LEARNING TO LEARN WITH SMOOTH REGULARIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent decades have witnessed great prosperity of deep learning in tackling various problems such as classification and decision making. The rapid development stimulates 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—when provided with similar states (a combination of some metrics to describe the optimizee), 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 neural 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 data. Furthermore, we show that our regularizer can improve the performance of neural optimizers on few-shot learning tasks.

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 (Chen et al., 2020; Lv et al., 2017). Effective algorithms such as SGD (Robbins & Monro, 1951), Adam (Kingma & Ba, 2014) and AdaBound (Luo et al., 2019) have been proposed to work well on a variety of tasks. In parallel to this line of hand-designed methods, Learning-to-Learn (L2L) (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2018; Lv et al., 2017; Chen et al., 2020), 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 (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2018; Lv et al., 2017; Chen et al., 2020). In addition, such advantages in faster training make the learned optimizer a great fit for few-shot learning (FSL) (Ravi & Larochelle, 2017; Hu et al., 2020), 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 (Metz et al., 2018) and short-horizon bias (Wu et al., 2018). 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 (Goodfellow et al., 2014), where the perturbed image can fool the classifier to make a wrong prediction. Inspired by the progress in adversarial training (Madry et al., 2017; Zhang et al., 2019) 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 the main contributions of this paper 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 data.
- In addition to generic neural network training, we also conduct experiments on few-shot learning based on a Meta-LSTM optimizer (Ravi & Larochelle, 2017) and SIB (Hu et al., 2020). Results show that our smoothness-inducing regularizer consistently improves the accuracy on two FSL benchmark datasets.

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 (Robbins & Monro, 1951), Adam (Kingma & Ba, 2014), 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 Cotter & Conwell (1990) leveraged recurrent neural networks to model adaptive optimization algorithms. The idea was further developed in Younger et al. (2001) where neural optimizers were trained to tackle fundamental convex optimization problems. Recently in the era of deep learning, a seminal work of Andrychowicz et al. (2016) 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 Lv et al. (2017) and Wichrowska et al. (2017) have improved the generalization and scalability of learned optimizers. In parallel to this work, automatic optimization can be considered from a Reinforcement Learning (RL) perspective. RL primarily aims at finding an optimal policy to schedule the learning rate, more like hyperparameter optimization (Li & Malik, 2016; Bello et al., 2017). Compared with RL-based methods, the 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 (Ravi & Larochelle, 2017), zeroth-order optimization (Ruan et al., 2019) and adversarial training (Xiong & Hsieh, 2020).

This paper is the first to investigate the smoothness of neural optimizers in the L2L framework. It is related to the notion of adversarial robustness in classification models. As observed in Goodfellow et al. (2014), 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 (Goodfellow et al., 2014), while for reinforcement learning the agent is likely to act differently under perturbed states (Shen et al., 2020). 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 (Madry et al., 2017), TRADES (Zhang et al., 2019), and SR^{2}L (Shen et al., 2020). 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

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

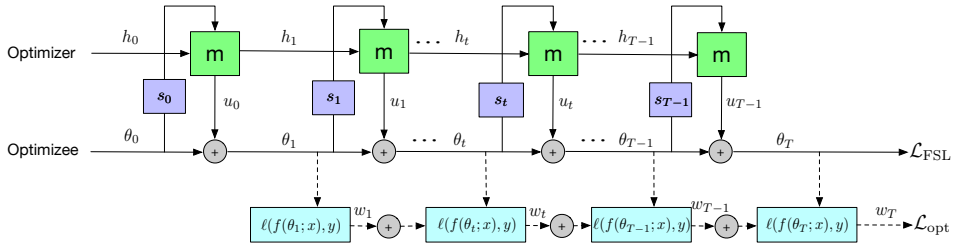


Figure 1: The framework of learning-to-learn. The dashed line shows the computation graph of the objective function $\mathcal{L}_{\text{opt}}$ for training the optimizer to learn a general update rule while the horizontal full line is the one for few-shot learning. Note that m is the neural optimizer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

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 such as the current parameter value, its gradient, or the exponentially weighted moving averages of gradient.
(b) Given the current state s_t and the hidden state h_t, the neural optimizer m parameterized by ϕ can 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 Andrychowicz et al. (2016), 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:

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

Here $\ell(\cdot, \cdot)$ 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 (Wang et al., 2019). In this paper, we mainly focus on FSL targeted at image classification, specifically N-way-K-shot classification. We deal with meta-sets $\mathcal{D}_{\text{meta}}$ in this task. Each $D = \{D_{\text{train}}, D_{\text{test}}\} \in \mathcal{D}_{\text{meta}}$, 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}.

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}. Like normal classification problems, in few-shot learning, all meta-sets are further divided into three separate sets: meta-training set for learning the optimizer, meta-validation set for hyperparameter optimization and model selection, and meta-testing set for performance evaluation.

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. Hampel (1974) and Shen et al. (2020) 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.

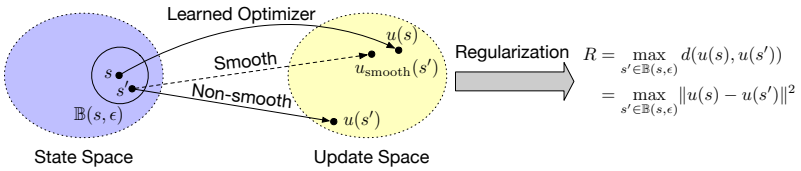


Figure 2: An illustration of the non-smoothness in the neural optimizer.

4.2 SMOOTHNESS REGULARIZATION

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. 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 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 also called as 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 $u(s'_t)$. Like adversarial training, it is intuitive to find the gap under the worst-case as the targeted difference, which takes the form of $\max d(u(s_t), u(s'_t))$. 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 in Madry's adversarial training (Madry et al., 2017) and KL-divergence in TRADES (Zhang et al., 2019) 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, the objective becomes

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

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

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

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 Zhang et al. (2019) and Shen et al. (2020), 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 last step at the same time (Chen et al., 2020), we adopt a linearly-increasing schedule that $w_t = t \bmod T$ where T is the number of step in each truncation. We present the whole training procedure in Algorithm 1.

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 $d(u(s), u(s'))$, backprobagate 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. Time cost by computing this regularization term has little impact on efficiency of the algorithm and more discussions are included in Appendix B.

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

Input: training data $\{(x, y)\}$, step sizes η_1 and η_2, number of inner iterations 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 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
    $\mathcal{L} \leftarrow \mathcal{L} + w_{t+1}\ell_{t+1}$ ▷ ℓ_{t+1} is computed by Eq. 4
    if $t \bmod T - 1 == 0$ **then**
      Update ϕ by $\mathcal{L}$ using Adam with the step size η_2, $\mathcal{L} \leftarrow 0$
    end if
  end for
until converged

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 displayed in learning curves of both training and testing loss.

5.1.1 EXPERIMENT SETTINGS

Specifically, we consider image classification on two popular datasets, MNIST and CIFAR10. Our learned optimizer with regularization is compared with hand-designed methods including SGD,

SGD with momentum (SGDM), and Adam, as well as neural optimizers including DMOptimizer (Andrychowicz et al., 2016) and SimpleOptimizer (Chen et al., 2020). 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 (Andrychowicz et al., 2016) and (Chen et al., 2020) respectively. It should be clarified that we have tried different hyperparameters for baselines and found that recommended ones are the best in our experiments. Our smoothed optimizers are almost based on 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 fives times with different seeds and the batch size used for following problems is 128. More implementation details are presented in Appendix B.

5.1.2 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 (Andrychowicz et al., 2016) and SimpleOptimizer (Chen et al., 2020), 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 Chen et al. (2020), we will apply it as our base optimizer in the later experiments.

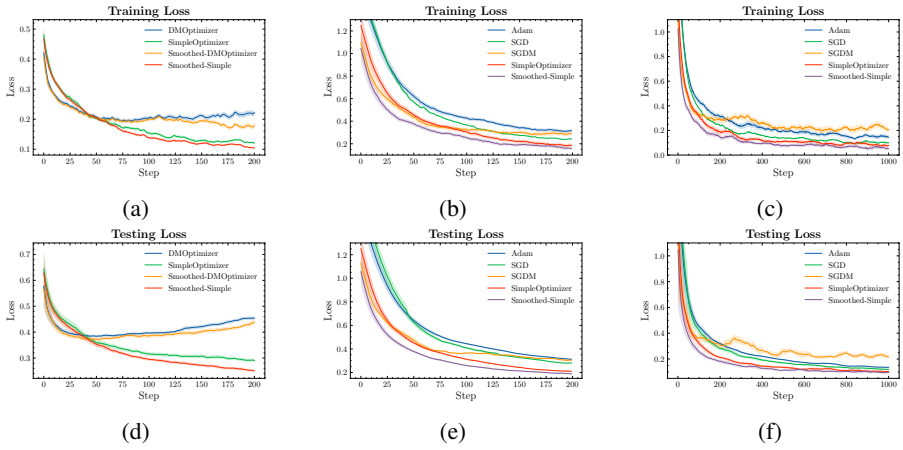


Figure 3: Learning curves of classification on MNIST. Training loss is shown in the first row and testing loss in the second row. (a) and (d) are results of two neural optimizer structures to show the compatibility of our proposed regularizer; (b) and (e) demonstrate performance of different optimizers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

5.1.3 Training on MNIST

In this experiment, we conduct experiments to train the neural optimizers for a 200-step optimization of LeNet on MNIST dataset. We observe its performance under the following 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 by a large margin.

(b) Generalization to more optimization steps. Following Andrychowicz et al. (2016), we also make an evaluation on optimization for more steps. Despite the fact that 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.

5.1.4 TRAINING ON CIFAR-10

It is insufficient to merely test different optimizers on MNIST, whose size is a 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 4d 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. Moreover, we transfer the optimizer to training another network structure, GoogLeNet. In Figure 4b and 4e, Smoothed-Simple without finetuning can still beat a majority of hand-designed methods except for SGD and SimpleOptimizer, reflecting its powerful transferability.

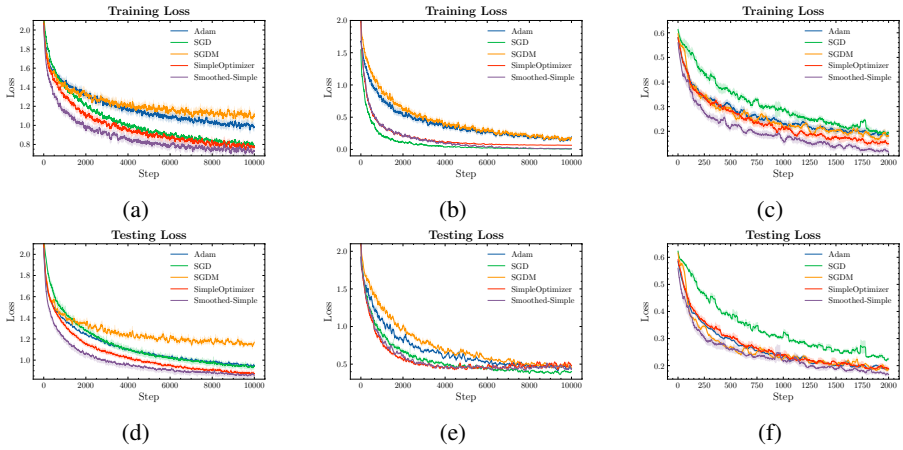


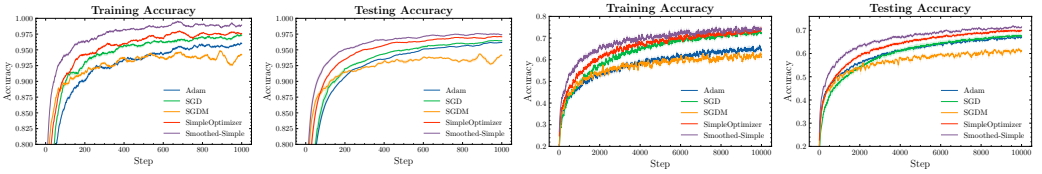
Figure 4: Learning curves of classification on CIFAR-10. (a) and (d) show performance of training a 3-layer CNN for 10000 steps while (b) and (e) are results of 10000-step optimization of GoogLeNet. Results of a designed binary classification are reported in (c) and (f).

We have shown that our neural optimizer can generalizing to the same dataset but longer training horizon and different network architectures. 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 with respect to optimization 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. In Figure 4c and 4f, we can see a comparison of learning curves among our smoothed optimizer, SimpleOptimizer, and the rest hand-designed methods for updating the classifier on two unseen classes. The smoothed optimizer learns much more quickly than other algorithms.

5.1.5 ADDITIONAL EVALUATION

Besides the metric of loss, we explore another aspect, classification accuracy, to show advantages of our smoothed neural optimizer. In Figure 5, we present curves of training and testing accuracy, for MNIST with LeNet and CIFAR10 with the 3-layer CNN. It can be observed that our method outperforms others with best final training and testing accuracy as well as convergence rate. We also

investigate the smoothness of optimizers in Appendix D, and results show that our method can boost the smoothness of neural optimizers confronted with state disturbance.



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

Figure 5: Learning curves 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 C. 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 (Vinyals et al., 2016) and tieredImageNet (Ren et al., 2018). The base structure we utilize here is Meta-LSTM, proposed in (Ravi & Larochelle, 2017) 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 (Ravi & Larochelle, 2017) and only tune ϵ and λ in a manner introduced in Section 5.1.1. Statistical results of 5 experiments with different random seeds are reported in Table 1. 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 Dong (2019). Thus, our results might lead to inconsistency with the original paper but do not affect the conclusion.

Table 1: 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 (Hu et al., 2020) on miniImageNet and CIFAR-FS. Results are presented in Table 2 and with regularization, SIB performs consistently better especially for 5-shot tasks.

Table 2: 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

This paper first investigates the smoothness of learned optimizers and takes it into consideration 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 and verify its effectiveness of consistently improving current algorithms for various tasks in classification and few-shot learning.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 459–468. JMLR. org, 2017.

Patrick H. Chen, Sashank Reddi, Sanjiv Kumar, and Cho-Jui Hsieh. Learning to learn with better convergence, 2020. URL `https://openreview.net/forum?id=S1xGCAVKvr`.

Neil E Cotter and Peter R Conwell. Fixed-weight networks can learn. In *1990 IJCNN International Joint Conference on Neural Networks*, pp. 553–559. IEEE, 1990.

Mark Dong. Pytorch implementation of optimization as a model for few-shot learning. `https://github.com/markdtw/meta-learning-lstm-pytorch`, 2019.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

Shell Xu Hu, Pablo G Moreno, Yang Xiao, Xi Shen, Guillaume Obozinski, Neil D Lawrence, and Andreas Damianou. Empirical bayes transductive meta-learning with synthetic gradients. *arXiv preprint arXiv:2004.12696*, 2020.

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

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

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

Kaifeng Lv, Shunhua Jiang, and Jian Li. 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.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, C Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. *arXiv preprint arXiv:1810.10180*, 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

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

Yangjun Ruan, Yuanhao Xiong, Sashank Reddi, Sanjiv Kumar, and Cho-Jui Hsieh. Learning to learn by zeroth-order oracle. *arXiv preprint arXiv:1910.09464*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with smooth policy. *arXiv preprint arXiv:2003.09534*, 2020.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Yaqing Wang, Quanming Yao, James Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *arXiv preprint arXiv: 1904.05046*, 2019.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3751–3760. JMLR. org, 2017.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

Yuanhao Xiong and Cho-Jui Hsieh. Improved adversarial training via learned optimizer. *arXiv preprint arXiv:2004.12227*, 2020.

A Steven Younger, Sepp Hochreiter, and Peter R Conwell. Meta-learning with backpropagation. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3. IEEE, 2001.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, pp. 7472–7482, 2019.

A Algorithm for Few-Shot Learning

Algorithm 2 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 Implementation Details

Extra time cost caused by our regularization term has little impact on the efficiency of training the neural optimizer. As the dimension of the state is relatively small, the gradient can be obtained efficiently considering the computational complexity of backprobagation. Specifically, training time

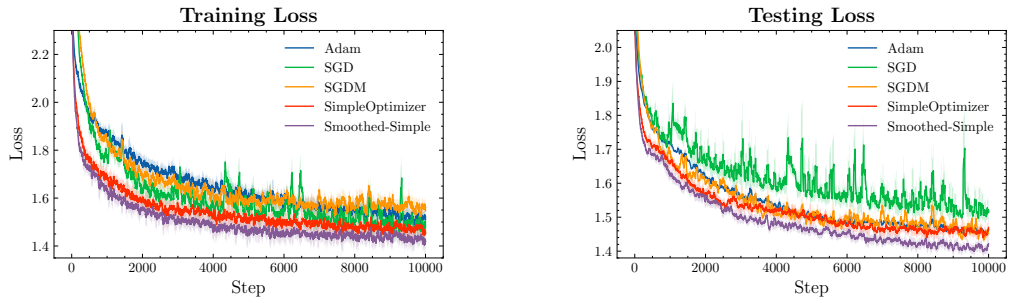
per epoch for SimpleOptimizer is 38.21s while for our method is 97.31s. As we only train the learned optimizer for a few epochs and then apply it to the target task, we care more about time for deploying the neural optimizer. This time is nearly the same as the hand-designed ones. Moreover, since PGD has been acknowledged as a practical and effective method in adversarial attacks, it is feasible to obtain s' in our case as well.

For DMOptimizer Andrychowicz et al. (2016), we adopt a 2-layer LSTM with the hidden size of 10, while for SimpleOptimizer Chen et al. (2020), 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 Ravi & Larochelle (2017). For the selection of ϵ, we observed the magnitude of states during training the classifier, and then determine the range for tuning ϵ as $[10^{-2}, 10]$ to allow for a reasonable neighborhood.

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 Andrychowicz et al. (2016). We only have the gradient normalized by the second momentum as the state in SimpleOptimizer Chen et al. (2020). The state of Meta-LSTM Ravi & Larochelle (2017) is based on DMOptimizer with an extra feature, the original gradient.

C EXPERIMENTS ON TINY-IMAGENET

We evaluate the optimizers on tiny-ImageNet, which is a relatively larger dataset extracted from ILSVRC-12 (Russakovsky et al., 2015). 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 6, 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

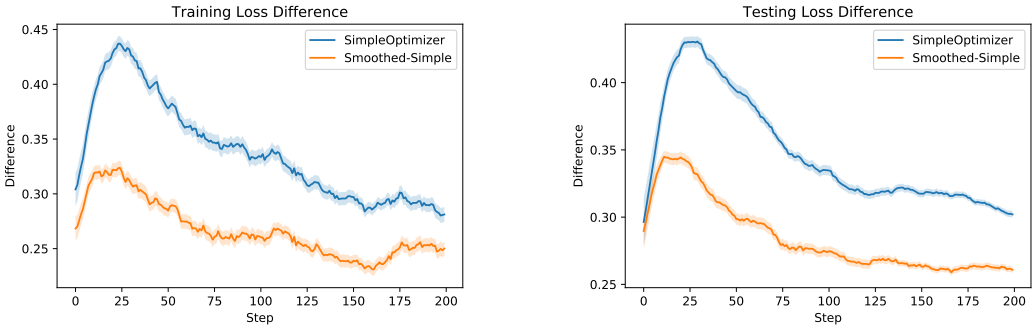
Figure 6: Learning curves of classification on tiny-ImageNet.

D SMOOTHNESS UNDER PERTURBATION

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 $\delta \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ 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 7. Note that the difference is calculated by

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

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.



(a) Training

(b) Testing

Figure 7: Loss difference of SimpleOptimizer and Smoothed-Simple.