AlphaGomoku: An Ablation Study of AlphaZero on the Game Gomoku

Ace Hu (acehu) **Siyu Liu** (siyuliu3) **Yanpei Tian** (yanpeit)

Abstract

First introduced by DeepMind in 2018, AlphaZero has been attracting much attention due to its superior ability to achieve super-human level performance on different board games. In this project, we plan to perform an ablation study on different building blocks of AlphaZero: State Representation, Neural Network Architecture, Monte Carlo Tree Search (MCTS). We plan to study the different contributions made by various parts of the model to its ability of achieving super-human level of playing on the game of Gomoku. We identify the important role played by state representation in the performance of the model, the relationship between model size and training, and the effect of using TD style bootstrapping during data collection.

1. Introduction

In October 2017, AlphaGo Zero was introduced by DeepMind (Silver & Hassabis, 2017), becoming the best Go player in the world. Later on, the algorithm in the original program was generalized to play the game Chess and Shogi (Silver et al., 2018), indicating the generalizability of the reinforcement learning algorithm used in the original version of AlphaGo Zero. In this project, we want to a explore specific setting of Board Game, namely Gomoku by using a reinforcement learning algorithm based on self-play.

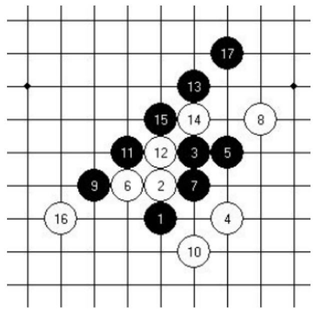


Figure 1. An example of Gomoku game board, with the black winning by forming 5 pieces in a row diagonally.

Gomoku, also called Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board with the aim of forming an unbroken chain of five stones horizontally, vertically, or diagonally.

At a very high level, the AlphaZero uses a modified version of Monte Carlo Tree Search (MCTS) assisted by a neural network to make decisions. It is no surprise that the model's performance will change significantly if we make changes to its building blocks: State representation, Neural Nertwork architecture, and MCTS implementation.

- **State representation:** The model relies on the state representation as input to give an estimation of next move distribution and current evaluation of the board. By changing the amount of information included in the state representation, we can analysis the effects of different state representations have on the performance of the model.
- **NN architecture:** Residual network (ResNet)(He et al., 2015) has been known as an effective way to build deeper neural networks. The skip connections presented in the network help with information flow and avoid the vanishing gradient problem. By building stacks of ResNets with different depth, we can gauge the relationship between model size and its performance.
- **MCTS:** In the original model, data collection was performed by self-play. Each round of self-play was executed to the end of the game. We propose a Temporal Difference (TD) style data collection to speed up the data collection process.

We perform an ablation study on these three parts of the algorithm and try to attribute contributions made by various parts in the overall model to its super-human level performance on board games.

2. Related Work

AlphaGo Zero is an improved version of AlphaGo(Silver et al., 2016) that is able to do self-training and learning without human knowledge. It is able to beat previous versions of AlphaGo, which in term defeats world champion Lee

Sedol and Ke Jie, being arguably the best Go player in the world. There are several new components that lead to its performance improvements.

1. AlphaGo Zero no longer needs human knowledge guidance and does its data collection and training all on its own. It only takes the current pieces on the board and recent history as input, which makes it no longer constrained by human knowledge. This feature also enables it to find novel ways of playing and discover new knowledge about a game.

2. Previous versions of AlphaGo have two neural networks in the model: the policy network is responsible for generating the next action distribution and the value network is used to give estimation of the current board. AlphaGo Zero combine these two networks, allowing for more efficient and holistic training and evaluation.

3. AlphaGo Zero uses value network to give current estimations for winning based on the current set up of the board, instead of using random rollouts.

AlphaZero generalizes AlphaGo Zero to play the game of chess and shogi. Its ability to learning these games all from scratch demonstrates the algorithm's generalizability.

Residual Network was first introduced in computer vision for image recognition. The skip connections and highway connections(Srivastava et al., 2015) presented in ResNet allow us to build much deeper representations and still train the network effectively. It is later on discovered to also be effective in solving reinforcement learning problems. ResNet will serve as a building block of the neural network in our model.

Monte Carlo Tree Search is a heuristic guided search algorithm used to make decisions in a game play.

Algorithm 1 Monte Carlo Tree Search

```
  **Input:** Game model $M_v$, Current state $s_t$
  **Output:** Next action $a_t$
  1. Build a search tree rooted at the current state $s_t$ according to the game model $M_v$
  2. Sample actions and next states
  3. Iteratively construct and update search tree by performing K simulation episodes starting from the root state
  4. After simulation, select the next action with maximum value in search tree.
```

$$a_t = argmax_{a \in A} Q(s_t, a)$$

3. Approach

3.1. State representation

Following Silver et al., 2018, we propose the following state representation. Although for the game of Gomoku, the current board setup contains all the information for one to come up with the best next move, empirical results show that adding recent history to the state representation helps with the model's performance. The argument behind this is that adding recent history provide an attention-like mechanism to let the model concentrate more on a specific area of the board. This is also the case for human players where we tend to pay more attention to a hot-spot on the board instead of giving equal amounts of consideration to every possible position. By varying the value of k, we effectively have control on how much recent history information we provide to the neural network and study the improvements brought by extra information in the state representation.

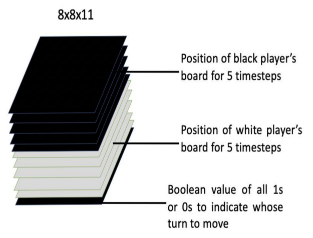

8x8x11
Position of black player's board for 5 timesteps
Position of white player's board for 5 timesteps
Boolean value of all 1s or 0s to indicate whose turn to move

Figure 2. State representation, serves as the input to the neural network. The first k (a tunable hyper-parameter) boards is the k-most recent history of black player, followed by the k-most recent history of white player. The last board is a binary-valued board, indicating whose turn it is currently.

3.2. Neural network architecture

In this section, we want to study the relationship between the complexity of the neural network and the performance of the model.

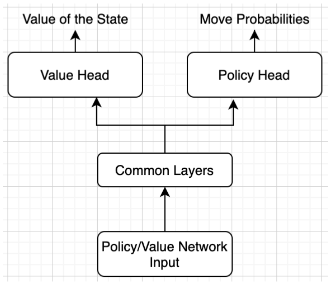


Figure 3. Network Structure

The three parts of the network are described below:

- **Value head:** Branch of the network used to give an estimation of reward based on the current board setup. It contains one convolution layer and two fully connected layer to output a value ranging in [-1,1].

- **Policy head:** Branch of the network used to give an probability distribution based on the current state representation. It contains one convolution layer and one fully connected layer to output a 8×8 action logits probabilities (before softmax).
- **Common layers:** This is the main body of the neural network, containig a stack of ResNets. By varying the number of ResNets in the stack, we can change the size of our model.

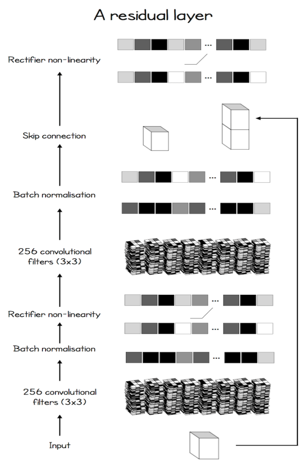


Figure 4. Residual layer used in this project. Each residual layer is composed of two convolution layers, each with batch normalization and ReLu non-linearity, and a skip connection. The skip connection helps with information flow in a deep neural network.

3.3. Temporal difference style data collection

In the original implementation of AlphaGo Zero, MCTS is used to select each move and each game is played until the end. We propose a TD style estimation (**Algorithm 2**) for the current reward estimation to speed up the data collection process. This feature should be more helpful during the beginning of the training as the network hasn't known much about game endings. In that case, the model will start to make randoms guesses as a game progress, making data collected under such condition less useful.

4. Experiments

4.1. Data

In Alphazero(Silver et al., 2018), data is collected through self-play with two MCTS players. The data collected includes states(S), MCTS probability(π) at each state action, and winner(z) after each epoch. For the current player specifically, z=1 if it is winner, z=-1 if it loses, and z=0 if there is a tie.

Algorithm 2 TD estimate of MCTS rollout

```
Result: Estimated reward value
repeat
    Perform rollout as usual
    if The board is more than half occupied and ($V(s) > 0.99$ or $V(s) < -0.99$) then
        Return $V(S)$ as the reward estimation
    end if
until We've reached the end of the game
Return +1 or -1 based on result of rollout
```

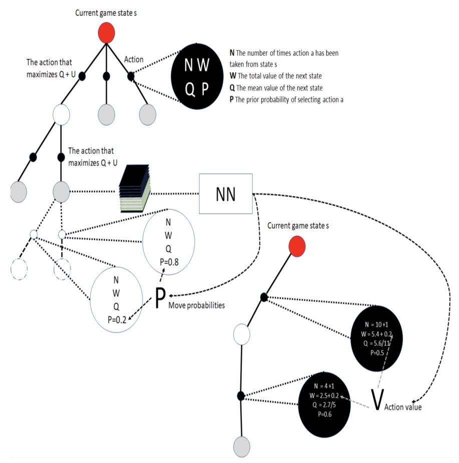

Current game state s
The action that maximizes Q + U
Action
N W Q P
N The number of times action a has been taken from state s
W The total value of the next state
Q The mean value of the next state
P The prior probability of selecting action a
The action that maximizes Q + U
NN
N W Q P=0.8
N W Q P=0.2
P Move probabilities
Current game state s
N = 10+1 W = 5.4+0.2 Q = 5.6/11 P=0.5
N = 4+1 W = 2.5+0.2 Q = 2.7/5 P=0.6
V Action value

Figure 5. TD style data collection during self-play. Calling the current neural network to get an reward estimate base on the current state when certain condition is satisfied.

Data is further augmented by rotating and flipping both states S, and MCTS probability π in total 8 ways. This could be implemented because symmetric property under rotation and transpose of Gomoku game.

4.2. Evaluation method

When enough data is collected, loss and entropy are updated after training in neural network and a new model is saved every 100 epochs. Evaluation is carried out by self-play again with MCTS player, win ratio in 10 games is then recorded and used to compare among different models.

Another way for evaluation is to look at loss over time, especially check for convergence and whether the loss is decreasing monotonically. Loss function in our algorithm is shown below:

$$Loss = (z - v)^2 - \pi^T log(p) \tag{1}$$

The first part represents the value loss, z is the estimated winner and v is the real winner. The second term represents policy loss, π is MCTS probability and p is probability of each action according to MCTS during training.

We can also track the entropy during training. Roughly speaking, entropy is the number of different actions that the agent is indifferent to choose from at a specific state. Lower entropy means the model is more certain to choose an action, which is a desired feature under effective training.

4.3. Experimental details

To conduct ablation study, we are experiementing with four different training models shown in the table below, and each varies in whether TD estimate of MCTS rollout, number of statespace used, and number of ResNet used.

No.	TD estimate	Num of Resnet	Num of statespace
1	**YES**	**1**	**11**
2	YES	5	11
3	YES	1	3
4	NO	1	11

Table 1. Different model configurations in experiment

The first model is used as reference to compare results in win ratio against pure MCTS player and loss in general. For other three models, only one of three variables is changed in each model. Thus, comparisons between experiment results can be established to interpret how each variable contributes to the final results.

We trained each model for 1500 epochs while saving a model every 100 epochs. 400 MCTS simulations are performed during training and evaluation.

4.4. Results

Shown below is the training results at the end of 1500 epochs.

No.	Loss	Entropy	Win ratio[a]
1	3.062	2.762	0.57
2	**2.336**	**2.169**	0.58
3	3.172	2.916	0.47
4	2.549	2.306	**0.68**

Table 2. Training results for different models

[a]Win ratio is smoothed

According to Table 2, the model that achieves the lowest training loss and entropy is model 2 with 11 states 5 ResNets and TD estimate. This is the model with the largest neural network with the biggest capacity in terms of model expressivity. The increased complexity in this model makes it easier to fit to training data thus to achieve lower training loss. The lower entropy achieved by this model can also be explained by the training loss:

$$Loss = (z - v)^2 - \pi^T log(p) \tag{2}$$

The second term in the training loss is the inner product between actual action distribution and predicted distribution according to the model, which serves to minimize the uncertainty of action prediction during training.

4.4.1. Training loss

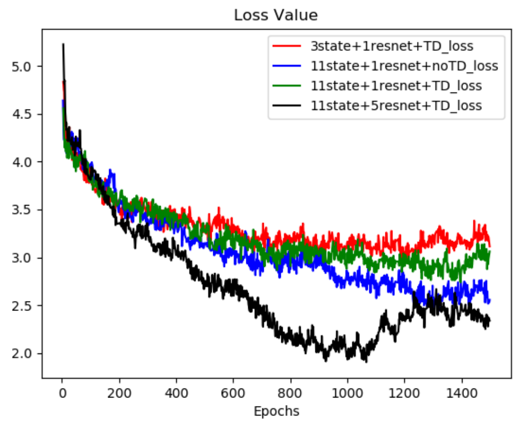


Figure 6. Training loss over 1500 epochs for 4 different model configurations

Training loss for the four model goes down during training, demonstrating effective learning. As expected, by increasing statespaces, number of ResNets and excluding TD estimate in MCTS rollout, we are able to observe further decrement in loss function and converge to lower loss. However, we also notice that the fluctuation in the curve for model 2 with 11state, 5 ResNet and TD shows the need for additional training. As model goes larger, the need for extra data and computation to train the model well also increases.

4.4.2. Entropy

Although we didn't explicitly include entropy in our loss function, the trend of entropy closely follows that of training loss during training. This phenomenon can be attributed to the following two factors:

- Inner product term in loss function, forcing predicted action distribution match the data collected by MCTS;
- MCTS simulation can generate accurate best next move distribution

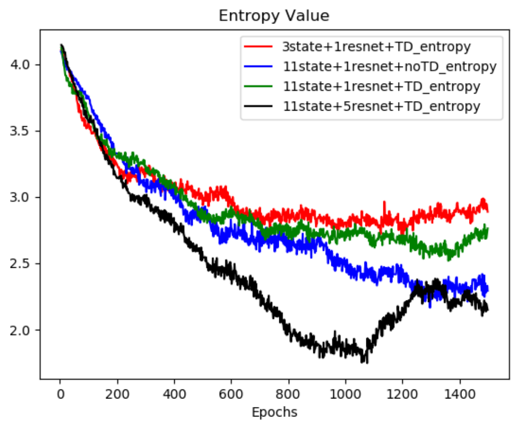


Figure 7. Entropy over 1500 epochs for 4 different model configurations

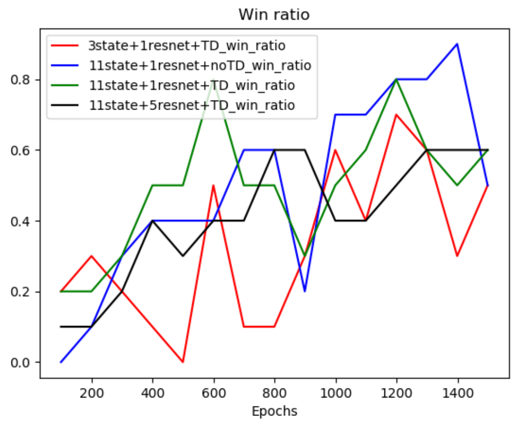


Figure 8. Win ratio over 1500 epochs for 4 different model configurations

4.4.3. WIN RATIO

Despite big variance in the plot, we can still see model 4 with 11 state, 1 ResNet and no TD achieves the highest performance among the four models. Compared to other models, model 4 don't use TD style data collection during self-play. One hypothesis for this is the data collected by playing each game to the end let the model see more "endgame" situations. And those "endgame" data points are more valuable during training to teach the model about how to choose action in the final moves in a game to win.

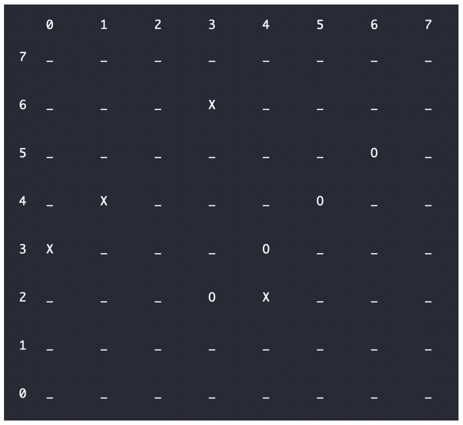
We can also see the worst performing model is model No.3 with 3 states, 1 ResNet and TD estimate. This result shows us the importance of recent history in state representation. With number of statespace equal to 3, we are trimming the number of history time-steps included in the state representation from 5 to 1.

Compared to model 4, model 1 with 11 state,1 ResNet and TD estimate performs slightly worse. As mentioned before, the hypothesis is that TD style data collection omit the part of training data that denotes key moves for the model to win a game.

The performance of model 2 with 11 state, 5 ResNet and TD is worse than expected as it is the model with the highest expressive capacity with 5 ResNet layers in its network stack. One possible explanation for this is the need for extra training of this model. The larger variance in its loss function during training also supports this reason.

5. Analysis

5.1. Pure MCTS model

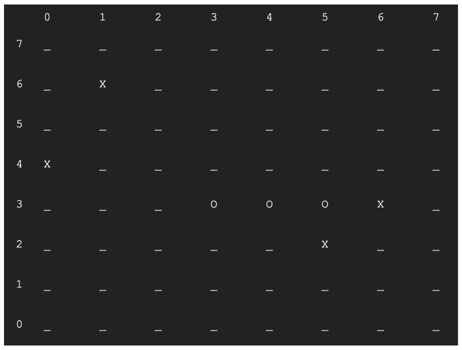


```
   0  1  2  3  4  5  6  7
7  _  _  _  _  _  _  _  _

6  _  _  _  X  _  _  _  _

5  _  _  _  _  _  _  O  _

4  _  X  _  _  _  O  _  _

3  X  _  _  _  O  _  _  _

2  _  _  _  O  X  _  _  _

1  _  _  _  _  _  _  _  _

0  _  _  _  _  _  _  _  _
```

Figure 9. Human vs pure MCTS agent with 1000 rollouts

Before any training, the MCTS performs random rollouts to make decision. Despite the increased number of rollouts (1000 for pure MCTS model vs 400 for models with a neural network) for each move, the model is not able to find a good strategy, starting a game by first place a stone (X for computer) neat the edge of the board, which is rarely seen in practice. Moreover, the model doesn't attempt to block the human player (0 for human player) after the human has achieved a "3 in a row".

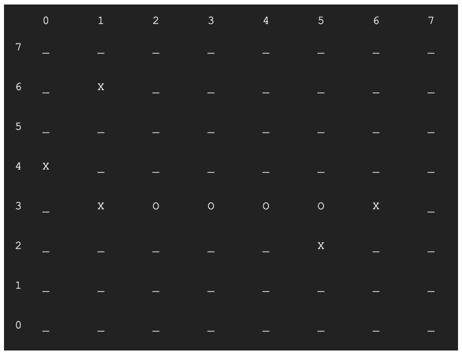
5.2. Reference model



```
   0  1  2  3  4  5  6  7
7  _  _  _  _  _  _  _  _
6  _  X  _  _  _  _  _  _
5  _  _  _  _  _  _  _  _
4  X  _  _  _  _  _  _  _
3  _  _  _  O  O  O  X  _
2  _  _  _  _  _  X  _  _
1  _  _  _  _  _  _  _  _
0  _  _  _  _  _  _  _  _
```

Figure 10. Human vs reference model with 400 rollouts

After 1500 epochs of training, the reference model (11 state, 1 ResNet and TD) (X for computer) is able to identify the threat of a ”3 in a row” by placing a stone along the line.

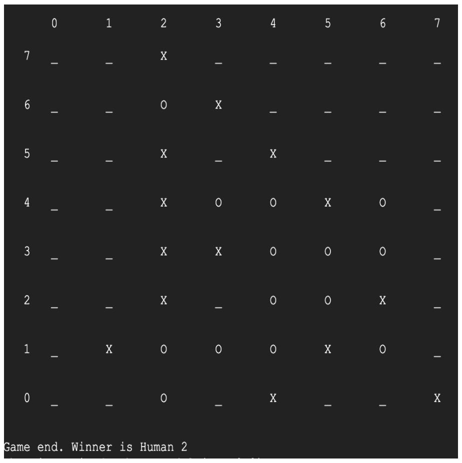


```
   0  1  2  3  4  5  6  7
7  _  _  _  _  _  _  _  _
6  _  X  _  _  _  _  _  _
5  _  _  _  _  _  _  _  _
4  X  _  _  _  _  _  _  _
3  _  X  O  O  O  O  X  _
2  _  _  _  _  _  X  _  _
1  _  _  _  _  _  _  _  _
0  _  _  _  _  _  _  _  _
```

Figure 11. Human vs reference model with 400 rollouts

Naturally, the model has also learned to deal with a common move made by beginner players to the game: extending a partially blocked ”3 in a row” by 1 toward the other direction. This time, the computer is able to identify the only action that can make it survive: blocking the row on both directions.

5.3. Best model



```
   0  1  2  3  4  5  6  7
7  _  _  X  _  _  _  _  _
6  _  _  O  X  _  _  _  _
5  _  _  X  _  X  _  _  _
4  _  _  X  O  O  X  O  _
3  _  _  X  X  O  O  O  _
2  _  _  X  _  O  O  X  _
1  _  X  O  O  O  X  O  _
0  _  _  O  _  X  _  _  X
Game end. Winner is Human 2
```

Figure 12. Human vs best model with 400 rollouts

With the best model (11 state, 1 ResNet and no TD), I, as a beginner human player to the game of Gomoku, can play a reasonable game. Our best model, after only 1500 epochs of training, no longer make random moves on the board. It also learns some basic tricks of the game, such as trying to form 2 ”3 in a row” at the same time, which will lead to a win.

6. Conclusion

In this project, we perform an ablation study of AlphaZero on the game of Gomoku. We studied the contributions made by various parts of the model to its ability of super-human level playing in board games. To be specific, we make change to the models state representation, neural network architecture, and data collection process during self-play.

For change of state representation, we conclude different representations can have very big influence on the model's performance. By including more recent history, the model can achieve better performance. Going forward, different forms of information can also be included in the state representation rather than focusing on the board's setup. One possible change is to include only the most recent piece in the board as the history. By setting only 1 position to 1 on the board, we can signify the notion of history to the model.

For change of neural network, extra training is needed to reach a definitive conclusion. By training a model with bigger neural network the same amount of time as we use to train a smaller model, the bigger model is not able to perform as well as the smaller model, although it has the lowest training loss and entropy.

We introduce the TD style data collection during self-play to speed up the training. However, the performance of the model is negatively affected. This result shows us the importance of ”endgame” data collected during last stage of a game. Training on these data enables the model to make better decisions on key moves to win the game.

References

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Silver, D. and Hassabis, D. Alphago zero: Starting from scratch. *London, United Kingdom: Deepmind*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

Silver, D., Hubert, T., Schrittwieser, J., and Hassabis, D. Alphazero: Shedding new light on the grand games of chess, shogi and go. *DeepMind blog*, 2018.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks, 2015.

A. Member Contributions

Equal contribution.

- **Ace Hu**
- **Siyu Liu**
- **Yanpei Tian**