Analysis and Implementation of Deep Reinforcement Learning Based Gobang

UCAS

Abstract

One of the long-term goals of artificial intelligence is to develop an algorithm that can start learning from scratch and can gradually learn superhuman proficiency in challenging domains. In recent years, deep learning and reinforcement learning have been widely used in the field of artificial intelligence, and a large number of achievements have emerged. In the case of Go, AlphaGo and AlphaGoZero developed by DeepMind became the first program that defeat the Go world champion. AlphaGo uses deep networks in Monte Carlo tree search to perform position evaluation and move selection. These deep networks use supervised learning methods to train the system with the human expert's Go game records and use reinforcement learning methods to train the system by letting the system play games with itself. Further, AlphaGoZero uses reinforcement learning to train the system, which needs no human Go data, guidance, and related knowledge except the basic Go rules. AlphaGoZero's deep network can simultaneously process position evaluation to predict winners and make move selection. This deep network can enhance the advantages of tree search, so that in the next iteration, it ensures a higher quality moves. What makes AlphaGoZero so amazing is that it can achieve an extraordinary ability in a short period of time despite learning from scratch.

In this paper,we will re-implement the algorithm in AlphaGoZero's paper and apply it to gobang game. We are going to let the program start learning from the scratch, generate training data through self-play and learn to become stronger gradually. Considering the shortage of hardware resources, we do some trade-offs as well as some optimizations during the process of re-implementation and training, in order to obtain a human-level Gobang program. We name our trained program as AlphaGobangZero. Finally, we do some experiments to test the performance of AlphaGobangZero, which shows that under the current hardware situation, AlphaGobangZero can reach the same level as amateur human players.

1. Introduction

As a puzzle and strategy game, board games combine intelligence, physical strength and willpower. They are widely enjoyed by the public. Compared with other games, board games are more portable, and they can be transplanted to the mobile terminal very well. As the rapid development of the mobile Internet, it has become one of the most important ways for people to enjoy themselves during their leisure. There are lots of different board games in the app market, and each board game uses a different algorithm to implement. The same feature of these algorithms is that they need plenty of data from expert games and to imitate the way of expert players' decision. Manually extracting domain knowledge is often costly, and human expert data is often expensive, unreliable or even unavailable. Even if reliable expert data are available, the supervised learning system often achieves a performance bottleneck. This paper hopes to find a unified algorithm that does not require customization of domain knowledge. It only requires the basic rules of games, which can adaptively learn different games from the beginning, and eventually achieve good performance.

At present, artificial intelligence is prevalent, among which the deep reinforcement learning algorithm represented by AlphaGo [5] and AlphaGoZero [6] has strong potential in application prospects. This paper will draw the idea of the paper of AlphaGoZero [6] and use Gobang as an example to implement a deep reinforcement learning version of the Gobang program AlphaGobangZero. Gobang is a two-player pure strategy board game in which both sides use black and white stones respectively. Both players can play the stones in the intersection of horizontal lines and vertical lines on the board. The side who first gets five linked stones wins. Experiments show that AlphaGobang has great performance. Our work also shows that it is very convenient to apply the deep reinforcement learning algorithm to other board games, achieving more powerful performance than traditional supervised learning methods and heuristic search methods.

The subsequent sections will be arranged as follows:

First of all, this paper will introduce related work, including the mainstream algorithm for implementing board games such as Gobang, and the deep reinforcement learning

algorithm in AlphaGoZero paper.

Secondly, this paper will explain in detail how to use the deep reinforcement learning algorithm to model the Gobang problem, how to abstract different components from the deep learning framework, and how to design the class diagram between different components. Considering the hardware shortage, we will make some trade-off, optimization, and adjustment to the algorithm, including adjustment of network structure, construction of input features, assessment of construction models, optimization of reinforcement learning algorithm, dynamic adjustment strategy of learning rate, and data augmentation strategy.

Thirdly, we will do different comparison experiments to examine the performance of AlphaGobangZero. These experiments include chess competitions in which we examine the performance of AlphaGobangZero against rollouts MCTS players and amateur human players, and parameter comparison experiments including network parameters (such as the number of convolution kernels, number of residual blocks and others.), Monte Carlo tree's search time, noise parameters, and exploration factor parameters. Finally, we will conduct network comparison experiments, such as feed-forward neural network, convolutional neural network and residual neural network.

Then we will summarize the work of this paper, including reinforcement learning modeling, algorithm design and implementation, and comparison experiments.

Finally, we will make conclusions and then figure out what can be improved in the future.

2. Related work

So far, all the board games are based on zero-sum game ideas. A zero-sum game means that the sum of the rewards of all players is zero or a constant. Based on this idea, a max-min search algorithm has been developed. This is the most commonly used zero-sum game search strategy. Minimax algorithms actually use depth-first search (DFS) to traverse all possible outcomes in the current situation and obtain next steps by maximizing their own rewards and minimizing the opponent's rewards. One of the most basic methods is to recursively exhaust the search tree. There are $b * d$ possibilities, b means the width of searching and d means the depth of searching. Finally the algorithm chooses the action with the highest probability of winning as a decision. Usually this kind of exhaustion is unreasonable because the search space is too huge. Therefore, it is necessary to design some methods for judging which actions will maximize their own rewards in the current situation, minimize the rewards of the opponent, and avoid unnecessary searches.

Most current methods utilize a value function for position evaluation, and a policy function for move selections, which represents the probability distribution of all possible actions in the next step. These two functions are learned in some way such as supervised learning-based of all expert players' records. Value function and policy function are ways to reduce the search space from two perspectives. The value function can reduce the depth of search. We can utilize it to evaluate the situation of games and predict which side is more likely to win. The core idea is to directly calculate the next step and it is not necessary to go to the end of MCTS to know who will win in the end. The policy function, which represents the probability distribution of the next possible actions, can reduce the breadth of the search and limit the next search steps to those actions with high probabilities. The core idea is to determine how to effectively transit between tree nodes, which means that it is not necessary to walk through all nodes.

At present, the value function or policy function is either based on a heuristic scoring function or based on a shallow, artificially designed feature linear combination. The weight of each feature is learned through supervised data. These methods include $\alpha - \beta$ pruning, Monte Carlo search tree and so on.

The core idea of $\alpha - \beta$ Pruning [3] is to maintain two values for each node in a very small search process, α, β. α represents a lower bound of the node's valuation known so far. If the maximum value of this node is required at this time, you can refer to it when calculating the value of its child node. For a child node that causes the parent node to have a value less than the lower bound α , no further search is required. Similarly, β represents an upper bound on the node's valuation known so far. If a minimum estimate of the node is required at this time, then reference to β can be made when evaluating the value of its child node. If the parent node has a child value greater than the upper bound α , no further search is required.

The core idea of the traditional Monte Carlo Search Tree (MCTS) [5] is to use Monte Carlo rollouts to randomly simulate the player's longest moving sequence and average all simulations. The significance of Monte Carlo Tree Search is that it can give a situation evaluation. According to its design, the search tree will automatically focus on "more searchable moves". If you find some good moves, Monte Carlo tree will expand the corresponding node and search it very deeply, which is similar to heuristic search. Finally, as the search tree grows, the Monte Carlo tree search will converge to a good solution if the number of simulations is large enough.

Both of the above methods are flawed. By using a simple heuristic scoring function, $\alpha - \beta$ pruning may cut off some valuable moves prematurely. And the traditional MCTS simulation will randomly sample actions at each step. Although the algorithm will converge when the number of simulations is large enough, the convergence speed is too slow.

In response to the above issues, AlphaGo [5] and Al-

phaGoZero [6] have made major improvements, introducing Deep Reinforcement Learning [4] to solve the problem of excessive search space. At present, deep neural networks have achieved unprecedented achievements in the visual and other fields. Inspired by this approach, AlphaGo and AlphaGoZero abandon the simple heuristic scoring function, and use deep neural networks to design and fit value function and policy function. At the same time, they no longer utilize rollouts to guide MCTS search. Instead, they utilize neural networks to guide the MCTS search, and collect environmental feedback with the help of the reinforcement of learning algorithms in order to continuously adjust the parameters of the network. Both the deep learning network and the MCTS have their own advantages and disadvantages. The deep networks evaluate the position directly based on the current board situation, while the MCTS looks ahead by looking at the moves in the future to evaluate the current situation. Therefore, The combination of both components can achieve better performance.

This article draws on the ideas of the AlphaGoZero algorithm. We will implement different components, including MCTS, neural networks, reinforcement learning algorithms, etc, and finally assemble them together to build a great Gobang program, AlphaGobangZero. According to [1], for the complete 15×15 board size, the state space of the Gobang is nearly 10^{105} and the game tree complexity is nearly 10^{70}, which is huge. Even if it is a 8×8 board, the game tree complexity reaches 10^{27}. Obviously, exhaustion is not wise. This article will solve this problem with the help of deep reinforcement learning. In addition, this paper will make some trade-offs and optimization adjustments due to hardware shortage, including adjustment of network structure, introducing attention mechanisms to construct feature maps to speed up convergence, Monte Carlo search process optimization, reinforcement learning algorithm optimization, dynamic learning rate adjustment strategy, data augmentation strategy, etc.

3. Algorithm

This section will introduce algorithms from Gobang problem modeling, algorithm design and implementation, algorithm optimization and so on.

3.1. Modeling the Gobang Problem

This paper will use reinforcement learning to construct model for Gobang problem, where Gobang refers to amateur Gobang, without considering some additional rules in professional Gobang. The core elements of reinforcement learning include: agent, environment, state space/observation space, action space, state transition, and reward. As shown in Fig. 1.

An agent has the ability to interact with environment. Reinforcement learning focuses on how an agent takes a series of behaviors in the environment to maximize the cumulative returns. Through reinforcement learning, an agent should know what action should be taken in a given state.

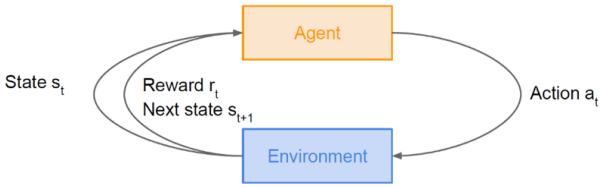

Figure 1. Core Element of Reinforcement Learning.

Therefore, the problem solved by agents is abstracted as environment. The goal of agents is to learn the internal state of the environment and the best strategy to deal with the environment through continuous interaction with the environment. The environment here is not the usual environment, but the environment that covers many core elements such as state space, action space, state transfer, feedback and so on.

Reinforcement learning is a trial-and-error learning. Since there is no direct guidance information, the agent needs to interact with the environment continuously and obtain the best strategy by the way of trial and error. The guidance information of reinforcement learning is very few, and is often given later (the last state), which leads to the problem of how to allocate the rewards to the previous state after obtaining positive or negative rewards.

The problem of Gobang can be modeled as the reinforcement learning problem mentioned above. In Gobang problem, We can abstract different parts corresponding to the core elements of reinforcement learning mentioned above.

(1) Agent

AlphaGobangZero, a Gobang program to be trained in this paper, is the agent in reinforcement learning.

(2) Environment

The rule constraint of Gobang is the environment. The environment mainly includes the way of playing Gobang, the chess board of Gobang, and the deciding rule of winning or losing in Gobang. The way of playing Gobang is that the black and white sides play with each other, and play a series of actions in order to maximize their probability of winning the game. The board of Gobang not only defines the action space, but also indirectly defines the state space and the transitions between states. The winning mechanism of Gobang defines the reward feedback. The game itself embodies the goal of reinforcement learning. In the process of playing game, a series of moves are needed to maximize the probability of winning the final victory. This is consistent with the goal of reinforcement learning in the environment in order to achieve the goal of maximizing cumulative rewards.

(3) Action Space

All possible moves under each position which corre-

spond to each location of the board construct the action space.

(4) State Space

A set of positions at each moment constructs state space. The state space of the Gobang is the same as the observation space.

(5) State Transition

The transition of previous position switching to the next position at each moment corresponds to state transition.

(6) Reward

The winning and losing decision mechanism of Gobang represents the reward feedback mechanism of reinforcement learning: the winning and losing mechanism of the Gobang provides that the black and white sides will win if five stones is continuously in a line. Therefore, if one side have five stones continuously in a line first, the game ends and rewards generate. for example, the winning party's reward is 1, and the failure party's reward is -1.

In a word, our Gobang program, AlphaGobangZero will play with itself and constantly switching from black to white repeatedly during games. Every time, it stands on one side and selects moves to maximize the probability of winning for the corresponding side. At the end of each game, AlphaGobangZero receives feedback, adjusts the decision of policy, and continues play with itself the next game. Repeatedly, it constantly receives feedback from the environment, adjusts itself, and ultimately trains a strong Gobang program.

3.2. Design of the algorithm

This part of algorithm design will be described in terms of the core process design of the algorithm, the core component design and the reinforcement learning framework design.

3.2.1 Design of The Core Processes

The whole algorithm includes three core processes, the self-play process, which produces training data, neural network optimization process, and model evaluation process. As shown in Fig. 2.

In the process of self playing, the previous neural network will be used to guide MCTS simulation and generate training data. In the process of neural network iterative optimization, the mini batch is randomly sampled from the training data, and the loss function is optimized. In the process of model evaluation, a checkpoint will be set up periodically to evaluate the model. It will not only observe the change of the loss, but also apply the current model to play the game with the previous optimal model, record the wining rate, and update the optimal model according to the rate. The schematic diagram of self play and neural network training process is as shown in Fig. 3.

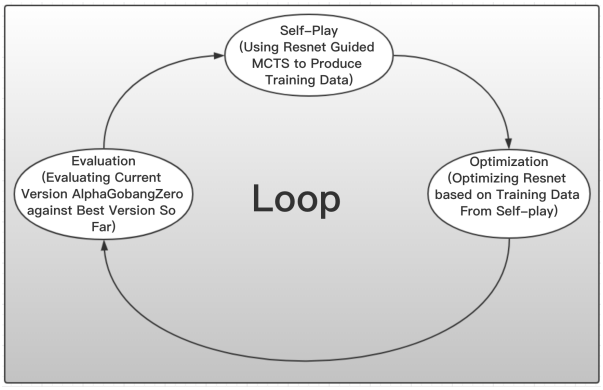

Figure 2. Three Core Processes.

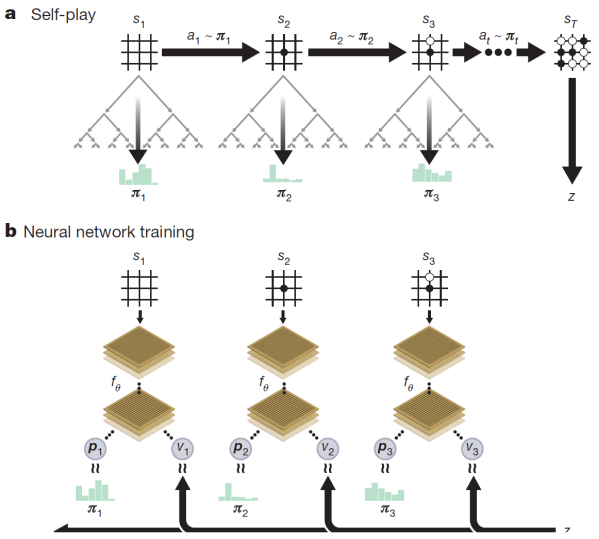

Figure 3. Self-Play and Optimization.

In the above figure, the self-play process uses MCTS to generate the probability distribution $\pi$of the action, and the final feedback of each game is z. Each position s and the label π, z together constitute training sample (s, π, z), and then transferred into the neural network to learn.

Two core components are involved in the above core process, neural network structure components and MCTS components. The details of the design and implementation of these two components are described in section 3.2.2 and 3.2.3.

3.2.2 Design of Neural Network Structure

The neural network architecture adopted in this paper is deep residual neural network. The network can solve the degradation problem, that is, the accuracy rate will rise first and then saturate, and the continuous increase in depth leads to the decrease of accuracy. This is not the problem

of overfitting, because the error of the training set itself will increase as the error increases on the test set. Deep Residual Learning enables the network to learn the residuals directly by introducing short connections, rather than learning a complete output, which can solve the problem of degradation well.

In AlphaGoZero, the author replaced the convolution neural network used in AlphaGo as a residual neural network, while the rest of the conditions were all consistent with AlphaGo. It eventually brought about 600 points of improvement, indicating that ResNet could learn more useful features.

This paper mainly uses ResNet [2] neural network to train Gobang model, AlphaGobangZero. The structure of the network is basically the same as that in the paper of AlphaGoZero. It mainly consists of 3 parts: common layers, policy head and value head. Common layers is shared by policy head and value head, the part of policy head will compute the probability distribution of the action, and the part of value head generates probability of winning. Among them, common layers consists of convolution blocks and residual blocks, and policy head or value Head mainly consist of convolutional layer and fully-connected layer. Considering the limit of hardware resources, we only uses 2 residual blocks, and the rest of the structure is consistent with the paper of AlphaGoZero [6].

The complete schematic diagram is as shown in Fig. 4:

Common Layers shared by Policy Head and Value Head consist of a single convolutional block followed by 2 residual blocks.

The Convolutional Block applies the following modules:

- A convolution of 256 filters of kernel size 3×3 with stride 1.
- Batch normalization.
- Relu, A rectifier nonlinearity.

The 2 Residual Blocks applies the following modules:

- A convolution of 256 filters of kernel size 3×3 with stride 1.
- Batch normalization.
- A convolution of 256 filters of kernel size 3×3 with stride 1.
- Batch normalization.
- Relu, A rectifier nonlinearity.
- A skip connection that adds the input to the block.
- Relu, A rectifier nonlinearity.

The output of the residual tower is passed into two separate heads for computing the policy and value.

The Policy Head applies the following modules:

- A convolution of 2 filters of kernel size 1×1 with stride 1.

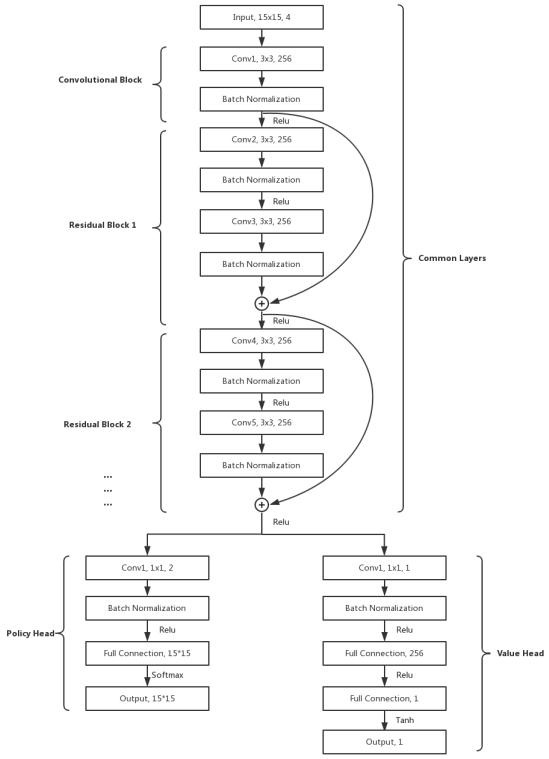

Figure 4. Network Architecture.

- Batch normalization.
- Relu, A rectifier nonlinearity.
- A fully connected linear layer that outputs a vector of size 6 6=36, corresponding to logit probabilities for all intersections and the pass move.
- Softmax, Outputting the probability distribution of action.

The Value Head applies the following modules:

- A convolution of 1 filters of kernel size 1×1 with stride 1.
- Batch normalization.
- Relu, A rectifier nonlinearity.
- A fully connected linear layer to a hidden layer of size 256.
- Tanh, Outputting a winning score in the range $[-1, 1]$.

3.2.3 Design of Monte Carlo simulation components

Monte Carlo Tree Search is used to evaluate the current chess situation by looking ahead. According to the paper of AlphaGoZero, we can summarize a general Monte Carlo framework, which contains several important steps in the MCTS simulation process: Select, Expand, Evaluate, Backup and Play.

The MCTS simulation process uses neural network $f_\theta$to guide. Each edge of the search tree stores a prior probability $P(s,a)$, the number of visits $N(s,a)$ and the action benefit value $Q(s,a)$.

(1) Select

Every simulation starts from the current root node which corresponds to the current position, continuously iterates through the tree and selects move a in order to maximize the sum of Q value and confidence upper bound $Q(s,a) + U(s,a)$, and go on until arriving the leaf node.

(2) Expand and Evaluate

Its time to expand and evaluate when the number of leaf nodes reaches a threshold. Neural networks are used to generate prior probabilities and evaluation values. $(P(s',),V(s')) = f_\theta(s')$. This shows how the neural network guides Monte Carlo Tree Search.

(3) Backup: Update the number of visits and Q values at the edge of the leaf node. The formula of updating the value of Q : $Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|s,a\to s'} V(s')$

(4) Play

After the multiple simulation process, the search probability distribution$\pi = \alpha_\theta(s)$ is obtained, The search probability is proportional to the number of visit times $\pi_a \propto N(s,a)^{1/\tau}$, τ is called the temperature parameter. Then the final action a can be sampled using the distribution π.

The MCTS steps is as shown in Fig. 5.

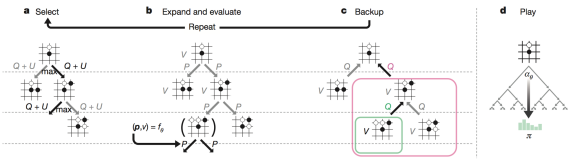

Figure 5. MCTS Steps.

3.3. Design of reinforcement learning framework

The above discussion is mainly about how AlphaGobangZero is designed (Agent). The key of reinforcement learning algorithm is the interaction between Agent and environment. Therefore, Agent and environment need to be unified in a reinforcement learning framework. This part will use the reinforcement learning algorithm of Policy Iteration, and unify the two core components, the residual neural network structure components and the MCTS simulation components into the reinforcement learning algorithm, and form a reinforcement learning framework based on the residual neural network and Monte Carlo Tree Search. The framework design is as shown in Fig. 6:

According to the above data flow, the position sample $(Position\ s)$ will be input into the residual neural network ResNet. Through Common layers, the initial probability of

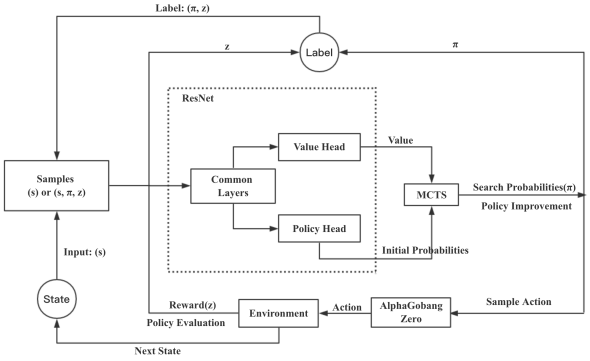

Figure 6. Reinforcement Learning Framework.

the current chess evaluation value and the initial probability of actions for the next step are output from the value head and policy head respectively. Then, MCTS executes the simulation process based on the two output values. The simulation process represents the policy improvement operation in the Policy Iteration reinforcement learning algorithm. After many simulations, the improved search probability π is finally obtained. Based on the search probability and samples action, the sampled action get from the agent AlphaGobangZero will be input into the environment (the Gobang chessboard). The environment will be responsible for two things, one part is to do state transition, that is, to switch to the next position, the other part is to judge whether the game is finished, if not, go on playing, otherwise, give the feedback which represents the value of z will be output. This step represents the Policy Evaluation operation of Policy Iteration reinforcement learning algorithm. At this time, the return value z and the search probability π constitute the label, and the position s produced during the game constitutes the input, and the (s,π,z) together constitutes the sample. Then the sample will be feed into the neural network for training and optimization. The above procedures will be repeated until AlphaGobangZero achieves expected performance.

3.4. Implementation of the algorithm

In order to achieve the algorithm better, it is necessary to abstract and decouple different components and objects from the algorithm according to the core process, core components and reinforcement learning algorithm framework mentioned above. Core objects include: Board, Game, Player, MCTS, TreeNode, residual neural network Resnet, neural network wrapper class PolicyValueNet. The detailed class diagram is as shown in Fig. 7:

The responsibilities of different objects are as follows.

- Board: Gobang chessboard class, representing the gobang chessboard. Board takes on the main rules of the game. The core functions include initializ-

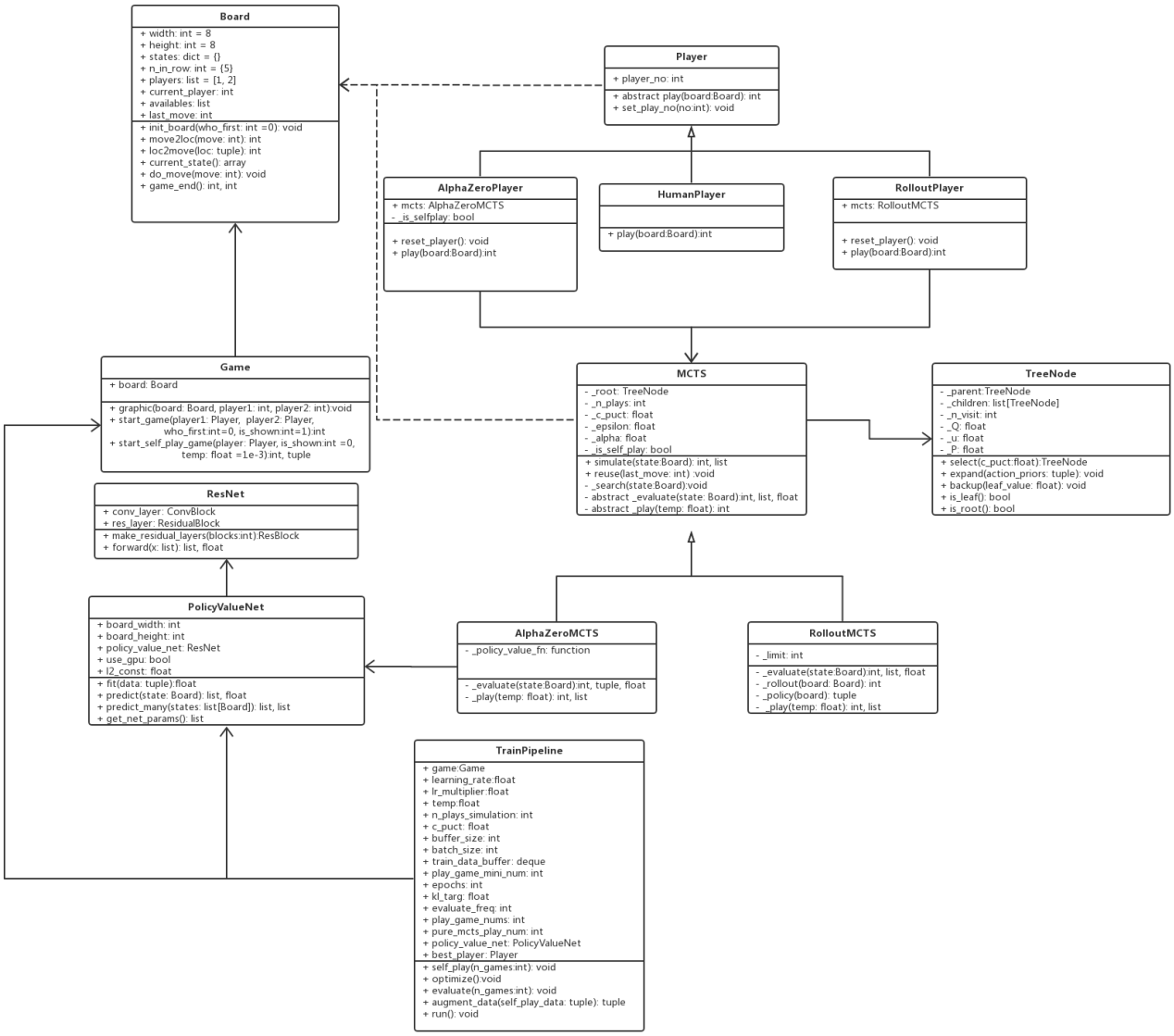

Figure 7. Class Diagram.

ing chessboard, building features from checkerboard, playing gobang, judging whether the game is over or not.

- Game: Game class, representing a game. It takes on the task of playing gobang, including the two sides playing, self-play and so on.
- Player: Player base class. represents the player. AlphaZeroPlayer, RolloutPlayer and HumanPlayer are subclasses of Player.
- MCTS: Monte Carlo search tree. The core functions includes Monte Carlo simulation process, MCTS reuse, call neural network for chgobang surface situation evaluation and calculation of action probability distribution. AlphaZeroMCTS and RolloutMCTS are subclasses.
- TreeNode: Monte Carlo search tree node, takes on the four stage of universal Monte Carlo tree search, including select, expand, backup and so on.
- ResNet: Residual neural network. The residual neural network is realized by using Pytorch. The core methods include building multiple residual blocks, forward propagation, loss optimization, and so on.
- PolicyValueNet: ResNet wrapper class, wrap Resnet, and serve as the interface for external AlphaZeroPlayer. The core methods include the training method fit, the prediction method predict (for the evaluate

phase of AlphaZeroMCTS), the prediction method predict_many (used for training evaluation model), the acquisition of model parameters (used for saving the model), and so on.

The pseudo code of 3 core processes is as follows.

Fistly, random weight θ_0 for ResNet f_{θ_0}. Then, at each iteration $i > 0$, self-play games are generated in the following steps.

step 1: MCTS samples search probabilities π_t based on the neural network from the previous iteration $f_{\theta_{i-1}}$, $\pi_t = \alpha_{\theta_{i-1}}(s_t), t = 1, 2, , T$.

step 2: move is sampled from π_t.

step 3: (s_t, π_t, z_t) for each t are stored for later training.

step 4: optimize new neural network f_{θ_i}. f_{θ_i} is trained in order to minimize the loss $\ell = (z - v)^2 - \pi^T log\boldsymbol{p} + c||\theta||^2$, loss ℓ makes $(\boldsymbol{p}, v) = f_\theta(s)$ more closely match the improved search probabilities and self-play winner (π, z).

Every 50 iterations, evaluate current f_{θ_i} guided MCTS player against $f_{\theta_{best}}$ guided MCTS player. if the win ratio of f_{θ_i} is larger than 55

3.5. Optimization of The Algorithm

This section will introduce aspects of algorithm optimization, such as feature engineering, data augmentation methods, experience replay, exploration-use tradeoffs, and dynamic adjustment of learning rates.

3.5.1 Data Preprocess

The data preprocessing involved in this paper includes feature engineering and data augmentation operations.

(1) Feature Engineering

AlphaGobangZero does not need to extract features about the knowledge of the Gobang field. This paper only extracts the feature of current game. The features mainly include four aspects:

- The binary feature map of the black side. The position of black chess pieces is set as 1 and the others set 0.
- The binary feature map of the white side. The position of white chess pieces is set as 1 and the others set 0.
- The unary feature map of current player. If the current turn is black, the chess pieces are all set as 1, and if it is white, they are all set as 0.
- The feature map of the position of the opponent's move in the previous step. The position of the opponent in the previous step is 1 and the others is 0.

The first three features are the same as those in the AlphaGoZero paper. The last feature map introduces a mechanism similar to attention, which makes the model to pay more attention to the opponent's move in the previous step and making the next move is near the previous move. Practice shows that the introduction of this feature can speed up the convergence of the model. The design of this feature takes advantage of the fact that Gomoku pays more attention to local characteristics. In addition, no field knowledge about Gobang is introduced.

(2) Data Augment

As the Gobang board has rotation invariance and mirror invariance, in this paper, data augmentation operations will be performed on each piece of game-generated observation data to improve the diversity of the data, and to a certain extent, the robustness of the convolution layer in the neural network can be improved. There are two methods of data augmentation, rotation and flipping. In this paper, the data of the chessboard will be rotated by 90 degrees, 180 degrees, and 270 degrees respectively, and the data will be flipped horizontally and flipped vertically. So that we can improve the diversity of the data.

3.5.2 Reinforcement Learning

The reinforcement learning algorithm used in this paper is Policy Iteration. As mentioned above, the two-phase policy improvement of Policy Iteration is that MCTS will improve the output probability of neural networks, to obtain search probabilities, and Policy Evaluation will use the feedback generated at the end of each game to evaluate. These two stages are iteratively executed in multiple rounds of chess and gradually improve the performance of AlphaGobangZero. There are two important issues that need to be solved in the learning process of reinforcement learning algorithms. The first one is the problem of correlation and non-stationary distribution between data, and the other is how to use and exploration of the data. These two issues will be discussed as follows.

(1) Data Correlation and Non-stationary Distribution

First of all is the correlation between data. There is a strong correlation between the chessboard observations produced in each game, and there are usually only one or two step differences. This is contrary to the hypothesis of data independence in deep learning, which assumes that the data samples are independent of each other. In order to use deep learning, it is necessary to solve the problem of strong correlation of data.

The second is the non-stationary distribution of the strategy. The reinforcement learning algorithm will learn the distribution of the strategy, and the distribution of the strategy is not stable. As the game progresses, the distribution will change. This is contrary to the assumption that the data is in the same distribution in deep learning, so the problem of non-stationary distribution must be solved.

This paper introduces the Experience Replay Mechanism to solve the above problems. Using an experience pool to store the observation data generated so far, usually to save the latest several chess data (eg 10000). During each train-

ing, instead of directly using the data generated in the previous round to train, we select random numbers data from the experience pool for training. Because most of the sampled data comes from different rounds of the game, the correlation between the data is solved to some extent. At the same time, this method can also solve the problem of non-stationary distribution of strategies. There may be large differences between the strategy distributions learned by using the data generated by a single round of chess playing. but if the data is generated from multiple rounds of matches, the result would be smooth to some extent.

Experiments show that this method can effectively solve the problem of data correlation and non-stationary distribution. The loss function value can be reduced steadily during the learning process.

(2) Exploitation and Exploration Trade-off

Another problem is the trade-off between exploitation and exploration. Exploitation refers to making the best decision under the current information. Exploration is trying different behaviors and collecting more information. Exploitation reflects the maximization of short-term benefits, which means that it is known, while exploration focuses on maximizing long-term benefits, but implies unknown. The best long-term strategy usually includes some measures to sacrifice some short-term benefits. By gathering more information, we can achieve the best long-term strategy. Therefore exploration and utilization are a contradiction.

In the Gobang game, exploitation means that each time the same game is played, one of the same moves that maximizes the current reward is selected. Exploration means that to get the long-term rewards, the current reward may be sacrificed, and more explorations will be conducted to find a more advantageous method. Obviously, during the training process, exploration is necessary. If every game is played in the same way, then the huge state space cannot be effectively excavated. Therefore, in the process of AlphaGobangZero self-play, it is necessary to introduce appropriate mechanisms to achieve exploration. Of course, after training a good model, you don't need to explore too much in the actual game, and you just need to choose the most favorable action each time.

In the self-play process of AlphaGobang Zero, this paper introduces three kinds of exploration mechanisms: Noise, Upper Bound, and Softmax temperature.

The noise method refers to adding noise to the MCTS simulation process or simulation results. In this paper, Dirichlet noise is added to the strategy distribution which is learned during the self-taught process, including adding noise to the probabilities of different branching from the root node of the MCTS simulation, and adding noise to the search probability after MCTS promotion. Let the noise ratio be ϵ, The parameter for the Dirichlet distribution is α. The strategy distribution before no noise is p, Then the policy distribution with noise p_{noise} is:

$$p_{noise} = (1 - \epsilon) * p + \epsilon * Dirichlet(\alpha)$$

Sampling based on the strategic distribution after adding noise can be explored to some extent.

The confidence upper bound method means that the number of visits per action will be recorded in the MCTS simulation process. Confidence in the upper bound will encourage MCTS to explore more actions with fewer visits, and using an exploration degree factor to measure the degree of emphasis on exploration, the confidence upper bound calculation formula is as follows:

$$u(s, a) = c_{puct} P(s, a) \frac{\sqrt{\sum_b N_r(s, b)}}{1 + N_r(s, a)}$$

Among them, $u(s, a)$ measures the confidence upper bound of action a in state s, the access probability will be higher as the confidence upper bound will be higher. c_{puct} is the exploratory degree factor, the system will be encouraged to exploration if this parameter is large. $P(s, a)$ is the prior probability, and $N_r(s, a)$ is the number of visits to the current location a, the system will be encouraged to exploration if this parameter is little.

The soft maximum temperature method refers to the introduction of temperature parameters when calculating the probability distribution of strategy. The formula is as follows:

$$\pi(a|s) = \frac{N(s, a)^{1/\tau}}{\sum_b N(s, b)^{1/\tau}}$$

Where $\pi(a|s)$ is the strategy distribution in state s. $N(s, a)$ is the number of visits per action after the MCTS simulation. τ is a positive temperature parameter. The greater the temperature, the more likely it is to have equal probability, that is to encourage exploration. The smaller, the sampling tends to have high probability of action, that is to encourage the exploitation.

3.5.3 Deep Learning

This section will introduce the optimization methods in the process of deep learning. It mainly includes the dynamic adjustment of learning rate and the evaluation method of the model.

(1) Learning Rate Adjustment

The dynamic adjustment of the learning rate is to ensure the stability of the learning process. Because there are always certain non-stationary distribution problems in the reinforcement learning problem, We hope that in the learning process, the strategy distribution learned by the neural network should be a gradual and steady process, and it does

not want the strategic distribution to show significant oscillations between different iterations.

This paper will dynamically adjust the learning rate from the steady changes in the distribution of strategies. Each time you study, you will examine the KullbackLeibler divergence between the previous learned strategy distribution and the currently learned strategy distribution to measure the degree of difference between the two strategy distributions. KL divergence formula is as follows:

$$KL(p||q) = \sum_{i=1}^{N} p(x_i).log\frac{p(x_i)}{q(x_i)}$$

p is the strategy distribution of the previous iteration. q is the strategic distribution of the current round. The larger the KL divergence, the greater the difference in the distribution of the two strategies. We hope that in the learning process, the strategy distribution will not change too much between successive iterations. If the change is too large, the learning rate should be decreased, making the subsequent iteration process more stable in learning. If the change is too small, increase the learning rate.

Another way to adjust the learning rate is to adjust to changes in loss. If the loss does not decrease after continuous multi-turn, then the learning rate should be reduced; Otherwise, if the loss is decreasing, you can increase the learning rate to speed up the convergence.

(2) Evaluation

This paper will set checkpoints during the process of neural network optimization to periodically evaluate the performance of the network. The goal is to select the best neural network so far and generate higher quality training data in the new game. The method of evaluating neural networks mainly examines the ability of the MCTS player under the guidance of the neural network, called NetworkMCTSPlayer. The evaluation will use pre-set opponents, let the new neural network guided MCTS players and opponents set up actual game, using the winning rate to evaluate the new neural network. For models that perform well, they will be saved and used for the next iteration to generate new training data; For poorly performing models, discard them. As the model continues to be stronger, it is clearly unreasonable to use the same opponent throughout.

This paper will use different opponents in different phases to evaluate. First, in the early stages of neural network training, performance was poor. Therefore, the MCTS adversary under the guidance of a random strategy is used for evaluation, which is Rollout MCTSPlayer, and the number of MCTS simulations is small at the beginning, such as 1000. As the capabilities of NetworkMCTSPlayer continue to increase, the number of MCTS simulations of RolloutMCTSPlayer is increased accordingly, and the opponent's ability is enhanced. If the number of simulations increases to large enough, NetworkMCTSPlayer still wins with an overwhelming advantage (100% win rate), then switch the opponents. Using the best NetworkMCTSPlayer as an opponent for evaluation so far, if it can beat the best NetworkMCTSPlayer so far with 55%, just replace the best.

4. Experiments and Analysis

At the begining of this section, the performance of the final learned model was examined. And then AlphaGobangZero played against amateur humans in order to obtain the winning results. At the same time, the positions at each moment were observed in order to discover the domaoin knowledge of Gobang learned by AlphaGobangZero. Finally, comparative experiments were carried out, including performance comparison experiments of AlphaGobangZero under different parameters and different type of neural networks.

4.1. Experiments for AlphaGobangZero

Due to the current limited hardware resources, it is impossible to train a large-scale chess board. Therefore, the experimental object in this section is the performance of AlphaGobangZero, a Gobang program obtained under the 8×8 board size scale. AlphaGobangZero's structure has already been mentioned above, including 1 convolution block, 2 residual block, 1 policy head, 1 value head. The specific network parameters can refer to the network structure figure above Fig.4.

4.1.1 Loss Change

In this section, the changes in loss during the training of AlphaGobangZero were analyzed. The loss change curve is as shown in Fig. 8:

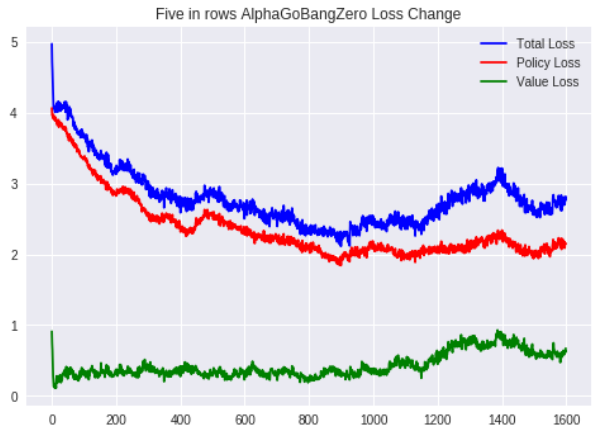

Figure 8. Loss Change Curve.

The green line in the figure is value loss, which represents the loss of Value Head. The red line is policy loss,

	RolloutMCTSPlayer	HumanPlayer
Win	30	11
Loss	0	9
Tie	0	10

Table 1. Results. Ours is better.

which represents the loss of Policy Head. The blue line is the total loss, including value loss, policy loss, and regularization items.

As can be seen in the figure 8, as the training progresses, the loss of policy is steadily declining, and the total loss is therefore steadily declining. However, the value loss has not changed much. with the increasing number of iterations, the value loss is increasing subsequently. The chiefly reason is probably because the number of residual block is too small(only two), and the convolutional layer in value Head uses only one convolution kernel, and the model is relatively simple, resulting in a degradation phenomenon when the training reaches a certain level. While it has been found that the lowest point of loss is not necessarily the best in actual competition. With the increasing number of iterations, the exploration of the state space is continuously strengthened. Even if the loss does not drop significantly, the actual performance may increase.

4.1.2 Competition Results

We examined the game results of AlphaGobangZero against rollouts MCTS player and human. A total of 30 games have be played in turn. The final outcome is as follows in Table 1.

The number of MTCS simulations of RolloutMCTSPlayer used above is set to 5000, and the number of MCTS simulations in AlphaGobangZero is 1000.

Despite the low number of MCTS simulations, AlphaGobangZero still beats rollout MCTS players with overwhelming advantages, wins all 30 games. In the process of confrontation with humans, it can be found that AlphaGobangZero has reached roughly the same level as humans. In addition to 10 draws, the number of wins and the number of failures were basically the same.

As can be seen from the figure above, with the increasing of iterations, the overall performance of AlphaGobangZero continues to increase, eventually reaching a level comparable to humans. The figure also shows that after peaking around 1100 rounds, the performance has declined, and then it continues to improve. The timing of this drop in performance coincides exactly with the degradation timing of the above loss curve, indicating that degradation has affected the performance of AlphaGobangZero to some extent.

4.1.3 Different Iterations

In this section, the games between AlphaGobangZero of different iteration rounds and AlphaGobangZero of the last round (1500) were examined. The above experiments have shown that the final round of AlphaGobangZero has comparable levels to humans. Therefore, this experiment studied the change of AlphaGobangZero capabilities as the iterations increase. The winning rate curve is as shown in Fig. 9.

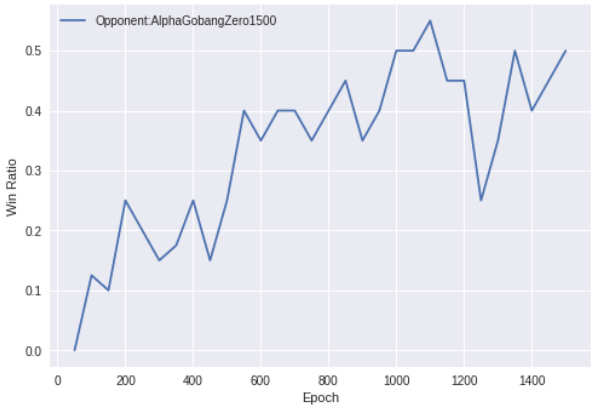

Figure 9. Winning rate of different iterations.

4.1.4 Parameter Settings

In this section, the performance of AlphaGobangZero trained under different parameter settings was examined. These parameters include the noise percentage of the Dirachlet during training, ϵ, the number of different Monte Carlo searches during training, n, and the different exploration degree factor during training, c_{puct}. For each experiment, only one parameter was considered. With the remaining parameters exactly the same, the performance comparison of one parameter under different values was launched.

First, the performance with different values of ϵ of AlphaGobangZero was analysed, including $\epsilon = 0, 0.2, 0.8$ and 30. 30 games were conducted between AlphaGobangZero with different ϵ, rollouts MCTS player and human. The final result of the games is shown in Fig. 10.

The first figure shows the winning ratio of $AlphaGobangZero$ with $\epsilon = 0$ against $\epsilon = 0.2$, $\epsilon = 0.8$, RandomRolloutMCTS and Human respectively. The second figure is the winning ratio of $AlphaGobangZero$ with $\epsilon = 0.2$ against the rest of the players. The third figure is as the same.

It can be seen that for the $\epsilon = 0$ without any noise, its performance is poor, it is only better than the rollous MCTS player, probably because it does not add any noise, making the exploration during training low, and can not fully

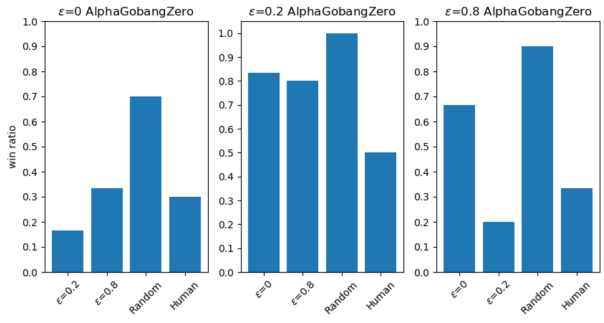

Figure 10. Results of different ϵ values.

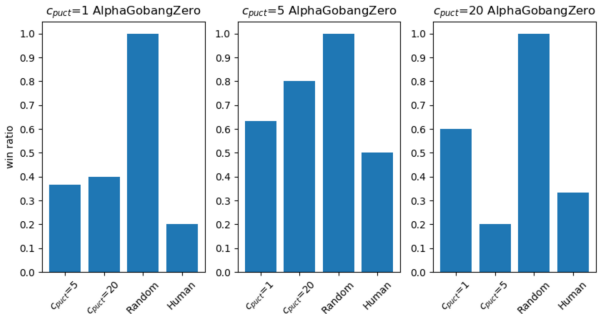

Figure 12. Results of different c_{puct} values.

explore the huge state space, resulting in poor final performance. AlphaGobangZero, which has a noise ratio of $\epsilon = 0.2$, has the best performance and beats other players to reach a level comparable to humans. For AlphaGobangZero with a noise ratio of 0.8, the performance is not as good as AlphaGobangZero with $\epsilon = 0.2$. The reason may be that the noise proportion is too large to play the role of exploitation. So we need to choose the right noise ratio.

Then the winning results of different Monte Carlo search times were examined, including $n = 10, 400, 1000$, against rollouts MCTS player and humans in 30 games. The result of the game is as shown in Fig. 11.

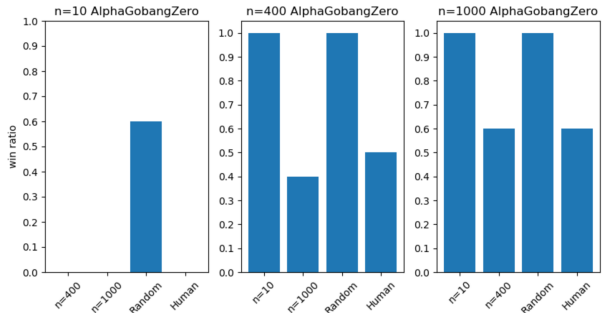

Figure 11. Results of different n values.

The first figure is the winning results of $n = 10$ against the other players. the second is the winning results of $n = 400$ against other players. and the third is the winning results of $n = 1000$ against other players. It can be seen that the performance increases as the number of searches increases. The more search times, the higher the performance. As the number of MCTS searches increases, the training time increases exponentially. Therefore, the training time must also be considered in practical applications.

Finally, experiments of AlphaGobangZero with different exploration degree factors c_{puct} were conducted, including $c_{puct} = 1, 5, 20$.AlphaGobangZero with different c_{puct} played against rollous MCTS player and humans in 30 games. The result is shown in Fig. 12.

The first figure is the winning results of $c_{puct} = 1$ against the other players. The second figure is the winning results of $c_{puct} = 5$ against the other players. And The third figure is the winning results of $c_{puct} = 20$ against the other players.

It can be seen that the models obtained under different training values are superior to the Rollous MCTS Player. $c_{puct} = 1$ does not perform as well as $c_{puct} = 5$ and humans. And $c_{puct} = 5$ performed best, reaching the same level as amateur humans. $c_{puct} = 20$ is slightly weaker than $c_{puct} = 5$. It can be seen from the above that the smaller the c_{puct} is, the lower the degree of exploration is in the MCTS search, and the less attention is paid to the prior probability of the output of the neural network, resulting in unsatisfactory performance. If c_{puct} is too large, the exploratory degree is too high and it depends too much on the prior probability of the output of the neural network. It does not pay much attention to the results obtained by the MCTS simulation and the performance is not ideal too. Therefore, a reasonable c_{puct} value is needed.

The final selected parameters were $\epsilon = 0.2, c_{puct} = 20, n = 400$. According to the paper in AlphaZero, the hyperparameter can be kept unchanged in subsequent experiments.

4.1.5 Domain Knowledge Learned by AlphaGobangZero

We observe that AlphaGobangZero gradually learned some field knowledge of Gobang in the training process. For example, the AlphaGobangZero training started with random moves, and any two consecutive steps were mostly far apart. With the increasing in the number of iterations, AlphaGobangZero learned the characteristics of the local-focused feature of Gobang, that is, the pieces between consecutive steps were mostly adjacent. In addition, we also found that AlphaGobangZero learn the "three-three links", which is when one player playing a stone, it will form two triples to obtain winning. As shown in Fig. 13.

It can be seen that AlphaGobangZero found "three-three links" chances at the 19th step, that is, (5,15,19), (13,11,19)

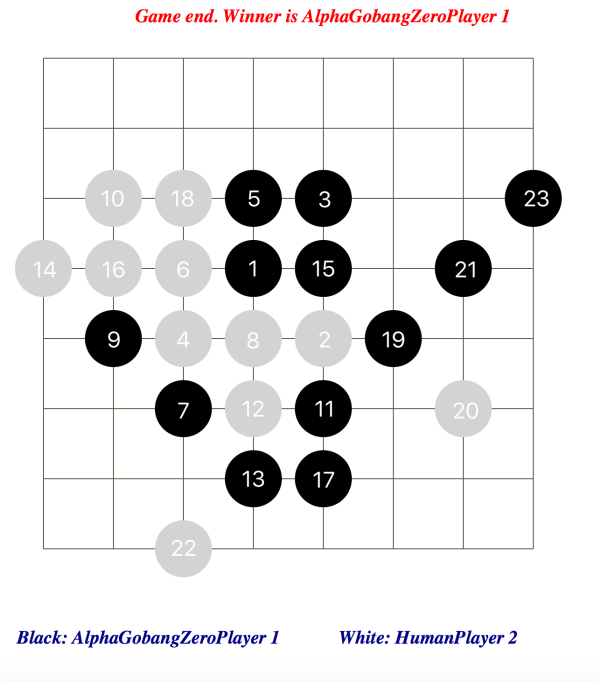

Figure 13. A game snapshot.

constitute a ”three-three links”, and then the human player blocked one of the three link at postion 20. Next time, AlphaGobangZero played a move in the position 21, forming a four-link, which means human had no hope to win. Finally AlphaGobangZero played the move in the position 23 and won.

It can be seen that AlphaGobangZero has learned some of the knowledge of Gobang in a certain degree.

4.1.6 Network Comparison

In this experiment, three neural networks with different structures were implemented, namely, feed-forward neural network, convolutional neural network, and residual network. The optimal models obtained from these three kinds of neural network training were selected. The residual network was trained with 1 residual block and 2 residual blocks respectively, and two models are obtained. There were 30 innings between every two models. Then, the results of the chess game with the Rollouts MCTS player and the results of the chess game with amateur human were joined. Get the following results in Fig.14.

The three pictures show the results of AlphaZero model (2 residual blocks), convolution neural network model and feed-forward neural network model respectively. It can be seen that the AlphaZero model is not dominant in the game with the convolutional neural network model. This is mainly due to the limitation of hardware conditions. The

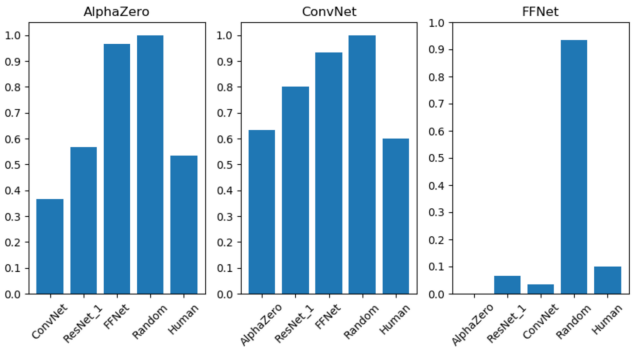

Figure 14. Results of different networks.

residual network used by AlphaZero contains only 2 residual blocks and the network is not deep, resulting in the lack of the advantage of the residual network. However, the model obtained by training two residual blocks is better than the model obtained by training only one residual block. The other results are more in line with our expectations. The training effect of feed forward network is the worst, but it is better than the random rollouts player. In the game with amateur human, the AlphaGobangZero performs well.

5. Conclusion and Future work

The main task of this article is to use the Gobang game as an example to re-implement the deep reinforcement learning algorithm in AlphaGoZero’s thesis. The article first introduced the current research’s related work, including deep learning, reinforcement learning, and general solutions for board games, the algorithm is then described in detail. First of all, starting from the perspective of the Gobang problem modeling, we discuss how to model the Gobang problem into a reinforcement learning problem. Then design and implement the algorithm, separates the core processes of the algorithm, abstracts out different components, and then re-designs the core components Including the neural network architecture and MCTS simulation components. And finally incorporates the core components into reinforcement learning. In the framework, a reinforcement learning framework based on deep residual network and MCTS is constructed, and how the data flow operates in this framework is described in detail. Then we draw a class diagram based on the above design to illustrate the responsibilities of different objects to be implemented in the code, and provided the pseudo-code of the core process. Next, we explains the optimization points existed in the process of algorithm implementation, including feature engineering and data augmentation methods used in data preprocessing; Using Experience Replay mechanism in reinforcement learning to solve the strong correlation problem of the data and the non-stationary distribution problems; Using noise method

together with Upper Confidence bound method, soft maximizing temperature method, etc, to solve the exploration problems in the learning process; As well as adjusting the learning rate dynamically according to the KL divergence and the method of evaluation in different stages.

Finally, this article also conducted a variety of experiments to evaluate the performance of AlphaGobangZero. First, conduct experiment on the optimal model, plot the change of loss during the training process, observe the related skill learned by AlphaGobangZero, and investigate the performance while playing with human gobang player. Then, several sets of comparative experiments were designed, include playing games using different models, playing games using models which are determined by different neural networks, and playing games using models with different parameters. The experiments of the optimal model show that, based on the current hardware resources, the trained AlphaGobangZero has roughly the same level as humans. Comparison experiments show that different neural networks and different parameters have a certain influence on the model performance. It is necessary to select suitable components and set reasonable parameters.

The work in this article has numbers of aspects that could be highly rated, but there are still a few aspects that can be improved, including the modeling part and code implementation part.

(1) Modeling Aspect

How to expand the algorithm to adapt to the constraint-based reinforcement learning model. For example, how to quickly expand a variety of rules in professional gobang.

How to improve algorithm for a complex, continuous state space or action space. The problems that currently addressed are discrete states or discrete action spaces which are relatively easy.

The Algorithm should be able to predict a tie game except for winning or losing situation.

(2) Code Implementation Aspect

How to speed up training while expanding the chess board. Currently the training is based on 8*8 board, we still need to expand the board to a larger one.

MCTS multi-thread search. Currently, only serial search is supported, which affects the operating speed. Introducing virtual losses to achieve MCTS multi- thread search could speed up the training process.

Improve GPU utilization. Currently the GPU is poorly utilized except in the process of optimizing the neural network. We need to improve the GPU utilization, as well as the code parallelism. For example, decouple the three core processes completely and executed in parallel.

Neural network continuous optimization problem in deep reinforcement learning algorithm. At present, the optimization process mainly draws from the continuous neural network in the DQN. And it is worth trying separating the iterative network from the target network to further solve problems of data correlation.

The loss of the value function converges too slowly, even with non-convergence problems. Experiments show that the loss of the value function converges too slowly. Although the Value Head used in this article is the same as that in the AlphaGoZero paper, this article only uses two residual blocks for training, while 19 or 39 residual blocks are used in AlphaGoZero, resulting in insufficient extraction of features in the residual layer. Lowering down the Value Head performance. We will consider improving the Value Head structure and observe the convergence of the value function.

References

[1] V. Allis. Searching for solutions is games and artificial intelligence. *Ph.d.thesis University of Limburg*, 25(6), 1994.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. pages 770–778, 2015.

[3] B. D. E. Knuth and R. W. Moore. An analysis of alpha beta pruning. In *Artificial Intelligence*, 1975.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, d. D. G. Van, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Appendix

A. Development Environment

(1) Program Language: Python3.6

(2) Deep Learning Framework: Pytorch 0.3.1

(3) Operation System: Mac / Linux

(4) Running Environment: Google Colab GPU Tesla K80

B. Division of Work

(1) Algorithm Survey: All members

(2) Algorithm Design and Implementation:

- Gobang Modeling: Wei Qin, Shizhi Jiang, Jiahao Wu, Ruipeng Su
- Reinforcement Learning Framework Design and Implementation: Taofeng Xue, Zhonghao Shi, Miao Li
- Class Diagram Design: Taofeng Xue, Miao Li, Xinyu Yang, Zhonghao Shi

- Neural Network Structure Design and Implementation: Zhonghao Shi, Shizhi Jiang, Taofeng Xue
- Monte Carlo Design and Implementation: Wei Qin, Jiahao Wu, Xinyu Yang, Ruipeng Su
- Algorithm Optimization: Ruipeng Su, Wei Qin, Xinyu Yang, Jiahao Wu

(3) UI Development and Implementation: Jiahao Wu, Taofeng Xue

(4) Experiment: Zhonghao Shi, Taofeng Xue, Shizhi Jiang, Miao Li.

(5)Document writing and translation: All members. Each person is responsible for their respective parts.

(6) LaTeX : Miao Li, Shizhi Jiang