Research Article

Improved Feature Learning: A Maximum-Average-Out Deep Neural Network for the Game Go

Xiali Li ¹, Zhengyu Lv ¹, Bo Liu ¹, Licheng Wu,¹ and Zheng Wang²

¹*School of Information Engineering, Minzu University of China, Beijing 100081, China*

²*Department of Management, Taiyuan Normal University, Shanxi 030619, China*

Correspondence should be addressed to Xiali Li; xiaer_li@163.com

Received 30 January 2020; Revised 5 March 2020; Accepted 16 March 2020; Published 9 April 2020

Guest Editor: Michael Z. Q. Chen

Copyright © 2020 Xiali Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Computer game-playing programs based on deep reinforcement learning have surpassed the performance of even the best human players. However, the huge analysis space of such neural networks and their numerous parameters require extensive computing power. Hence, in this study, we aimed to increase the network learning efficiency by modifying the neural network structure, which should reduce the number of learning iterations and the required computing power. A convolutional neural network with a maximum-average-out (MAO) unit structure based on piecewise function thinking is proposed, through which features can be effectively learned and the expression ability of hidden layer features can be enhanced. To verify the performance of the MAO structure, we compared it with the ResNet18 network by applying them both to the framework of AlphaGo Zero, which was developed for playing the game Go. The two network structures were trained from scratch using a low-cost server environment. MAO unit won eight out of ten games against the ResNet18 network. The superior performance of the MAO unit compared with the ResNet18 network is significant for the further development of game algorithms that require less computing power than those currently in use.

1. Introduction

Deep learning and reinforcement learning are increasingly being used in developing algorithms for playing various games such as backgammon, Atari, Shogi, chess, Go, StarCraft, Texas poker, and mahjong, where the artificial intelligence (AI) program applying the algorithms has reached or even surpassed the performance level of top human players. However, large-scale deep neural networks consume a lot of computing power in game search processes and training of the network, so huge amounts of computing power are required to run these programs.

Tesauro et al. first used temporal difference (TD) reinforcement learning algorithms [1, 2] and backpropagation neural networks [3] to achieve superhuman performance levels in backgammon, demonstrating that deep learning can be effectively combined with reinforcement learning algorithms. Various game programs, including the AlphaGo [4] series, AlphaGo Zero [5], Alpha Zero [6], Alpha Star [7],

Libratus [8], Pluribus [9], AI Suphx [10], and ELF Go [11] all use deep neural networks (DNNs) as their prediction model. These programs have beaten human players, thereby demonstrating the application prospects of deep convolution neural networks (CNNs) in the gaming field. However, such networks have many parameters, resulting in high computing requirements and slow training processes. Often, the game problems are solved using a distributed cluster or even a dedicated chip, such as a tensor processing unit. For small games like the Atari series games, including Atari using DQN (Deep Q Network) [12], C51 (51-atom agent) [13], QR-DQN (Distributional Reinforcement Learning with Quantile Regression) [14], HER (Hindsight Experience Replay) [15], TD3 (Twin Delayed Deep Deterministic) [16], DDPG (Deep Deterministic Policy Gradient) [17], SAC (Soft Actor-Critic) [18], A2C/A3C (Asynchronous Actor-Critic, Asynchronous Advantage Actor-Critic) [19, 20], TRPO (Trust Region Policy Optimization) [21], and PPO (Proximal Policy Optimization) [22], the end-to-end learning mode is

achieved using a DNN to learn from video inputs. In game research, most studies focus on the efficiency of the reinforcement learning algorithm itself but neglect the problems of neural network adaptation, selection, and design in the deep reinforcement learning algorithm. To achieve a powerful machine learning solution on a personal computer and enable the deep reinforcement learning algorithm to be extended to equipment with low computing power for fast training, we need a method that can achieve high performance with reduced computing power consumption.

In order to improve the performance of DNN [23–26] in the deep reinforcement learning algorithm, this paper presents a method to improve the learning speed and reduce the required parameters by optimizing the network structure. We propose a convolution layer-based network structure that reduces the overall computational cost by reducing the number of iterations executed with a higher rate of learning the signature graph on the network. The network structure uses a maximum-average-output (MAO) layer instead of a partial convolution layer in 17 layers of CNN. By randomly selecting the input channels during the training process, different convolution cores have the opportunity to learn different features and have better generalization ability and learning efficiency.

First, this study provides another way to optimize the deep convolution neural network. The output of this layer is based on the principle of the piecewise function [23]. It has different ideas in improving the deep convolution neural network from the improvement of ELM [25, 26]. In this paper, approaching the nonlinear function by piecewise linear function, the MAO layer has less computing power than some convolution layers using activation functions such as tanh or sigmoid functions and has better fitting performance than those using the ReLU activation function. In addition, MAO reduces computation latency by learning more features at each level to reduce the number of required layers. The network model is based on a full convolution network and therefore has fewer parameters, which allows it to run and train on a graphics processor with less memory than traditional programs.

Second, this study verifies that improving the learning efficiency of the network can also help to enhance the efficiency of the learning algorithm. The deep reinforcement learning model mentioned in this article uses the same reinforcement learning algorithm as AlphaGo Zero. By comparing the structure of ResNet18 [24] and MAO network used in the benchmark control group, we directly compare the learning effect of MAO and ResNet18 networks when they have the same affiliation trained in the game.

2. Maximum-Average-Out Deep Neural Network

The reinforcement learning method used in this study was a DQN. The general neural network needs a large quantity of data to support learning features through the DNN. Deep reinforcement learning needs significant search space and calculation time to generate a sufficient number of samples and requires multiple iterations to learn the corresponding

features. The MAO layer helps the convolution layer learn different feature maps by randomly dropping out [27] some feature maps during training, thus improving the efficiency of the convolution layer learning features. This principle is similar to that of the dense layer in the Maxout network [23]. By increasing the number of neurons in a single layer to increase the number of feature maps, a segmented function is formed to improve the learning efficiency of the feature map in this layer. The number of learning features within each training neural network is increased by selecting the feature map output with the maximum average value of these feature maps. The results calculated by the convolution layer in the structure can play a more significant role in the convolution layer than can be accomplished using the Maxout function directly. This makes the network learning more robust, indirectly reducing the neural network learning time and therefore reducing the total reinforcement learning time.

2.1. Structure of Maximum-Average-Out Unit. The MAO unit input port is a dropout-controlled channel selector where the input is n feature maps. The input selector is only working in the training stage, while the prediction step is in the all-pass state. The input data are M_0 , which is an $l \times m \times n$ tensor. The channel mask in training is randomly generated by a dropout function. The mask $V_{1 \times 1 \times n}$ is a $1 \times 1 \times n$ tensor, while in prediction it is a tensor of $V_{1 \times 1 \times n}$ for all elements. The selection of the d -th channel in the selected feature map M_0 is calculated using the following formula, and the symbol “:” represents the whole dimension of the referenced tensor:

$$M_1(:, :, d) = M_0(:, :, d) \times V(:, :, d). \quad (1)$$

The selected feature map M_1 is calculated with a size of $l \times m \times n$ in the convolution layer where the depth of M_1 is k and the size of the convolution kernel C_k is $3 \times 3 \times k$. The feature map M_2 is obtained using

$$M_2 = M_1 * C_k, \quad (2)$$

where the size of M_2 is $(l-2) \times (m-2) \times n$. The average vector V_{avg} of the d -th channel is calculated using

$$V_{\text{avg}}(1, 1, d) = \text{avg}(M_2(:, :, d)). \quad (3)$$

The size of V_{avg} is $1 \times 1 \times k$, and the maximum value v_{max} of the average vector V_{avg} is calculated using

$$v_{\text{max}} = \max(V_{\text{avg}}(1, 1, k)). \quad (4)$$

The selection mask vector V_{select} is given by

$$V_{\text{select}}(1, 1, d) = V_{\text{avg}}(1, 1, d) \geq v_{\text{max}}? 1: 0. \quad (5)$$

Through the selection mask vector, the feature map with the maximum average value is selected for

FIGURE 1: MAO unit.

output. The calculation is then divided into the following 2 steps:

- (i) Select the feature map M_2 so that M_3 can be obtained by using the selection mask V_{select} , as shown in 6. The elements in the selected feature map are unchanged, while the elements in the unselected feature map are assigned values of 0.

$$M_3(:, :, d) = M_2(:, :, d) \times V_{\text{select}}(1, 1, d). \quad (6)$$

- (ii) M_3 merges its channels by adding the feature maps (see (7)), where M_{out} represents the feature maps selected by the segmented function.

$$M_{\text{out}}(:, :) = \sum_d M_3(:, :, d). \quad (7)$$

A schematic diagram of the MAO unit of any given channel is shown in Figure 1. The channel selector is a $1 \times 1 \times n$ convolution layer, which is input into the channel selector from the upper-level feature map. During training, a group of selection vectors is generated by a dropout function and then input into the convolution to control the input of different feature maps. During prediction, however, the channel selector controlled by a dropout function will allow a feature map to pass through. The convolution layer of depth k is selected from the channel. There is no activation function in this layer for the feature input to the selector. By calculating the average value of each feature map, up to a vector of length k , the channel with the maximum average value is selected to output the feature map.

This structure is improved on the basis of convolution layer, referring to the construction idea of Maxout [23], and using dropout to select feature map in the training stage. This structure makes the convolution layer reduce the similarity between convolution kernels when learning features so as to improve the utilization of convolution kernels and increase the number of convolution layer learning features in this structure.

The MAO layer is composed of multiple channels, each of which contains multiple MAO units, as shown in Figure 2. Its use is similar to the convolution layer using rectified linear unit (ReLU).

FIGURE 2: MAO layer.

2.2. Overall Network Structure. The overall structure of the network is a full CNN with 19 layers. In this network, the MAO block (Figure 3) is composed of a MAO layer and mlpconv [23] layers. A mlpconv layer is a 2-layer CNN that is abbreviated by multilayer perceptron convolution. To ensure fewer parameters after using MAO layer, we combined MAO layer with 2 mlpconv blocks to form a MAO block. By using this structure, whole networks can be deepened, yielding superior generalization abilities and the capacity to learn more features, primarily by MAO layers. The convolution kernel size was 3×3 , and the activation function was ReLU.

In this paper, the input of the MAO network (Figure 4) is a state feature map represented by a $19 \times 19 \times 3$ tensor, which is input through the first mlpconv block [23] with a hidden layer width of 80 and then transferred to the middle MAO block. There are two common MAO blocks in the middle of the network, one policy and one value piece, followed by the output layer consisting of a MAO layer and GAP (global average pooling) layer. A MAO block with a width of 40 was used along with four groups of MAO layers. The MAO layer of the output layer is a group of outputs, and the width of the strategy portion is 362. After the GAP calculation, a vector p with a length of 362 is calculated, and the width of the board value estimation portion is 1. After the GAP, the output becomes a scalar value, and tanh is added as its activation function to output the scalar v . Using the full CNN, the generalization ability is ensured with fewer parameters, and the training speed and computational power can be reduced compared to other methods.

3. Results

In this study, programs using either the MAO or ResNet18 networks, combined with the AlphaGo Zero reinforcement learning algorithm, were developed and their training time

FIGURE 3: MAO block.

FIGURE 4: MAO network.

and parameters were compared using the experimental conditions shown in Table 1. The programs using the MAO and ResNet18 networks are henceforth referred to as the MAO program and the ResNet18 program, respectively.

In order to reduce the work caused by debugging superparameters and make comparison with ResNet18, the learning rate and minibatch in this paper adopt the parameters mentioned in paper [23, 24] and make fine adjustment within a reasonable range, that is, learning rate = 0.001 and minibatch = 128.

After seven days of training, the ResNet18 and MAO networks had learned from scratch through 60 games of self-play. Although the training time was limited, the results for the MAO program self-play still showed strong attributes. For example, in 10 games against the ResNet18 program, the MAO program took the opposing stones first. The networks were then pitted against each other in 10 games of Go, with the results shown in Table 2. Note that, in Table 2, the label “+Resign” indicates that the player won by resignation of the opponent. ResNet18 and MAO network models in deep reinforcement learning (DRL) verified a distinct effect in playing Go, so that faster DNN learning corresponded with faster DRL learning.

Because of its high learning efficiency, our MAO neural network learned more features than the ResNet18 network while utilizing fewer parameters, as evidenced by the total byte usage of 2 MB and 500 MB, respectively. Therefore, our network is expected to show higher reinforcement learning efficiency than the ResNet18 network. In the ten games played between the two different programs, the eyes which are important shape for Go game between the winner and loser was only 0.5 to 11.5. However, the MAO program won 8 games, while the ResNet18 program won only 2 games.

TABLE 1: Computer configuration.

Toolkit package	CNTK 2.7
Runtime	.Net 4.7.2
OS	Windows 10
CPU	AMD 2700X@4.0 GHz
RAM	32 GB
GPU	RTX2070
Threads used	16
Development environment	Visual studio 2017 community
Communicate environment	GTP + sabaki v0.43.3
Optimization function	Stochastic gradient descent (SGD)
Learning rate	0.001
Minibatch	128
Epoch	10

TABLE 2: Test results.

Comparison	Black side	White side	Result
1	MAO	ResNet18	B + resign
2	ResNet18	MAO	W + resign
3	MAO	ResNet18	B + resign
4	ResNet18	MAO	W + resign
5	MAO	ResNet18	W + resign
6	ResNet18	MAO	W + resign
7	MAO	ResNet18	B + resign
8	ResNet18	MAO	W + resign
9	MAO	ResNet18	B + resign
10	ResNet18	MAO	B + resign

This indicates that the program learned the rules and game patterns more quickly using the MAO network.

Figure 5 shows snapshots of Go boards during some sample games. Specifically, Figure 5(a) shows 2 stones,

FIGURE 5: Go boards during games between programs using the ResNet18 and MAO algorithms. In this figure, the numbers on the stones indicate the steps that each player plays and red boxes highlight the stones, represented by a number without a stone present, taken by the opponent in the enclosed area. (a) MAO (B) vs. Resnet18 (W), (b) Resnet18 (B) vs. MAO (W), and (c) Resnet18 (B) vs. MAO (W).

indicated by the red box, being taken by the black pieces, which in this case was the MAO program. Figure 5(b) shows one stone, indicated by the red box, being taken by the white player, which in this case was the MAO program. Contrarily, Figure 5(c) shows one stone, indicated by the red box, being taken by the black side, which in this case is the ResNet18 program. Figure 5 clearly demonstrates that the more efficient MAO program can learn the value of eating the opposing pieces more quickly than the ResNet18 program could, and it had a relatively high probability for using similar moves to force the opponent to resign in the middle of the game (winning by resignation). In addition, the MAO program seemed to apply the concept of occupying the board in the Go game more effectively than the ResNet18 program did and thus put more pressure on its opponent. This indicates that the neural network can learn to play for superior board status early in the game. From these results, we can conclude that deep reinforcement learning can be improved by pruning the search algorithm and state space or by using better expert knowledge to guide the early training and achieve rapid improvement in game-play performance. Furthermore, learning more features in a shorter time implies that the program can learn to prune the search algorithm and state space faster and thus generate knowledge to guide moves in subsequent rounds of self-play.

Figure 5(a) shows two stones (red box) taken by the black side when MAO (B) vs. ResNet18 (W) is playing chess, Figure 5(b) shows one stone (red box) taken by the white side when ResNet18 (B) vs. MAO (W) is playing chess, and Figure 5(c) shows one stone (red box) taken by the black side when ResNet18 (B) vs. MAO (W) is playing chess. As shown in Figure 5, the more efficient MAO can learn the value of eating more quickly than ResNet18, and it has a relatively high probability to use this similar move to make the opponent give up in the middle stage (relatively more purposeful), and it can learn better the concept of land occupation in go game and put pressure on the opponent.

4. Conclusion

This study explores the possibility of improving DNN structures to increase the efficiency of deep reinforcement learning for application in Go programs. Through the comparative experiments presented in this paper, this is shown to be a feasible and effective scheme for speeding up the deep reinforcement learning algorithm by improving the DNN. Moreover, the MAO network proposed in this paper also demonstrates its ability to improve the feature learning efficiency by controlling the channel input of the convolution layer during the training process. This work further shows that the improvement of the learning efficiency of deep reinforcement learning algorithms can be achieved by pruning the search space and improving the learning efficiency of the neural network. Notably, the latter is easier to achieve than the former. In the future, we intend to continue to perform experiments and obtain more data to further verify the superior performance of the proposed network, by applying this proposed network structure to image processing and other fields to verify and demonstrate its generalizability. On the other hand, we will consider to use multiagent [28, 29] constructure to implement and improve the efficiency of the proposed MAO network.

Data Availability

Data are available via the e-mail lzywindy@163.com.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was funded by the National Natural Science Foundation of China (61873291 and 61773416) and the MUC 111 Project.

References

- [1] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [2] G. Tesauro, *Practical Issues in Temporal Difference Learning*, Springer, vol. 8, pp. 257–277, Berlin, Germany, Berlin, Germany, 1992.
- [3] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.
- [4] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [5] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [6] D. Silver, T. Hubert, J. Schrittwieser et al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [7] V. Zambaldi, D. Raposo, S. Adam et al., "Relational deep reinforcement learning," 2018, <http://arxiv.org/abs/1806.01830>.
- [8] N. Brown and T. Sandholm, "Safe and nested subgame solving for imperfect-information games," in *Advances in Neural Information Processing Systems*, pp. 689–699, MIT Press, Cambridge, MA, USA, 2017.
- [9] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.
- [10] msra, <https://www.msra.cn/zh-cn/news/features/mahjong-ai-suphx>, 2019.
- [11] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. Lawrence Zitnick, "Elf: an extensive, lightweight and flexible research platform for real-time strategy games," in *Advances in Neural Information Processing Systems*, pp. 2659–2669, MIT Press, Cambridge, MA, USA, 2017.
- [12] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning. computer science," 2013, <http://arxiv.org/abs/1312.5602>.
- [13] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 449–458, Sydney, Australia, August 2017.
- [14] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, February 2018.
- [15] M. Andrychowicz, W. Filip, A. Ray et al., "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, pp. 5048–5058, MIT Press, Cambridge, MA, USA, 2017.
- [16] F. Scott, Herke van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, <http://arxiv.org/abs/1802.09477>.
- [17] T. P. Lillicrap, J. J. Hunt, P. Alexander et al., "Continuous control with deep reinforcement learning," 2015, <http://arxiv.org/abs/1509.02971>.
- [18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, <http://arxiv.org/abs/1801.01290>.
- [19] M. Babaeizadeh, I. Frosio, T. Stephen, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," 2016, <http://arxiv.org/abs/1611.06256>.
- [20] V. Mnih, A. P. Badia, M. Mirza et al., "Asynchronous methods for deep reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, pp. 1928–1937, New York, NY, USA, February 2016.
- [21] John Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 1889–1897, Lille, France, February 2015.
- [22] John Schulman, W. Filip, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, <http://arxiv.org/abs/1707.06347>.
- [23] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, <http://arxiv.org/abs/1312.4400>.
- [24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, December 2016.
- [25] X. Luo, J. Sun, L. Wang et al., "Short-term wind speed forecasting via stacked extreme learning machine with generalized correntropy," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4963–4971, 2018.
- [26] X. Luo, Y. Li, W. Wang, X. Ban, J.-H. Wang, and W. Zhao, "A robust multilayer extreme learning machine using kernel risk-sensitive loss criterion," *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 1, pp. 197–216, 2020.
- [27] P. Baldi and P. Sadowski, "The dropout learning algorithm," *Artificial Intelligence*, vol. 210, pp. 78–122, 2014.
- [28] B. Liu, N. Xu, H. Su, L. Wu, and J. Bai, "On the observability of leader-based multiagent systems with fixed topology," *Complexity*, vol. 2019, Article ID 9487574, 10 pages, 2019.
- [29] H. Su, J. Zhang, and X. Chen, "A stochastic sampling mechanism for time-varying formation of multiagent systems with multiple leaders and communication delays," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 12, pp. 3699–3707, 2019.