Planning and Reinforcement Learning based AI Tic-Tac-Toe Opponent

ZHAO Haosong (1730026153)

Abstract—**Tic-Tac-Toe (TTT) is one the most famous as well as simplest board games. Even though it is easy to play, the model given by TTT is quite complex and has far-reaching significance. On the one hand, by understanding the TTT model, we can use the same modelling way to solve the Gobang or even Go problem. On the other hand, TTT is a typical reinforcement learning problem that can give us the general ideas of this deep topic. As a well-modeled task, TTT solver can be trained using planning, a sequential decision-making approach. At the same time, it is indeed a Markov Decision Process (MDP) in Reinforcement Learning (RL). So, a more general solution is using the Q-learning Algorithm. In this project, two different Artificial Intelligent Tic-Tac-Toe Opponent (AITTTO) systems have been developed that can have a competition with the users in a real-time interaction manner based on planning and Q-learning, respectively. Considering the fact that TTT with two master players will always result in tie games, we can conclude these two AITTTOs are well-trained and can finish the task perfectly because both of them never lose in practical tests. And a competition with 10000 rounds between these two models is also hold and results in all tie games.**

Index Terms—**Markov Decision Process, Planning, Q-learning Algorithm, Reinforcement Learning.**

I. Motivation and Background

When we consider the nature of learning, it is easy to find that we can learn things from the interaction with the environment. When we communicate with others or drive a car, we can learn the skills from knowing the relationship between our behavior and the situation change. In other words, when we do these tasks, we are trying to use our action to influence the future environment. We can get the idea that interaction is the basis of learning. According to this thought, Reinforcement Learning (RL), a branch topic of machine learning, is born at this right moment. A lot of impressive work and researches have been done. One of them is the famous AlphaGo. Compared with other topics in machine learning, RL mainly solve the problem with a series of time-dependent decisions. In short, the input information of RL always contains time series. The learner or the problem solver generated by a RL can act extremely good at some tasks and make decisions just like human. Playing board game is one of them. With RL, a lot of AI opponents have been trained for entertainment as well as training the human chess players.

When we talk about the board game, it is worth to mention a really simple and easy game, Tic-tac-toe (TTT). By drawing up a table on paper, chalkboards even on sand, the old and the young can enjoy and master this game without a long-time learning process. Even though the game seems to be easy, but the TTT model is quite worthy of study. First, with a similar appearance with Gobang, the RL models based on TTT can be modified and become a Gobang AI player. But the difficulty to build a TTT AI player is much lower. It can be used as an entry-level problem. Besides, TTT is a perfect Markov Decision Process. The study based on it can give us a lot of inspire about how to solving a RL problem and make us familiar with some complex RL algorithms. Considering these reasons, this project has been developed. The work in this project mainly contains two parts which closely related two different Artificial Intelligent Tic-Tac-Toe Opponents (AITTTOs). In order to implement them, a planning algorithm based on TTT mathematical model and a modified Q-learning algorithm have been used. In the following sections, I will talk about all the technical details. In section two, I will give a brief introduction to the current related RL systems and useful algorithms. In section three, the theoretical knowledge used in this project including some lemma and statement proving will be listed. And then, the details about the whole system's implementation will be mentioned in section four. Finally, an analysis and a summary based on this project will be made in section five.

II. Related Works and Existing Techniques

As a RL topic, there are a lot of common applications and mature algorithms. First, as this project mainly talks about, RL has really good performance on playing games. In board games, Alpha Go is the most famous one after its competitions with LI Shishi and KE Jie. It almost shows the upper limit of human in playing Go. Besides the board games which have clear rules and limited possible scenarios, RL can also play extremely complex games very well, such as DOTA. The OpenAI Five have defeated a lot of human teams and got an overall 99% winning rate during its online test period. Due to its great ability in interacting with environment, it also makes a figure in man-machine conversation, Traveling Salesman Problem (Combinatorial Optimization), automatic drive, etc. Because the basic idea of RL, "try and learn", is quite intuitive, it is an "old and new" topic. Its history is old. Its applications are new. In general, the key point in solving a RL problem is to model it. Usually, all the tasks will be modeled as Markov Decision Processes (MDPs). To define an MDP, we need to extract five

.

key components from the task, they are A: actions, S: states, P: transition probability, R: reward, and γ: discounting rate. If the model is determined, we only need to choose a learning algorithm according to the action types (discrete action or continues action) and do the implementation. The remaining part in this section will talk about some common-used RL algorithms.

A. Q-learning

In MDP, we define an action-value function $q^{\pi}(s, a)$. Here, its value is the expected return starting from state s, taking action a, and following the policy π. It is easy to find that given all the possible states and its corresponding possible action, we can construct a big table containing all the action value, namely Q table. Q-learning algorithm solves the RL problem by trying to find the best Q table estimate and then make decisions based on it. For more details, please refer to section three, methodology.

B. SARSA

State-Action-Reward-State-Action (SARSA) algorithm also tries to construct the Q table and act according to it. The key different between it and Q-learning is that they have different updating methods. For SARSA, when it estimates the next state's action's value, it will also choose that action as the next step action. But for Q-learning, it estimates the next state's action only for Q table updating propose. When it reaches next state, it would choose the action with the max Q value in the updated Q table. Also, SARSA has an advanced variant called SARSA-lambda. It will keep the trace during the whole episode. When the reward is got, SARSA-lambda will update all the related Q value on the episode trace. The updating will also follow a rule that the action token near the reward matters more than the one token long time ago. So, it will speed up the learning process.

C. DQN

Deep Q Network (DQN) is a very popular RL algorithm. It is a combination of neural network and Q-learning. The two methods mentioned above are all based on Q table. It is quite intuitive that the Q table will become extremely large with the growth of state set and action set, and finally it would be hard to construct and store. So, a neural network is developed to take the place of the Q table. With the state or the action as input, the neural network can output estimated action value that can be used to make decisions. DQN is very powerful but relatively complex.

D. Monte-Carlo Policy Gradient

For the algorithm mentioned above, they made decisions based on action-value function (Q table). In other words, we know the value function and then define a policy to do the choice. But for policy gradient method, it directly does the choice based on the relation between actions and rewards. The good actions which bring reward will have a higher possibility to be chosen. The updating process are mainly focus on the policy improvement. By using this kind of method, if the actions are continuous or in a very high-dimensional space, it can still be solved without the action-value estimate.

E. Actor-Critic

Since now, we have talked about policy-based algorithm (Policy gradient) and value-based algorithm (Q-learning, etc.). When actions set gets bigger and bigger, we can only use policy gradient instead of value-based algorithms. But the updating in policy gradient takes place once for the whole episode. The learning process is slow and may converge to the local optimal. Actor-Critic algorithm solves these problems by combining these two types of methods. It has two parts, actor and critic. The actor will use policy gradient to get the possibility to choose action. And then the critic will use a value to measure the chosen action in a value-based way. Then the actor will update its policy according to critic's value. By using this method, Actor-Critic can achieve step-wise updating and become faster in learning compared with original policy-based methods. Its advanced variant, Deep Deterministic Policy Gradient, combined DQN into the Actor-Critic algorithm, makes it easier to converge.

III. Methodology

For this project, two AITTTO systems have been developed. The methods used to build them are planning and Q-learning. Planning is a very special case of sequential decision making. It is appliable when the model of how the environment works is given. Even though it is a method that is not need RL process, but it is the simplest way to solve the TTT problem. For analysis and comparison purpose, I used this method based on the TTT Optimization Decision Model

A. Tic-Tac-Toe Optimization Decision Model

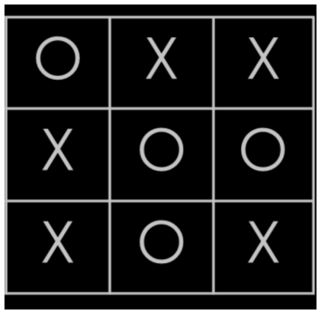


Fig. 1 Tie Game Result of TTT

The rule for TTT is: For two players, one uses the symbol "X" and the other use symbol "O" (namely, X and O for convenient). Each time, X or O can choose a blank space and marked with its symbol. Take this action one by one, repeatedly until one player occupied a whole row, a whole column or a whole diagonal and it will be the winner. If no one can do it and the board is full, we call it a tie game. In Fig.1, it is a standard tie game result of TTT. In order to model this game better, we call the cell in one row, one column and two diagonal center; the cell in one row, one column and one diagonal corner; and the cell only in one row and one column side. At the same time, TTT has a variant called Summation 15 (S15). The rule for S15 is: Two players choose a number from 1 to 9, one by one. Each number can only be chosen once. When one player's chosen numbers have three with a summation of 15, it wins. When all the numbers are chosen but no one have 3 numbers with summation 15. It is a tie game. TTT and Summation 15 are totally equivalent (See Fig.2).

4	3	8
9	5	1
2	7	6

Fig. 2 S15 in TTT format

The proof of their equivalent property is listed below:

1. There is a fact that using numbers in 1 to 9, only 8 combination of 3 numbers can have a summation of 15. It is equal to the number of the total of rows, columns and diagonals in TTT game. In other words, their number of winning ways are the same.
2. For 5, it is in 4 summation 15 combinations (S15Cs). For 2,4,6,8, they are in 3 S15Cs; and for 1,3,7,9, they are only in 2 S15Cs. It is obvious that they have a one-to-one mapping with the center, corners and rows in TTT.
3. 1 to 9 can be putted in to the TTT board and according to the one-to-one mapping, the summation of each row, column or diagonal is 15. So, marking a position on TTT board, equals to choosing that corresponding number in S15.

So, the two games are equivalent.

Besides, there is a fact that, if two players know the TTT game perfectly, they will always result in tie game. The proof of this statement is quite intuitive. First, even though finding all the possible result of TTT needs a lot of time, but it is still possible. No matter which move the player that make the first move choose (assume X), the second player can always lead the game to a tie game result. At the same time, we can think about the fact that winning the game at least needs three steps. And there are only 8 possible win methods. Once a player occupied any position in one-win method, the other player will never have chance to win in that way. The three steps that lead to a win give the opponent enough time to destroy its win way. So, if two players know TTT well, they will always result in a tie game.

Also, in literature, the action priorities are already given:

Self-win>prevent-win>corner>center>side. If one move can directly make the player win. It is obvious the best choice. Besides, if there exists a move that will lead your opponent win directly, you must occupy it in your turn to prevent the opponent's win. Then, a counterintuitive fact is that the corner is more value able than the center. As mentioned above, during the literature, when we draw all the possible results, it is the fact that corners have more value than the center. Some common winning pattern with first occupying the corner is shown in Fig.3.

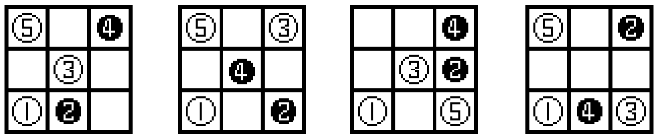
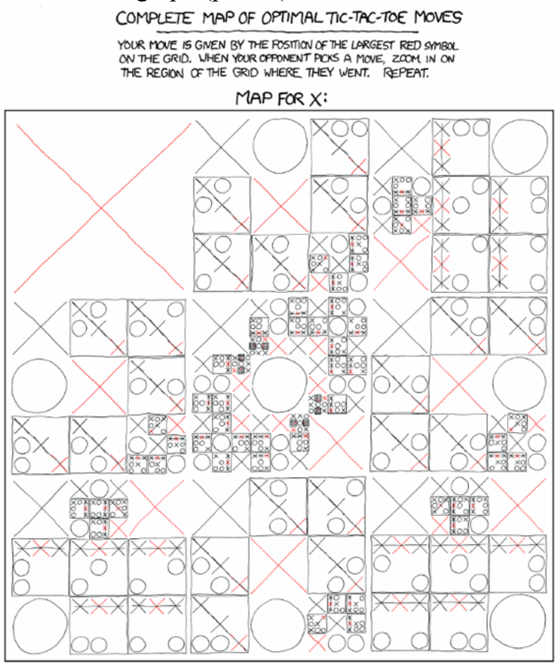


Fig. 3 Winning Pattern with Corner First

Also, you can think that when you have a center, you winning methods contains one row and one column that ask you to add two less valuable side positions. And the other two diagonal winning methods ask you to occupy two corners at the same time. Choosing corners in one diagonal to win is almost impossible because your opponent will stop you and occupy these valuable positions. Choosing sides also have a problem that sides are low value place. Choosing them at an early stage is wasting chances. You can also refer to the Fig.4 to see the all possible result graph (partial).



COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES
YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.
MAP FOR X:

Fig. 4 TTT Possible Result (Partial)

B. *Modified Q-learning Algorithm*

The second AITTTO system based on Q-learning algorithm. It is worth to mention that TTT has perfect full observability. And it is a very standard MDP. Consider the Markov property: $P(s_{t+1}|s_t) = \mathrm{P}(s_{t+1}|\mathrm{h_t})$ and $P(s_{t+1}|s_t, a_t) = \mathrm{P}(s_{t+1}|\mathrm{h_t}, \mathrm{a_t})$. In Fig.5, we can see three X and 2 O. But the order of them is not necessary anymore. Which X or which O is putted first will never influence the future game. With the current State we can get all the information. Besides, another good property of TTT is that its P transition possibility given A is listed below:

$$P(s_{t+1}|s_t, a_t) = 0 \text{ or } 1$$

It is because when the action, a move is made. The next state is already decided. The P here will never be a possibility but a fact. So, we can directly do the value computation instead of using its expectation.

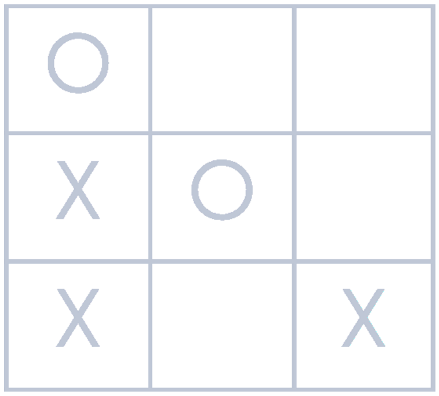
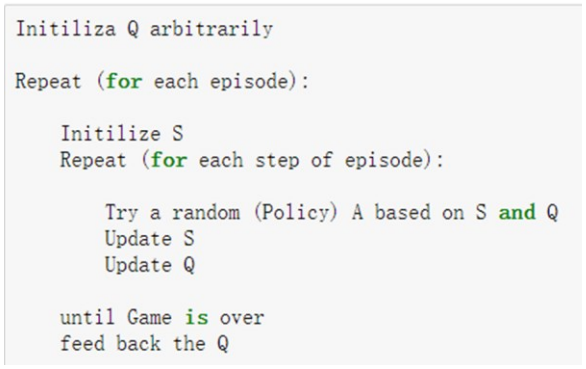


Fig. 5 TTT State Sample

The modified Q-learning Algorithm is listed in Fig.6.



```
Initiliza Q arbitrarily

Repeat (for each episode):

    Initilize S
    Repeat (for each step of episode):

        Try a random (Policy) A based on S and Q
        Update S
        Update Q

    until Game is over
    feed back the Q
```

Fig. 6 Modified Q-Learning Algorithm

The novel ideas in this algorithm is the updating formula and also the Q feed back updating step. First, consider the updating formula in original Q-learning algorithm:

$$Q(S,a)^{new} = (1-\alpha)Q(S,a) + \alpha(R(s,a) + \gamma \max_{\hat{a}\in A} Q(\hat{s},\hat{a})$$

We can see that we need to use next steps maximum Q value to do the updating of the current Q. The problem is that in TTT, we follow a manner that one by one make a move. It means that next step is made by the opponent. So how can we measure the next step's max Q? At the beginning I come up with an idea that think the opponent completely follow a stochastic process. It means the opponent will randomly choose a position to move. But this idea conflicts with the fact that usually the game will be a tie game with two master players. So, I consider the fact that the max Q means the maximum action-value for me. If it is my opponent turn, I would expect it to make mistakes and loss the game. So, if the next step is the opponent turn, why don't we use its minimal Q value to estimate our maximum action-value? In practice, it does make sense and the model works well using this Q value updating formula.

Besides, the feed back Q operation mainly consider the fact that TTT will always come to an even in 9 steps. So, 9 steps are quite small for us to store its trace. And when we get the result. It means that the final reword is known and there is no next step Q value. So, we can get an accurate Q value at last step. If we feed forward and set the reward to be 0 if the game does not generate a winner. Then all the Q value in the trace can be updated with the final step more accurate Q in a feeding backward approach. This operation can speed up the learning rate.

IV. AITTTO Systems Introduction

A. *Planning AI player*

For this AITTTO, for each step, it just makes a move according to the action priorities. With the state as input, it tries to take the highest priority action and then move to the relatively lower priority action.

B. *RL AI player*

For this AITTTO, at the beginning, it has a learning process that use 10000 episodes to do the Q-learning algorithm. When the learning process ended, with the Q-table, it just makes the decision according to the Q table. In other words, make a move with the max Q value. The total training time is about 0.9s. It is acceptable.

V. Conclusion and Future Work

Consider the fact that two master TTT players will always end in a tie game. A competition between these two AITTTOs has been hold. For 10000 rounds, all of them reach the tie games. And I also design some functions to interact with users. With these functions, everyone can play and try to beat my AITTTO. But since now, nobody has won them. So, I can conclude that both of them are good AITTTOs and the RL algorithms that I used are both correct and suitable for this project task.

For the improvement. Because of the modified Q-learning algorithm, the Q value is based on the agent itself. And it has a very strong concept of opponent. So, the AITTTO based on Q-learning algorithm can only move first. The order setting for this AITTTO can be added in the future. It is possible to hold all the iteration information of Q table. A good visualization can be applied to illustrate the model improvement process.

VI. Notes

For the codes of this project, they are almost 100% hand-typing. And all of them have detailed comment. Please feel free to have a look at them and play with these two AITTTOs. Thank you.