# HOW MUCH PROGRESS HAVE WE MADE IN NEURAL NETWORK TRAINING? A NEW EVALUATION PROTOCOL FOR BENCHMARKING OPTIMIZERS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Many optimizers have been proposed for training deep neural networks, and they often have multiple hyperparameters, which make it tricky to benchmark their performance. In this work, we propose a new benchmarking protocol to evaluate both end-to-end efficiency (training a model from scratch without knowing the best hyperparameter) and data-addition training efficiency (the previously selected hyperparameters are used for periodically re-training the model with newly collected data). For end-to-end efficiency, unlike previous work that assumes random hyperparameter tuning, which over-emphasizes the tuning time, we propose to evaluate with a bandit hyperparameter tuning strategy. A human study is conducted to show that our evaluation protocol matches human tuning behavior better than the random search. For data-addition training, we propose a new protocol for assessing the hyperparameter sensitivity to data shift. We then apply the proposed benchmarking framework to 7 optimizers and various tasks, including computer vision, natural language processing, reinforcement learning, and graph mining. Our results show that there is no clear winner across all the tasks.

## 1 INTRODUCTION

Due to the enormous data size and non-convexity, stochastic optimization algorithms have become widely used in training deep neural networks. In addition to Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951), many variations such as Adagrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014) have been proposed. Unlike classical, hyperparameter free optimizers such as gradient descent and Newton's method[1], stochastic optimizers often hold multiple hyperparameters including learning rate and momentum coefficients. Those hyperparameters are critical not only to the speed, but also to the final performance, and are often hard to tune.

It is thus non-trivial to benchmark and compare optimizers in deep neural network training. And a benchmarking mechanism that focuses on the peak performance could lead to a false sense of improvement when developing new optimizers without considering tuning efforts. In this paper, we aim to rethink the role of hyperparameter tuning in benchmarking optimizers and develop new benchmarking protocols to reflect their performance in practical tasks better. We then benchmark seven recently proposed and widely used optimizers and study their performance on a wide range of tasks. In the following, we will first briefly review the two existing benchmarking protocols, discuss their pros and cons, and then introduce our contributions.

**Benchmarking performance under the best hyperparameters** A majority of previous benchmarks and comparisons on optimizers are based on the best hyperparameters. Wilson et al. (2017); Shah et al. (2018) made a comparison of SGD-based methods against adaptive ones under their best hyperparameter configurations. They found that SGD can outperform adaptive methods on several datasets under careful tuning. Most of the benchmarking frameworks for ML training also assume knowing the best hyperparameters for optimizers (Schneider et al., 2019; Coleman et al., 2017; Zhu et al., 2018). Also, the popular MLPerf benchmark evaluated the performance of optimizers under the best hyperparameter. It showed that ImageNet and BERT could be trained in 1 minute using the combination of good optimizers, good hyperparameters, and thousands of accelerators.

[1]The step sizes of gradient descent and Newton's method can be automatically adjusted by a line search procedure (Nocedal & Wright, 2006).

Despite each optimizer's peak performance being evaluated, benchmarking under the best hyperparameters makes the comparison between optimizers unreliable and fails to reflect their practical performance. First, the assumption of knowing the best hyperparameter is unrealistic. In practice, it requires a lot of tuning efforts to find the best hyperparameter, and the tuning efficiency varies greatly for different optimizers. It is also tricky to define the "best hyperparameter", which depends on the hyperparameter searching range and grids. Further, since many of these optimizers are sensitive to hyperparameters, some improvements reported for new optimizers may come from insufficient tuning for previous work.

**Benchmarking performance with random hyperparameter search** It has been pointed out in several papers that tuning hyperparameter needs to be considered in evaluating optimizers (Schneider et al., 2019; Asi & Duchi, 2019), but having a formal evaluation protocol on this topic is non-trivial. Only recently, two papers Choi et al. (2019) and Sivaprasad et al. (2020) take hyperparameter tuning time into account when comparing SGD with Adam/Adagrad. However, their comparisons among optimizers are conducted on random hyperparameter search. We argue that these comparisons could over-emphasize the role of hyperparameter tuning, which could lead to a pessimistic and impractical performance benchmarking for optimizers. This is due to the following reasons: First, in the random search comparison, each bad hyperparameter has to run fully (e.g., 200 epochs). In practice, a user can always stop the program early for bad hyperparameters if having a limited time budget. For instance, if the learning rate for SGD is too large, a user can easily observe that SGD diverges in a few iterations and directly stops the current job. Therefore, the random search hypothesis will over-emphasize the role of hyperparameter tuning and does not align with a real user's practical efficiency. Second, the performance of the best hyperparameter is crucial for many applications. For example, in many real-world applications, we need to re-train the model every day or every week with newly added data. So the best hyperparameter selected in the beginning might benefit all these re-train tasks rather than searching parameters from scratch. In addition, due to the expensive random search, random search based evaluation often focuses on the low-accuracy region[2], while practically we care about the performance for getting reasonably good accuracy.

**Our contributions** Given that hyperparameter tuning is either under-emphasized (assuming the best hyperparameters) or over-emphasize (assuming random search) in existing benchmarking protocols and comparisons, we develop **new evaluation protocols** to compare optimizers to reflect the real use cases better. Our evaluation framework includes two protocols. First, to evaluate the **end-to-end training efficiency** for a user to train the best model from scratch, we develop an efficient evaluation protocol to compare the accuracy obtained under various time budgets, including the hyperparameter tuning time. Instead of using the random search algorithm, we adopt the Hyperband (Li et al., 2017) algorithm for hyperparameter tuning since it can stop early for bad configurations and better reflect the real running time required by a user. Further, we also propose to evaluate the **data addition training efficiency** for a user to re-train the model with some newly added training data, with the knowledge of the best hyperparameter tuned in the previous training set. We also conduct **human studies** to study how machine learning researchers are tuning parameters in optimizers and how that aligns with our proposed protocols.

Based on the proposed evaluation protocols, we **study how much progress has recently proposed algorithms made compared with SGD or Adam**. Note that most of the recent proposed optimizers have shown outperforming SGD and Adam under the best hyperparameters for some particular tasks, but it is not clear whether the improvements are still significant when considering hyper-parameter tuning, and across various tasks. To this end, we conduct comprehensive experiments comparing state-of-the-art training algorithms, including SGD (Robbins & Monro, 1951), Adam (Kingma & Ba, 2014), RAdam (Liu et al., 2019), Yogi (Zaheer et al., 2018), LARS (You et al., 2017), LAMB (You et al., 2019), and Lookahead (Zhang et al., 2019), on a variety of training tasks including image classification, generated adversarial networks (GANs), sentence classification (BERT fine-tuning), reinforcement learning and graph neural network training. Our main conclusions are: 1) On CIFAR-10 and CIFAR-100, all the optimizers including SGD are competitive. 2) Adaptive methods are generally better on more complex tasks (NLP, GCN, RL). 3) There is no clear winner among adaptive methods. Although RAdam is more stable than Adam across tasks, Adam is still a very competitive baseline even compared with recently proposed methods.

[2]For instance, Sivaprasad et al. (2020) only reaches $< 50\%$ accuracy in their CIFAR-100 comparisons.

## 2 Related Work

**Optimizers.** Properties of deep learning make it natural to apply stochastic first order methods, such as Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951). Severe issues such as a zig-zag training trajectory and a uniform learning rate have been exposed, and researchers have then drawn extensive attention to modify the existing SGD for improvement. Along this line of work, tremendous progresses have been made including SGDM (Qian, 1999), Adagrad (Duchi et al., 2011), RMSProp (Tieleman & Hinton, 2012), and Adam (Kingma & Ba, 2014). These methods utilize momentums to stabilize and speed up training procedures. Particularly, Adam is regarded as the default algorithm due to its outstanding compatibility. Then variants such as Amsgrad (Reddi et al., 2019), Adabound (Luo et al., 2019), Yogi (Zaheer et al., 2018), and RAdam (Liu et al., 2019) have been proposed to resolve different drawbacks of Adam. Meanwhile, the requirement of large batch training has inspired the development of LARS (You et al., 2017) and LAMB (You et al., 2019). Moreover, Zhang et al. (2019) has put forward a framework called Lookahead to boost optimization performance by iteratively updating two sets of weights.

**Hyperparameter tuning methods.** Random search and grid search (Bergstra & Bengio, 2012) can be a basic hyperparameter tuning method in the literature. However, the inefficiency of these methods stimulates the development of more advanced search strategies. Bayesian optimization methods including Bergstra et al. (2011) and Hutter et al. (2011) accelerate random search by fitting a black-box function of hyperparameter and the expected objective to adaptively guide the search direction. Parallel to this line of work, Hyperband (Li et al., 2017) focuses on reducing evaluation cost for each configuration and early terminates relatively worse trials. Falkner et al. (2018) proposes BOHB to combine the benefits of both Bayesian Optimization and Hyperband. All these methods still require huge computation resources. A recent work (Metz et al., 2020) has tried to obtain a list of potential hyperparameters by meta-learning from thousands of representative tasks. We strike a balance between effectiveness and computing cost and leverage Hyperband in our evaluation protocol to compare a wider range of optimizers.

## 3 Proposed Evaluation Protocols

In this section, we introduce the proposed evaluation framework for optimizers. We consider two evaluation protocols, each corresponding to an important training scenario:

- **Scenario I (End-to-end training)**: This is the general training scenario, where a user is given an unfamiliar optimizer and task, the goal is to achieve the best validation performance after several trials and errors. In this case, the evaluation needs to include hyperparameter tuning time. We develop an efficiency evaluation protocol to compare various optimizers in terms of CPE and peak performance.
- **Scenario II (Data-addition training)**: This is another useful scenario encountered in many applications, where the same model needs to be retrained regularly after collecting some fresh data. In this case, a naive solution is to reuse the previously optimal hyperparameters and retrain the model. However, since the distribution is shifted, the result depends on the sensitivity to that shift.

We describe the detailed evaluation protocol for each setting in the following subsections.

### 3.1 End-to-end Training Evaluation Protocol

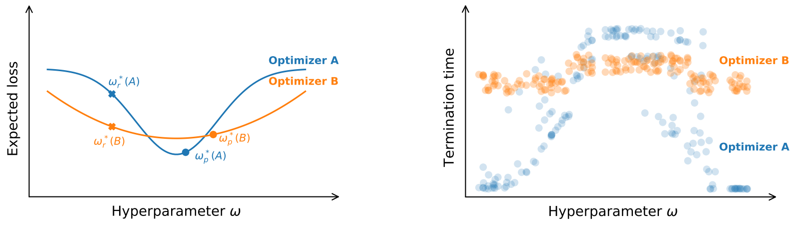
Before introducing our evaluation protocol for Scenario I, we first formally define the concept of optimizer and its hyperparameters.

**Definition 1.** *An optimizer is employed to solve a minimization problem $\min_\theta \mathcal{L}(\theta)$ and can be defined by a tuple $o \in \mathcal{O} = (\mathcal{U}, \Omega)$, where $\mathcal{O}$ contains all types of optimizers. $\mathcal{U}$ is a specific update rule and $\Omega = (\omega_1, \ldots, \omega_N) \in \mathbb{R}^N$ represents a vector of $N$ hyperparameters. Search space of these hyperparameters is denoted by $\mathcal{F}$. Given an initial parameter value $\theta_0$, together with a trajectory of optimization procedure $H_t = \{\theta_s, \mathcal{L}(\theta_s), \nabla\mathcal{L}(\theta_s)\}$, the optimizer updates $\theta$ by*

$$\theta_{t+1} = \mathcal{U}(H_t, \Omega).$$

We aim to evaluate the end-to-end time for a user to get the best model, including the hyperparameter tuning time. A recent work (Sivaprasad et al., 2020) assumes that a user conducts random search for finding the best hyperparameter setting. Still, we argue that the random search procedure will *over-emphasize* the importance of hyperparameters when tuning is considered — it assumes a user never stops the training even if they observe divergence or bad results in the initial training phase, which is unrealistic.

Figure 1 illustrates why random search might not lead to a fair comparison of optimizers. In Figure 1, we are given two optimizers, A and B, and their corresponding loss w.r.t. hyperparameter. According to Sivaprasad et al. (2020), optimizer B is considered better than optimizer A under a constrained budget since most regions of the hyperparameter space of A outperforms B. For instance, suppose we randomly sample the same hyperparamter setting for A and B. The final config $\omega_r^*(B)$ found under this strategy can have a lower expected loss than that of $\omega_r^*(A)$, as shown in Figure 1a. However, there exists a more practical search strategy which can invalidate this statement with the assumption of a limited searching budget: a user can early terminate a configuration trial when trapped in bad results or diverging. Hence, we can observe in Figure 1b that for optimizer A, this strategy early-stops many configurations and only allow a limited number of trials to explore to the deeper stage. Therefore, the bad hyperparameters will not affect the overall efficiency of Optimizer A too much. In contrast, for optimizer B, performances of different hyperparameters are relatively satisfactory and hard to distinguish, resulting in similar and long termination time for each trial. Therefore, it may be easier for a practical search strategy $p$ to find the best configuration $\omega_p^*(A)$ of optimizer A than $\omega_p^*(B)$, given the same constrained budget.



(a) (b)

Figure 1: An illustration example showing that different hyperparamter tuning methods are likely to affect comparison of optimizers. Optimizer A is more sensitive to hyperparamters than optimizers B, but it may be prefered if bad hyperparameters can be terminated in the early stage.

This example suggests that random search may over-emphasize the parameter sensitivity when benchmarking optimizers. To better reflect a practical hyperparameter tuning scenario, our evaluation assumes a user applies **Hyperband** (Li et al., 2017), a simple but effective hyperparameter tuning scheme to get the best model. Hyperband formulates hyperparameter optimization as a unique bandit problem. It accelerates random search through adaptive resource allocation and early-stopping, as demonstrated in Figure 1b. Compared with its more complicated counterparts such as BOHB (Falkner et al., 2018), Hyperband requires less computing resources and performs similarly within a constrained budget. The algorithm is presented in Appendix A.

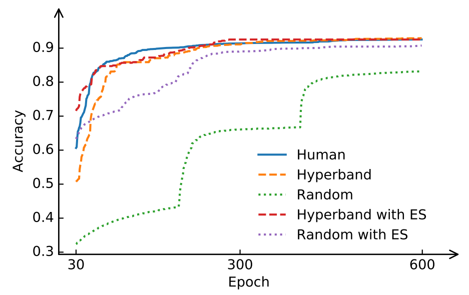


Figure 2: Hyperband tuning used in our evaluation protocol is closer to human behavior than random search.

Despite different hyperparameter tuning algorithms, human tuning by experts is still regarded as the most effective. To verify and support that Hyperband is an effective method and even competitive with humans, we conduct a human study as follows: for image classification on CIFAR10, given 10 learning rate configurations of SGD in the grid $[1.0 \times 10^{-8}, 1.0 \times 10^{-7}, 1.0 \times 10^{-6}, \ldots, 10]$, participants are requested to search the best one at their discretion. Namely, they can stop or pause a trial any time and continue to evaluate a new configuration until they feel it has already reached the best performance. 10 participants are sampled randomly from Ph.D. students with computer science backgrounds. We collect their tuning trajectories and average them as human performance, which is considered "optimal" in this human study. In Figure 2, we plot curves for hyperparameter tuning of human, Hyperband, random search, random search with an early stopping (ES) strategy in Sivaprasad et al. (2020), and Hyperband with ES. We find that Hyperband matches humans' behavior better, while random search tends to trap in suboptimal configurations although random with early stopping can mitigate this issue to some extent. This finding shows the advantage of Hyperband over random search regardless of early stopping, and justifies the use of Hyperband in optimizer benchmarking. More details of this human study can be found in Appedix B.

With Hyperband incorporated in end-to-end training, we assume that each configuration is run sequentially and record the best performance obtained at time step $t$ as $P_t$. Specifically, $P_t$ represents the evaluation metric for each task, e.g., accuracy for image classification and return for reinforcement learning. $\{P_t\}_{t=1}^T$ forms a trajectory for plotting learning curves on test set like Figure 3. Although it is intuitive to observe the performance of different optimizers according to such figures, summarizing a learning curve into a quantifiable, scalar value can be more insightful for evaluation. Thus, as shown in Eq. 1, we use $\lambda$-tunability defined in Sivaprasad et al. (2020) to further measure the performance of optimizers:

$$\lambda\text{-tunability} = \sum_{t=1}^{T} \lambda_t \cdot P_t, \text{ where} \sum_t \lambda_t = 1 \text{ and } \forall_t \lambda_t > 0. \tag{1}$$

One intuitive way is to set $\lambda_t = \mathbf{1}_{t=T}$ to determine which optimizer can reach the best model performance after the whole training procedure. However, merely considering the peak performance is not a good guidance on the choice of optimizers. In practice, we tend to take into account the complete trajectory and exert more emphasis on the early stage. Thus, we employ the Cumulative Performance-Early weighting scheme where $\lambda_t \propto (T - i)$, to compute $\lambda$-tunablity instead of the extreme assignment $\lambda_t = \mathbf{1}_{t=T}$. The value obtained is termed as *CPE* for simplicity.

We present our evaluation protocol in Algorithm 1. As we can see, end-to-end training with hyperparameter optimization is conducted for various optimizers on the given task. The trajectory $\{P_t\}_{t=1}^T$ is recorded to compute the peak performance as well as *CPE* value. Note that the procedure is repeated $M$ times to obtain a reliable result. We use $M = 3$ in all experiments. More details of time cost and acceleration of the algorithm can be found in Appendix E.

**Algorithm 1** End-to-End Efficiency Evaluation Protocol

```
Input: A set of optimizers $\mathcal{O} = \{o : o = (\mathcal{U}, \Omega)\}$, task $a \in \mathcal{A}$, feasible search space $\mathcal{F}$
1: for $o \in \mathcal{O}$ do
2:     for $i = 1$ to $M$ do
3:         Conduct hyperparameter search in $\mathcal{F}$ with the optimizer $o$ using HyperBand on $a$
4:         Record the performance trajectory $\{P_t\}_{t=1}^T$ explored by HyperBand
5:         Calculate the peak performance and CPE by Eq. 1
6:     end for
7:     Average peak and CPE values over $M$ repetitions for the optimizer $o$
8: end for
9: Evaluate optimizers according to their peak and CPE values
```

## 3.2 Data-addition Training Evaluation Protocol

In Scenario II, we have a service (e.g., a search or recommendation engine) and we want to re-train the model every day or every week with some newly added training data. One may argue that an online learning algorithm should be used in this case, but in practice online learning is unstable and industries still prefer this periodically retraining scheme which is more stable.

In this scenario, once the best hyperparameters were chosen in the beginning, we can reuse them for every training, so no hyperparameter tuning is required and the performance (including both efficiency and test accuracy) under the best hyperparameter becomes important. However, an implicit assumption made in this process is that *"the best hyperparameter will still work when the training task slightly changes"*. This can be viewed as transferability of hyperparameters for a particular optimizer, and our second evaluation protocol aims to evaluate this practical scenario.

We simulate data-addition training with all classification tasks, and the evaluation protocol works as follows: 1) Extract a subset $\mathcal{D}_\delta$ containing partial training data from the original full dataset $\mathcal{D}$ with a small ratio $\delta$; 2) Conduct a hyperparameter search on $\mathcal{D}_\delta$ to find the best setting under this scenario; 3) Use these hyperparameters to train the model on the complete dataset; 4) Observe the potential change of the ranking of various optimizers before and after data addition. For step 4) when comparing different optimizers, we will plot the training curve in the full-data training stage in Section 4, and also summarize the training curve using the *CPE* value. The detailed evaluation protocol is described in Algorithm 2.

**Algorithm 2** Data-Addition Training Evaluation Protocol

```
Input: A set of optimizers $\mathcal{O} = \{o : o = (\mathcal{U}, \Omega)\}$, task $a \in \mathcal{A}$ with a full dataset $\mathcal{D}$, a split ratio $\delta$
1: for $o \in \mathcal{O}$ do
2:     for $i = 1$ to $M$ do
3:         Conduct hyperparameter search with the optimizer $o$ using Hyperband on $a$ with a partial dataset $\mathcal{D}_\delta$, and record the best hyperparameter setting $\Omega_{\text{partial}}$ found under this scenario
4:         Apply the optimizer with $\Omega_{\text{partial}}$ on $\mathcal{D}_\delta$ and $\mathcal{D}$, then save the training curves
5:     end for
6:     Average training curves of $o$ over $M$ repetitions to compute CPE
7: end for
8: Compare performance of different optimizers under data-addition training
```

## 4 Experimental Results

**Optimizers to be evaluated.** As shown in Table 1, we consider 7 optimizers including non-adaptive methods using only the first-order momentum, and adaptive methods considering both first-order and second-order momentum. We also provide lists of tunable hyperparameters for different optimizers in Table 1. Moreover, we consider following two combinations of tunable hyperparameters to better investigate the performance of different optimizers: **a)** only tuning initial learning rate with the others set to default values and **b)** tuning a full list of hyperparameters. A detailed description of optimizers as well as default values and search range of these hyperparameters can be found in Appendix E. Note that we adopt a unified search space for a fair comparison following Metz et al. (2020), to eliminate biases of specific ranges for different optimizers. The tuning budget of Hyperband is determined by three items: maximum resource (in this paper we use epoch) per configuration $R$, reduction factor $\eta$, and number of configurations $n_c$. According to Li et al. (2017), a single Hyperband execution contains $n_s = \lfloor \log_\eta(R) \rfloor + 1$ of SuccessiveHalving, each referred to as a bracket. These brackets take strategies from least to most aggressive early-stopping, and each one is designed to use approximately $B = R \cdot n_s$ resources, leading to a finite total budget. The number of randomly sampled configurations in one Hyperband run is also fixed and grows with $R$. Then given $R$ and $\eta$, $n_c$ determines the repetition times of Hyperband. We set $\eta = 3$ as this default value performs consistently well, and $R$ to a value which each task usually takes for a complete run. For $n_c$, it is assigned as what is required for a single Hyperband execution for all tasks, except for BERT fine-tuning, where a larger number of configurations is necessary due to a relatively small $R$. In Appendix E, we give assigned values of $R$, $\eta$, and $n_c$ for each task.

| Optimizer | | Tunable hyperparameter |
|---|---|---|
| Non-adaptive | SGD | $\alpha_0$, $\mu$ |
| | LARS | $\alpha_0$, $\mu$, $\epsilon$ |
| Adaptive | Adam, RAdam, Yogi<br>Lookahead, LAMB | $\alpha_0$, $\beta_1$, $\beta_2$, $\epsilon$ |

Table 1: Optimizers to be evaluated with their tunable hyperparameters. Specifically, $\alpha_0$ represents the initial learning rate. $\mu$ is the decay factor of the first-order momentum for non-adaptive methods while $\beta_1$ and $\beta_2$ are coefficients to compute the running averages of first-order and second-order momentums. $\epsilon$ is a small scalar used to prevent division by 0.

**Tasks for benchmarking.** For a comprehensive and reliable assessment of optimizers, we consider a wide range of tasks in different domains. When evaluating end-to-end training efficiency, we implement our protocol on tasks covering several popular and promising applications in Table 2. Apart from common tasks in computer vision and natural language processing, we introduce two extra tasks in graph neural network training and reinforcement learning. For simplicity, we will use the dataset to represent each task in our subsequent tables of experimental results. (For the reinforcement learning task, we just use the environment name.) The detailed settings and parameters for each task can be found in Appendix D.

| **Domain** | **Task** | **Metric** | **Model** | **Dataset** |
|---|---|---|---|---|
| Computer Vision | Image Classification | Accuracy | ResNet-50 | CIFAR10<br>CIFAR100 |
| | VAE | NLL | CNN Autoencoder | CelebA |
| | GAN | FID | SNGAN network | CIFAR10 |
| NLP | GLUE benchmark | Accuracy | RoBERTa-base | MRPC |
| Graph network training | Node labeling | F1 score | Cluster-GCN | PPI |
| Reinforcement Learning | Walker2d-v3 | Return | PPO | × |

Table 2: Tasks for benchmarking optimizers. Details are provided in Appendix D.

### 4.1 End-to-end efficiency (Secnario I)

To evaluate end-to-end training efficiency, we adopt the protocol in Algorithm 1. Specifically, we record the average training trajectory with Hyperband $\{P_t\}_{t=1}^T$ for each optimizer on benchmarking tasks, where $P_t$ is the evaluation metric for each task (e.g., accuracy, return). We visualize these trajectories in Figure 3 for CIFAR10 and CIFAR100, and calculate *CPE* and peak performance in Table 3 and 9 respectively. More results for other tasks and the peak performance can be found in Appendix F. Besides, in Eq. 2 we compute *performance ratio* $r_{o,a}$ for each optimizer and each task, and then utilize the distribution function of a performance metric called *performance profile* $\rho_o(\tau)$ to summarize the performance of different optimizers over all the tasks. For tasks where a lower *CPE* is better, we just use $r_{o,a} = CPE_{o,a}/\min\{CPE_{o,a}\}$ instead to guarantee $r_{o,a} \geq 1$. The function $\rho_o(\tau)$ for all optimizers is presented in Figure 4. Based on the definition of performance profile (Dolan & Moré, 2002), the optimizers with large probability $\rho_o(\tau)$ are to be preferred. In particular, the value of $\rho_o(1)$ is the probability that one optimizer will win over the rest and can be a reference for selecting the proper optimizer for an unknown task. We also provided a probabilistic performance profile to summarize different optimizers in Figure 7 in Appendix F.

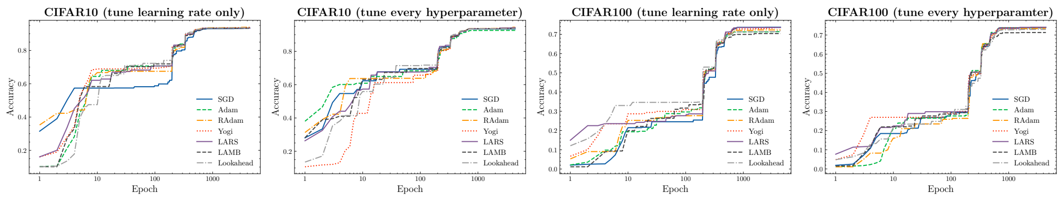
$$r_{o,a} = \frac{\max\{CPE_{o,a} : o \in \mathcal{O}\}}{CPE_{o,a}}, \quad \rho_o(\tau) = \frac{1}{|\mathcal{A}|}\text{size}\{a \in \mathcal{A} : r_{o,a} \leq \tau\}. \tag{2}$$

Our findings are summarized below:

- It should be emphasized from Table 3 and 9, that under our protocol based on Hyperband, SGD performs similarly to Adam in terms of efficiency as well as peak performance, and can even surpass it in some cases like training on CIFAR100. Under Hyperband, the best configuration of SGD is less tedious to find than random search because Hyperband can early-stop bad runs and thus they will affect less to the search efficiency and final performance.
- For image classification tasks all the methods are competitive, while adaptive methods tend to perform better in more complicated tasks (NLP, GCN, RL).
- There is no significant distinction among adaptive variants. Performance of adaptive optimizers tends to fall in the range within 1% of the best result.
- According to performance profile in Figure 4, the RAdam achieves probability 1 with the smallest $\tau$, and Adam is the second method achieving that. This indicates that RAdam and Adam are achieving relatively stable and consistent performance among these tasks.

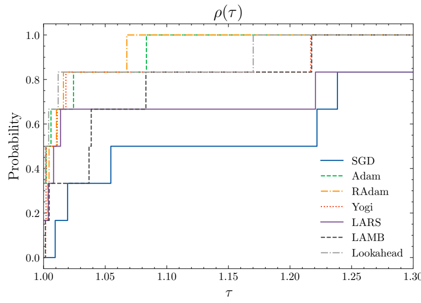
Table 3: CPE for different optimizers on benchmarking tasks. The best performance is highlighted in bold and blue and results within the 1% range of the best are emphasized in bold only.

| **Optimizer** | **CIFAR10 (%) ↑**<br>(classification) | **CIFAR100 (%) ↑**<br>(classification) | **CelebA ↓**<br>(VAE) | **MRPC ↑**<br>(NLP) | **PPI (%) ↑**<br>(GCN) | **Walker2d-v3 ↑**<br>(RL) |
|---|---|---|---|---|---|---|
| *Tune learning rate only:* | | | | | | |
| SGD | 88.87 ± 0.23 | **66.85 ± 0.12** | 0.1430 ± 0.0038 | 69.90 ± 0.69 | 76.77 ± 0.08 | 2795 ± 275 |
| Adam | **90.42 ± 0.10** | 65.88 ± 0.23 | **0.1356 ± 0.0001** | **84.90 ± 0.72** | **95.08 ± 0.01** | 3822 ± 78 |
| RAdam | **90.29 ± 0.11** | 66.41 ± 0.15 | **0.1362 ± 0.0001** | **85.41 ± 1.45** | 94.10 ± 0.04 | 3879 ± 201 |
| Yogi | **90.42 ± 0.04** | **67.37 ± 0.50** | 0.1371 ± 0.0004 | 70.19 ± 0.90 | 93.39 ± 0.02 | **4132 ± 205** |
| LARS | **90.25 ± 0.07** | **67.48 ± 0.04** | **0.1367 ± 0.0002** | 69.97 ± 0.54 | 93.79 ± 0.01 | 2986 ± 105 |
| LAMB | **90.19 ± 0.08** | 65.08 ± 0.06 | **0.1358 ± 0.0003** | 82.23 ± 1.49 | 87.79 ± 0.07 | 3401 ± 235 |
| Lookahead | **90.60 ± 0.06** | 65.60 ± 0.07 | **0.1358 ± 0.0004** | 72.99 ± 1.33 | **94.69 ± 0.02** | **4141 ± 264** |
| *Tune every hyperparameter:* | | | | | | |
| SGD | **90.20 ± 0.16** | **67.36 ± 0.10** | 0.1407 ± 0.0011 | 71.53 ± 1.21 | 94.64 ± 0.01 | 2978 ± 91 |
| Adam | 89.27 ± 1.40 | **67.57 ± 0.23** | 0.1389 ± 0.0002 | **85.23 ± 1.44** | 92.62 ± 0.04 | **4080 ± 459** |
| RAdam | **90.14 ± 0.44** | 66.90 ± 0.05 | **0.1366 ± 0.0006** | 84.32 ± 1.91 | 93.05 ± 0.04 | 3813 ± 103 |
| Yogi | **89.83 ± 0.21** | **67.65 ± 0.08** | 0.1401 ± 0.0019 | 68.42 ± 1.02 | 88.94 ± 0.11 | 3778 ± 249 |
| LARS | **90.42 ± 0.20** | **67.78 ± 0.28** | 0.1375 ± 0.0005 | 77.40 ± 3.09 | **96.34 ± 0.01** | 2728 ± 136 |
| LAMB | **90.27 ± 0.40** | 65.59 ± 0.03 | 0.1382 ± 0.0001 | **84.66 ± 2.61** | 93.18 ± 0.05 | 2935 ± 57 |
| Lookahead | **90.44 ± 0.11** | 66.46 ± 0.45 | **0.1360 ± 0.0005** | 79.05 ± 2.99 | 94.30 ± 0.04 | 3786 ± 137 |

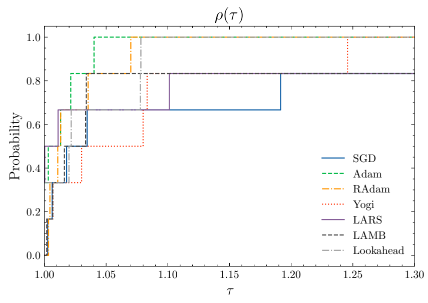


(a) (b) (c) (d)

Figure 3: End-to-end training curves with Hyperband on CIFAR10 and CIFAR100.



(a) Tune learning rate only.



(b) Tune every hyperparameter.

Figure 4: Performance profile of 7 optimizers in the range $[1.0, 1.3]$.

## 4.2 Data-addition Training (Scenario II)

We then conduct evaluation on data-addition training based on the protocol in Algorithm 2. We choose four classification problems on CIFAR10, CIFAR100, MRPC and PPI since the setting do not apply to RL. We search the best hyperparameter configuration, denoted by $\Omega_{\text{partial}}$, under the sub training set with the ratio $\delta = 0.3$. Here we tune all hyperparameters. Then we directly apply $\Omega_{\text{partial}}$ on the full dataset for a complete training process. The training curves are shown in Figure 5, and we also summarize the training curve with *CPE* by Eq. 1 in Table 4. We have the following findings:

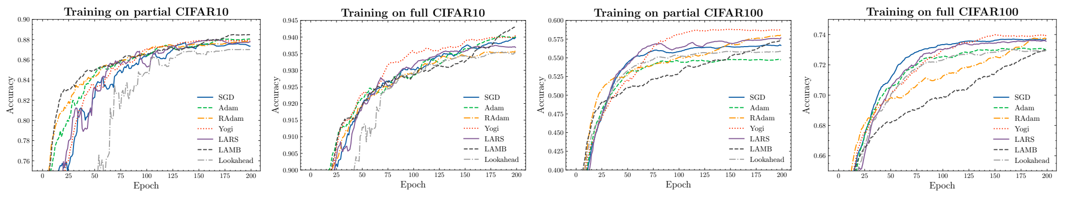
- There is no clear winner in data-addition training. RAdam is outperforming other optimizers in 2/4 tasks so is slightly preferred, but other optimizers except Lookahead are also competitive (within 1% range) on at least 2/4 tasks.
- To investigate whether the optimizer's ranking will change when adding 70% data, we compare the training curve on the original 30% data versus the training curve on the full 100% data in Figure 5. We observe that the ranking of optimizers slightly changes after data addition.

Table 4: CPE of different optimizers computed under curves trained with $\Omega_{\text{partial}}$ on four full datasets.

| Optimizer | CIFAR10 (%)↑ | CIFAR100 (%)↑ | MRPC (%)↑ | PPI↑ |
|---|---|---|---|---|
| SGD | **90.04 ± 0.16** | **67.91 ± 0.23** | 66.62 ± 3.47 | 66.830 ± 0.010 |
| Adam | **90.52 ± 0.03** | 67.04 ± 0.27 | 73.13 ± 1.16 | **70.420 ± 0.007** |
| RAdam | **90.30 ± 0.14** | 67.06 ± 0.17 | **79.01 ± 3.10** | **70.840 ± 0.010** |
| Yogi | **89.63 ± 0.39** | **67.58** ± 0.19 | 68.40 ± 1.68 | 67.990 ± 0.003 |
| LARS | **90.17 ± 0.13** | **67.29 ± 0.14** | 64.43 ± 2.72 | 68.400 ± 0.005 |
| LAMB | **90.51 ± 0.07** | 66.13 ± 0.02 | **78.94 ± 1.25** | 70.110 ± 0.008 |
| Lookahead | 88.36 ± 0.06 | 67.10 ± 0.31 | 68.81 ± 1.22 | 69.710 ± 0.003 |

## 5 Conclusions and Discussions

In conclusion, we found **there is no strong evidence that newly proposed optimizers consistently outperform Adam**, while each of them may be good for some particular tasks. When deciding the choice of the optimizer for a specific task, people can refer to results in Table 3 and 9. If the task is contained in Table 2, he/she can directly choose the one with the best CPE or best peak performance based on his/her goal of the task (easy to tune or high final performance). On the other hand, even though the desired task is covered, people can also gain some insights from the results of the most similar task in Table 2, or refer to the performance profile in Figure 4 to pick adaptive methods like Adam. Besides choosing the optimizer, it will contribute to designing a new optimizer as well. Using our protocol to evaluate a new optimizer can show whether it has obvious improvement over existing optimizers, and can serve as a routine to judge the performance of the optimizer thoroughly.



(a) (b) (c) (d)

Figure 5: Training curves under $\Omega_{\text{partial}}$ for both partial and full datasets.

In addition to the proposed two evaluation criteria, there could be other factors that affect the practical performance of an optimizer. First, the **memory consumption** is becoming important for training large DNN models. For instance, although Lookahead performs well in certain tasks, it requires more memory than other optimizers, restricting their practical use in some memory constrained applications. Another important criterion is the **scalability** of optimizers. When training with a massively distributed system, optimizing the performance of a large batch regime (e.g., 32K batch size for ImageNet) is important. LARS and LAMB algorithms included in our study are developed for large batch training. We believe it is another important metric for comparing optimizers worth studying further.

## References

Joshua Achiam. Spinning Up in Deep Reinforcement Learning, 2018.

Hilal Asi and John C Duchi. The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*, 116(46):22924–22930, 2019.

André MS Barreto, Heder S Bernardino, and Helio JC Barbosa. Probabilistic performance profiles for the experimental evaluation of stochastic algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 751–758, 2010.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pp. 2546–2554, 2011.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

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

Frank Schneider, Lukas Balles, and Philipp Hennig. Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Vatsal Shah, Anastasios Kyrillidis, and Sujay Sanghavi. Minimum weight norm models do not always generalize well for over-parameterized problems. *arXiv preprint arXiv:1811.07055*, 2018.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and Francois Fleuret. Optimizer benchmarking needs to account for hyperparameter tuning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in neural information processing systems*, pp. 4148–4158, 2017.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Advances in neural information processing systems*, pp. 9793–9803, 2018.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pp. 9597–9608, 2019.

Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. Tbd: Benchmarking and analyzing deep neural network training. *arXiv preprint arXiv:1803.06905*, 2018.

## A HyperBand

We present the whole algorithm for Hyperband in Algorithm 3, and you can refer to Li et al. (2017) for more details.

**Algorithm 3** Hyperband

```
Input: R, $\eta$
Initialization: $s_{\max} = \lfloor \log_\eta R \rfloor$ , $B = (s_{\max} + 1)R$
 1: for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ do
 2:     $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
 3:     // begin SuccessiveHalving with $(n, r)$ inner loop
 4:     $T$ = random_get_configuration($n$)
 5:     for $i \in \{0, \ldots, s\}$ do
 6:         $n_i = \lfloor n\eta^{-i} \rfloor$, $r_i = r\eta^i$
 7:         $L$ = {run_then_return_val_loss($t, r_i$): $t \in T$}
 8:         $T$ = top_k($T, L, \lfloor n_i/\eta \rfloor$)
 9:     end for
10: end for
return Hyperparameter configuration with the smallest loss seen so far
```

## B Details of Human Study

In this human study, 10 participants are sampled from Ph.D. students with computer science backgrounds (machine learning backgrounds specifically). They are recruited as follows: We first asked the administrators to distribute the program of this human study to Ph.D. students in machine learning labs in our institutions. Provided the population, we assumed that they had prior knowledge about some basic machine learning experiments, such as image classification on MNIST and CIFAR10. They were requested to conduct hyperparameter tuning of learning rate based on their knowledge. They were informed that the target experiment was image classification on CIFAR10 with SGD, and the search range of learning rate was in the grid $[1.0 \times 10^{-8}, 1.0 \times 10^{-7}, 1.0 \times 10^{-6}, \ldots, 10]$. Each configuration was run for 200 epochs at most. Moreover, we also told them that they could pause any configuration if they wanted to evaluate others, and even stop the whole tuning process only when they thought the accuracy could not be further improved and the number of total tuning epochs exceeded 600 at the same time. We collected results from 17 people and we determined the validity by checking whether the length of the trajectory was greater than 600 and removed 7 invalidity trajectories. Finally there remained 10 trajectories and we averaged them as the human performance in Figure 2.

## C OPTIMIZERS

**Notations.** Given a vector of parameters $\theta \in \mathbb{R}^d$, we denote a sub-vector of its i-th layer's parameters by $\theta^{(i)}$. $\{\alpha_t\}_{t=1}^T$ is a sequence of learning rates during the optimization procedure of a horizon $T$. $\{\phi_t, \psi_t\}_{t=1}^T$ represents a sequence of functions to calculate the first-order and second-order momentum of the gradient $g_t$, which are $m_t$ and $v_t$ respectively at time step $t$. Different optimization algorithms are usually specified by the choice of $\phi(\cdot)$ and $\psi(\cdot)$. $\{r_t\}_{t=1}^T$ is an additional sequence of adaptive terms to modify the magnitude of the learning rate in some methods. For algorithms only using the first-order momentum, $\mu$ is the , while $\beta_1$ and $\beta_2$ are coefficients to compute the running averages $m$ and $v$. $\epsilon$ is a small scalar (e.g., $1 \times 10^{-8}$) used to prevent division by 0.

**Generic optimization framework.** Based on, we further develop a thorough generic optimization framework including an extra adaptive term in Algorithm 4. The debiasing term used in the original version of Adam is ignored for simplicity. Note that for $\{\alpha_t\}_{t=1}^T$, different learning rate scheduling strategies can be adopted and the choice of scheduler is also regarded as a tunable hyperparameter. Without loss of generality, in this paper, we only consider a constant value and a linear decay (Shallue et al., 2018) in the following equation, introducing $\gamma$ as a hyperparameter.

$$\alpha_t = \begin{cases} \alpha_0, & \text{constant;} \\ \alpha_0 - (1-\gamma)\alpha_0 \frac{t}{T}, & \text{linear decay.} \end{cases}$$

With this generic framework, we can summarize several popular optimization methods by explicitly specifying $m_t$, $v_t$ and $r_t$ in Table 5. It should be clarified that Lookahead is an exception of the generic framework. In fact it is more like a high-level mechanism, which can be incorporated with any other optimizer. However, as stated in Zhang et al. (2019), this optimizer is robust to inner optimization algorithm, $k$, and $\alpha_s$ in Algorithm 5, we still include Lookahead here with Adam as the base for a more convincing and comprehensive evaluation. We consider Lookahead as a special adaptive method, and tune the same hyperparamters for it as other adaptive optimziers.

**Algorithm 4** Generic framework of optimization methods

```
Input: parameter value $\theta_1$, learning rate with scheduling $\{\alpha_t\}$, sequence of functions $\{\phi_t, \psi_t, \chi_t\}_{t=1}^T$ to compute $m_t$, $v_t$, and $r_t$ respectively.
1: for $t = 1$ to $T$ do
2:     $g_t = \nabla f_t(\theta_t)$
3:     $m_t = \phi_t(g_1, \cdots, g_t)$
4:     $v_t = \psi_t(g_1, \cdots, g_t)$
5:     $r_t = \chi_t(\theta_t, m_t, v_t)$
6:     $\theta_{t+1} = \theta_t - \alpha_t r_t m_t / \sqrt{v_t}$
7: end for
```

Table 5: A summary of popular optimization algorithms with different choices of $m_t$, $v_t$ and $r_t$.

| | $m_t$ | $v_t$ | $r_t$ |
|---|---|---|---|
| SGD(M) | $\mu m_{t-1} + g_t$ | $1$ | $1$ |
| Adam | $\beta_1 m_{t-1} + (1-\beta_1) g_t$ | $\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ | $1$ |
| RAdam | $\beta_1 m_{t-1} + (1-\beta_1) g_t$ | $\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ | $\sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$ |
| Yogi | $\beta_1 m_{t-1} + (1-\beta_1) g_t$ | $v_{t-1} - (1-\beta_2)\operatorname{sign}(v_{t-1} - g_t^2) g_t^2$ | $1$ |
| LARS | $\mu m_{t-1} + g_t$ | $1$ | $\|\theta_t^{(i)}\| / \|m_t^{(i)}\|$ |
| LAMB | $\beta_1 m_{t-1} + (1-\beta_1) g_t$ | $\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ | $\frac{\|\theta_t^{(i)}\|}{\|m_t^{(i)} / \sqrt{v_t^{(i)}}\|}$ |
| Lookahead* | $\beta_1 m_{t-1} + (1-\beta_1) g_t$ | $\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ | $1$ |

**Algorithm 5** Lookahead Optimizer

**Input:** Initial parameters $\theta_0$, objective function $f$, synchronization period $k$, slow weights step size $\alpha_s$, optimizer $A$

```
1: for $t = 1, 2, \ldots$ do
2:     Synchronize parameters $\hat{\theta}_{t,0} \leftarrow \theta_{t-1}$
3:     for $i = 1, 2, \ldots, k$ do
4:         Sample minibatch of data $d \in \mathcal{D}$
5:         $\hat{\theta}_{t,i} \leftarrow \hat{\theta}_{t,i-1} + A(L, \hat{\theta}_{t,i-1}, d)$
6:     end for
7:     Perform outer update $\theta_t \leftarrow \theta_{t-1} + \alpha_s(\hat{\theta}_{t,k} - \theta_{t-1})$
8: end for
return Parameters $\theta$
```

## D Task description

We make a concrete description of tasks selected for our optimizer evaluation protocol:

- Image classifcation. For this task, we adopt a ResNet-50 (He et al., 2016) model on CIFAR10 and CIFAR100 with a batch size of 128 and the maximum epoch of 200 per trial.
- VAE. We use a vanilla variational autoencoder in Kingma & Welling (2013) with five convolutional and five deconvolutional layers with a latent space of dimension 128 on CelebA. There are no dropout layers. Trained with a batch size of 144.
- GAN. We train SNGAN with the same network architecture and objective function with spectral normalization for CIFAR10 in Miyato et al. (2018), and the batch size of the generator and the discriminator is 128 and 64 respectively.
- Natural language processing. In this domain, we finetune RoBERTa-base on MRPC, one of the test suit in GLUE benchmark. For each optimizer, we set the maximal exploration budget to be 800 epochs. The batch size is 16 sentences.
- Graph learning. Among various graph learning problems, we choose node classification as semi-supervised classification. In GCN training, in there are multiple ways to deal with the neighborhood explosion of stochastic optimizers. We choose Cluster-GCN Chiang et al. (2019) as the backbone to handle neighborhood expansion and PPI as the dataset.
- Reinforcement learning. We select Walker2d-v3 from OpenAI Gym (Brockman et al. (2016)) as our training environment, and PPO (Schulman et al. (2017)), implemented by OpenAI SpinningUp (Achiam (2018)), as the algorithm that required tuning. We use the same architectures for both action value network $Q$ and the policy network $\pi$. We define 40,000 of environment interactions as one epoch, with a batch size of 4,000. The return we used is the highest average test return of an epoch during the training.

## E Implementation Details

Implementation details of our experiments are provided in this section. Specifically, we give the unified search space for all hyperparamters and their default values in Table 6. Note that we tune the learning rate decay factor for image classification tasks when tuning every hyperparamter. For the task on MRPC, $\gamma$ is tuned for all experiments. In other cases, we only tune original hyperparamters without a learning rate scheduler.

In addition, Hyperband parameter values for each task are listed in Table 7. These parameters are assigned based on properties of different tasks.

For time cost of our evaluation protocols, it depends on how many budgets are available. Specifically, in our paper, the unit of time budget is one epoch, then the total time will be $B_{epoch} * T_{epoch}$, where $B_{epoch}$ is the total available budget and $T_{epoch}$ is the running time for one epoch. There is no additional computational cost, i.e., running our protocol once takes the same time as running one hyperparameter search with Hyperband. In our experiment on CIFAR10, we roughly evaluated 200 hyperparameter configurations in one Hyperband running, while the same time can only allow about 50 configurations in random search.

| **Hyperparamter** | **Search space** | **Default value** |
|---|---|---|
| $\alpha_0$ | Log-Uniform$(-8, 1)$ | $\times$ |
| $\mu$ | Uniform$[0, 1]$ | 0 for SGD and 0.9 for LARS |
| $1 - \beta_1$ | Log-Uniform$(-4, 0)$ | 0.1 |
| $1 - \beta_2$ | Log-Uniform$(-6, 0)$ | 0.001 |
| $\epsilon$ | Log-Uniform$(-8, 1)$ | 1e-8 |
| $\gamma$ | Log-Uniform$(-4, 0)$ | $\times$ |

Table 6: Hyperparamter search space and default value

| **Task** | **Hyperband parameter** | | |
|---|---|---|---|
| | $R$ | $n_c$ | $\eta$ |
| Image Classification | 200 | 172 | $\eta = 3$ |
| VAE | 50 | 62 | |
| GAN | 200 | 172 | |
| GLUE benchmark | 10 | 200 | |
| Graph learning | 200 | 200 | |
| RL | 200 | 172 | |

Table 7: Hyperband parameters for each task.

Moreover, we can further accelerate our evaluation protocol by resampling, shown in Algorithm 6. The basic idea is that we keep a library of different hyperparameter settings. At the beginning, the library is empty. And in each repetition, we sample a number of configurations required by running Hyperband once. During the simulation of Hyperband, we just retrieve the value from the library if the desired epoch of current configuration is contained in the library. Otherwise, we run this configuration based on Hyperband, and store the piece of the trajectory to the library.

**Algorithm 6** Alternative End-to-End Efficiency Evaluation Protocol

```
Input: A set of optimizers $\mathcal{O} = \{o : o = (\mathcal{U}, \Omega)\}$, task $a \in \mathcal{A}$, search space $\mathcal{F}$, library size $S$
1: for $o \in \mathcal{O}$ do
2:     Sample $S$ configurations from $\mathcal{F}$, initialize the library with an empty list for each setting
3:     for $i = 1$ to $M$ do
4:         Simulate Hyperband with $o$ using configurations re-sampled from the library on $a$
5:         if the desired accuracy is pre-computed in the library then
6:             Retrieve the value directly
7:         else
8:             Training normally, and store values to the library
9:         end if
10:        Record the performance trajectory $\{P_t\}_{t=1}^{T}$ explored by HyperBand
11:        Calculate the peak performance and CPE by Eq. 1
12:    end for
13:    Average peak and CPE values over $M$ repetitions for the optimizer $o$
14: end for
15: Evaluate optimizers according to their peak and CPE values
```

## F Additional results

More detailed experimental results are reported in this section.

### F.1 Impact of $\eta$

Since there is an extra hyperparameter, the reduction factor $\eta$ in Hyperband, we conduct an experiment with different values ($\eta = 2, 3, 4, 5$) to observe the potential impact of this additional hyperparameter on our evaluation. Specifically, we use Hyperband to tune the learning rate for three optimizers, SGD, Adam, and Lookahead on CIFAR10, and results are presented in Table 8. As we can see, although the change of $\eta$ may lead to different CPE values, the relative ranking among three optimizers remains unchanged. Besides, they all achieve comparable peak performance at the end of training. Considering the efficiency of Hyperband, we choose $\eta = 3$ based on the convention in Li et al. (2017) in all our experiments.

Table 8: CPE on CIFAR10 with different $\eta$. The value in the round brackets is peak performance.

| **Optimizer** | **CIFAR10 (%) ↑** | | | |
|---|---|---|---|---|
| | $\eta = 2$ | $\eta = 3$ | $\eta = 4$ | $\eta = 5$ |
| *Tune learning rate only:* | | | | |
| SGD | 88.63 (93.42) | 88.87 (93.39) | 88.45 (93.34) | 87.97(93.73) |
| Adam | 90.35 (93.24) | 90.42 (93.65) | 90.06 (93.26) | 88.75 (93.23) |
| Lookahead | 90.46 (93.53) | 90.60 (93.60) | 90.33 (93.40) | 89.05 (93.62) |

## F.2 End-to-end training

Table 9 shows peak performance for optimizers on each task. For GAN, we only conduct evaluation on optimizers tuning learning rate due to time limit, and present its CPE and peak performance in Table 10. There is also an end-to-end training curve for GAN on CIFAR10 in Figure 6. Figures for end-to-end training curves on the rest of tasks are shown in Figure 9 and 10.

Table 9: Peak performance during end-to-end training. The best one for each task is highlighted in bold.

| **Optimizer** | **CIFAR10 (%) ↑** (classification) | **CIFAR100 (%) ↑** (classification) | **CelebA ↓** (VAE) | **MRPC (%) ↑** (NLP) | **PPI ↑** (GCN) | **Walker2d-v3 ↑** (RL) |
|---|---|---|---|---|---|---|
| *Tune learning rate only:* | | | | | | |
| SGD | 93.39 ± 0.12 | **73.68 ± 0.13** | 0.1351 ± 0.0003 | 71.05 ± 0.25 | 94.74 ± $2.7 \times 10^{-2}$ | 3589 ± 221 |
| Adam | 93.65 ± 0.13 | 71.51 ± 0.21 | 0.1326 ± 0.0001 | 84.90 ± 0.10 | **98.73 ± $\mathbf{5.9 \times 10^{-4}}$** | 4735 ± 91 |
| RAdam | **93.93 ± 0.09** | 72.30 ± 0.09 | **0.1325 ± 0.0002** | **85.41 ± 1.45** | 98.70 ± $1.0 \times 10^{-3}$ | 5020 ± 112 |
| Yogi | 93.58 ± 0.03 | 73.48 ± 0.93 | 0.1334 ± 0.0002 | 70.19 ± 0.92 | 98.18 ± $9.0 \times 10^{-4}$ | 5013 ± 439 |
| LARS | 93.46 ± 0.01 | 73.53 ± 0.17 | 0.1332 ± 0.0001 | 68.97 ± 0.85 | 98.45 ± $4.7 \times 10^{-4}$ | 4073 ± 443 |
| LAMB | 93.39 ± 0.03 | 70.38 ± 0.08 | 0.1329 ± 0.0002 | 82.23 ± 0.78 | 98.46 ± $2.2 \times 10^{-4}$ | 4219 ± 191 |
| Lookahead | 93.60 ± 0.04 | 71.75 ± 0.68 | 0.1327 ± 0.0001 | 72.99 ± 0.61 | 98.63 ± $1.4 \times 10^{-3}$ | **5246 ± 666** |
| *Tune every hyperparameter:* | | | | | | |
| SGD | 93.47 ± 0.02 | **73.94 ± 0.06** | 0.1344 ± 0.0003 | 72.80 ± 1.58 | 98.64 ± $1.5 \times 10^{-3}$ | 3647 ± 129 |
| Adam | 92.58 ± 1.63 | 73.82 ± 0.25 | 0.1327 ± 0.0002 | 88.46 ± 0.66 | **98.93 ± $\mathbf{7.5 \times 10^{-4}}$** | **4986 ± 404** |
| RAdam | **94.47 ± 0.24** | 73.50 ± 0.32 | **0.1326** ± 0.0001 | **88.78 ± 0.72** | 98.92 ± $6.0 \times 10^{-4}$ | 4886 ± 334 |
| Yogi | 93.75 ± 0.08 | 73.88 ± 0.25 | 0.1333 ± 0.0004 | 69.60 ± 1.20 | 98.85 ± $4.8 \times 10^{-4}$ | 4612 ± 370 |
| LARS | 94.22 ± 0.10 | 74.08 ± 0.04 | 0.1333 ± 0.0003 | 79.83 ± 6.50 | 98.88 ± $8.4 \times 10^{-4}$ | 3526 ± 312 |
| LAMB | 93.88 ± 0.32 | 71.31 ± 0.08 | 0.1332 ± 0.0002 | 87.80 ± 1.07 | 98.80 ± $1.0 \times 10^{-3}$ | 3654 ± 243 |
| Lookahead | 93.82 ± 0.18 | 72.90 ± 0.85 | 0.1330 ± 0.0005 | 86.15 ± 0.69 | 98.87 ± $1.9 \times 10^{-3}$ | 4614 ± 96 |

Table 10: CPE on GAN for end-to-end training. The value in the bracket is peak performance.

| **Optimizer** | **GAN-CIFAR10↓** |
|---|---|
| *Tune learning rate only:* | |
| SGD | 113.25 (50.08) |
| Adam | 77.04 (25.14) |
| RAdam | 73.85 (19.61) |
| Yogi | 76.80 (25.36) |
| LARS | 157.71 (73.82) |
| LAMB | 68.47 (25.55) |
| Lookahead | **65.61** (**20.40**) |

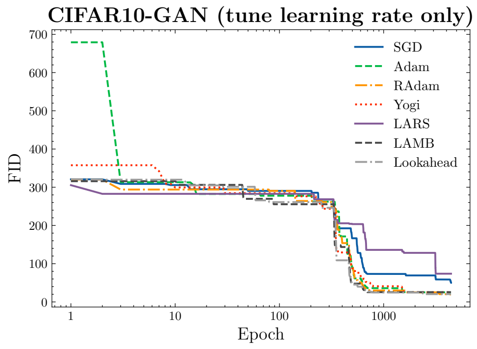
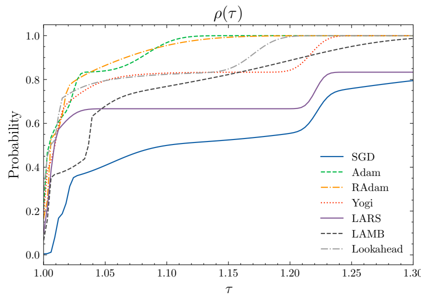


Figure 6: End-to-end training curves for GAN on CIFAR10.

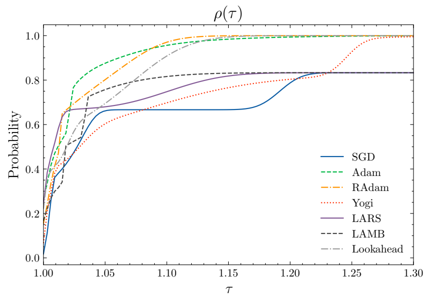
Besides the general performance profile, we present a probabilistic version described in Barreto et al. (2010) in Figure 7 to account for randomness. The probabilistic performance profile can be formulated as

$$\bar{\rho}_o(\tau) = \frac{1}{|\mathcal{A}|}\sum_a \mathcal{P}(\bar{r}_{o,a} \le \tau), \quad \bar{r}_{o,a} \sim \mathcal{N}(\frac{\mu_{o,a}}{b_a}, \frac{\sigma_{o,a}^2}{b_a^2}), \tag{3}$$

where $\mu_{o,a}$ and $\sigma_{o,a}$ are the mean and standard deviation of CPE of the optimizer $o$ on the task $a$ respetively, and $b_a$ is just the best expectation of CPE on $a$ among all optimizers. It can be seen in Figure 7 that the probabilistic performance profiles shows a similar trend to Figure 4.



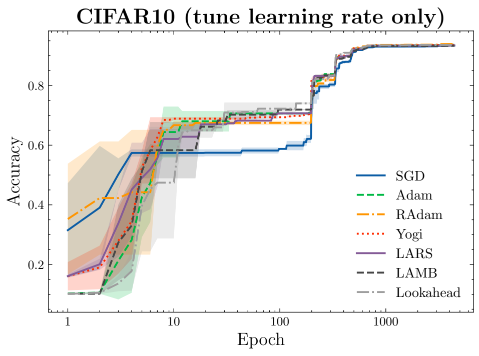
(a) Tune learning rate only.



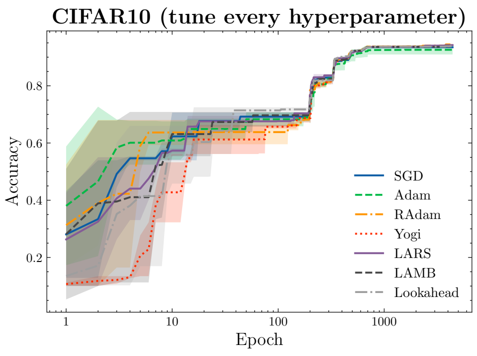
(b) Tune every hyperparameter.

Figure 7: Probabilistic performance profile of 7 optimizers in the range $[1.0, 1.3]$.

We also attach two end-to-end training trajectories on CIFAR10 with the error in Figure 8. Since it is hard to distinguish optimizers with the standard deviation added, we just instead report the std of CPE and peak performance in Table 3 and 9.

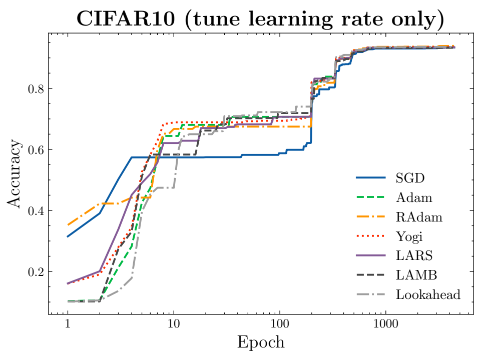


(a)

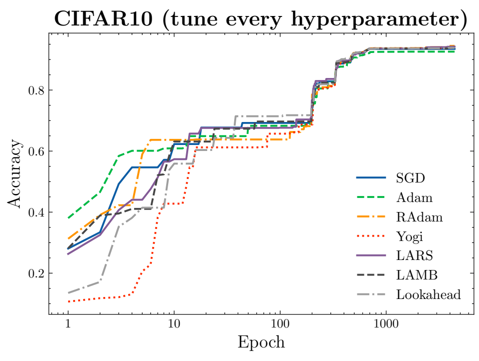


(b)

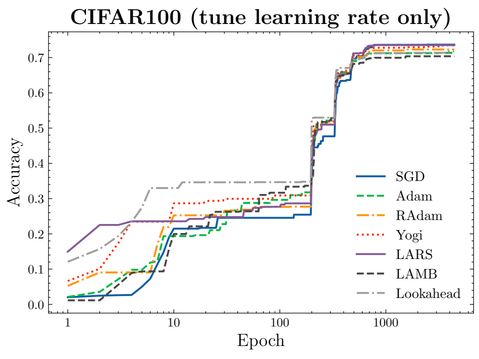
Figure 8: End-to-end training curves on CIFAR10



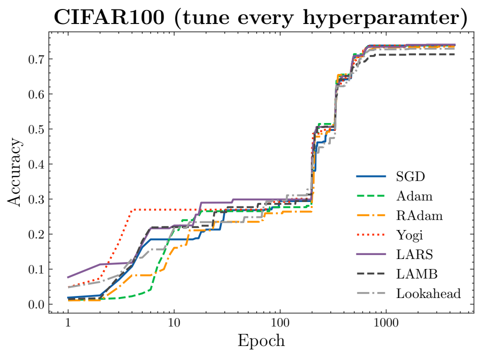
(a)



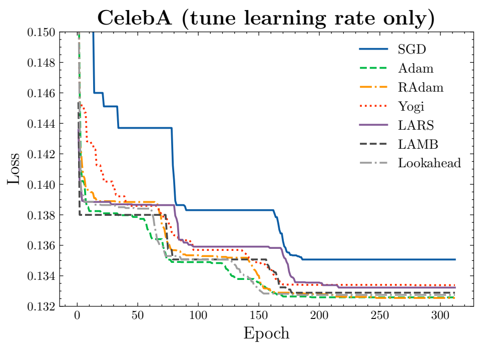
(b)



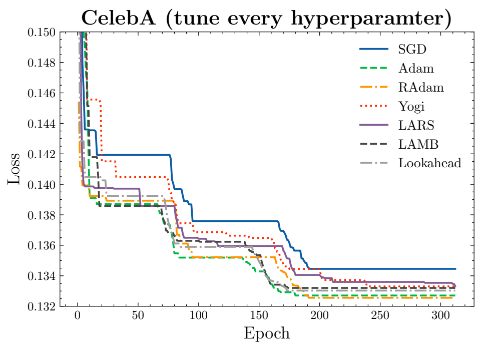
(c)



(d)



(e)

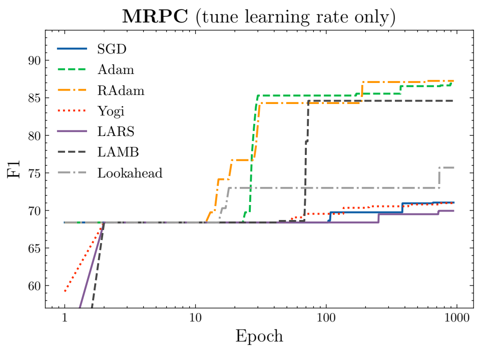


(f)

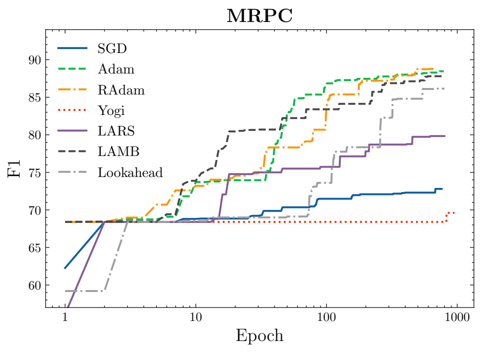
Figure 9: End-to-end training curves on CIFAR10, CIFAR100, and CelebA.

### F.3 Data addition training

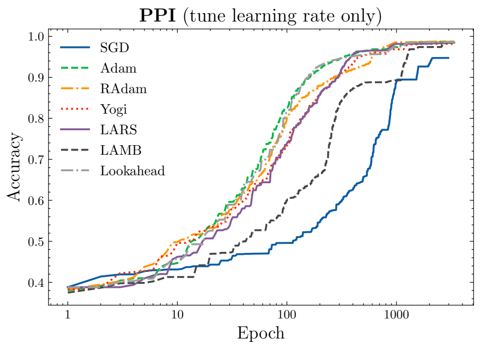
We provide training curves for data-addition training on full MRPC and PPI dataset in Figure 11.



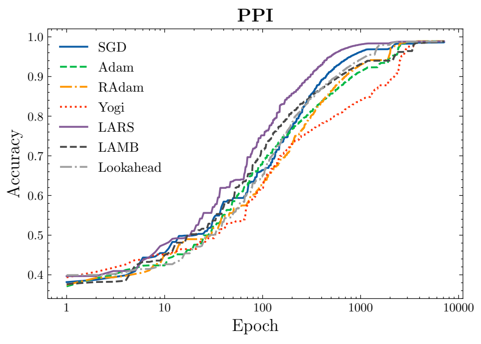
(a)



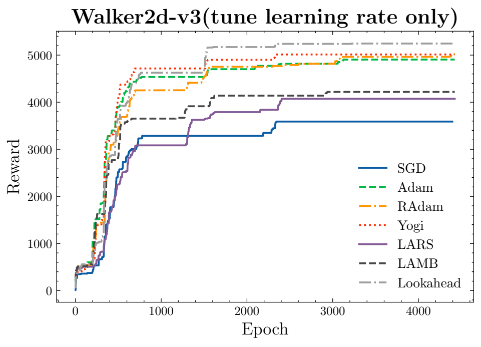
(b)



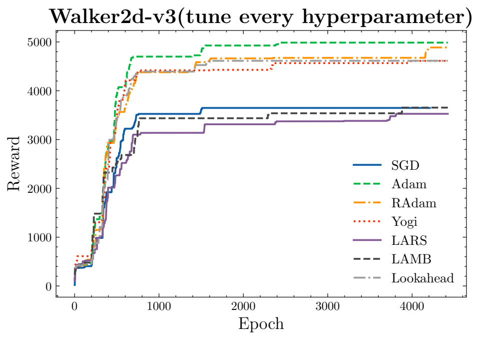
(c)



(d)

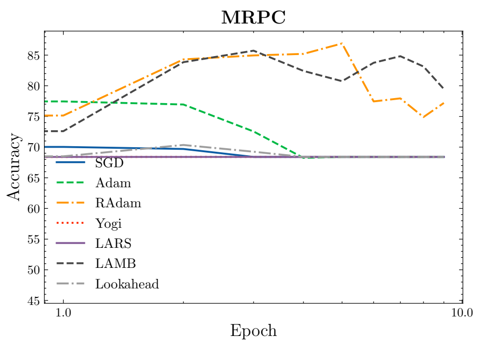


(e)

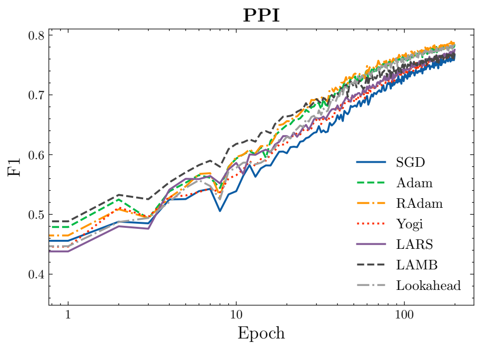


(f)

Figure 10: End-to-end training curves on MRPC, PPI, and Walker2d-v3.



(a) Training on full MRPC.



(b) Training on full PPI.

Figure 11: Data addition trainong on MRPC and PPI.