Optimizer Benchmarking Needs to Account for Hyperparameter Tuning

Prabhu Teja S [*1 2] **Florian Mai** [*1 2] **Thijs Vogels** [2] **Martin Jaggi** [2] **François Fleuret** [1 2]

Abstract

The performance of optimizers, particularly in deep learning, depends considerably on their chosen hyperparameter configuration. The efficacy of optimizers is often studied under near-optimal problem-specific hyperparameters, and finding these settings may be prohibitively costly for practitioners. In this work, we argue that a fair assessment of optimizers' performance must take the computational cost of hyperparameter tuning into account, i.e., how easy it is to find good hyperparameter configurations using an automatic hyperparameter search. Evaluating a variety of optimizers on an extensive set of standard datasets and architectures, our results indicate that Adam is the most practical solution, particularly in low-budget scenarios.

1. Introduction

With the ubiquity of deep learning in various applications, a multitude of first-order stochastic optimizers (Robbins & Monro, 1951) have been in vogue. They have varying algorithmic components like momentum (Sutskever et al., 2013) and adaptive learning rates (Tieleman & Hinton, 2012; Duchi et al., 2011; Kingma & Ba, 2015). As the field grows with newer variants being proposed, the standard method to benchmark the performance of these optimizers has been to compare the best possible generalization performance. While it is certainly an important characteristic to be taken into account, we argue that in practice an even more important characteristic is the performance achievable with available resources. A similar view of performance measurement has been recently argued for in the deep learning community owing to the strong debate on sustainable and GreenAI (Strubell et al., 2019; Schwartz et al., 2019).

The performance of optimizers strongly depends on the choice of hyperparameter values such as the learning rate. In the machine learning research community, the sensitivity of models to hyperparameters has been of great debate recently, where in multiple cases, reported model advances did not stand the test of time because they could be explained by better hyperparameter tuning (Lucic et al., 2018; Melis et al., 2018; Henderson et al., 2018; Dodge et al., 2019). This has led to calls for using automatic hyperparameter optimization methods (HPO) with a fixed budget for a fairer comparison of models (Sculley et al., 2018; Hutter et al., 2019b; Eggensperger et al., 2019). This eliminates biases introduced by humans through manual tuning. For industrial applications, automated machine learning (AutoML, Hutter et al., 2019a), which has automatic hyperparameter optimization as one of its key concepts, is becoming increasingly more important. In all these cases, an optimization algorithm that achieves good performances with relatively little tuning effort is arguably substantially more useful than

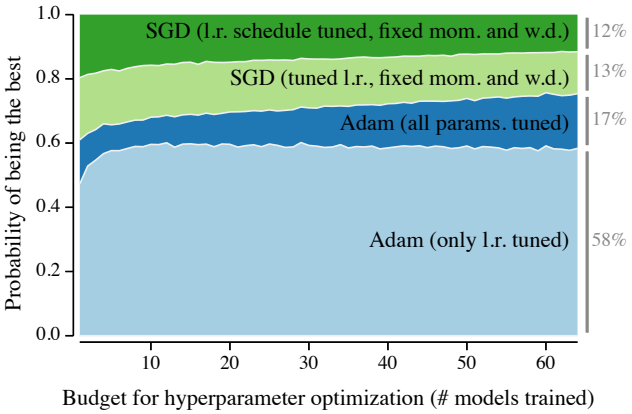


Figure 1: Hyperparameter optimization budget affects the performance of optimizers. We show the probability of finding a hyperparameter configuration for an optimizer that performs the best at a given search budget on any task (sampled from our benchmark). This is encoded by the height of the respective area in the chart. Generally, we see that tuning more hyperparameters becomes more useful with higher budgets. On our 9 diverse tasks that include vision problems, natural language processing, regression and classification, tuning only the learning rate for Adam is the most reliable option, even at large budgets.

[*]Equal contribution [1]Idiap Research Institute, Switzerland [2]EPFL, Switzerland. Correspondence to: Prabhu Teja S <prabhu.teja@idiap.ch>, Florian Mai <fmai@idiap.ch>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

Table 1: Experimental settings shown in the original papers of popular optimizers. The large differences in test problems and tuning methods make them difficult to compare. γ denotes learning rate, μ denotes momentum, λ is the weight decay coefficient.

Method	Datasets	Network architecture	Parameter tuning methods
SGD with momentum (Sutskever et al., 2013)	Artificial datasets MNIST	Fully-connected LSTM	$\mu = 0.9$ for first 1000 updates then $\mu \in \{0, 0.9, 0.98, 0.995\}$. other schedules for μ are used & $log_{10}(\gamma) \in \{-3, -4, -5, -6\}$
Adagrad (Duchi et al., 2011)	ImageNet ranking Reuter RCV1 MNIST KDD Census	Single layer Handcrafted features Histogram features	Perfomance on dev-set
Adam (Kingma & Ba, 2015)	IMDb MNIST CIFAR 10	Logistic regression Multi-layer perceptron Convolutional network	$\beta_1 \in \{0, 0.9\}$ $\beta_2 \in \{0.99, 0.999, 0.9999\}$ $log_{10}(\gamma) \in \{-5, -4, -3, -2, -1\}$
AdamW (Loshchilov & Hutter, 2019)	CIFAR 10 ImageNet 32×32	ResNet CNN	$log_2(\gamma) \in \{-11, -10 \cdots -1, 0\}$ $log_2(\lambda) \in log_2(10^{-3}) + \{-5, -4, \ldots, 4\}$

an optimization algorithm that achieves top performance, but reaches it only with a lot of careful tuning effort. Hence, we advocate that benchmarking the performance obtained by an optimizer must not only avoid manual tuning as much as possible, but also has to account for the cost of tuning its hyperparameters to obtain that performance.

Works that propose optimization techniques show their performance on various tasks as depicted in Table 1. It is apparent that the experimental settings, as well as the network architectures tested, widely vary, hindering a fair comparison. The introduction of benchmarking suites like DEEPOBS (Schneider et al., 2019) have standardized the architectures tested on, however, this does not fix the problem of selecting the hyperparameters fairly. Indeed, recent papers studying optimizer performances may employ grid search to select the best values, but the search spaces are still selected on a per-dataset basis, introducing significant human bias (Schneider et al., 2019; Wilson et al., 2017; Shah et al., 2018; Choi et al., 2019). Moreover, as only the best obtained performance is reported, it is unclear how a lower search budget would impact the results. This leads us to the question: how easy is an optimizer to use, i.e. how quickly can an automatic search method find a set of hyperparameters for that optimizer that result in satisfactory performance?

In this paper, we introduce a simple benchmarking procedure for optimizers that addresses the discussed issues. By evaluating on a wide range of 9 diverse tasks, we contribute to the debate of adaptive vs. non-adaptive optimizers (Wilson et al., 2017; Shah et al., 2018; Chen & Gu, 2018; Choi et al., 2019). To reach a fair comparison, we experiment with several SGD variants that are often used in practice to reach good performance. Although a well-tuned SGD variant is able to reach the top performance in some cases, our overall results clearly favor Adam (Kingma & Ba, 2015), as shown in Figure 1.

2. The Need to Incorporate Hyperparameter Optimization into Benchmarking

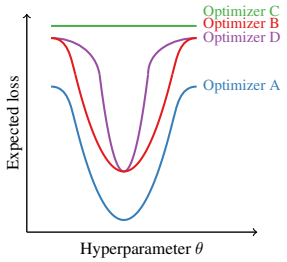
The problem of optimizer benchmarking is two-fold as it needs to take into account

1. how difficult it is to find a good hyperparameter configuration for the optimizer,
2. the absolute performance of the optimizer.

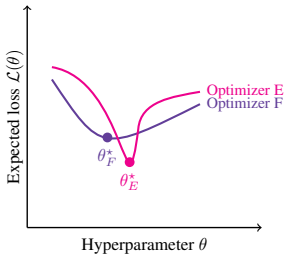
To see why both are needed, consider Figure 2.a, which shows the loss of four different optimizers as a function of their only hyperparameter θ (by assumption). If we only consider requirement #1, optimizer C would be considered the best, since every hyperparameter value is the optimum. However, its absolute performance is poor, making it of low practical value. Moreover, due to the same shape, optimizers A and B would be considered equally good, although optimizer A clearly outperforms B. On the other hand, if we only consider requirement #2, optimizers B and D would be considered equally good, although optimizer D's optimum is harder to find.

As we discuss in Section 3, no existing work on optimizer benchmarking takes both requirements into account. Here we present a formulation that does so in Procedure 1.

We have already established that fairly comparing optimizers needs to account for how easy it is to find good hyperparameter values. When proposing new optimization methods, most often algorithm designers only specify the permissible set of values the hyperparameters can take, and informally provide some intuition of good values. For example, for Adam, Kingma & Ba (2015) bound β_1, β_2 to $[0, 1)$ and specify that they should be close to 1. These are valuable information for users of their algorithm, but they do not allow to formally incorporate that information into a benchmarking procedure. Instead, we argue that we need



2.a: Illustration. It is important to consider both the absolute performance of optimizers as well as the tuning effort to get to good performances.



2.b: Illustration. While optimizer E can achieve the best performance after careful tuning, optimizer F is likely to provide better performance under a constrained HPO budget.

Procedure 1 Benchmark with 'expected quality at budget'

Input: Optimizer O, cross-task hyperparameter prior Θ_O, task T, tuning budget B
Initialization: Pre-compute a *library* of size $\gg B$ with validation losses achieved on task T with optimizer O using hyper-parameters sampled from Θ_O.
Initialize $list \leftarrow [\,]$.
for R repetitions **do**
  Simulate hyperparameter search with budget B:
  – $S \leftarrow$ sample B elements from *library*.
  – $list \leftarrow [\text{BEST}(S),\ \ldots list]$.
end for
return MEAN($list$), or other statistics

to redefine what constitutes an optimizer in such a way that prior knowledge over reasonable hyperparameter values is included.

Definition. *An optimizer is a pair $\mathcal{M} = (\mathcal{U}_\Theta, p_\Theta)$, which applies its update rule $\mathcal{U}(S_t; \Theta)$ at each step t depending on its current state S_t. It is parameterized through N hyper-parameters $\Theta = (\theta_1, \ldots, \theta_N)$ with respective permissible values $\theta_i \in H_i$ $\forall i$, and $p_\Theta : (\Theta \to \mathbb{R})$ defines a probability distribution over the hyperparameters.*

In the example above, we could describe the intuition that β_1, β_2 should be close to 1 by the random variables $\hat{\beta}_1 = 1 - 10^{c_1}, \hat{\beta}_2 = 1 - 10^{c_2}$, where $c_1, c_2 \sim U(-10, -1)$.

Let $\mathcal{L}(\Theta_1)$ refer to the performance (say, test loss) of $\mathcal{M}$ with the specific hyperparameter choice Θ_1.

Let us assume that there are two optimizers E & F, both with a single hyperparameter θ, but no prior knowledge of particularly good values, i.e., the prior is a uniform distribution over the permissible range. Let their loss surface be $\mathcal{L}_E$ and $\mathcal{L}_F$, respectively. As Figure 2.b shows, the minimum of $\mathcal{L}_E$ is lower than that of $\mathcal{L}_F$ (denoted by $\theta_E^\star$ and $\theta_F^\star$) i.e. $\mathcal{L}_E(\theta_E^\star) < \mathcal{L}_F(\theta_F^\star)$. However, the minimum of $\mathcal{L}_E$ is much sharper than that of $\mathcal{L}_F$, and in most regions of the parameter space F performs much better than E. This makes it easier to find configurations that perform well. This makes optimizer-F an attractive option when we have no prior knowledge of the good parameter settings. Previous benchmarking strategies compare only $\theta_E^\star$ and $\theta_F^\star$. It is obvious that in practice, optimizer-F may be an attractive option, as it gives 'good-enough' performance without the need for a larger tuning budget.

We incorporate the relevant characteristics of the hyperparameter optimization surface described above into benchmarking through Procedure 1. In the proposed protocol, we use Random Search (Bergstra & Bengio, 2012) with the optimizers' prior distribution to search the hyperparameter space. The quality of the optimizers can then be assessed by inspecting the maximum performance attained after k trials of random search. However, due to the stochasticity involved in random search, we would usually have to repeat the process many times to obtain a reliable estimate of the distribution of performances after budget k. We instead use the bootstrap method (Tibshirani & Efron, 1993) that resamples from the empirical distribution (termed $library$ in Procedure 1). When we need the mean and variance of the best attained performance after budget k, we use the method proposed by Dodge et al. (2019) to compute them exactly in closed form. We provide the details of the computation in Appendix D.

Our evaluation protocol has distinct advantages over previous benchmarking methods that tried to incorporate automatic hyperparameter optimization methods. First, our evaluation protocol is entirely free of arbitrary human choices that bias benchmarking: The only free parameters of random search itself are the search space priors, which we view as part of the optimizer. Secondly, since we measure and report the performance of Random Search with low budgets, we implicitly characterize the loss surface of the hyperparameters: In terms of Figure 2.b, optimizer-F with its wide minimum will show good performance with low budgets, whereas optimizer-E can be expected to show better performance with high budgets. Such characterizations would not be possible if one only considered the performance after exhausting the full budget. Finally, our evaluation protocol allows practitioners to choose the right optimizer for their budget scenarios.

Discussion of alternative choices In theory, our general methodology could also be applied with a different hyperparameter optimization technique that makes use of prior distributions, e.g., drawing the set of initial observations in Bayesian methods. However, those usually have additional hyperparameters, which can act as potential sources of bias. Moreover, the bootstrap method is not applicable when the hyperparameter trials are drawn dependently, and repeating the hyperparameter optimization many times is practically

infeasible.

In our protocol we consider the number of random search trials as the unit of budget, and not computation time. This is done so as to not violate the independence assumption in the method by Dodge et al. (2019) in Appendix D. We, empirically, show in Appendix F that the conclusions of this paper are still valid when time is used as the unit of budget as well.

3. Related Work

Benchmarking of optimizers is a relatively unstudied subject in literature. Schneider et al. (2019) recently released a benchmark suite for optimizers that evaluates their peak performance and speed, and the performance measure is assessed as the sensitivity of the performance to changes of the learning rate. Our work primarily takes its genesis from the study by Wilson et al. (2017) that finds SGD-based methods as easy to tune as adaptive gradient methods. They perform grid earch on manually chosen grids for various problems and conclude that both SGD and Adam require similar grid search effort. However, their study lacks a clear definition of what it means to be tunable (easy-to-use) and tunes the algorithms on manually selected, dataset dependent grid values. The study by Shah et al. (2018) applies a similar methodology and comes to similar conclusions regarding performance. Since both studies only consider the best parameter configuration, their approaches cannot quantify the efforts expended to find the hyperparameter configuration that gives the best setting; they would be unable to identify the difference between optimizer among B and D in Figure 2.a. In contrast, the methodology in our study is able to distinguish all the cases depicted in Figure 2.a.

There exist few works that have tried to quantify the impact of hyperparameter setting in ML algorithms. For decision tree models, Mantovani et al. (2018) count the number of times the tuned hyperparameter values are (statistically significantly) better than the default values. Probst et al. (2019) define tunability of an ML algorithm as the performance difference between a reference configuration (e.g., the default hyperparameters of the algorithm) and the best possible configuration on each dataset. This metric is comparable across ML algorithms, but it disregards entirely the absolute performance of ML algorithms; thereby being unable to differentiate between optimizers B and D in Figure 2.a.

In a concurrent study, Choi et al. (2019) show that there exists a hierarchy among optimizers such that some can be viewed as specific cases of others and thus, the general optimizer should never under-perform the special case (with appropriate hyperparameter settings). Like in our study, they suggest that the performance comparison of optimizers is strongly predicated on the hyperparameter tuning protocol. However, their focus is on the best possible performance achievable by an optimizer and does not take into account the tuning process. Also, the presence of a hierarchy of optimizers does not indicate how easy it is to arrive at the hyperparameter settings that help improve the performance of the more *general* optimizer. Moreover, while the authors claim their search protocol to be relevant for practitioners, the search spaces are manually chosen *per dataset*, constituting a significant departure from a realistic AutoML scenario considered in our paper. Since, the focus is only on the best attainable performance, it construed as being benchmarking theoretically infinite budget scenarios.

In a recent work, Dodge et al. (2019) propose to use the performance on the validation set along with the test set performance. They note that the performance conclusions reached by previously established NLP models differ widely from the published works when additional hyperparameter tuning budget is considered. They recommend a checklist to report for scientific publications that includes details of compute infrastructure, runtime, and more importantly the hyperparameter settings used to arrive at those results like bounds for each hyperparameter, HPO budget and tuning protocols. They recommend using expected validation performance at a given HPO budget as a metric, along with the test performance.

There has been recent interest in optimizers that are provably robust to hyperparameter choices, termed the APROX family (Asi & Duchi, 2019a;b). Asi & Duchi experimentally find that, training a Residual network (He et al., 2016) on CIFAR-10, SGD converges only for a small range of initial learning rate choices, whereas Adam exhibits better robustness to learning rate choices; their findings are in line with our experiments that it is indeed easier to find good hyperparameter configurations for Adam.

Metz et al. (2020) propose a large range of tasks, and propose to collate hyperparameter configurations over those. They show that the optimizer settings thus collated, that are problem agnostic like us, generalize well to unseen tasks too.

4. Optimizers and Their Hyperparameters

In Section 2, we argued that an optimizer is a combination of update equation, and the probabilistic prior on the search space of the hyperparameter values. Since we are considering a setup akin to AutoML with as little human intervention as possible, these priors have to be independent of the dataset. As we view the hyperparameter priors as a part of the optimizer itself, we argue that they should be prescribed by algorithm designers themselves in the future. However, in the absence of such prescriptions for optimizers like Adam and SGD, we provide a simple method to

estimate suitable priors in Section 4.2.

4.1. Parameters of the Optimizers

To compare the tunability of adaptive gradient methods to non-adaptive methods, we chose the most commonly used optimizers from both the strata; SGD and SGD with momentum for non-adaptive methods, and Adagrad and Adam for adaptive gradient methods. Since adaptive gradient methods are said to work well with their default hyperparameter values already, we additionally employ a default version of Adam where we only tune the initial learning rate and set the other hyperparameters to the values recommended in the original paper (Kingma & Ba, 2015) (termed Adam-LR). Such a scheme has been used by Schneider et al. too. A similar argument can be made for SGD with momentum (termed SGD-M): thus we experiment with a fixed momentum value of 0.9 (termed SGD-M^C), which we found to be the most common momentum value to lead to good performance during the calibration phase.

In addition to standard parameters in all optimizers, we consider weight decay with SGD too. SGD with weight decay can be considered as an optimizer with two steps where the first step is to scale current weights with the decay value, followed by a normal descent step (Loshchilov & Hutter, 2019). Therefore, we conduct two additional experiments for SGD with weight-decay: one where we tune weight-decay along with momentum (termed SGD-MW), and one where we fix it to 10^{-5} (termed SGD-M^{C}W^C) along with the momentum being fixed to 0.9, which again is the value for weight decay we found to be the best during calibration. We incorporate a "Poly" learning rate decay scheduler ($\gamma_t = \gamma_0 \times (1 - \frac{t}{T})^p$) (Liu et al., 2015) for SGD-M^{C}W^C (termed SGD-M^{C}D). This adds only one tunable hyperparameter (exponent p). We also experimented with Adam with learning rate decay scheduler (termed Adam-W^{C}D), but reserve this discussion for the Appendix B, as it did not yield sizeable improvements over Adam-LR or Adam in the problems tested. The full list of optimizers we consider is provided in Table 4, out of which we discuss Adam-LR, Adam, SGD-M^{C}W^C, SGD-MW, and SGD-M^{C}D in the main paper. The rest of them are presented in Appendix B.

Manually defining a specific number of epochs can be biased towards one optimizer, as one optimizer may reach good performance in the early epochs of a single run, another may reach higher peaks more slowly. In order to alleviate this, it would be possible to add the number of training epochs as an additional hyperparameter to be searched. Since this would incur even higher computational cost, we instead use validation set performance as stopping criterion. Thus we stop training when the validation loss plateaus for more than 2 epochs or if the number of epochs exceeds the predetermined maximum number as set in DEEPOBS.

4.2. Calibration of Hyperparameter Prior Distributions

As mentioned previously, we use random search for optimizing the hyperparameters, which requires distributions of random variables to sample from. Choosing poor distributions to sample from impacts the performance, resulting in unfair comparisions, and may break requisite properties (e.g. learning rate is non-negative). For some of the parameters listed in Table 2, obvious bounds exist due their mathematical properties, or have been prescribed by the optimizer designers themselves. For example, Kingma & Ba (2015) bound β_1, β_2 to $[0, 1)$ and specify that they are close to 1. In the absence of such prior knowledge, we devise a simple method to determine the priors.

We use Random Search on a large range of admissible values on each task specified in DEEPOBS to obtain an initial set of results. We then retain the hyperparameters which resulted in performance within 20% of the best result obtained. For each of the hyperparameters in this set, we fit the distributions in the third column of Table 2 using maximum likelihood estimation. Several recent works argue that there exists a complex interplay between the hyperparameters (Smith et al., 2018; Shallue et al., 2019), but we did not find modelling these to be helpful (Appendix H). Instead, we make a simplifying assumption that all the hyperparameters can be sampled independent of each other. We argue that these distributions are appropriate; the only condition on learning rate is non-negativity that is inherent to the log-normal distribution, momentum is non-negative with a usual upper bound of 1, βs in Adam have been prescribed to be less than 1 but close to it, ϵ is used to avoid division by zero and thus is a small positive value close to 0. We did not include p of the learning rate decay schedule in the calibration step due to computational constraints, and chose a fixed plausible range such that the value used by (Liu et al., 2015) is included. We report the parameters of the distributions obtained after the fitting in Table 2. The calibration step is not included in computing the final performance scores, as the calibrated priors are re-usable across tasks and datasets.

5. Experiments and Results

To assess the performance of optimizers for the training of deep neural networks, we benchmark using the open-source suite DEEPOBS (Schneider et al., 2019). The architectures and datasets we experiment with are given in Table 3. We refer the reader to Schneider et al. (2019) for specific details of the architectures. To obtain a better balance between vision and NLP applications, we added an LSTM network with the task of sentiment classification in the IMDB dataset (Maas et al., 2011), details of which are provided in Appendix A.

We aim at answering two main questions with our exper-

Table 2: Optimizers evaluated. For each hyperparameter, we calibrated a 'prior distribution' to give good results across tasks (Section 4.2). $\mathcal{U}[a, b]$ is the continuous uniform distribution on $[a, b]$. Log-uniform(a, b) is a distribution whose logarithm is $\mathcal{U}[a, b]$. Log-normal(μ,σ) is a distribution whose logarithm is $\mathcal{N}(\mu, \sigma^2)$

Optimizer	Tunable parameters	Cross-task prior
SGD	Learning rate	Log-normal(-2.09, 1.312)
	Momentum	$\mathcal{U}[0, 1]$
	Weight decay	Log-uniform(-5, -1)
	Poly decay (p)	$\mathcal{U}[0.5, 5]$
Adagrad	Learning rate	Log-normal(-2.004, 1.20)
Adam	Learning rate	Log-normal(-2.69, 1.42)
	β_1, β_2	1 - Log-uniform(-5, -1)
	ϵ	Log-uniform(-8, 0)

Table 3: Models and datasets used. We use the DeepOBS benchmark set (Schneider et al., 2019). Details are provided in Appendix A.

Architecture	Datasets
Convolutional net	FMNIST, CIFAR10/100
Variational autoencoder	FMNIST, MNIST
Wide residual network	SVHN
Character RNN	Tolstoi's War and Peace
Quadratic function	Artificial datatset
LSTM	IMDB

iments: First, we look at the performance of various optimizers examined. Related to this, we investigate what effect the number of hyperparameters being tuned has on the performance at various budgets (Section 5.1). Second, we consider a problem typically faced in an AutoML scenario: If no knowledge is available a priori of the problem at hand, but only the tuning budget, which optimizer should we use (Section 5.2)?

Table 4: Optimizers and tunable parameters. SGD(γ, μ, λ) is SGD with γ learning rate, μ momentum, λ weight decay coefficient. Adagrad(γ) is Adagrad with γ learning rate, Adam(γ, β_1, β_2, ϵ) is Adam with learning rate γ,

Optimizer label	Tunable parameters
SGD-LR	SGD(γ, μ=0, λ=0)
SGD-M	SGD(γ, μ, λ=0)
SGD-M^C	SGD(γ, μ=0.9, λ=0)
SGD-M^{C}W^C	SGD(γ, μ=0.9, λ=10^{-5})
SGD-M^{C}D	SGD(γ, μ=0.9, λ=10^{-5}) + Poly Decay(p)
SGD-MW	SGD(γ, μ, λ)
Adagrad	Adagrad(γ)
Adam-LR	Adam(γ, β_1=0.9, β_2=0.999, ϵ=10^{-8})
Adam	Adam(γ, β_1, β_2, ϵ)
Adam-W^{C}D	Adam-LR + Poly Decay(p)

5.1. When to Tune More Hyperparameters

To answer the question at which budgets tuning more hyperparameters is preferable, we compare Adam-LR to Adam, and SGD-MW to SGD-M^{C}W^C (Table 4). To this end, we show performance for increasing budgets K in Figure 4. Plots for the other optimizers and budgets are given in Figure 5.

On all classification tasks, Adam-LR and SGD-M^{C}W^C obtain higher performances on average than Adam and SGD-MW, respectively, till the budget of 16. Moreover, the first quartile is often substantially lower for the optimizers with many hyperparameters. For higher budgets, both outperform their counterparts on CIFAR-100 and FMNIST on average and in the second quartile, and Adam outperforms Adam-LR on IMDB as well. However, even for the largest budgets, Adam's first quartile is far lower than Adam-LR's.

On the regression tasks, tuning more hyperparameters only helps for SGD-MW on MNIST-VAE. In all other cases, tuning additional hyperparameters degrades the performance for small budgets, and achieves similar performance at high budgets.

5.2. Summarizing across datasets

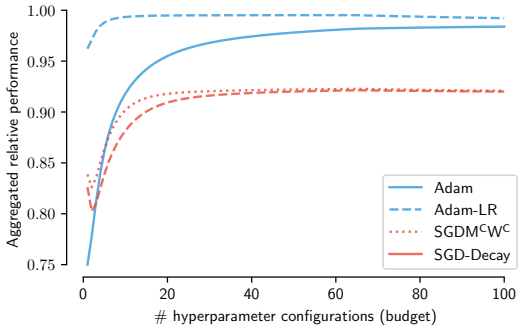


Figure 3: Aggregated relative performance of various optimizers across datasets

We, now, turn to the question of how our choice of optimizer would change in a setting where nothing is known about the problem, à la AutoML. To this end, we summarize performances across all datasets. First, for a given budget, we compute the probability that an optimizer outperforms the others on a randomly chosen task. In Figure 1, we compare Adam, Adam-LR, SGD-M^{C}W^C, and SGD-M^{C}D, because we found them to yield the overall best results. First, the results reflect the findings from Section 5.1 in that tuning more hyperparameters (Adam) becomes a better relative option the more budget is available. However, throughout all tuning budget scenarios, Adam-LR remains by far the most probable to yield the best results.

Figure 1 shows that Adam-LR is the most likely to get

the best results. However, it does not show the margin by which the SGD variants underperform. To address this issue, we compute summary statistics for an optimizer o's performance after k iterations in the following way:

$$S(o,k) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{o(k,p)}{\max_{o' \in \mathcal{O}} o'(k,p)},$$

where $o(k,p)$ denotes the expected performance of optimizer $o \in \mathcal{O}$ on test problem $p \in \mathcal{P}$ after k iterations of hyperparameter search. In other words, we compute the average relative performance of an optimizer to the best performance of any optimizer on the respective task, at budget k.

The results are in Figure 3 which show that Adam-LR performs very close to the best optimizer for all budgets. In early stages of HPO, the SGD variants perform 20% worse than Adam-LR. This gap narrows to 10% as tuning budgets increase, but the flatness of the curves for high budgets suggest that they are unlikely to improve further with higher budgets. Adam on the other hand steadily improves relative to Adam-LR, and only leaves a 2-3% gap at high budgets.

6. Discussion

The key results of our experiments are two-fold. First, they support the hypothesis that adaptive gradient methods are easier to tune than non-adaptive methods: In a setting with low budget for hyperparameter tuning, tuning only Adam's learning rate is likely to be a very good choice; it doesn't guarantee the best possible performance, but it is evidently the easiest to find well-performing hyperparameter configurations for. While SGD (variants) yields the best performance in some cases, its best configuration is tedious to find, and Adam often performs very close to it. Hence, in terms of Figure 2.b, SGD seems to be a hyperparameter surface with narrow minima, akin to optimizer E, whereas the minima of Adam are relatively wide, akin to optimizer F. We investigate the empirical hyperparameter surfaces in Appendix G to confirm our hypothesis. We, thus, state that the substantial value of the adaptive gradient methods, specifically Adam, is its amenability to hyperparameter search. This is in contrast to the findings of Wilson et al. (2017) who observe no advantage in tunabilty for adaptive gradient methods, and thus deem them to be of 'marginal value'. This discrepancy is explained by the fact that our evaluation protocol is almost entirely free of possible human bias: In contrast to them, we do not only avoid manually tuning the hyperparameters through the use of automatic hyperparameter optimization, we also automatically determine the HPO's own hyperparameters by estimating the distributions over search spaces.

Secondly, we find that tuning optimizers' hyperparameters apart from the learning rate becomes more useful as the available tuning budget goes up. In particular, we find that Adam approaches the performance of Adam-LR for large budgets. This is, of course, an expected result, and in line with recent work by Choi et al. (2019), who argue that, with sufficient hyperparameter tuning, a more general optimizer (Adam) should never under-perform any particular instantiation thereof (Adam-LR). Choi et al. (2019) claim that this point is already reached in 'realistic' experiments. However, in their experiments, Choi et al. (2019) tune the search spaces for each problem they consider, thereby assuming apriori knowledge of what constitutes meaningful hyperparameter settings for that specific problem. Our results, which are obtained with a protocol that is arguably less driven by human bias, tell a different story: Even with relatively large tuning budget, tuning only the learning rate of Adam is arguably the safer choice, as it achieves good results with high probability, whereas tuning all hyperparameters can also result in a better performance albeit with high variance. These observations suggest that optimizers with many tunable hyperparameters have a hyperparameter surface that is less smooth, and that is the reason why fixing e.g. the momentum and weight decay parameters to prescribed 'recipe' values is beneficial in low-resource scenarios.

Our study is certainly not exhaustive: We do not study the effect of a different HPO like a Bayesian HPO on the results, due to prohibitively high computational cost it incurs. By choosing uni-variate distribution families for the hyperparameters to estimate the priors, we do not account for complex relationships between parameters that might exist. We explore this in Appendix H where we use the notion of 'effective learning rate' (Shallue et al., 2019), and we find that it helps improve the performance in the lower budgets of hyperparameter optimization. We attribute this to the fact that SGDElrW is effective at exploiting historically successful (γ, μ) pairs. However, the literature does not provide methods to incorporate these into a probabilistic model that incorporates the causal relationships between them.

In the future, we suggest that optimizer designers not only study the efficacy and convergence properties, but also provide priors to sample hyperparameters from. Our study demonstrates this to be a key component in determining an optimizer's practical value.

7. Conclusion

We propose to include the process of hyperparameter optimization in optimizer benchmarking. In addition to showing peak performance, this showcases the optimizer's ease-of-use in practical scenarios. We hope that this paper encourages other researchers to conduct future studies on the performance of optimizers from a more holistic perspective, where the cost of the hyperparameter search is included.

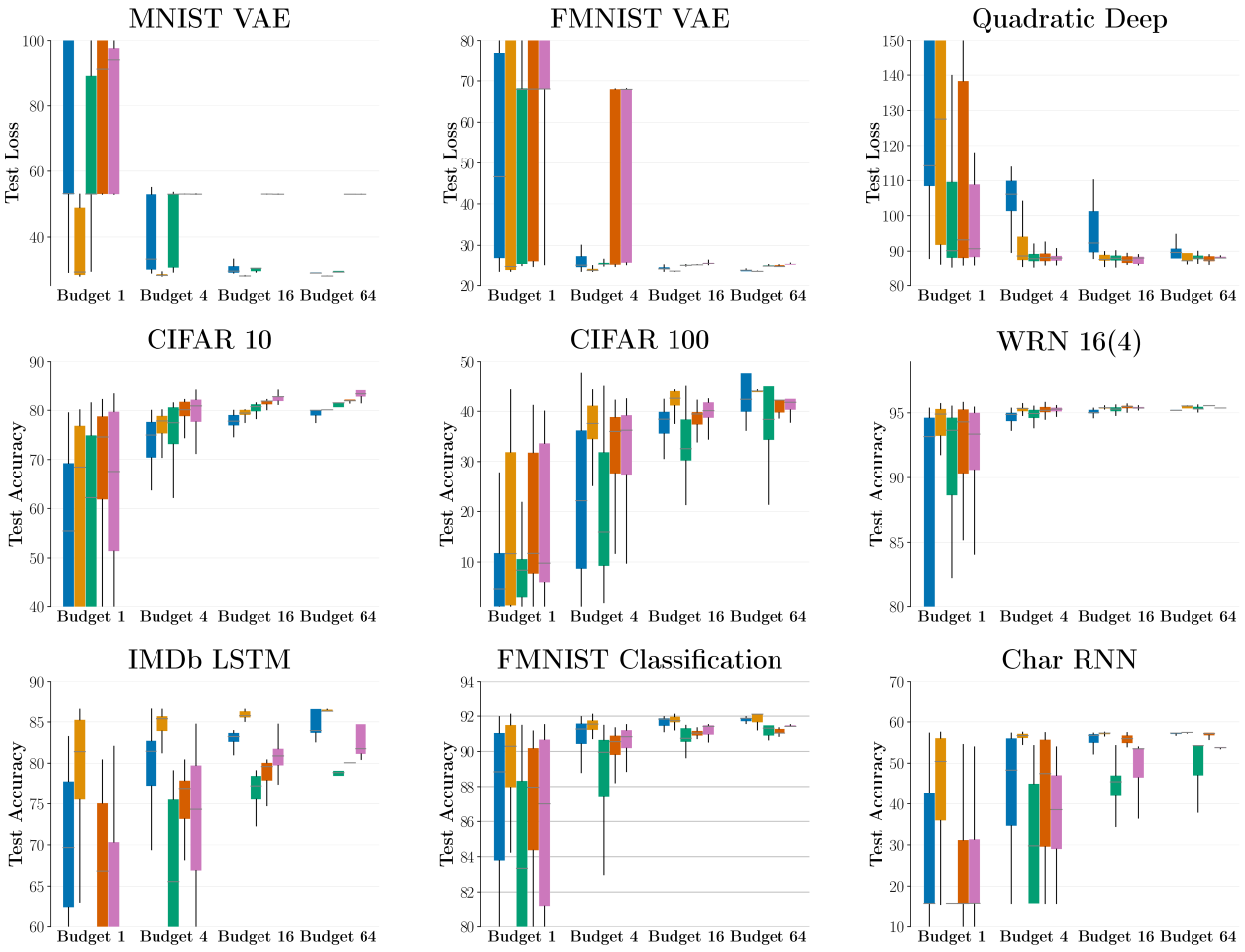


Figure 4: Performance of **Adam-LR**, **Adam**, **SGD-M^{C}W^C**, **SGD-MW**, **SGD-M^{C}D** at various hyperparameter search budgets. Image is best viewed in color. Some of the plots have been truncated to increase readability.

ACKNOWLEDGMENTS

Prabhu Teja was supported by the "Swiss Center for Drones and Robotics - SCDR" of the Department of Defence, Civil Protection and Sport via armasuisse S+T under project n°050-38. Florian Mai was supported by the Swiss National Science Foundation under the project Learning Representations of Abstraction for Opinion Summarisation (LAOS), grant number "FNS-30216". The authors thank Frank Schneider and Aaron Bahde for giving us access to the pre-release PyTorch version of DEEPOBS.

References

Asi, H. and Duchi, J. C. The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*, 2019a. ISSN 0027-8424. doi: 10.1073/pnas.1908018116. URL https://www.pnas.org/content/early/2019/10/29/1908018116.

Asi, H. and Duchi, J. C. Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*, 29(3):2257–2290, 2019b.

Bergstra, J. and Bengio, Y. Random search for hyperparameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Chen, J. and Gu, Q. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *CoRR*, abs/1806.06763, 2018. URL http://arxiv.org/abs/1806.06763.

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 arXiv:1910.05446*, 2019.

Dodge, J., Gururangan, S., Card, D., Schwartz, R., and Smith, N. A. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2185–2194, 2019.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Eggensperger, K., Lindauer, M., and Hutter, F. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Hutter, F., Kotthoff, L., and Vanschoren, J. Automated machine learning-methods, systems, challenges, 2019a.

Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Hyperparameter Optimization*, pp. 3–33. Springer International Publishing, Cham, 2019b. ISBN 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5_1. URL https://doi.org/10.1007/978-3-030-05318-5_1.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Liu, W., Rabinovich, A., and Berg, A. C. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015. URL http://arxiv.org/abs/1506.04579.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pp. 700–709, 2018.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Mantovani, R. G., Horváth, T., Cerri, R., Junior, S. B., Vanschoren, J., de Carvalho, A. C. P. d., and Ferreira, L. An empirical study on hyperparameter tuning of decision trees. *arXiv preprint arXiv:1812.02207*, 2018.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ByJHuTgA-.

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 arXiv:2002.11887*, 2020.

Probst, P., Boulesteix, A.-L., and Bischl, B. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019. URL http://jmlr.org/papers/v20/18-444.html.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Schneider, F., Balles, L., and Hennig, P. DeepOBS: A deep learning optimizer benchmark suite. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJg6ssC5Y7.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green AI. *CoRR*, abs/1907.10597, 2019. URL http://arxiv.org/abs/1907.10597.

Sculley, D., Snoek, J., Wiltschko, A. B., and Rahimi, A. Winner's curse? on pace, progress, and empirical rigor. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018. URL https://openreview.net/forum?id=rJWF0Fywf.

Shah, V., Kyrillidis, A., and Sanghavi, S. Minimum norm solutions do not always generalize well for over-parameterized problems. *arXiv preprint arXiv:1811.07055*, 2018.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. URL http://jmlr.org/papers/v20/18-789.html.

Smith, S. L., Kindermans, P.-J., and Le, Q. V. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1Yy1BxCZ.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. URL https://www.aclweb.org/anthology/P19-1355.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In Dasgupta, S. and McAllester, D. (eds.), *Proceedings of the 30th International Conference on Machine Learning*, number 3 in Proceedings of Machine Learning Research, pp. 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/sutskever13.html.

Tibshirani, R. J. and Efron, B. An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57:1–436, 1993.

Tieleman, T. and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

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*, pp. 4148–4158, 2017.

Appendix for:

Optimizer Benchmarking Needs to Account for Hyperparameter Tuning

Prabhu Teja S * **Florian Mai** * **Thijs Vogels** **Martin Jaggi** **François Fleuret**

A. Architectures of the Models Used in Experiments

Along with the architectures examined by (Schneider et al., 2019), we experiment with an additional network and dataset. We included an additional network into our experimental setup, as DEEPOBS does not contain a word level LSTM model. Our model uses a 32-dimensional word embedding table and a single layer LSTM with memory cell size 128, the exact architecture is given in Table 5. We experiment with the IMDB sentiment classification dataset (Maas et al., 2011). The dataset contains $50,000$ movie reviews collected from movie rating website IMDB. The training set has $25,000$ reviews, each labeled as positive or negative. The rest $25,000$ form the test set. We split 20% of the training set to use as the development set. We refer the readers to DEEPOBS (Schneider et al., 2019) for the exact details of the other architectures used in this work.

Table 5: Architecture of the LSTM network used for IMDb experiments

Layer name	Description
Emb	[Embedding Layer Vocabulary of 10000 Embedding dimension: 32]
LSTM_1	[LSTM Input size: 32 Hidden dimension: 128]
FC Layer	Linear$(128 \rightarrow 2)$
Classifier	Softmax(2)

B. Performance Analysis

We show the full performance plots of all variants of SGD, Adam, and Adagrad we experimented with in Figure 5.

C. How Likely Are We to Find Good Configurations?

In Figure 1 we showed the chance of finding the optimal hyperparameter setting for some of the optimizers considered, in a problem agnostic setting. Here we delve into the case where we present similar plots for each of the problems considered in Section 5.

A natural question that arises is: Given a budget K, what is the best optimizer one can pick? In other words, for a given budget what is the probability of each optimizer finding the best configuration? We answer this with a simple procedure. We repeat the runs of HPO for a budget K, and collect the optimizer that gave the best result in each of those runs. Using the classical definition of probability, we compute the required quantity. We plot the computed probability in Figure 6. It is very evident for nearly all budgets, Adam-LR is always the best option for 4 of the problems. SGD variants emerge to be better options for CIFAR-100 and Char-RNN at later stages of HPO. For some of the problems like VAEs, LSTM, it is

very obvious that Adam-LR is nearly always the best choice. This further strengthens our hypothesis that adaptive gradient methods are more tunable, especially in constrained HPO budget scenarios.

D. Computing the Expected Maximum of Random Samples

The following is a constructive proof of how to compute the expected value that the bootstrap method converges to in the limit of infinite re-sampling. It is a paraphrase of Dodge et al. (2019, Section 3.1), but due to inaccuracies in Equation (1) in their paper, we repeat it here for clarity.

Let $x_1, x_2 \dots x_N \sim \mathcal{X}$ be N independently sampled values. Let the random variable $\mathbf{Y}$ be the maximum of a random subset of size S from $x_1, x_2 \dots x_N$ where $S \leq N$. For representational convenience, let them be the first S samples. So, $\mathbf{Y} = \max\{x_1, \dots, x_S\}$. We are interested in computing $\mathbb{E}[\mathbf{Y}]$. This can be computed as

$$\mathbb{E}[\mathbf{Y}] = \sum_y y \cdot P(\mathbf{Y} = y)$$

for discrete $\mathbf{Y}$, with $P(\mathbf{Y} = y)$ be the probability mass function of $\mathbf{Y}$. We can write

$$P(\mathbf{Y} = y) = P(\mathbf{Y} \leq y) - P(\mathbf{Y} < y)$$

As $x_i \ \forall i$ are iid sampled,

$$\begin{aligned} P(\mathbf{Y} \leq y) &= P(\max_{i=1\dots S} x_i \leq y) \\ &= \prod_{i=1}^{S} P(x_i \leq y) \\ &= P(X \leq y)^S \end{aligned}$$

$P(X \leq y)$ can be empirically estimated from data as the sum of normalized impulses.

$$P(X \leq y) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{x_i \leq y} \tag{1}$$

Thus,

$$\mathbb{E}[\mathbf{Y}] = \sum_y y(P(X \leq y)^S - P(X < y)^S) \tag{2}$$

A very similar equation can be derived to compute the variance too. Variance is defined as $Var(\mathbf{Y}) = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. The second operand is given by Equation (2). The first operand (for discrete distributions) can be computed as

$$\mathbb{E}[\mathbf{Y}^2] = \sum_y y^2(P(X \leq y)^S - P(X < y)^S) \tag{3}$$

Given the iterates (not incumbents) of Random Search, the expected performance at a given budget can be estimated by Equation (2) and the variance can be computed by Equation (3).

E. Aggregating the Performance of Incumbents

In Procedure 1, we propose returning all the incumbents of the HPO algorithm. Here we propose the use of an aggregation function that helps create a comparable scalar that can used in a benchmarking software like DEEPOBS to rank the performance of optimizers that takes into cognizance the ease-of-use aspect too.

E.1. Aggregating Function

For the aggregation function discussed, we propose a simple convex combination of the incumbent performances and term it ω-tunability. If $\mathcal{L}_t$ be the incumbent performance at budget t, we define ω-tunability as

$$\omega\text{-tunability} = \sum_{t=1}^{T} \omega_t \mathcal{L}_t$$

where $w_t > 0 \; \forall$ t and $\sum_t w_t = 1$

By appropriately choosing the weights $\{\omega_t\}$, we can interpolate between our two notions of tunability in Section 2. In the extreme case where we are only interested in the peak performance of the optimizer, we can set $\omega_T = 1$ and set the other weights to zero. In the opposite extreme case where we are interested in the "one-shot tunability" of the optimizer, we can set $\omega_1 = 1$. In general, we can answer the question of "How well does the optimizer perform with a budget of K runs?" by setting $\omega_i = \mathbf{1}_{\mathrm{i=K}}$. Figure 4 and Figure 5 can also be computed as $\omega_i = \mathbf{1}_{\mathrm{i=K}}$.

While the above weighting scheme is intuitive, merely computing the performance after expending HPO budget of K does not consider the performance obtained after the previous $K-1$ iterations i.e. we would like to differentiate the cases where a requisite performance is attained by tuning an optimizer for K iterations and another for K_1 iterations, where $K_1 \gg K$. Therefore, we employ a weighting scheme as follows: By setting $\omega_i \propto (T-i)$, our first one puts more emphasis on the earlier stages of the hyperparameter tuning process. We term this weighting scheme *Cumulative Performance-Early*(*CPE*). The results of the various optimzers' *CPE* is shown in Table 6. It is very evident that Adam-LR fares the best across tasks. Even when it is not the best performing one, it is quite competitive. Thus, our observation that Adam-LR is the easiest-to-tune i.e. it doesn't guarantee the best performance, but it gives very competitive performances early in the HPO search phase, holds true.

For a benchmarking software package like DEEPOBS, we suggest the use of *CPE* to rank optimziers, as it places focus on ease-of-tuning. This supplements the existing peak performance metric reported previously.

Optimizer	FMNIST(%)↑	CIFAR 10(%)↑	CIFAR 100(%)↑	IMDb(%)↑	WRN 16(4)(%)↑	Char-RNN(%)↑	MNIST-VAE↓	FMNIST-VAE↓	Quadratic Deep↓
Adam-LR	**91.6**	78.8	**42.0**	85.9	**95.3**	56.9	28.9	**24.3**	89.9
Adam	91.3	77.3	38.1	83.4	94.5	54.2	33.1	25.7	95.4
SGD-M^{C}W^C	90.8	81.0	38.8	78.7	**95.3**	53.9	54.0	27.9	**87.4**
SGD-MW	90.5	79.6	33.2	75.2	95.0	44.4	35.2	26.5	87.5
SGD-M^{C}D	91.1	**82.1**	39.2	80.5	95.2	49.6	54.3	29.8	87.5
Adagrad	91.3	76.6	29.8	84.4	95.0	55.6	30.7	25.9	90.6
Adam-W^{C}D	**91.6**	79.4	35.1	**86.0**	95.1	**57.4**	**28.6**	**24.3**	92.8
SGD-LR	90.4	76.9	30.6	68.1	94.7	39.9	53.4	26.2	89.3
SGD-M	90.5	77.8	39.8	73.8	94.9	50.7	37.1	26.3	88.2
SGD-M^C	90.7	78.8	**42.0**	79.0	95.0	55.5	54.1	28.5	88.1

Table 6: *CPE* for the various optimizers experimented. It is evident that Adam-LR is the most competitive across tasks.

F. Results for Computation Time Budgets

Using number of hyperparameter configuration trials as budget unit does not account for the possibility that optimizers may require different amounts of computation time to finish a trial. While the cost for one update step is approximately the same for all optimizers, some require more update steps than others before reaching convergence.

To verify that our results and conclusions are not affected by our choice of budget unit, we simulate the results we would have obtained with a computation time budget in the following way. For a given test problem (e.g., CIFAR-10), we compute the minimum number of update steps any optimizer has required to finish 100 trials, and consider this number to be the maximum computation budget. We split this budget into 100 intervals of equal size. Using the bootstrap (Tibshirani & Efron, 1993), we then simulate 1,000 HPO runs, and save the best performance achieved at each interval. Note that sometimes an optimizer can complete multiple hyperparameter trials in one interval, and sometimes a single trial may take longer than one interval. Finally, we average the results from all 1,000 HPO runs and compute the same summary across datasets as in Section 5.2.

Figure 7 shows that the conclusions do not change when using computation time as budget unit. In fact, the graphs show almost the exact same pattern as in Figure 3, where number of hyperparameter trials is the budget unit.

G. Plotting Hyperparameter Surfaces

In Section 2, we hypothesize that the performance as a function of the hyperparameter, e.g., learning rate, of an optimizer that performs well with few trials has a wider extremum compared to an optimizer that only performs well with more trials.

In Figure 8, we show a scatter plot of the loss/accuracy surfaces of SGD-M^{C}W^C and Adam-LR as a function of the learning rate, which is their only tunable hyperparameter. The plots confirm the expected behavior. On MNIST VAE, FMNIST VAE, and Tolstoi-Char-RNN, Adam-LR reaches performances close to the optimum on a wider range of learning rates than SGD-M^{C}W^C does, resulting in substantially better expected performances at small budgets ($k = 1, 4$) as opposed to SGD-M^{C}W^C, even though their extrema are relatively close to each other. On CIFAR10, the width of the maximum is similar, leading to comparable performances at low budgets. However, the maximum for SGD-M^{C}W^C is slightly higher, leading to better performance than Adam-LR at high budgets.

H. Interplay between momentum and learning rate

We ran an additional experiment using 'effective learning rate' (Shallue et al., 2019) that combines learning rate γ, and momentum μ of SGD to compute the effective learning rate γ^{eff}. Intuitively, γ^{eff}quantifies the contribution of a given minibatch to the overall training. This is defined as

$$\gamma^{\text{eff}} = \frac{\gamma}{1-\mu}$$

We designed a variant of SGD-MW, called SGD-LR$_{\text{eff}}$, where we sampled γ and γ^{eff}independently from lognormal priors calibrated as usual, and compute the momentum(μ) as $\mu = \max(0, (1 - \frac{\gamma}{\gamma^{\text{eff}}}))$, hence accounting for interplay between learning rate and momentum. We plot the performance comparisons between SGD-MW and SGD-LR$_{\text{eff}}$ in Figure 9, and provide a plot of the aggregated relative performance in Figure 10. The results show that indeed SGD-LR$_{\text{eff}}$ improves over SGD-MW in the low-budget regime, particularly on classification tasks. We attribute this to the fact that SGD-LR$_{\text{eff}}$ is effective at exploiting historically successful (γ, μ) pairs. For large budgets, however, SGD-LR$_{\text{eff}}$ performs increasingly worse than SGD-MW, which can be explained by the fact that SGD-MW has a higher chance of exploring new configurations due to the independence assumption. Despite the improvement in low-budget regimes, SGD variants, including the new SGD-LR$_{\text{eff}}$ variant, remain substantially below Adam-LR in all budget scenarios. Hence, our conclusion remains the same.

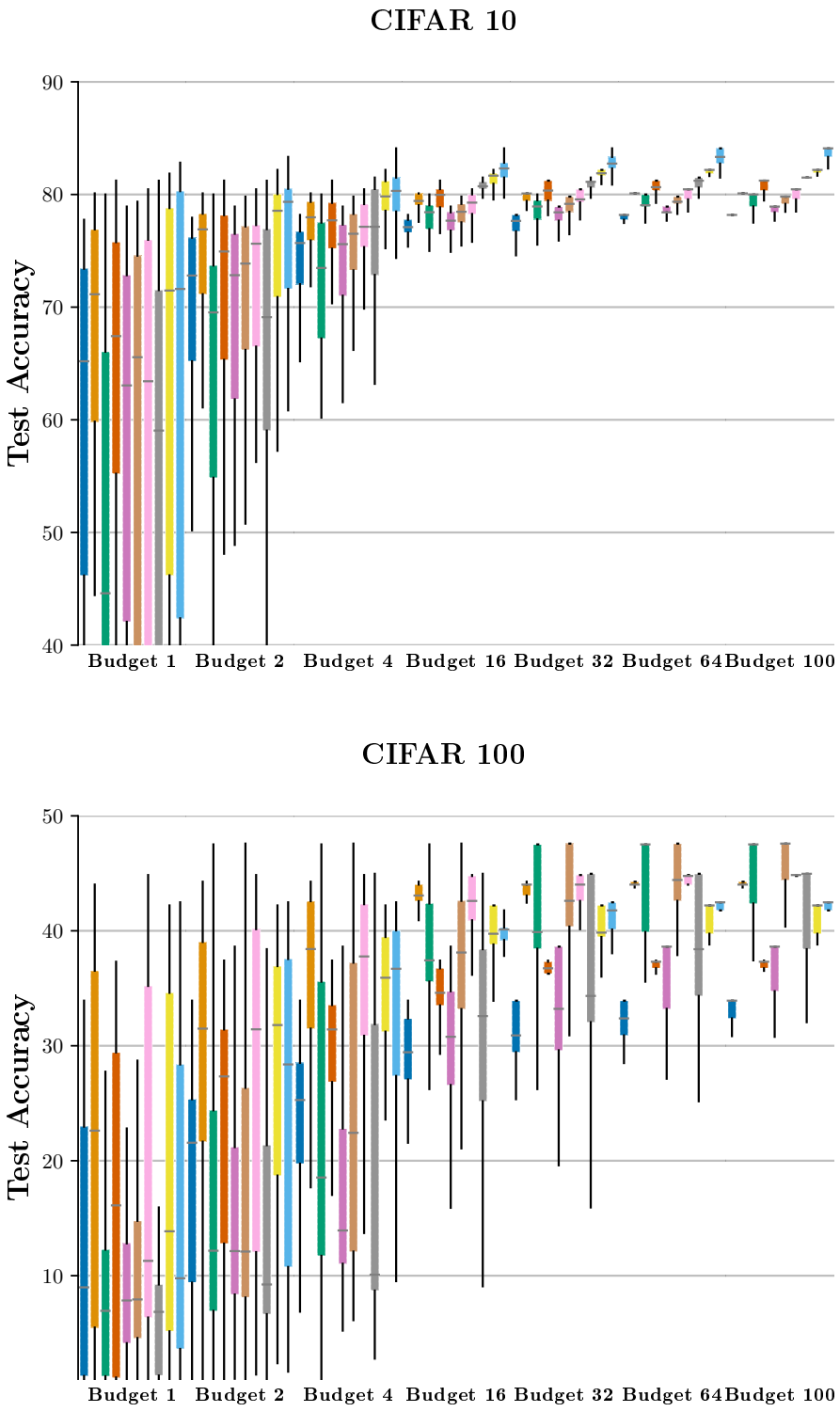


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

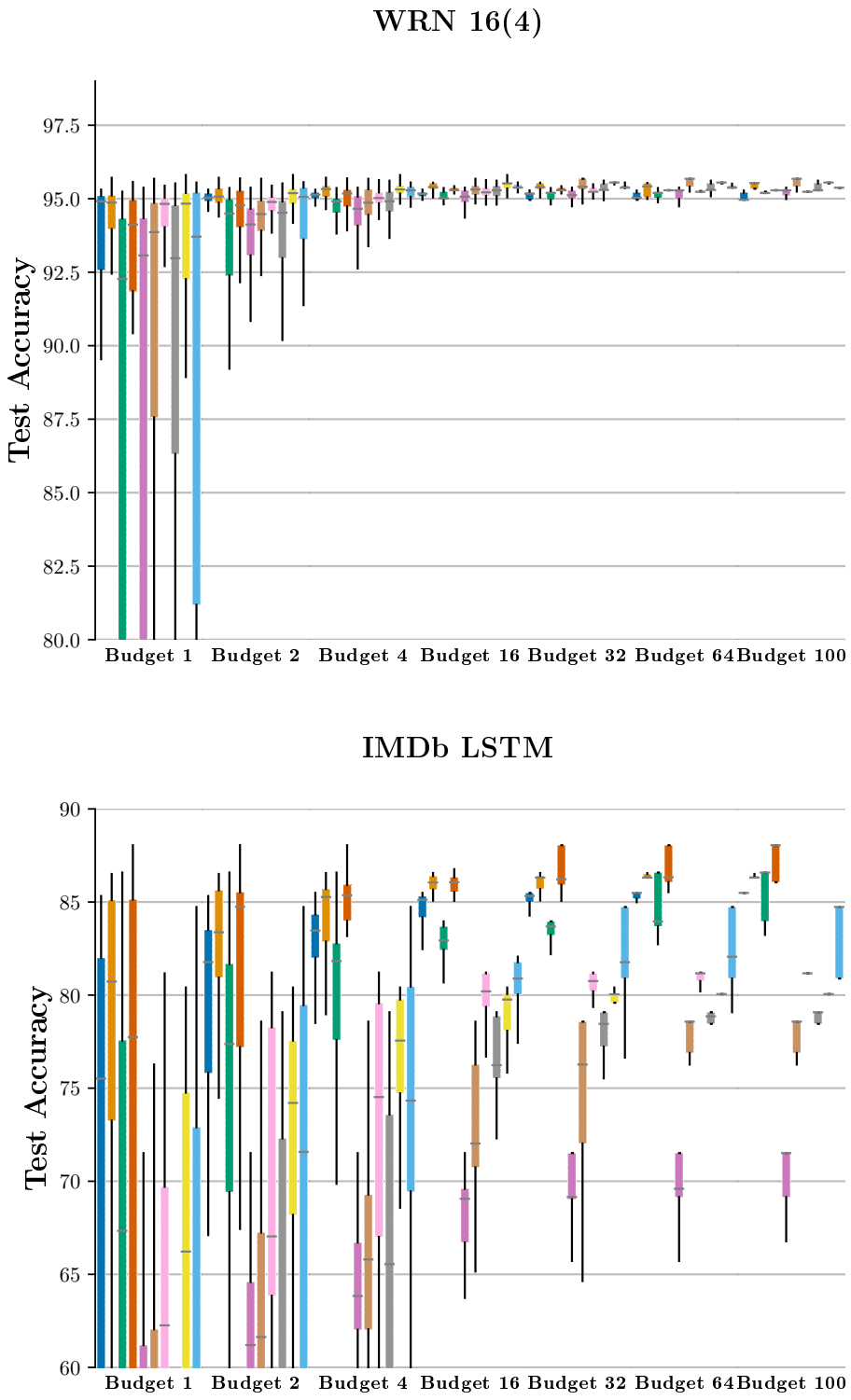


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

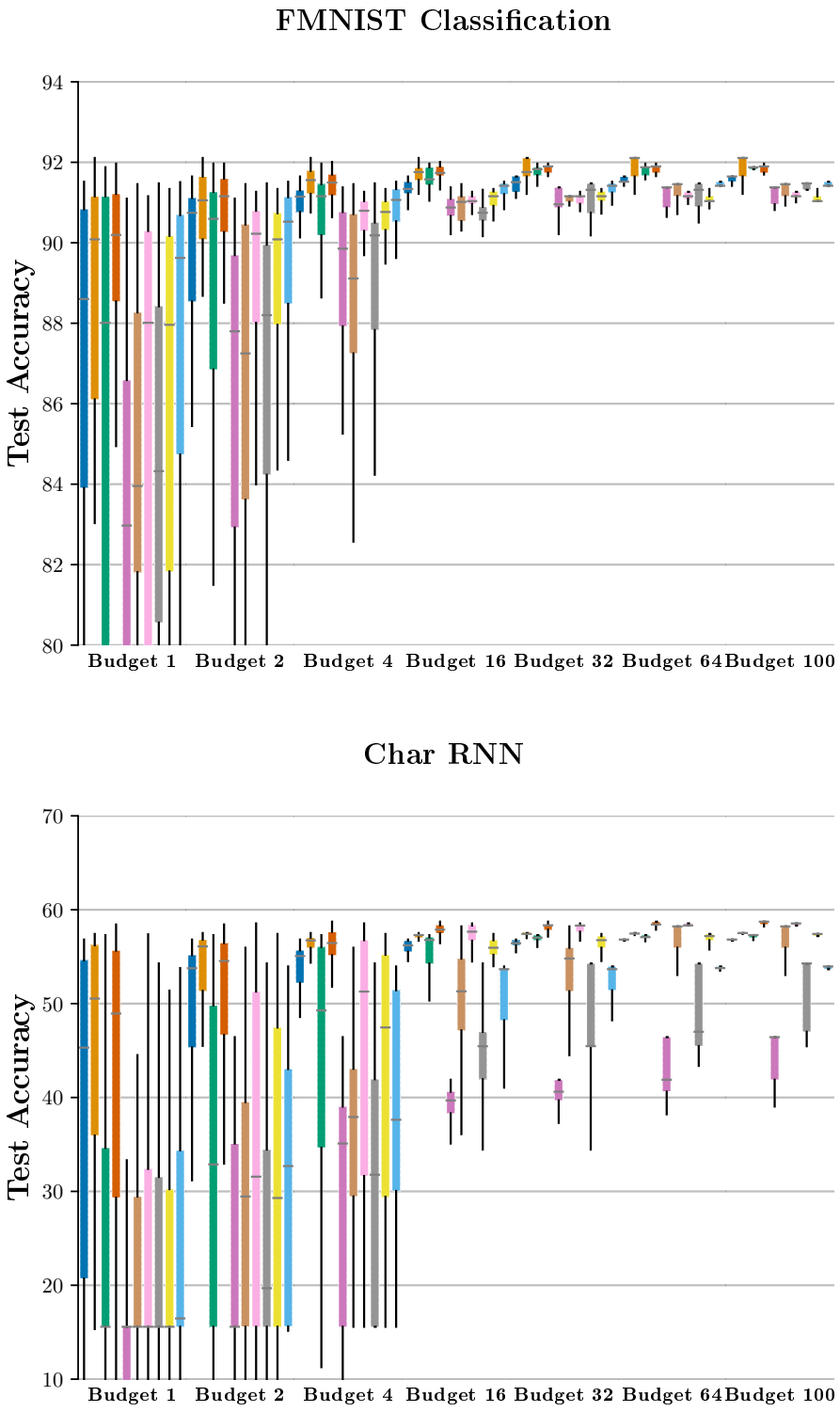


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

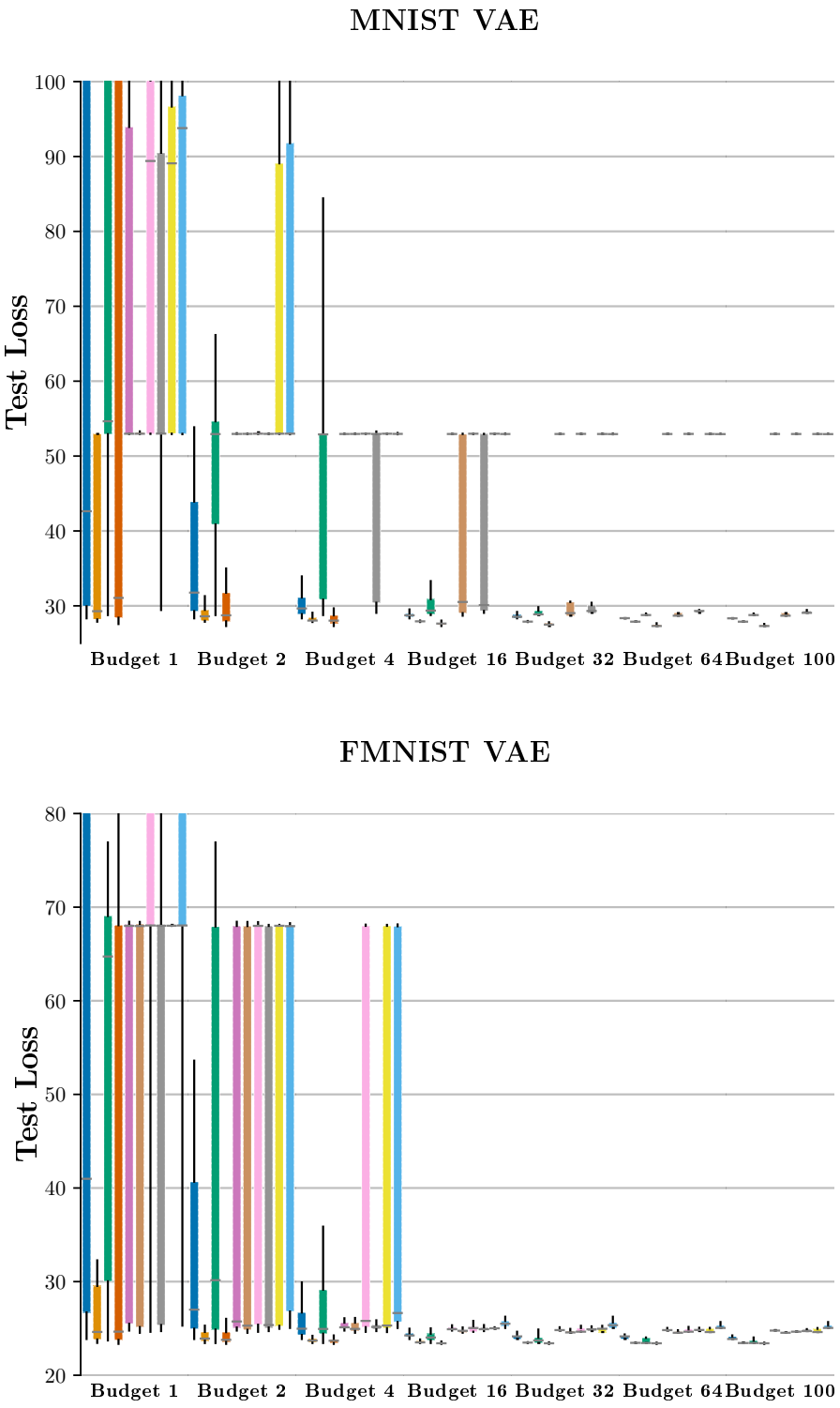


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

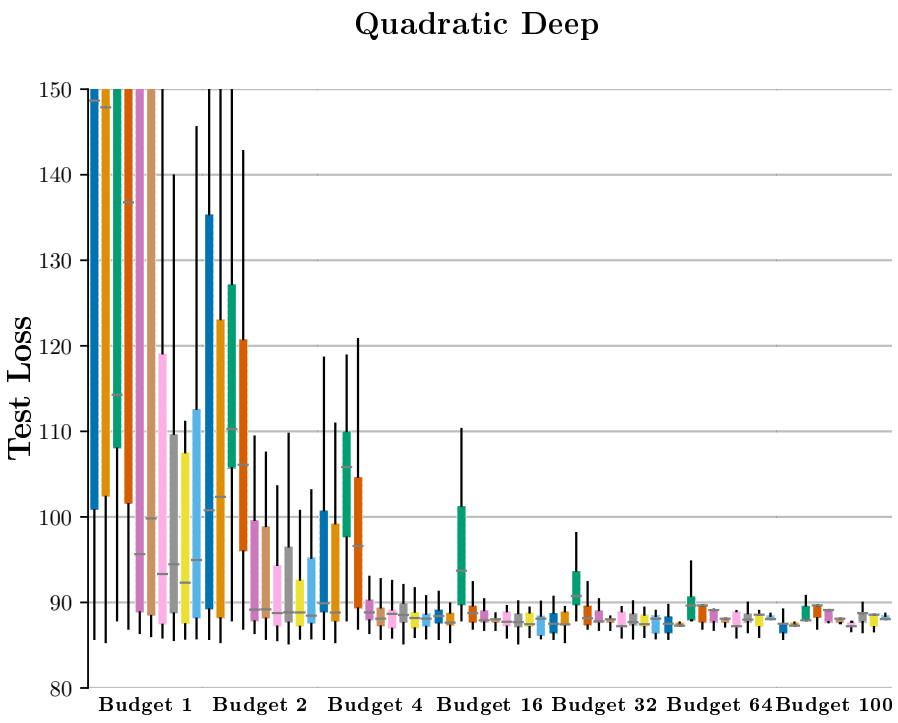


Figure 5: We show the performance of **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D** over all the experiments. We plot the on the x-axis the number of the hyperparameter configuration searches, on the y-axis the appropriate performance.

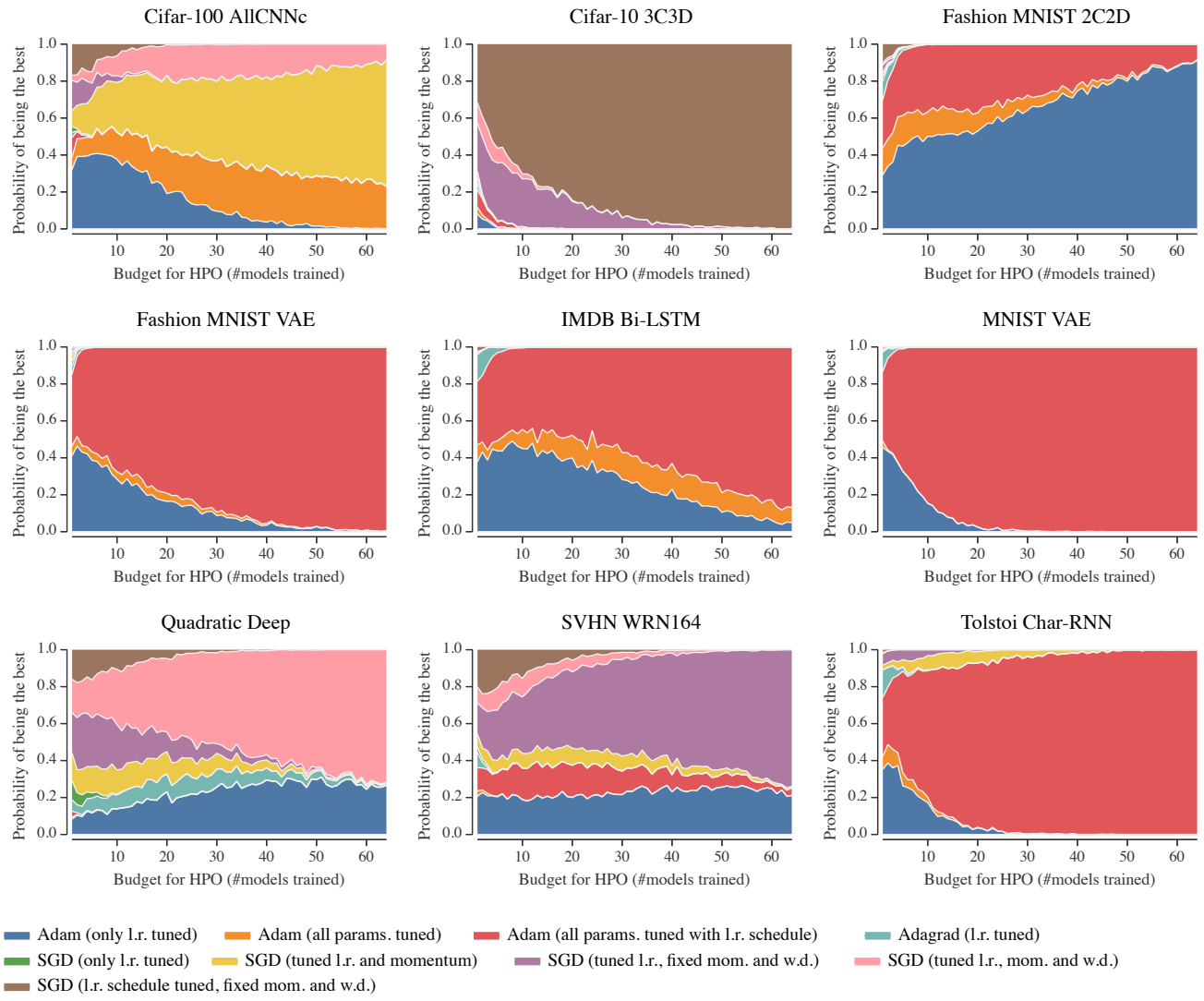


Figure 6: Which optimizer for which budget? Given a tuning budget K (x-axis), the stacked area plots above show how likely each optimizer (colored bands) is to yield the best result after K steps of hyperparameter optimization. For example, for the IMDB LSTM problem, for a small budget, Adam-LR is the best choice (with ~ 0.8 probability), whereas for a larger search budget > 50, tuning the other parameters of 'Adam' is likely to pay off.

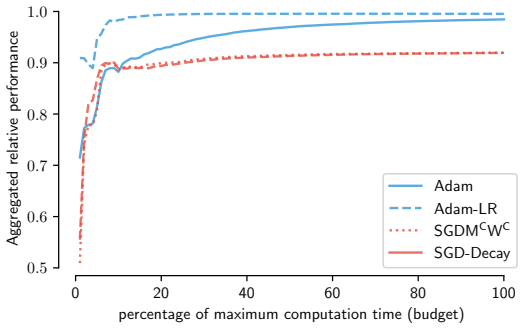


Figure 7: Aggregated relative performance of each optimizer across datasets.

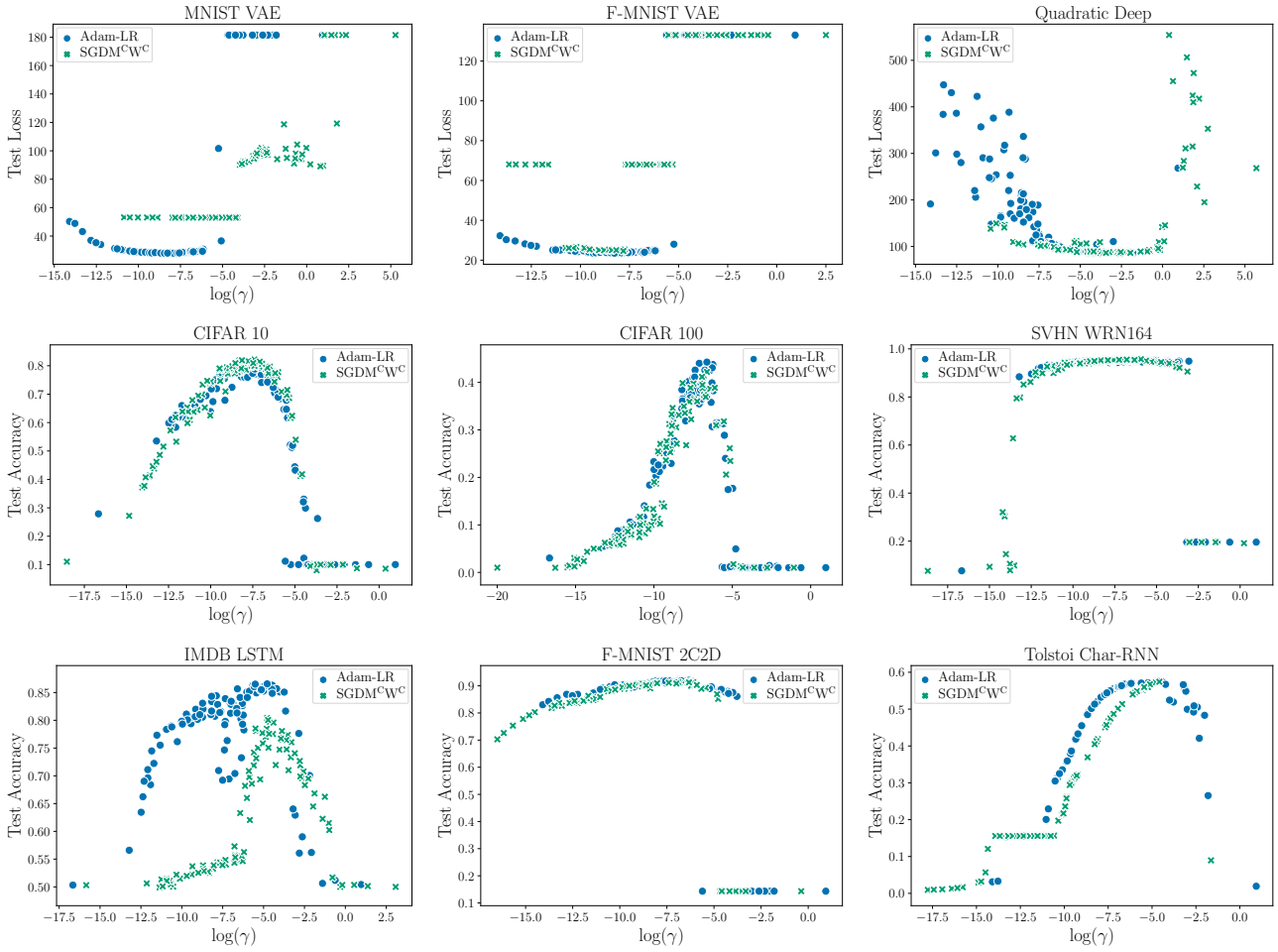


Figure 8: Scatter plot of performance of Adam-LR and SGD-M^{C}W^C by learning rate value. For better visibility, we shift the learning rate values of SGD-M^{C}W^C in such a way that the minima of both optimizers are at the same position on the x-axis.

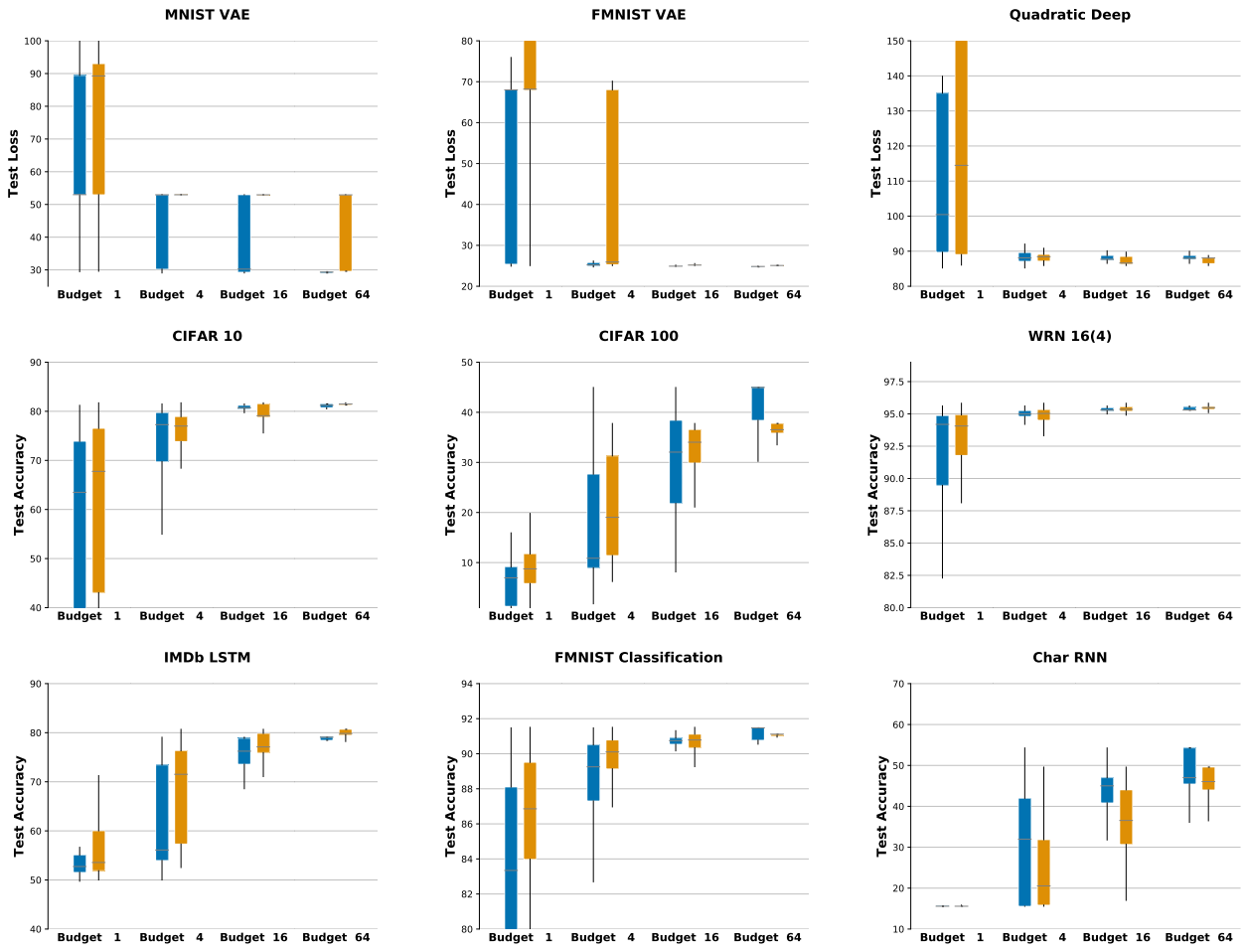


Figure 9: Performance of **SGD-MW**, **SGD-LR$_{\text{eff}}$**, at various hyperparameter search budgets. Image is best viewed in color. Some of the plots have been truncated to increase readability.

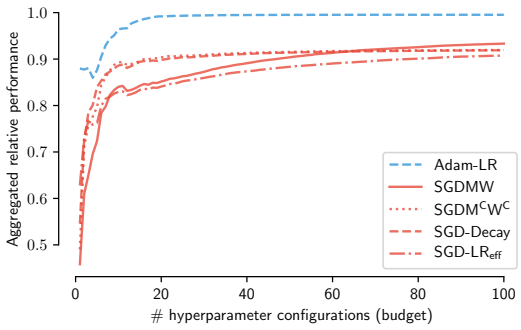


Figure 10: Aggregated relative performance of SGD-LR$_{\text{eff}}$ compared to other optimizers.