COMP423 - Reinforcement Learning and Dynamic Optimization
Poker Project Part 1 Report

Pantourakis Michail AM 2015030185
School of Electrical and Computer Engineering
Technical University of Crete

Date submitted: 30 June 2023

1 Introduction

A significant step towards understanding today's advancements in Reinforcement Learning (RL) is by first understanding the fundamental theory of Markov Decision Processes (MDP) for modelling decision making problems and the "exact" model-based algorithms that optimally solve these problems. However, these approaches are limited by their requirement for complete knowledge of the environment (model). A major breakthrough in RL was thus the development of the Q-learning algorithm, the first model-free learning algorithm with guaranteed convergence to the optimal policy.

In this report I present my course project implementing these algorithms and analyzing their ability to maximize earnings on a simplified, "toy" version of poker. I will start by first establishing the rules of this simplified game, the opponents designed as part of the environment, my model representation and how I implemented all of them in Python. With the problem formulation being established, I will then continue with how I applied Policy Iteration (model-based) and Q-learning (model-free) algorithms to this problem, reporting key observations, and most importantly, analyzing their behavior due to the nature of this game and what theory predicts.

2 Environment

2.1 Simplified game rules

Both "exact" algorithms are limited in solving problems with a relatively small number of states and actions. As poker is characterized by numerous parameters (number of players, position/order of players, legal actions, bidding round number, betting chip/token allowance, hidden and public cards, winning card combinations), the state-space quickly has a large number of states. To alleviate this issue for the purposes of this project, a simplified version of the most popular poker variant, Texas hold'em was used. Since Texas hold'em is a well-known game and its rules are also summarized in the project description, I will only provide a quick summary of its rules here, with emphasis on the unique deviations assumed here.

First of all, a dumped-down version of heads-up limit Texas hold'em is used as a basis, thus only 2 players participate. Both players start with a card in hand (not the normal of 2 cards) and with betting the same mandatory amount of 0.5 tokens (namely no small/big "blind" difference).

The game continues with a bidding round. Since this is a limit variant, either 1 (actions "bet" or "raise") or 0 (actions "check" or "fold") tokens can be placed by a player per turn of action, with a maximum of 2 tokens placed in total by each player per round. If a player folds, the other player is an instant winner and is rewarded with the full amount of tokens bet so far. Otherwise, 2 cards are further drawn and are publicly revealed, and a second bidding round, similar to the first one, begins. Unlike the real game, the order (position) of players in which they are required to take action remains the same for both rounds. For simplicity, no further rounds and card draws take place. Table **??** summarizes the legal order of actions in all game states.

Assuming no player has folded until the end of round two, the winner is then decided by comparing their "hand strength". To reduce the number of possible combinations, only cards with rank 'T', 'J', 'Q', 'K', 'A' (in ascending order) are available in deck (20 cards in total, 4 suits per rank). This simplification leaves only three winning combinations of hand and public cards: 1) 3 cards of the same kind (rank), 2) a pair of same kind, 3) no pairs. The rank of the pair or hand is used as a tie-breaker in rules (2) and (3). If all tie-breakers fail, the result is a tie and the tokens placed by both players are returned to them.

+++ Add table

2.2 Opponents as part of the Environment

Two different types of "static" (non-adversarial) agents were created in this project. These opponents serve as both necessary components of the full state-space formulation required by Policy Iteration, and benchmarks for the performance of Q-learning.

The first opponent, hereafter called *Random agent*, is a completely randomized agent. Each time an action is required by it, it randomly picks -with equal probability- one of the actions allowed according to the game's current phase. Since the Random agent disregards any other card and opponent information, its opponent cannot infer any meaningful information from its actions.

The second opponent, named *Threshold agent*, is a completely deterministic agent, using only the strengths of its cards to determine its next action. Table **??** summarizes the action this agent will perform at any possible step. In contrast to the Random agent, each and every action of the Threshold agent provides information on the range of possible ranks held in its hand. Therefore, its predictable nature can be used against it by any agent trained against it.

+++ Add table

2.3 MDP and state-space representation

Before proceeding with the implementation and results of this work, careful formulation of the state-space representation is crucial. This representation should balance between two opposing ends: 1) be as complete as possible to capture all information required by the "exact" algorithms used here, 2) have the smallest size possible to reduce computational demands without sacrificing completeness. Table 1 summarizes all features, also discussing both their necessity and how their value range was optimized.

Each state of the MDP is coupled with its legal actions only. Each state-action pair is then linked to one or more possible transitions. Each transition is in turn characterized by: 1) its conditional probability (given the state-action pair), 2) the next state, 3) the associated reward (which for a loss is negative), and 4) terminal status (as boolean). Note that in Texas Hold'em games, all intermediate states have 0 reward, and only terminal states may have non-zero values (and 0 only in case of ties).

2.4 Implementation

- State-action representation
- what was adapted from rlcard

3 MDP-based solution: Policy Iteration algorithm

- State space creation
- Implementation
- Average Results
- Analysis of Optimal Policy per opponent
- Demonstration of optimality

Feature	Value Range	Necessity	Minimization
position	'first', 'second'	Playing first or second has a direct consequence on the list of legal actions, and in case of Threshold agent, on the inferred hand range.	Unlike real poker, the simplification of retaining the same position for round 2 (flop) reduces number of possible transitions.
chips placed so far by currently acting player	'0.5', '1.5', '2.5', '3.5'	Knowing how many chips are already committed by player directly contributes to eventual rewards/losses.	'4.5' is omitted since it is only encountered in game terminal states
difference in chips of opposing player	'-1', '0', '1'	Knowing how many chips are already committed by opposing player efficiently merges information about eventual rewards/losses and current player's legal actions.	Since this is a limit hold'em game, this enumeration is smaller than using '0.5', '1.5', '2.5', '3.5', '4.5' directly.
rank of card in hand of currently acting player	'T', 'J', 'Q', 'K', 'A'	Strength of the card directly affects the player's chances of winning if no player folds until the end of round 2	Suit does not matter at all in the simplified winning conditions of this game version, therefore it can be completely omitted from state-space representation.
rank of remaining possible opposing player cards in hand in alphabetical order	'none' if not revealed yet, else all 10 combinations (e.g. 'AK', 'AJ', 'JK' etc.)	Strength of the card directly affects the player's chances of winning if no player folds until the end of round 2	Suit does not matter at all in the simplified winning conditions of this game version, therefore it can be completely omitted from state-space representation. Furthermore, the order of public cards does not matter in this variant, hence the alphabetical order offers a consistent way of reducing the number of states.
rank of remaining possible opposing player cards in hand in alphabetical order	Based on Table **??**: 'AJKQT' (no information), 'JQT', all 10 combinations of public hands, and all ranks (when the opposing hand's rank is surely known)	Needed only for Threshold agent, the strength of the opposing hand directly affects the player's chances of winning if no player folds until the end of round 2, and it also determines the probability of the Threshold agent's next action	When the opposing agent is not known to be the Threshold agent, only 'AJKQT' is used. See above too.

Table 1: Simplified poker state-space representation utilized by Policy Iteration and Q-learning algorithms.

4 Model-free solution: Q-learning algorithm

- Implementation
- Analysis of hyperparameter tuning
- Average Results
- Convergence analysis

5 Conclusion

- Main point for game size & performance of algorithms
- Suggested next steps/improvements