Data-intensive space engineering

Lecture 10

Carlos Sanmiguel Vila

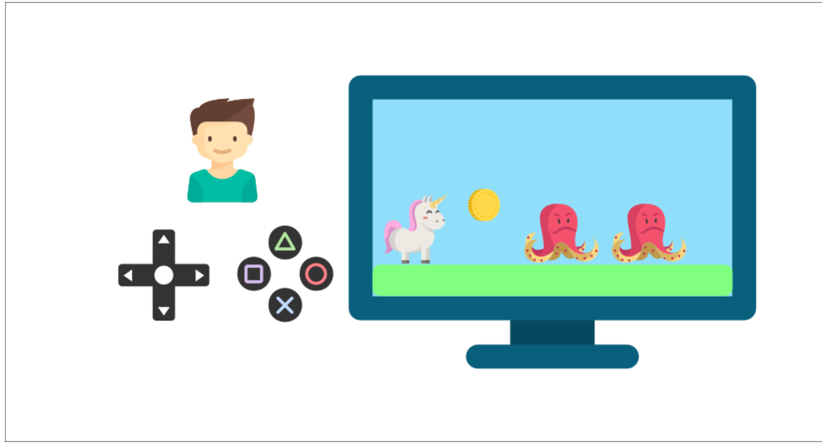
Based on Deep RL Course from Hugging Face

What is Reinforcement Learning?

- The idea behind Reinforcement Learning is that an agent (an AI) will learn from the environment by interacting with it (through trial and error) and receiving rewards (negative or positive) as feedback for performing actions.

- Learning from interactions with the environment comes from our natural experiences.

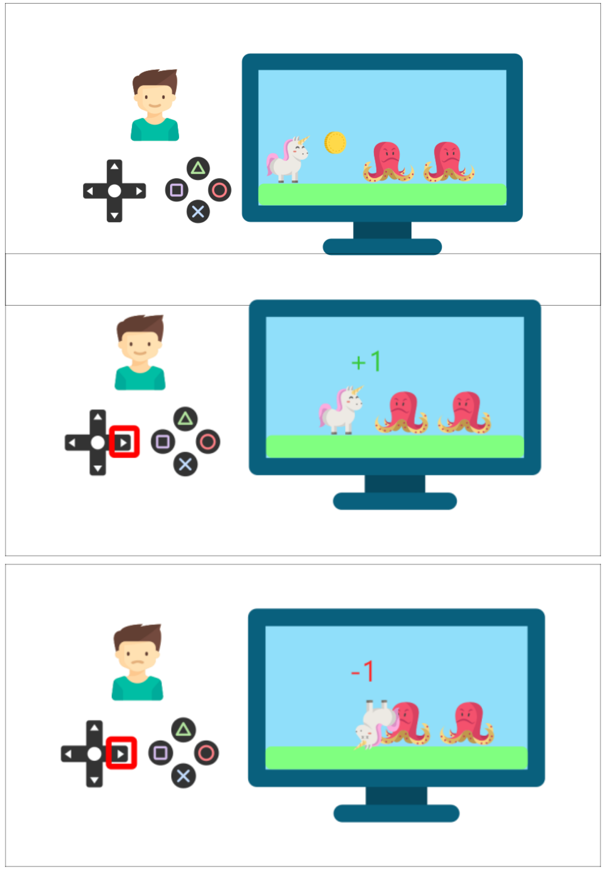
- For instance, imagine a child in front of a video game he never played. Giving him a controller and leaving him alone.



What is Reinforcement Learning?

- He will interact with the environment (the video game) by pressing the right button (action). He got a coin, that’s a +1 reward. It’s positive, he just understood that in this game **he must get the coins**. But then, **he presses the right button again** and he touches an enemy. He just died, so that’s a -1 reward.
- By interacting with his environment through trial and error, your little brother understands that he needs to get coins in this environment but avoid the enemies. Without any supervision, the child will get better and better at playing the game.

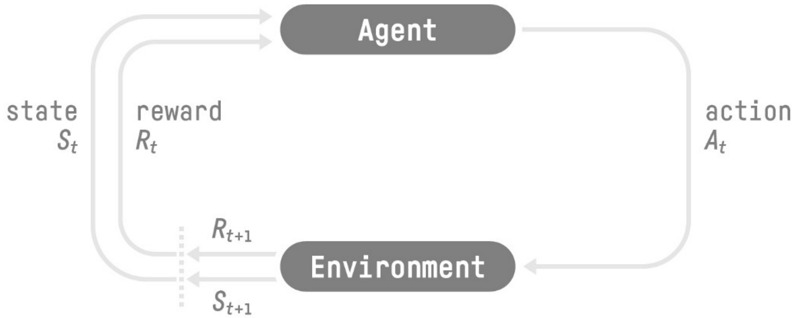
- That’s how humans and animals learn, through interaction. Reinforcement Learning is just a computational approach of learning from actions.



What is Reinforcement Learning?

We can now make a formal definition:

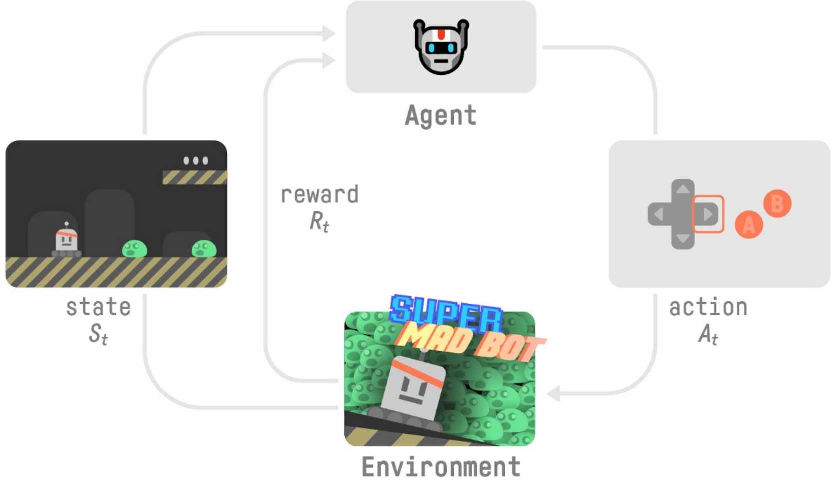
Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.



The RL Process

To understand the RL process, let’s imagine an agent learning to play a platform game:

- Our Agent receives **state *S0*** from the **Environment** — we receive the first frame of our game (Environment).
- Based on that **state *S0*,** the Agent takes **action *A0*** — our Agent will move to the right.
- The environment goes to a **new state *S1*** — new frame.
- The environment gives some **reward *R1*** to the Agent — we’re not dead *(Positive Reward +1)*.



The RL Process

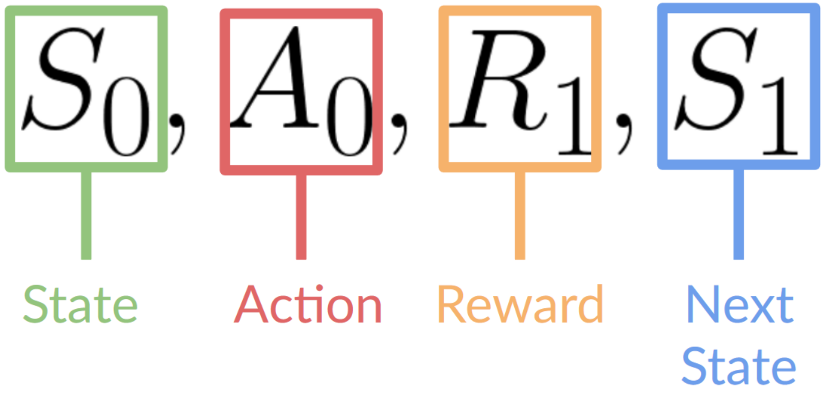
This RL loop outputs a sequence of **state, action, reward and next state.**

The agent’s goal is to maximize its cumulative reward, **called the expected return.**

Why is the goal of the agent to maximize the expected return?

- Because RL is based on the reward hypothesis, which is that all goals can be described as the maximization of the expecte return (expected cumulative reward).

- That’s why in Reinforcement Learning, to have the best behavior, we aim to learn to take actions that maximize the expected cumulative reward.



The RL Process

In papers, you’ll see that the RL process is called a **Markov Decision Process** (MDP).

We’ll talk again about the Markov Property in the following slides. But if you need to remember something today about it, it’s this: the Markov Property implies that our agent needs **only the current state to decide** what action to take and **not the history of all the states and actions** they took before.

Observations/States Space

Observations/States are the **information our agent gets from the environment**. In the case of a video game, it can be a frame (a screenshot). In the case of the trading agent, it can be the value of a certain stock, etc.

There is a differentiation to make between *observation* and *state*, however:

- State s: is **a complete description of the state of the world** (there is no hidden information). In a fully observed environment.

- In a chess game, we have access to the whole board information, so we receive a state from the environment. In other words, the environment is fully observed.

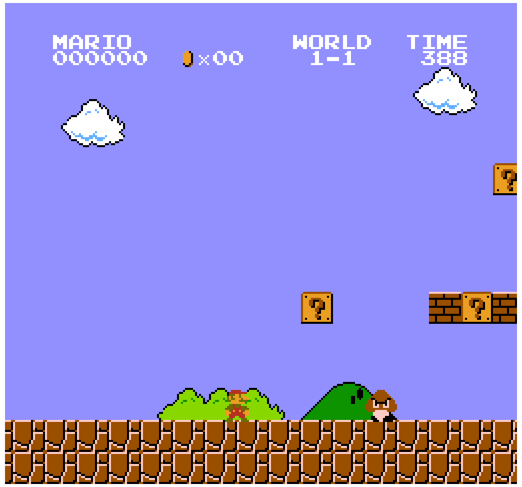
Observations/States Space

Observations/States are the **information our agent gets from the environment**. In the case of a video game, it can be a frame (a screenshot). In the case of the trading agent, it can be the value of a certain stock, etc.

There is a differentiation to make between *observation* and *state*, however:

- Observation o: is a **partial description of the state**. In a partially observed environment.

- In Super Mario Bros, we only see the part of the level close to the player, so we receive an observation. In Super Mario Bros, we are in a partially observed environment. We receive an observation since we only see a part of the level.



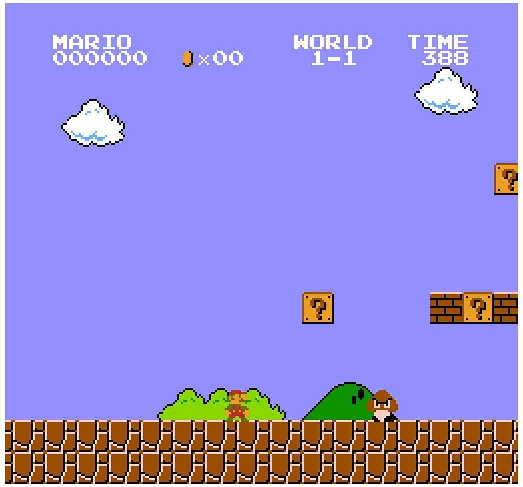
Action Space

The Action space is the set of **all possible actions in an environment.**

The actions can come from a *discrete* or *continuous space*:

- *Discrete space*: the number of possible actions is **finite**. Again, in Super Mario Bros, we have a finite set of actions since we have only 4 directions.

- *Continuous space*: the number of possible actions is infinite. A Self-Driving Satellite Mega-Constellation has infinite possible actions since it must perform whatever orbit modification is required.



Rewards and the discounting

The reward is fundamental in RL because it’s the only feedback for the agent. Thanks to it, our agent knows if the action taken was good or not.

The cumulative reward at each time step t can be written as:

$$\underbrace{R(\underbrace{\tau}_{\substack{\text{Trajectory (read Tau)}\\ \text{Sequence of states and actions}}})}_{\text{Return: cumulative reward}} = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \dots$$

Which is equivalent to:

$$R(\tau) = \sum_{k=0}^{\infty} r_{t+k+1}$$

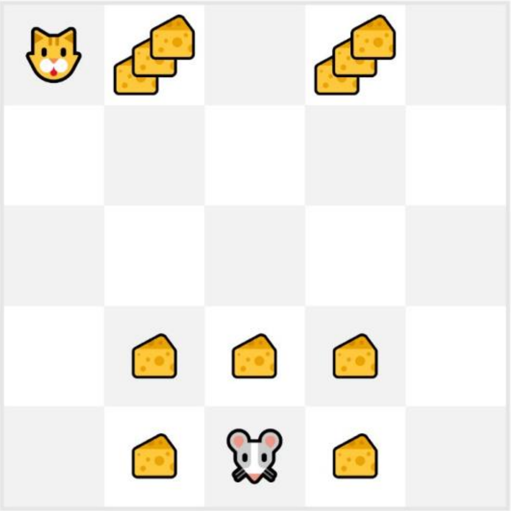
Rewards and the discounting

However, in reality, **we can't just add them like that.** The rewards that come sooner (at the beginning of the game) **are more likely to happen** since they are more predictable than the long-term future reward.

Let's say your agent is this tiny mouse that can move one tile each time step, and your opponent is the cat (that can move too). The mouse's goal is to eat the maximum amount of cheese before being eaten by the cat.

As we can see in the diagram, it's more probable to eat the cheese near us than the cheese close to the cat (the closer we are to the cat, the more dangerous it is).

Consequently, the reward near the cat, even if it is bigger (more cheese), will be more discounted since we're not really sure we'll be able to eat it.



Rewards and the discounting

To discount the rewards, we proceed like this:

1. We define a discount rate called gamma. **It must be between 0 and 1**. Most of the time between 0.95 and 0.99.
   - The larger the gamma, the smaller the discount. This means our agent **cares more about the long-term reward.**
   - On the other hand, the smaller the gamma, the bigger the discount. This means our **agent cares more about the short term reward (the nearest cheese).**

2. Then, each reward will be discounted by gamma to the exponent of the time step. As the time step increases, the cat gets closer to us, so the future reward is less and less likely to happen.

Rewards and the discounting

Our discounted expected cumulative reward is:

$$\underline{R(\boxed{\tau})} = r_{t+1} + \underline{\gamma} r_{t+2} + \underline{\gamma}^2 r_{t+3} + \underline{\gamma}^3 r_{t+4} + \dots$$

Return: cumulative reward

Gamma: discount rate

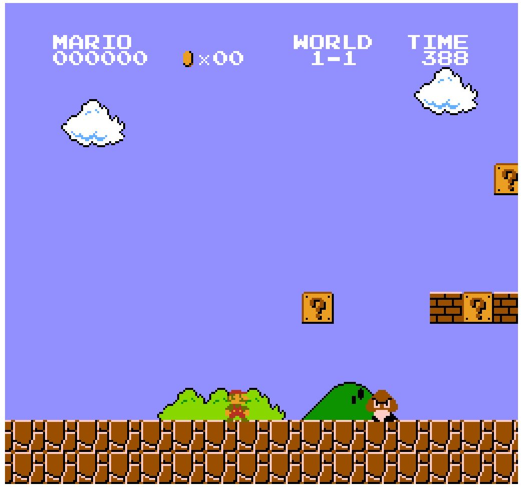
Trajectory (read Tau)
Sequence of states and actions

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Type of tasks

A task is an **instance** of a Reinforcement Learning problem. We can have two types of tasks: **episodic** and **continuing**.

Episodic task
In this case, we have a starting point and an ending point **(a terminal state). This creates an episode**: a list of States, Actions, Rewards, and new States. For instance, think about Super Mario Bros: an episode begins at the launch of a new Mario Level and ends **when you're killed or you reached the end of the level.**



Continuing tasks
These are tasks that continue forever (**no terminal state**). In this case, the agent must **learn how to choose the best actions and simultaneously interact with the environment.** For instance, an agent that does orbit maintenance. For this task, there is no starting point and terminal state. **The agent keeps running until we decide to stop it.**

The Exploration/Exploitation trade-off

Finally, before looking at the different methods to solve Reinforcement Learning problems, we must cover one more very important topic: *the exploration/exploitation trade-off.*

- *Exploration* is exploring the environment by trying random actions in order to **find more information about the environment.**

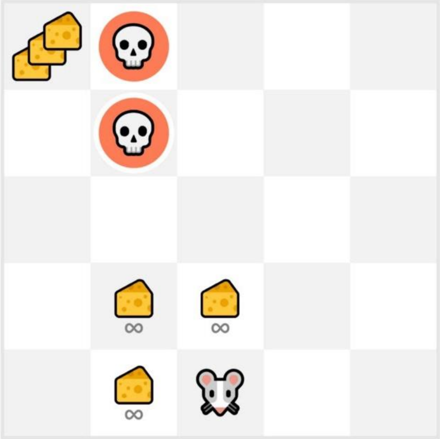
- *Exploitation* is **exploiting known information to maximize the reward.**

Remember, the goal of our RL agent is to maximize the expected cumulative reward. However, **we can fall into a common trap**.

The Exploration/Exploitation trade-off

Let’s take an example. In this game, our mouse can have an **infinite amount of small cheese** (+1 each). But at the top of the maze is a gigantic sum of cheese (+1000).

- If we only focus on exploitation, our agent will never reach the gigantic sum of cheese. Instead, it will only exploit **the nearest source of rewards,** even if this source is small (exploitation).
- But if our agent does a little bit of exploration, it can **discover the big reward** (the pile of big cheese).

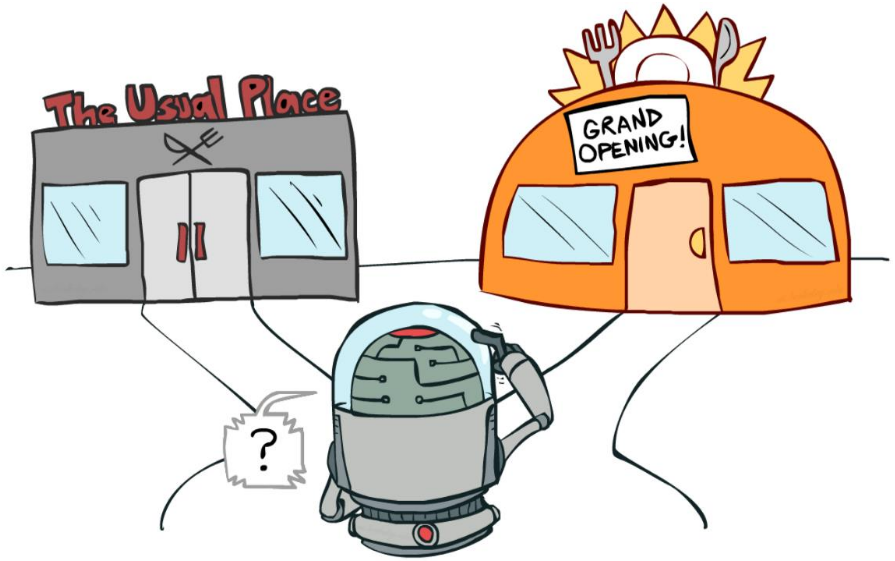


This is what we call the exploration/exploitation trade-off. We need to balance how much we **explore the environment** and how much we **exploit what we know about the environment.** Therefore, we must **define a rule that helps to handle this trade-off**.

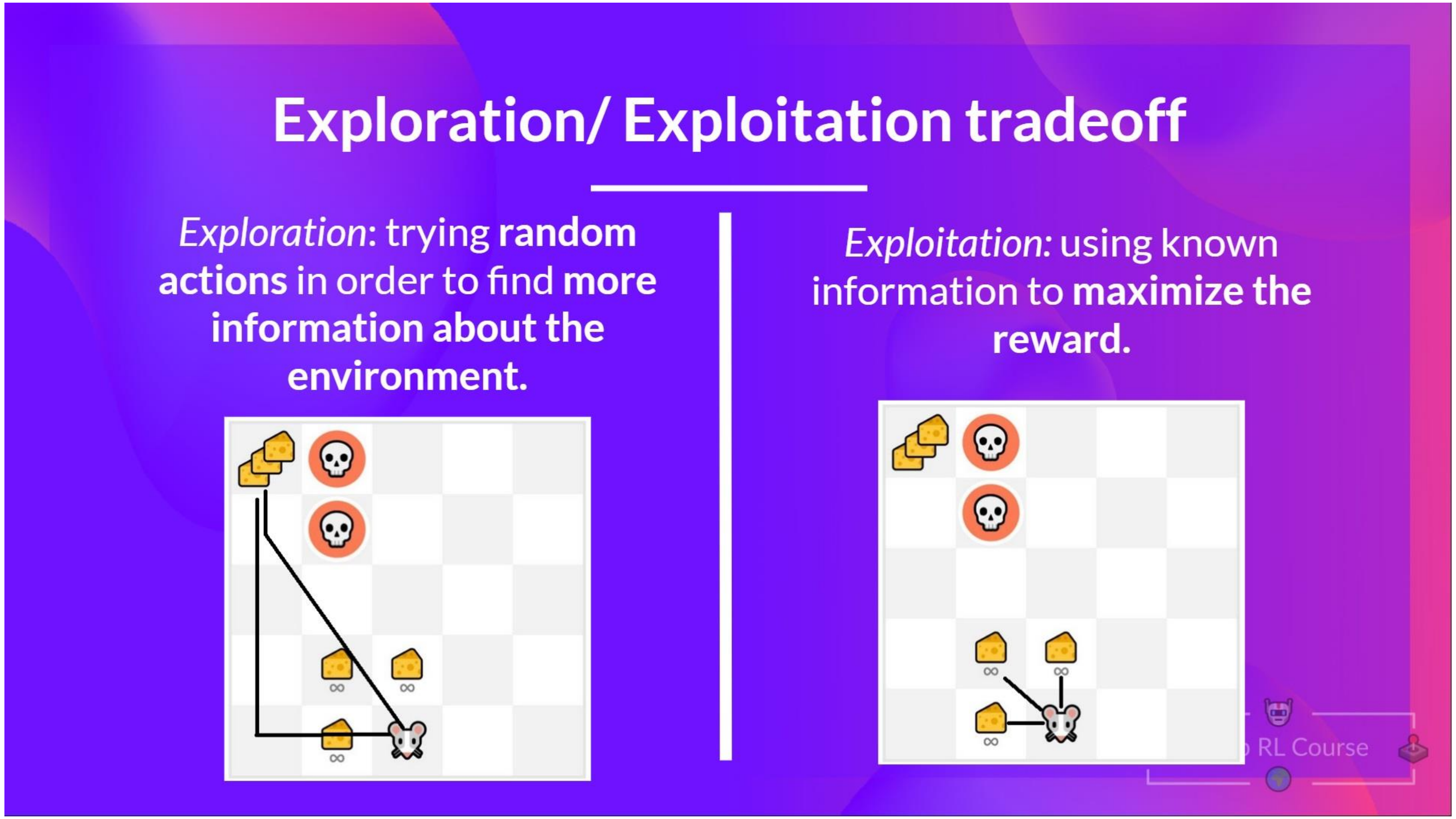
The Exploration/Exploitation trade-off

If it’s still confusing, think of a real problem: the choice of picking a restaurant:

Exploitation: You go to the same one that you know is good every day **and take the risk to miss another better restaurant.**
Exploration: Try restaurants you never went to before, with the risk of having a bad experience **but the probable opportunity of a fantastic experience**.



The Exploration/Exploitation trade-off



Exploration/ Exploitation tradeoff

Exploration: trying **random actions** in order to find **more information about the environment.**

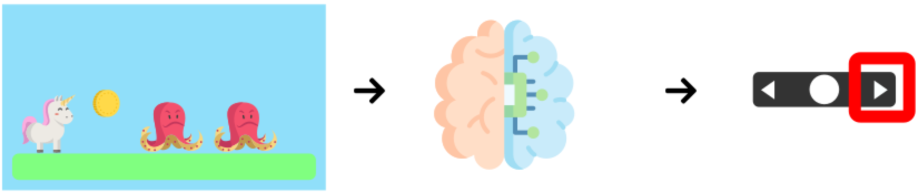
Exploitation: using known information to **maximize the reward.**

Two main approaches for solving RL problems

How do we build an RL agent that can **select the actions that maximize its expected cumulative reward?**

The Policy π: the agent's brain

The Policy **π** is the **brain of our Agent**, it's the function that tells us what **action to take given the state we are in.** So it **defines the agent's behavior** at a given time.



State → π(State) → Action

Two main approaches for solving RL problems

This Policy **is the function we want to learn**, our goal is to find the optimal policy π*, the policy that **maximizes expected return** when the agent acts according to it. We find this π* **through training.**

There are two approaches to train our agent to find this optimal policy π*:

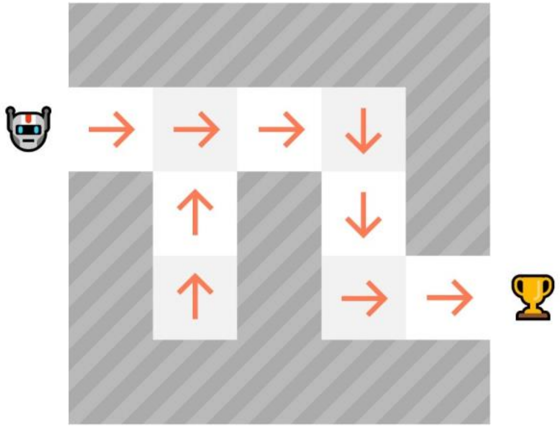
- **Directly,** by teaching the agent to learn which **action to take,** given the current state: **Policy-Based Methods.**
- Indirectly, **teach the agent to learn which state is more valuable** and then take the action that **leads to the more valuable states**: Value-Based Methods.

Policy-Based Methods

In Policy-Based methods, **we learn a policy function directly.**

This function will define a mapping from each state to the best corresponding action. Alternatively, it could define **a probability distribution over the set of possible actions at that state.**

As we can see here, the policy (deterministic) **directly indicates the action to take for each step.**

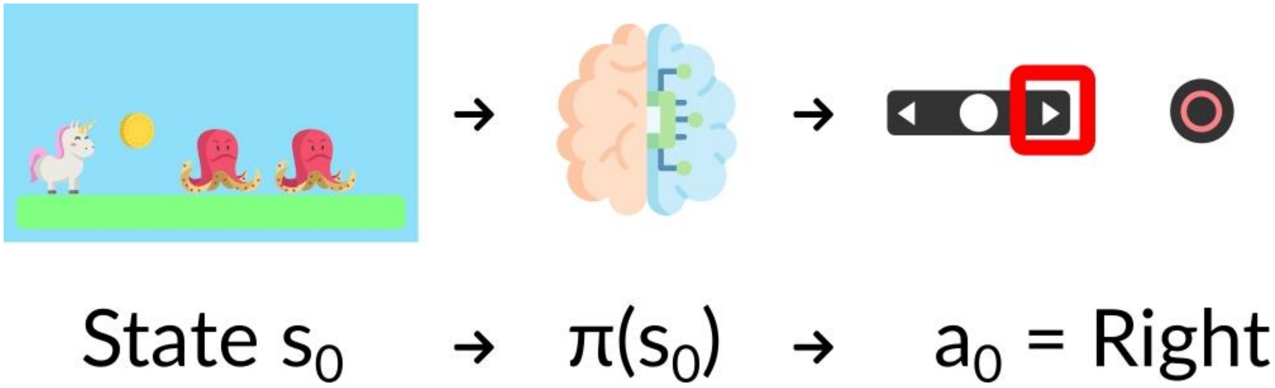


Policy-Based Methods

We have two types of policies:

- *Deterministic*: a policy in a given state **will always return the same action.**

$$a = \pi(s)$$



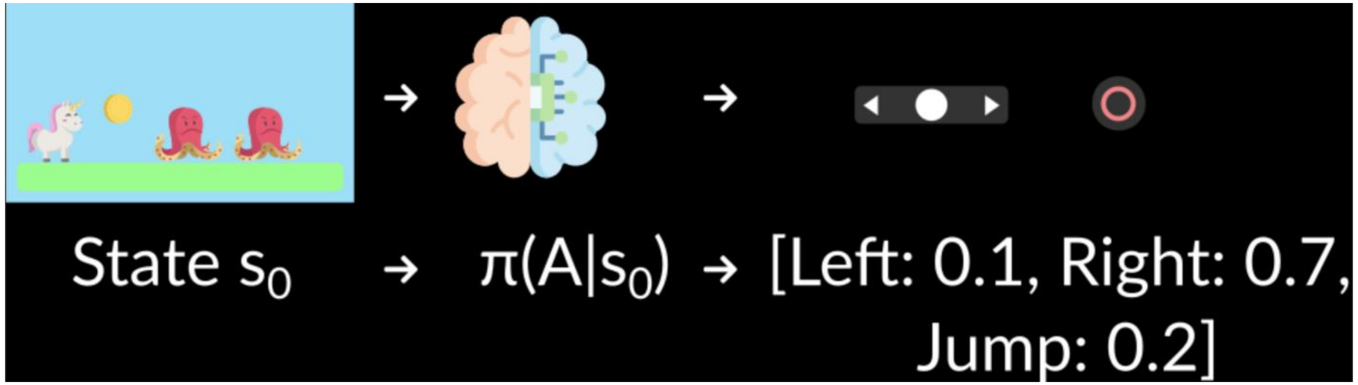
Policy-Based Methods

We have two types of policies:

- *Stochastic*: outputs **a probability distribution over actions.**

$$\pi(a|s) = \boxed{P[A|s]}$$

Probability Distribution over the set of actions given the state



State s_0 → $\pi(A|s_0)$ → [Left: 0.1, Right: 0.7, Jump: 0.2]

Value-based methods

In value-based methods, instead of learning a policy function, we **learn a value function** that maps a state to the expected value **of being at that state.**

The value of a state is the **expected discounted return** the agent can get if it **starts in that state, and then acts according to our policy.**

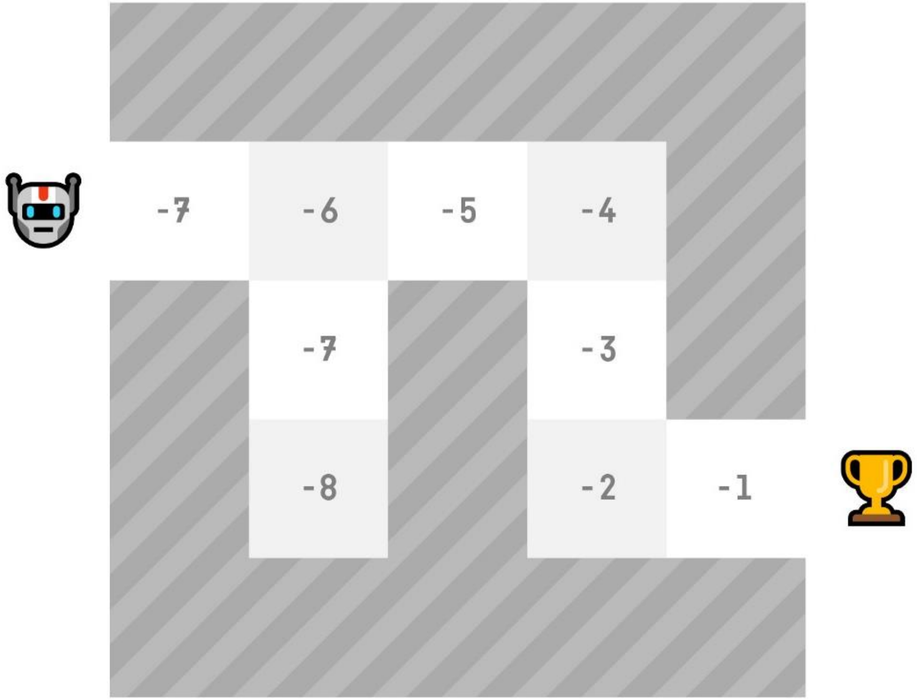
“Act according to our policy” just means that our policy is **“going to the state with the highest value”.**

$$\underbrace{v_\pi(s)}_{\text{Value function}} = \underbrace{\mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots\right.}_{\text{Expected discounted return}} \mid \underbrace{\left. S_t = s\right]}_{\text{Starting at state s}}$$

Value-based methods

Here we see that our value function **defined values for each possible state.**

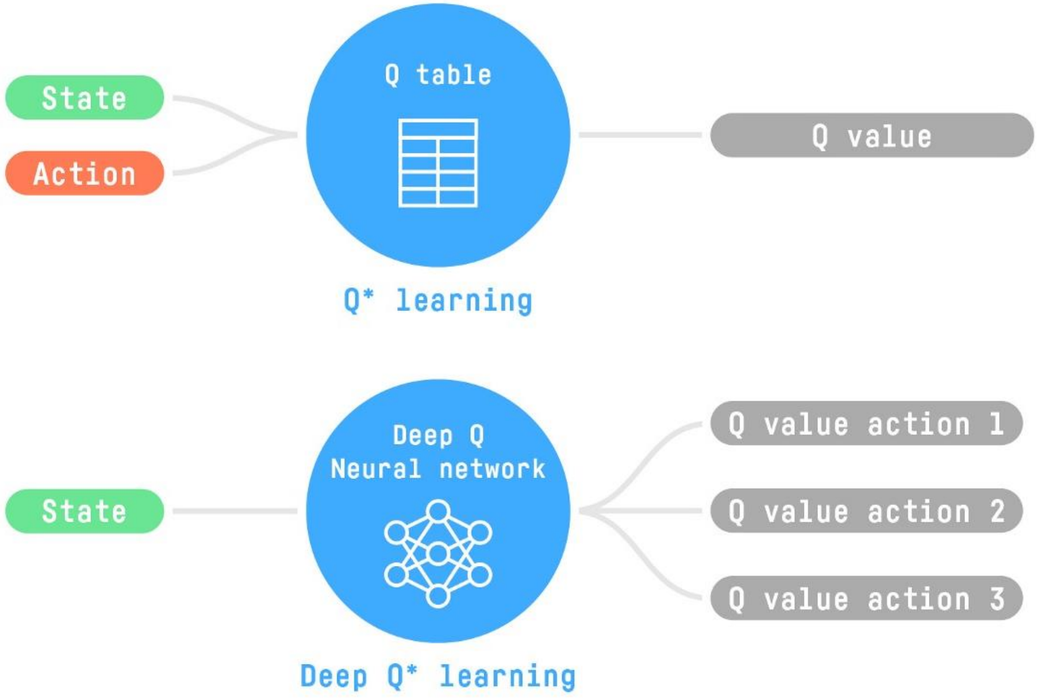
Thanks to our value function, at each step our policy will select the state with the biggest value defined by the value function: -7, then -6, then -5 (and so on) to attain the goal.



The “Deep” in Reinforcement Learning

- Deep Reinforcement Learning introduces deep neural networks to solve Reinforcement Learning problems — hence the name “deep”. For instance, we’ll learn about two value-based algorithms: Q-learning (classic Reinforcement Learning) and then Deep Q-learning.

- In the first approach, we use a traditional algorithm to create a Q table that helps us find what action to take for each state. In the second approach, we will use a Neural Network (to approximate the Q value).



Summary

RL is a computational approach of learning from actions. We build an agent that learns from the environment **by interacting with it through trial and error** and receiving rewards (negative or positive) as feedback.

The goal of any RL agent is to maximize its expected cumulative reward (also called expected return) because RL is based on the **reward hypothesis**, which is that **all goals can be described as the maximization of the expected cumulative reward.**

The RL process is a loop that outputs a sequence of **state, action, reward and next state.** To calculate the expected cumulative reward (expected return), we discount the rewards: the rewards that come sooner (at the beginning of the game) **are more probable to happen since they are more predictable than the long term future reward.**

Summary

To solve an RL problem, you want to **find an optimal policy**. The policy is the “brain” of your agent, which will tell us **what action to take given a state.** The optimal policy is the one which **gives you the actions that maximize the expected return.**

There are two ways to find your optimal policy:

- By training your policy directly: **policy-based methods.**
- By training a value function that tells us the expected return the agent will get at each state and use this function to define our policy: **value-based methods.**

Finally, we speak about Deep RL because we introduce **deep neural networks to estimate the action to take (policy-based) or to estimate the value of a state (value-based)** hence the name “deep”.