Learning to Play Texas Hold’em using Q-Learning

Aishwarya Kurnutala, Sai Rahul Ponnana
{kurnutala.a@northeastern.edu, ponnana.s@northeastern.edu}

Abstract

We are developing a game playing agent for the game Texas Hold’em using Q-Learning. Card games use multiple agents where we can use fundamental reinforcement learning algorithms and methods to make agents learn. We implement this on how Texas Hold'em's issue might be best approached using Q-Learning. To understand the most important pieces of information in Texas Hold'em, the game is modelled using a Markov Decision Process (MDP). Additionally, we execute Q-Learning using five distinct feature extraction algorithms and two baseline agents that follow fixed rules but no learning, since there are seven players in this game. We expect Q- Learning to occasionally outperform random, we show that it does not outperform a fixed policy of calling every bet. By comparing the performances of specific agents with the unique Q-Learning strategies on the model, as well as comparing those agents with baseline agents, we can compare the Q-Learning model’s performance.

Introduction

Poker is one of the most popular and extensively played gambling games, with millions of gamers playing a wide range of variations, from casual gamers who play for pennies to experts who compete in million-dollar tournaments. As a result, poker's success can be attributed to its basic mechanics. In order to make smart choices given their possibilities, players must observe their opponent’s traits. The stochastic aspect of poker introduces considerable statistical fluctuation, letting weak players win sometimes, but an excellent player will consistently dominate a suboptimal opponent. Texas Hold'em is a community card game which is one of three poker versions. To match human decision-making standards, this project will apply Artificial Intelligence techniques like Q-learning to Play Texas Hold’em.

This problem is interesting because currently, it is unknown whether Q-Learning would function appropriately when employed to solve Texas Hold'em. According to our findings, Q-Learning is not completely suited to handle complicated, multi-agent systems with extensive action space and state space as Texas Hold'em. Although Q-Learning occasionally outperforms random, it regularly outperforms a fixed policy that always chooses to call, saying that Q-Learning alone cannot solve the Texas Hold'em dilemma. Our research further demonstrates that Q-Learning is more successful when relevant data is gathered.

Each player employs a unique feature extractor to discover features of the game, or a fixed baseline policy without learning to play several hundred thousands of games between them. To achieve a better understanding of the most important data elements in Texas Hold'em, we develop a MDP model and employ Q-Learning with five distinct feature extraction algorithms. Q-Learning performs better as more data is extracted, which means that adding more accurate information to the state and action space is preferable to just summarizing more Q values.

Background

We will use the Texas Hold'em basic rules in our model, so knowledge of the rules will be necessary for the scope of this project. Playing the game involves the following steps:

- ROUND – 0 Two cards are dealt to each player. Three cards are drawn on the table without revealing the cards (the flop). In this game, the objective is to achieve the strongest five hand card possible by combining your hand cards with the flop cards. You can choose between folding, calling, or raising with that in mind and your current hand of two cards in hand. Your bet is placed into the "pot" at center of the table. To continue playing, it is necessary for all other players to match your raise if you raise. To continue playing, you must always at least match the bets of other players.

- ROUND – 1 The three flop cards are turned face-up. You can now think of flop cards as a part of your own hand. The game moves clockwise around the table for the second round of betting. The player may fold at this point.
- ROUND – 2 A third round of betting occurs once an additional card (turn card) is turned over.
- ROUND – 3 A fourth and ultimate round of betting occurs once an additional card (river card) is turned over. At this stage, whoever is still in the game and has the highest-valued five-card combination wins the pot. Alternatively, if everyone but one player folds before Round 3, the game is over. It is worth noting that if you folded earlier in the game, you would have lost the entire amount you bet before folding.

There is also an order of combinations of the five cards, from highest to lowest, which helps in making the decision on who wins the game.

Related Work

After implementing the Q-learning algorithm and pitting several games against each agent, we try to compare the earnings of each agent with another agent. Similarly, we try and compare the earnings of each feature extraction algorithm with another, it is expected that the feature extraction algorithm agents have more earnings than the baseline agents, and overtime the earnings increases because the agents learn on earning money from the pot. We try to prove Q-learning algorithms show greater improvement than random baseline agent.

Approach

In Texas Hold'em, the state is determined by a Markov Decision Process (MDP), which contains the below details:

- STATE['board'] = Represents a list of cards on the table.
- STATE ['pot'] = The amount in the pot is represented by an integer, which is the total of all the players' bets.
- STATE ['players'] = Each player is represented by a list of tuples (hand, bet). If the bet = −1, the player folds and hand = False
- STATE ['curBet'] = represents the round's current highest bet, bet for either raise or call.
- STATE ['curPlayer'] = represents the index of next player whose turn it is to bet.

Since there are seven agents in learning, we try to execute five agents under five feature extraction algorithm of Q-learning and two of the baseline agents, where no learning is involved:

- Standard: Extracts [state = board + hand] and [action = any action in Markov Decision Process State].
- Action Agnostic: Extracts only [state = board + hand].
- Binary Action: Extracts [state = board + hand] and [action = 0 if fold; 1 otherwise].
- Hand Rank: Extracts only [state = rank of (board + hand)].
- Hand Rank with Binary Action: Extracts [state = rank of (board + hand)] and [action = 0 if fold; 1 otherwise].
- Random Action: Pick a good random action based on: [action = randomly choose an action a ∈ Markov Decision Process State]
- Uniform Call Action: To choose whichever action calls (matches) the bet uniformly.

Q-learning Algorithm

Reinforcement learning has gained a lot of attention, with numerous successful applications in fields such as game theory and statistics. Q-learning is a simple technique for training agents how to behave optimally in controlled Markovian environments. It operates by gradually increasing its evaluations of the quality of certain actions at specific states. Many studies have described Q- learning's applications in reinforcement learning and artificial intelligence problems. There is, however, a knowledge gap on how these sophisticated algorithms might be used and included into general artificial intelligence operations. Since the q-learning function learns from actions that are outside the present policy, including performing random actions, a policy is not required, hence it is regarded as being off-policy. Q-learning aims to discover a policy that maximizes total reward, to be more precise. In q-learning, the "q" refers to quality. In this context, pertains to how helpful a particular action is in obtaining a potential reward.

The state/action table is the central component of the Q learning algorithm. It is a two-dimensional matrix that associates each action with a state and its associated values Q(s,a). The algorithm's purpose is to discover the optimal Q-table that maps the states to their appropriate Q-values repeatedly. We must utilize the Bellman equation for this:

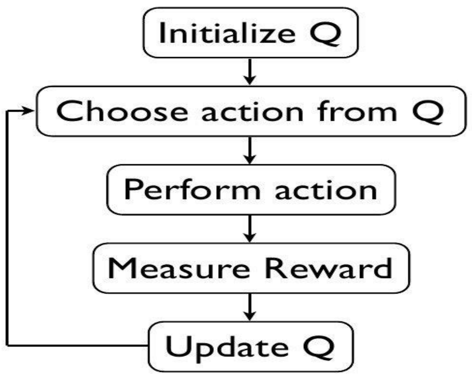
$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

After each step or action, an update is made, and it stops when an episode is finished. A terminal state might be anything, such as winning a game. After only one episode, the agent won't have learned anything, but with enough exploration (steps and episodes), it will ultimately converge and discover the ideal Q-values or Q*.

The basic steps are as follows:

- Agent begins in a state (s1), performs an action (a1), and is rewarded (r1)
- The agent chooses an action by looking up the highest value in the Q-table (max) OR at random (epsilon, ε)
- Update the q-values

Below is a flowchart for Q-learning:



Markov Decision Processes

Markov Decision Processes (MDP) are a basic and intuitive formulation for reinforcement learning (RL), and other stochastic learning applications. An environment is depicted as a set of states in this paradigm, and actions may be undertaken to control the system's state. The purpose is to alter the system so that some performance criterion is maximized.

MDPs are made up of states, actions, state transitions, and a reward function definition. The Markov decision process (MDP) is a mathematical model of sequential choices as well as a method of dynamic optimization. A MDP is made up of five components {T,S,A,p,r} where

- T represents all decision time sets.
- S is a set of discrete nonempty states that represents all possible system states.
- A is a collection of all possible decision-making behaviors in a given state of the system.
- p denotes the likelihood of the system moving to state j when it is in state i∈S and the decision-making behavior a ∈A(i) is used.

$$\sum_{j\in S} p(j|i,a) = 1$$

- r = r (i, a) is referred to as a reward function because it represents the expected reward gained when the system is in a state I ∈ S at any time and engages in a decision-making action a ∈ A(i).

The strategy specifies that we should take a certain action in each state. In the Markov decision-making process, the transition probability and reward are decided completely by the current state and the action made by the decision maker and have nothing to do with the previous actions.

Experiments and Results

In 1 million five-agent Texas Hold'em games, we run every single one of the five Q-Learning techniques against each other. It should be noted that each agent retains their unique weight in each game. Our results are as follows:

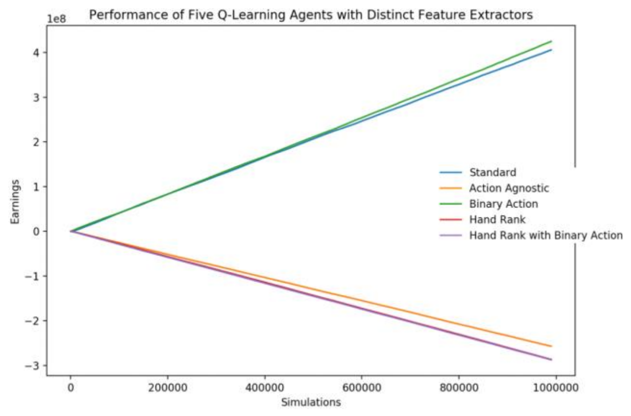


Fig 1: Performance of five Q-learning Agents

From the figure 1 we can illustrate that the earnings of Binary Action and Standard techniques increases with the number of simulations. And the rewards of the Hand rank with Binary Action and Action Agnostic drop drastically as the number of simulations increases. We may argue that Binary Action and Standard approaches perform better than the other methods. This appears to suggest that these techniques capture more detailed data.

We will then test whether Q-learning is superior to random approaches by comparing the least performing Q-Agent to the Random Action Agents.

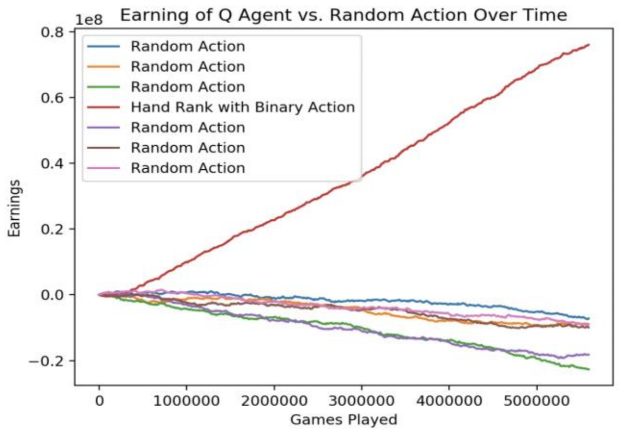


Fig 2: Performance of Q-Agent vs Random Action Agents

Figure 2 shows that Hand Rank with Binary Action is the best performing agent when compared to Random Agents, as the Random Agents' earnings decline dramatically over the games played. Rewards of the Hand Rank improve significantly as more games are played. The learning process takes longer than expected, therefore, a higher performance isn't noticed until later. It takes about 1 million games for the Hand Rank to begin to improve.

We then compare the earnings of Q-Agents over time against Randon Action and Uniform Action call.

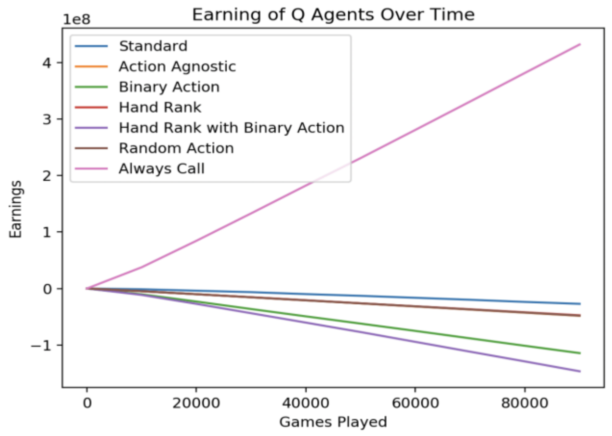


Fig 3: Earnings of Q-Agents over time

From Figure 3, we can see that Q-Learning outperforms Random Action for a lesser number of games, since the player's earnings fall drastically while employing Q-Learning, Q-Learning does not outperform Uniform Call action, also known as Always call. This is because the uniform agent is more likely to win if he repeatedly calls the bet than Q-Agent, who is more likely to give up.

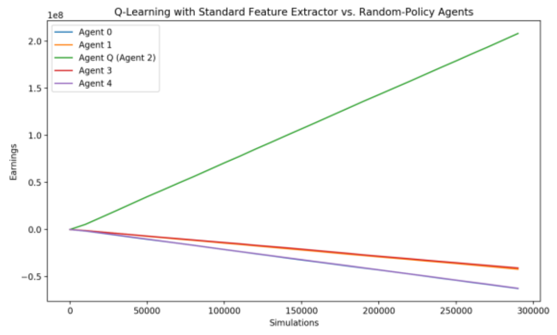


Fig 4: Q-learning Agent vs Random policy Agents

From Figure 4 we can see the earning of all the Agents. When every time an agent calls for a game, their earnings increase. The earnings of Q-agent is highest compared to all other agents here.

This indicates that, while Hand Rank generalizes better than Binary Action and Standard Action, it would need more time for training to converge to effective Q values. Alternatively, the MDP's action and state space seem to be vast enough to need quite a delay in observing progress in performance.

Conclusion

The Q-learning model works according to the probability of winning and Markov Decision process. Q-learning cannot function effectively if a person is bluffing or playing mind games. As a result, it lacks the ability to bluff and conduct mind games. The Q-agent calls for a game only if there is a high possibility of winning, else, it drops the game.

Even though Q-learning shows the best results when compared to random methods, It is not effectively designed to handle complex multi-agent problems. Despite being able to perceive the value of certain bets provided certain hands, there remains a lot of unpredictability in the model's decision making. We assume that in the case of high computing capacity, Q-Learning might be successful against random action agents. With additional computing power, it might be capable to construct its Q matrix quite comprehensively and allow the model to take smart and accurate decisions for a particular state and action.

Contributions

Our team consists of 2 members, and each of us have equally contributed and worked on different sub-tasks. The tasks like programming the structure of our MDP and creating data visualizations are performed by Sai Rahul Ponnana, while Aishwarya Kurnutala designed the MDP framework and programmed most of the Q-Learning.

Implementation and training of the models was performed by both of us. And Both of us have equally contributed to preparation of the report.

References

[1] Lai, H. (2019, November 12). RLCard: Building your own poker AI in 3 steps. Medium. Retrieved December 15, 2022, from https://towardsdatascience.com/rlcard-building-your-own-poker-ai-in-3-steps-398aa864a0db

[2] Watkins, C. J. C. H., & Dayan, P. (n.d.). Q-learning - machine learning. SpringerLink. Retrieved December 15, 2022, from https://link.springer.com/article/10.1007/BF00992698

[3] Author links open overlay panel Chelsea C.WhiteIIIDouglas J.White, C.WhiteIII, C., J.White, D., AbstractA review is given of an optimization model of discrete-stage, White, D. J., Striebel, C. T., MacQueen, J., Bellman, R., Bertsekas, D. P., Blackwell, D., CONDOR (Committee on the Next Decade in Operations Research), Denardo, E. V., Derman, C., Dirickx, Y. M. I., Henig, M. I., Heyman, D. P., Ho, Y. C., Howard, R., Kallenberg, L. C. M., … Popyak, J. L. (2011, January 13). Markov decision processes. European Journal of Operational Research. Retrieved December 15, 2022, from https://www.sciencedirect.com/science/article/pii/0377221789903482

[4] Violante, A. (2019, July 1). Simple reinforcement learning: Q-learning. Medium. Retrieved December 15, 2022, from https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[5] Worms, D. (n.d.). Applying deep reinforcement learning to poker. Adaltas RSS. Retrieved December 15, 2022, from https://www.adaltas.com/en/2019/01/09/applying-deep-reinforcement-learning-poker/

[6] Poker bot. Poker Bot: A Reinforced Learning Neural Network. (n.d.). Retrieved December 15, 2022, from https://www.cs.hmc.edu/~ktantia/poker.html