Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

EARIN

Laboratory 6
Group 21
Variant 1
Reinforcement Learning

Tymon Żarski 310992
Bartosz Peczyński 310703

WARSAW May 9, 2024

Contents

1. Introduction

1.1. Assigned task

Create an implementation of the Q-Learning algorithm to solve a toy Reinforcement Learning problem using the Gymnasium library. Use “CartPole-v1”. Notice that the observation space is continuous and requires quantization to put it in the Q-Table.

Please refer to the Gymnasium documentation for each environment to understand the problems, goals, actions, states, rewards, and termination conditions.

1.2. Running the program

To run the program, it is needed first to install all dependencies from *requirements.txt* using *pip3 install -r "requirements.txt"* and afterwards run *main.py* from CLI, using Python 3, for example: *python3 main.py*.

2. Q-Table algorithm

Q-learning is a model-free reinforcement learning algorithm that learns an action-value function to provide the expected utility of taking a given action in a given state and following the optimal policy thereafter. The algorithm updates its estimates based on the equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

where s is the current state, a is the current action, r is the reward received, s' is the new state, α is the learning rate, and γ is the discount factor. A crucial component of Q-learning is the exploration strategy, known as the epsilon-greedy strategy. This strategy selects actions randomly with a probability ϵ and the best-known action with a probability $1-\epsilon$. Initially, ϵ is set high to encourage exploration of the state space, and it is gradually reduced to enhance the exploitation of the learned values as the algorithm converges towards an optimal policy. This balance helps adequately explore the environment while exploiting the best strategies learned so far.

2.1. Space quantization

The CartPole observation space is continuous, so it requires discretization. This process has a few steps.

1. Narrow the size of variables from observation space that have limits from -infinity to infinity, such as the speed of the cart and the angular speed of the pole.
2. Make a linear scale from continuous to discretized space
3. Scale values onto a range of discretized space.
4. Round values to the nearest integer - a value from discretized space.

3. Experiments with different parameters

Due to the quantization limitation, we've added limitations to the highest and lowest values, which are bounded from infinity to minus infinity. These actions resulted in the following results.

```
self.highest_values = [
    self.highest_values[0],
    0.5,
    self.highest_values[2],
    math.radians(50) / 1.0,
]
self.lowest_values = [
    self.lowest_values[0],
    -0.5,
    self.lowest_values[2],
    -math.radians(50) / 1.0,
]
```

3.1. Initial assumptions

Using a trial and error approach by running the algorithm with different parameters, we found the following values that we selected for further investigation:

- Number of episodes: 1200
- Minimal epsilon: 0.0
- Maximal epsilon (start value): 1.0
- Decay rate: 0.005
- Discount factor: 1
- Learning rate: 0.1
- Discretized Space: (3, 3, 6, 6)

3.2. Hyperparameters testing

We tested different values starting from our initial parameters, which we found through trial and error. We changed one parameter at a time and assessed the algorithm's performance.

3.2.1. Epsilon decay

On the Figure 3.1, we have examples of training the system on the base parameters with changed decay rates from (5e-2), (5e-3), to (5e-4), respectively. from the graph, we can see the impact of such change. In the first pair of graphs, We noticed that the exploration face was too short, and then, to the end, the exploration phase kept a previously learned baseline. The second pair of graphs represents a more optimal decay rate for the base parameters. The third pair of graphs shows the situation where the exploration phase is too long, and in the end, we haven't got a chance to allow the model to stabilise and pick the approach.

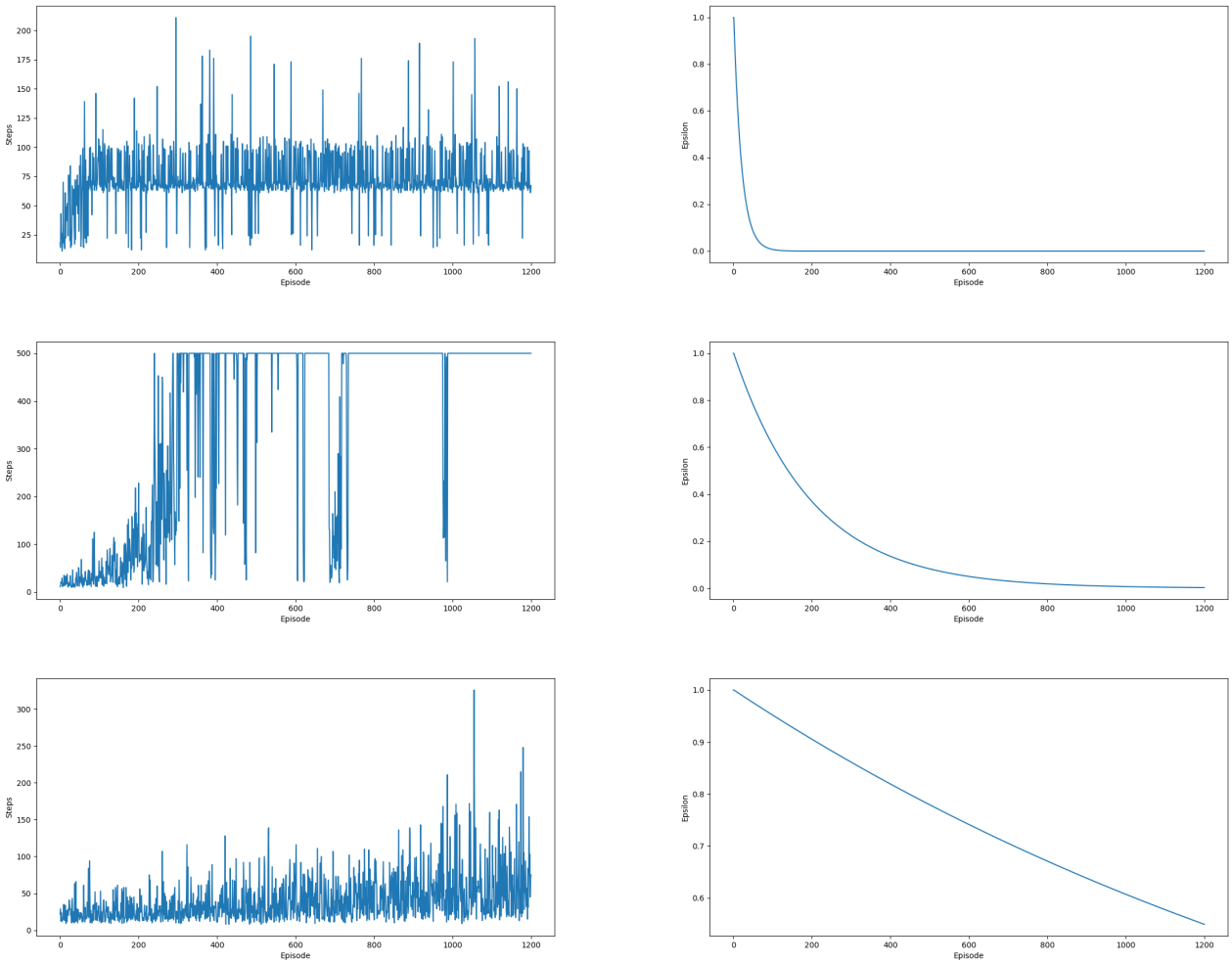

Figure 3.1. Training with different decay rate values

3.2.2. Amount of episodes

The impact of setting the correct number of training episodes is available in Figure 3.2 for 400 and 2000 steps compared to 1200 used before. We can see the difference where, in the first case, there was not enough time to go through all of the planning phases and processes needed prematurely. Conversely, the second parid of plots showed an example where training was unnecessarily long but did not impact the result because of the correct set decay rate. Because of that fact, we introduce the parameter **patience** that checks if a certain amount of episodes the reword in always maximal tills the training process.

3.3. Learning rate

The Figure 3.3 displays the model's behaviour under training with different learning rate values or, in this case, **0.05** and **0.35** in comparison to the *0.1*. This allows us to observe two different situations. The first pair of plots shows the situation where the learning rate is too low, and the differences between the values are less significant because the learning rate is one of the most important factors in calculating the updated QTable state. As a complete difference, the case of the high-value learning rate shows us the example of

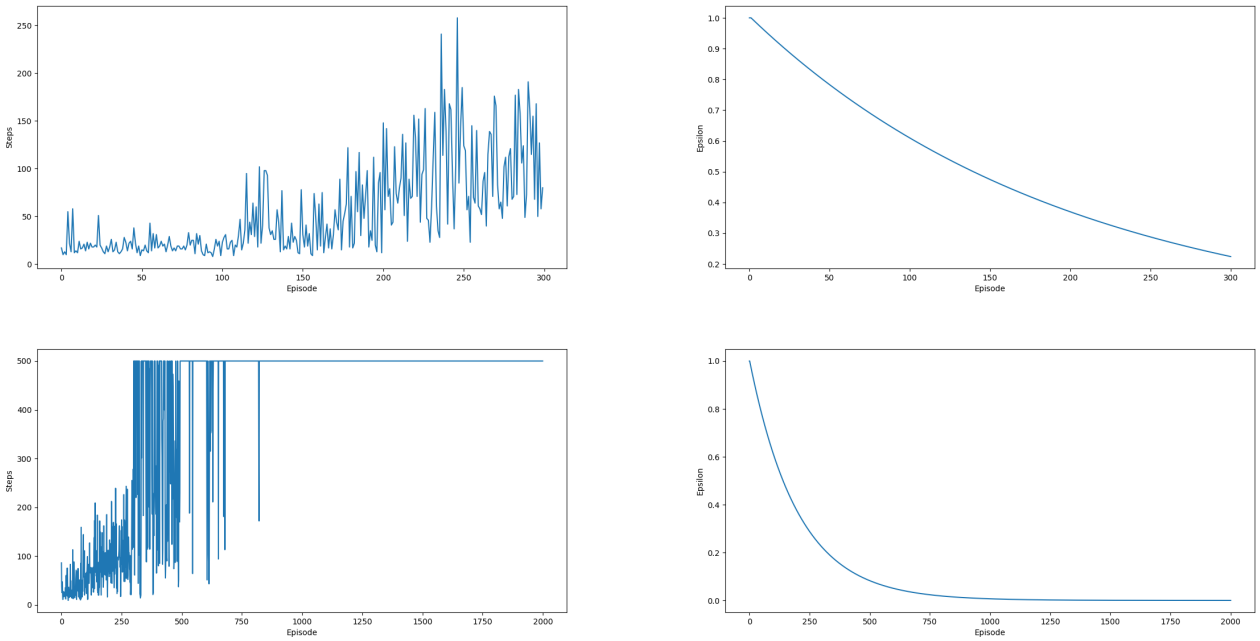

Figure 3.2. Training with different amount of episodes

overfitting because of the too-fast learning rate. It is visible on the plot that, in around 600-700 episodes, the performance is starting to drop.

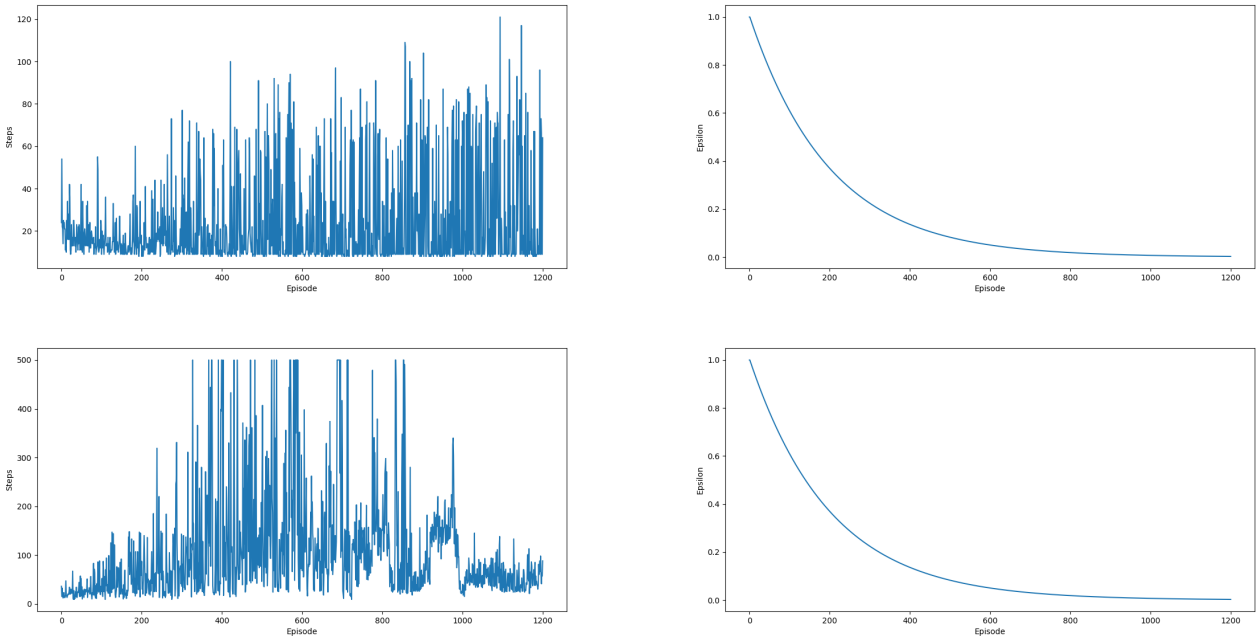

Figure 3.3. Training with different values of learning rate

3.4. Enlarged discrete space

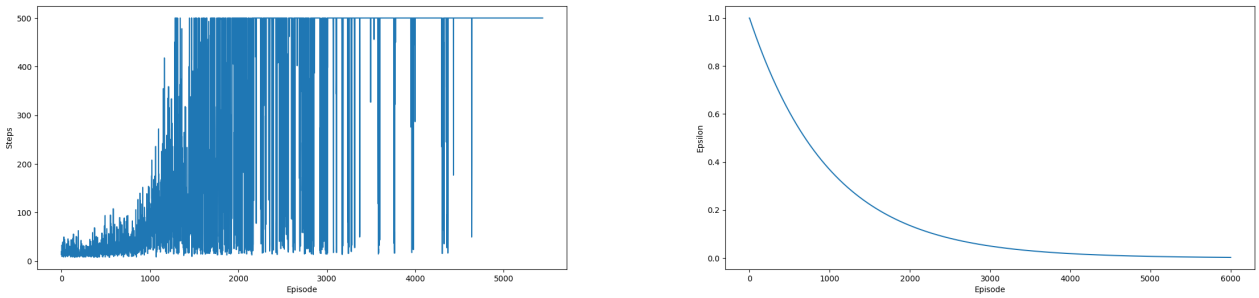

Figure 3.4. Trading with bigger discrete space

We increased the size of the discrete space to (3, 3, 8, 8). This change required modifying other parameters (mark in bold in the list below) to extend the time the algorithm was learning. Although the discretized space was bigger, the final results with a smaller space were the same.

- Patience: 400
- **Number of episodes: 5000,**
- Minimal epsilon: 0,
- Maximal epsilon: 1.0,
- **Decay rate: 0.001,**
- Discount: 1,
- Learning rate: 0.1,
- **Discretized space: (3, 3, 8, 8),**

4. Summary

The results of the Q-learning algorithm hugely depend on the hyperparameters the algorithm uses. Small changes in the parameter may lead to bad results, such as no algorithm convergence. We found that increasing the number of buckets used by the algorithm and, consequently, the size of the QTable must be followed by changing other parameters. This is due to the fact that bigger QTables require more training steps and parameters that allow the algorithm to learn further. We have also seen that the number of buckets for solving the CartPole problem can be quite small, and the algorithm will operate properly.