AlphaGomoku: An Ablation Study of AlphaZero on the Game Gomoku

Siyu Liu[1] , Xueying Hu[2], Yanpei Tian[1]

[1]Department of Electrical Engineering
[2]Department of Mechanical Engineering

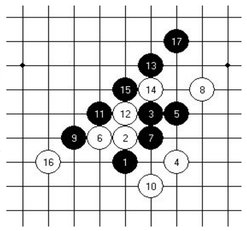
Stanford
Department of Computer Science

Motivation & Background

First introduced by DeepMind in 2018, AlphaZero has been attracting much attention due to its superior ability to achieve super-human level on different games. In this project, we plan to perform **an ablation study on different building blocks of AlphaZero**: State Representation, Neural Network Architecture, and Monte Carlo Tree Search (MCTS). We plan to identify 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. Ideally, our findings can guide us to find new approaches to improve the current model.

Gomoku

Gomoku is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board. In a game, the two players play adversarially by forming an unbroken chain of five stones to win the game. Traditionally, the game is played on a board of size 15×15 or 19×19. We use a board of **8×8 in our simulation** environment to expedite the computation process.



RL Algorithm

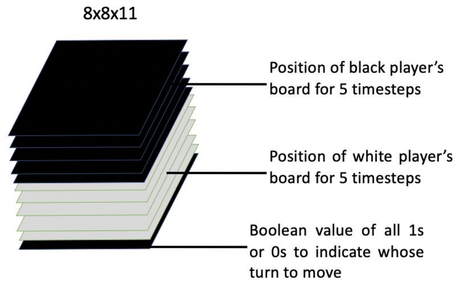
The RL algorithm involves the following parts:

1. **Self play**: During training, the agent is given no supervision from human. The agent is trained on the data collected by itself according to the following loss function:

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \vec{\pi}_t \cdot \log(\vec{p}_\theta(s_t))$$

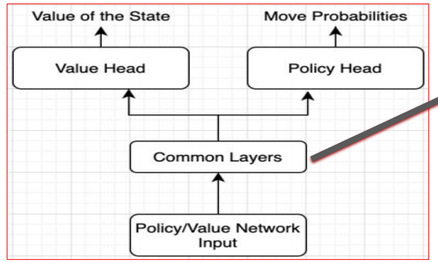
The first term is the L2 loss between the predicted reward and the actual reward observed by the agent during data collection; The second term is the cross-entropy loss between the predicted action distribution and the actual distribution during MCTS.

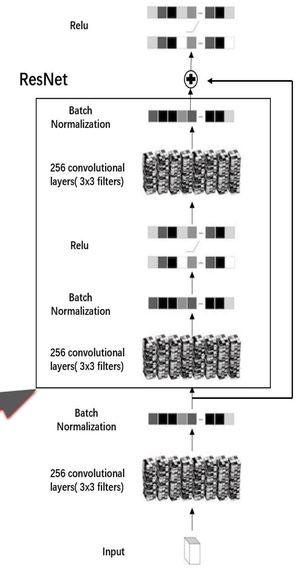
2. **State representation**, shown right, is made of recent history and current player indicator.



Neural Network Architecture

The neural network is comprised of three parts: **a stack of ResNet, a Policy head, and a Value head**. The input is first fed into the stack of ResNet. Then, the intermediate result from the stack is fed separately into policy head and value head to generate the value estimation and policy distribution.

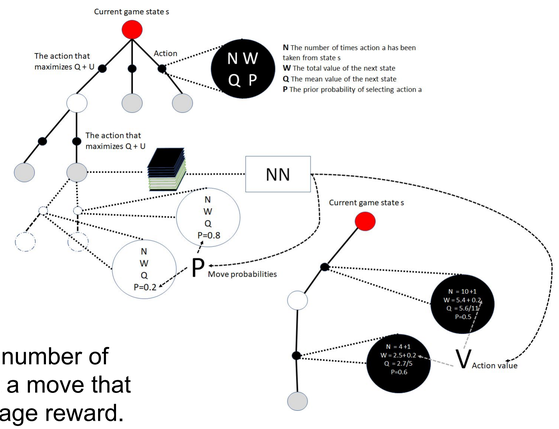




MCTS (Monte Carlo Tree Search)

Simulations:
1.Choose action that maximizes Q
2.Continue until terminal node
3.Back up previous edges
N->N+1
W->W + v
Q = W / N

After a predefined number of simulations, select a move that maximize the average reward.



Experiments

1. **Change of state representation**: change the amount of recent history in the state;
2. **Change of the NN architecture**: variate the number of ResNet in the stack;
3. **Change of MCTS**: use TD style estimate to speed up the data collection.

TD-Style Estimation

Note: TD style estimate in MTCS is a way to speed up data collection during rollout. Instead of playing to the end of the episode, we can use the current neural network to give an estimated value.

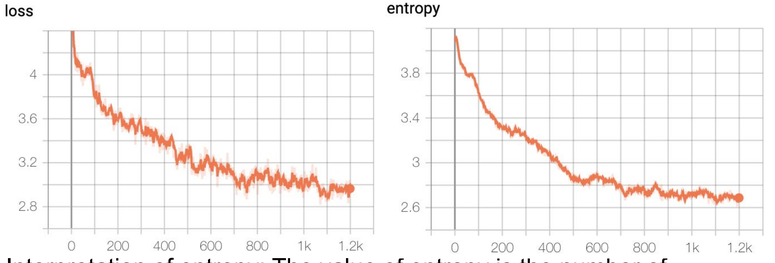
Algorithm 1: TD estimate of MCTS Rollout

Result: Estimated reward value
while *While game is not end* **do**
  perform rollout as usual;
  if *board is more that half occupied and (V(s) is greater than 0.99 or V(s) is less than -0.99)* **then**
    return V(s)
  else
end

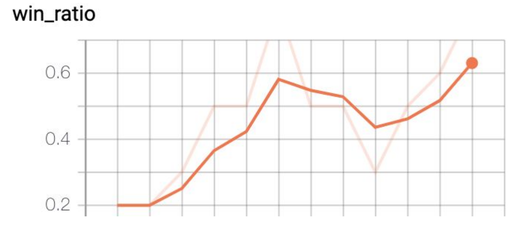
Results & Discussion

We have trained the following configuration for 1200 epochs:
ResNet stack of size 1;
11 channel state input: each player has 5 most recent playing history;
TD style reward estimation enabled.

Loss Curve and Entropy Curve:



Interpretation of entropy: The value of entropy is the number of different actions that the agent is indifferent to choose from.
Win Ratio Curve and Number of Rollouts:



Note: Despite effective training (reflected by the trending up win ratio), the variance still pose an issue to solve in future work.

Future Work

To further improve the performance of our implementation, we can use UCT (Upper Confidence Tree Search) to select action during MCTS. **UCT search** algorithm balances exploration and exploitation by assigning higher weights to actions that get less explored.