Probabilistic Linear Algebra for 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

Filip de Roos

aus Täby, Schweden

Tübingen
2021

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

Tag der mündlichen Qualifikation	7.4.2022
Dekan	Prof. Dr. Thilo Stehle
1. Berichterstatter	Prof. Dr. Philipp Hennig
2. Berichterstatter	Prof. Dr. Philipp Berens

Abstract

The emergent field of machine learning has by now become the main proponent of data-driven discovery. Yet, with ever more data, it is also faced with new computational challenges. To make machines "learn", the desired task is oftentimes phrased as an empirical risk minimization problem that needs to be solved by numerical optimization routines. Optimization in ML deviates from the scope of traditional optimization in two regards. First, ML deals with large datasets that need to be subsampled to reduce the computational burden, inadvertently introducing noise into the optimization procedure. The second distinction is the sheer size of the parameter space which severely limits the amount of information that optimization algorithms store. Both aspects together have made first-order optimization routines a prevalent choice for model training in ML. First-order algorithms use only gradient information to determine a step direction and step length to update the parameters. Inclusion of second-order information about the local curvature has a great potential to improve the performance of the optimizer if done efficiently.

Probabilistic curvature estimation for use in optimization is a recurring theme of this thesis and the problem is explored in three different directions that are relevant to ML training.
By iteratively adapting the scale of an arbitrary curvature estimate it is possible to circumvent the tedious work of manually tuning the optimizer's step length during model training. The general form of the curvature estimate naturally extends its applicability to various popular optimization algorithms.
Curvature can also be inferred with matrix-variate distributions by projections of the curvature matrix. Noise can then be captured by a likelihood with non-vanishing width, leading to a novel update strategy that uses the inherent uncertainty to estimate the curvature.
Finally, a new form of curvature estimate is derived from gradient observations of a nonparametric model. It expands the family of viable curvature estimates used in optimization.
An important outcome of the research is to highlight the benefit of utilizing curvature information in stochastic optimization. By considering multiple ways of efficiently leveraging second-order information, the thesis advances the frontier of stochastic optimization and unlocks new avenues for research on the training of large scale ML models.

Kurzfassung

Das aufstrebende Feld des maschinellen Lernens ist mittlerweile zur wichtigsten treibenden Kraft für datengetriebene Entdeckungen geworden. Doch mit immer mehr Daten steht das Feld auch vor immer neuen rechentechnischen Herausforderungen. Damit Maschinen "lernen", wird das zu lösende Problem oft als empirisches Risikominimierungsproblem formuliert, welches anschließend durch numerische Optimierungsroutinen gelöst werden muss. Optimierung im ML unterscheidet sich in seinen Herausforderungen in zweierlei Hinsicht von der traditionellen Optimierung. Erstens hat das ML mit so groSSen Datensätzen zu tun, dass um den Rechenaufwand zu verringern immer nur ein Teil des gesamten Datensatzes zur Verfügung steht. Dadurch flieSSt unbeabsichtigt Rauschen in das Optimierungsverfahren ein. Der zweite Unterschied ist die schiere GröSSe des Parameterraums, der die Menge der speicherbaren Informationen stark reduziert. Beide Aspekte zusammen haben dazu geführt, dass Optimierungsmethoden erster Ordnung zur Methode der Wahl für das Modelltraining im ML geworden sind. Algorithmen erster Ordnung verwenden nur Gradienteninformationen für die Berechnung der Schrittrichtung und Schrittlänge, um die Parameter zu aktualisieren. Das Einbeziehen von Information zweiter Ordnung über die lokale Krümmung hat groSSes Potenzial zur Verbesserung des Optimierers, wenn dies effizient geschieht.

Die probabilistische Krümmungsschätzung für die Optimierung ist das zentrale Thema dieser Arbeit. Es werden drei verschiedenen Richtungen zur Lösung dieses Problems untersucht.
Durch iterative Anpassung der Skala einer beliebigen Krümmungsschätzung ist es möglich die mühsame Arbeit der manuellen Kalibration der Schrittlänge des Optimierers während des Modelltrainings zu umgehen. Die allgemeine Form der Krümmungsschätzung ist allgemein auf verschiedene populäre Optimierungsalgorithmen anwendbar.
Die Krümmung kann auch durch Projektionen der Krümmungsmatrix unter der Annahme einer matrixvariaten Verteilungen geschätzt werden. Das Rauschen kann dann durch eine Likelihood mit nicht verschwindender Breite erfasst werden, was zu einer neuen Aktualisierungsstrategie führt, die die inhärente Unsicherheit nutzt um die Krümmung zu schätzen.
SchlieSSlich wird eine neue Form der Krümmungsschätzung aus Gradientenobservationen eines nichtparametrischen Modells abgeleitet. Sie erweitert die Familie der nützlichen Krümmungsschätzungen, die in der Optimierung verwendet werden.
Ein wichtiges Ergebnis der Forschungsarbeit ist der Nutzen der Krümmungsinformationen für die stochastische Optimierung. Durch die Betrachtungverschiedener Möglichkeiten zur Nutzung von Informationen zweiter Ordnung verschiebt diese Dissertation die Grenzen der stochastischen Optimierung und eröffnet neue Möglichkeiten für die Forschung zum Training von ML-Modellen in groSSem MaSSstab.

Acknowledgement

First off I would like to thank Prof. Philipp Hennig for his enthusiastic and unwavering supervision over the course of my doctorate and for guiding and motivating me to become a better researcher. I would also like to thank the additional members of my thesis advisory committee Prof. Philipp Berens and Prof. Andreas Geiger for helpful guidance outside the realms of my research. My graduation would not have been possible without the help of Dr. Leila Masri and the team of the IMPRS-IS graduate school. A huge thank you also to all the members of the PN group at the MPI, and later MoML at the University of Tübingen, for such a great time both at the office and outside of it. A special thanks goes out to Franziska Weiler for always helping out and keeping my spirits up when confronted with bureaucratic hurdles. A great thanks also to Alexandra Gessner for all the vibrant discussions and for putting up with all my eccentricities over the years. It has been an absolute pleasure to share office.

I am truly grateful to Prof. Thomas Schön and his research team at Uppsala University for hosting me during my stay. It was a fantastic opportunity to learn and connect on so many levels. In particular I want to thank Carl Jidling and Prof. Adrian Wills for providing such a great international research experience.

I would also like to thank my family as well as Waltraud and Wilhelm Scheel for all the support, both local and global over the years. Last but by far not least I want to thank Maren Scheel for her unwavering support and for always brightening my day along with Clara who joined us in the end and showed that there is more to life than science.

Filip de Roos

Contents

Contents . . . vi

Prologue 1

1 Introduction . . . 3

Introduction . . . 3
1.1 Computation in Machine Learning . . . 3
1.2 Computation as Inference . . . 5
1.3 Outline . . . 6
1.4 Publications . . . 7

Preliminaries 9

2 Optimization in Machine Learning . . . 11
2.1 Empirical Risk Minimization . . . 11
2.2 Learning as Optimization . . . 12
Automatic Differentiation . . . 12
Subsampling . . . 13
2.3 Deep Learning . . . 14
2.4 Limitations of Optimization . . . 16

3 Probabilistic Inference . . . 17
3.1 Probability Theory . . . 17
3.2 Gaussian Inference . . . 18
Multivariate Normal Distribution . . . 19
Parametric Regression . . . 22
3.3 Gaussian Processes . . . 22
Kernels . . . 24

4 Mathematical Optimization . . . 27
4.1 First-order Optimization . . . 28
4.2 Second-order Optimization . . . 29
4.3 Linear Algebra . . . 31
4.4 Step Length . . . 32
4.5 Step Direction . . . 33
Preconditioning . . . 33
General Metric . . . 35
4.6 Quasi-Newton Methods . . . 36
A Family of Updates . . . 37
Low-rank Update . . . 39
4.7 Summary . . . 40

Probabilistic Curvature Estimation **41**

5 Probabilistic Metric Adaptation **43**
5.1 Introduction 43
Contributions 44
5.2 Method 44
Probabilistic Model 45
Choice of Covariance 46
Accelerated Gradient Updates 47
5.3 Algorithm 48
Parameters 49
Implementation 50
Computation 51
5.4 Experiments 52
Discussion 58
5.5 Related Work 59
5.6 Conclusion 60
5.7 Future Directions 60

6 Matrix Inference for Curvature Learning **63**
6.1 Introduction 63
6.2 Related Work 65
6.3 Theory 66
6.4 Matrix Inference 66
6.5 Adding Noise 67
6.6 Active Inference 68
6.7 Algorithmic Details 69
6.8 High-Dimensional Modification 71
6.9 Experiments 72
Regression 72
Logistic Regression 73
Deep Learning 74
6.10 Conclusion 75
6.11 Future Directions 75
Constrained Optimization 76

7 Nonparametric Curvature Estimation **79**
7.1 Introduction 79
7.2 Related Work 80
7.3 Theory 81
Gaussian Processes 81
Exploiting Kernel Structure 82
Implementation 86
7.4 Applications 88
Optimization 88
Probabilistic Linear Algebra 89
7.5 Experiments 91
Linear Algebra 91
Nonlinear Optimization 92
Runtime Comparison 92
7.6 Discussion 94

7.7 Future Directions 95

Epilogue **97**

8 Summary and Outlook **99**

Bibliography **101**

Notation **109**

Alphabetical Index **113**

List of Figures

2.1 Visualization of overfitting. 14

3.1 1-d normal distribution 18
3.2 Marginal and conditional Gaussian distribution 19
3.3 Gaussian inference with uncertainty 21
3.4 Gaussian prior 21
3.5 Projection of a Gaussian distribution 21
3.6 Posterior of Gaussian inference 22
3.7 Stationary GP 24
3.8 Dot product GP 25

4.1 GD - Momentum comparison. 29
4.2 Equivalence of metric and preconditioning. 34
4.3 Distance measure with different metrics 35
4.4 Two 1-d examples of the secant condition. 37

5.1 Adaptive learning rate comparison for SGD with momentum. 48
5.2 Probabilistic Polyak step 48
5.3 Adaptive inference procedure on Rosenbrock. 51
5.4 Adaptive learning rate (F)-MNIST 53
5.5 Adaptive learning rate CIFAR-10 53
5.6 Adaptive learning rate comparison for the Adam optimizer. 54
5.7 Adaptive learning rate SVHN 54
5.8 Learning rate comparison for ResNet trained on cifar-100. 55
5.9 Comparison of training accuracy for various learning rate adaptations. 56
5.10 Comparison of test accuracy for various learning rate adaptations. 57

6.1 Comparison of SGD and preconditioned SGD on the linear test problem. 71
6.2 Performance of SGD and P-SGD for an MNIST logistic regression problem. . . . 73
6.3 Training loss of SGD and preconditioned SGD on CIFAR-10. 74
6.4 Test loss for SGD and preconditioned SGD on CIFAR-10. 74
6.5 Evolution of estimated learning rate. 75

7.1 GP gradient kernel and decomposition. 82
7.2 Optimization of a 100-dimensional quadratic function. 91
7.3 Optimization of a 100-dimensiona Rosenbrock function. 92
7.4 Runtime comparison of Cholesky and Woodbury decomposition. 93
7.5 Contours of the Rosenbrock function. 93

List of Tables

2.1 Nonlinearities employed in deep learning 15

5.1 Covariance matrices used for popular first-order optimization algorithms. . . 47

7.1 Required derivatives for gradient inference used in Eq. (7.2). 83
7.2 Examples of dot product kernels where $r = (\boldsymbol{x}_a - \boldsymbol{c})^\top \boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})$. 84
7.3 Examples for stationary kernels where $r = (\boldsymbol{x}_a - \boldsymbol{x}_b)^\top \boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{x}_b)$. 85
7.4 Memory requirements for building the full $\nabla \boldsymbol{K} \nabla'$ matrix versus the decomposition for multiplication. 87
7.5 Quantities required in Eq. (7.14) for Hessian inference with different families of kernels. 89

List of Algorithms

1 Step size adaptation for a provided optimization algorithm. 52

2 Preconditioning stochastic optimization 70

3 $\nabla \boldsymbol{K} \nabla'$-MVM 87

4 GP optimization 90

Prologue

Chapter 1

Introduction

Optimization is a driving force that underlies the fundamental laws of physics. Many occurrences such as the stable state of matter, trajectory of light and movement of objects are phenomena that minimize the exerted energy. Not only the laws of physics but life itself is an ongoing optimization problem continuously improved by evolution and natural selection[1] [29]. Evolution is an optimization process consisting of two parts that drives each species to better fit given their environment. The first part comes from the genetic drift from reproduction which in each new generation further hones and cultivates the traits that have proven beneficial for survival. The second is in the form of mutations which are random by nature and usually of no consequence, but sometimes it can significantly accelerate or completely change the optimization route by developing a new beneficial trait. The first process actually maintains a trajectory and slowly improves over time, making it a safer bet in the long run over the random search that occurs through mutations with occasional dividends. Information about a beneficial trajectory towards an optimum can significantly improve the optimization and should be used whenever possible[2] .

1: Immortalized through the quote "survival of the fittest".

[29]: Darwin (1859), *On the origin of species, 1859*

2: For the continuous optimization in later chapters this will constitute the gradient of a function which provides information about the local slope of the function.

Human history is also littered with optimization that has been paramount for the development of modern civilization [48] and has by now become an integral part of society. Nature is still a great source of inspiration and numerous innovations have resulted from mimicking nature [153]. Modern algorithms draw inspiration from observing behavior in nature [154]. We even try to optimize nature around us through selective breeding to better match properties that we deem beneficial.

[48]: Floudas and Pardalos (2008), *Encyclopedia of Optimization*

[153]: Vincent et al. (2006), 'Biomimetics: its practice and theory'

[154]: Wahde (2008), *Biologically inspired optimization methods: an introduction*

Jumping forward to modern times, optimization is evermore present in society. With the introduction of computers in the 20th century the previous line of optimization abruptly switched gear. By providing mathematicians and engineers with access to hardware that reduced the time of computing solutions, the problems grew in complexity. Computers offered the possibility of optimization through simulation, allowing more scenarios to be explored in search of a specific outcome. The parameters of the simulation could then also be optimized further to achieve a selected outcome[3] . As the complexity grew so also did the expectations of the corresponding optimization algorithms which has catapulted numerical optimization to the forefront of modern engineering [109].

3: A few examples can be to maximize return, minimize risk, find shortest path, find fastest path, minimize energy et.c.

[109]: Nocedal and Wright (2006), *Numerical Optimization*

1.1 Computation in Machine Learning

The recent expansion of data-driven modeling, commonly known as machine learning (ML), brought with it a renewed interest in stochastic

optimization[4] . Compared to the more traditional form of numerical modeling ML avoids the tedious task of programming a physical model for simulation. Instead the computer is tasked with finding a relevant model from available data. ML models define a class of functions with parameters that are trained to make precise predictions by learning from provided data. Finding useful patterns in data is generally a computationally expensive task[5] since the relevant search space of complex models can be enormous.

4: The encountered optimization problems existed before as well but are now of particular interest due to the popularity of ML.

5: Encoding a known invariance can reduce the search space and result in more efficient models and training [17]. Several physical quantities are conserved as a result of invariance through Noether's first theorem [110].

[17]: Bronstein et al. (2021), 'Geometric deep learning: Grids, groups, graphs, geodesics, and gauges'

[110]: Noether (1918), 'Invariante Variationsprobleme'

Several ML concepts are originate from probability theory. A full probabilistic treatment requires reasoning about the generative process of the data and is primarily captured by two quantities in conjunction with a model. The *likelihood* $p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M})$ defines the probability of observing the data under a model $\mathcal{M}$ with parameters $\boldsymbol{\theta}$. Any knowledge about the parameters that are available before observation of $\mathcal{D}$ is captured in the *prior* distribution $p(\boldsymbol{\theta} \mid \mathcal{M})$. Learning is then done through Bayes' theorem which provides a principled way of updating the prior belief of the parameters from observing data.

$$\underbrace{p(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{M})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M})}^{\text{likelihood}}\overbrace{p(\boldsymbol{\theta} \mid \mathcal{M})}^{\text{prior}}}{\underbrace{p(\mathcal{D} \mid \mathcal{M})}_{\text{evidence}}} \tag{1.1}$$

Explicit conditioning on the model $\mathcal{M}$ is important when the evidence of competing models must be considered, otherwise it is usually omitted.

Combining the prior distribution with the likelihood of observing the data results in the *posterior* distribution which contracts around the parameters that best explain the data. A predictive distribution for new data is obtained by integrating the likelihood of the new data weighted by the posterior belief

$$p(\mathcal{D}_* \mid \mathcal{D}, \mathcal{M}) = \int p(\mathcal{D}_* \mid \boldsymbol{\theta}, \mathcal{M})\, p(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{M})\, d\boldsymbol{\theta}. \tag{1.2}$$

Two problems stand in the way of this full Bayesian type of modeling. An explicit expression of the posterior distribution is seldom available so approximations must be employed [14] . As a consequence, the integration in Eq. (1.2) lacks an analytic solution and has to rely on a numerical approximation. Numerical integration, however, is notoriously difficult due to the curse of dimensionality[6] [12] and it often has to rely on sampling techniques for estimating the integrals.

[14]: Bishop (2006), *Pattern recognition and machine learning* §10

[12]: Bellman (1966), 'Dynamic Programming'

6: Integration must cover all probability mass contained in a volume that grows exponentially with the dimensionality of the problem.

Many ML models try to circumvent the problem by estimating the posterior distribution with a single parameter sample instead of a distribution. The task is then to find a parameter $\boldsymbol{\theta}$ of the model that best explains the data and use that for prediction. For a given model it will occur at the mode of the posterior distribution which corresponds to maximizing the numerator of Eq. (1.1).

$$\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M})\, p(\boldsymbol{\theta} \mid \mathcal{M})$$

It is often more efficient to optimize the logarithm of the marginal likelihood which results in the same optimum because the logarithm is a monotonic function.

The learning process has thus been turned intoC an optimization problem closely related to *empirical risk minimization* which is central to large parts of supervised learning. Predictions for new data now use the single parameter value and largely ignores the uncertainty[7] of the prediction

7: Uncertainty is becoming more important as more ML algorithms start interacting with the real world and adversarial attacks as well as new scenarios must be considered.

in the process.

Complex models encountered in ML prove difficult to optimize due to two important factors[8] .

8: These are taken to their extremes in the case of deep learning.

1. Contemporary ML models have a large number of trainable parameters to be able to capture intricate nonlinear relationships in the training data. This results in particularly high-dimensional optimization problems, making traditional optimization routines ill-suited due to storage and computational constraints.
2. The training is further complicated by the large datasets employed for the training. Performing computations with the full dataset results in a prohibitive computational overhead. By only processing a smaller batch of data instead of the whole dataset it is possible to reduce the per-iteration cost and significantly speed up training. A downside of using smaller batches is the stochasticity that arise due to the smaller batch size.

These factors combined with automatic differentiation libraries has led to first-order optimization algorithms playing a prevalent role in the training of ML models. First-order algorithms use gradient information to guide the optimization but they rely on hyperparameters which require tuning for successful training[9] . The tuning is usually done by running the same training procedure with varying hyperparameters which consumes a significant part of the ML pipeline in terms of computational resources. This common approach of optimization in ML indicates that algorithmic improvements can translate into a significant reduction in time to train models and by extension also the energy invested.

9: The optimization algorithm requires optimization.

1.2 Computation as Inference

A promising approach for improving the optimization in ML comes from the advent field of probabilistic numerics [66, 26], where computation itself is treated as probabilistic inference governed by Bayes' theorem. A Computational algorithm is treated as a learning agent (A) that interacts with a problem and iteratively updates its belief about the result from observations.

[66]: Hennig et al. (2015), 'Probabilistic numerics and uncertainty in computations'
[26]: Cockayne et al. (2019), 'Bayesian probabilistic numerical methods'

$$p(A \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid A)p(A)}{p(\mathcal{D})}$$

The agent updates its belief about the solution to a numerical problem according to Bayes' theorem and selects a new point based on the current belief.

New evaluation points are selected based on the updated belief in conjunction with a chosen search policy. Deterministic quantities like the outcome of a computation can also be reasoned about in a probabilistic fashion due to the epistemic uncertainty surrounding the solution. The Bayesian nature of the algorithms means that the agent can have knowledge about the problem a-priori to seeing any data which guides the exploration strategy[10] . Several deterministic numerical algorithms have been identified as probabilistic agents that operate in the limit of vanishing uncertainty[11] around the observation. In principle one could then extend the algorithms to operate under the assumption of non-vanishing uncertainty to capture noise in the process.

Examples include but are not limited to solutions of linear systems, trajectories of ODEs, Bayesian quadrature, optimization.

10: Bayesian decision making provides many plausible strategies like evaluating at the most likely point of solution, maximal reduction of uncertainty or updates that adhere to precision constraints.

11: The likelihood $p(\mathcal{D} \mid A)$ collapses to a δ-distribution around the exact value.

1.3 Outline

The focus of this thesis is to employ probabilistic models to speed up and make the optimization/training of complex machine learning models with large datasets more efficient[12] . Most research is centered around probabilistic numerics [66] by treating numerical algorithms as probabilistic agents that learn useful information about the problem. Noise arising from subsampling is specifically addressed by uncertainty in the likelihood.

12: Mainly supervised learning tasks optimized by empirical risk minimization.

Gaussian inference plays a key role in the developed probabilistic optimization routines. Some familiarity with probability theory, multivariate calculus, optimization and machine learning is expected and will only have a short introduction. Some notation and summary of core concepts that will be important throughout the thesis are presented in the first three chapters with references to further reading highlighted. Chapters 5, 6 and 7 contain the main contributions of the thesis. Each of the chapters are based on a project undertaken over the course of the PhD and will further elaborate on the necessary theory.

Noise from subsampling is assumed to be normal distributed by reasoning of the central limit theorem.

Chapter 2 introduces the basic theory that underlies most of the supervised machine learning that will appear throughout the manuscript. It will explain how learning occurs as a form of optimization, what tools are available to cater the learning and why it becomes difficult compared to traditional optimization.

Chapter 3 provides a short background of probabilistic inference with particular focus on Gaussian inference. The standard parametric form of Gaussian inference is used to present concepts that will be central to later chapters. A short overview of nonparametric models in the form of Gaussian processes is provided to cover the background information required for Ch. 7.

Chapter 4 mainly focuses on general theory relating to unconstrained optimization and establishes the notation for optimization used throughout the thesis. Several aspects relating to curvature estimation are presented and clarifies some important connections between linear algebra and optimization. It also explains the ideas behind several optimization routines and compares their suitability in light of the requirements imposed by ML.

Chapter 5 explores the usage of Gaussian inference to infer the optimizer of a local function approximation and is based on the work in [36]. The proposed framework gives rise to a probabilistic version of the Polyak step and generalizes the update beyond the standard isotropic Euclidean distance to new curvature metrics. A lower bound of the local function must be provided which is difficult to estimate in general. The probabilistic nature of the algorithm allows inclusion of uncertainty[13] to aid the estimation. A simple quadratic approximation of the lower bound is presented which results in a heuristic step-size adaptation for several first-order optimization algorithms used in ML.

[36]: de Roos et al. (2021), 'A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization'

13: Both *epistemic*-uncertainty from not knowing enough about the problem and *aleatoric*-uncertainty from stochasticity can and should be included.

Chapter 6 focuses on probabilistic estimation of the local curvature in the presence of noise. It extends previous work on probabilistic linear algebra [64] to a model that includes noisy observations. The model uses a matrix-variate normal distribution and observes noisy projections of the curvature matrix often encountered in stochastic optimization related to ML. An adaptive preconditioning scheme for asymmetric matrix priors is developed and evaluated on a popular benchmark. The presented research was originally published in [35] but has been extended with some additional theory for symmetric priors with uncertainty.

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

[35]: de Roos and Hennig (2019), 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'

Chapter 7 extends the traditional quadratic curvature estimation to the nonparametric setting of Gaussian processes. The main contribution is a computationally tractable approach for *exact* Gaussian process inference with gradient observations in high-dimensional input spaces. This opens up several application areas for Gaussian process modeling, in particular algorithmic advances in fields such as optimization stands to benefit. Several nonparametric curvature-estimates are developed and analyzed as a proof-of-concept for application in optimization and linear algebra. The research was originally published in [33].

[33]: de Roos et al. (2021), 'High-Dimensional Gaussian Process Inference with Derivatives'

Chapter 8 concludes the thesis with a high-level summary of the content and outline new avenues of research made possible through the contributions of the thesis.

1.4 Publications

The research conducted as a part of my PhD studies has been published in the following articles, of which the last three are relevant to this dissertation.

(I) F. de Roos and P. Hennig. 'Krylov Subspace Recycling for Fast Iterative Least-Squares in Machine Learning'. In: *arXiv preprint arXiv:1706.00241* (2017).

(II) F. de Roos and P. Hennig. 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019 for Ch. 6.

(III) F. de Roos, C. Jidling, A. Wills, T. Schön, and P. Hennig. 'A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization'. In: *arXiv preprint arXiv:2102.10880* (2021) for Ch. 5.

(IV) F. de Roos, A. Gessner, and P. Hennig. 'High-Dimensional Gaussian Process Inference with Derivatives'. In: *International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021 for Ch. 7.

A lot of the work has been the result of close collaboration without whom the basis of this thesis would not exist.

(I) The work of de Roos and Hennig [34] was conducted together with Philipp Hennig who provided the starting point of the main idea and contributed to the writing. All derivations, implementation and a lot of the writing was largely done by me.

(II) Philipp Hennig initiated the work of de Roos and Hennig [35] by formulating the problem, providing sources of the necessary background information and in the end contributed to the writing. Derivations, implementation and writing was largely done by me in collaboration with Philipp Hennig.

(III) The research was conducted during a visit to Uppsala University in collaboration with Carl Jidling, Thomas Schön along with Adrian Wills from the University of Newcastle. The main idea, derivations, implementation and writing was mostly done by me. Carl Jidling contributed to some derivations, illustrations and helped with the writing. Adrian Wills, Thomas Schön and Philipp Hennig provided useful discussion and contributed to the overall presentation.

(IV) de Roos et al. [33] was initiated by me in collaboration with Alexandra Gessner and Philipp Hennig. All the derivations and main idea was provided by me. Alexandra Gessner helped with the implementation and a lot of the presentation and tested the framework on an integration task (which has been omitted from the chapter). Philipp Hennig provided feedback on the final manuscript.

Preliminaries

Chapter 2

Optimization in Machine Learning

Machine learning (ML) has in recent years experienced a surge in popularity due to the possibility of making accurate predictions based on data instead of or combined with parametric models. The field is rapidly evolving with whole books devoted to specific subcategories. A complete overview of the various categories is well beyond the scope of the chapter which instead focuses on background knowledge that will be useful to understand the optimization in later chapters. The chapter will therefore not focus on the various algorithms or fields that exist within machine learning. Instead it will outline the general procedure and assumptions that occur within the field of supervised learning and details regarding the training. These elements will occur throughout the rest of the manuscript and they showcase the general importance of optimization in the field as well as the restrictions that appear when developing optimization algorithms. The latter will play a recurring role in this dissertation.

For a general treatment of the field see
[14]: Bishop (2006), *Pattern recognition and machine learning*
[95]: MacKay (2003), *Information theory, inference and learning algorithms*

2.1 Empirical Risk Minimization

A significant part of machine learning which will be central to this manuscript deals with so called *supervised learning*. In this setting we have a dataset $\mathcal{D}$ that consists of inputs $x \in \mathbb{X}$ and targets $y \in \mathbb{Y}$ for some input and output spaces $\mathbb{X}$, $\mathbb{Y}$ respectively. The goal is to train a predictor $m_{\boldsymbol{\theta}} : \mathbb{X} \to \mathbb{Y}$ with trainable parameters $\boldsymbol{\theta} \in \mathbb{R}^D$, so that meaningful predictions can be made for new unseen input data x_* from the same underlying distribution. We also need to define a *loss function* $\ell : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}$ to assess the performance that measures how much a prediction $\hat{y} = m_{\boldsymbol{\theta}}(x)$ deviates from the true target y.

Other fields include unsupervised learning, self-supervised leraning, reinforcement learning, ranking or compression to name but a few.

For regression with real-valued targets it is common to use the squared distance whereas classification with categorical targets often use cross-entropy.

In more formal terms we assume there is a joint distribution $p(x, y)$ from which the dataset $\mathcal{D}$ has been sampled. We now want to find parameters $\boldsymbol{\theta} \in \mathbb{R}^D$ of the predictor $m_{\boldsymbol{\theta}}$ that minimize the expected loss over this distribution.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(x,y)}[\ell(y, m_{\boldsymbol{\theta}}(x))] \tag{2.1}$$

There is usually no way of calculating the true expectation over $p(x, y)$ since we do not know the underlying data-generating process. Instead we approximate the expected loss with the empirical estimate of the dataset.

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} \ell(y_i, m_{\boldsymbol{\theta}}(x_i)) \tag{2.2}$$

The (x, y)-pairs are assumed to be sampled independent and identically distributed (i.i.d.) from $p(x, y)$.

This is known as the *empirical risk minimization* (ERM) problem which is the primary framework for training models in machine learning. By minimizing the empirical loss we assume and hope that the empirical minimizer will coincide with the true minimizer

The terms loss and risk are used interchangeably depending on the field.

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \approx \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{2.3}$$

so that predictions of $m_{\boldsymbol{\theta}^\star}(\cdot)$ on new inputs x_* from $p(x, y)$ will be mapped to the correct target y_*. In this way the predictor, once trained, is expected to generalize its predictions to new unseen input data from the underlying distribution. Throughout this manuscript we will mainly be dealing with ERM so the subscript $\mathcal{D}$ in eq. (2.2) is generally dropped for brevity and instead we let $\mathcal{L}(\boldsymbol{\theta})$ denote the empirical loss over the training set.

2.2 Learning as Optimization

The predictor $m_{\boldsymbol{\theta}}(\cdot)$ is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$ which can be adjusted to reduce the empirical loss. Equation (2.3) is already phrased as an optimization problem and solving it is what constitutes the training/learning phase of the predictor. In some cases there exists a closed-form expression for the optimal $\boldsymbol{\theta}$ so an exact solution is possible. This is however not true in general and even if a closed-form expression exists it can be infeasible to obtain the solution due to various constraints. When such limitations appear it is necessary to employ approximations or rely on an iterative scheme that improves the performance of $m_{\boldsymbol{\theta}}$ with each iteration. A straightforward way to guide the search for an optimal $\boldsymbol{\theta}$ would be to use the gradient of Eq. (2.1) w.r.t. $\boldsymbol{\theta}$ since it indicates the direction which would locally lead to the highest increase in the loss. One can then simply step in the opposite direction to decrease the loss.

Examples of parameters $\boldsymbol{\theta}$ include the weights of a deep neural network [57] or hyperparameters of a Gaussian process [120].

Computational constraints are most often cited but memory or numerical imprecision are also possible.

This setting is the norm rather than the exception so many fields within ML stand to benefit from improved iterative optimization schemes and approximations.

[57]: Goodfellow et al. (2016), *Deep Learning*
[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning*

Automatic Differentiation

Automatic differentiation frameworks (AD) [10, 96] such as TensorFlow [1] and PyTorch [113] are arguably the greatest benefactor of ML development together with the increased availability of data and specialized hardware such as GPUs and TPUs [75]. AD compiles a computational graph at runtime where each node encodes an operation with inputs along with the corresponding derivative of the operation. Computations are then localized such that a node is only responsible for providing a derivative of its output w.r.t. its inputs. Numerical values of derivatives can then be passed up and down the computational graph. Application of the chain rule of derivatives then provides the derivative of arbitrary nodes all the way down to the leaf nodes[1] . By utilizing automatic differentiation it is possible to construct intricate models with derivative information readily available for the optimization.

[10]: Baydin et al. (2018), 'Automatic differentiation in machine learning: a survey'
[1]: Abadi et al. (2016), 'Tensorflow: A system for large-scale machine learning'
[113]: Paszke et al. (2019), 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'
[75]: Jouppi et al. (2017), 'In-datacenter performance analysis of a tensor processing unit'

To give an intuition of how AD works in a typical ML setting it is useful to look at small example. Assume we have a composite function of the form $\ell(f(g(\theta)))$ which is often encountered in ML and we want the derivative of ℓ w.r.t. θ at $\theta = \bar{\theta}$. The following equations outline a typical AD computation where the first line contains the analytic expression for

1: Nodes that have no children in the computational graph.

For ease of exposition we assume $\theta \in \mathbb{R}$ and all the functions perform a mapping $\mathbb{R} \to \mathbb{R}$.

the derivative and subsequent highlight the local contraction that occurs once a numerical value is available.

$$\begin{aligned}
\left.\frac{\partial \ell}{\partial \theta}\right|_{\bar{\theta}} &= \left.\frac{\partial \ell}{\partial f}\right|_{\bar{f}} \left.\frac{\partial f}{\partial g}\right|_{\bar{g}} \left.\frac{\partial g}{\partial \theta}\right|_{\bar{\theta}} \\
&= \left((d\bar{f}) \cdot \left.\frac{\partial f}{\partial g}\right|_{\bar{g}} \right) \cdot \left.\frac{\partial g}{\partial \theta}\right|_{\bar{\theta}} \\
&= (d\bar{g}) \cdot \left.\frac{\partial g}{\partial \theta}\right|_{\bar{\theta}} \\
&= d\bar{\theta}
\end{aligned} \tag{2.4}$$

In the second line $d\bar{f} = \partial\ell/\partial f$ is absorbed to form $d\bar{g} = \partial\ell/\partial g$ which is required to construct $d\bar{\theta} = \partial\ell/\partial\theta$ in the next step. This way it is possible to keep the computations local and minimize the required memory during runtime.

Once a numerical value has been provided for $\theta = \bar{\theta}$ and a gradient is requested we recursively populate the derivatives with the corresponding numerical values ($d\bar{f}$, $d\bar{g}$ and $d\bar{\theta}$). This local computational structure makes AD a modular framework which provides a rich set of functionality not only limited to ML [100]. The composite structure of the function in the example is often used when constructing models in ML and the previously mentioned predictor would here correspond to $m_\theta = f(g(\theta))$ with the additional loss function ℓ on top for training. This setup allows complex models to be constructed that can easily provide gradient information for iterative optimization routines by using this form of backpropagation [126].

[100]: Margossian (2019), 'A review of automatic differentiation and its efficient implementation'

[126]: Rumelhart et al. (1986), 'Learning representations by back-propagating errors'

Also known as reverse-mode AD.

With gradient information available it is now possible to train the predictor. A training iteration now consists of the following steps:

1. Evaluate the empirical loss $\mathcal{L}(\boldsymbol{\theta}_t)$ of the predictor (Eq. (2.2)).
2. Backpropagate through the computational graph of the predictor to obtain the gradient $\nabla\mathcal{L}(\boldsymbol{\theta}_t)$.
3. Use the gradient to update the model parameters $\boldsymbol{\theta}_t$ for the next iteration.

These steps can be repeated *ad infinitum* or until a suitable convergence criterion has been satisfied. The last step is what constitutes the actual learning and will be the general focus of this dissertation.

Some general background and considerations on this part will be presented in Ch. 4.

Subsampling

Another driving force of the ML surge has been access to more powerful hardware that is better suited for the relevant computations. Better hardware allows the training of more complex models which in turn requires larger training sets to avoid overfitting[2] , see Fig. 2.1. More data in turn drives the need for faster hardware. Several iterations of this cycle has led to the development of powerful GPUs with massive internal memory, specialized hardware such as TPUs [75] and datasets that require Terabytes of storage. Training models on such large datasets are computationally infeasible if the whole dataset has to be processed.

2: Overfitting occurs when the model learns the minute details of the training data which are not important for general prediction, or when there is a discrepancy between the training data and that of the true underlying distribution. An example of overfitting can be seen in Fig. 2.1.

[75]: Jouppi et al. (2017), 'In-datacenter performance analysis of a tensor processing unit'

To avoid the cost of processing the whole dataset it is common use smaller batches, so called mini-batches, during training. A batch consists of a smaller subset of data $\mathcal{B} \subset \mathcal{D}$ such that $|\mathcal{B}| \ll |\mathcal{D}|$ and it is usually

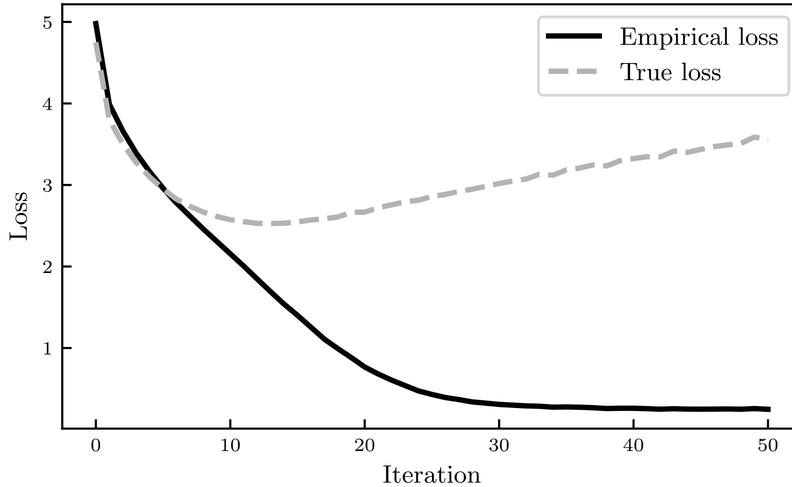


Figure 2.1: Loss curves for a model that overfitted the training data. The empirical loss over the training set keeps decreasing while the true loss reached a minimum and began increasing. Since we are mainly interested in applying the models to new data it would be better to stop the training at iteration 15 than to let empirical loss converge.

sampled i.i.d. from $\mathcal{D}$. The empirical loss is then approximated with the stochastic estimate

Other sampling strategies are possible depending on the problem and model.

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{(x_i, y_i) \in \mathcal{B}} \ell(y_i, m_{\boldsymbol{\theta}}(x_i)). \tag{2.5}$$

If the batch is sampled i.i.d. from the training set then $\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$ is an unbiased estimate of the full empirical loss[3] . The gradient of $\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$ is now also a stochastic estimate of the full gradient that is cheaper to compute allowing faster training of the model.

3: $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$ in Eq. (2.2)

$$\nabla \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{(x_i, y_i) \in \mathcal{B}} \nabla \ell(y_i, m_{\boldsymbol{\theta}}(x_i)) \tag{2.6}$$

A step in the direction $-\nabla \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$ does not necessarily lead to a reduction of $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$, only in expectation. Larger batches will lead to better estimates of the true gradient and there is now a trade-off to be made between the quality of the gradient estimate and the computational cost [7]. This stochasticity along with easy access to gradients has made stochastic optimization an integral part of machine learning and stochastic gradient descent (SGD) [122] one of the most important algorithms of the field.

[7]: Balles et al. (2017), 'Coupling Adaptive Batch Sizes with Learning Rates'

[122]: Robbins and Monro (1951), 'A stochastic approximation method'

2.3 Deep Learning

One of the currently most important fields of machine learning is deep learning (DL) [57] due to its strong empirical results for disparate input spaces such as images [61, 85], text [39, 18], graphs [17] protein folding [76]. The field is mentioned here because it lies at the extreme end of optimization problems encounter in ML with a large number of trainable parameters. Many experiments in the following chapters will be applied to problems involving DL. Therefore a short introduction to the basic building blocks is in order.

[57]: Goodfellow et al. (2016), *Deep Learning*
[61]: He et al. (2016), 'Deep Residual Learning for Image Recognition'
[85]: Krizhevsky (2009), *Learning multiple layers of features from tiny images*
[39]: Devlin et al. (2019), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'
[18]: Brown et al. (2020), 'Language Models are Few-Shot Learners'
[17]: Bronstein et al. (2021), 'Geometric deep learning: Grids, groups, graphs, geodesics, and gauges'
[76]: Jumper et al. (2021), 'Highly accurate protein structure prediction with AlphaFold'

The main idea behind deep learning in a supervised setting is to stack a sequence of computational layers on top of each other to build one large

composite model

$$m_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{h}^{(l)} \circ \boldsymbol{h}^{(l-1)} \circ \cdots \circ \boldsymbol{h}^{(1)}(\boldsymbol{x}). \tag{2.7}$$

Each layer, denoted by the superscript, then has a set of parameters that when combined make up the full set of trainable parameters $\boldsymbol{\theta} \in \mathbb{R}^D$. Since deep learning architectures usually have many layers the number of parameters D quickly grow and easily number in the millions, providing a significant bottleneck for optimization algorithms.

More on the effect of dimensionality in Ch. 4.

Depending on the task and input it is good to use layers that are better suited for the purpose. What follows is a quick overview of some of the most common types of layers that will feature in later chapters.

These layers are presented to convey the general idea behind the architecture and there now exist more powerful layers with better properties [90].

[90]: Liu et al. (2017), 'A survey of deep neural network architectures and their applications'

Dense layer One of the earliest and most common layers is the dense layer. Each layer performs an affine transformation of the previous layer's output followed by a nonlinearity[4] .

Also known as a fully-connected layer.

4: See Tab. 2.1 for examples.

$$\boldsymbol{h}^{(l)}(\boldsymbol{h}^{(l-1)}) = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right)$$

Dense layers takes a vector as input for a single datum and performs a mapping $\mathbb{R}^{d_{\text{in}}} \longrightarrow \mathbb{R}^{d_{\text{out}}}$. The trainable parameters of a dense layer are the matrix $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and bias $\boldsymbol{b}^{(l)} \in \mathbb{R}^{d_{\text{out}}}$.

[105]: Nair and Hinton (2010), 'Rectified linear units improve restricted boltzmann machines'

[24]: Clevert et al. (2016), 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)'

Name	Activation
Sigmoid	$\sigma(z) = 1/(1 + e^{-z})$
Tanh	$\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
ReLU [105]	$\sigma(z) = \max(z, 0)$
ELU [24]	$\sigma_\alpha(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{else} \end{cases}$

Table 2.1: Table of common nonlinearities employed in DL. The first two are known to saturate for high and low values of z leading to the vanishing gradient problem [69]. ReLU avoids the vanishing gradient problem and is arguably the most popular activation function but it can lead to many inactive parameters. ELU tries to alleviate the problems of the previous nonlinearities.

Convolutional layer (CNN) Similar in spirit to the dense layer but better suited for spatial data such as images is the convolutional layer [89]. The matrix multiplication of the dense layer is here replaced by a convolution with filters stored in the tensor $\mathsf{W}^{(l)}$.

[69]: Hochreiter (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions'

[89]: LeCun et al. (1998), 'Gradient-based learning applied to document recognition'

$$\boldsymbol{h}^{(l)}(\boldsymbol{h}^{(l-1)}) = \sigma\left(\mathsf{W}^{(l)} * \boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right)$$

The convolution operation $*$ can also be written as a matrix multiplication but requires reshaping of the data.

Convolutional layers can be applied to many forms of input data making the size and shape of the parameters differ depending on the application. In general the input has a spatial layout and an associated "depth" ($\mathbb{R}^{[d_{\text{in}}] \times d}$). The convolution operation then produces an output that maintains the spatial layout of the input but locally applies a stack of features which changes the depth ($\mathbb{R}^{[\tilde{d}_{in}] \times F}$, where F is the number of features). A typical application of CNNs is in image classification where the input is an RGB image with a width w, height h and depth $d = 3$ color channels. In this case $\mathsf{W}^{(l)} \in \mathbb{R}^{w_c \times w_h \times d \times F}$ matches the spatial layout with a width and height and it performs a 2-d convolution. The bias $\boldsymbol{b} \in \mathbb{R}^F$ behaves similar to the bias of the dense layer and is applied across the spatial layout to each output feature.

$[d_{\text{in}}]$ denotes a general input shape. An image with width w and height h would have $[d_{\text{in}}] = w \times h$.

The altered dimensions $[\tilde{d}_{in}]$ of the spatial layout differs depending on additional parameters such as stride, padding and dilation used by the layer [41].

[41]: Dumoulin and Visin (2016), 'A guide to convolution arithmetic for deep learning'

Convolutional layers are often combined with max-pooling layers which down-samples their input and propagates the maximal feature activation forward to the next layer.

Recurrent layer (RNN) A layer that is more suitable for temporal data is the recurrent layer [43]. It updates the computation of the dense layer to also include a recurrent connection to itself at a previous point in time.

$$\boldsymbol{h}_t^{(l)}(\boldsymbol{h}_t^{(l-1)}, \boldsymbol{h}_{t-1}^{(l)}) = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{U}^{(l)}\boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{b}^{(l)}\right)$$

The input and output of the recurrent layer are similar to the dense layer's but now the recurrent connection matrix $\boldsymbol{U} \in \mathbb{R}^{d_o \times d_o}$ is also included in the trainable parameters. The presented transformation is nowadays considered very basic and borderline obsolete but it captures the general idea of the recurrent connection. More powerful modern alternatives for sequential data include the LSTM [70], GRU [23] and the Transformer [149].

[43]: Elman (1990), 'Finding structure in time'

[70]: Hochreiter and Schmidhuber (1997), 'Long short-term memory'
[23]: Cho et al. (2014), 'Learning phrase representations using RNN encoder-decoder for statistical machine translation'
[149]: Vaswani et al. (2017), 'Attention is all you need'

2.4 Limitations of Optimization

There are several elements of model training in ML that make it problematic, some of which will be explored in later chapters. The two most prominent reasons which also sets ML-training apart from traditional unconstrained optimization are:

1. Large datasets lead to expensive processing of all the data. By using smaller batches the cost can be reduced but it inadvertently leads to stochastic estimates of the loss and gradient which complicates the optimization.
2. The sheer complexity of the models make the storage of information an important aspect to consider in optimization algorithm development. This is particularly prevalent in deep learning which is notoriously parameter-heavy. In such a setting it is important to not use too much storage and a memory requirement that scale beyond $\mathcal{O}(D)$ is considered prohibitive.

Many experiments in later chapters will be of this type.

Chapter 3

Probabilistic Inference

Probability theory naturally arises when we want to reason about correlated quantities when we have additional information about some of them. This information can either come from observational data or in the form of modeling assumptions encoded as prior knowledge. Probability theory then formalizes how the uncertainty of the unknown quantity changes in light of new information. This lies at the core of ML where we want to reason about new unseen test data after having observed data from a training set.

A special branch of probabilistic inference that will be of particular importance in this manuscript is Gaussian inference. Several of the probabilistic models presented in later chapters will rely on properties and assumptions relating to Gaussian inference. This chapter will provide the necessary background information in the form of important aspects and properties that govern probabilistic inference with Gaussian distributions. A basic familiarity with probability theory is assumed and the first section is mainly there to recount a few important properties of probability theory. For a more elaborate exposition the reader is encouraged to consult a book on probability theory [73] or sources dealing with probabilistic machine learning such as the books by MacKay [95], Bishop [14] or Murphy [103].

[73]: Jaynes (2003), *Probability theory: The logic of science*
[95]: MacKay (2003), *Information theory, inference and learning algorithms*
[14]: Bishop (2006), *Pattern recognition and machine learning*
[103]: Murphy (2012), *Machine learning: a probabilistic perspective*

The remainder of the chapter is then dedicated to properties of Gaussian distributions and how they can be used for efficient inference. First the standard form of parametric inference will be discussed which provides the necessary background of Ch. 5 and 6. The next step will be to move from parametric to nonparametric models which will be important in Ch. 7. This leads to the concept of Gaussian processes which are closely linked to kernel methods. Only some basic properties will be presented. Good sources for additional information on these topics are the books of Rasmussen and Williams [120] and Schölkopf and Smola [133].

[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning*
[133]: Schölkopf and Smola (2002), *Learning with kernels: support vector machines, regularization, optimization, and beyond*

3.1 Probability Theory

What follows is a short reminder of basic properties of probability theory which will be exemplified in later sections of the chapter. These rules of probability theory are a result of the axioms of probability[1] suggested by Kolmogorov in 1933 [84]. An extensive exposition of the subject is available in the book by Jaynes [73].

1: A different set of axioms that result in the same theory was proposed by Cox [27].

[84]: Kolmogorov (1933), *Grundbegriffe der Wahrscheinlichkeitsrechnung*
[27]: Cox (1946), 'Probability, frequency and reasonable expectation'
[73]: Jaynes (2003), *Probability theory: The logic of science*

One of the most important aspects of probability theory is the concept of correlation. It measures how much two random variables vary together and is vital for probabilistic inference. In the case of two correlated random variables A and B it is possible to improve the reasoning about

A by gaining information about B and vice versa. Two random variables are said to be correlated when the joint distribution does not factorize into the product of each marginal distribution, $p(A, B) \neq p(A)p(B)$.

A and B are said to be independent if $p(A, B) = p(A)p(B)$.

Two important properties of probability distributions are captured in the sum rule and the product rule informally provided below.

$$p(A) = \int p(A, B)\, dB \tag{S}$$

$$p(A, B) = p(A \mid B)\, p(B) = p(B \mid A)\, p(A) \tag{P}$$

The domain of integration is all possible values of B.
For discrete distributions the integral is replaced with a sum over all possible states of B.

The sum rule states that the probability that one of several outcomes occur is the sum of the individual probabilities and is important for proper normalization of the probabilities. The product rule allows a convenient way of expression the joint probability that two events occur depending on the available information.

By combining these two rules of probability distributions we get *Bayes' theorem* [11]

[11]: Bayes (1763), 'An Essay towards Solving a Problem in the Doctrine of Chances.'

$$p(A \mid B) = \frac{p(A, B)}{p(B)} = \frac{p(B \mid A)p(A)}{p(B)} \quad \text{for } p(B) \neq 0, \tag{B}$$

which is the primary tool for probabilistic reasoning. It relates how our belief over the random variable A changes by learning information about B. The components of Bayes' rule will be further explained in the next section along with important properties of Gaussian distributions.

3.2 Gaussian Inference

The normal distribution is arguably one of the most famous distributions due to its characteristic shape and common application to error-analysis.[2] For a 1-d input, the distribution is defined by a mean $\mu \in \mathbb{R}$ that determines the location of the distribution, and a variance $\sigma^2 \in \mathbb{R}_+$ which controls the shape.

2: Measurements are often assumed to be corrupted by normal distributed noise.

Also known as the bell distribution for its distinct shape or the Gaussian distribution due to its discovery by Carl Friedrich Gauß.
It can also be expressed in terms of the natural parameters μ/σ^2 and $-1/(2\sigma^2)$.

$$p(w) = \mathcal{N}\left(w; \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(w-\mu)^2}{2\sigma^2}\right) \tag{3.1}$$

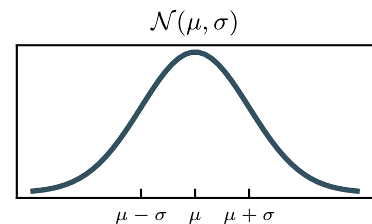


Figure 3.1: Shape of the one-dimensional probability density function for a normal distribution, Eq. (3.1).

The exponential term determines the shape of the distribution and the factor in front is a normalization constant, often denoted Z, that ensures the distribution integrates to 1.

Another reason for the importance of the Gaussian distribution is the central limit theorem (CLT) which states that the empirical mean estimate of a random variable that has been sampled i.i.d. from the same distribution, will approach a normal distribution as the number of samples increase. In empirical risk minimization we try to minimize the mean loss over the dataset and by approximating the loss over a smaller batch we naturally get an approximately normal distributed estimate.

The empirical mean of N i.i.d. random variables $\bar{x} = (x_1 + \ldots + x_N)/N$ approaches $\bar{x} \sim \mathcal{N}(\mu, \sigma^2/N)$ as $N \to \infty$, where $\mu = \mathbb{E}[x]$ and $\sigma^2 = \mathbb{V}[x]$ is the mean and variance of the underlying distribution.

There are also several important analytical and computational properties relating to random variables that have a joint Gaussian distribution that gives it a prevalent position in probabilistic inference.

For two random variables that have a joint Gaussian distribution Eq. (3.1) expands into

$$\begin{aligned} p(w_1, w_2) &= \mathcal{N}\left(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}; \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}\right) \\ &= \frac{1}{Z} \exp\left(-\frac{1}{2} \begin{bmatrix} w_1 - \mu_1 \\ w_2 - \mu_2 \end{bmatrix}^\top \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}^{-1} \begin{bmatrix} w_1 - \mu_1 \\ w_2 - \mu_2 \end{bmatrix}\right), \end{aligned} \tag{3.2}$$

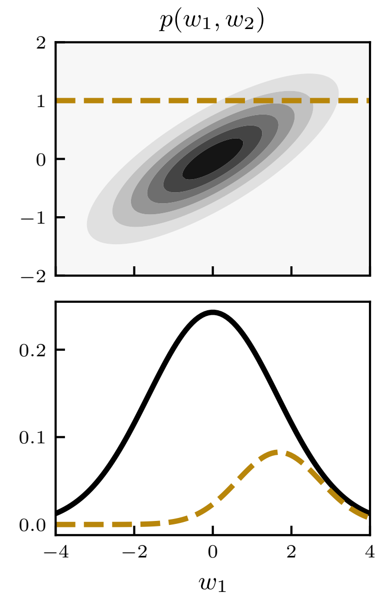


Figure 3.2: Top: joint Gaussian distribution $p(w_1, w_2)$. Bottom: marginal distribution $p(w_1)$ (━━) and conditional distribution $p(w_1 \mid w_2 = 1)$ (- - -). Observing $w_2 = 1$ updates the belief of w_1.

where σ_{11}, σ_{22} encode the marginal variance of the respective random variable *i.e.*

$$\begin{aligned} p(w_1) &= \mathcal{N}(\mu_1, \sigma_{11}), \\ p(w_2) &= \mathcal{N}(\mu_2, \sigma_{22}), \end{aligned}$$

and $\sigma_{12} = \sigma_{21}$ indicates how w_1 and w_2 correlate with each other. A visual presentation of these properties can be seen in Fig. 3.2 where σ_{11} and σ_{22} relate to the width of the distribution in w_1 and w_2 direction respectively and σ_{12} indicates the angle of the distribution. A $\sigma_{12} = 0$ corresponds to an axis-aligned distribution which additionally indicates that the two variables are independent of each other, *i.e.* $p(w_1, w_2) = p(w_1)p(w_2)$. In such a setting w_2 provides no information of w_1 and vice versa. The bottom panel in Fig. 3.2 shows the effect of conditioning the variable w_1 on an observation w_2. By learning information about w_2 the belief over w_1 has changed according to the correlation between them. When two random variables share a joint normal distribution there exists an analytic expression for conditioning one variable on an observation of the other. For the distribution in Eq. (3.2) the posterior belief over w_1 is updated in the following way by making the exact observation $w_2 = W$.

Exact as in we assume there is no uncertainty in the observation.

$$p(w_1 \mid w_2 = W) = \mathcal{N}\left(w_1; \mu_1 + \frac{\sigma_{12}}{\sigma_{22}}(W - \mu_2), \sigma_{11} - \frac{\sigma_{12}^2}{\sigma_{22}}\right) \tag{3.3}$$

This can be shown by inserting the above expression for $p(x, y)$ and $p(y)$ into Bayes' theorem in Eq. (B). See [14] for derivation.
[14]: Bishop (2006), *Pattern recognition and machine learning* §2.3

The expression shows that the posterior distribution over w_1 is again a Gaussian distribution with a shifted mean and reduced variance. Figure 3.2 shows the effect of conditioning a variable on another in a joint normal distribution. In the limit of $\sigma_{12} \to 0$ we see that the observation does not lead an update belief over w_1. This is to be expected since it means that the two random variables w_1 and w_2 are independent.

Multivariate Normal Distribution

Properties of Gaussian inference extend beyond the 2-dimensional example presented so far. In the remainder of the chapter we let the random variable $\boldsymbol{w}$ reside in a D-dimensional continuous space ($\boldsymbol{w} \in \mathbb{R}^D$) in order to generalize the expressions we have seen so far. Since high-dimensional distributions are difficult to visualize on paper we will rely on more

2-dimensional illustrations but the generalization to higher dimensions will be highlighted.

A general D-dimensional normal distribution is characterized by the mean vector $\boldsymbol{\mu} \in \mathbb{R}^D$ and the symmetric nonnegative definite[3] covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ as $\mathcal{N}\left(\boldsymbol{w};\boldsymbol{\mu},\boldsymbol{\Sigma}\right)$. The functional form of the distribution reads

$$\mathcal{N}\left(\boldsymbol{w};\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{w}-\boldsymbol{\mu})\right), \tag{3.4}$$

3: $\boldsymbol{\Sigma}$ must have eigenvalues $\lambda_i \geq 0$ for all i. $\lambda_i = 0$ indicates a rank-deficient distribution so there are directions in which Eq. (3.4) does not change, see Fig. (3.5) for an example.

with the normalization constant now requiring the determinant of $\boldsymbol{\Sigma}$ to evaluate. Most applications found later in the manuscript are focused more on the parameters and location of the distribution and not the actual scale of the distribution, which means that the normalization will be neglected and is just stated here for completeness. There are two properties of Gaussian distributions that will play a significant role throughout the manuscript which warrant some elaboration along with an example.

The first property is that the normal distribution is a conjugate distribution to itself w.r.t. the mean. This means that a Gaussian likelihood combined with a Gaussian prior will result in a joint Gaussian distribution, which by extension leads to a Gaussian posterior distribution.

Conjugate distributions is a property found among exponential families which ensures that the posterior distribution will have the same form as the prior for a conjugate likelihood [103].
[103]: Murphy (2012), *Machine learning: a probabilistic perspective* §9

$$\overbrace{p(\boldsymbol{y}\mid\boldsymbol{w})}^{\mathcal{N}(\boldsymbol{y}|\boldsymbol{w})}\,\overbrace{p(\boldsymbol{w})}^{\mathcal{N}(\boldsymbol{w})} = \overbrace{p(\boldsymbol{y},\boldsymbol{w})}^{\mathcal{N}(\boldsymbol{y},\boldsymbol{w})} \Rightarrow \overbrace{p(\boldsymbol{w}\mid\boldsymbol{y})}^{\mathcal{N}(\boldsymbol{w}|\boldsymbol{y})}$$

The second property is that a linear combination of normal distributed random variables is again normal distributed. A generic formulation of this property is that for a matrix $\boldsymbol{A} \in \mathbb{R}^{M\times D}$ and a Gaussian prior in Eq. (3.4) the distribution of $\boldsymbol{A}\boldsymbol{w}$ is

$$p(\boldsymbol{A}\boldsymbol{w}) = \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top).$$

If now an observable $\boldsymbol{y} \in \mathbb{R}^M$ can be phrased as a linear transformation of the Gaussian random variable $\boldsymbol{w} \in \mathbb{R}^D$ with additive Gaussian noise

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{\varepsilon}, \quad \text{with } \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{R}), \tag{3.5}$$

Note that $\boldsymbol{\varepsilon}$ is here a vector sampled from a multivariate normal distribution with covariance $\boldsymbol{R}$.

then the combination of both properties allows for analytic inference. An example of this is available in Fig. 3.3 and below the setup is explained in more detail.

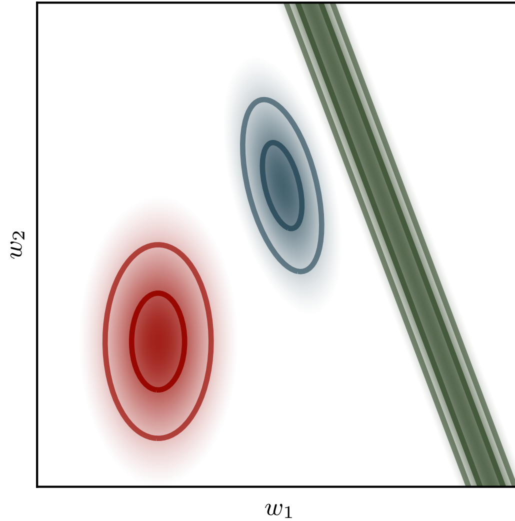


Figure 3.3: Bayesian inference visualized for Gaussian distributions. The elongated prior (▬) indicates a higher uncertainty of w_2 compared to w_1. A projection of the random variable is observed in the likelihood (▬). By conditioning on the observation $\boldsymbol{y}$ the belief of $\boldsymbol{w}$ is updated to the posterior (▬). The relative width of the prior and the likelihood determines the location of the posterior distribution. A decrease in width of either distribution moves the posterior towards the corresponding distribution.

Prior The prior distribution encodes any available knowledge of the parameters *prior* to seeing any data. With a Gaussian distribution it is possible to encode information in the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ respectively

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

The mean $\boldsymbol{\mu}$ can be used to center the distribution around the values we expect to be most probable and $\boldsymbol{\Sigma}$ should encode how sure we are about the estimate, or equivalently, how much the value is expected to deviate. A 2-dimensional multivariate Gaussian distribution can be seen in Fig. 3.4. The axis-aligned elliptic contour levels indicate that the two random variables are independent and that the prior uncertainty of w_2 is higher than of w_1.

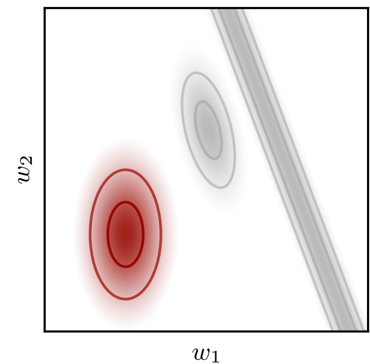


Figure 3.4: Gaussian prior $p(\boldsymbol{w})$

Likelihood The *likelihood* links the random variable of interest to an observable quantity. In Fig. 3.5 an observation as a linear projection of the variable $\boldsymbol{w}$ is highlighted. The generic expression for such a Gaussian likelihood is

$$p(\boldsymbol{y} \mid \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{A}\boldsymbol{w}, \boldsymbol{R}) \tag{3.6}$$

for $\boldsymbol{A} \in \mathbb{R}^{M \times D}$ and $\boldsymbol{R} \in \mathbb{R}^{M \times M}$, which is equivalent to the statement in Eq. (3.5). In general the observation will be an M-dimensional hyperplane in D-dimensional input space and can therefore be a rank-deficient distribution. This is the case in Fig. 3.5 where $M = 1$ and $D = 2$ results in a distribution that has infinitely long contour lines orthogonal to the image of $\boldsymbol{A}$.

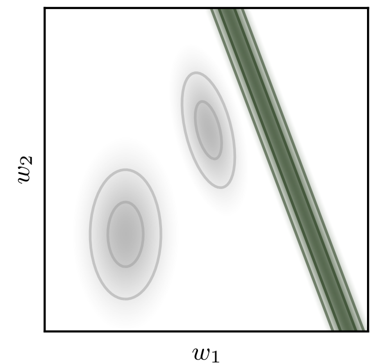


Figure 3.5: Gaussian likelihood $p(\boldsymbol{y} \mid \boldsymbol{w})$

Posterior By observing data in the form of Eq. (3.6), we gain information about which values of $\boldsymbol{w}$ could have produced the observation. The uncertainty about $\boldsymbol{w}$ is then reduced and captured in the *posterior* distribution which is also Gaussian by virtue of the conjugate Gaussian prior and likelihood.

$$p(\boldsymbol{w} \mid \boldsymbol{y}) = \mathcal{N}\left(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}\right), \quad \text{with parameters}$$
$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + \boldsymbol{R})^{-1}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\mu}) \tag{3.7}$$
$$\bar{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + \boldsymbol{R})^{-1}\boldsymbol{A}\boldsymbol{\Sigma}.$$

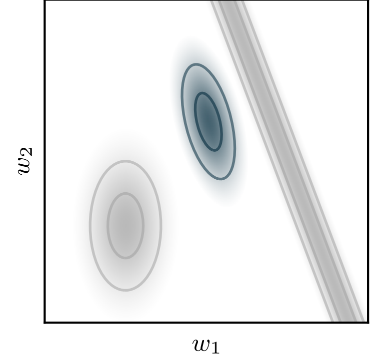


Figure 3.6: Gaussian posterior $p(\boldsymbol{w} \mid \boldsymbol{y})$

Note how the ellipse in Fig. 3.6 aligned with the observation. This is because only information orthogonal to the contour lines of the likelihood was obtained, but none along the contour lines. The posterior distribution also did not move towards the observation in the closest direction but deviated upwards. This is because the anisotropic prior covariance encodes higher uncertainty of w_2 (up-down direction) compared to w_1 (left-right direction).

Parametric Regression

The aforementioned inference scheme is of particular interest for parametric regression where a function is modeled as a linear combination of feature functions stacked into the row vector $\boldsymbol{\phi}(x)$ according to

$$f(x) = \boldsymbol{\phi}(x)\boldsymbol{w}.$$

The features in $\boldsymbol{\phi}(x)$ can also contain trainable parameters $\boldsymbol{\theta}$ which are here omitted for the sake of clarity.

If a set of input-output pairs $\{x_i, y_i\}_{i=1}^N$ has been collected and the observations are obtained with homoscedastic[4] Gaussian noise corresponding to the following model

4: Homogenous variance.

$$y_i = f(x_i) + \varepsilon_i \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

then Eq. (3.7) readily provides the posterior belief of $\boldsymbol{w}$ assuming the prior belief is Gaussian. To arrive at an explicit expression for the posterior we just have to replace $\boldsymbol{A}$ with the stacked feature matrix $\boldsymbol{\Phi}$ where row i contains the features evaluated at x_i. The posterior mean of Eq. (3.7) now reads

$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I})^{-1}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}) \quad \text{or equivalently}$$
$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu} + (\boldsymbol{\Sigma}^{-1} + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\sigma^{-2}\boldsymbol{\Phi}^\top(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}),$$

Both expressions are equivalent but with different computational cost depending on N and D. If $N < D$ then the first expression is cheaper to evaluate due to the smaller size of the matrix in the inverse and vice versa for the second.

which is the closed-form expression of the regularized least-squares estimate.

$$\bar{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{w}} \frac{1}{\sigma^2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|^2 + \|\boldsymbol{w} - \boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}^{-1}} \tag{3.8}$$

3.3 Gaussian Processes

The previous section showcased several properties of the Gaussian distribution that are useful for parametric inference with a finite dimensional input space. These properties can also be extended to infinite dimensional spaces which results in an object called a Gaussian process (GP). We will here consider the regression of a scalar function $f(\boldsymbol{x})$ over inputs $\boldsymbol{x} \in \mathbb{R}^D$. A Gaussian process is a distribution over functions which can be evaluated at arbitrary positions making the input infinite dimensional. The defining property of a Gaussian process is that a function evaluated

A GP is a nonparametric estimator closely related to kernel ridge regression, see [78] for a review.
Inputs are generally not limited to be real valued.

[78]: Kanagawa et al. (2018), 'Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences'
[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning*

at discrete points will have a joint normal distribution [120]. For scalar inference $f : \mathbb{R}^D \to \mathbb{R}$ it is defined by a mean function $\mu(\cdot) : \mathbb{R}^D \to \mathbb{R}$ and kernel function $k(\cdot, \cdot) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$.

$$p(f) = \mathcal{GP}\left(\mu(\cdot), k(\cdot, \cdot)\right)$$

Similar to the parametric case in Sec.3.2, the prior mean $\mu(\cdot)$ moves the distribution and for scalar functions it is best visualized as a shift in function values. The kernel function has a more intricate effect on the model compared to the mean. It encodes similarities between function values based on the input and determines properties of the model function which can greatly improve the prediction quality. When the GP is evaluated on a stacked set of inputs $X \in \mathbb{R}^{N \times D}$ it results in normal distribution with the following mean and covariance

Examples of properties include smoothness, invariances, periodicity et.c.

$$p\left(\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}\right). \quad (3.9)$$

The following expression is equivalent and will be the preferred notation moving forward $p(f_X) = \mathcal{N}\left(\boldsymbol{\mu}_X, \boldsymbol{K}_{XX}\right)$.

The joint normal distribution in Eq. (3.9) makes GPs particularly attractive for regression problems where function observations are corrupted by an additive normal distributed noise term. For a set of observations $\boldsymbol{Y} = \{y\}_{i=1}^N$ with individual observations phrased as either of

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$
$$p(y_i \mid f(\boldsymbol{x}_i)) = \mathcal{N}(f(\boldsymbol{x}_i), \sigma^2)$$

there exists an analytic expression for $p(f_X, \boldsymbol{Y})$. Due to the conjugacy of the normal distribution we get the marginal distribution

This property was outlined in the previous section and can be seen in Fig. 3.3 where the prior is multiplied with the likelihood to obtain the posterior.

$$p(\boldsymbol{Y}) = \mathcal{N}(\mu(X), \boldsymbol{K}_{XX} + \sigma^2 \boldsymbol{I}).$$

It is possible to infer the value of the function based on the noisy observations in $\boldsymbol{Y}$ by constructing the joint distribution. The posterior distribution of f is also a GP which can be evaluated at new points by using the joint Gaussian distribution

$$p\left(\begin{bmatrix} \boldsymbol{Y} \\ f(\boldsymbol{x}_*) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_X \\ \mu(\boldsymbol{x}_*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{XX} + \sigma^2 \boldsymbol{I} & k_{X\boldsymbol{x}_*} \\ k_{\boldsymbol{x}_* X} & k_{\boldsymbol{x}_*\boldsymbol{x}_*} \end{bmatrix}\right).$$

An important property of the covariance matrix, generally and henceforth referred to as the kernel matrix, is $K_{XX} \succeq 0$. A side effect of the positive semi-definiteness (p.s.d.) is that inference can be phrased as a convex optimization problem which can be solved analytically. The posterior predictive distribution of f evaluated at $\boldsymbol{x}_*$ is

This is intuitive since a random variable is perfectly correlated with itself and another variable cannot be more correlated with the first without introducing linearly dependent columns of the covariance.

$$p(f(\boldsymbol{x}_*) \mid \boldsymbol{Y}, X, \boldsymbol{x}_*) = \mathcal{N}(\mu(\boldsymbol{x}_*) + k_{\boldsymbol{x}_*, X}\left(\boldsymbol{K}_{XX} + \sigma^2 \boldsymbol{I}\right)^{-1}(\boldsymbol{Y} - \mu_X),$$
$$k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, X)\left(\boldsymbol{K}_{XX} + \sigma^2 \boldsymbol{I}\right)^{-1} k(X, \boldsymbol{x}_*))$$

Inference with GPs usually has a computational cost that scales cubically $\mathcal{O}(N^3)$, in the number of data points N. There are certain situations where inference can be made cheaper if the data allows it. There also exist

Temporally sorted data with Bayesian filters and grid-structured data are two examples.

various approximations that can be employed to speed up the inference procedure. A thorough review of GPs is beyond the scope of this thesis and the curious reader is referred to the work of Rasmussen and Williams [120] and Quinonero-Candela and Rasmussen [118] for good starting points.

[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning*
[118]: Quinonero-Candela and Rasmussen (2005), 'A unifying view of sparse approximate Gaussian process regression'

Kernels

Kernels can be defined over various input spaces and define properties of the function to be modeled. There exists a plethora of kernels that encode various properties which can be used to improve the predictive abilities of the model. We will here look more closely at two families of kernels that will be of importance in Ch. 7. It will be useful to define the kernels as a function of a scalar quantity, $r(\cdot,\cdot) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, that defines a similarity between the two inputs.

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(r(\boldsymbol{x}, \boldsymbol{x}'))$$

Stationary Some of the most famous kernel functions belong to the family of stationary kernels. These kernels only depend on the relative distance between the inputs, commonly written as $k(\boldsymbol{x}, \boldsymbol{x}') = k(|\boldsymbol{x} - \boldsymbol{x}'|)$. This also means that further from the input the posterior mean will revert to the prior mean. A typical parameterization of stationary kernels for inputs $\boldsymbol{x} \in \mathbb{R}^D$ is to use the Mahalanobis distance [97]

[97]: Mahalanobis (1936), 'On the generalized distance in statistics'

$$r(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{\Lambda} (\boldsymbol{x} - \boldsymbol{x}') \tag{3.10}$$

as similarity measure, but there are many possibilities. The matrix $\boldsymbol{\Lambda}$ can be any valid p.s.d. matrix which can be used to improve the performance of the model. A simple example where this is useful is if the input dimensions drastically vary in magnitude or the target function shows more fluctuations in some dimensions. $\boldsymbol{\Lambda}$ can then be used to rescale respective input dimensions to better capture meaningful variations in the target function. If $\boldsymbol{\Lambda} = \lambda \boldsymbol{I}$, then all input dimensions are treated equally at which point the kernel is also called isotropic.

Any valid distance measure can be used leading to plenty of possible formulations.

An important feature of stationary kernels is that they are shift-invariant, meaning that the kernel takes the same value after an arbitrary shift in input space.[5]

5: Only the relative distance between inputs is of importance, not the absolute location.

$$k(\boldsymbol{x} + \boldsymbol{c}, \boldsymbol{x}' + \boldsymbol{c}) = k(|\boldsymbol{x} + \boldsymbol{c} - \boldsymbol{x}' - \boldsymbol{c}|) = k(|\boldsymbol{x} - \boldsymbol{x}'|) = k(\boldsymbol{x}, \boldsymbol{x}')$$

A few popular kernels that use the distance $r(\cdot,\cdot)$ in Eq. (3.10) are found in the Matérn family of kernels. The kernels are parameterized by the factor ν which determines the smoothness of the functions. The general expression for a kernel with parameter ν of the Matérn family reads

$$k_\nu(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} r\right)^\nu J_\nu(\sqrt{2\nu} r),$$

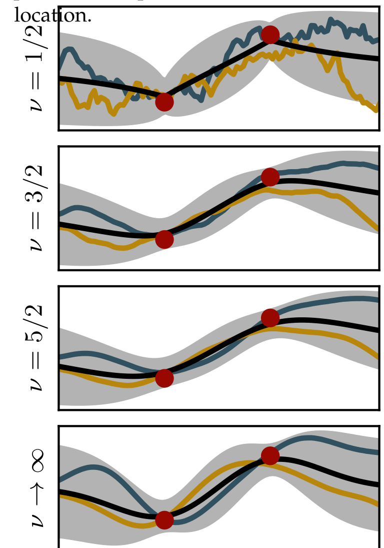


Figure 3.7: GPs with Matérn covariance functions found in Eq. (3.11) for different smoothness parameter ν. Each graph shows the posterior mean (▬) and two samples (▬ / ▬) as well the associated uncertainty in the shaded region.

which requires the gamma function and a modified Bessel function to evaluate [120]. For certain cases that are of particular interest to ML and stochastic differential equations this expression simplifies to contain polynomial and exponential components. When $\nu = p + 1/2$ for

$p = 0, 1, 2, \ldots$ the kernel expression simplifies and a sample from the GP will have p continuous derivatives. Kernel expressions for the most common members of this simplified family are listed in Eq. (3.11) and samples are visualized in Fig. 3.7.

$$
\begin{aligned}
\nu = 1/2: \quad & k(r) = \sigma^2 \exp(-\sqrt{r}) \\
\nu = 3/2: \quad & k(r) = \sigma^2 (1 - \sqrt{3r}) \exp(-\sqrt{3r}) \\
\nu = 5/2: \quad & k(r) = \sigma^2 (1 + \sqrt{5r} + \frac{5}{3} r) \exp(-\sqrt{5r}) \\
\nu \to \infty: \quad & k(r) = \sigma^2 \exp(-r/2)
\end{aligned}
\tag{3.11}
$$

Two of these are particularly famous and are commonly referred to by different names. The first occurs for $\nu = 1/2$ and the kernel is then known as the Laplace kernel due to its similarity to the Laplace distribution. Samples from the process also correspond to paths of the Ornstein-Uhlenbeck process [144, 129]. The second is the limiting function as $\nu \to \infty$ which is known as the exponentiated quadratic, squared exponential, radial-basis function or Gaussian kernel due to its functional form[6] . In this setting the ν parameter suggests that sample functions should have infinite continuous derivatives which is also the case and leads to very smooth samples.

[144]: Uhlenbeck and Ornstein (1930), 'On the theory of the Brownian motion'
[129]: Särkkä and Solin (2019), *Applied stochastic differential equations*

6: Several unfortunate misnomers that are common in the kernel community.

[136]: Shawe-Taylor, Cristianini, et al. (2004), *Kernel methods for pattern analysis*
[42]: Duvenaud (2014), 'Automatic model construction with Gaussian processes'
[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning*

Dot product Another family of kernels that occasionally appear are so called dot product kernels. These kernels are defined by a variant of the similarity measure

$$
r(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{c})^\top \boldsymbol{\Lambda} (\boldsymbol{x}' - \boldsymbol{c}). \tag{3.12}
$$

Sometimes $r(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{\Lambda} \boldsymbol{x}' + c)$ is used instead.

$\boldsymbol{\Lambda}$ is once again a p.s.d. matrix that defines the underlying distance measurement and $\boldsymbol{x} \in \mathbb{R}^D$. These kernels are rotation-invariant around the point $\boldsymbol{c}$ which can easily be seen by considering an arbitrary rotation matrix $\boldsymbol{R} \in \mathbb{R}^{D \times D}$ for the case $\boldsymbol{\Lambda} = \boldsymbol{I}$ and $\boldsymbol{c} = 0$.[7]

7: This looks like a restrictive assumption but is made general by using the auxiliary variable $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{c}$ instead.

A rotation matrix is a matrix with orthonormal columns such that $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}$

$$
k(\boldsymbol{R}\boldsymbol{x}, \boldsymbol{R}\boldsymbol{x}') = k(\boldsymbol{x}^\top \boldsymbol{R}^\top \boldsymbol{I} \boldsymbol{R} \boldsymbol{x}') = k(\boldsymbol{x}^\top \boldsymbol{I} \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}')
$$

One can also define the transformation for a general $\boldsymbol{\Lambda}$ but the transformation invariance is then dependent on the spectrum of $\boldsymbol{\Lambda}$. [8]

8: $\tilde{\boldsymbol{R}} = UD^{-1/2}\boldsymbol{R}$ where U and D are matrices of the eigendecomposition.

Two important examples of kernels that use r from Eq.(3.12) are listed below.

$$
\begin{aligned}
k(r) &= \sigma^2 \frac{r^p}{p!} \\
k(r) &= \sigma^2 \exp(r) = \sigma^2 \sum_{p=0}^{\infty} \frac{r^p}{p!}
\end{aligned}
\tag{3.13}
$$

The first models functions as polynomials of order p [133] while the second generalizes the expression to infinite order. A few examples of these kernels can be seen in Fig. 3.8 for different p.

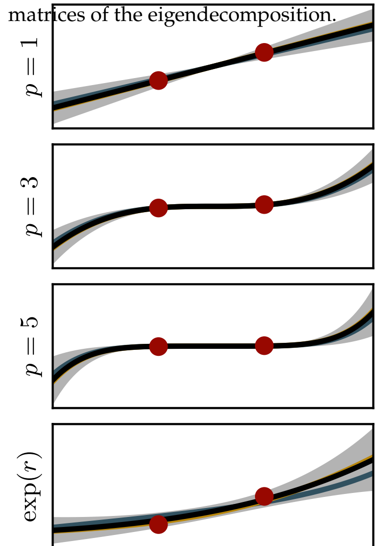


Figure 3.8: GPs with covariance functions found in Eq. (3.13) for different values of p. Each graph shows the posterior mean (▬) and two samples (▬ / ▬) as well as uncertainty in the shaded region. The parameter $\boldsymbol{c}$ is centered in the figure.

Generalization The kernels that have been presented so far might seem restrictive in their modeling possibilities but we have only looked at a small subset of available kernels for the sake of exposition. While

each kernel has its own benefits and limitations the real power of kernel methods lies in the many ways they can be combined for more expressive modeling [136, 42, 120]. A few such properties are listed below for two GPS $f \sim \mathcal{GP}(\mu_f, k_f)$ and $g \sim \mathcal{GP}(\mu_g, k_g)$.

- The sum of two GPS is another GP: $f + g \sim \mathcal{GP}(\mu_f + \mu_g, k_f + k_g)$
- The product of two GPS in another GP: $f \cdot g \sim \mathcal{GP}(\mu_f \cdot \mu_g, k_f \cdot k_g)$
- A GP can be defined over another feature space:

$$f(\psi(\cdot)) \sim \mathcal{GP}(\mu_f(\psi(\cdot)), k_f(\psi(\cdot), \psi(\cdot)))$$

By combining kernels in this way it is possible to encode a lot of prior knowledge that can be used to make better predictions with less data.

Hyperparameters Although GPS are commonly referred to as non-parametric methods they do contain certain hyperparameters which determine the overall shape of the functions they model. One example of such a parameter that often occur in kernel methods is a characteristic lengthscale. In Eq. (3.10) and Eq. (3.12) this lengthscale is encoded in $\mathbf{\Lambda}$ which determines how quickly functions are expected to change in each direction. A proper Bayesian treatment of the hyperparameters by integrating out the effect of the hyperparameter is usually intractable and would have to rely on costly sampling techniques. Instead it is common practice to find parameter values that maximize the logarithm of the marginal likelihood. A benefit of this approach is that it turns the problem of integration into one of optimization. It is also possible to differentiate the logarithm of the marginal likelihood to use gradient information of the hyperparameters to speed up the optimization.

See Ch. 4 for more details on possible optimization routines.

Chapter 4

Mathematical Optimization

So far we have seen that optimization is an important aspect of training models in machine learning. While availability of gradient information is a great benefactor of ML there are other details such as stochastic estimates due to large datasets and large parameter spaces that make its application more difficult. In this chapter we will go over some basic information from traditional mathematical optimization and see how it translates to the setting of ML. Most of the content is based on information from the books of Boyd and Vandenberghe [16], Nocedal and Wright [109] and Dennis Jr and Schnabel [38].

[16]: Boyd and Vandenberghe (2004), *Convex optimization*
[109]: Nocedal and Wright (2006), *Numerical Optimization*
[38]: Dennis Jr and Schnabel (1989), 'A view of unconstrained optimization'

The chapter will focus on optimization in the form of unconstrained minimization of a scalar function $f(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$. That means that the parameter $\boldsymbol{\theta}$ can take any value in $\mathbb{R}^D$ and the goal is to find

$$\arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}). \tag{4.1}$$

The minimization problem can be turned into a maximization problem by $\max f(\boldsymbol{\theta}) \equiv \min -f(\boldsymbol{\theta})$.

In constrained optimization $\boldsymbol{\theta}$ is restricted to a subset of $\mathbb{R}^D$ such as positive values or locations that satisfy an equality or inequality constraint.

In machine learning this function will usually occur as a regularized risk minimization problem (Eq. (2.2)) or as the negative log likelihood of a probabilistic model. The overarching goal is to find a global minimizer of the function, $\boldsymbol{\theta}_\star$, such that $f(\boldsymbol{\theta}_\star) \leq f(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$. In practice this condition is difficult to ensure and we often have to accept a local minimum instead.

There are two conditions that must be satisfied for a point to be a minimum of a function $f(\boldsymbol{\theta})$ with continuous second derivative [109]:

[109]: Nocedal and Wright (2006), *Numerical Optimization* Thm. 2.4

$$\begin{aligned} \nabla f(\boldsymbol{\theta}_\star) &= 0, \\ \nabla\nabla^\top f(\boldsymbol{\theta}_\star) &\succ 0. \end{aligned}$$

The Hessian matrix $\boldsymbol{H}(\boldsymbol{\theta}) = \nabla\nabla^\top f(\boldsymbol{\theta})$ has elements $[\nabla\nabla^\top f(\boldsymbol{\theta})]_{ij} = \partial^2 f(\boldsymbol{\theta})/\partial\theta_i\partial\theta_j$

The first condition indicates that $\boldsymbol{\theta}_\star$ is a change-point so the function is neither increasing nor decreasing. The second condition states that the Hessian (the matrix with all second-order partial derivatives) is symmetric and positive definite (spd). This condition ensures that $\boldsymbol{\theta}_\star$ is indeed a minimum because the gradient and by extension the function can only increase for other $\boldsymbol{\theta}$ in a small neighborhood around $\boldsymbol{\theta}_\star$. For the sake of explanation we will only assume the function to be sufficiently smooth to allow a continuous second derivative.

This is the multivariate version of saying "the second derivative is positive at $\boldsymbol{\theta}_\star$".

The second condition also be relaxed to allow positive semi-definiteness. In this setting the minimum is not unique but instead a whole subspace of $\mathbb{R}^D$ that lie in the null space of $\nabla\nabla^\top f(\boldsymbol{\theta}_\star)$ will be a valid minimum.

All of the optimization algorithms of this chapter will be of iterative nature. At iteration t the algorithm then performs the generic parameter update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \boldsymbol{d}_t, \tag{4.2}$$

which consists of a step direction $\boldsymbol{d}_t$ and a scalar step length $\eta_t \in \mathbb{R}^+$. Each update is then expected to result in $f(\boldsymbol{\theta}_{t+1}) < f(\boldsymbol{\theta}_t)$, and by iterating long enough a minimum will be reached.

In the remainder of the chapter we will explore different strategies for choosing η_t and $\boldsymbol{d}_t$ as this will be of central importance to the following chapters. We will go over two general classes of optimization and highlight their respective advantages. We will also explore the area in between the two classes, and elaborate on their connection to linear algebra.

4.1 First-order Optimization

The first group of optimization routines is known as first-order algorithms. They fall in this category because they use information up to the first derivative. These methods choose the next update direction $\boldsymbol{d}_t$ by using available gradient information. By virtue of AD frameworks that readily provide gradients, this family of methods have gained tremendous popularity in the ML community [131] .

In principle one could also use zeroth-order information *i. e.* function values and still call it a first-order algorithm. This is implicitly done by line search routines, see Sec. 4.4.

[131]: Schmidt et al. (2021), 'Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers' Appendix A

A simple example of a first-order algorithm is gradient descent with the standard update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \nabla f(\boldsymbol{\theta}_t), \tag{4.3}$$

where the step direction follows the negative gradient $\boldsymbol{d}_t = \nabla f(\boldsymbol{\theta}_t)$. The convergence rate of gradient descent along with most first-order methods are limited by certain properties of the Hessian.

For a multivariate quadratic function

$$f(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_\star)^\top \boldsymbol{Q}(\boldsymbol{\theta} - \boldsymbol{\theta}_\star) \quad \text{with } \boldsymbol{Q} = \boldsymbol{Q}^\top \text{ and } \boldsymbol{Q} \succ 0,$$

The constraints on $\boldsymbol{Q}$ indicate that the matrix is symmetric and positive definite (spd).

we get the constant Hessian $\nabla\nabla^\top f(\boldsymbol{\theta}) = \boldsymbol{Q}$. In this setting it is possible to upper bound the convergence rate by studying the convergence of an algorithm that uses the optimal step length $\eta_t = \arg\min f(\boldsymbol{\theta}_t - \eta \cdot \nabla f(\boldsymbol{\theta}_t))$. It is the possible to show that the distance to the optimum then decreases in the following fashion [93, 109]

[93]: Luenberger (1973), *Introduction to linear and nonlinear programming* §7.8

[109]: Nocedal and Wright (2006), *Numerical Optimization* Thm. 3.3

$$\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_\star\|_Q \leq \left(\frac{\kappa_Q - 1}{\kappa_Q + 1}\right) \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_\star\|_Q. \tag{4.4}$$

κ_Q is the condition number defined as the ratio of largest and smallest eigenvalue of $\boldsymbol{Q}$

$$\kappa_Q = \frac{\lambda_{\max}(\boldsymbol{Q})}{\lambda_{\min}(\boldsymbol{Q})}. \tag{4.5}$$

The subscripted norm $\|\cdot\|_Q$ in Eq. (4.4) refers to a general metric induced by the spd matrix $\boldsymbol{Q}$ that can be used to measure distances.

The rate in Eq. (4.4) can also be expressed in the more common euclidean norm $\|x - y\|_2$, which corresponds to $\boldsymbol{Q} = I$ in Eq. (4.6), by performing a change of variables.

$$\|\boldsymbol{x} - \boldsymbol{y}\|_Q^2 = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{Q}(\boldsymbol{x} - \boldsymbol{y}) \tag{4.6}$$

This norm is also known as the Mahalanobis distance and occurs frequently within probability theory and statistics [97]. Both the general norm and condition number will be recurring features of this manuscript.

[97]: Mahalanobis (1936), 'On the generalized distance in statistics'

The condition number in Eq. (4.5) measures the most extreme curvature ratio. A high κ_Q means that the direction with highest curvature has significantly higher curvature than the direction with the lowest curvature.

This conceptualizes as very long and narrow ravines in the optimization landscape. For a $\kappa_Q \gg 1$ the convergence constant in Eq. (4.4) $(\kappa_Q - 1)/(\kappa_Q + 1) \approx 1$ and the convergence is very slow for GD. Such a situation can be seen in Fig. 4.1 where GD keeps jumping across the ravine and makes little progress towards the minimum.

For more general optimization problems it is necessary to make additional assumptions about the function[1] , some of which can be difficult to verify in practice. We will not delve into the details but these assumptions usually correspond to properties of the Hessian to arrive at *local* convergence rates.

1: Lipschitz continuous derivatives and strong convexity are common assumptions.

Momentum Another kind of first-order parameter update worth mentioning is Polyak's momentum [117]. In addition to the gradient it also uses an accumulating momentum term $\boldsymbol{m}$ that determines the step direction

[117]: Polyak (1964), 'Some methods of speeding up the convergence of iteration methods'

$$\begin{aligned} \boldsymbol{m}_t &= \beta \boldsymbol{m}_{t-1} + \nabla f(\boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \cdot \boldsymbol{m}_t, \end{aligned} \tag{4.7}$$

By reformulating the momentum update to $\boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + (1-\beta)\nabla f(\boldsymbol{\theta}_t)$ it is possible to interpret the momentum as a biased running average [98, 6].

with a $\beta \in [0, 1]$. Under certain convexity conditions it can speed up the optimization and achieve a convergence rate that scales with the square root of the condition number

[98]: Mahsereci (2018), 'Probabilistic Approaches to Stochastic Optimization' §8
[6]: Balles and Hennig (2018), 'Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients'

$$\frac{\sqrt{\kappa_Q} - 1}{\sqrt{\kappa_Q} + 1}.$$

This requires an appropriately chosen momentum term β, and step length η but can lead to significantly better convergence than standard gradient descent [141, 54]. An example of this discrepancy in performance between the two algorithms can be seen in Fig. 4.1.

[141]: Sutskever et al. (2013), 'On the importance of initialization and momentum in deep learning'
[54]: Goh (2017), 'Why Momentum Really Works'

These two forms of first-order optimization algorithms constitute the backbone of the stochastic optimization routines used in training of DL models, but also features heavily in other areas of ML. Most of the first-order algorithms that come bundled with popular AD/ML frameworks employ a manipulated version of GD or momentum in their updates.

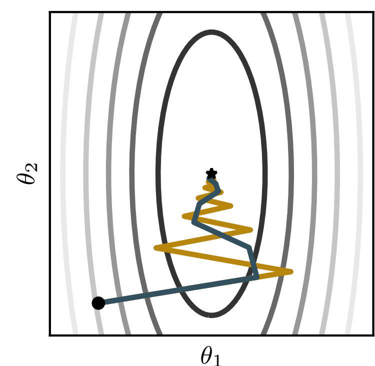


Figure 4.1: Gradient descent ( ) and Momentum ( ) on a quadratic optimization problem with highlighted contour lines. When the eigenvalues of the Hessian significantly differ ($\lambda_{\max}(H) \gg \lambda_{\min}(H)$), then gradient-based optimization algorithms tend to zigzag across the valleys leading to slow convergence. Momentum speeds up the convergence by averaging out the oscillations.

4.2 Second-order Optimization

The strong dependence on the condition number in first-order optimization algorithms means that the algorithm makes slow progress on ill-conditioned[2] problems. In traditional optimization it is common to use more efficient optimization algorithms that reduce or eliminate this dependency on condition number. The most prominent family of such algorithms are known as second-order algorithms. To see the benefit of these algorithms we start by approximating the function around the current iterate with a second-order Taylor expansion.

$$f(\boldsymbol{\theta}_t + \boldsymbol{d}) \approx f(\boldsymbol{\theta}_t) + \boldsymbol{d}^\top \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}\boldsymbol{d}^\top \underbrace{\left(\nabla\nabla^\top f(\boldsymbol{\theta}_t)\right)}_{\boldsymbol{H}(\boldsymbol{\theta}_t)} \boldsymbol{d} + O(|\boldsymbol{d}|^3) \tag{T}$$

2: Problems with $\kappa \gg 1$.

The Hessian matrix $\nabla\nabla^\top f(\boldsymbol{\theta}_t)$ features a prominent role in unconstrained optimization and will generally be denoted $\boldsymbol{H}(\boldsymbol{\theta}_t)$. From the truncated Taylor series we build a surrogate function which we aim to minimize in the current iteration.

$$\bar{f}(\boldsymbol{d}) = f(\boldsymbol{\theta}_t) + \boldsymbol{d}^\top \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}\boldsymbol{d}^\top \boldsymbol{H}(\boldsymbol{\theta}_t)\boldsymbol{d} \tag{4.8}$$

The minimum of this surrogate function is found by setting the derivative to zero assuming $\boldsymbol{H}(\boldsymbol{\theta})$ is symmetric and positive definite . The minimum of the surrogate occurs at the $\boldsymbol{d}_t$ that satisfies

The derivative w.r.t. the local variable $\boldsymbol{d}$ $\nabla_{\boldsymbol{d}} \bar{f} = 0$

The matrix $\boldsymbol{H}$ is spd if $\boldsymbol{H} = \boldsymbol{H}^\top$ and $\boldsymbol{v}^\top \boldsymbol{H} \boldsymbol{v} > 0 \; \forall \boldsymbol{v} \in \mathbb{R}^D$. This is equivalent to saying all eigenvalues of $\boldsymbol{H}$ $\lambda_i > 0$ $i = 1, \ldots, D$.

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{\theta}_t)\boldsymbol{d}_t &= -\nabla f(\boldsymbol{\theta}_t) \\ \Leftrightarrow \boldsymbol{d}_t &= -\boldsymbol{H}(\boldsymbol{\theta}_t)^{-1}\nabla f(\boldsymbol{\theta}_t). \end{aligned} \tag{N}$$

Solving this linear system and using the resulting $\boldsymbol{d}_t$ in Eq. (4.2) along with $\eta = 1$ leads to the famous Newton step [16] . If the function is close to a minimum so the surrogate Eq. (4.8) is a good approximation then the Newton update converges with a quadratic rate [109] .

[16]: Boyd and Vandenberghe (2004), *Convex optimization* §9.5

[109]: Nocedal and Wright (2006), *Numerical Optimization* Thm. 3.5

$$\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_\star\| \leq c \cdot \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_\star\|^2$$

This means that the effective scaling factor c in front decreases with each iteration. Another way to phrase it that the convergence rate is accelerating. Often just a few iterations of the Newton update are necessary to make significant progress in traditional optimization.

There are several advantages to the Newton update compared to first-order algorithms, and one in particular is worth highlighting. Simply performing a parameter update with the gradient and an arbitrary step length is not consistent in terms of units

$$[\boldsymbol{\theta}] = [\boldsymbol{\theta}] - \eta \cdot \frac{[f]}{[\boldsymbol{\theta}]}$$

This also makes the update sensitive to the scale of the function. If we multiplicatively rescale the function with $\alpha > 0$

$$\bar{f}(\boldsymbol{\theta}_t) = \alpha \cdot f(\boldsymbol{\theta}_t) \rightarrow \nabla \bar{f}(\boldsymbol{\theta}_t) = \alpha \cdot \nabla f(\boldsymbol{\theta}_t),$$

we see that the gradient changes by the corresponding amount. In order to still scale properly η must counteract the change imposed by α.

For an $\alpha > 1$ $\bar{f}(\boldsymbol{\theta}) > f(\boldsymbol{\theta})$ the step length η must decrease and vice versa for $0 < \alpha < 1$.

The corresponding analysis for the Newton step reveals an invariance to the scaling as well as consistency with the units.

$$\begin{aligned} [\boldsymbol{\theta}] &= [\boldsymbol{\theta}] - \eta \cdot \left(\frac{\alpha \cdot [f]}{[\boldsymbol{\theta}]^2}\right)^{-1} \left(\frac{\alpha \cdot [f]}{[\boldsymbol{\theta}]}\right) \\ &= [\boldsymbol{\theta}] - \eta \cdot \frac{\alpha}{\alpha}\left(\frac{[\boldsymbol{\theta}]^2}{[f]}\right)\left(\frac{[f]}{[\boldsymbol{\theta}]}\right) \\ &= [\boldsymbol{\theta}] - \eta \cdot [\boldsymbol{\theta}] \end{aligned}$$

A downside of second-order methods such as the Newton update is its poor scaling with dimensionality. Computing the Hessian, which is required for the update, is theoretically possible for a general function thanks to automatic differentiation. However, for a high-dimensional

input space it might not be possible to store the matrix in memory due to the $\mathcal{O}(D^2)$ storage requirement. Another aspect that is often cited as the main bottleneck is the $\mathcal{O}(D^3)$ computational scaling due to the matrix inversion in Eq. (N). In some cases the matrix $\boldsymbol{H}(\boldsymbol{\theta})$ exhibits certain structure that makes it possible to invert the matrix at reduced cost, but generally it will require the solution of a linear system which has a computational cost that scales cubically in the dimension of the problem. By comparing these restrictions to the $\mathcal{O}(D)$ storage and computation required for first-order methods one can understand the prevalence and necessity of the latter in high-dimensional optimization problems.

4.3 Linear Algebra

There are more aspects to the field of optimization that will be highlighted in subsequent sections. Before that we will look into one particular connection between linear algebra and optimization that was touched upon in the previous sections.

Finding the step direction in Eq. (N) is connected to a common task in linear algebra. If the matrix $\boldsymbol{H}(\boldsymbol{\theta})$ is spd then the local surrogate function Eq. (4.8), has a unique local optimum at the Newton step in Eq. (N). In this setting there is an important connection between optimization and iterative linear solvers encountered in linear algebra. Solving the linear system $\boldsymbol{H}\boldsymbol{d} = \boldsymbol{b}$ is equivalent to optimizing the quadratic function

$$\phi(\boldsymbol{d}) = \frac{1}{2}(\boldsymbol{d} - \boldsymbol{d}_*)^\top \boldsymbol{H}(\boldsymbol{d} - \boldsymbol{d}_*) + c, \tag{4.9}$$

with an arbitrary offset c that is generally neglected. The minimum occurs at $\boldsymbol{H}(\boldsymbol{d} - \boldsymbol{d}_*) = 0 \Leftrightarrow \boldsymbol{d} = \boldsymbol{d}_* = \boldsymbol{H}^{-1}\boldsymbol{b}$. Replacing the r.h.s. $\boldsymbol{b}$ with the negative gradient recovers the Newton step in Eq. (N).

[67]: Hestenes and Stiefel (1952), 'Methods of conjugate gradients for solving linear systems'
[137]: Shewchuk (1994), *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*

One of the most famous solvers for this kind of problem is the method of conjugate gradients (CG) [67]. It will solve a linear system with a spd matrix $\boldsymbol{H}$ by iteratively improving the current estimate of the solution. CG has several appealing properties making it one of the most popular iterative algorithms for solving linear systems with an spd matrix [137]. A few of these properties are listed below.

In linear algebra these linear systems are often denoted $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. To avoid cluttered notation it is only mentioned here for reference.

- The algorithm only requires access to matrix-vector multiplications (MVM) with the matrix *i.e.*, $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{v}$. Sometimes it is not possible to construct or store the matrix that is present in the linear system but it might be possible to multiply with it. This is particularly beneficial when $\boldsymbol{H}$ exhibits certain characteristics such as sparsity patterns or low-rank structure. CG has become particularly interesting in combination with AD libraries due to a trick that allows MVMs with the Hessian by performing 2 AD operations [114].
- Low memory requirements. It is possible to phrase the updates of CG to use only 3 vectors of size D making it similar in storage requirement to first-order methods.
- Converges to the solution after D iterations in *exact* arithmetic. The convergence can break down due to numerical imprecision and practitioners have to rely on efficient preconditioning or restarts to improve stability and convergence.

$\boldsymbol{H}\boldsymbol{v}$ can be calculated with 2 reverse-mode operations as $\boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{v} = \nabla(\boldsymbol{v}^\top \nabla f(\boldsymbol{\theta}))$. A combination of forward-mode and reverse-mode is also possible and more efficient but not generally available.

[114]: Pearlmutter (1994), 'Fast exact multiplication by the Hessian'

- The convergence rate of CG is usually referenced to depend on the square root of the condition number

$$\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right),$$

similar to GD with Momentum[3] , but without tuning of parameters. The referenced convergence rate is an upper bound and a tighter superlinear rate is often observed. Whether this occurs or not depends on the distribution of the eigenvalues [145]. A beneficial setup is when there are a few significantly larger eigenvalues or if there are clustered eigenvalues [109].
- CG can often converge faster to a good approximate solution in $k \ll D$ iterations, potentially leading to a good cost-performance trade-off.

3: CG chooses a step direction $\boldsymbol{s}_t = -\nabla f(\boldsymbol{\theta}_t) + \beta_t \cdot \boldsymbol{s}_{t-1}$ which can be seen as an adaptive form of Momentum that uses knowledge of the underlying problem to adjust the parameters (η_t, β_t) for local optimality.

The convergence is governed by an effective condition number that changes as the Ritz values converge to the extreme eigenvalues.

[145]: Van der Sluis and Vorst (1986), 'The rate of convergence of conjugate gradients'
[109]: Nocedal and Wright (2006), *Numerical Optimization* §5.1

CG can also be seen as an optimization algorithm within another optimization algorithm when used for solving linear systems that include the Hessian, such as the Newton update. Approximating the Newton step with CG is known as Newton-CG [109], truncated Newton [107] or Hessian-free optimization [101]. A lot of work has focused on generalizing the CG algorithm to nonlinear functions [45, 115, 28]. The main difference between these algorithms is in which assumptions are used implicitly to determine a new step direction[3] when properties of orthogonality and conjugacy are not fulfilled.

see also `scipy.optimize.minimize`
[107]: Nash (2000), 'A survey of truncated-Newton methods'
[101]: Martens (2010), 'Deep learning via Hessian-free optimization'

[45]: Fletcher and Reeves (1964), 'Function minimization by conjugate gradients'
[115]: Polak, E. and Ribiere, G. (1969), 'Note sur la convergence de méthodes de directions conjuguées'
[28]: Dai and Yuan (1999), 'A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property'

One downside of CG worth mentioning is its sensitivity to numerical imprecision. This restricts the applicability of CG within ML where subsampling is often necessary which introduces significant noise beyond the numeric precision.

4.4 Step Length

Talking about first-order optimization algorithms will invariably lead to a discussion about the step length parameter η in Eq. (4.2). Compared to second-order methods there is usually not a suitable standard value of the step length varies a lot between problems and will have a strong impact on the optimization. If the step length is chosen too large then the optimization algorithm might fail to converge or even diverge, leading to numeric overflows. If, on the other hand, the step length is chosen too small it will lead to minor and slow progress. Finding a suitable value for the step length usually requires a significant amount of trial and error. The process can be accelerated by running the same optimization problem with different learning rates in parallel. This is obviously only possible if there is hardware available. In ML, the iterative process of tuning the learning rate, either sequentially or in parallel, is a time-consuming yet necessary step of model development.

Also known as step size or learning rate. These terms will be used interchangeably throughout the manuscript but refer to the same quantity.

In the previous sections we saw that the convergence rate of first-order algorithms depends on the condition number κ. The optimal step length has a similar dependency but is mostly influenced by the largest eigenvalue of the Hessian, which encodes the highest curvature locally, as opposed to the ratio. In this way it is possible to interpret the step length as a crude approximation to the curvature.

For general optimization this corresponds to a ratio of problem-dependent Lipschitz constant L and strong convexity parameter μ. These parameters bound how much and little a function can deviate with a perturbation in the input.

In traditional deterministic optimization one can circumvent the problem of choosing a step size by employing a line search routine instead [109] . Such algorithms look for a local minimum along a predetermined search direction by repeatedly evaluating the function, and potentially gradient along the line. A step length is then chosen as

[109]: Nocedal and Wright (2006), *Numerical Optimization* §3

$$\eta_t = \arg\min_{\eta \in \mathbb{R}^+} f(\boldsymbol{\theta}_t - \eta \cdot \boldsymbol{d}_t), \quad \text{with } \boldsymbol{d}_t^\top \nabla f(\boldsymbol{\theta}_t) > 0. \tag{4.10}$$

Finding the exact minimum along $\boldsymbol{d}_t$ can be expensive and often it is more useful to trade a few exact parameter updates against more parameter updates with approximately optimal step lengths. To this end there are line search conditions that ensure a "sufficient" decrease in each iteration is achieved [160, 161, 4]. These conditions have been generalized in various ways to the stochastic setting encountered in ML [99, 150, 157]. Despite promising progress, line search routines are not so common in ML due to the multiplicatively increasing cost and time associated with repeated queries of function and gradient. Instead the same problem is often solved with a first-order algorithm in parallel on multiple machines with different fixed learning rates.

This mentality is mirrored in ERM where more iterations with smaller batches are favored over large batches with fewer iterations.

[160]: Wolfe (1969), 'Convergence conditions for ascent methods'
[161]: Wolfe (1971), 'Convergence conditions for ascent methods. II: Some corrections'
[4]: Armijo (1966), 'Minimization of functions having Lipschitz continuous first partial derivatives'
[99]: Mahsereci and Hennig (2017), 'Probabilistic line searches for stochastic optimization'
[150]: Vaswani et al. (2020), 'Adaptive Gradient Methods Converge Faster with Over-Parameterization (and you can do a line-search)'
[157]: Wills and Schön (2019), 'Stochastic quasi-Newton with line-search regularization'

4.5 Step Direction

In Sec. 4.1 and 4.2 we studied the benefits and limitations of first-order and second-order optimization algorithms which lie diametrically opposite each other in terms of computation and convergence. It is then natural to ask if there are ways to mitigate the limitations of first-order algorithms yet avoid the extreme cost of second-order algorithms. Here we will explore some of the options that lie in-between the two frameworks and see how it is possible to interpolate between the two. The first setting is preconditioning which is mainly used in linear algebra but also applies to optimization. The second is optimization with a general metric and we will see how the two can be made equivalent under certain assumptions.

Preconditioning

In Sec. 4.3 we saw how CG can be seen as a first-order optimization algorithm that matches the optimal convergence rate of GD with momentum. It is possible to speed up the convergence rate further by employing a technique called preconditioning. The idea behind preconditioning is to alter the spectral density of $\boldsymbol{H}$ in Eq. (4.9), and by extension the condition number κ, to make more progress in each iteration [109] . Determining a suitable preconditioner is a problem dependent art and employing it can be necessary to ensure convergence of CG due to numerical imprecision. Preconditioning is usually presented in terms of the specific problem where CG is applicable (Eq. (4.9)). To facilitate the comparison with a general metric in the next section we will instead look at preconditioning from a general optimization perspective.

[109]: Nocedal and Wright (2006), *Numerical Optimization* §5.1

The main idea behind preconditioning is to linearly transform the input space into a new beneficial basis, perform the optimization in the new

space and transform back. This requires an invertible map $\boldsymbol{P} : \mathbb{R}^D \longrightarrow \mathbb{R}^D$ in order to construct the transformed variable $\tilde{\boldsymbol{\theta}} = \boldsymbol{P}\boldsymbol{\theta}$. It is now possible to express the function in terms of the transformed variable $f(\boldsymbol{\theta}) = f(\boldsymbol{P}^{-1}\tilde{\boldsymbol{\theta}})$.

$\boldsymbol{P}$ is usually characterized with an invertible $D \times D$ matrix.

Since $\boldsymbol{P}$ is invertible it follows that $\tilde{\boldsymbol{\theta}} = \boldsymbol{P}\boldsymbol{\theta} \Leftrightarrow \boldsymbol{P}^{-1}\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}$

Using the chain rule of derivatives we can now express the gradient in the transformed space by expanding the derivative in its partial constituents

$$\frac{\partial f(\boldsymbol{P}^{-1}\tilde{\boldsymbol{\theta}})}{\partial \tilde{\theta}^{(i)}} = \underbrace{\frac{\partial f(\boldsymbol{\theta})}{\partial \theta^{(j)}}}_{\nabla f(\boldsymbol{\theta})} \underbrace{\frac{\partial \theta^{(j)}}{\partial \tilde{\theta}^{(i)}}}_{\boldsymbol{P}^{-1}} = \boldsymbol{P}^{-\top}\nabla f(\boldsymbol{\theta}). \tag{4.11}$$

The transpose appears because of the contraction of the j-index and we want the gradient to match the orientation of $\tilde{\boldsymbol{\theta}}$.

A single gradient descent step in the transformed space is now realized with the update

$$\tilde{\boldsymbol{\theta}}_{t+1} = \tilde{\boldsymbol{\theta}}_t - \eta_t \cdot \boldsymbol{P}^{-\top}\nabla f(\boldsymbol{\theta}_t). \tag{4.12}$$

We can then transform back to the original parameter by multiplying Eq. (4.12) from the left with $\boldsymbol{P}^{-1}$ to arrive at the final update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \underbrace{\boldsymbol{P}^{-1}\boldsymbol{P}^{-\top}}_{\boldsymbol{W}^{-1}} \nabla f(\boldsymbol{\theta}_t). \tag{4.13}$$

We now see that a preconditioned gradient step does not require the matrix $\boldsymbol{P}$ explicitly but rather the combined symmetric matrix $\boldsymbol{W}^{-1} = \boldsymbol{P}^{-1}\boldsymbol{P}^{-\top}$. The resulting matrix $\boldsymbol{W}^{-1}$ is spd if $\boldsymbol{P}$ is of full rank leading to a well-defined parameter update.

$\boldsymbol{P}$ can also be rank-deficient but it leads to optimization steps restricted to a limited subspace and the pseudo inverse must be used as a least-square solution.

To better understand the effect of preconditioning it is useful to look at the local Taylor series in the transformed space. Expressing the second-order Taylor expansion, see Eq. (T), in terms of the transformed variable leads to

$$f(\boldsymbol{P}^{-1}\tilde{\boldsymbol{\theta}} + \tilde{\boldsymbol{d}}) \approx f(\boldsymbol{\theta}) + \tilde{\boldsymbol{d}}^\top\boldsymbol{P}^{-\top}\nabla f(\boldsymbol{\theta}) + \frac{1}{2}\tilde{\boldsymbol{d}}^\top \underbrace{\boldsymbol{P}^{-\top}[\boldsymbol{H}(\boldsymbol{\theta})]\boldsymbol{P}^{-1}}_{\tilde{H}(\boldsymbol{\theta})} \tilde{\boldsymbol{d}}. \tag{4.14}$$

In the transformed space we have a local curvature induced by $\tilde{H}(\boldsymbol{\theta})$. If $\tilde{H}(\boldsymbol{\theta}) = I$ then the condition number $\kappa = 1$ and we can expect fast convergence. In this setting $\boldsymbol{W} = \boldsymbol{H}(\boldsymbol{\theta})$ and a step of Eq. (4.13) is equivalent to the Newton step.

[16]: Boyd and Vandenberghe (2004), *Convex optimization*

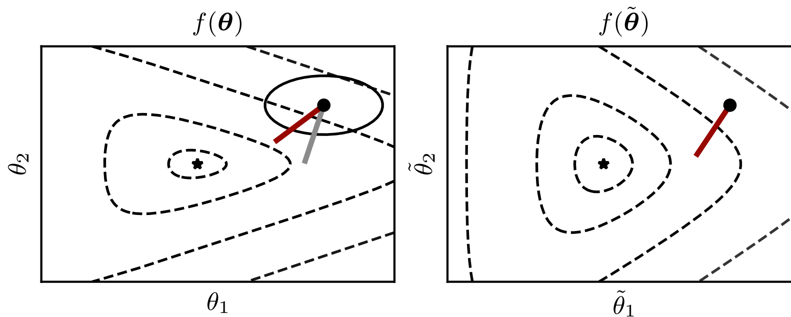


Figure 4.2: The left plot shows level curves for a convex function $f(\cdot)$ with gradient (▬) and rescaled gradient (▬) according to the elliptic metric. The right plot shows the level curves and gradient for the equivalent preconditioned input. Example function from [16].

General Metric

Another way of improving the convergence rate of first-order algorithms is to consider the optimization in a more general metric that better captures the underlying curvature [16] . One way of interpreting this is to add a quadratic regularization term to the parameter update that penalizes dimensions unevenly. A visualization of such a regularization can be seen in Fig. 4.3.

[16]: Boyd and Vandenberghe (2004), *Convex optimization* §9.4

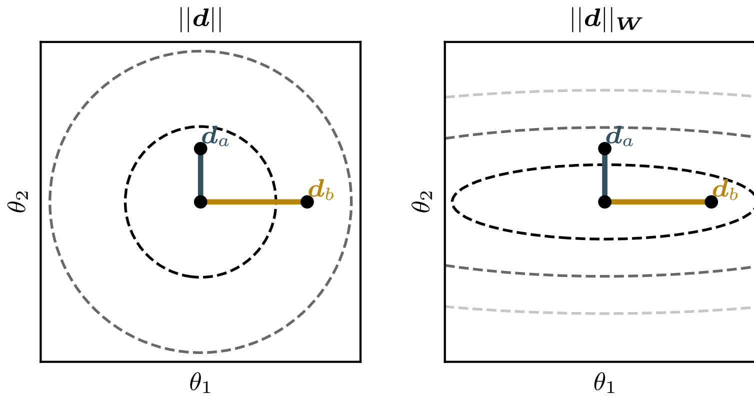


Figure 4.3: Corresponding level curves for the distance in standard L_2 norm (left) and with an arbitrary metric (right). In the left figure $||\boldsymbol{d}_a||$ (▬) < $||\boldsymbol{d}_b||$ (▬). The converse is true for the general metric (right). Note that the right contours does not have to be axis-aligned but is an attractive choice since $\boldsymbol{W}$ is then diagonal and cheap to invert.

There are two equivalent ways of writing the regularized update that are used interchangeably in the literature. The first is to phrase the local parameter update in terms of the new iterate

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) + (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2} \underbrace{(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \boldsymbol{W} (\boldsymbol{\theta} - \boldsymbol{\theta}_t)}_{\|\boldsymbol{\theta}-\boldsymbol{\theta}_t\|^2_W},$$

$\|\boldsymbol{a}-\boldsymbol{b}\|^2_W$ is the squared distance between $\boldsymbol{a}$ and $\boldsymbol{b}$ measured in $\boldsymbol{W}$-norm for $\boldsymbol{W} \succ 0$. In statistics this is known as the Mahalanobis distance [97].
[97]: Mahalanobis (1936), 'On the generalized distance in statistics'

which is suitable for illustrative purposes. The second option is to express the regularization in terms of the update $\boldsymbol{d} = (\boldsymbol{\theta} - \boldsymbol{\theta}_t)$

$$\boldsymbol{d}_t = \arg\min_{\boldsymbol{d}} f(\boldsymbol{\theta}_t) + \boldsymbol{d}^\top \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2} \underbrace{\left(\boldsymbol{d}^\top W \boldsymbol{d}\right)}_{\|\boldsymbol{d}\|^2_W} \tag{4.15}$$

which is more suitable for the connection to other optimization routines and will be the preferred option moving forward. If $\boldsymbol{W}$ is an spd matrix then there exists a unique minimum to Eq. (4.15) which occurs at

A valid metric requires that $\text{dist}(\boldsymbol{a},\boldsymbol{b}) = \text{dist}(\boldsymbol{b},\boldsymbol{a})$ (symmetry) and $\text{dist}(\boldsymbol{a},\boldsymbol{b}) \geq 0, \forall \boldsymbol{a},\boldsymbol{b}$ (positive definiteness). A generalization of this definition of distance with weaker constraints is known as a Bregman divergence.

$$\boldsymbol{W}\boldsymbol{d}_t = -\nabla f(\boldsymbol{\theta}_t) \Leftrightarrow \boldsymbol{d}_t = -\boldsymbol{W}^{-1} \nabla f(\boldsymbol{\theta}_t).$$

By combining the full parameter update of Eq. (4.2) with a step length we get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \boldsymbol{W}^{-1} \nabla f(\boldsymbol{\theta}_t). \tag{4.16}$$

There are several aspects of the update that are worth elaborating as they connect to previous sections of the chapter.

A connection to Gaussian inference will be explored in Ch. 5.

- For $\boldsymbol{W} = \boldsymbol{I}$ Eq. (4.16) is equivalent to the standard gradient descent step. This corresponds to circular contour lines so every dimension is treated equivalently.
- For $\boldsymbol{W} = \boldsymbol{H}(\boldsymbol{\theta}_t)$ with $\boldsymbol{H}(\boldsymbol{\theta}_t) \succ 0$ Eq. (4.16) recovers the Newton step. This way it is possible to interpret the Newton step as a

See Fig. 4.3.

parameter update with a regularization provided by the local curvature captured by the Hessian.
- For a general $\boldsymbol{W} > 0$ the update in Eq. (4.16) is equivalent to the preconditioned update in Eq. (4.13).

Knowledge about the optimization problem can be encoded in $\boldsymbol{W}$ to bring the convergence rate of first-order methods closer to that of second-order methods while still keeping the per-iteration cost low. The majority of stochastic optimization routines that are used in ML revolve around finding cheap metrics $\boldsymbol{W}$ that are updated during the course of the optimization.

$\boldsymbol{W}$ is usually a diagonal matrix to allow cheap inversion.

4.6 Quasi-Newton Methods

The introduction of a metric/preconditioner allows a convenient way of trading computational cost per iteration vs precision if the parameter update. This is useful when there is information about the optimization problem that can be encoded in $\boldsymbol{W}$ to approximate the curvature. It is in general difficult to come up with a cheap and efficient $\boldsymbol{W}$ for a specific problem and even more so for arbitrary problems. A potential workaround for this is to let $\boldsymbol{W}$ adapt to the problem during the optimization. Phrased differently, it is possible to have a model that *learns* the curvature during the optimization. Arguably, the most famous adaptive curvature estimators are known as Quasi-Newton (QN) methods. The name stems from the way that the algorithms operate, namely, by performing an approximate Newton step on a limited subspace which reduces the cost per-step compared to the full Newton update. These algorithms only require access to gradients which are collected during the optimization to estimate the curvature. Since the Hessian encodes the local change in gradients one can then try to fit a matrix with a constant curvature to approximate the Hessian[4] .

4: or the inverse of the Hessian

The whole procedure relies on the chain rule of derivatives together with the fundamental theorem of calculus as a way to link a difference in gradients to properties of the Hessian. We start by parameterizing the path between two consecutive parameter iterates of the optimization

$$\boldsymbol{r}_t(\tau) = \boldsymbol{\theta}_t + \tau(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t).$$

By now integrating the change in gradient along the we arrive at two possible parameterizations.

$$\int_0^1 \frac{\partial}{\partial \tau} \nabla f(\boldsymbol{r}_t(\tau)) \mathrm{d}\tau = \begin{cases} \nabla f(\boldsymbol{r}_t(1)) - \nabla f(\boldsymbol{r}_t(0)) = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t) \\ \int_0^1 \underbrace{\nabla \nabla^\top f(\boldsymbol{r}_t(\tau))}_{\boldsymbol{H}(\boldsymbol{r}_t(\tau))} (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \mathrm{d}\tau \end{cases}$$

If we now equate these two expressions we arrive at the full form of the famous secant equation [109]:

It is also known as the secant condition. [109]: Nocedal and Wright (2006), *Numerical Optimization* Thm. 2.1

$$\int_0^1 \boldsymbol{H}(\boldsymbol{r}_t(\tau))(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \mathrm{d}\tau = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t), \tag{4.17}$$

which connects the Hessian to gradients and is the foundation of all QN-methods. The parametric form of the Hessian in Eq. (4.17) is cumbersome to deal with so it is usually approximated as constant along the path. This approximation can be justified in two ways:

1. The averaged Hessian along the path $\boldsymbol{r}_t(\tau)$ is constant and fulfills the condition

$$\int_0^1 \underbrace{\bar{\boldsymbol{H}}_t(\tau)}_{\boldsymbol{H}_t}(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)\mathrm{d}\tau = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t).$$

2. For a twice continuously differentiable function there exists a point τ_* along $\boldsymbol{r}_t(\tau)$ such that

$$\int_0^1 \underbrace{\boldsymbol{H}(\boldsymbol{r}_t(\tau_*))}_{\boldsymbol{H}_t}(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)\mathrm{d}\tau = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t).$$

   A 1d example of this statement involving the function and derivative can be seen in Fig. 4.4.

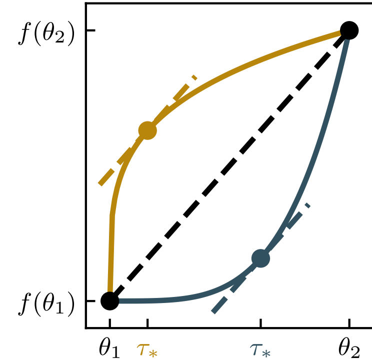


Figure 4.4: Two functions with a $\tau_* \in [\theta_1, \theta_2]$ such that $f'(\tau_*) = \Delta f / \Delta\theta$.

The Hessian is multiplied with the vector $(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)$ not evaluated at that point.

Regardless of justification, by approximating the Hessian as constant over $\boldsymbol{r}(\tau)$ Eq. (4.17) simplifies to

$$\boldsymbol{H}_t(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t), \tag{4.18}$$

which avoids the integration along a curve. This approximation is so common that many researchers refer to the approximation in Eq. (4.18) when talking about the secant equation. To avoid confusion when comparing to similar work we will also refer Eq. (4.18) as the secant equation for the remainder of the manuscript unless otherwise stated.

A Family of Updates

It turns out that there are many ways to fit a matrix such that a linear constraint is fulfilled. This leads to a whole family of matrix updates that each satisfy the secant equation, Eq. (4.18), in various ways. Some of these updates will reappear in later chapters and therefore merit further discussion. The following list of updates is by no means exhaustive but will only highlight a few of the more prevalent ones with different appealing properties. See [37, 59, 55, 109] and references therein for more updates, history, comparisons and theory.

[37]: Dennis and Moré (1977), 'Quasi-Newton methods, motivation and theory'
[59]: Haelterman (2009), 'Analytical study of the least squares quasi-Newton method for interaction problems'
[55]: Goldfarb (1970), 'A family of variable-metric methods derived by variational means'
[109]: Nocedal and Wright (2006), *Numerical Optimization*

To simplify the notation we start by defining two vectors that correspond to a difference in inputs as well as gradients

$$\begin{aligned} \boldsymbol{s}_t &= \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \quad \text{and} \\ \boldsymbol{y}_t &= \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t). \end{aligned}$$

With these vectors defined it is possible to write the secant equation in the shorter form

$$\boldsymbol{H}\boldsymbol{s}_t = \boldsymbol{y}_t$$

which makes the notation less cluttered moving forward. The goal of QN-methods is to use the secant equation in order to build an estimate of the Hessian that can be used to determine new step directions by imitating the Newton step

$$\boldsymbol{d}_t = -\boldsymbol{H}_t^{-1}\nabla f(\boldsymbol{\theta}_t). \tag{4.19}$$

We will here focus on the matrix updates but a successful implementation would require an additional line search routine to ensure convergence and stability of the resulting matrices.

Broyden's rank-1 The first and simplest update is the rank-1 update of Broyden [19]

$$\boldsymbol{H}_{t+1} = \boldsymbol{H}_t + \frac{(\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}_t)\boldsymbol{s}_t^\top \boldsymbol{W}}{\boldsymbol{s}_t^\top \boldsymbol{W}\boldsymbol{s}_t} \tag{Broyden}$$

[19]: Broyden (1965), 'A class of methods for solving nonlinear simultaneous equations'

with an arbitrary symmetric matrix $\boldsymbol{W}$ of full rank. This update satisfies $\boldsymbol{H}_{t+1}\boldsymbol{s}_t = \boldsymbol{y}_t$ which is easily verified by performing the multiplication. It is however not symmetric, which is a property of the Hessian for a twice continuously differentiable scalar function. Broyden's update can, despite the asymmetry, lead to good performance when used to determine a new step direction with Eq. (4.19).

$\boldsymbol{W}$ can also be rank-deficient but the important part is that $\boldsymbol{s}_t$ is not in the nullspace of $\boldsymbol{W}$ so $\boldsymbol{s}_t^\top \boldsymbol{W}\boldsymbol{s}_t \neq 0$.

Symmetric rank-1 Trying to make Broyden's update symmetric inevitably leads to the SR-1 update. The update can be made symmetric by exchanging $\boldsymbol{W}\boldsymbol{s}_t \leftrightarrow (\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}_t)$ in Broyden's update to arrive at

$$\boldsymbol{H}_{t+1} = \boldsymbol{H}_t + \frac{(\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}_t)(\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}_t)^\top}{\boldsymbol{s}_t^\top(\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}_t)}. \tag{SR-1}$$

It is still a rank-1 update but it retains the symmetric property as long as the initial matrix $\boldsymbol{H}_0$ is symmetric. A slight drawback is that the matrix is not positive definite in general and requires greater care if $\boldsymbol{s}_t^\top\boldsymbol{y}_t \approx \boldsymbol{s}_t^\top\boldsymbol{H}_t\boldsymbol{s}_t$. The first detail means that there is no well-defined minimum to the quadratic approximation in Eq. (4.8) and second leads to an ill-defined inverse of the matrix. To combat these shortcomings it is common to use the SR-1 update in conjunction with trust-region methods.

Trust-region methods are an alternative to line search methods. These define a "region of trust" in which a surrogate is minimized. If the surrogate is a good fit to the real function, the region is made larger and is made smaller if the surrogate matches poorly.

Davidon-Fletcher-Powell (DFP) The next logical step in developing a good update is to make the SR-1 update positive definite, which is feasible if we instead consider a rank-2 update. There are several possibilities to achieve such an update but one of the most famous options was developed by Davidon [31], Fletcher and Powell [47]. The DFP-update is often written

[31]: Davidon (1959), *Variable metric method for minimization*
[47]: Fletcher and Powell (1963), 'A rapidly convergent descent method for minimization'

$$\boldsymbol{H}_{t+1} = \left(\boldsymbol{I} - \rho_t\boldsymbol{y}_t\boldsymbol{s}_t^\top\right)\boldsymbol{H}_t\left(\boldsymbol{I} - \rho_t\boldsymbol{s}_t\boldsymbol{y}_t^\top\right) + \rho_t\boldsymbol{y}_t\boldsymbol{y}_t^\top, \tag{DFP}$$

with $\rho_t = (\boldsymbol{s}_t^\top\boldsymbol{y}_t)^{-1}$, to emphasize the recurrent nature of the update and for efficient implementation in algorithms. It can also be phrased in the

following way to make the rank-2 update clear.

$$H_{t+1} = H_t + [y_t, (y_t - H_t s)] \begin{bmatrix} \rho_t(\rho_t s_t^\top H_t s_t - 1) & \rho_t \\ \rho_t & 0 \end{bmatrix} \begin{bmatrix} y_t^\top \\ (y_t - H_t s_t)^\top \end{bmatrix}$$

The DFP-update satisfies the most important properties we want fulfilled when approximating the Hessian. It is symmetric, updates are of low-rank to facilitate fast inversion and the resulting matrix is positive definite if $s_t^\top H_t s_t > s_t^\top y_t > 0$. Despite ticking all the right boxes the DFP-update is often overshadowed by the next and last update which has become the standard algorithm for unconstrained optimization in several optimization packages.

Broyden-Fletcher-Goldfarb-Shanno (BFGS) The most popular QN method improves upon the DFP update in a simple yet important way. Instead of updating the estimate of the Hessian, it performs the update for the inverse. It is easiest understood by rewriting the secant equation (Eq. (4.18)) to involve the inverse Hessian

$$H s_t = y_t \Leftrightarrow s_t = H^{-1} y_t .$$

A more formal derivation is to write the update as the minimization problem solved by the Dennis family [37].
[37]: Dennis and Moré (1977), 'Quasi-Newton methods, motivation and theory'

By now adapting the DFP-update to rely on the above condition we get the update

$$H_{t+1}^{-1} = \left(I - \rho_t s_t y_t^\top\right) H_t^{-1} \left(I - \rho_t y_t s_t^\top\right) + \rho_t s_t s_t^\top , \qquad \text{(BFGS)}$$

with once again $\rho_t = (s_t^\top y_t)^{-1}$. The update was independently discovered around the same time by Broyden [20], Fletcher [46], Goldfarb [55] and Shanno [135] giving rise to its name and it is to this day one of the most popular optimization methods. An advantage of BFGS compared to DFP is that the inverse Hessian is already estimated so the step direction in Eq. (4.19) only requires a matrix multiplication and no matrix inversion.

[20]: Broyden (1970), 'The convergence of a class of double-rank minimization algorithms 1. general considerations'
[46]: Fletcher (1970), 'A new approach to variable metric algorithms'
[55]: Goldfarb (1970), 'A family of variable-metric methods derived by variational means'
[135]: Shanno (1970), 'Conditioning of quasi-Newton methods for function minimization'

Low-rank Update

All of the preceding updates have been presented as rank-1 or rank-2 updates and it was indicated that low-rank updates are beneficial. The main reason for this is captured in Woodbury's matrix inversion lemma [162] stated below. For matrices $A \in \mathbb{R}^{D \times D}$, $U, V \in \mathbb{R}^{D \times M}$ and $C \in \mathbb{R}^{M \times M}$ with $M < D$ it is possible to write the inverse of the low-rank update

[162]: Woodbury (1950), *Inverting modified matrices*

$$(A + UCV^\top)^{-1} = A^{-1} - A^{-1} U (C^{-1} + V^\top A^{-1} U)^{-1} V^\top A^{-1}. \qquad (4.20)$$

If A is cheap to invert then it is possible to solve linear systems with the low-rank update at cost $\mathcal{O}(M^3 + MD)$ (neglecting the cost of finding A^{-1}) which can be significantly lower than the naïve $\mathcal{O}(D^3)$. This is exactly the operation that is required to find the approximate Newton step in Eq. (4.19) making low-rank updates very attractive for QN algorithms.

This trick is only applicable when $M < D$ and preferably $M \ll D$ because there is a significant computational overhead when implementing

Eq. (4.20) in standard languages compared to the highly optimized low-level numerical routines that are called when solving linear systems.

An example of this will be presented in Ch.7.

In general there is no guarantee that the $\boldsymbol{H}_t$ will converge to the true $\boldsymbol{H}(\boldsymbol{\theta})$ as $t \to D$ for the presented QN updates unless the Hessian is indeed spd and constant everywhere. If that is the case then some of the updates will converge but the optimization problem is then better addressed with CG due to its lower computational cost and storage requirement. Since we can only learn the real Hessian locally but its inverse would be expensive it is more beneficial to only store a few M $(\boldsymbol{s}, \boldsymbol{y})$-pairs instead of the whole history. If M is small and the vectors capture high-curvature directions then the low-rank curvature estimates can be used as a cheap preconditioner to speed up the convergence according to the analysis in Sec. 4.5. All of the low-rank updates are amenable to this kind of limited memory version and the name is then updated to reflect the truncated history by adding an L in front, such as L-BFGS.

The updates of CG and the presented algorithms can be made equivalent in this case, but CG avoids storing the matrix.

The important part is to reduce the condition number and therefore also low-curvature regions can just as well be used.

4.7 Summary

We have now explored some central aspects of unconstrained optimization which one should be aware of when constructing new optimization algorithms. We have seen how the, on the surface, simple choice of determining a step direction and step length becomes a multifaceted problem with several possible ways to approach. Inclusion of curvature information helps alleviate the difficulty of finding these parameters but it comes with additional computational requirements that make it infeasible for high-dimensional optimization. A lot of work has been invested in finding a good trade-off between accuracy and cost of applying the curvature estimate. This is particularly obvious in deep learning where the available optimization algorithms grow by the day [131] .

This prohibitive scaling with dimension will be a recurring theme throughout this document.

While aspects of the preceding sections will appear in every following chapter, some chapters have a tighter connection to the background.

- Chapter 5 connects the notion of general metric (Sec. 4.5) with Gaussian inference to adjust the step length (Sec. 4.4) of general first-order optimization algorithms (Sec. 4.1).
- Chapter 6 uses matrix-variate Gaussian inference to learn a curvature estimate that can be used as a preconditioner (Sec. 4.5) for first-order algorithms (Sec. 4.1) in a manner similar to quasi-Newton methods (Sec. 4.6). The resulting algorithm is initially derived using the connection between optimization and linear algebra (Sec. 4.3).
- Chapter 7 deviates from the constant curvature assumptions employed so far and uses Gaussian processes to construct nonparametric counterparts to the traditional algorithms presented in sections 4.5 and 4.6. The chapter does not delve into details of the algorithmic development but solves an important issue that largely prevented the applicability of Gaussian processes in this direction.

[131]: Schmidt et al. (2021), 'Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers' Appendix A

Probabilistic Curvature Estimation

Chapter 5

Probabilistic Metric Adaptation

5.1 Introduction

The preliminary parts of chapter 2 and 4 outlined the need for cheap, yet effective optimization algorithms to deal with the high dimensional input space of contemporary ML and DL models. Since optimization constitutes an integral part of the learning procedure there now exists a zoo of viable first-order optimization algorithms with minor computational overhead that utilize automatic differentiation[1] . These algorithms all have several hyperparameters that influence their efficacy of which the most influential is the learning rate. A lot of work is invested into tuning the learning rate to get a good model in the end. This chapter will explore the connection between Gaussian inference[2] and optimization with a general metric[3] . Combining the two frameworks produces a probabilistic version of the Polyak step [116] for an ideal setup with non-trivial additional information. In the absence of specialized information we develop a heuristic that automatically tunes the learning rate for popular first-order optimization algorithms, which empirically exhibits robustness across a range of popular ML tasks and datasets. The second point can be interpreted as either empirically updating the scale of a prior distribution to achieve a better update from an observation, or it can be seen as updating the scale of general metric to automatically tune the step length of the optimization algorithm.

The manuscript was originally presented in [36].
[36]: de Roos et al. (2021), 'A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization'

1: See PyTorch.optim documentation for some of the most popular algorithms.

2: See Sec. 3.2.
3: See Sec. 4.5.

Details will be provided in Sec. 5.2.
[116]: Polyak (1987), *Introduction to Optimization*

Particular focus is placed on empirical risk minimization where a loss function f that is average sum of individual losses ℓ over elements $\boldsymbol{x}_i$ of the dataset as

$$f(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} \ell(\boldsymbol{\theta}, \boldsymbol{x}_i) + \mathcal{R}(\boldsymbol{\theta}), \tag{5.1}$$

We let $i \in \mathcal{D}$ denote an index that indicates a unique datum in the dataset.

where $\boldsymbol{\theta} \in \mathbb{R}^D$ denotes the parameter to be optimized and $\mathcal{R}(\cdot)$ is an additional regularization term. In traditional fashion, to facilitate efficient training at the cost of accuracy, sub-sampling is employed which instead considers the loss over smaller batches $\mathcal{B} \subset \mathcal{D}$ as an unbiased but noisy estimate of the true loss.

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \ell(\boldsymbol{\theta}, \boldsymbol{x}_i) + \mathcal{R}(\boldsymbol{\theta}) \tag{5.2}$$

If the elements of the batch are drawn i.i.d. and $1 \ll |\mathcal{B}| \ll |\mathcal{D}|$, then by the Central Limit Theorem, $\mathcal{L}_{\mathcal{B}}$ is approximately normal distributed around the true function value

$$p(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t)) = \mathcal{N}(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t); f(\boldsymbol{\theta}_t), R), \tag{5.3}$$

with a variance R that scales with the batch size as $\mathcal{O}(1/|\mathcal{B}|)$. While stochasticity drastically reduces computational cost and may have beneficial side-effects like improved generalization [60, 164], it also complicates parameter tuning. In contrast to classic numerical optimization routines, variants of stochastic gradient-descent (SGD [122, 82] expose free parameters to the user. Chief among them is the scalar learning rate η that determines the length of the step size taken in the direction $\boldsymbol{d}_t$ as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{d}_t, \tag{5.4}$$

with $\boldsymbol{d}_t$ chosen iteratively by the optimization routine (in the case of vanilla SGD $\boldsymbol{d}_t = \nabla \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t)$). The learning rate constitutes a crude approximation to local curvature and crucially affects the convergence of the training, and by extension the performance of the model. Its optimal value depends on parameters of the model such as the network architecture, the dataset, and the optimization algorithm. Combining these aspects results in a broad range of possible learning rates for each new problem and model. Although various semi-automated and fully automated routines have been proposed to tune the learning rate [9, 99, 151], practitioners still largely rely on a manual process of repeated training runs, causing significant use of computational resources [5] and manual labor.

[60]: Hardt et al. (2016), 'Train faster, generalize better: Stability of stochastic gradient descent'
[164]: Wu et al. (2020), 'On the noisy gradient descent that generalizes as sgd'
[122]: Robbins and Monro (1951), 'A stochastic approximation method'
[82]: Kiefer and Wolfowitz (1952), 'Stochastic Estimation of the Maximum of a Regression Function'

[9]: Baydin et al. (2018), 'Online Learning Rate Adaptation with Hypergradient Descent'
[99]: Mahsereci and Hennig (2017), 'Probabilistic line searches for stochastic optimization'
[151]: Vaswani et al. (2019), 'Painless stochastic gradient: Interpolation, line-search, and convergence rates'
[5]: Asi and Duchi (2019), 'The importance of better models in stochastic optimization'

Contributions

In this work, we describe a probabilistic inference scheme that can be used as an *add-on to existing* first-order optimization methods (Section 5.2). The procedure explicitly models observation noise caused by data subsampling which in the noise-free limit recovers a generalization of the Polyak (1987) step for parameter updates. In Section 5.2 we show how various well-known optimization methods can be included in the inference and pave the way for the identification of new ones. There are several parameters associated with the inference procedure. We analyze them in detail and based on the findings arrive at a learning rate adaptation scheme (Section 5.3). It relies on a local quadratic model of the loss function, implicitly defined by the underlying optimization algorithm. The learning rate is adapted during training thereby reducing the need for outer-loop tuning. We empirically show that the proposed update rule is robust w.r.t. the learning rate, leading to stable convergence for a range of popular optimization algorithms across common deep learning benchmarks.

[116]: Polyak (1987), *Introduction to Optimization*

5.2 Method

To set the scene, consider a re-factored second-order Taylor expansion of the scalar function $f(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$ in a vector $\boldsymbol{d}$ around the current location $\boldsymbol{\theta}_t$, using the (ground-truth, full-batch) gradient $\nabla f_t \triangleq \nabla f(\boldsymbol{\theta}) \,|_{\,\boldsymbol{\theta}=\boldsymbol{\theta}_t}$ and Hessian $\boldsymbol{H}_t \in \mathbb{R}^{D\times D}$ with $[\boldsymbol{H}_t]_{ij} \triangleq \frac{\partial^2 f(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_i \partial\boldsymbol{\theta}_j} \,|_{\,\boldsymbol{\theta}=\boldsymbol{\theta}_t}$. Assuming the Hes-

sian is invertible, we write this local approximation as

$$\begin{aligned}\bar{f}_t(\boldsymbol{\theta}_t + \boldsymbol{d}) = &\underbrace{\frac{1}{2}(\boldsymbol{d} + \boldsymbol{H}_t^{-1}\nabla f_t)^\top \boldsymbol{H}_t(\boldsymbol{d} + \boldsymbol{H}_t^{-1}\nabla f_t)}_{\phi_{\boldsymbol{H}}(\boldsymbol{d})} \\ &+ \underbrace{f(\boldsymbol{\theta}_t) - \frac{1}{2}\nabla f_t^\top \boldsymbol{H}_t^{-1}\nabla f_t}_{f^*}.\end{aligned} \tag{5.5}$$

When the Hessian is positive definite, the minimum value of this quadratic approximation f^* is attained at the well-known Newton update $\boldsymbol{d}_t^\star = -\boldsymbol{H}_t^{-1}\nabla f_t$ which occurs at the point $\boldsymbol{\theta}_t^\star = \boldsymbol{\theta}_t - \boldsymbol{H}_t^{-1}\nabla f_t$. Because computing this update is computationally costly, large parts of classic convex optimization (in particular, conjugate gradients and quasi-Newton methods [109]) are concerned with efficient estimation of $\boldsymbol{d}_t^\star$ from a sequence of observed gradients. Big-data machine learning adds a new challenge to this setting, for which these classic methods are ill-equipped: significant sub-sampling noise on the gradient and (if it is computed) the Hessian.

[109]: Nocedal and Wright (2006), *Numerical Optimization* §5 & §6

Probabilistic Model

We phrase the task of locating (inferring) the minimizer $\boldsymbol{\theta}_t^\star \in \mathbb{R}^D$ of the local quadratic model given in Eq. (5.5) at iteration t based on noisy observations of the cost $\mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t)$ from Eq. (5.2) as a probabilistic inference problem: We model $\boldsymbol{\theta}_t^\star$ as a random variable, denoted $\widehat{\boldsymbol{\theta}}_t^\star$, and compute the posterior distribution of $\widehat{\boldsymbol{\theta}}_t^\star$ conditioned on $\mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t)$ via Bayes rule

$$p(\widehat{\boldsymbol{\theta}}_t^\star \mid \mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t)) = \frac{p(\mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t) \mid \widehat{\boldsymbol{\theta}}_t^\star)\, p(\widehat{\boldsymbol{\theta}}_t^\star)}{p(\mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t))}. \tag{5.6}$$

The prior $p(\widehat{\boldsymbol{\theta}}_t^\star)$ is taken to be Gaussian and centered around the current parameter value $\boldsymbol{\theta}_t$:

$$p(\widehat{\boldsymbol{\theta}}_t^\star) = \mathcal{N}(\widehat{\boldsymbol{\theta}}_t^\star; \boldsymbol{\theta}_t, \boldsymbol{W}_t), \tag{5.7}$$

where $\boldsymbol{W}_t \in \mathbb{R}^{D\times D}$ is an arbitrary symmetric positive definite covariance matrix, which is discussed further in Section 5.2. To develop the likelihood $p(\mathcal{L}_\mathcal{B}(\boldsymbol{\theta}_t) \mid \widehat{\boldsymbol{\theta}}_t^\star)$, we start by rewriting Eq. (5.5) in terms of $\boldsymbol{\theta}_t^\star$ as

$$\begin{aligned}\bar{f}_t(\boldsymbol{\theta}_t)|_{\boldsymbol{d}=0} &= \bar{f}_t(\boldsymbol{\theta}_t - \boldsymbol{d}_t^\star + \boldsymbol{d}_t^\star) \\ &= \bar{f}_t(\boldsymbol{\theta}_t + \boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star + \boldsymbol{d}_t^\star).\end{aligned} \tag{5.8}$$

Inserting this statement in Eq. (5.5) and recalling that $\boldsymbol{\theta}_t^\star - \boldsymbol{\theta}_t = \boldsymbol{d}_t^\star = -\boldsymbol{H}_t^{-1}\nabla f_t$, we obtain

Inserting $\boldsymbol{d} = 0 = -\boldsymbol{d}_t^\star + \boldsymbol{d}_t^\star = \boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star + \boldsymbol{d}_t^\star$ in Eq. (5.5) results in the equivalent expression found in Eq. (5.9). The full expression is: $\phi_{\boldsymbol{H}_t}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star + \boldsymbol{d}_t^\star) + f^\star$.

$$\begin{aligned}\bar{f}_t(\boldsymbol{\theta}_t) &= \frac{1}{2}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star)^\top \boldsymbol{H}_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star) + f^*, \\ \text{and} \quad \nabla \bar{f}_t &= \boldsymbol{H}_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star).\end{aligned} \tag{5.9}$$

Since the quadratic approximation matches the true function and its gradient at $\boldsymbol{\theta}_t$, we note that $f(\boldsymbol{\theta}_t) = \bar{f}_t(\boldsymbol{\theta}_t) = \frac{1}{2}\nabla \bar{f}_t^\top(\boldsymbol{\theta}_t - \boldsymbol{\theta}_t^\star) + f^*$.

This finding motivates us to express the noisy observation $\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t)$ in the *probabilistic* model as the following linear projection

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t) = \frac{1}{2}\boldsymbol{g}_t^\top(\boldsymbol{\theta}_t - \widehat{\boldsymbol{\theta}}_t^\star) + f^* + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, R_t), \tag{5.10}$$

f^* is the minimum of a local quadratic approximation of $\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t)$.

or equivalently by the likelihood

$$p(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t) \mid \widehat{\boldsymbol{\theta}}_t^\star) = \mathcal{N}\left(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t); \frac{1}{2}\boldsymbol{g}_t^\top(\boldsymbol{\theta}_t - \widehat{\boldsymbol{\theta}}_t^\star) + f^*, R_t\right). \tag{5.11}$$

Here we use $\boldsymbol{g}_t$ to denote a the gradient of the loss over the mini-batch and R_t is the observation noise due to sub-sampling. Under the Gaussian prior (5.7) and linear likelihood (5.11) there is a closed-form expression for the posterior, stated in Eq. (5.6). The posterior mean will serve as our next estimate, $\boldsymbol{\theta}_{t+1}$, and is given by

For $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\theta}, \boldsymbol{\mu} \in \mathbb{R}^D, \boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ and $p(y \mid \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{A}\boldsymbol{\theta} + b, R)$ with $\boldsymbol{A} \in \mathbb{R}^{1\times D}$ and $b \in \mathbb{R}, R \in \mathbb{R}_+$, it holds that $p(\boldsymbol{\theta} \mid y) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu} + \frac{\boldsymbol{\Sigma} \boldsymbol{A}^\top (y - \boldsymbol{A}\boldsymbol{\mu} - b)}{\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + R}, \boldsymbol{\Sigma} - \frac{\boldsymbol{\Sigma}\boldsymbol{A}\boldsymbol{A}^\top\boldsymbol{\Sigma}}{\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top + R})$. See Ch. 3 and Eq. (3.7) for more details.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{W}_t \boldsymbol{g}_t \frac{2(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t) - f^*)}{\boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t + R_t}. \tag{5.12}$$

This parameter update is of the same form as the generic update in Eq. (5.4) *if* $\boldsymbol{d}_t = \boldsymbol{W}_t \boldsymbol{g}_t$. In the next section we will clarify how this update corresponds to popular first-order algorithms and later look into the different parameters that are required for the inference.

Choice of Covariance

The update in Eq. (5.12) leaves the prior covariance matrix $\boldsymbol{W}_t$ as a free parameter. To be a valid covariance, it must be symmetric and positive definite. For the batch gradient $\boldsymbol{g}_t$ (i.e. $\nabla\mathcal{L}_{\mathcal{B}}$), a step in direction $\boldsymbol{d}_t = \boldsymbol{W}_t \boldsymbol{g}_t$ is a descent direction (on the batch), because $-\boldsymbol{g}_t^\top \boldsymbol{d}_t < 0$. To clarify the connection to optimization algorithms with a provided learning rate we will refer to $\boldsymbol{W}_t$ as $\eta \cdot \boldsymbol{W}_t$, with $\eta \in \mathbb{R}^+$. For different choices of $\boldsymbol{W}_t$, Eq. (5.12) can be seen as a generalization of several existing optimization algorithms.

SGD The most straightforward connection is to SGD [122] : If we consider $\eta \cdot \boldsymbol{W}_t = \eta \cdot \boldsymbol{I}$, Eq. (5.12) becomes

[122]: Robbins and Monro (1951), 'A stochastic approximation method'

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{g}_t \frac{2(\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}_t) - f^*)}{\eta \|\boldsymbol{g}_t\|^2 + R_t}. \tag{5.13}$$

For $R_t = 0$ this update recovers the Polyak step [116, 92] if f^* is known. A correctly identified f^* in the probabilistic argument above motivates a learning rate adaptation for SGD. As the optimizer approaches the optimum, $\mathcal{L}_{\mathcal{B}} \gtrsim f^*$, the rate goes towards zero. If instead the fraction above is constant across iterations, we recover the standard update for SGD with fixed learning rate.

[116]: Polyak (1987), *Introduction to Optimization*
[92]: Loizou et al. (2020), 'Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence'

General Optimizer There has been a surge of stochastic first-order optimization algorithms to tackle the requirements of machine learning

Optimizer	$\eta \cdot \boldsymbol{W}_t$
SGD	$\eta \boldsymbol{I}$
Adagrad	$\eta \left(\boldsymbol{G}_{t-1} + \operatorname{diag}(\boldsymbol{g}_t \boldsymbol{g}_t^\top)\right)^{-1/2}$
RMSprop	$\eta \left(\alpha \boldsymbol{G}_{t-1} + (1-\alpha) \operatorname{diag}(\boldsymbol{g}_t \boldsymbol{g}_t^\top)\right)^{-1/2}$
Momentum	$\eta \left(\boldsymbol{I} + \tilde{\beta} \boldsymbol{m}_{t-1} \boldsymbol{m}_{t-1}^\top\right)$
Adam	$\eta \left((1-\beta_1)\boldsymbol{V}_t + \tilde{\beta}_1 \boldsymbol{V}_t \boldsymbol{m}_{t-1} \boldsymbol{m}_{t-1}^\top \boldsymbol{V}_t\right)$

Table 5.1: Covariance matrices used for popular optimization algorithms. Each consists of a diagonal matrix and the last two have an additional rank one update with a modified scaling which we elaborate on in Section 5.2. $\boldsymbol{G}_t$ for Adagrad and RMSprop are recursively defined as the quantity inside the parenthesis starting from $\boldsymbol{G}_0 = 0$. The diagonal matrix $\boldsymbol{V}_t$ for Adam is the $\boldsymbol{G}_t^{-1/2}$ of RMSprop scaled with additional bias correction.

and deep learning in particular, many of which employ an element-wise scaling to the batch gradient $\boldsymbol{g}_t$[131]. These element-wise updates correspond to a diagonal matrix $\boldsymbol{W}_t$ in Eq. (5.12) which is the definition of an axis-aligned Gaussian distribution for the prior. A few popular popular diagonal first-order optimizers are summarized in Tab. 5.1 along with a novel interpretation of updates that involve momentum.

[131]: Schmidt et al. (2021), 'Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers'

The inference is not limited to a diagonal $\boldsymbol{W}_t$, it is just a computationally efficient approach to speed up first-order methods. Several higher order optimization methods can be included as well. In particular, if $\boldsymbol{W}_t^{-1}$ is chosen as a curvature matrix, e.g., the Hessian, Gauss-Newton, or the Fisher information matrix, then the inference amounts to an adaptive version of a Newton step, a Gauss-Newton step or Natural Gradient Descent, respectively [102].

[102]: Martens (2020), *New insights and perspectives on the natural gradient method*

In addition to the posterior mean one could also consider the posterior covariance on $\widehat{\boldsymbol{\theta}}_t^\star$ [22]. It could be propagated through optimization using a predictive Chapman-Kolmogorov equation. This could be realized as a Kalman filter prediction step, but would introduce additional empirical parameters and cost. We omit the covariance update to avoid confusion and instead focus on the useful connection to first-order optimization algorithms.

[22]: Chen et al. (2020), 'Self-Tuning Stochastic Optimization with Curvature-Aware Gradient Filtering'

Accelerated Gradient Updates

Several popular optimization algorithms employ an exponential averaging of gradients in the form of momentum [116, 141, 83] to speed up the training. Such methods can be included in the probabilistic inference in two ways. The first is to interpret the momentum term as another estimate of the *true* gradient instead of the batch gradient, essentially replacing $\boldsymbol{g}_t$ with $\boldsymbol{m}_t$ in the derivations of Section. 5.2. The second way that we opted for is to retain the batch gradient but adjust the covariance with a rank one update.

[116]: Polyak (1987), *Introduction to Optimization*
[141]: Sutskever et al. (2013), 'On the importance of initialization and momentum in deep learning'
[83]: Kingma and Ba (2015), 'Adam: A Method for Stochastic Optimization'

Momentum The Pytorch implementation of SGD with momentum uses the following update:

$$\boldsymbol{m}_t = \beta \cdot \boldsymbol{m}_{t-1} + \boldsymbol{g}_t\,,$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \boldsymbol{m}_t\,,$$

A more intuitive update used by Adam is $\boldsymbol{m}_t = \beta \boldsymbol{m}_{t-1} + (1-\beta)\boldsymbol{g}_t$. Expanding the recursion leads to the geometric series $\boldsymbol{m}_t = (1-\beta)\sum_{i=0}^t \beta^i \boldsymbol{g}_{t-i} = (1-\beta^t)\bar{\boldsymbol{g}}$ for $\beta \in (0,1)$. Adam then rescales $\boldsymbol{m}_t$ with $(1-\beta^t)$ to reduce the bias and recover the estimate $\bar{\boldsymbol{g}}_t$.

where β is a hyperparameter controlling the influence of previous gradients. This update is identified in the probabilistic model with

$$\eta \cdot \boldsymbol{W}_t \boldsymbol{g}_t = \eta \cdot \left(\boldsymbol{I} + \frac{\beta}{\langle \boldsymbol{m}_{t-1}, \boldsymbol{g}_t \rangle} \cdot \boldsymbol{m}_{t-1} \boldsymbol{m}_{t-1}^\top \right) \boldsymbol{g}_t.$$

Adam In a similar manner it is possible to express the update of Adam [83] with the diagonal matrix

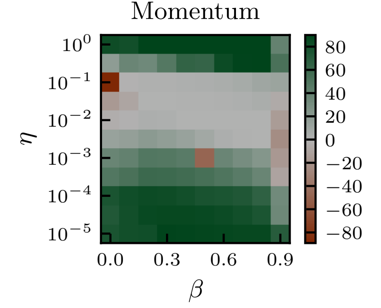


Figure 5.1: Performance of adapted vs fixed learning rate for different η and β for SGD with momentum. Green indicates an improvement of adaptation. For $\beta = 0.9$ the performance is generally the worst. See Sec. 5.4 and Fig. 5.6 for more details.

$$\boldsymbol{V}_t = \gamma_t \cdot \left(\beta_2 \cdot \boldsymbol{G}_{t-1} + (1 - \beta_2) \cdot \operatorname{diag}(\boldsymbol{g}_t \boldsymbol{g}_t^\top) \right)^{-1/2},$$
$$\gamma_t = \sqrt{(1 - \beta_2^t)/(1 - \beta_1^t)},$$

and the exponential average $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$ as

$$\eta \cdot \boldsymbol{W}_t \boldsymbol{g}_t = \eta \cdot \left((1 - \beta_1) \boldsymbol{V}_t + \tilde{\beta}_1 \boldsymbol{V}_t \boldsymbol{m}_{t-1} \boldsymbol{m}_{t-1}^\top \boldsymbol{V}_t \right) \boldsymbol{g}_t,$$
$$\tilde{\beta}_1 = \frac{\beta_1}{\langle \boldsymbol{V}_t \boldsymbol{m}_{t-1}, \boldsymbol{g}_t \rangle}.$$

Neither the Momentum nor Adam update is positive definite, per definition, in this form due to the scalar parameter in front of the rank-1 update. For SGD with momentum this scalar value will be negative if $\langle \boldsymbol{m}_{t-1}, \boldsymbol{g}_t \rangle < 0$ and violates the positive definiteness if $\beta \langle \boldsymbol{m}_{t-1}, \boldsymbol{g}_t \rangle < -\langle \boldsymbol{g}_t, \boldsymbol{g}_t \rangle$. This occurs when $\boldsymbol{m}_{t-1}$ is pointing significantly uphill at iteration t and the situation is more likely to arise for high values of β, see Fig. 5.1.

The conditions are analogous for Adam.

5.3 Algorithm

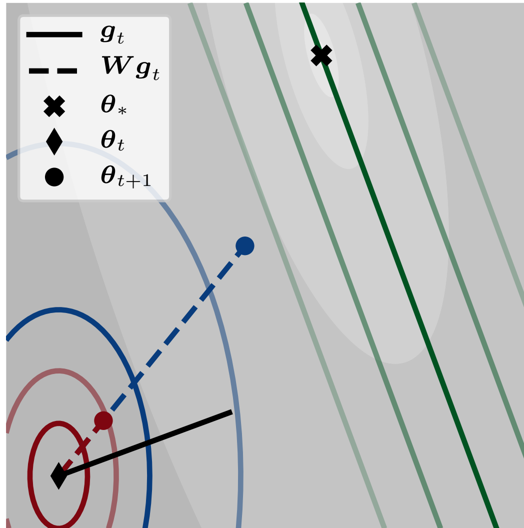


Figure 5.2: Illustration of the probabilistic inference scheme. The gray contour levels describe the local quadratic model $\bar{f}_t(\boldsymbol{\theta})$. The ( ) line comprises all *possible* values of $\widehat{\boldsymbol{\theta}}_t^\star$ (orthogonal to $\boldsymbol{g}_t$) that satisfy Eq. (5.10) (in the noise-free case); this includes the *true* value $\boldsymbol{\theta}_t^\star$. The lighter green levels are multiples of the standard deviation $\sqrt{R_t}$ representing the uncertainty in Eq. (5.10). The ( ) and ( ) colored ellipses illustrate two axis-aligned Gaussian distributions (Eq. (5.7)) centered at $\boldsymbol{\theta}_t$, corresponding to different scaling of the covariance $\eta \cdot \boldsymbol{W}_t$. The circles indicate the posterior mean ($\boldsymbol{\theta}_{t+1}$ in Eq. (5.12)) for each distribution. The blue distribution has a larger variance relative to R_t leading to a larger step towards the solution compared to the red.

The update in Eq. (5.12) is the most general form of the provided inference scheme from which one can approximate or specialize the update depending on available information and problem. While the structure of $\eta \cdot \boldsymbol{W}_t$ should be seen as a design choice addressing an algorithm within a larger family, the remaining parameters are typically problem-dependent

and play an important role in the convergence of the underlying algorithm. This section addresses this issue and constructs a simple learning rate adaptation scheme. An advantage of the probabilistic derivation is that it offers an interpretation of these parameters, which can be used to construct empirical estimators. Pseudo-code for our method that is implemented as a wrapper applicable to valid optimization algorithms is provided in Alg. 1.

Parameters

Depending on the problem, the update scheme requires estimation of up to three parameters: The scale of the prior variance/learning rate η, the lower bound f^* and the observation variance R_t. With the probabilistic motivation developed in Section 5.2, it is possible to link these parameters to function values and hence estimate them at runtime. These three parameters have an interesting interplay—the same update of $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$ (Eq. 5.12) can arise from different constellations of these three numbers. In traditional optimization, uncertainty of the observation is typically not considered, which is why R_t does not appear in the standard Polyak step.

Uncertainty R_t The most straightforward parameter to estimate in stochastic optimization is the observation variance R_t in Eq. (5.12). In the case of mini-batching it is possible to estimate the uncertainty of the full loss, as outlined via the CLT argument in Section 5.1. For general functions the situation is more complicated and since the inclusion of R_t corresponds to a reduction in step length, the same effect can be achieved by omitting R_t and instead decrease the difference in the numerator. In our algorithm we try to infer the local quadratic minimizer of the *batch* loss which yields an uncertainty estimate from the function variance of a batch.

If instead the local quadratic minimizer of the *true* loss is estimated then R_t would include additional terms to account for uncertainty in the gradient as well.

Scale η The second influential parameter is η, a scalar multiplication of the prior covariance corresponding to the learning rate of the optimizer. For the general case of Eq. (5.12) with $R_t > 0$, the ratio between R_t and $\eta \cdot \boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t$ becomes important for the overall step length and hence the convergence of the algorithm. This behavior is visualized in Fig. 5.2 where the variance of the red distribution is low compared to R_t, leading to a small update. A result of this is that the initial learning rate can be arbitrarily ill-calibrated with regards to the noise variance. It also suggests that updating the scale η during the optimization, once more parameters are estimated, could lead to improved performance. If instead $R_t = 0$ and f^* in Eq. (5.12) is known, then the update will not depend on the scale η.

Lower bound f^* The final parameter f^* is also the most important in terms of performance and stability. For a known f^* and $R_t = 0$, the Polyak step achieves a linear convergence rate *towards* f^*, independent of the Lipschitz constant which normally bounds the convergence of first-order optimization algorithms [116, 150]. If f^* is not known and set too large, then the optimizer will converge to parameters for which $f(\boldsymbol{\theta}) = f^*$ and

[116]: Polyak (1987), *Introduction to Optimization*
[150]: Vaswani et al. (2020), 'Adaptive Gradient Methods Converge Faster with Over-Parameterization (and you can do a line-search)'

not the actual minimum. If instead f^* is estimated below the minimum, then the Polyak step will try to reach function values below the minimum. This is problematic in flat regions, often resulting in steps that are too large which can undo a lot of progress. To combat this behavior, a maximum step length is often introduced as a problem dependent hyperparameter [13, 92]. The same authors showed that for Machine learning problems with overparameterized models that fulfill *interpolation*[4] it is possible to use $f^* = 0$ as a lower bound for the empirical risk minimization together with a maximum step length for fast convergence.

This is exemplified in Fig. 5.3.

[13]: Berrada et al. (2019), 'Training neural networks for and by interpolation'
[92]: Loizou et al. (2020), 'Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence'

4: models which can achieve a training loss close to 0 for all training samples simultaneously

All combined The main parameters of the probabilistic model (f^*, R_t, η) have a complicated interplay, which affects the *modus operandi* of the proposed algorithm depending on available information. Figure 5.3 highlights some of the difficulties related to naïvely setting the parameters in the probabilistic update for a deterministic optimization problem. If the global lower bound is known (0 in this case), then using it for f^* will not guarantee fast convergence in the general setting and can even lead the optimization to diverge. In such situations the optimization can be stabilized by introducing an uncertainty R_t that indicates how much f^* deviates from the minimum of a local quadratic approximation. Two examples of added uncertainty are visible in Fig. 5.3 but these are generally difficult to construct. Comparing Fig. 5.3a and Fig. 5.3b shows that even though a suitable model for R_t has been identified the performance still depends on the variance η. Adapting the variance on-the-go alleviates this problem when $R_t > 0$ and when the standard step of the underlying optimizer is used, see Fig. 5.3c. There are several ways in which these three parameters can be configured depending on the problem at hand.

This occurs for the gray line of Fig. 5.3.

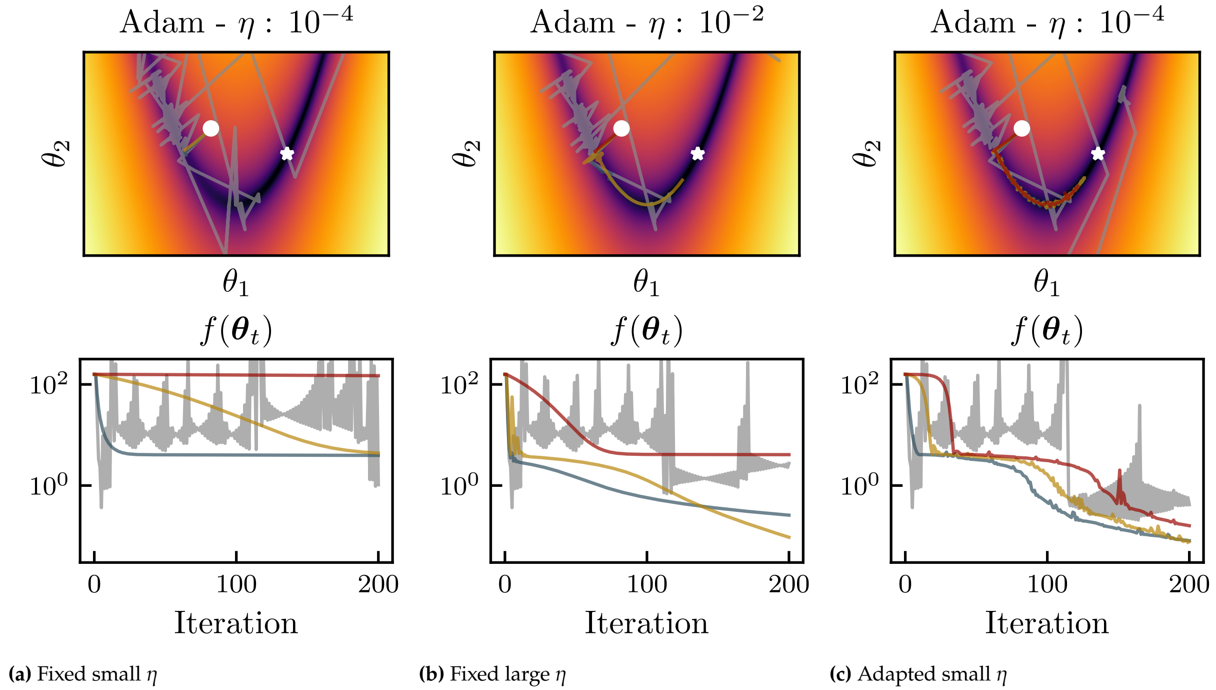
Implementation

The previous section showed that the relatively simple update of Eq. (5.12) requires careful consideration. Here we will outline the steps taken in Alg. 1 to address some of the shortcomings. Once a valid optimizer has been selected all the quantities up to and including line 5 are accounted for. A step of our algorithm then requests two parameters: a lower bound f^* and an uncertainty estimate R_t (defaults to 0). In the absence of externally provided lower bound f^* (default behavior), we still want the algorithm to adapt the learning rate during optimization in a useful manner. The approach that we use for this situation is to consider an *implied* quadratic, which given a function value $f(\boldsymbol{\theta}_t)$, gradient $\boldsymbol{g}_t$ and a spd matrix $\eta_t \cdot \boldsymbol{W}_t$ constructs a local surrogate

$$\phi_t(\boldsymbol{d}) = \phi_{\min} + \frac{1}{2} (\boldsymbol{d} + \eta_t \boldsymbol{W}_t \boldsymbol{g}_t)^\top (\eta_t \boldsymbol{W}_t)^{-1} (\boldsymbol{d} + \eta_t \boldsymbol{W}_t \boldsymbol{g}_t). \tag{5.14}$$

The parameters are chosen such that $\phi_t(\boldsymbol{0}) = f(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{d}} \phi|_{\boldsymbol{d}=0} = \boldsymbol{g}_t$. The minimum of this surrogate occurs at the step $\boldsymbol{d} = -\eta_t \boldsymbol{W}_t \boldsymbol{g}_t$ corresponding to a decrease of $\phi(0) - \phi(-\eta_t \boldsymbol{W}_t \boldsymbol{g}_t) = \eta_t \cdot \boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t / 2$. This lower bound in Eq. (5.12) amounts to a standard step of the underlying optimization algorithm if $R_t = 0$, but with an additional advantage. By re-evaluating the function at the new parameter, one can adapt the scale of the covariance/learning rate so $\phi_t - \phi_{t+1} = \eta_t \cdot \boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t / 2 \approx$

A parameter update can be seen as a step to the quadratic minimizer, Eq. (5.14) and Section 4.5, where the length is governed by the scale of W. By adapting the scale of W to better match the function it is possible to improve the convergence rate.



(a) Fixed small η **(b)** Fixed large η **(c)** Adapted small η

Figure 5.3: Influence of variance (learning rate) adaption for the inference step on the 2-d Rosenbrock (1960) function. Each figure shows the standard step of the optimizer (▬), the inference with $R_t = 0$, i.e., Polyak step (▬), inference with fixed $R_t = 0.1$ (▬) and the inference with adaptive $R_t = 0.05 f_t$ (▬). Each run used the Adam optimizer with $\beta_1 = 0.7$, $\beta_2 = 0.999$ and starting learning rate indicated in the figure title. Each of the inference steps use the correct $f^* = 0$. The gray line uses $R_t = 0$ and is therefore agnostic to the learning rate and should result in the same iterates for all three setups. This is the case up to approximately iteration 100 after which they deviate.

$f(\boldsymbol{\theta}_t) - f(\boldsymbol{\theta}_{t+1})$. The decision of which lower bound to use occurs in lines 6 to 9 of Alg. 1 and is followed by the update in Eq. (5.12).

In line 11 we re-evaluate the function (same batch) and then compare the ratio of observed and expected decrease. A ratio of 1 corresponds to a perfect match of curvature between the real function and the estimated quadratic in direction $\boldsymbol{d}_t$. If the ratio deviates from 1, we adapt the learning rate by multiplicatively increasing or decreasing η with $\alpha_\uparrow = 1.2$ and $\alpha_\downarrow = 1/2$, inspired by the update rates of Rprop [121]. Similar ideas are also employed in trust-region methods [109] to adapt the size of the trust region. One could simply choose a new η that makes the ratio 1 but this made the algorithm sensitive to outliers. Instead we apply the updates iteratively to guard against outliers which frequently occur due to the stochasticity of the problems considered. The asymmetry of the update is to penalize steps that are too large since these can be critical to the optimization procedure. We allow a bit of deviation from the optimal value of the ratio to account for stochasticity of the gradients. As the learning rate is adapted for each mini-batch it will find values that work well across batches and is suitable for the full-batch function.

[121]: Riedmiller and Braun (1992), 'Rprop-a fast adaptive learning algorithm'

[109]: Nocedal and Wright (2006), *Numerical Optimization* §4

If two successive batches have a flat and steep gradient respectively, then η can drastically increase in the first iteration leading to an excessive step in the second iteration.

Computation

The overall cost of our algorithm is essentially one forward-pass of the batch loss more expensive than that of the underlying optimizer. This is due to the requirement of re-evaluating the batch loss before

Algorithm 1: Step size adaptation for a provided optimization algorithm.

Input: $\boldsymbol{\theta}_1$: Starting point,
η: Initial learning rate,
$F(\boldsymbol{\theta})$: Function handle,
$G(\boldsymbol{\theta})$: Gradient handle,
$W(\boldsymbol{g})$: Step direction handle,
f^*: Optional lower bound,
$R(\boldsymbol{\theta}) = 0$: Optional noise estimate,

```
1  for i = 1 ... do
2      f_t = F(θ_t)
3      g_t = G(θ_t)
4      d_t = η · W(g_t)
5      φ_t = g_t^T d_t
6      if f* is available then
7          Δf = f_t − f*;                     // Use provided lower bound
8      else
9          Δf = f_t − φ_t/2;                  // Use estimated lower bound
10     θ_{t+1} = θ_t − d_t · 2 · Δf / (φ_t + R(θ_t))
11     f_+ = F(θ_{t+1});                      // Re-evaluate same batch
12     if (f_t − f_+)/(φ/2) > 4/3 then
13         η = 1.2 · η;                       // η too small α↑
14     else if (f_t − f_+)/(φ/2) > 3/4 then
15         η = 1/2 · η;                       // η too large α↓
```

the next iteration. Apart from the re-evaluation there are two non-scalar operations involved in each step: finding the step direction $\boldsymbol{d}_t = \eta \cdot \boldsymbol{W}_t \boldsymbol{g}_t$, and computing the inner product $\boldsymbol{d}_t^\top \boldsymbol{g}_t$ (i.e. $\eta \cdot \boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t$) in Eq. (5.12). The former operation is handled by the underlying optimization algorithm which does not incur any additional computational cost since the optimizer must compute it regardless. The latter operation scales linearly with the number of parameters once $\boldsymbol{d}_t$ is obtained, which is of similar complexity to the first-order optimizers in Tab. 5.1. Optimizers in Pytorch usually compute the update $\boldsymbol{d}_t$ per-parameter and apply the update immediately to save memory, thus never building the full $\boldsymbol{d}_t$. In order to keep the implementation as general as possible for the identified algorithms, we explicitly store the vector $\boldsymbol{d}_t$ for the inner product which incurs a storage requirement of an additional vector of size D. This storage can be avoided but would require implementing a new version of each optimizer.

5.4 Experiments

This section presents experimental results of relevant deep learning classification problems. We start by describing the different experiments and discuss the findings in the end. To ensure diversity in the problem set, reliable baseline comparisons and reproducible results, we made use of test problems provided by the DeepOBS benchmarking toolbox [132] . We implemented our method in Pytorch [113] ver. 1.4 as a wrapper to

[132]: Schneider et al. (2019), 'DeepOBS: A Deep Learning Optimizer Benchmark Suite' `https://deepobs.github.io`

[113]: Paszke et al. (2019), 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'

the implemented optimizers listed in Tab. 5.1. Across all experiments we used the default values of the parameters in Alg. 1. The adaptation scheme (with no f^* provided) is compared to a fixed learning rate, Hypergradient descent [9] and L4 [123] where applicable. In the absence of f^* we found no significant difference in results by including an estimate of the noise variance R_t or not, so it was left at 0. To show the effect of using a poor learning rate and the efficacy of our adaptation, we ran each experiment and optimizer with initial learning rates in the range 10^{-5} to 1. In the following figures a specific learning rate is encoded with a color ranging from 10^{-5} ( ) through 10^{-3} ( ) to 10^{0} ( ). For L4 we varied the α_{L4} parameter which scales the numerator of Eq.(5.12) in the recommended range from the default value of 0.15 ( ) to 0.25 ( ). To better show the robustness of the proposed algorithm we mainly report the results in terms of accuracy since it is constrained to [0, 1]. All of the DeepOBS results can be seen in Fig. 5.9 and 5.10. Results with the training loss follow the behaviour of Fig. 5.9 so they are presented on a *per-problem* basis in Figs. 5.4, 5.5 and 5.7 for SGD with momentum to highlight the behaviour. Apart from the momentum term of Adam and SGD with momentum for our adaptation which were set to 0.5, all other hyperparameters were kept at the Pytorch default values.

[9]: Baydin et al. (2018), 'Online Learning Rate Adaptation with Hypergradient Descent'

[123]: Rolinek and Martius (2018), 'L4: Practical loss-based stepsize adaptation for deep learning'

(F)-MNIST

DeepOBS provides several test problems that are applicable to both the Fashion MNIST and standard MNIST dataset due to the identical data format. Figure 5.4 show convergence results of the training loss for two optimization algorithms on three available problems. The models used in each problem are a logistic regressor, a 4 layered multilayered perceptron and an artificial neural network with 2 convolutional layers followed by 2 dense layers for the classification. The corresponding model is indicated on the side.

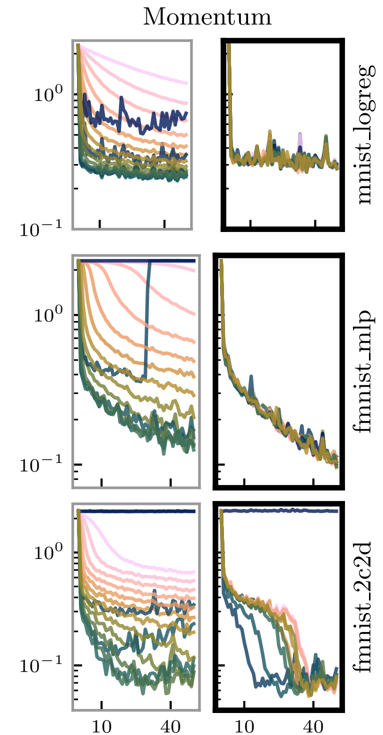


Figure 5.4: Training loss for SGD with momentum with a fixed learning rate on the left and adapted version on the right for (F)-MNIST problems.

CIFAR-10

The network used for the CIFAR-10 dataset [86] consists of 3 convolutional layers followed by 3 dense layers and l_2 regularization of $2 \cdot 10^{-3}$. Each optimizer ran for 100 epochs as opposed to 50 for the other experiments due to the slower convergence, see Fig. 5.5. The same architecture was also used to investigate how momentum affects our adaptation in deep learning. A typical result can be seen in Fig. 5.6 illustrating a sweep across learning rates (η) and momentum (β_1) for Adam. The figure shows that the performance tend to deteriorate for large values of β_1. A similar sweep took place for SGD with momentum (Fig. 5.1) to settle on a momentum β of 0.5 for both optimizers across all experiments. During the sweep we measured the average time it took to finish the training of one epoch and found that our algorithm required on average 41% longer than the standard update.

[86]: Krizhevsky (2009), 'Learning multiple layers of features from tiny images'

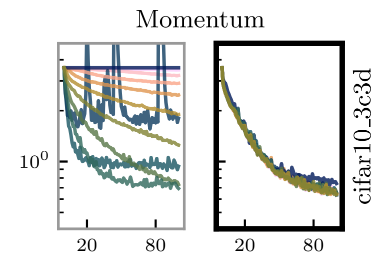


Figure 5.5: Training loss for fixed learning rate SGD with momentum on the left and adapted version on the right for a CIFAR-10 problem.

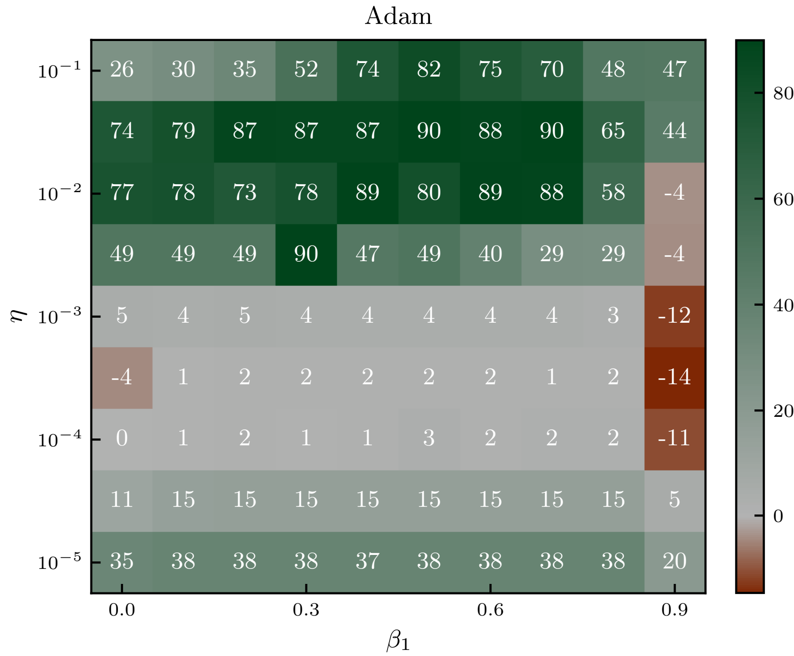


Figure 5.6: Difference in achieved training accuracy between proposed adaptation and fixed learning rate version of Adam for different initial learning rates (η) and momentum (β_1). Green color means adaptation improved and red signifies worse accuracy after 50 epochs of training on an own implementation of the test problem cifar10_3c3d found in DeepOBS. For each pair of parameters the experiment was repeated 3 times of which the best value is reported. Generally the adaptation leads to improvements except for large β_1, c.f. Section 5.2.

SVHN

The SVHN dataset [108] contains more than 600 000 images of house numbers seen from the street. One deepOBS architecture used for this experiment is a *wide resnet* [165], which is an extension of the *deep resnet* [61]. A key difference between the two architectures is that the wide resnet uses fewer and wider residual blocks, yielding improvements in training time, performance and number of parameters. The network consists of 16 convolutional layers with a widening factor of 4, and we used a batch-size of 128 and l_2 regularization of $5 \cdot 10^{-4}$. A typical convergence of training loss is seen in Fig. 5.7 and more optimizers and metrics are available in Fig. 5.9 and 5.10.

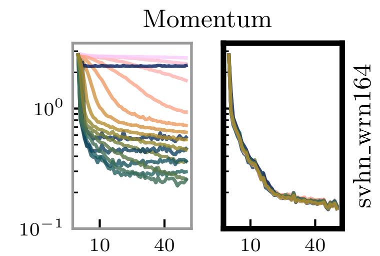


Figure 5.7: Training loss for fixed learning rate SGD with momentum on the left and adapted version on the right for the SVHN problem.

[108]: Netzer et al. (2011), 'Reading Digits in Natural Images with Unsupervised Feature Learning'
[165]: Zagoruyko and Komodakis (2016), 'Wide Residual Networks'
[61]: He et al. (2016), 'Deep Residual Learning for Image Recognition'

CIFAR-100

To test the optimization on a larger model and dataset we used the ResNet18 implementation from the Pytorch model zoo and trained the model on the CIFAR-100 dataset with a batch size of 128. The used l_2-regularization of $5 \cdot 10^{-4}$ was too low for the model which resulted in overfitting and poor generalization performance, but the overall trend compared to the problems from DeepOBS is still visible. In Fig. 5.8 we see different metrics evolve during the training and the learning rate. For each of the optimizers there seems to be an initial convergence point for the learning rate that then transitions into a more noisy regime.

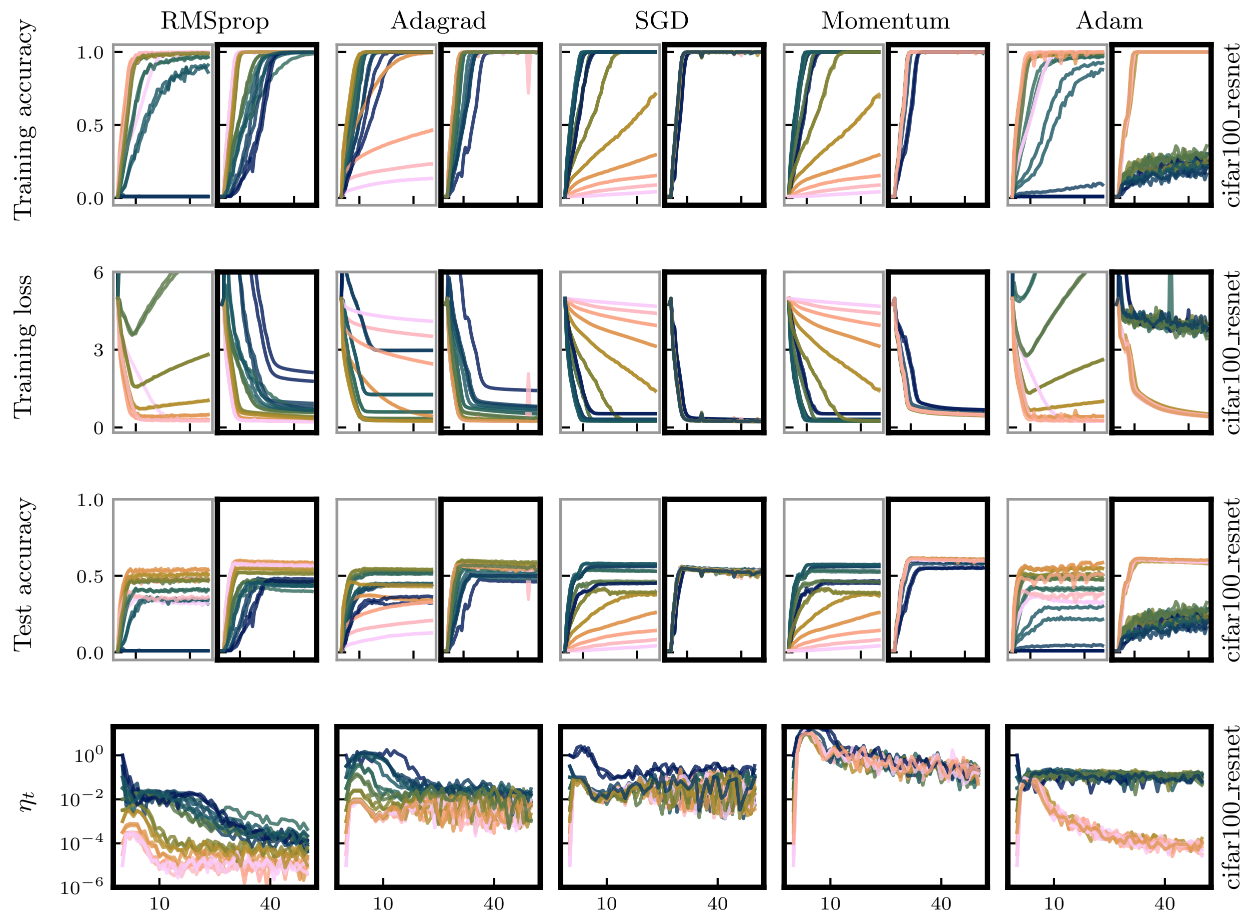


Figure 5.8: Results for a model trained on CIFAR-100 for 50 epochs. All the settings and limits are the same as the results from DeepOBS in Fig. 5.9 and Fig. 5.10. The last row shows the learning rate that was used at the end of each epoch for the proposed learning rate adaptation.

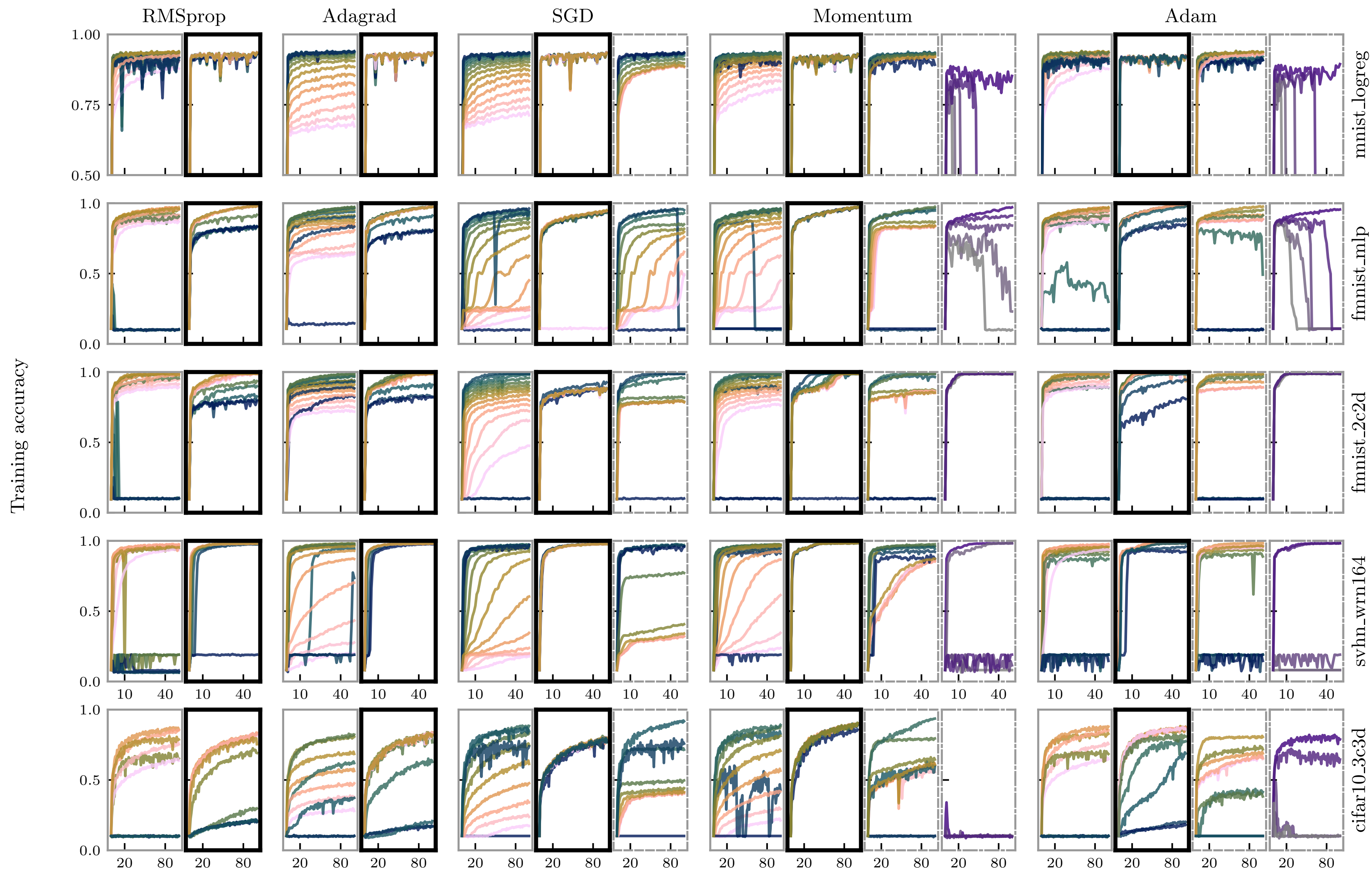


Figure 5.9: Training accuracy (higher is better) per epoch for different optimizers, learning rate adaptations and benchmark problems from DeepOBS. Each row shows the training accuracy for one test problem identified by a dataset and model description and each columngroup shows a family of optimizers. The leftmost graph in each group (thin gray border) has a fixed learning rate. Next to it (thick black border) is the proposed adaptation. A dashed border indicates results for Hypergradient descent and the dash-dotted show results for L4. Each graph contains experiments with initial learning rates in the range 10^{-5} ( ) through 10^{-3} ( ) to 10^{0} ( ). In the case of L4 the learning is replaced with α_{L4} with values between 0.15 ( ) and 0.25 ( ). In every problem each optimizer ran for 50 epochs except for cifar10_3c3d which ran for 100 epochs. All hyperparameters were left at the default values except for the momentum term of the proposed adaptation which was set to 0.5 instead of default 0.9 for Momentum and Adam. The graphs under SGD show a typical example of how sensitive the performance of a model is to a fixed learning rate during training and how the adaptation avoids this problem.

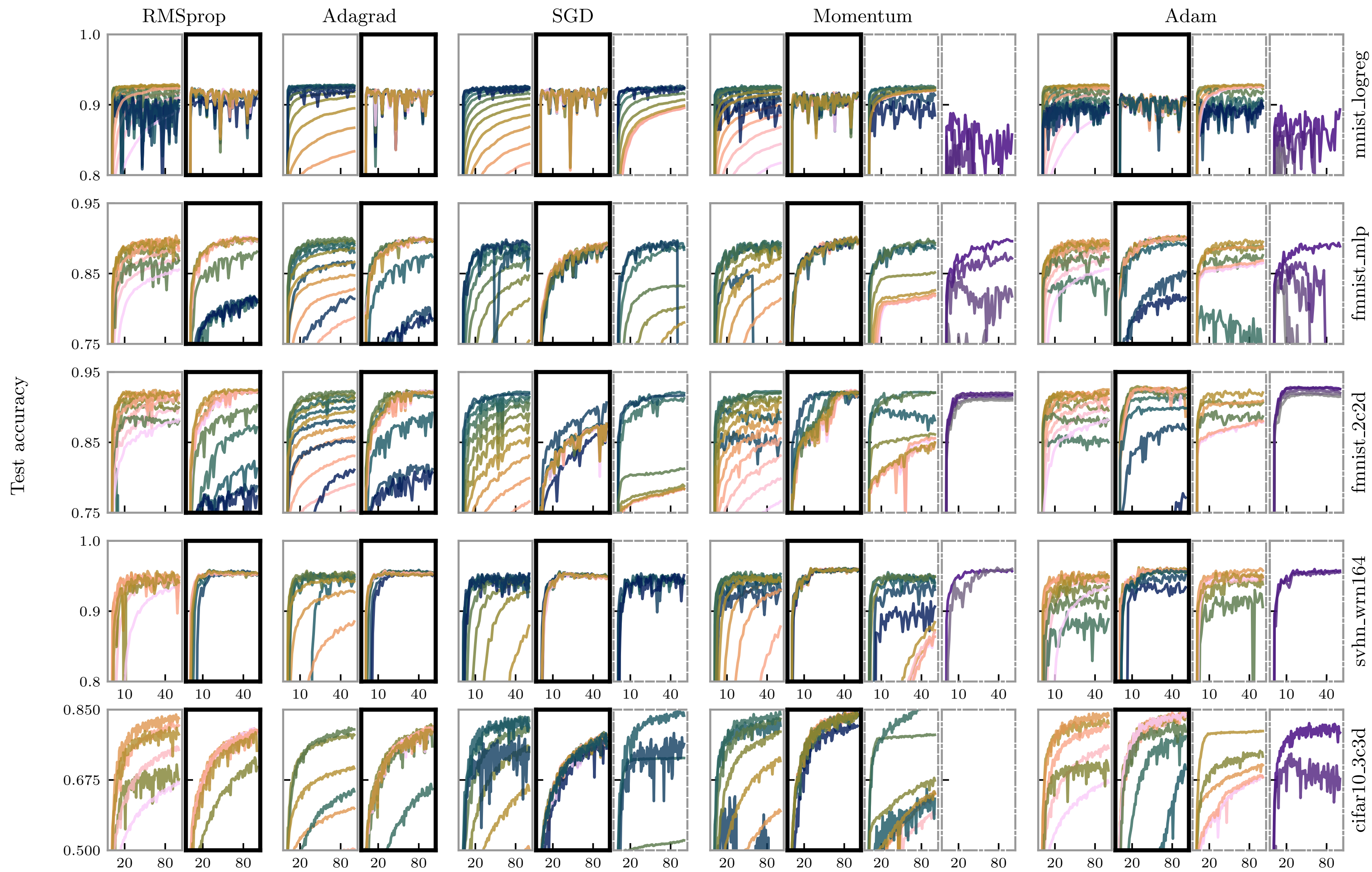


Figure 5.10: Training accuracy (higher is better) per epoch for different optimizers, learning rate adaptations and benchmark problems from DeepOBS. Each row shows the training accuracy for one test problem identified by a dataset and model description and each columngroup shows a family of optimizers. The leftmost graph in each group (thin gray border) has a fixed learning rate. Next to it (thick black border) is the proposed adaptation. A dashed border indicates results for Hypergradient descent and the dash-dotted show results for L4. Each graph contains experiments with initial learning rates in the range 10^{-5} ( ) through 10^{-3} ( ) to 10^{0} ( ). In the case of L4 the learning is replaced with α_{L4} with values between 0.15 ( ) and 0.25 ( ). In every problem each optimizer ran for 50 epochs except for cifar10_3c3d which ran for 100 epochs. All hyperparameters were left at the default values except for the momentum term of the proposed adaptation which was set to 0.5 instead of default 0.9 for Momentum and Adam. The graphs under SGD show a typical example of how sensitive the performance of a model is to a fixed learning rate during training and how the adaptation avoids this problem.

Discussion

The results presented in Fig. 5.9 and 5.10 show that the proposed learning rate adaptation is robust across a wide range of classification problems and optimizers. It reliably adjusts the learning rate which most times results in a final accuracy close to the best achieved accuracy on the task. The exception being SGD which in some cases falls behind due to the higher variance of $\eta \cdot \boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t$ compared to other algorithms.

Albeit the notable robustness, certain initializations of the learning rate are still too large for the optimization to converge, which is visible from the straggling dark/black lines in certain problems. In these cases the learning rates are several orders of magnitude larger than the optimal fixed learning rate and neither adaptation converges.

An example of too large learning rate is seen for Adam in the F-MNIST tasks.

Hypergradient descent often shows improvements over the corresponding fixed learning rate version but sometimes gets stuck for too small learning rates and it does not show the same agnosticism towards the initial learning rate. The update to the learning rate for Hypergradient descent is calculated from the inner product of two subsequent gradients and scaled with a small hyper learning rate. Since the size and architecture of the considered networks drastically vary between problems, the default value of the hyper learning rate is bound to be off for some architectures, requiring additional tuning for optimal performance. Our method instead updates the learning rate based on a dimensionless quantity making it less sensitive to variations in the network.

Hypergradient is seen struggling with a too small learning rate in the F-MNIST experiments.

L4 estimates f^* and uses a form of Polyak step in each iteration making each parameter update independent of the learning rate. The algorithm instead introduces additional hyperparameters which the authors empirically set for good performance. When L4 finished a training run without diverging it was usually among the fastest to reach a high accuracy, but the results in Fig. 5.9 show that the default parameter values would still require additional tuning depending on dataset and model making them less robust across problems.

Our implemented adaptation updates the learning rate for every batch throughout the training, leading to an overall computational cost on average $< 50\%$ higher than that of the underlying optimizer. The majority of the additional cost stems from re-evaluating the loss on the same batch. A simple remedy to reduce the overhead is to not evaluate every batch or epoch, but every 2^tth epoch for $t = 0, 1, ...$ This allows significant adaptation in the beginning to get the scale right and less frequently during later stages of training, see the learning rate in Fig. 5.8 for motivation. Overall, the additional cost of the re-evaluation is justified if it means that no additional runs are required to find a suitable learning rate.

As outlined in Sec.15.

One recurring observation from the experiments with the adaptation is that the smaller initial learning rates converge without exception, suggesting one could initialize the underlying optimizer with a small learning rate ($10^{-4} - 10^{-3}$) and let the adaptation accelerate to a suitable level.

5.5 Related Work

Stochastic gradient descent and its variants remain the workhorse for the stochastic optimization in deep learning, and big-data machine learning more generally. Several methods that improve the convergence over standard SGD by reducing the variance of the estimate [141], adapting the step-direction [40, 130, 30] or combinations thereof [83] have been proposed as substitutes. The learning rate is the single most important hyperparameter in these first-order optimization methods that are used in machine learning, with the model performance hinging on successful selection [57]. A too large value can ruin the convergence, while a too small value makes the progress very slow and the optimizer risks getting completely stuck without further improvement. A few recent ideas to reduce this influence are to include the learning rate as an additional parameter that can be optimized with backpropagation cf. results in Fig. 5.9 [9] or to train another model to predict the next step [2]. One cane also use a line search routine, with new iterates chosen to satisfy conditions that ensure suitable convergence [4]. Stochastic versions of these line searches were proposed by Mahsereci and Hennig [99] and Vaswani et al. [151]. An advantage, in terms of simplicity, of our framework over these methods is that it only requires a single additional function evaluation, keeping the iteration cost comparably low.

[141]: Sutskever et al. (2013), 'On the importance of initialization and momentum in deep learning'
[40]: Duchi et al. (2011), 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'
[130]: Schaul et al. (2013), 'No More Pesky Learning Rates'
[30]: Dauphin et al. (2015), 'RMSProp and equilibrated adaptive learning rates for non-convex optimization'
[83]: Kingma and Ba (2015), 'Adam: A Method for Stochastic Optimization'

See [125]: Ruder (2016), *An overview of gradient descent optimization algorithms* for an overview of algorithms.

[57]: Goodfellow et al. (2016), *Deep Learning*
[9]: Baydin et al. (2018), 'Online Learning Rate Adaptation with Hypergradient Descent'
[2]: Andrychowicz et al. (2016), 'Learning to learn by gradient descent by gradient descent'
[4]: Armijo (1966), 'Minimization of functions having Lipschitz continuous first partial derivatives'
[99]: Mahsereci and Hennig (2017), 'Probabilistic line searches for stochastic optimization'
[151]: Vaswani et al. (2019), 'Painless stochastic gradient: Interpolation, line-search, and convergence rates'

The Polyak (1987) step, which is a special case of our probabilistic treatment, has previously been used in machine learning for models that satisfy *interpolation* [92]. Loizou et al. [92] used the Polyak step together with SGD and proved a convergence rate for the algorithm. Around the same time Berrada et al. [13] proposed the ALI-G algorithm which also amounts to a stochastic Polyak step for SGD and a version that incorporates a form of momentum update.

[116]: Polyak (1987), *Introduction to Optimization*
[92]: Loizou et al. (2020), 'Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence'
[13]: Berrada et al. (2019), 'Training neural networks for and by interpolation'

The L4 optimizer of Rolinek and Martius [123] estimates $f*$ and uses the Polyak step to train deep models. Compared to our algorithm it relies on different estimators for the gradient to specifically speed up Momentum and Adam. It also avoids the function re-evaluation but instead introduces additional hyperparameters to estimate the lower bound f^*, making it more sensitive to varying problem setups, cf. results in Fig. 5.9. Such an estimate is also possible to include in our algorithm but was not considered further but instead we focused on the scaling of the covariance.

[123]: Rolinek and Martius (2018), 'L4: Practical loss-based stepsize adaptation for deep learning'

Another concurrent line of research is that of Vaswani et al. [150] who extend the line search of Vaswani et al. [151] and the Polyak step of Loizou et al. [92] for problems that satisfy interpolation. The main contribution was to use a general metric to recover additional optimization algorithms (the diagonal versions in Tab. 5.1) and analyze the convergence properties. Compared to our work it does not consider the connection to probabilistic inference nor the additional optimizers. It is similar to this work in the sense that Gaussian inference also uses a general metric induced by the inverse covariance matrix. Moreover, we do not specifically consider the interpolation setting but instead aim at adapting the learning rate for general problems. The usage of a line search introduces the need for ≥ 1 additional function evaluations per batch whereas ours rely on a single re-evaluation.

[150]: Vaswani et al. (2020), 'Adaptive Gradient Methods Converge Faster with Over-Parameterization (and you can do a line-search)'
[92]: Loizou et al. (2020), 'Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence'

The derivations of Sec. 5.2 are reminiscent of probabilistic linear algebra routines with additional noise [66, 35, 25]. Our algorithm could operate in a similar manner if the same batch and Hessian is used for repeated parameter updates and the posterior covariance is propagated. Instead we focused on the connection to first-order optimization algorithms for large-scale machine learning tasks.

[66]: Hennig et al. (2015), 'Probabilistic numerics and uncertainty in computations'
[35]: de Roos and Hennig (2019), 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'
[25]: Cockayne et al. (2019), 'A Bayesian conjugate gradient method'

5.6 Conclusion

We have proposed an algorithm motivated by Gaussian inference, to construct a family of update rules that perform a learning rate adaptation for popular first order stochastic optimization routines. The algorithm is applicable to optimization routines where the step direction can be phrased as the product of a symmetric, positive definite matrix with the gradient. It uses a local quadratic approximation of the loss function defined by the underlying optimization algorithm to adaptively scale the step size. In our experiments, the algorithm is able to efficiently adapt the learning rate across a wide range of initial learning rates, optimizers and deep learning problems. The robust algorithm achieves competitive performance compared to hand-tuned learning rates, Hypergradient descent and the L4 optimizer with less tuning required. The proposed adaptation scheme thus offers a way to automatically update the learning rate of deep learning optimizers within the inner loop – removing the need for outer-loop parameter tuning of the learning rate which comes at high cost in terms of human labor and hardware resources.

5.7 Future Directions

Arguably the most obvious extension of this chapter is to include other curvature estimates in the developed framework. While valuable in its own right the results in Sec. 5.4 suggest that there is a very similar performance between most of the adapted optimization algorithms and that this direction potentially only holds marginal benefits. Instead there are two more promising directions that could yield new insights and greater flexibility.

The first approach was already hinted at in Sec. 5.2 where the momentum term of several popular optimization algorithms can be seen as an exponential moving average of the gradient [123]. Adapting the presented framework to operate on the momentum instead of the gradient would only require minor modifications to the algorithm. These include making sure the gradient is an unbiased estimate by appropriate re-scaling, updating the inner product $\boldsymbol{g}_t^\top \boldsymbol{W}_t \boldsymbol{g}_t$ accordingly and properly scaling the function when updating the curvature estimate.

[123]: Rolinek and Martius (2018), 'L4: Practical loss-based stepsize adaptation for deep learning'

The second approach would be to wrap the framework in a more probabilistic interpretation. It could be possible to use a Kalman filter to track the scalar learning rate or the local lower bound f^*. The former would be straightforward to implement and could replace the seemingly arbitrary yet well-performing scaling factors $\alpha_\downarrow$ and $\alpha_\uparrow$ in Alg. 1. The latter would be more intricate but potentially with larger benefits. It would allow a

more natural learning rate decay that is based on available data and to a lesser extent the heuristics that are currently employed.

If a learning rate adaptation, like the one presented, could be combined with a momentum adaptation as well then a significant part of the difficulties introduced by stochastic optimization would be addressed. The resulting algorithm would be close to a stochastic version of CG which chooses a step length according to the aforementioned routines and updates the momentum according to observed data. This kind of data for the momentum could be obtained from the alignment of subsequent gradients similar to how conjugacy in CG is determined.

Chapter 6

Matrix Inference for Curvature Learning

Some of the most important algorithms for unconstrained optimization approximate the underlying curvature by observing gradients. This leads to algorithms that interpolate between GD and the Newton update in terms of parameter update cost and reduction in function value. Oftentimes these QN algorithms can drastically improve the convergence rate at a minor computational overhead, making them the go-to algorithms for deterministic unconstrained optimization [16].

The manuscript was originally published in [35].

[35]: de Roos and Hennig (2019), 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'

See introductory chapter Ch. 4.

[16]: Boyd and Vandenberghe (2004), *Convex optimization*

The usage of QN methods in stochastic optimization is so far quite limited. This is mainly because there has been no clear way to deal with the stochasticity of the gradient in a computationally feasible manner to estimate the curvature.

A situation often encountered in ML due to large datasets.

In this chapter we will address this issue with a probabilistic model of the curvature matrix along with uncertain gradient observations. A matrix-variate normal distribution is used to estimate of the curvature in the presence of noise. The estimate is used to speed up SGD for typical ML problems. The same rules and equations of Gaussian inference encountered in Sec. 3.2 are valid, they are just applied to an object of higher dimensionality with certain modeling choices to reduce the computational complexity.

6.1 Introduction

The most common optimization objective in machine learning that also is the focus of this chapter is empirical risk minimization. A loss function to be minimized is constructed in the form

See Ch. 2 for background.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}}^{|\mathcal{D}|} \ell_i(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta}), \tag{6.1}$$

where $\ell_i(\cdot)$ is the individual loss of a single datum from the dataset $\mathcal{D}$, $\boldsymbol{\theta}$ are model parameters to be optimized (e.g. the weights of a neural network) and $\mathcal{R}(\cdot)$ is a form of regularization. Evaluating the full loss in every iteration is computationally expensive for large datasets. A popular approach to reduce the computational overhead is to instead consider smaller batches. This data sub-sampling gives rise to a stochastic optimization problem. If a gradient is computed for a randomly sampled

batch $\mathcal{B} \subset \mathcal{D}$ of data points it results in a stochastic gradient

$$\nabla \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}}^{|\mathcal{B}|} \nabla \ell_i(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta}). \tag{6.2}$$

If $\mathcal{D}$ is large and $\mathcal{B}$ is sampled i.i.d., then $\nabla \tilde{\mathcal{L}}$ is an unbiased estimator and, by the multivariate Central Limit Theorem, is approximately Gaussian distributed (assuming $|\mathcal{B}| \ll |\mathcal{D}|$) around the full-data gradient [7, 146]

[7]: Balles et al. (2017), 'Coupling Adaptive Batch Sizes with Learning Rates'
[146]: van der Vaart (1998), *Asymptotic Statistics*

$$p\left(\nabla \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) \middle| \nabla \mathcal{L}\right) = \mathcal{N}\left(\nabla \tilde{\mathcal{L}}(\boldsymbol{\theta}); \nabla \mathcal{L}(\boldsymbol{\theta}), |\mathcal{B}|^{-1} \operatorname{cov}(\nabla \mathcal{L})\right), \tag{6.3}$$

where $\operatorname{cov}(\nabla \mathcal{L})$ is the empirical covariance over $\mathcal{D}$. For problems of large scale in both data and parameter-space cheap optimization algorithms like stochastic gradient descent [122] and its by now many variants (e.g., momentum [117], Adam [83], etc.) are standard procedure.

[122]: Robbins and Monro (1951), 'A stochastic approximation method'

[117]: Polyak (1964), 'Some methods of speeding up the convergence of iteration methods'

[83]: Kingma and Ba (2015), 'Adam: A Method for Stochastic Optimization'

For (non-stochastic) gradient descent, it is a classic result that the rate of convergence depends on the condition-number of the objective's Hessian $\boldsymbol{H}(\boldsymbol{\theta}) = \nabla\nabla^\top \mathcal{L}(\boldsymbol{\theta})$. For example, Thm. 3.4 in the book of Nocedal and Wright [109] states that the iterates of gradient descent with optimal local step sizes on a twice-differentiable objective $\mathcal{L}(\boldsymbol{\theta}) : \mathbb{R}^N \to \mathbb{R}$ converge to a local optimum $\boldsymbol{\theta}_*$ such that, for sufficiently large t,

[109]: Nocedal and Wright (2006), *Numerical Optimization*

$$\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_*) \leq r^2(\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}_*)), \quad \text{with} \quad r \in \left(\frac{\kappa - 1}{\kappa + 1}, 1\right), \tag{6.4}$$

where the condition number $\kappa = \lambda_1/\lambda_D$ is the ratio of between the largest and smallest eigenvalues of the Hessian $\boldsymbol{H}(\boldsymbol{\theta}_t)$. In noise-free optimization, it is therefore common practice to try and reduce the effective condition number of the Hessian by a linear re-scaling $\tilde{\boldsymbol{\theta}} = \boldsymbol{P}^\top \boldsymbol{\theta}$ of the input space using a *preconditioner* $\boldsymbol{P} \in \mathbb{R}^{D \times D}$.

See Sec. 4.5 for further details.

For ill-conditioned problems ($\kappa \gg 1$), an effective preconditioning strategy can drastically improve the convergence rate. There is however a trade-off to be made when considering options for preconditioners. A good preconditioner should decrease the condition number of the problem and be cheap to apply, either through sparseness, low-rank structure or other tricks [71]. Choosing a good preconditioner is a problem-dependent art and is seldom straightforward.

[71]: Hoffman et al. (2013), 'Stochastic variational inference.'

Sometimes preconditioners can be constructed as a good "a-priori" guess of the inverse Hessian $\boldsymbol{H}^{-1}$. If no such information is available, then in the deterministic/non-stochastic setting, iterative linear-algebra methods can be used to build a low-rank preconditioner. For example, if $\boldsymbol{P}$ spans the leading eigenvectors of the Hessian, the corresponding eigenvalues can be re-scaled to change the spectrum, and by extension the condition number. Iterative methods such as those based on the Lanczos process [56] try to consecutively expand a helpful subspace by choosing $\boldsymbol{H}$-conjugate vectors based on Hessian-vector products. These methods are sensitive to numerical accuracy and often fail in the presence of inexact computations [143] . Due to these intricate instabilities such algorithms tend not to work with the level of stochasticity encountered in practical machine learning applications.

[56]: Golub and Van Loan (2012), *Matrix computations* §10.1

[143]: Trefethen and Bau III (1997), *Numerical Linear Algebra* p. 282

This chapter proposes a framework for efficient construction of pre-

conditioners in settings with noise-corrupted Hessian-vector products available. The main algorithm consists of three components: We first build a probabilistic Gaussian inference model for matrices from noisy matrix-vector products by extending existing work on matrix-variate Gaussian inference[1] . Then we construct an algorithm that actively selects informative vectors aiming to explore the dominant eigendirections of the Hessian[2] . The structure of this algorithm is similar to the classic Arnoldi and Lanczos iterations designed for noise-free problems. Finally, we provide some "plumbing" to empirically estimate useful hyper-parameters and efficiently construct a low-rank preconditioner which can be applied to high-dimensional models[3] . We evaluate the algorithm on some simple experiments to empirically study its properties as a way to construct preconditioners and test it on both low-dimensional problems, and a high-dimensional deep learning problem.

1: Matrix-variate inference in Section 6.4.

2: Details regarding algorithm in Section 6.6 and 6.7.

3: Technical details in Section 6.8.

While we use preconditioning as the principal application, the main contribution of our framework is the ability to construct matrix-valued estimates in the presence of noise, and to do so at complexity *linear* in the width and height of the matrix. It is thus applicable to problems of large scale, also in domains other than preconditioning and optimization in general. In contrast to a simple averaging of random observations, our algorithm actively chooses projections in an effort to improve the estimate.

6.2 Related Work

Our approach is related to optimization methods that try to emulate the behavior of Newton's method without incurring its cubic per-step cost. That is, iterative optimization updates $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{d}_t$ that try to find a search direction $\boldsymbol{d}_t$ that is an approximate solution to the linear problem

$$\boldsymbol{H}(\boldsymbol{\theta}_t)\boldsymbol{d}_t = \nabla\mathcal{L}(\boldsymbol{\theta}_t). \tag{6.5}$$

This includes quasi-Newton methods (QN) like BFGS and its siblings [37], and Hessian-free optimization [114, 101]. These methods try to keep track of the Hessian during the optimization. Preconditioning is a simpler approach that separates the estimation of the (inverse) Hessian from the on-line phase of optimization and moves it to an initialization phase. Our algorithm could in principle be run in every single step of the optimizer, such as in Hessian-free optimization. However, this would multiply the incurred cost, which is why we here only study its use for preconditioning.

[37]: Dennis and Moré (1977), 'Quasi-Newton methods, motivation and theory'
[114]: Pearlmutter (1994), 'Fast exact multiplication by the Hessian'
[101]: Martens (2010), 'Deep learning via Hessian-free optimization'

There are stochastic variants of quasi-Newton algorithms originally constructed for noise-free optimization [134, 21]. These are generally based on collecting independent random (not actively designed) samples of quasi-Newton updates. Estimates can also be obtained by regularizing estimates [156] or partly update the stochastic gradient by reusing elements of a batch [15]. The conceptual difference between these methods and ours is that we actively try to design informative observations by explicitly taking evaluation uncertainty into account.

[134]: Schraudolph et al. (2007), 'A stochastic quasi-Newton method for online convex optimization'
[21]: Byrd et al. (2016), 'A stochastic quasi-Newton method for large-scale optimization'
[156]: Wills and Schön (2018), 'Stochastic quasi-Newton with adaptive step lengths for large-scale problems'
[15]: Bollapragada et al. (2018), 'A Progressive Batching L-BFGS Method for Machine Learning'

Our inference scheme is an extension of Gaussian models for inference on matrix elements, which started with early work by Dawid [32] and

[32]: Dawid (1981), 'Some matrix-variate distribution theory: Notational considerations and a Bayesian application'

was recently extended in the context of probabilistic numerical methods [65, 64, 156]. Our primary addition to these works is the algebra required for dealing with structured observation noise.

[65]: Hennig and Kiefel (2013), 'Quasi-Newton method: A new direction'
[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'
[156]: Wills and Schön (2018), 'Stochastic quasi-Newton with adaptive step lengths for large-scale problems'

6.3 Theory

Our goal in this work is to construct an active inference algorithm for low-rank preconditioning matrices that can deal with data in the form of Hessian-vector multiplications corrupted by Gaussian noise. To this end, we will adopt a probabilistic viewpoint with a Gaussian observation likelihood, and design an active evaluation policy that aims to efficiently collect informative, non-redundant Hessian-vector products. The algorithm will be designed so that it produces a Gaussian posterior measure over the Hessian of the objective function, such that the posterior mean is a low-rank matrix.

6.4 Matrix Inference

Bayesian inference on matrices $\boldsymbol{H} \in \mathbb{R}^{D\times D}$ can be realized efficiently in a Gaussian framework by re-arranging the matrix elements into a vector $\mathrm{vec}(\boldsymbol{H}) \in \mathbb{R}^{D^2\times 1}$, then performing standard Gaussian inference on this vector [32]. Although Hessian matrices are square and symmetric, the following derivations apply equally well to rectangular matrices. There are specializations for symmetric matrices [64], but since they significantly complicate the derivations below, we use this weaker model.

[32]: Dawid (1981), 'Some matrix-variate distribution theory: Notational considerations and a Bayesian application'

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

Assume that we have access to observations $\boldsymbol{Y} \in \mathbb{R}^{D\times M}$ of matrix-vector products $\boldsymbol{Y} = \boldsymbol{HS}$ along the search directions $\boldsymbol{S} \in \mathbb{R}^{D\times M}$. In the vectorized notation, this amounts to a linear projection of $\mathrm{vec}(\boldsymbol{H})$ through the Kronecker product matrix $(\boldsymbol{S}^\top \otimes \boldsymbol{I})$ [62]:

[62]: Henderson and Searle (1981), 'The vec-permutation matrix, the vec operator and Kronecker products: A review'

$$\begin{aligned}\mathrm{vec}(\boldsymbol{Y}) &= (\boldsymbol{S}^\top \otimes \boldsymbol{I})\,\mathrm{vec}(\boldsymbol{H})\\ \boldsymbol{Y} &= \boldsymbol{IHS}.\end{aligned}$$

If the observations are exact (noise-free), the likelihood function is a Dirac distribution,

$$\begin{aligned}p(\boldsymbol{Y}\,|\,\boldsymbol{H},\boldsymbol{S}) &= \delta(\mathrm{vec}(\boldsymbol{Y}) - (\boldsymbol{S}^\top \otimes \boldsymbol{I})\boldsymbol{H}) =\\ &= \lim_{\beta\to 0} \mathcal{N}(\boldsymbol{Y}; (\boldsymbol{S}^\top \otimes \boldsymbol{I})\boldsymbol{H}, \beta\boldsymbol{\Lambda}_0).\end{aligned} \tag{6.6}$$

For conjugate inference, we assign a Gaussian prior over $\boldsymbol{H}$, with a prior mean matrix $\boldsymbol{H}_0$ and a covariance consisting of a Kronecker product of 2 symmetric positive-definite matrices.

$$\begin{aligned}\mathcal{N}(\boldsymbol{H}, \boldsymbol{H}_0, \boldsymbol{V}\otimes\boldsymbol{W}) &= \frac{1}{((2\pi)^{n^2}/|\boldsymbol{V}|^n|\boldsymbol{W}|^n)^{1/2}}\\ \cdot \exp&\left(-\frac{1}{2}(\mathrm{vec}(\boldsymbol{H}) - \mathrm{vec}(\boldsymbol{H}_0))^\top(\boldsymbol{V}\otimes\boldsymbol{W})^{-1}(\mathrm{vec}(\boldsymbol{H}) - \mathrm{vec}(\boldsymbol{H}_0))\right).\end{aligned} \tag{6.7}$$

This is an equivalent re-formulation of the *matrix-variate Gaussian* distribution[4] . For simplicity, and since we are inferring a Hessian matrix $\boldsymbol{H}$

4: See [58] for extensive exposition of matrix-variate distributions.
[58]: Gupta and Nagar (2018), *Matrix variate distributions*

(which is square and symmetric), we set $V = W$. This combination of prior (6.7) and likelihood (6.6) has previously been discussed[5] . It gives rise to a Gaussian posterior distribution, whose mean matrix is given by

5: See [64] for detailed derivations.
[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

$$\begin{aligned} \boldsymbol{H}_m = &\boldsymbol{H}_0 + (\boldsymbol{W} \otimes \boldsymbol{W})(\boldsymbol{S} \otimes \boldsymbol{I}) \operatorname{vec}(\boldsymbol{X}) = \boldsymbol{H}_0 + \boldsymbol{W}\boldsymbol{X}\boldsymbol{S}^\top \boldsymbol{W} = \\ &\boldsymbol{H}_0 + (\boldsymbol{Y} - \boldsymbol{H}_0\boldsymbol{S})(\boldsymbol{S}^\top \boldsymbol{W}\boldsymbol{S})^{-1}\boldsymbol{S}^\top \boldsymbol{W} \end{aligned} \tag{6.8}$$

where $\boldsymbol{X}$ is found as the solution to the linear system

$$\underbrace{(\boldsymbol{S}^\top \boldsymbol{W}\boldsymbol{S} \otimes \boldsymbol{W})}_{\boldsymbol{G}} \operatorname{vec}(\boldsymbol{X}) = \operatorname{vec}(\underbrace{(\boldsymbol{Y} - \boldsymbol{H}_0\boldsymbol{S})}_{\operatorname{vec}(\boldsymbol{\Delta})}), \tag{6.9}$$

using the Kronecker product's property that $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$ (note that the matrix $(\boldsymbol{S}^\top \boldsymbol{W}\boldsymbol{S}) \in \mathbb{R}^{M\times M}$ can be inverted in $\mathcal{O}(M^3)$).

Assuming $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible.

6.5 Adding Noise

While noise-free observations of Gaussian-distributed matrices have been studied before, observation noise beyond the fully i.i.d. case [156], is a challenging complication. A lightweight inference-scheme for noisy observations is one of the core contributions of this paper. In empirical risk minimization problems, mini-batching replaces a large or infinite sum into a small sum of individual loss functions[6] . Analogous to Eq. (6.3), the Hessian $\tilde{\boldsymbol{H}}$ of the batch risk $\tilde{\mathcal{L}}$ is thus corrupted relative to the true Hessian $\boldsymbol{H}$ by a Gaussian likelihood:

[156]: Wills and Schön (2018), 'Stochastic quasi-Newton with adaptive step lengths for large-scale problems'

6: The population risk in Eq. (6.1) is replaced with the stochastic estimate in Eq. (6.2).

$$\begin{aligned} \tilde{\boldsymbol{H}}(\boldsymbol{\theta}) = \boldsymbol{H}(\boldsymbol{\theta}) + \boldsymbol{E} \quad \text{with} \quad &\boldsymbol{E} \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \\ \text{and} \quad &\boldsymbol{\Sigma} = \operatorname{cov}(\nabla\nabla^\top \mathcal{L})/|\mathcal{B}|. \end{aligned}$$

If we now compute matrix-vector products of this batch Hessian with vectors $\boldsymbol{S} \in \mathbb{R}^{D\times M}$, even under the simplifying assumption of Kronecker structure $\boldsymbol{\Sigma} = \boldsymbol{\Lambda} \otimes \boldsymbol{\Lambda}$ in the covariance, the observation likelihood becomes

$$p(\boldsymbol{Y} \mid \boldsymbol{H}, \boldsymbol{S}) = \mathcal{N}(\operatorname{vec}(\boldsymbol{Y}); \operatorname{vec}(\boldsymbol{H}\boldsymbol{S}), \underbrace{(\boldsymbol{S}^\top \boldsymbol{\Lambda}\boldsymbol{S})_{ii} \otimes \boldsymbol{\Lambda}}_{\boldsymbol{R}}). \tag{6.10}$$

The subscript ii in Eq. (6.10) is there to highlight the diagonal structure of the right matrix due to the independent batches we use to calculate the Hessian-vector products. Relative to Eq (6.9), this changes the Gram matrix to be inverted from $\boldsymbol{G}$ to $(\boldsymbol{G} + \boldsymbol{R})$.

To get the posterior of $\boldsymbol{H}$ in the noisy setting after M observations, instead of Eq. (6.9), we now have to solve the linear problem

$$\underbrace{(\boldsymbol{G} + \boldsymbol{R})}_{\mathbb{R}^{DM\times DM}} \operatorname{vec}(\boldsymbol{X}) = \operatorname{vec}(\Delta). \tag{6.11}$$

This is a more challenging computation, since the sum of Kronecker products does not generally have an analytic inverse. However, Eq. (6.11) is a so-called matrix pencil problem, which can be efficiently addressed

using a generalized eigendecomposition [56] . That is, by a matrix $\boldsymbol{V}$ and diagonal matrix $\boldsymbol{D} = \text{diag}(\boldsymbol{\lambda})$ such that

[56]: Golub and Van Loan (2012), *Matrix computations* §7.7

$$\boldsymbol{G}\boldsymbol{V} = \boldsymbol{D}\boldsymbol{R}\boldsymbol{V} \qquad \text{with} \qquad \boldsymbol{V}^\top \boldsymbol{R}\boldsymbol{V} = \boldsymbol{I}. \tag{6.12}$$

Eigen-decompositions distribute over a Kronecker product [147]. And this property is inherited by the generalized eigendecomposition, which offers a convenient way to rewrite a matrix in terms of the other. Eq. (6.11) can thus be solved with two generalized eigendecompositions. The left and right parts of the Kronecker products of Eq. (6.11) are written with separate generalized eigendecompositions as

[147]: Van Loan (2000), 'The ubiquitous Kronecker product'

$$\begin{aligned}
(\boldsymbol{S}^\top \boldsymbol{W}\boldsymbol{S})\boldsymbol{V} &= (\boldsymbol{S}^\top \boldsymbol{\Lambda}\boldsymbol{S})_{ii}\boldsymbol{V}\boldsymbol{\Omega}, \qquad & \boldsymbol{V}^\top (\boldsymbol{S}^\top \boldsymbol{\Lambda}\boldsymbol{S})_{ii}\boldsymbol{V} = \boldsymbol{I} \\
\boldsymbol{W}\boldsymbol{U} &= \boldsymbol{\Lambda}\boldsymbol{U}\boldsymbol{D}, & \boldsymbol{U}^\top \boldsymbol{\Lambda}\boldsymbol{U} = \boldsymbol{I}.
\end{aligned}$$

$\boldsymbol{U}, \boldsymbol{D}$ contain the generalized eigenvectors and eigenvalues from the left Kronecker term and $\boldsymbol{V}, \boldsymbol{\Omega}$ are analogous for the right Kronecker term.

$$\begin{aligned}
&\text{vec}(\boldsymbol{\Delta}) = (\boldsymbol{S}^\top \boldsymbol{W}\boldsymbol{S} \otimes \boldsymbol{W} + (\boldsymbol{S}^\top \boldsymbol{\Lambda}\boldsymbol{S})_{ii} \otimes \boldsymbol{\Lambda})\,\text{vec}(\boldsymbol{X}) \\
&= ((\boldsymbol{S}^\top \boldsymbol{\Lambda}\boldsymbol{S})_{ii}\boldsymbol{V} \otimes \boldsymbol{\Lambda}\boldsymbol{U})(\boldsymbol{\Omega} \otimes \boldsymbol{D} + \boldsymbol{I} \otimes \boldsymbol{I})(\boldsymbol{V}^{-1} \otimes \boldsymbol{U}^{-1})\,\text{vec}(\boldsymbol{X}) \\
&= (\boldsymbol{V}^{-T} \otimes \boldsymbol{U}^{-T})(\boldsymbol{\Omega} \otimes \boldsymbol{D} + \boldsymbol{I} \otimes \boldsymbol{I})(\boldsymbol{V}^{-1} \otimes \boldsymbol{U}^{-1})\,\text{vec}(\boldsymbol{X})
\end{aligned}$$

In the first step above, the left matrix is expressed in terms of the right matrix in both terms by means of the generalized eigendecomposition. In remaining step, the conjugacy property in Eq. (6.12) is used to simplify the expression to the inversion of a diagonal matrix and the Kronecker product of the generalized eigenvectors. The solution now becomes

$$\begin{aligned}
\text{vec}(\boldsymbol{X}) &= (\boldsymbol{U} \otimes \boldsymbol{V})\underbrace{(\boldsymbol{D} \otimes \boldsymbol{\Omega} + \boldsymbol{I} \otimes \boldsymbol{I})^{-1}\,\text{vec}(\boldsymbol{U}^\top \Delta \boldsymbol{V})}_{\frac{1}{(D_{jj}\Omega_{ii}+1)} \odot \boldsymbol{U}^\top \Delta \boldsymbol{V} = \boldsymbol{\Psi}_{ji}} \\
&= (\boldsymbol{U} \otimes \boldsymbol{V})\,\text{vec}(\boldsymbol{\Psi}) = \boldsymbol{U}\boldsymbol{\Psi}\boldsymbol{V}^\top
\end{aligned} \tag{6.13}$$

where $\odot$ refers to the Hadamard product[7] of the two matrices. Using this form, we can represent the posterior mean estimate for $\boldsymbol{H}$ with Eq. (6.8) where $\boldsymbol{X}$ is replaced with the solution from Eq. (6.13).

7: The Hadamard product is the element-wise multiplication of matrices, $(\boldsymbol{A}\odot\boldsymbol{B})_{ij} = a_{ij} \cdot b_{ij}$.

$$\boldsymbol{H}_m = \boldsymbol{H}_0 + \underbrace{\boldsymbol{W}\boldsymbol{X}}_{\mathbb{R}^{D\times M}}\,\underbrace{\boldsymbol{S}^\top \boldsymbol{W}}_{\mathbb{R}^{M\times D}}$$

If the prior mean $\boldsymbol{H}_0$ is chosen as a simple matrix (e.g. a scaled identity), then $\boldsymbol{H}_m$ would admit fast $\mathcal{O}(DM)$ multiplication and inversion (using the matrix inversion lemma)[8] due to its low-rank outer-product structure.

8: $(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{V}^\top)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{V}^\top \boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^\top \boldsymbol{A}^{-1}$

6.6 Active Inference

The preceding section constructed an inference algorithm that turns noisy projections of a matrix into a low-rank estimate for the latent matrix. The second ingredient of our proposed algorithm, outlined in this section, is an active policy that chooses non-redundant projection directions to efficiently improve the posterior mean. Algorithm 2 summarizes as pseudo-code.

The structure of this algorithm is motivated, albeit not exactly matched to, that of stationary iterative linear solvers and eigensolvers such as GMRES [127] and CG [67] (and the corresponding eigenvalue-iterations, the Arnoldi process and the Lanczos process. Cf. [56] and [143]). These algorithms can be interpreted as optimization methods that iteratively expand a low-rank approximation to (e.g. in the case of CG / Lanczos) the Hessian of a quadratic problem, then solve the quadratic problem within the span of this approximation. In our algorithm, the exact low-rank approximation is replaced by the posterior mean *estimate* arising from the Bayesian inference routine described in Section 6.4. This leads the algorithm to suppress search directions that are co-linear with those collected in previous iterations, focusing instead on the efficient collection of new information.

[127]: Saad and Schultz (1986), 'GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems'
[67]: Hestenes and Stiefel (1952), 'Methods of conjugate gradients for solving linear systems'
[56]: Golub and Van Loan (2012), *Matrix computations* §10
[143]: Trefethen and Bau III (1997), *Numerical Linear Algebra* §VI

Readers familiar with linear solvers, like CG, will recognize the structural similarity of Algorithm 2 to linear solvers, with two differing aspects. Each iteration constructs a projection direction $\boldsymbol{s}_i$, collects one matrix-vector multiplication, $\boldsymbol{y}_i = \tilde{\boldsymbol{H}}\boldsymbol{s}_i$ and rescales them by a step size β_i (here set to 1 and omitted). A linear solver would update the solution $\boldsymbol{\theta}_t$ and residual $\boldsymbol{r}_i$ using $\boldsymbol{s}_i$, $\boldsymbol{y}_i$ and β_i but we let the algorithm stay at $\boldsymbol{\theta}_0$ and sample new search directions and projections. The core difference to a solver is in line 5: where the classic solvers would perform a Gram-Schmidt step, we instead explicitly perform Gaussian inference on the Hessian B. In the noise-free limit the proposed method would choose the same search directions as projection methods, a superclass of iterative solvers containing algorithms such as GMRES and CG [64].

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

6.7 Algorithmic Details

Like all numerical methods, the proposed algorithm requires the tuning of some internal parameters and implementational engineering. All free parameters are determined in an empirical fashion, via cheap pre-computations, and use standard linear-algebra tools for the internals.

Estimating Parameters

The Gaussian prior (6.7) and likelihood (6.10) (also used as parameters of Alg. 2) have several parameters. For simplicity and to limit computational cost, we set all these to scaled identity matrices: prior mean $\boldsymbol{H}_0 = h_0\boldsymbol{I}$, variance $\boldsymbol{W} = w_0\boldsymbol{I}$ and noise covariance $\boldsymbol{\Lambda} = \lambda_0\boldsymbol{I}$. These parameters are determined empirically: before the method tries to estimate the Hessian, it gathers gradients and Hessian-gradient products locally, from a small number of initial batches. The prior of $\boldsymbol{H}$ is then defined by the parameters

$$1/h_0 = \sqrt{\frac{\boldsymbol{s}^\top \boldsymbol{H}\boldsymbol{s}}{\boldsymbol{s}^\top \boldsymbol{H}\boldsymbol{H}\boldsymbol{s}}} \qquad \text{and} \qquad w_0 = \frac{\boldsymbol{s}^\top \boldsymbol{H}\boldsymbol{s}}{\boldsymbol{s}^\top \boldsymbol{s}}. \tag{6.14}$$

The noise variance for the likelihood is set by an empirical estimate

$$\lambda_0 = (\mathbb{E}[g^2] - (\underbrace{\mathbb{E}[g]}_{\bar{g}})^2)/\sqrt{\boldsymbol{s}^\top \boldsymbol{s}}. \tag{6.15}$$

Since the batch elements can exhibit a high variance, a more robust choice is used by setting λ_0 to the median of the variance estimates. The mean gradient ($\bar{g}$) from the initial sampling is used for the first iteration of the Hessian-inference scheme, line 2. A new search direction along which a Hessian-vector product is to be computed, is obtained by applying the inverse of the current estimated Hessian to a stochastic gradient:

A minor computational saving by reusing information obtained by estimating λ_0 in Eq. (6.15).

$$s_{i+1} = -\hat{H}_i^{-1}\nabla\tilde{\mathcal{L}}(\theta). \tag{6.16}$$

The estimated Hessian is updated by first solving Eq. (6.11) and using the result in Eq. (6.8) to get the posterior mean: It is a sum of the prior mean and a low-rank outer product of two $D \times M$ matrices, with D the number of parameters and M the number of observations/iterations. A diagonal prior mean offers efficient inverse multiplication of the estimated Hessian[9] by the matrix inversion lemma.

9: required to determine a search direction in Eq. (6.16)

In the experiments below, the algorithm is used to construct a preconditioner for stochastic optimization algorithms. For this application, spends some time at the beginning of an optimization process, collecting several batches "in-place", each time computing a noisy matrix-Hessian product. To find the dominant k eigendirections the solver usually requires a number $m > k$ of iterations, producing a posterior estimate H_m of rank *identity* $+ m$. To reduce the computational cost and memory requirements in the subsequent actual optimization run, we reduce the rank of this approximation down to k using a standard fast singular value decomposition [10] and obtain the singular values Σ and the left singular vectors U. The approximate Hessian is then constructed as $H \approx U\Sigma U^\top$. This is equivalent to taking the symmetric part of a polar decomposition which yields the closest symmetric approximation to a matrix in Frobenius norm [68]. For a D-dimensional problem and a preconditioner of rank k, this method requires the storage of $\mathcal{O}(Dk)$ numbers and has a computational overhead of $\mathcal{O}(Dk)$ additional FLOPS compared to a standard SGD update. Such overhead is negligible for batch-sizes $|\mathcal{B}| > k$, the typical case in machine learning.

Particularly true if there is a significant amount of noise.

10: Available in `scipy.sparse.linalg`.

[68]: Higham (1988), 'Computing a nearest symmetric positive semidefinite matrix'

Algorithm 2: Active probabilistic matrix inference for linear problems of the form $Hd = b$, where multiplications with the matrix H can only be performed corrupted by noise.

Input : θ_0: Point where curvature is estimated,
$H(\cdot)$: Handle for Hvp,
$\nabla\mathcal{L}_{\mathcal{B}}(\cdot)$: Handle for gradient,
m: Number of iterations,
$p(H)$: Prior distribution,
$p(Y \mid H, S)$: Observation model

Output: $H_m = H_0 + WXS^\top W$; // a low rank matrix

1 **for** $i = 1 \ldots m$ **do**
2 $r_i = -\nabla\mathcal{L}_{\mathcal{B}}(\theta_0)$; // Noisy gradient
3 $s_i = H_{i-1}^{-1} r_{i-1}$; // Step direction
4 $y_i = H(s_i)$; // Observe Hvp on same batch as gradient
5 $H_i =$ `inferMatrix` $(p(Y \mid H, S), p(H), S, Y)$; // H in Sec. 6.5

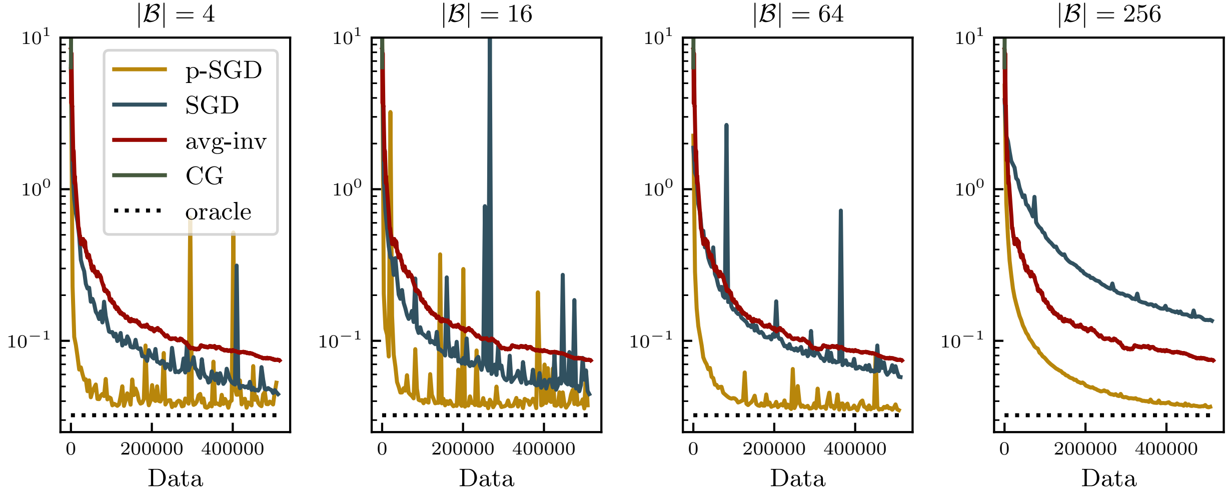


Figure 6.1: Comparison of SGD and preconditioned SGD on the linear test problem, along with other baselines. Details in text. The plots show data for four different choices of batch size $|\mathcal{B}|$ (and thus varying observation noise). For fairness to SGD, the abscissa is scaled in the number of data points loaded from disk (as opposed to the number of steps). Due to the noisy setting, vanilla CG is unstable and diverges in the first few steps.

Preconditioning

A preconditioner $\boldsymbol{P}$ is constructed to rescale the stochastic gradients in the direction of the singular vectors.

$$\boldsymbol{P}^{-1} = \boldsymbol{I} + \boldsymbol{U}[\beta \boldsymbol{I}_k / \sqrt{\boldsymbol{\Sigma}} - \boldsymbol{I}_k]\boldsymbol{U}^{\top} \tag{6.17}$$

See the introduction in Sec. ?? for more details. Note that in Sec. 4.5 $\boldsymbol{P}$ denotes an approximation to $\boldsymbol{H}$ whereas here it is an approximation to $\boldsymbol{H}^{-1}$.

By rescaling the gradients with $\boldsymbol{P}^{-1}\boldsymbol{P}^{-\top} = \boldsymbol{P}^{-2}$, the linear system in Eq. (6.5) is transformed into $\boldsymbol{P}^{-\top}\tilde{\boldsymbol{H}}(\boldsymbol{\theta}_t)\boldsymbol{P}^{-1}\boldsymbol{P}\boldsymbol{d}_t = \boldsymbol{P}^{-\top}\nabla\tilde{\mathcal{L}}(\boldsymbol{\theta}_t)$. The goal, as outlined above, is to reduce the condition number of the transformed Hessian $\boldsymbol{P}^{-\top}\boldsymbol{H}(\boldsymbol{\theta}_t)\boldsymbol{P}^{-1}$. If the estimated vectors $\boldsymbol{U}, \boldsymbol{\Sigma}$ are indeed the real eigenvectors and eigenvalues, this approach would rescale these directions to have length β. Theoretically $\beta = 1$ would be ideal if the real eigen-pairs are used. When instead an approximation of the subspace is used with poor approximations of the eigenvalues $\tilde{\lambda}_i$, it is possible to scale a direction too much so the eigenvectors corresponding to the largest eigenvalues become the new smallest eigenvalues. In our experiments this rarely happened because the Hessian contained many eigenvalues $\lambda_i \ll 1$ and so $\beta = 1$ could be used. The improved condition number allows a larger step-size to be used. The step-size is scaled by the empirical improvement of the condition number, i.e. the fixed step-size of SGD η is multiplied with $\eta^2 = \boldsymbol{\Sigma}_1/\boldsymbol{\Sigma}_k$ (in Eq. (6.17)), the ratio between largest and the smallest estimated eigenvalue. In the current notation a preconditioned SGD update can be written

$$\boldsymbol{\theta}_{+1} = \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{P}^{-2}\nabla\tilde{\mathcal{L}}(\boldsymbol{\theta}_t).$$

6.8 High-Dimensional Modification

Deep learning has become the benchmark for stochastic optimization algorithms and it imposes several new constraints on the algorithms. Due to the large number of parameters, even if access to the fraction

of eigendirections which make the Hessian ill-conditioned is available, it would likely be inefficient to use because each vector has the same size as the network, and the approach would only work if a few of the eigenvalues of the Hessian have a clear separation in magnitude. This changes the functionality we can expect from a preconditioner to keeping the parameter updates relevant with respect to the current condition number rather than finding all directions with problematic curvature.

Some simplifications are required in order to adapt the algorithm for deep learning. The most important change is that we approximate the Hessian as a layer-wise block matrix, effectively treating each layer of a deep net as an independent task. Hessian-vector products are calculated using automatic differentiation [114]. It was difficult to get good estimates of the eigenvalues for the algorithm because of large deviations, likely due to the noisy Hessian-vector products. To alleviate this problem we changed the multiplicative update of the step-size presented in section 5 to redefining the step-length. Each time the algorithm is used to build a preconditioner, a new step-length is set to the scalar prior mean in Eq. (6.14). The last modification is how the empirical parameters of the prior (section 6.7) are set. $1/h_0$ is used as the new step-length and gave better results when a smaller step-size of $\boldsymbol{s}^\top \boldsymbol{H}\boldsymbol{s}/\boldsymbol{s}^\top \boldsymbol{H}\boldsymbol{H}\boldsymbol{s}$ was used and λ_0 is estimated with $\lambda_0 = \sqrt{(\sum[g^\top g] - \hat{g}^\top \hat{g})/m}$, with $\hat{\boldsymbol{g}} = \sum_i^m \boldsymbol{g}_{\mathcal{B}_i}$. All the parameters of the prior and likelihood (h_0, w_0 and λ_0) are shared among the layers. No clear improvement was visible when treating the parameters separately for each layer.

[114]: Pearlmutter (1994), 'Fast exact multiplication by the Hessian'

$\hat{g}$ is the accumulated gradient obtained from collecting gradients and Hessian-vector products before estimating the preconditioner.

6.9 Experiments

The performance of the proposed algorithm is evaluated on a simple test problem along with two standard applications in machine learning.

Regression

Figure 6.1 shows the results from a conceptual test setup designed to showcase the algorithm's potential: an ill-conditioned linear problem of a scale chosen such that the analytical solution can still be found for comparison. Linear parametric least-squares regression on the SARCOS dataset [152] is used as a test setup. The dataset contains $|\mathcal{D}| = 44,484$ observations of a uni-variate response function $y_i = f(\boldsymbol{x}_i)$ in a 21-dimensional space $\boldsymbol{x}_i \in \mathbb{R}^{21}$. We used the polynomial feature functions $\phi(\boldsymbol{x}) = \boldsymbol{A}[\boldsymbol{x}, \text{vec}(\boldsymbol{x}\boldsymbol{x}^\top)] \in \mathbb{R}^{253}$, with a linear mapping, $\boldsymbol{A}$, manually designed to make the problem ill-conditioned. The model $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{\theta}$ with a quadratic loss function then yields a quadratic optimization problem,

[152]: Vijayakumar and Schaal (2000), 'Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space'

The dataset contains 7 such univariate target variables. Following convention, the first target dimension is used.

$$\begin{aligned}
\boldsymbol{\theta}_* &= \arg\min_{\boldsymbol{\theta}} \alpha\|\boldsymbol{\theta}\|^2 + \frac{1}{|\mathcal{D}|}\|\boldsymbol{\phi}^\top \boldsymbol{\theta} - \boldsymbol{y}\|^2 \\
&= \alpha\|\boldsymbol{\theta}\|^2 + \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}(\phi(\boldsymbol{x}_i)^\top \boldsymbol{\theta} - y_i)^2,
\end{aligned} \tag{6.18}$$

where $\boldsymbol{\phi} \in \mathbb{R}^{253\times 44,484}$ is the map from weights to data. The exact solution of this problem is given by the regularized least-squares estimate

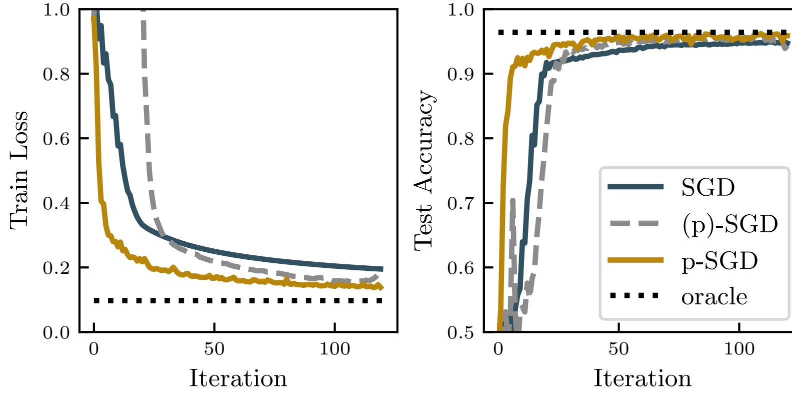


Figure 6.2: Progress of preconditioned and standard SGD on the MNIST logistic regression problem (details in text). Left plot shows progress on the raw optimization objective (train loss), the right plot shows generalization in terms of test accuracy. Two lines are shown for SGD ((p)-SGD is the second). The solid one uses a step-size η_{opt} optimized for good performance. This directly induces a step-size $\eta_{\text{prec.}}$ for the preconditioner. For comparison, we also show progress of SGD when directly using this step size—it makes SGD unstable.

$\boldsymbol{\theta}_* = (\boldsymbol{\phi}\boldsymbol{\phi}^\top/|\mathcal{D}| + \alpha \boldsymbol{I})^{-1}\boldsymbol{\phi}\boldsymbol{y}/|\mathcal{D}|$. For this problem size, the solution can be computed easily, providing an oracle baseline for our experiments. But if the number of features were higher (e.g. $\gtrsim 10^4$), then exact solutions would not be tractable. One could instead compute, as in deep learning, batch gradients from Eq. (6.18), and also produce a noisy approximation of the Hessian $\boldsymbol{H} = (\boldsymbol{\phi}\boldsymbol{\phi}^\top/|\mathcal{D}| + \alpha \boldsymbol{I})$ as the low rank matrix

$$\tilde{\boldsymbol{H}} = \alpha \boldsymbol{I} + \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \phi(x_b)\phi(x_b)^\top \tag{6.19}$$

where $\mathcal{B}$ is a batch. Clearly, multiplying an arbitrary vector with this low-rank matrix has cost $\mathcal{O}(|\mathcal{B}|)$, thus providing the functionality required for our noisy solver. Figure 6.1 compares the progress of vanilla SGD with that of preconditioned SGD if the preconditioner is constructed with the proposed algorithm. In each case, the construction of the preconditioner was performed in 16 iterations of the inference algorithm. Even in the largest case of $\mathcal{B} = 256$, this amounts to 4096 data read, and thus only a minuscule fraction of the overall runtime.

An alternative approach would be to compute the inverse of $\tilde{\boldsymbol{H}}$ separately for each batch, then average over the batch-least-squares estimate $\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{H}}^{-1}\boldsymbol{\phi}\boldsymbol{y}/|\mathcal{B}|$. In our toy setup, this can again be done directly in the feature space. In an application with larger feature space, this is still feasible using the matrix inversion lemma on Eq. (6.19), instead inverting a dense matrix of size $|\mathcal{B}| \times |\mathcal{B}|$. The Figure also shows the progression of this stochastic estimate labeled as *avg-inv*. It performs much worse unless the batch-size is increased, which highlights the advantage of the active selection of projection directions for identifying appropriate eigenvectors. A third option is to use an iterative solver with noise-corrupted observations to approach the optimum. In figure 6.1 a barely visible line labeled CG can be seen which used the method of conjugate gradients with a batch-size of 256. This method required a batch-size $|\mathcal{B}| > 10000$ to show reasonable convergence on the training objective but would still perform poorly on the test set.

The avg-inv estimator used $|\mathcal{B}| = 256$ for all experiments because smaller batch-sizes consistently diverged.

Logistic Regression

Figure 6.2 shows an analogous experiment on a more realistic, and *non-linear* problem: Classic linear logistic regression on the digits 3 and 5 from the MNIST dataset (i.e. using linear features $\phi(\boldsymbol{x}) = \boldsymbol{x}$, and $p(y \mid \boldsymbol{x}) =$

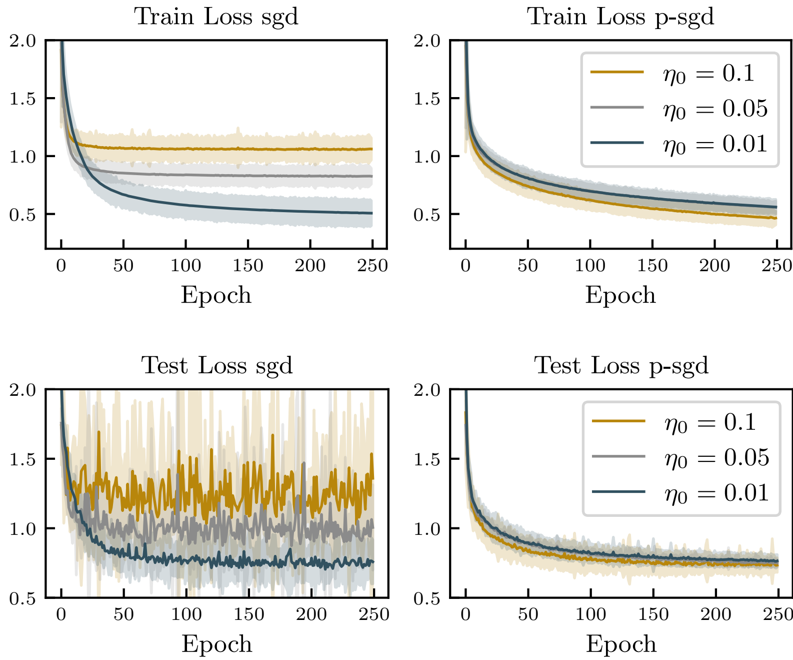


Figure 6.3: Training loss for SGD (left) and preconditioned SGD (right) on the CIFAR-10 dataset for different learning rates and batch-size of 32 over 250 epochs. Both graphs share y-axis and colors to facilitate comparison between the optimizers. The solid lines represent the mean of several individual runs plotted as translucent.

Figure 6.4: Test loss for SGD (left) and preconditioned SGD (right) on the CIFAR-10 dataset for different learning rates and batch-size of 32 over 250 epochs. Both graphs share y-axis and colors to facilitate comparison between the optimizers. These graphs were collected at the same runs as the results in Fig. 6.3.

$\sigma(\phi(\boldsymbol{x})^\top \boldsymbol{\theta})$). Here we used the model proposed in [120] , which defines a convex, non-linear regularized empirical risk minimization problem that again allows the construction of stochastic gradients, and associated noisy Hessian-vector products. Analogous to Figure 6.1, Figure 6.2 shows progress of SGD and preconditioned SGD. As before, this problem is actually just small enough to compute an exact solution by Newton optimization (── in Fig. 6.2). And as before, computation of the preconditioner takes up a small fraction of the optimization runtime.

[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning* §3.4

Deep Learning

For a high-dimensional test bed, a deep net was used that consists of convolutional and fully-connected layers to classify the CIFAR-10 dataset [85]. The architecture consisted of 3 convolutional layers with 64, 96 and 128 output channels of size 5×5, 3×3 and 3×3 followed by 3 fully connected layers of size 512, 256, 10 with cross entropy loss function on the output and l_2-regularization with magnitude 0.01. All layers used the ReLU nonlinearity and the convolutional layers had additional max-pooling.

[85]: Krizhevsky (2009), *Learning multiple layers of features from tiny images*

The proposed optimization algorithm was implemented in PyTorch [112], using the modifications listed in section 6.8. To stabilize the algorithm, a predetermined fixed learning-rate was used for the first epoch of the preconditioned SGD. Figure 6.3 and 6.4 compare the convergence for the proposed algorithm against SGD for training loss and test loss respectively on CIFAR-10.

[112]: Paszke et al. (2017), 'Automatic differentiation in PyTorch'

Both figures show that the preconditioned SGD has similar performance to a well-tuned SGD regardless of the initial learning rate. A rank 2 approximation of the Hessian was used to keep the cost of computations and memory low. This approximation was recalculated at the beginning of every epoch to have it adapt to alternating curvature. The cost of

building the rank 2 approximation was 2–5% of the total computational cost per epoch.

The improved convergence of preconditioned SGD over standard SGD is mainly attributed the adaptive step-size, which seems to capture the general scale of the curvature to efficiently make progress. This approach offers a more rigorous way to update the step-length over popular empirical approaches using decaying learning-rates or division upon plateauing [57] . Inspecting the scale of the found learning rate, see Fig. 6.5, shows that different initial learning rates tend follow the same trajectory although spanning values across four orders of magnitude for this task.

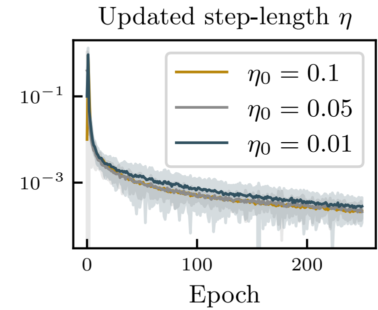


Figure 6.5: Evolution of the estimated learning rate η over 250 epochs for different initial values.

[57]: Goodfellow et al. (2016), *Deep Learning* §8

6.10 Conclusion

This chapter presented an active probabilistic inference algorithm to efficiently construct preconditioners in stochastic optimization problems. It consists of three conceptual ingredients: First, a matrix-valued Gaussian inference scheme that can deal with structuredăGaussian noise in observed matrix-vector products. Second, an active evaluation scheme aiming to collect informative, non-redundant projections. Third, additional statistical and linear algebra machinery to empirically estimate hyper-parameters and arrive at a low-rank preconditioner. The resulting algorithm was shown to significantly improve the behavior of SGD in typical test problems, even in the case of severe observation noise typical for contemporary machine learning problems. It scales from low- to medium- and high-dimensional problems, where its behavior qualitatively adapts from full and stable preconditioning to low-rank preconditioning and, eventually, scalar adaptation of the learning rate of SGD-type optimization.

6.11 Future Directions

The theory developed in Sec. 6.4 can on a high level be interpreted as a stochastic version of a singular value decomposition. The inference procedure approximates the subspace spanned by the largest eigenvectors of the Hessian, which then are used to precondition the updates of a first-order optimization algorithm. One could instead use the matrix estimate for more general subspace analysis and dimensionality reduction techniques.

A clear direction for future research lies in the prior knowledge that goes into the modeling of the matrix. For the Hessian it would be beneficial to include information about symmetry. One possible approach can be derived based on theory in the upcoming Ch. 7, but there is no conclusive answer yet regarding an efficient inference scheme.

See Sec. 4.6 for a discussion.

Another way to approach the problem of symmetric matrix inference with noise is to look at it as a constrained optimization problem.

Constrained Optimization

The following derivations are based on the work presented by Wills and Schön [157] who approached the problem of finding a symmetric QN-update (the posterior mean) in the noise-free case as a constrained optimization problem:

[157]: Wills and Schön (2019), 'Stochastic quasi-Newton with line-search regularization'

$$\begin{aligned} \max_{\boldsymbol{H}} \quad & (\operatorname{vec}(\boldsymbol{H}) - \operatorname{vec}(\boldsymbol{H}_0))^\top (\boldsymbol{W}^{-1} \otimes \boldsymbol{W}^{-1})(\operatorname{vec}(\boldsymbol{H}) - \operatorname{vec}(\boldsymbol{H}_0)) \\ \text{s.t.} \quad & \boldsymbol{\Gamma} \operatorname{vec}(\boldsymbol{H}) = \boldsymbol{0} \\ \text{and} \quad & (\boldsymbol{s} \otimes \boldsymbol{I})^\top \operatorname{vec}(\boldsymbol{H}) = \boldsymbol{y}, \end{aligned}$$

where $\boldsymbol{\Gamma}$ is the antisymmetric projection operator. A noisy optimization problem can be derived from this setup by replacing the last constraint with another quadratic term in the optimization.

$\boldsymbol{\Gamma} \operatorname{vec}(\boldsymbol{H}) = \operatorname{vec}(\boldsymbol{H}) - \operatorname{vec}(\boldsymbol{H}^\top)$

The task is to solve the following optimization problem:

$$\begin{aligned} \max_{\boldsymbol{H}} \quad & (\operatorname{vec}(\boldsymbol{H}) - \operatorname{vec}(\boldsymbol{H}_0))^\top (\boldsymbol{W}^{-1} \otimes \boldsymbol{W}^{-1})(\operatorname{vec}(\boldsymbol{H}) - \operatorname{vec}(\boldsymbol{H}_0)) + ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}||_{\Xi^{-1}} \\ \text{s.t.} \quad & \boldsymbol{\Gamma} \operatorname{vec}(\boldsymbol{H}) = 0. \end{aligned}$$

Solving this problem is equivalent to finding the posterior mean with a symmetric prior and likelihood $p(\boldsymbol{y} \mid \boldsymbol{H}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{H}\boldsymbol{s}, \Xi)$. To solve it we define the following parameters:

$$\begin{aligned} \boldsymbol{h} &\equiv \operatorname{vec}(\boldsymbol{H}) \\ \boldsymbol{h}_0 &\equiv \operatorname{vec}(\boldsymbol{H}_0) \\ \boldsymbol{V}^{-1} &\equiv \boldsymbol{W}^{-1} \otimes \boldsymbol{W}^{-1} \\ \boldsymbol{S} &\equiv \boldsymbol{s} \otimes \boldsymbol{I} \end{aligned}$$

and use the Lagrangian

$$\mathcal{L}(\boldsymbol{h}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{h}^\top \boldsymbol{V}^{-1}\boldsymbol{h} - \boldsymbol{h}^\top \boldsymbol{V}^{-1}\boldsymbol{h}_0 + \frac{1}{2}(\boldsymbol{h}^\top \boldsymbol{S})\Xi^{-1}(\boldsymbol{S}^\top \boldsymbol{h}) - \frac{1}{2}(\boldsymbol{h}^\top \boldsymbol{S})\Xi^{-1}\boldsymbol{y} - \boldsymbol{\lambda}^\top(\boldsymbol{\Gamma}\boldsymbol{h}). \tag{6.20}$$

Constant terms have been omitted. Now looking at the differential w.r.t. $\boldsymbol{h}$

$$\nabla_{\boldsymbol{h}}\mathcal{L} = \boldsymbol{V}^{-1}\boldsymbol{h} - \boldsymbol{V}^{-1}\boldsymbol{h}_0 + \boldsymbol{S}\Xi^{-1}(\boldsymbol{S}^\top \boldsymbol{h}) - \boldsymbol{S}\Xi^{-1}\boldsymbol{y} - \boldsymbol{\Gamma}^\top \boldsymbol{\lambda} = 0 \tag{6.21}$$

implies with unvectorization

$$\begin{aligned} \boldsymbol{V}^{-1}(\boldsymbol{h} - \boldsymbol{h}_0) + \boldsymbol{S}\Xi^{-1}((\boldsymbol{S}^\top \boldsymbol{h}) - \boldsymbol{y}) &= \boldsymbol{\Gamma}^\top \boldsymbol{\lambda} = \operatorname{vec}(\boldsymbol{\Lambda} - \boldsymbol{\Lambda}^\top) \\ \boldsymbol{W}^{-1}(\boldsymbol{H} - \boldsymbol{H}_0)\boldsymbol{W}^{-1} + \Xi^{-1}(\boldsymbol{H}\boldsymbol{s} - \boldsymbol{y})\boldsymbol{s}^\top &= \boldsymbol{\Lambda} - \boldsymbol{\Lambda}^\top. \end{aligned}$$

Now adding the transpose of $(\boldsymbol{\Lambda} - \boldsymbol{\Lambda}^\top)^\top$ results in

$$2\boldsymbol{W}^{-1}(\boldsymbol{H} - \boldsymbol{H}_0)\boldsymbol{W}^{-1} + \Xi^{-1}(\boldsymbol{H}\boldsymbol{s} - \boldsymbol{y})\boldsymbol{s}^\top + \boldsymbol{s}(\boldsymbol{H}\boldsymbol{s} - \boldsymbol{y})^\top \Xi^{-1} = 0, \tag{6.22}$$

or equivalently

$$2\boldsymbol{W}^{-1}\boldsymbol{H}\boldsymbol{W}^{-1} + \Xi^{-1}\boldsymbol{H}\boldsymbol{s}\boldsymbol{s}^\top + \boldsymbol{s}\boldsymbol{s}^\top \boldsymbol{H}\Xi^{-1} = 2\boldsymbol{W}^{-1}\boldsymbol{H}_0\boldsymbol{W}^{-1} + \Xi^{-1}\boldsymbol{y}\boldsymbol{s}^\top + \boldsymbol{s}\boldsymbol{y}^\top \Xi^{-1}. \tag{6.23}$$

Finding the $\boldsymbol{H}$ that satisfies this expression would provide the sought-after posterior mean. No general solution is available at this time but a special case could provide valuable insight.

Or a symmetric QN-update with modeled noise.

Special Case

For the special case where $\Xi = \beta^{-1}\boldsymbol{W}$ in Eq. (6.23) it is possible to progress further. From a probabilistic point of view it is difficult to justify that the noise would have a similar covariance to the prior. From an optimization point of view, especially if $\boldsymbol{W} = \boldsymbol{H}_{true}$ as is often the case , it could make sense to perform the minimization w.r.t. to this norm. Equation (6.23) can now be rewritten into

Beyond computational benefits.

See Sec. 4.6

$$(\boldsymbol{W}^{-1}+\beta\boldsymbol{s}\boldsymbol{s}^\top)\boldsymbol{H}\boldsymbol{W}^{-1}+\boldsymbol{W}^{-1}\boldsymbol{H}(\boldsymbol{W}^{-1}+\beta\boldsymbol{s}\boldsymbol{s}^\top) = 2\boldsymbol{W}^{-1}\boldsymbol{H}_0\boldsymbol{W}^{-1}+\beta\boldsymbol{W}^{-1}\boldsymbol{y}\boldsymbol{s}^\top+\beta\boldsymbol{s}\boldsymbol{y}^\top\boldsymbol{W}^{-1}.$$

Now multiplying with $-\boldsymbol{W}$ from the left and the right results in

$$\underbrace{(-(\boldsymbol{I}+\beta\boldsymbol{W}\boldsymbol{s}\boldsymbol{s}^\top))}_{A}\boldsymbol{H}+\boldsymbol{H}\underbrace{(-(\boldsymbol{I}+\boldsymbol{s}\boldsymbol{s}^\top\boldsymbol{W}\beta))}_{A^\top} = -\underbrace{\left(2\boldsymbol{H}_0+\boldsymbol{y}\boldsymbol{s}^\top\boldsymbol{W}\beta+\beta\boldsymbol{W}\boldsymbol{s}\boldsymbol{y}^\top\right)}_{Q}. \tag{6.24}$$

This is now a continuous Lyapunov equation in the standard form

$$\boldsymbol{A}\boldsymbol{H}+\boldsymbol{H}\boldsymbol{A}^\top = -\boldsymbol{Q}, \tag{6.25}$$

with several important properties [53, 80] The following definition will prove useful for later derivations.

[53]: Glad and Ljung (2014), *Control theory* §5
[80]: Khalil (2002), *Nonlinear systems* §12

$$\boldsymbol{A} = -(\boldsymbol{I}+\beta\boldsymbol{W}\boldsymbol{s}\boldsymbol{s}^\top) = -(\boldsymbol{I}+\boldsymbol{u}\boldsymbol{s}^\top)$$

Lemma 6.11.1 *Lemma 12.1 of [53]*
If $\boldsymbol{Q}$ is spd (psd) and $\boldsymbol{A}$ has eigenvalues with $\mathrm{Re}(\lambda_i) < 0$.
Then the solution to Eq. (6.25) $\boldsymbol{H}$ is spd (psd).

Theorem 6.11.2 *Theorem 5.3 of [53]*
If the eigenvalues of $\boldsymbol{A}$ have $\mathrm{Re}(\lambda_i) < 0$ and $\boldsymbol{Q}$ is spd (psd). Then the solution to Eq. (6.25) is

$$\boldsymbol{H} = \int_0^\infty e^{(\boldsymbol{A}t)}\boldsymbol{Q}e^{(\boldsymbol{A}^\top t)}dt \tag{6.26}$$

Lemma 6.11.3 *If $\boldsymbol{A} = -(\boldsymbol{I}+\boldsymbol{u}\boldsymbol{s}^\top)$ has eigenvalues with negative real part.*

$$\exp(\boldsymbol{A}t) = \exp(-t)\left(\boldsymbol{I}+\left(\frac{e^{-\alpha t}-1}{\alpha}\right)\boldsymbol{u}\boldsymbol{s}^\top\right) \tag{6.27}$$
$$\alpha = \boldsymbol{s}^\top\boldsymbol{u}$$

Proof.

$$
\begin{aligned}
\exp(\boldsymbol{A}t) &= \exp(-(\boldsymbol{I}+\boldsymbol{u}\boldsymbol{s}^\top)t) = [\boldsymbol{I} \text{ and } \boldsymbol{u}\boldsymbol{s}^\top \text{ commute}] = \\
&= \exp(-\boldsymbol{I}t)\exp(-\boldsymbol{u}\boldsymbol{s}^\top t) = e^{-t}\boldsymbol{I}\left(\sum_{k=0}^{\infty}\frac{(\boldsymbol{u}\boldsymbol{s}^\top)^k(-t)^k}{k!}\right) = \\
&= e^{-t}\left(\boldsymbol{I} - \boldsymbol{u}\boldsymbol{s}^\top t + \frac{\boldsymbol{u}\alpha\boldsymbol{s}^\top t^2}{2} - \frac{\boldsymbol{u}(\alpha)^2\boldsymbol{s}^\top t^3}{3!} + \dots\right) = \\
&= e^{-t}\left(\boldsymbol{I} - \left(\sum_{k=1}^{\infty}\frac{\alpha^{k-1}(-t)^k}{k!}\right)\boldsymbol{u}\boldsymbol{s}^\top\right) = \\
&= e^{-t}\left(\boldsymbol{I} + \left(\frac{e^{-\alpha t}-1}{\alpha}\right)\boldsymbol{u}\boldsymbol{s}^\top\right)
\end{aligned}
$$

□

Lemma 6.11.4 *If* $\boldsymbol{A}$ *follow Lemma 6.11.3 then Eq.* (6.26) *which solves Eq.* (6.25) *has a closed-form solution:*

$$
\boldsymbol{H} = \int_0^{\infty} e^{(\boldsymbol{A}t)}\boldsymbol{Q}e^{(\boldsymbol{A}^\top t)}dt = \dots = \frac{1}{2}\left(\boldsymbol{Q} - \frac{(\boldsymbol{u}\boldsymbol{s}^\top\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{s}\boldsymbol{u}^\top)}{\alpha+2} + \frac{\boldsymbol{u}\boldsymbol{s}^\top\boldsymbol{Q}\boldsymbol{s}\boldsymbol{u}^\top}{(\alpha+1)(\alpha+2)}\right) \tag{6.28}
$$

According to Lemma 6.11.1 $\boldsymbol{H}$ *will also be spd if* $\boldsymbol{Q}$ *is spd.*

Inserting $\boldsymbol{Q}$ from Eq. (6.24) into Eq. (6.28) now gives the solution to the optimization problem in Eq. (6.23) *i.e.* the posterior mean for the special case of $\boldsymbol{\Xi} = \beta\boldsymbol{W}$.

$$
\boldsymbol{H}_1 = \boldsymbol{H}_0 + \frac{1}{\alpha+2}\left((\boldsymbol{y}-\boldsymbol{H}_0\boldsymbol{s})\boldsymbol{s}^\top\boldsymbol{W} + \boldsymbol{W}\boldsymbol{s}(\boldsymbol{y}-\boldsymbol{H}_0\boldsymbol{s})^\top - \boldsymbol{W}\boldsymbol{s}\left(\frac{\boldsymbol{y}^\top\boldsymbol{s} - \boldsymbol{s}^\top\boldsymbol{H}_0\boldsymbol{s}}{\alpha+1}\right)\boldsymbol{s}^\top\boldsymbol{W}\right) \tag{6.29}
$$

A QN-update along this line could provide an iterative update that allows the posterior to remain spd by tuning the likelihood parameter β while still keeping the computational cost low. Since $\boldsymbol{W}$ always arise as the product $\boldsymbol{W}\boldsymbol{s}$ it would also be possible to employ the multiplication implicitly which results in various popular optimization routines [64].

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

Chapter 7

Nonparametric Curvature Estimation

Nonparametric models can improve the interpolation of regression tasks over parametric counterparts due to increased flexibility. It is therefore an interesting direction of research to extend the traditional parametric curvature estimation used until now to a nonparametric setting. One way of doing it is with kernel methods such as Gaussian processes (GPs). Extending the curvature estimate to nonparametric models is straightforward in theory but computationally challenging in practice when done naïvely. The overall goal of this chapter is to provide an efficient inference scheme to allow nonparametric curvature estimates from gradient observations in Gaussian processes. Contrary to the previous chapters which focused on the stochastic optimization, we will here restrict the exposition to deterministic optimization. Due to the probabilistic nature of Gaussian processes it would be straightforward to extend the relevant derivations to the stochastic regime. To not confuse readers familiar with the literature on Gaussian processes we will switch the notation to use $\boldsymbol{x} \in \mathbb{R}^D$ as input parameter to a scalar function f that we want to minimize.

The manuscript was originally published in [33].

[33]: de Roos et al. (2021), 'High-Dimensional Gaussian Process Inference with Derivatives'

See Ch. 3 for relevant background.

The input $\boldsymbol{x}$ is used instead of the previously used parameter $\boldsymbol{\theta}$.

7.1 Introduction

The closure of Gaussian processes (GPs) under linear operations is well-established in the literature [120] . Given a Gaussian process $f \sim \mathcal{GP}(\mu, k)$, with mean and covariance function μ and k, respectively, a linear operator $\mathcal{L}$ acting on f induces another Gaussian process $\mathcal{L}f \sim \mathcal{GP}(\mathcal{L}\mu, \mathcal{L}k\mathcal{L}')$ for the operator $\mathcal{L}$ and its adjoint $\mathcal{L}'$. The linearity of GPs has found extensive use both for conditioning on projected data, and to perform inference on linear transformations of f. Differentiation is a linear operation which has caused considerable interest in GP modeling due to the wide variety of applications in which derivative observations are available. However, each gradient observation of $\nabla f \in \mathbb{R}^D$ induces a block Gram matrix $\nabla k \nabla' \in \mathbb{R}^{D \times D}$. As dimension D and number of observations N grow, inference with gradient information quickly becomes prohibitive with the naïve computational scaling of $\mathcal{O}(N^3 D^3)$. In other words, one gradient observation comes at the same computational cost as D independent function evaluations and thus becomes increasingly disadvantageous as dimensionality grows. This is not surprising, as the gradient contains D elements and thus bears information about every coordinate. The unfavorable scaling has confined GP inference with derivatives to low-dimensional settings in which the information gained from gradients outweighs the computational overhead.

[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning* §9.4

See Sec. 7.2 for an overview.

This chapter will explore the structure of the Gram matrix for gradients and show that it enables inversion at cost linear in D. Such computational scaling unlocks the previously prohibitive use of gradient evaluations in high dimensional spaces for nonparametric models. Numerous machine learning algorithms that operate on high-dimensional spaces are guided by gradient information and bear the potential to benefit from an inference mechanism that avoids discarding readily available information. Examples for such applications that we here consider comprise optimization and linear algebra.

The original article also explored gradient-based hybrid Monte-Carlo methods which has here been omitted.

Contributions We analyze the structure of the Gram matrix with derivative observations for stationary and dot product kernels and report the following discoveries:

Code repository: https://github.com/fidero/gp-derivative

- The Gram matrix can be decomposed to allow *exact* inference in $\mathcal{O}(N^2D + (N^2)^3)$ floating point operations, which is useful in the limit of few observations ($N < D$).
- We introduce an efficient approximate inference scheme to include gradient observations in GPs even as the number of high-dimensional observations increases. It relies on exact matrix-vector multiplication (MVM) and an iterative solver to approximately invert the Gram matrix. This implicit MVM avoids constructing the whole Gram matrix and thereby reduces the memory requirements from $\mathcal{O}((ND)^2)$ to $\mathcal{O}(N^2 + ND)$.
- We demonstrate the applicability of the improved scaling in the low-data regime for high-dimensional optimization.
- We explore a special case of inference with application to probabilistic linear algebra for which the cost of inference can be further reduced to $\mathcal{O}(N^2D + N^3)$.

7.2 Related Work

Exact derivative observations have previously been used to condition GPs on linearizations of dynamic systems [138] as a way to condense information in dense input regions. This required the number of replaced observations to be larger than the input dimension in order to benefit. Derivatives have also been employed to speed up sampling algorithms by querying a surrogate model for gradients [119]. In both previous cases the algorithms were restricted to low-dimensional input but showed improvements over baselines despite the computational burden.

[138]: Solak et al. (2003), ‘Derivative Observations in Gaussian Process Models of Dynamic Systems’

[119]: Rasmussen (2003), ‘Gaussian processes to speed up hybrid Monte Carlo for expensive Bayesian integrals’

Modern GP models that use gradients always had to rely on various approximations to keep inference tractable. Solin et al. [139] linearly constrained a GP to explicitly model curl-free magnetic fields [74]. This involved using the differentiation operator and was made computationally feasible with a reduced rank eigenfunction expansion [140]. Angelis et al. [3] extended the quadrature Fourier feature expansion (QFF) [104] to derivative information. The authors used it to construct a low-rank approximation for efficient inference of ODEs with a high number of observations. Derivatives have also been included in Bayesian optimization but mainly in low-dimensional spaces [111, 91], or by relying on a single gradient observation in each iteration [163].

[139]: Solin et al. (2018), ‘Modeling and interpolation of the ambient magnetic field by Gaussian processes’
[74]: Jidling et al. (2017), ‘Linearly constrained Gaussian processes’
[140]: Solin and Särkkä (2020), ‘Hilbert space methods for reduced-rank Gaussian process regression’
[3]: Angelis et al. (2020), ‘SLEIPNIR: Deterministic and Provably Accurate Feature Expansion for Gaussian Process Regression with Derivatives’
[104]: Mutny and Krause (2018), ‘Efficient high dimensional Bayesian optimization with additivity and quadrature Fourier features’
[111]: Osborne et al. (2009), ‘Gaussian processes for global optimization’
[91]: Lizotte (2008), ‘Practical Bayesian optimization’
[163]: Wu et al. (2017), ‘Bayesian optimization with gradients’

A more task-agnostic approach was presented by Eriksson et al. [44]. The authors derived the gradient Gram matrix for the structured kernel interpolation (SKI) approximation [159], and its extension to products SKIP [51]. This was used in conjunction with fast matrix-vector multiplication on GPUs [50] and a subspace discovery algorithm to make inference efficient. Tej et al. [142] used a similar approach but further incorporated Bayesian quadrature with gradient inference to infer a noisy policy gradient for reinforcement learning to speed up training.

[44]: Eriksson et al. (2018), 'Scaling Gaussian process regression with derivatives'
[159]: Wilson and Nickisch (2015), 'Kernel interpolation for scalable structured Gaussian processes (KISS-GP)'
[51]: Gardner et al. (2018), 'Product Kernel Interpolation for Scalable Gaussian Processes'
[50]: Gardner et al. (2018), 'GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration'
[142]: Tej et al. (2020), 'Deep Bayesian Quadrature Policy Optimization'

An obvious application of gradients for inference is in optimization and in some cases linear algebra. These are two fields we will discuss further in Sec. 7.4. Probabilistic versions of linear algebra and quasi-Newton algorithms can be constructed by modeling the Hessian with a matrix-variate normal distribution and update the belief from gradient observations [64, 157, 35, 155]. In Sec. 11 we will connect this to GP inference for a special kernel. Inference in such models has cost $\mathcal{O}(N^2D + N^3)$.

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'
[157]: Wills and Schön (2019), 'Stochastic quasi-Newton with line-search regularization'
[35]: de Roos and Hennig (2019), 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'
[155]: Wenger and Hennig (2020), 'Probabilistic Linear Solvers for Machine Learning'

Extending classic quasi-Newton algorithms to a nonparameteric Hessian estimate has been done by Hennig and Kiefel [65] and followed up by Hennig [63]. The authors modeled the elements of the Hessian using a high-dimensional GP with the RBF kernel and a special matrix-variate structure to allow cost-efficient inference. They also generalized the traditional secant equation to integrate the Hessian along a path for observations, which was possible due to the closed-form integral expression of the RBF kernel. Wills and Schön [158] expanded this line of work in two directions. They used the same setup as Hennig and Kiefel [65] but explicitly encoded symmetry of the Hessian estimate. The authors also considered modeling the joint distribution of function, gradient and Hessian ($[f, \boldsymbol{g}, \boldsymbol{H}]$) for system identification in the presence of significant noise, and where the computational requirement of inference was less critical. In Section 7.4 we present two similar optimization strategies that utilize exact efficient gradient inference for nonparametric optimization.

[65]: Hennig and Kiefel (2013), 'Quasi-Newton method: A new direction'
[63]: Hennig (2013), 'Fast Probabilistic Optimization from Noisy Gradients'
[158]: Wills and Schön (2017), 'On the construction of probabilistic Newton-type algorithms'

7.3 Theory

Kernel matrices of Gaussian processes (GPs) built from gradient observations are highly structured. In this section, after reviewing GPs, we show that for standard kernels, the kernel Gram matrix can be decomposed into a Kronecker product with an additive low-rank correction, as exemplified in Fig. 7.1. Exploiting this structure, *exact* GP inference with gradients is feasible in $\mathcal{O}(N^2D + (N^2)^3)$ operations instead of $\mathcal{O}((DN)^3)$ when inverting the kernel matrix exactly. Furthermore, the same structure enables storage of $\mathcal{O}(N^2 + ND)$ values instead of $\mathcal{O}((ND)^2)$.

Gaussian Processes

Definition 7.3.1 *A Gaussian process $f \sim \mathcal{GP}(\mu, k)$ is a random process with mean function $\mu : \mathbb{R}^D \mapsto \mathbb{R}$ and covariance function $k : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ such that f evaluated at a finite set of inputs follow a multi-variate normal distribution [120].*

[120]: Rasmussen and Williams (2006), *Gaussian Processes for Machine Learning* §2.2

GPs are popular nonparametric models with numerous favorable properties, of which we highlight their closure under linear operations. A linear operator acting on a GP results again in a GP. Let $\mathcal{L}, \mathcal{M}$ be linear operators acting on f. Then the joint distribution of $\mathcal{L}f$ and $\mathcal{M}f$ is:

$$\begin{bmatrix} \mathcal{L}f \\ \mathcal{M}f \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} \mathcal{L}\mu \\ \mathcal{M}\mu \end{bmatrix}, \begin{bmatrix} \mathcal{L}k\mathcal{L}' & \mathcal{L}k\mathcal{M}' \\ \mathcal{M}k\mathcal{L}' & \mathcal{M}k\mathcal{M}' \end{bmatrix}\right), \tag{7.1}$$

where $\mathcal{L}'$ and $\mathcal{M}'$ act on the second argument of the covariance function k. The conditional $\mathcal{L}f \mid \mathcal{M}f$ is obtained with standard Gaussian algebra computations and requires the inversion of $\mathcal{M}k\mathcal{M}'$.

See Ch. 3 for standard Gaussian computations.

Examples of linear operators comprise projections, integration, and differentiation. We focus here on inference of either f itself, its gradient $\boldsymbol{g} = \nabla f$, or its Hessian matrix $\boldsymbol{H} = \nabla\nabla^\top f$ conditioned on gradient observations, i.e. $\mathcal{L} = \{\mathrm{Id}, \nabla, \nabla\nabla^\top\}$ and $\mathcal{M} = \nabla$.

Application of $\mathcal{L} = \mathrm{Id}$ does nothing, i.e. $\mathcal{L}f = f$.

Notation We collect gradient observations $\boldsymbol{g}_a \in \mathbb{R}^D$ at locations $\boldsymbol{x}_a \in \mathbb{R}^D$, $a = 1, \ldots, N$ which we vertically stack into the data matrices $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ and $\boldsymbol{G} \in \mathbb{R}^{D\times N}$. The object of interest is the Gram matrix $\nabla\boldsymbol{K}\nabla' \in \mathbb{R}^{DN\times DN}$ where $\boldsymbol{K} = k(\boldsymbol{X}, \boldsymbol{X})$ and ∇, ∇' act w.r.t. all elements of $\boldsymbol{X}$. We let subscripts a, b identify indices related to *data points*, e.g., $\boldsymbol{x}_a, \boldsymbol{x}_b$. Superscript indices i, j refer to indices along the input *dimension*. In further abuse of notation we will let the operation $\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{c}$ denote the subtraction of $\boldsymbol{c}$ from each column in $\boldsymbol{X}$.

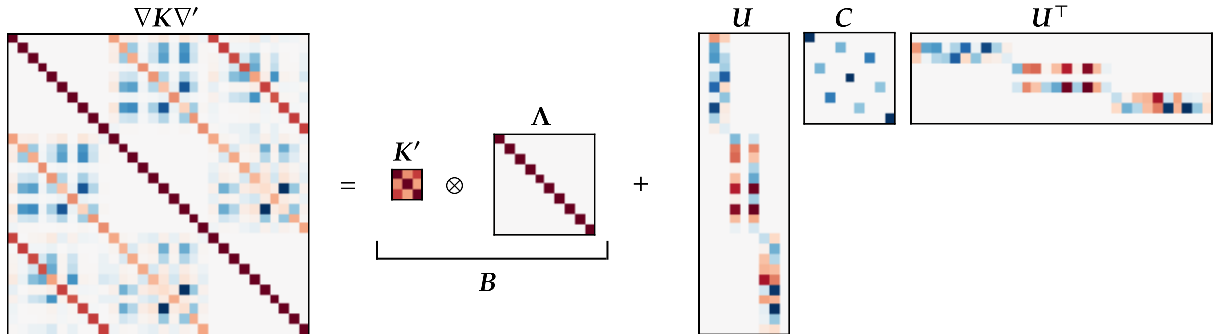


Figure 7.1: Gram matrix built from three 10-dimensional gradient observations using a stationary isotropic exponential quadratic kernel. Explicit expression *(left)* and its decomposition into a Kronecker product $\boldsymbol{B}$ and low-rank correction $\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top$ that allows for efficient inversion using Woodbury's matrix lemma (cf. Sec. 7.3) if $N < D$ *(right)*. Colored elements indicate nonzero values (red for positive, blue for negative) and white means the value is 0.

Exploiting Kernel Structure

The efficient inversion of $\nabla\boldsymbol{K}\nabla'$ relies on its somewhat repetitive structure involving a Kronecker product (see Fig. 7.1) caused by application of the product and chain rule of differentiation to the kernel. The Kronecker product $\boldsymbol{A} \otimes \boldsymbol{B}$ produces a matrix with blocks $a_{ij}\boldsymbol{B}$.

cf. Van Loan [148] for properties of the Kronecker product.
[148]: Van Loan (2000), 'The ubiquitous Kronecker product'

Any kernel $k(\boldsymbol{x}_a, \boldsymbol{x}_b)$ with inputs $\boldsymbol{x}_a, \boldsymbol{x}_b \in \mathbb{R}^D$, can be equivalently written in terms of a scalar function $r : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ as $k(r(\boldsymbol{x}_a, \boldsymbol{x}_b)) =: k_{ab}(r)$. Note the general definition of r, which in particular is more general than the distance used by stationary kernels. Since k is also a scalar function of $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, r could be equal to k if there is no way to further condense the relationship between $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$.

Definition 7.3.2 *Write $k(\boldsymbol{x}_a, \boldsymbol{x}_b) = k_{ab}(r)$ with $r \in \mathbb{R}$. Define $k'_{ab} = \frac{\partial k(r(\boldsymbol{x}_a, \boldsymbol{x}_b))}{\partial r}$, $k''_{ab} = \frac{\partial^2 k(r(\boldsymbol{x}_a, \boldsymbol{x}_b))}{\partial r^2}$ and $\partial_a{}^i = \frac{\partial}{\partial x_a{}^i}$ and similarly for $\partial_b{}^j$.*

The derivatives of k w.r.t. $\partial \boldsymbol{x}_a{}^i$ and $\partial \boldsymbol{x}_b{}^j$ can be written as

$$\begin{aligned} \partial_a{}^i k_{ab}(r) &= k'_{ab}(r)\, \partial_a{}^i r \\ \partial_b{}^j k_{ab}(r) &= k'_{ab}(r)\, \partial_b{}^j r \\ \partial_a{}^i \partial_b{}^j k(r) &= k'_{ab}(r) \cdot \partial_a{}^i \partial_b{}^j r + k''_{ab}(r) \cdot (\partial_a{}^i r)(\partial_b{}^j r) \end{aligned} \tag{7.2}$$

The $\nabla \boldsymbol{K} \nabla'$ kernel defines a matrix-variate covariance. It can be shown that columns sampled with this covariance are curl-free [106, 49] and simple modification can make them divergence-free [94, 72]. This is not surprising since we model the gradient of a scalar potential, yet these properties have been explicitly derived for the RBF kernel.

[49]: Fuselier Jr (2007), 'Refined error estimates for matrix-valued radial basis functions'

[106]: Narcowich and Ward (1994), 'Generalized Hermite interpolation via matrix-valued conditionally positive definite functions'

[94]: Macêdo and Castro (2010), *Learning divergence-free and curl-free vector fields with matrix-valued kernels*

[72]: Holderrieth et al. (2021), 'Equivariant Learning of Stochastic Fields: Gaussian Processes and Steerable Conditional Neural Processes'

This expression is still general but we can already see that the derivatives of k w.r.t. r depend only on the indices a, b of the data points and form $N \times N$ matrices that we call $\boldsymbol{K}'$ and $\boldsymbol{K}''$. Importantly, they do *not* depend on the dimensional indices i, j. While the abundance of indices invites for a tensor-like implementation, the prime gain comes from writing Eq. (7.2) in matrix form. Doing so permits linear algebra operations that are not applicable to tensors. To organize the matrix we use the convention of ordering the entries in the Gram matrix $\nabla \boldsymbol{K} \nabla'$ first according to the N data points $\boldsymbol{x}_{1:N}$, and then according to dimension, i.e.,

$$\nabla \boldsymbol{K} \nabla' = \begin{pmatrix} \nabla k(\boldsymbol{x}_1, \boldsymbol{x}_1)\nabla' & \dots & \nabla k(\boldsymbol{x}_1, \boldsymbol{x}_N)\nabla' \\ \vdots & \ddots & \vdots \\ \nabla k(\boldsymbol{x}_N, \boldsymbol{x}_1)\nabla' & \dots & \nabla k(\boldsymbol{x}_N, \boldsymbol{x}_N)\nabla' \end{pmatrix}, \tag{7.3}$$

where each block has the size $D \times D$. We highlight this ordering as it deviates from the conventional way found in the literature. Each element of the a, b^{th} block take the form $\partial_a{}^i \partial_b{}^j k(r)$ specified in Eq. (7.2), where no assumption on the structure of the kernel has been done at this point.

The same results can be derived for a dimension-prioritized ordering.

We specify the matrix form of the general expression of Eq. (7.2) for two overarching classes of kernels: the dot product and stationary class of kernels. For these kernels, r is defined as

$$\begin{aligned} r &= (\boldsymbol{x}_a - \boldsymbol{c})^\top \boldsymbol{\Lambda} (\boldsymbol{x}_b - \boldsymbol{c}) \quad \textit{(dot product kernels)}, \\ r &= (\boldsymbol{x}_a - \boldsymbol{x}_b)^\top \boldsymbol{\Lambda} (\boldsymbol{x}_a - \boldsymbol{x}_b) \quad \textit{(stationary kernels)}, \end{aligned}$$

with an arbitrary offset $\boldsymbol{c}$ and a symmetric positive definite scaling matrix $\boldsymbol{\Lambda}$. The necessary quantities of Eq. (7.2) are summarized in Tab. 7.1.

Family	$\partial_a{}^i r(\boldsymbol{x}_a, \boldsymbol{x}_b)$	$\partial_b{}^j r(\boldsymbol{x}_a, \boldsymbol{x}_b)$	$\partial_a{}^i \partial_b^j r(\boldsymbol{x}_a, \boldsymbol{x}_b)$
dot product	$[\boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})]^i$	$[\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{c})]^j$	$\boldsymbol{\Lambda}^{ij}$
stationary	$2 \cdot [\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{x}_b)]^i$	$-2 \cdot [\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{x}_b)]^j$	$-2 \cdot \boldsymbol{\Lambda}^{ij}$

Table 7.1: Required derivatives for gradient inference used in Eq. (7.2).

The factor 2 for stationary kernels often cancel due to scalar multiplication in $k'(r)$ and $k''(r)$.

Dot Product Kernels Combining the summary in Tab. 7.1 with Eq. (7.2) we arrive at the following Gram matrix for dot product kernels with gradients

$$\partial_a{}^i \partial_b{}^j k(r) = k'_{ab}(r) \cdot \boldsymbol{\Lambda}^{ij} + k''_{ab}(r) \cdot [\boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})]^i [\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{c})]^j.$$

The first term is of Kronecker structure which is easy to invert using properties of Kronecker products. The second consists of rank-1 corrections block-wise multiplied with the scalar value k''_{ab}. The input indices are flipped for the term i.e., b appears as a row index and a as column. This shuffling is what makes the structure of the gradient Gram matrix difficult, but it can be resolved with the perfect shuffle matrix $\boldsymbol{S}_{NN}$[148]. To derive the structure of the second term we start by defining the matrix $\tilde{\boldsymbol{X}} \in \mathbb{R}^{D\times N}$, $\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{c}$. We can then form the following outer product to get the structure:

[148]: Van Loan (2000), 'The ubiquitous Kronecker product'

S_{NN} is a fourth order isotropic tensor defined as $S_{ijkl} = \delta_{il}\delta_{jk}$.
$S_{NN}\,\mathrm{vec}(M) = \mathrm{vec}(M^\top)$ for $M \in \mathbb{R}^{N\times N}$.

$$
\begin{aligned}
[\boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})]^i \left[(\boldsymbol{x}_a - \boldsymbol{c})^\top \boldsymbol{\Lambda}^\top\right]^j &= [\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_b]^i[(\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_a)^\top]^j \\
&= \sum_{m,n}^{N}[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_n]^i[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_m]^j\delta_{am}\delta_{bn} \\
&= \sum_{n,n'}^{N}\sum_{m,m'}^{N}[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_n]^i[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_m]^j\delta_{am'}\delta_{bn'}\delta_{mm'}\delta_{nn'} \\
&= \sum_{n,n'}^{N}\sum_{m,m'}^{N}\left(\delta_{am'}\cdot[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_n]^i\right)\underbrace{(\delta_{mm'}\delta_{nn'})}_{S_{NN}}\left(\delta_{bn'}\cdot[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}_m]^j\right) \\
&= \sum_{n,n'}^{N}\sum_{m,m'}^{N}[\boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}]^i_{a,m'n}\,[\boldsymbol{S}_{NN}]_{m'n,n'm}\,[\boldsymbol{I}\otimes(\boldsymbol{\Lambda}\tilde{\boldsymbol{X}})^\top]^j_{n'm,b} \\
&= \left[(\boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}})\boldsymbol{S}_{NN}(\boldsymbol{I}\otimes\tilde{\boldsymbol{X}}\boldsymbol{\Lambda})^\top\right]^{ij}_{ab}
\end{aligned}
$$

To get the right scalar value for each block outer product one has to write the term like below.

$$
\underbrace{(\boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}})}_{\boldsymbol{U}}\underbrace{(\boldsymbol{S}_{NN}\,\mathrm{diag}(\mathrm{vec}(\boldsymbol{K}'')))}_{\boldsymbol{C}}\underbrace{(\boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}})^\top}_{\boldsymbol{U}^\top} \tag{7.4}
$$

with $\boldsymbol{C}_{m'n,n'm} = \boldsymbol{K}''_{mn}\delta_{mm'}\delta_{nn'}$ a symmetric $N^2 \times N^2$ matrix.

The values in $\boldsymbol{K}''$ has to be properly distributed which is what the expression for $\boldsymbol{C}$ ensures.

The full expression of the Gram matrix is

$$
\boldsymbol{K}'\otimes\boldsymbol{\Lambda} + (\boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}})\,\boldsymbol{C}\,(\boldsymbol{I}\otimes\tilde{\boldsymbol{X}}^\top\boldsymbol{\Lambda}), \tag{7.5}
$$

where $\boldsymbol{U} = \boldsymbol{I}\otimes\boldsymbol{\Lambda}\tilde{\boldsymbol{X}} = \boldsymbol{I}\otimes\boldsymbol{\Lambda}(\boldsymbol{X}-\boldsymbol{c})$ is of size $DN \times N^2$. The matrix $\boldsymbol{C} \in \mathbb{R}^{N^2\times N^2}$ is a permutation of $\mathrm{diag}\,(\mathrm{vec}(\boldsymbol{K}''))$, i.e. a diagonal matrix that has the elements of $\boldsymbol{K}''$ on its diagonal, such that $\boldsymbol{C}\,\mathrm{vec}(\boldsymbol{M}) = \mathrm{vec}(\boldsymbol{K}''\odot\boldsymbol{M}^\top)$, for $\boldsymbol{M}\in\mathbb{R}^{N\times N}$.

$\odot$ denotes the Hadamard (element-wise) product

Table 7.2: Examples of dot product kernels where $r = (\boldsymbol{x}_a - \boldsymbol{c})^\top\boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})$.

Kernel	$k(r)$	$k'(r)$	$k''(r)$
Polynomial(p)	$\frac{r^p}{p(p-1)}$	$\frac{r^{p-1}}{(p-1)}$	r^{p-2}
Polynomial(2)	$\frac{r^2}{2}$	r	1
Exponential/Taylor	$\exp(r)$	$\exp(r)$	$\exp(r)$

The kernel is denoted Taylor due to work of Karvonen et al. [79] who showed that conditioning the kernel on order $\alpha = 0, 1, \ldots$ derivatives gives a probabilistic Taylor expansion of order α around $\boldsymbol{c}$.
[79]: Karvonen et al. (2021), 'A Probabilistic Taylor Expansion with Applications in Filtering and Differential Equations'

Stationary kernels In the same fashion as the dot product we use the results in Tab. 7.1 together with Eq. (7.2) to get the Gram structure

$$
\partial_a{}^i\partial_b{}^j k(r) = -2k'_{ab}(r)\cdot\boldsymbol{\Lambda}_{jl} - 4k''_{ab}(r)\cdot[\boldsymbol{\Lambda}(\boldsymbol{x}_a-\boldsymbol{x}_b)]^i[(\boldsymbol{x}_a-\boldsymbol{x}_b)^\top\boldsymbol{\Lambda}]^j. \tag{7.6}
$$

The overall structure also results in a matrix of the form

$$\boldsymbol{K}' \otimes \boldsymbol{\Lambda} + \boldsymbol{U}\,\boldsymbol{C}\,\boldsymbol{U}^\top,$$

where the first term $\boldsymbol{K}' \otimes \boldsymbol{\Lambda}$ as well as $\boldsymbol{C}$ remain unaltered compared to the dot product.

Writing the second term in matrix form is a bit more intricate than Eq. (7.4) due to the mixing of indices a and b, but taking the same approach we get

$$\begin{aligned}
&[\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{x}_b)]^i[(\boldsymbol{x}_a - \boldsymbol{x}_b)^\top\boldsymbol{\Lambda}]^j = [\boldsymbol{\Lambda}\boldsymbol{x}_a]^i[\boldsymbol{x}_a^\top\boldsymbol{\Lambda}]^j - [\boldsymbol{\Lambda}\boldsymbol{x}_b]^i[\boldsymbol{x}_a^\top\boldsymbol{\Lambda}]^j - [\boldsymbol{\Lambda}\boldsymbol{x}_a]^i[\boldsymbol{x}_b^\top\boldsymbol{\Lambda}]^j + [\boldsymbol{\Lambda}\boldsymbol{x}_b]^i[\boldsymbol{x}_b^\top\boldsymbol{\Lambda}]^j \\
&= \sum_{mn} \delta_{am}\delta_{bn}\left([\boldsymbol{\Lambda}\boldsymbol{x}_m]^i[\boldsymbol{x}_m^\top\boldsymbol{\Lambda}]^j - [\boldsymbol{\Lambda}\boldsymbol{x}_n]^i[\boldsymbol{x}_m^\top\boldsymbol{\Lambda}]^j - [\boldsymbol{\Lambda}\boldsymbol{x}_m]^i[\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^j + [\boldsymbol{\Lambda}\boldsymbol{x}_n]^i[\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^j\right) \\
&= \sum_{mn}\left(\delta_{am}\left([\boldsymbol{\Lambda}\boldsymbol{x}_m]^i - [\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^i\right)\right)\left(\delta_{bn}\left([\boldsymbol{x}_m^\top\boldsymbol{\Lambda}]^j - [\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^j\right)\right) \\
&= \sum_{mnpp'}\left(\delta_{am}\left(\delta_{pm}[\boldsymbol{\Lambda}\boldsymbol{x}_p]^i - \delta_{pn}[\boldsymbol{x}_p^\top\boldsymbol{\Lambda}]^i\right)\right)\left(\delta_{bn}\left(\delta_{p'm}[\boldsymbol{x}_{p'}^\top\boldsymbol{\Lambda}]^j - \delta_{p'n}[\boldsymbol{x}_{p'}^\top\boldsymbol{\Lambda}]^j\right)\right) \\
&= \sum_{mnoo'pp'}\left([\boldsymbol{\Lambda}\boldsymbol{x}_p]^i\delta_{ao}\delta_{mo}(\delta_{pm} - \delta_{pn})\right)\left([\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^j\delta_{bo'}\delta_{no'}(\delta_{p'm} - \delta_{p'n})\right) \\
&= \sum_{mn}\sum_{op}\underbrace{\delta_{ao}[\boldsymbol{\Lambda}\boldsymbol{x}_p]^i}_{\boldsymbol{U}_{ai,op}}\underbrace{\delta_{om}(\delta_{pm} - \delta_{pn})}_{\boldsymbol{L}_{op,mn}}\sum_{o'p'}\underbrace{\delta_{o'n}(\delta_{p'm} - \delta_{p'n})}_{\boldsymbol{L}_{mn,o'p'}}\underbrace{\delta_{o'b}[\boldsymbol{x}_n^\top\boldsymbol{\Lambda}]^j}_{\boldsymbol{U}_{o'p',bj}}.
\end{aligned} \tag{7.7}$$

It is possible to expand the derivation further to include the matrix $\boldsymbol{C}$ in the same way as the dot product, but the important difference is the introduction of $\boldsymbol{L}$ that is already visible. $\boldsymbol{L}$ is a sparse $N^2 \times N^2$ matrix $\boldsymbol{L}$ that subtracts $\boldsymbol{\Lambda}\boldsymbol{x}_a$ from all columns of the ath block of the block diagonal matrix $\boldsymbol{I} \otimes \boldsymbol{\Lambda}\boldsymbol{X}$. For dot product kernels we used $\boldsymbol{U} = (\boldsymbol{I} \otimes \boldsymbol{\Lambda}(\boldsymbol{X} - \boldsymbol{c}))$, for stationary kernels we instead use $\boldsymbol{U} = (\boldsymbol{I} \otimes \boldsymbol{\Lambda}\boldsymbol{X})\boldsymbol{L}$.

The second term of the Gram matrix is formed by $\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top$ in the same way as Eq. (7.4). $\boldsymbol{U}$ is however no longer a Kronecker product which makes the algorithmic details more involved. It is therefore more convenient to use the $\boldsymbol{U}\boldsymbol{L}$ representation where $\boldsymbol{L}$ is a sparse linear operator such that $\boldsymbol{U}^\top \operatorname{vec}(g) = \boldsymbol{L}^\top \operatorname{vec}(\boldsymbol{X}^\top\boldsymbol{\Lambda}g)_{mn} = \operatorname{vec}(\boldsymbol{X}^\top\boldsymbol{\Lambda}g_{mn} - \boldsymbol{X}^\top\boldsymbol{\Lambda}g_{mm})$.

Figure 7.1 illustrates the decomposition for the exponential quadratic a.k.a. radial basis function (RBF) kernel. Note how the diagonal blocks of $\boldsymbol{U}$ are empty as a result of applying $\boldsymbol{L}$.

Table 7.3: Examples for stationary kernels where $r = (\boldsymbol{x}_a - \boldsymbol{x}_b)^\top\boldsymbol{\Lambda}(\boldsymbol{x}_a - \boldsymbol{x}_b)$.

Kernel	$k(r)$	$k'(r)$	$k''(r)$
Squared exponential	$\mathrm{e}^{-r/2}$	$-\frac{1}{2}k(r)$	$\frac{1}{4}k(r)$
Matérn $\nu = 1/2$	$\mathrm{e}^{-\sqrt{r}}$	$-\frac{k(r)}{2\sqrt{r}}$	$\frac{1}{4r^{3/2}}\left(\sqrt{r}+1\right)k(r)$
Matérn $\nu = 3/2$	$(1+\sqrt{3r})\,\mathrm{e}^{-\sqrt{3r}}$	$\frac{\sqrt{3}}{2\sqrt{r}}\left(\mathrm{e}^{-\sqrt{3r}} - k(r)\right)$	$\frac{\sqrt{3}}{2\sqrt{r}}\left(\frac{k(r)}{2r} - k'(r) - \mathrm{e}^{-\sqrt{3r}}\,\frac{1+\sqrt{3r}}{2r}\right)$
Matérn $\nu = 5/2$	$\left(1+\sqrt{5r}+\frac{5r}{3}\right)\mathrm{e}^{-\sqrt{5r}}$	$\left(\frac{\sqrt{5}}{2\sqrt{r}}+\frac{5}{3}\right)\mathrm{e}^{-\sqrt{5r}} - \frac{\sqrt{5}}{2\sqrt{r}}k(r)$	$\frac{\sqrt{5}}{2\sqrt{r}}\left(\frac{k(r)}{2r} - k'(r) - \mathrm{e}^{-\sqrt{5r}}\left(\frac{1+\sqrt{5r}}{2r}+\frac{5}{3}\right)\right)$
Rational quadratic	$\left(1+\frac{r}{2\alpha}\right)^{-\alpha}$	$-\frac{1}{2}\left(1+\frac{r}{2\alpha}\right)^{-\alpha-1}$	$\frac{\alpha+1}{4\alpha}\left(1+\frac{r}{2\alpha}\right)^{-\alpha-2}$

Implementation

The structure of the Gram matrix promotes two important tricks that enable efficient inference with gradients. Computational gains come into play when $N < D$, but for any choice of N, the uncovered structure enables massive savings in storage and enables efficient approximate inversion schemes.

Low-data Regime In the high-dimensional regime with a small number of observations $N < D$ the inverse of the Gram matrix can be efficiently obtained from Woodbury's matrix inversion lemma [162]

[162]: Woodbury (1950), *Inverting modified matrices*

$$\left(\boldsymbol{B} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top\right)^{-1} = \boldsymbol{B}^{-1} - \boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{B}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{U}^\top\boldsymbol{B}^{-1} \tag{7.8}$$

(if the necessary inverses exist), combined with inversion properties of the Kronecker product. If $\boldsymbol{B}$ is cheap to invert and the dimension of $\boldsymbol{C}$ is smaller than the dimension of $\boldsymbol{B}$, then the above expression can drastically reduce the computational cost of inversion. In our case $\boldsymbol{B} = \boldsymbol{K}' \otimes \boldsymbol{\Lambda}$ for which the inverse $\boldsymbol{B}^{-1} = (\boldsymbol{K}')^{-1} \otimes \boldsymbol{\Lambda}^{-1}$ requires the inverse of the $N \times N$ matrix $\boldsymbol{K}'$. The main bottleneck is the inversion of the $N^2 \times N^2$ matrix $\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{B}^{-1}\boldsymbol{U}$ which requires $\mathcal{O}(N^6)$ operations, which is still a benefit over the naïve scaling when $N < D$.

The low-rank structure along with properties of Kronecker products leads to a general solution of the linear system $[\nabla \boldsymbol{K} \nabla'] \operatorname{vec}(\boldsymbol{Z}) = \operatorname{vec}(\boldsymbol{G})$ of the form

$$\boldsymbol{Z} = \boldsymbol{\Lambda}^{-1}\boldsymbol{G}(\boldsymbol{K}')^{-1} - \tilde{\boldsymbol{X}}\boldsymbol{Q} \tag{7.9}$$

for dot product kernels with $\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{c}$ and gradient observations $\boldsymbol{G}$. $\boldsymbol{Q}$ is the unvectorized solution to

$$(\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{B}^{-1}\boldsymbol{U}) \operatorname{vec}(\boldsymbol{Q}) = \operatorname{vec}(\tilde{\boldsymbol{X}}^\top\boldsymbol{G}(\boldsymbol{K}')^{-1}). \tag{7.10}$$

Performing the closed-form kernel Gram inversion is summarized and exemplified in the following three steps for a dot product kernel:

1. $\boldsymbol{T} = \boldsymbol{U}^\top\boldsymbol{B}^{-1} \operatorname{vec}(\boldsymbol{G})$ with $\boldsymbol{T} \in \mathbb{R}^{N\times N}$.
   - $\boldsymbol{U}^\top\boldsymbol{B}^{-1} \operatorname{vec}(\boldsymbol{G}) = \tilde{\boldsymbol{X}}^\top\boldsymbol{G}(\boldsymbol{K}')^{-1}$
2. Solve: $\left(\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{B}^{-1}\boldsymbol{U}\right) \operatorname{vec}(\boldsymbol{Q}) = \operatorname{vec}(\boldsymbol{T})$ for $\boldsymbol{Q} \in \mathbb{R}^{N\times N}$
   - $\left(\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{B}^{-1}\boldsymbol{U}\right) = \left(\boldsymbol{C}^{-1} + (\boldsymbol{K}')^{-1} \otimes \tilde{\boldsymbol{X}}^\top\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}\right)$
3. $\operatorname{vec}(\boldsymbol{Z}) = \boldsymbol{B}^{-1} \operatorname{vec}(\boldsymbol{G}) - \boldsymbol{B}^{-1}\boldsymbol{U} \operatorname{vec}(\boldsymbol{Q})$ with $\boldsymbol{Z} \in \mathbb{R}^{D\times N}$.
   - $\boldsymbol{Z} = \boldsymbol{\Lambda}^{-1}\boldsymbol{G}(\boldsymbol{K}')^{-1} - \tilde{\boldsymbol{X}}\boldsymbol{Q}(\boldsymbol{K}')^{-1}$

Corresponding expressions for stationary kernels are obtained by including the factor $\boldsymbol{L}$ wherever $\boldsymbol{U}$ appears.

General Improvements The cubic computational scaling is frequently cited as the main limitation of GP inference, but often the quadratic storage is the real bottleneck. For gradient inference that is particularly true due to the required $\mathcal{O}((ND)^2)$ memory. A second observation that arises from the decomposition is that the the Gram matrix $\nabla \boldsymbol{K} \nabla'$ is fully defined by the much smaller matrices $\boldsymbol{K}'$, $\boldsymbol{K}''$ (both $N \times N$), $\boldsymbol{\Lambda}\boldsymbol{X}$ $(D \times N)$

Algorithm 3: $\nabla K \nabla'$-MVM for dot product kernels and stationary (in red). $\tilde{X}$ refers to $X - c$ for dot product kernels or simply X for stationary.

Input: V: Matrix to multiply $\mathbb{R}^{D \times N}$,
$\tilde{X}$: Evaluation points $\mathbb{R}^{D \times N}$,
Λ: Distance metric $\mathbb{R}^{N \times N}$,
$K'(X, X)$: First derivative matrix $\mathbb{R}^{N \times N}$,
$K''(X, X)$: Second derivative matrix $\mathbb{R}^{N \times N}$

1 $M = V^\top \Lambda \tilde{X}$
2 $m_1 = \operatorname{diag}(M)$; // Multiplication with $L^\top$
3 $M = M - m_1^\top$
4 $M = K'' \odot M$
5 $m_2 = \sum_a M_{ab}$; // Multiplication with L
6 $M = m_2^\top - M$

Output: $\Lambda V K' + \Lambda \tilde{X} M$

and Λ ($D \times D$, but commonly chosen diagonal or even scalar). Thus, it is sufficient to keep only those in memory instead of building the whole $DN \times DN$ matrix $\nabla K \nabla'$, which requires at most $\mathcal{O}(N^2 + ND + D^2)$ of storage. Importantly, this benefit arises for $D > 1$ and for any choice of N. It is further known how these components act on a matrix of size $D \times N$. For dot product kernels, a multiplication of the Gram matrix with vectorized matrix $V \in \mathbb{R}^{D \times N}$ is obtained by

$$(\nabla K \nabla') \operatorname{vec}(V) = \Lambda V K' + \Lambda \tilde{X} (K'' \odot V^\top \Lambda \tilde{X}) \tag{7.11}$$

A similar expression is obtained for stationary kernels and a general routine for the multiplication is visible in Alg. 3. This multiplication expression can be used with an iterative linear solver [52, 50] to exactly solve a linear system in DN iterations, in exact arithmetic. It can also be used to obtain an approximate solution in fewer iterations. The multiplication routine is further amenable to preconditioning which can drastically reduce the required number of iterations [44] and ensure convergence, as well as popular kernel sparsification techniques to lower the computational cost.

[52]: Gibbs and MacKay (1997), *Efficient implementation of Gaussian processes*
[50]: Gardner et al. (2018), 'GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration'
[44]: Eriksson et al. (2018), 'Scaling Gaussian process regression with derivatives'

Table 7.4: Memory requirements for building the full $\nabla K \nabla'$ matrix versus the decomposition for multiplication.

D\N	10	100	1000	10000
10	78.1 kB / 2.3 kB	7.6 MB / 164.1 kB	762.9 MB / 15.3 MB	74.5 GB / 1.5 GB
25	488.3 kB / 3.5 kB	47.7 MB / 175.8 kB	4.7 GB / 15.4 MB	465.7 GB / 1.5 GB
100	7.6 MB / 9.4 kB	762.9 MB / 234.4 kB	74.5 GB / 16.0 MB	7.3 TB / 1.5 GB
250	47.7 MB / 21.1 kB	4.7 GB / 351.6 kB	465.7 GB / 17.2 MB	45.5 TB / 1.5 GB
1000	762.9 MB / 79.7 kB	74.5 GB / 937.5 kB	7.3 TB / 22.9 MB	727.6 TB / 1.6 GB

7.4 Applications

Section 7.3 showed how gradient inference for GPs can be considerably accelerated when $N < D$. We outline two related applications that rely on gradients in high dimensions and that can benefit from a gradient surrogate: optimization and probabilistic linear algebra.

Optimization

Unconstrained optimization of a scalar function $f(\boldsymbol{x}) : \mathbb{R}^D \longrightarrow \mathbb{R}$ consists of locating an input $\boldsymbol{x}_*$ such that $f(\boldsymbol{x}_*)$ attains an optimal value, here this will constitute a minimum. This occurs at a point where $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$. We focus on Hessian inference from gradients in quasi-Newton-style methods and then suggest a new method that allows inferring the minimum from gradient evaluations. Pseudo-code for an optimization algorithm that uses the inference procedure is available in Alg. 4.

See Ch. 4 for more details.

Hessian Inference

Quasi-Newton methods are a popular group of algorithms that includes the famous BFGS rule [20, 46, 55, 135]. These algorithms either estimate the Hessian $\boldsymbol{H}(\boldsymbol{x}) = \nabla\nabla^\top f(\boldsymbol{x})$ or its inverse from gradients. A step direction at iteration t is then determined as $\boldsymbol{d}_t = -[\boldsymbol{H}(\boldsymbol{x}_t)]^{-1}\nabla f(\boldsymbol{x}_t)$. Hennig and Kiefel [65] showed how popular quasi-Newton methods can be interpreted as inference with a matrix-variate Gaussian distribution conditioned on gradient information. Here we extend this idea to the nonparametric setting by inferring the Hessian from observed gradients. In terms of Eq. (7.1), we consider the linear operator $\mathcal{L}$ as the second derivative, i.e., the Hessian for multivariate functions. Once a solution $\boldsymbol{Z}_b^l$ to $(\nabla \boldsymbol{K} \nabla^\top)\,\mathrm{vec}(\boldsymbol{Z}) = \mathrm{vec}(\boldsymbol{G})$ has been obtained as presented in Section 7.3, it is possible to infer the mean of the Hessian at a point $\boldsymbol{x}_a$

[20]: Broyden (1970), 'The convergence of a class of double-rank minimization algorithms 1. general considerations'
[46]: Fletcher (1970), 'A new approach to variable metric algorithms'
[55]: Goldfarb (1970), 'A family of variable-metric methods derived by variational means'
[135]: Shanno (1970), 'Conditioning of quasi-Newton methods for function minimization'
[65]: Hennig and Kiefel (2013), 'Quasi-Newton method: A new direction'

$$[\bar{\boldsymbol{H}}(\boldsymbol{x}_a)]^{ij} = \sum_{bl} (\partial_a^i \partial_a^j \partial_b^l k) \boldsymbol{Z}_b^l. \tag{7.12}$$

This requires the third derivative of the kernel matrix and an additional partial derivative of Eq. (7.2) which results in

$$\begin{aligned} \partial_a^i(\partial_a^j \partial_b^l k(r)) = {} & k''_{ab} \cdot \boldsymbol{\Lambda}^{jl}(\partial_a^i r) + k''_{ab} \cdot \boldsymbol{\Lambda}^{il}(\partial_a^j r) \\ & + k''_{ab} \cdot (\partial_a^i \partial_a^j r)(\partial_b^l r) \\ & + k'''_{ab} \cdot (\partial_a^i r)(\partial_a^j r)(\partial_b^l r). \end{aligned} \tag{7.13}$$

By performing the contraction in Eq. (7.12) first over index l and then b it is possible to write the Hessian as a matrix of the following form

$$\bar{\boldsymbol{H}}(\boldsymbol{x}_a) = \left[\boldsymbol{\Lambda}\tilde{\boldsymbol{X}}, \boldsymbol{\Lambda}\boldsymbol{Z}\right] \begin{bmatrix} \boldsymbol{M} & \hat{\boldsymbol{M}} \\ \hat{\boldsymbol{M}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{X}}^\top \boldsymbol{\Lambda} \\ \boldsymbol{Z}^\top \boldsymbol{\Lambda} \end{bmatrix} + \boldsymbol{\Lambda} \cdot \mathrm{Tr}(\check{\boldsymbol{M}}), \tag{7.14}$$

with $\boldsymbol{M}$, $\hat{\boldsymbol{M}}$ and $\check{\boldsymbol{M}}$ diagonal matrices of size $N \times N$. These matrices contain expressions of k'' and k''' which are listed in Tab. 7.5. $\tilde{\boldsymbol{X}}$ is either $(\boldsymbol{x}_a - \boldsymbol{X})$ for stationary kernels or $(\boldsymbol{X} - \boldsymbol{c})$ for the dot product kernels. The posterior mean of the Hessian is of diagonal + low-rank structure

for a diagonal $\boldsymbol{\Lambda}$, which is common for quasi-Newton algorithms and GP modeling. The matrix inversion lemma, Eq. (7.8), can then be applied to efficiently determine the new step direction. On a high level this means that once $\boldsymbol{Z}$ in Eq. (7.9) has been found, then the cost of inferring the Hessian with a GP and inverting it is similar to that of standard quasi-Newton algorithms.

Kernel	$\boldsymbol{M}_{bb}$	$\hat{\boldsymbol{M}}_{bb}$	$\check{\boldsymbol{M}}_{bb}$	$\boldsymbol{m}_b$
dot product	$\boldsymbol{k}'''_{ab} \odot \boldsymbol{m}_b$	$\boldsymbol{k}''_{ab}$	$\delta_{ab} \cdot \boldsymbol{k}''_{ab} \odot \boldsymbol{m}_b$	$[(\boldsymbol{x}_a - \boldsymbol{c})^\top \boldsymbol{\Lambda} \boldsymbol{Z}]_b$
stationary	$8 \cdot \boldsymbol{k}'''_{ab} \odot \boldsymbol{m}_b$	$-4 \cdot \boldsymbol{k}''_{ab}$	$-4 \cdot (\boldsymbol{k}''_{ab} \odot \boldsymbol{m}_b)$	$\sum_l [\tilde{\boldsymbol{X}} \odot \boldsymbol{\Lambda} \boldsymbol{Z}]_b^l$

Table 7.5: Quantities required in Eq. (7.14) for Hessian inference with different families of kernels.

Inferring the Optimum

The standard operation of Gaussian process regression is to learn a mapping $f(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$. With the gradient inference it is now possible to learn a nonparametric mapping $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}^D$, *but* this mapping can also be reversed to learn an input that corresponds to a gradient. In this way it is possible to learn $\boldsymbol{x}(\boldsymbol{g})$ and we can evaluate where the model believes $\boldsymbol{x}(\boldsymbol{g} = 0)$, i.e, the optimum $\boldsymbol{x}_*$, lies to construct a new step direction. The posterior mean of $\boldsymbol{x}_*$ conditioned on the evaluation points $\boldsymbol{X}$ at previous gradients $\boldsymbol{G}$ is

$$\begin{aligned} \bar{\boldsymbol{x}}_* &= \boldsymbol{x}_t + [\nabla \boldsymbol{K}(0, \boldsymbol{G}) \nabla](\nabla \boldsymbol{K}(\boldsymbol{G}, \boldsymbol{G}) \nabla)^{-1} (\boldsymbol{X} - \boldsymbol{x}_t) \\ &= \boldsymbol{x}_t + \boldsymbol{\Lambda} \boldsymbol{Z} \boldsymbol{K}'_{b*} + \boldsymbol{\Lambda} \tilde{\boldsymbol{G}} (\boldsymbol{K}''_{b*} \odot (\boldsymbol{Z}^\top \boldsymbol{\Lambda} \tilde{\boldsymbol{g}}_*)). \end{aligned} \tag{7.15}$$

Here we included a prior mean in the inference which corresponds to the location of the current iteration $\boldsymbol{x}_t$ and all the inference has been flipped, i.e., gradients are inputs to the kernel and previous points of evaluation are observations. This leads to a new step direction determined by $\boldsymbol{d}_{t+1} = \bar{\boldsymbol{x}}_* - \boldsymbol{x}_t$. For dot product kernels $\tilde{\boldsymbol{G}} \in \mathbb{R}^{D \times N} = \boldsymbol{G} - \boldsymbol{c}$ and $\tilde{\boldsymbol{g}}_* = -\boldsymbol{c}$. For stationary kernels $\tilde{\boldsymbol{G}} = (\boldsymbol{g}_* - \boldsymbol{G}) = -\boldsymbol{G}$ and $\boldsymbol{Z}^\top \boldsymbol{\Lambda} \tilde{\boldsymbol{g}}_*$ is replaced by $\sum_l \boldsymbol{Z}_b^l \cdot (\boldsymbol{\Lambda} \boldsymbol{G})_b^l$.

Probabilistic Linear Algebra

Assume the function we want to optimize is

$$f(\boldsymbol{x}) = \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_*)^\top \boldsymbol{A} (\boldsymbol{x} - \boldsymbol{x}_*), \tag{7.16}$$

with $\boldsymbol{A} \in \mathbb{R}^{D \times D}$ a symmetric and positive definite matrix. Finding the minimum is equivalent to solving the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, because the gradient $\nabla f(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_*)$ is zero when $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}_* := \boldsymbol{b}$. To model this function we use the second order polynomial kernel

See Ch. 4 for more on this connection.

$$k(\boldsymbol{x}_a, \boldsymbol{x}_b) = \frac{1}{2} \left[(\boldsymbol{x}_a - \boldsymbol{c})^\top \boldsymbol{\Lambda} (\boldsymbol{x}_b - \boldsymbol{c}) \right]^2,$$

and we include a prior mean of the gradient $\boldsymbol{g}_c = \nabla f(\boldsymbol{c}) = \boldsymbol{A}(\boldsymbol{c} - \boldsymbol{x}_*)$. For this setup the overall computational cost decreases from $O(N^2 D + (N^2)^3)$

Algorithm 4: Two optimization routines based on Hessian and solution inference: GP-[H/X].

Input: x_0: Starting point $\mathbb{R}^D$,
$f(\cdot)$: Function handle,
$g(\cdot)$: Gradient handle,
$k(\cdot,\cdot)$: Kernel used for GP,
m: Last m iterations to store,
ε: error tolerance

```
1  d_0 = -g(x_0)
2  while |g_t|/|g_0| > ε do
3      η_t = LineSearch(d_t, f(·), g(·))
4      x_t += η_t d_t
5      g_t = g(x_t)
6      H_t = inferH(x_t | X, G) ;                    // Eq. (7.14)
7      d_t = -H_t^{-1} g_t ;                         // quasi-Newton step
8      updateData(k, x_t, g_t) ;        // Keep last m observations
9      d_t = inferMin(0 | X - x_t, G) ;              // Eq. (7.15)
10     if d_t^T g_t > 0 then
11         d_t = -d_t ;                              // Ensure descent
```

to $O(N^2D + N^3)$ because Eq. (7.10) has the analytical solution

$$Q = \frac{1}{2}(\tilde{X}^\top \Lambda \tilde{X})^{-1}(\tilde{X}^\top A\tilde{X}),$$

which only requires the inverse of an $N \times N$ matrix instead of an $N^2 \times N^2$. The appearance of $(\tilde{X}^\top A\tilde{X})$ stems from

$$\begin{aligned}\tilde{X}^\top(G - g_c) &= \tilde{X}^\top(A(X - x_*) - A(c - x_*)) = \\ &= \tilde{X}^\top(A(X - c)) = \tilde{X}^\top A\tilde{X}.\end{aligned}$$

To see that Q is indeed a solution to Eq. (7.10) we look at the relevant terms for this setup and see how they operate.

$$\begin{aligned}\text{l.h.s.}: \quad & (C^{-1} + (\tilde{X}\Lambda\tilde{X})^{-1} \otimes (\tilde{X}^\top\Lambda\tilde{X}))\,\mathrm{vec}(Q) := Q^\top + (\tilde{X}^\top\Lambda\tilde{X})Q(\tilde{X}\Lambda\tilde{X})^{-1}\\ \text{r.h.s.}: \quad & \mathrm{vec}(T) = U^\top B^{-1}\,\mathrm{vec}((G - g_c)) = \mathrm{vec}(\tilde{X}^\top(G - g_c)(\tilde{X}^\top\Lambda\tilde{X})^{-1})\\ & T := (\tilde{X}^\top A\tilde{X})(\tilde{X}^\top\Lambda\tilde{X})^{-1}\end{aligned}$$

By inserting $Q = \frac{1}{2}(\tilde{X}^\top\Lambda\tilde{X})^{-1}(\tilde{X}^\top A\tilde{X})$ in the l.h.s. we obtain the r.h.s.

$$\begin{aligned}Q^\top + &(\tilde{X}^\top\Lambda\tilde{X})Q(\tilde{X}^\top\Lambda\tilde{X})^{-1}\\ &= \frac{1}{2}(\tilde{X}^\top A\tilde{X})(\tilde{X}^\top\Lambda\tilde{X})^{-1} + (\tilde{X}^\top\Lambda\tilde{X})[\frac{1}{2}(\tilde{X}^\top\Lambda\tilde{X})^{-1}(\tilde{X}^\top A\tilde{X})](\tilde{X}^\top\Lambda\tilde{X})^{-1}\\ &= \frac{1}{2}(\tilde{X}^\top A\tilde{X})(\tilde{X}^\top\Lambda\tilde{X})^{-1} + \frac{1}{2}(\tilde{X}^\top A\tilde{X})(\tilde{X}^\top\Lambda\tilde{X})^{-1}\\ &= (\tilde{X}^\top A\tilde{X})(\tilde{X}^\top\Lambda\tilde{X})^{-1} = T\end{aligned}$$

It is now possible to apply the Hessian and optimum inference from above to linear algebra at reduced cost. If the Hessian inference (cf. Sec. 7.4) is used in this setting, then it leads to a matrix-based probabilistic linear solver [8, 64]. If instead the reversed inference of the optimum is used (cf. Sec. 7.4), it will lead to an algorithm reminiscent of the solution-based

[8]: Bartels et al. (2019), 'Probabilistic Linear Solvers: A Unifying View'
[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

probabilistic linear solvers [8, 25]. A full comparison is beyond the scope of this manuscript and is left for future work.

[8]: Bartels et al. (2019), 'Probabilistic Linear Solvers: A Unifying View'
[25]: Cockayne et al. (2019), 'A Bayesian conjugate gradient method'

7.5 Experiments

In the preceding section we outlined two applications where nonparametric models could benefit from efficient gradient inference in high dimensions. Some of these ideas have been explored in previous work with the focus of improving traditional baselines, but always with various tricks to circumvent the expensive gradient inference. Since the purpose of this paper is to enable gradient inference and not develop new competing algorithms, the presented experiments are meant as a proof-of-concept to assess the feasibility of high-dimensional gradient inference for these algorithms. To this end, the algorithms only used available gradient information in concordance with the baseline.

Follow-up work could look into incorporation of additional information, efficient implementation and suitability of kernels.

Several extensions are outlined in Sec. 7.2.

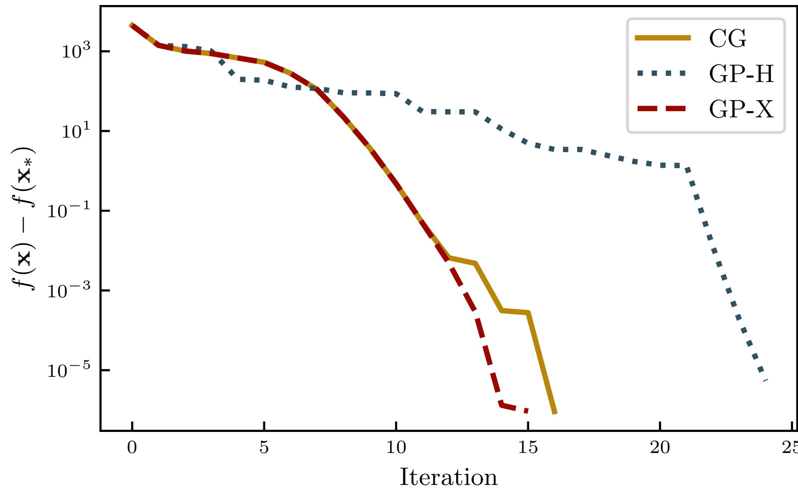


Figure 7.2: Optimization of a 100-dimensional quadratic function, Eq. (7.16), using Alg. 4 with a quadratic kernel as outlined in Sec. 11. The new solution-based inference shows performance similar to CG. The presented Hessian-based algorithm uses a fixed $\boldsymbol{c} = 0$ which compromises the performance.

Linear Algebra

Consider the linear algebra, i.e., quadratic optimization, problem in Eq. (7.16). Quadratic problems are ubiquitous in machine learning and engineering applications, since they form a cornerstone of nonlinear optimization methods. In our setting, they are particularly interesting due to the computational benefits highlighted in section 11. There has already been plenty of work studying the performance of probabilistic linear algebra routines [155, 8, 25], of which the proposed Hessian inference for linear algebra is already known [64]. We include a synthetic example of the kind Eq. (7.16) to test the new reversed inference of the solution in Eq. (7.15). Figure 7.2 compares the convergence of the gold-standard method of conjugate gradients (CG) [67] with Alg. 4 using the efficient inference of section 11. The matrix $\boldsymbol{A}$ was generated to have the spectrum

[155]: Wenger and Hennig (2020), 'Probabilistic Linear Solvers for Machine Learning'
[8]: Bartels et al. (2019), 'Probabilistic Linear Solvers: A Unifying View'
[25]: Cockayne et al. (2019), 'A Bayesian conjugate gradient method'

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

[67]: Hestenes and Stiefel (1952), 'Methods of conjugate gradients for solving linear systems'

$$\lambda_i = \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{N-1} \cdot \rho^{N-i} \cdot (N-i),$$

with $\lambda_{min} = 0.5$, $\lambda_{max} = 100$ yielding a condition number of $\kappa(\boldsymbol{A}) = 200$ and $\rho = 0.6$ so approximately the 30 largest eigenvalues are located in

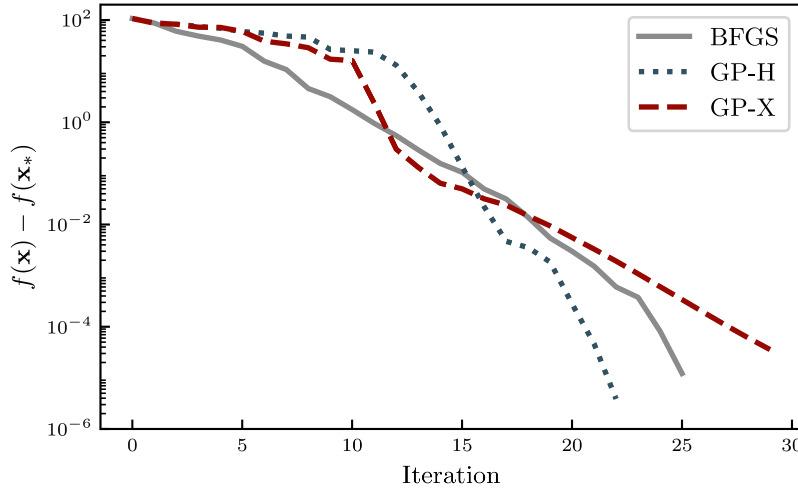


Figure 7.3: Comparison of Alg. 4 with an isotropic RBF kernel against `scipy`'s implementation of BFGS on a 100-dimensional version of Eq. (7.17). All algorithms shared the same line search routine and show similar performance.

$[1, 100]$ and the rest distributed in $[0.5, 1]$. The GP-algorithm retained all the observations to operate similarly to other probabilistic linear algebra routines. In particular, the probabilistic methods also use the optimal step length $\eta_t = -\boldsymbol{d}_t^\top \boldsymbol{g}_t / \boldsymbol{d}_t^\top \boldsymbol{A} \boldsymbol{d}_t$ that is typically used by CG.

The expression for η replaces line 3 in Alg. 4

A relative tolerance in gradient norm of 10^{-5} was used as termination criterion due to numerical instabilities. The starting and solution points were sampled according to $\boldsymbol{x}_0 \sim \mathcal{N}(0, 5^2 \cdot \boldsymbol{I})$ and $\boldsymbol{x}_* \sim \mathcal{N}(-2 \cdot \mathbf{1}, \boldsymbol{I})$. The Hessian-based optimization used a fixed $\boldsymbol{c} = \mathbf{0}$ and $\boldsymbol{g}_c = \boldsymbol{A}(\boldsymbol{c} - \boldsymbol{x}_*) = -\boldsymbol{A}\boldsymbol{x}_* = -\boldsymbol{b}$ in the linear system interpretation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

Nonlinear Optimization

The prospect of utilizing a nonparametric model for optimization is more interesting to evaluate in the nonlinear setting. In Fig. 7.3 the convergence of both versions of Alg. 4 is compared to `scipy`'s BFGS implementation. The nonparametric models use an isotropic RBF kernel with the last 2 observations for inference. All algorithms share the same line search routine. The function to be minimized is a relaxed version of a 100-dimensional Rosenbrock function [124]

The lengthscale $\boldsymbol{\Lambda} = 9 \cdot \boldsymbol{I}$ for GP-H and $\boldsymbol{\Lambda} = 0.05 \cdot \boldsymbol{I}$ for GP-X.

$$f(\boldsymbol{x}) = \sum_{i=1}^{D-1} x_i^2 + 2 \cdot (x_{i+1} - x_i^2)^2. \tag{7.17}$$

[124]: Rosenbrock (1960), 'An Automatic Method for Finding the Greatest or Least Value of a Function'

A hyperplane of the function can be seen on the left in Fig. 7.5 for the first two dimensions with every other dimension evaluated at 0.

The relaxed version was used to better control the magnitude of the gradients for the high-dimensional problem. This was important because the RBF kernels used for the optimization used a fixed $\boldsymbol{\Lambda}$, which could lead to numerical issues if the magnitude of the steps and gradients drastically changed between iterations.

Runtime Comparison

To better understand the computational saving from the presented decomposition we compare to the CPU runtime of building and solving

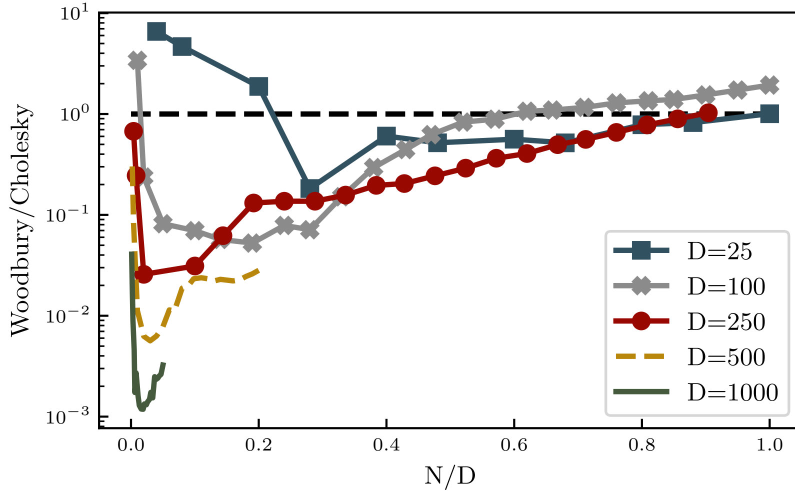


Figure 7.4: Relative CPU runtime comparison of the proposed Woodbury and traditional Cholesky decomposition to construct and solve $\nabla K(X, X)\nabla' \operatorname{vec}(Z) = \operatorname{vec}(\nabla f(X))$. For different dimensions (D) a range of observations (N) is presented for an isotropic RBF kernel.

$\nabla K \nabla' \operatorname{vec}(Z) = \operatorname{vec}(\nabla f(X))$ with the traditional Cholesky decomposition. The results are presented in Fig. 7.4 in terms of the ratio of observations to dimension for different dimensions. For the larger dimensions a maximum size of the Gram matrix was 50 000 at which point the hardware ran out of memory for the Cholesky decomposition. The Woodbury decomposition can fit more observations than the full Cholesky for a limited memory budget in the *high*-dimensional limit. To generate the data we sampled gradients from Eq. (7.17) for different dimensions and random input data $x \in [-2, 2]^D$. To alleviate numerical problems an identity matrix was added to the Gram matrix so all eigenvalues are larger than 1. Each data point in Fig. 7.4 is the average of three repetitions of a problem. In low D and at very small N/D our naïve `python` implementation suffers from overhead of pythonic operations compared to the vectorized implementation of the Cholesky decomposition in `scipy`. For larger dimensions with a small number of observations the computational cost can be reduced by several orders of magnitude despite the implementational overhead.

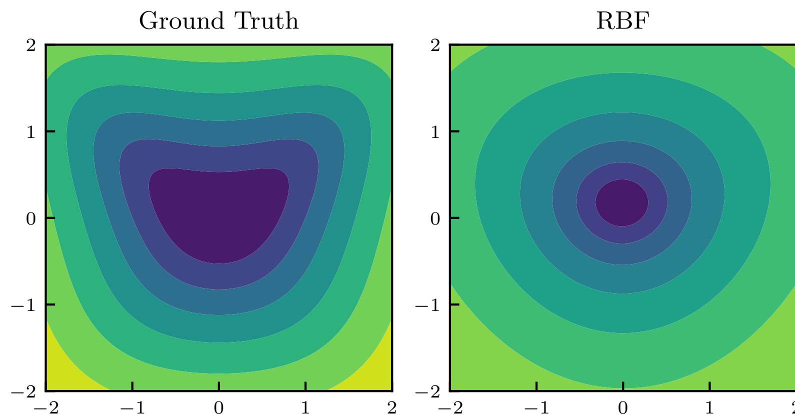


Figure 7.5: The first two dimensions of Eq. (7.17) along with the inferred curvature from 1000 randomly distributed samples. The inferred function has identified the minimum and a slight elongation of the function but not the minute details of the shape.

Another aspect of the decomposition is the MVM presented in Section 7.3 which can reduce computations and memory requirements. A qualitative evaluation is available in Fig. 7.5. The right plot shows a hyperplane with function values inferred from gradient observations evaluated at

$N = 1000$ uniformly randomly distributed evaluations in the hypercube $x_n^i \in [-2, 2]^{100}$. Constructing the Gram matrix for these observations would require $(1000 \cdot 100)^2$ floating point numbers, which for double precision would amount to > 74 GB of memory, see Tab. 7.4.

The multiplication in Eq. (7.11) was used in conjunction with an iterative linear solver to approximately solve the linear system. This approach required storage of $3ND + 3N^2$ numbers (CG requirements and intermediate matrices included) amounting to a total of only 25 MB of RAM. The solver ran for 520 iterations until a relative tolerance of 10^{-6} was reached, which took 4.9 seconds on a 2.2GHz 8-core processor. Extrapolating this time to $100 \cdot 1000$ iterations (the time to theoretically solve the linear system exactly) would yield approximately 16 minutes. Such iterative methods are sensitive to roundoff errors and are not guaranteed to converge for such large matrices without preconditioning. We did not employ preconditioning as our focus was on the computational feasibility rather than the best possible model. It would also require research into efficient utilization of the MVM structure for preconditioning and is therefore left for future work. The required number of iterations to reach convergence vary with the lengthscale of the kernel and chosen tolerance. For this experiment a lengthscale of $\ell^2 = 10 \cdot D$ was used with the isotropic RBF kernel, i.e., the inverse lengthscale matrix $\mathbf{\Lambda} = 10^{-3} \cdot \boldsymbol{I}$.

Alg. 3 shows the implementation of the multiplication.

7.6 Discussion

We have presented how structure inherent in the kernel Gram matrix can be exploited to lower the cost of GP inference with gradients from cubic to linear in the dimension. This technical observation principally opens up entirely new perspectives for high-dimensional applications in which gradient inference has previously been dismissed as prohibitive. We demonstrate on a conceptual level the great potential of this reformulation on various algorithms. The major intention behind the paper, however, is to spark research to overhaul probabilistic algorithms that operate on high-dimensional spaces and leverage gradient information.

The speed-up in terms of dimensionality does not come without limitations. Our proposed decomposition compromises the number of permissible gradient evaluations compared to the naïve approach of gradient inference. Hence, our method is applicable only in the low-data regime in which $N < D$. This property is unproblematic in applications that benefit from a local gradient model, e.g., in optimization. Nevertheless, we also found a remedy for the computational burden when $N > D$ using iterative linear solvers. Furthermore, the uncovered structure allows us to store only the quantities that are necessary to multiply the Gram matrix with an arbitrary vector. We thus showed that global models of the gradient are possible when a low-confidence gradient belief is sufficient. This is of particular interest for GP implementations that leverage the massive parallelization available on GPUs where available memory often becomes the bottleneck.

The most efficient numerical algorithms use knowledge about their input to speed up the execution. Explicit structural knowledge is usually reflected in hard-coded algorithms, e.g., linear solvers for matrices with

specific properties that are known a priori [56]. Structure can also be included in probabilistic numerical methods where the chosen model encodes known symmetries and constraints. At the same time, these methods are robust towards numeric uncertainty or noise, which can be included in the probabilistic model. Since Gaussian processes form a cornerstone of probabilistic numerical methods [66], our framework allows the incorporation of additional functional constraints into numerical algorithms for high-dimensional data. Actions taken by such algorithms are then better suited to the problem at hand. The cheap inclusion of GP gradient information in numerical routines might therefore enable new perspectives for algorithms with an underlying probabilistic model.

Other kind of matrix structures include: banded matrices, Toeplitz matrices, circulant et.c.

[56]: Golub and Van Loan (2012), *Matrix computations*

[66]: Hennig et al. (2015), 'Probabilistic numerics and uncertainty in computations'

7.7 Future Directions

Gradient information has been used in GPs before but always in conjunction with some efficient approximation technique. This meant that development of nonparametric optimization methods would also have to rely on said approximations or incur a hefty computational overhead to include gradient information. We realized the importance of this trade off beyond optimization and therefore spent less focus on the nonparametric optimization models and more on the general gradient inference. This means that there still exists plenty of opportunities for nonparametric optimization, of which a few will here be mentioned.

Traditional QN methods ignore the function evaluations when estimating the Hessian due to the chosen quadratic model. This is unfortunate since function evaluations can help determine the magnitude of the curvature estimate, but it is not necessary in the presence of a line search.For a quadratic model one must consider the following. If the objective function is a quadratic then the model is a perfect fit and additional function evaluations only provides information about the offset of the function. If the objective function is not a quadratic then the model has to approximately fit the data which imposes an additional weighting of function and gradient evaluations, resulting in more parameters to tune. A joint nonparametric model of $(f, \nabla f)$ could help alleviate this scaling issue by automatically choosing the weighting by choice of kernel. It also allows enough flexibility to satisfy both observations of functions and derivatives. The high flexibility of the model also means that additional information from a line search routine, that is otherwise simply discarded, can be used in the nonparametric model to further improve the model. An example of such a line search routine that is based on a GP surrogate was developed by Mahsereci and Hennig [99] with specific focus on ML optimization.

This was explored in Ch.5.

[99]: Mahsereci and Hennig (2017), 'Probabilistic line searches for stochastic optimization'

One theoretically pleasing and potentially strong selling point is to include the real secant equation as observation for the Hessian estimate [157]. This would require integration of the Hessian along paths which one would expect to result in a difficult expression for the kernel Gram matrix. For a scalar function f the path integrals are independent of the

See Eq. (4.17) in Sec.4.6 for explanation.

[157]: Wills and Schön (2019), 'Stochastic quasi-Newton with line-search regularization'

path so the kernel covariance takes the convenient form

$$\int_0^1 \int_0^1 (\nabla\nabla^\top) k(\tau, \tau')(\nabla\nabla^\top)' \, d\tau \mathrm{d}\tau' = \\ \nabla k(\boldsymbol{x}_a, \boldsymbol{x}_c)\nabla^\top + \nabla k(\boldsymbol{x}_b, \boldsymbol{x}_d)\nabla^\top - \nabla k(\boldsymbol{x}_a, \boldsymbol{x}_d)\nabla^\top - \nabla k(\boldsymbol{x}_b, \boldsymbol{x}_c)\nabla^\top,$$

where τ is a parameterization between $\tau(0) = \boldsymbol{x}_a$ and $\tau(1) = \boldsymbol{x}_b$, and τ' is analogous for $\boldsymbol{x}_c$ and $\boldsymbol{x}_d$. This expression only requires a minor manipulation of the derivations in Sec. 7.3 to be made computationally feasible.

Functional analysis is a fundamental tool for solving ordinary or partial differential equations [128], which are usually expressed in terms of derivatives. Due to the close connection between functional analysis and GPS it might be possible to provide data-driven solutions to above mentioned differential equations [88]. On a related note it would then also be interesting to analyze effect of prior assumptions encoded in the kernel such as smoothness or stationarity for the development of algorithms.

Commonly known as ODEs or PDEs.
[128]: Saitoh and Sawano (2016), *Theory of reproducing kernels and applications*

[88]: Lange-Hegermann (2018), 'Algorithmic linearly constrained Gaussian processes'

Epilogue

Chapter 8

Summary and Outlook

The goal of this thesis has been to tackle important aspects that relate to the training of contemporary ML models. A central part has focused on dealing with noise in stochastic optimization to make the parameter updates more robust and speed up convergence of the underlying optimization algorithm. The noise that arises due to subsampling has mainly been dealt with as uncertainty in the form of a Gaussian likelihood based on the application of the central limit theorem. This facilitated the use of probabilistic methods to achieve the goals in three different ways. Each of the preceding chapters provided a short overview of possible future directions that are tuned to the content of the corresponding chapter. Here we look at the larger picture to see what future directions emerge from the combination of the chapters.

Chapters 5 and 6 focused on optimization from the more traditional view of approximating curvature with a quadratic metric. Chapter 5 tried to scale an arbitrary underlying metric to improve the scale of the prior distribution for the sake of optimization. In the absence of oracle information a surrogate was introduced which was useful for determining a suitable learning rate when training ML models, particularly deep neural networks. The proposed update is applicable to a wide range of popular optimization algorithms. While the resulting algorithm is more expensive per-iteration than the fixed-step counterpart, it does show empirical advantages and most importantly it avoids the outer-loop learning rate tuning which therefore significantly reduces the total cost of training a model.[1] This is of particular importance since a significant amount of energy and labor goes into tuning the learning rate. The proposed scaling can also be extended to other forms of Bregman divergences, which can be useful for parameter training in other probabilistic models. The general idea of treating the metric as an underlying Gaussian distribution has since appeared in the work of Khan and Rue [81] , who further explored the connection between computational algorithms as special forms of approximate Bayesian inference. There is an interesting and potentially powerful connection between the curvature of a problem and the Gaussian covariance matrix. A principled approach to learning the mean and covariance is an interesting line of research since solving a quadratic problem can be seen as locating the mean of a Gaussian distribution.

1: A detail often neglected in the ML optimization community.

An important example is the KL-divergence [87].

[87]: Kullback and Leibler (1951), 'On information and sufficiency'

[81]: Khan and Rue (2021), *The Bayesian Learning Rule*

Chapter 6 focused on learning a suitable metric, similar to QN methods in the presence of noise. A computationally feasible algorithm was proposed for a matrix-variate model with normal distributed noise. The inference scheme relied on a general prior for the matrix and the posterior mean was made symmetric by a cheap singular value decomposition. In Sec. 6.11 a derivation of a posterior mean under symmetry constraints was presented with all the necessary details. A more general solution to

including symmetry knowledge in the model for the Hessian is hidden in Ch. 7. It turns out that the structure observed in the Gram matrix for gradients is the same that occur with a symmetry-encoding prior over matrix elements [64]. Furthermore, there is an almost exact equivalence between the gradient kernel used for linear algebra in Sec. 11

[64]: Hennig (2015), 'Probabilistic interpretation of linear solvers'

$$k(\boldsymbol{x}_a, \boldsymbol{x}_b) = \frac{1}{2}\left((\boldsymbol{x}_a - \boldsymbol{c})\boldsymbol{\Lambda}(\boldsymbol{x}_b - \boldsymbol{c})\right)^2,$$

Restated here for convenience.

and the symmetric matrix-variate normal distribution presented by Hennig [64]. This is maybe not surprising since the matrix-variate model assumes that there is a constant Hessian, which is also a property of the above kernel. To make the equivalence complete one has to update the observations used in Ch. 7 to use $\nabla f(\boldsymbol{x}_a) - \nabla f(\boldsymbol{x}_b)$ instead of $\nabla f(\boldsymbol{x}_a)$, which can easily be done as outlined in Sec. 7.7. The work in Ch. 6 could be combined with the results of Ch. 5 but due to the chronological order of publication it had to make due with a different learning rate estimate.

A shortcoming of probabilistic linear algebra at its current stage is that popular algorithms so far lack interpretation as a Kalman filter [77, 25]. That could allow cheaper inference and a natural transition from linear algebra to optimization, potentially with noise. The recurrence formula of CG and the tridiagonal structure uncovered by the related Lanczos iteration [56] seems to suggest that such an interpretation is possible. The connection between GP inference and linear algebra highlighted above might hold the key as it can assimilate both information of gradient and Hessian-vector products required by CG.

[77]: Kalman and Bucy (1961), 'New results in linear filtering and prediction theory'
[25]: Cockayne et al. (2019), 'A Bayesian conjugate gradient method'

[56]: Golub and Van Loan (2012), *Matrix computations* §10.1

One of the main strengths of the Bayesian approach to computation is the possibility to encode prior knowledge which can both improve the predictive capabilities and speed up the inference because less data is required. Finding ways to include even more structure is a promising future direction for the probabilistic linear algebra that made up the backbone of Ch. 6. There are plenty of opportunities when it comes to different matrix structures [56]. Of particular interest is banded structure which often appears in simulations where discretization of linear operators lead to sparse banded matrices. This means that only matrix elements up to a certain off-diagonal are populated. All other elements are 0, often resulting in sparse matrices. Banded matrices frequently occur in scientific computing when a stencil[2] is used to approximate derivatives by finite differences or statistics when observations are assumed to be Markovian[3].

General form of a k-diagonal matrix:

$$A_{ij} = \begin{cases} a_{ij} & \text{if} \quad |i-j| \leq k \\ 0 & \text{else} \end{cases}$$

2: The 3-point stencil $\frac{1}{2h^2}[1\ -2\,1]$ for a 1-dimensional function indicates that

$$f''(x) \approx \frac{1}{h^2}\left(f(x-h) - 2f(x) + f(x+h)\right)$$

where h is the length between the equidistant evaluation points.

3: A Markovian process compresses gained knowledge resulting in many variables being conditionally independent. An example is when the transition probability of a state in a time series only depends on the previous k states so $p(x_{t+1} \mid X_{1:t}) = p(x_{t+1} \mid X_{t-k:t})$.

Stochastic optimization with application in ML is a difficult yet rapidly evolving field with plenty of opportunities for improvement. By phrasing computation as inference we have seen a principled way of dealing with noise resulting from subsampling and drawn parallels between linear algebra and optimization. The high-dimensional optimization problems that arise in ML necessitates careful consideration of CPU and memory requirements and sets it apart from more traditional problems. Although each chapter of the thesis only advanced the state of knowledge in a minor capacity I believe they have, as a whole, answered important questions and identified new directions that will help future researchers advance the field even further.

Bibliography

References in alphabetic order.

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. 'Tensorflow: A system for large-scale machine learning'. In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016 (cited on page 12).

[2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. 'Learning to learn by gradient descent by gradient descent'. In: *Advances in neural information processing systems*. 2016 (cited on page 59).

[3] E. Angelis, P. Wenk, B. Schölkopf, S. Bauer, and A. Krause. 'SLEIPNIR: Deterministic and Provably Accurate Feature Expansion for Gaussian Process Regression with Derivatives'. In: *arXiv preprint* (2020) (cited on page 80).

[4] L. Armijo. 'Minimization of functions having Lipschitz continuous first partial derivatives'. In: *Pacific Journal of mathematics* 16.1 (1966) (cited on pages 33, 59).

[5] H. Asi and J. C. Duchi. 'The importance of better models in stochastic optimization'. In: *Proceedings of the National Academy of Sciences* 116.46 (2019) (cited on page 44).

[6] L. Balles and P. Hennig. 'Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2018 (cited on page 29).

[7] L. Balles, J. Romero, and P. Hennig. 'Coupling Adaptive Batch Sizes with Learning Rates'. In: *Conference on Uncertainty in Artificial Intelligence*. Association for Uncertainty in Artificial Intelligence, 2017 (cited on pages 14, 64).

[8] S. Bartels, J. Cockayne, I. Ipsen, and P. Hennig. 'Probabilistic Linear Solvers: A Unifying View'. In: *Statistics and Computing* 29 (2019) (cited on pages 90, 91).

[9] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood. 'Online Learning Rate Adaptation with Hypergradient Descent'. In: *International Conference on Learning Representations*. 2018 (cited on pages 44, 53, 59).

[10] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. 'Automatic differentiation in machine learning: a survey'. In: *Journal of machine learning research* 18 (2018) (cited on page 12).

[11] T. Bayes. 'An Essay towards Solving a Problem in the Doctrine of Chances.' In: *Philosophical Transactions (1683-1775)* 53 (1763) (cited on page 18).

[12] R. Bellman. 'Dynamic Programming'. In: *Science* 153.3731 (1966) (cited on page 4).

[13] L. Berrada, A. Zisserman, and M. P. Kumar. 'Training neural networks for and by interpolation'. In: *arXiv preprint arXiv:1906.05661* (2019) (cited on pages 50, 59).

[14] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006 (cited on pages 4, 11, 17, 19).

[15] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang. 'A Progressive Batching L-BFGS Method for Machine Learning'. In: *ArXiv e-prints* (2018) (cited on page 65).

[16] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cited on pages 27, 30, 34, 35, 63).

[17] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovi. 'Geometric deep learning: Grids, groups, graphs, geodesics, and gauges'. In: *arXiv preprint arXiv:2104.13478* (2021) (cited on pages 4, 14).

[18] T. Brown et al. 'Language Models are Few-Shot Learners'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020 (cited on page 14).

[19] C. G. Broyden. 'A class of methods for solving nonlinear simultaneous equations'. In: *Mathematics of computation* 19.92 (1965) (cited on page 38).

[20] C. G. Broyden. 'The convergence of a class of double-rank minimization algorithms 1. general considerations'. In: *IMA Journal of Applied Mathematics* 6 (1970) (cited on pages 39, 88).

[21] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. 'A stochastic quasi-Newton method for large-scale optimization'. In: *SIAM Journal on Optimization* 26.2 (2016) (cited on page 65).

[22] R. T. Q. Chen, D. Choi, L. Balles, D. Duvenaud, and P. Hennig. 'Self-Tuning Stochastic Optimization with Curvature-Aware Gradient Filtering'. In: *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*. Vol. 137. Proceedings of Machine Learning Research. PMLR, 2020 (cited on page 47).

[23] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. 'Learning phrase representations using RNN encoder-decoder for statistical machine translation'. English (US). In: *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. 2014 (cited on page 16).

[24] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)'. In: *International Conference on Learning Representations*. 2016 (cited on page 15).

[25] J. Cockayne, C. J. Oates, I. C. Ipsen, M. Girolami, et al. 'A Bayesian conjugate gradient method'. In: *Bayesian Analysis* 14 (2019) (cited on pages 60, 91, 100).

[26] J. Cockayne, C. J. Oates, T. J. Sullivan, and M. Girolami. 'Bayesian probabilistic numerical methods'. In: *SIAM Review* 61.4 (2019) (cited on page 5).

[27] R. T. Cox. 'Probability, frequency and reasonable expectation'. In: *American journal of physics* 14.1 (1946) (cited on page 17).

[28] Y. H. Dai and Y. Yuan. 'A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property'. In: *SIAM Journal on Optimization* 10.1 (1999) (cited on page 32).

[29] C. Darwin. *On the origin of species, 1859*. Murray, 1859 (cited on page 3).

[30] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. 'RMSProp and equilibrated adaptive learning rates for non-convex optimization'. In: *ICML workshop on Deep learning*. 2015 (cited on page 59).

[31] W. C. Davidon. *Variable metric method for minimization*. 5990. Argonne National Laboratory, 1959 (cited on page 38).

[32] A. Dawid. 'Some matrix-variate distribution theory: Notational considerations and a Bayesian application'. In: *Biometrika* 68.1 (1981) (cited on pages 65, 66).

[33] F. de Roos, A. Gessner, and P. Hennig. 'High-Dimensional Gaussian Process Inference with Derivatives'. In: *International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021 (cited on pages 7, 8, 79).

[34] F. de Roos and P. Hennig. 'Krylov Subspace Recycling for Fast Iterative Least-Squares in Machine Learning'. In: *arXiv preprint arXiv:1706.00241* (2017) (cited on pages 7, 8).

[35] F. de Roos and P. Hennig. 'Active Probabilistic Inference on Matrices for Pre-Conditioning in Stochastic Optimization'. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019 (cited on pages 7, 8, 60, 63, 81).

[36] F. de Roos, C. Jidling, A. Wills, T. Schön, and P. Hennig. 'A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization'. In: *arXiv preprint arXiv:2102.10880* (2021) (cited on pages 6, 7, 43).

[37] J. Dennis and J. Moré. 'Quasi-Newton methods, motivation and theory'. In: *SIAM Review* 19.1 (1977) (cited on pages 37, 39, 65).

[38] J. Dennis Jr and R. B. Schnabel. 'A view of unconstrained optimization'. In: *Handbooks in operations research and management science* 1 (1989) (cited on page 27).

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2019 (cited on page 14).

[40] J. Duchi, E. Hazan, and Y. Singer. 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'. In: *Journal of Machine Learning Research* (2011) (cited on page 59).

[41] V. Dumoulin and F. Visin. 'A guide to convolution arithmetic for deep learning'. In: *ArXiv e-prints* (2016) (cited on page 15).

[42] D. Duvenaud. 'Automatic model construction with Gaussian processes'. PhD thesis. University of Cambridge, 2014 (cited on pages 25, 26).

[43] J. L. Elman. 'Finding structure in time'. In: *Cognitive science* 14.2 (1990) (cited on page 16).

[44] D. Eriksson, K. Dong, E. Lee, D. Bindel, and A. G. Wilson. 'Scaling Gaussian process regression with derivatives'. In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018 (cited on pages 81, 87).

[45] R. Fletcher and C. M. Reeves. 'Function minimization by conjugate gradients'. In: *The Computer Journal* 7.2 (1964) (cited on page 32).

[46] R. Fletcher. 'A new approach to variable metric algorithms'. In: *The computer journal* 13 (1970) (cited on pages 39, 88).

[47] R. Fletcher and M. J. Powell. 'A rapidly convergent descent method for minimization'. In: *The computer journal* 6.2 (1963) (cited on page 38).

[48] C. A. Floudas and P. M. Pardalos. *Encyclopedia of Optimization*. Springer Science & Business Media, 2008 (cited on page 3).

[49] E. J. Fuselier Jr. 'Refined error estimates for matrix-valued radial basis functions'. PhD thesis. Texas A&M University, 2007 (cited on page 83).

[50] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. 'GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration'. In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018 (cited on pages 81, 87).

[51] J. Gardner, G. Pleiss, R. Wu, K. Weinberger, and A. Wilson. 'Product Kernel Interpolation for Scalable Gaussian Processes'. In: *The 21st International Conference on Artificial Intelligence and Statistics*. Vol. 84. Proceedings of Machine Learning Research. PMLR, 2018 (cited on page 81).

[52] M. Gibbs and D. MacKay. *Efficient implementation of Gaussian processes*. 1997 (cited on page 87).

[53] T. Glad and L. Ljung. *Control theory*. CRC press, 2014 (cited on page 77).

[54] G. Goh. 'Why Momentum Really Works'. In: *Distill* (2017) (cited on page 29).

[55] D. Goldfarb. 'A family of variable-metric methods derived by variational means'. In: *Mathematics of computation* 24 (1970) (cited on pages 37, 39, 88).

[56] G. H. Golub and C. F. Van Loan. *Matrix computations*. Vol. 3. JHU Press, 2012 (cited on pages 64, 68, 69, 95, 100).

[57] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016 (cited on pages 12, 14, 59, 75).

[58] A. K. Gupta and D. K. Nagar. *Matrix variate distributions*. Vol. 104. CRC Press, 2018 (cited on page 66).

[59] R. Haelterman. 'Analytical study of the least squares quasi-Newton method for interaction problems'. PhD thesis. Ghent University, 2009 (cited on page 37).

[60] M. Hardt, B. Recht, and Y. Singer. 'Train faster, generalize better: Stability of stochastic gradient descent'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2016 (cited on page 44).

[61] K. He, X. Zhang, S. Ren, and J. Sun. 'Deep Residual Learning for Image Recognition'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cited on pages 14, 54).

[62] H. Henderson and S. Searle. 'The vec-permutation matrix, the vec operator and Kronecker products: A review'. In: *Linear & Multilinear Algebra* 9 (1981) (cited on page 66).

[63] P. Hennig. 'Fast Probabilistic Optimization from Noisy Gradients'. In: *International Conference on Machine Learning*. Vol. 28. Proceedings of Machine Learning Research. PMLR, 2013 (cited on page 81).

[64] P. Hennig. 'Probabilistic interpretation of linear solvers'. In: *SIAM Journal on Optimization* 25 (2015) (cited on pages 7, 66, 67, 69, 78, 81, 90, 91, 100).

[65] P. Hennig and M. Kiefel. 'Quasi-Newton method: A new direction'. In: *Journal of Machine Learning Research* 14 (2013) (cited on pages 66, 81, 88).

[66] P. Hennig, M. A. Osborne, and M. Girolami. 'Probabilistic numerics and uncertainty in computations'. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2179 (2015) (cited on pages 5, 6, 60, 95).

[67] 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) (cited on pages 31, 69, 91).

[68] N. Higham. 'Computing a nearest symmetric positive semidefinite matrix'. In: *Linear Algebra and its Applications* 103 (1988) (cited on page 70).

[69] S. Hochreiter. 'The vanishing gradient problem during learning recurrent neural nets and problem solutions'. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998) (cited on page 15).

[70] S. Hochreiter and J. Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997) (cited on page 16).

[71] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. 'Stochastic variational inference.' In: *Journal of Machine Learning Research* 14.5 (2013) (cited on page 64).

[72] P. Holderrieth, M. J. Hutchinson, and Y. W. Teh. 'Equivariant Learning of Stochastic Fields: Gaussian Processes and Steerable Conditional Neural Processes'. In: *International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021 (cited on page 83).

[73] E. T. Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003 (cited on page 17).

[74] C. Jidling, N. Wahlström, A. Wills, and T. B. Schön. 'Linearly constrained Gaussian processes'. In: *Advances in Neural Information Processing Systems* 30 (2017) (cited on page 80).

[75] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. 'In-datacenter performance analysis of a tensor processing unit'. In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017 (cited on pages 12, 13).

[76] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, et al. 'Highly accurate protein structure prediction with AlphaFold'. In: *Nature* 596.7873 (2021) (cited on page 14).

[77] R. E. Kalman and R. S. Bucy. 'New results in linear filtering and prediction theory'. In: (1961) (cited on page 100).

[78] M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur. 'Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences'. In: *Arxiv e-prints* arXiv:1805.08845v1 (2018) (cited on page 22).

[79] T. Karvonen, J. Cockayne, F. Tronarp, and S. Särkkä. 'A Probabilistic Taylor Expansion with Applications in Filtering and Differential Equations'. In: *arXiv preprint arXiv:2102.00877* (2021) (cited on page 84).

[80] H. K. Khalil. *Nonlinear systems*. Prentice-Hall, 2002 (cited on page 77).

[81] M. E. Khan and H. Rue. *The Bayesian Learning Rule*. 2021 (cited on page 99).

[82] J. Kiefer and J. Wolfowitz. 'Stochastic Estimation of the Maximum of a Regression Function'. In: 23.3 (1952) (cited on page 44).

[83] D. P. Kingma and J. Ba. 'Adam: A Method for Stochastic Optimization'. In: *International Conference on Learning Representations*. Ed. by Y. Bengio and Y. LeCun. 2015 (cited on pages 47, 48, 59, 64).

[84] A. N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. 1933 (cited on page 17).

[85] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009 (cited on pages 14, 74).

[86] A. Krizhevsky. 'Learning multiple layers of features from tiny images'. MA thesis. Department of Computer Science, University of Toronto, 2009 (cited on page 53).

[87] S. Kullback and R. A. Leibler. 'On information and sufficiency'. In: *The Annals of Mathematical Statistics* 22.1 (1951) (cited on page 99).

[88] M. Lange-Hegermann. 'Algorithmic linearly constrained Gaussian processes'. In: *arXiv preprint arXiv:1801.09197* (2018) (cited on page 96).

[89] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (1998) (cited on page 15).

[90] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. 'A survey of deep neural network architectures and their applications'. In: *Neurocomputing* 234 (2017) (cited on page 15).

[91] D. J. Lizotte. 'Practical Bayesian optimization'. PhD thesis. University of Alberta, 2008 (cited on page 80).

[92] N. Loizou, S. Vaswani, I. Laradji, and S. Lacoste-Julien. 'Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence'. In: *arXiv preprint arXiv:2002.10542* (2020) (cited on pages 46, 50, 59).

[93] D. G. Luenberger. *Introduction to linear and nonlinear programming*. Vol. 28. Addison-wesley Reading, MA, 1973 (cited on page 28).

[94] I. Macêdo and R. Castro. *Learning divergence-free and curl-free vector fields with matrix-valued kernels*. IMPA, 2010 (cited on page 83).

[95] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003 (cited on pages 11, 17).

[96] D. Maclaurin, D. Duvenaud, and R. P. Adams. 'Autograd: Effortless gradients in numpy'. In: *ICML 2015 AutoML workshop*. Vol. 238. 2015 (cited on page 12).

[97] P. C. Mahalanobis. 'On the generalized distance in statistics'. In: *Proceedings of the National Institute of India*. National Institute of Science of India. 1936 (cited on pages 24, 28, 35).

[98] M. Mahsereci. 'Probabilistic Approaches to Stochastic Optimization'. PhD thesis. University of Tübingen, 2018 (cited on page 29).

[99] M. Mahsereci and P. Hennig. 'Probabilistic line searches for stochastic optimization'. In: *The Journal of Machine Learning Research* 18.1 (2017) (cited on pages 33, 44, 59, 95).

[100] C. C. Margossian. 'A review of automatic differentiation and its efficient implementation'. In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 9.4 (2019) (cited on page 13).

[101] J. Martens. 'Deep learning via Hessian-free optimization'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2010 (cited on pages 32, 65).

[102] J. Martens. *New insights and perspectives on the natural gradient method*. 2020 (cited on page 47).

[103] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012 (cited on pages 17, 20).

[104] M. Mutny and A. Krause. 'Efficient high dimensional Bayesian optimization with additivity and quadrature Fourier features'. In: *Advances in Neural Information Processing Systems* 31 (2018) (cited on page 80).

[105] V. Nair and G. E. Hinton. 'Rectified linear units improve restricted boltzmann machines'. In: *Icml*. 2010 (cited on page 15).

[106] F. J. Narcowich and J. D. Ward. 'Generalized Hermite interpolation via matrix-valued conditionally positive definite functions'. In: *Mathematics of Computation* 63.208 (1994) (cited on page 83).

[107] S. G. Nash. 'A survey of truncated-Newton methods'. In: *Journal of computational and applied mathematics* 124.1-2 (2000) (cited on page 32).

[108] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. 'Reading Digits in Natural Images with Unsupervised Feature Learning'. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011 (cited on page 54).

[109] J. Nocedal and S. J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006 (cited on pages 3, 27, 28, 30, 32, 33, 36, 37, 45, 51, 64).

[110] E. Noether. 'Invariante Variationsprobleme'. ger. In: *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* 1918 (1918) (cited on page 4).

[111] M. A. Osborne, R. Garnett, and S. J. Roberts. 'Gaussian processes for global optimization'. In: *International conference on learning and intelligent optimization*. Vol. 3. 2009 (cited on page 80).

[112] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. 'Automatic differentiation in PyTorch'. In: *NIPS-Workshops*. 2017 (cited on page 74).

[113] A. Paszke, S. Gross, F. Massa, and et. al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019 (cited on pages 12, 52).

[114] B. Pearlmutter. 'Fast exact multiplication by the Hessian'. In: *Neural computation* 6.1 (1994) (cited on pages 31, 65, 72).

[115] Polak, E. and Ribiere, G. 'Note sur la convergence de méthodes de directions conjuguées'. In: *R.I.R.O.* 3.16 (1969) (cited on page 32).

[116] B. Polyak. *Introduction to Optimization*. 1987 (cited on pages 43, 44, 46, 47, 49, 59).

[117] B. T. Polyak. 'Some methods of speeding up the convergence of iteration methods'. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964) (cited on pages 29, 64).

[118] J. Quinonero-Candela and C. E. Rasmussen. 'A unifying view of sparse approximate Gaussian process regression'. In: *The Journal of Machine Learning Research* 6 (2005) (cited on page 24).

[119] C. E. Rasmussen. 'Gaussian processes to speed up hybrid Monte Carlo for expensive Bayesian integrals'. In: *Seventh Valencia international meeting*. Vol. 7. Bayesian Statistics. 2003 (cited on page 80).

[120] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006 (cited on pages 12, 17, 22–26, 74, 79, 81).

[121] M. Riedmiller and H. Braun. 'Rprop-a fast adaptive learning algorithm'. In: *Proc. of ISCIS VII)*. Citeseer. 1992 (cited on page 51).

[122] H. Robbins and S. Monro. 'A stochastic approximation method'. In: *The Annals of Mathematical Statistics* 22.3 (1951) (cited on pages 14, 44, 46, 64).

[123] M. Rolinek and G. Martius. 'L4: Practical loss-based stepsize adaptation for deep learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018 (cited on pages 53, 59, 60).

[124] H. H. Rosenbrock. 'An Automatic Method for Finding the Greatest or Least Value of a Function'. In: *The Computer Journal* 3 (1960) (cited on pages 51, 92).

[125] S. Ruder. *An overview of gradient descent optimization algorithms*. Tech. rep. arXiv:1609.04747, 2016 (cited on page 59).

[126] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986) (cited on page 13).

[127] Y. Saad and M. Schultz. 'GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems'. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986) (cited on page 69).

[128] S. Saitoh and Y. Sawano. *Theory of reproducing kernels and applications*. Springer, 2016 (cited on page 96).

[129] S. Särkkä and A. Solin. *Applied stochastic differential equations*. Vol. 10. Cambridge University Press, 2019 (cited on page 25).

[130] T. Schaul, S. Zhang, and Y. Lecun. 'No More Pesky Learning Rates'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2013 (cited on page 59).

[131] R. M. Schmidt, F. Schneider, and P. Hennig. 'Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers'. In: *International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021 (cited on pages 28, 40, 47).

[132] F. Schneider, L. Balles, and P. Hennig. 'DeepOBS: A Deep Learning Optimizer Benchmark Suite'. In: *International Conference on Learning Representations*. 2019 (cited on page 52).

[133] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002 (cited on pages 17, 25).

[134] N. Schraudolph, J. Yu, and S. Günter. 'A stochastic quasi-Newton method for online convex optimization'. In: *Artificial Intelligence and Statistics*. 2007 (cited on page 65).

[135] D. F. Shanno. 'Conditioning of quasi-Newton methods for function minimization'. In: *Mathematics of computation* 24 (1970) (cited on pages 39, 88).

[136] J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004 (cited on pages 25, 26).

[137] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. 1994 (cited on page 31).

[138] E. Solak, R. Murray-Smith, W. Leithead, D. Leith, and C. E. Rasmussen. 'Derivative Observations in Gaussian Process Models of Dynamic Systems'. In: *Advances in Neural Information Processing Systems*. Vol. 15. 2003 (cited on page 80).

[139] A. Solin, M. Kok, N. Wahlström, T. B. Schön, and S. Särkkä. 'Modeling and interpolation of the ambient magnetic field by Gaussian processes'. In: *IEEE Transactions on robotics* 34 (2018) (cited on page 80).

[140] A. Solin and S. Särkkä. 'Hilbert space methods for reduced-rank Gaussian process regression'. In: *Statistics and Computing* 30 (2020) (cited on page 80).

[141] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. 'On the importance of initialization and momentum in deep learning'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research (2013) (cited on pages 29, 47, 59).

[142] A. R. Tej, K. Azizzadenesheli, M. Ghavamzadeh, A. Anandkumar, and Y. Yue. 'Deep Bayesian Quadrature Policy Optimization'. In: *arXiv preprint* (2020) (cited on page 81).

[143] L. Trefethen and D. Bau III. *Numerical Linear Algebra*. SIAM, 1997 (cited on pages 64, 69).

[144] G. E. Uhlenbeck and L. S. Ornstein. 'On the theory of the Brownian motion'. In: *Physical review* 36.5 (1930) (cited on page 25).

[145] A. Van der Sluis and H. A. van der Vorst. 'The rate of convergence of conjugate gradients'. In: *Numerische Mathematik* 48.5 (1986) (cited on page 32).

[146] A. van der Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998 (cited on page 64).

[147] C. Van Loan. 'The ubiquitous Kronecker product'. In: *Journal of Computational and Applied Mathematics* 123 (2000) (cited on page 68).

[148] C. F. Van Loan. 'The ubiquitous Kronecker product'. In: *Journal of computational and applied mathematics* 123 (2000) (cited on pages 82, 84).

[149] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Kaiser, and I. Polosukhin. 'Attention is all you need'. In: *Advances in neural information processing systems*. 2017 (cited on page 16).

[150] S. Vaswani, F. Kunstner, I. Laradji, S. Y. Meng, M. Schmidt, and S. Lacoste-Julien. 'Adaptive Gradient Methods Converge Faster with Over-Parameterization (and you can do a line-search)'. In: *arXiv preprint arXiv:2006.06835* (2020) (cited on pages 33, 49, 59).

[151] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien. 'Painless stochastic gradient: Interpolation, line-search, and convergence rates'. In: *Advances in Neural Information Processing Systems*. 2019 (cited on pages 44, 59).

[152] S. Vijayakumar and S. Schaal. 'Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2000 (cited on page 72).

[153] J. F. Vincent, O. A. Bogatyreva, N. R. Bogatyrev, A. Bowyer, and A.-K. Pahl. 'Biomimetics: its practice and theory'. In: *Journal of the Royal Society Interface* 3.9 (2006) (cited on page 3).

[154] M. Wahde. *Biologically inspired optimization methods: an introduction*. WIT press, 2008 (cited on page 3).

[155] J. Wenger and P. Hennig. 'Probabilistic Linear Solvers for Machine Learning'. In: *Advances in Neural Information Processing Systems*. 2020 (cited on pages 81, 91).

[156] A. Wills and T. Schön. 'Stochastic quasi-Newton with adaptive step lengths for large-scale problems'. In: *ArXiv e-prints* (2018) (cited on pages 65–67).

[157] A. Wills and T. Schön. 'Stochastic quasi-Newton with line-search regularization'. In: *arXiv preprint* (2019) (cited on pages 33, 76, 81, 95).

[158] A. G. Wills and T. B. Schön. 'On the construction of probabilistic Newton-type algorithms'. In: *Conference on Decision and Control*. Vol. 56. IEEE. 2017 (cited on page 81).

[159] A. Wilson and H. Nickisch. 'Kernel interpolation for scalable structured Gaussian processes (KISS-GP)'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2015 (cited on page 81).

[160] P. Wolfe. 'Convergence conditions for ascent methods'. In: *SIAM review* 11.2 (1969) (cited on page 33).

[161] P. Wolfe. 'Convergence conditions for ascent methods. II: Some corrections'. In: *SIAM review* 13.2 (1971) (cited on page 33).

[162] M. A. Woodbury. *Inverting modified matrices*. Statistical Research Group, 1950 (cited on pages 39, 86).

[163] J. Wu, M. Poloczek, A. G. Wilson, and P. Frazier. 'Bayesian optimization with gradients'. In: *Advances in Neural Information Processing Systems*. 2017 (cited on page 80).

[164] J. Wu, W. Hu, H. Xiong, J. Huan, V. Braverman, and Z. Zhu. 'On the noisy gradient descent that generalizes as sgd'. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research. PMLR, 2020 (cited on page 44).

[165] S. Zagoruyko and N. Komodakis. 'Wide Residual Networks'. In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2016 (cited on page 54).

Notation

The next list describes several symbols that are used within the body of the document.

General

$a, b, c, \ldots$ Scalar: lowercase

$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$ Vector: boldface lowercase

$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$ Matrix: boldface uppercase

D Dimension of parameters

N Number of data points

M Number of observations/matrix rank

$\mathcal{D}$ Data set consisting of N input-output pairs $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$

$\mathcal{B}$ Batch of data set

f General scalar function $\mathbb{R}^D \to \mathbb{R}$

ℓ Loss function $\mathbb{R}^D \to \mathbb{R}$ of single datum

$\mathcal{R}$ General regularization term $\mathbb{R}^D \to \mathbb{R}$

$\mathcal{L}$ (Regularized) Empirical loss $\sum_i \ell_i(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})$

$\mathcal{L}_{\mathcal{B}}$ Empirical loss of a batch

$\mathcal{L}_{\mathcal{D}}$ $\mathcal{L}_{\mathcal{D}} = \mathcal{L}$ is usually the empirical loss of the training set

$\mathcal{N}$ Normal distribution

ϵ Normal distributed corruption

∇ Gradient operator: vector with partial derivative w.r.t. each input dimension.

$\boldsymbol{g}_t$ Gradient of a scalar function at iteration t

$\boldsymbol{v}_t$ General step direction of optimizer

β Momentum scale for Momentum updates and Adam

$\boldsymbol{m}_t$ Momentum term for accumulating first moment of gradients

$\boldsymbol{\theta}$ D-dimensional parameters to be optimized

$\boldsymbol{\theta}_*$ Optimal parameters

$\boldsymbol{H}$ Hessian matrix of scalar function consisting all second-order partial derivatives

η_t Step length/learning rate at iteration t

$\boldsymbol{W}$ Covariance/metric matrix

$\boldsymbol{\Lambda}$ General distance matrix

$\boldsymbol{I}$ Identity matrix

$\boldsymbol{R}_t$ Noise covariance (also R_t in scalar observation)

$\boldsymbol{\phi}$ Feature matrix

ϕ Local quadratic estimate

Symbols used in Ch. 5

$\bar{f}$ Local surrogate function

ϕ Quadratic part of local surrogate function

$f^\star$ Minimum of local surrogate function

Δf Estimate of $f(\boldsymbol{\theta}_t) - f^\star$

$\boldsymbol{d}$ Local variable in quadratic surrogate function

$\hat{\boldsymbol{\theta}}_t$ Random variable for inference

$\boldsymbol{\theta}_t^\star$ Local minimizer so $\bar{f}(\boldsymbol{\theta}^\star = f^\star$

$\boldsymbol{G}_t$ Diagonal matrix for accumulating second moment of gradients used by RMSProp and AdaGrad

$\boldsymbol{V}_t$ Diagonal matrix used by Adam

Symbols used in Ch. 6

Γ Antisymmetric projection operator

λ_0 Scalar noise parameter

$\boldsymbol{\Delta}$ Difference between observation and prediction $\boldsymbol{Y} - \boldsymbol{H}_0\boldsymbol{S}$

$\boldsymbol{\Lambda}$ Scalar covariance of noise matrix

$\boldsymbol{E}$ Noise matrix

$\boldsymbol{G}$ Gram matrix $\boldsymbol{S}^\top \boldsymbol{W} \boldsymbol{S} \otimes W$

$\boldsymbol{H}_m$ Estimate of Hessian after $m = M$ observations

$\boldsymbol{P}$ Preconditioning matrix

$\boldsymbol{S}$ Search direction employed by matrix-variate model $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{S}$

$\boldsymbol{X}$ Solution to equation $(\boldsymbol{G} + \boldsymbol{R})\,\mathrm{vec}(\boldsymbol{X}) = \mathrm{vec}(\boldsymbol{\Delta})$

$\boldsymbol{Y}$ Observation of matrix-variate model $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{S} \in \mathbb{R}^{D\times M}$

$\otimes$ Kronecker product between two matrices $(\boldsymbol{A} \otimes \boldsymbol{B})_{ij,kl} = \boldsymbol{A}_{ik}\boldsymbol{B}_{jl}$

$\boldsymbol{\phi}$ Feature vector for parametric regression

h_0 Scalar mean parameter

w_0 Scalar covariance parameter

Symbols used in Ch. 7

$\mathcal{L}$ Linear operator

$\mathcal{GP}$ Gaussian process

$\mathcal{M}$ Linear operator

$\boldsymbol{G}$ Matrix of gradient observations

$\boldsymbol{K}'$ $\partial \boldsymbol{K}(r)/\partial r$

$\boldsymbol{K}''$ $\partial^2 \boldsymbol{K}(r)/\partial r^2$

$\boldsymbol{X}$ Input variable of data set ($\boldsymbol{X} = \{x_i\}_{i=1}^N$)

$\boldsymbol{x}$ Multidimensional single data point

$\boldsymbol{Y}$ All target values of data set $\boldsymbol{Y} = \{y_i\}_{i=1}^N$

$k(\cdot,\cdot)$ Kernel function

$K_{\boldsymbol{x}_* X}$ Kernel covariance between new point $\boldsymbol{x}_*$ and training set X

K_{XX} Kernel matrix evaluated at points in X

y Target value

Alphabetical Index

Adagrad, 47
Adam, 47, 48
automatic differentiation, 5, 12

Bayes' theorem, 4, 18
BFGS, 39, 88
Broyden's method, 38

central limit theorem, 18, 43
CIFAR-10, 53, 74
CIFAR-100, 54
computational graph, 12
condition number, 28, 64, 71
conditional distribution, 19
conjugate gradients, 31
conjugate prior, 20
constrained optimization, 76
convolutional layer, 15, 54, 74
covariance matrix, 20
curse of dimensionality, 4

deep learning, 14, 74
DeepOBS, 52
dense layer, 15
DFP, 38
Dirac distribution, 66
dot product kernel, 25, 83

empirical risk minimization, 4, 12, 43, 63
epistemic uncertainty, 5

F-MNIST, 53
feature matrix, 22
first-order optimization, 28
fully-connected layer, 15, 74

Gaussian distribution, 18
Gaussian process, 22, 81
gradient descent, 28

Hessian, 27, 30, 45, 88
Hessian-vector product, 31
homoscedastic, 22
Hypergradient descent, 53

ill-conditioned, 29
interpolation, 50
isotropic, 24

kernel function, 23
kernel matrix, 23
Kronecker product, 66, 82

L-BFGS, 40
L4 optimizer, 53
learning rate, 32
likelihood, 4, 21
linear operator, 82
loss function, 11
Lyapunov equation, 77

Mahalanobis distance, 28, 35
marginal distribution, 19
Matérn kernel, 24
matrix inversion lemma, 39, 68
matrix-variate normal distribution, 66
max-pooling, 16, 74
mini-batch, 13
MNIST, 53, 73
momentum, 29, 47

natural parameters, 18
Newton step, 30, 45
nonlinearity function, 15, 74
normal distribution, 18

overfitting, 13

parametric regression, 22
Polyak step, 46
posterior distribution, 4, 21
preconditioning, 33, 64, 71
prior distribution, 4, 21
probabilistic numerics, 5
product rule, 18

Quasi-Newton, 36, 65
quasi-Newton algorithm, 88

RBF kernel, 25, 92
recurrent layer, 16
regularization, 74
RMSprop, 47
Rosenbrock function, 51, 92
rotation-invariant, 25

SARCOS, 72
secant equation, 36
second-order optimization, 29
shift-invariant, 24
stationary kernel, 24, 84
stochastic gradient descent, 46
subsampling, 13

sum rule, 18
supervised learning, 4, 11
surrogate function, 30, 50
SVHN, 54
symmetric and positive definite, 30
symmetric rank-1, 38

Taylor expansion, 30, 44

unconstrained optimization, 27, 88