Descending through a Crowded Valley — Benchmarking Deep Learning Optimizers

Robin M. Schmidt [*1] **Frank Schneider** [*1] **Philipp Hennig** [1 2]

Abstract

Choosing the optimizer is considered to be among the most crucial design decisions in deep learning, 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 based on anecdotes. In this work, we aim to replace these anecdotes, if not with a conclusive ranking, then at least with evidence-backed heuristics. To do so, we perform an extensive, standardized benchmark of fifteen particularly popular deep learning optimizers while giving a concise overview of the wide range of possible choices. Analyzing more than $50{,}000$ 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 cannot discern an optimization method clearly dominating across all tested tasks, we identify a significantly reduced subset of specific optimizers and parameter choices that generally lead to competitive results in our experiments: ADAM remains a strong contender, with newer methods failing to significantly and consistently outperform it. Our open-sourced results[1] are available as challenging and well-tuned baselines for more meaningful evaluations of novel optimization methods without requiring any further computational efforts.

[*]Equal contribution [1]Methods of Machine Learning, University of Tübingen, Tübingen, Germany [2]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Robin M. Schmidt <robin.schmidt.97@web.de>, Frank Schneider <f.schneider@uni-tuebingen.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]https://github.com/SirRob1997/Crowded-Valley---Results

1. Introduction

Large-scale stochastic optimization drives a wide variety of machine learning tasks. Because choosing the right optimization method and effectively tuning its hyperparameters heavily influences the training speed and final performance of the learned model, it is an important, every-day challenge to practitioners. It is probably the task that requires the most time and resources in many applications. Hence, stochastic optimization has been a focal point of research, engendering an ever-growing list of methods (cf. Figure 1), many of them targeted at deep learning. The hypothetical machine learning practitioner who is able to keep up with the literature now has the choice among hundreds of methods (see Table 2 in the appendix), each with their own set of tunable hyperparameters, when deciding how to train a model.

There is limited theoretical analysis that clearly favors one of these choices over the others. Some authors have offered empirical comparisons on comparably small sets of popular methods (e.g. Wilson et al., 2017; Choi et al., 2019; Sivaprasad et al., 2020); but for most optimizers, the only 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 with colleagues or on social media. The result is an often unclear, ever-changing "state of the art" occasionally driven by hype. 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.

We conduct a large-scale benchmark of optimizers to ground the ongoing debate about deep learning optimizers on empirical evidence, and to help understand how the choice of optimization methods and hyperparameters influences the training performance. Specifically, we examine whether recently proposed methods show an improved performance compared to more established methods such as SGD or ADAM. Additionally, we assess whether there exist optimization methods with well-working default hyperparameters that are able to keep up with tuned optimizers. To this end, we evaluate fifteen optimization methods, selected for

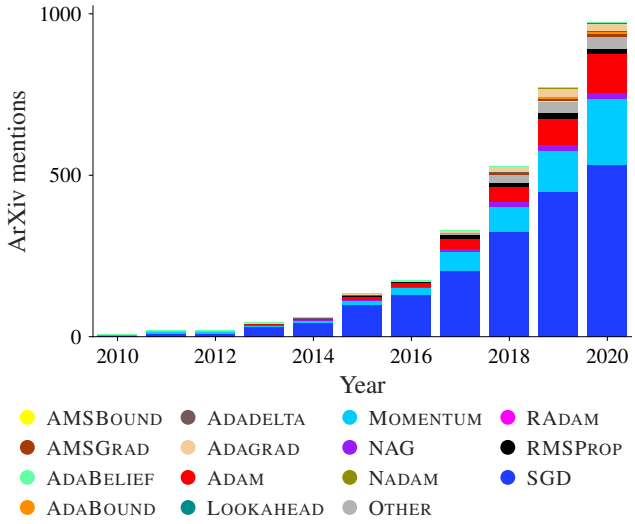

Figure 1: Number of times ArXiv titles and abstracts mention specific optimizer per year. All non-selected optimizers from Table 2 in the appendix are grouped into *Other*. This figure illustrates not only the expected increase in both methods and mentions, but also that our selection covers the most popular methods. In 2020, the excluded methods accounted for $< 4\,\%$ of the mentions (see Figure 9).

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 that it is impossible to include all optimizers (see Table 2 in the appendix), and to satisfy any and all expectations readers may have on tuning, initialization, or the choice of 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 the target audience 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 of our work are:

1. Assessing the progress in deep learning optimization. A literature review provides a compact but extensive list of recent advances in stochastic optimization. We identify more than a hundred optimization methods (see Table 2 in the appendix) and more than 20 families of hyperparameter schedules (see Table 3 in the appendix) proposed for deep learning. We conduct a large-scale optimizer benchmark, specifically focusing on problems arising in deep learning. We evaluate fifteen 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 to date (see Section 1.1 on related work).

2. Insights from more than 50,000 optimization runs. Our empirical experiments indicate that an optimizer's performance highly depends on the 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 problem. (3) While there is no optimizer that clearly dominates across all tested workloads, some of the methods we tested exhibited highly variable performance. Others demonstrated decent performance consistently. We deliberately abstain from recommending a single one among them, because we could not find a clear winner with statistical confidence.

3. An open-source baseline for future optimizer benchmarks and meta-learning approaches. Our results are available in an open and easily accessible form (see Footnote 1 on Page 1). This data set contains 53,760 unique runs, each consisting of thousands of individual data points, such as the mini-batch training losses of every iteration or epoch-wise performance measures, for example, the loss on the full validation set or test set accuracy. These results can be used as competitive and well-tuned baselines for future benchmarks of new optimizers, drastically reducing the amount of computational budget required for a meaningful optimizer comparison. This collection of training curves could also be used for meta-learning novel optimization methods, hyperparameter search strategies, or hyperparameter adaptation strategies. To encourage researches to contribute to this collection, we made our baselines easily expandable. [1]

The high-level result of our benchmark is, perhaps expectedly, *not* a clear winner. Instead, our comparison shows that, while some optimizers are frequently decent, they also generally perform similarly, often switching their positions in the ranking. This result is reminiscent, albeit not formally a rigorous result of the No Free Lunch Theorem (Wolpert & Macready, 1997). A key insight of our comparison is that a practitioner with a new 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 set of optimizers with their *default* settings 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

Following the rapid increase in publications on optimizers, *benchmarking* these methods for the application in deep learning has only recently attracted significant interest. Schneider et al. (2019) introduced a benchmarking framework called DEEPOBS, which includes a wide range of realistic deep learning problems together with standardized procedures for evaluating optimizers. Metz et al. (2020) presented TASKSET, another collection of optimization problems focusing on smaller but more numerous 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* optimizers or hyperparameters.

Both Choi et al. (2019) and Sivaprasad et al. (2020) analyzed specific aspects of the benchmarking process. Sivaprasad et al. (2020) used DEEPOBS to illustrate that the relative performance of an optimizer depends significantly on the used hyperparameter tuning budget. 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 defined a hyperparameter search space *per optimizer and problem* basing it either on prior published results, prior experiences, or pre-tuning trials.

Here, we instead aim to identify well-performing general-purpose optimizers for deep learning, especially when there is no prior knowledge about well-working hyperparameter values for each specific problem. We further elaborate on the influence of our chosen hyperparameter search strategy in Section 4 discussing the limitations of our empirical study.

Our work is also 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). By focusing on and reporting the *test set accuracy* we implicitly include the generalization capabilities of different optimizers in our benchmark results, an important characteristic of deep learning optimization.

2. Benchmarking process

Any benchmarking effort requires tricky decisions on the experimental setup that influence the results. Evaluating on a specific task or picking a certain tuning budget may favor or disadvantage certain methods (Sivaprasad et al., 2020). It is impossible to avoid these decisions or to cover all possible choices. Aiming for generality, we evaluate the performance on eight diverse real-world deep learning problems from different disciplines (Section 2.1). From a collection of more than a hundred deep learning optimizers (Table 2 in the appendix) we select fifteen of the most popular choices (see Figure 1) for this benchmark (Section 2.2). For each problem and optimizer we evaluate all possible combinations of four different tuning budgets (Section 2.3) and four selected learning rate schedules (Section 2.4), covering the following combinatorial space:

$$\underset{\text{Problem}}{\left\{\begin{matrix}\text{P1}\\ \text{P2}\\ \dots\\ \text{P8}\end{matrix}\right\}_{8}} \times \underset{\text{Optimizer}}{\left\{\begin{matrix}\text{ADAM}\\ \text{NAG}\\ \dots\\ \text{SGD}\end{matrix}\right\}_{15}} \times \underset{\text{Tuning}}{\left\{\begin{matrix}\text{one-shot}\\ \text{small}\\ \text{medium}\\ \text{large}\end{matrix}\right\}_{4}} \times \underset{\text{Schedule}}{\left\{\begin{matrix}\text{constant}\\ \text{cosine}\\ \text{cosine wr}\\ \text{trapez.}\end{matrix}\right\}_{4}} .$$

Combining those options results in 1,920 configurations, where each of the fifteen optimizers is evaluated in 128 settings (i.e. on *eight* problems, with *four* budgets and *four* schedules). Including hyperparameter search and estimating the confidence interval, our main benchmark consists of 53,760 unique training curves.

2.1. Problems

We consider the eight optimization tasks summarized in Table 1, available as the "small" (P1–P4) and "large" (P5–P8) problem sets in DEEPOBS. A detailed description of these problems, including architectures, training parameters, etc. can be found in the work of Schneider et al. (2019).[2] DEEPOBS provides several performance metrics, including the training and test loss, and the validation accuracy. 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 accuracy is not defined for this problem). This metric captures the optimizer's ability to generalize and is thus highly relevant for practical use. Our publicly released results include all metrics for completeness. An example of training loss performance is shown in Figure 17 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

In Table 2 in the appendix we collect over a hundred optimization methods introduced for or used in deep learning. This list was collected by multiple researchers trying to keep up with the field over recent years. It is thus necessarily

[2] All experiments were performed using version `1.2.0-beta` of DEEPOBS and TensorFlow version `1.15` (Abadi et al., 2015).

Table 1: Summary of problems used in our experiments. Exact model configurations can be found in Schneider et al. (2019).

	Data set	Model	Task	Metric	Batch size	Budget *in epochs*	Approx. run time[3]
P1	Artificial	Noisy quadratic	Minimization	Loss	128	100	< 1 min
P2	MNIST	VAE	Generative	Loss	64	50	10 min
P3	Fashion-MNIST	Simple CNN: *2c2d*	Classification	Accuracy	128	100	20 min
P4	CIFAR-10	Simple CNN: *3c3d*	Classification	Accuracy	128	100	35 min
P5	Fashion-MNIST	VAE	Generative	Loss	64	100	20 min
P6	CIFAR-100	*All-CNN-C*	Classification	Accuracy	256	350	4 h 00 min
P7	SVHN	*Wide ResNet 16-4*	Classification	Accuracy	128	160	3 h 30 min
P8	War and Peace	RNN	Character Prediction	Accuracy	50	200	5 h 30 min

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 benchmark with the degrees of freedom collected above. This is a serious problem for research: Even an author of a new optimizer, let alone a practitioner, cannot be expected to compare their work with every possible previous method.

We thus select a subset of fifteen 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" methods, but are either commonly used by practitioners and researchers, or have recently generated attention. Our selection is focused on first-order optimization methods, both due to their prevalence for non-convex optimization problems in deep learning as well as to simplify the comparison. Whether there is a significant difference between these optimizers or if they are inherently redundant is one of the questions this work investigates.

Our list focuses on optimizers over optimization techniques, although the line between the two is admittedly blurry. Techniques such as averaging weights (Izmailov et al., 2018, e.g.) or ensemble methods (Garipov et al., 2018, e.g.) have been shown to be simple but effective at improving the optimization performance. Those methods, however, can be applied to all methods in our lists, similar to regularization techniques, learning rate schedules, or tuning method. We have, therefore, decided to omit them from Table 2.

2.3. Tuning

Budget Optimization methods for deep learning regularly expose hyperparameters to the user. The user either relies on the default suggestion or sets them 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 four different *tuning budgets*.

[3] All approximations are for ADAM on a Tesla K80 GPU.

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 has great practical value, as it drastically reduces the computational resources required for successful training.

The *small*, *medium* and *large* budgets consist of 25, 50, and 75 tuning runs, where the parameters for each setting are sampled using random search. Tuning runs for the small and medium budget were sampled using the distributions defined in Table 4. The additional 25 tuning runs of the large budget, however, were sampled using refined bounds: For each combination of optimizer, problem, and learning rate schedule we use the same distribution as before, but restrict the search space, to contain all hyperparameter configurations of the top-performing 20 % tuning runs from the medium budget are included.

We use a single seed for tuning, but for all configurations repeat the best setting with ten different seeds. This allows us to report standard deviations in addition to means, assessing stability. Our tuning process can sometimes pick "lucky" seeds, which do not perform well when averaging over multiple runs. This is arguably a feature rather than a bug, since it reflects practical reality. If an optimizer is so unstable that ten random seeds are required for tuning—which would render this benchmark practically infeasible—it would be impractical for the end-user as well. Our scoring naturally prefers stable optimizers. Appendices C and D provide further analysis of these cases and the general stability of our benchmark, showing amongst other things that failing seeds occur in less than 0.5 % of the tuning runs.

Tuning method We tune parameters by random search without early-stopping for the small, medium and large budget. Random search is a popular choice due to its efficiency over grid search (Bergstra & Bengio, 2012) and its ease of implementation and parallelization compared to Bayesian optimization (further discussed in Section 4). A

minor complication of random search is that the sampling distribution affects the performance of the optimizer. The sampling distribution acts as a prior over good parameter settings, and bad priors consequently ruin performance. We followed the valid interval and intuition provided by the optimizers' authors for relevant hyperparameters. The resulting sampling distributions can be found in Table 4 in the appendix. Even though a hyperparameter might have a similar name in different optimization methods (e.g. learning rate α), its appropriate search space can differ. However, without grounded heuristics guiding the practitioner on how the hyperparameters differ between optimizers, the most straightforward approach for any user is to use the same search space. Therefore, in case there was no prior knowledge provided in the cited work we chose similar distributions for similar hyperparameters across different optimizers.

What should be considered a hyperparameter? There is a fuzzy boundary between (tunable) hyperparameters and (fixed) design parameters. A recently contentious example is the ε in adaptive 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 (see Choi et al. (2019) for a discussion). Under this view, one can actually consider several 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 $\text{ADAM}_{\alpha,\beta_1,\beta_2,\varepsilon}$, which can approximate SGD in its hyperparameter space. While both share the update rule, we consider them to be different optimizers. For each update rule, we selected one popular choice of tunable parameters, e.g. $\text{ADAM}_{\alpha,\beta_1,\beta_2}$ (see Table 4).

2.4. Schedules

The literature on learning rate schedules is now nearly as extensive as that on optimizers (see Table 3 in the appendix). *In theory*, schedules can be applied to all hyperparameters of an optimization method 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 (see Appendix E):

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

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 fifteen optimizers, we can construct a 15×15 matrix of performance gains. Figure 2 illustrates this on five problems showing improvements by a positive sign and an orange cell. Detailed plots for all problems are in Figures 10 and 11 in the appendix. For example, the bottom left cell of the largest matrix in Figure 2 shows that AMSBOUND *(1)* tuned using a small budget performs 2.4% better than SGD *(15)* with default parameters on this specific problem.

An **orange 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 badly-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. This is just a statement about the default parameters suggested by the authors or the popular frameworks; well-working default parameters might well exist for those methods. Conversely, a **white & blue row** signals a well-performing default setting, since even tuned optimizers do not significantly outperform it. ADAM, NADAM and RADAM, as well as AMSBOUND, ADABOUND and ADABELIEF all have white or blue rows on several (but not all!) problems, supporting the rule of thumb that adaptive methods have well-working default parameters. Conversely, **orange** (or **blue**) **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 10 to 13 in the appendix suggest an interesting alternative approach for machine learning practitioners: Instead of picking a single optimizer and tuning its hyperparameters extensively, trying out a few optimizers with default settings and picking the best one yields competitive results with less computational and tuning choice efforts. However, this might not hold for more complicated, structurally different tasks such as GANs (Goodfellow et al., 2014) or Transformer models (Vaswani et al., 2017). The similarity of those two approaches might be due to the fact that optimizers have implicit learning rate schedules (Agarwal et al., 2020) and trying out different optimizers is similar to trying out different (well-tested) schedules.

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

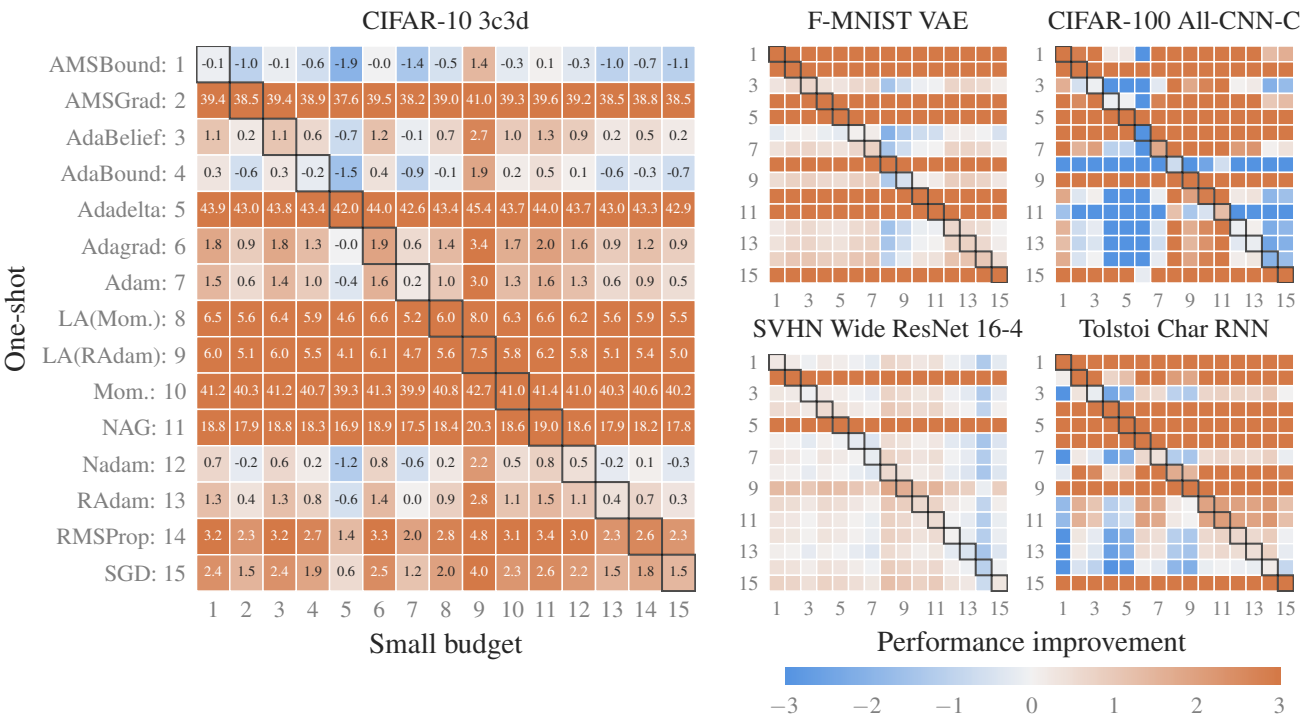

Figure 2: The test set performance improvement after switching from any untuned optimizer (y-axis, *one-shot*) to any tuned optimizer (x-axis, *small budget*) as an average over 10 random seeds for the *constant* schedule. For example, the bottom left cell of the largest matrix indicates that the tuned version of AMSBOUND (1) reaches a $2.4\,\%$ higher test accuracy than untuned SGD (15). We discuss the unintuitive occurrence of negative diagonal entries in Appendix G. The colormap is capped at ± 3 to improve presentation, although larger values occur.

median we observe that increasing the budget improves performance, albeit with diminishing returns. For example, using the medium budget without any schedule leads to a median relative improvement of roughly $3.4\,\%$ compared to the default parameters (without schedule).

Applying an untuned schedule improves median performance as well. For example, the large tuning budget coupled with a trapezoidal learning rate schedule leads to a median relative improvement of the performance of roughly $5.2\,\%$ compared to the default parameters. However, while these trends hold in the median, their individual effect varies wildly among optimizers and problems, as is apparent from the noisy structure of the individual lines shown in Figure 3.

Which optimizers work well after tuning? Figure 4 compares the optimizers' performance across all eight problems. There is no single optimizer that dominates its competitors across all tasks. Nevertheless, some optimizers generally perform well, while others can vary greatly in their behavior, most notably performing poorly on VAEs. 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 performance. Except for the two VAE problems, the best of those two untuned optimizers generally falls within the distribution of the well-tuned methods. Combining these runs with a *tuned* version of ADAM (or a variant thereof) provides stable and slightly improved results across many problems in our benchmark. To further increase the performance, our results suggest trying a different optimizer next, such as RMSPROP or NAG. Across multiple budgets and schedules, both optimizers show a consistently good performance on the RNN and ALL-CNN-C model, respectively.

Nevertheless, achieving (or getting close to) the absolute best performance still requires testing numerous optimizers. Which optimizer wins in the end is problem-dependent: optimizers that achieve top scores on one problem can perform poorly on other tasks. We note in passing that the individual optimizer rankings changes when considering e.g. a smaller budget or an additional learning rate schedule (see Figures 14 to 16 in the appendix). However, the overall trends described here are consistent.

The idea that optimizers perform consistently better or worse for specific model architectures or tasks has been regularly theorized and mentioned in the literature. Indeed, our results support this hypothesis, with NAG often beating ADAM on

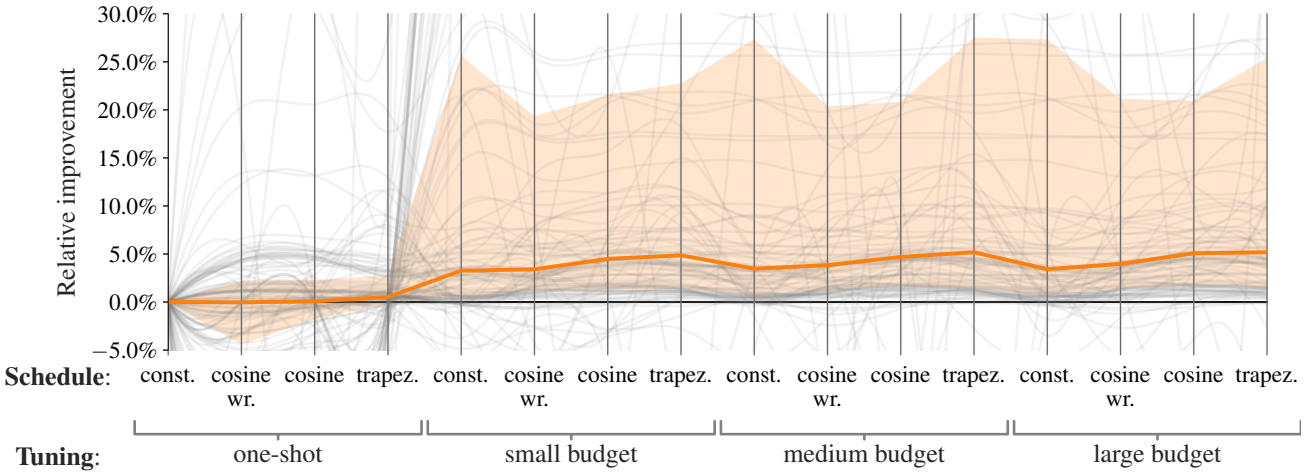

Figure 3: Lines in gray (—, smoothed by cubic splines for visual guidance only) show the relative improvement for a certain tuning budget and schedule (compared to the *one-shot* tuning without schedule) for all fifteen optimizers on all eight problems. The median over all lines is plotted in orange (—) with the shaded area (▮) indicating the area between the 25th and 75th percentile. With an increased budget and a schedule, one can expect a performance increase *on average* (orange lines), but not automatically for individual settings (i.e. gray lines can be unaffected or even decrease).

image classification tasks, and RMSPROP being consistently on top for the natural language modeling task (see Tables 6 to 9). Understanding whether and why certain optimizers favor specific problem types presents an interesting research avenue and might lead to more sophisticated optimizers that utilize the problem characteristics.

4. Limitations

Any empirical benchmark has constraints and limitations. Here we highlight some of ours' and characterize the context within which our results should be considered.

Generalization of the results By using the problems from DEEPOBS, which span models and data sets of varying complexity, size, and different domains, we aim for generalization. Our results are, despite our best efforts, reflective of not just these setups, but also of the chosen training parameters, the software framework, and further unavoidable choices. The design of our comparisons aims to be close to what an informed practitioner would encounter for a relatively novel problem in practice. It goes without saying that even a carefully curated range of problems cannot cover all challenges of machine learning or even just deep learning. In particular, our conclusions may not generalize to other workloads such as GANs, reinforcement learning, or applications where e.g. memory usage is crucial.

Similarly, our benchmark does not cover more large-scale problems such as ImageNet (Deng et al., 2009) or transformer models (Vaswani et al., 2017). While there is oft-mentioned anecdotal evidence that the characteristics of deep learning problems change for larger models, it would simply be impossible to perform the kind of combinatorial exploration of choices covered in our benchmark, even with significant hardware resources. The inclusion of larger models would require reducing the number of tested optimizers, schedules or tuning methods and would thus shift the focus of the benchmark. Studying whether there are systematic differences between different types of deep learning problems presents an interesting avenue for further research.

We do not consider this study the definitive work on benchmarking deep learning optimizers, but rather an important and significant step in the right direction. While our comparison includes many "dimensions" of deep learning optimization, e.g. by considering different problems, tuning budgets, and learning rate schedules, there are certainly many more. To keep the benchmark feasible, we chose to use the fixed L_2 regularization and batch size that DEEPOBS suggests for each problem. We also did not include optimization techniques such as weight averaging or ensemble methods as they can be combined with all evaluated optimizers and hence would increase the computational cost further. Future works could study how these techniques interact with different optimization methods. However, to keep our benchmark feasible, we have selected what we believe to be the most important aspects affecting an optimizer comparison. We hope, that our study lays the groundwork so that other works can build on it and analyze these questions.

Influence of the hyperparameter search strategy As noted by, e.g., Choi et al. (2019) and Sivaprasad et al. (2020), the hyperparameter tuning method, its budget, and its search

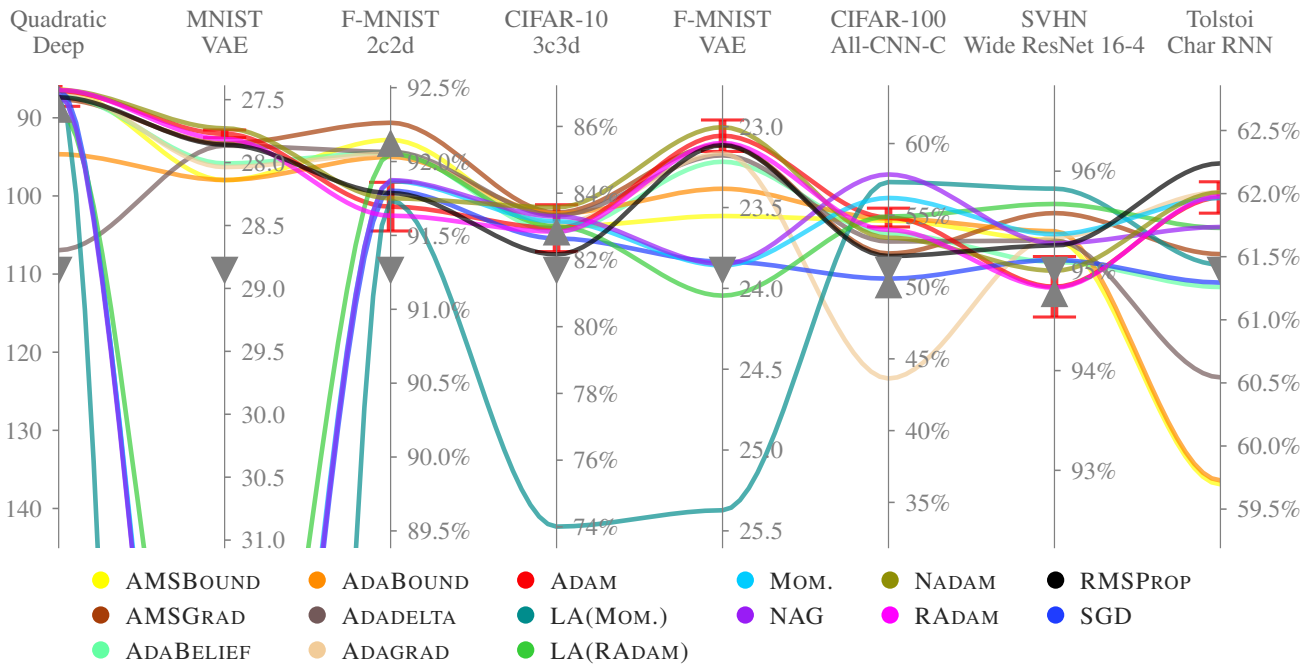

Figure 4: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and *no learning rate schedule*. One standard deviation for the *tuned* ADAM optimizer is shown with a red error bar (I; error bars for other methods omitted for legibility). The performance of *untuned* 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 chosen in relation to the performance of ADAM with default parameters. Tabular version available in the Appendix as Table 6.

domain, can significantly affect performance. By reporting results from four different hyperparameter optimization budgets (including the tuning-free one-shot setting) we try to quantify the effect of tuning. We argue 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, etc.) and thus significantly complicate the analysis.

The individual hyperparameter sampling distributions significantly affect the relative rankings of the optimizers. A poorly chosen search space can make successful tuning next to impossible. In our benchmark, we use relatively broad initial search spaces, dozens of tuning runs and a refining of those search spaces for the large budget. Note, though, that the problem of finding appropriate search spaces is inherited by practitioners. It is arguably an implicit flaw of an optimization method that expects hyperparameter tuning not to come with well-identified search spaces for those parameters and this should thus be reflected in a benchmark.

5. Conclusion

Faced with an avalanche of research developing new stochastic optimization methods, practitioners are left with the near-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 training. Despite efforts by the community, there is currently no method that clearly dominates the competition.

We have provided an extensive empirical benchmark of optimization methods for deep learning. It reveals structure in the crowded field of training methods for deep learning: First, although many methods perform competitively, a subset of methods tends to come up near the top across the spectrum of problems. Despite years of new research by many committed authors, ADAM remains a viable (but also not a clearly superior) choice for many problems, with NAG or RMSPROP being interesting alternatives that were able to boost performance on individual problems. Secondly, tuning helps about as much as trying other optimizers. Our open and extendable data set allows many, more technical observations, for example, that the 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. Different optimizers exhibit a surprisingly similar performance distribution compared to a single method that is re-tuned or simply re-run with different random seeds. It is thus questionable how much insight the development of new methods yields, at least if they are conceptually and functionally close to the existing population. We hope that benchmarks like ours can help the community to move 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.

ACKNOWLEDGMENTS

The authors gratefully acknowledge financial support by the European Research Council through ERC StG Action 757275 / PANAMA; the DFG Cluster of Excellence "Machine Learning - New Perspectives for Science", EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. Moreover, the authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Frank Schneider. We would like to thank Aaron Bahde for providing his analysis on the robustness to random seeds. Further, we are grateful to Lukas Balles, Frederik Künstner, and Felix Dangel for, among other things, helping to create the list of optimizers and providing feedback to the manuscript. Lastly, we want to thank Agustinus Kristiadi, Jonathan Wenger, Marius Hobbhahn, and Lukas Tatzel for their additional feedback.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL http://tensorflow.org/.

Agarwal, N., Anil, R., Hazan, E., Koren, T., and Zhang, C. Disentangling Adaptive Gradient Methods from Learning Rates. *arXiv preprint: 2002.11803*, 2020.

Bergstra, J. and Bengio, Y. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research, JMLR*, 13, 2012.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., and Dahl, G. E. On Empirical Comparisons of Optimizers for Deep Learning. *arXiv preprint: 1910.05446*, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. In *Advances in Neural Information Processing Systems 31, NeurIPS*, 2018.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27, NIPS*, 2014.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint: 1706.02677*, 2017.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and G, W. A. Averaging weights leads to wider optima and better generalization. In *Uncertainty in Artificial Intelligence - Proceedings of the 34th Conference, UAI 2018*, 2018.

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

Metz, L., Maheswaranathan, N., Sun, R., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint: 2002.11887*, 2020.

Schneider, F., Balles, L., and Hennig, P. DeepOBS: A Deep Learning Optimizer Benchmark Suite. In *7th International Conference on Learning Representations, ICLR*, 2019.

Sivaprasad, P. T., Mai, F., Vogels, T., Jaggi, M., and Fleuret, F. Optimizer Benchmarking Needs to Account for Hyperparameter Tuning. In *37th International Conference on Machine Learning, ICML*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention Is All You Need. In *Advances in Neural Information Processing Systems 30, NIPS*, 2017.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *Advances in Neural Information Processing Systems 30, NIPS*, 2017.

Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82, 1997.

Xing, C., Arpit, D., Tsirigotis, C., and Bengio, Y. A Walk with SGD. *arXiv preprint: 1802.08770*, 2018.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. *Dive into Deep Learning*. 2020. https://d2l.ai.