DESCENDING THROUGH A CROWDED VALLEY — BENCHMARKING DEEP LEARNING OPTIMIZERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Choosing the optimizer is considered to be among the most crucial decisions of deep learning engineers, and it is not an easy one. The growing literature now lists hundreds of optimization methods. In the absence of clear theoretical guidance and conclusive empirical evidence, the decision is often made according to personal anecdotes. In this work, we aim to replace these anecdotes, if not with evidence, then at least with heuristics. To do so, we perform an extensive, standardized benchmark of more than a dozen particularly popular deep learning optimizers while giving a concise overview of the wide range of possible choices. Analyzing almost $35\,000$ individual runs, we contribute the following three points: (i) Optimizer performance varies greatly across tasks. (ii) We observe that evaluating multiple optimizers with default parameters works approximately as well as tuning the hyperparameters of a single, fixed optimizer. (iii) While we can not determine an individual optimization method clearly dominating across all tested tasks, we identify a significantly reduced subset of specific algorithms and parameter choices that generally lead to competitive results in our experiments. This subset includes popular favorites and some lesser-known contenders. We have open-sourced all our experimental results, making them directly available as challenging and well-tuned baselines for more meaningful comparisons when evaluating novel optimization methods without requiring any further computational efforts.[1]

1 INTRODUCTION

Large-scale stochastic optimization drives a wide variety of machine learning tasks. Choosing the right optimization algorithm, and effectively tuning its hyperparameters, heavily influences the training speed and final performance of the learned model, and is an important, every-day challenge to practitioners. It is thus not surprising that stochastic optimization methods have been a focal point of research (cf. Figure 1), engendering an ever-growing list of optimization methods, many of them specifically targeted towards deep learning. The hypothetical machine learning practitioner who is able to keep up with the literature now has to choose among hundreds of optimization algorithms (cf. Table 2 in the appendix) — each with their own set of tunable hyperparameters — when deciding on how to train their model.

There is only very limited theoretical analysis that would clearly favor one of these choices over the others. Some authors have offered empirical comparisons on a limited set of popular methods e.g. (Choi et al., 2019; Sivaprasad et al., 2019; Wilson et al., 2017); but for most algorithms, the only formal empirical evaluation is offered by the original work introducing the method. Many practitioners and researchers, meanwhile, rely on personal and anecdotal experience, and informal discussion on social media or with colleagues. The result is an often unclear, occasionally emotional, but always changing "state of the art" that seems driven more by marketing and hype than objective performance. The key obstacle for an objective benchmark is the combinatorial cost of such an endeavor posed by comparing a large number of methods on a large number of problems, with the high resource and time cost of tuning each method's parameters and repeating each (stochastic) experiment repeatedly for fidelity. Here, we offer our best attempt to construct such a comparison.

We conduct a large-scale benchmark of optimizers to further inform the debate about deep learning optimizers, and to help understand how the choice of optimization method and hyperparameters

[1] `https://github.com/AnonSubmitter3/Submission543`

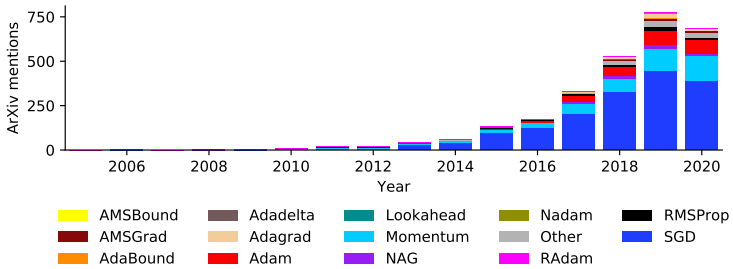


Figure 1: Number of times ArXiv titles and abstracts mention each optimizer per year. All non-selected optimizers from Table 2 in the appendix are grouped into *Other*.

influences the training performance. Specifically, we want to examine whether recently proposed methods show an improved performance compared to more established methods such as SGD or ADAM. Additionally, we are interested in assessing whether optimization methods with well-working default hyperparameters exist that are able to keep up with tuned optimization methods. To this end, we evaluate more than a dozen popular optimization algorithms, largely selected for their perceived popularity, on a range of representative deep learning problems (see Figure 4) drawing conclusions from tens of thousands of individual training runs.

Right up front, we want to state clearly that it is impossible to include all optimizers (cf. Appendix), and to satisfy any and all expectations readers and reviewers may have on tuning and initialization procedures, or the choice of benchmark problems — not least because everyone has different expectations in this regard. In our *personal opinion*, what is needed is an empirical comparison by a third party not involved in the original works. As a model reader of our work, we assume a careful practitioner who does not have access to near limitless resources, nor to a broad range of personal experiences. As such, the core contributions (in order of appearance, not importance) of our work are:

A concise summary of optimization algorithms and schedules A partly automated, mostly manual literature review provides a compact but extensive list of recent advances in stochastic optimization. We identify more than a hundred optimization algorithms (cf. Table 2 in the appendix) and more than 20 families of hyperparameter schedules (cf. Table 3 in the appendix) published at least as pre-prints.

An extensive optimizer benchmark on deep learning tasks We conduct a large-scale optimizer benchmark, specifically focusing on optimization problems arising in deep learning. Evaluating 14 optimizers on eight deep learning problems using four different schedules, tuning over dozens of hyperparameter settings, to our knowledge, this is the most comprehensive empirical evaluation of deep learning optimizers up-to-date (cf. Section 1.1 on related work).

An analysis of thousands of optimization runs Our empirical experiments indicate that an optimizer's performance highly depends on the test problem (see Figure 4). But some high-level trends emerge, too: (1) Evaluating multiple optimizers (with default hyperparameters) works approximately as well as tuning the hyperparameters for a fixed optimizer. (2) Using an additional (untuned) learning rate schedule helps on average, but its effect varies greatly depending on the optimizer and the test problem. (3) While there is no optimizer that clearly dominates across all tested workloads, some of the algorithms we tested exhibited highly variable performance, while others consistently worked decently. We deliberately refrain from recommending a single one among them, because we can not find a clear winner with statistical confidence.

An open-source baseline for future optimizer benchmarks We provide all results online, in easily accessible, open form. Our results can thus be used as competitive and well-tuned baselines for future benchmarks of new algorithms, drastically reducing the amount of computational budget required for a meaningful optimizer comparison. Our baselines can easily be expanded, and we encourage everyone to contribute to this collection.

The high-level result of our benchmark is, as one might perhaps expect, *not* a clear winner. Rather, our comparison shows that, while some optimizers are frequently decent, they also generally perform

similarly, switching their relative positions in the ranking. A key insight of our comparison is that, on balance, an uninformed practitioner with a new, personal deep learning task can expect to do about *equally well* by taking almost any method from our benchmark and *tuning* it as they would by investing the same computational resources into running a selection of optimizers from our benchmark in their *default* ("one-shot") setting and picking the winner.

Possibly the most important takeaway from our comparison is that "there are now enough optimizers." Methods research in stochastic optimization should focus on *significant* (conceptual, functional, performance) improvements – such as methods specifically suited for certain problem types, inner-loop parameter tuning or structurally novel methods. We make this claim not to discourage research but, quite on the contrary, to offer a motivation for more meaningful, non-incremental research.

1.1 Related work

Perhaps spurred by the rapid increase in publications on optimizers, *benchmarking* these methods for the application in deep learning has only recently attracted larger interest. Schneider et al. (2019) introduced a benchmarking framework called DeepOBS, which includes a wide range of realistic deep learning test problems together with standardized procedures for evaluating optimizers. Metz et al. (2020) presented TaskSet, another collection of optimization problems focusing on smaller but many more test problems. For the empirical analysis presented here, we use DeepOBS as it provides optimization problems closer to real-world deep learning tasks. In contrast to our evaluation of *existing* methods, TaskSet and its analysis focuses on meta-learning *new* algorithms or hyperparameters.

Both Choi et al. (2019) and Sivaprasad et al. (2019) analyzed specific aspects of the benchmarking process. Sivaprasad et al. (2019) used DeepOBS to illustrate that the relative performance of an optimizer depends significantly on the used hyperparameter tuning method. The analysis by Choi et al. (2019) supports this point, stating that "the hyperparameter search space may be the single most important factor explaining the rankings." They further stress a hierarchy among optimizers, demonstrating that, given sufficient hyperparameter tuning, more general optimizers can never be outperformed by special cases. In their study, however, they manually chose a hyperparameter search space *per optimizer and test problem* basing it either on prior published results, prior experiences, or pre-tuning trials. Here we instead aim to identify well-performing optimizers in the case of a less extensive tuning budget and especially when there is no prior knowledge about well-working hyperparameter values for each specific test problem. We further elaborate on the influence of our chosen hyperparameter search strategy in Section 4 discussing the limitations of our empirical study.

Apart from these, our work is related to empirical generalization studies of adaptive methods, such as that of Wilson et al. (2017) which sparked an extensive discussion whether adaptive methods (e.g. Adam) tend to generalize worse than standard first-order methods (i.e. SGD).

2 Benchmarking process

Benchmarking optimization algorithms comes with certain decisions of the experimental setup that will influence the result. Evaluating on a specific task or picking a certain tuning budget, for example, may favor or disadvantage certain algorithms (Sivaprasad et al., 2019). It is impossible to avoid these decisions and infeasible to test all possible choices. To make the result of our analysis as general as possible, we evaluate the performance on eight diverse real-world deep learning problems from different disciplines (see Section 2.1). From a collection of more than a hundred deep learning optimizers (see Table 2 in the appendix) we select 14 of the most popular and most promising choices (cf. Figure 1) for this benchmark (see Section 2.2). For each test problem and optimizer, we evaluate all possible combinations of three different tuning budgets (see Section 2.3) and four selected learning rate schedules (see Section 2.4). We thus fully cover the following combinatorial space:

$$\begin{array}{cccc} \textbf{Problem} & \textbf{Optimizer} & \textbf{Tuning} & \textbf{Schedule} \\ \left\{\begin{array}{l}\text{P1}\\ \text{P2}\\ \dots\\ \text{P8}\end{array}\right\}_{8} \times & \left\{\begin{array}{l}\text{AMSBound}\\ \text{AMSGrad}\\ \dots\\ \text{SGD}\end{array}\right\}_{14} \times & \left\{\begin{array}{l}\text{one-shot}\\ \text{small budget}\\ \text{large budget}\end{array}\right\}_{3} \times & \left\{\begin{array}{l}\text{constant}\\ \text{cosine decay}\\ \text{cosine warm restarts}\\ \text{trapezoidal}\end{array}\right\}_{4} . \end{array}$$

Table 1: Summary of test problems used in our experiments.

	Data set	**Model**	**Task**	**Metric**	**Batch size**	**Budget** *in epochs*
P1	Artificial data set	Noisy quadratic	Minimization	Loss	128	100
P2	MNIST	VAE	Generative	Loss	64	50
P3	Fashion-MNIST	2c2d	Classification	Accuracy	128	100
P4	CIFAR-10	3c3d	Classification	Accuracy	128	100
P5	Fashion-MNIST	VAE	Generative	Loss	64	100
P6	CIFAR-100	All-CNN-C	Classification	Accuracy	256	350
P7	SVHN	Wide ResNet 16-4	Classification	Accuracy	128	160
P8	War and Peace	RNN	NLP	Accuracy	50	200

Combining those options results in 1344 possible configurations and roughly $35\,000$ individual runs.

2.1 Test problems

For our benchmark, we consider the eight optimization tasks summarized in Table 1, available as the small (P1–P4) and large (P5–P8) problem set and the default suggestion of the benchmarking framework DeepOBS, respectively. The details of these optimization problems, such as the chosen architectures, training parameters, etc. can be found in the work of Schneider et al. (2019). All experiments were performed using version `1.2.0-beta` of DeepOBS and TensorFlow version `1.15` Abadi et al. (2015). DeepOBS' test problems provide several performance metrics, including the training and test loss, the validation accuracy, etc. While these are all relevant, any comparative evaluation of optimizers requires picking only a few, if not just one particular performance metric. For our analysis (Section 3), we focus on the final test accuracy (or the final test loss, if no accuracy is defined for this problem, see Table 1) of the trained model. This metric captures, for example, the optimizer's ability to generalize and is thus a highly relevant metric for practical use (our publicly released results include all metrics, for completeness, an example of training loss performance is shown in Figure 16 in the appendix). Accordingly, the tuning (Section 2.3) is done with respect to the validation metric. We discuss possible limitations resulting from these choices in Section 4.

2.2 Optimizer selection

Recent years have produced a surge of optimization algorithms. In Table 2 in the appendix we collected more than a hundred optimizers introduced for, suggested for, or used on deep learning problems. This list was manually and incrementally collected by multiple researchers trying to keep up with the field over recent years. It is thus necessarily incomplete, although it may well represent one of the most exhaustive of such collections. Even this incomplete list, though, contains too many entries for a meaningful benchmark with the degrees of freedom collected above. This is a serious problem for research: It means that even an author of a new optimizer, let alone a practitioner, could not possibly be expected to compare their work with every possible competing method.

To keep our comparison feasible, we selected a subset of 14 optimizers, which we consider to be currently the most popular choices in the community (see Table 4 in the appendix). These do not necessarily reflect the "best" optimization algorithms, but are either commonly used by practitioners and researchers, or have recently generated enough attention to gain a certain degree of interest. Our selection is focused on first-order optimization methods, both due to their prevalence for non-convex continuous optimization problems in deep learning as well as to simplify the comparison. Whether these optimizers are inherently redundant is one of the question this work investigates. Since both DeepOBS and the benchmark results presented in this paper are open-source, other methods can easily be added to the benchmark in the future.

2.3 Tuning

Budget Optimization methods for deep learning regularly expose hyperparameters to the user. The user sets them either by relying on the default suggestion; using experience from previous

experiments; or using additional tuning runs to find the best-performing setting. All optimizers in our benchmark have tunable hyperparameters, and we consider three different *tuning budgets*.

The first budget consists of just a single run. This *one-shot* budget uses the default values proposed by the original authors, where available (Table 4 in the appendix lists the default parameters). If an optimizer performs well in this setting, this naturally has great practical value as it drastically reduces the computational resources required for training. The other budgets consist of 25 and 50 tuning runs for what we call the *small* and *large budget* settings, respectively.

We only use a single seed for tuning, then repeat the best setting 10 times using different seeds. This allows us to report standard deviations in addition to means, assessing stability. Progressing in this way has the "feature" that our tuning process can sometimes pick "lucky" seeds, which don't perform as well when averaging over multiple runs. However, this arguable captures the real world faithfully. Stable optimizers should be preferred in practice, which is thus reflected in our benchmark. For a more detailed analysis see Appendix C in the appendix. The approach of using all 10 random seeds for tuning as well, would come at a drastically increased cost, not only for this benchmark, rendering it practically infeasible, but also as an approach for the practical user. In Appendix D we show with a re-tuning experiment that we can expect additional noise from this simpler tuning process, which tends to average out over multiple problems.

Tuning method We tune parameters by random search, for both the small and the large budget. Random search is a common choice in practice due to its efficiency advantage over grid search (Bergstra & Bengio, 2012) and its ease of implementation and parallelization compared to Bayesian optimization (see also Section 4). A minor complication of random search is that the sampling distribution affects the optimizer's performance (one can think of the sampling distribution as a prior over good parameter settings. Bad priors ruin performance). We followed the mathematical bounds and intuition provided by the optimizers' authors for relevant hyperparameters as much as possible. The resulting sampling distributions can be found in Table 4 in the appendix. Where there is no prior knowledge provided in the op.cit. we chose similar distributions for similar hyperparameters across different optimizers. Even though a parameter might have a similar naming throughout different optimization algorithms (e.g. learning rate α), its appropriate search space can differ across optimizers. Without grounded heuristics on how the parameters differ between optimizers, the most straightforward approach for any user is to use the same search space.

What should be considered a hyperparameter? There's a fuzzy boundary between (tunable) hyperparameters and (fixed) design parameters. A recently contentious example is the ϵ in adaptive learning rate methods like ADAM. It was originally introduced as a safeguard against division by zero, but has recently been re-interpreted as a problem-dependent hyperparameter choice (see Choi et al. (2019) for a discussion). Under this view, one can actually consider several separate optimizers called ADAM: From an easy-to-tune but potentially limited ADAM_α, only tuning the learning rate, to the tricky-to-tune but all-powerful $\text{ADAM}_{\alpha,\beta_1,\beta_2,\epsilon}$, which subsumes SGD as a corner case in its hyperparameter space. In our benchmark, we include $\text{ADAM}_{\alpha,\beta_1,\beta_2}$ as a popular choice. While they share the same update rule, we consider them to be different optimizer.

2.4 SCHEDULES

The literature on learning rate schedules is now nearly as extensive as that on optimizers (cf. Table 3 in the appendix). *In theory*, schedules can be applied to all hyperparameters of an optimization algorithm but to keep our configuration space feasible, we only apply schedules to the learning rate, by far the most popular practical choice (Goodfellow et al., 2016; Zhang et al., 2020). We choose four different learning rate schedules, trying to cover all major types of schedules:

- A *constant* learning rate schedule;
- A *cosine decay* (Loshchilov & Hutter, 2017) as an example of a smooth decay;
- A *cosine with warm restarts* schedule (Loshchilov & Hutter, 2017) as an example of a cyclical learning rate;
- A *trapezoidal* schedule (Xing et al., 2018) from the warmup schedules (Goyal et al., 2017).

We describe the schedules and their parameters in Appendix E in the appendix.

3 RESULTS

How well do optimizers work out-of-the-box? By comparing each optimizer's one-shot results against the tuned versions of all 14 optimizers, we can construct a 14×14 matrix of absolute performance changes. Figure 2 illustrates this on five of the eight test problems (see Figures 9 and 10 in the appendix for detailed plots of all eight test problems) showing performance improvements by a positive sign and a green cell.

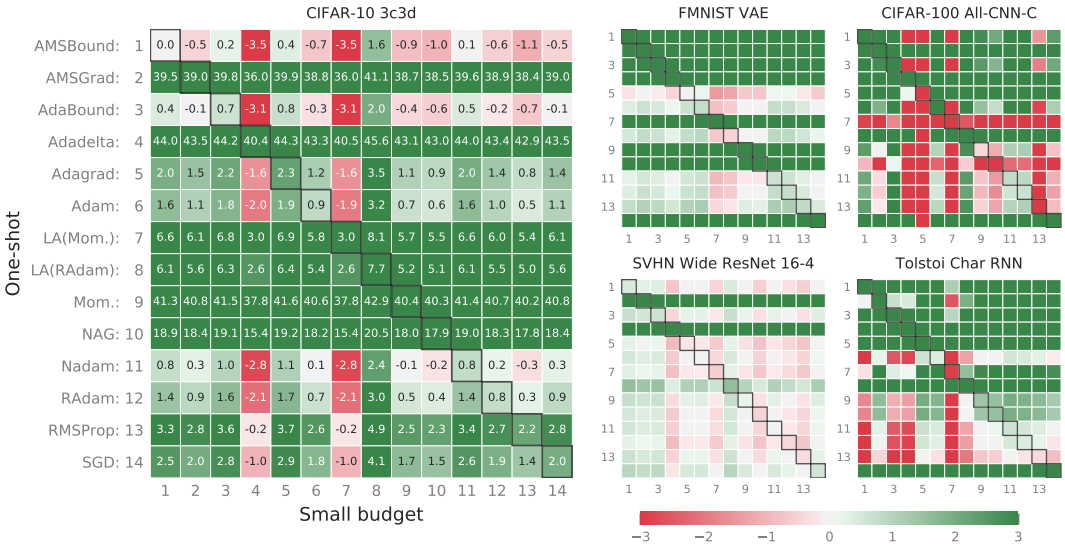


Figure 2: The absolute test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *small budget*) as an average over 10 random seeds for the *constant* schedule. We discuss the occurrence of negative diagonal entries in Appendix F. The joint colormap is capped at ± 3 to improve presentation, although larger values occur.

A **green row** in Figure 2 indicates that an optimizer's default setting is performing badly, since it can be beaten by any well-tuned competitor. We can observe bad-performing default settings for MOMENTUM, NAG and SGD, advocating the intuition that non-adaptive optimization methods require more tuning, but also for AMSGRAD and ADADELTA. Note, that this is just a statement about the default parameters suggested by the authors or the popular frameworks and well-working default parameters might exist for those methods. Conversely, a **white & red row** signals a well-performing default setting, since even tuned optimizers can not significantly outperform this algorithm. ADAM, NADAM and RADAM, as well as AMSBOUND and ADABOUND all have white or red rows on several (but not all!) test problems, supporting the rule of thumb that adaptive methods have well-working default parameters. Conversely, **green** (or **red**) **columns** highlight optimizers that, when tuned, perform better (or worse) than all untuned optimization methods. We do not observe such columns consistently across tasks. This supports the conclusion that an optimizer's performance is heavily problem-dependent and that there is no single *best* optimizer across workloads.

Figures 9 to 12 in the appendix and our conclusions from them suggest an interesting alternative approach for machine learning practitioners: Instead of picking a single optimizer and tuning its hyperparameters, trying out multiple default setting optimizers and picking the best one should yield competitive results with less computational and tuning choice efforts. The similarity of those two approaches might be due to the fact that optimizers have implicit learning rate schedules and trying out different optimizers is similar to trying out different (well-tested) schedules (Agarwal et al., 2020).

How much do tuning and schedules help? We consider the final performance achieved by varying budgets and schedules to quantify the usefulness of tuning and applying parameter-free schedules (Figure 3). While there is no clear trend for any individual setting (gray lines), in the median we observe that increasing the budget improves performance, albeit with diminishing returns. For example, using the large budget without any schedule leads to a median relative improvement of the performance of roughly $3.3\,\%$ compared to the default parameters (without schedule).

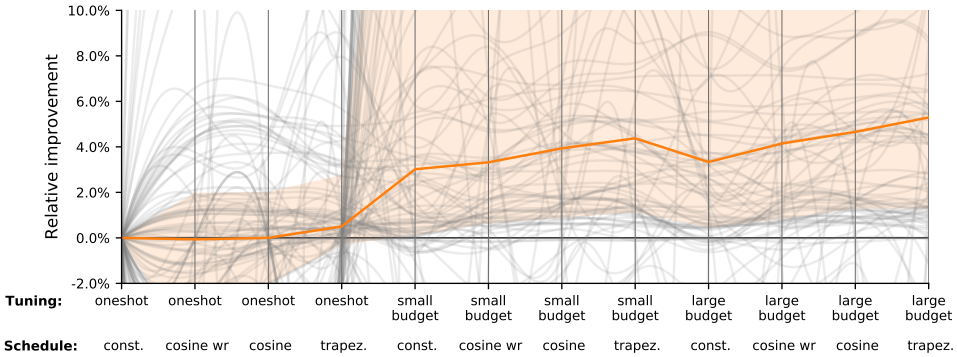


Figure 3: Lines in gray (—, smoothed by cubic splines for visual guidance only) show the relative improvement for a certain tuning and schedule (compared to the *one-shot* tuning without schedule) for all 14 optimizers on all eight test problems. The median over all lines is plotted in orange (—) with the shaded area (▮) indicating the area between the 25th and 75th percentile.

Similarly, applying a parameter-free (i.e. untuned) schedule improves performance in the median. For example, the large tuning budget coupled with a trapezoidal learning rate schedule leads to a median relative improvement of roughly $5.3\,\%$ compared to the default parameters. However, while these trends hold in the median, their individual effect varys wildly among optimizers and test problems, as is apparent from the noisy structure of the individual lines shown in Figure 3.

Which optimizers should you pick? Figure 4 compares the optimizers' performance across the test problems. There is no single optimizer that dominates its competitors across all tasks. Nevertheless, some optimizers generally perform well, while others vary wildly in their behaviour.

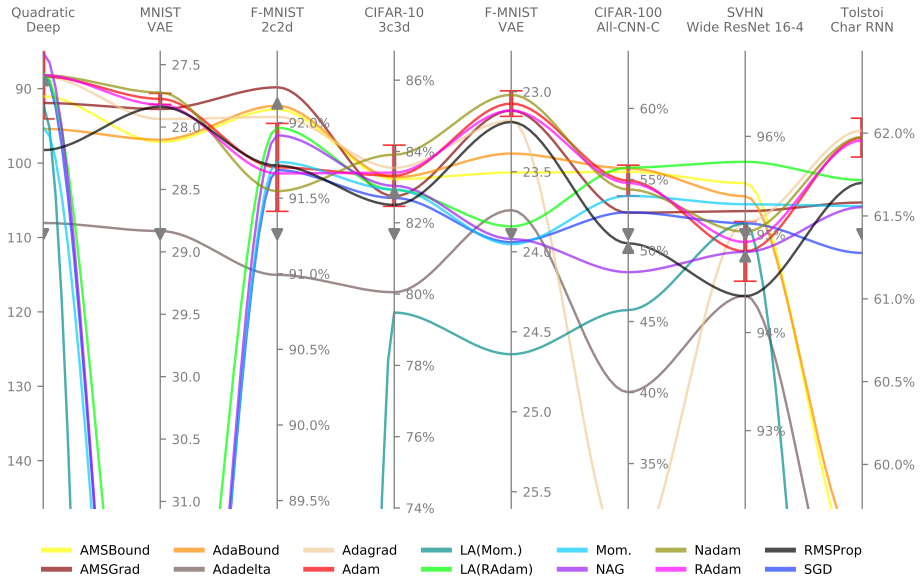


Figure 4: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and *no learning rate schedule*. One standard deviation for the *tuned* ADAM optimizer is shown with a red error bar (**I**; error bars for other methods omitted for legibility, available in our open data set). The performance of the *untuned* versions of ADAM (▼) and ADABOUND (▲) are marked for reference. The upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is based on the performance of ADAM with default parameters.

Further supporting the hypothesis of previous sections, we note that taking the best out of a small set of *untuned* optimizers — for example, ADAM and ADABOUND— frequently results in competitive

overall performance, even compared to well-tuned optimizers. Combining these runs with a *tuned* version of ADAM (or variants thereof) would offer competitive results in our benchmark. Nevertheless, to achieve (very close to) the absolute best performance, it is necessary to try out multiple optimizers. Which optimizer to choose to this end, though, is highly problem-dependent: optimizers that achieve top scores on one problem can perform rather badly on other tasks. We note in passing that the individual optimizer rankings shift quite substantially when considering e.g. a smaller budget or an additional learning rate schedule (see Figures 13 to 15 in the appendix). The overall trends mentioned in this section stay consistent, though.

4 LIMITATIONS

Any empirical benchmark has its constraints and limitations. Here we highlight some prominent limitations of our study and limit the context within which our results should be considered.

Generalization of the results By using the test problems from DEEPOBS, which span models and data sets of varying complexity, size, and different domains, we aimed for generalization. Despite our best efforts, of course, our results are specific not just to these setups, but also to the chosen training parameters, the software framework, and other choices we are forced to make. The design of our comparisons aims to be close to what informed practitioners would encounter in their daily work. It goes without saying, though, that even a carefully considered range of deep learning problems can not cover all relevant aspects of machine learning or even deep learning. In particular, our conclusions might not easily generalize to other types of workloads such as GANs, reinforcement learning, or applications where aspects such as memory usage are important.

Influence of the hyperparameter search strategy As noted by, e.g., Choi et al. (2019) and Sivaprasad et al. (2019), the hyperparameter tuning method, its budget, and its search domain, can significantly affect performance. By reporting results from three different hyperparameter optimization budgets (including the tuning-free one-shot setting) we try to quantify the effect of tuning. We believe that our random search process presents a realistic setting for many but certainly not all deep learning practitioners. One may criticize our approach as simplistic, but note that more elaborate schemes, in particular Bayesian optimization, would multiply the number of design decisions (kernels, search utilities, priors, and scales) and thus significantly complicate the analysis.

The individual sampling distributions for the hyperparameters significantly affect the relative rankings of the optimizers. A badly chosen search space can make tuning next to impossible. Note, though, that this problem is inherited by practitioners. It is arguably an implicit flaw of an optimizer to not come with well-identified search spaces for its hyperparameters.

5 CONCLUSION

The past years have seen a surge of new stochastic optimization methods. Practitioners are left with the impossible task of not just picking a method from this ever-growing list, but also to guess or tune hyperparameters for them, even to continuously tune them during optimization. Despite valiant efforts by the community, there is currently no method that clearly dominates the competition (be it in the sense of theoretical guarantees or clear empirical advantages).

We have provided an extensive empirical benchmark of optimization methods for deep learning. It reveals structure in the crowded field of optimization for deep learning. First, although many methods perform competitively, a subgroup of methods tends to come up near the top across the spectrum of problems. Secondly, tuning helps, about as much as trying other optimizers. Our open data set allows many further, more technical observations which we can not make here for space (e.g., as we discuss in the appendix, stability to re-runs is an often overlooked challenge).

Perhaps the most important takeaway from our study is hidden in plain sight: the field is in danger of being drowned by noise. We hope that benchmarks like ours can help the community to rise beyond inventing yet another optimizer and to focus on key challenges, such as automatic, inner-loop tuning for truly robust and efficient optimization. We are releasing our data to allow future authors to ensure that their method contributes to such ends.

REFERENCES

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.

Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Disentangling adaptive gradient methods from learning rates, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On Empirical Comparisons of Optimizers for Deep Learning, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C. Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies, 2020.

Frank Schneider, Lukas Balles, and Philipp Hennig. DeepOBS: A Deep Learning Optimizer Benchmark Suite. In *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–14, 2019.

Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and François Fleuret. On the Tunability of Optimizers in Deep Learning, 2019.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4148–4158, 2017. URL http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.

Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A Walk with SGD: How SGD Explores Regions of Deep Network Loss?, 2018.

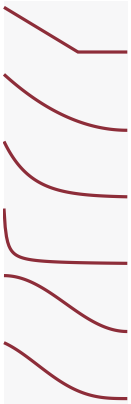
Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. https://d2l.ai.

A LIST OF OPTIMIZERS AND SCHEDULES CONSIDERED

Table 2: List of optimizers we considered for our benchmark. Note, that this is still far from being a complete list of all existing optimization methods applicable to deep learning, but only a subset, comprising of some of the most popular choices.

Name	Ref.	Name	Ref.
ACClip	(Savarese et al., 2019)	LARS	(You et al., 2017)
ADAHESSIAN	(Yao et al., 2020)	LookAhead	(Zhang et al., 2019)
ALI-G	(Berrada et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AMSBound	(Luo et al., 2019)	MTAdam	(Malkiel & Wolf, 2020)
AMSGrad	(Reddi et al., 2018)	Nesterov	(Nesterov, 1983)
ARSG	(Chen et al., 2019b)	Polyak	(Polyak, 1964)
AcceleGrad	(Levy et al., 2018)	MVRC-1	(Chen & Zhou, 2020)
AdaAlter	(Xie et al., 2019)	MVRC-2	(Chen & Zhou, 2020)
AdaBatch	(Devarakonda et al., 2017)	NAMSB	(Chen et al., 2019b)
AdaBayes	(Aitchison, 2018)	NAMSG	(Chen et al., 2019b)
AdaBayes-SS	(Aitchison, 2018)	ND-Adam	(Zhang et al., 2017)
AdaBlock	(Yun et al., 2019)	Nadam	(Dozat, 2016)
AdaBound	(Luo et al., 2019)	Noisy Adam	(Zhang et al., 2018)
AdaComp	(Chen et al., 2018)	Noisy K-FAC	(Zhang et al., 2018)
AdaFTRL	(Orabona & Pál, 2015)	NosAdam	(Huang et al., 2019)
AdaFix	(Bae et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdaFom	(Chen et al., 2019a)	PAGE	(Li et al., 2020b)
AdaLoss	(Teixeira et al., 2019)	PAL	(Mutschler & Zell, 2019)
AdaMax	(Kingma & Ba, 2015)	Padam	(Mittal et al., 2019)
AdaMod	(Ding et al., 2019)	PolyAdam	(Orvieto et al., 2019)
AdaScale	(Johnson et al., 2020)	PowerSGD	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	PowerSGDM	(Vogels et al., 2019)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci, 2018)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
AdaX	(Li et al., 2020a)	QHAdam	(Ma & Yarats, 2019)
AdaX-W	(Li et al., 2020a)	QHM	(Ma & Yarats, 2019)
Adadelta	(Zeiler, 2012)	RAdam	(Liu et al., 2020)
Adagrad	(Duchi et al., 2011)	RMSProp	(Tieleman & Hinton, 2012)
Adam	(Kingma & Ba, 2015)	RMSterov	(Choi et al., 2019)
AdamAL	(Tao et al., 2019)	Ranger	(Repository, b)
AdamNC	(Reddi et al., 2018)	RangerLars	(Repository, c)
AdamP	(Heo et al., 2020)	SAMSGrad	(Tong et al., 2019)
AdamT	(Zhou et al., 2020)	SAdam	(Wang et al., 2020b)
AdamW	(Loshchilov & Hutter, 2019)	SC-Adagrad	(Mukkamala & Hein, 2017)
AdamX	(Tran & Phong, 2019)	SC-RMSProp	(Mukkamala & Hein, 2017)
Adathm	(Sun et al., 2019)	SDProp	(Ida et al., 2017)
ArmijoLS	(Vaswani et al., 2019)	SGD	(Robbins & Monro, 1951)
AvaGrad	(Savarese et al., 2019)	SGD-BB	(Tan et al., 2016)
BAdam	(Salas et al., 2018)	SGD-G2	(Ayadi & Turinici, 2020)
BGAdam	(Bai & Zhang, 2019)	SGDM	(Liu & Luo, 2020)
BRMSProp	(Aitchison, 2018)	SGDP	(Heo et al., 2020)
BSGD	(Hu et al., 2020)	SGDR	(Loshchilov & Hutter, 2017)
C-ADAM	(Tutunov et al., 2020)	SHAdagrad	(Huang et al., 2020)
CProp	(Preechakul & Kijsirikul, 2019)	SKQN	(Yang et al., 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	S4QN	(Yang et al., 2020)
Compositional ADAM	(Tutunov et al., 2020)	SNGM	(Zhao et al., 2020)
Curveball	(Henriques et al., 2019)	SRSGD	(Wang et al., 2020a)
Dadam	(Nazari et al., 2019)	SWATS	(Keskar & Socher, 2017)
DeepMemory	(Repository, a)	SWNTS	(Chen et al., 2019c)
DiffGrad	(Dubey et al., 2020)	SADAM	(Tong et al., 2019)
Eve	(Longhi, 2017)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Gadam	(Zhang & Gouza, 2018)	SignAdam++	(Wang et al., 2019a)
GOLS-I	(Kafka & Wilke, 2019)	SignSGD	(Bernstein et al., 2018)
Gravilon	(Kelterborn et al., 2020)	SoftAdam	(Fetterman et al., 2019)
HAdam	(Jiang et al., 2019)	S-SGD	(Sung et al., 2020)
HyperAdam	(Wang et al., 2019b)	TAdam	(Ilboudo et al., 2020)
K-BFGS	(Goldfarb et al., 2020)	VAdam	(Khan et al., 2018)
K-BFGS(L)	(Goldfarb et al., 2020)	VR-SGD	(Shang et al., 2020)
KFAC	(Martens & Grosse, 2015)	WNGrad	(Wu et al., 2018)
KFLR	(Botev et al., 2017)	YellowFin	(Zhang & Mitliagkas, 2019)
KFRA	(Botev et al., 2017)	Yogi	(Zaheer et al., 2018)
L4Adam	(Rolínek & Martius, 2018)	vSGD-b	(Schaul et al., 2013)
L4Momentum	(Rolínek & Martius, 2018)	vSGD-fd	(Schaul & LeCun, 2013)
LAMB	(You et al., 2019)	vSGD-g	(Schaul et al., 2013)
LaProp	(Ziyin et al., 2020)	vSGD-l	(Schaul et al., 2013)

Table 3: Overview of commonly used parameter schedules. Note, while we list the schedules parameters, it isn't clearly defined what aspects of a schedule are (tunable) parameters and what is a-priori fixed. In this column, α_0 denotes the initial learning rate, α_{lo} and α_{up} the lower and upper bound, Δt indicates an epoch count at which to switch decay styles, k denotes a decaying factor.

Name		**Ref.**	**Illustration**	**Parameters**
Constant				α_0
Step Decay	constant factor			$\alpha_0, \Delta t_1, \ldots, k$
	multi-step			$\alpha_0, \Delta t_1, \ldots,$ $k_1, \ldots$
Smooth Decay	linear decay	e.g. (Goodfellow et al., 2016)		$\alpha_0, (\Delta t, \alpha_{\text{lo}})$
	polynomial decay			$\alpha_0, k, (\alpha_{\text{lo}})$
	exponential decay			$\alpha_0, k, (\alpha_{\text{lo}})$
	inverse time decay	e.g. (Bottou, 2012)		$\alpha_0, k, (\alpha_{\text{lo}})$
	cosine decay	(Loshchilov & Hutter, 2017)		$\alpha_0, (\alpha_{\text{lo}})$
	linear cosine decay	(Bello et al., 2017)		$\alpha_0, (\alpha_{\text{lo}})$
Cyclical	triangular	(Smith, 2017)		$\alpha_{\text{lo}}, \alpha_{\text{up}}, \Delta t$
	triangular + decay	(Smith, 2017)		$\alpha_{\text{lo}}, \alpha_{\text{up}}, \Delta t, k$
	triangular + exponential decay	(Smith, 2017)		$\alpha_{\text{lo}}, \alpha_{\text{up}}, \Delta t$
	cosine + warm restarts	(Loshchilov & Hutter, 2017)		$\alpha_{\text{up}}, \Delta t, (\alpha_{\text{lo}})$
	cosine + warm restarts + decay	(Loshchilov & Hutter, 2017)		$\alpha_{\text{up}}, \Delta t, k, (\alpha_{\text{lo}})$
Warmup	constant warmup	e.g. (He et al., 2016)		$\alpha_{\text{lo}}, \alpha_0, \Delta t$
	gradual warmup	(Goyal et al., 2017)		$\alpha_0, \Delta t, (\alpha_{\text{lo}})$
	gradual warmup + multi-step decay	(Goyal et al., 2017)		$\alpha_0, \Delta t, \Delta t_{\text{steps}}, k_1, \ldots, (\alpha_{\text{lo}})$
	gradual warmup + step number decay	(Vaswani et al., 2017)		$\alpha_0, \Delta t, (\alpha_{\text{lo}})$
	slanted triangular	(Howard & Ruder, 2018)		$\alpha_0, \Delta t, (\alpha_{\text{lo}})$
	long trapezoid	(Xing et al., 2018)		$\alpha_0, \Delta t_{\text{up}}, \Delta t_{\text{down}}, (\alpha_{\text{lo}})$
Super-Convergence	1cycle	(Smith & Topin, 2017)		$\alpha_{\text{up}}, \Delta t, \Delta t_{\text{cutoff}}, (\alpha_{\text{lo}})$

B LIST OF OPTIMIZERS SELECTED

Table 4: Selected optimizers for our benchmarking process with their respective color, hyperparameters, default values, tuning distributions and scheduled hyperparameters. Here, $\mathcal{LU}(\cdot,\cdot)$ denotes the log-uniform distribution while $\mathcal{U}\{\cdot,\cdot\}$ denotes the discrete uniform distribution.

Optimizer	Ref.	Parameters	Default	Tuning Distribution	Scheduled
AMSBound	(Luo et al., 2019)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		α_l	0.1	$\mathcal{LU}(10^{-3}, 0.5)$	
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		γ	10^{-3}	$\mathcal{LU}(10^{-4}, 10^{-1})$	
		ϵ	10^{-8}	✗	
AMSGrad	(Reddi et al., 2018)	α	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		ϵ	10^{-8}	✗	
AdaBound	(Luo et al., 2019)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		α_l	0.1	$\mathcal{LU}(10^{-3}, 0.5)$	
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		γ	10^{-3}	$\mathcal{LU}(10^{-4}, 10^{-1})$	
		ϵ	10^{-8}	✗	
Adadelta	(Zeiler, 2012)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		ϵ	10^{-8}	✗	
		ρ	0.95	$\mathcal{LU}(10^{-3}, 1)$	
Adagrad	(Duchi et al., 2011)	α	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓
		ϵ	10^{-7}	✗	
Adam	(Kingma & Ba, 2015)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		ϵ	10^{-8}	✗	
Lookahead Momentum abbr. LA(Mom.)	(Zhang et al., 2019)	α	0.5	$\mathcal{LU}(10^{-4}, 1)$	
		α_f	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓
		k	5	$\mathcal{U}\{1, 20\}$	
		ρ	0.99	$\mathcal{LU}(10^{-3}, 1)$	
Lookahead RAdam abbr. LA(RAdam)	(Zhang et al., 2019)	α	0.5	$\mathcal{LU}(10^{-4}, 1)$	
		α_f	10^{-3}	$\mathcal{LU}(1e-4, 1)$	✓
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		ϵ	10^{-7}	✗	
		k	5	$\mathcal{U}\{1, 20\}$	
Momentum	(Polyak, 1964)	α	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓
		ρ	0.99	$\mathcal{LU}(10^{-3}, 1)$	
NAG	(Nesterov, 1983)	α	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓
		ρ	0.99	$\mathcal{LU}(10^{-3}, 1)$	
NAdam	(Dozat, 2016)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		ϵ	10^{-7}	✗	
RAdam	(Liu et al., 2020)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		β_1	0.9	$\mathcal{LU}(0.5, 0.999)$	
		β_2	0.999	$\mathcal{LU}(0.8, 0.999)$	
		ϵ	10^{-7}	✗	
RMSProp	(Tieleman & Hinton, 2012)	α	10^{-3}	$\mathcal{LU}(10^{-4}, 1)$	✓
		ϵ	10^{-10}	✗	
		ρ	0.9	$\mathcal{LU}(10^{-3}, 1)$	
SGD	(Robbins & Monro, 1951)	α	10^{-2}	$\mathcal{LU}(10^{-4}, 1)$	✓

C ROBUSTNESS TO RANDOM SEEDS

Data subsampling, random weight initialization, dropout and other aspects of deep learning introduce stochasticity to the training process. As such, judging the performance of an optimizer on a single run may be misleading due to random fluctuations. In our benchmark we use 10 different seeds of the final setting for each budget in order to judge the stability of the optimizer and the results. However, to keep the magnitude of this benchmark feasible, we only use a single seed while tuning, analogously to how a single user would progress. This means that our tuning process can sometimes choose hyperparameter settings which might not even converge for seeds other than the one used for tuning.

Figure 5 illustrates this behavior on an example problem where we used 10 seeds throughout a tuning process using grid search. The figure shows that in the beginning performance increases when increasing the learning rate, followed by an area were it sometimes works but other times diverges. Picking hyperparameters from this "danger zone" can lead to unstable results. In this case, where we only consider the learning rate, it is clear that decreasing the learning rate a bit to get away from this "danger zone" would lead to a more stable, but equally well-performing algorithm. In more complicated cases, however, we are unable to use a simple heuristic such as this. This might be the case, for example, when tuning multiple hyperparameters or when the effect of the hyperparameter on the performance is less straight forward. Thus, this is a problem not created by improperly using the tuning method, but by an unstable optimization method.

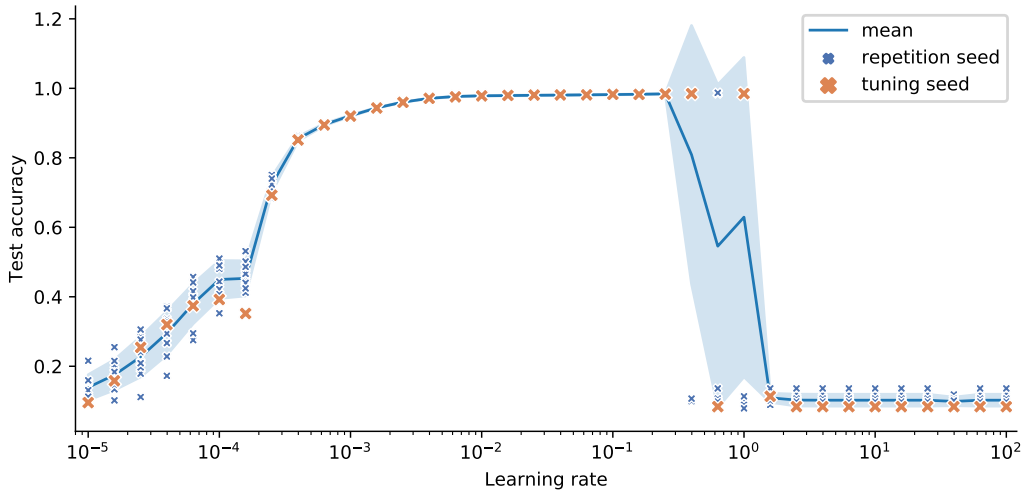


Figure 5: Performance of SGD on a simple multilayer perceptron. For each learning rate, markers in orange (✖) show the initial seed which would be used for tuning, blue markers (✖) illustrate nine additional seeds with otherwise unchanged settings. The mean over all seeds is plotted as a blue line (—), showing one standard deviation as a shaded area (▮).

In our benchmark, we observe in total 49 divergent seeds for the small budget and 56 for the large budget, or roughly 1% of the runs in each budget. Most of them occur when using SGD (23 and 18 cases for the small and large budget respectively), MOMENTUM (13 and 17 cases for the small and large budget respectively) or NAG (7 and 12 cases for the small and large budget respectively), which might indicate that adaptive methods are less prone to this kind of behavior. For the small budget tuning, none of these cases occur when using a constant schedule (4 for the large budget), and most of them occur when using the *cosine with warm restarts* schedule (27 and 25 cases for the small and large budget respectively). However, as our data on diverging seeds is very limited, it is not conclusive enough to draw solid conclusions.

D RE-TUNING EXPERIMENTS

In order to test the stability of our benchmark and especially the tuning method, we selected two optimizers in our benchmark and re-tuned them on all test problems a second time. We used completely independent random seeds for both tuning and the 10 repetitions with the final setting. Figure 6 and Figure 7 show the distribution of all 10 random seeds for both the original tuning as well as the re-tuning runs for RMSPROP and ADADELTA. It is evident, that re-tuning results in a shift of this distribution, since small (stochastic) changes during tuning can result in a different chosen hyperparameter setting.

These differences also highlight how crucial it is to look at multiple test problems. Individually, small changes, such as re-doing the tuning with different seeds can lead to optimization methods changing rankings. However, they tend to average out when looking at an unbiased list of multiple problems. These results also further supports the statement made in Section 3 that there is no optimization method clearly domination the competition, as small performance margins might vanish when re-tuning.

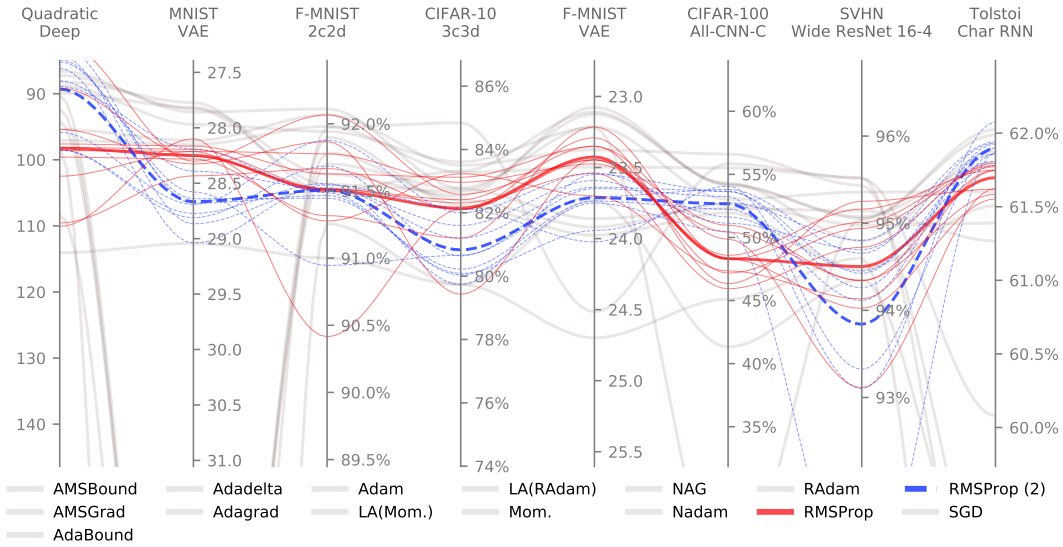


Figure 6: Mean test set performance of all 10 seeds of RMSPROP (—) on all eight optimization problems using the *small budget* for tuning and *no learning rate schedule*. The mean is shown with a thicker line. We repeated the full tuning process on all eight test problems using different random seeds, which is shown in dashed lines blue (- -). The mean performance of all other optimizers is shown in transparent gray lines.

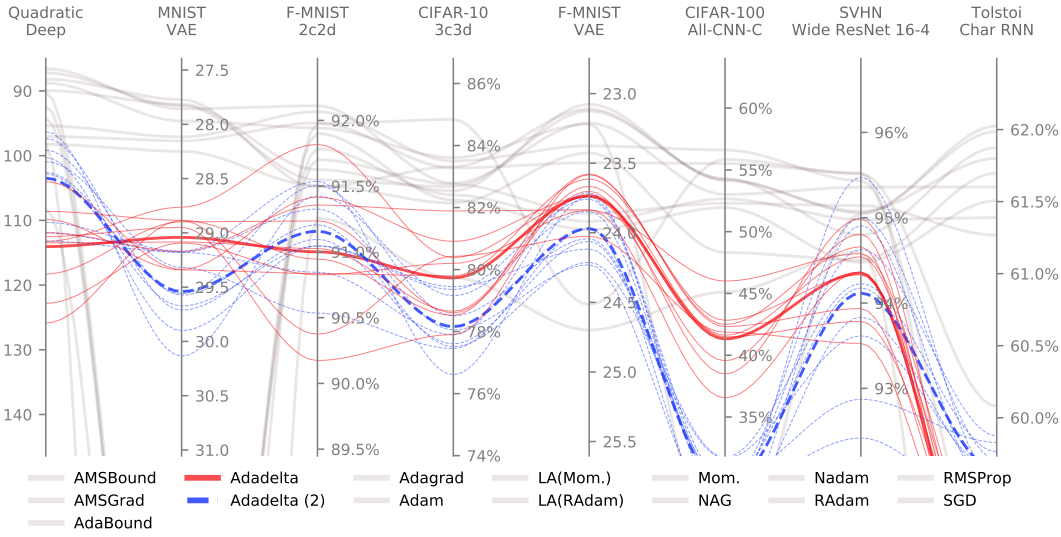


Figure 7: Mean test set performance of all 10 seeds of ADADELTA (—) on all eight optimization problems using the *small budget* for tuning and *no learning rate schedule*. The mean is shown with a thicker line. We repeated the full tuning process on all eight test problems using different random seeds, which is shown in dashed lines blue (- -). The mean performance of all other optimizers is shown in transparent gray lines.

E LIST OF SCHEDULES SELECTED

The schedules selected for our benchmark are illustrated in Figure 8. All learning rate schedules are multiplied by the initial learning rate found via tuning or picked as the default choice.

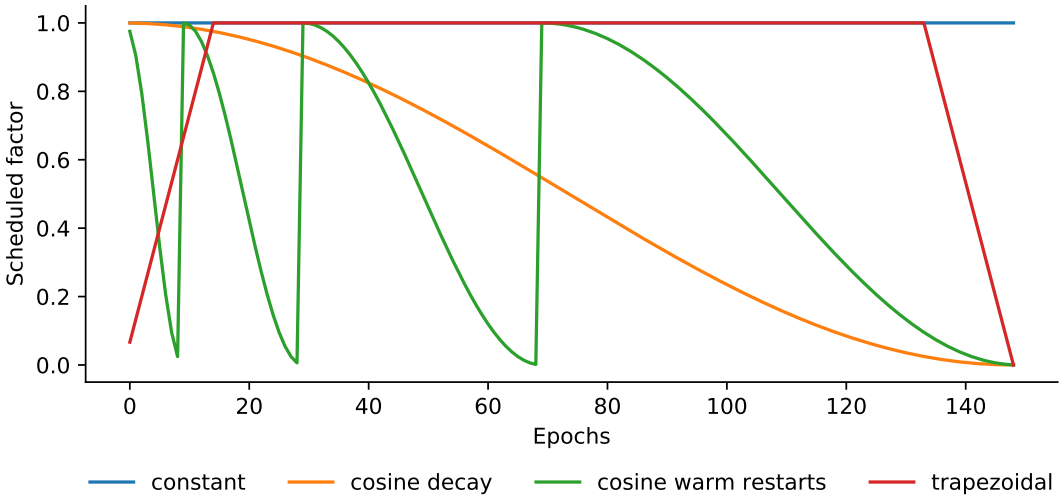


Figure 8: Illustration of the selected learning rate schedules for a training duration of 150 epochs.

We use a *cosine decay* (Loshchilov & Hutter, 2017) that starts at 1 and decays in the form of a half period of a cosine to 0. As an example of a cyclical learning rate schedule, we test a *cosine with warm restarts* schedule with a cycle length $\Delta t = 10$ which increases by a factor of 2 after each cycle without any discount factor. Depending on the number of epochs we train our model, it is possible that training stops shortly after one of those warm restarts. Since performance typically declines shortly after increasing the learning rate, we don't report the final performance for this schedule, but instead the performance achieved after the last complete period (just before the next restart). This approach is suggested by the original work of Loshchilov & Hutter (2017). However, we still use the final performance while tuning.

A representation of a schedule including warmup is the *trapezoidal* schedule from Xing et al. (2018). For our benchmark we set a warmup and cool-down period of $1/10$ the training time.

F IMPROVEMENT AFTER TUNING

When looking at Figure 2, one might realize that few diagonal entries contain negative values. Since diagonal entries reflect the intra-optimizer performance change when tuning on the respective task, this might feel quite counterintuitive at first. *In theory*, this can occur if the respective tuning distributions is chosen poorly, the tuning randomness simply got "unlucky", or we observe significantly worse results for our additional seeds (see Figure 5).

If we compare Figures 9 and 10 to Figures 11 and 12 we can see most negative diagonal entries vanish or at least diminish in magnitude. For the latter two figures we allow for more tuning runs and only consider the seed that has been used for this tuning process. The fact that the effect of negative diagonal entries reduces is an indication that they mostly result from the two latter reasons mentioned.

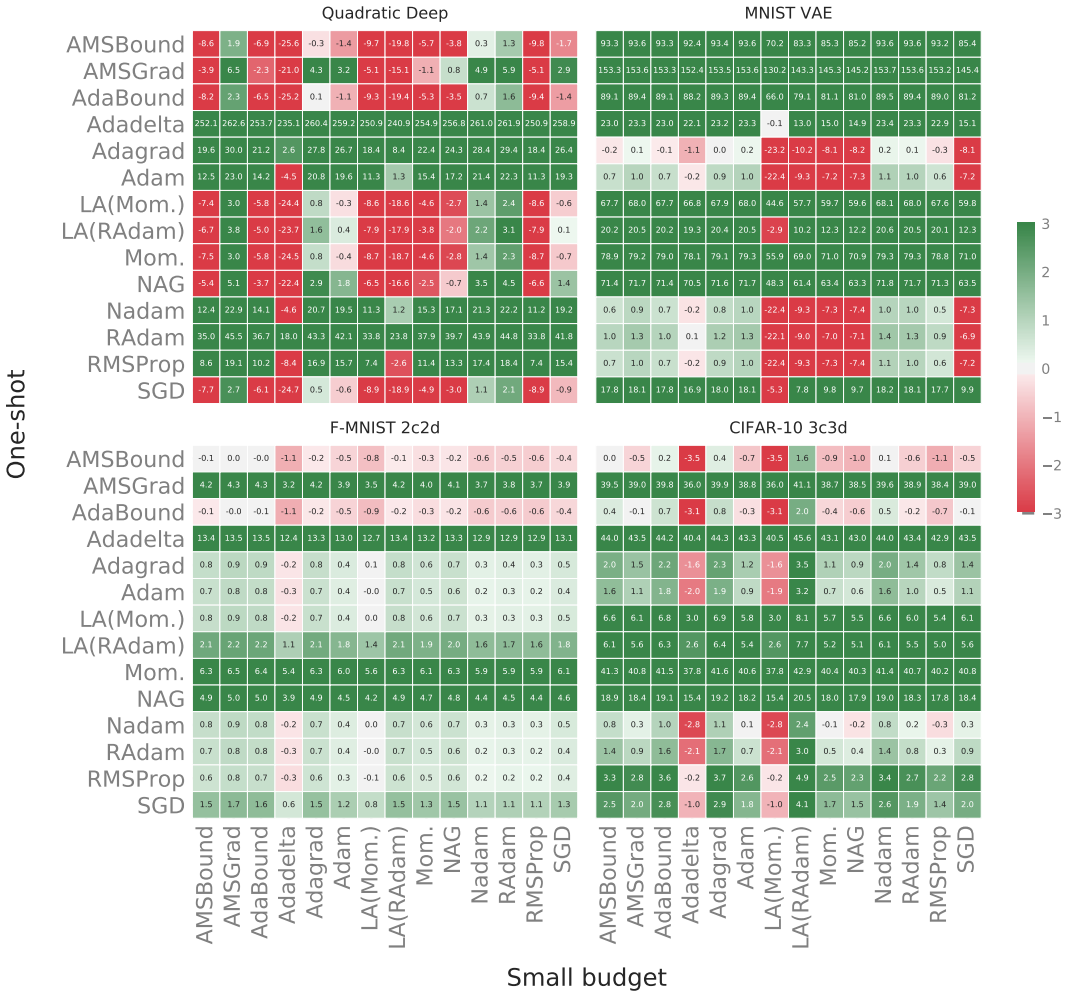


Figure 9: The absolute test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *small budget*) as an average over 10 random seeds for the *constant* schedule. This is a detailed version of Figure 2 in the main text showing the first four problems.

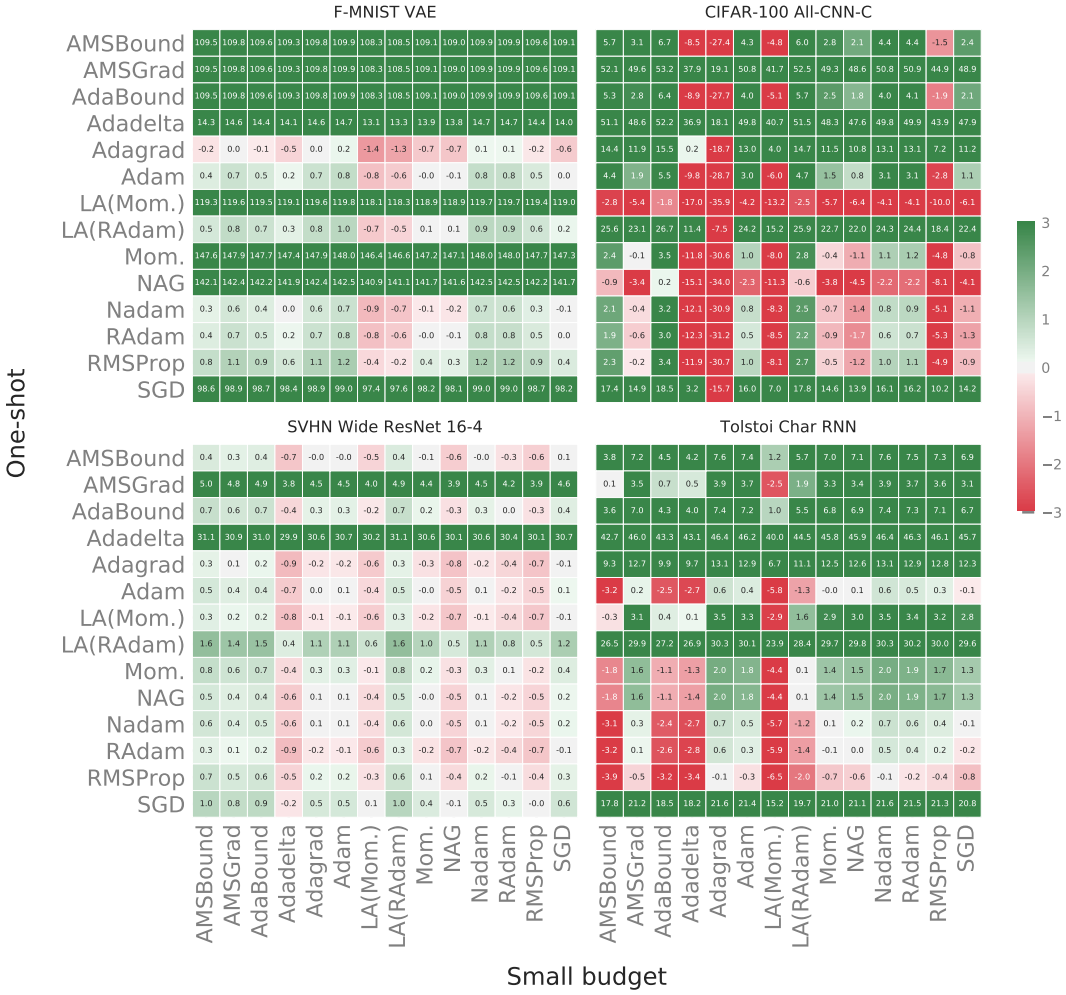


Figure 10: The absolute test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *small budget*) as an average over 10 random seeds for the *constant* schedule. This is a detailed version of Figure 2 in the main text showing the last four problems.

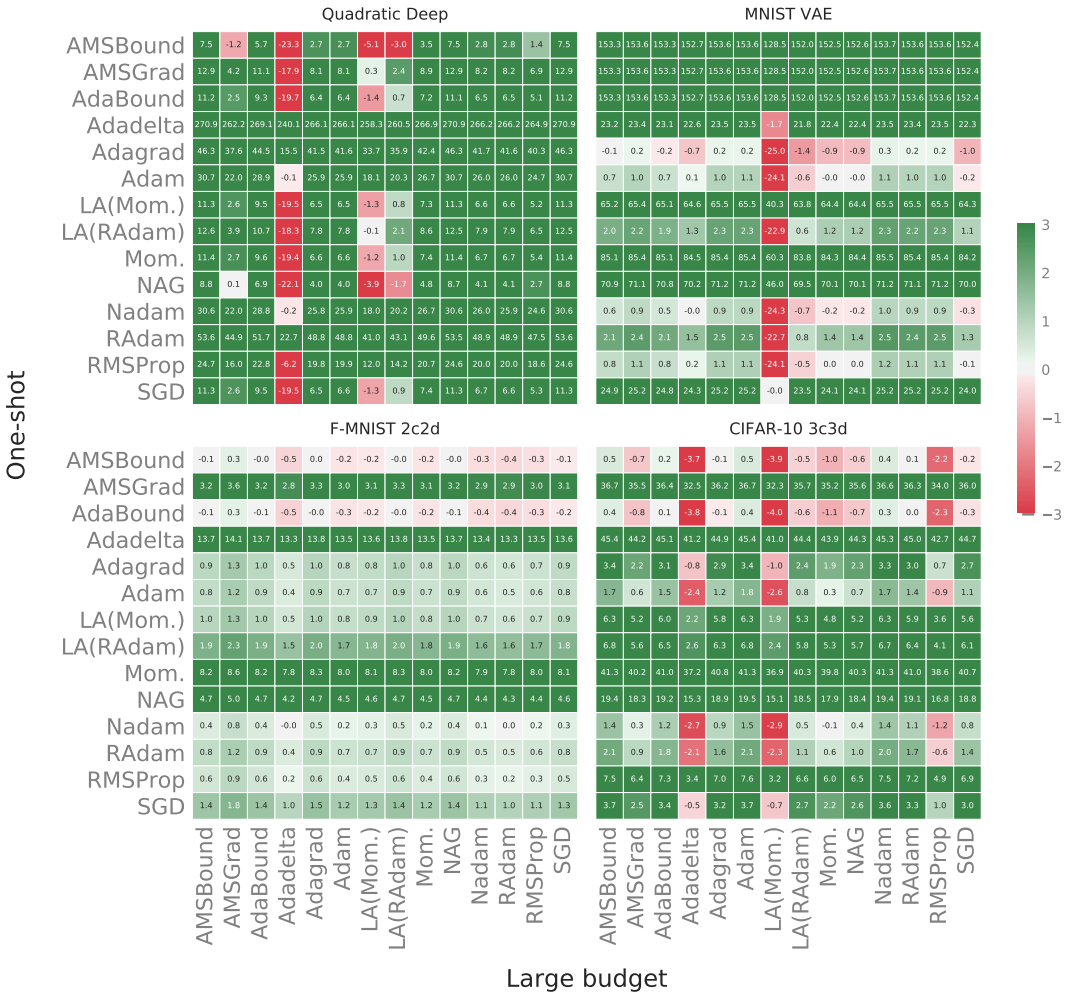


Figure 11: The absolute test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *large budget*) for the *constant* schedule. This is structurally the same plot as Figure 9 but comparing to the *large budget* and only considering the seed that has been used for tuning.

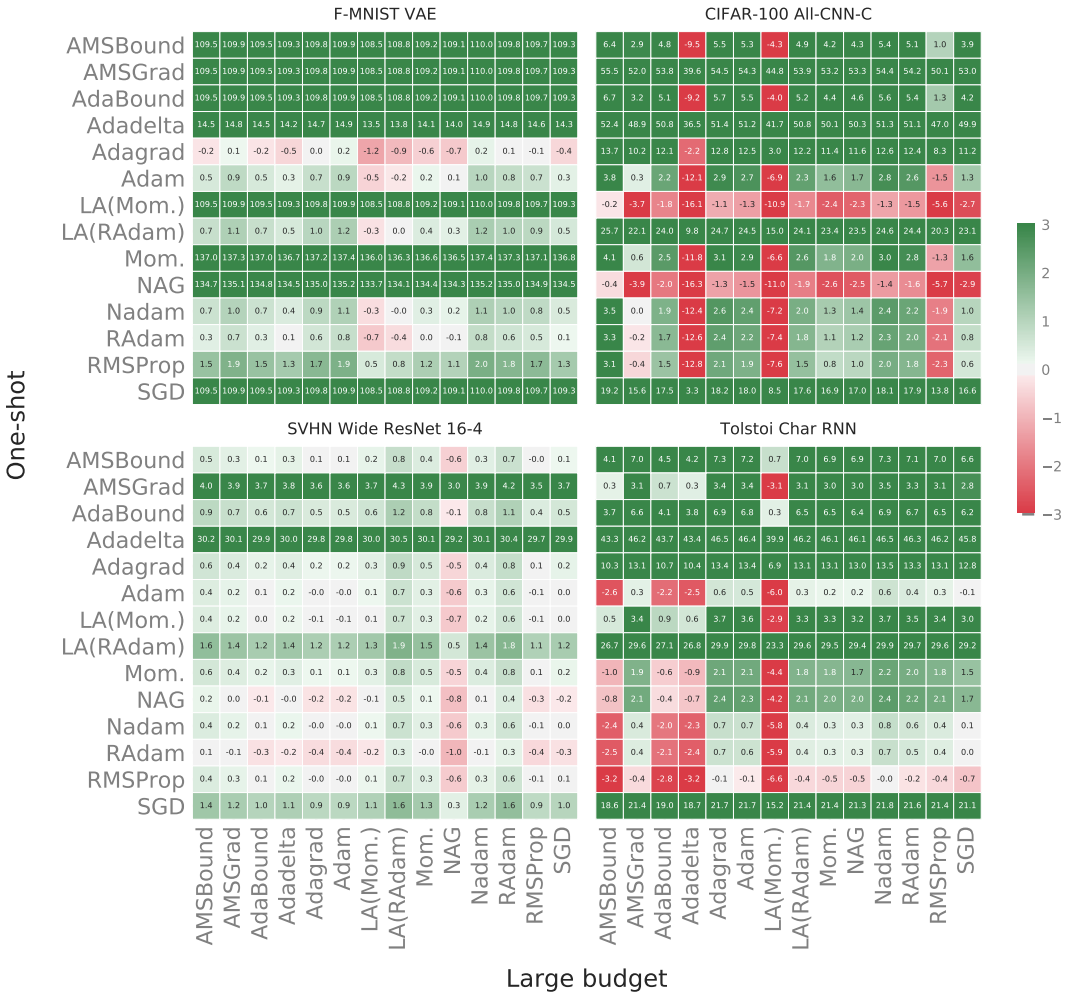


Figure 12: The absolute test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *large budget*) for the *constant* schedule. This is structurally the same plot as Figure 10 but comparing to the *large budget* and only considering the seed that has been used for tuning.

G OPTIMIZER PERFORMANCE ACROSS TEST PROBLEMS

Similarly to Figure 4, we show the corresponding plots for the *small budget* with *no learning rate schedule* in Figure 13 and the *large budget* with the *cosine* and *trapezoidal learning rate schedule* in Figures 14 and 15. Additionally, in Figure 16 we show the same setting as Figure 4 but showing the training loss instead of the test loss/accuracy.

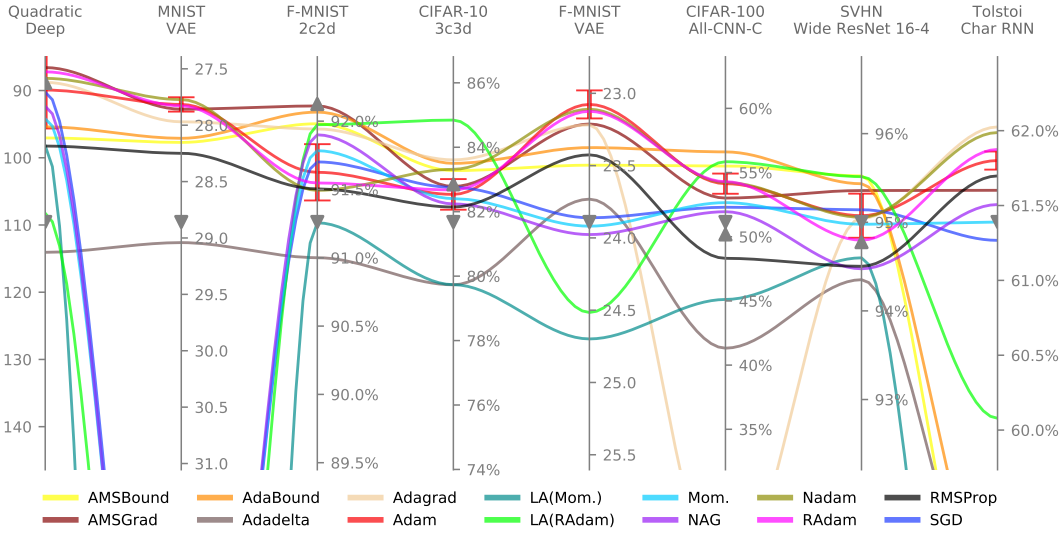


Figure 13: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *small budget* for tuning and *no learning rate schedule*. One standard deviation for the tuned ADAM optimizer is shown with a red error bar (I). The performance of the untuned versions of ADAM (▼) and ADABOUND (▲) are marked for reference. Note, the upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is based on the performance of ADAM with default parameters.

The high-level trends mentioned in Section 3 also hold for the smaller tuning budget in Figure 13. Namely, taking the winning optimizer for several untuned algorithms (here marked for ADAM and ADABOUND) will result in a decent performance in most test problems with much less effort. Adding a tuned version ADAM (or variants thereof) to this selection would result in a very competitive performance. The absolute top-performance however, is achieved by changing optimizers across different test problems.

Note, although the *large budget* is a true superset of the *small budget* it is not given that it will always perform better. Our tuning procedure guarantees that the *validation* performance on the seed that has been used for tuning is as least as good on the large budget than on the small budget. But due to averaging over multiple seeds and reporting *test* performance instead of *validation* performance, this hierarchy is no longer guaranteed. We discuss the possible effects of averaging over multiple seeds further in Appendix C.

The same high-level trends also emerge when considering the *cosine* or *trapezoidal learning rate schedule* in Figures 14 and 15. We can also see that the top performance in general increase when adding a schedule (cf. Figure 4 and Figure 15).

Comparing Figure 4 and Figure 16 we can assess the generalization performance of the optimization method not only to an unseen test set, but also to a different performance metric (accuracy instead of loss). Again, the overall picture of varying performance across different test problems remains consistent when considering the training loss performance. Similarily to the figures showing test set performance we cannot identify a clear winner, although ADAM ands its variants, such as RADAM perform near the top consistently. Note that while Figure 16 shows the training loss, the optimizers have still be tuned to achieve the best validation performance (i.e. accuracy if available, else the loss).

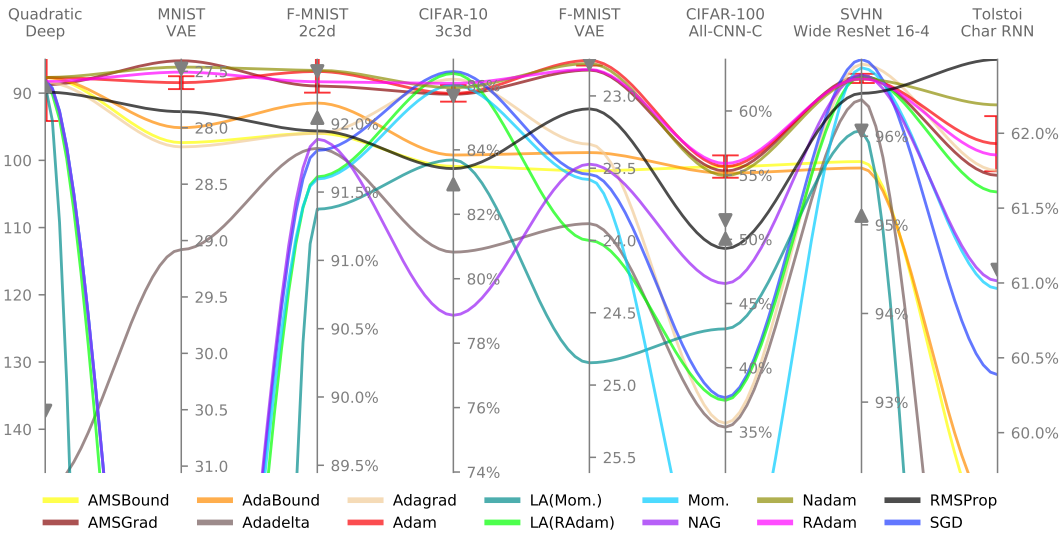


Figure 14: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and the *cosine learning rate schedule*. One standard deviation for the tuned ADAM optimizer is shown with a red error bar (I). The performance of the untuned versions of ADAM (▼) and ADABOUND (▲) are marked for reference (this time with the *cosine* learning rate schedule). Note, the upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is based on the performance of ADAM with default parameters (and no schedule).

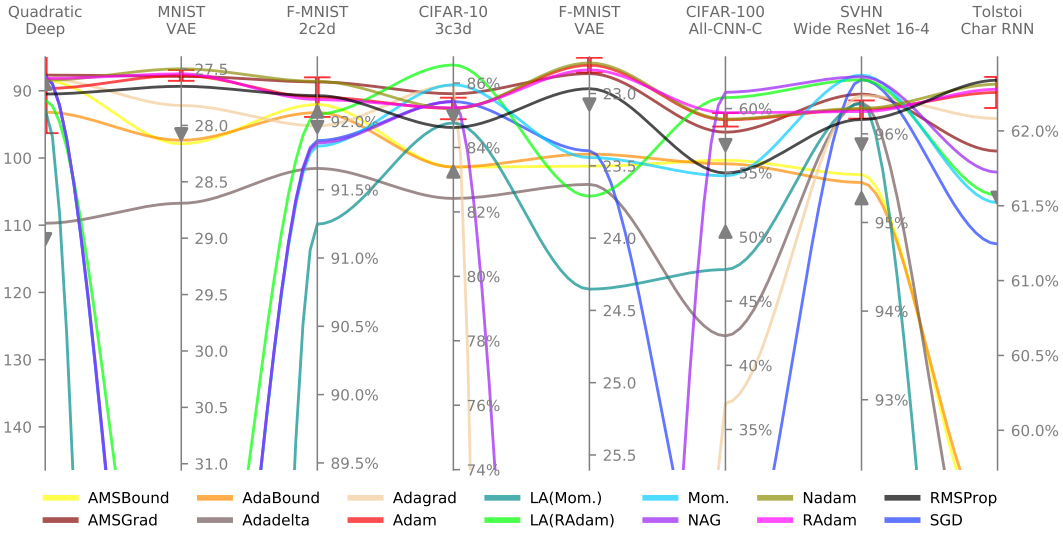


Figure 15: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and the *trapezoidal learning rate schedule*. One standard deviation for the tuned ADAM optimizer is shown with a red error bar (I). The performance of the untuned versions of ADAM (▼) and ADABOUND (▲) are marked for reference (this time with the *trapezoidal* learning rate schedule). Note, the upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is based on the performance of ADAM with default parameters (and no schedule).

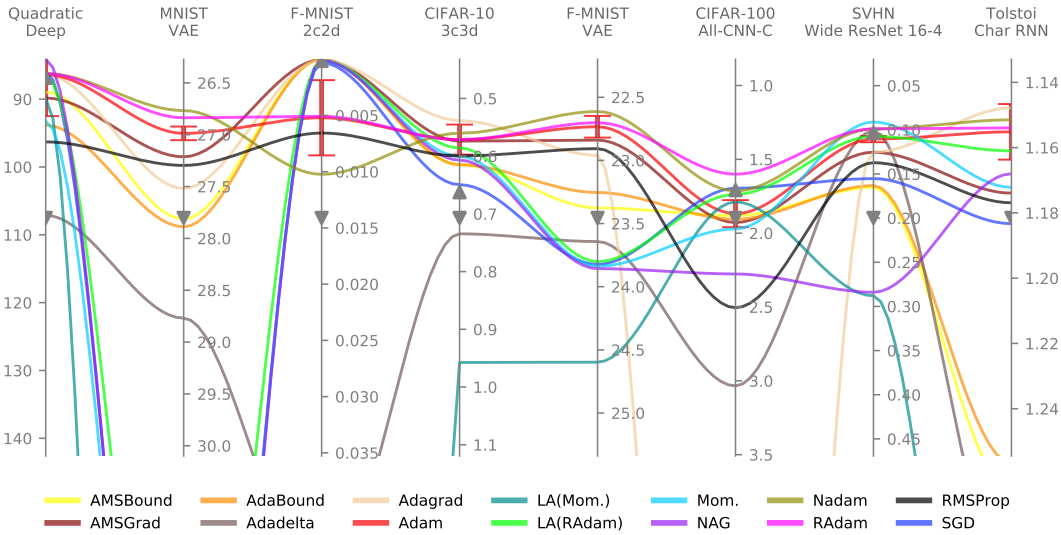


Figure 16: Mean *training* loss performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and *no learning rate schedule*. One standard deviation for the tuned ADAM optimizer is shown with a red error bar (I). The performance of the untuned versions of ADAM (▼) and ADABOUND (▲) are marked for reference. Note, the upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is based on the performance of ADAM with default parameters (and no schedule).

APPENDIX REFERENCES

Laurence Aitchison. Bayesian filtering unifies adaptive and non-adaptive neural network optimization methods, 2018.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Second order optimization made practical, 2020.

Imen Ayadi and Gabriel Turinici. Stochastic runge-kutta methods and adaptive SGD-G2 stochastic gradient descent, 2020.

Kiwook Bae, Heechang Ryu, and Hayong Shin. Does adam optimizer keep close to the optimal point?, 2019.

Jiyang Bai and Jiawei Zhang. Bgadam: Boosting based genetic-evolutionary adam for convolutional neural network optimization, 2019.

Lukas Balles and Philipp Hennig. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. In *35th International Conference on Machine Learning, ICML 2018*, 2018.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *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*, pp. 459–468. PMLR, 2017. URL http://proceedings.mlr.press/v70/bello17a.html.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. SIGNSGD: compressed optimisation for non-convex problems. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden*, volume 80 of *Proceedings of Machine Learning Research*, pp. 559–568. PMLR, 2018.

Leonard Berrada, Andrew Zisserman, and M. Pawan Kumar. Training neural networks for and by interpolation, 2019.

Oleksandr Borysenko and Maksym Byshkin. Coolmomentum: A method for stochastic optimization by langevin dynamics with simulated annealing, 2020.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 557–565, International Convention Centre, Sydney, Australia, 2017. PMLR.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pp. 421–436. Springer, 2012.

Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. Adacomp : Adaptive residual gradient compression for data-parallel distributed training. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2827–2835. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16859.

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of A class of adam-type algorithms for non-convex optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA*, 2019a.

Yushu Chen, Hao Jing, Wenlai Zhao, Zhiqiang Liu, Ouyi Li, Liang Qiao, Wei Xue, Haohuan Fu, and Guangwen Yang. An adaptive remote stochastic gradient method for training neural networks, 2019b.

Yushu Chen, Hao Jing, Wenlai Zhao, Zhiqiang Liu, Liang Qiao, Wei Xue, Haohuan Fu, and Guangwen Yang. NAMSG: an efficient method for training neural networks, 2019c.

Ziyi Chen and Yi Zhou. Momentum with variance reduction for nonconvex composition optimization, 2020.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On Empirical Comparisons of Optimizers for Deep Learning, 2019.

Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks, 2017.

Jianbang Ding, Xuancheng Ren, Ruixuan Luo, and Xu Sun. An adaptive and momental bound method for stochastic learning, 2019.

Timothy Dozat. Incorporating nesterov momentum into adam, 2016.

Shiv Ram Dubey, Soumendu Chakraborty, Swalpa Kumar Roy, Snehasis Mukherjee, Satish Kumar Singh, and Bidyut Baran Chaudhuri. diffGrad: An optimization method for convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2020.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Abraham J. Fetterman, Christina H. Kim, and Joshua Albrecht. Softadam: Unifying sgd and adam for better stochastic gradient descent, 2019.

Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, and Jonathan M. Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks, 2019.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In Jennifer G. Dy and Andreas Krause (eds.), *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*, pp. 1837–1845. PMLR, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pp. 770–778. IEEE Computer Society, 2016.

João F. Henriques, Sébastien Ehrhardt, Samuel Albanie, and Andrea Vedaldi. Small steps and giant leaps: Minimal newton solvers for deep learning. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 4762–4771. IEEE, 2019. doi: 10.1109/ICCV.2019.00486. URL https://doi.org/10.1109/ICCV.2019.00486.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Youngjung Uh, and Jung-Woo Ha. Slowing down the weight norm increase in momentum-based optimizers, 2020.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 328–339, 2018.

Yifan Hu, Siqi Zhang, Xin Chen, and Niao He. Biased stochastic gradient descent for conditional stochastic optimization, 2020.

Yuzheng Hu, Licong Lin, and Shange Tang. Second-order information in first-order optimization methods, 2019.

Haiwen Huang, Chang Wang, and Bin Dong. Nostalgic adam: Weighting more of the past gradients when designing the adaptive learning rate. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 2556–2562. ijcai.org, 2019.

Xunpeng Huang, Hao Zhou, Runxin Xu, Zhe Wang, and Lei Li. Adaptive gradient methods can be provably faster than sgd after finite epochs, 2020.

Yasutoshi Ida, Yasuhiro Fujiwara, and Sotetsu Iwamura. Adaptive learning rate via covariance matrix based preconditioning for deep neural networks. In Carles Sierra (ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 1923–1929. ijcai.org, 2017.

Wendyam Eric Lionel Ilboudo, Taisuke Kobayashi, and Kenji Sugimoto. Tadam: A robust stochastic gradient optimizer, 2020.

Zhanhong Jiang, Aditya Balu, Sin Yong Tan, Young M Lee, Chinmay Hegde, and Soumik Sarkar. On higher-order moments in adam, 2019.

Tyler B. Johnson, Pulkit Agrawal, Haijie Gu, and Carlos Guestrin. Adascale sgd: A user-friendly algorithm for distributed training, 2020.

Dominic Kafka and Daniel Wilke. Gradient-only line searches: An alternative to probabilistic line searches, 2019.

Chad Kelterborn, Marcin Mazur, and Bogdan V. Petrenko. Gravilon: Applications of a new gradient descent method to machine learning, 2020.

Nitish Shirish Keskar and Richard Socher. Improving Generalization Performance by Switching from Adam to SGD, 2017.

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In Jennifer G. Dy and Andreas Krause (eds.), *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*, pp. 2616–2625. PMLR, 2018.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

Kfir Yehuda Levy, Alp Yurtsever, and Volkan Cevher. Online adaptive methods, universality and acceleration. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 6501–6510, 2018.

Wenjie Li, Zhaoyang Zhang, Xinjiang Wang, and Ping Luo. Adax: Adaptive gradient descent with exponential long term memory, 2020a.

Zhize Li, Hongyan Bao, Xiangliang Zhang, and Peter Richtárik. Page: A simple and optimal probabilistic gradient estimator for nonconvex optimization, 2020b.

Liang Liu and Xiaopeng Luo. A new accelerated stochastic gradient method with momentum, 2020.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rkgz2aEKDr`.

Pietro Longhi. Wall crossing invariants from spectral networks. *Annales Henri Poincaré*, 19(3): 775–842, 2017.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive Gradient Methods with Dynamic Bound of Learning Rate. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and adam for deep learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Maren Mahsereci. *Probabilistic Approaches to Stochastic Optimization*. Ph.d. thesis, University of Tuebingen, 2018.

Itzik Malkiel and Lior Wolf. Mtadam: Automatic balancing of multiple training loss terms, 2020.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2408–2417. JMLR.org, 2015.

Harshal Mittal, Kartikey Pandey, and Yash Kant. ICLR reproducibility challenge report (padam : Closing the generalization gap of adaptive gradient methods in training deep neural networks), 2019.

Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. In Doina Precup and Yee Whye Teh (eds.), *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*, pp. 2545–2553. PMLR, 2017.

Maximus Mutschler and Andreas Zell. Parabolic approximation line search: An efficient and effective line search approach for dnns, 2019.

Parvin Nazari, Davoud Ataee Tarzanagh, and George Michailidis. Dadam: A consensus-based distributed adaptive gradient method for online optimization, 2019.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate O(1/kˆ2). *Soviet Mathematics Doklady*, 27:372–376, 1983.

Francesco Orabona and Dávid Pál. Scale-free algorithms for online linear optimization. In Kamalika Chaudhuri, Claudio Gentile, and Sandra Zilles (eds.), *Algorithmic Learning Theory - 26th International Conference, ALT 2015, Banff, AB, Canada, October 4-6, 2015, Proceedings*, volume 9355 of *Lecture Notes in Computer Science*, pp. 287–301. Springer, 2015.

Antonio Orvieto, Jonas Köhler, and Aurélien Lucchi. The role of memory in stochastic optimization. In Amir Globerson and Ricardo Silva (eds.), *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, pp. 128. AUAI Press, 2019.

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

Konpat Preechakul and Boonserm Kijsirikul. CProp: Adaptive Learning Rate Scaling from Past Gradient Conformity, 2019.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. In *6th International Conference on Learning Representations, ICLR 2018*, pp. 1–23, 2018.

Github Repository, a. URL `https://github.com/lessw2020/Best-Deep-Learning-Optimizers/tree/master/DeepMemory`.

Github Repository, b. URL `https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer`.

Github Repository, c. URL https://github.com/mgrankin/over9000.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Michal Rolínek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. *Advances in Neural Information Processing Systems*, 2018-Decem:6433–6443, 2018.

Arnold Salas, Samuel Kessler, Stefan Zohren, and Stephen Roberts. Practical bayesian learning of neural networks via adaptive subgradient methods, 2018.

Pedro Savarese, David McAllester, Sudarshan Babu, and Michael Maire. Domain-independent dominance of adaptive methods, 2019.

Tom Schaul and Yann LeCun. Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients. In Yoshua Bengio and Yann LeCun (eds.), *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Conference Track Proceedings*, 2013.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. 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*, pp. 343–351. JMLR.org, 2013.

Fanhua Shang, Kaiwen Zhou, Hongying Liu, James Cheng, Ivor W. Tsang, Lijun Zhang, Dacheng Tao, and Licheng Jiao. VR-SGD: A simple stochastic variance reduction method for machine learning. *IEEE Trans. Knowl. Data Eng.*, 32(1):188–202, 2020.

Leslie N. Smith. 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*, pp. 464–472. IEEE Computer Society, 2017. doi: 10.1109/WACV.2017.58. URL https://doi.org/10.1109/WACV.2017.58.

Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. 2017.

Huikang Sun, Lize Gu, and Bin Sun. Adathm: Adaptive gradient method based on estimates of third-order moments. In *Fourth IEEE International Conference on Data Science in Cyberspace, DSC 2019, Hangzhou, China, June 23-25, 2019*, pp. 361–366. IEEE, 2019.

Wonyong Sung, Iksoo Choi, Jinhwan Park, Seokhyun Choi, and Sungho Shin. S-sgd: Symmetrical stochastic gradient descent with weight noise injection for reaching flat minima, 2020.

Conghui Tan, Shiqian Ma, Yu-Hong Dai, and Yuqiu Qian. Barzilai-borwein step size for stochastic gradient descent. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 685–693, 2016.

Zeyi Tao, Qi Xia, and Qun Li. A new perspective in understanding of adam-type algorithms and beyond, 2019.

Brian Teixeira, Birgi Tamersoy, Vivek Singh, and Ankur Kapoor. Adaloss: Adaptive loss function for landmark localization, 2019.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude, 2012.

Qianqian Tong, Guannan Liang, and Jinbo Bi. Calibrating the adaptive learning rate to improve convergence of adam, 2019.

Phuong Thi Tran and Le Trieu Phong. On the convergence proof of amsgrad and a new version. *IEEE Access*, 7:61706–61716, 2019.

Rasul Tutunov, Minne Li, Alexander I. Cowen-Rivers, Jun Wang, and Haitham Bou-Ammar. Compositional adam: An adaptive compositional solver, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pp. 5999–6009. Neural information processing systems foundation, 2017.

Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates. (i), 2019.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 14236–14245, 2019.

Bao Wang, Tan M. Nguyen, Andrea L. Bertozzi, Richard G. Baraniuk, and Stanley J. Osher. Scheduled restart momentum for accelerated stochastic gradient descent, 2020a.

Dong Wang, Yicheng Liu, Wenwo Tang, Fanhua Shang, Hongying Liu, Qigong Sun, and Licheng Jiao. signadam++: Learning confidences for deep neural networks. In Panagiotis Papapetrou, Xueqi Cheng, and Qing He (eds.), *2019 International Conference on Data Mining Workshops, ICDM Workshops 2019, Beijing, China, November 8-11, 2019*, pp. 186–195. IEEE, 2019a.

Guanghui Wang, Shiyin Lu, Quan Cheng, Weiwei Tu, and Lijun Zhang. Sadam: A variant of adam for strongly convex functions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020b. URL `https://openreview.net/forum?id=rye5YaEtPr`.

Jiaxuan Wang and Jenna Wiens. Adasgd: Bridging the gap between sgd and adam, 2020.

Shipeng Wang, Jian Sun, and Zongben Xu. Hyperadam: A learnable task-adaptive adam for network training. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 5297–5304. AAAI Press, 2019b.

Xiaoxia Wu, Rachel Ward, and Léon Bottou. Wngrad: Learn the learning rate in gradient descent, 2018.

Cong Xie, Oluwasanmi Koyejo, Indranil Gupta, and Haibin Lin. Local adaalter: Communication-efficient stochastic gradient descent with adaptive learning rates, 2019.

Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A Walk with SGD: How SGD Explores Regions of Deep Network Loss?, 2018.

Yangyang Xu. Momentum-based variance-reduced proximal stochastic gradient method for composite nonconvex stochastic optimization, 2020.

Minghan Yang, Dong Xu, Yongfeng Li, Zaiwen Wen, and Mengyun Chen. Structured stochastic quasi-newton methods for large-scale optimization problems, 2020.

Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W. Mahoney. Adahessian: An adaptive second order optimizer for machine learning, 2020.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes, 2019.

Jihun Yun, Aurelie C. Lozano, and Eunho Yang. Stochastic gradient methods with block diagonal matrix adaptation, 2019.

Manzil Zaheer, Sashank J. Reddi, Devendra Singh Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 9815–9825, 2018.

Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method, 2012.

Guodong Zhang, Shengyang Sun, David Duvenaud, and Roger Grosse. Noisy natural gradient as variational inference. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5852–5861, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

Jian Zhang and Ioannis Mitliagkas. Yellowfin and the art of momentum tuning. In Ameet Talwalkar, Virginia Smith, and Matei Zaharia (eds.), *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019. URL `https://proceedings.mlsys.org/book/263.pdf`.

Jiawei Zhang and Fisher B. Gouza. Gadam: Genetic-evolutionary adam for deep neural network optimization, 2018.

Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead Optimizer: k steps forward, 1 step back. *Advances in Neural Information Processing Systems 32 (NIPS 2019)*, 2019.

Zijun Zhang, Lin Ma, Zongpeng Li, and Chuan Wu. Normalized direction-preserving adam, 2017.

Shen-Yi Zhao, Yin-Peng Xie, and Wu-Jun Li. Stochastic normalized gradient descent with momentum for large batch training, 2020.

Bingxin Zhou, Xuebin Zheng, and Junbin Gao. Adamt: A stochastic optimization with trend correction scheme, 2020.

Zhiming Zhou, Qingru Zhang, Guansong Lu, Hongwei Wang, Weinan Zhang, and Yong Yu. AdaShift: Decorrelation and Convergence of Adaptive Learning Rate Methods. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Liu Ziyin, Zhikang T. Wang, and Masahito Ueda. Laprop: a better way to combine momentum with adaptive gradient, 2020.