MARIA KOILALOU *JUSTINE LE BOURG*

Artificial Intelligence

Laboratory 6

Varient 1

Reinforcement learning is a machine learning technique that allows an agent to learn decision-making by interacting with an environment. In this example, we use the Q-Learning algorithm to train an agent to play the CartPole-v1 game.

Firstly, we import the necessary libraries, including Gymnasium, NumPy, and Matplotlib. Gymnasium is a popular library for reinforcement learning environments. We also define a function to discretize the continuous state of the environment.

We initialize several learning parameters, such as the learning rate (alpha), discount factor (gamma), initial exploration rate (epsilon), minimum exploration rate (epsilon_min), and exploration decay rate (epsilon_decay). We also specify the number of training episodes (n_episodes) and the maximum number of steps per episode (max_steps).

We create an array to store the rewards per episode. Additionally, we initialize a Q-matrix of size (24, 24, 48, 48, n_actions) to store the Q-values for each state-action pair. The dimensions of the Q-matrix correspond to the indices of the discretized state and the number of possible actions.

Next, we enter a loop to run the training episodes. In each episode, we reset the environment and discretize the initial state. We also initialize the total reward for the episode to zero.

At each step within an episode, the agent chooses an action using an epsilon-greedy policy. With a probability of epsilon, the agent takes a random action to explore the environment. Otherwise, it chooses the action with the highest Q-value for the current state. Then, the agent performs this action in the environment and observes the next state, reward, and termination/truncation information. The next state is discretized, and the Q-values are updated using the Q-Learning formula.

The Q-value update utilizes the Q-Learning formula: Q(s, a) = Q(s, a) + alpha * (reward + gamma * max(Q(s', a')) - Q(s, a)). Here, s is the current state, a is the chosen action, s' is the next state, a' is the chosen action in the next state, alpha is the learning rate, and gamma is the discount factor.

After each step, we update the current state, total episode reward, and check if the episode is terminated or truncated. If so, we exit the step loop within the episode. Otherwise, we continue until the episode reaches the maximum number of steps.

After each episode, we store the total episode reward in an array for further analysis.

We display the episode number, number of steps in the episode, and total episode reward. We also display the maximum Q-value and the sum of Q-values in the Q-matrix.

After each episode, we update the exploration rate epsilon by decaying it by a factor of epsilon_decay. This allows the agent to gradually reduce exploration and exploit learned knowledge more.

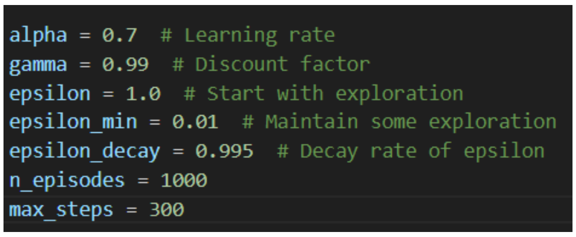
Once all episodes are completed, we plot the reward curve per episode using the Matplotlib library. This allows us to visualize the learning performance of the agent over time.

Finally, we test the trained agent by running the environment with optimal actions based on the learned Q-values. The agent chooses the action with the maximum Q-value for each state until the episode ends.

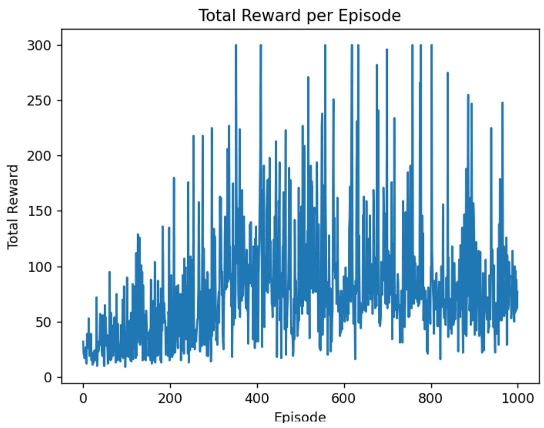
In this example, we used the Q-Learning algorithm to train an agent to play the CartPole-v1 game. The agent learned to make decisions based on the rewards obtained in the environment. By adjusting the learning parameters and using an epsilon-greedy policy, the agent achieved good performance. Reinforcement learning provides a powerful framework for training agents to solve a wide variety of problems.

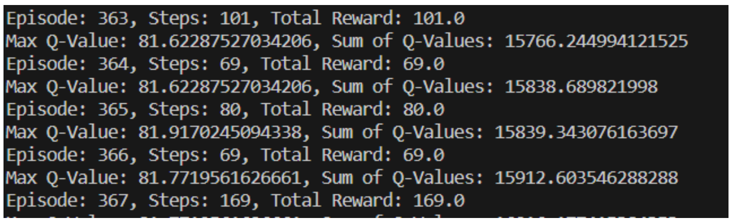
OUTPUT

Running the code with these hyperparameters, the outputs print the results for all 1000 episodes at each iteration, along with the corresponding graph.



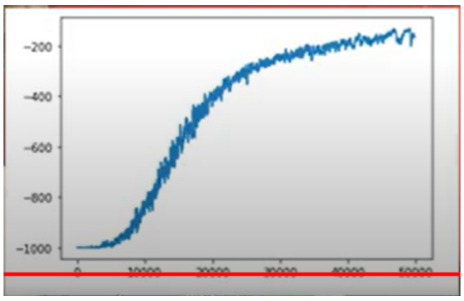
```
alpha = 0.7  # Learning rate
gamma = 0.99  # Discount factor
epsilon = 1.0  # Start with exploration
epsilon_min = 0.01  # Maintain some exploration
epsilon_decay = 0.995  # Decay rate of epsilon
n_episodes = 1000
max_steps = 300
```



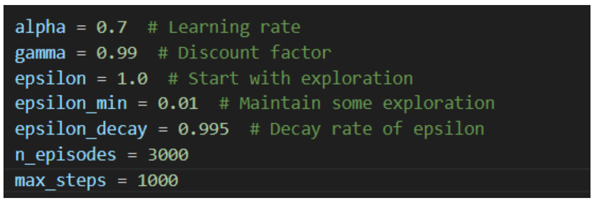


```
Episode: 363, Steps: 101, Total Reward: 101.0
Max Q-Value: 81.62287527034206, Sum of Q-Values: 15766.244994121525
Episode: 364, Steps: 69, Total Reward: 69.0
Max Q-Value: 81.62287527034206, Sum of Q-Values: 15838.689821998
Episode: 365, Steps: 80, Total Reward: 80.0
Max Q-Value: 81.9170245094338, Sum of Q-Values: 15839.343076163697
Episode: 366, Steps: 69, Total Reward: 69.0
Max Q-Value: 81.7719561626661, Sum of Q-Values: 15912.603546288288
Episode: 367, Steps: 169, Total Reward: 169.0
```

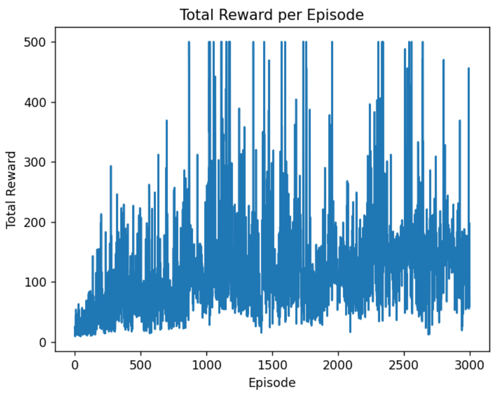
First of all, the graph represents the total rewards by episode, which means it shows the agent's performance over the course of the exercise. Unfortunately, this curve does not meet our expectations, we can see that it grows but not as wanted. We hope to approach a curve of this shape, which indicates improvement of the agent's performance in the exercise.



One theory that could easily explain this is a lack of episodes to train on. That's why we trained our algorithm with these hyperparameters:

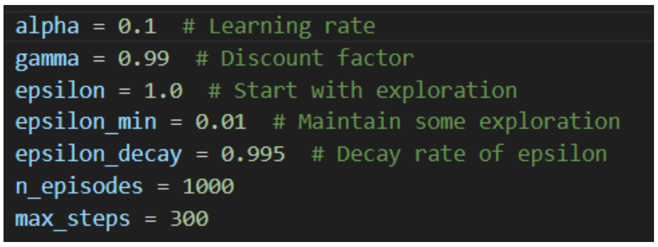


```
alpha = 0.7  # Learning rate
gamma = 0.99  # Discount factor
epsilon = 1.0  # Start with exploration
epsilon_min = 0.01  # Maintain some exploration
epsilon_decay = 0.995  # Decay rate of epsilon
n_episodes = 3000
max_steps = 1000
```

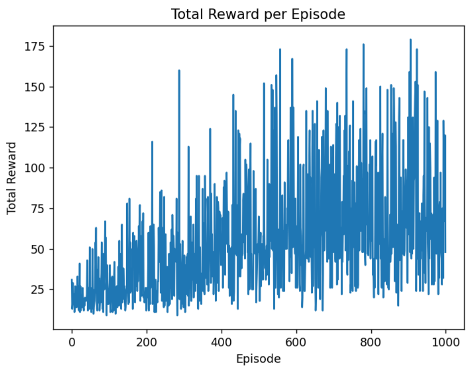


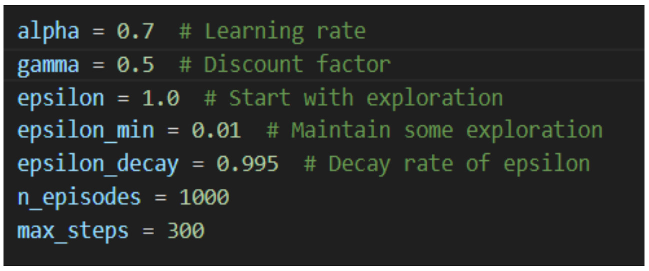
Unfortunately, we did not achieve the expected result.

So we tried changing the other hyperparameters:

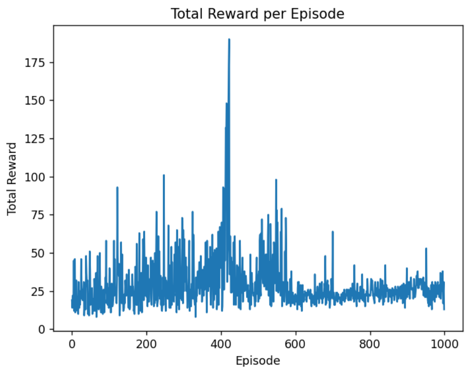


```python
alpha = 0.1  # Learning rate
gamma = 0.99  # Discount factor
epsilon = 1.0  # Start with exploration
epsilon_min = 0.01  # Maintain some exploration
epsilon_decay = 0.995  # Decay rate of epsilon
n_episodes = 1000
max_steps = 300
```





```python
alpha = 0.7  # Learning rate
gamma = 0.5  # Discount factor
epsilon = 1.0  # Start with exploration
epsilon_min = 0.01  # Maintain some exploration
epsilon_decay = 0.995  # Decay rate of epsilon
n_episodes = 1000
max_steps = 300
```



As we can see, the graphs are slightly different but not closer to the result expected. We suspect that there may be an implementation issue that we have been unable to identify.

In conclusion, despite our efforts to train our reinforcement learning algorithm on the CartPole environment, we did not achieve the expected results. The rewards per episode curve did not show significant improvement over time.
It is important to note that reinforcement learning is a complex field, and achieving satisfactory results may require careful adjustments of hyperparameters, algorithm architectures, and exploration/exploitation methods.

Ultimately, this experience has provided us with a better understanding of the challenges in reinforcement learning and the importance of an iterative and experimental approach. We plan to explore other algorithms and techniques to improve the performance of our agent in similar environments in the future.