Reinforcement Learning of (stylised) Blackjack

Balazs Agoston Nyiro and Felipe Duque-Quiceno
School of Psychology, University of Nottingham, Nottingham, NG7 2RD, UK
(Dated: January 19, 2022)

Blackjack is a popular gambling game that can be played episodically, providing a great opportunity to implement reinforcement learning (RL) algorithms. In the present project, several RL methods were tested, training an agent with each of them to learn the policy it believed optimal for maximizing the overall score at the end of the game. Results suggest that learning converged to a local optimum policy for each method. Further analysis could be done to establish a comparison between the results here found, and real human performance.

I. Introduction

Blackjack is a popular casino game played with French decks of cards (52 cards per deck: 2-10 plus J, Q, K, A, times 4 suits). An unlimited number of decks can be used during the game. The value of a card is equal to the number written on it, while the Jack, Queen and King have a value of 10. The Ace can have a value of 1 or 11, depending on whether the current hand sum goes above or below 21 respectively [1]. In the simplest form of the game, a single player can decide to ask for another card (hit) or to stand at the current hand sum (stick), until all cards in the deck are dealt (end of the episode). If the number of decks is infinite, the episode ends after each hand. A hand ends when the player decides to stick, or when the current sum of their cards goes over 21. The hand score is given by equation 1, where C_i is the value of the i–th card in the player's hand.

$$S = \begin{cases} (C_1 + ... + C_{n+1})^2 & \text{if } \ (C_1 + ... + C_{n+1}) \leq 21 \\ 0 & \text{if } \ C_1 + ... + C_{n+1}) > 21 \end{cases} \tag{1}$$

The aim of the present project is to train an agent in an environment of varying complexity, following common reinforcement learning algorithms. During the game, the agent aims to maximize its hand scores during an episode.

II. Methods

Presented in this section is a discussion of the various reinforcement learning algorithms that we tried during the project. For all of them, the agent was trained using an ϵ–greedy method, with $\epsilon = 0.1$, which means that 10% of the actions where chosen randomly for exploration, while the other 90% were taken following the policy

$$\pi(s) = \underset{a}{\operatorname{argmax}}(Q_\pi(a,s)) \tag{2}$$

Hand scores were given as reward for the agent, meaning that all transitions $s \to s'$ had a reward of 0, unless s' was the end of the hand. In the latter case, the reward would be S (see equation 1).

A. Tabular Methods

As a first approach, the expected reward for each state-action pair $\mathbb{E}(\sum_i r_i | s, a)$, was approximated with iterative updates of a table $Q(a,s)$, where the agent's goal was to find the optimal value for each of Q's elements. Due to the stochastic nature of the environment, plus the high number of possible paths each episode could have, the agent's training was designed to follow a simple Monte Carlo method, only updating the state-action pairs visited at each episode. Two versions of this method were implemented, as stated below.

1. *Hand sum as state*

For the simplest version, the hand sum was the only information provided to the agent as state of the environment. In this case, $Q(a,s)$ was a 22x2 table (22 possible hand sums, 2 possible actions), for a total of 44 elements to learn.

2. *Usable aces*

Knowing that the Ace's value is reduced if the hand sum goes over 21, one could expect the agent to learn to *hit* with more confidence if the hand contains an Ace with value 11 (usable Ace) [1]. Giving this additional information to the agent, resulted in a table $Q(a,s)$ of size 88 (22 possible hand sums, 2 possibilities for usable ace, 2 possible actions).

B. Function approximation

Estimating the optimal state-action value $Q_{\pi^*}(a,s)$ by an approximate polynomial function $Q_w(a,s)$, allows us to train the agent with the less parameters, as well as train it in a generalized environment, where the number of decks might be either finite or infinite, and the state of the agent may be represented by variables of infinite domain. For this case, the state $\mathbf{s}$ was composed of the hand sum h, number of decks at play d, and the likelihood of the next card being a high card (10-Ace) P_{high}. Numeric conventions where set as follows: action $a = 1$ was interpreted as *hit* and $a = 0$ as *stick*; $d = 0$ was interpreted as infinite number of decks.

The function Q_w was designed in a way that allows an easy 2D visualization of what the learned parameters mean (Figures 2 and 3), while allowing a separate training for infinite and finite decks.

$$\begin{aligned} Q_w = \delta_{d0}[(1-a)(w_0 + w_1 h + w_2 h^2) + \\ a(w_3 + w_4 h + w_5 h^2)] + \\ H(d)[(1-a)(w_6 + w_7 h + w_8 h^2) + \\ a(w_9 + w_{10} h + w_{11} h^2) + \\ aP_{high}(w_{12} + w_{13} h)] \end{aligned} \tag{3}$$

$$H(d) = \begin{cases} 1 & \text{if } d > 0 \\ 0 & \text{if } d \leq 0 \end{cases} \qquad \delta_{d0} = \begin{cases} 1 & \text{if } d = 0 \\ 0 & \text{if } d \neq 0 \end{cases} \tag{4}$$

The agent trained over 19000 episodes, 5000 of them with an infinite deck. Learning was computed by Stochastic Gradient Descent (SGD) after each episode [1].

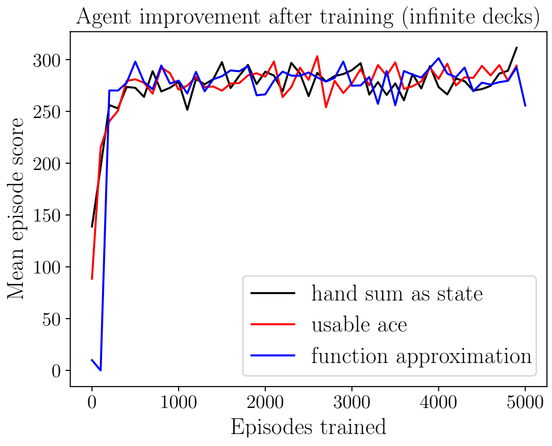

FIG. 1. The agent played a hundred episodes with each state-action function/table learned during training. When training with infinite decks, optimal policy converged early around the first thousand episodes played, independently of the method used to train –tabular (black and red), function approximation (blue).

III. Results

For both tabular cases, the agent converged to a policy of *hitting* if the hand sum was lower than 15, and *sticking* otherwise. If there was a usable ace in hand, the agent would *hit* for hand sums lower than 18. Performance over time can be seen in Figure 1.

For the function approximation method, the learned policy can be visualized in Figures 2 and 3. As the episode evolves, the likelihood of the next card being high (P_{high}) drifts the threshold where the agent passes from *hit* to *stick* (Figure 3).

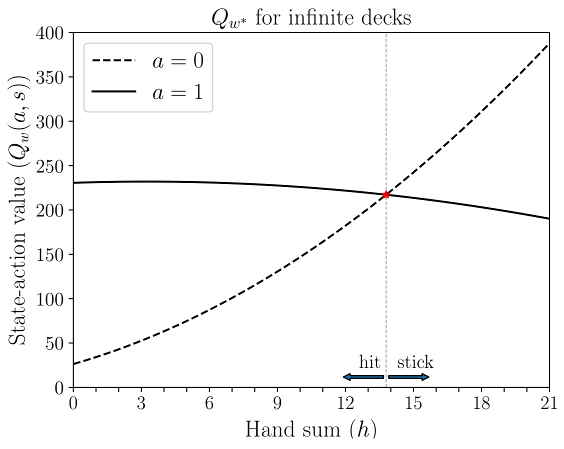

FIG. 2. Visualization of the learned policy for infinite decks, using polynomial function approximation method. The agent *hits* until the hand sum is over 13, then it *sticks*.

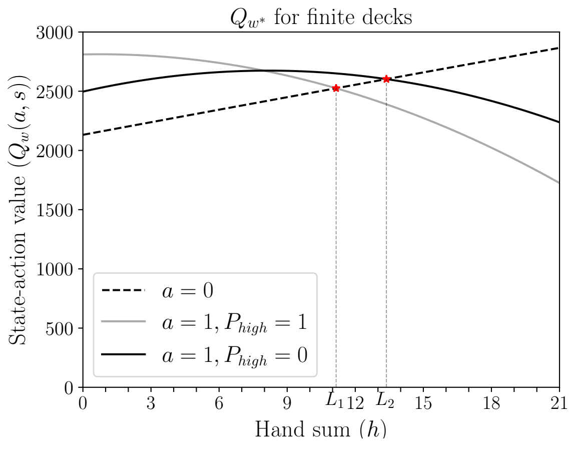

FIG. 3. Visualization of the learned policy for finite decks using polynomial function approximation method. P_{high} drifts the threshold for the agent to *stick*, between a lower and upper limit (L_1 and L_2).

IV. Conclusion

All the reinforcement learning algorithms here presented, suggest convergence to a locally optimum performance. Tabular Monte Carlo methods seem to have the same average performance than function approximation methods for infinite decks. Adding parameters to the state space allows for more intricate policies to be learned by function approximation methods, relevant for playing with finite decks and obtaining reasonably good scores. Further analysis should be done, to assess the performance of the trained agent here shown, in comparison to human performance.

[1] "Sutton & Barto Book: Reinforcement Learning: An Introduction." [Online]. Available: http://incompleteideas.net/book/the-book.html