MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT

ANALOGY OF REINFORCEMENT LEARNING

Sachin Gupta

May 6th, 2018

Abstract

In this project, we have done a comparative study of conventional Table based Q learning algorithm with Deep Q learning algorithm. The aim was to compare and contrast the performance of various algorithms on similar environment. We have used 'OpenAI Gym' environments like Cartpole and Mountain Car to do a comparative study of Deep Q learning and conventional Q learning algorithm. The results show that where conventional Q learning algorithms couldn't even achieve satisfactory results, Deep Q algorithms achieved almost perfect score in the same game environment.

Introduction

Reinforcement Learning is an area of machine learning, where an agent learns by interacting with its environment to achieve an objective. In the conventional method the model uses a Q table to store and retrieve the actions for every state based on the rewards earned by the model in the past. This lookup Q table helps the model decide the future course of action. Though this model can be efficient for small state space systems, a system with large number of state-action pairs cannot be handled using this conventional method. Also, this method employs a lookup table for the system which makes it limited to that particular system and for each new system a new model has to be designed which makes it inefficient and makes it distant from human-like learning.

In contrast Deep learning employs neural networks which make it closer to human-like learning as a single model can be used to work on different environments without any prior knowledge of the environment. The model learns by itself through the rewards it achieves on taking certain actions. Thus, a single model can be used to work on similar but different environments viz gaming environments that involve maximizing a particular score based on the actions taken on any state. Using Deep Q Learning a single model trained for a particular environment can be made to work on other models too which may have different rules, states and set of action. Thus the model is not limited to state-action pairs and is flexible to work on different environments too without much modification.

1. Objective

1. To study Reinforcement learning and in particular Q Learning
2. To study about conventional Q learning algorithms and to implement the algorithms on OpenAI gym environments
3. To study about Deep Q learning algorithms and to implement the algorithm on OpenAI gym environments
4. To compare and contrast the performance and results obtained using the two different methods.

Algorithms

Reinforcement Learning

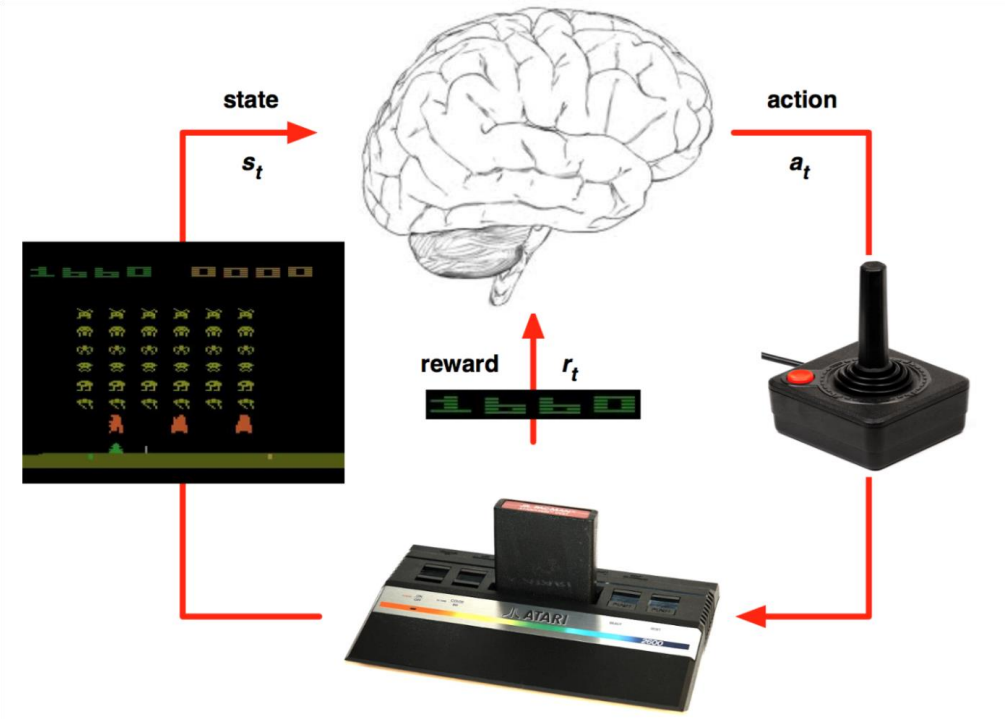


Figure 1: Reinforcement Learning

Reinforcement Learning is a type of machine learning that allows you to create AI agents that learn from the environment by interacting with it. Reinforcement learning provides the capacity for us not only to teach an artificial agent how to act, but to allow it to learn through it's own interactions with an environment.

One of the most important aspects of reinforcement learning is that it takes into account that taking an action does not necessarily only affect the immediate reward, but it may have an impact in the future rewards as well. This way, taking what seems to be the best action now may not be the best decision in the long term. This concept is known as **delayed reward**.

Another of the key aspects of reinforcement learning is that in order to maximize the reward we must follow what we have learned is best, but in order to learn that we have to also explore randomly sometimes. This is called the **exploration-exploitation dilemma**.

Q Learning Algorithm

Q Learning is a popular, **off-policy learning algorithm** that utilizes the **Q function**. It is based on the simple premise that the best policy is the one that selects the action with the highest total future reward.

In a reinforcement learning model, an agent takes actions in an environment with the goal of maximizing a cumulative reward. The basic reinforcement learning model consists of: a set of environment states **S**; a set of actions **A**; rules of transitioning between states; rules that determine the scalar immediate reward of a transition; and rules that describe what the agent observes.

Q-Learning is a **model-free** form of Reinforcement Learning. If **S** is a set of states, **A** is a set of actions, $\boldsymbol{\gamma}$ is the discount factor, is the step size. Then we can understand Q-Learning by this

Algorithm:

Initialize $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$ *arbitrarily*

Repeat (*for each episode*):

> *Initialize* $\boldsymbol{S}$
>
> *Repeat* (*for each step of episode*):
>
> *Choose* $\boldsymbol{a}$ *from* $\boldsymbol{s}$ *using policy derived from* $\boldsymbol{Q}$ ($e.\ g. \in -greedy$)
>
> *Take action* $\boldsymbol{a}$, *observe* $\boldsymbol{r}$, $\boldsymbol{s'}$ $\boldsymbol{Q'}(\boldsymbol{s'}, \boldsymbol{a'}) <-- \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}) + \boldsymbol{\alpha}[\boldsymbol{r} + \boldsymbol{\gamma}. \boldsymbol{max}\ \boldsymbol{Q}(\boldsymbol{s}\ ', \boldsymbol{a}\ ') - \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})]$
>
> $\boldsymbol{s} <-- \boldsymbol{s}\ '$
>
> *until* $\boldsymbol{s}$ *is termina*l

In Q-learning, the value of the Q function for each state is updated iteratively based on the new rewards it receives. At its most basic level, the algorithm looks at the difference between (a) its current estimate of total future reward and (b) the estimate generated from its most recent experience. After it calculates this difference (a - b), it adjusts its current estimate up or down a bit based on the number. Q-learning uses a couple of parameters to allow us to tweak how the process works. The first is the learning rate, represented by the Greek letter α (alpha). This is a number between 0 and 1 that determines the extent to which newly learned information will impact the existing estimates. A low α means that the agent will put less stock into the new information it learns, and a high α means that new information will more quickly override older information. The second parameter is the discount rate, g. The higher the discount rate, the less important future rewards are valued.

Below is the annotated equation for Q-learning:

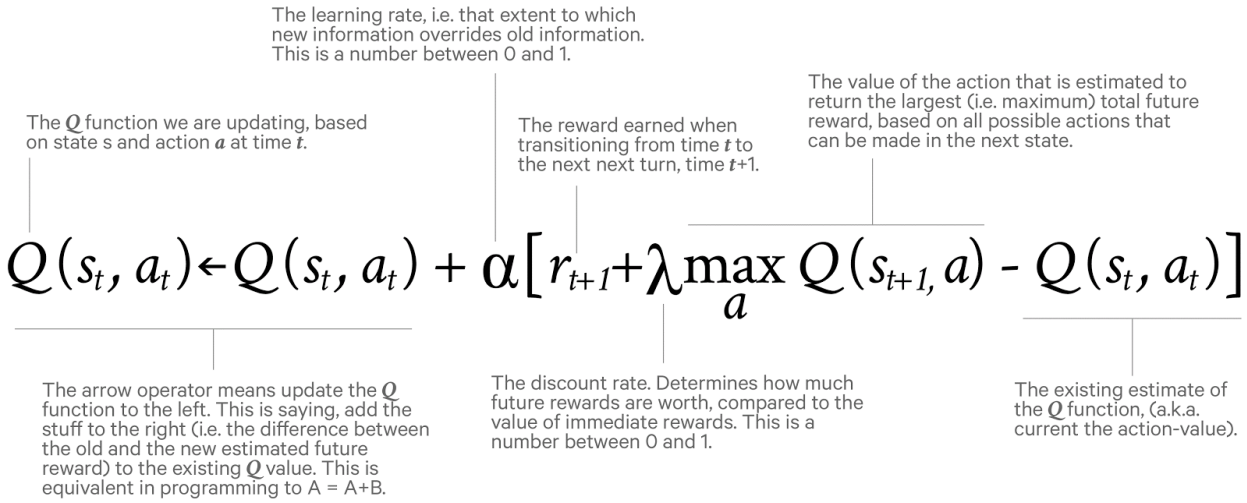


Figure 2: Annotated equation for Q Learning

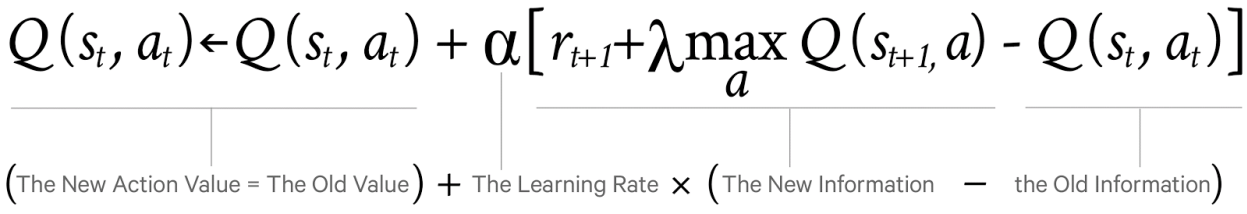


Figure 3: Simplified Annotated equation for Q Learning

Deep Q Learning

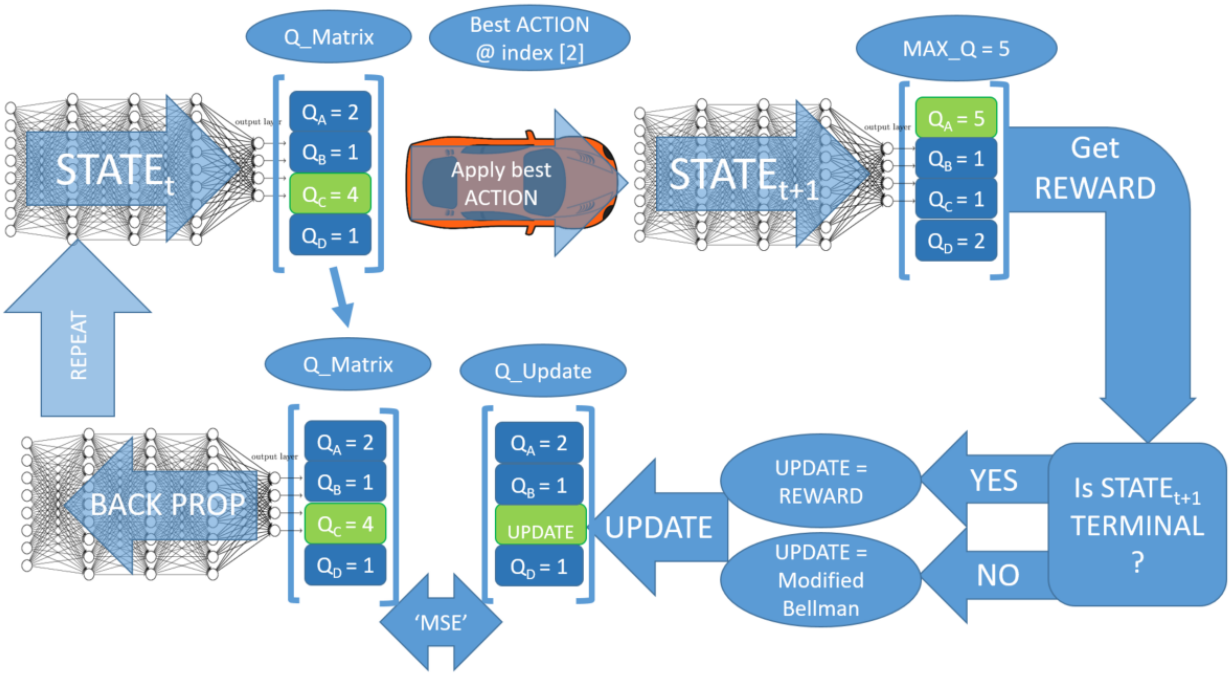


Figure 4: Deep Q Learning Algorithm

Algorithm:

initialize state, epsilon, alpha, epsilon
while training:
  observe state
  feed forward state through NN to get q_matrix
  if (epsilon > random_float_between_0_and_1):
    select random action
  else:
    action_idx = index of the element of q_matrix with largest value
apply action at action
  observe next_state
  observe reward
  if next_state is terminal:
    q_update = reward

```
else:
  feed forward next_state through NN to get q_matrix_next
  observe the largest q value (q_max) in q_matrix_next
  q_update = reward + gamma*q_max

make copy of q_matrix (q_target)
q_target[action_idx] = q_update
loss = MSE(q_matrix, q_update)
optimize NN using loss function
state = next_state
epsilon -= decay_rate
```

Normally in games, the *reward* directly relates to the score of the game. Imagine a situation where the pole from CartPole game is tilted to the right. The expected future reward of pushing right button will then be higher than that of pushing the left button since it could yield higher score of the game as the pole survives longer.

In order to logically represent this intuition and train it, we need to express this as a formula that we can optimize on. The loss is just a value that indicates how far our prediction is from the actual target. For example, the prediction of the model could indicate that it sees more value in pushing the right button when in fact it can gain more reward by pushing the left button. We want to decrease this gap between the prediction and the target (loss). We will define our loss function as follows:

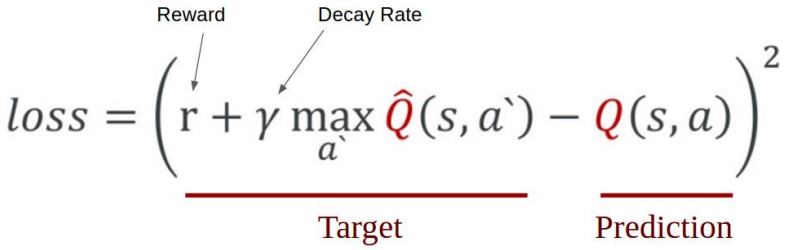


Figure 5: Mathematical representation of loss function

We first carry out an action *a* and observe the reward *r* and resulting new state *s`*. Based on the result, we calculate the maximum target Q and then discount it so that the future reward is worth less than immediate reward (It is a same concept as interest rate for money. Immediate payment always worth more for same amount of money). Lastly, we add the current reward to the discounted future reward to get the target value. Subtracting our current prediction from the target gives the loss. Squaring this value allows us to punish the large loss value more and treat the negative values same as the positive values.

Environments

Cartpole

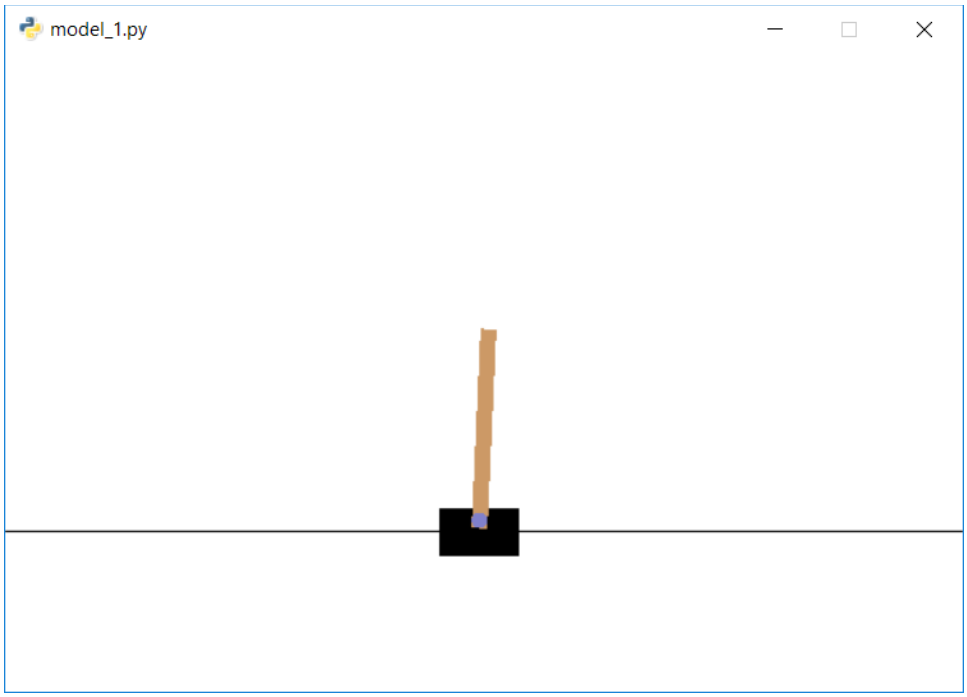


Figure 6: GUI for Cartpole Environment

CartPole is one of the simplest environments in OpenAI gym (a game simulator). The goal of CartPole is to balance a pole connected with one joint on top of a moving cart. Instead of pixel information, there are 4 kinds of information given by the state, such as angle of the pole and position of the cart. An agent can move the cart by performing a series of actions of 0 or 1 to the cart, pushing it left or right.

How the Agent Decides to Act

Our agent will randomly select its action at first by a certain percentage, called ‘exploration rate’ or ‘epsilon’. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward.

In the beginning, the agent explores by acting randomly.
It goes through multiple phases of learning.

1. The cart masters balancing the pole.
2. But goes out of bounds, ending the game.
3. It tries to move away from the bounds when it is too close to them but drops the pole.
4. The cart masters balancing and controlling the pole.

After several hundreds of episodes, it starts to learn how to maximize the score.
The final result is the birth of a skillful CartPole game player!

Cartpole game Environment

TABLE 1: STATE SPACE REPRESENTATION FOR CARTPOLE

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4

1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

TABLE 2: LIST OF VALID ACTIONS FOR CARTPOLE

Num	Action
0	Push cart to the left
1	Push cart to the right

Note: The amount the velocity is reduced or increased is not fixed as it depends on the angle the pole is pointing. This is because the center of gravity of the pole increases the amount of energy needed to move the cart underneath it.

EVALUATION CRITERIA

1. Score (0 to 499) : The number of frames for which the agent manages to balance the pole.

2. Reward : Reward is 1 for every step taken, including the termination step.

3. Average Score: The mean of the scores till the respective episode.

4. Average Reward: The reward of the scores till the respective episode.

STARTING STATE

All observations are assigned a uniform random value between ±0.05

Episode Termination

1. Pole Angle is more than ±12°
2. Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)
3. Episode length reaches 500 frames.

Mountain Car

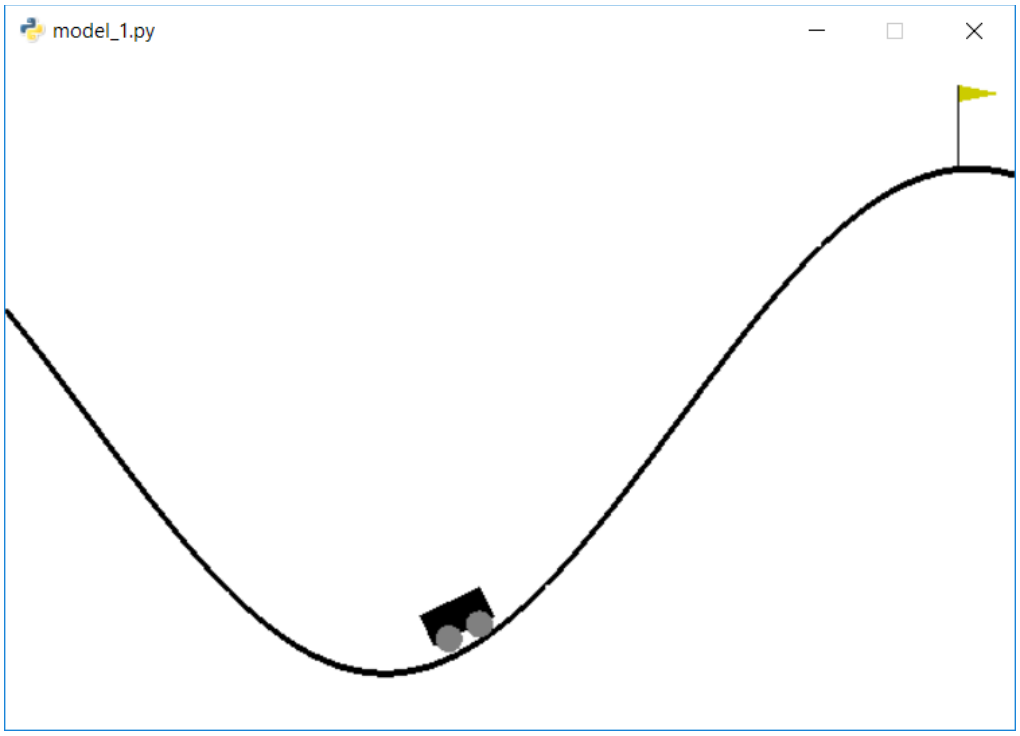


Figure 7: GUI for Mountain Car environment

Mountain Car is a problem in which an underpowered car must drive up a steep hill. 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. Since gravity is stronger than the car's engine, even at full throttle, the car cannot simply accelerate up the steep slope. The car is situated in a valley and must learn to leverage potential energy by driving up the opposite hill before the car is able to make it to the goal at the top of the rightmost hill.

Mountain Car game Environment

TABLE 3: STATE SPACE REPRESENTATION FOR MOUNTAIN CAR

NUM	OBSERVATION	MIN	MAX
0	POSITION	-1.2	0.6
1	VELOCITY	-0.07	0.07

TABLE 4: LIST OF VALID ACTION FOR MOUNTAIN CAR

NUM	OBSERVATION
0	PUSH LEFT
1	NO PUSH
2	PUSH RIGHT

Evaluation Criteria

1. Score: (-1.2 to 0.6) :): The highest coordinate that the agent manages to reach (Goal 0.5).
2. Reward : -1 for each time step, until the goal position of 0.5 is reached.
3. Average Score : The mean of the scores till the respective episode.
4. Average Reward : The mean of the rewards till the respective episode.

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.

Benchmark Model

Since the main objective of this project is to improve the performance of the Table based Q-learning model. It involves a comparative study of Random Exploration model, The Table Based Q learning Model and The Deep Q Learning Model.

Implementation

The entire code for the project has been saved in the repository at: https://github.com/coolsgupta/Udacity_Capstone.git.

```
def _build_model(self):
    # Neural Net for Deep-Q learning Model
    model = Sequential()
    model.add(Dense(24, input_dim=self.state_size, activation='relu'))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(self.action_size, activation='linear'))
    model.compile(loss='mse',
                  optimizer=Adam(lr=self.learning_rate))
    return model
```

Figure 8: The Deep Q Learning Model

```
for e in range(50):
    state = env.reset()
    total_reward = 0
    for time in range(500):
        action = random.randrange(env.action_space.n)
        next_state,reward,done,_ = env.step(action)
        reward = reward if not done else -10
        total_reward += reward
        state = next_state
```

Figure 9: The Random Exploration agent

```
def learn(self, state, action, reward, next_state):
   if self.learning:
       maxQ, maxQ_action = self.get_maxQ(next_state)
       self.Q_table[state][action] = self.Q_table[state][action] + (self.learning_rate*(reward + (self.gamma*maxQ)))
```

Figure 10: The Q Learning agent

Evaluation and Results

Learning Curves

Table Based Q Learning Agent

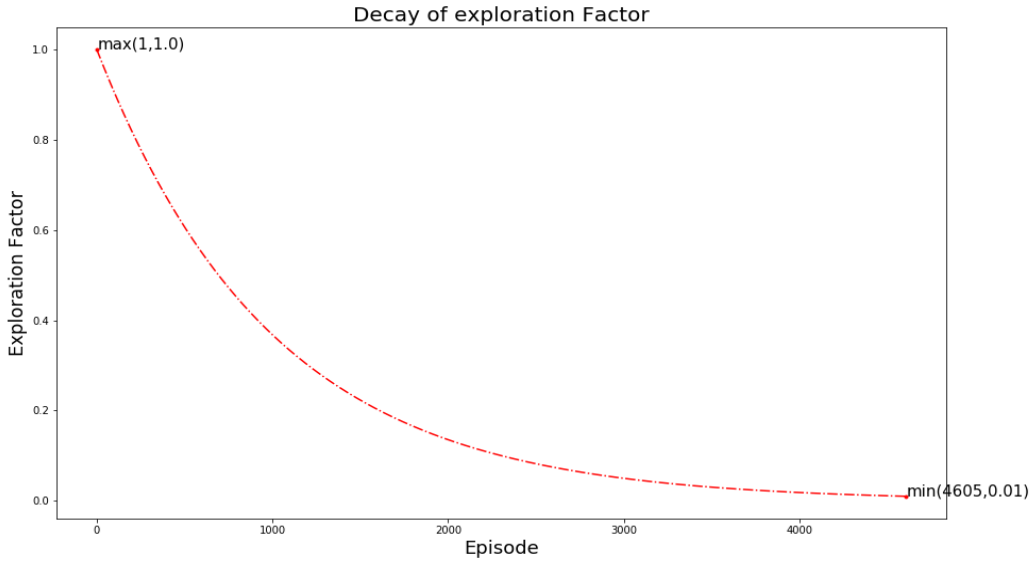


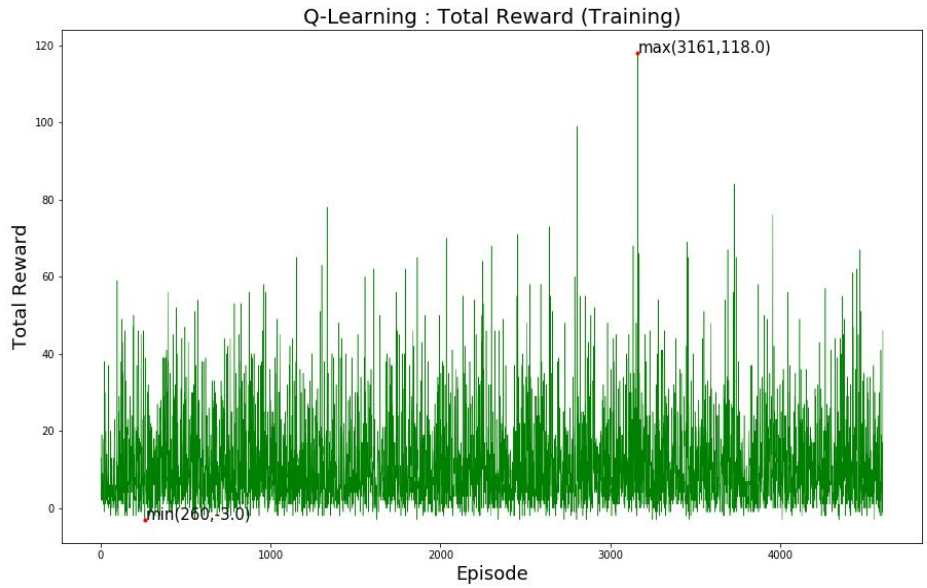
Figure 11: Decay of Exploration Factor (Cartpole and Mountain Car)

The Figure 11 depicts the decay of exploration factor with the increase in the number of trials demanding the agent to take more actions from experience than random exploration. Also, the training terminates when the exploration factor drops to 0.01.

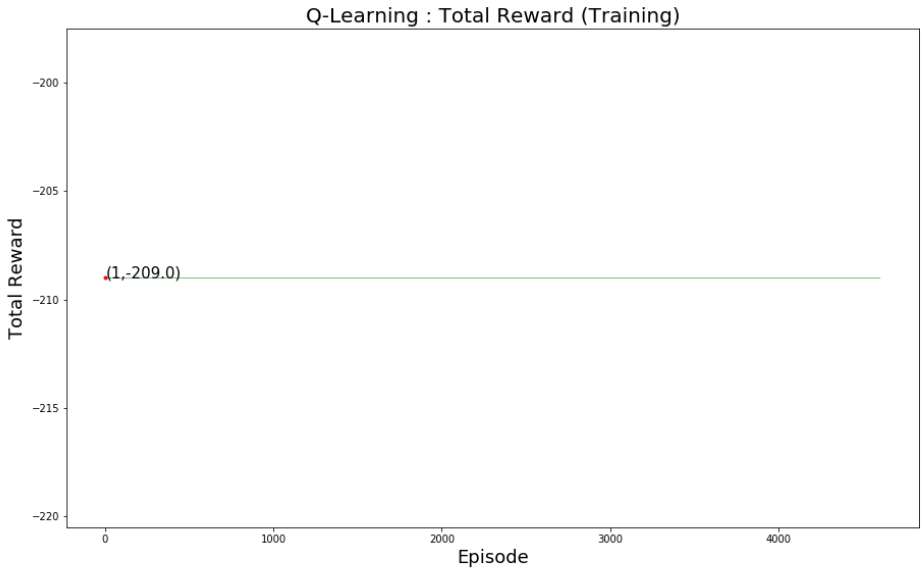
(a): Cartpole

(b): Mountain Car

Figure 12: Score vs Episodes(Q-learning)



(a): Cartpole

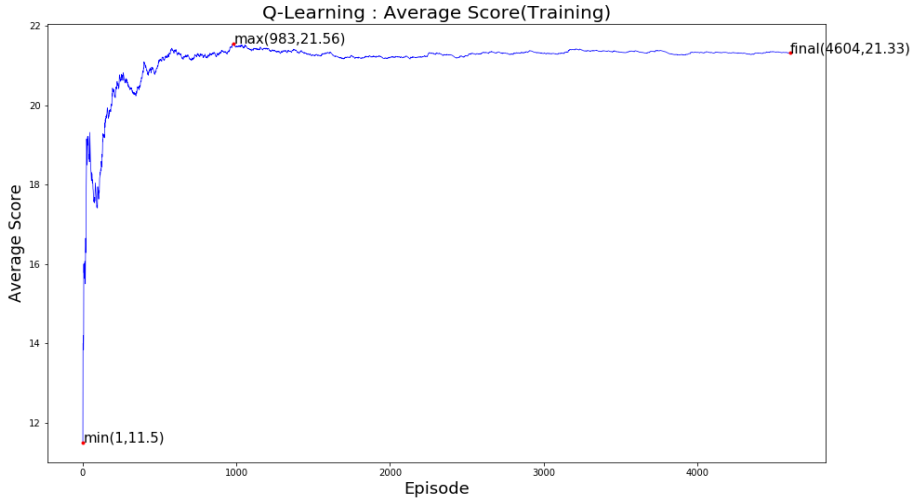


(b): Mountain Car

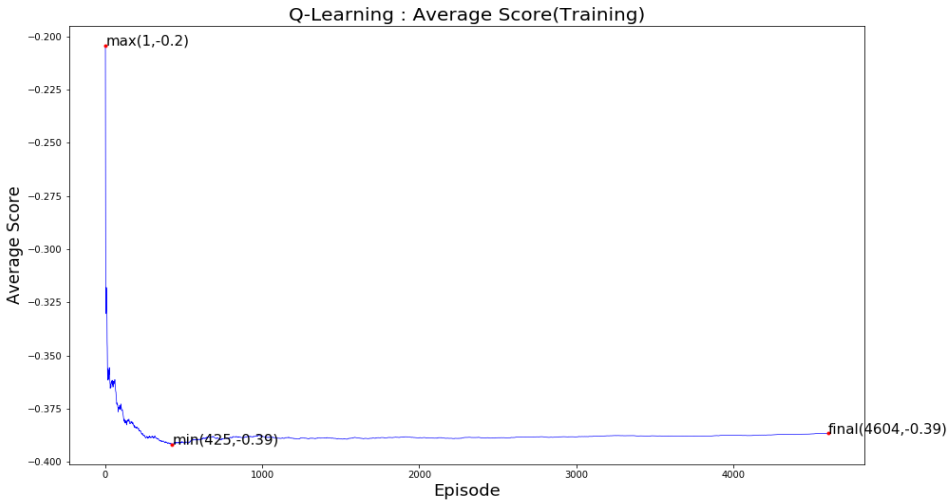
Figure 13: Reward vs Episodes(Q-learning)

The Figure 12 and Figure 13 show the performance of the Q learning agent when training for the cartpole environment (a) and the Mountain Car environment (b), as observed, the agent achieves

a maximum score of 128 and a maximum total reward of 118 in trial number 3161 for Cartpole and a maximum score of -0.03 in trial number 2119 and a constant total reward of -209 (showing that it never reached the goal state) for Mountain Car, making it apparent that the overall performance remains low and random.

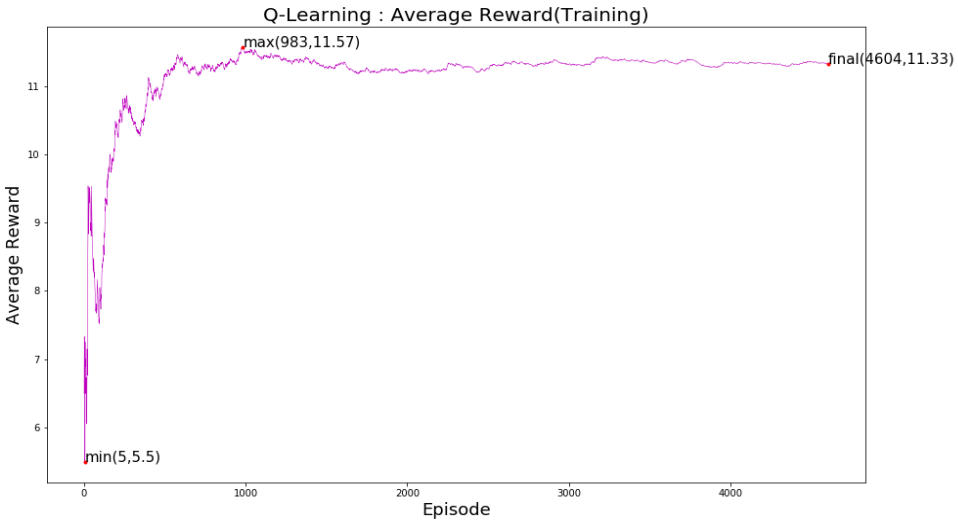


(a): Cartpole

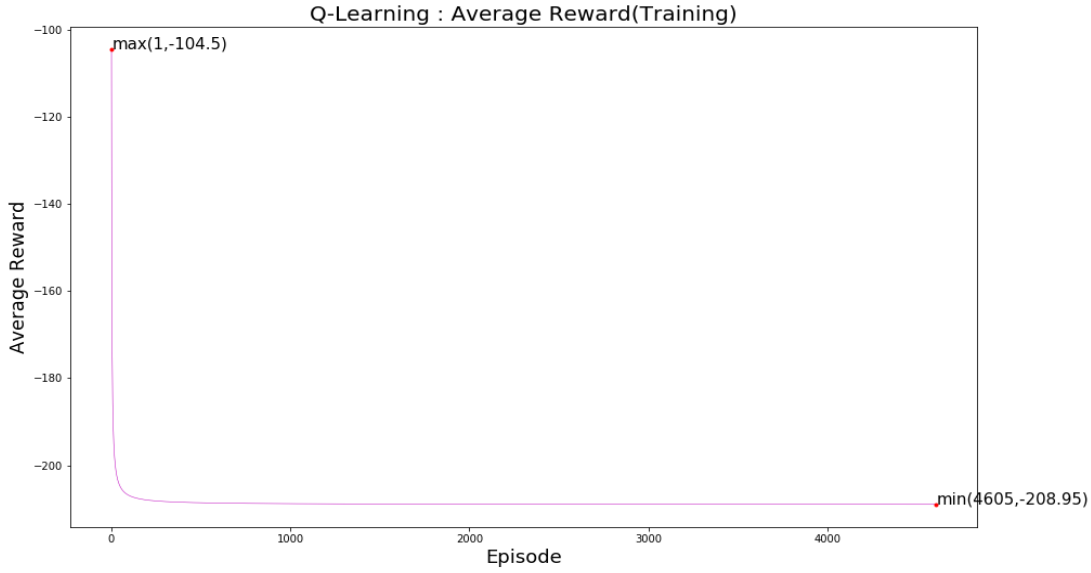


(b): Mountain car

Figure 14: Average Score vs Episodes(Q-learning)



(a) : Cartpole



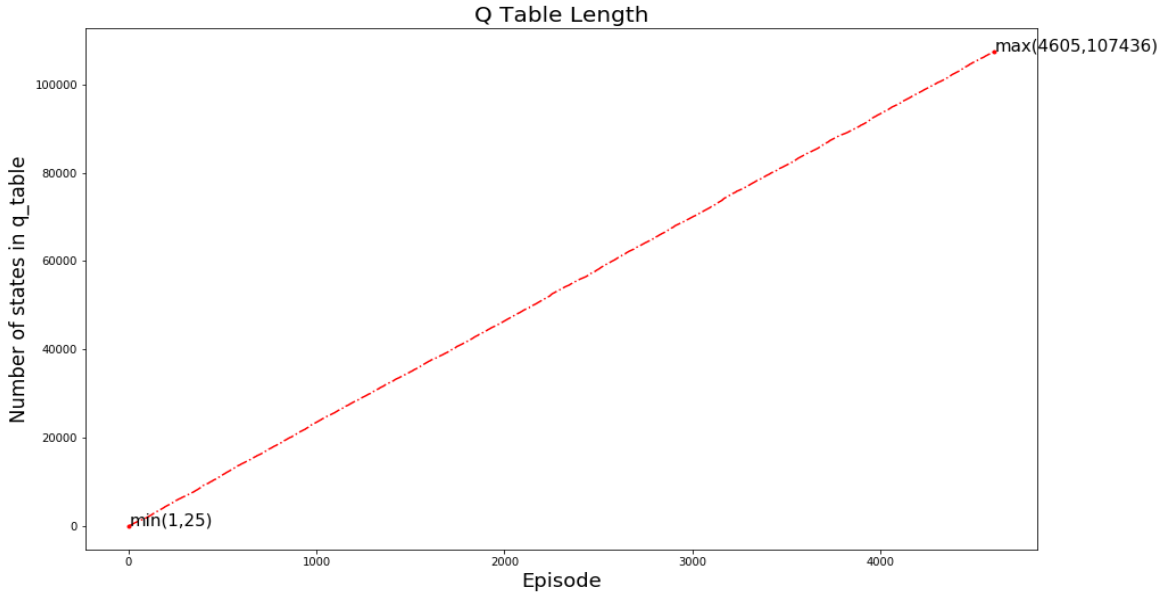
(b) : Mountain car

Figure 15: Average reward vs Episodes(Q-learning)

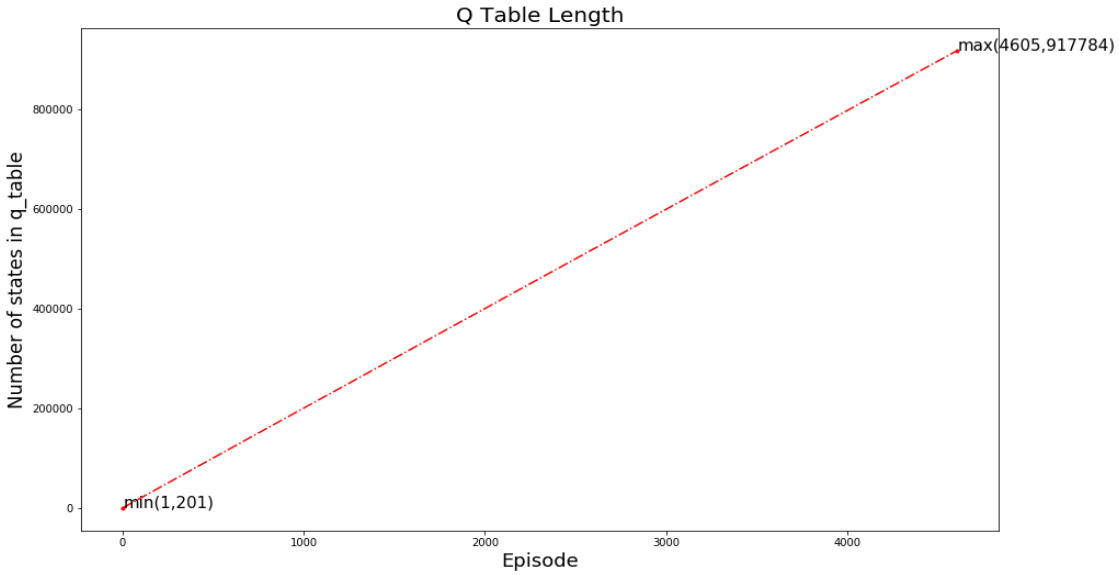
As shown in figure 14 and figure 15 the average score and average rewards per episode of the Q learning agent remain very low as 21.33 and 11.3 respectively for cartpole and -0.39 and -208.95 for Mountain car. Also, they dropped as low as 11.5 and 4.5 and reaching just 21.56 and 11.57 as

the maximum values for Cartpole. For Mountain Car the average score dropped to -0.39 while the average score remained close to -209 showing that the agent never reached the goal state. The cause for the same is shown in the figure 16, i.e. the growth of the Q table.

The Q learning agent when encounters a new state, creates an entry in the Q table and takes a random action for the instance and updates the Q values of the actions for that state accordingly so that it can refer to the same in order to possibly take a better action when it encounters the same state again. This leads to the problem known as the growth rate of the Q table. The same can be observed in figure 16 as it shows that by end of trial number 4605 the table already consisted of 107436 states for Cartpole and 917784 for Mountain Car and yet it can be observed from the slope that it would certainly increase further if subjected to more number of trials.



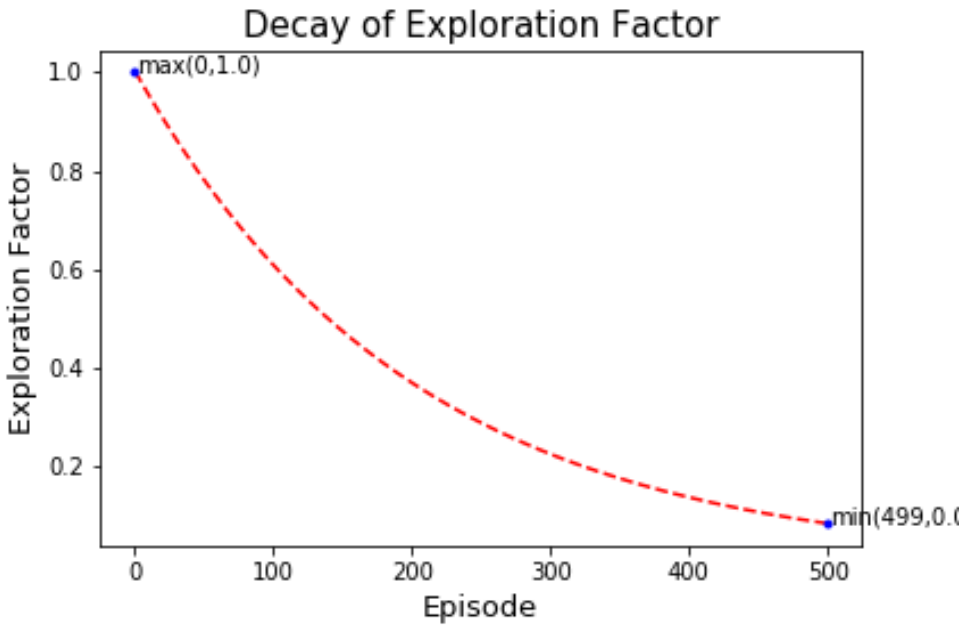
(a) : Cartpole



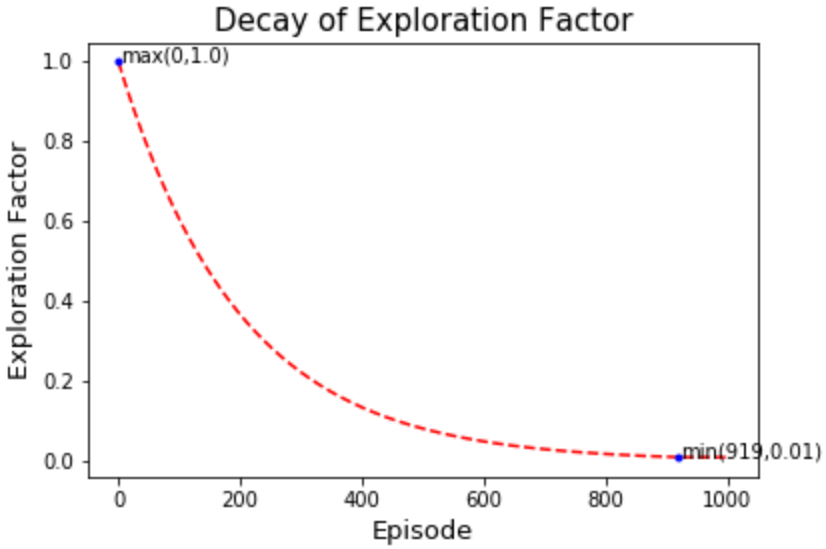
(b) : Mountain Car

Figure 16: Growth of the Q table(Q-learning)

4.2.2 Deep-Q-Learning Agent

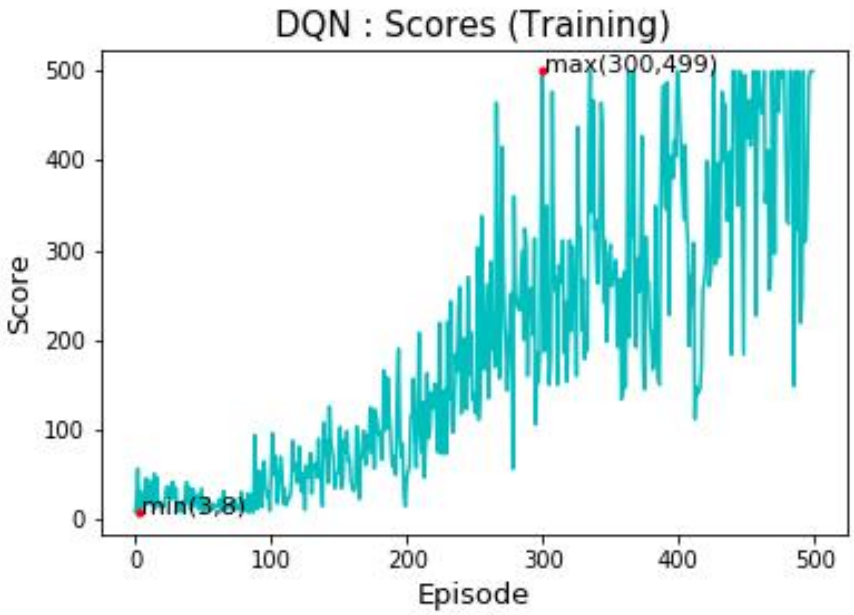


(a) : Cartpole

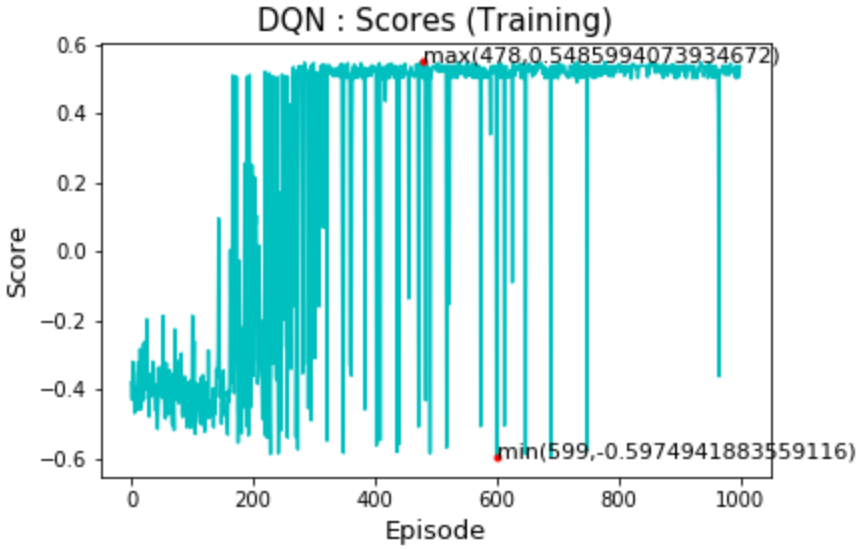


(b) : Mountain Car

Figure 17: Epsilon Decay rate(DQN)

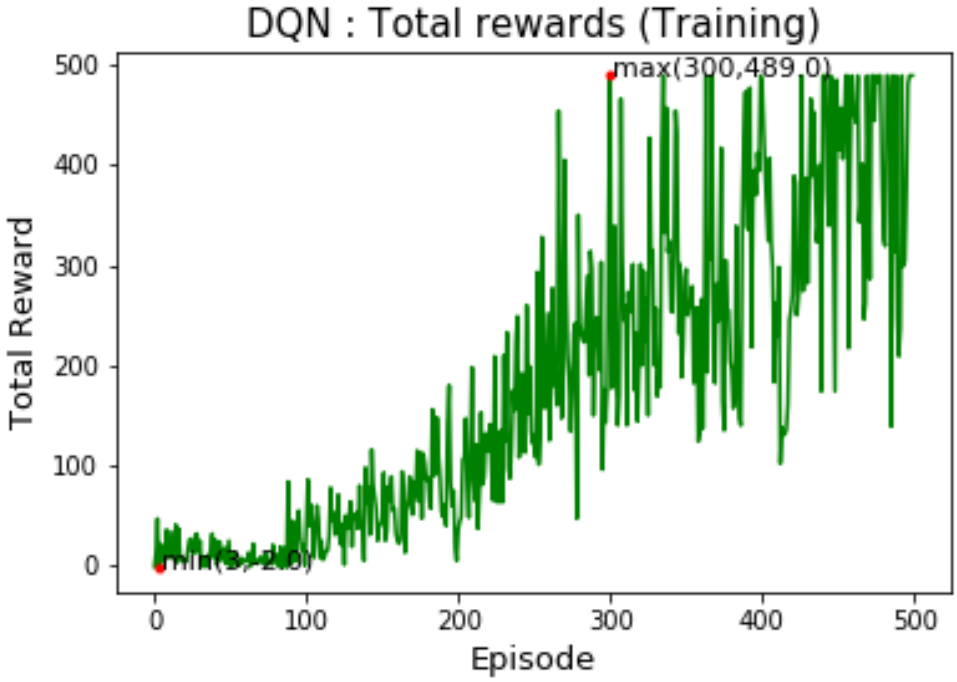


(a) : Cartpole

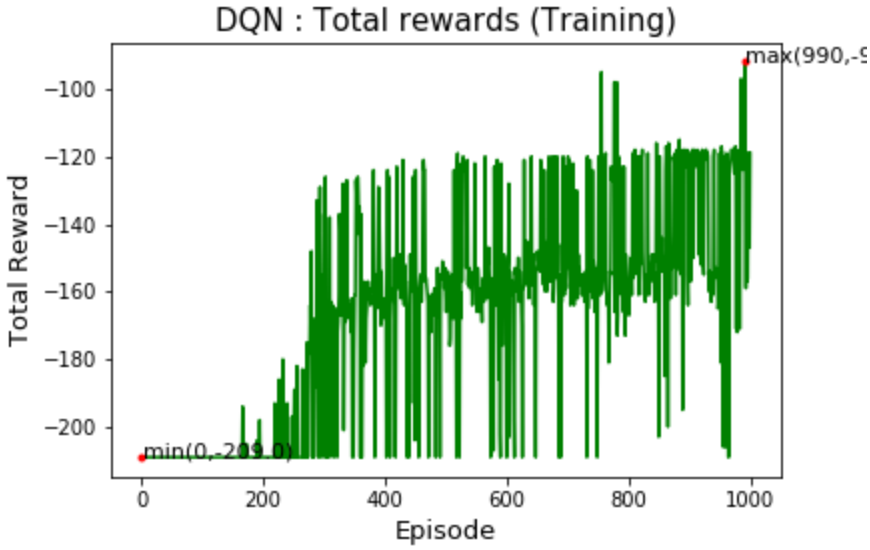


(b) : Mountain Car

Figure 18: Score vs Episode (DQN)



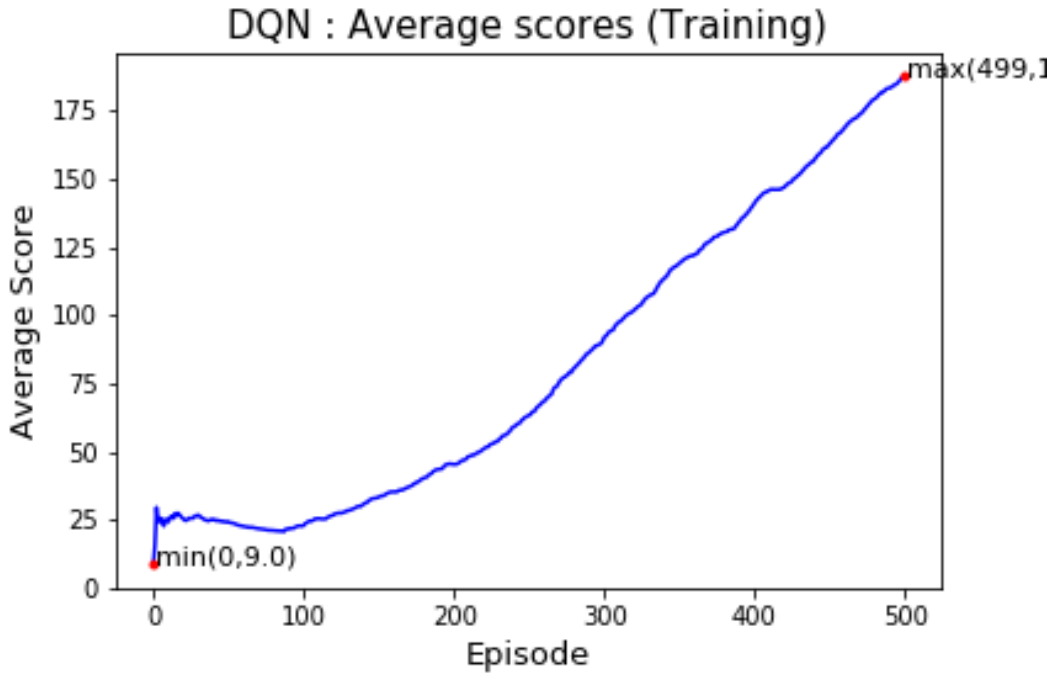
(a) : Cartpole



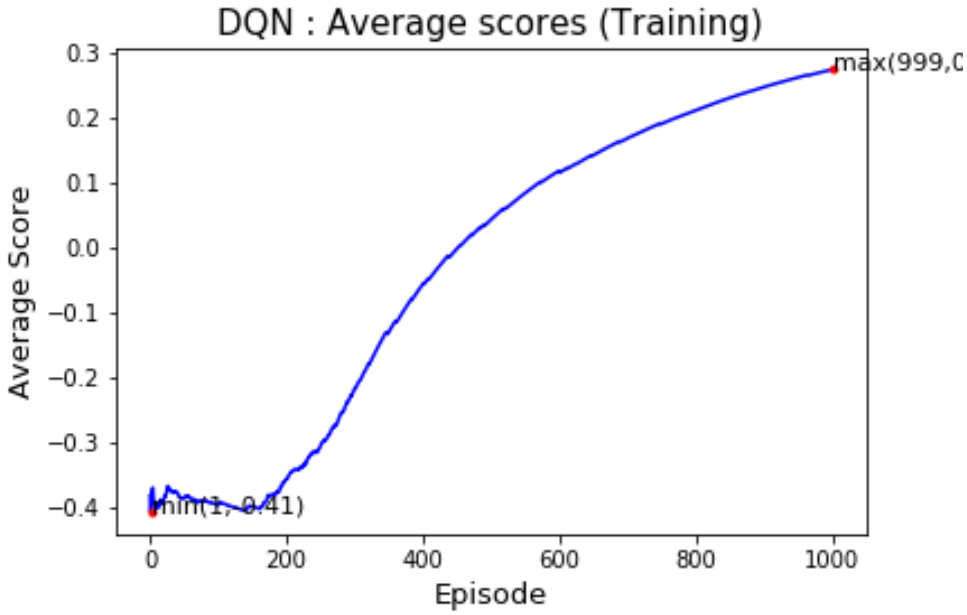
(b) : Mountain car

Figure 19: Reward vs Episode (DQN)

Figure 18 and Figure 19 show the performance of the Deep Q learning agent during training. The agent shows apparent growth in the score and total reward and manages to attain maximum score of 499 (perfect score for balancing the pole for all 500 frames) and maximum reward of 489 for Cartpole and a maximum score of 0.54 (goal state) and a maximum reward of -92.0 in the later trials. The agent manages to achieve the same in just 500 trials for cartpole and 100 trials for Mountain Car attaining maximum scores in nearly 300 trials.

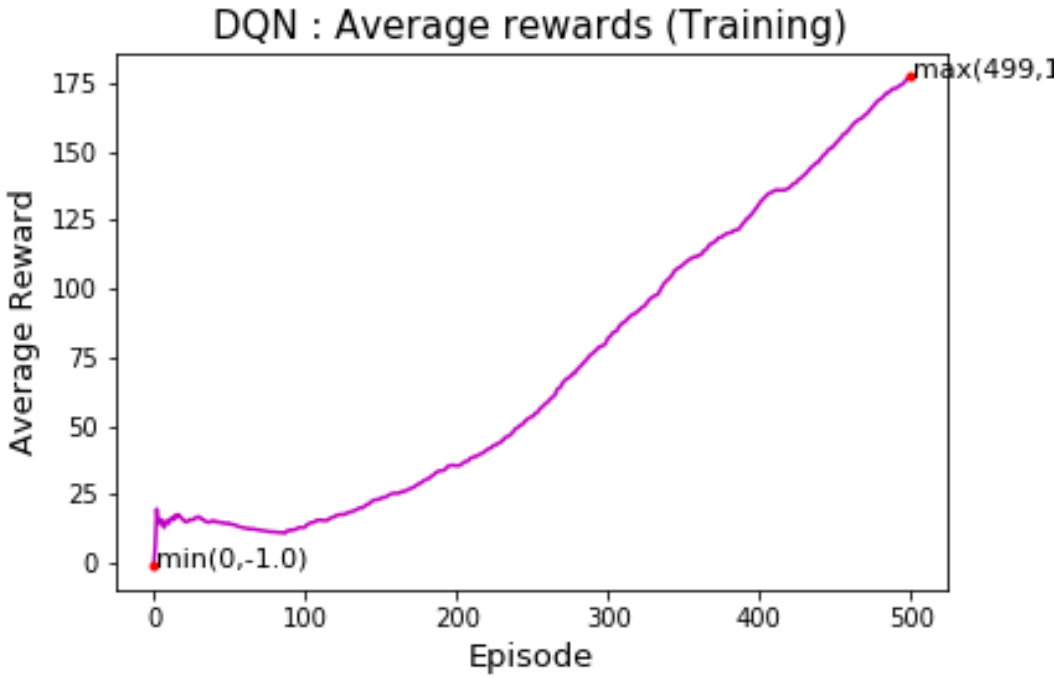


(a) : Cartpole

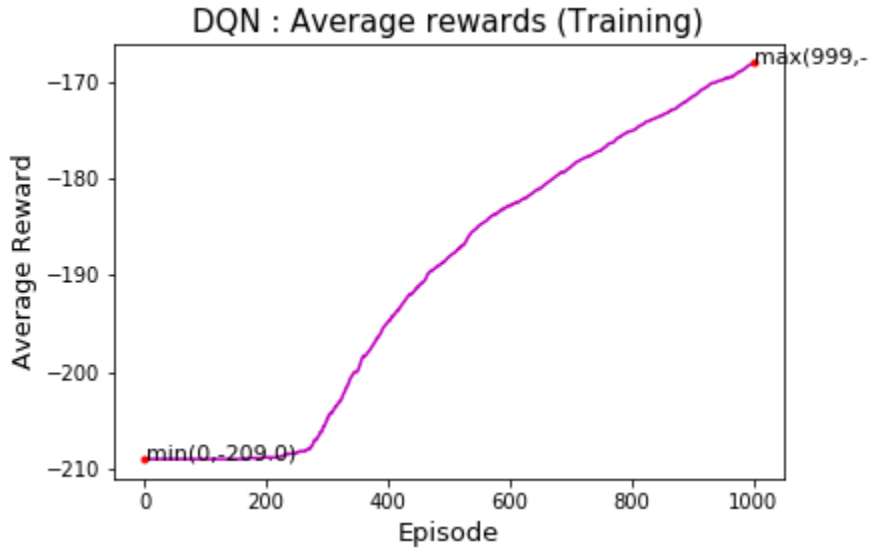


(b) : Mountain Car

Figure 20: Average Score vs Episode (DQN)



(a) : Cartpole



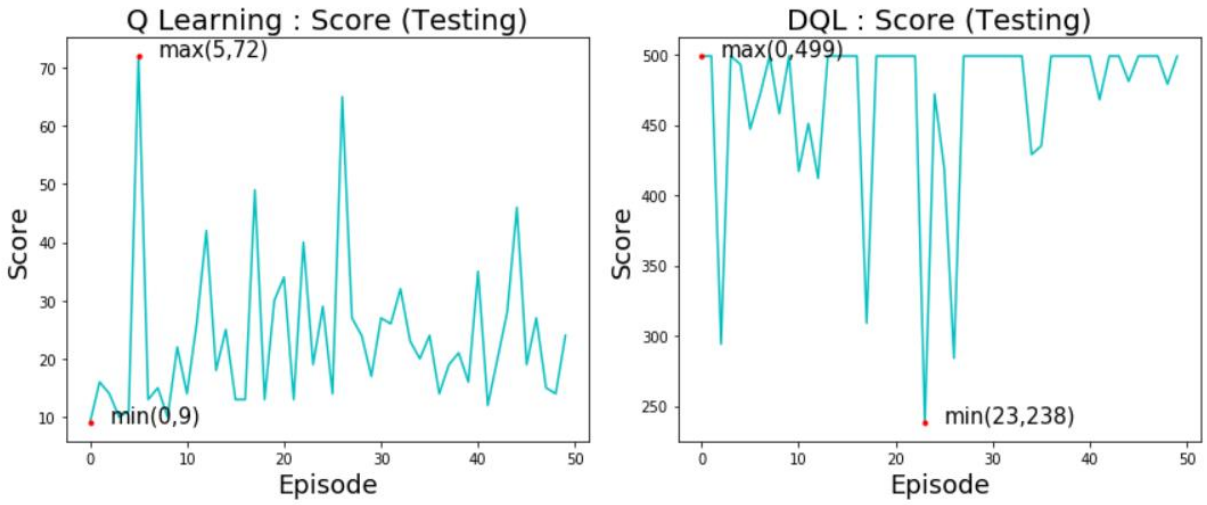
(b) : Mountain Car

Figure 21: Average Reward vs Episode (DQN)

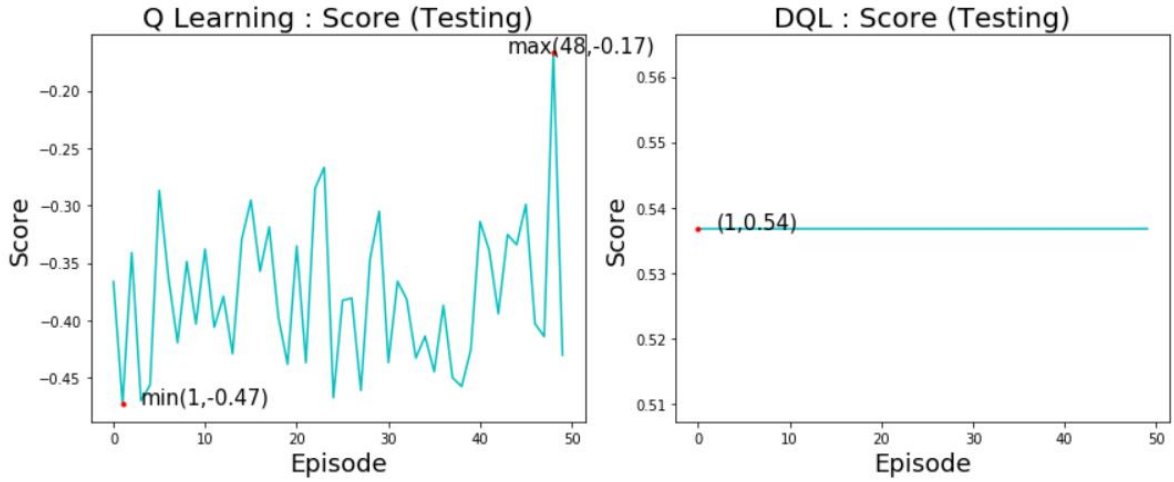
The Figure 20 and Figure 21 show the growth in the average score and average reward depicting the success of the agent to learn the patterns and policies of the environment.

The best part of the process being the non-requirement of the Q table as the agent updates the weights of the underlying neural network as per the experienced gained which result in better prediction of the Q values of the actions associated with all the states that the agent may encounter, unlike Q learning where only the value of the visited state was being updated leaving the other states unaffected. Hence enabling the agent to take better decision for both known and unprecedented states.

4.2.3 Testing Curves (Q-Learning Agent vs DQN Agent)

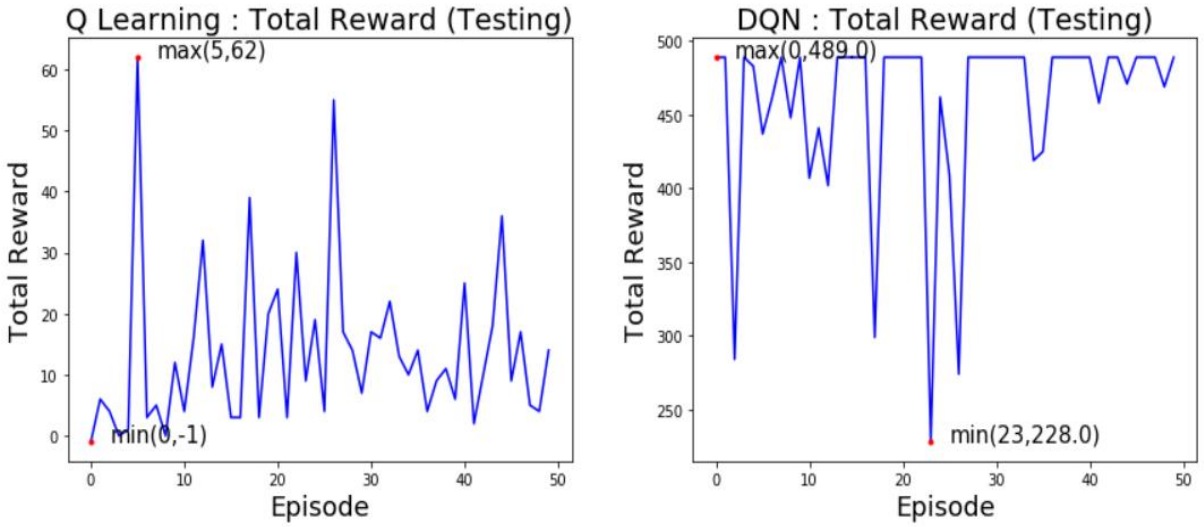


(a) : Cartpole

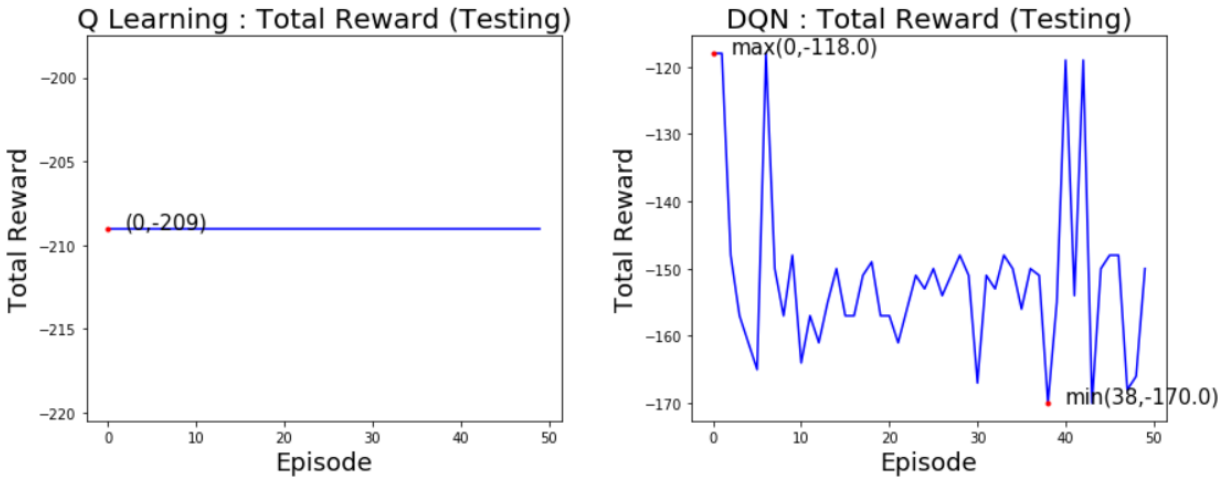


(b): Mountain Car

Figure 22: Score vs Episode (Q-learning vs DQN)

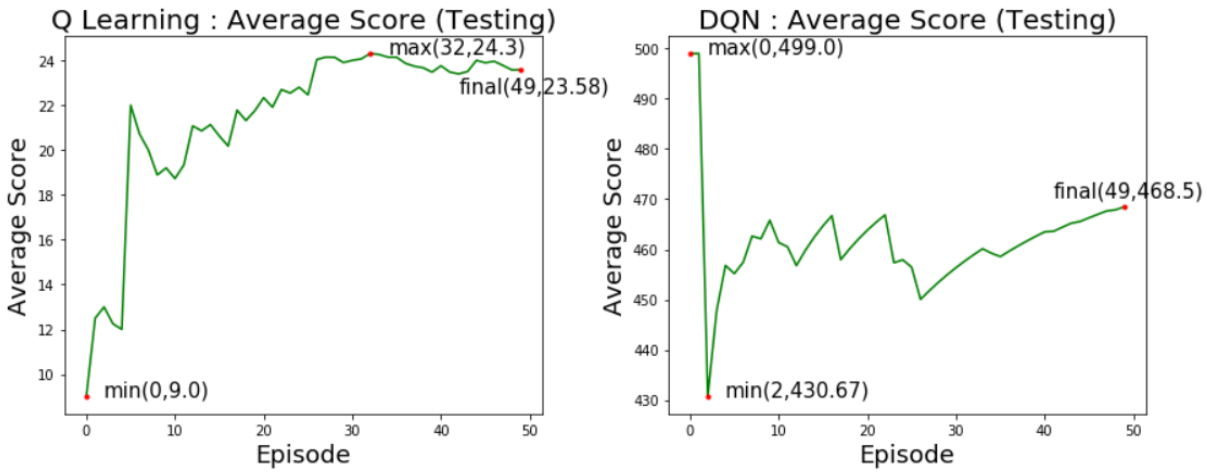


(a) : Cartpole

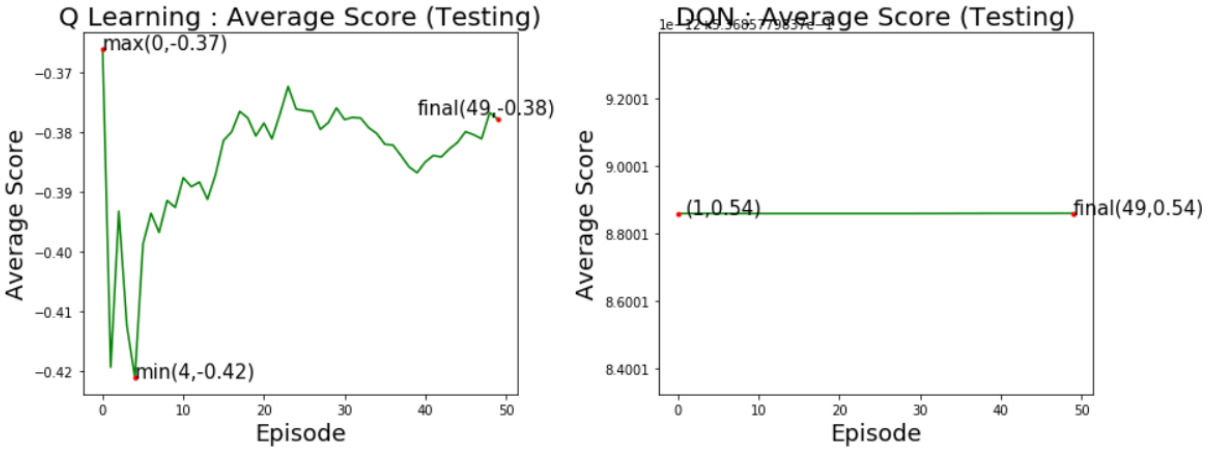


(b) : Mountain Car

Figure 23: Reward vs Episode (Q-learning vs DQN)

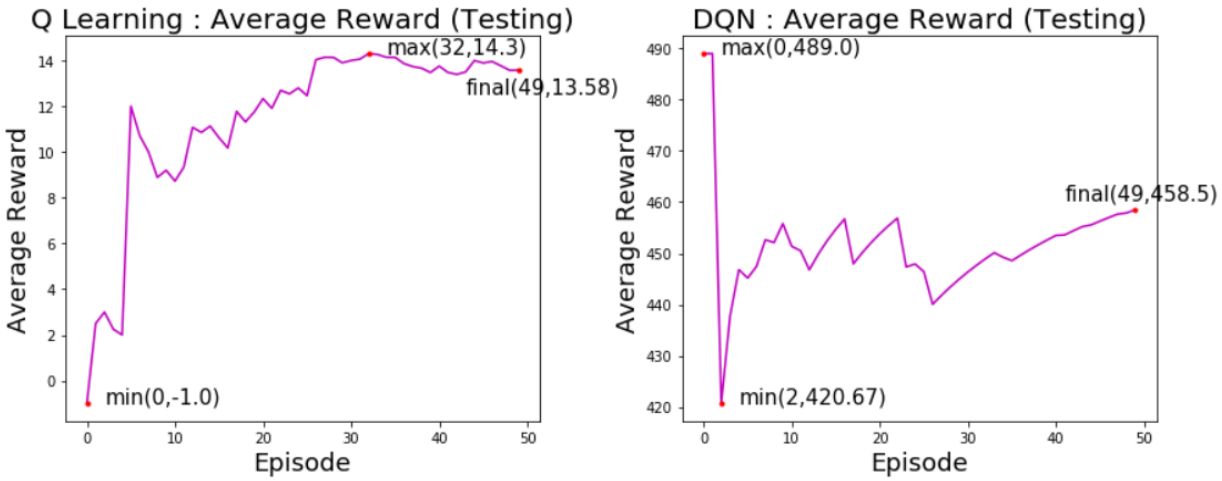


(a) : Cartpole

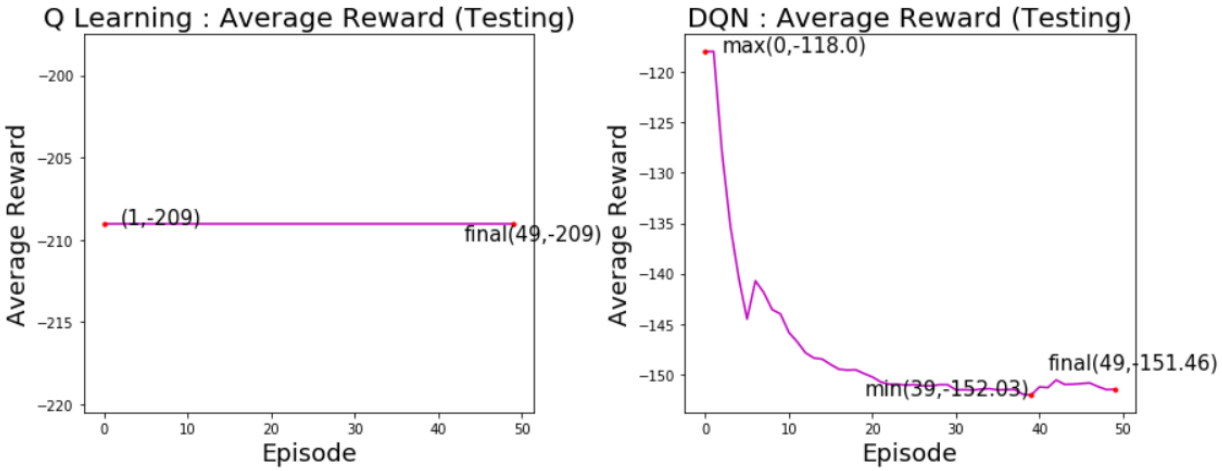


(b) : Mountain Car

Figure 24: Average Score vs Episode (Q-learning vs DQN)



(a) : Cartpole



(b) : Mountain Car

Figure 25: Average Reward vs Episode (Q-learning vs DQN)

The results from the testing trials confirm the inferences drawn from the training results showing that the Deep Q learning agent clearly outperforms the Q learning agent.

For Cartpole, The Q learning agent manages to achieve a maximum value of 72 and 62 for score and total reward respectively, attaining a final value of 23.58 and 13.58 for average score and average reward. These values however dropped as low as 9, -1, 9 and -0.42 for the four metrics

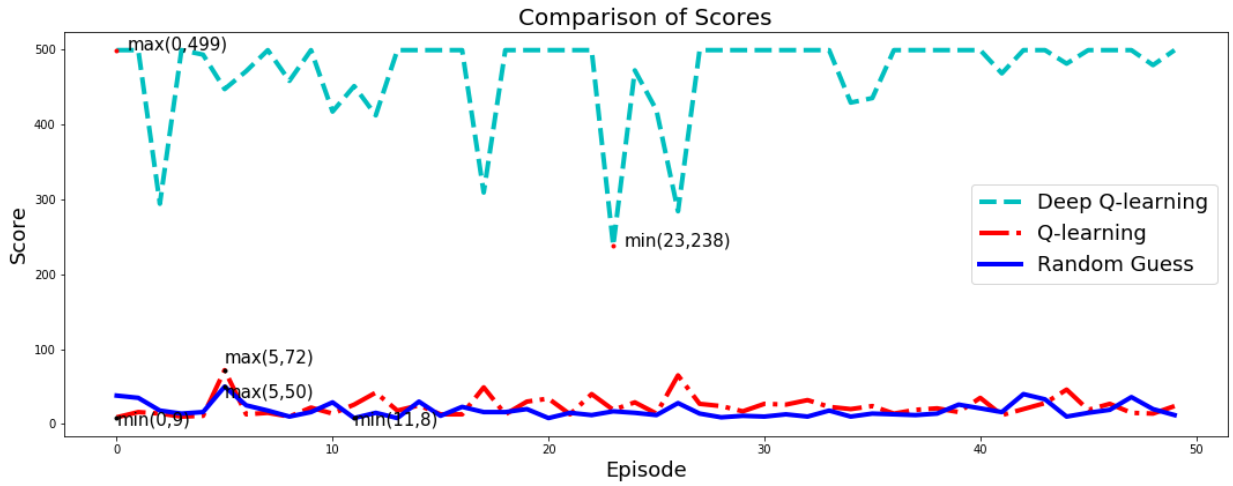
respectively. Hence it is clear that the Q learning agent was incapable for taking good decisions probably due to the occurrence of unprecedented states.

However, the Deep Q learning agent manages to attain 499 and 489 as maximum values of score and total reward respectively. The values did drop to 238, 228, 430.67 and 420.67 respectively for the four evaluation metrics but the averages reached a final value of 468.5 and 458.5 showing the appreciable performance of the agent.

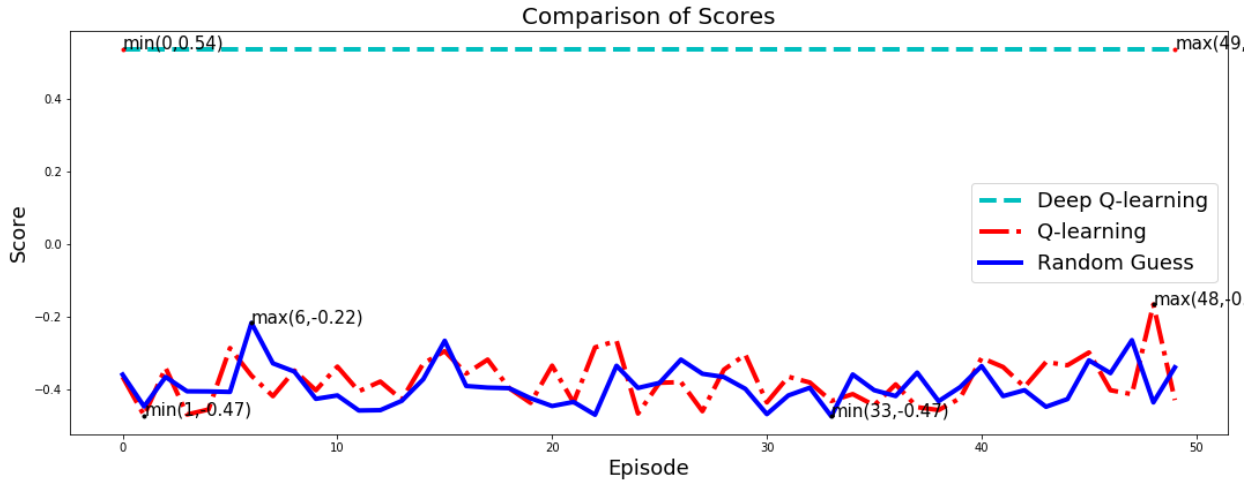
Similarly, for Mountain Car, the Q learning agent The Q learning agent manages to achieve a maximum value of -0.17 for score with the average score attaining a final value of 23.58 but the total reward and average reward remain constant at -209 showing that the agent never manages to reach the goal state, showing that the Q learning agent was incapable for taking good decisions probably due to the occurrence of unprecedented states.

However, the Deep Q learning agent manages to attain a constant score and average score of 0.54 showing that it managed to reach the goal state in all the testing trials performed attaining a maximum total reward of -118 and a final average reward of -151.46 showing the appreciable performance of the agent.

4.3 Random Exploration vs Q-Learning Agent vs DQN-Agent.

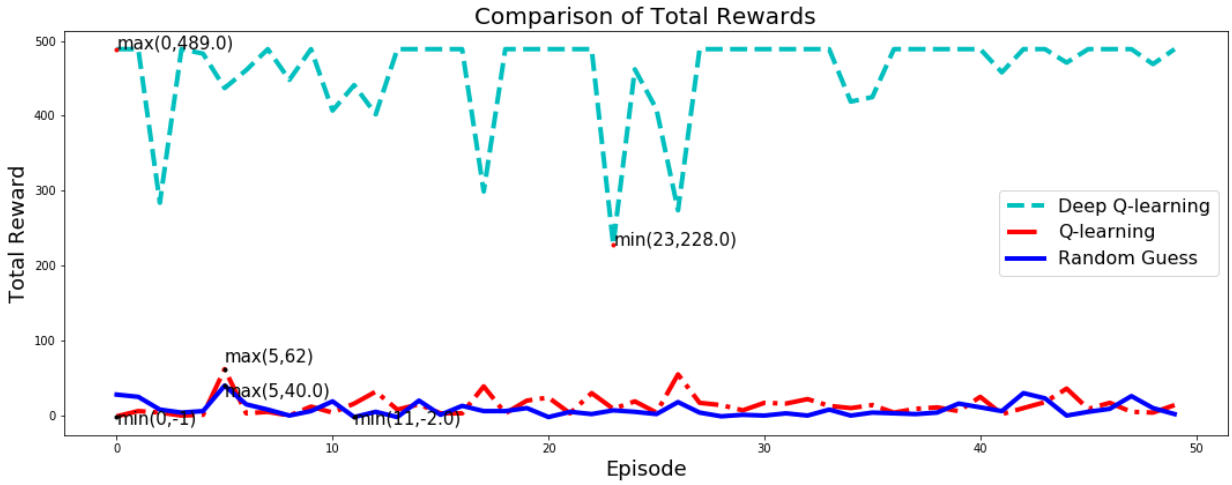


(a) : Cartpole

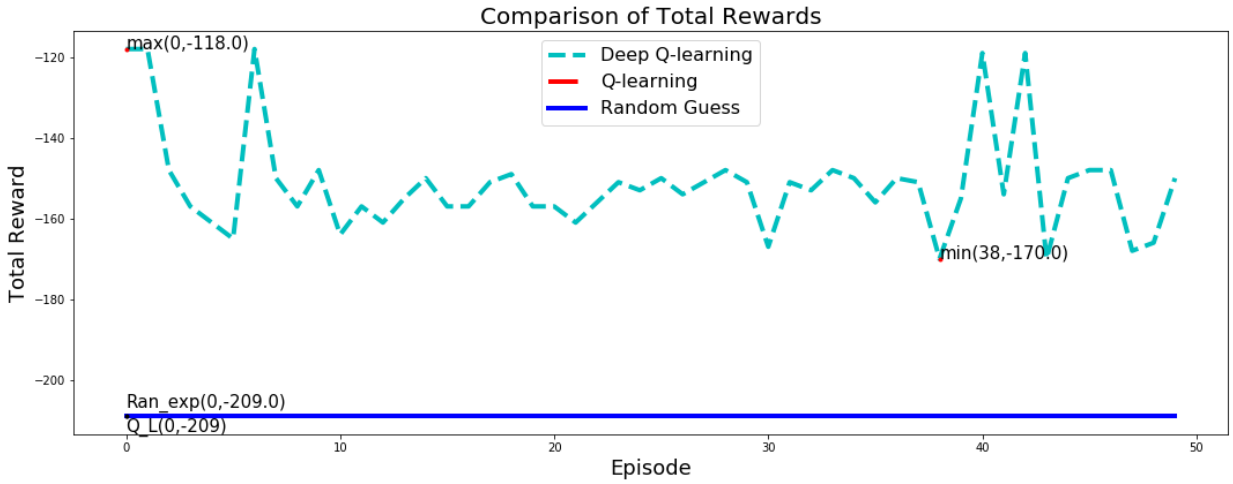


(b) : Mountain Car

Figure 26: Score vs Episode (R.E vs Q-learning vs DQN)

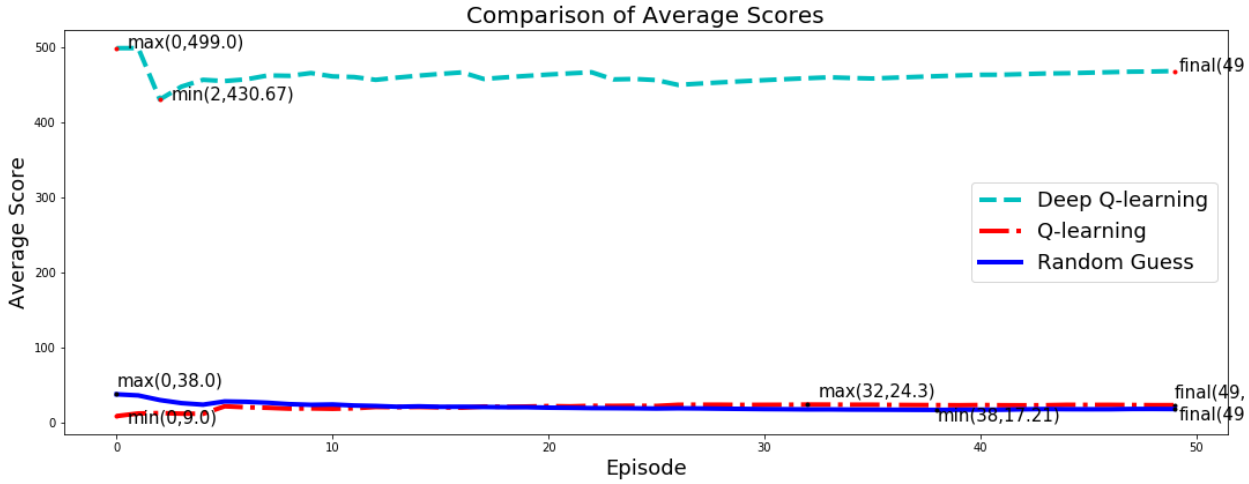


(a) : Cartpole

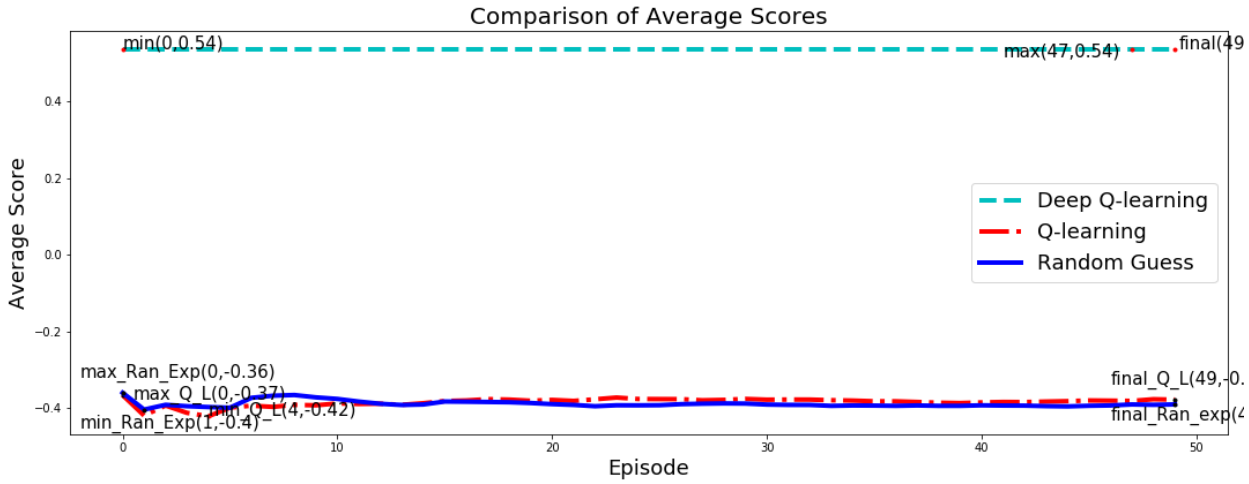


(b) : Mountain Car

Figure 27: Reward vs Episode (R.E vs Q-learning vs DQN)

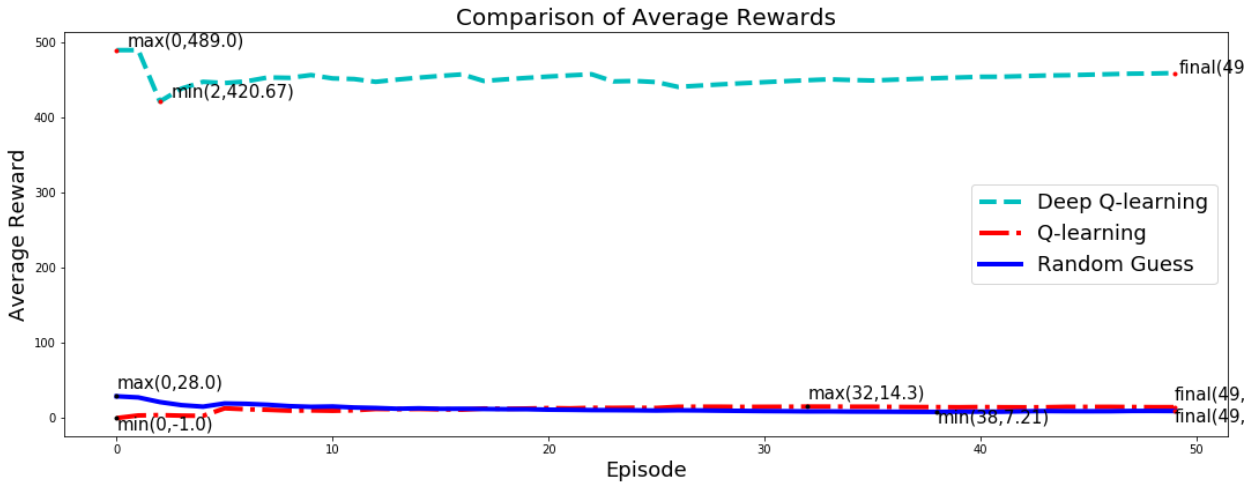


(a) : Cartpole

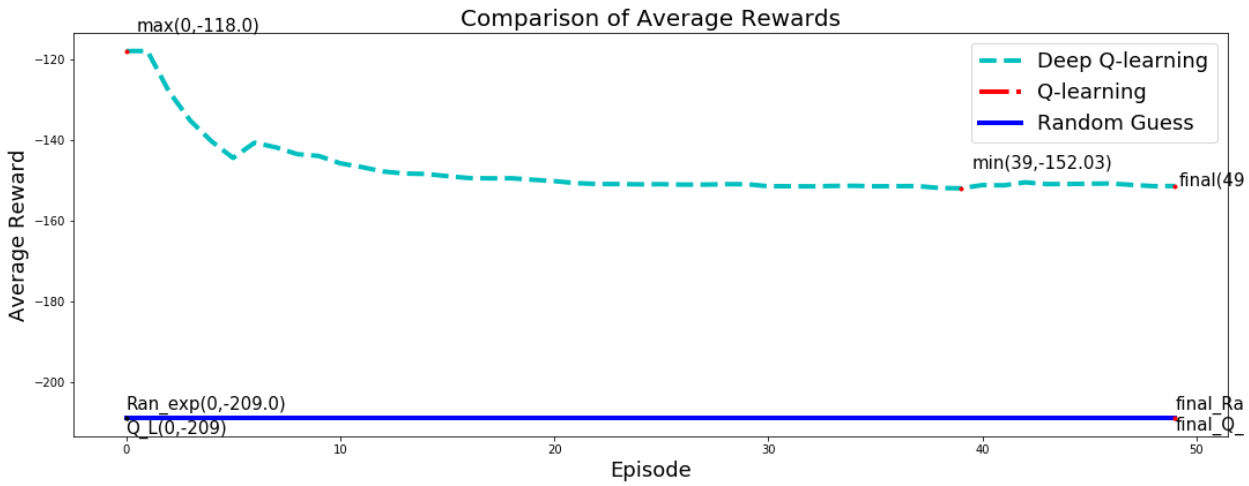


(b) : Mountain Car

Figure 28: Average Score vs Episode (R.E vs Q-learning vs DQN)



(a) : Cartpole



(b) : Mountain Car

Figure 29: Average Reward vs Episode (R.E vs Q-learning vs DQN)

The comparison of the Q learning agent and the Deep Q learning agent with Random Exploration confirmed that the Q learning agent performs slightly better than Random Exploration since it achieves greater max and final values for the considered evaluation metrics hence adhering to the fact that simple Q learning is not completely ineffective but only in-sufficient to handle such environments of very huge state space. The DQN agent however showed remarkable performance in both the environments and clearly outclassed the Q learning agent even when trained for much lesser number of trials than the Q learning agent.

Conclusion

The Results from the above simulation and study clearly show that Q-Learning shows remarkable performance in environments with a small State-Space, however fails to achieve the same in environments of higher dimensions.

The Q-Learning agent does perform better than Random-Exploration but fails to achieve higher rewards or reach the goal state in many cases. The reason behind being the 'Curse of Dimensionality', due to which the size of the Q-table increases exponentially and hence it becomes impossible to iterate over all the possible policies even with a high number of training trials resulting in the poor performance in such high dimensional spaces.

Deep-Reinforcement Learning (here, Deep-Q Learning) manages to overcome this problem by following generalization. The aim of the model is to figure the latent trends in the data gathered by exploring the environment and hence generate a model that can predict the possibly correct (not necessarily optimal) action.

The results substantiate the same since the model manages to achieve very high scores and, in many instances, 'a perfect score' even in much lesser training trials. This strengthens the fact that the DQN-Agent overcomes the limitation of Policy Iteration Q-learning, to explore all states individually, and even with a much-limited exploration can offer remarkable results and hence outperforms the Policy Iteration Q-Learning algorithm.

Scope for Improvement

The results can be further strengthened by extending the Deep Q Learning algorithm over The Atari 2600 games as done in [9]. However, it required advance hardware requirements and hence was not included in this report.

References

[1] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks", Proc. Int. Conf. Learning Representations, 2017.

[2] K. Arulkumaran, N. Dilokthanakul, M. Shanahan, and A. A. Bharath, "Classifying options for deep reinforcement learning," in Proc. IJCAI Workshop Deep Reinforcement Learning: Frontiers and Challenges, 2016.

[3] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in Proc. Int. Conf. Learning Representations, 2016

[4] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. de Maria, V. Panneershelvam, M. Suleyman et al., "Massively parallel methods for deep reinforcement learning", ICML Workshop on Deep Learning, 2015.

[5] N. Heess, J.J. Hunt, T.P. Lillicrap, D. Silver, "Memory-based control with recurrent neural networks", NIPS Workshop on Deep Reinforcement Learning, 2015.

[6] Varsha Lalwani and MasareAkshaySuni. "Playing Atari Games with Deep Reinforcement Learning." , Technical Report, IIT Kanpur, 2015

[7] KristjanKorjus, Ilya Kuzovkin, ArdiTampuu, Taivo Pungas. Replicating the Paper "Playing Atari with Deep Reinforcement Learning", Technical Report, Introduction to Computational Neuroscience, University of Tartu, 2013

[8] M.G. Bellemare, Y. Naddaf, J Veness and M. Bowling."The Arcade Learning Environment: An Evaluation Platform for General Agents.", Journal of Artificial Intelligence Research 47, Pages 253-279, 2013

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", NIPS Deep Learning Workshop, 2013

[10] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction.", Cambridge, MA: MIT Press, 1998.

[11] Christopher JCH Watkins and Peter Dayan. "Q-learning Machine learning," 8(3-4):279–292, 1992.