Playing Gomoku with AlphaZero

Kaixing Wu (wuk@carleton.edu)
James Yang (yangj2@carleton.edu)

June 10, 2019

1 Introduction

In 2015, the computer program AlphaGo became the first intelligent system to beat a professional Go player. In the AlphaGo implementation, a 13-layer policy network is trained using 30 million human expert moves and supervised learning is used to predict the most optimal next move combined with a reinforcement learning algorithm, where a value network is trained simultaneously to predict the expected outcome from the current state of the board (Silver et al., 2016). However, the need for human expert data in AlphaGo motivates researchers to develop AlphaGo Zero, which trains itself entirely via self-play using Monte Carlo Tree Search (Silver, Schrittwieser, et al., 2017). More recently, AlphaZero was proposed to generalize to games other than Go, such as chess, which may end in a drawn game (Silver, Hubert, et al., 2017).

However, all of the above algorithms contain a lot of hyperparameters, and it is important to assess the effects of various hyperparameters on the overall game performance and training convergence, as well as for tuning purposes. While existing research has studied a vast amount of hyperparameters such as number of epochs and batch size (Wang, Emmerich, Preuss, & Plaat, 2019) in AlphaZero, there has been a general lack of discussion about them and their effects on the model performance. In this paper, we aim to study and understand several hyperparameters in the AlphaZero training process using the game Gomoku.

Gomoku is a 2-player game using a board with square grids. Each player takes turns placing stones on the intersections of the board, and whoever manages to place 5 consecutive same-colored stones (black or white) either horizontally, vertically or diagonally wins the game. Moreover, it has been formally proven in 2001 that if playing optimally, the player who starts the game will at worst end with a draw (Wgner & Virg, 2001).

Our paper mainly consists of evaluating hypotheses about AlphaZero hyperparameters. However, aside from the experimental section of our work, we also implement an graphic user interface for humans to play against AI players, trained with either basic Monte Carlo Tree Search or AlphaZero. Our paper is organized as follows. Section 2 talks about our implementation, our hypotheses about the hyperparameter space and our experimental design. Section 3 showcases our results and explains the implications. Section 4 summarizes our work and discusses potential future work.

2 Methods

Our work consists of both a simple GUI implementation of the game Gomoku and some AI players implementation, as well as experiments to evaluate hypotheses about the underlying hyperparameter space. The game implementation is based on the Kaixing's final project of CS111. We added two AI players – MCTS players and AlphaZero players in the game. A snapshot of our GUI is shown below in Figure 1.

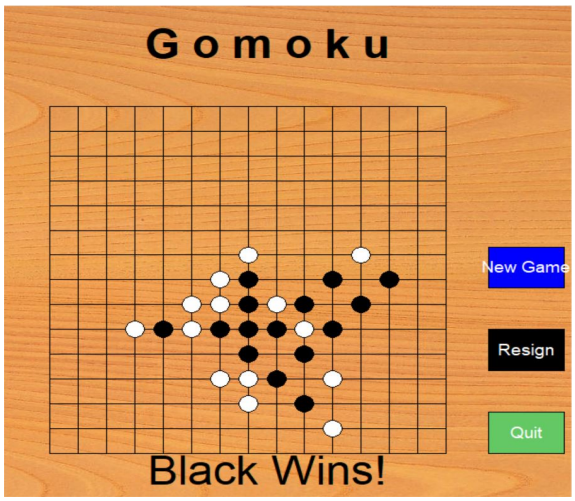


Figure 1: Graphic User Interface of our Gomoku game implementation

For our experiments, first we are interested in the effects of changing the input size to the neural network in AlphaZero. Unlike Go, which is not completely observable from the current state of the game, Gomoku can be fully represented by the current board. In the original AlphaGo Zero implementation, previous 7 moves are taken as additional inputs. However, in AlphaZero, the same is not required for simpler games such as Chess, Shogi or Gomoku. However, since good moves in Gomoku tend to be placed near previous moves, we hypothesize that including more previous moves as input to the neural network training will improve the model performance. In addition, we hypothesize that by providing more information along side each game state, the networks will develop more complicated game strategies and thus take longer to converge.

Next, we also want to assess the effects of transfer learning in training AlphaZero to play Gomoku. In short, transfer learning refers to machine learning practices where the result of a trained model is used to initialize another modeling task. This is often done to speed up training and improve performance with overall less resources. In the case of AlphaZero Gomoku, we are interested in the effects of training a larger board with weights transferred from a trained model using a smaller board, versus using randomly initialized weights.

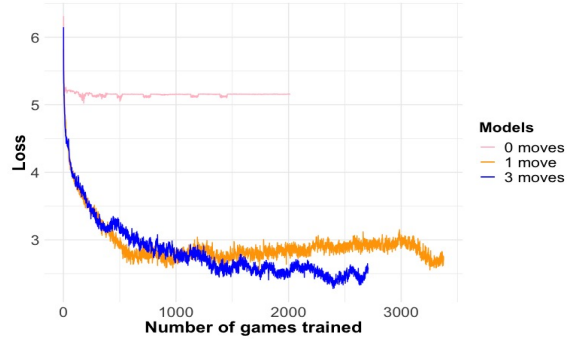
We will conduct 2 different sets of experiments to study the above hypotheses. For our first hypothesis, we are running 3 models, each with 0, 1 and 3 previous moves per player as additional input on an 8x8 board. For our second hypothesis, we will train 2 models both on 11x11 boards, one with randomly initialized weights and one with weights transferred from the trained 8x8 model. We will load the 8x8 pre-trained model into new 11x11 network. As most of the residual network layers can be re-used, we only use zero-padding for any mismatched positions of input and output layer. We will use a fully-implemented AlphaZero Gomoku repository to conduct the experiments (https://github.com/initial-h/AlphaZero_Gomoku_MPI) given the time constraint of building an AlphaZero implementation from scratch. The above models are ran in Google Cloud Engine under 5 different instances. One remote instance has the configuration of 7.2GB 8 vCPUs Ubuntu 18.04, and the other 4 instances have the 14.4GB 16 vCPUs Ubuntu 18.04 configuration. We let all models train for 4000 self-play games, and we set all other hyperparameters as the same for all 5 models (for specific values please see train.py).

We will compare the performance and convergence rate of each model in our experiments using network loss, entropy and winning rate against baseline MCTS. Network loss measures the deviations of predicted game values and action probabilities from actual training data in self-play. Entropy measures the extent of uncertainty of the action-probability distribution. In terms of our training, a lower entropy indicates more certainty of our model to choose a particular action. Furthermore, after every 100 game training, we let the trained network player play against a

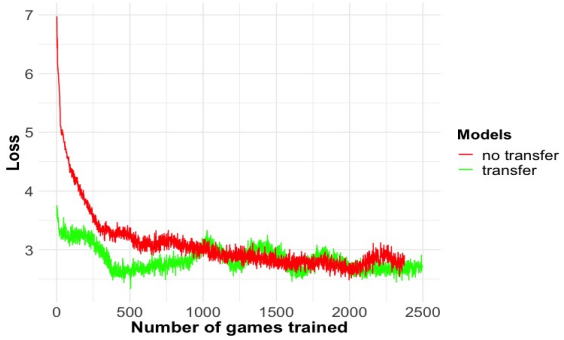
basic MCTS player starting with 1000 rollouts for 10 games, and compute the resulting winning rate. We increase the number of MCTS rollouts of the opponent player by 100 (thus increasing the ability of the basic MCTS player) every time we observe 100% win rate by AlphaZero, in order to differentiate the performances between models that have the same winning rate against the same MCTS player. AlphaZero is also played using a MCTS, only that instead of random rollout, the network outputs an action distribution and value, which MCTS uses to backpropagate. We let them play for 5 games each, with them respectively starting, given the known bias of Gomoku towards the starting player.

3 Results

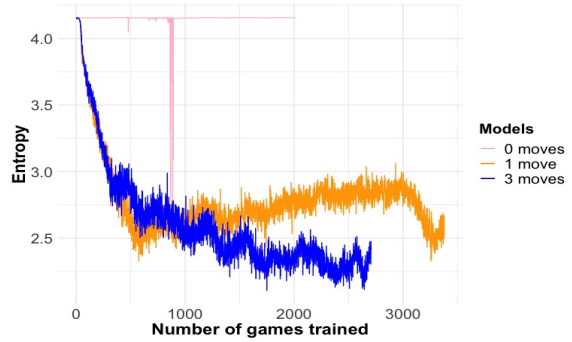
In this section, we will illustrate the results of our experiments. The line plots for loss, entropy and winning rate are shown below in Figure 2. Note that, given time and resource constraints (limited Google Cloud credits), none of the models manage to train for all 4000 games, and given the varying training speed of each model, the actual number of games trained can be observed from the x-axis for each plot.



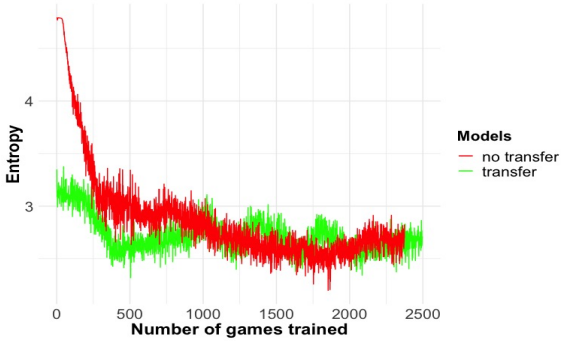
Loss for Hypothesis 1



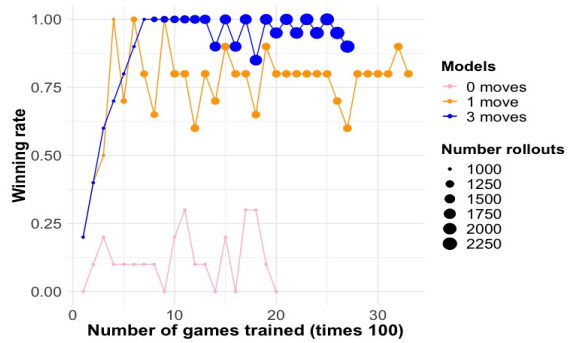
Hypothesis 2



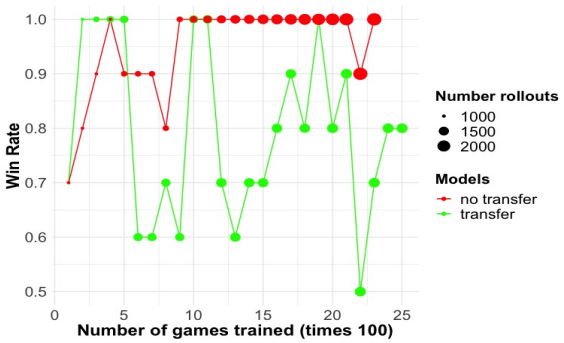
Entropy for Hypothesis 1



Hypothesis 2



Winning rate for Hypothesis 1



Hypothesis 2

Figure 2: Results for trained models using loss, entropy and winning rate

4 Discussion

We find that, for our first hypothesis, model without any input of previous moves seems to have the worst performance and the quickest convergence. It has significantly smaller winning rate (0 0.3) against MCTS players with only 1000 rollouts. We observe pretty steady loss and entropy (although we do not understand the sudden drop in entropy), which seems to suggest its failure to develop proper strategy due to lack of additional information.

On the other hand, once we provide additional moves as input, the loss and entropy still have not converged. This may be because that given additional information, it takes more self-play games for the models to fully explore all the possible combinations of input and develop proper strategies. However, we see that model with 3 moves per player as input has overall lower loss and entropy, which seems to suggest its higher predicting power. Moreover, the winning rate of model 3 is also much higher than that of the other 2 and has more number of rollouts of its opponent.

Overall, we find that with more moves per player as additional input, the model seems to converge much slower, but also with better predictions overall (low loss) and more certainty in move selection (low entropy), as well as significantly higher winning rate. Thus, even though AlphaZero does not require previous moves to fully represent its current state, it seems that a good strategy demands more information as input.

For our second hypothesis, both models end up with around 2500 out of 4000 total games trained. Both models seem to have converged in terms of loss and entropy. The model with transferred weights learns slightly faster during the first 800 games, but soon converges to similar loss and entropy as the other model.

In addition, in terms of the winning rate, it is interesting to see that the model with randomly initialized weights constantly outperforms the transferred model after the first 500 games. This may be because the transferred model does not perform well once the game on the 11x11 board goes onto areas outside of the 8x8 range, since the weights are trained using a 8x8 board. Specifically, since the 8x8 model may be trained to ignore moves on the edges of the 8x8 board (since limited space makes it harder to win), once the weights get transferred to the 11x11 model, the transferred model may be tempted to ignore those positions (i.e., not play on the 8x8 edges of the 11x11 board), which may in turn jeopardize its performance. Moreover, the model with randomly initialized weights also has a significantly lower variability than the transferred model. We think that the transferred model may have very variable performances since it is gradually trying to adjust its weights to adapt to the larger board.

In terms of future work for our project, we are interested in further investigating our first hypothesis by exploring the specific effects of knowing more previous moves. Specifically, we wish to visualize the strategic attention of the AlphaZero player and assess the differences in the amount of attention given to moves in close proximity. Since we already know that more previous moves as input seem to increase the accuracy of AlphaZero, we wonder if the increase in the AI performance is directly correlated with the strategy of playing in the same proximity of past moves, and if increasing the number of previous moves as input will decrease the attention region (i.e., more certain about which move to pick). Last but not least, if given more time and computing power, we hope to perform more training and on a larger board to observe any changes to the results for both of our hypotheses.

References

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... Hassabis,

D. (2016, January). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. doi: 10.1038/nature16961

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, *abs/1712.01815*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354-359.

Wang, H., Emmerich, M., Preuss, M., & Plaat, A. (2019, 03). Hyper-parameter sweep on alphazero general.

Wgner, J., & Virg, I. (2001). Solving renju. *ICGA JOURNAL*, *24*, 30–34.