AI 832: Reinforcement Learning

Project

Ankit Agrawal

IMT2019010

Files Submitted

The files submitted with my assignment are:

1. **Report.pdf** – The PDF file containing the report for this project.

2. **RL Model Training.ipynb** – This notebook contains the full code (environment and DQN agent) used for training the DQN agent and saving the models and plots.

3. **RL Model Testing.ipynb –** This notebook contains the code on which the trained model can be loaded and checked for performance (like running different episodes using the model).

4. **TM_lower.npy** – This NumPy file contains the time matrix array. Note that this array is different from the original time matrix array, because the zero entries were removed and replaced by the average of the non-zero entries in the time matrix. This should be used as a time matrix in both **RL Model Training.ipynb** and **RL Model Testing.ipynb** .

5. **Converged_1900.h5** – This is the trained, converged model. This model can be used in the **RL Model Testing.ipynb** for checking the performance. Please note that this is not a pickle object, as pickle doesn't allow us to save Keras models in pickle format. The loading and saving od models is done as mentioned in this link: https://www.tensorflow.org/guide/keras/save_and_serialize.

Problem Statement

In this project, we are required to create a DQN agent for a taxi driver. The main aim of the taxi driver is to maximize rewards obtained from the environment. The agent is to be trained so as to be able to:

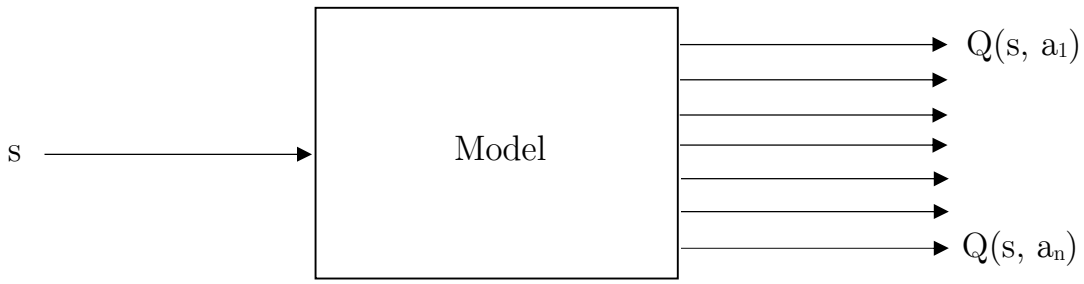
1. Decide which ride to take (from a set of rides provided by the environment).
2. Decide whether not to take any ride.

The environment is governed by the MDP that has been provided in the problem statement. In a nutshell, the environment has:

1. Total 5 locations, which are [1,2,3,4,5]
2. Total 21 actions, including the action where the driver doesn't take any ride
3. A time matrix

DNN Architecture

The model that we are training is based on a deep neural network. We are following the second architecture, in which given a state s, the model outputs the Q-values of all actions a possible from the action space (including the action [0,0], where the driver does not accept any ride). The architecture of such a model is shown below:



The summary of the model is shown below:

```
Model: "sequential"
_________________________________________________________________
    Layer (type)                 Output Shape              Param #
=================================================================
    dense (Dense)                (None, 64)                2368
_________________________________________________________________
    dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
    dense_2 (Dense)              (None, 21)                693
=================================================================

Total params: 5,141
Trainable params: 5,141
```

The hyperparameters used for training the model are:

$$\text{Discount factor } (\gamma) = 0.95$$
$$\text{Learning rate } (\alpha) = 0.01$$
$$\text{Epsilon } (\varepsilon) = 0.999$$
$$\text{Epsilon-decay} = 0.995$$
$$\text{Minimum epsilon } (\varepsilon_{min}) = 0.01$$

Note that the epsilon decay is applied after every 4 episodes. This helps the model to explore enough and at the same time converge in a reasonable amount of time.

Convergence Plots and Results

The model was trained for about 1900 episodes on CPU on Kaggle. I would have loved to have done a little more training on the model. However, due to limitations on maximum number of hours a notebook can be run on Kaggle (and the fact that my laptop doesn't have enough computing power to train such models), I could not train my model further.

Despite this, the results are very nice. The model has also converged reasonably well. This is shown in the plot below, which shows the rewards obtained for each episode:

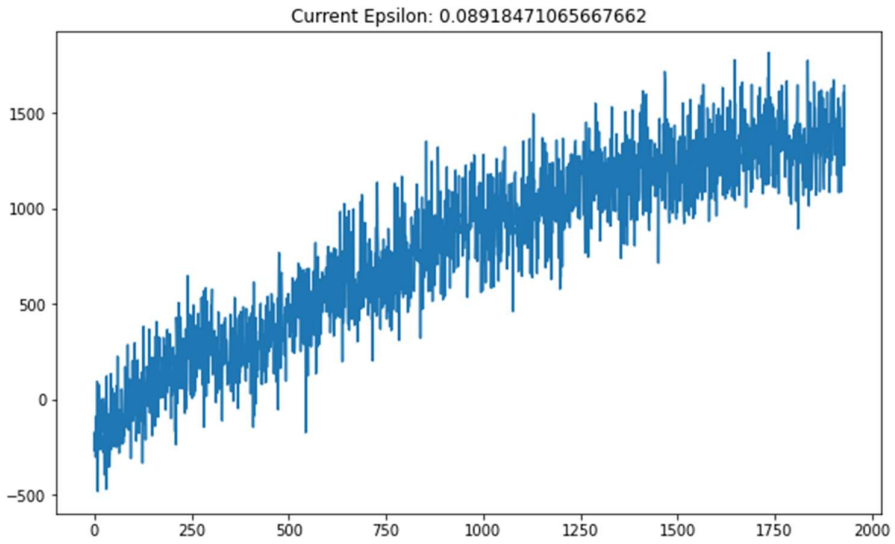


Figure showing plot of rewards obtained (y-axis) against the Episode number (x-axis). Notice that the reward has almost converged.

Note that the randomness in rewards can be attributed to the randomness of the environment. This is the reason why the rewards shown in the above plot are noisy. A smooth plot of the rewards obtained is also shown in the next page…

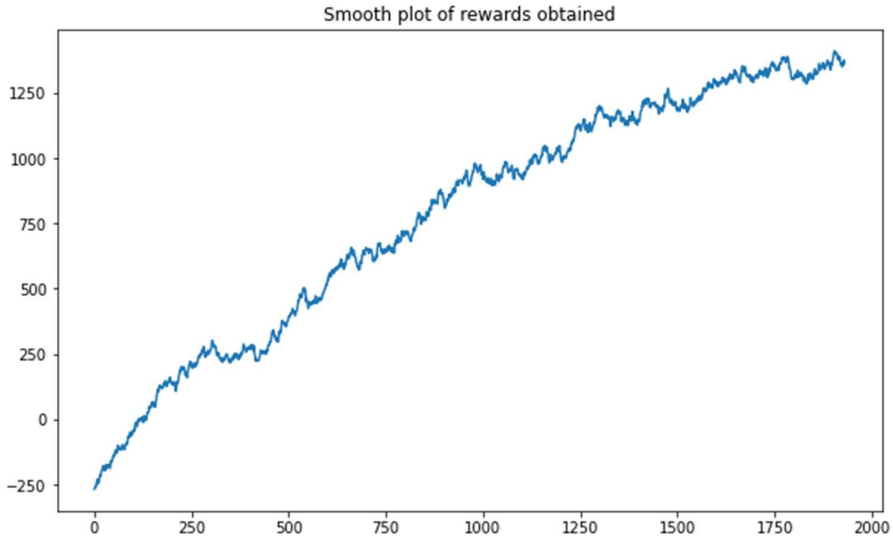


Figure showing a smooth plot of rewards. The y-axis represents reward and the x-axis represents the episode number

For plotting the Q-values, I chose the state as [1,0,0], i.e. location 1, time 0 and day 0 and the action I chose for plotting was [1,2]. The Q-value plot is shown below:

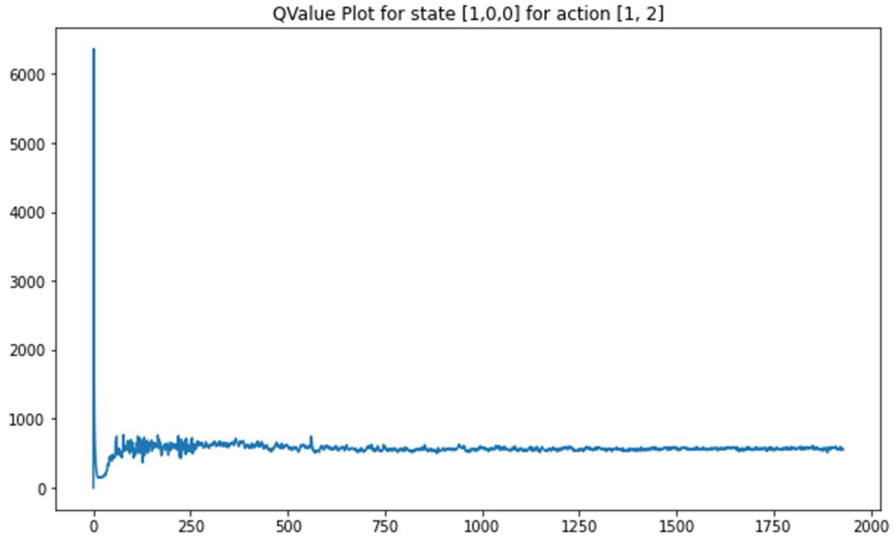


Figure showing the Q-value plot. The y-axis represents the Q-value and the x-axis represents the episode number. The Q-values have converged.

Notice that the Q-values have converged. Initially, the model had an over-estimate of the Q-value of this state-action pair (around 6000), but then it adjusted itself to a more realistic estimate.

A plot showing how the epsilon decay happens is also shown in the next page…

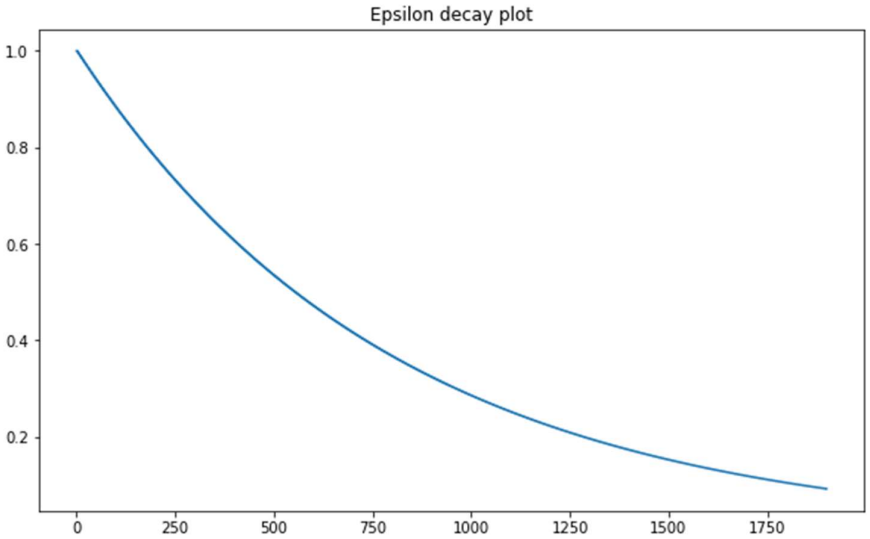


Figure showing the epsilon decay plot