Deep Reinforcement Learning

Lecture 1: Preliminaries

Dr. Matthia Sabatelli
m.sabatelli@rug.nl

university of groningen

The Deep Reinforcement Learning Team

- Lecturer & Coordinator: Matthia Sabatelli
- Lecturer: Nicole Orzan
- Special Guests

The Deep Reinforcement Learning Team

- Teaching Assistants:
  - Lennard Froma : l.c.froma@student.rug.nl
  - Jeroen Van Gelder: j.h.van.gelder.1@student.rug.nl

Learning Goals

By the end of the course you ...

- ... will be able to summarize the main algorithmic components that define a DRL agent.
- ... will be able to categorize DRL algorithms into three families of popular DRL techniques.
- ... will know how to assess the performance of different DRL algorithms on a large variety of tasks.
- ... will be able to adapt existing DRL algorithms to novel DRL testbeds.
- ... will have gained enough knowledge to program a DRL algorithm from scratch.

Outline

- Lecture 1: Preliminaries
- Lecture 2: Deep-Q Networks
- Lecture 3: Policy Gradients
- Lecture 4: Deep Model-Based Reinforcement Learning
- Lecture 5: Meta-Learning
- Lecture 6: Multi-Agent Reinforcement Learning
- Lecture 7: Applications & Future Challenges

Lecture Rooms

- Theoretical Lectures: Tuesday 9:00-11:00 (NB 5115.0013)
    - One **exception** on the 26th of May when we will be in the U-building
- Lab Sessions: Thursday 9:00-11:00 (LB 5173.0076)

Assessment

The **final grade** will be given by:

- Group Assignment (divided into 2 parts): 40%
    - You can work alone or in pairs
- Presentation: 10%
- Exam: 50%

To get a **passing grade** you need:

- at least a 5 for the final exam
- an averaged grade > 5.5

Course Material

All slides can be found on the Brightspace course page.

- There is no official textbook for this course but if you are interested in diving deeper into some of the material I recommend:
  - François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4), 219-354.
  - Plaat, Aske. "Deep Reinforcement Learning." arXiv preprint arXiv:2201.02135 (2022).

2 main

If you happen to have any question regarding either the theoretical lectures or assignments you can send an email to the TAs, but be sure to include Nicole and I in CC (**both of us!**):

- Matthia Sabatelli: m.sabatelli@rug.nl
- Nicole Orzan: n.orzan@rug.nl

Course Material

To help you **prepare** for the exam:

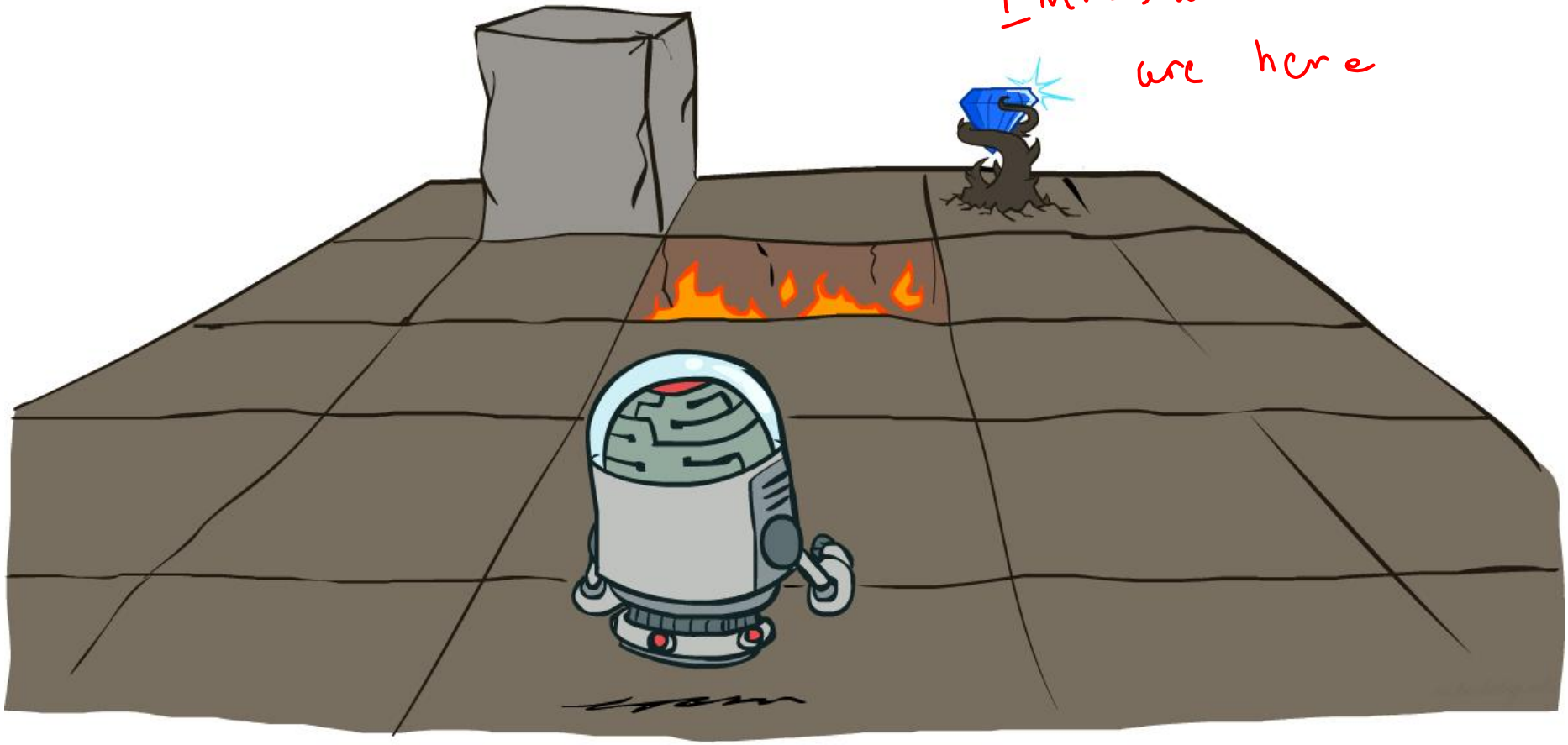
- At the end of every lecture, for each covered topic, I will upload on Brightspace a list of open questions that you can answer based on the material that is presented in the slides.
- The final exam will consist of a random sample of such open questions + some multiple choice questions.
    - You can expect at least one open question per topic!

Today's Agenda

We will cover all the Reinforcement Learning background that is needed before diving into DRL

- Markov Decision Processes
- Optimal Value Functions and Policies
- Learning Value Functions
- Reinforcement Learning & Regression

Markov Decision Processes (MDPs)



MDPs are the mathematical framework that is used for solving sequential decision making problems.

An MDP is a tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathfrak{R}, \gamma)$ where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the action space
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function
- $\mathfrak{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function returning r_t
- γ is the discount factor

The MDP is also known as the RL environment the agent interacts with.

Such interaction is governed by the agent's policy π which is a probability distribution over $a \in \mathcal{A}(s)$ for each $s \in \mathcal{S}$:

$$\pi(a|s) = \Pr\left(a_t = a | s_t = s\right), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}.$$

Policies can be of two different types:

- They can be **deterministic** if $\forall s : \pi(a|s) = 1$ for exactly one $a \in \mathcal{A}(s)$ and $\pi(b|s) = 0$ for all $b \in \mathcal{A}(s) \setminus a$.
- A policy is also defined as **stationary** if it does not change over time.

Different DRL algorithms learn different types of policies!

The elements of the MDP together with the agent's policy allow us to properly model the dynamics of an agent interacting with its environment:

- At each time-step t the environment provides the agent with a certain state s_t
- The agent then performs action a_t
- The environment returns the reward signal r_t
- The agent will enter into a new state s_{t+1}

This interaction can technically be infinite, but in practice it is finite as sooner or later the agent will encounter an absorbing state, i.e. a state which only transitions to itself with $r_t = 0$.

The interaction of the agent with the environment is defined as an episode, which consists of one or several trajectories τ that come in the form of the following sequence:

$$(s_t, a_t, r_t, s_{t+1}), t = 0, ..., T-1$$

where T is a random variable representing the length of the episode.

A key property of the environment is that it is Markovian meaning that the probability of visiting s_{t+1} is

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = p(s_{t+1}|s_t, a_t)$$

and that the probability of observing r_t is given by

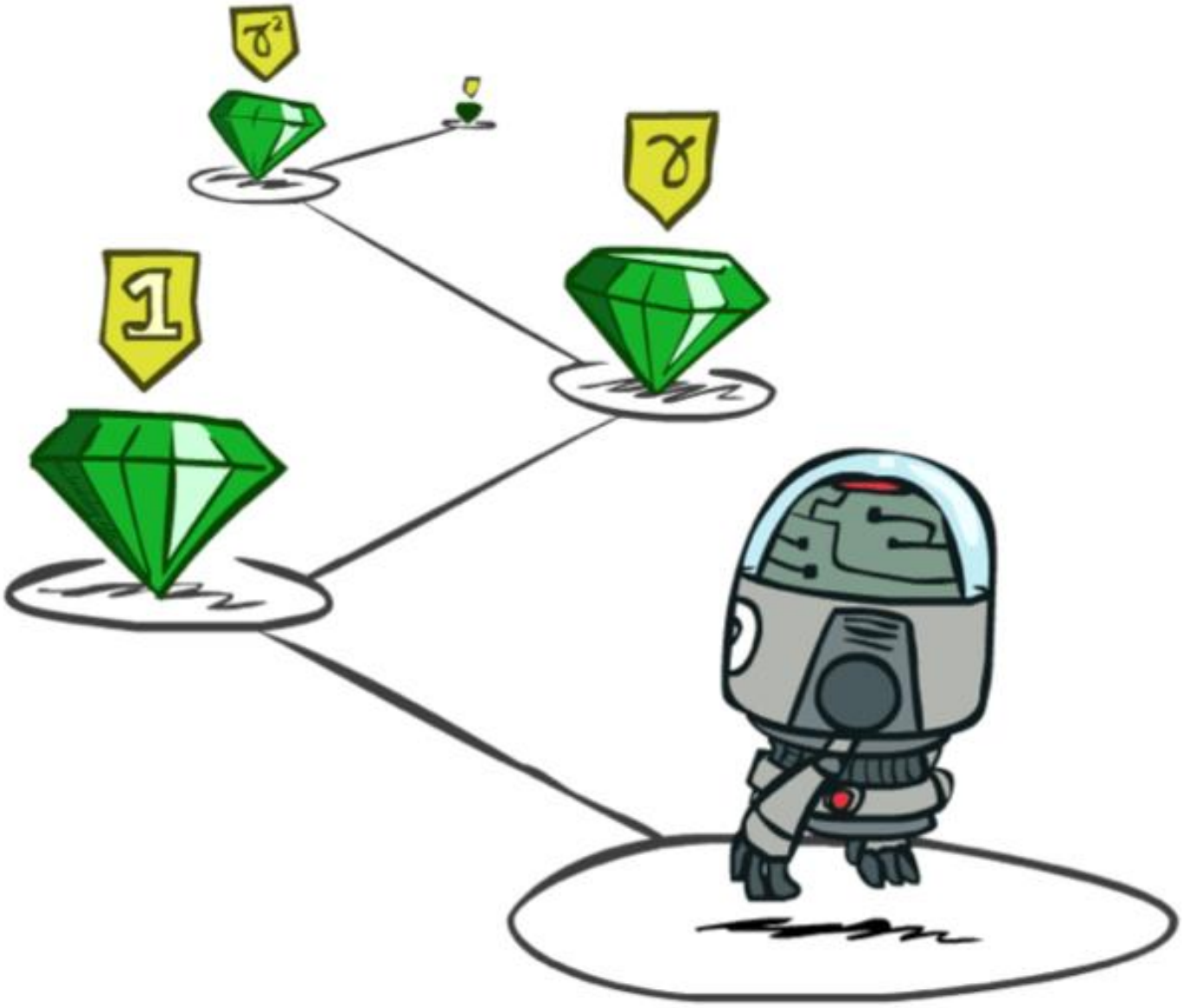
$$p(r_t|s_t, a_t, ..., s_1, a_1) = p(r_t|s_t, a_t).$$

The agent's goal is that of maximizing the total amount of reward it receives while interacting with the environment. In the simplest case, we can define this as:

$$G_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_T.$$

However, we typically consider the discounted case:

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \end{aligned}$$


γ^2
γ
1

Thanks to γ we can deal with infinite horizons and assign a different weight to each reward signal.

The notion of discounted return allows us to introduce the concept of value.

We can define the value of a state s as well as the value of a policy π.

The value of a state s is a function $V^\pi : \mathcal{S} \to \mathbb{R}$ that measures the expected return that an agent will receive when starting in state s and following π thereafter.

following π in state s → expected discounted reward

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, \pi\right].$$

We can also condition this value function on the action a that the agent takes and obtain $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, defined as:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi\right].$$

(s, a) → how good to take action in state

The difference between the quality function and the value function gives us the advantage function $A^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of policy π:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s).$$

- The advantage function essentially tells us how much better (or worse) a given action is than taking the average action in a particular state. A positive advantage value indicates that the action is better than average, while a negative advantage value indicates that the action is worse than average.

- The advantage function will play a crucial role when we will see **Policy Gradient** methods in Lecture 3!

Optimal Policies and Value Functions

In RL we are not interested in any value function, but rather in a value function that maximizes each state-value or state-action value.

This allows us to find an optimal policy $\pi^\star$, which is a policy that realizes the optimal expected return:

$$V^\star(s) = \max_\pi \ V^\pi(s), \ \text{for all } s \in \mathcal{S}$$

and the optimal Q value function:

$$Q^\star(s,a) = \max_\pi \ Q^\pi(s,a) \ \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}.$$

If the optimal Q function is learned, it becomes a straightforward task to derive an optimal policy since one only needs to select the action which has the highest value in each state as defined by:

$$\pi^\star(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q^\star(s,a) \text{ for all } s \in \mathcal{S}.$$

It is also worth noting that the Q function and the V function satisfy the following equality:

$$V^\star(s) = \max_{a \in \mathcal{A}}\ Q^\star(s,a) \text{ for all } s \in \mathcal{S}.$$

As we will later see throughout this lecture this equality is particularly important for the development of many RL algorithms.

Typically RL deals with 2 different type of problems which are both related to the concept of policy π:

- Evaluation
- Control

The problem of Policy Evaluation consists in assessing the perfomance of a given policy π. This corresponds to computing either $V^{\pi}(s)$; $Q^{\pi}(s, a)$ or even the expected return $J(\pi)$.

The problem of Control means finding the optimal policy $\pi^{\star}$, which corresponds to solving the MDP.

Throughout this course we will focus on the, arguably much more interesting, **Control** problem.

How can we do this?

We start from the assumption that some components of the RL environment are **not known**:

Given an MDP:

$$\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathfrak{R}, \gamma)$$

we consider the scenario where the transition function $\mathcal{P}$ and the reward function $\mathfrak{R}$ are unknown to the agent. → we only get small signals

This lack of information needs to be addressed, which results in 2 popular branches of RL:

- Model-Free Reinforcement Learning → learn either stuff and compensate for lack of information (P, R)
- Model-Based Reinforcement Learning

  → learn MDP components, later I can use other algorithms

Model-Free Reinforcement Learning algorithms do not care about the transition $\mathcal{P}$ and reward $\mathfrak{R}$ functions. Instead they focus on either learning a value function or a policy.

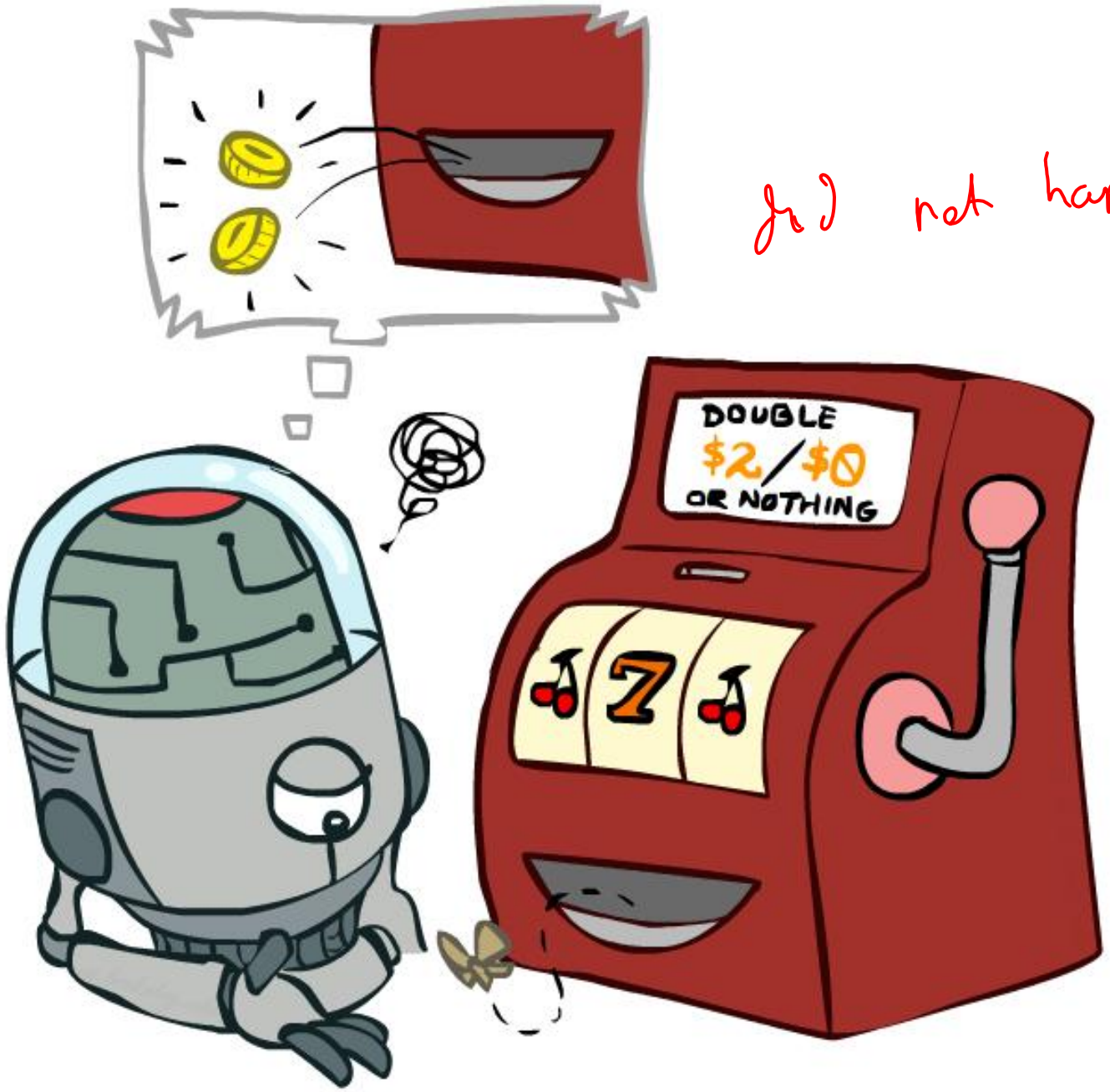
- **Value-Based** algorithms aim at learning the optimal state-action value function $Q^\star(s, a)$, from which an optimal behavior can be extracted by acting greedily as seen earlier:

$$\pi^\star(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; Q^\star(s, a) \text{ for all } s \in \mathcal{S}.$$

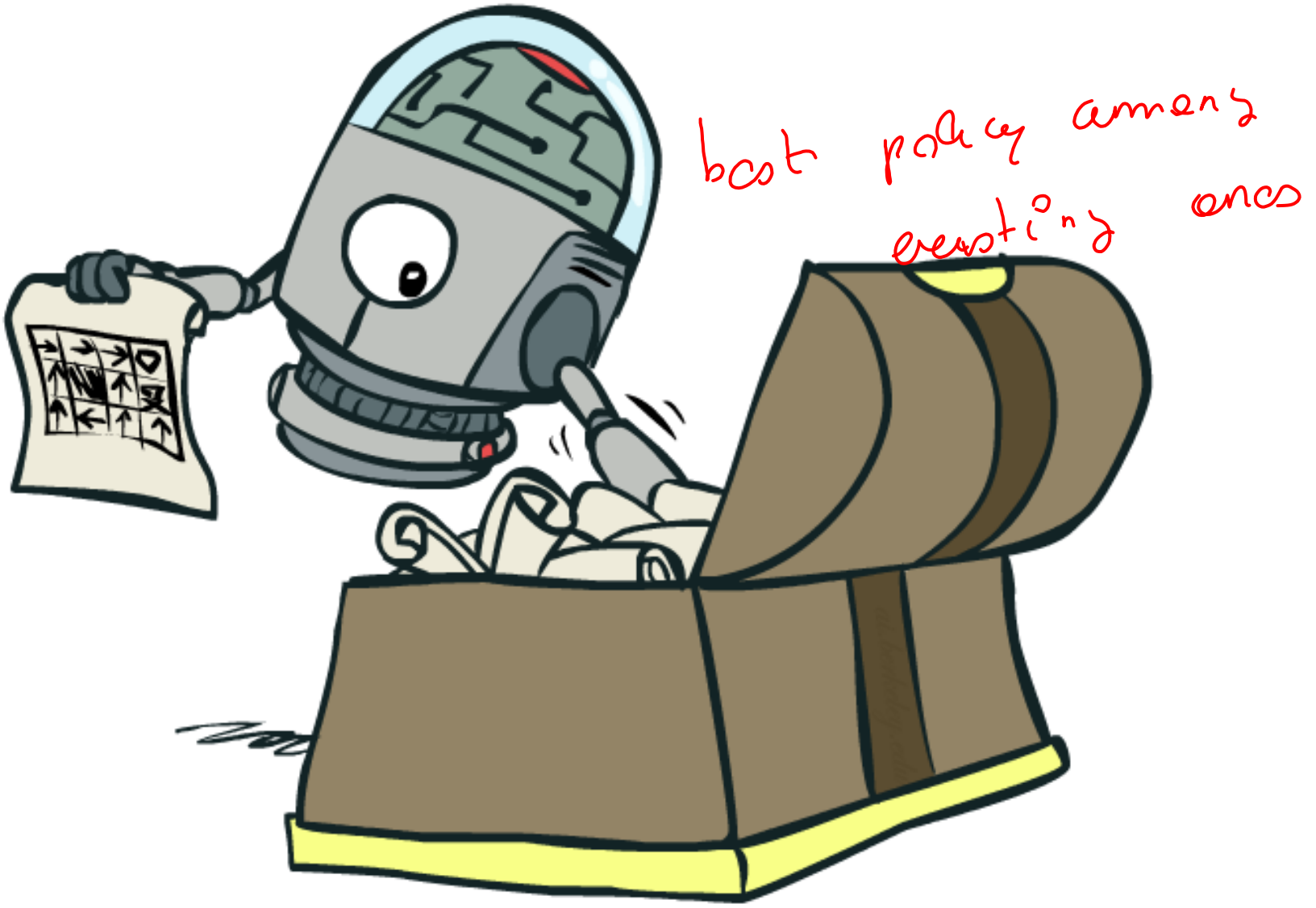
- **Policy-Search** algorithms directly look into the space of possible policies without modelling any value function. Here it is common to consider parametrized stochastic policies in the form of:

$$\pi(a|s; \theta) \text{ where } \theta \in \Theta$$

It is also possible to combine both methods and have **Actor-Critic** algorithms.



do not happen

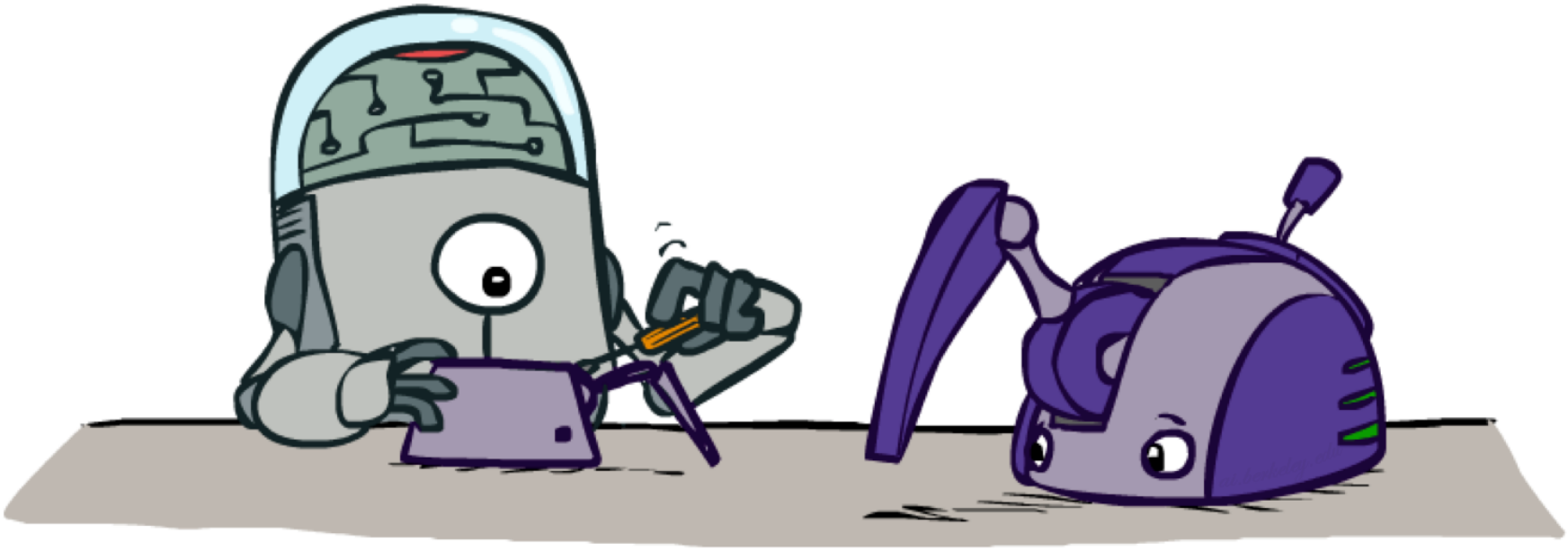

best policy among existing ones

Model-Based Reinforcement Learning algorithms focus on learning an approximation of the transition and reward functions:

- $\mathcal{P} \approx (\hat{\mathcal{P}}; \theta)$
- $\mathcal{R} \approx (\hat{\mathcal{R}}; \theta)$.

This typically corresponds to solving a Supervised Learning problem.

Once these approximations are learned it is possible to use Dynamic Programming and Planning algorithms to let the agent interact with the environment optimally.



Learning Value Functions

From now on we will consider the task of learning an optimal value function in the tabular case, meaning we will learn the exact value for either $V^{\pi}(s)$ or $Q^{\pi}(s, a)$.

We can do this with two different families of techniques:

- Monte Carlo (MC) Methods
- Temporal Difference (TD) Methods

Let's start with the first one ...

We consider the evaluation problem of finding the value function $V^{\pi}(s)$ for some state $s \in \mathcal{S}$.

The goal is to compute the actual sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

that an agent obtains once an episode finishes and use it for updating the value of a state with the following update rule:

$$V(s_t) := V(s_t) + \alpha \big[G_t - V(s_t) \big]$$

where $\alpha \in [0, 1]$ is the learning rate controlling how much we want to change the value estimate of a state based on G_t.

The properties of MC-Learning methods are:

- They only work for episodic tasks
- Result in low-variance estimates

Break

The biggest **limitation** of MC-Learning is that this family of techniques cannot learn from incomplete episodes.

We must wait until the episode ends to get a sample return G_t and start learning.

Temporal-Difference (TD) Learning allows us to overcome this issue by starting to learn after one single trajectory:

$$\tau = \langle s_t, a_t, r_t, s_{t+1} \rangle.$$

learn by timestep, not episode

Our goal is to update the value of a state with respect to the value of its successor state only.

For the evaluation problem we do this as follows:

$$V(s_t) := V(s_t) + \alpha \big[r_t + \gamma V(s_{t+1}) - V(s_t) \big].$$

We can see that the update of a state s_t now only depends on the discounted reward r_t and the value of the next state s_{t+1}.

Updating the value of a state with respect to the value of its successor state only is a technique that comes with the name of bootstrapping, and comes from the fact that the state-value function $V^{\pi}(s)$ satisfies the following recursive relationship:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| s_t = s, \pi\Big] \\
&= \mathbb{E}\Big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \Big| s_t = s, \pi\Big] \\
&= \mathbb{E}\Big[r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \dots) \Big| s_t = s, \pi\Big] \\
&= \mathbb{E}\Big[r_t + \gamma V^{\pi}(s_{t+1}) \Big| s_t = s, \pi\Big]
\end{aligned}
$$

A property which can be re-written as the Bellman equation:

$$
\sum_{a} \pi(a|s) \sum_{s_{t+1}} p(s_{t+1}|s,a) \Big[\Re(s_t, a, s_{t+1}) + \gamma V^{\pi}(s_{t+1})\Big].
$$

The same derivation can be applied to the state-action value function, which can therefore be expressed as follows:

$$Q^{\pi}(s,a) = \sum_{s_{t+1}} p(s_{t+1}|s,a)\big(\Re(s_t,a,s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1})Q^{\pi}(s_{t+1},a_{t+1})\big).$$

As a result one can use bootstrapping also for learning $Q^{\pi}(s,a)$.

We can do this in 2 different ways:

- On-Policy Learning
- Off-Policy Learning

An algorithm is said to learn on-policy if the policy that is being learned is also the one that is used while interacting with the environment. Examples of such algorithms are:

- SARSA
- QV(λ)-Learning

In an off-policy algorithm these two policies differ. For instance in the policy evaluation scenario, the agent might interact with the environment using some behavioral policy π_b with the aim of evaluating a target policy π. Examples of such algorithms are:

- Q-Learning
- Double Q-Learning

If $\pi_b = \pi$ we are in the on-policy scenario, whereas we are in the off-policy scenario if $\pi_b \neq \pi$.

The latter are much harder to train when combined with neural networks!

The goal of the SARSA algorithm is to learn an estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ via the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\Big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big],$$

bootstrap from

The algorithm learns on-policy as the state-action pair $Q(s_t, a_t)$ gets updated with respect to $r_t + \gamma Q(s_{t+1}, a_{t+1})$, therefore taking into account the next state and action that comes from the agent's policy.

Fun fact about the name of the algorithm:

- s_t
- a_t
- r_t
- s_{t+1}
- a_{t+1}

Also QV (λ) learns on-policy by keeping track of an estimate of the state-value function $V : \mathcal{S} \to \mathbb{R}$ alongside the usual estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

As two separate value functions are learned, the algorithm requires two separate update rules:

$$V(s) := V(s) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]e_t(s),$$

for the state-value function, and

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)\big]$$

for the state-action value function.

The algorithm learns on-policy as both value functions get updated with respect to $V(s_{t+1})$ which by definition is conditioned on π.

The arguably most popular off-policy learning algorithm is Q-Learning

Its goal is to update each visited state-action pair with the following rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \Big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \Big].$$

The key component of Q-Learning's update rule is the $\mathbf{max}$ operator, which defines it as an off-policy learning algorithm.

Since there are as many Q values as there are actions available to the agent, one must choose which Q value to use as a reference when updating the value of the state-action pair that the agent is currently visiting (π_b).

The $\mathbf{max}$ operator simply chooses the state-action pair with the largest Q value, which defines π as the greedy policy.

Note that the agent might not interact greedily with the environment, therefore $\pi \neq \pi_b$!

The $\max$ operator of Q-Learning has the appealing property of making Q-Learning converge to $Q^\star(s, a)$ with probability 1 as long as all state-action pairs are visited infinitely often.

However, it has the drawback of suffering from overestimation bias as $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ corresponds to the expected maximum value of a state, instead of its maximum expected value. → too optimistic / unrealistic value high
⟺ underestimation if using min

To overcome this issue one can keep track of two different state-action value functions, Q_1 and Q_2, which get alternatively used for selecting which action to perform.

This algorithm comes with the name of Double Q-Learning.

In Double Q-Learning one of the two Q functions determines the action that maximizes the state-action value of the next state, whereas the remaining value function is used for evaluating this estimate.

This can be achieved with the following update rule:

$$Q_1(s_t, a_t) := Q_1(s_t, a_t) + \alpha \left[r_t + \gamma \, Q_2(s_{t+1}, a^\star) - Q_1(s_t, a_t) \right],$$

where:

$$a^\star = \mathbf{argmax}_{a \in \mathcal{A}} Q_1(s_{t+1}, a).$$

Note that at each time step, only one of the two Q functions gets updated.

While training progresses, the choice of which Q function to update is determined randomly.

Exploration/Exploitation Dilemma

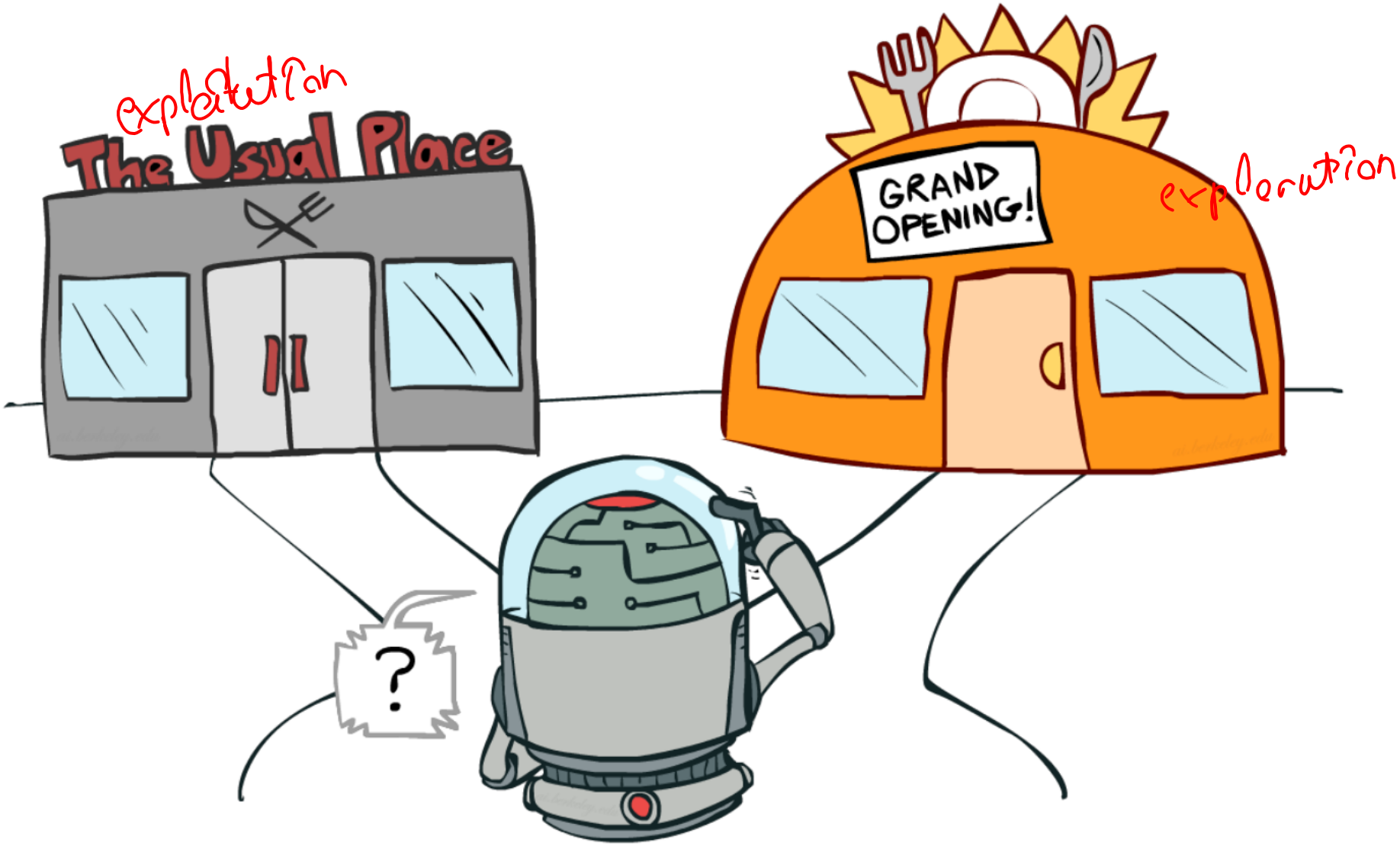
The difference between on-polcy VS off-policy learning algorithms does not have to be confused with the concept of exploration policy.

We have seen that the agent interacts with the environment based on policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Throughout this interaction the agent constantly needs to choose between 2 options:

- Rely on the learned $Q^{\pi}(s, a)$ function and therefore hope that it is possible to derive $\pi^{\star}(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \ Q^{\star}(s, a) \ \text{for all} \ s \in \mathcal{S}$ (exploitation).
- Keep interacting with the environment with the goal of improving the $Q^{\pi}(s, a)$ function (exploration). → random action

value function → "knowledge"


exploitation
The Usual Place
GRAND OPENING!
exploration
?

This is not an easy task as one must decide when to explore and when to exploit.

The arguably most popular selection policy is ϵ-**greedy** which defines the action that the agent takes as:

$$a_t = \begin{cases} \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s_t, a) & \text{with probability } 1 - \epsilon \\ a \sim \mathcal{U}(\mathcal{A}) & \text{with probability } \epsilon \end{cases}$$

ϵ is a hyperparameter that changes while training progresses.

During early training iterations, its value is close to 1, while it approaches 0 by the end of training.

This allows the agent to take actions that are representative of a large set of policies when the learned Q function does not yet correspond to $Q^\star(s, a)$, while it will favor greedy actions at the end of training.

Another common policy is the Boltzmann Policy which is defined as:

$$\pi(a|s) = \frac{e^{Q(s,a)/\eta}}{\sum_{a' \in \mathcal{A}} e^{Q(s,a')/\eta}}$$

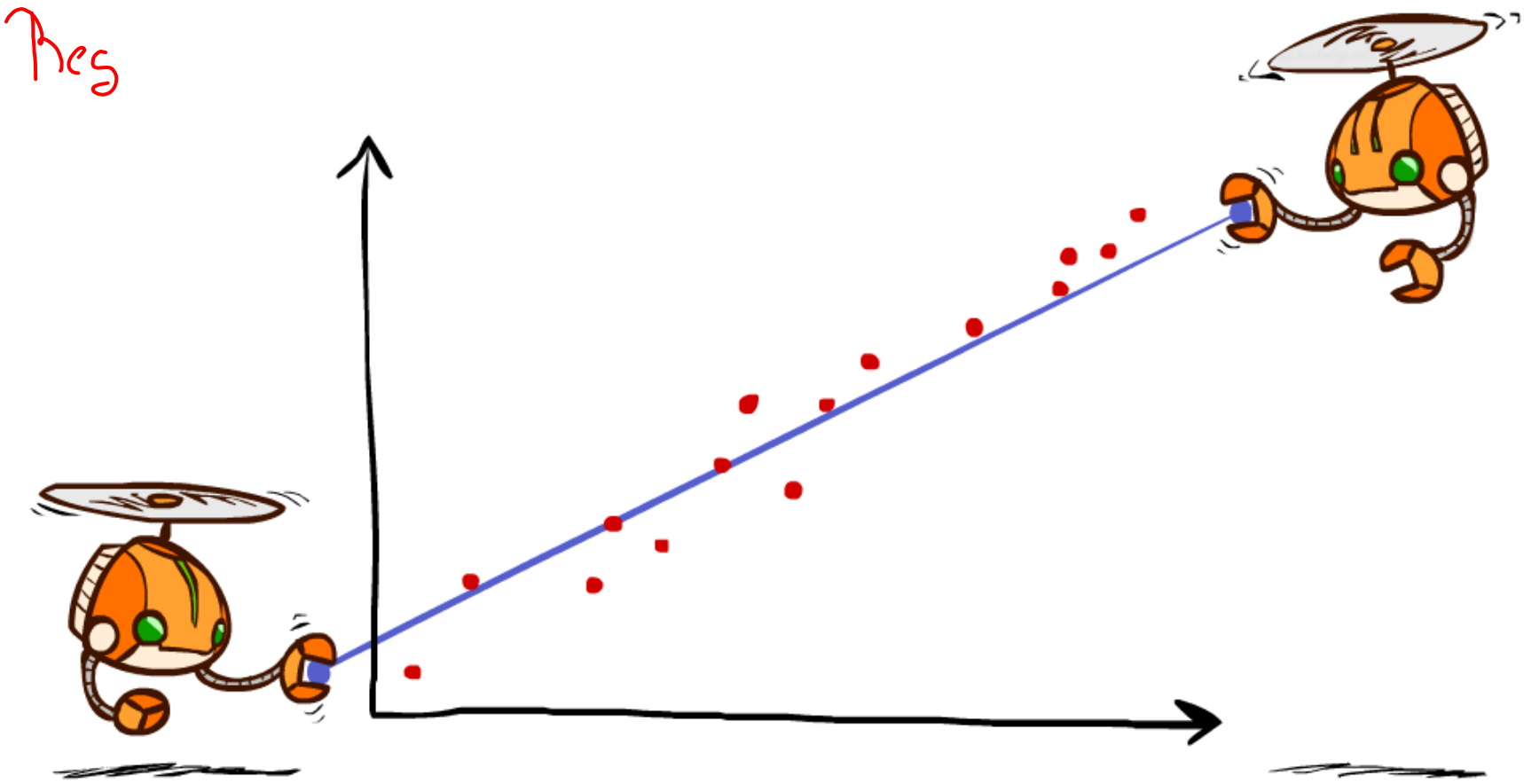
softmax over actions

where $\eta > 0$ is called the temperature parameter, which controls the spread of the probaility distribution.

The idea is to assign more probability to actions with larger values.

In the end, the more $\eta \to 0$, the more greedy the agent will be. Similarly to the aforementioned ϵ parameter also η can get annealed over time.

Reinforcement Learning & Regression



Updating a value function can be - in part - seen as dealing with a regression problem.

Given an

- Input space $\mathcal{X}$;
- An output space $\mathcal{Y}$;
- A joint probability distribution $P(X, Y)$;
- A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$;

we know from supervised learning that in a regression problem $\mathcal{Y} \in \mathbb{R}$ and that our goal is that of estimating a real number.

↳ In DRL we don't have it

Typical loss functions ℓ that are used in such a scenario are the squared error loss:

$$\ell(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2$$

or the absolute error loss

$$\ell(f(\mathbf{x}), y) = |y - f(\mathbf{x})|,$$

where f is the learning algorithm under scrutiny.

If we take a closer look at Q-Learning's update rule

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\big],$$

the following observations can be made:

- Our learning algorithm makes a prediction for a certain state-action pair $Q(s_t, a_t)$
- We take the difference between this prediction and what is called the Temporal-Difference Target: $y_t = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$
- α controls how much $Q(s_t, a_t)$ should get updated with respect to y_t.

This is not so different from supervised learning regression where $f(x)$ is $Q(s_t, a_t)$ and $\mathcal{Y}$ the space of all Temporal Difference targets.

There are however 2 major differences between the standard supervised learning scenario and the reinforcement learning one, which have to be taken into account:

- We are dealing with a problem that is **not-stationary**. The more the agent interacts with the environment, the more it will observe states that are new and that are associated to novel reward signals. As a result the temporal difference targets y_t will constantly change.
- The regression target that is used for learning is **itself** computed by the learning algorithm.

In supervised learning the target values used for regression are static, as the labels y of the output space $\mathcal{Y}$ do not change throughout learning and do not depend from the predictions $f(\mathbf{x})$ made by the learner.

References

- Sabatelli, Matthia. "Contributions to Deep Transfer Learning: from Supervised to Reinforcement Learning." (2022).
- Tirinzoni, Andrea. "Exploiting structure for transfer in reinforcement learning." (2021).
- Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

See you next week