Training to drive smart cabs using Q-learning models(Reinforcement Learning)

Reinforcement Learning is an area of Machine Learning concerned with how agents take actions in an environment in order to maximize the notion of cumulative reward.
It is different from the conventional supervised or unsupervised learning methods. In Reinforcement Learning, there is an agent. This agent gets a reward, which could be beneficial or detrimental in the form of a penalty.
The main steps of a reinforcement learning method are:
1.Prepare the agent with a set of initial policies and strategy
2.Observe environment and current state
3.Select optimal policy and perform action
4.Get corresponding reward (or penalty)
5.Update policies if needed
6.Repeat Steps 2 - 5 iteratively until the agent learns the most optimal policies.

Q-Learning

It is a basic form of Reinforcement Learning which uses Q-values or action values to improve the behaviour of the agent.

1.Q-Values or Action-Values: Q-values are defined for states and actions. Q(S,A)is an estimation of how good is it to take the action A at the state S(State-Action pairs).

2.Rewards and Episodes: An agent over the course of its lifetime starts from a start state, makes a number of transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in. At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point of time the agent ends up in one of the terminating states that means there are no further transition possible. This is said to be the completion of an episode.
3.Temporal Difference or TD-Update:
The Temporal Difference or TD-Update rule can be represented as follows :

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma Q(S', A') - Q(S, A)\right)$$

This update rule to estimate the value of Q is applied at every time step of the agents interaction with the environment. The terms used are explained below :

- S : Current State of the agent.
- A : Current Action Picked according to some policy.
- S' : Next State where the agent ends up.

- A' : Next best action to be picked using current Q-value estimation, i.e. pick the action with the maximum Q-value in the next state.
- R : Current Reward observed from the environment in Response of current action.
- γ(>0 and <=1) : Discounting Factor for Future Rewards. Future rewars are less valuable than current rewards so they must be discounted. Since Q-value is an estimation of expected rewards from a state, discounting rule applies here as well.
- α : Step length taken to update the estimation of Q(S, A).

4.Choosing the Action to take using ϵ-greedy policy:

ϵ-greedy policy of is a very simple policy of choosing actions using the current Q-value estimations. It goes as follows :

- With probability $(1-\epsilon)$ choose the action which has the highest Q-value.(Exploitation)
- With probability (ϵ) choose any action at random.(Exploration)

Here, we apply the Q-Learning algorithm to train a taxi agent.The goal is to navigate in the city and transport passengers from point A to B.The agent receives a high positive reward for a successful drop off and is penalized if it tries to drop off a passenger in wrong locations

The agent also gets a -1 with every step as the agent must take the passenger from his location to the destination as fast as possible.

Performance measure:

- Drop off the passenger at the right location.
- Save passenger's time by taking minimum time possible to drop off
- Take care of passenger's safety and traffic rules

Environment and its characteristics:

The environment is a 5x5 grid with 4 possible destinations(R,G,Y,B).

The passenger has to be picked up from one location and dropped off at the desired location.

The environment is **fully observable** as the agent can access the complete state of the environment at each point of time.

The environment is **deterministic** as agent’s current state and selected action can completely determine the next state of the environment.

The environment is **non-episodic or sequential** as memory of past actions is required to determine next best actions.

The environment is **static** and has a **single agent.**

It is **discrete** since there are a finite number of actions.

Actions:

There are 6 possible actions:

-South

-North

-East

-West

-Pickup

-Drop off

Implementation in Python:

First we need to import essential libraries.

```
import numpy as np
import gym
import random
```

Then we Create the environment.

```
env = gym.make("Taxi-v3")
env.render()
```

Now we create the Q-table.The number of actions, i.e. 6, gives number of columns and number of states, i.e.500, gives number of rows.

```
action_size = env.action_space.n
print("Action size",action_size)

state_size = env.observation_space.n
print("State size",state_size)
```

Then we Initialize values of Q-table(500,6).

```
qtable=np.zeros([state_size,action_size])
print(qtable)
```

The next step is to define hyperparameters including total episodes, total test episodes, maximum steps in each episode,learning rate,gamma,epsilon or exploration rate,min epsilon or exploration probability at the start,min epsilon or minimum exploration probability and decay rate or exponential decay rate for exploration.

As the training goes on, we progressively reduce epsilon value since we need less exploration and more exploitation.

```
total_episodes=50000
total_test_episodes=100
max_steps=99

learning_rate=0.7
gamma=0.618

epsilon=1.0
max_epsilon=1.0
min_epsilon=0.01
decay_rate=0.01
```

Then we implement the Q-Learning algorithm.

```
for episode in range(total_episodes):
    #Resetting the environment
    state = env.reset()
    step = 0
    done = False

    for step in range(max_steps):
        #Randomize a number
        exp_exp_tradeoff = random.uniform(0,1)

        #If the number>epsilon, then exploitation or action with largest Q-value
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])

        #Else exploration or random choice
        else:
            action = env.action_space.sample()

        #Take action and observe state and reward
        new_state, reward, done, info = env.step(action)

        #Update Q(S,A)
        qtable[state, action] = qtable[state, action] + learning_rate * (reward +
        gamma * np.max(qtable[new_state, :]) - qtable[state, action])


        #New state
        state = new_state

        #Finish episode if done
        if done == True:
                break

        #Reducing epsilon as we need minimum exploration
        epsilon = min_epsilon + (max_epsilon -
        min_epsilon)*np.exp(-decay_rate*episode)
```

Then we can test the agent and calculate score as sum of rewards/total_test_episodes.

```
env.reset()
rewards = []

for episode in range(total_test_episodes):
        state = env.reset()
        step = 0
        done = False
        total_rewards = 0

        for step in range(max_steps):
                env.render()
                action = np.argmax(qtable[state,:])

                new_state, reward, done, info = env.step(action)

                total_rewards += reward

                if done:
                        rewards.append(total_rewards)
                        break
                state = new_state
env.close()
print ("Score over time: " +    str(sum(rewards)/total_test_episodes))
```

OUTPUT:
We get a score of 8.16.