An AI for a Modification of Dou Di Zhu

CS298 Report

Presented to

Professor Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 298

By

Xuesong Luo

April 2020

SAN JOSÉ STATE UNIVERSITY

The Designated Project Committee Approves the Master's Project Titled

An AI for a Modification of Dou Di Zhu

By

Xuesong Luo

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett	Department of Computer Science
Dr. Mark Stamp	Department of Computer Science
Dr. Fabio Di Troia	Department of Computer Science

ACKNOWLEDGEMENT

I would like to express my sincere gratitude and appreciation to Dr. Chris Pollett for his guidance and support throughout the entire project. It was my honor to have him as my advisor.

I also want to extend my thanks to the committee members, Dr. Mark Stamp and Dr. Fabio Di Troia, for their suggestions and time.

Finally, I am grateful to my friends and family for always being a strong support for me throughout my career.

ABSTRACT

We describe our implementation of AIs for the Chinese game Dou Di Zhu. Dou Di Zhu is a three-player game played with a standard 52 card deck together with two jokers. One player acts as a landlord and has the advantage of receiving three extra cards, the other two players play as peasants. We designed and implemented a Deep Q-learning Neural Network (DQN) agent to play the Dou Di Zhu. At the same time, we also designed and made a pure Q-learning based agent as well as a Zhou rule-based agent to compare with our main agent. We show the DQN model has a 10% higher win rate than the Q-learning model and Zhou rule-based model when playing as the landlord, and a 5% higher win rate than the other models when playing as a peasant.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

AI has provided a variety of possible applications and has been widely entered into people's lives, learning and work, even people do not realize how many times they use it every day. Many people's lives are closely connected with Siri and Cortana. Just a few years before, AlphaGo successfully defeated human world champion Li Shishi in the Go game. As people know that Go is the most complex board game in the world, and its complexity was proved to be PSPACE-hard by Robertson and Munro in 1978 [12]. Although Go is the most complex board game, this game is fair for the players. The card games are different, and most cards game refers to the gambling. Playing cards is a famous pastime that people from all corners of the world enjoy, but how did it all get started? The first playing cards appeared in the 9th century during Tang Dynasty China [1]. The first reference to the card game in world history dates no later than the 9th century, when the Collection of Miscellanea at Duyang, written by Tang Dynasty writer Su E, described Princess Tongchang (daughter of Emperor Yizong of Tang) playing the "leaf game" with members of the Wei clan (the family of the princess' husband) in 868 [2][3]. Playing cards first appeared in Europe in the last quarter of the 14th century [4].

Nowadays, Dou Di Zhu is a famous card game in China. In the past, it was just a provincial game in China, originating in the Huangshan District and Anhui. However, just a few years ago, there were almost 1 million concurrent Dou Di Zhu players on the Tencent QQ game platform alone [5]. Except for Mahjong, the Chinese favorite card game is Dou Di Zhu.

In China, many people like playing Dou Di Zhu online, but not many researches focus on AI playing Dou Di Zhu. It does not like a board game: Go. Few research papers focus on the AI for playing Dou Di Zhu, like the Rule-Based, Decision Tree, and Q-Learning algorithm. Renzhi Wu, Shuai Liu, Shuqin Li, and Meng Ding [9] designed a Rule-Based model for playing Dou Di Zhu, and their model set many different rules to simulate the human behaviors for playing Dou Di Zhu. Their rule-based model can be easy to beat two Standard AIs (Pos. West/A and East/C) in Dou Di Zhu. Zhennan Yan, Xiang Yu, Tinglin Liu, and Xiaoye Han [10] designed a Decision Tree model. This model is similar to the rule-based model, by picking the best action beyond the previous actions, their model has more than 50% winning rate VS human beings.

In this project, we designed a Deep Q-learning Neural Network (DQN) for playing Dou Di Zhu. At the same time, we also compared the DQN model with the Q-learning

model and the Zhou rule-based model in the same environment to prove that the DQN model works better on Dou Di Zhu than the Q-learning model or rule-based model.

The report is organized in five chapters. The first chapter briefly introduces the main problems to be studied and related work in this project. The second chapter discusses the background information need to understand and related information need to know. The third chapter covers all the details of the design and implantation in the project. The fourth chapter talks about multiple experiments and tests of the project. Finally, the last chapter concludes the project results and the future works we can do.

CHAPTER 2

BACKGROUND

In this project, our purpose is to design a DQN model for playing the Dou Di Zhu. We need to know the game rules of the Dou Di Zhu. At the same time, in order to better understand DQN, we need first to understand what is Q-learning algorithm and Neural Network.

2.1 What is Dou Di Zhu?

Dou Di Zhu is a three players card game with a 54-card deck. There are two sides: one player will be the landlord, and two other players will be the peasants. The peasants are allies, and they need to compete with the landlord. During the game, some information on both sides is open to each player, like: the three landlord cards will show to every player; every player knows who the landlord is; and standard game includes dealing, biding, and playing.

2.1.1 Dealing

At the beginning of a game, 17 cards are dealt to each of the three players. The three remaining cards are given to the landlord after the landlord is selected.

2.1.2 Bidding Rule

After 51 cards have been dealt, players bid to become the landlord (in some variations of Dou Di Zhu, the landlord is selected by rolling dice). The dealer first says whether or not they want to become the landlord. If not, the person to his right can choose whether or not they are the landlord. Finally, if both pass, then the remaining player becomes the landlord. Suppose that the player who has the priority to be the landlord renounces the right to be a landlord, and that either of the other two players is eligible to become a new landlord. As for who may become the new landlord, it depends on the three players' standard default rules, such as, to decide by re-rolling the dice, or to decide by other rules, and so on. If all three players give up their eligibility to become a landlord, they can decide to reissue the cards for the next round of the game. Once a landlord, the player is entitled to the remaining three landlord cards, and should play card first at each game. The other two players become peasants and allies. As the game continues, the cooperation and competition between the three players are constantly changing, and their gains and losses are changing based on when someone decides to become a landlord.

2.1.3 Playing Order

In Dou Di Zhu, the landlord will play cards first, and then the two peasants will play their cards by order.

For example, there are A, B, and C three players, and assume A is the landlord. The play order is as follows:

A → B → C → A → B → C → A → B → C → ……

or

A → C → B → A → C → B → A → C → B → ……

Until a player finishes playing all his hands. This game round is over, and the winner will be the player and his teammate (the landlord does not have a teammate). For example, if B or C (peasant) is the first player to play all hand cards, both B and C win this round of the game. If A is the first player to be the landlord and finishes playing cards first, only A win this round of the game.

2.1.4 Playing Cards Rule

As players, how to play the card at themselves turn? In Dou Di Zhu, there are many different card combination: (shown in Table 2-1)

Table 2-1 The description of suit patterns

Card combination	Description
Rocket	Same as the Joker Bomb, both jokers (Red and Black), is the highest Bomb.
Bomb	Four cards with the same points. (e.g. AAAA)
Single	One single card. (e.g. A)

Pair	Two cards with the same points. (e.g. AA)
Triplet	Three cards with the same points. (e.g. AAA)
Triplet with an attached card/pair	Triplet with an attached card/pair. (e.g. AAA+B or AAA+BB)
Single Sequence	Five of more Singles in sequence excluding two and Jokers. (E.g. ABCDE or ABCDE...)
Double Sequence	Three of more pairs in sequence, excluding two and Jokers. (E.g. AABBCC or AABBCC...)
Pass	Choose not to play a card this turn. It is also called as a trivial pattern.

Players should follow the rules to play the different card combination:

1) The single card combination is ranked by:

   Red Joker, Black Joker, 2, A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3

2) Rocket is the highest rank card.

3) Bomb is ranked lower than Rocket, but higher than any other card combination. If you want to compare the rankings of two bombs here, you should rank based on the combination of single cards.

4) Except for Rocket and Bomb, comparison can only be performed within the same card combination based on the single card combination rank.

At the beginning, the landlord should play cards first, and he could play any combination of cards he likes. The next player should play the same combination of cards, and his card combination rank is higher than the landlord's card combination rank or use Rocket or Bomb to beat it.

Assume that just now, the last played cards dealt were Pair. The current player needs to play Pair cards (higher rank than last played cards of a Pair), or he also can use Rocket or Bomb to beat it. If the last played cards' owner is the same player as the current player, the current player can play any kind of card combination he likes to continue the game.

2.2 What is Q-Learning Algorithm?

Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action should be taken under what situations [6]. Q-learning learning is good at training the model on playing various games. People like to use the Q-learning algorithm to play video games, card games, or board games.

2.2.1 Model-Free Learning Algorithm

The term, Model-Free, means that the Q-learning algorithm is a kind of model, regardless of the environment, without making any modification, it can handle the rewards and transitions.

2.2.2 Reinforcement Learning Algorithm

The term, Reinforcement, means that the Q-learning algorithm model differs from the supervised learning model in not needing labeled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge) [8]. Reinforcement learning usually includes the two entities: agent and environment. The interaction between the two entities is as follow:

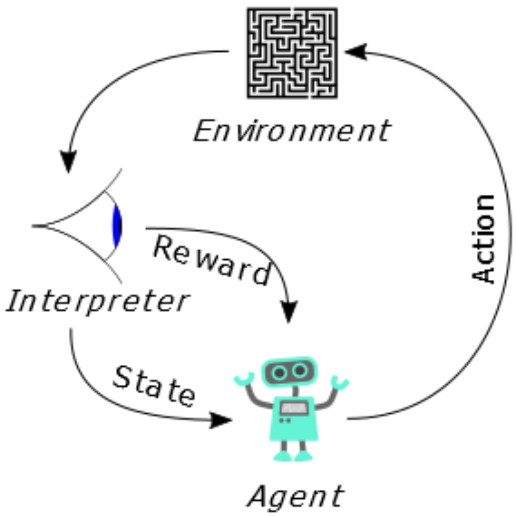

Figure 2-1 The interaction between Agent and Environment.

Figure 2-1 uses the relation graph to show us the interaction between the agent and the environment. An agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent. The problems applicable to reinforcement learning usually have the following characteristics:

1) Different actions cause different rewards

2) Reward is delayed

3) The reward for an action is based on the current state

2.2.3 Q-Learning

Q-learning is one kind of reinforcement learning algorithm. The kernel of the Q-learning algorithm is Q-Table. The rows and columns of Q-Table represent the values of state and action, respectively. The value of Q-table *Q (s, a)* measures the current state *s* is good or bad to take action *a*. Figure 2-2 shows us the interaction between the state, action, and Q-Table to get Q-value (reward).

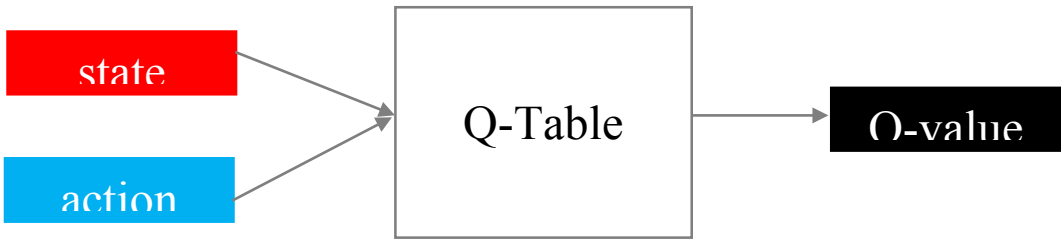

Figure 2-2 The interaction between the state, action, and Q-value

Let us probe into more detail about the Q-Table. At the beginning, the Q-Table is empty. If people hope to have a better Q-Table, they need to learn every value into the Q-Table to satisfy all situations in the environment.

Figure 2-3 shows us how to use the Q-learning algorithm to update the last Q-Table (good Q-Table).

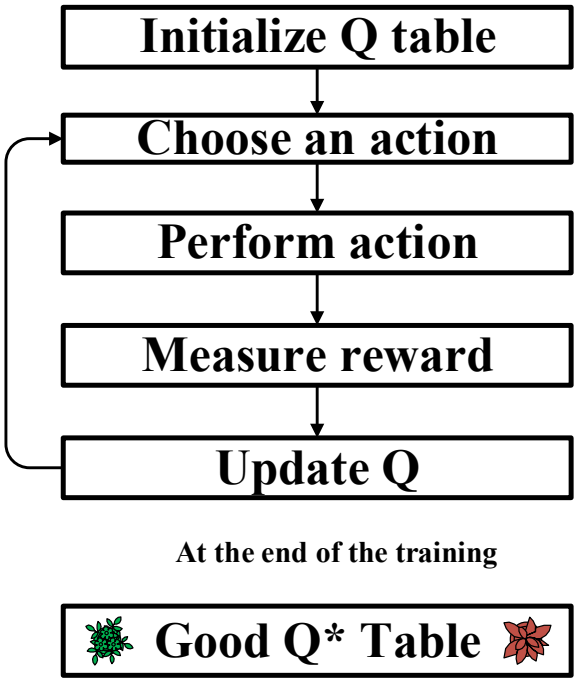

Figure 2-3. Q-learning Algorithm flow

First, we need to initialize an empty Q-table with all Q-value(reward) is 0. Then choose an action in the current state based on the current Q-value (choose the best reward action)

in the Q-Table. Furthermore, take the action and figure out the new state and new reward. Then through new rewards to update the Q-Table. The Q-learning pseudocode is shown in Table 2-2.

1) Initialize Q-values $(Q(s,a))$ arbitrarily for all state-action pairs.
2) For life or until learning is stopped...
3) Choose an action (a) and in the current world state (s) based on current Q-value estimates $(Q(s,\cdot))$.
4) Take the action (a) and observe the outcome state $\left(s'\right)$ and reward (r)
5) Update $Q(s,a) := Q(s,a) + \alpha\,[\,r + max_{a'} Q\left(s',a'\right) - Q(s,a)]$

Table 2-2 Q-learning pseudocode

2.3 What are Neural Networks?

A neural network is an interconnected assembly of simple processing elements, *units,* or *nodes,* is a computing system inspired by the biological neural network that constitutes the brain of animals [7]. The processing ability of the network is stored in the interunit connection strengths, or *weights,* obtained by the process of adaptation to, or *learning* from, a set of training patterns.

2.3.1 Perceptron

Historically, scientists have always wanted to simulate the human brain and create machines that can think. Why can people think? Scientists discovered that the reason is based on the human neural network. Figure 2-4 shows us the human brain's neuron.

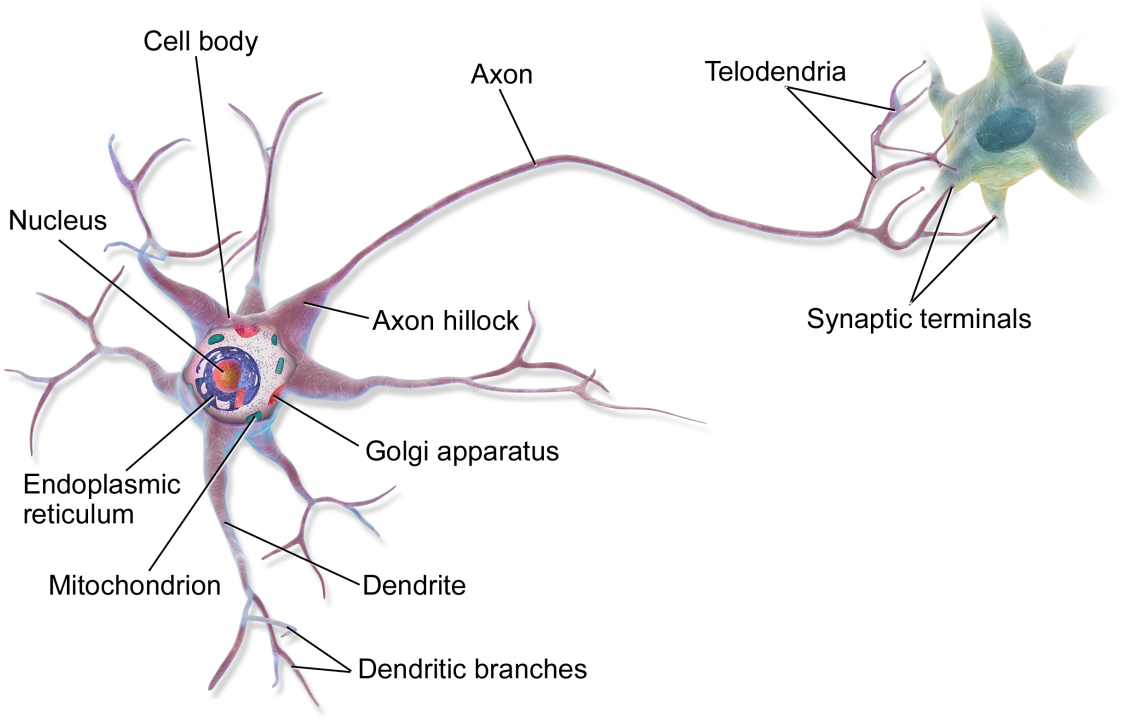

Figure 2-4 The human brain's neuron

External stimuli are converted into electrical signals and transfer to the human' neurons. Numerous neurons constitute the nerve center. The nerve center synthesizes various signals to make judgments. At last, the human body responds to external stimuli

based on instructions from the nerve center. Since the basis of thinking is neurons, if "artificial neurons" can be formed, artificial neural networks can be formed to simulate thinking. In the 1960s, the earliest "artificial neuron" model, called the "perceptron", was still used today.

In Figure 2-5, the circle represents a basic perceptron. It accepts multiple inputs (x_1, x_2, x_3...), and one output. Just like human' neurons feeling the changes of various external environments and finally generating electrical signals.

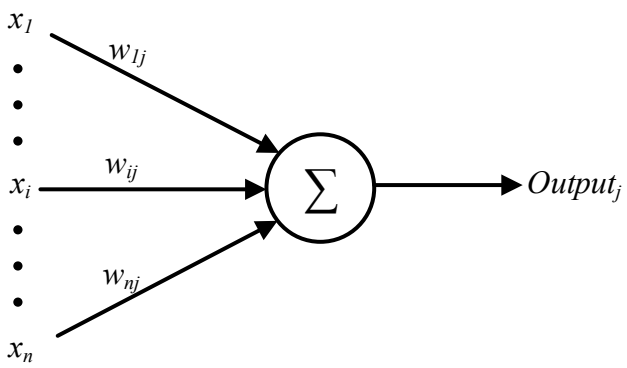

Figure 2-5 Basic perceptron

2.3.2 Weights and Thresholds

At most times, we have multiple inputs to represent multiple influencing factors. Moreover, some factors are decisive, and others are secondary factors. Therefore, weights can be assigned to these factors to represent their importance levels.

With different situations, we input different factors. Based on their weight, we can the different temp results. At this moment, we need a threshold to judge the temp result. If the temp result is greater than the threshold, the perceptron outputs 1; otherwise, it outputs 0.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \quad (2\text{-}1)$$

Formula 2-1 Weight and Threshold

In Formula 2-1, x is the input factor, w is the weight.

If we can make some modifications base on math, and change it into Formula 2-2.

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \quad (2\text{-}2)$$

Formula 2-2 Mathematical Expression of Perceptron

In Formula 2-2, x is the input factor, w is the weight, and b is the threshold.

When we use neural networks, we always need to build a complex network containing many neurons. Figure 2-6 shows us an example of Artificial Neural Networks (ANN). There are three input factors for the input layer, and there are two hidden layers, and each hidden layer has four perceptron and only one output in the output layer.

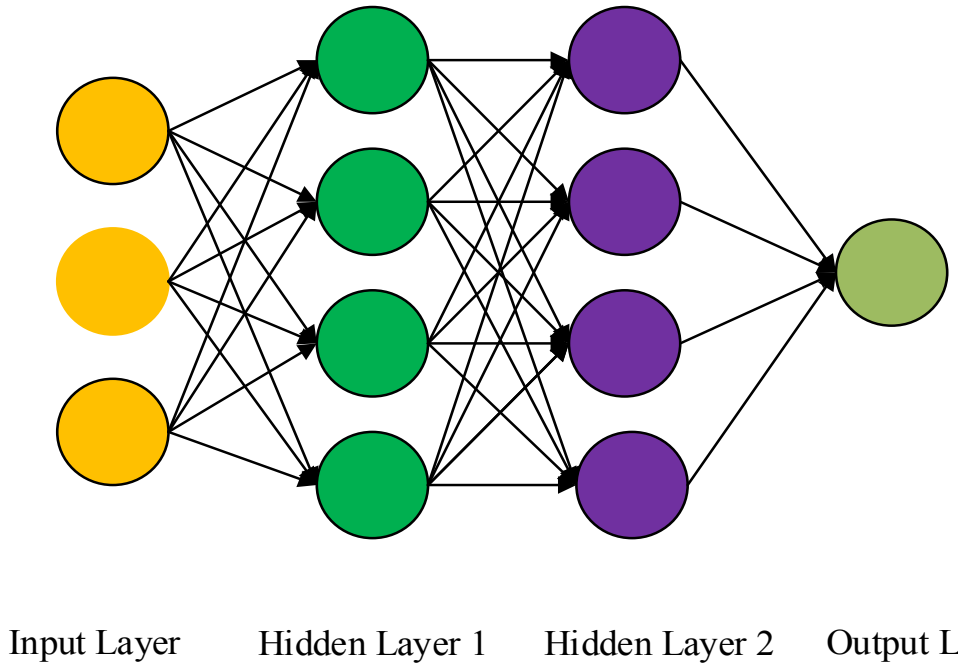

Figure 2-6 Artificial Neural Network

In the neural networks, the most difficult place is to determine the weight w and threshold b. If we hope to find the best weight w and threshold b, we could use one way: minor changes in w (or b) are recorded as Δw (or Δb), and then observe the output changes, as Figure 2-7 show us.

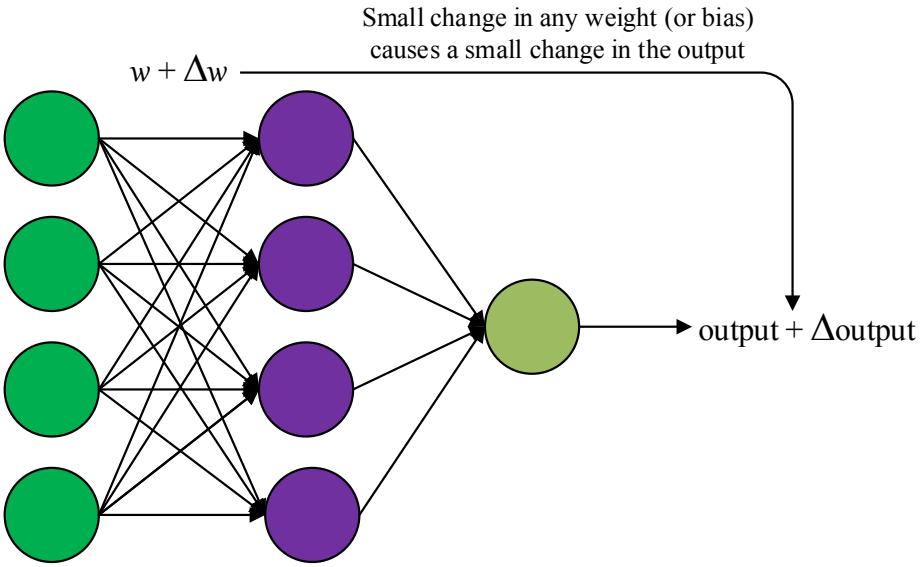

Figure 2-7 Change in weight or bias

2.3.4 Activation Function

As mentioned before, the output only gives you 0 or 1 to tell you it right or wrong. It cannot show us the output influence by changing in weight or bias. At this time, we need to use the Activation function, so that the output will be a continuity function.

Assume $z = wx + b$, we can image in a single neuron: $S = x_1 w_1 + x_2 w_2 + \cdots\cdots + x_n w_n + b$

if we use the original function, the output will look like in Figure 2-8.

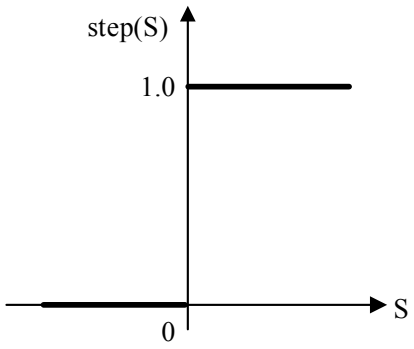

Figure 2-8 Step function

If the activation function we use is the Sigmoid, the Tanh, or the Relu. Like: Figure 2-9 (a), Figure 2-9 (b), or 2-10

$$Sigmoid(S) = \frac{1}{1 + e^{-S}}$$

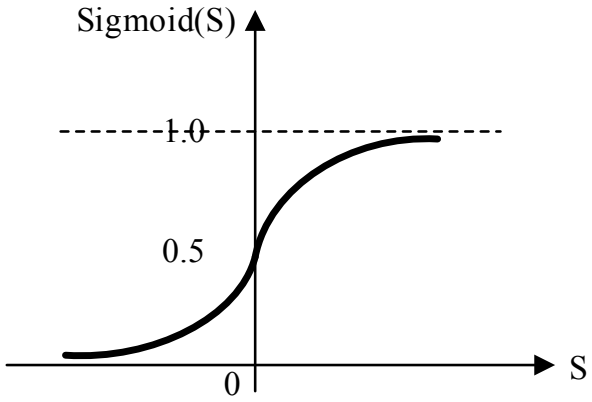

Figure 2-9 (a) Sigmoid function curve

$$Tanh(S) = \frac{2}{1 + e^{-2S}} - 1$$

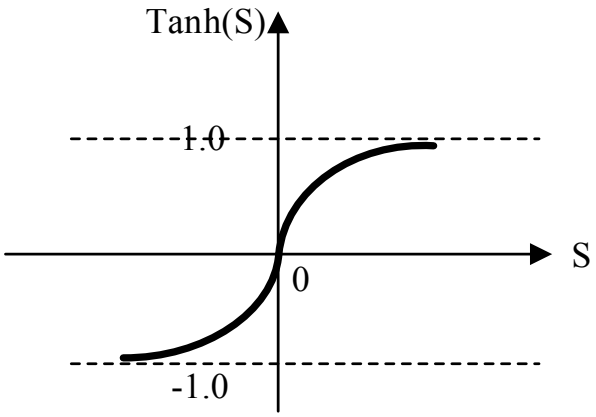

Figure 2-9 (b) Tanh function curve

$$Relu(S) = max(0, S)$$

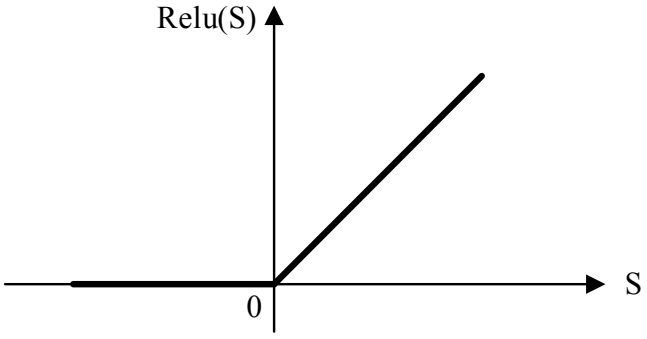

Figure 2-10 Relu function curve

2.4 What is Deep Q-learning (DQN)?

We already know that Q-learning has a Q-Table to save the different states, actions, and Q-values. However, there are many problems that are too complicated, and the state can be more than the stars in the sky (such as playing Go). If you use a Q-Table to store them, your computer might not have more memory, and it also needs a massive time for searching in the Q-Table.

1) We can use the state and action as the input of the neural network, and then get the Q-value of the action after the neural network analysis, so that we do not have to record the Q-value in the Q-Table, but directly use the neural network to generate the Q-value.

2)We can also input only the state value, output all the action values, and then directly select the action with the maximum value as the next action according to the principle of Q-learning.

Based on the second way, shown in Figure 2-10, we need the correct Q-values of (a_1, a_2). We will use the Q-value in Q-learning to replace it. Similarly, we also need a Q-Evaluation to update the neural network. So the parameters of the neural network are The old NN parameter plus the learning rate alpha (α) multiplied by the gap between Q-Target and Q-Evaluation.

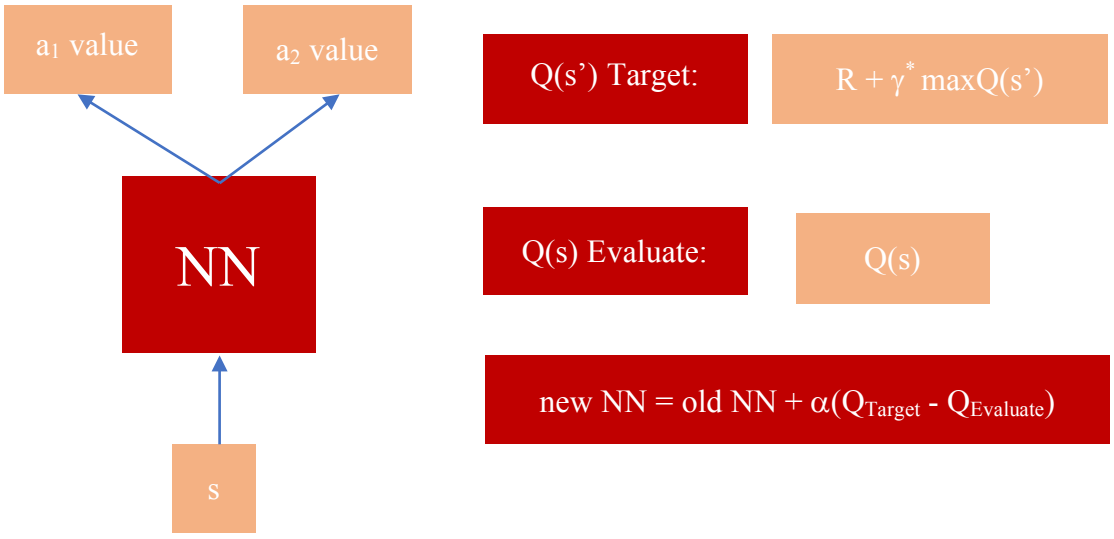

Figure 2-10 Update NN by Q-target and Q-evaluation

2.4.1 Experience replay

There is one crucial part of DQN, Experience replay. Q-learning is an off-policy offline learning method, and it can learn the experience of what is going through, it also can learn what has been experienced in the past, and even learn the experience of others. All experiences <s, a, r, s'> are stored in a retrospective memory pool. When the model is training, it randomly selects some previous experiences to learn. The random selection method disrupts the correlation between experiences and makes the neural network update more efficient.

2.4.2 Exploration and experience

At first, the Q-Table and Q-network are randomly initialized, and the following series of predictions are also random. If we select the action corresponding to the highest Q-value, then this action is naturally random. Now, the agent is doing "exploration". As the Q function converges, the returned Q-value will tend to be consistent, indicating that it has figured out the routine. The number of choices for exploration will also become less and less, we can say that exploration is also part of the Q-learning algorithm, but this exploration is "greedy", it will satisfy and find the first feasible routine

For the above problem, a simple and effective solution is an ε-greedy exploration, which is to use probability ε to choose whether to continue exploration or make decisions directly based on experience. Table 2-3 is the DQN pseudocode.

In line 11, DQN uses the experience reply, randomly selects the sample from the experience pool.

In line 12, DQN calculates the Q-Target value of Target Network.

In line 13, DQN perform the gradient descent to update the $\theta = \theta + \Delta\theta$

In line 14, DQN upgrade the Target Network value every C steps.

Table 2-3 DQN pseudocode

Algorithm Deep Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

for episode 1, M **do** Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$

for t =1, T **do**

With probability ε select a random action a_t

Otherwise select $a_t = \arg\max_a Q(\Phi(s_t), a; \theta)$

Execute action a_t in the emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$

Store experience $(\Phi_t, a_t, r_t, \Phi_{t+1})$ in D

Sample random minibatch of experience $(\Phi_j, a_j, r_j, \Phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & if\ episode\ terminatates\ at\ step\ j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\Phi_{j+1}, a'; \theta^{-}) & otherwise \end{cases}$

Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j;\ \theta))^2$ with respect to the weights θ

Every C steps reset $\hat{Q} = Q$

end for

end for

CHAPTER 3

DESIGN AND IMPLEMENTATION

In this section, the paper discusses the tools and environment used for the project. After that, the paper elaborates more details on the DQN model, which is used to train the agent that can play the card game Dou Di Zhu.

3.1 Tools and Environments

The thesis uses Python as the programming language for the project. The reason is that Python is a flexible language as the various artificial intelligence tools API. When using Python, the thesis can more easily use the Tensorflow, Scikit-learn, or Keras as the project's intelligence tools. By using Python, the thesis can focus on how to construct an architecture of the models rather than other programming languages. The version of Python that the thesis used is 3.6. The thesis also uses the Tensorflow as the model backend. The Tensorflow version is 2.2. The thesis uses the sublime text and Python IDEL as the editor.

The code is executed on two machines, one MacBook Pro and another desktop PC. The MacBook pro system is MacOS Mojave 10.14 with 2.7 GHz Intel Core i5 CPU, 8 GB

1867 MHz DDR3 memory, and graphic is Intel Iris Graphics 6100 1536 MB. The desktop PC system is Windows 10, with 4.2GHz Intel Core I7-7700K CPU, 25GM memory, and NVIDIA GeForce GTX 1070-Ti graphic. Because this project is not based on the images, so the work does not need to train my model on the cloud.

3.2 Dataset

The project trained a model to play the Dou Di Zhu card game. Based on these game rules, we know that players always use one deck with 54 cards for playing the game. The 54 cards will not change. At each game round, the players' roles (landlord and peasants) are different, the players' hand cards are different, and every turn for the players, their playing cards are also different. For training the model, the work needs to use the players played card at each turn as the input data.

Figure 3-1 Card suits.

One deck includes 54 cards, and four suits for each number: Hearts, Tiles, Clovers and Pikes, the card suits are showing in Figure 3-1. The work gives a number order for each card in the deck: 0, 1, 2, 3, …,50, 51, 52, 53. In this order: 0, 1, 2, 3 corresponds to a card:

3-Hearts, 3-Tiles, 3-Clovers, and 3-Pikes. 4, 5, 6, 7 corresponds to a card: 4-Hearts, 4-Tiles, 4-Clovers, and 4-Pikes, and so on. The 52 and 53 are Black Joker and Red Joker. In Dou Di Zhu, we do not need to check the card suits, only focus on the card number. The work can use the formula:

$$\text{card} = n \,//\, 4$$

n is the card order number, to calculate the real card number from 0 to 14. 0 is 3, 1 is 4, …, 9 is J, 10 is Q, …, 13 is Black Jack, 14 is Red Jack. Based on this way, the work succeeds in transferring cards into numbers that can be used in the code and model.

3.3. Flowchart of Technique

For building the machine learning model, we have designed the way to transfer card into the data could be used for the model. Figure 3-2 shows us how, before training the model, the flowchart of the method we need to do.

As Figure 3-2 shows us, after dealing the cards for players and bid the landlord, the current player's hand card needs to be decomposed and find out all possible card combinations based on current player hand cards and previously played cards. Then put these all possible card combinations as input data to feed the model for training. If we decide to train a Q-learning model, we can feed the data for the Q-learning algorithm, if we

decide to train a DQN model, we also can feed the data for the DQN model, and even if we can use random selection method. Then output the cards that current players need to play.

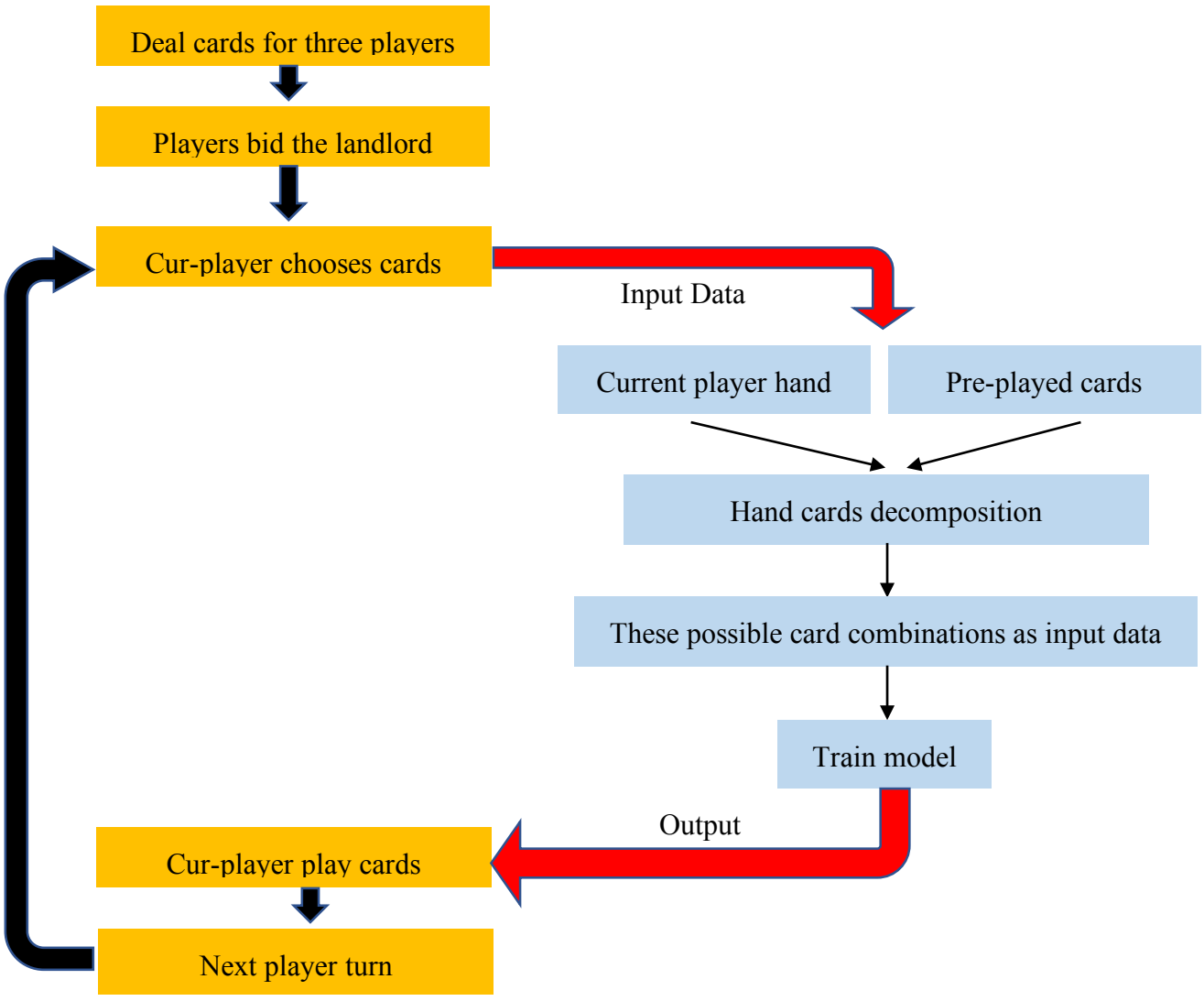

Figure 3-2 Game Flow

3.4 Card Decomposition

Before we feed the data for the training model, we need to find all possible card combinations that could be played. So, card decomposition is an essential part of this project.

Let us input the [*current player*, c*urrent player hand cards, previously played cards owner, previously played cards*] as the parameters for the card decomposition function. There one evidence need to check before:

previously played cards owner ≠ current player

or ***previously played cards owner ≠ None***

We need to follow the Dou Di Zhu rules base on ***previously played cards*** to decompose the hand cards if the above condition happened. Otherwise, we should decompose all possible card combinations of **c*urrent player hand cards***.

We prepared the function to check every card suit: single suit, pair suit, triple suit, bomb suit, rocket suit, and so on. Then use the get_total_moves function to find all possible card combinations and save them in the corresponding card type dictionary

separately. Then based on the *previously played cards* type, find out all card combinations that could play (beat the ***previously played cards***).

3.5 Q-Learning Architecture

Before designing the DQN model, we designed a Q-learning model for playing Dou Di Zhu. This Q-learning is very simple to understand. We utilized the Q-learning strategy that each player has an independent Q-Table to store the different playing action and corresponded reward.

In each game round, the players played cards should keep saving in a temporary list separately for each turn. When the round is over, these card combinations will transfer into Q-Table, and based on the win or loss of this game round, we plus one score or minus one score for these card combinations as the reward.

Figure 3-3 shows us how to update the Q-Table for one player. At the beginning of each game round, the “Historical card list” (list for saving the current player played card combinations in this round) is initialized. Whenever the player's turn, the player played cards would be saved in the “Historical card list”. During the game, we used a probabilistic

method for the Q-learning greedy, the probability based on the known and unknown card combinations in Q-Table. At the end of the whole game round, we can begin to update the Q-Table through “Historical card list” and win/lose.

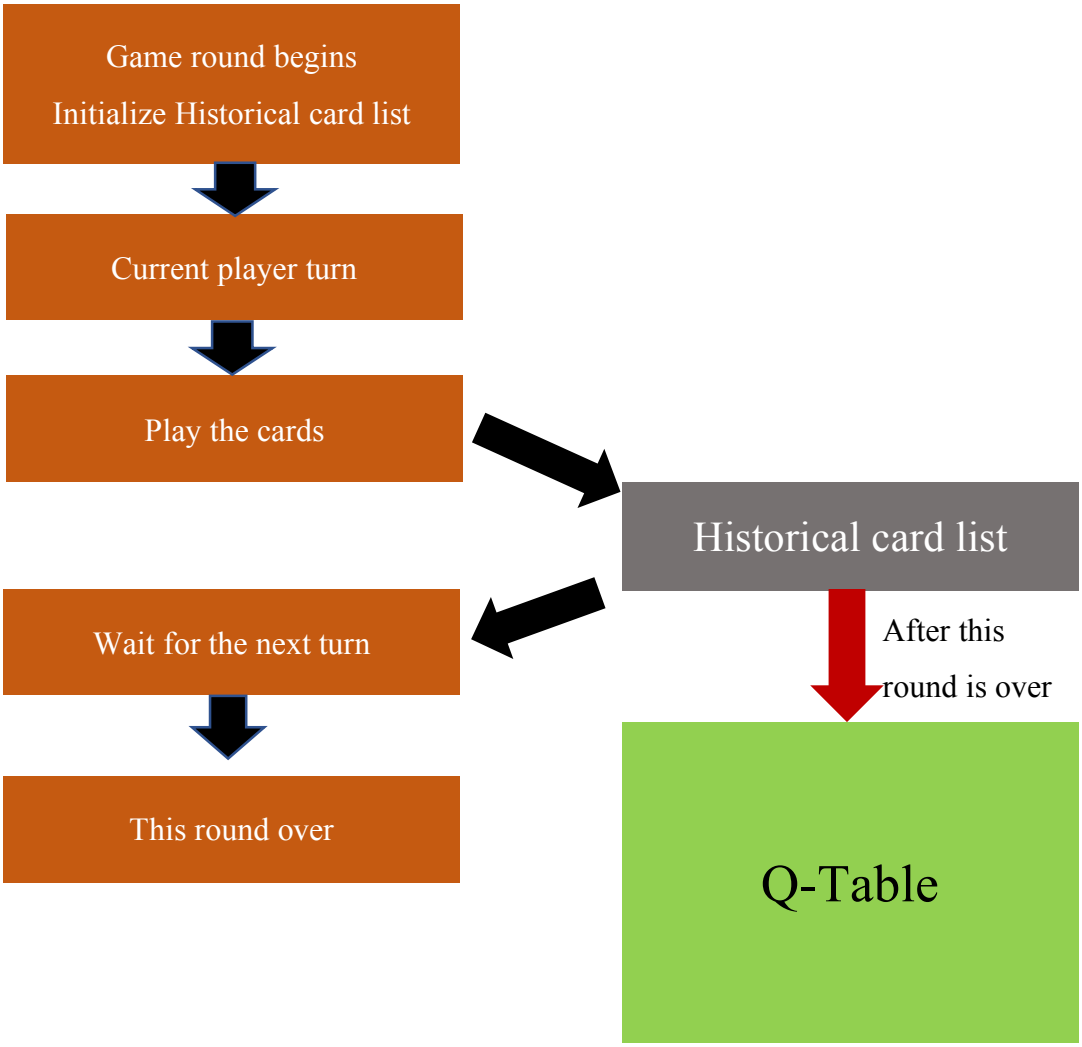

Figure 3-3 Update the Q-learning strategy

3.6 DQN Architecture

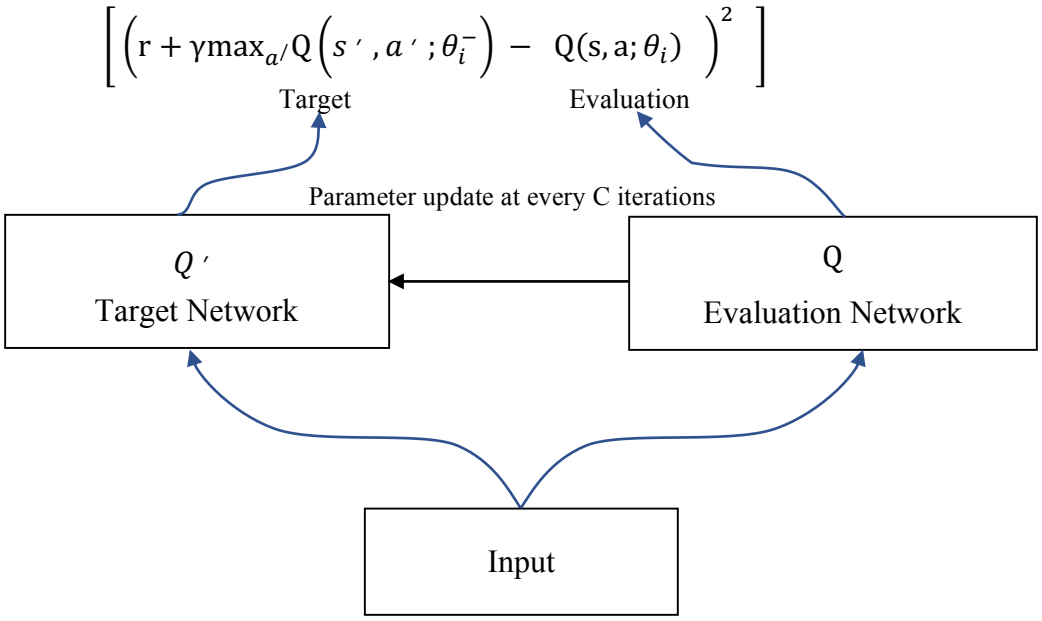

Figure 3-4 DQN model architecture

As shown in Figure 3-4, the DQN model is created by two same architecture networks: Target Network and Evaluation Network.

Our DQN model uses the Tensorflow to build the network. Before the data is inputted into the DQN model, this model needs to be set the parameters to build the network, and then the model could be trained. Because the model is prepared to train for Dou Di Zhu, and this game has many different card combinations, and an agent can play any combination in the game; So we set the model has two hidden layers as shows in the Figure 3-5. The input node numbers are the numbers of all card combinations. For the hidden layer 1, it has 256 nodes (neurons) with *Relu* activation function. For the hidden layer 2,

we also set 256 nodes (neurons) with *Relu* activation function. For the output layer, there are the same numbers of inputs. These layers are fully connected. For the parameters *weight* and *bias*, we use the ***tf.random_normal_initializer****(0., 0.3)* to initialize the *weight* value, and ***tf.constant_initializer****(0.1)* to initialize the *bias* value.

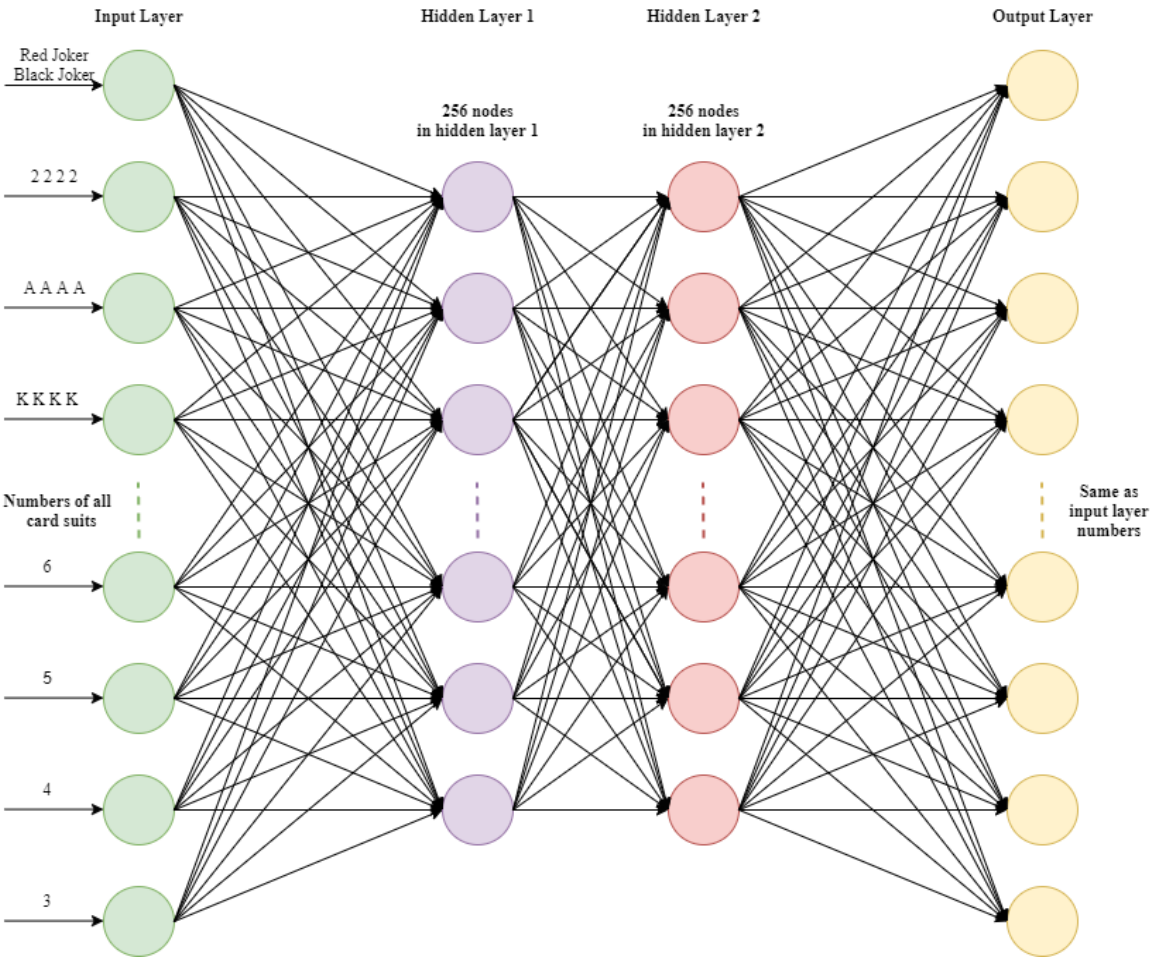

Figure 3-5 The network of Target and Evaluation

Both Target Network and Evaluation Network have the same setting. The different things between the two networks are: in the Target Network, we do not need to calculate

the *loss* or consider about train; in the Evaluation Network, the *loss* need the *loss function* to calculate (show in Formula 3-1). In this project, the initialized model has a vast number of input features. Nevertheless, when we train the model for playing the game, the model only feeds the all possible card combinations which could be played under each turn in the game round.

$$\text{Loss} = \text{E}\left[(q_{target} - q_{evaluation})^2\right] = E[(R + \gamma max_a Q\left(s^{'}, a^{'}\right) - max_a Q(s,a))^2]$$

Formula 3-1. the loss function of DQN

Except for the network setting, we also set the parameters for the memory pool. The size of the memory pool is called memory size, and the size value is 500, it means memory pool could save 500 *[action, state, reward, state']* value inside. We set *batch_size = 32*, so DQN would take 32 values from the memory pool each time when it needs to do the experience replay.

At last, we set the 100,000 times of training epoch. The first 2500 times for memory pool to save the record. After 2500 epoch, the DQN begins to learn and update. Every 300 epochs, in the DQN model, the Evaluation Network will replace the parameters of the Target Networks.

During the training, the model uses the "*e-greedy*" strategy to store the sample in memory pool and save them as *[action, state, reward, state']*. After a certain number of epochs, the DQN model begins to learn, when the model takes some "experience" from the memory pool, the Evaluation Network uses *state* from the "experience (*[action, state, reward, state']*)", and the Target Network uses *state'* from the "experience".

The *reward* is essential to influence the agent to select the *action*. In most cases, the agent has many different card combinations to select. So, we set different reward values for different card combinations (actions), like the model gets a minus reward if the agent can play a card or cards but does not play any card. The other reward rules are shown below:

1)The model gets a minus reward when it selects Bomb or Rocket if it also has other selection of card combinations.

2)The model prefers Single Sequence cards, more rewards with the longer sequence.

3)The model prefers Double Sequence cards, more rewards with the longer sequence.

4)The model prefers Triple with Single or Pair than Triple solo to get more rewards.

5)Win the game, and the model gets rewards; conversely, the model gets minus rewards.

CHAPTER 4

EXPERIMENT RESULTS

In this section, the results of the training models are discussed. We also compared the DQN model with the Q-learning model and the Zhou rule-based model we implement before.

4.1 Test

DQN VS Random

In this test, we compare with the DQN model with the random selection model.

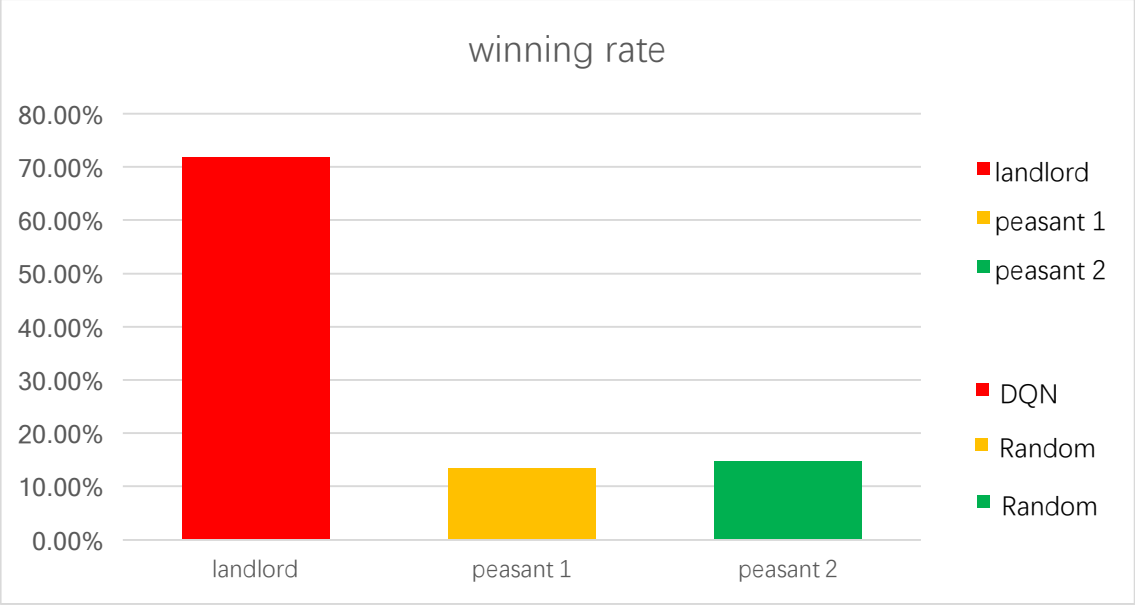

Figure 4-1 Result of DQN VS Random 1

After 100,000 times training and 10,000 times test, if DQN model trains the landlord agent, and the other two peasant agents use the random selection method, Figure 4-1 shows us the result: that DQN model has a 71.8% winning rate; peasant 1 has 13.4% winning rate, and peasant 2 has 14.7% winning rate; if DQN model trains the two peasant agents, and the landlord agent use the random selection method, Figure 4-2 shows us the result: that DQN model has 43.8% and 41.3% winning rate; the random selection landlord only has 14.9% winning rate.

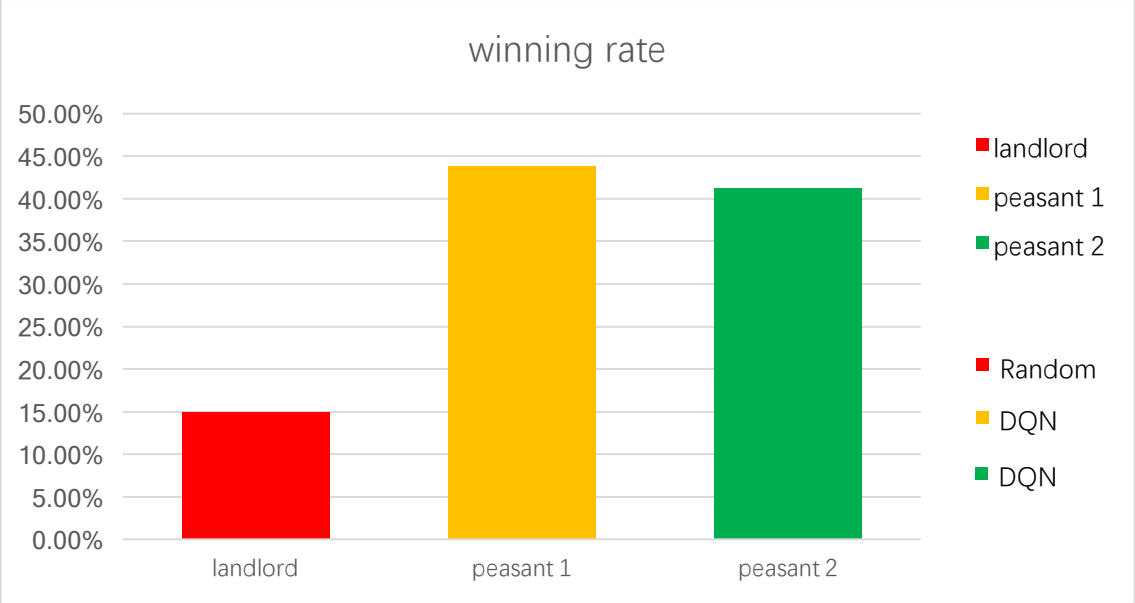

Figure 4-2 Result of DQN vs Random 2

DQN VS Q-learning

In this test, we compare with the DQN model with the Q-learning model.

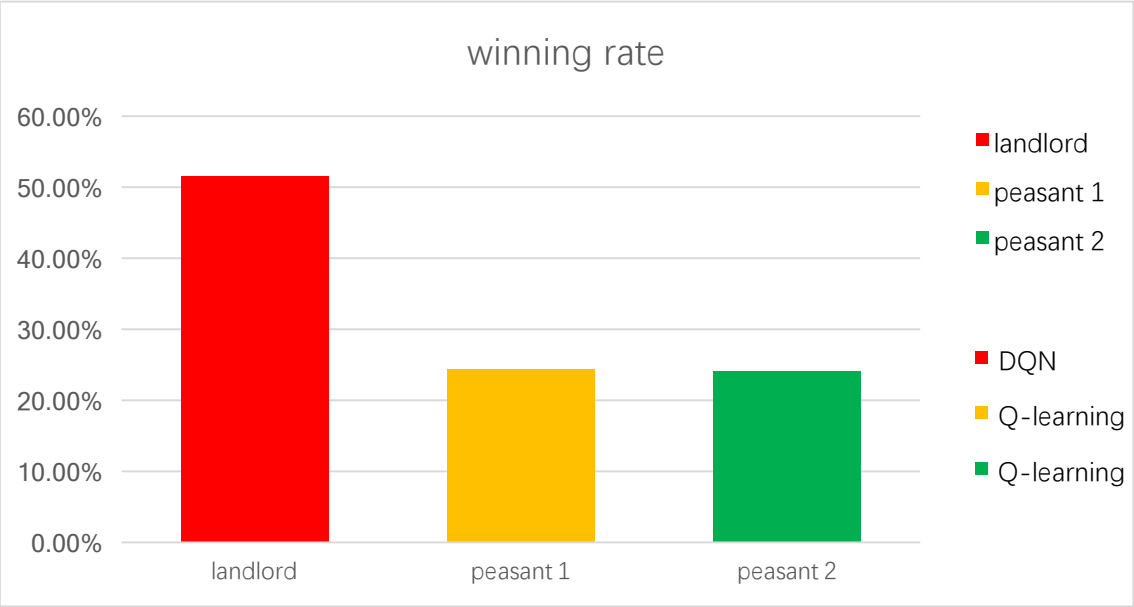

Figure 4-3 Result of DQN VS Q-learning 1

After 100,000 times training and 10,000 times test, if DQN model trains the landlord agent, and the other two peasant agents use the Q-learning model, Figure 4-3 shows us the result: that DQN model has a 51.6% winning rate; peasant 1 has 24.4% winning rate, and peasant 2 has 24% winning rate; if DQN model trains the two peasant agents, and the landlord agent uses the Q-learning model, Figure 4-4 shows us the result: that Q-learning model has 23.5% winning rate; the DQN peasants have 39.7% and 38.8% winning rate.

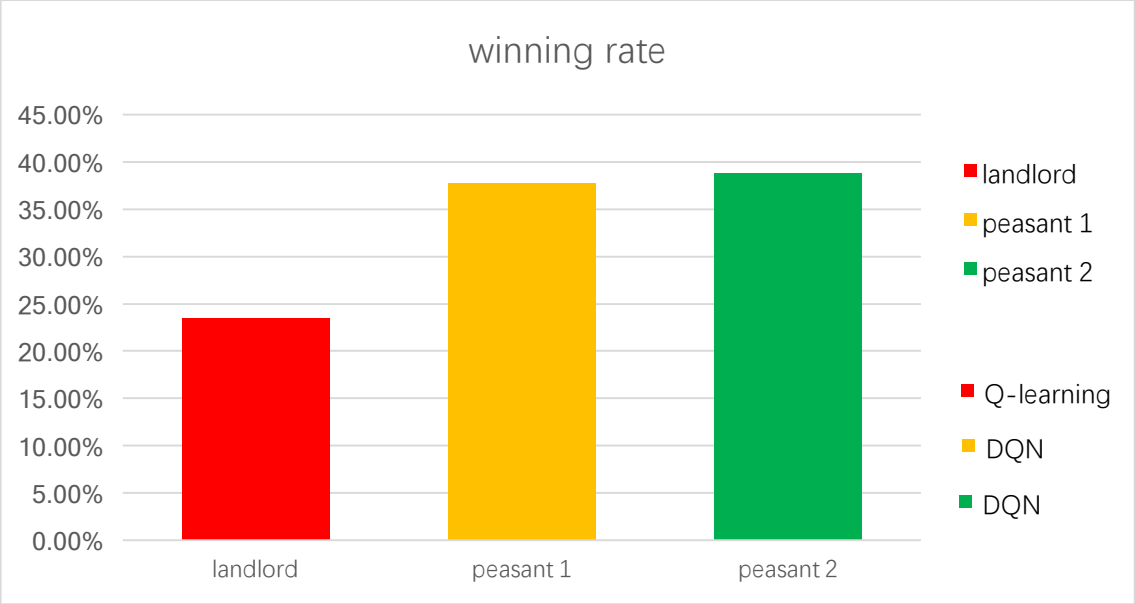

Figure 4-4 Result of DQN vs Q-learning 2

Q-learning VS Random

In this test, we compare the Q-learning model with the random selection method.

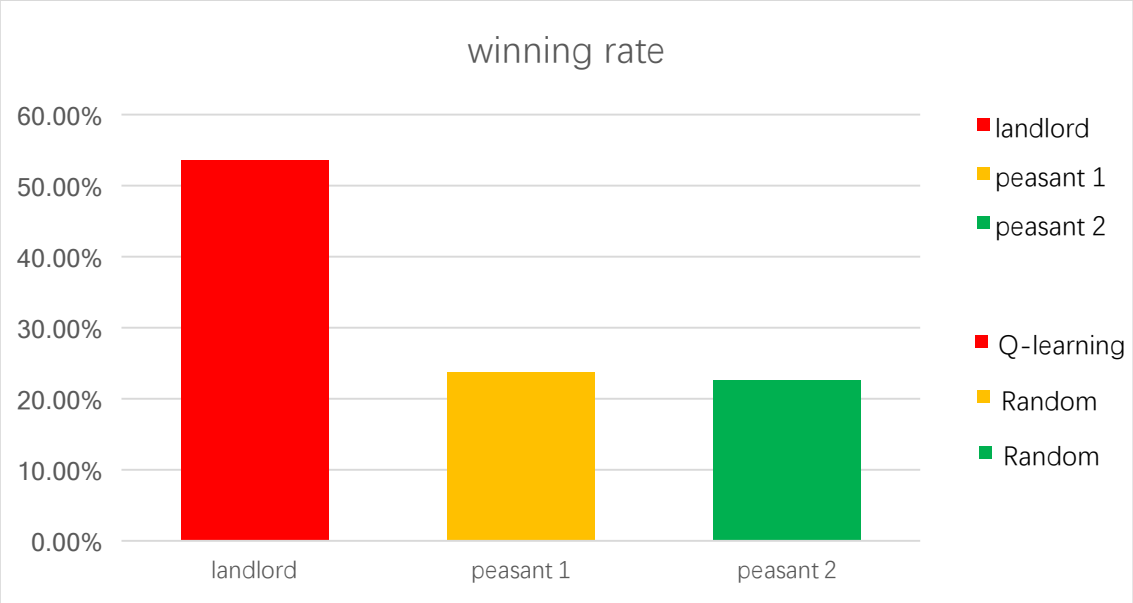

Figure 4-5 result of Q-learning VS Random 1

After 100,000 times training and 10,000 times test, if Q-learning model trains the landlord agent, and the other two peasant agents use the random selection method, Figure 4-5 shows us the result: that Q-learning model has 53.68% winning rate; peasant 1 has 23.70% winning rate, and peasant 2 has 22.62% winning rate; if Q-learning model trains the two peasant agents, and the landlord agent use the random method, Figure 4-6 shows us the result: that Q-learning model has 39.5% and 37.78% winning rate; the random selection landlord only has 22.72% winning rate.

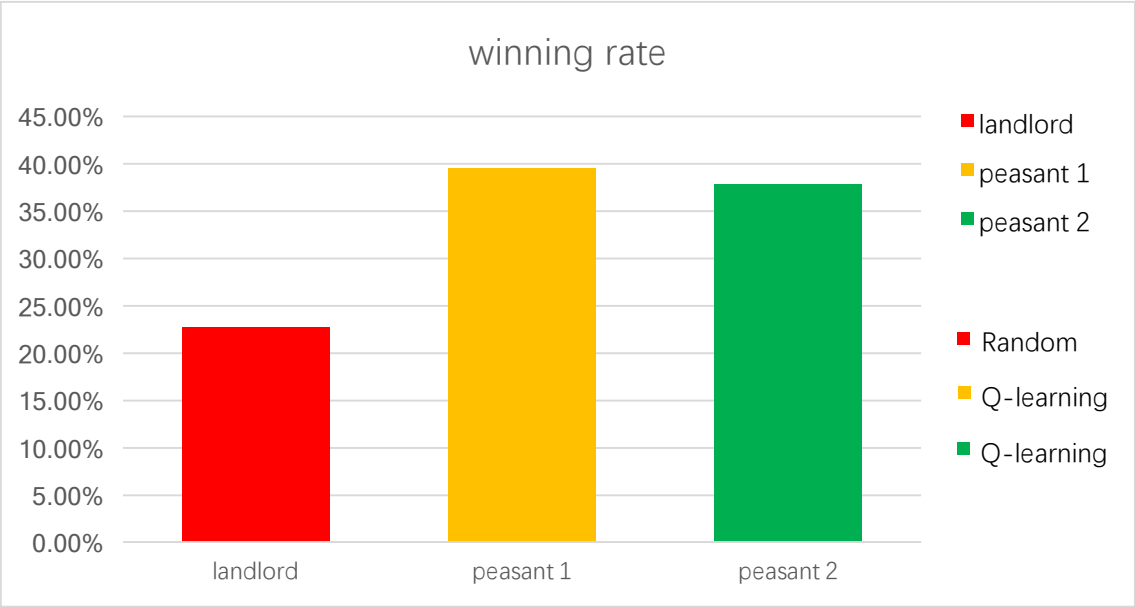

Figure 4-6 result of Q-learning VS Random 2

Rule-based model VS Random

This Zhou rule-based model is made by Zhouzhou [11]. We just re-implement the rule-based algorithm from his project.

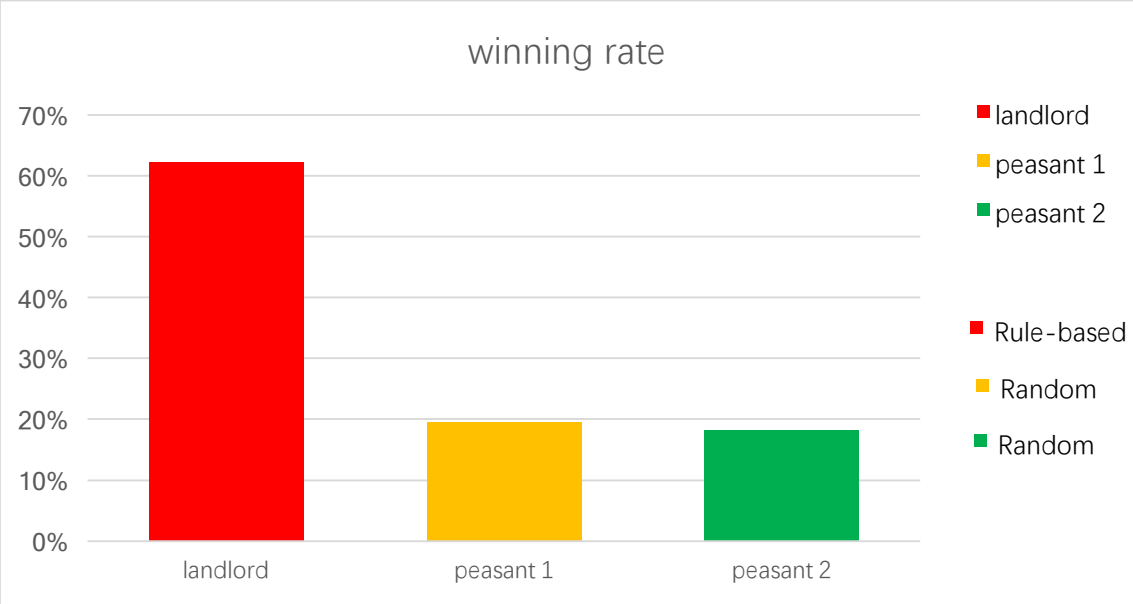

Figure 4-7 result of rule-based VS Random

As it is a Zhou rule-based model, we operate it for 10000 times. If the Zhou rule-based model trains the landlord agent and the other two peasant agents use the random selection method, Figure 4-7 shows us that the rule-based model (landlord) has 62.24% winning rate, peasant 1 (random selection) has 19.5% winning rate, and peasant 2 (random selection) has 18.26% winning rate. If the random selection method trains landlord agent and the other two peasant agents use the Zhou rule-based model, Figure 4-8 shows us that the landlord

(random) only has 19.1% winning rate, and two peasants (rule-based) have 39.2% and 41.7% winning rate.

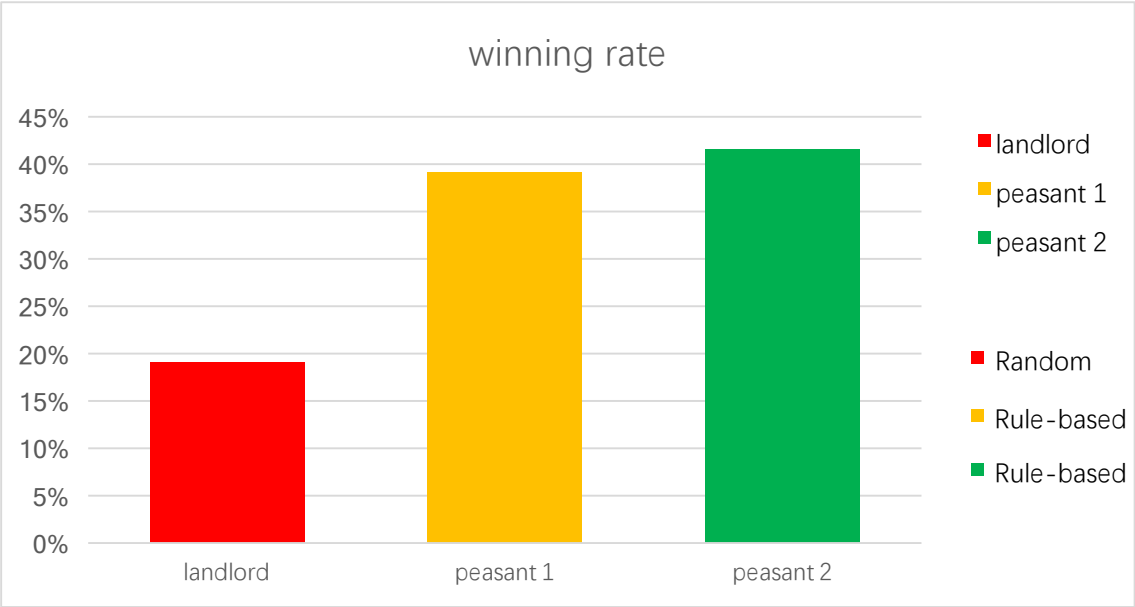

Figure 4-8 result of Rule-based VS Random

DQN VS Rule-based model

Except compare the Zhou rule-based model versus the random method, we also compare the DQN model versus the Zhou rule-based model.

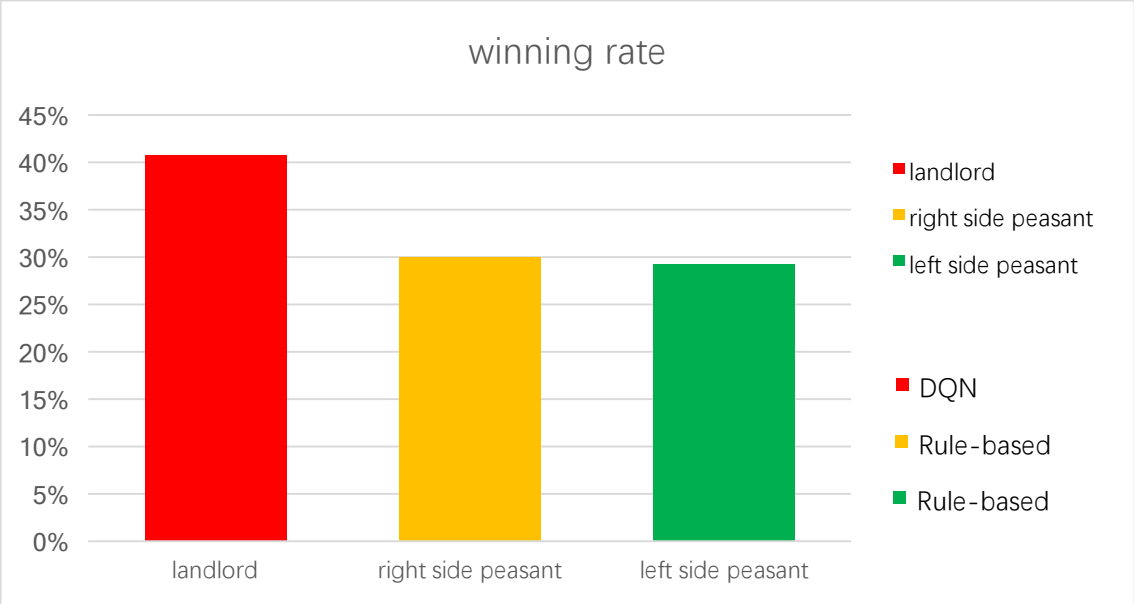

Figure 4-9 result of DQN VS Rule-based

After 100,000 times training and 10,000 times test, if DQN model trains the landlord agent, and the other two peasant agents use the Zhou rule-based model, the Figure 4-9 shows us the result: the landlord has 40.8% wining rate, and two peasants have 30.0% and 29.2% winning rate; if the Zhou rule-based mode trains the landlord agent, and the other two peasant agents use the DQN model, the Figure 4-10 shows us the result, the landlord has 28.8% winning rate, and the peasant right side of the landlord has 34.7%, and the peasant left side of the landlord has 36.5% winning rate.

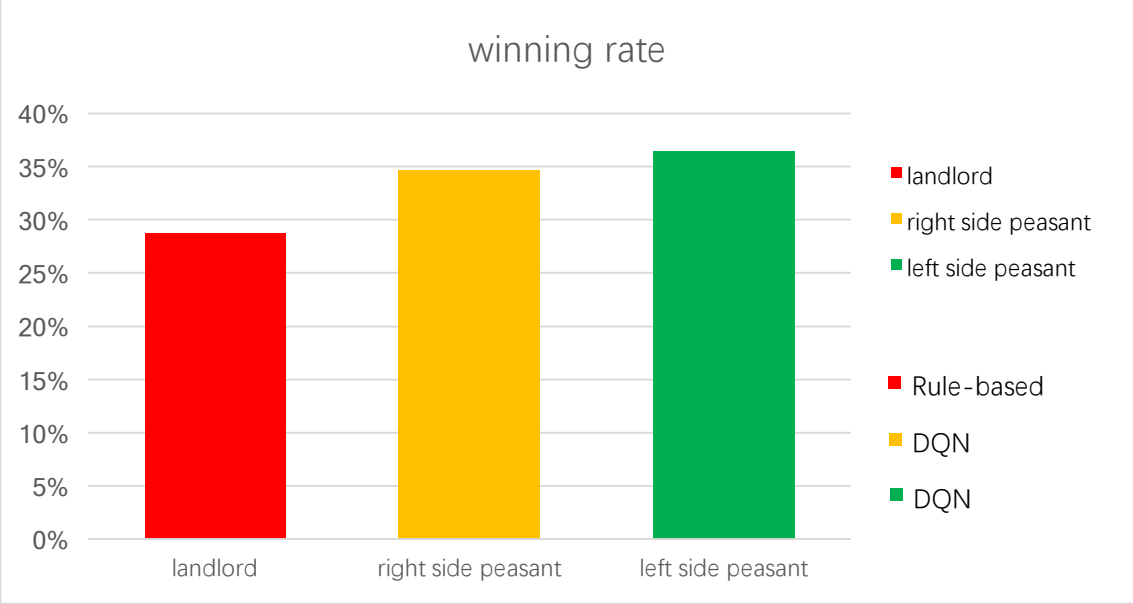

Figure 4-10 result of DQN VS Rule-based

DQN model VS Human

The last experience is about the DQN model versus the human player. First, the human player play as the landlord, and the two peasant agents use the trained DQN peasant agent models from the *DQN VS rule-based model* experience, which shows above. After ten games, the result is that: the human landlord get five game wins, and the two DQN peasant agents also get five game wins totally, like the Figure 4-11 shows us. The peasant, right side of the landlord, get two game wins, the peasant, left side of the landlord, gets three game wins.

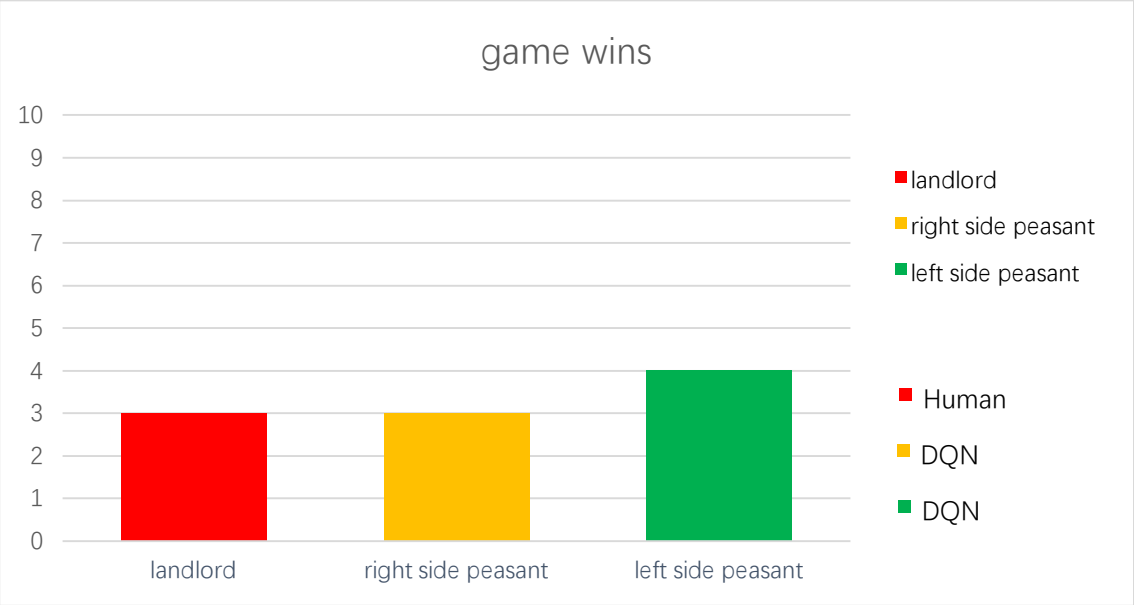

Figure 4-11 result of Human VS DQN

Next, two human players play as two peasants; the landlord uses the trained DQN landlord agent model from the *DQN VS rule-based model* experience, which shows above.

After ten game rounds, the result is that: the DQN landlord gets three game wins, and the two human peasants get seven game wins totally, like the Figure 4-12 shows us. The human peasant, right side of the landlord, gets three game wins, and the human peasant, left side of the landlord, gets four game wins.

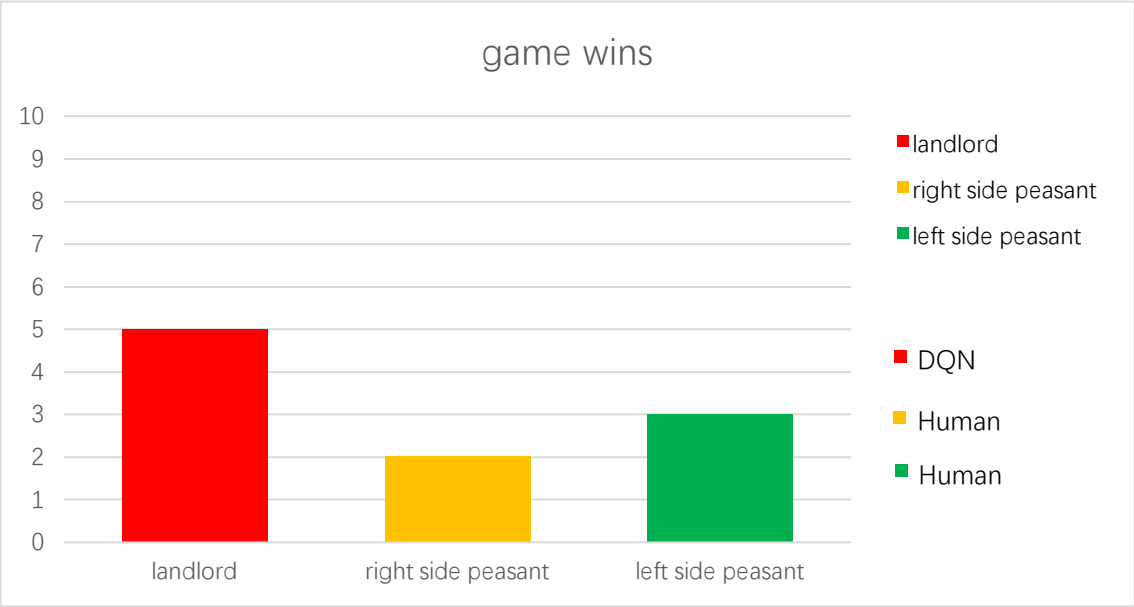

Figure 4-12 result of DQN VS Human

4.2 Observation

To compare the differences between the three models directly, we compare the results of the DQN model VS the random, the Q-learning model VS the random, and the Zhou rule-based model VS the random based on the previous test result.

Figure 4-9 shows us directly that three different models' winning rate on playing with the random selection methods. DQN has a 71.80% winning rate, Q-learning has a 53.68% winning rate, the Zhou rule-based model has a 62.24% winning rate. The DQN model has more than a 10% higher winning rate than the other two models.

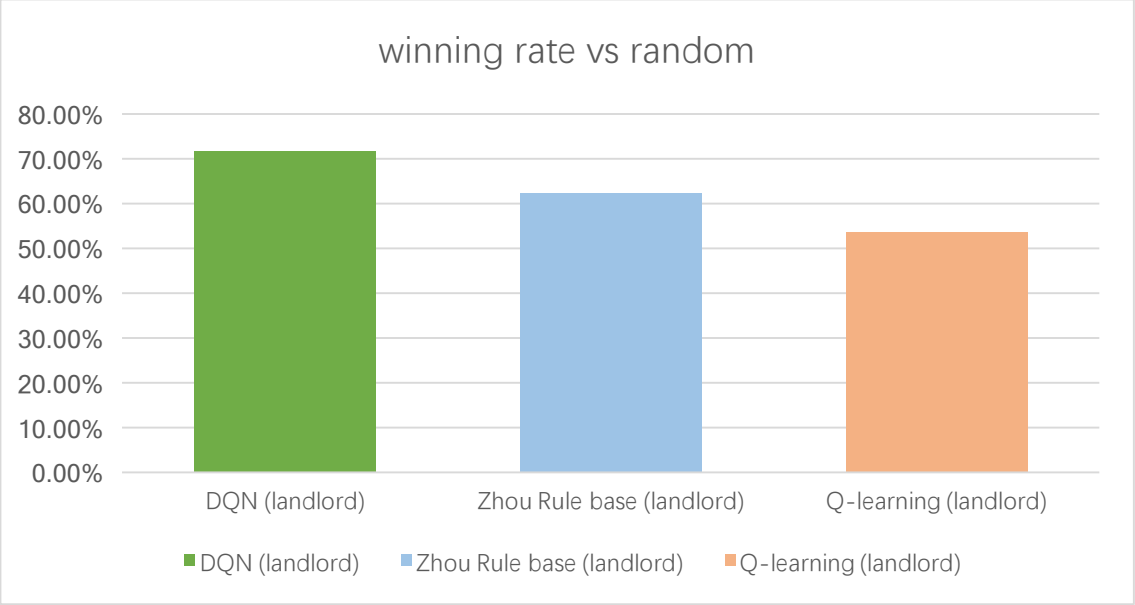

Figure 4-9 Three model comparison

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this project, we developed a DQN model for playing the Dou Di Zhu card game. We also made some comparisons between the DQN model, Q-learning model, and the Zhou rule-based model. Depends on the result, the DQN model is better than Q-learning and rule-based on playing Dou Di Zhu. In the same environment, we test the three models; each of them is set as the landlord agent and should play with random selection agents (peasants) in Dou Di Zhu. The result of the DQN model has the highest winning rate VS the random: 71.8%. The results of the Q-learning VS the random and the Zhou rule-based VS the random only have 60.24% and 55.79% winning rate.

Of the three models, DQN performed best, followed by the Q-learning model and the Zhou rule-based model. Our last experiment was to compare DQN versus human players. We performed a total of 20 test games. For the first ten games, we had DQN play the landlord agent against human peasants. For the last ten games, we had a human landlord play against two DQN peasants. As can be seen from our tables, the DQN agent performed only slightly worse than a human player, but it can still have a good game interaction with human players.

One important aspect of Dou Di Zhu we have not implemented is the process of bidding to become the landlord. Dou Di Zhu is a luck-based game like Texas Hold'em. In Dou Di Zhu, after the players are dealt their hands, they must bid to be or not to be the landlord. The initial hand assignment together with this bidding are critical in determining whether a player wins or loses. As a possible future work, it would be nice to programmatically implement this landlord bidding process. It would also be nice to consider different variants of the DQN model, such as DDQN, Prioritized ReplayDQN, and Dueling DQN.

BIBLIOGRAPHY

[1] Temple, Robert K.G. (2007). The Genius of China: 3,000 Years of Science, Discovery, and Invention (3rd edition). London: André Deutsch, pp. 130–131. ISBN 978 0-233-00202-6.

[2] Zhou, Songfang. "On the Story of Late Tang Poet Li He", Journal of the Graduates Sun Yat-sen University, 1997, Vol. 18, No. 3:31–35

[3] Needham, Joseph and Tsien Tsuen-Hsuin. (1985). Science and Civilization in China: Volume 5, Chemistry and Chemical Technology, Part 1, Paper and Printing. Cambridge University Press reprinted Taipei: Caves Books, Ltd.(1986), Page 131

[4] Parlett, David (1990), The Oxford guide to card games: a historical survey, Oxford University Press, ISBN 978-0-19-214165-1

[5] "Dou DiZhu." Wikipedia, Wikimedia Foundation, date of last modification, https://en.wikipedia.org/wiki/Dou_dizhu

[6] "Q-learning." Wikipedia, Wikimedia Foundation, date of last modification, https://en.wikipedia.org/wiki/Q-learning

[7] Chen, Yung-Yao; Lin, Yu-Hsiu; Kung, Chia-Ching; Chung, Ming-Han; Yen, I.-Hsuan (January 2019). "Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes". Sensors. 19 (9): 2047.

[8] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforceme-

nt Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. Archived from the original on 2001-11-20.

[9] Renzhi Wu, Shuai Liu, Shuqin Li, Meng Ding, "The design and implementation of a computer game algorithm of Dou Dizhu", 2017

[10] Zhennan Yan, Xiang Yu, Tinglin Liu, Xiaoye Han, "Fight the Landlord (Dou Di Zhu)"

[11] Zhouzhou, DouDiZhu AI, https://github.com/zhozhou/DouDizhuAI

[12] Zhaokun Yang, Chongjun Yang, "Challenges for future mathematicians", https://web.archive.org/web/20180712004127/http://episte.math.ntu.edu.tw/articles/mm/mm_10_2_04/page6.html