**Deep RL project report**

**Plan**

Description of our environment

Implementation

Learning algorithm

Hyperparameters

Model architecture and hyperparameters

Detailed result

1. ***Description of our environment***

**MountainCar v0**: A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum

**Observation:**

Num Observation Min Max

0 position -1.2 0.6

1 velocity -0.07 0.07

**Actions:**

Type: Discrete (3)

Num Action

0 push left

1 no push

2 push right

**Reward:**

-1 for each time step, until the goal position of 0.5 is reached. As with MountainCarContinuous v0, there is no penalty for climbing the left hill, which upon reached acts as a wall.

**Starting state:**

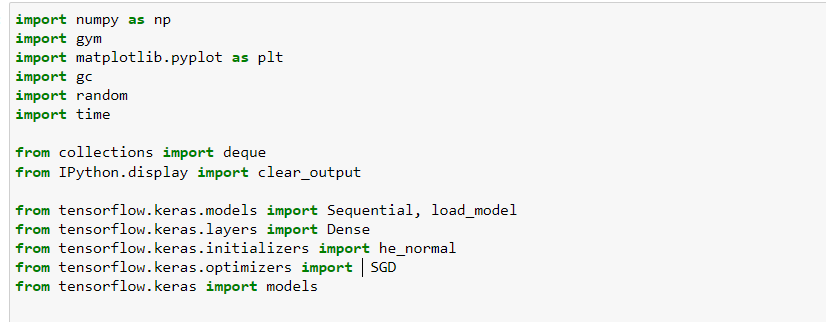
Random position from -0.6 to -0.4 with no velocity.

**Episode Termination:**

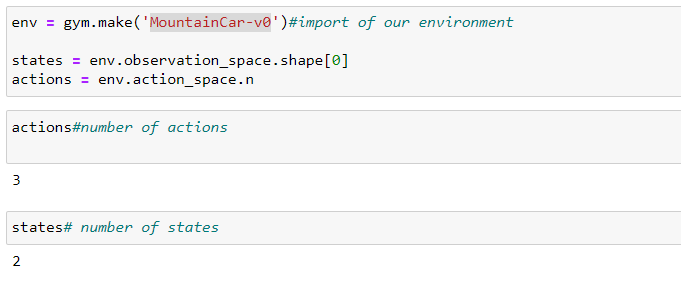
The episode ends when you reach 0.5 position, or if 200 iterations are reached.

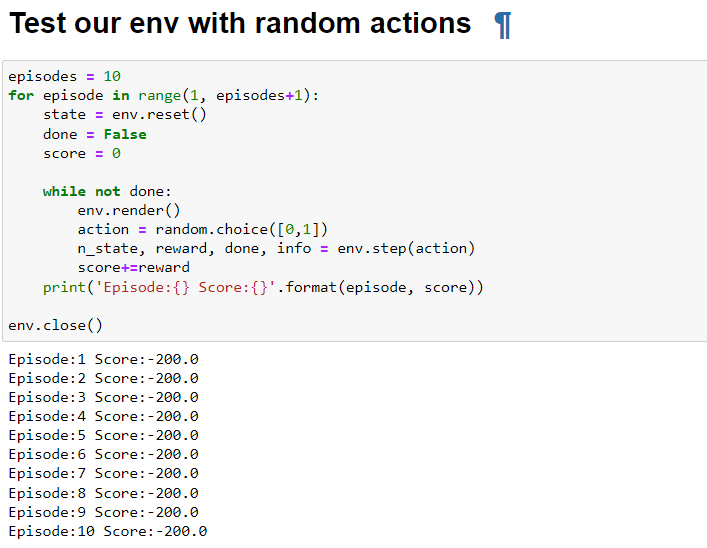
1. **Implementation**

Import of libraries

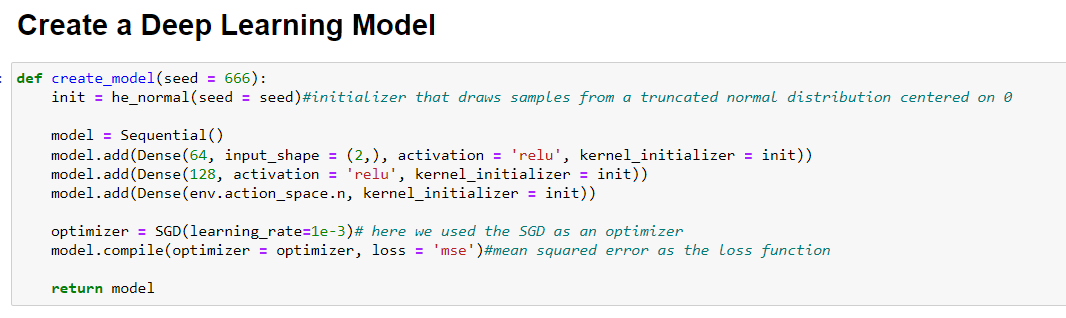
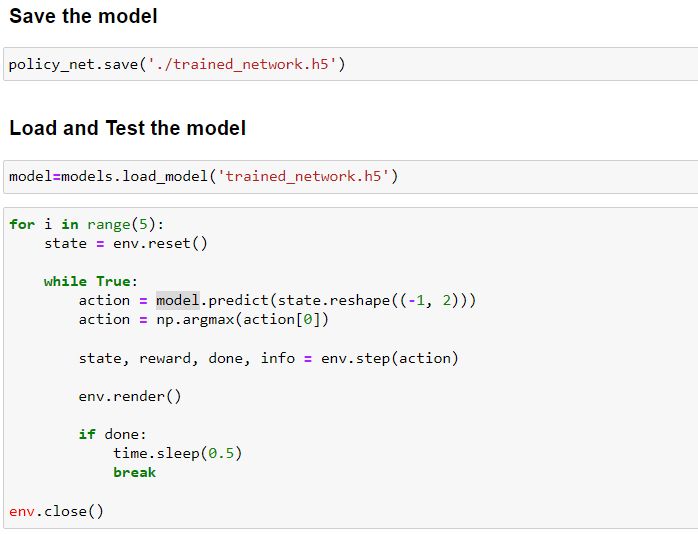


We imported our environment MountainCar-v0 using gym library, and as we see below we have 3 actions and 2 states

After, we tried to visualize how the environment evolve using random action, just to have a little perspective about it.

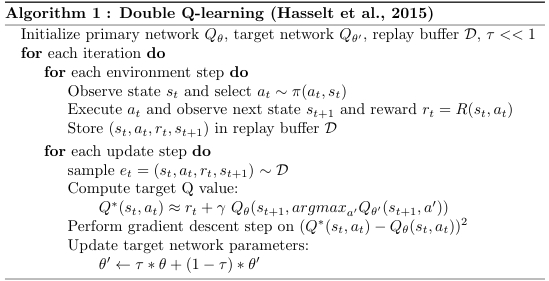


Create\_model is used to create two identical neural network models. One learns during the experience replay, just like DQN does, and the other one is a copy of the last episode of the first model. These two networks are composed from three layers. The number of neurons in the input layer is the environmental state dimension. The number of neurons in the output layer is set to the environmental action dimension for all the models. Relu (Rectified Linear Unit) is used as the activation function, and the SGD algorithm is selected as an optimizer.

 In the end, we saved our model “trained\_network.h5” and then we load it to test it on our environment and see how preferment it is.

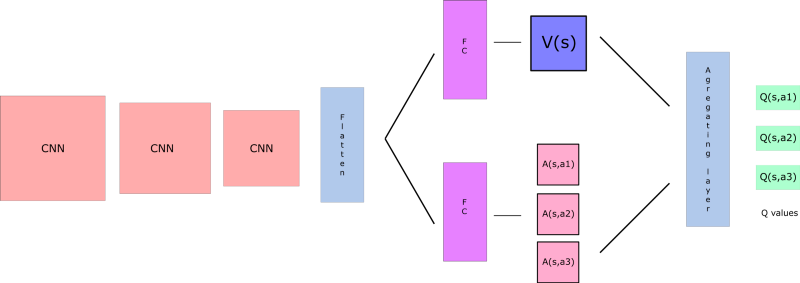
1. **Learning algorithme**

**Double DQN:**



1. **Model architecture**

One of the problems of the DQN algorithm is that it overestimates the true rewards. the Q-values think that the agent will get a higher return than what it will actually get. To solve this problem, the authors of the Double DQN algorithm suggest to use a simple trick, decouple the action selection from the action evaluation. So Instead of using the same Bellman equation as in the DQN algorithm,



1. **Hyperparameters**

15 episode to train on with max 200 step

The exploitation rate for choosing the action according to the e-greedy strategy.

Batch-size=32 (for each 32 transition in the experience replay memory update booth q-networks)

Discount-rate =0.99

