Poker Project - Team 07

Arielyte Tsen Chung Ming, Devarajan Preethi, James Pang Mun Wai, Lee Yi Wei Joel, Yip Seng Yuen
National University of Singapore
chungming.tsen@u.nus.edu, preethi_devarajan@u.nus.edu, jamespang@nus.edu, lywjoel@u.nus.edu, yip@u.nus.edu

1 Introduction

The game of poker has long been a field of interest for artificial intelligence (AI) researchers. There are many challenges with developing an AI poker agent capable of competing with humans, since poker is a complex game that forces players to make decisions with incomplete and imperfect information. AI poker agents are hence faced with the task of having to make the best decision at every street under such informational constraints. Additionally, conventional poker matches impose a time limit on players to make a decision, adding another constraint for AI poker agents to work with.

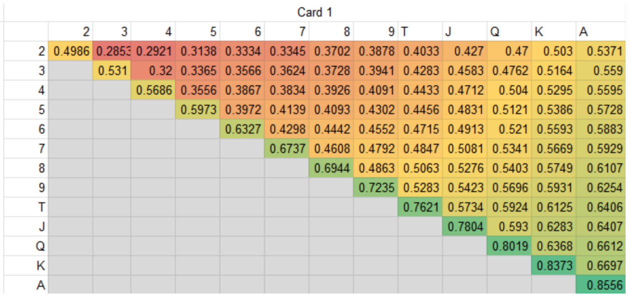
This paper outlines our design of an AI poker agent for Heads Up Limit Texas Hold'em, which aims to respond competently and accurately with limited information, within a limited amount of time. In tackling the issues above, we employed a three-pronged approach:

- Using Q-Learning, a reinforcement learning algorithm, to train our agent to play with the best possible strategy. (Section 3.1)
- Reducing the overall state size, by grouping similar states together. (Section 3.1, Clustering)
- Precomputing the estimated strength of hand ranks, storing them in look-up tables to allow for retrieval in constant time. (Section 3.3)

The following two sections will explain our agent's strategy within each betting round, namely the pre-flop and postflop streets.

2 Preflop Street

In order to decide what action the agent would take based only on the limited amount of information it has during the preflop street, we introduce the use of hand strength (HS), which is an estimate determined by the probability of winning with the two hole cards currently in the agent's hand. To estimate HS, we perform a Monte Carlo simulation of 10,000 games for each possible starting hand. The total number of possible starting hands is 91, which is obtained by summing up the total number of possible pocket pairs and the total number of possible hands disregarding the suits. For each game, we first pick the two starting cards for the agent, followed by randomly picking two starting cards for the opponent and the five community cards. Both players' hands are then revealed and the game result is determined as a win or a loss for the agent.



	Card 1												
Card 2	2	3	4	5	6	7	8	9	T	J	Q	K	A
2	0.4986	0.2853	0.2921	0.3138	0.3334	0.3345	0.3702	0.3878	0.4033	0.427	0.47	0.503	0.5371
3		0.531	0.32	0.3365	0.3566	0.3624	0.3728	0.3941	0.4283	0.4583	0.4762	0.5164	0.559
4			0.5686	0.3556	0.3867	0.3834	0.3926	0.4091	0.4433	0.4712	0.504	0.5295	0.5595
5				0.5973	0.3972	0.4139	0.4093	0.4302	0.4456	0.4831	0.5121	0.5386	0.5728
6					0.6327	0.4298	0.4442	0.4552	0.4715	0.4913	0.521	0.5593	0.5883
7						0.6737	0.4608	0.4792	0.4847	0.5081	0.5341	0.5669	0.5929
8							0.6944	0.4863	0.5063	0.5276	0.5403	0.5749	0.6107
9								0.7235	0.5283	0.5423	0.5696	0.5931	0.6254
T									0.7621	0.5734	0.5924	0.6125	0.6406
J										0.7804	0.593	0.6283	0.6407
Q											0.8019	0.6368	0.6612
K												0.8373	0.6697
A													0.8556

Figure 1: Estimated winning probabilities with starting hands

We then calculate HS using the formula:

$$HS = Pr(Win) = \frac{\#\, of\ wins}{Total\ games\ played}$$

where the total games played is fixed at 10,000. The results of the Monte Carlo simulation are seen in Figure 1.

At every preflop street, the agent chooses an action [fold, call, raise] to take based on the HS of the starting hand:

$$\left.\begin{array}{l} raise\colon HS > k \\ \\ call\colon HS > j \\ \\ fold\colon HS \leq j \end{array}\right\} \quad \text{where } 0 < j < k \leq 1$$

where j and k are pre-determined thresholds set for the agent.

3 Postflop Streets

The postflop streets consist of the Flop, Turn and River streets. In contrast to the preflop street, at least three community cards are revealed during the postflop streets, giving us more information to work with. Here, reinforcement learning is employed. Specifically, we use a modified version of the Q-Learning algorithm to train our agent.

3.1 Q-Learning

As per the Q-Learning algorithm, we maintain a table of states and actions as an input to the agent. Each state has a Q-Value for each action (Fold, Call or Raise), signifying

State	**Action**		
	fold	*call*	*raise*
1			
2			
3			
...			
S			

Table 1: Q-Learning Training

the expected reward for the action. An illustration of the Q-Learning table can be seen in Table 1.

The reward function is determined based on the pot size P - the total amount that has been bet by both players - at the end of the game, as seen in the formula below:

$$R_t = \begin{cases} +\frac{P}{2} & \text{if win,} \\ -\frac{P}{2} & \text{otherwise} \end{cases}$$

After each training round, the Q-Value of each state is obtained by summing up the previous Q-Value and the reward function for that round. More formally,

$$Q\left(s', a'\right) = Q\left(s, a\right) + \sum_{0}^{t} \lambda^t R_t\left(s, a\right)$$

where t is the number of turns between the current move and the terminal node, R_t is the reward for round t, λ is the discount factor, s is the state, a is the action and P is the pot size.

Experience Replay

Additionally, we apply the technique of experience replay to achieve more efficient use of previous experiences. We store the agent's experiences based on the formula

$$e_t = (s_t, a_t, R_t, s_{t+1})$$

The agent stores the data it discovers for each round in a table, and reinforcement learning takes place based on random sampling from the table. This reduces the amount of experience required to learn, replacing it with more computation and memory which are cheaper resources than the agent's continuous interactions with the environment [Schaul *et al.*, 2016].

ε-Greedy Algorithm

At every turn, with a probability of ε, a random action is chosen to be performed, otherwise an action with the best expected reward (highest Q-Value) is chosen.

The value of ε will slowly decrease as the agent acquires more training. A relatively large initial value ensures that all possible paths will be explored, to avoid the scenario where the agent settles into a sub-optimal pattern of calling the same action repeatedly. At every street, the state of the agent is determined by the following attributes:

$$State_i = \{EHS_i, S_i, P_i, \#OR_i, \#SR_i, OPS_i\}$$

where

EHS	refers to the agent's current Expected Hand Strength. More details can be found in Section 3.2.
S	refers to the current street of the round.
P	refers to the pot size, which is the total amount that has been bet by both players so far.
#OR	refers to the number of raises the opponent has made in the round so far.
#SR	refers to the number of raises the agent has made in the round so far.
OPS	refers to the Opponent Playing Style. More details can be found in Section 3.4.

Clustering

Using clustering to group information into buckets would reduce the massive state space, making it much more feasible to explore all possible states. We apply clustering to group the information for our Q-Learning algorithm as such:

- *EHS* is divided into groups with an increment of 0.01, from 0 to 1.
- *S* represents the three postflop streets - Flop, Turn and River.
- *P* ranges from \$0 to \$680 per game. We group it into increments of \$40, which is the 2 times the big blind amount, since every raise (big blind amount - \$20) must minimally be matched by the opponent. We get a range of 0 to 17.
- *#OR* simply ranges from 0 to 4.
- *#SR* simply ranges from 0 to 4.
- *OPS* is grouped into four categories. More details can be found in Section 3.4.

These groupings help to reduce the complexity and cut down the size of the total state space to: $101 \times 3 \times 18 \times 5 \times 5 \times 4 = 545,400$ states

3.2 Expected Hand Strength

The Expected Hand Strength, *EHS*, is the probability that a given player's current hand wins if the game reaches a showdown. It factors in all possible combinations for the opponent's hand and the remaining unrevealed board cards, and compares each of these hands to the agent's hand to determine which hand wins. The *EHS* of the agent's hand is calculated by finding the total number of times the agent's hand wins against the opponent's, out of all possible comparisons. More formally,

$$EHS(h) = \frac{Ahead(h) + \frac{Tied(h)}{2}}{Ahead(h) + Tied(h) + Behind(h)}$$

where the *Ahead*, *Tied*, and *Behind* functions respectively determine the number of times the given player's hand wins, ties or loses against its opponent's. The above formula used to derive the *EHS* was referenced from Teofilo *et al.* [2013a].

Note that the *EHS* may be required by the agent at any street of any round. However, the time required to accurately compute *EHS* based on large samples may exceed our time constraint of 200 milliseconds. This issue can be addressed by using a lookup table, which is explained in the next section.

3.3 *EHS* Lookup Table

We introduce a technique similar to the Average Rank Strength (*ARS*) heuristic [Teofilo *et al.*, 2013b] to improve the efficiency of *EHS*. In our method, three look-up tables are created - one for Flop, one for Turn and one for River. These tables consist of precomputed *EHS* values for all possible hand scores. The score of a particular hand refers to its strength, and is computed by PyPokerEngine's built-in $HandEvaluator.eval\_hand()$ function. For example, $HandEvaluator.eval\_hand()$ returns a numeric representation of the agent's hand score given some combination of hole cards and community cards which contain a Pair, which would be lower than if there were a Three-of-a-Kind. More details can be found in the PyPokerEngine repository.

Instead of computing *EHS* during gameplay, the agent now refers to the lookup table, to obtain its *EHS* based on its current hand score. The use of this technique allows the computation to run in linear time, and results in an approximate thousand-fold improvement in response time, while compromising negligibly on accuracy [Teofilo *et al.*, 2013b].

To enumerate the *EHS* table for the three postflop streets, for each street we performed a Monte Carlo simulation of 2.5 million samples of the agent's hand and the revealed community cards, also calculating the agent's hand score for each sample. Additionally, for each sample, we performed another simulation of 500 samples of the opponent's hand and the unrevealed community cards (if any) to calculate the *EHS* for the hand score. The results are displayed below:

Street	**Number of Distinct Scores**
Flop	5133
Turn	13,408
River	17,470

Table 2: Results of *EHS* Lookup Table Simulation

We are aware that not all possible hand scores may appear in the tables generated during the simulation. However, our results from simulating 100,000 games reveal that approximately 99.87 percent of all hand scores can be found in each table, showing that the lookup tables remarkably maintain a high degree of reliability and accuracy. In the case where an agent looks up a hand score that is not enumerated in the lookup table, we perform a limited Monte Carlo simulation of 100 samples for that particular hand score and add it to the lookup table.

3.4 Opponent Playing Styles

The playing style of an opponent can be classified into four categories [Rupeneite, 2010]. Each style distinctly describes the opponent's frequency of play and style of betting. The four categories of playing styles are Loose/Passive, Loose/Aggressive, Tight/Passive and Tight/Aggressive. A brief description of each style is given in Table 3 below.

Aggressiveness Factor, AF, is the ratio of a given player's number of raises to the number of calls. More formally:

$$AF = \frac{\#\, raises}{\#\, calls}$$

Playing Styles	**Description**
Tight	Plays few hands and often folds.
Loose	Plays multiple and varied hands.
Aggressive	Bets and raises a lot, almost always never checking or call.
Passive	Usually checks and call, unlikely to take the lead.

Table 3: Description of Playing Styles

Players are classified as either Aggressive or Passive based on their Aggressiveness Factor. Based on research by Rupeneite [2010], a threshold of 1 is used:

- Aggressive if $AF > 1$
- Passive if $AF \leq 1$

Player Tightness, PT, is the ratio of a given player's number of folds to the total number of games. More formally:

$$PT = \frac{\#\, folds}{\#\, games}$$

Players are classified as either Loose or Tight based on their Player Tightness. Based on research by Rupeneite [2010], a threshold of 0.28 is used:

- Tight if $PT < 0.28$
- Loose if $PT \geq 0.28$

Later, a classification process, introduced by Dinis and Reis [2008], combined these two heuristics to classify the opponent's style of play into four categories, as seen in Table 4.

	AF $\leq$ 1	**AF $>$ 1**
PT $\geq$ 0.28	Loose Passive	Loose Aggressive
PT $<$ 0.28	Tight Passive	Tight Aggressive

Table 4: Style of Play Classification

4 Limitations

4.1 Q-values converging

The Q-value refers to the expected reward at every state of the game. As mentioned in the ε-Greedy algorithm, at every turn there is a probability of ε that a random action is chosen, else the action with the highest Q-value is chosen. A Q-value that is reliable is one that converges, meaning multiple occurrences of that particular state returned Q-values that did not vary too much from each other. Ideally, we would want to allow the agent to learn until all Q-values converge, which would allow the agent to choose the best action at every state. However, due to the time constraints of the project, our agent did not receive sufficient training and hence has Q-values that do not converge. This means that it is not guaranteed that our agent will always be able to take the best action, due to the inaccuracy of some of the Q-values.

4.2 Counterfactual Regret Minimisation

Counterfactual Regret Minimisation (CFR) is another reinforcement learning algorithm that we could have used while implementing our poker agent. This is an algorithm that seeks to minimise regret about its decisions at each step of a game. There are two types of regret - positive and negative regret. Negative regret refers to the regret of having taken a particular action in a particular situation. It means the agent would have done better had it not chosen this action in this situation. Positive regret is mechanism by which the agent tracks actions that resulted in a positive outcome. Unlike Q-learning, which remembers the reward received for every action and chooses the action with the highest reward, CFR remembers the regret for every action and favours the action it regretted not having taken previously. However, CFR assumes that a terminal state is eventually reached and performs updates only after this occurs, which is not a requirement for traditional algorithms like Q-learning.

Acknowledgments

The preparation of this report would not have been possible without the help of Dr. Yair Zick and Arka Maity, National University of Singapore, School of Computing.

References

[Dinis and Reis, 2008] Felix Dinis and Luis Paulo Reis. An experimental approach to online opponent modeling in texas hold'em poker. In *Advances in Artificial Intelligence*, pages 83–92, Savador, Brazil, October 2008. Brazilian Symposium on Artificial Intelligence.

[Rupeneite, 2010] Annija Rupeneite. *Building Poker Agent Using Reinforcement Learning with Neural Networks*. SCITEPRESS, 2010.

[Russell and Norvig, 2014] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2014.

[Schaul *et al.*, 2016] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of the International Conference on Learning Representations 2016*, pages 1–21, San Juan, Puerto Rico, May 2016. Canada Institute for Scientific and Technical Information.

[Teofilo *et al.*, 2013a] Luis Filipe Teofilo, Luis Paulo Reis, and Henrique Lopes Cardoso. Computing card probabilities in texas hold'em. In *Proceedings of the 8th Iberian Conference on Information Systems and Technologies*, pages 988–993, Lisbon, Portugal, June 2013. Canada Institute for Scientific and Technical Information.

[Teofilo *et al.*, 2013b] Luis Filipe Teofilo, Luis Paulo Reis, and Henrique Lopes Cardoso. Speeding-up poker game abstraction computation: Average rank strength. In *Proceedings of the 27th Association for the Advancement of Artificial Intelligence Workshop*, pages 59–64, Bellevue, Washington, USA, July 2013. Association for the Advancement of Artificial Intelligence.