SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Machine Learning – Reinforcement Learning

2019 Fall

Nijat Mursali | 1919669

PROJECT

Rome, Italy

Table of Contents

List of Figures

Introduction

Before speaking about our algorithm, we have to first describe the problem statement itself and which tools will be used to implement our algorithm. In the following paragraphs, we will be describing the reinforcement learning, Q-Learning which is part of reinforcement learning and OpenAI Gym which is the environmental framework to implement our algorithm.

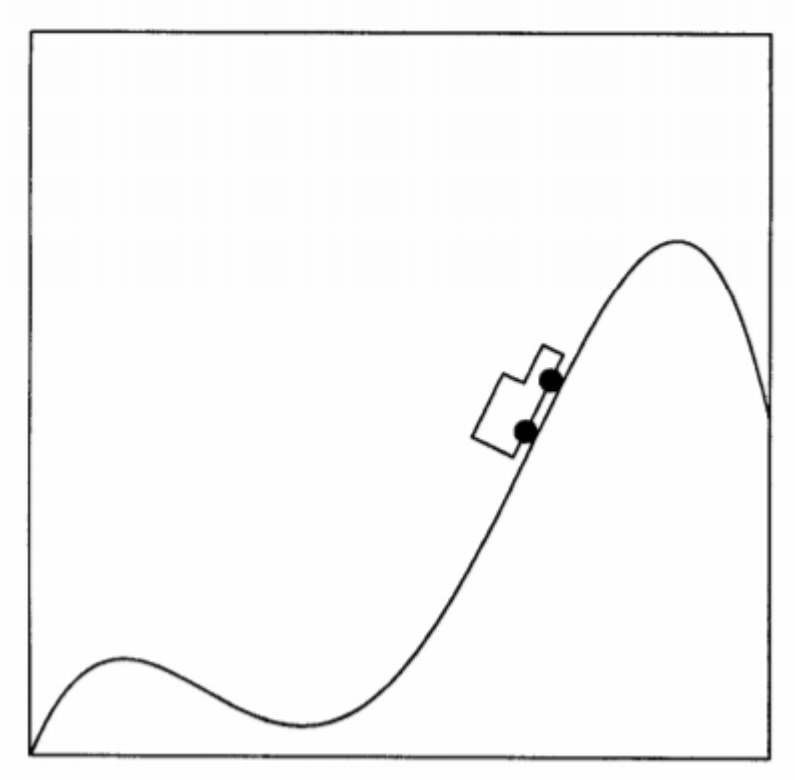

Fig 1.2 A Mountain Car by Andrew William Moore

A.W. Moore has explained this problem in his master thesis as seen in *Fig 1.2* where car drives in one-dimensional track over mountain range. It is almost same as our problem as we will be discussing and experimenting the implementation of the algorithm.

Reinforcement Learning

As we have already known, machine learning is about supervised and unsupervised learning where we work with labelled data in supervised learning; however, in unsupervised learning we don't have any labelled dataset or answers beforehand. Thus, in this case we have to learn by doing actions where reinforcement learning comes to our mind. Reinforcement learning

Q-Learning

To make it short, the fundamental goal of Q-learning is to learn a policy which tells an agent what action to take under what circumstances.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Fig 2.1 Q-learning

In this formula, change of *t* shows the steps and values the Q get after every iteration. *Discount factor* is a number between 0 and 1 and it affects the rewards and it is interpreted as the probability to succeed at every step of *t*. In the beginning of evaluation, *Q* value is possibly arbitrary fixed value which can be chosen by the programmer and after every iteration agent chooses an action (in our case going to left and right), getting the reward, entering to a new state *s* and updating *Q*. Learning rate is called step size where determines the new value information that overrides the previous one.

In following paragraphs, we will be talking about the OpenAI Gym which is powerful library in order to implement the algorithms.

OpenAI Gym

As written in their official repository, OpenAI Gym is one of the toolkits for developing and comparing reinforcement algorithms. Gym library itself is a collection of test problems which are environments that are used to implement reinforcement learning algorithms. Gym has different kind of important features such as environments, observations, spaces and so on. We will use *"MountainCar-v0"* environment for our report where the fundamental purpose will be to implement reinforcement learning to make car go to the goal state. Observations are used to make actions and rewards higher in the environments. Observations take several actions in itself such as observation, reward and done variable which states if the search is done or not.

Fig 2.2. OpenAI Gym and Q learning

In next paragraphs we will be describing how to import *gym* library itself and how to work with environment by implementing reinforcement learning.

Implementation

As seen *Fig 2.1* we need to initialize some variables such as learning rate, discount factor beforehand. For this project, we have chosen the learning rate 0.9 and discount factor 0.95.

To implement our algorithm, first we needed to add the *gym* library into our Jupyter code and create the environment of *MountainCar* as we can see from the documentation of

OpenAI Gym. We have illustrated the initial step of the algorithm where we have initialized several variables as we can see from *Fig 2.2.*

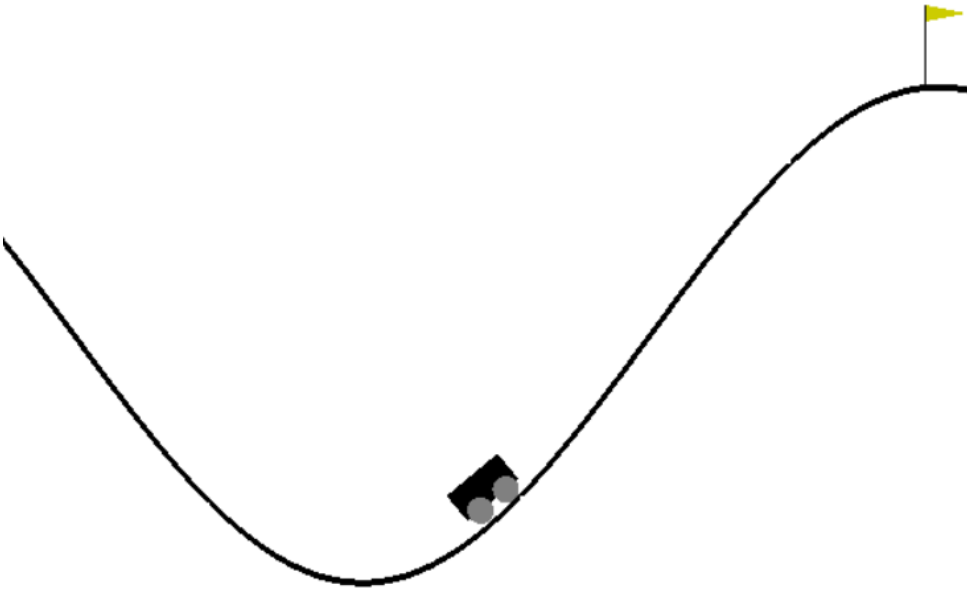

Fig 2.3 Visualization for default case

Whenever object interacts with the environment with *action*, then after every step it returns the different placement of the object, *reward* where he has achieved the goal or not, if *done is true* then we can reset the environment and *info* is for debugging and can be used to detect if the object reached the goal (last state) or not. *Done* has two outputs: either *true* or *false.* The output of the code for initial state is shown in *Fig 2.3.*

```
0
[-0.53144896 -0.00094824]
Step 1: State=[-0.53144896 -0.00094824], Reward = -1.0
1
[-0.53233833 -0.00088937]
Step 2: State=[-0.53233833 -0.00088937], Reward = -1.0
0
[-0.53416215 -0.00182383]
Step 3: State=[-0.53416215 -0.00182383], Reward = -1.0
1
[-0.53590677 -0.00174462]
Step 4: State=[-0.53590677 -0.00174462], Reward = -1.0
2
[-0.5365591  -0.00065233]
Step 5: State=[-0.5365591  -0.00065233], Reward = -1.0
2
[-5.36114250e-01  4.44849916e-04]
Step 6: State=[-5.36114250e-01  4.44849916e-04], Reward = -1.0
1
```

Fig 2.3.1 Output of the initial state

If we print out the environment *action space*, it is set of valid actions in that state and we can see that our environment is ***discrete***.

As we have already mentioned, our fundamental job here is to make the car climb to the hill. Thus, our environment is the 2-D grid and agent is the car that does actions through the environment and trying to find the better policy and Q-value. Thus, we tried to find the Q value and update it in every next iteration. We have tried to do ten thousand episodes in overall and showing five hundread episodes each. To mention, we have used the formula above in order to solve the problem and update it. As parameter, we have chosen learning rate of 0.1, discount factor of 0.95. Then, we are trying to get our discrete state by resetting the environment which will put the car in random location. As you might see from the *Fig 2.4*, we have received the maximum of 0.98% of success in 10000 steps.

```
Current episode: 500, success: 30, (0.06)
Current episode: 1000, success: 102, (0.204)
Current episode: 1500, success: 199, (0.398)
Current episode: 2000, success: 263, (0.526)
Current episode: 2500, success: 327, (0.654)
Current episode: 3000, success: 284, (0.568)
Current episode: 3500, success: 370, (0.74)
Current episode: 4000, success: 167, (0.334)
Current episode: 4500, success: 178, (0.356)
Current episode: 5000, success: 327, (0.654)
Current episode: 5500, success: 287, (0.574)
Current episode: 6000, success: 439, (0.878)
Current episode: 6500, success: 490, (0.98)
Current episode: 7000, success: 469, (0.938)
Current episode: 7500, success: 500, (1.0)
Current episode: 8000, success: 500, (1.0)
Current episode: 8500, success: 475, (0.95)
Current episode: 9000, success: 313, (0.626)
Current episode: 9500, success: 352, (0.704)
Current episode: 10000, success: 348, (0.696)
```

Fig 2.4 Iterations and Successes for Mountain Car

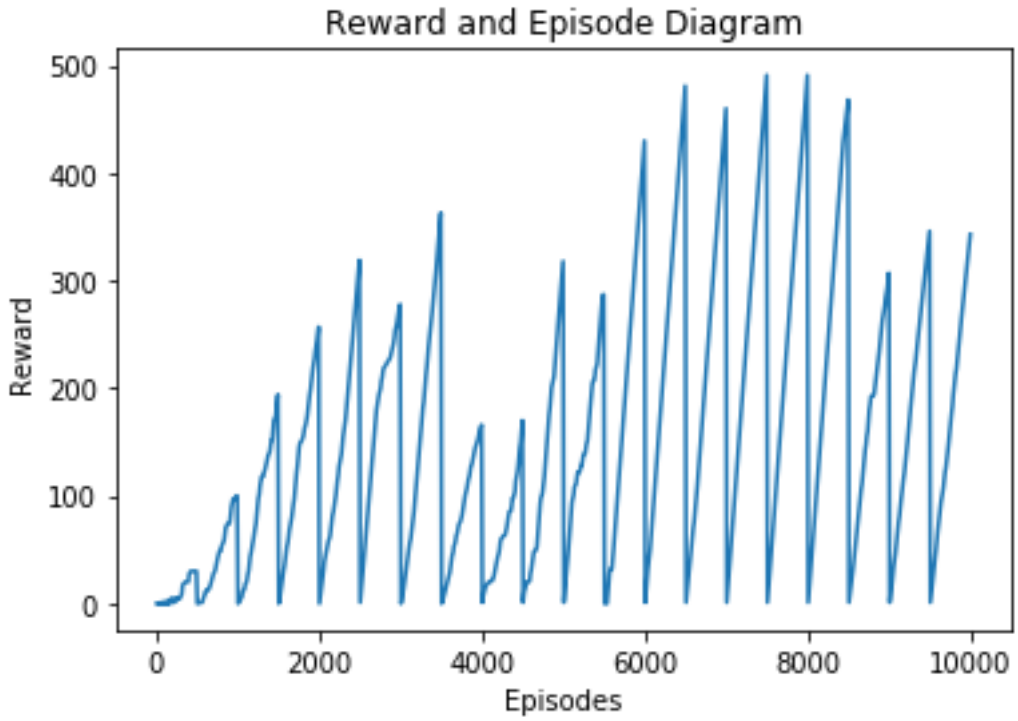

Fig 2.5 Plotting Mountain Car

Our next algorithm to solve was Bowling which is one of the famous games of Atari which you can see in the Fig 2.6.

Fig 2.6 Atari Bowling

The main idea here is to make the agent (which is the man) hit all the balls and win the game. We have used CNN in order to solve this problem and plotted the values into the graph which we will be showing in next paragraphs. To make it short, we have trained our model with one thousand episodes using the batch size of 32 and writing all the data into text file and plotting the data from it. In Fig 2.7 we have showed the number of episodes and the rewards for our problem.

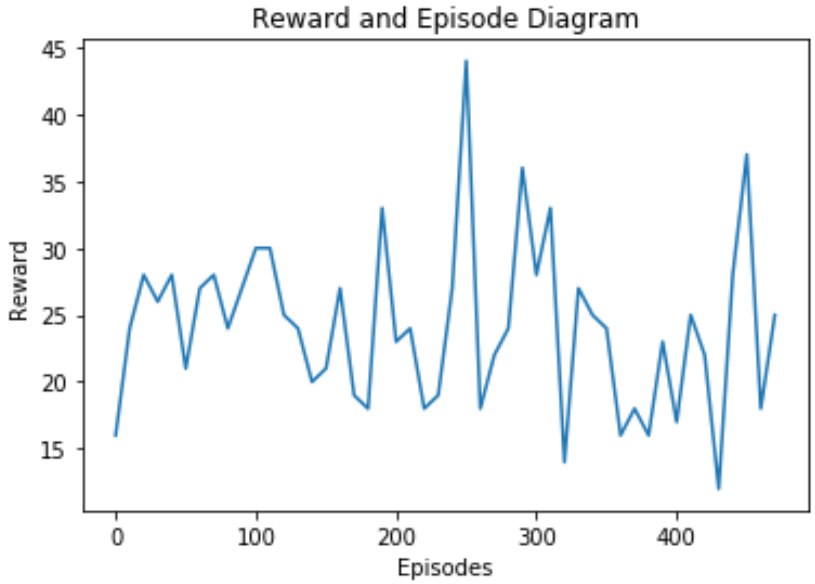

Fig 2.7 Plotting Bowling-v0

Conclusion

To conclude, in this report we have tried to explain two algorithms: mountain car and bowling from OpenAI. As we have seen from the graphs, we have achieved the accuracy (successes) of 0.98 in mountain car and 45 with bowling. For the future work, we will be trying to experiment those algorithms and get better results.

References

1. OpenAI Gym Documentation
   - https://gym.openai.com/docs/
   - https://gym.openai.com/envs/