Performance Comparision of Different Momentum Techniques on Deep Reinforcement Learning

Mehmet SARIGÜL
Cukurova University
Computer Engineering Department
Iskenderun Technical University
Computer Engineering Department
Email: msarigul@cu.edu.tr

Mutlu AVCI
Cukurova University
Biomedical Engineering Department
Email: mavci@cu.edu.tr

***Abstract*—Increase in popularity of deep convolutional neural networks in many different areas leads to increase in the use of these networks in reinforcement learning. Training a huge deep neural network structure by using simple gradient descent learning can take quite a long time. Some additional learning approaches should be utilized to solve this problem. One of these techniques is use of momentum which accelerates gradient descent learning. Although momentum techniques are mostly developed for supervised learning problems, it can also be used for reinforcement learning problems. However, its efficiency may vary due to the dissimilarities in two training learning processes. In this paper, the performances of different momentum techniques are compared for one of the reinforcement learning problems; Othello game benchmark. Test results show that the Nesterov accelerated momentum technique provided a more effective generalization on benchmark**

***Index Terms*—Deep reinforcement learning, momentum techniques, nesterov momentum**

I. Introduction

Deep neural network is a type of neural network which uses convolutional layers to extract abstraction of the input data. This allows processing of the large-scale data by artificial neural networks. Deep learning studies have been started at the 1940s. After 2006, the popularity of deep learning has risen and it has started to be used in many different scientific areas. Deep learning have achieved a very successful performance on large datasets including images [1], videos [2], texts [3] and speeches [4].

Deep neural networks are also used in reinforcement learning algorithms successfully. DQN [5] is one of the most popular algorithms. The trained DQN agent has managed to outperform human performance in 29 of 49 atari games. Another successful combination is Alpha-GO [6] which defeated European Go champion by 5 games to 0. Alpha-Go's policy network was firstly trained with expert human moves in the supervised fashion. After that, it was trained by self-play with reinforcement learning to be improved. DQN and Alpha-GO both use RMSprop momentum technique to accelerate the learning process of the artificial neural network.

In this study, deep convolutional neural networks were trained as policy functions for the Othello game. The performance of simple gradient descent and six different momentum techniques which are ordinary momentum, Nesterov accelerated momentum, Adagrad, Adadelta, RMSprop and Adam were compared over Othello training benchmark. While some other methods' efficiency was better in the early phases of the training, Nesterov accelerated momentum technique achieved a better performance for the related task in long run.

978-1-5090-5795-5/17/$31.00 © 2017 IEEE

II. Material and Methods

A. Deep Learning

Deep learning is a subsection of machine learning. A deep neural network is an artificial neural network with two or more hidden layers. Deep neural networks can include convolutional layers to extract an abstract representation of the input data. Convolutional layers are similar to ordinary neural network layers. They contain trainable weights and biases. A convolutional layer takes input and uses randomly weighted filters to extract a number of feature maps of the input. Filters can be selected in any size as needed. In each convolution layer, a selected number of filters can be used according to the work. Deep learning structure can be seen in Fig. 1.

Deep learning structures are mostly used for classification problems, but they can also be used as a policy function or a value function for reinforcement learning problems.

B. Reinforcement Learning

Reinforcement learning is a branch of machine learning that can be identified by four elements which are policy, reward function, value function and model of the environment [8]. A policy lets the agent choose its action at a given state. Reward function determines the reward gained with the selected actions. Value function determines the value of a state which represents how profitable being in that state. Model is the last element which contains all environmental rules. Value function of an environment can be calculated by an iterative form of Bellman Equation shown in (1) [9].

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \quad (1)$$

where $V(s_t)$ is value of the current state. $V(s_{t+1})$ is value of the next state. $r_{t+1}$is the reward obtained in transition from s_t to s_{t+1}. α is called as learning rate and γ is called

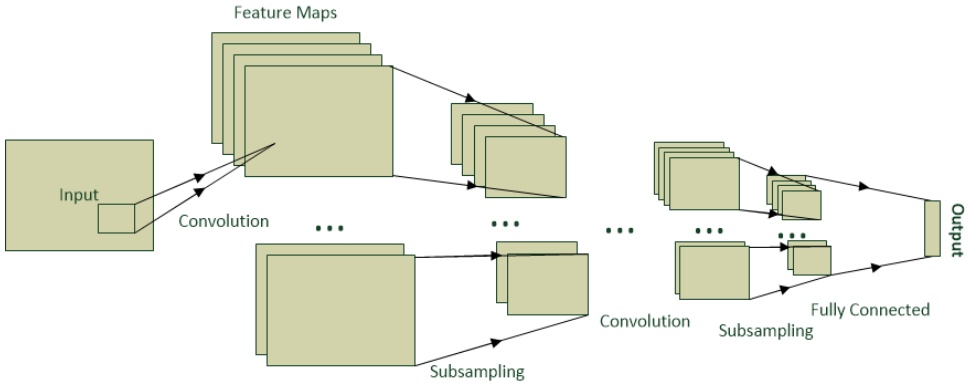

Fig. 1. Deep Learning Structure. Reprinted from [7].

as "discount rate". α and γ are predefined constants. It can take quite a long time to calculate the value function directly with this equation for environments with large state space. Therefore, a regression method is required to fit the value function. Many linear regression methods have been used for this purpose. Also, nonlinear function approximators such as neural networks can be used for generalization of the value function.

Using neural network for function approximation is problematic, for this reason, Williams sets some rules for using a neural network in reinforcement learning problems with REINFORCE algorithm [10]. REINFORCE algorithm shows that update of the weight of a neural network can be done with (2).

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})\frac{\partial ln g_i}{\partial w_{ij}}, \tag{2}$$

where α_{ij} is learning rate, b_{ij} is reinforcement baseline, g_i is the function to be approximated.

C. Momentum Techniques

Backpropagation with simple gradient descent is a well known technique which is used to train an artificial neural network. In this method, weights of the neural network are updated in the opposite direction of the derivative of the error function to minimize it as in (3).

$$w_t = w_{t-1} - \eta \nabla_w E(w_{t-1}), \tag{3}$$

Training of a huge deep neural network can require a very long processing time with simple gradient descent. Therefore, momentum techniques which accelerate the training process can be very useful to overcome this problem. Update equations of ordinary momentum [11] is shown as (4) and (5).

$$v_t = \gamma v_{t-1} + \eta \nabla_w E(w_{t-1}) \tag{4}$$

$$w_t = w_{t-1} - v_t \tag{5}$$

where γ is the momentum rate which is usually set to 0.9. Nesterov accelerated momentum [12] is a method which gives foresight to the momentum term. Since $w - \eta v_{t-1}$ shows the approximation of the next weight values, $\nabla_w E(w - \eta v_{t-1})$ can be used instead of $\nabla_w E(w)$ in (4) to calculate more accurate momentum term. New form is shown in (6).

$$v_t = \gamma v_{t-1} + \eta \nabla_w E(w_{t-1} - \gamma v_{t-1}), \tag{6}$$

$$w_t = w_{t-1} - v_t, \tag{7}$$

Adagrad [13] is a gradient-based optimization algorithm which regulates the learning rate in a way to be bigger for infrequent parameters and smaller for frequent ones. It is mostly used for sparse datasets. Adagrad update is shown in (8) and (9).

$$g_t = g_{t-1} + \nabla_w E(w_{t-1})^2 \tag{8}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{g_t + \epsilon}} \nabla_w E(w_{t-1}), \tag{9}$$

ϵ is a very small number used to prevent division by zero. RMSprop [14] is similar to Adagrad. Main difference between them is that the g_t term is calculated as exponentially decaying average. Update process of RMSprop can be seen in (10) and (11).

$$g_t = \gamma g_{t-1} + (1-\gamma) \nabla_w E(w_{t-1})^2 \tag{10}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{g_t + \epsilon}} \nabla_w E(w_{t-1}), \tag{11}$$

Adadelta [15] also uses a decaying amount of the all past sum squared of gradients to prevent rapid shrinkage of the learning rate. Adadelta update rule is shown in (12),(13), (14) and (15).

$$g_t = \gamma g_{t-1} + (1-\gamma) \nabla_w E(w_{t-1})^2 \tag{12}$$

$$v_t = \frac{\sqrt{x_{t-1} + \epsilon}}{\sqrt{g_t + \epsilon}} \nabla_w E(w_{t-1}) \tag{13}$$

$$x_t = \gamma x_{t-1} + (1-\gamma) v_t^2, \tag{14}$$

$$w_t = w_{t-1} - v_t, \tag{15}$$

Adam [16] is another momentum algorithm which uses not only average of sum of the squared gradients but also average

of the sum of the gradients in the update process as shown in (16), (17), (18), (19) and (20).

$$m_t = \gamma_1 m_{t-1} + (1-\gamma_1)\nabla_w E(w_{t-1}), \tag{16}$$

$$g_t = \gamma_2 g_{t-1} + (1-\gamma_2)\nabla_w E(w_{t-1})^2, \tag{17}$$

$$\hat{m}_t = \frac{m_t}{1-\gamma_1^t}, \tag{18}$$

$$\hat{g}_t = \frac{g_t}{1-\gamma_2^t}, \tag{19}$$

$$w_t = w_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{g}_t + \epsilon}}. \tag{20}$$

III. Experiments

Othello is a well known strategy game which have been played for decades. It is a strategy game played between two opponents on an eight by eight board. Each player is represented by either black or white discs. The starting position of the game can be seen in Fig. 2. Each player puts a disc on the board when it comes to his/her turn. The opposite colored discs existing between the disc has been put and the disc having the same color with it, are reversed. At the end of the game, player who has the largest number of discs on the board won.

Othello game was chosen for the experiment because visited states are not repeated during the game. Each player has to play forward. This feature of the game prevents the agent to learn a defensive optimal strategy and forces it to travel within the state space. Since the states in the simulated games are constantly changing, value function can be trained by only self-play. Tesauros TD-gammon [17] is the most known application which includes a neural network trained by self-play. TD-gammon was successful because of the fact that backgammon has a high rate of stochasticity. By throwing dice for each state, it is provided that the algorithm travels in all around the state-action space. The trained network may find an allegedly optimal policy and diverge in deterministic games. Despite this fact, a policy network was successfully trained by only self-play during the experiment for Othello game.

Alpha-GO used a 13-convolutional layered neural network with 192 filters for each layer to represent the game board of Go. A similar structure which takes the state of the board as an input and gives a probability distribution over all action as an output was used in the experiments. The game board is represented with 8 by 8 matrix in which white discs are represented with 1, black discs are represented with -1 and empty cells are represented with 0. There were not any extra hand crafted or game-dependent features. 3 by 3 filters were used for each convolutional layer with 2 padding over the picture. Therefore, each convolutional layer had a 8 by 8 pictures as and input and output. Before the logarithmic softmax layer which calculates the probabilities of actions, an extra layer which adds bias values to the outputs were used.

The first experiment was aimed to find the effective structure for the Othello game. Different structures were tried to determine the structure to be used in momentum experiment. Winning rate against an agent which plays uniformly random was used as a performance measure. While this measure is not suitable to determine the expertise level of the agent for the game, it is still a useful metric to determine the best momentum method. REINFORCE algorithm was used to train the artificial neural network structures. Reinforcement baseline was taken 0 for all experiments. The reward was taken as 1 for all states for a game which is won and it was taken -1 for all states for a game which is lost. After determining the optimal structure for the game number of tests were tried over the network with different momentum techniques.

TABLE I
Performance of Different Structures

# of Conv. Layers	# of Filters	Max. Performance
6	64	61.23
7	128	63.85
8	128	81.93
9	192	89.88

Batch learning method was used to train the deep neural network since interactive learning was failed to convergence a good solution. Using unbalanced data in the sense of winning and losing were also unsuccessful in the experiments. When whole states of a game were used for training, this also led poor results. As a result of these experiences, batches contain 100 states; while 50 of them were randomly selected from last 50 game which was won and the others were selected from last 50 games which were lost were used. After each game, only 1 batch was passed through the network. The training rate was taken 0.003 for all experiments. Torch framework was used for all experiments [18].

IV. Results

Different deep neural network structures which have a different number of convolutional layers and filters were tested to find the optimal structure for the Othello game. Different structures exhibited different performances which were between 61.23 and 89.88. Performance comparison of the structures is shown in Table I.

9 convolutional layered structure with 192 filters each layer was selected for the momentum experiment. Due to the time complexity, bigger structures were not tested. For each momentum technique, an agent was run for 1 million games. Training was started after first 100 games which 50 of them were won and 50 of them were lost. Therefore, nearly 1 million batches were used for each training. Each technique was run for 5 experiments. A full experiment nearly took 2 days over 3600 cored GPU with CUDA support. In the experiments, when a deep neural network started to memorize, the performance of the network dropped dramatically. In this case, the training was halted and the previous performance before the drop was taken as the performance of the experiment. Since different initial weight values led to different performances especially at the beginning of the training, during experiments,

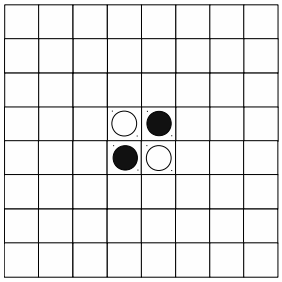

Fig. 2. Othello Starting Position

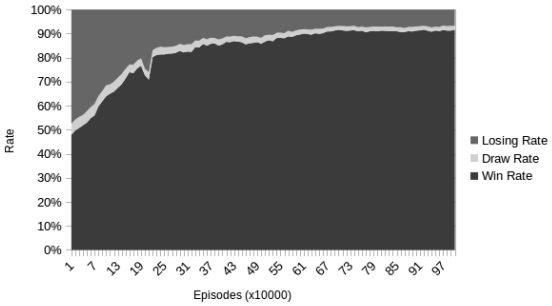

Fig. 3. Nesterov Momentum Result

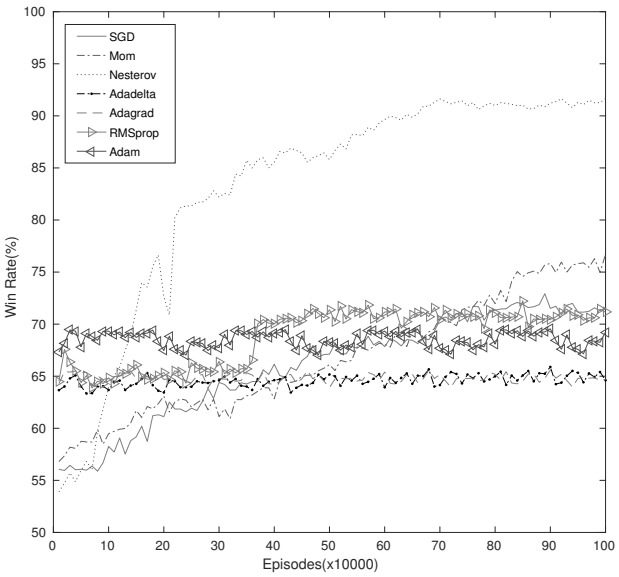

Fig. 4. Comparision of Momentum Techniques

the same initial weight values were used for all momentum techniques.

Performance comparison of the techniques can be seen in Fig. 4. While Adadelta, Adagrad, RMSprop and Adam provided better results in the beginning of the training process, Nesterov momentum increased the performance slowly and regularly. Different techniques provided different performance values between 65.59 and 91.65. Maximum performance values reached by the techniques are shown in table II.

TABLE II
MAX. PERFORMANCE OF DIFFERENT TECHNIQUES

Momentum Technique	Ave. Max. Performance
SGD	72.92
Momentum	76.71
Nesterov	91.65
Adagrad	65.89
Adadelta	65.59
RMSProp	72.27
Adam	69.55

V. CONCLUSION

Deep learning has become a very popular tool for many different scientific areas lately. Backpropagation with simple gradient descent is mostly chosen for training due to its stability, but it can be very slow in the training of this huge structures. Momentum techniques can be useful to speed up the training process of the gradient descent. Although momentum techniques are mostly developed for supervised learning problems, it can also be used for reinforcement learning problems. However, its efficiency may vary due to the dissimilarities in two training learning processes. Unlike the supervised learning, a state which is considered as very profitable in the beginning of the training can be found as nonprofitable during the reinforcement learning training process.

6 different momentum techniques and also simple gradient descent without momentum were compared over Othello game benchmark. In contrast to methods such as Adagrad, Adadelta, RMSprop, and Adam which are more successful for supervised learning, Nesterov momentum was more efficient in the experiments. While Nesterov momentum's performance was lower than that of other methods for nearly the first 100000 episodes, Nesterov momentum achieved 91.65 performance rate with specified performance metric in the long run. This result is far better than all another compared techniques.

As a result of the experiments, it can be said that Nesterov momentum technique can be more suitable for this type of applications because it is relatively fast when compared to simple gradient descent and also flexible enough to be used for reinforcement learning.

REFERENCES

[1] Krizhevsky, A., Sutskever, I., Hinton, G. E. *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems (pp. 1097-1105) 2012.

[2] Karpathy, Andrej, et al. *Large-scale video classification with convolutional neural networks.* Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014.

[3] Collobert, R., Weston, J. *A unified architecture for natural language processing: Deep neural networks with multitask learning.* In Proceedings of the 25th international conference on Machine learning (pp. 160-167). ACM. 2008.

[4] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... Kingsbury, B. *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups.* IEEE Signal Processing Magazine, 29(6), 82-97. 2012.

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

[6] Silver, David and Huang, Aja and Maddison, Chris J and Guez, Arthur and Sifre, Laurent and Van Den Driessche, George and Schrittwieser, Julian and Antonoglou, Ioannis and Panneershelvam, Veda and Lanctot, Marc and others *Mastering the game of Go with deep neural networks and tree search*.Nature 529.7587: 484-489, 2016.

[7] M. SARIGÜL and M. AVCI *Comparison of different deep structures for fish classification*.International Conference on Computer and Information Technology 2017, accepted on 30 December 2016.

[8] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction.* Vol. 1. No. 1. Cambridge: MIT press, 1998.

[9] Sutton, Richard S. *Learning to predict by the method of temporal differences* Machine Learning (Springer) 3: 944 1988.

[10] Williams, Ronald J. *Simple statistical gradient-following algorithms for connectionist reinforcement learning.*.Machine learning 8.3-4 229-256, 1992.

[11] Qian, Ning. *On the momentum term in gradient descent learning algorithms.*.Neural networks 12.1 145-151 1999.

[12] Nesterov, Yurii. *A method for unconstrained convex minimization problem with the rate of convergence O (1/k2).* Doklady an SSSR. Vol. 269. No. 3. 1983.

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

[14] Tieleman, Tijmen, 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.

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

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

[17] Tesauro, G. *Temporal difference learning and TD-Gammon.* Communications of the ACM, 38(3), 58-68. 1995.

[18] Ronan, Clment, Koray and Soumith. *http://torch.ch/.*