RLdrive: Using Reinforcement Learning to make better Driving Decisions

Aditya Jain
IIIT-Delhi
New Delhi, India
aditya14129@iiitd.ac.in

Manasi Malik
IIIT-Delhi
New Delhi, India
manasi15146@iiitd.ac.in

Abstract

In this paper, we aim to solve the classical grid world problem using a Reinforcement Learning (RL) technique - Q Learning. In the mid-semester report, we explored DP methods (policy evaluation, policy iteration, value iteration) to find the optimal paths. Standard techniques (like BFS, DFS, Dijkstra etc.) for finding shortest paths were used for verifying our results. In this final project report, we explore value and policy iteration for static obstacles in the grid. Next, we show Q Learning results for the same environment. Finally, we explore a novel idea wherein the position of obstacles in the grid change with every episode (but picked from a fixed distribution) i.e. the environment is dynamic. We aim to evaluate if the agent is able to learn the obstacle distribution during its encounter with the environment in different episodes and is able to reach the terminal state in a test grid, given different grid configurations.

1. Introduction

1.1. Motivation

In a not so distant future, when we will have a connected network of autonomous cars on the roads, can we leverage the use of spatial and temporal information obtained from different vehicles to make better decisions regarding cruise speed, overtaking and lane-changing?

1.2. Problem Statement

With the above motivation, we are scaling down the problem to a grid-like environment where different cells represents their state (occupied/free of obstacle) and the agent has to travel from the start to the goal position (or cell) using RL techniques, thus maximizing the overall rewards. In addition to above, obstacles may also change their positions in the grid (just like cars on the road) at each iteration.

With the above scenario, can the agent cover the distance in the shortest time as well as maximize the total reward?

2. Related Work

While lot of people have used RL techniques to solve various grid related problems with different constraints and optimizations, none have really looked into querying by agents and dynamic state of the environment. Galstyan in [1] discusses about resource sharing among multiple agents, [3] explores multi-agent RL approach for job scheduling in grid computing. [2] addresses the traveling salesman problem using RL techniques.

3. Methodology

To scale down the problem statement, we are solving the grid world problem (reach from start to goal position using the shortest path while maximizing the rewards) in [4] with our added constraints. The environment is defined using a 4x4 grid which gives total 16 cells. The general terminologies used are defined as:

agent - learner and decision maker; **environment -** everything outside the agent with which it interacts; ***s* -** current state of the agent i.e. position in the grid; ***r* -** reward which the agent collects while interacting with the environment ; ***a* -** action which the agent can take to go from one state to another (up, down, right and left in the grid problem); π **-** policy, decision-making rule; ***v*π*(s)* -** value of state s under policy π (expected return); ***q*π*(s,a)* -** value of taking action a in state s under policy π; **terminal state -** final goal position/state,state can't change once reached

In this version of the report, we have implemented policy iteration, value iteration and Q learning for static obstacles. We also explored Q learning for randomly placed obstacles from a fixed distribution. These methods are described below:

3.1. Policy and Value Iteration for Static Obstacles

The obstacles are deterministically placed in the grid and are represented by '7'. The terminal state is represented by '5'. We evaluated policy and value iteration on the same.

The results are discussed in section 4. Dynamic programming is limited in its usage because of two major reasons: it assumes a perfect model of the environment and is computationally expensive. Thus, Q learning is discussed in the below section.

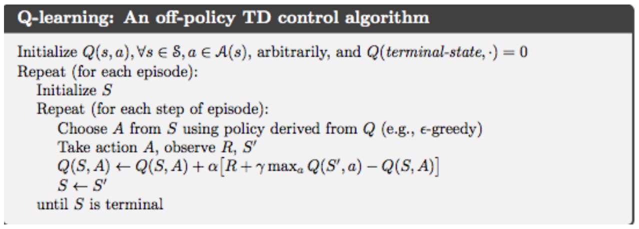
3.2. Q Learning for Static Obstacles

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Figure 1. Action-Value function update in Q learning

In this case, the learned action-value function, Q, directly approximates q*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which stateaction pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. Below shows the Q-learning algorithm in procedural form:



Q-learning: An off-policy TD control algorithm

```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Repeat (for each step of episode):
      Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      Take action $A$, observe $R$, $S'$
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
      $S \leftarrow S'$
   until $S$ is terminal
```

Figure 2. Q-learning algorithm

The results of Q-learning for the case of static obstacles in the grid are discussed in section 4. The obstacles are generated randomly from a known distribution.

3.3. Q Learning for Changing Obstacles

Unlike earlier where the obstacle positions were fixed for all episodes of training, now they are randomly picked from a known distribution in each episode. The chosen distribution is **Gaussian, centered at 7 with standard deviation equal to 1.** Thus most of the values obtained are between 4 and 10, with the number "7" representing the obstacle position in the grid.

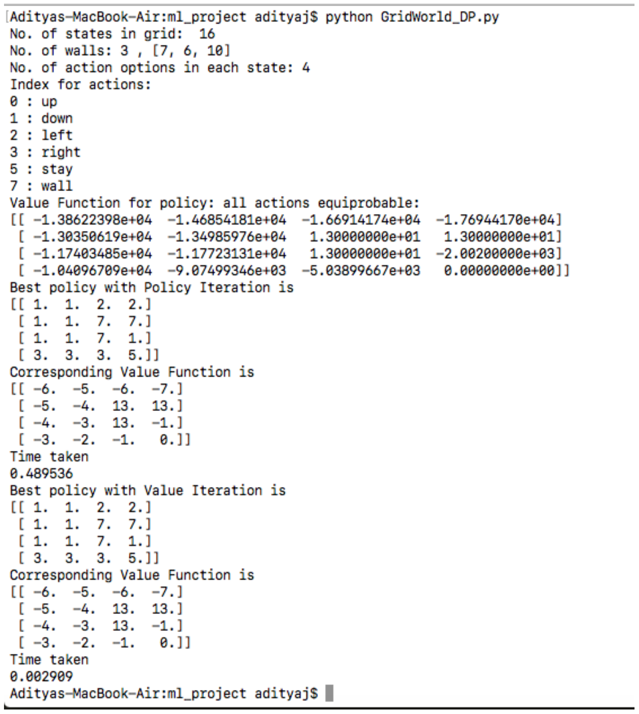
Since the obstacle position is changing with every episode now, the entire grid will now be the state space for the agent so that it can learn the obstacle distribution and hence make better decisions. The **total number of possible state spaces is 15!/(3!*11!) = 5460.** To test the learning curve of the agent, we **run the algorithm for N episodes** and store the best action for each unique state. Then given a test grid with obstacle positions derived from the same gaussian distribution, we search from the state-action pair learned by the agent during training from the start till the terminal state in the test grid. If the agent is able to reach the terminal state, it is considered a success .The results and analysis will be discussed in greater detail in the next section.

4. Results and Analysis

4.1. Policy and Value Iteration for Static Obstacles

The environment i.e. 4x4 grid was taken as follows: **No. of states** $\rightarrow$ 16; **Possible actions in each state** $\rightarrow$ (up,down,left,right); **Reward** $\rightarrow$ -1 for each step taken; **Terminal states** $\rightarrow$ state 16; **Discount Rate** $\rightarrow$ 1.0 **Threshold** $\rightarrow$ 0.0001 Actions that take the agent off the grid will leave the state unchanged.

The top left cell is state 0 and bottom right 15. Policy and value iteration were tested for two sets of obstacle configuration: **[7, 6, 10]** and **[4, 6]**. The results are shown in **figure 3 and 4**:

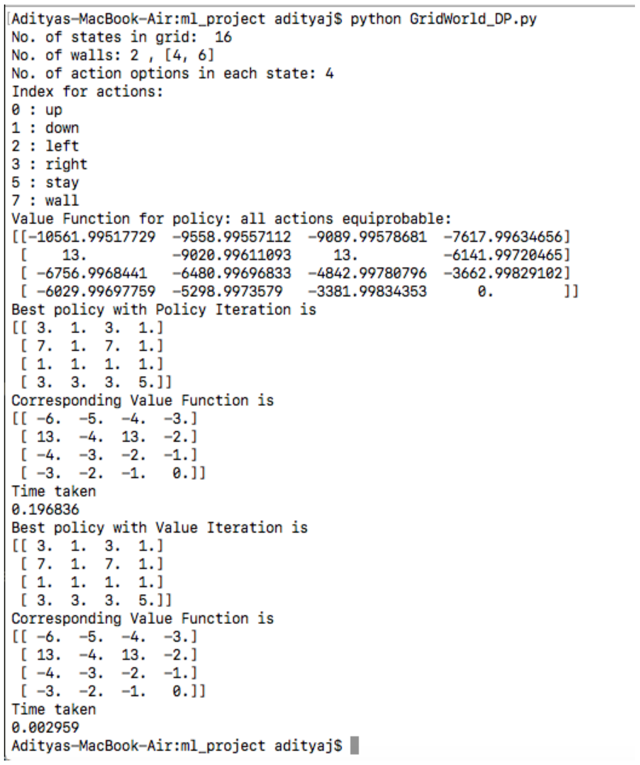


```
Adityas-MacBook-Air:ml_project adityaj$ python GridWorld_DP.py
No. of states in grid:  16
No. of walls: 3 , [7, 6, 10]
No. of action options in each state: 4
Index for actions:
0 : up
1 : down
2 : left
3 : right
5 : stay
7 : wall
Value Function for policy: all actions equiprobable:
[[ -1.38622398e+04  -1.46854181e+04  -1.66914174e+04  -1.76944170e+04]
 [ -1.30350619e+04  -1.34985976e+04   1.30000000e+01   1.30000000e+01]
 [ -1.17403485e+04  -1.17723131e+04   1.30000000e+01  -2.00200000e+03]
 [ -1.04096709e+04  -9.07499346e+03  -5.03899667e+03   0.00000000e+00]]
Best policy with Policy Iteration is
[[ 1.  1.  2.  2.]
 [ 1.  1.  7.  7.]
 [ 1.  1.  7.  1.]
 [ 3.  3.  3.  5.]]
Corresponding Value Function is
[[ -6.  -5.  -6.  -7.]
 [ -5.  -4.  13.  13.]
 [ -4.  -3.  13.  -1.]
 [ -3.  -2.  -1.   0.]]
Time taken
0.489536
Best policy with Value Iteration is
[[ 1.  1.  2.  2.]
 [ 1.  1.  7.  7.]
 [ 1.  1.  7.  1.]
 [ 3.  3.  3.  5.]]
Corresponding Value Function is
[[ -6.  -5.  -6.  -7.]
 [ -5.  -4.  13.  13.]
 [ -4.  -3.  13.  -1.]
 [ -3.  -2.  -1.   0.]]
Time taken
0.002909
Adityas-MacBook-Air:ml_project adityaj$
```

Figure 3. PI and VI for 1st Configuration

4.2. Q Learning for Static Obstacles

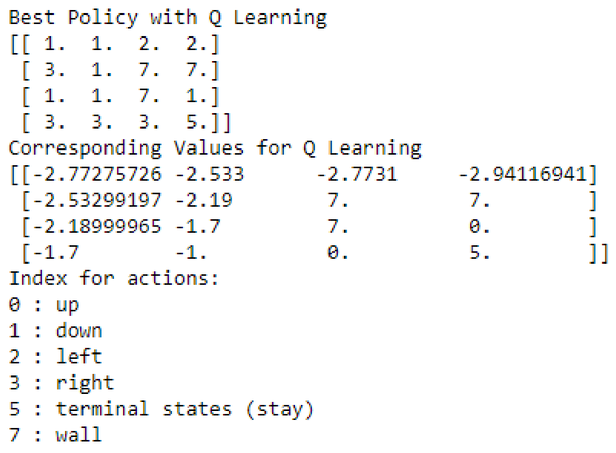
For the same obstacle environment as above, Q learning was tested and its results are shown in **figure 5** and **figure 6**:



```
Adityas-MacBook-Air:ml_project adityaj$ python GridWorld_DP.py
No. of states in grid:  16
No. of walls: 2 , [4, 6]
No. of action options in each state: 4
Index for actions:
0 : up
1 : down
2 : left
3 : right
5 : stay
7 : wall
Value Function for policy: all actions equiprobable:
[[-10561.99517729  -9558.99557112  -9089.99578681  -7617.99634656]
 [    13.          -9020.99611093     13.          -6141.99720465]
 [ -6756.9968441   -6480.99696833  -4842.99780796  -3662.99829102]
 [ -6029.99697759  -5298.9973579   -3381.99834353      0.        ]]
Best policy with Policy Iteration is
[[ 3.  1.  3.  1.]
 [ 7.  1.  7.  1.]
 [ 1.  1.  1.  1.]
 [ 3.  3.  3.  5.]]
Corresponding Value Function is
[[ -6.  -5.  -4.  -3.]
 [ 13.  -4.  13.  -2.]
 [ -4.  -3.  -2.  -1.]
 [ -3.  -2.  -1.   0.]]
Time taken
0.196836
Best policy with Value Iteration is
[[ 3.  1.  3.  1.]
 [ 7.  1.  7.  1.]
 [ 1.  1.  1.  1.]
 [ 3.  3.  3.  5.]]
Corresponding Value Function is
[[ -6.  -5.  -4.  -3.]
 [ 13.  -4.  13.  -2.]
 [ -4.  -3.  -2.  -1.]
 [ -3.  -2.  -1.   0.]]
Time taken
0.002959
Adityas-MacBook-Air:ml_project adityaj$
```

Figure 4. PI and VI for 2nd Configuration

The parameters taken were: **Reward** → -1 for each step taken, -10 for wall encountered; **No. of episodes** → 2000; **Max steps in an episode** → 200; **Learning rate** → 0.1; **Discount** → 0.7; **Epsilon** → 0.1

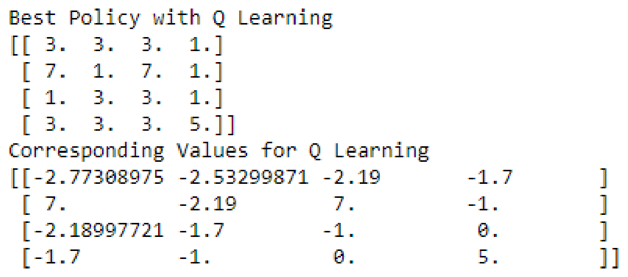


```
Best Policy with Q Learning
[[ 1.  1.  2.  2.]
 [ 3.  1.  7.  7.]
 [ 1.  1.  7.  1.]
 [ 3.  3.  3.  5.]]
Corresponding Values for Q Learning
[[-2.77275726 -2.533      -2.7731     -2.94116941]
 [-2.53299197 -2.19        7.          7.        ]
 [-2.18999965 -1.7         7.          0.        ]
 [-1.7        -1.          0.          5.        ]]
Index for actions:
0 : up
1 : down
2 : left
3 : right
5 : terminal states (stay)
7 : wall
```

Figure 5. Q-learning with static obstacles 1st Configuration

4.3. Q Learning for Changing Obstacles

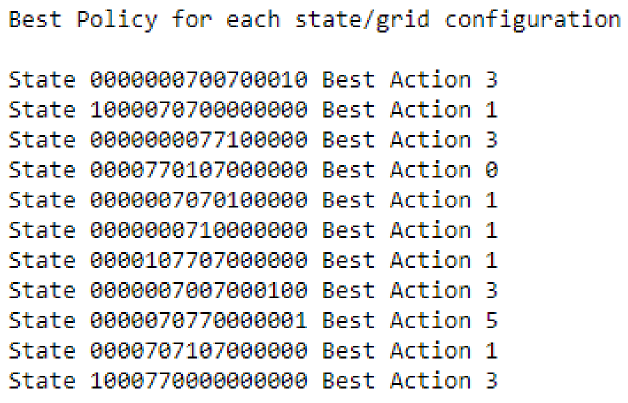
Since the agent has to learn the obstacle distribution during training in different episodes, the entire grid is now the state space for the agent. A sample of the training is shown in **figure 7** (the number besides every state denotes the best action for that agent being in that state).



```
Best Policy with Q Learning
[[ 3.   3.   3.   1.]
 [ 7.   1.   7.   1.]
 [ 1.   3.   3.   1.]
 [ 3.   3.   3.   5.]]
Corresponding Values for Q Learning
[[-2.77308975 -2.53299871 -2.19       -1.7        ]
 [ 7.         -2.19        7.         -1.         ]
 [-2.18997721 -1.7        -1.          0.         ]
 [-1.7        -1.          0.          5.         ]]
```

Figure 6. Q-learning with static obstacles 2nd Configuration

The parameter values used during training were: **Reward** → -1 for each step taken, -10 for wall encountered, +100 for goal state, -5 for going out of the grid; **No. of episodes** → 10000; **Max steps in an episode** → 200; **Learning rate** → 1; **Discount** → 0.9;



```
Best Policy for each state/grid configuration

State 0000000700700010 Best Action 3
State 1000070700000000 Best Action 1
State 0000000077100000 Best Action 3
State 0000770107000000 Best Action 0
State 0000007070100000 Best Action 1
State 0000000710000000 Best Action 1
State 0000107707000000 Best Action 1
State 0000007007000100 Best Action 3
State 0000070770000001 Best Action 5
State 0000707107000000 Best Action 1
State 1000770000000000 Best Action 3
```

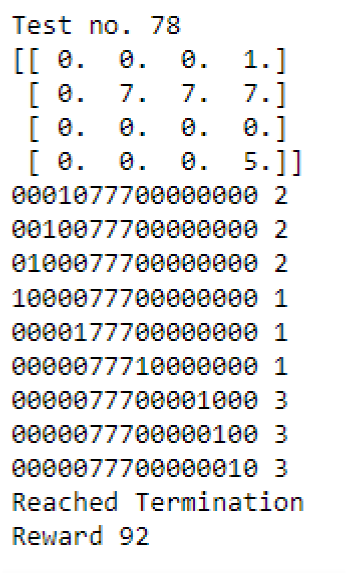
Figure 7. Example state space of the agent

Figures 9 - 11 shows the total number of states learnt by the agent v/s the total number of episodes the agent is trained for.

Figures 12 - 14 exhibits the total number of rewards collected by the agent until it reaches the terminal state with every episode. As seen in the plots, the **rewards become less negative** with each episode.

4.4. Evaluation

To test the model learnt by the agent, we generate 100 test grids with the position of obstacles derived from the same distribution used in training. A random start position is chosen in the test grid and a start state is generated. This state is now searched in the state space learnt by the agent and the corresponding best action. The action takes place giving us the new state. This process is repeated until the agent reaches the terminal state. If the agent is able to reach the terminal state using the state-action sequences learnt, it is considered a success. **Figure 8** denotes a sample test:



```
Test no. 78
[[ 0.  0.  0.  1.]
 [ 0.  7.  7.  7.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  5.]]
0001077700000000 2
0010077700000000 2
0100077700000000 2
1000077700000000 1
0000177700000000 1
0000077710000000 1
0000077700001000 3
0000077700000100 3
0000077700000010 3
Reached Termination
Reward 92
```

Figure 8. Run on a test grid

Accuracy - **100%** i.e. the agent was able to reach the terminal state in all the test grids

5. Contributions

Both the team members have contributed equally in the design, implementation and evaluation of the problem statement. All the reinforcement learning methods and environment have been coded from scratch.

6. File Descriptions

1. **GridWorldDP.py:** Value and Policy Iteration with static obstacles
2. **Q_Learning_StaticObstacles.py:** Q Learning with static obstacles
3. **Q_Learning_MovingObstacles.py:** Q Learning with changing obstacles
4. **BFS.m:** BFS for comparison with DP
5. **Qresult_MovingObs_Final.pickle:** Saved model (Q values) for training

References

[1] K. C. Aram Galstyan and K. Lerman. Resource allocation in the grid using reinforcement learning, 2004.

[2] L. M. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem, 1995.

[3] P. Z. Jun Wu, Xin Xu and C. Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing, 2010.

[4] R. Sutton and A. Barto. Reinforcement learning: An introduction, 2014.

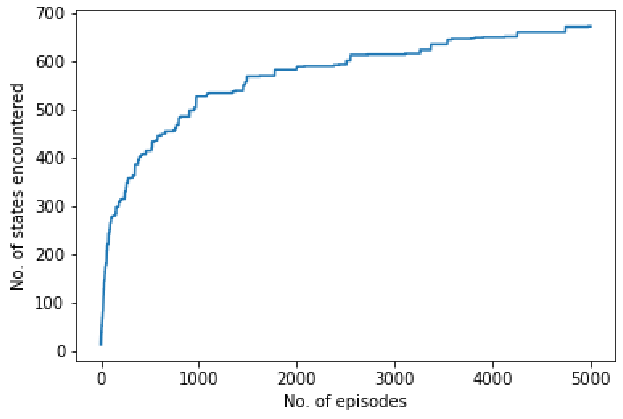


Figure 9. State spaces learnt for 5000 episodes

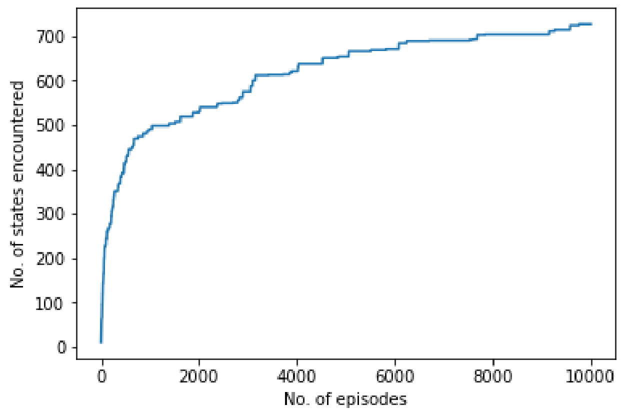


Figure 10. State spaces learnt for 10000 episodes

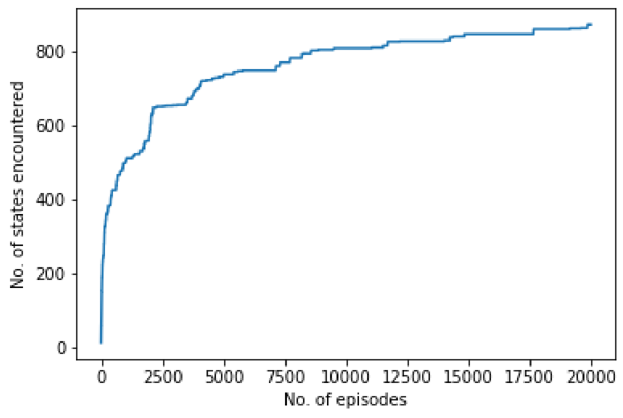


Figure 11. State spaces learnt for 20000 episodes

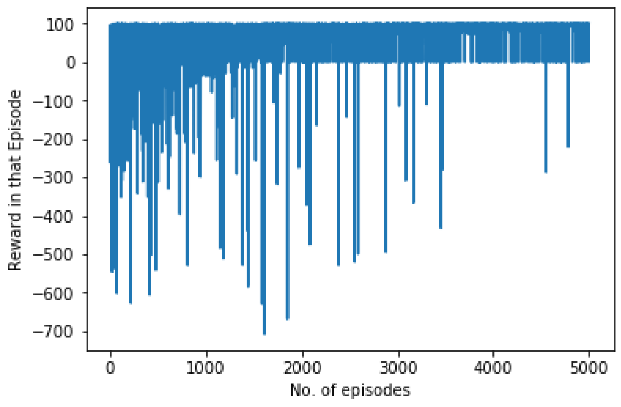


Figure 12. Total rewards for 5000 episode runs

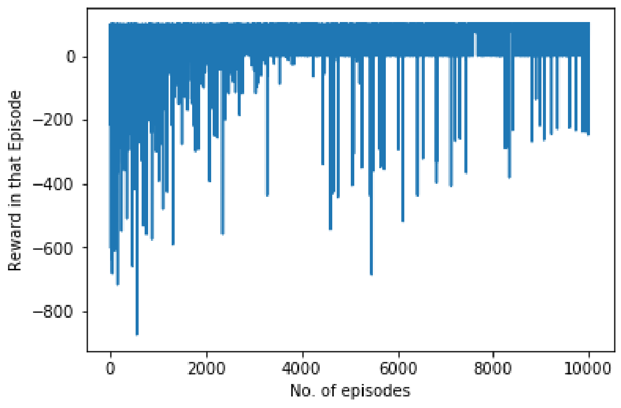


Figure 13. Total rewards for 10000 episode runs

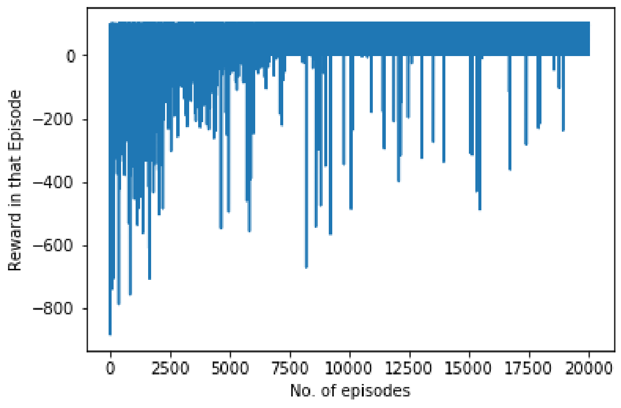


Figure 14. Total rewards for 10000 episode runs