2023 Foundation of Artificial Intelligence Final Project
Texas Hold'em

詹子慶
ID: b10902006
National Taiwan University

Abstract

This project is to implement an artificial intelligence(AI) that plays Texas Hold'em. I approached the problem in three ways, the Game Theory and Probability approach, the Dynamic Programming approach, and the Machine Learning approach. Among them, the Dynamic Programming approach performed the best, with an average **77**% win rate competing to TA baselines.

1. Texas Hold'em Rules

The poker game consists of 20 rounds and only two players, each player starts with 1000 in their stack. In each round, the small blind bets 5 and the big blind bets 10. Whoever has more money after the 20 rounds wins.

2. Observations

2.1. The Win Line

The first thing that caught my eye is the rule "Whoever has more money after the 20 rounds wins." This is crucial to the extent that it changes the whole game. If someone has earned a lot of money at some point, he can fold the rest of the rounds and is still **guaranteed to win**. I call this the "win line", if anyone crosses the win line at any point in the game and recognizes it, he wins. The win line:

$$\begin{cases} 1000 + \frac{15}{2}(20 - r) & \text{if } r \text{ is even} \\ \begin{cases} 1005 + \frac{15}{2}(19 - r) & \text{if } i = 0 \\ 1010 + \frac{15}{2}(19 - r) & \text{if } i = 1 \end{cases} & \text{if } r \text{ is odd} \end{cases} \tag{1}$$

where r is the round number, i is the index of our seat.

2.2. All In

After we talked about the win line, it is not hard to see that "raise to the win line" is strictly better than "all in". If we all in, we win or lose the game entirely depending on this round. However, if we raise to the win line, winning this round also guarantees a win to the game, but we might be still in the game after losing this round. In conclusion, whenever we want to all in, raise to the win line.

3. Approach 1: Game Theory and Probability

3.1. Why Game Theory and Probability

The game theory applies when there are several players, and each player tries to make choices each turn to maximize or minimize their given objectives. In game theory, each player might even adopt "a probability distribution of choices" for a response, for example, one might adopt "50% fold and 50% call" in some circumstances. Being able to respond by a probability distribution involves conditional probability, which also needs to be taken into account when calculation.

It is proven that every game has a Nash Equilibrium [1], so my goal is to calculate the Nash Equilibrium and find the optimal probability distribution of choices in every state in the game.

3.2. The States

The states are described by:

- The round number
- The money we have
- The street we are in
- Whether we are the first move of this street
- The money we and our opponent have contributed in the pot
- The probability distribution of our hole cards and our opponent's hole cards
- The probability of winning for each pair of our and our opponent's hole cards

The probability of the hole cards is a crucial part of this approach: the hole cards of our opponent are unknown, so there is a probability distribution for them apparently, but our own hole cards are visible to us, why should we consider the probability of our own hole cards?

The reason is because the ability of **bluffing**. The probability of our hole cards is seen from a third-person point of view, so if we have a probability of having a dominating hand, a full house for example, we can still bluff to have it, regardless if we really have one.

3.3. The Calculations

Every state has an expected win rate, and there is an optimal probability distribution of choices according to it. We can list some equations to calculate the win rate of each state.

1) The probability of the choices in every state must add up to 1, according to probability.
2) The distribution must be the "best" distribution in every state, meaning there must not exist any other distribution that results in a higher win rate if we only alter the distribution one state at a time, according to the definition of Nash Equilibrium.
3) The expected win rate must match the sum of all win rates of the next states times their probabilities, according to probability.

Note that the probability distribution of hole cards must also be taken into account. For example, if I raised in the preflop, I would have a higher probability of having pocket aces than an offsuit 27, so my hole cards distribution must also be changed after I raised in the preflop. To achieve this, we need to calculate the conditional probability of each former state and adjust the distribution accordingly.

3.4. Issue: Complexity

In theory, it can be computed, but as you might notice, this is a "very" complicated task. The first problem is, I do not know how to calculate the expected win rates and the optimal probability distributions, as this involves a **recursive** definition. Being recursive, we cannot use the dynamic programming method, but only iterative methods such as gradient descent. However, iterating costs too much time, which is far beyond the time limit of 5 seconds. This leaves us to pure math, precalculating with matrices using linear algebra. This is way off my abilities for math, so I found another simpler way to implement it.

4. Approach 2: Dynamic Programming

4.1. Reduction from Approach 1

Game theory and probabilistic approaches might be accurate, but it is too complicated. Therefore, I reduced the description of states, and added more constraints to the choices we can perform.

One can only choose to "raise to the win line" or "fold" in the preflop, only informing the round number, the money we have, and our starting hand, i.e. hole cards. If "raise to the win line" is chosen, then he must not fold in this round, that is, he is **"betting" on this round** and is very likely to be the round determining the outcome of the game.

4.2. The Starting Hand

To avoid calculation, I looked for existing data. The most famous chart I can get is the win rate for each starting hand [2]. This chart not only gives me a much more accurate feeling of how much win rate is determined in the preflop (35% win rate in the worst case and 86% win rate in the best case!), and also enables me to calculate a function $f(x)$, which means: "x percent of the time, we will get a starting hand with a win rate higher than $f(x)$".

4.3. The Calculations

`DP[r][m]` is defined as the win rate when one starts as the big blind, where r is the round number and m is the money he have. The dynamic programming transition is as follows:

$$\texttt{DP[r][m]} = \begin{cases} \max_x(\min_y(1-x)\texttt{DP[r+1][m-5]} + \\ x(1-y)\texttt{DP[r+1][m+10]} + xy \cdot g(x,y)) & \text{if } r \text{ is even} \\ \min_y(\max_x(1-y)\texttt{DP[r+1][m+5]} + \\ y(1-x)\texttt{DP[r+1][m-10]} + xy \cdot g(x,y)) & \text{if } r \text{ is odd} \end{cases} \tag{2}$$

where

$$g(x,y) = \frac{h(x)(1-h(y))}{h(x)(1-h(y)) + h(y)(1-h(x))} \tag{3}$$

where

$$h(x) = \frac{\int_0^x f(t)dt}{x} \tag{4}$$

$h(x)$ represents the expected win rate given that the starting hand is among the top x percent. $g(x,y)$ represents the probability of winning when my starting hand is among the top x percent and my opponent's starting hand is among the top y percent. `DP` is calculated by the game tree of each round, where x and y are the probability that "raise to the win line" is chosen by me and my opponent respectively.

4.4. The Implementation

Each state in `DP` is calculated by iteration over 20 x values and 20 y values evenly distributed from 0 to 1. After the dynamic programming is done, we can use it in every round in the game. In every round, I compare the win rate of my starting hands and the following value:

$$\begin{cases} f(x \text{ of } \texttt{DP[r][m]}) & \text{if I start as the big blind} \\ f(y \text{ of } \texttt{DP[r][2000-m]}) & \text{if I start as the small blind} \end{cases} \tag{5}$$

in the preflop to determine whether I should "raise to the win line" or "fold". If we both made it through the preflop, do not fold, as it violates the probability of winning we calculated.

4.5. Some Optimizations

In theory, we have a win rate according to `DP`, but in practice, we can do better in several ways.

1) Call rather than fold when calling does not cost additional money. In some cases, for example, the big blind, one has already more or equal money in the pot among the players. If one want to fold in those cases, just call for free instead, in other words, "check".

2) Continue to raise after raising to the win line. This is an **exploit to my opponent's "maybe" mistakenly implemented AI**. After raising to the win line, I am "betting" on this round, so I want to maximize the win rate of it. Raising continuously may increase the chances of my opponent choosing to fold, which is even better than comparing hole cards after the river.

3) Raise to the win line or all in after crossing the "lose line". Just like how we defined the win line, we can also define a lose line similarly. This is also an exploit to my opponent's AI. If my opponent overlooked the fact that he has already won when I cross the lose line, I might be back in the game by winning another round with raising to the win line or all in.

4) Consider the community cards after the preflop. We said to continue calling or raising regardless of the community cards after the preflop due to the calculation of probability. This is rather absurd because why don't we just take a look at the cards that provide information while they are just there for us to look at? So I implemented the most basic way to determine whether this is a good flop or not: count the number of pairs.

4.6. Performance

I tested the TA baselines 700 times each.

TA baseline	Win Rate
1	86%
2	81%
3	100%
4	56%
5	62%

Although the numbers does not look very promising, I still think this is an acceptable win rate. That is because many of those losses is resulted by losing the comparison after the river, which is unavoidable, as this game is fundamentally of game of luck!

4.7. Issue: After Preflop

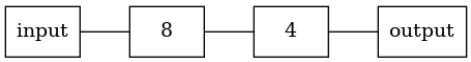
As we have mentioned, we count the number of pairs after the preflop. Why do not go any further? Maybe we could lower the possibility of **losing in the river**! We may notice that the three most common rankings we may have is one pair, two pairs, and three of a kind. All of those can be detected by counting pairs. Other higher rankings can be neglected as they happen too rarely.

However, sometimes they still occur. To take them into consideration, we could add more `if else` statements, or we could do this in a much cleaner way: machine learning.

5. Approach 3: Machine Learning

5.1. The Configurations

I used the **reinforcement learning** algorithm with the `keras` package [3]. I chose reinforcement learning over other machine learning methods because there is no existing data we can learn from, only trial and error is available, which exactly fits the concept of reinforcement learning. I used deep learning and the neural network structure is as follows:



Picture 1. Neural Network Structure

where the input is the state, the output is the action and reward. I used the Adam optimizer with learning rate 0.01 and the Huber loss function.

5.2. States, Actions, and Rewards

The states are described by:

- The round number
- The money we have
- The street we are in
- Whether we are the first move of this street
- The money we and our opponent have contributed in the pot
- Our hole cards
- The community cards

There are three actions available for each state: fold, call, or raise to the win line. All actions in the same round is stored and later rewarded by the money we gain or lose this round.

5.3. Performance

I tested the TA baselines 700 times each.

TA baseline	Win Rate
1	90%
2	67%
3	100%
4	25%
5	11%

Very obviously, the machine learning model is not performing well. It may seem to have a high win rate against baseline 1 and 3, but that is because those baselines are too weak. For stronger baselines such as 4 and 5, it loses most of the time.

5.4. Issue: Not Enough Rounds

The main problem of the machine learning approach is the **lack of data**. We all know from various researches, machine learning cannot perform well without enough data. This is especially true for reinforcement learning, because the reward is given to multiple actions, and with the probabilistic nature of the game, it is even harder for the learning process to converge.

Another reason is the **lack of time**. With almost every TA baseline AI thinking up to 5 seconds, it is really time consuming to train. To test 1000 times against a single baseline AI, it takes one whole day without stopping to finish the execution. This is consuming far more time than what we have.

Being able to learn how our opponent is responding and guess his hole cards might be an advantage of machine learning. However, it is even harder to learn this information while in play, because there are only 20 rounds in each game, which is far from enough.

6. Conclusion

6.1. Comparisons

The most complete model is the Game Theory and Probability approach, if we are able to calculate the expected win rate and the optimal probability distribution of choices for each state, it will be "the optimal solution". However, it is **too complicated** so that I failed to implement it, and only stayed as a "thought experiment".

In contrast, the model of the Dynamic Programming approach is much simpler and implementable. It preserves the important features of the game, for example, being able to fold or all in by some optimal probabilities. However, this model may be **too simple**, for it does not take our opponent's actions into account, and does not perform well after the preflop. It is a model with low implementation difficulty, and rely heavily on probability.

The Machine Learning model is an attempt to use reinforcement learning to capture the important features of a game state, including the community cards after the preflop, and act accordingly. However, the major downside of this method is the **inability to fully train** the model until it converges. A reinforcement model that has not yet converged is very likely to perform below expectations.

6.2. Final Submission

The final model I choose to submit is the Dynamic Programming approach. Although it relies heavily on probability and might not win as expected during judging, it still has tolerable results during testing and is worth a try.

References

[1] https://cs.brown.edu/courses/cs295-8/nash.pdf

[2] https://imgur.com/Cnn2is2

[3] https://keras.io/examples/rl/actor_critic_cartpole/