

Contents

Introduction..................................................................................3

What is reinforcement learning…………………………………3

Taxi problem……………………………………………………5

* Random policy…………………………………………...6
* Q-learning………………………………………………..7
* Q-learning vs random policy…………………………….9

Cart pole problem………………………………………………10

* Random policy…………………………………………..11
* Q-learning……………………………………………….12
* Q-learning vs random policy……………………………13

Q-learning( Taxi problem vs Cart-pole problem )……………..14

Mountain problem……………………………………………...15

* Mathematical model……………………………………....16
* Q-learning………………………………………………....17
* Limitation…………………………………………………18

TD- Learning…………………………………………………….19

**Introduction:**

In this task, we will investigate an innovative approach for solving several real-world problems that are fascinating and daunting. One in which, through contact with the external environment and awareness of a reinforcement or input signal, an agent discovers a workable solution. This field is typically referred to as reinforcement learning (RL). RL can also be used as an online way to solve the Markov Decision problem.

This study examines the deployment of multiple forms of RL agents provided by the OpenAI gym in three different environments. This project was developed in python code, which implements various kinds of mathematical operations, Q- learning and TD- learning

Moreover, agent is equipped with certain skills, evaluated and results are visualised using graphs.

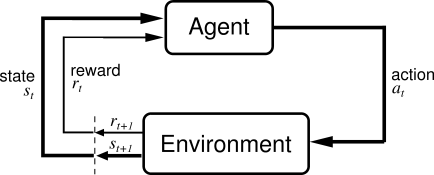
The codes used are submitted in the submission folder as well, but here is the Git hub link: https://github.com/Vir-4/Aritifical-intelligence-Reinforcement-learning-

**Reinforcement learning:**

Reinforcement learning (RL) is a field of AI continuing to solve the problem how programming operators should take activities in a domain ,so as to amplify the idea of cumulative reward. The emphasis was on educating users to act using incentives and punishments in their environment. (Reinforcement learning, 2020)

Supervised learning ,unsupervised learning, reinforcement learning are three essential machine learning paradigms. (Reinforcement learning, 2020)

Reinforcement Learning is just like coaching our pet to do tricks: if your pet does the trick we want, we offer a reward, otherwise punish him by not handling him (Just an example) .RL connects with learning by engagement and suggestions, or in other words, learning to overcome a problem by trial and error.

 (Bhatt, 2020)

Let us understand the terminologies

* The learner and the decision-maker are agents.
* Setting, in which the agent studies and chooses what actions to take is environment.
* Action: a sequence of actions that can be performed by the agent.
* State: the state of the environmental agent.
* Reward: The environment offers a reward for each action chosen by the agent. (Reinforcement Learning algorithms — an intuitive overview, 2020)

**Factors to determine which is better solution ( Compared in all three environments ahead):**

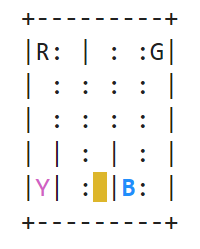
* Average number of penalties per episode: The lower the number, the better our agent's performance. Q-learning leads to less penalties ( proved below ) (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)
* Average number of timesteps per trip: As the shortest and less time-consuming path to the goal is always preferred , so less timesteps is considered better . Random has more timesteps ( proved below )
* Average incentives per move: Rewards are provided when the agent does the right work. Better performance and work tend to more rewards. That's why a key part of Reinforcement Learning is choosing rewards. ( Q learning in all the three environments lead to more rewards ( proved below ) (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

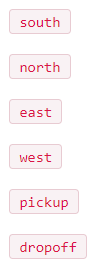
**Taxi problem:**

The problem:

The issue we are interested in solving is the optimization using reinforcement learning (RL) of a self-driving taxi. The taxi issue includes the creation of a system to easily find the shortest path and safely pick up and drop off passengers. Only a small set of states, acts and rewards, Taxi-v2 is well-structured.

The state space is restricted to the 5x5 grid for 25 potential taxi locations, the range of all possible circumstances the taxi could occupy. In addition, there are 5 potential locations for travellers (including where the passenger is in the taxi) and 4 locations for destinations. R, G, Y, B in the grid represent the positions. Whereas the taxi is represented by yellow colour rectangle, our taxi environment has, thus, 5 x 5 x 5 x 4 = 500 cumulative possible states. (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)



This is the action space: the collection of all the activities in each state that our agent will take. Each of the 500 states will consult with the representative to take concrete steps. The space of action, the set of all acts that can be taken by our agent in each state, consists of six potential actions: (Reinforcement Learning for Taxi-v2, 2020)

As the agent is incentive-motivated and can learn how to operate the cab through environmental trial encounters, we need to assess the reward and/or punishments and their severity accordingly.

For actions, the environment gives the following rewards/penalties:

* A high positive reward for valid drop-off = + 20
* A penalty for wrong drop off location = - 10
* With any behaviour in which the passenger is not successfully dropped off or picked-up, a slight negative reward is awarded. = -1 (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

Now will talk about the random policy and the Q-learning applied to taxi problem

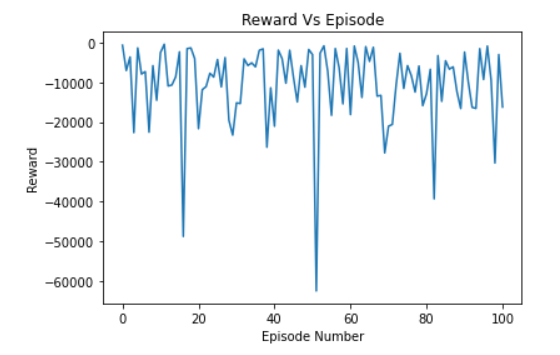
Random policy:

The random agent, operating with no knowledge, takes an action sampled from the available action space. This includes the random making of the agent

It passes until a client is appropriately picked up and dropped off using the incentive table. For the right drop-off, the highest is considered in the reward table 20.

Being random it is extremely ineffective approach as there are a vast number of timesteps before an episode. The random officer does not learn, which results in more unsuccessful drop off and pick off leading to more penalties.

The graph of when we plot the reward vs episodes is provided below which shows that the rewards trend is not uniform and very distributed: (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)





Random policy code snippet in my notebook and results (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

Q-learning:

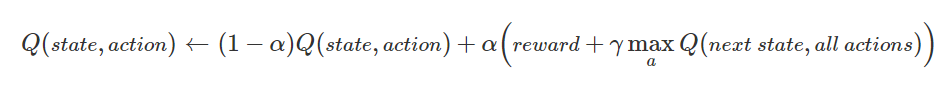
Q-Learning is an algorithm for reinforcement learning (RL) which, considering the state, seeks to determine the best action the agent can take. The aim is to define a programme that maximises the total reward that is anticipated.

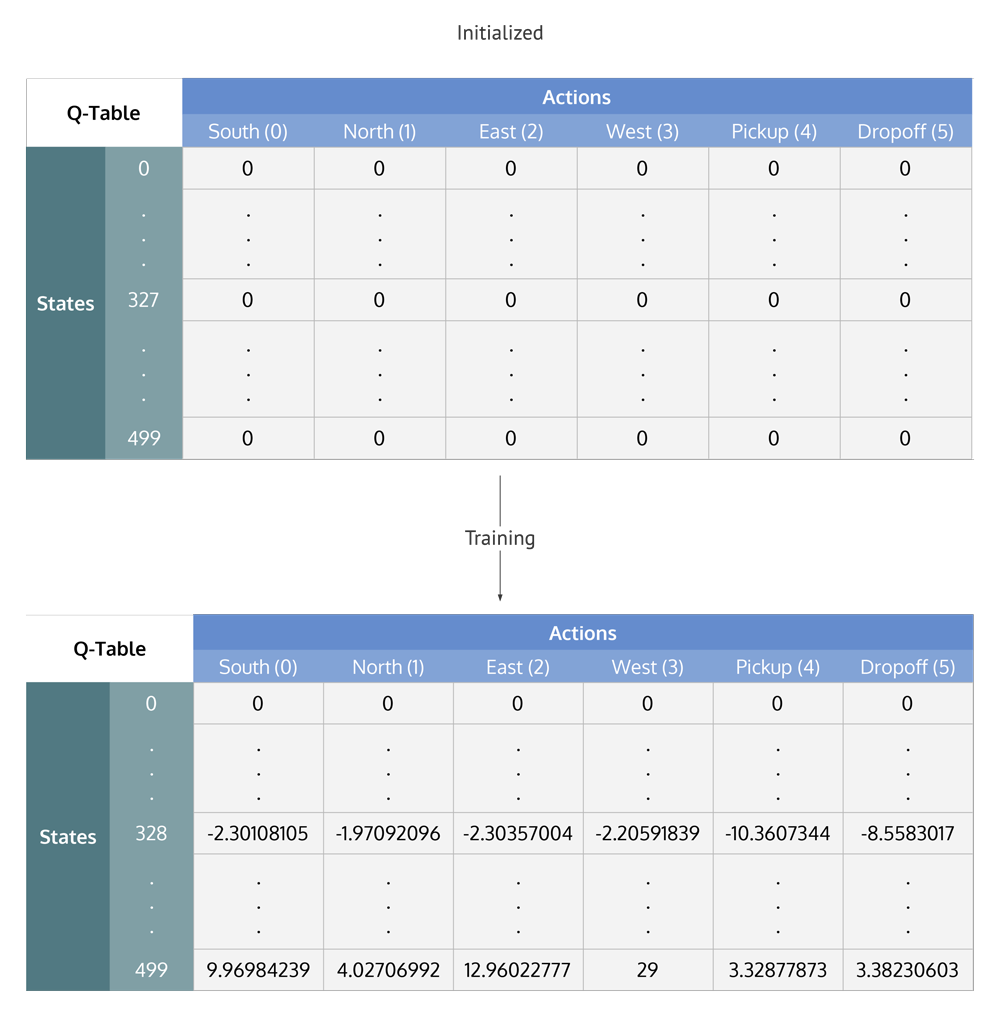
in simple words , Q-learning lets the agent use the rewards of the world to learn the right step to take in each state over time.

We have the reward table, P, in our Taxi environment, that the agent can benefit from. By looking to get a credit for performing an action in the current state, it does something, then updating a Q-value to recall if that action was helpful.

The values contained in the Q-table are referred to as Q-values, and a combination of (state, action) is mapped. (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

Q-values are initialised to an arbitrary value, and the Q-values are modified. using the equation provided below, the agent introduces itself to the world and earns various rewards by performing various actions:

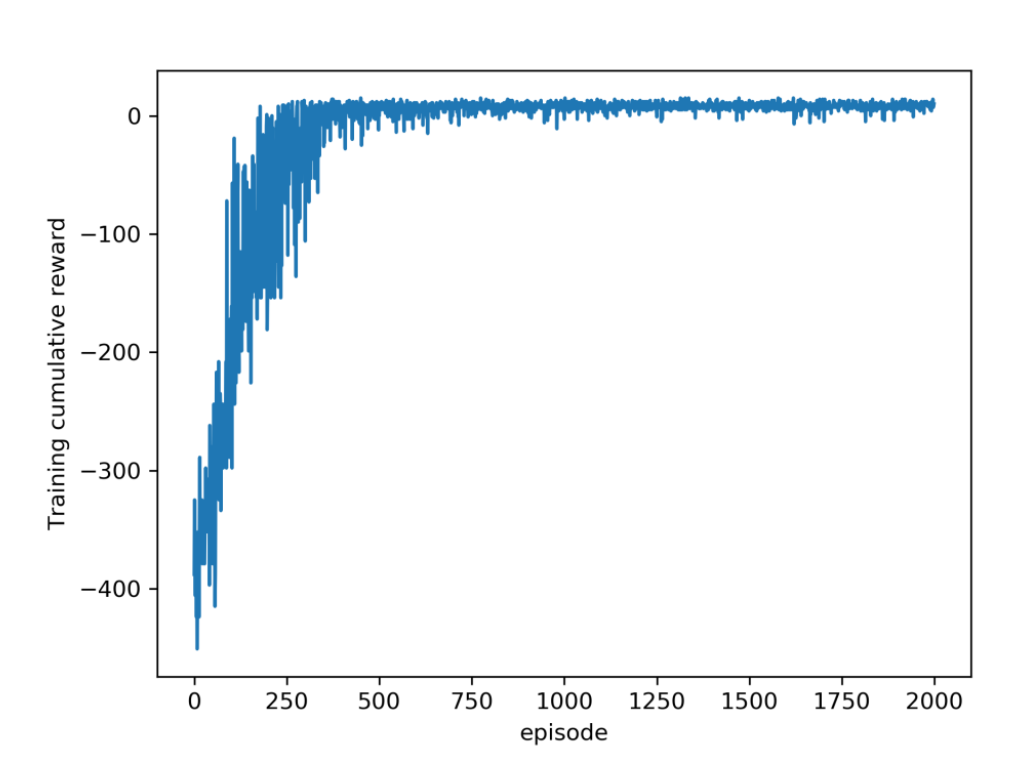
Meaning of different terms are as follows:

* (α ) alpha is the learning rate (0 < α ≤ 1). alpha is the degree to which in every iteration our Q-values are changed.
* (γ) The discount factor (0 ≤ γ ≤ 1)γ (gamma)-decides how much significance we choose to assign to potential incentives. The long-term productive award is caught by a high value for the discount factor (close to 1).
* By first taking a weight (1-a) of the old Q-value, then applying the learned value, we assign (almost), or change, the Q-value of the current state and operation of the agent. The learned benefit is a variation of the incentive for taking the current action in the current state, and once we take the current action, the discounted maximum incentive from the next state we will be in. (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

Q-table given above is made after the Q-value is determined; The Q-table is a matrix of states \* behaviour with the Q-values for each step the taxi has made

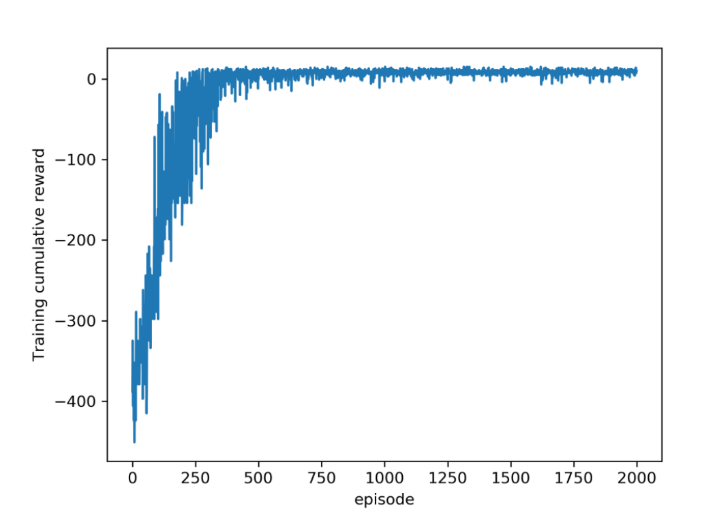
* A trade off occurs between experimentation (choosing a random action) and manipulation (choosing actions based on Q-values that have already been learned). We want to stop constantly following the same path and probably overfitting the action, so we'll be adding another parameter called ε "epsilon" during training to cater to this. (Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, 2020)

The output of the reward vs episode graph after Q-learning:



Q-learning vs random policy:

Initially during discovery, with Q-learning agent commits errors but once it has reasonably explored (seen much of the states), it can behave wisely optimising the incentives making good steps. As per the graphs and data sets.



In the above two graphs although the reward and episode numbers are different, but it is evident that Q-Learning performed better as the time progressed because it kept on learning and improving whereas the random policy could collect that many rewards in less timesteps

**Cart pole problem:**

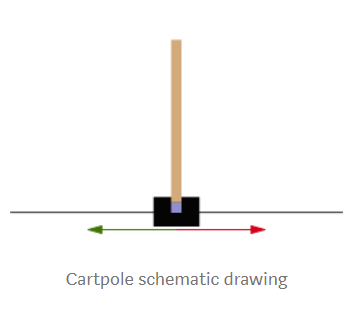
The problem:

Cart pole - A pendulum with a centre of gravity above its pivot point attached to a cart. It is erratic, but it can be controlled by moving the pivot point under the centre of mass. The goal is to keep the cartpole balanced by applying a pivot point of adequate forces. (Using Q-Learning to solve the CartPole balancing problem, 2020)

The problem consists of balancing a pole on top of a travelling cart attached with one joint. Adding a force of -1 to move the cart left and +1 to move the cart to right, moving it left or right, is the only operation available.

This Problem is somewhat like the Taxi problem, (Markov Decision Mechanism) as we are splitting the issue into potential actions, states and rewards . The essence of state space is the most immediate contrast between the cart-pole problem and the taxi problem. Since there is a discrete state space for the taxi problem, the cart-pole problem state is constant across 4 variables.

The state space of the cart-pole problem state is represented in the setting by some floating-point numbers: (Using Q-Learning to solve the CartPole balancing problem, 2020)

* The position of a cart on the track
* Velocity of the Cart
* Angle of the pole
* At the tip, pole velocity

For any moment that the pole stays upright, a reward of + 1 is given(ever timestep). At last the sum of all the rewards are calculated in the episode.

If the pole is more than +-15 degrees from vertical, the episode stops, or the cart travels more than +-2.4 units from the centre. (Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning), 2020)

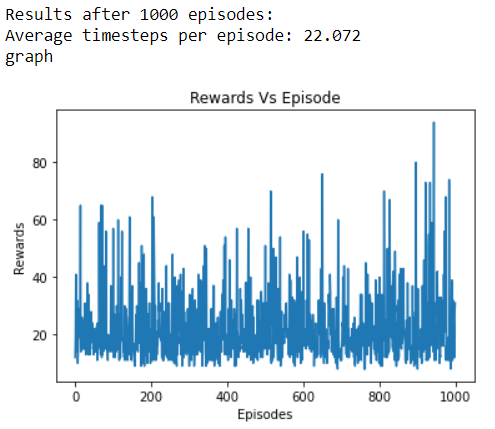
In comparison to the taxi problem, cart pole problem has only one main goal whereas taxi problem had goal which are broken in sub- problems.

Now will talk about the random policy and the Q-learning applied to cart-pole problem.

Random policy:

For this problem, we are using the same approach as taxi problem, Random Policy .That is, random movements were made in the expectation that these movements would prevent the pole from collapsing. As a baselining test, the random agent was applied and worked as planned.

The agent sometimes struggles to quickly stabilise the pole and earns poor incentive ratings, often obtaining marginally better outcomes at random. As this approach is randomized, and no learning takes place is the results were as expected: (Deep learning, 2020)



As in the above graphs we compare the rewards and the no. of episode , it evident from the visualisation that it very distributed and non – uniform because of the randomized nature , and the timesteps value is also greater.

Q-learning:

The concept of Q-learning is already discussed in brief, In the taxi problem . But the implementation here is a bit different and complex . The environment of Cart Pole gives us, as descriptors of the state, the position of the cart, its momentum, the angle of the pole and the momentum at the tip of the pole. All of these, though, are continuous variables. We need to convert these states in order to be able to solve this issue, since otherwise, despite being small, it will take ages to get values for any of the potential combinations of each state. "The solution is to group into the same" bucket "many values of each of the variables and regard them as identical states. (Using Q-Learning to solve the CartPole balancing problem, 2020)



You must be wondering what are( x, x’, theta, theta’)

The environment of Cart-Pole consists of a cart that drives around the horizontal axis and an anchored pole on the cart.

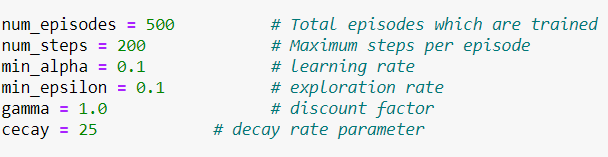
The measurable states of this environment are:

* position (x),
* velocity (x-dot),
* angle (theta),
* angular velocity (theta-dot)

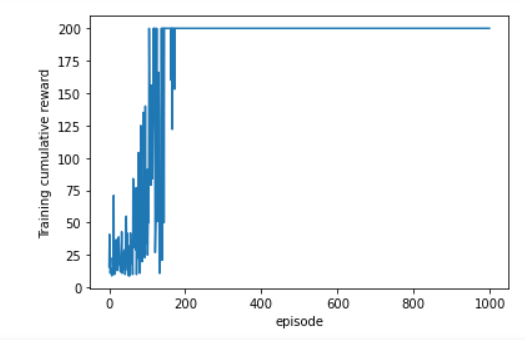
Also, the cart has only two potential acts at any stage: moving to the left or moving to the right.

In other words, there are four dimensions of continuous values in the state-space of the Cart-Pole and the action-space has one dimension of two discrete values. (Using Q-Learning to solve the CartPole balancing problem, 2020)

Whereas the Different hyperparameters used in code are :

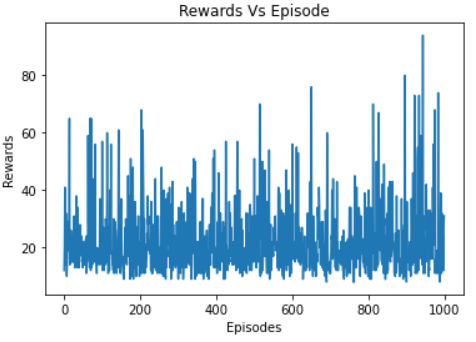
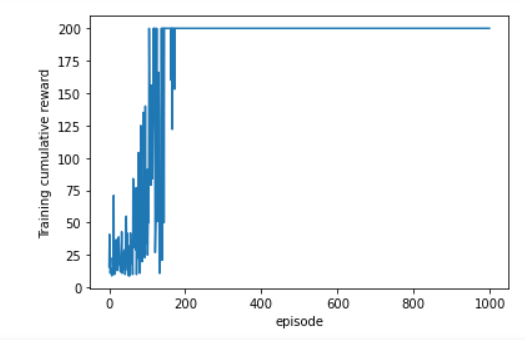


This graph is plotted for the no. of episodes and rewards :



In conclusion this our solution using Q-learning reached 200 steps without dropping the pole. As observed from the graphs , at first the learning is in progress but as it moves ahead the performance is improved.

Q-learning vs random policy:



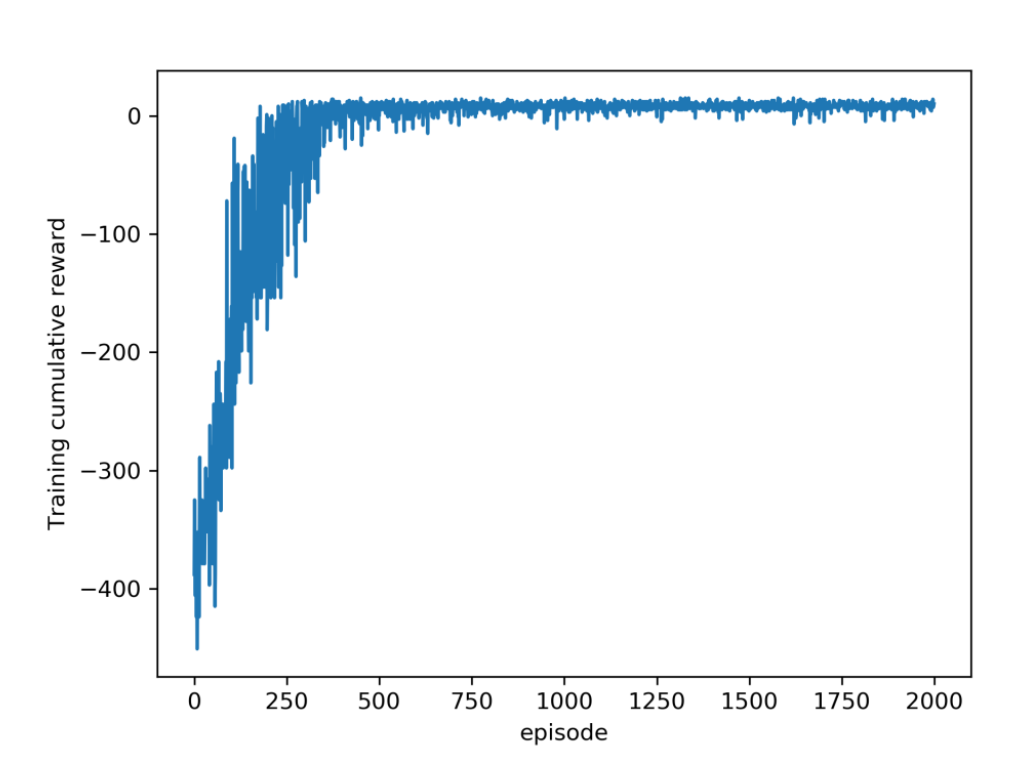
In the above two graphs although the reward and episode numbers are different, but it is evident that Q-Learning performed better as the time progressed because it kept on learning and improving whereas the random policy could not collect that many rewards as in comparison to the Q -learning solution. Also, random policy took many more timesteps compared to Q- learning solution.

**Q – Learning(Taxi problem vs Cart-pole problem):**

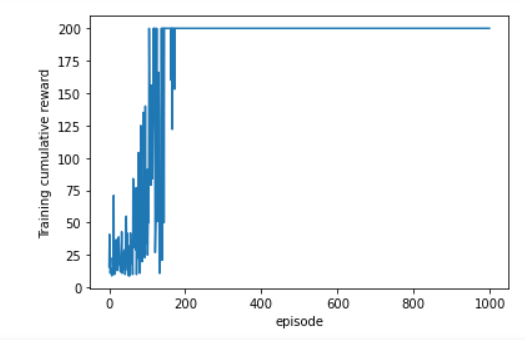
Carefully examining the above two scenarios, both random policy and Q-learning models. Plotting the graphs and values and then comparing them, we concluded that Q- learning performed better in both the cases.

Whereas in this part of the project we will compare the Q-learning models of both the environments.

Taxi problem Q learning graph:



Cart-pole problem Q-learning graph:



Comparing the two graphs , although the implementation was same from the graphs, we can conclude that taxi problem performed way better than cart- pole problem . In the taxi problem at first the agent is making mistakes but is constantly improving to achieve rewards over time. The main thing to look out in these graphs is the shape.

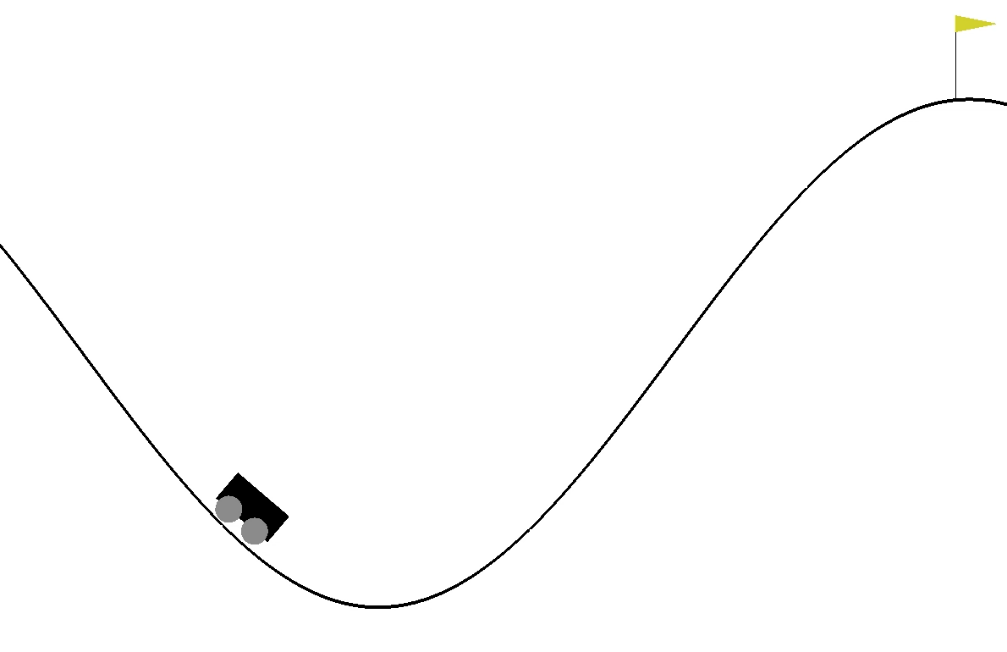
Whereas, in cart pole problem, the till episode 200 the algorithm making many mistakes and receiving less rewards , but after that a stable graph could be seen as stable , the reasons are , looking for short term goals which leads a hindrance the final goal, and the concept of buckets is not that optimised for Q-learning (cart pole ) as in the case of taxi problem states (Q-learning) .

In short, in the same way it did for the taxi problem, Q-learning does not easily achieve the ideal cart-pole environment strategy. A discrete, episodic problem with a granular reward scheme and modest state space is optimal for this approach.

**Mountain car problem:**

The problem:

In Reinforcement Learning, Mountain Car, a common training domain, is a problem in which an under-powered car would accelerate up a steep slope. Since gravity is greater than the engine of the vehicle, although at full throttle, it is not possible for the vehicle to simply drive up a steep slope. The car is positioned in a valley and must learn by going up the opposite hill to maximise potential energy before the car can make it to the target at the top of the right-most hill. (Mountain car problem, 2020)



While this problem is basic, the mountain car problem is widely applied because it involves a reinforcement learning agent to learn about two continuous variables: (Solving Mountain Car with Q-Learning, 2020)

* Location
* speed.

The car is lowered into the valley and assigned an initial location and velocity as a vector as the problem starts. This is the State of the Vehicle.

Then our agent would order the vehicle to take one of three actions:

* drive left,
* do nothing,
* drive right.

This behaviour is sent to the algorithm for the Mountain Car setting that returns a new state (position and velocity) and a reward.

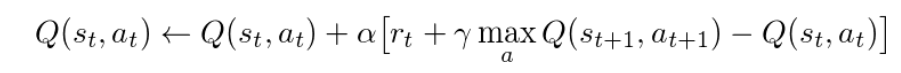
The environment returns a reward of -1 for each move that the car does not meet the target, positioned at location 0.5.

To solve the Mountain Car dilemma, we will use these incentives in our Q-Learning algorithm. Awards never changes until the car reaches the top the algorithm is a bit difficult to solve. (Solving Mountain Car with Q-Learning, 2020)

Q-learning:

We will try to solve this problem using Q-learning, by deciding which behaviour is best in the current state as well as all potential states, Q-Learning is able to solve the issues. We call this function the function of the action value or Q(a, s), where Q is the value of the action an in states.

To be able to work with the continuous state space of the problem, Q-learning and related techniques for mapping discrete states to discrete acts need to be expanded. Approaches mostly fall into one of two categories: discretization of state space or approximation of functions. (Reinforcement Learning with TensorFlow, 2020)

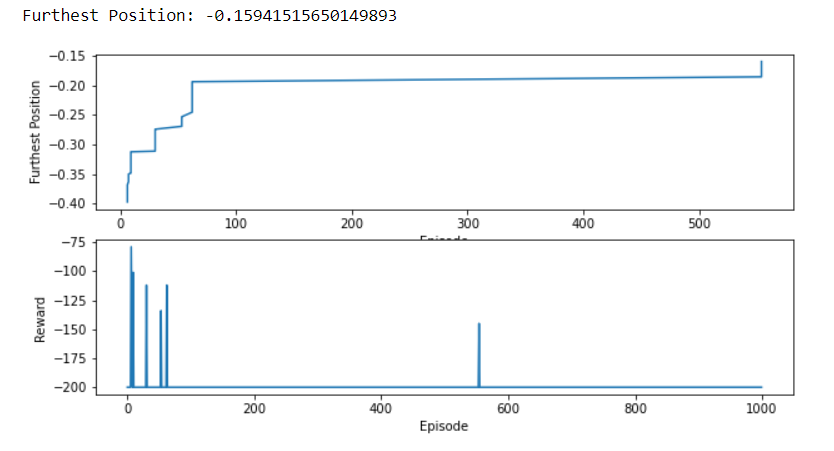
The equation we will use to solve this issue is:

Any time we move forward in the environment by driving left, right, or doing nothing, one phase in the future, we can change Q depending on the incentive for the action of the agent and the maximum future action value feature. For our neural network, the section inside the brackets becomes the loss function where Q(st, at) is the performance of our network and rt + γ max Q(st+1,at+1) is the target Q value and also the label for our neural network , making the problem into a training data problem that can be solved using gradient descent where an is our learning rate. (Solving Mountain Car with Q-Learning, 2020)

Code snippet for plotting the graphs for furthest position and reward:



Output for episode vs ( reward and further position ) :



In a run of 1000 episodes, running the setting with random acts yields no good episodes. For any phase that does not result with a good completion of the episode, the agent gets a reward of -1 so we should change the political learning to reward improvement and enable the agent to travel on and further to the right of the setting. (Solving Mountain Car with Q-Learning, 2020)

From the graphs we can conclude that at first the car is not able to achieve the rewards as the car is learning and not repeating the mistakes, it starts to receive better performance and results.

Limitations:

In the Mountain car problem agent will only receive the reward if the car reaches the top of the hill/ completion of the task. So, the chance of an iterative process of constantly improving is low compared to other environments discussed above in the report. Whereas we can utilise one more method of modifying the reward-based system, rewards would be given based on the position, distance from the goal, etc.

Awards never changes until the car reaches the top the algorithm is a bit difficult to solve

(Solving Mountain Car with Q-Learning, 2020)

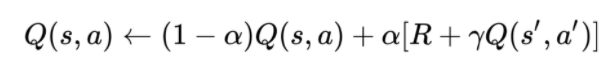
**TD – Learning(Mountain car vs Cart-pole problem):**

To approximate these value functions, Temporal Difference (TD) Learning approaches may be used. If the value functions were to be determined without calculation, before any state-action pair values could be modified, the agent would need to wait before the final compensation was obtained. If the final incentive has been obtained, it will be appropriate to track back the journey taken to reach the final state and change each value accordingly. Q-learning is an algorithm within the category of TD learning. To equate Q-learning to TD learning is inherently ambiguous since the broader type of Q-learning is TD learning. This research considers a implementation of TD learning called SARSA. (Temporal difference learning, 2020) (Reinforcement Learning - TD-Learning, 2020)

An on-policy learning problem is the State-Action-Reward-State-Action (SARSA) algorithm. SARSA is also a temporal difference learning dilemma, just like Q-learning , i.e., it looks forward to the next phase in the episode to predict potential rewards. The biggest difference between SARSA and Q-learning is that the maximum Q-value of the action is not used to update the existing state-action pair's Q-value. (Reinforcement Learning with TensorFlow, 2020)

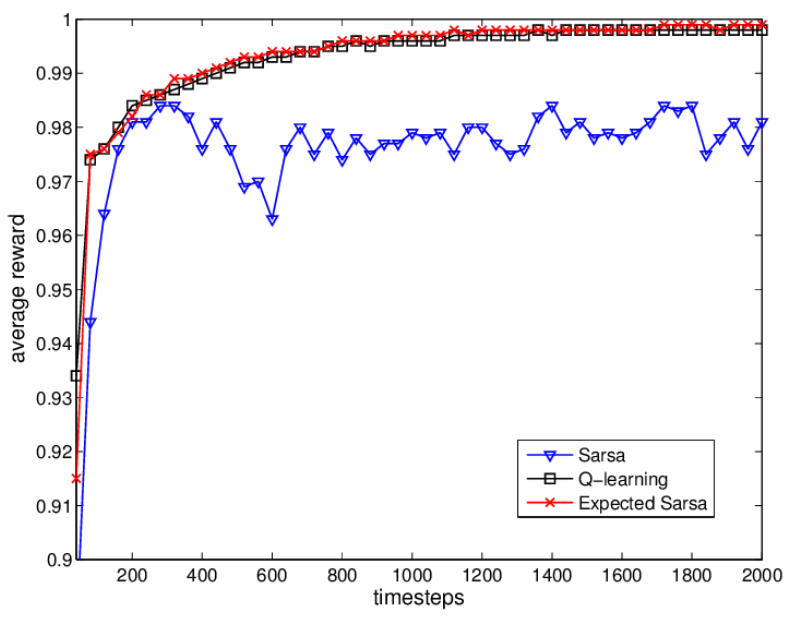
The name SARSA derives from the fact that a quintuple Q(s,a,r,s',a ') is used to change the Q-value, where:

* S, a: existing situation and action situation
* R: reward observed after action is taken a
* S: 'the next condition entered after action has been taken a
* A: 'operation to be carried out at s' state

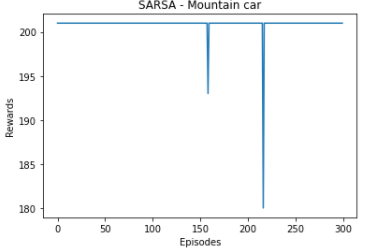
The equation used in SARSA is : (Reinforcement Learning with TensorFlow, 2020)

Temporal Difference – Cart pole

From the graph portrayed below we could conclude that performance of Q-learning and SARSA for the car pole issue is similar in the case of the expected SARSA whereas . compared to SARSA it has minute differences. If we compare SARSA and Q-learning . the performance of Q-learning is better as the agent tends to be more resilient to deceptive learning events and More durable over time, as compared to SARSA. For example, at time step 1000 , SARSA value for expected rewards is less than the Q- learning , although it is a difference of 0.1 value.

 (A.Wiering, 2020)

Temporal Difference – Mountain car



As we validate the graph above the SARSA is seemed to not to be as effective for the mountain car problem as the 230th episode the rewards significantly drop , but SARSA implementation in this environment is similar to the Q-learning graph of the Mountain car problem.

References:

* Medium. 2020. *Using Q-Learning To Solve The Cartpole Balancing Problem*. [online] Available at: <https://medium.com/@flomay/using-q-learning-to-solve-the-cartpole-balancing-problem-c0a7f47d3f9d> [Accessed 1 October 2020].
* Learndatasci.com. 2020. *Reinforcement Q-Learning from Scratch In Python With Openai Gym*. [online] Available at: <https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/> [Accessed 1 October 2020].
* Medium. 2020. *Reinforcement Learning For Taxi-V2*. [online] Available at: <https://medium.com/@anirbans17/reinforcement-learning-for-taxi-v2-edd7c5b76869> [Accessed 2 October 2020].
* Medium. 2020. *Reinforcement Learning Algorithms — An Intuitive Overview*. [online] Available at: <https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc> [Accessed 2 October 2020].
* En.wikipedia.org. 2020. *Reinforcement Learning*. [online] Available at: <https://en.wikipedia.org/wiki/Reinforcement\_learning#:~:text=Reinforcement%20learning%20(RL)%20is%20an,supervised%20learning%20and%20unsupervised%20learning.> [Accessed 2 October 2020].
* Datamachinist.com. 2020. *Part 5 – Q-Learning To Solve The Taxi Problem – Data Machinist*. [online] Available at: <https://www.datamachinist.com/reinforcement-learning/part-5-q-learning-to-solve-the-taxi-problem/> [Accessed 10 October 2020].
* En.wikipedia.org. 2020. *Mountain Car Problem*. [online] Available at: <https://en.wikipedia.org/wiki/Mountain\_car\_problem> [Accessed 3 October 2020].
* Deep learning, S., 2020. [online] Available at: <https://www.youtube.com/watch?v=XlBVoHP1rrA> [Accessed 10 October 2020].
* Medium. 2020. *Cartpole - Introduction To Reinforcement Learning (DQN - Deep Q-Learning)*. [online] Available at: <https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288> [Accessed 3 October 2020].
* Bhatt, S., 2020. *5 Things You Need To Know About Reinforcement Learning - Kdnuggets*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html> [Accessed 3 October 2020].
* Medium. 2020. *Solving Mountain Car With Q-Learning*. [online] Available at: <https://medium.com/@ts1829/solving-mountain-car-with-q-learning-b77bf71b1de2> [Accessed 3 October 2020].
* En.wikipedia.org. 2020. *Temporal Difference Learning*. [online] Available at: <https://en.wikipedia.org/wiki/Temporal\_difference\_learning> [Accessed 1 October 2020].
* Medium. 2020. *Solving Mountain Car With Q-Learning*. [online] Available at: <https://medium.com/@ts1829/solving-mountain-car-with-q-learning-b77bf71b1de2> [Accessed 3 October 2020].
* O’Reilly Online Learning. 2020. *Reinforcement Learning With Tensorflow*. [online] Available at: <https://learning.oreilly.com/library/view/reinforcement-learning-with/9781788835725/cf3f66f3-c9af-4bba-bcda-108ab5dac85b.xhtml> [Accessed 2 October 2020].
* Cse.unsw.edu.au. 2020. *Reinforcement Learning - TD-Learning*. [online] Available at: <https://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html> [Accessed 2 October 2020].
* Datamachinist.com. 2020. *Part 5 – Q-Learning To Solve The Taxi Problem – Data Machinist*. [online] Available at: <https://www.datamachinist.com/reinforcement-learning/part-5-q-learning-to-solve-the-taxi-problem/> [Accessed 1 October 2020].
* A.Wiering, M., 2020. *Fig. 9. The Learning Performance Of The Different Methods On The Cart...*. [online] ResearchGate. Available at: <https://www.researchgate.net/figure/The-learning-performance-of-the-different-methods-on-the-cart-pole-The-results-are\_fig1\_224446259> [Accessed 3 October 2020].