MLR-SNet: Transferable LR Schedules for Heterogeneous Tasks: Supplementary Material

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

Abstract—In this supplementary material, we present more related experiment details, and more necessary performance evaluations of the proposed method as compared with other competing methods. We also provide the proof details for the convergence analysis of the proposed MLR-SNet method. Besides, we demonstrate the pseudo-code of the MLR-SNet for Pytorch implementation. The source code of our method is released at https://github.com/xjtushujun/MLR-SNet.

1 Detailed comparisons with Meta-SGD

The Meta-SGD algorithm [1] meta-learns the learning rates in an end-to-end manner, which aims to learn possibly fast and accurate meta-learners. Specifically, similar to our method, the LR schedule learned by Meta-SGD can also be readily transferred for new query tasks, and has been substantiated to be effective in applications like few-shot learning tasks [1]. Since this method is mostly related to our MLR-SNet, we make a detailed comparison between the two methods in the following for better illuminating the specific characteristics of the proposed method:

- **Optimized variables.** Recall that the updating equations of the vanilla SGD algorithm for Meta-SGD and MLR-SNet can be respectively written as:

$$w_{t+1} = w_t - \alpha_{\mathbf{t}} \nabla_w f^{Tr}(D_t; w_t), \quad \text{(Meta-SGD)}$$
$$w_{t+1} = w_t - \mathcal{A}(\mathbf{f_t}, \theta_{\mathbf{t}}; \phi) \nabla_w f^{Tr}(D_t; w_t), \text{(MLR-SNet)}$$

  where $f_t = f^{Tr}(D_t; w_t)$ and $\theta_t = (h_t, c_t)^T$. $\mathcal{A}(f_t, \theta_t; \phi)$ outputs the LR (α_t) at the t-th iteration, ϕ is the parameter of MLR-SNet, f_t is the loss of the batch samples D_t at the t-th 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 t-th iteration ($t = 0, \cdots, T-1$), d' represents the dimension of the state vectors (i.e., the size of hidden nodes). It can be evidently seen that MLR-SNet and Meta-SGD aim to optimize different learning variables, which are the weight parameters ϕ of meta-learner $\mathcal{A}$, and a sequence of LRs $\alpha_t, t = 1, 2, ..., T$, respectively.
- **Scalability.** Once the architecture of the MLR-SNet is determined, the size of weight parameters ϕ to learn is naturally fixed. Since ϕ does not depend on the the number of training steps T of the algorithm, MLR-SNet can flexibly generate any length of LRs by inputting training losses along iteration to the meta-learner. This means that MLR-SNet can easily scale to "long-horizons problems" [2], i.e., optimization with large steps of training iterations. Comparatively, Meta-SGD would learn a series of LRs $\alpha_t, t = 1, 2, ..., T$ with the iteration number of the algorithm. This means that it has a varying number of variables to learn when training steps increase. The computation cost is thus proportional to the number of training steps. This naturally conducts the issue that Meta-SGD tends to have relatively less scalability to iteration number of the algorithm as compared with MLR-SNet, especially when we need a large number of iterations but only with limited computation resource.
- **Task-transferable ability.** As aforementioned, the LR schedules learned by both Meta-SGD and MLR-SNet can be readily transferred for new query tasks. Meta-SGD aims to learn fixed learning rates shared across training tasks, and in the meta-test stage, Meta-SGD would use this fixed LR sequence for different query tasks. This tends to be with less adaptability to the variance of heterogeneous query tasks. Comparatively, LR schedules generated by the MLR-SNet depend on the input losses of training dynamics, which is allowed to be variant against different query tasks. This implies that it is possible for our MLR-SNet to generate different LR schedules, allowed with different LR values and entire length, according to the training dynamics of different query tasks, and thus the MLR-SNet is likely to better adapt to various query tasks. In a nutshell, MLR-SNet should have more adaptability to heterogeneous query tasks than Meta-SGD.

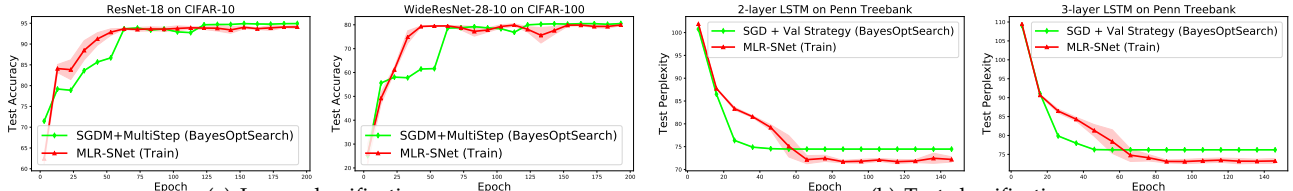
2 Hyper-parameter for experiments

The hyper-parameters of hand-designed LR schedules follows the recommend setting in the popular deep learning library (e.g., Pytorch) and the standard benckmarks [3], [4], [5], [6]. We just run their standard setting, and obtain the performance as reported in the original papers, e.g.,

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

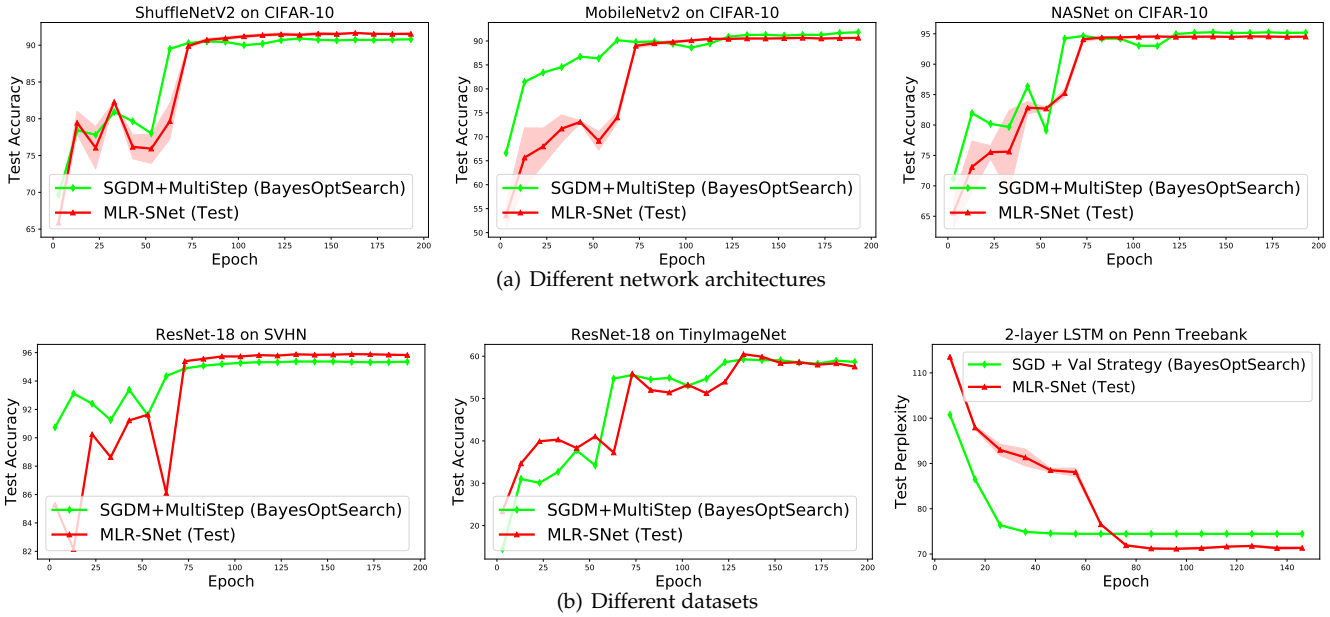
TABLE 1
The obtained hyperparamters for MultiStep LR schedule searched by Bayesian optimization methods.

Experimental setting	BayesOptSearch		TuneBOHB	
	α_0	γ	α_0	γ
ResNet-18 on CIFAR10	0.0864373149647392	0.0864491338167939	0.0394282991363671	0.111438341277555
WideResNet-28-10 on CIFAR-100	0.0868576230292215	0.0891149564802997	0.0117155877231516	0.0322469754925965
2-layer LSTM on Penn Treebank	20	0.5	8.51966678824753	0.0813423635903117
3-layer LSTM on Penn Treebank	19.0147789851342	0.193524658235207	12.0425319117129	0.137699823402115
ShuffleNetv2 on CIFAR-10	0.0752356594496997	0.0494370877605217	0.0394193161569907	0.0315894665307492
MobileNetv2 on CIFAR-10	0.0123472564508115	0.281309533029029	0.0356461197616519	0.282565199949938
NASNet on CIFAR-10	0.0864373149647392	0.0864491338167939	0.0419026854160617	0.111194065190743
ResNet-18 on SVHN	0.0503162127310114	0.128996623089215	0.06198005310463	0.321604407001577
ResNet-18 on TinyImageNet	0.0864373149647392	0.0864491338167939	0.0191591264630583	0.0257302389909082



(a) Image classification (b) Text classification

Fig. 1. Changing tendencies in terms of test accuracy (for image datasets) and perplexity (for text datasets) in iterations of MultiStep LR schedule with hyperparameters tuned by Bayesian optimization search methods and MLR-SNet in the meta-training stage on (a) image and (b) text classification datasets.



(a) Different network architectures

(b) Different datasets

Fig. 2. Changing tendencies in terms of test accuracy (for image datasets) and perplexity (for text datasets) in iterations of MultiStep LR schedule with hyperparameters tuned by Bayesian optimization search methods and directly transferred from MLR-SNet in the meta-test stage on (a) different network architectures and (b) different datasets.

CIFAR-10 and CIFAR-100. Actually, these hyperparameters configurations are usually searched by grid search method, as recommended by many current literatures (e.g., [7], https://github.com/hysts/pytorch_image_classification). To better compare the efficiency of conventional and proposed method, we also attempted the Bayesian optimization method to search the hyperparameters involved in hand-designed LR schedules, which is generally superior to random search and grid search [8].

Specifically, we have taken the MultiStep strategy, whose learning rate setting scheme is:

$$\alpha_t = \alpha_0 \times (\gamma_M)^i, l_{i-1} \leq E_{cur} \leq l_i, \quad (1)$$

as the testing example. For image classification tasks, we sample α_0 in $[10^{-2}, 5 \times 10^{-1}]$ log-uniformly, and γ_M in $[10^{-2}, 5 \times 10^{-1}]$ log-uniformly, and we fix the decay intervals as 60 epoches. For text classification tasks, we sample α_0 in $[10^{-2}, 20]$ log-uniformly, and γ_M in $[10^{-2}, 5 \times 10^{-1}]$ log-uniformly. The text classification tasks also use the Val strategy to determine when to decay the LR. We choose BayesOptSearch and TuneBOHB libraries in Ray (https://docs.ray.io/en/latest/tune/api_docs/suggestion.html#summary) to implement these experiments. The obtained hyperparameters for MultiStep LR schedule searched by Bayesian optimization methods are presented in Table 1. And then we use MultiStep LR schedule strategy equipped with these hyperparameters to train DNNs. The achieved performances are shown in Figs. 1 and 2 for the meta-training and meta-test experiments, respectively.

From Fig 1, it can be easily observed that our method

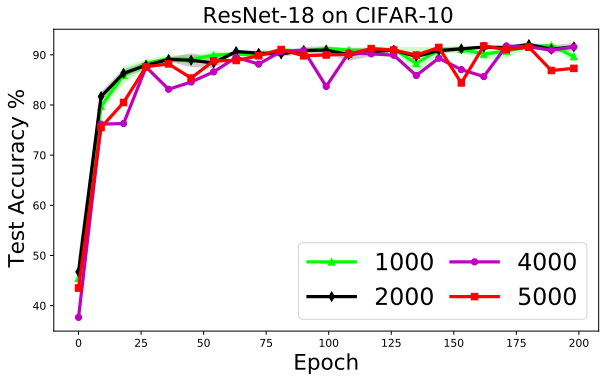


Fig. 3. Effect of the validation data size on the test performance of CIFAR-10 with ResNet-18 for RTHO.

obtains a comparable or even better performance for finding proper LR schedules in the meta-training stage compared with Bayesian optimization strategies. Especially, the MLR-SNet can achieve better performance on text classification compared with hand-designed LR schedules.

In the meta-test stage with task transfer setting, the hand-designed LR schedules need to search hyperparameters from scratch for achieving better performance. Otherwise, it might possibly cause performance degradation if we directly use hand-designed LR schedules assembling hyperparameters obtained from one task to train another new tasks (please see more analysis in Setion 5). Comparatively, our learned MLR-SNet can be directly used to set LR schedule to train new tasks without need of additional LR schedule learning from scratch. Specifically, we transfer the LR schedules learned on CIFAR-10 to train other network architectures, including ShuffleNetV2, MobileNetV2, NASNet, and other datasets, including SVHN, TinyImageNet, Penn Treebank, to implement comparison experiments in the meta-test process. The results are shown in Fig. 2. As comparison, we still use the Bayesian optimization method to search proper hyperparameters for MultiStep LR schedules for the training of all tasks. From the figure, we can see that the LR schedules, directly transferred from the meta-training stage by the proposed method, can achieve comparable or better performance among these meta-test tasks. Considering its largely saved computational cost for LR hyper-parameter tuning, it should be rational to say that the proposed method is efficient and useful in practice.

It should be emphasized that for the hyperparameters of competing LR schedule adaptation methods, inlcude L4, HD, RTHO, we have also employed the recommended settings in their original papers. We have just reproduced these methods based on their official implementation schemes released at:

- L4: https://github.com/martius-lab/l4-optimizer,
- HD: https://github.com/gbaydin/hypergradient- descent,
- RTHO: https://github.com/lucfra/RFHO.

3 Effect of Different Validation Set Size on RTHO

We attempt to increase the size of the validation set (for CIFAR-10) for the RTHO method to test its performance with respect to the validation scale. Specifically, we set the sizes of validation set as 1000, 2000, 4000, 5000, respectively, and show the corresponding test performance in Fig.3. It can be clearly seen that the performance of the RTHO method is not substantially improved with the increasing size of the validation set. Especially, its performance, even using larger-scale validation set, still do not surpass that of the MLR-SNet using less validate samples as reported in the manuscript.

This can be rationally explained by the fact that current LR adaptation methods, including RTHO, are more or less absent of the past training history information to guide the learning of LR, which makes them relatively more easily fall into bad local minima especially at the early training stage. For example, it is easily seen that the training loss of RTHO drops more quickly than our MLR-SNet at the early training stage (as seen in the 0-40 epoch shown in Fig.3, and 0-80 epoch depicted in Fig.4 in the main manuscript), but at the later training stage, RTHO fails to further decrease the training loss, reflecting that it might be possibly trapped into unexpected local minima. Comparatively, the utilization of the LSTM architecture makes our MLR-SNet capable of better adapting a sound learning rate schedule at a global scale, and thus naturally leads to its better generalization capability. This can be easily observed by the relatively more abundant variation details across its local areas along the meta-learned learning rate schedule, which can also be clearly observed in Fig. 3 and Fig. 4 of the main manuscript.

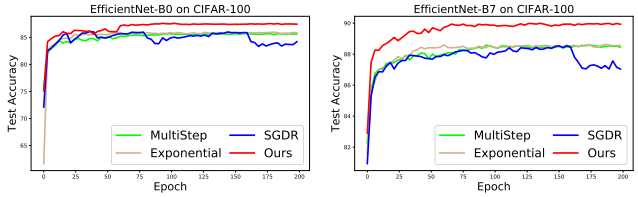
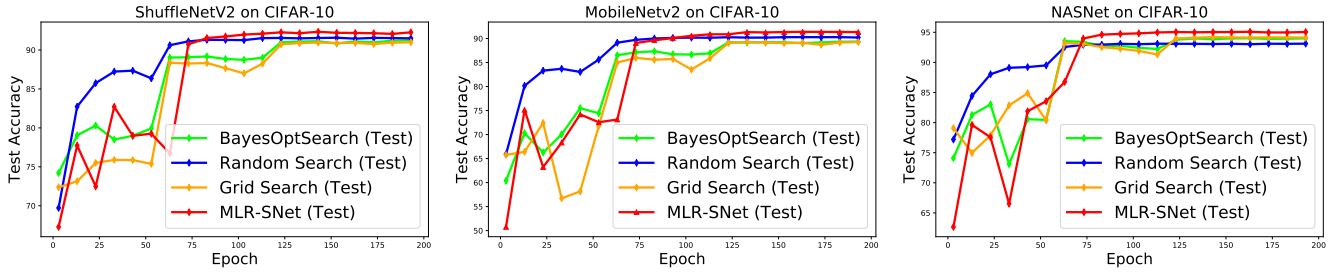


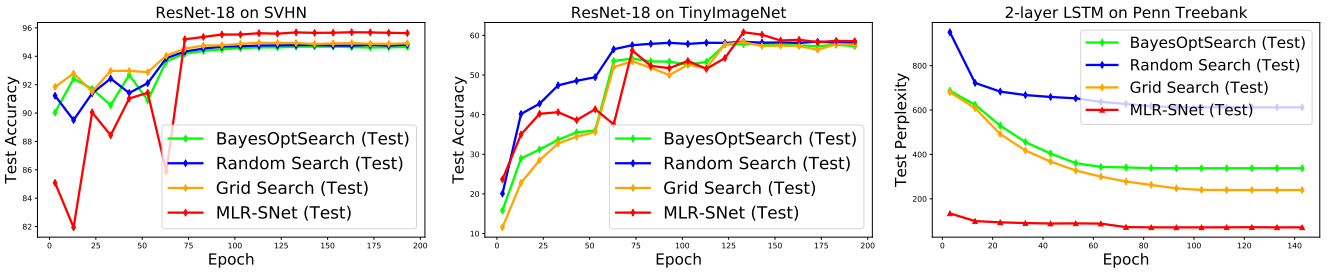
Fig. 4. The iterative performance of EfficientNet-B0 and EfficientNet-B7 on CIFAR-100 trained with the LR schedules produced by our MLR-SNet and hand-designed methods.

4 Generalization to SOTA EfficientNet

To further validate that our method can be applied to SOTA network architectures, we attempt to transfer the LR schedules meta-learned on ResNet-18 to EfficientNet [9]. Specifically, we train CIFAR-100 with EfficientNet-B0 and EfficientNet-B7 using Nesterov momentum with a momentum parameter of 0.9, weight decay 10^{-5} and a batch size of 256 as suggested in [9], [10]. Note that EfficientNet uses grid search method to set the LR schedules as it states [9], [10]. The hand-set grid consisted of 7 logarithmically spaced learning rates between 0.0001 and 0.1 and 7 logarithmically spaced weight decay to learning rate ratios between 10^{-6} and 10^{-3}. The computation cost of grid search method can be roughly estimated as training EfficientNet-B0 or EfficientNet-B7 with 200 epochs for 49 rounds (cost = 5.3M $\times$ 49 = 259.7M for EfficientNet-B0 and cost = 66M $\times$ 49 = 3234M for EfficientNet-B7), where the sizes of EfficientNet-B0 and EfficientNet-B7's weight parameters are 5.3M and 66M, respectively. Thus the computation cost of grid search method to set LR policy is about 259.7M for EfficientNet-B0 and 3234M for EfficientNet-B7, respectively. Recall that the



(a) Different network architectures



(b) Different datasets

Fig. 5. Changing tendencies in terms of test accuracy (for image datasets) and perplexity (for text datasets) in the meta-test tasks, with (a) different network architectures and (b) different datasets, calculated by our MLR-SNet and MultiStep LR schedules with hyperparameters obtained by Bayesian optimization, random search, grid search methods obtained from the meta-training stage on CIFAR-10.

computation cost of our MLR-SNet for setting LR policy is about the cost of meta-training MLR-SNet on CIFAR-10 with ResNet-18, which is about training ResNet-18 with 200 epochs for one round according to Fig.12 (b) in the main paper. Thus the cost is around 11.2 M $\times$ 1, where the size of ResNet-18's weight parameters is 11.2 M. Therefore, the computation cost of setting LR policy for grid search method is about 49 and 289 times more than MLR-SNet on training EfficientNet-B0 and EfficientNet-B7, respectively. We can thus see that MLR-SNet is relatively more computationally efficient than the grid search method.

To possibly reduce the computational resources of the grid search to be the same as the proposed MLR-SNet and make the comparison fair purely in sense of their costed computation resource, just similar to the task-transferring manner of MLR-SNet, we attempt to directly transfer the hyperparameters of hand-designed LR schedules tuned on CIFAR-10 by grid search method to both meta-test cases. It should be indicated that in such meta-training stage, the computational resource required by the grid search is still evidently larger than MLR-SNet, but considering the meta-test process, both need not to tune hyper-parameters involved in the LR schedule, and thus should be fair in computation cost in this stage.

Fig. 4 presents the test performance of two models trained with LR schedules produced by our MLR-SNet meta-learned on CIFAR-10 and hand-designd LR schedules with hyperparameters tuned by grid search on CIFAR-10 (see more details in Section 4.1.1 of the main paper for the hyperparameter setting). The advantage of the meta-trained LR schedule by MLR-SNet is evident, showing the method does be helpful to guide a sound learning tendency for SGD training compared with the hand-designed LR schedule directly tuned on CIFAR-10 by grid search. For the latter, the performance degradation can be rationally explained by its less adaptability to the new query tasks compared with MLR-SNet.

Note that the final test accuracies obtained by our transferred MLR-SNet on the dataset are 87.7% and 90.6% for EfficientNet-B0 and EfficientNet-B7, respectively, as compared with 88.1% and 91.7% as reported in the original paper. Though we do not perform the results reported in original paper, we use lower computation resources than searching proper hyperparameters for hand-designed LR schedules using grid search method. This makes the proposed method more efficient and friendly for general users, who might possibly lack sufficient computer resources and understandings to the network architectures and datasets.

5 Task-transferable Ability Comparison with Hand-designed LR Schedules

In Section 4.2 of the main paper, the hyperparameters of the compared hand-designed LR schedules are tuned from scratch as strong baselines to show the task-transferable potential of our meta-learned MLR-SNet. To make a fair comparison of the task-transferable LR setting policy, we further evaluate the hand-designed LR schedules on new query tasks equipped with hyperparameters tuned on CIFAR-10 with ResNet-18 in the meta-training stage.

Specifically, we have firstly adopted the Bayesian optimization, grid search and random search methods to search the optimal hyperparameters for MultiStep LR schedule on CIFAR-10 with ResNet-18. For Bayesian optimization and random search, we sample α_0 in $[10^{-2}, 5 \times 10^{-1}]$ log-uniformly, and γ_M in $[10^{-2}, 5 \times 10^{-1}]$ log-uniformly. For grid search, we search the α_0 and γ_M from the candidate set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09\}$. The Tune library in the Ray libraries (https://github.com/ray-project/ray) has been employed to implement these experiments. The best search results of BayesOptSearch, TuneBOHB, random

search and grid search are $\{\alpha_0 = 0.0864, \gamma_M = 0.0864\}$, $\{\alpha_0 = 0.0394, \gamma_M = 0.1114\}$, $\{\alpha_0 = 0.0154, \gamma_M = 0.0152\}$ and $\{\alpha_0 = 0.1, \gamma_M = 0.1\}$, respectively.

We then use the MultiStep LR schedule equipped with the above obtained hyperparameters to train on other network architectures, including ShuffleNetV2, MobileNetV2, NASNet, and other datasets, including SVHN, TinyImageNet, Penn Treebank, in such meta-test process. The achieved results, as compared with those obtained under our meta-trained MLR-SNet, are presented in Fig. 5 for easy observation. As can be easily seen, when the query tasks are certainly similar to CIFAR-10, all methods obtain approximately comparable fine performance under the transferred MultiStep LR schedule. However, when being applied to the text classification task evidently dissimilar to the meta-trained CIPAR-10, the performances tend to be evidently degraded for the transferred MultiStep LR schedule compared with MultiStep LR schedules with hyperparameters searched from scratch. While in such case, our method still obtains good performance, as clearly shown in Fig. 5 . This demonstrates that our method has a relatively stronger task-transferrable ability beyond hand-designed LR schedules, with a wider range of transferable testing tasks. Such superiority can be rationally explained by the fact that the proposed MLR-SNet is capable of adaptively adjusting its learned LR schedule forms for DNN training against specific characteristics of different meta-test tasks. Comparatively, the hand-designed LR schedules have to use fixed LRs, albeit having been optimized from the meta-training stage, in such task-transferable process, making them less adaptable to the variations of new query tasks.

In Section 4.2 of the main paper, we have directly compared our transferable MLR-SNet with hand-designed LR schedules with hyperparameters tuned from scratch, aiming to search possibly optimal LR schedules for the training task at hand. Even without specifically tuning its LR schedule hyperparameters on the task, our directly transferred MLR-SNet can achieve comparable or even better performance, as shown in Figs. 9 and 10 of the main paper. This shows the task-transferable potential of our MLR-SNet, i.e., it can approximate the performance of the hand-designed LR schedules with hyperparameters well tuned by hyperparameter optimization techniques. Attributed to its large saving of hyperparameter tuning cost, it should be rational to say that the proposed method is efficient and useful in practice.

We further implemented experiments on another data to verify the advantage of our method through task-transferable manner than directly using an LR schedule searched from it. Specifically, we consider to transfer learned MLR-SNet on a language modeling Wikitext-2 dataset [11], which is sourced from curated Wikipedia articles and is approximately twice the size of the PTB data set. The text is tokenized and processed using the Moses tokenizer, frequently used for machine translation, and features a vocabulary of over 33,000 words. The experimental setting is similar to the Penn Treebank dataset, i.e., the learning schedule of the proposed MLR-SNet is transferred from the model meta-trained on the CIFAR10 dataset, and the comparison method uses SGD with LR tuned using a additional validation set. In particular, we train the PyTorch word-level language model example (https://github.com/pytorch/examples/blob/main/word_language_model) on the Wikitext-2 dataset, and used 650-dimensional embeddings, 650 hidden units, tied weights, and dropout 0.5, and gradient clipping with threshold 0.25. The comparison results are shown in Fig.6. From the figure, it can be seen that our learned MLR-SNet achieves better performance than using the LR schedule directly searched from current dataset. This further verifies the superiority of the proposed method.

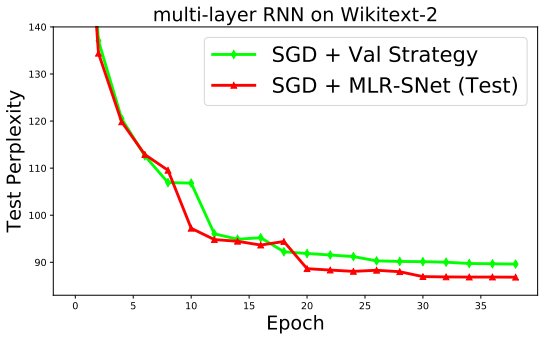


Fig. 6. Changing tendencies in terms of test perplexity for Wikitext-2 datasets trained with SGD using MultiStep LR schedule with hyperparameters tuned using a additional validation set of the current datasets and the LR schedule directly transferred from MLR-SNet obtained in the meta-training stage on CIFAR10.

6 Convergence Analysis of the MLR-SNet

6.1 Proof of Theorem 1

In the following we provide the proof details for the result of Theorem 1 in the maintext.

Proof. Let f^* be the infimum of $f(w)$, and then under the assumption A1, we have

$$f(w_{t+1}) \leq f(w_t) - \langle \nabla f(w_t), \alpha_t v_t \rangle + \frac{L}{2}\alpha_t^2 \|v_t\|^2. \quad (2)$$

Taking expectation on both sides, we have

$$\begin{aligned} \mathbb{E}f(w_{t+1}) - \mathbb{E}f(w_t) \leq & - \langle \mathbb{E}\nabla f(w_t), \alpha_t \mathbb{E}v_t \rangle + \frac{L}{2}\alpha_t^2 \mathbb{E}\|v_t\|^2 \\ = & - \alpha_t \mathbb{E}\|\nabla f(w_t)\| + \frac{L}{2}\alpha_t^2 \mathbb{E}\|v_t\|^2. \end{aligned}$$

According to the assumption A3, it produces that

$$\mathbb{E}\|v_t\|^2 \leq (\kappa+1)\mathbb{E}\|\nabla f(w_t)\| + \sigma.$$

Therefore, we have

$$\begin{aligned} & \mathbb{E}f(w_{t+1}) - \mathbb{E}f(w_t) \\ & \leq - \alpha_t \mathbb{E}\|\nabla f(w_t)\| + \frac{L}{2}\alpha_t^2 [(\kappa+1)\mathbb{E}\|\nabla f(w_t)\|^2 + \sigma] \\ & = - \left(\alpha_t - \frac{L(\kappa+1)}{2}\alpha_t^2\right) \mathbb{E}\|\nabla f(w_t)\|^2 + \frac{L}{2}\alpha_t^2 \sigma \\ & \leq - \frac{1}{2}\alpha_t \mathbb{E}\|\nabla f(w_t)\|^2 + \frac{L}{2}\alpha_t^2 \sigma, \end{aligned} \quad (3)$$

where the last inequality holds since $\alpha_t \leq \frac{1}{L(\kappa+1)}$. Let $\delta_t = \mathbb{E}f(w_t) - f^*$, and then we get

$$\delta_{t+1} \leq \delta_t - \frac{1}{2}\alpha_t \mathbb{E}\|\nabla f(w_t)\|^2 + \frac{L}{2}\alpha_t^2 \sigma. \quad (4)$$

Based on the assumption A2, we can get $-\frac{1}{2}\|\nabla f(w_t)\|^2 \leq \mu\delta_t$. Now, Eq(4) can be written as

$$\begin{aligned}
&\delta_{T+1} \leq (1-\mu\alpha_T)\delta_T + \frac{L}{2}\alpha_T^2\sigma \\
&\leq (1-\mu\alpha_T)\left[(1-\mu\alpha_{T-1})\delta_{T-1} + \frac{L}{2}\alpha_{T-1}^2 b\right] + \frac{L}{2}\alpha_T^2\sigma \\
&= (1-\mu\alpha_T)(1-\mu\alpha_{T-1})\delta_{T-1} + \frac{L\sigma}{2}\left[(1-\mu\alpha_T)\alpha_{T-1}^2 + \alpha_T^2\right] \\
&= \cdots \\
&= \prod_{t=1}^{T}(1-\mu\alpha_t)\delta_1 + \frac{L\sigma}{2}\sum_{t=1}^{T}\prod_{i=t+1}^{T}(1-\mu\alpha_i)\alpha_t^2.
\end{aligned}$$

Since $1-\mu\alpha_t \leq \exp(-\mu\alpha_t), t = 1, \cdots, T$, we have

$$\begin{aligned}
\delta_{T+1} \leq& \prod_{t=1}^{T}\exp(-\mu\alpha_t)\delta_1 + \frac{L\sigma}{2}\sum_{t=1}^{T}\prod_{i=t+1}^{T}\exp(-\mu\alpha_i)\alpha_t^2 \\
=& \exp(-\mu\sum_{t=1}^{T}\alpha_t)\delta_1 + \frac{L\sigma}{2}\sum_{t=1}^{T}\exp(-\mu\sum_{i=t+1}^{T}\alpha_i)\alpha_t^2.
\end{aligned} \tag{5}$$

Since $\alpha_t = \alpha_{t-1}\beta_t, 1/K \leq a \leq \beta_t$, then $\alpha_t \geq \alpha_0 a^t$,

$$\begin{aligned}
\sum_{t=1}^{T}\alpha_t &\geq \alpha_0\frac{a-a^{T+1}}{1-a} = \alpha_0\frac{a(1-a^T)}{1-a} \\
&\geq \frac{\alpha_0}{K}\frac{1-a^T}{1-a} = \frac{\alpha_0}{K}\frac{1-M/T}{1-a} \\
&\geq \frac{\alpha_0}{K}\frac{1-M/T}{1/T\ln(T/M)} = \frac{\alpha_0(T-M)}{K\ln(T/M)},
\end{aligned}$$

where we use the result that

$$1-x \leq \ln(1/x), \forall x$$

in the last inequallity. Thus we have

$$\begin{aligned}
\exp(-\mu\sum_{t=1}^{T}\alpha_t) &\leq \exp\left(-\mu\alpha_0\frac{T-M}{K\ln(T/M)}\right) \\
&= C(M)\exp\left(-\frac{\mu T}{KL(1+\kappa)\ln(T/M)}\right),
\end{aligned}$$

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

$$\sum_{i=t+1}^{T}\alpha_i = \alpha_0\frac{a^{t+1}-a^{T+1}}{1-a} \geq \frac{\alpha_0 T\left(a^t-a^T\right)}{K\ln(T/M)},$$

we can deduce that

$$\begin{aligned}
&\sum_{t=1}^{T}\exp(-\mu\sum_{i=t+1}^{T}\alpha_i)\alpha_t^2 \leq \sum_{t=1}^{T}\exp\left(-\mu\alpha_0 T\frac{a^t-a^T}{K\ln(T/M)}\right)\alpha_t^2 \\
=& C(M)\sum_{t=1}^{T}\exp\left(\frac{-\mu\alpha_0 T a^t}{K\ln(T/M)}\right)\alpha_t^2 \\
\leq& C(M)\sum_{t=1}^{T}\left(\frac{2K\ln(T/M)}{e\mu\alpha_0 a^t T}\right)^2\alpha_t^2 \\
\leq& C(M)\sum_{t=1}^{T}\left(\frac{2K\ln(T/M)}{e\mu\alpha_0 a^t T}\right)^2\alpha_0^2 b^{2t} \\
=& 4K^2C(M)\sum_{t=1}^{T}\frac{\ln^2(T/M)}{e^2\mu^2T^2}(N/M)^{2t/T} \\
=& \frac{4K^2C(M)\ln^2(T/M)}{e^2\mu^2T^2}\frac{(N/M)^{2/T}-(N/M)^{2+2/T}}{1-(N/M)^{2/T}}. \\
\leq& \frac{4K^2C(M)\ln^2(T/M)}{e^2\mu^2T^2}\frac{(N/M)^{2+2/T}}{(N/M)^{2/T}-1} \\
=& \frac{4K^2C(M)\ln^2(T/M)}{e^2\mu^2T^2}\frac{(N/M)^2}{1-(M/N)^{2/T}} \\
\leq& \frac{4K^2C(M)\ln^2(T/M)}{e^2\mu^2T^2}\frac{T(N/M)^2}{2-2M/N} \\
=& \frac{2K^2C(M)\ln^2(T/M)(N/M)^2}{e^2\mu^2(1-M/N)T},
\end{aligned}$$

where the second inequality holds since $\exp(-x) \leq (s/ex)^s$, $\forall x > 0, \forall s > 0$, and the last inequality is based on the Bernoulli inequality $(M/N)^{2/T} = (1+M/N-1)^{2/T} \leq 1+\frac{2M/N-2}{T}$. Putting all above results together, Eq.(5) can be bounded by

$$\begin{aligned}
\delta_{T+1} \leq& C(M)\exp\left(-\frac{\mu T}{KL(1+\kappa)\ln(T/M)}\right)\delta_1 \\
&+\frac{2K^2C(M)\ln^2(T/M)(N/M)^2}{e^2\mu^2(1-M/N)T}.
\end{aligned}$$

Thus the conclusion holds. □

6.2 Proof of Theorem 2

In the following we provide the proof details for the result of Theorem 2 in the maintext.

Proof. According to the proof process of Theorem 1, under the assumption **A1**,**A2** and the setting that $\alpha_0 = \frac{1}{L(1+\kappa)}$, it can be deduced that Eq.(3) holds, i.e.,

$$\mathbb{E}f(w_{t+1}) - \mathbb{E}f(w_t) \leq -\frac{1}{2}\alpha_t\mathbb{E}\|\nabla f(w_t)\|^2 + \frac{L}{2}\alpha_t^2\sigma. \tag{6}$$

Summing up above inequalities over $t = 1, 2, \cdots, T$, and rearranging the terms, we can obtain

$$\frac{1}{2}\sum_{t=1}^{T}\alpha_t\mathbb{E}\|\nabla f(w_t)\|^2 \leq \mathbb{E}f(w_1) - \mathbb{E}f(w_T) + \frac{L\sigma}{2}\sum_{t=1}^{T}\alpha_t^2.$$

Thus, we can deduce that

$$\min_{0\le t\le T}\mathbb{E}\|\nabla f(w_t)\|^2 \le \frac{\sum_{t=1}^T \alpha_t \mathbb{E}\|\nabla f(w_t)\|^2}{\sum_{t=1}^T \alpha_t} \le \frac{2\mathbb{E}f(w_1) - 2\mathbb{E}f(w_T) + L\sigma\sum_{t=1}^T \alpha_t^2}{\sum_{t=1}^T \alpha_t}.$$

Observing that

$$\begin{aligned}\sum_{t=1}^T \alpha_t^2 \le& \sum_{t=1}^T \alpha_0^2 b^{2t} = \alpha_0^2 \frac{b^2 - b^{2T+2}}{1-b^2}\\ \le& \alpha_0^2 \frac{1-b^{2T}}{1-b^2} = \alpha_0^2 \frac{1-(N/T)^2}{1-(N/T)^{2/T}}\\ =& \alpha_0^2 \frac{1-(N/T)^2}{1-\exp(2/T\ln(N/T))}\\ \le& \alpha_0^2 \frac{2\ln(T/N)}{1-1/(1-2/T\ln(N/T))}\\ =& \alpha_0^2\left(T + 2\ln(T/N)\right) = \frac{T+2\ln(T/N)}{c^2L^2(1+\kappa)^2},\end{aligned} \tag{7}$$

where the last inequality holds since $\exp(x) \le 1/(1-x), \forall x < 1$. Recall the following intermediate result of the proof in Theorem 1,

$$\sum_{t=1}^T \alpha_t \ge \frac{\alpha_0(T-M)}{K\ln(T/M)} = \frac{T-M}{cKL(1+\kappa)\ln(T/M)},$$

we can then obtain

$$\begin{aligned}\min_{0\le t\le T}\mathbb{E}\|\nabla f(w_t)\|^2 \le& \frac{2cKL(1+\kappa)\ln(T/M))}{T-M}\\ [\mathbb{E}f(w_1) - \mathbb{E}f(w_T)] + \mathcal{O}&\left(\frac{\sigma KT}{c(1+\kappa)(T-M)}\right).\end{aligned}$$

Thus the conclusion holds. □

6.3 Proof of Theorem 3

In the following we provide the proof details for the result of Theorem 3 in the maintext. First we need prove a necessary lemma as follows:

Lemma 1. *Suppose that the loss function f is Lipschitz smooth with respect to the model parameter w with constant L, and has ρ-bounded gradients with respect to the training/validation data. And the $\mathcal{A}(\theta)$ is differential with a δ-bounded gradient and twice differential with its Hessian bounded by $\mathcal{B}$. Then it holds that the gradient of MLR-SNet parameter θ with respect to the loss is also Lipschitz smooth.*

Proof. The gradient of MLR-SNet parameter θ with respect to the loss at data point j can be written as

$$\begin{aligned}\nabla_\theta f_j(\hat{w}_t(\theta))|_{\theta_t} =& \frac{\partial f_j(\hat{w}_t(\theta))}{\partial \hat{w}_t(\theta)}\frac{\partial \hat{w}_t(\theta)}{\partial \mathcal{A}(\theta)}\frac{\partial \mathcal{A}(\theta)}{\partial \theta}\\ =& \frac{-\alpha_t}{n}\sum_{i=1}^n\left(\frac{\partial f_j(\hat{w}_t(\theta))}{\partial \hat{w}_t(\theta)}\frac{\partial \ell_i(w_t)}{\partial w_t}\right)\frac{\partial \mathcal{A}(\theta)}{\partial \theta}\Big|_{\theta_t},\end{aligned}$$

Let $G_{ij} = \frac{\partial \ell_j(\hat{w}_t(\theta))}{\partial \hat{w}_t(\theta)}\frac{\partial \ell_i(w_t)}{\partial w_t}$, and then take gradient of θ in both sides of the above equality. We then have

$$\nabla^2_{\theta^2} f_j(\hat{w}_t(\theta))|_{\theta_t} = \frac{-\alpha_t}{n}\sum_{i=1}^n\left[\frac{\partial G_{ij}}{\partial \theta}\frac{\partial \mathcal{A}(\theta)}{\partial \theta} + G_{ij}\frac{\partial \mathcal{A}^2(\theta)}{\partial \theta^2}\right]. \tag{8}$$

Algorithm 1 Adam Algorithm

Input: $\theta_1 \in \mathbb{R}^{d'}$, learning rate $\{\eta_t\}_{t=1}^T$, decay parameters $0 \le \beta_1, \beta_2 \le 1, \epsilon > 0$.
Output: MLR-SNet parameter θ_T

```
Set m_0 = 0, v_0 = 0.
for t = 0 to T − 1 do
    D_n ← SampleMiniBatch(D_Val, n).
    Compute g_t = ∇_θ f^Val(D_n, θ_t).
    m_t = β_1 m_{t−1} + (1 − β_1) g_t
    v_t = v_{t−1} − (1 − β_2)(v_{t−1} − g_t^2)
    θ_{t+1} = θ_t − η_t m_t / (√v_t + ϵ)
end for
```

For the first term in the right hand side, we have that

$$\begin{aligned}&\left\|\frac{\partial G_{ij}}{\partial \theta}\frac{\partial \mathcal{A}(\theta)}{\partial \theta}\right\| \le \delta\left\|\frac{\partial f_j(\hat{w}_t(\theta))}{\partial \hat{w}_t(\theta)\partial\theta}\frac{\partial f_i(w_t)}{\partial w_t}\right\|\\ =&\delta\left\|\frac{\partial}{\partial \hat{w}_t(\theta)}\left(\frac{-\alpha_t}{n}\sum_{i=1}^n\left(\frac{\partial f_j(\hat{w}_t(\theta))}{\partial \hat{w}_t(\theta)}\frac{\partial f_i(w_t)}{\partial w_t}\right)\frac{\partial \mathcal{A}(\theta)}{\partial \theta}|_{\theta_t}\right)\frac{\partial f_i(w_t)}{\partial w_t}\right\|\\ =&\delta\left\|\left(\frac{-\alpha_t}{n}\sum_{i=1}^n\left(\frac{\partial^2 f_j(\hat{w}_t(\theta))}{\partial \hat{w}_t^2(\theta)}\frac{\partial f_i(w_t)}{\partial w_t}\right)\frac{\partial \mathcal{A}(\theta)}{\partial \theta}|_{\theta_t}\right)\frac{\partial f_i(w_t)}{\partial w_t}\right\| \le \alpha_t L\rho^2\delta^2.\end{aligned} \tag{9}$$

For the second term in the right hand side, we have that

$$\left\|G_{ij}\frac{\partial \mathcal{A}^2(\theta)}{\partial \theta^2}\right\| \le \mathcal{B}\rho^2. \tag{10}$$

Combining the above two inequalities Eq.(9) and (10), we have

$$\|\nabla_\theta f_j(\hat{w}_t(\theta))|_{\theta_t}\| \le \alpha\rho^2(\alpha_t L\delta^2 + \mathcal{B}). \tag{11}$$

Define $L_A = \alpha\rho^2(\alpha_t L\delta^2 + \mathcal{B})$, and based on the Lagrange mean value theorem, we have:

$$\left\|\nabla f^{Val}(\hat{\mathbf{w}}_t(\theta_1)) - f^{Val}(\hat{\mathbf{w}}_t(\theta_2))\right\| \le L_A\left\|\theta_1 - \theta_2\right\|. \tag{12}$$

Thus the conclusion holds. □

Now we present the proof of Theorem 3.

Proof. Suppose that we have a small validation set with B samples $\{x_1, x_2, \cdots, x_M\}$, each associating with a validation loss function $\ell_i(w(\theta))$, where w is the parameter of the model, and θ is the parameter of the MLR-SNet. The overall validation loss is then:

$$f^{Val}(w) = \frac{1}{B}\sum_{i=1}^B f_i^{Val}(w(\theta)), \tag{13}$$

where B is the minibatch size. According to the updating Algorithm 1, we have:

$$\begin{aligned}&\mathbb{E}f^{Val}(\hat{w}_{t+1}(\theta_{t+1})) - \mathbb{E}f^{Val}(\hat{w}_t(\theta_t))\\ =&\underbrace{\left\{\mathbb{E}f^{Val}(\hat{w}_{t+1}(\theta_{t+1})) - \mathbb{E}f^{Val}(\hat{w}_t(\theta_{t+1}))\right\}}_{(a)}\\ &+\underbrace{\left\{\mathbb{E}f^{Val}(\hat{w}_t(\theta_{t+1})) - \mathbb{E}f^{Val}(\hat{w}_t(\theta_t))\right\}}_{(b)}.\end{aligned} \tag{14}$$

For the above term (a), it holds that

$$\begin{aligned}&\mathbb{E}f^{Val}(\hat{w}_{t+1}(\theta_{t+1})) - \mathbb{E}f^{Val}(\hat{w}_t(\theta_{t+1}))\\ \le&\left\langle\mathbb{E}\nabla_w f^{Val}(\hat{w}_{t+1}(\theta_{t+1})), \mathbb{E}\hat{w}_{t+1}(\theta_{t+1}) - \mathbb{E}\hat{w}_t(\theta_{t+1})\right\rangle\\ &+\frac{L}{2}\mathbb{E}\left\|\hat{w}_{t+1}(\theta_{t+1}) - \hat{w}_t(\theta_{t+1})\right\|_2^2.\end{aligned} \tag{15}$$

According to Eq (7) in the maintext, we have

$$\hat{w}_{t+1}(\theta_{t+1}) - \hat{w}_t(\theta_{t+1}) = -\alpha_t \nabla_w f^{Tr}(\hat{w}_t(\theta_{t+1})).$$

Then Eq (15) can be written as

$$\begin{aligned} a \leq & - \langle \mathbb{E}\nabla_w f^{Val}(\hat{w}_{t+1}(\theta_{t+1})), \alpha_t \mathbb{E} v_t \rangle + \frac{L}{2}\alpha_t^2 \mathbb{E}\|v_t\|^2 \\ \leq & - \langle \mathbb{E}\nabla_w f^{Val}(\hat{w}_{t+1}), \alpha_t \mathbb{E} v_t \rangle + \frac{L}{2}\alpha_t^2[(\kappa+1)\mathbb{E}\|\nabla f(w_t)\|^2 + \sigma] \\ \leq & \alpha_t \rho^2 + \frac{L}{2}\alpha_t^2[(1+\kappa)\rho^2 + \sigma]. \end{aligned}$$

For the term (b) in Eq. (14), according to Lemma 1, i.e., the validation loss is Lipschitz smooth with respect to the MLR-SNet parameter θ with L, we have

$$\begin{aligned} & \mathbb{E} f^{Val}(\hat{w}_t(\theta_{t+1})) - \mathbb{E} f^{Val}(\hat{w}_t(\theta_t)) \\ \leq & \left\langle \mathbb{E}\nabla_\theta f^{Val}(\hat{w}_t(\theta_t)), \mathbb{E}\theta_{t+1} - \mathbb{E}\theta_t \right\rangle + \frac{L}{2}\mathbb{E}\left\|\theta_{t+1} - \theta_t\right\|_2^2. \end{aligned} \tag{16}$$

Here we adopt Adam algorithm [12] (Algorithm 1) to update the parameter of MLR-SNet, $\theta_{t+1} - \theta_t$ in Eq.(16) is updated by

$$\theta_{t+1,i} = \theta_{t,i} - \eta_t \frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon}, i = 1, 2, \cdots, d. \tag{17}$$

Now, we have

$$\begin{aligned} b \leq & - \eta_t \sum_{i=1}^{d} \left\langle \mathbb{E}\nabla_\theta \mathcal{L}_{Val}^i(\hat{w}_t(\theta_t)), \mathbb{E}\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon} \right\rangle \\ & + \frac{L\eta_t^2}{2}\mathbb{E}\sum_{i=1}^{d}\frac{g_{t,i}^2}{(\sqrt{v_{t,i}} + \epsilon)^2}. \end{aligned} \tag{18}$$

Based on the proof process in [13] (Eq. (4) in pp. 13), we can deduce that

$$\begin{aligned} b \leq & - \frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)}\mathbb{E}\|\nabla_\theta f^{Val}(\hat{w}_t(\theta_t))\|_2^2 \\ & + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2}{B}. \end{aligned} \tag{19}$$

Now Eq.(14) can be reformulated as:

$$\begin{aligned} & \mathbb{E} f^{Val}(\hat{w}_{t+1}(\theta_{t+1})) - \mathbb{E} f^{Val}(\hat{w}_t(\theta_t)) \\ & \leq \alpha_t \rho^2 + \frac{L}{2}\alpha_t^2[(1+\kappa)\rho^2 + \sigma] - \frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)} \\ & \mathbb{E}\|\nabla_\theta f^{Val}(\hat{w}_t(\theta_t))\|_2^2 + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2}{B}, \end{aligned} \tag{20}$$

By rearranging the inequality (20), we can then obtain:

$$\begin{aligned} & \mathbb{E}\left[\frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)}\|\nabla_\theta \mathcal{L}_{Val}(\hat{w}_t(\theta_t))\|_2^2\right] \\ \leq & \alpha_t \rho^2 + \frac{L}{2}\alpha_t^2(\rho^2 + \sigma^2) - \mathbb{E} f^{Val}(\hat{w}_{t+1}(\theta_{t+1})) \\ & + \mathbb{E} f^{Val}(\hat{w}_t(\theta_t)) + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2}{B}. \end{aligned}$$

Using telscoping sum, we obtain

$$\begin{aligned} & \sum_{t=1}^{T}\frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)}\mathbb{E}\left\|\nabla_\theta f^{Val}(\hat{w}_t(\theta_t))\right\|_2^2 \\ \leq & \mathbb{E} f^{Val}(\hat{w}_1(\theta_1)) - \mathbb{E} f^{Val}(\hat{w}_{T+1}(\theta_{T+1})) + \rho^2\sum_{t=1}^{T}\alpha_t \\ & + \frac{L}{2}(\rho^2 + \sigma^2)\sum_{t=1}^{T}\alpha_t^2 + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2 T}{B} \\ \leq & f^{Val}(\hat{w}_1(\theta_1)) + \rho^2\sum_{t=1}^{T}\alpha_t + \frac{L}{2}(\rho^2 + \sigma^2)\sum_{t=1}^{T}\alpha_t^2 \\ & + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2 T}{B}. \end{aligned} \tag{21}$$

Therefore,

$$\begin{aligned} & \min_t \mathbb{E}\left[\left\|\nabla_\theta f^{Val}(\hat{w}_t(\theta_t))\right\|_2^2\right] \\ \leq & \frac{\sum_{t=1}^{T}\frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)}\mathbb{E}\left\|\nabla_\theta f^{Val}(\hat{w}_t(\theta^{(t)}))\right\|_2^2}{\sum_{t=1}^{T}\frac{\eta_t}{2(\sqrt{\beta_2}\rho + \epsilon)}} \\ \leq & \frac{f^{Val}(\hat{w}_1(\theta_1)) - f^{Val}(\hat{w}_{T+1}(\theta_{T+1})) + S}{1/2(\sqrt{\beta_2}\rho + \epsilon) \times \sum_{t=1}^{T}\eta_t} \\ \leq & \frac{2(\sqrt{\beta_2}\rho + \epsilon)}{T\eta} \times \left\{f^{Val}(\hat{w}_1(\theta_1)) + S\right\}, \end{aligned}$$

where $S = \frac{L}{2}(\rho^2 + \sigma^2)\sum_{t=1}^{T}\alpha_t^2 + + \left(\frac{\eta\rho\sqrt{1-\beta_2}}{\epsilon^2} + \frac{L\eta^2}{2\epsilon^2}\right)\frac{\sigma^2 T}{B} + \rho^2\sum_{t=1}^{T}\alpha_t$. Taking a similar process as in Eq.(7), we have that

$$\begin{aligned} \sum_{t=1}^{T}\alpha_t \leq & \frac{\ln(T/N) + T}{cL(1+\kappa)\ln(T/N)}, \\ \sum_{t=1}^{T}\alpha_t^2 \leq & \frac{2\ln(T/N) + T}{c^2 L^2(1+\kappa)^2\ln(T/N)}. \end{aligned}$$

Therefore, we can obtain

$$\min_t \mathbb{E}\left\|\nabla_\theta f^{Val}(\hat{w}_t(\theta_t))\right\|_2^2 \leq \mathcal{O}(\frac{1}{c^2\ln(T)} + \sigma^2)$$

Thus the conclusion holds. □

7 PYTORCH IMPLEMENTATION OF MLR-SNET

Here we also demonstrate the pseudo-code of the MLR-SNet for Pytorch implementation as follows, to make readers easily reproduce our algorithm. The full source code of our method is released at https://github.com/xjtushujun/MLR-SNet.

```
class LSTMCell(nn.Module):
  def __init__(self, num_inputs, hidden_size):
    super(LSTMCell, self).__init__()
    self.hidden_size = hidden_size
    self.fc_i2h = nn.Sequential(
      nn.Linear(num_inputs, hidden_size),
      nn.ReLU(),
      nn.Linear(hidden_size, 4 * hidden_size))
    self.fc_h2h = nn.Sequential(
      nn.Linear(hidden_size, hidden_size),
      nn.ReLU(),
      nn.Linear(hidden_size, 4 * hidden_size))
  def forward(self, inputs, state):
    hx, cx = state
    i2h = self.fc_i2h(inputs)
    h2h = self.fc_h2h(hx)
    x = i2h + h2h
    gates = x.split(self.hidden_size, 1)
    in_gate = torch.sigmoid(gates[0])
    forget_gate = torch.sigmoid(gates[1])
    out_gate = torch.sigmoid(gates[2])
    in_transform = torch.tanh(gates[3])
    cx = forget_gate * cx + in_gate * in_transform
    hx = out_gate * torch.tanh(cx)
    return hx, cx

class MLRNet(nn.Module):
  def __init__(self, num_layers, hidden_size):
    super(MLRNet, self).__init__()
    self.hidden_size = hidden_size
    self.layer1 = LSTMCell(1, hidden_size)
    self.layer2 = nn.Linear(hidden_size, 1)
  def forward(self, x, gamma):
    self.hx, self.cx =
      self.layer1(x, (self.hx, self.cx))
    x = self.hx
    x = self.layer2(x)
    out = torch.sigmoid(x)
    return gamma * out
```

REFERENCES

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

[2] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," in *ICML*, 2017.

[3] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.

[4] N. Loizou, S. Vaswani, I. Laradji, and S. Lacoste-Julien, "Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence," *arXiv:2002.10542*, 2020.

[5] https://github.com/hysts/pytorch_image_classification.

[6] https://github.com/pytorch/examples/tree/master/word_language_model.

[7] K. Tian, Y. Xu, J. Guan, and S. Zhou, "Network as regularization for training deep neural networks: Framework, model and performance," in *AAAI*, 2020.

[8] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *NeurIPS 2020 Competition and Demonstration Track*, 2021.

[9] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[10] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *CVPR*, 2019.

[11] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *ICLR*, 2017.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[13] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *NeurIPS*, 2018, pp. 9793–9803.