Automatic, Dynamic, and Nearly Optimal Learning Rate Specification by Local Quadratic Approximation

Yingqiu Zhu
rozen0maiden@126.com
School of Statistics, Renmin University of China
Beijing, China

Yu Chen
yu.chen@pku.edu.cn
Guanghua School of Management, Peking University
Beijing, China

Danyang Huang*
dyhuang@ruc.edu.cn
Center for Applied Statistics, Renmin University of China
School of Statistics, Renmin University of China
Beijing, China

Bo Zhang
mabzhang@ruc.edu.cn
Center for Applied Statistics, Renmin University of China
School of Statistics, Renmin University of China
Beijing, China

Hansheng Wang
hansheng@pku.edu.cn
Guanghua School of Management, Peking University
Beijing, China

ABSTRACT

In deep learning tasks, the learning rate determines the update step size in each iteration, which plays a critical role in gradient-based optimization. However, the determination of the appropriate learning rate in practice typically replies on subjective judgement. In this work, we propose a novel optimization method based on local quadratic approximation (LQA). In each update step, given the gradient direction, we locally approximate the loss function by a standard quadratic function of the learning rate. Then, we propose an approximation step to obtain a nearly optimal learning rate in a computationally efficient way. The proposed LQA method has three important features. First, the learning rate is automatically determined in each update step. Second, it is dynamically adjusted according to the current loss function value and the parameter estimates. Third, with the gradient direction fixed, the proposed method leads to nearly the greatest reduction in terms of the loss function. Extensive experiments have been conducted to prove the strengths of the proposed LQA method.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Batch learning*.

KEYWORDS

neural networks, gradient descent, learning rate, machine learning

ACM Reference Format:
Yingqiu Zhu, Yu Chen, Danyang Huang, Bo Zhang, and Hansheng Wang. 2020. Automatic, Dynamic, and Nearly Optimal Learning Rate Specification by Local Quadratic Approximation. In *CIKM '20: 29th ACM International Conference on Information and Knowledge Management, October 19–23, 2020, Galway, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In recent years, the development of deep learning has led to remarkable success in visual recognition [7, 10, 14], speech recognition [8, 29], natural language processing [2, 5], and many other fields. For different learning tasks, researchers have developed different network frameworks, including deep convolutional neural networks [14, 16], recurrent neural networks [6], graph convolutional networks [12] and reinforcement learning [19, 20]. Although the network structure could be totally different, the training methods are typically similar. They are often gradient decent methods, which are developed based on backpropagation [24].

*Corresponding Author

CIKM '20, October 19–23, 2020, Galway, Ireland

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Given a differentiable objective function, gradient descent is a natural and efficient method for optimization. Among various gradient descent methods, the stochastic gradient descent (SGD) method [23] plays a critical role. In the standard SGD method, the first-order gradient of a randomly selected sample is used to iteratively update the parameter estimates of a network. Specifically, the parameter estimates are adjusted with the negative of the random gradient multiplied by a step size. The step size is called the *learning rate*. Many generalized methods based on the SGD method have been proposed [1, 4, 11, 25, 26]. Most of these extensions specify improved update rules to adjust the direction or the step size. However, [1] pointed out that, many hand-designed update rules are designed for circumstances with certain characteristics, such as sparsity or nonconvexity. As a result, rule-based methods might perform well in some cases but poorly in others. Consequently, an optimizer with an automatically adjusted update rule is preferable.

An update rule contains two important components: one is the update direction, and the other is the step size. The learning rate determines the step size, which plays a significant role in optimization. If it is set inappropriately, the parameter estimates could be suboptimal. Empirical experience suggests that a relatively larger learning rate might be preferred in the early stages of the optimization. Otherwise, the algorithm might converge very slowly. In contrast, a relatively smaller learning rate should be used in the later stages. Otherwise, the objective function cannot be fully optimized. This phenomenon inspires us to design a method to automatically search for an optimal learning rate in each update step during optimization.

To this end, we propose here a novel optimization method based on local quadratic approximation (LQA). It tunes the learning rate in a dynamic, automatic and nearly optimal manner. The method can obtain the best step size in each update step. Intuitively, given a search direction, what should be the best step size? One natural definition is the step size that can lead to the greatest reduction in the global loss. Accordingly, the step size itself should be treated as a parameter, that needs to be optimized. For this purpose, the proposed method can be decomposed into two important steps. They are the *expansion* step and the *approximation* step. First, in the *expansion* step, we conduct Taylor expansion on the loss function, around the current parameter estimates. Accordingly, the objective function can be locally approximated by a quadratic function in terms of the learning rate. Then, the learning rate is also treated as a parameter to be optimized, which leads to a nearly optimal determination of the learning rate for this particular update step.

Second, to implement this idea, we need to compute the first- and second-order derivatives of the objective function on the gradient direction. One way to solve this problem is to compute the Hessian matrix for the loss function. However, this solution is computationally expensive. Because many complex deep neural networks involve a large number of parameters, this makes the Hessian matrix have ultra-high dimensionality. To solve this problem, we propose here a novel *approximation* step. Note that, given a fixed gradient direction, the loss function can be approximated by a standard quadratic function with the learning rate as the only input variable. For a univariate quadratic function such as this, there are only two unknown coefficients. They are the linear term coefficient and the quadratic term coefficient. As long as these two coefficients can be determined, the optimal learning rate can be obtained. To estimate the two unknown coefficients, one can try, for example, two different but reasonably small learning rates. Then, the corresponding objective function can be evaluated. This step leads to two equations, which can be solved to estimate the two unknown coefficients in the quadratic approximation function. Thereafter, the optimal learning rate can be obtained.

Our contributions: We propose an automatic, dynamic and nearly optimal learning rate tuning algorithm that has the following three important features.

(1) The algorithm is automatic. In other words, it leads to an optimization method with little subjective judgment.

(2) The method is dynamic in the sense that the learning rate used in each update step is different. It is dynamically adjusted according to the current status of the loss function and the parameter estimates. Typically, larger rates are used in the earlier iterations, while smaller rates are used in the latter iterations.

(3) The learning rate derived from the proposed method is nearly optimal. For each update step, by the novel quadratic approximation, the learning rate leads to almost the greatest reduction in terms of the loss function. Here, "almost" refers to the fact that the loss function is locally approximated by a quadratic function with unknown coefficients numerically estimated. For this particular update step, with the gradient direction fixed, and among all the possible learning rates, the one determined by the proposed method can result in nearly the greatest reduction in terms of the loss function.

The rest of this article is organized as follows. In Section 2, we review related works on gradient-based optimizers. Section 3 presents the proposed algorithm in detail. In Section 4, we verify the performance of the proposed method through empirical studies on open datasets. Then, concluding remarks are given in Section 5.

2 RELATED WORK

To optimize a loss function, two important components need to be specified: the update direction and the step size. Ideally, the best update direction should be the gradient computed for the loss function based on the whole data. For convenience,

we refer to it as the *global gradient*. Since the calculation of the global gradient is computationally expensive, the SGD method [23] uses the gradient estimated based on a stochastic subsample in each iteration, which we referred to as a *sample gradient*. It leads to a fairly satisfactory empirical performance. The SGD method has inspired many new optimization methods, most of which enhance their performance by improving the estimation of the global gradient direction. A natural improvement is to combine sample gradients from different update steps so that a more reliable estimate for the global gradient direction can be obtained. This improvement has led to the momentum-based optimization methods, such as those proposed in [3, 15, 25, 27]. In particular, [3] adopted Nesterov's accelerated gradient algorithm [21] to further improve the calculation of the gradient direction.

There exist other optimization methods that focus on the adjustment of the step size. [4] proposed AdaGrad, in which the step size is iteratively decreased according to a prespecified function. However, it still involves a parameter related to the learning rate, which needs to be subjectively determined. More extensions of AdaGrad have been proposed, such as RMSProp [26] and AdaDelta [30]. Particularly, RMSProp introduced a decay factor to adjust the weights of previous sample gradients. [11] proposed an adaptive moment estimation (Adam) method, that combined RMSProp with a momentum-based method. Accordingly, the step size and the update direction are both adjusted during each iteration. However, because step sizes are adjusted without considering the loss function, the loss reduction obtained for each update step is suboptimal. Thus, the resulting convergence rate can be further improved.

To summarize, most existing optimization methods suffer from one or both of the following two limitations. First, they are not automatic, and human intervention is required. Second, they are suboptimal because the loss reduction achieved in each update step can be further improved. These pioneering researchers inspired us to develop a new method for automatic determination of the learning rate. Ideally, the new method should be automatic with little human intervention. It should be dynamic so that the learning rate used for each update step is particularly selected. Mostly importantly, in each update step, the learning rate determined by the new method should be optimal (or nearly optimal) in terms of the loss reduction, given a fixed update direction.

3 METHODOLOGY

In this section, we first introduce the notations used in this paper and the general formulation of the SGD method. Then, we propose an algorithm based on local quadratic approximation to dynamically search for an optimal learning rate. This results in a new variant of the SGD method.

3.1 Stochastic gradient descent

Assume we have a total of N samples. They are indexed by $1 \le i \le N$ and collected by $\mathcal{S} = \{1, 2, \cdots, N\}$. For each sample, a loss function can be defined as $\ell(X_i; \theta)$, where X_i is the input corresponding to the i-th sample and $\theta \in \mathbb{R}^p$ denotes the parameter. Then the global loss function can be defined as

$$\ell(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(X_i; \theta) = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \ell(X_i; \theta).$$

Ideally, one should optimize $\ell(\theta)$ by a gradient descent algorithm. Assume there are a total of T iterations. Let $\hat{\theta}^{(t)}$ be the parameter estimate obtained in the t-th iteration. Then, the estimate in the next iteration $\hat{\theta}^{(t+1)}$ is given by,

$$\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - \delta \nabla \ell(\hat{\theta}^{(t)}),$$

where δ is the learning rate and $\nabla\ell(\hat{\theta}^{(t)})$ is the gradient of the global loss function $\ell(\theta)$ with respect to θ at $\hat{\theta}^{(t)}$. More specifically, $\nabla\ell(\hat{\theta}^{(t)}) = N^{-1}\sum_{i=1}^{N} \nabla\ell(X_i; \hat{\theta}^{(t)})$, where $\nabla\ell(X_i; \hat{\theta}^{(t)})$ is the gradient of the local loss function for the i-th sample.

Unfortunately, such a straightforward implementation is computationally expensive if the sample size N is relatively large, which is particularly true if the dimensionality of θ is also ultrahigh. To alleviate the computational burden, researchers proposed the idea of SGD. The key idea is to randomly partition the whole sample into a number of nonoverlapping batches. For example, we can write $\mathcal{S} = \cup_{k=1}^{K}\mathcal{S}_k$, where $\mathcal{S}_k$ collects the indices of the samples in the k-th batch. We should have $\mathcal{S}_{k_1} \cap \mathcal{S}_{k_2} = \emptyset$ for any $k_1 \ne k_2$ and $|\mathcal{S}_k| = n$ for any $1 \le k \le K$, where n is a fixed batch size. Next, instead of computing the global gradient $\nabla\ell(\hat{\theta}^{(t)})$, we can replace it by an estimate computed based on the k-th batch. More specifically, each iteration (e.g., the t-th iteration) is further decomposed into a total of K batch steps. Let $\hat{\theta}^{(t,k)}$ be the estimate obtained during the k-th ($1 \le k \le K$) batch step during the t-th iteration. Then, we have

$$\hat{\theta}^{(t,k+1)} = \hat{\theta}^{(t,k)} - \frac{\delta}{n}\sum_{i\in\mathcal{S}_k} \nabla\ell(X_i; \hat{\theta}^{(t,k)}),$$

where $k = 1, \cdots, K-1$. In particular, $\hat{\theta}^{(t+1,1)} = \hat{\theta}^{(t,K)} - \delta n^{-1}\sum_{i\in\mathcal{S}_K} \nabla\ell(X_i; \hat{\theta}^{(t,K)})$.

By doing so, the computational burden can be alleviated. However, the tradeoff is that the batch-sample-based gradient estimate could be unstable, which is particularly true if the batch size n is relatively small. To fix this problem, various momentum-based methods have been proposed. The key idea is to record gradients in previous iterations and integrate them together to form a more stable estimate.

3.2 Local quadratic approximation

In this work, we assume that for each batch step, the estimate for the gradient direction is given. It can be obtained by different algorithms. For example, it could be the estimate obtained by a standard SGD algorithm or an estimate that involves rule-based corrections, such as that from a momentum-based method. We focus on how to specify the learning rate in an optimal (or, nearly optimal) way.

To this end, we treat the learning rate δ as an unknown parameter. It is remarkable that the optimal learning rate could dynamically change in different batch steps. Thus, we use $\delta_{t,k}$ to denote the learning rate in the k-th batch step within the t-th iteration. Since the reduction in the loss in this batch step is influenced by the learning rate $\delta_{t,k}$, we express it as a function of the learning rate $\Delta\ell(\delta_{t,k})$.

To find the optimal value for $\delta_{t,k}$, we investigate the optimization of $\Delta\ell(\delta_{t,k})$ based on the Taylor expansion. For simplicity, we use $g_{t,k} = n^{-1}\sum_{i\in\mathcal{S}_k}\nabla\ell(X_i;\hat{\theta}^{(t,k)})$ to denote the current gradient. Given $\hat{\theta}^{(t,k)}$ and $g_{t,k}$, the loss reduction could be expressed as

$$\begin{aligned}\Delta\ell(\delta_{t,k}) &= \frac{1}{n}\sum_{i\in\mathcal{S}_k}\left[\ell\left(X_i;\hat{\theta}^{(t,k+1)}\right) - \ell\left(X_i;\hat{\theta}^{(t,k)}\right)\right]\\ &= \frac{1}{n}\sum_{i\in\mathcal{S}_k}\left[\ell\left(X_i;\hat{\theta}^{(t,k)} - \delta_{t,k}g_{t,k}\right) - \ell\left(X_i;\hat{\theta}^{(t,k)}\right)\right]\end{aligned}$$

Then, two estimation steps are conducted to determine an appropriate value for $\delta_{t,k}$ in this batch step.

(1) Expansion Step. By a Taylor expansion of $\ell(X_i;\theta)$ around $\hat{\theta}^{(t,k)}$, we have $\ell\left(X_i;\hat{\theta}^{(t,k)} - \delta_{t,k}g_{t,k}\right) =$

$$\begin{aligned}&\ell\left(X_i;\hat{\theta}^{(t,k)}\right) - \nabla\ell\left(X_i;\hat{\theta}^{(t,k)}\right)\delta_{t,k}g_{t,k}\\ &+\frac{1}{2}\delta_{t,k}^2 g_{t,k}^T\nabla^2\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k} + o\left(\delta_{t,k}^2 g_{t,k}^T g_{t,k}\right),\end{aligned}$$

where $\nabla\ell\left(X_i;\hat{\theta}^{(t,k)}\right)$ and $\nabla^2\ell\left(X_i;\hat{\theta}^{(t,k)}\right)$ denote the first- and second-order derivatives of the local loss function, respectively. As a result, the reduction is

$$\begin{aligned}\Delta\ell(\delta_{t,k}) &= \frac{1}{n}\sum_{i\in\mathcal{S}_k}\left[-\nabla\ell\left(X_i;\hat{\theta}^{(t,k)}\right)\delta_{t,k}g_{t,k}\right.\\ &\left.+\frac{1}{2}\delta_{t,k}^2 g_{t,k}^T\nabla^2\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k} + o\left(\delta_{t,k}^2 g_{t,k}^T g_{t,k}\right)\right]\\ &= -\left\{\frac{1}{n}\sum_{i\in\mathcal{S}_k}\nabla\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k}\right\}\delta_{t,k}\\ &+\left\{\frac{1}{2n}\sum_{i\in\mathcal{S}_k} g_{t,k}^T\nabla^2\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k}\right\}\delta_{t,k}^2\\ &+ o\left(n^{-1}\delta_{t,k}^2 g_{t,k}^T g_{t,k}\right). \qquad (1)\end{aligned}$$

According to (1), $\Delta\ell(\delta_{t,k})$ is a quadratic function of $\delta_{t,k}$. For simplicity, the coefficient of the linear term and the coefficient of the quadratic term are denoted as

$$\begin{aligned}a_{t,k} &= \frac{1}{n}\left\{\sum_{i\in\mathcal{S}_k}\nabla\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k}\right\}, \text{ and}\\ b_{t,k} &= \left\{\frac{1}{2n}\sum_{i\in\mathcal{S}_k} g_{t,k}^T\nabla^2\ell\left(X_i;\hat{\theta}^{(t,k)}\right)g_{t,k}\right\},\end{aligned}$$

respectively. Since the Taylor remainder here could be negligible, (1) can be simply denoted by

$$\Delta\ell(\delta_{t,k}) \approx -a_{t,k}\delta_{t,k} + b_{t,k}\delta_{t,k}^2. \qquad (2)$$

To maximize $\Delta\ell(\delta_{t,k})$ with respect to $\delta_{t,k}$, we take the corresponding derivative of the loss reduction, which leads to,

$$\frac{\partial\Delta\ell(\delta_{t,k})}{\partial\delta_{t,k}} \approx -a_{t,k} + 2b_{t,k}\delta_{t,k} = 0.$$

As a result, the optimal learning rate in this batch step can be approximated by,

$$\delta_{t,k}^* = (2b_{t,k})^{-1}a_{t,k}. \qquad (3)$$

Note that the computation of $b_{t,k}$ involves the first- and second-order derivatives. For a general form of loss function, this calculation may be computationally expensive in real applications. Thus, an approximation step is preferred to improve the computational efficiency.

(2) Approximation Step. To compute the coefficients $a_{t,k}$ and $b_{t,k}$ and avoid the computation of second derivatives, we consider the following approximation method. The basic idea is to build 2 equations with respect to the 2 unknown coefficients.

Let $g_{t,k}$ be a given estimate of the gradient direction. We then compute

$$\begin{aligned}\sum_{i\in\mathcal{S}_k}\ell\left(X_i;\hat{\theta}^{(t,k)} - \delta_0 g_{t,k}\right) &= \sum_{i\in\mathcal{S}_k}\ell\left(X_i;\hat{\theta}^{(t,k)}\right) - a_{t,k}\delta_0 n\\ &\quad + b_{t,k}\delta_0^2 n, \qquad (4)\\ \sum_{i\in\mathcal{S}_k}\ell\left(X_i;\hat{\theta}^{(t,k)} + \delta_0 g_{t,k}\right) &= \sum_{i\in\mathcal{S}_k}\ell\left(X_i;\hat{\theta}^{(t,k)}\right) + a_{t,k}\delta_0 n\\ &\quad + b_{t,k}\delta_0^2 n, \qquad (5)\end{aligned}$$

for a reasonably small learning rate δ_0. A natural choice for δ_0 could be $\delta_{t,k-1}^*$ if $k > 1$ and $\delta_{t-1,K}^*$ if $k = 1$. By solving (4)

and (5), we have

$$\tilde{b}_{t,k} = \frac{1}{2n\delta_0^2} \sum_{i \in \mathcal{S}_k} \left[\ell\left(X_i; \hat{\theta}^{(t,k)} + \delta_0 g_{t,k}\right) + \ell\left(X_i; \hat{\theta}^{(t,k)} - \delta_0 g_{t,k}\right) - 2\ell\left(X_i; \hat{\theta}^{(t,k)}\right) \right], \quad (6)$$

$$\tilde{a}_{t,k} = \frac{1}{2n\delta_0} \sum_{i \in \mathcal{S}_k} \left[\ell\left(X_i; \hat{\theta}^{(t,k)} + \delta_0 g_{t,k}\right) - \ell\left(X_i; \hat{\theta}^{(t,k)} - \delta_0 g_{t,k}\right) \right], \quad (7)$$

where $\tilde{a}_{t,k}$ and $\tilde{b}_{t,k}$ could serve as the approximations of $a_{t,k}$ and $b_{t,k}$, respectively. Then, we apply these results back to (3), which gives the approximated optimal learning rate $\hat{\delta}^*_{t,k}$. Because $\hat{\delta}^*_{t,k}$ is optimally selected, the reduction in the loss function is nearly optimal for each batch step. As a consequence, the total number of iterations required for convergence can be much reduced, which makes the whole algorithm converge much faster than usual. In summary, **Algorithm 1** illustrates the pseudocode of the proposed method.

Algorithm 1 Local quadratic approximation algorithm

Require: T: number of iterations; K: number of batches within one iteration; $\ell(\theta)$: loss function with parameters θ; θ_0: initial estimate for parameters (e.g., a zero vector); δ_0: initial (small) learning rate.
$t \leftarrow 1$;
$\hat{\theta}^{(1,1)} \leftarrow \theta_0$;
while $t \leq T$ **do**
  $k \leftarrow 1$;
  while $k \leq K - 1$ **do**
    Compute the gradient $g_{t,k}$;
    Compute $\tilde{a}_{t,k}$ and $\tilde{b}_{t,k}$ according to (6) and (7);
    $\hat{\delta}^*_{t,k} \leftarrow (2\tilde{b}_{t,k})^{-1}\tilde{a}_{t,k}$;
    $\hat{\theta}^{(t,k+1)} \leftarrow \hat{\theta}^{(t,k)} - \hat{\delta}^*_{t,k} g_{t,k}$;
    $k \leftarrow k + 1$;
  end while
  Compute the gradient $g_{t,K}$;
  Compute $\tilde{a}_{t,K}$ and $\tilde{b}_{t,K}$ according to (6) and (7);
  $\hat{\delta}^*_{t,K} \leftarrow \left(2\tilde{b}_{t,K}\right)^{-1} \tilde{a}_{t,K}$;
  $\hat{\theta}^{(t+1,1)} \leftarrow \hat{\theta}^{(t,K)} - \hat{\delta}^*_{t,K} g_{t,K}$;
  $t \leftarrow t + 1$;
end while
return $\hat{\theta}^{(T+1,1)}$, the resulting estimate.

It is remarkable that the computational cost required for calculating the optimal learning rate is ignorable. The main cost is due to the calculation of the loss function values (not its derivatives) at two different points. The cost of this step is substantially smaller than computing the gradient and it is particularly true if the unknown parameter θ's dimension is ultrahigh.

4 EXPERIMENTS

In this section, we empirically evaluate the proposed method based on different models and compare it with various optimizers under different parameter settings. The details are listed as follows.

Classification Model. To demonstrate the robustness of the proposed method, we consider three classic models. They are multinomial logistic regression, multilayer perceptron (MLP), and deep convolutional neural network (CNN) models.

Competing Optimizers. For comparison purposes, we compare the proposed LQA method with other popular optimizers. They are the standard SGD method, the SGD method with momentum, the SGD method based on Nesterov's accelerated gradient (NAG), AdaGrad, RMSProp and Adam. For simplicity, we use "SGD-M" to denote the SGD method with momentum and "SGD-NAG" to denote the SGD method based on the NAG algorithm.

Parameter Settings. For the competing optimizers, we adopt three different learning rates, $\delta = 0.1, 0.01$, and 0.001. For all the optimizers, the minibatch size is 64, and the initial values for all the parameters are zero. If there are other hyperparameters (e.g., decay rates) in the models, they are set by default.

Performance Measurement. To gauge the performance of the optimizers, we report the *training loss* of the different optimizers, which is defined as the negative log-likelihood function. The results of different optimizers in each iteration are shown in figures for comparison purposes.

4.1 Multinomial Logistic Regression

We first compare the performance of different optimizers based on multinomial logistic regression. It has a convex objective function. We consider the MNIST dataset [17] for illustration. The dataset consists of a total of 70,000 (28×28) images of handwritten digits, each of which corresponds to a 10-dimensional one-hot vector as its label. Then, the images are flattened as 784-dimensional vectors to train the logistic regression classifier. Figure 1 displays the performance of the proposed LQA method against the competing optimizers. We can draw the following conclusions.

Learning Rate. For the competing optimizers, the training loss curves of different learning rates clearly have different shapes, which means different convergence speeds because the convergence speed is greatly affected by δ. The best δ in this case is 0.01 for the SGD, SGD-M, SGD-NAG and AdaGrad methods. However, that for RMSProp and Adam is 0.001. Note that for RMSProp and Adam, the loss may

fail to converge with an inappropriate δ (e.g., $\delta = 0.1$). It is remarkable that with the LQA method, the learning rate is automatically determined and dynamically updated. Thus, the proposed method can provide an automatic solution for the training of multinomial logistic regression classifiers. Next, we compare LQA and the competing optimizers with their best learning rates.

Loss Reduction. First, the loss curve of LQA remains lower than those of the SGD optimizers during the whole training process. This finding means that LQA converges faster than the SGD optimizers. For example, LQA reduces the loss to 0.256 in the first 10 iterations. It takes 40 iterations for the standard SGD optimizer with $\delta = 0.1$ to achieve the same level. Second, for AdaGrad and RMSProp with $\delta = 0.1$, although the training loss curves are slightly lower than that of LQA in the early stages (e.g. the first 5 iterations), LQA performs better in the later stages. Third, the best performance for Adam in this case is achieved when $\delta = 0.001$. Although the performances are quite similar for LQA and Adam with $\delta = 0.001$, LQA has lower loss values in the early stages.

4.2 Multilayer Perception

MLP models are powerful neural network models and have been widely used in machine learning tasks [22]. They contain multiple fully connected layers and activation functions between those layers. An MLP can approximate arbitrary continuous functions over compact input sets [9].

To investigate the performance of the proposed method in this case, we consider the MNIST dataset. Following the model setting in [11], the MLP is built with 2 fully connected hidden layers, each of which has 1,000 units, and the ReLU function is adopted as the activation function. Figure 2 shows the performances of different optimizers. The following conclusions can be drawn.

Learning Rate. In this case, the best learning rates for the competing methods are quite different: (1) for the standard SGD method, the best learning rate is $\delta = 0.1$; (2) for the SGD-M, SGD-NAG and AdaGrad methods, the best learning rate is 0.01; (3) for RMSProp and Adam, $\delta = 0.001$ is the best. It is remarkable that even for the same optimizer, different learning rates could lead to a different performance, if the model changes. Thus, determining the appropriate learning rate in practice may depend on expert experience and subjective judgement. In contrast, the proposed method can avoid such effort in choosing δ and give a comparable and robust performance.

Loss Reduction. First, compared with the standard SGD optimizers, LQA performs much better, which could be seen from the lower training loss curve. Second, compared with the SGD-M, SGD-NAG, RMSProp and Adam optimizers, the performance of LQA is comparable to their best performances in the early stages (e.g., the first 5 iterations). In the later stages, the LQA method continues to reduce the loss, which makes the training loss curve of LQA lower than that of the other methods. For example, the smallest loss corresponding to the Adam optimizer in the 20th iteration is 0.011, while that of the LQA method is 0.002. Third, although AdaGrad converges faster when δ =0.01, the performance of the proposed method is slightly better than AdaGrad after the 12th iteration.

4.3 Deep Convolutional Neural Networks

CNNs have brought remarkable breakthroughs in computer vision tasks over the past two decades [7, 10, 14] and play a critical role in various industrial applications, such as face recognition [28] and driverless vehicles [18]. In this subsection, we investigate the performance of the LQA method with respect to the training of CNNs. Two classic CNNs are considered. They are LeNet [17] and ResNet [7]. More specifically, LeNet-5 and ResNet-18 are studied in this paper. The MNIST and CIFAR10 [13] datasets are used to demonstrate the performance. The CIFAR10 dataset contains 60,000 (32×32) RGB images, which are divided into 10 classes.

LeNet. Figure 3 and Figure 4 show the results of experiments on the MNIST and the CIFAR10 datasets, respectively. The following conclusions can be drawn: (1) For both datasets, the loss curves of the LQA method remain lower than those of the standard SGD and AdaGrad optimizers. This finding suggests LQA converges faster than those optimizers during the whole training process. (2) LQA performs similarly to the SGD-M, SGD-NAG, RMSProp and Adam optimizers in the early stages (e.g., the first 20 iterations). However, in the later stages, the proposed method can further reduce the loss and lead to a lower loss than those optimizers after the same number of iterations. (3) For the CIFAR10 dataset, a large δ (e.g., $\delta = 0.1$) may lead to an unstable loss curve for a standard SGD optimizer. Although the loss curve of LQA is unstable in the early stages of training, it becomes smooth in the later stages because the proposed method is able to automatically and adaptively adjust the update step size to accelerate training. It is fairly robust.

ResNet. Figure 5 displays the training loss of ResNet-18 corresponding to different optimizers on the CIFAR10 dataset. Accordingly, we make the following conclusions. First, the LQA method performs similarly to the other optimizers in the early stages of training (e.g., the first 15 iterations). However, it converges faster in the later stages. Particularly, the proposed method leads to a lower loss than RMSProp and Adam within the same number of iterations. Second, in this case, the loss curves of the SGD optimizers and AdaGrad are quite unstable during the whole training period. The LQA method is much more stable in the later stages of the training than early stages.

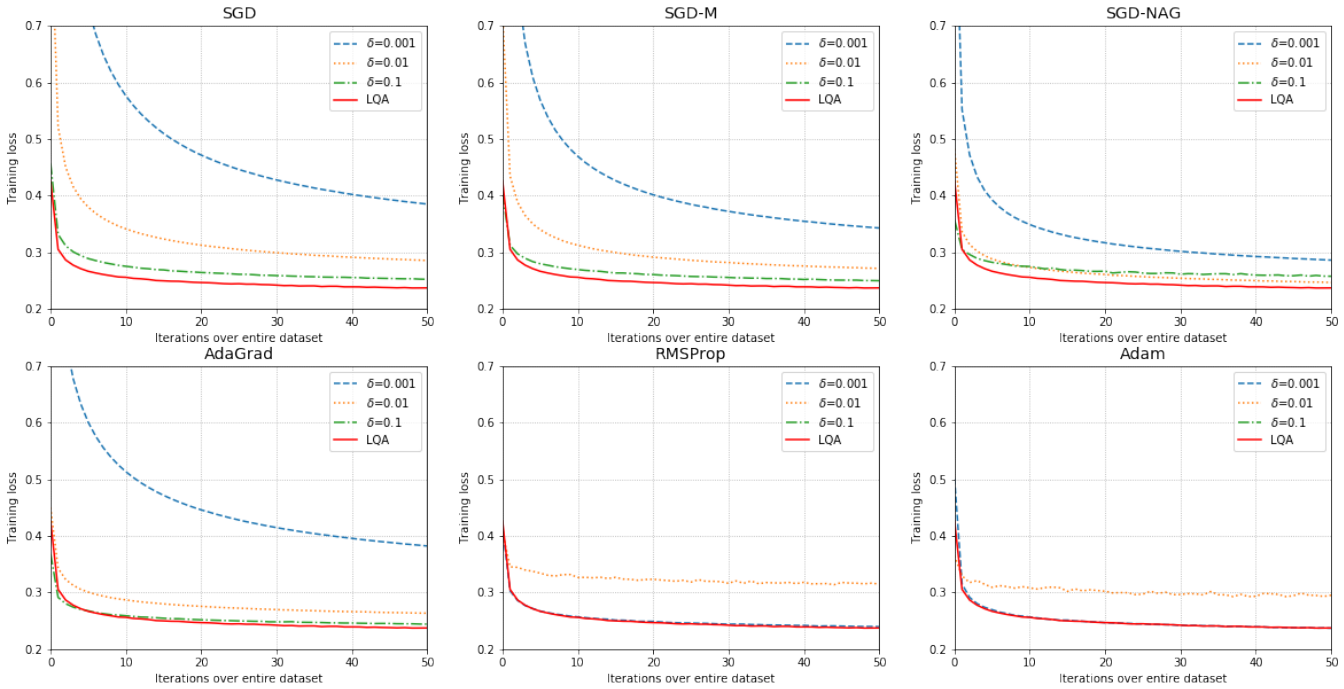

Figure 1: Training loss of multinomial logistic regression on the MNIST dataset.

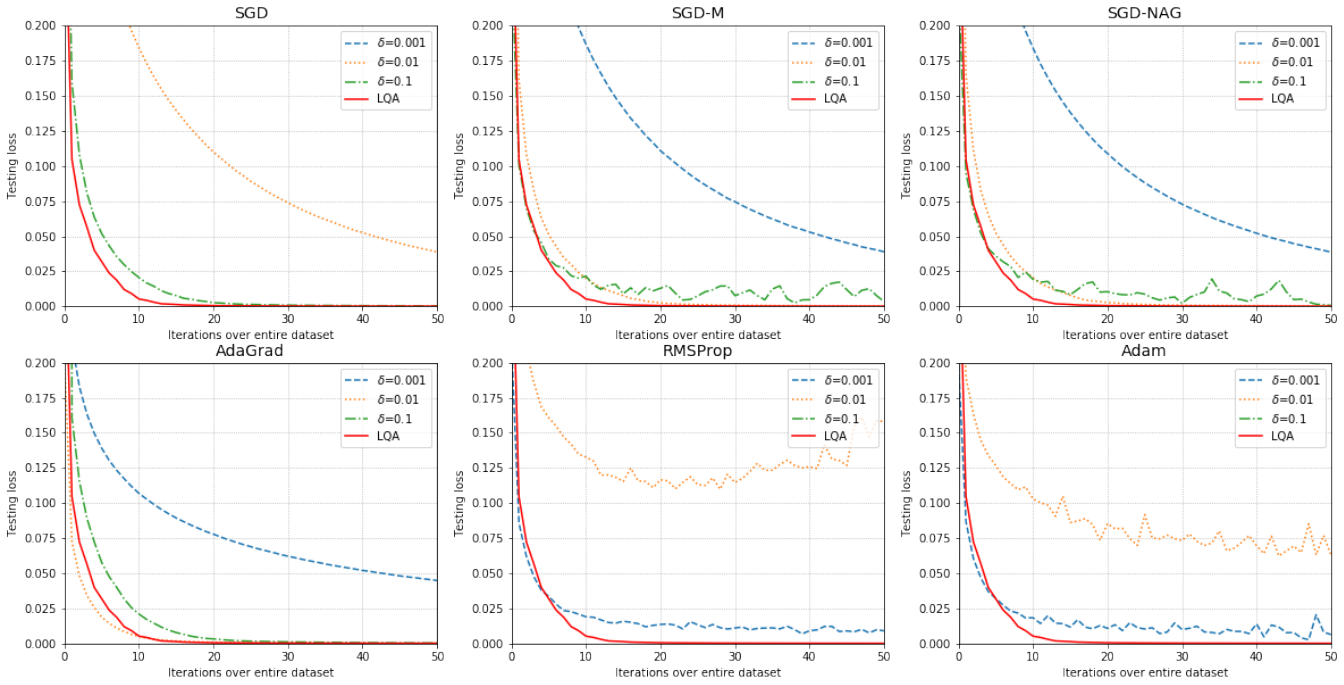

Figure 2: Training loss of MLP on the MNIST dataset.

5 CONCLUSIONS

In this work, we propose LQA, a novel approach to determine the nearly optimal learning rate for automatic optimization.

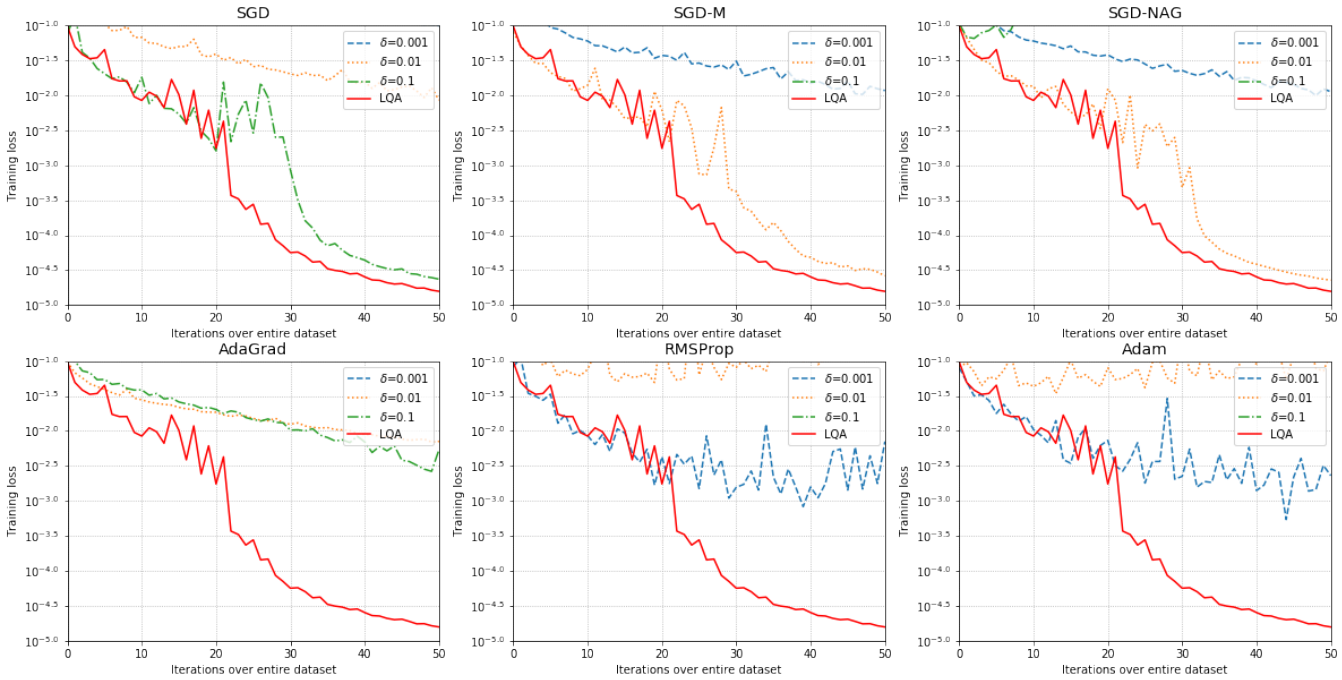

Figure 3: Training loss of LeNet-5 on the MNIST dataset.

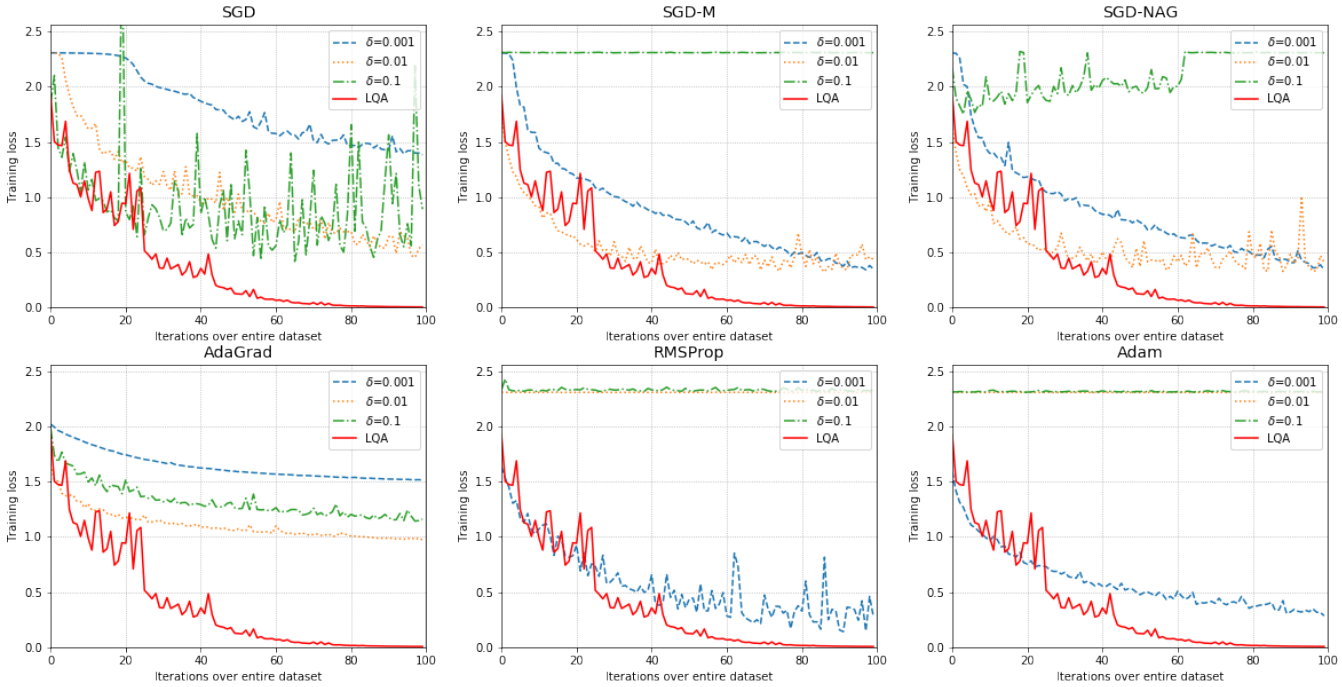

Figure 4: Training loss of LeNet-5 on the CIFAR10 dataset.

Our method has three important features. First, the learning rate is automatically estimated in each update step. Second, it is dynamically adjusted during the whole training process. Third, given the gradient direction, the learning rate

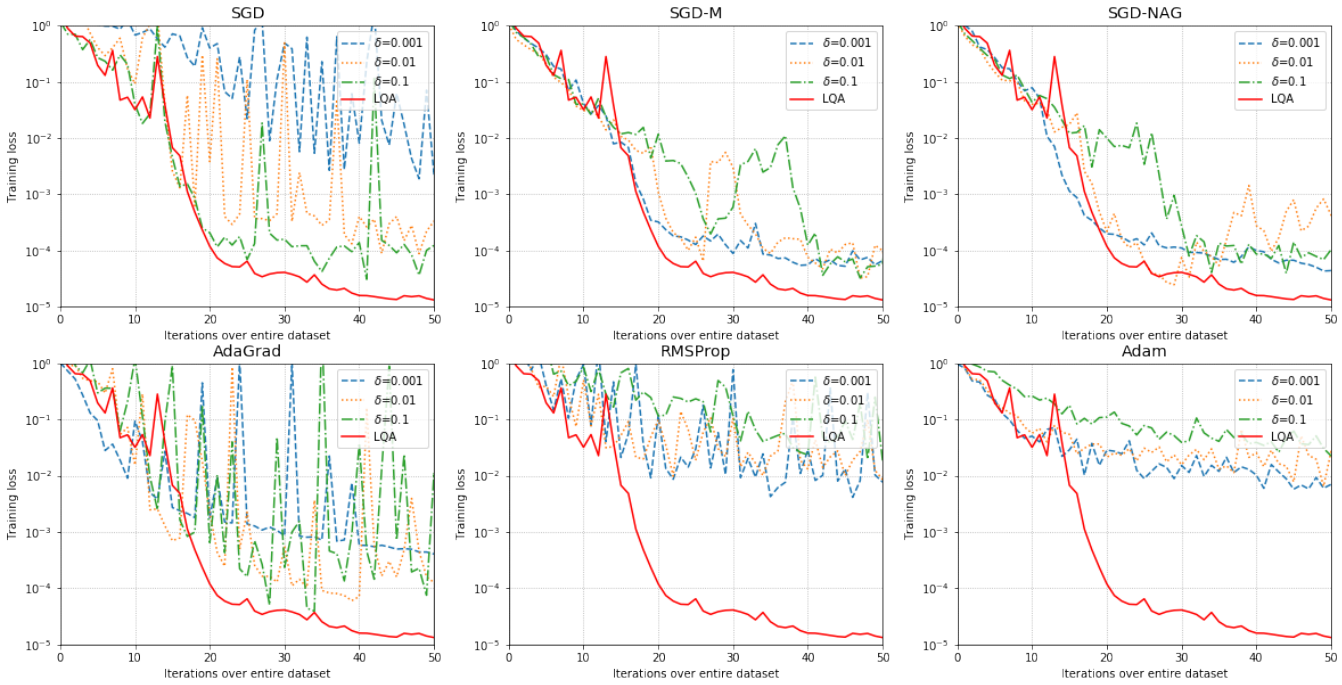

Figure 5: Training loss of ResNet-18 on the CIFAR10 dataset.

leads to nearly the greatest reduction in the loss function. Experiments on openly available datasets demonstrate its effectiveness.

We discuss two interesting topics for future research. First, the optimal learning rate derived by LQA is shared by all dimensions of the parameter estimate. A potential extension is to allow for different optimal learning rates for different dimensions. Second, in this paper, we focus on accelerating the training of the network models. We do not discuss the overfitting issue and sparsity of the gradients. To further improve the performance of the proposed method, it is possible to combine dropouts or sparsity penalties with LQA.

REFERENCES

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*. 3981–3989.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. 2011. Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems*. 1647–1655.

[4] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, Jul (2011), 2121–2159.

[5] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.

[9] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[11] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations 2015*.

[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[13] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Guanghui Lan. 2012. An optimal method for stochastic composite optimization. *Mathematical Programming* 133, 1-2 (2012), 365–397.

[16] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[18] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. 2019. Stereo r-cnn based 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7644–7652.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[21] Yu Nesterov. 1983. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, Vol. 27. 372––376.

[22] Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. 2016. Multilayer Perceptron: Architecture Optimization and Training. *International Journal of Interactive Multimedia and Artificial Intelligence* 4, 1 (2016), 26–30.

[23] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[24] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. 1995. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications* (1995), 1–34.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.

[26] Tijmen Tieleman and Geoffrey Hinton. 2012. *RMSProp, COURSERA: Neural networks for machine learning*. Technical Report.

[27] Paul Tseng. 1998. An incremental gradient (-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization* 8, 2 (1998), 506–531.

[28] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. 2016. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*. Springer, 499–515.

[29] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256* (2016).

[30] Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).