Machine Learning Project
Artificial Intelligence & Machine Learning 2023/24

The Blackjack environment

SAPIENZA
UNIVERSITÀ DI ROMA

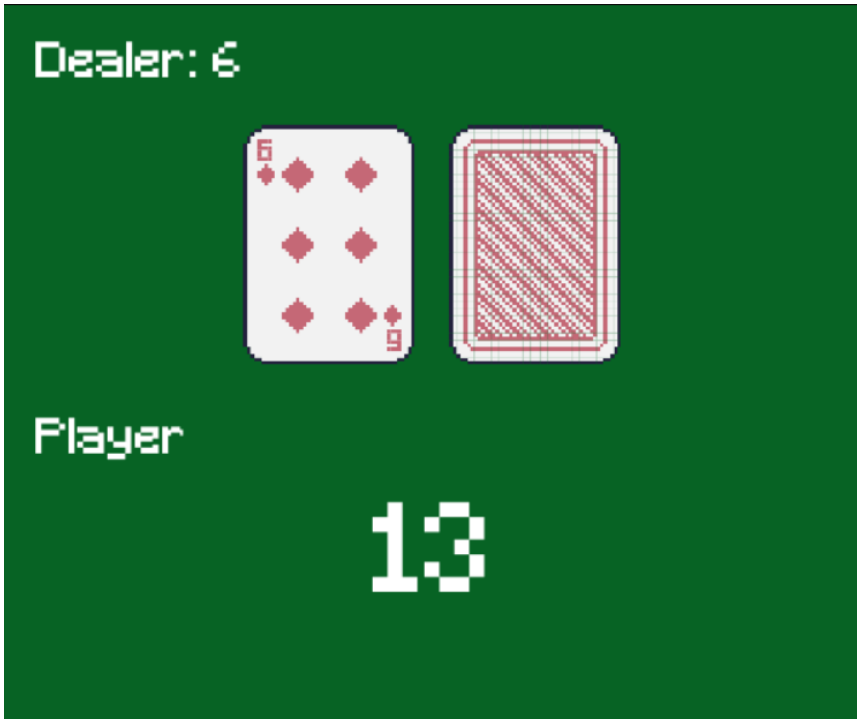
Project presented by:

Saverio Dieni 1946039

Introduction

My machine learning project is about an agent who learns how to play Blackjack using reinforcement learning techniques. The objective of the agent is to learn a policy that, in every situation, determines the optimal action to beat the dealer by obtaining a hand closer to 21 than the dealer's hand, without exceeding 21. The game starts with the dealer having one face up and one face down card, while the player has two face up cards. All cards are drawn from deck with replacement. The card values are:

- Face cards (Jack, Queen, King) have a point value of 10.
- Aces can either count as 11, called a 'usable ace', or 1.
- Numerical cards, from 2 to 9, have a value equal to their number.



The player can request additional cards (hit) until she/he decides to stop (stick) or get a sum greater than 21. After the player sticks, the dealer reveals their facedown card, and draws cards until their sum is 17 or greater. If the dealer exceeds 21, the player wins. If neither the player nor the dealer busts, the winner of the game is decided by whose sum is closer to 21.

The environment

The Blackjack environment is part of the Toy Text environments of Gymnasium which contains general information about the environment. We are interested in the following information:

Observation Space

The observation consists of a 3-tuple containing:

- The player's current sum, the initial value is at least 4 and at most 11, in general this value is at most 31.
- The value of the dealer's one showing card, the value is in the range 1-10, where 1 is ace.
- A Boolean value telling whether the player holds a usable ace.

The observation is returned as (int(), int(), int()).

Action Space

There are only 2 possible actions indicating whether to stick or hit. The action stick is associated with value 0, and the action hit is associated with the value 1.

Rewards

The agent gets a reward when an episode ends. Reward values:

- +1: the episode ends and the player is the winner.
- -1: the episode ends and the dealer is the winner.
- 0: the episode ends and both the dealer and the player have the same sum.

The episode ends whether the player sticks or the player hits and the sum exceeds 21.

Solutions

In this section I will describe the techniques I used to make an agent able to solve this environment.
I implemented 2 variations of the Q-learning algorithm, one of them using the Q-Table, and the other one using a single DQN (Deep Q-Network).

Q-learning algorithm

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment and its goal is to learn the optimal policy by exploring the environment. Given the observation space S and the action space A, it can learn the Q function: $S \times A \rightarrow \mathbb{R}$, where each Q(s,a) is the value of the expected utility of executing action a in state s.

The algorithm initializes each Q(s,a) to 0 and given an initial state it choose an action based on the epsilon-greedy policy, gets the reward for executing a in s, updates the value Q(s,a) and in the end produce the new state s' obtained by executing a in s. It repeats this step for many iterations. The epsilon-greedy policy is used to balance the number of time that an action is chosen randomly, to explore the environment, and the number of the time that is chosen the action that gives more reward. Usually at the beginning of the algorithm any action is chosen randomly, and at every iteration the probability of exploitation increases, while the probability of exploration decreases.

Since the described environment is not deterministic (indeed, the outcome of each game is not deterministically defined only by the state and the action, but it also depends on the value of the next cards), at every step the value Q(s,a) is updated following the rule where:

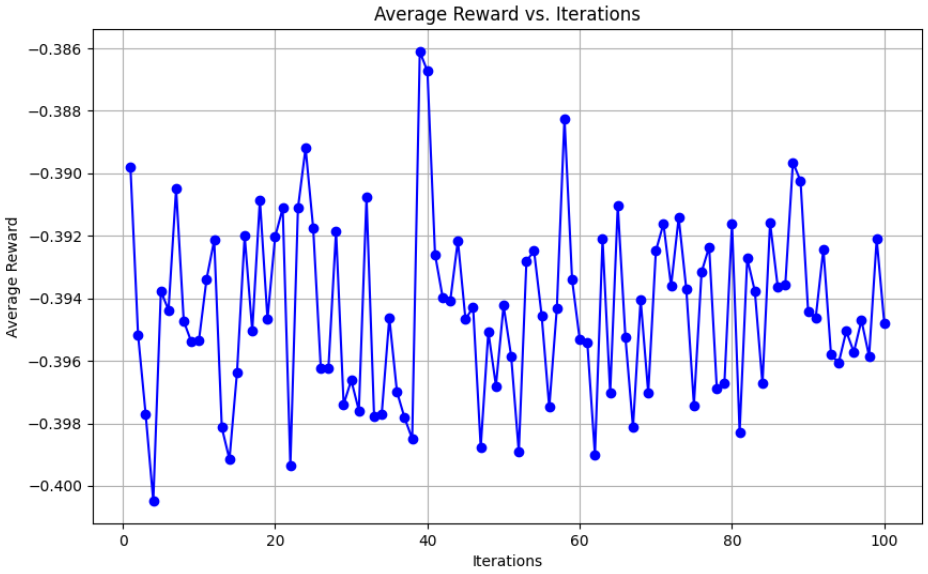
$$Q(s,a) \coloneqq Q(s,a) + \alpha \left[r(s,a) + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right]$$

- Q(s,a) in the right hand-side indicates the current Q-value executing a certain action a in state s.
- Q(s,a) in the left hand-side indicates the current Q-value executing a certain action a in state s.
- α indicates the learning rate.
- ɣ indicates the discount factor.
- r(s,a) is the reward obtained by executing action a in state s.
- s' is the new state obtained by executing action a in state s.

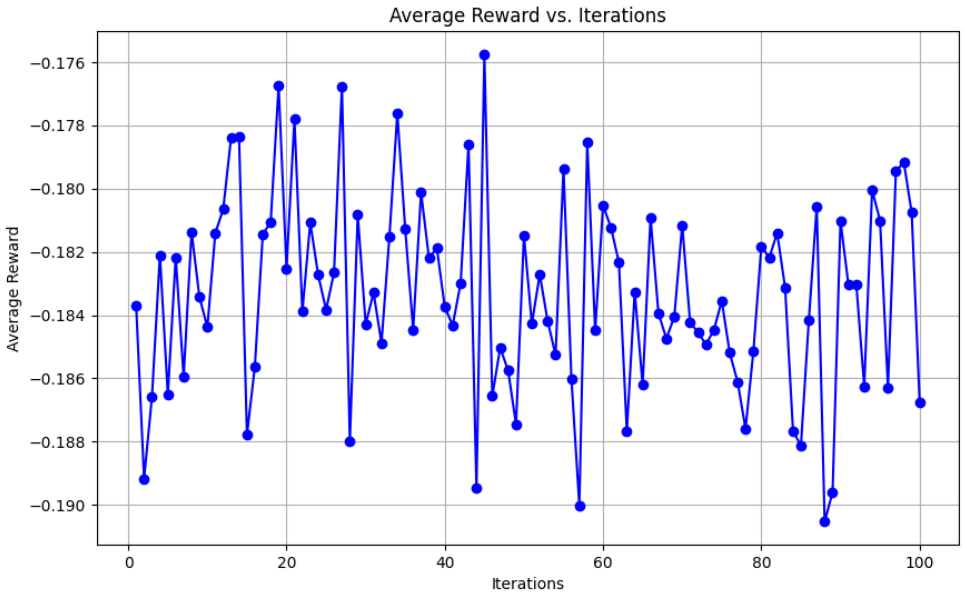
Solution using the Q-table

The solution using the Q-table is naturally derived from the algorithm we have seen so far. We simply store all the tuples inside a table, and we update it. This solution is very effective for small and discrete environments, otherwise the table could be too large to be stored and to be used. Fortunately, this environment is discrete, and this solution is easily applicable.

First, I measured the results obtained with an agent following a very simple policy, in particular I wanted to understand the performances of agent that takes only random actions, and the performances of an agent that always sticks. In both cases, for 100 iterations I measured the average reward obtained over 100000 episodes, I collected those data, and I produced the following plots.



Reward with random actions



Reward choosing always action 0 (stick)

In my implementation of the agent using the Q-table I used this set of parameters:

- Learning rate, this parameter controls how quickly the model updates its values based on new experiences.
- Discount factor, this parameter is used to determine the weight given to future rewards.
- Exploration decay, this parameter is used to decrease the exploration probability at every when an episode ends.
- Train episodes, this parameter tells us how many episodes are used to update the values in the Q-table. Increasing the number of training episodes may lead the model to overfit to the training phase.
- Test episodes, this parameter tells us how many episodes are used to test the agent.

```
# Training phase
cumulative_reward = 0
for episode in range(train_episodes):
  observation, info = env.reset()
  done = False
  episode_reward = 0
  current_state = get_state(observation)
  while not done:
    # Epsilon-greedy action selection
    if np.random.rand() < exploration_prob:
      action = env.action_space.sample()
    else:
      action = np.argmax(Q_table[current_state])

    # Take action and observe the result
    observation, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated
    next_state = get_state(observation)

    # Q-learning update
    best_next_action = np.argmax(Q_table[next_state])
    Q_table[current_state][action] += learning_rate * (
        reward + discount_factor * Q_table[next_state][best_next_action] - Q_table[current_state][action])

    episode_reward += reward
    current_state = next_state

  cumulative_reward += episode_reward

  # Decrease exploration probability
  exploration_prob = max(min_exploration_prob, exploration_prob * exploration_decay)
  if (episode + 1) % 1000 == 0:
    avg_reward = cumulative_reward / (episode + 1)
    print(f"Train episodes: {episode + 1}, Average Reward: {avg_reward}")
```

This is essentially the main core of the agent, for every training episode it starts from a random state and following the epsilon-greedy policy it chooses action, updates the Q-table, until the end of

the episode. After that it update the exploration probability and after 1000 episodes it prints the current average cumulative reward.

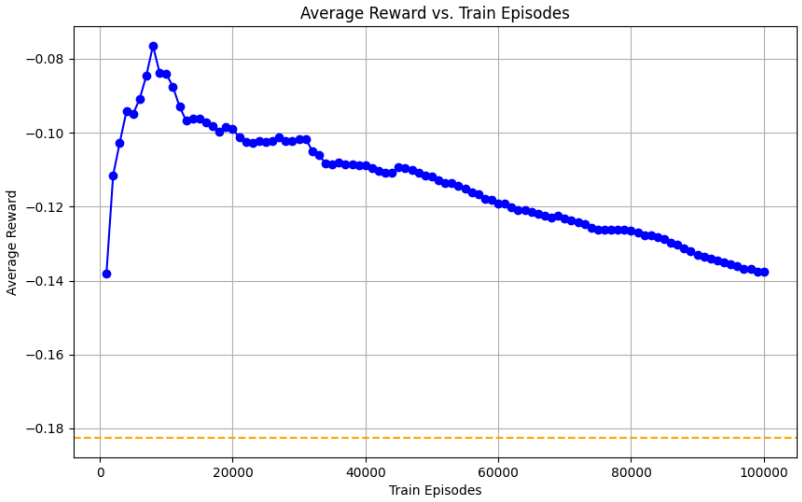
Configurations

I tried 4 different configurations, with aim to obtain the best possible reward:

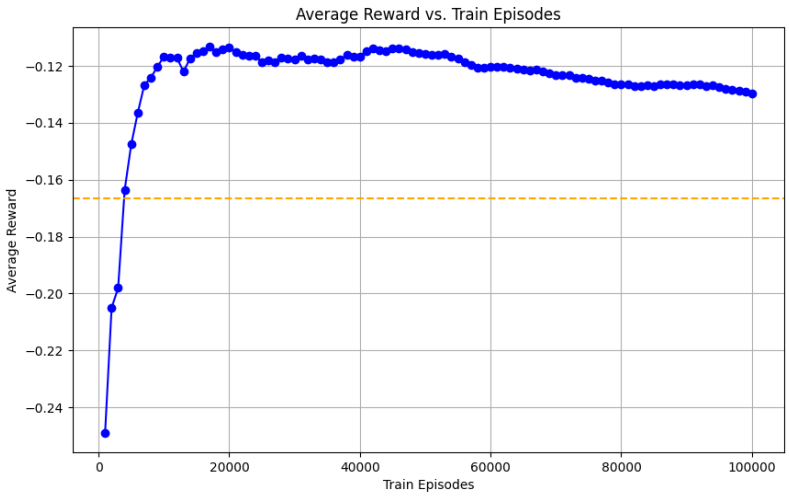
- Configuration 1: train_episodes=100000, test_episodes=100000, discount_factor=0.9, learning_rate=0.01, exploration_decay=0.99
- Configuration 2: train_episodes=100000, test_episodes=100000, discount_factor=0.9, learning_rate=0.01, exploration_decay=0.999
- Configuration 3: train_episodes=100000, test_episodes=100000, discount_factor=0.9, learning_rate=0.001, exploration_decay=0.999
- Configuration 4: train_episodes=100000, test_episodes=100000, discount_factor=0.8, learning_rate=0.001, exploration_decay=0.99

I collected data produced from the previous code, and I used them to produce a plot to evaluate the performance of the agent. In the next plots the orange dashed line indicates the average reward obtained by the agent during the test episodes.

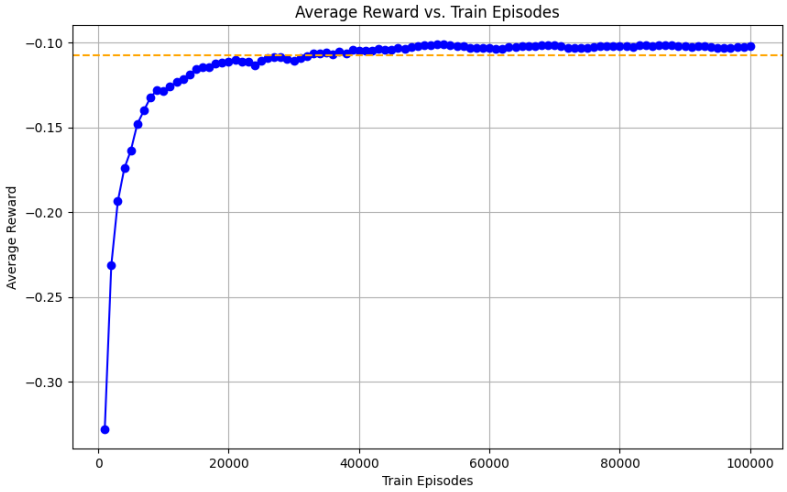
- Configuration 1:



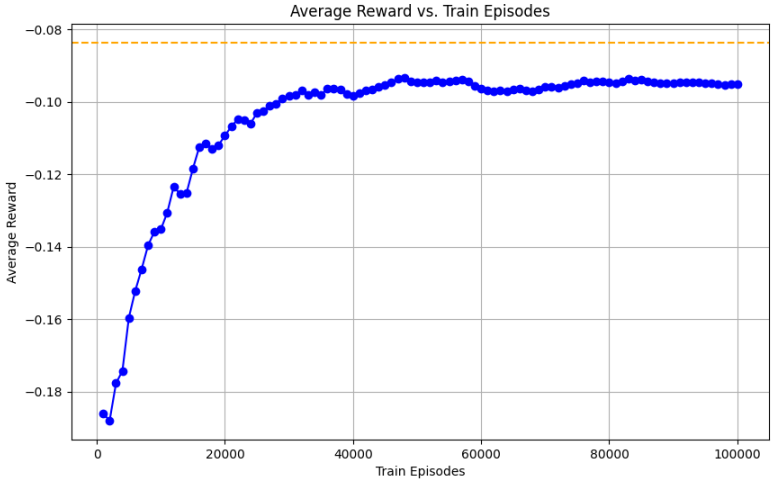
- Configuration 2:



- Configuration 3:



- Configuration 4 :



The first configuration of the agent gets an average reward similar to that of an agent that always takes the same action (action 0), so it is not learning effectively. However, the cumulative reward, which is represented by the blue line, indicates that the agent might be overfitting to the training data. The second configuration gets a slightly better average reward, but it suffers of the problems of the first configuration.

The configuration 3 and 4 are the best, they get a significantly higher average reward. From the plot of the third configuration we can notice that the blue line and the orange line (representing the average reward during the testing phase) are close to each other, even for a high number of training episodes, and this implies that the agent is not overfitting to the training data. From the plot of the fourth configuration we notice that the average reward obtained during the testing phase is higher than the average reward.

LEARNED Q-TABLE (CONF.3)	LEARNED Q-TABLE (CONF.4)
(4, 1, 0): [-0.0020002 -0.00123485]	(4, 1, 0): [-0.0029997 -0.00257032]
(4, 10, 0): [-0.01496864 -0.01433363]	(4, 10, 0): [-0.01498273 -0.01489083]
(4, 2, 0): [-0.001 -0.00045699]	(4, 2, 0): [-0.0019998 -0.00126065]
(4, 3, 0): [-0.0019998 -0.00106612]	(4, 3, 0): [-0.0010013 -0.00099217]
(4, 4, 0): [-0.0019996 -0.00109985]	(4, 4, 0): [-0.001999 -0.00152101]
(4, 5, 0): [0.00693223 0.]	(4, 5, 0): [-0.0010001 -0.00096979]
(4, 6, 0): [-0.00101156 -0.00017986]	(4, 6, 0): [-0.001 -0.00057508]
(4, 7, 0): [-0.00099944 -0.00087147]	(4, 7, 0): [-0.001 -0.00071871]
(4, 8, 0): [-0.00199852 -0.00095678]	(4, 8, 0): [-0.00299907 -0.00205582]
(4, 9, 0): [-0.00099926 -0.00099904]	(4, 9, 0): [-0.0019998 -0.00102828]
(5, 1, 0): [-0.004998 -0.00400805]	(5, 1, 0): [-0.00599787 -0.00510355]
(5, 10, 0): [-0.03386703 -0.033058]	(5, 10, 0): [-0.03887495 -0.03854766]
(5, 2, 0): [-0.0029994 -0.00288153]	(5, 2, 0): [-0.0039993 -0.00364204]
(5, 3, 0): [-0.0019998 -0.00183976]	(5, 3, 0): [-0.0029997 -0.0022693]
(5, 4, 0): [0.00398466 -0.00104998]	(5, 4, 0): [-0.00199933 -0.0020239]
(5, 5, 0): [0.00399549 -0.00102437]	(5, 5, 0): [-0.00399966 -0.00211407]
(5, 6, 0): [-7.38609385E-06 -1.01161037E-03]	(5, 6, 0): [-0.00299736 -0.00100883]
(5, 7, 0): [-0.0029994 -0.0024535]	(5, 7, 0): [-0.0039972 -0.00373166]
(5, 8, 0): [-0.0039988 -0.00348147]	(5, 8, 0): [-0.004003 -0.00340002]
(5, 9, 0): [-0.0039988 -0.00376021]	(5, 9, 0): [-0.0019978 -0.00404564]
(6, 1, 0): [-0.00699286 -0.00683484]	(6, 1, 0): [-0.00899411 -0.00877206]
(6, 10, 0): [-0.05651381 -0.05606458]	(6, 10, 0): [-0.058684 -0.05785593]
(6, 2, 0): [-0.0069939 -0.00442007]	(6, 2, 0): [-0.00499724 -0.00475632]
(6, 3, 0): [-0.0039984 -0.00357984]	(6, 3, 0): [-0.00399608 -0.00408372]
(6, 4, 0): [-0.00099581 -0.00202651]	(6, 4, 0): [-0.0040032 -0.00381237]
(6, 5, 0): [-0.002999 -0.00240565]	(6, 5, 0): [-0.00301885 -0.00283831]
(6, 6, 0): [-0.00400876 -0.00275774]	(6, 6, 0): [-0.0020004 -0.00178083]
(6, 7, 0): [-0.005997 -0.00543902]	(6, 7, 0): [-0.00899561 -0.00807277]
(6, 8, 0): [-0.00599624 -0.00507628]	(6, 8, 0): [-0.0069945 -0.00653259]
(6, 9, 0): [-0.00598902 -0.00509233]	(6, 9, 0): [-0.0069971 -0.00660443]
(7, 1, 0): [-0.01297986 -0.01243598]	(7, 1, 0): [-0.01098923 -0.01100204]
(7, 10, 0): [-0.07696473 -0.07652622]	(7, 10, 0): [-0.08824751 -0.0871372]
(7, 2, 0): [-0.00599569 -0.00500883]	(7, 2, 0): [-0.00499779 -0.00466532]
(7, 3, 0): [-0.00499732 -0.00455268]	(7, 3, 0): [-0.004999 -0.00442237]
(7, 4, 0): [-0.00399851 -0.00376738]	(7, 4, 0): [-0.00599367 -0.00513419]
(7, 5, 0): [-0.00199941 -0.0016466]	(7, 5, 0): [-0.00499687 -0.00289303]
(7, 6, 0): [-0.00306221 -0.00290245]	(7, 6, 0): [-0.00099978 -0.00069217]
(7, 7, 0): [-0.00599471 -0.00590221]	(7, 7, 0): [-0.00699666 -0.00671002]
(7, 8, 0): [-0.0089863 -0.00862773]	(7, 8, 0): [-0.0119915 -0.01145658]
(7, 9, 0): [-0.00898422 -0.00805239]	(7, 9, 0): [-0.01099143 -0.01045668]
(8, 1, 0): [-0.01496558 -0.01465423]	(8, 1, 0): [-0.01697557 -0.01658393]
(8, 10, 0): [-0.10794264 -0.1073884]	(8, 10, 0): [-0.11200944 -0.11208392]
(8, 2, 0): [-0.0029994 -0.00229703]	(8, 2, 0): [-0.0029986 -0.00218452]
(8, 3, 0): [-0.00300021 -0.00267629]	(8, 3, 0): [-0.0010001 0.00038135]
(8, 4, 0): [-0.00101589 -0.00036903]	(8, 4, 0): [-0.00199932 -0.00118134]
(8, 5, 0): [-0.00199915 -0.00015283]	(8, 5, 0): [-0.00199929 0.00170932]
(8, 6, 0): [-0.00101137 0.00050274]	(8, 6, 0): [-0.00199746 0.00208667]
(8, 7, 0): [-0.00199898 0.00169806]	(8, 7, 0): [-0.0010004 0.00226974]
(8, 8, 0): [-0.00599664 -0.00570085]	(8, 8, 0): [-0.004996 -0.00419429]

(8, 9, 0): [-0.00998987 -0.00919902]	(8, 9, 0): [-0.0119851 -0.01194181]
(9, 1, 0): [-0.01693807 -0.01677809]	(9, 1, 0): [-0.02097717 -0.02096559]
(9, 10, 0): [-0.08471745 -0.08433006]	(9, 10, 0): [-0.09224899 -0.09342766]
(9, 2, 0): [0.0010029 0.00467656]	(9, 2, 0): [-0.00200129 0.00767172]
(9, 3, 0): [-0.00199899 0.00158547]	(9, 3, 0): [-0.00198645 0.00540174]
(9, 4, 0): [-0.001 0.00801245]	(9, 4, 0): [-0.00200236 0.00885177]
(9, 5, 0): [-0.00199639 0.00622645]	(9, 5, 0): [-0.00199733 0.00594506]
(9, 6, 0): [-0.00099374 0.00726819]	(9, 6, 0): [-0.00099628 0.00665035]
(9, 7, 0): [-0.00199895 0.01254353]	(9, 7, 0): [0. 0.01275795]
(9, 8, 0): [-0.001 0.00607712]	(9, 8, 0): [3.44091750e-06 9.50612334e-03]
(9, 9, 0): [-0.0059956 -0.00488542]	(9, 9, 0): [-0.00798972 -0.0055717]
(10, 1, 0): [-0.01198497 -0.00941925]	(10, 1, 0): [-0.00998804 -0.00977631]
(10, 10, 0): [-0.0065227 0.11297551]	(10, 10, 0): [-0.00949728 0.16009946]
(10, 2, 0): [3.48702022E-05 2.68626378E-02]	(10, 2, 0): [7.24774719e-07 2.98086178e-02]
(10, 3, 0): [-0.00198818 0.02573607]	(10, 3, 0): [-0.00196609 0.02875911]
(10, 4, 0): [-2.00000000E-07 2.37013533E-02]	(10, 4, 0): [-0.00297127 0.0248788]
(10, 5, 0): [-0.00297473 0.02670072]	(10, 5, 0): [-0.00198523 0.0370627]
(10, 6, 0): [5.22243583E-06 2.02152678E-02]	(10, 6, 0): [-0.00098962 0.03289276]
(10, 7, 0): [-0.00098756 0.03085678]	(10, 7, 0): [-0.00196612 0.03138737]
(10, 8, 0): [3.42920395E-05 2.62367262E-02]	(10, 8, 0): [-0.00499001 0.03607223]
(10, 9, 0): [-0.00299483 0.02138786]	(10, 9, 0): [-0.0010001 0.02359509]
(11, 1, 0): [-0.00998391 -0.00962269]	(11, 1, 0): [-0.00499093 -0.00290962]
(11, 10, 0): [-0.01052071 0.12158728]	(11, 10, 0): [-0.01638564 0.12271279]
(11, 2, 0): [-0.00198589 0.019313]	(11, 2, 0): [-0.00199488 0.01661958]
(11, 3, 0): [-0.00398279 0.0143953]	(11, 3, 0): [-0.00298815 0.0174213]
(11, 4, 0): [-0.00199259 0.01611867]	(11, 4, 0): [0.00101863 0.01493018]
(11, 5, 0): [-1.13838531E-05 1.84056886E-02]	(11, 5, 0): [-0.00296918 0.02494644]
(11, 6, 0): [-0.00099636 0.01465281]	(11, 6, 0): [1.09978056e-05 2.13508384e-02]
(11, 7, 0): [0.00200676 0.01716508]	(11, 7, 0): [0.00104081 0.02656798]
(11, 8, 0): [-0.00198727 0.01817242]	(11, 8, 0): [-0.00196353 0.01912943]
(11, 9, 0): [-0.0019924 0.01028041]	(11, 9, 0): [-0.00296333 0.01681364]
(12, 1, 0): [-0.16997218 -0.16931972]	(12, 1, 0): [-0.1748995 -0.17494546]
(12, 1, 1): [-0.002 -0.00106168]	(12, 1, 1): [-0.001 -0.00037913]
(12, 10, 0): [-0.29503608 -0.29219264]	(12, 10, 0): [-0.29876768 -0.28912215]
(12, 10, 1): [-0.0039986 -0.00236875]	(12, 10, 1): [-0.00499677 0.00236271]
(12, 2, 0): [-0.11036097 -0.11399114]	(12, 2, 0): [-0.11984196 -0.11797632]
(12, 2, 1): [-0.001 -0.00040987]	(12, 2, 1): [-0.00199909 -0.00109328]
(12, 3, 0): [-0.0801143 -0.08040415]	(12, 3, 0): [-0.10478395 -0.10396908]
(12, 3, 1): [1.03076585E-06 6.76153738E-04]	(12, 3, 1): [-0.0010007 -0.00025406]
(12, 4, 0): [-0.0640249 -0.06782956]	(12, 4, 0): [-0.09615046 -0.09560846]
(12, 4, 1): [-1.00039968E-03 9.58071131E-05]	(12, 4, 1): [-0.001 0.0003693]
(12, 5, 0): [-0.09671446 -0.0960133]	(12, 5, 0): [-0.0958608 -0.09289128]
(12, 5, 1): [-0.001 0.00022466]	(12, 5, 1): [0.00498143 0.]
(12, 6, 0): [-0.08783772 -0.0883878]	(12, 6, 0): [-0.03366393 -0.03418991]
(12, 6, 1): [-3.97857753E-07 1.33391054E-03]	(12, 6, 1): [-0.001 0.0009929]
(12, 7, 0): [-0.13106498 -0.13333859]	(12, 7, 0): [-0.12164581 -0.1205896]
(12, 7, 1): [-0.001 -0.00044893]	(12, 7, 1): [-1.00009999e-03 -9.63976962e-05]
(12, 8, 0): [-0.13403382 -0.13512523]	(12, 8, 0): [-0.12943227 -0.1285339]
(12, 8, 1): [-1.99906819E-03 -9.35231037E-05]	(12, 8, 1): [-0.0010001 -0.00047628]
(12, 9, 0): [-0.13854224 -0.13942351]	(12, 9, 0): [-0.12908382 -0.12527479]

(12, 9, 1): [-0.001 -0.00052434]	(12, 9, 1): [-0.001 0.00013401]
(13, 1, 0): [-0.20954602 -0.20960412]	(13, 1, 0): [-0.19555509 -0.19550936]
(13, 1, 1): [-0.0019998 -0.00135617]	(13, 1, 1): [-0.0029997 -0.00213985]
(13, 10, 0): [-0.35804663 -0.34345356]	(13, 10, 0): [-0.38930446 -0.36950405]
(13, 10, 1): [-0.01297328 -0.01237565]	(13, 10, 1): [-0.0149763 -0.01382112]
(13, 2, 0): [-0.12762108 -0.12842148]	(13, 2, 0): [-0.11827079 -0.1207649]
(13, 2, 1): [-0.00299717 -0.00030982]	(13, 2, 1): [-0.0020004 -0.00095647]
(13, 3, 0): [-0.14343421 -0.14305347]	(13, 3, 0): [-0.11624879 -0.11604786]
(13, 3, 1): [-0.0020006 -0.00082204]	(13, 3, 1): [0.0049988 -0.0010633]
(13, 4, 0): [-0.09553239 -0.09544845]	(13, 4, 0): [-0.1129525 -0.11622452]
(13, 4, 1): [-0.00199926 0.00099974]	(13, 4, 1): [-0.0019998 -0.00100692]
(13, 5, 0): [-0.09058186 -0.08969116]	(13, 5, 0): [-0.11285818 -0.10834044]
(13, 5, 1): [-1.00000000E-03 9.92043233E-05]	(13, 5, 1): [-0.0010004 -0.00021198]
(13, 6, 0): [-0.06745498 -0.07335536]	(13, 6, 0): [-0.10322601 -0.10247008]
(13, 6, 1): [-0.001 0.00040202]	(13, 6, 1): [-0.0010001 -0.00016103]
(13, 7, 0): [-0.14701125 -0.14638141]	(13, 7, 0): [-0.14473871 -0.14401496]
(13, 7, 1): [-0.00199853 0.0005377]	(13, 7, 1): [-0.0010004 0.00024368]
(13, 8, 0): [-0.17421676 -0.1737416]	(13, 8, 0): [-0.16189944 -0.16156111]
(13, 8, 1): [-0.001 -0.00065695]	(13, 8, 1): [-2.99729533e-03 -9.06557372e-05]
(13, 9, 0): [-0.16186833 -0.16090411]	(13, 9, 0): [-0.17790224 -0.17708126]
(13, 9, 1): [-0.0019996 -0.00194971]	(13, 9, 1): [-0.0019999 -0.00117757]
(14, 1, 0): [-0.22474305 -0.22382358]	(14, 1, 0): [-0.23556463 -0.23604147]
(14, 1, 1): [-0.00399819 -0.00392752]	(14, 1, 1): [-0.0049983 -0.00399719]
(14, 10, 0): [-0.40960159 -0.40452723]	(14, 10, 0): [-0.4191398 -0.3900893]
(14, 10, 1): [-0.02192295 -0.01723528]	(14, 10, 1): [-0.0149903 -0.01369417]
(14, 2, 0): [-0.14910922 -0.14980603]	(14, 2, 0): [-0.133742 -0.13327336]
(14, 2, 1): [-0.00199951 -0.00136631]	(14, 2, 1): [-0.00199904 -0.00169497]
(14, 3, 0): [-0.13564634 -0.13471008]	(14, 3, 0): [-0.12071766 -0.12030098]
(14, 3, 1): [-0.0019996 -0.00113149]	(14, 3, 1): [-0.00299886 -0.00058259]
(14, 4, 0): [-0.11178865 -0.1106133]	(14, 4, 0): [-0.10979175 -0.11112512]
(14, 4, 1): [-0.00199837 0.00026518]	(14, 4, 1): [-0.0010001 0.00048781]
(14, 5, 0): [-0.1000998 -0.10482089]	(14, 5, 0): [-0.10994595 -0.11158067]
(14, 5, 1): [-1.00658717E-03 2.38082376E-05]	(14, 5, 1): [-0.001 -0.00109253]
(14, 6, 0): [-0.10921263 -0.10941885]	(14, 6, 0): [-0.06685473 -0.07198989]
(14, 6, 1): [-0.0019998 -0.00021252]	(14, 6, 1): [-0.001 0.00017424]
(14, 7, 0): [-0.154264 -0.15264726]	(14, 7, 0): [-0.16021513 -0.15867083]
(14, 7, 1): [-9.99007009E-04 -5.89541958E-05]	(14, 7, 1): [-0.0019999 -0.00104322]
(14, 8, 0): [-0.16778464 -0.16693048]	(14, 8, 0): [-0.19239647 -0.19154626]
(14, 8, 1): [-0.001 0.0005368]	(14, 8, 1): [-0.0029994 -0.00293001]
(14, 9, 0): [-0.16107189 -0.16101189]	(14, 9, 0): [-0.17937836 -0.17960978]
(14, 9, 1): [-0.00199965 -0.00109906]	(14, 9, 1): [-0.0019999 -0.00198471]
(15, 1, 0): [-0.23661759 -0.23682669]	(15, 1, 0): [-0.24713699 -0.24825018]
(15, 1, 1): [-0.004998 -0.00455851]	(15, 1, 1): [-0.00599705 -0.00520838]
(15, 10, 0): [-0.43301336 -0.43081782]	(15, 10, 0): [-0.46930864 -0.46465688]
(15, 10, 1): [-0.03085142 -0.02997226]	(15, 10, 1): [-0.02495586 -0.02424155]
(15, 2, 0): [-0.10769545 -0.1154497]	(15, 2, 0): [-0.14204082 -0.14114203]
(15, 2, 1): [-0.00199901 -0.00097436]	(15, 2, 1): [-0.0019997 -0.00028218]
(15, 3, 0): [-0.13753577 -0.13730424]	(15, 3, 0): [-0.13612737 -0.13544526]
(15, 3, 1): [-0.00199949 -0.00111793]	(15, 3, 1): [-0.0029993 -0.00206942]
(15, 4, 0): [-0.09369416 -0.09503189]	(15, 4, 0): [-0.11143682 -0.11323973]

(15, 4, 1): [-0.0010002 0.0009588]	(15, 4, 1): [-0.00299875 -0.00242492]
(15, 5, 0): [-0.08250443 -0.08231125]	(15, 5, 0): [-0.13740921 -0.13700428]
(15, 5, 1): [-0.00199937 -0.00045858]	(15, 5, 1): [-0.00199952 -0.00123813]
(15, 6, 0): [-0.1102643 -0.11091634]	(15, 6, 0): [-0.10430154 -0.10466052]
(15, 6, 1): [-0.00199905 -0.00070533]	(15, 6, 1): [-0.000999 0.00195319]
(15, 7, 0): [-0.16378943 -0.16310226]	(15, 7, 0): [-0.17936495 -0.17812147]
(15, 7, 1): [-0.0029994 -0.00266892]	(15, 7, 1): [-0.0029992 -0.00263958]
(15, 8, 0): [-0.19778255 -0.19689583]	(15, 8, 0): [-0.19729703 -0.19764187]
(15, 8, 1): [-0.00299895 -0.00228344]	(15, 8, 1): [-0.004999 -0.00436886]
(15, 9, 0): [-0.19531472 -0.19714597]	(15, 9, 0): [-0.20474397 -0.20382515]
(15, 9, 1): [-0.0019994 -0.00172562]	(15, 9, 1): [-0.00499728 -0.00301884]
(16, 1, 0): [-0.251446 -0.25131625]	(16, 1, 0): [-0.28288687 -0.28230008]
(16, 1, 1): [-0.00399832 -0.00303078]	(16, 1, 1): [-0.006997 -0.00611648]
(16, 10, 0): [-0.5392125 -0.53699465]	(16, 10, 0): [-0.49077189 -0.48339772]
(16, 10, 1): [-0.03781504 -0.03618371]	(16, 10, 1): [-0.04481708 -0.04468699]
(16, 2, 0): [-0.10109589 -0.11548704]	(16, 2, 0): [-0.15155883 -0.15198786]
(16, 2, 1): [-0.001 -0.00204228]	(16, 2, 1): [-0.00199908 -0.00029625]
(16, 3, 0): [-0.14947795 -0.15136251]	(16, 3, 0): [-0.13574284 -0.13540493]
(16, 3, 1): [-0.0029984 -0.00222543]	(16, 3, 1): [-0.0019999 -0.00170638]
(16, 4, 0): [-0.12572596 -0.12742099]	(16, 4, 0): [-0.10883375 -0.10620541]
(16, 4, 1): [-0.0019998 -0.00177192]	(16, 4, 1): [-0.0019987 -0.00136772]
(16, 5, 0): [-0.09186284 -0.09312225]	(16, 5, 0): [-0.11806363 -0.11689608]
(16, 5, 1): [-0.0020016 -0.00099323]	(16, 5, 1): [-0.0020008 -0.00122184]
(16, 6, 0): [-0.10057434 -0.10184799]	(16, 6, 0): [-0.0901314 -0.08957956]
(16, 6, 1): [-0.001 -0.00030277]	(16, 6, 1): [-0.001 0.00230864]
(16, 7, 0): [-0.20779492 -0.20779975]	(16, 7, 0): [-0.20415527 -0.20267915]
(16, 7, 1): [-0.0039966 -0.00323733]	(16, 7, 1): [-0.0049981 -0.00432947]
(16, 8, 0): [-0.21119988 -0.21274952]	(16, 8, 0): [-0.20776502 -0.2070658]
(16, 8, 1): [-0.00299802 -0.00239466]	(16, 8, 1): [-0.0059979 -0.0058107]
(16, 9, 0): [-0.21094747 -0.21140377]	(16, 9, 0): [-0.22627946 -0.22593531]
(16, 9, 1): [-0.0039987 -0.00399284]	(16, 9, 1): [-0.0049984 -0.00404141]
(17, 1, 0): [-0.28131281 -0.2813964]	(17, 1, 0): [-0.25907026 -0.25752277]
(17, 1, 1): [-0.0069946 -0.00657227]	(17, 1, 1): [-0.0059985 -0.00571674]
(17, 10, 0): [-0.53278838 -0.52505158]	(17, 10, 0): [-0.53128419 -0.48804878]
(17, 10, 1): [-0.04172878 -0.04200262]	(17, 10, 1): [-0.05871183 -0.05818822]
(17, 2, 0): [-0.08794529 -0.09062707]	(17, 2, 0): [-0.09883276 -0.09968721]
(17, 2, 1): [-0.00399807 -0.00314183]	(17, 2, 1): [-0.0030001 -0.00220007]
(17, 3, 0): [-0.06151946 -0.06555984]	(17, 3, 0): [-0.04199755 -0.04529477]
(17, 3, 1): [-0.0020018 -0.00150533]	(17, 3, 1): [-0.00299831 -0.00248645]
(17, 4, 0): [-0.06719801 -0.06806682]	(17, 4, 0): [-0.06204104 -0.06436324]
(17, 4, 1): [-0.00099885 -0.00096326]	(17, 4, 1): [-0.001 0.000491]
(17, 5, 0): [0.00967915 -0.0197014]	(17, 5, 0): [-0.00175589 -0.01943127]
(17, 5, 1): [-0.0010002 0.00121666]	(17, 5, 1): [-0.0010001 0.00024273]
(17, 6, 0): [-0.01372928 -0.03817637]	(17, 6, 0): [0.0215194 -0.00875212]
(17, 6, 1): [-0.0010004 -0.00074585]	(17, 6, 1): [6.86835190e-03 -4.61492031e-05]
(17, 7, 0): [-0.04937951 -0.0521062]	(17, 7, 0): [-0.05437276 -0.0593351]
(17, 7, 1): [-0.00200785 -0.00201788]	(17, 7, 1): [-0.0039967 -0.00309866]
(17, 8, 0): [-0.19516022 -0.1954052]	(17, 8, 0): [-0.20349414 -0.2041659]
(17, 8, 1): [-0.00599141 -0.00567549]	(17, 8, 1): [-0.004997 -0.00414792]
(17, 9, 0): [-0.19957734 -0.19909158]	(17, 9, 0): [-0.21063726 -0.21057344]

(17, 9, 1): [-0.0059954 -0.00508665]	(17, 9, 1): [-0.0049993 -0.00462862]
(18, 1, 0): [-0.20555186 -0.20494942]	(18, 1, 0): [-0.19223844 -0.1929854]
(18, 1, 1): [-0.00799619 -0.0078731]	(18, 1, 1): [-0.00998242 -0.00902466]
(18, 10, 0): [-0.46818178 -0.47929775]	(18, 10, 0): [-0.49543476 -0.49221705]
(18, 10, 1): [-0.03630912 -0.03596053]	(18, 10, 1): [-0.04450451 -0.04264947]
(18, 2, 0): [0.08307574 -0.00382397]	(18, 2, 0): [0.13166579 -0.00586227]
(18, 2, 1): [0.00579183 -0.00012102]	(18, 2, 1): [-0.0010004 0.00137406]
(18, 3, 0): [0.08318024 -0.002997]	(18, 3, 0): [0.13702341 -0.00390052]
(18, 3, 1): [-0.001 0.00045355]	(18, 3, 1): [-0.0010001 0.00220888]
(18, 4, 0): [0.11433717 -0.00057684]	(18, 4, 0): [0.06488961 -0.001999]
(18, 4, 1): [-0.00099495 0.00158315]	(18, 4, 1): [1.20391710e-06 2.13072045e-03]
(18, 5, 0): [0.10677621 -0.00177939]	(18, 5, 0): [0.14538278 -0.00896021]
(18, 5, 1): [0.04326297 0.]	(18, 5, 1): [0.00399039 0.00094185]
(18, 6, 0): [0.15500941 -0.00991347]	(18, 6, 0): [0.18787242 -0.00696888]
(18, 6, 1): [0.03329938 0.]	(18, 6, 1): [0.04071359 0.]
(18, 7, 0): [0.25306552 -0.00497707]	(18, 7, 0): [0.29743102 -0.002997]
(18, 7, 1): [5.04111382E-02 3.56918422E-05]	(18, 7, 1): [0.04870642 0.]
(18, 8, 0): [0.05242342 -0.00778159]	(18, 8, 0): [0.10326652 -0.00694747]
(18, 8, 1): [0.01096787 -0.00100768]	(18, 8, 1): [-0.0019996 -0.00164192]
(18, 9, 0): [-0.12194156 -0.1220323]	(18, 9, 0): [-0.16715635 -0.16784165]
(18, 9, 1): [-0.00400069 -0.00375795]	(18, 9, 1): [-0.00699497 -0.00605702]
(19, 1, 0): [-0.1135972 -0.11996732]	(19, 1, 0): [-0.12359115 -0.12615948]
(19, 1, 1): [-0.00599298 -0.00552411]	(19, 1, 1): [-0.00698661 -0.006323]
(19, 10, 0): [-0.02119166 -0.03810091]	(19, 10, 0): [-0.04021194 -0.05149841]
(19, 10, 1): [-0.00559253 -0.01304406]	(19, 10, 1): [0.00019958 -0.02019815]
(19, 2, 0): [0.28550727 -0.0019029]	(19, 2, 0): [0.28780371 -0.00896408]
(19, 2, 1): [-0.001 0.00509061]	(19, 2, 1): [0. 0.00569935]
(19, 3, 0): [0.25497516 -0.00499001]	(19, 3, 0): [0.2501949 -0.00797206]
(19, 3, 1): [-0.00299463 0.00310583]	(19, 3, 1): [5.96756923e-02 -2.51261064e-05]
(19, 4, 0): [0.24802011 -0.00281841]	(19, 4, 0): [0.29427403 -0.00683727]
(19, 4, 1): [3.28384716E-06 8.92762691E-03]	(19, 4, 1): [0.05957945 0.]
(19, 5, 0): [0.27279734 -0.001]	(19, 5, 0): [0.24385352 -0.00597597]
(19, 5, 1): [-7.99840008E-07 5.76554422E-03]	(19, 5, 1): [6.95030450e-02 -4.83935154e-05]
(19, 6, 0): [0.30330344 -0.00598502]	(19, 6, 0): [0.27223895 -0.00398773]
(19, 6, 1): [0.07885813 0.]	(19, 6, 1): [5.56219058e-02 5.39604233e-06]
(19, 7, 0): [0.42385083 -0.00092139]	(19, 7, 0): [0.44975042 -0.0065527]
(19, 7, 1): [9.64729772E-02 -7.71784600E-07]	(19, 7, 1): [-0.00099223 0.00794607]
(19, 8, 0): [0.44357466 -0.00499001]	(19, 8, 0): [0.46341073 -0.00498911]
(19, 8, 1): [0.10051123 0.]	(19, 8, 1): [0.11403721 0.]
(19, 9, 0): [0.18546012 -0.003994]	(19, 9, 0): [0.16446762 -0.00625042]
(19, 9, 1): [0.03630851 -0.00015116]	(19, 9, 1): [3.77063522e-02 1.71513568e-05]
(20, 1, 0): [0.10827361 -0.01290403]	(20, 1, 0): [0.17005625 -0.00779751]
(20, 1, 1): [0.0165099 -0.0001563]	(20, 1, 1): [0.02782912 -0.00200065]
(20, 10, 0): [1.43228083 -0.026451]	(20, 10, 0): [1.71068224 -0.03730536]
(20, 10, 1): [-0.00093584 0.08460467]	(20, 10, 1): [0.00101369 0.08348124]
(20, 2, 0): [0.59320258 -0.001999]	(20, 2, 0): [0.64799479 -0.00493007]
(20, 2, 1): [9.94244188E-02 -5.74770780E-05]	(20, 2, 1): [0.10502958 0.00043024]
(20, 3, 0): [0.5951649 -0.00598502]	(20, 3, 0): [0.62683501 -0.00596917]
(20, 3, 1): [-0.001 0.01573952]	(20, 3, 1): [0.10518287 -0.00020273]
(20, 4, 0): [0.57218888 -0.002997]	(20, 4, 0): [0.62515322 -0.00797206]

(20, 4, 1): [1.23733357E-01 9.12684228E-06]	(20, 4, 1): [0.10801411 0.]
(20, 5, 0): [0.60402908 -0.00589921]	(20, 5, 0): [0.65919939 -0.01094516]
(20, 5, 1): [1.19867208E-01 -7.71831881E-06]	(20, 5, 1): [0.10227304 0.]
(20, 6, 0): [0.56759511 -0.00499001]	(20, 6, 0): [0.61999881 -0.00697903]
(20, 6, 1): [0.11530749 0.]	(20, 6, 1): [0.1141179 0.00039329]
(20, 7, 0): [0.74481619 -0.002997]	(20, 7, 0): [0.79034145 -0.00773721]
(20, 7, 1): [0.1258421 0.]	(20, 7, 1): [1.26920001e-01 -9.00539811e-07]
(20, 8, 0): [0.76617557 -0.0117553]	(20, 8, 0): [0.90101129 -0.01291565]
(20, 8, 1): [1.42489713E-01 -9.27886511E-05]	(20, 8, 1): [0.00201497 0.01821832]
(20, 9, 0): [0.75146605 -0.001999]	(20, 9, 0): [0.83803067 -0.00598502]
(20, 9, 1): [0.12311355 0.]	(20, 9, 1): [0.11308409 0.00044049]
(21, 1, 0): [0.24186408 -0.001]	(21, 1, 0): [0.3008987 -0.00499001]
(21, 1, 1): [2.96562982E-01 1.75721831E-04]	(21, 1, 1): [2.72025933e-01 2.25001026e-04]
(21, 10, 0): [1.43362637 -0.00797206]	(21, 10, 0): [1.48450176 -0.01489545]
(21, 10, 1): [1.29077395E+00 1.4763378E-04]	(21, 10, 1): [1.50034276 0.00325703]
(21, 2, 0): [0.2563248 0.]	(21, 2, 0): [0.27650622 -0.001999]
(21, 2, 1): [4.05092994E-01 1.01697097E-04]	(21, 2, 1): [4.64582806e-01 -4.49495961e-06]
(21, 3, 0): [0.27163003 -0.002997]	(21, 3, 0): [0.22804853 -0.002997]
(21, 3, 1): [4.30703870E-01 3.53156402E-04]	(21, 3, 1): [0.31868942 0.00185541]
(21, 4, 0): [0.22704494 0.]	(21, 4, 0): [0.23296879 -0.001999]
(21, 4, 1): [4.19799107E-01 1.12812668E-06]	(21, 4, 1): [4.21654942e-01 -3.59820108e-06]
(21, 5, 0): [0.2206094 -0.001999]	(21, 5, 0): [0.24469918 -0.003994]
(21, 5, 1): [4.17309859E-01 3.09284548E-04]	(21, 5, 1): [0.3890798 0.00042199]
(21, 6, 0): [0.20430944 -0.001]	(21, 6, 0): [0.18210423 -0.001]
(21, 6, 1): [4.01787204E-01 3.84950537E-04]	(21, 6, 1): [4.39817532e-01 8.66498832e-05]
(21, 7, 0): [0.30276051 -0.001]	(21, 7, 0): [0.34936616 -0.00598502]
(21, 7, 1): [4.09865269E-01 2.45636174E-05]	(21, 7, 1): [3.78498046e-01 1.40503069e-04]
(21, 8, 0): [0.3455727 -0.001]	(21, 8, 0): [0.39049783 -0.002997]
(21, 8, 1): [0.42200655 0.00046928]	(21, 8, 1): [4.51442511e-01 -4.85630128e-05]
(21, 9, 0): [0.33742917 -0.001]	(21, 9, 0): [0.37539488 -0.003994]
(21, 9, 1): [4.16536537E-01 -2.87669361E-04]	(21, 9, 1): [4.21686380e-01 1.64049096e-04]
(22, 1, 0): [0. 0.]	(22, 1, 0): [0. 0.]
(22, 10, 0): [0. 0.]	(22, 10, 0): [0. 0.]
(22, 2, 0): [0. 0.]	(22, 2, 0): [0. 0.]
(22, 3, 0): [0. 0.]	(22, 3, 0): [0. 0.]
(22, 4, 0): [0. 0.]	(22, 4, 0): [0. 0.]
(22, 5, 0): [0. 0.]	(22, 5, 0): [0. 0.]
(22, 6, 0): [0. 0.]	(22, 6, 0): [0. 0.]
(22, 7, 0): [0. 0.]	(22, 7, 0): [0. 0.]
(22, 8, 0): [0. 0.]	(22, 8, 0): [0. 0.]
(22, 9, 0): [0. 0.]	(22, 9, 0): [0. 0.]
(23, 1, 0): [0. 0.]	(23, 1, 0): [0. 0.]
(23, 10, 0): [0. 0.]	(23, 10, 0): [0. 0.]
(23, 2, 0): [0. 0.]	(23, 2, 0): [0. 0.]
(23, 3, 0): [0. 0.]	(23, 3, 0): [0. 0.]
(23, 4, 0): [0. 0.]	(23, 4, 0): [0. 0.]
(23, 5, 0): [0. 0.]	(23, 5, 0): [0. 0.]
(23, 6, 0): [0. 0.]	(23, 6, 0): [0. 0.]
(23, 7, 0): [0. 0.]	(23, 7, 0): [0. 0.]
(23, 8, 0): [0. 0.]	(23, 8, 0): [0. 0.]

(23, 9, 0): [0. 0.]	(23, 9, 0): [0. 0.]
(24, 1, 0): [0. 0.]	(24, 1, 0): [0. 0.]
(24, 10, 0): [0. 0.]	(24, 10, 0): [0. 0.]
(24, 2, 0): [0. 0.]	(24, 2, 0): [0. 0.]
(24, 3, 0): [0. 0.]	(24, 3, 0): [0. 0.]
(24, 4, 0): [0. 0.]	(24, 4, 0): [0. 0.]
(24, 5, 0): [0. 0.]	(24, 5, 0): [0. 0.]
(24, 6, 0): [0. 0.]	(24, 6, 0): [0. 0.]
(24, 7, 0): [0. 0.]	(24, 7, 0): [0. 0.]
(24, 8, 0): [0. 0.]	(24, 8, 0): [0. 0.]
(24, 9, 0): [0. 0.]	(24, 9, 0): [0. 0.]
(25, 1, 0): [0. 0.]	(25, 1, 0): [0. 0.]
(25, 10, 0): [0. 0.]	(25, 10, 0): [0. 0.]
(25, 2, 0): [0. 0.]	(25, 2, 0): [0. 0.]
(25, 3, 0): [0. 0.]	(25, 3, 0): [0. 0.]
(25, 4, 0): [0. 0.]	(25, 4, 0): [0. 0.]
(25, 5, 0): [0. 0.]	(25, 5, 0): [0. 0.]
(25, 6, 0): [0. 0.]	(25, 6, 0): [0. 0.]
(25, 7, 0): [0. 0.]	(25, 7, 0): [0. 0.]
(25, 8, 0): [0. 0.]	(25, 8, 0): [0. 0.]
(25, 9, 0): [0. 0.]	(25, 9, 0): [0. 0.]
(26, 1, 0): [0. 0.]	(26, 1, 0): [0. 0.]
(26, 10, 0): [0. 0.]	(26, 10, 0): [0. 0.]
(26, 2, 0): [0. 0.]	(26, 2, 0): [0. 0.]
(26, 3, 0): [0. 0.]	(26, 3, 0): [0. 0.]
(26, 4, 0): [0. 0.]	(26, 4, 0): [0. 0.]
(26, 5, 0): [0. 0.]	(26, 5, 0): [0. 0.]
(26, 6, 0): [0. 0.]	(26, 6, 0): [0. 0.]
(26, 7, 0): [0. 0.]	(26, 7, 0): [0. 0.]
(26, 8, 0): [0. 0.]	(26, 8, 0): [0. 0.]
(26, 9, 0): [0. 0.]	(26, 9, 0): [0. 0.]
(27, 1, 0): [0. 0.]	(27, 1, 0): [0. 0.]
(27, 10, 0): [0. 0.]	(27, 10, 0): [0. 0.]
(27, 2, 0): [0. 0.]	(27, 2, 0): [0. 0.]
(27, 3, 0): [0. 0.]	(27, 3, 0): [0. 0.]
(27, 4, 0): [0. 0.]	(27, 4, 0): [0. 0.]
(27, 5, 0): [0. 0.]	(27, 5, 0): [0. 0.]
(27, 6, 0): [0. 0.]	(27, 6, 0): [0. 0.]
(27, 7, 0): [0. 0.]	(27, 7, 0): [0. 0.]
(27, 8, 0): [0. 0.]	(27, 8, 0): [0. 0.]
(27, 9, 0): [0. 0.]	(27, 9, 0): [0. 0.]
(28, 1, 0): [0. 0.]	(28, 1, 0): [0. 0.]
(28, 10, 0): [0. 0.]	(28, 10, 0): [0. 0.]
(28, 2, 0): [0. 0.]	(28, 2, 0): [0. 0.]
(28, 3, 0): [0. 0.]	(28, 4, 0): [0. 0.]
(28, 5, 0): [0. 0.]	(28, 5, 0): [0. 0.]
(28, 6, 0): [0. 0.]	(28, 6, 0): [0. 0.]
(28, 7, 0): [0. 0.]	(28, 7, 0): [0. 0.]
(28, 8, 0): [0. 0.]	(28, 8, 0): [0. 0.]
(28, 9, 0): [0. 0.]	(28, 9, 0): [0. 0.]

(29, 1, 0): [0. 0.]	(29, 1, 0): [0. 0.]
(29, 10, 0): [0. 0.]	(29, 10, 0): [0. 0.]
	(29, 2, 0): [0. 0.]
(29, 3, 0): [0. 0.]	(29, 3, 0): [0. 0.]
(29, 4, 0): [0. 0.]	(29, 4, 0): [0. 0.]
	(29, 5, 0): [0. 0.]
(29, 6, 0): [0. 0.]	(29, 6, 0): [0. 0.]
	(29, 7, 0): [0. 0.]
(29, 8, 0): [0. 0.]	(29, 8, 0): [0. 0.]
(29, 9, 0): [0. 0.]	(29, 9, 0): [0. 0.]
(30, 1, 0): [0. 0.]	(30, 1, 0): [0. 0.]
(30, 10, 0): [0. 0.]	(30, 10, 0): [0. 0.]
(30, 3, 0): [0. 0.]	(30, 3, 0): [0. 0.]
(30, 4, 0): [0. 0.]	(30, 4, 0): [0. 0.]
(30, 5, 0): [0. 0.]	(30, 5, 0): [0. 0.]
(30, 6, 0): [0. 0.]	
(30, 7, 0): [0. 0.]	(30, 7, 0): [0. 0.]
(30, 8, 0): [0. 0.]	(30, 8, 0): [0. 0.]
(30, 9, 0): [0. 0.]	(30, 9, 0): [0. 0.]
	(31, 1, 0): [0. 0.]
	(31, 10, 0): [0. 0.]
	(31, 3, 0): [0. 0.]
	(31, 5, 0): [0. 0.]
	(31, 7, 0): [0. 0.]
	(31, 8, 0): [0. 0.]
(31, 9, 0): [0. 0.]	(31, 9, 0): [0. 0.]

These are the Q-Tables learned by the agent in the two configurations. We can easily notice that the states where the player's card sum exceeds 21 are never updated; in these states, no action is required as the episode terminates with the player losing the game. There are many states where the expected rewards for both possible actions are close and negative, indicating that the subsequent actions and resulting states generally lead to a defeat for the player. On the other hand, in states where the player's card sum is high, action 0 (stick) tends to have a higher expected reward, while action 1 (hit) has a lower value.

Solution using a DQN

Another possible implementation for solving the problem based on the Q-learning algorithm is to use a Deep Q-Network. Instead of storing the Q-values inside a table, we use a Neural Network to take a good approximation of the Q-function. This problem may be used for every kind of environment, but it is suitable for those environments that are continuous or too large to be stored inside a table. In fact, if would use a tabular approach with a continuous environment, we will have to discretize the state, and this is not an easy task. If the division is too fine, it's likely to have a large number of states never visited and this will lead to a huge waste of resources. On the other hand, if it is too rough, then the approximation will not be precise enough.

Replay Buffer

The replay buffer is a data structure used during the training of the model. Initially it is initialized as an empty buffer with a fixed size. During the training phase, every time that an action is taken, a tuple containing <current_state, action, reward, new_state, terminated> is added to the buffer. The replay buffer behaves like a queqe, elements are inserted in a FIFO manner, and when the maximum capacity is reached, older elements are removed by the buffer. Finally, we sample a group of experiences from the buffer, and we use them to update the values of the Q-function. The use of the replay buffer introduces other 2 parameters:

- The deque_maxlen, this parameter tells us what the maximum size of the buffer is.
- The batch_size, this parameter tells us how many experiences are sampled from the buffer when we want to update the Q-function.

```
# Train the agent
for episode in range(train_episodes):
  if episode%25==0:
    # Save the trained model
    agent.test(episode)
    agent.save_model(f"blackjack_dqn_model_{episode}.h5")
  state = agent.one_hot_encode(env.reset())
  for step in range(max_steps):
    action = agent.action(state)
    new_state, reward, terminated, _ = env.step(action)
    new_state = agent.one_hot_encode(new_state)
    agent.add_experience(state, action, reward, new_state, terminated)
    state = new_state
    if terminated:
      break
  agent.replay(batch_size)
  print(f"Episode: {episode + 1}, Epsilon: {agent.epsilon:.2f}")

agent.test(500)
agent.save_model(f"blackjack_dqn_model_500.h5")
```

The main structure of the algorithm for the implementation is essentially the same we have seen before. The main differences are caused by the replay buffer, at every step a new experience is added to the buffer, and at the end of every episode it uses the buffer to adjust the values of the Q-function. For every 25 episodes the model is saved and tested over 1000 episodes, in this way we produce data to evaluate the training of the model.

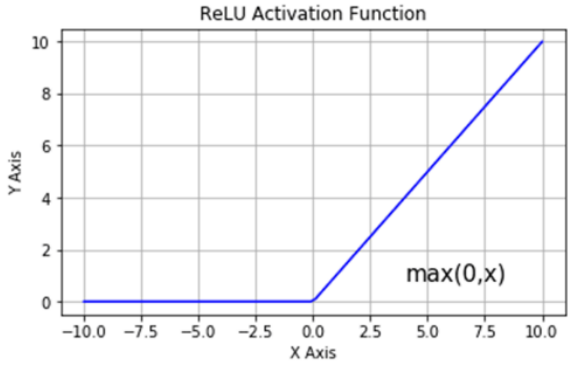
```
def action(self, state):
    if np.random.rand() < self.epsilon:
        return self.action_space.sample()
    return np.argmax(self.model.predict(state)[0])

def predict(self, state):
    return np.argmax(self.model.predict(state)[0])
def replay(self, batch_size):
    if len(self.experience) < batch_size:
        return
    minibatch = random.sample(self.experience, batch_size)
    for state, action, reward, new_state, terminated in minibatch:
        target = reward
        if not terminated:
            target += self.gamma * np.max(self.model.predict(new_state)[0])
        target_function = self.model.predict(state)
        target_function[0][action] = target
        self.model.fit(state, target_function, epochs=1, verbose=0)

    if self.epsilon > self.min_eps:
        self.epsilon *= self.eps_decay
```

The replay function samples the experiences in the buffer to create a minibatch. The agent then iterates over each experience extracted from the replay buffer, computes the new target value for each, and assigns this new value to the target function. The final line of the loop adjusts the weights of the model based on the updated target function. The advantage of using multiple experiences is to obtain a more suitable approximation of the Q-function, makes the learning phase more stable and this technique reduces the risk of overfitting to the training data. The action to perform is chosen as previously seen, following the epsilon-greedy policy, the only difference is that the best action is not chosen by looking at the table and it chooses the higher value, but it is produced as output of the network itself. This approach, where the same network it is used to obtain the necessary values for its own training, it is known as single DQN.

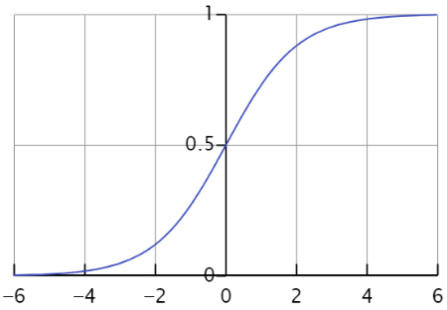
The architecture

In order to evaluate the learning of the model I executed several experiments, changing the parameters that I had presented so far and changing the model of the neural network. In this section I will talk about the architecture of the network. In particular, I used 3 different models and all of them were made by one input layer with 24 nodes, one output layer with 2 nodes, one node for each action in the action space and one hidden layer with 24 nodes. The network is dense, this means that every node belonging to the i-th layer is connected to all the nodes of the i-1-th layer and of the i+1-th layer, if any. In the output layer I always used a linear function as activation function, what I changed are the activation functions of the other layers. I tried with 3 different activation functions:

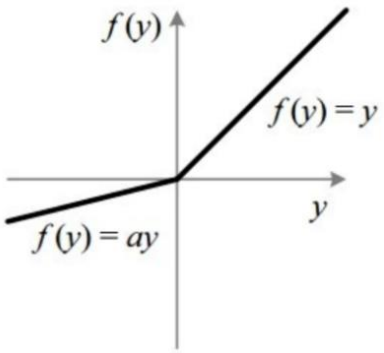
- In the first case I used a ReLU (Rectified Linear Unit) function as activation function for the nodes of the first 2 layers. A ReLU function has the following shape.



- In the second case I used a Sigmoid function as activation function for the nodes of the first 2 layers. A Sigmoid function has the following shape.



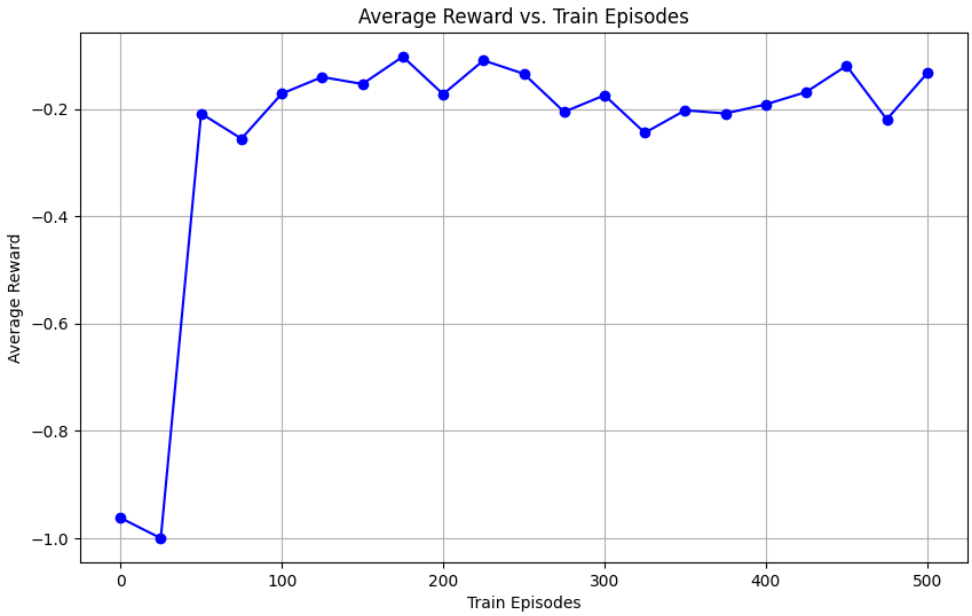
- In the third case I used a Leaky ReLU function as activation function for the nodes of the first 2 layers. A Leaky ReLU function has the following shape, it is very similar to the ReLU function, but it has a small slope for negative values instead of a flat slope. The value for the parameter a is decided before the training, and in all my experiments I set it to 0.01.



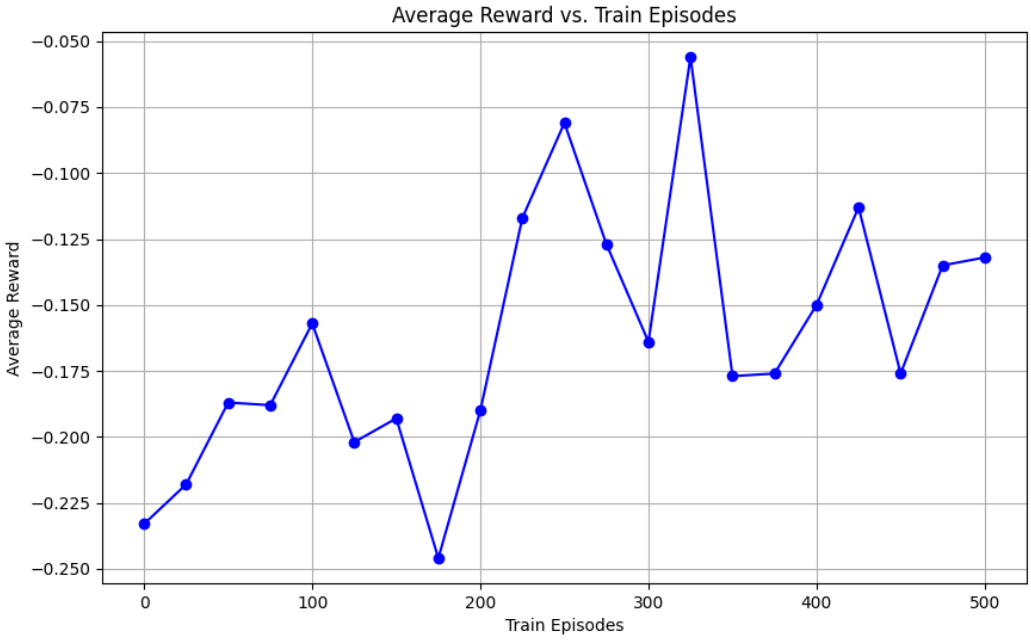
Configurations

In this section I will show the configurations I used for the training of the network with the obtained results. The results represent the average reward obtained over 1000 testing episodes and I measured the results obtained after every 25 episodes of training.

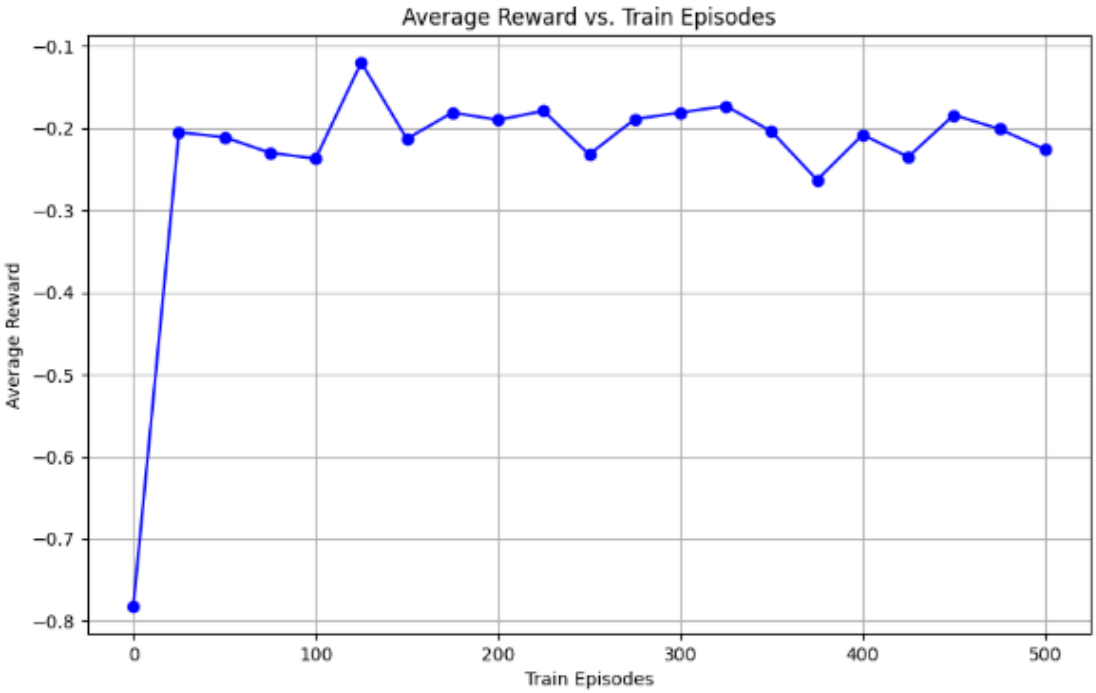
- Configuration 1: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



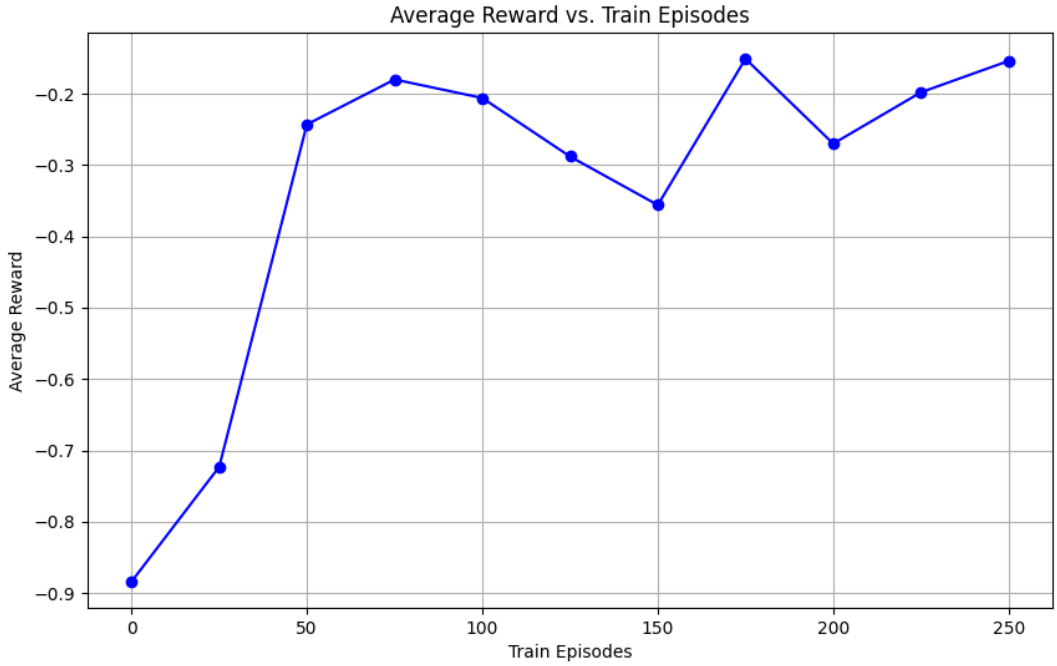
- Configuration 2: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.9, gamma = 0.99.



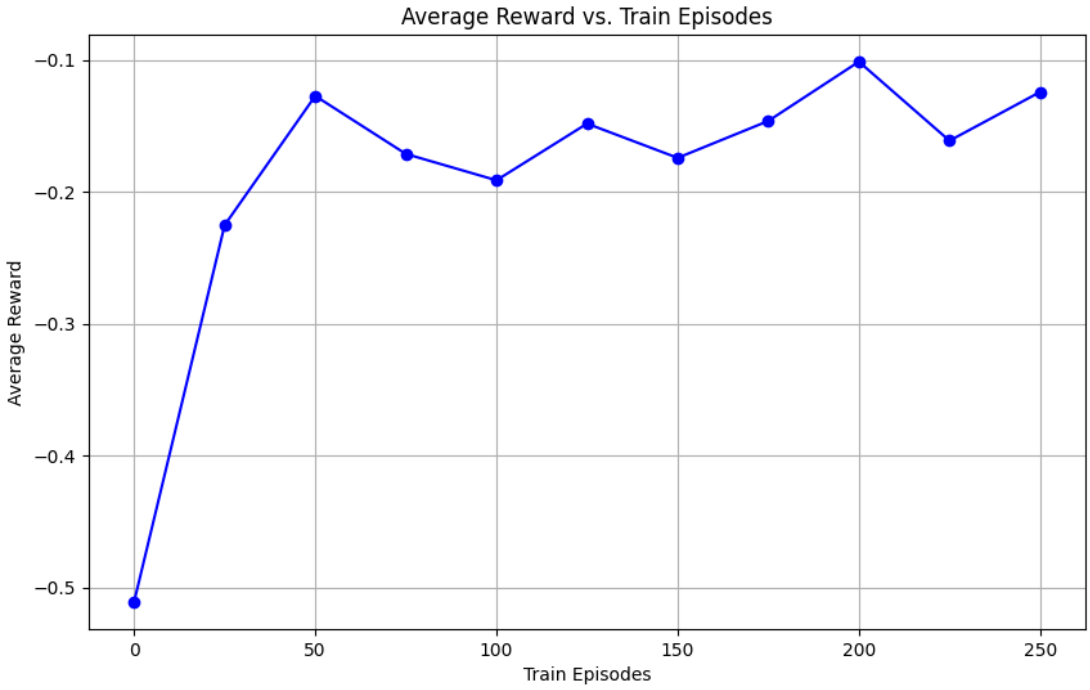
- Configuration 3: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.9, gamma = 0.9.



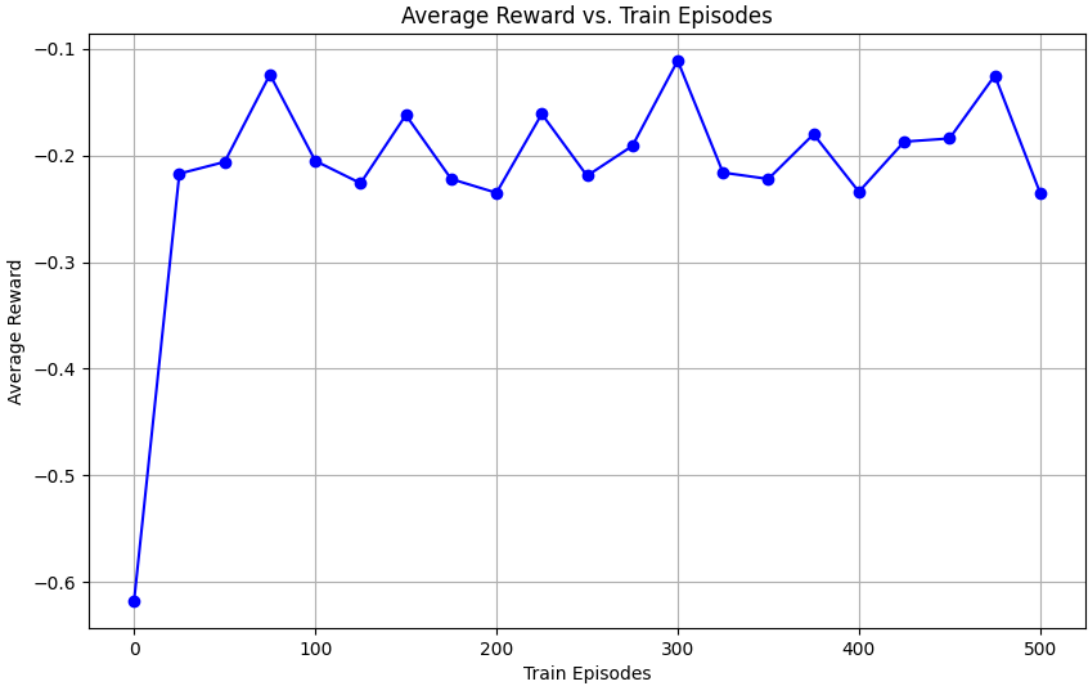
- Configuration 4: the network uses the ReLU function as activation functions, train_episodes = 250, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.9, gamma = 0.9.



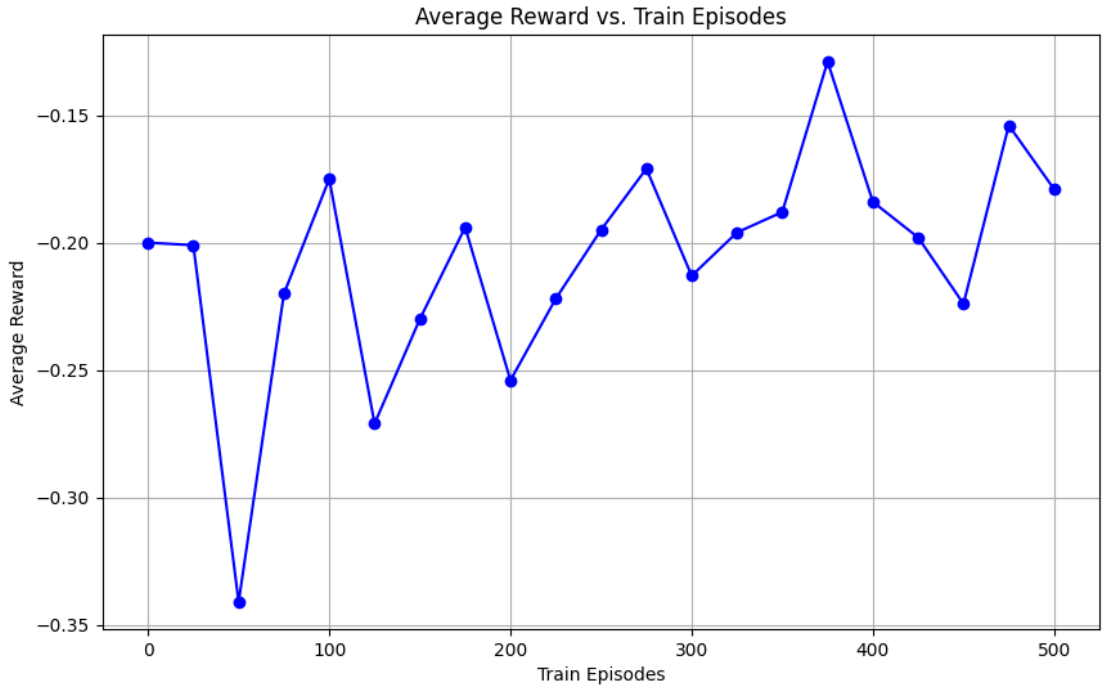
- Configuration 5: the network uses the ReLU function as activation functions, train_episodes = 250, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.99, gamma = 0.99.



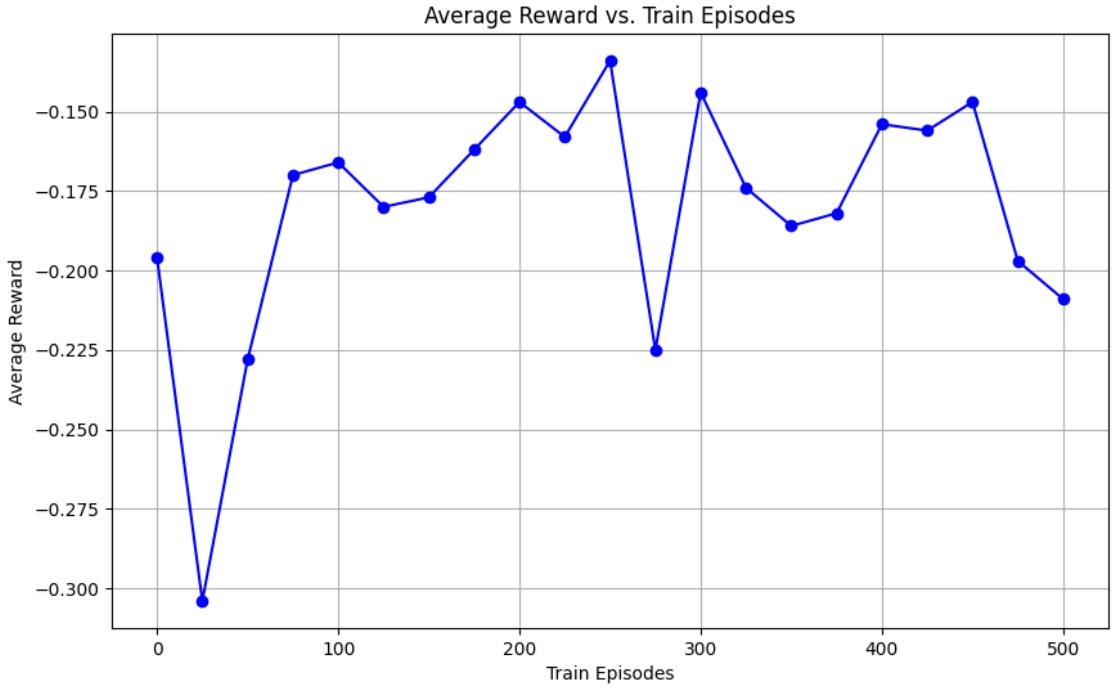
- Configuration 6: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=16, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



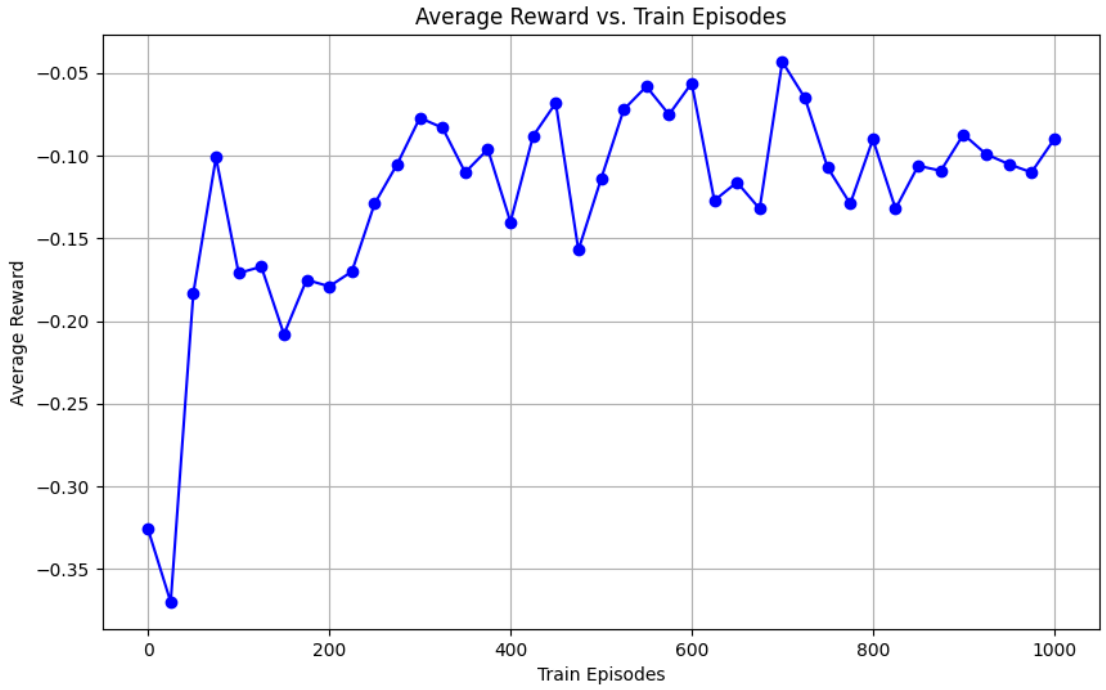
- Configuration 7: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 10000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



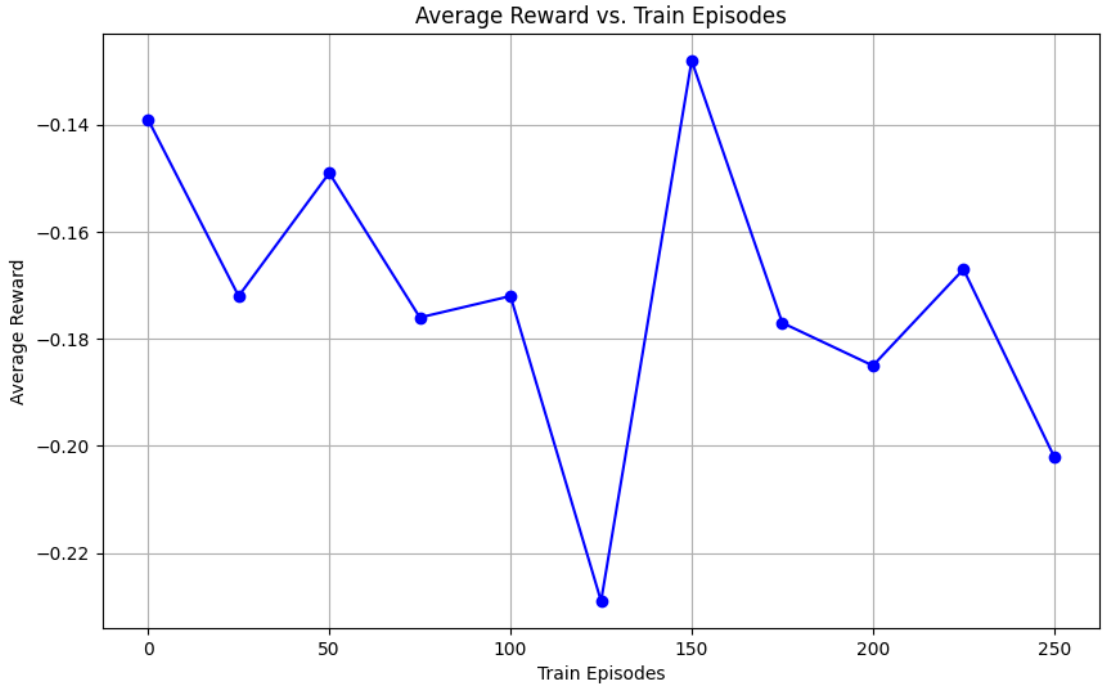
- Configuration 8: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=16, deque_maxlen = 10000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



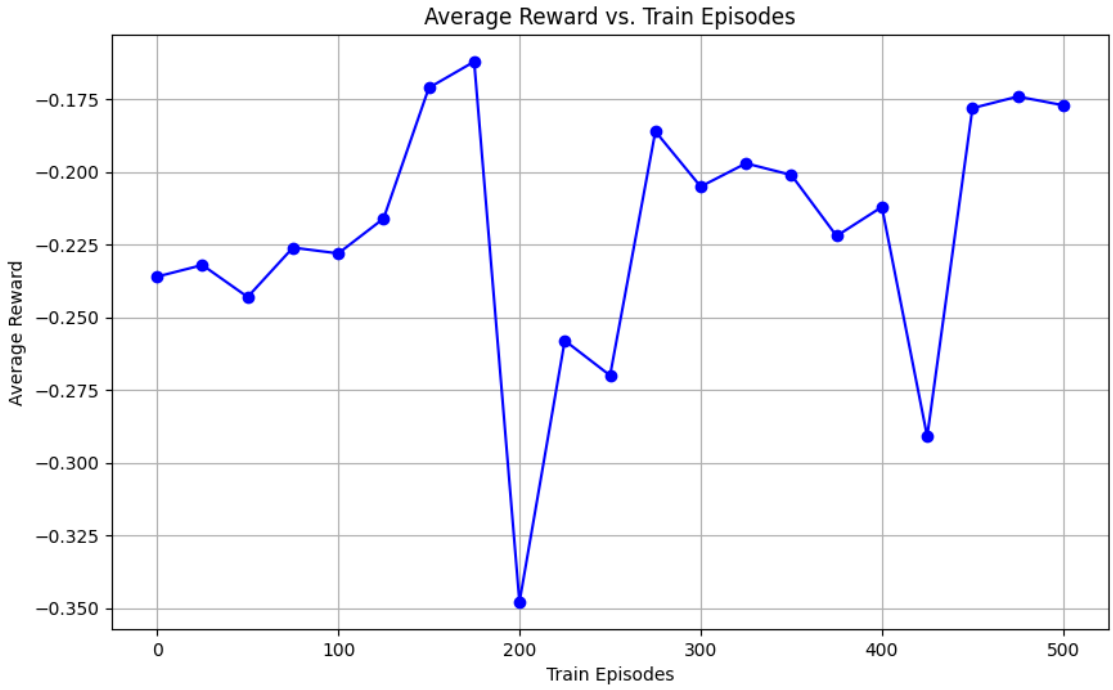
- Configuration 9: the network uses the ReLU function as activation functions, train_episodes = 1000, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



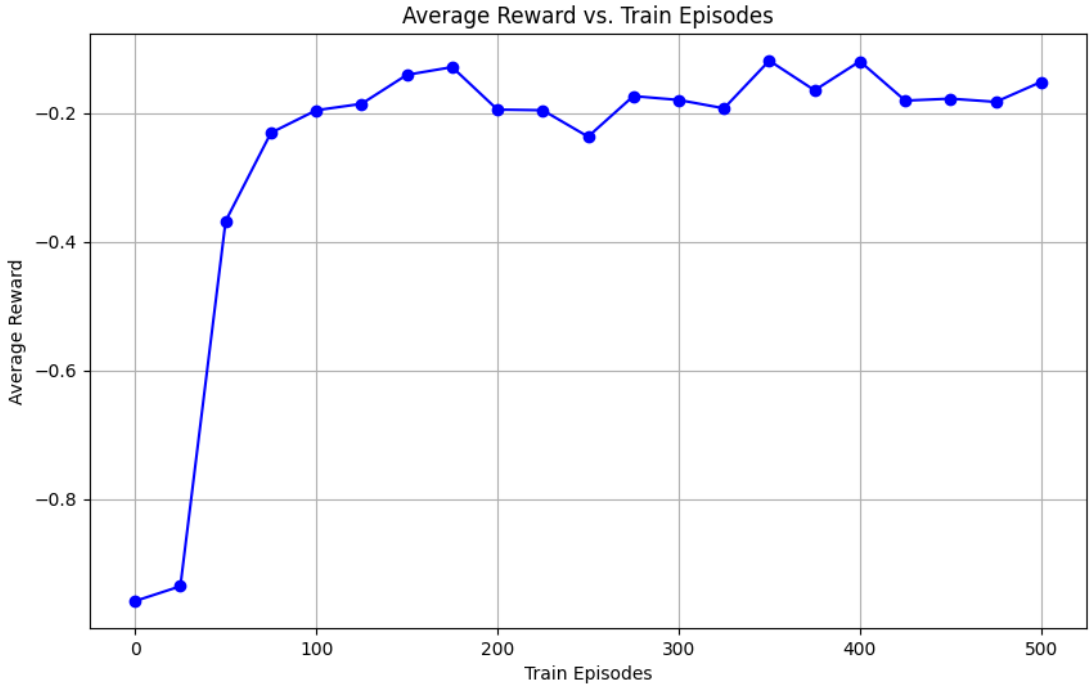
- Configuration 10: the network uses the ReLU function as activation functions, train_episodes = 250, batch_size=64, deque_maxlen = 10000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



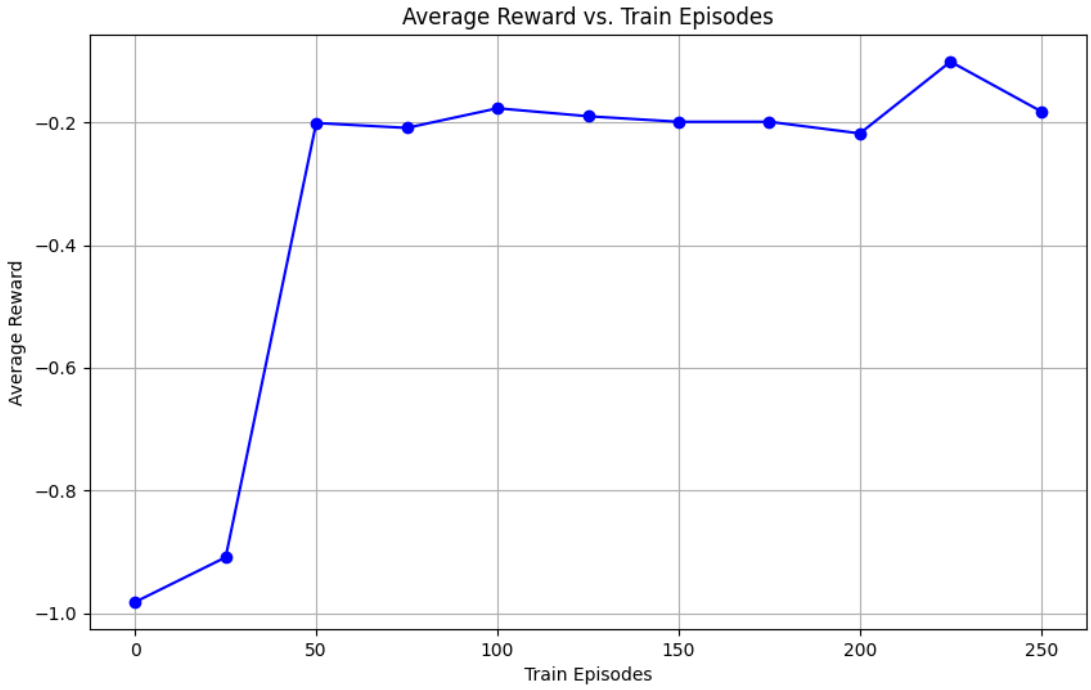
- Configuration 11: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=64, deque_maxlen = 10000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



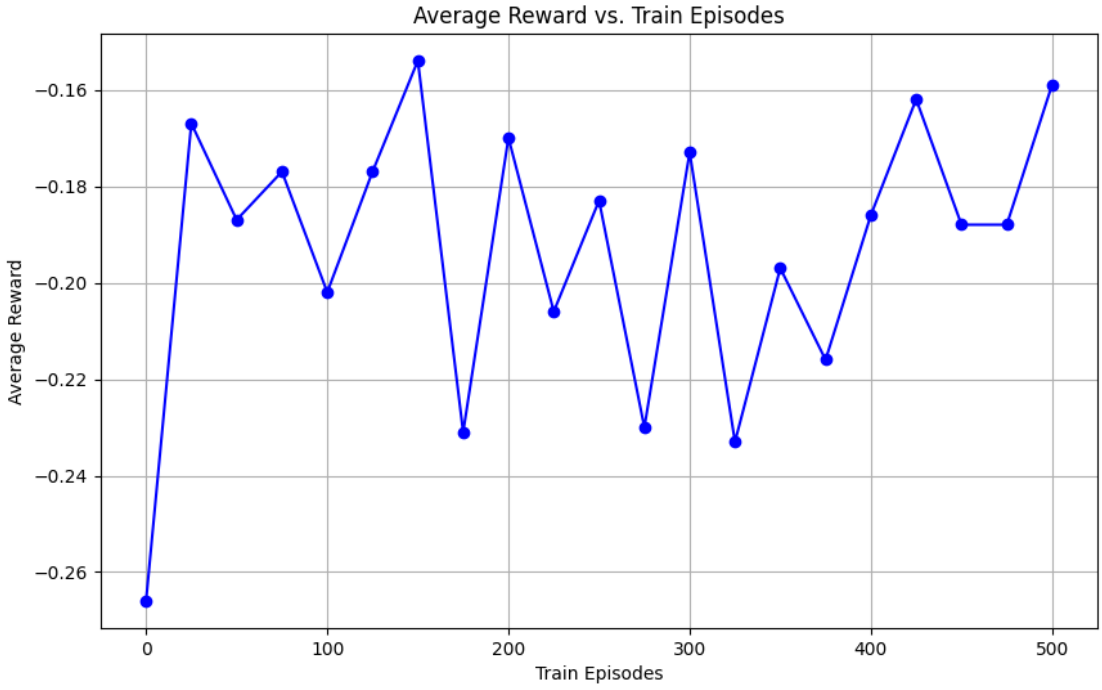
- Configuration 12: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.0001, eps_decay = 0.995, gamma = 0.99.



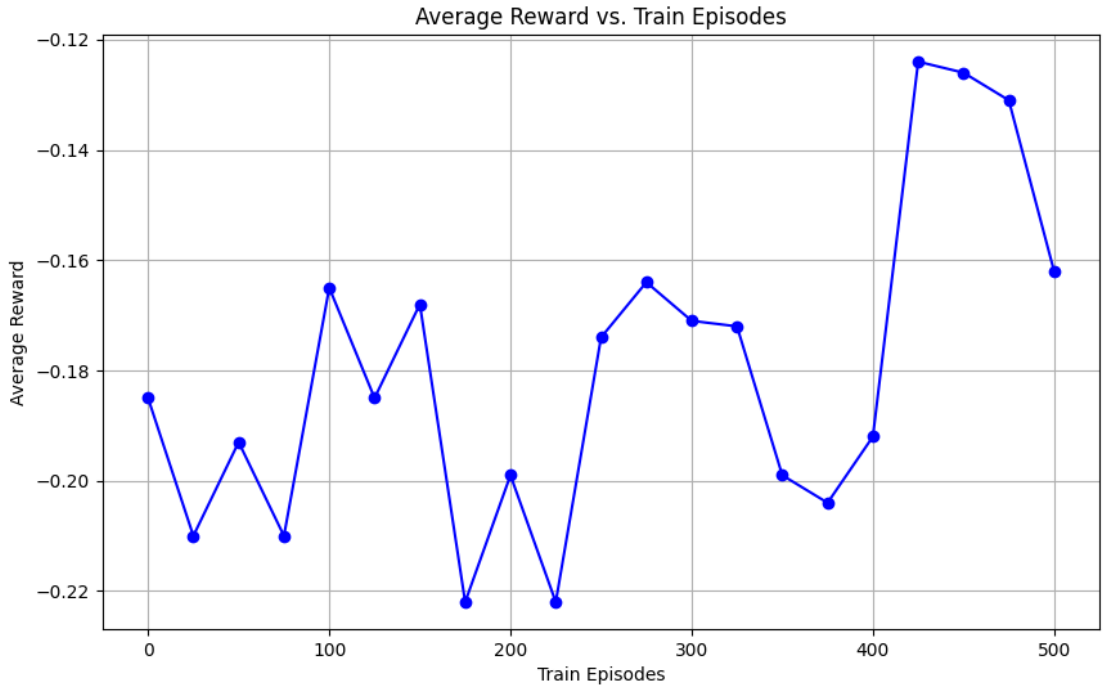
- Configuration 13: the network uses the ReLU function as activation functions, train_episodes = 250, batch_size=32, deque_maxlen = 100000, learning_rate=0.0001, eps_decay = 0.995, gamma = 0.99.



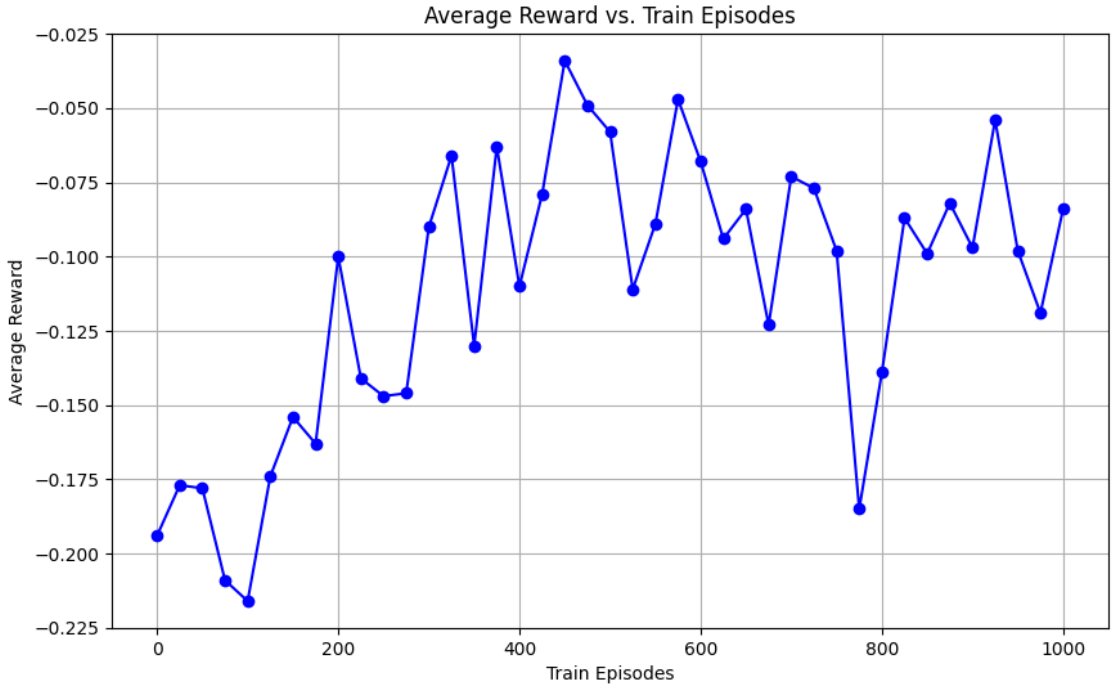
- Configuration 14: the network uses the ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.1, eps_decay = 0.995, gamma = 0.99.



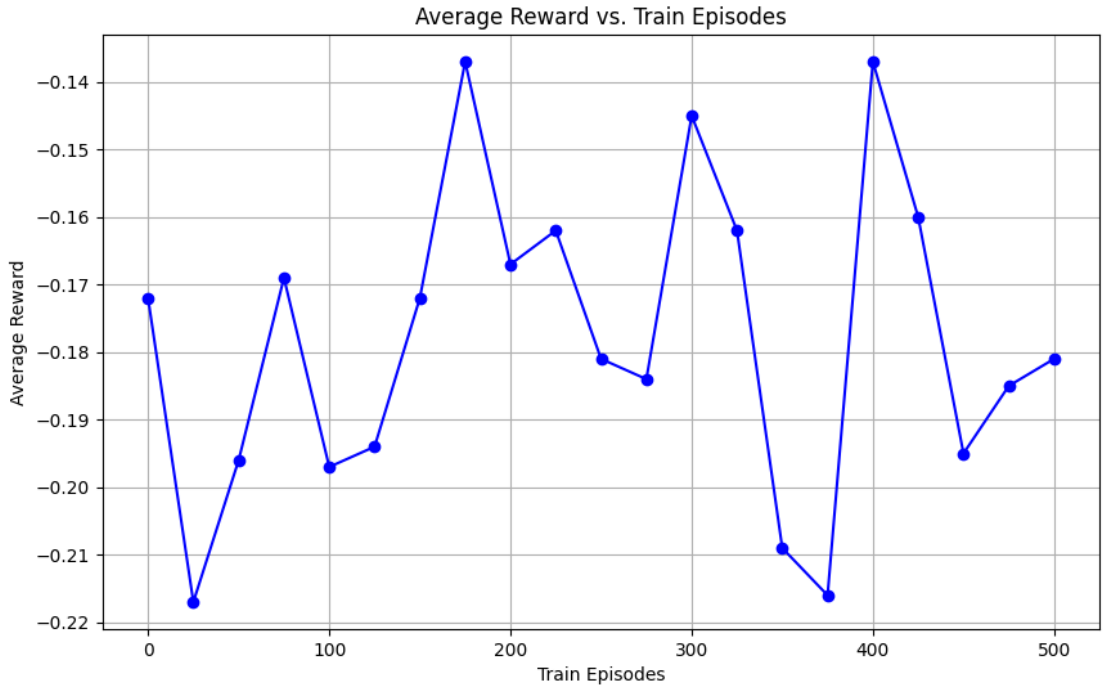
- Configuration 15: the network uses the Sigmoid function as activation functions, train_episodes = 500, batch_size=16, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



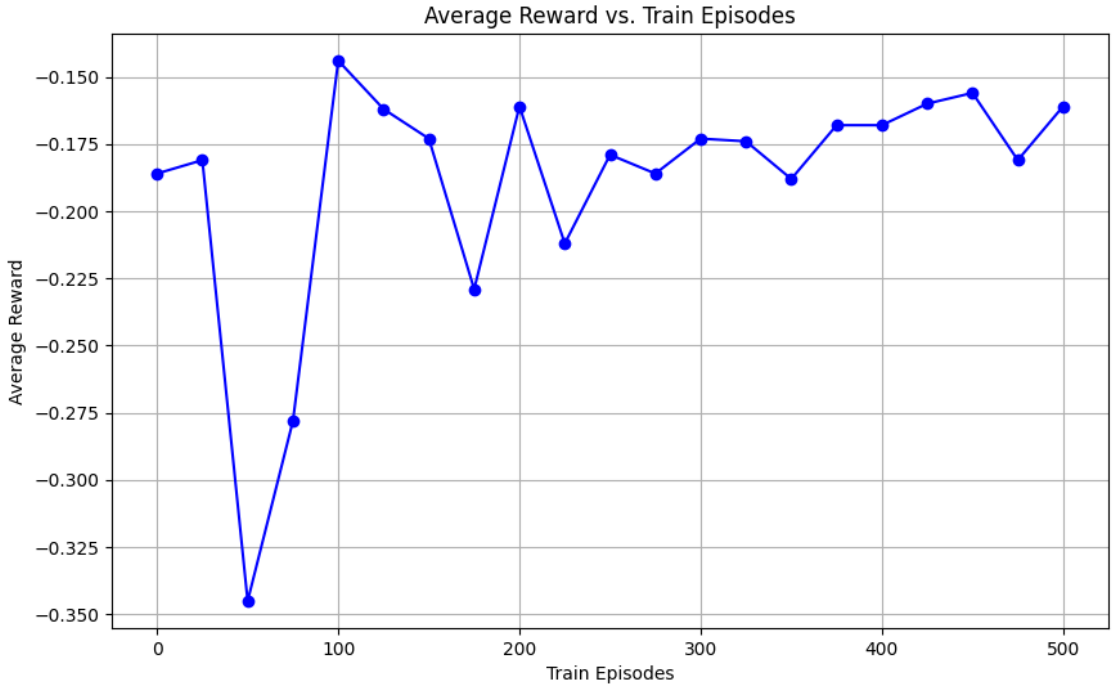
- Configuration 16: the network uses the Sigmoid function as activation functions, train_episodes =1000, batch_size=16, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.9.



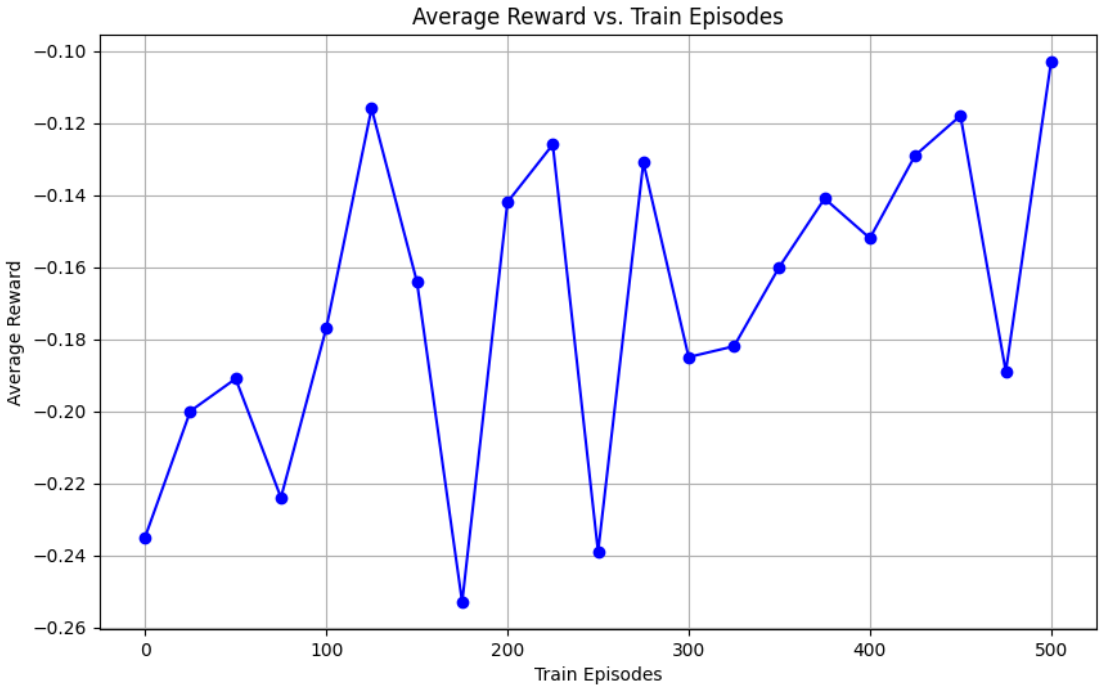
- Configuration 17: the network uses the Sigmoid function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.0001, eps_decay = 0.99, gamma = 0.99.



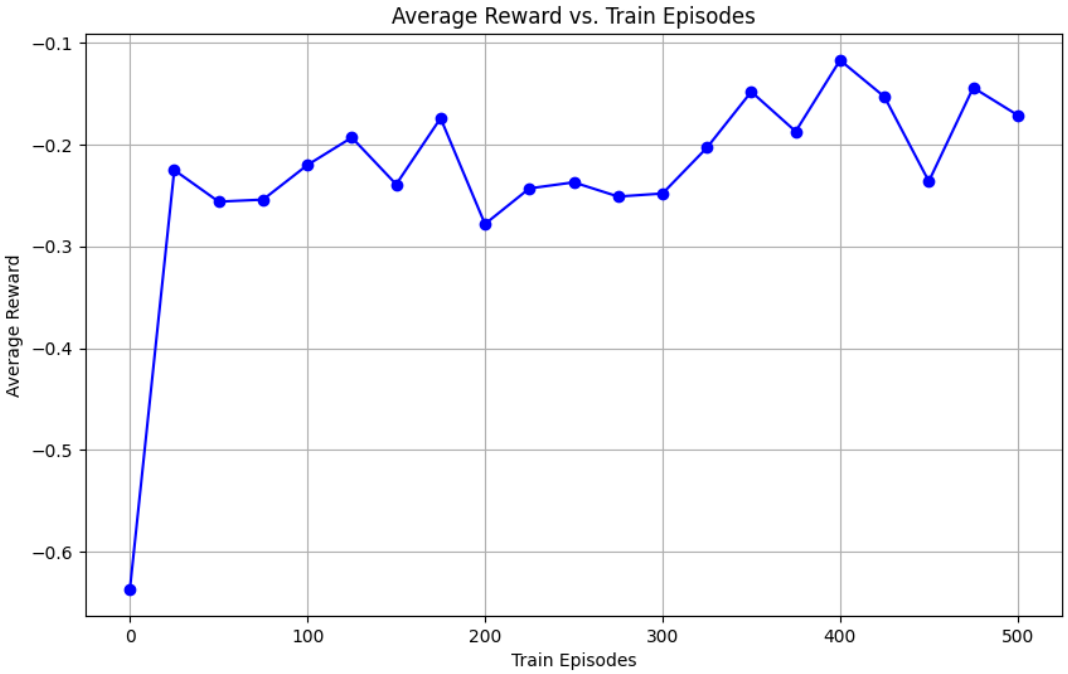
- Configuration 18: the network uses the Sigmoid function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.01, eps_decay = 0.995, gamma = 0.99.



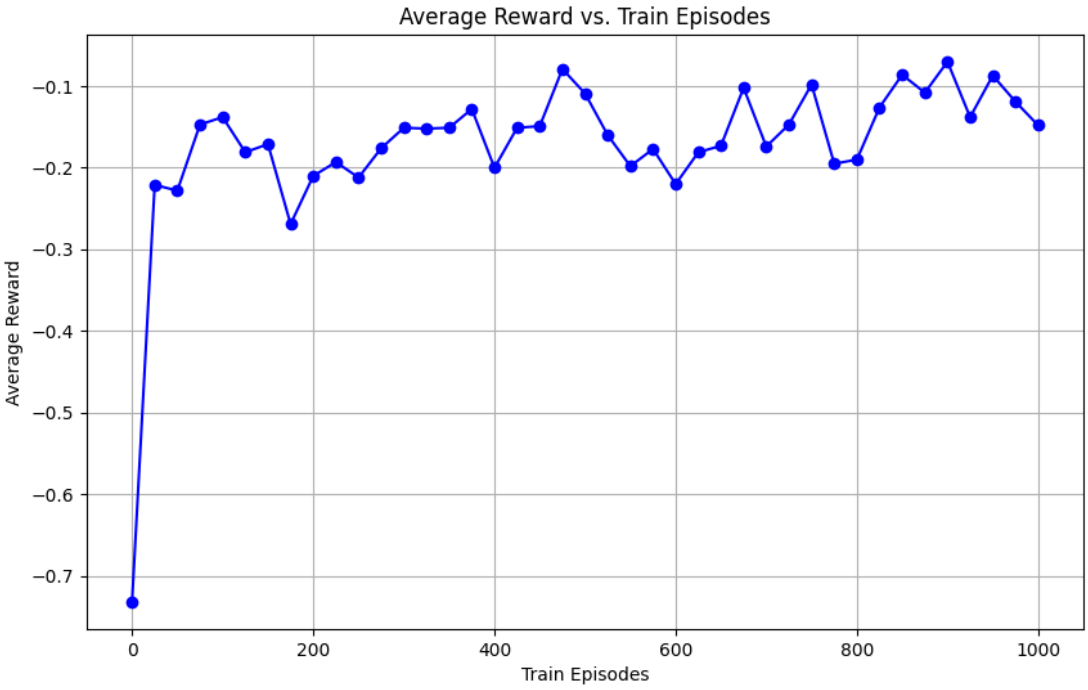
- Configuration 19: the network uses the Leaky ReLU function as activation functions, train_episodes = 500, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



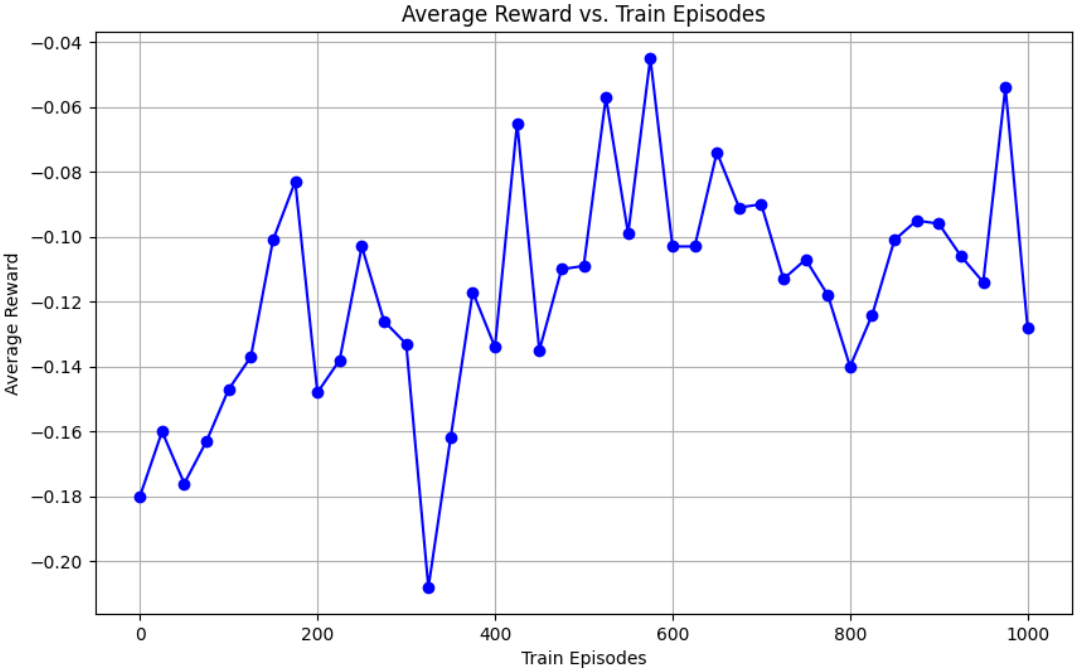
- Configuration 20: the network uses the Leaky ReLU function as activation functions, train_episodes = 500, batch_size=16, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



- Configuration 21: the network uses the Leaky ReLU function as activation functions, train_episodes = 1000, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.995, gamma = 0.99.



- Configuration 22: the network uses the Leaky ReLU function as activation functions, train_episodes = 1000, batch_size=32, deque_maxlen = 100000, learning_rate=0.001, eps_decay = 0.9, gamma = 0.9.



From these graphs, I found that the best configurations are the configurations 2, 16,19 and 22. These configurations achieve results similar to the configurations 3 and 4 used by an agent with a Q-Table. Overall, the results obtained using a DQN are similar to those obtained using a Q-Table.

Conclusions

In conclusion, I implemented a reinforcement learning agent able to solve a non-deterministic environment like Blackjack. The final results obtained from both solutions, the Q-table and the DQN, are similar, though the DQN is a more powerful and sophisticated technique. This behaviour can be attributed to the size of the observation space. The Blackjack environment is relatively small, making the use of a Q-table very efficient. This efficiency allows for an increase in the number of training episodes within a reasonable time frame, leading to effective learning. On the other hand, while the DQN technique is powerful and can achieve similar results with fewer training episodes, adjusting the weights of the network still requires a significant amount of time.