National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual – 11

Course Code: AI-2002	**Course: Artificial Intelligence Lab**
Instructor(s):	**Kariz Kamal, Saeeda Kawal, Sania Urooj, Shakir Hussain, Omer Qureshi, Ali Fatmi**

Objectives:

- Introduction to Reinforcement learning
- Different Reinforcement learning algorithms
- OpenAI gym environment
- Implementation of SARSA and Q-learning in python

Introduction

Reinforcement Learning is a part of machine learning. Here, agents are self-trained on reward and punishment mechanisms. It's about taking the best possible action or path to gain maximum rewards and minimum punishment through observations in a specific situation. It acts as a signal to positive and negative behaviors. Essentially an agent (or several) is built that can perceive and interpret the environment in which is placed, furthermore, it can take actions and interact with it. Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.

Since there is no labeled data, so the agent is bound to learn by its experience only.

RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.

The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.

It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.

Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.

The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.

The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.

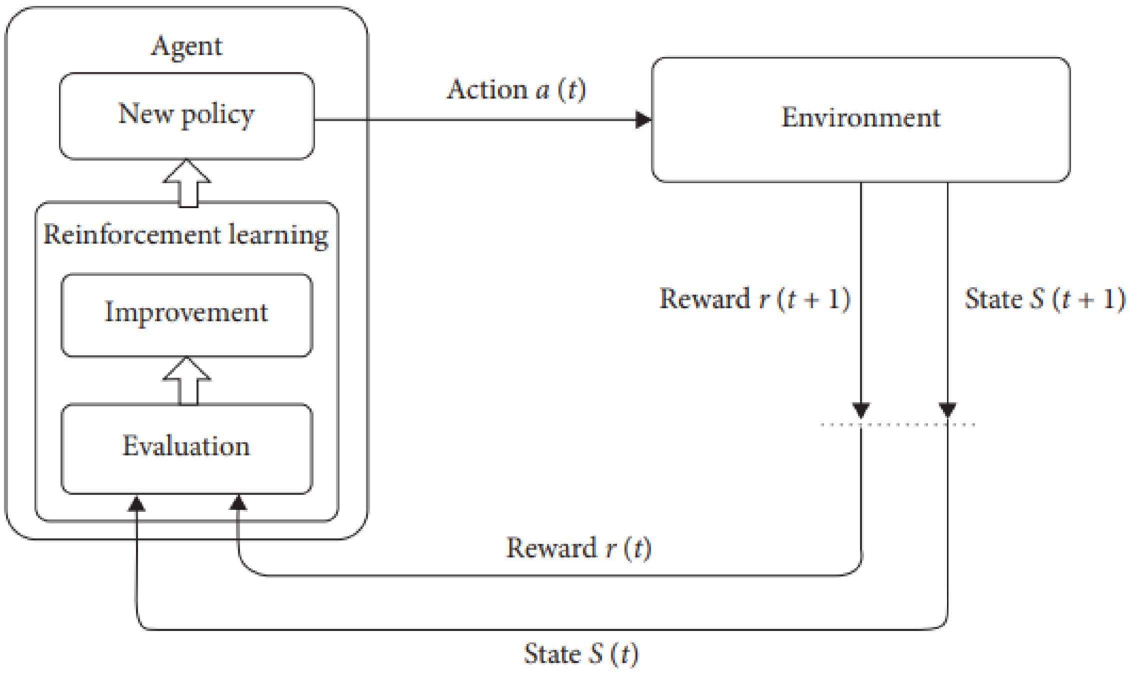


Figure 1: Simple reinforcement learning model.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

- Policy
- Reward Signal
- Value Function
- Model of the environment

1) Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For deterministic policy: $a = \pi(s)$

For stochastic policy: $\pi(a \mid s) = P[At = a \mid St = s]$

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the immediate signal for each good and bad action, whereas a value function specifies the good state and action for the future. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward. The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems with the help of the model are termed as the model-based approach. Comparatively, an approach without using a model is called a model-free approach.

Common reinforcement learning algorithms

The field of reinforcement learning is made up of several algorithms that take somewhat different approaches. The differences are mainly due to their strategies for exploring their environments.

- **State-action-reward-state-action (SARSA)**. This reinforcement learning algorithm starts by giving the agent what's known as a policy. The policy is essentially a probability that tells it the odds of certain actions resulting in rewards, or beneficial states.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- **Q-learning.** This approach to reinforcement learning takes the opposite approach. The agent receives no policy, meaning its exploration of its environment is more self-directed.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- **Deep Q-Networks.** These algorithms utilize neural networks in addition to reinforcement learning techniques. They utilize the self-directed environment exploration of reinforcement learning. Future actions are based on a random sample of past beneficial actions learned by the neural network.

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- Positive Reinforcement
- Negative Reinforcement

Positive Reinforcement: The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Negative Reinforcement: It is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

Python toolkit for RL

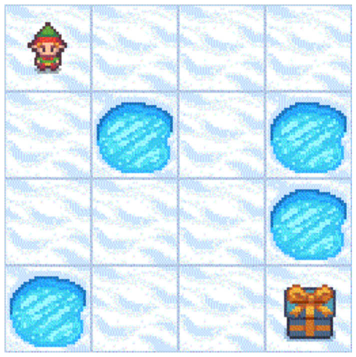
OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a set of environments, which are simulations of various games, puzzles, and other scenarios, where agents can interact with the environment and learn how to perform better over time using reinforcement learning techniques.

The Gym toolkit provides a standardized interface for working with different environments, which makes it easier to compare and evaluate different reinforcement learning algorithms. It also provides a set of tools for visualizing the agent's performance and for recording and analyzing the results of different experiments.

Example: implementation of SARSA algorithm using the OpenAI's gym (frozen-lake)

Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake.



```
import numpy as np

import gym

# Create the environment

env = gym.make('FrozenLake-v1')


# Set the hyperparameters

alpha = 0.1

gamma = 0.99

num_episodes = 5000


# Initialize the Q-table

num_states = env.observation_space.n

num_actions = env.action_space.n

Q = np.zeros((num_states, num_actions))Q = np.zeros((env.observation_space.n, env.action_space.n))
```

```
# Run the Q-learning algorithm
for i in range(num_episodes):
    # Reset the environment
    state = env.reset()
    done = False
    total_reward = 0

    while not done:
        # Choose an action using the epsilon-greedy policy
        if np.random.uniform(0, 1) < 0.5:
            action = env.action_space.sample() # random action
        else:
            action = np.argmax(Q[state, :]) # action with maximum Q-value

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

        # Update the Q-value of the (state, action) pair
        Q[state, action] = Q[state, action] + alpha * (reward + gamma * np.max(Q[next_state, :]) - Q[state, action])

        # Update the state and the total reward
        state = next_state
        total_reward += reward

    # Print the total reward obtained during the episode
    print(f"Episode {i}: Total reward = {total_reward}")
```

```
# Test the learned policy
num_test_episodes = 100
num_test_steps = 100
num_successes = 0

for i in range(num_test_episodes):
    state = env.reset()
    done = False
    steps = 0

    while not done and steps < num_test_steps:
        # Choose the action with the highest Q-value
        action = np.argmax(Q[state, :])

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

        # Update the state and the step count
        state = next_state
        steps += 1

    if state == 15:
        num_successes += 1

print("Success rate:", num_successes/num_test_episodes)
```

Tasks

1. Implement Q-learning algorithm using OpenAI gym environment.
2. The Smart cab's job is to pick up the passenger at one location and drop them off in another.

   The agent should receive a high positive reward for a successful drop off because this behavior is highly desired

   The agent should be penalized if it tries to drop off a passenger in wrong locations

   The agent should get a slight negative reward for not making it to the destination after every time-step.

   The passenger can be in one of the four possible locations: R, G, Y, B, which are represented in row, column coordinates as (0,0), (0,4), (4,0), (4,3) respectively. Additionally, we need to consider a fifth state where the passenger is already inside the taxi. Therefore, the number of possible states for the passenger's location is 5.

   The destination can be one of the four possible locations: R, G, Y, B, which are also represented in row, column coordinates. Therefore, the number of possible states for the destination is 4.

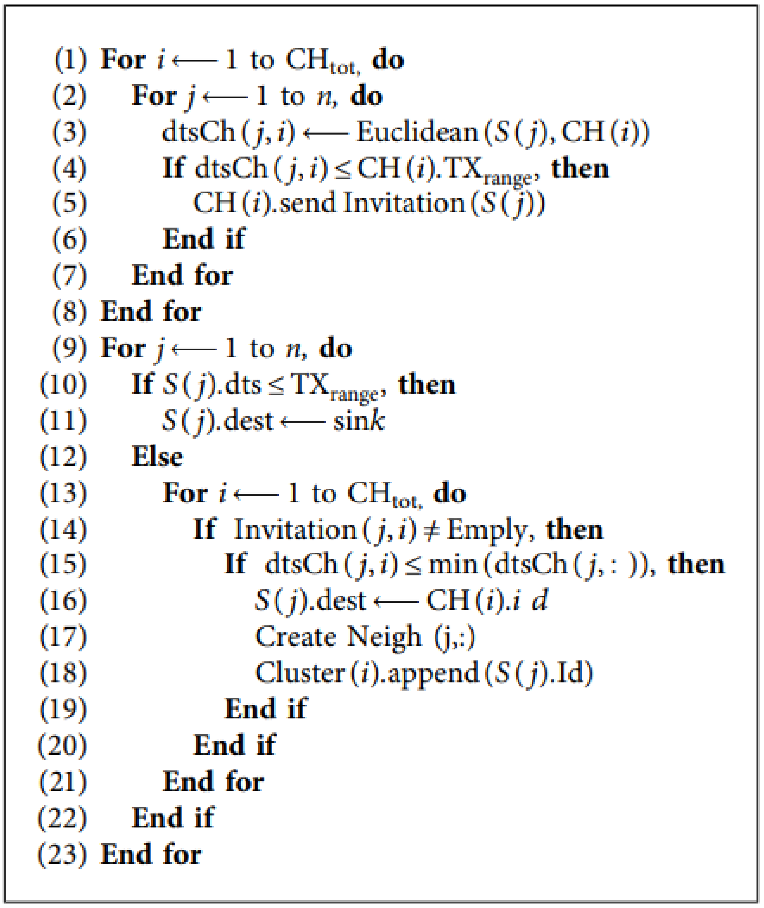
   We have six possible actions:

   1. south
   2. north
   3. east
   4. west
   5. pickup
   6. drop off

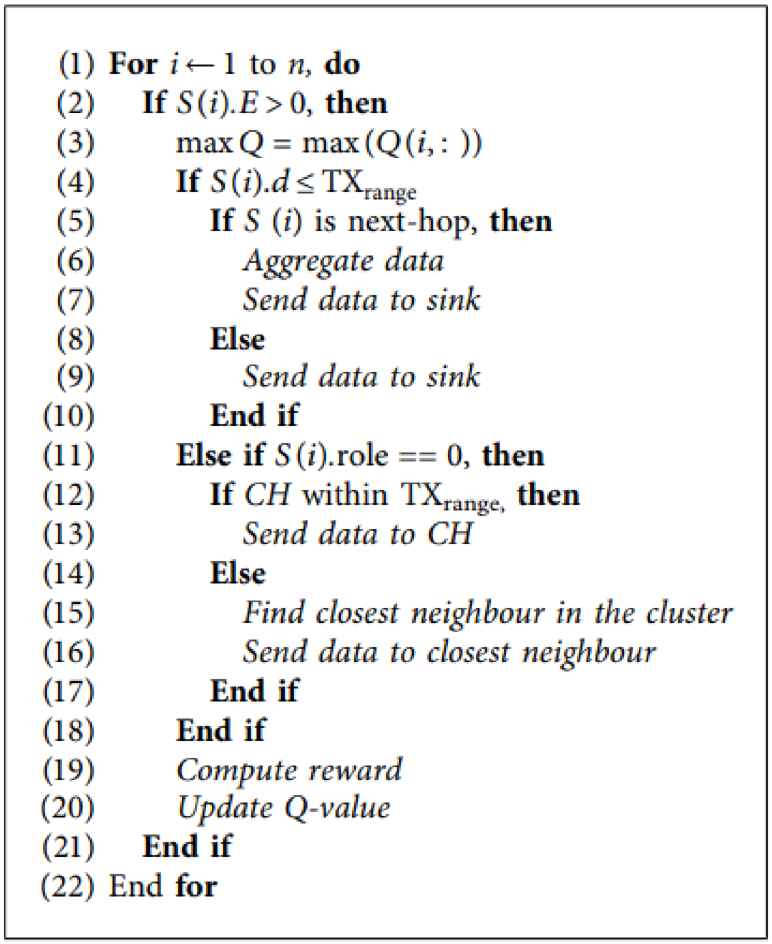
   Implement the above problem using Gym environment called **Taxi-V2.**

3. A well equipped state of the art hospital monitors data of the patients using AI devices remotely. Each patient has different sensors deployed on his/her body to acquire the values of sugar, Blood Pressure etc. Day and night expelling of the data requires infinite lifetime which is the ideal situation and nearly impossible due to DC devices deployment but to increase its lifetime we can go for certain AI techniques e.g reinforcement learning by measuring its nearest path to the sink from the cluster head. Develop such algorithm to increase the network lifetime upto maximum extent using Reinforcement learning. Your code must contain the cost factor while calculating the reward function for the next state.

Pseudo codes are given below for your reference only. You are not allowed to produce the same thing as shown in algorithm below.



```
(1)  For $i \longleftarrow 1$ to $CH_{tot,}$ do
(2)      For $j \longleftarrow 1$ to $n$, do
(3)          $dtsCh(j,i) \longleftarrow Euclidean(S(j), CH(i))$
(4)          If $dtsCh(j,i) \le CH(i).TX_{range}$, then
(5)              $CH(i).send\ Invitation(S(j))$
(6)          End if
(7)      End for
(8)  End for
(9)  For $j \longleftarrow 1$ to $n$, do
(10)     If $S(j).dts \le TX_{range}$, then
(11)         $S(j).dest \longleftarrow sink$
(12)     Else
(13)         For $i \longleftarrow 1$ to $CH_{tot,}$ do
(14)             If $Invitation(j,i) \ne Emply$, then
(15)                 If $dtsCh(j,i) \le \min(dtsCh(j,:))$, then
(16)                     $S(j).dest \longleftarrow CH(i).i\ d$
(17)                     Create Neigh (j,:)
(18)                     $Cluster(i).append(S(j).Id)$
(19)                 End if
(20)             End if
(21)         End for
(22)     End if
(23) End for
```



```
(1)  For $i \leftarrow 1$ to $n$, do
(2)    If $S(i).E > 0$, then
(3)      $\max Q = \max(Q(i, :))$
(4)      If $S(i).d \leq \mathrm{TX}_{\mathrm{range}}$
(5)        If $S(i)$ is next-hop, then
(6)          Aggregate data
(7)          Send data to sink
(8)        Else
(9)          Send data to sink
(10)       End if
(11)     Else if $S(i)$.role == 0, then
(12)       If CH within $\mathrm{TX}_{\mathrm{range}}$, then
(13)         Send data to CH
(14)       Else
(15)         Find closest neighbour in the cluster
(16)         Send data to closest neighbour
(17)       End if
(18)     End if
(19)     Compute reward
(20)     Update Q-value
(21)   End if
(22) End for
```