Appendix for Rethinking the Role of Hyperparameter Tuning in Optimizer Benchmarking

Yuanhao Xiong[1], **Xuanqing Liu**[1], **Li-Cheng Lan**[1], **Yang You**[2], **Si Si**[3], **Cho-Jui Hsieh**[1]
[1]Department of Computer Science, UCLA
[2]Department of Computer Science, National University of Singapore
[3]Google Research
{yhxiong,xqliu,lclan}@cs.ucla.edu, youy@comp.nus.edu.sg
sisidaisy@google.com, chohsieh@cs.ucla.edu

A HyperBand

We present the whole algorithm for Hyperband in Algorithm 1, and you can refer to Li et al. [7] for more details.

B 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, μ is the , while β_1 and β_2 are coefficients to compute the running averages m and v. ϵ is a small scalar (e.g., 1×10^{-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 2. 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 [10] in the following equation, introducing γ 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 1. 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. [11], this optimizer is robust to inner optimization algorithm, k, and α_s in Algorithm 3, 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.

C Task description

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

Submitted to the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks. Do not distribute.

Algorithm 1 Hyperband

Input: R, η
Initialization: $s_{\max} = \lfloor \log_\eta R \rfloor$, $B = (s_{\max} + 1)R$

for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SuccessiveHalving with (n, r) inner loop
  $T =$ random_get_configuration(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$, $r_i = r\eta^i$
    $L = \{$run_then_return_val_loss(t, r_i): $t \in T\}$
    $T =$ top_k$(T, L, \lfloor n_i/\eta \rfloor)$
  end for
end for

Return Hyperparameter configuration with the smallest loss seen so far

Algorithm 2 Generic framework of optimization methods

Input: parameter value θ_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.

for $t = 1$ **to** T **do**
  $g_t = \nabla f_t(\theta_t)$
  $m_t = \phi_t(g_1, \cdots, g_t)$
  $v_t = \psi_t(g_1, \cdots, g_t)$
  $r_t = \chi_t(\theta_t, m_t, v_t)$
  $\theta_{t+1} = \theta_t - \alpha_t r_t m_t / \sqrt{v_t}$
end for

- Image classifcation. For this task, we adopt a ResNet-50 [5] 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 and Welling [6] 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. [8], 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 [4] as the backbone to handle neighborhood expansion and PPI as the dataset.
- Reinforcement learning. We select Walker2d-v3 from OpenAI Gym ([3]) as our training environment, and PPO ([9]), implemented by OpenAI SpinningUp ([1]), as the algorithm that required tuning. We use the same architectures for both action value network Q and the policy network π. 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.

D 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 2. Note that we tune the learning rate decay factor for image classification tasks when tuning every hyperparamter. For the task on MRPC, γ is tuned for all experiments. In other cases, we only tune original hyperparamters without a learning rate scheduler.

Table 1: 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 3 Lookahead Optimizer

Input: Initial parameters θ_0, objective function f, synchronization period k, slow weights step size α_s, optimizer A

for $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\hat{\theta}_{t,0} \leftarrow \theta_{t-1}$
  for $i = 1, 2, \ldots, k$ **do**
    Sample minibatch of data $d \in \mathcal{D}$
    $\hat{\theta}_{t,i} \leftarrow \hat{\theta}_{t,i-1} + A(L, \hat{\theta}_{t,i-1}, d)$
  end for
  Perform outer update $\theta_t \leftarrow \theta_{t-1} + \alpha_s(\hat{\theta}_{t,k} - \theta_{t-1})$
end for

Return Parameters θ

In addition, Hyperband parameter values for each task are listed in Table 3. These parameters are assigned based on properties of different tasks.

Table 2: Hyperparamter search space and default value

Hyperparamter	**Search space**	**Default value**
α_0	Log-Uniform$(-8, 1)$	$\times$
μ	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
ϵ	Log-Uniform$(-8, 1)$	1e-8
γ	Log-Uniform$(-4, 0)$	$\times$

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.

Moreover, we can further accelerate our evaluation protocol by resampling, shown in Algorithm 4. 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.

E Additional results

More detailed experimental results are reported in this section.

Table 3: Hyperband parameters for each task.

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

Algorithm 4 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

for $o \in \mathcal{O}$ **do**
  Sample S configurations from $\mathcal{F}$, initialize the library with an empty list for each setting
  for $i = 1$ **to** M **do**
    Simulate Hyperband with o using configurations re-sampled from the library on a
    if the desired accuracy is pre-computed in the library **then**
      Retrieve the value directly
    else
      Training normally, and store values to the library
    end if
    Record the performance trajectory $\{P_t\}_{t=1}^{T}$ explored by HyperBand
    Calculate the peak performance and *CPE* by Eq. 1
  end for
  Average peak and *CPE* values over M repetitions for the optimizer o
end for
Evaluate optimizers according to their peak and *CPE* values

E.1 Impact of η

Since there is an extra hyperparameter, the reduction factor η 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 4. As we can see, although the change of η 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. [7] in all our experiments.

Table 4: CPE on CIFAR10 with different η. 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)

E.2 End-to-end training

Table 5 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 6. There is also an end-to-end training curve for GAN on CIFAR10 in Figure 1. Figures for end-to-end training curves on the rest of tasks are shown in Figure 4 and 5.

Table 5: 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	$94.74 \pm 2.7 \times 10^{-2}$	3589 ± 221
Adam	93.65 ± 0.13	71.51 ± 0.21	0.1326 ± 0.0001	84.90 ± 0.10	$\mathbf{98.73 \pm 5.9 \times 10^{-4}}$	4735 ± 91
RAdam	**93.93 ± 0.09**	72.30 ± 0.09	**0.1325 ± 0.0002**	**85.41 ± 1.45**	$98.70 \pm 1.0 \times 10^{-3}$	5020 ± 112
Yogi	93.58 ± 0.03	73.48 ± 0.93	0.1334 ± 0.0002	70.19 ± 0.92	$98.18 \pm 9.0 \times 10^{-4}$	5013 ± 439
LARS	93.46 ± 0.01	73.53 ± 0.17	0.1332 ± 0.0001	68.97 ± 0.85	$98.45 \pm 4.7 \times 10^{-4}$	4073 ± 443
LAMB	93.39 ± 0.03	70.38 ± 0.08	0.1329 ± 0.0002	82.23 ± 0.78	$98.46 \pm 2.2 \times 10^{-4}$	4219 ± 191
Lookahead	93.60 ± 0.04	71.75 ± 0.68	0.1327 ± 0.0001	72.99 ± 0.61	$98.63 \pm 1.4 \times 10^{-3}$	**5246 ± 666**
Tune every hyperparameter:						
SGD	93.47 ± 0.02	**73.94 ± 0.06**	0.1344 ± 0.0003	72.80 ± 1.58	$98.64 \pm 1.5 \times 10^{-3}$	3647 ± 129
Adam	92.58 ± 1.63	73.82 ± 0.25	0.1327 ± 0.0002	88.46 ± 0.66	$\mathbf{98.93 \pm 7.5 \times 10^{-4}}$	**4986 ± 404**
RAdam	**94.47 ± 0.24**	73.50 ± 0.32	**0.1326** ± 0.0001	**88.78 ± 0.72**	$98.92 \pm 6.0 \times 10^{-4}$	4886 ± 334
Yogi	93.75 ± 0.08	73.88 ± 0.25	0.1333 ± 0.0004	69.60 ± 1.20	$98.85 \pm 4.8 \times 10^{-4}$	4612 ± 370
LARS	94.22 ± 0.10	74.08 ± 0.04	0.1333 ± 0.0003	79.83 ± 6.50	$98.88 \pm 8.4 \times 10^{-4}$	3526 ± 312
LAMB	93.88 ± 0.32	71.31 ± 0.08	0.1332 ± 0.0002	87.80 ± 1.07	$98.80 \pm 1.0 \times 10^{-3}$	3654 ± 243
Lookahead	93.82 ± 0.18	72.90 ± 0.85	0.1330 ± 0.0005	86.15 ± 0.69	$98.87 \pm 1.9 \times 10^{-3}$	4614 ± 96

Table 6: 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**)

Besides the general performance profile, we present a probabilistic version described in Barreto et al. [2] in Figure 2 to account for randomness. The probabilistic performance profile can be formulated as

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

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 2 that the probabilistic performance profiles shows a similar trend to Figure 4 in the main paper.

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

E.3 Data addition training

We provide training curves for data-addition training on full MRPC and PPI dataset in Figure 6.

E.4 Training time comparison

In this part, we report training time for each optimizer of running one epoch on CIFAR100 in Table 7. As we can see, SGD runs faster than all other advanced optimizers. Besides, we plot the performance of different optimizers along with the training time in Figure 7. Here the unit is the running time for one epoch of SGD. The relative ranking of selected optimizers is consistent with that under the budget of training epochs.

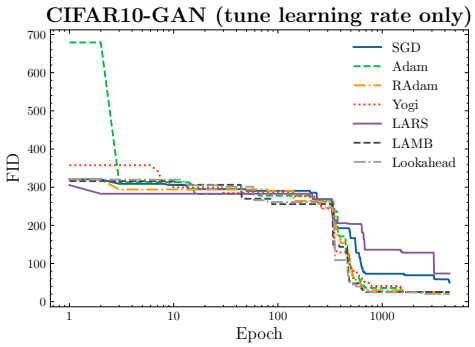
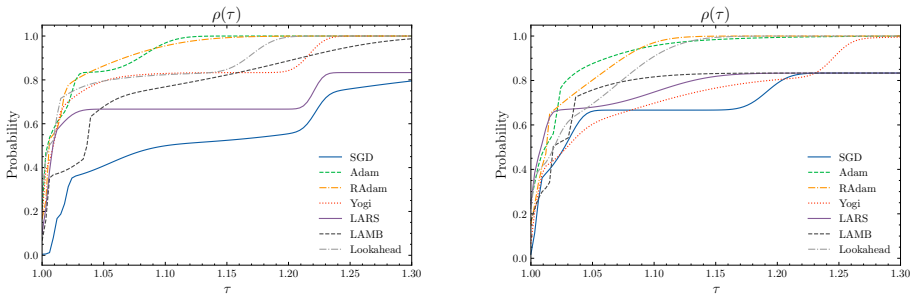
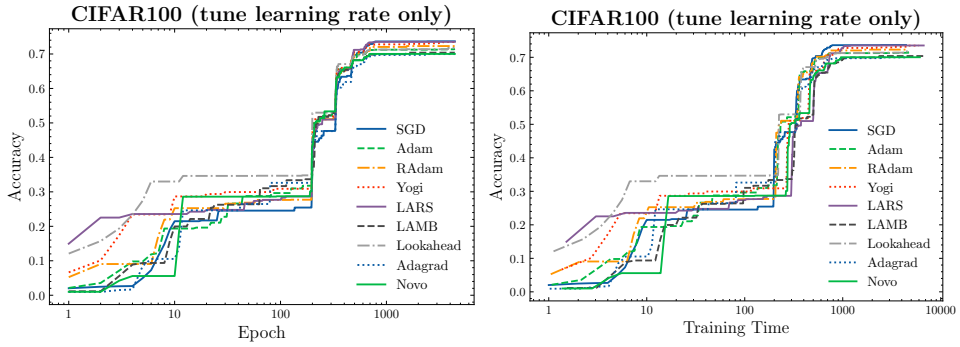


Figure 1: End-to-end training curves for GAN on CIFAR10.



(a) Tune learning rate only. (b) Tune every hyperparameter.

Figure 2: Probabilistic performance profile of 7 optimizers in the range $[1.0, 1.3]$.



(a) (b)

Figure 7: End-to-end training curves on CIFAR100 of different time budget units: epoch and training time.

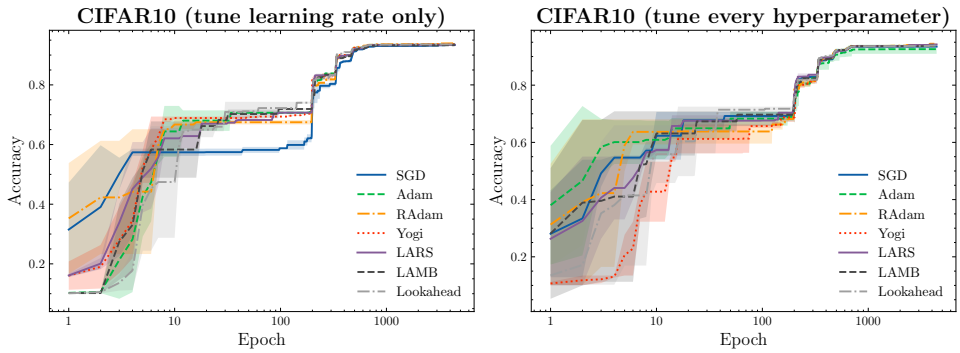
References

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning, 2018.

[2] 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*, pages 751–758, 2010.

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

[4] 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*



(a) (b)

Figure 3: End-to-end training curves on CIFAR10

Table 7: Training time of optimizers for running one epoch on CIFAR100 (unit: second).

Optimzer	SGD	Adam	RAdam	Yogi	LARS	LAMB	Lookahead	Adagrad	NovoGrad
Training Time	**43.19**	58.43	46.29	66.00	65.47	48.32	45.26	44.18	59.85

of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 257–266, 2019.

[5] 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*, pages 770–778, 2016.

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

[7] 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.

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

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

[10] 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.

[11] 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*, pages 9597–9608, 2019.

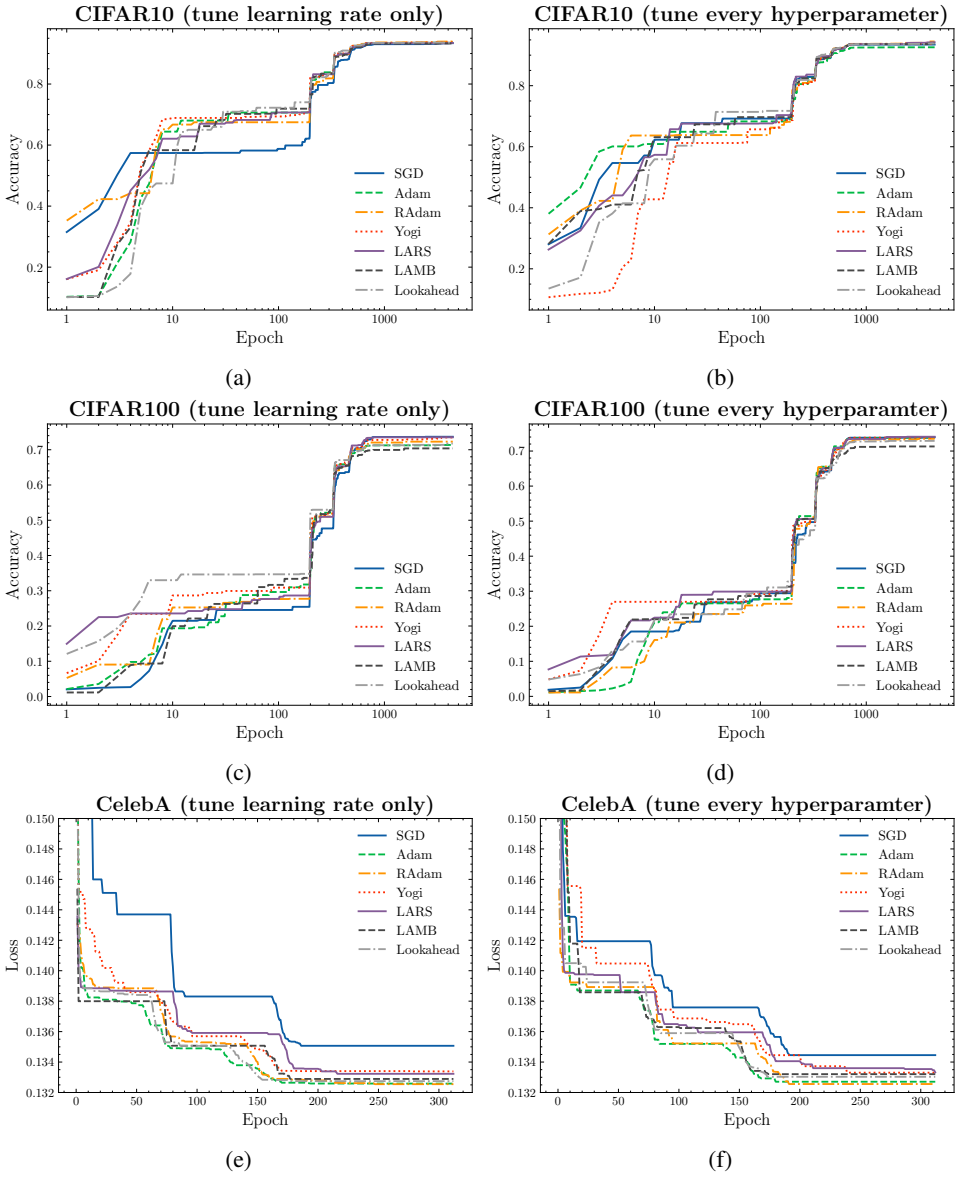


Figure 4: End-to-end training curves on CIFAR10, CIFAR100, and CelebA.

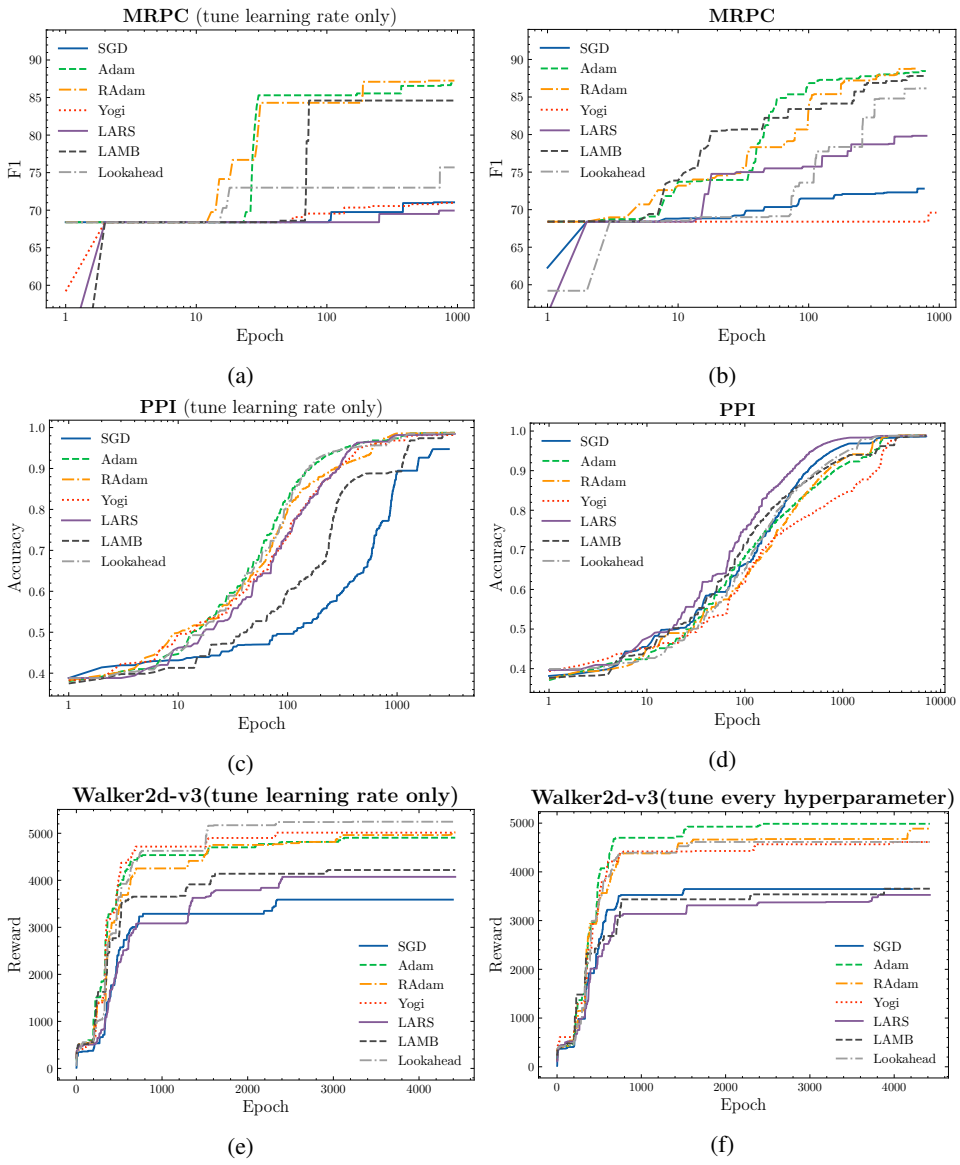
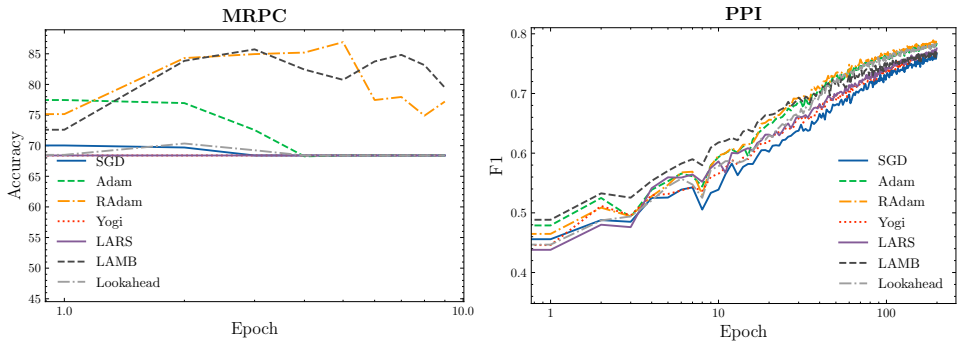


Figure 5: End-to-end training curves on MRPC, PPI, and Walker2d-v3.



(a) Training on full MRPC.

(b) Training on full PPI.

Figure 6: Data addition trainong on MRPC and PPI.