Scalable Learning to Optimize: A Learned Optimizer Can Train Big Models

Xuxi Chen[1⋆], Tianlong Chen[1⋆], Yu Cheng[2], Weizhu Chen[2]
Ahmed Awadallah[2], and Zhangyang Wang[1]

[1] The University of Texas at Austin, Austin TX 78712, USA
{xxchen,tianlong.chen,atlaswang}utexas.edu
[2] Microsoft Research
{yu.cheng,wzchen,hassanam}@microsoft.com

Abstract. Learning to optimize (**L2O**) has gained increasing attention since it demonstrates a promising path to automating and accelerating the optimization of complicated problems. Unlike manually crafted classical optimizers, L2O parameterizes and learns optimization rules in a data-driven fashion. However, the primary barrier, *scalability*, persists for this paradigm: as the typical L2O models create massive memory overhead due to unrolled computational graphs, it disables L2O's applicability to large-scale tasks. To overcome this core challenge, we propose a new scalable learning to optimize (**SL2O**) framework which (*i*) first constrains the network updates in a tiny subspace and (*ii*) then explores learning rules on top of it. Thanks to substantially reduced trainable parameters, learning optimizers for large-scale networks with a single GPU become feasible for the first time, showing that **the scalability roadblock of applying L2O to training large models is now removed**. Comprehensive experiments on various network architectures (i.e., ResNets, VGGs, ViTs) and datasets (i.e., CIFAR, ImageNet, E2E) across vision and language tasks, consistently validate that SL2O can achieve significantly faster convergence speed and competitive performance compared to analytical optimizers. For example, our approach converges 3.41 $\sim$ 4.60 times faster on CIFAR-10/100 with ResNet-18, and 1.24 times faster on ViTs, at nearly no performance loss. Codes are in https://github.com/VITA-Group/Scalable-L2O.

1 Introduction

Gradient-based optimization methods are prevailing in the deep learning field, and over years dozens of gradient-based optimizers have been designed by researchers based on their expertise. Most of these optimizers apply specific rules to calculate the update of parameters from their gradients, such as using momentum [34] or normalized gradients [13]. These manually crafted optimizers can mostly be expressed in a handful of analytical formulas, and they are often equipped with theoretical guarantees on some classes of optimization tasks [5].

⋆ Equal Contribution.

However, recent works have pointed out that when focusing on a specific category of optimization tasks, one can pursue a different pathway to *learn* an optimizer instead of applying these hand-crafted optimizers to achieve better performance. This alternative paradigm, called Learning to Optimize (L2O) [28], aims at learning a more effective optimization algorithm from data. As depicted in Figure 1, L2O normally takes the optimizee's training dynamic as input, and output optimization rules. Such learnable optimizers are capable of learning "shortcuts" that hand-crafted optimization algorithms fail to leverage [28], and they have demonstrated faster convergence speed and higher solution quality [5] and even save energy cost [25].

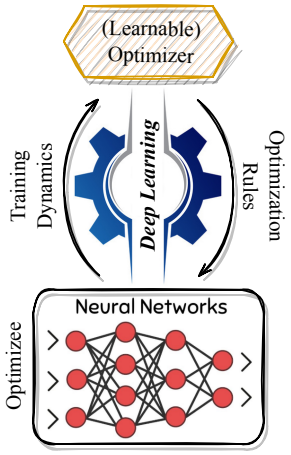

Fig. 1: The L2O pipeline.

L2O methods typically require a *meta-training* stage where a optimizer is learned with a set of *optimizees* sampled from a given *task distribution*. The learned optimizers are parameterized by neural networks, typically by recurrent neural networks [2,4,30,38]. However, L2O optimizers suffer from low *scalability*: the memory overhead due to the unrolled computational graphs required by training L2O optimizers limits the scales of optimization problems. For instance, [4] studied problems at matrix multiplication levels. [6] studied a three-layer multi-layer perceptron (MLP) ($\sim 10^4$ parameters) and a two-layer convolutional networks ($\sim 10^4$ parameters). [31,46] studied multi-layer MLPs for MNIST and CIFAR10 classification ($\sim 10^5$ parameters). [50] used the learned optimizer's weight to optimize Inception-V3 [41] but did not perform meta-training on it. [3] performed meta-training on Wide ResNet but they require thousands of CPU hours to parallelly train their RNN optimizers with multiple nodes. This main hurdle obstacles the more general application of L2O methods.

Our proposed solution, which aims at tackling the aforementioned obstacle, is a **subspace training framework for L2O**. Recent works [26,15,20,29] have suggested an alternative but effective way to train neural networks, *i.e.*, constraining the weight updates in tiny subspaces. The number of *independent* parameters is smaller in the subspace compared to full fine-tuning; therefore the corresponding subspace optimization problem is simplified. Motivated by these recent signs of progress on subspace training, we propose to reparameterize optimizee's weight updates inside a low dimensional subspace, making the optimization problem more memory-friendly for L2O. By reducing the number of independent parameters, the L2O models will track fewer intermediate representations for parameters, leading to smaller computational graphs and trimmed memory costs. For the first time, we enable the training of L2O models on giant models such as Vision Transformer [12] ($\sim 10^8$ parameters) and GPT-2 [35] ($\sim 10^8$ parameters) on a single GPU. Contributions are summarized as follows:

- ⋆ We for the first time demonstrate that L2O can be scaled up to large-scale models such as ViT, removing the previous scalability roadblock of applying L2O methods. The keys behind scalability are simple subspace re-parameterization techniques.
- ⋆ We propose a novel L2O training framework, **SL2O**, that seamlessly integrates subspace re-parameterization methods. We show that SL2O is applicable on a broad range of architectures like ResNets, VGG, and ViT[3], and can bring significantly better convergence and improved performance.
- ⋆ Extensive experiments on vision (CIFAR-10, CIFAR-100, ImageNet) and language tasks (E2E) with large-scale networks validate the superiority of our proposals. For example, our learned optimizer obtains nearly unimpaired improvements with 29.3%/21.7% training iterations and 40 training parameters on ResNet-18 with CIFAR-10/CIFAR-100, which leads to impressive resource-efficiency compared to vanilla network training.

2 Related Work

Low-Rank Structure in Training Neural Networks. Literature [54,26,32,16] point out that the intrinsic dimensionality of trained over-parameterized models is naturally low-rank. For example, [26,15] perform optimization in a reduced subspace formed by random bases, leading to around 90% performance of regular SGD training. More works focus on imposing explicit low-rank constraints during training [21,33,37,56,57] and transferring [48,20], which obtain considerable parameter efficiency. In the meantime, such low-rank structures enable more powerful optimization algorithms to address existing learning barriers like convergence speed. Specifically, thanks to largely reduced optimization variables, [44,40,29] exploit higher-order information and design delicate training approaches using curvature or Hessian, while maintaining overall computation efficiency and improving convergence. Different from previous works, we consider leveraging a superior learned optimizer to update within the tiny update subspace.

Learning to Optimize. Instead of hand-crafted optimization rules (e.g., SGD, Adam, and RMSprop), learning to optimize (L2O) leverages a data-driven learned model as the optimizer, which has achieved various successes in machine learning problems including black-box optimization [9], Bayesian swarm optimization [4], min-max optimization [38], domain adaptation [25,7], adversarial training [22,51], graph learning [53], and noisy label training [8]. [1] invents the first L2O pipeline parameterized by a long short-term memory (LSTM), which takes optimizee's gradients as input and outputs its update rules. It adopts a coordinate-wise manner that allows learned optimizers to be applicable for optimizees with different amounts of parameters. Another alternative reinforcement learning framework is proposed by [28], while it is limited in generalization to unseen optimizees. Latter, several efforts are made to empower the generalization ability of L2O. Specifically, [30,25] propose regularizers such as random

[3] We also include GPT-2 results in the supplementary.

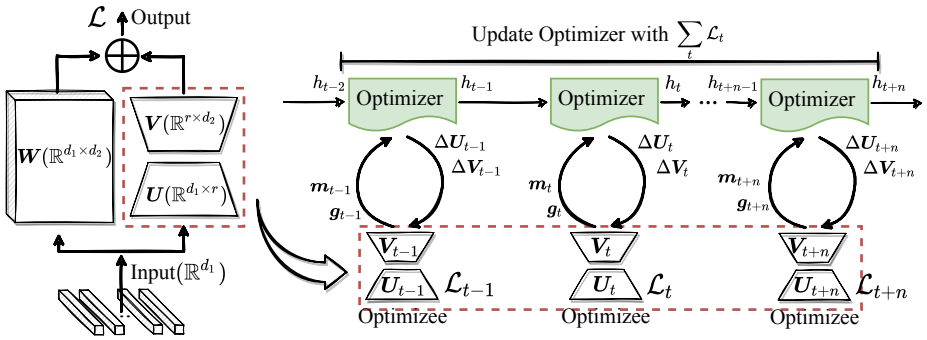

Fig. 2: The framework overview of our proposed scalable learning to optimize (SL2O). The network updates are constrained within certain intrinsic tiny subspaces, i.e., $\boldsymbol{U}$ and $\boldsymbol{V}$. Note that $\boldsymbol{W}$ can be the concatenated weight matrix of CNN or a single transformer layer.

scaling, objective convexifying, and Jacobian constraint; [50] designs a more sophisticated hierarchical recurrent neural networks (RNN) as an L2O; [6] utilizes advanced training techniques like curriculum learning and imitation learning; [31,47] constructs unbiased gradient estimators to learn enhanced optimizer. Recently, a survey paper of L2O [5] summarizes and benchmarks most of the achievements in this field.

3 Methods

3.1 Preliminaries

Tiny Subspace of Network Updates. The parameters inside a neural network are strongly correlated in multiple ways. For instance, the gradient back-propagation [18] process will relate gradients in different layers so that the parameters are also related. Therefore, it is possible to reduce the number of *independent* variables in a neural network. Recent studies [26,15,20] have pointed out that deep neural networks (DNN) can be trained in a tiny subspace. One can first identify a fixed number of "basis" vectors and then only optimize the *coefficient* of these basis vectors to get a sufficiently well-trained DNN. Generally speaking, a set of basis vectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N\}$ is calculated for a network (whose initialization is $\boldsymbol{W}_0$), and the optimization goal is converted to learning a coefficient vector $\boldsymbol{w} \in \mathbb{R}^N$ for weighing the bases. To derive the network's weights $\boldsymbol{W}$ from $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N\}$ and $\boldsymbol{w}$, a simple multiply-and-sum method would suffice: $\boldsymbol{W} = \boldsymbol{W}_0 - \sum_{i=1}^{N} w_i \boldsymbol{u}_i$. Therefore, the network is updated in the subspace spanned by the bases $\{\boldsymbol{u}_i\}_{i=1}^N$. Researchers have taken various methods to construct the bases and operate subspace training. [26,15] randomly generated these bases, and optimizing in the subspace spanned by these random bases can achieve surprisingly sufficient performance (over 90% of the full training accuracy). More recently, [29] sampled weights from the model training trajectory

and performed spectral decomposition to derive orthogonal bases. Training in the subspace spanned by these orthogonal bases can match or even surpass the performance of optimizing all parameters in the model. [20] performed subspace training in a more refined layer-wise and efficient manner. The basis vectors are randomly initialized and can be optimized during training. This method has demonstrated its effectiveness on various language tasks, reaching on-par or even higher testing performance.

Learning to Optimize. In Learning-to-Optimize (L2O), an *optimization task* is optimizing a network $f(\cdot;\boldsymbol{\theta})$, which we call *optimizee*, over a dataset. $\boldsymbol{\theta}$ is the weights of the optimizee. The goal of L2O is to learn an *optimizer* for solving tasks from a *task distribution* $\mathcal{F}$, *i.e.*, a set of similar optimization tasks. For example, $\mathcal{F}$ can be {Optimizing ResNet-20 on CIFAR-10}. Such a learned optimizer $\texttt{opt}$, parameterized by $\boldsymbol{\phi}$, predicts the update for optimizee's weights as $\texttt{opt}(\boldsymbol{z}_t;\boldsymbol{\phi})$. In literature, $\texttt{opt}$ is usually modeled by a neural network. $\boldsymbol{z}_t$ is a vector containing observations of historical training dynamics accessible at step t, such as the values and the gradients of $\boldsymbol{\theta}_t$. The optimizee's weights $\boldsymbol{\theta}$ are updated by $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \texttt{opt}(\boldsymbol{z}_t;\boldsymbol{\phi})$.

Learning an optimal update rule for an optimization task is equivalent to finding optimal optimizer weights $\boldsymbol{\phi}$. A direct approach is to minimize the weighted sum of the optimizee's objectives over a time interval T (i.e., *unroll length*):

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}\left[\sum_{t=1}^{T} \omega_t \mathcal{L}_t(f(\boldsymbol{x}_t;\boldsymbol{\theta}_t), y_t)\right], \text{ with } \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \texttt{opt}(\boldsymbol{z}_t;\boldsymbol{\phi}),$$

where $(\boldsymbol{x}_t, y_t)$ are training samples and $\mathcal{L}_t(\cdot,\cdot)$ is the function for calculating the optimizee's objective, such as cross-entropy loss or mean-squared error. We set $\omega_t = 1, t = 1, 2, \ldots, T$ to assign equal importance to all training steps, and obtain $\boldsymbol{\phi}$ by minimizing $\mathcal{L}$. Note that $\boldsymbol{\phi}$ is correlated with $\mathcal{L}_t(f(\boldsymbol{x}_t;\boldsymbol{\theta}_t), y_t)$ since $\boldsymbol{\phi}$ partially determines the optimizee's weights $\boldsymbol{\theta}_t$.

Typically, the pipeline of L2O can be split into two stages: *meta-training* where $\boldsymbol{\phi}$ is being optimized on tasks from the distribution $\mathcal{F}$, and *meta-testing* where $\boldsymbol{\phi}$ is fixed and the learned optimizer g is used to optimize a new optimization task. The meta-training stage is often done in an offline fashion since it requires time-consuming algorithms like truncated back-propagation through time [49]. However, the meta-training cost can be easily amortized at the meta-testing stage, which is expected to have a faster convergence speed.

3.2 Scalable Learning to Optimize

Combining Subspace Training with L2O. Previous L2O techniques directly predict updates for all the parameters, so most works can only perform meta-trainings on small-scale networks and require a large number of computational resources. In contrast, SL2O leverages the subspace training technique to reduce the number of independent parameters, and scale up to large models such as

VGG-16 and ViT. For CNNs having d parameters, we derive the orthogonal basis matrix $P \in \mathbb{R}^{r \times d}$ by performing SVD matrix decomposition and set $\boldsymbol{\theta} \in \mathbb{R}^{1 \times r}$ as the coefficients for P, where r is the number of orthogonal basis vectors. Consequently, the weights of a CNN can be represented by $\boldsymbol{W} := \boldsymbol{W}_0 - \boldsymbol{\theta} P$. We do not directly embed $\boldsymbol{\theta}$ in CNNs and consider it as a *virtual* parameter. The gradient on $\boldsymbol{\theta}$ can be calculated by the following formula:

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{\theta}} = \nabla_{\boldsymbol{W}} \mathcal{L}_t \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{\theta}} = (\nabla_{\boldsymbol{W}} \mathcal{L}_t) P^T,$$

suggesting that we only need a projection operation to construct the gradients. The predicted update for $\boldsymbol{\theta}$ from SL2O will also be projected back to update $\boldsymbol{W}$. By using such an indirect interacting method, we need not to store the value of $\boldsymbol{\theta}_t$ and save more memory. By default we set the number of independent trainable variables r to 40, which is negligible compared to d.

For transformer-based models (ViT, DeiT, and GPT-2), we use a slightly different subspace method for transformer-based models since performing SVD decomposition at such scales requires enormous memory. Alternatively, we seek a more refined layerwise we decompose the update of weights $\Delta \boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ into two matrices $\boldsymbol{U} \in \mathbb{R}^{d_1 \times r}$ and $\boldsymbol{V} \in \mathbb{R}^{r \times d_2}$, and constrain $\Delta \boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}$. $\boldsymbol{V}$ can be seen as a learnable and **shared** basis vectors since different rows in $\Delta \boldsymbol{W}$ are all linear combinations of row vectors in $\boldsymbol{V}$. For different layers in a network, we learn different decomposition matrices so that the subspace structures are more fine-grained. We explicitly embed $\boldsymbol{U}$ and $\boldsymbol{V}$ in the optimizee, and let $\boldsymbol{\theta} = \{\boldsymbol{U}, \boldsymbol{V}\}$ so that they would be the target parameters of SL2O. In this case we set r to be 16; the number of trainable parameters in $\boldsymbol{U}$ and $\boldsymbol{V}$ is $(d_1 + d_2) \times r$, which is significantly smaller than $d_1 \times d_2$ if r is small. We follow [20] to only apply the re-parameterization on attention weights.

The meta-training and meta-testing stage of SL2O optimizers. To train our SL2O optimizer, we sample multiple optimization tasks from a task distribution and train the optimizer on each task for a certain number of iterations. Previous works [4,6,46] also followed the same pipeline. For each task, the orders of training data are different so that the optimizer will not memorize the data order [52]. Following [30], we use the scaled gradients and their momentum as the observations of training dynamics. We limit the number of training steps to be N for each optimization task, and we split the whole training sequence (N training steps) into sub-sequences of length T. The value of N is set as 1000 so the effort of training an L2O optimizer is small and can be easily amortized. The unroll length T is fixed to be 10, and we will demonstrate that the choice of T does not have a dominating effect in Section 4.5. We evaluate the model with the updated parameters $\boldsymbol{\theta}_N$ on the testing set $\mathcal{D}_{\text{valid}}$, and choose the best optimizer parameters according to the testing performance. A detailed algorithm for the meta-training stage is shown in Algorithm 1.

After we obtain the best parameters of the learned optimizers, we switch to the meta-testing stage and use the learned optimizer to train a new optimization

Algorithm 1 The general training pipeline of SL2O.

Input: optimizee $f(\cdot;\cdot)$, optimizer $opt(\cdot;\cdot)$, current training step t, initial optimizee weights $\boldsymbol{\theta}_0$, optimizer weights $\boldsymbol{\phi}_t$, optimizee's objective $\mathcal{L}_t(\cdot,\cdot)$, a training set $\mathcal{D}_{\text{train}}$, a testing set $\mathcal{D}_{\text{test}}$, coefficients β_1 and β_2, unroll length T, training steps N for each epoch, and number of epochs E
Output: Optimal optimizer weights $\boldsymbol{\phi}$
for epoch $< E$ **do**
  Initialize $\boldsymbol{m} = \mathbf{0}$ and $\boldsymbol{v} = \mathbf{0}$
  for i=$0, T, 2T \ldots, ([N/T]-1) \times T$ **do** ▷ $[N/T]$ sub-sequences
    Set $\mathcal{L} \leftarrow 0$
    for j=$0, 1, 2 \ldots, T-1$ **do**
      $t := i + j$
      Sample a batch $\mathcal{B}$ from the training set $\mathcal{D}_{\text{train}}$
      Calculate the training loss on $\mathcal{B}$: $\mathbb{E}_{(\boldsymbol{x},y)\in\mathcal{B}}\mathcal{L}_t(f(\boldsymbol{x};\boldsymbol{\theta}_t),y)$ and the gradient on $\boldsymbol{\theta}_t$: $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}}\mathbb{E}_{(\boldsymbol{x},y)\in\mathcal{B}}\mathcal{L}_t(f(\boldsymbol{x};\boldsymbol{\theta}_t),y)$
      Update $\boldsymbol{m} \leftarrow \beta_1\boldsymbol{m} + (1-\beta_1)\boldsymbol{g}_t$ and $\boldsymbol{v} \leftarrow \beta_2\boldsymbol{v} + (1-\beta_2)\boldsymbol{g}_t^2$
      Calculate $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m}/(1-\beta_1^{t+1})$, $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}/(1-\beta_2^{t+1})$
      Calculate $\tilde{\boldsymbol{g}} \leftarrow \boldsymbol{g}_t/\sqrt{\hat{\boldsymbol{v}}+\epsilon}$, $\tilde{\boldsymbol{m}} \leftarrow \hat{\boldsymbol{m}}/\sqrt{\hat{\boldsymbol{v}}+\epsilon}$ ▷ features calculation
      Construct $\boldsymbol{z}_t$ from z_t from $\tilde{\boldsymbol{g}}$ and $\tilde{\boldsymbol{m}}$
      Update $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \texttt{opt}(\boldsymbol{z}_t;\boldsymbol{\phi})$ ▷ Update the optimizee's weights
      $\mathcal{L} \leftarrow \mathcal{L} + \mathbb{E}_{(\boldsymbol{x},y)\in\mathcal{B}}\mathcal{L}_t(f(\boldsymbol{x};\boldsymbol{\theta}_t),y)$ ▷ Loss calculation
    Update $\boldsymbol{\phi}$ by minimizing $\mathcal{L}$ using gradient descent-based methods for one step
  Evaluate $f(\cdot;\boldsymbol{\theta}_N)$ on $\mathcal{D}_{\text{test}}$ and find the optimal $\boldsymbol{\phi}$

task. Note that during this meta-testing stage we will fix $\boldsymbol{\phi}$. The learned optimizer receives the same set of observations of the optimizee's training dynamics and predicts updates for $\boldsymbol{\theta}$ at every training step t.

4 Experiments

4.1 Implementation Details

Architectures and Datasets. We study two sets of networks: (*i*) CNNs, including ResNet-20 [17], ResNet-18 [17], and VGG-16 [39]; (*ii*) Transformer-based models [45], including ViT [12], DeiT [43] (and in supplementary, GPT-2 [35]). We study four datasets: CIFAR-10 [24], CIFAR-100 [24], and ImageNet [11], and E2E [14] dataset for the GPT-2 experiments. We also study a small network ResNet-8 (results deferred to the supplementary files).

Baseline Optimizers. We choose several widely used optimizers as the baselines for comparison: Stochastic Gradient Descent (SGD) [36], Momentum [34], Adam [23], and RMSProp [42]. The optimizer settings are reported in Table 1 in the supplementary file. We also apply two of them (SGD and Momentum) with the subspace training techniques, which we call "SGD†" and "Momentum†" (the † superscription means subspace training).

Table 1: Comparison of testing accuracy using different optimizers with four network architectures on CIFAR-10. We report both the average and the confidence interval of the best testing accuracy. The superscription † means that the model is updated in a tiny subspace.

Optimizer	Testing Accuracy			Convergence Steps		
	ResNet-20	ResNet-18	VGG-16	ResNet-20	ResNet-18	VGG-16
SGD	$88.72_{\pm 0.09}\%$	$93.48_{\pm 0.15}\%$	$92.81_{\pm 0.22}\%$	5073.8	2351.2	1250.4
Momentum	$91.27_{\pm 0.30}\%$	$93.90_{\pm 0.37}\%$	$92.96_{\pm 0.13}\%$	3280.8	4316.8	2972.0
Adam	$90.19_{\pm 0.21}\%$	$93.54_{\pm 0.13}\%$	$92.06_{\pm 0.29}\%$	3554.0	2688.0	6767.0
RMSProp	$90.16_{\pm 0.20}\%$	$93.01_{\pm 0.22}\%$	$86.26_{\pm 0.29}\%$	1776.2	1856.4	13189.4
SGD†	$90.61_{\pm 0.08}\%$	$92.99_{\pm 0.08}\%$	$89.03_{\pm 0.26}\%$	1468.0	1318.2	2315.8
Momentum†	$90.97_{\pm 0.03}\%$	$93.70_{\pm 0.01}\%$	$92.77_{\pm 0.05}\%$	333.0	216.8	383.4
SL2O	$91.00_{\pm 0.05}\%$	$93.61_{\pm 0.02}\%$	$92.74_{\pm 0.03}\%$	90.2	63.6	57.4

Metrics. On image datasets (CIFAR-10, CIFAR-100, and ImageNet), we use the accuracy on testing set and the first convergence step of training as our metrics. We say the training is converged at step t if $\text{std}(\mathcal{L}_{t-19}, \ldots, \mathcal{L}_t) < 0.1$ and $\mathcal{L}_t < \mathcal{L}^* + 0.1$, where $\text{std}(\cdot)$ is the standard deviation, $\mathcal{L}_t$ is the training loss at step t and $\mathcal{L}^*$ is the globally minimal training loss. On the language dataset E2E, we report the validation loss. Note that most existing works on L2O evaluate their methods **only by reporting training losses**; we take one more step and also take the testing loss/accuracy into consideration.

Meta-Training and Meta-Testing Settings. We meta-train the optimizer on 20 sampled optimization tasks (i.e., 20 epochs) from every task distribution. We use Adam with a learning rate of 0.01 to train our optimizer, and maintain the same learning rate across the whole meta-training process. For the meta-testing stage, the numbers of training epochs for (ResNets, VGGs, ViT, DeiT) are by default (100, 100, 20, 20) respectively, and the training batch sizes are 128 for all experiments. For baseline optimizers, we share the same training epochs and batch sizes with our SL2O optimizer. The structure of our SL2O optimizer is based on an LSTM [19], and we will show the details of the architecture of our SL2O optimizer in the supplementary files. SL2O operates in a coordinate-wise fashion [9], which enables SL2O to optimizees with distinctive numbers of parameters. To be specific, for each parameter in the subspace, SL2O takes its observation vector as input and produces its update.

4.2 Superior Performance on ResNets

We first conduct experiments on CIFAR-10, and report the testing accuracy and the convergence steps in Table 1. We can draw several conclusions from these tables: 1) On CIFAR-10, Momentum is the overall best optimizer for all

Table 2: Comparison of testing accuracy using different optimizers with four network architectures on CIFAR-100. We report both the average and the confidence interval of the best testing accuracy. The superscription † means that the model is updated in a tiny subspace.

Optimizer	Testing Accuracy			Convergence Steps		
	ResNet-20	ResNet-18	VGG-16	ResNet-20	ResNet-18	VGG-16
SGD	$63.18_{\pm 0.40}\%$	$73.96_{\pm 0.10}\%$	$70.43_{\pm 0.14}\%$	20286.6	6353.0	11558.6
Momentum	$67.29_{\pm 0.36}\%$	$73.92_{\pm 0.41}\%$	$70.61_{\pm 0.28}\%$	26242.2	6860.6	19693.4
Adam	$64.59_{\pm 0.39}\%$	$73.76_{\pm 0.23}\%$	$66.05_{\pm 0.20}\%$	18397.6	7947.0	19828.8
RMSProp	$63.53_{\pm 0.56}\%$	$65.91_{\pm 0.57}\%$	$14.62_{\pm 6.84}\%$	15459.6	6707.8	15589.0
SGD†	$66.64_{\pm 0.18}\%$	$74.74_{\pm 0.05}\%$	$59.96_{\pm 0.95}\%$	4420.4	1787.2	10198.8
Momentum†	$66.89_{\pm 0.04}\%$	$74.76_{\pm 0.06}\%$	$70.98_{\pm 0.04}\%$	636.2	283.2	2104.8
SL2O	$67.04_{\pm 0.07}\%$	$74.67_{\pm 0.14}\%$	$71.02_{\pm 0.09}\%$	105.4	61.6	168.2

architectures regarding the testing accuracy. However, it needs over 1000 training steps to reach convergence; 2) Momentum† and SL2O achieve nearly the same level of testing accuracy, and their performances are both comparable with the performance of Momentum; 3) The convergence speeds of SL2O are significantly faster than all other baselines. On {ResNet-20, ResNet-18, VGG-16}, our method can achieve convergence with {90.2,63.6,57.4} training steps on average, while the best analytical baseline optimizer Momentum† needs {333.0,216.8,383.4} training steps for convergence on average, which is {3.69,3.41,6.68} times slower. We further show the training loss and testing accuracy in the first 10000 steps in Figure 3. The convergence speeds of the L2O optimizer surpass the convergence speeds of other baselines: SL2O achieves nearly full accuracy after hundreds of training iterations, and the speed of training loss decreasing is the highest among all baseline methods. Moreover, the advantage on convergence speed of SL2O compared to other methods on CIFAR-10 seems to be bigger as the number of parameters gets larger, manifested by the enlarging spaces between early training loss curves collected from ResNet-20 to VGG-16 models.

We continue to test our method on CIFAR-100 with ResNet-20, ResNet-18, and VGG-16, and we have drawn a similar conclusion from the experiments. The testing accuracy and the convergence steps are shown in Table 2. We can see that: 1) Among all optimization methods, our L2O optimizer has the fastest convergence speed: on {ResNet-20,ResNet-18,VGG-16}, SL2O can achieve convergence with only {105.4, 61.6, 168.2} steps on average, while the most competitive baseline Momentum† needs {636.2, 283.2, 2104.8} steps, which is {6.04, 4.60, 12.51} times more, respectively; 2) Compared to CIFAR-10, SL2O performs much better on CIFAR-100 in terms of testing accuracy, achieving superior performance on some cases compared to other analytical optimizers, with or without the subspace reparameterization technique. Specifically, we achieve higher testing accuracy on ResNet-20 and VGG-16 compared to Momentum†,

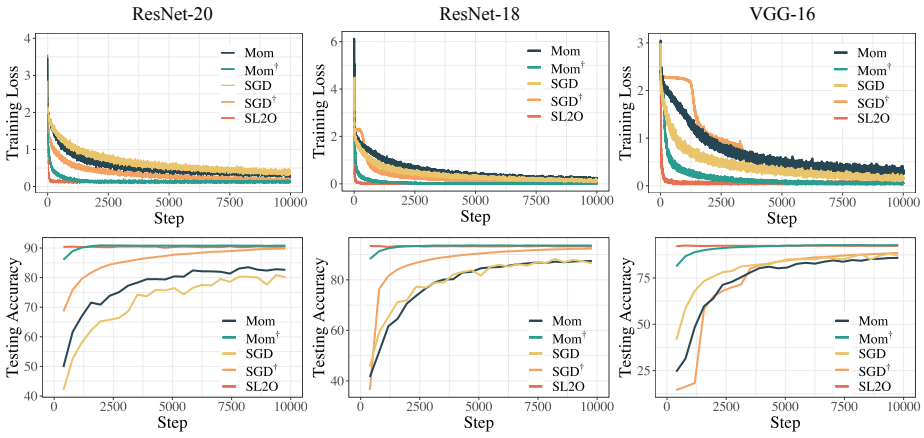

Fig. 3: Comparison of training loss and testing accuracy on CIFAR-10 with various optimizers. Results of the first 10,000 training steps are presented. The superscription † means that the model is updated in a tiny subspace.

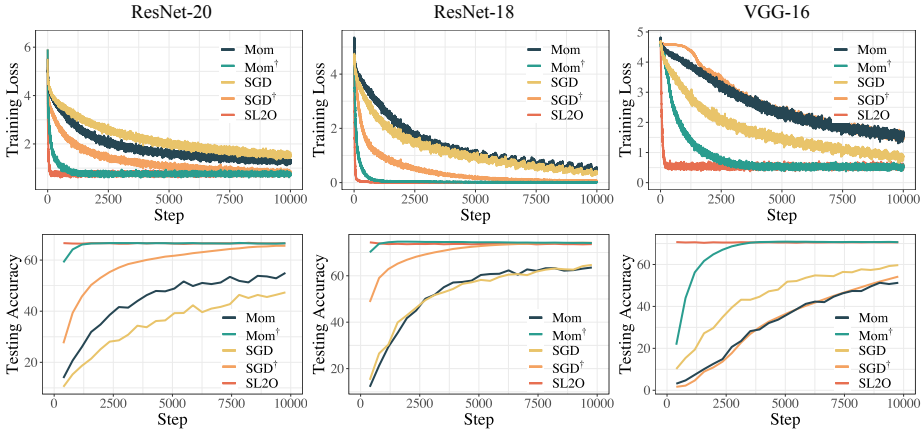

Fig. 4: Comparison of training loss and testing accuracy on CIFAR-10 with various optimizers. Results of the first 10,000 training steps are presented. The superscription † means that the model is updated in a tiny subspace.

although the confidence intervals overlap. From Figure 4 we can further validate the superiority of our SL2O method. SL2O achieves high accuracy with only hundreds of training steps, and the gaps between early training loss curves are more significant as the model sizes get larger.

Finally, we validate our SL2O optimizer on a large-scale dataset, *i.e.,* ImageNet, with ResNet-18. We present the training loss in the first 11000 steps and the best testing accuracy after training for 40 epochs in Figure 5. We can see

from the figure that SL2O significantly outperforms other methods regarding the convergence speed since the loss curve for SL2O becomes stable in less than 1000 training steps while the most competitive baseline (Mom†) needs around 5000 training steps, which is approximately 5 times faster. After training for 40 epochs, the testing accuracy of ResNet-18 optimized with SL2O (68.78%) is comparable to SGD† (68.86%) and Momentum† (68.83%), only with slight accuracy loss. In summary, our SL2O optimizer is capable of achieving convergence faster than all analytical baseline optimizers and comparable performance (testing accuracy) on various CNN optimizees.

4.3 Superior Performance on Transformer-based models

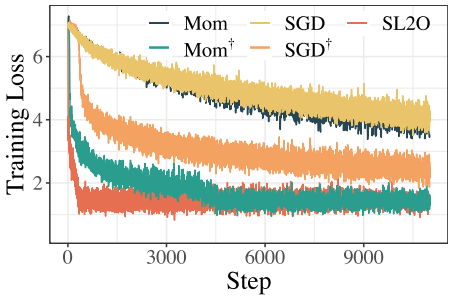

Fig. 5: Training loss on ImageNet with ResNet-18.

We deploy the SL2O optimizer to finetune two transformer-based vision models, ViT and DeiT, on CIFAR-10. To be more concrete, we use the official ViT-B/16 ($\sim$ 83M parameters) pretrained on ImageNet and the DeiT-tiny-distilled ($\sim$ 8M parameters) pretrained as the starting points for fine-tuning. The number of the fine-tuning epoch is set to 10 since we observe no improvement afterward. Figure 6 shows the performance of optimizing with Mom, Mom† with different learning rates, and the performance of SL2O on ViT-B/16. We can see from the figure that SL2O achieves lower training loss and fastest convergence while preserving high testing accuracy. Numerically, SL2O hits a testing accuracy of 98.5% with only two epochs and consistently outperforms other baselines with subspace training. Training by using SL2O converges at the 484-th step while $\text{Mom}^\dagger_{0.04}$ needs 602 steps, which is 1.24 times longer. SL2O eventually gets surpassed by the analytical optimizer Momentum; however, it is still an efficient optimizer and demonstrates the effectiveness of SL2O on optimizees at the level of 10^8 parameters. It is noteworthy that almost all existing L2O approaches benefit more at the early stage rather than the final performance [30], which is a fundamental yet unsolved problem in L2O, *i.e.*, meta-testing with a "longer horizon" [30]. Note that the previous best L2O is capable of maintaining superior meta-testing performance for $\sim$ 4k iterations on small MLPs, while our `SL2O` enables an improved "horizon" like 5k iterations in the ViT training. Lastly, we offer a simple remedy to this challenging problem - leveraging both learned and analytical optimizers: (*i*) apply `SL2O` in the early stage such as the first 5k steps then (*ii*) switch to the analytical optimizer, and we obtain an extra 0.12% accuracy boost and match `Mom` in the late stage. We have observed

similar results and validness on DeiT; therefore, we defer these results to the supplementary files.

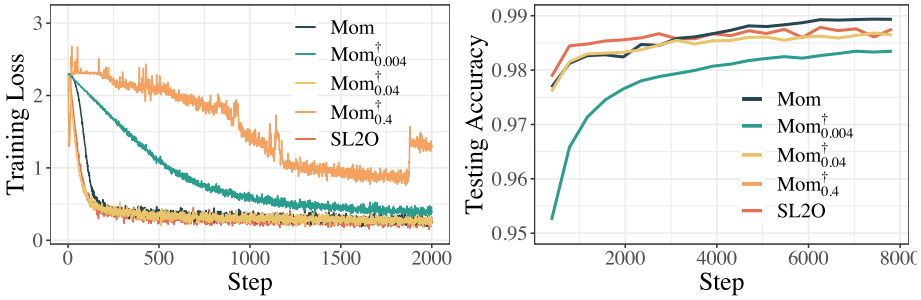

Fig. 6: Training loss and testing accuracy of ViT-B/16 on CIFAR-10. We report the training loss in the first 2000 steps and the testing accuracy in the first 8000 steps. The superscription † means that the model is updated in a tiny subspace, and the subscription means the initial learning rate of the optimizers. $\text{Mom}^{\dagger}_{0.4}$ does not appear in the figure of testing accuracy since the accuracies are below the lower bar of 0.95.

4.4 The Transferability of SL2O

We conduct two experiments to study the transferability of SL2O between different task distributions $\mathcal{F}$: using an SL2O optimizer meta-trained on ResNet-20 to optimize ResNet-18, and one meta-trained on ResNet-18 to optimize ResNet-20. We empirically prove that SL2O is **transferable** between optimizee's architectures in Figure 7. The left figure shows the training loss curves on ResNet-18, and the right figure shows the curves on ResNet-20. ResNet-20→ResNet-18 means that the transferred SL2O optimizer is trained on ResNet-20, ResNet-18→ResNet-20 means the opposite. We can see from the figures that: 1) SL2O meta-trained on ResNet-18 can optimize ResNet-20 well, and vice versa; 2) When optimizing ResNet-20, the SL2O optimizer meta-trained on ResNet-18 can even out-perform the optimizer meta-trained on ResNet-20 at the early training stage. This may suggest that the L2O models meta-trained on more complicated problems (e.g., ResNet-18) can solve the easier problems (e.g., ResNet-20) more efficiently. Table 3 shows the average best testing accuracy of different combinations of meta-training/meta-testing schemes. We can see that meta-training and meta-testing on the same task distribution yield the best performance while transferring the trained optimizers only brings slight performance loss. These promising results show that the meta-training cost of SL2O can potentially be further amortized by training fewer optimizers and sharing between different optimization optimizees. We also demonstrate the transferability of SL2O between

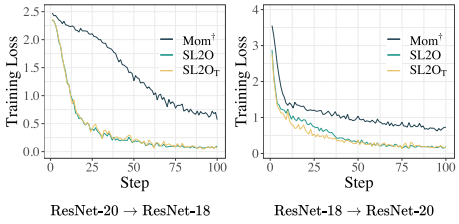

Fig. 7: Training loss of optimizing ResNet-18 (left) and ResNet-20 (right) on CIFAR-10. SL2O$_T$ means the optimizer is meta-trained on another architecture.

Table 3: The average of best testing accuracy with different combinations of meta-training and meta-testing datasets.

Train on \ Test on	ResNet-20	ResNet-18
ResNet-20	91.00%	93.59%
ResNet-18	90.96%	93.61%

various datasets in the supplementary file, which suggests the SL2O trained on one dataset can be seamlessly generalized across multiple datasets.

4.5 Ablation Study and Visualizations

The effects of unroll length and different training iterations. We study the effects of different unroll lengths T and training iterations N in the supplementary files. We have validated: 1) the unroll length T does not have a dominating effect; 2) longer unroll lengths do not bring extra performance gain, while a smaller unroll length would result in slightly weak performance; 3) shorter training iterations probably make SL2O overfit to shorter training horizons. Detailed analysis are provided in the supplementary files.

Learned update rules. To understand how our L2O models generate update rules for models, we provide two sets of visualizations on ResNet-20. The first visualization is the training trajectory and the loss landscapes [27] of models trained by different optimizers. We use three optimizers, *i.e.* Momentum, Momentum† and SL2O, to train ResNet-20 on CIFAR-10 from the same initialization weights. We focus on the first 100 training steps to demonstrate how optimizers behave at the early training stage. The endpoints of the training trajectory are set to be $(0, 0)$. The three landscapes are presented in Figure 8. We can draw multiple conclusions from these figures: 1) The training trajectories are similar for the three optimizers. We interpret such a similarity as a successful sanity check demonstrating the SL2O optimizer has learned valid optimization rules since it shares a similar optimization pathway with Momentum and Momentum†; 2) SL2O arrives at a lower loss region compared to Momentum† and Momentum after 100 training steps, again showing that SL2O can be more efficient at reducing training loss; 3) SL2O reaches a region where the loss landscape is flatter, manifested by sparser contour lines around the endpoint $(0, 0)$ in the loss landscape of SL2O. The flatness of loss landscapes is believed to measure the generalization ability of neural networks [55]; therefore, the model optimized by SL2O

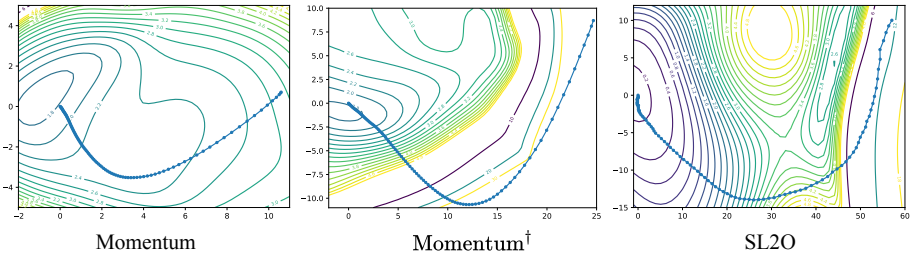

Fig. 8: The visualization of loss landscapes and training trajectories of ResNet-20 models optimized by Momentum, Momentum†, and SL2O, respectively.

has potentially higher generalization ability and better quality compared to the Momentum† optimized model after 100 training steps. The second visualization is what SL2O outputs. We provide the visualization of Momentum† and SL2O on ResNet-20 in the supplementary files, showing that SL2O learns sophisticated update rules rather than always optimizing in the largest principal direction.

Explainability. We use a classical tool, symbolic regression [10], to uncover a mathematical formula from the fitted optimizers. The symbolic regression (SR) will search within a space of mathematical formulas, and find one that best describes the numerical rules learned by SL2O. Different from normal regression techniques, the symbolic regression does not need pre-defined regression structures (e.g., $y = ax + b$) but finds reasonable structures automatically from the space. Such a property makes SR more flexible and explainable. The results are deferred to the supplementary files due to space constraints.

5 Discussion and Conclusions

In this paper, we challenge the "common experience" that learnable optimizers can only be meta-trained on extremely shallow networks without massive computation resources, which severely limits their piratical usage in real-world scenarios. We remove this primary barrier, scalability, for L2O by first projecting the network updates in a tiny subspace and then learning optimization rules on top of it. In this way, we for the first time enable the L2O training on large-scale models including Vision Transformer and GPT. Meanwhile, our scalable L2O has demonstrated superior convergence and comparable or even better testing performance. In future work, we would be interested in examining more about the transferability of SL2O to better amortize the meta-training cost of SL2O optimizers. We would be interested in combining L2O optimizers with analytical optimizers to see if we can further improve the convergence speed and testing performance. Exploring more efficient reparameterization techniques is also one of our future goals. As for the social impact, we see our proposals substantially reduce the computation cost of data-driven L2O training, which offers energy- and financial-saving solution.

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 (NeurIPS) (2016)
2. Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NeurIPS). vol. 29. Curran Associates, Inc. (2016)
3. Bello, I., Zoph, B., Vasudevan, V., Le, Q.V.: Neural optimizer search with reinforcement learning. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 459–468. PMLR (06–11 Aug 2017), `https://proceedings.mlr.press/v70/bello17a.html`
4. Cao, Y., Chen, T., Wang, Z., Shen, Y.: Learning to optimize in swarms. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 15018–15028 (2019)
5. Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., Yin, W.: Learning to optimize: A primer and a benchmark. arXiv preprint arXiv:2103.12828 (2021)
6. Chen, T., Zhang, W., Zhou, J., Chang, S., Liu, S., Amini, L., Wang, Z.: Training stronger baselines for learning to optimize. arXiv preprint arXiv:2010.09089 (2020)
7. Chen, W., Yu, Z., Wang, Z., Anandkumar, A.: Automated synthetic-to-real generalization. In: International Conference on Machine Learning (ICML). pp. 1746–1756 (2020)
8. Chen, X., Chen, W., Chen, T., Yuan, Y., Gong, C., Chen, K., Wang, Z.: Self-pu: Self boosted and calibrated positive-unlabeled training. In: International Conference on Machine Learning (ICML). pp. 1510–1519 (2020)
9. Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., De Freitas, N.: Learning to learn without gradient descent by gradient descent. In: International Conference on Machine Learning (ICML). pp. 748–756 (2017)
10. Cranmer, M.: Pysr: Fast & parallelized symbolic regression in python/julia (2020)
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
12. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
13. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(7) (2011)
14. Dušek, O., Howcroft, D.M., Rieser, V.: Semantic noise matters for neural natural language generation. In: Proc. of the 12th International Conference on Natural Language Generation. pp. 421–426. Association for Computational Linguistics, Tokyo, Japan (Oct–Nov 2019). https://doi.org/10.18653/v1/W19-8652, `https://www.aclweb.org/anthology/W19-8652`
15. Gressmann, F., Eaton-Rosen, Z., Luschi, C.: Improving neural network training in low dimensional random bases. arXiv preprint arXiv:2011.04720 (2020)

16. Gur-Ari, G., Roberts, D.A., Dyer, E.: Gradient descent happens in a tiny subspace. arXiv preprint arXiv:1812.04754 (2018)
17. 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)
18. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
20. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Chen, W.: Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685 (2021)
21. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: Proceedings of the British Machine Vision Conference. BMVA Press (2014)
22. Jiang, H., Chen, Z., Shi, Y., Dai, B., Zhao, T.: Learning to defense by learning to attack. arXiv preprint arXiv:1811.01213 (2018)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
25. Li, C., Chen, T., You, H., Wang, Z., Lin, Y.: Halo: Hardware-aware learning to optimize. In: European Conference on Computer Vision (ECCV). pp. 500–518. Springer (2020)
26. Li, C., Farkhoor, H., Liu, R., Yosinski, J.: Measuring the intrinsic dimension of objective landscapes. arXiv preprint arXiv:1804.08838 (2018)
27. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. Advances in neural information processing systems **31** (2018)
28. Li, K., Malik, J.: Learning to optimize. arXiv preprint arXiv:1606.01885 (2016)
29. Li, T., Tan, L., Tao, Q., Liu, Y., Huang, X.: Low dimensional landscape hypothesis is true: Dnns can be trained in tiny subspaces (2021)
30. Lv, K., Jiang, S., Li, J.: Learning gradient descent: Better generalization and longer horizons. In: International Conference on Machine Learning (ICML). pp. 2247–2255 (2017)
31. Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., Sohl-Dickstein, J.: Understanding and correcting pathologies in the training of learned optimizers. In: International Conference on Machine Learning. pp. 4556–4565. PMLR (2019)
32. Oymak, S., Fabian, Z., Li, M., Soltanolkotabi, M.: Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. arXiv preprint arXiv:1906.05392 (2019)
33. Povey, D., Cheng, G., Wang, Y., Li, K., Xu, H., Yarmohammadi, M., Khudanpur, S.: Semi-orthogonal low-rank matrix factorization for deep neural networks. In: Interspeech. pp. 3743–3747 (2018)
34. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural networks **12**(1), 145–151 (1999)
35. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)
36. Robbins, H.E.: A stochastic approximation method. Annals of Mathematical Statistics **22**, 400–407 (2007)

37. Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE international conference on acoustics, speech and signal processing. pp. 6655–6659. IEEE (2013)
38. Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z.: Learning a minimax optimizer: A pilot study. In: International Conference on Learning Representations (ICLR) (2021)
39. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
40. Sohl-Dickstein, J., Poole, B., Ganguli, S.: Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In: International Conference on Machine Learning. pp. 604–612. PMLR (2014)
41. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
42. Tieleman, T., Hinton, G.: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning (2012)
43. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: International Conference on Machine Learning. pp. 10347–10357. PMLR (2021)
44. Tuddenham, M., Prügel-Bennett, A., Hare, J.: Quasi-newton's method in the class gradient defined high-curvature subspace. arXiv preprint arXiv:2012.01938 (2020)
45. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
46. Vicol, P., Metz, L., Sohl-Dickstein, J.: Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In: International Conference on Machine Learning. pp. 10553–10563. PMLR (2021)
47. Vicol, P., Metz, L., Sohl-Dickstein, J.: Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 10553–10563. PMLR (18–24 Jul 2021), `https://proceedings.mlr.press/v139/vicol21a.html`
48. Wang, Z., Wohlwend, J., Lei, T.: Structured pruning of large language models. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 6151–6162 (2020)
49. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE **78**(10), 1550–1560 (1990)
50. 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: International Conference on Machine Learning (ICML) (2017)
51. Xiong, Y., Hsieh, C.J.: Improved adversarial training via learned optimizer (2020)
52. Yin, M., Tucker, G., Zhou, M., Levine, S., Finn, C.: Meta-learning without memorization. arXiv preprint arXiv:1912.03820 (2019)
53. You, Y., Chen, T., Wang, Z., Shen, Y.: L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2127–2135 (2020)

54. Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7370–7379 (2017)
55. Zhang, S., Wang, M., Liu, S., Chen, P.Y., Xiong, J.: Why lottery ticket wins? a theoretical perspective of sample complexity on sparse neural networks. Advances in Neural Information Processing Systems **34** (2021)
56. Zhang, Y., Chuangsuwanich, E., Glass, J.: Extracting deep neural network bottleneck features using low-rank matrix factorization. In: 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 185–189. IEEE (2014)
57. Zhao, Y., Li, J., Gong, Y.: Low-rank plus diagonal adaptation for deep neural networks. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5005–5009. IEEE (2016)