Reinforcement Learning

Agenda

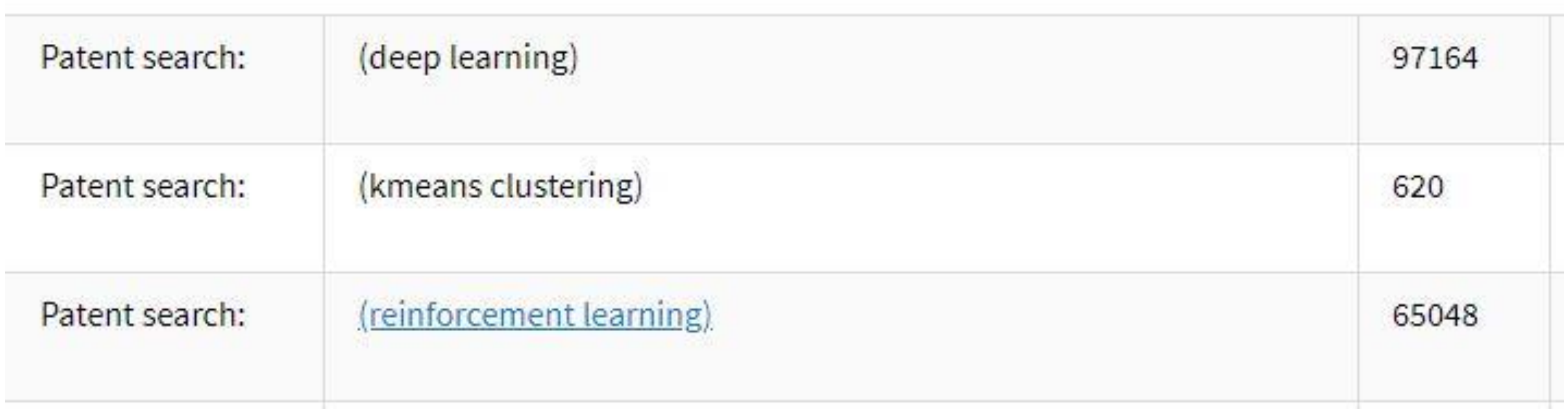
- What is reinforcement learning?
- Patents in different technologies
- SL vs. USL vs. RL
- Examples
- Introduction to RL
  - Important components
- Q-learning
- Case Study: Smart Taxi
- HandsOn1: Frozen Lake 4x4
- Exploration Vs. Exploitation
- Case Study: Frozen Lake 4x4
- HandsOn2: Frozen Lake 8x8
- SARSA
- Case Study: Stock Market - Optimal Order Execution

Learning Outcome

By the end of this module, you will be able to

- Explain the limitations of classical machine learning techniques and how reinforcement learning addresses them
- Describe scenarios in which reinforcement learning is applicable
- Explain various models in reinforcement learning

Patents analysis: DL Vs. Kmeans Vs. RL



Patent search:	(deep learning)	97164
Patent search:	(kmeans clustering)	620
Patent search:	(reinforcement learning)	65048

Patents analysis: DL Vs. Kmeans Vs. RL

https://www.lens.org/

https://patents.google.com/

Supervised Learning

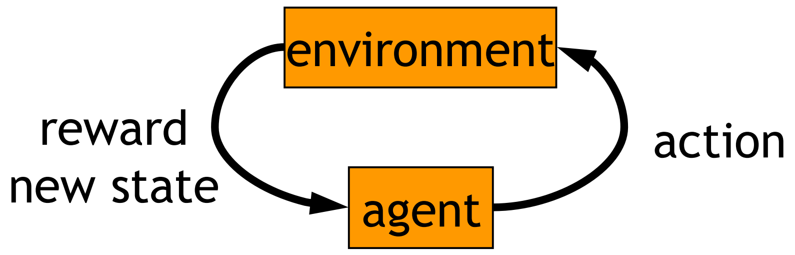
- Data: (x, y) where x is feature space and y is target
- Goal: Learn a function to map x to y
- Examples:
  - Classification
  - Regression
  - Object detection
  - Image captioning

Unsupervised Learning

- Data: (x) where x is feature space
  - Just feature space, no labels
- Goal: Learn some underlying hidden structure of the data
- Examples:
  - Clustering
  - Dimensionality reduction
  - Feature learning
  - Density estimation

Reinforcement Learning

- More general than supervised/unsupervised learning
- Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals
- Goal: Learn how to take actions in order to maximize reward



Reinforcement learning is the science of decision making

Example: Teach dog new tricks

Consider the scenario of teaching a dog new tricks. The dog doesn't understand our language, so we can't tell him what to do. Instead, we follow a different strategy. We emulate a situation (or a cue), and the dog tries to respond in many different ways. If the dog's response is the desired one, we reward them with snacks. Now the next time the dog is exposed to the same situation, the dog executes a similar action with even more enthusiasm in expectation of more food. That's like learning "what to do" from positive experiences. Similarly, dogs will tend to learn what not to do when faced with negative experiences.

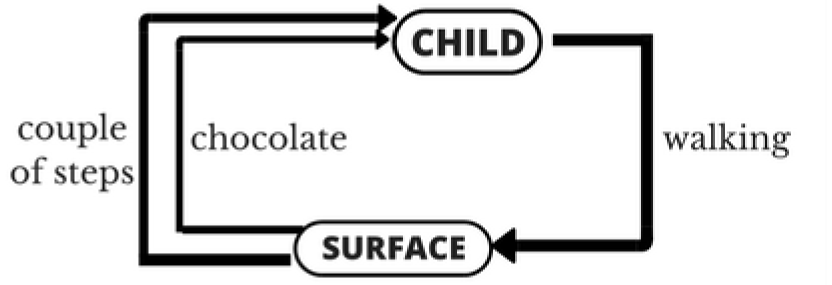
Teach dog new tricks

How Reinforcement Learning works in a broader sense:

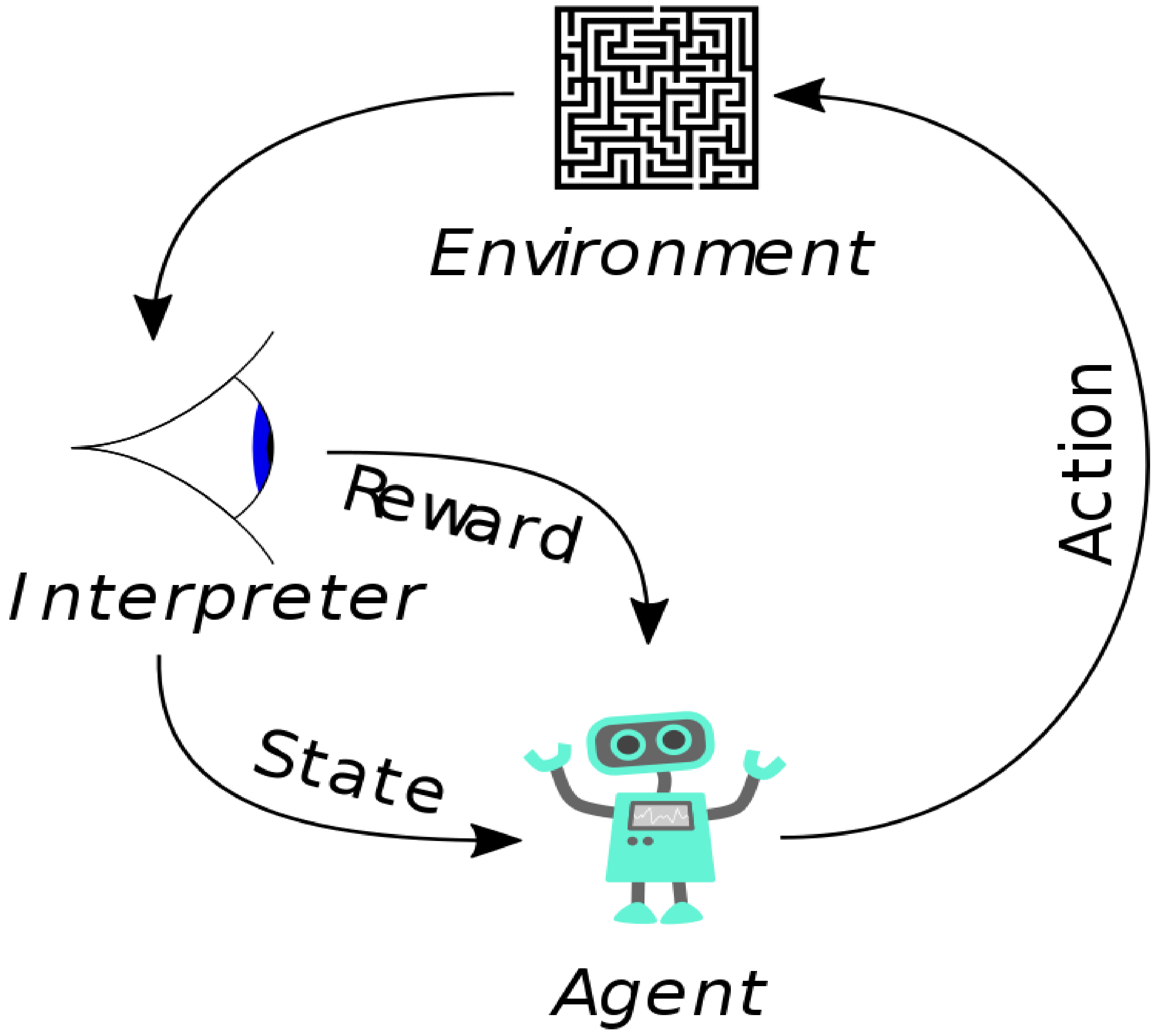
- Your dog is an "agent" that is exposed to the **environment**. The environment could in your house, with you.
- The situations they encounter are analogous to a **state**. An example of a state could be your dog standing and you use a specific word in a certain tone in your living room
- Our agents react by performing an **action** to transition from one "state" to another "state," your dog goes from standing to sitting, for example.
- After the transition, they may receive a **reward** or **penalty** in return. You give them a treat! Or a "No" as a penalty.
- The **policy** is the strategy of choosing an action given a state in expectation of better outcomes.

Example: Baby learning how to walk

- *The "problem statement" of **how to walk**, where **the child is an agent** trying to manipulate the **environment (which is the surface on which it walks)** by **taking actions (viz walking)** and he/she tries to go from one **state (viz each step he/she takes)** to another. The child gets a **reward (let's say chocolate)** when he/she accomplishes a **sub module of the task (viz taking couple of steps)** and will not receive any chocolate **(a.k.a negative reward)**when he/she is not able to walk.*



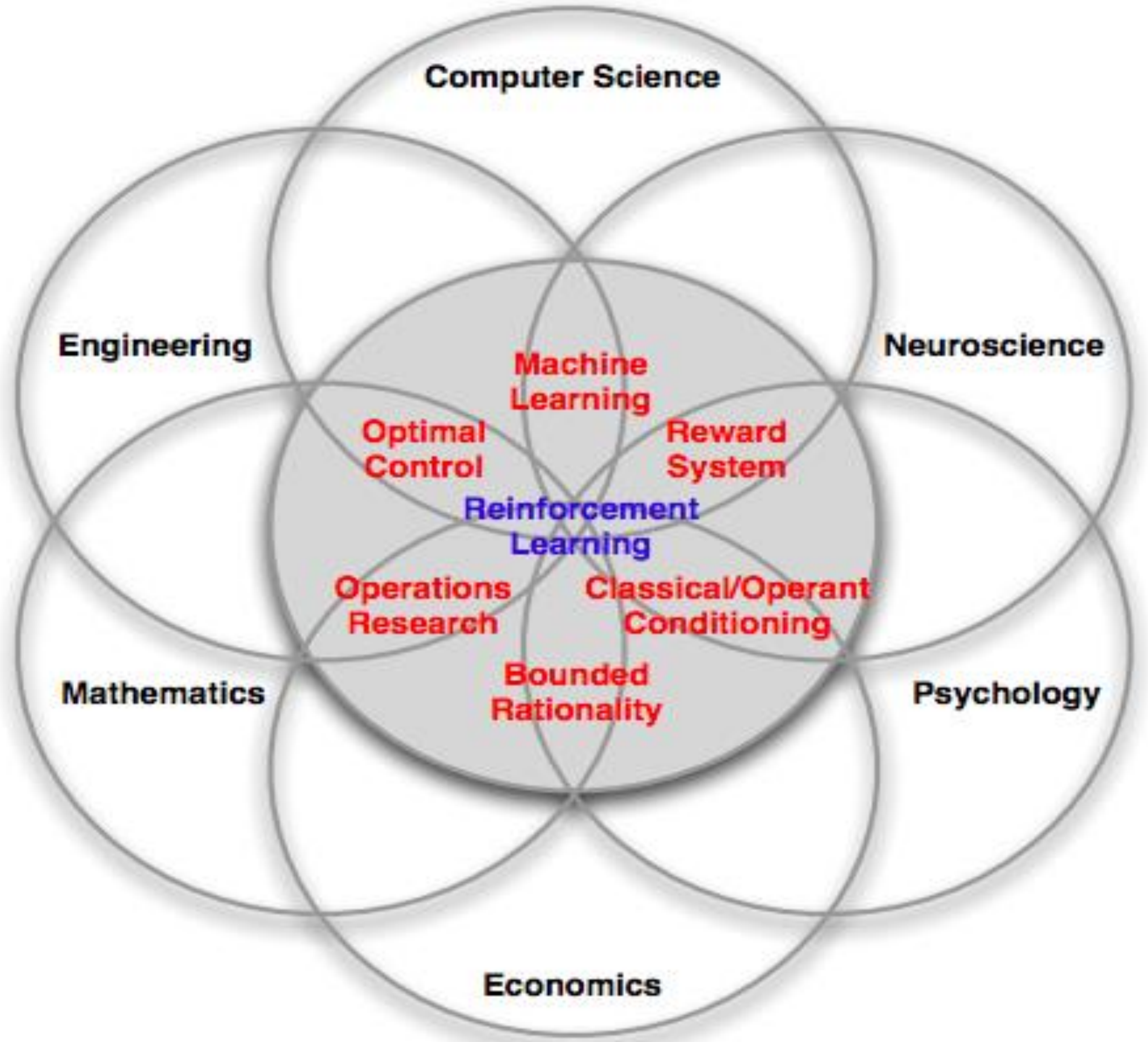
How RL works?



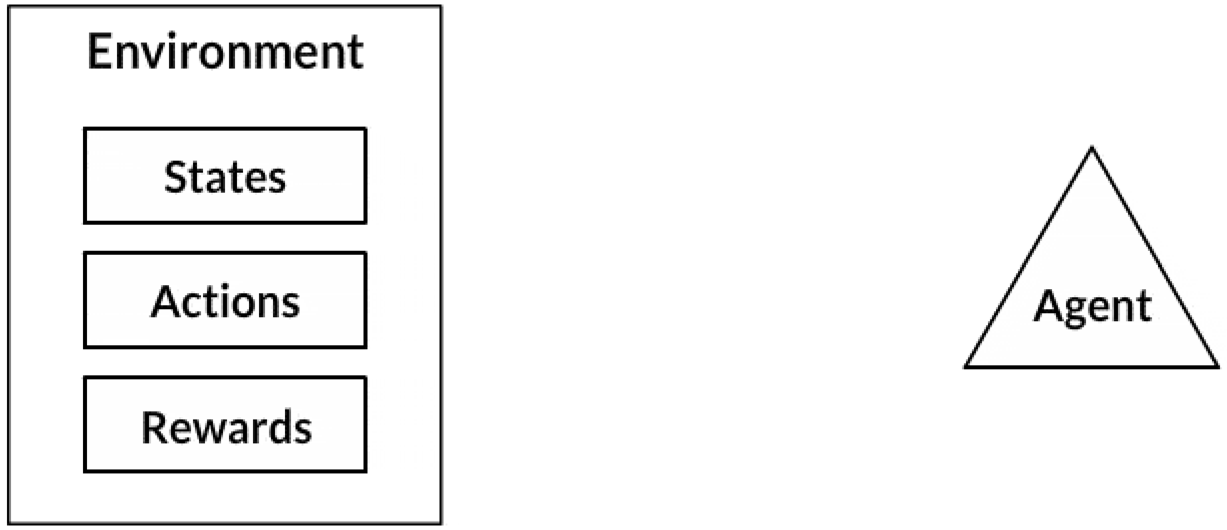
RL and other science domains

- RL is the intersection of various fields :
  - Engineering: Optimum Control (finding an optimal control point for a system).
  - Mathematics: Operations Research (deals with the application of advanced analytical methods to help make better decisions).
  - Economics: Game Theory, Utility Theory etc.
  - Neuroscience: The human brain has a remarkable ability for adapting to environmental changes.

RL and other science domains



Reinforcement Learning Process



Reinforcement Learning Process

- In a way, Reinforcement Learning is the science of making optimal decisions using experiences. Breaking it down, the process of Reinforcement Learning involves 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

Main components in RL process: Rewards

It is a scalar feedback signal(single number) which tells us that how well the agent is doing and the agent tries to maximise the total reward accumulated over time

Main components in RL process: Rewards

Examples

- A dog receives a reward from a trainer for completing a certain task and no reward for failing to do so
- Fly stunt manoeuvres in a helicopter (https://www.youtube.com/watch?v=0JL04JJjocc)
  - +ve reward for following desired trajectory
  - -ve reward for crashing
- Defeat the world champion at Backgammon
  - +/- ve reward for winning/losing a game
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - -ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - -ve reward for falling over
- Play many dierent Atari games better than humans
  - +/- ve reward for increasing/decreasing score

Main components in RL process: Goals and Actions

- The goal of reinforcement learning problem is to maximise the total reward by taking certain actions with respect to policy. It is time dependent. The goal could be to get an immediate reward or sacrificing the immediate rewards to reach a long-term goal.
- Example: Agent working on a long-term strategy in the game of chess where it sacrifices some pieces to do a checkmate on the opponent.

Main components in RL process: State

- It is the summary of the information that is used to determine what happens next that is, the current situation in an environment. Like a frame in a Pacman game is a state which describes the opponents location, ones own location, the location of the rewards etc.
- *Agent and environment state:*
  - The agent’s state is private to itself. The information in agent’s state is used by RL algorithms to pick actions.
  - The environment gets an action from the agent, emits observations and rewards to the agent. The environment state may or may not be visible to the agent and even be useless for the agent.

Main components in RL Agent: Policy

The policy is the behaviour of the agent. It is a set of rules that the agent follows to get the most reward. It maps the state to an action. It can be deterministic or it can be stochastic(for a given state, the probability of taking some action).

Main components in RL Agent: Model

It is the agent’s representation of the environment. It helps to learn about the environment and then figure out a plan for what to do next. It can be divided into two states:

- **Transition**: It predicts the next state. e.g., given the dynamics such as the velocity, position of an object what will the environment do next.

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

This equation tells us the probability of being in the next state, **s’** given the current state, **s** and action, **a**.

- **Reward**: It predicts the immediate reward. Or how much reward will the object get following an action.

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

This equation tells us the expected reward, **R** we will get given the current state, **s** and action, **a**.

RL Framework

OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms.

Q-learning

- Q- Learning is an **off-policy**, **model-free** learning method. Off-policy means, it does not require to follow a specific policy, the agent’s actions could be random and despite this, it can find an optimal policy.

Q-learning: Algorithm

For an environment, we build a table called Q-table which has the dimensions SxA where S and A are the number 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 as **state-action value**.

Initially the values of the Q-table are 0. An action is chosen for a state. If that action gives a good reward for the next state then the Q-value for the state-action is increased otherwise, the Q-value is decreased.

$$R = \begin{array}{c|rrrrrr} & \multicolumn{6}{c}{\text{Action}} \\ \text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Q-learning: Algorithm

- The updation of the Q-values are done using the following equation:

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

Here **s(t)** and **a(t)** are the previous state and action 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 this error with some amount called as **learning rate**.

Q-learning: Algorithm

Algorithm Main Steps:

1. Initialise Q for all states and actions.
2. For N episodes, follow steps from 3–9:
3. Initialise state, S.
4. For each step of the episode follow steps from 5–8:
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. Update the Q value for S and A using the above equation.
8. Set S as the current state (S = S’).
9. Terminate if the state S is the terminal.

The key thing in Q-learning is the learned value or target. The estimation of the optimal future value 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. If both have the same Q-values then an action is chosen randomly.

Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically

Example: Smart Taxi

The Smart Taxi's task is to pick up the passenger at one location and drop them off in another. Here are a few things that we like our Smart Taxi to take care of:

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

Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically

Rewards

Since the agent (the imaginary driver) is reward-motivated and is going to learn how to control the cab by trial experiences in the environment, we need to decide the **rewards** and/or **penalties** and their magnitude accordingly. Here a few points to consider:

- The agent should receive a high positive reward for a successful dropoff 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. "Slight" negative because we would prefer our agent to reach late instead of making wrong moves trying to reach to the destination as fast as possible

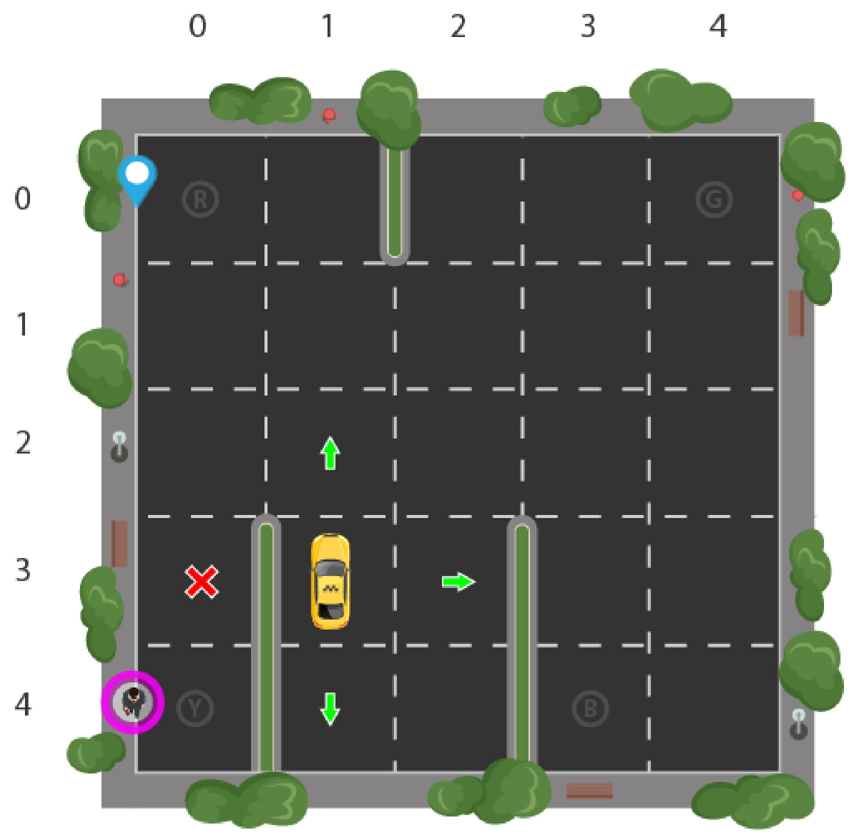
Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically

States

In Reinforcement Learning, the agent encounters a state, and then takes action according to the state it's in.

The **State Space** is the set of all possible situations our taxi could inhabit. The state should contain useful information the agent needs to make the right action.

Let's say we have a training area for our Smart Taxi where we are teaching it to transport people in a parking lot to four different locations (R, G, Y, B):



Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically

States

Let's assume Smart Taxi is the only vehicle in this parking lot. We can break up the parking lot into a 5x5 grid, which gives us 25 possible taxi locations. These 25 locations are one part of our state space. Notice the current location state of our taxi is coordinate (3, 1).

You'll also notice there are four (4) locations that we can pick up and drop off a passenger: R, G, Y, B or [(0,0), (0,4), (4,0), (4,3)] in (row, col) coordinates. Our illustrated passenger is in location Y and they wish to go to location R.

When we also account for one (1) additional passenger state of being inside the taxi, we can take all combinations of passenger locations and destination locations to come to a total number of states for our taxi environment; there's four (4) destinations and five (4 + 1) passenger locations.

So, our taxi environment has $5×5×5×4=500$ total possible states.

Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically

Actions

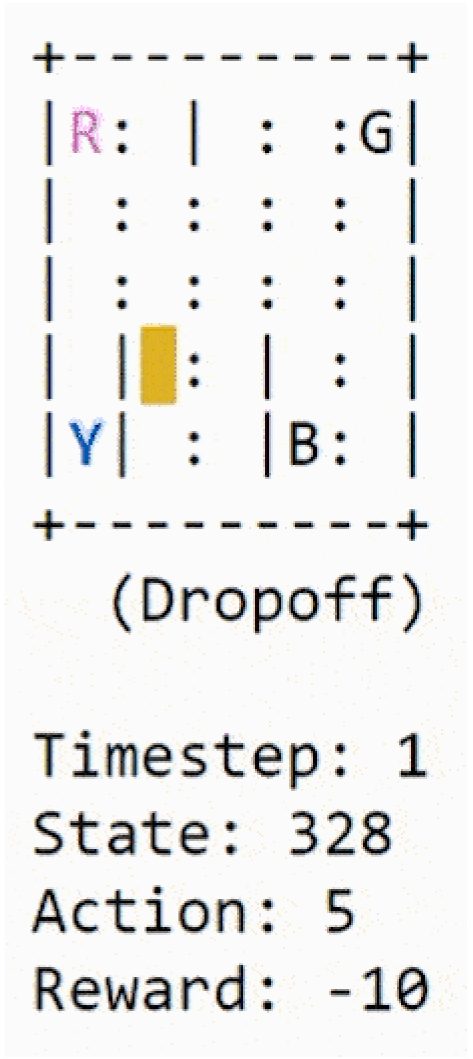
The agent encounters one of the 500 states and it takes an action. The action in our case can be to move in a direction or decide to pickup/dropoff a passenger.

In other words, we have six possible actions: south, north, east, west, pickup, dropoff

This is the action space: the set of all the actions that our agent can take in a given state.

We observe that the taxi cannot perform certain actions in certain states due to walls. In environment's code, we will simply provide a -1 penalty for every wall hit and the taxi won't move anywhere. This will just rack up penalties causing the taxi to consider going around the wall.

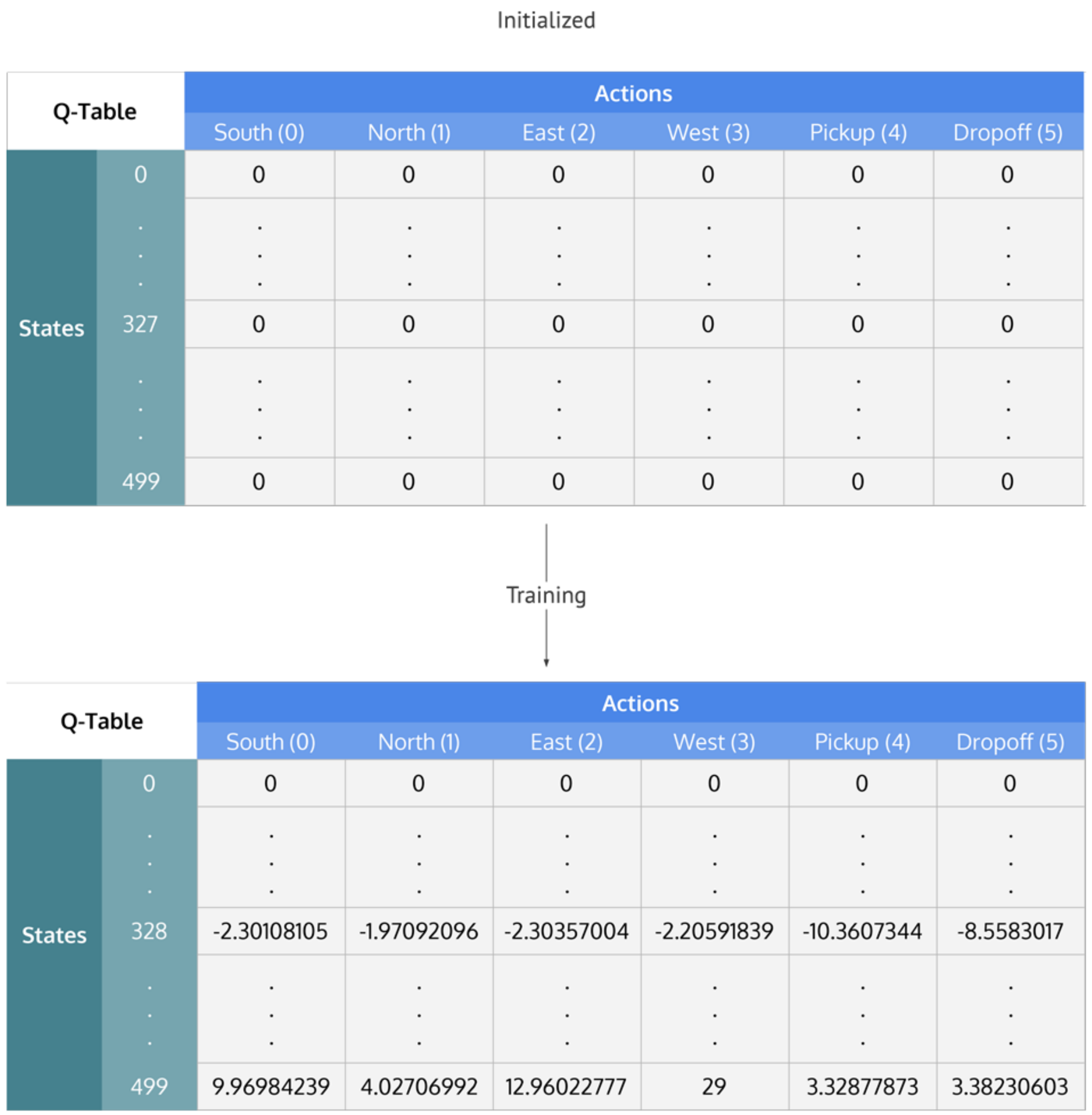
Develop an agent to learn Taxi Driver to pickup passenger and drop passenger automatically



```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Timestep: 1
State: 328
Action: 5
Reward: -10
```

Q-table


Initialized
Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
States	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

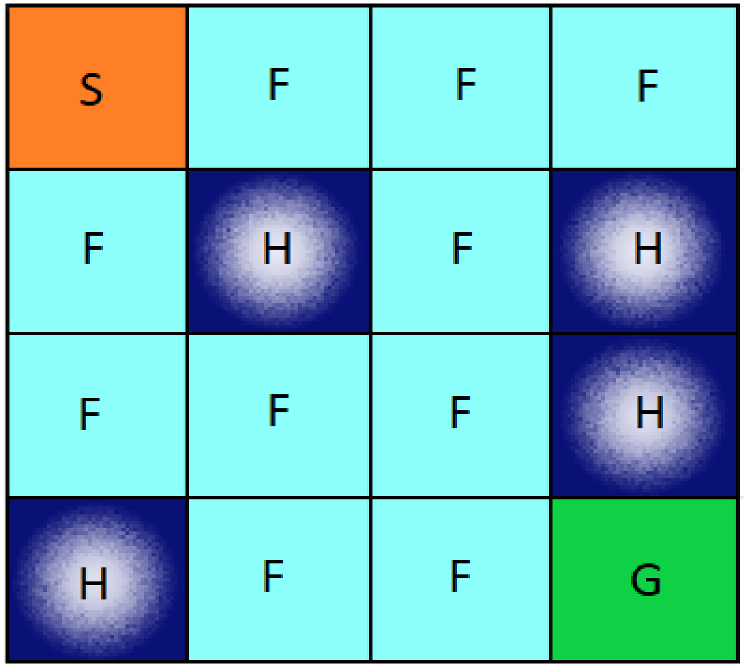
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
States	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Develop an agent to learn to walk in a frozen lake automatically on its own

HandsOn: Frozen Lake

Example: Frozen Lake

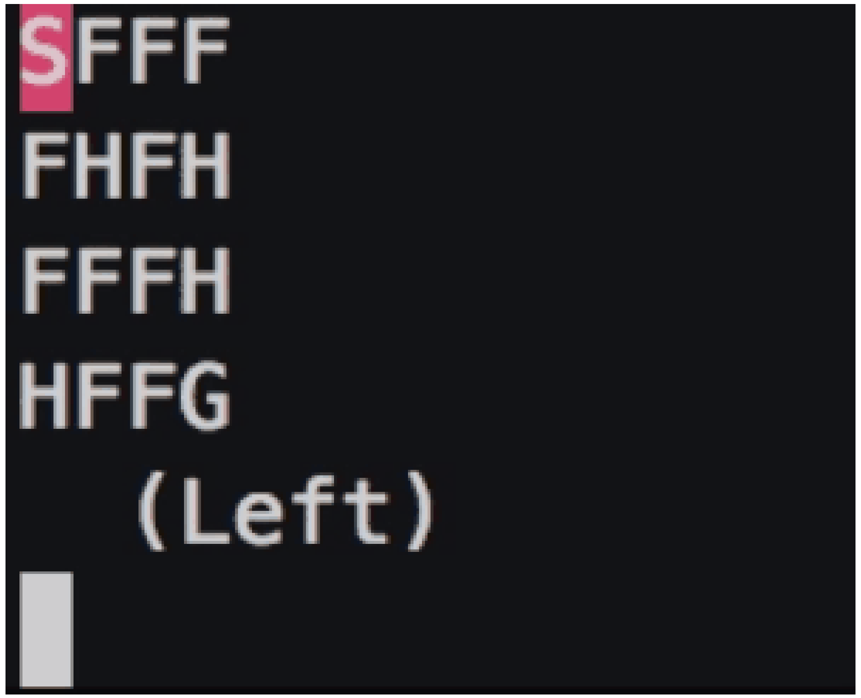
- Imagine, you are standing on a frozen lake. The lake is not all frozen, there are some parts where the ice is very thin. You goal is to go from place S to G without falling into the holes
- Here, S is the starting point, G is the goal, F is the solid ice where the agent can stand and H is the hole where if the agent goes, it falls down.
- The agent has 4 possible moves which are represented in the environment as 0, 1, 2, 3 for left, right, down, up respectively.
- For every state F, the agent gets 0 reward, for state H it gets -1 reward as in state H the agent will die and upon reaching the goal, the agent gets +1 reward.



Example: Frozen Lake

- The game upon rendering in the terminal looks like this

- The states here are F, S and G. That is there are 4x4=16 states and 4 actions.



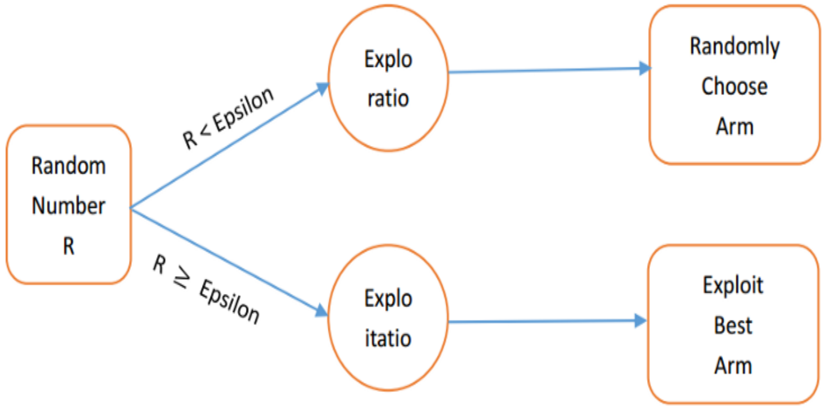
```
SFFF
FHFH
FFFH
HFFG
  (Left)
```

Example: Frozen Lake

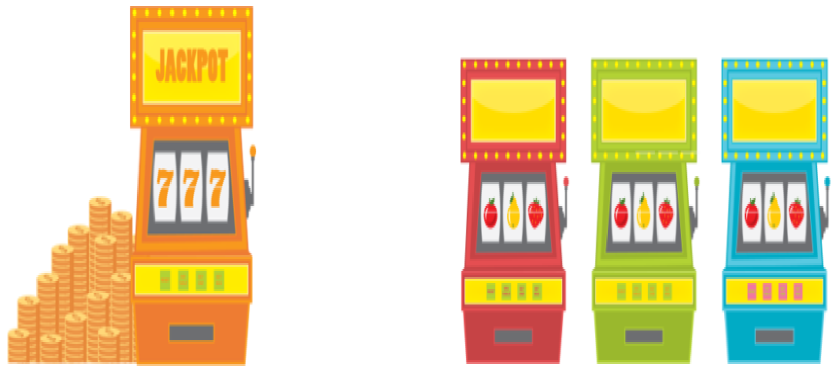
- Derive an optimal policy to reach the goal in lesser time

Q-learning: Algorithm

- In the above algorithm at step 5, we can derive policy using epsilon-greedy approach. Let’s understand it using an example.
- Suppose you are presented two doors, D1 and D2 and behind one of them there is a gift for you. You chose the door D2 randomly and didn’t get any gift. The next time you will choose door D1 as D2 was a failure. This time you got the gift. Now you saw that the door D1 has more probability of giving a gift and you will be choosing it every time, not exploring the possibility of a gift behind D2. Same in Q-learning we are choosing the max state-action value without exploring other actions. For solving this we generate a random value and check if it’s smaller than epsilon. If it is then we choose an action randomly and otherwise we choose the max state-action value.



Exploration Vs. Exploitation



EXPLOITATION

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

EXPLORATION

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

Suppose in a treasure-hunt game, our goal is to reach to the treasure as fast as possible and we go on and try different paths. If we try many different paths, explore them, see if the treasure is there or not then this won't guarantee that we would win the game and also if we choose a path and go on exploring it to the depth, then this strategy will also not guarantee that we will win the game. The best approach would be a mixture of both. Exploit and also explore a little bit.

Exploration means to get more information about the environment

Exploitation means to make use of the information already found to maximise the reward

This tradeoff of exploring and exploiting in an RL problem needs to be balanced without losing much reward.

Exploration Vs. Exploitation

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

Example: Frozen Lake

- Derive an optimal policy using Q-learning to reach the goal in lesser time
- Compare optimal policy derived using random approach verses optimal policy derived using Q-learning

RL Observations

- Reinforcement Learning lies between the spectrum of Supervised Learning and Unsupervised Learning, and there's a few important things to note:
  - **Being greedy doesn't always work**
    There are things that are easy to do for instant gratification, and there's things that provide long term rewards. The goal is to not be greedy by looking for the quick immediate rewards, but instead to optimize for maximum rewards over the whole training.
  - **Sequence matters in Reinforcement Learning**
    The reward agent does not just depend on the current state, but the entire history of states. Unlike supervised and unsupervised learning, time is important here.

Hands-On: Smart Taxi

- Smart Taxi

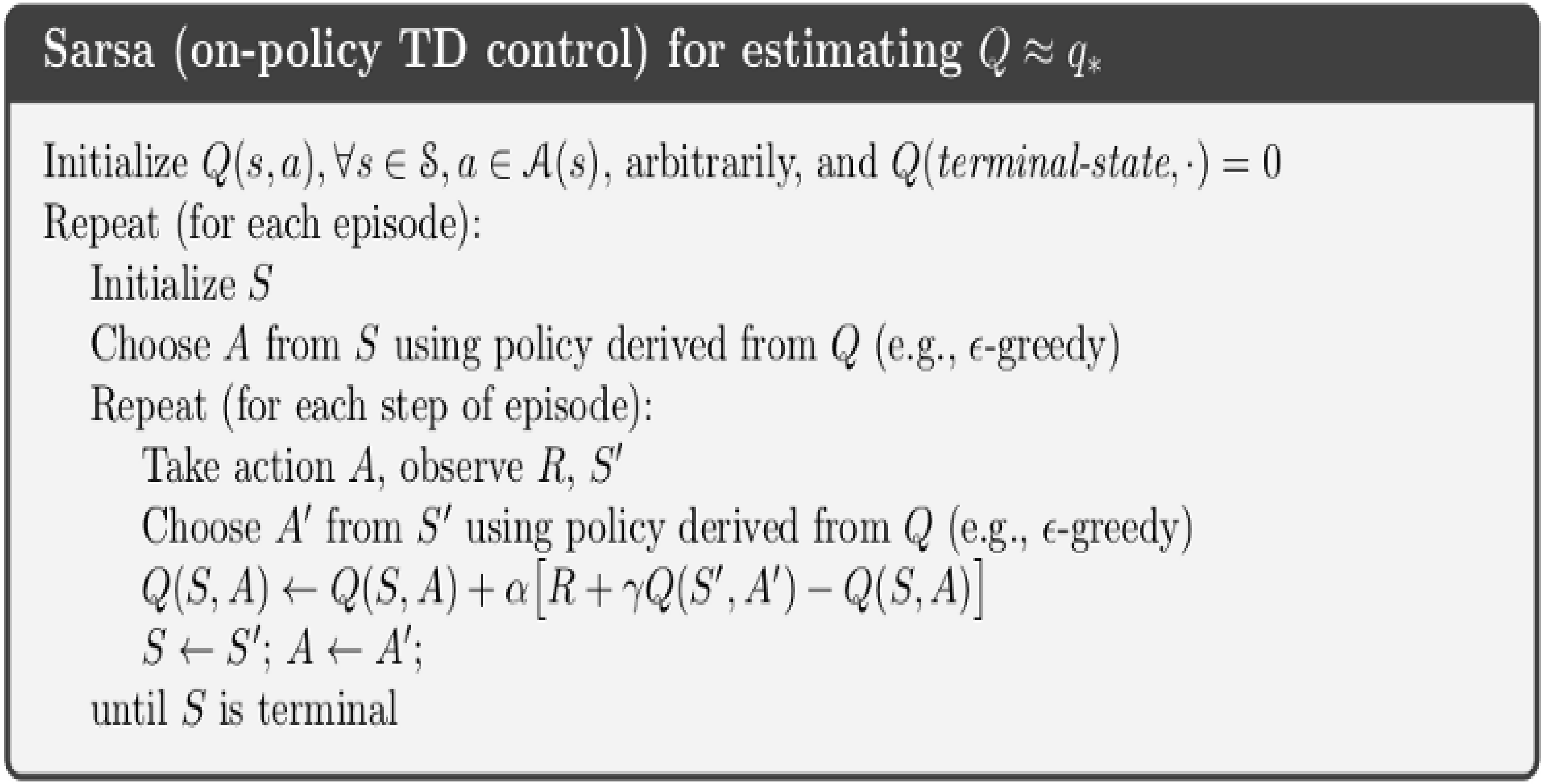
Hands-On: Frozen Lake 8x8

- Frozen Lake 8x8

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 next state, S1 and takes action, A1 in S1. Therefore, the tuple (S, A, R, S1, A1) stands for the acronym **SARSA**.
- It is called an on-policy algorithm because it updates the policy based on actions taken.

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 $Q(\textit{terminal-state}, \cdot) = 0$
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)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    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

- Reinforcement learning
  - use when need to make decisions in uncertain environment
- Solution methods
  - Q-learning
  - SARSA
- Most work
  - algorithms simple
  - need to design features, state representation, rewards

Questions?