MLR-SNet: Transferable LR Schedules for Heterogeneous Tasks

Jun Shu, Yanwen Zhu, Qian Zhao, Deyu Meng, and Zongben Xu

Abstract—The learning rate (LR) is one of the most important hyperparameters in stochastic gradient descent (SGD) algorithm for training deep neural networks (DNN). However, current hand-designed LR schedules need to manually pre-specify a fixed form, which limits their ability to adapt to practical non-convex optimization problems due to the significant diversification of training dynamics. Meanwhile, it always needs to search proper LR schedules from scratch for new tasks, which, however, are often largely different with task variations, like data modalities, network architectures, or training data capacities. To address this learning-rate-schedule setting issue, we propose to parameterize LR schedules with an explicit mapping formulation, called *MLR-SNet*. The learnable parameterized structure brings more flexibility for MLR-SNet to learn a proper LR schedule to comply with the training dynamics of DNN. Image and text classification benchmark experiments substantiate the capability of our method for achieving proper LR schedules. Moreover, the explicit parameterized structure makes the meta-learned LR schedules capable of being transferable and plug-and-play, which can be easily generalized to new heterogeneous tasks. We transfer our meta-learned MLR-SNet to query tasks like different training epochs, network architectures, data modalities, dataset sizes from the training ones, and achieve comparable or even better performance compared with hand-designed LR schedules specifically designed for the query tasks. The robustness of MLR-SNet is also substantiated when the training data are biased with corrupted noise. We further prove the convergence of the SGD algorithm equipped with LR schedule produced by our MLR-SNet, with the convergence rate comparable to the best-known ones of the algorithm for solving the problem. The source code of our method is released at https://github.com/xjtushujun/MLR-SNet.

Index Terms—Meta learning, learning to learn, transferable to heterogeneous tasks, DNN training, stochastic gradient descent

1 Introduction

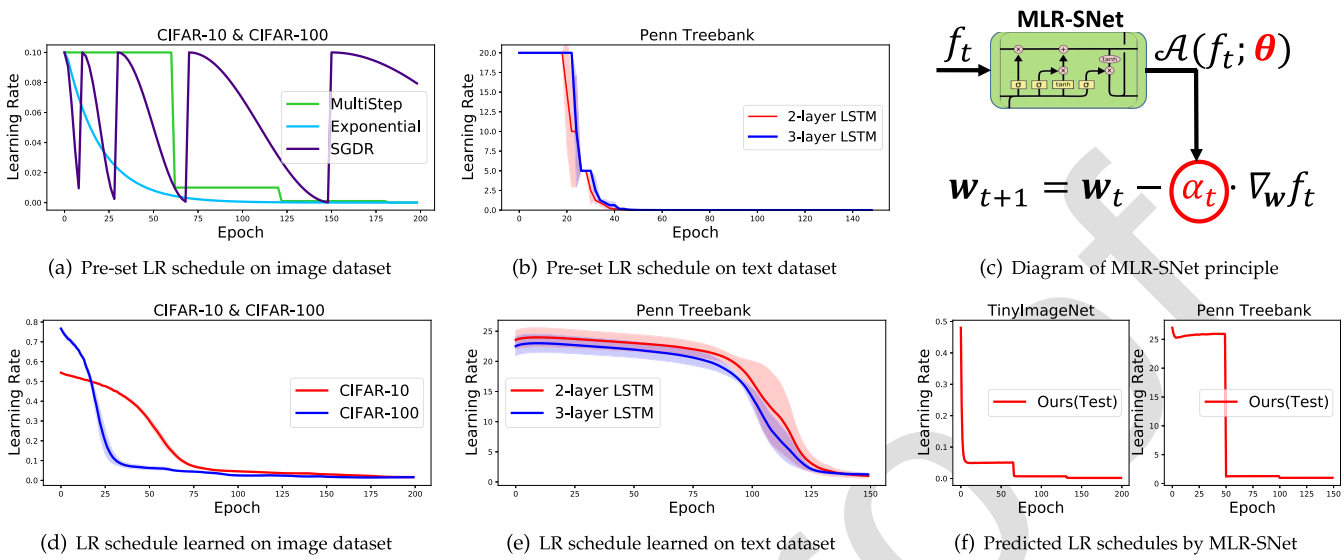
Stochastic gradient descent (SGD) and its many variants [3], [4], [5], [6], [7], have been served as the cornerstone of modern machine learning with big data. It has been empirically shown that DNNs achieve state-of-the-art generalization performance on a wide variety of tasks when trained with SGD [8]. Recent researches observe that SGD tends to select the so-called flat minima, which seems to generalize better in practice, partially explaining its underlying working mechanism [9], [10], [11], [12], [13], [14].

Scheduling learning rate (LR) for the SGD algorithm is one of the most widely studied aspects to help improve the training for DNNs. Specifically, it has been experimentally studied how the LR [15] essentially influences minima solutions found by SGD. This issue has also been investigated from a theoretical perspective. For example, Wu *et al.*, [12] theoretically analyzed that LR plays an important role in minima selection from a dynamical stability perspective. Furthermore, they used stochastic differential equations to prove that the higher the ratio of the LR to the batch size, the flatter minimum inclines to be selected. Besides, He *et al.*, [14] provided PAC-Bayes generalization bounds for DNN trained by SGD, which are highly correlated with LR. In summary, it is being more widely recognized that designing a proper LR schedule tends to highly influence the generalization performance of DNN training result [17], [18], [19], [20].

There mainly exist three kinds of hand-designed LR schedules: (1) Pre-defined LR schedule policies. Typical ones include decaying and cyclic LR [21], [22] (as depicted in Figs. 1a and 1b), with a good training efficiency in practice. This line of methods have been mostly used in current DNN training, and become the default setting across the current popular deep learning libraries like Pytorch [23]. Some theoretical works have further proved that the decaying schedule can yield faster convergence [24], [25] or avoid strict saddles [26], [27] under some mild conditions. (2) Adaptive gradient descent methods. Typical methods in this category include AdaGrad [4], RMSProp [6], and Adam [7], often using the

- *Jun Shu and Zongben Xu are with the School of Mathematics and Statistics and Ministry of Education Key Lab of Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China. E-mail: xjtushujun@gmail.com, zbxu@mail.xjtu.edu.cn.*
- *Yanwen Zhu and Qian Zhao are with the School of Mathematics and Statistics and Ministry of Education Key Lab of Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: zywwyz@stu.xjtu.edu.cn, timmy.zhaoqian@mail.xjtu.edu.cn.*
- *Deyu Meng is with the School of Mathematics and Statistics and Ministry of Education Key Lab of Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China, and with the Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China, and also with the Macau Institute of Systems Engineering, Macau University of Science and Technology, Taipa, Macau, 999078, China. E-mail: dymeng@mail.xjtu.edu.cn.*

Manuscript received 13 Apr. 2021; revised 23 Mar. 2022; accepted 11 June 2022. Date of publication 0 . 2022; date of current version 0 . 2022.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFA0713900 in part by the National Natural Science Foundation of China (NSFC) Project under Grants 62076196, 61721002, and U1811461, and in part by the Macao Science and Technology Development Fund under Grant 061/2020/A2, and in part by the Major Key Project of PCL under Grant PCL2021A12.
(Corresponding authors: Qian Zhao and Deyu Meng.)
Recommended for acceptance by R. Feris.
Digital Object Identifier no. 10.1109/TPAMI.2022.3184315



(a) Pre-set LR schedule on image dataset (b) Pre-set LR schedule on text dataset (c) Diagram of MLR-SNet principle

(d) LR schedule learned on image dataset (e) LR schedule learned on text dataset (f) Predicted LR schedules by MLR-SNet

Fig. 1. Pre-defined LR schedules used in our paper for (a) image and (b) text classification experiments. (c) Visualization of how we input current loss f_t to MLR-SNet, which then outputs a proper LR α_t to help SGD find a better minima. LR schedules meta-learned by the proposed MLR-SNet on (d) image and (e) text classification experiments (meta-training stage). (f) The predicted LR schedules, learned from CIFAR-10, on image (TinyImageNet) and text (Penn Treebank) classification datasets (meta-test stage).

adaptive LR for each model parameters based on some gradient information. (3) LR search methods. The main idea is to borrow LR search strategies, such as Polyak's update rule [28], Frank-Wolfe algorithm [29], and Armijo line-search [30], used traditional optimization approaches [31] to DNN training, by searching LR adaptively.

Although the above LR schedules can achieve competitive results on certain learning tasks, they still have evident deficiencies in practice. On the one hand, these policies need to manually pre-specify the formulation of the LR schedules, inevitably suffering from the limited flexibility to adapt to the complicated DNN optimization problems due to the significant variation of its training dynamics. On the other hand, when solving new heterogeneous tasks, it always needs to redesign proper LR schedules from scratch, as well as tune their involved hyperparameters. This process is often time and computation expensive, which tends to further raise their application difficulty in real problems.

To alleviate the aforementioned issues, this paper aims to develop a model to learn a plug-and-play LR schedule under the meta-learning framework. The main idea is to parameterize the LR schedule as an LSTM network [32], which is capable of dealing with such a long-term information dependent problem. As shown in Fig. 1c, with a parameterized structure, the proposed model has the capacity to fit an explicit loss-LR dependent relationship to adapt to the complicated training dynamics. We learn the LSTM network from data in a meta-learning manner, which is able to adaptively predict the LR schedule for an SGD algorithm to help improve the DNN training performance. We call this method Meta-LR-Schedule-Net (*MLR-SNet* for brevity). Meanwhile, the parameterized structure makes it possible to transfer the meta-learned LR schedule to be readily used in new query tasks. In a nutshell, this paper mainly makes the following five-fold contributions.

(1) The MLR-SNet is proposed to learn an adaptive LR schedule for SGD algorithm, which is capable of dynamically adjusting LR during the DNN training process based on current training loss as well as the information delivered from past training histories stored in the MLR-SNet. Due to the explicit parameterized formulation of the MLR-SNet, it can be more flexible than hand-designed policies to find a proper LR schedule for specific learning tasks.

(2) The proposed method is model-agnostic, and can be applied to the SGD implementation on general DNN models. That is naturally feasible since the proposed MLR-SNet is with general loss information as its inputs, which is independent of the structure of the DNN models. The MLR-SNet is thus able to be generally applied to different DNN training problems, e.g., image and text classification problems, as shown in Figs. 1d and 1e. It can be seen that the meta-learned LR schedules have a similar tendency as specifically pre-defined ones, as depicted in Figs. 1a and 1b, but with more adaptive variations at their locality. This validates the capability and efficacy of our method for adaptively scheduling LR.

(3) With an explicit parameterized structure, it is possible to readily transfer the meta-trained MLR-SNet for helping schedule LR of SGD on new heterogeneous tasks. Different from hand-designed LR schedules often requiring to redesign the LR schedules or re-tune the hyperparameters for new query tasks, the meta-learned MLR-SNet is plug-and-play, and without additional hyperparameters to tune. To verify this point, we transfer the meta-learned MLR-SNet to different training epochs, datasets and network architectures, and achieve comparable performance with the corresponding best hand-designed LR schedules in the test data. Since it is directly employed as an off-the-shelf LR-schedule setting function, it is with similar computational complexity as the hand-designed LR schedules. Besides, it has been empirically verified that the generalization performance of meta-learned MLR-SNet is slightly related to the size of the meta-training dataset, while relatively weakly related to the similarity between meta-training and meta-test tasks and DNN models. This reveals the potential of transferring meta-learned LR schedules to improve the DNN training for the unseen tasks,

and is hopeful to save large labor and computation cost for DNN training in more real applications.

(4) The MLR-SNet is meta-learned to improve the generalization performance of the learned model on unseen data. We validate that with sound guidance of clean data as meta-data, our MLR-SNet can help achieve better robustness when training data are biased with corrupted noise than hand-designed LR schedules.

(5) We theoretically prove that the DNN models trained with the SGD algorithm, using LR schedules produced by our MLR-SNet, can obtain a convergence guarantee. Meanwhile, we can also prove the convergence guarantee for our MLR-SNet updated by the Adam algorithm guided by the validation loss under some mild conditions.

The paper is organized as follows. Section 2 reviews the related works. Section 3 presents the MLR-SNet model as well as its learning algorithm. Section 4 demonstrates the experimental evaluations to validate the adaptability, transferability and robustness of the MLR-SNet, as compared with current LR schedules policies. Section 5 provides some analysis on MLR-SNet, e.g., its convergence and computational complexity. The paper is finally concluded.

2 Related Work

Meta Learning for Optimization. Meta learning, or learning to learn has a long history in psychology [33], [34]. Meta learning for optimization can date back to 1980s-1990s [35], [36], aiming to meta-learn the optimization process of learning itself. Inspired from such beneficial attempts, many researches were proposed to meta-learn the optimization process of different learning tasks. The early work is proposed by Schmidhuber *et al.* [35], developing an end-to-end differentiable system to jointly train both the network and the learning algorithm by gradient descent, making the network able to modify its own weights. Bengio *et al.* [36] also proposed to learn parameterized local neural net update rules that avoid back-propagation. Furthermore, Hochreiter *et al.* [37] jointly train two networks, in which the output of back-propagation from one network was fed into an additional learning network to attain the learning algorithm.

Recently, [38], [39], [40], [41], [42], [43] have attempted to scale this idea to larger DNN optimization problems. The main idea is to construct a meta-learner as the optimizer, which takes the gradients as input and outputs the whole updating rules. These approaches tend to make selecting appropriate training algorithms, scheduling LR and tuning other hyperparameters in an automatic way. The meta-learner of these approaches can be updated by minimizing the generalization error on the validation set. Furthermore, [41] utilized reinforcement learning and [42] used test error of few-shot learning tasks to train the meta-learner. Except for solving continuous optimization problems, some works employ these ideas to other optimization problems, such as black-box functions [37], few-shot learning [44], [45], model's curvature [46], evolution strategies [47], combinatorial functions [48], MCMC proposals [49], etc.

Though faster in decreasing training loss than traditional optimizers in some cases, the learned optimizers by this line of methods always could not generalize well to varying problems from the training ones, especially longer horizons [41] and larger scale optimization problems [40]. Comparatively, the meta-learned LR schedule by our proposed method can be easily transferred to new heterogeneous tasks, as clearly shown in our experiments reported in Section 4.2.

HPO and LR Schedule Adaptation. Hyper-parameter optimization (HPO) was historically investigated by selecting proper values for algorithm hyperparameters to obtain better performance on the validation set (see [50] for an overview). Typical methods include grid search, random search [51], Bayesian optimization [52], gradient-based methods [53], [54], [55], etc. Recently, some works attempt to adapt a proper LR schedule under the framework of gradient-based HPO[54], [56]. Typical works along this line include Meta-SGD [45], RTHO [53] and HD [56]. Albeit making certain progress on the LR adaptation task, these methods mainly focus on properly adapting learning rates themselves along the training process. This setting makes them relatively hardly scale to long-horizons problems [43], [58], and less adaptable to the variations of new heterogeneous query tasks. More discussions about this point can refer to Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3184315. Along this research line, the most related work to ours is E^3 BM [57], which uses meta-learners to generate LRs and combination weights for different base-learners against few-shot learning tasks. This method takes concatenated mean values of sample data and training gradient as input, and mainly focuses on predicting learning rates for base-learners. The learning rates for training more cumbersome feature extractors, however, are largely not considered. Comparatively, our method uses more concise training loss knowledge as input, and attempts to get the learning rates of the entire network parameters, including both feature extractor and base learners. This makes it feasible to solve more challenging LR setting tasks, e.g., ImageNet.

Transfer to Heterogeneous Tasks. Transfer learning [59] aims to transfer knowledge obtained from source task to help the learning on the target task. Most transfer learning approaches assume the source and target tasks consist of similar instances, features or model spaces [60], which greatly limits their application range. Recently, meta learning [44] aims to learn common knowledge/methodology shared over observed tasks, such that the learned knowledge/methodology is expected to be transferred to unseen tasks [1], [2]. Similarly, our method aims to realize such a methodology-level transfer learning for the LR-schedule setting task, i.e., learn a general LR schedule predictor which is plug-and-play and easy to transfer to new query tasks. Such task-transferable capability, however, is not possessed by conventional hand-designed LR schedules and HPO methods.

3 MLR-SNet

The problem of training DNNs can be formulated as the following non-convex optimization problem

$$\min_{w\in\mathbb{R}^d} f^{Tr}(D_{Tr};w) := \frac{1}{N}\sum_{i=1}^{N} f_i^{Tr}(w), \tag{1}$$

where f_i^{Tr} is the training loss function for data samples $i \in D_{Tr} = \{1, 2, \ldots, N\}$, which characterizes the deviation of the model prediction from the data labels, and $w \in \mathbb{R}^d$ represents the parameters of the model (e.g., the weight matrices in the trained DNN) to be optimized. SGD [3], [61] and its variants, including Momentum [62], Adagrad [4], Adadelta [5], RMSprop [6], Adam [7], are often used for DNN training. In general, these algorithms can be expressed as the following formulation:

$$w_{t+1} = w_t + \Delta w_t, \Delta w_t = \mathcal{O}_t(\nabla f_w^{Tr}(D_{Tr}; w_t), \mathcal{H}_t; \Theta_t), \tag{2}$$

where w_t is tth updating model parameters, $\nabla f_w^{Tr}(D_{Tr}; w_t)$ denotes the gradient of f^{Tr} at w_t, $\mathcal{H}_t$ represents the historical gradient information, and Θ_t is the hyperparameter of the optimizer $\mathcal{O}$, e.g., LR, in the current interation. To present our method's efficiency, we focus on the following vanilla SGD algorithm in this paper,[1]

$$w_{t+1} = \xi_t(w_t, \alpha_t) = w_t - \alpha_t \nabla_w f^{Tr}(D_t; w_t), \tag{3}$$

where $\nabla_w f^{Tr}(D_t; w_t) = \frac{1}{|D_t|} \sum_{i \in D_t} \nabla_w f_i^{Tr}(w_t)$, $D_t \subset D_{Tr}$ denotes the batch samples randomly sampled from the training dataset D_{Tr}, $|D_t|$ denotes the batch size, $\nabla_w f_i^{Tr}(w_t)$ denotes the gradient of sample i computed at w_t and α_t is the LR at tth iteration.

3.1 Existing LR Schedule Strategies

As [17] demonstrated, the choice of LR plays a central role for effective DNN training with SGD. In this part, we will recall LR schedules proposed in the previous works.

The following presents the commonly used pre-defined LR schedules for current DNN training:

$$\begin{aligned}
&\text{(Fixed)} \quad \alpha_t = \alpha_0, \\
&\text{(MultiStep)} \quad \alpha_t = \alpha_0 \times (\gamma_M)^i, l_{i-1} \le E_{cur} \le l_i, \\
&\qquad \text{for given epochs } l_0, l_1, \ldots, l_n, \\
&\text{(Exponential)} \quad \alpha_t = \alpha_0 \times (\gamma_E)^{E_{cur}-1}, \\
&\text{(SGDR)} \ \alpha_t = \alpha_{\min} + 0.5(\alpha_{\max} - \alpha_{\min})\left(1 + cos\left(\frac{E_{cur}}{E_{per}}\pi\right)\right),
\end{aligned} \tag{4}$$

where α_0 denotes the initial LR and α_t denotes the LR at t-iteration, $[\alpha_{\min}, \alpha_{\max}]$ specifies a range for LR setting of SGDR. E_{cur} accounts for how many epochs have been performed, and E_{per} denotes that after E_{per} epochs SGDR restarts to decrease the LR, and it generally sets $E_{per} = E_0 \times (T_{Mul})^k$ for the kth restart. $\gamma_M, \gamma_E < 1$ denote the decay factors for MultiStep and Exponential, respectively.

Inspired by current meta-learning developments [44], [63], [64], some researches proposed to learn a generic optimizer from data [38], [39], [40], [41], [42], [43]. The main idea among them is to learn a meta-learner as the optimizer to guide the learning of the whole updating rules. For example, [38] tries to replace Eq. (2) with the following formulation:

$$w_{t+1} = w_t + g_t, [g_t, h_{t+1}]^T = m(\nabla_t, h_t; \phi), \tag{5}$$

where g_t is the output of a LSTM net m, parameterized by ϕ, whose state is h_t. This strategy has been expected to make selecting appropriate training algorithms, scheduling LR and tuning other hyperparameters in a unified and automatic way. Though faster in decreasing training loss than the traditional optimizers in some cases, the learned optimizer, however, might not always generalize well to more variant and diverse problems, like longer horizons [43] and large scale optimization problems [40] since the framework is too flexible to be relatively easy to overfit training tasks.

Rather than the entire learning rules, a natural compromise for the task is to focus on the LR schedules while keep using the gradient knowledge across the meta-training/testing stages. Inspired by this motivation, recently some methods [53], [56] consider the following constrained optimization problem to search the optimal LR schedule α^* such that the produced models are associated with a small validation error

$$\begin{aligned}
&\min_{\alpha=\{\alpha_0,\ldots,\alpha_{T-1}\}} f^{Val}(D_{Val}; w_T), \\
&s.t.\ w_{t+1} = \xi_t(w_t, \alpha_t),\ t = 0, 1, \ldots, T-1,
\end{aligned} \tag{6}$$

where f^{Val} denotes the validation loss function, $D_{Val} = \{1, 2, \ldots, M\}$ denotes hold-out validation set, α_t is to-be-solved LR hyperparameter, $\xi_t : \mathbb{R}^d \times \mathbb{R}_+ \to \mathbb{R}^d$ is a stochastic weight update dynamics, like the updating rule of the vanilla SGD in Eq. (3), and T is the maximum iteration step. Though achieving comparable results on some tasks with hand-designed LR schedules and meta-learned optimizers, when generalized to new tasks, the meta-learned LR schedules keep constant. This makes it hardly well adapt to the task variations, and thus leads to possible performance degradation. Namely, it still requires learning the LR schedules from scratch especially for new heterogeneous tasks to obtain satisfied performance, which is sometimes time and computation expensive.

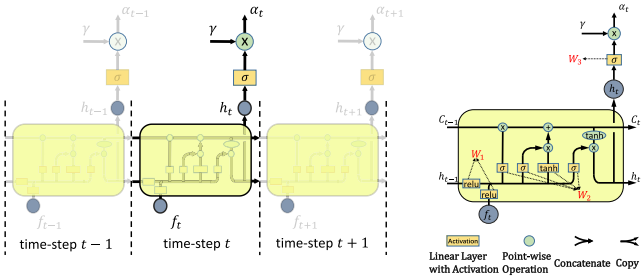
3.2 Proposed Meta-LR-Schedule-Net Method

To address the aforementioned issues, we propose to design a meta-learner with an explicit mapping formulation to parameterize LR schedules as shown in Fig. 1c, called Meta-LR-Schedule-Net (*MLR-SNet* for brevity). The parameterized structure can bring two benefits: 1) It gives fine flexibility to learn a proper LR schedule to comply with the significantly changed training dynamics of DNNs; 2) It makes the meta-learned LR schedules become transferable and plug-and-play, able to be readily applied to new heterogeneous tasks, without requiring to re-learn or tune additional hyperparameters.

3.2.1 Formulation of MLR-SNet

The computational graph of MLR-SNet is depicted in Fig. 2a. Let $\mathcal{A}(\cdot, \cdot; \phi)$ denote MLR-SNet. Then the updating equation of the vanilla SGD algorithm in Eq. (3) can be rewritten as

1. For different learning tasks, the commonly used optimizers are different. For example, image tasks often use SGD with Momentum, while text tasks always employ SGD or Adam. To guarantee the chosen optimizer able to be applied to various tasks, we learn the LR schedules for the vanilla SGD in this paper. We further validate that MLR-SNet can be applied to other optimizers, e.g., Adam (refer to Section 5.5).



(a) Computational graph of MLR-SNet (b) One step of MLR-SNet

Fig. 2. The structure and computational graph of our proposed MLR-SNet.

$$\begin{aligned} w_{t+1} &= w_t - \mathcal{A}(f_t, \theta_t; \phi)\nabla_w f^{Tr}(D_t; w_t), \\ \text{where } f_t &= f^{Tr}(D_t; w_t),\ \theta_t = (h_t, c_t)^T, \end{aligned} \tag{7}$$

where $\mathcal{A}(f_t, \theta_t; \phi)$ outputs the LR (α_t) at the tth iteration, ϕ is the parameter of MLR-SNet, f_t is the loss of the batch samples D_t at the tth iteration, and $\theta_t = \{h_{t-1}, c_{t-1}\}$, where $h_t, c_t \in R^{d'}$ denote the output and state of the LSTM cell at the tth iteration ($t = 0, \ldots, T-1$), d' represents the dimension of the state vectors (i.e., the size of hidden nodes). At each SGD iteration, $\mathcal{A}(f_t, \theta_t; \phi)$ can learn an explicit loss-LR dependent relationship, such that the net can adaptively predict LR according to the current input loss f_t, as well as the historical training information θ_t stored in the net. For every iteration step, the whole forward computation process can be written as (as shown in Fig. 2b)

$$\begin{aligned} \begin{pmatrix} I_t \\ F_t \\ O_t \\ g_t \end{pmatrix} &= \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W_2 \begin{pmatrix} \text{relu} \\ \text{relu} \end{pmatrix} W_1 \begin{pmatrix} h_{t-1} \\ f_t \end{pmatrix} \\ c_t &= F_t \odot c_{t-1} + I_t \odot g_t \\ h_t &= O_t \odot \tanh(c_t) \\ p_t &= \sigma(W_3 h_t) \\ \alpha_t &= \gamma \cdot p_t, \end{aligned} \tag{8}$$

where I_t, F_t, O_t denote the Input, Forget and Output gates in the current iteration, and $\sigma, \tanh, \text{relu}$ denote the Sigmoid, Tanh and ReLU activation functions, respectively. The MLR-SNet parameter is $\phi = (W_1, W_2, W_3)$, where $W_1 \in \mathbb{R}^{d' \times (d'+1)}, W_2 \in \mathbb{R}^{4d' \times 2d'}, W_3 \in \mathbb{R}^{1 \times d'}$. Different from the vanilla LSTM, the input h_{t-1} and the training loss f_t are preprocessed by a fully-connected layer W_1 with ReLU activation function. Then it works as the LSTM and obtains the output h_t. Subsequently, the predicted value p_t is obtained by a linear transform W_3 on the h_t with a Sigmoid activation function. Finally, we introduce a scale factor γ to guarantee the final predicted LR located in the interval of $[0, \gamma]$. In our paper, we set $\gamma = \frac{f_0^{1/2} \log |f_0 * C|}{4C^{1/4}}$, where f_0 denotes the initial loss, and C accounts for the number of classes. Albeit simple, this net is known to be capable of finely dealing with such long-term information dependent problem, and thus expected to learn a proper LR schedule to comply with the training dynamics of DNNs.

Remark. On the one hand, different from Eq. (6) directly learning the LR schedules themselves, we use the MLR-SNet parameterized by ϕ to learn the LR schedules. This parameterized meta-learner helps extract the latent methodology of how to design a proper LR schedule for generally handling a DNN training problem, rather than only the hyperparameters for a specific problem. Therefore, the meta-learned MLR-SNet can be readily transferred to new DNN training tasks for designing the LR schedules. On the other hand, compared with learning the whole updating rules as represented in Eq. (5), our MLR-SNet learns the most important LR schedules for the SGD algorithm while keeps using the gradient knowledge of the learned problem, making it relatively easier to learn and under better control. This can explain why MLR-SNet always tends to make the DNN training procedure more robust and efficient in experiments.

3.2.2 Learning Algorithm of MLR-SNet

(1) Meta-Train: adapting to the training dynamics of DNN. The MLR-SNet can be meta-trained to improve the generalization performance on unseen validation data for DNN training by solving the following optimization problem:

$$\begin{aligned} &\min_{\theta} f^{Val}(D_{Val}; w_T(\phi)), \\ &\text{s.t. } w_{t+1}(\phi) = \xi_t(w_t, \mathcal{A}(f_t, \theta_t; \phi)),\ t = 0, \ldots, T-1. \end{aligned} \tag{9}$$

where $f_t = f^{Tr}(D_t; w_t)$ and $\xi_t(w_t, \alpha_t)$ corresponds to Eq. (3). Now the important question is how to efficiently meta-learn the parameter ϕ for the MLR-SNet. We employ the online approximation technique in [64] to jointly update ϕ and model parameter w to explore a proper LR schedule with better generalization for DNNs training. However, the step-wise optimization for ϕ is still expensive to handle large-scale datasets and huge DNN structures. To address this issue, we attempt to update ϕ once after updating w several steps (T_{val}). The updating process can then be formulated as:

Algorithm 1. The Meta-Train Algorithm of MLR-SNet

Input: Training data D_{Tr}, validation set D_{Val}, max iterations T, updating period T_{val}.
Output: Model parameter w_T and MLR-SNet parameter $\phi_s, s \in S \subset \{1, \ldots, T\}$

```
Initialize model parameter w_0, MLR-SNet cell θ_0 = (h_0, c_0)^T,
  and MLR-SNet parameter φ_0.
for t = 0 to T − 1 do
  D_t ← SampleMiniBatch(D_Tr) with batch size |D_t|.
  if t % T_val = 0, then
    D_t^(v) ← SampleMiniBatch(D_Val) with batch size |D_t^(v)|.
    Update φ_{t+1} by Eq. (10).
  end if
  Update w_{t+1} by Eq. (12).
end for
```

Updating ϕ. When it does not satisfy the updating conditions, ϕ keeps fixed; otherwise, ϕ will be updated using the model parameter w_t and MLR-SNet parameter ϕ_t obtained in the last step by minimizing the validation loss defined in Eq. (9). Adam algorithm can be utilized to optimize the validation loss, expressed as:

$$\phi_{t+1} = \phi_t + Adam(\nabla_\theta f^{Val}(D_t^{(v)}; \hat{w}_{t+1}(\theta)); \eta_t), \tag{10}$$

where $Adam$ denotes the Adam algorithm, whose input is the gradient of validation loss with respect to MLR-SNet parameter ϕ on mini-batch samples $D_t^{(v)}$ from D_{Val}. η_t denotes the LR of Adam. $\hat{w}_{t+1}(\phi)$[2] is virtually formulated on a mini-batch training samples D_t from D_{Tr} as follows:

$$\hat{w}_{t+1}(\phi) = w_t - \mathcal{A}(f^{Tr}(D_t, w_t), \theta_t; \phi) \cdot \nabla_w f^{Tr}(D_t, w)\big|_{w_t}. \quad (11)$$

Updating w. Then, the updated ϕ_{t+1} is employed to ameliorate the model parameter w, i.e.,

$$w_{t+1} = w_t - \mathcal{A}(f^{Tr}(D_t, w_t), \theta_t; \phi_{t+1}) \cdot \nabla_w f^{Tr}(D_t, w)\big|_{w_t}. \quad (12)$$

The whole algorithm in the meta-training stage can then be summarized in Algorithm 1. All computations of gradients can be efficiently implemented by automatic differentiation libraries, like PyTorch [23], and easily used to general DNN architectures. It can be seen that the MLR-SNet can be gradually optimized during the learning process and adjust the LR dynamically based on the training dynamics of DNNs.

Algorithm 2. The Meta-Test Algorithm of MLR-SNet

Input: Training data D_{Tr}^{μ} for new task μ, max iterations T_μ, meta-learned MLR-SNet $\mathcal{A}(\cdot,\cdot;\phi_s), s \in S$.
Output: Model parameter u_T.
1: Initialize model parameter u_0, MLR-SNet cell $\theta_0 = (h_0, c_0)^T$, and choose the subset of meta-learned MLR-SNet $\phi_s, s \in S \subset \{1, \ldots, T\}$ for test.
2: **for** $t = 0$ **to** $T_\mu - 1$ **do**
3: $D_t^{\mu} \leftarrow$ SampleMiniBatch(D_{Tr}^{μ}) with batch size $|D_t^{\mu}|$.
4: Compute the loss $f^{Tr}(D_t^{\mu}, u_t)$, and then MLR-SNet predicts the LR $\mathcal{A}(f^{Tr}(D_t^{\mu}, u_t), \theta_t; \phi_s)$ for current iteration.
5: Update u_{t+1} by Eq. (13).
6: **end for**

(2) Meta-Test: generalization to new heterogeneous tasks. After the meta-training stage, the meta-learned MLR-SNet with parameter ϕ_T is expected to be transferred to guide the SGD running on new DNN training tasks. To better preserve the proper LR changing dynamics during DNN training, we prefer to keep several MLR-SNet forms with parameters $\phi_s, s \in S \subset \{1, \ldots, T\}$ (e.g., $\phi_{T/3}, \phi_{2T/3}, \phi_T$ as employed in our experiments) and use them as LR schedules along with different iterations in the meta-testing stage. The new DNN parameter u for the new task is then updated by (the whole meta-test process refers to Algorithm 2)

$$u_{t+1} = u_t - \mathcal{A}(f^{Tr}(D_n, u_t), \theta_t; \phi_s) \cdot \nabla_u f^{Tr}(D_n, u)\big|_{u_t}, \quad (13)$$

where $\phi_s, s \in S$ is the parameters of the subset of the meta-learned MLR-SNets. This means that we restore several LR schedule setting rules, and dynamically employ specific ones along different range of DNN training iterations. It is seen that the meta-learned MLR-SNets are plug-and-play, and involve no additional hyperparameters to tune.

2. Notice that $\hat{w}_{t+1}(\phi)$ here is a function of ϕ to guarantee the gradient in Eq. (10) to be able to be feasibly computed.

4 Experimental Results

To evaluate the proposed MLR-SNet, we first conduct experiments to show our method can learn proper LR schedules compared with baseline methods (Section 4.1). Then we transfer the meta-learned LR schedules to various tasks for meta-test to show its superiority in generalization (Section 4.2). What influences the generalization performance of meta-learned LR schedules is discussed in Section 4.3. Finally, we show our method behaves robust and stable when training data contain different data corruptions (Section 4.4).

4.1 Meta-Train: Evaluation of the LR Schedules Meta-Learned by MLR-SNet

In this section, we attempt to evaluate the capability of MLR-SNet to learn proper LR schedules for various tasks.

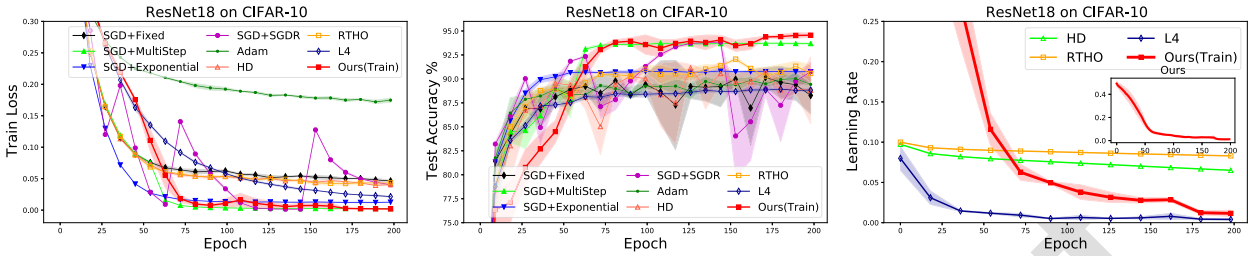
4.1.1 Image Classification Benchmarks

Datasets. We choose CIFAR-10 and CIFAR-100 to present the efficiency of our method, which include 32×32 color images arranged in 10 and 100 classes, respectively. Both datasets contain 50,000 training and 10,000 test images.

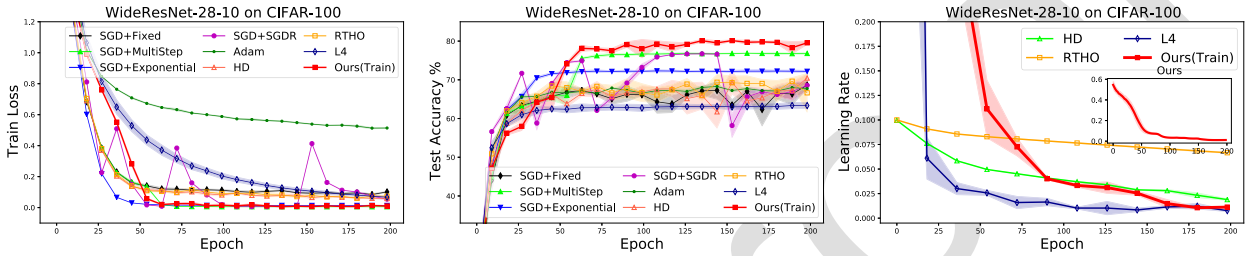
Baselines. The compared methods include the SGD with hand-designed LR schedules (the formulation is expressed as Eq. (4)): 1) *Fixed* LR, 2) *Exponential* decay, 3) *MultiStep* decay, 4) SGD with restarts (*SGDR*) [22]. Meanwhile, we compare with adaptive gradient method: 5)*Adam*, LR search method: 6) *L4* [28], and current LR schedule adaptation method: 7) hyper-gradient descent (*HD*) [56], 8) real-time hyperparameter optimization (*RTHO*) [53]. We run all experiments with 3 different seeds reporting accuracy. Our algorithm and RTHO [53] randomly select 1,000 clean images in the training set of CIFAR-10/100 as validation data.

Hyperparameter Setting. We employ ResNet-18 on CIFAR-10 and WideResNet-28-10 [65] on CIFAR-100. All compared methods and MLR-SNet are trained for 200 epochs with batch size 128. For baselines involving SGD as base optimizer, we set the initial LR as 0.1, and weight decay as $5e^{-4}$. While for *Adam*, we just follow the default parameter setting. As for each LR schedule, *MultiStep* decays LR by 10 every 60 epochs (i.e., $\gamma_M = 0.1, l_0 = 0, l_1 = 60, l_2 = 120, l_3 = 180, l_4 = 200$); *Exponential* multiplys LR with $\gamma_E = 0.95$ every epoch; *SGDR* sets $\alpha_{\min} = 1e^{-5}, \alpha_{\max} = 0.1$, and $E_0 = 10, T_{Mult} = 2$. *L4*, *HD* and *RTHO* update LR every data batch, and we use the recommended setting in the original paper. *HD* and *RTHO* search different hyper-LRs from $\{1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}, 1e^{-7}\}$ reporting the best performing hyper-LR. The detailed discussion can be found in the supplementary material, available online.

MLR-SNet Architecture. The architecture of MLR-SNet is illustrated in Section 3.2. In our experiment, the size of hidden nodes (i.e., d') is set as 50. The initialization of MLR-SNet follows the default setting in PyTorch. We employ Adam optimizer to train MLR-SNet, and set the LR as $1e^{-3}$, and the weight decay as $1e^{-4}$. The input of MLR-SNet is the training loss of a mini-batch samples. Every iteration LR is predicted by MLR-SNet and we update it every 100 iterations ($T_{val} = 100$) according to the loss of the validation data.



(a) CIFAR-10 with ResNet18 SGD



(b) CIFAR-100 with WideResNet-28-10 for SGD

Fig. 3. Changing tendencies in terms of training loss (left column) and test accuracy (middle column) in iterations of all comparison methods on image classification datasets in the meta-train stage. The LR schedules (right column) employed by all methods are also compared.

Results. Figs. 3a and 3b show changing tendencies of training loss and test accuracy on CIFAR-10 and CIFAR-100 datasets in iterations of all competing methods, respectively, and Table 1 shows the corresponding classification accuracy on the test set. It can be observed that: 1) our MLR-SNet obtains better test performance than all other competing methods, and the learned LR schedules by MLR-SNet have similar shapes as the corresponding hand-designed policies (as depicted in Fig. 1d), while with more elaborate variation details in the locality for better adapting training dynamics. Besides, such LR schedules learned by MLR-SNet empirically justify the general trend of hand-designed LR schedules (e.g., MultiStep, Exponential) for CIFAR-10 and CIFAR-100 datasets. This implies that the proposed method could be potentially useful to verify the validity of hand-designed LR schedules for relatively complicated and variant training tasks. 2) The Fixed LR decreases the loss sharing the similar performance to other baselines at the early training, while fails to further decrease loss at the later training stages. This implies that this strategy could not finely adapt to such DNN training dynamics. 3) The MultiStep LR drops the LR at some epochs, and such an elegant strategy overcomes the issue of Fixed LR and decreases loss substantially after dropping the LR. Thus it obtains higher test performance. Besides, though MultiStep and MLR-SNet can decrease the loss to 0 approximately, our MLR-SNet achieves better generalization performance since the outer objective in Eq. (9) tends to help learn the LR schedules to find a better minima. 4) The Exponential LR decreases loss with a faster speed at the early training steps than other baselines, while makes slow progress due to smaller LR at the later stages. 5) The SGDR LR uses the cyclic LR, decreasing loss as fast as the Exponential LR. 6) Though Adam has an adaptive coordinate-specific LR, it behaves worse than MultiStep and Exponential LR as demonstrated in [66]. An extra tuning is thus necessary for better performance. 7) L4 greedily searches LR locally to decrease loss, making it fairly hard to adapt the complex DNNs training dynamics, and even with worse test performance than Fixed LR. 8) HD and RTHO perform similarly to hand-designed LR schedules. The LR schedules of these competing methods behave more flatter than our MLR-SNet. This is because that they are largely absent of the past training history information to guide the learning of LR, and they are easily fall into bad local minima at the early training stage, and hardly escape from this bad local minima, which then hampers their generalization performance. As compared, our MLR-SNet has a larger range of LR with more adaptive variations at their locality. It is seen that it drops quickly at the early training stage, and then gradually converges. This illustrates that LR schedules produced by MLR-SNet can better adapt to complicated training dynamics, naturally leading to its better generalization performance. 9) Since the image tasks often use SGD algorithm with Momentum (SGDM) to train DNNs, we also present the test performance of baseline methods trained with SGDM with momentum 0.9 in Table 2. They obtain a remarkable improvement when trained with SGD. Though not using extra historical gradient information to help optimization, our MLR-SNet is capable of achieving comparable results with baselines, since it also insightfully stores the historical LR training information in the net.

TABLE 1
Test Accuracy (%) of CIFAR Datasets With SGD Baselines

Optimizer	CIFAR-10	CIFAR-100
SGD+Fixed	92.26 ± 0.12	70.67 ± 0.34
SGD+MultiStep	93.82 ± 0.09	77.04 ± 0.17
SGD+Exponential	90.93 ± 0.11	76.88 ± 0.08
SGD+SGDR	93.92 ± 0.11	72.52 ± 0.34
Adam	90.86 ± 0.15	68.94 ± 0.24
SGD+L4	89.15 ± 0.14	63.61 ± 0.65
SGD+HD	92.34 ± 0.09	72.22 ± 0.30
SGD+RTHO	92.60 ± 0.18	72.32 ± 0.47
MLR-SNet (Meta-train)	$\mathbf{94.80 \pm 0.10}$	$\mathbf{80.44 \pm 0.17}$

TABLE 2
Test Accuracy (%) of CIFAR Dataset With SGDM Baselines

Optimizer	CIFAR-10	CIFAR-100
SGDM+Fixed	87.69 ± 0.14	70.88 ± 0.12
SGDM+MultiStep	$\mathbf{95.08 \pm 0.13}$	80.74 ± 0.19
SGDM+Exponential	94.64 ± 0.05	78.87 ± 0.04
SGDM+SGDR	95.06 ± 0.17	$\mathbf{80.93 \pm 0.05}$
Adam	90.86 ± 0.15	68.94 ± 0.24
SGDM+L4	91.03 ± 0.14	66.51 ± 2.83
SGDM+HD	93.99 ± 0.12	76.80 ± 0.19
SGDM+RTHO	93.17 ± 0.49	76.14 ± 0.29
MLR-SNet (Meta-train)	94.80 ± 0.10	80.44 ± 0.17

TABLE 3
Test Perplexity on the Penn Treebank Dataset

Optimizer	2-layer LSTM	3-layer LSTM
SGD+Val Strategy	74.33 ± 0.23	76.05 ± 0.39
Adam+Val Strategy	71.17 ± 0.23	74.80 ± 0.73
SGD+L4	82.58 ± 1.32	92.27 ± 0.92
SGD+HD	76.90 ± 0.33	78.63 ± 0.08
SGD+RTHO	76.69 ± 0.11	78.52 ± 0.16
MLR-SNet (Meta-train)	$\mathbf{70.53 \pm 0.25}$	$\mathbf{72.28 \pm 0.25}$

4.1.2 *Text Classification Benchmarks*

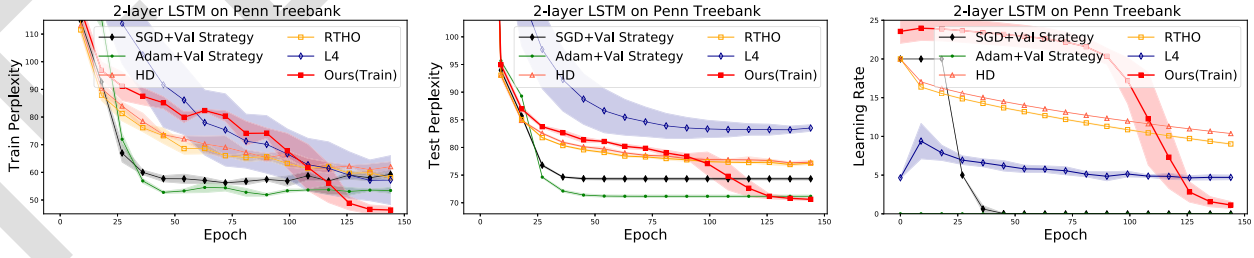
Dataset. We choose Penn Treebank dataset [67] for evaluation, which consists of 929k training words, 73k validation words, and 82k test words, with a 10k vocabulary in total.

Baselines. We compare with 1) SGD, 2) Adam with LR tuned using a validation set (*SGD+Val Strategy* and *Adam +Val Strategy*). They drop the LR by a factor of 4 when the validation loss stops decreasing. Also, we compared with 3) *L4*, 4) *HD*, 5) *RTHO*. We run all experiments with 3 different seeds reporting accuracy. Our algorithm and RTHO [53] regard the validation set as validation data.

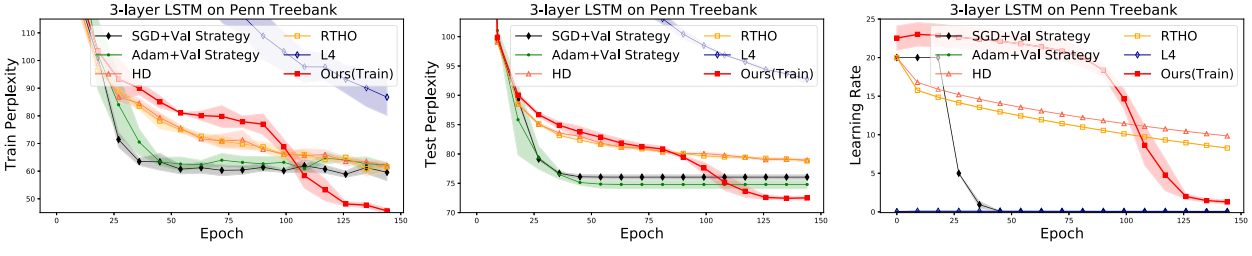
Hyperparameter Setting. We use a 2-layer and 3-layer LSTM network which follows a word-embedding layer and the output is fed into a linear layer to compute the probability of each word in the vocabulary. Hidden size of LSTM cell is set to 512 and so is the word-embedding size. We tie weights of the word-embedding layer and the final linear layer. Dropout is applied to the output of word-embedding layer together with both the first and second LSTM layers with a rate of 0.5. As for training, the LSTM net is trained for 150 epochs with a batch size of 32 and a sequence length of 35. We set the base optimizer SGD to have an initial LR of 20. For Adam, the initial LR is set to 0.01 and weight for moving average of gradient is set to 0. We apply a weight decay of $5e^{-6}$ to both base optimizers. All experiments involve a 0.25 clipping to the network gradient norm. For both SGD and Adam, we decrease LR by a factor of 4 when performance on validation set shows no progress. For L4, we try different α in $\{0.1, 0.05, 0.01, 0.005\}$ and report the best test perplexity among them. For both *HD* and *RTHO*, we search the hyper-lr lying in $\{1, 0.5, 0.1, 0.05\}$, and report the best results.

MLR-SNet Architecture. We keep the same setting as the image classification, while we take $\frac{\mathcal{L}_{Tr}}{\log(vocabulary\ size)}$ as input of MLR-SNet to deal with the influence of large scale classes for text dataset.

Results. Figs. 4a and 4b show the train and test perplexity in iterations on the Penn Treebank dataset with 2-layer and 3-layer LSTM, respectively. The test perplexity of final trained model is presented in Table 3. It can be observed that: 1) The Val Strategy heuristically drops LR when the validation loss stops decreasing. This hand-designed LR schedules can decrease the loss quickly at the early training stage to find a good minima, while it is hard to further search for a better solution. 2) Our MLR-SNet predicts LR according to training dynamics and updates its parameters by minimizing the validation loss, i.e., if the LR schedules produced by the MLR-SNet are of high quality, then a DNN model trained with such LR schedules should achieve low loss on a separate validation dataset. This process is a relatively more intelligent way to employ the validation dataset than Val Strategy. Thus

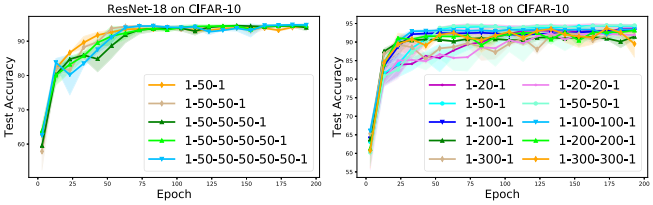


(a) Penn Treebank with 2-layer LSTM for SGD



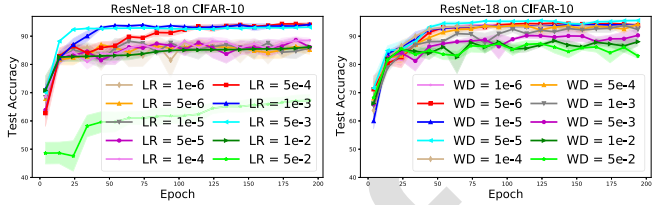
(b) Penn Treebank with 3-layer LSTM for SGD

Fig. 4. Changing tendencies in terms of training perplexity (left column) and test perplexity (middle column) in iterations of all comparison methods on text classification datasets in the meta-train stage. The LR schedules (right column) employed by all methods are also compared.



(a) Effect of layer size (b) Effect of node size

Fig. 5. Test accuracy on CIFAR-10 with ResNet-18 with different settings of (a) network layers and (b)hidden layer nodes of MLR-SNet architectures.



(a) Effect of learning rate (b) Effect of weight decay

Fig. 6. Test accuracy on CIFAR-10 with ResNet-18 with different settings of (a) learning rates and (b) weight decays of meta-optimizer 'Adam'.

our method achieves comparable or even better performance than Adam and SGD. The meta-learned LR schedules of the MLR-SNet are shown in Fig. 1e, which first decreases slowly at the early training stage, then is demonstrated as a quick drop process, and finally gradually becomes stable. 3) L4 often falls into a bad minimum since it greedily searches LR locally. 4) Since HD and RTHO lack an explicit parameterized structure, they directly learn LR schedules themselves by minimizing the validation loss, which tends possibly to fall into a bad local minima without considering the past training history information, and lead to performance degradation. As compared, our MLR-SNet can predict LR schedules based on complicated training dynamics, and brings performance improvement. 5) When the number of LSTM's layers increases, the LR schedules predicted by MLR-SNet show more advantages for such an LSTM training problem, and bring more performance improvements compared with hand-designed LR schedules.

Remark. Actually, the performance of compared baselines can be approximately regarded as the best/upper performance bound. Since these strategies have been tested to work well for specific tasks, and they are written into the standard deep learning library. For different image and text tasks, our MLR-SNet can achieve a similar or even slightly better performance compared with the best baselines. We thus believe that these experiments can demonstrate the effectiveness and generality of our proposed method.

4.1.3 Ablation Study

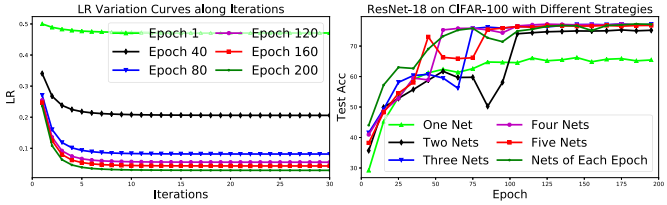
To learn the LR schedule, it produces new hyperparameters, like the structure of meta-learner and the hyperparameters of meta-optimizer. We conduct ablation study of them.

The Architecture of MLR-SNet. Fig. 5 shows the test accuracy on CIFAR-10 with ResNet-18 of different MLR-SNet architecture configurations. We have set varying layers for our MLR-SNet with fixed hidden nodes to see its performance variation. As shown in Fig. 5a, it is seen that the depth of the MLR-SNet has unsubstantial influence on the final performance. As for the width of the MLR-SNet, we have also tested different node number settings with fixed 1 and 2 layers of MLR-SNet. As shown in Fig. 5b, it can be observed that when the node size of the hidden layer is set small, e.g., 20, it inclines to show slightly slower convergence tendency at the early training stage compared with that depicted under larger node size. Besides, when the node size of the hidden layer is set relatively large, e.g., 300, the accuracy curve inclines to perform relatively more unstable at the whole training process even it also achieves good performance at final. Yet in most cases, the method performs consistently well. This can validate that our algorithm is not substantially sensitive to the configuration setting of the MLR-SNet's architecture. We easily suggest to use the one hidden layer with 50 nodes for MLR-SNet architecture as in our experiments attributed to its simplicity and relatively low cost for computation.

The Hyperparameters of the Meta-Optimizer. Fig. 6 shows the test accuracy on CIFAR-10 with ResNet-18 obtained under different hyperparameter configurations of the meta-optimizer. Fig. 6a presents the test accuracy of varying LR values from 1×10^{-6} to 5×10^{-2} of meta-optimizer with fixed weight decay $1e{-}4$. It can be easily observed that LR ranging from 5×10^{-3} to 5×10^{-4} achieves almost similarly good performance. Besides, when the LR increases to larger than the order of 10^{-3}, it tends to certainly hamper the convergence tendency of the Adam optimizer, and thus lead to performance degradation. Furthermore, when learning rate decreases to smaller than the order of 10^{-4}, the updating speed of MLR-SNet becomes slower, and thus the produced learning rates incline to hardly adapt to the complicated training dynamics. This also leads to performance degradation. It is thus suggested to set the LR of the adopted Adam meta-optimizer in the range from 5×10^{-4} to 5×10^{-3} to facilitate users to more easily reproduce results of our MLR-SNet method.[3] It has also been substantiated to be an effective specification throughout all our experiments.

Fig. 6b presents the test accuracy of varying weight decay values from 1×10^{-6} to 5×10^{-2} of meta-optimizer with fixed LR $1e{-}3$. It can be seen from the figure that when the weight decay varies from 1×10^{-6} to 1×10^{-3}, similarly good performance can be consistently achieved. This complies with the commonly setting rage of weight decay for Adam optimizer in practice. When weight decay increases larger than the order of $\times 10^{-3}$, the performance becomes gradually degraded. This can be rationally explained by the fact that larger weight decay tends to decrease the norm of the meta-learner weights, and could be seen as increasing the effective learning rate of Adam optimizer to a certain extent, as illustrated in [70], [71]. As shown in the learning rate testing experiments, this inclines to slightly degrade the performance as it behaves in Fig. 6a. We thus simply suggest users to adopt weight decay 10^{-4}, which has been tested effective throughout all our experiments.

3. In the released codes of our algorithm, we have specifically set its default setting as 1×10^{-3}. Such a default learning rate setting has also been used for the Adam optimizer in PyTorch [70], and also has been commonly adopted by many previous literatures like [9], [71].



(a) (b)

Fig. 7. (a) The LR variation curves along iterations with the same input loss (we set it as 5) predicted by a single meta-learned MLR-SNet obtained at certain epoch of meta-training stage. As is shown, when iteration increases, the LR is almost constant. This implies that the meta-learned MLR-SNet at certain epoch fails to predict the long trajectories LR. (b) The recording test accuracy on CIFAR-100 with ResNet-18 using different meta-test strategies.

4.2 Meta-Test: Transferability and Generalization Capability of the LR Schedules Meta-Learned by MLR-SNet

As aforementioned, the meta-learned LR schedules are transferable and plug-and-play, attributed to its explicit parameterized mapping form. We then validate its transferability and generalization to new heterogeneous tasks.

4.2.1 Baselines

The L4, HD, RTHO methods learn the LR schedules specifically for given tasks, and they do not learn transferable structure allowing to be generalized to new tasks. We thus do not compare them in this part. The employed comparison methods for image classification include SGDM[4] with hand-designed LR schedules: 1) *Fixed* LR, 2) *Exponential* decay, 3) *MultiStep* decay, and 4) *SGDR*, as well as the adaptive gradient method *Adam*. As for the text classification experiments, we compare with SGD and Adam algorithm with Val Strategy LR schedule. The hyperparameters of these hand-designed LR schedules are tuned from scratch as strong baselines to show the task-transferable potential of our meta-learned MLR-SNet.[5]

We use the MLR-SNet meta-learned on CIFAR-10 with ResNet-18, as introduced in Section 4.1.1, as the plug-and-play LR schedules to directly predict the LR for SGD algorithm to new heterogeneous tasks. As discussed in Section 3.2.2, we save several meta-learned MLR-SNets at different epochs in the whole one meta-train run for helping setting LR schedules in the meta-testing stage. The motivation can be easily observed from Fig. 7a, which reveals that if we only use the single meta-learned MLR-SNet at certain epoch to predict LR, then the predicted LR will converge to a constant after several iterations. This implies that if we directly select one single MLR-SNet learned by our algorithm, it will raise the risk of the overfitting issue.

This thus inspired us to select more MLR-SNets learned during the meta-training iterations participating in meta-test process. Generally, if we want to select k nets for meta-test, the MLR-SNet learned at $[\frac{T*l}{k}]$th epoch ($l = 1, 2, \ldots, k$) should be chosen, where $[\cdot]$ denotes ceiling operator, and T is the iteration number in training. Fig. 7b show the test accuracy with ResNet-18 on CIFAR-100 of different test strategies, i.e., choosing different k MSR-SNets to transfer. It can be seen that once we choose more than three nets, similar performance can be obtained. We thus easily set k as 3 throughout all our experiments.

4.2.2 Generalization to Different Training Epochs

The plug-and-play MLR-SNet is meta-trained with epoch 200, and we transfer it to other different training epochs, e.g., 100, 400, 1200. All the methods are trained with ResNet-18 on CIFAR-100 with batch size 128 with varying epochs. The hyperparameter setting for compared hand-designed LR schedules is the same as that in Section 4.1.1 as illustrated above, except for MultiStep LR. For epoch 100, 400 and 1200, *MultiStep* decays LR by 10 every 30, 120, 360 epochs, respectively. For our method, we use the transferring MLR-SNet as below: 1) For epoch 100, we employ the 3 nets at 0-33, 33-67, 67-100 epoch, respectively; 2) For epoch 400, we employ the 3 nets at 0-133, 133-267, 267-400 epoch, respectively; 3) For epoch 1200, we employ the 3 nets at 0-400, 400-800, 800-1200 epoch, respectively.

As shown in Fig. 8, our MLR-SNet has the ability to train the SGD algorithm in the meta-test stage for longer horizons and achieves comparable performance as the best baseline MultiStep LR. The Fixed LR shakes at the later stage for the longer epochs. This substantiated that the learned MLR-SNet is capable of generalized to setting LR schedules with such longer horizons problems.

4.2.3 Generalization to Different Datasets

We transfer the LR schedules meta-learned on CIFAR-10 to SVHN [72], TinyImageNet,[6] and Penn Treebank [67] datasets to validate the generalization of our method to different datasets, especially varying data modalities. For SVHN and TinyImageNet datasets, we train a ResNet-18 with 200 epoch. For Penn Treebank classification, we train a 3-layer LSTM with 150 epoch. The hyperparameters of all compared methods are with the same setting as CIFAR-10 and Penn Treebank introduced in Section 4.1. The results are presented in Fig. 9. It is worth noting that the LR schedules for image task and text task have different forms, while our MLR-SNet can still obtain a relatively stable and comparable generalization performance for different tasks with the corresponding best baseline methods.

4.2.4 Generalization to Different Net Architectures

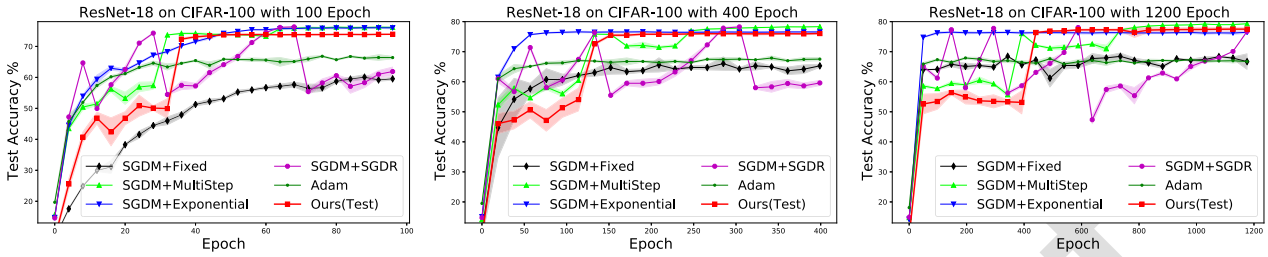
To further validate that our method can be applied to different network architectures, we also transfer the LR schedules meta-learned on ResNet-18 to ShuffleNetV2 [73], MobileNetV2 [74] and NASNet [75].[7] These network architectures are different from ResNet-type network, especially the NASNet is learned from data, not the artificial constructed network. As shown in Fig. 10, our method can achieve comparable results and even get better performance to the best baseline method. This

4. Here we present stronger baseline results compared with trained with SGD, while our MLR-SNet still predicts LR schedules for SGD.

5. A fair task-tranferable setting for these hand-designed LR schedules is to assemble hyperparameters tuning on CIFAR-10 with ResNet-18 to train new query tasks. More details refer to Supplementary Material, available online.

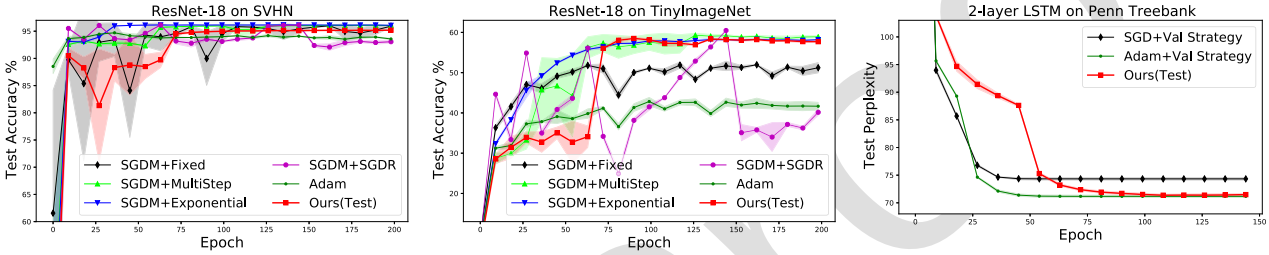
6. It can be downloaded at https://tiny-imagenet.herokuapp.com

7. The pytorch codes of all these networks can be found on https://github.com/weiaicunzai/pytorch-cifar100



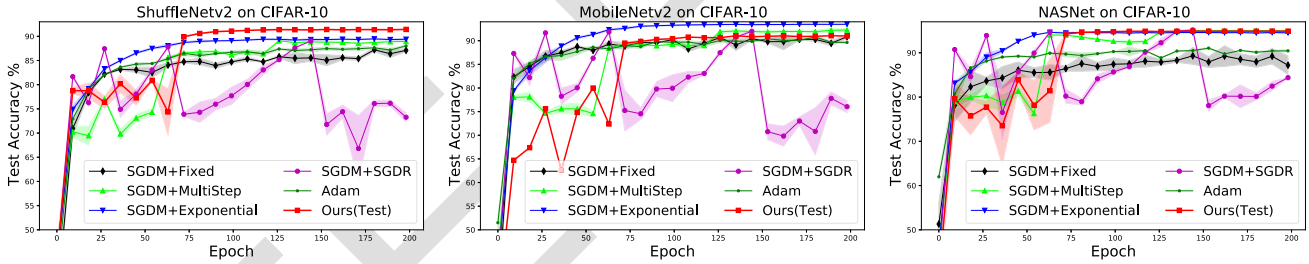
(a) Training with 100 Epochs (b) Training with 400 Epochs (c) Training with 1200 Epochs

Fig. 8. Test accuracy on CIFAR-100 of ResNet-18 with *varying epochs* for our transferred MLR-SNet in the meta-test stage.



(a) SVHN dataset (b) TinyImageNet dataset (c) Penn Treebank dataset

Fig. 9. Test accuracy with *different datasets* for our transferred MLR-SNet in the meta-test stage.



(a) ShuffleNetV2 (b) MobileNetV2 (c) NASNet

Fig. 10. Test accuracy on CIFAR-10 with *different network architectures* for our transferred MLR-SNet in the meta-test stage.

further shows that our MLR-SNet is able to be transferred to varying types of network training. We also transfer to EfficientNet [76], and the experimental details can be found in the supplementary material, available online.

4.2.5 Generalization to Large Scale Optimization Problem

In this part, we attempt to use the meta-learned LR schedules to train DNN on ImageNet dataset [77]. To our best knowledge, only [40] had attempted this task among existing learning-to-optimize literatures. However, it can only be executed for thousands of steps, and then its loss begins to increase dramatically, thus not able to be implemented in the optimization process in practice. We transfer the LR schedules meta-trained on CIFAR-10 with ResNet-18 to ImageNet dataset with ResNet-50.[8] All compared methods are trained by SGDM with a momentum 0.9, a weight decay $5e^{-4}$, an initial learning rate 0.1 for 90 epochs, and batch size 256. *MultiStep* decays LR by 10 every 30 epochs; *Exponential* multiplies LR with $\gamma_E = 0.95$ every epoch; *SGDR* sets $\alpha_{\min} = 1e^{-5}, \alpha_{\max} = 0.1$, and $E_0 = 10, T_{Mult} = 2$. Following [66], we decay LR by 10 every 30 epochs for Adam.

The test accuracy on ImageNet validation set is presented in Fig. 12a. It can be seen that the performance of our method is competitive with those hand-designed LR schedules methods, though we train the model with SGD using the LR schedules predicted by our transferred MLR-SNets. Meanwhile, the LR schedules predicted by MLR-SNet brings non-extra computation complexity in the DNN training process. This implies that our method is hopeful to be effectively and efficiently used to deal with such large scale optimization problems, making learning-to-optimize ideas towards more practical applications.

4.3 How do Meta-Training Tasks Influence the Generalization Performance of Meta-Learned LR Schedules

In this section, we empirically study how meta-training tasks influence the generalization performance of meta-learned LR schedules. To conduct ablation study for answering this question, we construct three groups of meta-training tasks to character the influence factors for the generalization performance. An overview of them is shown in Table 4. The meta-test task is

8. The training codes of baseline methods can be found on https://github.com/pytorch/examples/tree/master/imagenet

TABLE 4
Variants Constructed From Meta-Training Tasks

Influence factors	Tasks design
Task similarity	MNIST[a], SVHN[a], CIFAR-10
Task scale	1/10 CIFAR-100[b], 1/2 CIFAR-100[b], CIFAR-100
Architecture	ResNet-18, ResNet-34, ResNet-50

a: uniformly downsample to 50000 samples.
b: uniformly sample to certain proportion of full CIFAR-100.

set as training a ResNet-18 on full CIFAR-100 with meta-learned LR schedules. The hyperparameter setting follows those introduced in Section 4.1.1.

The Similarity Between Meta-Training and Meta-Test Tasks. Grayscale digits (MNIST), RGB digits (SVHN) and natural photos (CIFAR-10) represent incremental similarity between meta-training and meta-test tasks. We use the three datasets to meta-learn MLR-SNet with ResNet-18, respectively. As shown in Fig. 11a, three transferred LR schedules meta-learned from different datasets achieve very similar final performance on the meta-test task. This validates that such similarity difference has a relatively weak influence on the generalization of meta-learned LR schedules.

Scale of Meta-Training Tasks. The scale of meta-training tasks is also taken into consideration. We uniformly sampled 50, 250, 500 samples per class in CIFAR-100 as training datasets, denoted by 1/10 CIFAR-100, 1/2 CIFAR-100 and CIFAR-100, respectively. We use the three datasets to meta-learn MLR-SNet with ResNet-18. Fig. 11b shows the generalization performance of three kinds of such meta-learned LR schedules. As is shown, the performance deteriorates when the size of training task set is small. If the scale of training task set is in the same order of magnitude, it tends to obtain similar generalization performance.

Architectures of Training Models. Different network architectures in the meta-training stage may produce different LR schedules. We adopt three different classifier networks, including ResNet-18, ResNet-34, and ResNet-50, to meta-learn MLR-SNet on CIFAR-100. Fig. 11c shows that three transferred LR schedules achieve similar generalization performance, even though they are meta-learned based on different classifier networks.

Remark. We have empirically verified that the generalization performance of the meta-learned LR schedules is not sensitive to the similarity between meta-training and meta-test tasks, and network architectures in the meta-training stage. This can be rationally explained by the fact that our MLR-SNet is sufficiently simple to make it less rely on the task-related information. Besides, it is also verified that the size of meta-training task could slightly influence the final generalization performance. This might possibly due to that few meta-training data could not provide enough information to fit the proper LR schedules. Furthermore, these empirically results state that our MLR-SNet is easy to be meta-trained for achieving an admirable performance on the meta-test tasks.

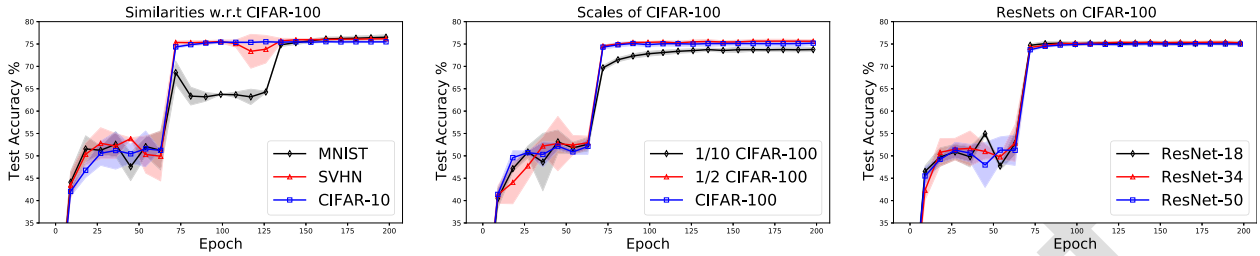
4.4 Robustness on Data Corruptions

In this section, we further validate whether our MLR-SNet behaves robust against corrupted training data guided by a clean validation set. To this aim, we design experiments as follows: we take CIFAR-10-C and CIFAR-100-C [78] as our training set,[9] consisting of 15 types of algorithmically generated corruptions from noise, blur, weather, and digital categories. These corruptions contain Gaussian Noise, Shot Noise, Impulse Noise, Defocus Blur, Frosted Glass Blur, Motion Blur, Zoom Blur, Snow, Frost, Fog, Brightness, Contrast, Elastic, Pixelate and JPEG. All the corruptions are generated on 10,000 test set images of CIFAR-10/100 dataset, and each corruption contains 50,000 images since each type of corruption has five levels of severity. We treat CIFAR-10-C or CIFAR-100-C dataset as training set, and the original training set of CIFAR-10 or CIFAR-100 as test set. We train models with ResNet-18 for each corrupted dataset. Finally, we can obtain 15 models for CIFAR-10-C or CIFAR-100-C dataset. The average accuracy of 15 models on test data is used to evaluate the robust performance of each LR schedules strategy. All compared hand-designed LR schedules are trained with a ResNet-18 by SGDM with a momentum 0.9, a weight decay $5e^{-4}$, an initial learning rate 0.1 for 100 epochs, and batch size 128. *Exponential* LR multiplies LR with 0.95 every epoch; *MultiStep* LR decays LR by 10 every 30 epochs; *SGDR* sets $\alpha_{\min} = 1e^{-5}, \alpha_{\max} = 0.1$, and $E_0 = 10, T_{Mult} = 2$; *Adam* just uses the default parameter setting. We update the MLR-SNet under the guidance of a small set of validation set without corruptions, to guarantee that the final learned models finely generalize to clean test set. We randomly choose 10 clean images for each class as validation set in this experiment.

Table 5 shows the mean test accuracy of 15 models ($\pm$std) on the training set of CIFAR-10 or CIFAR-100 dataset. As can be seen, our proposed MLR-SNet is capable of achieving better generalization performance on clean test data than baseline methods, which implies that our method behaves more robust and stable than the pre-set LR schedules when the learning tasks in which the distribution of training and test data are mismatched. This is due to the fact that our MLR-SNet has more flexibility to adapt the variation of the data distribution than the pre-set LR schedules, and it can find a proper LR schedule through minimizing the generalization error which is based on the knowledge specifically conveyed from the given validation data.

Furthermore, we attempt to explore the generalization of our meta-learned LR schedules. Different from the above experiments where all 15 models are trained under the guidance of a small set of validation set, we just meta-learn the MLR-SNet on the Gaussian Noise corruption dataset, and then transfer the meta-learned LR schedules to other 14 corruptions datasets. We report the average accuracy of 14 models on test data to show the robust performance of our transferred LR schedules. All the methods are meta-tested with a ResNet-18 for 100 epochs with batch size 128. The hyperparameter setting of hand-designed LR schedules keeps the same as above. Table 6 shows the mean test accuracy of 14 models on the training set of CIFAR-10 or CIFAR-100 dataset. As can be seen, our transferred LR schedules obtain the best performance in the last epoch compared with

9. They can be downloaded at https://zenodo.org/record/2535967#.Xt4mVigzZPY and https://zenodo.org/record/3555552#.Xt4mdSgzZPY



(a) Task similarity with CIFAR-100 (b) Scale of meta-training tasks (c) Architectures of training models

Fig. 11. Illustration of meta-training tasks influencing the generalization performance of meta-Learned LR schedules.

TABLE 5
Test Accuracy (%) on CIFAR-10 and CIFAR-100 Training Sets of Compared Models Trained on CIFAR-10-C and CIFAR-100-C with Hand-Designed LR Schedules and Meta-Trained MLR-SNet (Meta-Training)

Datasets/Methods		Fixed	MultiStep	Exponential	SGDR	Adam	Ours(Train)
CIFAR-10-C	Best	79.78 ± 3.95	85.52 ± 1.72	83.48 ± 1.45	85.94 ± 1.52	81.45 ± 1.42	$\mathbf{86.04 \pm 1.51}$
	Last	77.88 ± 3.91	85.36 ± 1.71	83.32 ± 1.43	78.21 ± 2.01	80.29 ± 1.64	$\mathbf{85.87 \pm 1.54}$
CIFAR-100-C	Best	46.74 ± 3.03	52.26 ± 2.58	49.72 ± 1.97	52.54 ± 2.49	45.45 ± 1.94	$\mathbf{52.56 \pm 2.26}$
	Last	44.79 ± 3.91	52.16 ± 2.59	49.58 ± 1.98	41.58 ± 3.24	43.76 ± 2.22	$\mathbf{52.42 \pm 2.34}$

Best and Last denote the best test result and the last epoch test result, respectively. The Bold *and* Underline Bold *denote the first and second best results, respectively.*

TABLE 6
Test Accuracy (%) on CIFAR-10 and CIFAR-100 Training Sets of Compared Models Trained on CIFAR-10-C and CIFAR-100-C with Hand-Designed LR Schedules and Transferred MLR-SNet (Meta-Test)

Datasets/Methods		Fixed	MultiStep	Exponential	SGDR	Adam	Ours(Test)
CIFAR-10-C	Best	79.96 ± 4.09	85.64 ± 1.71	83.63 ± 1.38	$\mathbf{86.10 \pm 1.44}$	81.57 ± 1.39	$\underline{\mathbf{85.73 \pm 1.71}}$
	Last	77.89 ± 4.05	$\underline{\mathbf{85.48 \pm 1.71}}$	83.47 ± 1.37	78.46 ± 1.92	80.39 ± 1.65	$\mathbf{85.62 \pm 1.76}$
CIFAR-100-C	Best	46.91 ± 3.08	52.38 ± 2.43	49.90 ± 1.93	$\mathbf{52.80 \pm 2.39}$	45.58 ± 1.95	$\underline{\mathbf{52.51 \pm 2.38}}$
	Last	44.81 ± 5.98	$\underline{\mathbf{52.28 \pm 2.44}}$	49.75 ± 1.94	41.68 ± 3.33	43.94 ± 2.18	$\mathbf{52.35 \pm 2.46}$

Best and Last denote the best test result and the last epoch test result, respectively. The Bold *and* Underline Bold *denote the first and second best results, respectively.*

hand-designed LR schedules. This implies that our transferred LR schedules can also perform robust and stable for the learning tasks in which the distribution of training and test data are mismatched. Besides, our transferring LR schedules are plug-and-play, and have no additional hyperparameters to tune when transferred to new heterogeneous tasks.

5 Further Analysis on MLR-SNet

In this section, we first provide the convergence guarantee for the SGD algorithm with LR schedules produced by our MLR-SNet, as well as the convergence guarantee for the meta-learning of the MLR-SNet (Section 5.1). In Section 5.2, we further analyze the computational complexity for the MLR-SNet. The "width" of the solution is visualized in Section 5.3. In Section 5.4, we further verify that the LSTM-type meta-learner behaves more superiorly than MLP-type meta-learner. Finally, we show that the MLR-SNet can be applied to Adam optimizer in Section 5.5.

5.1 Convergence Analysis of MLR-SNet

The preliminary experimental evaluations show that our method gives good convergence performance on various tasks. We find that the meta-learned LR schedules in our experiments follow a consistent trajectory as shown in Fig. 1, almost obeying a decay LR form. Without loss of generality, we assume that the learning rate can be represented by

$$\alpha_t = \alpha_{t-1}\beta_t, \ t = 1, 2, \ldots, T, \tag{14}$$

where α_t denotes the learning rate predicted by MLR-SNet at the tth iteration, and β_t denotes the decay factor at the tth iteration, $1/K \leq a \leq \beta_t \leq b \leq 1$, where $a = (M/T)^{1/T}, b = (N/T)^{1/T}$, and $a \neq b, M, N \propto T$, and K is the arbitrarily large constant. We denote by $\mathbb{E}[\cdot]$ the expectation with respect to the underlying probability space. To present the convergence results, we also assume that[10]:

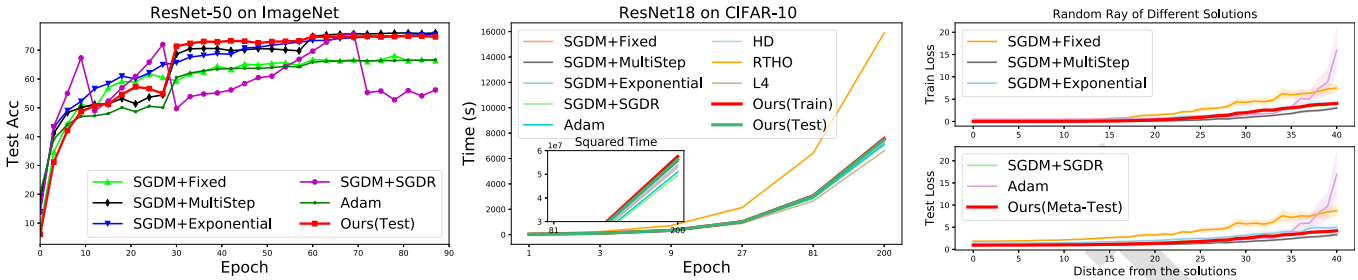
(*A1*) The loss function $f(w) : \mathbb{R}^d \to \mathbb{R}$ is L-smooth, i.e., f is differentiable and its gradient $\nabla f(w)$ is L-Lipschitz.

(*A2*) f satisfies the μ-PL condition, that is, their exists some $\mu > 0$, $\frac{1}{2}\|\nabla f(w)\|^2 \geq \mu(f(w) - f^*)$, holds for any w, where f^* represents the infimum of $f(w)$.

(*A3*) For $t = 1, 2, \ldots, T$, we assume $\mathbb{E}_t[\|v_t - \nabla f(w_t)\|^2] \leq \kappa\|\nabla f(w_t)\|^2 + \sigma$, where $\kappa, \sigma > 0$, and v_t is an unbiased estimate of the gradient of f at point w_t, i.e., $\mathbb{E}_t v_t = \nabla f(w_t)$.

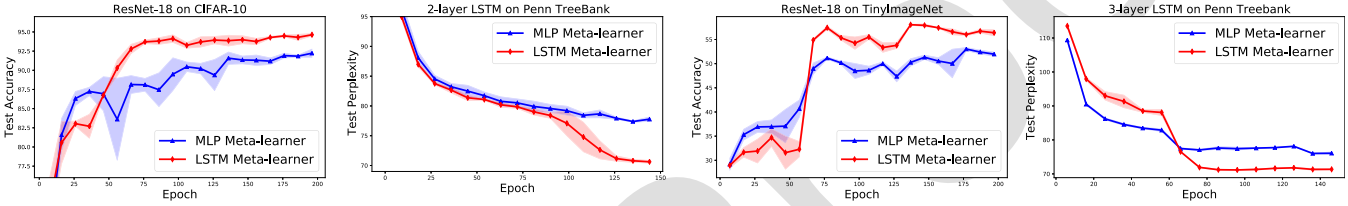
First, we consider the case where the function is smooth and satisfies the Polyak-Lojasiewicz (PL) condition [82], [83]. The proofs of all Theorems are listed in the appendix

10. They are commonly used for existing SGD convergence theories [79], [80], [81].



(a) (b) (c)

Fig. 12. (a)Test accuracy on ImageNet validation set with ResNet-50. (b)Computational time costed by different LR schedule methods. (c) (*Upper*) Train lossf and (*Lower*) test loss as a function of a point on a random ray starting at the solutions for different methods on CIFAR-100 with ResNet-18.



(a) Meta-training Results (b) Meta-test Results

Fig. 13. Performance comparison of two types of meta-learners. (a) Two types of meta-learners are trained on CIFAR-10 and Penn Treebank datasets following the experiment setting in Section 4.1. The figure presents the test performance of two tasks. (b) The LR schedules meta-learned on CIFAR-10 is transferred to TinyImageNet and Penn Treebank datasets following the experiment setting in Section 4.2. The meta-test performance are shown in the figure.

file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3184315.

Theorem 1. *Assume* (A1,A2,A3) *hold, and the SGD is with learning rate (14), where* $\alpha_0 = (L(1+\kappa))^{-1}$. *Then for a given* $T \geq \max\{3, M, N\}$, *the* w_t *generated from SGD satisfies*

$$\mathbb{E}f(\mathbf{w}_{t+1}) - f^* \leq C(M)\exp\left(-\frac{\mu T}{KL(1+\kappa)\ln(T/M)}\right) (f(\mathbf{w}_1) - f^*) + \frac{2K^2C(M)\ln^2(T/M)(N/M)^2}{e^2\mu^2(1-M/N)T}.$$

where $C(M) = \exp(\frac{\mu M}{KL(1+\kappa)\ln(T/M)})$.

Theorem 1 states that SGD with learning rate produced by our MLR-SNet as described in Eq. (14) can obtain an approximately linear convergence rate, achieving the best-known rates for the non-convex optimization [79]. While the assumption (A2) means that all stationary points are optimal point, which is not always true for deep learning, the following theorem discusses the case where the PL condition is not satisfied.

Theorem 2. *Assume* (A1,A3) *hold, and the SGD is with learning rate (14), where* $\alpha_0 = (cL(1+\kappa))^{-1}, c > 1$. *Then for* w_t *generated using SGD, we have the following bound*

$$\min_t \mathbb{E}\|\nabla f(w_t)\|^2 \leq \frac{2cKL(1+\kappa)\ln(T/M)}{T-M}[\mathbb{E}f(w_1) - \mathbb{E}f(w_T)] + \mathcal{O}\left(\frac{\sigma KT}{c(1+\kappa)(T-M)}\right).$$

It can be seen that when $\sigma \neq 0$, if we set $c \propto \sqrt{T}$ and $\sigma = \mathcal{O}(1)$, it would give the $\mathcal{O}(1/\sqrt{T})$ rate; when $\sigma = 0$, if we set $c = \mathcal{O}(1)$, it would give the $\mathcal{O}(1/T)$. It is worth noting that the condition $\sigma = 0$ holds in many practical scenarios, e.g., [84]. On the other hand, we provide a convergence analysis of the MLR-SNet updated by the validation loss.

Theorem 3. *Assume* (A1,A3) *hold,* f *has* ρ*-bounded gradients with respect to training/validation data, and the* $\mathcal{A}(\theta)$ *is differential with a* δ*-bounded gradient and twice differential with its Hessian bounded by* $\mathcal{B}$. *Assume that the learning rate* $\alpha_t = \mathcal{A}(\theta_t)$ *predicted by MLR-SNet obey Eq. (14). We suppose that the learning rate of Adam algorithm for updating MLR-SNet satisfies* $\eta_t = \eta$ *for all* $t \in [T], \eta \leq \frac{\epsilon}{2L}$ *and* $1 - \beta_2 \leq \frac{\epsilon^2}{16\rho^2}$, *where* β_2, ϵ *are the hyperparameters of the Adam algorithm (It can be found in Appendix, available in the online supplemental material). Then for* θ_t *generated using Adam, we have the following bound:*

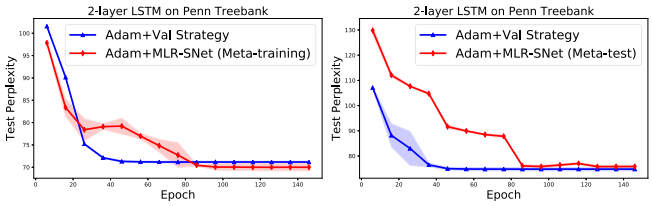
$$\min_{0\leq t\leq T} \mathbb{E}[\|\nabla \mathcal{L}_{Val}(\hat{\mathbf{w}}_t(\theta_t))\|_2^2] \leq \mathcal{O}(\frac{1}{c^2\ln(T)} + \sigma^2). \quad (15)$$

It can be seen that when $\sigma \neq 0$, if we set $c \propto \sqrt{T}$, and $\sigma = \mathcal{O}(1)$, it would lead to the $\mathcal{O}(\frac{1}{T\ln(T)} + \sigma^2)$ convergence rate; when $\sigma = 0$, if we set $c = \mathcal{O}(1)$, it would give the $\mathcal{O}(\frac{1}{\ln(T)})$ convergence rate. It can then be proved that the convergence of the proposed method.

Theorems 1 and 2 proposed in the paper are inspired by the theoretical analysis for standard SGD optimizers, but under different learning rate assumptions from conventional. Specifically, classical convergence analysis on the SGD algorithm requires pre-assume the conditions on the stepsize sequence $\alpha_t, t = 1, \ldots, \infty$ that they should satisfy that

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \text{and } \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \quad (16)$$

Different from these conventional assumptions, our theory further considers the relationship between subsequent stepsizes α_{t-1} and α_t. The decay factor β_t so assumed is varied



(a) Meta-training Results (b) Meta-test Results

Fig. 14. Applying MLR-SNet on Top of Adam Algorithm. (a) The MLR-SNet for Adam is meta-trained on Penn Treebank datasets with 2-layer LSTM following the experiment setting in Section 4.1. The figure present the test perplexity. (b) The meta-learned LR schedules are transferred to train the three-layer LSTM on Penn Treebank dataset. The test perplexity is depicted.

along algorithm iterations, complying with the abundant and dynamic variations along with the training process delivered by the LSTM meta-learner, as clearly shown in Figs. 3 and 4. Besides, Theorem 3 presents the convergence analysis of the LSTM-based meta-learner. This is also different from the convergence analysis for standard SGD optimizers, which aims to solve a single level optimization problem. Comparatively, our analysis is built on the bilevel optimization, and thus needs to evaluate the convergence property of meta-level optimization problem. Thus this result is specific to the proposed meta-learning problem beyond conventional SGD convergence analysis.

5.2 Computational Complexity Analysis

In the meta-training stage, our MLR-SNet learning algorithm can be roughly regarded as requiring two extra full forward and backward passes of the network (step 6 in Algorithm 1) in the presence of the normal network parameters update (step 8 in Algorithm 1), together with the forward passes of MLR-SNet for every LR. Therefore compared to normal training, our method needs about $3\times$ computation time for one iteration. Since we periodically update MLR-SNet after several iterations, this will not substantially increase the computational complexity compared with normal network training. In the meta-test stage, our transferred LR schedules predict LR for each iteration by a small MLR-SNet (step 4 in Algorithm 2), whose computational cost should be significantly less than the cost of the normal network training. To empirically show the computational complexity differences between baselines and our MLR-SNet, we conduct experiments with ResNet-18 on CIFAR-10 and report the running time for all methods. All experiments are implemented on a computer with Intel Xeon (R) CPU E5-2686 v4 and a NVIDIA GeForce RTX 2080 8GB GPU. We follow the corresponding settings in Section 4.1, and results are shown in Fig. 12b. It is seen that except that *RTHO* costs significantly more time, our MLR-SNet takes a similar time to complete the meta-training and meta-test phase compared to hand-designed LR schedules with hyperparameters found by search methods (the computation cost of searching hyperparameters is not included). Considering its good transferability and generalization capability, it should be rational to say that it is efficient.

5.3 Visualizing the "Width" of Solutions

We further point out that visualizing the "width" of a given solution w in a low-dimensional space may help understand why the model has fine generalization capability. Generally, [10], [11] suggested that the wider optima leads to better generalization. We use the visualization technique in [13] to show how the loss changes along many random directions drawn from the d-dimensional Gaussian distribution. Fig. 12c visualizes the "width" of the solutions learned on CIFAR-100 with ResNet-18 for different LR schedules. It can be seen that our method, as well as the competitive baselines, lies in a wide flat region of the train loss. This could explain why they achieve better generalization performance. Deeper understanding of this point will be further investigated.

5.4 Why do we Need LSTM Meta-Learner

We regard scheduling LR as a long-term information dependent problem, and thus we parameterize the LR schedules as an LSTM network. As we know MLP (multilayer perceptron) network can also learn an explicit mapping but ignores the temporal information, here we compare the performance of the two types of meta-learners. Fig. 13 compares the performance of two types of meta-learners for both meta-training and meta-test procedures. As is shown, the MLP meta-learner achieves better performance in the early learning stage for both meta-training and meta-test procedure. While at the later training stage, the LSTM meta-learner gradually brings a notable performance increase compared with the MLP meta-learner. This might be possibly due to that the MLP meta-learner easily falls into the local optimal LR learning, while lacking in considering the overall significantly changed training dynamics. Though MLP meta-learner can also depict the loss-LR relationship, it ignores the more important training dynamics information involved for the scheduling LR. The LSTM meta-learner, however, is capable of accumulating temporal information on complicated training dynamics, and thus inclines to help find a more proper LR schedule for such DNNs training.

5.5 Applying MLR-SNet on Top of Adam Algorithm

To further demonstrate the versatility of our method, we apply the MLR-SNet on top of the Adam algorithm. Fig. 14 shows that our method can help find better LR schedules than the Val Strategy. And the transferred LR schedules can also attain comparable performance with the hand-designed LR schedules. This implies that our framework is hopeful to learn the proper LR schedules for various optimizers.

6 Conclusion and Discussion

In this paper, we have proposed to learn an adaptive and transferrable LR schedule in a meta learning manner. To this aim, we have designed an LSTM-type meta-learner (MLR-SNet) to parameterize LR schedules, which gives more flexibility to adaptively learn a proper LR schedule to comply with the complex training dynamics of DNNs. Meanwhile, the meta-learned LR schedules are plug-and-play and transferrable, which can be readily transferred to schedule LR for SGD to new heterogeneous tasks. Comprehensive experiments have been implemented, and the results substantiate the superiority of our method on various image and text benchmarks in its adaptability, transferability and robustness, as compared with current LR schedules policies. The MLR-SNet is hopeful to be useful in practical problems as it requires a negligible increase in the parameter size and computation time, and a small transferrable cost for new tasks. We will make a further endeavor to further ameliorate our proposed method to make it a general and

useful tool for helping improve current DNN training. More practical applications will also be attempted to further verify its effectiveness in general learning tasks. It is also an important issue to further enhance the convergence theory and especially discover new skills and techniques useful to general meta-learning algorithms for this line of research. Provable transferable LR schedules theory, as done for methodology-level transfer [1], is also of interest.

REFERENCES

[1] J. Shu, D. Meng, and Z. Xu, "Learning an explicit hyperparameter prediction policy conditioned on tasks," 2021, *arXiv:2107.02378*.

[2] J. Shu, X. Yuan, D. Meng, and Z. Xu, "Cmw-net: Learning a class-aware sample weighting mapping for robust deep learning," 2022, *arXiv:2202.05613*.

[3] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. Jul., pp. 2121–2159, 2011.

[5] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.

[6] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *Neural Netw. Mach. Learn.*, vol. 4, pp. 26–31, 2012.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–15.

[8] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–15.

[9] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural Comput.*, vol. 9, no. 1, pp. 1–42, 1997.

[10] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–16.

[11] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1019–1028.

[12] L. Wu, C. Ma, and E. Weinan, "How SGD selects the global minima in over-parameterized learning: A dynamical stability perspective," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 8289–8298.

[13] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," in *Proc. Conf. Uncertainty Artif. Intell.*, 2018, pp. 876–885.

[14] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6391–6401.

[15] S. Jastrzebski *et al.*, "Three factors influencing minima in SGD," 2017, *arXiv:1711.04623*.

[16] F. He, T. Liu, and D. Tao, "Control batch size and learning rate to generalize well: Theoretical and empirical evidence," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1143–1152.

[17] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Berlin, Germany: Springer, 2012, pp. 437–478.

[18] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 343–351.

[19] K. Nar and S. Sastry, "Step size matters in deep learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 3440–3448.

[20] K. Liu, L. Ziyin, and M. Ueda, "Stochastic gradient descent with large learning rate," 2020, *arXiv:2012.03636*.

[21] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik, "SGD: General analysis and improved rates," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5200–5209.

[22] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–16.

[23] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[24] R. Ge, S. M. Kakade, R. Kidambi, and P. Netrapalli, "The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, Art. no. 1341.

[25] D. Davis, D. Drusvyatskiy, and V. Charisopoulos, "Stochastic algorithms with geometric step decay converge linearly on sharp functions," 2019, *arXiv:1907.09547*.

[26] J. D. Lee, I. Panageas, G. Piliouras, M. Simchowitz, M. I. Jordan, and B. Recht, "First-order methods almost always avoid saddle points," *Math. Program.*, vol. 176, pp. 311–337, 2019.

[27] I. Panageas, G. Piliouras, and X. Wang, "First-order methods almost always avoid saddle points: The case of vanishing step-sizes," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 6471–6480.

[28] M. Rolinek and G. Martius, "L4: Practical loss-based stepsize adaptation for deep learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6434–6444.

[29] L. Berrada, A. Zisserman, and M. P. Kumar, "Deep Frank-Wolfe for neural network optimization," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–32.

[30] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, Art. no. 335.

[31] J. Nocedal and S. Wright, *Numerical Optimization*. Berlin, Germany: Springer, 2006.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] L. B. Ward, "Reminiscence and rote learning," *Psychol. Monographs*, vol. 49, no. 4, p. i, 1937.

[34] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behav. Brain Sci.*, vol. 40, 2017, Art. no. e253.

[35] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Comput.*, vol. 4, no. 1, pp. 131–139, 1992.

[36] Y. Bengio, S. Bengio, and J. Cloutier, "Learning a synaptic learning rule," in *Proc. Int. Joint Conf. Neural Netw.*, 1991, pp. 969–975.

[37] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Proc. Int. Conf. Artif. Neural Netw.*, 2001, pp. 87–94.

[38] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3988–3996.

[39] Y. Chen *et al.*, "Learning to learn without gradient descent by gradient descent," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 748–756.

[40] O. Wichrowska *et al.*, "Learned optimizers that scale and generalize," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3751–3760.

[41] K. Li and J. Malik, "Learning to optimize neural nets," 2017, *arXiv:1703.00441*.

[42] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–11.

[43] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2247–2255.

[44] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[45] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few-shot learning," 2017, *arXiv:1707.09835*.

[46] E. Park and J. B. Oliva, "Meta-curvature," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, Art. no. 298.

[47] R. Houthooft *et al.*, "Evolved policy gradients," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5405–5414.

[48] N. Rosenfeld, E. Balkanski, A. Globerson, and Y. Singer, "Learning to optimize combinatorial functions," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4374–4383.

[49] T. Wang, Y. Wu, D. Moore, and S. J. Russell, "Meta-learning MCMC proposals," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4150–4160.

[50] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Berlin, Germany: Springer, 2019.

[51] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[52] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.

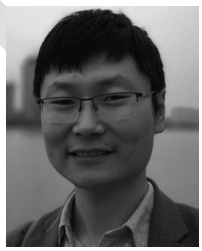
[53] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1165–1173.

[54] J. Shu, Q. Zhao, K. Chen, Z. Xu, and D. Meng, "Learning adaptive loss for robust learning with noisy labels," 2020, *arXiv:2002.06482*.

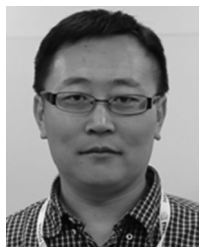
[55] J. Shu, Q. Zhao, Z. Xu, and D. Meng, "Meta transition adaptation for robust deep learning with noisy labels," 2020, *arXiv:2006.05697*.
[56] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–11.
[57] Y. Liu, B. Schiele, and Q. Sun, "An ensemble of epoch-wise empirical bayes for few-shot learning," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 404–421.
[58] Y. Wu, M. Ren, R. Liao, and R. Grosse, "Understanding short-horizon bias in stochastic meta-optimization," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–17.
[59] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[60] Q. Yang, Y. Zhang, W. Dai, and S. J. Pan, *Transfer Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2020.
[61] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964.
[62] P. Tseng, "An incremental gradient (-projection) method with momentum term and adaptive stepsize rule," *SIAM J. Optim.*, vol. 8, no. 2, pp. 506–531, 1998.
[63] J. Shu, Z. Xu, and D. Meng, "Small sample learning in big data era," 2018, *arXiv:1808.04572*.
[64] J. Shu *et al.*, "Meta-weight-net: Learning an explicit mapping for sample weighting," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1917–1928.
[65] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016, pp. 1–15.
[66] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4151–4161.
[67] M. P. Marcus and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The penn treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
[68] 2021. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam
[69] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–19.
[70] E. Hoffer, R. Banner, I. Golan, and D. Soudry, "Norm matters: Efficient and accurate normalization schemes in deep networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 2164–2174.
[71] G. Zhang, C. Wang, B. Xu, and R. Grosse, "Three mechanisms of weight decay regularization," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–16.
[72] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NeurIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.
[73] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 122–138.
[74] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[75] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.
[76] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[78] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–16.
[79] H. Karimi, J. Nutini, and M. Schmidt, "Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2016, pp. 795–811.
[80] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 314–323.
[81] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.
[82] B. T. Polyak, "Gradient methods for minimizing functionals," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, vol. 3, no. 4, pp. 643–653, 1963.
[83] S. Lojasiewicz, "A topological property of real analytic subsets," *Coll. du CNRS, Les Équations aux Dérivées Partielles*, vol. 117, pp. 87–89, 1963.
[84] S. Vaswani, F. Bach, and M. Schmidt, "Fast and faster convergence of SGD for over-parameterized models and an accelerated perceptron," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 1195–1204.

Jun Shu received the BE degree from Xi'an Jiaotong University, Xi'an, China, in 2016, where he is currently working toward the PhD degree, under the tuition of Prof. Deyu Meng and Prof. Zongben Xu. His current research interests include machine learning and computer vision, especially on meta learning, robust deep learning, and AutoML.

Yanwen Zhu received the BSc and MSc degrees from Xi'an Jiaotong University, Xi'an, China, in 2018 and 2021, respectively. His research interests include meta learning and hyper-parameter optimization.



Qian Zhao received the BSc and PhD degrees from Xi'an Jiaotong University, Xi'an, China, in 2009 and 2015, respectively. He was a visiting scholar with Carnegie Mellon University, Pittsburgh, Pennsylvania, from 2013 to 2014. He is currently an associate professor with the School of Mathematics and Statistics, Xi'an Jiaotong University. His current research interests include low-rank matrix/tensor analysis, Bayesian modeling, and meta learning.



Deyu Meng received the BSc, MSc, and PhD degrees from Xi'an Jiaotong University, Xi'an, China, in 2001, 2004, and 2008, respectively. He was a visiting scholar with Carnegie Mellon University, Pittsburgh, Pennsylvania, from 2012 to 2014. He is currently a professor with the School of Mathematics and Statistics, Xi'an Jiaotong University, and an adjunct professor with the Faculty of Information Technology, Macau University of Science and Technology, Taipa, Macau, China. His research interests include model-based deep learning, variational networks, and meta learning.

Zongben Xu received the PhD degree in mathematics from Xi'an Jiaotong University, Xi'an, China, in 1987. He currently working as the academician of the Chinese Academy of Sciences, the chief scientist of the National Basic Research Program of China (973 Project), and the director of the Institute for Information and System Sciences with Xi'an Jiaotong University. His current research interests include nonlinear functional analysis and intelligent information processing. He was a recipient of the National Natural Science Award of China, in 2007, and the winner of the CSIAM Su Buchin Applied Mathematics Prize, in 2008.