Deep Strategical Reinforcement Learning for Gomoku Game

Xin Huang [*1] **Ziqi Zhang** [*1]

Abstract

In this project, we will review the knowledge of reinforcement learning and have research on the Gomoku Game. We start from the simpler Tic-tac-toe Start with the game, explore reinforcement learning strategies and applications with deep learning.

The difference from traditional reinforcement learning methods, we developed a strategical reinforcement learning scheme by taking the strategies in Gomoku Games, and it works much better than vanilla reinforcement learning schemes in the same training epochs.

1. Introduction

Reinforcement Learning is a efficient scheme in modeling and solving sequential problems, especially in decision making process. In the past decades, researchers have developed numerous algorithms in reinforcement learning for discovering optimal policies in certain models. As one of the classic algorithms in reinforcement learning, Q-learning (Watkins & Dayan, 1992) has shown its advantages in handling model-free problems, it can compare the expected utility of available operations (for a given state) without the need for an environmental model. At the same time, it can handle random transitions and rewards without adjustment. It has been proven that for any limited MDP(Markov decision process), Q-learning will eventually find an optimal strategy, that is, starting from the current state, the expected value of the total return of all successive steps is the maximum that can be achieved.

In recent years, scholars have made important breakthroughs and achievements in the field of deep learning. In the field of computer vision, image recognition and natural language processing, deep learning neural models have achieved great success. With a series of sequence models such as CNN and RNN (Schmidhuber, 2015), and the proposal of AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2015) and a series of networks and frameworks, deep learning ushered in the culmination of research and application.

Q-learning is a classic method in reinforcement learning, and deep learning and neural networks are the most popular important concepts and tools in recent years. The Deepmind team from Google proposed to combine Q-learning with neural networks and proposed DQN (Deep Q -Network) (Mnih et al., 2013), and achieved outstanding results in the game field. From Atari games (Mnih et al., 2013) to Google's Alpha Go (Silver et al., 2016) as we know it, reinforcement learning continuously updates strategies in multiple iterations, so as to achieve human-like or even human-like game "manipulation" capabilities.

Gomoku, also known as five-game winning streak, is an abstract strategy board game. It is traditionally played with Go (black and white pieces) on the Go board. It uses a 15×15 board for the game, the game is widely known in many countries and regions under different names. In this project, we will apply the Deep Learning and Reinforcement Learning knowledge in to Gomoku Game to build a smart and competitive agent.

2. Background

In general, Reinforcement Learning can be divided into model-based RL and model-free RL. Model-based RL is a method that attempts to overcome the issue of a lack of prior knowledge by enabling the agent to construct a functional representation of its environment. One case for that is Dynamic Programming, the specific process is:

Firstly evaluate the policy π:

$$v_\pi(s) = E\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] \quad (1)$$

And then improve the policy by acting greedily:

$$\pi' = greedy\left(v_\pi\right) \quad (2)$$

However, in our project, the agent does not know the transition function nor the reward function, in that case, it is very difficult to use model-free RL.We should pay our attention to model-free RL.

[*]Equal contribution [1]Department of Electrical Engineering, Columbia University, New York, United States. Correspondence to: Xin Huang <xh2510@columbia.edu>, Ziqi Zhang <zz2881@columbia.edu>.

ELEN 6885 Final Project, Project Instructor: Chong Li & Chonggang Wang.

Model-free RL is a model that we don't have to learn a model of the environment to find a good policy. One of the most classic examples is Q-learning, which directly estimates the optimal Q-values of each action in each state from which a policy may be derived by choosing the action with the highest Q-value in the current state. In our project, since our data is not high-dimensional, it is feasible to store the Q value of each state-action pair in a Q-Table to record the result of each state.

Although the reliability of making a Q-table, the hugeness and complexity of our data makes it very difficult and time-consuming to construct the model. To solve that, we could further turn the Q-Table update problem into a function fitting problem by the following formula:

$$Q(s, a; \theta) \approx Q'(s, a) \tag{3}$$

We could update θ to make Q function approximate the optimal Q value. Since deep neural network can automatically extract complex features, we can further combine Convolution Neural Network with Q-learning (Mnih et al., 2013). Each state is used as the input of network, and the output is the value function corresponding to each Value Function.

2.1. Label Construction

Since it is necessary to fit the function of the Q-Table in a high-dimensional and continuous state, we first need to construct a function to represent the difference between the fitted value and the true value. This function is called loss function.

According to the Q-learning formula, if the optimal value $Q(s, a)$ of the sequence s at the next time-step was known for all possible actions a, then the optimal strategy is to select the action a maximising the expected value of $r + \gamma Q(s, a)$:

$$Q^*(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s, a) \right) \tag{4}$$

We can deduce the loss function:

$$L_i\left(\theta_i\right) = E_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s, a; \theta_i\right)\right)^2\right] \tag{5}$$

where y_i is the target in i^{th} iteration.

$$y_i = E_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) \mid s, a\right] \tag{6}$$

We use ϵ-greedy as our strategy, and the parameters from the previous iteration θ_{i-1} are held fixed when optimising the loss function L_i.

Finally, we can use the stochastic gradient descent method to find the optimal value of θ for L_i.

2.2. Experience Replay

Considering that we need to use SGD, which requires that the training data is independent and identically distributed and when the Agent interacts with the Environment,the sequence of experience tuples needs to be highly correlated, we need to put the sample(S, A, R, S') into the reply buffer (a large buffer of our past training data).

We only store the last N experience tuples in the playback memory and sample them evenly. This may cause the memory buffer to make no difference in the importance of the input data. In the further experiments, we will set different weights to emphasize transitions from which we can learn the most.

2.3. Neural network

We have used two CNN to approximate the optimal Q-Value and update the target Q-Value. In this way, in a period of time, the target Q-Value remains unchanged, which reduces the correlation between the current Q-Value and the target Q-Value to a certain extent, and improves the stability of the algorithm.

3. Tic-tac-toe Game

At the beginning of this section, we will analyze the simple game of Tic-tac-toe (with a 3×3 board). Compared with the huge 15×15 board of Gomoku and the more stringent victory conditions, tic-tac-toe simulates to a certain extent the game idea of Gomoku with smaller complexity.

3.1. Game Model

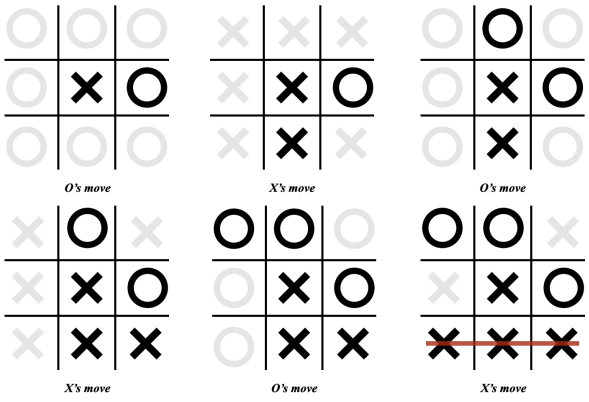


Figure 1. Tic-tac-toe Game: A full epoch of game.

The Figure 1 above shows a typical epoch of the Tic-tac-toe game.

For the Tic-tac-toe game, because the game board is only 3×3, the amount of iterative calculation required to update

the Q-Table is small, so we take the method of directly updating the Q-Tabel to train the agent. During training, when the agent wins the game, we give a reward of 1, and if it loses the game, there is no reward. Based on this, we count the game winning rate of the agent.

3.2. Experiment

In the Tic-tac-toe game, it's worth noting that the Tic-tac-toe game has a first-hand advantage. Based on this, the game winning rate calculated in our simulation is 5000 times for the agent first and second, a total of 10000 times. Table 1 below shows the training rate of our agent battling against a 2000-trained agent.

Table 1. Winning rate of our agent against a computer trained 2000 epochs.

Training Epochs	*Winning Rate*
2000	0.5879
4000	0.7623
6000	0.8665
8000	0.9422
10000	0.9751

Also, we carried the experiment on our agent battling against an agent with different training epochs. Here's the statistics.

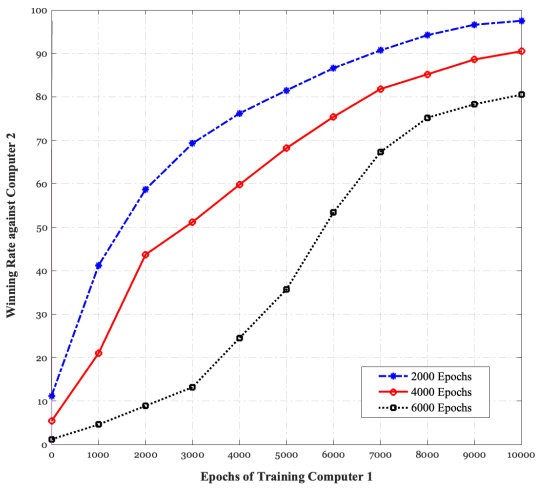


Figure 2. Winning rate of the agent against an agent with varied epochs of training.

From the simulation, we can tell that the method of updating Q-Table is quite useful in Tic-tac-toe games, and our agent shows its competitiveness in the games.

4. Gomoku Game

4.1. Game Model

In the previous section, we introduced the Tic-tac-toe game, and briefly introduced the strategy of updating the Q-Table based on the basic Q Value. The traversal complexity of Tic-tac-toe is low, and the number of trainings increases. Next, the agent in Tic-tac-toe can learn the game strategy well and win the game.

Gomoku Game and Tic-tac-toe are very similar in form: a board game; n pieces of chess can be connected together to win, etc. The picture below shows a game of Gomoku. In the end, the white player won.

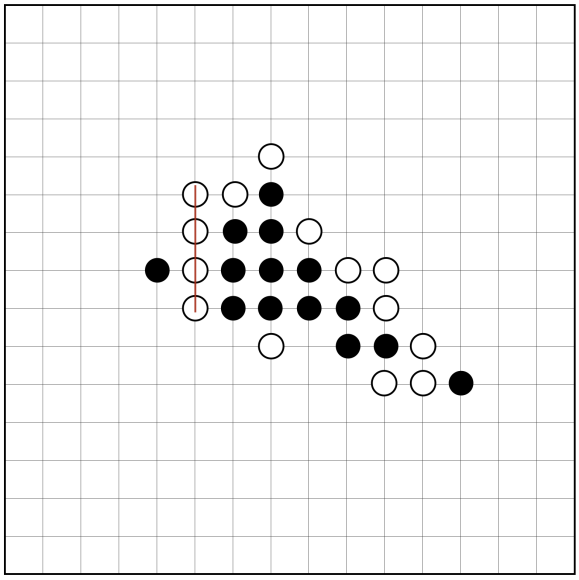
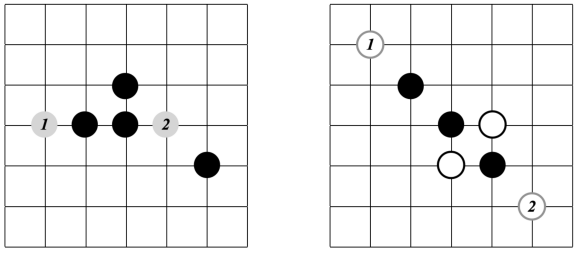


Figure 3. Gomoku Games.

One point different from the simple Tic-tac-toe is that we noticed that there are a lot of technical strategies in the Gomoku Game, such as the construction of "Double-three" and other strategies.



(1) Offense Strategy: Make Double-Three *(2) Defense Strategy: Make Better Block*

Figure 4. Example of strategies in Gomoku Games.

(1) is an example of offense strategy in Gomoku Games, called Make Double-Three; (2) corresponds to an example of defense strategy in Gomoku Games, called Make Better Block.

We recall the Equation (6),

$$y_i = E_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) \mid s, a \right] \quad (7)$$

The reward r in Equation (6) can be now divided as a strategical reward r_{str} of three weighted rewards.

$$r_{str} = \gamma_1 \cdot r_1 + \gamma_2 \cdot r_2 + \gamma_3 \cdot r_3 \quad (8)$$

where γ_i is the weighted factor, and $\gamma_1 + \gamma_2 + \gamma_3 = 1$. Here, we set $\gamma = [\gamma_1 = 0.8, \gamma_2 = 0.1, \gamma_3 = 0.1]$ corresponding to the weighted factor of three rewards: Victory Reward r_1, Offense Reward r_2 and Defense Reward r_3.

The purpose of doing so is that we observe that the 15×15 board brings a complexity for training, even if we use DQN to achieve that.

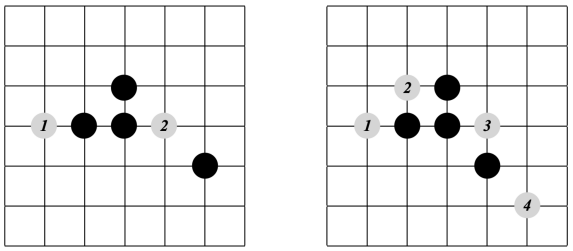
We realize there are certain strategies in Gomoku Game, we add a weighted reward to Gomoku game to make the agent learn from not only wins but also from loses when they take right moves (strategies).

Therefore we consider 3 key factors, the results of wins, the offense and defense moves.

And in following subsections, we will discuss the reward functions more.

4.1.1. Offense Reward

The figure below shows examples of strategic moves in Gomoku Games.



(1) Offense Strategy: Make Double-Three *(2) Offense Strategy: Make Potential Three*

Figure 5. Examples of offense strategies.

In order to take the offense strategies into consideration, as shown in Equation (8), we weighted the reward of three terms, and offense reward is denoted as:

$$r_2 = log(max(1 + \epsilon, \frac{N_b}{N_w})) \cdot (N_{s+1} - N_s + \epsilon) \quad (9)$$

Here, we assume we take the black and opposite takes the white, N_b and N_w denotes the number of pieces of black and white in the 5×5 window where the move is at the center, shown in Figure below.

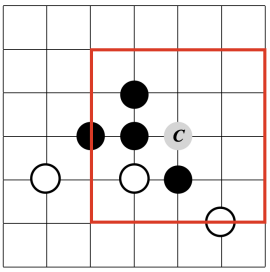


Figure 6. A 5×5 window centralized at move C, marked in red.

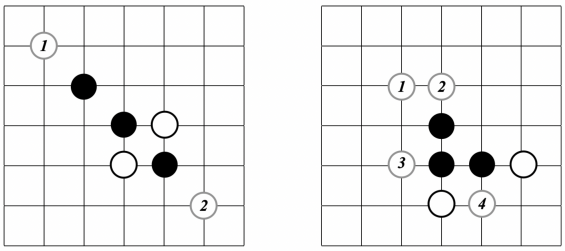
In the Figure 6 shown above, the $N_b = 3$ and $N_w = 2$.

N_s denotes the number of "Threes" at state s, similarly, N_{s+1} denotes the number of "Threes" at state $s + 1$, and ϵ denotes a small number, usually, $\epsilon = 10^{-6}$. In this case $N_s = 0$ and $N_{s+1} = 1$.

Equation (9) measures the choice of moves, whether it's a good move or a bad one, whether the move can lead to a strong offense.

4.1.2. Defense Reward

In the Gomoku Games, besides offense, the defense plays a critical rule as well. Figure 7 shows two examples of defense strategies.



(1) Defense Strategy: Make Better Block *(2) Defense Strategy: Handling Potential Three*

Figure 7. Examples of defense strategies.

Similar to offense reward we discuss in Section 4.1.1, we give reward to agent while they take a strategical offense move, the defense reward r_3 is denoted as:

$$r_3 = log(max(1 + \epsilon, \frac{N_w}{N_b})) \cdot \frac{1}{N_{s+1} - N_s + 1} \quad (10)$$

Here, corresponding to the figures, we assume the strategical agent takes white and vanilla agent take black, N_b and N_w

denotes the number of pieces of black and white in the 5×5 window, as illustrated in Figure 7 below.

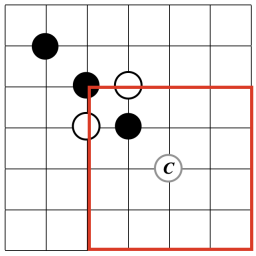


Figure 8. A 5×5 window centralized at move C, marked in red.

In the Figure 7 shown above, the $N_w = 3$ and $N_b = 2$.

Similarly, N_s denotes the number of "Threes" at state s, similarly, N_{s+1} denotes the number of "Threes" at state $s + 1$, and ϵ denotes a small number, usually, $\epsilon = 10^{-6}$. In this case $N_s = 1$ and $N_{s+1} = 1$.

4.1.3. Victory Reward

This part is simple, the victory reward r_{vic} is the same as the conventional calling of reward in reinforcement learning, and $r_{vic} = 1$ while agent makes the win and $r_{vic} = 0$ for loses.

But as we are handling a complex model, we add a bonus term to the victory reward, it's called Immediate Victory Reward, which can be written as:

$$r_1 = r_{vic} + r_{imm} \tag{11}$$

And the immediate reward r_{imm} corresponds to a fact whether the player can have a immediate win after taking the strategical move(mainly offense strategy move), especially in t moves after taking the strategical move.

$$r_{imm} = \alpha \cdot \frac{1}{t} \tag{12}$$

And α is the strategical learning rate, in this scenario, we take $\alpha = 1$. The term r_{imm} is to award the agent if it takes an effective strategical move(mainly offense strategy move) and make an immediate win regarding that move.

So far, we have modified the original reward function to a novel reward function of a comprehensive consideration of offense strategy, defense strategy and game result.

4.2. Experiment

For Gomoku Games, due to the shortage of time and limited hardware resource, we only trained the agent for limited epochs, due to the computational complexity of a 15×15 board and moves, the agent play poorly against human players. Therefore, we only make comparisons between trained agents with strategical algorithm (Section 4.1) and pre-trained computers with simple Q-Table updating algorithm.

We trained our agent with strategical weighted rewards and vanilla agent for 1000 epochs (it takes nearly two days to train, even 1000 epochs), and the table below shows the winning rate of our agent against vanilla agent, and one thing to notice, we simulated for 500 games and 250 games our agent took first move and 250 games with second move(since the first move takes a great advantage in games, but we thought with very few training epochs, the advantage won't be obvious since the agent isn't smart enough).

Table 2. Winning rate of our strategical agent against vanilla agent.

Epochs	***Winning Rate***
1000	0.617
2000	-
...	...

From the statistics, we can tell that our strategical reinforcement learning scheme shows the potential in competitiveness in Gomoku Games, especially in limited training epochs(surely, this is also an assumption that the vanilla agent will supersede our strategical agent in a great amout of training epochs, since the vanilla is obviously under-trained in this case).

5. Conclusion

In this project, we started from Tic-tac-toe Games which have simple computational complexity to Gomoku Games which is apparently more complicated and hard to train.

We took the strategical moves in Gomoku Games into consideration and modified the original reward function in reinforcement learning to a more comprehensive and weighted one with offense reward and defense reward. In limited training epochs, our strategical agent started to show its potential.

Due to the time shortage and hardware resource limitation, we can only show part of our model in experiment and simulation part, but we are quite glad to share the ideas of add reward of strategical move in Gomoku Games, and without doubt, for the offense and defense strategies, and model isn't the best and the reward function may not be the best to fit the strategies.

We believe the ideas of strategical reinforcement learning will have a broader use in more games and applications.

Acknowledgements

Here we would like to thank Professor Chong Li and Professor Chonggang Wang for their teaching and assistance, as well as the assistants for answering questions about us in their spare time and Office Hour.

In the process of completing this project, we read a lot of materials and tutorials on the Internet, as well as the code of the basic environment that uses some open source reinforcement learning. We also thank these evangelists with knowledge in the academic field.

References

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

A. Code

Code (some parts) of our project is on Github:

https://github.com/xinhuang09/Deep-Strategical-Reinforcement-Learning-for-Gomoku-Game

B. Team Member

Name	*UNI*	*Contribution*
Xin Huang	xh2510	*Equal Contribution
Ziqi Zhang	zz2881	*Equal Contribution