**SIT – 215 Artificial and Computational Intelligence**

**RAHUL VERMA**

**218606468**

**REINFORCEMENT LEARNING**

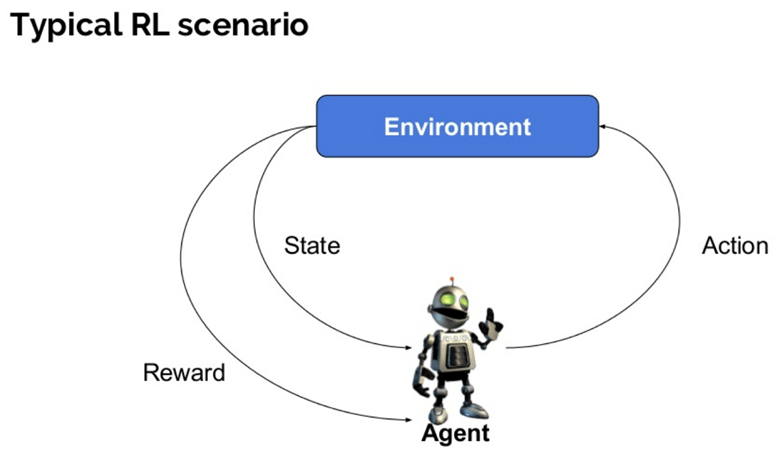
Reinforcement Learning is characterized as a Machine Learning strategy that is basically based on how programming specialists would take necessary actions in any given situation. Reinforcement Learning is an aspect of the profound learning technique that causes a person to amplify some part of the cumulative reward. [1]

For an agent, which could be a self-driving vehicle which whenever interacts with its a environment, gets a prize state contingent upon how it performs, for example, heading to its final destination securely. Then again, if there are any sorts of errors, the agent gets a penalty as such here to go off the street.

Therefore the final call for the agent will be to win rewards and thereby will try to boost and limit its punishment utilizing dynamic programming. In the context of Artificial intelligence, it is beneficial as it permits an AI program to learn without a software engineer illuminating how an agent is ought to play out the given task.

Some of the common terms which I will use in the report ahead are defined below-

* **Agent:** It is the thing which performs activities in a situation to increase its reward. eg- a self-driving car.
* **Environment:** A situation where the agent is put.
* **Reward:** A prompt return given to the agent when it performs as expected.
* **State:** State alludes to the current circumstances prevailing in the environment.
* **Policy:** Next move to be done by the agent after strategizing. [2]



Reinforcement Learning [2]

**Working on the Taxi Problem**

As I had discussed in the example above let’s try to create a self-driving taxi problem.

**Aim:** To introduce a self driving car in the environment by utilizing the RL strategies to produce a safe and a comfortable ride for the people.

People would use this cab to travel from one place to another so some of the things which our agent has to keep a mind of include –

* Pickup the person from the correct location rather than making him/her walk and same scenario for the case of dropping him/her at their destination.
* Must deal with traveler's wellbeing and follow all the traffic rules.
* Spare traveler's time by taking least time conceivable to drop off.

The various perspectives that should be considered while demonstrating a RL answer for this issue include **rewards, states, and activities.**

1. **Prizes**

Since the main aim for the agent will be to collect the reward, will figure out how to control the taxi by preliminary encounters in the environment, we have to choose the prizes as well as punishments and their extent in like manner. Here we need to keep in mind:

* The agent ought to get a high sure award for an effective dropoff in light of the fact that this conduct is profoundly wanted.
* The agent ought to be punished on the off chance that it attempts to drop off a traveler in wrong areas.
* The agent ought to get a small negative compensation for not making it to the objective after each time-step.

\* By “small” compensation since we do not want our agent to arrive late by making incorrect moves in attempting to reach to the objective as quick as could reasonably be expected.

1. **State Space**

In RL, the operator experiences a state, and afterward makes a move as per the situation it is in.

As defined earlier in state, the State Space is the arrangement of all potential circumstances our taxi could possess. The state should contain helpful data which is beneficial for the operator.

Let us assume a situation where we want our smart cab to transfer individuals in a parking area to four distinct areas (R, G, Y, B ):

**3. Activity Space or action space**

Our agent has to experience one of the 500 states and thereby make a move. The activity for our situation can be to move towards a path or choose to pick up/drop off a traveler.

As it were, we have six potential activities:

1. Picking up the person
2. Dropping of the person
3. South
4. North
5. East
6. West

This is the activity space that is the arrangement of the apparent multitude of activities that our agent can follow in a given state.

Note - We will see in the end that the taxi can't play out specific activities in specific states because of dividers. In condition's code, we will just give a - 1 punishment to each divider hit and the taxi won't move anyplace. This will simply pile on punishments making the taxi think about circumventing the divider.

**Working on the Taxi Problem using OpenAI Gym**

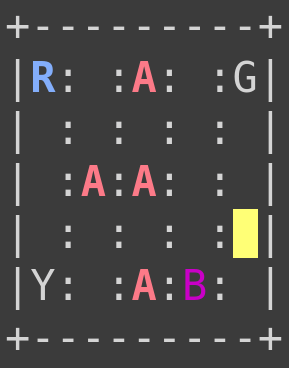
Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides different gaming environments which can be used with codes and tested using an agent.

**Gym Library -** The gym library is an assortment of test issues and situations that one can use in order to work out on RL calculations. These situations have a common interface; permitting a person to compose general calculations. It also provides an Application programming interface (API) that ensures that the relevant data or information needed by the agent is possible and available.

The first step is to download the gym library which I have further executed in pycharm.

Gym has a core interface that is called ***env*** that is a unifying environment interface.

The next step is to define the environment of the taxi as below:



**Code-**

|  |
| --- |
| env.reset() |
|  | env.render() |
|  |  |
|  | print("Action Space {}".format(env.action\_space)) |
|  | print("State Space {}".format(env.observation\_space)) |
|  |  |
|  |  |

1. **Rules:** There are 4 assigned areas on the grid as shown above that are demonstrated by a letter: Red, Blue, Green, and Yellow. At the point when the scene begins, the taxi begins at an arbitrary square and the traveler is at an irregular area. The taxi drive to the traveler's area, get the traveler, drive to the traveler's objective (another of the four determined areas), and afterward drop off the traveler. When the traveler is dropped off, the scene closes. You get +20 focuses for a fruitful dropoff, and lose 1 point for each timestep it takes. There is additionally a 10 point punishment for illicit get and drop-off activities.
2. **Goal-** The objective is to get a traveler at one of the 4 potential areas and to drop him off in another.
3. This maps the action to number:
   * 0 = south
   * 1 = north
   * 2 = east
   * 3 = west
   * 4 = pickup
   * 5 = drop off
4. The color coding is as follows:  
   🡺 Blue: client

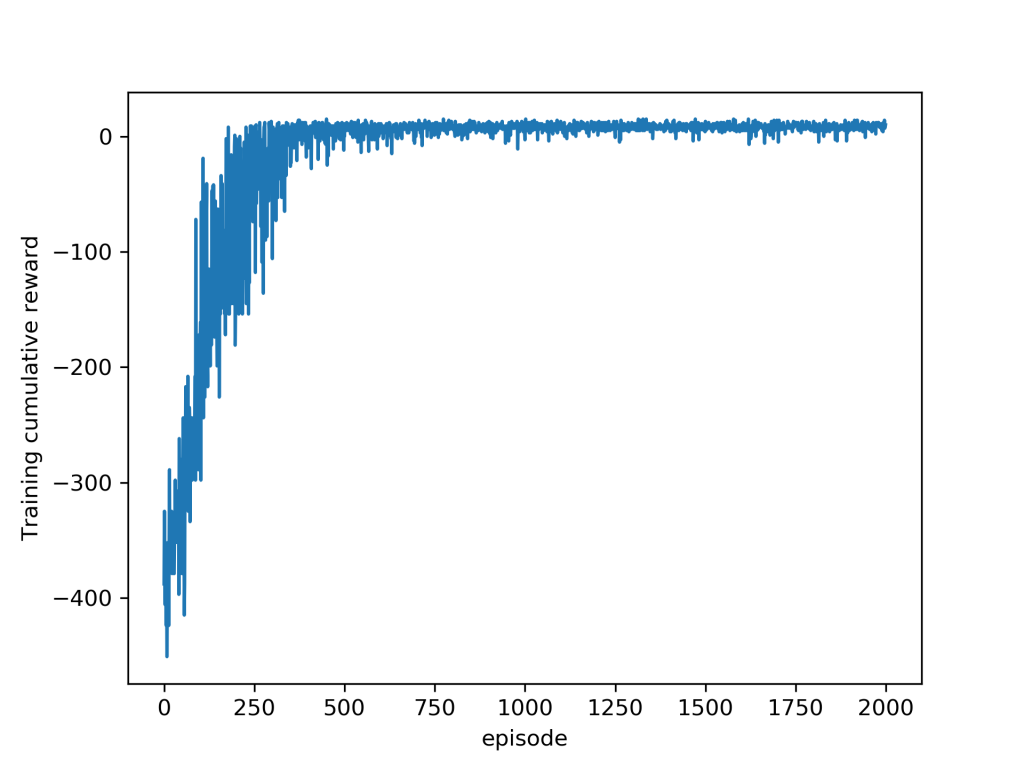
🡺 Magenta: final destination  
🡺 Yellow: empty taxi  
🡺 Green: full taxi  
🡺 Other alphabets: different locations

The next step is to build the agent; this involves developing a simple interface for interaction with the environment.

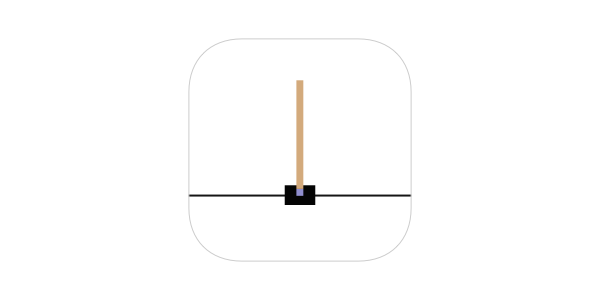
Use the choose\_action technique when we need our agent to make a decision and act. By then, after the reward and new state from the environment are viewed, our agent will pick up from its exercises using the learn method. The other part that is intriguing is instating Q-learning Table utilizing \_\_init\_\_ function. Our strategy is genuinely direct. We draw a sporadic number from a uniform flow some place in the scope of 0 and 1.

In case this number is smaller than epsilon and therefore we have to explore, we make a non-uniform move. Otherwise, we make the best move reliant on our present information. Learning incorporates the update of the Q-table using the Q-learning condition and diminishing the examination rate ϵ if the scene is done.The next step is training of the agent we have created. Next, use 40000 episodes and record episode rewards over time. [3]

|  |
| --- |
| total\_episodes =40000 |
|  | total\_test\_episodes =10 |
|  |  |
|  | agent = Agent(env.observation\_space.n, env.action\_space.n) |
| C:\Users\KenyaGeek\Downloads\image.gif |  |



**Cartpole Problem**



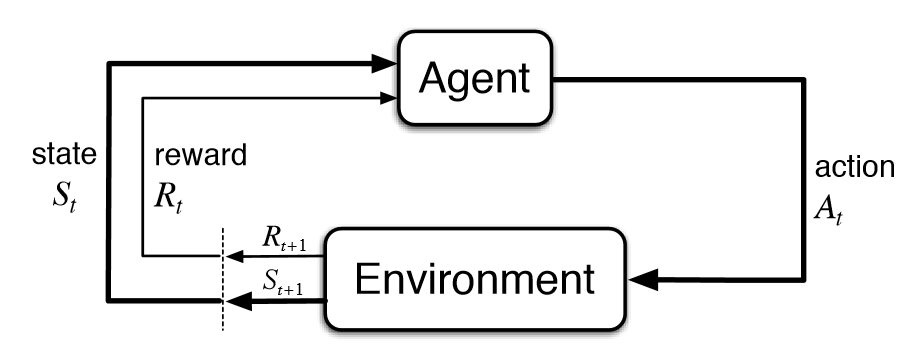
**Cart pole -** referred to likewise as a Modified Pendulum is a pendulum with a focal point of gravity over its turn point. It's flimsy, yet can be constrained by moving the pivot point under the focal point of mass. The objective is to keep the cart pole adjusted by applying suitable powers to a pivot point.

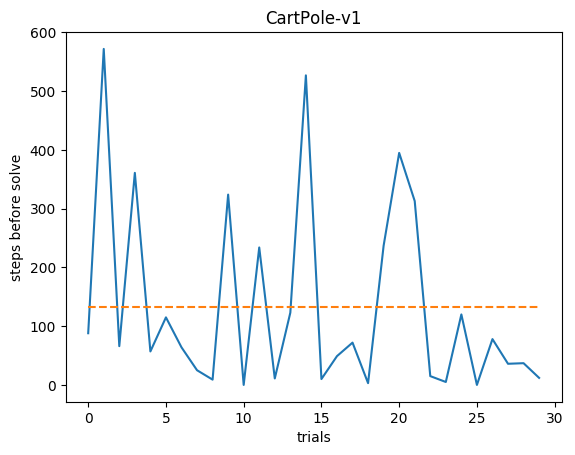
The pole is appended by an un-incited joint to a cart, which rotates on a track without any friction. The framework is constrained by applying a power of +1 or - 1 to the cart. The pendulum begins upstanding, and the objective is to keep it from falling over. A prize of +1 is accommodated in each timestep that it stays upstanding. The scene closes when the shaft is in excess of 15 degrees from the vertical, or if the cart further moves more than 2.4 units from the middle. [4]

At first, we characterize a vector of loads, each weight comparing to one of the perceptions taken above. Then, apply a power of +1 or - 1 to the cart

Cart-pole is unstable. To comprehend the Markov Chain, one would need to comprehend the Markov Model. The Markov Model necessitates that one has to keep in mind their present status and the activity to be performed to foresee your future state. The key about Markov Models however is that there is no need to think about the past states.

The underlying condition we start with just incorporates a state (S\_t). At that point for every cycle, the operator takes the present status (S\_t), picks the best activity (A\_t), and performs it on the situation. Thereafter, the environment restores a prize (R\_t+1) for the activity performed and the new state (S\_t+1). The Cartpole game is based on the Markov Chain. [5]



[4]

**Difference between the CartPole problem and Taxi problem and comparison of Q-learning**

**Cartpole -** Here we let the operator learn in excess of 20 scenes of 100 time steps each. At each time step, a sporadic action is picked by the plan. A prize of +1 is given for each timestamp that results positive in the environment and the complete prize is resolved towards the completion of the scene.

The scene ends if:

* The shaft edge is more than ±12°.
* The truck position is more than ±2.4 (for example the focal point of the cart arrives at the edge of the presentation).
* The scene length is more prominent than 200.
* The issue is viewed as unraveled when the normal prize is more noteworthy than or equivalent to 195 more than 100 successive preliminaries.

Obviously, on the grounds that we are just taking irregular activities, we can't anticipate any improvement additional time.

**Taxi Problem-** The greater part of the current self-driving vehicles utilizes numerous calculations to drive. Moreover, the majority of the methodologies utilize administered figuring out how to prepare a model to drive the vehicle self-sufficiently. This methodology prompts human predisposition being joined into the model. We actualize the Deep Q-Learning calculation to control a mimicked vehicle, start to finish, self-sufficiently. The calculation depends on support realizing which shows machines what to do through cooperations with the earth. The use of fortification learning for driving is of high pertinence as it is profoundly reliant on co operations within the environment. This is actualized by and by a supposed ε-voracious approach. At each time step, a number ε is appointed an incentive somewhere in the range of 0 and 1. Another irregular number is additionally chosen. On the off chance that this number is bigger than ε, a regular activity is chosen and in the event that it is lower, an irregular activity is picked. By and large, ε is diminished from 1 to 0 at each time venture during the scene.

**Difference in terms of Q-learning--** Q-learning is one of the most straightforward Reinforcement Learning calculations. One of the fundamental restrictions of Q learning is that it must be applied to issues with a limited number of states, for example, the taxi model. For situations with a constant state space –, for example, the cart pole issue – it is not, at this point conceivable to characterize a Q table (else it would require an interminable number of columns).

**Q-Learning**

Basically, Q-learning lets the agent utilize environment’s rewards to learn, after some time, the best move to make in a given state.

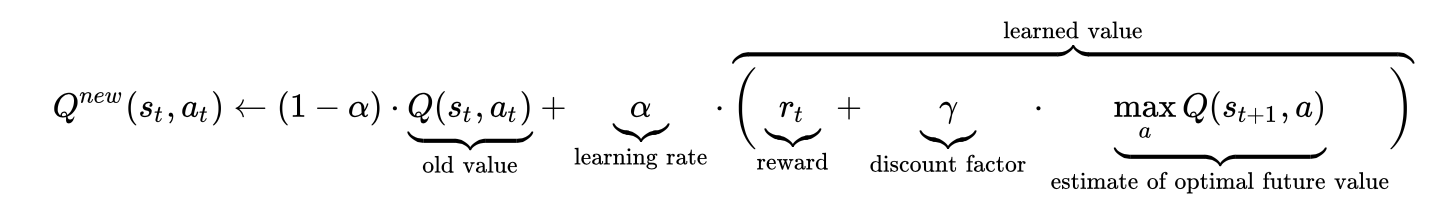
In our Taxi condition, we have the prize table P, from where the operator will gain rewards. It does thing by looking accepting a compensation for making a move in the present status, at that point refreshing a Q-worth to recall whether that activity was valuable.

The qualities stored in the Q-table are known as a Q-qualities, and they guide to a (state, activity) mix.

A Q-esteem for a specific state-activity mix is illustrative of the "quality" of an activity taken from that state. Better Q-values suggest better odds of getting more prominent prizes.

For instance, if the taxi is confronted with an express that incorporates a traveler at its present area, all things considered, the Q-esteem for pickup is higher when contrasted with different activities, as dropoff or north.

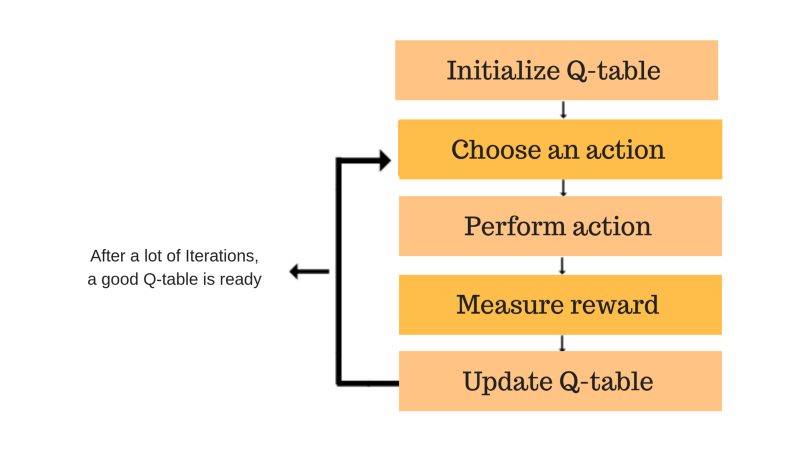
Q-values are introduced to a subjective worth, and as the operator opens itself to the environment and gets various awards by executing various activities, the Q-values are refreshed utilizing the condition:



Where:

- α (alpha) is the learning rate (0<α≤1) - Simply like in administered learning settings, α is the degree to which our Q-values are being refreshed in each cycle.

- γ (gamma) is the markdown factor (0≤γ≤1) - decides how much significance we need to provide for potential compensations. A high incentive for the rebate factor (near 1) catches the drawn out viable honor, though, a markdown factor of 0 causes our operator to think about just prompt prize, henceforth making it covetous.

[6] Steps in Q-learning

Separating it into steps as shown in the figure above, we get:-

* Initialize the Q-table by each of the zeros.
* Start investigating activities: For each state, select any one among all potential activities for the present status (S).
* Travel to the following state (S') because of that activity.
* For all potential activities from the state (S') select the one with the most noteworthy Q-esteem.
* Update Q-table qualities utilizing the condition.
* Set the following state as the present status.
* If objective state is reached, at that point end and rehash the cycle. [6]

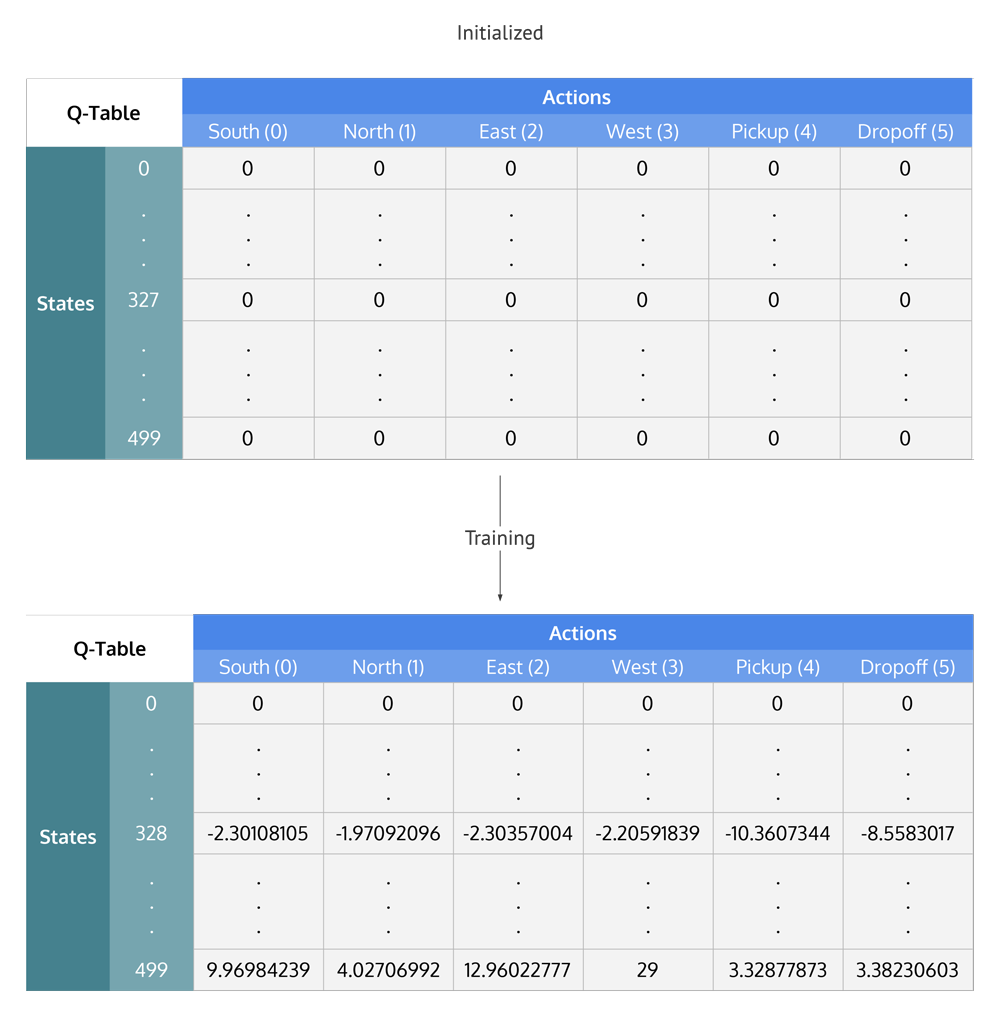
**Explanation of what is done**

We are allotting (←), or refreshing, the Q-estimation of the agent's present status and move by first making a weight (1−α) of the old Q-esteem, at that point including the educated worth. The final value is a blend of the reward for making the current move in the present status, and the discounted maximum reward from the following state we will be in once we make the current move.

Fundamentally, we are learning the correct move to make in the present status by having a reward for the present status, and the maximum awards for the following state. This will in the long run cause our taxi to consider the course with the best rewards hung together.

The Q-estimation of a state-activity pair is the total of the reward received at that particular moment and the future reward. The manner in which we store the Q-values for each state and activity is through a Q-table

**Q-Table -** The Q-table is where we have a line for each state and a column for each activity. It's originally instated to 0, and afterward values are refreshed subsequent to preparing. [7]

[7] Q-table

**The video representation for the code is --** [**https://www.youtube.com/watch?v=jPguCXmTwGc**](https://www.youtube.com/watch?v=jPguCXmTwGc)

**Reference List**

1. [1]"Reinforcement Learning: What is, Algorithms, Applications, Example", *Guru99*. [Online]. Available: https://www.guru99.com/reinforcement-learning-tutorial.html. [Accessed: 04- Oct- 2020].
2. [2]"Reinforcement Learning (RL)", *Technopedia*, 2020. [Online]. Available: https://www.techopedia.com/definition/32055/reinforcement-learning-rl. [Accessed: 05- Oct- 2020].
3. [3]"Part 5 – Q-learning to solve the taxi problem", *Data Machinist*. [Online]. Available: https://www.datamachinist.com/reinforcement-learning/part-5-q-learning-to-solve-the-taxi-problem/. [Accessed: 05- Oct- 2020].
4. [4]G. Surma, "Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning)", *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288. [Accessed: 06- Oct- 2020].
5. [5]N. Kaleem, "Training an AI to Play OpenAI’s Cartpole", *Medium*, 2020. [Online]. Available: https://medium.com/datadriveninvestor/training-an-ai-to-play-openais-cartpole-395e17db847f. [Accessed: 06- Oct- 2020].
6. [6]"Part 5 – Q-learning to solve the taxi problem", *Data Machinist*. [Online]. Available: https://www.datamachinist.com/reinforcement-learning/part-5-q-learning-to-solve-the-taxi-problem/. [Accessed: 05- Oct- 2020].
7. [7]S. Kansal, "Reinforcement Q-Learning from Scratch in Python with OpenAI Gym", *LearnDataSci*, 2020. [Online]. Available: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/. [Accessed: 06- Oct- 2020].