上海科技大学
ShanghaiTech University

本科毕业论文（设计）

题　　目：基于完全自我学习的五子棋程序实现

学生姓名：林宇森

学　　号：57098385

入学年份：2014 年

所在学院：信息科学与技术学院

攻读专业：计算机科学

指导教师：虞晶怡 教授

二 零 一 八 年 五 月

上海科技大学
ShanghaiTech University

THESIS OF BACHELOR

Subject: The implement of a Five-in-row program based on self -play with a general reinforcement study learning algorithm

Student Name: Yusen Lin

Student ID No: 57098385

Date of Attendance: 2014

College: School of Information Science and Technology

Major: Computer Science

Advisor: Prof. Jingyi Yu

Date: 05/18/2018

摘要

棋类游戏是典型的完全信息博弈游戏，自从计算机出现以来，研究棋类游戏的 AI 就成了开发计算机“智能”的流行手段。从深蓝到 AlphaGo，AI 在每一种棋类中击败人类顶尖高手都被大众当作计算机智能发展的里程碑，都可以引发全世界的热烈讨论，会下棋的电脑也被当作人工智能的一大重要标志。本次毕业设计将分析谷歌训练 AlphaZero 的方法，接着将 AlphaZero 的训练方法运用到五子棋上，训练一个在较小棋盘上竞技水平出色的五子棋 AI，然后使用 python 完成游戏 UI。

关键词：五子棋，人工智能，蒙特卡洛树搜索，反馈学习

ABSTRACT

Board games are typical complete information games. Since the born of computer, researching AI for playing board games has been a popular method to develop the intelligence of computer. From Deep Blue to AlphaGo, victories toward top human players in different board games are always been recognized as milestones in the history of AI. “A computer who can play chess” is a profound impression when many people think of artificial intelligence.

This Graduation design will at first learn the method Google used to train AlphaZero. Then apply the AlphaZero method on Five-in-row game (also called Gomoku or Renju), to train a strong AI in relatively small board. Then I will use python to design a Game UI for playing with it.

Key words: Five-in-row(Gomoku), Artificial Intelligence, Monte-Carlo Tree Search, Reinforcement Study

Contents

Chapter I AI on playing board games

The study of computer chess is as old as computer science itself. Chess, as well as other board games, became the grand challenge task for AI researchers for many years. Most of powerful AI of board games are highly tuned to their own domain until recent three years, and they cannot be cultivated without human's knowledge on playing these games.

1.1 Traditional method: Alpha-Beta Search

The most widely used method to train these board games programs is adversarial search. Adversarial games are discrete and deterministic games with perfect information and usually have two players need to beat each other. Chess, Go and Five-in-row are included. Adversarial search is an algorithm to search for a best move for current player in a certain case. The quintessence of it is to maximize own value and minimize the opponent's value. So that it is also called Minimax search. The value of a phase is decided by human. For example, in some chess programs, different pieces will get a different value, for instance, the queen can have ten points, a soldier can have one point, the king has infinite points. And the sum of points of pieces are a kind of definition of value. And in some games like five-in-row, only the value of terminal states can be defined, and the value of middle states should be obtained by counting their subsequences' values.

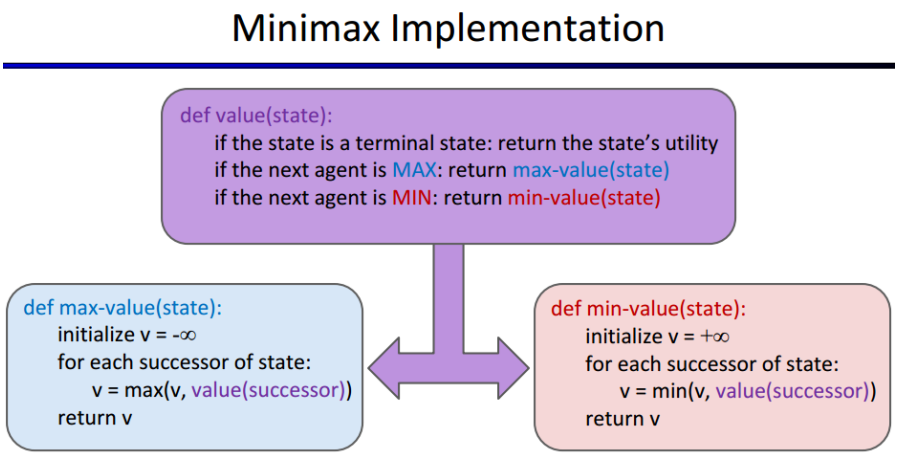


FIGURE 1.1 Minimax implementation

Theoretically speaking, minimax search can perfectly solve all adversarial games. But human cannot afford so much computer resource to achieve this goal. A typical chess game may include 40 steps, and in each step a player may have 40 possible choices on average, which results in an extremely huge number of possibilities which is impossible for modern computer to traverse. So most of simple programs can only take ten or less steps into account (That's why it is necessary to define values of middle states directly.), other programs may restrict possible choices per step with the help of human's prior knowledge.

Alpha-Beta pruning is always used in minimax search. The method should maintain two parameters: alpha and beta, alpha is the max's best option on path to root and beta is the min's best option on path to root. By comparing states' values to alpha and beta, some sub-trees can be deleted before being scanned, which obviously improves the efficiency of minimax search. So it can also be called Alpha-Beta search.

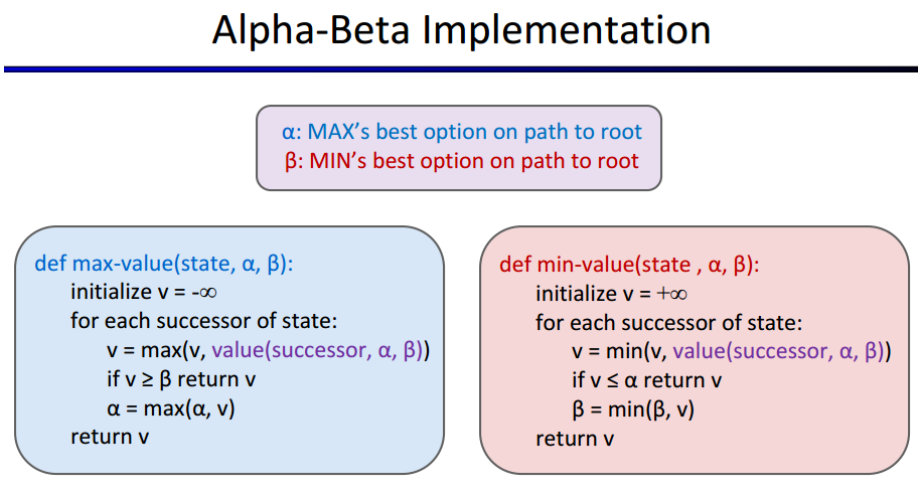


FIGURE 1.2 Minimax implementation with Alpha-Beta pruning

The most famous chess program Deep Blue is based on this method. It evaluates positions using features hand-crafted by human grandmasters and carefully tuned weights, combined with a high-performance alpha-beta search that expands a huge search tree using a large number of clever heuristics and domain-specific adaptions. Other strong chess programs, even include the world champion Stockfish on the 2016 Top Chess Engine Championship, are using very similar architectures.

1.2 Modern method: Monte-Carlo Tree Search

As mentioned above, there are two main problems of minimax search and all its improved models. First problem is that the number of possible steps and the depth of search tree are overwhelming. To solve the first problem researchers must restrict the depth of search tree per move, which results in the second problem: How to evaluate a state correctly.

Monte-Carlo Tree Search (MCTS) is an algorithm who can solve those two problems better. It can deliver an evaluation to a state, though not precise at first. According to its design, the search tree will continually concentrate at some "moves more deserve to search". If a good move is found, MCTS will simulate remain moves after this move quickly and get a simulated result. In a word, MCTS combines the benefit of Deep First

Search and Board First Search.

Monte-Carlo tree is a tree with nodes represent states. Descendants of a node mean states can be achieved after take one move based on current node. A node has such a label: A/B, B is the number of times the node is visited and A is the number of wins of player 1 in those visits. Take Five-in-row as the example, the root node will be the empty board labeled as 0/0 at first. Then repeat following steps to do MCTS:

First, Selection. Going down from the root node, choose the action that is the most deserve to do, until found a state not in the tree.

Second, Expansion. Add a descendent node on current node with label 0/0, and it represents the state has not been visited in the tree.

Third, Simulation. From the new node, use rollout policy to finish the game quickly and get a result: win or lose. The rollout policy at first is pure random policy, and will be better after many times updating. The policy should not be too time consuming since a weak policy with more simulations, usually, is better than a strong policy with less simulations.

Forth, Backpropagation. If a result is obtained after simulation that the black wins (player 1 wins), and the node it started to simulate is a black node (player 1 takes a move). Then, the node and all its black ancestors' labels will plus 1/1, and all its white ancestors' labels will plus 0/1.

In first step "the node who is deserve to visit" is mentioned about, now a score called Upper Confidence Tree(UCT) score should be defined for a certain node. A node with a higher score is more deserve to visit.

$$\text{Score} = x_{child} + C \cdot \sqrt{\frac{\ln(N_{parent})}{N_{child}}} \tag{1-1}$$

X is the winning rate of current node, for example X(Black node)=A/B, X(White node)=1-A/B.

C is a parameter, who can control the inclination of MCTS, a big C urges MCTS to expand new nodes, a small C urges MCTS to visit known best nodes.

N is the number of times of visit, N-parent equals to the sum of all N-child.

1.3 Modern tool: Convoluted Neural Networks

CNN is a powerful tool in the field ofsupervised learning, which is first raised by Geoffrey Hinton in 1980s. Although be look down on by the mainstream of machine learning scientists in the next twenty years, CNN proved to be a prospective image recognition method in 2012, as the method of training the champion model of ImageNet that year. Now with the help of the strong calculating ability of GPU, this method is become very popular and optimized.

The first motivation of using convoluted network is to restore the space relationship of input image. The aim of convolution is to extract hidden information from input images. Using different convolutional kernels, different types of information can be extracted. And such information will be highly abstract and people will find it hard to recognize the link between output images of convolution layers and the original input images. A typical CNN will include following steps.

First one is convolution. In convolutional layers, input tensors will be convoluted by convolutional kernels and get the output tensors. If the size of a kernel is 3×3, the value of a certain pixel in output tensors is the sum of the convolution of the kernel and a piece of pixels in input images, whose size is same as the kernel's and whose center position is same as the certain output pixel. There is a question that after convoluted by such a kernel, the size of tensor is become smaller. So in order to keep the size, some zero pixels can be added to the outside of borders of input images, and this is called same padding. A convolution layer can have many filters, each filter is a kernel with different parameters. If there are k filters in a layer and the size of an input tensor is n×n, then the output will be a k×n×n tensor. Although parameters of all filters are initialized as the same, after many times back propagation, they will be different and detect different features. Usually, there will be more than one convolution layers in a network, since some features are difficult to be detected by single-layer filters, instead, they can be detected by multi-layer filters.

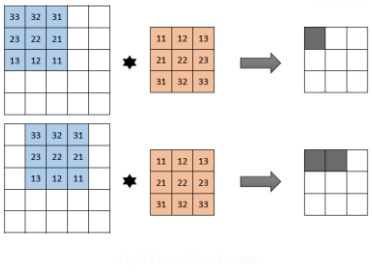


FIGURE 1.3 An example of convolution

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

FIGURE 1.4 An example of pooling

Second one is the activating function. If a little change in the structure of input tensor can lead to a totally different result, that will be awful. And the aim of using activating function is to add some non-linear factors on linear models. The process of convolution is linear, and linear models have some defects on representing features. Activating function should have some of following features: 1. Non-linear; 2. Continuously differentiable; 3. Monotonic; 4. Nearly linear near the base point. Sigmoid function and Tanh function are traditional activating functions. In CNN, Relu function

$$y = \max(0, x) \tag{1-2}$$

is the most popular activating function. When x>0, the gradient of this function will equals to 1, ensures the fast converging speed. And it also increases the sparsity of the network, let the remain non-trivial features be more representative.

The third is the Pooling layer. In order to avoid the curse of dimension, it is necessary to decrease the number of dimensions. If the output tensor of convolutional layers has a relatively large size, pooling is a way to restrict it. Choosing the largest pixel in an area to represents the whole area is a typical way of pooling. If the area is 2×2, the size of output tensor of pooling layer will be one fourth of the size of input tensor, while most of information are preserved. Also, choosing the average is another kind of way

of pooling. Sometimes, the pooling layer not only decreases the number of dimensions, but also prevents overfitting and improves the performance.

The fourth is the Fully-Connected layer. The process of previous three steps can be viewed as extracting features. And the function of FC layer is to use these extracted features to classify the input. The structure of FC layer is just similar to the convolutional layer, with the filter's size equals to the size of input tensor. Which means all pixels in the input tensor will be weighted and sum to one value. According to the output of FC layer, initial input tensor will be classified as one type.

Comparing the final output of FC layer and the label (the ground truth), people can calculate the loss. The loss function is usually MSE. Then the only thing left to do is using gradient descent or other methods to decrease the loss by updating all weights in all filters in all layers.

Labeled tensors are used as training set to train weights of CNN, then use the same way but treat the output of FC layer as prediction people can classify test tensors.

1.4 Modern tool: Reinforcement Study

In order to let the program finds better and better policy by itself, reinforcement study and DNN are used in AlphaZero.

A deep neural network

$$f_\theta(s) = [p, v] \tag{1-3}$$

with parameter θ, in which s represents input game states, f is a DNN function, and [p,v] are output. P is the predicted vector of conditional probability of each action given the state s. And v is an estimate value of the expected outcome from position s. The simulated final outcome z satisfies $v \approx E(z|s)$. Another vector π representing a probability distribution over moves should be defined. The value of vector π is calculated from the actual performance of simulated games during MCTS. The parameter of the network is totally produced based on self-play and self-optimization, so at first θ is initialized randomly.

The loss function is:

$$l = (v - z)^2 + \pi^T \ln(p) + c||\theta||^2 \quad (1\text{-}4)$$

The first term is the value loss, shows the distance between predicted state value and simulated state value. The second term is the policy loss, shows the distance between predicted probability vector and simulated probability vector. And c is a parameter controlling the level of L2 weight regularization. By using stochastic gradient descent with annealing learning rate, the parameter θ will be updated.

In AlphaGo Lee and AlphaGo Zero, some properties of Go like invariance to rotation and reflection are utilized, so that one piece of training data can be expanded into 8 pieces by rotating and flipping the board. In order to train all board games, all such specialized properties are dropped by AlphaZero.

Chapter II Apply AlphaZero algorithm on Five-in-row

2.1 The introduction to other Five-in-row programs

There are some reasons of choosing Five-in-row as an example. First, it is a very popular board game among eastern Asia countries. It is a traditional game in ancient China, as a game using same board and pieces as Go. Second, the rule of Five-in-row is very simple, a rookie can understand the rule in one minute and play it immediately. Third, unlike Chess and Shouji, a fixed size board is not necessarily needed to play Five-in-row, since the board is empty at first and there is not any constraint on the place of pieces. So sometimes people even play Five-in-row without defining the border. This property is important for AI training, let it be possible to train a powerful Five-in-row program in personal PC.

According to Google Deep Mind, the AlphaZero chess program who defeats Stockfish is trained in four hours. They used 5000 first-generation TPUs to generate self-play games and 64 second-generation TPUs to train the neural networks. The resource they use is extremely huge and if one wants to use single computer or server to train his own chess champion it will take more than one thousand years.

In 1992, Victor Allis has proved that if player 1 (the player who takes a move at first, usually controls black pieces) using a certain strategy, he will definitely win. The theory is work in 15×15 Five-in-row without restricted move or other rules for balance. And after few years, people have realized that even with some rules of restricted moves, player 1 can still definitely win, according to the proof by Janos Wagner in 2001. In formal competitions, more and more rules are set to balance the winning rate of two players. For example, player 1 should take two moves together in his third move, and player 2 can choose one of them as the third move of player 1. The Five-in-row game become more and more unpopular as a professional competing game. So in 2003, professor YiChen Wu invented a new game called connect 6. Player 1 moves first in

this game and then two players take two moves together alternatively. The player who let his six pieces in a row can win the game. Till now, no computer program can prove that either player 1 or player 2 can win or draw connect 6 definitely. It's a balance game and is way more difficult than Five-in-row. Actually, these "definitely win" theories of Five-in-row may not work in smaller board, if the size of board is only 5×5, it is very easy for both players to avoid losing.

2.2 The definition of the rule of Five-in-row

The size of Five-in-row board should be not big enough so that an obvious result can be observed in few days. The original size 19×19 or the competition size 15×15 are too big, in this experiment, size of 9×9 is enough.

The game state is a vector with 81 elements, each represents a place on the board. There are three possible values an element can have, 1 represents a black piece, -1 represents a white piece and 0 represents an empty place.

Two players controlling black and white pieces are included in a game. They take a move alternatively. Putting a piece on an empty place is the only valid move.

There are three kinds of terminal states of the game and the occasion of judging whether the game is over is after any player takes a move. First, if there exists five pieces with same color in a row (next to each other horizontally, vertically or diagonally), the current player wins the game. Second, if there is no empty place on the board, the result of the game is draw.

2.3 Details in implementation

Tensorflow in python is a popular machine-learning framework, with the help of which it is handful to build a deep neural network. The policy value net mentioned before should be implemented through Tensorflow.

2.3.1 The structure of the policy-value net

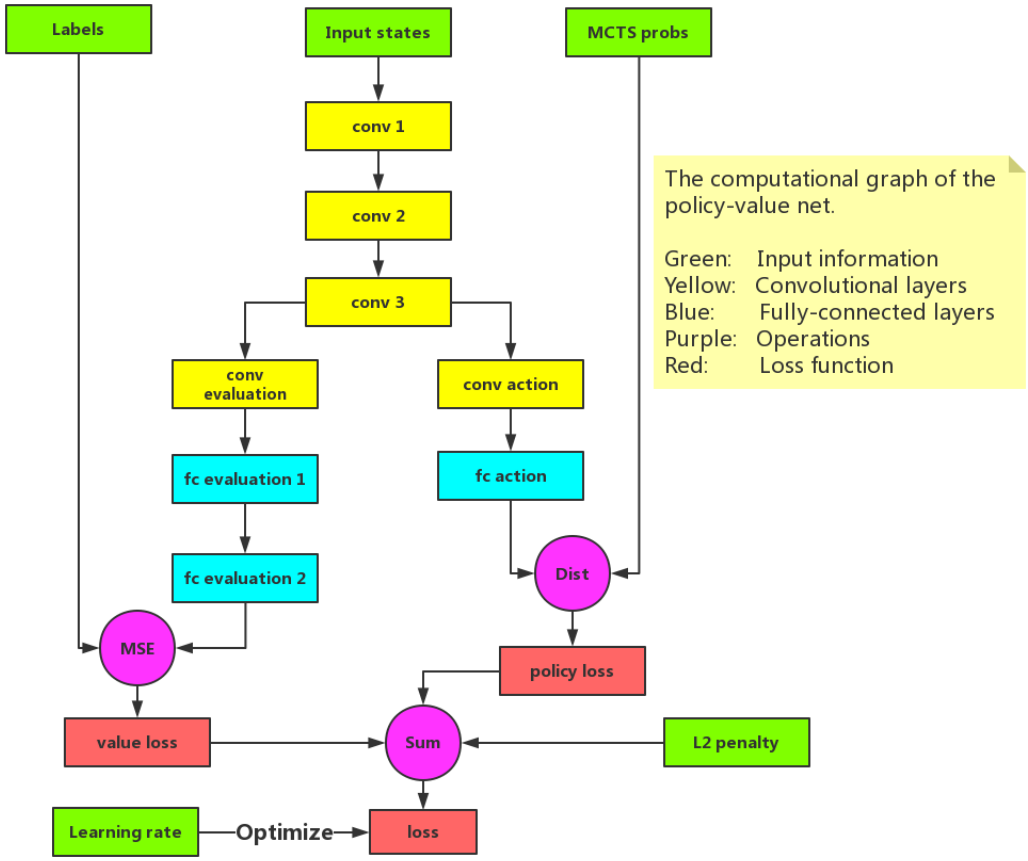


FIGURE 2.1 The structure of the policy-value net

Figure 2.1 is the computational graph of building a CNN as the policy-value net. This net can not only deliver a judge of the value of a state, but also suggest a choice of next action. There are five inputs in the net, in which learning rate and l2 penalty are static variable, while other three are important. As the paper mentioned above, a self-play game based on MCTS can produce a Monte Carlo tree. There are many states on a tree, each state has a label containing the information that whether the game wins or not.

These labels will be stored in an array and used as the input of the policy-value net. A state is a vector whose length equals to the multiply of board width and board height, and the variable space is [-1,0,1], representing three possible states of one place. These state vectors will be ranked the same as labels and used as the input of the policy-value net. What's more, in MCTS, each state has its own probability graph. The probability vector is as long as the state vector, each element in the vector corresponds to the probability of doing next action on this place. Probability vectors should be used as the input of the policy-value net, too.

Labels and mcts-probs can be viewed as ground truth, and what the net want is to train enough samples, and give a precise label and mcts-probs when getting a totally new state. So processing the input states is the main part of the computational graph.

Input states will go through three convolutional layers. First layer has 32 filters with kernel size 3×3.It is used to detect facial and simple features of board states. The padding way is same padding. So the size of output data will be 32 times of the size of input data. And the activating function is Relu function, whose benefits have been introduced in the previous section. The output of first convolutional layer will be the input of the second convolutional layer. The second and third convolutional layers are similar to the first layer, they only have difference on the number of filters. The second layer has 64 filters and the third layer has 128 filters. Because deeper layers should detect more complex features, and the number of types of complex features is far bigger than that of simple features, so the number of filters in deeper layers is increasing. After going though these three layers, the size of data will be very large.

Since the net combines the processing of two kinds of loss, the state information should be divided into two parts. One for calculating value loss and another for calculating policy loss. The output of conv3, will pass through two sub-tree, as showed in the graph. The evaluation convolutional layer has 4 filters with kernel size 1×1. The padding way is also same padding. The action convolutional layer has $4 \times \text{board\_height} \times \text{board\_width}$ filters and other settings are same as the action layer. After this division,

the process of convolution is finished.

These two output data of the tree of convolutional layers will then be input of fully-connected layers. They should be reshaped to two dimensional tensors at first.

There are two evaluation fully-connected layer. First layer has 64 units and second layer has 1 unit. Since the output is a single value, so it is obvious that the second layer only has one unit. The reason why two layers are needed is that the size of data is large, and two layers adapt different activating functions. The first layer adapts Relu function, so that it can quickly squeezes the important parameters from the huge input tensor. The second layer adapts tanh function

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2-1}$$

since the output value can be a negative value. It takes a 64-length tensor as input and deliver an evaluation value over the input state.

There is one action fully-connected layer. With the number of units equals to the board size. Because the output of the policy layer is a prediction of the probability of next action. To make sure that the sum of probability equals to one, softmax function is a good choice as the activating function. Softmax function is a function to convert different values into the relationship of probability. An element V_i in vector V has softmax value

$$S_i = \frac{e^{V_i}}{\sum_j e^{V_j}} \ . \tag{2-2}$$

Figure 2.2 is an example of softmax function.

Until this step, the whole system can deliver an evaluation value and an action-probability vector. These two data are unreliable at first, so the distance between the net-born labels and the ground truth labels should be calculated and minimized.

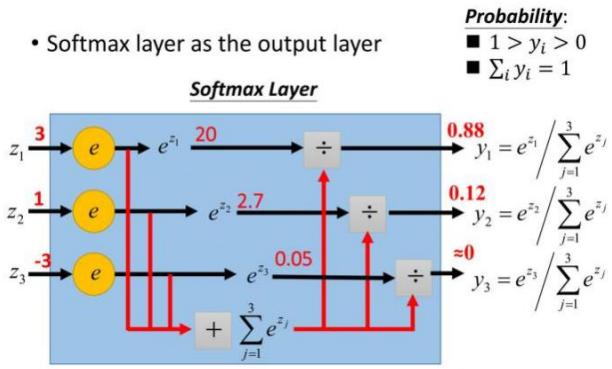


FIGURE 2.2 An example of softmax function

The loss function including three parts, as mentioned before. The first part is the value loss, which is the mean square error of the output value of evaluation net and the ground truth value in labels. The second part is the policy loss, which is the mean of the sum of the multiply of two probability vectors. Since the multiply of two vectors can represent the distance between them. The third part is L2 penalty. L2 regularization is a method to prevent overfitting. And it is produced from the variance of all trainable variables in the net. The sum of these three parts is the loss, and what remains to do is to minimize the loss.

Gradient descent is a common way of minimizing loss function. Considered of the huge size of variables, stochastic gradient descent should be a better method. The learning rate is not a constant, it adjusts according to the relationship between new probability vectors and old probability vectors. The current model will be saved when changed.

2.3.2 The training pipeline

The training pipeline includes three steps: 1. Self-play, 2. Training, 3. Update. Showed in Figure 2.3.

In the first step, two players play with each other according to MCTS, and their playing data will be collected as the input of the policy-value net. One MCTS player simulates 400 times at each action, and its policy is the policy trained by the policy-value net. The player's opponent is the pure MCTS player, which simulates 1000 times at each

action at first, and its policy is the rollout policy. They play with each other, using Monte-Carlo Tree search as mentioned before. When they finish one game, all states in the tree and their corresponding labels and probability vectors will be recorded.

In the second step, the recorded data will be sent to the policy-value net to minimize the loss function and optimize the policy. Considering of the rotational invariance and reversal invariance of Five-in-row, one state can be expanded to eight states by rotating and flipping. Labels and probability vectors corresponding to states also do same actions along with states. This method is used in AlphaGo zero since Go has these properties, too.

The third step will start every 100 self-play games finish. A tournament with ten games between the trained MCTS player and the pure MCTS player is executed. If the winning rate of the trained MCTS player is higher than the best winning rate in the history. The best policy will be updated to the current policy. If the winning rate is 1, which means that the trained MCTS player is totally overwhelming the pure MCTS player, the pure MCTS player will upgrade by adding 1000 simulations at each action. And the best winning rate will be also reset to zero.

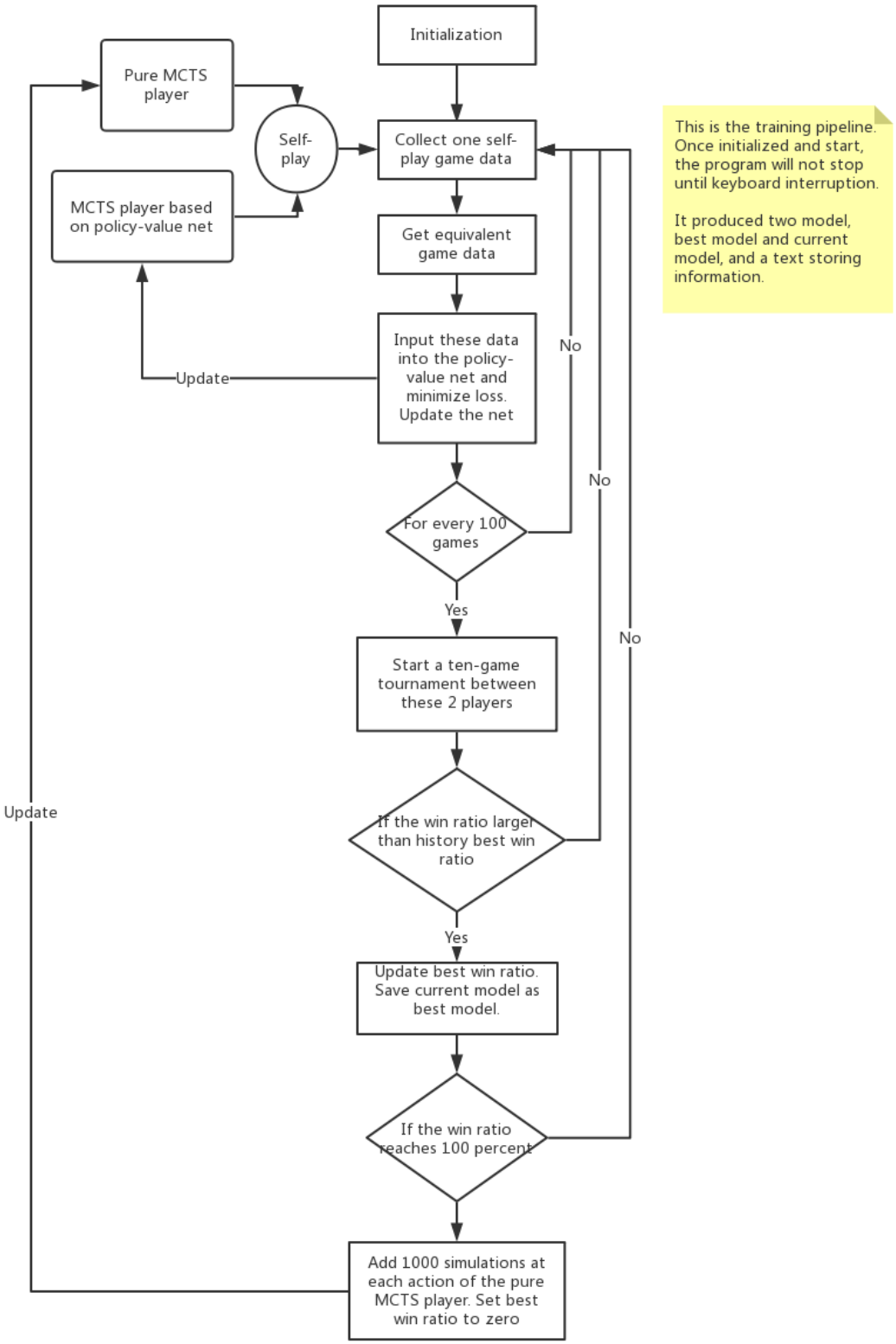


FIGURE 2.3 The flowchart of the training pipeline

Chapter III Results analysis and Game implementing

3.1 The performance of this program

Figure 3.1 shows the loss and the entropy decreases when the number of self-play games increases. These 4000 9×9-size games took about two days to simulate on my personal computer. The loss and entropy decreases nearly a half after the training. Figure 3.2, 3.3, 3.4 are loss-entropy graphs with smaller board: 8×8, 7×7, 6×6. Compared to the case in bigger boards, loss and entropy decreases faster in smaller boards. Because there are fewer choices among MCTS in smaller boards so the learning process will be easier. In the smallest board, it is obvious to observe many ties in self-playing, which means the performance of the program is very close to the optimal player. But in the 9×9 board, the performance of the program is still not strong since I can beat it. It takes two or three days for the 9×9 program to reach this level and then after even ten days it cannot beat its pure MCTS opponent with a higher winning rate so it cannot update the best policy.

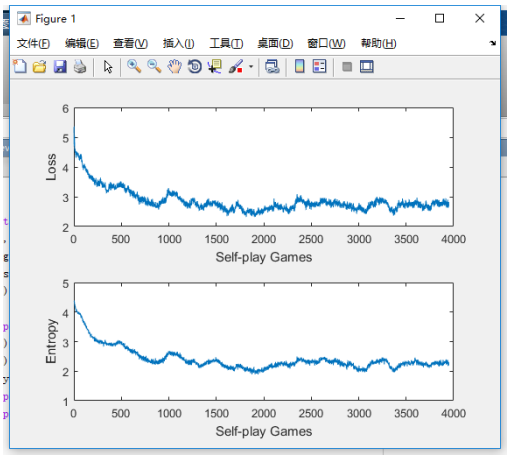


FIGURE 3.1

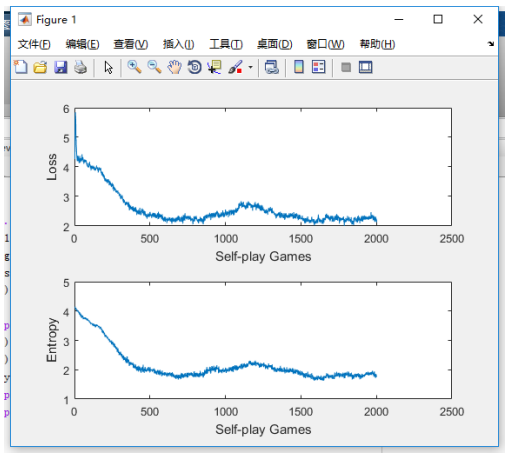
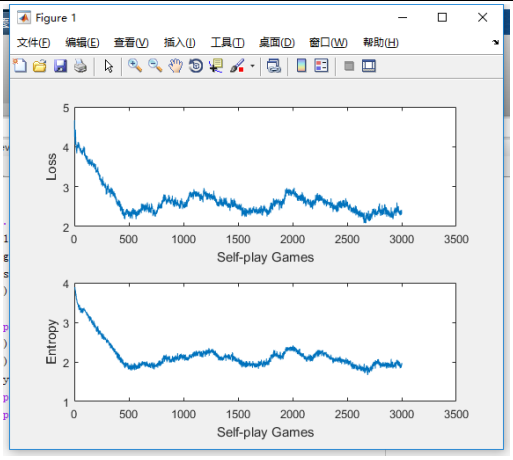


FIGURE 3.2



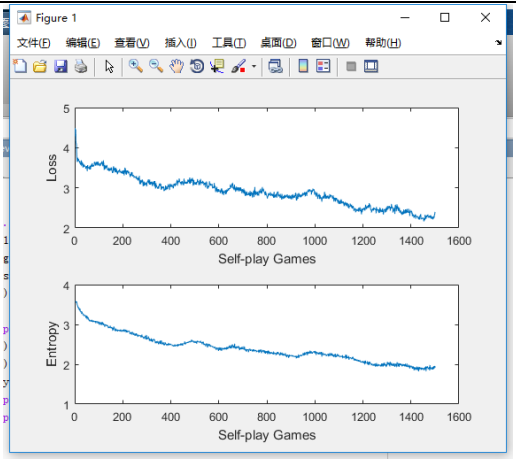


FIGURE 3.3 **FIGURE 3.4**

FIGURE The games-loss and games-entropy graph of four kinds of boards

Here attached some sample games between human player and the program. Only relatively bigger board are chosen since five-in-row is really boring in too small board. Each tuple represents the position of moves.

According to my experience of playing with the program, the 9×9 size board AI is still a little weak and I can beat it easily use white piece. The 8×8 size board AI is stronger, sometimes it can beat me and sometimes there will be a tie.

BLACK: Computer, White: Human, Board Size: 9×9

1.(5, 5)(6, 6) 2.(6, 5)(4, 5) 3.(7, 5)(8, 5) 4.(3, 5)(5, 3) 5.(7, 3)(3, 6) 6.(7, 4)(7, 6)

7.(7, 2)(7, 1) 8.(3, 3)(3, 4) 9.(3, 7)(4, 6) 10.(5, 6)(4, 7) 11.(5, 7)(4, 4) 12.(5, 9)(4, 8)

WHITE WINS

BLACK：Computer, White: Human, Board Size: 8×8

1.(5, 4)(4, 5) 2.(5, 5)(5, 6) 3.(3, 4)(4, 7) 4.(4, 4)(2, 4) 5.(6, 4)(7, 4) 6.(6, 5)(7, 6)

7.(4, 6)(7, 3) 8.(7, 5)(3, 3) 9.(4, 2)(5, 3) 10.(6, 3)(6, 2) 11.(6, 6)(6, 7) 12.(4, 3)(2, 5)

13.(7, 7)(8, 8) 14.(3, 2)(2, 1) 15.(2, 2)(3, 6) 16.(5, 2)(1, 2) 17.(5, 7)(1, 4) 18.(5, 8)(2, 7)

19.(8, 4)(4, 8) 20.(2, 8)(2, 6) 21.(3, 7)

BLACK WINS

BLACK：Human, White: Computer, Board Size: 8×8

1.(4, 4)(5, 5) 2.(5, 4)(6, 4) 3.(4, 3)(4, 6) 4.(7, 3)(3, 7) 5.(2, 8)(6, 6) 6.(2, 3)(6, 5)

7.(6, 3)(6, 7) 8.(6, 8)(5, 3) 9.(4, 5)(7, 2) 10.(2, 5)(3, 6) 11.(2, 6)(2, 7) 12.(3, 4)(1, 4)

13. (5, 1)(5, 7) 14.(1, 6)

BLACK WINS

3.2 Use pygame to implement a python game

Pygame is a library of python, can do graphics and make a game. Figure 3.2 is the flow chart of the game program. Players can play with the computer or play with each other, as what they can do in any online Five-in-row games. Since there are different types of AI being trained, player can define whether the size of board is 6×6, 7×7, 8×8 or 9×9. During the game, the program fetches mouse click signal as the player's action. And it will deliver an AI's move automatically.

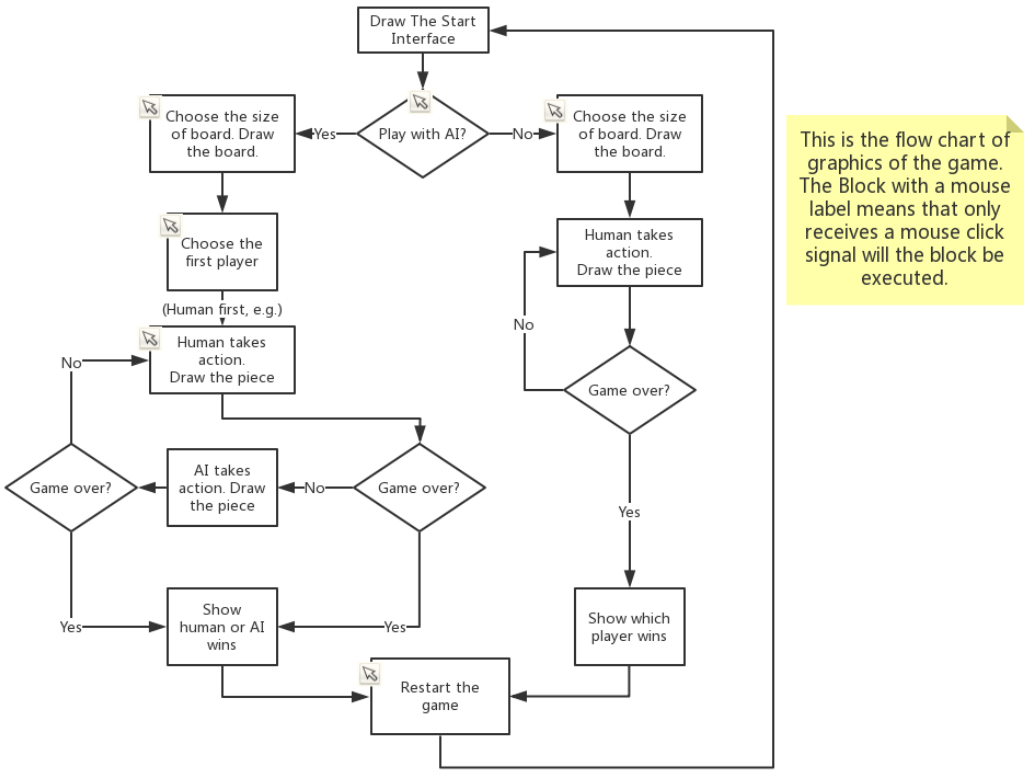


FIGURE 3.5 The flowchart of the game implementation

Figure 3.6 to 3.9 are all screenshots of the game program.

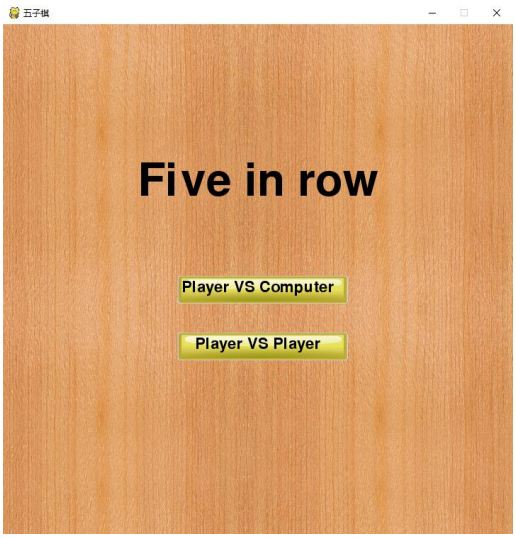


FIGURE 3.6

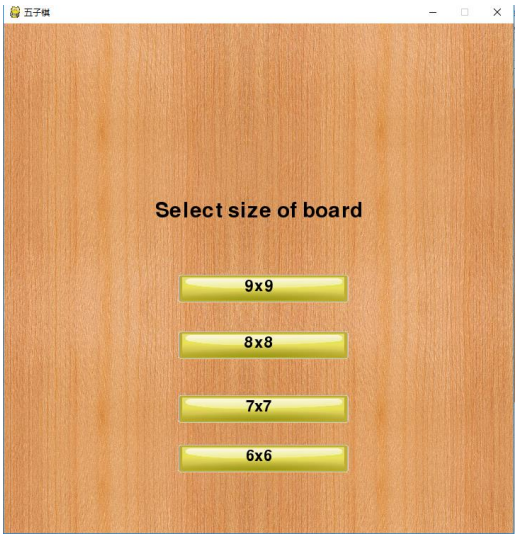


FIGURE 3.7

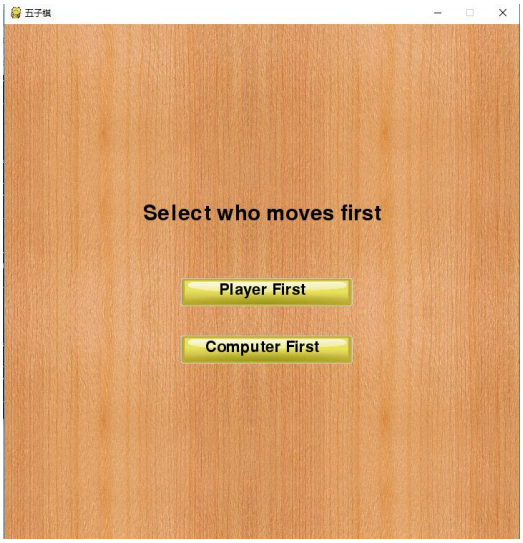


FIGURE 3.8

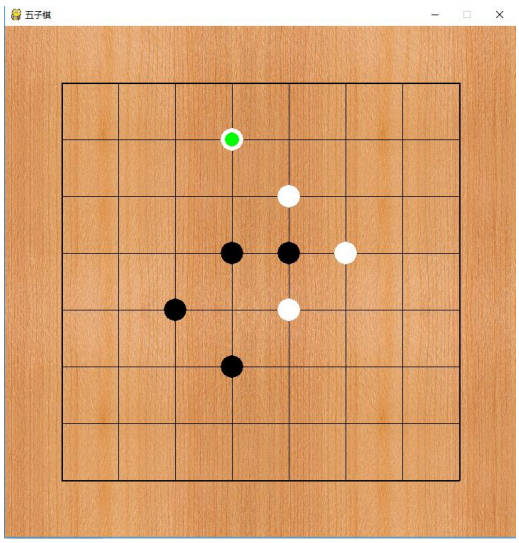


FIGURE 3.9

FIGURE Screenshots of the game program

Conclusion

This is an experiment that implements a method of training board game program. The process of training is completely based on self-plays and is unrelated to any human knowledge of the game. Using Monte Carlo Tree Search to keep self-playing and neural network to train the policy, the program will upgrade. The ability of playing well is hided in parameters of the policy value network, who will deliver a value judging the state and a probability graph of next move when given a state as input. Five-in-row is a simple game to show the feasibility of this method. The only flaw, compared to traditional board game programs, is that this AI takes too much computational resource to train and the thinking time of each step is also relatively longer.

Future works can be concluded in two ways. First, some optimization method can be applied to improve the efficiency of the training process. For example, some pruning rule can be used to avoid over-fitting. Second, the method can be expanded to other games, especially incomplete information games, like poker. That will be very interesting.

REFERENCE

[1] David Silver, Thomas Hubert, Julian Schrittwieser, et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs.AI]

[2] David Silver, Julian Schrittwieser, el al. Mastering the game of Go without human knowledge[J]. Nature 550, 352-359(19 October 2017)

[3] David Silver, Aja Huang, el al. Mastering the game of Go with deep neural networks and tree search[J]. Nature 529, 484-489(28 January 2016)

[4] Omid E David, Nathan S Netanyahu, and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess[C]. In International Conference on Artificial Neural Networks, pages 88–96. Springer, 2016.

[5] J. Veness, D. Silver, A. Blair, and W. Uther. Bootstrapping from game tree search[J]. Advances in Neural Information Processing Systems. pages 1937–1945, 2009.

[6] Victor Allis. Searching for Solutions in Games and Artificial Intelligence[M]. Ph.D. Thesis, University of Limburg. Maastricht, The Netherlands. ISBN 90-9007488-0. 1994.

[7] Janos Wagner, Istvan Virag. Solving Renju[J]. ICGA Journal. March 2001

[8] I-Chen Wu, Dei-Yen Huang, Hsiu-Chen Chang. CONNECT6[J]. ICGA Journal. December 2005

[9] Sakata, G. and Ikawa, W. Five-In-A-Row, Renju. The Ishi Press, Inc. Tokyo, Japan .1981.

[10] O. Patashnik, Qubic: 4x4x4 Tic-Tac-Toe, Mathematics Magazine 53,202-216. 1980.

[11] Kevin P. Murphy. Machine Learning, A probabilistic Perspective[M]. The MIT Press, Cambridge, Massachusetts, London, England

[12] AlphaZero 实战：从零学下五子棋

https://zhuanlan.zhihu.com/p/32089487

[13] 那么蒙特卡洛树搜索(Monte Calro Tree Search, MCTS)究竟是啥

https://blog.csdn.net/natsu1211/article/details/50986810

[14] 28 天自制你的 AlphaGo（五）：蒙特卡洛树搜索（MCTS）基础
https://www.leiphone.com/news/201702/poAxdPGhfQFrxsXS.html?vt=4

[15] Visualization of Convoluted Nearual Networks

scs.ryerson.ca/~aharley/vis/conv/

[16] Al Sweigart. Making game with python and pygame[M].

Acknowledgement

First of all, I would like to extend my sincere gratitude to my supervisor, professor Jingyi Yu, for his instructive advice and useful suggestions on my research. He delivered me many opportunities to be responsible for some projects, which cultivated my skills in using different tools to solve different problems. What's more, his recommendation letter played a vital important role in my future education.

High tribute shall be paid to Wenguang Ma and Shiying Li. They spent much time helping me correcting flaws and errors in this article. And their advice on how to be a good researcher is important to me.

I am also indebted to professor Shenghua Gao, who led me to the wonderful world of computer vision, gave me suggestions on how to be a good lecturer and wrote a recommendation letter for me.

Special thanks should go to my friends who put considerable time and effort into their comments on the draft, and my parents for their continuous support and encouragement.