Project 4: Reinforcement Learning

UBIT Name: Prashi Khurana
UBIT Number: 50316796
prashikh@buffalo.edu

Abstract

In this project, we will learn how to train an agent to achieve a task goal. It will be trained by Q-Learning. Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.

1 What is Reinforcement Learning

1.1 Formal Definition

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience. [1]

1.2 Understanding Reinforcement Leaning in Basic Terms

The main concept is make mistakes and learn. In very lame concepts this can also be considered as make mistakes and learn. Consider the very case for humans, if we make a mistake and we get hurt we are probable to never do that agian. A machine on the other hand will get rewards for every action it takes and the basic goal is to maximise the rewards.

1.3 Advantages and disadvantages of Reinforcement Learning

Some **advantages**:

Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques

This technique is preferred to achieve long-term results which are very difficult to achieve

This learning model is very similar to the learning of human beings. Hence, it is close to achieving perfection

The model can correct the errors occurred during the training process

Once an error is corrected by the model, the chances of occurring the same error are very less. It can create the perfect model to solve a particular problem.

Some **disadvantages**:

Reinforcement learning as a framework is wrong in many different ways, but it is precisely this quality that makes it useful.

Too much reinforcement learning can lead to an overload of states which can diminish the results.

Reinforcement learning is not preferable to use for solving simple problems.

Reinforcement learning needs a lot of data and a lot of computation. It is data-hungry. That is why it works really well in video games because one can play the game again and again and again, so getting lots of data seems feasible[2]

2 Understanding Reinforcement Leaning in terms of Mathematics and Learning the Formula

2.1 Some basic understanding of Initial variables involved.

Agent: An agent takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is the agent.

Action (A): A is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents usually choose from a list of discrete, possible actions. In video games, the list might include running right or left, jumping high or low, crouching or standing still.

Discount factor: The discount factor is multiplied by future rewards as discovered by the agent in order to dampen these rewards' effect on the agent's choice of action. Why? It is designed to make future rewards worth less than immediate rewards; i.e. it enforces a kind of short-term hedonism in the agent.

Environment: The world through which the agent moves, and which responds to the agent. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state.

State (S): A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes.

Reward (R): A reward is the feedback by which we measure the success or failure of an agent's actions in a given state.

Policy (π): The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.

Value (V): The expected long-term return with discount, as opposed to the short-term reward R.

Q-value or action-value (Q): Q-value is similar to Value, except that it takes an extra parameter, the current action a. Qπ(s, a) refers to the long-term return of an action taking action a under policy π from the current state s.

Trajectory: A sequence of states and actions that influence those states.

Key distinctions: Reward is an immediate signal that is received in a given state, while value is the sum of all rewards you might anticipate from that state. Value is a long-term expectation, while reward is an immediate pleasure.[3]

2.2 Some basic concepts in Reinforcement Learning

2.2.1 Markov Decision Process(MDP):

In an MDP, we have a set of states $\boldsymbol{S}$, a set of actions $\boldsymbol{A}$, and a set of rewards $\boldsymbol{R}$. We'll assume that each of these sets has a finite number of elements.

At each time step $t = 0, 1, 2, \cdots$, the agent receives some representation of the environment's state $S_t \in \boldsymbol{S}$. Based on this state, the agent selects an action $A_t \in \boldsymbol{A}$. This gives us the state-action pair (S_t, A_t).

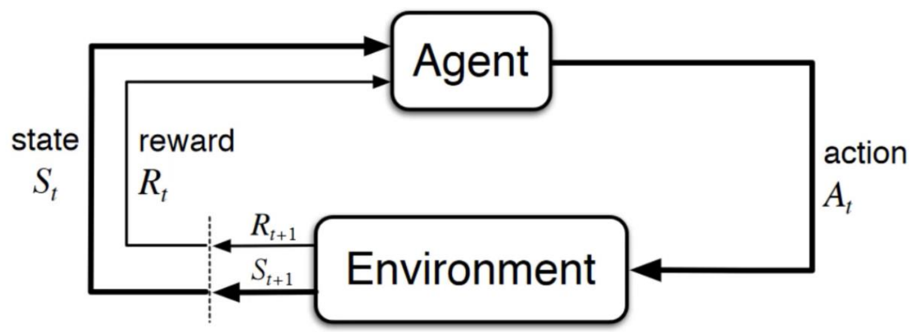
Time is then incremented to the next time step $t+1$, and the environment is transitioned to a new state $\boxed{S_{t+1} \in \boldsymbol{S}}$. At this time, the agent receives a numerical reward $R_{t+1} \in \boldsymbol{R}$ for the action A_t taken from state S_t.

We can think of the process of receiving a reward as an arbitrary function f that maps state-action pairs to rewards. At each time t, we have

$$f(S_t, A_t) = R_{t+1}.$$

The trajectory representing the sequential process of selecting an action from a state, transitioning to a new state, and receiving a reward can be represented as

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$$



Transition probabilities

Since the sets $\boldsymbol{S}$ and $\boldsymbol{R}$ are finite, the random variables R_t and S_t have well defined probability distributions. In other words, all the possible values that can be assigned to R_t and S_t have some associated probability. These distributions depend on the *preceding* state and action that occurred in the previous time step $t-1$.

For example, suppose $s' \in \boldsymbol{S}$ and $r \in \boldsymbol{R}$. Then there is *some* probability that $S_t = s'$ and $R_t = r$. This probability is determined by the particular values of the *preceding* state $s \in \boldsymbol{S}$ and action $a \in \boldsymbol{A}(s)$. Note that $\boldsymbol{A}(s)$ is the set of actions that can be taken from state s.

Let's define this probability.

For all $s' \in \boldsymbol{S}$, $s \in \boldsymbol{S}$, $r \in \boldsymbol{R}$, and $a \in \boldsymbol{A}(s)$, we define the probability of the transition to state s' with reward r from taking action a in state s as

$$p\left(s', r \mid s, a\right) = \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}.$$

[4]

2.2.2 Cumulative Rewards:

For now, we can think of the return simply as the sum of future rewards. Mathematically, we define the return G at time t as

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

where T is the final time step. [4]

This is important because to maximize this, is the ultimate goal of the agent.

But since the time T can be infinite so can the can the rewards, so we will use the concept of Discounted Rewards.

2.2.3 Discounted Rewards:

To define the discounted return, we first define the discount rate, γ, to be a number between 0 and 1. The discount rate will be the rate for which we discount future rewards and will determine the present value of future rewards. With this, we define the *discounted return* as

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \end{aligned}$$

Now, check out this relationship below showing how returns at successive time steps are related to each other. We'll make use of this relationship later.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+3} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

[4]

Main goal is now maximise G_t.

2.2.4 State Value Function vs Action Value Function :

State-value function

The *state-value function* for policy π, denoted as v_π, tells us how good any given state is for an agent following policy π. In other words, it gives us *the value of a state* under π.

Formally, the value of state s under policy π is the expected return from starting from state s at time t and following policy π thereafter. Mathematically we define $v_\pi(s)$ as

$$\begin{aligned} v_\pi(s) &= E\left[G_t \mid S_t = s\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]. \end{aligned}$$

Action-value function

Similarly, the *action-value function* for policy π, denoted as q_π, tells us how good it is for the agent to take any given action from a given state while following following policy π. In other words, it gives us *the value of an action* under π.

Formally, the value of action a in state s under policy π is the expected return from starting from state s at time t, taking action a, and following policy π thereafter. Mathematically, we define $q_\pi(s, a)$ as

$$\begin{aligned} q_\pi(s, a) &= E\left[G_t \mid S_t = s, A_t = a\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]. \end{aligned}$$

[4]

Here The action value function is called Q-Function which we will be implementing in our project.

2.2.5 Optimal Policies

Optimal action-value function

Similarly, the optimal policy has an *optimal* action-value function, or *optimal* Q-function, which we denote as q_* and define as

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

for all $s \in \boldsymbol{S}$ and $a \in \boldsymbol{A}(s)$. In other words, q_* gives the largest expected return achievable by any policy π for each possible state-action pair. [4]

This q_* must follow what we call the Bellman optimality equation, given by

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$ [4]

2.2.6 Epsilon Greedy Strategy

Initially set Epison to 1. This will tell the agent to explore the environment rather than exploit it. But as the agent learns more and more about the environment the epsilon begins to decay. The agent will then soon become greedy to exploit the environment than explore.

So how do we get the agent to choose exploitation or exploration. We chose a random number r (between 0,1) and if r>epsilon then we choose exploitation, that is it will choose the highest q value.

If r<epsilon, then it will choose exploration and randomly choosing its action,

2.2.7 Learning Rate

What is Learning Rate ?
It is a value between 0 and 1.
This determines how fast is the agent ready to leave the previous q value in the table for a new q value.
Or we can use the learning rate to keep how much information of the old q value we need to keep.

It states that for any state-action pair(s,a) at time t, the expected return from starting in state si selecting action a and following the optimal policy thereafter is going to be the expected reqard we get from taking action a in state s which is Rt+1 plus the maximum expected discounted return that can be achieved from any possible next-state-action pair (s',a').

2.2.8 Updating the Q-value table

The formula for calculating the new Q-value for state-action pair (s, a) at time t is this:

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s', a')\right)}^{\text{learned value}}$$

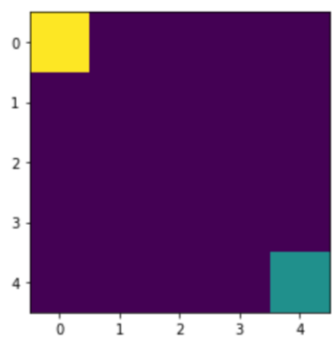
[4]

3 Understanding our Program Parameters

3.1 Environment

The environment we provide is a basic deterministic n x n grid-world environment (the initial state for an 4 X 4 grid-world is shown in Figure 3) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.

The environment's state space will be described as an n x n matrix with real values on the interval [0; 1] to designate different features and their positions. The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of +1 for moving closer to the goal and -1 for moving away or remaining the same distance from the goal.



3.2 Agent

The agent given here is the yellow box.

3.3 Goal

Goal: The agent has to learn Green box using Q-Learning.

4. Implementation of code

4.1 Policy

Theory**:**
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.

$$\pi\left(s_t\right) = \underset{a \in A}{\operatorname{argmax}} Q_\theta\left(s_t, a\right)$$

Implementation:

```
if np.random.uniform(0, 1) < epsilon:
        return np.random.choice(self.action_space.n)
 else:
        observation = observation.astype(int)
        return np.argmax(self.q_table[observation[0], observation[1]])
```

4.2 Update Q table

Theory:
We will use the given formula.

$$Q^{new}\left(s_t, a_t\right) \leftarrow (1-\alpha) \cdot \underbrace{Q\left(s_t, a_t\right)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q\left(s_{t+1}, a\right)}_{a}\right)}^{\text{learned value}}$$

Implementation:

```
state = state.astype(int)
next_state = next_state.astype(int)
self.q_table[state[0],state[1], action] = (self.q_table[state[0],state[1], action]) + (self.lr
*(reward + (self.gamma * np.max(self.q_table[next_state[0], next_state[1]])) -
self.q_table[state[0],state[1],action]))
```

4.3 Define Training

Theory:

A Reinforcement Learning task is about training an Agent which interacts with its Environment. The Agent transitions between different scenarios of the Environment, referred to as states, by performing actions. Actions, in return, yield rewards, which could be positive, negative or zero. The Agent's sole purpose is to maximize the total reward it collects over an episode, which is everything that happens between an initial state and a terminal state. Hence, we reinforce the Agent to perform certain actions by providing it with positive rewards, and to stray away from others by providing negative rewards. This is how an Agent learns to develop a strategy, or a policy.[5]

Implementation:

```
for episode in range(episodes):
    obs = env.reset()
    done=False
    total_reward = 0

    print('Episode: {episode}')
    print('Epsilon: {agent.epsilon}')
    epsilons.append(agent.epsilon)
    while not done:
        action = agent.step(obs)
        state = copy.deepcopy(obs)
        obs, reward, done, info = env.step(action)
        total_reward += reward
```

```
        next_state = copy.deepcopy(obs)
        agent.update(state, action, reward, next_state)

    agent.set_epsilon(agent.epsilon - (5/episodes)*agent.epsilon)
    # agent.set_epsilon(agent.epsilon - delta_epsilon)
    total_rewards.append(total_reward)
```

5 Understanding the 3 agents

5.1 Random Agent

This is the agent which randomly selects the next action to do. In maximum cases, this agent will never reach the goal.

5.2 Hueristic Agent

Heuristic is a technique of solving a problem, normally used as an aid to learning or discovery by experimental and especially trial-and-error methods.

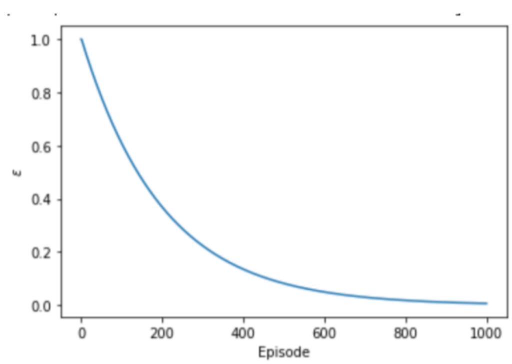
5.3 Q Learning Agent

As learnt in the above sections, we will use the policy defined, and update the Q table to learn the next state and action for the Q-Learning agent.

6 Result and Graphs For Q Learning Agent

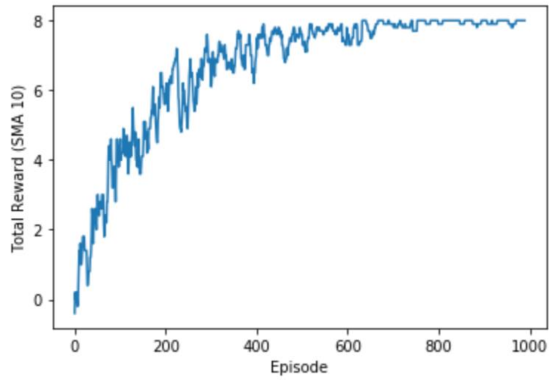
Now we will learn how to implement these models on our dataset.

6.1 Episode vs Epsilon



As we can the epsilon exponentially decays with each episode.

6.2 Episode and Total Reward:



.

As we can see towards the end the agent finally learns and achieves a good reward.

6.3 Actual Result

The agent was able to reach the goal in 9 steps.

7 Results and Conclusion

- Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed[6].
- We can see in our code, that the agent is able to reach the goal in 9 steps after being trained for some episodes. This means that the agent has slowly learned a better path to reach the goal.
- There are some practical applications of Reinforcement Learning. Reinforcement learning algorithms can be built to reduce transit time for stocking as well as retrieving products in the warehouse for optimizing space utilization and warehouse operations. Reinforcement learning is used to solve the problem of Split Delivery Vehicle Routing[7].

References

[1] https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[2] https://pythonistaplanet.com/pros-and-cons-of-reinforcement-learning

[3] https://skymind.ai/wiki/deep-reinforcement-learning

[4]https://www.youtube.com/watch?v=mo96Nqlo1L8&list=PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv&index=7

[5]https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677

[6] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[7] https://www.quora.com/What-are-some-practical-applications-of-reinforcement-learning