Selecting and Composing Learning Rate Policies for Deep Neural Networks

YANZHAO WU and LING LIU, Georgia Institute of Technology, USA

The choice of learning rate (LR) functions and policies has evolved from a simple fixed LR to the decaying LR and the cyclic LR, aiming to improve the accuracy and reduce the training time of Deep Neural Networks (DNNs). This article presents a systematic approach to selecting and composing an LR policy for effective DNN training to meet desired target accuracy and reduce training time within the pre-defined training iterations. It makes three original contributions. First, we develop an LR tuning mechanism for auto-verification of a given LR policy with respect to the desired accuracy goal under the pre-defined training time constraint. Second, we develop an LR policy recommendation system (LRBench) to select and compose good LR policies from the same and/or different LR functions through dynamic tuning, and avoid bad choices, for a given learning task, DNN model, and dataset. Third, we extend LRBench by supporting different DNN optimizers and show the significant mutual impact of different LR policies and different optimizers. Evaluated using popular benchmark datasets and different DNN models (LeNet, CNN3, ResNet), we show that our approach can effectively deliver high DNN test accuracy, outperform the existing recommended default LR policies, and reduce the DNN training time by 1.6–6.7× to meet a targeted model accuracy.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Heuristic function construction**; **Learning settings;**

Additional Key Words and Phrases: Learning rate, hyper-parameter optimization, Deep Neural Network, deep learning, training, accuracy

ACM Reference format:
Yanzhao Wu and Ling Liu. 2023. Selecting and Composing Learning Rate Policies for Deep Neural Networks. *ACM Trans. Intell. Syst. Technol.* 14, 2, Article 22 (February 2023), 25 pages.
https://doi.org/10.1145/3570508

1 INTRODUCTION

Hyperparameter tuning is widely recognized as a critical optimization for efficient training of **deep neural networks** (**DNNs**). A DNN is trained iteratively over the input training data $\mathcal{D}$ through forward and backward processes to update a set of trainable model parameters Θ based on the configuration of its hyperparameters $\mathcal{H}$ and the optimization algorithm of its loss function. The learning rate (η) is one of the most important hyperparameters for efficiency optimization of the DNN training algorithms. The learning rate (LR) function and the configuration policy are known

This research is partially sponsored by the National Science Foundation under Grants No. NSF 2038029 and No. NSF 1564097, and an IBM faculty award.
Authors' address: Y. Wu and L. Liu, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, Georgia 30332-0765; emails: yanzhaowu@gatech.edu, lingliu@cc.gatech.edu.

2157-6904/2023/02-ART22 $15.00
https://doi.org/10.1145/3570508

to have direct impacts on both the training efficacy and the test accuracy of the trained model. However, it is challenging to choose a good LR function, to select a good LR policy (e.g., a specific LR parameter configuration) given an LR function, and avoid bad LR policies. Even for the fixed LR function, it is non-trivial to choose a good value and avoid a bad one, since too small or too large LR value may impair the DNN training progress on both model accuracy and training time, resulting in slow convergence or even model divergence [5, 9]. The typical trial-and-error approach will try different LR values each time for training, which is tedious and time-consuming to tune the single LR value. Even with a reduced search space such as $[0.0001, 0.1]$, the possible LR values for trial-and-error can be inexplicable. Bearing with the difficulty of determining a good LR value for fixed LR, a growing trend of research efforts have been devoted to more complex LR functions ($\eta(t; \mathcal{P})$), which have multiple LR parameters instead of a single fixed value, and will change as a function of the training iterations (t). As a result, finding a good LR function and selecting a good LR policy will demand tuning multiple LR parameters for each LR function, making the hyperparameter tuning for LR a far-reaching challenge [6, 10, 22, 30, 36, 41]. Moreover, good LR policies for a given LR function tend to vary based on the specific datasets and learning tasks, and the DNN algorithms used for model training [1, 11, 28, 29, 40, 42]. In practice, empirical approaches are typically used to manually select an LR function and configure a good LR policy by choosing the concrete LR parameters through trials and errors. For example, most of the DL frameworks (e.g., TensorFlow, Caffe, and PyTorch) recommend different LR policies for different benchmark learning tasks and datasets as their default LR policies in their public releases with accuracy/training time benchmark results. Even for the same learning task and dataset, each of these DL frameworks often has different LR policies for different DNN models as the recommended default LRs. For example, TensorFlow uses a constant LR (fixed LR) with a specific LR value as its recommended LR policy for CIFAR-10 when training using AlexNet, and uses a decaying LR (NSTEP) as its default LR policy when ResNet is used for training a CIFAR-10 classifier. Many popular DNN training optimizers, such as **Stochastic Gradient Descent** (**SGD**) [8], SGD with Momentum [33], and Adam [24], utilize the LR in their optimization execution, indicating that the LR is a critical hyperparameter for DNN training. For example, for MNIST, TensorFlow chooses Adam optimizer with the fixed LR of 0.0001, and Caffe, Torch, and Theano all choose SGD optimizer with a fixed LR, but they set their default choice for the fixed LR to 0.01, 0.05, and 0.1, respectively. In comparison, for CIFAR-10, TensorFlow, Torch, and Theano choose SGD with fixed LR values of 0.1, 0.001, and 0.01, respectively, while Caffe changes its LR function to a two-step decay LR policy with 0.001 as the LR value for the first 4,000 iterations in training and 0.0001 as the updated LR value for the last 1,000 iterations of training [42]. However, when a new DNN model is used for training or an existing DNN model is trained for a new learning task or a new dataset, domain scientists and engineers have found it hard to select and compose a good LR policy and avoid worse LR choices for effective training of DNN models. The manual-tuning task for finding a good or acceptable LR policy with respect to the training accuracy objective and training time constraint can be labor-intensive and error-prone, especially given the large search space of LR values for LR functions of either single-parameter or multi-parameters. There is a high demand for designing and developing a systematic approach to selecting and composing a good LR policy for a given learning task and dataset and a given DNN training algorithm. We argue that a good LR policy can notably improve the DNN training performance on both model test accuracy and model training time, significantly alleviate the frustration of manual-tuning difficulty and costs, and more importantly, can help avoiding the bad LR policy choices that will lead to below average or poor training performance.

Bearing these objectives in mind, we present a systematic study of 15 representative LR functions from four LR algorithm families. This article makes three original contributions. *First*, we

develop an LR tuning mechanism to enable dynamically tuning and verification of LR policies with respect to desired accuracy goal and training time constraints, e.g., the pre-set #Iterations or #Epochs (the number of complete pass-throughs of the training data $\mathcal{D}$). *Second*, we develop an LR policy recommendation system (LRBench) to select and compose good LR policies from the same and/or different LR function(s) and avoid bad ones for a given learning task and a given dataset and DNN model. *Third*, we incorporate the support of different DNN optimizers and the recommendation of adaptive composite LR policy. The adaptive composite LR policy can further improve the quality of LR policy selection by enabling the creation of a multi-policy LR by combining multi-LR policies from different LR functions at different stages of the training process, boosting the overall performance of DNN model training on accuracy and training time. We evaluate our approach using four benchmark datasets—MNIST [26], CIFAR-10 [25], SVHN [32], and ImageNet [35]—and three families of DNN backbone algorithms for model training—LeNet [26], CNN3 [21], and ResNet [17]. The results show that our approach is effective and the LR policies chosen by LRBench can consistently deliver high DNN model accuracy; outperform the existing recommended default LR policies for a given DNN model, learning task, and dataset; and reduce the DNN training time by 1.6–6.7× to meet a targeted accuracy.

2 PROBLEM STATEMENT

The DNN training with a given set of hyperparameters will output a trained model F with dataset-specific model parameters (Θ). During the training, an optimizer is used to update the model parameters and improve the model performance iteratively with two important optimizations. (1) A loss function (L) is computed statistically and used to measure the prediction deviation of the DNN model output to the ground truth, which enables the optimizer to reduce and minimize the loss value (error) throughout the iterative model update process. (2) The LR policy ($\eta(t)$) is leveraged by the optimizer to control and adjust the amount of model parameter updates to be exercised during each training iteration t, which enables the optimizer to tune the rate of the update to the model parameters between slow and fast based on the specific LR value given at each training iteration. There are three primary goals for DNN training to adjust the extent of the update on the model parameters based on a specific LR policy: (i) to control the model learning speed, (ii) to avoid over-fitting to the single mini-batch, and (iii) to ensure that the model converges to global/local optimum.

Non-convex optimization algorithms are widely adopted as the optimizer for DNN training, such as SGD [8], SGD with Momentum [33], Adam [24], Nesterov [38], and so forth. For SGD, the DNN parameter update can be formalized as follows:

$$\Theta_{t+1} = \Theta_t - \eta(t)\nabla L, \tag{1}$$

where t represents the current iteration, L is the loss function, ∇L is the gradients, and $\eta(t)$ is the LR at iteration t that controls the extent of the update to the model parameters (i.e., $\Theta_{t+1} - \Theta_t = -\eta(t)\nabla L$).

For other optimizers, such as SGD with Momentum (Momentum) and Adam, they adopt a similar method to update model parameters. For example, Momentum will update the model parameters Θ as Formula (2) shows.

$$V_t = \gamma V_{t-1} - \eta(t)\nabla L, \quad \Theta_{t+1} = \Theta_t + V_t, \tag{2}$$

where V_t is the accumulated gradients at iteration t to be updated to the model parameters and γ is a coefficient applied to the previous V_{t-1}, which is typically set to 0.9. Adam is another popular

optimizer widely used in DNN training. It updates the model parameters Θ as Formula (3) shows.

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\nabla L, \quad V_t = \beta_2 V_{t-1} + (1-\beta_2)(\nabla L)^2,$$
$$\hat{M}_t = \frac{M_t}{1-\beta_1^t}, \quad \hat{V}_t = \frac{V_t}{1-\beta_2^t}, \quad \Theta_{t+1} = \Theta_t - \frac{\eta(t)}{\sqrt{\hat{V}_i}+\epsilon}\hat{M}_t, \tag{3}$$

where β_1 and β_2 are the coefficients to balance the previous accumulated gradients and the square of gradients. Typically, we will set $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Formulas (1)–(3) all contain the LR policy $\eta(t)$ as other popular optimizers do, such as Nesterov [38] and AdaDelta [43].

In addition, [4] proposed to search the optimizers for training DNNs by modeling an optimizer as Formula (4):

$$\Theta_{t+1} = \Theta_t - \eta(t) b(u_1(op_1), u_2(p_2)), \tag{4}$$

where op_1 and op_2 are the operands, such as ∇L, M_t, and V_t in Formula (3), and $u_1(.)$, $u_2(.)$, and $b(.,.)$ denote the unary and binary functions, respectively, such as mapping the input x to $-x$ and $log|x|$ for the unary functions, and addition and multiplication for the binary functions. In particular, the LR policy $\eta(t)$ still plays a critical role in the searching and optimization process.

LR optimization is a subproblem of hyperparameter optimization that is only for LR η. For DNN training, given an optimizer O and a DNN F_Θ with trainable model parameters Θ, the optimizer O minimizes the loss $L(x; F_\Theta)$ over i.i.d. samples x from a natural (ground-truth) distribution $\mathcal{G}_x$. In practice, the optimizer O will map a training dataset $\mathcal{X}^{train}$ to data-specific model parameters Θ for a given DNN F_Θ, that is, $\Theta = O(\mathcal{X}^{train})$. An important hyperparameter for the optimizer O is the LR policy $\eta = \eta(t)$. With the chosen η, we have the optimizer O_η and $\Theta = O_\eta(\mathcal{X}^{train})$. LR optimization aims at identifying a good LR policy η to minimize the generalization error $\mathbb{E}_{x\sim\mathcal{G}_x}[L(x; F_{O_\eta(\mathcal{X}^{train})})]$. In practice, we use a validation dataset $\mathcal{X}^{val}$ to estimate the generalization error, that is, $L_{x\in\mathcal{X}^{val}}(x; F_{O_\eta(\mathcal{X}^{train})})$. $\mathcal{P}$ denotes the set containing all possible LR policies, and the LR optimization problem is formalized as Formula (5):

$$\hat{\eta} = \underset{\eta\in\mathcal{P}}{argmin}\, L_{x\in\mathcal{X}^{val}}(x; F_{O_\eta(\mathcal{X}^{train})}). \tag{5}$$

Different from other hyperparameters, such as the weight decay rate, number of filters, and kernel size, which are typically constant in the entire training process, the LR may change over the training iteration t. In practice, we choose a finite set of S LR policies, consisting of different LR functions, denoted as $\mathcal{P}^* \subseteq \mathcal{P}$ and $\mathcal{P}^* = \{\eta^1(t), \eta^2(t), \ldots, \eta^S(t)\}$, e.g., $\eta^1(t) = k$ (a fixed LR) and $\eta^2(t) = \gamma^t$ ($\gamma < 1$). Hence, we can formalize the LR optimization as Formula (6). That is to select or compose the optimal LR policy from the candidate set $\mathcal{P}^* = \{\eta^1(t), \eta^2(t), \ldots, \eta^S(t)\}$.

$$\hat{\eta} = \underset{\eta\in\mathcal{P}^*=\{\eta^1(t),\ldots,\eta^S(t)\}}{argmin} L_{x\in\mathcal{X}^{val}}(x; F_{O_\eta(\mathcal{X}^{train})}). \tag{6}$$

3 LR SELECTION AND COMPOSITION

LR is a function of the training iteration t with a set of parameters and a method to determine the LR value at each iteration t of the overall training process. An LR policy specifies a concrete parameter setting of an LR function. For example, a fixed LR of 0.01 is an LR policy of constant LR method with a fixed value of 0.01 throughout all iterations of the model training. Another example is the two-step LR policy of 0.01 in the first half of the training iterations and 0.001 in the second half of the training iterations. In this section, we will cover a total of 15 functions from three families of LR functions: fixed LRs, decaying LRs, and cyclic LRs. We use the term of single LR policy to refer to the LR policy that corresponds to a single LR function, and refer to the LR policy that is defined by combining multiple LR policies from two or more LR functions as composite LR

Table 1. Decaying Functions $g(t)$ for Decaying LRs

Abbr.	$g(t)$	Schedule	Param	#Param
STEP	$\gamma^{floor(t/l)}$	t, l	γ, l	2
NSTEP	$l_0, \ldots, l_{n-1}$, $\gamma^i,\ i \in \mathbb{N}\ s.t.$ $l_{i-1} \leq t < l_i$	t, l_i	γ, l_i	$n+1$
EXP	γ^t	t	γ	1
INV	$\frac{1}{(1+t\gamma)^p}$	t	γ, p	2
POLY	$(1-\frac{t}{max\_iter})^p$	t	p	1

or multi-policy LR. We first describe our approach to select single LR policy for a given learning task, dataset, and DNN backbone algorithm for model training. Then, we introduce our composite LR scheme for selecting and composing an adaptive LR policy to further boost the overall training performance in terms of accuracy or training time given a target accuracy.

3.1 Single Policy LR

Fixed LRs (FIX), also called constant LRs, use a pre-selected fixed LR value throughout the entire training process, represented by $\eta(t) = k$ with k as the only hyperparameter to tune. However, too small k value may slow down the training progress significantly. Too large k value may accelerate the training progress at the cost of causing the loss function to fluctuate wildly, making the training fail to converge, resulting in very low accuracy. The conservative approach is popularly used, which uses a small value to ensure the model convergence and avoid oscillating loss (e.g., 0.01 for MNIST on LeNet and 0.001 for CIFAR-10 on CNN3). However, choosing a small and yet good fixed LR value is challenging. Even for the same learning task and dataset, e.g., CIFAR-10, different DNN models need to choose different constant values to meet the target accuracy goal (e.g., for CIFAR-10, 0.001 on CNN3 and 0.1 on ResNet-32). Another limitation of fixed LR policies is that it cannot adapt to the needs of different learning speeds during different training stages of the entire iterative learning process, and thus suffers from reaching the peak accuracy due to missing the speed-up opportunity when the training is on the plateau or failing to converge at the end of training.

3.2 Composite Policy LRs

There are two types of composite LR schemes, according to whether the LRs are composed using the same LR function or using two or more different LR functions. The former is coined as the *homogeneous multi-policy* LRs and the latter is called the *heterogeneous multi-policy* LRs.

Decaying LRs improve the limitation of the fixed LRs by using decreasing LR values during training. Similar to simulated annealing, training with a decaying LR starts with a relatively large LR value, which is reduced gradually throughout the training, aiming to accelerate the learning process while ensure that the training converges with good accuracy or meeting the target accuracy. A decaying LR policy is defined by a decay function $g(t)$ and a constant coefficient k, denoted by $\eta(t) = kg(t)$. $g(t)$ gradually decreases from the upper bound of 1 as the number of iterations (t) increases, and the constant k value serves as the starting LR.

Table 1 lists the five most popular decaying LRs, supported in LRBench. The STEP function defines the LR policy at iteration t with two parameters: a fixed step size l ($l > 1$) and an exponential factor γ. The LR value is initialized with k and decays every l iteration by γ. The NSTEP enriches STEP by introducing n variable step sizes, denoted by $l_0, l_1, \ldots, l_{n-1}$, instead of the one fixed step size l. NSTEP is initialized by k ($g(t) = 1$ when $i = 0, t < l_0$) and computed by γ^i (when $i > 0$ and $l_{i-1} \leq t < l_i$). EXP is an LR function defined by an exponential function (γ^t). Although EXP, STEP,

Table 2. Cyclic Functions $g(t)$ for Cyclic LRs

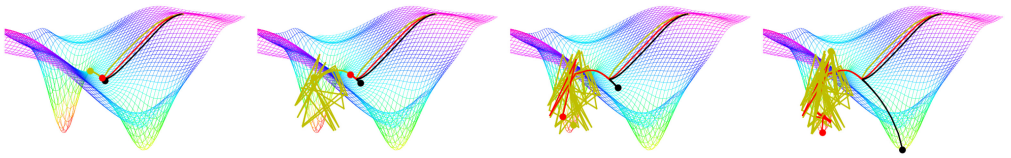
Abbr.	$g(t)$	Schedule	Param	#Param
TRI	$TRI(t) = \frac{2}{\pi}\lvert arcsin(sin(\frac{\pi t}{2l}))\rvert$	t, l	l	1
TRI2	$\frac{1}{2^{floor(\frac{t}{2l})}} TRI(t)$	t, l	l	1
TRIEXP	$\gamma^t TRI(t)$	t, l	γ, l	2
COS	$COS(t) = \frac{1}{2}(1 + cos(\pi \frac{t}{l}))$	t, l	l	1
SIN	$SIN(t) = \lvert sin(\pi \frac{t}{2l})\rvert$	t, l	l	1

and NSTEP all use an exponential function to define $g(t)$, their choice of concrete γ is different. To avoid the LR decaying too fast due to the exponential explosion, EXP uses a γ that is close to 1, e.g., 0.99994 and reduces the LR value every iteration. In contrast, STEP and NSTEP employ a small γ, e.g., 0.1, and decay the LR value using one fixed step size l or using n variable step sizes l_i. The total number of steps is determined, for STEP, by the step size and the pre-defined training #Iterations (or #Epochs), and for NSTEP, n is typically small, e.g., 2–5 steps. Other decaying LRs are based on **the inverse time function** (**INV**) and the **polynomial function** (**POLY**) with parameter p as shown in Table 1. A good selection of the value settings for these parameters is challenging and yet critical for achieving effective training performance when using decaying LR policies.

Several studies [5, 17, 25] show that with a good selection of the decaying LR policies, they are more effective than the fixed LRs for improving training performance on accuracy and training time. However, [30, 36, 37, 41] recently show that decaying LRs tend to miss the opportunity to accelerate the training progress on the plateau in the middle of training when initialized with too small LR values and result in slow convergence and/or low accuracy. While larger initial values for decaying LRs can lead to the same problem as those for fixed LRs.

STEP is an example of homogeneous multi-policy LRs created using a single LR function FIX by employing multiple fixed LRs defined by the LR update schedule based on training iteration t and step size l. NSTEP is another example of homogeneous multi-policy LRs by employing n different FIX policies, each changing the LR value based on t and the n different step size: l_i $(i = 0, \ldots, n-1)$. **Cyclic LRs (CLRs)** are proposed recently by [30, 36, 37], to address the above issue of decaying LRs. CLRs by design change the LR cyclically within a pre-defined value range, instead of using a fixed value or reducing the LR by following a decaying policy, and some target accuracy thresholds can be achieved earlier with CLRs in shorter training time and smaller #Epochs or #Iterations. In general, cyclic LRs can be defined by $\eta(t) = |k_0 - k_1|g(t) + min(k_0, k_1)$, where k_0 and k_1 specify the upper and lower value boundaries, $g(t)$ represents the cyclic function whose domain ranges from 0 to 1, and $min(k_0, k_1) \leq \eta(t) \leq max(k_0, k_1)$. For each CLR policy, three important parameters should be specified: k_0 and k_1, which specify the initial cyclic boundary, and the half-cycle length l, defined by the half of the cycle interval, similar to the step size l used in decaying LRs. The good selection of these LR parameter settings is challenging and yet critical for the DNN training effectiveness under a cyclic LR policy (see Section 5 for details).

We support three types of CLRs currently in LRBench: triangle-LRs, sine-LRs, and cosine-LRs as Table 2 shows. TRI is formulated with a triangle wave function $TRI(t)$ bounded by k_0 and k_1. TRI2 and TRIEXP are two variants of TRI by multiplying $TRI(t)$ with a decaying function, $\frac{1}{2^{floor(\frac{t}{2l})}}$ for TRI2 and γ^t for TRIEXP. TRI2 reduces the LR boundary ($|k_0 - k_1|$) every $2l$ iterations while TRIEXP decreases the LR boundary exponentially. [30] proposed a cosine function with warm restart and can be seen as another type of CLRs, and we denote it as COS. We implement COS2 and COSEXP as the two variants of COS corresponding to TRI2 and TRIEXP in LRBench and also implement SIN, SIN2, and SINEXP as the sine-CLRs in LRBench.



(a) At the 70th iteration (b) At the 100th iteration (c) At the 120th iteration (d) At the 149th iteration

Fig. 1. Visualization of the training process with different LRs.

We visualize the optimization process of three LR functions, FIX, NSTEP, and TRIEXP, from the above three categories in Figure 1. The corresponding LR policies are FIX ($k = 0.025$) in black, NSTEP ($k = 0.05, \gamma = 0.5, l = [120, 130]$) in red, and TRIEXP ($k_0 = 0.05, k_1 = 0.25, \gamma = 0.9, l = 25$) in yellow. All three LRs start from the same initial point, and the optimization process lasts for 150 iterations. The color on the grid marks the value of the cost function to be minimized, where the global optimum corresponds to red. In general, we observe that different LRs lead to different optimization paths. Although three LRs exhibit little difference in terms of the optimization path up to the 70th iteration, the FIX lands at a different local optimum at the 149th iteration rather than the global optimum (red). It shows that the accumulated impacts of LR values could result in sub-optimal training results. Also, at the beginning of the optimization process, TRIEXP achieved the fastest progress according to Figure 1(a) and (b). This observation indicates that starting with a high LR value may help accelerate the training process. However, the relatively high LR values of TRIEXP in the late stage of training also introduce high "kinetic energy" into this optimization and therefore, the cost values are bouncing around chaotically as the TRIEXP optimization path (yellow) shows in Figure 1(b)–(d), where the model may not converge. This example also illustrates that simply using a single LR function may not achieve the optimal optimization progress due to the lack of flexibility to adapt to different training states.

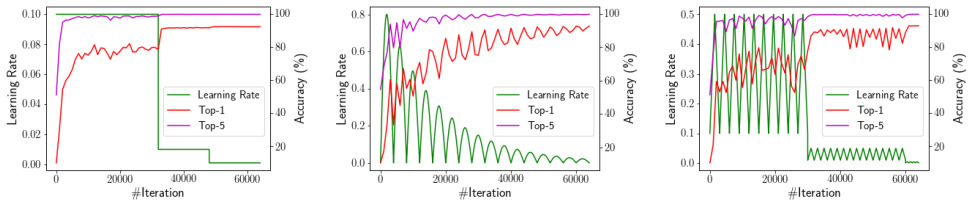
Studies by [30, 36, 37, 41] confirm independently that increasing the LR cyclically during the training is more effective than simply decaying LRs continuously as training progresses. On the one hand, with relatively large LR values, cyclic LRs can accelerate the training progress when the model is trapped on the plateau and escape the local optimum by updating the model parameters more aggressively. On the other hand, cyclic LRs, especially decaying cyclic LRs (i.e., TRI2, TRIEXP) can further reduce the boundaries of LR values, as training progresses through different stages, such as the training initialization phase, the middle stage of training in which the model being trained is trapped on a plateau or a local optimum, and the final convergence phase. We argue that the LR should be actively changed to accommodate different training phases. However, existing LRs, such as the fixed LRs, decaying LRs, and cyclic LRs are all defined by one function with one set of parameters. Such design limits its adaptability to different training phases and often results in slow training and sub-optimal training results.

According to our characterization of single-policy and multi-policy LRs, we can view TRI2, COS2, and SIN2 as examples of homogeneous multi-policy LRs, because they are created by combining multiple LR policies through a simple and straightforward integration of decaying LR policy to refine the cyclic LR policy using decaying (k_0, k_1) by one half every $2l$ iterations. We have analyzed how and why these multi-policy LRs provide more flexibility and adaptability in selecting and composing a multi-policy LR mechanism for more effective training of DNNs in this section. Our empirical results also confirm consistently that multi-policy LRs hold the potential to further improve the test accuracy of the trained DNN models.

Advanced Composite LRs. There are different combinations of multi-policy LRs that can be beneficial for further boosting the DNN model training performance. Consider a new multi-policy

Table 3. Different LR Policies for CIFAR-10 on ResNet-32, Showing the Benefit of Advanced Composite LRs

LR Policy		#Iter	Accuracy (%)
Category	LR Function		
Fixed LR	FIX (k = 0.1)	61,000	86.08
Decaying LR	NSTEP (k = 0.1, γ = 0.1, l = [32,000, 48,000])	53,000	92.38 ± 0.04
Cyclic LR	SINEXP (k_0 = 0.0001, k_1 = 0.9, γ = 0.99994)	64,000	92.81 ± 0.08
Advanced Composite LR	0–30,000 iter: TRI (k_0 = 0.1, k_1 = 0.5, l = 1500)	64,000	**92.91**
	30,000–60,000 iter: TRI (k_0 = 0.01, k_1 = 0.05, l = 1000)		
	60,000–64,000 iter: TRI (k_0 = 0.001, k_1 = 0.005, l = 500)		



(a) Decaying LR (NSTEP) (b) Cyclic LR (SINEXP) (c) Multi-policy LR

Fig. 2. Comparison of NSTEP, SINEXP, and Heterogeneous Multi-policy LRs (CIFAR-10, ResNet-32), showing the benefit of advanced composite LR in performance improvement during the latter half of the training.

LR mechanism; it is created by composing three triangle LRs in three different training stages with decaying cyclic upper and lower bounds and varying decaying step sizes over the total training iterations. Table 3 shows the effectiveness of this multi-policy LR policy through a comparative experiment on ResNet-32 with the CIFAR-10 dataset using the Caffe DNN framework. The experiment compares four scenarios. (1) The single-policy LR with the fixed value of 0.1 achieves an accuracy of 86.08% in 61,000 iterations out of the default total training iterations of 64,000. (2) The decaying LR policy of the NSTEP function with specific k, γ, l achieves the test accuracy of 92.38% for the trained model with only 53,000 iterations out of 64,000 total iterations. (3) The cyclic LR policy of the SINEXP function with specific LR parameters (k_0, k_1, γ) achieves the accuracy of 92.81% at the end of 64,000 total training iterations. (4) The advanced composite multi-policy LR mechanism is created by LRBench for CIFAR-10 (ResNet-32), which achieves the highest test accuracy of 92.91% at the total training rounds of 64,000, compared to other three LR mechanisms. Figure 2 further illustrates the empirical comparison results. It shows the LR (green curves) and Top-1/Top-5 accuracy (red/purple curves) for the top three performing LR policies: NSTEP, SINEXP, and the advanced composite LR.

We highlight three interesting observations. *First,* this multi-policy LR achieved the highest accuracy of 92.91%, followed by the cyclic LR (SINEXP, 92.81%) and decaying LR (NSTEP, 92.38%) while the fixed LR achieved the lowest 86.08% accuracy. *Second,* from the LR value y-axis (left) and the accuracy y-axis (right) in Figure 2(a) and (b), we observe that the accuracy increases as the overall LRs decrease over the iterative training process (t on the x-axis). However, without dynamically adjusting the LR values in a cyclic manner, NSTEP missed the opportunity to achieve higher accuracy by simply decaying its initial LR value over the iterative training process. *Third,* Figure 2(b) shows that even with a cyclic LR policy such as SINEXP, it fails to compete with the new multi-policy LR that we have designed because it fails to decay faster and at the end of the training iterations, it still uses much higher LRs. In comparison, Figure 2(c) shows the training efficiency using our new multi-policy LR for CIFAR-10 on ResNet-32. It uses three different CLR

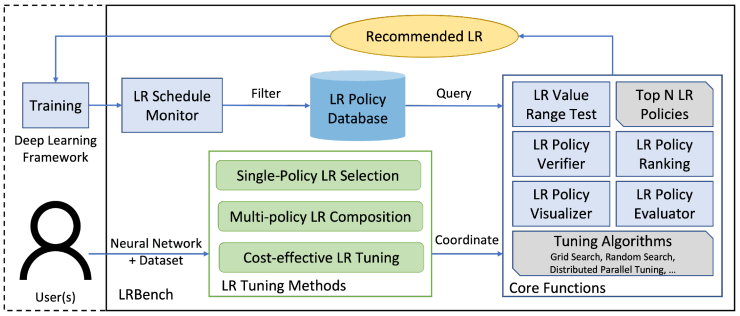


Fig. 3. LRBench architecture overview.

functions in three different stages of the overall 64,000 training iterations, each stage reducing the cyclic range by one-tenth decaying of k_0 and k_1 and reducing the step size l by 500 iterations.

Recent research has shown that variable LRs are advantageous by empowering training with adaptability to dynamically change the LR value throughout the entire training process to adapt to the need of different learning modes (learning speed, step size, and direction for changing learning speed) at different training phases. A key challenge for defining a good multi-policy LR, be it homogeneous or heterogeneous, is related to the problem of how to determine the dynamic settings of its parameters, such as l and k_0, k_1. Existing hyperparameter search tools are typically designed for tuning the parameters at the initialization of the DNN training, which will not change during training, such as the number of DNN layers, the batch size, the number of iterations/epochs, and the optimizer (SGD, Adam, etc.). Thus, an open problem of selecting and composing LR policies with evolving LR parameters is to develop a systematic approach to selecting and composing LRs. In the next section, we present our framework, called LRBench, to create a good LR policy by dynamically tuning, composing, and selecting a good LR policy for a learning task and a given dataset on a chosen DNN model.

4 DESIGN OVERVIEW

LRBench is a **Learning Rate Benchmarking** system for tuning, composing, and selecting LR policies for DNN training. The current prototype of LRBench implemented 15 LR functions in four families of LR policies and eight evaluation metrics, covering utility, cost, and robustness of LR policies defined by [41]. Figure 3 shows the architecture of LRBench, consisting of eight functional components. (1) The LR schedule monitor tracks the training status and triggers the LR value update based on the specific LR policy. (2) The LR policy database stores the empirical LR tuning results organized by the specific neural network and dataset. (3) The LR value range test will estimate a good LR value range to reduce the LR search space. (4) The LR policy verifier will perform the verification of the given LR policy in terms of whether it can meet the desired target accuracy or achieve optimal training performance. (5) The LR policy visualizer will help users visualize the LR values and DNN training progress. (6) The LR policy ranking algorithm will select the top N ranked LR policies based on the empirical LR tuning results from the LR policy database for a pre-defined utility, cost, or robustness metric. (7) The LR policy evaluator estimates the good LR parameters, such as the step size l, for evaluating and comparing alternative LR policies and dynamically tuning LR values. (8) The tuning algorithms in LRBench are provided by Ray Tune [28], including grid search, random search, and distributed parallel tuning algorithms to effectively explore the LR search space.

In practice, users can leverage a set of functional APIs provided by LRBench in their DNN training. For example, LRBench provides three main LR tuning methods: (1) single-policy LR selection,

(2) multi-policy LR composition, and (3) cost-effective LR tuning. The first two LR tuning methods aim at achieving the highest possible model accuracy under a training time constraint, such as the number of training iterations. The third tuning method targets at reducing the training time cost for achieving a desired target accuracy. In some cases, such as the pruning in neural network search [44], the goal for DNN training is to achieve a certain accuracy threshold, where a low training cost, such as using a small number of iterations, will significantly improve the training efficiency. LRBench leverages two complementary principles in LR tuning, which are **exploration** and **exploitation**. Exploration helps LRBench explore different LR functions and LR parameters in DNN training, which can avoid falling into sub-optimal LRs or even bad ones. Exploitation allows LRBench to follow a few good LR policies and verify whether these LRs can help the DNN converge to high accuracy under the given training time constraints. A proper balance between these two principles is crucial for efficient LR tuning. For example, the single-policy LR selection restricts the LR to a single policy and puts more effort into exploitation, compared to the multi-policy LR composition, which allows composing multiple LR policies and emphasizes more on exploration.

Implementation Details. We have open-sourced LRBench on GitHub at https://github.com/git-disl/LRBench, which is written in Python and has a flexible modular design that allows users to leverage different modules to perform LR tuning. The frontend and LR policy database are implemented with Django and PostgreSQL, respectively. LRBench supports popular deep learning frameworks, such as Caffe, PyTorch, Keras, and TensorFlow through their Python APIs.

In this section, we describe LR tuning and LR verification, which are the two important features in LRBench to support automatic creation of new multi-policy LRs in the scenarios where the target accuracy is not met or can be improved by LRBench during its LR verification process.

4.1 LR Tuning

The LR is an important hyperparameter, critical for efficient DNN training. Even for the fixed LR, it is non-trivial to find a good value. Advanced LRs, such as cyclic LRs and composite LRs, introduce more LR parameters to control the LR values, making this process even harder. LRBench is designed to mitigate these issues with three important components: the LR Policy Database, LR Value Range Test, and LR Schedule Monitor.

LR Policy Database stores the empirical LR policy tuning results, including good LR value ranges, organized by the specific dataset, the specific learning task, the chosen DNN model (e.g., LeNet, ResNet, AlexNet), and the deep learning framework. With the LR policy database, LRBench is able to rank existing LR policies based on a set of metrics, such as the loss, accuracy, and training cost, and recommend a good LR policy for DNN training for a learning task on a given dataset with a chosen DNN model. It does so by first searching the LR policy database, and if the relevant LR policies are found, LRBench will use the stored LR policies, denoted as $\hat{\mathcal{P}} \subseteq \mathcal{P}$, as the starting point for performing LR tuning. Upon identifying a set of three good LR policies that can meet the target accuracy within the pre-defined training iterations, LRBench will select the highest ranked LR policy as the tuning output. Otherwise, LRBench will trigger the new LR policy generation process, which will perform the LR value range test to find the range of good LR values.

LR Value Range Test. The goal of the LR value range test is to significantly reduce the search space of LR parameters for finding the good candidate LR policies. We first illustrate by example how to determine the proper ranges for training CNN3 on CIFAR-10. Figure 4 shows the results of Top-1 accuracy (y-axis) by varying the fixed LR values in a base-10 log scale (x-axis). The two red vertical dashed lines represent two fixed LR values, $k = 0.0005$ and 0.006, and the three black dashed lines represent another three fixed LR values, $k = 0.0001$, 0.001, and 0.01. The different colors of curves correspond to the Top-1 accuracy obtained when

training using different total #Epochs. We observe from Figure 4 that the appropriate LR ranges is $[0.0005, 0.006]$ marked by the two red dashed lines, and the accuracy drops much faster after $k = 0.006$. Thus, setting the upper bound of the LR range to be around 0.006 is recommended. Similarly, the lower bound by $k = 0.0005$ is the LR value in which the accuracy increase is relatively stabilized, indicating that $[0.0005, 0.006]$ is a good LR value range for CNN3 training on CIFAR-10. Another interesting observation is that with five different accuracy curves obtained using five different training #Epochs (in five colors), the general accuracy trend over varying LR values (x-axis) reveals similar patterns. This shows that it is sufficient for LRBench to examine the LR value range using a small number of variations in #Epochs instead of enumerating a large number of possible #Epochs, enabling LRBench to accelerate this process, which can also be automated by using grid or random search algorithms in LRBench. In comparison, the total of 140 epochs is used in Caffe as the recommended training time constraint for training CNN3 on CIFAR-10 (the default). This verification and tuning process also enables LRBench to remove worse LR policies through a small number of trial-and-error operations. For instance, by investigating the relationship between test accuracy and the good LR range, we can choose the fixed LR value from a small set of good LR values and significantly reduce the search space of finding a good LR policy by 99.4% for CIFAR-10, computed by $(1 - (0.006 - 0.00005)) \times 100\%/1$, compared to using the search space defined by the domain $[0, 1]$ of any given LR function for finding a good LR value under a given training #Epochs.

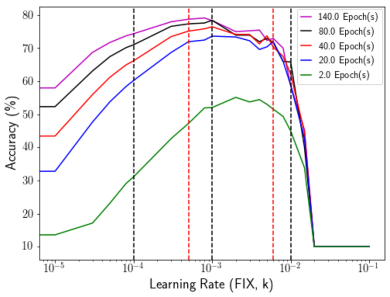


Fig. 4. Acc with varying k (FIX, CNN3, CIFAR-10).

LR Schedule Monitor is a system facility that is used for both LR tuning and LR verification. It monitors the training status, such as the training/validation loss, and dynamically changes the LR value to ensure effective training for meeting the target accuracy. Reduce-LR-On-Plateau is a popular method to tune LR on-the-fly during training included in [23], which monitors the training/validation loss during the DNN training, when the learning stagnates on plateau, e.g., no improvement for the loss, Reduce-LR-On-Plateau will reduce the LR value to facilitate DNN training. However, Reduce-LR-On-Plateau is limited to multiple decreasing fixed LR values and the FIX LR function. When the training is trapped on a plateau, increasing LR by a CLR function or even using a decaying LR function can be more beneficial and lead to faster training convergence and higher accuracy according to empirical studies [12, 37]. In LRBench, we implement the Change-LR-On-Plateau method as Algorithm 1 shows to compose multi-policy LRs. Our Change-LR-On-Plateau method allows changing the current LR value at different phases of the training based on the specific LR function ($p(t)$) used, be it a decaying LR function or a CLR function, rather than being limited to only the decreasing fixed LR values as Reduce-LR-On-Plateau. i indicates the current LR function index in the input LR functions $\mathcal{P}$ (ordered in the descending order) for the LR value η, which will be used in the optimizer $O_\eta(X^{train})$. $\mathcal{M}$ will keep track of the monitored metrics m, and t indicates the current training iteration. Is-Trapped-On-Plateau-Action($\mathcal{M}, m, t$) will take these three variables and judge whether the current training is trapped on the plateau, such as for a certain number of consecutive training iterations, e.g., five iterations, whether the improvement of the monitored metrics are beyond a threshold, e.g., 0.05 for the training loss. Then, if the current training is not trapped on plateau, it will just return no action. Otherwise, it will return specific actions to change the index i to switch between the LR functions. For example, during the beginning

ALGORITHM 1: Change-LR-On-Plateau

procedure CHANGE-LR-ON-PLATEAU(m, $\mathcal{P}$, i, training stop condition)
  Input: (1) the monitored metrics m, such as the training loss; (2) the chosen n LR functions $\mathcal{P} = \{p^1(t), p^2(t), \ldots, p^n(t)\}\ s.t.\ p^1(t) \geq p^2(t) \geq \cdots \geq p^n(t)$; (3) the starting LR function index i; (4) the training stop condition, such as a pre-defined #Iterations.
  Output: The trained DNN model F_Θ.
  ▷ $\mathcal{M}$: store monitored metrics, t: the current iteration
  Initialize $t = 0$, $\eta = p^i(t)$, $\mathcal{M} = \{\}$;
  Initialize the DNN model F_Θ;
  while the training stop condition is not met **do**
    training F_Θ with $O_\eta(X^{train})$ for one iteration;
    obtain the monitored metrics m;
    action = Is-Trapped-On-Plateau-Action($\mathcal{M}$, m, t);
    if action == "increase" **then**
      $i = i - 1$;
    end if
    if action == "decrease" **then**
      $i = i + 1$;
    end if
    $\mathcal{M}$.append(m); $t = t + 1$;
  end while
return F_Θ;
end procedure

and middle of the training (i.e., t is smaller than a pre-defined threshold), Is-Trapped-On-Plateau-Action($\mathcal{M}, m, t$) will return "increase" to increase the LR value with $i = i - 1$, allowing the training to quickly escape the plateau or local optimum, and when the training approaches the end of the pre-defined training iterations, it will return "decrease" with $i = i + 1$ to reduce the LR value to help converge the DNN training.

4.2 LR Verification

When a new DNN model is chosen for an existing dataset and learning task such as CIFAR-10, even when we use the same DL framework such as Caffe or TensorFlow, the system recommended LR policy for CIFAR-10 on AlexNet is no longer a good LR policy for ResNet. Similarly, the LR policy for MNIST with the 10-class classifier is not a good policy for CIFAR-10 either with only 77.26% accuracy compared to 81.61% of the Caffe default NSTEP policy on CNN3. This motivates us to incorporate a new functional component, called the LR policy verification, in our LRBench.

Our LR policy verification is carried out in three validation phases. In the first validation phase, we utilize the DNN training to perform the verification of the given LR policy in terms of whether it meets the desired target accuracy given by the user. If yes, then this policy is verified. Otherwise, we enter the second validation phase, which searches the LR policy database according to the learning task, the dataset, or the DNN model and select the top three ranked LR policies based on the Top-1 accuracy, for example. Other ranking metrics, such as Top-5 accuracy, training cost, and robustness of the trained model, can also be employed in conjunction with Top-1 accuracy. Upon identifying the top three good LR policies, we select the highest ranked LR policy and compare it with the LR policy being verified. By good, we mean that all meet the target accuracy within the pre-defined training time constraints (e.g., #Iterations or #Epochs). If the LR policy being verified is no longer the winner in ranking, the verification will output the top ranked LR policy as the LRBench recommended LR policy as the replacement. The verification process is terminated. If the policy being verified remains to be the winner, the verification enters the last phase, in which LRBench will trigger the new LR policy generation process, which will perform the LR value range

test. It aims to effectively reduce the search space of LR parameters for finding the good candidate LR policies through several mechanisms, such as the grid search or random search algorithms in LRBench. A challenge for the third verification phase is to estimate the good LR value based on the current or the recent past model parameters. One approach is to use the optimal learning rate (denoted as M-opt LR) of the three most recent intermediate learning outcomes across two steps. Let Θ_t denote the model parameter in the current training iteration t, η_{t+1} be the actual LR value used for the iteration $t+1$, and η^* denote the calculated LR value across two consecutive model parameter updates, i.e., $(\Theta_{t+1} - \Theta_t)$ and $(\Theta_{t+2} - \Theta_{t+1})$. One can estimate the M-opt η^* using the following formula [37] on Θ_t, Θ_{t+1} and Θ_{t+2}:

$$\begin{aligned} \eta^* &= \eta_{t+1} \frac{||\Theta_{t+1} - \Theta_t||_1}{||(\Theta_{t+1} - \Theta_t) - (\Theta_{t+2} - \Theta_{t+1})||_1} \\ &= \eta_{t+1} \frac{||\Theta_{t+1} - \Theta_t||_1}{||2\Theta_{t+1} - \Theta_t - \Theta_{t+2}||_1}. \end{aligned} \tag{7}$$

The intuition of Formula (7) is from the Newton's method for optimization, which will use the Hessian matrix (second derivative) H_L of the loss function L to update the model parameters as Formula (8) shows. The high computation cost of the Hessian matrix makes it infeasible to obtain the exact Hessian matrix for real optimization. [31] proposed the Hessian-free optimization to obtain an estimate of the Hessian $h_L(\vartheta) \in H_L$ where ϑ is a single variable in Θ from two gradients as Formula (9) shows and δ should be in the direction of the steepest descent. Based on the estimation, [16] further proposed an efficient learning rate calculated with Formula (10) on the single variable $\vartheta \in \Theta$. From Formula (1) on SGD, we have Formulas (11) and (12) on the single variable ϑ. With Formulas (10), (11), and (12), we derive Formula (13) based on $\eta_{t+1} \approx \eta_t$, indicating the effective LR for a single variable $\vartheta \in \Theta$. Hence, for estimating the optimal LR for Θ, we applied the L-1 normalization in Formula (7) as [37] suggests.

$$\Theta_{t+1} = \Theta_t - H_L^{-1} \nabla L, \tag{8}$$

$$h_L(\vartheta) = \lim_{\delta \to 0} \frac{\nabla L(\vartheta + \delta) - \nabla L(\vartheta)}{\delta}, \tag{9}$$

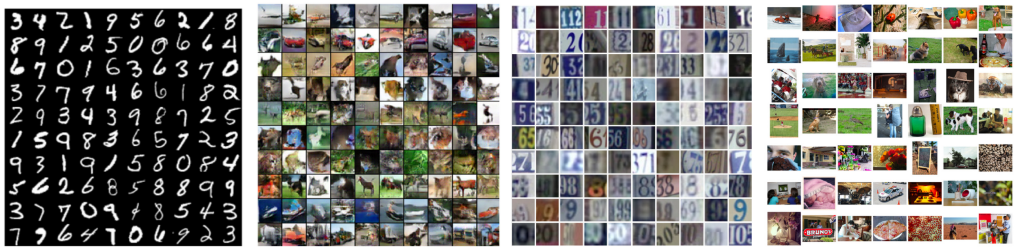
$$\eta^*_\vartheta \approx \frac{\vartheta_{t+1} - \vartheta_t}{\nabla L(\vartheta_{t+1}) - \nabla L(\vartheta_t)}, \tag{10}$$

$$\vartheta_{t+1} - \vartheta_t = -\eta_t \nabla L(\vartheta_t), \tag{11}$$

$$\vartheta_{t+2} - \vartheta_{t+1} = -\eta_{t+1} \nabla L(\vartheta_{t+1}), \tag{12}$$

$$\eta^*_\vartheta \approx \eta_{t+1} \frac{\vartheta_{t+1} - \vartheta_t}{2\vartheta_{t+1} - \vartheta_t - \vartheta_{t+2}}. \tag{13}$$

If the computed M-opt LR value is better than the one produced by the given LR policy to be verified, LRBench will use the computed M-opt LR values and output it as a recommended replacement. Thus, the LR verification process is performing both validation and tuning in order to produce a better LR policy for the given learning task, dataset, and DNN model for a given deep learning framework (e.g., Caffe, TensorFlow, PyTorch).



(a) MNIST (b) CIFAR-10 (c) SVHN (d) ImageNet

Fig. 5. The comparison of four datasets.

5 EXPERIMENTAL ANALYSIS

We conduct experiments using LRBench on top of Caffe with four popular benchmark datasets, MNIST, CIFAR-10, SVHN, and ImageNet, and three DNN models, LeNet, CNN3, and ResNet. Figure 5 shows the example images from these four datasets. We report four important results. *First,* we show the impact of the LR selection on different datasets and DNN models and the experimental characterization of different types of LR policies, with respect to highest accuracy within the training time constraint. *Second,* we show how to use LRBench to select and compose good LR policies for a given learning task, dataset, and DNN model as well as the effectiveness of LR verification, and demonstrate that advanced composite LRs are beneficial for accelerating DNN training and achieving high accuracy. *Third,* we show how to efficiently choose good LR policies to achieve the desired accuracy with reduced training time cost. *Fourth,* we consider other hyperparameters, such as the choice of a specific optimizer, and how different optimizers will impact the DNN training performance with different LR policies. In all experiments, we only vary the LR policy and keep all the other settings of hyperparameters unchanged for the given learning task with the dataset, e.g., CIFAR-10, and the chosen DNN model, i.e., CNN3 or ResNet-32. We use the averaged values from five repeated experiments with mean±std for the most critical experiments. All experiments reported in this article were conducted on an Intel Xeon E5-1620 server with Nvidia GTX 1080Ti GPU, installed with Ubuntu 16.04, CUDA 8.0, and cuDNN 6.0.

5.1 LR Policy Selection

Here, we perform an experimental comparison of different LR policies with LRBench using MNIST and CIFAR-10 to show the impact of selecting a good LR policy on the DNN model accuracy.

MNIST (LeNet). Table 4 shows the results of LeNet training on MNIST with the highest accuracy (Highest Acc (%)) found under the Caffe default training iterations of 10,000 and batch size of 100. We highlight three interesting observations. *First,* the model accuracy trained by the baseline fixed LR is sensitive to the specific k values. For example, when $k = 0.1$, the LeNet failed to converge and resulted in a very low model accuracy, i.e., 11.35%. Hence, it is very hard to select a proper constant LR value for the fixed LR. *Second,* all decaying LRs can produce high accuracy above 99.12%. These decaying LR policies are recommended by LRBench based on the optimal $k = 0.01$ from these fixed LRs. This observation shows that such selection of LR parameters by LRBench is very effective. *Third,* seven CLRs recommended by LRBench produce much higher accuracy, ranging from 99.27% to 99.33%, than other LRs. In particular, SIN2 provides the highest accuracy, which is 99.33%, significantly outperforming the baseline fixed LR policies by 0.22%–87.98%.

This set of experiments demonstrates that LRBench can help identify good LR policies and significantly improve the DNN model accuracy. In general, decaying and CLR policies can provide better training results than the baseline fixed LR policies. Similar observations can be found on CIFAR-10, with two different DNN models, CNN3 and ResNet-32.

Table 4. Accuracy Comparison of 13 LR Policies with Training LeNet on MNIST

Category	LR Function	$k(k_0)$	k_1	γ	p	l	Highest Acc (%)
Fixed LR	FIX	0.1					11.35
	FIX	0.01					**99.11**
	FIX	0.001					98.76
	FIX	0.0001					95.10
Decaying LR	NSTEP	0.01		0.9		5,000, 7,000, 8,000, 9,000, 9,500	99.12 ± 0.01
	STEP	0.01		0.85		5,000	99.12
	EXP	0.01		0.99994			99.12
	INV	0.01		0.0001	0.75		99.12
	POLY	0.01			1.2		**99.13**
CLR	TRI	0.01	0.06			2,000	99.28 ± 0.02
	TRI2	0.01	0.06			2,000	99.28 ± 0.01
	TRIEXP	0.01	0.06	0.99994		2,000	99.27 ± 0.01
	SIN	0.01	0.06			2,000	99.31 ± 0.04
	SIN2	0.01	0.06			2,000	**99.33 ± 0.02**
	SINEXP	0.01	0.06	0.99994		2,000	99.28 ± 0.04
	COS	0.01	0.06			2,000	99.32 ± 0.04

Table 5. Accuracy Comparison of 16 LR Policies with Training CNN3 on CIFAR-10

Category	LR Function	$k(k_0)$	k_1	γ	p	l	Accuracy (%)
Fixed LR	FIX	0.1					10.02
	FIX	0.01					69.63
	FIX	0.001					**78.62**
	FIX	0.0001					75.28
Decaying LR	NSTEP	0.001		0.1		60,000, 65,000	**81.61 ± 0.18**
	STEP	0.001		0.85		5,000	79.95
	EXP	0.001		0.99994			79.28
	INV	0.001		0.0001	0.75		78.87
	POLY	0.001			1.2		81.51
CLR	TRI-S	0.001	0.006			2,000	79.39 ± 0.17
	TRI2-S	0.001	0.006			2,000	79.95 ± 0.14
	TRIEXP-S	0.001	0.006	0.99994		2,000	80.16 ± 0.12
	TRI	0.00005	0.006			2,000	81.75 ± 0.13
	TRI2	0.00005	0.006			2,000	80.71 ± 0.14
	TRIEXP	0.00005	0.006	0.99994		2,000	81.92 ± 0.13
	SIN	0.00005	0.006			2,000	81.76 ± 0.12
	SIN2	0.00005	0.006			2,000	80.79 ± 0.14
	SINEXP	0.00005	0.006	0.99994		2,000	**82.16 ± 0.08**
	COS	0.00005	0.006			2,000	81.43 ± 0.14

CIFAR-10 (CNN3). Table 5 shows the empirical results of training CNN3 on CIFAR-10 with the highest Top-1 accuracy within Caffe default total training of 70,000 iterations, and the batch size is the default 100. The three triangle LR policies with S as the suffix are recommended in [36]. We make two interesting observations. *First,* all the decaying and CLR policies identified by LRBench outperform the best baseline fixed LR policy (FIX, k = 0.001) of 78.62% accuracy. For example, SINEXP achieved the highest accuracy and improved the accuracy of this best fixed LR by 3.54% and the best decaying LR, NSTEP, by 0.55%. *Second,* the CLRs recommended by LRBench (the last seven LRs) achieved over 80.71% accuracy, all outperforming the corresponding TRI-S, TRI2-S, and TRIEXP-S in [36] in terms of the highest accuracy within the training time constraint. This

Table 6. Accuracy Comparison of 13 LR Policies with Training ResNet-32 on CIFAR-10

Category	LR Function	$k(k_0)$	k_1	γ	p	l	Accuracy (%)
Fixed LR	FIX	0.1					**86.08**
	FIX	0.01					85.41
	FIX	0.001					82.66
	FIX	0.0001					66.96
Decaying LR	NSTEP	0.1		0.1		32,000, 48,000	**92.38 ± 0.04**
	STEP	0.1		0.99994			91.10
	EXP	0.1		0.85		5,000	91.03
	INV	0.1		0.0001	0.75		88.70
	POLY	0.1			1.2		**92.39**
CLR	TRI	0.0001	0.9			2,000	76.91
	TRI2	0.0001	0.9			2,000	91.85
	TRIEXP	0.0001	0.9	0.99994		2,000	**92.76 ± 0.14**
	SIN	0.0001	0.9			2,000	72.78
	SIN2	0.0001	0.9			2,000	91.98
	SINEXP	0.0001	0.9	0.99994		2,000	**92.81 ± 0.08**
	COS	0.0001	0.9			2,000	76.77

demonstrates that LR selection is critical for effective DNN training. Even for the same LR function, different LR parameters will lead to different training performance.

CIFAR-10 (ResNet-32). ResNet can achieve over 90.00% accuracy on CIFAR-10, a significant improvement over CNN3. Table 6 shows the results of training ResNet-32 on CIFAR-10 with the batch size 128 and the default of 64,000 training iterations. We make two interesting observations. *First,* four LRs recommended by LRBench, i.e., NSTEP (92.38%), POLY (92.39%), TRIEXP (92.76%), and SINEXP (92.81%) achieved over 92.38% accuracy, significantly outperforming the best baseline fixed LR with only 86.08% accuracy by over 6.3%. *Second,* different CLR policies with different CLR functions may share the same LR range and step size parameters, such as TRI, TRI2, SIN, SIN2, and COS; they result in different training performance. For example, TRI2 and SIN2 achieve high accuracy over 91.85% while the other three CLRs fail to achieve the accuracy of 77%. This indicates that selecting a good LR policy involves both the proper LR function and the proper LR parameter setting for effective DNN training.

To verify whether a given default LR policy is good for the given learning task and dataset under the chosen DNN model, LRBench will select the top N policies from the LR policy database instead of performing an exhaustive search. Also, for tuning and verification efficiency, we perform the experiments on N machines in parallel when feasible. Consider the MNIST LeNet classifier with the NSTEP LR, and $N = 5$; LRBench will verify this LR policy by comparing NSTEP with the top five LRs selected in the LR policy database by ranking on the accuracy for all LR policies associated with MNIST, assuming SIN2 is one of the stored LRs in the database. Since SIN2 achieved higher accuracy (99.33%) with LeNet, it will be chosen as one of the top five for verification. The five LRs can be verified in parallel by running the DNN training on MNIST, with verifying each LR policy on one machine, where SIN2 has the highest accuracy. The output of the LR verification will show the performance comparison of NSTEP (99.12% accuracy) with the other five selected LR policies and also recommend SIN2 as the near-optimal replacement LR policy. Our empirical results show that the choice of N is in general 3–5 for an effective LR verification with a reasonable size of LR policy database, e.g., over 100 policies.

5.2 Multiple LR Policy Composition

We then focus on how to compose different LR functions and parameters to form composite LRs. Two types of composite LRs should be considered: one is the homogeneous composite LR

Table 7. Advanced Composite LRs for CIFAR-10 with CNN3

LR Policy		Accuracy (%)
Category	LR Function	
Decaying LR	NSTEP (k = 0.001, γ = 0.1, l = [60,000, 65,000])	81.61 ± 0.18
CLR	SINEXP (k_0 = 0.00005, k_1 = 0.006, γ = 0.99994, l = 2,000)	82.16 ± 0.08
Advanced Composite LR (Homogeneous)	NSTEP-new (k = 0.001, $\eta \in [0.0000008, 0.005]$)	82.28
Advanced Composite LR (Heterogeneous)	0–60,000 iter: TRI (k_0 = 0.001, k_1 = 0.005, l = 2000)	**82.53**
	60,000–65,000 iter: TRI2 (k_0 = 0.0001, k_1 = 0.0005, l = 1000)	
	65,000–70,000 iter: TRI2 (k_0 = 0.00001, k_1 = 0.00005, l = 500)	

consisting of the same LR function with different LR parameters, and the other one is the heterogeneous composite LR that is comprised of different LR functions.

Homogeneous Composite LRs. Recall Table 3 and Figure 2, we have shown several examples of homogeneous composite LRs. NSTEP is a special case of homogeneous composite LRs with n different FIX LRs, defined by n fixed LR values, denoted by $\eta_0, \eta_1, \ldots, \eta_{n-1}$, and n training iteration boundaries $l_0, l_1, \ldots, l_{n-1}$, such that the LR value is η_0 when $i = 0,\ t < l_0$, and η_i when $i > 0$ and $l_{i-1} \le t < l_i$ for $i = 1, \ldots, n-1$. In the next set of experiments, we show the effectiveness of using Change-LR-On-Plateau in LRBench to select, compose, and verify homogeneous composite LRs by training CNN3 on CIFAR-10 as Table 7 and Figure 6 show.

By setting the initial LR value for the composite LR (NSTEP-new) at 0.001, LRBench will automatically scale it within the range of $[0.0000008, 0.005]$. The key difference between this composite LR (NSTEP-new) and the decaying LR (NSTEP) is that our NSTEP-new allows the increase of LR values in the training while NSTEP will only reduce the LR values. Figure 6 shows the detailed changes of the LR value (left y-axis) and the Top-1/Top-5 accuracy (right y-axis). We observe that increasing the LR value at the beginning phase of training is beneficial, confirming that LR values should be dynamically updated to accommodate the proper learning speeds required in different training phases, instead of decaying throughout the entire training iterations. Next, by monitoring the training loss throughout the 70,000 iterations set by the Caffe CNN3 default training time on CIFAR-10, we can identify and create a new NSTEP LR policy with improved Top-1 accuracy, compared to the decaying NSTEP for Caffe CNN3 on CIFAR-10. Also, this homogeneous composite LR (NSTEP-new) with $k = 0.001$ and $\eta \in [0.0000008, 0.005]$ provides slightly improved accuracy of 82.28% over 82.16% of the best single-policy LR SINEXP (computed over five repeated runs). It also improves the best decaying NSTEP accuracy (81.61%) by 0.67%.

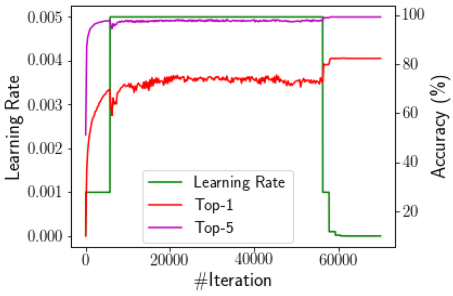
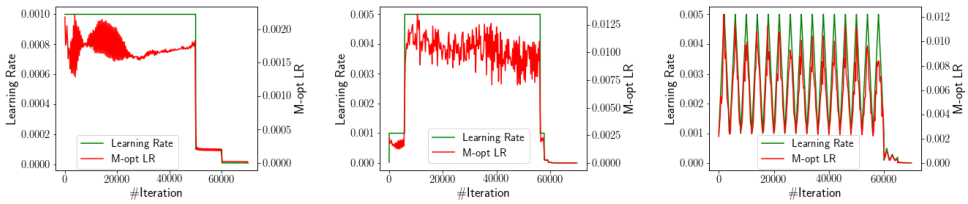


Fig. 6. NSTEP-new LR (CIFAR-10, CNN3).

Heterogeneous Composite LRs. Table 7 shows the effectiveness of a heterogeneous composite LR, which is composed by three cyclic LRs from two different CLR functions, TRI and TRI2. The highest accuracy achieved by this heterogeneous composite LR is 82.53%, improving the best decaying NSTEP accuracy (81.61%) by 0.92%, and further improving the accuracy (82.28%) of the NSTEP-new from LRBench by 0.25%. This shows that composite LRs offer great possibilities to select and compose good LR policies and avoid worse ones.

M-opt LR Verification. Next, we show the other method in LRBench to verify the LR values by using Formula (7) to calculate the M-opt LR values every M iterations. Here, we set M = 250



(a) Decaying (NSTEP) (b) Composite (NSTEP-new) (c) Composite (TRI,TRI2)

Fig. 7. M-opt LR verification for CIFAR-10 with CNN3.

and perform this set of experiments with training CNN3 for CIFAR-10 using the best decaying LR, NSTEP, and the above two composite LRs, the homogeneous composite LR (NSTEP-new) and the heterogeneous composite LR with TRI and TRI2. Figure 7 shows the results. Two interesting observations should be highlighted. *First,* the LR values from the heterogeneous composite LR (TRI, TRI2) matched well with its M-opt LR values, indicating the applied LR values facilitate efficient model parameter updates. This also explains why this heterogeneous composite LR policy can achieve the highest accuracy. However, for the NSTEP and NSTEP-new LRs, the two curves show a certain interval of mismatch, which means that the LR values for this interval are not optimal. *Second,* we also observe the benefits of using LRBench for selecting and composing LR policies. On the one hand, Figure 7(b) further proves that the NSTEP-new dynamically tuned by LRBench is a good LR policy, demonstrating effectiveness of the tuning principles in LRBench. On the other hand, through our M-opt LR verification, practitioners can not only verify whether the chosen LR policy is optimal but also locate the potential improvements.

LR Tuning for SVHN. We further perform experiments using LRBench on another dataset, SVHN. For a new dataset, it still remains a problem as to whether these good LR policies are applicable to this new dataset if it shares similar features with the existing dataset in the LR policy database, such as the content and data format. We choose the **Street View House Numbers** (**SVHN**) dataset [32] as Figure 5(c) shows to study this problem. SVHN consists of 73,257 training images and 26,032 testing images with 10 class labels, from 1 to 10. SVHN share more similar contents with MNIST than CIFAR-10, which is 10 digits (see Figure 5(a)). However, SVHN and CIFAR-10 share the same data format, which is $32 \times 32 \times 3$ size for each image, and they are both colorful images (see Figure 5(b)). The default neural network in Caffe uses the same CNN3 architecture with 70,000 training iterations and the batch size of 100. The default LR policy for SVNH is NSTEP as shown in Table 8. We use the best LR policies that produce the highest accuracy for training LeNet on MNIST (CLR: SIN2) and for training CNN3 on CIFAR-10 (Advanced Composite LR: TRI, TRI2) for this set of experiments on SVHN. We measure the highest Top-1 accuracy in Table 8. Two interesting observations should be highlighted. *First*, the advanced composite LR (best LR for CIFAR-10 with CNN3) achieved the highest accuracy 94.23%, indicating that it is effective to apply the LR policy to a new dataset sharing similar features, such as the same data format and DNN architecture. *Second*, even though MNIST and SVHN share the similar contents, the LR policy (SIN2) producing the highest accuracy on MNIST failed to converge the DNN model on SVHN, and it ended with only 19.65% test accuracy. Therefore, this set of experiments shows that it is applicable to use good LR policies for a new learning task from empirical results, which depends on the specific similarity in the data and/or DNN model between the new dataset and the known one.

For a new dataset, LRBench can help users efficiently select and compose good LR policies. If the LR policy database contains the LR tuning results for an existing dataset (or DNN) that is similar

Table 8. LR Tuning for SVHN with CNN3

LR Policy		Accuracy (%)
Category	LR Function	
Decaying LR	NSTEP (k = 0.001, γ = 0.1, l = [60,000, 65,000])	93.97
Cyclic LR	SIN2 (k_0 = 0.00005, k_1 = 0.006, γ = 0.99994, l = 2,000)	19.65
Advanced Composite LR	0–60,000 iter: TRI (k_0 = 0.001, k_1 = 0.005, l = 2000)	**94.23**
	60,000–65,000 iter: TRI2 (k_0 = 0.0001, k_1 = 0.0005, l = 1000)	
	65,000–70,000 iter: TRI2 (k_0 = 0.00001, k_1 = 0.00005, l = 500)	

Table 9. LR Tuning for ImageNet with ResNet-18

LR Policy		Top-1 (%)	Top-5 (%)
Category	LR Function		
Fixed LR	FIX (k = 0.1)	50.50	76.51
Decaying LR	STEP (k = 0.1, γ = 0.1, l = 30 epoch)	68.40	88.37
Advanced Composite LR	0–5 epoch: FIX (k = 0.1)	**69.05**	**88.76**
	5–35 epoch: SINEXP (k_0 = 0.1, k_1 = 0.6, l = 15 epoch, γ = 0.999987)		
	35–50 epoch: FIX (k = 0.01)		
	50–60 epoch: FIX (k = 0.001)		

to this new dataset (or DNN), LRBench will directly recommend these LR policies for training a DNN model on this new dataset, which avoids the typical tedious trial-and-error manual tuning.

LR Tuning for ImageNet. However, if a new dataset is very different from these existing datasets stored in the LR policy database, LRBench will perform LR tuning by performing the LR value range test and dynamic LR tuning functions to significantly narrow down the LR search space, such as a reduction by over 90% (see Section 4.1), and quickly deliver good DNN training results, which is still more efficient than the time-consuming manual tuning process.

For example, ImageNet is one of the large-scale image datasets with over 1.2 million labeled images and 1,000 classes, which is more complex than the MNIST and CIFAR-10 datasets (see Figure 5(d)). It is challenging to specify proper LR policies for training DNNs on such a large dataset. Table 9 shows the experimental comparison of three LR policies recommended by LRBench for training ResNet-18 on ImageNet for 60 epochs with the default batch size of 256. We highlight three interesting observations. *First*, the baseline fixed LR (k = 0.1) can train this DNN model and produce 50.50% Top-1 and 76.51% Top-5 accuracy. *Second*, the decaying LR STEP recommended by LRBench can significantly improve the Top-1 and Top-5 accuracy from 50.50% to 68.40% by 17.9% for Top-1 accuracy and from 76.51% to 88.37% by 11.86% for Top-5 accuracy, compared to the baseline fixed LR. *Third*, LRBench can further identify a good composite LR through dynamic LR tuning, which combines two LR functions, FIX and SINEXP. This advanced composite LR further improved the Top-1 and Top-5 accuracy of ResNet-18 on ImageNet to 69.05% and 88.76%, respectively.

5.3 Cost-Effective LR Tuning

In some cases, the goal for DNN training is to achieve a target accuracy threshold at a low cost, such as a smaller number of training iterations. One example is the pruning in Neural Network Search by [44]. Hence, we show the minimum #Iterations that LRBench took to obtain the desired target accuracy for MNIST and CIFAR-10 in Table 10. We highlight two interesting observations. *First,* it is very hard to achieve the target accuracy with the baseline fixed LRs. For MNIST, only one fixed LR (FIX, k = 0.01) can achieve the target accuracy of 99.1% with a high cost of the entire 10,000 training iterations. For CIFAR-10, all baseline fixed LRs failed to achieve the target accuracy, which is 80% for CNN3 and 90% for ResNet-32. *Second,* cyclic LRs recommended by LRBench can achieve the desired target accuracy at a lower training cost than other LRs. For MNIST, COS reached the

Table 10. Cost Comparison of LR Policies for Achieving Target Accuracy

Dataset-Model	LR Function	#Iter @ Target Acc	Speedup
MNIST (LeNet, Target Acc = 99.1%)	FIX (k = 0.01)	10,000	1.00×
	NSTEP (k = 0.01, γ = 0.9, l = [5,000, 7,000, 8,000, 9,000]	10,000	1.00×
	POLY (k = 0.01, p = 1.2)	7,000	1.43×
	SIN2 (k_0 = 0.01, k_1 = 0.06, l = 2,000)	**3,500**	2.86×
	COS (k_0 = 0.01, k_1 = 0.06, l = 2,000)	**1,500**	**6.67**×
CIFAR-10 (CNN3, Target Acc = 80%)	NSTEP (k = 0.001, γ = 0.1, l = [60,000, 65,000]	61,000	1.00×
	POLY (k = 0.001, p = 1.2)	59,000	1.03×
	TRIEXP (k_0 = 0.00005, k_1 = 0.006, γ = 0.99994, l = 2,000)	**16,000**	3.81×
	SINEXP (k_0 = 0.00005, k_1 = 0.006, γ = 0.99994, l = 2,000)	**12,000**	**5.08**×
CIFAR-10 (ResNet-32, Target Acc = 90%)	NSTEP (k = 0.1, γ = 0.1, l = [32,000, 48,000])	33,000	1.00×
	POLY (k = 0.1, p = 1.2)	51,000	0.65×
	TRI2 (k_0 = 0.0001, k_1 = 0.9, l = 2,000)	**20,000**	**1.65**×
	SIN2 (k_0 = 0.0001, k_1 = 0.9, l = 2,000)	**20,000**	**1.65**×

target accuracy of 99.10% at 1,500 iterations, significantly reducing the training time cost by 6.67× compared to the baseline fixed LR with 10,000 training iterations. SIN2 achieves the target accuracy at 3,500 iterations, also reducing the cost by 2.86×. For training CNN3 on CIFAR-10, the two CLRs, TRIEXP and SINEXP, achieved the target accuracy of 80.0% at 12,000 and 16,000 iterations, reducing the training time by 3.81× and 5.08×, respectively, compared to the baseline NSTEP of 61,000 iterations. For ResNet-32, it only took TRI2 and SIN2 20,000 iterations to reach the target accuracy of 90% on CIFAR-10. Compared to the best decaying NSTEP of 33,000 training iterations, TRI2 and SIN2 can still significantly reduce the training cost by 1.65×.

Execution Time of LRBench. Overall, LRBench can efficiently identify a good LR policy. If used from scratch, LRBench may take a couple of minutes to a few days to find a good LR policy, depending on the size of the training dataset and the complexity of the DNN backbone algorithm used for training. For a small and medium dataset, such as MNIST or CIFAR-10, it takes LRBench a few minutes or hours to find a good LR policy from scratch. For MNIST, we spend about 5 minutes to obtain a good LR policy for LeNet. For CIFAR-10, both the dataset and the DNN backbone algorithm used for training are more complex, and the CIFAR-10 model training usually takes a much longer time than MNIST. For example, it takes about 12 minutes to train CNN3 on CIFAR10 while it only takes about 1 minute for training LeNet on MNIST. As a result, LRBench took about 1 hour to find a good LR policy to train the CNN3 model on CIFAR-10. Moreover, if we use ResNet-32 for CIFAR-10, given that ResNet-32 takes much longer training time on CIFAR-10 (about 2 hours), LRBench also takes about $K \times 2$ hours to find a good LR policy, where K is the number of candidate policies that LRBench is configured to use. For large datasets, it takes much longer time. Using our experimental server, it takes about 10 days on ImageNet for LRBench to identify a good LR policy, due to both the large dataset and the complex neural network architecture. One way to effectively reduce such time costs is to run LRBench in parallel. Currently, users can run multiple LR policies in LRBench in parallel; each performs one of the K candidate policies. We are working on an auto-parallel version of the LRBench to automate such runtime parallelism optimization.

5.4 Impact of LR Policies on Different Optimizers

The choice of a specific DNN training optimizer is another important hyperparameter in DNN training. Given that these optimizers still require the LR to control the model parameter updates, we perform a set of experiments to study how different optimizers will impact on the choice of LR policies and the overall training performance on accuracy and training time. Tables 11 and 12 show the experimental results with two measurements: (1) the highest accuracy found and (2) the corresponding #Iterations under the default training iterations of 10,000 for MNIST and 70,000 for

Table 11. Impact of Different Optimizers (Momentum and SGD) on Training LeNet on MNIST

LR	$k(k_0)$	k_1	γ	p	l	#Iter@Highest Acc		Highest Accuracy (%)	
						Momentum	SGD	Momentum	SGD
FIX	0.01					10,000	9,500	99.11	98.71
NSTEP	0.01		0.9		5,000, 7,000, 8,000, 9,000, 9,500	10,000	10,000	99.12 ± 0.01	98.63
STEP	0.01		0.85		5,000	10,000	9,500	99.12	98.65
EXP	0.01		0.99994			10,000	9,500	99.12	98.53
INV	0.01		0.0001	0.75		10,000	10,000	99.12	98.58
POLY	0.01			1.2		7,000	8,500	99.13	98.16
TRI	0.01	0.06			2,000	**4,000** (2.5×)	8,000	99.28 ± 0.02	99.02
TRI2	0.01	0.06			2,000	**4,000** (2.5×)	8,000	99.28 ± 0.01	99.03
TRIEXP	0.01	0.06	0.99994		2,000	**4,000** (2.5×)	7,500	99.27 ± 0.01	99.05
SIN	0.01	0.06			2,000	**4,000** (2.5×)	8,000	99.31 ± 0.04	99.03
SIN2	0.01	0.06			2,000	**4,000** (2.5×)	7,500	**99.33 ± 0.02**	98.92
SINEXP	0.01	0.06	0.99994		2,000	**4,000** (2.5×)	7,500	99.28 ± 0.04	99.06
COS	0.01	0.06			2,000	10,000	9,500	99.32 ± 0.04	**99.08**

Table 12. Impact of Different Optimizers (Momentum, SGD, Adam) on Training CNN3 on CIFAR-10

LR	$k(k_0)$	k_1	γ	p	l	#Iter@Highest Acc			Highest Accuracy (%)		
						Momentum	SGD	Adam	Momentum	SGD	Adam
FIX	0.001					45,000	70,000	4,500	78.62	75.26	69.64
NSTEP	0.001		0.1		60,000, 65,000	62,250	65,250	7,500	81.61 ± 0.18	76.27	69.98
STEP	0.001		0.85		5,000	68,000	69,750	15,000	79.95	71.91	69.58
EXP	0.001		0.99994			70,000	65,250	69,750	79.28	67.91	70.45
INV	0.001		0.0001	0.75		66,000	70,000	8,000	78.87	70.72	70.66
POLY	0.001			1.2		70,000	69,500	0	81.51	**80.84**	10.13
TRI	0.00005	0.006			2,000	68,000	68,000	4,000	81.75 ± 0.13	79.55	69.15
TRI2	0.00005	0.006			2,000	70,000	70,000	8,000	80.71 ± 0.14	70.93	70.16
TRIEXP	0.00005	0.006	0.99994		2,000	68,000	67,750	12,000	81.92 ± 0.13	75.34	**70.71**
SIN	0.00005	0.006			2,000	68,000	68,000	24,000	81.76 ± 0.12	80.01	68.01
SIN2	0.00005	0.006			2,000	70,000	68,250	8,000	80.79 ± 0.14	72.16	69.38
SINEXP	0.00005	0.006	0.99994		2,000	52,000	64,750	8,000	**82.16 ± 0.08**	75.84	68.99
COS	0.00005	0.006			2,000	70,000	70,000	250	81.43 ± 0.14	79.87	10.02

CIFAR-10 with the default batch size set as 100. Concretely, Table 11 shows the choice of Momentum optimizer and SGD optimizer on different LR schedules for training LeNet MNIST models and Table 12 shows the choice of Momentum, SGD, and Adam optimizers on different LR schedules for training CNN3 CIFAR-10 models. We make three observations: *First,* the combo of the momentum optimizer and the composite LR function SIN2 offers the best performance of 99.33% accuracy with 2.5× faster in the overall training time (10,000/4,000 = 2.5) on MNIST. *Second,* the combo of the momentum optimizer and the composite LR function of SINEXP achieves the best accuracy of 82.16% with 1.35× faster in overall training time (70,000/52,000 ≈ 1.35). *Third,* the momentum optimizer combined with a composite LR schedule, such as SIN2 and SINEXP, can boost the overall training performance on accuracy and training time, followed by SGD optimizer. Adam optimizer has relatively lower performance no matter which LR schedule is used.

Summary. Through the four sets of experiments, we have shown (i) the selection of the good LR policies and schedules on different datasets and different DNN backbone algorithms to train a DNN model on the same dataset (CIFAR-10); (ii) the composition of multi-policy LR schedules for four different datasets and their corresponding learning tasks with their respective DNN training algorithms; and (iii) the cost-effective LR tuning with a targeted accuracy.

LRBench can be utilized for performing semi-automated or automated LR tuning in several ways: (i) If the user is providing a set of LR functions and/or their LR parameters, the LRBench can be used to verify whether and which of the LR policies from this given set are good or bad. (ii) A user can turn on the dynamic LR tuning option in LRBench. This will enable LRBench to set up the LR auto-tuning module, which monitors the training progress over a sample of the training dataset in every M iterations ($1 \leq M \leq T/2$) during the total T training iterations, and dynamically tune the LRs by switching between the chosen candidate LR functions and the given range of each LR parameter based on the specific training progress conditions: for example, LRBench may switch to a larger LR to accelerate the training progress when the training stagnates on the plateau in the middle of training and switch to a smaller LR to facilitate the convergence of the model training when the training is close to the end of total training iterations. (iii) When the dynamic tuning is disabled, LRBench will verify the given candidate LR policies one by one and select the best performing LR schedule based on the accuracy and training time. (iv) For a new dataset and learning task, if the user does not have any specific LR policy as the initial candidate, LRBench can help such users to determine the initial LR functions and the specific LR parameters to use by leveraging the LR policy database, the LR value range test, the dataset similarity evaluation, and the M-opt LR auto-tuning method. One of our ongoing efforts is to introduce similarity evaluation methods for clustering DNN backbone algorithms into different clusters such that the DNN models with similar impact on the choice of LR schedules will be grouped together.

6 RELATED WORK

The LR is widely recognized as an important hyperparameter for training DNNs and it is critical for achieving high test accuracy [30, 36, 42, 45]; however, there are few studies to date dedicated to this topic. To the best of our knowledge, our work is the first to present a systematic study on the selection and composition of good LR policies. We also design and implement an LR recommendation tool, LRBench, to help practitioners to systematically select, compose, and verify good LR policies that best fit their DNN training goals. Besides different LR functions that are introduced in Section 3, we summarize the related work in the following four respects.

DNN Hyperparameter Search. The hyperparameter search for finding the optimal settings for different hyperparameters for DNN training is one of the relevant research themes. Existing general-purpose hyperparameter search frameworks include Ray Tune [28], Hyperopt [7], SMAC [20], and Optuna [1]. They use general hyperparameter search algorithms, e.g., grid search, random search, and Bayesian optimization, and provide limited support for LR tuning, such as only supporting the fixed LR. LRBench can be used in conjunction with these hyperparameter tuning frameworks. For example, LRBench incorporates the support for Ray Tune, which allows LRBench to leverage existing hyperparameter tuning algorithms, such as grid search, random search, and distributed parallel tuning functions, to further improve LR tuning efficiency.

Hypergradient Descent Based LR Adaptation. The hypergradient descent based methods model the LR as a trainable variable and apply gradient descent to optimize an objective function w.r.t. this LR variable to perform automatic updates on the LR value during DNN training [2, 3, 13, 15]. However, these methods introduce an additional hyperparameter, the hypergradient LR, which still requires careful tuning.

Model-Update-Aware LR Adaptation. Several orthogonal yet complementary studies focus on improving the DNN training optimizers to adaptively adjust the LR values for each model parameter based on the accumulated model updates, such as Adagrad [14], Adam [24], AdaDelta [43], and ADASECANT [16]. However, these DNN optimizers still require an LR value to control the model parameter updates (see Section 2). Our experiments in Section 5.4 demonstrate that for these

optimizers, such as Adam, different LRs still have high impacts on the DNN training efficiency and model accuracy.

Automated Machine Learning (AutoML). AutoML aims to minimize the dependence of manual labor and human assistance in developing high performance deep learning systems [18, 19, 27, 34, 39]. In addition to hyperparameter optimization [18, 27], recent AutoML efforts include neural architecture search [44] and AutoML for model compression [19].

7 CONCLUSION

We have presented a systematic approach for selecting and composing good LR policies for DNN training with three major contributions. *First,* we build a systematic LR tuning mechanism to dynamically tune LR values and verify LR policies with respect to the target accuracy and/or training time constraints. *Second,* we develop LRBench, an LR policy recommendation system, to facilitate efficient selection and composition of LR policies by leveraging different LR functions and parameters and to avoid bad ones for a given learning task, dataset, and DNN model. Our experiments show that both homogeneous and heterogeneous LR compositions are attractive and advantageous for DNN training. *Third,* we show that the good LR policies, including the composite LR policies, are applicable to a new dataset as well as the significant mutual impact of LR policies on different optimizers through extensive experiments.

ACKNOWLEDGMENTS

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*. ACM, New York, NY, 2623–2631.

[2] Luís B. Almeida, Thibault Langlois, José D. Amaral, and Alexander Plakhov. 1999. *Parameter Adaptation in Stochastic Optimization*. Cambridge University Press, 111–134.

[3] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. 2018. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkrsAzWAb.

[4] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. 2017. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 459–468.

[5] Yoshua Bengio. 2012. *Practical Recommendations for Gradient-Based Training of Deep Architectures*. Springer, Berlin, 437–478.

[6] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 10 (2012), 281–305. http://jmlr.org/papers/v13/bergstra12a.html.

[7] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, 115–123. https://proceedings.mlr.press/v28/bergstra13.html.

[8] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*. Springer, 421–436.

[9] Thomas M. Breuel. 2015. The effects of hyperparameters on SGD training of neural networks. arXiv:1508.02788 [cs.NE].

[10] Pedro Carvalho, Nuno Lourenço, Filipe Assunção, and Penousal Machado. 2020. AutoLR: An evolutionary approach to learning rate policies. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO'20)*. Association for Computing Machinery, New York, NY, 672–680. https://doi.org/10.1145/3377930.3390158

[11] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. In *NIPS ML Systems Workshop*.

[12] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv:1406.2572 [cs.LG].
[13] Michele Donini, Luca Franceschi, Orchid Majumder, Massimiliano Pontil, and Paolo Frasconi. 2021. MARTHE: Scheduling the learning rate via online hypergradients. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*. Article 293, 7 pages.
[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (July 2011), 2121–2159. http://dl.acm.org/citation.cfm?id=1953048.2021068.
[15] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. 2017. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1165–1173. https://proceedings.mlr.press/v70/franceschi17a.html.
[16] Caglar Gulcehre, Marcin Moczulski, and Yoshua Bengio. 2014. ADASECANT: Robust adaptive secant method for stochastic gradient. arXiv:1412.7419 [cs.LG].
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.
[18] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622. https://doi.org/10.1016/j.knosys.2020.106622
[19] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 815–832.
[20] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, Carlos A. Coello Coello (Ed.). Springer, Berlin, 507–523.
[21] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia (MM'14)*. ACM, New York, NY, 675–678.
[22] Yuchen Jin, Tianyi Zhou, Liangyu Zhao, Yibo Zhu, Chuanxiong Guo, Marco Canini, and Arvind Krishnamurthy. 2021. AutoLRS: Automatic learning-rate schedule by Bayesian optimization on the fly. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SlrqM9_lyju.
[23] Keras Developers. 2019. "Callbacks - Keras Documentation". Retrieved August 23, 2019 from https://keras.io/callbacks/.
[24] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980. http://arxiv.org/abs/1412.6980.
[25] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).
[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (Nov. 1998), 2278–2324. https://doi.org/10.1109/5.726791
[27] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020.
[28] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. 2018. Tune: A research platform for distributed model selection and training. arXiv:1807.05118.
[29] Ling Liu, Yanzhao Wu, Wenqi Wei, Wenqi Cao, Semih Sahin, and Qi Zhang. 2018. Benchmarking deep learning frameworks: Design considerations, metrics and beyond. In *Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS'18)*. 1258–1269. https://doi.org/10.1109/ICDCS.2018.00125
[30] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic gradient descent with warm restarts. arXiv:1608.03983 [cs.LG].
[31] James Martens. 2010. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, 735–742.
[32] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.
[33] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks* 12, 1 (1999), 145–151.
[34] Esteban Real, Chen Liang, David So, and Quoc Le. 2020. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 8007–8019. https://proceedings.mlr.press/v119/real20a.html.

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[36] Leslie N. Smith. 2015. Cyclical learning rates for training neural networks. arXiv:1506.01186 [cs.CV].

[37] Leslie N. Smith and Nicholay Topin. 2017. Super-convergence: Very fast training of neural networks using large learning rates. arXiv:1708.07120 [cs.LG].

[38] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*. PMLR, Atlanta, Georgia, 1139–1147.

[39] Anh Truong, Austin Walters, Jeremy Goodsitt, Keegan Hines, C. Bayan Bruss, and Reza Farivar. 2019. Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools. In *Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI'19)*. 1471–1479. https://doi.org/10.1109/ICTAI.2019.00209

[40] Yanzhao Wu, Wenqi Cao, Semih Sahin, and Ling Liu. 2018. Experimental characterizations and analysis of deep learning frameworks. In *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data'18)*. 372–377. https://doi.org/10.1109/BigData.2018.8621930

[41] Yanzhao Wu, Ling Liu, Juhyun Bae, Ka-Ho Chow, Arun Iyengar, Calton Pu, Wenqi Wei, Lei Yu, and Qi Zhang. 2019. Demystifying learning rate policies for high accuracy training of deep neural networks. In *Proceedings of the 2019 IEEE International Conference on Big Data (Big Data'19)*. 1971–1980. https://doi.org/10.1109/BigData47090.2019.9006104

[42] Yanzhao Wu, Ling Liu, Calton Pu, Wenqi Cao, Semih Sahin, Wenqi Wei, and Qi Zhang. 2022. A comparative measurement study of deep learning as a service framework. *IEEE Transactions on Services Computing* 15, 1 (2022), 551–566. https://doi.org/10.1109/TSC.2019.2928551

[43] Matthew D. Zeiler. 2012. ADADELTA: An adaptive learning rate method. arXiv:1212.5701 [cs.LG].

[44] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *CoRR* abs/1611.01578 (2016). arXiv:1611.01578.

[45] Hafidz Zulkifli. 2018. Understanding Learning Rates and How It Improves Performance in Deep Learning. Retrieved September 23, 2018 from https://towardsdatascience.com/.

Received 14 December 2020; revised 17 August 2022; accepted 12 October 2022