

---

# A Probabilistically Motivated Learning Rate Adaptation for Stochastic Optimization

---

Filip de Roos<sup>1,2</sup>

Carl Jidling<sup>3</sup>

Adrian Wills<sup>4</sup>

Thomas B. Schön<sup>3</sup>

Philipp Hennig<sup>1,2</sup>

<sup>1</sup>Department of Computer Science, University of Tübingen, Germany

<sup>2</sup>Max Planck Institute for Intelligent Systems, Tübingen, Germany

<sup>3</sup>Department of Information Technology, Uppsala University, Sweden

<sup>4</sup>School of Engineering, University of Newcastle, Australia

## Abstract

Machine learning practitioners invest significant manual and computational resources in finding suitable learning rates for optimization algorithms. We provide a probabilistic motivation, in terms of Gaussian inference, for popular stochastic first-order methods. As an important special case, it recovers the Polyak step with a general metric. The inference allows us to relate the learning rate to a dimensionless quantity that can be automatically adapted during training by a control algorithm. The resulting meta-algorithm is shown to adapt learning rates in a robust manner across a large range of initial values when applied to deep learning benchmark problems.

## 1 INTRODUCTION

Empirical risk minimization, including in particular deep learning, requires optimization of an objective  $f$  that is the sum of individual losses  $\ell$  over elements  $\mathbf{x}_d$  of the dataset  $\mathcal{D}$  with  $|\mathcal{D}| = D$  as

$$f(\boldsymbol{\theta}) = \frac{1}{D} \sum_{d=1}^D \ell(\boldsymbol{\theta}, \mathbf{x}_d) + \mathcal{R}(\boldsymbol{\theta}), \quad (1)$$

where  $\boldsymbol{\theta} \in \mathbb{R}^N$  denotes the parameter to be optimized and  $\mathcal{R}$  is an additional regularization term. It is standard practice to sub-sample the dataset into batches  $\mathcal{B} \subset \mathcal{D}$  with  $|\mathcal{B}| = B$  when evaluating the loss  $f$  and its gradient, yielding a stochastic, noisy observation

$$\ell_B(\boldsymbol{\theta}) = \frac{1}{B} \sum_{b=1}^B \ell(\boldsymbol{\theta}, \mathbf{x}_b) + \mathcal{R}(\boldsymbol{\theta}). \quad (2)$$

If the elements of the batch are drawn i.i.d. and  $1 \ll B \ll D$ , then by the Central Limit Theorem,  $\ell_B$  is approximately

Gaussian distributed around the true function value

$$p(\ell_B) = \mathcal{N}(\ell_B; f(\boldsymbol{\theta}), R), \quad (3)$$

with variance  $R$  that scales as  $\mathcal{O}(1/B)$ . While stochasticity drastically reduces computational cost and may have beneficial side-effects like improved generalization [Hardt et al., 2016, Wu et al., 2020], it also complicates parameter tuning. In contrast to classic numerical optimization routines, variants of stochastic gradient-descent (SGD) expose free parameters to the user. Chief among them is the scalar learning rate  $\eta$  that determines the step size taken in the direction  $\mathbf{v}_i$  as

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \mathbf{v}_i, \quad (4)$$

with  $\mathbf{v}_i$  chosen iteratively by the optimization routine (in the case of vanilla SGD,  $\mathbf{v}_i = \nabla \ell_B(\boldsymbol{\theta}_i)$ ). 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 the network architecture, the dataset, and the optimization method. Although various semi-automated and fully automated routines have been proposed to tune learning rates [Baydin et al., 2018, Mahsereci and Hennig, 2017, Vaswani et al., 2019], practitioners still largely rely on a manual process of repeated training runs, causing significant use of computational resources [Asi and Duchi, 2019].

## 1.1 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 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 2.2 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 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.

## 2 METHOD

To set the scene, consider a refactored second-order Taylor expansion of the function  $f(\boldsymbol{\theta}) : \mathbb{R}^N \rightarrow \mathbb{R}$  in a vector  $\mathbf{d}$  around the current location  $\boldsymbol{\theta}_i$ , using the (ground-truth, full-batch) gradient  $\nabla f_i \triangleq \nabla f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_i}$  and Hessian  $\mathbf{B}_i \in \mathbb{R}^{N \times N}$  with  $[\mathbf{B}_i]_{mn} \triangleq \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_m \partial \theta_n} |_{\boldsymbol{\theta}=\boldsymbol{\theta}_i}$ . Assuming the Hessian is invertible, we write this local approximation as

$$\begin{aligned} \bar{f}_i(\boldsymbol{\theta}_i + \mathbf{d}) &= \underbrace{\frac{1}{2}(\mathbf{d} + \mathbf{B}_i^{-1}\nabla f_i)^\top \mathbf{B}_i (\mathbf{d} + \mathbf{B}_i^{-1}\nabla f_i)}_{\phi_B(\mathbf{d})} \\ &+ \underbrace{f(\boldsymbol{\theta}_i) - \frac{1}{2}\nabla f_i^\top \mathbf{B}_i^{-1}\nabla f_i}_{f^*}. \end{aligned} \quad (5)$$

When the Hessian is positive definite, the minimum value of this quadratic approximation  $f^*$  is attained at the well-known Newton update  $\mathbf{d}_i^* = -\mathbf{B}_i^{-1}\nabla f_i$  which occurs at the point  $\boldsymbol{\theta}_i^* = \boldsymbol{\theta}_i - \mathbf{B}_i^{-1}\nabla f_i$ . Because computing this update is computationally costly, large parts of classic convex optimization (in particular, conjugate gradients and quasi-Newton methods [e.g. Nocedal and Wright, 2006, §5 & 6, respectively]) are concerned with efficient estimation of  $\mathbf{d}_i^*$  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.

### 2.1 PROBABILISTIC MODEL

We phrase the task of locating (inferring) the minimizer  $\boldsymbol{\theta}_i^* \in \mathbb{R}^N$  of the local quadratic model given in Eq. (5) at iteration  $i$  based on noisy observations of the cost  $\ell_B(\boldsymbol{\theta}_i)$  from Eq. (2) as a probabilistic inference problem: We model  $\boldsymbol{\theta}_i^*$  as a random variable, denoted  $\widehat{\boldsymbol{\theta}}_i^*$ , and compute the posterior distribution of  $\widehat{\boldsymbol{\theta}}_i^*$  conditioned on  $\ell_B(\boldsymbol{\theta}_i)$  via Bayes rule

$$p(\widehat{\boldsymbol{\theta}}_i^* | \ell_B(\boldsymbol{\theta}_i)) = \frac{p(\ell_B(\boldsymbol{\theta}_i) | \widehat{\boldsymbol{\theta}}_i^*) p(\widehat{\boldsymbol{\theta}}_i^*)}{p(\ell_B(\boldsymbol{\theta}_i))}. \quad (6)$$

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

$$p(\widehat{\boldsymbol{\theta}}_i^*) = \mathcal{N}(\widehat{\boldsymbol{\theta}}_i^*; \boldsymbol{\theta}_i, \mathbf{W}_i), \quad (7)$$

where  $\mathbf{W}_i \in \mathbb{R}^{N \times N}$  is an arbitrary symmetric positive definite covariance matrix, which is discussed further in Section 2.2. To develop the likelihood  $p(\ell_B(\boldsymbol{\theta}_i) | \widehat{\boldsymbol{\theta}}_i^*)$ , we start by rewriting Eq. (5) in terms of  $\boldsymbol{\theta}_i^*$  as

$$\begin{aligned} \bar{f}_i(\boldsymbol{\theta}_i)|_{\mathbf{d}=\mathbf{0}} &= \bar{f}_i(\boldsymbol{\theta}_i - \mathbf{d}_i^* + \mathbf{d}_i^*) \\ &= \bar{f}_i(\boldsymbol{\theta}_i + \boldsymbol{\theta}_i - \boldsymbol{\theta}_i^* + \mathbf{d}_i^*). \end{aligned} \quad (8)$$

Inserting this statement in Eq. (5) and recalling that  $\boldsymbol{\theta}_i^* - \boldsymbol{\theta}_i = \mathbf{d}_i^* = -\mathbf{B}_i^{-1}\nabla f_i$ , we obtain

$$\begin{aligned} \bar{f}_i(\boldsymbol{\theta}_i) &= \frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^*)^\top \mathbf{B}_i (\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^*) + f^*, \\ \text{and } \nabla \bar{f}_i &= \mathbf{B}_i (\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^*). \end{aligned} \quad (9)$$

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

This finding motivates us to express the noisy observation  $\ell_B(\boldsymbol{\theta}_i)$  in the *probabilistic* model as the following linear projection

$$\begin{aligned} \ell_B(\boldsymbol{\theta}_i) &= \frac{1}{2}\mathbf{g}_i^\top (\boldsymbol{\theta}_i - \widehat{\boldsymbol{\theta}}_i^*) + f^* + \epsilon_i, \\ \epsilon_i &\sim \mathcal{N}(0, R_i), \end{aligned} \quad (10)$$

or equivalently by the likelihood

$$p(\ell_B(\boldsymbol{\theta}_i) | \widehat{\boldsymbol{\theta}}_i^*) = \mathcal{N}\left(\ell_B(\boldsymbol{\theta}_i); \frac{1}{2}\mathbf{g}_i^\top (\boldsymbol{\theta}_i - \widehat{\boldsymbol{\theta}}_i^*) + f^*, R_i\right). \quad (11)$$

Here we use  $\mathbf{g}_i$  to denote a the gradient of the loss over the mini-batch and  $R_i$  is the observation noise due to subsampling. Under the Gaussian prior (7) and linear likelihood (11) there is a closed-form expression<sup>1</sup> for the posterior, stated in Eq. (6). The posterior mean will serve as our next estimate,  $\boldsymbol{\theta}_{i+1}$ , and is given by

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mathbf{W}_i \mathbf{g}_i \frac{2(\ell_B(\boldsymbol{\theta}_i) - f^*)}{\mathbf{g}_i^\top \mathbf{W}_i \mathbf{g}_i + R_i}. \quad (12)$$

This parameter update is of the same form as the generic update in Eq. (4) if  $\mathbf{v}_i = \mathbf{W}_i \mathbf{g}_i$ . 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.

### 2.2 CHOICE OF COVARIANCE

The update in Eq. (12) leaves the prior covariance matrix  $\mathbf{W}_i$  as a free parameter. To be a valid covariance, it must be symmetric and positive definite. For the batch gradient

<sup>1</sup>For  $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$  with  $\mathbf{x}, \boldsymbol{\mu} \in \mathbb{R}^N$ ,  $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$  and  $p(y | \mathbf{x}) = \mathcal{N}(y; \mathbf{A}\mathbf{x} + b, R)$  with  $\mathbf{A} \in \mathbb{R}^{1 \times N}$  and  $b \in \mathbb{R}$ ,  $R \in \mathbb{R}_+$ , it holds that  $p(\mathbf{x} | y) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu} + \frac{\boldsymbol{\Sigma}\mathbf{A}^\top(y - \mathbf{A}\boldsymbol{\mu} - b)}{\mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top + R}, \boldsymbol{\Sigma} - \frac{\boldsymbol{\Sigma}\mathbf{A}^\top\boldsymbol{\Sigma}}{\mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top + R})$ .

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

Table 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 2.3.  $\mathbf{G}_i$  for Adagrad and RMSprop are recursively defined as the quantity inside the parenthesis starting from  $\mathbf{G}_0 = 0$ . The diagonal matrix  $\mathbf{V}_i$  for Adam is the  $\mathbf{G}_i^{-1/2}$  of RMSprop scaled with additional bias correction.

$\mathbf{g}_i$  (i.e.  $\nabla \ell_B$ ), a step in direction  $\mathbf{v}_i = \mathbf{W}_i \mathbf{g}_i$  is a descent direction (on the batch), because  $-\mathbf{g}_i^\top \mathbf{v}_i < 0$ . To clarify the connection to optimization algorithms with a provided learning rate we will refer to  $\mathbf{W}_i$  as  $\eta \cdot \mathbf{W}_i$ , with  $\eta \in \mathbb{R}^+$ . For different choices of  $\mathbf{W}_i$ , Eq. (12) can be seen as a generalization of several existing optimization algorithms.

**SGD** The most straightforward connection is to SGD [Robbins and Monro, 1951]: If we consider  $\eta \cdot \mathbf{W}_i = \eta \cdot \mathbf{I}$ , Eq. (12) becomes

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \mathbf{g}_i \frac{2(\ell_B(\boldsymbol{\theta}_i) - f^*)}{\eta \|\mathbf{g}_i\|^2 + R_i}. \quad (13)$$

For  $R_i = 0$  this update recovers the Polyak step [Polyak, 1987, Loizou et al., 2020] 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,  $\ell_B \gtrsim f^*$ , 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.

**General Optimizer** There has been a surge of stochastic first-order optimization algorithms to tackle the requirements of machine learning and deep learning in particular, many of which employ an element-wise scaling to the batch gradient  $\mathbf{g}_i$ . These element-wise updates correspond to a diagonal matrix  $\mathbf{W}_i$  in Eq. (12) which is the definition of an axis-aligned Gaussian distribution for the prior. Tab. 1 summarizes a few popular diagonal first-order optimizers in the probabilistic view together with a novel interpretation of acceleration in the form of momentum.

The inference is not limited to a diagonal  $\mathbf{W}_i$ , 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  $\mathbf{W}_i^{-1}$  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 Newton, Gauss-Newton or Natural Gradient Descent, respectively.

In addition to the posterior mean one could also consider the posterior covariance on  $\hat{\boldsymbol{\theta}}_i^*$ . 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.

### 2.3 ACCELERATED GRADIENT UPDATES

Several popular optimization algorithms employ an exponential averaging of gradients in the form of momentum [Polyak, 1987, Sutskever et al., 2013, Kingma and Ba, 2014] 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  $\mathbf{g}_i$  with  $\mathbf{m}_i$  in the derivations of Section. 2. The second way that we opted for is to retain the batch gradient but adjust the covariance with a rank one update.

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

$$\begin{aligned} \mathbf{m}_i &= \beta \cdot \mathbf{m}_{i-1} + \mathbf{g}_i, \\ \boldsymbol{\theta}_i &= \boldsymbol{\theta}_{i-1} - \eta \cdot \mathbf{m}_i, \end{aligned}$$

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

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

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

$$\begin{aligned} \mathbf{V}_i &= \gamma_i \cdot (\beta_2 \cdot \mathbf{G}_{i-1} + (1 - \beta_2) \cdot \text{diag}(\mathbf{g}_i \mathbf{g}_i^\top))^{-1/2}, \\ \gamma_i &= \sqrt{(1 - \beta_2^i)/(1 - \beta_1^i)}, \end{aligned}$$

and the exponential average  $\mathbf{m}_i = \beta_1 \mathbf{m}_{i-1} + (1 - \beta_1) \mathbf{g}_i$  as

$$\begin{aligned} \eta \cdot \mathbf{W}_i \mathbf{g}_i &= \eta \cdot \left( (1 - \beta_1) \mathbf{V}_i + \tilde{\beta}_1 \mathbf{V}_i \mathbf{m}_{i-1} \mathbf{m}_{i-1}^\top \mathbf{V}_i \right) \mathbf{g}_i, \\ \tilde{\beta}_1 &= \frac{\beta_1}{\langle \mathbf{V}_i \mathbf{m}_{i-1}, \mathbf{g}_i \rangle}. \end{aligned}$$

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  $\langle \mathbf{m}_{i-1}, \mathbf{g}_i \rangle < 0$  and violates the positive definiteness if  $\beta \langle \mathbf{m}_{i-1}, \mathbf{g}_i \rangle < -\langle \mathbf{g}_i, \mathbf{g}_i \rangle$ . This occurs when  $\mathbf{m}_{i-1}$  is pointing significantly uphill and the situation is more likely to arise for high values of  $\beta$ . The conditions are analogous for Adam.

### 3 ALGORITHM

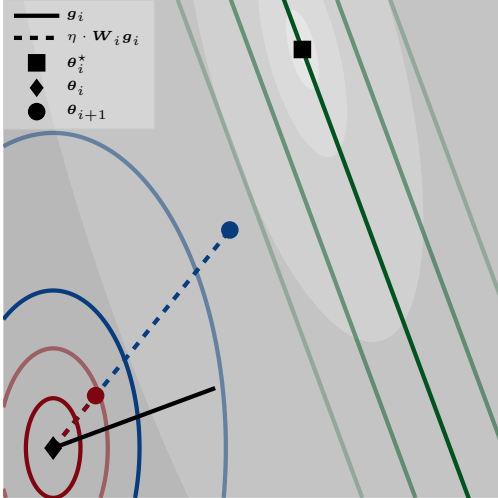


Figure 1: Illustration of the probabilistic inference scheme. The gray contour levels describe the local quadratic model  $\bar{f}_i(\theta)$ . The dark green line comprises all possible values of  $\hat{\theta}_i^*$  (orthogonal to  $\mathbf{g}_i$ ) that satisfy Eq. (10) (in the noise-free case); this includes the true value  $\theta_i^*$ . The lighter green levels are multiples of the standard deviation  $\sqrt{R_i}$  representing the uncertainty in Eq. (10). The red and blue ellipses illustrate two axis-aligned Gaussian distributions (Eq. (7)) centered at  $\theta_i$ , corresponding to different scaling of the covariance  $\mathbf{W}_i$ . The circles indicate the posterior mean ( $\theta_{i+1}$  in Eq. (12)) for each distribution. The blue distribution has a larger variance relative to  $R_i$  leading to a larger step towards the solution compared to the red.

The update in Eq. (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  $\eta \cdot \mathbf{W}_i$  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.

### 3.1 PARAMETERS

Depending on the problem, the update scheme requires estimation of up to three parameters: The scale of the prior variance  $\eta$ , the lower bound  $f^*$  and the observation variance  $R_i$ . With the probabilistic motivation developed in Section 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  $\theta_{i+1}$  (Eq. 12) can arise from different constellations of these three numbers. In traditional optimization, uncertainty of the observation is typically not considered, which is why  $R_i$  does not appear in the standard Polyak step.

**Uncertainty  $R_i$**  The most straightforward parameter to estimate is the observation variance  $R_i$  in Eq. (12). In the case of minibatching it is possible to estimate the uncertainty of the full loss, as outlined via the CLT argument in Section 1. For general functions the situation is more complicated and since the inclusion of  $R_i$  corresponds to a reduction in step length, the same effect can be achieved by omitting  $R_i$  and instead decrease the difference in the numerator.

**Scale  $\eta$**  The second influential parameter is  $\eta$ , a scalar multiplication of the prior covariance corresponding to the learning rate of the optimizer. For the general case of Eq. (12) with  $R_i > 0$ , the ratio between  $R_i$  and  $\eta \cdot \mathbf{g}_i^T \mathbf{W}_i \mathbf{g}_i$  becomes important for the overall step length and hence the convergence of the algorithm. This behavior is visualized in Fig. 1 where the variance of the red distribution is low compared to  $R_i$ , leading to small updates. 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  $\eta$  during the optimization, once more parameters can be estimated, could lead to improved performance. If instead  $R_i = 0$  and  $f^*$  in Eq. (12) is known, then the update will not depend on the scale  $\eta$ .

**Lower bound  $f^*$**  The final parameter  $f^*$  is also the most important in terms of performance and stability. For a known  $f^*$  and  $R_i = 0$ , 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 [Polyak, 1987, Vaswani et al., 2020]. If  $f^*$  is not known and set too large, then the optimizer will converge to parameters for which  $f(\theta) = f^*$  and not the actual minimum. If instead  $f^*$  is not known and estimated below the minimum, then the Polyak step will try to reach function values below the minimum. This is problematic in flat regions, resulting in steps that are too large which can undo a lot of progress (see Appendix A). To combat this behavior, a maximum step length is often introduced as a problem dependent hyperparameter [Berrada et al., 2019, Loizou et al., 2020]. The same authors showed that for Ma-

chine learning problems with overparameterized models that fulfill *interpolation* (models which can achieve a training loss close to 0 for all training samples simultaneously) it is possible to use  $f^* = 0$  as a lower bound for the empirical risk minimization for fast convergence.

**Algorithm 1** Step size adaptation for a provided function evaluator  $F(\theta)$ , gradient estimator  $G(\theta)$ , and search direction selector  $W(g)$  defined by the underlying optimizer.

---

```

1: procedure LR-ADAPT( $\theta_1, \eta, F(\theta), G(\theta), W(g), f^*, R_i = 0$ )
2:   for  $i = 1 \dots$  do
3:      $f_i \leftarrow F(\theta_i)$             $\triangleright$  Evaluate function
4:      $g_i \leftarrow G(\theta_i)$         $\triangleright$  Obtain gradient
5:      $v_i \leftarrow \eta \cdot W(g_i)$      $\triangleright v_i = \eta \cdot W_i g_i$ 
6:      $\phi_i \leftarrow g_i^\top v_i$ 
7:
8:     if  $f^*$  is available then
9:        $\Delta f \leftarrow f_i - f^*$ 
10:    else
11:       $\Delta f \leftarrow f_i - \phi_i/2$ 
12:
13:     $\theta_{i+1} \leftarrow \theta_i - v_i \cdot 2 \cdot \frac{\Delta f}{\phi_i + R_i}$ 
14:
15:     $f_+ = F(\theta_{i+1})$             $\triangleright$  Re-evaluate same batch
16:
17:    if  $(f_i - f_+)/(\phi/2) > 4/3$  then
18:       $\eta \leftarrow 1.2 \cdot \eta$         $\triangleright \eta$  too small  $\alpha_\uparrow$ 
19:    else if  $(f_i - f_+)/(\phi/2) < 3/4$  then
20:       $\eta \leftarrow 1/2 \cdot \eta$        $\triangleright \eta$  too large  $\alpha_\downarrow$ 

```

---

### 3.2 IMPLEMENTATION

The previous section showed that the relatively simple update of Eq. (12) requires careful consideration, see Appendix A for an example. Here we will outline the steps taken in Alg. 1 to address this. Once a valid optimizer has been selected all the quantities up to and including line 7 are accounted for. A step of our algorithm then requests two parameters: a lower bound  $f^*$  and an uncertainty estimate  $R_i$  (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  $f(\theta_i)$ , gradient  $g_i$  and a spd matrix  $\eta_i \cdot W_i$  constructs a local surrogate

$$\phi_{W_i}(d) = \phi_{\min} + \frac{1}{2} (d + \eta_i W_i g_i)^\top (\eta_i W_i)^{-1} (d + \eta_i W_i g_i). \quad (14)$$

The parameters are chosen such that  $\phi_{W_i}(\mathbf{0}) = f(\theta)$  and  $\nabla_d \phi|_{d=0} = g_i$ . The minimum of this surrogate occurs at

the step  $d = -\eta_i W_i g_i$  corresponding to a decrease of  $\phi(0) - \phi(-\eta_i W_i g_i) = \eta_i \cdot g_i^\top W_i g_i / 2$ . This lower bound in Eq. (12) amounts to a standard step of the underlying optimization algorithm if  $R_i = 0$ , 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  $\phi_i - \phi_{i+1} = \eta_i \cdot g_i^\top W_i g_i / 2 \approx f(\theta_i) - f(\theta_{i+1})$ . The decision of which lower bound to use occurs in lines 8 to 12 of Alg. 1 and is followed by the update in Eq. (12).

In line 15 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  $v_i$ . If the ratio deviates from 1, we adapt the learning rate by multiplicatively increasing or decreasing  $\eta$  with  $\alpha_\uparrow = 1.2$  and  $\alpha_\downarrow = 1/2$ , inspired by the updates of Rprop [Riedmiller and Braun, 1992]. Similar ideas are also employed in trust-region methods [Nocedal and Wright, 2006, Ch. 4] to adapt the size of the trust region. One could simply choose a new  $\eta$  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. 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 approaches a value that is suitable for the full-batch function.

### 3.3 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 the next iteration. Apart from the re-evaluation there are two non-scalar operations involved in each step: finding the step direction  $v_i = \eta \cdot W_i g_i$ , and computing the inner product  $v_i^\top g_i$  (i.e.  $\eta \cdot g_i^\top W_i g_i$ ) in Eq. (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  $v_i$  is obtained, which is of similar complexity to the first-order optimizers in Tab. 1. Optimizers in Pytorch usually compute the update  $v_i$  per-parameter and apply the update immediately to save memory. In order to keep the implementation as general as possible for the identified algorithms, we explicitly store the vector  $v_i$  for the inner product. This storage can be avoided but would require implementing a new version of each optimizer.

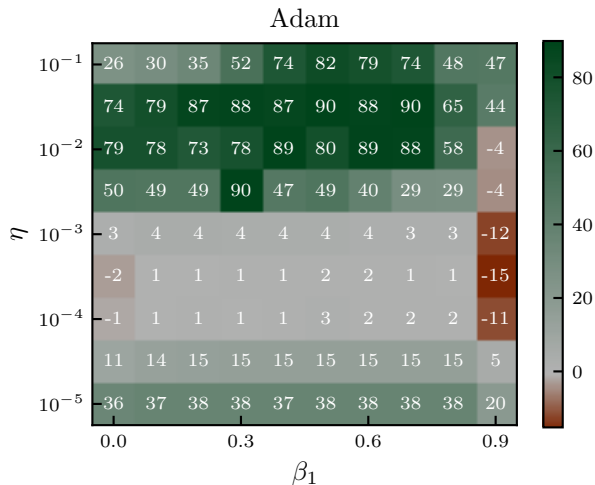


Figure 2: Difference in achieved training accuracy between proposed adaptation and fixed learning rate version of Adam for different initial learning rates ( $\eta$ ) and momentum ( $\beta_1$ ). Green color means adaptation improved and red signifies worse accuracy after 50 epochs of training on an own implementation of the test problem cifar10\_3c3d. 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  $\beta_1$ , c.f. Section 2.3.

## 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 [Schneider et al., 2019].<sup>2</sup> We implemented our method in Pytorch [Paszke et al., 2019] ver. 1.4 as a wrapper to the implemented optimizers listed in Tab. 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 [Baydin et al., 2018] and L4 [Rolinek and Martius, 2018] where applicable. In the absence of  $f^*$  we found no significant difference in results by including an estimate of the noise variance  $R_i$  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. For L4 we varied  $\alpha_{L4}$  which scales the numerator of Eq.(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 report the results in terms of training accuracy since it is constrained to  $[0, 1]$ . All of the results can be seen in Fig. 3 and additional metrics and optimizers can be found in the supplementary material.

<sup>2</sup><https://deepobs.github.io>

Apart from the momentum term of Adam and SGD with momentum for our adaptation (further discussed below), all other hyperparameters were kept at the Pytorch default values.

### 4.1 (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. The first three rows of Fig. 3 show convergence results for models using logistic regression, a 4 layered multilayered perceptron and an artificial neural network with 2 convolutional layers followed by 2 dense layers for the classification.

### 4.2 CIFAR-10

The network used for the CIFAR-10 dataset [Krizhevsky, 2009] 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. The same architecture was used to investigate how momentum affects our adaptation in deep learning. A typical result can be seen in Fig. 2 illustrating a sweep across learning rates ( $\eta$ ) and momentum ( $\beta_1$ ) for Adam. The figure shows that the performance tend to deteriorate for large values of  $\beta_1$ . A similar sweep took place for SGD with momentum to settle on a momentum  $\beta$  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.

### 4.3 SVHN

The SVHN dataset [Netzer et al., 2011] contains more than 600 000 images of house numbers seen from the street. One deepOBS architecture<sup>2</sup> used for this experiment is a *wide resnet* [Zagoruyko and Komodakis, 2016], which is an extension of the *deep resnet* [He et al., 2016]. 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  $5 \cdot 10^{-4}$ .

## 4.4 DISCUSSION

The results presented in Fig. 3 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



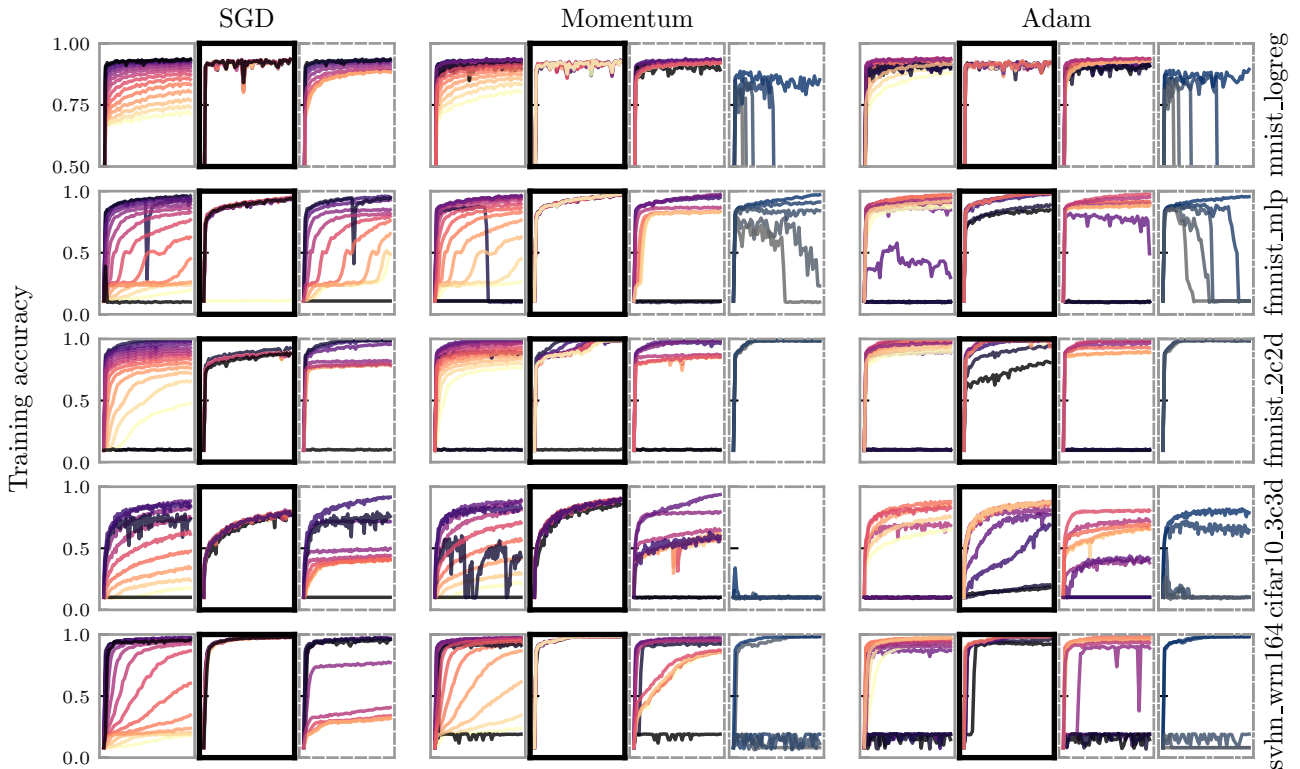


Figure 3: Training accuracy (higher is better) per epoch for different optimizers, learning rate adaptations and benchmark problems from DeepOBS. Each row shows the training accuracy for one test problem identified by a dataset and model description and each columngroup shows a family of optimizers. 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  $\alpha_{L4}$  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.

SGD which in some cases falls behind due to the higher variance of  $\eta \cdot \mathbf{g}_i^T \mathbf{W}_i \mathbf{g}_i$  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 (an example is Adam for the fmnist tasks). In these cases the learning rates are several orders of magnitude larger than the optimal fixed learning rate and neither adaptation converges.

Hypergradient descent often shows improvements over the corresponding fixed learning rate version but sometimes gets stuck for too small learning rates (see fmnist experiments) 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. Our method instead updates the learning rate based on a dimensionless quantity making it less sensitive to variations in the network.

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. 3 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 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  $2^i$ th epoch for  $i = 0, 1, \dots$ . This allows significant adaptation in the beginning to get the scale right and less frequently during later stages of training, see Appendix B.1 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.

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 learning rate of  $10^{-4} - 10^{-3}$  and let the adaptation accelerate.

## 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 [Sutskever et al., 2013], adapting the step-direction [Duchi et al., 2011, Zeiler, 2012, Schaul et al., 2013, Dauphin et al., 2015] or combinations thereof [Kingma and Ba, 2014] have been proposed as substitutes [see Ruder, 2016, for an overview]. 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 [Goodfellow et al., 2016]. Some recent ideas to reduce this influence are to include the learning rate as an additional parameter that can be optimized with backpropagation [Baydin et al., 2018, cf. results in Fig. 3] or to train another model to predict the next step [Andrychowicz et al., 2016].

In traditional optimization the learning rate problem is usually avoided by use of a line search routine, with new iterates chosen to satisfy conditions that ensure suitable convergence [Armijo, 1966, Nocedal and Wright, 2006]. Stochastic versions of these line searches were proposed by Mahsereci and Hennig [2017] and Vaswani et al. [2019]. 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.

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* [Loizou et al., 2020]. Loizou et al. [2020] used the Polyak step together with SGD and proved a convergence rate for the algorithm. Around the same time Berrada et al. [2019] 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.

The L4 optimizer of Rolinek and Martius [2018] 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. 3. 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.

Another similar line of work is that of Vaswani et al. [2020] who extend the line search of Vaswani et al. [2019] and the Polyak step of Loizou et al. [2020] for problems that satisfy interpolation. The main contribution was to use a general metric to recover additional optimization algorithms (the diagonal versions in Tab. 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  $\geq 1$  additional function evaluations per batch whereas ours rely on a single re-evaluation.

The derivations of Sec. 2 are reminiscent of probabilistic linear algebra routines with additional noise [Hennig et al., 2015, de Roos and Hennig, 2019, Cockayne et al., 2019]. 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.

## 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 several 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.



## References

- Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems (NIPS)*, pages 3981–3989, 2016.
- Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.
- Hilal Asi and John C Duchi. The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*, 116(46):22924–22930, 2019.
- Atilim Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.
- Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations (ICLR)*, 2018.
- Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Training neural networks for and by interpolation. *arXiv preprint arXiv:1906.05661*, 2019.
- Jon Cockayne, Chris J Oates, Ilse CF Ipsen, Mark Girolami, et al. A Bayesian conjugate gradient method. *Bayesian Analysis*, 14, 2019.
- Yann N Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. In *ICML workshop on Deep learning*, 2015.
- Filip de Roos and Philipp Hennig. Active probabilistic inference on matrices for pre-conditioning in stochastic optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*. PMLR, 2019.
- John Duchi, Elad Hazan, and Yoram Singer. Adaptive sub-gradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*. PMLR, 2016.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, 2015.
- Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2014.
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto, 2009.
- Nicolas Loizou, Sharan Vaswani, Issam Laradji, and Simon Lacoste-Julien. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. *arXiv preprint arXiv:2002.10542*, 2020.
- Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *The Journal of Machine Learning Research*, 18(1):4262–4320, 2017.
- Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bisacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- Adam Paszke, Sam Gross, Francisco Massa, and et. al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.
- Boris Polyak. *Introduction to Optimization*. 1987.
- Martin Riedmiller and Heinrich Braun. Rprop—a fast adaptive learning algorithm. In *Proc. of ISICIS VII*. Citeseer, 1992.
- Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, 22(3):400–407, 1951.
- Michal Rolínek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6433–6443, 2018.
- H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3, 1960.

- Sebastian Ruder. An overview of gradient descent optimization algorithms. Technical report, arXiv:1609.04747, 2016.
- Tom Schaul, Sixn Zhang, and Yann Lecun. No more pesky learning rates. In *International Conference on Machine Learning (ICML)*, 2013.
- Frank Schneider, Lukas Balles, and Philipp Hennig. Deep-OBS: A deep learning optimizer benchmark suite. In *International Conference on Learning Representations (ICLR)*, May 2019.
- Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, pages 1139–1147, 2013.
- Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pages 3732–3745, 2019.
- Sharan Vaswani, Frederik Kunstner, Issam Laradji, Si Yi Meng, Mark Schmidt, and Simon Lacoste-Julien. Adaptive gradient methods converge faster with overparameterization (and you can do a line-search). *arXiv preprint arXiv:2006.06835*, 2020.
- Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the noisy gradient descent that generalizes as sgd. In *International Conference on Machine Learning*. PMLR, 2020.
- Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL <https://dx.doi.org/10.5244/C.30.87>.
- Matthew D Zeiler. Adadelata: An adaptive learning rate method. Technical report, arXiv:1212.5701, 2012.

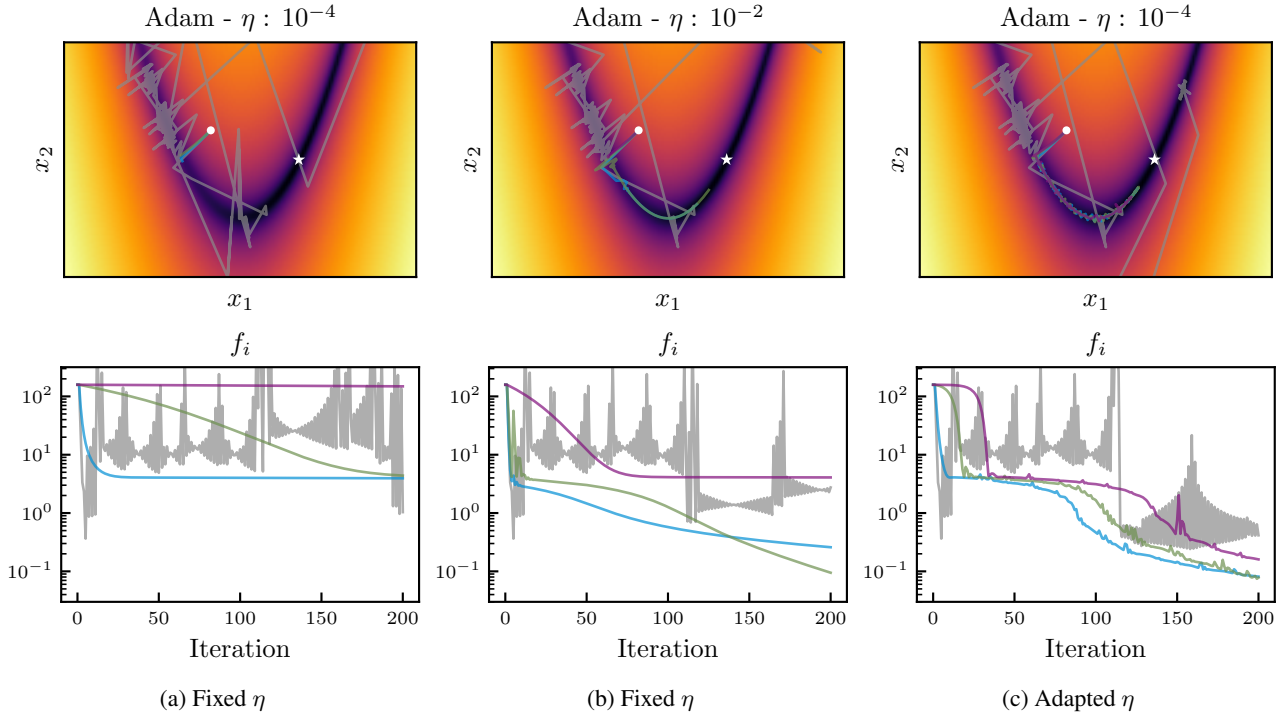


Figure 4: Influence of variance (learning rate) adaption for inference step on 2-d Rosenbrock (1960) function. Each figure shows the standard step of the optimizer (—), the inference with  $R_i = 0$ , i.e., Polyak step (—), inference with fixed  $R_i = 0.1$  (—) and the inference with adaptive  $R_i = 0.05f_i$  (—). 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  $R_i = 0$  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.

## A PROBABILISTIC MODEL

The main parameters of the probabilistic model ( $f^*$ ,  $R_i$ ,  $\eta$ ) have a complicated interplay which affects the *modus operandi* of the algorithm, depending on available information. Fig. 4 highlights some of the difficulties related to setting the parameters  $\eta$ ,  $f^*$  and  $R_i$  in the probabilistic update. If the global lower bound  $f^*$  is known (0 in this case) it is still not straightforward to properly set  $\eta$  w.r.t.  $R_i$ . Adapting the variance on-the-go alleviates this problem when  $R_i > 0$  and when the standard step of the underlying optimizer is used. One could just as easily consider an algorithm where  $f^*$  is estimated and provided externally instead of  $R_i$ .

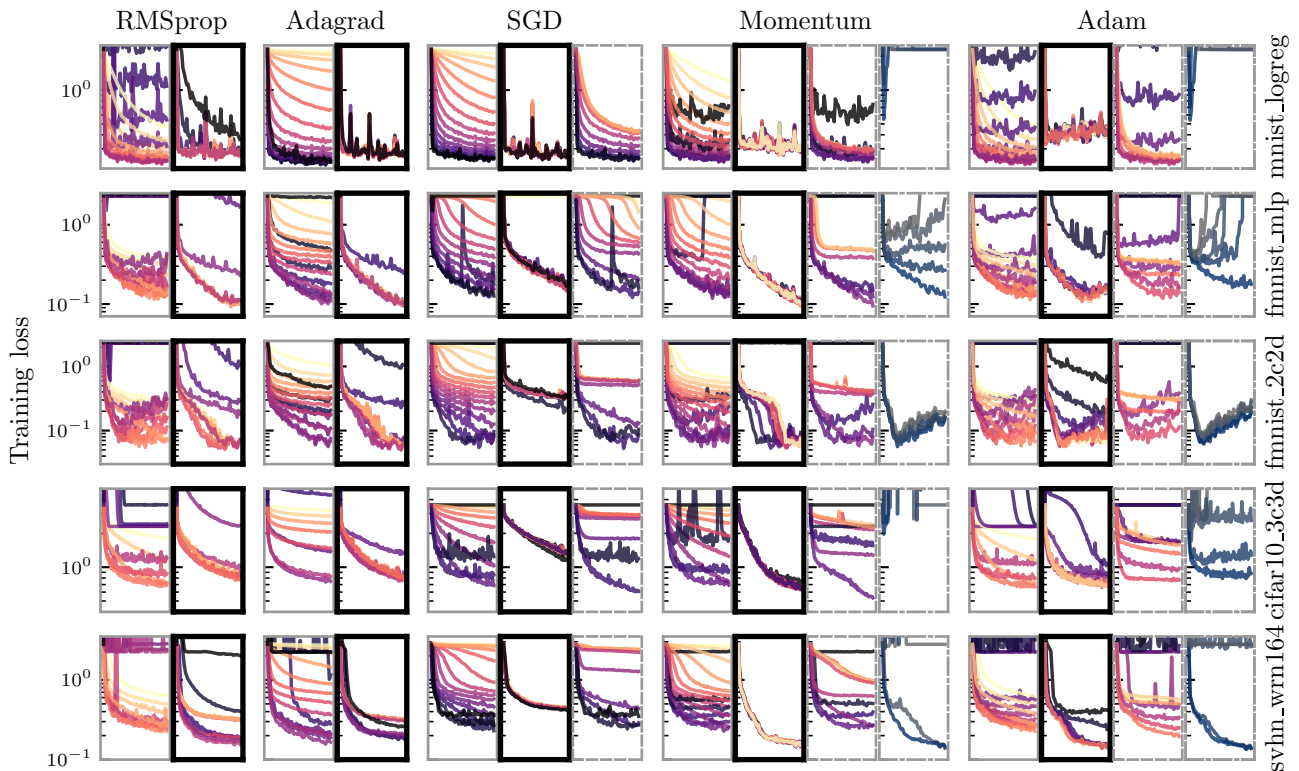


Figure 5: Training loss for the experiments presented in the main part of the paper. For details see Sec. B.

## B ADDITIONAL EXPERIMENTS

The experiments are presented in the same way as the results of the paper. Meaning that each figure shows a loss metric for different test problems (rows) and optimizers (columns). The leftmost graph in each group (thin gray border) has a fixed learning rate and next to it (thick black border) is the proposed adaptation. A dashed border indicates results for hypergradient descent Baydin et al. [2017] and the dash-dotted are results for L4 Rolinek and Martius [2018]. Each graph contains experiments with initial learning rates indicated by colors in the range  $10^{-5}$  (—) through  $10^{-3}$  (—) to  $10^0$  (—). In the case of L4 the learning is replaced with  $\alpha_{L4}$  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 of the optimizers 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. Here we additionally include the results of RMSprop and Adagrad in the comparison. The training loss of the experiments in the main paper are available in Fig. 5 and a zoomed-in version of the test accuracy can be seen in Fig. 6.

### B.1 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. 7 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.

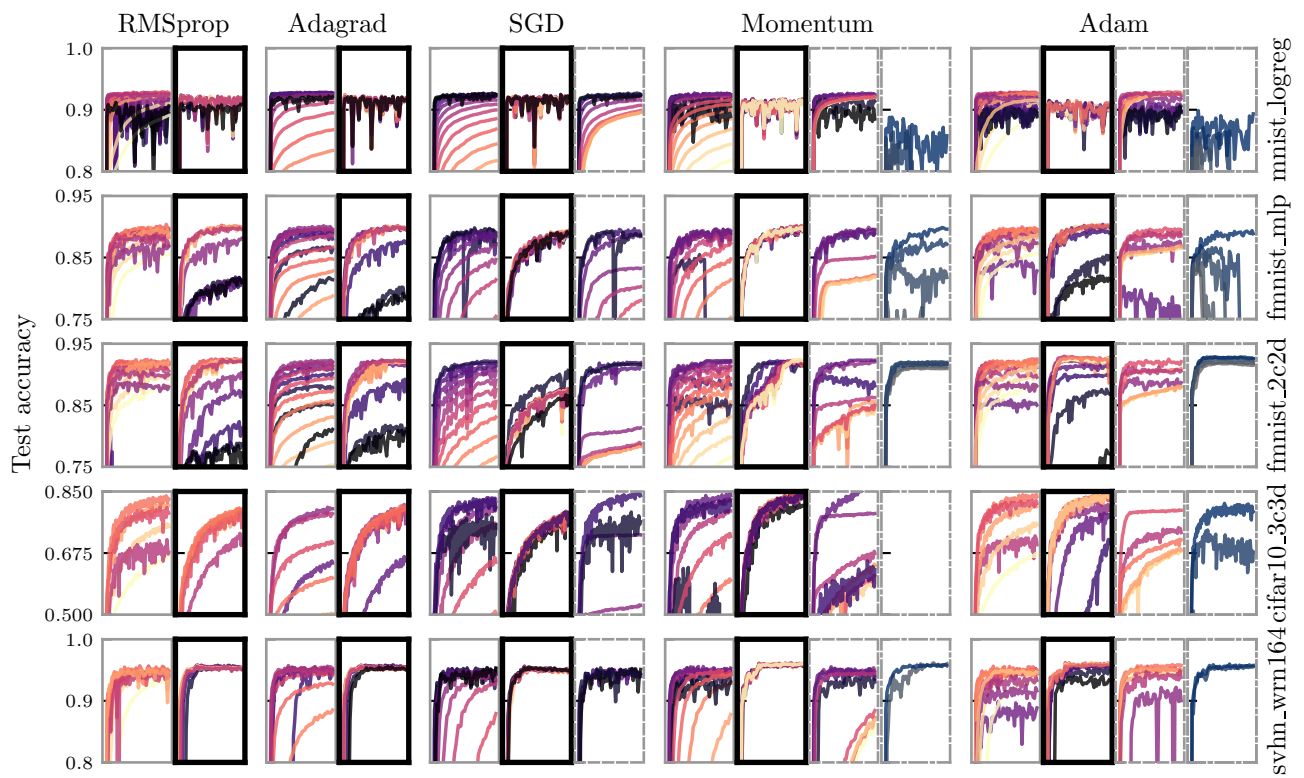


Figure 6: Test accuracy for the experiments presented in the main part of the paper. For details see Sec. B.

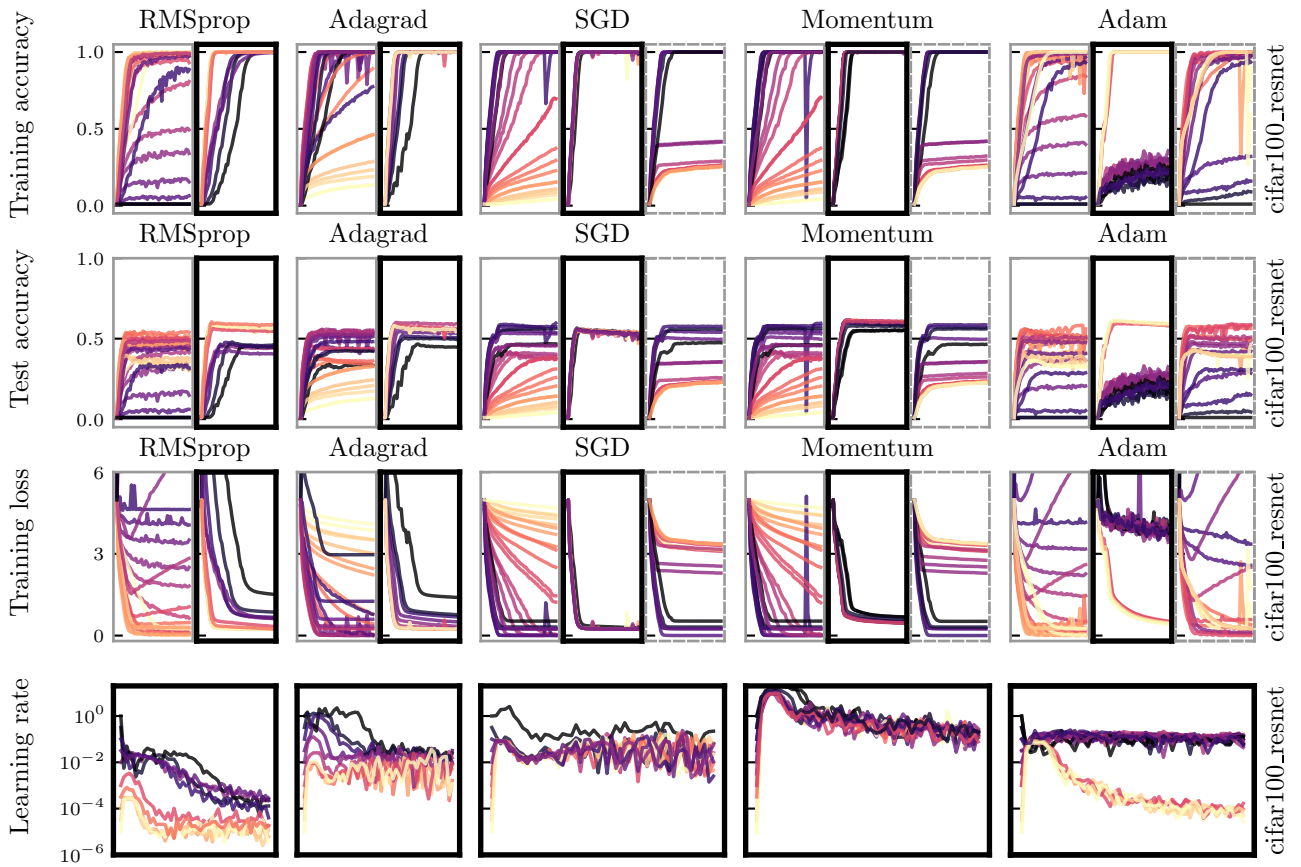


Figure 7: Results for a model trained on CIFAR-100 for 50 epochs. All the settings and limits are the same as the results from DeepOBS. The last row shows the learning rate that was used at the end of each epoch for the proposed learning rate adaptation.