

# Reinforcement Learning

By Dr. Narayana

# Overview

- Types of learning
- Reinforcement Learning
- RL Examples - How humans learn using Reinforcement Learning?
- How Reinforcement learning works?
- Reinforcement learning process
- Reinforcement learning framework
- Q-learning
- Q-learning Examples
- Exploration Vs. Exploitation
- RL Observations
- SARSA
- Q-Learning vs SARSA
- Summary

# Learning outcome

- Understand the limitations of classical machine learning techniques and how reinforcement learning addresses them
- Understanding of scenarios where RL can be applied.
- Understanding of various RL models.

# Types of Learning

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

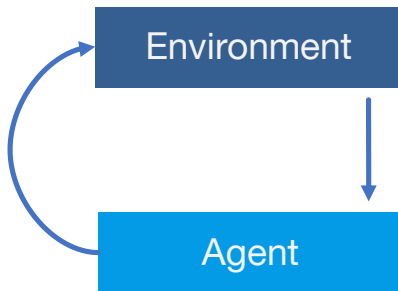
# Supervised and Unsupervised learning

- Data:  $(x, y)$  where  $x$  is feature space and  $y$  is target
- Goal: Learn a function to map  $x$  to  $y$
- Techniques:
  - Classification
  - Regression
  - Object detection

- Data:  $(x)$  where  $x$  is feature space, just feature space, no labels
- Goal: Understand data and learn its hidden structures.
- Techniques:
  - Dimensionality reduction
  - Clustering
  - Feature learning

# Reinforcement learning

- More general than supervised and unsupervised learning.
- Agents gets some reward or penalty as an outcome on interacting with the environment.
- The goal is to maximize the reward, and take only those actions that will help achieve that.



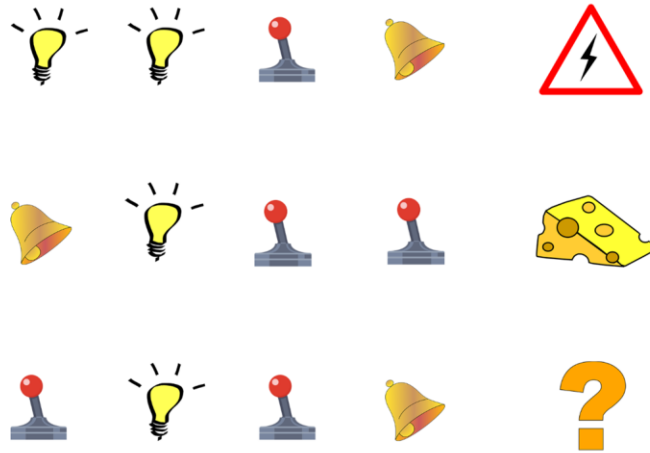
*Reinforcement learning can be considered as the science of decision making*

# RL Example: Teach dog new tricks

- A dog does not know our language, so how do train the dog say to bring the ball that is thrown in a direction
- Give some clue to the dog to do some action
  - If the dog does the desired action reward him with food.
  - Next time if the same situation is given, the dog will do the same action with more enthusiasm for getting more food.
  - It is an example of learning to do things from positive experiences.
- Say the dog does something undesirable, we punish the dog with some negative experience. Dogs tend to avoid them so as to avoid the negative experience ( say Isolation)
- One can also control by magnitude and frequency of rewards and punishments to inculcate decided behavior

# RL Example: Teach cat new tricks

- What if the agent state equals to last 3 items in sequence?
- What if the agent state equal to counts for lights, bells, and livers?
- What if the agent state equals to complete sequence?



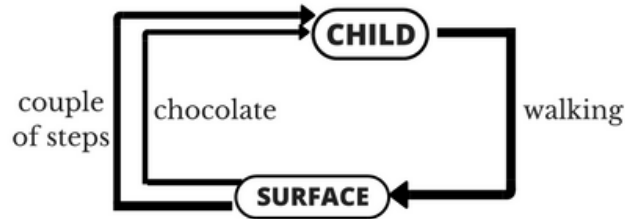


# RL Example: Baby learning how to walk

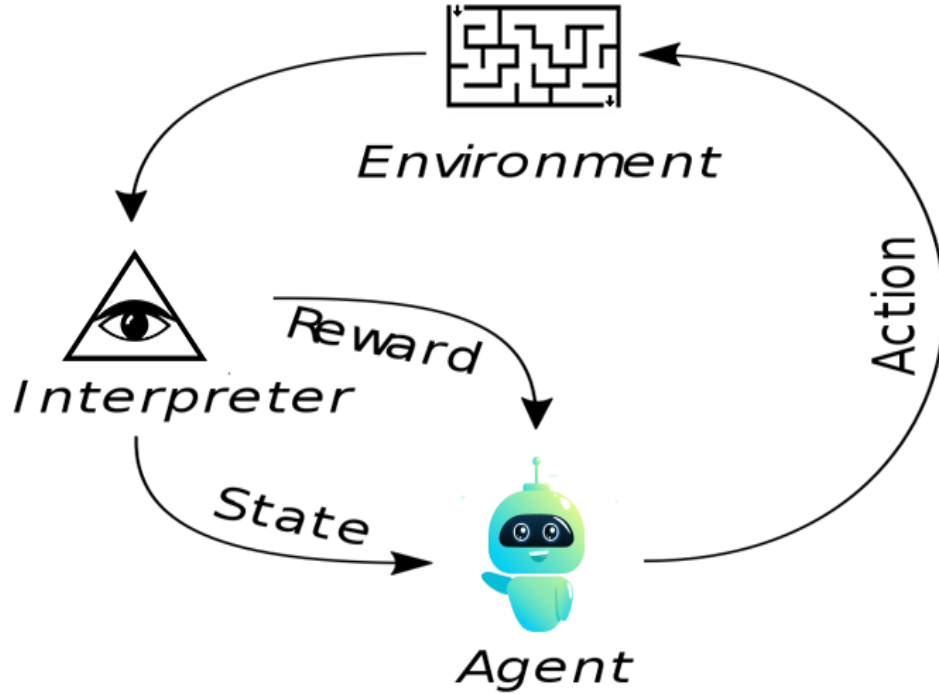
Let's look an example of a child learning to walk. Here child is an agent and the surface on which he/she is walking is the environment.

For every correct step, child is rewarded a chocolate. No reward will be given if the child doesn't move forward.



























In terms of RL, every step can be a sub task, a chocolate is a positive reward, no chocolate is a negative reward. Child's movement can show movement from one state to another state.



# How RL works?



# Example: Blood Bank Ops Optimization

| Recipient Blood Types | Donor Blood Types |   |   |   |  |   |   |   |
|-----------------------|-------------------|---|---|---|--|---|---|---|
|                       | O-                | O+  | B-  | B+  | A-   | A+  | AB-   | AB+   |
|                       | AB+               |  |  |  |  |  |  |  |
|                       | AB-               |  |   |  |  |  |   |  |
|                       | A+                |  |  |   |  |  |  |   |
|                       | A-                |  |   |   |  |  |   |   |
|                       | B+                |  |  |  |  |   |   |   |
|                       | B-                |  |   |  |  |   |   |   |
|                       | O+                |  |  |   |  |   |   |   |
|                       | O-                |  |   |   |  |   |   |   |

# RL Process



# RL Process

Reinforcement learning is an experience based decision making technique. The process involves below steps:

|         |      |   |
|---------|------|---|
| Observe | ---> | environment of the agent                        |
| Decide  | ---> | decision as per the observation                 |
| Act     | ---> | action on the decision                          |
| Receive | ---> | getting rewarded or penalised as per the action |
| Learn   | ---> | from previous actions and improve               |
| Iterate | ---> | repeat the entire process till success          |

# Main components in RL process: Rewards

Reward is a scalar feedback signal (single number). Basically, it conveys the overall progress of the agent. Agent's task is to maximize the reward.

- Drop/pickup a passenger from a driverless car
  - +/- ve reward for correct/wrong location
- Fly stunt maneuvers in a fighter plane
  - +ve reward for following the desired trajectory
  - -ve reward for crashing
- Defeat a world class chess player
  - +/- ve reward for victory/defeat
- Manage a bank portfolio
  - +ve/-ve reward for profit/loss
- Create a walking robot
  - +ve reward if it walks in right direction
  - -ve reward if it falls
- Beat world level Go player
  - +/- ve reward for victory/defeat

# Main components in RL process: Goals and Actions

Goal is to maximize the rewards by taking actions according to the policy. Goals depend upon the time.

RL algorithm's goal can be to get an instant rewards or it can leave some rewards to achieve a long-term goal.

Example: In chess, an agent can sacrifice some pieces for doing checkmate on the opponent.

# Main components in RL process: State

- State contains all the information about the environment that can be helpful in deciding the next steps. For example a frame in a game of chess defines the opponent's situation, your own situation and the location of possible rewards.

## Agent state and environment state:

- Agent state is private to that particular agent. Algorithms use the agent's state for choosing actions.
- Agent does some action in the environment and gets rewards or penalties depending upon the action. Agent may or may not be aware of the environment state.



# Main components in RL process: Policy

Policy is the rules followed by the agent to get the maximum rewards. It can also be described as the behaviour of the agent.

The policy can be deterministic or stochastic (probability of taking some action for a given state).

# Main components in RL process: Model

Agent represents the environment in the form of a Model. It helps in figuring out a plan for the next steps by learning about the environment. It can be divided into two states:

**Transition:** It is used to predict the future state. e.g., if the position or velocity of an object is given it can predict what will the environment do next

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

The above equation tells us that given  $s$  and  $a$  what would be  $s'$ . Where  $s$ ,  $a$  and  $s'$  are the current state, action and the probability of being in the next state respectively.

**Reward:** It predicts the immediate reward. Following an action, it predicts the reward the object would get.

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

The above equation tells us that given  $s$  and  $a$  what would be  $R$ . Where  $s$ ,  $a$  and  $R$  are the current state, action and the expected reward respectively.

# RL Framework

Reinforcement learning can be applied to a very large set of problems, ranging from teaching a computer to play a game to understanding certain chemical reactions.

But the major problem is to get started as there is a serious lack of simulated learning environments available for experimentation.

# RL Framework

**OpenAI Gym:** *It is python package containing several reinforcement learning environments. Ranging from basic to advanced. Also includes video games and robotics simulated environments.*

The aim of developing this was to have a standard environment and benchmark for research in RL.



# Q-Learning

Q-learning is a model-free learning method.

Also known as an off-policy learning method.

The agent's action can be random and still it can find an optimal policy, there is no requirement to follow a specific policy, that is the meaning of off-policy.

# Q-learning: Algorithm

|       |   | ACTION |    |    |    |    |     |
|-------|---|--------|----|----|----|----|-----|
| STATE |   | 0      | 1  | 2  | 3  | 4  | 5   |
| R=    | 0 | -1     | -1 | -1 | -1 | 0  | -1  |
|       | 1 | -1     | -1 | -1 | 0  | -1 | 100 |
|       | 2 | -1     | -1 | -1 | 0  | 0  | -1  |
|       | 3 | -1     | 0  | 0  | -1 | 0  | -1  |
|       | 4 | 0      | -1 | -1 | 0  | -1 | 100 |
|       | 5 | -1     | 0  | -1 | -1 | 0  | 100 |

For an environment, we build a table called Q-table which has the dimensions  $S \times A$  where  $S$  and  $A$  are the numbers of states and actions respectively. For every state, there are actions and the likeliness of choosing a particular action depends on the values in Q-table called state-action value.

Initially, the values of the Q-table are 0. An action is chosen for a state.

Q-value is increased for the state-action if that action gives a good reward for the next state, else it is decreased.

# Q-learning: Algorithm

Q-values are updated using this equation:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \left( \underbrace{\overbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)}^{\text{learned value}}}_{\text{reward} + \text{discount factor} \times \text{estimate of optimal future value}} - \underbrace{Q(s_{t+1}, a_t)}_{\text{old value}} \right)$$

Where the previous state and action are  $s(t)$  and  $a(t)$  and  $s(t+1)$  and  $r(t+1)$  are the current state and reward. The learned value is the target and the old value is the prediction and the difference between them is the error. We then fix the old value using error with the learning rate.

# Q-learning: Algorithm

Algorithm main steps:

1. Initialize Q for all states and actions.
2. For N episodes, follow steps from 3–9:
3. Initialize state, S.
4. Follow the steps from 5-8 for each step of the episode
5. Choose an action A, from state S, with some policy derived from Q
6. Now take action, A and get new state S' and reward R.
7. For S and A, update the Q value from the above equation
8. Set  $S = S'$ , as the current state
9. Terminate if the state S is the terminal.

The key thing in Q-learning is the learned value or target.

The optimal future value estimate is calculated as the max of all the actions in state S'.

Suppose for a state S, you have two possible actions A1 and A2, then the action with a greater Q-value is chosen.

Action is chosen on a random basis if both have the same Q-values.



# Example: Driverless cab

Goal is to pick and drop the passenger from one location to another.

We would like our automated cab to :

- Drop the passenger to the right location.
- Take minimum time possible to drop off
- Ensure passenger safety and abide by traffic rules

# Example: Driverless cab : Rewards

Here an imaginary cab driver can be considered as an agent whose focus is on maximizing the rewards and it will learn about the environment and will gain control on the cab through trial and error.

So, we can decide the magnitude of rewards and penalties. Some points on that:

- High rewards for dropping off the passenger successfully.
- Penalty on dropping off on wrong locations.
- Magnitude of reward/ Penalty
  - As we would prefer our agent to reach with a few minute delay rather than making the wrong moves in order to reach the destination faster, the penalty will be "Slight" negative.

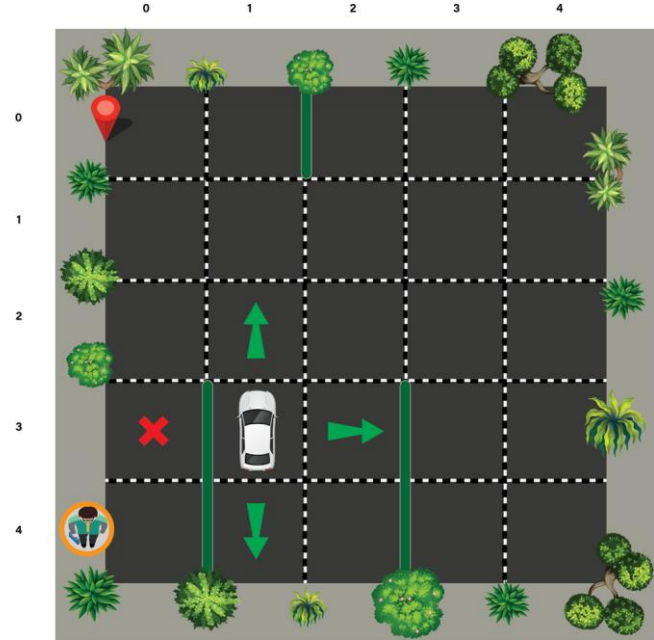
# Example: Driverless cab : States

Agent takes an action according to the current state it has encountered.

The set of all the circumstances the taxi can inhabit is referred to as the state space.

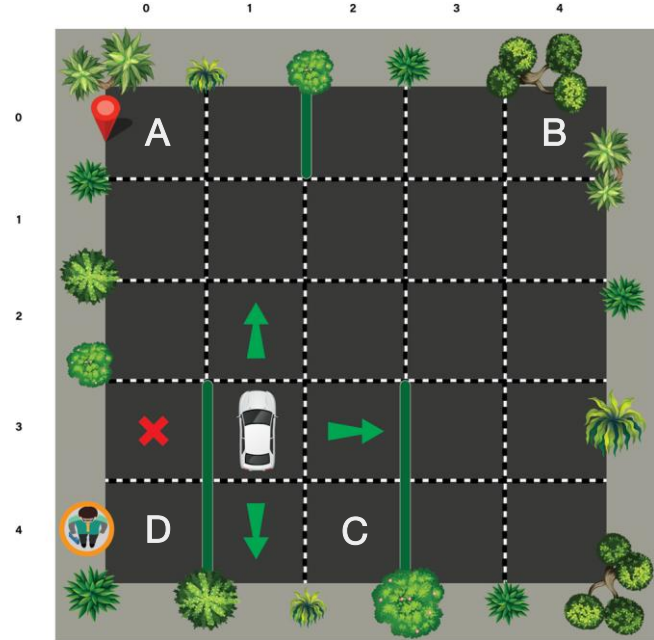
Agent makes the actions based on the information in the state. So, state needs to have all the necessary information.

For example: Let's train a cab for particular training area where we train it to reach four parking points A, B, C and D.



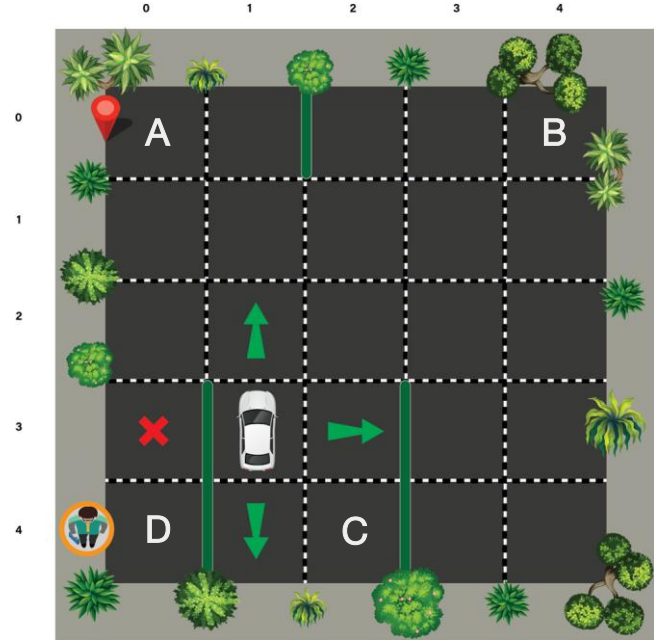
# Example: Driverless cab : States

- Say, only our Driverless Cab is in this parking lot
- The parking lot can be assumed to be made of 5 x 5 grid , giving 25 possible locations. The current location of our cab is (3,1)
- There are 4 locations where we can pick up/drop passenger A, B, C and D.
- Our passenger is in location D and needs to go to A
- Passenger states = 5 locations With Passenger state of being inside taxi.
- Total possible states =  $5 \times 5 \times 5 \times 4 = 500$



# Example: Driverless cab : Actions

- Agent considers one of the 500 states and takes action.
- Action in this case
  - Move in a direction or decide to pick up or drop off a passenger.
  - It can move in either of the 4 directions (north, east, west, south)
  - 6 possible actions
- Action Space - all possible actions that an agent can take.
- Some actions cannot be performed because of the walls, that is penalised in the environment code by -1.



# Example: Driverless cab : Q Table

Initialization

| Q-TABLE |     | ACTIONS   |           |          |          |            |             |
|---------|-----|-----------|-----------|----------|----------|------------|-------------|
|         |     | South (0) | North (1) | East (2) | West (3) | Pickup (4) | Dropoff (5) |
| STATES  | 0   | 0         | 0         | 0        | 0        | 0          | 0           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | 327 | 0         | 0         | 0        | 0        | 0          | 0           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | .   | .         | .         | .        | .        | .          | .           |
|         | 499 | 0         | 0         | 0        | 0        | 0          | 0           |

Training

| 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  |

# Example: Driverless cab

## Functions Utilized

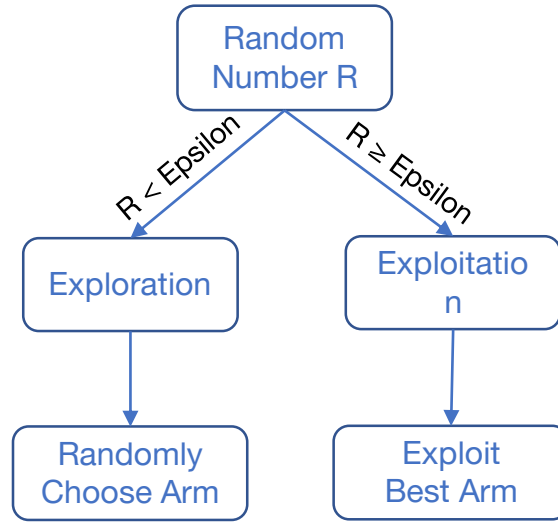
|                                |   |
|--------------------------------|---|
| <code>env.reset:</code>        | Returns a random initial state and also resets the environment.             |
| <code>env.step(action):</code> | Increases time step in the environment.                                     |
| <code>env.render:</code>       | This is helpful in visualization as it renders one frame of an environment. |

## Environment Returns

|              |  |
|--------------|--|
| Observation: | Environment-specific object representing your observation of the environment |
| Reward:      | Rewards achieved from the action done previously.                            |
| Done:        | Tells if environment need to be reset  |
| Info:        | Used for debugging   |

# Q-learning: Algorithm

In the Q learning algorithm in step 5, we can derive policy using the epsilon-greedy approach.





# Exploration Vs. Exploitation

In a treasure hunt game, every player tried to find the treasure as fast as possible. There is no surety of winning the game even if the player tries to explore many different paths and checks if the treasure is present there. The other strategy of choosing one path and exploring it to the depth is also not a sure shot way to win the game. So, the players need to try both the approaches, exploration as well as exploitation.

## EXPLOITATION



Playing the machine that (Currently) pays out the most.

## EXPLORATION



Playing the other machine to see if any pay out more.

Exploration refers to getting more details about the environment.

Exploitation aims for maximizing the reward by making use of the information which is already found.

Basically, there should be a balance between both the strategies so that we don't lose much on rewards.

# Exploration Vs. Exploitation

## Trekking

- Exploitation: Go on a predefined path
- Exploration: Try to explore a new path

## Movie recommendation

- Exploitation: Show the highest rated movie
- Exploration: Show a different movie

## Store Opening

- Exploitation: Open your store at the most popular location
- Exploration: Try a new location

## Playing Chess

- Exploitation: Play the move you know is best
- Exploration: Experiment with a new move

# RL Observations

RL lies in between the spectrum of Supervised and Unsupervised learning.

## **Being greedy is not always helpful**

There is no need to go for short-term results and instant rewards. Focus should be on optimizing the algorithm for maximizing rewards for a longer term.

## **Sequence is important**

Reward is time dependant as it depends both on current state and all the past states.

# SARSA Algorithm

SARSA is an on-policy algorithm where, in the current state,  $S$  an action,  $A$  is taken and the agent gets a reward,  $R$  and ends up in the next state,  $S_1$  and takes action,  $A_1$  in  $S_1$ . If you club up all these things in a tuple  $(S, A, R, S_1, A_1)$ , it stands for the acronym SARSA.

The policy is updated on the basis of actions taken, that's why it is referred to as an on-policy algorithm.

# SARSA Algorithm

Sarsa (on-policy TD control) for estimating  $Q \approx q_*$

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

    Repeat (for each step of episode):

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

    until  $S$  is terminal

# Q-learning Vs. SARSA

Q-learning: off-policy

- use any policy to estimate Q

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

- Q directly approximates  $Q^*$
- independent of the policy being followed
- only requirement: keep updating each (s,a) pair

SARSA: on-policy

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

The difference between these two algorithms is that SARSA chooses an action following the same current policy and updates its Q-values whereas Q-learning chooses the greedy action, that is, the action that gives the maximum Q-value for the state, that is, it follows an optimal policy.

# Summary

What Reinforcement learning does?

- Helps us to make decisions in new or uncertain conditions

We can use these algorithms

- Q-learning
- SARSA

Structure

- Algorithms simple
- Feature design
- Representation of state
- Rewards

Thank you! :)