Learning to Learn with Smooth Regularization

Yuanhao Xiong and Cho-Jui Hsieh

University of California, Los Angeles, CA 90095, USA
{yhxiong, chohsieh}@cs.ucla.edu

Abstract. Recent decades have witnessed great advances of deep learning in tackling various problems such as classification and decision making. The rapid development gave rise to a novel framework, Learning-to-Learn (L2L), in which an automatic optimization algorithm (optimizer) modeled by neural networks is expected to learn rules for updating the target objective function (optimizee). Despite its advantages for specific problems, L2L still cannot replace classic methods due to its instability. Unlike hand-engineered algorithms, neural optimizers may suffer from the instability issue—under distinct but similar states, the same neural optimizer can produce quite different updates. Motivated by the stability property that should be satisfied by an ideal optimizer, we propose a regularization term that can enforce the smoothness and stability of the learned optimizers. Comprehensive experiments on the neural network training tasks demonstrate that the proposed regularization consistently improve the learned neural optimizers even when transferring to tasks with different architectures and datasets. Furthermore, we show that our smoothness-inducing regularizer can improve the performance of neural optimizers on few-shot learning tasks. Code can be found at https://github.com/xyh97/SmoothedOptimizer.

Keywords: Learning to learn, smoothness-inducing regularizer

1 Introduction

Optimization is always regarded as one of the most important foundations for deep learning, and its development has pushed forward tremendous breakthroughs in various domains including computer vision and natural language processing [3,16]. Effective algorithms such as SGD [22], Adam [12] and AdaBound [15] have been proposed to work well on a variety of tasks. In parallel to this line of hand-designed methods, Learning-to-Learn (L2L) [1,28,18,16,3], a novel framework aimed at an automatic optimization algorithm (optimizer), provides a new direction to performance improvement in updating a target function (optimizee). Typically, the optimizer, modeled as a neural network, takes as input a certain state representation of the optimizee and outputs corresponding updates for parameters. Then such a neural optimizer can be trained like any other network based on specific objective functions.

Empirical results have demonstrated that these learned optimizers can perform better optimization in terms of the final loss and convergence rate than general hand-engineered ones [1,28,18,16,3]. In addition, such advantages in faster

training make the learned optimizer a great fit for few-shot learning (FSL) [20,11], where only a limited number of labelled examples per class are available for generalizing a classifier to a new task.

However, instability concealed behind the algorithm impedes its development significantly. There are some unsolved issues challenging the promotion of neural optimizers such as gradient explosion in unrolled optimization [18] and short-horizon bias [29]. One of the most essential problems is that contrary to traditional optimizers, the learned ones modeled as neural networks cannot guarantee smoothness with respect to input data. Specifically, an ideal optimizer is expected to conduct similar updates given similar states of the target optimizee. For instance, SGD updates a parameter by a magnitude proportional to its original gradient. However, current meta learners neglect this property and suffer from the issue that they would produce a quite different output while merely adding a small perturbation to the input state.

Such a phenomenon has been widely observed in other machine learning problems like image classification [7], where the perturbed image can fool the classifier to make a wrong prediction. Inspired by the progress in adversarial training [17] where the worst-case loss is minimized, we propose an algorithm that takes the smoothness of the learned optimizer into account. Through penalizing the non-smoothness by a regularization term, the neural optimizer is trained to capture a smooth update rule with better performance.

In summary, we are the first to consider the smoothness of neural optimizers, and our main contributions include:

- A smoothness-inducing regularizer is proposed to improve the existing training of learned optimizers. This term, representing the maximal distance of updates from the current state to the other in the neighborhood, is minimized to narrow the output gap for similar states.
- We evaluate our proposed regularization term on various classification problems using neural networks and the learned optimizer outperforms hand-engineered methods even if transferring to tasks with different architectures and datasets.
- In addition to generic neural network training, we also conduct experiments on few-shot learning based on a Meta-LSTM optimizer [20] and SIB [11]. Results show that our regularizer consistently improves the accuracy on FSL benchmark datasets for 5-way few shot learning problems.

2 Related Work

Gradient-based optimization has drawn extensive attention due to its significance to deep learning. There are various algorithms that have been proposed to improve training of deep neural networks, including SGD [22], Adam [12], RMSProp [10], and the like. On the other hand, a profound thought of updating the optimizee automatically rather than using hand-engineered algorithms has broken the routine and shown great potentials in improving performance for specific problems. Early attempts can be dated back to 1990s when [5] leveraged

recurrent neural networks to model adaptive optimization algorithms. The idea was further developed in [31] where neural optimizers were trained to tackle fundamental convex optimization problems. Recently in the era of deep learning, a seminal work of [1] designed a learning-to-learn framework with an LSTM optimizer, which obtained better performance than some traditional optimizers for training neural networks. Follow-up work in [16] and [28] have improved the generalization and scalability of learned optimizers. L2L framework is easier to train and can adaptively determine the step size and update direction in the meanwhile. L2L has also been extended to various applications such as few-shot learning [20], zeroth-order optimization [23] and adversarial training [30].

This paper is the first to investigate the smoothness of neural optimizers. It is related to the notion of adversarial robustness in classification models. As observed in [7], neural network based models are vulnerable to malicious perturbations. In particular, for image classification the classifier would be fooled by adversarial examples to make a wrong prediction [7], while for reinforcement learning the agent is likely to act differently under perturbed states [25]. Our learned optimizers might be affected by this issue as well. In other domains some algorithms have been proposed to mitigate the non-smooth property of neural networks such as adversarial training [17], and SR^{2}L [25]. In this paper, our method utilizes the idea of minimizing the worst-case loss to regularize training of neural optimizers towards smoothness. In contrast to previous algorithms targeted at classification, we design a specific regularizer to neural optimizers.

3 Background on the L2L Framework

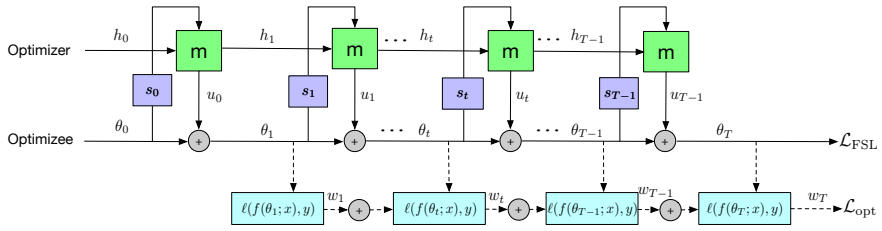

Fig. 1. The framework of learning-to-learn. The dashed line shows the computation graph of the objective function $\mathcal{L}_{\text{opt}}$ for training the optimizer to learn a general update rule while the horizontal full line is the one for few-shot learning. Note that m is the neural optimizer parameterized by ϕ, and s_t is the state of the optimizee taking the form of $s_t = (\theta_t, \ldots, \nabla_\theta \ell)^T$.

In this section, we present the framework of learning to learn with neural optimizers for tackling problems of general optimization for classification and few-shot learning.

3.1 Optimization

As shown in Figure 1, like any traditional optimization methods, we can apply the learned optimizer in following steps:

(a) At each time step t, feed a batch of training examples $\{(x, y)\}$ from the distribution $\mathcal{D}$ into the target classifier f parameterized by θ, and the state of the optimizee s_t can be described by several values. In the paper, we use $s_t = (\theta_t, \nabla_{\theta_t}\ell, \mu_t, \nu_t)$. μ_t and ν_t are first and second moment respectively.

(b) Given the current state s_t and the hidden state h_t, the neural optimizer m parameterized by ϕ accordingly outputs the increment of the parameter and the next hidden state by $u_t, h_{t+1} = m(s_t, h_t)$.

(c) Then the optimizer just updates the parameter by $\theta_{t+1} = \theta_t + u_t$.

Note that all operations are coordinate-wise, which means the parameters of the optimizee are updated by a shared neural optimizer independently and maintain their individual hidden states.

The exploitation of the learned optimizer is straightforward but how can we train it? Following [1], since parameters of the optimizee depend implicitly on the optimizer, which can be written as $\theta_t(\phi)$, the quality of the optimizer can be reflected by performance of the optimizee for some horizon T, leading to the objective function below to evaluate the optimizer:

$$\mathcal{L}_{\text{opt}}(\phi) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sum_{t=1}^{T} w_t \ell(f(\theta_t(\phi); x), y)\right]. \tag{1}$$

$\ell(\cdot, \cdot)$ represents cross-entropy and w_t is the weight assigned for each time step.

3.2 Few-shot Learning

Apart from optimization, the superiority of learned optimizers is a natural fit for few-shot learning. Generally, FSL is a type of machine learning problems with only a limited number of labeled examples for a specific task [27]. In this paper, we mainly focus on FSL targeted at image classification, specifically N-way-K-shot classification. We deal with a group of meta-sets $\mathcal{D}_{\text{meta}}$ in this task. Each element in $\mathcal{D}_{\text{meta}}$ is a meta-set $D = (D_{\text{train}}, D_{\text{test}})$, where D_{train} is composed of K images for each of the N classes (thus $K \cdot N$ images in total) and D_{test} contains a number of examples for evaluation. The goal is to find an optimization strategy that trains a classifier leveraging D_{train} with only a few labeled examples to achieve good learning performance on D_{test}. All meta-sets are further divided into three separate sets: meta-training set $\mathcal{D}_{\text{meta-train}}$, meta-validation set $\mathcal{D}_{\text{meta-val}}$, and meta-testing set $\mathcal{D}_{\text{meta-test}}$. More concretely, $\mathcal{D}_{\text{meta-train}}$ is utilized to learn an optimizer and $\mathcal{D}_{\text{meta-val}}$ is for hyperparameter optimization. After the optimizer is determined, we conduct evaluation on $\mathcal{D}_{\text{meta-test}}$: we first update the classifier with the learned optimizer on the training-set in $\mathcal{D}_{\text{meta-test}}$; then we use the average accuracy on the test set in $\mathcal{D}_{\text{meta-test}}$ to evaluate the performance of the optimizer.

The N-way-K-shot classification problem can be simply incorporated into the L2L framework, where the optimization strategy is modeled by the learned optimizer. As we aim at training a classifier with high average performance on the testing set, instead of harnessing the whole optimization trajectory, the objective can be modified to attach attention only to the final testing loss:

$$\mathcal{L}_{\text{FSL}} = \mathbb{E}_{D\sim\mathcal{D}_{\text{meta}}}\mathbb{E}_{(x,y)\sim D_{\text{test}}}\left[\ell\left(f(\theta_T(\phi);x),y\right)\right], \tag{2}$$

where θ_T is updated based on a procedure described in Section 3.1 under examples from D_{train}.

4 Method

4.1 Motivation

Despite great potentials of neural optimizers in improving traditional optimization and few-shot learning, there exists a significant problem impeding the development of L2L. In contrast to classical hand-engineered optimization methods, those learned ones cannot guarantee a smooth update of parameters, i.e., producing similar outputs for similar states, where by state we mean the gradient or parameters of the optimizee. In Figure 2, we demonstrate the non-smoothness of the learned optimizer explicitly. This is a typical phenomenon in various neural-network-based algorithms such as image classification and reinforcement learning. [8] and [25] have pointed out advantages of smoothness of a function to mitigate overfitting, improve sample efficiency and stabilize the overall training procedure. Thus, enforcing the smoothness of the learned optimier can be crucial to improve its performance and stability.

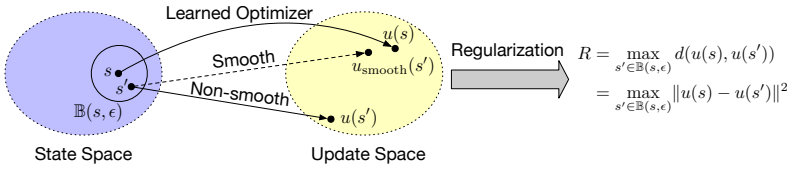

Fig. 2. An illustration of non-smoothness in the neural optimizer.

4.2 Smoothness Regularization

Some techniques, such as L_2 regularization and gradient clipping, have been developed and utilized in training neural optimizers to enforce smoothness but they are shown insufficient to reduce non-smoothness [1,16,18]. We propose to robustify the learned optimizer through a smoothness-inducing training procedure where a regularization term is introduced to narrow the gap between outputs of

two similar input states. It is also known as an effective method to constrain the Lipschitz constant of neural networks to boost smoothness.

To describe our method clearly, we first denote two states before updating the optimizee at the time step $t+1$ by s_t and s'_t. Note that s_t and s'_t are distinct but similar states, i.e., $s'_t \in \mathbb{B}(s_t, \epsilon)$, where $\mathbb{B}(s_t, \epsilon)$ represents the neighborhood of s within the ϵ-radius ball in a certain norm and ϵ is perturbation strength. In this paper, we just use ℓ_∞ norm without loss of generality. Fix the hidden state h_t, then u_t and u'_t, which are the corresponding parameter increments of s_t and s'_t, can be written as functions of the state $u(s_t)$ and $u(s'_t)$ explicitly. An ideal optimizer is expected to produce similar updates and thus to attain such an optimizer, our goal is to minimize the discrepancy $d(\cdot, \cdot)$ between $u(s_t)$ and $u(s'_t)$. Inspired by adversarial training [17], it is intuitive to find the gap under the worst-case as the targeted difference that takes the form of $\max d(u(s_t), u(s'_t))$ and minimize this term directly. However, the optimizer that takes the state of optimizee as input and the update as output, is different from the classifier whose input is an image and output is a vector of softmax logits. There is no classification for the optimizer so distance metrics such as cross-entropy and KL-divergence are not applicable to our problem. Since the output is a scalar value, we measure the distance with the squared difference and the desired gap at the time step $t+1$ becomes

$$R_{t+1}(\phi) = \max_{s'_t \in \mathbb{B}(s_t, \epsilon)} d(u(s_t), u(s'_t)) = \max_{s'_t \in \mathbb{B}(s_t, \epsilon)} \|u(s_t) - u(s'_t)\|^2. \tag{3}$$

After the regularization term is determined, we can then add it to the original objective function of L2L as a regularizer. For each time step, we have the following training objective:

$$\ell_t(\phi) = \ell(f(\theta_t(\phi); x), y) + \lambda R_t(\phi), \tag{4}$$

where λ is the regularization coefficient and the parameters ϕ of the optimizer is updated by

$$\min_\phi \mathcal{L}_{\text{opt}}(\phi) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\sum_{t=1}^{T} w_t \ell_t(\phi) \right]. \tag{5}$$

For few-shot learning, we store regularization terms during the training procedure with D_{train} and simply add the accumulation of them to Eq. 2, leading to the training of the learned optimizer as

$$\begin{aligned} \min_\phi \mathcal{L}_{\text{FSL}}(\phi) = & \mathbb{E}_{D\sim\mathcal{D}_{\text{meta}}} \Bigg[\mathbb{E}_{(x,y)\sim D_{\text{test}}} \ell \left(f(\theta_T(\phi); x), y \right) \\ & + \mathbb{E}_{(x,y)\sim D_{\text{train}}} \lambda \sum_{t=1}^{T} R_t(\phi) \Bigg]. \end{aligned} \tag{6}$$

It should be emphasized that our proposed regularizer can be applied to any neural optimizer-based algorithms in meta-learning, such as methods in [1,16,11].

4.3 Training the Optimizer

The key component for training the optimizer is the calculation of the regularization term in Eq. 3. As stated in [25], in practice we can effectively approximate the solution of the inner maximization by a fixed number of Projected Gradient Descent (PGD) steps:

$$s' = \Pi_{\mathbb{B}(s,\epsilon)}(\eta \operatorname{sign}(\nabla_{s'} d(u(s), u(s'))) + s'), \tag{7}$$

where Π is the projection operator to control the state located within the given radius of the neighborhood. Note that we use truncated Backpropagation Through Time (BPTT) to update our RNN optimizer in case of a too long horizon. For the predefined weight in Eq. 5, to make best use of the optimization trajectory and concentrate more on the loss of the last step at the same time [3], we adopt a linearly-increasing schedule that $w_t = t \bmod (T+1)$ where T is the number of step in each truncation. We present the whole training procedure in Algorithm 1 below.

Algorithm 1 Learning-to-Learn with Smooth-inducing regularization

Input: training data $\{(x, y)\}$, step sizes η_1 and η_2, inner steps K, total steps T_{total}, truncated steps T, classifier parameterized by θ, optimizer parameterized by ϕ
repeat
  Initialize θ randomly, reset RNN hidden state
  $\mathcal{L} \leftarrow 0$
  for $t = 0, \ldots, T_{\text{total}} - 1$ **do**
    Sample a batch of data (x, y), feed it 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
      $s'_t \leftarrow \Pi_{\mathbb{B}(s_t,\epsilon)}(\eta_1 \operatorname{sign}(\nabla_{s'_t} d(u(s_t), u(s'_t))) + s'_t)$
    end for
    $R_{t+1} \leftarrow \|u(s_t) - u(s'_t)\|^2$ ▷ Regularization term
    $\mathcal{L} \leftarrow \mathcal{L} + w_{t+1}\ell_{t+1}$ ▷ ℓ_{t+1} is computed by Eq. 4
    if $t \bmod T - 1 == 0$ **then**
      Update ϕ by $\mathcal{L}$ using Adam with η_2
      $\mathcal{L} \leftarrow 0$
    end if
  end for
until converged

Specifically, since our aim is to find a perturbed state in the neighborhood of the original state, we can obtain it as follows: a) Starting from the original state s, we add an imperceptible noise to initialize s'; b) Compute the current value of $d(u(s), u(s'))$, backpropagate the gradient back to s' to calculate $\nabla_{s'} d(u(s), u(s'))$, and then adjust the desired state by a small step η in the direction, i.e., $\operatorname{sign}(\nabla_{s'} d(u(s), u(s')))$, that maximizes the difference; (c) Run K steps in Eq. 7 to approximate the regularization term in Eq. 3.

5 Experimental Results

We are implementing comprehensive experiments for evaluation of our proposed regularizer. Detailed results are presented in Section 5.1 for neural network training and Section 5.2 for few-shot learning. All algorithms are implemented in PyTorch-1.2.0 with one NVIDIA 1080Ti GPU.

5.1 L2L for neural network training

In this part, we evaluate our method through the task of learning the general update rule for training neural networks. The performance of different optimization algorithms is primarily displayed in learning curves of both training and testing loss, as suggested in previous studies [1,16,18,4]. As loss and accuracy do not necessarily correlate, we also report accuracy curves for thoroughness.

Experiment settings We consider image classification on two popular datasets, MNIST [14] and CIFAR10 [13]. Our learned optimizer with regularization is compared with hand-designed methods including SGD, SGD with momentum (SGDM), Adam, AMSGrad, and RMSProp, as well as neural optimizers including DMOptimizer [1] and SimpleOptimizer [3]. For hand-designed optimizers, we tune the learning rate with grid search over a logarithmically spaced range $[10^{-4}, 1]$ and report the performance with the best hyperparameters. As to baseline neural optimizers, we use recommended hyperparameters, optimizer structures, and state definitions in [1] and [3] respectively. We have tried different hyperparameters for baselines and found that recommended ones are the best in our experiments. Our smoothed optimizers adopt original settings, except for two extra hyperparemeters for training, the perturbation strength ϵ and the regularization coefficient λ. In particular, ϵ and λ in our method are also determined by a logarithmic grid search with the range $\epsilon \in [10^{-2}, 10]$ and $\lambda \in [10^{-1}, 10^{2}]$. Neural optimizers are learned with Adam of the learning rate 10^{-4} with the number of total steps $T_{\text{total}} = 200$ and truncated steps $T = 20$. Note that for all neural optimizers we only tune the hyperparameters during training and directly apply them to a new optimization problem, while for hand-engineered algorithms, the learning rate is always tuned for the specific task. Experiments for each task are conducted five times with different seeds and the batch size used for following problems is 128. More implementation details are presented in Appendix C.

Compatibility of the proposed regularizer First of all, we conduct an experiment to demonstrate that the proposed regularization term can be combined with various L2L structures. We demonstrate the performance of learned optimizers including training loss and testing loss for training a 2-layer MLP on MNIST. As can be seen in Figure 3a and 3d, two L2L architectures, DMOptimizer [1] and SimpleOptimizer [3], are compared. With the regularizer, the smoothed version of both optimizers make an improvement in the final training and testing loss, and obtain a faster convergence rate at the same time. In

addition, since SimpleOptimizer performs better than DMOptimizer, which is consistent with the observation in [3], we will apply it as our base optimizer in the later experiments.

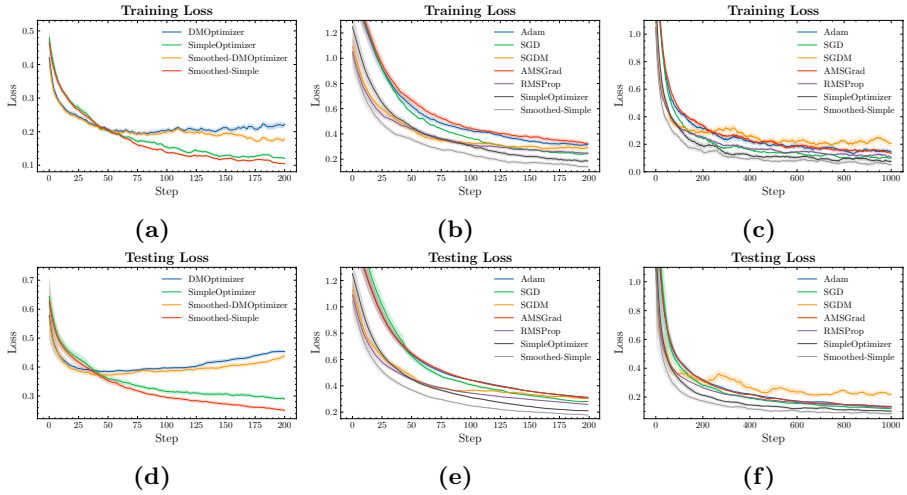

Fig. 3. Learning curves of classification on MNIST. Training loss is shown in the first row and testing loss in the second row. (a) and (d) are results of two neural optimizer structures to show the compatibility of our proposed regularizer; (b) and (e) demonstrate performance of different optimizers for training LeNet of 200 steps, while (c) and (f) extend the optimization to 1000 steps.

Training on MNIST In this experiment, we conduct experiments to train the neural optimizers for a 200-step optimization of LeNet on MNIST. We observe its performance under two scenarios:

(a) Training LeNet with different initializations. As the learned optimizer is originally trained to update parameters of LeNet, we directly apply it to optimize networks with the same architecture but distinct initializations. Performances of various optimizers in training and testing loss are presented in Figure 3b and 3e. We can see that our proposed smoothed optimizer outperforms all other baselines including hand-designed methods and the original SimpleOptimizer.

(b) Generalization to longer steps. Following [1], we also make an evaluation on optimization for more steps. Although the neural optimizer is only trained within 200 steps, it is capable of updating the optimizee until 1000 steps with faster convergence rate and better final loss consistently, as shown in Figure 3c and 3f.

Training on CIFAR-10 It is insufficient to merely test different optimizers on MNIST, whose size is relatively small. Therefore, we add to the difficulty of

the targeted task and focus on image classification on CIFAR-10. The classifier of interest is a 3-layer convolutional neural network with 32 units per layer and the learned optimizer is employed to update the optimizee for 10000 steps. It should be pointed out that the neural optimizer is still trained within 200 steps and the optimization step for evaluation is 50 times larger than what it has explored during training. Figure 4a and 4b demonstrate its great generalization ability: the smooth version of the learned optimizer can converge faster and better than hand-engineered algorithms such as SGD and Adam, even though it only observes the optimization trajectory in the limited steps at the very beginning. Our smoothed variant also outperforms the original learned optimizer.

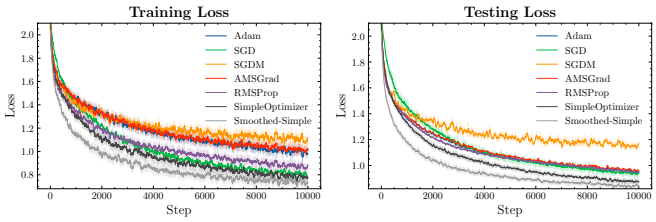

Fig. 4. Performance of training a 3-layer CNN for 10000 steps on CIFAR10.

Transferrablity to different settings After obtaining a neural optimizer trained on CIFAR-10 with a 3-layer CNN, we evaluate its transferrability in multiple aspects. Specifically, we first transfer the optimizer to training another network structure, ResNet-18 [9] for 10000 steps. In Figure 5a and 5d, Smoothed-Simple without finetuning can still beat all hand-designed methods. Besides, it should be emphasized that SimpleOptimizer oscillates violently at the end of training and loses its advantages over traditional methods for this transferring task, while the performance of our Smoothed-Simple is consistent and robust.

Moreover, since we have shown that our neural optimizer can generalize to longer training horizon and different network architectures on optimizees with the same dataset, this naturally leads to the following question: can our neural optimizer learn the intrinsic update rule so that it can generalize to unseen data? To answer this question, we modify the experimental setting to evaluate our proposed optimizer on unseen data. We split the original CIFAR-10 dataset into three different sets: a training set containing 6 classes, a validation set and a testing set with 2 classes respectively. When training the optimizer, we sample 2 classes from training set and minimize the objective function for a binary classification problem. Images in the validation set are exploited to select the optimizer which achieves best final testing loss in the 200-step optimization. A comparison of learning curves among our smoothed optimizer, SimpleOptimizer, and the rest hand-designed methods for updating the classifier on two unseen classes is shown in Figure 5b and 5e. We can observe that the smoothed optimizer learns more quickly and better than other algorithms.

Finally, we test the performance of our optimizer in the most difficult setting: training a ResNet-18 on CIFAR-100, where both the network structure and the dataset are modified. It can be observed in Figure 5c and 5f that Smoothed-Simple has comparable performance with fine-tuned hand-engineered optimizers in terms of testing loss. On the contrary, SimpleOptimizer is incapable of dealing with this scenario with a large dataset and a complicated network.

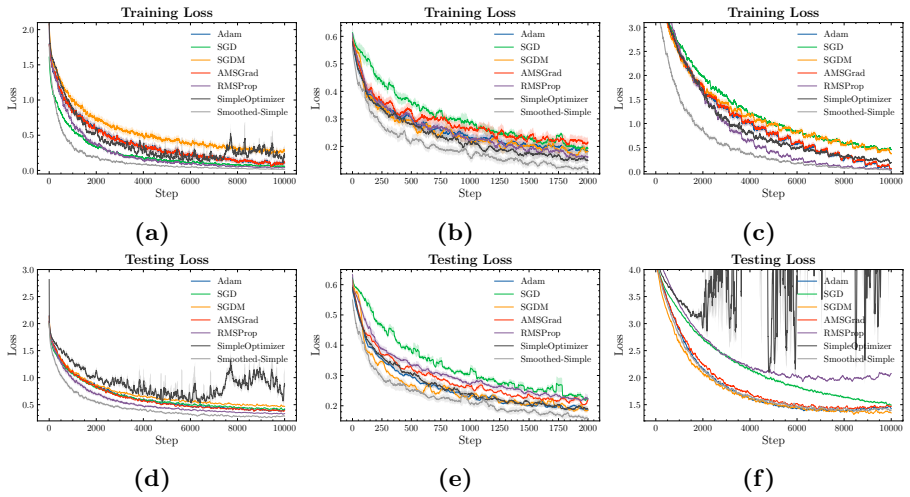

Fig. 5. Performance under transferred settings. (a) and (d) are results of 10000-step optimization of ResNet-18. Results of a designed binary classification are reported in (b) and (e). (c) and (f) are results of a transferring task to train a ResNet-18 on CIFAR100.

Comparison with other regularization methods. As the proposed method serves as a novel regularization term, for the completeness of experiments, we compare our adversarial regularization with three representative techniques: ℓ_2 regularization [24], orthogonal regularization [2], and spectral normalization [19]. In detail, we train a 3-layer CNN on CIFAR-10 for 10000 steps, following the setting in Section 5.1. Results of training and testing loss can be found in Figure 6, and we also report the test accuracy for reference in Table 1. It can be observed that orthogonal regularization even worsens the learned optimizer, which cannot provide meaningful updates and only leads to a random-guess classifier with 10.00% on test accuracy. While ℓ_2 regularization and spectral normalization can improve the SimpleOptimizer to 69.69% and 69.35% respectively, our proposed Smoothed-Simple still outperforms them significantly with 72.50%. This experiment shows that the smoothness-inducing regularization obtained by PGD can achieve performance gain against other popular regularization techniques and is more suitable in training a neural optimizer.

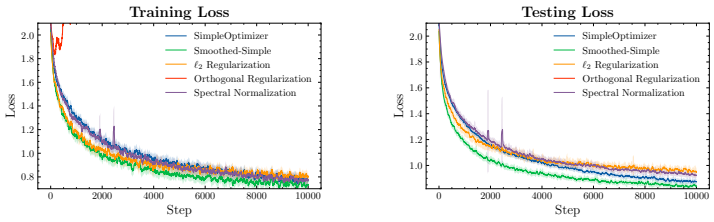

Fig. 6. Performance of neural optimizers with different regularization methods.

Table 1. Test accuracy of different regularizers.

Regularizer	**Test accuracy**
SimpleOptimizer	69.02 ± 0.58%
Simple-Smoothed	**72.50 ± 0.49%**
ℓ_2 Regularization	69.69 ± 0.56%
Orthogonal Regularization	10.00 ± 0.04%
Spectral Normalization	69.35 ± 1.23%

Smoothness with Perturbation In this section, we analyze the optimizer's smoothness property with perturbation. Detailedly, we sample 1000 points from $\mathcal{N}(0, 0.1)$ to form a set of perturbed states around 0. Then these states are fed into the simple and smoothed optimizer respectively, and corresponding outputs are shown in Figure 7 (x-axis in (a) is the state number while (b) sorts the specific state values.) Around the zero state with zero gradient, the update with a small magnitude is expected for an ideal optimizer. We can see that the smooth version can produce much more stable updates around zero state, while Simple-Optimizer suffers from non-smoothness.

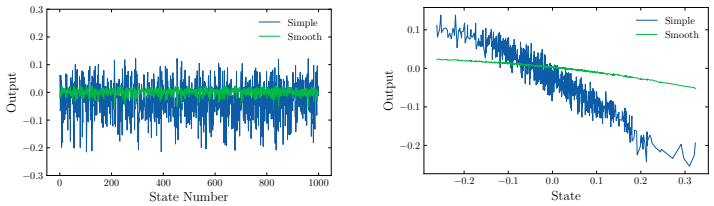

Fig. 7. A comparison of smoothness between simple and smoothed optimizer.

Additional Evaluation Besides the metric of loss, we explore classification accuracy which is another important performance indicator, to show advantages of our smoothed neural optimizer. In Figure 8, we present curves of training and testing accuracy, for MNIST with LeNet and CIFAR10 with the 3-layer CNN. We can observe that the relative ranks of all optimizers do not change if

the evaluation metric is switched to accuracy, and our smoothed optimizer still outperforms other algorithms with best final training and testing accuracy as well as convergence rate. Furthermore, we conduct experiments on a comparatively large-scale dataset, tiny-ImageNet in Appendix D. Similar performance on this dataset shows the effectiveness of our proposed method. We also include preliminary experimental results on a sentiment analysis task in Appendix D.

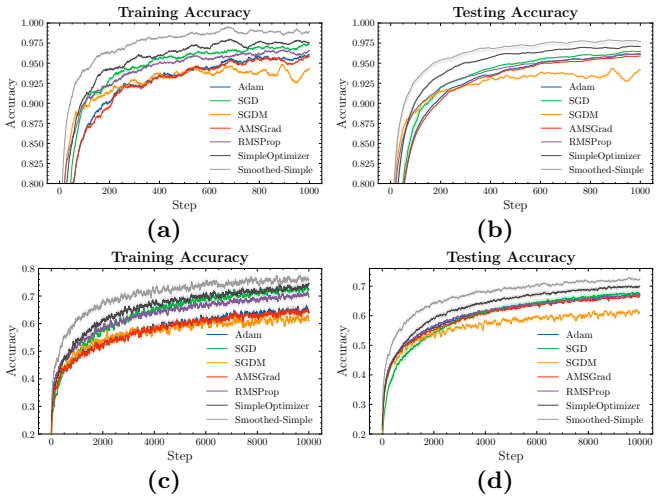

Fig. 8. Performance of different optimizers in training and testing accuracy. (a)-(b) for MNIST with LeNet and (c)-(d) for CIFAR-10 with a 3-layer CNN.

Furthermore, we conduct experiments on a comparatively large-scale dataset, tiny-ImageNet in Appendix D.3. Similar performance on this dataset shows the effectiveness of our proposed method.

5.2 Few-Shot Learning with LSTM

Apart from improving the training procedures, L2L can be applied to few-shot learning as well. Therefore, in this part we primarily explore the effectiveness of our smoothed neural optimizer in FSL, in particular, N-way-K-shot learning. We consider 5-way-1-shot and 5-way-5-shot problems on two benchmark datasets, miniImageNet [26] and tieredImageNet [21]. The base structure we utilize is Meta-LSTM, proposed in [20] to train an LSTM-based meta learner to learn the optimization rule in the few-shot regime. We compare it with our smoothed version. We keep all hyperparameters the same as reported in [20] and only tune ϵ and λ in a manner introduced in Section 5.1. Statistical results of 5 experiments with different random seeds are reported in Table 2. Our smoothed Meta-LSTM attains 2% percents improvement over all scenarios against the baseline. It should be emphasized that the performance boost is purely credited to the regularizer

since we apply our regularization term to the exactly same structure as Meta-LSTM. Since the official code for Meta-LSTM is written in lua and is out-of-date, we use the latest PyTorch implementation in [6]. Thus, our results might lead to inconsistency with the original paper but do not affect the conclusion.

Table 2. Average accuracy of 5-way few shot learning on miniImageNet and tieredIamgeNet.

Model	miniImageNet		tieredImageNet	
	1-shot	5-shot	1-shot	5-shot
Meta-LSTM	$38.20 \pm 0.73\%$	$56.56 \pm 0.65\%$	$36.43 \pm 0.65\%$	$53.45 \pm 0.61\%$
Smoothed Meta-LSTM	$\mathbf{40.42 \pm 0.68\%}$	$\mathbf{58.90 \pm 0.61\%}$	$\mathbf{36.74 \pm 0.76\%}$	$\mathbf{55.14 \pm 0.60\%}$

In addition, we integrate our proposed regularizer into one of the most recent methods involving a neural optimizer, SIB [11] on miniImageNet and CIFAR-FS. Results are presented in Table 3 and with regularization, SIB performs consistently better especially for 5-shot tasks.

Table 3. Average accuracy of 5-way few shot learning problems on miniImageNet and CIFAR-FS.

Model	Backbone	miniImageNet		CIFAR-FS	
		1-shot	5-shot	1-shot	5-shot
SIB($\eta = 1e^{-3}$, $K = 3$)	WRN-28-10	$69.6 \pm 0.6\%$	$78.9 \pm 0.4\%$	$78.4 \pm 0.6\%$	$85.3 \pm 0.4\%$
Smoothed SIB	WRN-28-10	$\mathbf{70.0 \pm 0.5\%}$	$\mathbf{80.8 \pm 0.3\%}$	$\mathbf{79.2 \pm 0.4\%}$	$\mathbf{86.1 \pm 0.4\%}$

6 Conclusion and Discussion

This paper first investigates the smoothness of learned optimizers and leverage it to achieve performance improvement. Specifically, we propose a regularization term for neural optimizers to enforce similar parameter updates given similar input states. Extensive experiments show that the regularizer can be combined with different L2L structures involving neural optimizers, and verify its effectiveness of consistently improving current algorithms for various tasks in classification and few-shot learning. Despite promising results, currently the learned optimizer is constrained to a group of problems with a moderate number of optimization steps and cannot replace hand-crafted ones in such settings. Training a powerful optimizer that can generalize to longer horizon still remains a challenge and can be a potential future direction. Besides, how to design a neural optimizer to deal with language tasks with RNNs or even more complex models like Transformers is also an interesting problem to be explored.

Acknowledgements. This work is supported by NSF IIS-2008173, IIS-2048280 and Google.

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. Bansal, N., Chen, X., Wang, Z.: Can we gain more from orthogonality regularizations in training deep cnns? arXiv preprint arXiv:1810.09102 (2018)
3. Chen, P.H., Reddi, S., Kumar, S., Hsieh, C.J.: Learning to learn with better convergence (2020), https://openreview.net/forum?id=S1xGCAVKvr
4. Chen, T., Zhang, W., Jingyang, Z., Chang, S., Liu, S., Amini, L., Wang, Z.: Training stronger baselines for learning to optimize. Advances in Neural Information Processing Systems **33** (2020)
5. Cotter, N.E., Conwell, P.R.: Fixed-weight networks can learn. In: 1990 IJCNN International Joint Conference on Neural Networks. pp. 553–559. IEEE (1990)
6. Dong, M.: Pytorch implementation of optimization as a model for few-shot learning. https://github.com/markdtw/meta-learning-lstm-pytorch (2019)
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
8. Hampel, F.R.: The influence curve and its role in robust estimation. Journal of the american statistical association **69**(346), 383–393 (1974)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
10. Hinton, G., Srivastava, N., Swersky, K.: Neural networks for machine learning lecture 6a overview of mini-batch gradient descent (2012)
11. Hu, S.X., Moreno, P.G., Xiao, Y., Shen, X., Obozinski, G., Lawrence, N.D., Damianou, A.: Empirical bayes transductive meta-learning with synthetic gradients. arXiv preprint arXiv:2004.12696 (2020)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Krizhevsky, A., Nair, V., Hinton, G.: The cifar-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html **55** (2014)
14. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
15. Luo, L., Xiong, Y., Liu, Y., Sun, X.: Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:1902.09843 (2019)
16. 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)
17. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
18. 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)
19. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: International Conference on Learning Representations (2018)
20. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR (2017)

21. Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J.B., Larochelle, H., Zemel, R.S.: Meta-learning for semi-supervised few-shot classification. arXiv preprint arXiv:1803.00676 (2018)
22. Robbins, H., Monro, S.: A stochastic approximation method. The annals of mathematical statistics pp. 400–407 (1951)
23. Ruan, Y., Xiong, Y., Reddi, S., Kumar, S., Hsieh, C.J.: Learning to learn by zeroth-order oracle. arXiv preprint arXiv:1910.09464 (2019)
24. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks **61**, 85–117 (2015)
25. Shen, Q., Li, Y., Jiang, H., Wang, Z., Zhao, T.: Deep reinforcement learning with smooth policy. arXiv preprint arXiv:2003.09534 (2020)
26. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: Advances in neural information processing systems. pp. 3630–3638 (2016)
27. Wang, Y., Yao, Q., Kwok, J., Ni, L.M.: Generalizing from a few examples: A survey on few-shot learning. arXiv preprint arXiv: 1904.05046 (2019)
28. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3751–3760. JMLR. org (2017)
29. Wu, Y., Ren, M., Liao, R., Grosse, R.: Understanding short-horizon bias in stochastic meta-optimization. arXiv preprint arXiv:1803.02021 (2018)
30. Xiong, Y., Hsieh, C.J.: Improved adversarial training via learned optimizer. arXiv preprint arXiv:2004.12227 (2020)
31. Younger, A.S., Hochreiter, S., Conwell, P.R.: Meta-learning with backpropagation. In: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222). vol. 3. IEEE (2001)