Reinforcement Learning for Optimal Strategy in Texas Hold'em Game

Ye Chen

Department of Electrical Engineering
Stanford University
Stanford, USA
yechen9@stanford.edu

Jamie Song

Department of Computer Science
Stanford University
Stanford, US
jamiesxy@stanford.edu

March 20, 2023

***Abstract*—This project models the popular poker game Texas hold'em as a partially observable Markov decision process (POMDP) and proposes to build an optimal strategy for a single hold'em game using two reinforcement learning algorithms, Q-learning and Sarsa. Our dataset for Texas hold'em game records is generated with stated assumptions of the hold'em game in real-world settings, while we abstract the game into 6 states and 52 states respectively. We trained the reinforcement learning models based on the two simulated datasets and crossly compared the performance of our model. We concluded that the 52-state model account for the hold'em game better based on quantitative results and qualitative analysis using the strategy and experience developed by human players. To summarize, this project has achieved our goal of suggesting an accountable policy for a single hold'em game that performed significantly better than random actions.**

***Index Terms*—Hold'em optimal strategy, POMDP, Q-learning, Sarsa**

I. Introduction

Texas Hold'em is a popular poker game around the world, and the main goal for this project thus the paper is to solve the decision-making problem involved in playing this game. As players, how to choose whether to fold from the game, to stay in the game with no change in number of chips, or to raise the number? Does the luck plays an important role in making the decision? And how the player's action should change based on the change of other players? We want to study how the expected number of chips gained from the game changes with the procedure of the game, and how to maximize it. We've modelled the question into two MDP problems, one embodies not looking at the cards, and the other embodies looking at the cards before making the decision. Non-agent player behaviors are encoded with several parameters. We can also alter the ordering of the players to see whether the seating matters, and if there are any changes when sitting in different positions. We've implemented both Q-learning [4] and Sarsa [3] algorithms. For each MDP problem, we compare the results of the random policy and the two policies generated by the two algorithms; we also compare the changes of our learned policy when the non-agent players are more likely to raise, check, or hold. After restricting the number of players to be 4, we also compare the policies and expected rewards when the agent locates at each of the 4 positions. Two simulators, the policy files, state transition csv files, and the learners, are all public on our Github repo (https://github.com/cynthiayechen/AA228-FinalProject-HoldemGame). We've recorded our data in the tables, together with our analysis on those data, in Section IV.

II. Texas Hold'em

A. Original Game

The website [2] states the game process. Here we paraphrase the basic rules of the game:

Normally, the game uses a standard 52-card poker deck, including numbers 1 to 10, Jack, Queen, and King, in all four suits (heart, diamond, club, spade).
Within each game, the players' goal is to construct five pokers out of seven, to make the combination be the greatest. Of the seven cards, two are private, and five are public. The players are free to use any of the combinations. The comparison rule will be described later.

At the beginning of each game, each player receives two cards, and they're secret – no other players know for sure what these two cards are. Then it is the first betting round. The detail of betting round will be described later. Then, the dealer shows three community cards, which are visible to all players. Then it's the second betting round. Then, the dealer shows the second community card. Then it's the betting round again. Then the dealer shows the last community card. Then, it's another betting around. After all betting actions are completed, the game enters the showdown phase. The details of the showdown will be described later.

There is a player with a dealer button, who is responsible for dealing the cards during that game. This button rotates clockwise around the table per game. The player sitting immediately left to the dealer is the "small blind", forced to put one chip into the game at the very beginning. The player sitting immediately left to the "small blind" is the "big blind", forced to put two chips into the game at the very beginning.

During each betting round, the first player to act is the one sitting immediately left to the "big blind". Each player has three potential actions: call, raise, and fold. Call refers to putting chips into the game, so that the number of chips putting into the game by that player reaches the minimum of number of chips by other players so far. Raise refers to putting more chips into the game, so that other players have to put at least this amount of chips to stay in the game. Fold refers to throwing away the cards and quitting the game. The previous chips putting into the pool are a sunk cost of this player, and can no longer be taken out.

During the game, if all but one player has folded, then the remaining player wins the game, and the later procedure no longer needs to be continued. Otherwise, during showdown, each player remaining in the game shows their two private cards, and the player with the greatest combination wins the card. The winner gets all chips in the pool.
Here is the order of combinations:

1) **Royal Flush**: Five cards of the same suit, ranking from 10 to A.
2) **Straight Flush**: Five cards of the same suit, and forms a rank.
3) **Four of a Kind**: Four cards with the same rank.
4) **Full House**: Three cards with the same rank, two other cards with the same rank.

5) **Flush**: Any five cards with the same suit.
6) **Straight**: Any five forming a rank.
7) **Three of a Kind**: Three cards with the same rank.
8) **Two Pair**: Two cards with the same rank, two other cards with the same rank.
9) **One Pair**: Two cards with the same rank.
10) **High Card**: Any five cards not matched to any previous patterns.

B. Previous Works

There are many previous works on the problem.

1) Texas Hold'em

Fredrik once worked on the problem, and stated his results in [1]. His results show that value-based reinforcement learning, such as TD-learning and Q-learning, is not applicable to games of imperfect information, such as Texas Hold'em. However, he has a very different problem simplification strategy as ours. For example, he only considers the two-player situation, without the consideration of big or small blind. Besides, he assumes both players are equally smart, and has an equal pace of learning the best strategy. In contrast, we want to focus on learning how a single player (our agent) should act. In addition, we are also interested in seeing how fast the agent can learn, based on different strategies adopted by other non-agent players, on the raising step, and on which player the agent is. Meanwhile, this paper does show some interesting perspectives on how we can analyze the game.

2) Algorithms

The algorithms we used for reinforcement learning in this project are the Q-learning method adapted from Watkins [4], and Sarsa adapted from Rummery [3].

Q-learning in this paper is presented by Watkins as an incremental dynamic programming method for reinforcement learning to control a Markov decision process (MDP) and showed the algorithm's convergence.

Sarsa is presented by Rummery with another name, on-line learning algorithms, to provide a reinforcement learning algorithm that are more robust to the choice of training parameters than standard Q-learning updates than Watkins.

We utilized these two algorithms in our project for learning the optimal policy for our agent to act in a single Texas Hold'em game. The exact implementation of these two algorithms and the reason why we choose these two algorithms specifically is justified in Section III C.

C. Problem Simplification

Here are the simplified assumptions we have made when implementing the simulator:

- The total number of players, including the agent, is 4. That is, there is one big blind, one small blind, and two other players. The agent can be any of the four players.
- When the action is to raise, there is a fixed number of chips the player can raise.
- We do not consider the "all-in" circumstance, which means running out of chips. We always assume all players have enough number of chips, feasible of raising, checking, or folding.
- We assume the action chosen by all non-agent players are a random strategy formed by 5 parameters: n_1,n_2,$[p_1]$,$[p_2]$,$[p_3]$, where n_1 and n_2 divide the ranking into 3 disjoint intervals, consider them as 3 levels. At each level, there is a probability of raising, checking, and folding, covered in the lists $[p_1]$, $[p_2]$, and $[p_3]$.
- We do not specifically consider bluffing. The bluffing probability for non-agent players is already contained in their behavior definition.

III. Approach

A. Problem Formulation

We originally modeled the Texas Hold'em game as a MDP with the following model:

- Action Space:
  The action space of this model has size 3, where each player can perform an action out of the following three: *Fold*, *Check*, and *Raise*.
  Although in real life, the player can choose to raise arbitrary chips, as described above, we simplify the rule for action *raise* specifically for our model, that each *raise* can only be raising x chips, where x is a parameter, default to be 5.
  The rules for *fold* and *check* are exactly the same as those in the real-life Texas Hold'em game.
- State Space:
  1) **Compulsory bets**: big blind & small blind prior to any cards are drawn
  2) **Pre-flop**: the agent is given the two hold cards
  3) **Flop**: three community cards are faced up
  4) **Turn**: the fourth community card is faced up
  5) **River**: the fifth community card is faced up
  6) **Fold**: the agent chooses *fold* in any of the states 2, 3, 4, or 5
- State Transition:

$$\begin{tikzcd}
& 1 \arrow[dl, "\epsilon"'] & & & & & \\
2 \arrow[rr, "check", "raise"'] \arrow[drrr, "fold"] & & 3 \arrow[rr, "check", "raise"'] \arrow[dr, "fold"] & & 4 \arrow[rr, "check", "raise"'] \arrow[dl, "fold"] & & 5 \arrow[dlll, "fold"] \\
& & & 6 & & &
\end{tikzcd}$$

  Here the ϵ means it's an autonomous transition with no specific action (by the agent) needed.
- Reward Model:
  Our reward is a function of s (previous state), a (action), and sp (resulting state).
  We denote the total number of chips betted by the agent during the game till computing to be C. We denote the total number of chips in the pool during the game till computing to be P.
  If the agent is the only remaining player, then the reward in this case is P.
  If the agent chooses to fold anytime during the game, then the reward is $-C$.
  If the agent stays in the game until showdown, and turns out not winning the game, then the reward is $-C$.
  If the agent stay in the game until showdown, and turns out winning the game, then the reward is P/num, where num is the number of players sharing the winning status.

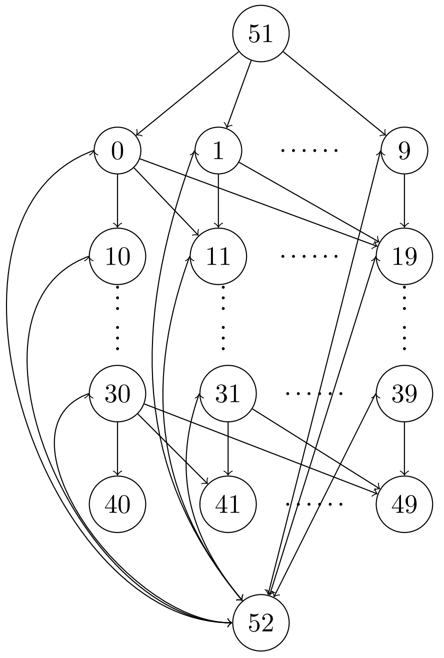
Later, we decided to add more information disclosed by the states, so we reformulated the problem as an MDP with some modifications on the states and the resulted state transitions, but no change in the action space and the reward model.

- State Space:

51) **Compulsory bets**: same as above
0)-9) **Pre-flop**: same as above
10)-19) **Flop**: same as above
20)-29) **Turn**: same as above
30)-39) **River**: same as above
40)-49) **After river**: the state to compute the reward of the agent after all betting rounds
52) **Fold**: same as above

Here from 0) to 49), the least significant digit indicates the current ranking the player has.
When at least 5 cards are known, select the maximum rank out of any five cards.
When only 2 cards are known, we found out a chart online to indicate the ranking of the two cards.

- State Transition:



For the sake of clarify, the labels are not shown in the graph, but here is the transition function:
- For all transitions $f(51,x)$, where $x \in [0,9]$, $f(51,x) = \epsilon, p_{51,x}$.
- For all transitions $f(x,y)$, where $x \in [0,39]$ and $y \in [x+10, (x//10+1)*10+9]$, $f(x,y) =$ raise,$p_{1_{x,y}}$, check,$p_{2_{x,y}}$.
- For all transitions $f(x,52)$, where $x \in [0,39]$, $f(x,52) =$ fold,1.

B. Our Approach

We applied two model-free reinforcement learning methods to this project, Q-learning [4] and Sarsa [3], where each of the methods incrementally estimates the action value function Q(s, a) from samples (rewards) and does not require a state transition function.

For both Q-learning and Sarsa, the action value function Q is stored as a table of the size number of total states times the number of total actions, which is a (6×3) / (52×3) matrix. The learning rate alpha is 0.10 for the Hold'em game, and the discount factor is set to 0.95.

The Q-learning equation used for updating the action value function $Q(s,a)$ is:

$Q[s,a] \leftarrow Q[s,a] + \alpha * (r + \gamma * maximum(Q[s',:]) - Q[s,a])$.

The Sarsa equation used for updating the action value function $Q(s,a)$ is:

$Q[s,a] \leftarrow Q[s,a] + \alpha * (r + \gamma * (Q[s',a']) - Q[s,a])$.

The main difference between the two approaches is that Sarsa updates the action-value function Q using the actual next action a' taken by our exploration policy, while Q-learning updates Q with maximizing over all possible actions.

After computing the action value function Q, we generated our policy by acting greedily based on the estimated action-value function. If the state in the problem does not have actions and rewards associated with them in the small data set provided, we assign a random action to it.

Our project generated the data for Texas Hold'em game using a simulator and we train our model offline for optimal policies. Besides, we have also included an implementation for Q-learning combined with the exploration strategy ϵ-greedy in our code, originally for the sake of convergence of our action-value function.

C. Appropriateness of Approach

We applied two model-free reinforcement learning methods, Q-learning and Sarsa, as our trainers for the optimal decision/policy that our agent should take in a single Hold'em game.

One of the reasons that we prefer model-free over model-based methods is that our problem, the series of optimal decisions to make during a single Texas Hold'em game, is a high-dimensional optimization problem. The problem contains up to 52 states in our current model, while the 52-state model is already a simplified version of the actual game as stated in our assumption.

Therefore, we choose to learn from the action value in our simulated game data directly instead of building explicit representations of the transition and reward models. The main reason we did so is that building explicit transition and reward models for Texas Hold'em will introduce assumptions that cannot be fully justified and thus may not be applicable to real-life circumstances, which will impact our model's accountability negatively.

IV. Analysis and Results

Qualitatively, our generated Q-learning and Sarsa policy for the 6-state dataset is to suggest our agent raising for the flop and turn round, and checking for the river round. This makes sense in the actual Texas Holdem game setting, as the flop and turn round are gathering more information about the current pool and how good the cards our agent currently holds are, while the last round our agent would like to check and stay in the game to see if the agent will win.

For the 52-state dataset, the Q-learning and Sarsa policy is naturally more complicated and sophisticated, as our policy will tend to instruct our agent to raise for the cards that are in the pattern of the most winnable combinations, to check for the cards that are moderately winnable, and to fold for the card combination that is possibly losing. The Q-learning and Sarsa policy also abide the same tendency to check/raise for previous rounds (flop and turn), and more likely to be conservative for the river round (fold/check), which is also a typical real-world Texas Hold'em strategy.

We qualitatively compared the randomly generated policies with Q-learning and Sarsa policy for both the 6-state simulated dataset and the 52-state simulated dataset in Table I below, with the final rewards computed for each different policy under the corresponding dataset.

To analyze the table above, we start with comparing policy-wise for the same dataset. For both the 6-state and 52-state datasets, the random policy performs the worst, as expected. The Q-learning and Sarsa both perform significantly better than the random policy on the same dataset. For the 6-state, Q-learning has a slightly better performance than Sarsa policy, while for the 52-state, Sarsa has a

	Random policy	Q-learning	Sarsa
6-state dataset	-9.8216	-8.734	-8.5545
52-state dataset	-0.1938	12.2089	7.9463

TABLE I
THE COMPUTED REWARDS COMPARISON FOR POLICIES IN DIFFERENT DATASETS

slight advantage than Q-learning. We attribute such a difference to the fact that Q-learning chooses policy among the possible actions that maximize the action value, while Sarsa chooses policy using the actual next state explored. The total number of games we used to train 6-state dataset and the 52-state dataset are the same (10,000 games in total), and it is easier for the 6-state dataset to be able to explore an optimal solution than it the 52-state dataset, meaning that Sarsa will possibly perform slightly better at the 6-state dataset.

We also compared our computed rewards across the dataset. For the 6-state dataset, the rewards obtained by the Q-learning policy and Sarsa policy are smaller than the rewards received by corresponding policies in the 52-state dataset. By manually analyzing the policies, we found that our agent never folds in the 6-state Q-learning and Sarsa policies and does fold in certain states in the 52-state policy. This means that our 6-state policy suggests that theoretically, the rewards earned in-game will be maximized if the player never folds. However, in reality, the amount of chips the player is capable of buying is not infinite, and the 6-state policy is therefore neither generalizable to real-life circumstances and not the most reward-winning policy. Therefore, our 52-state policies are numerically more rewarding than 6-state policies and actually more accurate and accountable optimal policies that model real-life circumstances better.

Afterwards, we start the analysis on different non-agent behaviors and the position of the agent.

To quantify the behavior of the non-agent behaviors, we used 5 parameters. Since totally there are 10 rankings of the cards, the first two parameters are the ones dividing the 10 ranks into 3, and the latter 3 parameters each contain 3 parameters to quantize the probability to raise, check, or fold. Then, we divided all behaviors into 3 categories: more likely to raise(0), more likely to check(1), and more likely to fold(2).

There are totally 4 possible positions for the agent: 0, 1, 2, and 3. Sitting at 0 means the first player to act; sitting at 1 means the second player to act; sitting at 2 means the small blind, also being the third player to act; sitting at 3 means the big blind, also being the fourth player to act.

dataset vs non-agent	0	1	2
6-state dataset	-11.1843 / -11.1788	-8.734 / -8.5545	16.9015 / 16.7458
52-state dataset	13.5105 / 8.2065	12.2089 / 7.9463	22.0721 / 28.9641

TABLE II
THE COMPUTED REWARDS COMPARISON FOR NON-AGENT BEHAVIORS WHEN AGENT=0 IN DIFFERENT DATASETS

dataset vs non-agent	0	1	2
6-state dataset	-8.708 / -6.1927	-6.7051 / -5.8438	7.2778 / -0.7326
52-state dataset	15.6016 / 5.3238	16.5712 / 5.1629	25.8674 / 28.0115

TABLE III
THE COMPUTED REWARDS COMPARISON FOR NON-AGENT BEHAVIORS WHEN AGENT=1 IN DIFFERENT DATASETS

The above four tables contain the data for when the agent is 0, 1, 2, or 3, respectively. Each column represents the type of behavior of non-agent players, 0 representing them more likely to raise, 2 representing them more likely to fold. Each row represents the data generated by each model. Each cell contains two values separated by the '/' symbol, with the first one meaning the expected reward trained from Q-learning, and the other one meaning that trained from Sarsa.

dataset vs non-agent	0	1	2
6-state dataset	-8.9619 / -9.0678	-8.5559 / -8.888	-9.0909/-8.7814
52-state dataset	6.9997 / 7.8479	7.8795 / 8.2849	15.2465 / 15.1735

TABLE IV
THE COMPUTED REWARDS COMPARISON FOR NON-AGENT BEHAVIORS WHEN AGENT=2 IN DIFFERENT DATASETS

dataset vs. non-agent	0	1	2
6-state dataset	-9.2188 / -9.1247	-8.999 / -9.1118	-9.1065 / -9.1065
52-state dataset	16.1657 / 12.0	12.708 / 10.5624	16.0579 / 11.6731

TABLE V
THE COMPUTED REWARDS COMPARISON FOR NON-AGENT BEHAVIORS WHEN AGENT=3 IN DIFFERENT DATASETS

For each table, we trained the data on both models, using both Q-learning and Sarsa, for the non-agent type being 1, and applied the resulting policy on all non-agent types. This is because in reality, the players know their own position, so that they could train the policy accordingly; in contrast, they might not know the type of other players, so they could assume the others to be the most general type, which is more likely to check (category 1).

To note our result, we found that in all starting positions of our agent and in all non-agent behavior categories, the 52-state policy performs significantly better than the 6-state policy, which is in accordance with our analysis for policy-wise comparison above. Also, the rewards that are won by our agent are generally greater if the non-agent has a tendency of folding more. This is because if the non-agent folds more, their current bet in the pool will be much more likely to win by our agent. For the non-agent behavior in category 0 and 1, our agent's estimated rewards do not differ in an obvious pattern, as the non-agent stays in the game either by raising or checking, and our agent is therefore less likely to be able to collect the rewards in the pool.

We also found out from Table II to V that if our agent is in the starting position 0 and 1, which is the first and second player to start acting, it is easier to collect a higher reward. The small blind (position 2) and the big blind (position 3) in general are rewarded less, as the agent has to bet a small amount blindly in the first place. This observation is also in accordance with the real-life observation of strategy summarized by Texas Hold'em players, meaning that our model has learned an accountable strategy for the game.

V. Discussion and Conclusion

One of the most notable findings in this project is that we found that our generated policies using Q-learning and Sarsa algorithms have achieved a significantly better performance than the random policy, as shown in Table I of our result section.

This shows that we have successfully trained a model whose strategy (policy) that can be applied to the hold'em game. Moreover, by refining the model for our hold'em game into a 52-state model, we are certain that the 52-state policy can win considerable rewards (positive rewards) in the game. The 6-state policy only performs better than the random policy, but is still possible to lose (earn negative rewards) in the game, while the agent following 52-state policy is able to win the game (earn positive rewards). By analyzing this result, we conclude the reason for such differences is that the 52-state model takes consideration into how good the cards our agent currently holds are, such that the agent can decide better based on current information. On the contrary, the 6-state model does not consider the cards and only decides based on the state of the game, which is therefore reasonable that our agent can only earn negative rewards in the 6-state model policy, but win in the 52-state refined model.

To answer the questions in the very beginning, we found out that in theory, it's better never to fold from the game, but considering the reality of the chip limit, it's probability better to make more considerate decisions with the knowledge of the current cards, instead of completely depending on luck and blindly raising or checking. On the other hand, the seat position does not matter too much, as long as there're no blind bets used. But even though with big and small blinds, if properly learning the strategies, it's still possible to gain a positive reward.

To conclude, while the actual hold'em game has more possibilities mainly on how the non-agent will tend to act in the game, we have found a winning policy for the Texas hold'em game abstracted version and it can be further improved in accuracy (the value of rewards earned) by accounting for more possibilities of different non-agents.

VI. Contribution

We contributed equally, either verbal discussion or document written, to the Project Proposal (at week 3), and project status update (at week 8), and the final written project paper (at week 10).
We contributed equally to the API discussion between the policy learner and the game simulator.
Ye participated in discussing the design of the simulator, and implemented the policy learner.
Jamie participated in discussing the algorithms of the policy learner, and implemented the simulator.

References

[1] Fredrik A. Dahl. "A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold'em Poker". In: *Machine Learning: ECML 2001*. Ed. by Luc De Raedt and Peter Flach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 85–96. ISBN: 978-3-540-44795-5.

[2] PokerNews. *Texas Holdem Poker: How to play.* Retrieved: 2023-01-19. URL: https://www.pokernews.com/poker-rules/texas-holdem.htm.

[3] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. CUED/F-INFENG/TR 166. Cambridge University Engineering Department, Sept. 1994. URL: ftp://svr-ftp.eng.cam.ac.uk/reports/rummery_tr166.ps.Z.

[4] Christopher John Cornish Hellaby Watkins. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, May 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.