OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Using a one dimensional parabolic model of the full-batch loss to estimate learning rates during training

Maximus Mutschler & Andreas Zell
University of Tübingen
Sand 1, D-72076 Tübingen, Germany
{maximus.mutschler, andreas.zell}@uni-tuebingen.de

Abstract

A fundamental challenge in Deep Learning is to find optimal step sizes for stochastic gradient descent automatically. In traditional optimization, line searches are a commonly used method to determine step sizes. One problem in Deep Learning is that finding appropriate step sizes on the full-batch loss is unfeasibly expensive. Therefore, classical line search approaches, designed for losses without inherent noise, are usually not applicable. Recent empirical findings suggest, inter alia, that the full-batch loss behaves locally parabolically in the direction of noisy update step directions. Furthermore, the trend of the optimal update step size changes slowly. By exploiting these and more findings, this work introduces a line-search method that approximates the full-batch loss with a parabola estimated over several mini-batches. Learning rates are derived from such parabolas during training. In the experiments conducted, our approach is on par with SGD with Momentum tuned with a piece-wise constant learning rate schedule and often outperforms other line search approaches for Deep Learning across models, datasets, and batch sizes on validation and test accuracy. In addition, our approach is the first line search approach for Deep Learning that samples a larger batch size over multiple inferences to still work in low-batch scenarios.

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 hyper-parameter tuning. Therefore, although well-working hand-designed schedules such as *piece-wise constant learning rate* or *cosine decay* exist (see [20, 32]), it is desired to infer problem depended and better learning rate schedules automatically. This work builds on recent empirical findings; among those are that the full-batch loss tends to have a simple parabolic shape in SGD update step direction [25, 26] and that the trend of the optimal update step changes slowly [26]. Exploiting the found observations, we introduce a line search approach, which approximates the full-batch loss along lines in SGD update step direction occasionally with parabolas sampled over multiple mini-batches.

The major contribution of this work 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 ([15, 22, 25, 35]) across models, 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 contribution is that we are the first to analyze how the considered line searches perform under high gradient 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 gradient noise by iteratively sampling larger batch sizes over several inferences. For related work please see App. A.

2. Our approach: Large-Batch Parabolic approximation line search (LabPal)

2.1. Mathematical Foundations

In this subsection, we introduce the mathematical background relevant for line searches and challenges that must be solved in order to perform line searches in Deep Learning.
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:

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

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:

$$\mathcal{L}_\mathbb{B} \; : \; \mathbb{R}^n \to \mathbb{R}, \; \boldsymbol{\theta} \mapsto \frac{1}{|\mathbb{B}|} \sum_{d \in \mathbb{B} \subset \mathbb{D}} L_d(\boldsymbol{\theta}), \tag{2}$$

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

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:

$$l_{\mathbb{B},t} \; : \; \mathbb{R} \to \mathbb{R}, \; s \mapsto \mathcal{L}_\mathbb{B}(\boldsymbol{\theta}_t + s \cdot -\hat{\boldsymbol{g}}_{\mathbb{B}_t,t}), \tag{3}$$

where s is the step size along the line. The corresponding full-batch loss along the same line is given by: $l_t \; : \; \mathbb{R} \to \mathbb{R}, \; s \mapsto \mathcal{L}(\boldsymbol{\theta}_t + s \cdot -\hat{\boldsymbol{g}}_{\mathbb{B}_t,t})$.
Let the step size to the first encountered minimum of l_t be $s_{\text{min},t}$.

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 [7]. The simplest form is the direction of steepest descent [21]. 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.

2.2. Deriving the algorithm

In the following, we derive our line search approach on the full-batch loss by iteratively exploiting empirically found observations of [26] and solving the challenges for a line search on the full-batch loss (see Section 2.1). Given default values are inferred from a detailed hyperparameter analysis

(App. C) **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 $\mathcal{L}$* (App. Fig. 4 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 App. Fig. 3).
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 with multiple inferences. Thus, the approximation batch size $\mathbb{B}_a$, is significantly larger as 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 as $\mathbb{B}_i$. 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 l_t do not change locally significantly.* (Figure 4 left, red curve).

Algorithm 1 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 parenthesis. For LABPAL&NSGD SGD is replaced with NSGD, and the update step is measured instead of the learning rate.

```
Input: Hyperparameters:
    - initial parameters $\theta_0$
    - approximation batch size $|\mathbb{B}_a|$ (1280)
    - inference batch size $|\mathbb{B}_i|$ (128)
    - SGD steps $n_{\text{SGD}}$ (1000),    # or NSGD steps
    - step size adaptation $\alpha > 1$ (1.8)
    - training steps $t_{max}$ (150000)
    - batch size schedule $k(t) = \begin{cases} 1, & \text{if } t \leq \lfloor t_{max} \cdot 0.5 \rfloor \\ 2, & \text{elif } t \leq \lfloor t_{max} \cdot 0.75 \rfloor \\ 4, & \text{elif } t > \lfloor t_{max} \cdot 0.75 \rfloor \end{cases}$
 1: # Variables have global scope
 2: sampledBatchSize ← 0
 3: performedSGDsteps ← 0
 4: learningRate ← 0
 5: $\theta \leftarrow \theta_0$
 6: state ← 'line search'
 7: direction ← current batch gradient
 8: $t \leftarrow 0$
 9: while $t < t_{max}$ do
10:     if state is 'line search' then
11:         PERFORM_LINE_SEARCH_STEP()
12:     end if
13:     if state is 'SGDTraining' then
14:         PERFROM_LARGE_BATCH_SGD_STEP()
15:     end if
16: end while
17: return $\theta$
18: procedure PERFORM_LINE_SEARCH_STEP()
19:     if sampledBatchSize $< |\mathbb{B}_a|$ then
20:         update estimate $\hat{\mathcal{L}}$ of $\mathcal{L}$ with
            over multiple inferences sampled $\mathcal{L}_{\mathbb{B}_t,t}$ with
            $|\mathbb{B}_t| = k(t) \cdot |\mathbb{B}_i|$)
21:         increase sampledBatchSize by $|\mathbb{B}_t|$ and $t$ by $k(t)$
22:     else
23:         learningRate ← perform parabolic approximation
            with 3 values of $\hat{\mathcal{L}}$ along the search direction and es-
            timate the learning rate.
24:         learningRate ← learningRate $\cdot\alpha$
25:         set sampledBatchSize and performedSGDsteps to 0
26:         state ← 'SGDTraining'
27:     end if
28: end procedure
29:
30: procedure PERFROM_LARGE_BATCH_SGD_STEP()
31:     if performedSGDsteps $< n_{\text{SGD}}$ then
32:         $\theta$ ← perform SGD update with learningRate and over
            multiple inferences sampled $\mathcal{L}_{\mathbb{B}_t,t}$
            with $|\mathbb{B}_t| = k(t) \cdot |\mathbb{B}_i|$)
33:         increase $t$ by $k(t)$
34:         increase performedSGDsteps by 1
35:     else
36:         direction ← current batch gradient
37:         state ← 'line search'
38:     end if
39: end procedure
```

Observation 4: *$s_{min,t}$ and the direction defining batch's $||\boldsymbol{g}_{\mathbb{B}_t,t}||$ are almost proportional during training.* (App. Fig. 4 right).
Derivation Step 3: Using measurements of $l_{\mathbb{B}_a,t}$ to approximate l_t with a parabola is by far to 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 2.1). Now, we will overcome the second challenge; approximating the full batch loss gradient for each weight update step:
For this, we revisit [34] who approximates the magnitude of random gradient fluctuations, that appear if training with a mini-batch gradient, by the *noise scale* $\nu \in \mathbb{R}$:

$$\nu \approx {}^{(\lambda|\mathbb{D}|)}\!/_{|\mathbb{B}|}, \tag{4}$$

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* (App. Fig. 4 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 [7] to determine appropriate batch sizes, we stick to a simple piecewise constant batch size schedule doubling the batch size after two and after three-quarters of the training.
Observation 6: *On a global perspective a 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 $\alpha \in]1,2[$:

$$\lambda_{\text{new}} = \alpha\lambda = {}^{\alpha s_{\text{min},t}}\!/_{||\boldsymbol{g}_{\mathbb{B},t}||} \quad \text{or} \quad s_{\text{upd}} = \alpha s_{\text{min},t} \tag{5}$$

Note that under out parabolic property, the first wolfe condition w_1, which is commonly used for line searches, simply relates to α: $w_1 = -\frac{\alpha}{2} + 1$ (see Appendix G).

Combining all derivations leads to our line search named *large-batch parabolic approximation line search* (LABPAL), which is given in Algorithm 1. 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 $||\boldsymbol{g}||$.

3. 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**

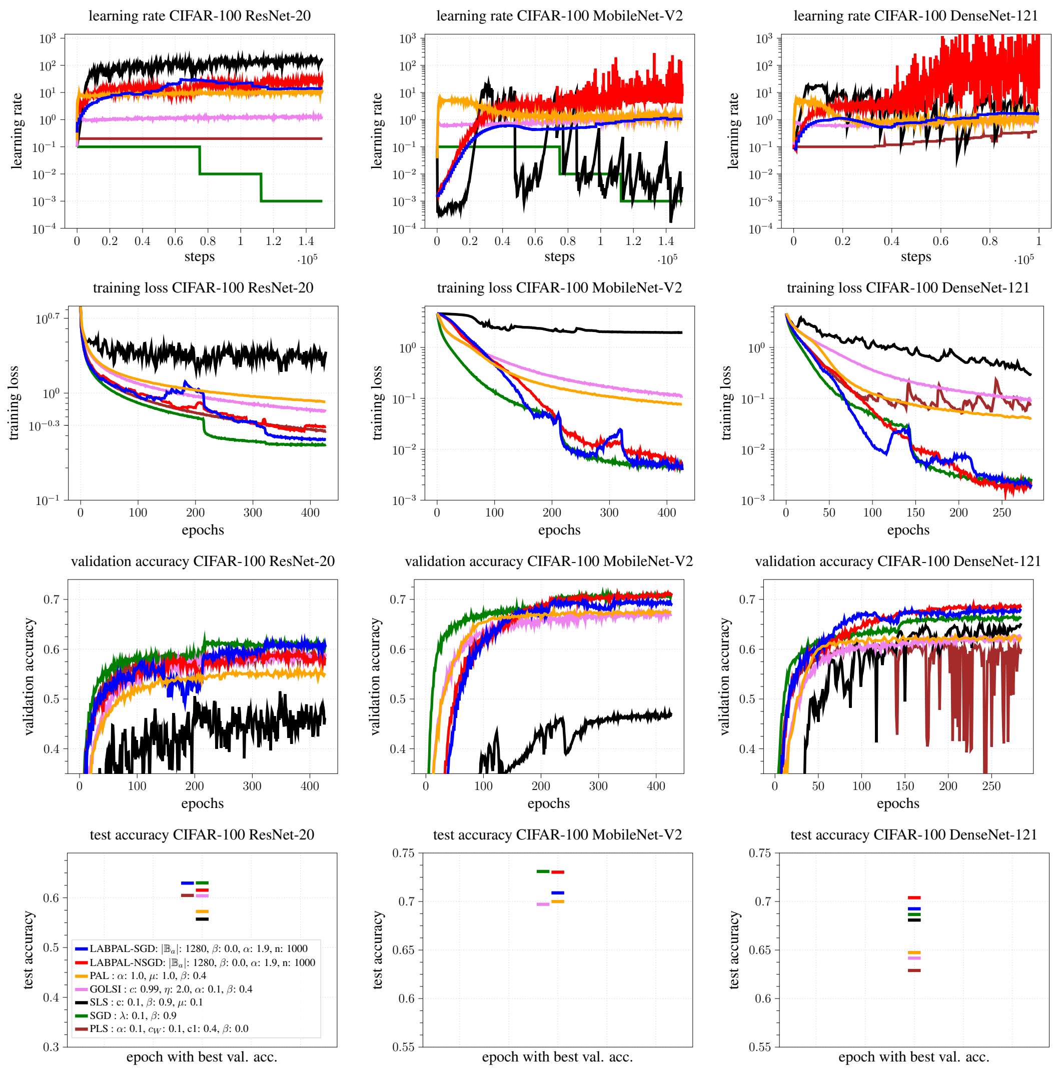

Figure 1: Performance comparison on **CIFAR-100** of our approach LABPAL in the SGD and NSGD variants against several line searches and SGD with momentum. Optimal hyperparameters for CIFAR-10 ResNet-20 found with a grid search are reused (Appendix H.1). Here, our approaches surpass the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics. Results for CIFAR-10 and SVHN are given in App. Figures 8, and 9. The batch-size used is 128. Due to a too high memory consumption we could not run PLS on MobileNet-V2.

step **as the sampling of a new input 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 [29], Adagrad [9] or Adam [16]. Results are averaged over 3 runs. A performance analysis on ground truth full-batch loss and proof of concept is found in App B.1. For a hyperparameter sensitivity analysis see Appndix C.

3.1. Performance comparison to SGD and to other line search approaches

We compare the SGD and NSGD variants of our approach against the line search approaches PLS [22], GOLSI [15], PAL [25], SLS [35] and against SGD with Momentum [29] tuned by a piece-wise constant learning rate schedule (see App. Sec. A). We compare the optimizers on ResNet-20 [12], DenseNet-40 [13], and MobileNetV2 [31] trained on CIFAR-10 [17], CIFAR-100 [18], and SVHN [27]. We concentrate on classification problems since the empirical observations are inferred from a classification task. For each optimizer, we perform a comprehensive hyper-parameter grid search on ResNet-20 trained on CIFAR-10 (see Appendix H.1). The best performing hyper-parameters 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, as is usually the case in practice. 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 H.

Figure 1 as well as App. Fig. 8, 9 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 hyper-parameter tuning for LABPAL is not needed to achieve good results across several models and datasets. However, this also is 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 [26], 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 1, 8, 9 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 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 Appendix F.

3.2. Adaptation to varying gradient noise

Recent literature, E.g., [25], [35], [15] show 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 sizes leading to varying gradient noise*. E.g., Figure 2 shows that training with PAL, GOLSI or PLS and a batch size of 8 on CIFAR-10 does not work at all. However, we can adapt LABPAL to work in these scenarios by holding the *noise scale* it is exposed to approximately constant. 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 4), we directly estimate a noise adaptation factor $\epsilon \in \mathbb{R}$ to adapt LABPAL's hyperparameter:

$$\epsilon := \frac{\nu_{new}}{\nu_{ori}} = \frac{|\mathbb{B}_{ori}|}{|\mathbb{B}_{new}|}\frac{|\mathbb{D}_{new}|}{|\mathbb{D}_{ori}|} = \frac{128}{|\mathbb{B}_{new}|}\frac{|\mathbb{D}_{new}|}{40,000} \tag{6}$$

The original batch size $|\mathbb{B}_{ori}|$ and the original dataset size $|\mathbb{D}_{ori}|$ originate from our training on CIFAR-10 with a training set size of 40,000, a batch size of 128, and 150,000 training steps. We set

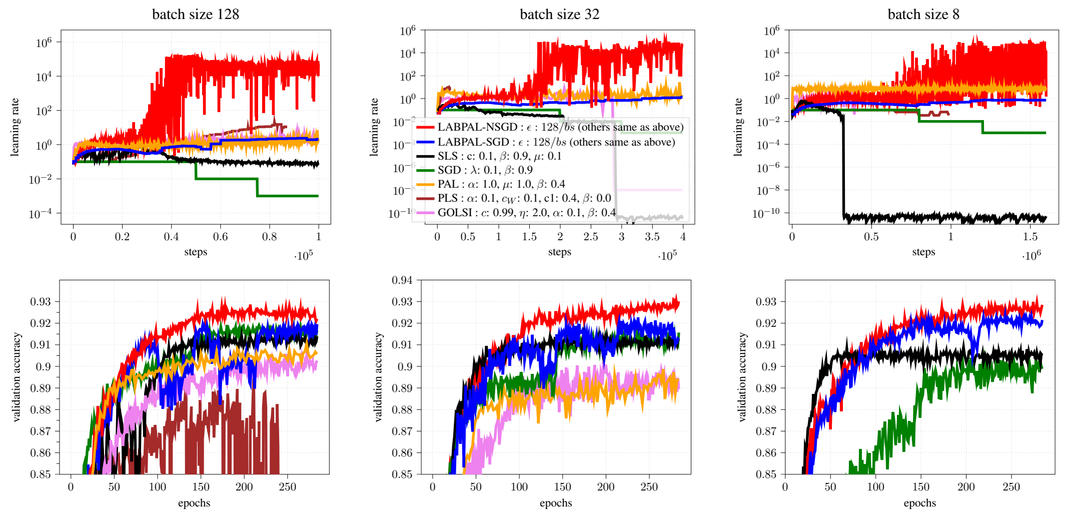

Figure 2: Performance comparison of a **DenseNet-121**, trained on **CIFAR-10** for **batch sizes 128, 32** and **8**. For the evaluation on ResNet-20 and MobileNet-V2 see Appendix Figure 10 & 11. Training steps are increased by a factor of 4 and 16 for batch size 32 and 8 respectively.

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 less in practice (at least for the image classification scenario), since hyperparameters do not have to be adapted across models, batch sizes and datasets. Figure 2 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 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 10 & 11. 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 limits of this work.

4. 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 3.1 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 lacking since the optimization field still does not know the reason for the local parabolic behavior of l_t is, and consequently, what an appropriate function space to consider for convergence is.

5. 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 line search work that analyses and adapts to varying gradient noise.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Lukas Balles. Probabilistic line search tensorflow implementation, 2017. URL https://github.com/ProbabilisticNumerics/probabilistic_line_search/commit/a83dfb0.

[3] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *ICLR*, 2018.

[4] Leonard Berrada, Andrew Zisserman, and M. Pawan Kumar. Training neural networks for and by interpolation. *ICML*, 2020.

[5] Nick Bostrom and Eliezer Yudkowsky. The ethics of artificial intelligence. *The Cambridge handbook of artificial intelligence*, 1:316–334, 2014.

[6] Younghwan Chae and Daniel N. Wilke. Empirical study towards understanding line search approximations for training neural networks. *arXiv*, 2019.

[7] Soham De, Abhay Kumar Yadav, David W. Jacobs, and Tom Goldstein. Big batch SGD: automated inference using adaptive batch sizes. *arXiv*, 2016.

[8] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. *ICML*, 2018.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[10] Stanislav Fort and Stanislaw Jastrzebski. Large scale structure of neural network loss landscapes. *NeurIPS*, 2019.

[11] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *ICLR*, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, 2017.

[14] Nocedal Jorge and Wright Stephen. *Numerical Optimization*. Springer series in operations research. Springer, 2nd ed edition, 2006. ISBN 9780387303031,0387303030.

[15] Dominic Kafka and Daniel Wilke. Gradient-only line searches: An alternative to probabilistic line searches. *arXiv*, 2019.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[17] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[19] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *NeurIPS*, 2018.

[20] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. *ICLR*, 2017.

[21] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[22] Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *J. Mach. Learn. Res.*, 18:119:1–119:59, 2017.

[23] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv*, 2018.

[24] Luke Muehlhauser and Louie Helm. The singularity and machine ethics. In *Singularity Hypotheses*, pages 101–126. Springer, 2012.

[25] Maximus Mutschler and Andreas Zell. Parabolic approximation line search for dnns. *NeurIPS*, 2020.

[26] Maximus Mutschler and Andreas Zell. Empirically explaining sgd from a line search perspective. *ICANN*, 2021.

[27] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. *NeurIPS Workshop*, 2011.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

[29] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[30] Michal Rolinek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. *NeurIPS*, 2018.

[31] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.

[32] Leslie N. Smith. Cyclical learning rates for training neural networks. *WACV*, 2017.

[33] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. *ICLR*, 2018.

[34] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *ICLR*, 2018.

[35] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. *NeurIPS*, 2019.

[36] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A walk with sgd. *arXiv*, 2018.

[37] Eliezer Yudkowsky et al. Artificial intelligence as a positive and negative factor in global risk. *Global catastrophic risks*, 1(303):184, 2008.

Appendix A. Related Work

Deterministic line searches: According to [14, §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 this scenario since they shrink the search space inadequately or use too inexact approximations of the loss. [14, §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. The following motivates the goal of our work to introduce a simple, reasonably fast, and well-performing line-search approach that approximates full-batch loss.

Many exact and inexact line search approaches for Deep Learning are applied on mini-batch losses [3, 4, 25, 30, 35]. [25] approximates 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 [14]). For these, convergence on convex stochastic functions can be assured under the interpolation condition [35], 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 [35]. 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 3. [25, 26] 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 [21]). 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) [22] addresses this problem by performing Gaussian Process 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 Gradient-only line search (GOLSI) [15]. 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 properties: [6, 8, 10, 11, 19, 22, 25, 26, 36]. [22, 25, 26, 36] suggest 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. [26] 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 3. Relevant insights and found properties of these works will be introduced and exploited to derive our algorithm in Section 2.

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 piratical 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*, 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 [34], which is exploited by our algorithm.

Appendix B. Additional Figures describing the empirical observations of [26]

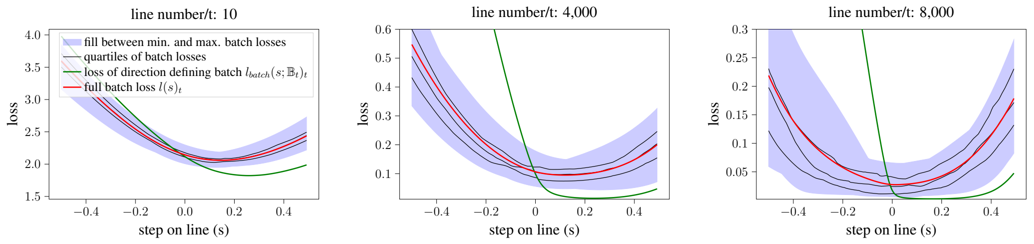

Figure 3: Losses along the lines of the SGD training processes exhibit a parabolic shape. The loss of the direction defining mini-batch (green) is excluded from the distribution of mini-batch losses to show that it is significantly different. This makes line searches on it unfavorable. In addition, the parabolic property articulates stronger for the full-batch loss (red); thus, this work aims to approximate it efficiently with a parabola. This introducing figure is created with code and data from [26].

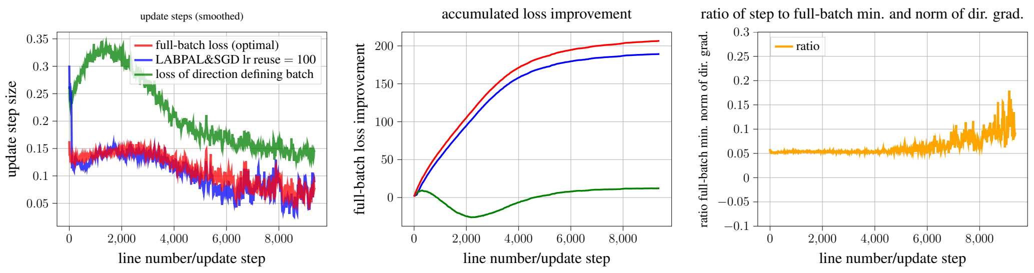

Figure 4: Several metrics to compare update step strategies on the full-batch losses along 10,000 lines measured by [26]: 1. update step sizes, 2. accumulated loss improvement per step given as: $l(0) - l(s_{upd})$ where s_{upd} is the update step of a specific optimizer. This is the locally optimal 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. Plotting code based on [26] with addition of our proposed approach.

B.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 extended the experiments of [26] by LABPAL. [26] 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 4 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 [26] 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 5 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 λs 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.

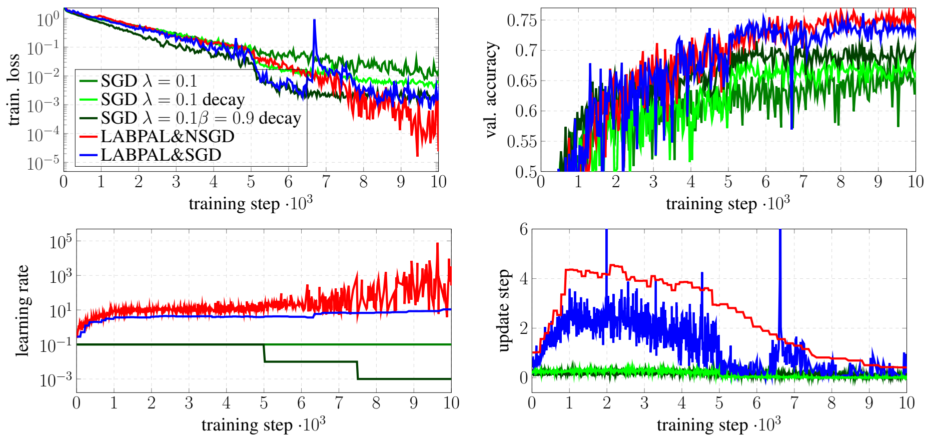

Figure 5: Training process on the problem of which the empirical observations were inferred (ResNet-20 trained on 8% of CIFAR-10 with SGD). LABPAL&NSGD and LABPAL&SGD outperform SGD. Interestingly LABPAL&NSGD estimated huge λs, whereas s_{upd}s are decreasing

Appendix C. Hyperparamter 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 1). Appendix Figure 6 and 7 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 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. [26] 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 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. In addition, we considered a momentum factor and conclude that a value between 0.4 and 0.6 increases the performance for both LABPAL approaches (row 5).

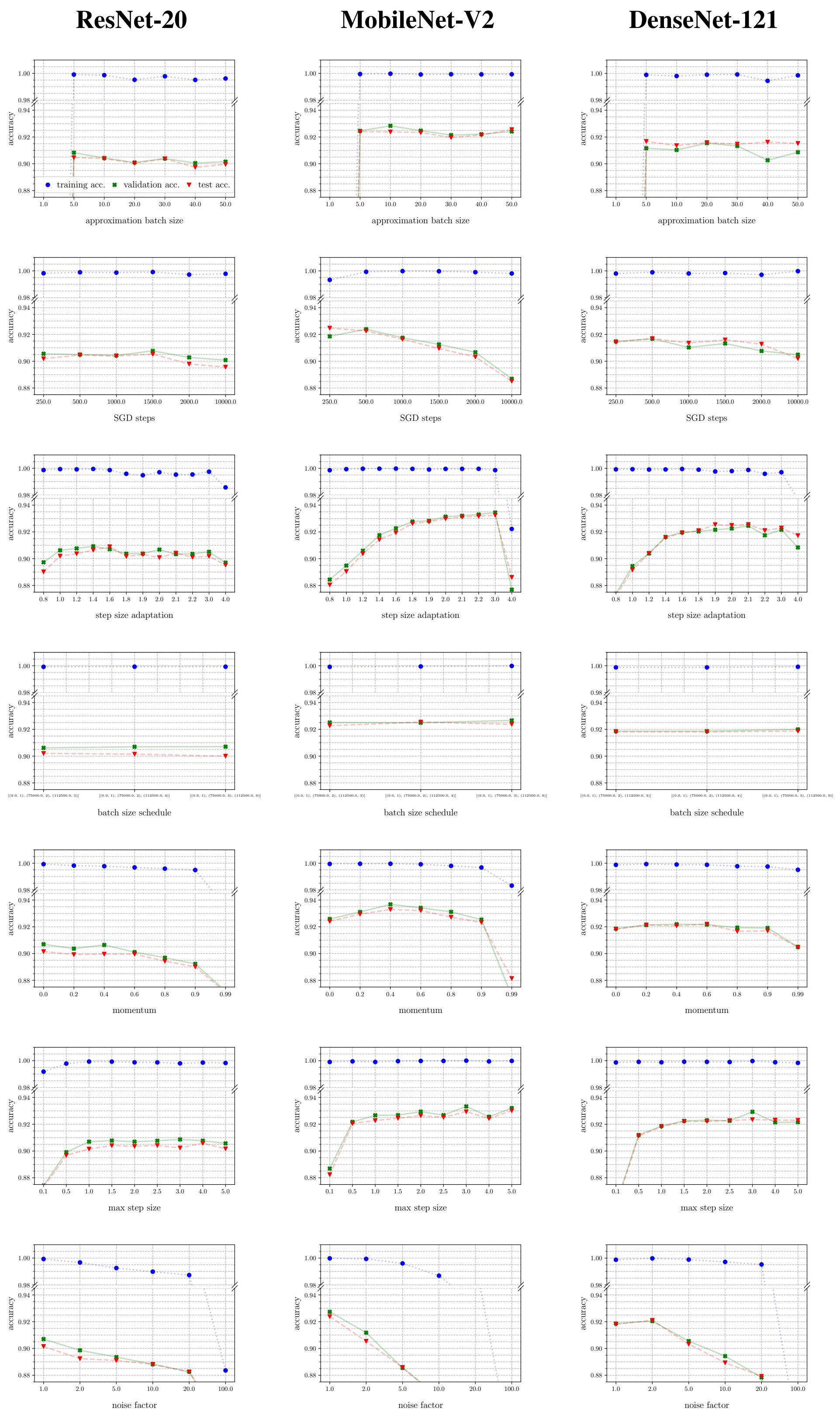

Figure 6: Sensitivity analysis of parameters of **LABPAL&SGD**. The observations of LABPAL&NSGD described in Figure 7 are also valid for LABPAL&NSGD.

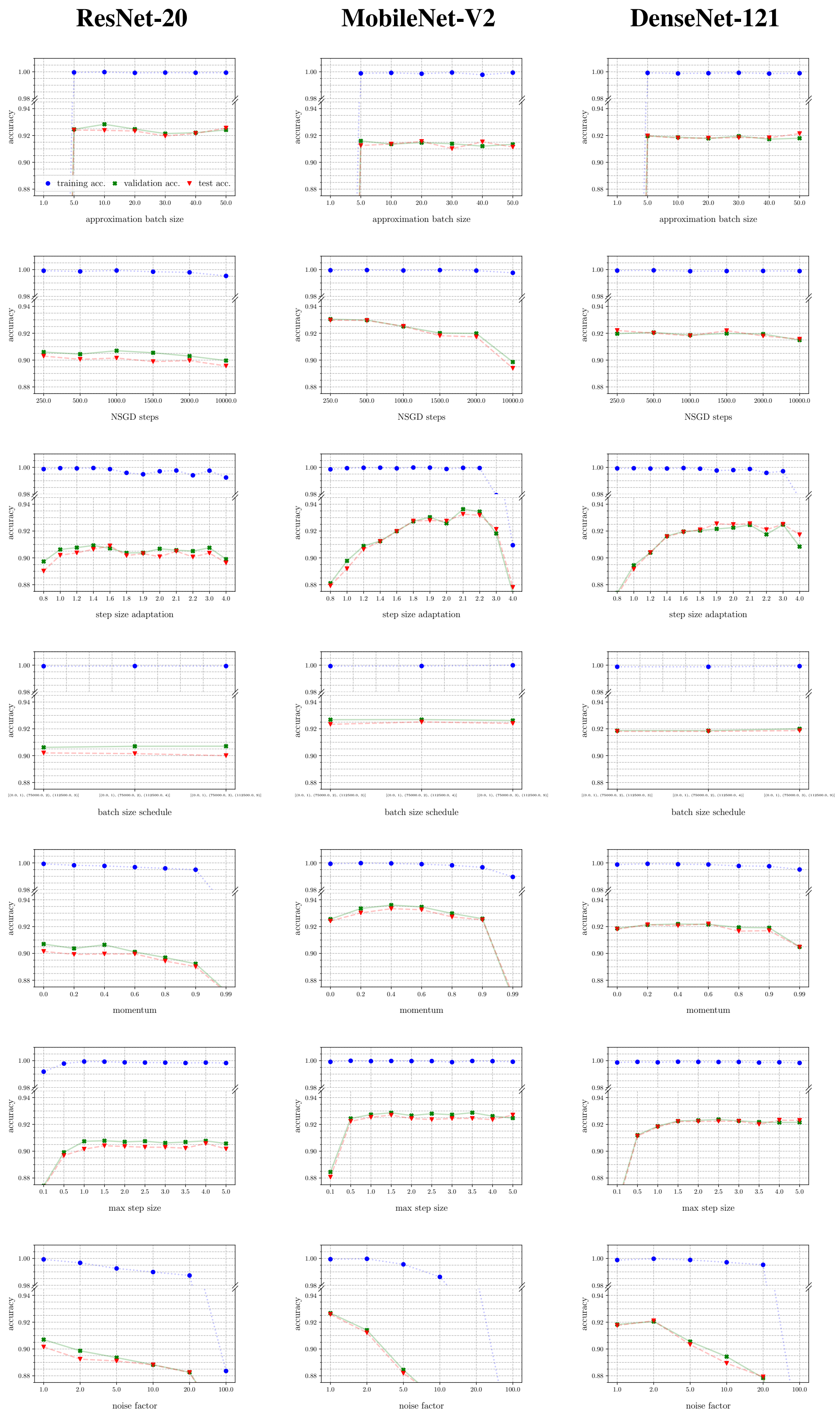

Figure 7: Sensitivity analysis of parameters of **LABPAL&NSGD**. The default parameters are: approximation batch size $\mathbb{B}_a = 1280$, SGD steps $s = 1000$, step size adaptation $\alpha = 1.8$, batch size schedule $k = (0{:}1,\ 75000{:}2,\ 112500{:}4)$, momentum $\beta = 0$, maximal step size $= 1.0$, noise-factor $\epsilon = 1$. For $\mathbb{B}_a$ the factor 128 is multiplied with is given on the x axis.

Appendix D. Further performance comparisons

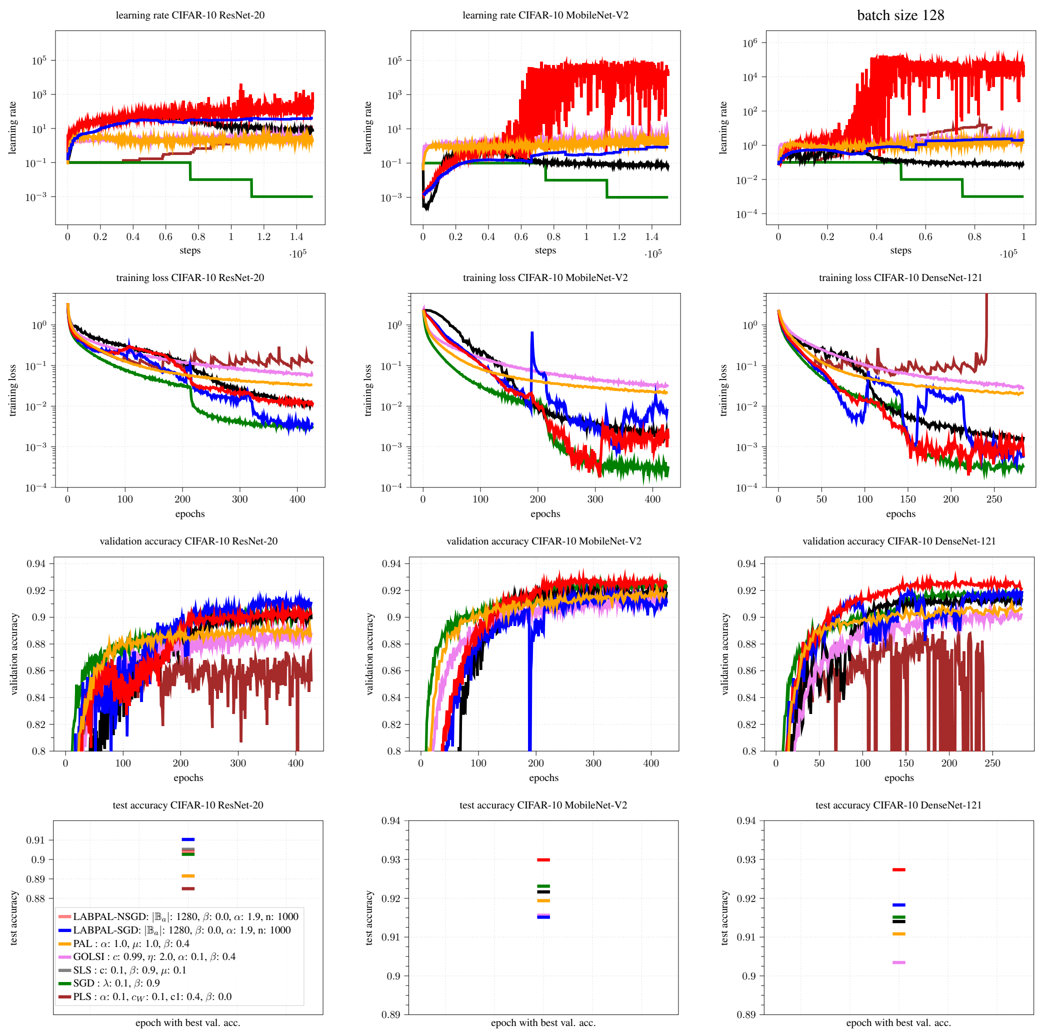

Figure 8: Performance comparison on **CIFAR-10** of our approach LABPAL in the SGD and NSGD variants against several line searches and SGD. Optimal hyperparameters are found with an elaborate grid search. Our approaches challenge and sometimes outperform the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics.

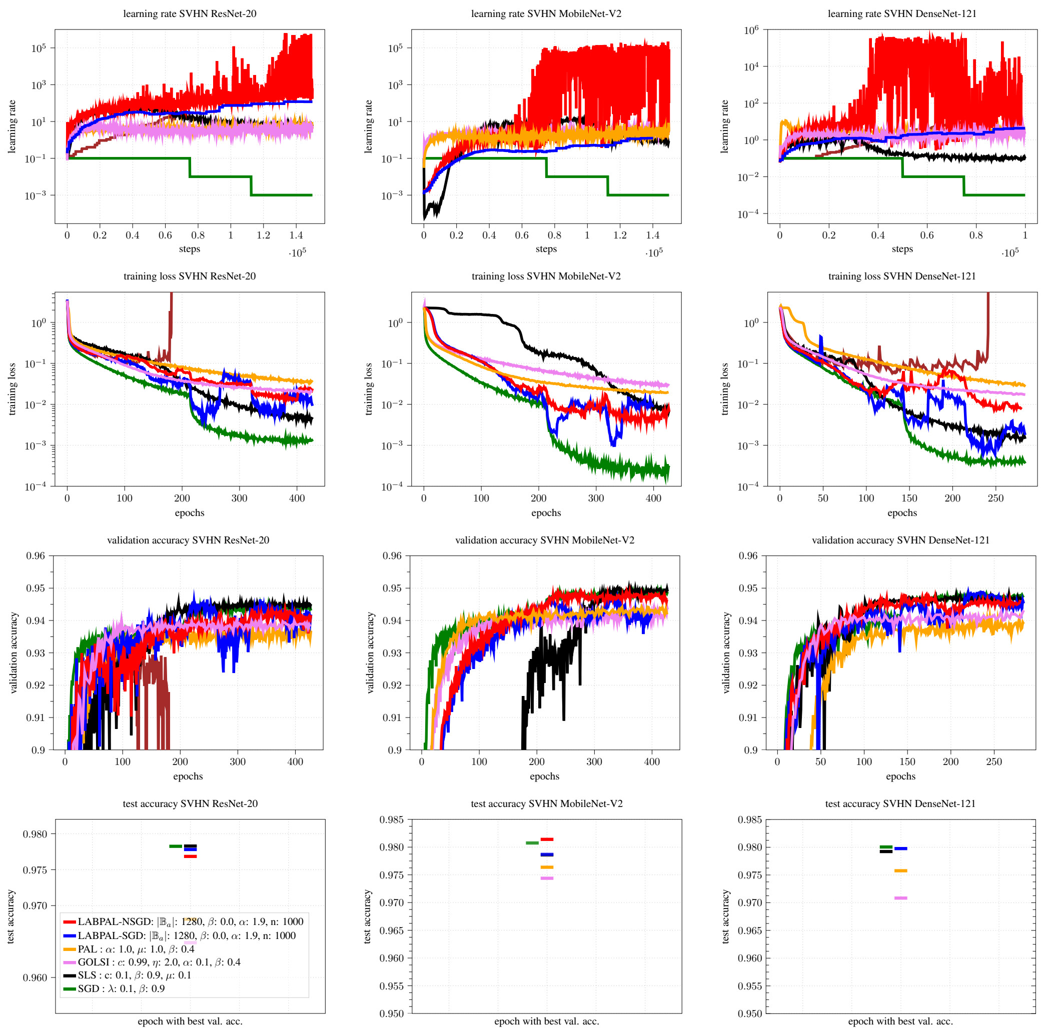

Figure 9: Performance comparison on **SVHN** of our approach LABPAL in the SGD and NSGD variants against several line search and SGD. Optimal hyperparameters found with a detailed grid search for CIFAR-10 are reused. Our approaches challenge and sometimes surpass the other approaches on training loss, validation, and test accuracy. Columns indicate different models. Rows indicate different metrics.

Appendix E. Further results for batch sizes 32 and 8

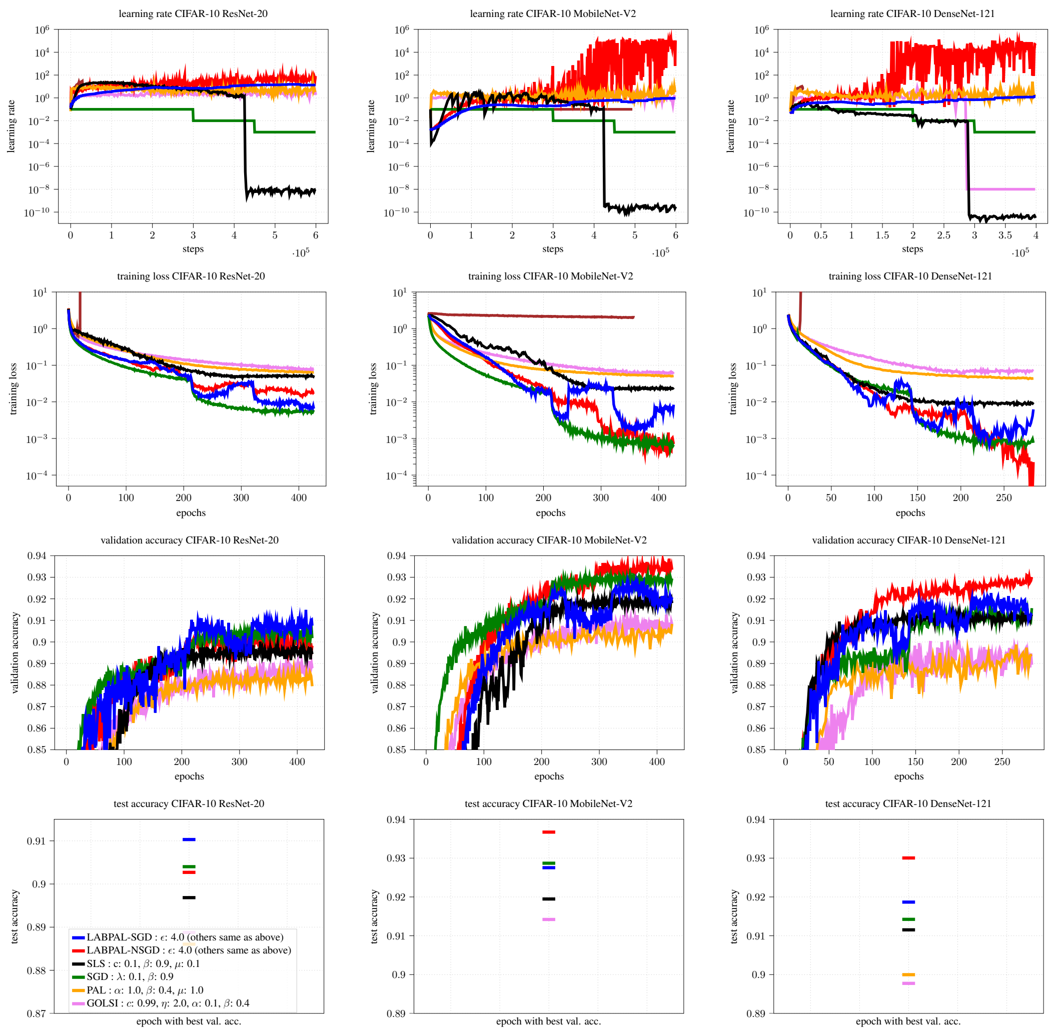

Figure 10: Performance comparison of several models on CIFAR-10 with **batch size 32**. **The same hyperparameters are used as for batch size 128** (see Figure 8). Due to the batch size adaptation to keep the noise scale on a similar level the LABPAL approaches perform almost identical compared to batch size 128. The performance off all other line searches decreases; SGD still performs well. Not that training steps were increased by a factor of 4.

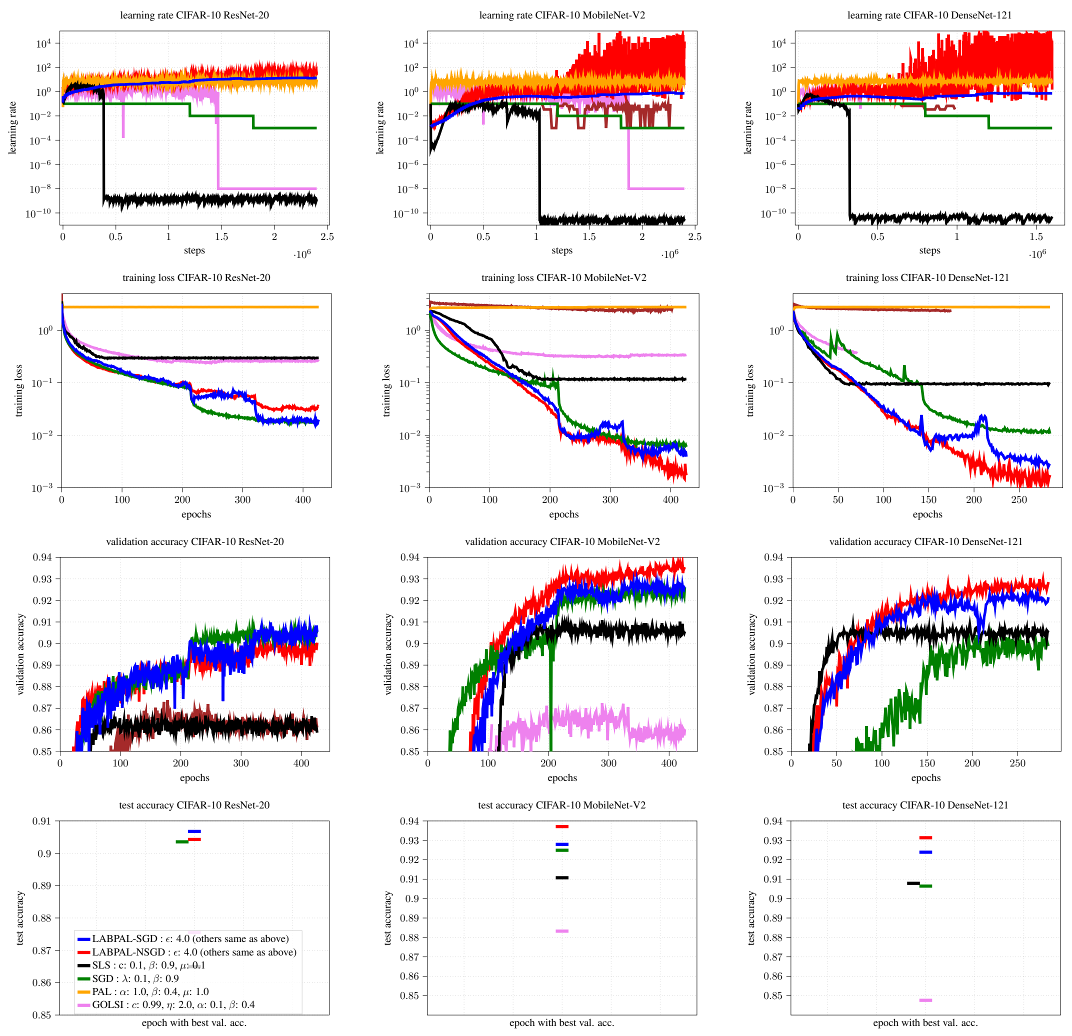

Figure 11: Performance comparison of several models on CIFAR-10 with **batch size 8**. **The same hyperparameters are used as for batch size 128** (see Figure 8). Due to the batch size adaptation to keep the noise scale on a similar level the LABPAL approaches still perform well. PAL and PLS fail to 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 degenerated. Note that training steps were increased by a factor of 16.

Appendix F. Wall clock time and GPU memory comparison

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 cheap.

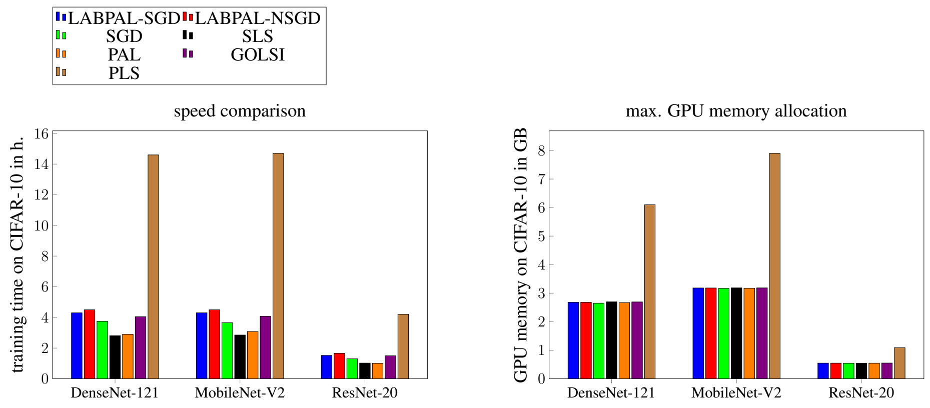

Figure 12: **Left:** Training time comparison on CIFAR-10. SGD, SLS, and PAL show similar training times. GOLSI, and both variants of LABPAL are slightly slower (up to 19.6%). However, a slightly longer training time is acceptable if less time has to be spent in hyper-parameter tuning. PLS is significantly slower. Note that in comparison to SGD, the implementations of the other optimizers are not optimized on CUDA level. **Right:** Maximum allocated memory comparison on CIFAR-10. Except for PLS all approaches need approximately the same amount of memory.

Appendix G. Relation of update step adaptation α and the first wolfe constant w_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):

$$f(x_0 + y) \leq f(x_0) - y\nabla f(x_0)w_1 \qquad \text{in our case } x_0 = 0, w_1 \text{ wolfe constant} \tag{7}$$

$$f(y) \leq f(0) + ybw_1 \tag{8}$$

$$ay^2 + by + c \leq c + ybw_1 \qquad \text{use quadratic shape, } \nabla f(x_0) = b \tag{9}$$

$$ay^2 + by - ybw_1 \overset{!}{=} 0 \tag{10}$$

$$\frac{ay^2 + by}{by} = \frac{ay}{b} + 1 = w_1 \tag{11}$$

$$-\frac{\alpha}{2} + 1 = w_1 \qquad \text{set } y = \alpha\frac{-b}{2a}, \alpha \in [1, 2) \tag{12}$$

$$-2w_1 + 2 = \alpha \tag{13}$$

Appendix H. Further experimental details

Further experimental details for the optimizer comparison in Figure 8,1,9,**??**,**??**,**??** of Sections 3.1 & 3.2.
PLS: We adapted the only available and empirically improved TensorFlow [1] implementation of PLS [2], which was transferred to PyTorch [28] by [35], to run on several state-of-the-art models and datasets.
The training steps for the experiments in section Section 3 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/train 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.

H.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 step 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	μ_{max}	$\{10.0\}$

For SLS no momentum term is considered since [35] already showed SLS variants using momentum like acceleration methods to be non-beneficial.