2016 International Symposium on Nonlinear Theory and Its Applications,
NOLTA2016, Yugawara, Japan, November 27th-30th, 2016

Efficient Board Feature Extraction for Strategy Improvement in Computer Go

Hayato Mitsuoka† and Koujin Takeda†

†Department of Intelligent Systems Engineering, Ibaraki University
4–12–1, Nakanarusawa Hitachi, 316-8511 Ibaraki, Japan
Email: 15nm925f@vc.ibaraki.ac.jp, koujin.takeda.kt@vc.ibaraki.ac.jp

Abstract—Recently, significant progress was made in computer Go by deep learning. However, huge computer resource is required for achieving professional player's skill at present, which seems over-engineering for feature extraction on Go board. From this background, we discuss how to construct an efficient feature extraction method on Go board under deep learning framework. By making use of knowledge on image recognition by deep learning, we propose a method to reduce computational cost of board feature extraction without degrading Go playing performance.

1. Introduction

In artificial intelligence study, several famous board games have been intensively investigated over years, because the conquest of them by artificial intelligence is thought to be a milestone. Regarding chess or Shogi, computer playing programs have already been able to compete or beaten most proficient professional players.

Go is known as the most difficult board game for artificial intelligence, due to a huge number of possible combination of moves and the equivalence of each stone value unlike chess or Shogi. Nevertheless, steady progress in computer Go program has been made for decades. Monte Carlo tree search[1, 2] is one of the important ideas for the progress, which enables to find an appropriate next move by pruning unnecessary branch of moves on the game tree after self-random play by computer. Even with such progress, the best computer Go program could not compete with any professional Go player at all, at the stage of several years ago.

In this year, quite significant breakthrough was made by AlphaGo[3] by Google DeepMind. Their success was achieved with deep learning. Incorporation of deep learning into Go programming has also been attempted formerly by other groups[4, 5, 6] (a pioneering work with this strategy is found in [7]), and in AlphaGo they also implemented very deep neural network with a huge number of neurons like other studies. Then their program finally beat a professional player. Furthermore, by self-reinforcement learning, AlphaGo surprisingly won against the most outstanding Go player in the world, which means that the technology of the best computer Go program at last surpasses the human player strategy.

However, in AlphaGo, there remains computational cost problem. In AlphaGo they used deep convolutional neural network (DCNN), in which there are about millions of neurons on layers and kernels between layers. For the operation of their DCNN, thousands of CPUs and hundreds of GPUs are required in the whole hardware, which cannot be prepared in personal use. In DeepMind's project, their final objective of deep neural network study is the realization of artificial intelligence for general purpose use, and their hardware specification may be over-engineering for computer Go.

In this article, we revisit the structure of deep neural network in computer Go, and discuss how to simplify it without degrading Go playing performance. As stated, DCNN is implemented in AlphaGo program, which is in general used mainly in image recognition. This fact suggests that the "picture" of black/white stone pattern plays an important role for human's decision of next move. In image recognition with DCNN, various ideas are proposed for reduction of computational cost. DropConnect[8] is one of such ideas, where a fraction of edges are disconnected. Accordingly, it enables us to reduce the computational cost in learning and to avoid overfitting problem. We apply DropConnect to computer Go program, and discuss how it affects Go playing performance by the experiment of next move prediction. If DropConnect does not highly degrade the performance of next move prediction, we will be able to reduce the computational cost of DCNN without paying high price. As a consequence, we expect to obtain a key to the method of efficient board feature extraction by DCNN.

2. Model

DCNN is one of multi-layered neural networks, which is mainly used for image recognition. DCNN consists of multiple convolutional/pooling layers, which are alternately layered. Kernel filters are defined on edges between layers. By the operation of convolution as in Figure 1, the system naturally handles overlapped information, which enables us to cope with defect or displacement in original image appropriately and leads to more successful image recognition.

DCNN is also used for board feature extraction on Go board. In computer Go, basic input information is stone pattern on board, and additional set of tactical board information is also often taken as input. The objective of board feature extraction is the correct next move predic-

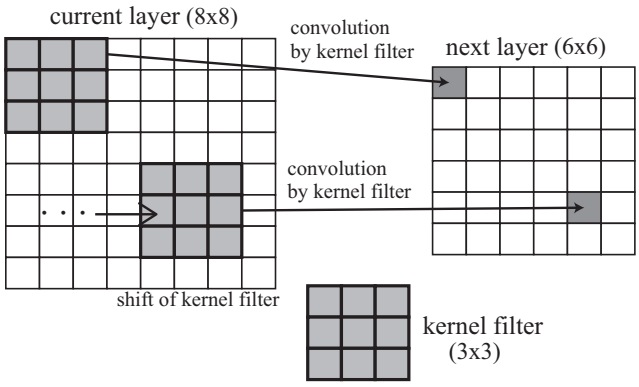

Figure 1: An example of convolutional operation between layers by kernel filters: In this example, the size of input information is 8×8. The size of kernel filter is 3×3. By operation of convolution, output information on the next layer is 6×6.

tion, where "correct" indicates professional player's move. In DCNN in computer Go, the dimension of input information for single input channel is usually the same as the board size, i.e. 19×19 for standard 19×19 board. In general we prepare multiple input channels, and input different information into each channel: for example black or white stone pattern, information of connected strings(="ren"), information of "liberty"(= vacancy adjacent to a point or connected string), information of player's skill and so on.

In AlphaGo, they use DCNN with 13 layers and 3×10^7 board pattern data for training. In their move prediction experiment, the accuracy of move prediction improves as the number of filters(=edges) increases. However, in actual Go game it is reported that neural network with less filters gives higher game winning rate. This is due to the computational cost: For larger neural network, the evaluation of next move prediction is more time-consuming, which is the problem in actual Go game with limited playing time. Then we reduce the computational cost by simplifying the network not to highly worsen the move prediction performance.

3. Method

3.1. Architecture of DCNN in our experiment

Even if we simplify the structure of neural network, training of DCNN requires much time for standard 19×19 board. Therefore, in this work we restrict ourselves to the case on small 9×9 board.

The network structure in our framework is shown in Figure 2. For a single channel input, we prepare 11×11 size image for 9×9 go board including boundary of width one. We prepare 3 input channels totally; for black stone pattern, white stone pattern, and the information of board boundary.

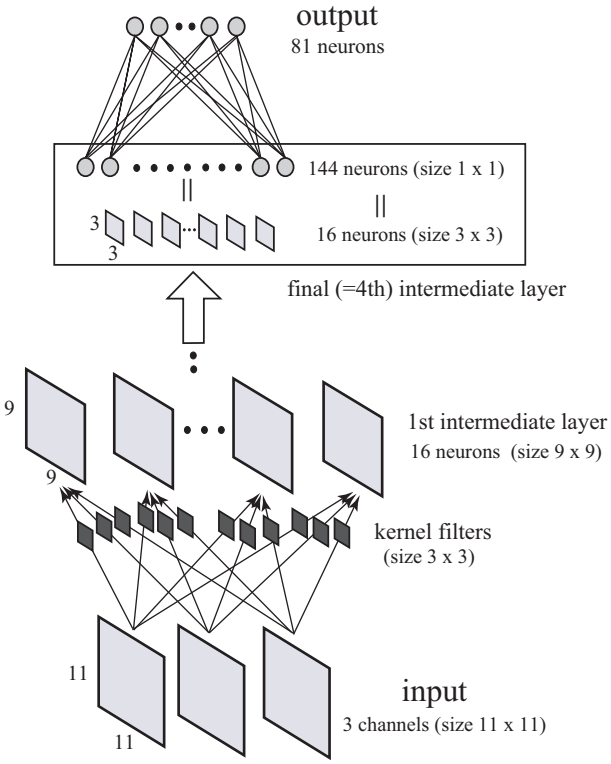

Figure 2: Our DCNN architecture: We show 4-layer DCNN. We input 3 channels with 11×11 size including boundary; black/white stone pattern and board boundary. The size of neuron gradually decreases as 9×9 (1st) $\rightarrow$ 7×7 (2nd) $\rightarrow$ 5×5 (3rd) $\rightarrow$ 3×3 (4th) by the operation of convolution. The network between final intermediate layer and output layer is fully connected network of 1×1 size neurons. The output layer has 81 neurons for next move prediction on 9×9 board.

Then we propagate input information to the neuron on the next layer by the operation of kernel filter and activation function. We use DCNN with 2 or 4 intermediate layers, where all layers are convolutional layers. We do not use pooling layer for feature extraction, as pooling layer is not included in DCNN in preceding works of computer Go. On each intermediate layer 16 neurons are located. We always use 3×3 size kernel filter. Accordingly, on final intermediate layer we put 16 neurons with 7×7 size for 2-layer DCNN, and with 3×3 size for 4-layer DCNN. Finally, for propagation to output layer we divide these neurons into pieces of 1×1 size, and make fully-connected network with output layer. Output layer consists of $81(=9 \times 9)$ neurons of 1×1 size, which describes the position on board. The neuron with the largest value is taken as the result of next move prediction.

3.2. Learning algorithm

As stated, our objective is next move prediction, and we expect DCNN to output as precise next move position as

possible by training.

We follow the scheme of standard supervised learning on multi-layer neural networks. We prepare stone pattern on board in actual game as input, and corresponding next move as output. For training we must determine a loss function to minimize, and we choose mean squared error as the loss function E,

$$E := \sum_n E_n = \sum_n \left(\sum_k \frac{1}{2}(y_{nk} - t_{nk})^2 \right). \quad (1)$$

In the above definition subscript n represents the label of training data, and k the label of output neuron. y_{nk} is the output signal from DCNN and t_{nk} is the correct signal of training data, which indicates the correct next move. E_n is the loss function for data with label n. As the activation function $h(x)$ for forward propagation, we use rectified linear function,

$$h(x) := \max\{0, x\}. \quad (2)$$

After having output y_{nk} by forward propagation, we change values of kernel filter entries by standard back propagation algorithm for minimization of E. Here we apply stochastic gradient descent method with Hadamard product (AdaGrad) for back propagation.

$$g_{ij}^{(t+1)} = g_{ij}^{(t)} + \left(\frac{E_n}{\partial W_{ij}} \right)^2,$$
$$W_{ij}^{(t+1)} = W_{ij}^{(t)} - \frac{\alpha}{\sqrt{g_{ij}^{(t+1)}}} \frac{E_n}{\partial W_{ij}}. \quad (3)$$

$W_{ij}^{(t)}$ is the weight of kernel filter between neuron i on an intermediate layer and neuron j on the next layer at tth iteration. Note that $W_{ij}^{(t)}$ is the matrix(=kernel filter) and the subscript of matrix element is omitted. $g_{ij}^{(t)}$ is an auxiliary variable at tth step for computation of $W_{ij}^{(t)}$. α is learning rate and $\alpha = 5 \times 10^{-3}$ in our experiment.

In equation (3), we must calculate the derivative $\partial E_n / \partial W_{ij}$ analytically, as E_n is expressed as a function of W_{ij}. This can be done by standard back-propagation formulation.

3.3. DropConnect

DropConnect[8] is a method of training by regularizing neural network with a huge number of edges. In DropConnect, we randomly select a fraction of edges and set them zero, namely we disconnect them. By DropConnect, reduction of overfitting is expected in general, when we remove an appropriate fraction of edges. As a result, we can reduce the computational cost of forward/backward propagation.

Our key idea of this work is; we apply this method to move prediction in Go to reduce computational cost and to avoid overfitting.

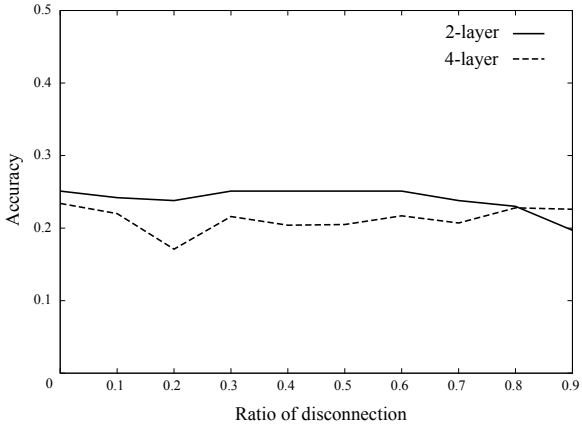

Figure 3: Dependence of next move prediction accuracy on the ratio of disconnection.

4. Experiment

4.1. Programming environment and dataset

For programming of DCNN we use tiny-cnn[9], which is a C++ library for deep learning. As training/test dataset for next move prediction experiment on 9×9 board, we use "Igo Quest"[10] dataset by 50 best rating players among all. We assign 7500 games (including 362508 board patterns) to the training data, and 116 games (including 51616 patterns) to the test data for evaluation of next move prediction accuracy.

4.2. Move prediction accuracy (1): dependence on DropConnect and intermediate layer depth

We evaluate next move prediction accuracy under DropConnect by numerical experiment. In DropConnect, we remove a fraction of edges between 1st and 2nd intermediate layers (totally $16 \times 16 = 256$ edges), or equivalently kernel filters on corresponding edges are set to be null matrix. We vary the ratio of disconnection from 0 to 0.9 by 0.1 step. We also vary intermediate layer depth: DCNNs with 2 and 4 intermediate layers are used. Before evaluation of accuracy, we train DCNN with 30 training iterations.

The result is shown in Figure 3. First, the accuracy is almost constant with respect to the ratio of disconnection, and does not show clear fall-off both for 2- and 4-layer DCNNs. Even if we disconnect 90% of edges, the accuracy is almost the same as the fully-connected case. This indicates that the edges in the original fully-connected DCNN is redundant for 9×9 board feature extraction, many of which can be removed without degrading move prediction accuracy.

As for intermediate layer depth, the accuracy by 2-layer DCNN slightly surpasses 4-layer DCNN. This suggests that training by 4-layer DCNN may lead to slight overfitting, and yield poorer performance than 2-layer DCNN.

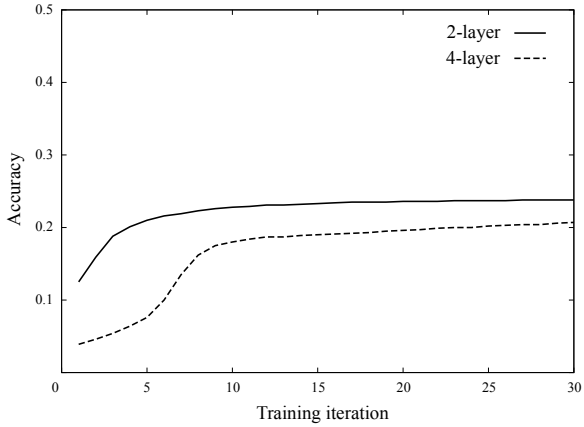

Figure 4: Dependence of next move prediction accuracy on the duration of training.

Therefore, under our experimental setup, we conclude that 2-layer DCNN is sufficient for next move prediction.

However, we should note that the value of accuracy by our experiment is much smaller than by AlphaGo. For next move prediction, AlphaGo yields accuracy of 0.57 for standard 19×19 board. Hence, we should keep in mind that DropConnect may worsen move prediction performance under different experimental setup, for example higher accuracy case.

4.3. Move prediction accuracy (2): dependence on training duration

We also study the dependence of accuracy on training duration under 2- and 4-layer DCNNs. The result is depicted in Figure 4, which shows the accuracy by 2-layer DCNN increases more rapidly and saturates earlier than 4-layer DCNN. From this result, we conclude that 2-layer DCNN is more suitable than 4-layer DCNN for fast training, which also helps the reduction of huge training time for DCNN.

We also vary the ratio of disconnection in this experiment, which do not affect the speed of training as a consequence.

5. Summary and Discussion

We discussed how to reduce the computational cost for training and move prediction by DCNN. Our result indicates that we can reduce the number of intermediate layers and edges(=kernel filters) without degrading performance of next move prediction, which leads to the reduction of computational cost.

As future works, we should verify how additional board information inputs affect the result. Furthermore, we should also study the dependence on learning algorithm: Improvement of move prediction accuracy might be possible when we replace the loss function with more appropriate one, or change AdaGrad to another gradient descent algorithm.

Experiment on standard 19×19 board is of course a remaining work. After having the result on standard board, we will combine our method with Monte Carlo tree search algorithm to verify playing performance in actual Go games.

Acknowledgments

This work is supported by KAKENHI Nos. 24700007, 25120013 (KT).

References

[1] B. Brugmann, ”Monte Carlo Go,” unpublished technical note, 1993.

[2] R. Coulom, ”Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search,” *Proc. of Intern. Conf. on Computers and Games*, 2006.

[3] D. Silver et al., ”Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol.529, pp.484-489, 2016.

[4] C. Maddison et al., ”Move Evaluation in Go Using Deep Convolutional Neural Networks,” *proc. of Intern. Conf. on Learning Representations*, 2015.

[5] C. Clark and A. Storkey, ”Teaching Deep Convolutional Neural Networks to Play Go,” *proc. of Intern. Conf. on Machine Learning*, pp.1766-1774, 2015.

[6] Y. Tian and Y. Zhu, ”Better Computer Go Player with Neural Networks and Long-Term Prediction,” *proc. of Intern. Conf. on Learning Representations*, 2016.

[7] I. Sutskever and V. Nair, ”Mimicking Go Experts with Convolutional Neural Networks,” *Proc. of Intern. Conf. on Artificial Neural Networks*, pp.101-110, 2008.

[8] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, ”Regularization of Neural Networks using DropConnect,” *proc. of Intern. Conf. on Machine Learning*, pp.1058-1066, 2013.

[9] https://github.com/nyanp/tiny-cnn

[10] Y. Tanase: Dataset by software distributed at the following URL, http://wars.fm/ja