

SUPPLEMENTARY MATERIAL FOR “MLR-SNET: TRANSFERABLE LR SCHEDULES FOR HETEROGENEOUS TASKS”

Anonymous authors

Paper under double-blind review

ABSTRACT

In this supplementary material, we present more experimental details and illustrations of our algorithm in the main paper, as well as computational complexity analysis and additional comparison results with LR Controller method.

A EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS IN SECTION 4.1

In this section, we attempt to evaluate the capability of MLR-SNet to learn LR schedules compared with baseline methods. Here, we provide implementation details of all experiments.

Datasets. We choose two datasets in image classification (CIFAR-10 and CIFAR-100), and one dataset in text classification (Penn Treebank) to present the efficiency of our method. CIFAR-10 and CIFAR-100 Krizhevsky (2009), consisting of 32×32 color images arranged in 10 and 100 classes, respectively. Both datasets contain 50,000 training and 10,000 test images. Penn Treebank Marcus & Marcinkiewicz is composed of 929k training words, 73k validation words, and 82k test words, with a 10k vocabulary in total. Our algorithm and RTHO Franceschi et al. (2017) randomly select 1,000 clean images in the training set of CIFAR-10/100 as validation data, and directly use the validation set in Penn Treebank as validation data.

CIFAR-10 & CIFAR-100. We employ ResNet-18 on CIFAR-10 and WideResNet-28-10 Zagoruyko & Komodakis (2016) 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 to 0.1, weight decay parameter to $5e^{-4}$ and momentum to 0.9 if used. While for **Adam**, we just follow the default parameter setting. The hyper-parameters of hand-designed LR schedules are listed below: **Exponential** decay, multiplying LR with 0.95 every epoch; **MultiStep** decay, decaying LR by 10 every 60 epochs; **SGDR**, setting T_0 to 10, T_{Mult} to 2 and minimum LR to $1e^{-5}$. **L4**, **HD** and **RTHO** update LR every data batch, and we use the recommended setting in the original paper of **L4** ($\alpha = 0.15$) and search different hyper-lrs from $\{1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}, 1e^{-7}\}$ for **HD** and **RTHO**, reporting the best performing hyper-lr.

Penn Treebank. 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 without momentum, 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 reporting 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 and parameter setting. The architecture of MLR-SNet is illustrated in Section 3.2. In our experiment, the size of hidden nodes is set as 40. The initialization of MLR-SNet follows the default setting in Pytorch. The Pytorch implementation of MLR-SNet is listed below.

```

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

```

We employ Adam optimizer to train MLR-SNet, and just set the parameters as originally recommended with a LR of $1e^{-3}$, and a weight decay of $1e^{-4}$, which avoids extra hyper-parameter tuning. For image classification tasks, the input of MLR-SNet is the training loss of a mini batch samples. Every data batch's LR is predicted by MLR-SNet and we update it twice per epoch according to the loss of the validation data. While for text classification tasks, we take $\frac{\mathcal{L}_{Tr}}{\log(\text{vocabulary size})}$ as input of MLR-SNet to deal with the influence of large scale classes of text. MLR-SNet is updated every 100 batches due to the large number of batches per epoch compared to that in image datasets.

Results. Due to the space limitation, we only present the test accuracy in the main paper. Here, we present the training loss and test accuracy of our method and all compared methods on image and text tasks, as shown in Fig.1. For image tasks, except for Adam and SGD with fixed LR, other methods can decrease the loss to 0 almostly. Though local minima can be reached by these methods, the generalization ability of the these mimimas has a huge difference, which can be summarized from test accuracy curves. As shown in Fig. 1(a),1(b),1(g),1(h), when using SGD to train DNNs, the compared methods SGD with Exponential LR, L4, HD, RTHO fail to find such good solutions to generalize well. Especially, L4 greedily searches LR to decrease loss to 0, making it fairly hard to adapt the complex DNNs training dynamic and obtain a good mimima, while our method can adjust LR to comply with the significant variations of training dynamic, leading to a better generalization

solution. As shown in Fig. 1(d),1(e),1(j),1(k), when baseline methods are trained with SGDM, these methods make a great progress in escaping from the bad minimas. In spite of this, our method still shows superiority in finding a solution with better generalization compared with these competitive training strategies.

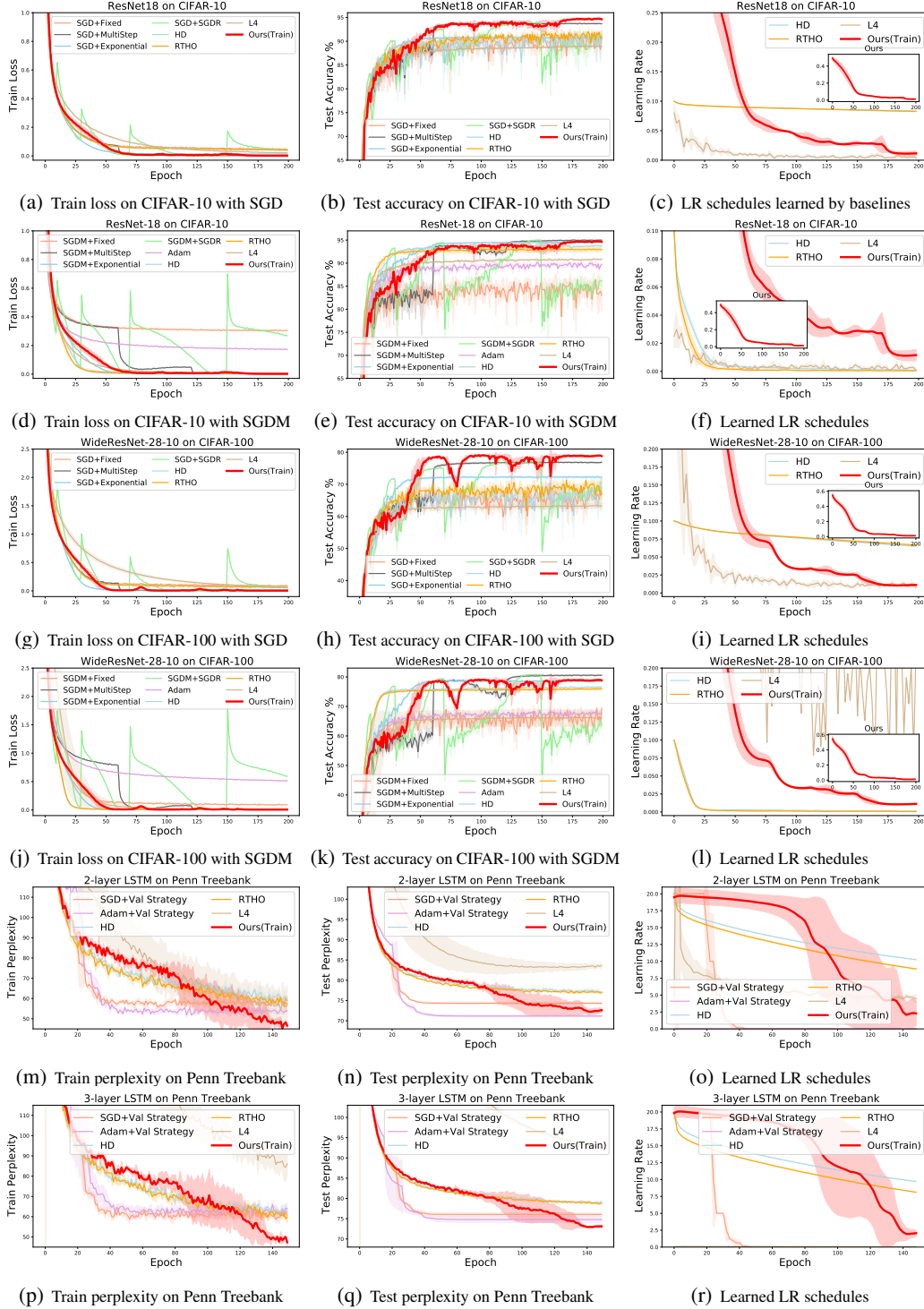
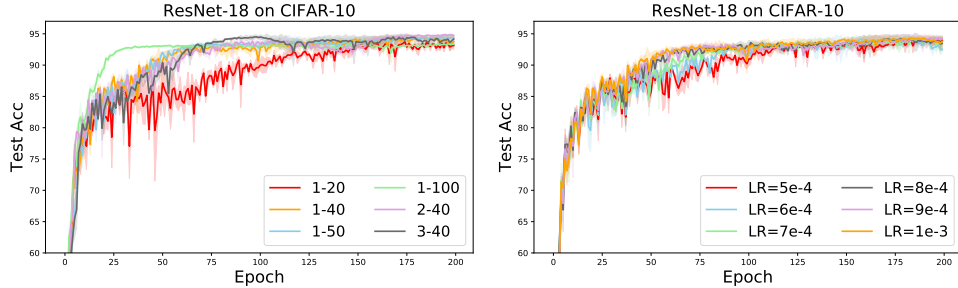


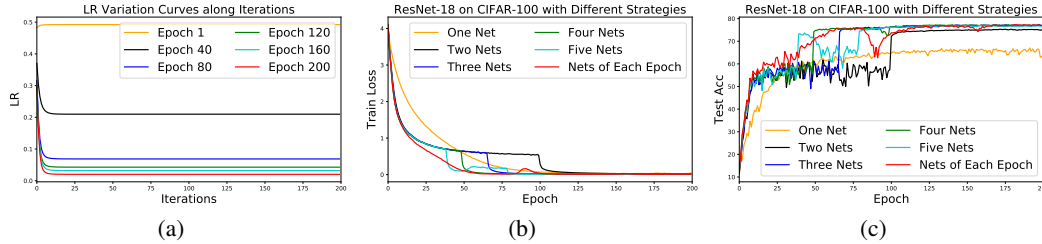
Figure 1: Train loss (perplexity), test accuracy (perplexity) and learned LR schedules of our methods (train) and compared baselines on different tasks.



(a) Different Architectures of MLR-SNet

(b) Different LR of Meta Optimizer

Figure 2: Ablation study. (a) Test accuracy on CIFAR-10 with ResNet-18 of different architectures of MLR-SNet. ‘a-b’ denotes the configurations of MLR-SNet, where ‘a’ represents the number of layers, and ‘b’ represents the number of hidden nodes. (b) Test accuracy on CIFAR-10 with ResNet-18 of different LR of meta optimizer ‘Adam’.



(a)

(b)

(c)

Figure 3: (a) We plot the LR variation curves along iterations with the same input for learned MLR-SNet at different epochs. As is shown, when iteration increases, the LR is almost constant. This means the learned MLR-SNet overfits the short trajectories, while fails for the long trajectories. (b),(c) show the recording train loss and test accuracy with ResNet-18 on CIFAR-100 of different test strategies.

In the third column in Fig. 1, we plot learned LR schedules of compared methods and our method. As can be seen, our method can learn LR schedules approximating the hand-designed LR schedules while with more locally varying. HD and RTHO often have the same trajectory while producing lower or faster downward trend than ours. This tends to explain our final performances on test set is better than HD and RTHO, since our method can adaptively adjust LR utilizing the past training histories explicitly. L4 greedily searches a LR to decrease the loss. This often leads to a large value causing fluctuations or even divergence (Fig. 1(l)), or a small value causing slow progress (Fig. 1(r)), or both of them (Fig. 1(f) 1(i) 1(o)). Such LR schedules often result in bad minimas. Moreover, all compared methods regard LR as hyper-parameter to learn without a transferable formulation, and the learned LR schedules can not generalize to other learning tasks directly. Generally, they just try to find a proper LR schedule from scratch for new tasks. However, our meta-learned MLR-SNet is plug-and-play and transferrable, which can directly transfer how to schedule LR for SGD to heterogeneous tasks without additional learning.

Ablation study. (1) **The architecture of MLR-SNet.** In Fig.2(a) shows the test accuracy on CIFAR-10 with ResNet-18 of different architectures of MLR-SNet. As can be seen, our algorithm is not sensitive to the choose of the MLR-SNet’s architectures. This implies that our algorithm is robust and stable for helping improve DNN training. (2) **The gobal LR of the meta optimizer.** To further validate that whether our MLR-SNet behaves robust to the meta optimizer. We adapt Adam optimizer to search the proper LR schedules. Fig. 2(b) shows that our MLR-SNet achieves the similar performance even for different global LR. This implies our MLR-SNet needs not carefully tune the LR of the meta optimizer, which makes it easy to reproduce and apply to various problems.

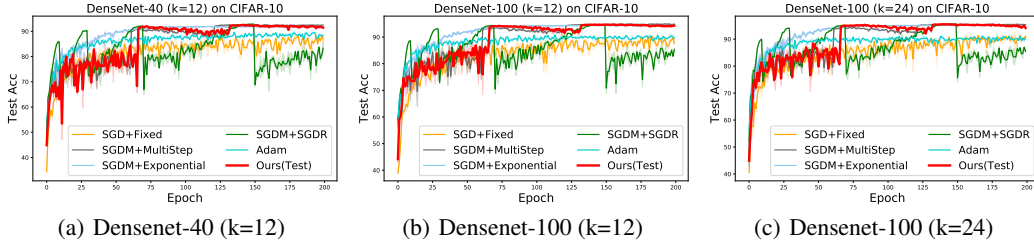


Figure 4: Test accuracy on CIFAR-100 of different DenseNet architectures.

B EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS IN SECTION 4.2

We investigate the transferability of the learned LR schedule when applied to various tasks in Section 4.2 of the main paper. We use the MLR-SNet meta-learned on CIFAR-10 with ResNet-18 in Section 4.1 to directly predict the LR for SGD algorithm to new heterogeneous tasks. Here, we provide implementation details of all experiments.

As is shown in Fig.3(a), it can be seen that the predicted LR by the learned LR schedules converges after several iterations. This is because that the training trajectories are long in our experiments, and the learned MLR-SNet can not memory all the information since we locally adjust our MLR-SNet according to the validation error. If we directly select one MLR-SNet learned at any epoch, that will raise overfitting issues as shown in Fig.3(a). Thus we should select more than two learned MLR-SNets for test. Here, we propose a heuristic strategy to select MLR-SNets for test. Generally, if we want to select k nets for test, the MLR-SNet learned at $\lceil \frac{200 \times l}{k-1} \rceil$ -th epoch ($l = 0, 1, 2, \dots, k-1$) should be chosen, where $\lceil \cdot \rceil$ denotes ceiling operator. Fig.3(b) and 3(c) show the train loss and test accuracy with ResNet-18 on CIFAR-100 of different test strategies, i.e., choosing different number of nets to transfer. It can be seen that almost choosing more than three nets have similar performance. Therefore, in the following experiments we choose three MLR-SNets to show the transferability.

Transfer to different epochs. We transfer the LR schedules meta-trained with epoch 200 to other different epochs, e.g., 100, 400, 1200. All the methods are trained with ResNet-18 on CIFAR-100 with batch size 128 for different epochs. The hyper-parameter setting for compared hand-designed LR schedules is the same with Section 4.1 in the main paper as illustrated above, except for MultiStep LR. For epoch 100, MultiStep LR decays LR by 10 every 30 epochs; For epoch 400, MultiStep LR decays LR by 10 every 120 epochs; For epoch 1200, MultiStep LR decays LR by 10 every 360 epochs. Other hyper-parameters of MultiStep LR keep unchanged. For our method, we use the transferred strategy 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.

Transfer to different datasets. We transfer the LR schedules meta-learned on CIFAR-10 to SVHN (Netzer et al., 2011), TinyImageNet¹, and Penn Treebank (Marcus & Marcinkiewicz). For image classification, we train a ResNet-18 on SVHN and TinyImageNet, respectively. The hyper-parameters of all compared methods are set the same as those of CIFAR-10. For text classification, we train a 3-layer LSTM on Penn Treebank. The hyper-parameters of all compared methods are with the same setting as introduced in Section 4.1.

Transfer to different net architectures. We transfer the learned LR schedules for different net architectures training. All the methods are trained on CIFAR-10 with different net architectures. The hyper-parameters of all methods are the same with the setting of CIFAR-10 with ResNet-18. We test the meta-learned LR schedule to different configurations of DenseNet Huang et al. (2017). As shown in Fig. 4, our method perform slightly stable than MultiStep strategy at about 75-125 epochs. This tends to show the superiority of adaptive LR to train the DenseNets. Also, we transfer the LR schedules to several novel networks, the results are presented in Fig.8 in the main paper.

Transfer to large scale optimization. We transfer the learned LR schedules for the training of the large scale optimization problems. The predicted LR by MLR-SNet will not substantially increase the complexity compared with hand-designed LR schedules for DNNs training. This makes it feasible and reliable to transfer our meta-learned LR schedules to such large scale optimization problems. We train a ResNet-50 on ImageNet with hand-designed LR schedules and our trans-

¹It can be downloaded at <https://tiny-imagenet.herokuapp.com>.

Table 1: Test accuracy (%) on CIFAR-10 and CIFAR-100 training set of different methods trained on CIFAR-10-C and CIFAR-100-C. Best and Last denote the results of the best and the last epoch. The **Bold** and **Underline Bold** denote the first and second best results, respectively.

Datasets/Methods		Fixed	MultiStep	Exponential	SGDR	Adam	Ours(Train)
CIFAR-10-C	Best	79.96 \pm 4.09	85.64 \pm 1.71	83.63 \pm 1.38	86.10\pm1.44	81.57 \pm 1.39	85.73\pm1.71
	Last	77.89 \pm 4.05	85.48\pm1.71	83.47 \pm 1.37	78.46 \pm 1.92	80.39 \pm 1.65	85.62\pm1.76
CIFAR-100-C	Best	46.91 \pm 3.08	52.38 \pm 2.43	49.90 \pm 1.93	52.80\pm2.39	45.58 \pm 1.95	52.51\pm2.38
	Last	44.81 \pm 5.98	52.28\pm2.44	49.75 \pm 1.94	41.68 \pm 3.33	43.94 \pm 2.18	52.35\pm2.46

ferred LR schedules. The training code can be found on <https://github.com/pytorch/examples/tree/master/imagenet>, and the parameter setting keeps unchanged except the LR. All compared hand-designed LR schedules 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. **Fixed** LR uses 0.1 LR during the whole training; **Exponential** LR multiplies LR with 0.95 every epoch; **MultiStep** LR decays LR by 10 every 30 epochs; **SGDR** sets T_0 to 10, T_Mult to 2 and minimum LR to $1e^{-5}$; **Adam** just uses the default parameter setting. The results are presented in Fig. 9 in the main paper.

C EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS IN SECTION 4.3

The datasets CIFAR-10-C and CIFAR-100-C Hendrycks & Dietterich (2019) can be downloaded at <https://zenodo.org/record/2535967#.Xt4mVigzZPY>, <https://zenodo.org/record/3555552#.Xt4mdSgzZPY>. Each dataset contains 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, 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 train a model with ResNet-18 for each corruption dataset. Finally, we can obtain 15 models for CIFAR-10/100-C. Each corruption can be roughly regarded as a task, and the average accuracy of 15 models on test data ² is used to evaluate the robust performance of different tasks for each LR schedules strategy.

For experimental setting in Section 4.3, 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. **Fixed** LR uses 0.1 LR during the whole training; **Exponential** LR multiplies LR with 0.95 every epoch; **MultiStep** LR decays LR by 10 every 30 epochs; **SGDR** sets T_0 to 10, T_Mult to 2 and minimum LR to $1e^{-5}$; **Adam** just uses the default parameter setting. Our method trains the ResNet-18 by SGD with a weight decay $5e^{-4}$, and the MLR-SNet is learned under the guidance of a small set of validation set without corruptions. We randomly choose 10 clean images for each class as validation set. The experimental result is listed in Table 1 in the main paper.

Additional robustness results of transferrable LR schedules on different data corruptions. Furthermore, we want to explore the robust performance of different tasks for our transferrable LR schedules. Different from above experiments where all 15 models are trained under the guidance of a small set of validation set, we just train a ResNet-18 on Gaussian Noise corruption to meta-learn the MLR-SNet, and then transfer the meta-learned LR schedules to other 14 corruptions. 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 hyper-parameter setting of hand-designed LR schedules keeps same with above. Table 1 shows the mean test accuracy of 14 models. As can be seen, our transferrable LR schedules obtain the final best performance compared with hand-designed LR schedules. This implies that our transferrable LR schedules can also perform robust and stable than the pre-set LR schedules when the learning tasks are changed. However, our transferrable LR schedules are plug-and-play, and have no additional hyper-parameters to tune when transferred to new heterogeneous tasks.

²We use the original 50,000 train images of CIFAR-10/100 as test data.

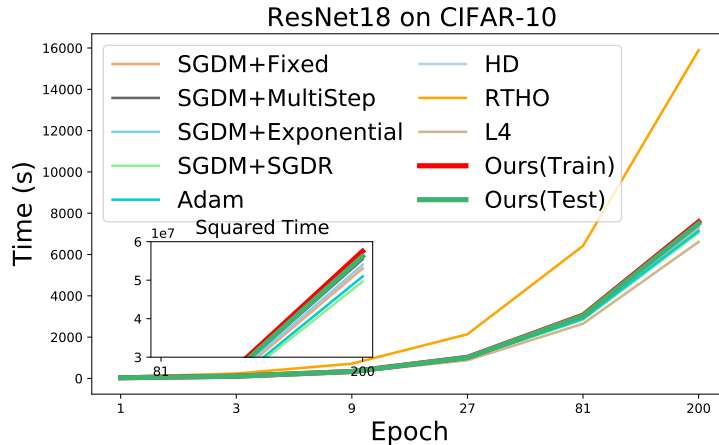


Figure 5: Time that every method consumes in training ResNet-18 on CIFAR-10 (Best viewed in color.).

D COMPUTATIONAL COMPLEXITY ANALYSIS

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. On the other hand, our transferred LR schedules predict LR for each iteration by a small MLR-SNet, whose computational cost should be significantly less than the cost of the normal network training. To empirically show the differences between hand-designed LR schedules and our method, 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 Figure 5. Except that **RTHO** costs significantly more time, other methods including MLR-SNet training and testing give similar results. Our MLR-SNet takes barely longer time to complete the training phase and due to the light-weight structure of MLR-SNet, and little extra time is added in the testing phase compared to hand-designed LR schedules. Thus our method is completely capable of practical application.

E EXPERIMENTAL RESULTS OF ADDITIONAL COMPARED METHOD LR CONTROLLER

In this section, we present the experimental results of LR Controller Xu et al. (2019), which is a related work of ours but under the reinforcement learning framework. Due to their learning algorithm is relatively computationally expensive and not very easy to optimize, we will show our method has a superiority in finding such a good LR schedule that scales and generalizes.

To start a fair comparison, we follow all the training settings and structure of LR Controller proposed in Xu et al. (2019) except that we modify the batch size to 128 and increase training steps to cover 200 epochs of data to match our setup in Section 4.1³. Firstly, we train LR Controller on CIFAR-10 with ResNet-18 and CIFAR-100 with WideResNet-28-10 as we do in Section 4.1. As shown in Fig. 6, our method demonstrates evident superiority in finding a solution with better generalization compared with LR Controller strategies. LR Controller performs steadily in the early training phase, but soon fluctuates significantly and fails to progress. This tends to show that the LR Controller suffers from a severe stability issue when training step increases, especially being compared to our MLR-SNet.

Then we transfer the LR schedules learned on CIFAR-10 for our method and LR Controller to CIFAR-100 to verify their transferability. Test settings are the same with those related in Section 4.2. As shown in Fig. 7, LR Controller makes a comparatively slower progress in the whole training

³Code for LR Controller can be found at <https://github.com/nicklashansen/adaptive-learning-rate-schedule>

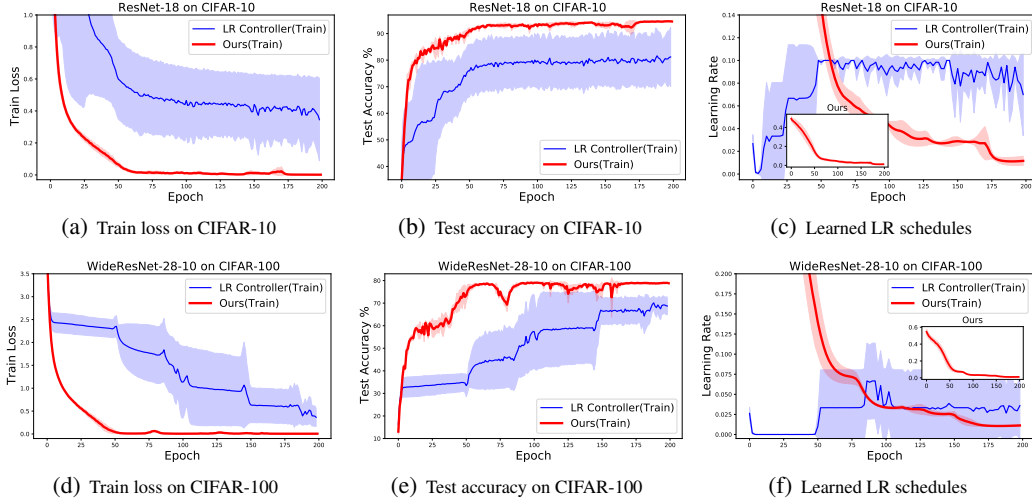


Figure 6: Train loss, test accuracy and learned LR schedules of our method(train) and LR Controller(train) on CIFAR-10 and CIFAR-100.

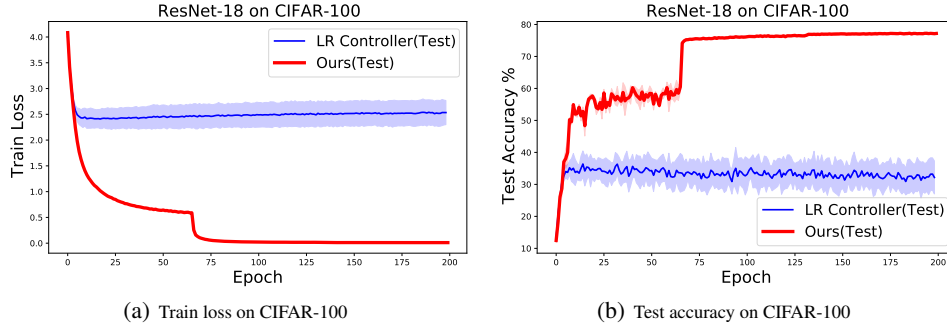


Figure 7: Train loss, test accuracy of our method(test) and LR Controller(test) on CIFAR-100.

process. While our method achieves a competitive performance, which indicates the capability of transferring to other tasks for our method.

REFERENCES

- Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *ICML*, 2017.
- Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.
- Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.
- Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
- Mitchell P Marcus and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2).
- Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- Zhen Xu, Andrew M Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule. *arXiv:1909.09712*, 2019.
- Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.