GoGoGo: Improving Deep Neural Network Based Go Playing AI with Residual Networks

Xingyu Liu

xyl@stanford.edu

1. Introduction

The game of Go has a long history and has been viewed as the most challanging classical game due to the enormous amount of possible moves and the lack of precise evaulation tools. AlphaGo [13], a Go-playing AI built by Google DeepMind, used a new approach of combining deep neural networks with tree search to solve the Go playing problem. It narrows down the search space by introducing "policy network" that gives reasonable moves instead of all possible moves. It then combine the Monte Carlo tree search (MCTS) with a "value network" that evaluates the winning chance of current state to search deeply for the future moves. In a game with the 18-time Go world champion winner Lee Sedol by 4 games to 1. This is the first time that an artificial intelligence program defeated the Go world champion.

However, the "policy network" and "value network" used by AlphaGo are vanilla convolutional neural networks consisting of stacks of convoluation layers. The winner of 2015 ImageNet competition has shown that deep residual networks might be a better choice of image classification problem [12]. We assume that in the task of prediction and regression of "policy network" and "value network", deep residual network might also win. To train the skills learned in this course, including deep neural networks, search algorithms and reinforcement learning, we decided to reimplement a similar Go playing AI, GoGoGo, that uses deep residual networks instead of vanilla convoluational networks, as the course project. We used Ing Changki rule [3] for the game.

2. Related Work

Zen [8] is a Go playing engine developed by Yoji Ojima. It won a gold medal in 14th Computer Olympiad in May 2009 and the Computer Go UEC Cup in 2011, 2014, and 2016. Pachi [6] by Petr Baudis and Fuego [1] by University of Alberta is a Go program using Monte Carlo.

With the success of AlphaGo, Go AI was recently powered by using deep learning algorithms and was improved significantly. Zen was improved to be "DeepZenGo" by using deep learning algorithms. It had three games against against Cho Chikun 9d and won one of them. Darkforest [15], the Go AI developed by FAIR, also used deep learning algorithms.

3. Methodology

3.1. Training data Collection and Processing

We used similar approach of training data collection and processing as in AlphaGo. We used existing Kifu [4] by professional human Go player. We have collected over 65000 Kifu from StoneBase software dataset [7]. Each Kifu consisting more 150 moves on average, which will provide a massive training set. However, the Kifus only records the position of each move. We need to explicitly expand each Kifu we collected into a series of four-channel feature map, whose total number equals to the number of moves in the Kifu.

We wrote our own program based on Bensen's Algorithm [10] to eliminate dead stones at each step. The Bensen's Algorithm also generates the positions that is not allowed to be placed on (due to Ko fight [5] for example). This is important in the subsequent playing stage where forbidden positions are eliminated from the board. Given the

moves, we generated state-action pairs and stored them.

The Kifus also recorded the result of the game. Like AlphaGo, we didn't pay attention to the margin of winning, since the games are held under different rules. For each of the over 65000 Kifus, we generate more than 150 four-channel feature maps on average together with the boolean variable indicating the result of the game and stored them as hdf5 files so that it can be accessed efficiently from disk. We generated 13915316 images in total. We split the training data into two parts: training and validation dataset. The validation data set include randomly selected 700000 images.

3.2. Convolutional Neural Network Architecture

Inspired by the implementation of AlphaGo [13], We decided that the policy network and value network should be the same architecture except for the very last layer where the policy network output a 19×19 score map and value network output a single value. The reason we use the same architecture is that if one of the network architecture is sufficient to extract useful information from the existing game state, changing only the last layer would still be able to extract sufficient information.

The network architecture of the policy network and value network is illustrated in Figure. 1. Since we used Ing-Cup rule, the game state is only related to the position of stones but not related to the number of stones captured. The input now only consists of 4 channels of feature maps instead of 48: 3 channels represent the position of black and white stones and 1 channel indicate whether white or black should do the next move. The specification of input data is listed in Table. 1.

Table 1. Input description

	# of Planes	Description
Stone color	3	Black / White / Empty
Player Playing	1	Black player is playing
Illegal Move	1	The move is illegal

The networks have totally 17 layers. Each of the layers in the network has 64 output channels. AlphaGo used fewer numbers of layers (13) and thicker layers in their networks (128-256). The reason we chose more and thinner layers is that for both the efficiency consideration as well as exploring new network architecture reason.

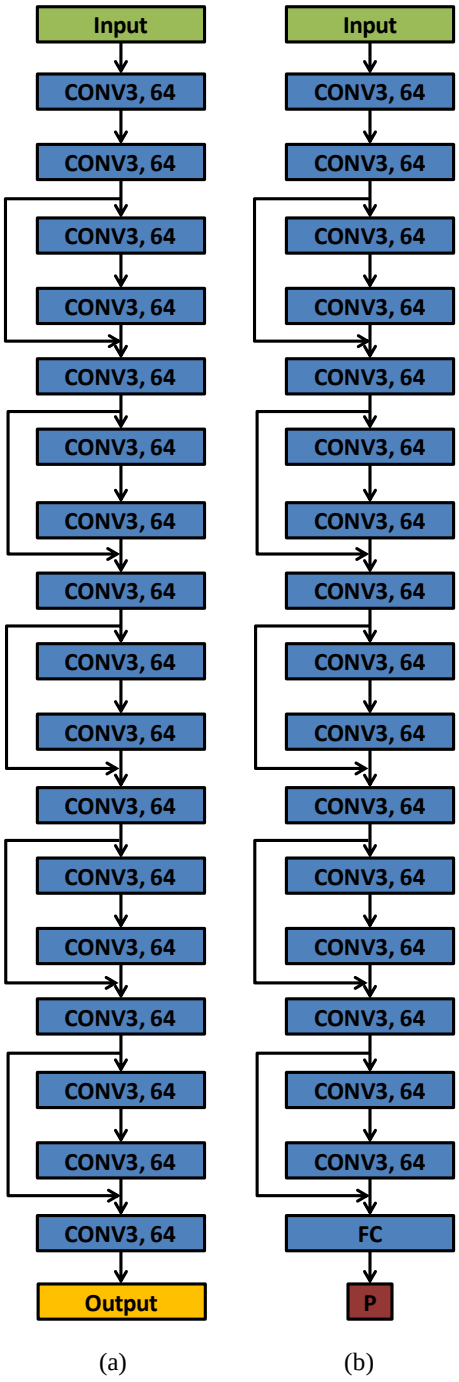


Figure 1. The architecture of the convolutional neural network used in this project: (a) policy network, (b) value network

The last layer of value network is an FC layer with a single output, which is different from the policy network. Each layer in both network is followed by a ReLU layer that is not

shown in the figure. A basic problem with residual network is whether the element-wise summation with bypassed layer occur before ReLU layer or after ReLU layer. As pointed out by Szegedy et. al [14], putting ReLU layer after the element-wise summation is a better choice. So we follow this approach.

3.3. Policy Network Supervised Training

In the first stage of the training pipeline, the data described in Section 3.1 are used to train the policy network described in Section 3.2. During supervised training, when given the parameters σ, current state of the game s and human player action from the training data a, we would like to maximize $p_\sigma(a \mid s)$. We used the stochastic gradient ascent to maximize the above possibility. That is

$$\sigma_{t+1} = \sigma_t + \alpha \cdot \nabla_\sigma(p_\sigma(a \mid s))$$

We implemented the supervised training using TensorFlow [9]. The reason we chose this framework is that TensorFlow has good python interface and it can be integrated with other program logics more easily.

We trained tried several parameters during the training. The current settings we used are with constant step size of $\alpha = 0.003$ and softmax temperature of $\beta = 0.67$, both following the parameter suggested in AlphaGo nature paper [13]. We used two NVIDIA TITAN X GPU and trained for two days. The training accuracy can reach 36%, however, the testing accuracy is less than 26%. This may be due to not enough training or due to the ordering of training data not being shuffled. One of the reasons that the training is slow is that Tensorflow is slow on GPUs. However, there's nothing we could do about it.

3.4. Policy Network Reinforcement Learning

The second stage of the training pipeline is the Reinforcement Learning (RL) of the policy network from the supervised learning model. Denote the terminal time step to be T, the reward function of a game state at time t is $r(s_t) = +1$ for winning and $r(s_t) = -1$ for losing. The training set of this stage is the game records we generated by ourselves through letting the current policy network play against other randomly selected previous iteration of the policy network. The weights are updated at each time step by stochastic gradient ascent:

$$\rho_{t+1} = \rho_t + \alpha \cdot r(s_T) \cdot \nabla_\rho(p_\rho(a \mid s))$$

3.5. Value Network Reinforcement Learning

The final stage of the training pipeline is the reinforcement training of value network. Value network has a similar architecture as the policy network, but the last layer is an FC layer and produces a single number indicating the winning probability given the input of current game status. The value network with weight θ is trained using the formula below:

$$\theta_{t+1} = \theta_t + \alpha \cdot (z - v_\theta(s_t)) \cdot \nabla_\theta(v_\theta(s_t))$$

3.6. Monte Carlo Tree Search

We also implemented Monte Carlo Tree Search (MCTS) framework in Python based on the literature [11] and the course lecture.

Given a state s_t, an action a_t is selected from s_t in the following way:

$$a_t = \arg\max_a(Q(s_t, a) + u(s_t, a))$$

where the utility $u(s_t, a)$ is proportional to $\dfrac{P(s,a)}{1+N(s,a)}$. The $P(s, a)$ and $N(s, a)$ are obtained from MCTS simulation. In ith simulation, we denote $1(s, a, i)$ as the sign function indicating whether the edge (s, a) was traversed. Then we have:

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

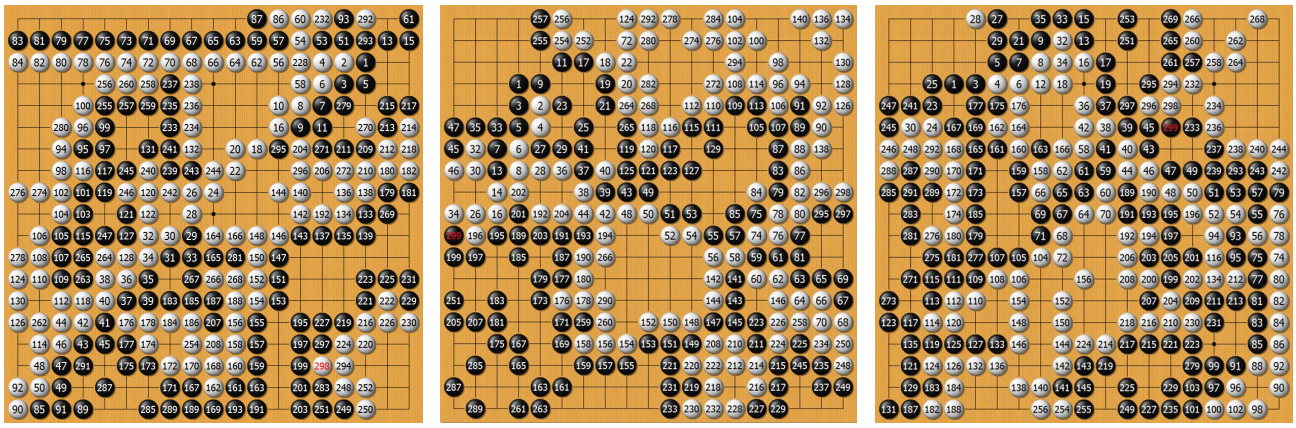
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) v(s_L^i)$$

3.7. Graphics User Interface

Since the GUI is only used to facilitate our debugging and improve the final user experience, we don't want to spend much time on it. We took used the existing code using Qt in python on github by nhnifong [2] and seamlessly integrated it into our own program.

3.8. System Optimization

Storing all kifus by human players on disk takes huge space. We proposed and used a dynamic kifu expansion



(a) (b) (c)

Figure 2. Three kifus of supervised-trained policy network play against itself

mechanism. It takes two numbers representing the position of the next played stone and dynamically generate the input which contains the board state and other information. When dynamically expanding the kifu, the forbidden position can also be obtained.

We used a two-level batch training method. First we select a batch (1,000) of kifus, dynamically expanding each of them into board states associated with moves. Second, we randomly shuffle the board-move pairs (200,000) so that training data from different kifus can overlap to prevent overfitting.

4. Experiment Result

Due to the limit of computation resource and time, we only completed the supervised training of policy network. The reinforcement learning of policy network and value network is still ongoing. We let the existing pre-RL policy network play against Pachi [6], it didn't win a single game. It shows that both the RL of policy network and value network is necessary to ensure stronger AI.

In supervised training of the policy network, the training accuracy can reach 31%, the validation accuracy can reach 36%. The training loss of supervised learning is illustrated in Fig. 3.

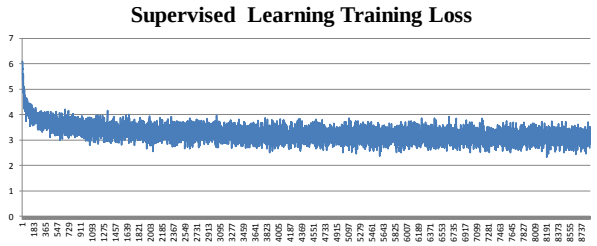


Figure 3. The training loss of supervised learning of policy network

5. Future Work

Then we will generate massive amount of Kifus by using GoGoGo to play against itself. The generated Kifu will be used in the reinforcement learning of the policy network and the value network. The training framework will be implemented so that the generating of Kifus and the learning from Kifus can be done automatically by the learning framework.

We will invite some non-professional but strong Go player at Stanford to test its strength. The challenge are: 1) the time spent on the training of two networks is huge; 2) the number of self-generated Kifus is huge and needs a lot of space to store. Topics in the course including reinforcement learning, neural networks, Monte Carlo will be used.

If time allowed, we will upload our AI onto online Go playing platforms to test its strength. We expect it to be at least as strong as past hand-crafted Go AI.

References

[1] Fuego. https://fuego.sourceforge.net/.

[2] GoGoGo. "https://github.com/nhnifong/GoGoGo".

[3] Ing Cup. "https://en.wikipedia.org/wiki/Ing_Cup".

[4] Kifu Wikipedia. https://en.wikipedia.org/wiki/Kifu.

[5] Ko fight. "https://en.wikipedia.org/wiki/Ko_fight".

[6] Pachi - Board Game of Go / Weiqi / Baduk. https://pachi.or.cz/.

[7] StoneBase software. http://senseis.xmp.net/?StoneBase.

[8] Zen (software). https://en.wikipedia.org/wiki/Zen_(software).

[9] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015.

[10] David B Benson. Life in the game of go. *Information Sciences*, 10(1):17–29, 1976.

[11] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 72–83, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[13] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[14] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[15] Yuandong Tian and Yan Zhu. Better computer go player with neural network and long-term prediction. *arXiv preprint arXiv:1511.06410*, 2015.