE( $tt, x_0, \alpha_0, d, T, Y, dY, N, \beta, f$ )
2:   // evaluate objective function at  $tt$ 
3:    $[y, dy, \Sigma_f, \Sigma_{df}] \leftarrow f(x_0 + tt \cdot \alpha_0 \cdot d)$ 
4:
5:   // scale output
6:    $y \leftarrow (y - f_0) / (\alpha_0 \cdot \beta)$ 
7:    $dy \leftarrow (dy' \cdot d) / \beta$ 
8:
9:   // storage
10:   $T \leftarrow \text{append } tt$ 
11:   $Y \leftarrow \text{append } y$ 
12:   $dY \leftarrow \text{append } dy$ 
13:   $N \leftarrow N + 1$ 
14:
15:  return  $[y, dy, \Sigma_f, \Sigma_{df}, T, Y, dY, N]$ 
16: end function

```

---

```

1: function CUBICMINIMUM( $t, T, A, N$ )
2:   // compute necessary derivatives of GP mean at  $t$ 
3:    $d1m_t \leftarrow D1M(t, T, A)$ 
4:    $d2m_t \leftarrow D2M(t, T, A)$ 
5:    $d3m_t \leftarrow D3M(t, T, A, N)$ 
6:    $a \leftarrow 0.5 \cdot d3m_t$ 
7:    $b \leftarrow d2m_t - t \cdot d3m_t$ 
8:    $c \leftarrow d1m_t - d2m_t \cdot t + 0.5 \cdot d3m_t \cdot t^2$ 
9:
10:  // third derivative is almost zero  $\rightarrow$  single extremum
11:  if  $|d3m_t| < 1^{-9}$  then
12:     $t_{\text{cubMin}} \leftarrow -(d1m_t - t \cdot d2m_t) / d2m_t$ 
13:    return  $t_{\text{cubMin}}$ 
14:  end if
15:
16:  // roots are complex, no extremum
17:   $\lambda \leftarrow b^2 - 4 \cdot a \cdot c$ 
18:  if  $\lambda < 0$  then
19:     $t_{\text{cubMin}} \leftarrow +\infty$ 
20:    return  $t_{\text{cubMin}}$ 
21:  end if
22:
23:  // compute the two possible roots
24:   $LR \leftarrow (-b - \text{sign}(a) \cdot \sqrt{\lambda}) / (2 \cdot a)$  // left root
25:   $RR \leftarrow (-b + \text{sign}(a) \cdot \sqrt{\lambda}) / (2 \cdot a)$  // right root
26:
27:  // compute values of left and right root (up to a constant)

```

```

28:   $dt_L \leftarrow LR - t$  // distance to left root
29:   $dt_R \leftarrow RR - t$  // distance to right root
30:   $CV_L \leftarrow d1m_t \cdot dt_L + 0.5 \cdot d2m_t \cdot dt_L^2 + (d3m_t \cdot dt_L^3)/6$ 
31:   $CV_R \leftarrow d1m_t \cdot dt_R + 0.5 \cdot d2m_t \cdot dt_R^2 + (d3m_t \cdot dt_R^3)/6$ 
32:
33:  // find the minimum and return it.
34:  if  $CV_L < CV_R$  then
35:     $t_{\text{cubMin}} \leftarrow LR$ 
36:  else
37:     $t_{\text{cubMin}} \leftarrow RR$ 
38:  end if
39:
40:  return  $t_{\text{cubMin}}$ 
41: end function

```

---

```

1: function UPDATEGP( $T, Y, dY, N, \sigma_f, \sigma_{df}$ )
2:  // initialize kernel matrices
3:   $k_{TT} \leftarrow [N \times N]$  matrix with zeros
4:   $kd_{TT} \leftarrow [N \times N]$  matrix with zeros
5:   $dkd_{TT} \leftarrow [N \times N]$  matrix with zeros
6:
7:  // fill kernel matrices
8:  for  $i = 1$  to  $N$  do
9:    for  $j = 1$  to  $N$  do
10:      $k_{TT}(i, j) \leftarrow \kappa(T(i), T(j))$ 
11:      $kd_{TT}(i, j) \leftarrow \kappa_D(T(i), T(j))$ 
12:      $dkd_{TT}(i, j) \leftarrow \text{DKD}(T(i), T(j))$ 
13:    end for
14:  end for
15:
16:  // diagonal covariance of Gaussian likelihood  $[2N \times 2N]$ .
17:   $\Lambda \leftarrow \begin{bmatrix} \text{diag}(\sigma_f^2)_{N \times N} & 0_{N \times N} \\ 0_{N \times N} & \text{diag}(\sigma_{df}^2)_{N \times N} \end{bmatrix}$ 
18:
19:   $G \leftarrow \begin{pmatrix} k_{TT} & kd_{TT} \\ kd'_{TT} & dkd_{TT} \end{pmatrix} + \Lambda$  //  $[2N \times 2N]$  matrix
20:
21:  // residual between observed and predicted data
22:   $\Delta \leftarrow \begin{pmatrix} Y \\ dY \end{pmatrix}$  //  $[2N \times 1]$  vector
23:
24:  // compute weighted observations  $A$ .
25:   $A \leftarrow G \backslash \Delta$  //  $[2N \times 1]$  vector
26:

```

```

27:   return [G, A]
28: end function

```

---

```

1: function M(t, T, A)
2:   // posterior mean at t
3:   return [κ(t, T'), κD(t, T')] · A
4: end function
5:
6: function D1M(t, T, A)
7:   // first derivative of mean at t
8:   return [Dκ(t, T'), DκD(t, T')] · A
9: end function
10:
11: function D2M(t, T, A)
12:   // second derivative of mean at t
13:   return [DDDκ(t, T'), DDDκD(t, T')] · A
14: end function
15:
16: function D3M(t, T, A, N)
17:   // third derivative of mean at t
18:   return [DDDDκ(t, T'), ZEROS(1, N)] · A
19: end function
20:
21: function V(t, T, G)
22:   // posterior variance of function values at t
23:   return κ(t, t) - [κ(t, T'), κD(t, T')] · (G \ [κ(t, T'), κD(t, T')]')
24: end function
25:
26: function VD(t, T, G)
27:   // posterior variance of function values and derivatives at t
28:   return κD(t, t) - [κ(t, T'), κD(t, T')] · (G \ [Dκ(t, T'), DκD(t, T')]')
29: end function
30:
31: function DVD(t, T, G)
32:   // posterior variance of derivatives at t
33:   return DκD(t, t) - [Dκ(t, T'), DκD(t, T')] · (G \ [Dκ(t, T'), DκD(t, T')]')
34: end function
35:
36: function V0F(t, T, G)
37:   // posterior covariances of function values at t = 0 and t
38:   return κ(0, t) - [κ(0, T'), κD(0, T')] · (G \ [κ(t, T'), κD(t, T')]')
39: end function
40:
41: function VD0F(t, T, G)
42:   // posterior covariance of gradient and function value

```

```

43: // at  $t = 0$  and  $t$  respectively
44: return  $\text{DK}(0, t) - [\text{DK}(0, T'), \text{DKD}(0, T')] \cdot (G \setminus [\kappa(t, T'), \kappa_D(t, T')])'$ 
45: end function
46:
47: function  $\text{V0DF}(t, T, G)$ 
48: // posterior covariance of function value and gradient
49: // at  $t = 0$  and  $t$  respectively
50: return  $\kappa_D(0, t) - [\kappa(0, T'), \kappa_D(0, T')] \cdot (G \setminus [\text{DK}(t, T'), \text{DKD}(t, T')])'$ 
51: end function
52:
53: function  $\text{VD0DF}(t, T, G)$ 
54: // same as  $\text{V0F}(\_)$  but for gradients
55: return  $\text{DKD}(0, t) - [\text{DK}(0, T'), \text{DKD}(0, T')] \cdot (G \setminus [\text{DK}(t, T'), \text{DKD}(t, T')])'$ 
56: end function

```

---

```

1: // all following procedures use the same design parameter:
2:  $\tau \leftarrow 10$ 
3:
4: function  $\kappa(a, b)$ 
5: // Wiener kernel integrated once in each argument
6: return  $1/3 \odot \min(a + \tau, b + \tau)^{\odot 3} + 1/2 \odot |a - b| \odot \min(a + \tau, b + \tau)^{\odot 2}$ 
7: end function
8:
9: function  $\kappa_D(a, b)$ 
10: // Wiener kernel integrated in first argument
11: return  $1/2 \odot (a < b) \odot (a + \tau)^{\odot 2} + (a \geq b) \odot ((a + \tau) \cdot (b + \tau) - 1/2 \odot (b + \tau)^{\odot 2})$ 
12: end function
13:
14: function  $\text{DK}(a, b)$ 
15: // Wiener kernel integrated in second argument
16: return  $1/2 \odot (a > b) \odot (b + \tau)^{\odot 2} + (a \leq b) \odot ((a + \tau) \cdot (b + \tau) - 1/2 \odot (a + \tau)^{\odot 2})$ 
17: end function
18:
19: function  $\text{DKD}(a, b)$ 
20: // Wiener kernel
21: return  $\min(a + \tau, b + \tau)$ 
22: end function
23:
24: function  $\text{DDK}(a, b)$ 
25: // Wiener kernel integrated in second argument
26: // and 1x derived in first argument
27: return  $(a \leq b) \odot (b - a)$ 
28: end function
29:
30: function  $\text{DDKD}(a, b)$ 

```

```

31: // Wiener kernel 1x derived in first argument
32: return (a ≤ b)
33: end function
34:
35: function DDDK(a, b)
36: // Wiener kernel 2x derived in first argument
37: // and integrated in second argument
38: return -(a ≤ b)
39: end function

```

---

```

1: function PROBWOLFE(t, T, A, G)
2: // design parameters
3: c1 ← 0.05 // constant for Armijo condition
4: c2 ← 0.5 // constant for curvature condition
5: cW ← 0.3 // threshold for Wolfe probability
6:
7: // mean and covariance values at start position (t = 0)
8: m0 ← M(0, T, A)
9: dm0 ← D1M(0, T, A)
10: V0 ← V(0, T, G)
11: Vd0 ← VD(0, T, G)
12: dVd0 ← DVd(0, T, G)
13:
14: // marginal mean and variance for Armijo condition
15: ma ← m0 - M(t, T, A) + c1 · t · dm0
16: Vaa ← V0 + (c1 · t)2 · dVd0 + V(t) + 2 · (c1 · t · (Vd0 - VD0F(t)) - V0F(t))
17:
18: // marginal mean and variance for curvature condition
19: mb ← D1M(t) - c2 · dm0
20: Vbb ← c22 · dVd0 - 2 · c2 · VD0DF(t) + DVd(t)
21:
22: // covariance between conditions
23: Vab ← -c2 · (Vd0 + c1 · t · dVd0) + c2 · VD0F(t) + V0DF(t) + c1 · t · VD0DF(t) - VD(t)
24:
25: // small variances → deterministic evaluation
26: if Vaa ≤ 10-9 and Vbb ≤ 10-9 then
27: pWolfe ← (ma ≥ 0) · (mb ≥ 0)
28:
29: // accept?
30: pacc ← pWolfe > cW
31: return pacc
32: end if
33:
34: // zero or negative variances (maybe something went wrong?)
35: if Vaa ≤ 0 or Vbb ≤ 0 then

```

```

36:     return 0
37: end if
38:
39:     // noisy case (everything is alright)
40:      $\rho \leftarrow V_{ab} / \sqrt{V_{aa} \cdot V_{bb}}$  // correlation
41:
42:     // lower and upper integral limits for Armijo condition
43:      $low_a \leftarrow -m_a / \sqrt{V_{aa}}$ 
44:      $up_a \leftarrow +\infty$ 
45:
46:     // lower and upper integral limits for curvature condition
47:      $low_b \leftarrow -m_b / \sqrt{V_{bb}}$ 
48:      $up_b \leftarrow (2 \cdot c_2 \cdot (|dm_0| + 2 \cdot \sqrt{dVd_0}) - m_b) / \sqrt{V_{bb}}$ 
49:
50:     // compute Wolfe probability
51:      $p_{Wolfe} \leftarrow \text{bvn}(low_a, up_a, low_b, up_b, \rho)$ 
52:
53:     // accept?
54:      $p_{acc} \leftarrow p_{Wolfe} > c_W$ 
55:     return  $p_{acc}$ 
56:

```

The function  $\text{bvn}(low_a, up_a, low_b, up_b, \rho)$  evaluates the  
2D-integral

$$\int_{low_a}^{up_a} \int_{low_b}^{up_b} \mathcal{N} \left( \begin{bmatrix} a \\ b \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \right) da db.$$

```

57:
58:
59: end function

```

---

```

1: function GAUSSCDF(z)
2:     // Gauss cumulative density function
3:     return  $0.5 \odot (1 + \text{erf}(z / \sqrt{2}))$ 
4: end function
5:
6: function GAUSSPDF(z)
7:     // Gauss probability density function
8:     return  $\exp(-0.5 \odot z^{\odot 2}) \odot \sqrt{2\pi}$ 
9: end function
10:
11: function EXPECTEDIMPROVEMENT( $m, s, \eta$ )
12:      $\zeta \leftarrow (\eta - m) \odot s$ 
13:     return  $s \odot [\zeta \odot \text{GAUSSCDF}(\zeta) + \text{GAUSSPDF}(\zeta)]$ 
14: end function

```

---



## Bibliography

---

- [1] R. Adler. *The Geometry of Random Fields*. Wiley, 1981 (cit. on p. 103).
- [2] S.-I. Amari. “Natural Gradient Works Efficiently in Learning.” In: *Neural Comput.* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667. DOI: [10.1162/089976698300017746](https://doi.org/10.1162/089976698300017746). URL: <http://dx.doi.org/10.1162/089976698300017746> (cit. on p. 43).
- [3] L. Armijo. “Minimization of functions having Lipschitz continuous first partial derivatives.” In: *Pacific Journal of Mathematics* 16.1 (1966), pp. 1–3 (cit. on p. 53).
- [4] P. Auer. “Using confidence bounds for exploitation-exploration trade-offs.” In: *Journal of Machine Learning Research* 3 (2003), pp. 397–422 (cit. on p. 64).
- [5] L. Balles and P. Hennig. “Follow the Signs for Robust Stochastic Optimization.” In: *ArXiv e-prints* (May 2017). arXiv: [1705.07774](https://arxiv.org/abs/1705.07774) [cs.LG] (cit. on pp. 11, 92, 123).
- [6] L. Balles, M. Mahsereci, and P. Hennig. “Automating Stochastic Optimization with Gradient Variance Estimates.” In: *ICML AutoML Workshop* (2017) (cit. on pp. 13, 71, 72, 123).
- [7] L. Balles, J. Romero, and P. Hennig. “Coupling Adaptive Batch Sizes with Learning Rates.” In: *ArXiv e-prints* (Dec. 2016). arXiv: [1612.05086](https://arxiv.org/abs/1612.05086) [cs.LG] (cit. on pp. 121, 124).
- [8] T. D. Barfoot, C. H. Tong, and S. Särkkä. “Batch Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression.” In: *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*. 2014. URL: <http://www.roboticsproceedings.org/rss10/p01.html> (cit. on p. 27).
- [9] T. Bayes. “On a problem in the doctrine of chances.” In: *Philosophical Transactions of the Royal Society* (1763) (cit. on p. 17).
- [10] S. Becker and Y. LeCun. “Improving the Convergence of Back-Propagation Learning with Second-Order Methods.” In: *Proc. of the 1988 Connectionist Models Summer School*. Ed. by D. Touretzky, G. Hinton, and T. Sejnowski. San Mateo: Morgan Kaufman, 1989, pp. 29–37 (cit. on p. 148).
- [11] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. “Algorithms for hyper-parameter optimization.” In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554 (cit. on p. 31).
- [12] A. C. Berry. “The accuracy of the Gaussian approximation to the sum of independent variates.” In: *Transactions of the American Mathematical* 49.49 (1941), pp. 122–136 (cit. on p. 65).
- [13] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on pp. 33, 77, 78).
- [14] L. Bottou. “Large-scale machine learning with stochastic gradient descent.” In: *Proceedings of the 19th Int. Conf. on Computational Statistics (COMPSTAT)*. Springer, 2010, pp. 177–186 (cit. on p. 99).
- [15] L. Bottou, F. E. Curtis, and J. Nocedal. “Optimization Methods for Large-Scale Machine Learning.” In: *ArXiv e-prints* (June 2016). arXiv: [1606.04838](https://arxiv.org/abs/1606.04838) [stat.ML] (cit. on p. 52).
- [16] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ Press, 2004 (cit. on p. 36).

- [17] T. Broderick, N. Boyd, A. Wibisono, A. Wilson, and M. Jordan. “Streaming Variational Bayes.” In: *Advances in Neural Information Processing Systems (NIPS 26)*. 2013, pp. 1727–1735 (cit. on p. 99).
- [18] C. Broyden. “A class of methods for solving nonlinear simultaneous equations.” In: *Math. Comp.* 19.92 (1965), pp. 577–593 (cit. on p. 46).
- [19] C. Broyden. “A new double-rank minimization algorithm.” In: *Notices of the AMS* 16.4 (1969), p. 670 (cit. on pp. 11, 47).
- [20] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. “A Stochastic Quasi-Newton Method for Large-Scale Optimization.” In: *ArXiv e-prints* (Jan. 2014). arXiv: [1401.7020](https://arxiv.org/abs/1401.7020) [math.OC] (cit. on pp. 52, 161, 162).
- [21] A.-L. Cauchy. “Méthode générale pour la résolution des systèmes d’équations simultanées.” In: *Compte Rendu des S’éances de L’Acad’emie des Sciences XXV S’erie A.25* (Oct. 1847), pp. 536–538 (cit. on pp. 6, 39).
- [22] C.-C. Chang and C.-J. Lin. *LIBSVM: A library for support vector machines*. 2011. URL: <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html> (cit. on pp. 83, 88).
- [23] H.-S. Chang, E. Learned-Miller, and A. McCallum. “Active Bias: Training More Accurate Neural Networks by Emphasizing High Variance Samples.” In: *ArXiv e-prints* (Apr. 2017). arXiv: [1704.07433](https://arxiv.org/abs/1704.07433) [stat.ML] (cit. on p. 35).
- [24] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. “Entropy-SGD: Biasing Gradient Descent Into Wide Valleys.” In: *CoRR* abs/1611.01838 (2016). URL: <http://arxiv.org/abs/1611.01838> (cit. on pp. 35, 84).
- [25] R. Cox. “Probability, frequency and reasonable expectation.” In: *American Journal of Physics* 14.1 (1946), pp. 1–13 (cit. on p. 17).
- [26] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. “Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization.” In: *Proceedings of the 27th International Conference on Neural Information Processing Systems. NIPS’14*. Montreal, Canada: MIT Press, 2014, pp. 2933–2941. URL: <http://dl.acm.org/citation.cfm?id=2969033.2969154> (cit. on pp. 34, 50, 83, 161–163).
- [27] A. DeMoivre. *The Doctrine of Chances. A Method of Calculating the Probabilities of Events in Play*. 2nd. Woodfall, 1738 (cit. on p. 65).
- [28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database.” In: *CVPR09*. 2009 (cit. on p. 123).
- [29] J. Dennis. “On some methods based on Broyden’s secant approximations.” In: *Numerical Methods for Non-Linear Optimization*. Dundee, 1971 (cit. on pp. 11, 47).
- [30] J. J. Dennis and J. Moré. “Quasi-Newton methods, motivation and theory.” In: *SIAM Review* 19.1 (1977), pp. 46–89 (cit. on pp. 45–47, 54).
- [31] P. Diaconis. “Bayesian numerical analysis.” In: *Statistical decision theory and related topics IV.1* (1988), pp. 163–175 (cit. on p. 9).
- [32] P. Diaconis and M. Shahshahani. “The subgroup algorithm for generating uniform random variables.” In: *Probability in Engineering and Informational Sciences* 1.15-32 (1987), p. 40 (cit. on p. 84).
- [33] Z. Drezner and G. Wesolowsky. “On the computation of the bivariate normal integral.” In: *Journal of Statistical Computation and Simulation* 35.1-2 (1990), pp. 101–107 (cit. on p. 106).



- [34] J. Duchi, E. Hazan, and Y. Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159 (cit. on p. 51).
- [35] A. Einstein. “Die Grundlage der allgemeinen Relativitätstheorie.” In: *Annalen der Physik* 354.7 (1916), pp. 770–822 (cit. on pp. 145, 175).
- [36] A. Einstein. “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen.” In: *Annalen der Physik* 322 (1905), pp. 549–560 (cit. on p. 26).
- [37] C.-G. Esseen. “On the Liapounoff limit of error in the theory of probability.” In: *Ark. Mat. Astr. Fysik* 28A.9 (1942), pp. 1–19 (cit. on p. 65).
- [38] Euclid and R. Fitzpatrick. *Euclid’s Elements*. Ed. by J. Heiberg. Trans. by R. Fitzpatrick. Richard Fitzpatrick, 2009, p. 545 (cit. on p. 6).
- [39] W. Feller. *An Introduction to Probability Theory and Its Applications*. 2nd. Vol. 2. John Wiley & Sons, Inc., 1971 (cit. on p. 65).
- [40] R. Fletcher. “A new approach to variable metric algorithms.” In: *The Computer Journal* 13.3 (1970), p. 317 (cit. on pp. 11, 47).
- [41] R. Fletcher and M. Powell. “A rapidly convergent descent method for minimization.” In: *The Computer Journal* 6.2 (1963), pp. 163–168 (cit. on p. 47).
- [42] R. Fletcher and C. Reeves. “Function minimization by conjugate gradients.” In: *The Computer Journal* 7.2 (1964), pp. 149–154 (cit. on pp. 48, 100).
- [43] A. George and W. Powell. “Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming.” In: *Machine Learning* 65.1 (2006), pp. 167–198 (cit. on p. 99).
- [44] D. Goldfarb. “A family of variable metric updates derived by variational means.” In: *Math. Comp.* 24.109 (1970), pp. 23–26 (cit. on pp. 11, 47).
- [45] G. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins Univ Pr, 1996 (cit. on p. 158).
- [46] J. González, Z. Dai, P. Hennig, and N. D. Lawrence. “Batch Bayesian Optimization via Local Penalization.” In: *ArXiv e-prints* (May 2015). arXiv: 1505.08052 [stat.ML] (cit. on p. 64).
- [47] J. González, M. Osborne, and N. D. Lawrence. “GLASSES: Relieving The Myopia Of Bayesian Optimisation.” In: *ArXiv e-prints* (Oct. 2015). arXiv: 1510.06299 [stat.ML] (cit. on p. 64).
- [48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016 (cit. on pp. 30, 33, 34, 36, 77, 78, 121).
- [49] J. Greenstadt. “Variations on variable-metric methods.” In: *Math. Comp* 24 (1970), pp. 1–22 (cit. on p. 47).
- [50] A. Grigorievskiy and J. Karhunen. “Gaussian Process Kernels for Popular State-Space Time Series Models.” In: *ArXiv e-prints* (Oct. 2016). arXiv: 1610.08074 [stat.ML] (cit. on p. 27).
- [51] A. Grigorievskiy, N. Lawrence, and S. Särkkä. “Parallelizable sparse inverse formulation Gaussian processes (SpInGP).” In: *ArXiv e-prints* (Oct. 2016). arXiv: 1610.08035 [stat.ML] (cit. on p. 27).
- [52] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. “Result Analysis of the NIPS 2003 Feature Selection Challenge.” In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, 2005, pp. 545–552 (cit. on p. 112).

- [53] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I – Nonstiff Problems*. Springer, 1987 (cit. on p. 114).
- [54] J. Hartikainen and S. Särkkä. “Kalman filtering and smoothing solutions to temporal Gaussian process regression models.” In: *IEEE International Workshop on Machine Learning for Signal Processing (MLSP), 2010*. 2010, pp. 379–384 (cit. on p. 27).
- [55] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on p. 78).
- [56] P. Hennig. “Probabilistic Interpretation of Linear Solvers.” In: *SIAM J on Optimization* 25.1 (2015), pp. 210–233 (cit. on pp. 9, 11, 12, 58, 59, 145, 151, 152, 154–156, 158).
- [57] P. Hennig and M. Kiefel. “Quasi-Newton methods – a new direction.” In: *International Conference on Machine Learning (ICML)*. 2012 (cit. on pp. 9, 11, 146).
- [58] P. Hennig, M. Osborne, and M. Girolami. “Probabilistic numerics and uncertainty in computations.” In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 471.2179 (2015) (cit. on pp. 9, 57, 170).
- [59] P. Hennig and C. Schuler. “Entropy Search for Information-Efficient Global Optimization.” In: *Journal of Machine Learning Research* 13 (June 2012), pp. 1809–1837 (cit. on p. 64).
- [60] J. Hensman, M. Rattray, and N. Lawrence. “Fast variational inference in the conjugate exponential family.” In: *Advances in Neural Information Processing Systems (NIPS 25)*. 2012, pp. 2888–2896 (cit. on p. 99).
- [61] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. “Predictive Entropy Search for Efficient Global Optimization of Black-box Functions.” In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 918–926. URL: <http://papers.nips.cc/paper/5324-predictive-entropy-search-for-efficient-global-optimization-of-black-box-functions.pdf> (cit. on p. 64).
- [62] M. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems.” In: *Journal of Research of the National Bureau of Standards* 49.6 (1952), pp. 409–436 (cit. on p. 47).
- [63] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors.” In: *ArXiv e-prints* (July 2012). arXiv: 1207.0580 (cit. on p. 34).
- [64] G. Hinton. “A Practical Guide to Training Restricted Boltzmann Machines.” In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by G. Montavon, G. B. Orr, and K.-R. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599–619. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8\_32. URL: [http://dx.doi.org/10.1007/978-3-642-35289-8\\_32](http://dx.doi.org/10.1007/978-3-642-35289-8_32) (cit. on p. 121).
- [65] G. Hinton and T. Sejnowski. “Optimal Perceptual Inference.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (1983) (cit. on p. 30).
- [66] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory.” In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: <http://dx.doi.org/10.1162/neco.1997.9.8.1735> (cit. on p. 30).

- [67] M. Hoffman, D. Blei, C. Wang, and J. Paisley. “Stochastic variational inference.” In: *Journal of Machine Learning Research* 14.1 (2013), pp. 1303–1347 (cit. on pp. 99, 123).
- [68] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. eprint: <http://www.pnas.org/content/79/8/2554.full.pdf>. URL: <http://www.pnas.org/content/79/8/2554.abstract> (cit. on p. 30).
- [69] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential model-based optimization for general algorithm configuration.” In: *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523 (cit. on p. 31).
- [70] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” In: *CoRR* abs/1502.03167 (2015). URL: <http://arxiv.org/abs/1502.03167> (cit. on p. 34).
- [71] D. Jones, M. Schonlau, and W. Welch. “Efficient global optimization of expensive black-box functions.” In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492 (cit. on pp. 64, 104).
- [72] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems.” In: *Journal of Fluids Engineering* 82.1 (1960), pp. 35–45 (cit. on p. 23).
- [73] I. Karatzas and S. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, 1991 (cit. on pp. 20, 24, 26).
- [74] D. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization.” In: *CoRR* abs/1412.6980 (2014) (cit. on pp. 11, 51).
- [75] A. Kolmogorov. “Grundbegriffe der Wahrscheinlichkeitsrechnung.” In: *Ergebnisse der Mathematik und ihrer Grenzgebiete* 2 (1933) (cit. on p. 17).
- [76] A. Krizhevsky and G. Hinton. “Learning multiple layers of features from tiny images.” In: (2009). URL: <https://www.cs.toronto.edu/~kriz/cifar.html> (cit. on pp. 112, 123).
- [77] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 25. 2012, pp. 1097–1105 (cit. on pp. 35, 78).
- [78] A. Krogh and J. A. Hertz. “A simple weight decay can improve generalization.” In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 4. 1991, pp. 950–957 (cit. on pp. 34, 78).
- [79] H. Kushner. “A New Method of Locating the Maximum Point of an Arbitrary Multipipeak Curve in the Presence of Noise.” In: *Transactions of ASME, Series D, Journal of Basic Engineering* 86 (1964), pp. 97–106 (cit. on pp. 63, 64).
- [80] P. Laplace. “Mémoire sur la probabilité des causes par les évènements.” In: *Mémoires de mathématique et de physique présentés à l’Académie royale des sciences, par divers savans, et lûs dans ses assemblées* 6 (1774), pp. 621–656 (cit. on p. 17).
- [81] Y. LeCun, L. Bottou, G. Orr, and K. Müller. “Efficient BackProp.” In: *Lecture notes in Computer Science* (1998), pp. 9–50 (cit. on pp. 32, 36).
- [82] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 83, 112).

- [83] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. "Object Recognition with Gradient-Based Learning." In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. ISBN: 978-3-540-46805-9. DOI: [10.1007/3-540-46805-6\\_19](https://doi.org/10.1007/3-540-46805-6_19). URL: [https://doi.org/10.1007/3-540-46805-6\\_19](https://doi.org/10.1007/3-540-46805-6_19) (cit. on p. 30).
- [84] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." In: *ArXiv e-prints* (Mar. 2016). arXiv: [1603.06560](https://arxiv.org/abs/1603.06560) [cs.LG] (cit. on p. 31).
- [85] C. van Loan. "The ubiquitous Kronecker product." In: *J of Computational and Applied Mathematics* 123 (2000), pp. 85–100 (cit. on pp. 59, 175).
- [86] D. Maclaurin, D. Duvenaud, and R. P. Adams. *Early Stopping is Nonparametric Variational Inference*. Tech. rep. arXiv:1504.01344 [stat.ML]. 2015 (cit. on p. 79).
- [87] D. Maclaurin, D. Duvenaud, and R. P. Adams. "Early Stopping is Nonparametric Variational Inference." In: *ArXiv e-prints* stat.ML (1504.01344 2015) (cit. on p. 35).
- [88] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig. *Early Stopping without a Validation Set*. 2017. eprint: [arXiv:1703.09580](https://arxiv.org/abs/1703.09580) (cit. on pp. 13, 77).
- [89] M. Mahsereci and P. Hennig. "Probabilistic Line Searches for Stochastic Optimization." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 28. 2015, pp. 181–189 (cit. on pp. 13, 52, 99, 113).
- [90] M. Mahsereci and P. Hennig. "Probabilistic Line Searches for Stochastic Optimization." In: *Journal of Machine Learning Research* 18.119 (2017), pp. 1–59. URL: <http://jmlr.org/papers/v18/17-049.html> (cit. on pp. 13, 52, 69, 99).
- [91] J. Martens. "Deep learning via Hessian-free optimization." In: *International Conference on Machine Learning (ICML)*. 2010 (cit. on pp. 52, 112).
- [92] J. Martens. "New insights and perspectives on the natural gradient method." In: *ArXiv e-prints* (Dec. 2014). arXiv: [1412.1193](https://arxiv.org/abs/1412.1193) [cs.LG] (cit. on p. 43).
- [93] J. Močkus. "On Bayesian Methods for Seeking the Extremum." In: *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*. Ed. by G. I. Marchuk. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 400–404. ISBN: 978-3-540-37497-8. DOI: [10.1007/3-540-07165-2\\_55](https://doi.org/10.1007/3-540-07165-2_55). URL: [https://doi.org/10.1007/3-540-07165-2\\_55](https://doi.org/10.1007/3-540-07165-2_55) (cit. on p. 63).
- [94] N. Morgan and H. Bourlard. "Generalization and parameter estimation in feedforward nets: Some experiments." In: *Proceedings of the 2nd International Conference on Neural Information Processing Systems*. MIT Press. 1989, pp. 630–637 (cit. on pp. 35, 78).
- [95] P. Moritz, R. Nishihara, and M. I. Jordan. "A Linearly-Convergent Stochastic L-BFGS Algorithm." In: *ArXiv e-prints* (Aug. 2015). arXiv: [1508.02087](https://arxiv.org/abs/1508.02087) [math.OC] (cit. on p. 52).
- [96] L. Nazareth. "A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms." In: *SIAM J Numerical Analysis* 16.5 (1979), pp. 794–800 (cit. on p. 48).
- [97] Y. Nesterov. "A method of solving a convex programming problem with convergence rate  $O(1/\sqrt{k})$ ." In: *Soviet Mathematics Doklady* 27 (1983), pp. 372–376 (cit. on p. 40).
- [98] I. Newton. *Philosophiæ naturalis principia mathematica*. Ed. by C. R. 3rd. Innys, 1726 (cit. on p. 42).
- [99] J. Nocedal. "Updating quasi-Newton matrices with limited storage." In: *Math. Comp.* 35.151 (1980), pp. 773–782 (cit. on p. 161).

- [100] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Verlag, 1999 (cit. on pp. 10, 36, 37, 45, 47, 52, 53, 100).
- [101] A. O’Hagan. “Some Bayesian Numerical Analysis.” In: *Bayesian Statistics 4* (1992), pp. 345–363 (cit. on p. 9).
- [102] B. Pearlmutter. “Fast exact multiplication by the Hessian.” In: *Neural Computation* 6.1 (1994), pp. 147–160 (cit. on p. 52).
- [103] H. Poincaré. *Calcul des probabilités*. Paris: Gauthier-Villars, 1896 (cit. on p. 9).
- [104] G. Pólya. “Über den zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung und das Momentenproblem.” In: *Mathematische Zeitschrift, Band 15* Band 15 (1922) (cit. on p. 65).
- [105] B. T. Polyak. “Some methods of speeding up the convergence of iteration methods.” In: *U.S.S.R. Comput. Math. Math. Phys.* 4 (5 1964), pp. 1–17 (cit. on pp. 11, 40).
- [106] M. Powell. “A new algorithm for unconstrained optimization.” In: *Nonlinear Programming*. Ed. by O. L. Mangasarian and K. Ritter. AP, 1970 (cit. on p. 47).
- [107] L. Prechelt. “Early Stopping — But When?” In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by G. Montavon, G. B. Orr, and K.-R. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. ISBN: 978-3-642-35289-8. DOI: [10.1007/978-3-642-35289-8\\_5](https://doi.org/10.1007/978-3-642-35289-8_5) (cit. on pp. 35, 78).
- [108] R. Rajesh, W. Chong, D. Blei, and E. Xing. “An Adaptive Learning Rate for Stochastic Variational Inference.” In: *30th International Conference on Machine Learning (ICML)*. 2013, pp. 298–306 (cit. on p. 100).
- [109] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT, 2006 (cit. on pp. 20, 21, 102, 103).
- [110] H. E. Rauch, C. Striebel, and F. Tung. “Maximum likelihood estimates of linear dynamic systems.” In: *AIAA journal* 3.8 (1965), pp. 1445–1450 (cit. on p. 26).
- [111] R. Reed. “Pruning algorithms—a survey.” In: *IEEE transactions on Neural Networks* 4.5 (1993), pp. 740–747 (cit. on pp. 35, 78).
- [112] W. Davidon. *Variable metric method for minimization*. Tech. rep. Argonne National Laboratories, Ill., 1959 (cit. on pp. 46, 47).
- [113] H. Robbins and S. Monro. “A stochastic approximation method.” In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407 (cit. on pp. 7, 11, 49).
- [114] F. Rosenblatt. “The Perceptron—A Perceiving and Recognizing Automaton.” In: *Cornell Aeronautical Laboratory Report 85-460-1* (1957) (cit. on p. 30).
- [115] N. Roux and A. Fitzgibbon. “A fast natural Newton method.” In: *27th International Conference on Machine Learning (ICML)*. 2010, pp. 623–630 (cit. on p. 100).
- [116] D. Rumelhart, G. Hinton, and R. Williams. “Learning representations by back-propagating errors.” In: *Nature* 323.6088 (1986), pp. 533–536 (cit. on pp. 11, 32, 40).
- [117] S. Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013 (cit. on pp. 22, 25).
- [118] S. Särkkä. “Recursive Bayesian Inference on Stochastic Differential Equations.” PhD thesis. Helsinki University of Technology, 2006 (cit. on p. 24).
- [119] T. Schaul, S. Zhang, and Y. LeCun. “No more pesky learning rates.” In: *30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 343–351 (cit. on p. 100).



- [120] N. Schraudolph. “Fast curvature matrix-vector products for second-order gradient descent.” In: *Neural Computation* 14.7 (2002), pp. 1723–1738 (cit. on p. 52).
- [121] A. Shah, A. Wilson, and Z. Ghahramani. “Student-t Processes as Alternatives to Gaussian Processes.” In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by S. Kaski and J. Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, Apr. 2014, pp. 877–885. URL: <http://proceedings.mlr.press/v33/shah14.html> (cit. on pp. 63, 65).
- [122] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. “Taking the Human Out of the Loop: A Review of Bayesian Optimization.” In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218. URL: <https://doi.org/10.1109/JPROC.2015.2494218> (cit. on p. 64).
- [123] D. Shanno. “Conditioning of quasi-Newton methods for function minimization.” In: *Math. Comp.* 24.111 (1970), pp. 647–656 (cit. on pp. 11, 47).
- [124] J. Sietsma and R. J. Dow. “Creating artificial neural networks that generalize.” In: *Neural networks* 4.1 (1991), pp. 67–79 (cit. on pp. 35, 78).
- [125] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” In: *CoRR abs/1409.1556* (2014) (cit. on p. 78).
- [126] J. Skilling. “Bayesian solution of ordinary differential equations.” In: *Maximum Entropy and Bayesian Methods, Seattle* (1991) (cit. on p. 9).
- [127] J. Snoek, H. Larochelle, and R. P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2012 (cit. on p. 31).
- [128] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. “Scalable Bayesian Optimization Using Deep Neural Networks.” In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2171–2180. URL: <http://proceedings.mlr.press/v37/snoek15.html> (cit. on p. 65).
- [129] A. Solin and S. Särkkä. “Explicit link between periodic covariance functions and state space models.” In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*. Vol. 33. JMLR Workshop and Conference Proceedings. 2014, pp. 904–912 (cit. on p. 27).
- [130] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design.” In: *International Conference on Machine Learning (ICML)*. 2010 (cit. on pp. 64, 104).
- [131] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html> (cit. on p. 34).
- [132] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. “On the importance of initialization and momentum in deep learning.” In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Vol. 28. JMLR Workshop and Conference Proceedings, 2013 (cit. on p. 112).
- [133] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 78).

- [134] R. Tibshirani. “Regression shrinkage and selection via the lasso.” In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288 (cit. on pp. 34, 78).
- [135] T. Tieleman and G. Hinton. *RMSprop Gradient Optimization*. 2015. URL: [http://www.cs.toronto.edu/%5C~tijmen/csc321/slides/lecture%5C\\_slides%5C\\_lec6.pdf](http://www.cs.toronto.edu/%5C~tijmen/csc321/slides/lecture%5C_slides%5C_lec6.pdf) (cit. on pp. 51, 91).
- [136] C. Van Loan and N. Pitsianis. “Approximation with Kronecker Products.” In: *Linear Algebra for Large Scale and Real Time Applications*. Kluwer Publications, 1993, pp. 293–314 (cit. on p. 175).
- [137] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. “Extracting and composing robust features with denoising autoencoders.” In: *Proceedings of the 25th International Conference on Machine Learning (ICML)*. ACM, 2008, pp. 1096–1103 (cit. on pp. 35, 78).
- [138] G. Wahba. *Spline models for observational data*. CBMS-NSF Regional Conferences series in applied mathematics 59. SIAM, 1990 (cit. on p. 102).
- [139] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. “Regularization of Neural Networks using DropConnect.” In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1058–1066. URL: <http://proceedings.mlr.press/v28/wan13.html> (cit. on p. 34).
- [140] W. H. Wolberg, W. N. Street, and O. L. Mangasarian. *UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set*. Jan. 2011. URL: [http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) (cit. on pp. 83, 112).
- [141] P. Wolfe. “Convergence conditions for ascent methods.” In: *SIAM Review* (1969), pp. 226–235 (cit. on p. 53).
- [142] J. Wu, M. Poloczek, A. G. Wilson, and P. I. Frazier. “Bayesian Optimization with Gradients.” In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.04389 [stat.ML] (cit. on p. 64).
- [143] M. D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method.” In: *CoRR* abs/1212.5701 (2012) (cit. on p. 51).
- [144] T. Zhang. “Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms.” In: *Twenty-first International Conference on Machine Learning (ICML 2004)*. 2004 (cit. on p. 99).
- [145] P. Zhao and T. Zhang. “Stochastic Optimization with Importance Sampling.” In: *ArXiv e-prints* (Jan. 2014). arXiv: 1401.2753 [stat.ML] (cit. on p. 35).
- [146] R. Zhao, W. B. Haskell, and V. Y. F. Tan. “Stochastic L-BFGS: Improved Convergence Rates and Practical Acceleration Strategies.” In: *ArXiv e-prints* (Mar. 2017). arXiv: 1704.00116 [math.OC] (cit. on p. 52).





## Colophon

This document was typeset with  $\text{\LaTeX}$ , using a blend of `classicthesis` developed by André Miede and `tufte-latex`, which is based on Edward Tufte's *Beautiful Evidence*. The design was mostly inspired by the PhD thesis of Edgar D. Klenske who followed Aaron Turon, Christian A. Larsson. Most of the graphics in this thesis were generated using `pgfplots` and `pgf/tikz`.