Playing Blackjack with Reinforcement Learning

Albert Albesa[1], Kamyab Mirhosseini[1], Matthieu Paques[1]
[1] *University of Nottingham, School of Physics and Astronomy, Nottingham, United Kingdom*

(Dated: January 27, 2021)

This is a report for a Machine Learning project covering Reinforcement Learning. We have employed several techniques as Monte Carlo Exploration, Q-Learning, DeepQ-Learning or Policy Gradient in order to solve two proposed versions of the Blackjack game. Defining and constructing the environments according to a particular set of rules has been part of the project too.

Keywords: Reinforcement Learning, Blackjack, Machine Learning, Q-Learning, Policy Gradient

I. INTRODUCTION

In the recent years, Reinforcement Learning (RL) has been proposed as the building substrate for potentially simulating human-like intelligence. Moreover, the combination of this long-standing Machine Learning (ML) framework with the advances in some other techniques as Neural Networks (NN) has boosted the first in the scientific, technological and social planes.
In this report, we discuss both some of the most basic and recently published RL techniques by means of their practical application in solving a very well-known game: Blackjack.

II. ENVIRONMENT DEFINITIONS

Two versions of the game are proposed to solve: one where the cards dealt are replaced (current state does not affect the probabilities of next state, called *Infinite Deck*, (ID)) and one where there is a fixed number of decks (D) and the game lasts until all cards are dealt (*Finite Deck*, (FD)). We construct two environments, corresponding to each version of the game, by defining the different spaces and quantities found in Markov Decision Processes (MDP).

Markov Decision Process

A MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ in which an action $a \in \mathcal{A}$ taken from a state $s \in \mathcal{S}$, leads with probability $P_a(s, s^{'})$ to a state $s^{'}$, which has the Markov Property (MP), i.e. does not depend on states prior to s. This transition has an associated reward $r(s, s') \in \mathcal{R}$, and discount factor $\gamma \in [0, 1]$, which factors a reward in the future by γ^{n-1}, depending on the number n of transitions involved. In a MDP, $Q(s, a)$ and $V(s)$ are stochastic, and they are evaluated in terms of their expectation value.
Regarding our environments, the two versions of the game fix $\mathcal{A} = \{hit, stick\}$. At the same time, the nature of the deck fixes a correspondence between $\mathcal{S}$ and $\mathcal{P}$, and the game score has a natural relation to a reasonable reward space. In recent papers reviewing Blackjack in RL, γ is usually a value close to 1, in our case we set it to 0.999999.

Infinite Deck

The state space in the ID is defined by two elements, the current sum of the hand and the *usable_ ace*, which tells the agent whether the sum includes an ace with value 11 or not. This last element has an impact on penalty for aggressiveness, as surpassing 21 when *usable_ ace* is *true* will not immediately turn into a 0 return. The episodes in this environment are finished when a hand is finished, i.e. when the player sticks, reaches 21 or surpasses it. The reward obtained is the square of the sum, normalized by 441 (the maximum score obtainable in a hand).

Finite Deck (FD)

The state space is comprised by the current sum, *usable_ ace*, the accumulated score, and the number of cards left of each figure. The accumulated score is the sum of the squares of the scores obtained in previous hands, divided by 441.
This environment rewards the agent only when the episode finishes, with a value corresponding to that of the accumulated score.
Because we know the $(a + b)^2 \geq a^2 + b^2$, it is clear that the optimal strategy should always try to maximize each hand (so, trying to reach 21 if is still possible, 20 if not, and so on). The difficulty of this environment definition is that the agent has to learn both the described episode strategy and the optimization strategy for each hand.

III. METHODS

A. Exploration

We propose an ϵ-greedy policy that, together with a parameter in each of our agents: $model\_free = \{true, false\}$ explores according to Algorithm 1

Algorithm 1 Exploration

```
1: procedure EXPLORATION($s, \epsilon$)
2:     initialize random $\epsilon_0$ in $[0, 1]$
3:     if $\epsilon_0 < 1 - \epsilon$ then
4:         if not agent.model_free then
5:             $markov\_dict \leftarrow \{s : P(s, s')\}$
6:             initialize $\mathrm{E}[Q_\psi(s, 0)] = 0$
7:             for $s'$ in markov_ dict do
8:                 $\mathrm{E}[Q_\psi(s, 0)] \leftarrow \mathrm{E}[Q_\psi(s, 0)] + P(s, s') max_a Q_\psi(s', a)$
9:             end for
10:            $a \leftarrow argmax_{a'}\{\mathrm{E}[Q_\psi(s, a' = 0)], Q_\psi(s, a' = 1)\}$
11:        else
12:            $a \leftarrow argmax_{a'} Q_\psi(s, a')$
13:        end if
14:    else
15:        choose $a$ randomly
16:    end if
17:    return a
18: end procedure
```

B. Updating Techniques In order to achieve learning in the different environments, we use 4 main approaches: Monte Carlo, Temporal-Differences, Policy Gradient and *Pen and Paper*.

B.1 Monte Carlo This method computes a $Q_\psi(s,a)$ table by randomly exploring the action-state space. $Q_\psi(s,a)$ is initialized at zero and is then updated to be the average of the different returns observed:

$$Q_\psi(s,a) = \frac{1}{N}\sum_{i=1}^{N} R_i = \frac{1}{N}\sum_{i=1}^{N}\sum_{j(i)} \gamma^j r_{t+k+1} | s_t = s \quad (1)$$

where i corresponds to every observation of (s,a) and j to the times following that observation.

B.2 Temporal Differences (TD): When $Q_\psi(s,a)$ is an approximation of Q(s, a), the value of

$$\delta(t) = r + \gamma \cdot \max_a [Q_\psi(s_{t+1}, a)] - Q_\psi(s_t, a) \quad (2)$$

is different to zero, corresponding to a difference between the $Q_\psi(s,a)$ of the moment and a new observation. When model-based,$[Q_\psi(s_{t+1}, 0)]$ is substituted by $\mathrm{E}[Q_\psi(s_{t+1}, 0)]$, computed as in lines 5 - 9 in Algorithm 1. An approach to obtain an improved $Q_\psi(s,a)$ is trying to minimize the MSE of this quantity. 3 agents use this approach:

B.2.1 Q-Learning Agent: The Q-Learning agent makes use of the TD to minimize the difference between $Q_\psi(s,a)$ and $Q(s,a)$, by updating it according to:

$$Q_{\psi(t+1)}(s,a) = (1-\alpha) Q_{\psi(t)}(s,a) + \alpha[\delta(t) + Q_{\psi(t)}(s,a)] \quad (3)$$

where α is the learning rate.

B.2.2 DeepQ-Learning Agent: This agent uses a NN as an approximator of $Q(s,a)$. We have been inspired by the implementation done in the famous article *Playing Atari Games with Deep Reinforcement Learning* [1]. The input of the NN is the state-action pair and the output is a vector function of the dimension of the action space. For each (s,a) observed, a label is constructed as follows:

$$y(s,a) = r + \gamma \cdot \max_a [Q_\psi(s_{t+1}, a)] \quad (4)$$

and the loss defined as the Mean Square Error (MSE). The NN is trained through experience replay. This is done by storing $((s,a), y(s,a))$ tuples in a memory, and training by means of stochastic gradient descent randomly sampling a *batch_ size* number of elements in the memory. The memory has a limited capacity, so that after a certain number of elements stored, it starts forgetting the oldest ones in order to remember new observations.

B.2.3 Quantum Agent: Recent papers [2] have explored the use of Quantum Variational Circuits (QVC) as function approximators in Deep Reinforcement Learning. We have only tested the Quantum Agent against the ID, due to computational limitations. In such context, there is a two channels variational circuit, accepting as input an encoded state $(q(usable\_ace), q(current\_sum))$ and with output the averaged measurement of these channels. The loss is also the MSE associated to the labels defined in (4); the gradient of the QVC is computed using the shift-parameter rule:

$$\frac{d}{d\theta} f(\theta) = r\big[f(\theta + \frac{\pi}{4r}) - f(\theta - \frac{\pi}{4r})\big] \quad (5)$$

where f is the measurement of the circuit, θ a variational parameter and r a constant that depends on the eigenvalues of the gate associated to θ (kept constant to 1/2 in our experiments).

B.3 Policy Gradient (PG) This technique intends to maximize the probability to chose the action with highest return. In this context the neural network (or in general the approximator) is directly updated:

$$\theta = \theta + \alpha \nabla \mathrm{E}_\pi \big[R_t\big] \;= \theta + \alpha G_t \nabla_\theta log(\pi_\theta) \quad (6)$$

We propose two different architectures: one with a sigmoid output and one with a two output softmax function. At the same time, we also use two different methods for computing the returns:

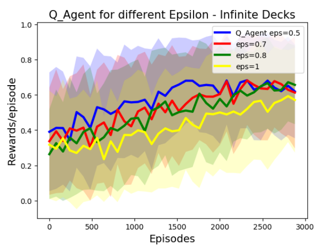
- Reinforce: $G_t = R_t = \sum_{t=t'}^{T-1} \gamma^{t'-t} r_{t'}$
- TD: $G_t = r_t + \gamma \sum_{t'=t+2}^{T-1} \gamma^{t'-t} r_{t'} - \sum_{t'=t+1}^{T-1} \gamma^{t'-t} r_{t'}$

which gives a total of 4 different combinations for the PG Agent.

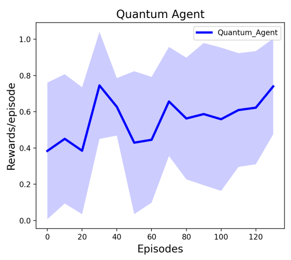
B.4 Pen and Paper (PP) This agent plays with a predefined policy obtained with analytical procedures.
For the Infinite Deck, it is possible to recursively compute the $\mathrm{E}[Q(s,a)]$ of the full action-state space, given that the reward for sticking, reaching 0 or 21 is deterministic and the probabilities of transition between states is known.
In the case of the Finite Deck, as advanced in section (II), the general strategy comes from knowing that the total reward depends on the squares of the scores obtained at each hand. We did transform it into an optimization problem and solved it numerically with *scipy* package, and obtained that the best possible episode (with probability in your favor) is achieving 18 hands of 21 and one single hand with a 2, which implies a score of 7942 and a reward ≈ 18.
With a state space of the order of 10^{12}, calculating the optimal policy for this environment is computationally intractable. Nevertheless, we modified the PP Agent to, at each observation made, compute the optimal policy for an infinite deck in which the probabilities of each card are given by the current state of the deck. This is an approximation, given that does not take into account the fact now the probabilities change within the recursive iterations.

IV. RESULTS AND DISCUSSION

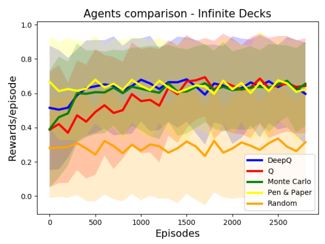
Learning Curves We will briefly review some interesting observations of the Learning Curves (LC) in figure 1. In general, all agents succeed in achieving a policy close to the optimal (figures 1b, 1c, 1d). In figure 1a, we experimentally find the best initial ϵ to be 0.5. In the case of the FD, the DeepQ (model-based) agent fails to achieve PG and PP policy for any of the tried learning rates, and model-free fails to learn at all (figure 1e). We can also see how an increasing number of decks results in poorer learning (rewards normalized by maximum possible reward, figure 1f).



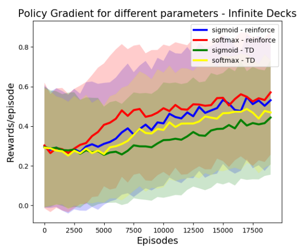
(a) learning curves for Q-Agent and different initial ϵ (ID)



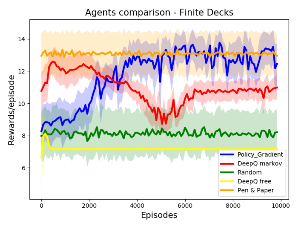
(b) learning curve for the Quantum Agent (ID)



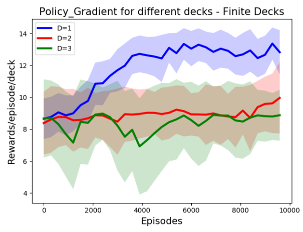
(c) learning curve of optimal MC, Q and DeepQ Agents (ID)



(d) learning curve for different PG Agents (ID)



(e) learning curve for optimal DeepQ and PG Agents (FD)



(f) learning curves for different number of Decks

Figure 1. Learning curves for different experiments. Thick line is the average reward obtained and the shadow indicates variance (both computed over a fixed number of episodes).

Learning Metrics

- $\%\pi$ is the percentage of states in which the action given by $Q_\psi(s,a)$ coincides with the optimal policy.
- MSE(Q) measures the mean squared error of the $Q_\psi(s,a)$ with respect to $Q(s,a)$ (for visited states).
- E(π) sums the absolute value of the difference in $Q(s,0)$ and $Q(s,1)$ for the states for which the derived policy was wrong.
- score averages over 10^5 episodes, with the exception of the Quantum Agent (10^4) and PP FD (10^3)

First three metrics were evaluated after training for 3000 episodes, with the exception of the Quantum Agent, due to computational limitations (150), and only for ID. The distances to $Q(s,a)$ indicate how the models are better at predicting π than Q. We checked that the policy derived from Q goes from the indicated $\%\pi$ to 100% when the agent chooses according with algorithm 1, as the decision is averaged over all the possible future states and the error damped. Scores show clear signs of learning when one compares trained agents to random playing (for both environments).

Conclusions and Future Work

We have successfully built and compared different RL algorithms in order to train agents to play Blackjack, as well as experienced the process of doing research in order to solve a problem from scratch. Results, although always improvable, show the agents are able to learn and improve their policies by training.
Further research could include, for instance, improving DeepQ learning algorithm by implementing DoubleQ method, as well as allowing the model to build decision trees that allow it to see further steps in the future.

Table I. Parameters and metrics for each agent

	Infinite Deck				Finite Deck
Agent	% π	MSE(Q)	E(π)	score	score
Expression	$\frac{\#(\pi_\psi(s)=\pi(s))}{0.01\#(s)}$	$\frac{\sum\left(Q_\psi(s,a)-Q(s,a)\right)^2}{\#(s \text{ visited})}$	$\sum\lvert Q(s_w,0)-Q(s_w,1)\rvert$	$\overline{x}\pm t_{0.95}\frac{\sigma}{\sqrt{N}}$	-
Monte Carlo	83	0.039	0.81	0.6240 ± 0.0016	-
Q-Learning	72	0.145	0.62	0.6412 ± 0.0017	-
DeepQ-Learning	83	0.054	0.84	0.628 ± 0.0017	10.867 ± 0.0018
Quantum	86	0.174	0.42	0.6347 ± 0.0049	-
Policy Gradient	90	-	0.19	0.6397 ± 0.0016	13.230 ± 0.019
PP	100	0	0	0.6409 ± 0.0017	13.090 ± 0.084
Random	-	-	-	0.2836± 0.0018	8.124 ± 0.029

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

[2] Chen, S.Y.C., Yang, C.H.H., Qi, J., Chen, P.Y., Ma, X. and Goan, H.S., 2020. Variational quantum circuits for deep reinforcement learning. *IEEE Access, 8, pp.141007-141024.*