Reinforcement Learning Project

F. ABEILLON, J-P. CHOU, N. GYS

Code available at: https://gitlab.binets.fr/nicolas.gys/ia-blackjack

***Abstract* — Blackjack is a very popular card game, played in casinos all over the world. It is very (un)famous for being one of the only gambling games in which the player's expectations can beat the casino's, thanks to cards counting. In this paper, we try to train an Artificial Intelligence to play Blackjack and to develop a winning strategy. Our AI was trained with various Reinforcement Learning techniques – Q-Learning, Double Q-Learning, Deep Q-Learning and a Neural Network approach; we then compared the results with those from simpler strategies.**

***Keywords* — Blackjack, Reinforcement Learning, Artificial Intelligence, Q-Learning, Double Q-Learning, Deep Q-Learning, Neural Network, cards counting.**

INTRODUCTION

The rules of Blackjack [1] are rather simple: the goal of each game is to have a *hand score* higher than the dealer's, without exceeding 21 – the value of each card being its nominal one, except for the faces (10-value) and the aces (11 "*soft ace*" value; or 1, "*hard ace*" value, if the *soft hand* score exceeds 21).

At the beginning of each game, every player gets two cards, one facing up and one facing down, and so does the dealer. Then begins the first pass: each player will successively show his/her second card and then be asked whether to *hit* (to draw a card) or to *stand* (to keep his/her hand as it is). After one or several passes, when all the players have either *stood* or *busted* (meaning that their *hand scores* exceeded 21), the dealer reveals his/her second card and *hits* as many times as needed to have a *hand score* equal or higher than (a *soft*) 17. Note that the players simultaneously compete against the dealer and not against each other.

The player:

- wins if he/she has not *busted*, but the dealer has; or if he/she has a *hand score* strictly higher than the dealer's (the player not having *busted*);
- draws with the dealer if he/she has a *hand score* equal to the dealer's (the player not having *busted*);
- loses if he/she has *busted*; or if he/she has a *hand score* strictly lower than the dealer's (the dealer not having *busted*).

Special case: a *hand score* of 21 with the two first cards is called a *natural blackjack*, and beats any dealer's *hand* – except another *natural blackjack*.

Note that there are additional rules (*double*, *separate*...); however, we'll only use the hereinabove ones to create our (not too intricate) environment. Only one player will take part of the environment in this project.

The purpose of this project is to build an autonomous agent capable of playing Blackjack while minimizing its losses – and eventually of designing a strategy with a positive expected value.

It is important to note the real asymmetry in favor of the dealer, due to the playing order (the player can *bust* without the dealer even needing to play) and of the lack of information on the full dealer's *hand*. The stochastic nature of Blackjack and its huge number of combinations may look like an impossible optimization task – even for Machine Learning algorithms. However, there is a particularity in the way Blackjack is played in a real-world scenario: along the games, cards are not shuffled until the deck is (nearly) empty. This particularity makes it possible to count cards and to develop a strategy which can reverse the advantage in favor of the player.

In addition, the simplicity of the dealer's strategy and the power of ML algorithms may be taken advantage of to possibly win more games than the dealer. Indeed, ML is particularly interesting in the real-world scenario where cards are not replaced but discarded after being drawn (from now on we refer to this case as the "w*ithout replacement* scenario"), as we can exploit the huge memory of machines to implement a very efficient cards counting strategy.

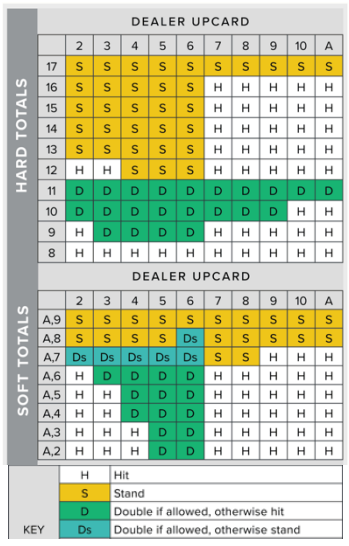
As the environment is partially observable (we do not know the actual reward function), we chose a *model-free* approach. Several techniques were tested in this project:

- Q-Learning [2, 3, 4]
- Double Q-Learning
- Deep Q-Learning
- Neural Network [5]

Note that each method - except the Neural Network - comes with a decaying ε-greedy policy to balance exploitation and exploration.

In order to compare the results of our autonomous agents with simpler strategies, we have built 4 basic algorithms:

- Random: each action is selected at random;
- Stochastic-without-memory: each action is selected according to the information available at each game (namely the cards in the algorithm's *hand* and the dealer's facing up card – but not the cards drawn in previous games), as the algorithm draws the probability of *busting* if it *hits*;
- Range actions: the algorithm *hits* if its current score is equal to or lower than a certain threshold;
- Basic Strategy: as Blackjack is a gambling game, it is no surprise that people developed optimal moves according to every possible state. The table below sums up the optimal strategy to follow depending on the player's *hand score*, whether he/she has a *soft ace*, and the dealer's facing up card. Thus, we implemented the Basic Strategy adapted to our environment (i.e without the possibility of *splitting* and *doubling*).



HARD TOTALS

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
17	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	D	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H

SOFT TOTALS

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
A,9	S	S	S	S	S	S	S	S	S	S
A,8	S	S	S	S	Ds	S	S	S	S	S
A,7	Ds	Ds	Ds	Ds	Ds	S	S	H	H	H
A,6	H	D	D	D	D	H	H	H	H	H
A,5	H	H	D	D	D	H	H	H	H	H
A,4	H	H	D	D	D	H	H	H	H	H
A,3	H	H	H	D	D	H	H	H	H	H
A,2	H	H	H	D	D	H	H	H	H	H

KEY

H	Hit
S	Stand
D	Double if allowed, otherwise hit
Ds	Double if allowed, otherwise stand

Figure 1 – Basic Strategy table. Source: https://www.blackjackapprenticeship.com/blackjack-strategy-charts/

We compared all those algorithms in two different environments:

- The "*with replacement*" scenario: the perfect scenario, as you may encounter in online tournaments, with a purely random deck – as if the whole deck was mixed at the end of every round;
- The "*without replacement*" scenario: the real-world scenario, with a finite number of decks (we used only one deck in this project, in order to make it easier to count) – the cards used in a game are not put back in the deck, allowing players to try to count cards.

BACKGROUND AND RELATED WORK

We are not the first ones to think of applying RL techniques to Blackjack. However, the rules applied from hey casino to another different and changed many times (minimum and maximum bet, *doubling*, *splitting*, number of players...). And even slight changes to the rules can lead to utterly different results. We chose to work on a very specific case – as described hereinabove – but here are some state-of-the-art results:

In *Teaching A Neural Net To Play Blackjack* [5], T. Yiu reached with a Neural Network an expected value of – 0.07 with the same rules as us, and worked on the probability of tie or win based on the player's *hand value*.

C. de Granville, from the *University of Oklahoma*, in his paper *Applying Reinforcement Learning to Blackjack Using Q-Learning* [3], applied Q-Learning to Blackjack. Again, his results never exceeded the Basic Strategy, which shows all the difficulty to optimize this game. In addition, both the Basic Strategy and the Q-Learning failed to beat the casino in his implementation, with negative expected values.

A. Wu, from *Stanford University*, applied Deep Q-Learning to Blackjack [6]. He implemented an environment with all the rules, and reached an expected value of – 0.107, whereas the Basic Strategy expected return is –0.096.

Thus, these three examples demonstrate that, even if RL algorithms can approximate the Basic Strategy (which is the optimal policy if the player does not count cards), they can never beat it. Nonetheless, these articles were not exploiting the possibility to count the cards.

In fact, counting cards enables to drastically reduce the casino's advantage. In some cases (with certain rules and enough players), it is theoretically possible to beat the casino and to have a positive expected value [7]. Hans Hellemons' paper gives full details about the mathematics behind various strategies in a lot of different cases.

THE AGENT AND THE ENVIRONMENT

"*WITH REPLACEMENT*" SCENARIO

In this scenario, we used *GYM*'s "*blackjack-v0*" environment. This environment is stochastic and partially observed.

The state space is composed of 18 (possible *hand scores*, from 4 to 21) * 2 (whether the agent has a *soft ace* or not) * 10 (dealer's facing up card) = 360 dimensions. We can reduce the number of dimensions by noting that one cannot have a *hand score* lower than 12 and have a *soft ace* - hence (8 + 10 * 2) * 10 = 280 dimensions. To speed up our agent's learning process, we added at one point the possibility to force the agent to *hit* whenever its *hand score* is equal or lower than 11 – as it cannot *bust* with such *hand scores* - and to *stand*

whenever it is equal to 21 (*soft* or *hard)*: then, there would be 9 * 2 * 10 = 180 dimensions in the state space.

The actions are to *hit* and to *stand*. We did not implement the *double*/*split*/*surrender*/*insure* actions.

The possible rewards are +1 (the player wins), 0 (draw) or -1 (the player loses). A *natural blackjack* is also rewarded by +1.5, so as to prevent the agent from having the idea to *stand* in that state.

The cards that have been drawn are immediately put back into the deck so that the distribution never changes from game to game (equivalent of drawing a card from an infinite number of mixed decks); the player cannot count cards over several games in this case.

"*WITHOUT REPLACEMENT*" SCENARIO

This scenario is pretty much the same as the previous one; the only difference lies on the non-absolute randomness of the deck. Indeed, every card that has been drawn will be discarded and won't be drawn again. We can now take advantage of the previous games in order to count cards. Here are some strategies to do so:

- The most basic strategy is to remember every card that was ever drawn, and to use that to compute the probabilities to draw each card;
- The Hi-Lo strategy from E. Thorp [8]: the idea is to attribute a score to the deck. We start from 0 and increment the score every time a card is drawn according to its value (-1 for an ace, a face card or a 10; +1 for 2, 3, 4, 5, 6; 0 for the rest). The higher the score is, the more advantageous it is for the player.

In this case, the state space grows during the game. If we call n the number of decks used, the number of dimensions is $(4n+1)^9$ (the number of aces that have been drawn – between 0 and $4n$ – times the number of 2 that have been drawn, etc.) $*$ $(16n+1)$ (the number of 10-value cards that have been drawn) $\sim 4 * (4n)^{10}$ times the "*with replacement*" scenario one (360 * 4,5.10^6 ~ 1,5 billion dimensions with n = 1; 360 * 2,5.10^{14} ~ 91 millions of billion dimensions with n = 6; obviously, the more decks there are, the hardest it is to count cards as the environment expands exponentially). Therefore, counting cards is not compatible with every RL technique (namely "basic" Q-Learning).

Moreover, the number of players greatly influences the counting. The more players there are, the more information one can have about the deck. Still, we will limit our study to the simplest case: only one player – and a single deck.

THE AGENTS

Q-LEARNING

The principle of Q-Learning is to explore the environment to map the possible couples (state, action) with an action-value function Q. Given a state s, the more likely an action a is to lead to a high reward (i.e win), the higher $Q[s, a]$ is:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid \pi, s_0 = s, a_0 = a\right]$$

$Q^{\pi}(s, a)$ corresponds to the expected gain when doing the action a from state s, and then following policy π.

As the "*without replacement*" scenario spaces state is not that big, we play a great number of games, so as to be able to have a rather good estimation of the action-value function. Still, the "*with replacement*" scenario states space is far too big to even think of implementing Q-Learning in this case: that is why the "*with replacement*" scenario will only be explored with Deep Q-Learning and the Neural Network approach.

We then implemented an autonomous agent based on Q-Learning to apprehend Blackjack. We trained our agent for 100,000 games (which is far more than the states space dimension). After several tweaks of the various parameters, we came up with the following empirical values to optimize our results: initialize the exploration rate at 1, and start its decay (at the rate of 0.1% per game) until the 3,000th game (the agent then explores the environment and gathers information) before turning it to 0 (the agent then follows a greedy policy).

When a game is done, Q is updated retroactively – meaning that each action (and not just the final one) taken during the game will lead to an update of Q, and each update will use the final reward. We used as the learning rate α = 0.115 (also decaying at the rate of 0.1% per game) and as the discount factor γ = 0.15, as we found those values gave the best results when updating Q according to the following formula:

$$Q[s,a] := (1-\alpha)Q[s,a] + \alpha\left(r + \gamma \max_{a'} Q[s',a']\right)$$

After the training phase, the agent has built a table Q. We then injected it into another agent, that will follow a greedy policy using this pre-computed table (it does not explore other options) for 100,000 games.

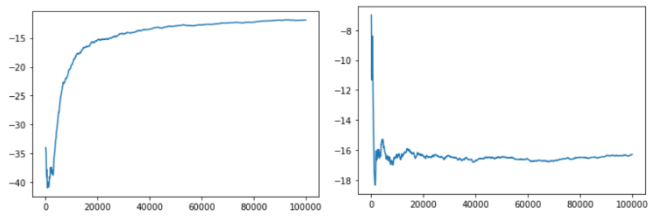


Figure 2 – Learning curve for Q-Learning during the training phase (on the left-hand side), and during the testing phase (on the right-hand side).

At the end of the testing phase, our agent is able to win around 38% of games, and loses 54% of them (the rest is ties). We will use the expected return (i.e the probability to lose minus the probability to win) as the main measure of our agent's performance: in this case, the expected value is –0.16.

We extracted the strategy our agent obtained – through the *Q* table: below are two tables, which represent the actions our agent will do given its *hand score* and the dealer's facing up card, if the player has a *soft ace* or not:

With a hard ace

	2	3	4	5	6	7	8	9 10 or any face	ace
4	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
5	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
6	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
7	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
8	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
9	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
10	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
11	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
12	HIT	STAND	HIT	HIT	HIT	HIT	HIT	HIT	STAND	HIT
13	HIT	STAND	HIT	HIT	HIT	HIT	HIT	STAND	HIT	HIT
14	HIT	HIT	STAND	HIT	HIT	HIT	STAND	HIT	HIT	HIT
15	HIT	HIT	HIT	HIT	HIT	STAND	STAND	HIT	HIT	HIT
16	HIT	HIT	STAND	HIT	HIT	HIT	STAND	HIT	HIT	HIT
17	STAND	STAND	HIT	STAND	STAND	HIT	STAND	HIT	HIT	HIT
18	STAND	STAND	HIT	HIT	STAND	HIT	STAND	HIT	HIT	STAND
19	STAND	STAND	STAND	HIT	STAND	STAND	STAND	HIT	HIT	HIT
20	HIT	STAND	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT
21	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

With a soft ace

	2	3	4	5	6	7	8	9 10 or any face	ace
12	HIT	HIT	HIT	HIT	STAND	HIT	HIT	STAND	STAND	STAND
13	HIT	HIT	STAND	HIT	HIT	STAND	STAND	HIT	HIT	HIT
14	HIT	HIT	STAND	HIT	HIT	HIT	HIT	HIT	HIT	HIT
15	HIT	STAND	HIT	HIT	STAND	STAND	STAND	HIT	HIT	STAND
16	HIT	HIT	HIT	STAND	STAND	HIT	HIT	HIT	HIT	HIT
17	HIT	STAND	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT
18	STAND	HIT	HIT	STAND	STAND	HIT	HIT	HIT	HIT	STAND
19	STAND	HIT	STAND	HIT	HIT	HIT	STAND	HIT	HIT	HIT
20	HIT	HIT	HIT	STAND	HIT	HIT	HIT	HIT	HIT	HIT
21	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

Figure 3 – Q *table built by our agent. You can see the preferred action depending on the agent's* hand score *(rows) and the dealer's facing up card (columns).*

We easily see that this *Q* table is not optimal, as for example the agent prefers to *hit* even if its *hand score* is close to 21 (especially when it has a *soft ace* – which is not absurd, but clearly not optimal).

As we have access to the Basic Strategy table, let us feed our agent with the Basic Strategy *Q* table and see the results: the agent is now able to win 43.1% of games, and loses 47.5% of them, leading to an expected return of -0.044. We can introduce a new measure, which is the percentage of different actions between the learnt *Q* table and the optimal one: we have with Q-Learning a 23% error rate.

With a hard ace

	2	3	4	5	6	7	8	9 10 or any face	ace
4	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
5	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
6	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
7	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
8	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
9	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
10	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
11	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
12	HIT	HIT	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT
13	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT
14	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT
15	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT
16	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT
17	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
18	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
19	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
20	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
21	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

With a soft ace

	2	3	4	5	6	7	8	9 10 or any face	ace
12	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
13	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
14	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
15	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
16	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
17	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
18	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT	HIT	HIT
19	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
20	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
21	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

Figure 4 – Basic Strategy Q *table.*

In order to help our agent and to reduce its convergence time, we tried to give him a "boost" by telling him to *hit* whenever its *hand score* is equal or lower than 11, and to *stand* whenever it is equal to 21; yet, the results were not concluding, as the convergence time was equivalent. In addition, we wanted to see if a change in the reward function would increase our agent's performance (either making the agent risk-averse, or risk-taking by augmenting the negative/positive rewards; we also tried to reward with 0.5 the *hit* without *busting*), but we eventually reached no conclusive results.

DOUBLE Q-LEARNING

The main inconvenient with Q-Learning is that it tends to overestimate the value of the actions, which tends to slow the overall learning. Double Q-learning uses two different action-value functions, making different experiments; the update is then crossed between the two.

We implemented this technique, with the following formula :

$$Q^A_{t+1}(s_t,a_t) = Q^A_t(s_t,a_t) + \alpha_t(s_t,a_t)\left(r_t + \gamma\, Q^B_t\left(s_{t+1}, \arg\max_a Q^A_t(s_{t+1},a)\right) - Q^A_t(s_t,a_t)\right)$$

$$Q^B_{t+1}(s_t,a_t) = Q^B_t(s_t,a_t) + \alpha_t(s_t,a_t)\left(r_t + \gamma\, Q^A_t\left(s_{t+1}, \arg\max_a Q^B_t(s_{t+1},a)\right) - Q^B_t(s_t,a_t)\right)$$

However, and even after several tweaks of the different parameters (exploration rate and its decay rate, learning rate and its decay rate, discount factor, boost, rewards…), the agent is winning only 36% of games, and losing 55% of them, for an expected value of -0.19 (the built *Q* table vary on 30% of the actions, compared to the Basic Strategy table).

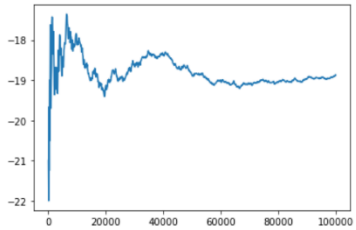


Figure 5 – Learning curve for Double Q-Learning (with the same parameters as with Q-Learning).

As it seems that the agent has a hard time "un-learning" an action that led to a win once, but that was not the best move overall. Yet, as the learning rate decreases over the games, it is hard for the agent to compensate this "youthful error"; as there are two tables in Double Q-Learning, it might be even harder for the agent to compensate such errors.

NEURAL NETWORK

We also implemented an atypical method using Deep Learning. We first worked in the "*with replacement*" scenario. We set up a Fully-Connected Neural Network that takes the state of the game as its input (the agent's *hand*, whether the agent has a *soft ace*, and the dealer's facing up card), and outputs the best action. The training database was not trivial to build as "which action is the best" is hard to define: indeed, no action enables the agent to win in some particular cases. We first ran 100,000 games based on random actions that led to a win. We then let the Neural Network develop its own strategy by learning from these successes, but it proved to be ineffective: 38% of wins / 7% of ties / 55% of loss / -0.17 of expected value.

Layer	Number of units	Activation
Dense	16	ReLU
Dense	256	ReLU
Dense	128	ReLU
Dense	32	ReLU
Dense	8	ReLU
Dense	1	Sigmoid

Figure 6 – Architecture of our Neural Network

We guessed that these disappointing results came from the fact that the games won were not systematically representative of the winning strategy (kind of the same problem as with Q-Learning). By playing randomly, the algorithm learnt to *hit* at 17, and *stood* at 10 for instance. Like in T. Yiu's work, we helped the agent by providing the correct action to the Neural Network by taking the first round of 100,000 games based on random actions but keeping all of them (won and lost), and taking the other action (*hit* if it *stood*, *stand* if it *hit*) if the game was lost. This way, every case was well represented. After tuning the hyperparameters (ADAM optimizer, MSE Loss with 20 training epochs), this strategy enabled us to get better results: 42.5% of wins, 8.4% of ties, 49.1% of loss on over 100,000 games, which gives us an expected value of –0.066.

In the "*without replacement*" scenario, with a single deck and a single player, we added additional inputs to teach the Neural Network to count the cards: the Hi-Lo score, and a list that stored the number of times each card was drawn. Using the Hi-Lo score as the input was the most efficient approach: with the same hyperparameters as before, we obtained 44% of win, 8.2% of ties 47.8% of loss for an expected value of –0.038 (for 100,000 games) – and an area under the ROC Curve of 0.668. Despite the good results, we must not forget that the expected value is still negative and that even the best plays not always lead to a win.

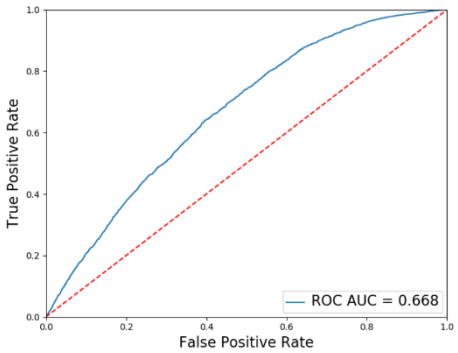


Figure 7 – ROC Curve of our Neural Network

The other features did not enable us to get better results, probably because the number of trained games were not sufficient (= 100,000 << 10^{15}). Our computers were unfortunately not fast enough to train the Neural Network with more data and to verify this hypothesis.

DEEP Q-LEARNING

Deep Q-Learning combines the upper method with Q-Learning. The idea is that, as a Neural Network is able to approximate any continuous function, ours will be able to approximate the action-value function *Q*.

As the dimension of our state space is quite big, it can be complicated to get the best *Q* table with Q-Learning. Thus, Deep Q-Learning makes it possible to improve the results of "basic" Q-Learning. In addition, compared to the Neural Network approach, the Deep Q-Learning Neural Network learns from a bigger set of data.

We choose a network organised as following:

1. Input: 3 units (4 if we implement cards counting)
2. Fully-Connected layer: 39 units (ReLU activation)
3. Batch Normalization
4. Fully-Connected layer: 9 units (ReLU activation)
5. Batch Normalization
6. Fully-Connected layer (Output): 4 units

We trained our Deep Q-Learning Neural Network with 10 epochs of 1,000 games each. This training was done twice: once without a cards counting implementation, and once with one. Counting cards was done the same way as in the Neural Network approach (Hi-Lo strategy).

We obtained very good results:

	Wins	Ties	Losses	Expected return
No cards counting	44%	7.5%	48.5%	-0.045
Cards counting	46%	6.9%	47.1%	-0.011

Figure 8 – Results of our agent with Deep Q-Learning

These are the best results we obtained overall. Note that we did not beat the Basic Strategy – which is the optimal policy – without cards counting. This is logical: our network is only able to approximate this optimal policy. As we believed, cards counting improves the results. However, it is to be noted that we did not succeed in obtaining a positive return, even with this promising technique.

With a soft ace	2	3	4	5	6	7	8	9	10 or any face	ace
4	HIT	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT
5	HIT	HIT	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT
6	HIT	HIT	HIT	STAND	STAND	STAND	STAND	STAND	STAND	HIT
7	HIT	HIT	HIT	STAND	STAND	STAND	STAND	STAND	STAND	HIT
8	HIT	HIT	HIT	STAND	STAND	STAND	STAND	STAND	STAND	HIT
9	HIT	HIT	HIT	STAND	STAND	HIT	HIT	HIT	HIT	HIT
10	HIT	HIT	HIT	STAND	STAND	HIT	HIT	HIT	HIT	HIT
11	HIT	HIT	HIT	STAND	HIT	HIT	HIT	HIT	HIT	HIT
12	HIT	HIT	HIT	STAND	HIT	HIT	HIT	HIT	HIT	HIT
13	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT
14	HIT	STAND	STAND	STAND	STAND	HIT	HIT	HIT	STAND	HIT
15	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT
16	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT
17	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	HIT	STAND
18	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
19	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
20	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

With a hard ace	2	3	4	5	6	7	8	9	10 or any face	ace
4	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
5	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
6	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
7	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
8	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
9	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
10	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
11	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
12	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND
13	STAND	STAND	STAND	STAND	STAND	STAND	HIT	HIT	STAND	STAND
14	STAND	STAND	STAND	HIT	HIT	HIT	HIT	HIT	HIT	STAND
15	STAND	STAND	HIT	HIT	HIT	HIT	HIT	HIT	STAND	STAND
16	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	STAND
17	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	STAND
18	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	HIT	STAND
19	HIT	HIT	HIT	HIT	STAND	STAND	STAND	STAND	STAND	STAND
20	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND	STAND

Figure 9 – Deep Q-Learning Q *table.*

RESULTS AND DISCUSSION

We compared the results of our agents with those of simple strategies:

	Algorithms	Results *Wins/Ties/Loss / Expected value*	
		"*With replacement*" scenario	
Simple algorithms	Random	28% / 4% / 68% / -0.40	
	Stochastic-without-memory	41% / 9.5% / 49.5% / -0.085	
	Range actions (for 14)	41% / 9% / 50% / -0.09	
	Basic Strategy	43.1% / 9.4% / 47.5% / -0.044	
RL methods	Q-Learning	38% / 8% / 54% / -0.16	
	Double Q-Learning	36% / 9% / 55% / -0.19	
		"*With replacement*" scenario	"*Without replacement*" scenario
	Neural Network	42.5% / 8.4% / 49.1% / -0.066	44% / 8.2% / 47.8% / -0.038
	Deep Q-Learning	44% / 7.50% / 48.50% / -0.045	46% / 6.9% / 47.1% / -0.011

Figure 10 – Summary of the results of our project.

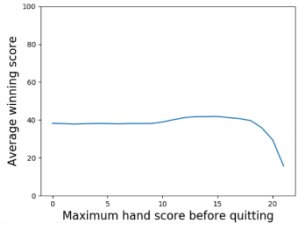


Figure 11 – Average winning score given the maximum threshold in the Range actions strategy. The maximum is reached for a threshold of 14.

One thing we noticed is that whether we replace the cards or not does not affect the results of the strategies computed in the "*with replacement*" scenario.

Q-Learning allows our agent to reach pretty good results, as the agent is able to find an (almost) optimal *Q* table; yet, the decaying learning rate may prevent our agent from re-learning incorrect prediction estimations. Additionally, Q-Learning is limited by the number of states: we cannot count cards with this technique – thus limiting the expected value we can reach. The Neural Network also reaches good results when counting cards, but is limited as its performance depends a lot on the database that we build for it to learn from. We might get better results by changing the way we generate this database (by using the Basic Strategy for example), but it may be considered as cheating… Our best strategy comes from the Deep Q-Learning method, which is logical: it takes the best of both worlds.

As expected, without counting cards, we did not beat the optimal solution - the Basic Strategy – but we managed to get very close to it. When we counted cards, we managed to beat the Basic Strategy; yet, and despite our numerous attempts, we did not succeed in getting a positive expected return – but we know that it is technically possible with advanced methods (the Point count and the Zen count strategies are two examples allowing one to reach such results).

CONCLUSION AND FUTURE WORKS

We have several strategies to improve our agent's results: in the "*without replacement*" scenario, it could be interesting to simulate games with several players – each one being possibly an instance of our agent – thus leading to games with more information; or to work on several decks rather than a single one to test the consistency of our cards counting implementation.

According to the literature, some other approaches could lead to better results: the most promising ones being the Bayesian Q-Learning – Q-Learning with probabilities – and Double Deep Q-Learning – an Actor-Critic technique which improves Deep Q-Learning by applying the principles of Double Q-Learning. As Deep Q-Learning was the technique which gave the best results, this option could lead to even better ones.

To be more thorough, we could continue this project by implementing all the rules of Blackjack (the 60 to 75 final cards are not used in casinos and could be implemented in the evaluation, and not in the training, the possibility for the player to *double*/*draw*/*split*/*surrender*/*insure*, the number of players etc.), as this might be the only way to reach a positive expected value. We did not implement them yet to keep our very first models rather simple.

REFERENCES

[1] *How to Play: Blackjack*, visited on https://bicyclecards.com/how-to-play/blackjack/

[2] P. Bijja, *Teaching a computer blackjack using Reinforcement Learning*, December 2018, visited on https://curiouscoder.space/blog/machine%20learning/teaching-a-computer-blackjack-using-reinforcement-learning/

[3] J. Zhang, *Reinforcement Learning – Solving Blackjack*, June 2019, visited on https://towardsdatascience.com/reinforcement-learning-solving-blackjack-5e31a7fb371f

[4] C. de Granville, *Applying Reinforcement Learning to Blackjack Using Q-Learning*, June 2019, visited on https://www.cs.ou.edu/~granville/paper.pdf

[5] T. Yiu, *Teaching A Neural Net To Play Blackjack*, October 2019, visited on https://towardsdatascience.com/teaching-a-neural-net-to-play-blackjack-8ec5f39809e2

[6] A. Wu, *Playing Blackjack with Deep Q-Learning*, http://cs230.stanford.edu/files_winter_2018/projects/6940282.pdf

[7] H. Hellemons, *Can you still beat the dealer?*, July 2011, https://beta.vu.nl/nl/Images/werkstuk-hellemons_tcm235-225781.pdf

[8] *Hi-Lo strategy*, https://en.wikipedia.org/wiki/Blackjack#Card_counting