applied sciences

MDPI

Article

A Bounded Scheduling Method for Adaptive Gradient Methods

Mingxing Tang [1], Zhen Huang [1,*], Yuan Yuan [2], Changjian Wang [2] and Yuxing Peng [1]

[1] Science and Technology on Parallel and Distributed Laboratory, National University of Defense Technology, Changsha 410073, China
[2] College of Computer, National University of Defense Technology, Changsha 410073, China
* Correspondence: huangzhen@nudt.edu.cn

Received: 22 July 2019; Accepted: 28 August 2019; Published: 1 September 2019

Abstract: Many adaptive gradient methods have been successfully applied to train deep neural networks, such as Adagrad, Adadelta, RMSprop and Adam. These methods perform local optimization with an element-wise scaling learning rate based on past gradients. Although these methods can achieve an advantageous training loss, some researchers have pointed out that their generalization capability tends to be poor as compared to stochastic gradient descent (SGD) in many applications. These methods obtain a rapid initial training process but fail to converge to an optimal solution due to the unstable and extreme learning rates. In this paper, we investigate the adaptive gradient methods and get the insights on various factors that may lead to poor performance of Adam. To overcome that, we propose a bounded scheduling algorithm for Adam, which can not only improve the generalization capability but also ensure the convergence. To validate our claims, we carry out a series of experiments on the image classification and the language modeling tasks on several standard benchmarks such as ResNet, DenseNet, SENet and LSTM on typical data sets such as CIFAR-10, CIFAR-100 and Penn Treebank. Experimental results show that our method can eliminate the generalization gap between Adam and SGD, meanwhile maintaining a relative high convergence rate during training.

Keywords: deep neural networks; adaptive gradient methods; stochastic gradient descent; bounded scheduling method; image classification; language modeling

1. Introduction

Deep neural networks (DNNs) [1] have achieved great successes in many applications, such as image recognition [2], object detection [3], speech recognition [4,5], face recognition [6] and machine translation [7]. How to train DNNs quickly and accurately has attracted the attention of many researchers. Training neural networks is equivalent to solving the following non-convex optimization problems:

$$\min_{w \in R^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w), \tag{1}$$

where w is the parameter to train, n is the number of instances, $f_i(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a loss function defined on the instance with d dimensions and indexed i. Training algorithms need to search parameters to minimize the loss function.

Stochastic gradient descent (SGD) [8] has become the dominant training algorithm for DNNs. Simple as it is, SGD performs well in many applications. SGD obtains a smaller loss by moving the parameters of the model in the negative direction of gradient evaluated on a minibatch. The iteration of SGD can be described as follows:

$$w_k = w_{k-1} - \eta \nabla f_{i_k}(w_{k-1}), \tag{2}$$

where η is learning rate, i_k is the instance index at the k-th iteration, $\nabla f_{i_k}(w_{k-1})$ denotes the stochastic gradient computed at w_{k-1}.

There are two main drawbacks of SGD. The first one is SGD needs to find an appropriate learning rate, which means that excessive learning rate will cause the loss function unable to converge to the optimal value and exceptionally small learning rate will slow down the convergence speed of loss function. The other one is SGD scales the gradient uniformly in all directions, which leads that the ill-scaled or sparse problems cannot be solved well [9].

To train DNNs, SGD uses a standard decreasing learning rate scheme, where the learning rate is initialized as a large value at the beginning and decreases gradually with iteration. However, a suitable initial learning rate is difficult to tune. Linear search [10] and grid search are often used to find the optimal learning rate, but the computational overhead is high. Cyclical learning rates method [11] changes the learning rate periodically within a fixed bound, which can practically eliminate the need to experimentally find the best values and schedule for the global learning rates. Then a super-convergence method [12] is proposed to train networks with one learning rate cycle and a large maximum learning rate, which can achieve an increase in performance compared with standard methods. However, the uniformly scaled gradient still makes these methods perform poorly when the data set is sparse or ill-scaled.

In recent years, a series of adaptive gradient methods have been proposed. These methods scale the gradient by some form of squared past gradients, which can achieve a rapid training speed with an element-wise scaling term on learning rates [13]. Adagrad [9] is the first popular algorithm to use an adaptive gradient, which has obviously better performance than SGD when the gradients are sparse. However, the learning rate of Adagrad will drop rapidly because of its accumulation of the squared gradients in the denominator, which may lead to deterioration in the case that the loss functions are non-convex or gradients are dense. Then Adadelta [14], RMSprop [15], Adam [16], Nadam [17] are proposed to fix this issue, which use the exponential moving averages of squared past gradients to avoid the rapid drop of learning rate. These algorithms have been successfully applied to a variety of practical problems, especially Adam has become the default algorithm for training neural networks.

When training DNNs with adaptive gradient methods, the loss function decreases rapidly in the early stage of training, but the final training loss and test loss are worse than that of SGD in many applications. Moreover, since the learning rate of Adam does not decrease monotonously, the training process will diverge in some applications [18]. Some work has proposed a hybrid scheme of Adam and SGD to solve these problems. SWATS [19] proposes a strategy that Adam can be switched to SGD when a triggering condition is satisfied, which can close the generalization gap between Adam and SGD. ADABOUND [13] can achieve a gradual and smooth transition from adaptive methods to SGD by employing dynamic bounds on learning rates. For these hybrid algorithms, the switching time of Adam and SGD and the learning rate of SGD after switching still have a great impact on the performance of the algorithm, which should be tuned elaborately.

In this paper, we study the adaptive gradient algorithms and propose a bounded scheduling method for Adam, called Bsadam, to improve the performance when training neural networks. The major contributions of this paper include:

1. We investigate the factors that lead to the poor performance of Adam while training complex neural networks.
2. We set effective bounds for the learning rate of Adam without manual tuning, which can improve the generation capability.
3. We schedule the bounds of learning rate to improve the performance of Adam. Firstly we fix the upper bound and increase the lower bound gradually to find wide, flat minima. Then we fix the lower bound and decrease the upper bound gradually to ensure the convergence of training. At last, a fixed learning rate is used to make the algorithm converge to the optimal solution.

4. We train multiple tasks on several models to evaluate the algorithm. MNIST [20] is trained on simple neural networks, CIFAR-10 [21] and CIFAR-100 [21] are trained on ResNet [22], DenseNet [23] and SENet [24], Penn Treebank [25] is trained on LSTM [26]. All these experiments show that our method is capable of eliminating the generalization gap between Adam and SGD and maintaining a higher convergence speed in training.

The rest of our paper is organized as follows. In Section 2, the background of this paper is reviewed, where the traditional learning rate methods and adaptive gradient methods are described. In Section 3, we introduce the bounded scheduling scheme for Adam. In Section 4, we present a series of experiments to verify the effectiveness of our method. In Section 5, we summarize the paper.

2. Background

2.1. Traditional Learning Rate Methods

Learning rate is one of the most important hyper-parameters of gradient-based optimization methods, there have been many related works on it. Line search [10] is often used to find the learning rate of the full gradient. The line search method will set a large initial learning rate and try a learning rate at each iteration, if the loss function does not fall a certain distance than the current value, the learning rate will decrease proportionally and iterate again, until the learning rate satisfying the fall condition is found. Line search needs a large amount of computation and is often used when the data set is small. A line search method for SGD is also proposed [27]. This method uses random samples to do basic line search and estimates the Lipschitz constant L, then deduces the theoretical optimal learning rate based on L. However, the optimal learning rate, in theory, is different from that in practice and this method can not guarantee convergence.

Barzilai-Borwen method [28,29] is also often used to estimate the learning rate. The Barzilai-Borwen method is based on the quasi-Newton method and uses second-order derivative information to evaluate the learning rate, which requires little extra computational overhead. However, the learning rate estimated by the Barzilai-Borwen method may lead to the divergence of the training process. Yann Ollivier et al. proposed a method to view the whole performance of the learning trajectory as a function of the learning rate, then adapt the learning rate by performing a gradient descent on the learning rate itself [30]. Although these methods do not need to search the learning rate, their performance is not good enough compared with the manually tuned optimal learning rate. Cyclical learning rate method [11] does not need to use a certain learning rate, but makes the learning rate vary periodically in a certain range. Then super-convergence [12] is proposed to train DNNs with one cycle and a large maximum learning rate, which provides a boost in performance. Traditional learning rate methods scale the gradient uniformly in all directions, the performance of which will decrease when data sets are sparse or ill-scaled.

2.2. Adaptive Gradient Methods

The recently proposed adaptive gradient methods can provide an element-wise scaling term on learning rates without the need to tune the learning rate manually. These methods use historical information to estimate the curvature of the loss function and adopt different learning rates for each parameter, so the learning rate is a vector and each element for a parameter, which is different from the traditional learning rate methods. The representative adaptive gradient methods are Adagrad [9], RMSprop [15], Adam [16], AMSgrad [18], etc.

Adagrad [9] is the first proposed adaptive gradient method. Its main idea is to adopt a smaller learning rate for the parameters corresponding to frequent features and a larger learning rate for the parameters corresponding to infrequent features. Therefore, Adagrad is very suitable for training sparse data, which can improve the robustness of SGD. The update of Adagrad can be shown as follows:

$$w_k = w_{k-1} - \eta \frac{\nabla f(w_{k-1})}{\sqrt{v_k} + \epsilon}, \tag{3}$$

where

$$v_k = \Sigma_{j=0}^{k-1} \nabla f(w_j)^2, \tag{4}$$

ϵ is a smoothing term that avoids division by zero, η is general learning rate.

Adagrad uses the accumulation of the squared gradients and the squared gradients are positive, which will lead to a rapid decline in learning rate to infinite small and the standstill of loss function. RMSprop [15] was proposed to solve the problem of the rapid disappearance of the gradient for Adagrad. The update rule of RMSprop is the same as (3), but the updating of v_k adopts exponential decaying average of square gradients, which can be shown as follows:

$$v_k = \beta v_{k-1} + (1 - \beta) \nabla f(w_{k-1})^2, \tag{5}$$

where $\beta \in [0, 1)$ is the hyper-parameter that controls the exponential decay rate of average. The use of the exponential moving averages of squared past gradients can prevent the rapid rise of v_k and the learning rate will not decline rapidly.

Adam [16] can also calculate adaptive learning rate for each parameter. As a complement to RMSprop, Adam preserves the exponential moving averages of squared past gradients, as well as the exponential moving averages of past gradients, which gives the gradient momentum. The update formula of Adam is shown as follows:

$$w_k = w_{k-1} - \eta \cdot \frac{\sqrt{1 - \beta_2^k}}{1 - \beta_1^k} \cdot \frac{m_k}{\sqrt{v_k} + \epsilon}, \tag{6}$$

where

$$\begin{aligned} m_k &= \beta_1 m_{k-1} + (1 - \beta_1) \nabla f(w_{k-1}), \\ v_k &= \beta_2 v_{k-1} + (1 - \beta_2) \nabla f(w_{k-1})^2, \end{aligned} \tag{7}$$

where $\beta_1, \beta_2 \in [0, 1)$ are hyper-parameters that control the exponential decay rate of moving average.

Reddi et al. pinpoint that the use of exponential moving averages of squared past gradients may make Adam fail to converge to the optimal solution. As a result, AMSGrad was proposed [18]. Unlike Adam, AMSGrad uses the maximum of exponential moving averages of squared past gradients, the update rule of v_k is show as follows:

$$\begin{aligned} \widehat{v}_k &= \beta_2 \widehat{v}_{k-1} + (1 - \beta_2) \nabla f(w_{k-1})^2, \\ v_k &= \max(\widehat{v}_k, v_{k-1}). \end{aligned} \tag{8}$$

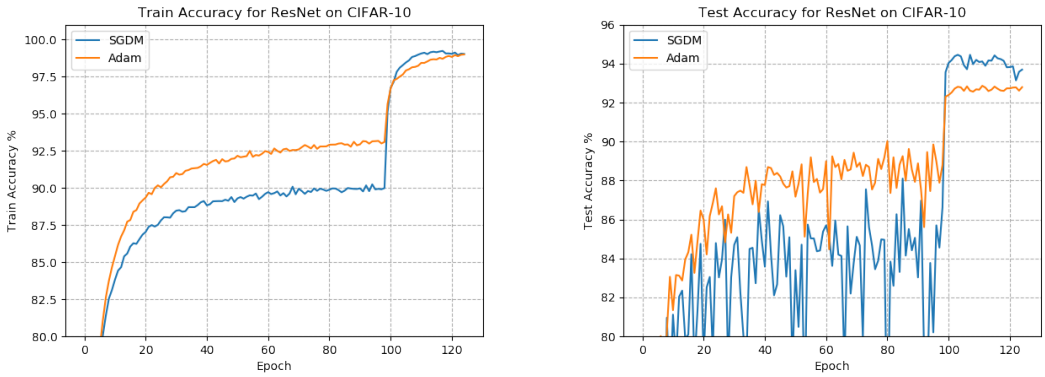
The adaptive gradient methods has low generalization ability in training complex models and its performance is worse than the optimal learning rate tuned manually.

3. Bsadam: Bounded Scheduling Method for Adam

3.1. Preliminaries

Firstly, we use an empirical study to illustrate the existence of the generalization gap in Adam. We use SGD and Adam to do image classification for CIFAR-10 data set on ResNet-34 architecture and present training accuracy and test accuracy in Figure 1. As can be seen from Figure 1, the training and test accuracy of Adam both increased faster than that of SGD in the early stage. However, when the learning rate is reduced by 10 after 100 epochs, the training and test accuracy of Adam are lower than that of SGD. Although the final training accuracy of Adam reaches the level of SGD, its test accuracy is still 1% to 2% lower than that of SGD, which means that its generalization gap is larger than SGD.

There may be various factors that may lead to the weakly empirical generalization capability of Adam. Based on previous researches [13,19,31–33], we summarize these factors and work to eliminate them. The main factors can be listed as follows.



(**a**) train (**b**) test

Figure 1. Training the ResNet-34 architecture on the CIFAR-10 data set with stochastic gradient descent (SGD) and Adam. Adam has a faster initial convergence speed, but the final test accuracy is lower than that of SGD.

- The non-uniform scaling of the gradients will lead to the poor generalization performance of adaptive gradients methods. SGD is uniformly scaled and low training error will generalize well [19,31].
- The exponential moving average used in Adam can't make the learning rate monotonously decline, which will cause it to fail to converge to an optimal solution and arise the poor generalization performance [32,33].
- The learning rate learned by Adam may circumstantially be too small for effective convergence, which will make it fail to find a right path and converge to a suboptimal point [13].
- Adam may aggressively increase the learning rate, which is detrimental to the overall performance of the algorithm [13,18].

Taking all these factors into account, some improvements needs to be considered for Adam. Upper and lower bounds should be specified to avoid the side effect caused by extreme large and small learning rate. At the later stage of training, learning rate should be monotonous decreased to ensure the convergence and be uniformly scaled to improve generalization performance.

3.2. Specify Bounds for Adam

In this paper, we use the curve of loss function obtained by learning rate range test (LR range test) [11] to determine the upper and lower bounds of the learning rate for Adam. When training a new model or data set, the LR range test is a very effective way to find a reasonable learning rate range for SGD, although it can not find a specific learning rate. LR range test uses SGD to train the model for several epochs and makes the learning rate increase linearly from small to large, then the approximate range of reasonable learning rate can be estimated by the curve of the loss function. Specifically, when the loss decreases, it means that the current learning rate is reasonable when the loss rises, it means that the current learning rate is inappropriate.

However, as a result that Adam itself has the function of adjusting the learning rate, the standard of specifying the bounds for Adam is different from the classical LR range test, we need a wider range of bounds. Specifically, the lower bound can be set to the point where the loss function begins to decline and the upper bound can be set to the point where the loss function begins to rise. What is more, in order to get better generalization ability, the upper bound can be enlarged within five times.

For example, we use Resnet-34 architecture to perform the LR range test on CIFAR-10 and obtain the curve of loss function along with the learning rate. The result is shown in Figure 2. As can be seen from Figure 2, the loss begins to decline obviously when the learning rate is 0.001, so 0.001 can be used as the lower bound of the learning rate for Adam. When the learning rate is 0.1, the loss starts to rise and the training process starts to diverge, so 0.1 can be used as the upper bound of the learning rate for Adam. However, through experiments, we find that the algorithm can get better minima by increasing the upper bound appropriately, so the upper bound can be set to 0.5. The upper and lower bounds of learning rate are limited, the negative effects of too large or too small learning rate on Adam can be eliminated.

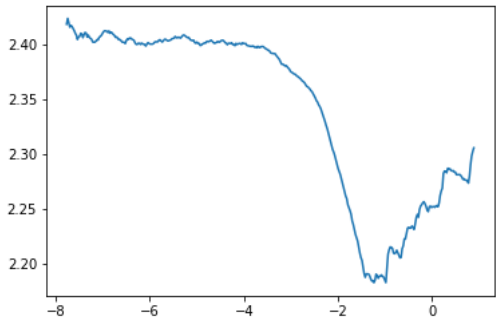


Figure 2. Learning rate range test. The x-axis is learning rate (log scale), the y-axis is training loss.

3.3. Schedule Bounds for Adam

We improve the empirical generalization capability of Adam by scheduling its lower and upper bounds, which can reduce the adverse effects of the non-uniform scaling of the gradients and the non-monotonically decreasing learning rate. According to the updated formula of Adam, we can regard $\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}$ as the learning rate of Adam and m_k as gradients with momentum of Adam. Gradient clipping can constrain the learning rate to a certain range, which is an effective method to solve the problem of gradient disappearance or gradient explosion. We use gradient clipping to clip the learning rate of Adam which exceeds the threshold. Consider applying the following operations to Adam

$$Clip(\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}, min\_lr, max\_lr), \tag{9}$$

which can clip the learning rate of Adam element-wisely such that each element of the learning rate is limited in the range of $[min\_lr, max\_lr]$, where min_lr and max_lr are lower bound and upper bound found in Section 3.2 respectively.

Then we will schedule the bounds of learning rate. The scheduling process is divided into three phases, which are finding minima, converging and uniform scaling. The scheduling details for each phase are described in detail below.

3.3.1. Finding Minima

In this phase, we use the concept of super-convergence, which implies that a large maximum learning rate can achieve better generalization capability. Using a relatively large learning rate in the early stage of training can make the loss function skip the suboptimal solution more easily and find wide, flat minima. Therefore, we fix the upper bound of learning rate and gradually increase the lower bound of learning rate, so that each element of learning rate can gradually rise to the upper bound. In this phase, gradient clipping can be expressed as follows:

$$Clip(\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}, ascending(t), max\_lr), \tag{10}$$

where $ascending(t)$ is a function that lower bound increases gradually from min_lr to max_lr with iteration and t means the progress of epoch in this phase. $ascending(t)$ can be linear, exponential and trigonometric, which can be formulated as follows:

- linear rise:

$$ascending(t) = min\_lr + t \cdot \frac{max\_lr - min\_lr}{T}, \tag{11}$$

- exponential rise:

$$ascending(t) = min\_lr \cdot (\frac{max\_lr}{min\_lr})^{\frac{t}{T}}, \tag{12}$$

- trigonometric rise:

$$ascending(t) = min\_lr + (max\_lr - min\_lr)\sin(\frac{t}{T} \cdot \frac{\pi}{2}), \tag{13}$$

where T is the total epochs in this phase.

3.3.2. Converging

In this phase, to avoid the divergence or poor generalization performance caused by the non-monotonic decline of learning rate, we need to make sure that the learning rate of Adam is decreasing. Therefore, we fix the lower bound of learning rate and gradually decrease the upper bound of learning rate, so that each element of learning rate can gradually decrease to the lower bound. In this phase, gradient clipping can be expressed as follows:

$$Clip(\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}, min\_lr, descending(t)), \tag{14}$$

where $descending(t)$ is a function that upper bound decreases gradually from max_lr to min_lr with iteration and t means the progress of epoch in this phase. $descending(t)$ can be linear, exponential and trigonometric, which can be formulated as follows:

- linear decrease:

$$descending(t) = max\_lr - t \cdot \frac{max\_lr - min\_lr}{T}, \tag{15}$$

- exponential decrease:

$$descending(t) = max\_lr \cdot (\frac{min\_lr}{max\_lr})^{\frac{t}{T}}, \tag{16}$$

- trigonometric decrease:

$$descending(t) = max\_lr - (max\_lr - min\_lr)\sin(\frac{t}{T} \cdot \frac{\pi}{2}), \tag{17}$$

where T is the total epochs in this phase.

3.3.3. Uniform Scaling

There is a conventional phase for training neural networks, which is reducing the learning rate by 10 in the final stage of training, so that the algorithm will converge to the near minimum. In our algorithm, at the end of the converging phase, upper bound are reduced to min_lr, so the gradients are uniformly scaled. We use min_lr as a learning rate continuing training model. The training accuracy and test accuracy will be further improved and stabilized and the algorithm will eventually converge. In this phase, the gradients are uniformly scaled, which will help improve generalization performance.

3.4. Algorithm Overview

Based on the above analysis, in this subsection, we propose a new variant of the optimization methods, named Bsadam, which can maintain the fast convergence of the algorithm in the early stage and obtain a good finally generalization capacity.

Empirically, the number of epoch in the first phase is the same as that in the second phase and the number of epoch in the third phase should be less than that in the former two phases. Specifically, if the total number of training epochs is T, the allocation of the number of epochs for three phases are $\frac{2T}{5}$, $\frac{2T}{5}$ and $\frac{T}{5}$, respectively. The details of Bsadam are illustrated in Algorithm 1, where max_lr and min_lr can be found by the method mentioned in Section 3.2, β_1, β_2 and η is the hyper-parameters of Adam itself, $data\_loader()$ is a function that combines a data set and a sampler and provides an iterable process over the given data set.

Algorithm 1 Bsadam.

Parameters : total epochs T, max_lr, min_lr
Initialize : w_0, β_1, β_2, η

set $v_0 = 0$, $m_0 = 0$, $k = 0$
for $t = \{1, 2, ..., \frac{2T}{5}\}$ **do**
  $ascending\_lr = ascending(t)$
  for *data* in $data\_loader()$ **do**
    $k = k + 1$
    compute gradient $g_k = \nabla f(w_{k-1})$ on *data*
    $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
    $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
    $lr = Clip(\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}, ascending\_lr, max\_lr)$
    $w_k = w_{k-1} - lr \cdot m_k$
  end for
end for
for $t = \{1, 2, ..., \frac{2T}{5}\}$ **do**
  $ascending\_lr = descending(t)$
  for *data* in $data\_loader()$ **do**
    $k = k + 1$
    compute gradient $g_k = \nabla f(w_{k-1})$ on *data*
    $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
    $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$
    $lr = Clip(\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{\eta}{\sqrt{v_k}+\epsilon}, min\_lr, descending\_lr)$
    $w_k = w_{k-1} - lr \cdot m_k$
  end for
end for
for $t = \{1, 2, ..., \frac{T}{5}\}$ **do**
  $lr = min\_lr$
  for *data* in $data\_loader()$ **do**
    $k = k + 1$
    compute gradient $g_k = \nabla f(w_{k-1})$ on *data*
    $w_k = w_{k-1} - lr \cdot g_k$
  end for
end for

4. Experiments

To illustrate the effectiveness of our algorithm, we experimented with different models on different data sets to compare the new variant with other popular optimization methods, such as SGD with momentum (SGDM), Adagrad and Adam. We mainly consider two problems that are often solved by deep neural networks: image classification and language modeling. The models used in the experiment include feedforward neural network, convolutional neural network [34], deep convolutional neural network and recurrent neural network, The data sets used in the experiment are MNIST [20], CIFAR-10 [21], CIFAR-100 [21], Penn Treebank [25]. All these models or data sets are often encountered in deep learning.

4.1. Experimental Setup

We implemented these experiments on a server configured as 2 NVIDIA TITAN XP GPUs, 1 Intel I7-6800K CPU, 16G*8 DDR4, 240G SSD and 1T SATA. These experiments were coded in PyTorch, each experiment was run three times and we chose the best one.

The algorithms under consideration have many hyper-parameters and the setting of hyper-parameters has a great influence on the performance of the optimization algorithm. Here we will describe how we adjust hyper-parameters. We use a logarithmical grid search on a large space of learning rate and then fine-tune it, the results are shown in Table 1. Specifically, the learning rate of each algorithm is adjusted as follows:

- **SGD(M)** We used SGDM for image classification tasks and SGD for language modeling tasks. When using SGDM, we set the momentum parameter to the default value 0.9. we roughly tuned the learning rate for SGD(M) on a logarithmic scale from 10^{-3} to 10^{2} first and then fine-tune the learning rate.
- **Adagrad** The general learning rates used for Adagrad are chosen from {0.0005, 0.001, 0.005, 0.01, 0.1}.
- **Adam** The general learning rates used for Adam were chosen from {0.0005, 0.001, 0.005, 0.01, 0.05}. We set $\{\beta_1, \beta_2\}$ as the recommended default value {0.9, 0.999}. The perturbation value ϵ is set to 10^{-8}.
- **Bsadam** We used the same hyper-parameter settings for Adam. The upper and lower bounds were determined by learning rate range test and the rise and decrease of bounds are linear.

Other hyper-parameters such as batch size and weight decay use the default values recommended by the model.

Table 1. Summarizing the models and the data sets utilized for our experiments. The optimal hyperparameters for stochastic gradient descent (SGD) with momentum (M), Adagrad, Adam and Bsadam for all experiments are also listed.

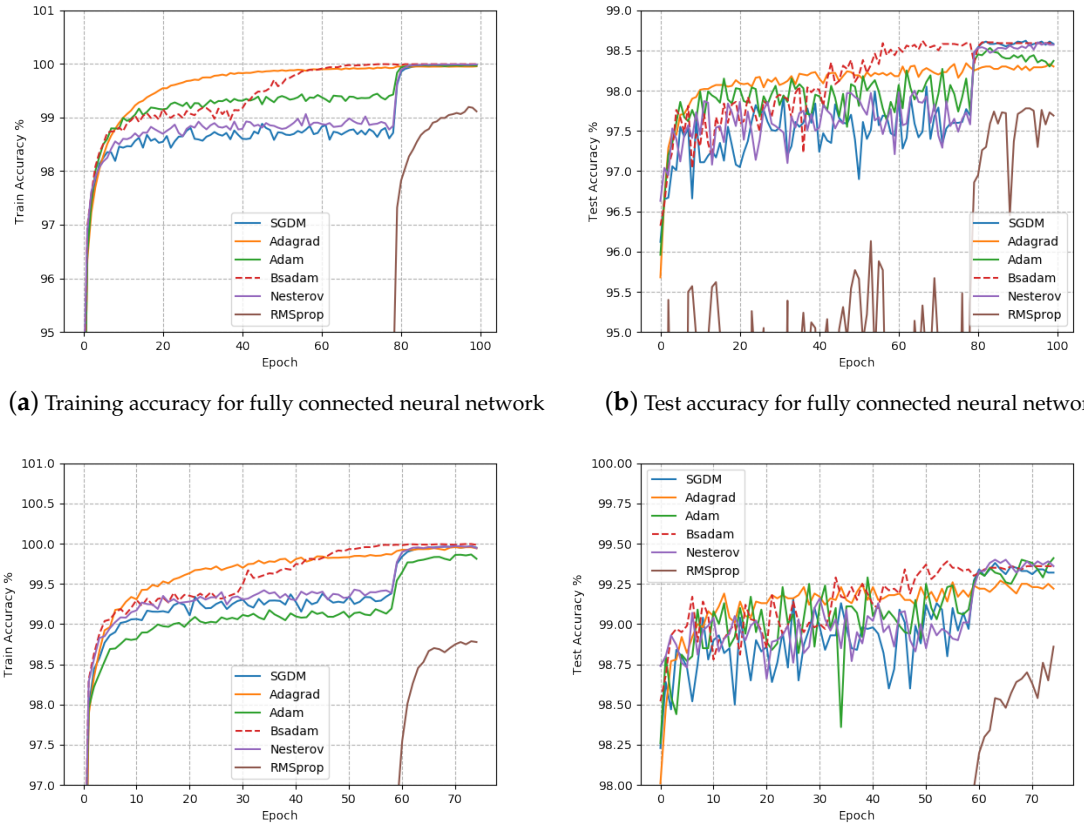
Data Set	Model	Network Type	SGD(M)	Adagrad	Adam	Bsadam
MNIST	1-Layer Perceptron	Feedforward	0.1	0.001	0.001	(0.01,0.5)
MNIST	1-Layer Convolutional	Convolutional	0.1	0.001	0.001	(0.01,0.5)
CIFAR-10	ResNet	Deep Convolutional	0.1	0.001	0.001	(0.01,0.5)
CIFAR-10	DenseNet	Deep Convolutional	0.1	0.001	0.001	(0.01,0.5)
CIFAR-10	SENet	Deep Convolutional	0.1	0.001	0.001	(0.01,0.5)
CIFAR-100	ResNet	Deep Convolutional	0.3	0.001	0.001	(0.05,1)
CIFAR-100	DenseNet	Deep Convolutional	0.1	0.001	0.001	(0.05,1)
CIFAR-100	SENet	Deep Convolutional	0.1	0.001	0.001	(0.01,0.5)
Penn Treebank	1-Layer LSTM	Recurrent	50	0.001	0.001	(5,100)
Penn Treebank	2-Layer LSTM	Recurrent	50	0.001	0.001	(5,100)

4.2. Image Classfication

4.2.1. Simple Neural Network

The MNIST database of handwritten digits has a training set of 60,000 images, and a test set of 10,000 images, which can be divided into 10 classes. We train a simple fully connected neural network with one hidden layer and a one-layer convolutional network with one convolutional layer and one fully connected layer for the image classification problem on the MNIST dataset. We run 100 epochs and decay the learning rate by 10 at 80th epoch for fully connected neural network, then we run 75 epochs and decay the learning rate by 10 at 60th epoch for convolutional network.

Figure 3 shows the learning curve of each optimization method, which includes training accuracy and test accuracy. We find that all the optimization algorithms can achieve nearly 100% accuracy on the training set. However, the accuracy of each algorithm will be different on the test set. Among these algorithms, Adagrad converges fastest on the training set, but achieves lower accuracy on the test set and SGDM has a slightly better accuracy on the test set than Adam and Adagrad. Our proposed Bsadam has better convergence speed than SGDM in the early stage. Especially in the converging phase, the convergence speed of Bsadam is faster than all the compared algorithms on both training and test set. Moreover, the final test accuracy of Bsadam is as good as fine-tuned SGDM, which means that our algorithm can get faster training speed without sacrificing accuracy when training simple neural networks. We also run RMSProp and Nesterov with default setting on MNIST. We find that RMSProp has worst convergence speed and test accuracy, Nesterov has similar performance with SGD with momentum. So our method still has advantages over these methods.



(a) Training accuracy for fully connected neural network

(b) Test accuracy for fully connected neural network

(c) Training accuracy for one-layer convolutional neural Network

(d) Test accuracy for one-Layer convolutional neural network

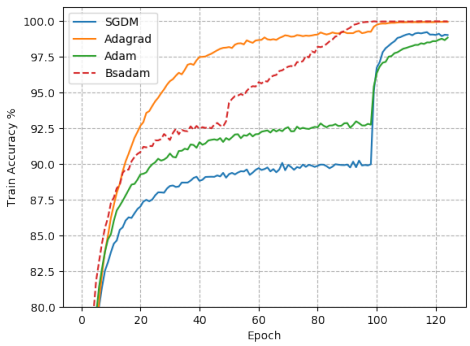
Figure 3. Training and test accuracy for fully connected neural network and one-layer convolutional neural network on MNIST.

4.2.2. Deep Convolutional Network

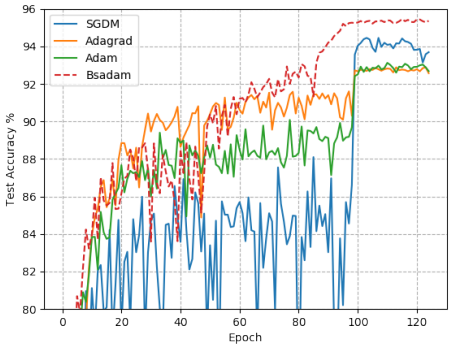
We evaluate our algorithm on a more complex deep convolutional network. Specifically, we perform experiments with three architectures: ResNet-34 [22], DenseNet-121 [23] and SENet-34 [24] on CIFAR-10 and CIFAR-100 data sets for image classification tasks. These data sets have a training set of 50,000 32 $\times$ 32 RGB images, and a test set of 10,000 images, which can be divided into 10 classes for CIFAR-10 and 100 classes for CIFAR-100. We ran 125 epochs for all the compared algorithms and decay the learning rate by 10 at 100th epoch.

Figure 4 shows the learning curve of each optimization method running on CIFAR-10, which includes training accuracy and test accuracy. As we can see, Adagrad had faster convergence speed and higher accuracy on training set, its accuracy is the lowest on test set, which indicates that its generalization gap is relatively large. Adam converges faster than SGDM in the early training, but the final test accuracy is lower than SGDM. SGDM has the slowest convergence speed on training set and test set, but its final test accuracy is higher than Adam and Adgrad, which means that its generalization capability is better than adaptive gradient methods. Our proposed Bsadam converges faster than fine-tuned SGDM in the early training. Especially in the converging phase, the convergence speed of Bsadam is faster than all the compared algorithms on both training and test set. The final training and test accuracy of Bsadam are the highest among all the compared algorithms, which indicates that our algorithm can accelerate the training process and improve the accuracy for complex deep neural networks.

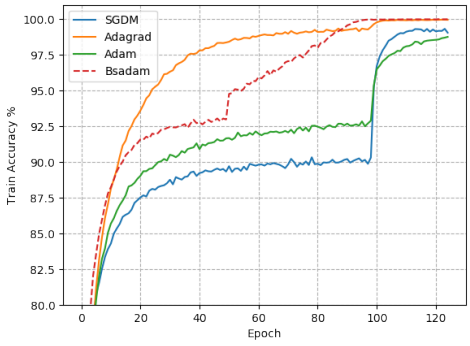
Figure 5 shows the learning curve of each optimization method running on CIFAR-100, which includes training accuracy and test accuracy. The experimental results shown in Figure 5 are similar to Figure 4. The adaptive gradient methods often exhibit a relatively large generalization gap. Bsadam can achieve faster convergence speed and higher convergence accuracy on both training and test set.



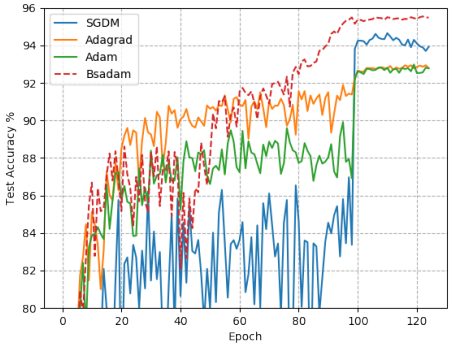
(**a**) Training accuracy for ResNet-34



(**b**) Test accuracy for ResNet-34

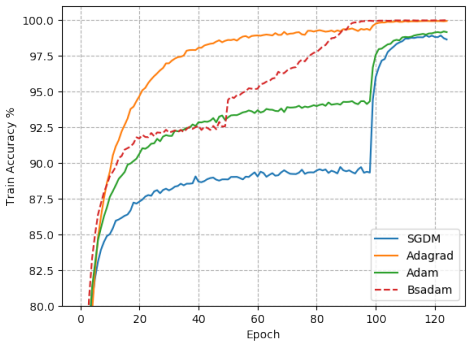


(**c**) Training accuracy for SENet-34

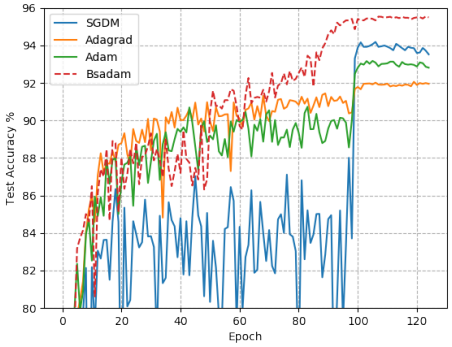


(**d**) Test accuracy for SENet-34

Figure 4. *Cont.*

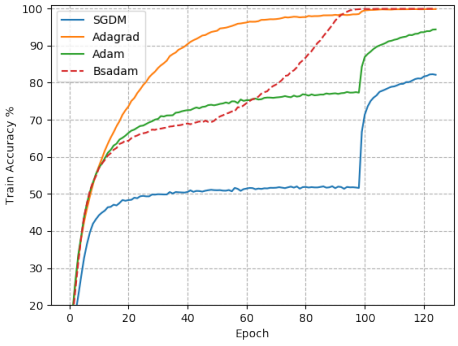


(**e**) Training accuracy for DenseNet-121

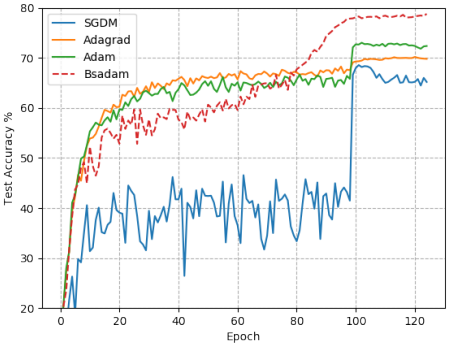


(**f**) Test accuracy for DenseNet-121

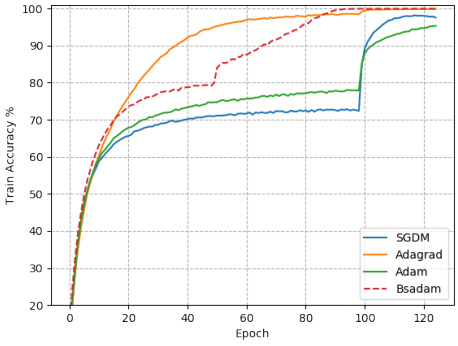
Figure 4. Training and test accuracy for ResNet-34, SENet-34 and DenseNet-121 on CIFAR-10.



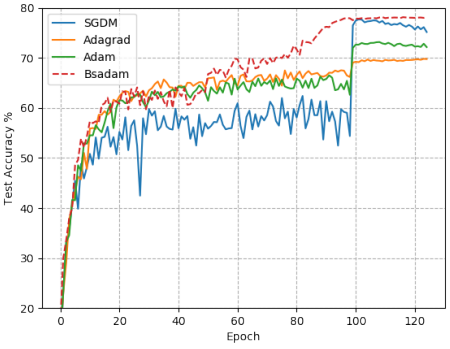
(**g**) Training accuracy for ResNet-34



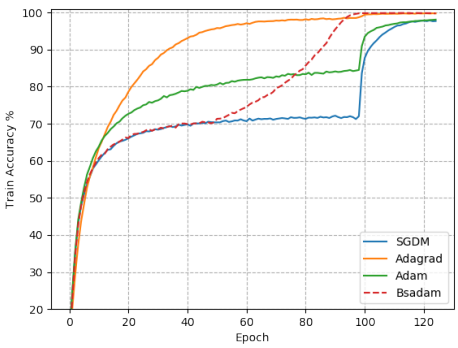
(**h**) Test accuracy for ResNet-34



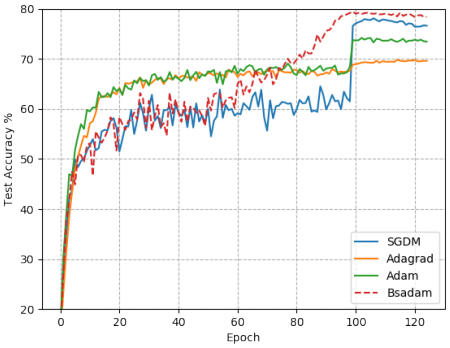
(**i**) Training accuracy for SENet-34



(**j**) Test accuracy for SENet-34



(**k**) Training accuracy for DenseNet-121



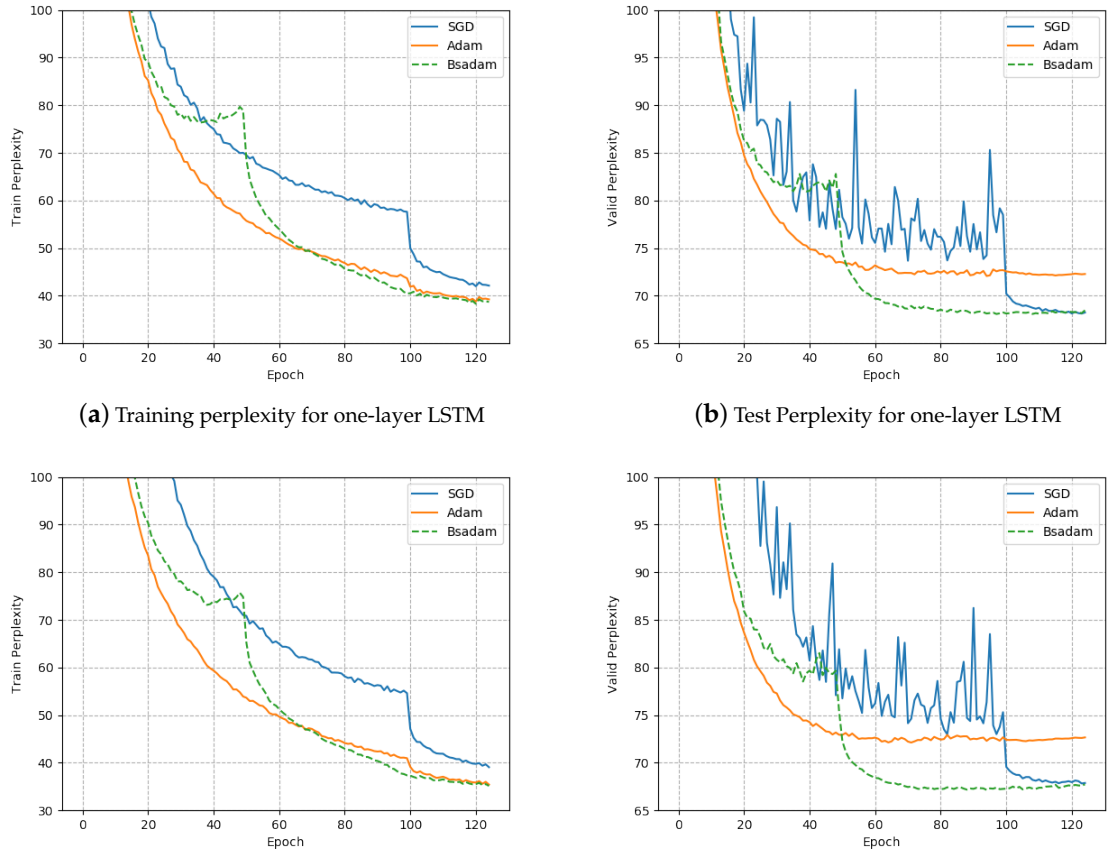
(**l**) Test accuracy for DenseNet-121

Figure 5. Training and test accuracy for ResNet-34, SENet-34 and DenseNet-121 on CIFAR-100.

4.3. Language Modeling

To illustrate the wide applicability of our algorithm, we also conduct experiments with the recurrent network. Specifically, we perform experiments with long short-term memory (LSTM) network [26] on Penn Treebank data set for word-level language modeling tasks. We compare our algorithm with Adam and SGD without the moment. We ran 125 epochs for all the compared algorithms and decay the learning rate by 10 at 100th epoch.

Figure 6 shows the learning curve of each optimization method running on Penn Treebank, which includes training accuracy and test accuracy. We find that the training perplexity of a two-layer LSTM is lower than a one-layer LSTM, but the valid perplexity is almost the same, which indicates that the complexity of the network may weaken the generalization capability of the algorithm. Although Adam achieves a lower perplexity on the training set, the final perplexity on a valid set is relatively high. SGD converges slowly in the early stage on a valid set, but the final perplexity is lower than Adam. Bsadam converges slowly in finding minima phase, but in converging phase, training and valid perplexity both decrease rapidly and the overall convergence speed is faster than SGD. What is more, Bsadam can get a similar or better final perplexity compared to fine-tuned SGD.



(**a**) Training perplexity for one-layer LSTM

(**b**) Test Perplexity for one-layer LSTM

(**c**) Training Perplexity for two-layer LSTM

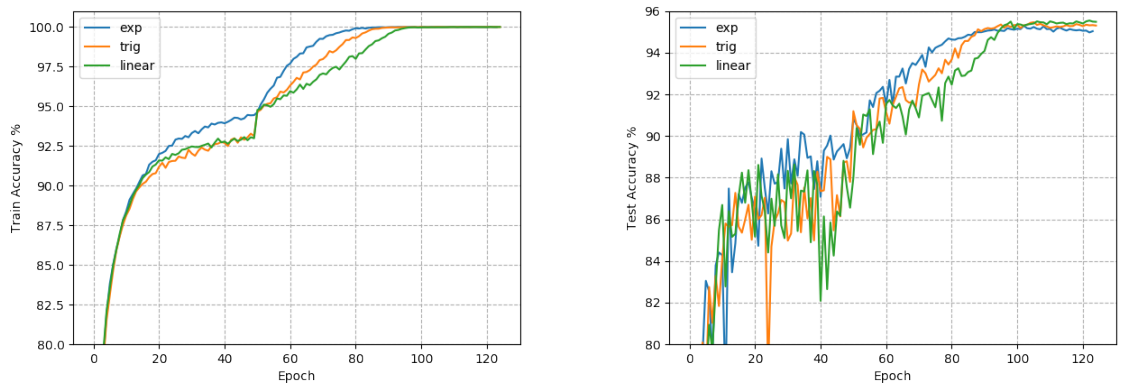
(**d**) Test Perplexity for two-layer LSTM

Figure 6. Training and valid perplexity for long short-term memory (LSTM) with different layers on Penn Treebank.

4.4. Comparison of Different Scheduling Methods

In this paper, we propose three bounded scheduling methods: linear scheduling, exponential scheduling and trigonometric scheduling. We use these three bounded scheduling methods to train SENet-34 on CIFAR-10 and the learning curve is shown in Figure 7. As we can see, these scheduling methods have similar performance, but the details of the learning curve are slightly

different. Exponential scheduling method has the fastest convergence speed among three scheduling methods, but the final test accuracy is lowest. Linear scheduling method has the highest final test accuracy, but the convergence speed is slowest among three scheduling methods.



(**a**) Training Accuracy for SENet-34 (**b**) Test Accuracy for SENet-34

Figure 7. Training and test accuracy of different scheduling methods for SENet-34 on CIFAR-10.

5. Conclusions

Towards the poor generalization capability of adaptive gradient methods in training deep neural networks, a bounded scheduling method, called Bsadam, is proposed in this paper. We first find the upper and lower bound for Adam, then divide the training process into three phases: finding minima phase allows the algorithm to overcome the suboptimal solutions by raising the lower bound of Adam, converging phase ensures the convergence of the algorithm by decreasing the upper bound of Adam and uniform scaling phase allows the algorithm converge to the minimum. We evaluate our algorithm by using simple neural networks, deep convolution networks and recurrent network to perform image classification and language modeling tasks. The experimental results show that our algorithm outperforms SGD(M) and the adaptive gradient methods in convergence speed and accuracy.

Author Contributions: Conceptualization, M.T.; methodology, M.T.; software, M.T.; validation, Z.H., Y.Y. and C.W.; formal analysis, M.T. and Z.H.; investigation, M.T.; resources, Y.P.; data curation, M.T.

Funding: This research was funded by The National Key Research and Development Program of China grant number 2016YFB1000100.

Acknowledgments: All authors thank the referees for their valuable suggestions and help.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
2. Seo, S.; Kim, J. Efficient Weights Quantization of Convolutional Neural Networks Using Kernel Density Estimation based Non-uniform Quantizer. *Appl. Sci.* **2019**, *9*, 2559. [CrossRef]
3. Song, K.; Yang, H.; Yin, Z. Multi-Scale Attention Deep Neural Network for Fast Accurate Object Detection. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, 1. [CrossRef]
4. Maas, A.L.; Qi, P.; Xie, Z.; Hannun, A.Y.; Lengerich, C.T.; Jurafsky, D.; Ng, A.Y. Building DNN acoustic models for large vocabulary speech recognition. *Comput. Speech Lang.* **2017**, *41*, 195–213. [CrossRef]
5. Hinton, G.; Deng, L.; Yu, D.; Dahl, G.; Mohamed, A.R.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Kingsbury, B.; et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.* **2012**, *29*, 82–97. [CrossRef]

6. Violante, M.G.; Marcolin, F.; Vezzetti, E.; Ulrich, L.; Billia, G.; Di Grazia, L. 3D Facial Expression Recognition for Defining Users' Inner Requirements—An Emotional Design Case Study. *Appl. Sci.* **2019**, *9*, 2218. [CrossRef]
7. Zhang, J.; Zong, C. Deep Neural Networks in Machine Translation: An Overview. *IEEE Intell. Syst.* **2015**, *30*, 16–25. [CrossRef]
8. Robbins, H.; Monro, S. A Stochastic Approximation Method. *Ann. Math. Stat.* **1951**, *22*, 400–407. [CrossRef]
9. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2001**, *12*, 2121–2159.
10. Nocedal, J.; Wright, S. *Numerical Optimization*; Springer Science & Business Media: Berlin, Germany, 2006.
11. Smith, L.N. Cyclical learning rates for training neural networks. In Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), Santa Rosa, CA, USA, 24–31 March 2017; pp. 464–472.
12. Smith, L.N.; Topin, N. Super-convergence: Very fast training of neural networks using large learning rates. *Artif. Intell. Mach. Learn. Multi-Domain Oper. Appl.* **2019**, *11006*, 1100612.
13. Luo, L.; Xiong, Y.; Liu, Y.; Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv* **2019**, arXiv:1902.09843.
14. Zeiler, M.D. ADADELTA: An adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701.
15. Tieleman, T.; Geoffrey, H. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* **2012**, *4*, 26–31.
16. Kingma, D.P.; Ba, J.L. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
17. Dozat, T. Incorporating Nesterov Momentum into Adam. *ICLR Workshop* **2016**, *1*, 2013–2016.
18. Reddi, S.J.; Kale, S.; Kumar, S. On the Convergence of Adam and Beyond. *arXiv* **2019**, arXiv:1904.09237.
19. Nitish, S.K.; Richard, S. Improving generalization performance by switching from Adam to SGD. *arXiv* **2017**, arXiv:1712.07628.
20. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
21. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; University of Toronto: Toronto, ON, Canada, 2009.
22. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
23. Iandola, F.; Moskewicz, M.; Karayev, S.; Girshick, R.; Darrell, T.; Keutzer, K. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv* **2014**, arXiv:1404.1869.
24. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 7132–7141.
25. Mitchell, P.M.; Mary, A.M.; Beatrice, S. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.* **1993**, *19*, 313–330.
26. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
27. Roux, N.L.; Schmidt, M.; Bach, F. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. *Adv. Neural Inf. Process. Syst.* **2012**, *4*, 2663–2671.
28. Fletcher, R. On the barzilai-borwein method. In *Optimization and Control with Applications*; Springer: Boston, MA, USA, 2005; pp. 235–256.
29. Raydan, M. On the barzilai and borwein choice of steplength for the gradient method. *IMA J. Numer. Anal.* **1993**, *13*, 321-326. [CrossRef]
30. Massé, P.-Y.; Ollivier, Y. Speed learning on the fly. *arXiv* **2015**, arXiv:1511.02540.
31. Xiangyi, C.; Sijia, L.; Ruoyu, S.; Mingyi, H. On the convergence of a class of Adam-type algorithms for non-convex optimization. *arXiv* **2018**, arXiv:1808.02941.
32. Ashia, C.W.; Rebecca, R.; Mitchell, S.; Nati, S.; Benjamin, R. The marginal value of adaptive gradient methods in machine learning. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 4148–4158.

33. Hardt, M.; Recht, B.; Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv* **2015**, arXiv:1509.01240.
34. Zhang, R. Making convolutional networks shift-invariant again. *arXiv* **2019**, arXiv:1904.11486.