Learning Rate Dropout

Huangxing Lin, Weihong Zeng, Yihong Zhuang, Xinghao Ding, *Member, IEEE*, Yue Huang, *Member, IEEE*, and John Paisley, *Member, IEEE*

***Abstract*— Optimization algorithms are of great importance to efficiently and effectively train a deep neural network. However, the existing optimization algorithms show unsatisfactory convergence behavior, either slowly converging or not seeking to avoid bad local optima. Learning rate dropout (LRD) is a new gradient descent technique to motivate faster convergence and better generalization. LRD aids the optimizer to actively explore in the parameter space by randomly dropping some learning rates (to 0); at each iteration, only parameters whose learning rate is not 0 are updated. Since LRD reduces the number of parameters to be updated for each iteration, the convergence becomes easier. For parameters that are not updated, their gradients are accumulated (e.g., momentum) by the optimizer for the next update. Accumulating multiple gradients at fixed parameter positions gives the optimizer more energy to escape from the saddle point and bad local optima. Experiments show that LRD is surprisingly effective in accelerating training while preventing overfitting.**

***Index Terms*— Neural network, optimization algorithm, regularization.**

I. Introduction

THE principle of training a deep neural network is to minimize a high-dimensional nonconvex loss function. This is performed by gradient-descent-based algorithms, which update the network parameters along the opposite direction of their gradients. The existing optimization algorithms exhibit differentiated convergence behavior when searching for the optimal point in the high-dimensional parameter space due to different update paradigms. Specifically, nonadaptive algorithms (e.g., stochastic gradient descent momentum (SGDM) [1]) converge slowly, but tend to reach good local optima, where the generalization of the trained network is decent. In contrast, adaptive algorithms (Adam [2], Amsgrad [3], RMSprop [4], RAdam [5], *etc*) apply adaptive learning rates to accelerate training, but usually converge to bad local optima. Given the imperfections of these optimization algorithms, how to achieve rapid convergence while maintaining good generalization has become a focus topic.

To accelerate the training process, the most widely used technique is batch normalization (BN) [6]. BN normalizes the features along the sample dimension in a mini-batch for training, and its efficiency is related to a proper batch size. Another popular technique is momentum, which uses gradient statistics to stabilize training. Beyond these sophisticated methods, the most effective and simplest trick for accelerating training is to reduce the parameters to be updated. As is known, small networks are easier to converge than big networks. The rationale behind is that a small network has fewer parameters to be updated, so it is easier for the optimizer to find a local optimum in the parameter space. This inspires us to freeze some parameters of a big network and update only the remaining parameters to speed up convergence (as shown in Fig. 1).

Recently, some regularization techniques (e.g., dropout [7], weight decay [8], noisy label [9]) have achieved great success in improving model generalization. The most impressive one is dropout, which can prevent feature coadaptation (a sign of overfitting) effectively by randomly dropping the hidden units (i.e., their activation is zeroed). Dropout can be interpreted as a way to apply Bernoulli noise to the hidden units. Other methods can achieve similar regularization effects by injecting noise into gradients [10], label [9], and activation functions [11]. The effectiveness of these regularization techniques stems from their perturbation to the training process. These unexpected perturbations give the optimizer more opportunities to escape from bad local optima and find better results. However, many researchers have found that these regularization methods can improve generalization at the cost of training time [12]. This is because the random perturbation at each training step hinders the network from learning and memorizing the training data. Therefore, it is hard to achieve both rapid convergence and good generalization.

Based on the above observations, we propose a new optimization technique, learning rate dropout (LRD), to help the optimizer accelerate the training process and improve the model generalization capability. The key difference between LRD and standard dropout [7] is that LRD randomly drops the learning rate of model parameters rather than dropping hidden units. This means that LRD does not interfere with the forward propagation of the network, but only performs in the process of gradient back propagation. During training,

Manuscript received December 29, 2020; revised October 28, 2021 and January 20, 2022; accepted February 24, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 82172033, Grant U19B2031, Grant 61971369, and Grant 52105126; and in part by the Science and Technology Key Project of Fujian Province under Grant 2019HZ020009. *(Corresponding author: Xinghao Ding.)*

Huangxing Lin is with the School of Informatics and the National Institute for Data Science in Health and Medicine, Xiamen University, Xiamen 361005, China.

Weihong Zeng and Yihong Zhuang are with the School of Informatics, Xiamen University, Xiamen 361005, China.

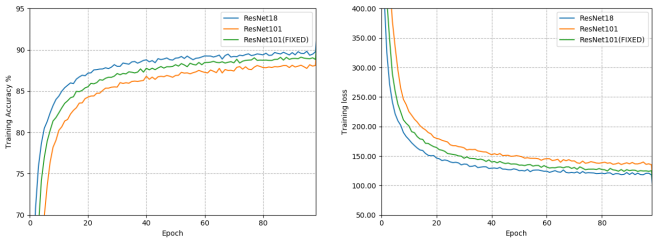
Xinghao Ding is with the School of Informatics, the National Institute for Data Science in Health and Medicine, and the Institute of Artificial Intelligence, Xiamen University, Xiamen 361005, China (e-mail: dxh@xmu.edu.cn).

Yue Huang is with the School of Informatics and the Institute of Artificial Intelligence, Xiamen University, Xiamen 361005, China.

John Paisley is with the Department of Electrical Engineering and the Data Science Institute, Columbia University, New York, NY 10027 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3155181.

Digital Object Identifier 10.1109/TNNLS.2022.3155181



(a) (b)

Fig. 1. Learning curves for different networks (ResNet18, ResNet101) on CIFAR-10. ResNet101 (FIXED) means that half of the parameters of ResNet101 are fixed to their initial values, while the remaining half is updated by the optimizer. Reducing the amount of parameters to be updated significantly improves the convergence speed. (a) Training accuracy. (b) Loss curves.

the optimizer dynamically calculates the gradient and assigns a learning rate to each updated parameter. LRD works by randomly determining which parameters are not updated (i.e., the learning rate is set to zero) for the current training iteration. To do this, the learning rate for each parameter is retained with a fixed probability p, independently of other parameters, or is dropped with probability $1 - p$. A dropped learning rate is temporarily set to zero, which does not affect the update of other parameters or subsequent updates.

At each training step, LRD allows a random subset of network parameters to be updated, while the rest are fixed. Since the parameters to be updated are reduced, the optimizer finds the local optimum in the parameter space more easily. Therefore, convergence is accelerated [13]. For parameters whose learning rate is dropped, the update is paused (parameter values are temporarily fixed), but their gradients are stored by the gradient accumulation component (e.g., momentum) of the optimizer. This means that LRD enables the optimizer to perform multiple gradient estimation and accumulation on the parameters (whose values are fixed). The accumulated gradient gives the optimizer more energy to escape from the bad local optima. Extensive experiments demonstrate the effectiveness of LRD. LRD can be trivially embedded into the existing optimization algorithms and consistently show improvements in both training speed and generalization.

II. Related Work

Various optimization algorithms and techniques are devoted to improving neural network training. The following is a brief review of some related work.

A. Optimization Algorithms

The task of training a neural network is an optimization problem which is commonly solved by gradient-descent-based algorithms. SGD [14] is a widely used approach, which performs well in many research fields. However, it has been observed that SGD has slow convergence since it scales the gradient uniformly in all the directions. To address this issue, variants of SGD adaptively rescale the gradient to achieve fast convergence. Examples include RMSprop [4], Adadelta [15], and Adam [2]. In particular, Adam is the most popular adaptive optimization algorithm due to its rapid training speed. However, Adam [16] tends to converge to the bad local optimum, resulting in disappointing generalization. Recently, Amsgrad [3] and Adabound [17], two variants of Adam, were proposed to solve the convergence issues of Adam by bounding the learning rates. Furthermore, RAdam [5] also aims to further improve Adam by rectifying the variance of the adaptive learning rate. While these adaptive methods [18] often display faster progress in training, they have also been observed to fail to converge well in many cases.

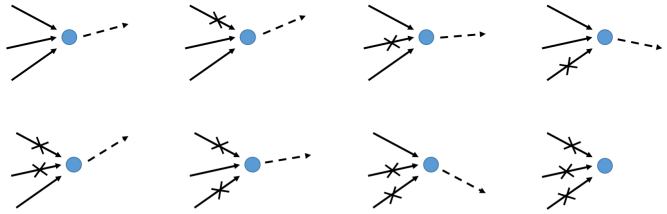


Fig. 2. Applying LRD to training. This model contains three parameters, so there are $2^3 = 8$ parameter subsets to choose in each iteration. Different subsets lead to different update directions. The blue dot is the model state. The solid line is the gradient of each parameter. The dashed line is the resulting update for the model. "$\times$" represents dropping the learning rate.

B. Optimization Techniques

Many techniques enhance model generalization by introducing noise into different parts of training (e.g., input [19], weight [20], activation [21], label [9], and gradient [10]). Kirkpatrick *et al.* [22] demonstrate that noise can help stabilize the state of a complex system using small random perturbations to change it. Bishop [23] proves that applying additive noise to the input is equivalent to a regularization term in a loss function if the noise amplitude is sufficiently small. Gulcehre *et al.* [11] show that noisy activation functions (e.g., softmax [21] and tanh) are easier to optimize. Replacing the nonlinearities by their noisy counterparts usually leads to better results. Neelakantan *et al.* [10] add noise to gradients to avoid overfitting and result in lower training loss. Xie *et al.* [9] imposes regularization within the loss layer by randomly setting the labels to be incorrect. Similarly, the standard dropout [7] is a way of regularizing a neural network by adding Bernoulli noise to its hidden units. Wan *et al.* [24] further propose DropConnect, which is a generalization of dropout. DropConnect sets a randomly selected subset of weights to zero, rather than hidden units. The random perturbation caused by these regularization techniques makes the trained model not easily trapped in bad local optima, but slows down the convergence [25]. Slow convergence is annoying, especially for training deep neural networks. A fast and efficient training is what researchers desire. However, compromise often occurs between training speed and model generalization. The LRD in this article can effectively alleviate these problems, which significantly contributes to the training speed and model generalization.

III. Methodology

We start our discussion on LRD by integrating it into an online optimization problem [26].

Algorithm 1 Generic Framework of Optimization With LRD. $\circ$ Indicates Elementwise Multiplication

Require: α : learning rate, $\{\phi_t, \psi_t\}_{t=1}^{T}$: functions to calculate momentum and adaptive rate, W_0 : initial parameters, $f(W)$: stochastic objective function, p : dropout rate, and $AdaptiveMethod$: False or True.

Ensure: W_T : resulting parameters.

for $t = 1$ **to** T **do**
  $G_t = \nabla f_t(W_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t).
  $M_t = \phi_t(G_1, \cdots, G_t)$ (Accumulation of past and current gradients).
  if $AdaptiveMethod$ is ***True*** **then**
    $V_t = \psi_t(G_1, \cdots, G_t)$ (Accumulation of squared gradients).
    $\triangle W_t = M_t / \sqrt{V_t}$.
  else
    $\triangle W_t = M_t$.
  end if
  Randomly sample learning rate dropout mask D_t with each element $d_t^{(i)} \sim Bernoulli(p)$.
  $A_t = \alpha D_t$ (Randomly drop learning rates at timestep t).
  $W_t = W_{t-1} - A_t \circ \triangle W_t$.
end for

A. Online Optimization

A generic framework of optimization methods with LRD is presented in Algorithm 1 (all multiplications are elementwise). Consider a neural network with weight parameters W. Assume $W = (w^{(1)}, \ldots, w^{(n)})$, where n is the number of parameters. In the online setup, at each time step t, a mini-batch of data passes through the network. A loss function f_t is then revealed, and the model incurs loss $f_t(W_{t-1})$. The optimization algorithm modifies the current parameters W_{t-1} using the gradient $G_t = \nabla f_t(W_{t-1})$ and possibly other terms including earlier gradients.

Algorithm 1 encapsulates many popular adaptive and nonadaptive methods by the definition of gradient accumulation terms $\phi(\cdot)$ and $\psi(\cdot)$. In particular, the adaptive methods are distinguished by the choice of $\psi(\cdot)$, which is absent in nonadaptive methods. Most methods contain the similar momentum component

$$\begin{aligned} M_t &= \phi_t(G_1, \ldots, G_t) \\ &= \beta M_{t-1} + \eta G_t \end{aligned} \tag{1}$$

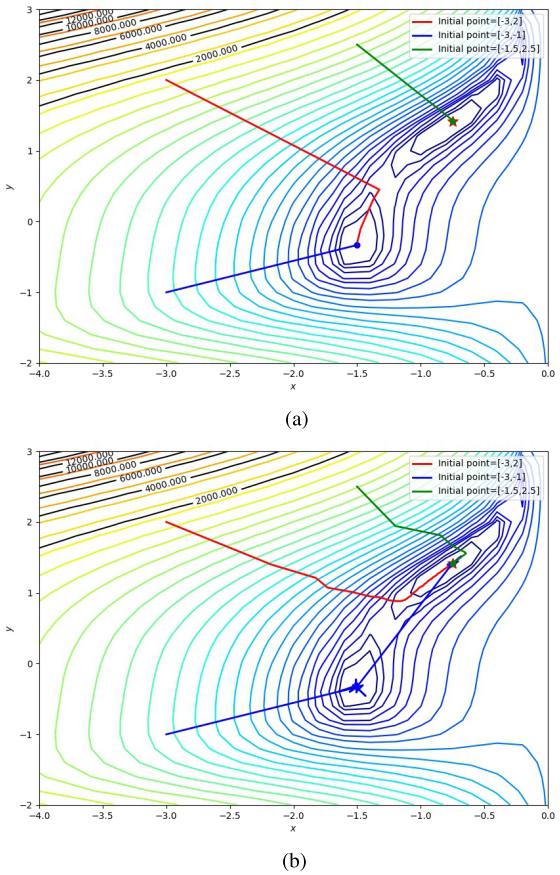
where $\beta > 0$, $\eta > 0$, and β are the momentum factors. The momentum accumulates the exponentially moving average (EMAvg) of previous gradients to correct the current update.

B. Learning Rate Dropout

During training, the optimizer assigns a learning rate α, which is typically a constant, to each parameter. Embedding LRD into the optimization process, the learning rate of each parameter at each training step is kept with probability p, independently of other parameters, or set to zero otherwise. At each time step t, a random binary mask D_t (D_t has the same dimension as W) is sampled to encode the learning rate information with each element $d_t^{(i)} \sim \text{Bernoulli}(p)$. Then the masked learning rate A_t at time step t is obtained by

$$A_t = \alpha D_t. \tag{2}$$

For a model parameter $w^{(i)}$, a learning rate of 0 means not updating its value. For a model, applying LRD is equivalent to randomly sampling one from 2^n parameter subsets and updating it at the current training step (see Fig. 2).



(a)

(b)

Fig. 3. Visualization of the loss descent paths. The LRD encourages Adam to actively explore various parameter update directions. More importantly, LRD helps the model traverse quickly through the "transient" plateau (e.g., saddle points or local minima) and gives the model more chances to find a better and flatter minimum. "$\star$" is the optimal point $(-0.74, 1.40)$. (a) Adam. (b) Adam with LRD.

C. Gradient Accumulation

In general training (without LRD), each model parameter is updated at each training step. Therefore, for a parameter $w^{(i)}$, the time interval λ between its two adjacent updates is 1. With LRD, each parameter is updated intermittently because the learning rate may be dropped. Then, the update interval of the parameter $w^{(i)}$ is $\lambda \geq 1$, and its expectation is the reciprocal of the dropout rate p

$$E(\lambda) = \frac{1}{p}. \tag{3}$$

TABLE I

TEST ACCURACY (MEAN AND STD COMPUTED OVER FIVE TRIALS) ON IMAGE CLASSIFICATION (%)

Dataset	Model	Optimization algorithms (without / with learning rate dropout)				
		SGDM / LRD	RMSprop / LRD	Adam / LRD	AMSGrad / LRD	RAdam / LRD
MNIST	FCNet	97.75±0.32 / **98.46±0.28**	97.92±0.29 / **98.40±0.25**	98.06±0.35 / **98.23±0.38**	98.09±0.20 / **98.50±0.31**	97.97±0.29 / **98.31±0.26**
CIFAR-10	ResNet-34	95.25±0.16 / **95.54±0.20**	92.71±0.35 / **93.68±0.29**	93.05±0.27 / **93.77±0.26**	93.31±0.19 / **93.73±0.28**	94.24±0.21 / **94.61±0.23**
CIFAR-100	DenseNet-121	79.09±0.14 / **79.42±0.18**	70.21±0.43 / **74.02±0.37**	72.55±0.36 / **74.34±0.41**	73.91±0.26 / **75.23±0.22**	71.81±0.30 / **73.04±0.25**

Note that LRD disturbs the update of parameters, but all gradient information is stored by gradient accumulation technology (e.g., momentum). Generally, momentum only stores the gradient once for the parameter $w^{(i)}$ at its value $w_{t-1}^{(i)}$ [i.e., $\nabla f_t(w_{t-1}^{(i)})$]

$$m_t^{(i)} = \beta m_{t-1}^{(i)} + \eta \nabla f_t\left(w_{t-1}^{(i)}\right). \tag{4}$$

Then, $w^{(i)}$ is updated by

$$w_t^{(i)} = w_{t-1}^{(i)} - \alpha m_t^{(i)}. \tag{5}$$

However, LRD changes this routine. LRD makes the parameter update interval λ greater than or equal to 1, which means that the optimizer calculates the gradient for $w_{t-1}^{(i)}$ λ times until $w_{t-1}^{(i)}$ is updated (i.e., the learning rate is not dropped)

$$w_{t-1+\varepsilon}^{(i)} = \begin{cases} w_{t-1}^{(i)}, & 1 \le \varepsilon < \lambda \\ w_{t-1}^{(i)} - \alpha m_{t-1+\lambda}^{(i)}, & \varepsilon = \lambda. \end{cases} \tag{6}$$

According to (4) and (6), $m_{t-1+\lambda}^{(i)}$ can be expressed as

$$\begin{aligned} m_{t-1+\lambda}^{(i)} &= \beta m_{t-2+\lambda}^{(i)} + \eta \nabla f_{t-1+\lambda}\left(w_{t-2+\lambda}^{(i)}\right) \\ &= \beta m_{t-2+\lambda}^{(i)} + \eta \nabla f_{t-1+\lambda}(w_{t-1}^{(i)}). \end{aligned} \tag{7}$$

Further expanding (7), then

$$m_{t-1+\lambda}^{(i)} = \beta^{\lambda} m_{t-1}^{(i)} + \eta \left[\sum_{\epsilon=1}^{\lambda} \beta^{\lambda-\epsilon} \nabla f_{t-1+\epsilon}(w_{t-1}^{(i)}) \right]. \tag{8}$$

In (8), all the gradients for $w_{t-1}^{(i)}$ are stored by momentum. This multiple gradient estimation helps correct the update of the parameters. Moreover, accumulating multiple gradients at the same parameter position (i.e., $w_{t-1}^{(i)}$) gives the optimizer more energy to escape from the local optimum. Specifically, accumulating gradients may significantly amplify the gradients of some parameters. When updating the parameters, the amplified gradient forces the optimizer to deviate from the poor local optimum and search for other flatter optima. In the following toy example, we will more intuitively demonstrate the effect of gradient accumulation.

D. Toy Example

LRD randomly samples some parameters at each training step to update. Due to the limitation of the parameter space, the speed of the optimizer to search local optimum is significantly improved. More importantly, LRD enables optimizers with gradient accumulation components to actively explore in the parameter space, thus preventing overfitting.

To better understand how LRD helps training, we show a toy example to visualize the loss descent path during optimization. Consider the following nonconvex function:

$$\begin{aligned} f(x, y) = (1.5 - x^2 + xy)^2 + (2.25 - x^2 + xy^2)^2 \\ + (2.625 - x^2 + xy^3)^2 \end{aligned} \tag{9}$$

where $x \in [-4, 0]$, $y \in [-2.0, 3.0]$. The optimizer is Adam, which searches for the minima of this function in the 2-D parameter space. For this function, the point $(-0.74, 1.40)$ is the optimal solution. As shown in Fig. 3, the convergence of Adam is sensitive to the initial point. Different initializations lead to different convergence results. Adam is easily trapped by the minimum near the initial point, even if it is a bad local minimum. In contrast, LRD makes Adam more active and exploratory. During training, the optimizer explores various optimization directions. Even if the optimizer reaches a local minimum, it tends to search for other better and flatter minima due to gradient accumulation and LRD. This example illustrates that the LRD can effectively help the optimizer escape from suboptimal points and find a better result.

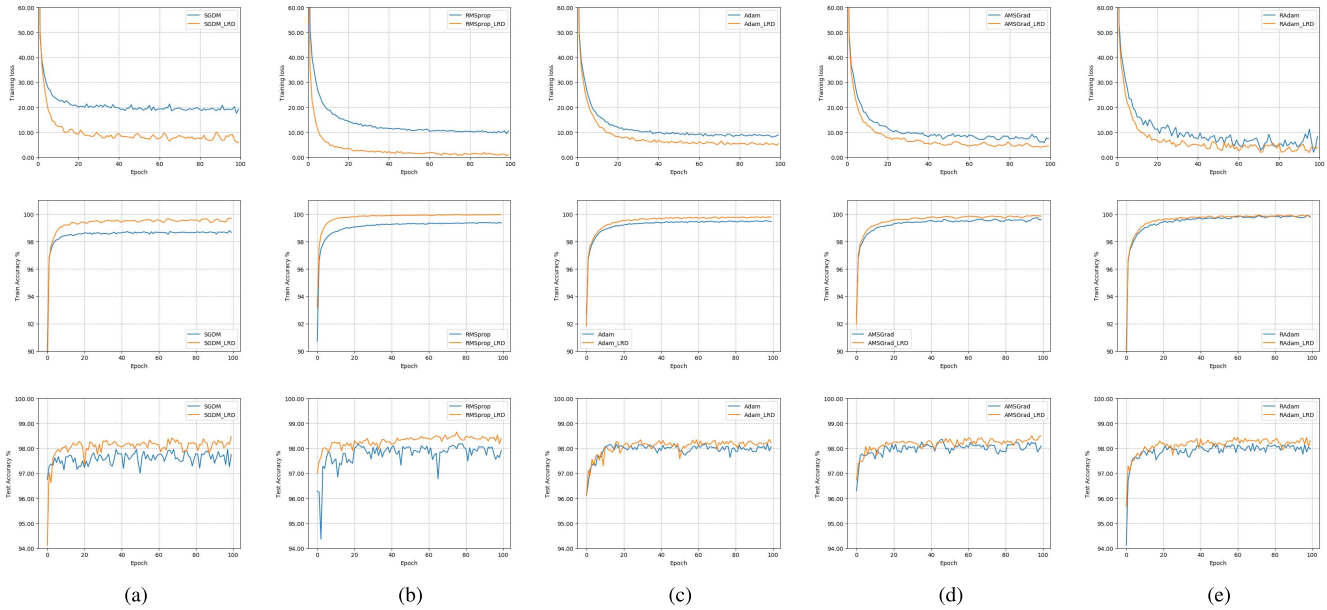
IV. EXPERIMENTS

In this section, extensive experiments are performed to validate the effectiveness of LRD. The experiments involve multiple computer vision tasks, including image classification, segmentation and detection. To further demonstrate the generality of LRD, different models and optimization algorithms are used in the experiments.

A. Image Classification

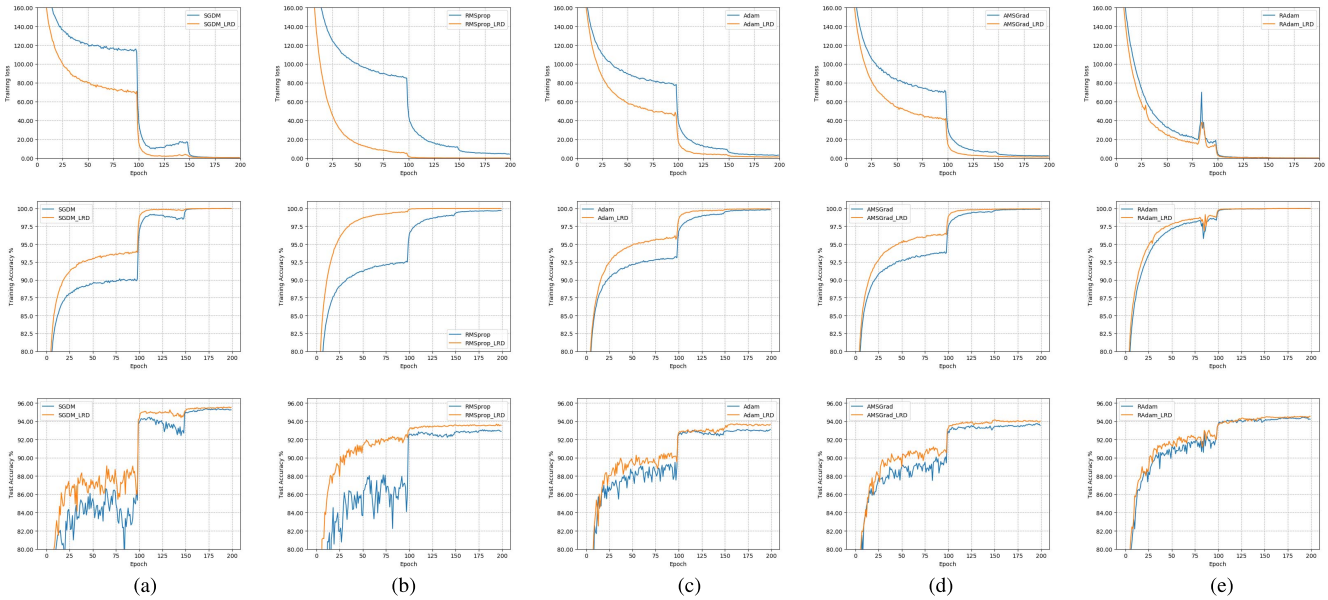
LRD is first applied to multiclass classification problems. The datasets are MNIST, CIFAR-10/100, tiny ImageNet, and ImageNet. To make a comprehensive evaluation, various optimizers are used for training, including SGDM [1], RMSprop [4], Adam [2], AMSGrad [3], and RAdam [5]. The hyperparameters of each optimizer keep their default settings. For example, the initial learning rates for SGDM, RMSprop, Adam, AMSGrad, and RAdam are 0.1, 0.001, 0.001, 0.001, and 0.03, respectively. For optimizers with LRD, if not specified, the dropout rate p (the probability of performing an individual parameter update in any given iteration) is set to 0.5. Each experiment is repeated five times, and the results are reported in the format of mean ± standard deviation (std). All the experiments are performed on a server with two NVIDIA RTX6000 GPUs.

1) MNIST: The MNIST digits dataset [27] contains 60 000 training and 10 000 test images of size 28×28. The task is to classify the images into ten digit classes. Following the previous publications [28], a fully connected neural network (FCNet) with two hidden layers is adopted



(a) (b) (c) (d) (e)

Fig. 4. Learning curves for a fully connected network on MNIST. Top: training loss. Middle: training accuracy. Bottom: test accuracy. (a) SGDM. (b) RMSprop. (c) Adam. (d) AMSGrad. (e) RAdam.



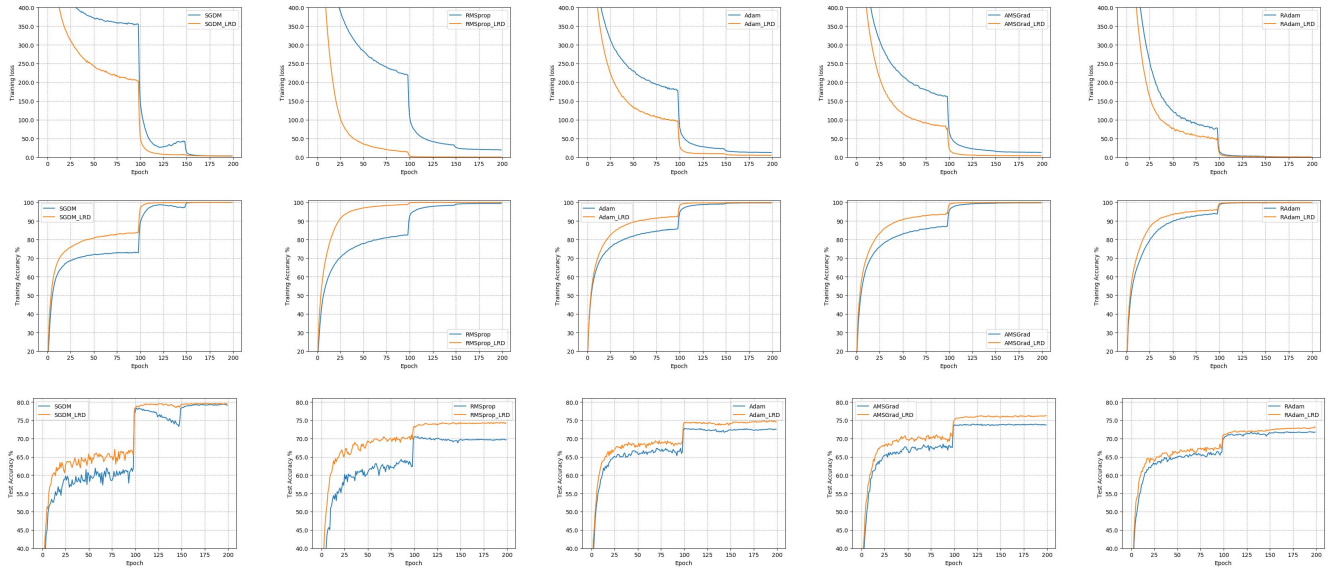
(a) (b) (c) (d) (e)

Fig. 5. Learning curves for ResNet-34 on CIFAR-10. Top: training loss. Middle: training accuracy. Bottom: test accuracy. (a) SGDM. (b) RMSprop. (c) Adam. (d) AMSGrad. (e) RAdam.

to handle this digit classification problem. Each hidden layer contains 1000 neurons and is followed by the rectified linear unit (ReLU) [29] activation function. The training is on mini-batches with 128 images per batch for 100 epochs through the training set. A decay scheme is not used. The learning curves are shown in Fig. 4, and the test accuracies are reported in Table I.

2) CIFAR: The CIFAR-10 and CIFAR-100 datasets consist of 60 000 RGB images of size 32×32, drawn from 10 and 100 categories, respectively. About 50 000 images are used for training and the rest for testing. In both the datasets, training and testing images are uniformly distributed over all the categories. To show the broad applicability of the proposed method, we use ResNet-34 [30] for CIFAR-10 and DenseNet-121 [31] for CIFAR-100. Following the parameter setting in [17], both the models are trained for 200 epochs with a mini-batch size 128. The learning rate is reduced by ten times at the 100th and 150th epochs. The weight decay rate is set to $5e-4$. The results are reported in Figs. 5 and 6 and Table I.

3) Tiny ImageNet: The Tiny ImageNet [32] dataset is a subset of the ILSVRC2014 dataset with 200 categories. Each



(a) (b) (c) (d) (e)

Fig. 6. Learning curves for DenseNet-121 on CIFAR-100. Top: training loss. Middle: training accuracy. Bottom: test accuracy. (a) SGDM. (b) RMSprop. (c) Adam. (d) AMSGrad. (e) RAdam.

class contains 500 training images and 50 validation images. Each image has been downsampled to the size of 64×64. This classification task is performed by ResNeXt50(32-4d) [33], which is trained by SGDM. This network is trained for 90 epochs, and the learning rate is reduced by multiplying 0.1 at the 30th and 60th epochs. The weight decay rate is set to $5e-4$, and the batch size is 100. The comparison results are shown in Fig. 7.

4) ImageNet: ILSVRC-2012 [34] is a large-scale image classification dataset with 1000 classes. The training data contain 1.2 million images, and the validation set contains 50 000 images. LRD is used by ResNet-50. This network is trained by SGDM with an initial learning rate of 0.1. The learning rate is divided by 10 at the 20th, 40th, and 60th epochs, and the training ends at the 80th epoch. The weight decay rate is set to $5e-4$, and the batch size is 256. The top-1 and top-5 classification accuracies are reported in Fig. 8.

5) Discussion: The above results validate the effectiveness of LRD for various classification tasks, ranging from the simple MNIST to the difficult ImageNet. Classification networks with LRD consistently show performance improvements, which proves the broad applicability of LRD. Its effectiveness is not limited to a specific task or network architecture. Besides the optimizers with and without LRD exhibit distinct convergence behaviors. These optimization algorithms without LRD either have slow training or lead to poor generalization. In contrast, LRD benefits all the optimization algorithms by improving the training speed and generalization. In particular, by embedding LRD into the nonadaptive method SGDM, its promoted convergence speed is even comparable to the adaptive methods. This is achieved by reducing the parameters updated at each iteration. For other adaptive methods, the classification accuracies are significantly improved. This is because the gradient accumulation in LRD helps prevent the trained models from overfitting.

6) Computational Efficiency: LRD is simple and efficient. It can be easily embedded into the existing gradient descent algorithms by adding two lines of code. LRD only performs two additional operations, sampling a random matrix and multiplying it with a predefined learning rate. In each iteration, updating a model with or without LRD has the same complexity $O(n)$, where n is the number of model parameters. The additional computational cost incurred by LRD is negligible. For example, it takes 31.3 s per epoch to train ResNet-34 on CIFAR-10 with SGDM. To perform LRD, it only takes an additional 1.2 s (i.e., 32.5 s per epoch).

B. Image Segmentation

Image segmentation can be regarded as a pixelwised classification task, which assigns each pixel in an image a semantic label [35]. It is more difficult than image classification. To further evaluate LRD, we conduct experiments on the PASCAL Visual Object Classes (VOC) 2012 semantic segmentation dataset [36]. This dataset consists of 20 object categories and one background category. Following the conventional setting in [37], the dataset is augmented by extra annotated VOC images provided in [38], which results in 10 582, 1449, and 1456 images for training, validation, and testing, respectively. The state-of-the-art segmentation model PSPNet [39] is used to perform the experiments. The optimizer is Adam, and its initial learning rate is 0.001. Other hyperparameters such as batch size and weight decay follow the setting in [39]. The segmentation performance is measured by the mean of classwise intersection over union (mean IoU) and pixelwise accuracy (pixel accuracy). The results for this experiment are reported in Fig. 9. With our proposed LRD, the network yields

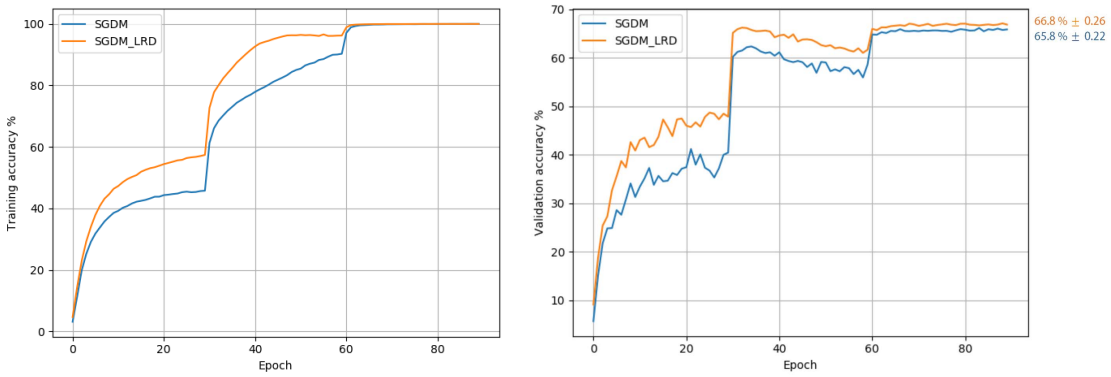


Fig. 7. Learning curves for ResNeXt50(32-4d) on Tiny-Imagenet. Left: training accuracy. Right: validation accuracy.

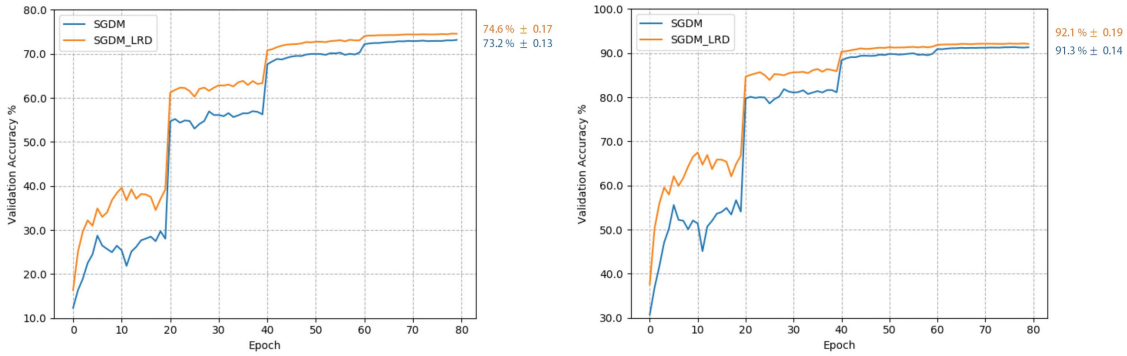


Fig. 8. Validation accuracy of ResNet-50 on Imagenet. Left: top-1 accuracy. Right: top-5 accuracy.

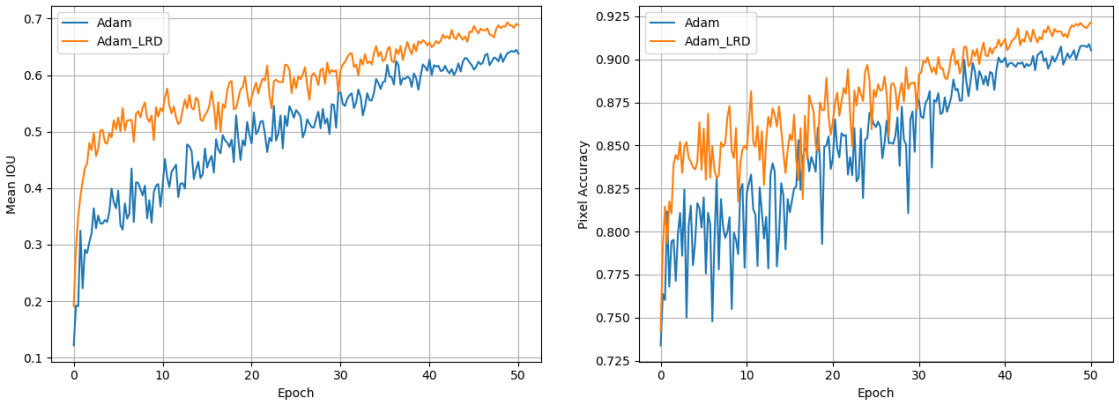


Fig. 9. Results for PSPNet on the VOC2012 semantic segmentation dataset. Left: mean IOU. Right: pixel accuracy.

results 0.688/0.921 in terms of mean IoU and pixel accuracy, exceeding the vanilla Adam of 0.637/0.905. In addition, the mean IOU of the network with LRD is 0.5 in the fifth epoch of training. Without LRD, similar results will not be reached until the 20th epoch. These experiments demonstrate that the segmentation network with LRD has stronger learning ability. With the help of LRD, the network can quickly learn useful knowledge from the training data. More importantly, LRD also helps the segmentation network avoid the problem of overfitting. Therefore, the network can output more accurate segmentation results.

C. Object Detection

Object detection is a challenging and important problem in the computer vision community. In this section, LRD is applied to an object detection task. The training data are a mixture of VOC2012 trainval and VOC2007 trainval, and the validation data are VOC2007 test. A single-stage network single shot multibox detector (SSD) [40] is used to detect objects from these data. SSD is trained with the optimizer Adam. Its initial learning rate is 0.001. Other hyperparameter settings are the same as in [40]. To show the effect of LRD on training, training loss and validation loss are used as evaluation

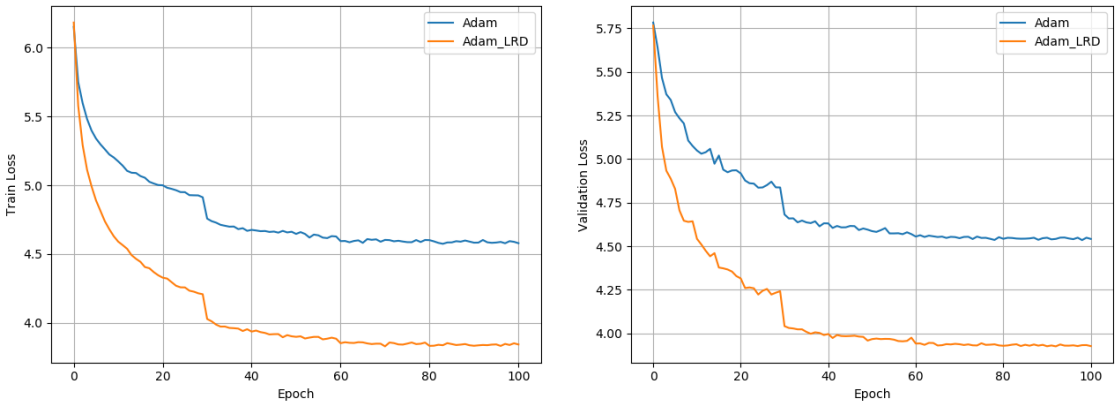


Fig. 10. Results for object detection. Left: training loss. Right: validation loss.

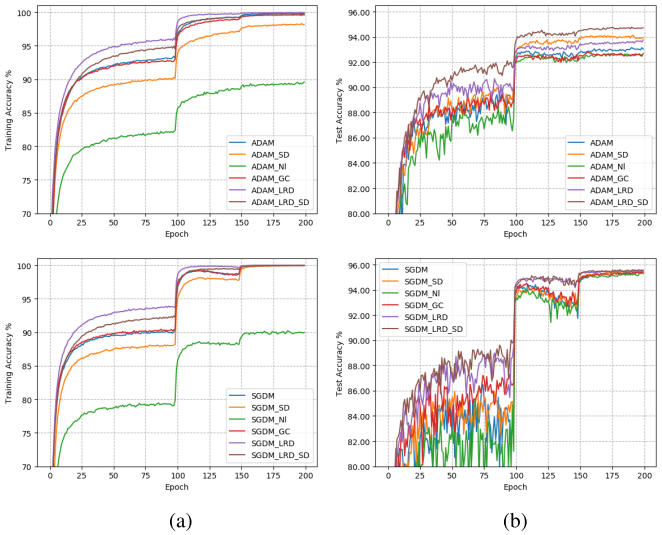


Fig. 11. Results on CIFAR-10 using different regularization methods. Top: Adam. Bottom: SGDM. (a) Training accuracy. (b) Test accuracy.

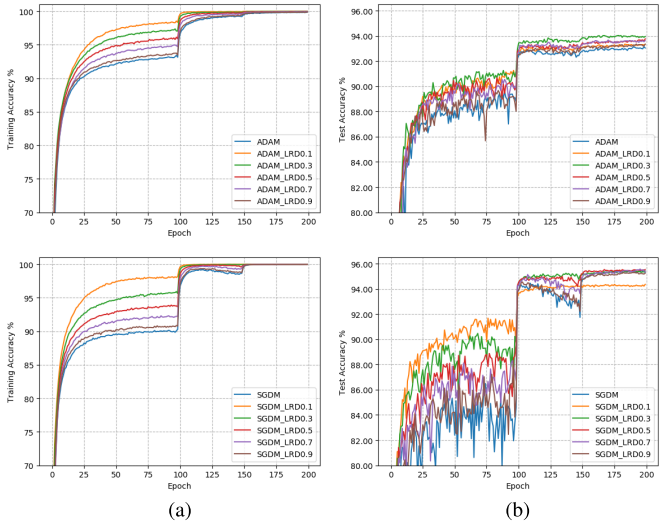


Fig. 12. Results on CIFAR-10 with different dropout rates p. Top: Adam. Bottom: SGDM. (a) Training accuracy. (b) Test accuracy.

TABLE II
TEST ACCURACY ON CIFAR-10 USING DIFFERENT REGULARIZATION METHODS (%)

	Adam	SGDM
No regularization	93.05±0.27	95.25±0.16
Standard dropout (SD)	93.98±0.38	95.36±0.25
Noise label (NL)	92.73±0.34	95.29±0.23
Gradient centralization (GC)	92.69±0.21	95.40±0.14
Learning rate dropout (LRD)	93.77±0.26	95.54±0.20
LRD and SD	**94.70±0.39**	**95.58±0.27**

metrics. The detection model is trained for 100 epochs, and the results are shown in Fig. 10. It is obvious that LRD causes the training loss and validation loss to drop faster and lower. Without LRD, the model is easily trapped in a local optimum so that the training loss flattens prematurely. Instead, LRD gives the optimizer more opportunities to escape from local optima and search for better results. Therefore, the detector can achieve higher detection accuracy.

D. Comparison With Other Regularizations

To show the superiority of LRD over other regularization methods, we compare LRD with three regularization methods: standard dropout [7], noisy label [9], and gradient centralization [25]. The standard dropout regularizes the network on hidden units, and the probability that each hidden unit is retained is set to 0.9. The noisy label disturbs each training sample with the probability 0.1 (i.e., a label is correct with a probability 0.9). For each disturbed sample, the label is randomly drawn uniformly from the other labels except the correct one. The gradient centralization regularizes both the weight space and the output feature space to prevent overfitting. These regularization techniques are applied to ResNet-34 trained on CIFAR-10. This model is trained multiple times using SGDM and Adam. For clarity, we use the terms "_SD," "_NL," and "_GC" to denote training with standard dropout, noisy label, and gradient centralization, respectively.

The learning curves are reported in Fig. 11. As can be seen, LRD can speed up training, while other regularization methods (SD and NL) slow down convergence. This is because

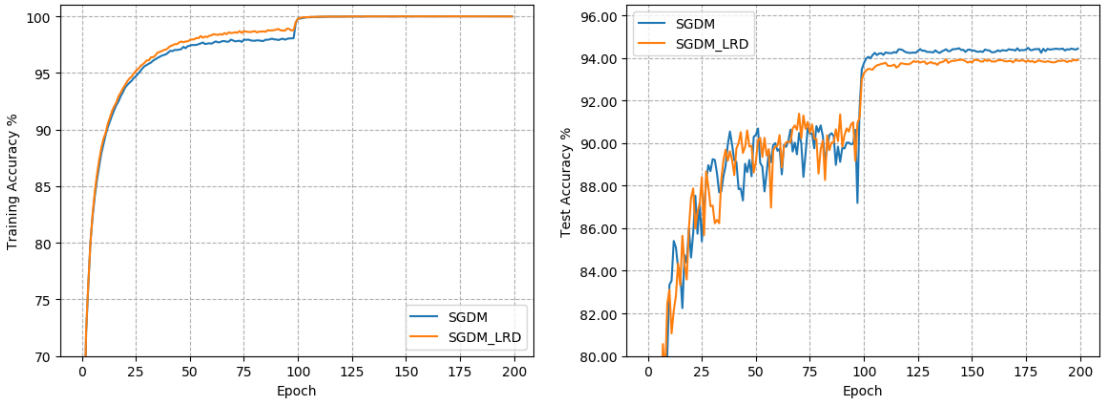


Fig. 13. Learning curves on CIFAR-10. The momentum factor β is set to 0. (a) Training accuracy. (b) Test accuracy.

TABLE III
TEST ACCURACY ON CIFAR-10 WITH DIFFERENT DROPOUT RATES p (%)

	Adam	SGDM
$p = 1$(No LRD)	93.05±0.27	95.25±0.16
$p = 0.1$	93.35±0.39	94.37±0.32
$p = 0.3$	**93.96±0.33**	95.42±0.21
$p = 0.5$	93.77±0.26	**95.54±0.20**
$p = 0.7$	93.63±0.23	95.40±0.15
$p = 0.9$	93.35±0.28	95.26±0.18

TABLE IV
TEST ACCURACY ON CIFAR-10 WITH DIFFERENT MOMENTUM FACTORS β (%)

β	0	0.5	0.9 (default)	0.99
SGDM w/o LRD	94.45±0.19	94.67±0.30	95.25±0.16	92.53±0.39
SGDM w/ LRD	93.91±0.22	95.08±0.29	95.54±0.20	93.36±0.42

these methods impose random perturbations on the training process, hindering network learning, and memorizing training data. The test accuracies are reported in Table II. The results show that the standard dropout and our LRD can effectively improve generalization, while the effects of noisy label and gradient centralization are not satisfactory. Furthermore, it is surprising that LRD can be complementary to other regularization methods. As can be seen from Fig. 11 and Table II, the combination of LRD and dropout can further improve the model performance. Note that the suffix "_LRD_SD" indicates that LRD and standard dropout are used in the same network. These results suggest that LRD is not limited to being a substitute for dropout or other methods.

E. Ablation Study

We provide ablation studies to explore the effects of each hyperparameter on LRD.

1) Effect of Dropout Rate p: As mentioned, the hyperparameter p can be tuned to control the amount of the parameters to be updated. Since p is a key hyperparameter of LRD, it is necessary to explore the effect of various p. The experiments are performed with ResNet-34 trained on CIFAR-10, where p is chosen from a broad range $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The smaller the p, the greater the probability that the learning rate of each parameter will be dropped. Adam and SGDM are adopted to train the models, and other hyperparameters are the same as previous experiments. The results are shown in Fig. 12. For any p, LRD can speed up training, but the smaller the p, the faster the convergence. This is because the smaller p leads to fewer parameters to be updated at each training step, so it is easier for the optimizer to find the local optimum in the parameter space. The test accuracies are reported in Table III. As can be seen, almost all the values of p lead to an improvement of test accuracy. To balance the training speed and generalization, it is recommended to set p in the range of 0.3–0.7.

2) Effect of Momentum Factor β: The existing optimization algorithms are typically equipped with gradient accumulation components, such as momentum. Here, we discuss the effect of momentum factor β (1) on LRD. The experiments are performed with a ResNet-34 trained on CIFAR-10. The optimizer is SGDM, where β is selected from $\{0, 0.5, 0.9, 0.99\}$. Note that in the above experiments, β defaults to 0.9. The comparison results are reported in Table IV. As observed, LRD results in significant performance gains when β is not zero. The key for LRD to improve model generalization is gradient accumulation (i.e., momentum). $\beta = 0$ means canceling momentum, which prevents LRD from accumulating gradients. In the absence of momentum, LRD still accelerates convergence but leads to overfitting (see Fig. 13). When momentum is available, LRD enables the optimizer to accumulate multiple gradients for model parameters whose value is fixed (the learning rate is dropped), so the optimizer has more energy to escape from the bad local optimum and achieve better model generalization.

3) Effect of Batch Size: Batch size is a critical hyperparameter for training classification networks. Studying the effect of batch size can give us more insight into LRD. The experimental results are shown in Table V. As observed, the batch size has a great impact on classification accuracy. An unreasonably large batch size may lead to overfitting and much computational cost. In this task (i.e., CIFAR-10), 128 is

TABLE V
TEST ACCURACY ON CIFAR-10 WITH DIFFERENT BATCH SIZES (%)

Batch size	32	64	128	256	512	1024
SGDM w/o LRD	93.93±0.37	95.02±0.31	95.25±0.16	95.17±0.24	94.83±0.21	94.26±0.19
SGDM w/ LRD	95.24±0.30	95.51±0.19	95.54±0.20	95.40±0.27	94.98±0.24	94.50±0.28

TABLE VI
TEST ACCURACY ON CIFAR-10 WITH DIFFERENT MOMENTUM UPDATE SCHEMES (%)

	EMAvg	Avg ($k = 10$)	Avg ($k = 100$)	Avg ($k = 500$)
SGDM w/o LRD	95.25±0.16	94.52±0.37	93.75±0.21	90.65±0.29
SGDM w/ LRD	95.54±0.20	95.03±0.34	94.46±0.30	91.39±0.38

an appropriate batch size for SGDM. In addition, the smaller the batch size, the better the effect of LRD. The possible reason is that small batch sizes tend to induce large gradients. By accumulating multiple gradients, LRD prevents overfitting more effectively.

4) Change Momentum: The above experiments have demonstrated the significance of momentum for LRD. In most optimizers, momentum is updated by (1). We further test the impact of the momentum update scheme on LRD. Specifically, the EMAvg term in (1) is changed to a simple average (Avg) of k gradients

$$M_t = \frac{1}{k}\sum_{\varepsilon=0}^{k-1} G_{t-\varepsilon}. \tag{10}$$

The results are reported in Table VI. The simple average gives k gradients the same weight at time step t. Obviously, this is not a good solution compared with EMAvg. G_t is more important than the previous $k-1$ gradients, so it deserves a larger weight. It is also observed that LRD still contributes to model generalization even if the momentum update scheme is changed. These experiments further prove the effectiveness of gradient accumulation to prevent overfitting.

F. Limitations and Future Work

LRD contains a tunable parameter p, and the choice of p will affect the training results. Although LRD is simple and effective, it is sometimes necessary to repeat experiments to find the best p. To further simplify LRD, a plausible solution is to set p as a trainable parameter. In this way, p is automatically updated rather than manually tuned, which may lead to better training results. We will demonstrate the feasibility of this solution in later work.

V. CONCLUSION

LRD is new gradient descent technique for regularizing neural network training. This technique encourages the optimizer to actively explore in the parameter space by randomly dropping some learning rates. At each training step, only the parameters whose learning rate is not dropped can be updated. Since only fewer parameters are to be updated, convergence becomes easier. LRD allows the optimizer to accumulate multiple gradients for model parameters whose value is fixed (the learning rate is dropped). The accumulated gradient helps the optimizer escape from the bad local optimum. The experiments show the substantial ability of LRD to accelerate training and enhance generalization. In addition, this technique is found to be effective in a wide variety of application domains including image classification, segmentation, and object detection. This shows that LRD is a general technique, which has great potential in practical applications.

REFERENCES

[1] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, 1999.
[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015, pp. 1–15.
[3] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *Proc. ICLR*, 2018, pp. 1–23.
[4] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[5] L. Liu *et al.*, "On the variance of the adaptive learning rate and beyond," in *Proc. ICLR*, 2019, pp. 1–14.
[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015, pp. 1–9.
[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.
[8] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[9] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian, "DisturbLabel: Regularizing CNN on the loss layer," in *Proc. CVPR*, Jun. 2016, pp. 4753–4762.
[10] A. Neelakantan *et al.*, "Adding gradient noise improves learning for very deep networks," 2015, *arXiv:1511.06807*.
[11] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy activation functions," in *Proc. ICML*, 2016, pp. 3059–3068.
[12] S. Wang and C. Manning, "Fast dropout training," in *Proc. ICML*, 2013, pp. 118–126.
[13] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, 2015.
[14] H. Robinds and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.
[15] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.
[16] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. NIPS*, 2017, pp. 1–11.
[17] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Proc. ICLR*, 2019, pp. 1–19.
[18] J. Chen, D. Zhou, Y. Tang, Z. Yang, Y. Cao, and Q. Gu, "Closing the generalization gap of adaptive gradient methods in training deep neural networks," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 1–17.
[19] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Comput.*, vol. 8, no. 3, pp. 643–674, Apr. 1996.

[20] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proc. ICML*, 2015, pp. 1–10.

[21] B. Chen, W. Deng, and J. Du, "Noisy softmax: Improving the generalization ability of DCNN via postponing the early softmax saturation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5372–5381.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[23] C. M. Bishop, "Training with noise is equivalent to Tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, 1995.

[24] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. ICML*, 2013, pp. 1058–1066.

[25] H. Yong, J. Huang, X. Hua, and L. Zhang, "Gradient centralization: A new optimization technique for deep neural networks," in *Proc. ECCV*, 2020, pp. 635–652.

[26] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. ICML*, 2003, pp. 928–936.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[28] W. An, H. Wang, Q. Sun, J. Xu, Q. Dai, and L. Zhang, "A PID controller approach for stochastic optimization of deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8522–8531.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1–9.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[31] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[33] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1492–1500.

[34] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[35] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder–decoder with atrous separable convolution for semantic image segmentation," in *Proc. ECCV*, 2018, pp. 801–818.

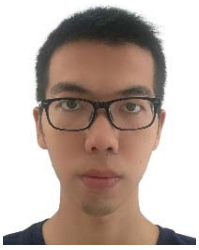
[36] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and W. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2010.

[37] G. Lin, C. Shen, A. van den Hengel, and I. Reid, "Efficient piecewise training of deep structured models for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 3194–3203.

[38] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 991–998.

[39] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2881–2890.

[40] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. ECCV*, 2016, pp. 21–37.



Huangxing Lin received the B.S. degree from Beijing Jiaotong University, Beijing, China, in 2015, and the M.S. degree from Xiamen University, Xiamen, China, in 2018, where he is currently pursuing the Ph.D. degree in signal and information processing.

His research interests include image processing, medical image analysis, and machine learning.

Weihong Zeng received the B.S. and M.S. degrees from the Department of Informatics and Communication Engineering, Xiamen University, Xiamen, China, in 2018 and 2021, respectively.

His current research interests include machine learning and image processing.

Yihong Zhuang received the B.S. degree from Fujian Agriculture and Forestry University, Fuzhou, China, in 2020. He is currently pursuing the master's degree with the Department of Informatics and Communication Engineering, Xiamen University, Xiamen, China.

His research interests mainly focus on self-supervised learning.

Xinghao Ding (Member, IEEE) was born in Hefei, China, in 1977. He received the B.S. and Ph.D. degrees from the Department of Precision Instruments, Hefei University of Technology, Hefei, in 1998 and 2003, respectively.

He was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, from 2009 to 2011. Since 2011, he has been a Professor with the School of Information Science and Engineering, Xiamen University, Xiamen, China. His main research interests include machine learning, representation learning, medical image analysis, and computer vision.

Yue Huang (Member, IEEE) received the B.S. degree from Xiamen University, Xiamen, China, in 2005, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2010.

She was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA, from 2015 to 2016. She is currently an Associate Professor with the Department of Communication Engineering, School of Information Science and Engineering, Xiamen University. Her main research interests include machine learning and image processing.

John Paisley (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Duke University, Durham, NC, USA, in 2004, 2007, and 2010, respectively.

He was a Post-Doctoral Researcher with the Computer Science Department, University of California at Berkeley, Berkeley, CA, USA, and the Computer Science Department, Princeton University, Princeton, NJ, USA. He is currently an Associate Professor with the Department of Electrical Engineering, Columbia University, New York, NY, USA, where he is also a member of the Data Science Institute. His current research is machine learning, focusing on models and inference techniques for text and image processing applications.