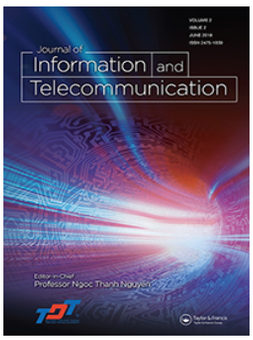

Journal of Information and Telecommunication

ISSN: 2475-1839 (Print) 2475-1847 (Online) Journal homepage: https://www.tandfonline.com/loi/tjit20

Performance comparison of different momentum techniques on deep reinforcement learning

M. Sarigül & M. Avci

To cite this article: M. Sarigül & M. Avci (2018) Performance comparison of different momentum techniques on deep reinforcement learning, Journal of Information and Telecommunication, 2:2, 205-216, DOI: 10.1080/24751839.2018.1440453

To link to this article: https://doi.org/10.1080/24751839.2018.1440453

Published online: 23 Feb 2018.

JOURNAL OF INFORMATION AND TELECOMMUNICATION, 2018
VOL. 2, NO. 2, 205–216
https://doi.org/10.1080/24751839.2018.1440453

Performance comparison of different momentum techniques on deep reinforcement learning*

M. Sarigül [a,b] and M. Avci[c]

[a]Computer Engineering Department, Cukurova University, Adana, Turkey; [b]Computer Engineering Department, Iskenderun Technical University, Iskenderun, Turkey; [c]Biomedical Engineering Department, Cukurova University, Adana, Turkey

ABSTRACT

Recently, the popularity of deep artificial neural networks has increased considerably. Generally, the method used in the training of these structures is simple gradient descent. However, training a deep structure with simple gradient descent can take quite a long time. Some additional approaches have been utilized to solve this problem. One of these techniques is the momentum that accelerates gradient descent learning. Momentum techniques can be used for supervised learning as well as for reinforcement learning. However, its efficiency may vary due to the dissimilarities in two learning processes. While the expected values of inputs are clearly known in supervised learning, it may take long-running iterations to reach the exact expected values of the states in reinforcement learning. In an online learning approach, a deep neural network should not memorize and continue to converge with the more precise values that exist over time during these iterations. For this reason, it is necessary to use a momentum technique that both adapt to reinforcement learning and accelerate the learning process. In this paper, the performance of different momentum techniques is compared with the Othello game benchmark. Test results show that the Nesterov momentum technique provided a more effective generalization with an online reinforcement learning approach.

ARTICLE HISTORY
Received 23 October 2017
Accepted 11 February 2018

KEYWORDS
Deep reinforcement learning; momentum techniques; Nesterov momentum; policy function approximation

1. Introduction

A deep neural network is a type of neural network which uses convolutional layers to extract abstraction from the input data. This allows processing of the large-scale data and learning much-complicated relations by an artificial neural network. However, it may take a lot of processing time to train these big structures. Hence, various methods have been developed to accelerate the training process. These methods are called momentum techniques. The performance of momentum techniques may vary according

CONTACT M. Sarigül msarigul@cu.edu.tr, mhmtsarigul@gmail.com

*This paper is an extended and detailed version of Inista 2017 Conference Paper (Sarigul & Avci, 2017b). In this paper, in addition to the Inista paper, performance comparisons of 12 different deep learning structures are given. Another momentum technique is also added to the momentum comparison experiment.

to the method of training and the structure of the problem being trained. In this paper, deep learning structures were trained with an online reinforcement learning approach as Othello game agents. The performance of the momentum techniques in the training of this structure was compared. The aim of this study is to determine the most successful momentum technique that can be used while training deep learning structures for Othello and similar applications with online reinforcement learning methods.

Deep learning studies have been started in the 1940s (McCulloch & Pitts, 1943). Convolutional neural networks were proposed as an effective solution for pattern recognition problems in 1998 (LeCun, Bottou, Bengio, & Haffner, 1998). After 2006, the popularity of deep learning and convolutional neural networks have 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 (Krizhevsky, Sutskever, & Hinton, 2012), videos (Karpathy et al., 2014), texts (Collobert & Weston, 2008) and speeches (Hinton et al., 2012).

Deep neural networks have also been used in reinforcement learning algorithms successfully. DQN (Mnih et al., 2013) is one of the most popular algorithms. The DQN algorithm was used to train deep learning structures for various arcade games. In that study, the pixel values on the Atari screen were given as input data to the neural network structure. The neural network outputs a probability distribution over the probable actions for the game. The changes in the game score resulting from the selected actions were used to train the network. The trained DQN agent has managed to outperform human performance in 29 of 49 Atari games. Another successful combination is Alpha-GO (Silver et al., 2016) which defeated European Go champion by 5 games to 0. Alpha-GO used two different neural networks, one as value function and one as policy function. Each of these networks contains 13 convolutional layers. The policy network gives a probability distribution over the possible actions, while the value network gives the expected value of a state. Alpha-GO's value function was trained with self-play games. On the other hand, the policy network was firstly trained with expert human moves in the supervised fashion with a database of around 29.4 million moves, 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 deep neural network. Lai (2015) used deep learning techniques to train a chess game agent. Nesterov accelerated momentum, AdaGrad and AdaDelta methods were tried to accelerate learning of the chess agent. AdaDelta was the most successful method in this experiment because it uses different learning rates for each parameter in the network. Since some states in chess are very rare, it is advantageous to have a higher learning rate for the network part that is responsible for these states. Yakovenko, Cao, Raffel, and Fan (2016) used convolutional neural network structures to train a poker game agent and used the Nesterov accelerated momentum technique in the learning process.

It has been seen that different momentum techniques are more successful for different problems and methods in the studies carried out to this day. In this study, it was aimed to find the best momentum technique that can be used while deep learning structures are trained with reinforcement learning methods for checker-like games played with fewer stone types on a board. The Othello game was chosen for the experiment. Deep convolutional neural network structures were trained to be used as policy functions for the Othello game. Two different experiments were performed. In the first experiment, the

performance of the different structures was measured as policy functions. The structure with 9 convolutional layers with 192 filters in each layer achieved the best performance on the benchmark. This structure was selected for the second experiment. In the second experiment, the performance of simple gradient descent and seven different momentum techniques which are ordinary momentum, Nesterov accelerated momentum, Adagrad, Adadelta, RMSprop, Adam and AdaMax were compared with the Othello training benchmark on the selected structure. While some other methods' efficiency was better in the early phases of the training, the Nesterov accelerated momentum technique achieved a better performance for the related task in the long run. It provided 19.48% improvement over the nearest performing methodology.

2. Materials and methods

In this study, multiple methods were used together in the training of the Othello game agent. The agent was trained by an online reinforcement learning methodology. Reinforcement learning methods may be used in an online or offline fashion. In online learning methodology, the structure is simultaneously trained while running on the problem. The last produced data is used in the training. On the contrary, a certain amount of data is generated without changing the structure in the offline learning approach and then this produced data is used to train the structure. Training in offline learning is similar to supervised learning. It is more challenging to train a structure with an online learning approach. If the training data is not sufficiently sparse, training may be stuck in local minima. Therefore, the training methodology and the structure must be carefully decided.

The error of a selected action was calculated with the REINFORCE (Williams, 1992) algorithm during the training. This computed error was used in the training of the deep learning structure. The deep structure was used as a policy function. In other words, the input of the network is the state of the game board and output of the network is a probability distribution on the possible actions. In order to determine the effective deep learning structure, the performance of many different structures was compared. After the effective structure was selected, a second experiment was conducted to compare the performance of the momentum techniques. In that experiment, gradients of the deep neural network had been updated according to various momentum techniques. At the end of the work, the performance of the different momentum techniques was compared.

2.1. 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. Pooling layers which mostly follows the convolutional layers take feature maps as inputs and generate new smaller feature maps which can be called summary of the input feature maps.

Convolutional layer and pooling layer pairs end up with a fully connected neural network. The output layer of this structure mostly contains a softmax layer for classification purposes. The back-propagation algorithm is used in the training process of the structure in the majority of the works. A general deep learning structure can be seen in Figure 1.

Deep learning structures are mostly used for image classification problems, but they can also be used as a policy function or a value function for reinforcement learning problems for many other purposes. While deep learning is used for reinforcement learning, much larger structures need to be used than they may have to represent much more complex relationships between the inputs. This causes the prolongation of the training process. It is imperative to develop methods to shorten this period.

2.2. 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 (Sutton & Barto, 1998). A policy determines the agent how to choose its action in a given state. The reward function determines the reward gained with the selected actions. The 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. One of the approaches to reinforcement learning is to learn a value function to represent the values of the states and to transition to states that will maximize this value function. This value function of an environment can be calculated by an iterative form of the Bellman equation shown below (Sutton, 1988):

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

where $V(s_t)$ is the value of the current state, $V(s_{t+1})$ is the 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 as 'discount rate', and 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.

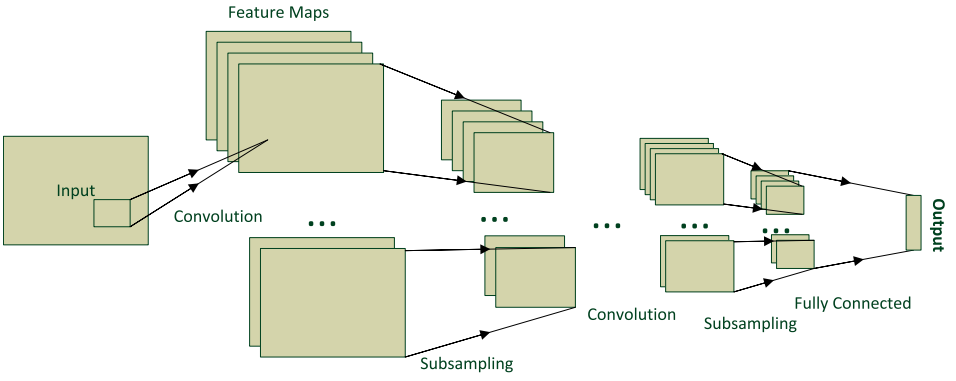

Figure 1. Deep learning structure. Reprinted from Sarigül and Avci (2017a).

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 the REINFORCE algorithm (Williams, 1992). The REINFORCE algorithm shows that update of the weight of a neural network can be done with the following equation:

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

where α_{ij} is the learning rate, b_{ij} is the reinforcement baseline and g_i is the function to be approximated. It is possible to train such structures as deep learning structures by using this algorithm.

2.3. Momentum techniques

Back-propagation 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 the following equation:

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

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. In the ordinary momentum, a discounted cumulative sum of the previous gradient values is used instead of using the gradient directly. This change reduces the oscillations when approaches the local minimum of the error function. Update equations of ordinary momentum (Qian, 1999) is shown as follows:

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

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

where γ is the momentum rate which is usually set to 0.9. Nesterov (1983) accelerated momentum 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 Equation (4) to calculate more accurate momentum term. The new form is shown in the following equation:

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

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

Adagrad (Duchi, Elad, & Yoram, 2011) 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 the following equations:

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

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

ϵ is a very small number used to prevent division by zero. RMSprop (Tieleman & Geoffrey, 2012) is similar to Adagrad. The main difference between them is that the g_t term is calculated as exponentially decaying average. The update process of RMSprop can be seen in the following equations:

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

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

Adadelta (Zeiler, 2012) also uses a decaying amount of the all past sum squared of gradients to prevent rapid shrinkage of the learning rate. The Adadelta update rule is shown in the following equations:

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

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

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

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

Adam (Kingma & Ba, 2014) is another momentum algorithm which uses not only the average of sum of the squared gradients but also the average of the sum of the gradients in the update process as shown in the following equations:

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

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

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

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

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

Adamax (Kingma & Ba, 2014) is a variant of Adam. In this method, ℓ_∞ norm is used instead of ℓ_2 norm in the calculation of g_t term. The update process can be seen in the following equations:

$$m_t = \gamma_1 m_{t-1} + (1-\gamma_1)\nabla_w E(w_{t-1}), \tag{21}$$

$$g_t = \gamma_2 g_{t-1} + (1-\gamma_2)|\nabla_w E(w_{t-1})|^\infty = \max(\gamma_2 g_{t-1}, |\nabla_w E(w_{t-1})|), \tag{22}$$

$$\hat{m}_t = \frac{m_t}{1-\gamma_1^t}, \tag{23}$$

$$w_t = w_{t-1} - \frac{\eta \hat{m}_t}{g_t}. \tag{24}$$

3. Experiments

Othello is a well-known strategy game which has been played for decades. It is played between two opponents on an 8×8 board. There are 64 discs, which are white on one side and black on the other side. Each player is represented by either black or white colour. The starting position of the game can be seen in Figure 2. Each player puts a disc on the board when it comes to his/her turn. When the player represented by white puts a white disc on the game board, the black discs between the disc placed and the other white discs are reversed. For the player represented by black, it is applied in a similar way with the opposite colors. At the end of the game, the player who has the largest number of discs in his/her color on the board won.

In this work, Othello game agents were trained in online reinforcement learning methodology. First of all, a deep learning network structure with random weights was established. The agent played the game by giving the current state of the board as input to this network and choosing the action that has the highest expected return according to the output values of the network. The opponent plays by selecting random moves in any state. This increased the sparsity of the data. Only the data from last 100 games were used in each iteration for the training. After each iteration, the network was trained for a batch containing 100 different positions and the same network was continued to be used in the simulations. Online training may fail if the sparsity of the data is not provided in such applications. 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 which lets the agent circulate within the same states by selecting same actions. The agent is forced to travel within the state space during the play. This had resulted in a more accurate learning because it increases the diversity of selected actions. Tesauros TD-gammon (Tesauro, 1995) 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 play, it is provided that the algorithm travels in all around

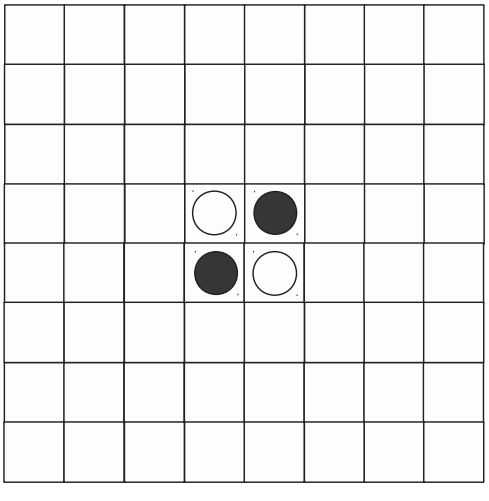

Figure 2. Othello starting position.

the state-action space. In contrary, 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 the Othello game.

Alpha-GO used a 13-convolutional layered neural network with 192 filters for each layer to represent the state-action space of Go game. A similar structure which takes the state of the board as input and gives a probability distribution over all possible action as output was used in the experiments. The game board was represented with 8×8 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 given to the neural network during the training process. 3×3 filters were used for each convolutional layer with two padding over the picture. Therefore, each convolutional layer had an 8×8 picture as input and output. Before the logarithmic softmax layer which calculates the probability distribution over the actions, an extra layer which adds bias values to the outputs was used.

The first experiment was aimed to find the effective deep learning structure for the Othello game. Different structures were tried as policy functions. Structures with six to nine convolutional layers which contain 64,128 and 192 filters were tried in the 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 most efficient structure. The REINFORCE algorithm was used to train the artificial neural network structures. Reinforcement baseline was taken as 0 for all experiments. The reward was taken as 1 for all states for a game which is won and it was -1 for all states for a game which is lost. The 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 was 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 the last 50 games which were won and the others were selected from the last 50 games which were lost were used in the training process. After each game, only one batch was passed through the network. After determining the optimal structure for the Othello game, this structure was trained with different momentum techniques by using the same parameters. Finally, these trained structures were compared according to winning rates. Each experiment was repeated 10 times and the averages of the performance values were taken. The training rate was taken as 0.003 and the Torch framework was used for all experiments (Ronan, Clement, Koray, & Soumith, 2017).

4. Results

In the first experiment, 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. Structures with six convolutional layers showed a performance between 61.23% and 67.42%. Seven convolutional layered structures gained a performance between 63.85% and 78.68%. the performance of eight convolutional layered structures was between 66.79% and 83.41%. Structures with nine convolutional layers which

Table 1. Performance of different structures.

	# of Filters		
# of Conv. layers	64	128	192
6	61.23	65.45	67.42
7	63.85	75.87	78.68
8	66.79	81.93	83.41
9	72.13	84.27	89.88

performed the best performance on this experiment had a performance between 72.13% and 89.88%. While increasing the number of the filters provided a performance boost up to 24.88%, increasing the number of the convolutional layers provided a performance boost up to 33.31%. These results showed that increasing the number of convolutional layers or number of the filters in the convolutional layers may also increase the performance of the structure. Performance comparison of the structures is shown in Table 1.

Nine convolutional layered structure with 192 filters each layer was selected for the momentum experiment. Due to the time complexity, bigger structures were not tested. An agent was run for 1 million games for each momentum technique. 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 10 experiments. A full experiment nearly took 2 days over 2800 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, the same initial weight values were used for all momentum techniques.

Performance comparison of the techniques can be seen in Figure 4. While Adadelta, Adagrad, RMSprop, Adam, and AdaMax provided better results at the beginning of the

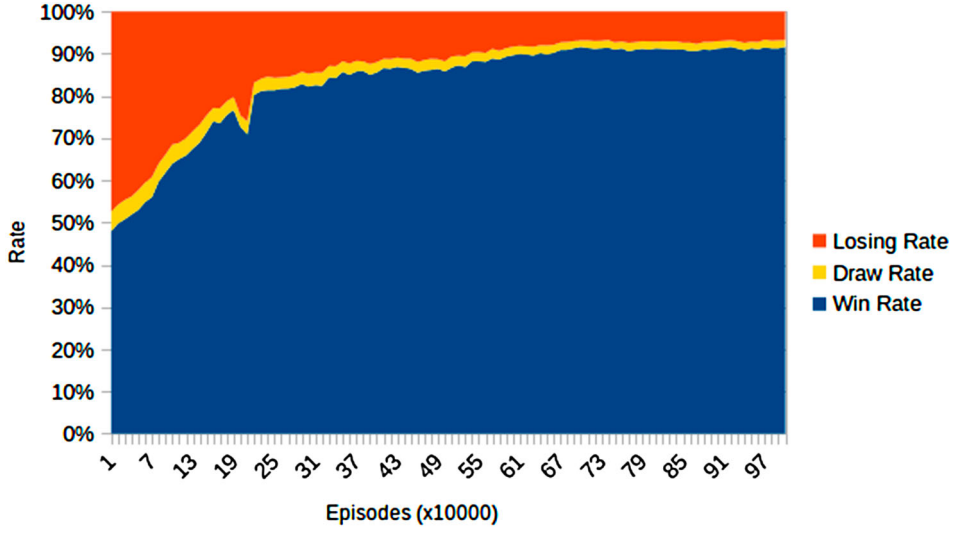

Figure 3. Nesterov momentum result.

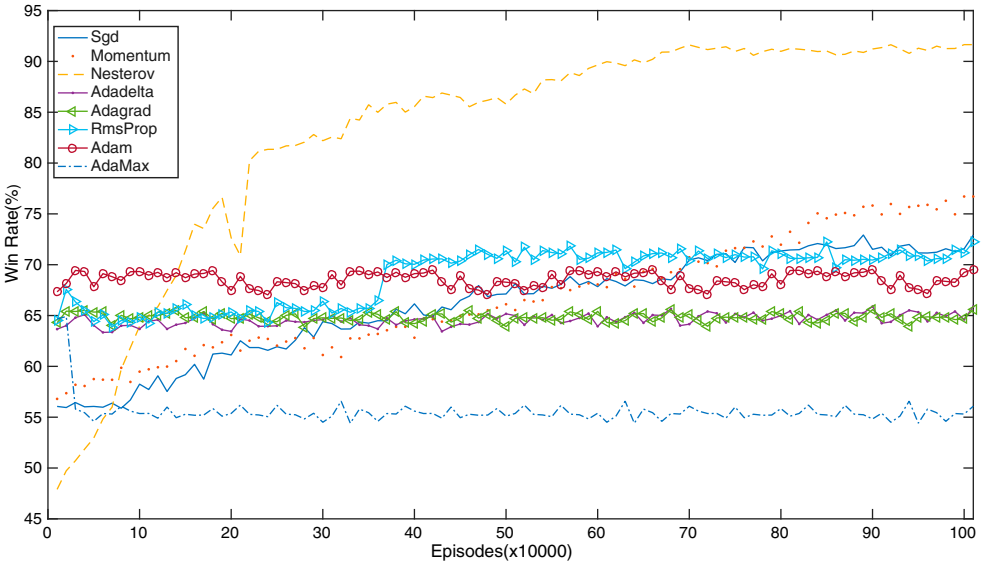

Figure 4. Comparison of momentum techniques.

Table 2. 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
AdaMax	64.91

training process, Nesterov momentum increased the performance slowly and regularly (Figure 3). Different techniques provided different performance values between 64.91 and 91.65. The maximum performance values reached by the techniques are shown in Table 2.

5. Conclusion

Deep learning has become a very popular tool for many different scientific areas lately. Back-propagation with simple gradient descent is mostly chosen for the training of deep structures due to its stability, but it may 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 non-profitable during the reinforcement learning training process. Therefore, the momentum technique to be used must be flexible enough to accommodate this change in the policy and yet it should accelerate the training process.

In this paper, two separate experiments were carried out. The first one was aimed to find the appropriate deep neural network structure to be used as a policy function for the Othello game and the second one was aimed to determine the most successful momentum technique for this process. The first experiment showed that increasing the number of convolutional layers or the number of filters of a deep neural network may improve its performance for reinforcement learning problems. In the second experiment, seven different momentum techniques and also simple gradient descent without momentum were compared with the Othello game benchmark. In contrast to methods such as Adagrad, Adadelta, RMSprop, Adam and AdaMax 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 100,000 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 increasing the number of convolutional layers or the number of filters of a deep neural network may be tried to increase the performance of the neural network for a reinforcement learning problem and additionally, 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 with online reinforcement learning methodology.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

M. Sarigül http://orcid.org/0000-0001-7323-6864

Notes on contributors

Mehmet Sarigül, is complete his undergraduate education in Computer engineering department at Cukurova university. He is working as research assistant in computer engineering department at Cukurova university. Email: msarigul@cu.edu.tr

Mutlu Avci, is completed his undergraduate and graduate education in Electrical and Electronics Engineering and Ph.D. in 2005 in Electronics and Communication Engineering. He is working as an associate professor in the Department of Biomedical Engineering at Cukurova University. His research areas are artificial intelligence, microelectronics, analog and mixed VLSI. Email: mavci@cu.edu.tr

References

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. Proceedings of the 25th international conference on machine learning (pp. 160–167). Helsinki: ACM.

Duchi, J., Elad, H., & Yoram, S. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

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

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition, Colombus, OH.

Kingma, D., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097–1105), Lake Tahoe, Nevada, USA.

Lai, M. (2015). *Giraffe: Using deep reinforcement learning to play chess*. arXiv preprint arXiv:1509.01549.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence O (1/k2). *Doklady Akademii Nauk SSSR*, 269(3) pp. 543–547.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12* (1), 145–151.

Ronan, Clement, Koray, & Soumith. Retrieved from http://torch.ch/

Sarigül, M., & Avci, M. (2017a). Comparison of different deep structures for fish classification. International conference on computer and information technology, Kuşadası.

Sarigul, M., & Avci, M. (2017b). Performance comparison of different momentum techniques. IEEE international conference on INnovations in Intelligent SysTems and Applications (INISTA), Gdynia.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., & Hassabis, D., 2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1, No. 1)Cambridge: MIT Press.

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

Tieleman, T., & Geoffrey, H. (2012). *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. Coursera: Neural Networks for Machine Learning 4.2.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3–4), 229–256.

Yakovenko, N., Cao, L., Raffel, C., & Fan, J. (2016). *Poker-CNN: A pattern learning strategy for making draws and bets in poker games using convolutional networks*. AAAI.

Zeiler, M. D. (2012). *ADADELTA: An adaptive learning rate method*. arXiv preprint arXiv:1212.5701.