SMU - Reinforcement Learning

by José Ananías Hilario Reyes

1. Propose three possible nontrivial reasonable ways how to define the state in the game.

a) $S =< X, Y >$, where: $\begin{cases} \text{X is the set of cards the player is holding} \\ \text{Y is the set of cards the dealer is holding} \end{cases}$

b) $S =< v, w >$, where: $\begin{cases} \text{v is the value of the hand the player is holding} \\ \text{w is the value of the hand the dealer is holding} \end{cases}$

c) $S =< v\_has\_11, v, w\_has\_11, w >$, where: $\begin{cases} \text{v\_has\_11 player has Ace being counted as 11} \\ \text{v is the value of the hand the player is holding} \\ \text{w\_has\_11 dealer has Ace being counted as 11} \\ \text{w is the value of the hand the dealer is holding} \end{cases}$

2. For each state space representation from 1, estimate the overall number of states.

a)

Upper bound: $\begin{cases} \text{X} = \{\text{A, A, 2, 2, 6, 3, 3, 3}\} = 21 \\ \text{Y} = \{\text{A, A, 2, 2, 6, 3, 4}\} = 19 \end{cases}$

$|UB| \leq 15$

$|S_{UB}| \leq \binom{52}{15} = 4.48\text{x}10^{12}$

Upper bound can be too loose. We'll find the lower bound as well, and estimate an average.

Lower bound: $\begin{cases} \text{X} = \{\text{Q, A}\} = 21 \\ \text{Y} = \{\text{J, K}\} = 20 \end{cases}$

$$|LB| \geq 4$$

$$|S_{LB}| \geq \binom{52}{4} = 270725$$

$$E\left[|X| + |Y|\right] \approx \frac{|UB| + |LB|}{2} = 8$$

$$|S_{E[|X|+|Y|]}| = \binom{52}{8} = 7.525\text{x}10^8$$

The resulting state space size for representation a) is huge. We can benefit from some abstraction that captures nearly the same amount of information but drops redundancy.

b)
It is possible to simplify the state space size by eliminating redundant information. In this representation we will focus on the value of the hands held by the player and the house.

$$|v_{min}| = 2, |v_{max}| = 21 \implies v[2, 3, \cdots, 20, 21]\ .$$

$$|w_{min}| = 2, |w_{max}| = 21 \implies w[2, 3, \cdots, 20, 21]\ .$$

$$|S| = |v| \times |w| = 19 \times 19 = 361$$

c)
We can extend the state to hold the information of each player having an Ace valued as 11 for the current hand. We can ignore the information of the dealer only showing one card; since the player plays first anyway, his showing card can be seen simply as the current value he is holding.

$$|v_{min}| = 2, |v_{max}| = 21 \implies v \in [2, 3, \cdots, 20, 21]\ .$$

$$|w_{min}| = 2, |w_{max}| = 21 \implies w \in [2, 3, \cdots, 20, 21]\ .$$

$$|v\_has\_11| = 2, |w\_has\_11| = 2$$

$$|S| = |v| \times |w| = 2 \times 19 \times 2 \times 19 = 1444$$

3. Pick one of the state space representations you proposed in 1. Explain why you consider it the best one and answer the following questions. Does this representation capture all information that can be used for agent decision? Or is there any simplification? If yes, will the simplification influence the result (final policy, utility values)? If yes, how much will the result be influenced? Can you use exact methods (value iteration/policy iteration) to solve the game? If yes, how? If not, why?

Representation c) holds all the information relevant for the game in the form that we will be playing it; ie, deck will be shuffled after each termination of a game. If this wasn't the case, the state could be improved by a card counting scheme (such as HALVES or HiLo), which attempts to capture the history of cards seen in a concise fashion.

Model c) can be learnt much faster than a), since it drops much redundant information, since many cards can be considered equal and a lot of their characteristics could be abstracted for our particular game. By encoding in the state the information if an Ace is held, we obtain versatility in the policy, since this can give us an improvement in our strategy: we can take a risk when holding an Ace, since its value can turn from 11 to 1.

Aside from that, the only information missing to be encoded is the fact that at the start of the game the dealer actually has two cards, but is showing only one. This is irrelevant for us, since the game can be seen as turn-based due to the fact that if we bust, we lose, so we play first, and we lose if we bust. This changes when the house decides to stop drawing cards, but the player can still be drawing.

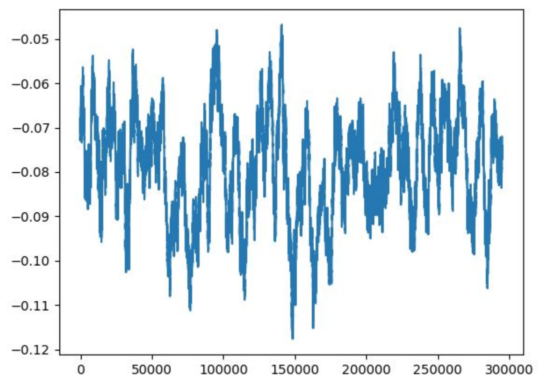
The state space can be further improved if the information on the house decision is also encoded. In a case where the house is not drawing any more cards, it becomes important to know if it is worth anything to try to get as close to 21 as possible, or if its only necessary to be above the house. The influence of this addition should not be to big, since, fundamentally, the value held by the house is taken into consideration, and it doesn't change when it decides to stop drawing cards, and it has to get above 17 anyways, which is very close to 21.

Exact methods, such as value iteration, could be used. In order to do so we need to model the Markov Decision Process (MDP), which implies devising the transition probabilities of the environment, and the reward function.

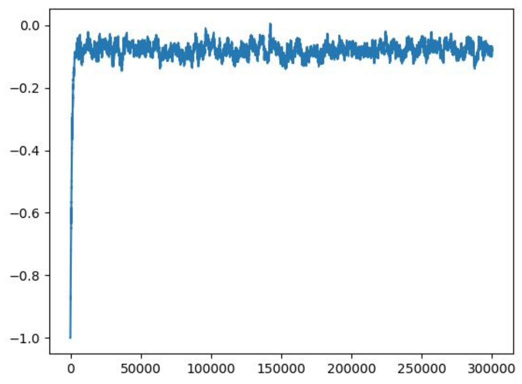
More specifically, in Reinforcement Learning we assume that the world is a Markov Decision Process (MDP) *as the ones that can be solved by value iteration* but neither its transition model, nor its reward function, is known. The goal of Reinforcement Learning is to learn an optimal policy under these circumstances.

6. Compare how successful various strategies are. What is their expected or average utility? How fast do algorithms implemented learn? Does the learned utility contradict your intuition? What is the utility for drawing a card when you have club nine, diamond jack and spades two in your hand and dealer has club four? What is utility of situation when you have diamond ace and spades five and dealer spades ace? Is it better to draw a card in this situation or not? Did your utility values/q values converge?

TD-learning Agent:



Moving average



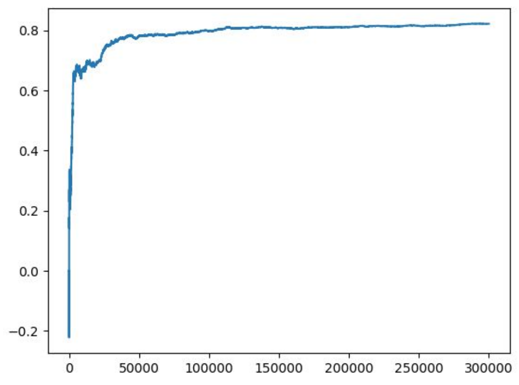
Exponential moving average

The state space here is defined by the value of the player's hand, since the policy is predicated only by this argument.

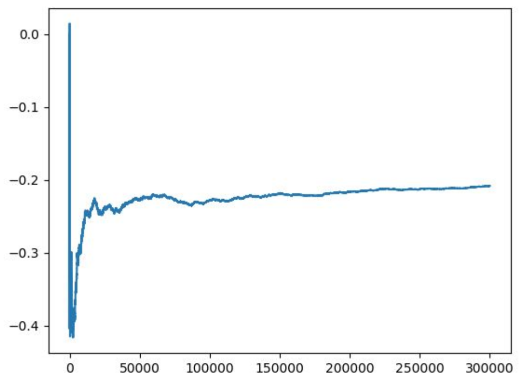
$\alpha = 0.8, \gamma = 0.8$

$U^{\pi}(4) =$	0.044329469620593316
$U^{\pi}(5) =$	0.037949050138593864
$U^{\pi}(6) =$	-0.07288223630906589
$U^{\pi}(7) =$	-0.3026006083565276
$U^{\pi}(8) =$	-0.03824146137057353
$U^{\pi}(9) =$	0.3868928163884582
$U^{\pi}(10) =$	0.7933773331947868
$U^{\pi}(11) =$	0.993261383124476
$U^{\pi}(12) =$	0.004898496604013553
$U^{\pi}(13) =$	-0.04390451754082274
$U^{\pi}(14) =$	-0.09800680847001489
$U^{\pi}(15) =$	-0.17542003430554234
$U^{\pi}(16) =$	-0.22600318226230492
$U^{\pi}(17) =$	-1.39049549920338
$U^{\pi}(18) =$	-0.3955844942851487
$U^{\pi}(19) =$	1.3027886239953463
$U^{\pi}(20) =$	2.582846165957385
$U^{\pi}(21) =$	3.6832699457680516

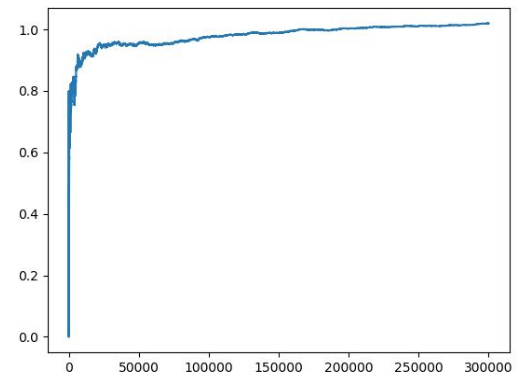
In the following plots we can see that the utility values converge. Only some of them are being shown, but the behavior is the same in all of them.



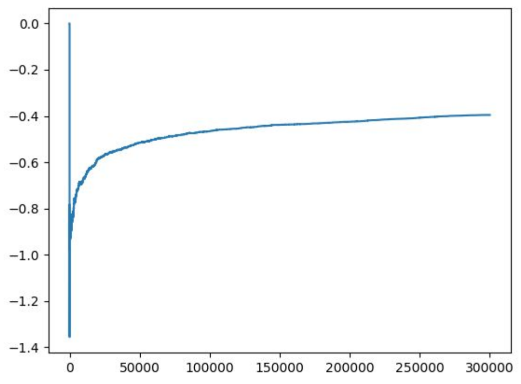
U[10] .



U[16] .



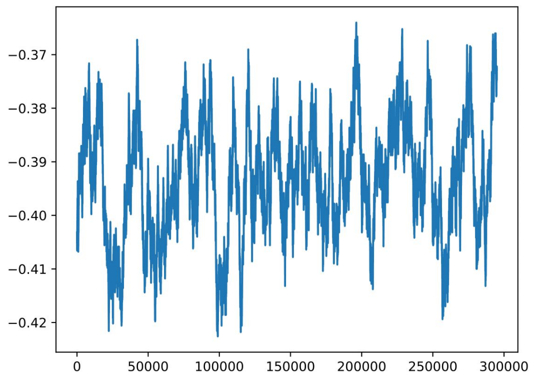
U[11] .



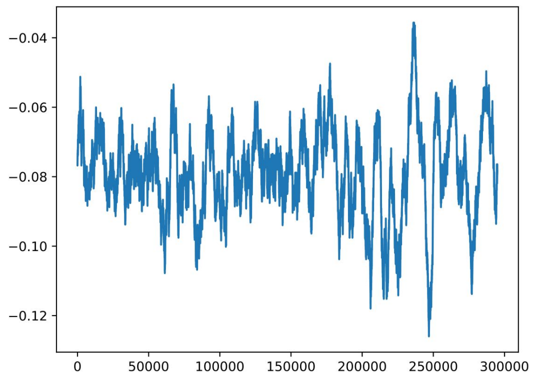
U[18] .

Random Agent:

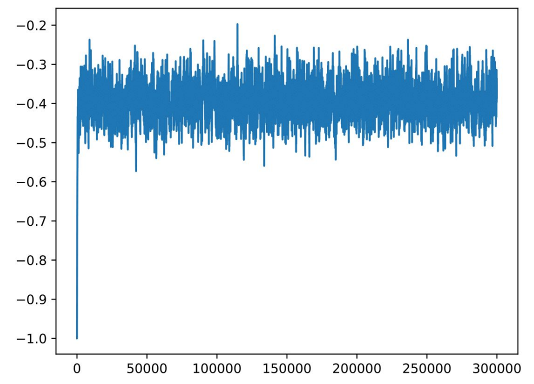
Dealer Agent:



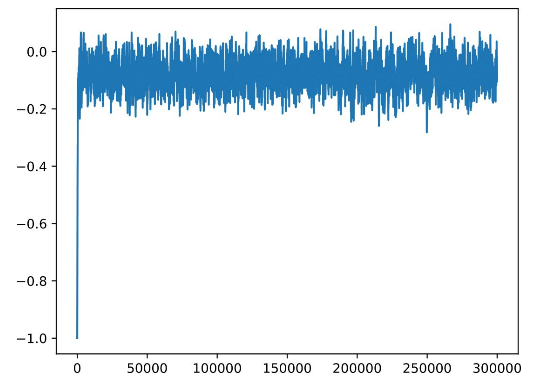
Moving average



Moving average

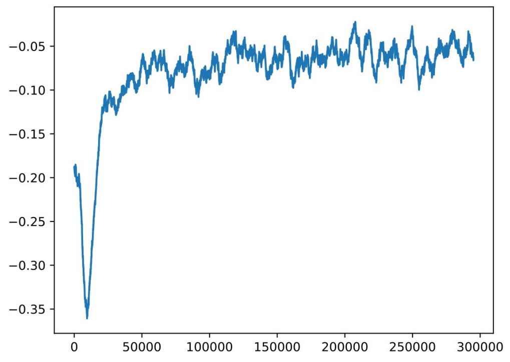


Exponential moving average

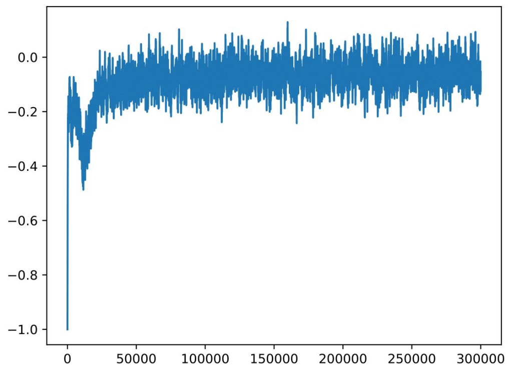


Exponential moving average

SARSA-learning Agent:



Moving average



Exponential moving average

We can see that the policy learned by the SARSA agent is an improvement over the implementation of the dealer's strategy as the player's own.

For the parameters chosen, the learning phase appears to converge around 150000 epochs.

The average rewards obtained in each case are the following, for when convergence is attained:

Random Agent	$E[r] = -0.38835$
Dealer Agent	$E[r] = -0.0741$
TD Agent	$E[r] = -0.0779$
SARSA Agent	$E[r] = -0.05385$

As expected, the results for the TD ager and the Dealer agent are the same. As w know, TD learning is passive and is jus useful for determining the utility value for a fixed policy.

$X = \{9, J, 2\}, Y = \{4\} \implies v = 21, w = 4 \implies s =< \text{False}, 21, \text{False}, 4 >$
$Q(s =< \text{False}, 21, \text{False}, 4 >, a = \text{HIT}) = -1$
SARSA strategy: STAND   Officially suggested strategy: STAND

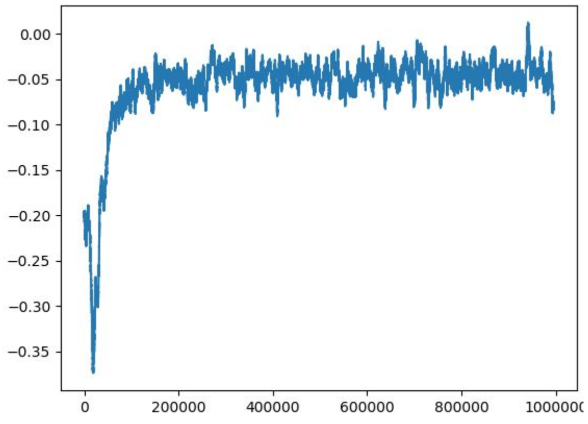
$X = \{A, 5\}, Y = \{A\} \implies v = 16, w = 11 \implies s =< \text{True}, 16, \text{True}, 11 >$
$Q(s =< \text{True}, 16, \text{True}, 11 >, a = \text{STAND}) = -0.8675964768844708$
$Q(s =< \text{True}, 16, \text{True}, 11 >, a = \text{HIT}) = -0.2978233715907907$
SARSA strategy: HIT   Officially suggested strategy: HIT

Some improvements were added to the SARSA agent: epsilon-greedy action selection was added to the optimistic utilities scheme.

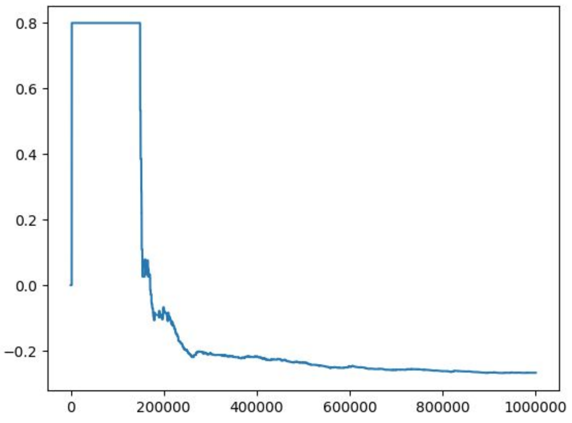
Agent complies with GLIE scheme: the values 100, 0.8, 0.8 were assigned to the parameters N_e, α, γ, respectively

More epochs were needed to reach convergence. The learning consisted of 1000000 epochs, with convergence appearing at around the epoch 800000.

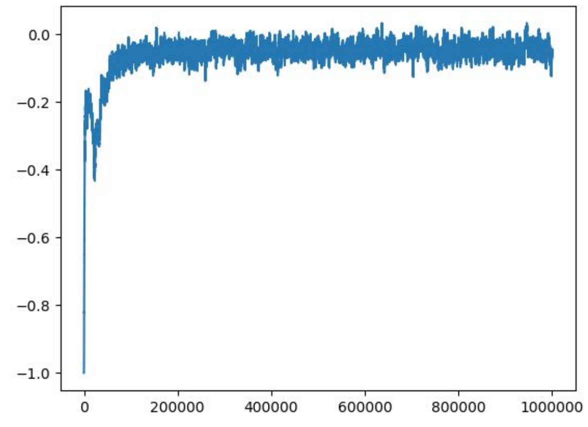
The average reward obtained was of , E[r] = -0.045, consistently in the average.



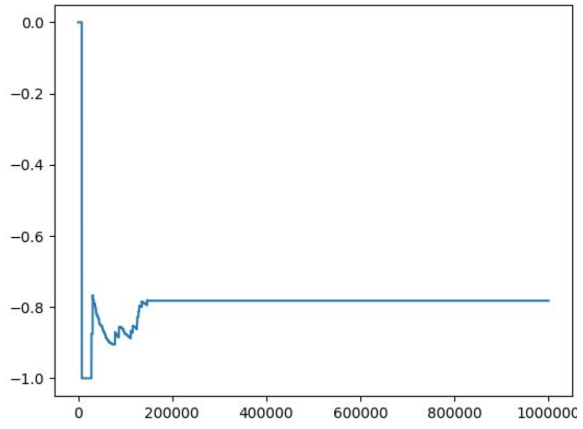
Moving average



$Q(<\text{True}, 16, \text{True}, 11>, \text{HIT})$



Exponential moving average



$Q(<\text{True}, 16, \text{True}, 11>, \text{STAND})$