Reinforcement Learning Blackjack

Dalila M. Islas Sanchez and Divyansh Oze
University of Nottingham

In this project, we use Reinforcement Learning, a learning technique based on finding the best actions given an environment and for the agent to complete an objective by learning from the rewards it gets over the trajectory for moving towards the optimal goal. We focus on training an agent to play a stylised version of blackjack, i.e. a famous card casino game in which the player is put against the dealer and needs a score of 21 or greater than that of the dealer's score to win the blackjack. In our case, the dealer is passive, and the agent attempts to reach a goal of having a score of 21, which is blackjack or a score closer to 21, as there are no opponents. We take the help of modelling frameworks and different learning methods like Markov Decision Processes, Monte Carlo and Q- learning to train our agent for different settings to get the maximum score possible with improving the policy and striking a balance between exploitation and exploration.

Introduction– The version of blackjack implemented in this project has slightly different rules than the traditional one played in the casino. We play with a particular assumption of having no opponent facing the player; the dealer is passive and just helps in dealing cards. Moreover, there are no splits or surrender moves like in the casino. The game is played with a number D of decks of 52 cards per deck; the suit for each card is irrelevant. All of the cards in the deck have a numerical value assigned to them, which is equal to their number of {2-10} and face cards {J, K, Q} having a value of 10 each, respectively. The aces in the game are valued as 11 if the sum of all the card values stays below 21, i.e., if $C_1 + \cdots + C_{n+1} \leq 21$ otherwise, its value is considered as 1. Our aim for this game is to maximise the value of each hand with playing under the given constraints.

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

Based on that, the environment will give the agent a reward of 0 for each hit with a score less than 17, 2 for a score close to or 21, and -1 for each sum of points greater than 21t.

RL Frameworks.– A standard Reinforcement Learning framework puts an agent to interact with the environment, which in return gives us some definite or indefinite states $\mathcal{S}_t \in \mathcal{S}$ to identify and know where we are in the given environment. Time would be taken as a discrete sequence of time steps [t = 0,1,2,3]. Now the agent has the liberty to make some actions in those states using its action state-space $\mathcal{A}(t) \in \mathcal{A}(s)$, given by the policy π, which maps the states to their respective actions π: $\mathcal{S} \rightarrow \mathcal{A}$ the response from the environment to this action is a reward, $R_{t+1} \in \mathcal{R}$.
This standard trajectory follows a sequence of events like this:

$$[S_0, A_0, R_1, S_1, A_t, R_2, S_2, A_2, R_3, \ldots.] \quad (2)$$

- **Markov Decision Processes:**

In a finite case of Markov Decision Processes (MDP), which is a standard framework for Reinforcement learning, values of variables like R_t and S_t have well-defined probability distributions to them that we use to calculate the state-transition probabilities or 'p' as the dynamics of the environment given below [1]:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \text{for all } s \in \mathcal{S}, a \in \mathcal{A}(t). \quad (3)$$

During the trajectory flow, the agent tries out different actions to maximise cumulative rewards. It is common to break the trajectory given in Eq(2) by defining the terminal time steps; for our project, we mainly take up episodic tasks, which are simply the sub-sequences for the interaction with the environment. In our case, the terminal time steps 'T' is defined for instances where the agent crosses a sum $C_1 + \cdots + C_{n+1} > 21$ and gets busted or decides to choose an action of A={stick} at a particular sum in hand. We define the total returns during the time in trajectory as $G_t = R_{t+1} + \gamma * G_{t+1}$, where γ is the discounted variable. Discounting helps us get a present value of rewards that we might see in the future states. MDP environments, therefore, have enough knowledge about the model. Bellman Equations are used with the help of Dynamic Programming procedures to evaluate and predict for $V_\pi(S)$ and $Q_\pi(S)$ to help reach their optimal state-value functions q_* (as shown in Eq 4) and improve for policy π^* using those updated V_π and Q_π. [1]

$$q_*(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \quad (4)$$

Note for MDP to find a good policy:

We take actions with regard to π to get some experience in MDP, obtain some reasonable estimates for Psa Later take up values for Q_π using Bellman equations and

then update the policy by choosing the maximum action for those state-action pairs, natural for the process to alternate between evaluating for Q_π and using the state-action pairs to suggest an improved policy.

- **Monte Carlo:**

Monte Carlo (MC) is a free-model method we use without any explicit knowledge or structure of the environment. It learns from the experience by letting the agent run a few episodes on the environment. It is particularly beneficial to estimate values for the state-action pairs, i.e. $Q_{(s,a)}$ when we do not have an available model.
Unlike MDP, we calculate the values for state-action pairs by averaging the sample returns. We follow the same policy iteration used in MDP and Dynamic Programming to evaluate Q_π using some arbitrary π, and further hope to attain optimality for $\pi \approx \pi^*$ and $Q \approx q^*$ as shown in Figure 1

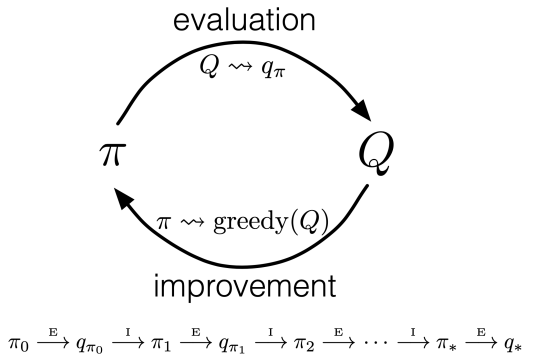

FIG. 1. **Evaluation-improvement.**

We deal with episodic tasks and further define any occurrence of states in an episode as a 'visit'. In the **first-visi**t MC method, we estimate $Q_\pi(s)$ as the average of returns following just one visit to the state 's', whereas every-visit MC would average the returns following all the visits to the state 's' [1].

- **Q- Learning:**

The Q-learning simplifies everything to a much greater level, while it still follows the procedure of visiting the state-action pairs and updating the Q(s, a). It guarantees the convergence for Q, with a probability of 1 to q^* is independent of the policy followed over the trajectory [1].

Methodology.–

The difficulty of this game depends on the number of decks involved. That is why it was separated into two settings: i) infinite and ii) finite, the latter being the most complex.

(i) Infinite decks $D \to \infty$

For the infinite setting, the card drawn from the the infinite deck is independent and identical to all the other cards. It's safe to assume that any information from the previous states is irrelevant, as we can not use make use of probability to get the outcomes for the current hand. To solve for this setting, we use a model-free approach using Monte Carlo Learning methods. Since, we do not have complete and accurate knowledge about the dynamics of the environment as shown in eq. 4. given with the working of Markov Decision Processes and Dynamic Programming. We estimate the Q_π values from averaging the sample returns. The experience of the agent in the environment must be divided into subsequences i.e. episodes, and at the end of each episode, the value of state-pairs are is estimated, and the policy is improved. For us the episodes terminate with time-step 'T', when either the sum goes above 21 and dealer gets busted, or when the agent decides to stick with a good enough score. Specifically, we use On-policy First-Visit MC control to solve for this setting, which helps bring a better solution to the unrealistic assumption of Monte ES (Exploration Starts). In ϵ-Greedy Policy improvement as shown in figure 1, all the non-greedy actions get by $\frac{\epsilon}{|A(s)|}$ and the remaining bulk of the probability, which takes the greedy actions are controlled by $1 - \epsilon + \frac{\epsilon}{|A(s)|}$. The ϵ-soft policy guarantees the greedy-policy π' $>> \pi$.

$$\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{if } a = A^* \\ \frac{\epsilon}{|A(s)|} & \text{if } a \neq A^* \end{cases} \tag{5}$$

(i) Finite decks $D < \infty$

Here, we have a finite number of decks as the name suggests and the probability of each draw is no longer independent of the previous states [memory].

Since better decisions can be made striking a balance between exploitation/exploration we make use of ϵ, which decays over time and leads the policy 'π' to go for more exploitation instead of exploration.

To solve for this setting we used, Q-learning (off-policy TD control) for estimating $\pi \approx \pi^*$. We implement the agorithm following the following pseudocode:

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

FIG. 2. **Q-learning (off-policy TD control).**

Conclusion/Results.– The result of this project was to manage to train an agent to play blackjack following the policy methods and learning mentioned above, plots of the quadratic score over episodes were obtained as we did the tasks for episodic events. It can be seen that it effectively maximizes the score, the Q-learning algorithm being the most effective and which converges faster.

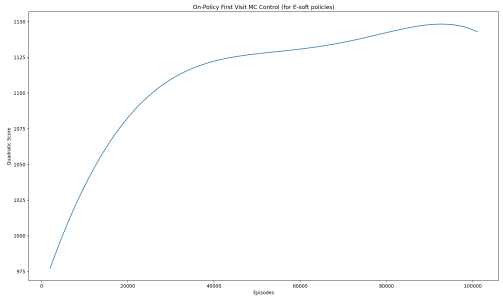

FIG. 3. **On-Policy First Visit MC Control (for E-soft policies).**

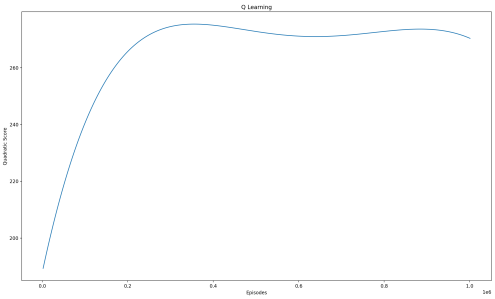

FIG. 4. **Q-Learning Off-Policy TD Control.**

[1] S. Sutton, (2018). Reinforcement Learning: An Introduction *MIT Press*.