SIT215 – Project

Investigating Reinforcement Learning

Deakin University

Name: Truong Phuc Le (Daniel)

ID: 220090063

Submission Due: Saturday 28 May 2022, 8:00 pm AEST (Week 11)

Table of Contents

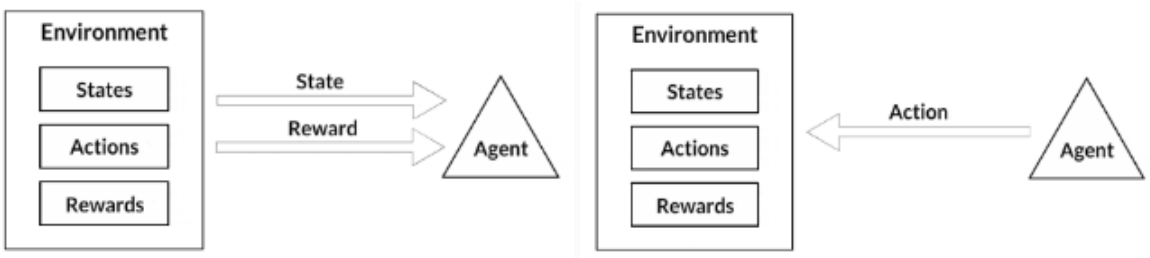
I. Abstract

Providing knowledge and overview of the principal advantage of Reinforcement Learning (RL) is its ability to learn from the interaction with the environment and provide optimal control strategy and method (Q-Learning method as well as TD (0) Learning). Given the efficiency and difference of the above methods, our results are obtained when RL is explored in the control context of the Taxi Problem, Cart-Pole Problem and Mountain Car Problem without a dynamical system with no prior knowledge of the dynamics.

Keywords — Reinforcement Learning; Q-Learning; TD Learning; Policy; Taxi Problem; Cart-Pole Problem; Mountain Car Problem.

II. Reinforcement Learning (RL)

1. Introduction to RL



Reinforcement learning or RL is a type of machine learning technique. The combination of reinforcement learning and artificial neural network programs can be designed which can define agents to learn the best actions possible in a virtual environment to achieve their desired goals.

2. Goal in RL

In Reinforcement learning, the objective is to alter the environment in some way. For example, we'd like the agent to guide a robot to a specified location in space. It will receive a positive reward along with the observation for this timestep if it succeeds in doing so (or makes some progress towards that aim). If the agent has not yet succeeded, the reward may be negative or zero (or did not make any progress). Next, the agent will be taught to maximise the reward it earns over a long period.

3. Basic Method

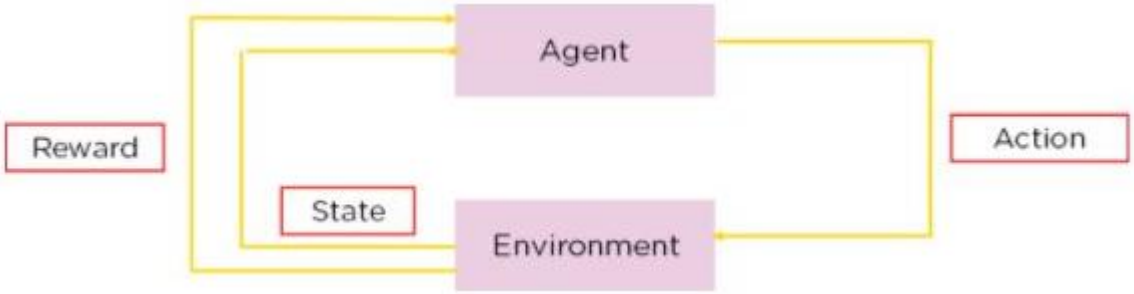
Reinforcement Learning Process: the science of making the best judgments based on past experiences. The method of Reinforcement Learning may be broken down into the following simple steps:

- Observation of the environment
- Deciding how to act using some strategy
- Acting accordingly
- Receiving a reward or penalty
- Learning from the experiences and refining our strategy
- Iterate until an optimal strategy is found

4. Q-Learning Method

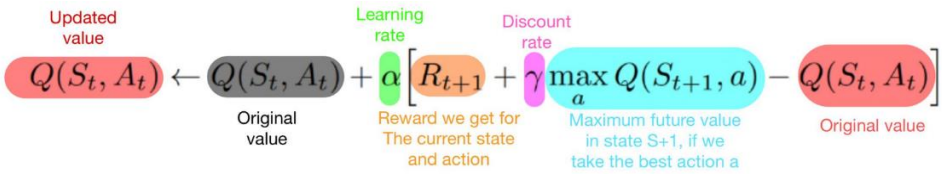
i. Introduction to Q-Learning

Given the current state, Q-Learning is a Reinforcement learning strategy that will determine the next optimal action. It selects this action at random to maximise the reward. Given the present state of the agent, Q-learning is a model-free, off-policy reinforcement learning that will determine the optimum course of action. The agent will pick what action to do next based on where it is in the environment.



ii. Goal in Q-Learning: The goal of Q-learning is to develop an optimal policy in which the expected value of the total reward overall succeeding steps is the highest possible. In other words, the purpose of Q learning is to learn the optimal Q-values for each state-action pair to discover the best policy.

iii. Term and Equation



Important Q-Learning Terms

- **States**: The State, S, represents the current position of an agent in an environment.
- **Action**: The agent's action, A, is the action it takes when it is in a certain condition.
- **Rewards**: The agent will receive a positive or negative reward for each activity.
- **Episodes**: When an agent ends up in a terminating state and can't take a new action.
- **Q-Values**: Used to measure the quality of an Action, A, performed at a certain state, S. Q: (A, S).
- **Temporal Difference**: A formula for calculating the Q-Value based on the present and prior state and action values.

The equation is defined by Richard E. Bellman, as that is used to calculate the worth of a state and to assess how good it is to be in/take that state. The maximum ideal value will be obtained in the optimal state.

iv. Q-Table

We will come across many solutions when executing our algorithm, and the agent will choose multiple pathways. This is accomplished by compiling our findings into a Q-Table. A Q-Table assists us in determining the optimal course of action for each environmental state. At each state, we utilise the Bellman Equation to calculate the projected future state and reward, which we keep in a database to compare to other states.

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

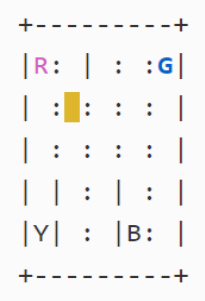
5. Random Policy and Optimal Policy

Non- Reinforcement Method (Random Policy) and Optimal Policy Q-Learning	
Policy-based Reinforcement Learning	**Optimal policy Q-Learning**
Reinforcement Learning with Policy is the agent learns the correct policy through an iterative process known as policy-based reinforcement learning. This is set up at random at the start. This implies the agent performs a random action at each stage and then assesses whether the actions result in positive or poor outcomes. The agent begins to develop a solid policy that can create positive incentives and lead the agent to its objective after numerous iterations. $\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*,$	The optimal Q-function Q*(s, a) means the highest possible Q value for an agent starting from state s and choosing action a. There, Q*(s, a) is an indication of how good it is for an agent to pick action while being in state s. $Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$ Then, to update Q(s, a) we use the **Bellman equation**: $NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$ New Q value for that state and that action · Current Q value · Reward for taking that action at that state · Learning Rate · Discount rate · Maximum expected future reward given the new s' and all possible actions at that new state

III. Taxi Problem

1. Taxi Problem Environment

The taxi problem is a Reinforcement learning problem that involves a variety of strategies. The taxi problem state, that a taxi must pick up and drop off people from one location to another. While doing so, the taxi must take the shortest and quickest route possible. Making software that can discover the quickest path for the Agent (taxi) to pick up N numbers of passengers from their location and drop them off at their intended destination is part of the taxi issue.



Reward: the taxi is rewarded for the actions that it makes. The reward and be positive or negative depending upon the taxi.

- If the taxi makes a correct pick-up and drop-off passenger, the taxi will receive a reward such as positive feedback.
- If the taxi makes a mistake and fails to drop the passenger at the correct drop off location the taxi will receive a penalty.

State: The taxi problem is set on a 5x5 grid. All the squares in the grid are either the pick-up or drop-off locations for the taxi.

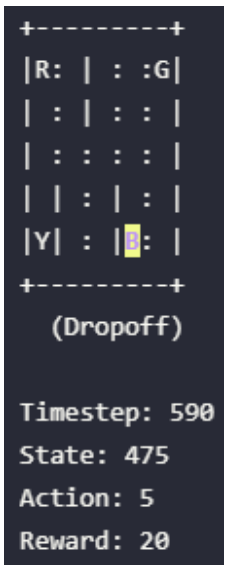
Action: To solve the problem, we need to break down the problem into possible states. The agent has six possible states which it can take: **North, South, East, West, Pick-up and Drop-off.**

2. Solution

i. Without Reinforcement Learning

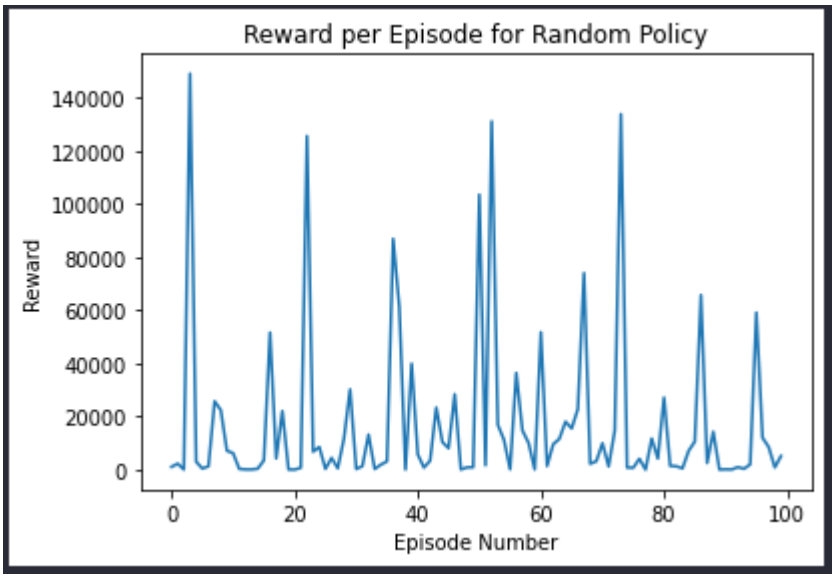
```
Set timesteps
Set Rewart Table with penalties, reward
Loop
    Set agent action
    Set state, reward, action
    if Statement
        increase penalty for wrong pick/drop action
    end if
    increase timestep every loop
```

```
Timesteps taken: 590
Penalties incurred: 180
```



Creating a reward table (called P), which has the number of states as a row and the number of actions as a Column. A loop has been made which will run until one episode. An episode is when a passenger has been picked up from one place and reached a destination. The loop stops when the reward is received which is 20.

This technique is inefficient since it takes too many timesteps and creates several errors while only dropping one passenger. We cannot learn from our past failures using reinforcement learning, which is why it takes too many timesteps and the solution is inefficient, necessitating Reinforcement Learning.



ii. With Q-Learning

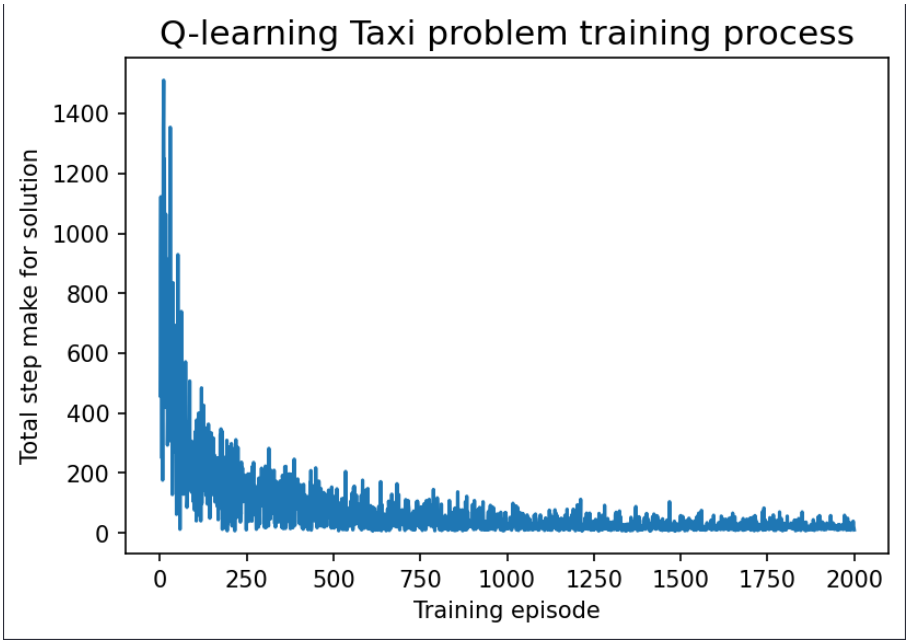
While using the Q-learning method to solve the problem, the agent learns from their prior failures. The reward table in our taxi problem solution is called P, and the agent will utilise it to build a Q-value. The states and actions are listed in the Q table. The Agent can use the Q-value to identify the best course of action for them to take. If the agent's Q-values are high, they will be rewarded well.

```
Q-learning Algorithm
Initialize Q-value (Q(s,a)) for all state-action
loop Statement until learning is stopped
    Choose an action (a) in state(s) based on current Q-value
    take the action(a) and observe the outcome(S) and reward(r)
    Update Q(s,a) by Bellman Equation
```

```
[[  0.          0.          0.          0.          0.
    0.        ]
 [ -2.41837062  -2.36395109  -2.41837066  -2.3639511   -2.27325184
  -11.36395101]
 [ -1.870144    -1.45024001  -1.870144    -1.45024002  -0.7504
  -10.45023983]
 ...
 [ -1.07144845   0.41599913  -1.07759449  -1.24075639  -3.51253754
   -5.1296328 ]
 [ -2.15694074  -2.12207078  -2.1321825   -2.12206989  -7.52405523
   -7.94456731]
 [  3.3383567    1.45671306   4.18092155  11.          -2.62545953
   -2.48491047]]
```

Q-Table is a matrix. In a Q-Table, there is a row for every state. there is a column for every action which has 6. Initially, the values are 0 and the values are updated after the training of the agent. The Q-table gets updated using the equation.

```
Results after 100 episodes:
Average timesteps per episode:  13.05
Average penalties per episode:  0.0
```



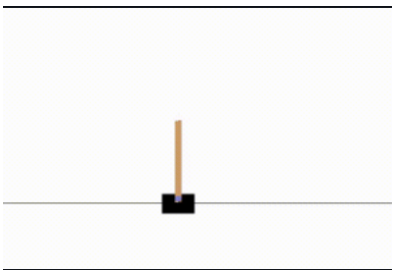
3. Conclusion

When we compare the two solution methods, we can observe that the Q-learning approach is far more efficient. This is because when the programme is run without the RL algorithm, it takes a lot of steps and incurs a lot of penalties. The reason for this is that without the RL algorithm, the agent does not learn from its past mistakes, but Q-learning is much more efficient, and the agent learns from its previous mistakes with Q-learning. As a result, the solution is more efficient, and the agent gets rewarded more.

IV.Cart-Pole Problem

1. Cart-Pole Environment

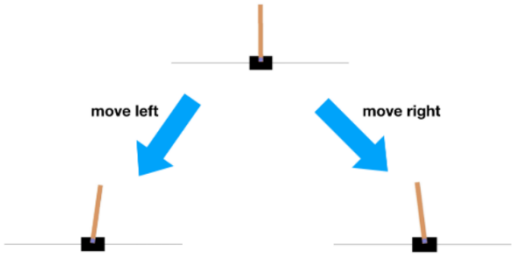
The Cart-Pole Problem With a centre of mass above its pivot point, the Cart-Pole is also known as an Inverted Pendulum. It is unsteady and falls over, but the cart may be moved to regulate it. The purpose of the challenge is to maintain the pole balanced by moving the cart left or right with proper pressures applied to the pivot point.



A Cart-pole is joined to a cart that runs over a frictionless track via an unactuated joint. A force of (+1) or (-1) is applied to the cart to regulate the mechanism. The idea is to keep the pendulum from falling over when it begins upright. Every timestep that the pole remains erect is rewarded with a (+1) bonus. When the pole is more than 15 degrees off vertical or the cart goes more than 2.4 units away from the centre, the episode terminates.

The balance a rod on top of a cart. The number of action spaces is 2. Action space is the discrete following:

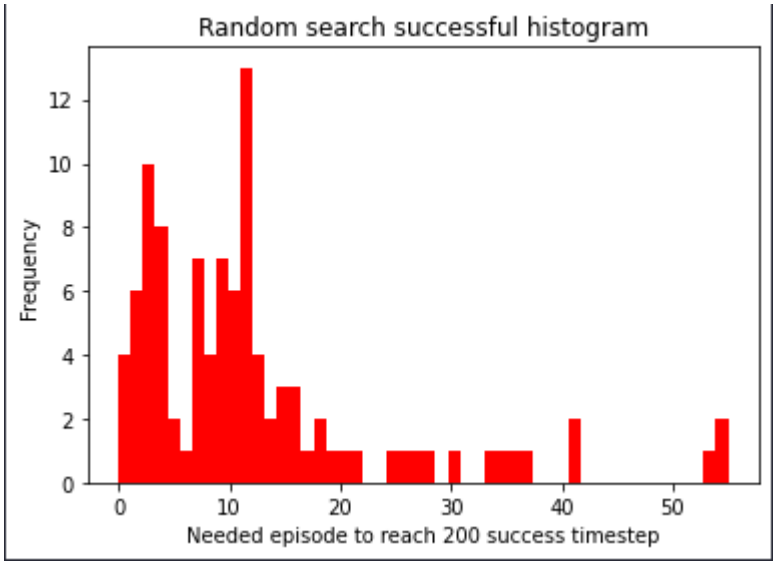
- 0 – move the cart to the left
- 1 - move the cart to the right



2. Solution

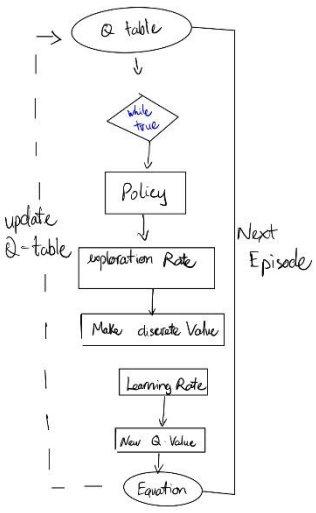
i. Without Reinforcement Learning

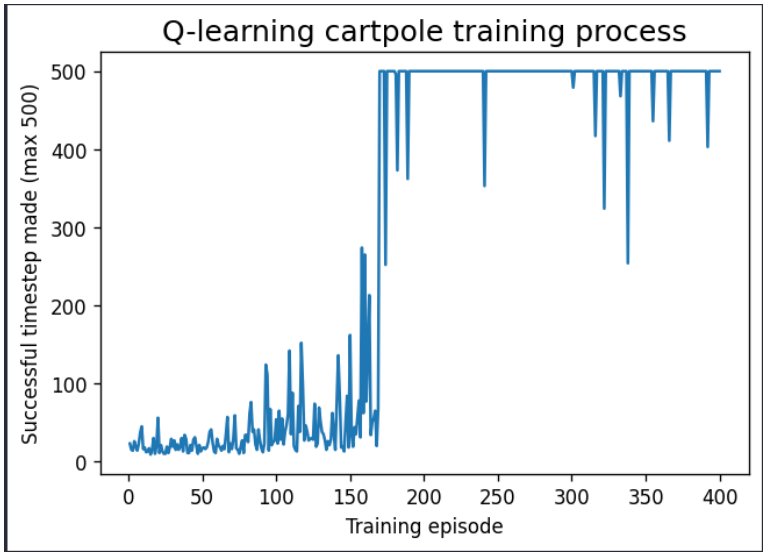
One of the options for solving the cart problem is the random policy reinforcement. The algorithm in this approach performs random motions to maintain the pole balanced, and it utilises over 1000 episodes. This strategy works in the same way that it does in the taxi problem, and it is a good solution. Can be observed from the code that this method takes 13.738 episodes for 200 timesteps on average to keep the pole straight.



ii. With Q-learning

The Q-learning algorithm will work the same as it works for the taxi problem. It creates Q-table and stores in Q-values from the agent will learn their current moves based on the previous moves. The agent can make only two possible actions which are to move left '+1' or to move right '-1'. Using this algorithm, we can see by the graph below which is generated. The agent is trained for 500 episodes by default.





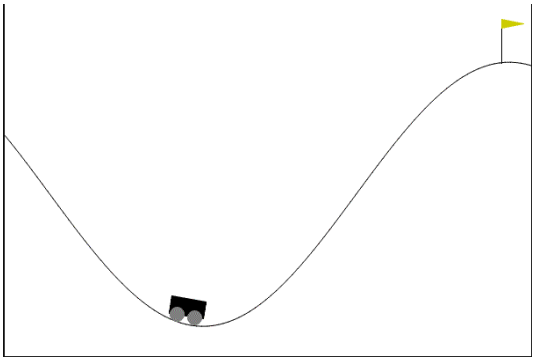
iii. Conclusion

When comparing the random algorithm and the Q-learning algorithm, we can see that the random algorithm is much more efficient than the Q-learning algorithm. This is because, for the cart pole problem, we need a larger number of timesteps for each episode, and the Q-learning algorithm has a smaller number of timesteps than the Random algorithm, so it is inefficient for this problem.

V. Mountain Car Problem

1. Mountain Car Environment

In the game in which the goal is to drive an automobile up a mountain. The automobile is travelling on a one-dimensional track between two "mountains." The aim is to drive up the right-hand mountain; unfortunately, the car's engine isn't powerful enough to do it in one go. As a result, the only way to succeed is to build up momentum by driving back and forth.



Teaching the car to reach the goal position which is at the top of the mountain. The number of action spaces is 3. Action space is the discrete environment following

- 0 - move the car to left
- 1 - do nothing
- 2 - move the car to the right

Reward: The aim is for the agent to reach the flag on top of the right hill as rapidly as possible; as a result, the agent is penalised with a reward of -1 for each timestep it does not achieve the goal and is not penalised (reward = 0) when it does reach the goal.

Starting State: The position of the car is assigned a uniform random value in [-0.6, -0.4]. The starting velocity of the car is always assigned to 0.

Episode Termination: The episode terminates if either of the following happens:

- The position of the car is greater than or equal to 0.5 (the goal position on top of the right hill)
- The length of the episode is 200.

Tabular TD(0) for estimating v_π

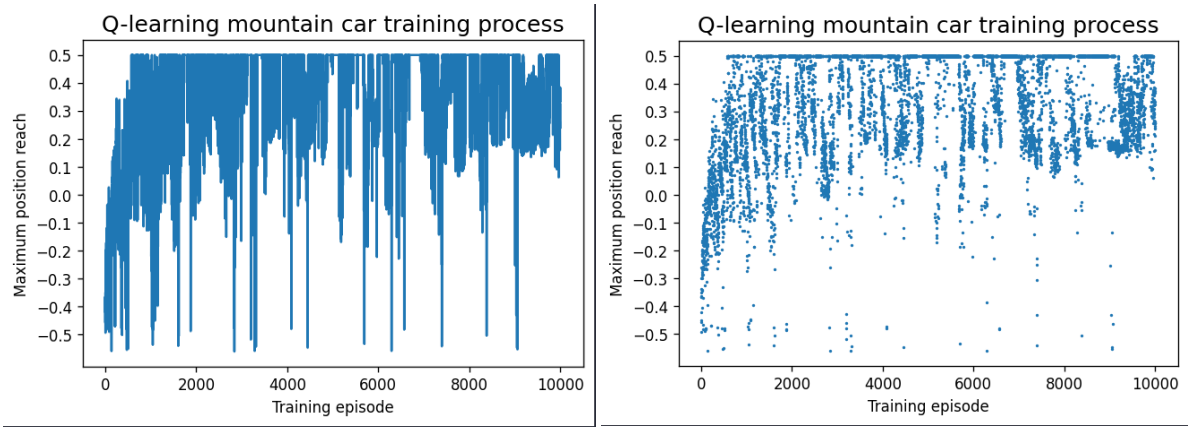
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

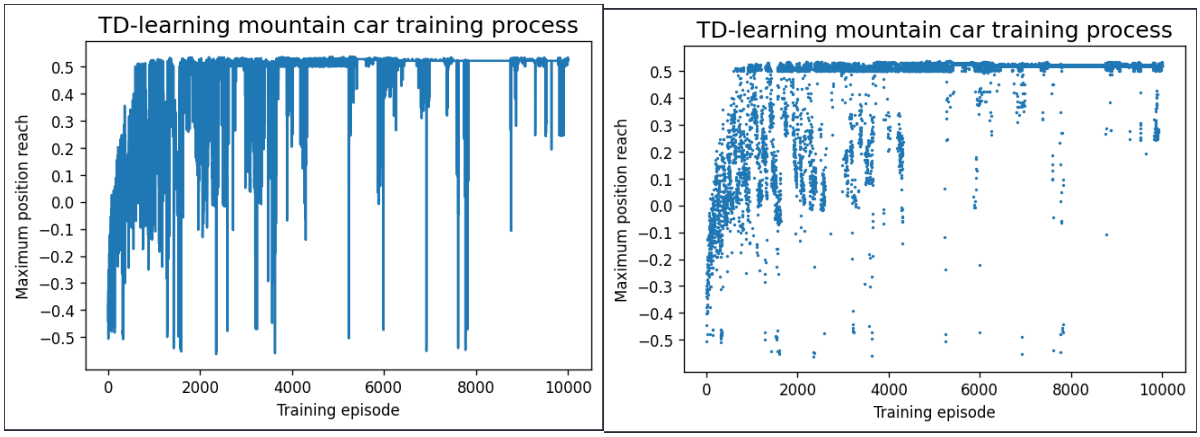
TD Learning Algorithm

2. Solution

The fundamental difference between the two methods is that Q-Learning divides the state space into vehicles and learns the best value for each state, allowing the model to learn extremely complicated mappings. It learns a smooth mapping between the state space and the value function using the TD Learning approach.



Implement Q-Learning



Implement TD Learning

VI.Conclusion

Taxi Problem	Cart-Pole Problem	Mountain Car Problem
The Q-learning method is effective for the taxi problem because it allows the agent to learn from its failures and make the next best move. As a result, the incentives are lower at the start of the algorithm and grow as the number of episodes increases, because the algorithm is efficient and the agent performs the optimal actions to complete each episode.	Whereas in the Cartpole issue, we require a larger number of timesteps for each episode, which Q-learning cannot provide, thus the Random method is employed for the Cartpole algorithm since it is more efficient.	In the Mountain Car problem, it learns to randomly perform actions until it recognizes the actions that give it a higher score. It gets a small boost to its score if it reaches the flag. And it gets more and more points if it gets to the flag fast. Therefore, using the TD learning to train the Mountain Car to get to the flag as fast as possible and efficient.

This report has covered most of the important ideas behind on-policy model-free RL algorithms by examining these two methods. Off-policy approaches and model-based RL methods (in which the agent predicts how the environment will react to its actions rather than learning from experience) are also essential. There are also policy Q-Learning approaches, which do not use value functions to learn optimum policies. Deep Learning RL approaches haven't been discussed yet (such as Deep Q-Learning). According to Richard S. Sutton and Andrew G. Barto all the above methods and the details.

VII. References

Americana C, 2021, "*Reinforcement Learning and Q learning —An example of the 'taxi problem' in Python*", Americana C, retrieved 18 May 2022, <https://towardsdatascience.com/reinforcement-learning-and-q-learning-an-example-of-the-taxi-problem-in-python-d8fd258d6d45>.

Satwik K and Brendan M, .n.d, "*Reinforcement Q-Learning from Scratch in Python with OpenAI Gym*", Satwik K and Brendan M, retrieved 15 May 2022, <https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/>.

Gatti, C., 2014. *Design of experiments for reinforcement learning*. Springer.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

Gym Documentation, .n.d, "*Cart-Pole*", Gym Documentation, retrieved 15 May 2022, < https://www.gymlibrary.ml/environments/classic_control/cart_pole/>.

Gym Documentation, .n.d, "*Mountain Car*", Gym Documentation, retrieved 15 May 2022, < https://www.gymlibrary.ml/environments/classic_control/mountain_car/>.