Gradient Descent Optimization by Reinforcement Learning

Yingda Zhu, Teruaki Hayashi, Yukio Ohsawa

Department of Systems Innovation, Graduate School of Engineering, The University of Tokyo

Gradient descent, which helps to search the global minimum of a complex (high dimension) function, is widely used in the deep neural network to minimize the total loss. The representative methods: stochastic gradient descent (SGD) and ADAM (Kingma & Ba, 2014) are the dominant ones to train neural network today. While some sensitive hyper-parameters like learning rate will affect the descent speed or even the convergence. In previous work, these hyper-parameters are often fixed or set by feedback and experience. I propose using reinforcement learning (RL) to optimize the gradient descent process with neural network feedback as input and hyper-parameter action as output to control these hyper-parameters. The experiment results of using RL based optimizer in both fixed and random start point shows better performance than normal optimizers which are set by default hyper-parameters.

1. Introduction

With the rapid development of deep learning these years, a lot of methods and models are proposed. While their cores, which are based on deep neural networks (DNN), are to minimize the DNN functions. Different from usual function minimization, dealing with DNN function which contains millions or even billions of parameters, it is nearly impossible to search the optimum solution. Gradient descent (GD) is an effective method to find the greedy solution which is good enough to be utilized in most cases. According to the related work these years, among the several GD methods, SGD and ADAM (Kingma & Ba, 2014) are the most popular and powerful ones which are used in diverse deep learning researches and applications. Some sensitive hyper-parameters in such methods (like learning rate in SGD) should be artificially set and may affect the descent speed and convergence. And related works use more complex optimizer structure to realize weight decay, but the optimization of GD might be affected by more diverse information which needs neural networks to express.

Reinforcement learning (RL), using states and rewards from the environment as input and the actions from the agent as output, whose training process is to find the actions in continuous states to maximize the total rewards. The feature of RL model, especially deep RL model, let it have strong ability in making strategy and actions which could deal with complex environment and adjust the optimizers’ hyper-parameters through the observed states and given rewards in GD process.

2. Related Works

In this section, the related work of GD method and RL method will be reviewed.

The development of gradient descent was a long time before the popularity of deep learning these years. SGD is the simplest but most essential one which is currently widely used even if many latest complex methods are proposed. Momentum SGD (Jacobs, 1988) solved the problem that GD process may slow down or stop in a saddle point or a local minimum. After 2008, several methods which benefit to deep neural networks were proposed since the backpropagation was widely used. Adagrad (Duchi et al. 2011) and RMSprop (Tieleman & Hinton, 2012) use the gradient square accumulation in the denominator to decay the learning rate. lr control in SGD by Actor-Critic is also proved possible (Xu et al. 2017). And Adam (Kingma & Ba, 2014) which combines the feature of Momentum and RMSprop shows the relatively good performance in various tasks. While AMSGrad (Reddi et al. 2018) revises some parts of ADAM which could lead nonconvergence in some special cases due to the fixed hyper-parameters. It seems hard to explore the best optimizer which could lead fast and stable descent process in many cases for the reason that the hyper-parameters in these methods are set by human experience or theoretical calculation which made the performance good in average.

Reinforcement learning (Sutton & Barto, 1988), different from supervised or unsupervised learning, was also put forward early and can solve the problem with no concrete labels. Q-learning (Watkins & Dayan, 1992) and SARSA (Rummery & Niranjan, 1994) are two typical value-based RL algorithm and deep RL combine RL and DNN which let multidimensional input become possible. Policy-based RL like DDPG (TP Lillicrap, 2015) and PPO (Schulman et al. 2017) can output continuous actions which perform well in special tasks that need precise control. Recent work like Deep Q-Network (Mnih et al. 2015) and AlphaGo (Silver et al.,2016) stabilized the learning and achieved outstanding results. The advantage of RL is not only to search the proper action in each step but make a strategy to maximize the long-term rewards.

3. Method and Algorithm

In this section, RL based optimizer structure and the detailed algorithm will be shown. The optimizer is mainly based on related work in gradient descent research, and the algorithm combines Deep Q-Network and GD process in target neural network.

3.1 Optimizer Structure

According to related work of optimizers in Table 1, the structure of optimizer can be summarized to two parts, learning rate decay with the root of quadratic polynomial (second-moment estimate) and momentum with linear polynomial (first-moment estimate) which are widely used in recent gradient descent research.
Table 1: Optimizer functions in related works

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adamax</td>
<td>$g_t = \max(\beta_t v_{t-1},</td>
</tr>
<tr>
<td>SGRD</td>
<td>$g_t = \eta_\text{SGRD} g_t$, where $\Delta x_0 = 0$ and $\Delta x_t = \beta_2 \Delta x_{t-1} + (1 - \beta_2) g_t \odot g_t'$</td>
</tr>
<tr>
<td>AMSGrad</td>
<td>$\tilde{g}<em>t = \frac{\eta}{\sqrt{\tau_t + \epsilon}} v_t$, where $\epsilon = \max(s_t - s</em>{t-1})$</td>
</tr>
</tbody>
</table>

Algorithm 1: CNN gradient descent process with Deep Q-learning optimizer $h(\mu_0)$

1. Initialize main and target network $Q$ and $Q'$ with parameter $\mu$ and $0^\dagger$, with action list $a$ and $a'$ respectively, hyper-parameters of optimizer $\mu_0$, threshold score $c_0$, constant $C$.
2. For episode $= 1$ to $M$ do
   1. Initialize CNN model $P$, sequence $s_1 = [x_1]$ and preprocessed $s_0 = \phi(s_0)$, score $= 0$, action $a_1$, reward $r_1$
   2. Update parameter of $P$ once with $h(\mu_0)$
   3. For $t = 1$ to $T$ do
      1. Random action $a_2$ in possibility $\epsilon$ and in possibility $\epsilon$
      2. Argmax $Q'(\phi(s_1), a; \theta)$ otherwise
      3. Update parameter of $P$ once with $h(\mu_2)$ by $a_{2+1}$
      4. Calculate state $x_{t+1}$, reward $r_{t+1}$ with CNN feedback
      5. Update $s_{t+1} = s_t$, $\phi(s_{t+1}) = \phi(s_{t+1})$
      6. Store $(\phi_t, a_t, r_t, \phi_{t+1})$, sample random $(\phi_j, a_j, r_j, \phi_{j+1})$
      7. Perform a gradient descent step on $(y_j - Q(\phi_j, a'_j, \theta))^2$ Update $Q$ with $y_{t+1}$, reset $Q = 0$ every $C$ steps, score = score + $r_{t+1}$
   8. If score < score$_{\text{end}}$, break
3. End

4. Experiment and Results

The experiments are based on shallow CNN (3 layers with 28938 parameters) and use 8000 MNIST data in training and validation. The training process of CNN contains 10 episodes, each of which has 250 steps with batch size equals 32. The DQN optimizer is based on SGD which control changeable learning rate $\eta$ and the action and reward definitions are highly related to loss behavior with different learning rates ($lr$).

4.1 Action and Reward Definition

Action, which equals the change of $lr$ in this case, is initially set at $lr = 1$, $act = (0.9, 1.1, 0.9)$ are three choices of action and let $lr = lr + act$ in each step. Figure 1 shows that there are 4 types of performance of the loss curve with different learning rates. The blue curve shows loss performance will stay at a high level when $lr$ are too big. In this case, the loss is hard to decrease later in any learning rate, which should be avoided as much as possible. The orange curve shows that with a smaller $lr$, the loss could be much smaller. Big $lr$ can help decrease fast in steep

Figure 1: Performance comparison of different learning rate which with the same start point and same batch order, the learning rates are 0.5, 0.4, 0.2, 0.01 respectively.
condition but hard to converge at last. The green one shows the curve performance when lr is proper. But notice that, the appropriate lr could also have some fluctuation which can help it escape some saddle points or local minimums. The red curve shows the situation when lr is too small. The curve is almost monotonically decreasing, but speed is low.

With the performance in Figure 1, rewards are separated into 3 priorities. The latter priority will be ignored if one reward is given. The first priority is to set the upper/lower bound of lr. If over bound, give large negative reward and break (In this experiment, upper = 0.5 and lower = 0.001 are set). The second priority uses the range of fluctuation in the latest 20 losses to judge whether the lr is too small. If so, continuous negative rewards would be given. In the third priority, if the current loss is larger than the minimum of first half of the loss data, lr will be judged too big and give a normal negative reward. If the loss reaches a new minimum or the threshold loss (0.02 in this experiment), a positive reward will be given according to the total steps.

4.2 States Definition

The simplest way to define states is using all the parameters directly. But simultaneously it means that there are too many states which should be considered and the model will be hard to train. Proper states are significant to ensure the convergence and training speed to a RL model. In this experiment, following types of information of CNN are considered:

(1) Total step (2) Loss based: max(Loss), min(Loss), previous Loss, delta(Loss), (3) Parameter based: increase/decrease parameter number, parameter average change ratio (4) Gradient based: gradient in previous 3 epochs, absolute value of gradient, sum of positive/negative gradient (5) others: monotonically decrease number of loss, hyper-parameters.

4.3 Results in fixed start point

Using the action, reward, states definition in 4.1 and 4.2, the training model will have 3 actions and 18 states. With the experience replay size of 2000 and target network update frequency of 100, the DQN model was trained for 50 epochs with the fixed start point. Figure 2.(a) shows the loss performance of the training process that becomes better in first 30 epochs. Compared to SGD, ADAM (figure 2.(c)), the performance of DQN optimizer shows strong ability to reach the lower loss and has larger fluctuation. Learning rate of DQN in figure 2.(b) goes to 0.2 at an early period and decrease to 0.1 whose fluctuation range is between 0.06 to 0.14.

4.4 Results in random start point

Without the limitation of a fixed start point, it needs more training episodes to ensure the convergence of DQN. Each start point is saved and trained with ADAM again to do the comparison. From the lr performance shown in Figure 3.(a), the average lr decrease in the first 250 epochs, then turn to increase and fluctuate. Combined with Figure 4, we found that the adjustment of strategy is first to decrease the average lr and limit the fluctuation which could avoid to get a large negative reward and relatively conservative. After the lr is near 0.05 which could let it get continuous negative reward written in 4.1, it began to rise and fluctuate in a range. In Figure 3(b), the minimum step curve to reach the threshold (0.02) is shown which indicates that if the DQN model reaches the best solution under the current strategy, it will ‘try’ to change strategy as to find a better solution. The convergence of cycle is not obvious, but get a result which is better than Adam in each cycle.
5. Conclusion

The hyper-parameters in optimizers can greatly affect the decrease speed and convergence stability in gradient descent process. Such like SGD, learning rate should be adjusted lower if the loss curve seems to converge but higher when met a saddle point.

In this work, reinforcement learning is used to consider more information which was feedbacked by the target network and adjust the hyper-parameter quickly and precisely. The general performance of DQN-based optimizer shows good potential within limited training episodes both in fixed and random condition. Simultaneously using DQN optimizer would not take much more time than other optimizers during the training process. If a DNN model is planning to train several times, for the structure of the network will not change, the DQN optimizer might be a good method to choose.

6. Future Direction

DQN model is the basic model in value-based deep reinforcement learning. In the future work, other latest models like Rainbow (Hessel et al. 2018) or policy-based models like DDPG (Lillicrap et al. 2015) can also be considered to improve the result. Also, the layer depth of CNN in this work is shallow (3 layers). Deeper neural networks or other types like RNN could be tested.

More hyper-parameters and a new type of optimizer structure are contradictions in GD process and the required minimum loss level will affect related reward design. A changeable parameter in the reward system can also operate training tendencies and lead to different results.

7. Acknowledgement

This work was funded by JSPS KAKENHI JP16H01836, JP16K12428, and industrial collaborators.

References


