SAPIENZA
UNIVERSITÀ DI ROMA

Master in Engineering in Computer Science

A.A. 2022/2023

Artificial intelligence and machine learning

Mountain Car

Russo Lorenzo 2091186

Introduction

This project is based on reinforced learning. Reinforcement learning agent learns by interacting with the environment through actions, observes next state and uses observed reward for each action as feedback signal to improve policy.

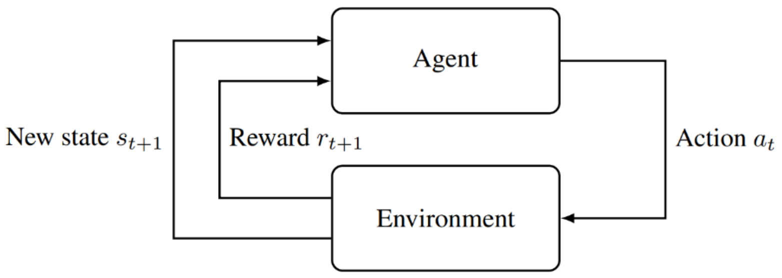
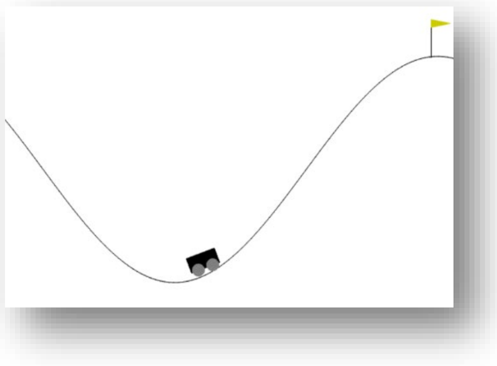


Figure 1

In this project has been implemented a Reinforcement Learning Agent based on Q-learning, which uses NN to approximate the Q-function.

In particular, in this project a DQN (Deep Q-Network) algorithm has been implemented to train an agent to reach the goal in the Mountain Car gym environment. The training performances were analyzed using the graphs of the reward and the maximum position assumed by the car. Finally, a test phase was carried out to analyze the performance of the trained model obtained in the previous training phase; the trained model was also evaluated against a random agent. Furthermore, the mountain car classic environment was modified in a new variant, in this way it was possible to evaluate the behavior and performance of the trained model (that of the classic environment).

1. Mountain Car



The Mountain Car MDP is a deterministic MDP that consists of a car placed stochastically at the bottom of a sinusoidal valley, with the only possible actions being the accelerations that can be applied to the car in either direction. The goal of the MDP is to strategically accelerate the car to reach the goal state on top of the right hill.

Observation Space

The observation is a ndarray with shape (2,) where the elements correspond to the following:

- 0: position of the car along the x-axis
- 1: velocity of the car

Action Space

There are 3 discrete deterministic actions:

- 0: Accelerate to the left
- 1: Don't accelerate
- 2: Accelerate to the right

Given an action, the mountain car follows the following transition dynamics:

$$velocity_{t+1} = velocity_t + (action - 1) \cdot force - \cos(3 \cdot position_t) \cdot gravity$$

$$position_{t+1} = position_t + velocity_{t+1}$$

where force = 0.001 and gravity = 0.0025. The collisions at either end are inelastic with the velocity set to 0 upon collision with the wall. The position is clipped to the range [-1.2, 0.6] and velocity is clipped to the range [-0.07, 0.07].

Reward:

The goal is to reach the flag placed on top of the right hill as quickly as possible, as such the agent is penalised with a reward of -1 for each timestep.

Starting State

The position of the car is assigned a uniform random value in [-0.6 , -0.4]. The starting velocity of the car is always assigned to 0.

Episode End

The episode ends if either of the following happens:

- Termination: The position of the car is greater than or equal to 0.5 (the goal position on top of the right hill)
- Truncation: The length of the episode is 200.

1.1 Problem solution

The DQN architecture used to train the agent is composed of two neural nets, the Q network and the Target networks, and a component called Experience Replay. The Q network is the agent that is trained to produce the Optimal State-Action value. Experience Replay interacts with the environment to generate data to train the Q Network. Experience Replay interacts with the environment to generate data to train the Q Network.

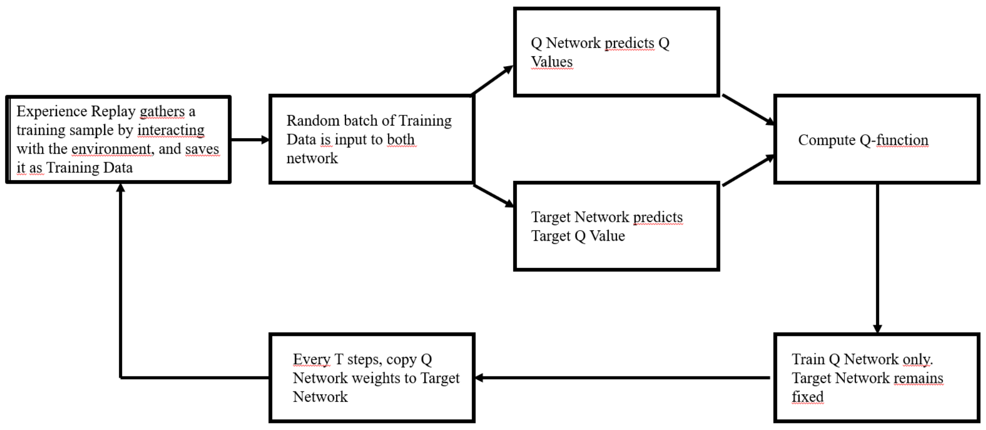

Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data
Random batch of Training Data is input to both network
Q Network predicts Q Values
Target Network predicts Target Q Value
Compute Q-function
Train Q Network only. Target Network remains fixed
Every T steps, copy Q Network weights to Target Network

Figure 1.1

The figure 1.1 summarizes the operations that are performed by the algorithm for compute the Q-function starting from a data structure that stores past experience, then a random batch is extracted (according to the set batch dimension) and put into input to the two networks Q-network and Target network to predict. As output from the Q-network we get the Q-values (Q(s, a)) and from the target network we get the target Q-values (Q(s', a')), i.e. the Q-values of the next state s' . Once this is achieved, the training rule to learn Q is applied:

$$Q(s,a) = r + \gamma \; max_{a'} Q(s', a')$$

In particular:

- r: immediate reward (from executing a in s)
- s': state resulting from executing a in s
- γ: discount factor for future reward (in this solution it is set to 0.99)

Two neural networks were used: Q-network and target network. The Q-network is the network being trained, however the target network takes the next state from each data sample and predicts the best Q value out of all actions that can be taken from that state. This is the 'Target Q Value'. This operation can also be done without the use of the second network, but in studies it has been found that the use of a target network makes the train more stable. it is possible to construct a DQN with a single Q network and no target network. In that case, we make two passes through network Q, first to output the expected Q value and then to output the target Q value. But this could create a potential problem. The Q network weights are updated at each time step, which improves the prediction of the expected Q value. However, since the network and its weights are the same, the direction of our predicted Target Q values also changes. They do not remain stable but may fluctuate after each update. By employing a second network that is not trained, we ensure that the Target Q values remain stable, at least for a short time. But those Target Q values are also predictions after all and we want them to improve, so a trade-off is made. After a preconfigured number of time steps (in this solution every 5 episodes), the weights learned from network Q are copied to the destination network.

The Q-network and the target network have the same structure:

- *input layer*: has 24 neurons, ReLU activation function and the input shape is based on the *state_shape* variable representing the shape of the environment's observation space
- *hidden layer*: has 48 neurons, ReLU activation function

- *output layer*: the number of the neurons in this layer is determined by the *action_space,* which represents the number of possible actions in the environment. It uses the linear activation function, which means the layer outputs the Q-values for each action.

Another important function in your code is the function that returns the action to be performed. The policy that has been implemented is *ε-Greedy strategy*:

- fix ε ∈ [0, 1]
- select a random action with probability ε, *exploration*
- select a best action with probability 1- ε (action that maximizes Q(s,a)), *exploitation*
- ε decrese over time with decay rate 0.95 (prefer exploration first, then exploitation)

1.2 Training analysis

The following train parameters were used for the agent train:

- episodes number = 300
- batch size = 128
- γ (discount factor) = 0.99
- learning rate = 0.001
- starting ε = 1
- min ε = 0.01
- decay rate for ε (exploration or exploitation) = 0.95

To evaluate the agent's training, the cumulative rewards and the maximum positions reached by the car for each episode were memorized. These values were used to generate graphs to visualize training progress.

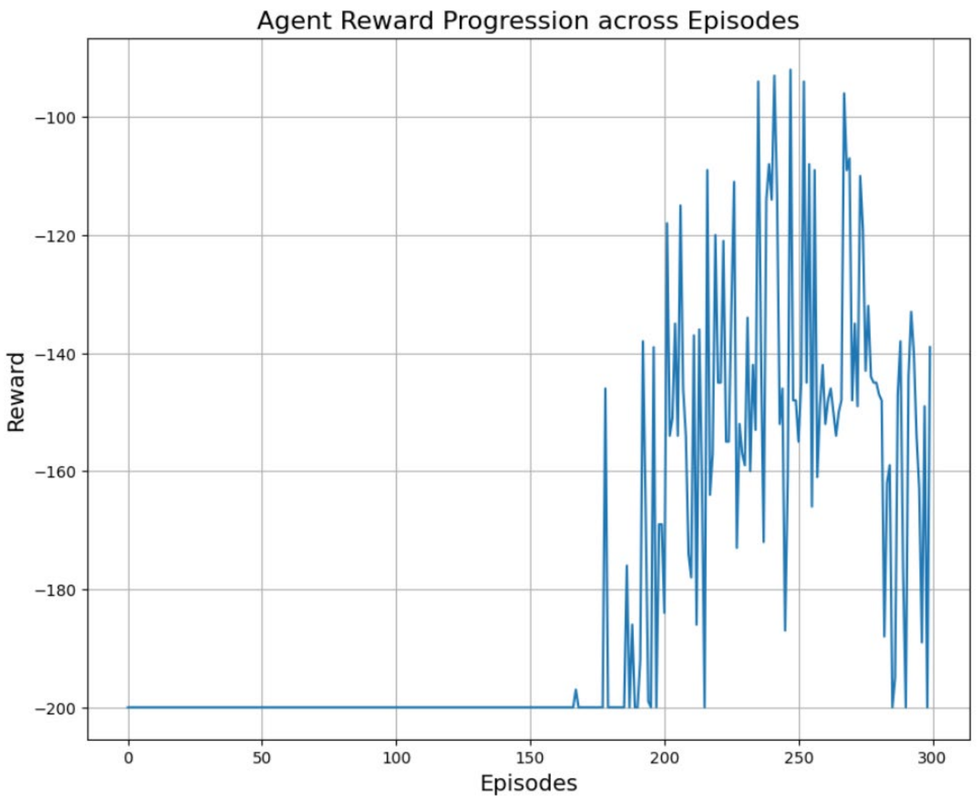


Figure 1.2.1

In graph 1.2.1 it can be seen that in the initial phase the reward remains constant -200, which means that the agent never reaches the goal (in particular, in this environment the car never reaches the flag on top of the mountain) ; as you progress through the episodes, the goal is increasingly achieved (the goal is achieved if the reward is greater than -200, which indicates the timestamps it took the machine to reach the flag), until the approximately episode 200 after which the agent converges on the target.

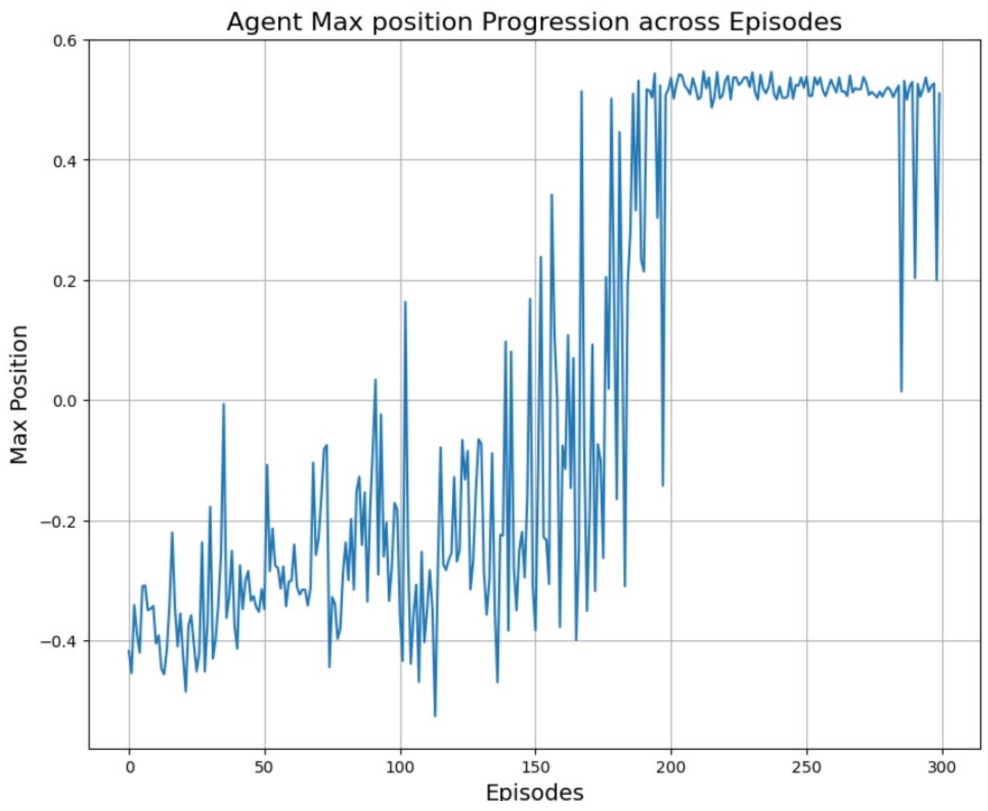


Figure 1.2.2

In graph 1.1.2 it can be seen that as the episodes increase, the maximum position reached by the car increases, until it reaches and goes beyond episode 200 where the agent converges towards the objective, i.e. the maximum position coincides with the position of the flag (0.5).

1.3 Test analysis

Once a first training phase was completed and a trained neural network was obtained, this same network was used for a test phase. In this test phase, the trained network was used to extract the actions that the car must do to reach the goal. 100 iterations were performed in order to analyze the number of times the car hits the target versus the number of times the car fails. 100% of successes were achieved on 100 repetitions performed. It can be concluded not absolutely, but with good accuracy that the trained model performs well for this type of environment. The plot of the cumulative rewards on 100 episodes is in figure 1.2.3.

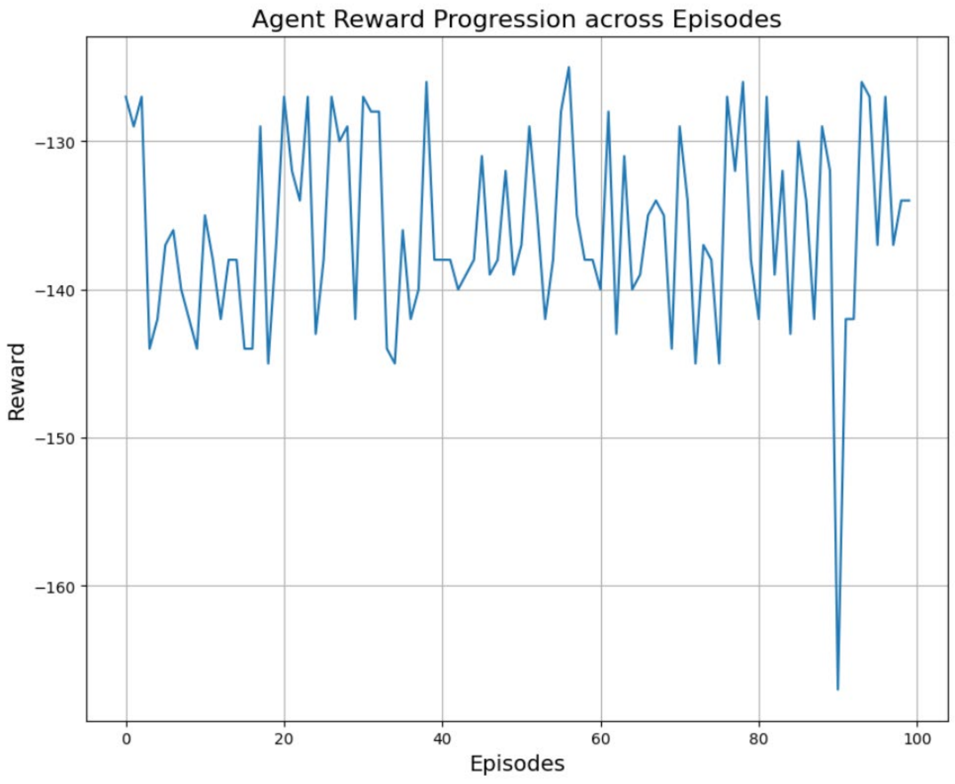


Figure 1.2.3

1.4 Analysis between agent with train NN and random agent

Finally, the use of the trained NN with respect to a random agent, which performs random actions, was also analysed. The random agent out of 100 repetitions never reached the objective, compared to the trained model which, as previously mentioned, achieved the objective 100% of the cases.

2. Mountain Car Variant

This variant of the Mountain Car has been implemented to evaluate the performance of the agent previously trained in the classic environment. In this way, the trained model can be evaluated in another environment than the one in which it was trained.

The idea of this variant remains the same as the Mountain Car, leaving the actions, rewards, goal, end condition, etc. unchanged; the addition was that of an intermittent laser. The role of this laser is as follows: when the car crosses this laser it is stopped, in particular the speed is set to 0.

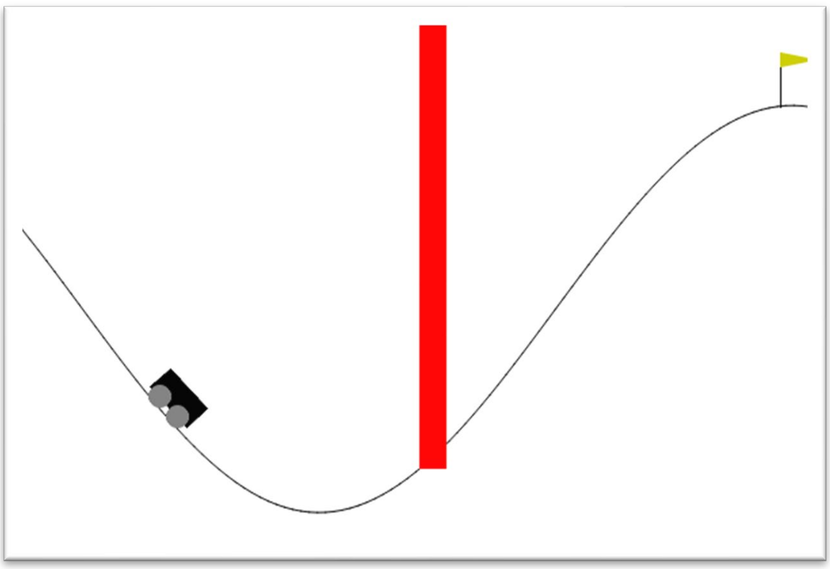


Figure 2.1

Graph 2.2 shows the reward trend in the new environment, considering 100 episodes, using the previously trained network to extract the actions to be performed.

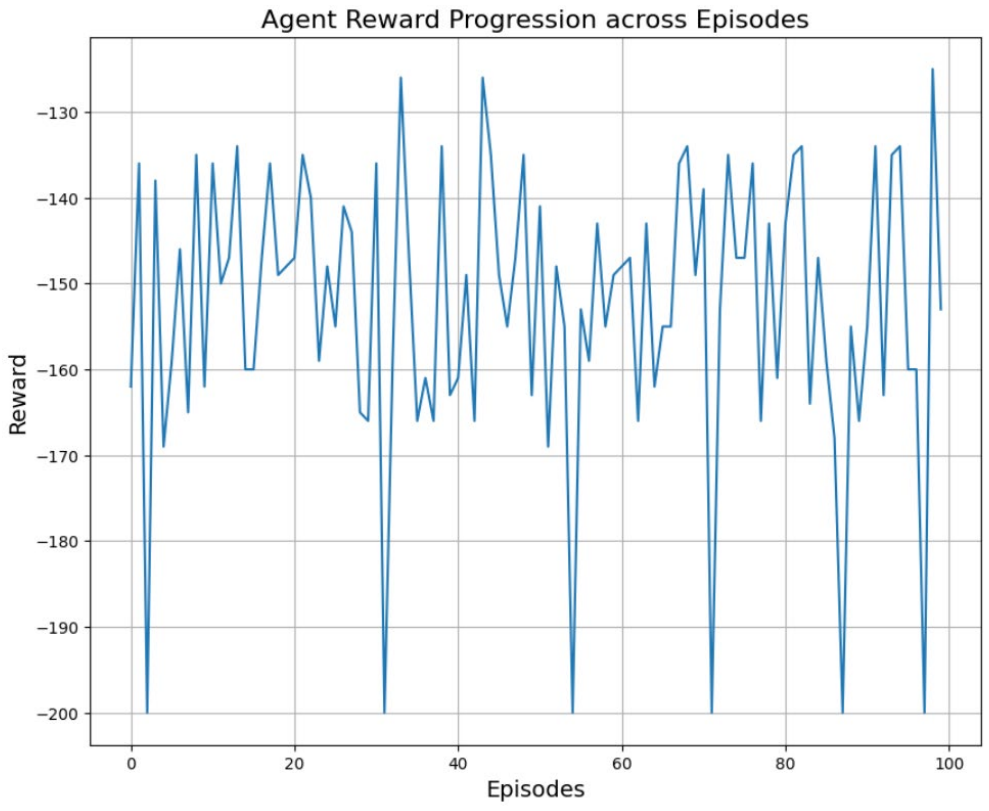


Figure 2.2

From the graph it can be seen that the trained model performs well even in a new environment; in fact, considering 100 episodes only in 6 episodes the machine did not reach the goal.