Reinforcement Learning

Done by:

Tarek Abou Chakra: 5450275

Submitted for:

Deep Learning Lab

Exercise 1

Abstract:

This report aims to provide a comprehensive overview of reinforcement learning using DQN. Mainly focusing on the application of reinforcement learning using neural networks and the Python programming language. Additionally, the report will examine how various parameters can affect the final result of these networks.

1- Introduction:

Reinforcement learning has gained significant attention in recent years due to its ability to learn complex decision-making tasks in an autonomous manner. One of the most popular and effective reinforcement learning algorithms is Deep Q-Networks (DQN), which has been successfully applied to various environments, including games, robotics, and control tasks. In this report, we will discuss our experience in applying the DQN algorithm to two different environments: The Cartpole and CarRacing tasks. Through our experimentation, we have gained valuable insights into the strengths and limitations of DQN and its applicability to different scenarios.

2- Cartpole Hyperparameters:

In this section, I will focus on the Cartpole project, which involves maintaining the balance of a stick on a cart by moving it left or right. The Cartpole problem is relatively easy to compute because it only involves four states, and we do not require convolutional neural networks (CNNs) to solve it. Therefore, this project allows us to achieve good results in a short amount of time by even being greedier in the environment. My hyperparameter setting started with the following:

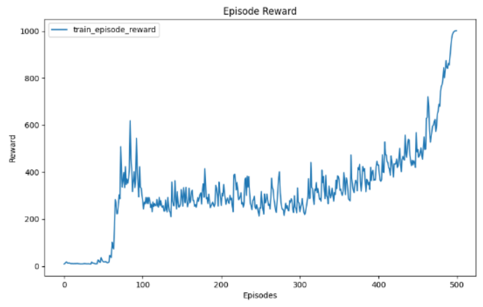
1- First, I focused on the exploration vs. exploitation tradeoff, controlled by the epsilon hyperparameter. I set the initial epsilon value high to increase exploration (set to 0.95), allowing the model to search for a solution that might lead to better results in the long term. I then gradually decayed epsilon by a factor of 0.99 until it reached a minimum of 0.001, where the model most likely only used greedy actions.

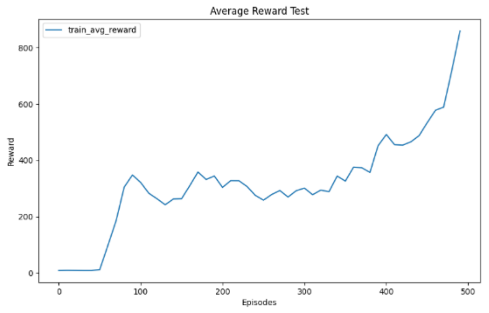
2- The capacity of the replay buffer is another important hyperparameter in the DQN algorithm. Since the Cartpole problem is relatively simple, a replay buffer capacity of 3000 is sufficient to provide most representations with good actions.

3- Finally, I found that training the model for 500 episodes was enough to achieve optimal performance. During the training, the model reached a maximum reward of mean=1002 and standard deviation=0, which is the maximum number of timesteps given for testing.

3- Cartpole Graphs and Results:

Figures of the training and accuracy:





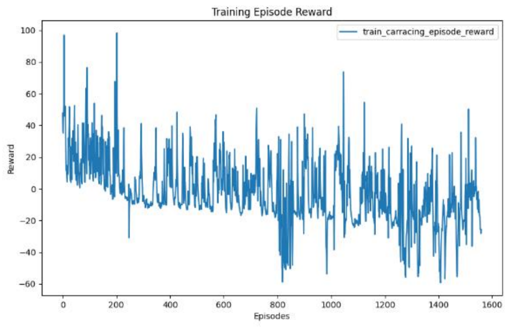
As shown in the graphs, once the epsilon becomes small enough for the model to always search for greedy action. After that, it is only a matter of fine-tuning the action-state relationship. Overall, the DQN algorithm was able to achieve good results on the CartPole environment with a relatively small amount of training.

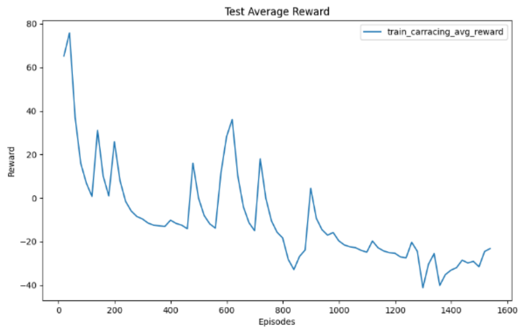
4- Carracing Hyperparameters:

I will discuss the hyperparameters used in the car racing project, which is a more demanding environment compared to the cartpole project. The car racing environment provides states in the form of images, making it more challenging to compute due to the vast number of states obtained. Additionally, the car racing project has five different actions that can be given out, which require the use of a convolutional neural network (CNN) for this case.

1- Epsilon: For the car racing project, epsilon was set slightly larger at 0.99. The slower decay rate allowed for more exploration, which is highly required in this environment (0.999). The minimum decay reached was 0.1, which ensured that the model is not completely greedy because it can always discover better solutions.
2- Capacity: Car racing requires a more detailed understanding of the environment, and a larger capacity replay buffer allows the agent to learn from a more diverse set of experiences. A larger replay buffer leads to better performance and a more robust agent that can handle a wider range of situations. Additionally, with a larger replay buffer, the agent can store more experiences and reduce the risk of forgetting valuable past experiences. Therefore, it was set to 25000.
3- Episode number: Since the car racing environment is large, the episodes to study are also large. Hence, more than 1500 episodes were tried to train this model.
4- CNN Architecture: In contrast to the imitation learning project, a reduced number of parameters CNN architecture was used for the car racing DQN project, considering the large size of the environment. A smaller architecture allows for faster training times.
5- Max time steps adjustment: I tried to start the timesteps at 150 and gradually increase it after a number of episodes to reach 1000 so that the model would learn consecutively the best action.

5- Carracing Graphs and Results:





Several reasons can lead to the result seen on the graphs above:

1- Catastrophic forgetting which is a phenomenon that can occur in machine learning, including in DQN agents, where the agent's ability to perform well on previously learned tasks deteriorates when it is trained on new tasks.

2- Not enough exploration where in this case epsilon should be increased or remain high for longer in order for the model to explore more actions for different states.
3- Adjusting the CNN architecture more, where shallow architectures will not be able to learn and deep architectures will take longer time to train and might tend to overfit.
4- Poor choice of hyperparameters: DQN relies on several hyperparameters, including the learning rate, discount factor, batch size, and replay buffer size. Poor choices of hyperparameters can lead to suboptimal performance.

In conclusion, the initial strategy of setting a low maximum number of timesteps per episode did work effectively for this particular environment. However, the agent did not remember the perfect action in the long term and had a disappointing performance in the end.