Implementing Q-learning for Limit Texas Hold'em

Teo Yue Yang[1] , **Mark Erenberg**[1,2] , **Stephan Breimair**[1,3] and **Sharad Kochhar**[1,4]

[1]National University of Singapore
[2]University of Waterloo
[3]Technical University of Munich
[4]Simon Fraser University

1 Introduction

Since the inception of the modern artificial intelligence (AI), games have played a pivotal role in the its development; serving as a testbed for measuring AI progress. In the recent decades, AI has progressed remarkably and trumped over humans in games such as checkers [1], chess [2], and Go [3]. However, these games are structured with perfect information, enabling the AI to strategize possible moves at relatively low complexity. Games with imperfect information, such as poker, are much more complicated than their counterparts [4]. Imperfect-information games obscures information from the players, making it computationally expensive to devise an optimal strategy. However, this very property mirrors real world decision-making settings and hence proposes imperfect-information games as a benchmark to assess AI's capability in problem solving. Amongst these games, poker has long been used to evaluate AI in tackling imperfect-information games [4], [5]. One variant of poker, Limit Texas Hold'em, has reduced complexity due to the limit placed on the bets, making it an ideal game for a preliminary assessment of an AI in imperfect-information games [5]. Therefore, this paper aims to design an AI to play Limit Texas Hold'em.

Reinforcement learning (RL) is a potent learning tool in which an agent chooses the best action to maximize its cumulative rewards within an unknown environment over time [6]. This is extremely applicable to a reward-based game such as Limit Texas Hold'em [7]. There are two main approaches to RL: model-based versus model-free. The former approximates a model for the environment while the latter merely updates the value-function based on the reward without assuming an underlying model for the environment. Although there has been a long deliberation as to which variation is better [8], we decided to utilize model-free RL based on the following reasons. The model-free paradigm is online, require less space, and much more flexible than the model-based paradigm [9]. These characteristics, especially its flexibility, enables a model-free approach to be suitable to Limit Texas Hold'em where it is inefficient to model the environment variables like opponent's behaviour.

For this paper, we chose to adopt a model-free RL algorithm, Q-learning. Q-learning allows the agent to find the optimal action-selection policy that maximizes the reward variable over all the states observed [10]. We fashioned two alterations of Q-learning: Q-learning (QL) and Q-learning with Minimax (QLM), and we compared the trained models against each other and against an irrational random agent.

2 Implementation of Models

2.1 Q-Learning Model

For QL implementation, we adopted a lookup table (Q-table) with the states against the actions as showed in Table 1.

Table 1: Q-table format.

State (S_i)	Actions (a_1, ..., a_m)
S_1	S_1a_1, ..., S_1a_m
S_2	S_2a_1, ..., S_2a_m
...	
S_n	S_na_1, ..., S_na_m

The state parameters chosen are the pot amount, the cards combination, the street and the condensed version of the opponent's actions. We believed that these parameters are essential to choosing the optimal actions and the rationale are summarized as shown in Table 2. The action space contains fold, call and raise. Hence, the Q-table has a size of 4554 x 3.

Table 2: Rationale for state parameters

State parameter	Formulation	Range	Rationale
Pot Amount	pot/20	[4,72]	A high pot is worth the risk to call or raise
Card Combo	Each card combo is indexed	[0,10]	A good hand is worth the risk to call or raise
Street	Each street is indexed	[1,3]	The street correlates to the pot and community card as each street reveals new information
Condensed Opponent's Actions	If the opponent raised more or called more	[0,1]	If the opponent raised more, they might have better cards

For the street parameter, preflop was ignored as we concluded that any decision made is too premature considering the minimal information. Hence at the preflop level, the action is chosen randomly. For the random action, refer to Section 2.3, the implementation of the random agent.

For every state and every action, there exists a quality value (Q-value) in the Q-table (which is the S_ia_j in Table 1). The

Q-value is a scoring system of the action taken in a specific state; the higher the Q-value, the better the action is for the agent. The Q-value is adapted from [11], formulated as shown in Eqn 1.

$$Q(S_t, a_i) = (1-\alpha) \cdot Q(S_t, a_i) + \alpha \cdot (r + \gamma \cdot (\max_a \{Q(S_{t+1}, a)\})) \quad (1)$$

$Q(S_t, a_i)$ refers to the Q-value for the action i at state t, α refers to the learning rate, γ is the discount factor and r is the reward. The learning rate decides how fast the algorithm learns from the data and the discount factor determines how much does the scaling factor of the future reward. They are set at 0.4 and 0.5 respectively. These values were obtained via a optimization protocol elaborated in Section 3.

The Q-table was initialized as a zero matrix. The QL algorithm works as such: if the state is novel and has yet to be observed, the QL algorithm will take a random action; conversely, if the state has been observed before, the QL algorithm will choose the action with the maximum Q-value in the action space of the given state. Consequently, this means that in the early stages, the QL algorithm will most likely choose random actions.

For every game round, the QL algorithm tracks all the states with the respective action taken. It only updates the Q-table at the end of the round when the reward, the amount of points won or lost, is obtained using Eqn 1.

We also decided to modify the algorithm to a a semi-deterministic algorithm. We implemented it as such so as to make it harder for opponents to learn our actions. We set a probability threshold, ρ, based on an optimization protocol (refer to Section 3); if the pseudorandom generator generates a number below the threshold, it will output a random action, else it will choose the action based on the QL model.

2.2 Q-learning with Minimax Model

The Minimax (MM) algorithm is a decision rule that either minimizes the maximum loss of the agent or maximizes the minimum loss of the agent. In the context of a zero-sum two player game such as Limit Texas Hold'em, it generates a strategy for both players such that neither side can benefit from changing their own strategy as long as the opponent does not change their strategy [7]. For Limit Texas Hold'em, the main idea behind the algorithm is that the agent will choose an action that maximize its stack while assuming that the opponent will choose an action to minimize the agent's stack. This recurring relation will run to a certain depth or a terminal state.

The MM algorithm was constructed as a recursive function where it calls itself with the updated parameters. The MM algorithm terminates when it reaches either the maximum depth or maximum raise limit. We set the maximum depth arbitrarily at 4 and the raise limit was set at 4.

The QLM algorithm is identical to the QL algorithm with the following exception. The action space is reduced to only fold and call, and these are the terminal states for the MM implemented. The raise action was removed so as to facilitate the integration of the MM into the QL algorithm. In the QLM, when the raise action is considered, it moves to the next depth where the state parameters are updated: the pot increases according to the raise amount.

2.3 Random Model

The random model was devised with a pseudorandom number generator. Based on the pseudorandom number, x, ($0 \leq x \leq 1$) conceived by the generator, the random model will take action accordingly. If $0 \leq x \leq 0.5$, call action will be taken; if $0.5 < x \leq 0.9$ and raise limit has not been met, raise action will be taken; for all other cases of x, fold action will be taken.

2.4 Code Reference for QL Model

The $init$ section of the implementation in Python describes most of the parameters of the Q-learning. Apart from creating the Q-table itself, starting parameters such as the probabilistic threshold, ρ, for the semi-deterministic strategy, the stack size and others are set. $card\_to\_score$ translates cards with letters instead of numbers into their according values, which is necessary to calculate the value of a certain hand in $hand\_combo$ and $update\_combo$. The different combinations, e.g. flush or fullhouse, are also stored in numerical values in those methods. The higher these values the more valuable the hand is. The combinations are given by the number of equal numbers and suits. Furthermore, the opponents previous actions as given in Table 2 are stored by $update\_opp\_actions$. The new Q-values are calculated according to Eqn 1 in $update\_Qtable$. Finally, $declare\_action$ will return the desired action based on the updated Q-table to the poker engine.

3 Optimizing the QL Model

The α and γ constants from Eqn 1 and probability threshold, ρ, were optimized by running permutations of these constants: ranging from 0.0001 to 1 for α, 0.0 to 1.0 with a step of 0.2 for γ and 0.0 to 0.8 with a step of 0.2 for ρ. The QL model then played against a random model for 200 games of 1000 rounds. The ratio of the QL model's pot to the random model's pot was measured for every game and averaged out for each permutation of α, γ and ρ values. (If an agent has 0 pot left, the pot will become the minimum pot which is 1.) The α, γ and ρ constants with the maximum average ratio were selected.

4 Training Phase

The QL and QLM models were trained in several manners. We first trained the models against an irrational agent, which is the random agent. Another set of models were trained against themselves; QL versus QL and QLM versus QLM. Lastly, a final set of models with no training at all to act as controls. All training were done in 500 games of 1000 rounds.

5 Testing Phase

The QL and QLM models were first evaluated against a random model to assess how they fare against an irrational agent. The QL model was then pitted against the QLM model to test which model is better. During the testing phase, the QL and

QLM models were allowed to update their Q-tables. All testing were done in 500 games of 1000 rounds.

6 Results

6.1 Optimization Results

The optimization process revealed that the the combination of $\rho = 0.2$, $\alpha = 0.01$ and $\gamma = 0.6$ produced the highest average ratio of the QL model agent's pot against the random agent's.

To visualize the effects of the semi-deterministic strategy, we averaged the pot ratios of the different α and γ for each ρ (Figure 1). We observed that as a semi-deterministic QL model performs better than a purely deterministic model ($\rho = 0.0$) and as the model tends towards a more random model (as ρ increases), the model's performance dropped. This result validated our speculation that a semi-deterministic player is much better than a deterministic or random model.

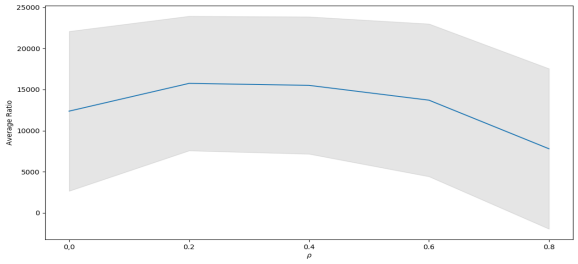

Figure 1: Average ratio of QL model's pot to random agent's pot against ρ, probability threshold for semi-deterministic strategy. Gray region represents the standard deviation of the ratios.

We selected the best performing probability threshold, $\rho = 0.2$ and the worst performing probability threshold, $\rho = 0.8$ to analyze the consequences of the learning rate, α and discount factor,γ (Figure 2). The analysis showed that at a low ρ, the different α and γ values performed similarly, with the exception of $\alpha = 1$. The models with higher γ values performed marginally better than the ones with lower ones and there is no trend for the α values ranging from 0.0001 to 0.1. However for the models with $\rho = 0.8$, it can be observed that the α and γ values have minimal to no effect. This was expected since as the model becomes more random, it will rely less on the Q-table.

6.2 Training Results

The win rates and overall pot ratios (QL agent to training agents) of the QL agent against itself and the random agent were measured to observe the training results (Table 3). It can be observed that the QL agent trained with itself has a win rate of 48.6%. This means that the QL algorithm is relatively stable and unbiased. The QL agent also had an incredibly high win rate of 99.6% and overall pot ratio of 249.1 against the random agent, which reinforces the QL algorithm's superiority over the random agent.

Similarly, the win rates and overall pot ratios (QLM agent to training agents) of the QLM agent against itself and the random agent were also measured (Table 4). The QLM agent

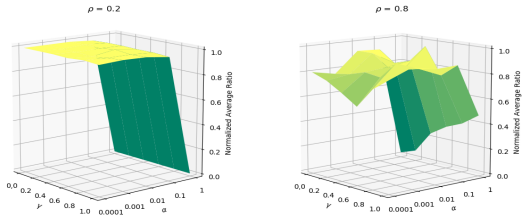

Figure 2: Average normalized ratio of QL model's pot to random agent's pot at $\rho = 0.2$ (left) and 0.8 (right) against α and γ.

Table 3: Win Rates for the QL agent during Training

	Self-trained	Random-trained	Untrained
Win Rate	48.6%	99.6%	-
Overall Pot Ratio	0.927	249.1	-

trained with itself has a win rate of 55.2%, suggesting that the QLM algorithm is also relatively stable and unbiased. Equivalently, the QLM agent also had a high win rate of 95.6% and overall pot ratio of 8.676 against the random agent.

Table 4: Win Rates for the QLM agent during Training

	Self-trained	Random-trained	Untrained
Win Rate	55.2%	95.6%	-
Overall Pot Ratio	1.104	8.676	-

Preliminary analysis of the training results favours the QL agent over the QLM agent for two reasons. Firstly, the QL algorithm is more stable than the QLM algorithm as it had a win rate closer to 50% when trained with itself. Furthermore, the QL agent fared much better than the QLM agent as it had a higher win rate when competing against the random agent.

6.3 Testing Results

The results of the games between the QL agents and the random agent and the games between the QLM agents and the random agent are shown in Tables 5 and 6 respectively. Unsurprisingly, the trained agents performed much better, with higher win rates and overall pot ratios than the untrained agent. A interesting finding was that the QL agent that trained with the random agent won the entire pot across all the games and both QL and QLM agents performed the best when trained with the random agent. This suggests that training with the opponent agent produces the optimal QL agent.

Next, we pitted the QL agent against the QLM agent (Refer to Table 7). The win rates in the Table 7 refers to the win rates of the QL agent. Evidently, we observed that across all variations of QL and QLM agents, the QL agent is dominantly better in playing Limit Texas Hold'em. Expectedly, the trained agents performed better than untrained agents, which is consistent with our previous findings. However, we were unable to determine the training agent that will maximize the win rate of the QL agent and we postulate that the QL agent we implemented is adaptive and can change strategies with ease

Table 5: Win Rates for the QL agent during Testing

	Self-trained	Random-trained	Untrained
Win Rate	100.0%	100.0%	99.8%
Overall Pot Ratio	928.138	Won Entire Pot*	395.408

Table 6: Win Rates for the QLM agent during Testing

	Self-trained	Random-trained	Untrained
Win Rate	99.2%	99.8%	90.0%
Overall Pot Ratio	15.436	23.277	5.128

according to the opponent agent. Needless to say, this requires more testing. In conclusion, we concluded that the QL agent is the optimal agent for Limit Texas Hold'em.

Table 7: Win Rates for QL agent versus QLM agent

	QL(Self)	QL(Random)	QL(Untrained)
QLM(Self)	99.6%	99.4%	94.8%
QLM(Random)	99.2%	99.4%	96.2s%
QLM(Untrained)	99.8%	99.2%	99.4%

7 Deep Q Learning

Another option considered was the use of a deep Q-learning network to implement reinforcement learning with a deeper memory store and stronger computational power. A Deep Q Network (DQN) was created using the open source TensorFlow library. The DQN contained 2 hidden layers, and minimized model loss using the Adam optimizer. The Q-value formula implemented is shown in Eqn 1.

To generate the states and actions of the next state, the Emulator object from the PyPokerEngine framework was used to simulate the possible states deriving from each action. However, the Emulator object requires an input of hole cards for all players at the table, meaning that the opponent's hole cards had to be estimated stochastically using Monte Carlo simulations. In addition, a memory store object was created to save the previous Q-values in lieu of a Q-table. This allowed the DQN to randomly sample previous states for batch gradient descent using the Adam optimizer.

During the training phase, the results of the DQN were unsatisfactory. The parameterization of the opponent's hole cards proved to be a severe limitation of this approach, as the simulations took extra computing time and did not accurately reflect a proper reward policy. As such, it was concluded that the DQN's would not perform well under testing due to the limitations of the Emulator object and the time constraints, and as such this approach was disregarded.

8 Limitations of Implementation

Although the QL and QLM agents performed exceptionally well against the random agent, the QL and QLM agents are not without flaws. One of the limitations of the agents is the state parameters. The state parameters chosen are discretized and summarized to reduce the space complexity of the agents. However, in doing so, we traded off a more refined state space, which can be essential against more complex agents that engages in intentional behaviours.

Another limitation of the agent is the optimization process and testing protocol. The optimization and testing was solely based on the random agent. Although the optimized agents performed well in the games, the optimized parameters may not work as efficiently while pitted against more complex models. Furthermore, since the testing was only done on the random agent, it remains a challenge to measure the QL agent's actual efficiency.

Additionally, the implementation of random agent and the semi-deterministic QL agent is not random in reality. This is due to the deterministic nature of the pseudorandom number generator. Hence, it creates a bias for the supposedly probabilistic nature of the agents.

9 Future Directions and Outlook

As mentioned in the previous sections, the agents are only optimized and tested with the random agent, making it impossible to quantify the actual capability of the agents. Hence, a future direction would be to optimize and test these agents against more complex and deterministic agents. This will allow a more veracious approach to assess the agents.

Other than the current implementation of the Q-learning with Q-table, there exists other forms of implementations such as deep Q-learning networks as shown in Section 7. Although the agents implemented with the Q-table were sufficient to trump the random agents, it is still imperative to test out the other implementations. By doing so, we can compare the different implementations and subsequently, devise better Q-learning agents.

Since the semi-deterministic strategy worked in our favour, we believe that the randomness confers flexibility to the agent, allowing the agent to potentially mimic bluffing. This promises another future direction: comparing the results to another sufficiently complex agent that is capable of bluffing or even playing against humans to test if the humans can tell the "bluff".

In conclusion, the QL and QLM agents we designed in this paper were effective in playing Limit Texas Hold'em against a random agent. The QL agent was superior compared to the QLM agent. In addition, a novel finding we discovered was that a semi-deterministic algorithm with a low ρ can allow the agents flexibility and consequently a less predictable and more potent strategy. If we can further prove that the randomness mirrors bluffing, it can potentiate more complex agents that can bluff comparably to humans. This supports the fact that although the semi-deterministic QL agent was simplistic, it's successful implementation provides a foundational architecture for future Q-learning or RL agents.

References

[1] J. Schaeffer, "One jump ahead: Challenging human supremacy in checkers," *ICGA Journal*, vol. 20, no. 2, pp. 93–93, 1977.

[2] M. Campbell, A. Hoane, and F. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 57–83, 2002.

[3] D. et al. Silver, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2017.

[5] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, 2015.

[6] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[7] A. Dahl, "A reinforcement learning algorithm applied to simplified two-player texas hold'em poker," *European Conference on Machine Learning*, vol. 1, pp. 85–96, 2001.

[8] D. Marc and R. C. E, "Pilco: A model-based and data-efficient approach to policy search," *International Conference on machine learning*, vol. 11, pp. 465–472, 2011.

[9] C. Jin., Z. Allen-Zhu., S. Bubeck, and M. Jordan, "Is q-learning provably efficient?" *In Advances in Neural Information Processing Systems*, pp. 4868–4878, 2019.

[10] F. Melo, P. Lisboa, and M. Ribeiro., "Convergence of q-learning with linear function approximation," *Proceedings of the European Control Conference*, pp. 2671–2678, 2007.

[11] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.