Empirics-based Line Searches for Deep Learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Maximus Mutschler

aus Waldshut-Tiengen

Tübingen

2022

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

Tag der mündlichen Qualifikation:	16.02.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

To all humankind and AI-kind

Abstract

This dissertation takes an empirically based perspective on optimization in deep learning. It is motivated by the lack of empirical understanding of the loss landscape's properties for typical deep learning tasks and a lack of understanding of why and how optimization approaches work for such tasks. We solidified the empirical understanding of stochastic loss landscapes to bring color to these white areas on the scientific map with empiric observations. Based on these observations, we introduce understandable line search approaches that compete with and, in many cases outperform, state-of-the-art line search approaches introduced for the deep learning field.

This work includes a comprehensive introduction to optimization focusing on line searches in the deep learning field. Based on and guided by this introduction, empirical observations of typical image-classification benchmark tasks' loss landscapes are presented. Further, observations of how optimizers perform and move on such loss landscapes are given. From these observations, the line search approaches Parabolic Approximation Line Search (PAL) and Large Batch Parabolic Approximation Line Search (LABPAL) are derived. In particular, the latter method outperforms all competing line searches in this field in most cases. Furthermore, these observations reveal that well-tuned Stochastic Gradient Descent is already well approximating an almost exact line search, which in parts explains why it is so hard to beat.

Given the empirical observations made, it is straightforward to comprehend why and how our optimization approaches work. This contrasts the methodology of many optimization papers in this field which builds upon non-empirically justified theoretical assumptions. Consequently, a general contribution of this work is that it justifies and demonstrates the importance of empirical work in this rather theoretical field.

Kurzfassung

Diese Dissertation nimmt eine empirisch basierte Perspektive auf Optimierung im Feld Deep Learning ein. Sie ist durch das fehlende empirische Verständnis von Eigenschaften der Fehlerlandschaften für typische Deep-Learning-Aufgaben und das mangelnde Verständnis, warum und wie Optimierungsansätze für solche Aufgaben funktionieren, motiviert. Um diese weißen Flecken auf der wissenschaftlichen Landkarte mit empirischen Beobachtungen zu beleuchten, haben wir das empirische Verständnis von stochastischen Fehlerlandschaften gefestigt. Auf der Grundlage dieser Beobachtungen stellen wir interpretierbare Liniensuchverfahren vor, die mit den modernsten Liniensuchverfahren des Bereichs Deep Learning konkurrieren und diese in vielen Fällen sogar übertreffen.

Diese Arbeit enthält eine umfassende Einführung in die Optimierung mit Schwerpunkt auf Liniensuchverfahren im Bereich des Deep Learning. Basierend auf dieser Einführung werden empirische Beobachtungen der Verlustlandschaften typischer Bildklassifizierungs-Benchmark-Aufgaben und Beobachtungen, wie Optimierer sich bei diesen Aufgaben verhalten, vorgestellt. Aus diesen Beobachtungen werden die Liniensuchverfahren Parabolic Approximation Line Search (PAL) und Large Batch Parabolic Approximation Line Search (LABPAL) abgeleitet. Insbesondere die letztgenannte Methode übertrifft in den meisten Fällen alle konkurrierenden Liniensuchverfahren in diesem Bereich. Darüber hinaus zeigen diese Beobachtungen, dass ein gut parametrisierter stochastischer Gradientenabstieg bereits ein nahezu exaktes Liniensuchverfahren approximiert, was zu Teilen erklärt, warum dieser so schwer zu übertreffen ist.

Angesichts der empirischen Beobachtungen ist es einfach zu verstehen, warum und weshalb unsere Optimierungsansätze funktionieren. Dies steht im Gegensatz zur Methodik vieler Optimierungsarbeiten in diesem Bereich, die auf nicht-empirisch begründeten theoretischen Annahmen aufbauen. Folglich besteht ein allgemeiner Beitrag dieser Arbeit darin, dass sie die Bedeutung von empirischen Beobachtungen in diesem eher theoretischen Bereich hervorhebt und demonstriert.

Contents

Chapter 1

Introduction

1.1 Motivation

Optimization is an omnipresent and intrinsic mechanism of nature. In the course of evolution, the intelligence, efficiency, and effectiveness of life forms have improved step by step. As a result, today's life forms can quickly adapt to the environment through innate, highly optimized, and self-learning brains or, put simply, information processing units. The human brain, in particular, is so powerful that up today, the field of artificial intelligence and information processing has not yet been able to design an algorithm that is as intelligent as a human across tasks. The field catches up, however, and in recent decades increasingly powerful artificial intelligence algorithms have been designed that surpass humans at specific tasks: for example, at Chess (Campbell *et al.*, 2002; Silver *et al.*, 2017), Go (Silver *et al.*, 2016), and Atari games (Espeholt *et al.*, 2018). In addition, it begins to challenge humans in real-world applications, such as driving cars (Yurtsever *et al.*, 2020), playing table tennis (Tebbe *et al.*, 2021), creating realistic images of people (Karras *et al.*, 2021), or pattern recognition (Dodge and Karam, 2017). The current flagships of artificial intelligence algorithms are deep neural networks (see Goodfellow *et al.* (2016)), which can be trained on a task quickly, even faster than a human, and significantly faster than using evolution.

At the very heart of these achievements is optimization. Specifically, training a neural network means numerically solving an extraordinarily high-dimensional, non-convex, stochastic optimization problem. The latter is based on an approximately known loss function used to search for a deep neural network in a specific function-hypothesis space. Solving such optimization problems is mainly guided by intuition-based heuristics in practice and relatively strong assumptions in theory. For example, in most cases, it is unknown why commonly used hyperparameters work well in practice (Gencoglu *et al.*, 2019); if theoretical results support these hyperparameters, they are often based on simplifying assumptions such as a convex loss function. This lack of knowledge of why and how specific hyperparameters or optimization routines work originates from a vague understanding of typical properties of loss functions. In this work, we first empirically uncover properties of the

loss function of image classification tasks that help to explain why and how specific optimization approaches work; or in other terms, why and how neural networks learn. Based on this increased understanding, we then exploit these properties of loss-functions to design optimization methods for the deep learning field. We focus on line search-based approaches because, first, they are closely related to our empirical observations and, second, although solved in the deterministic scenario (Nocedal and Wright, 2006, §3), they are still an unsolved field in the stochastic deep learning scenario. Furthermore, line search methods make the tuning of the learning rate redundant. The latter is a sensitive hyperparameter that often has to be tediously adjusted by hand. All in all, this work is motivated by the long-term goal of making optimization for deep neural networks faster, better, more robust, and most importantly for science, more explainable. Working towards this goal will pave the path towards more intelligent algorithms that could challenge human intelligence in even more domains.

1.2 Contribution & Outline

This dissertation improves the empirical understanding of deep learning tasks' loss landscape. We exploit this new understanding to explain, design, and improve optimization approaches in the stochastic line search subfield of optimization. Thereby, our focus is on image classification tasks as they are standard benchmarks for optimization in deep learning.

This work is mainly based upon three peer-reviewed and published conference or workshop papers:

1. Mutschler, M. and Zell, A. (2020). **Parabolic Approximation Line Search for DNNs.** *Advances in Neural Information Processing Systems, 33, 5405-5416 (NeurIPS)*

2. Mutschler, M. and Zell, A. (2021). **Empirically explaining SGD from a Line Search Perspective.** *International Conference on Artificial Neural Networks (pp. 459-471). Springer, Cham. (ICANN)*

3. Mutschler, M., Laube K., and Zell, A. (2021). **Using a one dimensional parabolic model of the full-batch loss to estimate learning rates during training.** *13th International Workshop on Optimization for Machine Learning (NeurIPS Optimization Workshop)*

This work is further structured as follows:

Chapter 2 **Introduction to Optimization and Line Searches in Deep Learning:** We begin with a comprehensive introduction to optimization in deep learning. This, inter alia, includes explaining empirical risk minimization (Section 2.1), surrogate losses (Section 2.2), neural networks (Section 2.3), line searches in the deterministic and stochastic scenario (Section 2.6), the simple loss landscape (Section 2.9) and finally, the relationship between batch sizes and learning rates (Section 2.10).

Chapter 3 **Experimental platform:** Since our empirical evaluations are computationally expensive, we introduce the experimental platform we used and designed: the TCML-Cluster. It has been essential for this work and a part of the project that funded this research.

Chapter 4 **Parabolic approximation line search:** This chapter shows that the mini-batch loss along the negative gradient direction tends to have a locally parabolic shape for typical image classification tasks. We further demonstrate, that the minimum of the mini-batch loss along such a line estimated by a parabolic approximation is a good estimator for the minimum of the full-batch loss along the same line. The resulting optimizer Parabolic Approximation Line search (PAL) challenges other line search approaches across base-line models and datasets. This chapter is based on (Mutschler and Zell, 2020a).

Chapter 5 **Empirically explaining SGD from a line search perspective:** Here, we analyze the shape of the full-batch loss and mini-batch losses along lines in negative gradient directions encountered during a Stochastic Gradient Descent (SGD) training. In addition, we assess how optimal SGD's update steps are and how the batch size affects the optimal learning rate. We made the following core observations:

- The full-batch loss along lines exhibits an almost parabolic shape locally.
- Exact line searches on the mini-batch loss perform poorly.
- SGD performs almost exact line searches on the full-batch loss.
- From a global perspective, a step size larger than the step to the minimum of the full-batch loss along a line performs better, although it yields less improvement locally.

This chapter is based on (Mutschler and Zell, 2021).

Chapter 6 **Large batch parabolic approximation line search:** We exploit the observations made in Chapter 4 and 5 to build a robust line search approach called Large Batch Parabolic Approximation Line Search (LABPAL), which efficiently approximates the full-batch loss. It mostly outperforms state-of-the-art line search approaches introduced for deep learning and challenges SGD tuned with a piece-wise constant learning rate schedule. This chapter is based on (Mutschler *et al.*, 2021).

Chapter 7 Finally, we discuss and summarize our results and derive further interesting research questions that might take this field another step forward.

Chapter 2

Introduction to Optimization and Line Searches in Deep Learning

This chapter provides an introduction to empirical risk minimization (Section 2.1), surrogate losses (Section 2.2), neural networks (Section 2.3), line searches in the deterministic and stochastic scenario (Sections 2.5, 2.6), relevant optimization methods (Sections 2.4, 2.7, 2.8), the simple loss landscape (Section 2.9), and finally, the relationship between batch sizes and learning rates (Section 2.10). We conclude with a controversial high-level optimizer and a critical reflection on this field in Section 2.11.

2.1 Empirical Risk Minimization with Relations to Deep Learning

The following is based on (Shalev-Shwartz and Ben-David, 2014) providing a comprehensive introduction to empirical risk minimization (ERM), which we have extended to include the relations to deep learning. ERM is the underlying framework for supervised machine learning and deep learning tasks:

Let the **training set** of size m be $\mathbb{T} := ((x_1, y_1), \ldots, (x_m, y_m)) \subset \mathbb{X} \times \mathbb{Y}$, where $\mathbb{X}$ is the input set and $\mathbb{Y}$ the label set. A learning algorithm receives a training set $\mathbb{T}$ sampled from an unknown data-generating distribution $\mathcal{D}$ as input. This algorithm aims to find a predictor $h_{\mathbb{T}} : \mathbb{X} \to \mathbb{Y}$, which maps any input to its corresponding label, both sampled from $\mathcal{D}$, by only knowing $\mathbb{T}$. The **empirical zero-one loss**, also known as empirical error or empirical risk, is given by:

$$\boldsymbol{L}_{\mathbb{T}} \; : \; \mathbb{H} \to \mathbb{Q}, \; h_{\mathbb{T}} \mapsto \frac{|(x_i, y_i) \in \mathbb{T} : h_{\mathbb{T}}(x_i) \neq y_i|}{|\mathbb{T}|}. \tag{2.1}$$

This corresponds to one minus the **training accuracy** commonly used in deep learning. The optimal ERM algorithm $ERM_{\mathbb{H}}$ will find the best predictor from the

space of possible predictors. The latter is also called the hypothesis class $\mathbb{H}$:

$$\mathrm{ERM}_{\mathbb{H}} : \mathbb{X} \times \mathbb{Y} \to \mathbb{H},\ \mathbb{T} \mapsto \arg\min_{h \in \mathbb{H}} \boldsymbol{L}_{\mathbb{T}}(h). \tag{2.2}$$

However, without restricting $\mathbb{H}$ the following worst-case predictor could be chosen:

$$h_{\text{worst\_case}}(x_i) = \begin{cases} y_i & \text{if } (x_i, y_i) \in \mathbb{T} \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

This predictor does not generalize to any data not included in $\mathbb{T}$. The phenomenon of a predictor focusing too much on training data and, therefore, generalizing worse to unseen data is known as **overfitting**. Since the predictor of Equation 2.3 does not generalize to any unknown data, $\mathbb{H}$ has to be restricted to a particular set of predictors. This restriction is usually referred to as **inductive bias**. In deep learning, $\mathbb{H}$ is usually called the **neural network**, **model**, or **architecture**. A neural network's internal structure provides specific inductive biases. A fundamental question in learning theory is for which hypothesis classes/models the ERM results in less overfitting.

The **true zero-one loss**, also known as the true risk, measured over the data generating distribution, is given by:

$$\boldsymbol{L}_{\mathcal{D}} : \mathbb{H} \to \mathbb{R},\ h \mapsto E_{(x,y)\sim\mathcal{D}} \left[\begin{cases} 1 & \text{if } h(x) \neq y, \\ 0 & \text{otherwise.} \end{cases} \right] \tag{2.4}$$

In deep learning, this measure is approximated by the **test accuracy**, which measures the zero-one loss over a set of unseen data. We call the distance between the zero-one loss of a predictor chosen by a learning algorithm and the optimal true zero-one loss the **true error** or the Bayesian error. Let h^* be the optimal predictor that minimizes $\boldsymbol{L}_{\mathcal{D}}(h)$ and $h^*_{\mathbb{H}}$ the optimal predictor in $\mathbb{H}$ and h the predictor chosen by ERM. The true error can now be divided into two components: first, the **approximation error** given by the distance of the true zero-one loss of $h^*_{\mathbb{H}}$ and h^*. Second, the **estimation error** given by the distance of the true zero-one loss of h and h^*:

$$\underbrace{\boldsymbol{L}_{\mathcal{D}}(h) - \boldsymbol{L}_{\mathcal{D}}(h^*)}_{\text{true error}} = \underbrace{(\boldsymbol{L}_{\mathcal{D}}(h^*_{\mathbb{H}}) - \boldsymbol{L}_{\mathcal{D}}(h^*))}_{\text{approximation error}} + \underbrace{(\boldsymbol{L}_{\mathcal{D}}(h) - \boldsymbol{L}_{\mathcal{D}}(h^*_{\mathbb{H}}))}_{\text{estimation error}} \tag{2.5}$$

This relationship is also figuratively explained in Figure 2.1. The approximation error is a measure of the inductive bias, or in other words, it indicates how much of the true zero-one loss is induced by using a specific hypothesis class. This error is independent of the training set size. The estimation error indicates the distance of loss between the predictor found by ERM and the actual best optimizer in $\mathbb{H}$.

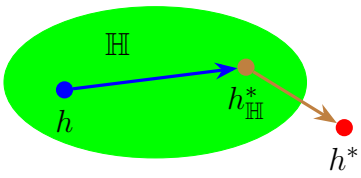


Figure 2.1: Figurative explanation of the estimation error between $h \in \mathbb{H}$ and the best predictor $h^*_{\mathbb{H}}$ of the hypothesis class $\mathbb{H}$ and the approximation error between $h^*_{\mathbb{H}} \in \mathbb{H}$ and the best predictor h^* possibly not in $\mathbb{H}$.

This error can be reduced by increasing the training data size. If $\mathbb{T}$ is a -possibly infinite- set with exactly the same distribution as $\mathcal{D}$ the estimation error vanishes.

In deep learning, two different sub-fields have formed to minimize one of these two errors, respectively. **Architecture Search** searches for neural networks, or in other words, for $\mathbb{H}$. Consequently, it focuses on minimizing the approximation error. Moreover, it aims to provide a $\mathbb{H}$ where it is easy to find $h^*_{\mathbb{H}}$. On the other hand, **Optimization** searches for an optimal solution contained in $\mathbb{H}$. Therefore, it focuses on the estimation error. **Optimization** can then again be divided into **Minimization** - mechanisms to decrease the loss - and **Regularization**. Regularization usually modifies the loss function so that the solutions found are closer to the optimal solution of the hypothesis class considered. Consequently, it minimizes the estimation error.

2.2 Surrogate Losses in Deep Learning

This section is based on (Goodfellow *et al.*, 2016) and (Shalev-Shwartz and Ben-David, 2014). Optimization in deep learning is the search for a predictor in $\mathbb{H}$ that minimizes the estimation error. This search is based on the training data and some regularization assumptions. However, performing this search is not trivial since the empirical zero-one loss $\boldsymbol{L}_{\mathbb{T}}$ (Equation 2.1), and the true zero-one loss $\boldsymbol{L}_{\mathcal{D}}$ are not differentiable. Consequently, it is hard to find a good descent direction that guides optimization on these functions. For a differentiable function, the steepest descent direction, which is the negative gradient, is often used to guide numerical optimization methods. To make the latter applicable to ERM, differentiable surrogate losses of $\boldsymbol{L}_{\mathbb{T}}$ and a differentiable predictor h have to be used. The differentiable surrogate loss of $\boldsymbol{L}_{\mathcal{D}}$ is the **true loss** $\mathcal{L}_{\mathbf{true}}$, which replaces the zero-one sample-wise loss by a differentiable loss:

$$\mathcal{L}_{\text{true}} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto E_{(x,y)\sim\mathcal{D}}\left[L_{(x,y)}(\boldsymbol{\theta})\right], \tag{2.6}$$

where $\mathcal{D}$ denotes the data generating distribution, $\boldsymbol{\theta}$ the parameters to optimize, and L a differential loss function applied to each sample, penalizing the distance

between the predicted label and y. Often used examples of such sample losses are the Cross-Entropy (CE) loss and the quadratic loss. The CE loss is:

$$L^{CE}_{(\boldsymbol{x},\boldsymbol{y})} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto \boldsymbol{y}^\top(-\log(h_{\boldsymbol{x}}(\boldsymbol{\theta})). \tag{2.7}$$

In this case, $\boldsymbol{y}$ and $h_x(\boldsymbol{\theta})$ are column vectors of probabilities. The CE loss minimizes the CE between the distribution of the true labels $\boldsymbol{y}$ and the distribution of the predicted labels $h_x(\boldsymbol{\theta})$. In detail, the CE measures the average amount of information (number of bits) required if labels are encoded in an encoding that is optimal for the distribution of the predicted labels $h_x(\boldsymbol{\theta})$ rather than encoded in an encoding that is optimal for the distribution of the correct labels. Consequently, minimizing the CE makes both distributions more similar.

The quadratic loss minimizes the squared Euclidean distance between the vector of predicted labels and true labels:

$$L^{quadratic}_{(\boldsymbol{x},\boldsymbol{y})} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto (h_{\boldsymbol{x}}(\boldsymbol{\theta}) - \boldsymbol{y})^\top(h_{\boldsymbol{x}}(\boldsymbol{\theta}) - \boldsymbol{y}). \tag{2.8}$$

In practice, $\mathcal{D}$ is usually unknown. However, we can get a finite dataset $\mathbb{D}$ of elements sampled from it. Consequently, the best approximation of $\mathcal{L}_{\text{true}}$ we can obtain is the **full-batch loss** $\mathcal{L}$ which takes into account the full dataset and resembles a surrogate for $\boldsymbol{L}_{\mathbb{T}}$:

$$\mathcal{L} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{D}|} \sum_{(x,y)\in\mathbb{D}} L_{(x,y)}(\boldsymbol{\theta}). \tag{2.9}$$

Since computing $\mathcal{L}$ for large $\mathbb{D}$ is computationally impractically, a subset $\mathbb{B}$ of $\mathbb{D}$ is used in practice. This subset is referred to as mini-batch. The resulting mini-batch loss $\mathcal{L}_{\mathbb{B}}$ is a noisy surrogate of $\mathcal{L}_{\text{true}}$ and $\mathcal{L}$, which can be efficiently computed if $|\mathbb{B}|$ is reasonably small:

$$\mathcal{L}_{\mathbb{B}} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{B}|} \sum_{(x,y)\in\mathbb{B}\subset\mathbb{D}} L_{(x,y)}(\boldsymbol{\theta}), \tag{2.10}$$

with $|\mathbb{B}| \ll |\mathbb{D}|$. However, this comes at the cost of inducing noise into the optimization process: a new mini-batch is sampled for each evaluation of $\mathcal{L}_{\mathbb{B}}$. Fortunately, all samples in a batch are drawn from $\mathcal{D}$. If we additionally assume that they are drawn independently, they are **independent and identically distributed (i.i.d.)**. In this case, $\mathcal{L}_{\mathbb{B}}$ is an unbiased estimator of $\mathcal{L}_{\text{true}}$ and has a standard error of $\sigma/\sqrt{|\mathbb{B}|}$, where σ is the true standard deviation of the distribution of sample losses. The denominator shows that the error decreases less than linearly if a larger batch size is used. This is also relevant for $\mathcal{L}$ since it resembles just a special case of $\mathcal{L}_{\mathbb{B}}$ with the largest batch size possible. Considering the last two points, the most

straightforward and most efficient approach we can do to minimize $\mathcal{L}_{\text{true}}$ in practice is to find a minimum of $\mathcal{L}$ with evaluations of $\mathcal{L}_{\mathbb{B}}$. Additionally, regularization heuristics are applied to increase the chance of finding a minimizer, which is also a minimizer of $\mathcal{L}_{\text{true}}$.

We denote the mini-batch gradient of $\mathcal{L}_{\mathbb{B}}$ $(= \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}))$ as $\boldsymbol{g}_{\mathbb{B}}$, and correspondingly the gradient of $\mathcal{L}$ $(= \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}))$ as $\boldsymbol{g}$. It is often assumed that gradients of sample losses are i.i.d. or even independent and Gaussian distributed (Mahsereci and Hennig, 2015).

In practice, however, those assumptions are usually not entirely given. For example, sample losses and gradients are not independent if regularization techniques such as BatchNorm (Ioffe and Szegedy, 2015) are used, resulting in network-intrinsic information flows between all sample losses in one batch. Further, elements in datasets are usually not independent.

2.3 Neural Networks from an Optimization Perspective

From an optimization point of view, each neural network is a special kind of hypothesis class containing a set of predictors. Optimization methods are then used to identify a predictor that fits the given data and has a low estimation error. In deep learning, this process is called training. The identified predictor is called the **trained network**. Neural networks are functions mapping from parameter space $\theta \subset \mathbb{R}^n$ to $\mathbb{Y}$. Having the parameters as input may feel wrong to a machine learner, but from an optimization perspective, the parameters $\boldsymbol{\theta} \in \theta$ are inputs, and elements of the dataset are handled as constants. The hypothesis classes that artificial neural network (NN) and deep neural network (DNN) represent are not exactly defined. However, NNs often, but not always, share the following characteristics:

- $|\boldsymbol{\theta}|$ is large. They often contain more than a million parameters.
- The whole network or part of it have a layer-like structure. Usually, each layer consists of a linear map on which a simple, point-wise nonlinearity is applied:

$$\text{NN}_{\boldsymbol{x}} : \theta \to \mathbb{Y},\ \boldsymbol{\theta} \mapsto f^{(n)}(\boldsymbol{\theta}_n, f^{(\dots)}(f^{(2)}(\boldsymbol{\theta}_2, f^{(1)}(\boldsymbol{\theta}_1, \boldsymbol{x})))), \tag{2.11}$$

  where a layer $f^{(i)}(\boldsymbol{\theta}_i, \boldsymbol{x})$ is of the form $\sigma(\boldsymbol{x}^\top \boldsymbol{\theta}_{i_1} + \boldsymbol{\theta}_{i_2})$. σ is a point-wise applied nonlinear mapping. $\boldsymbol{\theta}_{i_1}$ and $\boldsymbol{\theta}_{i_1}$ are disjunct parts of $\boldsymbol{\theta}_i$ which are a part of $\boldsymbol{\theta}$. $\boldsymbol{\theta}_{i_1}$ are called the weights and perform a linear transformation on $\boldsymbol{x}$. $\boldsymbol{\theta}_{i_2}$ are known as the biases, performing a linear translation (Goodfellow *et al.*, 2016). For σ the ReLU function $\boldsymbol{x} \mapsto \boldsymbol{x}^+$ is a typical candidate.

- The first derivative with respect to $\boldsymbol{\theta}$ exists. If a layer-like structure is present, this derivative can be numerically efficiently computed by the Backpropagation algorithm (Rumelhart *et al.*, 1986; LeCun *et al.*, 2012).
- The loss landscape is non-convex (Li *et al.*, 2018)(see also Section 2.9).

The boundary between NNs and DNNs is vaguely defined by the number of layers. Often NNs are referred to as DNNs if they have more than three layers and as shallow NNs if they have three or fewer. Several subfamilies of DNNs vary the layer-like network structure of Equation 2.11: In recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997), some layers have an internal state, influencing the next evaluation. Convolutional neural networks (CNNs) (LeCun *et al.*, 1999) use weight sharing in $\boldsymbol{\theta}$ to apply convolutional-like filters on spatial input vectors. residual neural networks (ResNets) (He *et al.*, 2016) use an additional linear residual connection for each layer similar to $\sigma(\boldsymbol{x}^\top \boldsymbol{\theta}_{i_1} + \boldsymbol{\theta}_{i_2}) + \boldsymbol{x}$. Layers in dense neural networks (DenseNets) (Huang *et al.*, 2017) do use the output not only of the previous layer but of multiple previous layers. MobileNets (Sandler *et al.*, 2018; Howard *et al.*, 2019) increase the granularity of convolutional filters for each or a subset of the layer's input.

2.4 Gradient Descent and Stochastic Gradient Descent

Due to the non-linearity and high dimensionality of NNs, it, is by far too costly to derive an analytic solution for a minimum. Thus, numerical optimization methods are required to minimize the mini-batch or full-batch loss (Equation 2.10 and 2.9). A helpful heuristic to do so is to follow the direction of steepest descent. Since NNs are differentiable, this direction is given by the negative gradient. Exploiting this heuristic, Gradient Descent (GD) (Cauchy *et al.*, 1847) is an optimization method that consecutively performs a step in the direction of the negative gradient with respect to the current parameters. However, in our scenario, the exact gradient is never known. Recall that we want to minimize the true loss $\mathcal{L}_{\text{true}}$ but only have its stochastic estimators $\mathcal{L}_{\mathbb{D}}$ or $\mathcal{L}_{\mathbb{B}}$ to do so. Therefore, a stochastic but unbiased estimate of $\mathcal{L}_{\text{true}}$'s gradient is used in practice. The resulting algorithm is Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951). Its update rule is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla_{\boldsymbol{\theta}_t} \mathcal{L}_{\mathbb{B}_t}(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t - \lambda_t \boldsymbol{g}_{\mathbb{B},t}, \tag{2.12}$$

where λ is the learning rate and t is the current update step. The pseudo-code of SGD is given in Algorithm 1. Note that $\boldsymbol{g}_{\mathbb{B},t}$ is not necessarily the direction of steepest descent of $\mathcal{L}_{\text{true}}(\boldsymbol{\theta}_t)$ and could even point in a direction where $\mathcal{L}_{\text{true}}(\boldsymbol{\theta}_t)$ increases. Thus, it is not apparent that this algorithm decreases the loss or even

converges to a minimum. Fortunately, (Robbins and Monro, 1951) were able to show that the sequence of SGD's consecutive $\boldsymbol{\theta}_t$ converges in probability to a minimizer $\boldsymbol{\theta}^*$ if the sequence of λ's used satisfies

$$\sum_{t=0}^{\infty} \lambda_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \lambda_t^2 < \infty, \tag{2.13}$$

and $\boldsymbol{\theta}^*$ is the only stationary point. Convergence in probability means that $\lim_{t\to\infty} E[||\boldsymbol{\theta}_t - \boldsymbol{\theta}^*||_2^2] \to 0$, where the random variable is the gradient of step $t-1$. Simply put, SGD must follow an infinitely long path with decreasing steps to converge. An exemplary schedule fulfilling Equation 2.13 is $\lambda_t = \frac{1}{t}$. Note that the convergence is in probability; thus, SGD might never exactly reach the minimum. In practice, other well-performing learning rate schedules such as *piece-wise constant learning rate* or *cosine decay* are often used (see (Loshchilov and Hutter, 2017; Smith, 2017)). von Bachmann and Mutschler (2020), analyzed the convergence of SGD in practice and found that SGD tends to move consistently away from already visited positions, even when a schedule with decreasing learning rate is used. Figuratively speaking, it moves continuously in a low-loss subspace.

Algorithm 1 Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951)

Input:

symbol	explanation	default value
$\mathcal{L}_{\mathbb{B}}$	mini-batch loss	–
$\lambda(t)$	learning rate schedule	usually starting with 10^{-1} to 10^{-4}
$\boldsymbol{\theta}_0$	initial parameters	usually sampled with Xavier (Glorot) initialization (Glorot and Bengio, 2010)
-	*stopping criterion*	usually a fixed number of update steps

1: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
2: $t \leftarrow 0$
3: **while** *stopping criterion* not met **do**
4: $\quad \mathbb{B}_t \leftarrow$ sampleMiniBatch()
5: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda(t)\boldsymbol{g}_{\mathbb{B}_t}$
6: $\quad t \leftarrow t+1$
7: **end while**
8: **return** $\boldsymbol{\theta}$

In non-stochastic optimization, a typical improvement of first-order methods such as GD are second-order methods such as the Newton method. These methods exploit the second derivative with respect to $\boldsymbol{\theta}$ known as the Hessian $\boldsymbol{H}$, and therefore approximate the loss locally by a parabola of $dim(\boldsymbol{\theta}) \times dim(\boldsymbol{\theta})$ dimensions. However, the computation of $\boldsymbol{H}$ is practically unfeasible because memory capacities

are often too small to hold $dim(\boldsymbol{\theta}) \times dim(\boldsymbol{\theta})$ full-precision floating-point numbers. E.g., the gradient for a ResNet requires approximately 150 MB of memory, resulting in 22.5 GB of memory for the Hessian, which exceeds the memory capacities of most today's GPUs. For example, an Nvidia GeForce RTX 3080 Ti (NVIDIA, 2021) has 12 GB of memory. Consequently, classical second-order methods are not applicable for DNNs. Nevertheless, some research considers stochastic second-order methods, showing that those can achieve higher per-step performance than SGD (Schraudolph *et al.*, 2007; Berahas *et al.*, 2021; Martens and Grosse, 2015; Ramamurthy and Duffy, 2017; Botev *et al.*, 2017; Dangel *et al.*, 2020). However, this advantage is compensated by significantly longer wall-clock time required for one update step.

2.5 Line Searches in the Classical, Deterministic Scenario

The following section is based on (Nocedal and Wright, 2006) and (Luenberger *et al.*, 1984). GD performs a step of size λ in the negative gradient direction; however, if this step is too large, the loss might increase; if it is too small, optimization might take long since the loss is only slightly decreased at each step. Therefore, it is important to automatically infer a λ that decreases the loss "sufficiently" far. This eliminates λ as a hyperparameter, which is often a very sensitive hyperparameter to tune. Doing this by hand can be very tedious. Line searches are a general approach to estimate λ's. In classical non-stochastic and low-dimensional optimization, line searches are straightforward to apply because the measurements are exact and computationally inexpensive. However, as we will discuss later, this is not the case in the stochastic scenario.

Let $f(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$ be a non-stochastic differentiable function to be optimized. Moreover, let f be bounded from below and continuously differentiable. f along a line through the current position $\boldsymbol{\theta}_t \in \mathbb{R}^n$ in unit direction $\boldsymbol{d}_t \in \mathbb{R}^n$ is given by:

$$f_{\text{line}} : \mathbb{R} \to \mathbb{R},\ s \mapsto f(\boldsymbol{\theta_t} + s \cdot \boldsymbol{d}_t), \tag{2.14}$$

where s is the **step size** along the line. Note that we now have a univariate domain to optimize only depending on s. In this work, we will use the term **step size** (s) if it represents a step in unit direction (i.e., the direction vector has length 1) and the term **learning rate** (λ) otherwise. $f'_{line}(s)$ is the directional derivative given by the projected gradient:

$$f'_{\text{line}} : \mathbb{R} \to \mathbb{R},\ s \mapsto \nabla_{\boldsymbol{\theta_t} + s \cdot \boldsymbol{d}_t} f(\boldsymbol{\theta_t} + s \cdot \boldsymbol{d}_t)^\top \boldsymbol{d}_t. \tag{2.15}$$

In the case where $\boldsymbol{d}_t$ is the negative unit gradient, $\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)^\top \boldsymbol{d}_t$ simplifies to $-||\boldsymbol{d}_t||$

since $\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)^\top \frac{-\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)}{||-\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)||} = -\frac{||\boldsymbol{d}_t||^2}{||\boldsymbol{d}_t||} = -||\boldsymbol{d}_t||$. An exact line search estimates an step size s that is a global minimizer along the line:

$$s_{\min} = \arg\min_s f_{\text{line}}(s). \tag{2.16}$$

However, exact line searches are only applicable if f_{line} can be approximated by a reasonably simple function, e.g., a lower-order polynomial. Otherwise, a search for $s_{\min}$ must be performed, e.g., by bracketing or backtracking, which is computationally expensive. Consequently, if no suitable approximation is known, a λ must be determined that satisfies specific conditions that assure sufficiently fast decrease and convergence. The two most common conditions are the **sufficient decrease condition**, also known as the **first Wolfe condition** or **Armijo condition**, and the **curvature condition**, also known as the **second Wolfe condition**. The sufficient decrease condition holds if s satisfies

$$f_{\text{line}}(s) \leq f(\boldsymbol{\theta}_t) + c_1 s f'_{\text{line}}(0), \tag{2.17}$$

where c_1 is the sufficient decrease constant. A commonly used value for c_1 is 10^{-4}. As shown in Figure 2.2 all values of $f_{\text{line}}(s)$ that are below the line $f(\boldsymbol{\theta}_t) + c_1 s \nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)^\top \boldsymbol{d}_t$ are of sufficient decrease.

The sufficient decrease condition is insufficient to ensure reasonable progress since unnecessary small s are valid, which would decrease f only slightly. The **curvature condition** solves the latter:

$$f'_{\text{line}}(s) \geq c_2 f'_{line}(0), \tag{2.18}$$

where c_2 denotes the curvature constant, typically chosen to be 0.9. As shown in Figure 2.2, this condition holds for all values of $f_{\text{line}}(s)$ that have a larger slope than $c_2 f'_{\text{line}}(0)$. Considering convergence, it is evident that the loss decreases at each step if both conditions are satisfied. In addition, global convergence to a stationary point - meaning that $\lim_{t\to\infty} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) = \mathbf{0}$ - is given if the cosine of the angle α_t between $\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)$ and $\boldsymbol{d}_t$,

$$\cos(\alpha_t) = \frac{\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)^\top \boldsymbol{d}_t}{||\nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)||\ ||\boldsymbol{d}_t||} \tag{2.19}$$

is bounded away from 0, which means that there exists a δ such that $\cos(\alpha_t) \geq \delta > 0$ for all t (Nocedal and Wright, 2006, §3.2).

As an alternative to the curvature condition, a backtracking line search (see Algorithm 2) can be performed to avoid too small step sizes: An initial large extrapolation step size $\hat{s}$ is chosen. Then it is decreased by a factor ρ in $(0,1)$ until the sufficient decrease condition is satisfied. As long as $\hat{s}$ is reasonably large, a

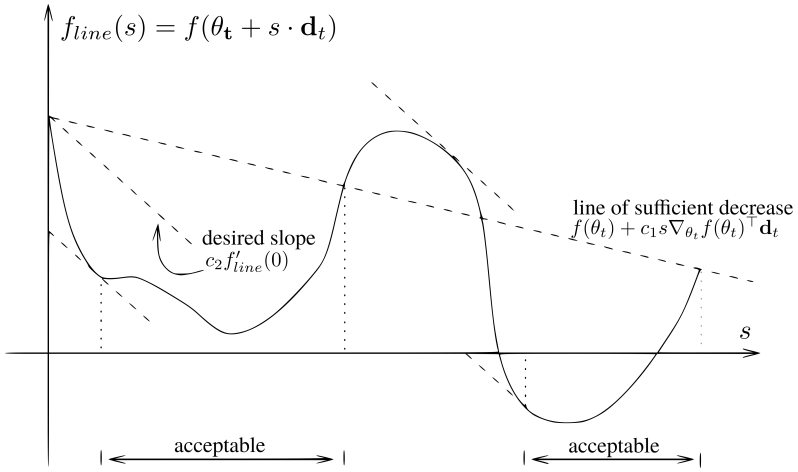


Figure 2.2: Figurative explanation of update step intervals that fulfill the sufficient decrease (Eq. 2.17) and curvature conditions (Eq. 2.18). The sufficient decrease condition excludes all s for which $f_{\text{line}}(s)$ is greater than the line of sufficient decrease. The curvature condition additionally excludes all s for which $f'_{\text{line}}(s)$ is smaller than the desired slope to ensure that very small s are excluded. Note that $\boldsymbol{\theta}_t$ is located at the origin $s = 0$. In practice, points of acceptance are determined by sampling along the line. Further note that the line of the desired slope is plotted multiple times at relevant positions. Figure from (Nocedal and Wright, 2006); used and adapted with permission.

suitable s is found before unsuitable small ones. The latter lead to an unnecessary small decrease of loss. Consequently, checking the curvature condition is not required. $\hat{s}$ is often chosen to be 1. Global convergence on real analytic functions was shown by (Absil *et al.*, 2005). Moreover, for strictly convex stochastic

Algorithm 2 Backtracking Line Search. This covers only the line search routine, omitting the weight update.

	symbol	explanation	default value
	$f_{\text{line}}(s)$	univariate line function of f	–
Input:	$\hat{s}$	extrapolation step size > 0	1
	β	step size decrease factor $\in (0, 1)$	0.9 or 0.99
	c_1	sufficient decrease constant $\in (0, 1)$	10^{-4}

1: $s \leftarrow \hat{s}$
2: **while** $f_{line}(s) \geq f_{\text{line}}(0) + c_1 s f'_{\text{line}}(0)$ **do**
3: $\quad s \leftarrow \beta s$
4: **end while**
5: **return** s

functions, global convergence was shown by (Bertsekas and Tsitsiklis, 2000) if the gradient is Lipschitz continuous and the assumptions of Equation 2.13 hold. A function $f : \mathbb{R} \to \mathbb{R}$ is Lipschitz contiuous if there exists a constant $k \in \mathbb{R}^+$ for which $|f(x_1) - f(x_2)| \leq k|x_1 - x_2|$ holds for all x_1 and x_2.

Another branch of line searches uses lower-order polynomials or splines to approximate f_{line} (Nocedal and Wright, 2006, § 3.5). In particular, the quadratic case is efficient since only three measurements are needed: $f_{\text{line}}(0)$, $f_{\text{line}}(s')$ and $f'_{\text{line}}(0)$, where s' is the sample step size. A proof of convergence on a deterministic quadratic function is given in (Luenberger *et al.*, 1984, page 235). On non-quadratic functions, global convergence can often not be assured; e.g., parabolic approximations already diverge on x^4. Chapter 4 and 6 will introduce and analyze two parabolic approximation line searches for the stochastic scenario.

2.6 Line Searches in the Stochastic Scenario

This section introduces line searches designed for the stochastic scenario. We will focus on practically applicable approaches, which can be used to train neural networks efficiently, but in most cases, do not provide theoretical justifications. After two introductory sections about definitions and challenges for stochastic line searches (Section 2.6.1, 2.6.2), we introduce the Stochastic Line Search (Vaswani *et al.*, 2019) (Section 2.6.3), the Gradient Only Line Search that is Inexact (Kafka and Wilke, 2019) (Section 2.6.4) and the Probabilistic Line Search (Mahsereci and Hennig, 2015) (Section 2.6.5).

2.6.1 Definitions for Stochastic Line Searches

In the following, we will define the notions of full-batch and mini-batch losses along lines and their corresponding gradients and projected gradients. These definitions and associated symbols will be retained in this and all of the following chapters.

The full-batch loss $\mathcal{L}$ (see Equation 2.9) along a line in the direction of the current mini-batch gradient $\boldsymbol{g}_{\mathbb{B},t}$ through the current parameter position $\boldsymbol{\theta}_t \in \mathbb{R}^n$ at optimization step $t \in \mathbb{N}$ is defined as:

$$l_t : \mathbb{R} \to \mathbb{R},\ s \mapsto \mathcal{L}(\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B},t}). \tag{2.20}$$

The univariate derivative along the line is given by:

$$l'_t : \mathbb{R} \to \mathbb{R},\ s \mapsto \frac{\mathrm{d}}{\mathrm{d}s}\mathcal{L}(\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B},t}) = \nabla_{\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B},t}}\mathcal{L}(\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B},t})^\top \boldsymbol{g}_{\mathbb{B},t}. \tag{2.21}$$

Similarly, the loss along a line of the mini-batch loss $\mathcal{L}_{\mathbb{B}_i,t}$ in direction $\boldsymbol{g}_{\mathbb{B}_j,t}$ is

defined as:

$$l_{\mathbb{B}_i,t} : \mathbb{R} \to \mathbb{R},\ s \mapsto \mathcal{L}_{\mathbb{B}_i,t}(\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B}_t,t}). \tag{2.22}$$

Note that the batch $\mathbb{B}_i$ from which the mini-batch loss is computed might differ from the batch $\mathbb{B}_t$ the gradient is computed from. The batch indices are omitted if both batches are identical. Also, the subscript t is omitted in the following if it is irrelevant in the context. $l_{\mathbb{B}_i,t}$'s directional derivative is given by:

$$l'_{\mathbb{B}_i,t} : \mathbb{R} \to \mathbb{R},\ s \mapsto \frac{\mathrm{d}}{\mathrm{d}s}\mathcal{L}_{\mathbb{B}_i,t}(\boldsymbol{\theta_t} + s\boldsymbol{g}_{\mathbb{B}_t,t}) = \nabla_{\boldsymbol{\theta_t}+s\boldsymbol{g}_{\mathbb{B}_t,t}}\mathcal{L}_{\mathbb{B}_i,t}(\boldsymbol{\theta}_t + s\boldsymbol{g}_{\mathbb{B}_t,t})^\top \boldsymbol{g}_{\mathbb{B}_t,t}. \tag{2.23}$$

2.6.2 Challenges for Stochastic Line Searches

In the deterministic scenario, function evaluations are usually cheap and exact. This makes it easy to validate the sufficient decrease or curvature condition or perform a parabolic approximation. Further, since measurements are exact, the search space of s can be narrowed down efficiently. In the stochastic scenario, this is not the case since function evaluations are noisy and, thus, it is not clear whether s has to be increased or decreased to find a desired position along the line. Furthermore, it is not clear whether a condition valid for $l_{\mathbb{B},t}(s)$ also holds for $l_t(s)$. Consequently, deterministic line searches in general cannot be directly applied to the stochastic scenario, i.e., on $\mathcal{L}_{\mathbb{B}}$. However, they still work under certain assumptions or observed properties, such as Stochastic Backtracking Line Search (Section 2.6.3 and Parabolic Approximation Line Search (Chapter 4)) do. In addition, specific line searches for the stochastic scenario were designed, such as Gradient Only Line Search (Section 2.6.4), Probabilistic Line Search (Section 2.6.5) and Large-Batch Parabolic Approximation Line Search (Chapter 6). All in all, line searches in the deterministic scenario are considered a solved problem (Nocedal and Wright, 2006, §3), whereas science is still struggling to develop efficient and effective stochastic line searches with proven convergence guarantees.

2.6.3 Stochastic Line Search

This section is based on (Vaswani *et al.*, 2019). Stochastic Line Search (SLS) (Vaswani *et al.*, 2019) is a slightly adapted variant of a typical Backtracking Line Search (see Algorithm 2) used in the deterministic scenario. SLS works in the stochastic case if the interpolation assumption holds:

Assumption 1 (Interpolation)**.** *The gradient with respect to each mini-batch converges to zero at the optimum of a function. This implies that if $\mathcal{L}(\boldsymbol{\theta})$ is minimized at $\boldsymbol{\theta}^*$ and thus $\nabla_{\boldsymbol{\theta}^*}\mathcal{L}(\boldsymbol{\theta}^*) = \mathbf{0}$, then for all $L_{(x,y)}(\boldsymbol{\theta})$, with $(x,y) \in \mathbb{D}$, $\nabla_{\boldsymbol{\theta}^*}L_{(x,y)}(\boldsymbol{\theta}^*) = \mathbf{0}$ holds.*

For example, interpolation is satisfied for a linear model with the squared hinge loss for binary classification on linearly separable data (Vaswani *et al.*, 2019). In addition, interpolation is assumed that it might hold for other typical deep learning scenarios using the CE loss (Equation 2.7) or the quadratic loss (Equation 2.8) on non-linear models, especially if the models are over-parameterized, meaning that $n = dim(\boldsymbol{\theta}) \gg |\mathbb{D}|$. SLS is proven to converge on strongly convex and convex functions assuming interpolation, and Lipschitz continuous gradients.

In the following, we introduce the best performing variant of SLS, which adapts SGD to perform line searches based on the sufficient decrease condition. The pseudo-code is found in Algorithm 3. It performs a basic backtracking line search (see Algorithm 2) on $\mathcal{L}_{\mathbb{B}}$; thus, it evaluates the sufficient decrease condition on noisy estimates of $\mathcal{L}$. Since it considers $\boldsymbol{g}_{\mathbb{B}}$ as the direction vector and not the normalized variant $\hat{\boldsymbol{g}}_{\mathbb{B},t}$ we use the symbol λ for the step size along the line. As decreasing the learning rate continuously may not work for non-convex problems, a heuristic is introduced that increases λ before each line search by $\gamma^{|\mathbb{B}|/|\mathbb{D}|}$, where γ is a learning rate increase factor, $|\mathbb{B}|$ the batch size and $|\mathbb{D}|$ the dataset size. This is a

Algorithm 3 SGD + sufficient decrease condition variant of SLS (adapted from Vaswani *et al.* (2019))

Input:

symbol	explanation	default value
$\mathcal{L}_{\mathbb{B}}$	mini-batch loss	–
$\boldsymbol{\theta}_0$	initial parameters	–
$\lvert\mathbb{D}\rvert$	data set size	–
$\lvert\mathbb{B}\rvert$	batch size	128
λ_{init}	initial learning rate	1
c_1	sufficient decrease constant	0.1
β	learning rate decrease factor	0.9
γ	learning rate increase factor	2.0
T	amount of training steps	-

$\lambda \leftarrow \lambda_{\text{init}}$
for $t = 0, \ldots, T$ **do**
  $\mathbb{B}_t \leftarrow$ sample mini-batch of size $|\mathbb{B}|$
  $\lambda \leftarrow \lambda\gamma^{|\mathbb{B}|/|\mathbb{D}|}/\beta$
  repeat
    $\lambda \leftarrow \beta\lambda$
    $\tilde{\boldsymbol{\theta}}_t \leftarrow \boldsymbol{\theta}_t - \lambda \boldsymbol{g}_{\mathbb{B}_t}$
  until $\mathcal{L}_{\mathbb{B}_t}(\tilde{\boldsymbol{\theta}}_t) \leq \mathcal{L}_{\mathbb{B}_t}(\boldsymbol{\theta}_t) - c_1\lambda||\boldsymbol{g}_{\mathbb{B}_t}||^2$
  $\boldsymbol{\theta}_{t+1} \leftarrow \tilde{\boldsymbol{\theta}}_t$
end for
return $\boldsymbol{\theta}_{t+1}$

well-established heuristic that indirectly accounts for the noise introduced into optimization by using mini-batches. This heuristic is also considered by (Schmidt *et al.*, 2017, 2015; Paquette and Scheinberg, 2018; Truong and Nguyen, 2018). Vaswani *et al.* (2019) showed in their experiments that SLS can outperform SGD on training loss and validation accuracy if used to train a ResNet-34 (He *et al.*, 2016) or a bottleneck DenseNet-121 (Huang *et al.*, 2017) on CIFAR-10 or CIFAR-100 (Krizhevsky *et al.*, 2009).

2.6.4 Gradient-Only Line Search that is Inexact

This section is based on Kafka and Wilke (2019). A line search approach designed specifically for the stochastic scenario is Gradient-Only Line Search that is Inexact (GOLS-I) (Kafka and Wilke, 2019). In this approach, directional derivatives in line direction are sampled until two consecutive measure points are found, for which the directional derivative changes from negative to positive. In the deterministic case, these points have to enclose at least one minimum. Kafka and Wilke (2019) show empirically that the noise of batch-wise directional derivatives along a line is much lower than the noise of mini-batch losses. Consequently, searching for a point near which the sign of the directional derivative changes from negative to positive is a better indicator of a minimum on l, than a minimum on $l_{\mathbb{B}}$'s is.

In detail, the algorithm works as follows: First, it checks whether the current λ satisfies a modified strong curvature condition (compare to Equation 2.18 and 2.23):

$$0 \leq l'_{\mathbb{B}_i}(\lambda) \leq c_2 |l'_{\mathbb{B}_j}(0)|, \tag{2.24}$$

with $c_2 > 0$ and $\mathbb{B}_i$, $\mathbb{B}_j$ denote two different batches drawn from the same distribution. If this condition is satisfied, λ is returned. If not, and if $l'_{\mathbb{B}_i}(\lambda)$ is negative, λ is consecutively increased by a factor $\eta > 1$ until $l'_{\mathbb{B}_i}(\lambda)$ becomes positive. In the same way, if $l'_{\mathbb{B}_i}(\lambda)$ is positive λ is consecutively divided by η until $l'_{\mathbb{B}_i}(\lambda)$ becomes negative. Note that for each evaluation of $l'_{\mathbb{B}_i}(\lambda)$ a different $\mathbb{B}_i$ is used and that the algorithm assumes the existence of a minimum along the line. Figure 2.3 shows an explaining illustration of this procedure. A detailed pseudo code is provided in Algorithm 4. [§4.2](Kafka and Wilke, 2019) prove global convergence under specific assumptions using Lyapunov's global stability theorem (Lyapunov, 1992). These assumptions include that the algorithm decreases the loss in expectation at each update step, and that a unique global minimizer exists.

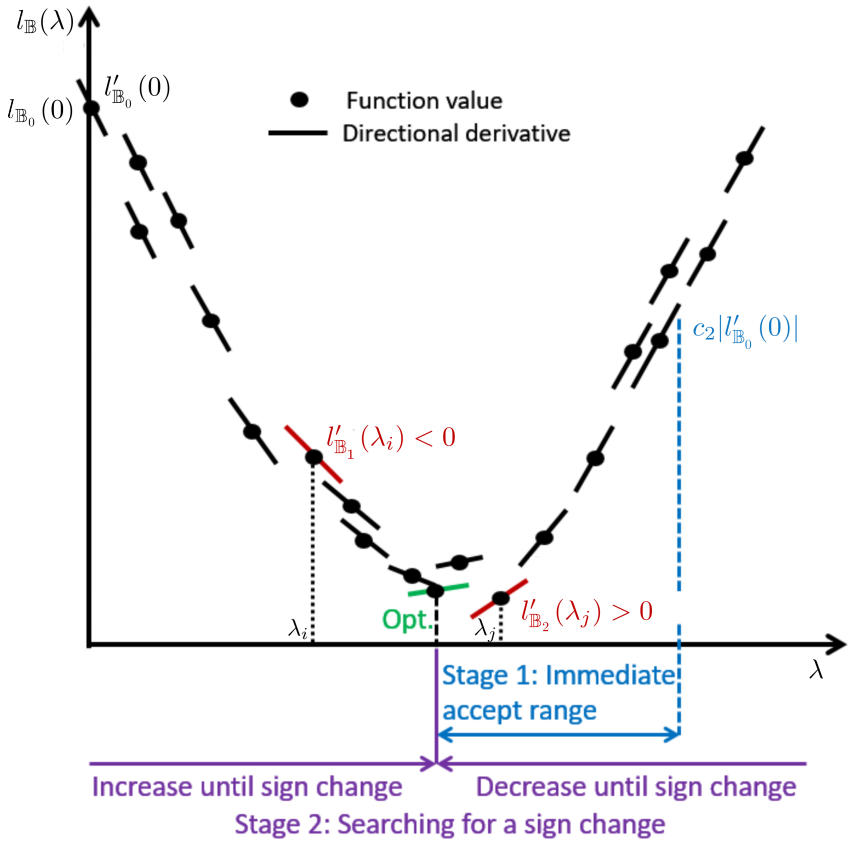


Figure 2.3: Illustrative explanation of GOLS-I's behavior along a line. In stage one, it is evaluated whether the first step λ_i is in the immediate acceptance range (see Eq. 2.24). If so, it is returned. In this example, this is not the case. Since $l'_{\mathbb{B}_1}(\lambda_i)$ is negative, λ_i is increased to λ_j. Now $l'_{\mathbb{B}_2}(\lambda_j)$ is positive, therefore, a sign change occurred and λ_j is returned. Note that each directional derivative is measured with a different batch. Figure adapted with permission from (Kafka and Wilke, 2019).

Algorithm 4 Gradient-Only Line Search that is Inexact (GOLS-I) (adapted from Kafka and Wilke (2019)). The weight update is omitted for clarity. Consequently, only the internal line search routine is described.

Input:

symbol	explanation	default value
$l'_{\mathbb{B}}$	directional mini-batch derivative along a line (see Eq. 2.23)	–
λ_{init}	initial learning rate	10^{-8} or λ of last search
c_2	curvature constant from Eq 2.24	0.9
η	learning rate scaling factor	2
λ_{min}	minimal learning rate	10^{-8}

$\lambda_{max} = min(\frac{1}{||\boldsymbol{d}||_2}, 10^7)$
$\lambda \leftarrow$ clip_to_limit$(\lambda_{\text{init}}, \lambda_{min}, \lambda_{max})$
$\mathbb{B}_0, \mathbb{B}_1 \leftarrow$ sample_batches()
if $0 \leq l'_{\mathbb{B}_1}(\lambda) \leq c_2 |l'_{\mathbb{B}_0}(0)|$ **then** ▷ # see Eq. 2.24
  return λ
else if $l'_{\mathbb{B}_1}(\lambda) < 0$ **then**
  flag $\leftarrow 1$, increase step size
else
  flag $\leftarrow 2$, decrease step size
end if
while True **do**
  $\mathbb{B}_i \leftarrow$ sample_batch()
  if flag $== 1$ **then**
    $\lambda \leftarrow \lambda \cdot \eta$
    if $\lambda > \frac{\lambda_{max}}{\eta}$ or $l'_{\mathbb{B}_i}(\lambda) \geq 0$ **then**
      return λ
    end if
  end if
  if flag $== 2$ **then**
    $\lambda \leftarrow \frac{\lambda}{\eta}$
    if $\lambda < \lambda_{min} \cdot \eta$ or $l'_{\mathbb{B}_i}(\lambda) < 0$ **then**
      return λ
    end if
  end if
end while

2.6.5 Probabilistic Line Search

This section is based on Mahsereci and Hennig (2015). Probabilistic Line Search (PLS) (Mahsereci and Hennig, 2015) is designed specifically for the stochastic scenario. It combines traditional concepts of line searches in the deterministic scenario with methods of Bayesian optimization. Specifically, a Gaussian Process (GP) posterior is utilized as a surrogate for l_t. The latter is estimated over several measurements of $l_{\mathbb{B},t}$. Then, a Bayesian Optimization objective is used to estimate the next sample position along the line. Finally, a probabilistic version of the sufficient decrease and curvature conditions (see Equation 2.17 and 2.18) is used as the termination criterion for a candidate position. Figure 2.4 exemplarily illustrates the difference between a traditional interpolating line search and a GP posterior. Note that the line direction is not normalized; consequently, we use the symbol λ for steps along the line. In this section, following Mahsereci and Hennig (2015), we will use the terms first and second Wolf condition for the sufficient decrease and curvature conditions, respectively.

Before we begin the detailed derivation of the algorithm, let us briefly dive into the basics of probability calculus: Let X, Y be random variables. Their conditional probability is given as $p(X|Y) = \frac{p(X,Y)}{p(Y)}$, where $p(X,Y)$ is their joined probability. The fundamental Bayesian Theorem is given as:

$$\underbrace{p(X|Y)}_{\text{posterior}} = \frac{\overbrace{p(Y|X)}^{\text{likelihood}}\overbrace{p(X)}^{\text{prior}}}{\underbrace{p(Y)}_{\text{evidence}}}, \quad \text{with} \quad p(Y) \neq 0. \tag{2.25}$$

$p(X = x)$ denotes the probability that X takes the value x; in short, $p(x)$. If X is continuous, p denotes a probability density function (pdf). $p(X \leq x)$ is then given as $\int_{-\infty}^{x} p(a)da$. An often assumed distribution is the univariate Gaussian distribution: $p(x) = \mathcal{N}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, where μ and σ denote the mean and standard deviation, respectively. The univariate case can be generalized to the multivariate case, where $\boldsymbol{x}$ is a vector of n random variables:

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{2.26}$$

where $\boldsymbol{\mu}$ is a vector of means and $\boldsymbol{\Sigma}$ is a positive definite covariance matrix. In the case that the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$ is a multivariate Gaussian, $\boldsymbol{\Sigma}$ can be separated into blocks: $\boldsymbol{\Sigma} = \begin{Bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x},\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{x},\boldsymbol{y}} \\ \boldsymbol{\Sigma}_{\boldsymbol{y},\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{y},\boldsymbol{y}} \end{Bmatrix}$. A multivariate Gaussian distribution can solely represent a finite-dimensional vector of random variables; thus, an infinite-dimensional function cannot be modeled with it. For this, GPs are the ap-

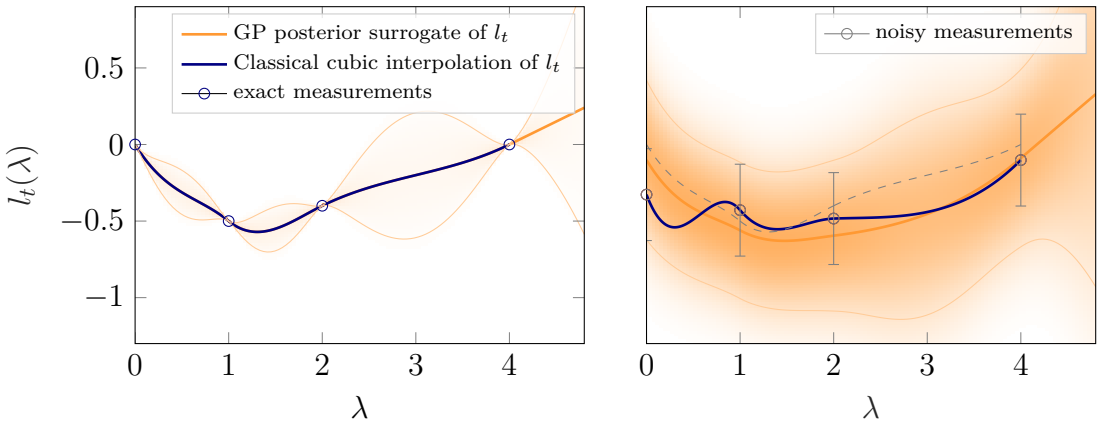


Figure 2.4: Comparison of l_t's approximation with a once integrated Wiener Process' posterior introduced in Eq. 2.30, and a traditional interpolating line search: GP posterior mean is given in solid orange, the two standard deviations in thinner solid orange, and the local pdf marginal as shading. Function value observations are shown as gray circles. Dark blue represents the noise ignoring interpolation by piece-wise cubic splines as used in traditional line searches. *Left:* observations are exact; the mean of the GP and the cubic spline interpolator of a classic line search coincide. *Right:* same observations with additive Gaussian noise (error-bars indicate $\pm$ 1 standard deviations). The noise-free interpolator is given in dashed gray for comparison. The classic interpolator in dark blue, which matches the observations exactly, becomes unreliable; the GP reacts robustly to noisy observations; the GP-mean still consists of piece-wise cubic splines. Caption and figure from (Mahsereci and Hennig, 2015); adapted with permission.

propriate tool. A GP is a potentially infinite set of random variables $\{f(i) : i \in \mathbb{I}\}$ indexed by a set $\mathbb{I}$ such that the joint distribution of every finite subset of random variables is a multivariate Gaussian (Chuong, D, 2008). E.g., $\mathbb{I}$ might index a line in $\mathbb{R}$. A GP is fully specified by a mean function $\mu(i) : \mathbb{I} \to \mathbb{R}$ and a covariance or kernel function $k(z, z') : \mathbb{I} \times \mathbb{I} \to \mathbb{R}$, representing a potentially infinite-dimensional covariance matrix. It is of use that the set of Gaussian distributions and the set of GPs are closed under linear maps.

Coming back to PLS, its goal is to find a probabilistic surrogate of l_t that best describes l_t subject to measurements of $l_{\mathbb{B},t}$ and $l'_{\mathbb{B},t}$ at different λ's. Such measurements are interpreted below as realizations of Gaussian random variables given as vectors: $\boldsymbol{l}_{\mathbb{B},t}$ and $\boldsymbol{l}'_{\mathbb{B},t}$. For the probabilistic surrogate, a reasonable approximation of the posterior probability $p(l_t \mid \boldsymbol{l}_{\mathbb{B},t}, \boldsymbol{l}'_{\mathbb{B},t})$ has to be found, whose mean is the surrogate for l_t.

Assuming Gaussian distributed losses and gradients, the conditional probability of a single $l_{\mathbb{B},t}$ and $l'_{\mathbb{B},t}$ given l_t, also called the likelihood, is given as:

$$p(l_{\mathbb{B},t}(\lambda), l'_{\mathbb{B},t}(\lambda) | l_t(\lambda)) = \mathcal{N}\left(\begin{bmatrix} l_{\mathbb{B},t}(\lambda) \\ l'_{\mathbb{B},t}(\lambda) \end{bmatrix}; \begin{bmatrix} l_t(\lambda) \\ l'_t(\lambda) \end{bmatrix}, \begin{bmatrix} \sigma^2_{l_t(\lambda)} & 0 \\ 0 & \sigma^2_{l'_t(\lambda)} \end{bmatrix}\right), \tag{2.27}$$

where $\mathcal{N}$ denotes a Normal distribution and σ^2 is the variance. For simplicity, $l_{\mathbb{B},t}(\lambda)$ and $l'_{\mathbb{B},t}(\lambda)$ are assumed to be independent.

To keep things clear, we will at first consider the prior for l_t and afterwards introduce the joint prior for l_t and l'_t. As prior, a once-integrated Wiener process, i.e., a GP $p(l_t) = \mathcal{GP}(l_t; 0, k)$ with zero mean and covariance function $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is assumed:

$$k(\lambda, \lambda') = m^2 \left(\frac{1}{3}(\min(\lambda, \lambda') + \tau)^3 + \frac{1}{2}|\lambda - \lambda'|(\min(\lambda, \lambda') + \tau)^2\right), \tag{2.28}$$

where m is a scaling factor, which can be eliminated by specific scaling of $l_{\mathbb{B}}$ and $l'_{\mathbb{B}}$ (see (Mahsereci and Hennig, 2015, §3.4) and τ a shifting factor. In the final algorithm, the τ is chosen to be 10, and λ and λ' are chosen to be greater than or equal to 0. With the latter property Equation 2.28 can be simplified to:

$$k(\lambda, \lambda') = \begin{cases} m^2(\frac{1}{2}\tilde{\lambda}\tilde{\lambda}'^2 - \frac{1}{6}\tilde{\lambda}'^3) & \text{if } \lambda > \lambda' \\ m^2(\frac{1}{2}\tilde{\lambda}^2\tilde{\lambda}' - \frac{1}{6}\tilde{\lambda}^3) & \text{if } \lambda \leq \lambda', \end{cases} \tag{2.29}$$

where $\tilde{\lambda} = \lambda + \tau$ and $\tilde{\lambda}' = \lambda' + \tau$.

Using Equation 2.28 as a prior is reasonable because, first, discrete candidate points can be computed analytically and, second, the equation is robust to the noise introduced by using $l_{\mathbb{B},t}$ and $l'_{\mathbb{B},t}$.

Equation 2.28, which defines a once integrated Wiener Process for l_t, suggests a Wiener Process for l'_t because the derivative is a linear operator (Papoulis, 1991, §10). The derived bivariate Wiener Process $p(l_t; l'_t)$ is:

$$p(l_t; l'_t) = \mathcal{GP}\left(\begin{bmatrix} l_t \\ l'_t \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k & k^{\partial} \\ {}^{\partial}k & {}^{\partial}k^{\partial} \end{bmatrix}\right), \tag{2.30}$$

$$k^{\partial} = \frac{\partial k(\lambda, \lambda')}{\partial \lambda'} = \begin{cases} m^2(\tilde{\lambda}\tilde{\lambda}' - \frac{1}{2}\tilde{\lambda}'^2) & \text{if } \lambda > \lambda', \\ m^2\frac{1}{2}\tilde{\lambda}^2 & \text{if } \lambda \leq \lambda' \end{cases}, \tag{2.31}$$

$${}^{\partial}k = \frac{\partial k(\lambda, \lambda')}{\partial \lambda} = \begin{cases} m^2\frac{1}{2}\tilde{\lambda}'^2 & \text{if } \lambda > \lambda', \\ m^2(\tilde{\lambda}\tilde{\lambda}' - \frac{1}{2}\tilde{\lambda}^2) & \text{if } \lambda \leq \lambda' \end{cases}, \tag{2.32}$$

$${}^{\partial}k^{\partial} = \frac{\partial^2 k(\lambda, \lambda')}{\partial \lambda \partial \lambda'} = m^2 \min(\tilde{\lambda}, \tilde{\lambda}') \tag{2.33}$$

Now, let a set of evaluations $(\boldsymbol{\lambda}, \boldsymbol{l}_{\mathbb{B},t}, \boldsymbol{l}'_{\mathbb{B},t})$ (vectors, with elements λ_i, $l_{\mathbb{B}_i,t}$, (λ_i), $l'_{\mathbb{B}_j,t}(\lambda_j)$) with independent likelihoods (Eq. 2.27) be given. Combining the joined likelihood $p(\boldsymbol{l}_{\mathbb{B},\boldsymbol{t}}, \boldsymbol{l}'_{\mathbb{B},\boldsymbol{t}}|l_t)$ (Eq. 2.27, 2.30) with the prior $p(l_t)$ (Eq. 2.28) yields a $\mathcal{GP}$ posterior surrogate of l_t: $p(l_t \mid \boldsymbol{l}_{\mathbb{B},\boldsymbol{t}}, \boldsymbol{l}'_{\mathbb{B},\boldsymbol{t}})$. Its mean function μ and covariance function $\tilde{k}$ are:

$$\mu(\lambda) = \underbrace{\left[k(\boldsymbol{\lambda},\lambda)\, {}^{\partial}k(\boldsymbol{\lambda},\lambda)\right] \left(\begin{bmatrix} k(\boldsymbol{\lambda},\boldsymbol{\lambda}) + \sigma_{l_t}^2 \boldsymbol{I} & k^{\partial}(\boldsymbol{\lambda},\boldsymbol{\lambda}) \\ {}^{\partial}k(\boldsymbol{\lambda},\boldsymbol{\lambda}) & {}^{\partial}k^{\partial}(\boldsymbol{\lambda},\boldsymbol{\lambda}) + \sigma_{l'_t}^2 \boldsymbol{I} \end{bmatrix} \right)^{-1}}_{=:\boldsymbol{v}^\top(\lambda)} \begin{bmatrix} \boldsymbol{l}_{\mathbb{B},\boldsymbol{t}} \\ \boldsymbol{l}'_{\mathbb{B},\boldsymbol{t}} \end{bmatrix} \tag{2.34}$$

and

$$\tilde{k}(\lambda,\lambda') = k(\boldsymbol{\lambda},\boldsymbol{\lambda}') - \boldsymbol{v}^\top(\lambda) \begin{bmatrix} k(\boldsymbol{\lambda},\lambda') \\ {}^{\partial}k(\boldsymbol{\lambda},\lambda') \end{bmatrix}, \tag{2.35}$$

where ${}^{\partial}k$ is the derivative with respect to the first entry, k^{∂} the derivative with respect to the second entry, and ${}^{\partial}k^{\partial}$ the derivative with respect to both entries. Between each of the N measurements at different λ's μ is a cubic spline. A cubic spline is a continuous piece-wise cubic function, having continuous first and second derivatives. That μ is a cubic spline is shown by (Mahsereci and Hennig, 2015, Eq. 8) by proving that μ has at most three non-vanishing derivatives . For N values of $l_{\mathbb{B},t}$ and $l'_{\mathbb{B},t}$ a local minimum of μ can be found in $\mathcal{O}(N)$. This is because only the cubic spline between each measurement position must be considered. The local minimum of a cubic spline can be computed inexpensively and straightforward.

During the line search, and after evaluating the two values of $l_{\mathbb{B},t}$ and $l'_{\mathbb{B},t}$ at the current candidate position λ_{cand} the next candidate position has to be chosen. Therefore, a list of candidate λ's ($\boldsymbol{\lambda}_{\text{cand}}$) consisting of less than or equal to N local minimizers of μ and one additional extrapolation position at $\lambda_{\max} + \alpha$ is generated, where $\lambda_{\max}$ is the currently largest evaluated λ and α an extrapolation step size, which is set to one in the beginning and then doubled after each extrapolation step.

Two factors are considered in selecting the next λ_{cand} from $\boldsymbol{\lambda}_{\text{cand}}$. First, the expected improvement: this measures the expected amount by which l_t might be smaller than the smallest of the N measurements $l_{\min} = \min(l(\lambda_{\text{cand}}) : \lambda_{\text{cand}} \in \boldsymbol{\lambda}_{\text{cand}})$:

$$u_{\text{EI}}(\lambda) = \mathbb{E}_{p(l_t|\boldsymbol{l}_{\mathbb{B},\boldsymbol{t}}, \boldsymbol{l}'_{\mathbb{B},\boldsymbol{t}})}[\min(0, l_{\min} - l_t(\lambda))]. \tag{2.36}$$

Second, a probabilistic belief over the first and second Wolf conditions is considered. Let $a(\lambda)$ and $b(\lambda)$ be normal distributed random variables representing a probabilistic belief over the first and second Wolfe condition, respectively:

$$p(a(\lambda), b(\lambda)) = \mathcal{N}\left(\begin{bmatrix} a(\lambda) \\ b(\lambda) \end{bmatrix}; \begin{bmatrix} \mu(0) - \mu(\lambda) + c_1 \lambda \mu'(0) \\ \mu'(\lambda) - c_2 \mu'(0) \end{bmatrix}, \begin{bmatrix} C^{aa}(\lambda) & C^{ab}(\lambda) \\ C^{ba}(\lambda) & C^{bb}(\lambda) \end{bmatrix} \right), \tag{2.37}$$

The exact form of the Covariances C^{**} is found in (Mahsereci and Hennig, 2015, Equation 13). The probability for both conditions to hold is given by:

$$p^{\text{Wolfe}}(\lambda) = p(a(\lambda) > 0 \wedge b(\lambda) > 0), \tag{2.38}$$

which is calculated by integration over $p(a(\lambda), b(\lambda))$.
The λ out of $\boldsymbol{\lambda}_{\text{cand}}$ maximizing the product $p^{\text{Wolfe}}(\lambda) \cdot u_{\text{EI}}(\lambda)$ is chosen as the next measurement point λ_{cand}. An exemplary determination of $p^{\text{Wolfe}}(\lambda) \cdot u_{\text{EI}}(\lambda)$ is illustrated in Figure 2.5 With the $l_{\mathbb{B},t}(\lambda_{\text{cand}})$ and $l'_{\mathbb{B},t}(\lambda_{\text{cand}})$ measurements, the GP is

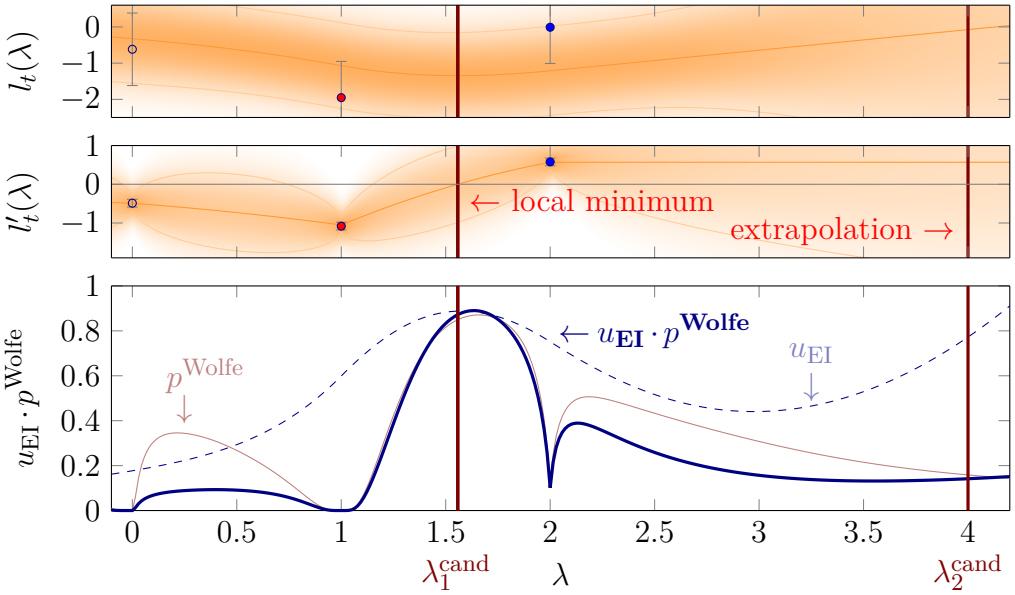


Figure 2.5: *Candidate selection of PLS. Top:* GP posterior. The mean is solid orange, two standard deviations in thinner solid orange, local pdf marginal as shading. The red and the blue point are $l_{\mathbb{B},t}$ evaluations, collected by the line search. *Middle:* GP marginal posterior of corresponding derivatives. The colors are the same as in the top plot. In all three plots, the locations of the *two* candidate points are indicated as vertical dark red lines. The left one at about $\lambda_1^{\text{cand}} \approx 1.54$ is a local minimum of the posterior mean between the red and blue points. The right one at $\lambda_2^{\text{cand}} = 4$ is a candidate for extrapolation. *Bottom:* Decision criterion in arbitrary scale: the expected improvement u_{EI} (Eq. 2.36) is shown in dashed light blue, the Wolfe probability p^{Wolfe} (Eq. 2.37) in light red, and their decisive product in solid dark blue. For illustrative purposes, all criteria are plotted for the whole λ-space. In practice, solely the values at λ_1^{cand} and λ_2^{cand} are computed and compared, and the candidate with the higher $u_{\text{EI}} \cdot p^{\text{Wolfe}}$ value is chosen for evaluation. In this example, this would be the candidate at λ_1^{cand}. Caption and Figure from (Mahsereci and Hennig, 2015); adapted with permission.

Algorithm 5 Sketch of Probabilistic Line Search adapted from Mahsereci and Hennig (2015). The weight update is omitted for clarity; consequently, only the internal line search routine is described. The subscript $_t$s are omitted. $\odot$ is the element-wise multiplication.

Input:

symbol	explanation	default value
$l_{\mathbb{B}}$	mini-batch loss function) along direction $\boldsymbol{d}_t$	–
$l'_{\mathbb{B}}$	derivative of $l_{\mathbb{B}}$	–
$\sigma_{l(0)}$	estimated full-batch loss' variance at position 0	estimation over mini-batch losses of first batch
$\sigma_{l'(0)}$	estimated full-batch directional derivative's variance at position 0	estimation over mini-batch derivatives of first batch
c_W	threshold for wolfe probability (Eq. 2.38)	0.3
c_1	first wolfe (sufficient decrease) constant	0.05
c_2	second wolfe (curvature) constant	0.8
$N_{\max}$	maximal number of $l_{\mathbb{B}_i}$ evaluations	10

$\mathbb{B}_0 \leftarrow$ sampleBatch()
$GP \leftarrow$initGaussianProcess($l_{\mathbb{B}_0}(0)$,$l'_{\mathbb{B}_0}(0)$, $\sigma_{l(0)}$, $\sigma_{l'(0)}$)
$\boldsymbol{\lambda}, \boldsymbol{l}_{\mathbb{B}}, \boldsymbol{l}'_{\mathbb{B}} \leftarrow$initStorage(0, $l_{\mathbb{B}_0}(0)$, $l'_{\mathbb{B}_0}(0)$) ▷ with observed measur. at $\lambda = 0$
$\lambda \leftarrow 1$ ▷ initial learning rate candidate
while budget $N_{\max}$ not used and no Wolfe-point found do
  $\mathbb{B}_i \leftarrow$ sampleBatch()
  $\boldsymbol{\lambda}, \boldsymbol{l}_{\mathbb{B}}, \boldsymbol{l}'_{\mathbb{B}} \leftarrow$updateStorage($\lambda$, $l_{\mathbb{B}_i}(\lambda)$, $l'_{\mathbb{B}_i}(\lambda)$)
  $GP \leftarrow$updateGP($l_{\mathbb{B}_i}(\lambda)$,$l'_{\mathbb{B}_i}(\lambda)$)
  $\boldsymbol{p}^{\text{Wolfe}} \leftarrow$probWolfe($\boldsymbol{\lambda}$,$c_1$,$c_2$ GP) ▷ get Wolfe prob. (Eq. 2.38) for all λ_i in $\boldsymbol{\lambda}$
  if any element $p_j \in \boldsymbol{p}^{\text{Wolfe}} > c_W$ **then**
    return λ_j ▷ return Wolfe-Point λ_j
  else
    $\boldsymbol{\lambda}_{\text{cand}} \leftarrow$computeCandidates($GP$) ▷ new learning rate candidates
    $\boldsymbol{e} \leftarrow$expectedImprovement($\boldsymbol{\lambda}_{\text{cand}}$, GP)
    $\boldsymbol{p} \leftarrow$probWolfe($\boldsymbol{\lambda}_{\text{cand}}$,$c_1$,$c_2$, GP)
    $\lambda \leftarrow$ where $(\boldsymbol{e} \odot \boldsymbol{p})$ is maximal ▷ find best candidate among $\boldsymbol{\lambda}_{\text{cand}}$
  end if
end while
return λ_i in $\boldsymbol{\lambda}$ with lowest GP mean since $\forall p_j \in \boldsymbol{p}^{\text{Wolfe}} : p_j \leq c_W$

updated, and then a new λ_{cand} is estimated. This is repeated until either a λ_{cand}

is found for which $p^{\text{Wolfe}}(\lambda) > c_W$ holds or if the budget for new measurements is exhausted. The latter defaults to 10. In the first case, λ_{cand} (called a Wolfe Point) is returned. In the second case, the $\lambda \in \boldsymbol{\lambda}$ with the lowest $\mu(\lambda)$ is returned.

A pseudo-code of the whole algorithm containing the essentials can be found in Algorithm 5.

PLS is essentially based on two assumptions: first, that sample losses and their gradients are Gaussian distributed, and second, that l_t can be locally approximated with cubic splines. The PLS paper does not provide empirical evidence that these assumptions hold; however, as we will see in Chapter 4, 5, and 6 a cubic spline approximation is appropriate because l_t almost always behaves locally quadratically. Further, the results of (Smith *et al.*, 2018) and (Smith and Le, 2018) support that losses and gradients could be Gaussian distributed. No theoretical convergence results are provided for this algorithm, yet.

2.7 Descent Direction choosing Methods

For numerical optimization algorithms, not only the step size along a direction is of relevance, but also the direction itself. In classical deterministic optimization, the direction of steepest descent (i.e., the negative gradient), the conjugate gradient direction, or the Newton direction (Nocedal and Wright, 2006; Luenberger *et al.*, 1984) are commonly used candidates. Especially in the case of a quadratic loss, the conjugate gradient direction and the Newton direction lead to significantly faster convergence but are harder to compute than the direction of steepest descent. Let $E(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^\top \boldsymbol{Q}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$ be a quadratic loss, where $\boldsymbol{\theta}^*$ is the minimizer, $\boldsymbol{\theta}$ are the parameters to optimize - both of dimension N -, and $\boldsymbol{Q}$ is a square matrix of dimension N×N. When we consider exact line searches on $E(\boldsymbol{\theta})$ and using the steepest descent direction, the convergence depends on the condition number $r = \frac{v_{\max}(\boldsymbol{Q})}{v_{\min}(\boldsymbol{Q})}$ of the problem, where v denotes the eigenvalue. The decrease of loss per step is bounded by $E(\boldsymbol{\theta}_{t+1}) \leq (\frac{r-1}{r+1})^2 E(\boldsymbol{\theta}_t)$. Consequently for ill-conditioned problems, an exact line search following the steepest descent direction can take unreasonably long to converge. An exact line search using the conjugate gradient direction, i.e., a direction $\boldsymbol{d}_t$ for which $\boldsymbol{d}_{t-1}^\top \boldsymbol{Q} \boldsymbol{d}_t = 0$ holds, requires N steps to converge in the worst case, while using the Newton direction requires at most one one step (Luenberger *et al.*, 1984).

Applying the introduced directions is only suitable if better update steps compensate for the longer time needed for their computation. However, in the stochastic scenario, already calculating the exact gradient $\nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\theta}_t)$ is impractically expensive. Consequently, the computation of a conjugate gradient direction, which is based on the gradient, is also impractical. One could use noisy estimates of these directions. However, calculating the conjugate gradient direction requires the gradient with respect to the parameters of the previous update step, but on the mini-batch of the

current update step. This calculation has to be done in addition and is expensive. Computing the noisy Newton direction of a batch is often impractical because GPU memory is usually too small to hold the Hessian.

Consequently, other directions, heuristically proven to be practical, are considered in the stochastic scenario. They usually focus on reducing the noise of consecutive update steps and consequently follow a smoother optimization path. This is done by preferring less noisy dimensions. This effect is exemplified in Figure 2.6. Such methods are called adaptive methods. Most adaptive approaches compute heuristics for each dimension independently and neglect dependencies between dimensions entirely. The two most prominent adaptive methods, **SGD with momentum** and **ADAM** are introduced below:

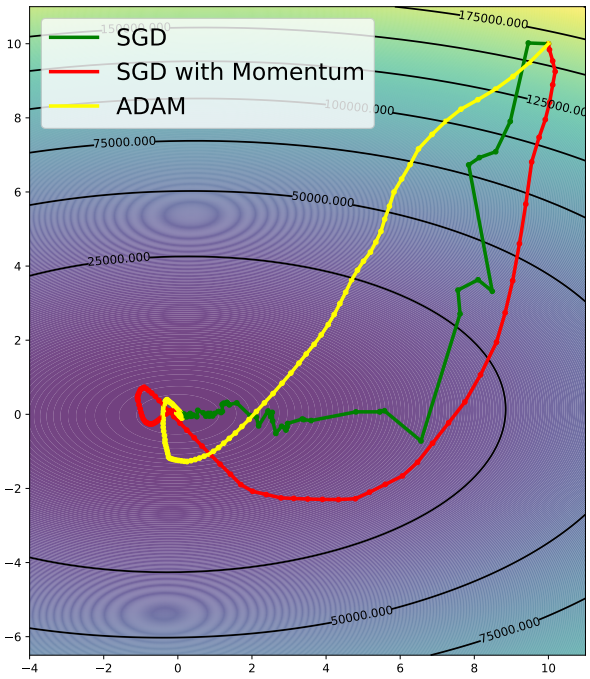


Figure 2.6: A noisy quadratic toy example that gives insight into the behavior of SGD's, SGD with momentum's and ADAM's training process. ADAM and SGD with momentum choose a smoother path. The plotted loss is the average over 100 quadratics whose parameters are sampled from a normal distribution. One of these quadratics is chosen for each update step of an optimizer.

SGD with momentum: The most widely used direction in the stochastic scenario is the momentum direction, an exponential moving average over all previously measured mini-batch gradients. SGD using this direction is called SGD with momentum. The update rule changes to (compare to Algorithm 1):

$$\boldsymbol{m}_t = \begin{cases} \boldsymbol{g_t} & \text{if } t = 0 \\ \beta \boldsymbol{m}_{t-1} + \boldsymbol{g_t} & \text{else} \end{cases} \tag{2.39}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \boldsymbol{m}_t, \tag{2.40}$$

where $\boldsymbol{m}_t$ is an additional momentum term that must be held in memory, and $\beta \in [0,1)$ is a factor that regulates the influence of previously measured gradients. Typically, β is chosen to be 0.9. A pictorial explanation of this method is a ball rolling down the loss landscape, building up some momentum influenced by the acceleration $\boldsymbol{g_t}$ and the drag $-(1-\beta)\ \boldsymbol{m}_{t-1}$. Equation 2.39 does influence not only the direction but also the step size as the norm of the direction changes. If the gradient is constant, each element $(\boldsymbol{m}_t)_i$ converges to $\frac{(\boldsymbol{g}_t)_i}{1-\beta}$ since $\boldsymbol{m}_t = \sum_{t=0}^{T} \beta^{T-t} \boldsymbol{g_t}$. Considering $\boldsymbol{g_t}$ as a vector of random variables, $(\boldsymbol{m}_t)_i$ converges to $\frac{E[(\boldsymbol{g}_t)_i]}{1-\beta}$. This implies that dimensions for which $|E[(\boldsymbol{g}_t)_i]|$ is small are less considered in the update direction. This is particularly apparent if the noise leads to frequent sign changes in a dimension. The latter is not preferable because in this case SGD oscillates in this dimension and, thus, does not make much improvement. The pseudo-code of SGD with momentum is given in Algorithm 6. SGD with momentum consistently outperforms SGD and is the most widely used optimizer in deep learning. In the deterministic scenario, it is usually argued that momentum performs well since it can escape local minima. This intuition tends not to hold in the deep learning scenario, as it is assumed that all local minima are almost as good as a global one (Kawaguchi, 2016; Kawaguchi and Kaelbling, 2020; Fort and Jastrzebski, 2019). Although momentum is a popular method, its theoretical analysis is limited in the stochastic setting (Sutskever *et al.*, 2013; Arnold *et al.*, 2019). Surprisingly, some analyses even show slower convergence rates for $\beta > 0$ (Schmidt *et al.*, 2011); the latter implies that SGD with momentum does not have to converge faster than SGD. Recent work might provide even better convergence analysis. However, the theoretical analysis of adaptive methods is not our primary focus.

Algorithm 6 Stochastic Gradient Descent with momentum (SGD with momentum) (Polyak, 1964)

Input:

symbol	explanation	default value
$\mathcal{L}_{\mathbb{B}}$	mini-batch loss	–
$\lambda(t)$	learning rate schedule	usually starting with 10^{-1} to 10^{-4}
β	momentum parameter	0.9 to 0.99
$\boldsymbol{\theta}_0$	initial parameters	usually sampled with Xavier/-Glorot initialization (Glorot and Bengio, 2010)
-	*stopping criterion*	usually a fixed number of update steps

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
$t \leftarrow 0$
$\boldsymbol{m} \leftarrow \mathbf{0}$
while *stopping criterion* not met **do**
  $\mathbb{B}_t \leftarrow$ sampleBatch()
  $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} + \boldsymbol{g}_{\mathbb{B}_t}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda(t)\boldsymbol{m}$
  $t \leftarrow t + 1$
end while
return $\boldsymbol{\theta}$

ADAM Adaptive Momentum estimation (ADAM) (Kingma and Ba, 2015) is another widespread heuristic to choose a less noise-prone update direction than the noisy mini-batch gradient. This can be exemplarily seen in ADAM's optimization trajectory in Figure 2.6. ADAM is derived from the popular methods RMSProp (see 2.8) (Tieleman and Hinton, 2012) and AdaGrad (Duchi *et al.*, 2011). As in these approaches, each gradient dimension is considered independently, and strengthened or weakened depending on the estimated noise. In particular, ADAM approximates the first mathematical moment, also known as the mean $E[\boldsymbol{g}_t]$, and the second moment, also known as the uncentered variance $E[\boldsymbol{g}_t^2]$, for the gradient of each parameter, respectively. Note that the square of $\boldsymbol{g}_t^2$ is element-wise. Here, $\boldsymbol{g}_t$ is interpreted as a vector of independent random variables. Then Kingma and Ba (2015) take the fraction of the first and the root of the second moment to get the so-called signal-to-noise ratio: $\frac{E[\boldsymbol{g}_t]}{\sqrt{E[\boldsymbol{g}_t^2]}}$. This ratio is an indicator for the distribution of $\boldsymbol{g}_t$ independently of any linear scale change of $\boldsymbol{\theta}$. An exponential moving average approximates the first moment $\boldsymbol{m}$ over the last mini-batch gradients:

$$E[(\boldsymbol{g}_t)_i] \approx (\boldsymbol{m_t})_i = \beta_1 (\boldsymbol{m}_{t-1})_i + (1-\beta_1)(\boldsymbol{g}_{\mathbb{B},t})_i, \tag{2.41}$$

where i indicates the i-th element. Similarly, the second moment $\boldsymbol{v}$ of the gradient is approximated by an exponential moving average over the squared mini-batch gradient:

$$E[(\boldsymbol{g}_t)_i^2] \approx (\boldsymbol{v_t})_i = \beta_2(\boldsymbol{v}_{t-1})_i + (1-\beta_2)(\boldsymbol{g}^2_{\mathbb{B},t})_i. \tag{2.42}$$

These approximations are improved by eliminating the initialization bias originating from a zero vector initialization of $\boldsymbol{m}$ and $\boldsymbol{v}$. In the case of $\boldsymbol{v}$, the discrepancy between $E[(\boldsymbol{g}_t)_i^2]$ and $E[(\boldsymbol{v}_t)_i^2]$ is considered. $E[(\boldsymbol{v}_t)_i^2]$ rather than $(\boldsymbol{v}_t)_i^2$ is of relevance since the average error that appears when $E[(\boldsymbol{g}_t)_i^2]$ is approximated by $(\boldsymbol{v}_t)_i^2$ has to be considered:

$$\begin{aligned}
E[\boldsymbol{v}_t] &= E\left[\beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2\right] \\
&\text{In iterative from:} \\
&= E\left[(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i}\boldsymbol{g}_i^2\right] \\
&\text{Assuming } \boldsymbol{g}_i^2 \text{ to be stationary and introducing error term } \zeta: \\
&= E[\boldsymbol{g}_t^2](1-\beta_2)\sum_{i=1}^{t}(\beta_2^{t-i}) + \zeta \\
&\text{Factoring the brackets out:} \\
&= E[\boldsymbol{g}_t^2](\sum_{i=0}^{t-1}\beta_2^t - \sum_{i=1}^{t}\beta_2^t) + \zeta \\
&= E[\boldsymbol{g}_t^2](1-\beta_2^t) + \zeta,
\end{aligned} \tag{2.43}$$

where all operations are point-wise, and ζ is the error term that arises if $E[\boldsymbol{g}_i^2]$ is non-stationary. In practice, ζ is small because the exponential average assigns exponentially decreasing weights to gradients in the past but a large weight to the current gradient. The derivation of $E[\boldsymbol{m}_t]$ is done in the same way. The term $(1-\beta_2^t)$ is caused by initializing the running average with $\mathbf{0}$. Consequently, the division of $\boldsymbol{v}$ by $(1-\beta_2^t)$ leads to the correction of this initialization. The same holds for $\boldsymbol{m}$, which must be divided by $(1-\beta_1^t)$. The resulting error-corrected approximations are:

$$E[(\boldsymbol{g}_t)] \approx \hat{\boldsymbol{m}}_{\boldsymbol{t}} = \frac{\boldsymbol{m_t}}{1-\beta_1^t} \tag{2.44}$$

$$E[(\boldsymbol{g}_t^2)] \approx \hat{\boldsymbol{v}}_{\boldsymbol{t}} = \frac{\boldsymbol{v_t}}{1-\beta_2^t}. \tag{2.45}$$

Finally, the linear scale-invariant parameter update step is given by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \cdot \hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}} + \epsilon), \tag{2.46}$$

where ϵ is added to avoid division by zero. See Algorithm 7 for the complete pseudo code.

Since $\hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}}+\epsilon)$ approximates $\frac{E[\boldsymbol{g}_t]}{\sqrt{E[\boldsymbol{g}_t^2]}}$ its size is approximately bounded by ± 1 because $\left|\frac{E[\boldsymbol{g}_t]}{\sqrt{E[\boldsymbol{g}_t^2]}}\right| \leq 1$. The latter holds because $E[\boldsymbol{g}_t^2] = E[\boldsymbol{g}_t]^2 + \mathrm{VAR}[\boldsymbol{g}_t] \geq E[\boldsymbol{g}_t]^2$. $(\hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}}))_i$ is often interpreted as the signal-to-noise ratio for parameter i. If we further interpret noise as sign changes, the following can be intuitively deduced: If the past gradients were large, similar, and of the same sign, then $(\hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}}))_i$ is large, since in this case, $\sqrt{E[(\boldsymbol{g}_t^2)_i]} \approx E[(\boldsymbol{g}_t)_i]$ holds. In general, $\sqrt{E[(\boldsymbol{g}_t^2)_i]}$ grows equal to or faster than $E[(\boldsymbol{g}_t)_i]$. Especially in the case of sign changes, $E[(\boldsymbol{g}_t^2)_i]$ grows significantly faster. This holds since the expectation is over the always positive distance to zero. Consequently, the i-th parameter's influence

Algorithm 7 ADAM, an algorithm for stochastic optimization. See section 4.4 for details. Algorithm adapted from Kingma and Ba (2015). $\sqrt{}$ denotes the element-wise root and $\odot$ denotes the element-wise multiplication.

Input:

symbol	explanation	default value
$\boldsymbol{\theta}_0$	initial parameters	–
$\mathcal{L}_{\mathbb{B},t}$	mini-batch loss	–
λ	learning rate	10^{-3}
β_1	momentum factor for the gradient	0.9
β_2	momentum factor for the element-wise squared gradient	0.999
ϵ	for numerical stability	10^{-8}

$\boldsymbol{m} \leftarrow \boldsymbol{0}$ ▷ Initialize initial 1st moment vector
$\boldsymbol{v} \leftarrow \boldsymbol{0}$ ▷ Initialize initial 2nd moment vector
$t \leftarrow 0$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
while $\boldsymbol{\theta}$ not converged **do**
  $\mathbb{B}_t \leftarrow$ sampleBatch()
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}_t}(\boldsymbol{\theta})$ ▷ Get mini-batch loss gradient
  $\boldsymbol{m} \leftarrow \beta_1 \cdot \boldsymbol{m} + (1-\beta_1) \cdot \boldsymbol{g}$ ▷ Update biased first moment estimate
  $\boldsymbol{v} \leftarrow \beta_2 \cdot \boldsymbol{v} + (1-\beta_2) \cdot \boldsymbol{g} \odot \boldsymbol{g}$ ▷ Update biased second raw moment estimate
  $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m}/(1-\beta_1^{t+1})$ ▷ Compute bias-corrected first moment estimate
  $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}/(1-\beta_2^{t+1})$ ▷ Compute bias-corrected second raw moment estimate
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \cdot \hat{\boldsymbol{m}}/(\sqrt{\hat{\boldsymbol{v}}}+\epsilon)$ ▷ Update parameters
  $t \leftarrow t+1$
end while
return $\boldsymbol{\theta}$ ▷ Resulting parameters

on the search direction converges to zero if the noise is high and converges to one if the noise is low.

(Kingma and Ba, 2015) show that ADAM converges on a sum over convex mini-batch loss functions, with bounded gradients, bounded update step sizes, and a learning rate decay of $\frac{1}{\sqrt{t}}$.

Combining line searches with direction given algorithms such as ADAM and SGD with momentum raises the problem that direction heuristics rely on very small learning rates to obtain sufficiently good low noise direction estimate from past gradients. Line searches estimate larger learning rates; thus, the gradient estimates become worse. Further, it is unclear whether applying line searches along such directions is fruitful.

2.8 Further Optimization Methods

In the following section, the optimization methods **RMSprop**, **SGD-HD**, and **ALI-G** are introduced in short. These methods are not as fundamental for the line search field or the adaptive direction field as the already introduced algorithms. However, the method we will present in Chapter 4 will be compared to these, as it was requested by external experts.

RMSprop Root mean square propagation (RMSprop or RMSP) (Tieleman and Hinton, 2012) is an adaptive optimization method similar to ADAM, often used for recurrent neural networks. It differs form ADAM (2.7) in that it uses only the second momentum estimate in combination with the gradient for an update step:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \lambda \frac{1}{\sqrt{\boldsymbol{v}} + \epsilon} \boldsymbol{g}_t. \tag{2.47}$$

See Algorithm 7 line 12 for comparison.

SGD-HD The idea of Hypergradient Descent (SGD-HD) (Baydin *et al.*, 2018) is to compute the derivative with respect to the learning rate and in a next step update the learning rate by using a gradient descent algorithm on a hyper level. The derivative with respect to λ is simply given as:

$$\frac{\partial}{\partial \lambda} \mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_{t-1} - \lambda \boldsymbol{g}_{t-1}) = -\mathcal{L}'_{\mathbb{B},t}(\boldsymbol{\theta}_{t-1} - \lambda \boldsymbol{g}_{t-1}) \boldsymbol{g}_{t-1} = -\mathcal{L}'_{\mathbb{B},t}(\boldsymbol{\theta}_t) \boldsymbol{g}_{t-1} = -\boldsymbol{g}_t^\top \boldsymbol{g}_{t-1}. \tag{2.48}$$

This is an equivalent derivation to that of (Baydin *et al.*, 2018), which -in our opinion- is clearer. Now a gradient descent step is performed to update λ:

$$\lambda_{t+1} = \lambda_t + \alpha \boldsymbol{g}_t^\top \boldsymbol{g}_{t-1}, \tag{2.49}$$

where α is the learning rate of the hyper level gradient descent optimizer. This algorithm can be considered as a line search approach that changes the learning rate such that a step closer to the minimum of $l_{\mathbb{B},t}$ is taken and after that the line search is terminated. To clarify this approach Figure 2.7 describes the different data flows of SGD and SGD-HD. (Baydin *et al.*, 2018) argue that the sensitivity of α is lower than that of λ and therefore easier to tune. However, in some sample experiments, which we performed on training scenarios other than those of (Baydin *et al.*, 2018), it was hard to find a working α (Elkersh *et al.*, 2020).

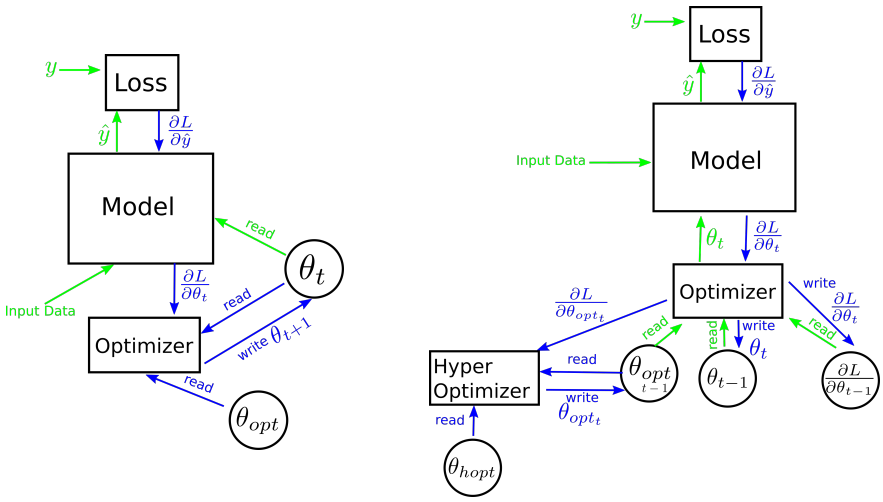


Figure 2.7: Figurative comparison of data flows in SGD (left) and SGD-HD (right). In our case, the parameters of the optimizer θ_{opt} is just the learning rate, but in general can be the parameters of any optimizer, which have a derivative.

ALI-G Adaptive Learning rates for Interpolation with Gradients (ALI-G) (Berrada *et al.*, 2020) utilizes a linear approximation of $l_{\mathbb{B},t}$ to perform an update step to the point where the approximation becomes zero. Under the interpolation assumption (Assumption 1 page 16) and the assumption that $\mathcal{L}$'s minimum value is 0, they show that linear approximations of $l_{\mathbb{B},t}$ converge in the stochastic convex setting. For each update step, λ is automatically inferred by:

$$\lambda_{\text{upd}} = \min\left(\frac{\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)}{||\boldsymbol{g}_{\mathbb{B},t}||^2 + \epsilon}, \eta\right) = \min\left(\frac{l_{\mathbb{B},t}(0)}{l'_{\mathbb{B},t}(0) + \epsilon}, \eta\right), \tag{2.50}$$

where ϵ is for numerical stability, η is a maximal step size, and $||\boldsymbol{g}_{\mathbb{B},t}||^2$ is the directional derivative in $\boldsymbol{g}_{\mathbb{B},t}$ direction. From another perspective, this algorithm is the Newton method applied to find a root in gradient direction bounded by a maximal step size. The resulting learning rate is also known as the Polyak Step Size (Polyak, 1969).

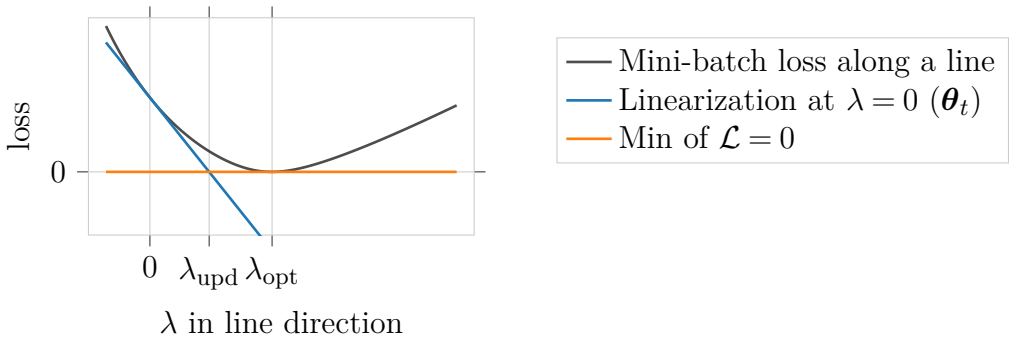


Figure 2.8: Illustration of ALI-G's update rule. A linear function approximates the mini-batch loss along a line in mini-batch gradient direction. An update step to the zero point of the approximation is performed. Illustration adapted from Berrada *et al.* (2020).

COCOB The COntinuous COin Betting (COCOB) (Orabona and Tommasi, 2017) optimizer based on gambling theory. It simulates a gambler who repeatedly bets that the next gradient has the same sign as the performed weight update. The latter is performed per dimension. The reward is the weight update times the gradient. Consequently, if both have the same sign, the reward is positive, otherwise it is negative. A fraction of the accumulated reward is then used as the next update step. This results in large update steps if the gradient noise is low, and small update steps if the gradient noise is high. See (Orabona and Tommasi, 2017) for further information.

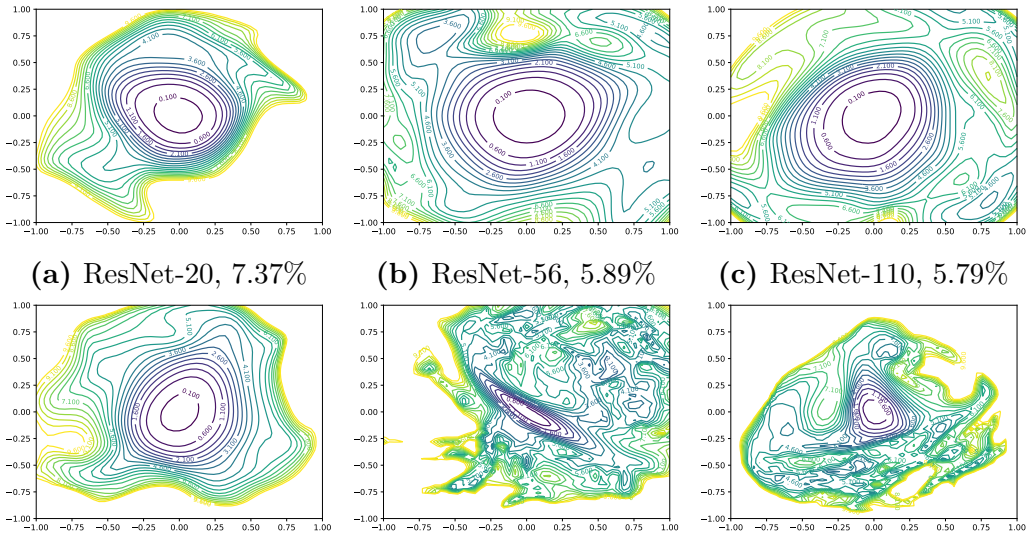
2.9 The simple Loss Landscape

Considering the full-batch loss formula 2.9, loss landscapes of deep learning problems, in general, can be highly non-convex and thus challenging to optimize. As seen throughout the chapter, all optimization methods presented only converge under specific assumptions. However, it is rarely verified whether and to what extent these assumptions hold in practice. Fortunately, there is at least some empirical evidence that loss landscapes of image classification problems typically considered tend to have simple shapes, making the assumptions more plausible (Li *et al.*, 2018; Xing *et al.*, 2018; Chae and Wilke, 2019; Mahsereci and Hennig, 2015; Goodfellow and Vinyals, 2015; Fort and Jastrzebski, 2019; Draxler *et al.*, 2018; Hille and Mutschler, 2020; Hochreiter and Schmidhuber, 1994; Keskar *et al.*, 2017):

(Li *et al.*, 2018) presents a simple method to visualize and compare different minima by considering 2D contour plots:

$$C(\alpha, \beta) = \mathcal{L}(\boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\mu}), \tag{2.51}$$

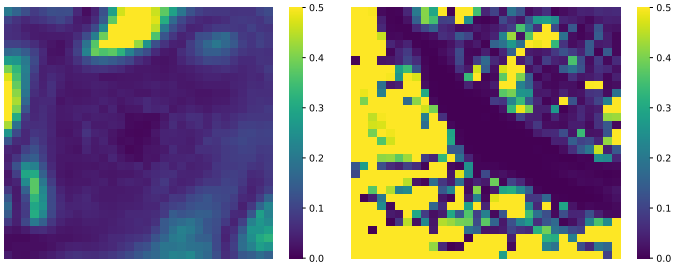
where $\boldsymbol{\theta}^*$ is the center point of the contour and α, β are step sizes in the direction of $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$, respectively. The contour directions are chosen randomly from a Gaussian distribution. To overcome the problem that weights of NNs can be rescaled without changing the output, each dimension/weight of the direction is scaled by the Frobenius norm of the convolution filter it is part of. The latter is referred to as *filter normalization*. Furthermore, the smoothness of minima is measured by considering the absolute ratio $\left|\frac{\lambda_{\min}}{\lambda_{\max}}\right|$ of the minimal and maximal eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ of the Hessian. This is comprehensible since the eigenvalue is the directional curvature (second derivative) in eigenvector direction. The following observations have been made by (Li *et al.*, 2018), which, however, might be questionable for their application to generality based on their limited number of samples: For classical networks that do not have skip-connections, such as a VGG-Net (Simonyan and Zisserman, 2015), the loss landscape behaves roughly convex if the network is shallow; the deeper it becomes, the more chaotic it becomes. However, when skip-connections are used, the near convex shape is maintained even when the network becomes deeper. In addition, skip-connections tend to lead to flat minima. Some example visualizations of minima of networks with and without skip-connections



(a) ResNet-20, 7.37% **(b)** ResNet-56, 5.89% **(c)** ResNet-110, 5.79%

(d) ResNet-20-NS, 8.18% **(e)** ResNet-56-NS, 13.31% **(f)** ResNet-110-NS,16.44%

Figure 2.9: 2D visualizations of the loss landscape around minima of ResNets (He *et al.*, 2016) and VGGNet-like ResNets (Simonyan and Zisserman, 2015) without skip-connections (NS). Different depths of these networks are considered. One can observe that -at least for the considered perpendicular random directions $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$- minima of ResNet are much smoother and more convex than those of the VGG-like ResNets. The number after the model name indicates the test error. Figure from (Li *et al.*, 2018).



(a) ResNet-56 **(b)** Resnet-56 no shortcut

Figure 2.10: For each point in the filter-normalized surface plots of Figure 2.9 (ResNet-56, middle row), the absolute value of the ratio of the maximum and minimum eigenvalue of the Hessian are plotted. The eigenvalue is the directional curvature in eigenvector direction. One can observe that with skip-connections, the ratio of eigenvalues (curvatures) is significantly lower in most cases, and therefore, the loss is smoother in all directions. Figure from (Li *et al.*, 2018).

can be found in Figure 2.9; the corresponding eigenvalue analysis is shown in Figure 2.10.

Several works support (Li *et al.*, 2018)'s observation that the loss landscape is rather simple, albeit from different perspectives: (Xing *et al.*, 2018) shows that the full-batch loss is roughly convex along SGD update step directions and that SGD bounces of walls of a *valley-like structure* during training. (Chae and Wilke, 2019) reveal that the mini-batch loss along update step directions is locally almost parabolic for simple examples. Similarly, (Mahsereci and Hennig, 2015) assume that cubic splines can fit the full-batch loss along negative gradient directions (see Section 2.6.5). (Goodfellow and Vinyals, 2015) points out that optimizers do not encounter any significant areas of increasing loss values in the loss landscape on a straight path from initialization to solution. Other works showed empirically that the loss landscape is not fully convex, as several minima exist; however, they are connected by low-loss sub-spaces: (Fort and Jastrzebski, 2019) models the loss landscape as a set of high-dimensional wedges and demonstrates the existence of a low-loss subspace connecting a set of minima. Similarly, (Draxler *et al.*, 2018) constructs continuous low-loss paths between minima and suggests that minima are best viewed as points on a single connected low-loss sub-space. In (Hille and Mutschler, 2020) we combined the approaches of both works and measured as well as visualized two-dimensional low-loss sub-spaces with mesh-grids. Further on, (Hochreiter and Schmidhuber, 1994) introduces a simple algorithm to find connected flat regions. (Keskar *et al.*, 2017) shows that wide minima are found by SGD with small batch size, whereas, with increasing batch size the area around the found minimum becomes sharper.

2.10 The Relation of Batch Sizes and Learning Rates

Besides the learning rate λ, the batch size $|\mathbb{B}|$ is important in the stochastic scenario since it controls the noise in the optimization process. This section considers the mutual influence of λ and $|\mathbb{B}|$. (Smith and Le, 2018) introduces the expression:

$$\nu = \lambda \left(\frac{|\mathbb{D}|}{|\mathbb{B}|} - 1 \right), \tag{2.52}$$

where ν is the gradient *noise scale*, which is a factor that scales the covariance matrix of mini-batch gradients in simplified terms. A larger covariance matrix indicates larger fluctuations (noise) of the gradient and, thus, is an indicator of the error of using mini-batch losses instead of the full-batch loss. This formula reveals how the optimal step size depends on λ, $|\mathbb{B}|$ and the size the training set size $|\mathbb{D}|$. Further on, the formula explains why increasing the batch size has a similar effect to decreasing the learning rate in practice. The latter has been empirically shown for typical classification tasks by (Smith *et al.*, 2018). We will consider this effect again in Chapter 5.

In the following, we will derive expression 2.52 in detail and introduce the assumptions on which it is built. We have adapted and supplemented (Smith and Le, 2018)'s derivation to make it syntactically more convenient and significantly easier to understand. First, the SGD gradient update is rephrased as follows:

$$\Delta\boldsymbol{\theta} = -\lambda(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) + \underbrace{(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}))}_{\substack{\text{mini-batch gradient} \\ \text{error } \boldsymbol{\alpha}}}), \tag{2.53}$$

where $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is the full-batch gradient, and $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathbb{B}}$ is its mini-batch gradient estimate. $\mathcal{L}_{\mathbb{B}}$ is the mean of sample losses L_i (see Equation 2.9 and 2.10). All losses and gradients are $\in \mathbb{R}^N$, where N represents the number of parameters in $\boldsymbol{\theta}$. Each $(\nabla_{\boldsymbol{\theta}} L_i)_n \in \mathbb{R}$ (the sample loss gradient of parameter n) is interpreted as a Gaussian distributed random variable with mean $(\nabla_{\boldsymbol{\theta}}\mathcal{L})_n$. (Smith and Le, 2018, Figure 9) empirically provides some evidence that this holds when using batch sizes larger than 30. The random vectors $\nabla_{\boldsymbol{\theta}} L_i$ are assumed to be i.i.d., and consequently have the same covariance matrix $\boldsymbol{K}_{\boldsymbol{\theta}}$ in $\mathbb{R}^{N\times N}$. $\boldsymbol{K}_{\boldsymbol{\theta}}$ describes the covariances of the gradients with respect to each parameter in $\boldsymbol{\theta}_n$. The i.i.d. assumption for the gradients is appropriate if, first, i.i.d dataset elements are assumed and, second, if a network does not have any exchange of information between consecutive inputs. The frequently used Batch Normalization Layer (Ioffe and Szegedy, 2015) leads to some exchange of information between inputs but does not harm in practice, since empirically, the following derivations still hold (Smith *et al.*, 2018). Under these

assumptions, the mini-batch gradient error $\boldsymbol{\alpha} \overset{2.53}{=} (\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}))$ is Gaussian distributed due to the central limit theorem of probability theory (Patrick, 1995, Page 357). The latter states that the distribution of the difference of the average of i.i.d. random variables to their identical mean values approximates a normal distribution for an increasing number of variables. In Equations 2.54 to 2.59 we derive that $\boldsymbol{E}[\boldsymbol{\alpha}] = \mathbf{0}$, and $\boldsymbol{E}[\boldsymbol{\alpha}\boldsymbol{\alpha}^\top] = \frac{1}{|\mathbb{D}|}(\frac{|\mathbb{D}|}{|\mathbb{B}|} - 1)\boldsymbol{K}_{\boldsymbol{\theta}}$, where $\boldsymbol{\alpha}\boldsymbol{\alpha}^\top$ is an outer product describing the covariance of $\boldsymbol{\alpha}$ since $\boldsymbol{E}[\boldsymbol{\alpha}] = \mathbf{0}$. We added these essential derivations because they are lacking in Smith and Le: Let us define the following:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{D}|} \sum_{i=1}^{|\mathbb{D}|} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}) \in \mathbb{R}^N \tag{2.54}$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{B}|} \sum_{j=1}^{|\mathbb{B}|} \nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta}) \in \mathbb{R}^N \tag{2.55}$$

$\nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta})$ are i.i.d. and normal distributed with the following expectation vector and covariance matrix:

$$\begin{aligned} &\boldsymbol{E}[\nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta})] := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\ &\boldsymbol{K}_{\boldsymbol{\theta}} \in \mathbb{R}^{N \times N} := \mathrm{COV}[\nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta})], \end{aligned}$$

where $\nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta})$ is now the expected value over a discrete distribution of random variables. It is simple to show that $\boldsymbol{E}[\boldsymbol{\alpha}] = \mathbf{0}$:

$$\begin{aligned} \boldsymbol{E}[\boldsymbol{\alpha}] &= \boldsymbol{E}\left[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\right] \\ &= \boldsymbol{E}\left[\frac{1}{|\mathbb{B}|} \sum_{j=1}^{|\mathbb{B}|} \nabla_{\boldsymbol{\theta}} L_j(\boldsymbol{\theta}) - \frac{1}{|\mathbb{D}|} \sum_{i=1}^{|\mathbb{D}|} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})\right] \\ &= \frac{|\mathbb{B}|}{|\mathbb{B}|} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) - \frac{|\mathbb{D}|}{|\mathbb{D}|} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\ &= \mathbf{0}. \end{aligned} \tag{2.56}$$

In the following, we derive that $\boldsymbol{E}[\boldsymbol{\alpha}\boldsymbol{\alpha}^\top] = \frac{1}{|\mathbb{D}|}(\frac{|\mathbb{D}|}{|\mathbb{B}|} - 1)\boldsymbol{K}_{\boldsymbol{\theta}}$ holds:

$$
\begin{aligned}
\boldsymbol{E}[\boldsymbol{\alpha}\boldsymbol{\alpha}^\top] =& \boldsymbol{E}\left[(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}))(\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}))^\top\right] \\
=& \boldsymbol{E}\left[(\frac{1}{|\mathbb{B}|}\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}) - \frac{1}{|\mathbb{D}|}\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta}))(\frac{1}{|\mathbb{B}|}\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}) - \frac{1}{|\mathbb{D}|}\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta}))^\top\right] \\
=& \underbrace{\boldsymbol{E}\left[\frac{1}{|\mathbb{B}|^2}\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})(\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}))^\top\right]}_{a} + \underbrace{\boldsymbol{E}\left[\frac{1}{|\mathbb{D}|^2}\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta})(\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta}))^\top\right]}_{b} \\
& \underbrace{- \boldsymbol{E}\left[\frac{2}{|\mathbb{B}||\mathbb{D}|}\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})(\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta}))^\top\right]}_{c}.
\end{aligned}
\tag{2.57}
$$

Now, we simplify a, b and c:

$$
\begin{aligned}
a &= \frac{1}{|\mathbb{B}|^2}\boldsymbol{E}\left[\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})\right]\boldsymbol{E}\left[\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})\right]^\top + \frac{1}{|\mathbb{B}|^2}\mathrm{COV}\left[\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}), \sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})\right] \\
&= \frac{1}{|\mathbb{B}|^2}|\mathbb{B}|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})|\mathbb{B}|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top + \frac{1}{|\mathbb{B}|^2}\sum_{j=1}^{|\mathbb{B}|}\sum_{k=1}^{|\mathbb{B}|}\mathrm{COV}\left[\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}L_k(\boldsymbol{\theta})\right] \\
&= \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top + \frac{1}{|\mathbb{B}|}\boldsymbol{K}_{\boldsymbol{\theta}}.
\end{aligned}
\tag{2.58}
$$

The latter holds since $\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})$ are i.i.d.; consequently, the covariance matrix is non-zero only if $j = k$. In the same way as above it can be shown that $b = \nabla_{\boldsymbol{\theta}}\mathcal{L}\nabla_{\boldsymbol{\theta}}\mathcal{L}^\top + \frac{1}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}}$.
Next, c is:

$$
\begin{aligned}
c &= \frac{2}{|\mathbb{B}||\mathbb{D}|}\boldsymbol{E}\left[\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta})\right]\boldsymbol{E}\left[\sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta})\right]^\top + \frac{2}{|\mathbb{B}||\mathbb{D}|}\mathrm{COV}\left[\sum_{j=1}^{|\mathbb{B}|}\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}), \sum_{i=1}^{|\mathbb{D}|}\nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta})\right] \\
&= \frac{2}{|\mathbb{B}||\mathbb{D}|}|\mathbb{B}|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})|\mathbb{D}|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top + \frac{2}{|\mathbb{B}||\mathbb{D}|}\sum_{j=1}^{|\mathbb{B}|}\sum_{i=1}^{|\mathbb{D}|}\mathrm{COV}\left[\nabla_{\boldsymbol{\theta}}L_j(\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}L_i(\boldsymbol{\theta})\right] \\
&= 2\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top + \frac{2}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}}.
\end{aligned}
\tag{2.59}
$$

Finally we obtain:

$$\begin{aligned}
\boldsymbol{E}[\boldsymbol{\alpha}\boldsymbol{\alpha}^\top] &\overset{2.57}{=} a+b-c \\
&= \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top + \frac{1}{|\mathbb{B}|}\boldsymbol{K}_{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top \\
&+ \frac{1}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}} - 2\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})^\top - \frac{2}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}} \\
&= \frac{1}{|\mathbb{B}|}\boldsymbol{K}_{\boldsymbol{\theta}} + \frac{1}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}} - \frac{2}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}} \\
&= \frac{1}{|\mathbb{B}|}\boldsymbol{K}_{\boldsymbol{\theta}} - \frac{1}{|\mathbb{D}|}\boldsymbol{K}_{\boldsymbol{\theta}} \\
&= (\frac{|\mathbb{D}|}{|\mathbb{B}|} - 1)\frac{\boldsymbol{K}_{\boldsymbol{\theta}}}{|\mathbb{D}|}.
\end{aligned} \tag{2.60}$$

After these side steps, we proceed to derive expression 2.52 as a function of λ, $|\mathbb{B}|$ and $|\mathbb{D}|$. Following (Li *et al.*, 2017; Gardiner, 1985), we reinterpret Equation 2.53 as the discrete update of a stochastic differential equation. In detail, the stochastic differential equation describing the dynamic changes of the parameter is modeled as follows:

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} + \boldsymbol{\eta}(t), \tag{2.61}$$

where t is a continuous variable, $\boldsymbol{\eta}(t)$ models the mini-batch induced gradient-noise with $\boldsymbol{E}[\boldsymbol{\eta}(t)] = 0$ and

$$\boldsymbol{E}[\boldsymbol{\eta}(t)\boldsymbol{\eta}(t')^\top] = \boldsymbol{COV}[\boldsymbol{\eta}(t), \boldsymbol{\eta}(t')] = \nu\frac{\boldsymbol{K}_{\boldsymbol{\theta}}}{|\mathbb{D}|}\delta(t-t'), \tag{2.62}$$

where δ is the Dirac delta function, implying that $\boldsymbol{E}[\boldsymbol{\eta}(t)\boldsymbol{\eta}(t')^\top]$ is non-zero only if $t = t'$, which in turn implies that the noise $\boldsymbol{\eta}$ is independent in time. Using this model, we can see that the *noise scale* ν controls the scale of random fluctuations in the dynamics, i.e., the covariance matrix. The division by $|\mathbb{D}|$ is needed for the covariance matrix to have the correct scale since we consider the mean of random variables.

To relate Equation 2.61 to the SGD update step in Equation 2.53, we consider the weight update over a time interval (learning rate) λ:

$$\Delta\boldsymbol{\theta} = \int_0^\lambda \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t}\mathrm{d}t = -\lambda\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} + \int_0^\lambda \boldsymbol{\eta}(t)\mathrm{d}t. \tag{2.63}$$

To obtain an expression for ν, we compare the mini-batch gradient error times the learning rate in the discrete case ($\boldsymbol{\alpha}\lambda$) with the integrated gradient noise $\int_0^\lambda \boldsymbol{\eta}(t)dt$. Specifically, we equate the covariances of the gradient error times the

learning rate $\boldsymbol{E}[\lambda\boldsymbol{\alpha}(\lambda\boldsymbol{\alpha})^\top]$ and the covariance of the integrated gradient noise $\boldsymbol{E}[\int_0^\lambda \boldsymbol{\eta}(t)dt \int_0^\lambda \boldsymbol{\eta}(t')^\top dt']$:
It is simple to derive that

$$\boldsymbol{E}[\lambda\boldsymbol{\alpha}(\lambda\boldsymbol{\alpha})^\top] \overset{2.60}{=} \frac{\lambda^2}{|\mathbb{D}|}\left(\frac{|\mathbb{D}|}{|\mathbb{B}|}-1\right)\boldsymbol{K}_{\boldsymbol{\theta}} \tag{2.64}$$

and

$$\boldsymbol{E}[\int_0^\lambda \boldsymbol{\eta}(t)dt \int_0^\lambda \boldsymbol{\eta}(t')^\top dt'] = \int_0^\lambda \int_0^\lambda \boldsymbol{E}[\boldsymbol{\eta}(t)\boldsymbol{\eta}(t')^\top]dtdt' \overset{2.62}{=} \lambda\nu\frac{\boldsymbol{K}_{\boldsymbol{\theta}}}{|\mathbb{D}|} \tag{2.65}$$

holds. Equating both results in:

$$\frac{\lambda^2}{|\mathbb{D}|}\left(\frac{|\mathbb{D}|}{|\mathbb{B}|}-1\right)\boldsymbol{K}_{\boldsymbol{\theta}} = \lambda\nu\frac{\boldsymbol{K}_{\boldsymbol{\theta}}}{|\mathbb{D}|}. \tag{2.66}$$

After rearranging, the SGD *noise scale* reads:

$$\nu = \lambda(\frac{|\mathbb{D}|}{|\mathbb{B}|}-1) \approx \lambda\frac{|\mathbb{D}|}{|\mathbb{B}|}. \tag{2.67}$$

The latter approximation holds if $|\mathbb{D}| \gg |\mathbb{B}|$, which is typically the case. Thus, ν is inversely proportional to $|\mathbb{B}|$ and proportional to $|\mathbb{D}|$ and λ. This property will be used in Chapter 6

Figure 2.11 shows that Equation 2.67 holds in practice.

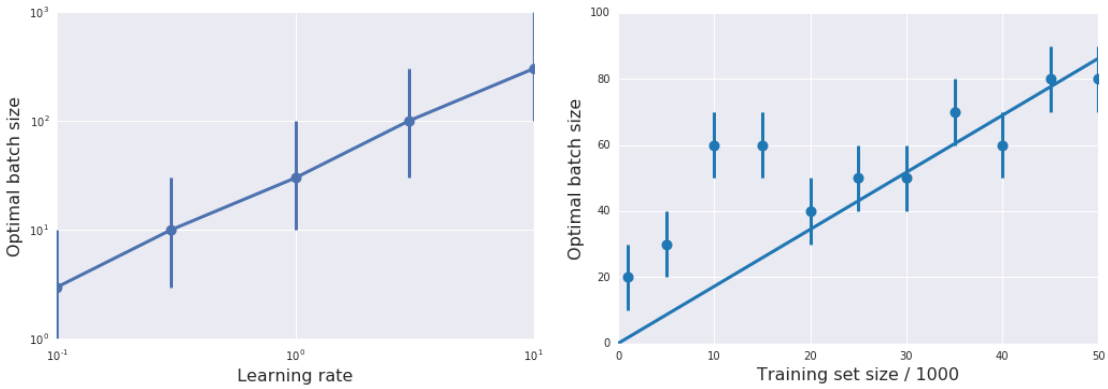


Figure 2.11: Experiments of (Smith and Le, 2018) show that Equation 2.67 holds in practice. A shallow neural network with 800 hidden units and ReLU activation function is trained on MNIST using SGD with momentum. (Smith *et al.*, 2018) shows similar results when training ResNets on CIFAR-10. Figure from (Smith and Le, 2018).

Some further works consider the relation of batch size and learning rate: (McCandlish *et al.*, 2018) introduces another empirically-based scale that predicts the largest beneficial batch size over datasets and models. (De *et al.*, 2016) adaptively increases the batch size over update steps to assure that the negative gradient is a descent direction. (Balles *et al.*, 2017) introduces an optimization algorithm that jointly adapts the batch size and the learning rate. In detail, the algorithm builds on the assumption that as the batch size increases, the gradient variance decreases proportionally to an exponential running average of former mini-batch losses. (Jastrzebski *et al.*, 2017) shows that the ratio of learning rate to batch size is an indicator for the width of minima and thus of the generalization ability. The larger the ratio, the wider the found minimum gets. Furthermore, they show that learning rate schedules can be replaced by batch size schedules, which will be of interest in Chapter 6.

2.11 Grad Student Descent

In this section, based on (Gencoglu *et al.*, 2019), we consider optimization in the deep learning field from a more general perspective. From this perspective, the most successful and controversial optimization algorithm in the deep learning field in recent years is **Grad(uate) Student Descent** (Gencoglu *et al.*, 2019). As discussed in Section 2.1 and 2.2, optimization is not only about minimizing the empirical loss, but also about selecting a good hypothesis space. In practice, numerous design decisions have to be made for the hypothesis space: the model design (including the number of layers, the number of neurons, the activation function, ...), hyperparameters for weight initialization, data augmentation, data normalization, et cetera. The problem is that there are almost no direct relationships between the hyperparameters and the model performance known, not even between the hyperparameters themselves. Consequently, those hyperparameters are manually engineered for each specific application. This contrasts the traditional hypothesis-driven scientific approach: Instead of forming hypotheses based on theory or detailed empirical studies, grad student descent is applied:

Improvement on a specific problem is achieved by assigning it to multiple graduate students, who then try out what works and what does not. Each student follows an iterative approach, starting with a baseline architecture with state-of-the-art results and then applying modifications by trial-and-error. These modifications are usually not based on hypotheses derived from theory or solid empirical evidence. Once some improvements are found, further research is done in this direction until a local optimum is reached and the results are published.

Usually, no profound empirical or theoretical explanation is given as to why the found approach works and excels. Often hypotheses are made after the results are

known, masquerading posterior hypotheses as prior hypotheses, which arguably resembles improper scientific practice. In short, this practice continuously leads to better and better results but not to comprehensive explanations of why things work. Consequently, we generally do not know why and when SGD performs better than other optimizers or why ResNets (He *et al.*, 2016) and DenseNets (Huang *et al.*, 2017) perform better than VGGNets (Simonyan and Zisserman, 2015). In the following chapters, we do better by deriving optimization methods based on priorly measured and comprehensive empirical observations. This provides a more comprehensive understanding of when and why these methods work.

Chapter 3

Experimental Platform:

In this chapter, we introduce our experimental platform so that our methodology can be comprehended and our results can be reproduced.

3.1 The TCML-Cluster

A large number of processing resources was needed for the experiments performed in this work. Therefore it was of great advantage that the project accompanying this dissertation was to set up and maintain the Training Center Machine Learning-Cluster (TCML-Cluster, also see (University of Tuebingen, 2021)) within the BMBF project "Training Center Machine Learning, Tübingen" with grant number 01|S17054. The setup and maintenance was mainly done by the author of this work with support by Uli Ulmer, Klaus Bayreuther and Martin Meßmer. This often tedious to maintain, but simple to use and very flexible computing environment paved the way for at least 30 scientific works, including (Mutschler and Zell, 2020a; Mutschler *et al.*, 2021; Mutschler and Zell, 2021; Laube and Zell, 2021; Laube, 2021; Laube and Zell, 2019a,b; Gao *et al.*, 2021; Tebbe *et al.*, 2021, 2020; ul Moqeet Riaz *et al.*, 2022, 2020; Lange *et al.*, 2019, 2020; Bolz *et al.*, 2019; Mutschler and Zell, 2020b; Sanzenbacher *et al.*, 2020; Butz *et al.*, 2019b,a; Otte *et al.*, 2019a,b; Hobbhahn *et al.*, 2020; Varga *et al.*, 2022a; Laube *et al.*, 2022; Varga *et al.*, 2021; Varga and Zell, 2021; Rahim *et al.*, 2021; Kiefer *et al.*, 2021; Varga *et al.*, 2022b) and (Shamsafar *et al.*, 2022). In addition, it has been used by approximately 400 students for lecture exercises and a considerable amount of bachelor's and master's theses, programming projects, and practical courses. Consequently, maintaining the TCML-Cluster has made a small but important contribution to a large body of scientific work. Also, the TCML-Cluster was essential for this work to compute often highly resource demanding calculations such as measuring empirical data of loss landscapes or to compare several optimization approaches on benchmark problems. The cluster is used for all experiments in this work presented in Chapter 4, 5 and 6.

The following provides a detailed overview of the hardware and software used:

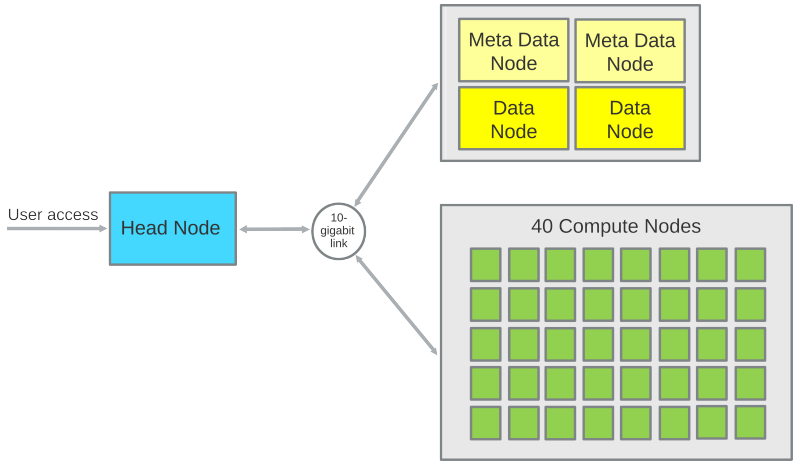


Figure 3.1: Physical overview of the TCML-Cluster consisting of 40 compute nodes used for calculations, four storage nodes holding the users' data and, the head node that controls the functionalities of the cluster and is the access point for users. This cluster was used for all experiments performed in this work.

3.1.1 Hardware

An overview of the cluster's physical structure is shown in Figures 3.1 and 3.2. To separate control access, computations, and data storage, the cluster is divided into one head node, 40 compute nodes and four storage nodes. All computations for experiments are performed on the compute nodes(University of Tuebingen, 2021).

Each compute node originally had the following hardware specifications: 2 TB of SSD space; 256 GB of memory; an Intel XEON CPU E5-2650 v4 processor; and four Nvidia GeForce GTX 1080 graphics cards; In 2021, the GPUs of two nodes got replaced with Nvidia RTX A4000 GPUs; however, Nvidia GeForce GTX 1080 graphics cards were used for all the experiments of this work. The two data nodes and meta-data nodes hold all the cluster's user data on 146 TB of storage. They are connected to the compute nodes with a 10-gigabit link. This is a bottleneck in the speed at which data can be transferred to the compute nodes, particularly relevant when training on larger datasets.

The head (master) node controls all the functionalities of the cluster in a centralized manner. It ensures that the jobs (tasks) on the cluster are properly scheduled and monitored. Users are only allowed to access this node and have to submit their jobs there.

3.1.2 Software

The heart of the TCML-Cluster is the workload manager Slurm (Yoo *et al.*, 2003), an open-source, fault-tolerant, and highly scalable cluster management and job

scheduling system for Linux clusters. It assigns users access to the compute nodes for a particular duration of time and amount of hardware resources. The automatic job scheduling system was beneficial for the many grid searches performed in this work.

In order to allow users to use any Linux working environment, the high-performance-computing container visualization software Singularity (Sylabs, 2015) is used. Consequently, a suited Linux operating system with any libraries installed can be used for any use case. Unlike a virtual machine, a singularity container does not require an emulated kernel but runs directly on the host kernel as a process. This makes the execution of an application in a container almost as fast as that of an ordinary application. Singularity containers are fully compatible with the often-used Docker (Docker, Inc., 2013) containers, allowing the use of a huge amount of preconfigured containers provided at `hub.docker.com`. For the experiments in this work, we used the cluster's default containers that provided environments with the latest versions of CUDA (NVIDIA, 2007), cuDNN (NVIDIA, 2015), TensorFlow (Abadi *et al.*, 2016), and PyTorch (Paszke *et al.*, 2019) versions at the time of executing the corresponding experiment.

Figure 3.2: Server racks of the TCML-Cluster.

Chapter 4

Parabolic Approximation Line Search

This chapter introduces *PAL*, our line search approach operating on the mini-batch loss ($\mathcal{L}_{\mathbb{B},t}$). Supported by empirical evidence, it approximates the mini-batch loss along a line ($l_{\mathbb{B},t}$) with a positively curved parabola. It is one of the few optimization methods for deep learning that is derived from empirical evidence about the shape of the loss landscape, and thus provides a more comprehensive understanding than other approaches of why and how it works in practice. The following chapter is based on (Mutschler and Zell, 2020a).

4.1 Motivation and Introduction

Automatic determination of optimal step sizes for each update step of stochastic gradient descent is a major challenge in current optimization research for deep learning (Rolínek and Martius, 2018; Berrada *et al.*, 2020; Mahsereci and Hennig, 2015; Kafka and Wilke, 2019; Paquette and Scheinberg, 2018; Vaswani *et al.*, 2019; De *et al.*, 2016; Baydin *et al.*, 2018; Lancewicki and Kopru, 2020). One default approach to tackle this challenge is to apply line search methods. Several of these have been introduced for deep learning (Mahsereci and Hennig, 2015; Kafka and Wilke, 2019; Paquette and Scheinberg, 2018; Vaswani *et al.*, 2019; De *et al.*, 2016). However, these approaches have not analyzed the shape of loss functions in update step direction in detail, which is important since the optimal step size stands in strong relation to this shape. To shed light on this, we empirically analyzed the shape of the loss function along update step direction for deep learning scenarios often considered in optimization. We further elaborate on the properties found to define a simple and empirically justified optimizer. In relation to Section 2.11 about Grad Student Descent we derive our approach from a clear prior hypothesis.
Our contributions to the community are as follows:
1: Our empirical analysis suggests that the mini-batch loss in the negative gradient direction mostly exhibits locally convex shapes. Furthermore, we empirically show that parabolic approximations are well suited to estimate the minima in these di-

rections (Section 4.3).
2: Exploiting this parabolic observation, we build a simple line search optimizer that constructs its loss function-dependent learning rate schedule. The performance of our optimization method is extensively analyzed, including a comprehensive comparison to other optimization methods (Sections 4.4,4.5).
3: We provide a theoretical convergence analysis that supports our empirical results under strong assumptions (Section 4.4.4).
4: We provide a general investigation of exact line searches on batch losses and their relation to line searches on the exact loss as well as their relation to our line search approach (Section 4.6) and, finally, analyze the relation of our approach to interpolation (Section 4.7).

In this chapter, we consider $\mathcal{L}_{\mathbb{B},t}$ at optimization step t in negative gradient direction as defined in Equation 2.22 but we normalize the direction:

$$l_{\mathbb{B},t} : \mathbb{R} \to \mathbb{R}, s \mapsto \mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t + s\frac{-\boldsymbol{g}_{\mathbb{B},t}}{||\boldsymbol{g}_{\mathbb{B},t}||}), \tag{4.1}$$

where $\boldsymbol{g}_{\mathbb{B},t}$ is $\nabla_{\boldsymbol{\theta}_t}\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)$. We denote l_t as *mini-batch loss along a line* and s as the step along this line. The motivation of our work builds upon the following assumption:

Assumption 2. *(Informal) The position* $\boldsymbol{\theta}_{min} = \boldsymbol{\theta}_t + s_{min}\frac{-\boldsymbol{g}_{\mathbb{B},t}}{||\boldsymbol{g}_{\mathbb{B},t}||}$ *of a minimum of* $l_{\mathbb{B},t}$ *is a good enough estimator for the position of a minimum of the full-batch loss* $\mathcal{L}$ *on the same line to perform a successful optimization process.*

We empirically analyze Assumption 2 further in Section 4.6.

4.2 Related Work

Our optimization approach is based on well-known methods, such as line search, the non-linear conjugate gradient method (see Section 2.7), and quadratic approximation, which can be found in Numerical Optimization (Nocedal and Wright, 2006). In addition, (Nocedal and Wright, 2006, §3.5) describes a similar line search routine for the deterministic setting. The concept of parabolic approximations is also exploited by the well-known line search of (Moré and Thuente, 1994), which is also designed for the deterministic setting. Our work contrasts common optimization approaches in deep learning by directly exploiting the observation of $l_{\mathbb{B},t}$ being locally parabolic (see Section 4.3). Similarly, *SGD-HD* (Baydin *et al.*, 2018) (see Section 2.8) performs update steps towards a minimum of $l_{\mathbb{B},t}$, by performing gradient descent on the learning rate. The *L4* adaptation scheme (Rolínek and Martius, 2018) as well as *ALIG* (Berrada *et al.*, 2020) (see Section 2.8) estimate

step sizes by approximating the loss function linearly in negative gradient direction, whereas our approach approximates the loss function parabolically in negative gradient direction. (Chae and Wilke, 2019) explores a similar direction as this work by analyzing possible line search approximations for DNN loss landscapes but does not exploit these for optimization.

Stochastic Line Search (*SLS*) (Vaswani *et al.*, 2019) (in detail introduced in Section 2.6.3), searches -like our line search- for a position on $l_{\mathbb{B},t}$. However, it does not perform an exact line search, (i.e., determining a minimum almost exactly along the line), but checks that the first Wolfe condition is fulfilled. Whereas our approach builds upon empirical findings, their approach relies on the rather theoretical interpolation assumption (Assumption 1 Page 16). *SLS* exhibits competitive performance against multiple optimizers on several DNN tasks. (Paquette and Scheinberg, 2018) introduces a related idea but does not provide empirical results for DNNs. (De *et al.*, 2016) also uses a backtracking Armijo line search, but to regulate the optimal batch size. The methodically appealing but complex *Probabilistic Line Search* (*PLS*) (in detail introduced in Section 2.6.5) (Mahsereci and Hennig, 2015) approximates $\mathcal{L}$ along lines with a GP posterior based on realizations of $\mathcal{L}_{\mathbb{B}_i}$ for different batches $\mathbb{B}_i$. The mean of this posterior is a cubic spline. A suitable update step size is found by exploiting a probabilistic formulation of the Wolf conditions. In contrast to *PLS*, *PAL* does not approximate the full-batch loss at all and relies on a non-probabilistic parabolic approximations of $\mathcal{L}_{\mathbb{B},t}$. *Gradient Only Line Search* (*GOLSI*) (Kafka and Wilke, 2019)(in detail introduced in Section 2.6.4) searches for a minimum on lines by looking for a sign change of the first directional mini-batch derivative in search direction. For each measurement, another $\mathbb{B}_i$ is chosen. Our approach, in contrast, just sticks to one $\mathbb{B}_i$ for a line search and uses empirical evidences to approximate $\mathcal{L}_{\mathbb{B},t}$.

From the perspective of assumptions about the shape of the loss landscape, second order methods such as *oLBFGS* (Schraudolph *et al.*, 2007), *KFRA* (Botev *et al.*, 2017), *L-SR1* (Ramamurthy and Duffy, 2017), *QUICKPROP* (Fahlman *et al.*, 1988), *S-LSR1* (Berahas *et al.*, 2021), and *KFAC* (Martens and Grosse, 2015) generally assume that the loss function can be approximated locally by a parabola of the same dimension as the loss function. However, those methods are computationally expensive. Thus, our optimizer follows the cheaper approach by using a one dimensional parabolic approximation in negative gradient direction. Adaptive methods such as *SGD* with momentum (Robbins and Monro, 1951) (see Section 2.8), *ADAM* (Kingma and Ba, 2015) (Section 2.8), *ADAGRAD* (Duchi *et al.*, 2011), *ADABOUND* (Luo *et al.*, 2019), *AMSGRAD* (Reddi *et al.*, 2018) or *RMSProp* (Tieleman and Hinton, 2012) (Section 2.8) focus more on finding directions of less gradient noise than on shape assumptions. Our approach could be easily applied upon such directions, too.

4.3 Empirical Analysis of the Shape of mini-batch Losses along Lines

In this section, we analyze mini-batch losses along lines (see Eq. 4.1) during the training of multiple architectures and show that they locally exhibit mostly convex shapes, which are well suited for parabolic approximations. We focus on CIFAR-10, as it is extensively analyzed in optimization research for deep learning. However, we observed similar results on MNIST, CIFAR-100, and ImageNet, considering random samples. We analyzed mini-batch losses along the lines in the first 10,000 SGD update step directions for four commonly used architectures (ResNet-32 (He *et al.*, 2016), DenseNet-40 (Huang *et al.*, 2017), EfficientNet (Tan and Le, 2019),

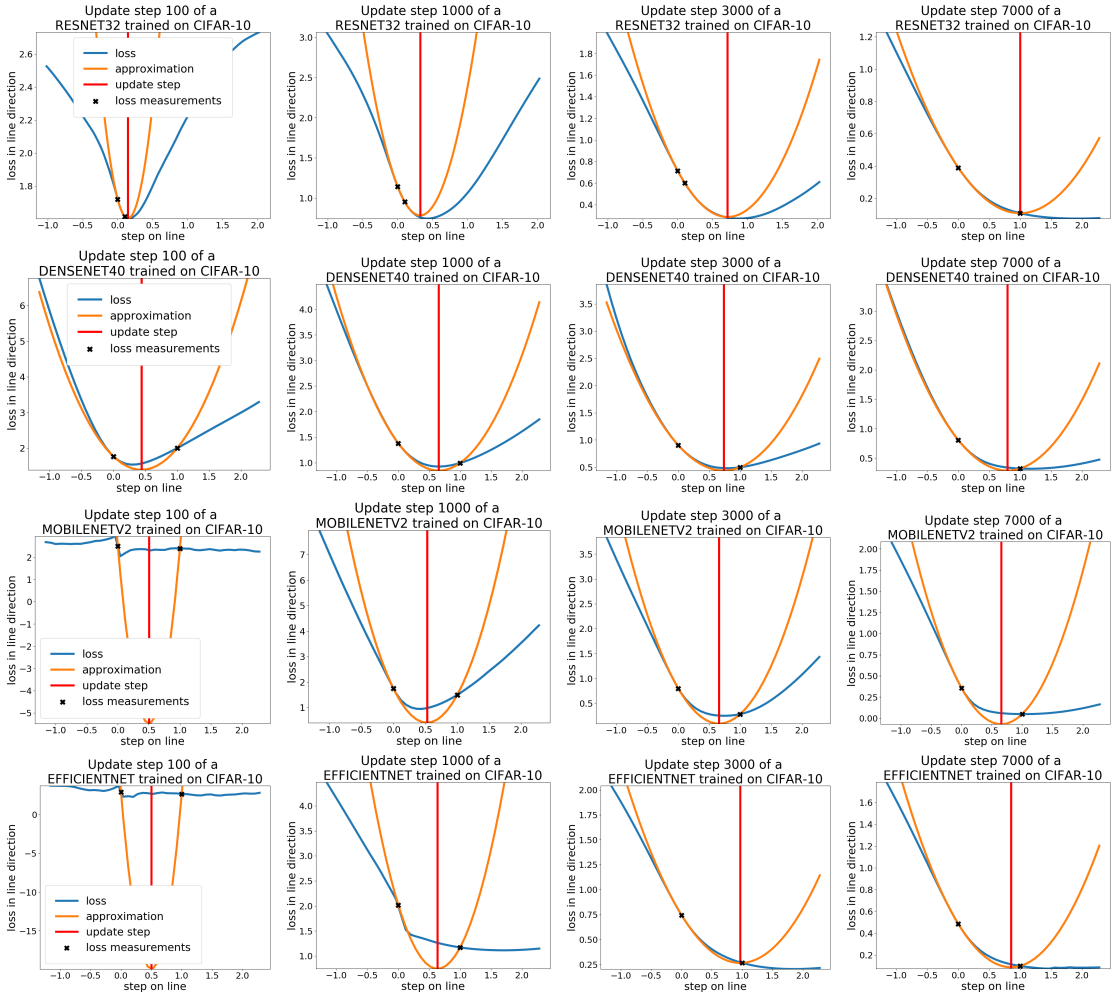


Figure 4.1: Representative mini-batch losses along lines in negative normalized gradient direction (blue), parabolic approximations (orange) and the position of the approximated minima (red). Further plots are provided in Appendix A.1. **Row 1**: ResNet32, **Row 2**: DenseNet40. **Row 3**: MobilenetV2. **Row 4**: EfficientNet.

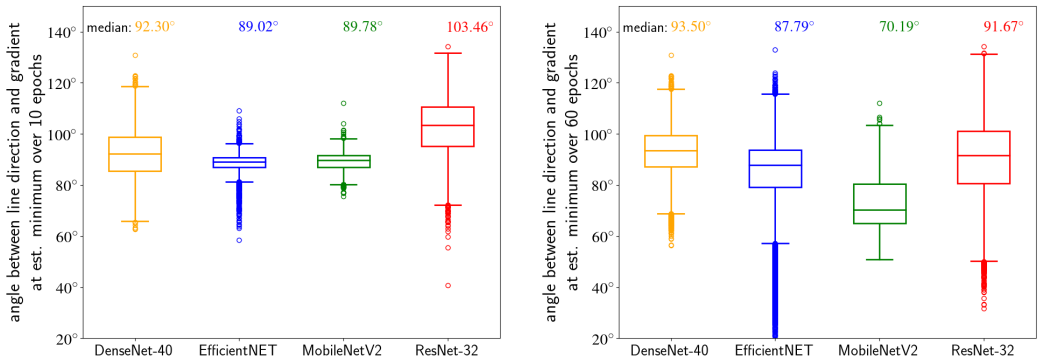


Figure 4.2: Angles between the line direction and the gradient at the estimated minimum measured on the same mini-batch. If the angle is 90°, the estimated minimum is a real local minimum. We know from additional line plots that the found extrema or saddle points are minima. **Left**: measurement over the first 10 epochs. **Right**: measurement over the first 60 epochs. Update step adaptation (see Section 4.4.3) is applied.

MobileNetV2 (Sandler *et al.*, 2018)). Along each line, we sampled 50 losses and performed a parabolic approximation (see Section 4.4). A representative selection of our results on a ResNet32 is shown in Figure 4.1. Further results are given in Appendix A.1. In accordance with (Xing *et al.*, 2018), we conclude that the mini-batch losses along the analyzed lines tend to be locally convex; except for the approximately first 200 lines when training on EfficientNet and MobileNet. In addition, one-dimensional parabolic approximations of the form $f(s) = as^2 + bs + c$ with $a \neq 0$ are well suited to estimate the position of a minimum along such directions. For EfficientNet the latter does not hold; but as we will see in Section 4.5 applying parabolic approximations still leads to a good optimization process.

To substantiate the later observation, we analyzed the angle between the line direction and the gradient at the -by a parabola- estimated minimum during training. A position is a local extremum or saddle point of the loss along a line if and only if the angle between the line direction and the gradient at the position is 90°, if measured on the same mini-batch.[1] Measuring step sizes and update step adaptations factors (see Sections 4.4.1 and4.4.3) were chosen to fit the mini-batch loss along the line decently. As shown in Figures 4.2 and 4.3, the parabolic observation holds well for several architectures trained on MNIST, CIFAR-10, CIFAR-100 and ImageNet. The observation fits best for MNIST and gets worse for more complex tasks such as ImageNet. We can ensure that the extrema found are minima since

[1]This holds because if the directional derivative of the measured gradient in line direction is 0, the current position is an extremum or saddle point along the line and the angle is 90°. If the position is not an extremum or saddle point, the directional derivative is not 0 (Nocedal and Wright, 2006).

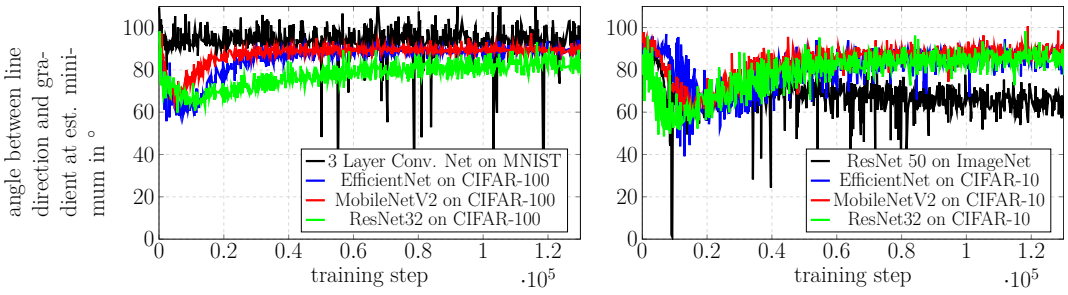


Figure 4.3: Angles between the line direction and the gradient at the estimated minimum measured on the same batch plotted over a whole training process on several networks and datasets. This figure clarifies that the parabolic observation holds also on further datasets and during the training process. It fits best for MNIST and becomes worse for ImageNet. Measuring step sizes and update step adaptations factors (see Sections 4.4.1,4.4.3) were used to fit the loss along the line decently.

we additionally plotted the mini-batch loss along the line for each update step. In addition, we analyzed the mini-batch loss along lines in conjugate-like directions and random directions. Mini-batch losses along lines in conjugate like directions also tend to have convex shapes (Figure 4.6). However, mini-batch losses along lines in random directions rarely exhibit convex shapes.

4.4 The Line Search Algorithm

We introduce **P**arabolic **A**pproximation **L**ine Search (*PAL*), which exploits the observation that parabolic approximations are suited to estimate the minimizer of the mini-batch loss along lines. This simple approach combines well-known methods from basic optimization such as parabolic approximation and line search (Nocedal and Wright, 2006) to perform an efficient line search. We note that the general idea of this method can be applied to any optimizer that provides an update step direction, such as ADAM, RMSP, or AdaGrad. *PAL*'s pseudo code is given in Algorithm 8.

4.4.1 Parameter Update Rule

An intuitive explanation of *PAL*'s parameter update rule based on a parabolic approximation is given in Figure 4.4. Since $l_{\mathbb{B},t}$ (see Eq. 4.1) is assumed to exhibit a convex and almost parabolic shape, we approximate it with $\hat{l}_t(s) = as^2 + bs + c$ with $a \in \mathbb{R}^+$ and $b, c \in \mathbb{R}$. Consequently, we need three measurements of $l_{\mathbb{B},t}$ to define a, b and c. Those are given by the current loss $l_{\mathbb{B},t}(0)$, the derivative in gradient direction $l'_{\mathbb{B},t}(0) = -||\boldsymbol{g}_{\mathbb{B},t}||$ (see Eq. 4.4) and an additional loss $l_{\mathbb{B},t}(\mu)$

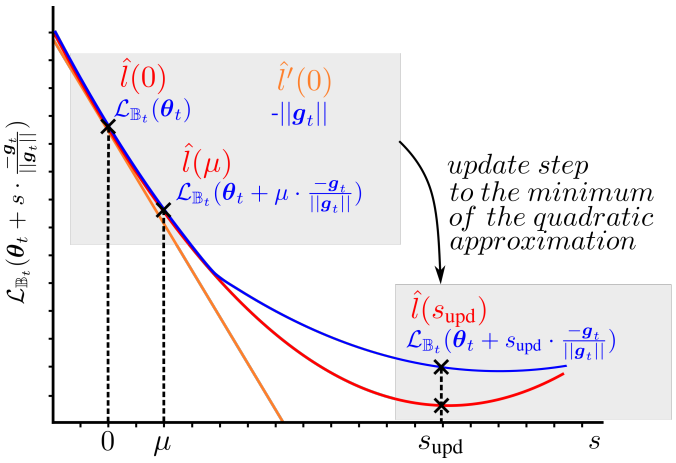


Figure 4.4: Basic idea of *PAL*'s parameter update rule. The blue curve is the mini-batch loss along the negative gradient starting at $\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)$. It is defined as $l_{\mathbb{B},t}(s) = \mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t + s\frac{-\boldsymbol{g}_t}{||\boldsymbol{g}_{\mathbb{B},t}||})$ where $\boldsymbol{g}_t$ is $\nabla_{\boldsymbol{\theta}_t}\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)$. The red curve is its parabolic approximation $\hat{l}(s)$. $l(0)$, $l(\mu)$ and $\boldsymbol{g}_{\mathbb{B},t}$ (orange) are the three parameters needed to determine the update step s_{upd} to the minimum of the parabolic approximation.

with measuring distance $\mu \in \mathbb{R}^+$. It is simple to show that $a = \frac{l_{\mathbb{B},t}(\mu) - l_{\mathbb{B},t}(0) - l'_{\mathbb{B},t}(0)\mu}{\mu^2}$, $b = l'_{\mathbb{B},t}(0)$, and $c = l_{\mathbb{B},t}(0)$. The update step s_{upd} to the minimum of the parabolic approximation $\hat{l}_t(s)$ is thus given by:

$$s_{\text{upd}_t} = -\frac{\hat{l}'_t(0)}{\hat{l}''_t(0)} = -\frac{b}{2a} = \frac{-l'_{\mathbb{B},t}(0)}{2\frac{l_{\mathbb{B},t}(\mu) - l_{\mathbb{B},t}(0) - l'_{\mathbb{B},t}(0)\mu}{\mu^2}}. \tag{4.2}$$

Note, that $\hat{l}''_t(0)$ is the second derivative of the approximated parabola and is only identical to the exact directional derivative $\frac{-\boldsymbol{g}_{\mathbb{B},t}}{||\boldsymbol{g}_{\mathbb{B},t}||} H(\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)) \frac{-\boldsymbol{g}^T_{\mathbb{B},t}}{||\boldsymbol{g}_{\mathbb{B},t}||}$ if the parabolic approximation fits. The normalization of the gradient to unit length (Equation 4.1) was chosen to have the measuring distance μ independent of the gradient size and of weight scaling. Note that two network inferences are required to determine $l_{\mathbb{B},t}(0)$ and $l_{\mathbb{B},t}(\mu)$. Consequently, *PAL* needs two forward passes and one backward pass through a model. Further on, $\mathcal{L}_{\mathbb{B},t}$ may include random components, but to ensure continuity during one line search, drawn random numbers have to be reused for each value determination of $\mathcal{L}_{\mathbb{B},t}$ at t (e.g., for Dropout (Srivastava *et al.*, 2014)). The memory required by *PAL* is similar to *SGD* with momentum, since only the last update direction has to be saved.

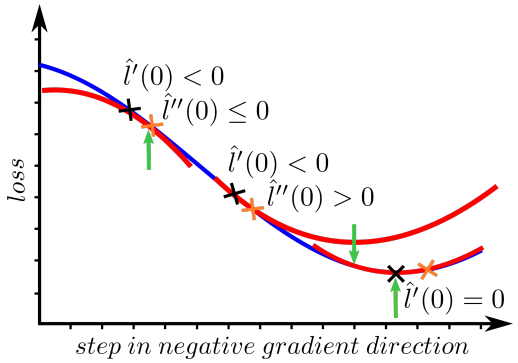


Figure 4.5: *PAL*'s case discrimination as described in Section 4.4.2. Note that $b = \hat{l}'(0) = l'_{\mathbb{B},t}(0)$ and $a = 0.5\hat{l}''(0) \approx 0.5\ l''_{\mathbb{B},t}(0)$. The blue curve is the real mini-batch loss; red are its approximations; black crosses are the first measuring point; orange crosses the second measuring point. Green arrows show the corresponding resulting update step positions. From left to right the exemplary cases correspond to case two, one and three of Section 4.4.2.

4.4.2 Case Discrimination of Parabolic Approximations

Since not all parabolic approximations are suitable for parameter update steps, the following cases are considered separately. Note that $b = \hat{l}'(0) = l'_{\mathbb{B},t}(0)$ and $a = 0.5\hat{l}''(0) \approx 0.5 l''_{\mathbb{B},t}(0)$. They are figuratively explained in Figure 4.5. **1:** $a > 0$ and $b < 0$: the parabolic approximation has a minimum in line direction, thus, the parameter update is done as described in Section 4.4.1. **2:** $a \leq 0$ and $b < 0$: the parabolic approximation has a maximum in negative line direction or is a line with a negative slope. In those cases, a parabolic approximation is inappropriate. s_{upd} is set to μ, since the second measured point has a lower loss than the first. **3:** Since $b = -||\boldsymbol{g}_{\mathbb{B},t}||$ cannot be greater than 0, the only case left is an extremum at the current position ($l'(0) = 0$). In this case, no weight update is performed. However, the mini-batch loss function changes with the next batch, which likely does not have an extremum at exactly the same point. In accordance with Section 4.3, cases two and three appeared very rarely in our experiments.

4.4.3 Additions

Here multiple additions to fine-tune *PAL*'s performance and handle degenerate cases are introduced. Our hyperparameter sensitivity analysis (Appendix A.2.2) suggests that the influence of the introduced hyperparameters on the optimizer's performance is low. Thus, they only need to be adapted to fine-tune the results.

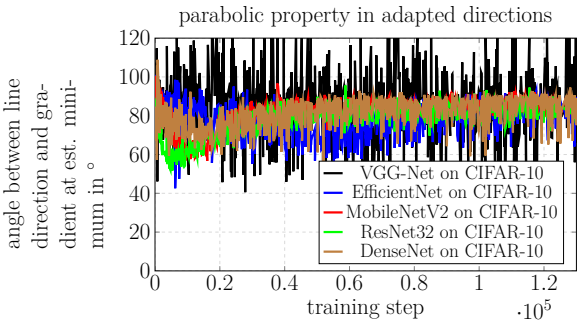


Figure 4.6: Angles between the line direction and the gradient at the estimated minimum measured on the same batch are plotted over a whole training process on several networks on CIFAR-10. This figure clarifies, that parabolic observation is also valid if a **direction adaptation factor of 0.4** is applied. Measuring step sizes and update step adaptations factors (see Sections 4.4.1,4.4.3) were set to fit loss along the line decently.

Direction adaptation: Instead of following the direction of the negative gradient we follow an adapted conjugate-like direction $\boldsymbol{d}_t$:

$$\boldsymbol{d}_t = -\nabla_{\boldsymbol{\theta}_t}\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t) + \beta \boldsymbol{d}_{t-1} \quad \boldsymbol{d}_0 = -\nabla_{\boldsymbol{\theta}_0}\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_0), \tag{4.3}$$

with $\beta \in [0,1]$. Since now an adapted direction is used, $l'_{\mathbb{B},t}(0)$ changes to:

$$l'_{\mathbb{B},t}(0) = \nabla_{\boldsymbol{\theta}_t}\mathcal{L}_{\mathbb{B},t}(\boldsymbol{\theta}_t)\frac{\boldsymbol{d}_t}{||\boldsymbol{d}_t||}. \tag{4.4}$$

This approach aims to find a more optimal search direction than the negative gradient. We implemented and tested the formulas of Fletcher-Reeves (Fletcher and Reeves, 1964), Polak-Ribière (Ribière and Polak, 1969), Hestenes-Stiefel (Hestenes and Stiefel, 1952), and Dai-Yuan (Dai and Yuan, 1999) to determine conjugate directions under the assumption that the loss function is a quadratic. However, choosing a constant β of value 0.2 or 0.4 performs equally well. The influence of β and dynamic update steps on *PAL*'s performance is discussed in Section 4.5.3. In the analyzed scenario, β can both increase and decrease the performance, whereas dynamic update steps mostly increase the performance. A combination of both is needed to achieve optimal results. Figure 4.6 suggests that the parabolic observation also holds in these adapted conjugate-like directions.

Update step adaptation: Our preliminary experiments revealed a systematic error caused by constantly approximating with slightly too narrow parabolas. Therefore, s_{upd} is multiplied by a parameter $\alpha \geq 1$ (compare to Equation 4.2). This is useful to estimate the minimum's position along a line more exactly but has minor effects on training performance.

Maximum step size: To hinder the algorithm from failing due to inaccurate parabolic approximations, we use a maximum step size $s_{\max}$. The new update step is given by $\min(s_{\text{upd}}, s_{\max})$. However, most of our experiments with $s_{\max} = 10^{0.5} \approx 3.16$ never reached this step size and still performed well.

Algorithm 8 *PAL*, our proposed line search algorithm for DNNs. See Section 4.4 for details. This algorithm provides the full description of *PAL* including the additions described in Section 4.4.3. PyTorch and TensorFlow implementations are found at `https://github.com/cogsys-tuebingen/PAL`.

	symbol	explanation	default value
	$\mathcal{L}_{\mathbb{B},t}$	mini-batch loss function	–
	$\boldsymbol{\theta}_0$	initial parameter vector	–
Input:	μ	measuring step size	$0.1, 0.01$
	α	update step adaptation	$1.0, 1.6$
	β	direction adaptation factor	$0.4, 0$
	$s_{\max}$	maximum step size	3.16

$t \leftarrow 0$
$\boldsymbol{d}_t \leftarrow \mathbf{0}$ ▷ initialize direction vector
while $\boldsymbol{\theta}_t$ not converged **do**
  $\mathbb{B}_t \leftarrow$ sampleBatch()
  $l_0 \leftarrow \mathcal{L}_{\mathbb{B}t}(\boldsymbol{\theta}_t)$ ▷ get first loss at line origin
  $\boldsymbol{d}_t \leftarrow -\nabla_{\boldsymbol{\theta}_t} \mathcal{L}_{\mathbb{B}t}(\boldsymbol{\theta}_t) + \beta \boldsymbol{d}_{t-1}$ ▷ define new line direction
  $l_\mu \leftarrow \mathcal{L}_{\mathbb{B}t}(\boldsymbol{\theta}_t + \mu \frac{\boldsymbol{d}_t}{||\boldsymbol{d}_t||})$ ▷ get second loss value along the line
  $b \leftarrow \nabla_{\boldsymbol{\theta}_t} \mathcal{L}_{\mathbb{B}t}(\boldsymbol{\theta}_t) \frac{\boldsymbol{d}_t}{||\boldsymbol{d}_t||}$ ▷ get $b =$ directional derivative at line origin
  $a \leftarrow \frac{l_\mu - l_0 - b\mu}{\mu^2}$ ▷ get $a =$ approx. second derivative at line origin / 2
  if $a > 0$ **and** $b < 0$ **then**
    $s_{\text{upd}} \leftarrow -\alpha \frac{b}{2a}$ ▷ get Newton update step to minimum of the app. parabola
  else if $a \leq 0$ **and** $b < 0$ **then** ▷ saveguard against inoperable parabolas (see Section 4.4.2)
    $s_{\text{upd}} \leftarrow \mu$ ▷ step to μ decreases the loss
  else
    $s_{\text{upd}} \leftarrow 0$ ▷ step in line direction would incraese the loss
  end if
  if $s_{\text{upd}} > s_{\max}$ **then** ▷ saveguard against excessive large update steps
    $s_{\text{upd}} \leftarrow s_{\max}$
  end if
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + s_{\text{upd}} \frac{\boldsymbol{d}_t}{||\boldsymbol{d}_t||}$ ▷ perform parameter update
  $t \leftarrow t + 1$
end while
return $\boldsymbol{\theta}_t$

4.4.4 Theoretical Considerations

Usually, convergence in deep learning is shown for convex stochastic functions with an L-Lipschitz continuous gradient. However, since our approach originates from empirical results, it is not given that a profound theoretical analysis is possible. In order to show any convergence guarantees for parabolic approximations, we have to fall back to uncommonly strong assumptions, which lead to quadratic models. If those assumptions are valid at all, they are likely more valid locally than globally.

We assume that each slice of the loss function is a one-dimensional parabolic function:

Assumption 3. *Let $n \in \mathbb{N}$ be the number of parameters and let $\boldsymbol{p}, \boldsymbol{d} \in \mathbb{R}^n$ be vectors. Then for all $\boldsymbol{p}, \boldsymbol{d}$ there exists $a, b, c \in \mathbb{R}$ with $a > 0$, such that $\mathcal{L}_{\mathbb{B}}(\boldsymbol{p} + \boldsymbol{d}s) = as^2 + bs + c$ for all $s \in \mathbb{R}$.*

This strong assumption is a simplified adaptation to our empirical results that lines in negative gradient direction behave locally almost parabolic (see Section 4.3). For the following derivations we assume a basic *PAL* without the additions introduced in Section 4.4.3. Proofs are provided in Appendix A.1.1. At first we show that $\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta})$ is an n-dimensional parabolic function:

Lemma 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a k-times continuously differentiable function. Furthermore, assume there exists $a, b, c \in \mathbb{R}$ with $a > 0$, such that $f(\boldsymbol{p} + \boldsymbol{d}s) = as^2 + bs + c$ for all $s \in \mathbb{R}$. Then there exist $c \in \mathbb{R}, \boldsymbol{r} \in \mathbb{R}^n$ and a positive definite Matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ such that $f(\boldsymbol{x}) = c + \boldsymbol{r}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^n$.*

Now we show that *PAL* converges on $\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta})$:

Proposition 1. *PAL converges on $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto c + \boldsymbol{r}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ with $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ hermitian and positive definite.*

In this scenario, *PAL* is identical to the method of steepest descent for which the convergence, including convergence rates on quadratics, is already proven in (Luenberger *et al.*, 1984, Page 235). Nevertheless, we have added our own agnostic proof which better adapts to this scenario and was made independent of the existing proof.

For a noisy scenario where each batch defines a quadratic, *PAL* has no convergence guarantee. Given two shifted one-dimensional parabolas, $ax^2 + bx + c$ and $a(x + d)^2 + b(x + d) + c$, which are presented to *PAL* alternately, *PAL* will always perform an update step to the minimum position of one of these but never to the minimum position of the average of both. By slightly changing the training procedure and assuming that each $\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta})$ has the same $\boldsymbol{Q}$ this can be fixed:

Proposition 2. *If $\mathcal{L}(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$ $\boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} c_i + \boldsymbol{r}_i^T\boldsymbol{\theta} + \boldsymbol{\theta}^T\boldsymbol{Q}_i\boldsymbol{\theta}$ and $c_i + \boldsymbol{r}_i^T\boldsymbol{\theta} + \boldsymbol{\theta}^T\boldsymbol{Q}_i\boldsymbol{\theta} = \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta})$ with m being the number of batches $\mathbb{B}_i$. (Each batch defines*

a parabola. The empirical loss $\mathcal{L}(\boldsymbol{\theta})$ *is the mean of these parabolas). And for all* $i, j \in \mathbb{N}$ *it holds that* $\boldsymbol{Q}_i = \boldsymbol{Q}_j$ *and that* $\boldsymbol{Q_i}$ *is positive definite. Then* $\arg\min_{\theta} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \arg\min_{\theta} \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta})$ *holds.*

This implies that under Assumption 3 and a fixed $\boldsymbol{Q}$ the position of the minimum of the empirical loss is given by the average of the positions of the mini-batch losses' minima. The minimum position of full-batch loss is found by *PAL*, by slightly adapting *PAL* to search on one batch until it finds the minimum's position and then averages over the minimum of each batch. As a result, *PAL* converges in this noisy scenario. However, we have to emphasize at this point that our assumptions about $\boldsymbol{p}$ and $\boldsymbol{Q}$ are likely not valid for general deep learning scenarios. Nevertheless, if it is locally valid, this direction might be a further explanation, in addition to those of (He *et al.*, 2019; Fort and Jastrzebski, 2019), why averaging the weights of each update step in the last epoch (stochastic weight averaging) (Izmailov *et al.*, 2018) performs well.

4.5 Evaluation

4.5.1 Experimental Design

We performed a comprehensive evaluation to analyze the performance of *PAL* on a variety of deep learning optimization tasks. Therefore, we tested *PAL* on commonly used architectures trained on CIFAR-10, CIFAR-100 (Krizhevsky *et al.*, 2009) and ImageNet (Deng *et al.*, 2009).

For CIFAR-10 and CIFAR-100, we evaluated on DenseNet40 (Huang *et al.*, 2017), EfficientNetB0 (Tan and Le, 2019), ResNet32 (He *et al.*, 2016) and MobileNetV2 (Sandler *et al.*, 2018). On ImageNet we evaluated on DenseNet121 and ResNet50. In addition, we considered an RNN trained on the Tolstoi war and peace text prediction task.

We compare *PAL* to *SLS* (Vaswani *et al.*, 2019), whose Armijo variant is state-of-the-art in the line search field for DNNs at the time of writing. In addition, we compare against the following well studied and widely used first order optimizers: *SGD* with momentum (Robbins and Monro, 1951), *ADAM* (Kingma and Ba, 2015), and *RMSProp* (Tieleman and Hinton, 2012) as well as against *SGDHD* (Baydin *et al.*, 2018) and *ALIG* (Berrada *et al.*, 2020), which automatically estimate learning rates in negative gradient direction and, finally, against the coin betting approach *COCOB* (Orabona and Tommasi, 2017). For more details about these optimizers see Section 2.8.

To perform a fair comparison, we compared various hyperparameter combinations of commonly used hyperparameters for each optimizer. In addition, we utilized those combinations to analyze the hyperparameter sensitivity for each optimizer.

Since a grid search on ImageNet was too expensive, the best hyperparameter configuration of CIFAR-100's evaluation was used to test hyperparameter transferability.

A detailed explanation of the experiments including hyperparameters and data augmentations used, is given in Appendix A.2.3.

All in all, we trained over 4500 networks with TensorFlow 1.15 (Abadi *et al.*, 2016) on Nvidia Geforce GTX 1080 TI graphics cards (see Chapter 3). Since *PAL* is a line search approach, the predefined learning rate schedules of SGD and the generated schedules of *SLS*, *ALIG*, *SGDHD*, and *PAL* were compared. Due to normalization, *PAL*'s learning rate is given by $s_{\text{upd}_t}/||\boldsymbol{d}_t||$.

4.5.2 Results

A selection of our results on DenseNets is given in Figure 4.7. Figure 4.8,4.9 show the results of other architectures trained on CIFAR-10 and CIFAR-100, respectively. The results for ImageNet and Tolstoi are found in Appendix Figure A.4. A table with exact numerical results of all experiments is provided in Appendix A.2.4.

In most cases *PAL* decreases the training loss faster and to a lower value than the other optimizers (row 1 of Figures 4.7,4.8,4.9, and App. Figure A.4). Considering validation and test accuracy, *PAL* surpasses *ALIG*, *SGDHD* and *COCOB*, competes with *RMSProp* and *ADAM* but gets surpassed by *SGD* (rows 2,3 of Figures 4.7,4.8,4.9, and App. Figure A.4). However, *RMSProp*, *ADAM* and *SGD* were tuned with a step size schedule. If we compare *PAL* to their basic implementations without a schedule, which roughly corresponds to the first plateau reached in row 2 of Figures 4.7,4.8,4.9, and App. Figure A.4, *PAL* would surpass the other optimizers and shows that it can find a well performing step size schedule. This is especially interesting for problems for which default schedules might not work.

SLS decreases the training loss further than the other optimizers on a few problems but shows weak performance and poor generalization on most. This contrasts to the results of (Vaswani *et al.*, 2019), where *SLS* behaves robustly and excels. To exclude the possibility of errors on our side, we reimplemented the *SLS* experiment on ResNet34 and could reproduce a similar well performance as in (Vaswani *et al.*, 2019) (Appendix A.2.1). Our results suggest that the interpolation assumption on which *SLS* is based is not always valid for the considered tasks.

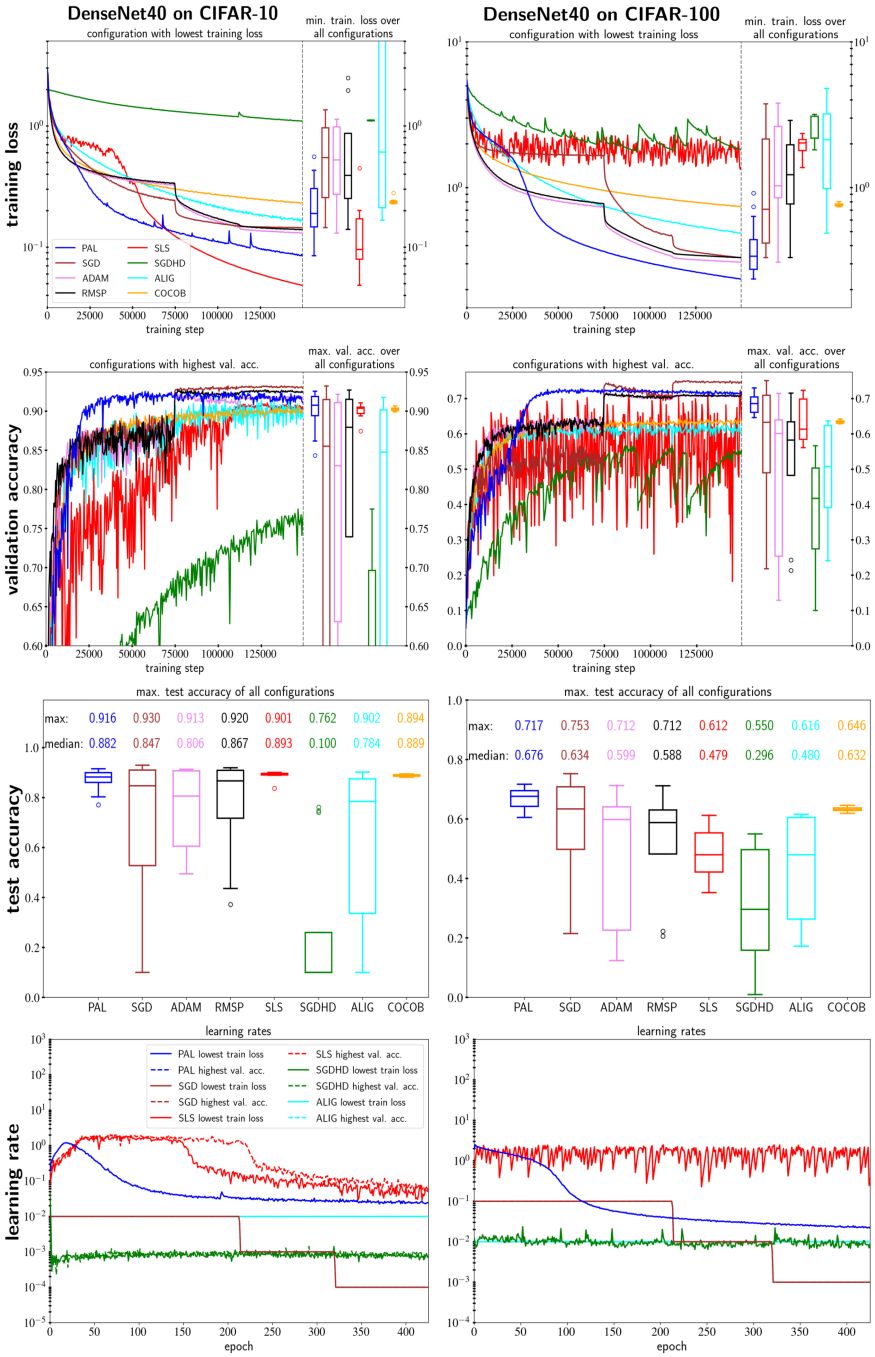


Figure 4.7: Comparison of *PAL* on **DenseNets** against *SLS*, *SGD*, *ADAM*, *RMSProp*, *ALIG*, *SGDHD*, and *COCOB* on training loss (row 1), val. acc. (row 2), test. acc. (row 3) and *SLS*, *SGD*, *ALIG*, *SGDHD*, and *PAL* on learning rates (row 4). Comparison is done across several datasets and models. Further results are found in Figure (4.8, 4.9 and Appendix Figure A.4). Results are averaged over 3 runs. Box plots result from comprehensive hyperparameter grid searches in plausible intervals. Learning rates are averaged over epochs. *PAL* surpasses, *ALIG*, *SGDHD*, and *COCOB* and competes well against all other optimizers.

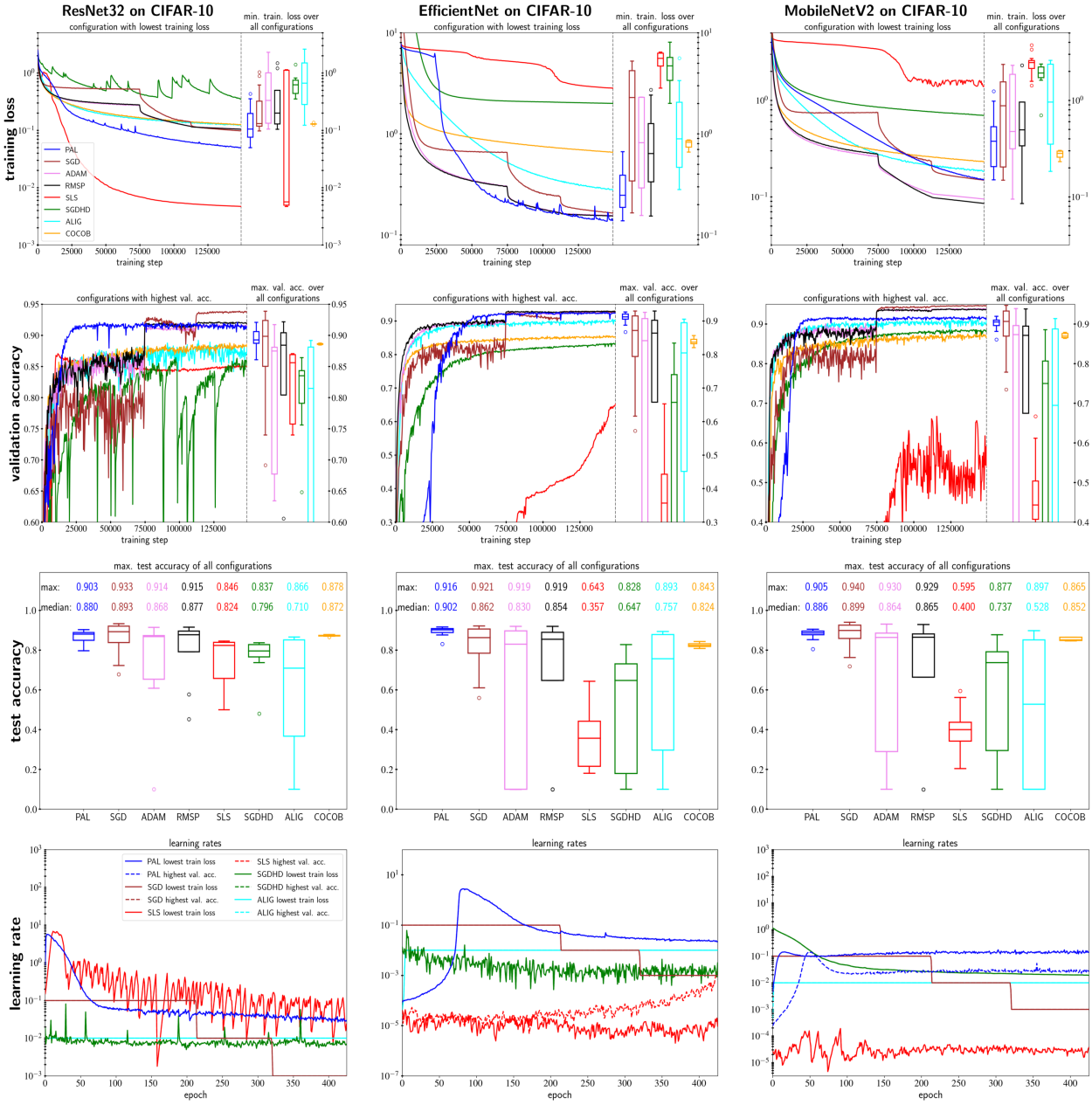


Figure 4.8: Comparison of *PAL* on **CIFAR-10** against *SLS*, *SGD*, *ADAM*, *RMSProp*, *ALIG*, *SGDHD*, and *COCOB* on training loss (row 1), val. acc. (row 2), test. acc. (row 3) and *SLS*, *SGD*, *ALIG*, *SGDHD* and *PAL* on learning rates (row 4). Results are averaged over 3 runs. Box plots result from comprehensive hyperparameter grid searches in plausible intervals. Learning rates are averaged over epochs. *PAL* surpasses *SLS*, *ALIG*, *SGDHD* and competes against all other optimizers. The learning rate schedule comparison shows that *PAL* performs competitive although elaborating significantly different schedules.

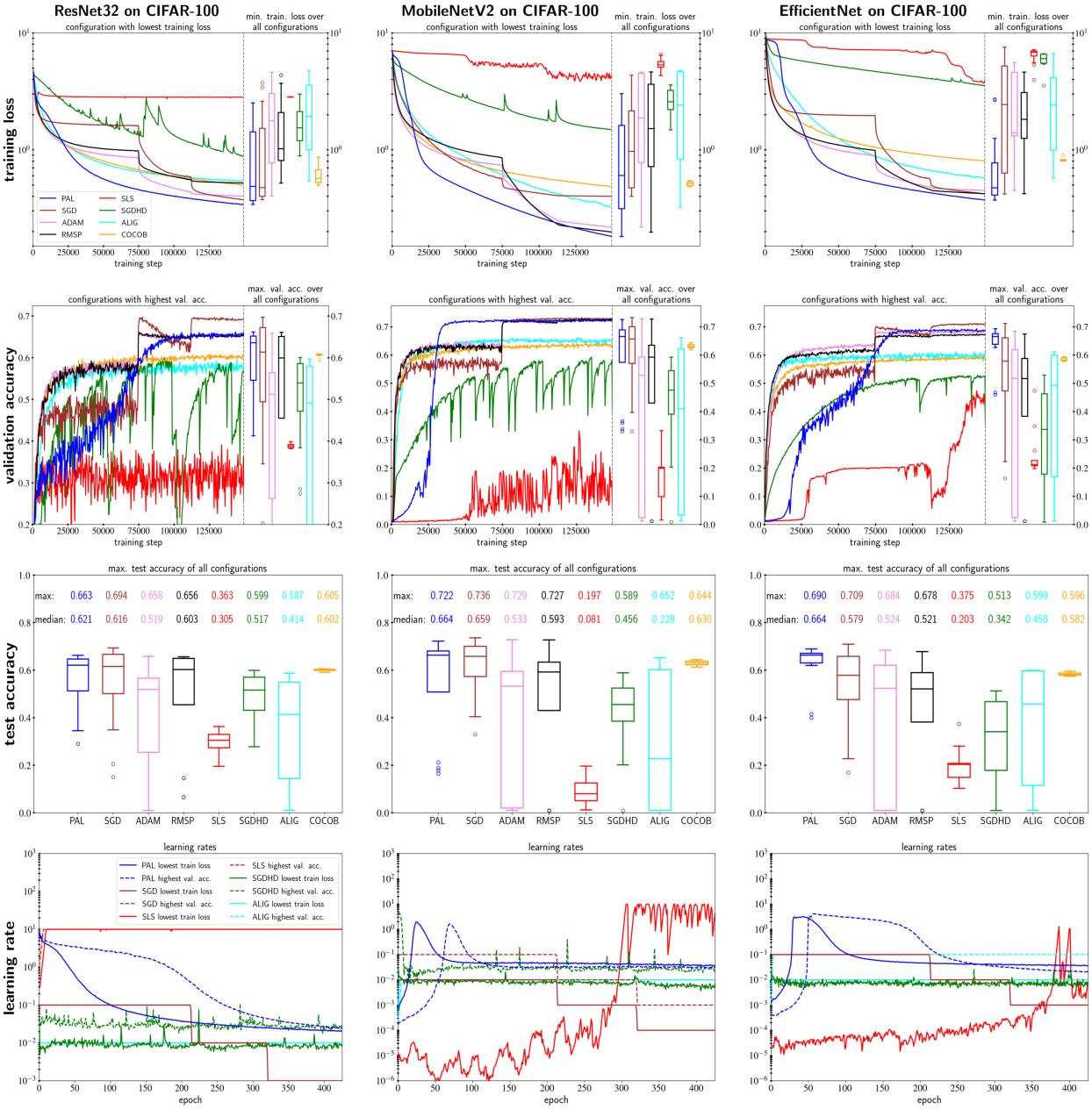


Figure 4.9: Comparison of *PAL* on **CIFAR-100** against *SLS*, *SGD*, *ADAM*, *RMSProp*, *ALIG*, *SGDHD*, and *COCOB* on training loss (row 1), val. acc. (row 2), test. acc. (row 3) and *SLS*, *SGD*, *ALIG*, *SGDHD* and *PAL* on learning rates (row 4)). Results are averaged over 3 runs. Box plots result from comprehensive hyperparameter grid searches in plausible intervals. Learning rates are averaged over epochs. *PAL* surpasses *SLS*, *ALIG*, *SGDHD* and competes against all other optimizers. The learning rate schedule comparison shows that *PAL* performs competitive although elaborating significantly different schedules.

Table 4.1: Required seconds per epoch of *PAL*, *SLS*, *ALIG*, *SGDHD*, *COCOB* and *SGD* on CIFAR-10. RMSP and ADAM reach a similar speed as SGD. The comparison was performed on a Nvidia Geforce GTX 1080 TI. *PAL* and *SLS* perform slower, since they have to measure additional losses, whereas the additional operations of *ALIG*, *SGDHD*, *COCOB* tend to be cheap.

	seconds / epoch					
network	PAL	SLS	SGD	ALIG	SGDHD	COCOB
ResNet32	20.9	21.7	10.7	11.0	11.1	16.4
MobilenetV2	53.2	52.4	34.1	34.01	34.2	36.6
EfficientNet	55.5	52.2	30.7	31.2	32.2	37.5
DenseNet40	88.8	87.5	59.7	61.3	64.6	61.4

Considering the box plots of Figures 4.7 and 4.9, which represent the sensitivity to hyperparameter combinations, one would likely try on a new unknown objective. We can see that *PAL* has a strong tendency to exhibit low sensitivity in combination with good performance. To emphasize this statement, a sensitivity analysis of *PAL*'s hyperparameters (Appendix Figure A.6) shows that *PAL* performs well on a wide range for each hyperparameter on a ResNet32.)

Considering the learning rate schedules of *PAL* (row four of Figures 4.7,4.8,4.9,A.4) we achieved unexpected results. *PAL*, which estimates the learning rate directly from approximated local shape information, does not follow a schedule that is similar to the one of *SLS*, *ALIG*, *SGDHD* or any of the commonly used handcrafted schedules such as piece-wise constant or cosine decay. However, it achieves similar results. An interesting side result is that *ALIG* and *SGDHD* tend to perform best if hyperparameters are chosen in a way that the learning rate is only changed slightly and, therefore, virtually an SGD training with a fixed learning rate is performed.

On wall-clock-time *PAL* performs as fast as *SLS* but slower than the other optimizers, which achieve similar speeds (Table 4.1). However, an automatic, well-performing learning rate schedule might compensate for the slower speed depending on the scenario.

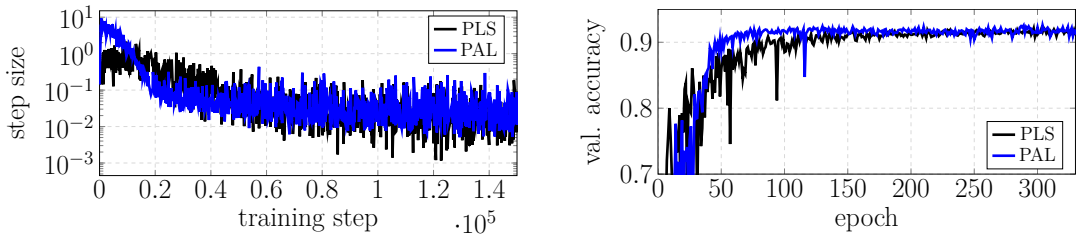


Figure 4.10: Comparison of *PAL* to *PLS* (Mahsereci and Hennig, 2015).

For the sake of completeness we also did a basic comparison to *PLS*: We used the only existing implementation of *PLS* (Mahsereci and Hennig, 2015) for TensorFlow 1 (Lukas Balles, 2017). This implementation is empirically improved and slightly variates from (Mahsereci and Hennig, 2015). However, the sum of squared gradients still has to be derived manually for each layer, which is a considerable amount of work for modern architectures. Consequently, we limited our comparison to a ResNet-32 trained on CIFAR-10. Figure 4.10 shows that *PAL* and *PLS* perform similarly in this scenario when default hyperparameters were used.

4.5.3 Influence of Dynamic Step Sizes and of the Direction Adaptation

This section analyzes, whether *PAL*'s performance originates from dynamically chosen step sizes or from the non-linear conjugate gradient-like update step adaptation. We consider EfficientNets trained on CIFAR-10, since for those the update step adaptation factor β is needed to achieve optimal results. We consider the following 6 scenarios:
1,2) *PAL* without update step adaptation ($\beta = 0$) and with and without dynamic step sizes (Figure 4.11 left).
3,4) *PAL* with an update step adaptation of $\beta = 0.2$ and with and without dynamic step sizes (Figure 4.11 middle).
5,6) *PAL* with an update step adaptation of $\beta = 0.4$ with and without fixed step sizes (Figure 4.11 right). The case with fixed step sizes result in normalized SGD (NSGD) with a adaptation factor β. As fixed update step size we use the measuring step size μ.

The results show that dynamic step sizes always increase the performance if direction adaptation is not applied and if it is applied in six out of eight cases. Direction adaptation can increase or decrease the performance in both, the dynamic and the fixed step size cases. The best performance is achieved with a direction

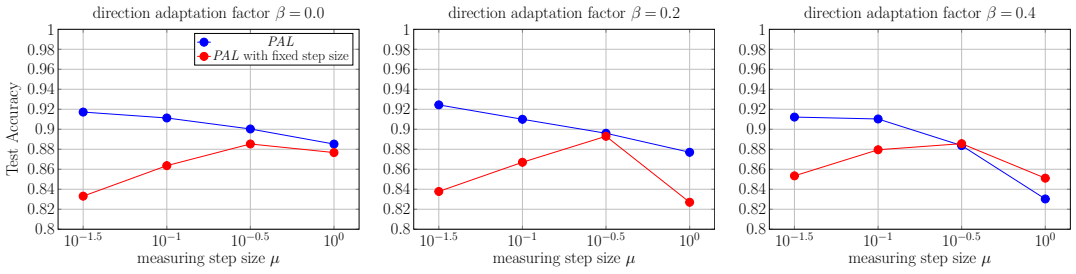


Figure 4.11: Analysis of the influences of dynamic step sizes and the direction adaptation factor β.

adaptation factor of 0.2 and a measuring step size of $10^{-1.5}$, which shows that both factors influence the best results in this scenario.

4.6 On the Exactness of Line Searches on mini-batch Losses

In this section, we investigate the general question of whether line searches, that estimate the position of a minimum of mini-batch losses exactly, are beneficial. In Figure 4.2 we showed that *PAL* can perform an almost exact line search on batch losses if we use a fixed update step adaptation factor (Section 4.4.3). However, *PAL*'s best hyperparameter configuration does not perform an exact line search (see Figure 4.12). Consequently, we analyzed how an exact line search, which exactly estimates a minimum along the line, behaves. We implemented an inefficient binary line search, which measured up to 20 values on each line to estimate the position of a minimum. The results shown in Figure 4.12, indicate that an optimal line search does not optimize well. Thus, the reason why *PAL* performs well is not the exactness of its update steps. In fact, slightly inexact update steps seem to be beneficial.
These results query Assumption 2, which assumes that the position of a minimum on a line in negative gradient direction of $\mathcal{L}_{\mathbb{B},t}$ is a suitable estimator for the minimum of the full-batch loss $\mathcal{L}$ on this line to perform a successful optimization process. To investigate this further, we measured $\mathcal{L}$ and the distribution of mini-batch losses for multiple SGD update directions on a ResNet32. Our results suggest, as exemplarily shown in Figure 4.13 that the position of $\mathcal{L}_{\mathbb{B}i}$'s minimum along a line in negative gradient direction is not always a good estimator for the position of $\mathcal{L}$'s corresponding minimum. This explains why exact line searches on the batch loss perform weakly.

Corollaries are that the empirical loss on the investigated lines also tends to be

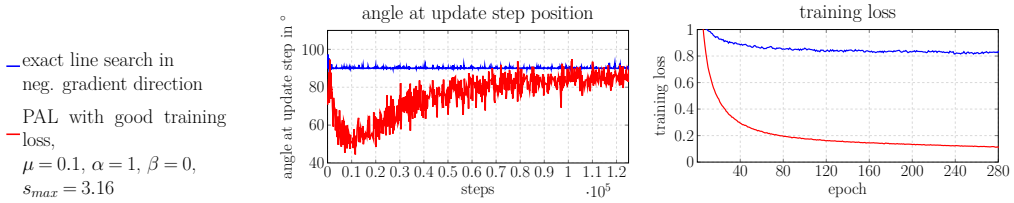


Figure 4.12: Comparison of *PAL* against an exact line search. The first plot shows the angle between the direction and gradient vector at the update step position. A ResNet32 was trained on CIFAR-10. One can observe that an exact line search (blue) exhibits poor performance.

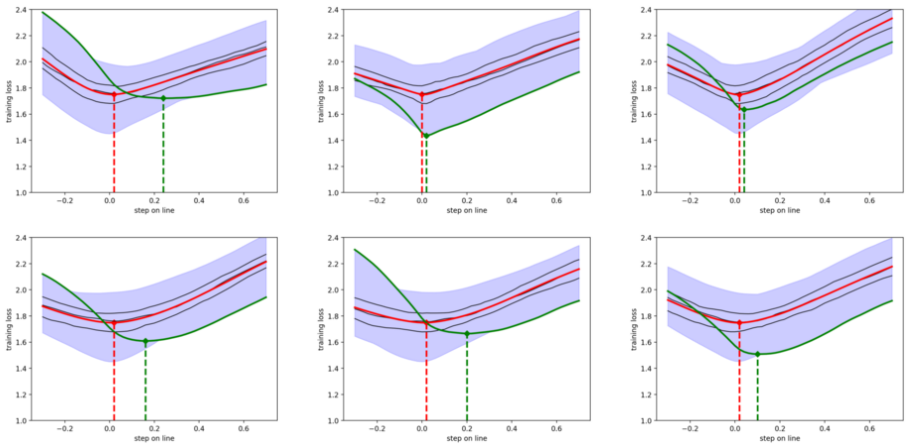


Figure 4.13: Distributions (blue) overall mini-batch losses along representative lines during a training of a ResNet32 on a subset of CIFAR-10. The full-batch loss, which is the mean value of the distribution, is given in red. Quartiles are given in black. The mini-batch loss, whose negative gradient defines the search direction, is given in green. It can be observed that the minimum of the green mini-batch loss is not always an adequate estimator of the minimum of the full-batch loss along the line.

locally convex and that the optimal step size tends to be smaller than the step size given by the mini-batch loss along such lines. This is a possible explanation why the slightly too narrow parabolic approximations of *PAL* without update step adaptation perform well.

4.7 PAL and Interpolation

This section analyzes how the interpolation condition is related to *PAL*'s performance. Formally, interpolation requires that the gradient with respect to each sample converges to zero at the optimum (see Assumption 1, page 16). We repeated the experiments of the *SLS* paper (see (Vaswani *et al.*, 2019) Section 7.2 and 7.3), which analyzes the performance on problems for which interpolation does or does not hold.

Figure 4.14 shows that *PAL* such as *SLS* converge faster to an artificial optimization floor on non-over-parameterized models ($k = 4$) of the matrix factorization problem of (Vaswani *et al.*, 2019, §7.2). In the interpolation case *PAL* and *SLS* converge linearly to machine precision. On the binary classification problem of (Vaswani *et al.*, 2019) Section 7.3, which uses a softmax loss and RBF kernels on the mushrooms and IJCNN datasets, we observe that *PAL* and *SLS* converge fast on the mushrooms task, for which the interpolation condition holds (Figure 4.15).

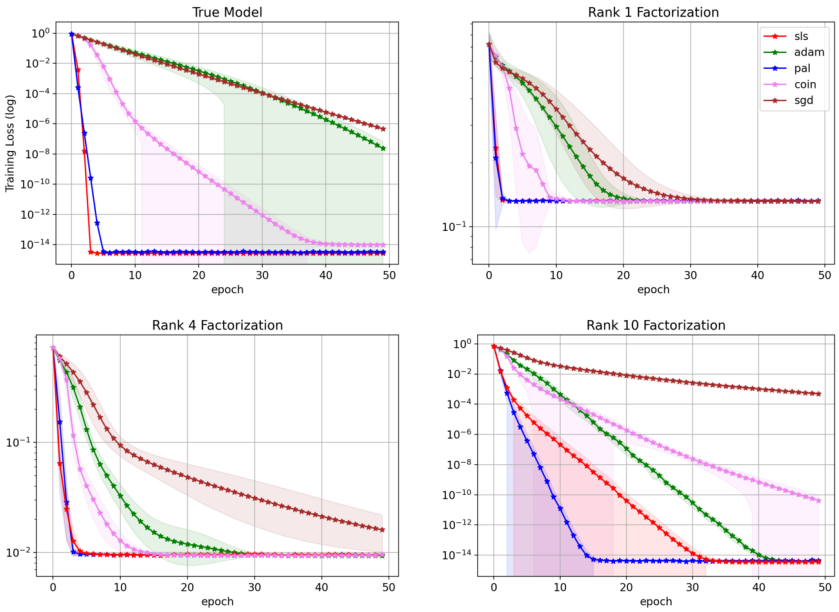


Figure 4.14: The matrix factorization problem of (Vaswani *et al.*, 2019, §7.2). For $k = 1$ and $k = 4$ interpolation does not hold. Rank 1 factorization is under-parameterized, whereas rank 4 and rank 10 factorizations are over-parameterized.

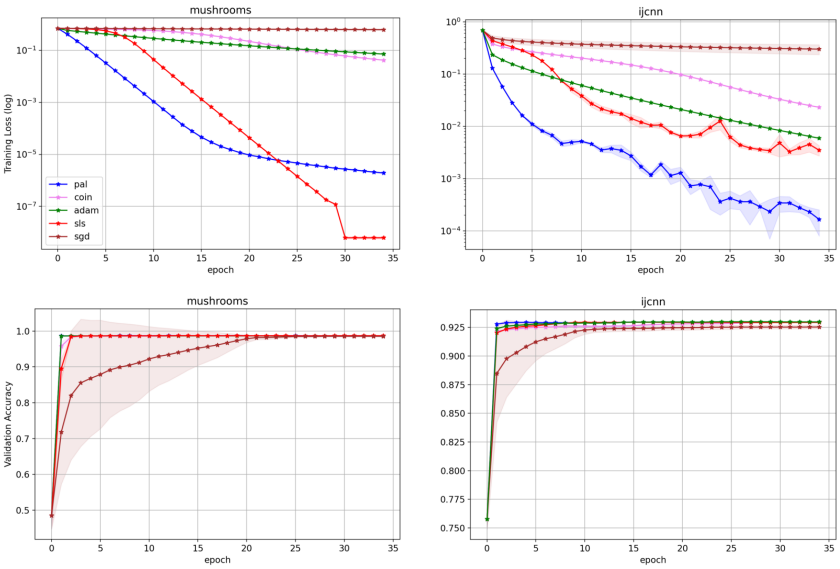


Figure 4.15: Binary classification task of (Vaswani *et al.*, 2019, §7.3) using a softmax loss and RBF kernels for mushrooms and ijcnn datasets. With RBF kernels, the mushrooms dataset is linearly separable in kernel-space with the selected kernel bandwidths, while the IJCNN dataset is not.

However, *PAL* converges faster on the IJCNN task, for which the interpolation condition does not hold.

The results indicate that the interpolation condition is beneficial for *PAL*, but, *PAL* performs also robust when the condition is likely not satisfied (see Figure 4.7,4.8,4.9,A.4). In those experiments *PAL* mostly performs competitive but *SLS* does not. However, the relation of the parabolic observation to interpolation needs to be investigated more closely in the future.

4.8 Conclusions and Outlook

This chapter tackled a major challenge in current optimization research for deep learning: to automatically find optimal step sizes for each update step. In detail, we focused on line search approaches to deal with this challenge. We introduced a robust and straightforward line search approach based on one-dimensional parabolic approximations of mini-batch losses. The introduced algorithm *PAL* is an alternative to *SGD* for objectives where default decays are unknown or do not work.

Loss functions of DNNs are commonly perceived as being highly non-convex. Our analysis suggests that this intuition does not hold locally since lines of loss landscapes across models and datasets can be approximated parabolically to high accuracy. This new knowledge might further help to explain why update steps of specific optimizers perform well.

To gain more comprehensive insights into line searches in general, we analyzed how an expensive but exact line search on batch losses behaves. Intriguingly, its performance is weak, which lets us conclude that the minor inaccuracies of the parabolic approximations are beneficial for training.

This chapter focused heavily on empirical observations of $\mathcal{L}_{\mathbb{B},t}$. However, such observations can only be exploited if $\mathcal{L}_{\mathbb{B},t}$ is a good estimator of $\mathcal{L}$. In other words, if the batch size is large enough. Therefore, the following chapter focuses on the empirical analysis of $\mathcal{L}$ along lines in SGD-update direction. In addition, we will explain why and how optimizers that rely on $\mathcal{L}_{\mathbb{B},t}$ perform on $\mathcal{L}$. Furthermore, we will consider the influence of batch sizes, which has been neglected so far since we have stuck to default batch sizes typically used in optimization. In Chapter 6 we will present some weaknesses of the *PAL* approach found later and based on the empirical findings of Chapter 5 we will introduce an even better line search approach for the stochastic setting in Chapter 6.

4.9 Retrospective from 02/2022

After publishing the *PAL* paper (Mutschler and Zell, 2020a), we conducted further research in this field and found that *PAL*'s performance is sensitive to the weight initialization and batch size used. In general, it does not get to its full potential when Pytorch's default weight initialization is used. Note that the experiments here were performed with TensorFlow 1.5. In addition, it is not trivial to find hyper-parameters that compete with SGD. Nevertheless, (Paren *et al.*, 2021) was able to reproduce *PAL*'s good performance and assured that it is one of the top-performing line search approaches, especially in the interpolation scenario. Furthermore, (Hao *et al.*, 2021) also reproduced *PAL*'s good performance and designed an improved version of *PAL* by combining it with conjugate gradient and momentum adaptation techniques.

Chapter 5

Empirically explaining SGD from a Line Search Perspective

This chapter provides a more detailed, empirically-based understanding of how $\mathcal{L}$ is shaped along the trajectory taken by SGD and how it changes during the training process from a line search perspective. Specifically, a resource expensive quantitative analysis of $\mathcal{L}$ along SGD trajectories of commonly used models trained on a subset of CIFAR-10 is performed. This chapter is based upon Mutschler and Zell (2021).

5.1 Introduction

Although the field of deep learning has made impressive progress in recent years, both in theory and application, little is known about why and how approaches work in detail. In general, deep learning approaches are based on vague intuitions in practice or rather strong assumptions in theory,[1] without providing comprehensive empirical evidence that their intuitions and assumptions hold (e.g.: (Smith, 2017; Rolínek and Martius, 2018; Rumelhart *et al.*, 1986; Berrada *et al.*, 2020; Vaswani *et al.*, 2019; Ioffe and Szegedy, 2015; He *et al.*, 2016; Huang *et al.*, 2017; Simonyan and Zisserman, 2015)).[2] Consequently, empirical analyses that search for a deeper reaching understanding and explain why specific approaches work are rare to find.

This is particularly valid for optimization, which, in this domain, is optimizing the mean of a stochastic loss function with an extremely high-dimensional parameter space. The landscape of such a loss function is generally assumed to be highly non-convex; however, recent works (Li *et al.*, 2018; Xing *et al.*, 2018; Mutschler and Zell, 2020a; Chae and Wilke, 2019; Mahsereci and Hennig, 2015; Goodfellow and Vinyals, 2015; Fort and Jastrzebski, 2019; Draxler *et al.*, 2018) claim that loss landscapes look rather simplistic for typical deep learning benchmarks used in optimization.[3] This is shown to be valid for $\mathcal{L}$ with low evidence and for $\mathcal{L}_{\mathbb{B}}$ with stronger evidence.

[1] E.g., convexity, lipschitz continuity, interpolation, skip connections, batch normalization.
[2] Better performance does not imply that the assumptions used are correct.
[3] Image classification on MNIST, SVHN, CIFAR-10, CIFAR-100 and ImageNet.

So far, there existed no detailed analysis of the relation of mini-batch losses to the full-batch loss to be optimized and of the actual performance of approaches using mini-batches on the full-batch loss. Globally, such an empirical analysis is not feasible in terms of resources and time, even if performed for a single model only. To nevertheless shed light on the subject, this work focuses on the quantitative analysis of $\mathcal{L}$ and $\mathcal{L}_{\mathbb{B}}$ along lines in SGD update step directions of a ResNet-20, a ResNet-18 (He *et al.*, 2016) and a MobileNetV2 (Sandler *et al.*, 2018) trained on a computationally feasible subset of CIFAR-10 (Krizhevsky *et al.*, 2009). Since the evaluation on each of the models supports our claims, we concentrate on the results of ResNet-20. Results for the other models are given in Appendix B. Note that the analysis of a small set of problems might not provide general evidence, and thus, our results have to be handled with care if applied to different scenarios.

Our core results are:

1. We provide further quantitative evidence that the full-batch loss along lines in update step direction behaves locally parabolically to a high degree (Sections 5.3,5.4).

2. We analyze the behavior of SGD (Rumelhart *et al.*, 1986), Parabolic Approximation Line Search (Mutschler and Zell, 2020a) and further approaches on the full-batch loss when trained on mini-batch losses (Section 5.5). We empirically show that there exists a leaning rate for which SGD always performs almost exact line searches on $\mathcal{L}$. The former is since the optimal update step size on $\mathcal{L}$ and the norm of $\mathcal{L}_{\mathbb{B},t}$'s gradient behave approximately proportional.

3. We consider the behavior of optimization approaches for different batch sizes (Section 5.6) and, from a new perspective, can quantitatively explain why increasing the batch size has virtually the same effect as decreasing the learning rate by the same factor, as experienced by (Smith *et al.*, 2018).

5.2 Closely Related Work

The following are extracts of relevant parts of Sections 2.6 (stochastic line searches), 2.9 (the simple loss landscape) and 2.10 (batch size and learning rate). These sections introduce the corresponding fields in more detail.

SGD trajectories: Similar to the work introduced in this chapter, (Xing *et al.*, 2018) analyzes the loss along SGD trajectories, but with less focus on line searches and the exact shape of the full-batch loss. (Jastrzebski *et al.*, 2019) and (Li *et al.*, 2020) consider second-order information along SGD trajectories. Where (Jastrzebski *et al.*, 2019) investigates the spectral norm of the Hessian (highest curvature) along the SGD trajectory and shows, inter alia, that SGD initially visits increasingly sharp regions.

The simple loss landscape: For more detailed information see Section 2.9. Loss landscapes of deep learning problems can generally be highly non-convex, and thus, hard to optimize. In practice, however, loss landscapes tend to be simple: (Li *et al.*, 2018) suggests that loss landscapes of networks with skip connections behave smoothly. (Xing *et al.*, 2018) shows that the full-batch loss along SGD update step directions is roughly convex and that SGD bounces of walls of a *valley like structure*. (Mutschler and Zell, 2020a; Chae and Wilke, 2019) reveal that $\mathcal{L}_{\mathbb{B}}$ along SGD update step directions is almost parabolically, and (Mutschler and Zell, 2020a) (see Chapter 4) suggests with weak empirical evidence that this also holds for the full-batch loss. Regarding this, (Mahsereci and Hennig, 2015) claims that $\mathcal{L}$ can be fitted by cubic splines along negative gradient directions. (Goodfellow and Vinyals, 2015) points out that on a straight path from initialization to solution, optimizers do not encounter any significant obstacles on the loss landscape. This chapter will provide further empirical evidence that the parabolic observation also holds for $\mathcal{L}$. Furthermore, all of the here introduced observations are supported by our empirical study.

Batch size and learning rate: (Smith *et al.*, 2018) claims that decreasing the learning rate has virtually the same effect as increasing the batch size by the same factor (see Section 2.9). In this chapter we explain from a new perspective, why this is the case. For more detailed information see Section 2.10.

5.3 The Empirical Method

For the empirical analysis, a deep learning problem has to be chosen, which is (a) computationally so cheap that the analysis of $\mathcal{L}$ can be performed in a reasonable amount of time and (b) still is representative for typical deep learning benchmarks used in optimization. Therefore, this work considers the problem of training a ResNet-20 (He *et al.*, 2016) on eight percent of the CIFAR-10 dataset(Krizhevsky *et al.*, 2009). ResNet-like architectures are widely used in practice and CIFAR-10 is a commonly used baseline. The dataset is scaled down, so that computations for one training process take less than three weeks on a single Nvidia Geforce GTX 1080 TI graphic card. Typical data augmentation is applied: Cropping, horizontal flipping and normalization with mean and standard deviation. Using PyTorch (Paszke *et al.*, 2019), the model is trained with SGD (Robbins and Monro, 1951) with learning rate $\lambda = 0.1$, which is the best performing λ chosen of a grid search over $\{10^{-i} | i \in \{0, 1, 1.3, 2, 3, 4\}\}$, batch size 128 and momentum β of 0 and 0.9 for 10000 steps.

Figure 5.1 shows the results of these SGD trainings. We note that the shown accuracies and losses do not provide much insight on what happens on a deeper level. E.g., it does not provide much information why SGD performs well. To

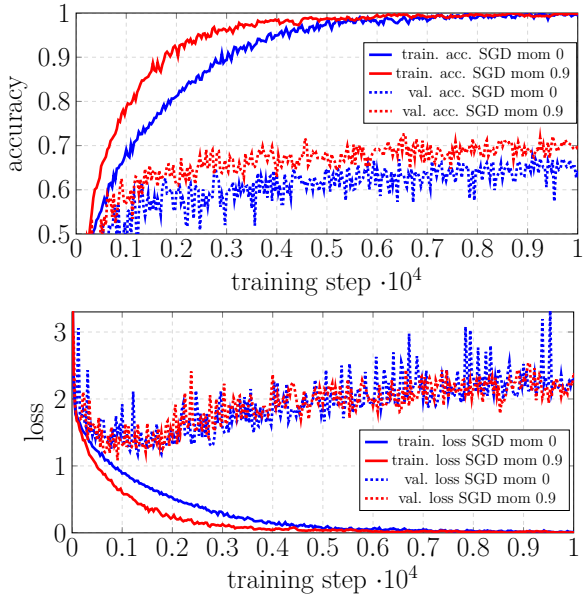


Figure 5.1: Training processes of a ResNet-20 trained on 8% of CIFAR-10 with SGD with momentum 0 and 0.9. In the course of this work, these processes will be analyzed in significantly deeper detail.

deal with this and further issues, the full-batch loss for each SGD update step is measured along lines in update step direction. We revise l from Equation 2.20 and consider a line along a direction $\boldsymbol{d}$ (similar to Equation 2.20). In detail, it is the full-batch loss $\mathcal{L}$ along a direction $\boldsymbol{d}$ through the current parameters $\boldsymbol{\theta}_t$ at update step t:

$$l_t : \mathbb{R} \to \mathbb{R},\ s \mapsto \mathcal{L}(\boldsymbol{\theta}_t + s\boldsymbol{d}) = \frac{1}{|\mathbb{D}|} \sum_{d \in T} L_d(\boldsymbol{\theta}_t + s\boldsymbol{d}), \tag{5.1}$$

where s is the step size along the line, L_d is the loss of sample d and $\mathbb{D}$ is the dataset. In the case of SGD without momentum, $\boldsymbol{d}$ is the negative unit gradient $-\boldsymbol{g}_{\mathbb{B}}/||\boldsymbol{g}_{\mathbb{B}}||$ of the original SGD trajectory whereas, in the case of SGD with momentum, $\boldsymbol{d}$ is the negative unit momentum direction $-\boldsymbol{m}/||\boldsymbol{m}||$.

For each of the 10000 update steps, we analyze the full-batch loss along the corresponding line in the interval $s \in [-0.5, 0.5]$ with a fine-grained resolution of 0.006. For each of the 167 sample step sizes along the line the sample loss of each element in the dataset is calculated. Then, all losses at a step size are averaged. All in all, this procedure requires more than 52 million inferences or 1.67 million epochs.

Representative visualizations of mini- and full-batch losses along such lines are given in Figure 5.2. The following is observed considering all 10000 visualizations: The full-batch loss along lines has a simple, almost parabolic shape and does not

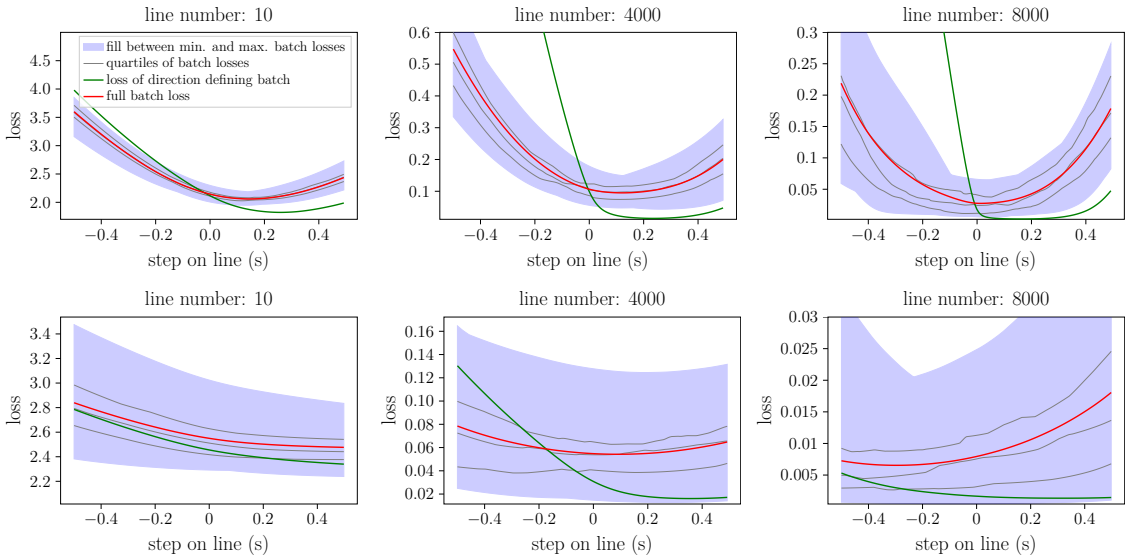


Figure 5.2: Losses along lines of the SGD training processes exhibit a simple shape. There is a significant difference between the full batch loss (red) and the loss of the direction defining batch (green). The loss of the direction defining batch is always steeper around 0. A mini-batch size of 128 is used. **Row 1:** SGD with momentum 0.0. **Row 2:** SGD with momentum 0.9. The mini-batch loss distributions exclude the direction defining mini-batch.

change substantially across all lines. Furthermore, the slope of the direction defining mini-batch's loss is consistently steeper than the full-batch loss around $s = 0$. The following sections provide further quantitative evidence that these observations hold.

In addition, we found the following interesting observations but do not investigate them further. There is a significant difference between the full-batch loss and the loss of the direction defining mini-batch. Further, the loss of the direction defining mini-batch does not follow the distribution of any other mini-batch loss along the line, especially for SGD without momentum. In addition, for SGD without momentum this loss is always lower and steeper than the other mini-batch losses.

5.4 On the Similarity of the Shape of full-batch Losses along Lines

The visualization of the full-batch loss along 10000 lines suggests that the shape of this loss does not vary significantly during the training process. For a more detailed investigation, the Mean Absolute Error (MAE) of the full-batch loss between each

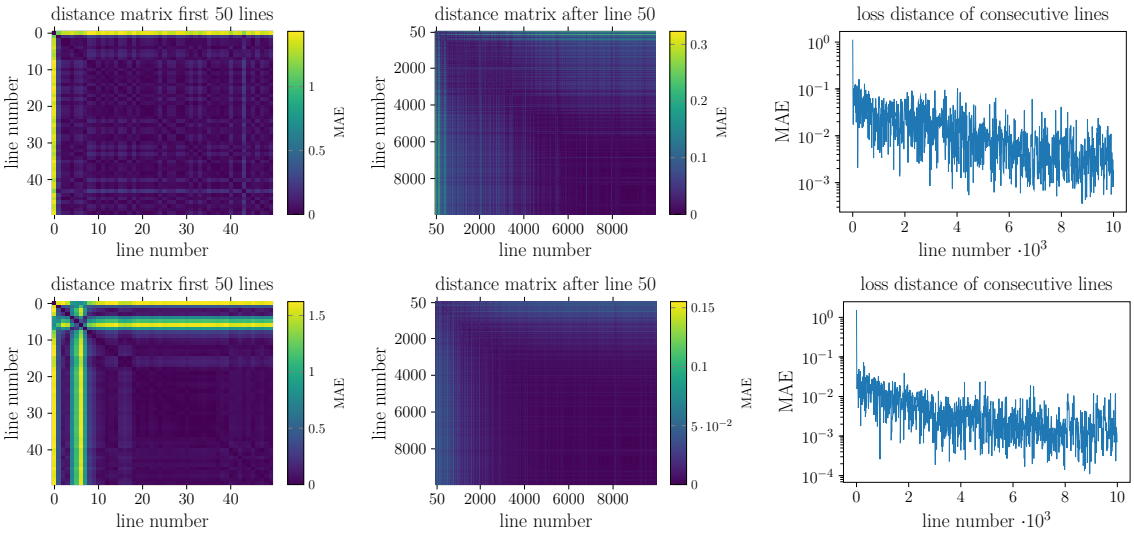


Figure 5.3: Distances of the shape of full-batch losses along lines in a window around the current position $s = 0$. **Row 1:** SGD without momentum. **Row 2:** SGD with momentum. Since the offset is not of interest the minimum is shifted to 0 on the y-axis. The distances are rather high for the first 10 lines (left). For the following lines the distances are less than 0.3 MAE (middle) and concentrate around 0.01. The MAEs of the full-batch loss of pairs of consecutive lines are given on the right.

pair of lines is analyzed on a relevant interval. Since solely the shape of the loss is of interest and not the offset, each loss along a line is shifted along the y-axis, such that the minimum is at zero. The interval from $s \in [-0.2, 0.2]$ is considered for SGD and from $s \in [-0.5, 0.5]$ for SGD with momentum. The latter ensures that the minimum position and the origin are always included. The resulting distance

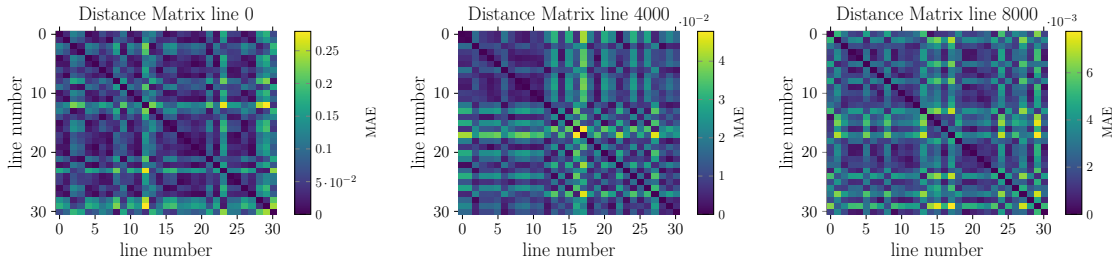


Figure 5.4: Distances (MAE) of the shape of full-batch losses along lines in multiple noisy gradient direction in a window of 0.3 around the line origin $s = 0$. The minimum is shifted to 0 on the y-axis. At fixed positions in parameter space the full-batch loss along lines in several noisy gradient directions reveals low distances. Those plots are representative for the 100 positions that we analyzed.

matrices are depicted in Figure 5.3. They show that **only the shapes of the full-batch loss of the very first lines vary strongly, whereas, later shapes behave more alike. In particular, the full-batch loss along consecutive lines behaves similarly.** This favors optimization with fixed step sizes, since the optimal update step does not change much. These results are also valid for the full-batch loss along each line in multiple noisy gradient directions starting from the same position in parameter space (Figure 5.4). This implies from an optimization point of view that it does not matter which of the descent directions is taken.

Figure 5.2 also indicates that **the full-batch loss along lines exhibits an almost parabolic shape locally** (core result 1). Figure 5.5 shows in detail that this is valid since the fitting error of a parabola is always low. In addition, we can see that the curvature of the fitted parabolas (i.e., the second directional derivative) decreases. This implies that **the approximated loss becomes flatter and suggests that SGD follows a simple valley-like structure which becomes continuously wider**. Considering the even faster-decreasing curvature of SGD with momentum, its valley becomes even wider (see also Figure 5.2). This might be a reason why SGD with momentum optimizes and generalizes better (Keskar *et al.*, 2017; Hochreiter and Schmidhuber, 1994). In accordance with (Jastrzebski *et al.*, 2019), we also found that the curvature is increasing rapidly during the very first steps and then decreases.

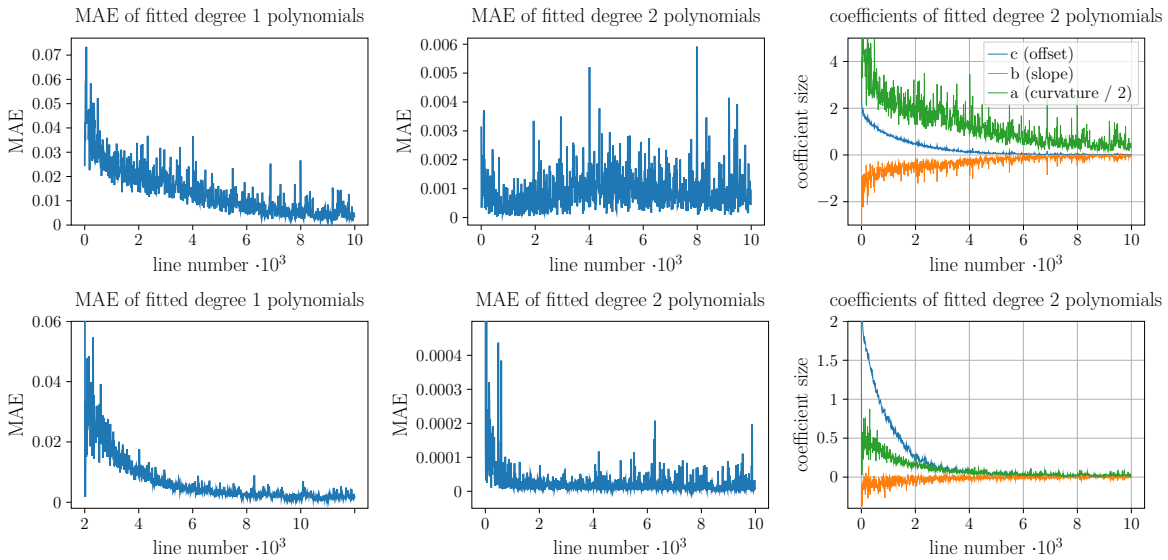


Figure 5.5: MAE of polynomial approximations of the full-batch loss of degree 1 and 2. **Row 1:** SGD without momentum. **Row 2:** SGD with momentum. Full-batch losses along lines can be well fitted by polynomials of degree 2. The slope of the approximation stays roughly constant whereas the curvature decreases.

Supporting results are obtained for ResNet-18 and for MobileNetV2, see Appendix Figures B.2, B.3, B.4, B.5.

5.5 On the Behavior of Line Search Approaches on the full-batch Loss

The previous section showed that the full-batch loss along lines in update step direction behaves parabolically and exhibits positive curvature. This means that $l(s) \approx as^2 + bs + c$ with $a > 0$ (see Equation 5.1). In the following, the performance of several parabolic approximation line searches applied on the direction defining mini-batch loss are analyzed. From now on, we concentrate on SGD without momentum, but, Figure B.1 (Appendix) shows that the upcoming results for SGD with momentum mostly support the derivations.

For SGD the mini-batch loss and its gradient $\boldsymbol{g}_{\mathbb{B}}$ are given at the origin ($s = 0$) of a line. In addition, the directional derivative, which is the negative norm of $\boldsymbol{g}_{\mathbb{B}}$, can be computed easily ($-\boldsymbol{g}_{\mathbb{B}}/||\boldsymbol{g}_{\mathbb{B}}|| \boldsymbol{g}_{\mathbb{B}}^T = -||\boldsymbol{g}_{\mathbb{B}}||$). To perform a parabolic approximation, either one additional loss along the line has to be considered or the curvature has to be estimated. PAL was introduced in Chapter 4. For repetition, its default update step was defined as:

$$s_{\text{pal}} = -\frac{b}{2a} = -\frac{l'_{\mathbb{B}}(0)\mu^2}{2(l_{\mathbb{B}}(\mu) - l_{\mathbb{B}}(0) - l'_{\mathbb{B}}(0)\mu)} \quad , \tag{5.2}$$

where $l_{\mathbb{B}}$ is the mini-batch loss along a line in the direction of $\boldsymbol{g}_{\mathbb{B}}$ and μ is the sample step size for the second loss. The second approach is a reinterpretation of SGD as a parabolic approximation line search with estimated curvature. SGD's update step is given as $-\lambda \boldsymbol{g}_{\mathbb{B}}$, where λ is the learning rate. Considering a normalized gradient

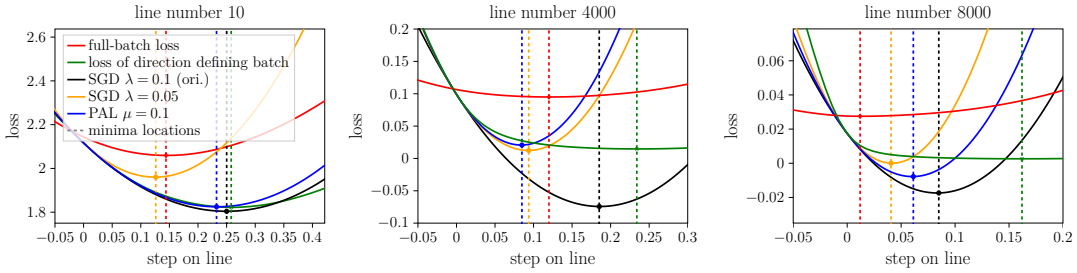


Figure 5.6: Several parabolic line approximations and their minimum positions on representative losses along lines. The optimal update step, from a local perspective, is depicted by the red dashed line. The other update steps are derived from the direction defining mini-batch loss.

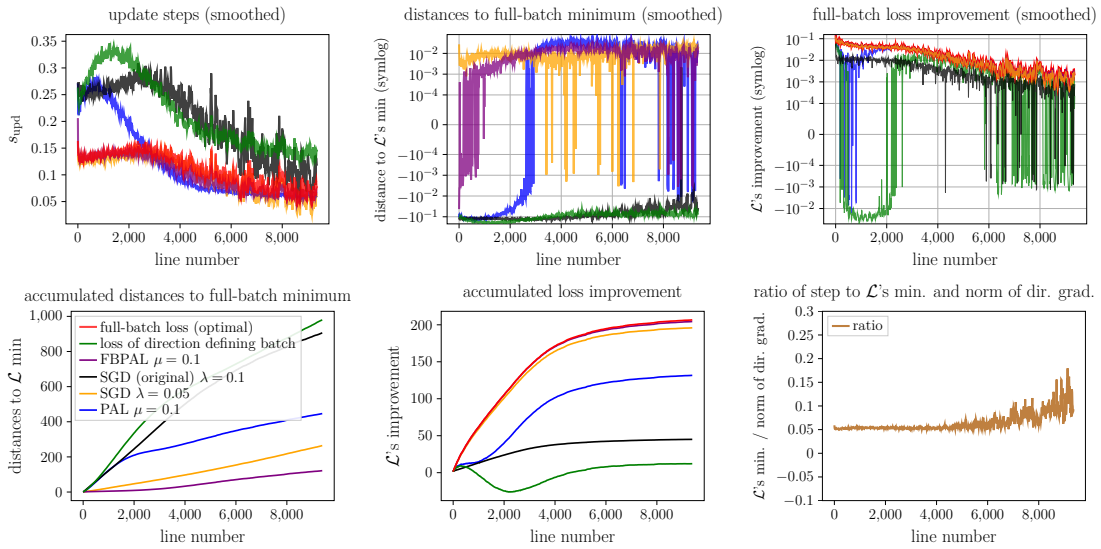


Figure 5.7: Several metrics to compare update step strategies: 1. update step sizes (s_{upd}). 2. the distance to the minimum of the full batch loss ($s_{\text{opt}} - s_{\text{upd}}$), which is the optimal update step from a local perspective. 3. the loss improvement per step given as: $l(0) - l(s_{\text{upd}})$ where s_{upd} is the update step of a strategy. Average smoothing with a kernel size of 25 is applied. The right lower plot shows almost proportional behavior between s_{opt} and the directional derivative of the direction defining mini-batch loss.

and defining $k = \frac{1}{\lambda}$ as the curvature, we get

$$-\lambda \boldsymbol{g}_{\mathbb{B}} = \lambda ||\boldsymbol{g}_{\mathbb{B}}|| \cdot \frac{-\boldsymbol{g}_{\mathbb{B}}}{||\boldsymbol{g}_{\mathbb{B}}||} = \frac{||\boldsymbol{g}_{\mathbb{B}}||}{k} \cdot \frac{-\boldsymbol{g}_{\mathbb{B}}}{||\boldsymbol{g}_{\mathbb{B}}||} = -\frac{\frac{-\boldsymbol{g}_{\mathbb{B}}}{||\boldsymbol{g}_{\mathbb{B}}||}\boldsymbol{g}_{\mathbb{B}}^T}{k} \cdot \frac{-\boldsymbol{g}_{\mathbb{B}}}{||\boldsymbol{g}_{\mathbb{B}}||} = -\frac{\text{first directional derivative}}{\text{second directional derivative}} \cdot \text{direction}. \tag{5.3}$$

Note that the latter is a Newton update step.

To get a first intuition of how these approaches operate, several parabolic approximations and their resulting update steps on representative lines are shown in Figure 5.6.

The next step is to compare several update step strategies using three metrics. Beforehand, we have to define s_{opt} as the step size to the minimum of the full-batch loss along a line, which is the optimal update step size from a local perspective. s_{upd} is the update step size of an arbitrary optimization strategy considered. The metrics are: the update step size s_{upd}, the distance of s_{upd} to the minimum of the full-batch loss ($s_{\text{opt}} - s_{\text{upd}}$), and the loss improvement per step, given as: $l(0) - l(s_{\text{upd}})$, where l is the full-batch loss along a line (see Equation 5.1). Note that this improvement measure does not represent actual training performance since the next considered line is independent of the previous update step size. This is valid for all strategies except for SGD since we consider its training process. However, it does represent

the performance on full-batch losses along lines, which are likely to occur during training.

Figure 5.7 shows that some strategies exhibit varying behavior on the metrics. To strengthen our previous observation, a parabolic approximation on the full-batch loss (FBPAL) yields almost optimal performance. Surprisingly, SGD with $\lambda = 0.05$ estimates the minima of the full-batch loss almost as well. This is because **the step to the minimum of the full-batch loss s_{opt} is almost proportional to the directional derivative ($-||\boldsymbol{g}_{\mathbb{B}}||$) of the direction defining mini-batch loss** (core result 2), as shown in the lower plot of Figure 5.7. We observe that the variance becomes larger during the end of the training, and thus the proportionality holds less. **This almost proportional behavior explains why a constant learning rate can lead to a good performance, since it is sufficient to control the update step size with the norm of the noisy mini-batch gradient**. In practice, however, this locally optimal learning rate is unknown. The globally best performing learning rate of 0.1 always does a step far beyond the locally optimal step. The latter is what (Xing *et al.*, 2018) described as *bouncing off walls of a valley-like structure*. Contrary to their intuition, we have not found any boundaries at all in the valley. Finally, Figure 5.7 suggests that **exact line searches on the mini-batch loss perform poorly**.

Supporting results are obtained for SGD with momentum, for ResNet-18 and for MobileNetV2 see Appendix Figures B.1, B.6, B.7, B.8, B.9. However, in the case of SGD with momentum the line search is constantly not as exact.

Combining the last results suggest that **the locally optimal step size s_{opt} can be well approximated by a Newton step on the full-batch loss or by a simple proportionality**:

$$s_{\text{opt}} \approx -\frac{-||\boldsymbol{g}_{\mathcal{L}}||}{\frac{\boldsymbol{g}_{\mathcal{L}}}{||\boldsymbol{g}_{\mathcal{L}}||}\boldsymbol{H}_{\mathcal{L}}\frac{\boldsymbol{g}_{\mathcal{L}}^T}{||\boldsymbol{g}_{\mathcal{L}}||}} \approx c \cdot -||\boldsymbol{g}_{\mathcal{L}_{\mathbb{B}}}||, \tag{5.4}$$

where $\boldsymbol{g}_{\mathcal{L}}$ and $\boldsymbol{g}_{\mathcal{L}_{\mathbb{B}}}$ are the gradients of $\mathcal{L}$ and $\mathcal{L}_{\mathbb{B}}$, respectively. $\boldsymbol{H}_{\mathcal{L}}$ is the Hessian of $\mathcal{L}$ at $\boldsymbol{\theta}_t$. However, **on a global perspective a step size larger than s_{opt} can perform better, although it yields locally lower improvement** (Figure 5.8).

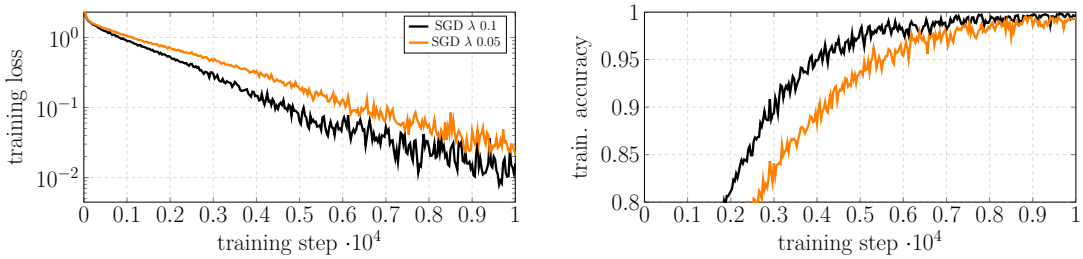


Figure 5.8: SGD with a locally optimal learning rate of 0.05 performs worse than SGD with a globally optimal learning rate of 0.01. Trainings are performed on a ResNet-20 and 8% of CIFAR-10 with SGD without momentum.

5.6 On the Influence of the Batch Size on Update Steps

This section analyzes to which extent the performance of SGD and PAL changes with varying batch sizes. In addition, we show why, on the losses along lines measured, increasing the batch size has almost the same effect as decreasing the learning rate by the same factor, as suggested by (Smith *et al.*, 2018).

The presented results are simplified, assuming that the SGD trajectory keeps identical with changing batch size. Thus, the same losses over lines can be considered. The original batch size is 128. For larger batch sizes, additional sample losses from the set of all measured losses are drawn without replacement. For smaller batch sizes, the sample losses with the highest directional derivatives are removed, assuming that for smaller batch sizes steeper steepest directions are found.

The upper plots of Figure 5.9 show that SGD performs significantly worse for smaller batch sizes than PAL does. Both approaches become significantly more accurate at larger batch sizes. A batch size of 512 is already sufficient to perform

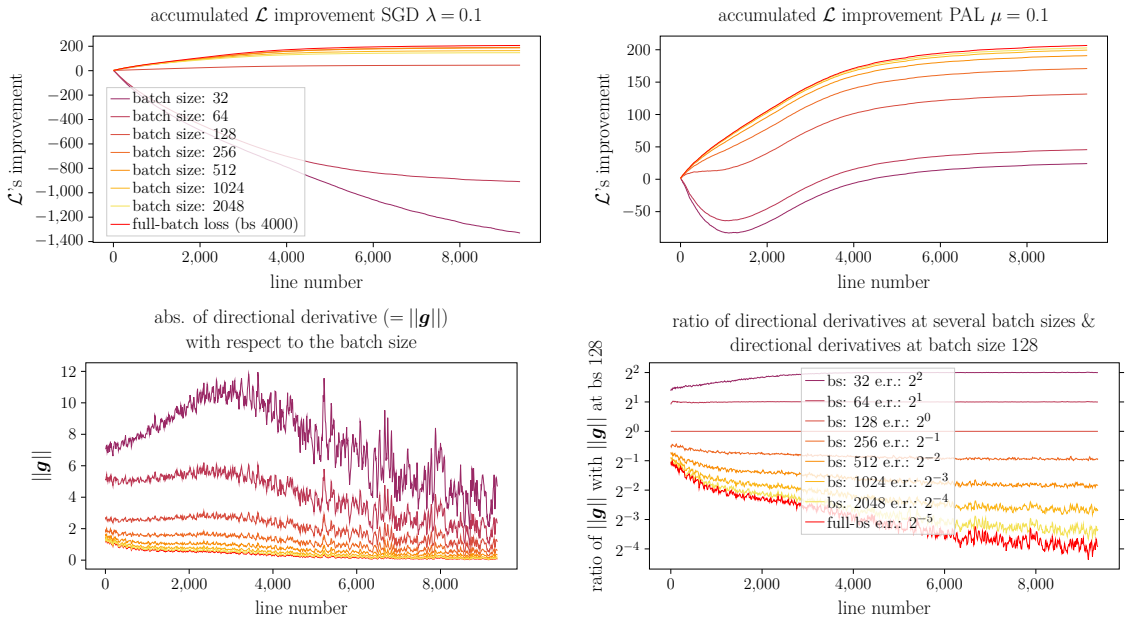


Figure 5.9: Row 1 Comparing the influence of the batch size on the loss improvement. Left: SGD with the original learning rate of 0.1. Right: PAL. **Note the different offsets and scaling on the y-axes**. **Row 2:** Analysis of the relation of the batch size to the absolute directional derivative (= gradient norm) which shows in detail that increasing the batch size has a similar effect as decreasing the learning rate by the same factor. **e.r.** stands for expected ratio.

.

almost optimally.

(Smith *et al.*, 2018) shows that when training a ResNet-50 (He *et al.*, 2016) on ImageNet (Deng *et al.*, 2009), increasing the batch size has virtually the same effect as decreasing the learning rate by the same factor. Their interpretation is based on the noise on the full-batch gradient introduced by mini-batches, whereas, we argue from the perspective of mini-batch losses. The SGD update step length on losses along a line is the absolute of the learning rate times the directional derivative ($\lambda \cdot |l'_m(0)| = \lambda \cdot ||\boldsymbol{g}_{\mathbb{B},t}||$). The lower left plot of Figure 5.9 shows that with larger batch sizes, the absolute of the directional derivative, and thus the step size, decreases. This can be figuratively explained with the help of Figure 5.2. As the batch size increases, the loss of the direction defining batch becomes more similar to the full-batch loss; consequently, the absolute of the directional derivative decreases. The lower plot of Figure 5.9 shows by which factor the directional derivative is divided when the batch size is multiplied by a factor. **For batch size 32 to 256 the assumption that if the batch size is increased by a factor, then the update step size decreases by the same factor, is valid during the whole training** (core result 3). For larger batch sizes, the directional derivative is divided by a lower factor at the beginning of the training, then the batch size is multiplied but converges towards the same factor during the training. Based on the data collected, we cannot estimate the momentum term for a different batch size for each line; therefore, this analysis was not performed for SGD with momentum. Supporting results are obtained on ResNet-18 (He *et al.*, 2016) and a MobileNetV2 (Sandler *et al.*, 2018), see appendix Section B.2 Figure B.10 and B.11.

5.7 Discussion and Outlook

With this work, we provided a more comprehensive understanding of what happens in detail during SGD training from a line search perspective. Inter alia, we quantitatively showed that the full-batch loss along lines in update step direction locally is highly parabolic. Further on, we found a learning rate for which SGD always performs an almost optimal line search. This questions if line searches for deep learning can ever outperform SGD in general. Finally, we quantitatively analyzed the relation of learning rate and batch size in detail and provided a new perspective on why increasing the batch size has almost the same effect as decreasing the learning rate by the same factor.

We have to emphasize that this work only focused on a small set of representative problems. Therefore, our results have to be handled with care. To get a more general view about the behavior of SGD and other optimizers across models and datasets, we propose to repeat these or similar experiments for as many times as possible. This can be easily done with the published code but is highly time-consuming (see `https://github.com/cogsys-tuebingen/empirically_expla`

`ining_sgd_from_a_line_search_perspective`).

In general, we want to emphasize that a prospective goal of future studies in deep learning should be, beyond reporting good results, to provide empirical evidence that the assumptions used hold.

The following chapter will exploit the observations we found to design an improved version of the *PAL* line search. This version achieves better performance and is more robust to changing batch sizes due to the empirically validated relation of batch sizes and update steps.

Chapter 6

Large Batch Parabolic Approximation Line Search

In this chapter, we exploit the observations made in Chapter 5 to develop a more robust, effective, and efficient line search approach than *PAL* (introduced in Chapter 4). The introduced approach outperforms other line searches introduced for deep learning in most cases and performs equally well as SGD with momentum tuned by a piece-wise constant learning rate schedule. In addition, it is easy to understand why and when this algorithm works since it is directly derived from observations. This chapter is based on Mutschler *et al.* (2021).

6.1 Introduction

Automatic determination of an appropriate and loss function-dependent learning rate schedule to train models with stochastic gradient descent (SGD) or similar optimizers is still not solved satisfactorily for deep learning tasks. The long-term goal is to design optimizers that work out-of-the-box for a wide range of deep learning problems without requiring hyperparameter tuning. Therefore, although well-working hand-designed schedules such as *piece-wise constant learning rate* or *cosine decay* exist (see (Loshchilov and Hutter, 2017; Smith, 2017)), it is desirable to infer problem dependent and better learning rate schedules automatically.

This chapter builds on empirical findings from chapter 5; among those are that the full-batch loss tends to have a simple parabolic shape in SGD update step direction (Mutschler and Zell, 2021, 2020a) (see Figure 6.1) and that the trend of the optimal update step changes slowly (Mutschler and Zell, 2021) (see Figure 6.2). Exploiting these and further observations, we introduce a line search approach, approximating the full-batch loss along lines in SGD update step direction with parabolas. One parabola is sampled over several batches to obtain a more exact approximation of the full-batch loss. The learning rate is then derived from this parabola. As the trend of the locally optimal update step-size on the full-batch loss changes slowly, the line search only needs to be performed occasionally; usually, every 1000th step.

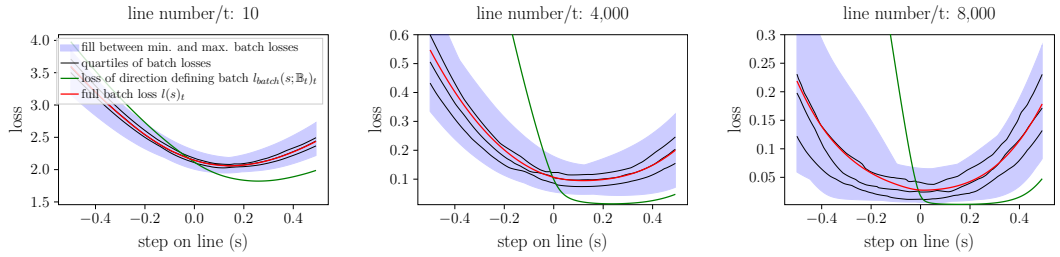


Figure 6.1: Losses along the lines of the SGD training processes exhibit a parabolic shape. The loss of the direction defining mini-batch (green) is excluded from the distribution of mini-batch losses to show that it is significantly different. This makes line searches on it unfavorable. In addition, the parabolic property articulates stronger for the full-batch loss (red); thus, this work aims to approximate it efficiently with a parabola.

The major contribution of this chapter is the combination of recent empirical findings to derive a line search method, which is built upon real-world observations and less on theoretical assumptions. This method outperforms the most prominent line search approaches introduced for deep learning across models and datasets usually considered in optimization for deep learning, in almost all experiments. In addition, it is on par with SGD with momentum tuned with a piece-wise constant learning rate schedule.

The second important contribution is that we analyze how the considered line searches perform under high noise that originates from small batch sizes. While all considered line searches perform poorly -mostly because they rely on mini-batch losses only-, our approach adapts well to increasing noise by iteratively sampling larger batch sizes over several inferences.

The chapter is organized as follows: Section 6.2 provides an overview of related work. Section 6.3 derives our line search approach and introduces its mathematical and empirical foundation in detail. In Section 6.4 we analyze the performance of our approach across datasets, models, and gradient noise levels. Also, a comprehensive hyperparameter, runtime, and memory consumption analysis is performed. Finally, we end with a discussion including limitations in Sections 6.5 & 6.6.

6.2 Closely Related Work

The following sums up related work for this chapter. Some of the mentioned works got explained in detail in Chapter 2.

Deterministic line searches: (see also Section 2.5) According to Nocedal and Wright (2006, §3), line searches are considered a solved problem in the noise-free case. However, such methods are not robust to gradient and loss noise and often fail in such a scenario since they shrink the search space inadequately or use too inexact

approximations of the loss. Further, Nocedal and Wright (2006, §3.5) introduces a deterministic line search using parabolic and cubic approximations of the loss, which motivated our approach.

Line searches on mini-batch and full-batch losses and why to favor the latter. (See also Section 2.6) The following motivates the goal of our work to introduce a simple, reasonably fast, and well-performing line search approach that approximates the full-batch loss.

Many exact and inexact line search approaches for deep learning are applied on mini-batch losses: i.e., ALI-G, SGD-HD, SLS, PAL (see Sections 2.8, 2.6.3, and Chapter 4). In Chapter 4, we approximated an exact line search by estimating the minimum of the mini-batch loss along lines with a one-dimensional parabolic approximation. The other approaches perform inexact line searches by estimating positions of the mini-batch loss along lines, which fulfill specific conditions. Such, inter alia, are the Goldberg, Armijo, and Wolfe conditions (see Nocedal and Wright (2006)). For these, convergence on convex stochastic functions can be assured under the interpolation condition (see Section 2.6.3 and Assumption 1), which holds if the gradient with respect to each batch converges to zero at the optimum of the convex function. Under this condition, the convergence rates match those of gradient descent on the full-batch loss for convex functions (Vaswani *et al.*, 2019). However, relying on those assumptions and on mini-batch losses only does not lead to robust optimization, especially not if the gradient noise is high, as will be shown in Section 6.4. In Chapters 4, 5, we even showed that exact line searches on mini-batch losses are not working at all.

Line searches on the non-stochastic full-batch loss show linear convergence on any deterministic function that is twice continuously differentiable, has a relative minimum, and only positive eigenvalues of the Hessian at the minimum (see Luenberger *et al.* (1984)). In addition, they are independent of gradient noise. Therefore, it is reasonable to consider line searches on the full-batch loss. However, these are cost-intensive since a massive amount of mini-batch losses for multiple positions along a line must be determined to measure the full-batch loss.

Probabilistic Line Search (PLS, see Section 2.6.5) Mahsereci and Hennig (2015) address this problem by performing GP Regressions, which result in multiple one-dimensional cubic splines. In addition, a probabilistic belief over the first (aka Armijo condition) and second Wolfe condition is introduced to find appropriate update positions. The major drawback of this conceptually appealing method is its high complexity and slow training speed. A different approach working on the full-batch loss is GOLSI (see Section 2.6.4) (Kafka and Wilke, 2019). It approximates a line search on the full-batch loss by considering consecutive noisy directional derivatives whose noise is considerably smaller than the noise of the mini-batch losses.

Empirical properties of the loss landscape: In deep learning, loss landscapes of the true loss (over the whole distribution), the full-batch loss, and the mini-

batch loss can, in general, be highly non-convex. However, to efficiently perform a line search, some properties of these losses have to be apparent. Little is known about such properties from a theoretical perspective; however, several works suggest that loss landscapes tend to be simple and have some characteristic properties: (Mutschler and Zell, 2021; Li *et al.*, 2018; Xing *et al.*, 2018; Mutschler and Zell, 2020a; Chae and Wilke, 2019; Mahsereci and Hennig, 2015; Goodfellow and Vinyals, 2015; Fort and Jastrzebski, 2019; Draxler *et al.*, 2018).

Mahsereci and Hennig (2015) suggest, according to our observations, that the full-batch loss $\mathcal{L}$ along lines in negative gradient directions tend to exhibit a simple shape for a set of deep learning problems. This set includes at least classification tasks on CIFAR-10, CIFAR-100, and ImageNet. In chapter 5, we sampled the full-batch loss along the lines in SGD update step directions. This was done for $10,000$ consecutive SGD and SGD with momentum update steps of a ResNet18's, ResNet20's and MobileNetv2's training process on a subset of CIFAR-10. Representative plots of their $10,000$ measured full-batch losses along lines are presented in Figure 6.1. Relevant insights and found properties of these works will be introduced and exploited to derive our algorithm in Section 6.3. See also Section 2.9 for further information.

Using the batch size to tackle gradient noise: Besides decreasing the learning rate, increasing the batch size remains an important choice to tackle gradient noise. McCandlish *et al.* exploits empirical information to predict the largest practical batch size over datasets and models. De *et al.* adaptively increases the batch size over update steps to assure that the negative gradient is a descent direction. Smith and Le introduces the *noise scale* (see Section 2.10), which controls the magnitude of the random fluctuations of consecutive gradients interpreted as a differential equation. The latter leads to the observation that increasing the batch size has a similar effect as decreasing the learning rate (Smith *et al.*, 2018), which is exploited by our algorithm. See Section 2.10 for further information.

6.3 The Approach

6.3.1 Mathematical Foundations

In this subsection, we revise the mathematical background relevant for line searches and challenges that must be solved in order to perform line searches in deep learning. This in parts, is a summary of Section 2.6.1 slightly adapted to fit to the current case.

We consider the problem of minimizing the full-batch loss $\mathcal{L}$, which is the mean over a large amount of sample losses L_d:

$$\mathcal{L} : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{D}|} \sum_{d \in \mathbb{D}} L_d(\boldsymbol{\theta}), \tag{6.1}$$

where $\mathbb{D}$ is a finite dataset and $\boldsymbol{\theta}$ are n parameters to optimize. To increase training speed generally, a mini-batch loss $\mathcal{L}_{\mathbb{B}}$, which is a noisy estimate of $\mathcal{L}$, is considered:

$$\mathcal{L}_{\mathbb{B}} : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{B}|} \sum_{d \in \mathbb{B} \subset \mathbb{D}} L_d(\boldsymbol{\theta}), \tag{6.2}$$

with $|\mathbb{B}| \ll |\mathbb{D}|$. We define the mini-batch gradient at step t as $\boldsymbol{g}_{\mathbb{B},t} \in \mathbb{R}^n$ which is $\nabla_{\boldsymbol{\theta}_t} \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}_t)$.

For our approach, we need the full-batch loss along the direction of the negative normalized gradient of a specific mini-batch loss. At optimization step t with current parameters $\boldsymbol{\theta}_t$ and a direction defining batch $\mathbb{B}_t$, $\mathcal{L}_{\mathbb{B}}$ along a line with origin $\boldsymbol{\theta}_t$ in the negative direction of the normalized batch gradient $\hat{\boldsymbol{g}}_{\mathbb{B},t} = \boldsymbol{g}_{\mathbb{B},t} / ||\boldsymbol{g}_{\mathbb{B},t}||$ is given as:

$$l_{\mathbb{B},t} : \mathbb{R} \to \mathbb{R},\ s \mapsto \mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}_t + s \cdot -\hat{\boldsymbol{g}}_{\mathbb{B}_t,t}), \tag{6.3}$$

where s is the step size along the line. The corresponding full-batch loss along the same line is given by:

$$l_t : \mathbb{R} \to \mathbb{R},\ s \mapsto \mathcal{L}(\boldsymbol{\theta}_t + s \cdot -\hat{\boldsymbol{g}}_{\mathbb{B}_t,t}). \tag{6.4}$$

Let the step size to the first encountered minimum of l_t be $s_{\min,t}$.

Two major challenges have to be solved in order to perform line searches on $\mathcal{L}$:

1. To measure l_t exactly, it is required to determine every $L_d(\boldsymbol{\theta}_t + s \cdot -\hat{\boldsymbol{g}}_{\mathbb{B}_t,t})$ for all $d \in \mathbb{D}$ and for all step sizes s on a line.

2. To assure convergence line searches have to be performed in a descent direction (De *et al.*, 2016). The simplest form is the direction of steepest descent (Luenberger *et al.*, 1984). Therefore, the full-batch gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L} : \mathbb{R}^n \to \mathbb{R}^n,\ \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{D}|} \sum_{d \in \mathbb{D}} \nabla L_d(\boldsymbol{\theta}_t)$ has to be approximated.

To be efficient, l_t has to be approximated sufficiently well with as little data points d and steps s as possible, and one has to use as little d as possible to approximate $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ approximated sufficiently well. Such approximations are highly dependent on properties of $\mathcal{L}$. Due to the complex structure of deep neural networks, little is known about such properties from a theoretical perspective. Thus, we fall back to empirical properties.

6.3.2 Deriving the Algorithm

In the following, we derive our line search approach on the full-batch loss by iteratively exploiting empirically found observations of (Mutschler and Zell, 2021) and solving the challenges for a line search on the full-batch loss (see Section 6.3.1).

Given default values are inferred from a detailed hyperparameter analysis (Section 6.4.4)

Observation 1: *Minima of $l_{\mathbb{B},t}$ can be at significantly different points than minima of l_t and can even lead to update steps, which increase $\mathcal{L}$* (Figure 6.2 center, green and red curve).

Derivation Step 1: This consolidates that line searches on a too low mini-batch loss are unpromising. Consequently, we concentrate on a better way to approximate l_t.

Observation 2: *l_t can be approximated with parabolas of positive curvature, whose fitting errors are of less than $0.6 \cdot 10^{-2}$ mean absolute distance* (exemplarily shown in Figure 6.1).

Derivation Step 2: We approximate l_t with a parabola ($l(s)_t \approx a_t s^2 + b_t s + c_t$ with $a_t > 0$). A parabolic approximation needs three measurements of l_t. However, already computing l_t for one s only is computationally unfeasible. Assuming i.i.d sample losses, the standard error of $l_{\mathbb{B},t}(s)$, decreases with $1/\sqrt{|\mathbb{B}|}$. Thus, $l_{\mathbb{B},t}$ -with a reasonable large batch size- is already a good estimator for the full-batch loss parabola. Consequently, we approximate l_t with $l_{\mathbb{B}_a,t}$ by averaging over multiple $l_{\mathbb{B}_i,t}$ measured over multiple inferences. Thus, the approximation batch size $\mathbb{B}_a$, is significantly larger than the, by GPU memory limited, possible batch size $\mathbb{B}_i$. In our experiments, $\mathbb{B}_a$ is usually chosen to be 1280, which is 10 times larger than $\mathbb{B}_i$. In detail, we measure $l_{\mathbb{B}_a,t}$ at the points $s = 0, 0.0001$ and 0.01, then we simply infer the parabola's parameters and the update step to the minimum. These values of s empirically lead to the best and numerically most stable approximations.

Observation 3: *The trend of $s_{min,t}$ of consecutive l_t changes slowly and consecutive*

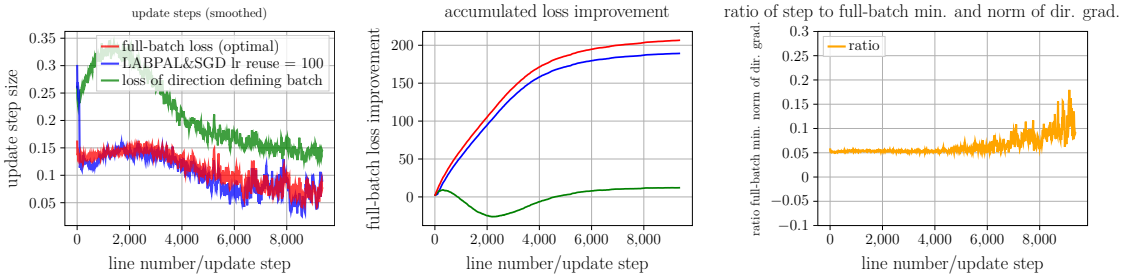


Figure 6.2: Several metrics to compare update step strategies on the full-batch losses along 10,000 lines measured by Mutschler and Zell (2021): 1. update step sizes, 2. accumulated loss improvement per step given as: $l(0) - l(s_{\text{upd}})$ where s_{upd} is the update step of a specific optimizer. This is the locally optimal improvement to the minimum of the full-batch loss along a line. The right plot shows almost proportional behavior between the optimal update step and the negative gradient norm of the direction defining mini-batch loss. The LABPAL&SGD version of our approach performs almost optimal on ground truth data. Results LABPAL&NSGD are almost identical and thus omitted.

l_t *do not change locally significantly.* (Figure 6.2 left, red curve).

Observation 4: $s_{min,t}$ *and the direction defining batch's* $||\boldsymbol{g}_{\mathbb{B}_t,t}||$ *are almost proportional during training.* (Figure 6.2 right).

Derivation Step 3: Using measurements of $l_{\mathbb{B}_a,t}$ to approximate l_t with a parabola is by far too slow to compete against SGD if done for each weight update. By exploiting Observation 3 we can approximate l_t after a constant amount of steps and reuse the measured learning rate λ or update step size s_{upd} for subsequent steps. In this case, λ is a factor multiplied by $\boldsymbol{g}_{\mathbb{B},t}$, whereas s_{upd} is a factor multiplied by $\hat{\boldsymbol{g}}_{\mathbb{B},t}$. Consequently, with λ we perform a step in gradient direction (as SGD also does), whereas, with s_{upd} we perform a step in normalized gradient direction, ignoring the norm of the gradient. Observation 4 allows us to reuse λ. In our experiments, it is sufficient to measure a new λ or s_{upd} every 1000 steps only.

Derivation Step 4: So far, we can approximate l_t efficiently and, thus, overcome the first challenge (see Section 6.3.1). Now, we will overcome the second challenge; approximating the full batch loss gradient for each weight update step:

For this, we revisit (Smith *et al.*, 2018) who approximates the magnitude of random gradient fluctuations, that appear if training with a mini-batch gradient, by the *noise scale* $\nu \in \mathbb{R}$:

$$\nu \approx {}^{(\lambda|\mathbb{D}|)}/_{|\mathbb{B}|}, \tag{6.5}$$

where λ is the learning rate, $|\mathbb{D}|$ the dataset size and $|\mathbb{B}|$ the batch size. If the random gradient fluctuations are reduced, the approximation of the gradient gets better. Since we want to estimate the learning rate automatically, the only tunable parameter to reduce the *noise scale* is the batch size.

Observation 5: *The variance of consecutive* $s_{min,t}$ *is low, however, it increases continuously during training* (Figure 6.2 left, red curve).

Derivation Step 5: It stands to reason that the latter happens because the random gradient fluctuations increase. Consequently, during training, we increase the batch size for weight updates by iteratively sampling a larger batch with multiple inferences. This reduces the variance of consecutive $s_{\text{min},t}$ and lets us reuse estimated the λ or s_{upd} for more steps. After experiencing unusable results with the approach of (De *et al.*, 2016) to determine appropriate batch sizes, we stick to a simple piece-wise constant batch size schedule, doubling the batch size after two and after three-quarters of the training.

Observation 6: *On a global perspective an* s_{upd} *that overestimates* $s_{min,t}$ *optimizes and generalizes better.*

Derivation Step 6: Thus, after estimating λ (or s_{min}) we multiply it with a factor $\alpha \in]1, 2[$:

$$\lambda_{\text{new}} = \alpha\lambda = {}^{\alpha s_{\text{min},t}}/_{||\boldsymbol{g}_{\mathbb{B},t}||} \quad \text{or} \quad s_{\text{upd}} = \alpha s_{\text{min},t}. \tag{6.6}$$

Note that under our parabolic property, the first Wolfe condition w_1, which is

commonly used for line searches, simply relates to α: $w_1 = -\frac{\alpha}{2} + 1$: Let $f : \mathbb{R} \to \mathbb{R}$ be of form $x \mapsto ay^2 + by + c$. We start with the first Wolfe condition (a.k.a. Armijo condition, sufficient decrease condition):

$$\begin{aligned}
f(x_0 + y) &\leq f(x_0) - y\nabla f(x_0) w_1 && \text{in our case } x_0 = 0, w_1 \text{ wolfe constant} \\
f(y) &\leq f(0) + ybw_1 && \\
ay^2 + by + c &\leq c + ybw_1 && \text{use quadratic shape}, \nabla f(x_0) = b \\
ay^2 + by - ybw_1 &\stackrel{!}{=} 0 && \\
\frac{ay^2 + by}{by} &= \frac{ay}{b} + 1 = w_1 && \\
-\frac{\alpha}{2} + 1 &= w_1 && \text{set } y = \alpha\frac{-b}{2a}, \alpha \in [1, 2) \\
-2w_1 + 2 &= \alpha &&
\end{aligned} \tag{6.7}$$

Combining all derivations leads to our line search named *Large-Batch Parabolic Approximation Line Search* (LABPAL), which is given in Algorithm 9. It samples the desired batch size over multiple inferences to perform a close approximation of the full-batch loss and then reuses the estimated learning rate to train with SGD (LABPAL&SGD), or it reuses the update step to train with SGD with a normalized gradient (LABPAL&NSGD). While LABPAL&SGD elaborates Observation 4, LABPAL&NSGD completely ignores information from $||\boldsymbol{g}_{\mathbb{B},t}||$.

Algorithm 9 LABPAL&SGD. Simplified conceptional pseudo-code of our proposed algorithm, which estimates update steps on a parabolic approximation of the full-batch loss. See the published source code for technical details. Default values are given in parentheses. For LABPAL&NSGD SGD is replaced with Normalized SGD (NSGD), and the update step is measured instead of the learning rate. PyTorch code is provided at `https://github.com/cogsys-tuebingen/LABPAL`.

Input: Hyperparameters:
- initial parameters θ_0
- approximation batch size $|\mathbb{B}_a|$ (1280)
- inference batch size $|\mathbb{B}_i|$ (128)
- SGD steps n_{SGD} (1000), # or NSGD steps
- step size adaptation $\alpha > 1$ (1.8)
- training steps t_{max}(150000)
- batch size schedule $k(t) = \begin{cases} 1, & \text{if } t \leq \lfloor t_{max} \cdot 0.5 \rfloor \\ 2, & \text{elif } t \leq \lfloor t_{max} \cdot 0.75 \rfloor \\ 4, & \text{elif } t > \lfloor t_{max} \cdot 0.75 \rfloor \end{cases}$

```
# Variables have global scope
sampledBatchSize ← 0
performedSGDsteps ← 0
learningRate ← 0
θ ← θ_0
state ← 'line search'
direction ← current batch gradient
t ← 0
while t < t_max do
    if state is 'line search' then
        PERFORM_LINE_SEARCH_STEP()
    end if
    if state is 'SGDTraining' then
        PERFROM_LARGE_BATCH_SGD_STEP()
    end if
end while
return θ

procedure PERFORM_LINE_SEARCH_STEP()
    if sampledBatchSize < |B_a| then
        update estimate L̂ of L with
        over multiple inferences sampled L_{B_t,t} with
        |B_t| = k(t) · |B_i|
        increase sampledBatchSize by |B_t| and t by k(t)
    else
        learningRate ← perform parabolic approximation with 3 values of L̂ along the search direction and
        estimate the learning rate.
        learningRate ← learningRate ·α
        set sampledBatchSize and performedSGDsteps to 0
        state ← 'SGDTraining'
    end if
end procedure

procedure PERFROM_LARGE_BATCH_SGD_STEP()
    if performedSGDsteps < n_SGD then
        θ ← perform SGD update with learningRate and over multiple inferences sampled L_{B_t,t}
        with |B_t| = k(t) · |B_i|)
        increase t by k(t)
        increase performedSGDsteps by 1
    else
        direction ← current batch gradient
        state ← 'line search'
    end if
end procedure
```

6.4 Empirical Analysis

Our two approaches are compared against other line search methods across several datasets and networks in the following. **To reasonably compare different line search methods, we define a *step* as the sampling of a new batch.** Consequently, the steps/batches that LABPAL takes to estimate a new learning rate/step size are considered, and optimization processes are compared on their data efficiency.
Note that the base ideas of the introduced line search approaches can be applied upon any direction giving technique such as momentum, Adagrad (Duchi *et al.*, 2011) or ADAM (Kingma and Ba, 2015). Results are averaged over 3 runs.

6.4.1 Performance Analysis on ground truth full-batch Loss and Proof of Concept

To analyze how well our approach approximates the full-batch loss along lines, we extend the experiments of Chapter 5 by LABPAL. In Chapter 5 we measured the full-batch loss along lines in SGD update step directions of a training process; thus, this data provides ground truth to test how well the approach approximates the full-batch loss. In this scenario, LABPAL&SGD uses the full-batch size to estimate the learning rate and reuses it for 100 steps. No update step adaptation is applied. Figure 6.2 shows that LABPAL&SGD fits the update step sizes to the minimum of the full-batch loss and performs near-optimal local improvement. The same holds for LABPAL&NSGD.
We now test how our approaches perform in a scenario for which we can assure that the used empirical observations hold. Therefore, we consider the optimization problem of Chapter 5 from which all empirical observations were inferred, which is training a ResNet20 on 8% of CIFAR10. $\mathbb{B}_a$ of 1280 is used for both approaches. Learning rates are reused for 100 steps, and $\alpha = 1.8$ is considered. The batch size is doubled after 5000 and 7500 steps. For SGD λ is halved after the same steps. A grid search for the best λ is performed. Figure 6.3 shows that LABPAL&NSGD with update step adaptation outperforms SGD, even though 9% of the training steps are used to estimate new update step sizes. This shows that using the estimated learning rates and step sizes leads to better performance than keeping them constant or decaying them with a piece-wise schedule. Interestingly huge λ's of up to $80,000$ are estimated, whereas s_{upd}'s are decreasing. LABPAL&SGD shows similar performance as SGD; however, it seems beneficial to ignore gradient size information as the better performance of LABPAL&NSGD shows.

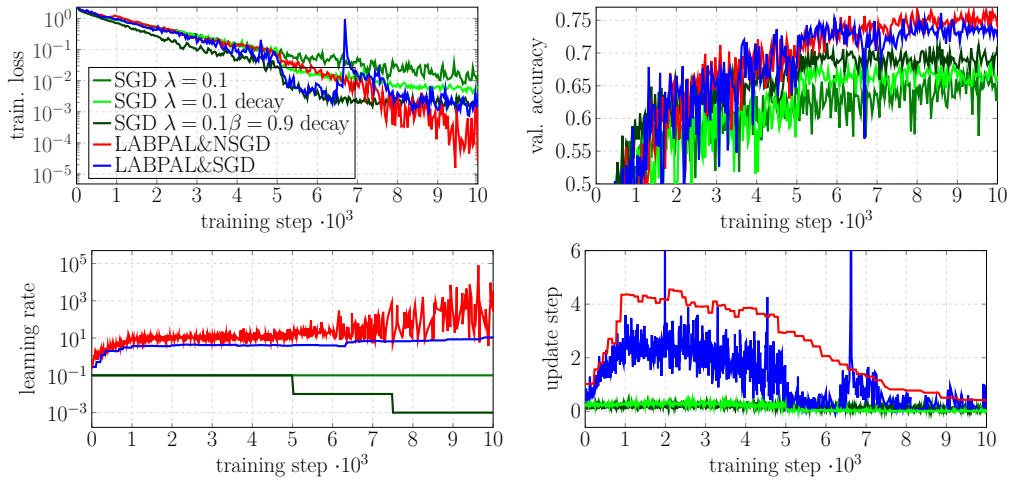


Figure 6.3: Training process on the problem of which the empirical observations were inferred (ResNet-20 trained on 8% of CIFAR-10 with SGD). LABPAL&NSGD and LABPAL&SGD outperform SGD. Interestingly LABPAL&NSGD estimated huge λ's, whereas s_{upd}'s are decreasing

6.4.2 Performance Comparison to SGD and to other Line Search Approaches

We compare the SGD and NSGD variants of our approach against PLS (Mahsereci and Hennig, 2015), GOLSI (Kafka and Wilke, 2019), PAL (Mutschler and Zell, 2020a), SLS (Vaswani *et al.*, 2019) and SGD with momentum (Robbins and Monro, 1951). The latter is a commonly used optimizer for deep learning problems and can be reinterpreted as a parabolic approximation line search on mini-batch losses (Mutschler and Zell, 2021). PLS is of interest since it approximates the full-batch loss to perform line searches. PAL, GOLSI, SLS on the other hand are line searches optimizing on mini-batch losses directly. For SGD with momentum, a piece-wise constant learning rate schedule divides the learning rate after two and again after three-quarters of the training by a factor of 10.

Comparison is done across several datasets and models. Specifically, we compare ResNet-20 (He *et al.*, 2016), DenseNet-40 (Huang *et al.*, 2017), and MobileNetV2 (Sandler *et al.*, 2018) trained on CIFAR-10, CIFAR-100 (Krizhevsky *et al.*, 2009), and SVHN (Netzer *et al.*, 2011). We concentrate on classification problems since the empirical observations are inferred from a classification task and since those problems are usually considered as benchmarks for new optimization approaches.

For each optimizer, we perform a comprehensive hyperparameter grid search on ResNet-20 trained on CIFAR-10 (see Appendix C.4.1). The best performing hyperparameters on the validation set are then reused for all other experiments. The latter is done to check the robustness of the optimizer by handling all other datasets as if they were unknown. This is usually the case in practice, since for

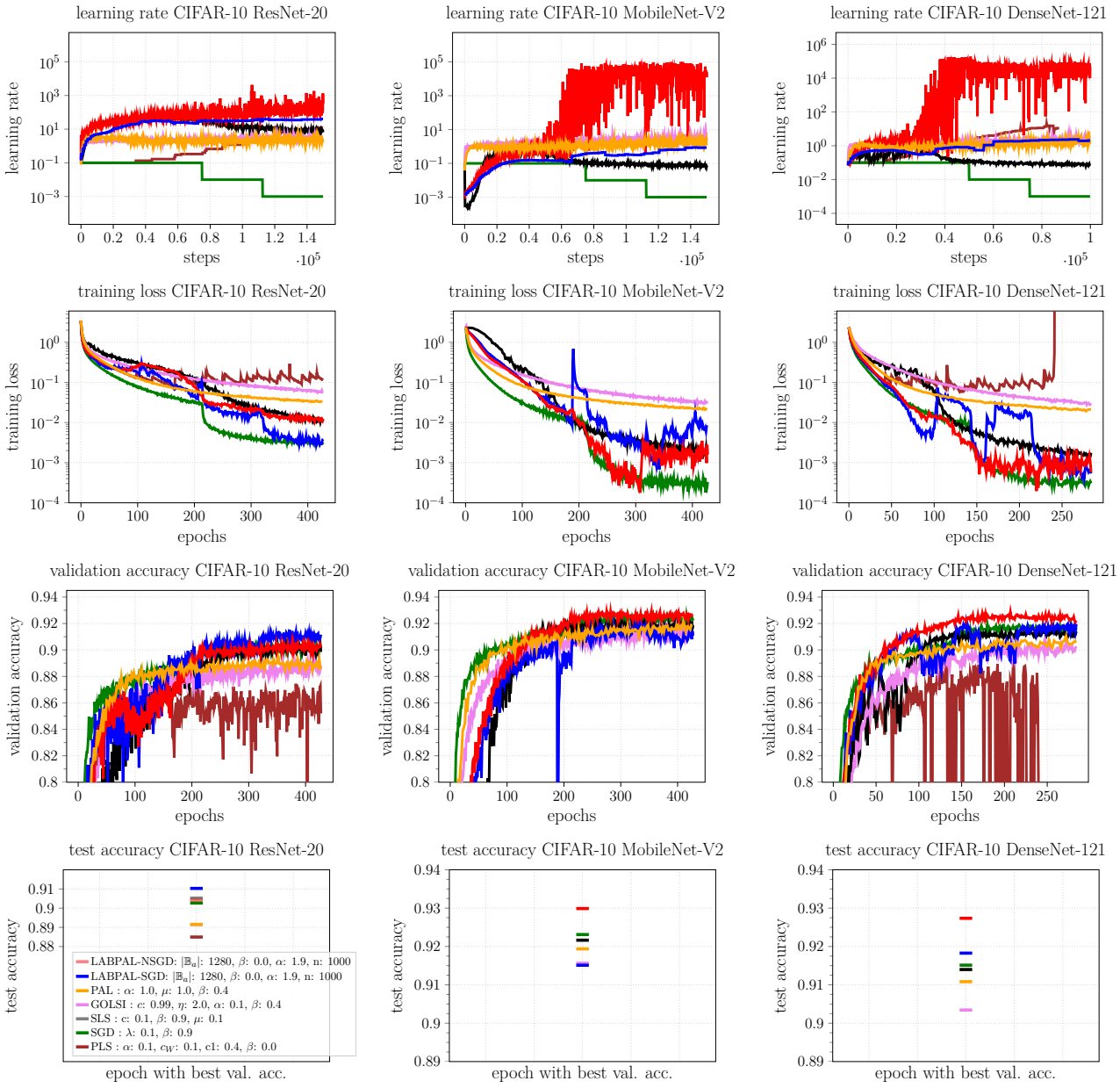


Figure 6.4: Performance comparison on **CIFAR-10** of our approach LABPAL in the SGD and NSGD variants against several line searches and SGD with momentum. Optimal hyperparameters are found with an elaborate grid search. Our approaches challenge and sometimes outperform the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics.

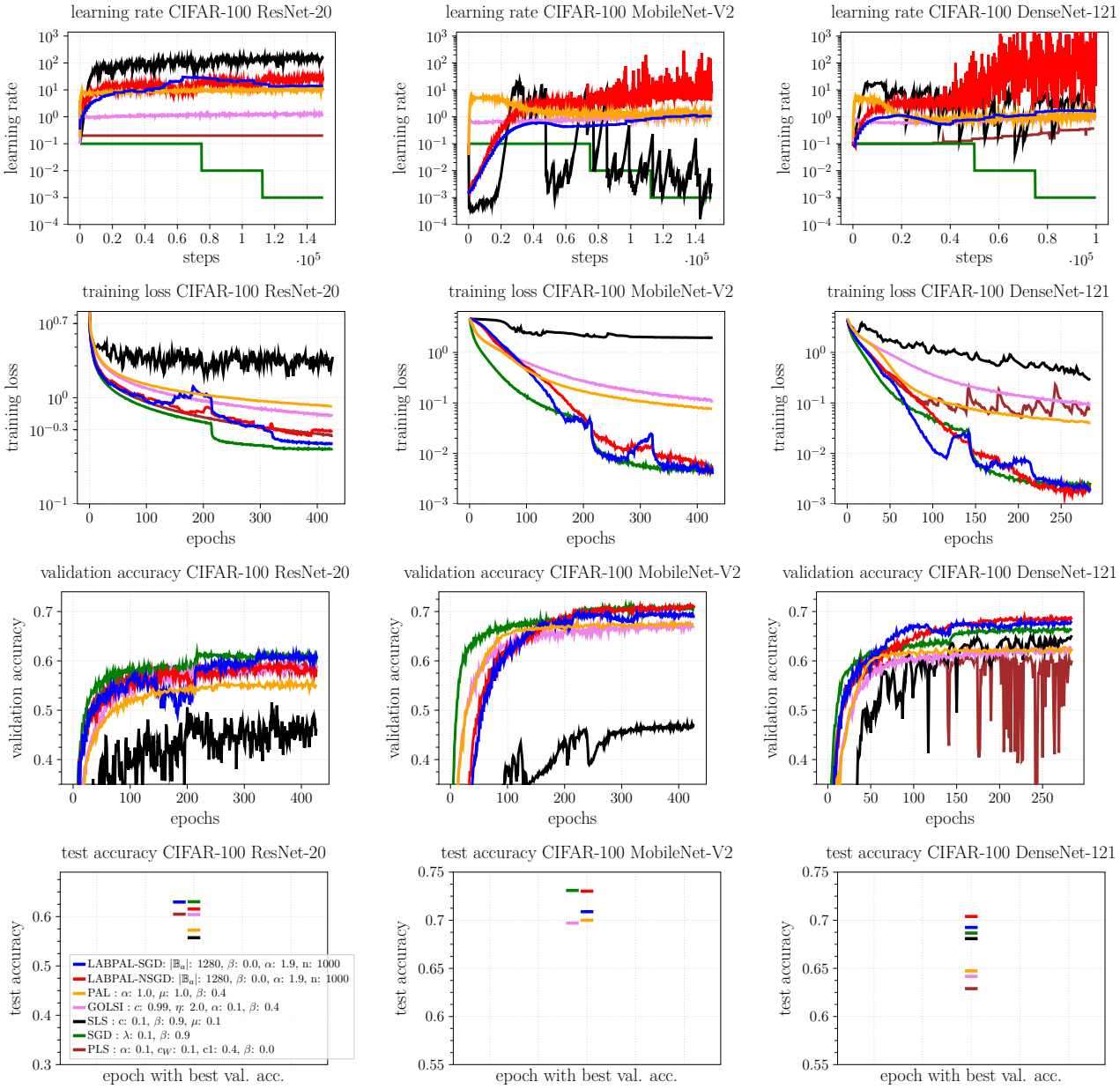


Figure 6.5: Performance comparison on **CIFAR-100** of our approach LABPAL in the SGD and NSGD variants against several line searches and SGD with Momentum. Optimal hyperparameters for CIFAR-10 ResNet-20 found with a grid search are reused (Appendix C.4.1). Here, our approaches surpass the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics. Results for CIFAR-10 and SVHN are given in appendix Figures 6.4, and C.1. The batch-size used is 128. Due to a too high memory consumption we could not run PLS on MobileNet-V2.

specific datasets no optimal hyperparameters are known beforehand. Our aim here is to show that satisfactory results can be achieved on new problems without any fine-tuning needed. Further experimental details are found in Appendix C.4.

Figure 6.5 as well as Figures 6.4, C.1 show that both LABPAL approaches outperform PLS, GOLSI and PAL considering training loss, validation accuracy, and test accuracy. LABPAL&NSGD tends to perform more robust and better than LABPAL&SGD. LABPAL&NSGD is on pair with SGD with momentum and challenges SLS on validation and test accuracy. The important result is that hyperparameter tuning for LABPAL is not needed to achieve good results across several models and datasets. However, this is also true for pure SGD, which suggests that the simple rule of performing a step size proportional to the norm of the momentum term is sufficient to implement a well-performing line search. This also strengthens the observation of (Mutschler and Zell, 2021), which states that SGD, with the correct learning rate, is already performing an almost optimal line search.

The derived learning rate schedules of the LABPAL approaches are significantly different from a piece-wise constant schedule (Figure 6.5, 6.4, C.1 first row). Interestingly they show a strong *warm up (increasing) phase* at the beginning of the training followed by a rather *constant phase* which can show minor learning rate changes with an increasing trend. The NSGD variant sometimes shows a second increasing phase, when the batch size is changed. The *warm up* phase is often seen

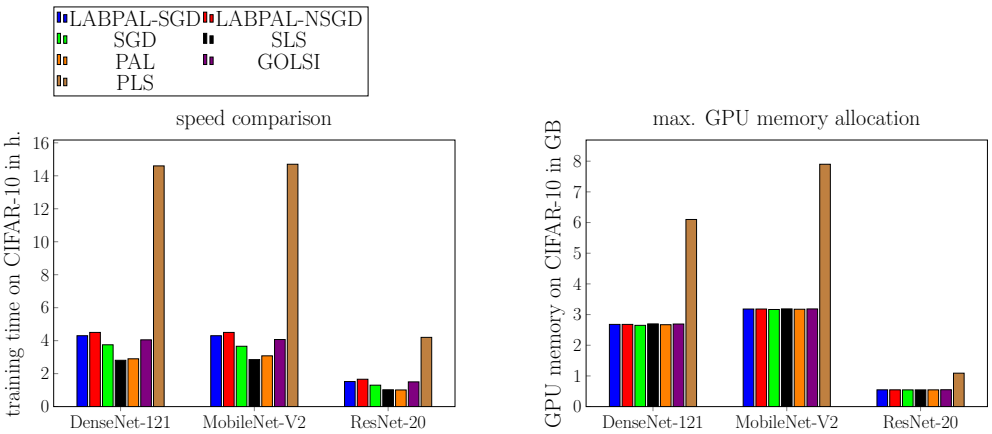


Figure 6.6: Left: Training time comparison on CIFAR-10. SGD, SLS, and PAL show similar training times. GOLSI, and both variants of LABPAL are slightly slower (up to 19.6%). However, a slightly longer training time is acceptable if less time has to be spent in hyperparameter tuning. PLS is significantly slower. Note that in comparison to SGD, the implementations of the other optimizers are not optimized at CUDA level. **Right:** Maximum allocated memory comparison on CIFAR-10. Except for PLS all approaches need approximately the same amount of memory.

in sophisticated learning rate schedules for SGD; however, usually combined with a *cool down* phase. The latter is not apparent for LABPAL since we increase the batch size. LABPAL&NSGD indirectly uses learning rates of up to 10^6 but still trains robustly. Of further interest is that all line search approaches do not decrease the learning rate at the end of the training as significantly as SGD, which hinders the line searches to converge.

A comparison of training speed and memory consumption is given in Figure 6.6. In short, LABPAL has identical GPU memory consumption as SGD and is on average only 19.6% slower. However, for SGD usually a grid search is needed to find a good λ, which makes LABPAL considerably cheaper.

6.4.3 Adaptation to Varying Gradient Noise

Recent literature, e.g., (Mutschler and Zell, 2020a), (Vaswani *et al.*, 2019), (Kafka and Wilke, 2019) shows that line searches work with a relatively large batch size of 128 and a training set size of approximately 40000 on CIFAR-10. *However, a major, yet not comprehensively considered problem is that line searches operating on the mini-batch loss vary their behavior with another batch- and training set size leading to varying gradient noise.* E.g., Figure 6.7 shows that training with PAL, GOLSI or PLS and a batch size of 8 on CIFAR-10 does not work at all. The reason is that the gradient noise induced by mini-batches, and with it the difference between the full-batch loss and the mini-batch loss, increases. However, we can adapt LABPAL to work in these scenarios by holding the *noise scale* approximately at the same value as it was for the hyperparameter grid search. As the learning rate is inferred directly, the batch size has to be adapted. Based on the linear approximation of the *noise scale* (see Equation 6.5), we directly estimate a noise adaptation factor $\epsilon \in \mathbb{R}$ to adapt LABPAL's hyperparameter:

$$\epsilon := \frac{\nu_{\text{new}}}{\nu_{\text{ori}}} = \frac{|\mathbb{B}_{\text{ori}}|}{|\mathbb{B}_{\text{new}}|} \frac{|\mathbb{D}_{\text{new}}|}{|\mathbb{D}_{\text{ori}}|} = \frac{128}{|\mathbb{B}_{\text{new}}|} \frac{|\mathbb{D}_{\text{new}}|}{40,000}. \tag{6.8}$$

The original batch size $|\mathbb{B}_{\text{ori}}|$ and the original dataset size $|\mathbb{D}_{\text{ori}}|$ originate from our search for best-performing hyperparameters on CIFAR-10 with a training set size of 40,000, a batch size of 128, and 150,000 training steps. We set the number of training steps to $150,000\epsilon$ and multiply the batch sizes in the batch size schedule k by ϵ. This rule makes the approach fully parameter free in practice (at least for the image classification scenario), since hyperparameters do not have to be adapted across models, batch sizes and datasets. Figure 6.7 shows a performance comparison over different batch sizes. Hyperparameters are not changed. By changing the batch size the noise adaptation factor of the LABPAL approaches gets adapted, which lets them still perform well with low batch sizes since they iteratively sample larger batch sizes over multiple inferences. The performance of PLS, PAL and GOLSI

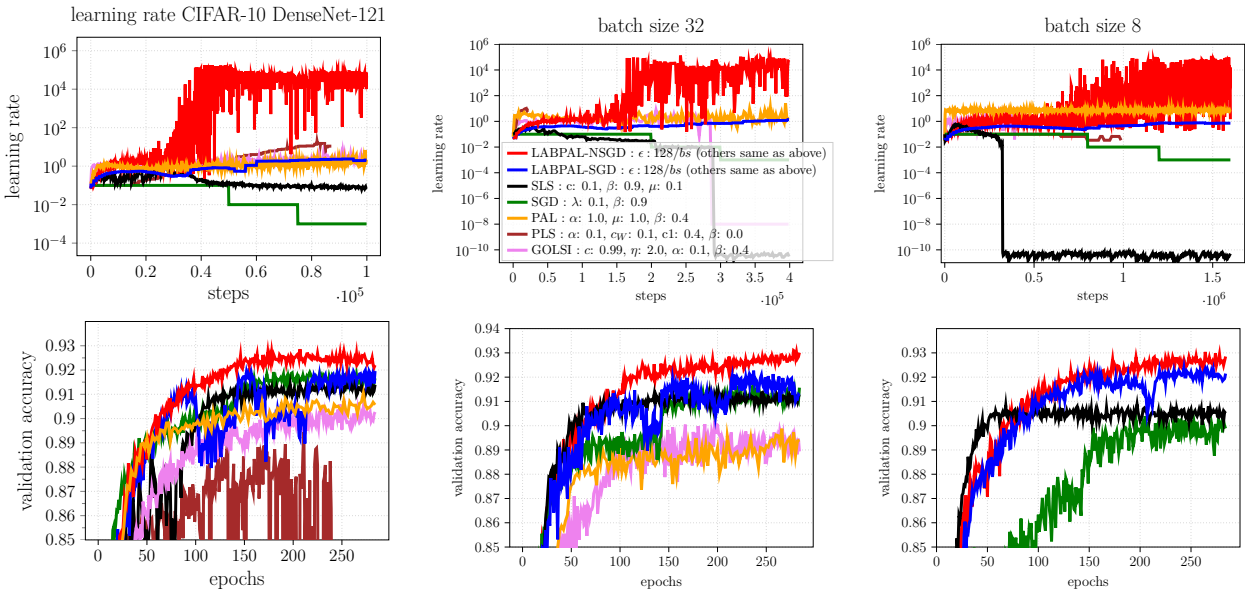


Figure 6.7: Performance comparison of a **DenseNet-121**, trained on **CIFAR-10** for **batch sizes 128, 32 and 8**. Hyperparameters are not changed. (For the evaluation on ResNet-20 and MobileNet-V2 see Appendix Figure C.2 & C.3). PLS curves are incomplete since training failed. Training steps are increased by a factor of 4 and 16 for batch size 32 and 8, respectively.

decreases with lower batch size. SLS's performance stays similar but its learning rate schedule degenerates. For the evaluation on ResNet-20 and MobileNet-V2 see Appendix Figure C.2 & C.3. We note that this batch size adaptation approach to keep the noise scale on a similar level could also be applied to all other line searches, however this will exceed the scope of this work.

6.4.4 Hyperparameter Sensitivity Analysis

We performed a detailed hyperparameter sensitivity analysis for LABPAL&SGD and LABPAL&NSGD. To keep the calculation cost feasible, we investigated the influence of each hyperparameter, keeping all other hyperparameters fixed to the default values (see Algorithm 9). Figure 6.8 and 6.9 show the following characteristics: Estimating new s_{upd} or λ with $\mathbb{B}_a$ smaller than 640 decreases the performance since l_t is not fitted well enough (row 1). The performance also decreases if reusing the λ (or s_{upd}) for more update steps (row 2), and if using a step size adaptation α of less than 1.8 (row 3, except for ResNet). This shows that optimizing for the locally optimal minimum in line direction only is not beneficial. From a global perspective, a slight decrease of the loss by performing steps to the other side of the parabola shows more promise. Interestingly, even using α larger than two still leads to good results. (Mutschler and Zell, 2021) showed that the loss valley in line direction becomes wider during training.

This might be a reason why these update steps, which should actually increase

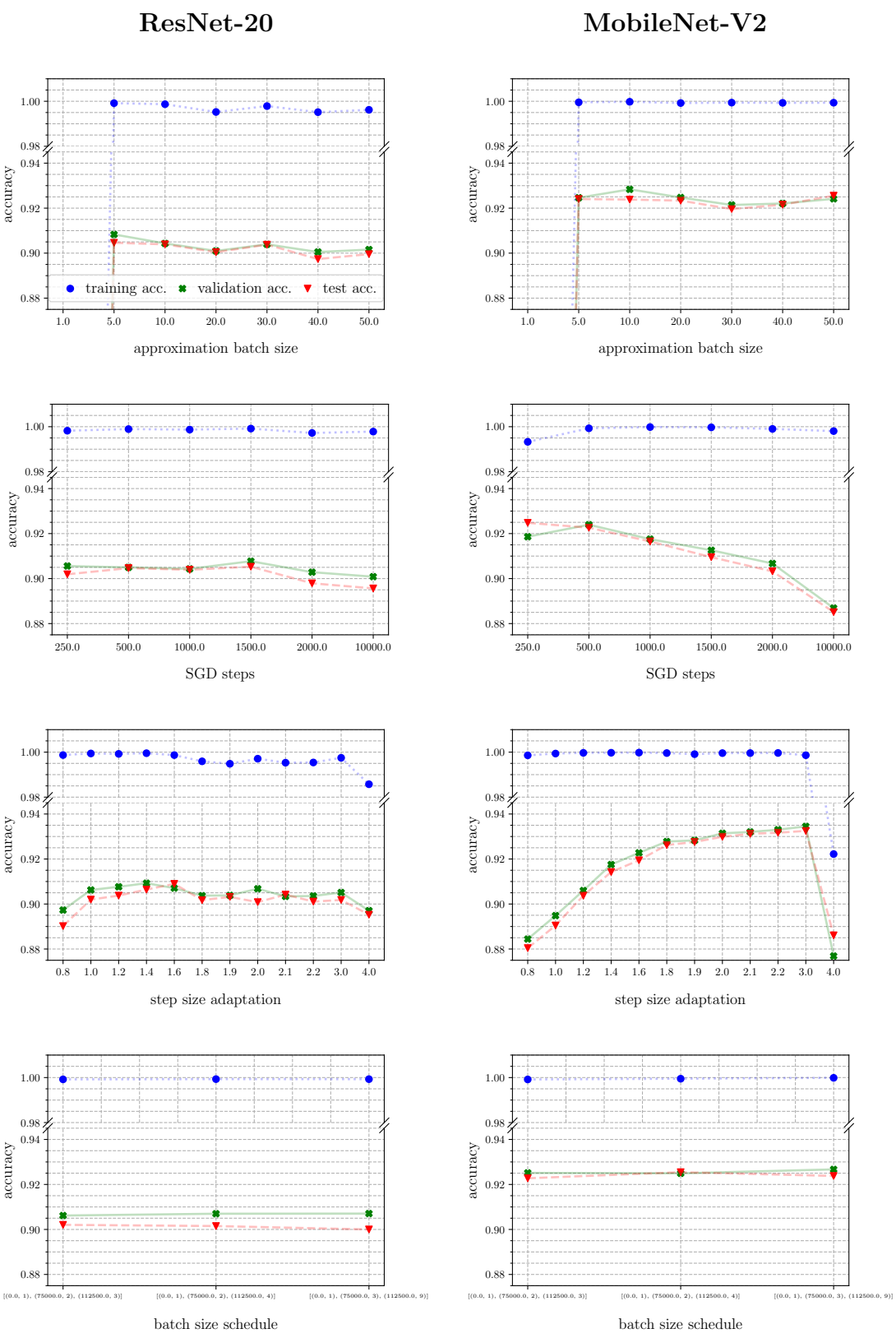


Figure continues on next page.

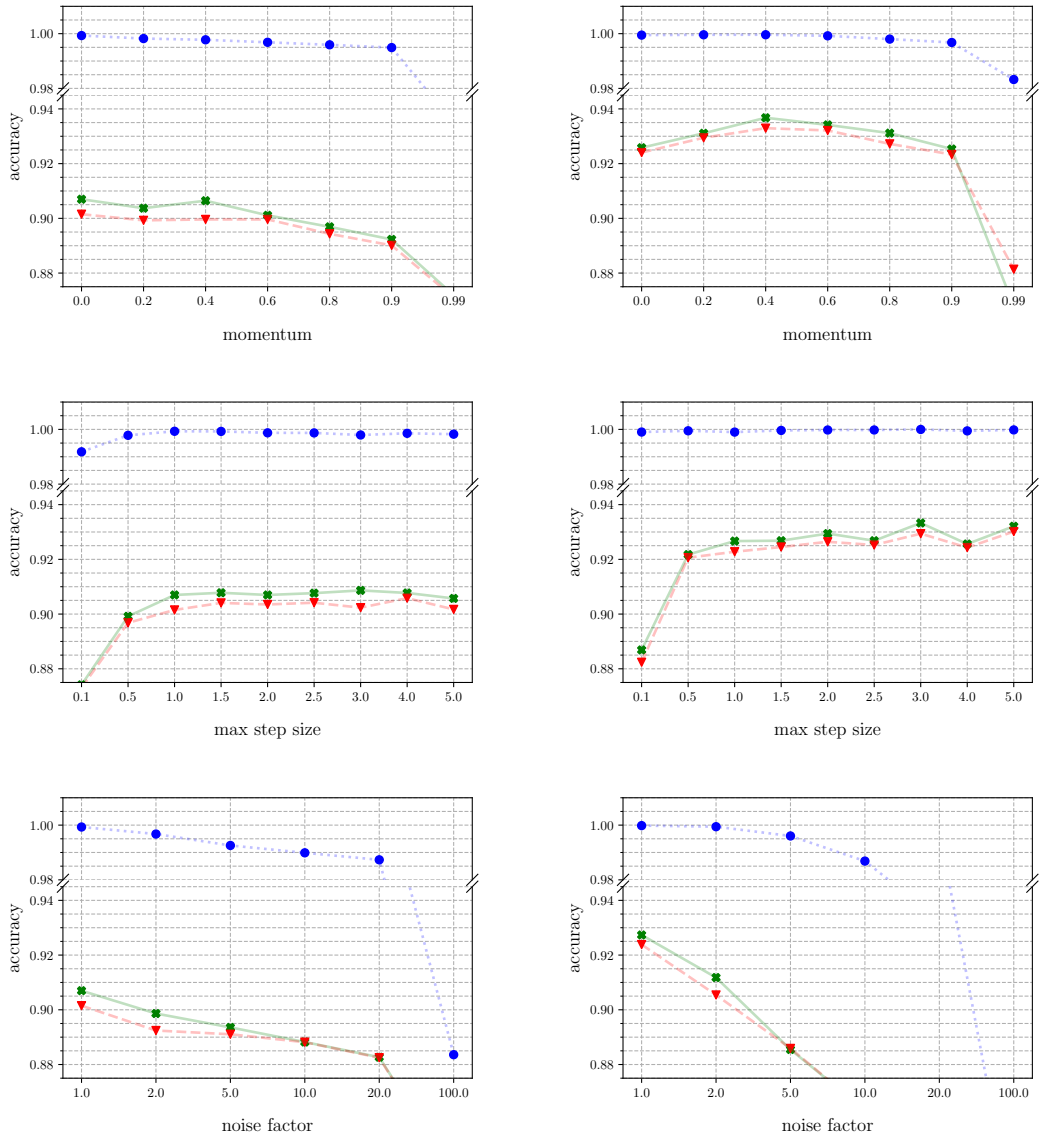


Figure 6.8: Sensitivity analysis of **LABPAL&SGD**'s parameters. The default parameters are: approximation batch size $\mathbb{B}_a = 1280$, SGD steps $s = 1000$, step size adaptation $\alpha = 1.8$, batch size schedule $k =$ (0:1, 75000:2, 112500:4), momentum $\beta = 0$, maximal step size $= 1.0$, noise-factor $\epsilon = 1$. For the approximation batch size (row one) the factor 128 is multiplied on the x axis. Results on DenseNet-121 are given in Appendix C.4.2.

the loss, work. Using a maximal step size of less than 1.5 (row 7) and increasing the noise adaptation factor ϵ (row 9) while keeping the batch size constant also decreases the performance. The latter indicates that the inherent noise of SGD is essential for optimization. Further, we conclude that a momentum factor between 0.4 and 0.6 increases the performance for both LABPAL approaches (row 5).

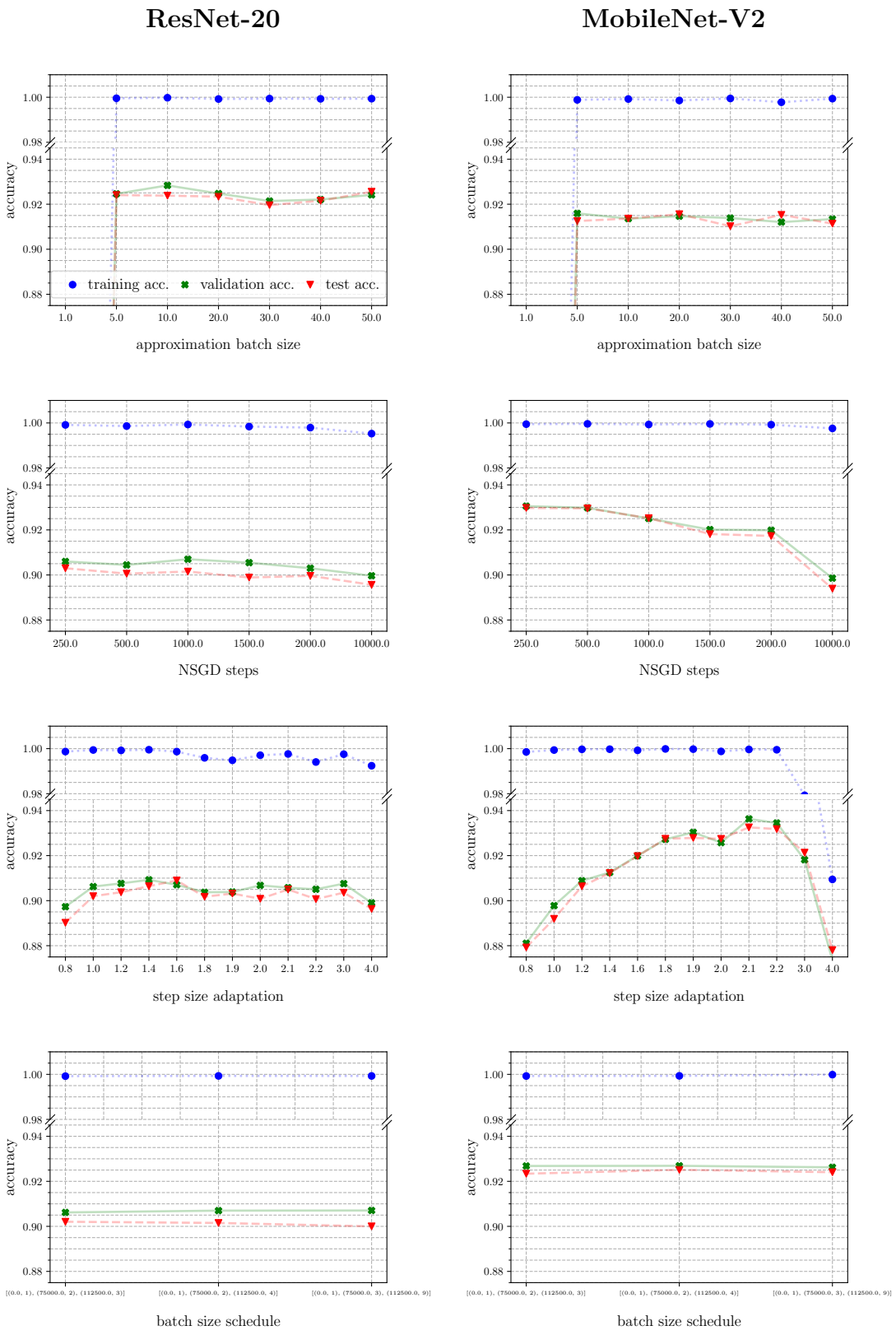


Figure continues on next page.

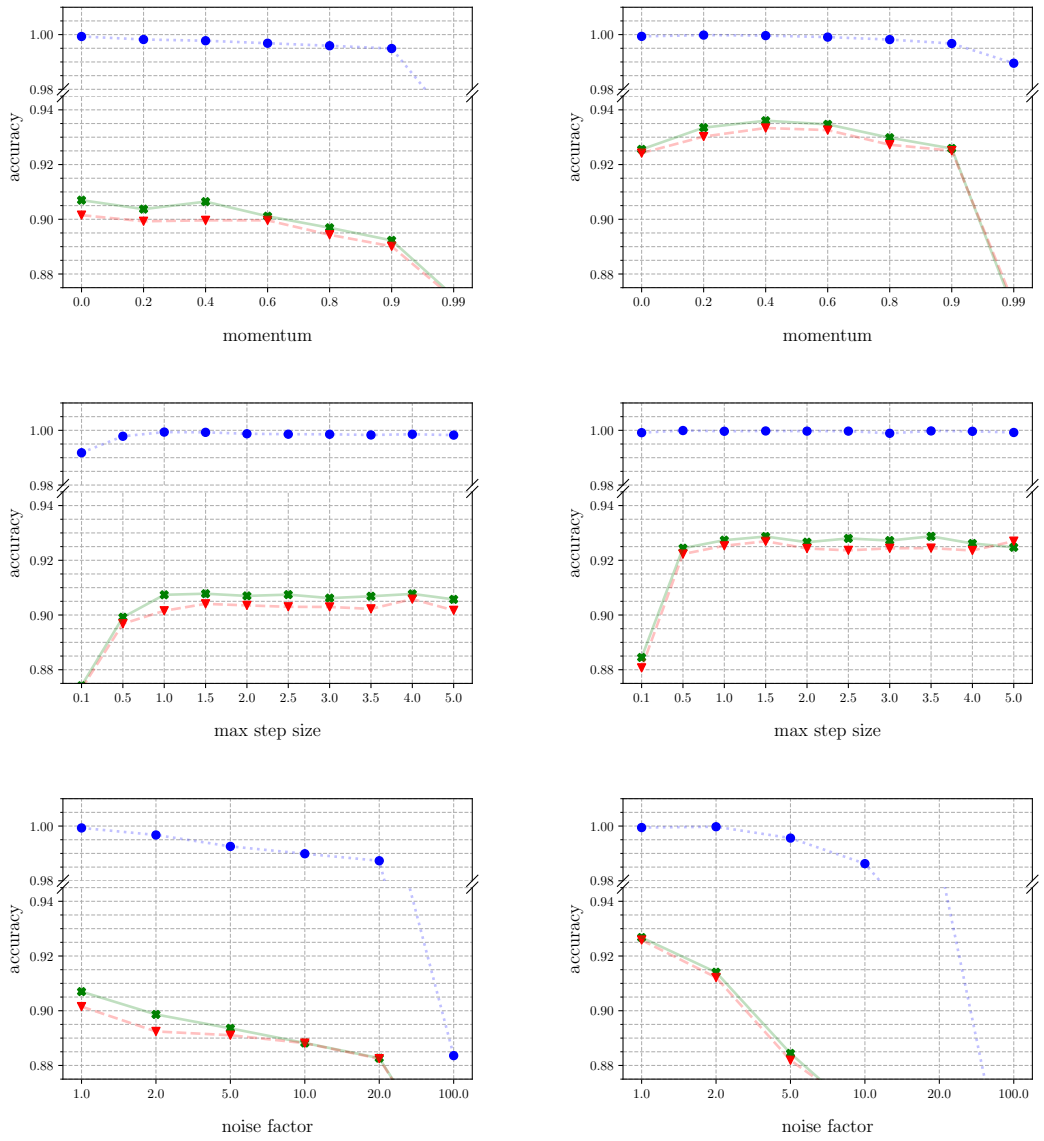


Figure 6.9: Sensitivity analysis of parameters of **LABPAL&NSGD**. The default parameters are: approximation batch size $\mathbb{B}_a = 1280$, SGD steps $s = 1000$, step size adaptation $\alpha = 1.8$, batch size schedule $k =$ (0:1, 75000:2, 112500:4), momentum $\beta = 0$, maximal step size $= 1.0$, noise-factor $\epsilon = 1$. For the approximation batch size (row one) the factor 128 is multiplied on the x axis. Results on DenseNet-121 are given in Appendix C.4.2.

6.5 Limitations

Our approach can only work if the empirically found properties we rely on are apparent, or are still a well enough approximation. In Section 6.4.2 we showed that this is valid for classification tasks. In additional sample experiments, we observed that our approach also works on regression tasks using the square loss. However, it tends to fail if different kinds of losses from significantly different heads of a model are added, as it is often the case for object detection and object segmentation.

A theoretical analysis is absent, since the optimization field still does not know the reason for the local parabolic behavior of l_t, and consequently, what an appropriate function space to consider for convergence is.

6.6 Discussion & Outlook

This work introduced a robust line search approach for deep learning problems based upon empirically found properties of the full-batch loss. Our approach estimates learning rates well across models, datasets, and batch sizes. It mostly surpasses other line search approaches and challenges SGD with momentum tuned with a piece-wise constant learning rate schedule. We are the first to analyze and adapt line searches to varying gradient noise. Furthermore, we show that mini-batch gradient norm information is not necessary for training. In the future, we will analyze the causes for the local parabolic behavior of the full-batch loss along lines to provide further insights about DNN loss landscapes and, in particular, to understand when specific optimization approaches work.

Chapter 7

Overall Conclusion

7.1 Summary

With this dissertation, we have taken the sub-field of line searches in deep learning a step forward. In particular, we improved the understanding of the stochastic loss landscape for deep learning image classification tasks and developed line search approaches that rival or outperform previous approaches. We did this by following an empirical perspective, rarely found in the more theoretical field of optimization for deep learning. Thus, another general contribution of this work is that it justifies and demonstrates the importance of empirical work in this rather theoretical field.

In Chapter 4 we showed -based on (Mutschler and Zell, 2020a)- that parabolic approximation line searches, which are well known in the non-stochastic setting (see (Nocedal and Wright, 2006)), are applicable in typical deep learning classification scenarios. Specifically, we demonstrated with a comprehensive analysis of mini-batch losses along lines in negative gradient direction that mini-batch losses along such lines behave locally parabolic. Further, we have shown that PAL competes with other line search approaches designed for the stochastic scenario and proved its convergence on quadratics. Especially in the interpolation scenario, PAL excels.

In Chapter 5 -based on (Mutschler and Zell, 2021)- we introduced a comprehensive analysis of the full-batch loss along lines in mini-batch gradient direction, leading to several insights that explain the shape of the loss-landscape and the performance of optimizers on a more comprehensive level. The key observations here are:
(1) With a suitable learning rate, SGD already performs a near-exact line search, making it for other approaches hard to compete against SGD.
(2) The effect of halving the learning rate is almost identical to doubling the batch size and can be simply explained by our measured data.
(3) Update steps to the minimum of the full-batch loss along lines change slowly, which allows reusing estimated update steps over longer periods.
(4) Taking an update step that overshoots the minimum, resulting in a locally lower loss decrease, leads to better global optimization performance.

In Chapter 6 -based on (Mutschler *et al.*, 2021)- we exploited the observations

derived in Chapter 5 to design a line search approach that outperforms previous line search approaches and performs on par with SGD with momentum. In addition, it is the first line search in the stochastic scenario that performs well, even when training with small batch sizes such as 32 or 8.

7.2 Outlook

This work focused on optimization for image classification tasks, which are standard benchmarking tasks mostly considered in optimization for deep learning (see (Kingma and Ba, 2015; Duchi *et al.*, 2011; Vaswani *et al.*, 2019; Berrada *et al.*, 2020)). However, little is known about the extent to which our observations hold and our algorithms perform in other deep learning sub-fields such as image synthesis, natural language processing, or reinforcement learning. Therefore, further empirical studies should be conducted to validate whether our observations generalize and optimizers work on such sub-fields. And if not, whether other observations can be found that lead to better-performing algorithms and a more comprehensive understanding of the loss landscape in such sub-fields.

Of further interest is how line searches perform in combination with other optimization methods. In particular, line searches applied on the directions given by sophisticated adaptive methods such as ADAM (Kingma and Ba, 2015), AdaGrad (Duchi *et al.*, 2011), RMSProp (Tieleman and Hinton, 2012), or Nero (Liu *et al.*, 2021) should be considered, as these approaches already show good performance in many domains.

The empirical study we presented in Chapter 5 was limited to SGD, SGD with momentum (Robbins and Monro, 1951), and PAL (Mutschler and Zell, 2020a). It is of great interest to analyze how the loss landscape behaves in other directions, for example, directions used by the adaptive methods mentioned above, and how these methods behave along lines in such directions.

All in all, we hope that our empirical findings will pave the way for further theoretical and also empirical insights.

Finally, we hope that in the future, the field will pay more attention to empirical studies that lead to a more comprehensive understanding of why and how optimization methods work in practice. This comprehensive understanding is of great importance because it guides optimizers to unbiased, well-generalizing, and well-performing minimizers.

Appendix A

Parabolic Approximation Line Search

A.1 Further Line Plots

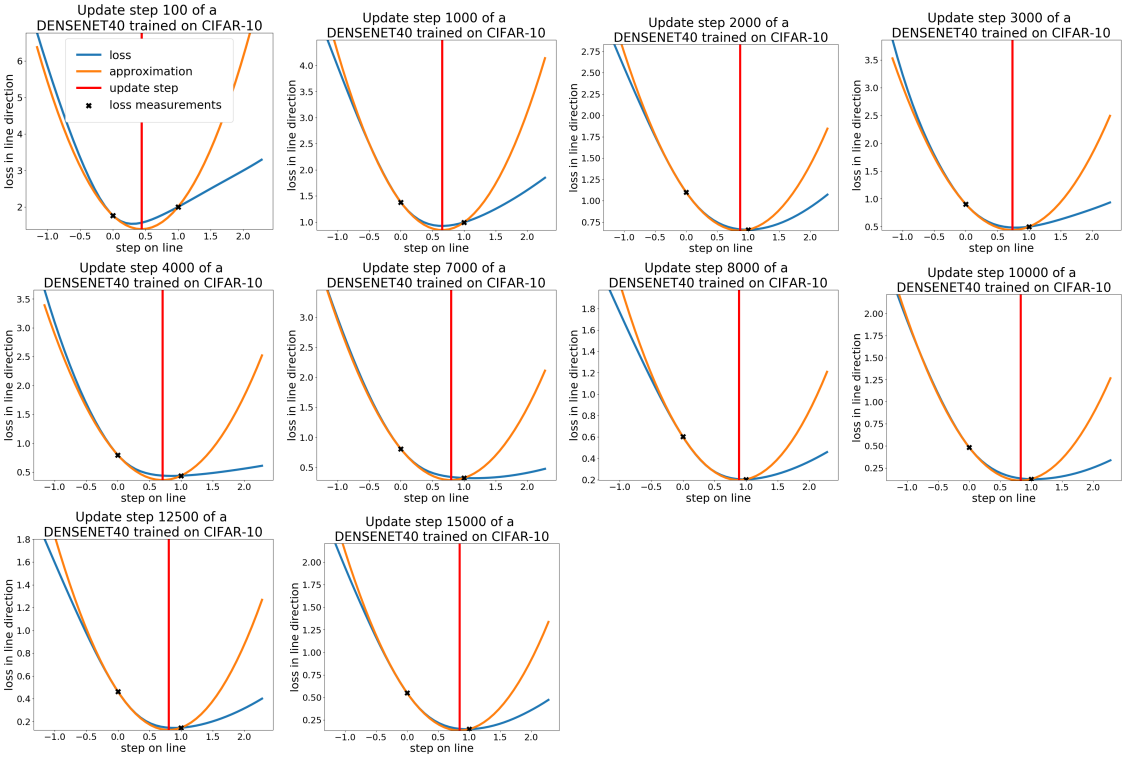


Figure A.1: DenseNet40 mini-batch losses along in negative gradient direction (blue) combined with our parabolic approximation (orange) and the position of the minimum (red). The unit of the horizontal axis is the change of θ.

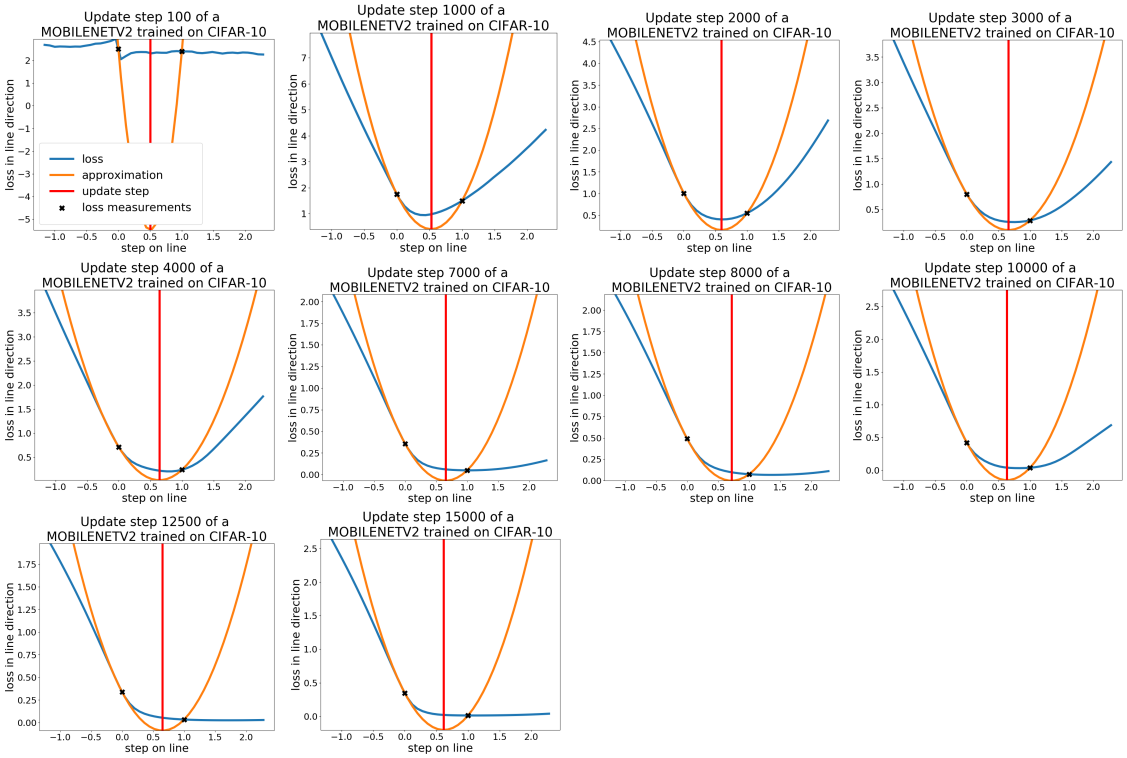


Figure A.2: Mini-batch losses along lines of **MobileNetV2**. For explanations see Figure A.1. During training the parabolic approximation fits less accurately on the right hand side and during the first 150 steps it does not fit at all, however, the minimum of the parabola is still a good estimator for a low loss value on the line.

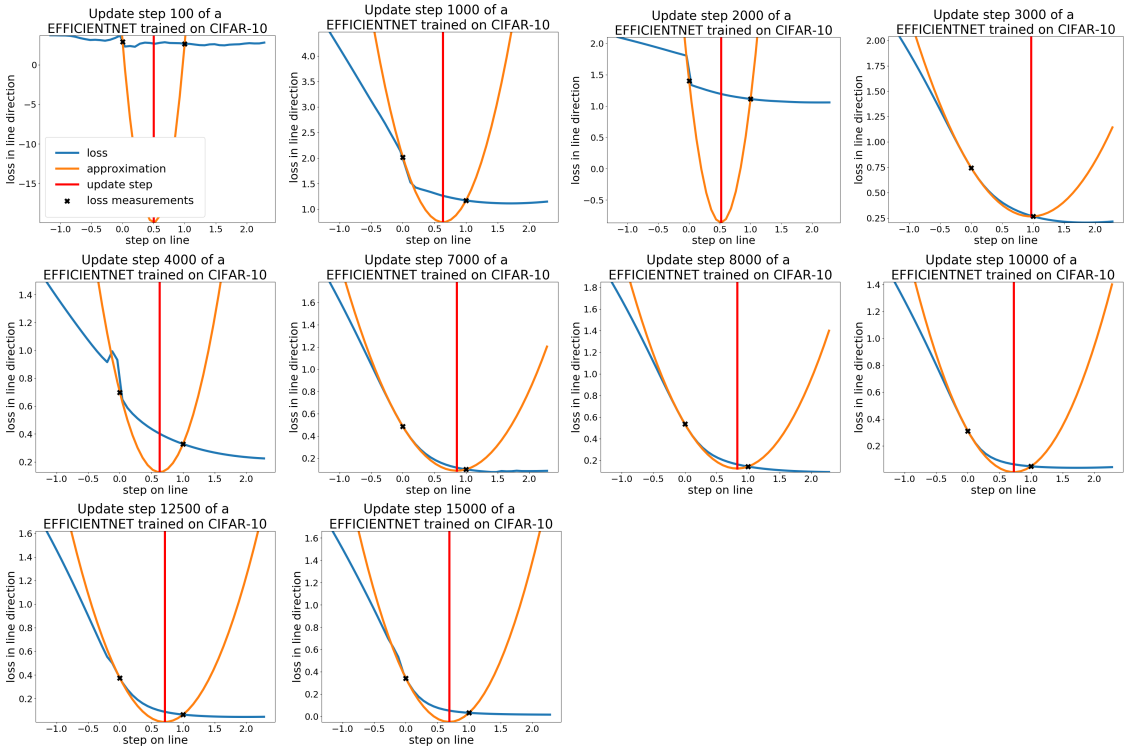


Figure A.3: Mini-batch losses along lines of **EfficientNet**. For explanations see Figure A.1. The parabolic approximation fits only on the left hand side and during the first 150 steps it does not fit at all, however, the minimum of the parabola is still a good estimator for a low loss value on the line.

A.1.1 Proofs

Lemma 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a k-times continuously differentiable function. Furthermore, assume there exists $a, b, c \in \mathbb{R}$ with $a > 0$, such that $f(\boldsymbol{p} + \boldsymbol{d}s) = as^2 + bs + c$ for all $s \in \mathbb{R}$. Then there exist $z \in \mathbb{R}, \boldsymbol{r} \in \mathbb{R}^n$ and a positive definite Matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ such that $f(\boldsymbol{x}) = c + \boldsymbol{r}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^n$.*

Proof.

$$\begin{aligned} g(\boldsymbol{x}) = u + \boldsymbol{v}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{x} \text{ for some } u \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^n \text{ and } \boldsymbol{W} \in \mathbb{R}^{n \times n} \\ \Leftrightarrow \forall \boldsymbol{p}, \boldsymbol{d} \in \mathbb{R}^n \wedge ||\boldsymbol{d}|| = 1 : \sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} \frac{\partial^3 g(\boldsymbol{p})}{\partial x_j, \partial x_k, \partial x_l} d_j d_k d_l = 0 \end{aligned} \tag{A.1}$$

$\Rightarrow$ holds since we have a polynomial of degree 2 and its third derivative is always a $\mathbf{0}$ tensor.
$\Leftarrow$ holds since the reminder of the quadratic Taylor expansion is always 0.
In our case the right part is 0 since:

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} \frac{\partial^3 f(\boldsymbol{p})}{\partial x_j, \partial x_k, \partial x_l} d_j d_k d_l = \frac{\partial}{\partial s^3} f(\boldsymbol{p} + \boldsymbol{d}s) = 0 \tag{A.2}$$

In words: $f(\boldsymbol{x})$ is a parabolic function if and only if for each location $\boldsymbol{p}$ the third directional derivative of $f(\boldsymbol{p})$ in each direction $\boldsymbol{d}$ is 0. Which is the case, since the third derivative of each intersection is 0.
$\boldsymbol{W}$ is positive definite since:

$$\forall \boldsymbol{d}, \boldsymbol{p} \in \mathbb{R}^n \wedge ||\boldsymbol{d}|| = 1 : \boldsymbol{d}^T\boldsymbol{W}\boldsymbol{d} = \frac{1}{2}\boldsymbol{d}^T\boldsymbol{H}(\boldsymbol{p})\boldsymbol{d} = \frac{1}{2}\frac{\partial}{\partial s^2} f(\boldsymbol{p} + \boldsymbol{d}s) = a > 0 \tag{A.3}$$

where $\boldsymbol{H}$ is the Hessian. □

Proposition 1. *PAL converges on $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto c + \boldsymbol{r}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ with $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ hermitian and positive definite.*

Proof.
For this prove we consider a basic *PAL* without the features introduced in Section 4.4.3. Note that along the proof we will see, that $a > 0$ and $b < 0$. Thus, only the update step for this case has to be considered (see Section 4.4.2.

$f(\boldsymbol{x})$ is convex since $\boldsymbol{Q}$ is positive definite. Thus, it has one minimum.

Without loss of generality we set $c = 0, \boldsymbol{r} = \boldsymbol{0}, \boldsymbol{x}_n \neq 0$

$$f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \text{ and } \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = f'(\boldsymbol{x}) = 2\boldsymbol{Q}\boldsymbol{x} \tag{A.4}$$

The values of $f(x)$ along a line through $\boldsymbol{x}$ in the direction of $-f'(\boldsymbol{x})$ are given by:

$$f(-f'(\boldsymbol{x})\hat{s} + \boldsymbol{x}) \tag{A.5}$$

Now we expand the line function:

$$\begin{aligned} f(-f'(\boldsymbol{x})\hat{s} + \boldsymbol{x}) &= f(-2\boldsymbol{Q}\boldsymbol{x}\hat{s} + \boldsymbol{x}) \\ &= (-2\boldsymbol{Q}\boldsymbol{x}\hat{s} + \boldsymbol{x})^T \boldsymbol{Q}(-2\boldsymbol{Q}\boldsymbol{x}\hat{s} + \boldsymbol{x}) \\ &= \underbrace{4\boldsymbol{x}^T\boldsymbol{Q}^3\boldsymbol{x}}_{=:a}\hat{s}^2 + \underbrace{-4\boldsymbol{x}^T\boldsymbol{Q}^2\boldsymbol{x}}_{=:b}\hat{s} + \underbrace{\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}}_{=:c} \end{aligned} \tag{A.6}$$

Here we see that $f(\hat{s})$ is indeed a parabolic function with $a > 0$, $b < 0$ and $c > 0$ since $\boldsymbol{Q}^3$, $\boldsymbol{Q}^2$ and $\boldsymbol{Q}$ are positive definite.
The location of the minimum $s_{\min}$ of $f(\hat{s})$ is given by:

$$\hat{s}_{\min} = \arg\min_{\hat{s}} f(-f'(\boldsymbol{x})\hat{s} + \boldsymbol{x}) = -\frac{b}{2a} \tag{A.7}$$

PAL determines $\hat{s}_{\min}$ exactly with $\hat{s}_{\min} = \frac{s_{\text{upd}}}{||f'(x)||}$ (see equation 1 and 2). $||f'(x)|| > 0$ since otherwise we are already in the minimum.

The value at the minimum is given by:

$$f(\hat{s}_{\min}) = a(\frac{-b}{2a})^2 + b(\frac{-b}{2a}) + c = -\frac{b^2}{4a} + c = -\underbrace{\frac{(-\boldsymbol{x}^T\boldsymbol{Q}^2\boldsymbol{x})^2}{\boldsymbol{x}^T\boldsymbol{Q}^3\boldsymbol{x}}}_{=:g(\boldsymbol{x})} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} = -g(\boldsymbol{x}) + f(\boldsymbol{x}) \tag{A.8}$$

Since $\boldsymbol{Q}^2$ and $\boldsymbol{Q}^3$ are positive definite and $\boldsymbol{x} \neq 0$:

$$g(\boldsymbol{x}) > 0 \tag{A.9}$$

Now we consider the sequence $f(\boldsymbol{x}_n)$, with $\boldsymbol{x}_n$ defined by *PAL* (see Equation 1):

$$\boldsymbol{x}_{n+1} = -\frac{f'(\boldsymbol{x}_n)}{||f'(x_n)||}\hat{s}_{\text{upd}} + \boldsymbol{x}_n = -f'(\boldsymbol{x}_n)\hat{s}_{\min} + \boldsymbol{x}_n \tag{A.10}$$

It is easily seen by induction that:

$$0 < f(\boldsymbol{x}_{n+1}) < f(\boldsymbol{x}_n) = \sum_{i=0}^{n-1} -g(\boldsymbol{x}_i) + f(\boldsymbol{x}_0) < f(\boldsymbol{x}_0). \tag{A.11}$$

$g(\boldsymbol{x}_n)$ converges to 0 since $\forall n : g(\boldsymbol{x_n}) > 0$ and $\sum_{i=0}^{n-1} -g(\boldsymbol{x}_i)$ is bounded.

Now we have to show that $\boldsymbol{x}_n$ converges to 0.
We have:

$$g(\boldsymbol{x}_n) = \frac{(\boldsymbol{x}_n^T \boldsymbol{Q}^2 \boldsymbol{x}_n)^2}{\boldsymbol{x}_n^T \boldsymbol{Q}^3 \boldsymbol{x}_n} = \frac{\langle \boldsymbol{x}_n, \boldsymbol{Q}^2 \boldsymbol{x}_n \rangle^2}{\langle \boldsymbol{x}_n, \boldsymbol{Q}^3 \boldsymbol{x}_n \rangle} \tag{A.12}$$

We use the theorem of Courant-Fischer:

$$\begin{aligned} \langle x, x \rangle \min\{\lambda_1, \ldots, \lambda_n\} \leq \langle x, Ax \rangle \leq \langle x, x \rangle \max\{\lambda_1, \ldots, \lambda_n\} \\ \text{for any symmetric } A \in \mathbb{R}^{n \times n} \text{ with } \lambda_1, \ldots, \lambda_n \end{aligned} \tag{A.13}$$

And get:

$$g(\boldsymbol{x}_n) \geq \frac{\lambda^2_{\boldsymbol{Q}^2 \min} \langle \boldsymbol{x}_n, \boldsymbol{x}_n \rangle^2}{\lambda_{\boldsymbol{Q}^3 \max} \langle \boldsymbol{x}_n, \boldsymbol{x}_n \rangle} = C \frac{||\boldsymbol{x}_n||^4}{||\boldsymbol{x}_n||^2} = C ||\boldsymbol{x}_n||^2 \tag{A.14}$$

with

$$C = \frac{\lambda^2_{\boldsymbol{Q}^2 \min}}{\lambda_{\boldsymbol{Q}^3 \max}} > 0 \text{ since all } \lambda \text{ of the positive definite } \boldsymbol{Q} \text{ are positive} \tag{A.15}$$

Thus, we have:

$$g(\boldsymbol{x}_n) \geq C ||\boldsymbol{x}_n||^2 \geq 0 \tag{A.16}$$

Since $g(\boldsymbol{x}_n)$ converges to 0, $C||\boldsymbol{x}_n||^2$ converges to 0.
This means, $\boldsymbol{x}_n$ converges to $\mathbf{0}$, which is the location of the minimum. □

Proposition 3. *If* $\mathcal{L}(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}\ \boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m c_i + \boldsymbol{r}_i^T \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{Q}_i \boldsymbol{\theta}$ *and* $c_i + \boldsymbol{r}_i^T \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{Q}_i \boldsymbol{\theta} = \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta})$ *with* m *being the number of batches* $\mathbb{B}_i$. *(Each batch defines a parabola. The empirical loss* $\mathcal{L}(\boldsymbol{\theta})$ *is the mean of these parabolas). And for all* $i, j \in \mathbb{N}$ *it holds that* $\boldsymbol{Q}_i = \boldsymbol{Q}_j$ *and that* $\boldsymbol{Q_i}$ *is positive definite. Then* $\arg\min_\theta \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \arg\min_\theta \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta})$ *holds.*

Proof.
Since $\mathcal{L}(\boldsymbol{\theta})$ is a sum of convex functions, it is also convex and has one minimum.
At first we determine the derivative of $\mathcal{L}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\boldsymbol{r}_i + 2\boldsymbol{Q}_i \boldsymbol{\theta}) = 2\boldsymbol{Q}\boldsymbol{\theta} + \frac{1}{m} \sum_{i=1}^m \boldsymbol{r}_i \tag{A.17}$$

Then we determine the minima:

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \Leftrightarrow \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbf{0} \Leftrightarrow \boldsymbol{\theta} = -\frac{1}{2}(\sum_{i=1}^{m} \boldsymbol{Q}_i)^{-1} \sum_{i=1}^{m} \boldsymbol{r_i} = -\frac{1}{2m} \boldsymbol{Q}^{-1} \sum_{i=1}^{m} \boldsymbol{r_i} \quad \text{(A.18)}$$

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta}) = -\frac{1}{2} \boldsymbol{Q}^{-1} \boldsymbol{r}_i \quad \text{(A.19)}$$

Thus, we get:

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2m} \boldsymbol{Q}^{-1} \sum_{i=1}^{m} \boldsymbol{r}_i = \frac{1}{m} \sum_{i=1}^{m} -\frac{1}{2} \boldsymbol{Q}^{-1} \boldsymbol{r}_i = \frac{1}{m} \sum_{i=1}^{m} \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathbb{B}i}(\boldsymbol{\theta}) \quad \text{(A.20)}$$

□

A.2 Further Experimental Results

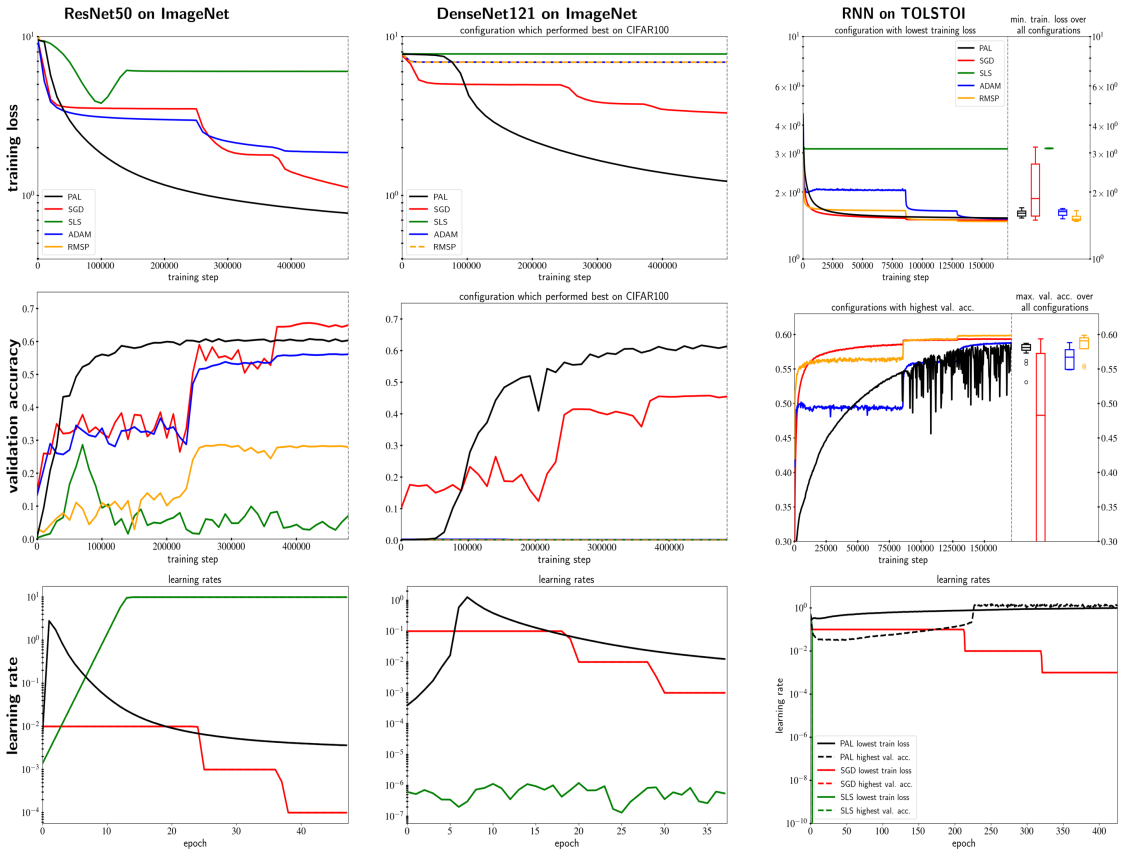


Figure A.4: Comparison of *PAL* to *SGD*, *SLS*, *ADAM*, *RMSProp* on training loss, validation accuracy and learning rates on **ImageNet**, and a simple RNN, trained on the **Tolstoi** War and Peace dataset. Learning rates are averaged over epochs. For ImageNet the best hyperparameter configuration from the CIFAR-100 evaluation were used to test hyperparameter transferability.

A.2.1 SLS ResNet34 Test Case Re-Implementation

In the conducted experiments and in contrast to the evaluation of *SLS* in (Vaswani *et al.*, 2019), we used TensorFlow default Xavier weight initialization (Glorot and Bengio, 2010) versus PyTorch default Lecun initialization (LeCun *et al.*, 2012). In addition, we used L2 regularisation versus no regularization. Furthermore, default implementations of networks for both frameworks have small differences. All in all, those differences usually influence the optimizer performance only marginally, as given by the fact that all other investigated optimizers perform well. However, in

this case of *SLS* we see significant differences.
To prove that our implementation of *SLS* is correct, we re-implemented (Vaswani *et al.*, 2019)'s ResNet34 test case on CIFAR-10 in TensorFlow and achieved similar results as (Vaswani *et al.*, 2019). SLS shows good performance and is not significantly overfitting as it does in Section 4.5.2.

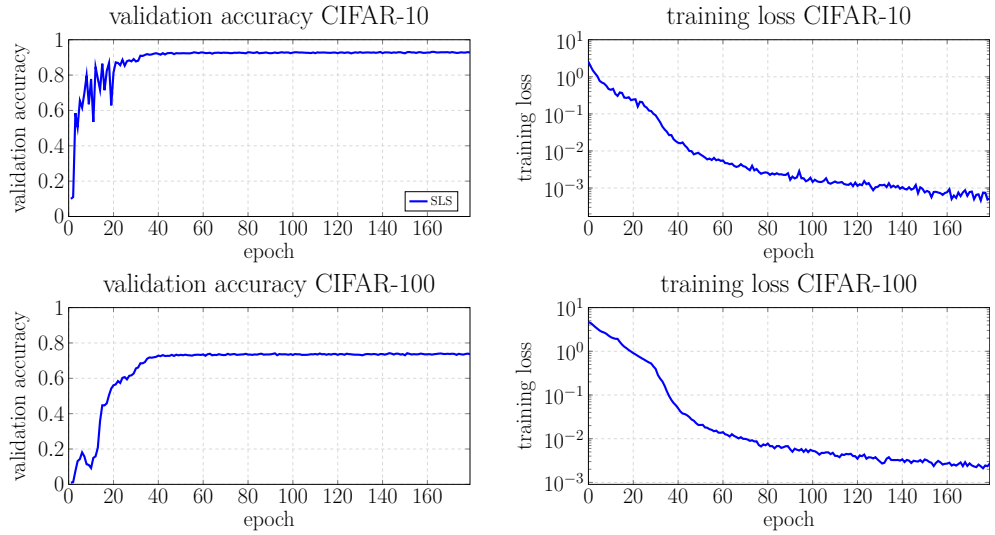


Figure A.5: On the re-implemented ResNet34 test case of (Vaswani *et al.*, 2019) SLS shows good performance and is not significantly overfitting as it does in Section 4.5.2

A.2.2 Sensitivity Analysis:

All in all, *PAL* tends to have a low hyperparameter sensitivity as shown in Figure A.6. Since μ is the most sensitive hyperparameter we analyzed its sensitivity over several further models trained on CIFAR-10 (see Figure A.7). After analyzing the best hyperparameter combinations of *PAL* over all experiments, we suggest to use values from the following parameter intervals: $\mu = [0.1, 1]$, $\alpha = [1.0, 1.6]$, $\beta = [0, 0.4]$, $s_{\max} = 3$. Where α, β and $s_{\max}$ usually have a low sensitivity.

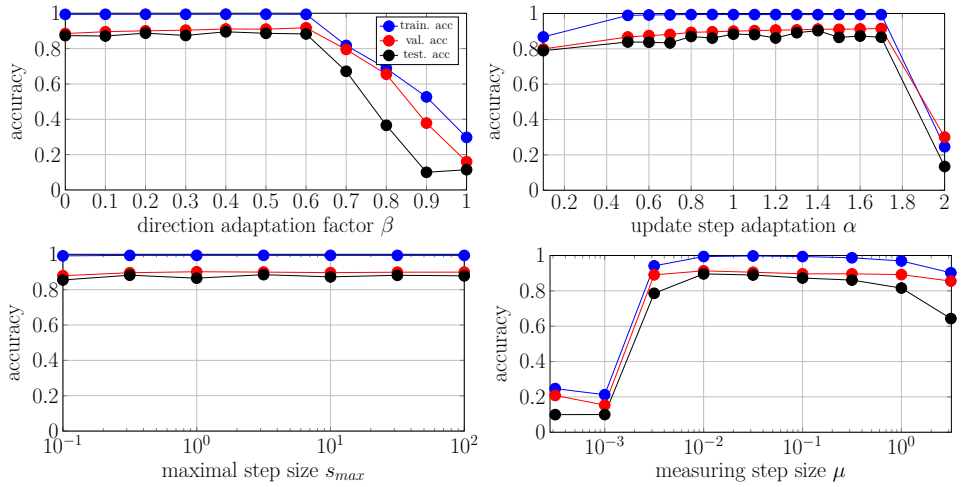


Figure A.6: Sensitivity analysis for PAL on a ResNet32 trained on CIFAR-10. The baseline parameters are: $\mu = 0.1, \beta = 0.2, \alpha = 1.0, s_{\max} = 10$. It shows that β should be chosen ≤ 0.6. α has a low sensitivity, but with a value of 1.4 it reaches best performance. $s_{\max}$ has a low sensitivity and all investigated values perform similarly. μ should be chosen between 10^{-2} and $10^{-0.5}$.

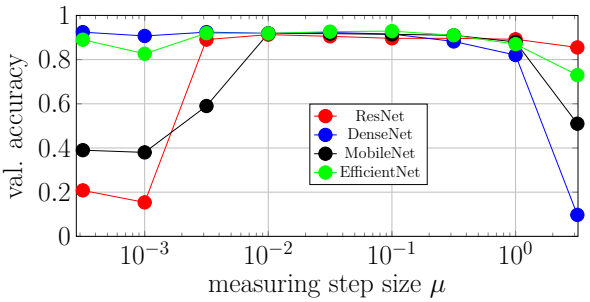


Figure A.7: Sensitivity of *PAL*'s the measuring step size μ for several models on CIFAR-10. *PAL* shows low sensitivity.

A.2.3 Further Experimental Design Details

Training Procedure

On CIFAR-10 and CIFAR-100 we trained 150k steps. On ImageNet each network was trained for 500k steps. We performed a piecewise constant learning rate decay by dividing the learning rate by 10 at 50% and 75% of the steps.
The training set to evaluation set split was 45k to 15k for CIFAR-10 and CIFAR-100. At the time of writing, the default TensorFlow classes did not support the reuse of the same randomly sampled numbers for multiple inferences, therefore, we implemented and used our own Dropout (Srivastava *et al.*, 2014) layer.
To get a fair comparison of the optimizers capabilities, we compared the training loss, the validation accuracy and the test accuracy metrics. For all metrics we provided the median and the quartiles to analyze the hyperparameter sensitivity. For each hyperparameter combination we averaged our results over 3 runs using the seeds 1, 2 and 3 for reproducibility. All in all, we trained over 4500 networks with TensorFlow 1.15 (Abadi *et al.*, 2016) on Nvidia Geforce GTX 1080 TI graphic cards.

Data Augmentation

On CIFAR-10 we performed the following augmentations (He *et al.*, 2016):
4 pixel padding and cropping, horizontal image flipping with probability 0.5.
On ImageNet we applied an initial random crop to 224x224 pixels. In addition, we applied lighting as described in (Krizhevsky *et al.*, 2012). For CIFAR-10, CIFAR-100 all images were normalized by channel-wise mean and variance. For the Tolstoi War and Peace dataset we omitted augmentation.

Hyperparameter grid search

For our evaluation we used all combinations out of the following commonly used hyperparameters. The batch size is 128 except for DenseNets trained with *ALIG*, *SGDHD* and *COCOB* for which we encountered memory overflows and had to reduce the batch size to 100. Weight decay is always 10^{-4}.
On ImageNet, such a comprehensive grid search was not possible. In this case we compared with the best hyperparameter combinations found on CIFAR-100.

ADAM:

hyperparameter	symbol	values
learning rate	λ	$\{1, 0.1, 0.01, 0.001, 0.0001\}$
first momentum	β_1	$\{0.9, 0.95\}$
second momentum	β_2	$\{0.999\}$
epsilon	ϵ	$\{1e-8\}$

We did not vary the first or second momentum much since (Kingma and Ba, 2015) states that the values chosen are already good defaults.

SGD:

hyperparameter	symbol	values
learning rate	λ	$\{0.1, 0.01, 0.001, 0.0001\}$
momentum	α	$\{0.85, 0.9, 0.95\}$

RMSProp:

hyperparameter	symbol	values
learning rate	λ	$\{0.1, 0.01, 0.001, 0.0001\}$
discounting factor	f	$\{0.9, 0.95\}$
epsilon	ϵ	$\{1e-8\}$

PAL:

hyperparameter	symbol	values
measuring step size	μ	$\{10^0, 10^{-0.5}, 10^{-0.1}, 10^{-0.15}\}$
direction adaptation factor	β	$\{0, 0.4\}$
update step adaptation	α	$\{1, \frac{1}{0.8}\}$
maximum step size	s_{max}	$\{10^{0.5} (\approx 3.16)\}$

In our implementation we worked with a inverse update step adaptation $\gamma = \frac{1}{\alpha}$.

SLS:

hyperparameter	symbol	values
initial step size	μ	$\{0.1, 1\}$
step size decay	β	$\{0.9, 0.99\}$
step size reset	γ	$\{2.0, 2.5\}$
Armijo constant	c	$\{0.1, 0.01\}$
maximum step size	μ_{max}	$\{10.0\}$

ALIG:

hyperparameter	symbol	values
maximal learning rate	λ	$\{10, 1.0, 0.1, 0.01\}$
momentum	β	$\{0.85, 0.9, 0.95\}$

COCOB:

hyperparameter	symbol	values
restriction factor	α	$\{25, 50, 75, 100, 125, 150, 175, 200\}$

SGDHD:

hyperparameter	symbol	values
learning rate	λ	$\{0.1, 0.01, 0.001\}$
hyper gradient learning rate	β	$\{0.1, 0.01, 0.001, 0.0001\}$

A.2.4 Detailed Numerical Results

Table A.1: Performance comparison of *PAL*, *RMSProp*, *ADAM*, *COCOB*, *SGDHD*, *ALIG* and *SGD*. All hyperparameter combinations given in Appendix A.2.3 were evaluated for each architecture. Results were averaged over three runs starting from different random seeds, except for training on ImageNet, for which results were not averaged. Note that tests on ImageNet were performed with the best hyperparameters found on CIFAR-100 to test the transferability of hyperparameters. Medians and quartiles describe the distribution of results over reasonable hyperparameter ranges.

dataset	network	optimizer	training loss		validation accuracy		test accuracy	
			min	median; p25; p75	max	median; p25; p75	max	median; p25; p75
CIFAR-10	EfficientNet	COCOB	0.659	0.824; 0.739; 0.855	0.857	0.837; 0.832; 0.845	0.843	0.824; 0.818; 0.832
		ALIG	0.279	0.89; 0.464; 1.911	0.906	0.805; 0.451; 0.895	0.893	0.757; 0.297; 0.878
		SGDHD	2.002	6.239; 4.357; 7.803	0.834	0.657; 0.18; 0.74	0.828	0.647; 0.179; 0.731
		SLS	2.837	5.596; 4.681; 6.292	0.653	0.357; 0.211; 0.443	0.643	0.357; 0.216; 0.442
		RMSP	0.154	0.637; 0.333; 1.261	**0.93**	0.864; 0.658; 0.902	0.919	0.854; 0.648; 0.889
		ADAM	0.155	0.818; 0.292; 2.275	0.926	0.841; 0.211; 0.907	0.919	0.83; 0.1; 0.896
		SGD	0.165	2.287; 0.343; 4.221	**0.93**	0.872; 0.794; 0.915	**0.921**	0.862; 0.784; 0.906
		PAL	**0.137**	**0.244**; 0.186; 0.388	0.927	**0.912**; 0.906; 0.921	0.916	**0.902**; 0.889; 0.908
CIFAR-10	MobileNetV2	COCOB	0.232	**0.282**; 0.257; 0.295	0.879	0.87; 0.866; 0.876	0.865	0.852; 0.848; 0.865
		ALIG	0.183	0.938; 0.347; 1.926	0.914	0.695; 0.233; 0.888	0.897	0.528; 0.1; 0.851
		SGDHD	0.698	2.234; 1.835; 4.366	0.886	0.75; 0.298; 0.807	0.877	0.737; 0.295; 0.791
		SLS	1.387	2.462; 2.011; 2.584	0.667	0.443; 0.407; 0.504	0.595	0.4; 0.343; 0.437
		RMSP	**0.085**	0.493; 0.337; 0.918	0.938	0.872; 0.675; 0.895	0.929	0.865; 0.664; 0.882
		ADAM	0.095	0.477; 0.314; 1.861	0.939	0.874; 0.309; 0.896	0.93	0.864; 0.289; 0.886
		SGD	0.149	0.878; 0.204; 1.552	**0.947**	**0.907**; 0.87; 0.933	**0.94**	**0.899**; 0.859; 0.925
		PAL	0.15	0.377; 0.205; 0.531	0.92	0.905; 0.896; 0.91	0.905	0.886; 0.877; 0.896
CIFAR-10	DenseNet40	COCOB	0.228	0.234; 0.23; 0.24	0.907	0.903; 0.901; 0.904	0.894	0.889; 0.885; 0.892
		ALIG	0.188	0.604; 0.227; 2.903	0.918	0.848; 0.438; 0.902	0.902	0.784; 0.336; 0.875
		SGDHD	1.094	2.279; 1.349; 2.908	0.775	0.341; 0.099; 0.696	0.762	0.1; 0.1; 0.26
		SLS	**0.065**	**0.115**; 0.104; 0.189	0.91	0.904; 0.897; 0.905	0.901	**0.893**; 0.89; 0.897
		RMSP	0.147	0.398; 0.256; 0.915	0.927	0.879; 0.737; 0.915	0.92	0.867; 0.717; 0.909
		ADAM	0.138	0.749; 0.274; 1.028	0.922	0.777; 0.611; 0.91	0.913	0.806; 0.605; 0.907
		SGD	0.147	0.794; 0.396; 1.746	**0.932**	0.855; 0.537; 0.914	**0.93**	0.847; 0.528; 0.91
		PAL	0.099	0.217; 0.165; 0.343	0.925	**0.907**; 0.894; 0.919	0.916	0.882; 0.861; 0.9
CIFAR-10	ResNet32	COCOB	0.125	0.128; 0.127; 0.129	0.888	0.886; 0.885; 0.887	0.878	0.872; 0.871; 0.874
		ALIG	0.122	0.658; 0.279; 1.485	0.892	0.815; 0.47; 0.881	0.866	0.71; 0.367; 0.852
		SGDHD	0.35	0.464; 0.413; 0.701	0.864	0.835; 0.791; 0.843	0.837	0.796; 0.766; 0.827
		SLS	**0.005**	**0.006**; 0.005; 0.827	0.871	0.856; 0.758; 0.869	0.846	0.824; 0.657; 0.839
		RMSP	0.105	0.199; 0.129; 0.498	0.922	0.884; 0.804; 0.904	0.915	0.877; 0.792; 0.896
		ADAM	0.105	0.332; 0.133; 1.004	0.917	0.875; 0.677; 0.881	0.914	0.868; 0.654; 0.873
		SGD	0.098	0.131; 0.118; 0.322	**0.939**	**0.899**; 0.85; 0.924	**0.933**	**0.893**; 0.838; 0.92
		PAL	0.05	0.105; 0.075; 0.195	0.921	0.893; 0.887; 0.906	0.903	0.88; 0.849; 0.888
CIFAR-100	DenseNet40	COCOB	0.739	0.761; 0.75; 0.772	0.642	0.633; 0.631; 0.637	0.646	0.632; 0.629; 0.637
		ALIG	0.488	2.125; 0.988; 3.128	0.637	0.508; 0.391; 0.623	0.616	0.48; 0.264; 0.605
		SGDHD	1.78	2.6; 2.179; 3.465	0.566	0.418; 0.274; 0.504	0.55	0.296; 0.159; 0.497
		SLS	1.367	1.908; 1.446; 1.96	0.719	0.593; 0.572; 0.698	0.612	0.479; 0.422; 0.554
		RMSP	0.348	1.238; 0.78; 1.972	0.716	0.583; 0.481; 0.634	0.712	0.588; 0.482; 0.631
		ADAM	0.326	1.114; 0.859; 3.53	0.715	0.601; 0.165; 0.637	0.712	0.599; 0.226; 0.641
		SGD	0.376	0.713; 0.431; 2.154	**0.75**	0.633; 0.489; 0.709	**0.753**	0.634; 0.498; 0.708
		PAL	**0.275**	**0.376**; 0.312; 0.459	0.73	**0.686**; 0.66; 0.705	0.717	**0.676**; 0.642; 0.695
CIFAR-100	EfficientNet	COCOB	0.802	0.817; 0.807; 0.822	0.594	0.583; 0.581; 0.59	0.596	0.582; 0.58; 0.588
		ALIG	0.57	2.4; 0.995; 4.085	0.612	0.494; 0.169; 0.6	0.599	0.458; 0.115; 0.597
		SGDHD	3.545	6.528; 5.519; 8.917	0.529	0.337; 0.178; 0.463	0.513	0.342; 0.179; 0.468
		SLS	3.731	6.713; 6.348; 6.857	0.474	0.212; 0.208; 0.227	0.375	0.203; 0.149; 0.208
		RMSP	0.422	1.823; 1.253; 2.968	0.675	0.517; 0.383; 0.588	0.678	0.521; 0.382; 0.59
		ADAM	0.45	1.394; 1.312; 4.606	0.684	0.518; 0.025; 0.619	0.684	0.524; 0.01; 0.621
		SGD	0.42	2.44; 0.633; 5.214	**0.712**	0.579; 0.473; 0.661	**0.709**	0.579; 0.476; 0.658
		PAL	**0.372**	**0.471**; 0.409; 0.772	0.693	**0.666**; 0.638; 0.676	0.69	**0.664**; 0.63; 0.671
CIFAR-100	MobileNetV2	COCOB	0.486	**0.513**; 0.492; 0.536	0.644	0.63; 0.626; 0.637	0.644	0.63; 0.623; 0.638
		ALIG	0.323	2.396; 0.817; 4.247	0.661	0.41; 0.034; 0.623	0.652	0.229; 0.01; 0.602
		SGDHD	1.485	3.307; 2.425; 7.002	0.593	0.476; 0.39; 0.545	0.589	0.456; 0.385; 0.525
		SLS	3.857	5.086; 5.031; 5.64	0.332	0.2; 0.099; 0.203	0.197	0.081; 0.052; 0.126
		RMSP	0.198	1.518; 0.718; 3.368	0.728	0.593; 0.43; 0.635	0.727	0.593; 0.431; 0.634
		ADAM	0.218	1.873; 0.776; 4.524	0.729	0.528; 0.025; 0.593	0.729	0.533; 0.02; 0.595
		SGD	0.4	0.974; 0.473; 2.151	**0.733**	0.657; 0.57; 0.7	**0.736**	0.659; 0.573; 0.701
		PAL	**0.181**	0.602; 0.314; 1.571	0.726	**0.666**; 0.574; 0.689	0.722	**0.664**; 0.509; 0.681

CIFAR-100	ResNet32	COCOB	0.498	0.569; 0.524; 0.673	0.609	0.608; 0.607; 0.608	0.605	0.602; 0.599; 0.604
		ALIG	0.537	1.932; 0.995; 3.572	0.597	0.491; 0.19; 0.58	0.587	0.414; 0.144; 0.549
		SGDHD	0.881	1.359; 1.06; 1.772	0.601	0.539; 0.472; 0.586	0.599	0.517; 0.431; 0.571
		SLS	2.62	2.808; 2.78; 2.82	0.399	0.388; 0.384; 0.392	0.363	0.305; 0.274; 0.33
		RMSP	0.519	1.019; 0.807; 2.083	0.661	0.599; 0.455; 0.651	0.656	0.603; 0.455; 0.65
		ADAM	0.402	1.772; 0.768; 3.038	0.659	0.513; 0.262; 0.564	0.658	0.519; 0.255; 0.567
		SGD	0.375	**0.474**; 0.4; 1.522	**0.697**	0.614; 0.494; 0.672	**0.694**	0.616; 0.502; 0.667
		PAL	**0.339**	0.485; 0.369; 1.424	0.662	**0.636**; 0.546; 0.652	0.663	**0.621**; 0.512; 0.647
TOLSTOI	RNN	COCOB	1.506	1.56; 1.533; 1.593	0.589	0.58; 0.573; 0.584	0.582	0.572; 0.566; 0.577
		ALIG	1.501	1.562; 1.528; 1.766	0.591	0.579; 0.523; 0.586	0.584	0.571; 0.513; 0.577
		SGDHD	2.282	2.433; 2.379; 2.445	0.375	0.338; 0.336; 0.348	0.369	0.334; 0.332; 0.344
		SLS	3.128	3.149; 3.136; 3.156	0.169	0.159; 0.158; 0.165	0.168	0.158; 0.157; 0.164
		RMSP	**1.475**	**1.509**; 1.492; 1.556	**0.599**	**0.591**; 0.579; 0.595	**0.592**	**0.583**; 0.572; 0.587
		ADAM	1.516	1.655; 1.596; 1.681	0.588	0.567; 0.55; 0.578	0.581	0.561; 0.543; 0.571
		SGD	1.496	1.872; 1.56; 2.675	0.594	0.483; 0.278; 0.573	0.587	0.476; 0.275; 0.566
		PAL	1.528	1.569; 1.547; 1.588	0.587	0.581; 0.577; 0.586	0.579	0.571; 0.556; 0.575
ImageNet	ResNet50	COCOB	2.0	–	0.51	–	0.518	–
		ALIG	1.854	–	0.539	–	0.512	–
		SGDHD	2.742	–	0.498	–	0.495	–
		RMSP	9.485	–	0.286	–	0.28	–
		ADAM	1.863	–	0.562	–	0.559	–
		SLS	3.808	–	0.286	–	0.069	–
		SGD	1.123	–	**0.656**	–	**0.65**	–
		PAL	**0.773**	–	0.608	–	0.608	–
ImageNet	DenseNet121	COCOB	6.9	–	0.006	–	0.006	–
		ALIG	2.142	–	0.533	–	0.512	–
		SGDHD	2.939	–	0.343	–	0.362	–
		RMSP	6.901	–	0.0	–	0.0	–
		ADAM	6.901	–	0.001	–	0.0	–
		SLS	7.768	–	0.001	–	0.001	–
		SGD	3.308	–	0.458	–	0.452	–
		PAL	**1.228**	–	**0.617**	–	**0.611**	–

Appendix B

Empirically Explaining SGD from a Line Search Perspective

B.1 Further Results on ResNet-20

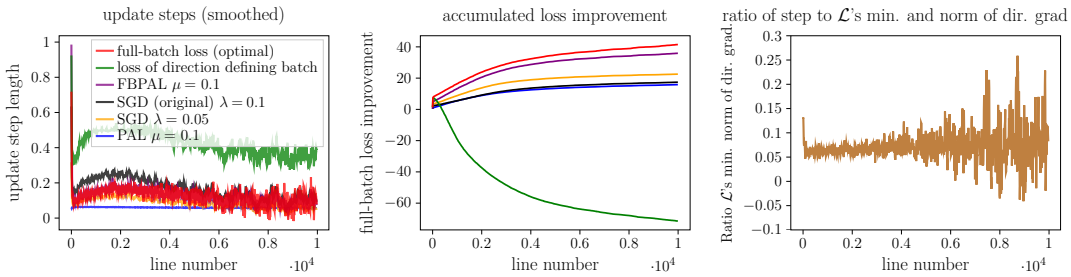


Figure B.1: SGD training process with momentum 0.9. See Figure 5.7 for explanations. The core difference is, that for the proportionality, the noise is higher than in the SGD case. In addition, SGD with momentum overshoots the locally optimal step size less and does not perform an as exact line search.

B.2 Analyses of ResNet-18 and MobileNetV2

B.2.1 Distance Matrices

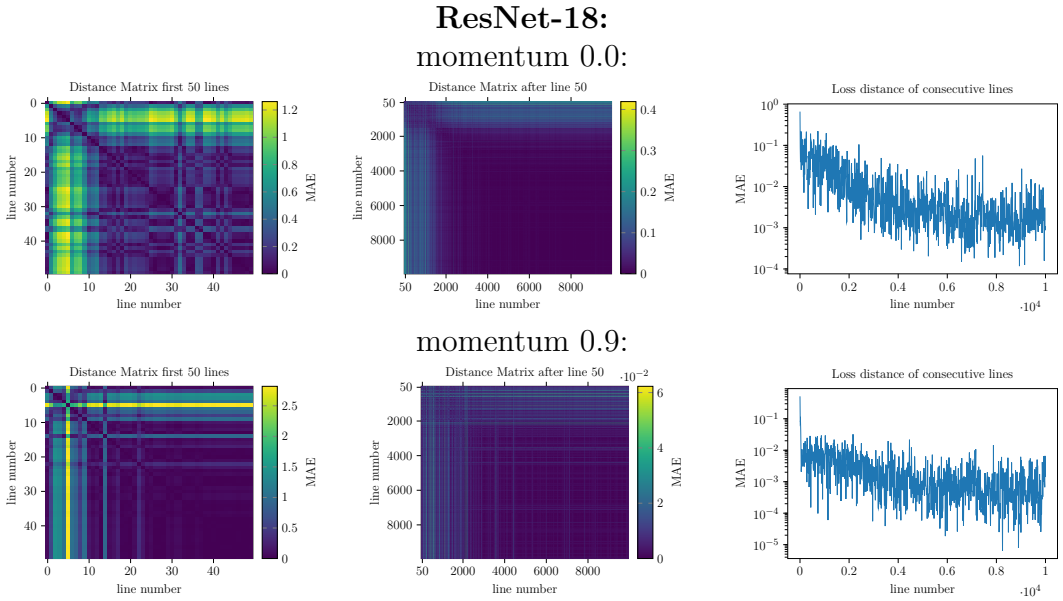


Figure B.2: ResNet-18: Distances of the shape of full-batch losses along lines in a window around the current position $s = 0$. **Row 1:** SGD without momentum. **Row 2:** SGD with momentum. Since the offset is not of interest the minimum is shifted to 0 on the y-axis. The distances are rather high for the first 20 lines (left). For the following lines the distances are less than 0.4 MAE (middle) and concentrate around 0.005. The MAEs of the full-batch loss of pairs of consecutive lines are given on the right.

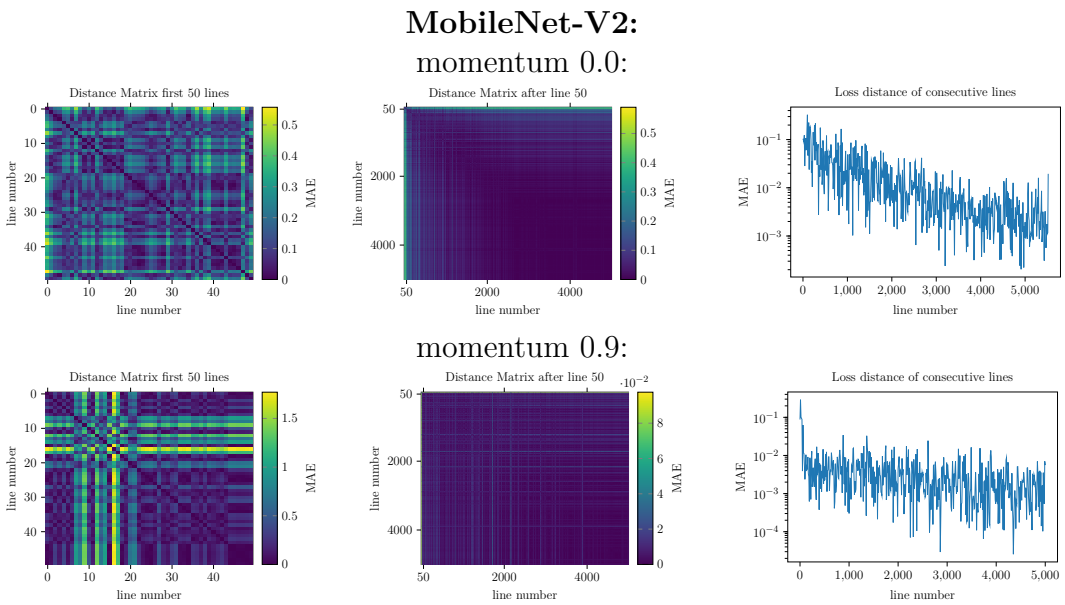


Figure B.3: MobileNet-V2: See Figure B.2 for explanations. The distances are rather high for the first 25 lines (left). For the following lines the distances are less then 0.6 MAE (middle) and concentrate around 0.01.

B.2.2 Parabolic Approximation

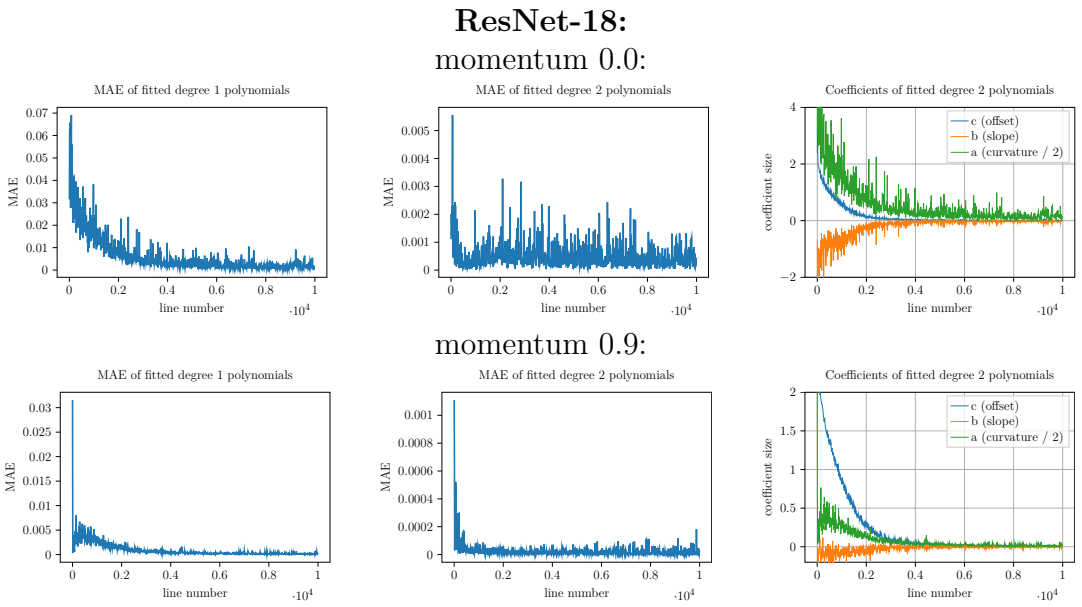


Figure B.4: ResNet-18: MAE of polynomial approximations of the full-batch loss of degree one and two. **Row 1:** SGD without momentum. **Row 2:** SGD with momentum. Full-batch losses along lines can be well fitted by polynomials of degree 2. The slope of the approximation stays roughly constant whereas the curvature decreases.

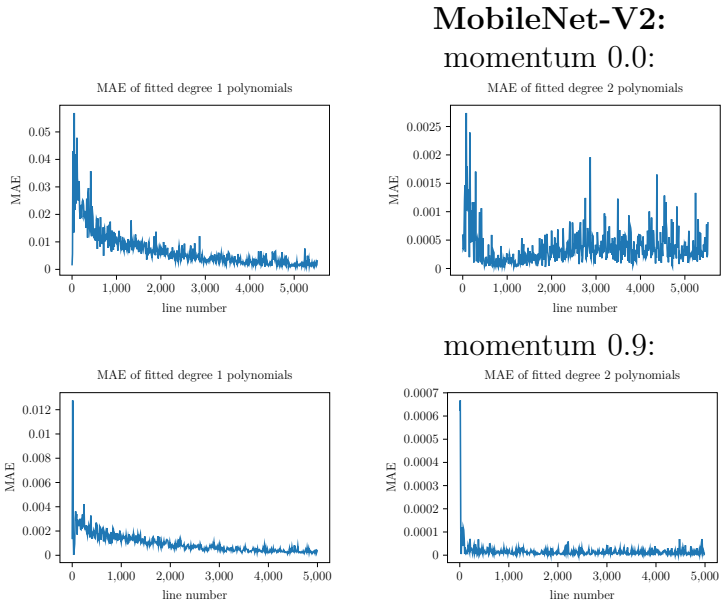


Figure B.5: MobileNet-V2: For explanations and interpretations see Figure B.4.

B.2.3 Optimization Strategy Metrics

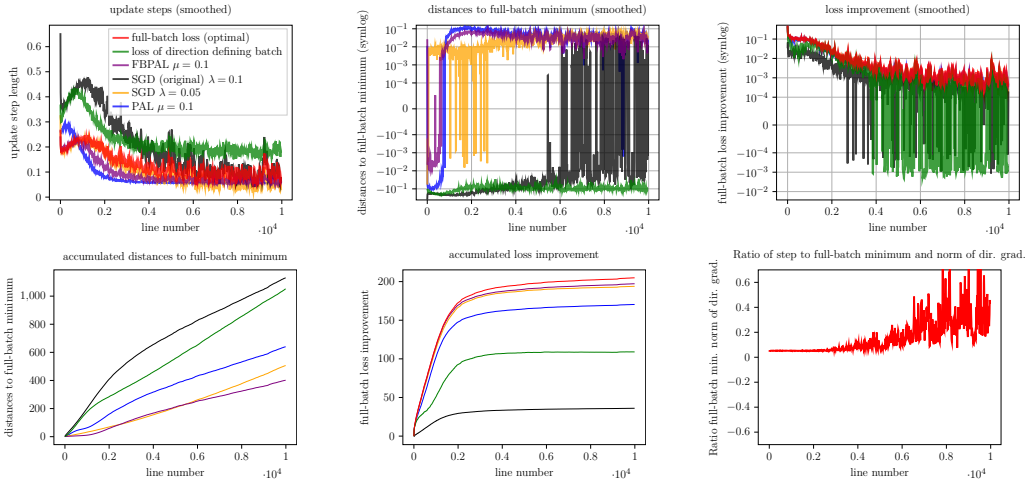


Figure B.6: SGD training process without momentum on ResNet18. Several metrics to compare update step strategies: 1. the performed update steps, 2. the distance to the minimum of the full batch loss ($s_{\text{opt}} - s_{\text{upd}}$), which is the optimal update step from a local perspective. 3. the loss improvement per step given as: $l(0) - l(s_{\text{upd}})$) where s_{upd} is the update step of a strategy. Average smoothing with a kernel size of 25 is applied. In this case the ratio of the full batch minimum location with the norm of the direction defining gradient increases during the end of the training. The proportionality is only given in the beginning of the training.

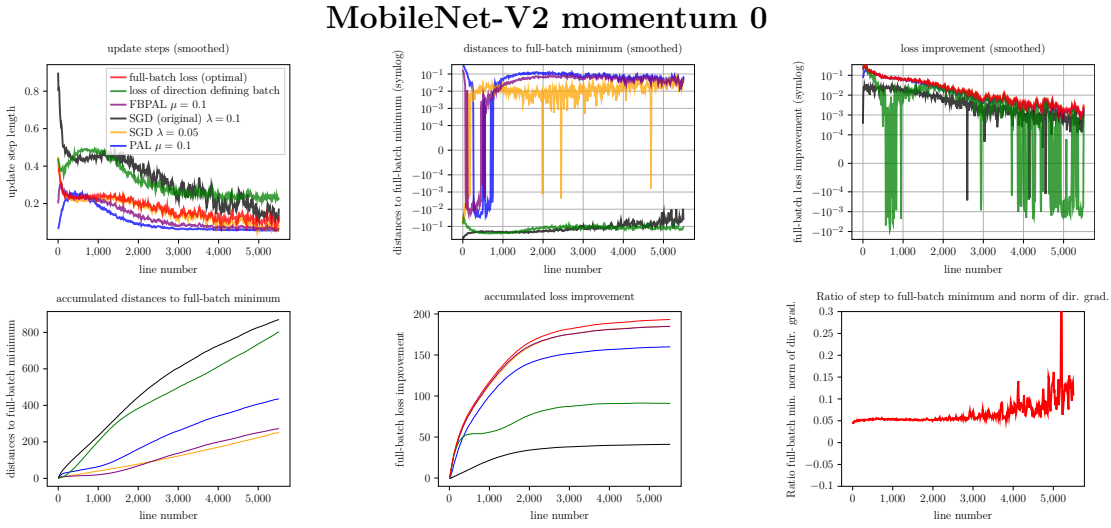


Figure B.7: MobileNet-V2 momentum 0. See Figure B.6 for explanations.

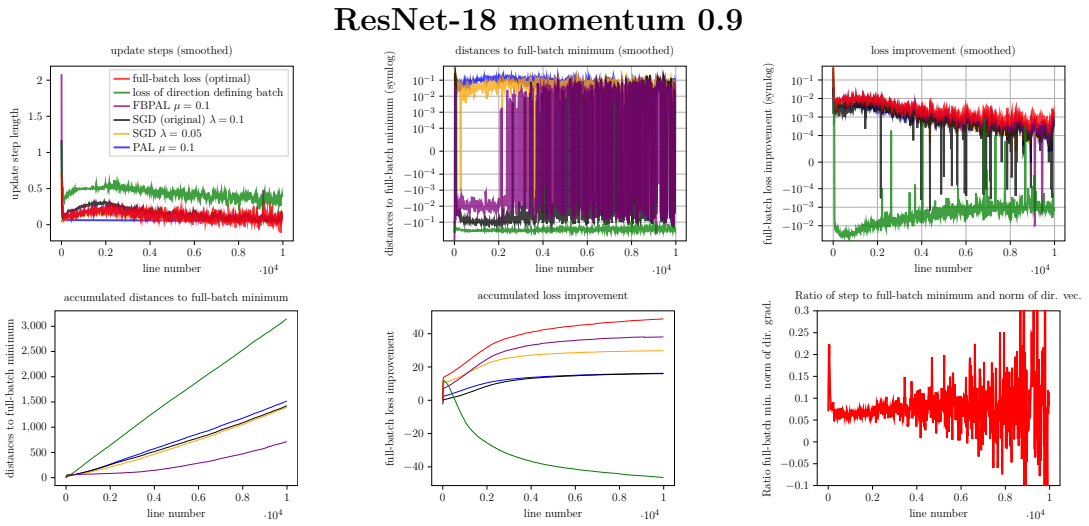


Figure B.8: ResNet-18 momentum 0.9. See Figure B.6 for explanations. In the case of momentum SGD is not able to perform such an exact line search as in the case without monemtum since the norm of the momentum vector is not directly related to the loss of the current line considered.

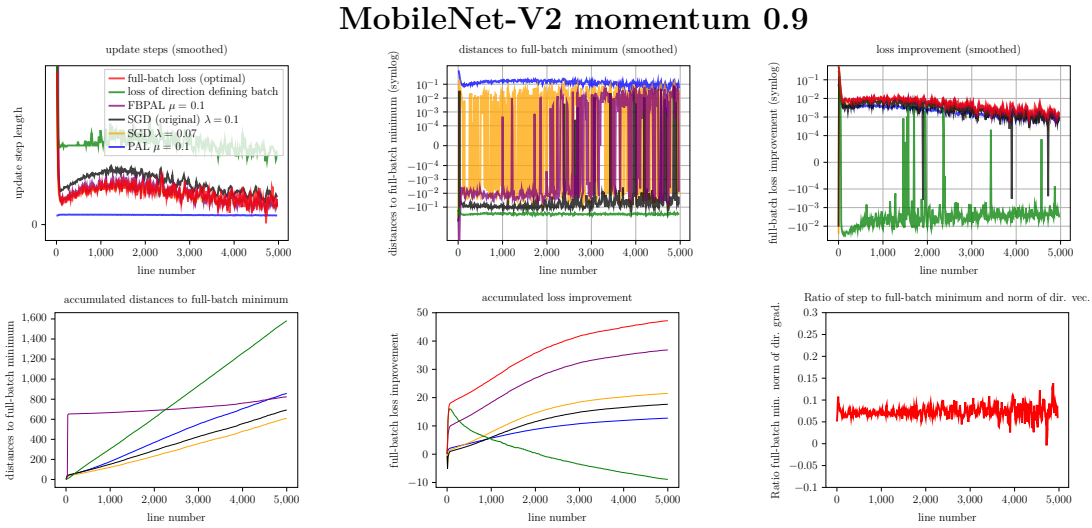


Figure B.9: MobileNet-V2 momentum 0.9. See Figure B.6 and B.8 for explanations and interpretations.

B.2.4 Batch Size Comparison

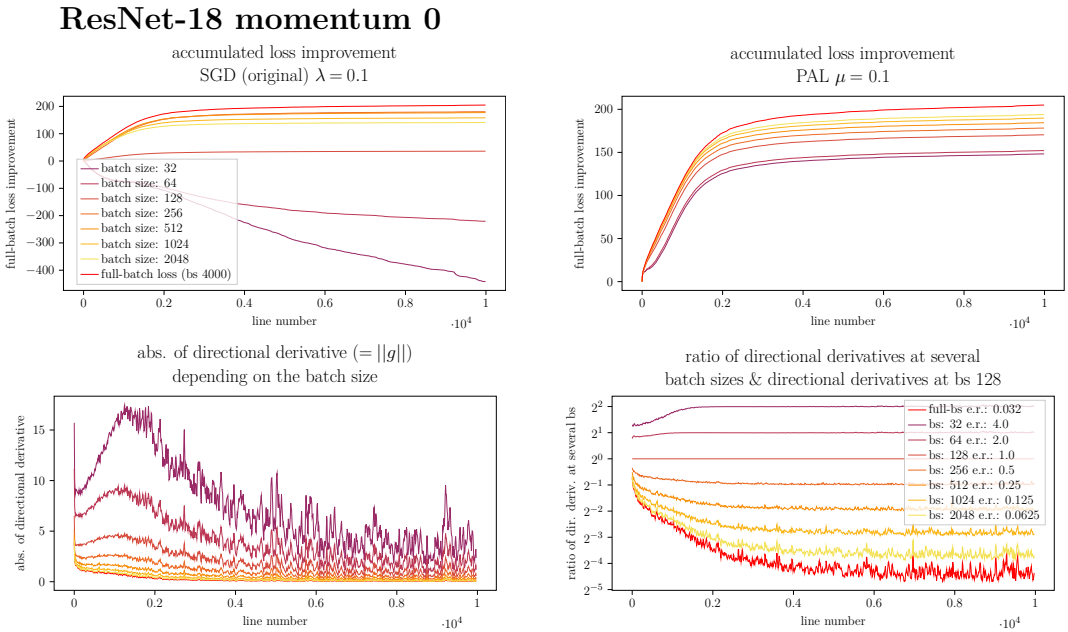


Figure B.10: Row 1: Comparing the influence of the batch size on the loss improvement. Left: SGD with the original learning rate of 0.1. Right: PAL. **Row 2:** Analysis of the relation of the batch size to the absolute directional derivative (=gradient norm) which shows in detail that increasing the batch size has a similar effect as decreasing the learning rate by the same factor. **e.r.** stands for expected ratio.

.

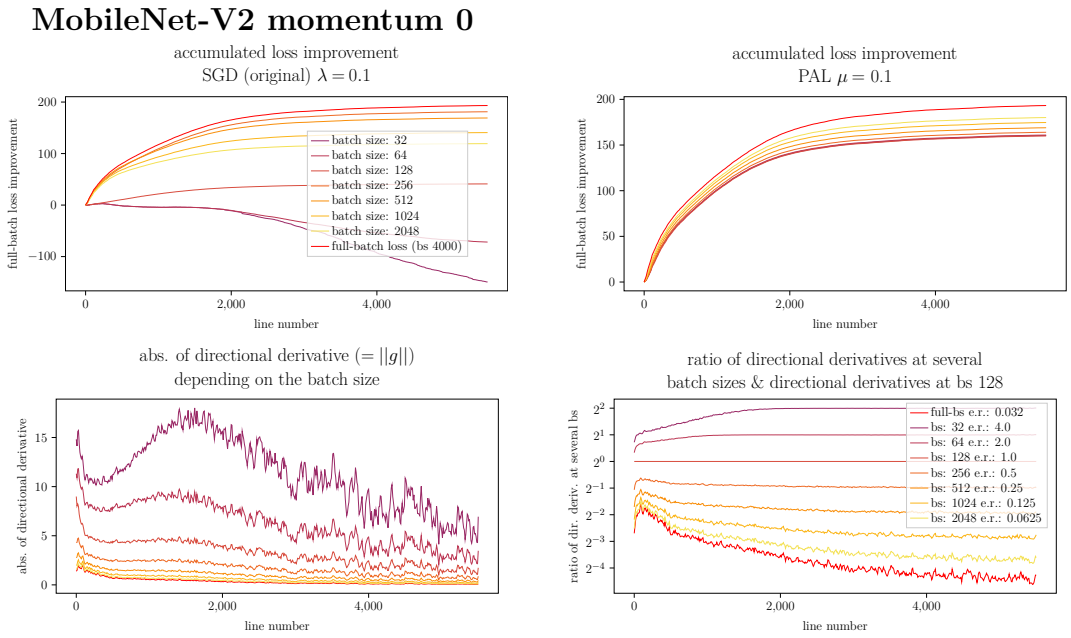


Figure B.11: MobileNet-V2 momentum 0: See Figure B.10 for explanations.

Appendix C

Large Batch Parabolic Approximation Line Search

C.1 Further Performance Comparisons

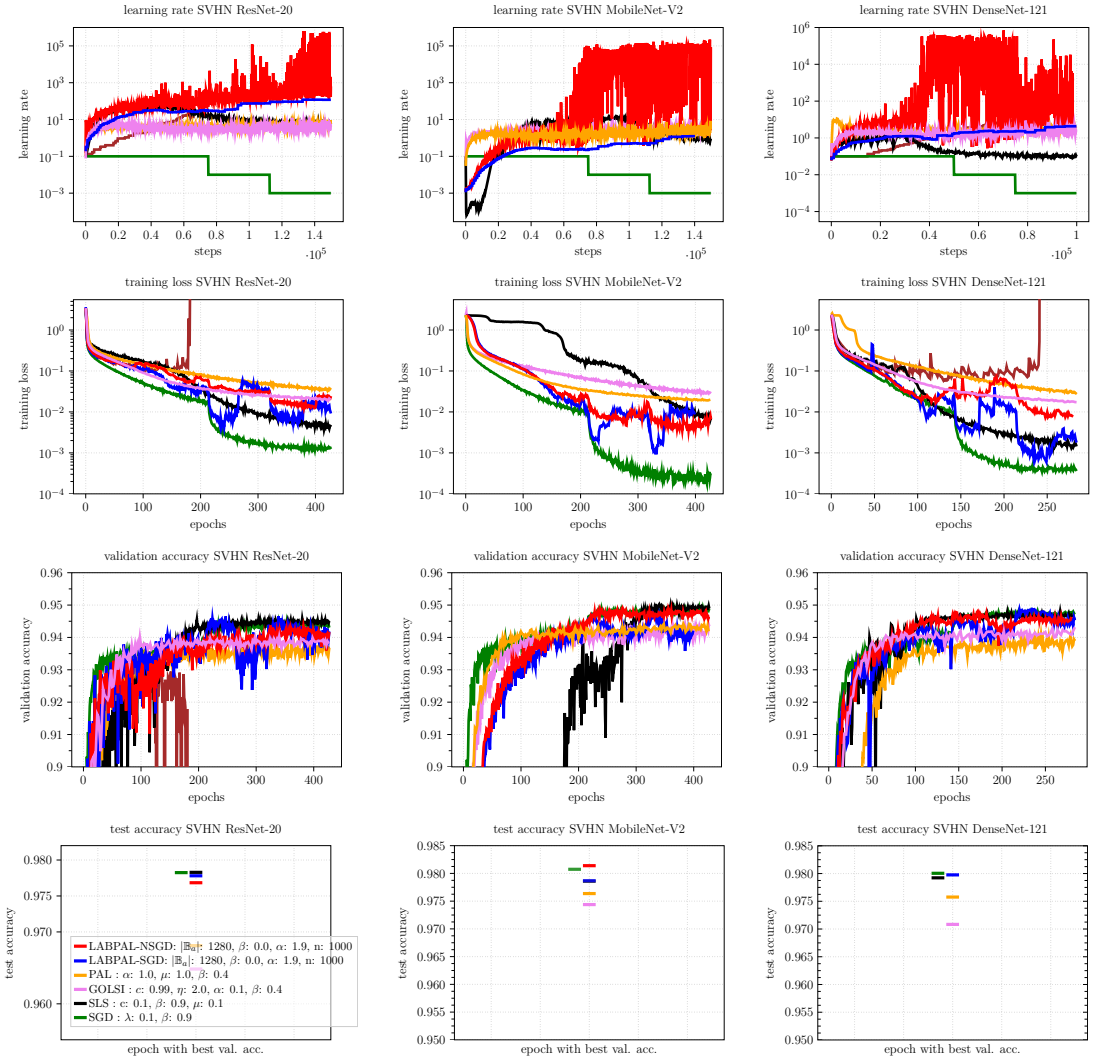


Figure C.1: Performance comparison on **SVHN** of our approach LABPAL in the SGD and NSGD variants against several line search and SGD. Optimal hyperparameters found with a detailed grid search for CIFAR-10 are reused. Our approaches challenge and sometimes surpass the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics.

C.2 Further Results for Batch Sizes 32 and 8

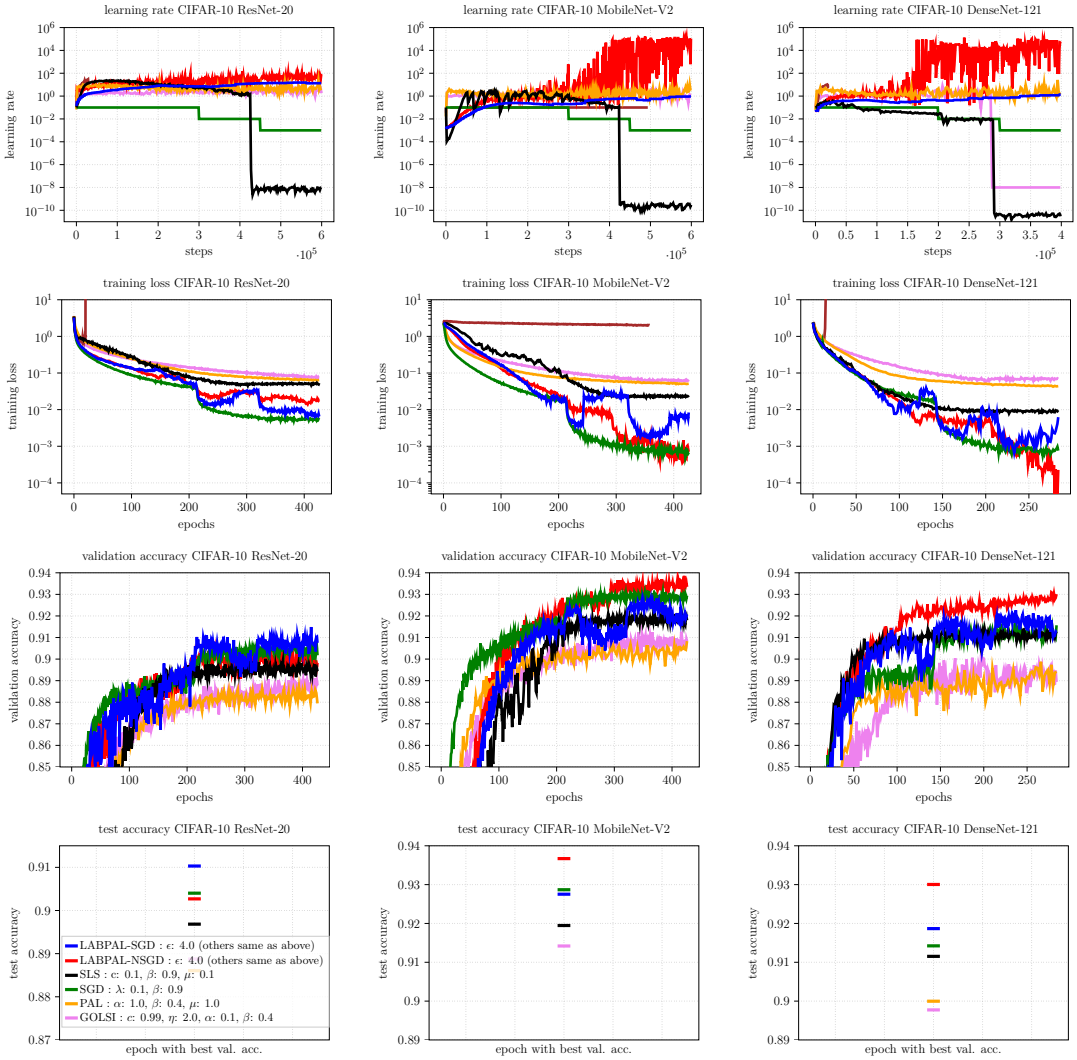


Figure C.2: Performance comparison of several models on CIFAR-10 with **batch size 32**. **The same hyperparameters are used as for batch size 128** (see Figure 6.4). Due to the batch size adaptation to keep the noise scale on a similar level, the LABPAL approaches perform almost identical compared to batch size 128. The performance off all other line searches decreases; SGD still performs well. Note that training steps were increased by a factor of 4.

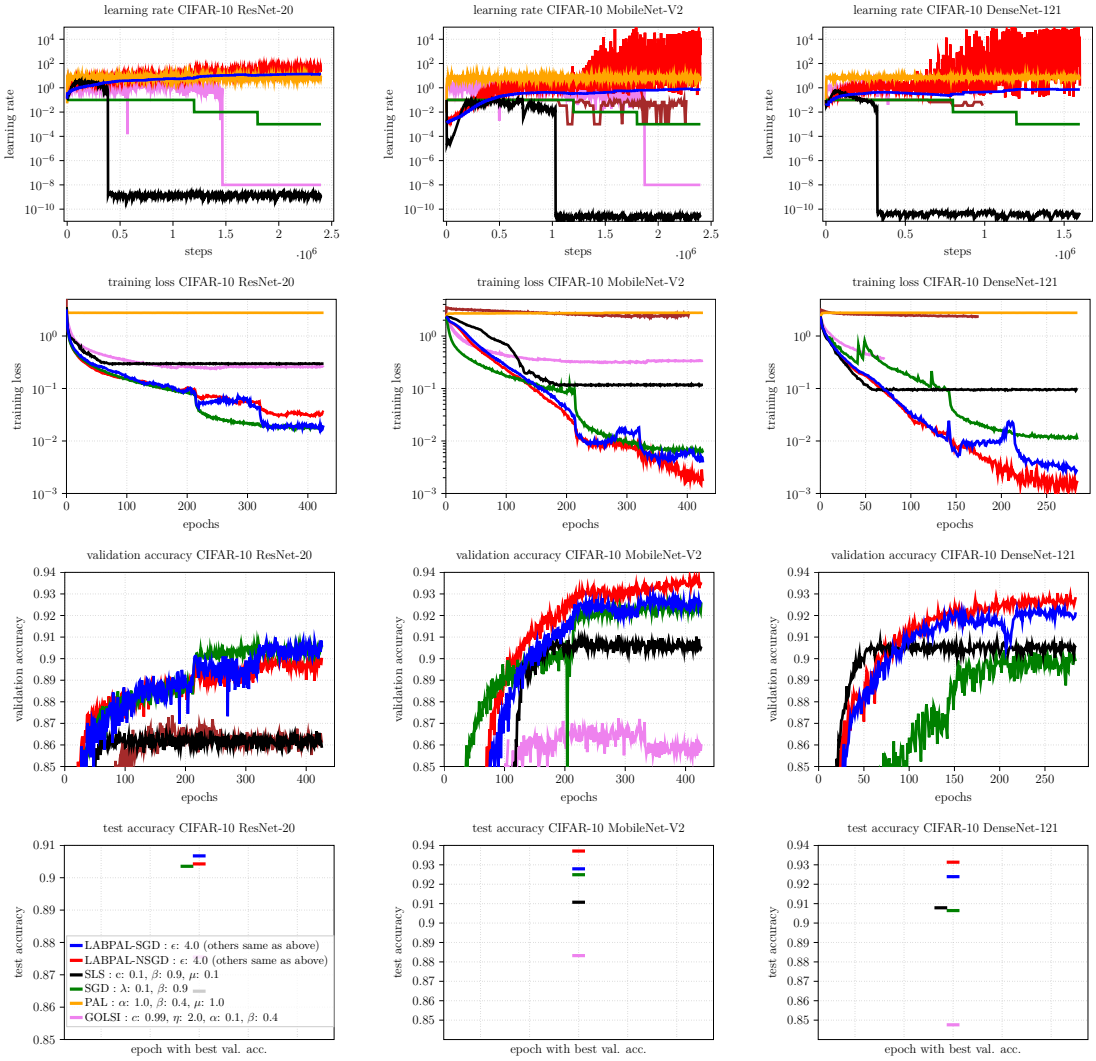


Figure C.3: Performance comparison of several models on CIFAR-10 with **batch size 8**. **The same hyperparameters are used as for batch size 128** (see Figure 6.4). Due to the batch size adaptation to keep the noise scale on a similar level the LABPAL approaches still perform well. PAL and PLS fail to optimize at all. SGD still performs well. SLS still shows good performance, however if training longer this will not hold since the learning rate schedules degenerate. Note that training steps were increased by a factor of 16.

C.3 Theoretical Considerations

As the field has not yet identified the reason for the local parabolic behavior of the full-batch loss is and, thus, what an appropriate function space to consider for convergence is, we refer to the theoretical analysis of (Mutschler and Zell, 2020a); in that we show convergence on a quadratic loss. This is also valid for LABPAL, with the addition that each mini batch-loss can be of any form as long as the mean over these losses is a quadratic function.

C.4 Further Experimental Details

Further experimental details for the optimizer comparison in Figure 6.4, 6.5, C.1, C.2, C.3 of Sections 6.4.2 & 6.4.3.
PLS: We adapted the only available and empirically improved TensorFlow (Abadi *et al.*, 2016) implementation of PLS (Lukas Balles, 2017), which was transferred to PyTorch (Paszke *et al.*, 2019) by (Vaswani *et al.*, 2019), to run on several state-of-the-art models and datasets.
The training steps for the experiments in Section 6.4 were 100,000 for DenseNet and 150,000 steps for MobileNetv2 and ResNet-20. Note that we define one training step as processing one input batch to keep line search approaches comparable.
The batch size was 128 for all experiments. The validation/training set splits were: 5,000/45,000 for CIFAR-10 and CIFAR-100 20,000/45,000 for SVHN.
All images were normalized with a mean and standard deviation determined over the dataset. We used random horizontal flips and random cropping of size 32. The padding of the random crop was 8 for CIFAR-100 and 4 for SVHN and CIFAR-10.
All trainings were performed on Nvidia Geforce 1080-TI GPUs.
Results were averaged over three runs initialized with three different seeds for each experiment.
For implementation details, refer to the source code provided at
`https://github.com/cogsys-tuebingen/LABPAL`.

C.4.1 Hyperparameter Grid Search on CIFAR-10

For our evaluation, we used all combinations out of the following hyperparameters.

SGD:

hyperparameter	symbol	values
learning rate	λ	$\{0.001, 0.01, 0.1, 1.0\}$
momentum	α	$\{0, 0.4, 0.9\}$
learning rate schedule		$\begin{cases} \lambda, & \text{if } t \leq \lfloor t_{\max} \cdot 0.5 \rfloor \\ \lambda/10, & \text{elif } t \leq \lfloor t_{\max} \cdot 0.75 \rfloor \\ \lambda/100, & \text{elif } t > \lfloor t_{\max} \cdot 0.75 \rfloor \end{cases}$, where t_{max} is the amount of training steps

PAL:

hyperparameter	symbol	values
measuring step size	μ	$\{0.01, 0.1, 1\}$
direction adaptation factor	β	$\{0.0, 0.4, 0.9\}$
update step adaptation	α	$\{1, 1.66\}$
maximum step size	s_{max}	$\{3.16\ (\approx 10^{0.5})\}$

LABPAL (SGD and NSGD):

hyperparameter	symbol	values
step size adaptation	α	$\{1.0, 1.8, 1.9\}$
momentum		$\{0, 0.4, 0.9\}$
SGD steps	n_{SGD}	$\{1000, 5000\}$
approximation batch size	$\lvert\mathbb{B}_a\rvert$	$\{640, 1280\}$
batch size schedule	$k(t)$	$\begin{cases} 1, & \text{if } t \leq \lfloor t_{\max} \cdot 0.5 \rfloor \\ 2, & \text{elif } t \leq \lfloor t_{\max} \cdot 0.75 \rfloor \\ 4, & \text{elif } t > \lfloor t_{\max} \cdot 0.75 \rfloor \end{cases}$, where $t_{\max}$ is the amount of training steps
measure points for $l_{\mathbb{B}_a,t}$		$\{(0, 0.0001, 0.01)\}$

GOLSI:

hyperparameter	symbol	values
initial step size	μ	$\{0.001, 0.01, 0.1, 1.0\}$
momentum	β	$\{0, 0.4, 0.9\}$
step size scaling parameter	η	$\{0.2, 2.0\}$
modified Wolfe condition parameter	$c2$	$\{0.9, 0.99\}$

PLS:

hyperparameter	symbol	values
first Wolfe condition parameter	c_1	$\{0.3, 0.4\}$
acceptance threshold for the wolfe probability	cW	$\{0.1, 0.2\}$
initial step size	α_0	$\{0.001, 0.01, 0.1, 1.0\}$
momentum	β	$\{0, 0.4, 0.9\}$

SLS:

hyperparameter	symbol	values
initial step size	μ	$\{0.001, 0.01, 0.1, 1.0\}$
step size decay	β	$\{0.9, 0.99\}$
step size reset	γ	$\{2.0\}$
Armijo constant	c	$\{0.1, 0.01\}$
maximum step size	$\mu_{\max}$	$\{10.0\}$

For SLS no momentum term is considered since (Vaswani *et al.*, 2019) already showed SLS variants using momentum like acceleration methods to be non-beneficial.

C.4.2 Further Hyperparameter Sensitivity Analysis

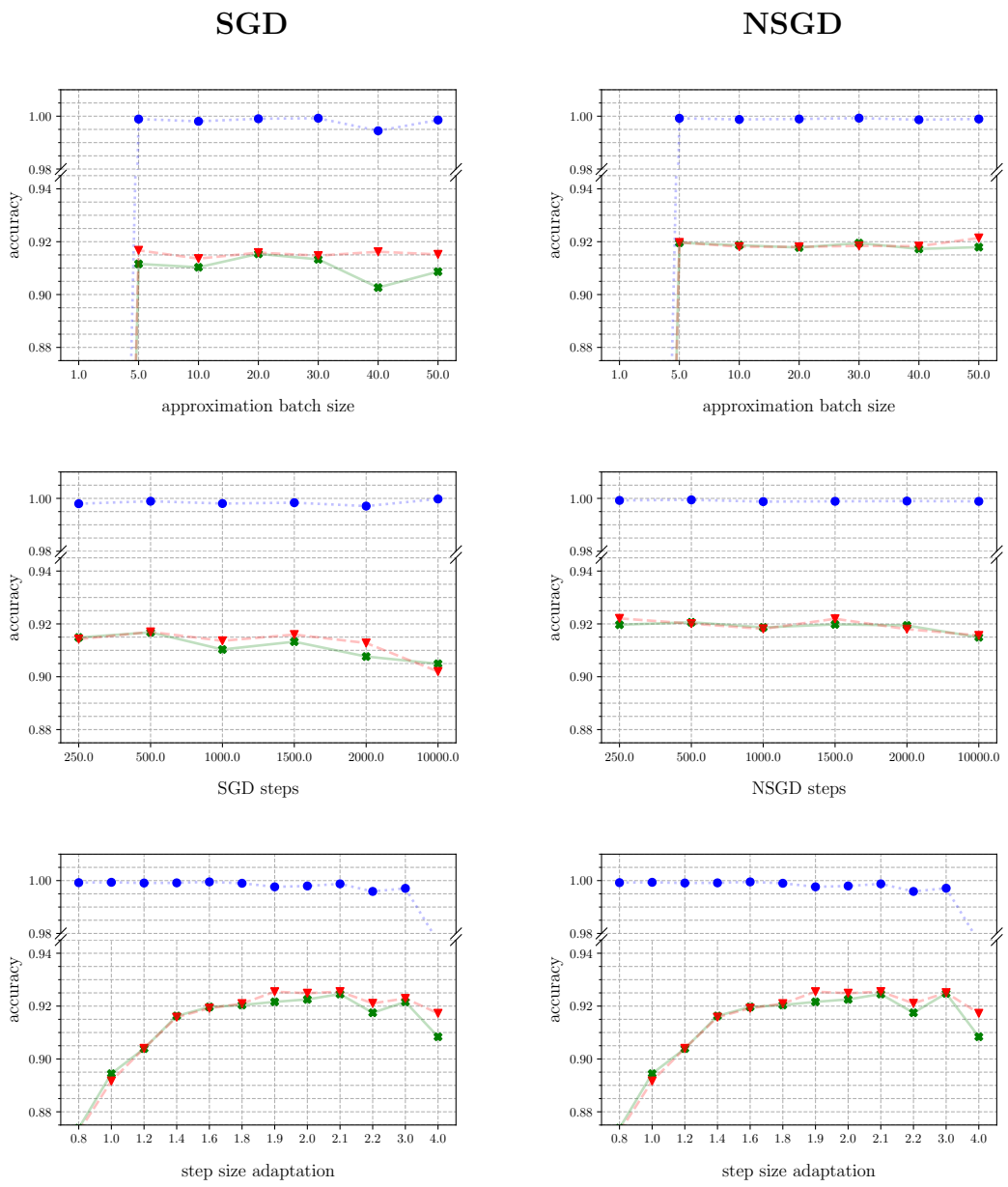


Figure continues on next page.

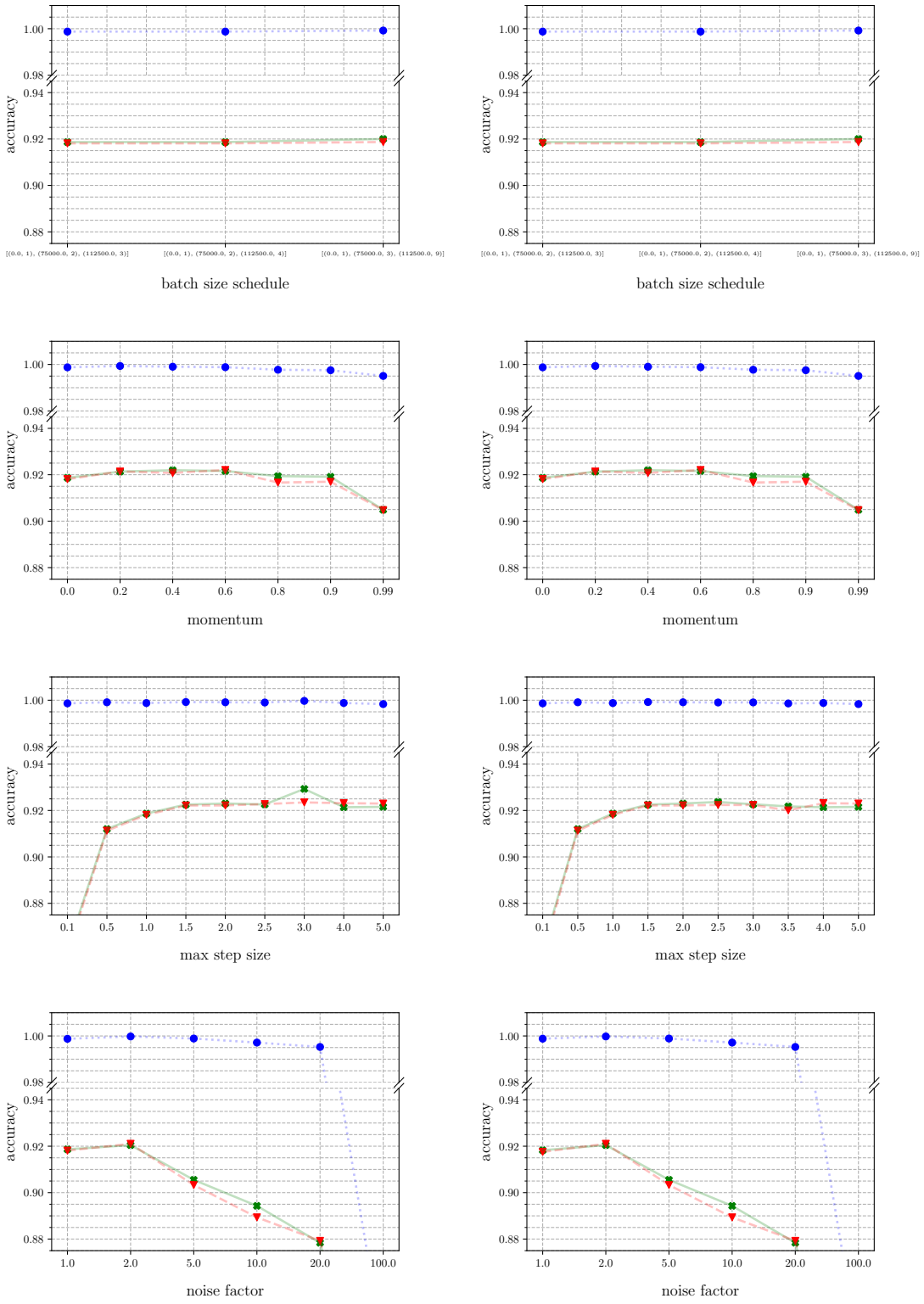


Figure C.4: Sensitivity analysis of parameters of **LABPAL&SGD** and **LABPAL&NSGD** on a DenseNet-121. The default parameters are: approximation batch size $\mathbb{B}_a = 1280$, SGD steps $s = 1000$, step size adaptation $\alpha = 1.8$, batch size schedule $k = (0{:}1,\ 75000{:}2,\ 112500{:}4)$, momentum $\beta = 0$, maximal step size $= 1.0$, noise-factor $\epsilon = 1$. For the approximation batch size (row one) the factor 128 is multiplied on the x axis.

Symbols

$\odot$	Hadamard product a.k.a. point wise product
x	scalar
$\boldsymbol{x}$	vector
$\hat{\boldsymbol{x}}$	normalized vector a.k.a. unit vector
$\boldsymbol{X}$	matrix
$\mathbb{X}$	set

t	number of the current parameter update step
λ	learning rate (factor multiplied with the gradient $\hat{\boldsymbol{g}}$)
s	update step size (factor multiplied with the normalized gradient $\boldsymbol{g}$)
s_{upd}	performed update step
s_{min}	update step to the minimum of the loss along a line
$\mathbb{R}$	set of real numbers
$\mathbb{Q}$	set of rational numbers
$\mathbb{N}$	set of integers
$\mathbb{B}$	mini-batch
$\mathbb{D}$	dataset
$\mathcal{L}$	full-batch loss
L	loss function e.g. cross entropy (see Section 2.2)
$\mathcal{L}_{\mathbb{B}}$	mini-batch loss (see Section 2.6.1)
l	full-batch loss along a line in mini-batch gradient direction (see Section 2.6.1). In Chapter 4 to 6 the normalized gradient direction is used.
$l_{\mathbb{B}}$	mini-batch loss along a line in mini-batch gradient direction (see Section 2.6.1). In Chapter 4 to 6 the normalized gradient direction is used.
$\boldsymbol{\theta}$	parameters to optimize
$\boldsymbol{g}_{\mathbb{B}}$	gradient of the mini-batch loss with respect to $\boldsymbol{\theta}$
$\hat{\boldsymbol{g}}$	normalized gradient (unit gradient)

Acronyms

ADAM	ADAptive Momentum (Kingma and Ba, 2015)
ALI-G	Adaptive Learning rates for Interpolation with Gradients (Berrada *et al.*, 2020)
BMBF	Bubdesministerium für Bildung und Forschung (German Federal Ministry of Education and Research)
CE	Cross Entropy
CNN	convolution neural network (LeCun *et al.*, 1999)
COCOB	COntinuous COin Betting (Orabona and Tommasi, 2017)
DenseNet	dense neural network (a CNN) (Huang *et al.*, 2017)
DNN	deep neural network
ERM	Empirical Risk Minimization
GD	Gradient Descent
GOLSI	Gradient Only Line Search which is Inexact (Kafka and Wilke, 2019)
GP	Gaussian Process
GPU	Graphics Processing Unit
i.i.d.	independent and identically distributed
LABPAL	Large Batch Parabolic Approximation Line search
MobileNet	mobile neural network (a CNN) (Sandler *et al.*, 2018)
NN	neural network
NSGD	Normalized stochastic gradient descent
PAL	Parabolic Approximation Line search
PLS	Probabilistic Line Search (Mahsereci and Hennig, 2015)
ResNet	residual neural network (a CNN) (He *et al.*, 2016)
RMSP	Root Mean Square Propagation (Tieleman and Hinton, 2012)
RNN	recurrent neural network (Hochreiter and Schmidhuber, 1997)
SGD	Stochastic Gradient Descent (Robbins and Monro, 1951)
SGD-HD	Stochastic Hyper gradient Descent (Baydin *et al.*, 2018)
SLS	Stochastic Armijo backtracking Line Search (Vaswani *et al.*, 2019)

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., *et al.* (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. Software available at `https://www.tensorflow.org/install`.

Absil, P.-A., Mahony, R., and Andrews, B. (2005). Convergence of the iterates of descent methods for analytic cost functions. *SIAM Journal on Optimization*, 16(2), 531–547.

Arnold, S. M. R., Manzagol, P., Babanezhad, R., Mitliagkas, I., and Roux, N. L. (2019). Reducing the variance in online optimization by transporting past gradients. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5392–5403.

Balles, L., Romero, J., and Hennig, P. (2017). Coupling Adaptive Batch Sizes with Learning Rates. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press.

Baydin, A. G., Cornish, R., Martínez-Rubio, D., Schmidt, M., and Wood, F. (2018). Online Learning Rate Adaptation with Hypergradient Descent. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Berahas, A. S., Jahani, M., Richtárik, P., and Takáč, M. (2021). Quasi-Newton methods for machine learning: forget the past, just sample. *Optimization Methods and Software*, pages 1–37.

Berrada, L., Zisserman, A., and Kumar, M. P. (2020). Training Neural Networks for and by Interpolation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 799–809. PMLR.

Bertsekas, D. P. and Tsitsiklis, J. N. (2000). Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3), 627–642.

Bolz, V., Rueß, J., and Zell, A. (2019). Power Flow Approximation Based on Graph Convolutional Networks. In *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16-19, 2019*, pages 1679–1686. IEEE.

Botev, A., Ritter, H., and Barber, D. (2017). Practical Gauss-Newton Optimisation for Deep Learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 557–565. PMLR.

Butz, M. V., Menge, T., Humaidan, D., and Otte, S. (2019a). Inferring Event-Predictive Goal-Directed Object Manipulations in REPRISE. In *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I*, volume 11727 of *Lecture Notes in Computer Science*, pages 639–653. Springer.

Butz, M. V., Bilkey, D. K., Humaidan, D., Knott, A., and Otte, S. (2019b). Learning, planning, and control in a monolithic neural event inference architecture. *Neural Networks*, 117, 135–144.

Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1-2), 57–83.

Cauchy, A. *et al.* (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847), 536–538.

Chae, Y. and Wilke, D. N. (2019). Empirical study towards understanding line search approximations for training neural networks. *arXiv preprint arXiv:1909.06893*.

Chuong, D (2008). Gaussian processes `https://cs229.stanford.edu/section/cs229-gaussian_processes.pdf`. Accessed: 2022-10-07.

Dai, Y.-H. and Yuan, Y. (1999). A nonlinear conjugate gradient method with a strong global convergence property. *SIAM Journal on optimization*, 10(1), 177–182.

Dangel, F., Harmeling, S., and Hennig, P. (2020). Modular Block-diagonal Curvature Approximations for Feedforward Architectures. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 799–808. PMLR.

De, S., Yadav, A., Jacobs, D., and Goldstein, T. (2016). Big batch SGD: Automated inference using adaptive batch sizes. *arXiv preprint arXiv:1610.05792*.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

Docker, Inc. (2013). Docker. `https://www.docker.com/`. Accessed: 2022-10-07.

Dodge, S. F. and Karam, L. J. (2017). A Study and Comparison of Human and Deep Learning Recognition Performance under Visual Distortions. In *26th International Conference on Computer Communication and Networks, ICCCN 2017, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*, pages 1–7. IEEE.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. A. (2018). Essentially No Barriers in Neural Network Energy Landscape. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1308–1317. PMLR.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of machine learning research*, 12, 2121–2159.

Elkersh, H., Mutschler, M., and Laube, K. (2020). Hypergradient Descent. *Student Research Project at the University Of Tuebingen*. For access, contact maximus.mutschler@uni-tuebingen.de.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1406–1415. PMLR.

Fahlman, S. E. *et al.* (1988). *An empirical study of learning speed in back-propagation networks*. Carnegie Mellon University, Computer Science Department Pittsburgh, PA, USA.

Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients. *The computer journal*, 7(2), 149–154.

Fort, S. and Jastrzebski, S. (2019). Large Scale Structure of Neural Network Loss Landscapes. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6706–6714.

Gao, Y., Tebbe, J., and Zell, A. (2021). Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis. In *Artificial Neural Networks and Machine Learning - ICANN 2021 - 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14-17, 2021, Proceedings, Part I*, volume 12891 of *Lecture Notes in Computer Science*, pages 379–390. Springer.

Gardiner, C. W. (1985). *Handbook of stochastic methods - for physics, chemistry and the natural sciences, Second Edition*. Springer series in synergetics. Springer. ISBN-13: 978-3540616344.

Gencoglu, O., van Gils, M., Guldogan, E., Morikawa, C., Süzen, M., Gruber, M., Leinonen, J., and Huttunen, H. (2019). HARK Side of Deep Learning– From Grad Student Descent to Automated Machine Learning. *arXiv preprint arXiv:1904.07633*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*. MIT press Cambridge. ISBN-13: 978-0262035613.

Goodfellow, I. J. and Vinyals, O. (2015). Qualitatively characterizing neural network optimization problems. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Hao, Z., Jiang, Y., Yu, H., and Chiang, H. (2021). Adaptive Learning Rate and Momentum for Training Deep Neural Networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part III*, volume 12977 of *Lecture Notes in Computer Science*, pages 381–396. Springer.

He, H., Huang, G., and Yuan, Y. (2019). Asymmetric Valleys: Beyond Sharp and Flat Local Minima. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2549–2560.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49, 409–435.

Hille, T. and Mutschler, M. (2020). Adaptive Approximation niedrig dimensionaler Mannigfaltigkeiten zwischen lokalen Minima in den Fehlerlandschaften von Neuronalen Netzen. *Master Thesis University of Tuebingen*. For access, contact maximus.mutschler@uni-tuebingen.de.

Hobbhahn, M., Butz, M. V., Fabi, S., and Otte, S. (2020). Sequence Classification using Ensembles of Recurrent Generative Expert Modules. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pages 333–338.

Hochreiter, S. and Schmidhuber, J. (1994). Simplifying Neural Nets by Discovering Flat Minima. In *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 529–536. MIT Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural computation*, 9(8), 1735–1780.

Howard, A., Pang, R., Adam, H., Le, Q. V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y. (2019). Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1314–1324. IEEE.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2018). Averaging Weights Leads to Wider Optima and Better Generalization. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 876–885. AUAI Press.

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. (2017). Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*.

Jastrzebski, S., Kenton, Z., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. J. (2019). On the Relation Between the Sharpest Directions of DNN Loss and the SGD Step Length. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Kafka, D. and Wilke, D. (2019). Gradient-only line searches: An alternative to probabilistic line searches. *arXiv preprint arXiv:1903.09383*.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., and Aila, T. (2021). Alias-Free Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 852–863.

Kawaguchi, K. (2016). Deep Learning without Poor Local Minima. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 586–594.

Kawaguchi, K. and Kaelbling, L. P. (2020). Elimination of All Bad Local Minima in Deep Learning. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 853–863. PMLR.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Kiefer, B., Messmer, M., and Zell, A. (2021). Diminishing Domain Bias by Leveraging Domain Labels in Object Detection on UAVs. In *20th International Conference on Advanced Robotics, ICAR 2021, Ljubljana, Slovenia, December 6-10, 2021*, pages 523–530. IEEE.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Krizhevsky, A., Hinton, G., *et al.* (2009). Learning multiple layers of features from tiny images, `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf`.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

Lancewicki, T. and Kopru, S. (2020). Automatic and Simultaneous Adjustment of Learning Rate and Momentum for Stochastic Gradient-based Optimization Methods. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3127–3131. IEEE.

Lange, M., Schweinfurth, F., and Schilling, A. (2019). DLD: A Deep Learning Based Line Descriptor for Line Feature Matching. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 5910–5915. IEEE.

Lange, M., Raisch, C., and Schilling, A. (2020). WLD: A Wavelet and Learning based Line Descriptor for Line Feature Matching. In *25th International Symposium on Vision, Modeling and Visualization, VMV 2020, Tübingen, Germany, September 28 - October 1, 2020*, pages 39–46. Eurographics Association.

Laube, K. A. (2021). The UniNAS framework: combining modules in arbitrarily complex configurations with argument trees. *arXiv preprint arXiv:2112.01796*.

Laube, K. A. and Zell, A. (2019a). Prune and Replace NAS. In *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16-19, 2019*, pages 915–921. IEEE.

Laube, K. A. and Zell, A. (2019b). ShuffleNASNets: Efficient CNN models through modified Efficient Neural Architecture Search. In *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*, pages 1–6. IEEE.

Laube, K. A. and Zell, A. (2021). Inter-choice dependent super-network weights. *arXiv preprint arXiv:2104.11522*.

Laube, K. A., Mutschler, M., and Zell, A. (2022). What to expect of hardware metric predictors in NAS. In *First Conference on Automated Machine Learning, AutoML Conference 2022, Baltimore, US, July 25-27, 2022*.

LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In *Shape, Contour and Grouping in Computer Vision*, volume 1681 of *Lecture Notes in Computer Science*, page 319. Springer.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6391–6401.

Li, Q., Tai, C., and E, W. (2017). Stochastic Modified Equations and Adaptive Stochastic Gradient Algorithms. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2101–2110. PMLR.

Li, X., Gu, Q., Zhou, Y., Chen, T., and Banerjee, A. (2020). Hessian based analysis of SGD for Deep Nets: Dynamics and Generalization. In *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020, Cincinnati, Ohio, USA, May 7-9, 2020*, pages 190–198. SIAM.

Liu, Y., Bernstein, J., Meister, M., and Yue, Y. (2021). Learning by Turning: Neural Architecture Aware Optimisation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6748–6758. PMLR.

Loshchilov, I. and Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Luenberger, D. G., Ye, Y., *et al.* (1984). *Linear and nonlinear programming*, volume 2. Springer. ISBN-13: 978-0201157949.

Lukas Balles (2017). Tensorflow implementation of Probabilistic Line Search. `https://github.com/ProbabilisticNumerics/probabilistic_line_search/commit/a83dfb0`. Accessed: 2022-10-07.

Luo, L., Xiong, Y., Liu, Y., and Sun, X. (2019). Adaptive Gradient Methods with Dynamic Bound of Learning Rate. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Lyapunov, A. M. (1992). The general problem of the stability of motion. *International journal of control*, 55(3), 531–534.

Mahsereci, M. and Hennig, P. (2015). Probabilistic Line Searches for Stochastic Optimization. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 181–189.

Martens, J. and Grosse, R. B. (2015). Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2408–2417. JMLR.org.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. (2018). An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*.

Moré, J. J. and Thuente, D. J. (1994). Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw.*, 20(3), 286–307.

Mutschler, M. and Zell, A. (2020a). Parabolic Approximation Line Search for DNNs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mutschler, M. and Zell, A. (2020b). A straightforward line search approach on the expected empirical loss for stochastic deep learning problems. *arXiv preprint arXiv:2010.00921*.

Mutschler, M. and Zell, A. (2021). Empirically Explaining SGD from a Line Search Perspective. In *Artificial Neural Networks and Machine Learning - ICANN 2021 - 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14-17, 2021, Proceedings, Part II*, volume 12892 of *Lecture Notes in Computer Science*, pages 459–471. Springer.

Mutschler, M., Laube, K., and Zell, A. (2021). Using a one dimensional parabolic model of the full-batch loss to estimate learning rates during training. *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021 Optimization Workshop, NeurIPS 2021 Optimization Workshop, December 6-14, 2021, virtual*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. *Advances in Neural Information Processing Systems 24: Workshop on Deep Learning and Unsupervised Feature Learning, NeurIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning, December 12-17, 2011, Granada, Spain*.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer series in operations research. Springer, 2nd ed edition. ISBN-13: 9780387303031.

NVIDIA (2007). CUDA. `https://developer.nvidia.com/cuda-toolkit`. Accessed: 2022-10-07.

NVIDIA (2015). cuDNN. `https://developer.nvidia.com/cudnn`. Accessed: 2022-10-07.

NVIDIA (2021). NVIDIA GeForce RTX 3080 TI technical details. `https://cdn-reichelt.de/documents/datenblatt/E810/MSI_V389-208R_DB-EN.pdf`. Accessed: 2022-02-27.

Orabona, F. and Tommasi, T. (2017). Training Deep Networks without Learning Rates Through Coin Betting. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2160–2170.

Otte, S., Rubisch, P., and Butz, M. V. (2019a). Gradient-Based Learning of Compositional Dynamics with Modular RNNs. In *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I*, volume 11727 of *Lecture Notes in Computer Science*, pages 484–496. Springer.

Otte, S., Stoll, J., and Butz, M. V. (2019b). Incorporating Adaptive RNN-Based Action Inference and Sensory Perception. In *Artificial Neural Networks and Machine Learning - ICANN 2019: Text and Time Series - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part IV*, volume 11730 of *Lecture Notes in Computer Science*, pages 543–555. Springer.

Papoulis, A. (1991). *Random Variables and Stochastic Processes*. McGraw-Hill Publishing Company, 3rd ed edition. ISBN-13: 9780070484771.

Paquette, C. and Scheinberg, K. (2018). A stochastic line search method with convergence rate analysis. *arXiv preprint arXiv:1807.07994*.

Paren, A., Poudel, R., and Kumar, M. P. (2021). Faking Interpolation Until You Make It. *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021 Optimization Workshop, NeurIPS 2021 Optimization Workshop, December 6-14, 2021, virtual*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Patrick, B. (1995). *Probability and measure*. John Wiley and Sons. ISBN-13: 9780471007104.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1–17.

Polyak, B. T. (1969). Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3), 14–29.

Rahim, R., Shamsafar, F., and Zell, A. (2021). Separable Convolutions for Optimizing 3D Stereo Networks. In *2021 IEEE International Conference on Image Processing, ICIP 2021, Anchorage, AK, USA, September 19-22, 2021*, pages 3208–3212. IEEE.

Ramamurthy, V. and Duffy, N. (2017). L-SR1: A Second Order Optimization Method for Deep Learning, `https://openreview.net/forum?id=By1snw5gl`.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the Convergence of Adam and Beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Ribière, G. and Polak, E. (1969). Note sur la convergence de directions conjugées. *Rev. Francaise Informat Recherche Opertionelle*, 16, 35–43.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22, 400–407.

Rolínek, M. and Martius, G. (2018). L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6434–6444.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society.

Sanzenbacher, P., Mescheder, L., and Geiger, A. (2020). Learning Neural Light Transport. *arXiv preprint arXiv:2006.03427*.

Schmidt, M., Roux, N. L., and Bach, F. R. (2011). Convergence Rates of Inexact Proximal-Gradient Methods for Convex Optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1458–1466.

Schmidt, M., Babanezhad, R., Ahmed, M. O., Defazio, A., Clifton, A., and Sarkar, A. (2015). Non-Uniform Stochastic Average Gradient Method for Training Conditional Random Fields. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, volume 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Schmidt, M., Roux, N. L., and Bach, F. R. (2017). Minimizing finite sums with the stochastic average gradient. *Math. Program.*, 162(1-2), 83–112.

Schraudolph, N. N., Yu, J., and Günter, S. (2007). A Stochastic Quasi-Newton Method for Online Convex Optimization. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007*, volume 2 of *JMLR Proceedings*, pages 436–443. JMLR.org.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press. ISBN-13: 978-1-10-705713-5.

Shamsafar, F., Woerz, S., Rahim, R., and Zell, A. (2022). MobileStereoNet: Towards Lightweight Deep Networks for Stereo Matching. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 677–686. IEEE.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman,

S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., *et al.* (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Smith, L. N. (2017). Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*, pages 464–472. IEEE Computer Society.

Smith, S. L. and Le, Q. V. (2018). A Bayesian Perspective on Generalization and Stochastic Gradient Descent. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Smith, S. L., Kindermans, P., Ying, C., and Le, Q. V. (2018). Don't Decay the Learning Rate, Increase the Batch Size. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1139–1147. JMLR.org.

Sylabs (2015). Singularity. `https://sylabs.io/singularity/`. Accessed: 2022-10-07.

Tan, M. and Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California,*

USA, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR.

Tebbe, J., Klamt, L., Gao, Y., and Zell, A. (2020). Spin Detection in Robotic Table Tennis. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 9694–9700. IEEE.

Tebbe, J., Krauch, L., Gao, Y., and Zell, A. (2021). Sample-efficient Reinforcement Learning in Robotic Table Tennis. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 4171–4178. IEEE.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning. *University of Toronto, Technical Report*, 6.

Truong, T. T. and Nguyen, T. H. (2018). Backtracking gradient descent method for general C1 functions, with applications to Deep Learning. *arXiv preprint arXiv:1808.05160*.

ul Moqeet Riaz, H., Benbarka, N., and Zell, A. (2020). FourierNet: Compact Mask Representation for Instance Segmentation Using Differentiable Shape Decoders. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 7833–7840. IEEE.

ul Moqeet Riaz, H., Benbarka, N., Höfer, T., and Zell, A. (2022). FourierMask: Instance Segmentation Using Fourier Mapping in Implicit Neural Networks. In *Image Analysis and Processing - ICIAP 2022 - 21st International Conference, Lecce, Italy, May 23-27, 2022, Proceedings, Part II*, volume 13232 of *Lecture Notes in Computer Science*, pages 587–598. Springer.

University of Tuebingen (2021). Training Center for Machine Learning Cluster. `https://uni-tuebingen.de/de/126096`. Accessed: 2022-10-07.

Varga, L. A. and Zell, A. (2021). Tackling the Background Bias in Sparse Object Detection via Cropped Windows. In *IEEE/CVF International Conference on Computer Vision Workshops, ICCVW 2021, Montreal, BC, Canada, October 11-17, 2021*, pages 2768–2777. IEEE.

Varga, L. A., Makowski, J., and Zell, A. (2021). Measuring the Ripeness of Fruit with Hyperspectral Imaging and Deep Learning. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pages 1–8. IEEE.

Varga, L. A., Kiefer, B., Messmer, M., and Zell, A. (2022a). SeaDronesSee: A Maritime Benchmark for Detecting Humans in Open Water. In *IEEE/CVF*

Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022, pages 3686–3696. IEEE.

Varga, L. A., Kiefer, B., Messmer, M., and Zell, A. (2022b). SeaDronesSee: A Maritime Benchmark for Detecting Humans in Open Water. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 3686–3696. IEEE.

Vaswani, S., Mishkin, A., Laradji, I. H., Schmidt, M., Gidel, G., and Lacoste-Julien, S. (2019). Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3727–3740.

von Bachmann, P. and Mutschler, M. (2020). Forcing Deep Neural Networks to explore the Loss Landscape. *Bachelor Thesis University Of Tuebingen*. For access, contact maximus.mutschler@uni-tuebingen.de.

Xing, C., Arpit, D., Tsirigotis, C., and Bengio, Y. (2018). A walk with sgd. *arXiv preprint arXiv:1802.08770*.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). SLURM: Simple Linux Utility for Resource Management. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer.

Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. (2020). A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, 8, 58443–58469.