Reinforcement Learning for Learning Rate Control

Chang Xu[1]**,Tao Qin**[2]**,Gang Wang**[1]**,Tie-Yan Liu**[2]
[1]College of Computer and Control Engineering, Nankai University, Tianjin, China
[2]Microsoft Research, Beijing, China
[1]{changxu, wgzwp}@nbjl.nankai.edu.cn, [2]{taoqin, tie-yan Liu}@microsoft.com

Abstract

Stochastic gradient descent (SGD), which updates the model parameters by adding a local gradient times a learning rate at each step, is widely used in model training of machine learning algorithms such as neural networks. It is observed that the models trained by SGD are sensitive to learning rates and good learning rates are problem specific. We propose an algorithm to automatically learn learning rates using neural network based actor-critic methods from deep reinforcement learning (RL). In particular, we train a policy network called actor to decide the learning rate at each step during training, and a value network called critic to give feedback about quality of the decision (e.g., the goodness of the learning rate outputted by the actor) that the actor made. The introduction of auxiliary actor and critic networks helps the main network achieve better performance. Experiments on different datasets and network architectures show that our approach leads to better convergence of SGD than human-designed competitors.

1 Introduction

While facing large scale of training data, stochastic learning such as stochastic gradient descent (SGD) is usually much faster than batch learning and often results in better models. An observation for SGD methods is that their performances are highly sensitive to the choice of learning rate [LeCun *et al.*, 2012]. Clearly, setting a static learning rate for the whole training process is insufficient, since intuitively the learning rate should decrease when the model becomes more and more close to a (local) optimum as the training goes on over time [Maclaurin *et al.*, 2015]. Although there are some empirical suggestions to guide how to adjust the learning rate over time in training, it is still a difficult task to find a good policy to adjust the learning rate, given that good policies are problem specific and depend on implementation details of a machine learning algorithm. One usually needs to try many times and adjust the learning rate manually to accumulate knowledge about the problem. However, human involvement often needs domain knowledge about the target problems, which is inefficient and difficult to scale up to different problems. Thus, a natural question arises: can we learn to adjust the learning rate? This is exactly the focus of this work and we aim to learn learning rates for SGD based machine learning (ML) algorithms without human-designed rules or hand-crafted features.

By examining the current practice of learning rate control/adjustment, we have two observations. First, learning rate control is a sequential decision process: We set an initial learning rate at the beginning, and then at each step we decide whether to change the learning rate and how to change it, based on the current model and loss, training data at hand, and maybe history of the training process. As suggested in [Orr and Müller, 2003], one well-principled method for estimating the ideal learning rate that is to decrease the learning rate when the weight vector oscillates, and increase it when the weight vector follows a relatively steady direction. Second, although at each step some immediate reward (e.g., the loss decrement) can be obtained by taking actions, we care more about the performance of the final model found by the ML algorithm. Consider two different learning rate control policies: the first one leads to fast loss decrease at the beginning but gets saturated and stuck in a local minimum quickly, while the second one starts with slower loss decrease but results in much smaller final loss. Obviously, the second policy is better. That is, we prefer long-term rewards over short-term rewards.

Combining the two observations, it is not difficult to see that the problem of finding a good policy to control/adjust learning rate falls into the scope of reinforcement learning (RL) [Sutton and Barto, 1998]. Inspired by the recent success of RL for sequential decision problems, in this work, we leverage RL techniques and try to learn learning rate for SGD based methods.

We propose an algorithm to learn learning rate within the actor-critic framework [Sutton, 1984; Sutton *et al.*, 1999; Barto *et al.*, 1983; Silver *et al.*, 2014], which is widely used in RL. An actor network is trained to take an action that decides the learning rate for current step, and a critic network is trained to give feedback to the actor network about long-term performance and help the actor network to adjust itself so as to perform better in the future. To reduce oscillation during training, we take gradient disagreement among training samples into account. By feeding different training samples to the actor network and the critic network, learning rate is en-

couraged to be small when gradients oscillate, which is consistent with the suggestion for ideal learning rate strategy in [Orr and Müller, 2003]. A series of experiments on different datasets and network architectures validate the effectiveness of our proposed algorithm for learning rate control.

The main contributions of this paper include:

- We propose to use actor-critic based auxiliary networks to learn and control the learning rate for ML algorithms, so that the ML algorithm can achieve better convergence.
- Long-term rewards are exploited in our approach rather than only immediate rewards (e.g., the decrease of loss for one step). The expected total decrease of loss in future steps is modeled by the critic network, so that the actor can make far-sighted decision for learning rate.

2 Related Work

We review some related work in this section.

2.1 Improved Gradient Methods

Our focus is to improve gradient based ML algorithm through automatic learning of learning rate. Different approaches have been proposed to improve gradient methods, especially for deep neural networks.

Since SGD solely rely on a given example (or a mini-batch of examples) to compare gradient, its model update at each step tends to be unstable and it takes many steps to converge. To solve this problem, momentum SGD [Jacobs, 1988] is proposed to accelerate SGD by using recent gradients. RMSprop [Tieleman and Hinton, 2012] utilizes the magnitude of recent gradients to normalize the gradients. It always keeps a moving average over the root mean squared gradients, by which it divides the current gradient. Adagrad [Duchi *et al.*, 2011] adapts component-wise learning rates, and performs larger updates for infrequent and smaller updates for frequent parameters. Adadelta [Zeiler, 2012] extends Adagrad by reducing its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size. Adam [Kingma and Ba, 2014] computes component-wise learning rates using the estimates of first and second moments of the gradients, which combines the advantages of AdaGrad and RMSProp.

[Senior *et al.*, 2013; Sutton, 1992; Darken and Moody, 1990] focus on predefining update rules to adjust learning rates during training. A limitation of these methods is that they have additional free parameters which need to be set manually. [Schaul *et al.*, 2013] proposes a method to choose good learning rate for SGD, which relies on the square norm of the expectation of the gradient, and the expectation of the square norm of the gradient. The method is much more constrained than ours and several assumption should be met. Another recent work [Daniel *et al.*, 2016] investigates several hand-tuned features and uses the Relative Entropy Policy Search method as controller to select step size for SGD and RMSprop.

2.2 Reinforcement Learning

Since our proposed algorithm is based on RL techniques, here we give a very brief introduction to RL, which will ease the description of our algorithm in next section.

Reinforcement learning [Sutton, 1988] is concerned with how an agent acts in a stochastic environment by sequentially choosing actions over a sequence of time steps, in order to maximize a cumulative reward. In RL, a state s^t encodes the agent's observation about the environment at a time step t, and a policy function $\pi(s^t)$ determines how the agent behaves (e.g., which action to take) at state s^t. An action-value function (or, Q function) $Q_\pi(s^t, a^t)$ is usually used to denote the cumulative reward of taking action a^t at state s^t and then following policy π afterwards.

Many RL algorithms have been proposed [Sutton and Barto, 1998; Watkins and Dayan, 1992], and many RL algorithms [Sutton, 1984; Sutton *et al.*, 1999; Barto *et al.*, 1983; Silver *et al.*, 2014] can be described under the actor-critic framework. An actor-critic algorithm learns the policy function and the value function simultaneously and interactively. The policy structure is known as the actor, and is used to select actions; the estimated value function is known as the critic, and it criticizes the actions made by the actor.

Recently, deep reinforcement learning, which uses deep neural networks to approximate/represent the policy function and/or the value function, have shown promise in various domains, including Atari games [Mnih *et al.*, 2015], Go [Silver *et al.*, 2016], machine translation [Bahdanau *et al.*, 2016], image recognition [Xu *et al.*, 2015], etc.

3 Method

In this section, we present an actor-critic algorithm that can automate the learning rate control for SGD based machine learning algorithms.

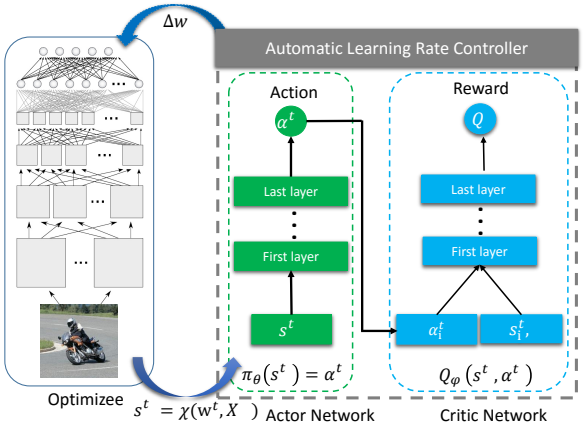


Figure 1: The framework of our proposed automatic learning rate controller.

Many machine learning tasks need to train a model with parameters ω by minimizing a loss function f defined over a set X of training examples:

$$\omega^* = \arg\min_{\omega} f_\omega(X). \quad (1)$$

A standard approach for the loss function minimization is gradient descent, which sequentially updates the parameters using gradients step by step:

$$\omega^{t+1} = \omega^t - a^t \nabla f^t, \tag{2}$$

where a^t is the learning rate at step t, and ∇f^t is the local gradient of f at ω^t. Here one step can be the whole batch of all the training data, a mini batch of tens/hundreds of examples, or a random sample.

It is observed that the performance of SGD based methods is quite sensitive to the choice of a^t for non-convex loss function f. Unfortunately, f is usually non-convex with respect to the parameters w in many ML algorithms, especially for deep neural networks. We aim to learn a learning rate controller using RL techniques that can automatically control a^t.

Figure 1 illustrates our automatic learning rate controller, which adopts the actor-critic framework in RL. The basic idea is that at each step, given the current model ω^t and training sample x, an actor network is used to take an action (the learning rate a^t, and it will be used to update the model ω^t), and a critic network is used to estimate the goodness of the action. The actor network will be updated using the estimated goodness of a^t, and the critic network will be updated by minimizing temporal difference (TD) [Sutton and Barto, 1990] error.

We describe the details of our algorithm in the following subsections.

3.1 Actor Network

The actor network, which is called policy network in RL, plays the key role in our algorithm: it determines the learning rate control policy for the primary ML algorithm[1] based on the current model, training data, and maybe historical information during the training process.

Note that ω^t could be of huge dimensions, e.g., one widely used image recognition model VGGNet [Simonyan and Zisserman, 2014] has more than 140 million parameters. If the actor network takes all of those parameters as the inputs, its computational complexity would dominate the complexity of the primary algorithm, which is unfordable. Therefore, we propose to use a function $\chi(\cdot)$ to process and yield a compact vector s^t as the input of the actor network. Following the practice in RL, we call $\chi(\cdot)$ the state function, which takes ω^t and the training data x as inputs:

$$s^t = \chi(\omega^t, X). \tag{3}$$

Then the actor network $\pi_\theta(\cdot)$ parameterized by θ yields an action a^t:

$$\pi_\theta(s^t) = a^t, \tag{4}$$

where the action $a^t \in \mathbb{R}$ is a continuous value. When a^t is determined, we update the model of the primary algorithm by Equation 2.

Note that the actor network has its own parameters and we need to learn them to output a good action. To learn the actor network, we need to know how to evaluate the goodness of an actor network. The critic network exactly plays this role.

[1]Here we have two learning algorithms. We call the one with learning rate to adjust as the primary ML algorithm, and the other one which optimizes the learning rate of the primary one as the secondary ML algorithm.

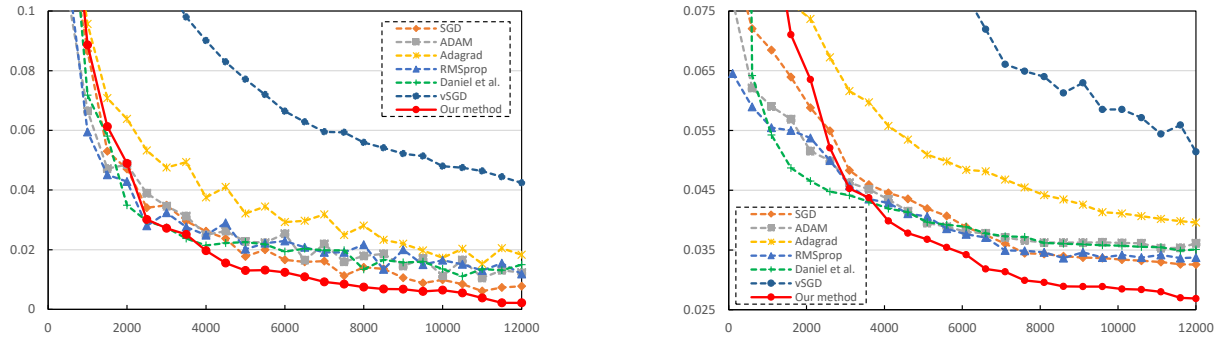
Algorithm 1 Actor-Critic Algorithm for Learning Rate Learning

Inputs: Training steps T ; training set X; loss function f; state function χ; discount factor: γ ; mini-batch size m_θ of actor network; mini-batch size m_φ of critic network; reset frequency of the model e

Initialize model parameters ω as ω_0, policy parameters θ of the actor network as θ_0, and value parameters φ of the critic network as φ_0.
for $t = 1, ..., T$ **do**
  Sample $x_i \in X, i \in 1, ..., N$.
  Extract state vector: $s_i^t = \chi(\omega^t, x_i)$.
  //Actor network selects an action.
  Computes learning rate $a_i^t = \pi_\theta(s_i^t)$.
  //Update model parameters ω.
  Compute $\nabla f^t(x_i)$.
  Update ω: $\omega^{t+1} = \omega^t - a_i^t \nabla f^t(x_i)$.
  //Update critic network by minimizing square error between estimation and label.
  $r^t = f^t(x_i) - f^{t+1}(x_i)$
  Extract state vector: $s_i^{t+1} = \chi(\omega^{t+1}, x_i)$
  Compute $Q_\varphi(s_i^{t+1}, \pi_\theta(s_i^{t+1})), Q_\varphi(s_i^t, a_i^t)$
  Compute δ^t according to Equation 7:
    $\delta^t = r^t + \gamma Q_\varphi(s_i^{t+1}, \pi_\theta(s_i^{t+1})) - Q_\varphi(s_i^t, a_i^t)$
  Compute the gradients of critic network according to Equation 8 :
    $\nabla \varphi^t = \delta^t \nabla_\varphi Q_\varphi(s_i^t, a_i^t)$
  if $t \bmod m_\varphi = 0$ **then**
    Update φ by $\nabla \varphi = \frac{1}{m_\varphi} \sum_{u=0}^{m_\varphi - 1} \nabla \varphi^{t-u}$
  end if
  // Update actor network
  Sample $x_j \in X, j \in 1, ..., N, j \neq i$.
  Extract state vector: $s_j^{t+1} = \chi(\omega^{t+1}, x_j)$.
  Compute $a_j^{t+1} = \pi_\theta(s_j^{t+1})$.
  Compute the gradients of actor network according to Equation 9:
    $\nabla \theta^t = \nabla_\theta \pi_\theta(s_j^{t+1}) \nabla_a Q_\varphi(s_j^{t+1}, a_j^{t+1})|_{a=\pi_\theta(s)}$
  if $t \bmod m_\theta = 0$ **then**
    Update θ by $\nabla \theta = \frac{1}{m_\theta} \sum_{u=0}^{m_\theta - 1} \nabla \theta^{t-u}$
  end if
  if $t \bmod e = 0$ **then** set $\omega_{t+1} = \omega_0$ **end if**
end for
return ω, θ, φ;

3.2 Critic Network

Recall that our goal is to find a good policy for learning rate control to ensure that a good model can be learnt eventually by the primary ML algorithm. For this purpose, the actor network needs to output a good action a^t at state s^t so that finally a low training loss $f(\cdot)$ can be achieved. In RL, the Q function $Q_\pi(s, a)$ is often used to denote the long term reward of the state-action pair s, a while following the policy π to take future actions. In our problem, $Q_\pi(s^t, a^t)$ indicates the accumulative decrement of training loss starting from step t. We define the immediate reward at step t as the one step



(a) (b)

Figure 2: Results on MNIST. (a) Training loss. (b) Test loss. The x-axis represents the number of mini batches. The y-axis represents loss value.

loss decrement:

$$r^t = f^t - f^{t+1}. \tag{5}$$

The accumulative value R_π^t of policy π at step t is the total discounted reward from step t:

$$R_\pi^t = \Sigma_{k=t}^{T} \gamma^{k-t} r(s^k, a^k),$$

where $\gamma \in (0, 1]$ is the discount factor.

Considering that both the states and actions are uncountable in our problem, the critic network uses a parametric function $Q_\varphi(s, a)$ with parameters φ to approximate the Q value function $Q_\pi(s, a)$.

3.3 Training of Actor and Critic Networks

The critic network has its own parameters φ, which is updated at each step using TD learning. More precisely, the critic is trained by minimizing the square error between the estimation $Q_\varphi(s^t, a^t)$ and the target y^t:

$$y^t = r^t + \gamma Q_\varphi(s^{t+1}, a^{t+1}). \tag{6}$$

The TD error is defined as:

$$\begin{aligned} \delta^t &= y^t - Q_\varphi(s^t, a^t) \\ &= r^t + \gamma Q_\varphi(s^{t+1}, \pi_\theta(s^{t+1})) - Q_\varphi(s^t, a^t) \end{aligned} \tag{7}$$

The weight update rule follows the on-policy deterministic actor-critic algorithm. The gradients of critic network are:

$$\nabla\varphi = \delta^t \nabla_\varphi Q_\varphi(s^t, a^t), \tag{8}$$

The policy parameters θ of the actor network is updated by ensuring that it can output the action with the largest Q value at state s^t, i.e., $a^* = \arg\max_a Q_\varphi(s^t, a)$. Mathematically,

$$\nabla\theta = \nabla_\theta \pi_\theta(s^{t+1}) \nabla_a Q_\varphi(s^{t+1}, a^{t+1})|_{a=\pi_\theta(s)}. \tag{9}$$

3.4 The Algorithm

The overall algorithm for learning rate learning is shown in Algorithm 1. In each step, we sample an example (Line 3), extract the current state vector (Line 4), compute the learning rate using the actor network (Line 6), update the model (Lines 8-9), compute TD error (Lines 11-15), update the critic network (Line 16-18), and sample another example (Line 20) to update the actor network (Line 21-26). We would like to make some discussions about the algorithm.

First, in the current algorithm, for simplicity, we consider using only one example for model update. It is easy to generalize to a mini batch of random examples.

Second, one may notice that we use one example (e.g., x_i) for model and the critic network update, but a different example (e.g., x_j) for the actor network update.

Doing so we can reduce oscillation during training. Suppose that the gradient direction of current example (or mini batch of examples) is quite different from others in this stage of training process. Intuitively at this step the model will be changed a lot to fit the example, consequently resulting in oscillation of the training, as shown in our experiments. As aforementioned, one principle for ideal learning rate control is to decrease it when the gradient vector oscillates, and increase it when the gradient vector follows a relatively steady direction. Therefore, we try to alleviate the problem by controlling learning rate according to gradient disagreement.

By feeding different examples to the actor and critic networks, it is very likely the critic network will find that the gradient direction of the example fed into the actor network is inconsistent with its own training example and thus criticize the large learning rate suggested by the actor network. More precisely, the update of ω is based on x_i and the learning rate suggested by the actor network, while the training target of the actor network is to maximize the output of the critic network on x_j. If there is big gradient disagreement between x_i and x_j, the update of ω, which is affected by actor's decision, would cause the critic's output on x_j to be small. To compensate this effect, the actor network is forced to predict a small learning rate for big gradient disagreement in this situation.

4 Experiments

We conducted a set of experiments to test the performance of our learning rate learning algorithm and compared with several baseline methods. We report the experimental results in this section.

4.1 Experimental Setup

We specified our actor-critic algorithm in experiments as follows. Given that stochastic mini-batch training is a common practice in deep learning, the actor-critic algorithm also operated on mini-batches, i.e., each step is a mini batch in our

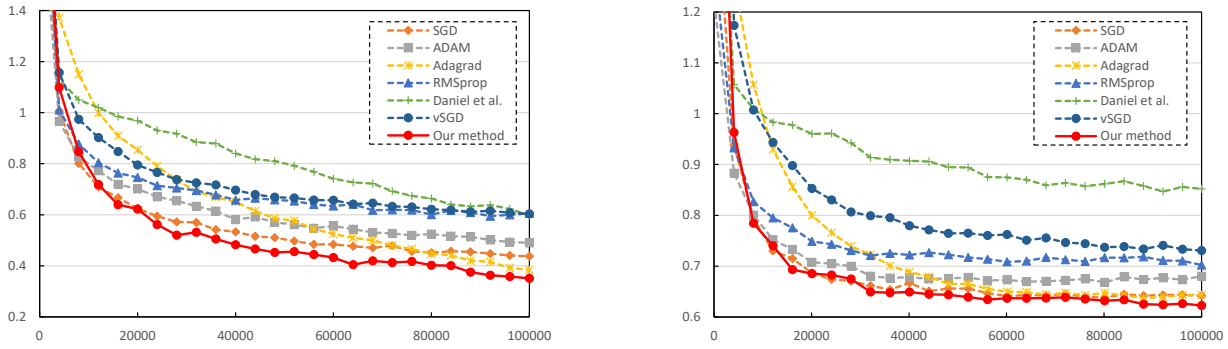


Figure 3: Results on CIFAR10. (a) Training loss. (b) Test loss. The x-axis is the number of mini batches. The y-axis represents loss value.

experiments. The state $s^t = \chi(\omega^t, X_i)$ is defined as the average loss of learning model ω^t on the input mini batch X_i.The actor network is specified as a two-layer long short-term memory (LSTM) network with 20 units in each layer, considering that a good learning rate for step t depends on and correlates with the learning rates at previous steps while LSTM is well suited to model sequences with long-distance dependence. The critic network is specified as a simple neural network with one hidden layer and 10 hidden units. Adam with the default setting in toolbox is used to train the learning rate learner in all the experiments.

We compared our method with several mainstream SGD algorithms, including SGD, Adam [Kingma and Ba, 2014], Adagrad [Duchi *et al.*, 2011] and RMSprop [Tieleman and Hinton, 2012]. We also compare our method with a recent work by [Daniel *et al.*, 2016][2]. This work identifies several hand-designed features and use an RL method (Relative Entropy Policy Search) to learn a learning rate controller. Another baseline method is "vSGD" [Schaul *et al.*, 2013][2], which automatically adjusts learning rates to minimize the expected error. It tries to compute learning rate at each update by optimizing the expected loss after the next update according to (1) the square norm of the expectation of the gradient and (2) the expectation of the square norm of the gradient.

4.2 Experimental Results

To verify the effectiveness of our method on different datasets and model structures, experiments are conducted on two widely used image classification datasets: MNIST [LeCun *et al.*, 1998] and CIFAR-10 [Krizhevsky and Hinton, 2009]. For simplicity, the primary ML algorithm adopted the CNN models and settings from tensorflow [Abadi *et al.*, 2015] tutorial, whose source code can be found at [TensorflowExamples,] For each of these algorithms and each dataset, we tried the following learning rates $10^{-4}, 10^{-3}, ..., 10^{0}$. We report the best performance of these algorithms over those learning rates. If an algorithm needs some other parameters to set, such as decay coefficients for Adam, we used the default setting in the toolbox. For each benchmark and our proposed method, five independent runs are averaged and reported in all of the following experiments. We trained all the baseline models until convergence.

[2]Thank the authors for providing the source code.

Results on MNIST

MNIST is a dataset for handwritten digit classification task. Each example in the dataset is a 28×28 black and white image containing a digit in $\{0, 1, \cdots, 9\}$. The CNN model used in the primary ML algorithm is consist of two convolutional layers, each followed by a pooling layer, and finally a fully connected layer. There are 60,000 training images and 10,000 test images in this dataset. We scaled the pixel values to the [0,1] range before inputting to all the algorithms. Each mini batch contains 50 randomly sampled images.

Figure 2 shows the results of our actor-critic algorithm and the baseline methods, including the curves of training loss and test loss. We have the following observations.

- Although the loss of our algorithm does not decrease very fast at the beginning, our algorithm achieves the best performance at the end. One may expect that our algorithm should have significantly faster convergence speed from the beginning considering that our algorithm learns both the learning rate and the CNN model, while most of the baseline methods only learn the CNN model and choose the learning rate per some predefine rules. However, this is not the case. Since our method targets at future long-term rewards rather than immediate rewards, it can make far-sighted decision and lead to better performance in long term.
- The loss curves of our approach is more smooth and stable than others. That is because we carefully design the algorithm and feed different samples to the actor network and critic network. As discussed in Section3.4, doing so we can reduce oscillation during training.

Results on CIFAR-10

CIFAR-10 is a dataset consisting of 60000 natural 32×32 RGB images in 10 classes: 50,000 images for training and 10,000 for test. We used a CNN with 2 convolutional layers (each followed by max-pooling layer) and 2 fully connected layers for this task.Before inputting an image to the CNN, we subtracted the per-pixel mean computed over the training set from each image.

Figure 3 shows the results of all the algorithms on CIFAR-10, including the curves of training loss and test loss. While the convergence speed of our method is similar to that of the

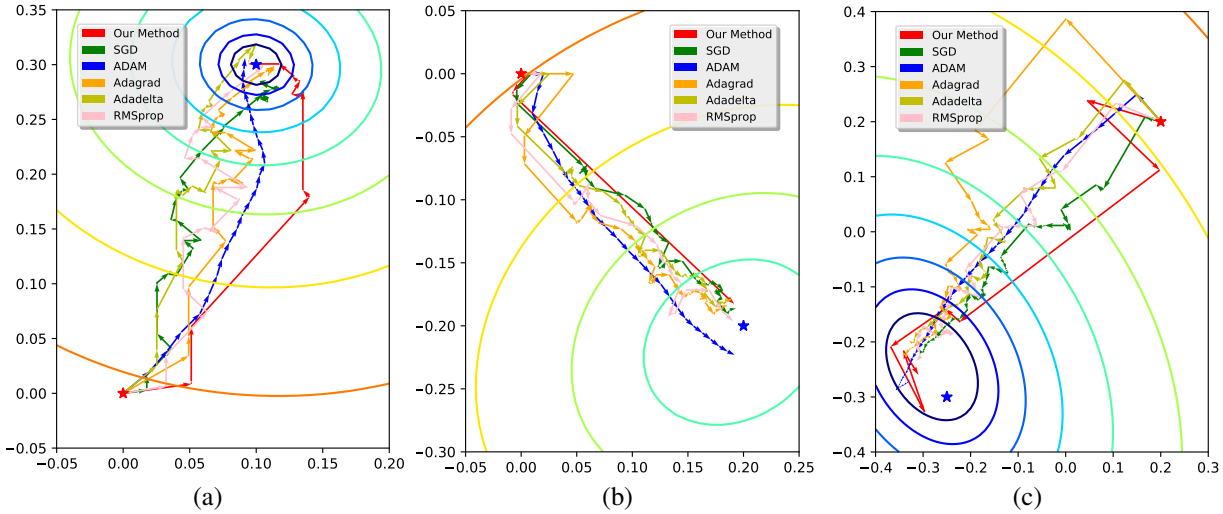


Figure 4: Trajectories produced by different algorithms on three random two-dimensional regression problems. The axes represent the values of the two dimensions. The contours outline the area with the same target value, and the target value is gradually decreasing from orange area to blue area. Each arrow represents one iteration of an algorithm, whose tail and tip correspond to the preceding and subsequent iterations respectively.

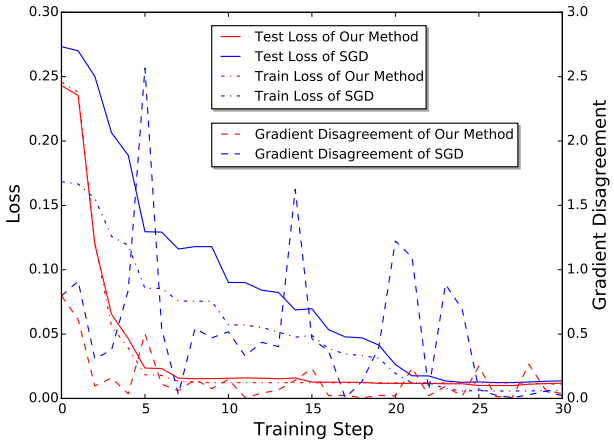


Figure 5: Gradient disagreement, training loss and test loss of SGD and our method on a two-dimensional regression problem.

baselines, the final performance of our method is the best among all the compared algorithms.

4.3 Further Analysis

In order to verify our intuitive explanation that by considering gradient disagreement, our method can make the learning process of the primary ML algorithm stable, here we conducted another experiment. In this experiment, we investigate the relationship between gradient disagreement and loss in the training process of a simple two-dimensional regression problem. We quantify gradient disagreement by using Euclidean distance between gradient on current batch of data and the overall gradient.

Figure 5 shows the gradient disagreement, training loss and test loss of SGD and our method. We can observe the correlation among them from the figure. As discussed in Section 3.4, by feeding different samples to actor and critic networks, our method would encourage the learning rate to be small when gradient disagreement is large, so that the oscillation of the training process would be relieved.

It is easy to see from the figure that test loss of our method is stable when there is big gradient disagreement, while the loss of SGD oscillates along with gradient disagreement, leading to slow speed of convergence. The test loss of SGD may increase when gradient disagreement increases, while in overall, our test loss decline in monotonous in the figure. Therefore, we need to feed different training data to the actor network and the critic network to ensure the performance of the algorithm.

To get deeper insight, we visualized the optimization process of our method. From Figure 4, we can find that our method get to convergence with fewer steps and the optimization trajectory is relatively smooth compared to other methods.

5 Conclusions and Future Work

In this work, we have studied how to control learning rates for gradient based machine learning methods and proposed an actor-critic algorithm, to help the main network achieve better performance. The experiments on two image classification tasks have shown that our method (1) can successfully adjust learning rate for different datasets and CNN model structures, leading to better convergence, and 2) can reduce oscillation during training.

For the future work, we will explore the following directions. In this work, we have applied our algorithm to control the learning rates of SGD. We will apply to other variants of SGD methods. We have focused on learning a learning rate for all the model parameters. We will study how to learn an individual learning rate for each parameter. We have considered learning learning rates using RL techniques. We will consider learning other hyperparameters such as step-

dependent dropout rates for deep neural networks.

References

[Abadi *et al.*, 2015] Martın Abadi, Ashish Agarwal, Paul Barham, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015.

[Bahdanau *et al.*, 2016] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

[Barto *et al.*, 1983] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[Daniel *et al.*, 2016] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning step size controllers for robust neural network training. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[Darken and Moody, 1990] Christian Darken and John Moody. Fast adaptive k-means clustering: some empirical results. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 233–238. IEEE, 1990.

[Duchi *et al.*, 2011] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[Jacobs, 1988] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LeCun *et al.*, 2012] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[Maclaurin *et al.*, 2015] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Orr and Müller, 2003] Genevieve B Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[Schaul *et al.*, 2013] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *ICML (3)*, 28:343–351, 2013.

[Senior *et al.*, 2013] Andrew Senior, Georg Heigold, Ke Yang, et al. An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6724–6728. IEEE, 2013.

[Silver *et al.*, 2014] David Silver, Guy Lever, and Nicolas Heess. Deterministic policy gradient algorithms. 2014.

[Silver *et al.*, 2016] David Silver, Aja Huang, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Simonyan and Zisserman, 2014] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Sutton and Barto, 1990] Richard S Sutton and Andrew G Barto. Time-derivative models of pavlovian reinforcement. pages 497–537, 1990.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[Sutton *et al.*, 1999] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[Sutton, 1984] Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

[Sutton, 1988] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[Sutton, 1992] Richard S Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*, pages 171–176, 1992.

[TensorflowExamples,] TensorflowExamples. Tensorflow examples. https://github.com/tensorflow/models/tree/master/tutorials/im

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Xu *et al.*, 2015] Kelvin Xu, Jimmy Ba, et al. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.

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