Back Prediction in the Game of Go

Tang Jiachen[1,a)] Tomoyuki Kaneko[2,b)]

Abstract: The AlphaGo Zero series of algorithms have achieved superhuman performance in the game of Go by use of deep neural network trained by reinforcement learning. However, according to ELF OpenGo, the ability to solve with the inductive tactics, which human could get mastered quickly such as ladder moves, oscillates significantly over the training process, even with significant amount of computation. In this work, we propose a method which combines deep learning with back prediction to reconstruct the game state in the game of Go. With our techniques, we estimate on the ladder moves dataset and found that the back prediction network performs well. Although we so far have not tested on more datasets, we hope that our techniques hold promise for making the neural network more robust and future research of combining with the computer Go program.

Keywords: The Game of Go, Deep Learning, Back Prediction, Capsule Network

1. Introduction

In artificial intelligence, the game of Go is always viewed as the most challenging game partly because of the difficulty of evaluating the complex board positions and existing a large action space. Along with the history of computer Go's development, from program Zen to Alpha Go Fan [1], and finally AlphaZero [2], the artificial intelligence over-performed even the most powerful player in this area. With the development of computer Go, there comes many effective methods, such as tree search algorithms which CrazySto used, Monte-Carlo Tree Search (MCTS) [3], Neural Network (NN) and so on. In March 2016, Alpha Go Lee with deep neural network and tree search, which is trained by the combination of supervised learning from human expert games and reinforcement learning from games of self-play, defeated Lee Se-dol 4-1 in Seoul. After a year, a more powerful program, AlphaGo Zero, which solely depends on self-play reinforcement learning that is based on deep neural network from scratch, achieved state-of-the-art performance of winning 100-0 against the previously published Alpha Go without human data, guidance or domain knowledge.

According to David Silver et al. [4], AlphaGo Zero achieved state-of-the-art performance, however, reproduction of AlphaGo Zero algorithms requires large amounts of computation which is an unattainable level of computing resources for the majority of the research community, and according to ELF OpenGo's experiment [5], even at the expense of phenomenal computation, deep reinforcement learning models cannot perfectly master some basic tactics, such as ladder moves, which beginning human players can also master without difficulty, and ability of playing this kind of inductive move remains unstable status.

According to Hinton et al. [6], reconstruction can be regarded as a regularization method to improve neural network's robustness. In this paper, we introduce a neural network called Back Prediction Neural Network(BPNN) to reconstruct previous Go board state. Predicting previous move which needs important foresight is effortless when compared to calculating the next move of the best, and it purely requires small amount of computation and costs only several hours to train, therefore, we are hoping for solve the problem of fluctuation in learning inductive capacity with our method. We train and test the neural network on ladder scenario with several kinds of networks. we try to equipment the model with the ability of inductive reasoning, and the model has learned to predict inductive moves that require important foresight to some extent, however, we observe that there is high variance in the model's strength during training process, and this property holds over the training period. Nonetheless, it is promising that our techniques can serve as the seed for future work of combining with the AlphaGo style program to make a relatively robust agent.

2. Related Work

2.1 The game of Go

The game of Go is an adversarial two-player board game with the objective of occupying more territories of the board with one's stones than the opponent. As the game progresses, the players place stones on the board to map out formations and potential territories. Contests between opposing formations are often extremely complex and may result in the expansion, reduction, or wholesale capture and loss of formation stones. The board consists with

[1] Graduate School of Arts and Sciences, The University of Tokyo
[2] Graduate School of Interdisciplinary Information Studies, The University of Tokyo
[a)] tangjc@g.ecc.u-tokyo.ac.jp
[b)] kaneko@graco.c.u-tokyo.ac.jp

a plain grid of 19 horizontal and 19 vertical lines. The stones are supposed to be placed on the intersection of the board if the position is empty. Black player makes the first move, after which it is white player, and black player switches turn to their moves, and one player can choose to place a stone or pass his turn. When all the intersections directly adjacent to a stone of group of stones are occupied by the stones of the other color, they will be removed from the board . The game ends when both players choose to pass their turn. Players try to claim their territory by walling off sections of the board and surrounding each other's stones, and the winner determined by the score of final position. A player's score is calculated by all the positions which the player has either occupied or surrounded. According to the final scores of two players, the state decides the winner who has a higher score than the other. Traditionally, a bonus ("komi", most commonly 7.5) is given to white as compensation for going second [7]. There are different rule-sets (Japanese, Chinese, AGA, etc.), which are almost entirely equivalent, except for certain special-case positions.

2.2 Classical Search

In the first stage of game research for two-person zero-sum games with perfect information, classical search is widely applied on board games, such as Chess, and some programs have defeated professional human players even world champion in most classic games. Tree search is an important method of classical search that recursively computes the optimal value function in a search tree which consists of b^d possible moves, where b is the game's breadth (number of legal moves per position) and d is the tree depth (game length), in some games, exhaustive search is feasible so that these games are supposed to be solved by this method. Some classical search methods, such as minimax search augmented by alpha-beta pruning [8], also enjoyed initial success in the domain of board games. However, classical search is not a practicable method to lead to even amateur level performance in large games, especially the game of Go which is of great complexity.

2.3 Monte Carlo Tree Search (MCTS)

A Monte-Carlo evaluation consists in estimating a position by averaging the outcome of several random continuations. The method can serve as an evaluation function at the leaves of a search tree [9]. Monte Carlo Tree search (MCTS) is a significant search method in board games which combines tree search and Monte-Carlo evaluation [10]. Each search consists of a series of simulated games that traverse as tree from root s_{root} to leaf [11]. At each state, the value is determined by Monte-Carlo rollouts, and then it chooses child nodes of high values and nodes that have not been completely explored. With the simulation proceeding, the search tree becomes larger and the value of each state develops into more accurate. MCTS treats the tree search as an exploitation/exploration trade-off. A common exploitation-exploration heuristic is Upper Confidence bounds applied to Trees (UCT) algorithm. Concisely, UCT provides an exploration bonus proportional to the inverse square root of a game state's visit frequency. Nowadays, most Go AIs collaborate with MCTS to improve the model's strength.

2.4 AlphaGo Series Algorithms

AlphaGo series algorithms are significant methods of deep reinforcement learning, and they enjoy great success due to the superhuman performance.

2.4.1 AlphaGo

AlphaGo proposed a new approach which uses value network to estimate the board position and policy network to select moves, and these two deep neural networks are trained by the combination of supervised learning from human expert games and reinforcement learning from games of self-play [1]. The neural networks join force with Monte-Carlo Tree Search to achieve superhuman performance. With this method, AlphaGo is proved to be the best of the computer Go programs and defeated a human professional player on the 19×19 board for the first time.

2.4.2 AlphaGo Zero

Different from AlphaGo, AlphaGo Zero learns the game of Go from scratch, without human data, guidance or domain knowledge beyond game rules. Value network and policy network have been integrated into a single neural network and the neural network is trained to predict own move selections of AlphaGo and also the winner of self-play games, starting from random play. Through training solely by deep reinforcement learning, AlphaGo Zero outperforms AlphaGo with 100-0 match experiment. AlphaGo Zero achieves rapid improvement, precise learning and stable learning [4].

2.4.3 Contemporary Reproductions

Besides of AlphaGo Zero, many implementations of AlphaGo Zero series of algorithms with no extra data or guide, such as Leela Zero [12], ELF OpenGo and PhoenixGo [13], have made it available to the research community in recent years. These open source reproductions master superhuman skill, and these programs promote the research community to better understand and analyze many of considerations for large-scale deep reinforcement learning.

2.5 Capsule Network

2.5.1 Capsules

A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific of entity such as an object or an object part, and the length of the activity vector represents the probability of existing entity [6]. Capsule Network introduces a new method called dynamic routing between capsules, and by use of dynamic routing, capsules network achieves state-of-the-art performance on MNIST dataset and reaches a low test error previously only achieved by more complex networks.

Capsules are supposed to better represent hierarchical relationships inside of internal knowledge of features captured by neural network and learn more robust representations than traditional convolutional neural network.

2.5.2 Dynamic Routing

Capsule Network treats the length of vector as the probability of existing instantiation, therefore, vectors are supposed to be solved by a non-linear "squashing" function so that long vectors' length get diminished to a small degree which is below 1, and short vectors' length narrow down to almost zero.

$$v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2}\frac{s_j}{\|s_j\|} \tag{1}$$

where v_j is the vector output of capsule j and s_j is its total input.

Output u_i of capsule i in the layer below is multiplied by a weight matrix W_{ij} to produce a prediction vector $\hat{u}_{j|i}$, and the input to capsule s_j is represented by a weighted sum over prediction vector.

$$\hat{u}_{j|i} = W_{ij}u_i, s_j = \sum_i c_{ij}\hat{u}_{j|i} \tag{2}$$

where the c_{ij} are coupling coefficients calculated by the iterative dynamic routing process.

The coupling coefficients represent the log prior probabilities which capsule i should be coupled to capsule j, therefore, we should solve the initial logits b_{ij} with routing softmax function to make c_{ij} sum to 1.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \tag{3}$$

The coupling coefficients can be learned discriminatively at the same time as all the other weights.

Capsule layers outputs a local grid of vectors which represent different information of the neural network.

Algorithm 1 Routing algorithm.

1: procedure ROUTING($\hat{u}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: for r iterations do
4: for all capsule i in layer l : $c_{ij} \leftarrow \text{softmax}(b_i)$
5: for all capsule j in layer $l+1$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
6: for all capsule j in layer $l+1$: $v_j \leftarrow squash(s_j)$
7: for all capsule i in layer l and capsule in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$.
8: end for
9: return v_j
10: end procedure

3. Proposed Method

We will propose a neural network called back prediction network, and the network is trained to use the current board state to predict previous board state. Based on supervised learning method, we train the network on ladder scenario.

3.1 Data Representation

In recent years, deep convolutional neural network achieves superhuman performance across a wide range of domains, handwritten digits recognition, face recognition, image classification and so on. We treat the board of Go as an image, and employ convolutional neural networks to capture features of the position. We test two different kinds of input and output to train our network.

3.1.1 One-plane Representation

The input of the network is simply set to be a 19 ×19 image stack, and the output is a 361-dimension one-hot vector to represent the move from previous board state to the current board state.

3.1.2 Multi-plane Representation

We separate the board into 3 parts, a black board and a white board which illustrate the presence of the current player's stones, and a feature board which indicates the next player's color. We will take 5 board states as an example to explain the input of the network, as shown in Figure 1. The input to the back prediction network's shape is set to be as $19 \times 19 \times 11$, and this network consists of current position of white and black stones, 10 feature planes which represents the 5 previous states of black player and white player respectively, and a plane which determines the turn to move (if it is black player's turn to play, the plane is set to be all 1, otherwise 0), and the output is represented by a 722-dimension one-hot vector which indicates the move and the turn(if it is black player's turn to play, the value 1 is placed on the first 361-dimension location, otherwise the last 361-dimension location).

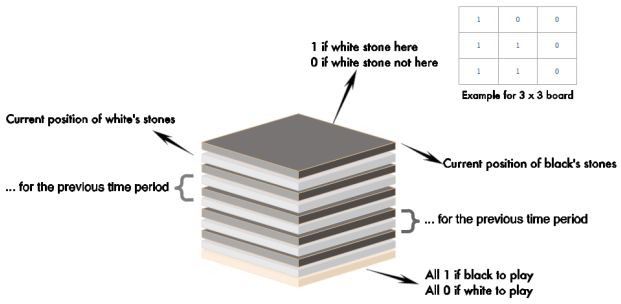

Fig. 1 Input: Input with board size × 11 stack

3.2 Architecture

3.2.1 Basic Architecture

As shown in Figure 2, We employ traditional convolutional neural network to train our model on ladder scenario. Our basic network is composed of 4 convolutional layers, and in order to stop the board size from becoming extremely small, we zero pad the input of every convolutional layer.Then two fully-connected layers are followed.

The first convolutional layer filters the 19 × 19 × num_states input image which represents the Go board with several consecutive board states, (The board image is from Sabaki [14]) with 48 kernels of size 7×7 with a stride

of 1 and ReLU activation function. The second convolutional layer takes the output of the first convolutional layer as the input, and the input is filtered by 32 kernels of size 5×5. The third and fourth layer function similarly with the second layer. The first fully-connected layers have 512 neurons with ReLU activation, and the second fully-connected layer generates the final output vector with softmax activation.

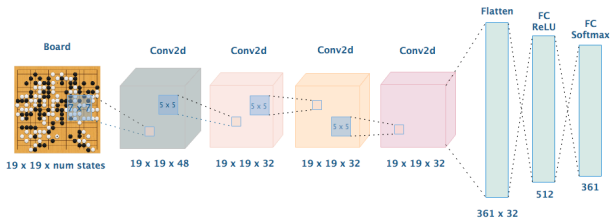

Fig. 2 Convolutional Neural Network Architecture

3.2.2 Additional Architecture

A simple capsule network architecture is shown in Figure 3. The network is combined with convolutional neural network. The architecture is shallow with one convolutional layer and one capsule layer. Convolutional layer has 256, 9 × 9 kernels with a stride of 1 and ReLU activation. The layer scans the image to capture features of the board and passes them to the capsule layer. The capsule layer scans the input with 361×*dim* kernels(dim is set to be same as the input of board states to save each states' information) and then generates 361 vectors which represent the information of every position on the board. The length of the activity vector indicates presence of an each position.

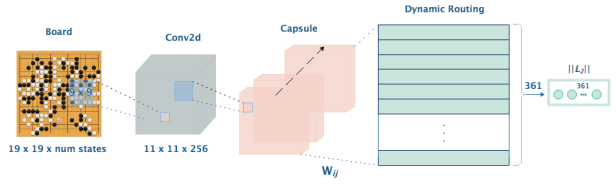

Fig. 3 Capsule Neural Network Architecture

4. Experiments and Results

Our training uses NVIDIA® GeForce GPU 1080TI with 12GB of memory. All our training codes are run on an AMD ® Ryzen 7 1800X Eight-Core Processor machine with an NVIDIA® GeForce GTX 1080 Ti GPU. The Python 3 interpreter version is 3.7.3, the Keras version is 2.2.4, and the Tensorflow backend version is 1.14.0.

4.1 Dataset

Ladder move is an inductive tactic which even beginning human player can master effortlessly however, it is difficult for computer Go program to master the skill.

As shown in Figure 4 [5], the left figure shows the beginning of a ladder scenario, and the right figure indicates the end of a ladder scenario.

We train and test our networks on ladder move dataset [15]. Ladder scenario consists of 116 go games which include ladder moves, and the dataset yields about 11000 states-move pairs for training and testing.

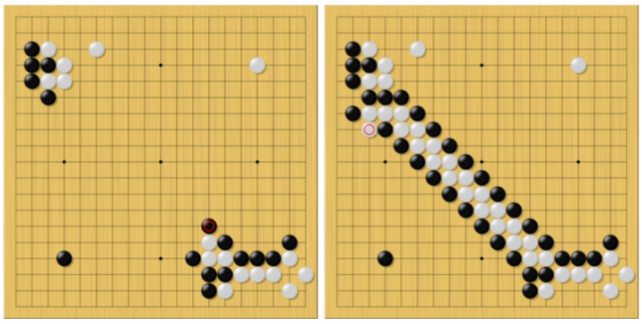

Fig. 4 A sample of ladder moves

4.2 Evaluation

We will curate ladder dataset to evaluate the model's ability of handling ladder move, and observe the accuracy and validation accuracy to judge whether the model is able to learn the ability of inductive reasoning.

4.3 Preconditions

We parse the ladder moves dataset and train the network modified from the betago program [16], and we set convolutional neural network parameters the same as the betago program. Additionally, we also adjust the capsule program [17] to fit for our model.

4.4 Details of Learning

We partition the datasets into two parts, training set and validation set, which is compose of states-move pairs from different games of Go. We use 85% for training and 15% for validation. Dropout and pooling is not adopted in our program, partly because doing so would lose some information. We trained our model using adam with a batch size of 128 examples, and a learning rate of 0.001, and the loss function is set to be categorical crossentropy.

We make eight experiments which use 1-8 continuous states to predict previous move, and we find that using 5 states is more robust and effective so that the following experiment is supposed to take the experiment of predicting previous move with 5 continuous board states as an example to illustrate. Other results of ConvNet with one-plane representation, ConvNet with multi-plane representation and CapsNet are shown in Appendix A, B and C separately.

4.5 ConvNet Model with One-plane Representation

We have trained the model with one-plane input with 4-layer convolutional neural network. As shown in Figure 5, we can observe that the model can achieve high accuracy, and the accuracy on the validation set converges

to over 50%. The loss continuously decreases, however, after nearly 30 epochs, the validation loss begins to increase. It shows that we can get a network of high quality rapidly but the network starts to overfit the training set when validation loss sets about increasing.

A 4-layer convolution neural network with one-plane representation can reach good performance in the early training period. We can extract the network with robustness and effectiveness by the technique of early stop.

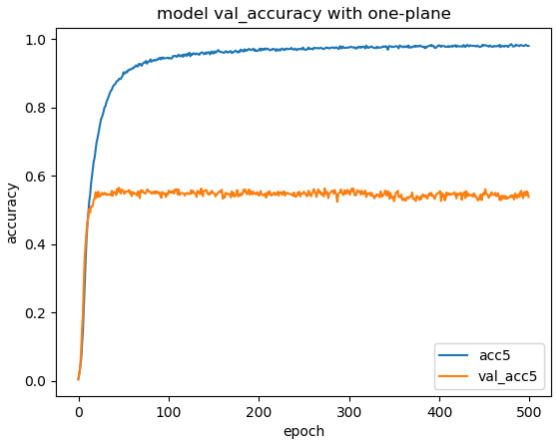

Fig. 5 Convolutional Networks with One-plane Accuracy. acc5 indicates the model accuracy of predicting previous move with 5 continuous board states, and val_acc shows the level of validation.

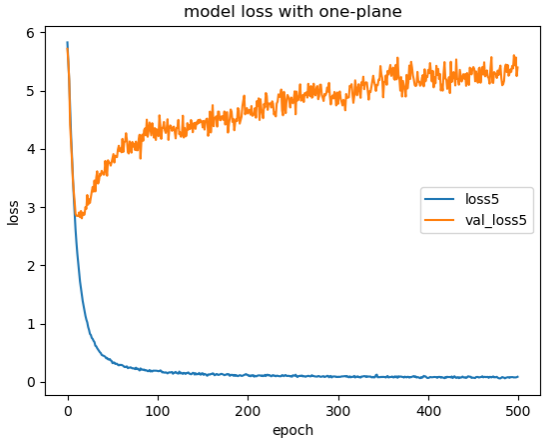

Fig. 6 Convolutional Networks with One-plane Loss. loss5 indicates the model accuracy of predicting previous move with 5 continuous board states, and val_loss shows the loss of validation.

4.6 ConvNet Model with Multi-plane Representation

Considering that the game of Go is not fully observable from the current board state, we divide the input image into 3 binary channels, the board of black stones, the board of white stones and a board which indicates the color feature. As shown in Figure 7, the accuracy is approaching close to 100% and the validation accuracy converges to about 40%. The Figure 8 illustrates that the training loss is almost reaching 0 but validation loss is increasing rapidly from the beginning. Compared to the model with one-plane representation, we utilize more board features and network parameters, but the problem of overfitting seems like more serious.

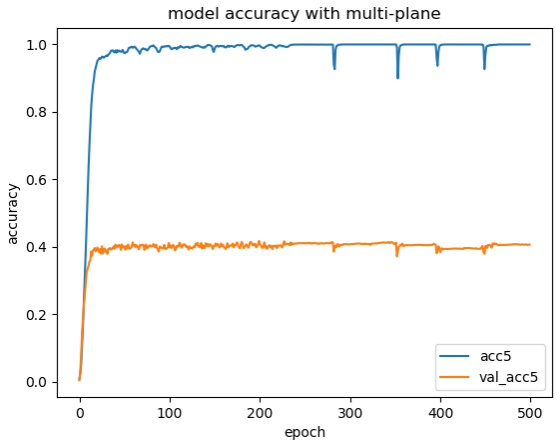

Fig. 7 Convolutional Networks with Multi-plane Accuracy.

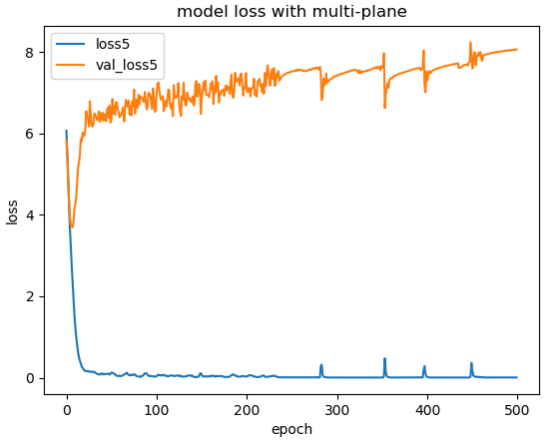

Fig. 8 Convolutional Networks with Multi-plane Loss.

4.7 Capsule Network

Capsule network has many approaches to construct, and we purely give a simple implementation and show that capsule network makes sense and dynamic routing helps. We train our model with one-plane representation, from the Figure 9 and 10, we found that the accuracy fluctuates in the training process but the validation loss converges to a certain value about 4. The result indicates that the network has some effect on reducing overfitting.

5. Discussion

Through supervised learning, the network is supposed to learn the ability of predicting ladder moves, and three kinds of networks achieve high performance. Compared to ConvNet with multi-plane representation, ConvNet with one-plane representation achieves better performance and

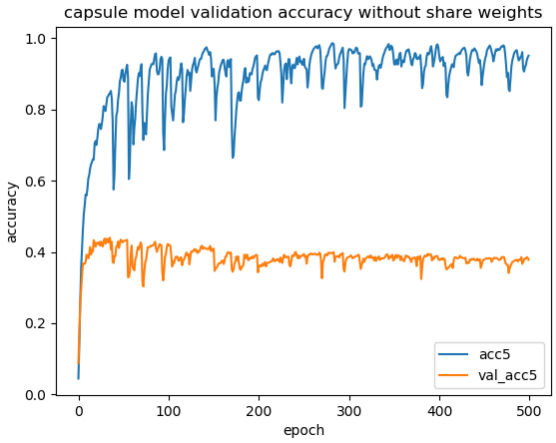

Fig. 9 Capsule Networks Accuracy without Share Weights

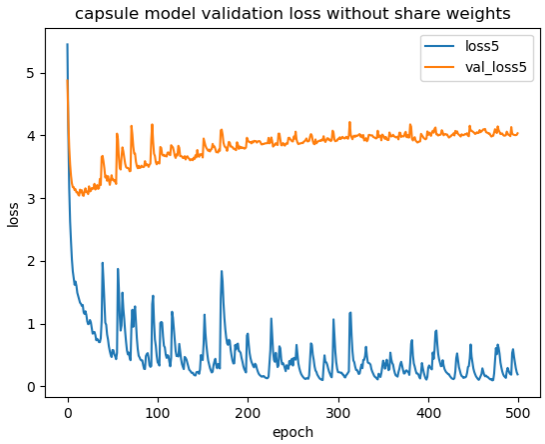

Fig. 10 Capsule Networks Loss without Share Weights

saves training time, partly because there is no need to consider about the issue of illegal moves and ko states for prediction of previous move, therefore, using a simple data representation is more fit for this experiment. However, there exists a problem of overfitting on the 2 convolutional networks. Indicated by the result, capsule network which has a shallow structure can reduce overfitting and learn more robust representations, additionally, the structure is supposed to save more important details. We also found that using more than 2 board states can make the training process quicker and robust by conducting the experiments of predicting the previous move with 1-8 continuous board states.

6. Conclusion and Future Work

In this work, we have introduced a method called back prediction. We found that a shallow neural network can acquire inductive ability from back prediction effectively. Back prediction is a simple but efficient method to simplify the problem of increasing the model's robustness, and back prediction network works well on ladder scenario without significant computation resource, and the most obvious next step is to test our method on other datasets. We hope that our method is helpful for combining with value network and policy network to make a more robust Go AI.

The easiest and most common method to reduce overfitting is to artificially enlarge the dataset [18], transforming the input board image into 8 symmetrical transformed image is a promising method to enlarge the ladder move dataset, and we would like to reduce overfitting by data augmentation.

References

[1] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al.: Mastering the game of Go with deep neural networks and tree search, nature, Vol. 529, No. 7587, p. 484 (2016).

[2] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. 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).

[3] Kocsis, L. and Szepesvári, C.: Bandit based monte-carlo planning, European conference on machine learning, Springer, pp. 282–293 (2006).

[4] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al.: Mastering the game of go without human knowledge, Nature, Vol. 550, No. 7676, p. 354 (2017).

[5] Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J. and Zitnick, L.: ELF OpenGo: an analysis and open reimplementation of AlphaZero, International Conference on Machine Learning, pp. 6244–6253 (2019).

[6] Sabour, S., Frosst, N. and Hinton, G. E.: Dynamic routing between capsules, Advances in neural information processing systems, pp. 3856–3866 (2017).

[7] Wikipedia contributors: Rules of Go — Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Rules_of_Go&oldid=905795805 (2019). [Online; accessed 22-July-2019].

[8] Knuth, D. E. and Moore, R. W.: An analysis of alpha-beta pruning, Artificial intelligence, Vol. 6, No. 4, pp. 293–326 (1975).

[9] Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search, International conference on computers and games, Springer, pp. 72–83 (2006).

[10] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S.: A survey of monte carlo tree search methods, IEEE Transactions on Computational Intelligence and AI in games, Vol. 4, No. 1, pp. 1–43 (2012).

[11] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm, arXiv preprint arXiv:1712.01815 (2017).

[12] Pascutto, G.-C.: Leela Zero, https://github.com/gcp/leela-zero (2017).

[13] Zeng, Q., Z. J. Z. Z. L. Y. C. M. and Liu, S.: Phoenixgo, https://github.com/Tencent/PhoenixGo (2018).

[14] Shen, Y.: SabakiHQ, https://sabaki.yichuanshen.de/ (2017).

[15] Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J. and Zitnick, L.: ELF OpenGo: an analysis and open reimplementation of AlphaZero, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, pp. 6244–6253 (online), available from ⟨http://proceedings.mlr.press/v97/tian19a.html⟩ (2019).

[16] maxpumperla: betago, https://github.com/maxpumperla/betago (2015).

[17] Su, J.: Capsule, https://github.com/bojone/Capsule (2017).

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

A. Results of ConvNet with One-plane Representation

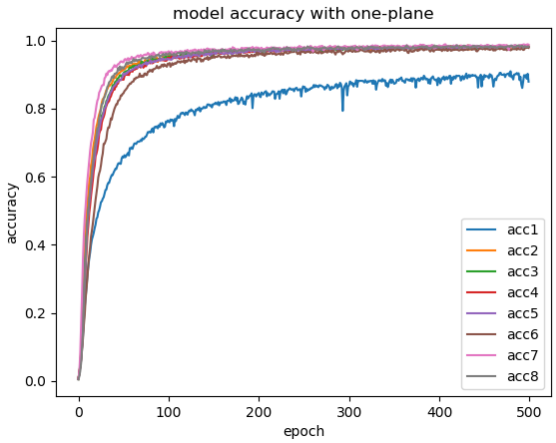

Fig. 11 CNN Model Accuracy with One-plane

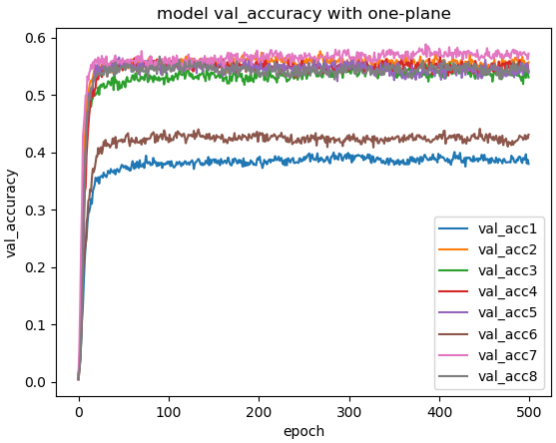

Fig. 12 CNN Model Validation Accuracy with One-plane

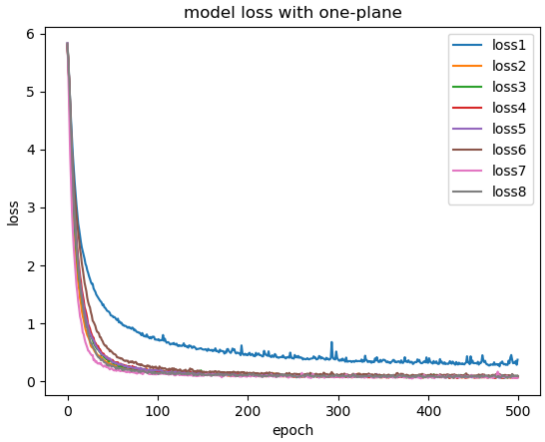

Fig. 13 CNN Model Loss with One-plane

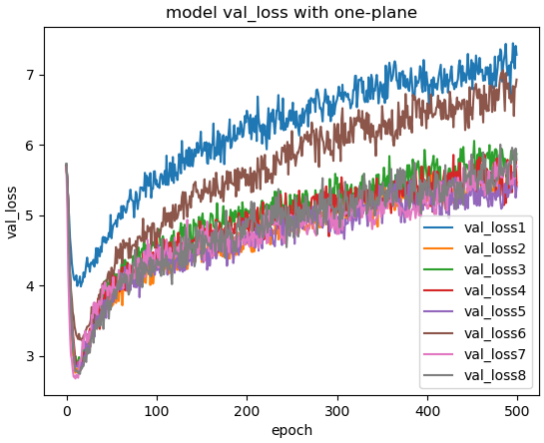

Fig. 14 CNN Model Validation Loss with One-plane

B. Results of ConvNet with Multi-plane Representation

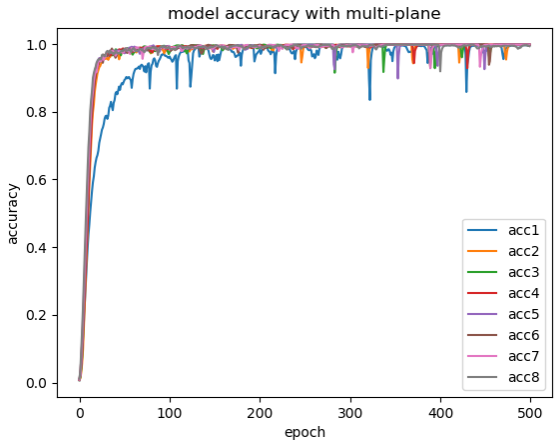

Fig. 15 CNN Model Accuracy with Multi-plane

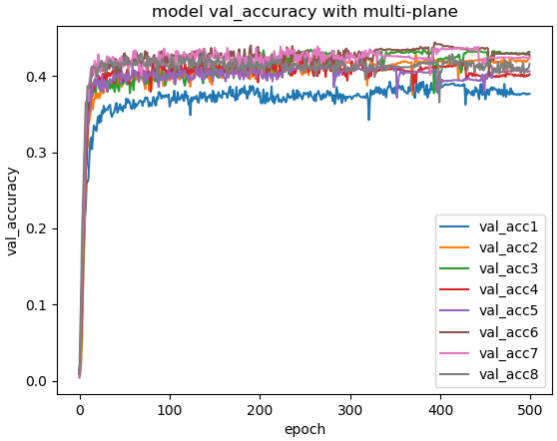

Fig. 16 CNN Model Validation Accuracy with Multi-plane

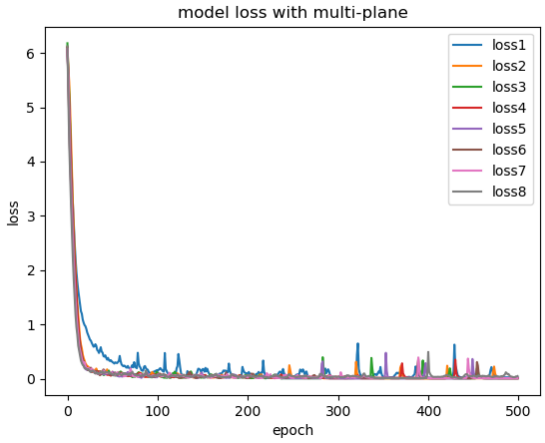

Fig. 17 CNN Model Loss with Multi-plane

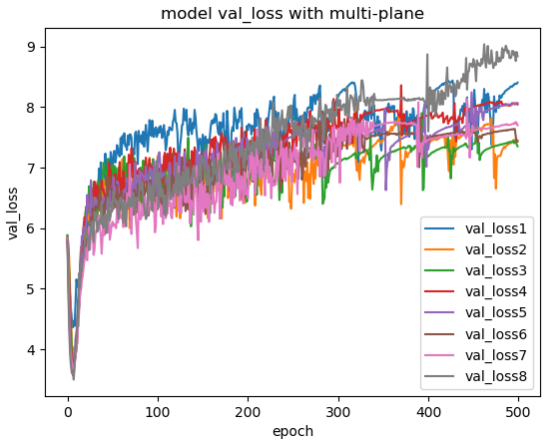

Fig. 18 CNN Model Validation Loss with Multi-plane

C. Results of CapsNet

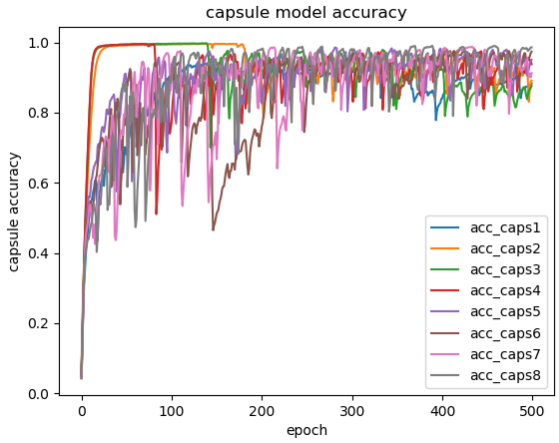

Fig. 19 Capsule Network Accuracy

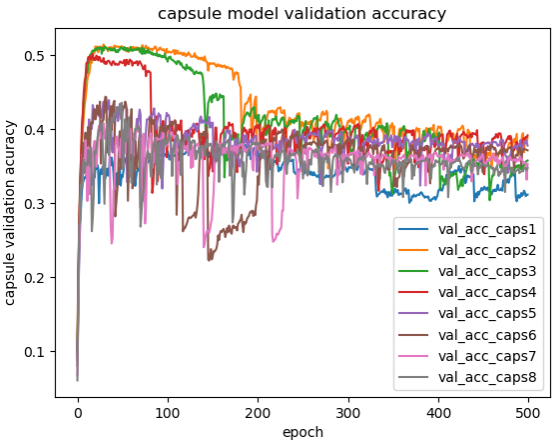

Fig. 20 Capsule Network Validation Accuracy

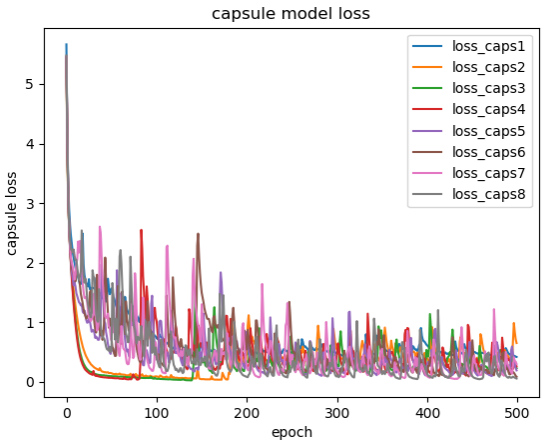

Fig. 21 Capsule Network Loss

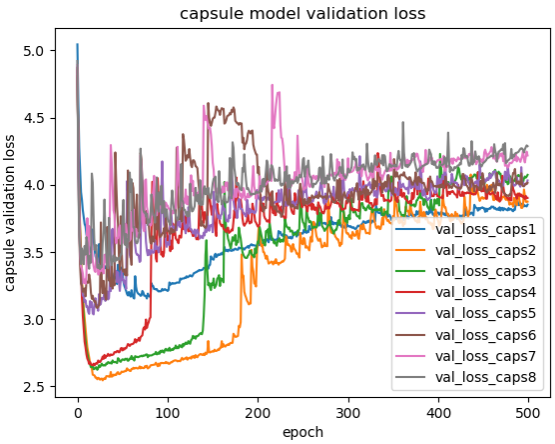

Fig. 22 Capsule Network Validation Loss