A Algorithm for Few-Shot Learning

Algorithm 1 Meta-LSTM for few-shot learning with smoothness-inducing regularization

Input: the set of meta-sets $\mathcal{D}_{\text{meta}}$, step sizes η_1 and η_2, number of inner iterations K, the number of total steps T_{total}, classifier parameterized by θ, optimizer parameterized by ϕ
repeat
  Initialize θ randomly, reset LSTM hidden state
  Sample a dataset $D = \{D_{\text{train}}, D_{\text{test}}\}$ from $\mathcal{D}_{\text{meta}}$
  $\mathcal{L} \leftarrow 0$
  for $t = 0, \ldots, T_{\text{total}} - 1$ **do**
    Feed a batch of data (x, y) from D_{train} to the classifier, obtain state s_t
    Update θ as demonstrated in Section 3.1
    $s'_t \leftarrow s_t + 0.05 * \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    for $k = 1, \ldots, K$ **do** ▷ Find the perturbed state iteratively
      $s'_t \leftarrow \Pi_{\mathbb{B}(s_t, \epsilon)}(\eta_1 \operatorname{sign}(\nabla_{s'_t} d(u(s_t), u(s'_t))) + s'_t)$
    end for
    $R_{t+1} \leftarrow \|u(s_t) - u(s'_t)\|^2$ ▷ Regularization term
    $\mathcal{L} \leftarrow \mathcal{L} + \lambda R_{t+1}$
  end for
  Sample a batch of data (x, y) from D_{test}
  $\mathcal{L} \leftarrow \mathcal{L} + \ell\left(f(\theta_{T_{\text{total}}}(\phi); x), y\right)$
  Update ϕ by $\mathcal{L}$ using Adam with the step size η_2
until converged

B Experiment Summary

We provide a summary of experiments for image classification in Table x.

Table 1. Test accuracy of different regularizers.

Dataset	Descritption	Result
MNIST	Training LeNet with different initializations	Figure 3b, 3e
	Generalization to longer steps	Figure 3c, 3f
CIFAR10	Training a 3-layer CNN for 10K steps	Figure 4
	Transferred training of ResNet-18	Figure 5a, 5d
	Training on unseen data	Figure 5b, 5e
	Transferred training of ResNet-18 on CIFAR100	Figure 5c, 5f

C Implementation Details

Extra time cost caused by our regularization term has little impact on the efficiency of applying the neural optimizer. As the dimension of the state is relatively small, the gradient can be obtained efficiently considering the computational complexity of backpropagation. Specifically, training time per epoch for SimpleOptimizer is 38.21s while for our method is 97.31s. As we only train the learned optimizer for a few epochs and then apply it to the target task, we care more about time for deploying the neural optimizer. This time is nearly the same as the hand-designed ones. Moreover, since PGD has been acknowledged as a practical and effective method in adversarial attacks, it is feasible to obtain s' in our case.

For DMOptimizer [1], we adopt a 2-layer LSTM with the hidden size of 10, while for SimpleOptimizer [2], we just use a 1-layer RNN with the hidden size of 10. As to few-shot learning, Meta-LSTM leverages its specific 2-layer LSTM structure described in [7].

As mentioned, we use grid search to determine the learning rate for each hand-designed optimizer. Specifically, for all hand-engineered optimizers including SGD, SGDM, Adam, AMSGrad, and RMSProp, we only tune one hyperparameter, learning rate, and leave the rest with their default values. We adopt a simple yet effective grid search to find the best learning rate for these optimizers, and the search space is $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$, which is a frequently used range in many papers related to optimization algorithms [3]. For the SimpleOptimizer, we use recommended hyperparameters, optimizer structures, and state definitions in [1] and [2] respectively. Finally, we set the learning rate to 0.1 for SGD and SGDM, 10^{-3} for Adam, AMSGrad and RMSProp. As to our smoothed optimizer, with all other settings the same as SimpleOptimizer, we tune two additional hyperparameters ϵ and λ, where HPO is also conducted by grid search with $\epsilon \in [10^{-2}, 10^{-1}, 1]$ and $\lambda \in [10^{-1}, 1, 10, 10^2]$. We tune ϵ and λ for each task. During evaluation, we train the optimizee by a specific optimizer without early-stopping.

Furthermore, these neural optimizers have their different state descriptions. The input state is the concatenation of preprocessed gradient and the original parameter value in DMOptimizer [1]. We only have the gradient normalized by the second momentum as the state in SimpleOptimizer [2]. The state of Meta-LSTM [7] is based on DMOptimizer with an extra feature, the original gradient.

On the other hand, for fairness of the comparison, it is helpful to adopt a more performant HPO method. Therefore, we conduct Hyperband [4] to search for the best learning rate for our five hand-designed optimizers. We choose hyperparameters for Hyperband with $\eta = 3, R = 10000$. Results are presented in Figure 4 in the link. Marginal improvement can be observed over grid search but our smoothed optimizer still outperforms all the baselines when training a 3-layer CNN on CIFAR10. In fact, image classification tasks on CIFAR10 have been explored extensively, and a grid search within $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$ is sufficient to lead to a comparable performance with other effective HPO al-

gorithms such as Hyperband. Hence, it is efficient to use grid search, which guarantees a fair comparison at the same time.

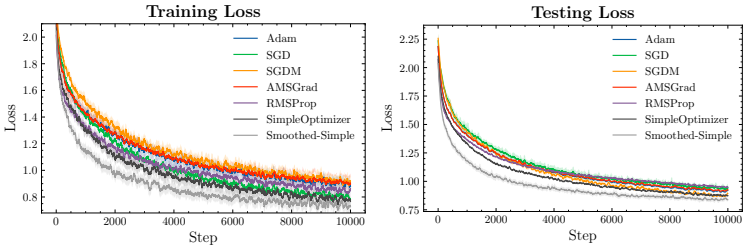


Fig. 1. Learning curves of different optimizers with Hyperband as HPO algorithm.

D Additional Experimental Results

D.1 Accuracy with a longer horizon

Since we try to point out and solve current problems in existing learned optimizers, and do not aim at outperforming the state-of-the-art results, we choose the number of iterations as 1000 for MNIST and 10000 for CIFAR10, based on settings of previous neural optimizers [1,5,6]. To support the effectiveness of our smoothed optimizer, we showed the results of MNIST for a longer horizon of 10000 steps in Figure 2. It can be observed that our proposed method is still the best with fastest convergence rate and best testing accuracy. In addition, all baselines achieves satisfactory accuracies, which are comparable to existing ones in LeNet. Specifically, among those baselines, SGD obtains the best test accuracy of 98.93% while our smoothed optimizer reaches 99.16%. This experiment further validates advantages of the smoothness-inducing neural optimizer.

D.2 Smoothness Comparison

In this section, we conduct an experiment on MNIST to demonstrate the smoothness of the neural optimizer trained with our proposed regularization. Specifically, we perturb the state which is the optimizer input with the noise $\delta \sim \mathcal{N}(0, 0.1)$ for each time step and observe performance changes in terms of learning curves. We plot differences in loss between the optimizer with normal states and with perturbed states in Figure 3. Note that the difference is calculated by

$$\text{diff} = \text{loss}_{ps} - \text{loss}_{ns}$$

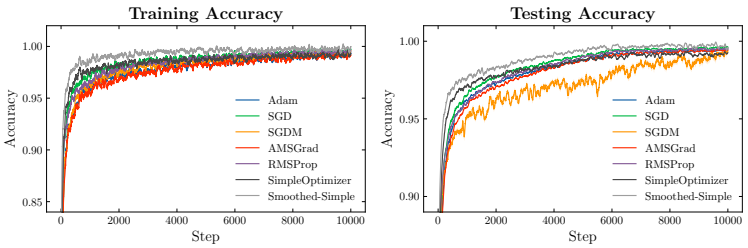
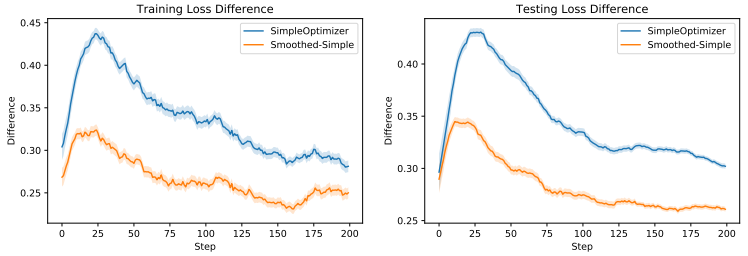


Fig. 2. Accuracy curves on MNIST with LeNet for 10000 steps.



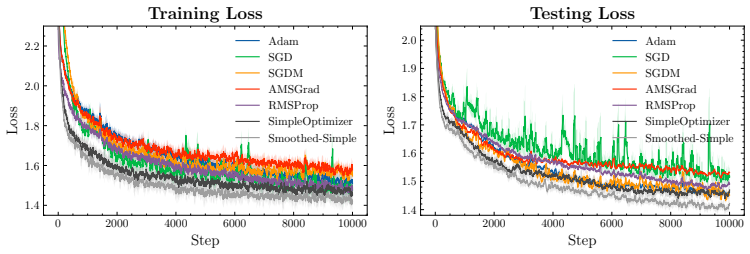
(a) Training **(b)** Testing

Fig. 3. Loss difference of SimpleOptimizer and Smoothed-Simple.

where *ps* means perturbed states and *ns* means normal states. As we can see, performances of both SimpleOptimizer and SmoothedSimple decline while on the other hand, the drop is relatively smaller of our smoothed optimizer than that of the original one. This phenomenon shows that our method can boost the smoothness of neural optimizers confronted with state disturbance.

D.3 Results on tiny-ImageNet

We evaluate the optimizers on tiny-ImageNet, which is a relatively larger dataset extracted from ILSVRC-12 [8]. 100 classes are selected from this dataset, among which the training set, validation set, and testing set consist of 70, 20, and 10 classes respectively. We implement an experiment targeted at a 10-class classification problem, in which the optimizer is trained on the training set, selected on the validation set, and evaluated on the testing set. To avoid overfitting, we adopt a L_1 regularization when training classifiers. As can be observed in Figure 4, the learned optimizer with a smooth regularizer performs consistently better than hand-designed methods. Specifically, it learns about 2 times faster than Adam, and converges to a lower loss value in both training and testing stage. Furthermore, our method improves the non-smooth SimpleOptimizer noticeably as well.

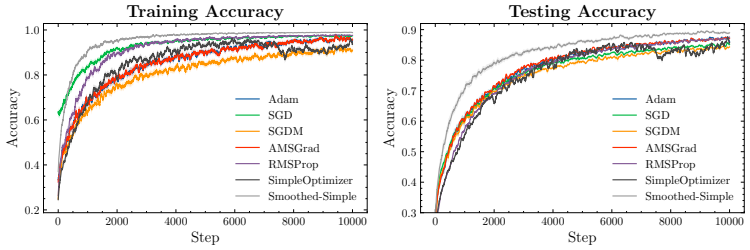


(a) Training loss **(b)** Test loss

Fig. 4. Learning curves of classification on tiny-ImageNet.

D.4 Accuracy Curves

In this section, we provide more trajectories of accuracy.



(a) **(b)**

Fig. 5. Accuracy curves of different optimizers on CIFAR10, ResNet-18.

D.5 Results on RNN

To our knowledge, existing methods of learning to learn [1,6] still focus on computer vision and there is little discussion about applying learned optimizers to language tasks. However, we tried to leverage the neural optimizer to train a simple RNN for binary sentiment analysis on the SST dataset. Specifically, the network under investigation is a bi-directional LSTM with the hidden size of 100. We train the neural optimizer for 200 steps as well on this task and evaluate it for 2000 training steps. Currently we have some preliminary results in the following table as well as in Figure 7. As we can see, there exists a gap between neural optimizers and conventional ones but our proposed regularization still works to improve the vanilla simple optimizer. For more complex architectures like Transformer, the current design of the neural optimizer like BackproPagation Through Time (BPTT) is incapable to handle such a large model due to

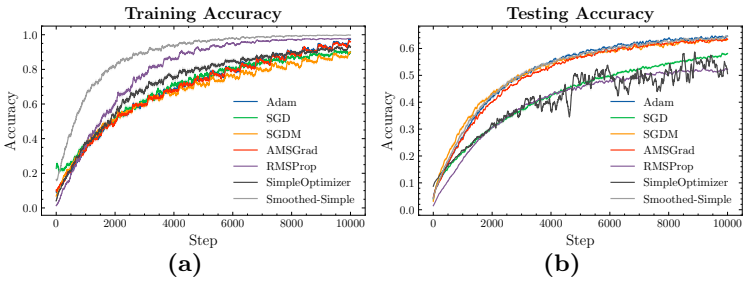
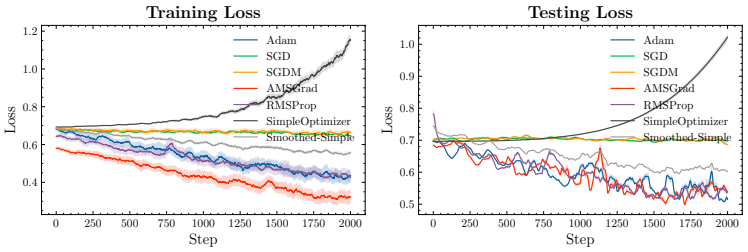


Fig. 6. Accuracy curves of different optimizers on CIFAR100, ResNet-18.

memory constraints. Overall, it remains a research challenge to extend the neural optimizer to other domains besides computer vision and this is an interesting future direction.



(a) Training loss **(b)** Test loss

Fig. 7. Learning curves of sentiment analysis on SST.

References

1. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in neural information processing systems. pp. 3981–3989 (2016)
2. Chen, P.H., Reddi, S., Kumar, S., Hsieh, C.J.: Learning to learn with better convergence (2020), https://openreview.net/forum?id=S1xGCAVKvr
3. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
4. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research **18**(1), 6765–6816 (2017)
5. Lv, K., Jiang, S., Li, J.: Learning gradient descent: Better generalization and longer horizons. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2247–2255. JMLR. org (2017)

Optimizer	Test Accuracy
Adam	76.85 ± 1.23%
SGD	54.23 ± 1.58%
SGDM	55.65 ± 2.32%
AMSGrad	74.33 ± 0.86%
RMSProp	75.79 ± 1.04%
SimpleOptimizer	50.04 ± 0.05%
Smoothed-Simple	63.57 ± 1.43%

Table 2. Test accuracy of sentiment analysis.

6. Metz, L., Maheswaranathan, N., Nixon, J., Freeman, C.D., Sohl-Dickstein, J.: Understanding and correcting pathologies in the training of learned optimizers. arXiv preprint arXiv:1810.10180 (2018)
7. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR (2017)
8. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)