Bandits & Reinforcement Learning - Advanced Machine Learning

Beltrán Castro Gómez
beltran.castro.gomez@gmail.com

Abstract

This report presents a survey of several reinforcement learning algorithms on two different problems.

Firstly, we will compare the Incremental Uniform, UCB and ϵ-greedy apporaches in the RL classical Multi-Armed Bandits problem.

Secondly, we will see the differences between classical Q-learning and Deep Q-learning on a labyrinth-like game with the help of the Gym (Brockman et al., 2016) library.

1. Multi-Armed Bandits Problem

The multi-armed bandits problem or k-armed bandits problem consists in repeatedly chosing among k different options or actions, that provide a numerical reward chosen from a stationary probability distribution and which objective is to maximize the expected total reward over some time period (Sutton & Barto, 2018).

Our experimental setting involves the 8 armed bandits with uniform and truncated normal (generated with *truncnorm* from scipy) distributions that can be observed in Table 1.

Arm	Distribution	Range
0	Uniform	[8, 10]
1	Uniform	[2, 3]
2	Uniform	[3, 10]
3	Uniform	[5, 7]
4	Gaussian	[7, 10]
5	Gaussian	[3, 5]
6	Gaussian	[0, 1]
7	Gaussian	[5, 6]

Table 1. Armed bandits used during the experiments.

1.1. Bandit Algorithms

The 3 algorithms tested during this experiment are:

1. **Incremental Uniform Algorithm**. This is kind of a *brute-force* algorithm that just explores all diferent options or arms equally without trying to exploit the best one. It pulls each arm once at each iteration.

2. ε**-Greedy Algorithm**. This is a simple approach that selects the best current arm - the one that has yield the greater average reward at that time - with a probability of $1 - \epsilon$ and selects a totally random arm with probability of ϵ.

   It requires a trade-off between exploration and exploitation since the smaller the value of ϵ, the longer it will take to find the best arm and if this value is too large, the algorithm will end up selecting too often a sub-optimal arm.

3. **Upper Confident Bound Algorithm**. UCB alternates between exploration - since there is always uncertainty about the accuracy of our estimates - and exploitation. Contrarily to ϵ-greedy, that choses non-greedy actions randomly at a given rate, this approach explores other actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates (Sutton & Barto, 2018).

   The action *A* at timestep *t* is selected according to the formula in Equation 1:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \quad (1)$$

   The Q_t refers to the estimated value of action a at time t, which corresponds to the average reward and the right part of the equation will stimulate exploration. $N_t(a)$ introduces the amount of times action a has been selected at time t. Thus, when an action has barely been selected, the uncertainty of our estimates will be high and so this term, and it will also grow with time due to the logarithm.

1.2. Algorithm Comparison

All of the algorithms were executed a total of $m = 250$ trials, performing $n = 1000$ pulls at each trial. As we can see in Table 1, the best theoretical armed-bandit involved in the experiment is arm 0 with a mean reward of 9.0.

For the **Incremental Uniform** algorithm, the selected best arm was arm 0 with an average reward of 9.0021 and the general average reward obtained was 5.0076 and the general average regret was 3.9924. In Figure 1, we can observe the average reward and regret for the different arms.

For **UCB**, 3 different tests were performed, corresponding to different values of the c constant: 1, 2 and 10.
In all of the three cases, the arm 0 was selected as the best one with 990.42 average pulls and an average reward of 8.9982 for the first case, 980.02 average pulls and an average reward of 9.0012 for $c = 2$ and, finally, 742.908 average pulls and an average reward of 9.0022 when setting c to 10.
In the best case, with for $c = 1$, the algorithm obtained a total average reward of 8.9606.
As we can observe in Figure 2, the average reward and regret converge to the optimal value slightly quicker for the case of $c = 1$ than with $c = 2$, and for the case of $c = 10$, it is not able to converge.
In Figure 3, we can see that the cumulative reward and regret stabilizes very fast for all cases, but is always far from the optimal value.

For the ϵ**-greedy** algorithm, also 3 different tests were performed, corresponding to different values of ϵ: 0.1, 0.25 and 0.5.
Again, in all of the three cases, the arm 0 was selected as the best one with 908.28 average pulls and an average reward of 8.9988 for the first case, 776.296 average pulls and an average reward of 8.9987 for $\epsilon = 0.25$ and, finally, 559.952 average pulls and an average reward of 9.0020 when setting ϵ to 0.5.
In the best case, with ϵ set to 0.1, the algorithm obtained a total average reward of 8.5862, which is worse than the best UCB case.
In Figure 4, we can observe higher variance in the averages of the reward and regret with respect to the UCB case, due to the implicit randomness of the ϵ-greedy approach. Here, in none of the cases we see an approximation or convergence towards the optimal values of reward and regret and we see a bigger distance between the three curves. Furthermore, in Figure 5, we can see that the cumulative reward and regret are also more far away from each other than in the UCB case, but now they are a lot closer to their respective optimal values than with UCB.

2. Reinforcement Learning

With the objective of studying different Reinforcement Learning approaches, the proposed labyrinth game environment in which our agent has to find a treasure, avoiding the poison and advancing through a NxN room has been implemented with the Gym library (Brockman et al., 2016). In this said environment, the action space is defined by a discrete space of four possible actions regarding the agent's movement:

- **Action 0**. Move up.
- **Action 1**. Move left.
- **Action 2**. Move right.
- **Action 3**. Move down.

With respect to the observation space, the states can be defined in different ways depending on the purpose:

- **Classical Q-learning**. In this case, it is sufficient with representing the state as the position of the agent in the room with an index from 0 to $N^2 - 1$. The positions of the poison, treasure and wardrobe are not that relevant since the room disposition will not change.
- **Deep Q-learning**. When working with neural networks, we will need to encode the agent's position as a one-hot encoded vector of length N^2 representing every cell in the room. It will have a value of 1 in the position where the agent is located and 0 everywhere else.
- **Deep Q-learning with the room changing every two consecutive games**. In this last case, we will have to add new columns for the wardrobe, poison, treasure or empty cell cases. So, the observation space will be defined by a matrix of shape $(N^2, 5)$ that will take values 0 and 1.
  The rows represent again the cells of the room and columns of the matrix represent $[empty\_cell, agent, wardrobe, poison, treasure]$. It is not one-hot encoding, since two rows can have values $[0, 1, 0, 1, 0]$ or $[0, 1, 0, 0, 1]$ that represent that the agent reached either the poison or the treasure (respectively), and are necessary for associating negative and positive rewards to this ending states.

Both classical and deep methods have been tested on three different room sizes: $5x5$, $10x10$ and $20x20$ and with the same room configurations to properly compare them.

2.1. Classical Q-learning

For approximating the optimal state-action values in Classical Q-learning, we update the Q table following the Bellman equation presented in Equation 2.
The new values are obtained by summing the old values

times $1-\alpha$ plus the learned values multiplied by the learning rate. This learned value concerns the obtained reward for the taking the action a_t in state s_t and the discounted estimate of the optimal future value.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)\cdot Q(s_t, a_t) + \\ +\alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \tag{2}$$

Two experiments have been performed on classical Q-learning. Firstly, in a greedy way, i.e. which means no exploration at all. And, secondly, following an epsilon-greedy approach with different epsilon values. In all cases, the training has taken place during 10000 episodes and the the learned Q-tables have been tested during 1000 episodes.

2.1.1. GREEDY Q-LEARNING

For the greedy approach, three alpha values have been tested: 0.01, 0.1 and 0.5. The first one gives a lot of importance to old q values and the last one gives equal importance to old and newly learned values.
The results for mean episode reward, average steps taken per episode and treasure success rate on test time can be visualized in Table 2. The treasure succes rate represents the ratio of treasures that the agent reaches with respect to the number of episodes. It is important to remark that the theoretical maximum value for this ratio is given by the following expression:

$$\max(Tr) = 1 - \frac{1}{N^2-1} = \frac{N^2-2}{N^2-1} \tag{3}$$

since there is a probability of $\frac{1}{N^2-1}$ for the agent to start the episode in the poison cell.

N	ALPHA	EP. REWARD	STEPS	TREASURE RATE
5	0.01	1.9402	3.36	0.956
5	0.1	1.9426	3.401	0.962
5	0.5	1.7736	3.544	0.953
10	0.01	-0.0212	10.531	0.945
10	0.1	0.7219	6.23	0.988
10	0.5	0.6395	6.536	0.988
20	0.01	-0.9613	75.292	0.266
20	0.1	-0.5427	21.993	0.916
20	0.5	-0.3833	15.66	0.933

Table 2. Greedy Q-learning results on test time

For N=5, we can observe the results in Figure 6. In this case, we cannot appreciate the impact of the learning rate since this is a very simple case.
However, for N=10 (see Figure 7) and, specially for N=20 (see Figure 8), we can observe that small alpha values provoke in the agent that the limits of the intervals that describe the episode reward grow and increases the time for them to stabilize.
We can also see that the smaller the learning rate, the more often the agent is completely clueless. These are the cases where the episode reward goes down to -100, and this means that the agent has been moving around for 100 steps (and getting -1 as reward every time) and it gets finally stopped since that is the maximum number of steps per episode set to avoid the agent falling into infinite loops. The greedy agent learns to go directly for the treasure, avoiding the poison and wardrobe blocks.

2.1.2. EPSILON-GREEDY Q-LEARNING

For the epsilon-greedy approach, the learning rate used based on the previous results was 0.5. Furthermore, three epsilon values have been tested: 0.1, 0.25 and 0.5. The epsilon parameter represents the rate or probability with which the agent will chose a random action over a greedy action at training time.
The results for mean episode reward, average steps taken per episode and treasure success rate on test time can be visualized in Table 3. For N=5 and N=20, adding a little randomness slightly improves the best values of mean episode reward obtained in the previous case, with epsilon equal to 0.

N	EPS	EP. REWARD	STEPS	TREASURE RATE
5	0.1	1.9103	3.34	0.951
5	0.25	2.0061	3.353	0.966
5	0.5	1.9943	3.239	0.949
10	0.1	0.6532	6.555	0.993
10	0.25	0.7138	6.306	0.992
10	0.5	0.6773	6.434	0.991
20	0.1	-0.3148	15.699	0.988
20	0.25	-0.3160	15.929	0.995
20	0.5	-0.3253	16.299	1.0

Table 3. Epsilon-Greedy Q-learning results at test time.

If we compare the epsilon-greedy results (see Figure 9 for N=5, Figure 10 for N=10 and Figure 11 for N=20) with the Greedy Q-learning results for the cases with alpha equal to 0.5, we can conclude that the bigger the epsilon value, the more the agent will wander around the room and explore it, and thus the more variations the episode reward will have in training time. In test time, though, the difference cannot be appreciated in the graphics. In contrast with the greedy agent, when we force the agent to take some random moves, not only it will still learn how to reach the treasure and avoid the other blocks, but it will discover new paths and finally find the most efficient ways to it.

2.2. Deep Q-learning

For being able to introduce deep learning for aproximating the Q-table, I initially tried to replicate in the context of our problem the Keras official webpage tutorial (Chapman & Lechner, 2020) that implements with the Keras and Tensorflow the Deep Q Network model presented in 2015, and that obtained very good results playing Atari games.(Mnih et al., 2015).
After not obtaining good results, I then tried with Keras-RL2 (Mcnally, 2021), an implementation of several state of the art deep reinforcement learning algorithms. The results presented in this section were obtained thanks to it.

2.2.1. Deep Q-learning with non-changing room

For this simple case, the DQN model was used. For being able to estimate the q values with a neural network, this model introduces two concepts.
The first one is *experience replay*, which consists on storing previous experiences with the form of $(s_t, a_t, d_t, s_{t+1}, r_{t+1})$ in a memory buffer and using them for training in mini-batches of randomly drawn samples.
The second novelty that DQN introduces is the use of a target network that is updated periodically with the latest weights to prevent the variations of a moving target. Then, the latter this periodical updates are done, the more stable the training will be.

In Figure 12, we can observe the results from a DQN trained with a very simple MLP with *relu* activation functions and using a *Boltzmann* policy, which builds a probability law on q-values and returns an action selected randomly according to this law. This policy has a temperature parameter that helps the agent alternate between picking the action completely randomly and always picking the most optimal action.
For the N=5 and N=10 cases, the agent's performance improved with respect to previous cases, achieving a mean test episode reward of 7.471 for N=5 after only 1525 training iterations, and a mean test episode reward of 4.594 for N=10 after 9573 training iterations. For the case of N=20, it outputted a mean test episode reward of -4.958 after 9284 training episodes. In this last case, though, the episode reward is not that much representative of the agent's performance, since the neural network can easily learn how to reach the treasure, but the final episode reward will most of the times be negative (even when reaching the treasure) due to the dimensions of the board.

2.2.2. Deep Q-learning with changing room

The situation of having the room change at every episode was too challenging for the DQN so I decided to try with another Deep RL model.
In this case, I went for the Dueling DQN algorithm (Wang et al., 2015), which is quite similar to DQN, but it splits the last layer of the network into two, one being the estimate of the state-value function $V(s)$ and the other the estimate of the advantage function $A(s, a)$, and afterwards summing these two to output the estimate of the Q-values.
The value function $V(s)$ represents how much reward state s will produce, and $A(s, a)$ represents how much better one action is compared to the rest.
After being trained in a room with $N = 5$ for 500000 timesteps, we observed that the agent is able to reach the treasure and avoid the poison in a general way but, sometimes, and depending on the room disposition and the starting point for the agent, it may get stuck trying to advance through the wardrobe or any of the room walls.
This is the reason why it obtained a mean episode reward of 0.0117 when testing during 10000 episodes, since the reward of this episodes where the agent gets stuck goes down to -100.
I only tested this agent for $N = 5$, considering the difficulty of the task and the fact that training it for this small room was already quite costly.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chapman, J. and Lechner, M. Deep q-learning for atari breakout, 2020. URL `https://keras.io/examples/rl/deep_q_network_breakout/`.

Mcnally, T. Keras rl2, 2021. URL `https://github.com/taylormcnally/keras-rl2`.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. Dueling network architectures for deep reinforcement learning, 2015. URL `https://arxiv.org/abs/1511.06581`.

A. Figures

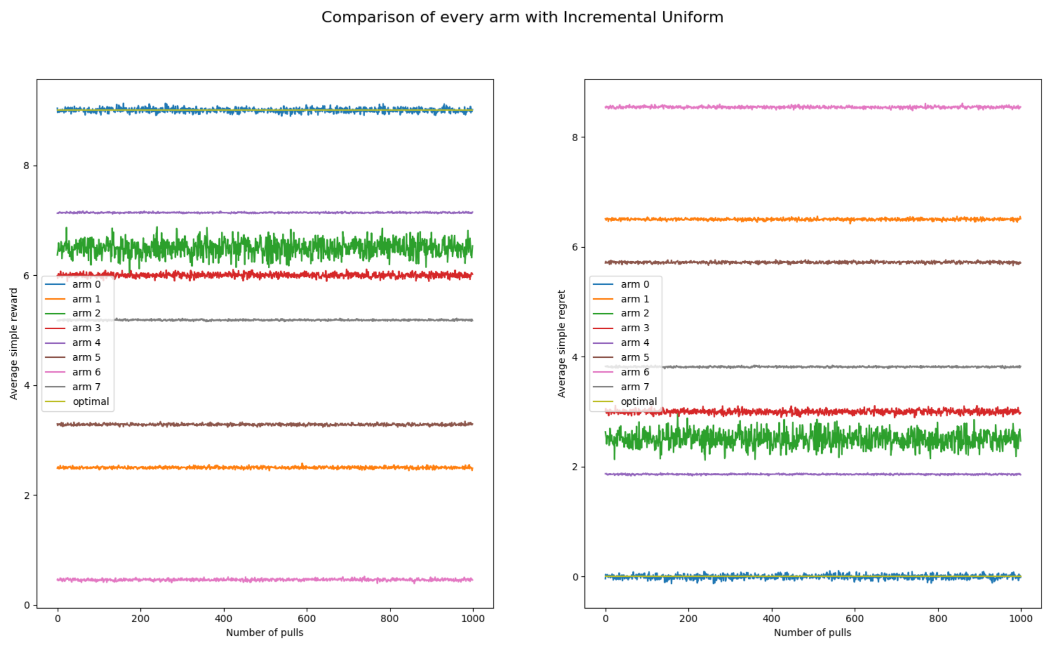

Figure 1. Average reward and regret of every arm with Incremental Uniform.

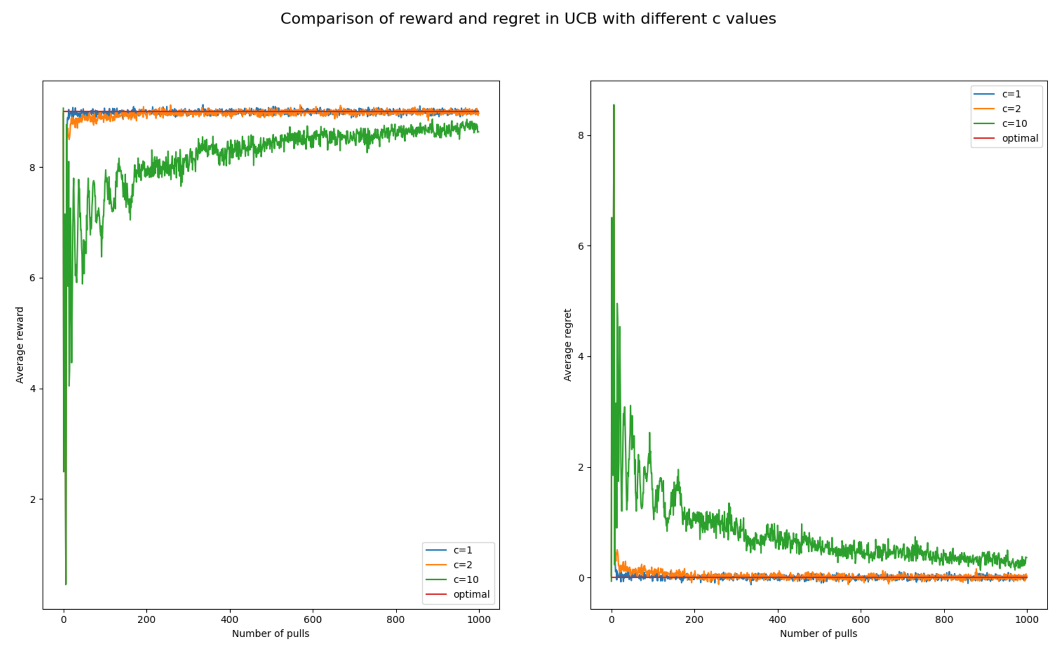

Figure 2. Comparison of average reward and regret of UCB for different c values.

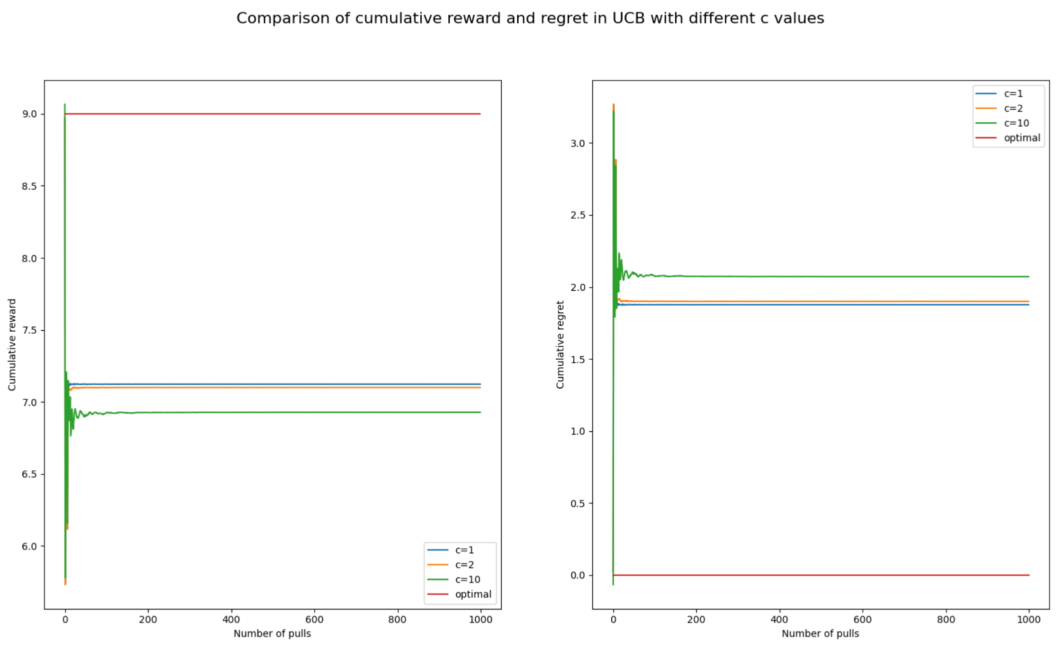

Figure 3. Comparison of average cumulative reward and regret of UCB for different c values.

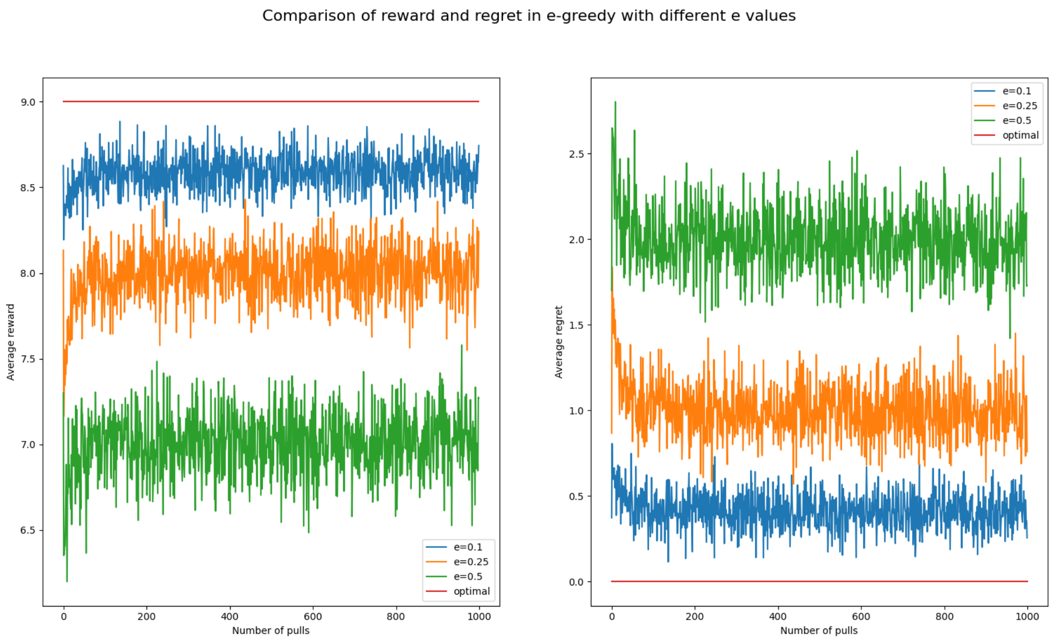

Figure 4. Comparison of average reward and regret of ϵ-greedy for different ϵ values.

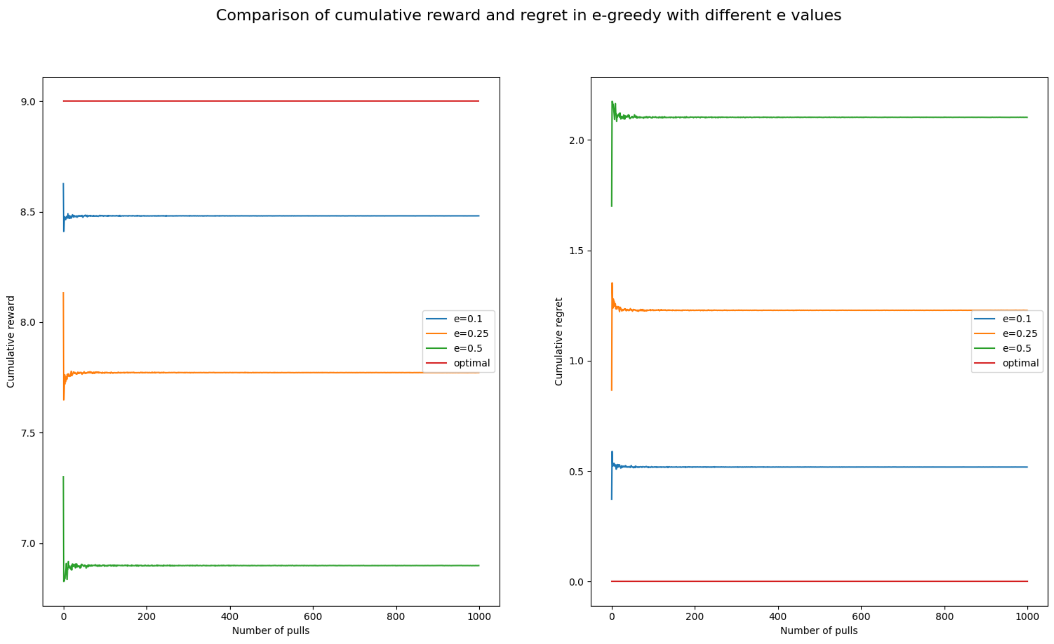

Figure 5. Comparison of average cumulative reward and regret of ϵ-greedy for different ϵ values.

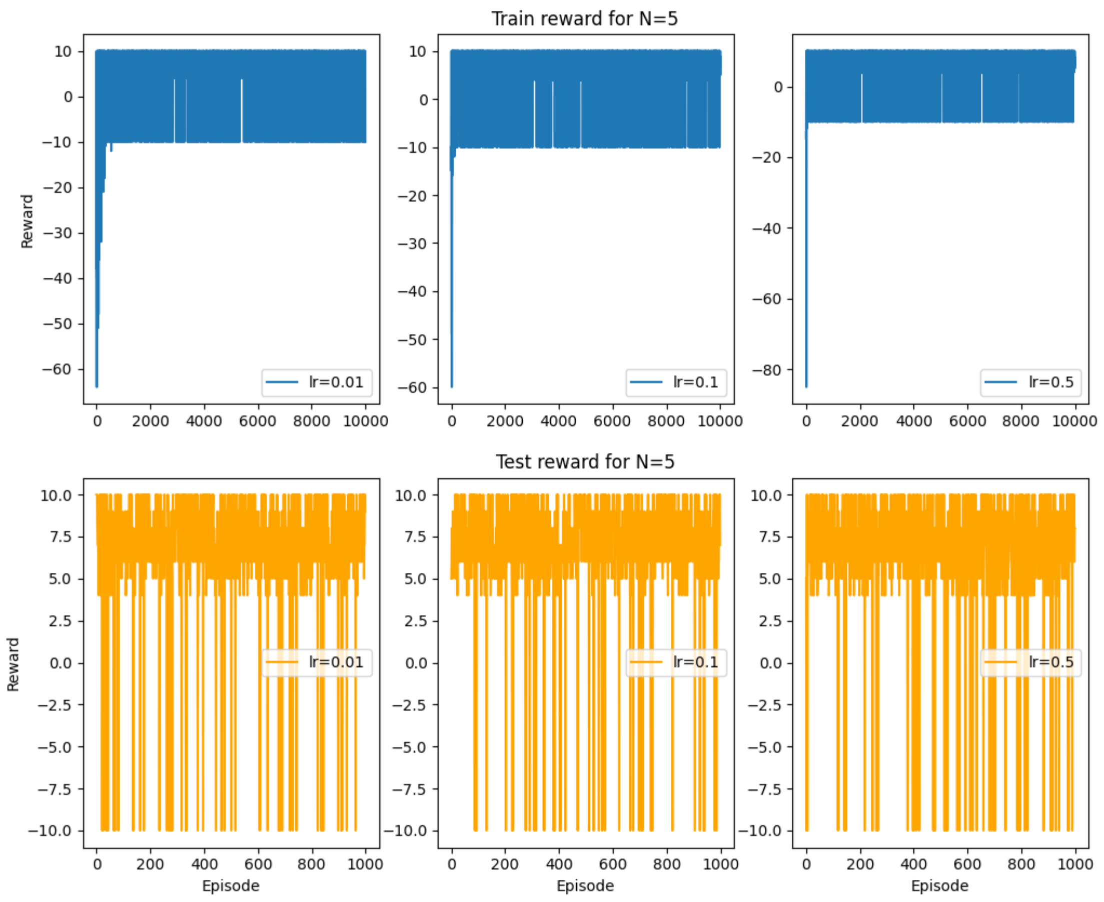

Figure 6. Evolution of episode reward in Greedy Q-learning for N=5

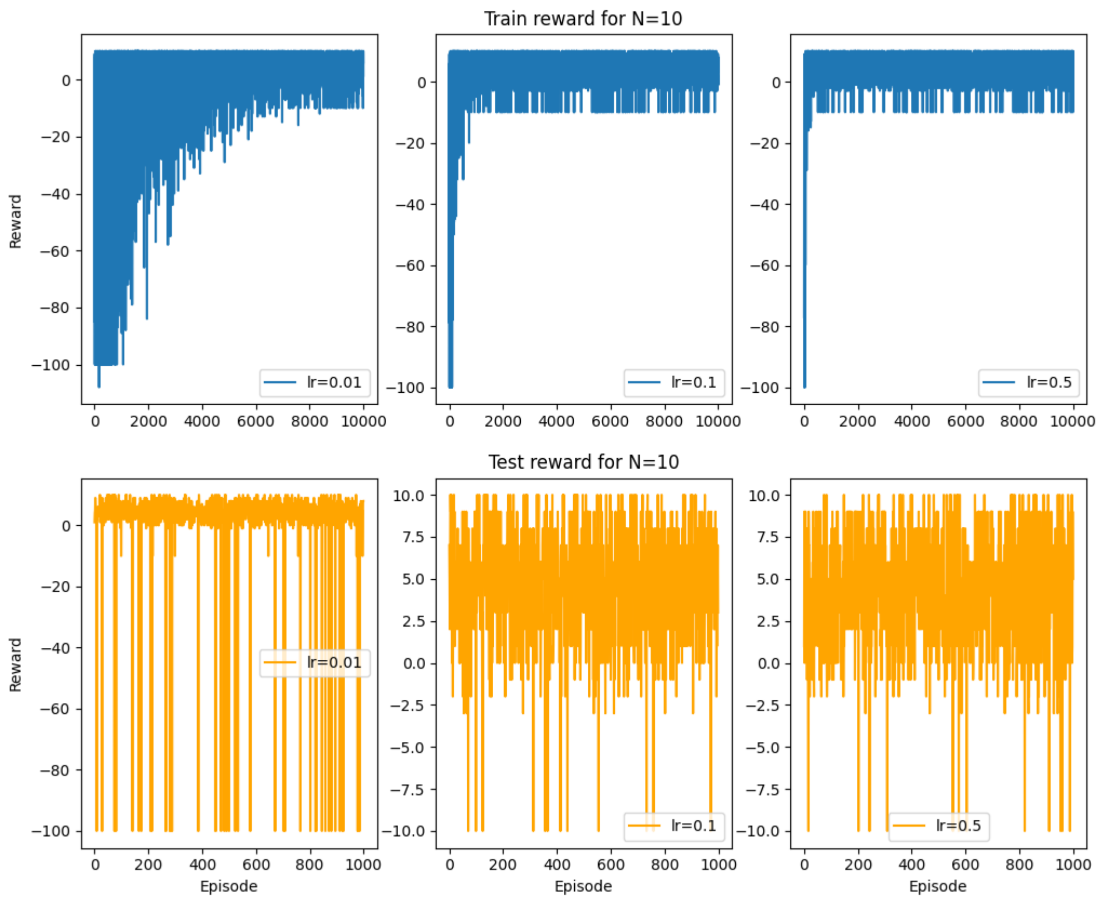

Figure 7. Evolution of episode reward in Greedy Q-learning for N=10

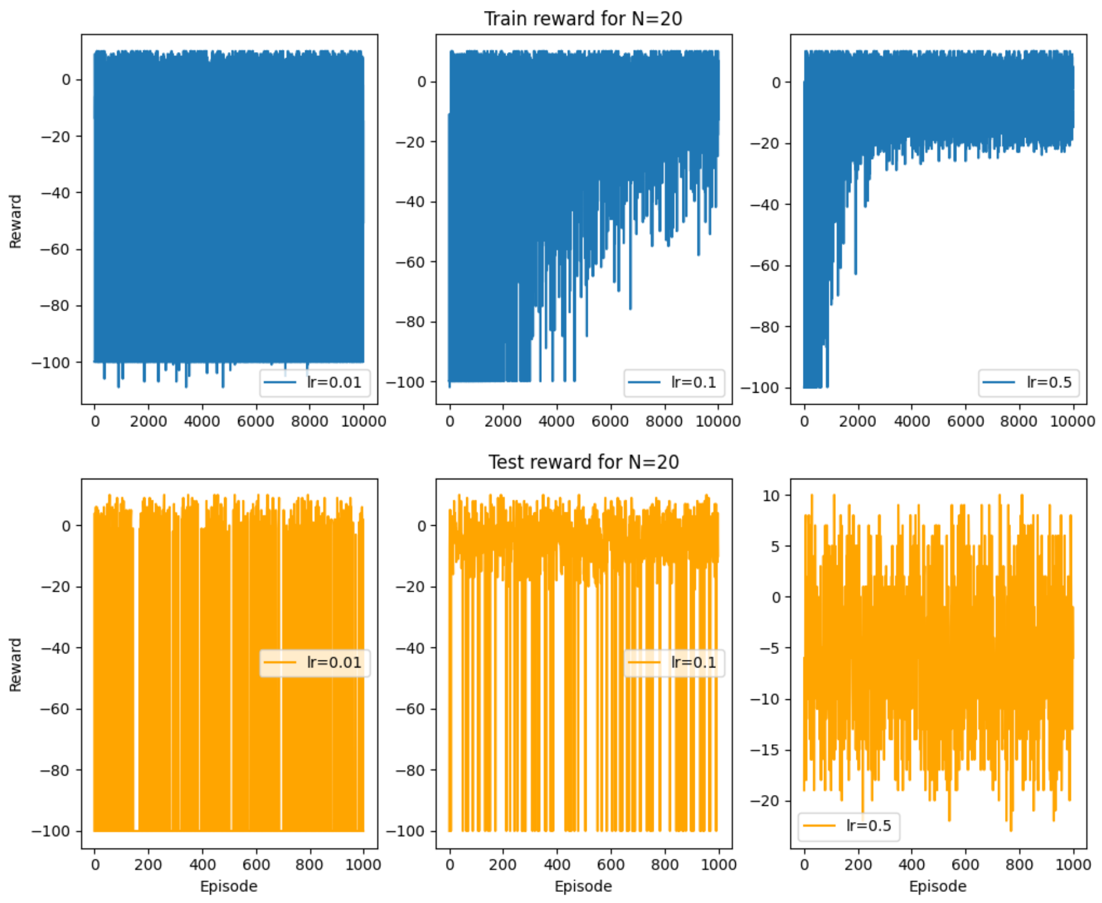

Figure 8. Evolution of episode reward in Greedy Q-learning for N=20

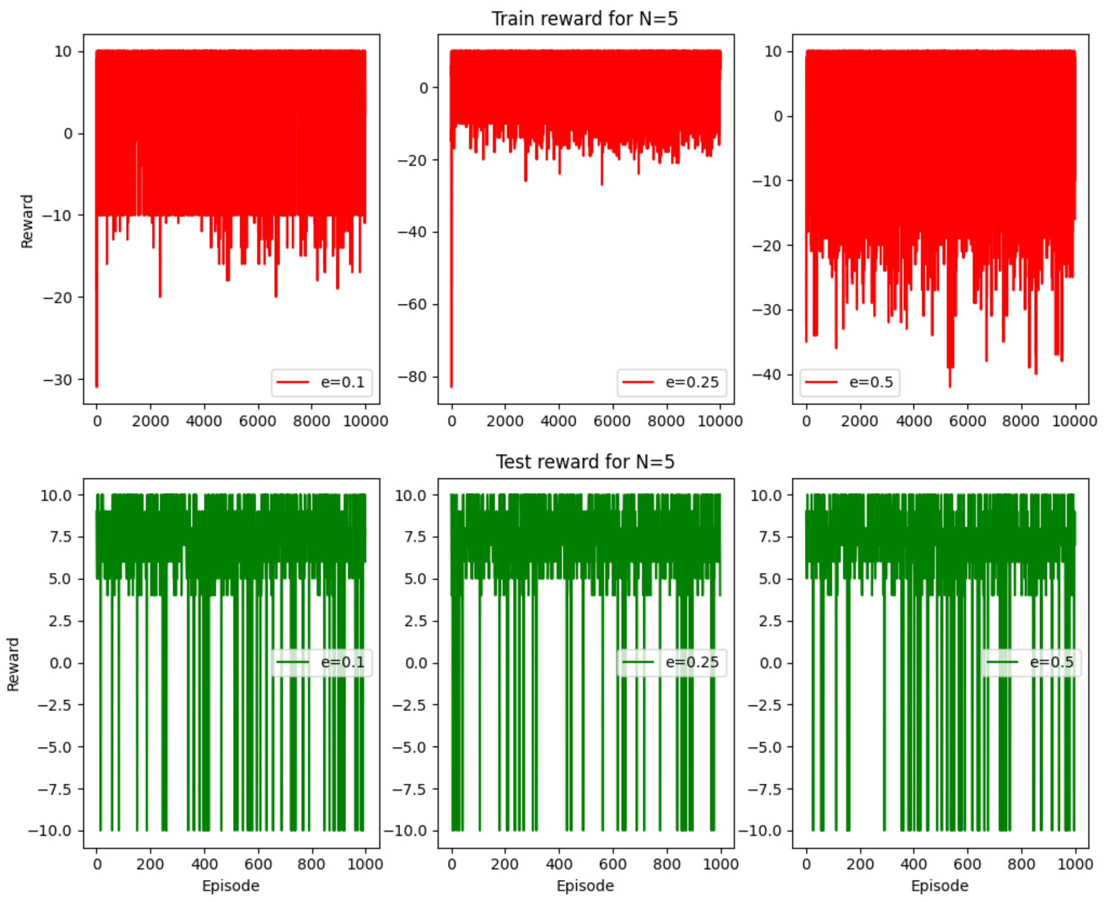

Figure 9. Evolution of episode reward in Epsilon-Greedy Q-learning for N=5

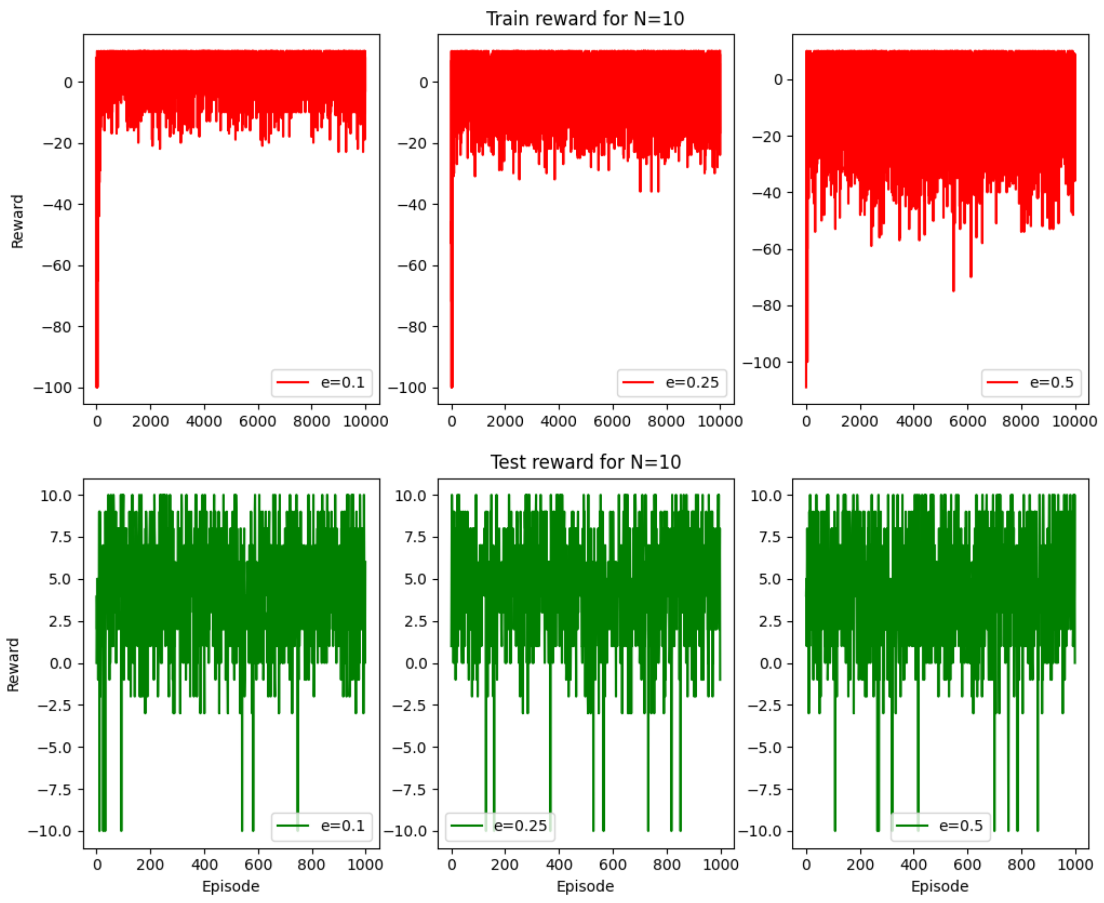

Figure 10. Evolution of episode reward in Epsilon-Greedy Q-learning for N=10

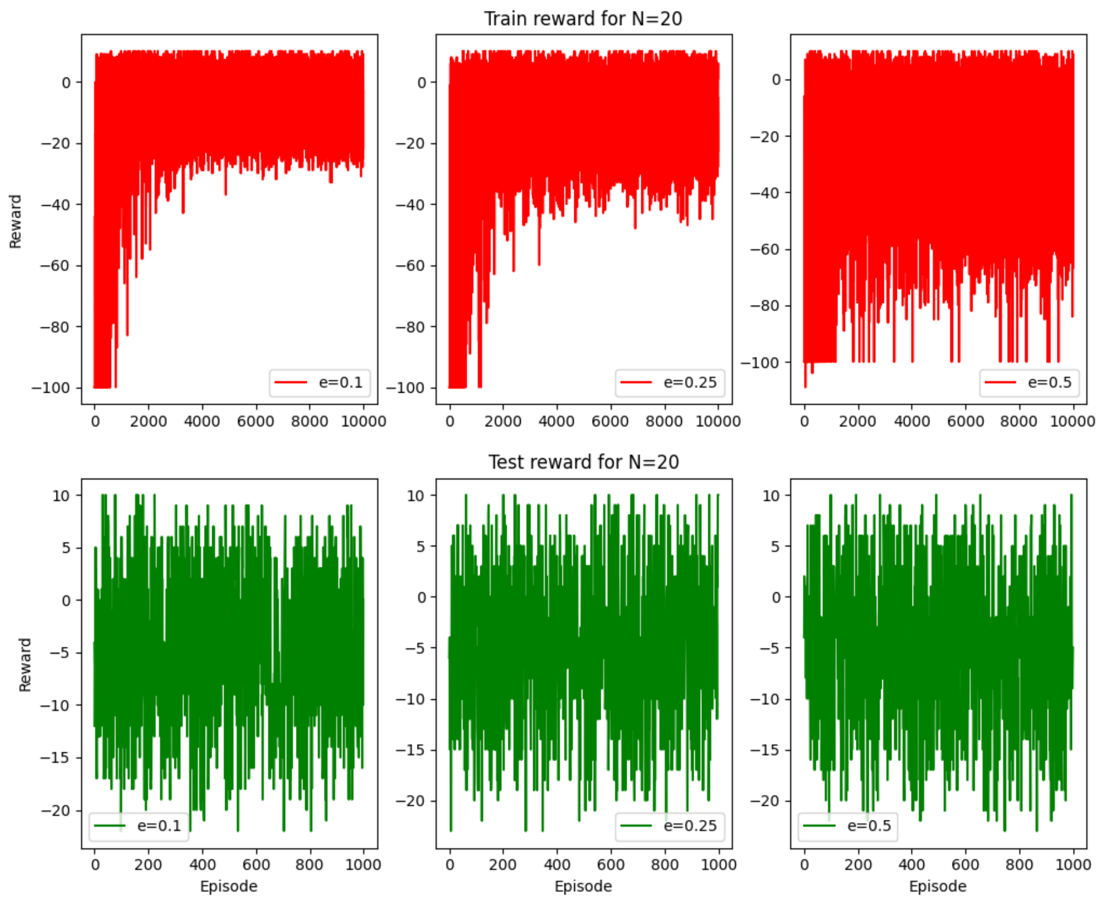

Figure 11. Evolution of episode reward in Epsilon-Greedy Q-learning for N=20

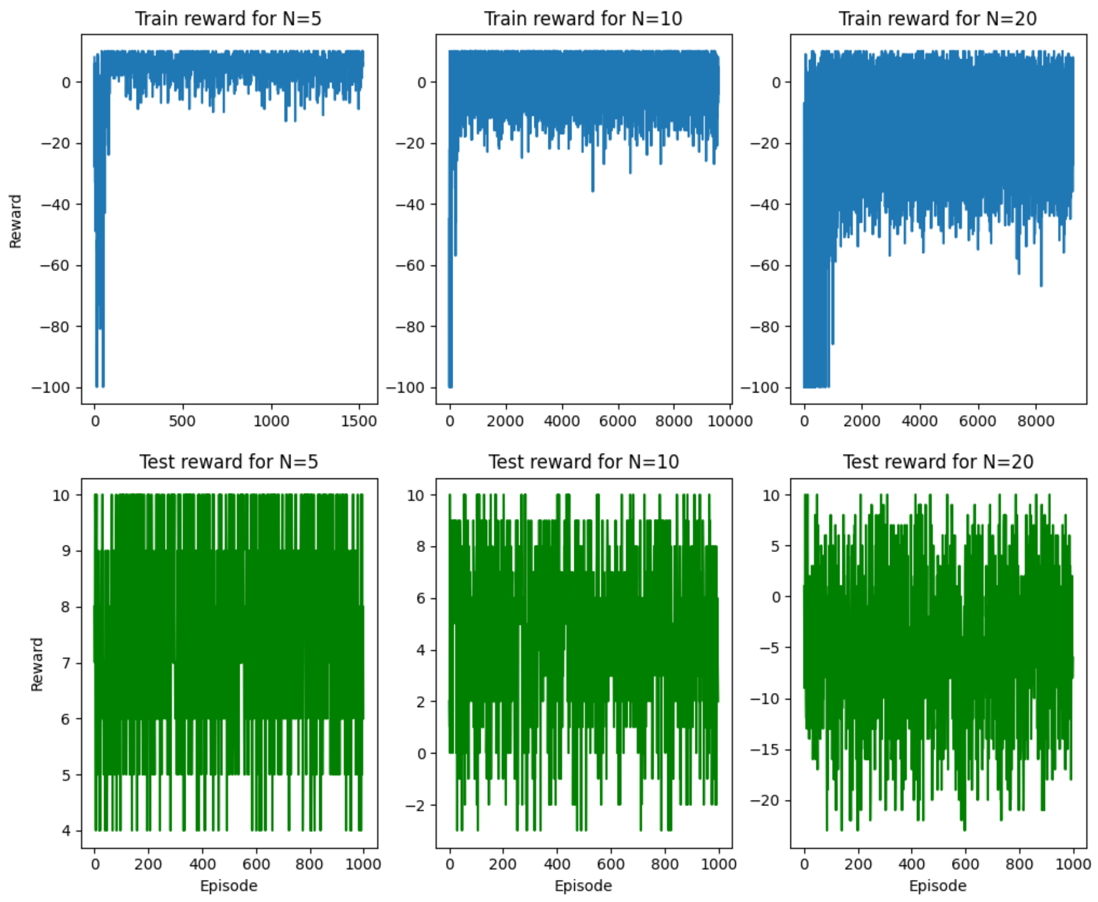

Figure 12. Evolution of episode reward in Epsilon-Greedy Deep Q-learning with a DQN.