



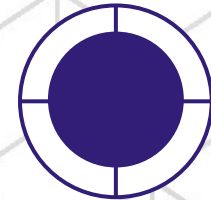
Reinforcement Learning for Decision-Making Problems



Imen Jendoubi

AI Research Engineer at InstaDeep
i.jendoubi@instadeep.com

17/02/2025



Who this course is for

The main target audience are should have some knowledge in Machine Learning, but interested to get a practical understanding of the Reinforcement Learning domain. The attendee should be familiar with Python and the basics of deep learning and machine learning.



What this course covers

Introduction to RL and main formal models.

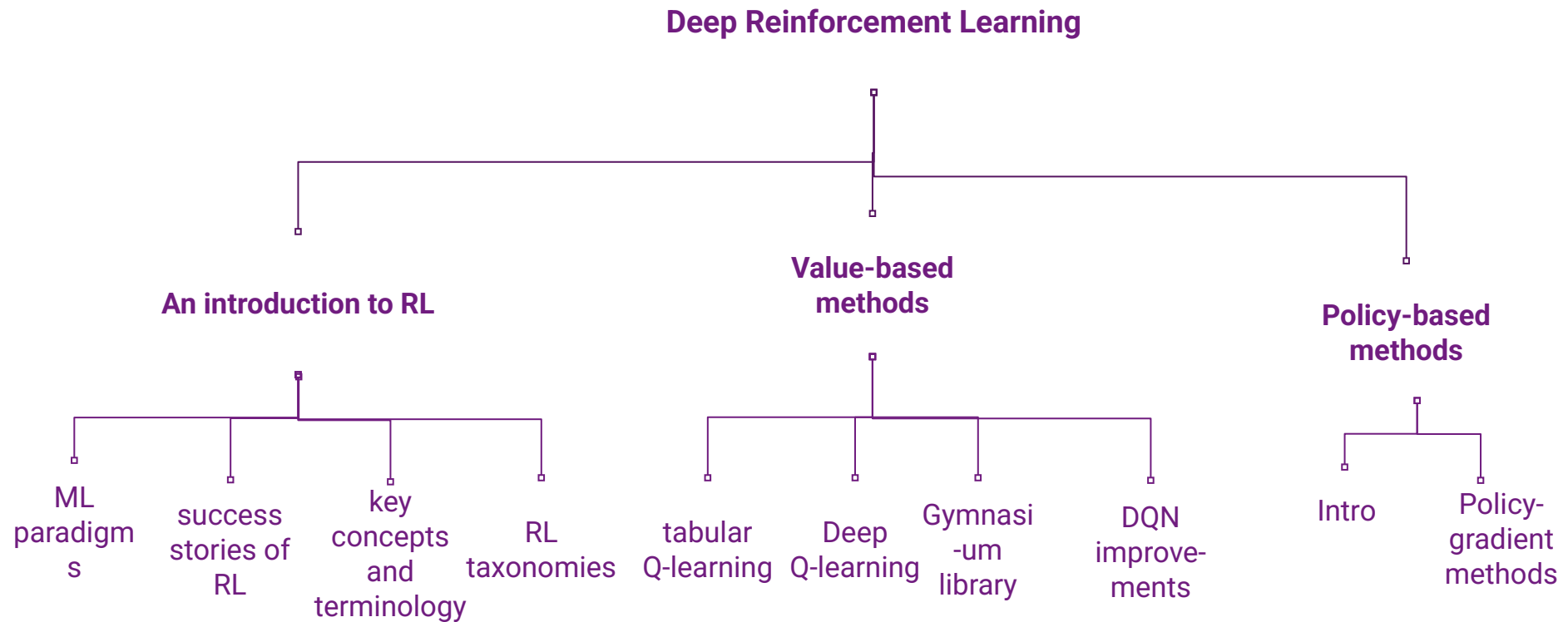
Aspect of practical RL using open-source library gym.

Value based methods

Policy based methods



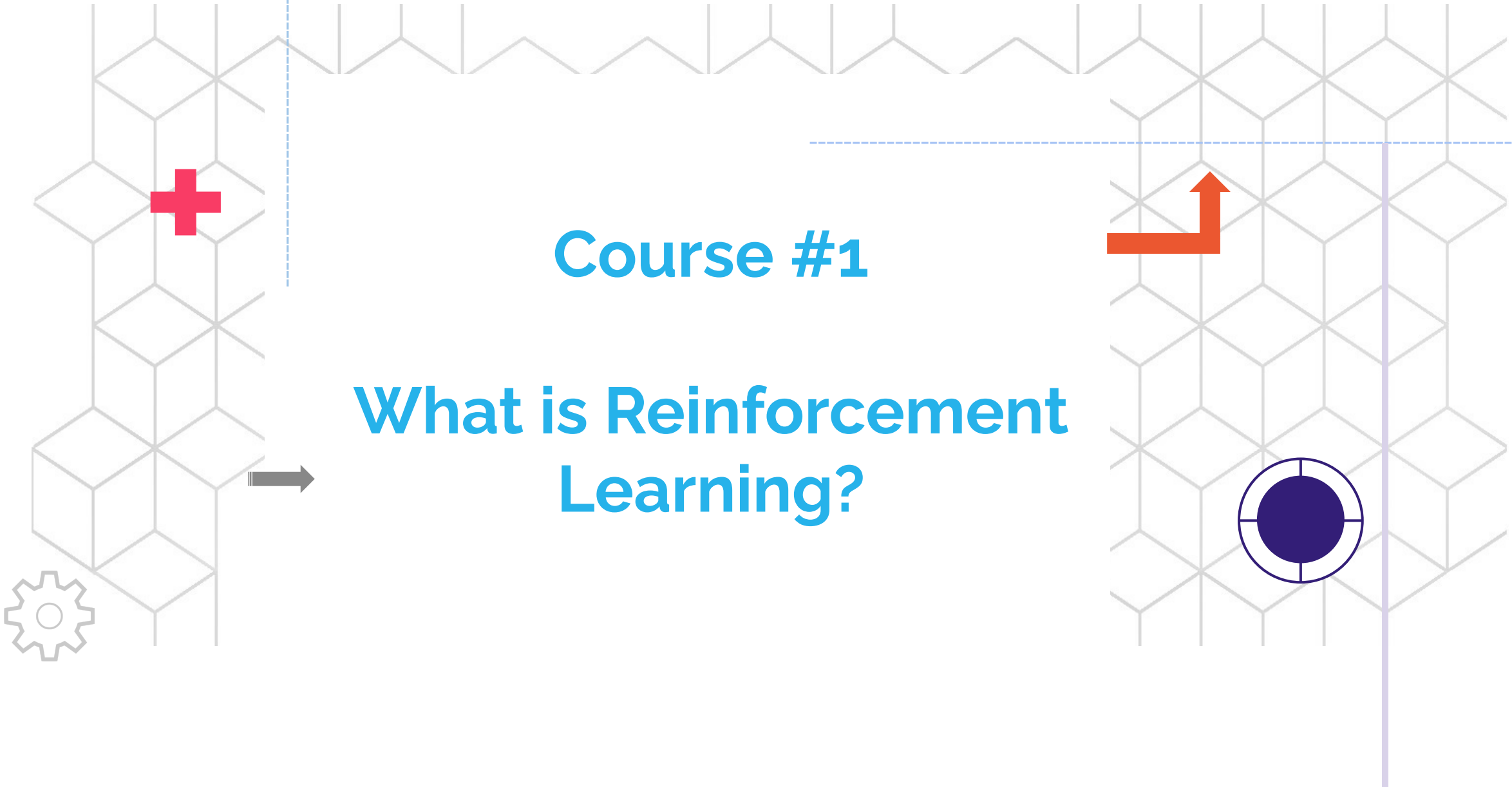
Course Plan:





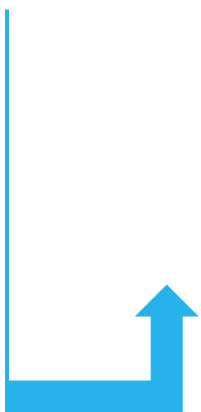
Course #1

What is Reinforcement Learning?

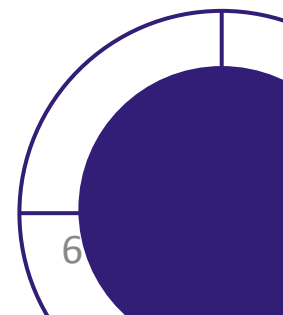




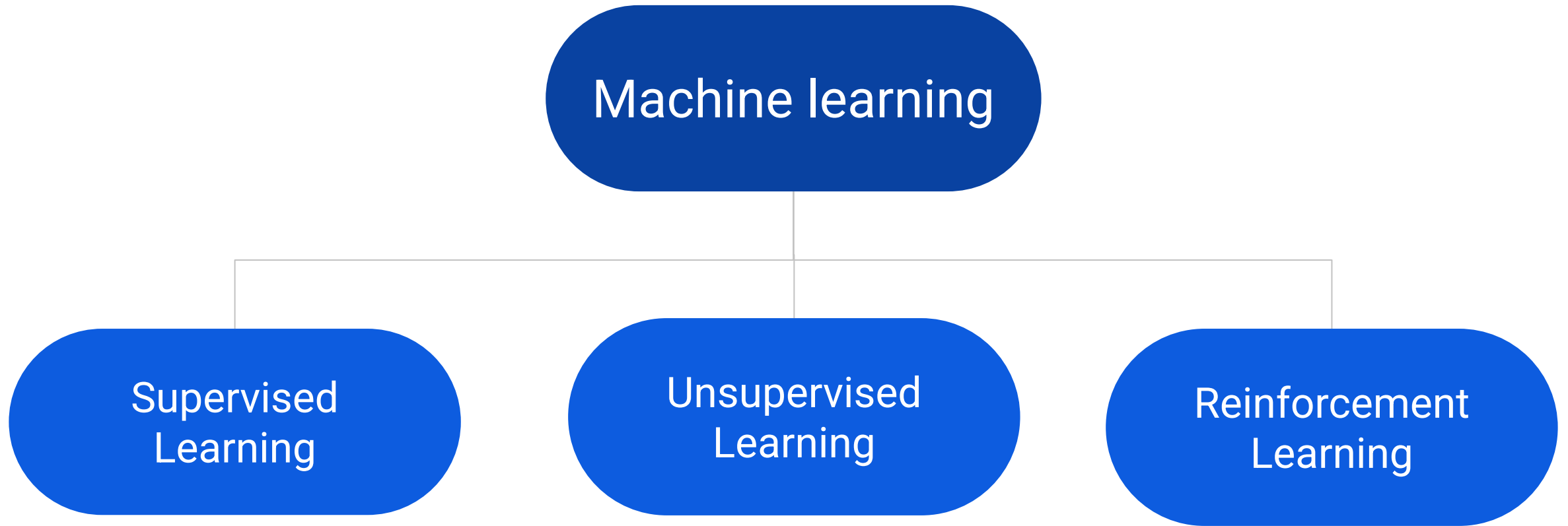
01



Machine learning paradigms



Machine learning paradigms

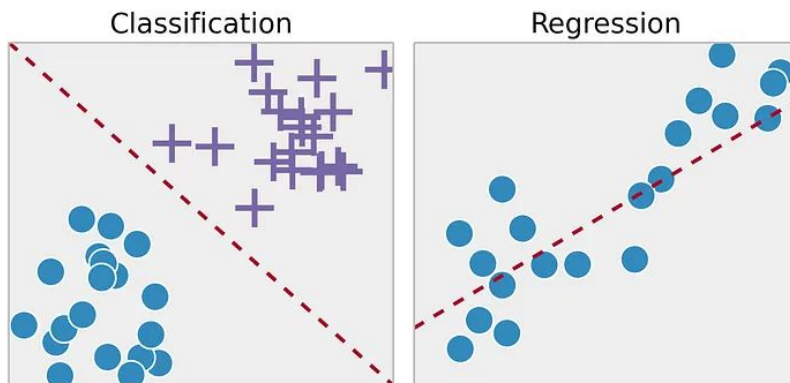


Supervised learning

Data: (X,Y)

- X is i.i.d sampled data,
- Y is label

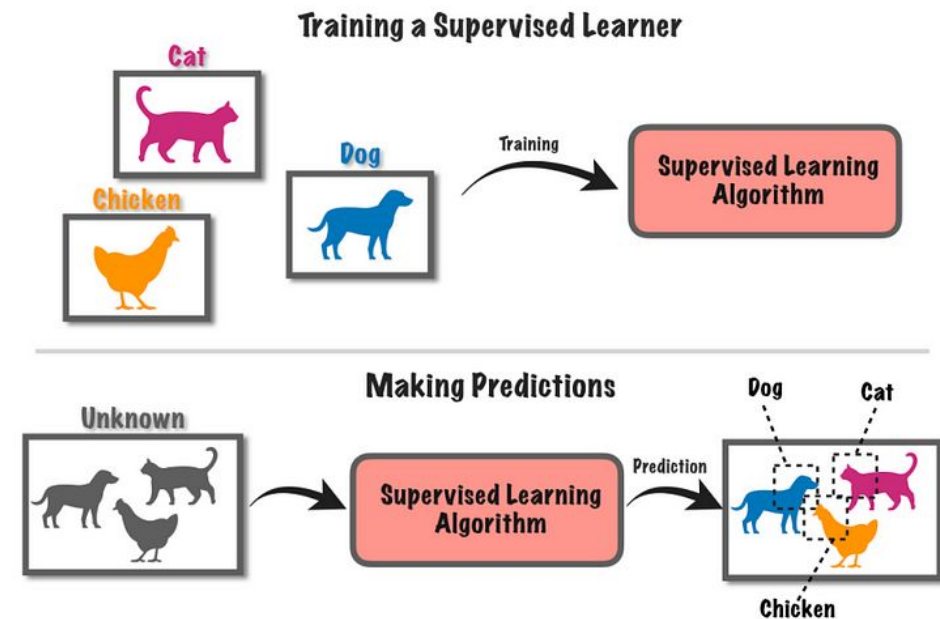
Goal: build a function that maps some input into some output $X \rightarrow Y$, when given a set of example pairs.



Source: Supervised vs. Unsupervised learning, by Devin Soni, 2018, Towards Data Science

Applications

- Text classification
- Image classification and object location
- Regression problems
- Sentiment analysis



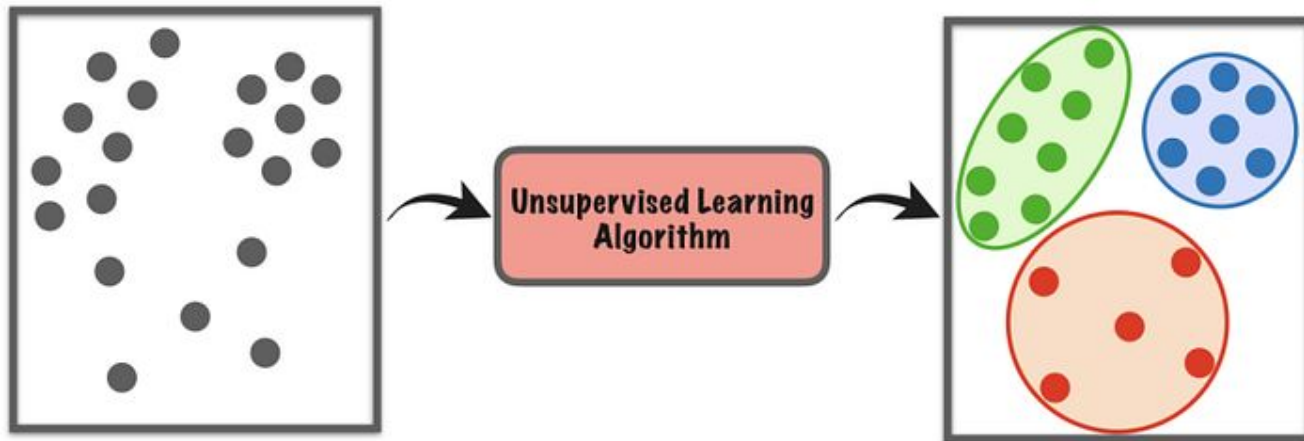
Source: Supervised vs. Unsupervised learning in 3 Minutes, by Alan Jeffares, 2018, Towards Data Science

Unsupervised learning

Data: X

Just data, no labels!

Goal: Learn some underlying hidden structure of the data



Source: Introduction to Unsupervised Learning (Kmeans clustering), by Sachin, 2021, Medium

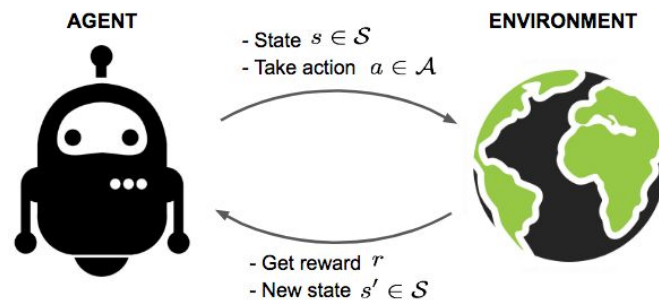
Applications

- Resolves Clustering, dimensionality reduction, etc.

Reinforcement learning

An autonomous agent learns how to solve a task, through trial and error, from interaction with an environment

Goal: Problems involving an agent interacting with an environment which provides numeric reward signals that reflect how good its actions were.



Source: Reinforcement Learning real-world examples, by Ajitesh Kumar, 2022, VitaFlux

Reinforcement Learning



Learn through trial and error from interaction with an environment.

- ✓ States & actions
- ✓ No data set
- ✓ 'Find actions that maximize reward'
- ✓ Decision making
- ✓ Learning to play a game, movie recommendation system

Comparison

Takeaways:

- Supervised Learning is about learning to predict from examples of correct predictions
- Unsupervised learning is about learning hidden patterns within unlabeled data
- Reinforcement learning is about agents learning by themselves how to take good actions in their environments.

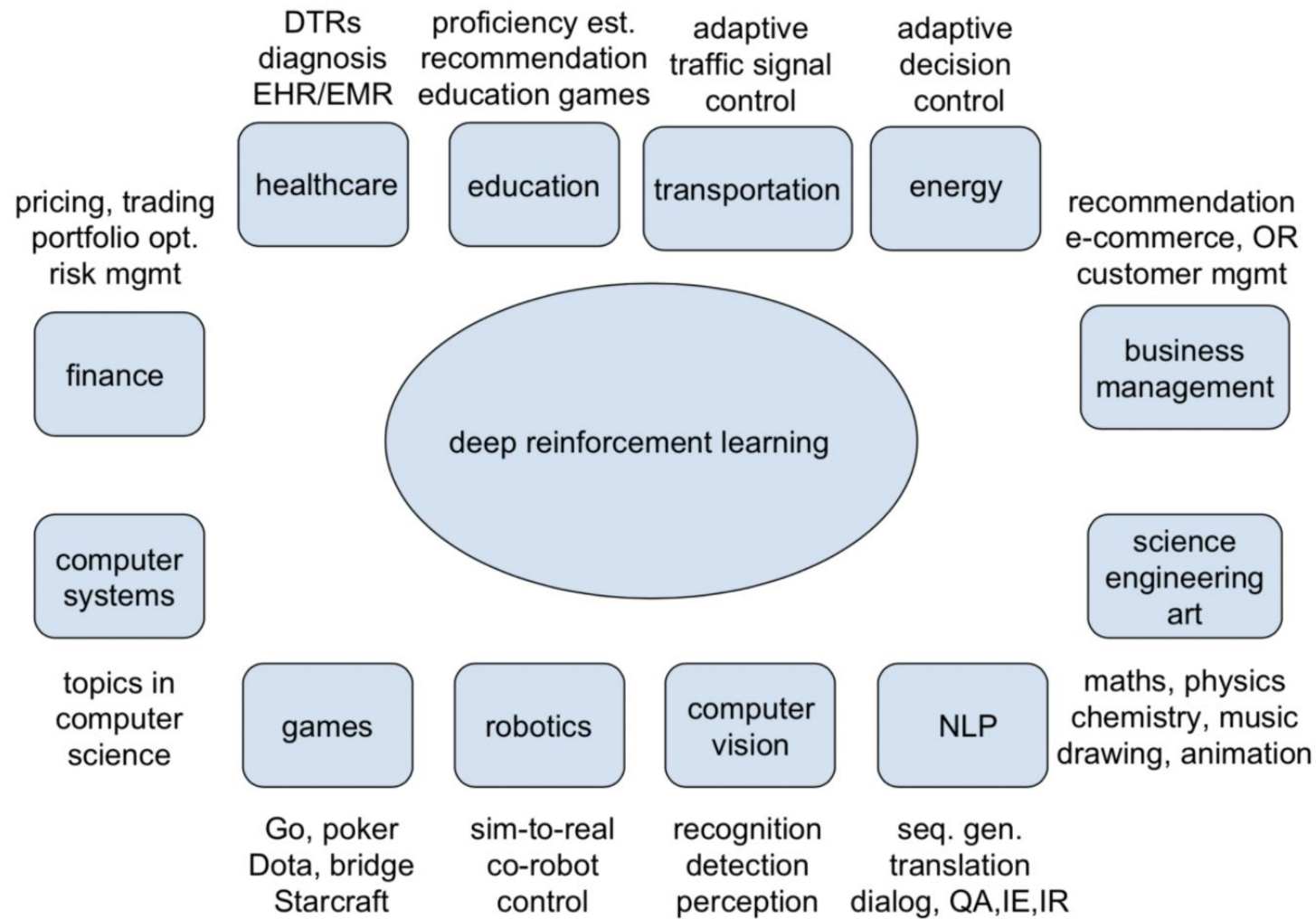
02

Success stories of

RL



Success stories of RL: The Tip of the Iceberg!



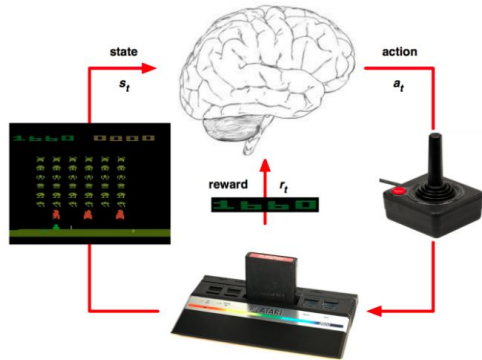
Yuxi Li, Deep Reinforcement Learning, arXiv, 2018

Success stories of RL: Games

#1 ATARI

Source: D. Silver, "Lecture Notes in advanced topics in Machine Learning", 2020.

From https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf



#2 AlphaGo

Source: AlphaGo Movie, 2017.

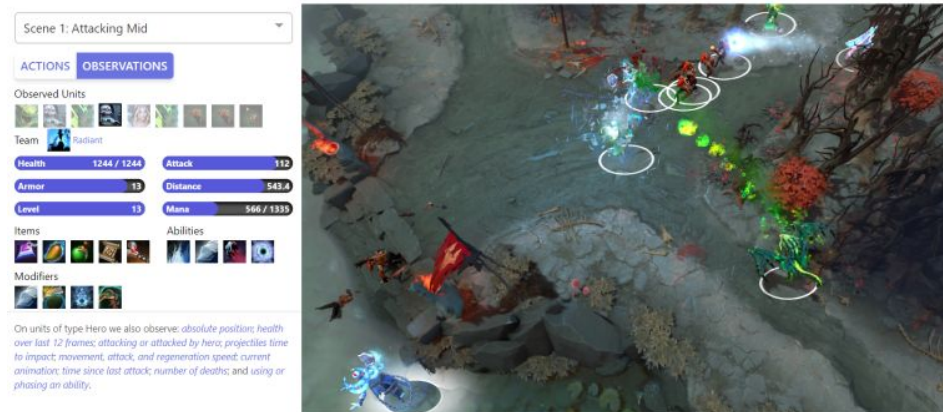
From https://www.youtube.com/watch?v=8tg1C8spV_g&t=1s



#3 AlphaZero

Source: Google DeepMind, 2017.

From <https://www.youtube.com/watch?v=WXHFqTvfFSw>



#4 Dota2

Source: Nested - Artificial Intelligence, 2020.

From <https://nested.ai/2020/10/26/dota-2-with-large-scale-deep-reinforcement-learning/>

Success stories of RL: Biology

Example: Protein Design

Science News

from research organizations

Reinforcement learning: From board games to protein design

Protein design software developers have adapted an artificial intelligence strategy proven adept at chess and Go

[Reinforcement learning: From board games to protein design](#)

Success stories of RL: Biology

Why is protein folding important?

I think that we shall be able to get a more thorough understanding of the nature of disease in general by investigating the molecules that make up the human body, including the abnormal molecules, and that this understanding will permit...the problem of disease to be attacked in a more straightforward manner such that new methods of therapy will be developed.

Linus Pauling, 1960

Scientists have long been interested in determining the structures of proteins because a protein's form is thought to dictate its function. Once a protein's shape is understood, its role within the cell can be guessed at, and scientists can develop drugs that work with the protein's unique shape.

AlphaFold: Using AI for scientific discovery

Source: Deepmind, 2020

From <https://www.deepmind.com/blog/alphafold-using-ai-for-scientific-discovery-2020>



Protein folding explained

Source: Google DeepMind, 2020.

From <https://www.youtube.com/watch?v=KpedmJdrTpY>

Success stories of RL: Autonomous driving



Self-Driving Cars

[Source](#): 9 Reinforcement Learning Real-Life Applications, by Pragati Baheti, 2022, V7 Labs

Success stories of RL: Robotics

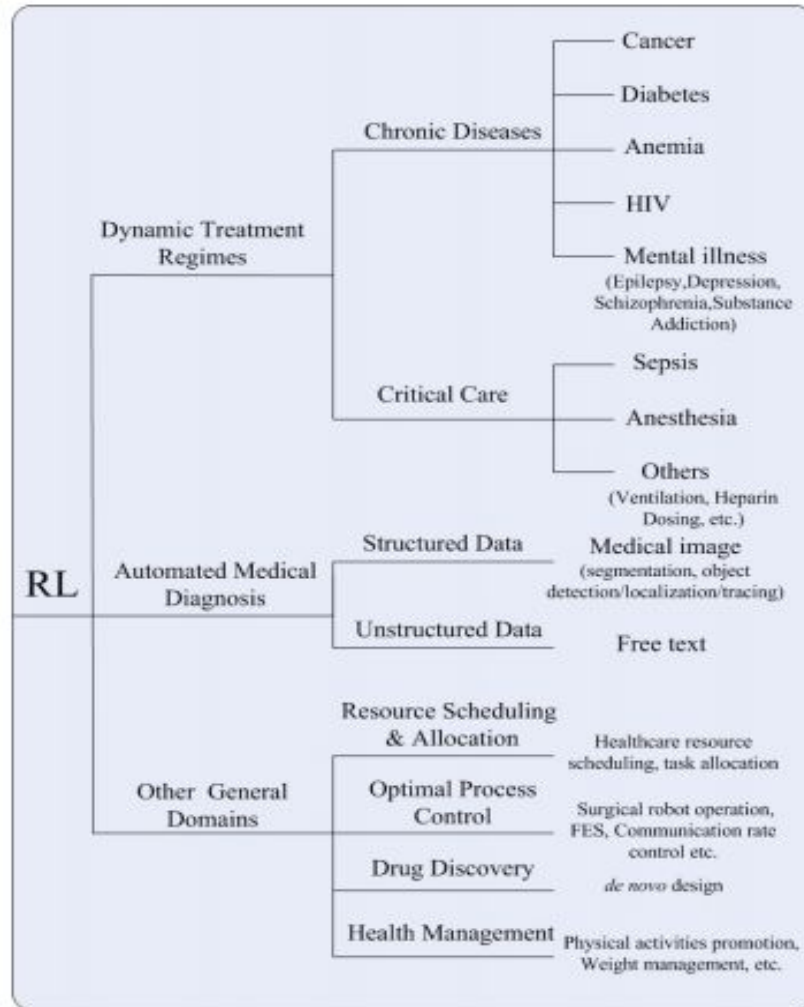


Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

Source: Peter Pastor [Youtube channel], 2018

From: <https://www.youtube.com/watch?v=W4joe3zzglU>

Success stories of RL: Healthcare



Example of dynamic treatment regimes (DTRs): To create a DTR, someone must input a set of clinical observations and assessments of a patient. Using previous outcomes and patient medical history, the learning system will then output a suggestion on treatment type, drug dosages, and appointment timing for every stage of the patient's journey.

Fig. 2. The outline of application domains of RL in healthcare.

Source: Yu, C., Liu, J., Nemati, S., & Yin, G. (2021). Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1), 1-36.

Success stories of RL: Natural Language Processing

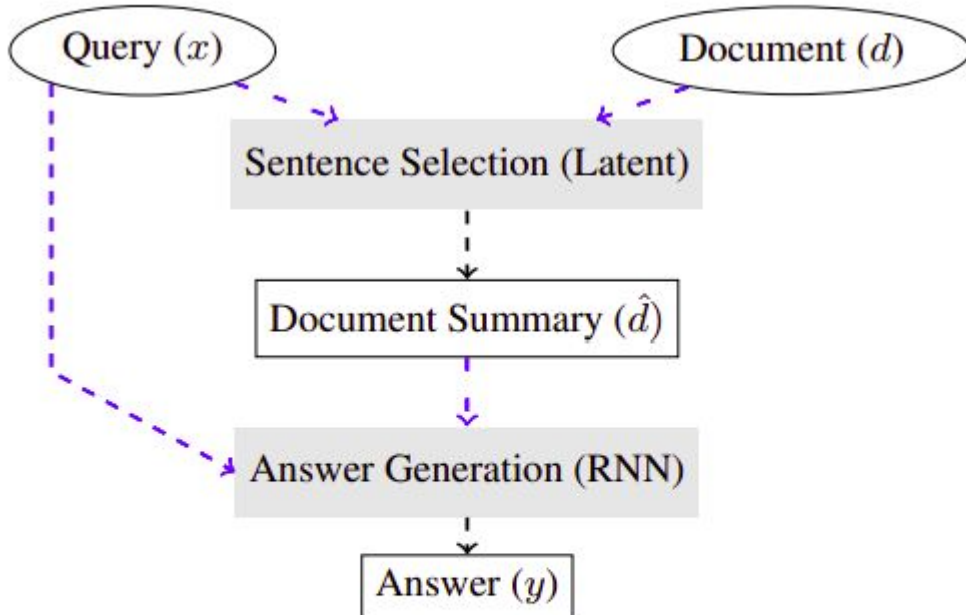


Figure 1: Hierarchical question answering: the model first selects relevant sentences that produce a document summary (\hat{d}) for the given query (x), and then generates an answer (y) based on the summary (\hat{d}) and the query x .

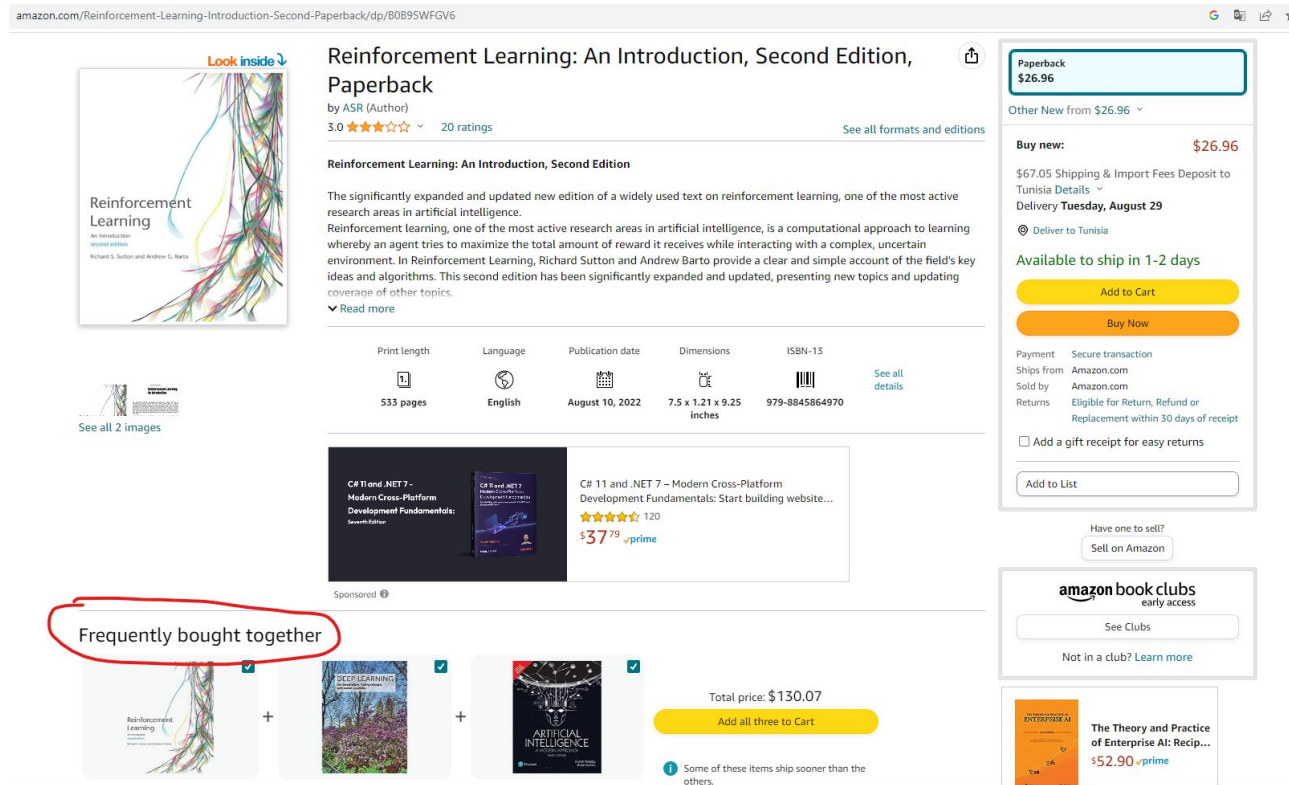
[Source](#): Neptune.ai
retrieved from: <https://neptune.ai/blog/reinforcement-learning-applications>

In NLP, RL can be used in text summarization, question answering, and machine translation just to mention a few.

For more details: [Link](#)

Success stories of RL: Recommendation systems

amazon.com/Reinforcement-Learning-Introduction-Second-Paperback/dp/B0B95WFGV6



Reinforcement Learning: An Introduction, Second Edition, Paperback
by ASR (Author)
3.0 ★★★★★ 20 ratings

Reinforcement Learning: An Introduction, Second Edition
The significantly expanded and updated new edition of a widely used text on reinforcement learning, one of the most active research areas in artificial intelligence. Reinforcement learning, one of the most active research areas in artificial intelligence, is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex, uncertain environment. In Reinforcement Learning, Richard Sutton and Andrew Barto provide a clear and simple account of the field's key ideas and algorithms. This second edition has been significantly expanded and updated, presenting new topics and updating coverage of other topics.

Print length: 533 pages
Language: English
Publication date: August 10, 2022
Dimensions: 7.5 x 1.21 x 9.25 inches
ISBN-13: 979-8845864970

Frequently bought together

Reinforcement Learning + Deep Learning + Artificial Intelligence: A Modern Approach

Total price: \$130.07
Add all three to Cart

Some of these items ship sooner than the others.

Paperback \$26.96
Other New from \$26.96

Buy new: \$26.96
\$67.05 Shipping & Import Fees Deposit to Tunisia
Delivery Tuesday, August 29
Deliver to Tunisia

Available to ship in 1-2 days
Add to Cart
Buy Now

Payment: Secure transaction
Ships from: Amazon.com
Sold by: Amazon.com
Returns: Eligible for Return, Refund or Replacement within 30 days of receipt
Add a gift receipt for easy returns
Add to List

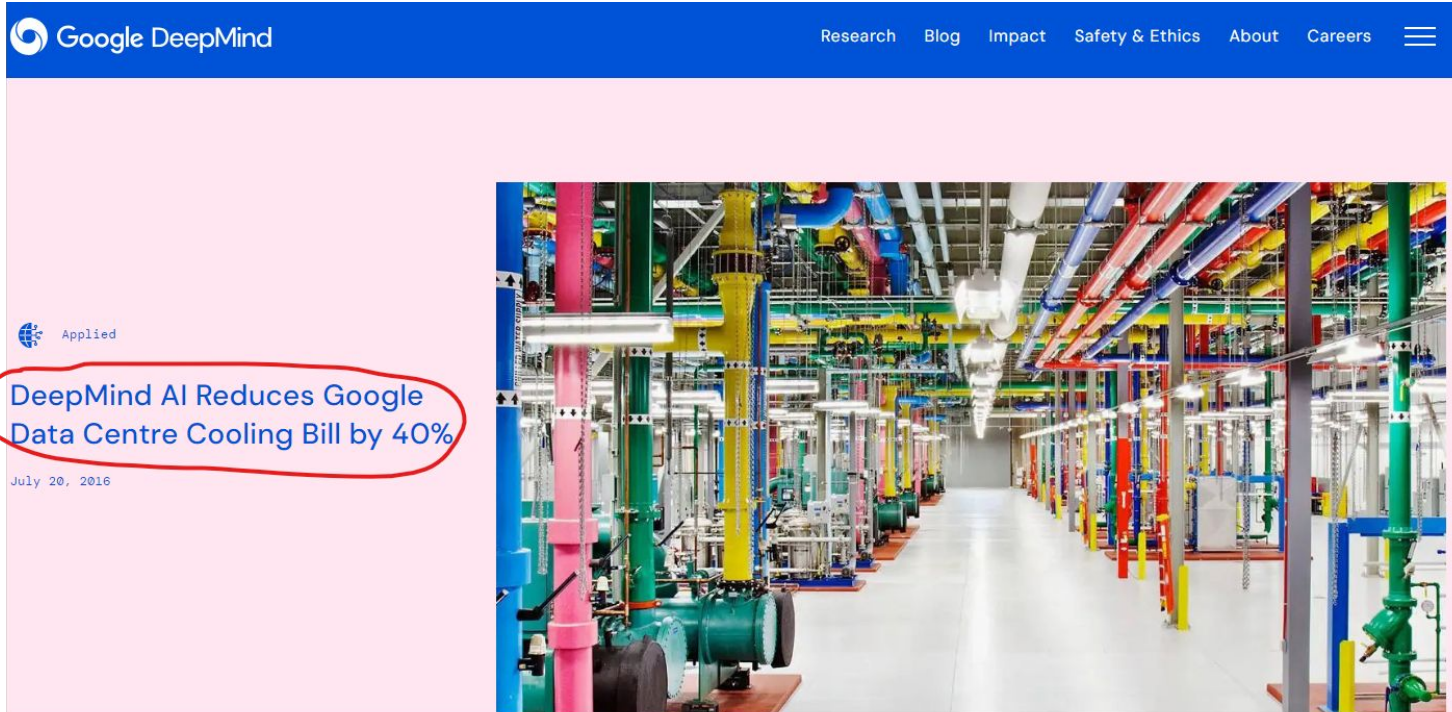
Have one to sell?
Sell on Amazon

amazon book clubs
early access
See Clubs
Not in a club? Learn more

The Theory and Practice of Enterprise AI: Recip...
\$52.90 prime

The “Frequently Bought Together” section on Amazon, a “Customers Also Liked” tab online at Target, and the “Recommended Reading” articles from news outlets all utilize learning machines to generate recommendations. Specifically for news reading, RL agents can track the types of stories, topics, and even author names someone prefers so that the system can queue the next story they think they would enjoy.

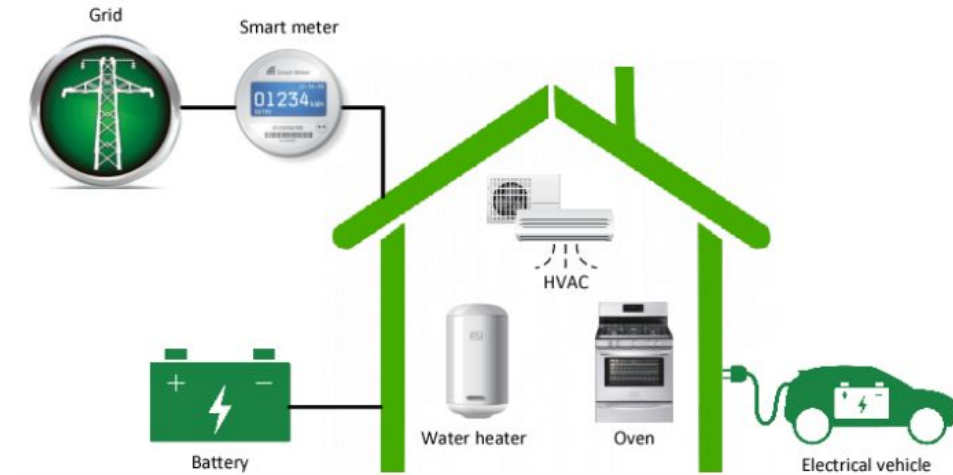
Success stories of RL: Energy Conservation



From smartphone assistants to image recognition and translation, machine learning already helps us in our everyday lives. But it can also help us to tackle some of the world's most challenging physical problems – such as energy consumption. Large-scale commercial and industrial systems like data centres consume a lot of energy, and while much has been done to stem the growth of energy use, there remains a lot more to do given the world's increasing need for computing power.

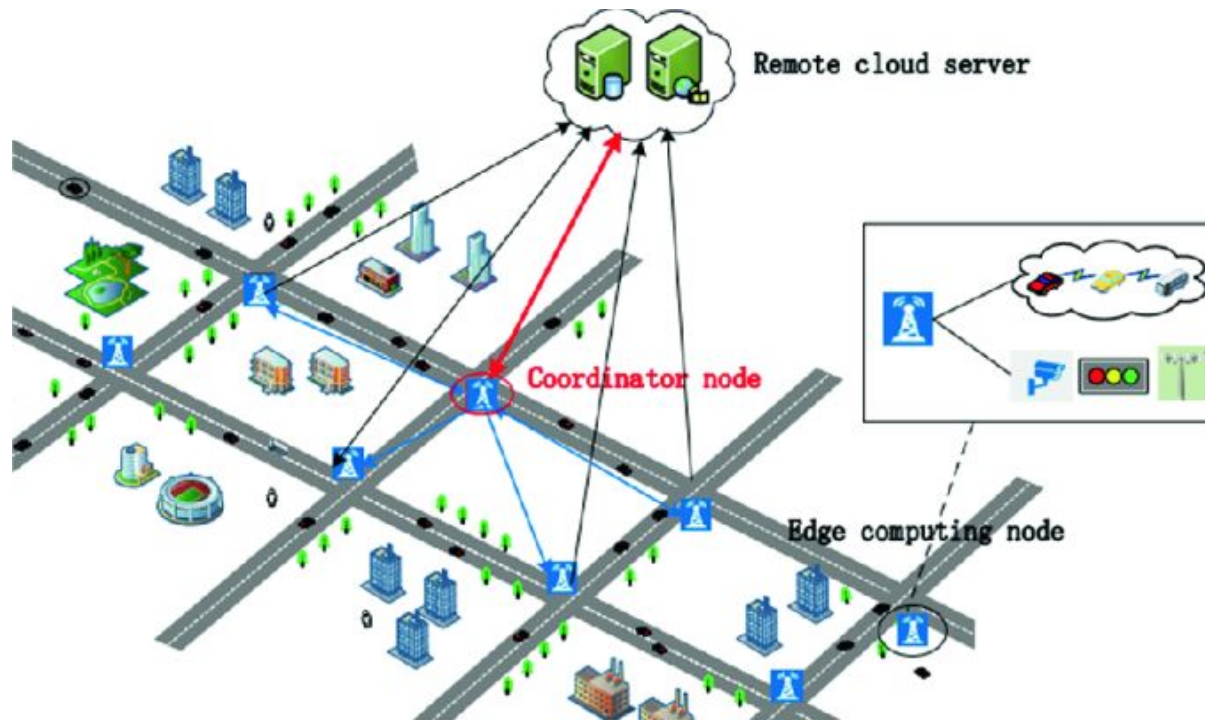
Reducing energy usage has been a major focus for us over the past 10 years: we

[Source](#)



RL-based Home energy management systems

Success stories of RL: Traffic Light Control



Source: Kim, D., & Jeong, O. (2019). Cooperative traffic signal control with traffic flow prediction in multi-intersection. *Sensors*, 20(1), 137.

The Continuous traffic monitoring in complex urban networks helps build a literal and figurative “map” of traffic patterns and vehicle behavior.

Success stories of RL: Marketing and advertising



For example, marketing and advertising platforms can use RL to associate similar companies, products, and services to prioritize for certain customers.

[Source](#)



03



Key concepts and terminology



Reinforcement Learning

Reinforcement Learning is the study of agents and how they learn to perform complex tasks by trial and error. It formalizes the idea that rewarding or punishing an agent for its behavior makes it more likely to repeat or forego that behavior in the future.

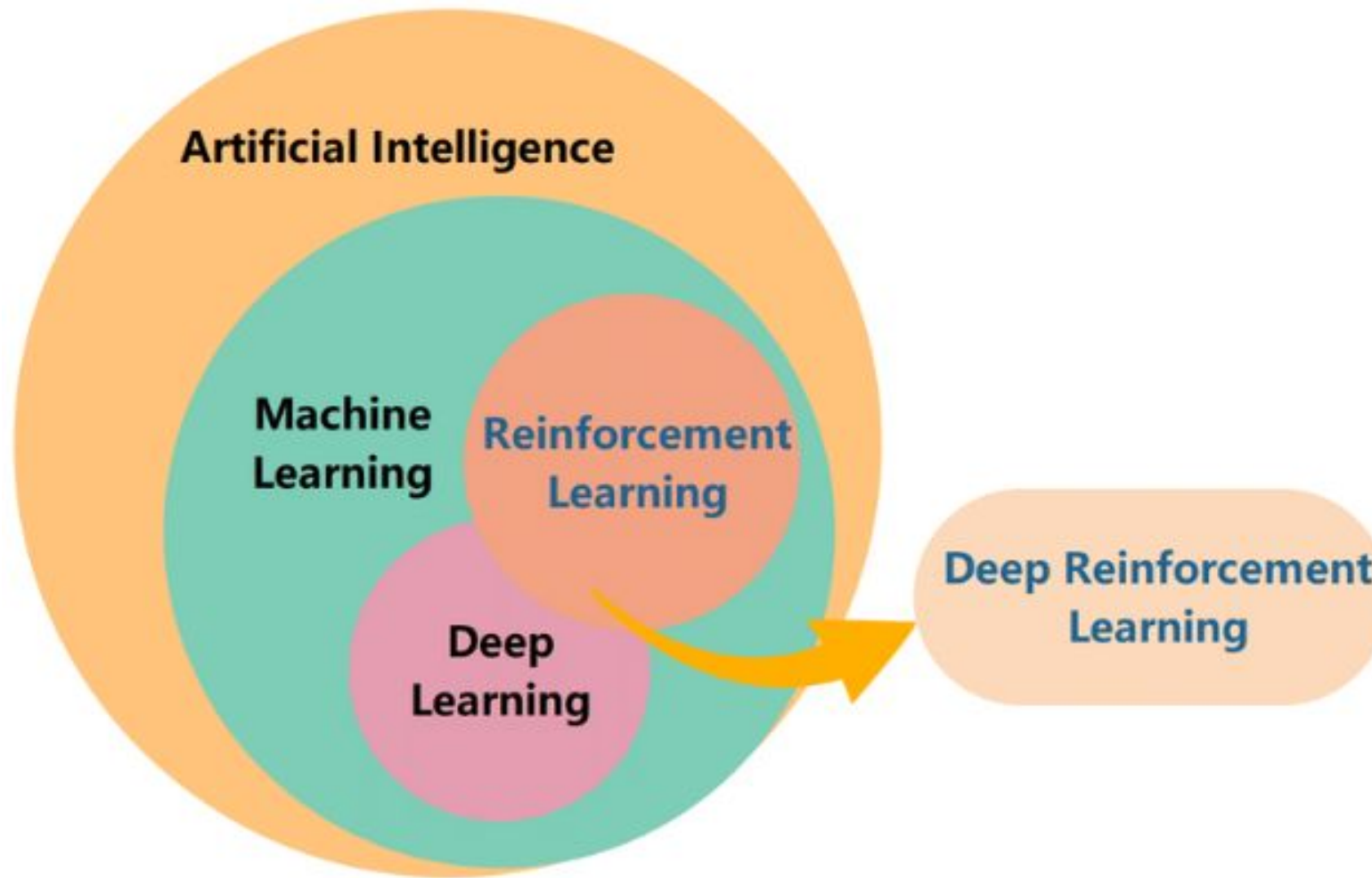


Source: TDM: From Model-Free to Model-Based Deep Reinforcement Learning, by Vitchyr Pong, 2018, Berkeley Artificial Intelligence Research



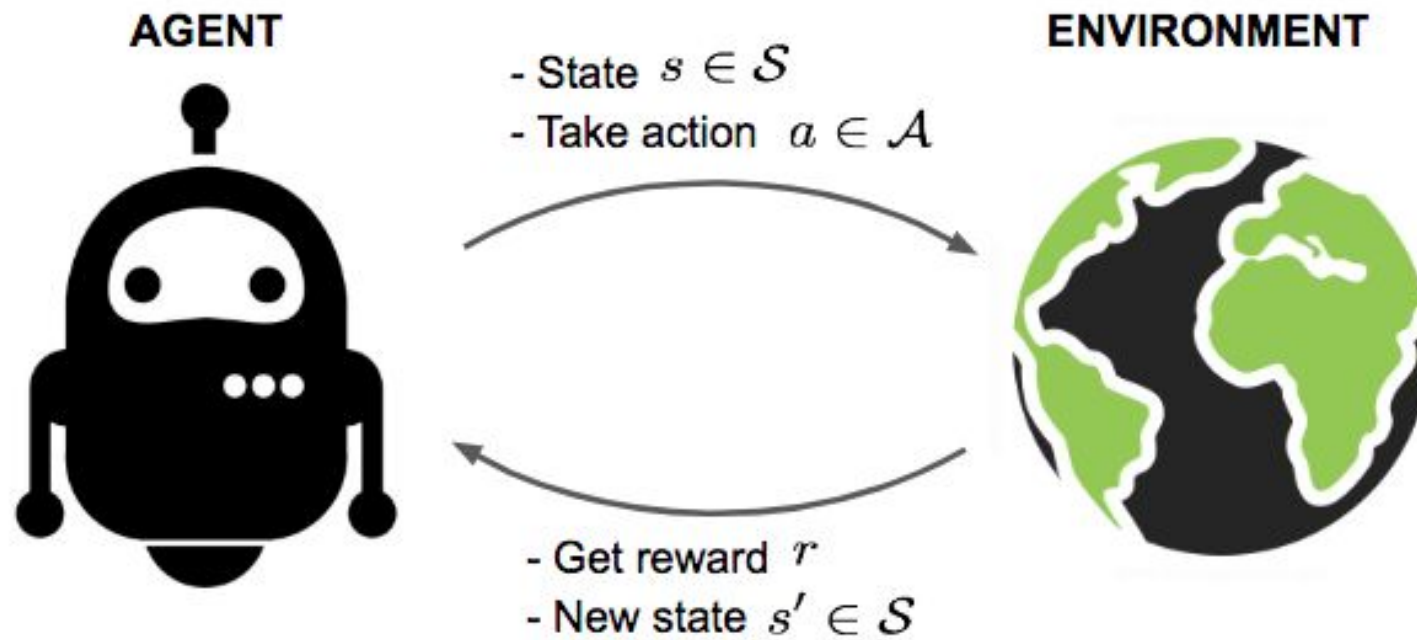
Source: Simple Beginner's guide to Reinforcement Learning & its implementation, by JalFaizy Shaikh, 2017, Analytics Vidhya

Deep Reinforcement Learning



Source: Kim, D., & Jeong, O. (2019). Cooperative traffic signal control with traffic flow prediction in multi-intersection. *Sensors*, 20(1), 137.

RL framework



Source: Reinforcement Learning Real-world examples, by Ajitesh Kumar, 2022, Analytics Yogi.

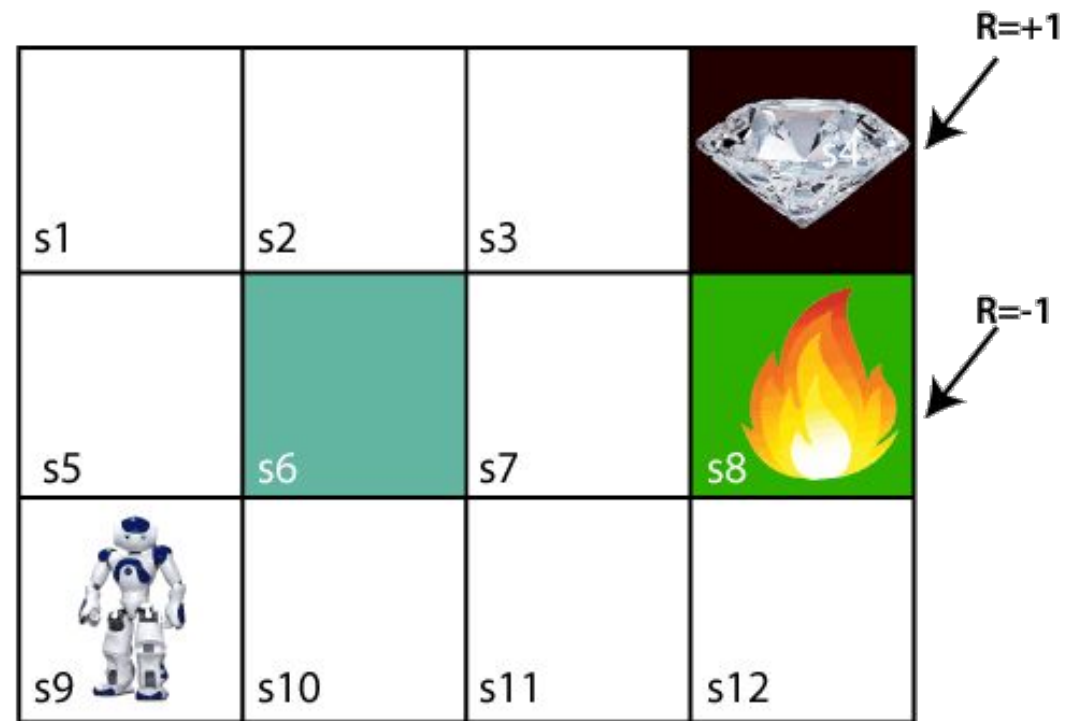
RL framework: example#1

Goal: reach the diamond block (S4)

State: the position of the agent (s1 or s2, or s3, etc.)

Action: move up, down, right, left

Reward: (+1) reward if it reaches the diamond block and (-1) reward if it reaches the fire pit.



Source: Reinforcement Learning Tutorial, Javatpoint

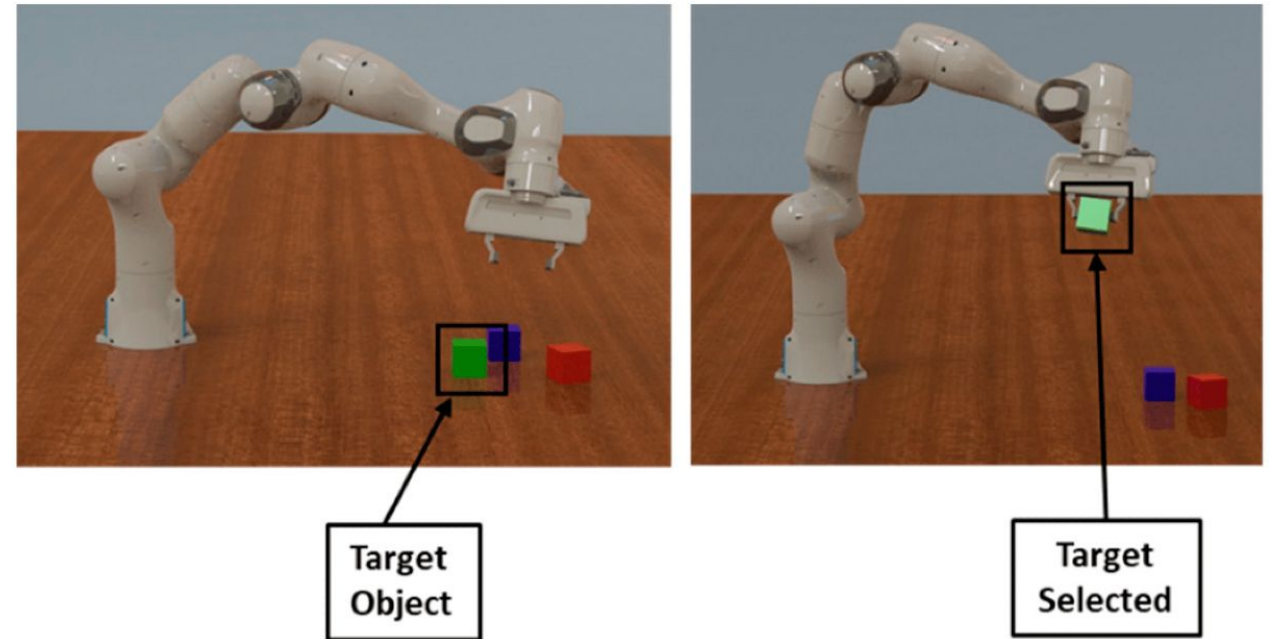
RL framework: example#2

Goal: pick objects with different shapes

State: Raw pixels from camera

Action: move arm, grasp

Reward: positive (+1) when pickup is successful



Source: Introduction to Deep Reinforcement Learning (Deep RL) by Lex Fridman, 2019, MIT Deep Learning 6.S191

Markov decision process

An RL problem is typically formulated as a Markov Decision Process (MDP)

A (finite) MDP consists in a tuple (S, A, P, R, γ) where:

- S is a (finite) set of states
- A is a (finite) set of possible actions
- $P: S \times A \times S \rightarrow [0, 1]$, P is the transition probability distribution modeling the system dynamics: $P(s, a, s') = P(st+1=s' \mid st=s, at=a)$
- $R: S \times A \times S \rightarrow \mathbb{R}$ (set of real numbers) is the reward function
- γ : the discount factor

Markov decision process

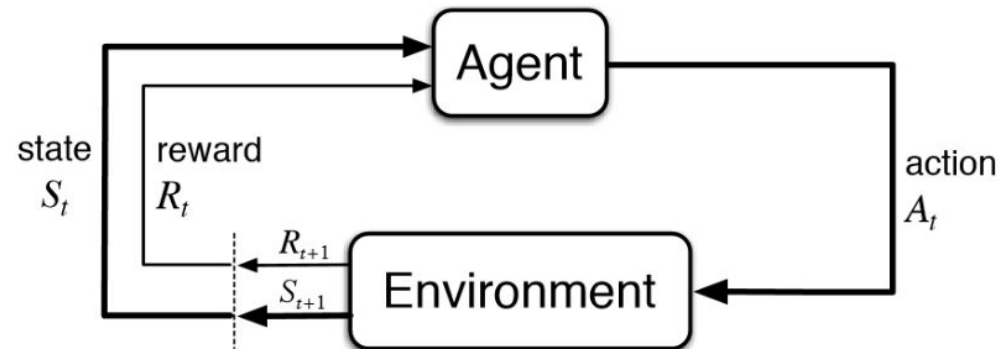
Markov Property:

- the future is independent of the past given the present

A state S is Markov if and only if:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \dots, S_t]$$

- Once the current state is known, the history of information encountered so far may be thrown away, and that state is a sufficient statistic that gives us the same characterization of the future as if we have all the history.

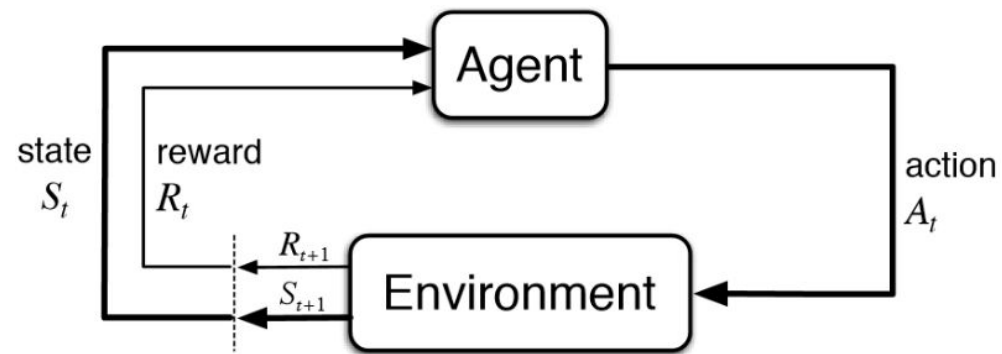


Markov decision process

In a typical RL problem, at time t , an agent interacts with the environment and takes an action a_t when the environment is in state s_t . Consequently, the environment moves to a new state s_{t+1} , and the agent receives a reward r_{t+1} that expresses how good its action was.

Note that 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.

The goal of the MDP is to find a policy, often denoted as π , that yields the optimal long-term reward.



Markov decision process

Markov Property:

- Could the blackjack card game verify the Markov Property?



Source: Mobile Premier league, 2022

Markov decision process

Markov Property:

- Could the blackjack card game verify the Markov Property?



Source: Mobile Premier league, 2022

- The game can be played successfully just by knowing our current state (what cards we have in hand and the opponent's one face-up card)

MDP: Quiz!

- Q1: Which of the following control problem or decision task could have a Markov property?

Driving a car (1)

Decide whether to Invest in a stock or not (2)

Choose a medical treatment for a patient (3)

Diagnose a patient's illness (4)



States and observation

A state \mathbf{s} describes the environment completely to the agent. Nothing is hidden in the state.

- e.g: a robot might have states like joint angles, velocity, position as the states defining it.

An observation \mathbf{o} is a partial description of a state, which may omit information.

- **Complete Observation:** when the agent is able to observe the complete state information that defines the status of the environment/ world after the agent has acted out a certain action, in it.
 - e.g: chess can be represented completely by the positions of all the pieces on the board.
- **Partial Observation:** when the agent is able to observe only partial information regarding the state of the environment.
 - e.g: poker since a player cannot observe other players' cards

States and observation

A state be represented as:

- Scalar (rank-0 tensor): temperature
- Vector (rank-1 tensor): [position, velocity, angle, angular velocity]
- Matrix (rank-2 tensor): grayscale pixels from an Atari game
- Data cube (rank-3 tensor): RGB color pixels from an Atari game

State preprocessing:

- Cleanup
- Numerical representation
- Standardization: so that each feature has a similar range and mean
- ...

Actions

Actions are the things that the agent can perform in order to influence the environment. Different environments allow different kinds of actions. We distinguish:

- **Discrete actions** mean that the agent has a finite action space to take the action from. Example: In a maze, the agent can go up, down, left or right..
- **Continuous actions** have some value attached to the action. The agent has an infinite action space. Example: For a robotic arm, actions involve controlling the angles or positions of its joints, which exist in a continuous space.
- **Hybrid actions:** mix of both. Example: for a robot learning to navigate and pick up objects, actions include discrete movements (forward, backward, turn) and continuous actions for grasping objects (adjust gripper position or force)

The set of all valid actions in a given environment is called the **action space**.

Reward

A reward is a scalar feedback signal that indicates how well the agent is doing at a given step.

$$r_{t+1} = R(s_t, a_t, s_{t+1})$$

where $R(s_t, a_t, s_{t+1})$ denotes the the reward received for being in a state s_t , taking an action a_t and ending up in a state s_{t+1} .

The reward is frequently simplified to just a dependence on the current state $r_{t+1} = R(s_t)$ or state-action pair $r_{t+1} = R(s_t, a_t)$

The agent's sole objective is to maximize the total reward it receives over the long run. The reward should therefore be designed in a way that incentivizes the desired behavior.

Reward

A reward signal can be dense or sparse:

- A **sparse** reward is one which produces a neutral reward signal (usually $r = 0$) for most of the time steps, and a positive and negative reward only when the environment terminates.
- A **dense** reward is the opposite: it provides a lot of nonneutral reward signals indicating whether the last action was good or bad, so that in most time steps an agent will receive a positive or negative reward.
- Sparse reward are usually more challenging than dense rewards

Sparse Rewards

4					+1k
3					
2					
1					
0	I				
	0	1	2	3	4

Dense Rewards

4					+1k
3	+3	...			
2	+2	+3	...		
1	+1	+2	+3	...	
0	I	+1	+2	+3	
	0	1	2	3	4

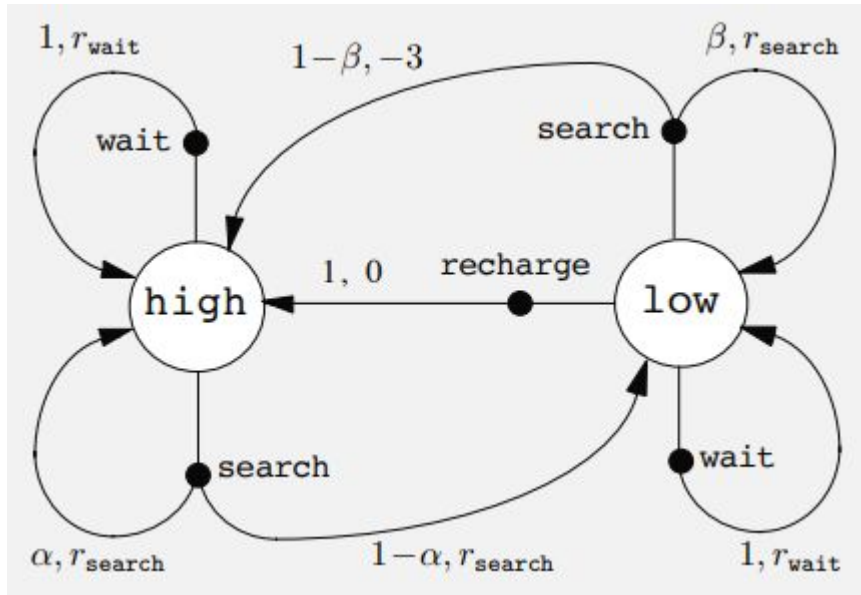
Example of MDP Formulation

- A mobile robot has the job of collecting empty soda cans
- It runs on a rechargeable battery
- Robot decisions are based on the energy level: high or low
- In each state, the agent can decide to (1) search for a can for a certain period of time (2) remain stationary and wait for someone to bring it a can (3) recharge battery
- Reward is
 - Generally equal to the number of collected cans
 - 0 in recharge mode
 - negative if runs out of power while searching (very bad as agent needs to be rescued)



Source: Google Research, 2023

Example of MDP Formulation



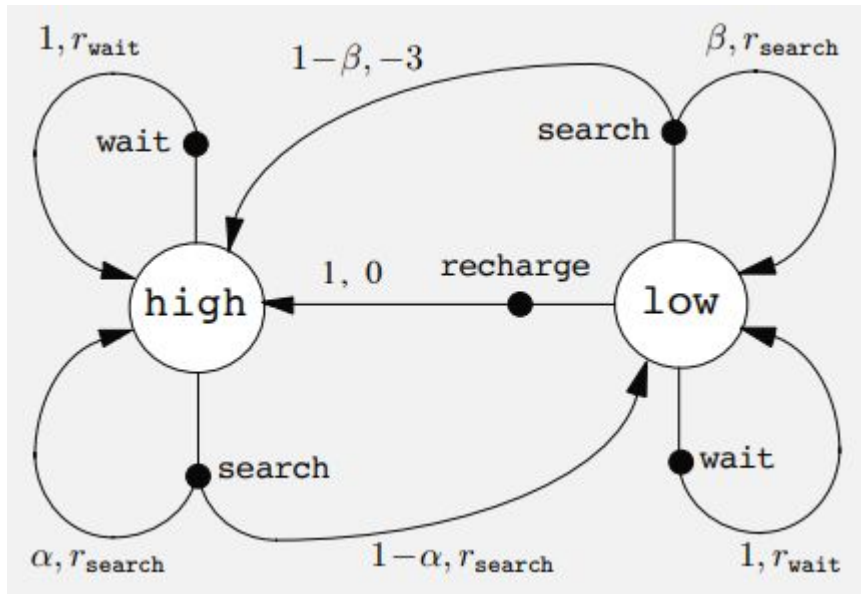
Source: Reinforcement Learning-An Introduction, a book by Richard Sutton



Source: Google Research, 2023

- State Space?
- Set of possible actions ? $A(\text{low})$ and $A(\text{high})$
- Reward?
- Transition probabilities?

Example of MDP Formulation: State



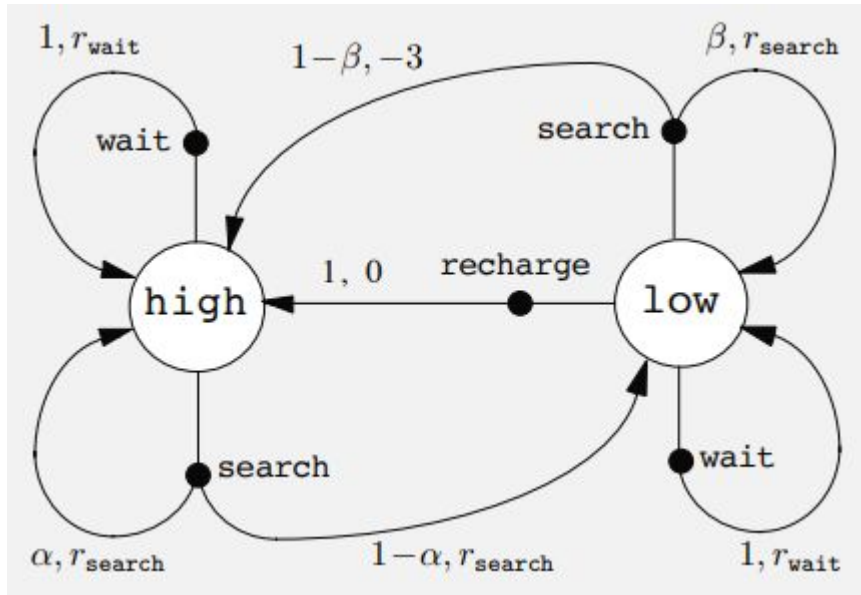
Source: Reinforcement Learning-An Introduction, a book by Richard Sutton



Source: Google Research, 2023

- State Space={high, low}

Example of MDP Formulation: Action



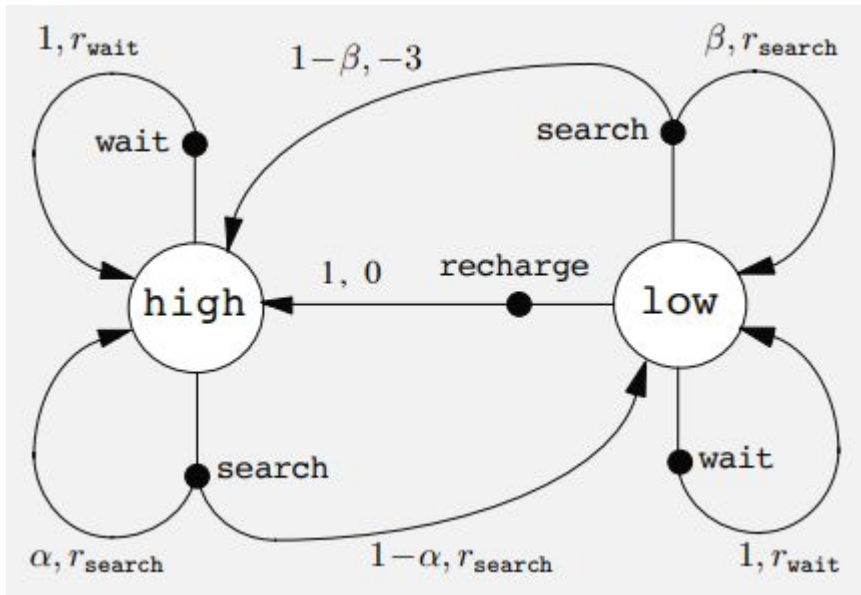
Source: Reinforcement Learning-An Introduction, a book by Richard Sutton



Source: Google Research, 2023

- Set of possible actions ?
 - $A(\text{low}) = \{\text{search}, \text{recharge}, \text{wait}\}$
 - $A(\text{high}) = \{\text{search}, \text{wait}\}$

Example of MDP Formulation: Transition Probability



Source: Reinforcement Learning-An Introduction, a book by Richard Sutton

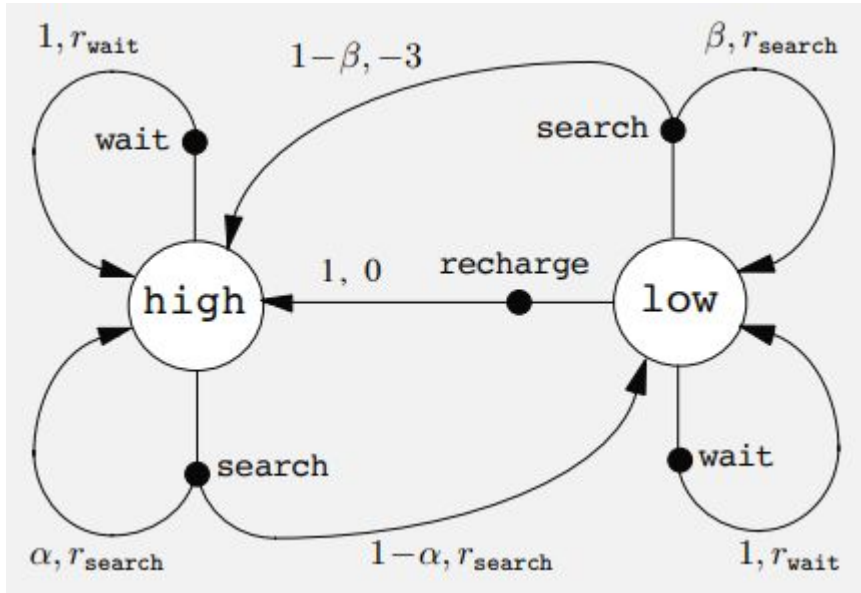


Source: Google Research, 2023

- Transition probabilities?

s	a	s'	$p(s' s, a)$
high	search	high	α
high	search	low	$1 - \alpha$
low	search	high	$1 - \beta$
low	search	low	β
high	wait	high	1
high	wait	low	0
low	wait	high	0
low	wait	low	1
low	recharge	high	1
low	recharge	low	0

Example of MDP Formulation: Reward



Source: Reinforcement Learning-An Introduction, a book by Richard Sutton



Source: Google Research, 2023

- Reward?

s	a	s'	$r(s, a, s')$
high	search	high	r_{search}
high	search	low	r_{search}
low	search	high	-3
low	search	low	r_{search}
high	wait	high	r_{wait}
high	wait	low	$-$
low	wait	high	$-$
low	wait	low	r_{wait}
low	recharge	high	0
low	recharge	low	$-$

Episodic and continuing tasks

We refer to a complete sequence of interaction, from start to finish, as an **episode**.

$$S_1, A_1, R_2, S_2, A_2, \dots, S_T$$

Episodes are also called Trajectories or rollouts.

Episodic tasks come to an end whenever the agent reaches a terminal state.
e.g: super mario

Continuing tasks are tasks that continue forever (no terminal state). e.g:
stock trading

Episodic and continuing tasks

Episodic tasks have a starting point and an ending point (a terminal state)



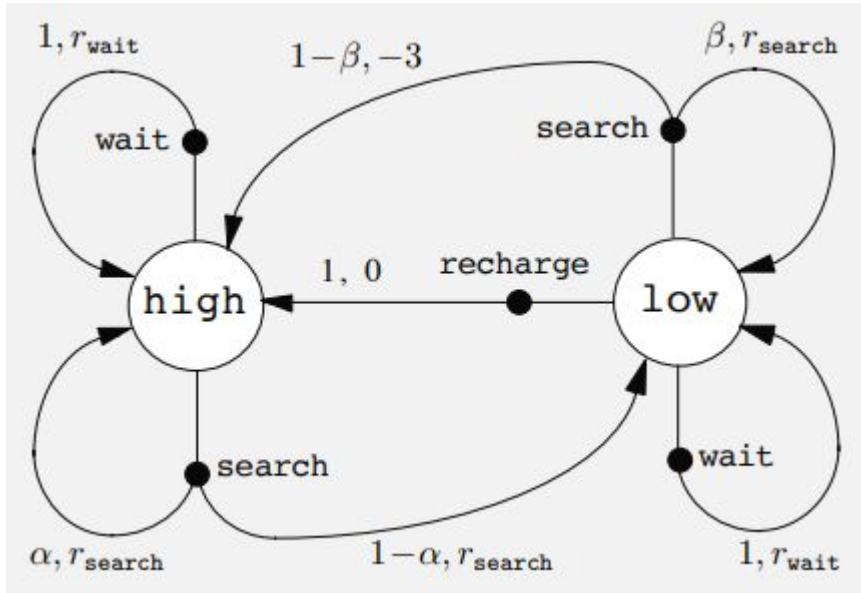
Source: jeuxvideo.com, 2023

Continuing tasks continue forever (no terminal state)



Source: droitdunet.fr, 2023

Example of MDP Formulation: Reward



Source: Reinforcement Learning-An Introduction, a book by Richard Sutton



Source: Google Research, 2023

- Reward?

$$r(s, a) = \sum_{s'} p(s'|s, a) r(s, a, s')$$

s	a	s'	$r(s, a, s')$
high	search	high	r_{search}
high	search	low	r_{search}
low	search	high	-3
low	search	low	r_{search}
high	wait	high	r_{wait}
high	wait	low	-
low	wait	high	-
low	wait	low	r_{wait}
low	recharge	high	0
low	recharge	low	-

Return

The return G_t is the total discounted reward from time-step t .

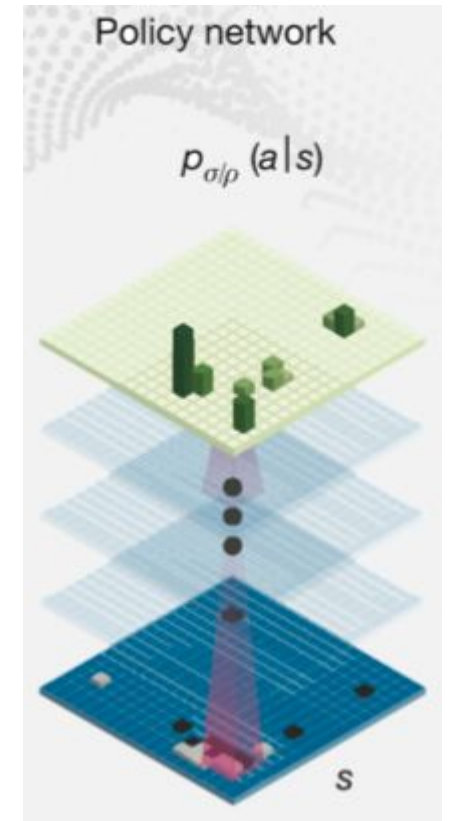
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

The discount $\gamma \in [0, 1]$ characterizes the “**foresightedness**” of the agent:

- When this discount rate is close to zero, the agent will care more about the immediate reward.
- When it is close to 1, the agent will care more about the long term reward.
- Most of the time, is set to something between 0.9 and 0.99:
 - In this case, we look into future rewards, but not too far

Policy

- A policy is the “brain” of the agent: It is the function that tells us what action to take given the state we are in. It maps states into action in order to enable the agent of maximum reward.
- We often denote the parameters of such a policy by θ :
 - Deterministic policy: is a mapping $\pi: S \rightarrow A$.
 - At a given state, the policy will always return the same action
 - Stochastic policy: is a mapping
 - $\pi: S \times A \rightarrow [0,1]$
 $\pi(a|s) = P(A_t=a | S_t = s)$
 - The policy outputs a probability distribution over actions



Policy: Quiz!

- Q1: Consider a deterministic policy $\pi: S \rightarrow A$ where: $\pi(\text{low}) = \text{search}$, $\pi(\text{high}) = \text{search}$

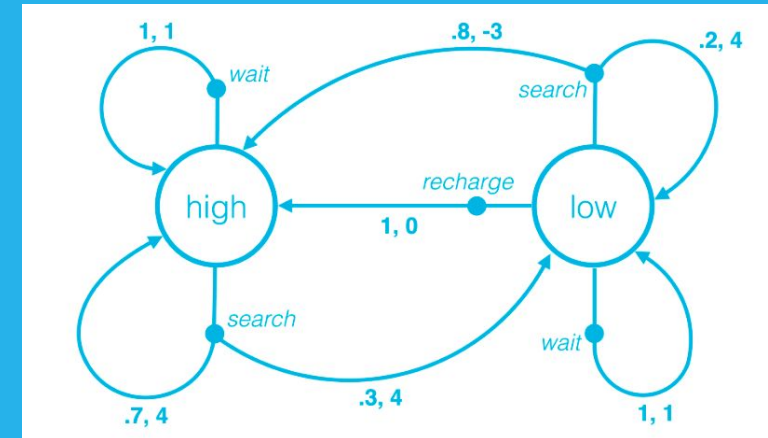
Which of the following statements are true, if the agent follows the policy?

If the state is low, the agent chooses action search

If the action is low, the agent chooses state search.

The agent will always search for cans at every time step.

If the state is high, the agent chooses to wait for cans.

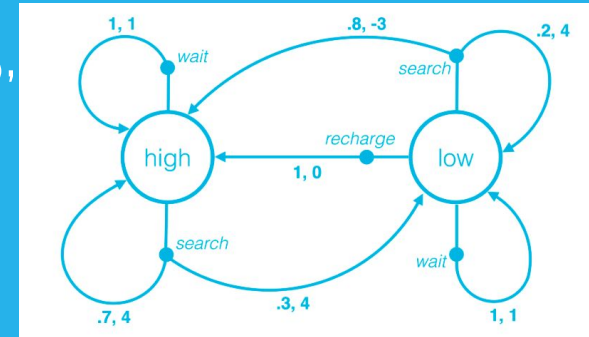


Source: Reinforcement Learning 101 by Srimanth Tenneti, 2020, Analytics Vidhya

Policy: Quiz!

- Q2: Consider a different stochastic policy
- $\pi: S \times A \rightarrow [0,1]$ where: $\pi(\text{recharge} | \text{low})=0.3$, $\pi(\text{wait} | \text{low})=0.5$, $\pi(\text{search} | \text{low})=0.2$, $\pi(\text{search} | \text{high})=0.6$ $\pi(\text{wait} | \text{high})=0.4$

Which of the following statements are true, if the agent follows the policy?



Source: Reinforcement Learning 101 by Srimanth Tenneti,
2020 Analytics Vidhya

If the battery level is low, the agent will always decide to wait for cans.

If the battery level is high, the agent chooses to search for a can with 60% probability, otherwise waits for a can.

If the battery level is low, the agent is most likely to decide to wait for cans.

Gridworld example

Environment: The world is primarily composed of nine patches of grass. But two out of the nine locations have large mountains.

States: We will think of each of the nine possible locations in the world as states of the environment.

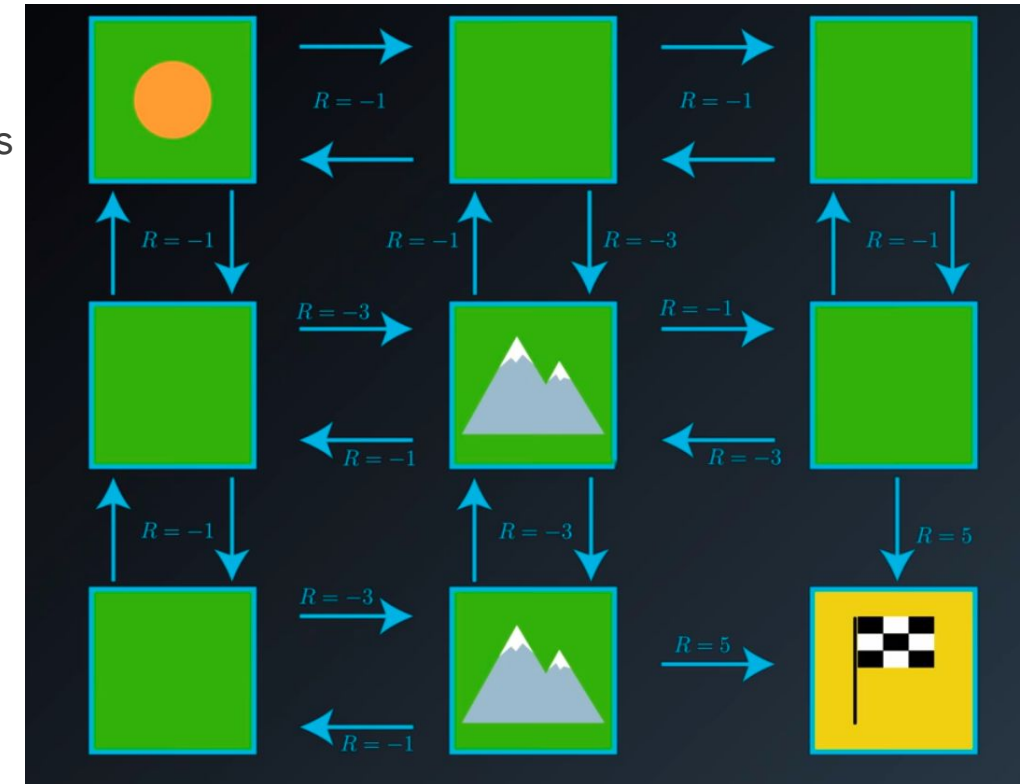
Actions: At each point in time the agent can only move up, down, left or right and can only take actions that lead it to not get off the grid. The arrows show the possible movements that we are allowed to take.

Goal of the agent: is to get to the bottom right hand corner of the grid as quickly as possible.

Episode ending condition: the episode finishes when the agent reaches the goal.

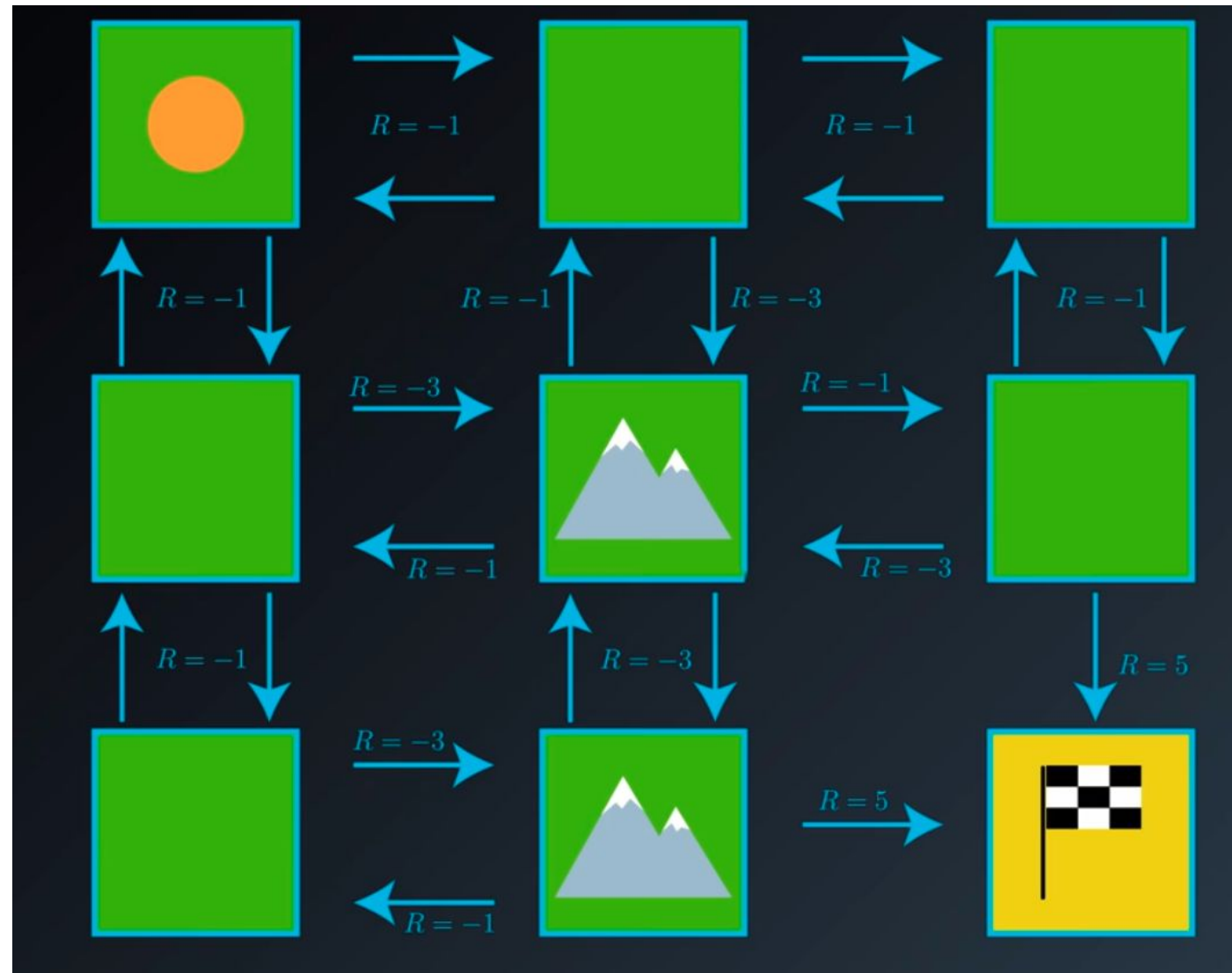
Reward: the agent receives a reward of:

- -1 for most transitions.
- If an action leads the agent to encounter a mountain it receives -3.
- And if it reaches the goal state it gets a reward of 5 and the episode ends.



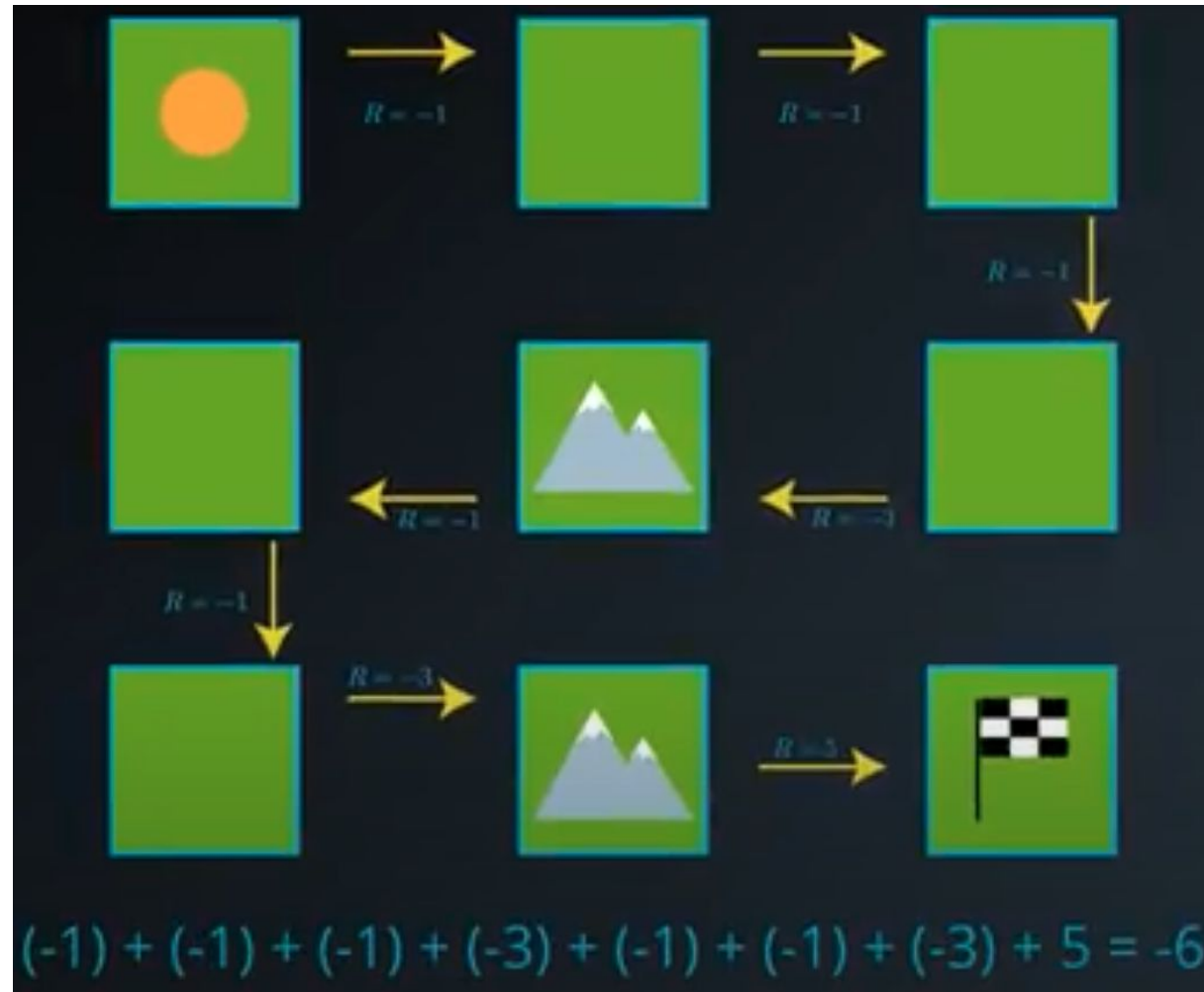
Source: Deep Reinforcement learning nanodegree program, Udacity 2022

Gridworld example



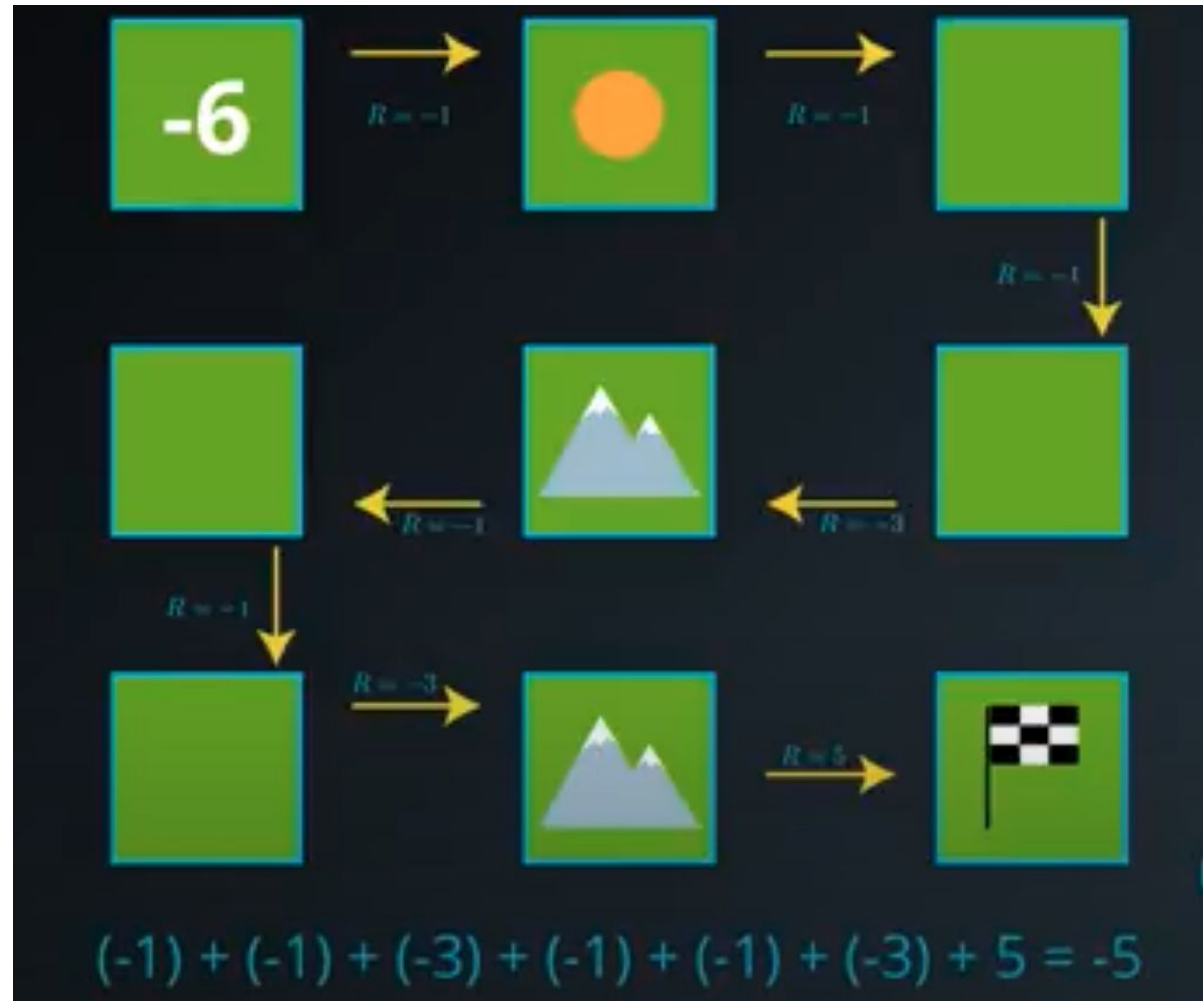
Source: Deep Reinforcement learning nanodegree program, Udacity 2022

Gridworld example

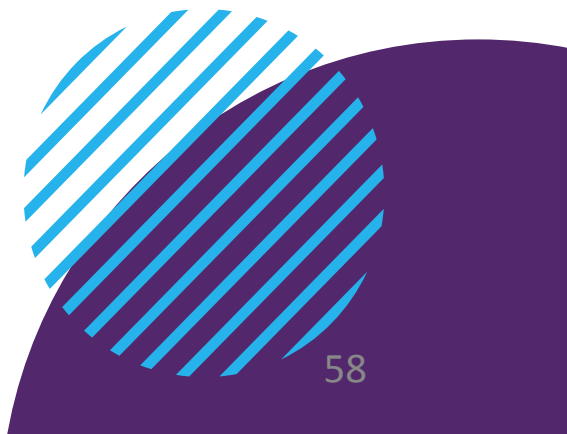


Source: Deep Reinforcement learning nanodegree program, Udacity 2022

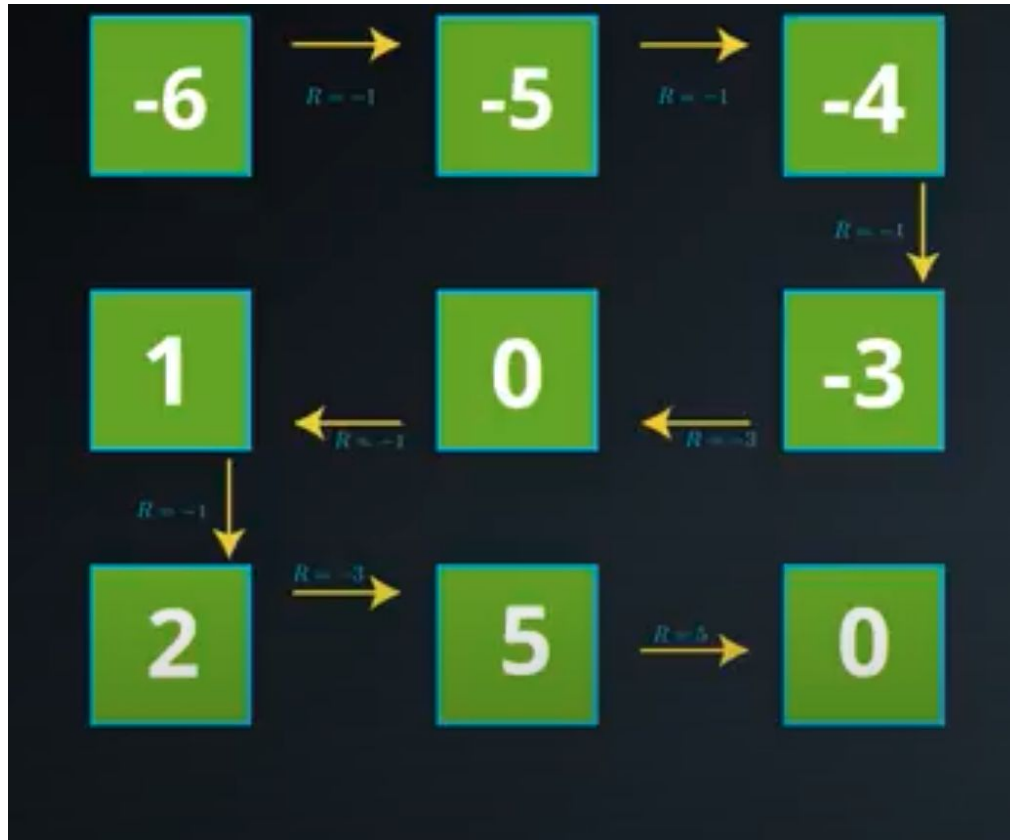
Gridworld example



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

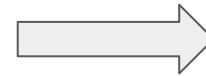


Gridworld example



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

For each state, the state-value function yields the expected return, if the agent started in that state, and then followed the policy for all time steps.



-6	-5	-4
1	0	-3
2	5	0

Value functions

The **value function** is the sum of the rewards the agent is expected to accumulate starting from the current state given a policy π . It evaluates the quality of the states. The state that will be chosen by the agent will be the state with the maximum value function. It is presented by the following equation:

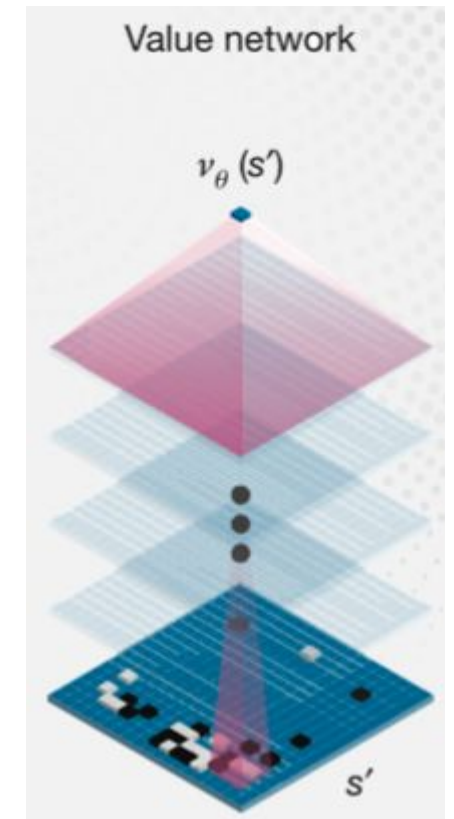
$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right],$$

For each **state s**

It yields the **expected return**

if the agent starts in **state s**

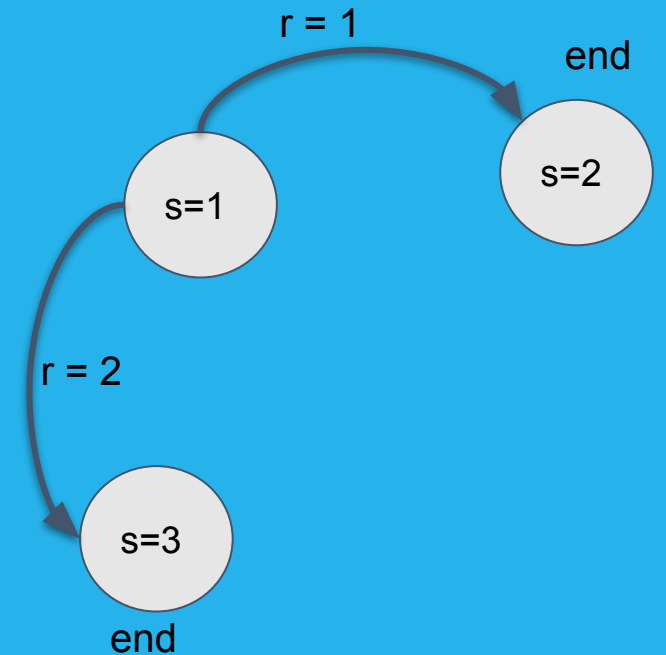
and then uses the **policy**



Value function: Quiz!

- Q1: Which of the following statements are correct:

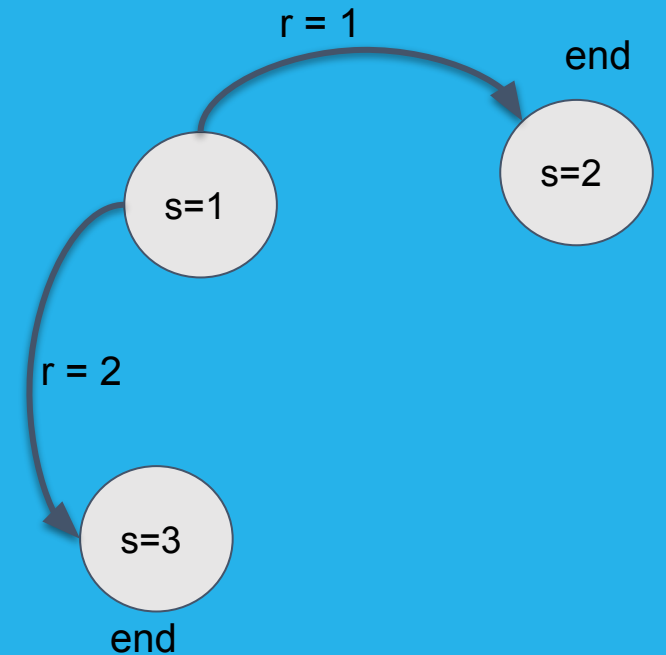
	The value of a state may differ depending on the policy that is being followed by the agent
	The value of a state depends on the initial state only regardless of the policy being followed
	Given a policy, the value function maps the states to the expected returns
	If $\gamma=1$, the the value function maps the states to the expected immediate reward



Value function: Quiz!

- Q1: Which of the following statements are correct:

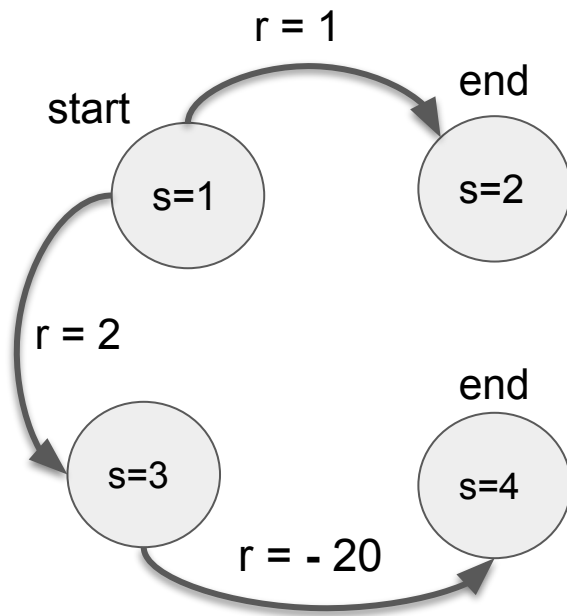
✗	The value of a state may differ depending on the policy that is being followed by the agent
	The value of a state depends on the initial state only regardless of the policy being followed
✗	Given a policy, the value function maps the states to the expected returns
	If $\gamma=1$, the the value function maps the states to the expected immediate reward



Value functions

An important observation is that the **value of a state** is not dependent on the immediate rewards only, but it considers the long term rewards (optionally discounted)

If you have the false impression that we should always take the action with the highest immediate rewards, then have a look at the following example:

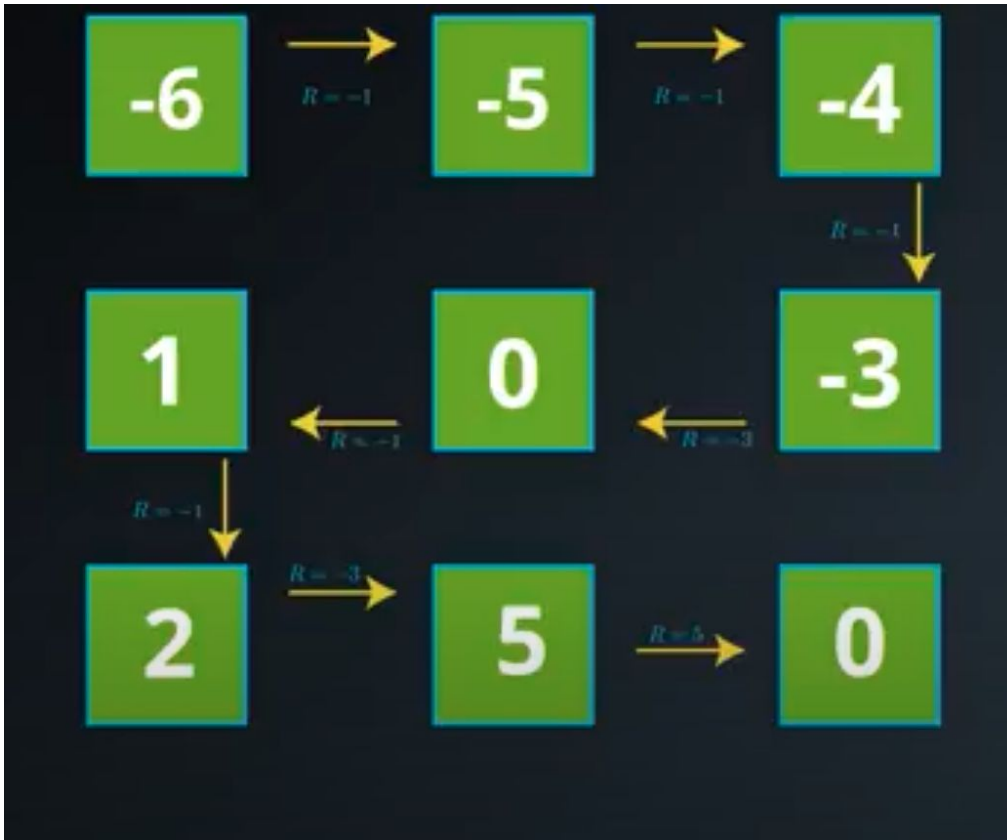


Following the states with maximum immediate reward could lead to a trap!



Source: iStockPhoto.com, 2023

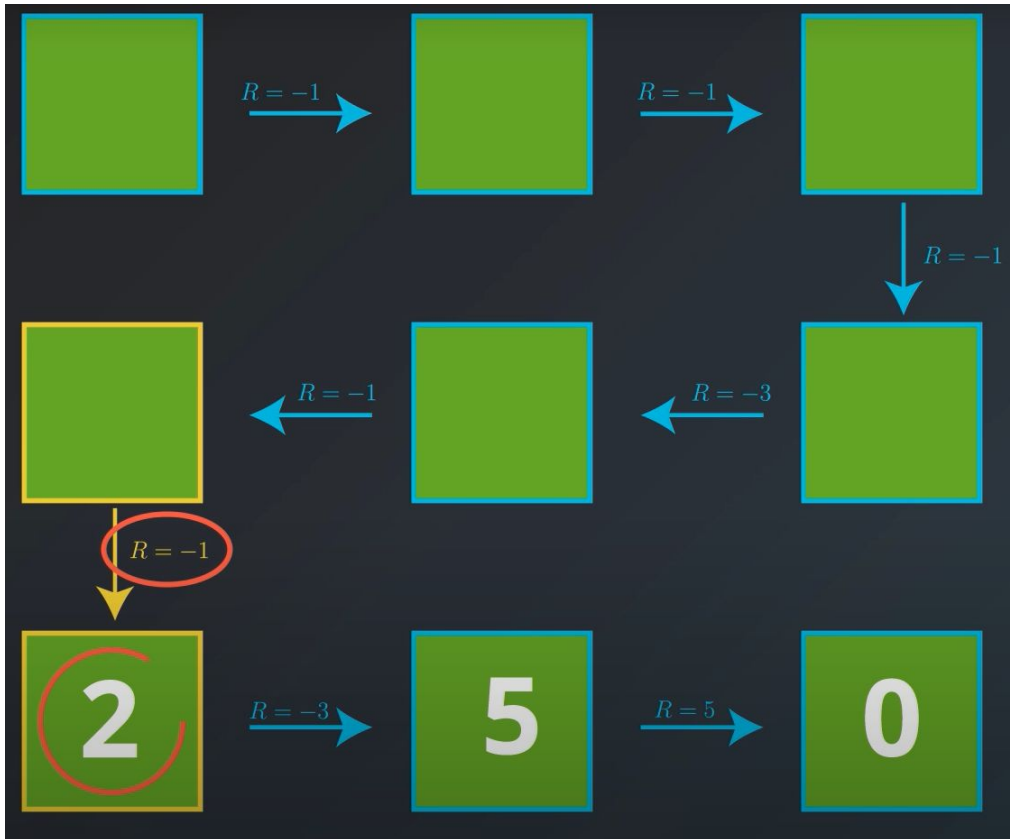
Gridworld example



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

$$\begin{aligned}(-1)+(-1)+(-1)+(-3)+(-1)+(-1)+(-3)+5 &= -6 \\(-1)+(-1)+(-3)+(-1)+(-1)+(-3)+5 &= -5 \\(-1)+(-3)+(-1)+(-1)+(-3)+5 &= -4 \\(-3)+(-1)+(-1)+(-3)+5 &= -3 \\(-1)+(-1)+(-3)+5 &= 0 \\(-1)+(-3)+5 &= 1 \\(-3)+5 &= 2 \\5 &= 5\end{aligned}$$

Gridworld example



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

$$V(st) = (-1) + (-3) + 5 = (-1) + V(st+1) = (-1) + (2) = 1$$

⇒ The value of any state can be expressed as the sum of the immediate reward and the **discounted** value of the state that follows.

(Suppose no discount $\gamma = 1$)

Bellman expectation equation

All bellman equations attest the fact that value functions satisfy recursive relationships.

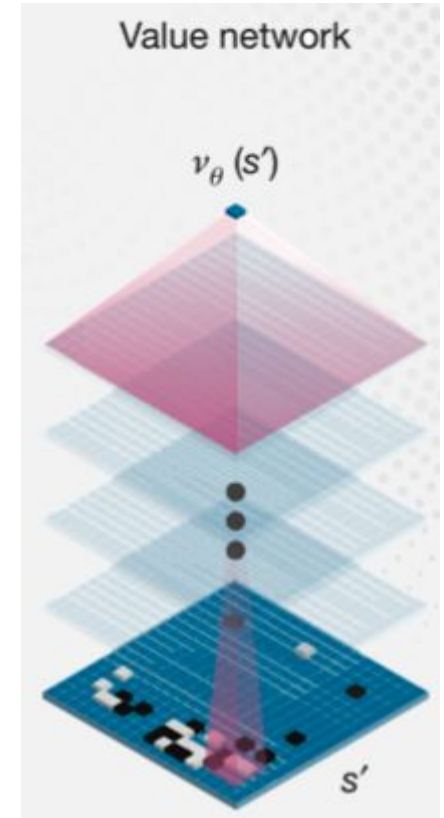
The state value function can be decomposed into immediate reward plus discounted value of successor state.

$$v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$$

The value of any **state** =

the **immediate reward** + the **discounted**

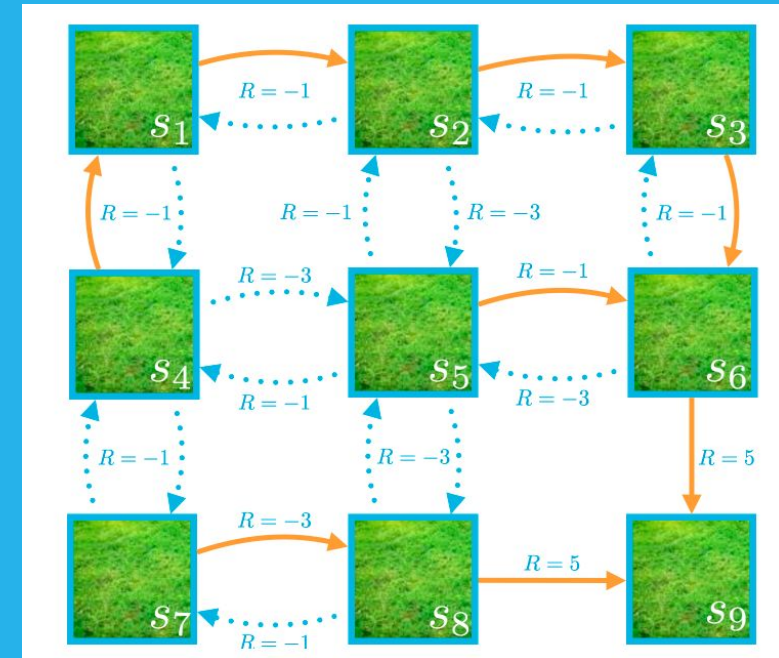
value of state that follows following **the policy**



Value function: Quiz!

- Q1: The policy given by: $\pi(s_1) = \text{right}$, $\pi(s_2) = \text{right}$, $\pi(s_3) = \text{down}$, $\pi(s_4) = \text{up}$, $\pi(s_5) = \text{right}$, $\pi(s_6) = \text{down}$, $\pi(s_7) = \text{right}$, $\pi(s_8) = \text{right}$
Assume $\gamma = 1$. What is $v(s_1)$?

	-1
	0
	1
	2

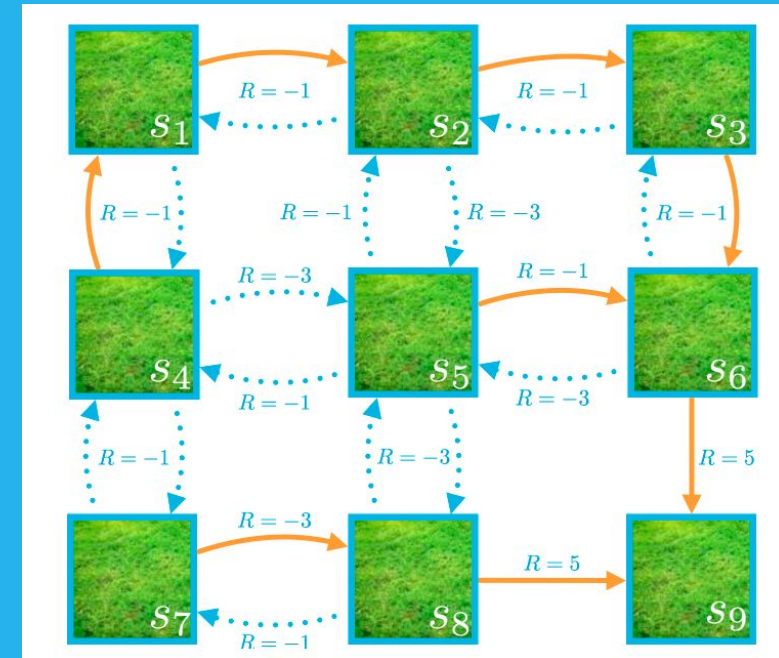


Source: Deep Reinforcement learning nanodegree program, Udacity 2022

Value function: Quiz!

- Q1: The policy given by: $\pi(s_1) = \text{right}$, $\pi(s_2) = \text{right}$, $\pi(s_3) = \text{down}$, $\pi(s_4) = \text{up}$, $\pi(s_5) = \text{right}$, $\pi(s_6) = \text{down}$, $\pi(s_7) = \text{right}$, $\pi(s_8) = \text{right}$
Assume $\gamma = 1$. What is $v(s_4)$?

	-1
	0
	1
	2



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

Value function: Quiz!

- Q3: Select the statements that are true:

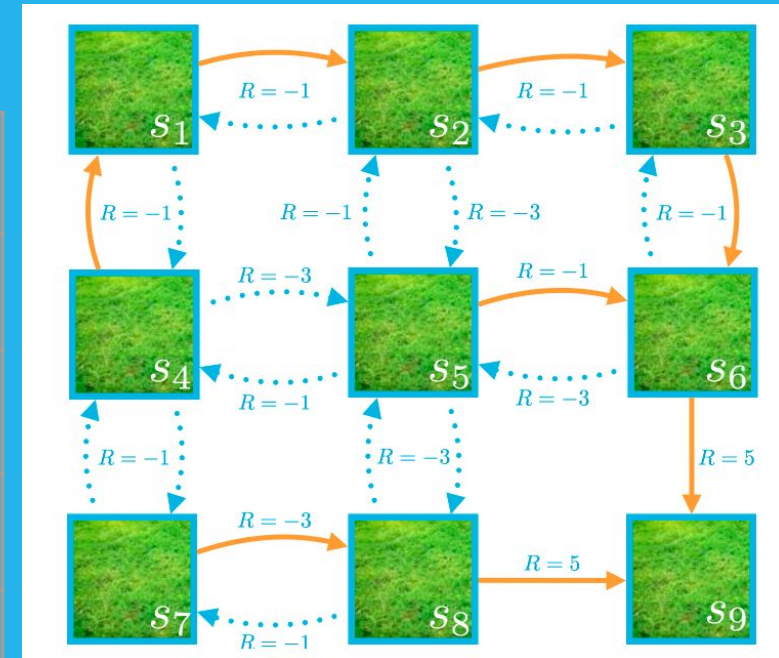
$$v(s_6) = -1 + v(s_5)$$

$$v(s_7) = -3 + v(s_8)$$

$$v(s_1) = -1 + v(s_2)$$

$$v(s_4) = -3 + v(s_7)$$

$$v(s_8) = -3 + v(s_5)$$



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

What's being optimal

- When is a RL problem solved?
 - When we find a policy π that achieves a lot of reward **over the long run** → it's not about maximizing the immediate rewards
- What can help us in solving this optimality problem is the findings of the mathematician Bellman and, in particular, it's his famous Bellman equation. We'll see this in more details later on.
- As we are looking for the optimal policy for making decisions, we need a criteria that allows us to order policies or give a rank for each one.
 - Value functions define a partial ordering over policies



Source: Wikipedia, 2023

Richard Ernest Bellman
(August 26, 1920 – March 19, 1984)

"In the first place I was interested in planning, in decision making, in thinking. But planning is not a good word for various reasons. I decided therefore to use the word, "programming." I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying-"

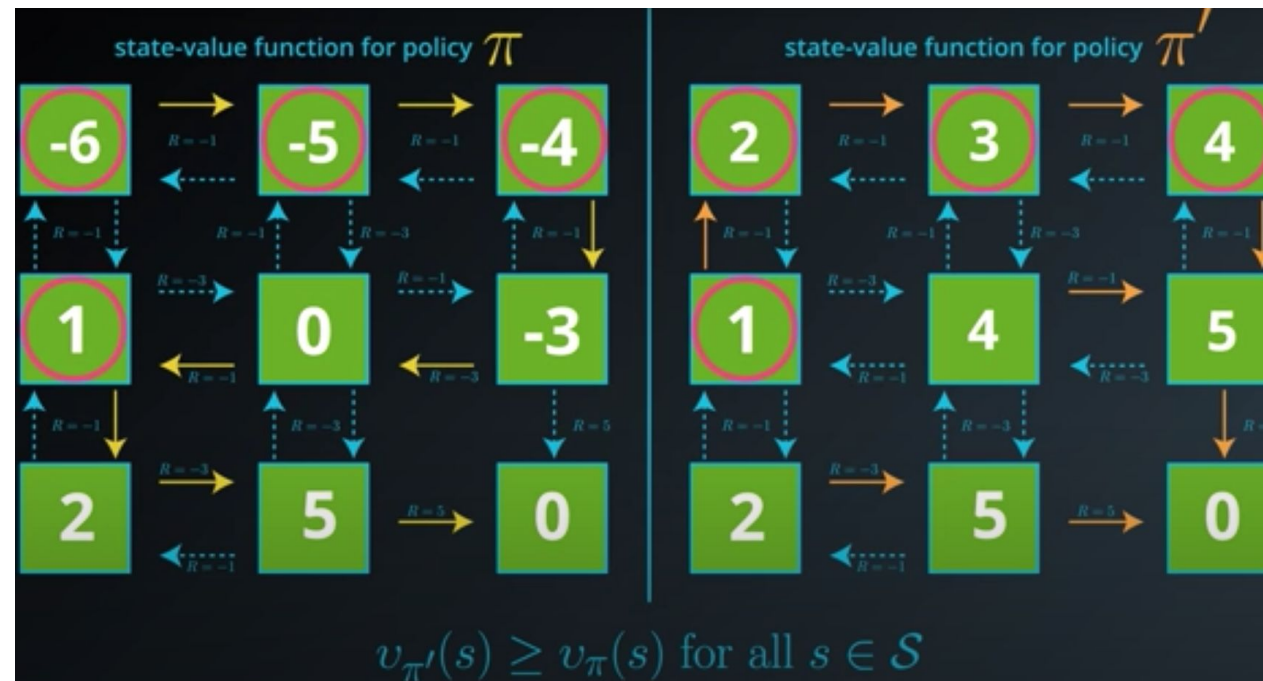


What's being optimal

A policy π is defined to be better than or equal to another policy π' if its expected return is greater than or equal to that of the other policy π for all states.

More formally,

$$\pi' \geq \pi \text{ if and only if } v_{\pi'}(s) \geq v_{\pi}(s) \text{ for all } s \in \mathcal{S}$$



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

What's being optimal

Note: It is often possible to find two policies that cannot be compared.

There may be more than one optimal policy, we denote the optimal policies by π^*
It is guaranteed to exist but may not be unique \Rightarrow It's the solution to the MDP.

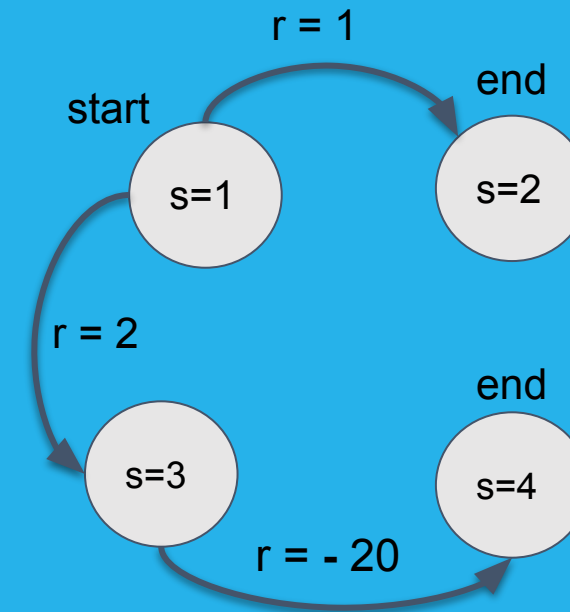
They share the same state-value function, denoted v^* :

$$v_*(s) = \max_{\pi} v_{\pi}(s), \quad \text{for all } s \in \mathcal{S}.$$

Ordering Policies: Quiz!

- Q1: Select the statements that are true (assume $\gamma=1$):

<input type="checkbox"/>	$\pi_1 \succeq \pi_2$ and $\pi_3 \succeq \pi_2$
<input type="checkbox"/>	$\pi_2 \succeq \pi_1$ and $\pi_2 \succeq \pi_3$
<input type="checkbox"/>	$\pi_1 \succeq \pi_3$ and $\pi_3 \succeq \pi_4$
<input type="checkbox"/>	$\pi_4 \succeq \pi_2$
<input type="checkbox"/>	$\pi_2 \succeq \pi_4$



π_1 : agent always goes right

π_2 : agent always goes down

π_3 : agent goes right with a probability of 0.5 and down with a probability of 0.5

π_4 : agent goes right in 10% of cases and in 90% of cases executes the "down" action

Action-Value functions

The action-value function of a state s and action a under a policy π , denoted by $q_{\pi}(s,a)$, is the expected return when starting in the state, taking the action and following the policy thereafter.

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

For each state s and action a

it yields the **expected return**

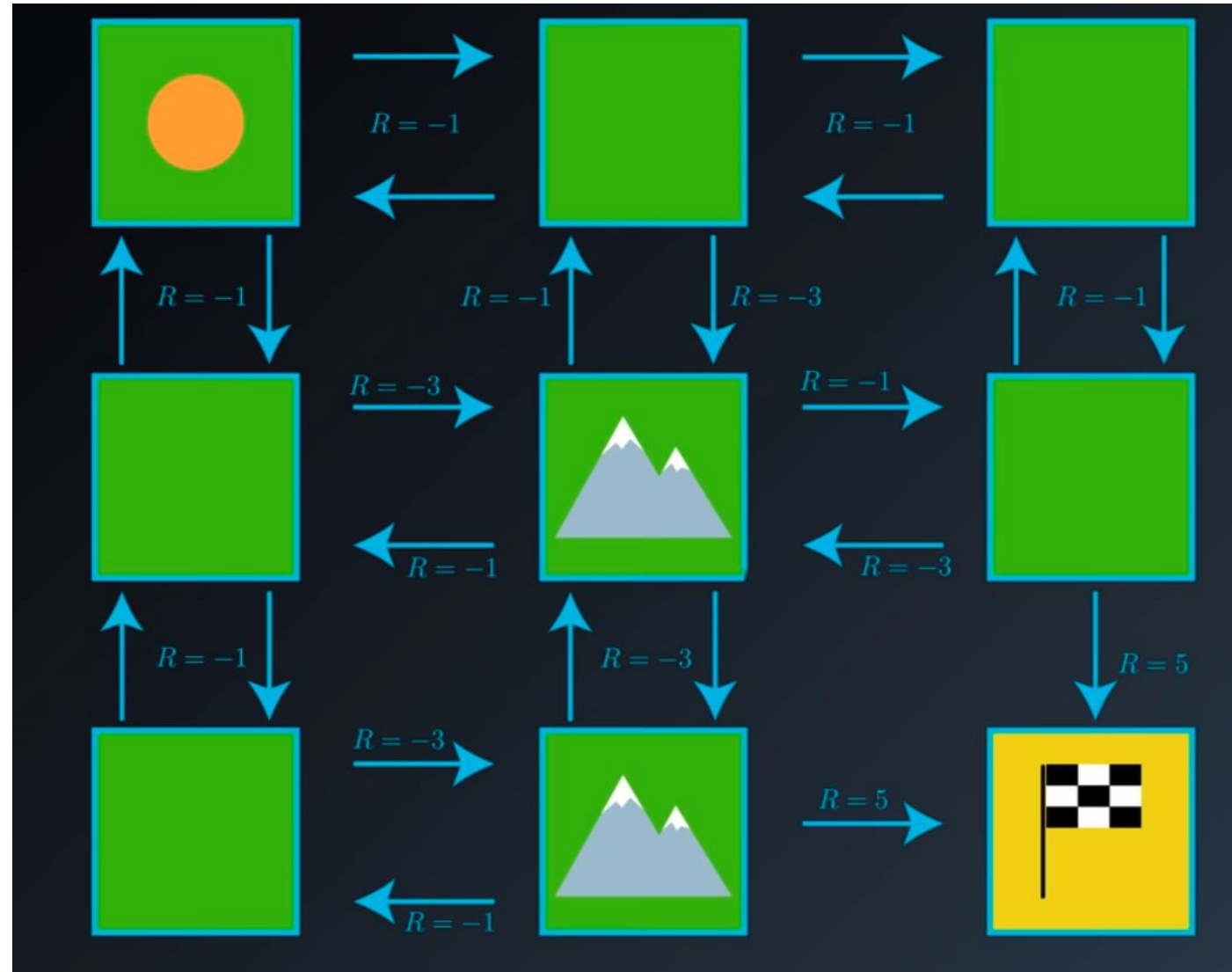
if the agent starts in state s

and takes action a

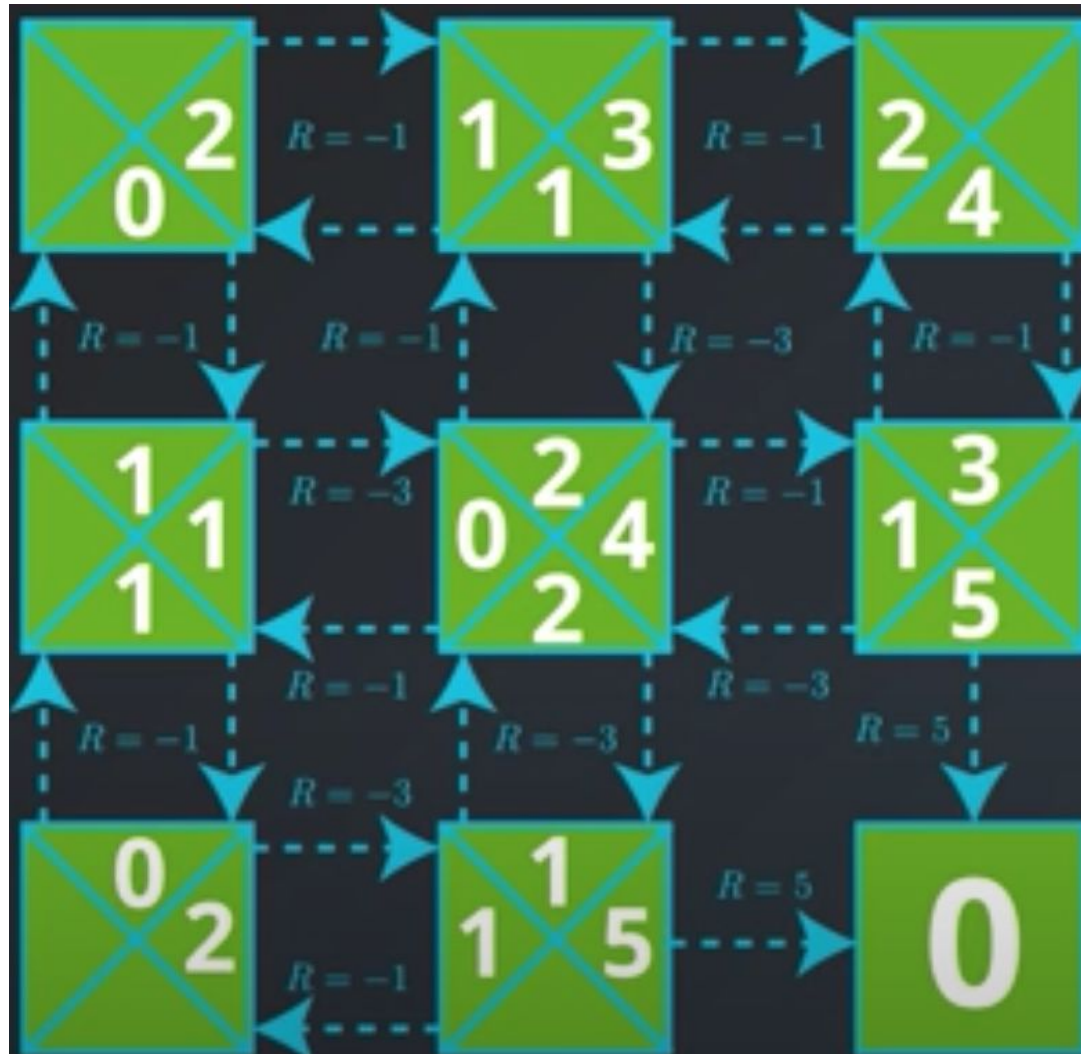
and then uses the **policy** to choose its actions for

all time steps

Action-Value functions



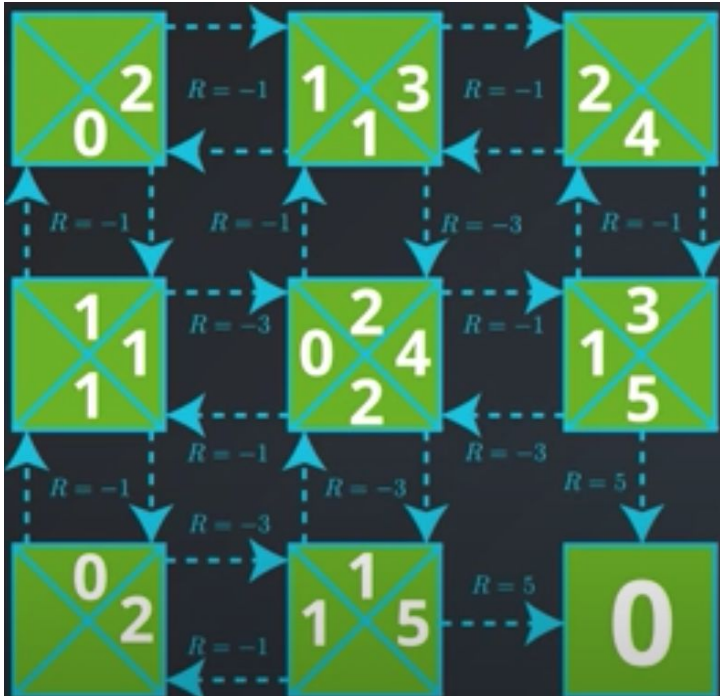
Action-Value functions



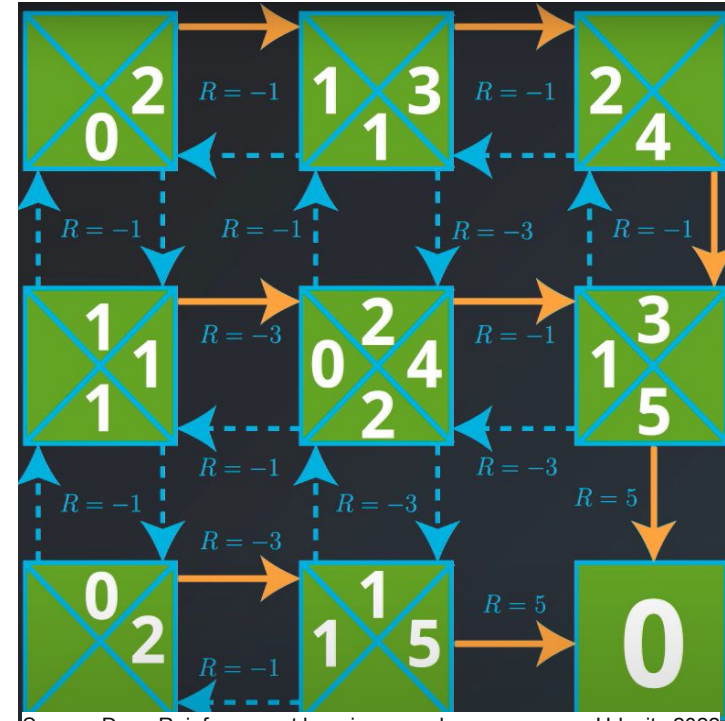
Source: Deep Reinforcement learning nanodegree program, Udacity 2022

What is being Optimal

By interacting with the environment the agent estimates the **optimal action-value** function



Source: Deep Reinforcement learning nanodegree program, Udacity 2022



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

We obtain q^*

π_*

From that estimation, it can quickly obtain an optimal policy π_* . For each state, we pick the action that yields the highest expected return.

What is being Optimal

Optimal policies also share the same optimal action-value function, denoted q_*

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



04





Taxonomy of RL algos



Taxonomy of RL algorithms

All the methods in RL can be classified into various aspects:

- **Model-free or model-based**
- **Value-based or policy-based**
- **On-policy or off-policy**



Model-free and Model-based RL



Model: A model predicts what the environment will do next. The term model refers to the transition function and the reward function

- **Transitions:** P predicts the next state (i.e., system dynamics)

$$P(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$$

- **Rewards:** R predicts the next immediate reward (e.g., $r_{t+1} = E(R_{t+1} \mid s_t = s, a_t = a)$)

Model-based: The agent either has the model or tries to build an explicit representation of the environment based on its interactions with the environment

Model-free: The agent does not build a model of the environment. Instead, the agent uses the interactions with the environment to find out a policy and/or value function.

Value-Based and Policy-Based learning

- **Policy-based**
 - policy
 - No value function
- **Value-based**
 - No policy (implicit)
 - A value function
- **Actor critic**
 - policy
 - value function

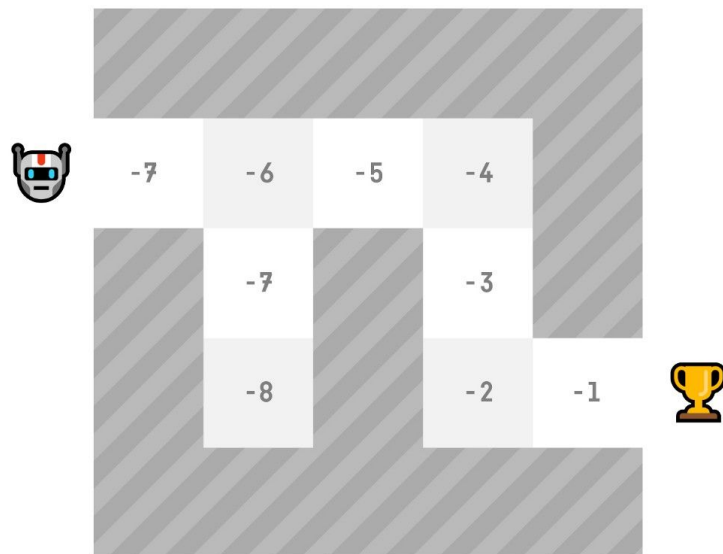


Value-Based and Policy-Based learning

$$v(s) / Q(s, a)$$

Value-Based Learning

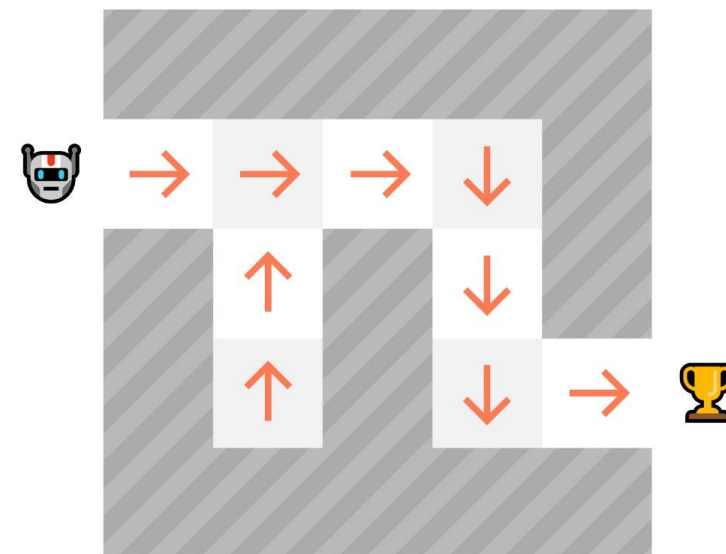
The agent optimize the value function, that it uses to select the action to take at each step, e.g. the action with the highest value estimate



$$\pi(a|s)$$

Policy-Based Learning

The agent optimizes its policy right away without passing through a value function. The agent takes the action with the highest probability.

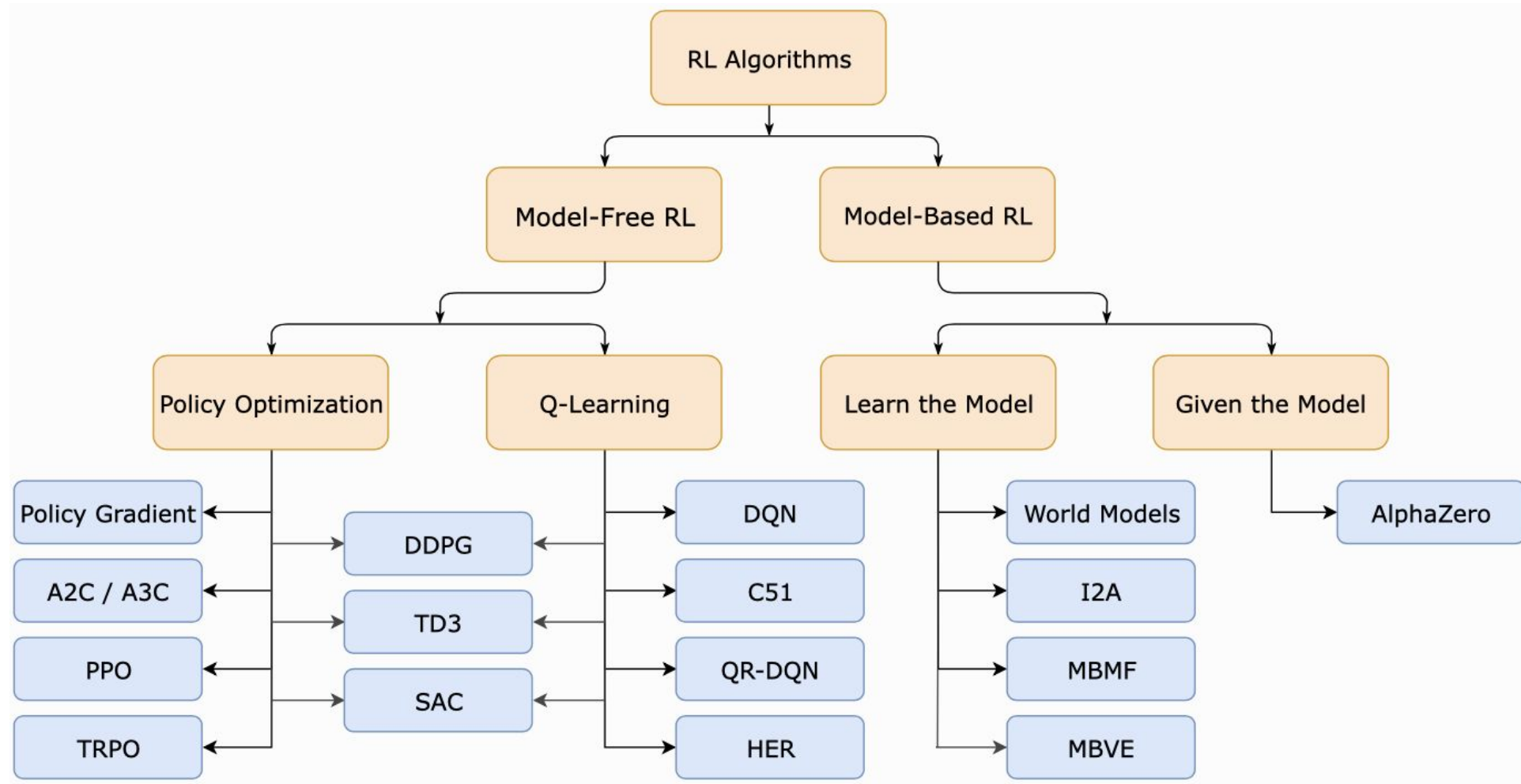


On-Policy and Off-Policy

This affects how training iterations make use of data.

- **On-Policy:** it learns on the policy—that is, training can only utilize data generated from the current policy π . This implies that as training iterates through versions of policies, $\pi_1, \pi_2, \pi_3, \dots$, each training iteration only uses the current policy at that time to generate training data. As a result, all the data must be discarded after training, since it becomes unusable.
- **Off-Policy:** Any data collected can be reused in training.

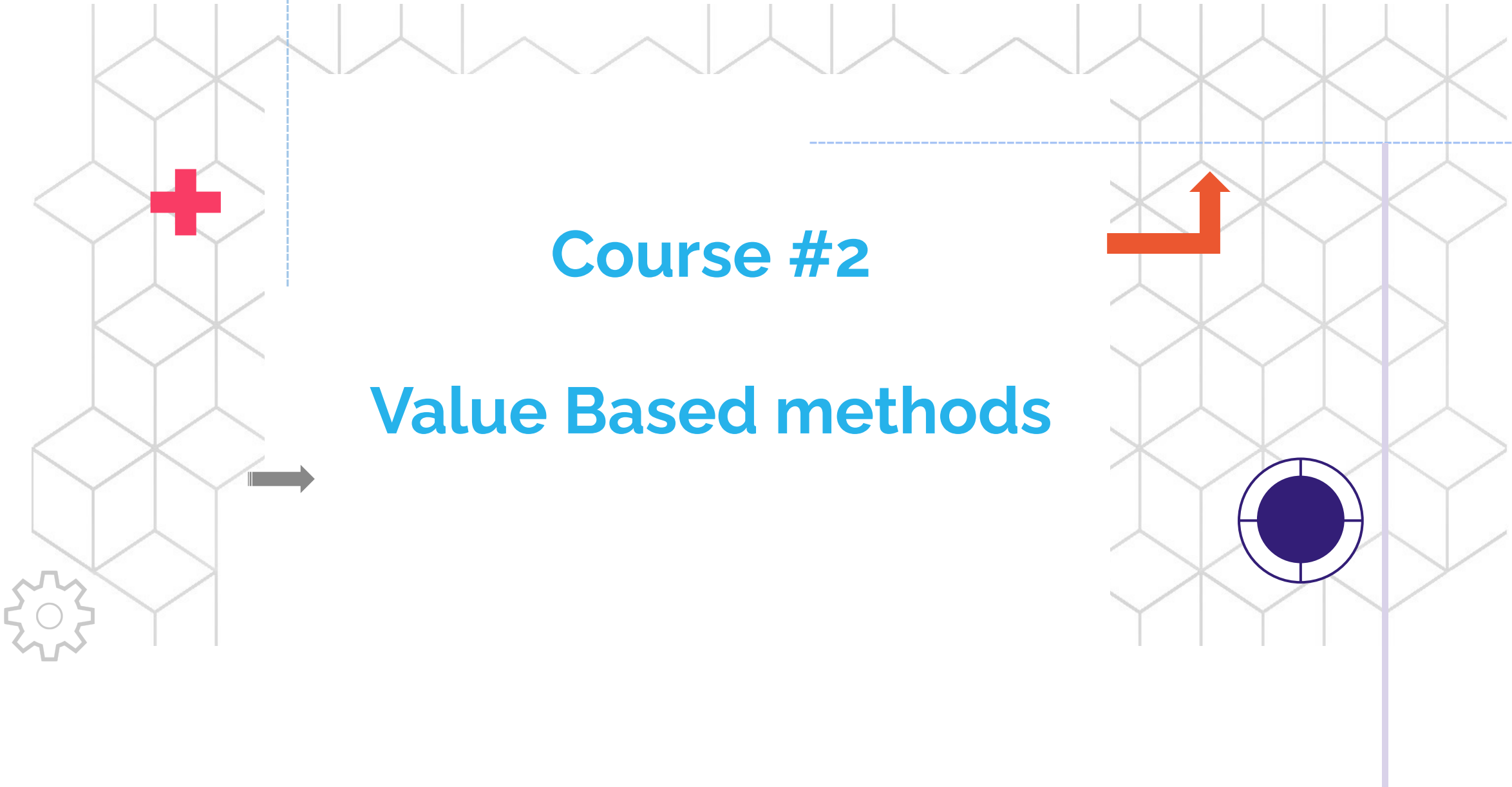
Taxonomy of RL algorithms





Course #2

Value Based methods



Value-Based learning

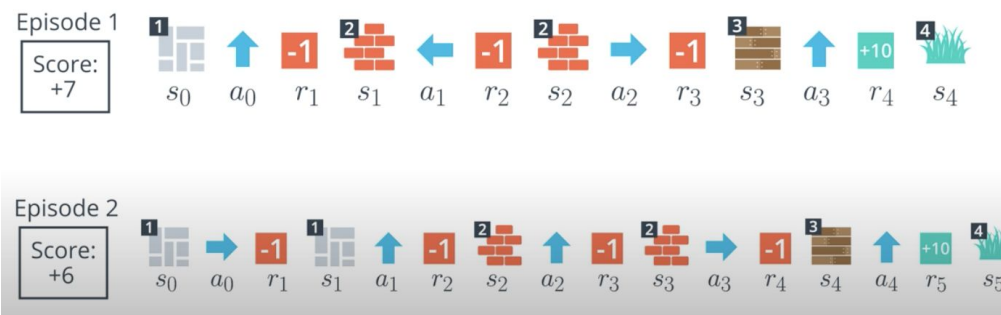
Recall that value-based methods: Train a value function to learn which state is more valuable and use this value function to take the action that leads to it.

How can the agent consolidate his experience to learn this value function?

- The agent interacts with the environment and collects trajectories. The accumulated experience can be used to learn the value function.

The agent needs more episodes to collect better informed decisions and truly understand the environment.


- The agent hasn't attempted each action from each state.
- The environment dynamics are stochastic.




Source: Deep Reinforcement learning nanodegree program, Udacity 2022



01





Tabular Q-learning

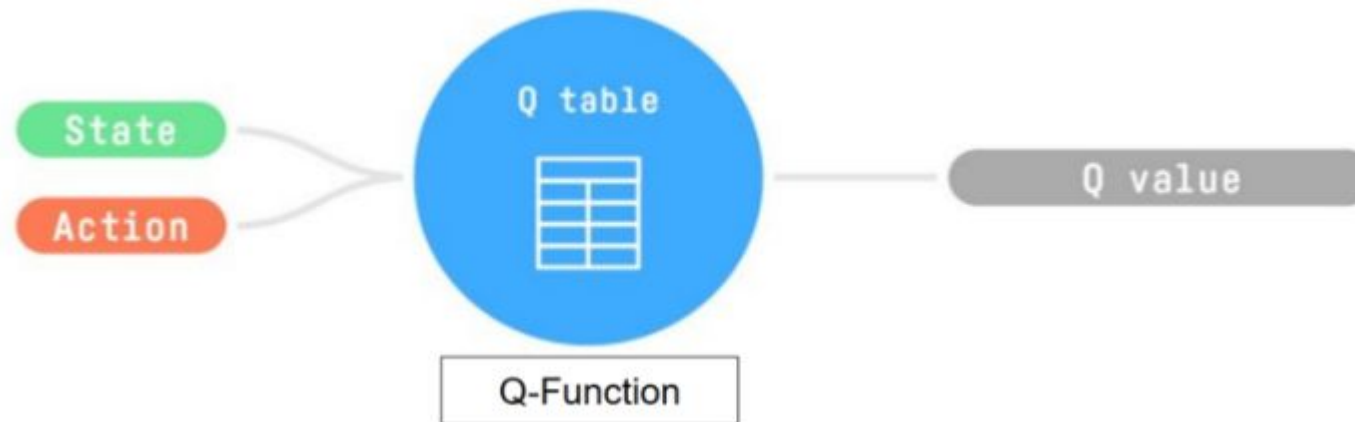


Q-learning: Big picture

Q-Learning is an off-policy value-based method that uses Bellman equation as a basis to train its action-value function.

Q-Learning is the algorithm we use to train our Q-function, an action-value function that determines the value of being at a particular state and taking a specific action at that state.

Q-function is encoded by a Q-table, a table where each cell corresponds to a state-action pair value



Q-learning: The link between values and policy

Question: How can we find the optimal policy once we have this Q-table (or the value of each state-action pair)?

Note that whenever the value of each state-action pair is known, the optimal action can be determined from the following equation:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

In other words, if we have an optimal Q-function, we have an optimal policy since we know the best action to take at each state.

Q-learning: estimating Q values

Question: How can we fill in this Q-table so that it includes the values of each state-action pair?

We want to find the Q values without the need for prior knowledge of environment dynamics (model-free RL), and, instead, leverage our interactions with the environment for determining the value of each state-action pair.

If, for every action, the reward and the next state can be observed, one trick is to use the following update rule which is based on Bellman equation:

$$Q(S_t, A_t) \leftarrow \underbrace{R_{t+1}}_{\substack{\text{Immediate} \\ \text{Reward}}} + \underbrace{\gamma \max_a Q(S_{t+1}, a)}_{\substack{\text{Discounted Estimate} \\ \text{optimal Q-value} \\ \text{of next state}}}$$

For each interaction with the environment, we update the Q value based on the equation above. Then, after a large number of interactions with the environment, the action values Q will converge to the true Q values.

Q-learning: The Exploration-exploitation tradeoff

Before giving an example on how to use Q-learning, note that In the beginning, our Q-table is useless since it gives arbitrary values for each state-action pair (most of the time, we initialize the Q-table to 0). The agent must therefore continually explore its environment by attempting different states and actions and observing its obtained reward.

By only exploring his environment, the agent risks to end in the bad state. At some point, the agent needs to start taking advantage of what it has learned so far. As the agent explores the environment and we update the Q-table, it will give us a better and better approximation to the optimal policy. This is one of challenges of RL: finding the right balance between exploration and exploitation.

Q-learning: The Exploration-exploitation tradeoff

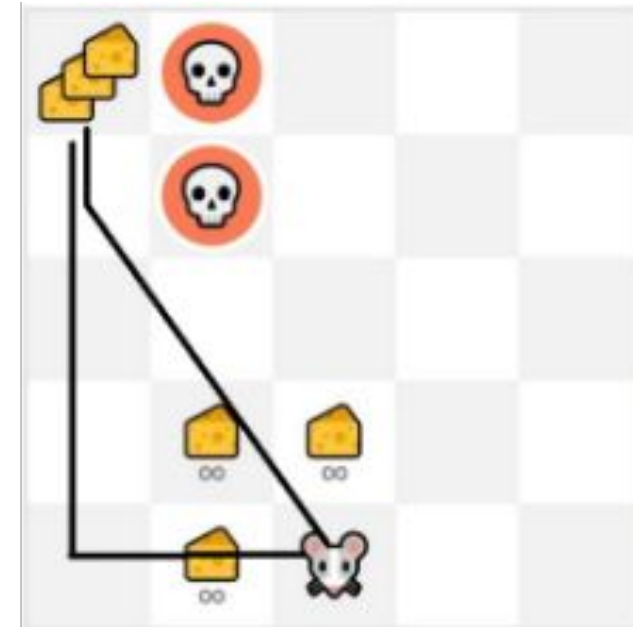
Exploitation

use known information to
maximize reward



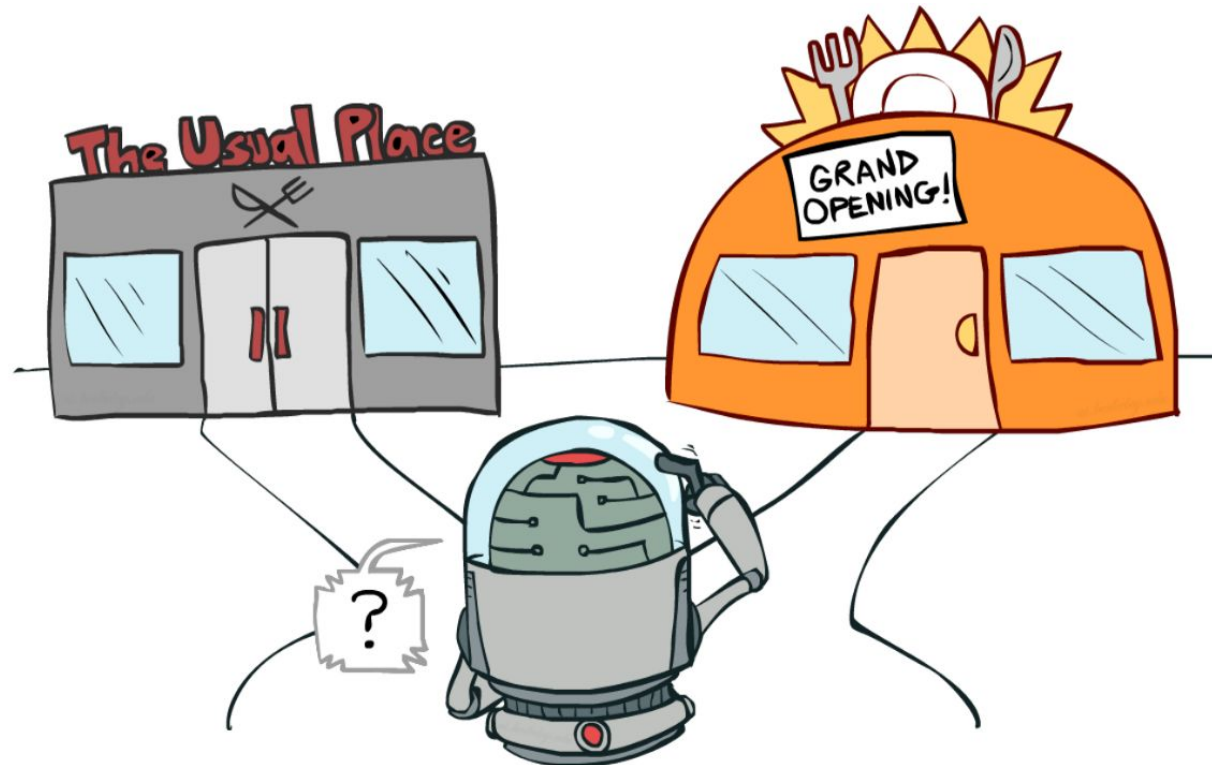
Exploration

Try different random actions in
order to discover more
information about the
environment



Q-learning: The Exploration-exploitation tradeoff

At each step, the agent has two choices. He should either act **greedily** (take the best action based on the known information), or to try to find more information about the environment in order to improve his knowledge and to discover a way to obtain better rewards in the future.

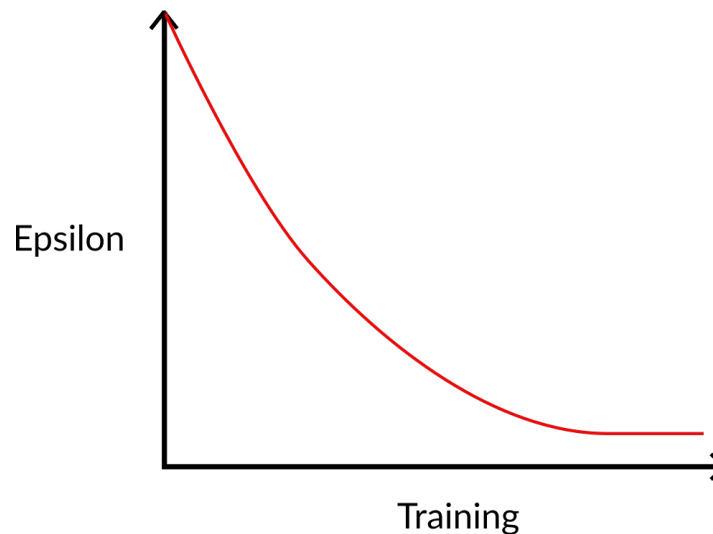


Epsilon greedy policy

Exploration can be achieved using an epsilon-greedy policy. This policy consists of choosing the action among the possible ones with the highest Q value with $1-\epsilon$ probability, or to explore the environment by choosing it randomly with ϵ .

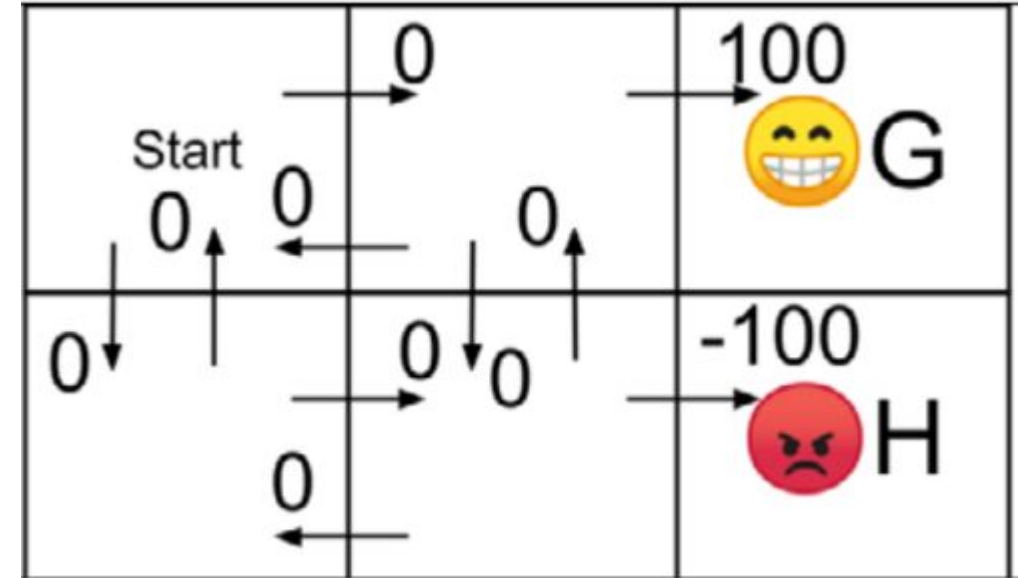


→ It's common to start with a high ϵ and to reduce its value as your policy goes through more iterations.



Q-learning Example

- The reward is non-zero in two cases:
Transition to the Goal (G) state has a +100 reward, while moving into the Hole (H) state has a -100 reward. These two states are terminal states and constitute the end of one episode from
- The agent assumes a policy that selects a random action 90% of the time and exploits the Q-table 10% of the time.
- $\gamma=0.9$



Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.

Q-learning Example

We initialize the Q-table

Since the agent has not learned anything yet about its environment, the Q-table has zero initial values.

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.

(0,0)	(0,1)	 G (0,2)
(1,0)	(1,1)	 H (1,2)

Environment

State \ Action	Action			
	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0

Q Table

Q-learning Example

EPISODE#1:

Suppose: **action#1** is randomly selected and indicates a move to the right

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.

Start (0, 0)	0 (0, 1)	😊 G (0, 2)
		😡 H (1, 2)
(1, 0)	(1, 1)	

Mode: Exploration

$$Q((0,0), \rightarrow) = \text{reward} + \gamma * \max_{a'} Q((0,1), a')$$

$$Q((0,0), \rightarrow) = 0 + 0.9 * \max(0, 0, 0, 0) = 0$$

Action \ State	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0

Q Table

Q-learning Example

EPISODE#1:

Suppose: **action#2** is randomly selected and indicates a move in **downward direction**

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.



Mode: Exploration

$$Q((0,1), \downarrow) = \text{reward} + \gamma * \max_a Q((1,1), a')$$

$$Q((0,1), \downarrow) = 0 + 0.9 * \max(0, 0, 0, 0) = 0$$

Action \ State	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0

Q Table

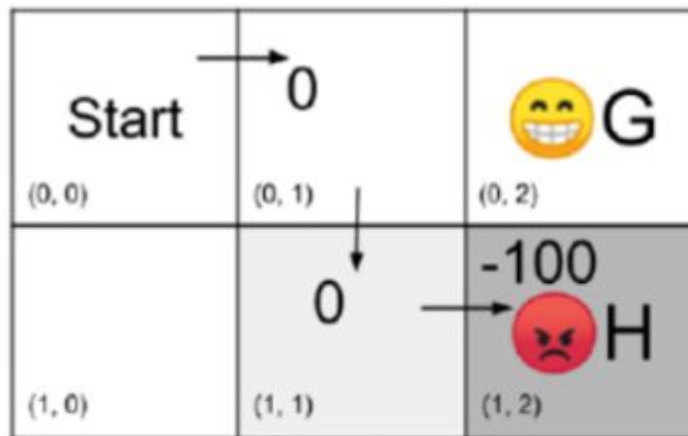
Q-learning Example

EPISODE#1:

Suppose: **action#3** is randomly selected and indicates a **move to the right**

It encountered the **H state** and received a **-100 reward**. The episode has just finished, and the agent returns to the Start state.

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.



Mode: Exploration

$$Q((1,1), \rightarrow) = \text{reward} = -100$$

Action \ State	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

Q-learning Example

EPISODE#2:

Suppose: The random actions chosen by the agent are two successive moves to the right

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras., Packt Publishing Ltd.

Start (0, 0)	0 (0, 1)	100 😊 G (0, 2)
(1, 0)	(1, 1)	(1, 2)

Mode: Exploration

$$Q((0,1),\rightarrow) = \text{reward} = 100$$

Action \ State	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

Q-learning Example

EPISODE#3:

The first random action taken by the agent is a move to the right.

The Q value of state (0, 0) is now updated with a non-zero value. It is like giving credit to the earlier states that helped in finding the **G** state.

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras., Packt Publishing Ltd.



Mode: Exploration

$$Q((0,0), \rightarrow) = \text{reward} + \gamma * \max_{a'} Q((0,1), a')$$

$$Q((0,0), \rightarrow) = 0 + 0.9 * \max(0, 0, 0, 100) = 90$$

Action \ State	←	↓	→	↑
(0,0)	0	0	90	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table



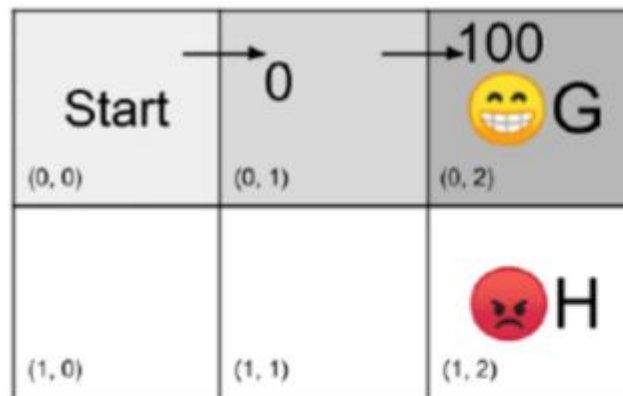
Q-learning Example

EPISODE#4:

For this episode, suppose the agent decides to **exploit** the Q-table instead of randomly exploring the environment.

The Q-table suggests moving to the right for both states, which allows reaching the **G state**.

Source: Gulli, A., Kapoor, A., & Pal, S. (2019). Deep learning with TensorFlow 2 and Keras,. Packt Publishing Ltd.



Mode: Exploitation

$$\max_a Q((0,0),a') = 90 \rightarrow \text{with } s'=(0,1)$$

$$\max_a Q((0,1),a') = 100 \rightarrow \text{with } s'=(0,2) \text{ or Goal}$$

Action \ State	←	↓	→	↑
(0,0)	0	0	90	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

Q-learning Example

If the Q-learning algorithm continues to run indefinitely, the Q-table will converge.

```
Q-Table (Epsilon: 0.25)
[[ 0.      72.9    90.      81.   ]
 [ 0.      81.    100.     90.   ]
 [ 0.       0.      0.      0.    ]
 [ 0.      72.9    81.      81.   ]
 [ 72.9    65.61 -100.     90.   ]
 [ 0.       0.      0.      0.    ]]
```

Tabular Q-learning algorithm

- 1- Start with an empty table $Q(s,a)$
- 2- By interacting with the environment, obtain the tuple (s, a, r, s') .
- 3- Update the $Q(s,a)$ table using the Bellman approximation:

$$Q(s, a) \leftarrow r + \gamma \max_{a' \in A} Q(s', a')$$

- 4- Repeat from step 2 until convergence

Note: For step 2, there is no single way for selecting the action. As such, the exploration-exploitation trade-off should be taken into account when selecting actions..

Q-learning: The “blended” update rule

We have been using the following Bellman approximation for updating the Q-table.

$$Q(s, a) \leftarrow r + \gamma \max_{a' \in A} Q(s', a')$$

As we take samples from the environment, it's generally a bad idea to assign new values on top of existing values, as training can become unstable. What is usually done in practice is updating the Q-table using a “**blending**” technique, which is simply averaging between old and new values of Q using a learning rate $\alpha \in [0, 1]$

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') \right)$$

This allows values of Q to converge **smoothly**, even if our environment is noisy.

Tabular Q-learning: Final version

- 1- Start with an empty table $Q(s,a)$
- 2- By interacting with the environment, obtain the tuple (s, a, r, s') .
- 3- Update the $Q(s,a)$ table using the “blended” Bellman approximation:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') \right)$$

- 4- Repeat from step 2 until convergence

Tabular Q-learning: pseudocode

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π , positive integer $num_episodes$, small positive fraction α , GLIE $\{\epsilon_i\}$

Output: value function Q ($\approx q_\pi$ if $num_episodes$ is large enough)

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

for $i \leftarrow 1$ **to** $num_episodes$ **do**

$\epsilon \leftarrow \epsilon_i$

 Observe S_0

$t \leftarrow 0$

repeat

 Choose action A_t using policy derived from Q (e.g., ϵ -greedy)

 Take action A_t and observe R_{t+1}, S_{t+1}

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

$t \leftarrow t + 1$

until S_t is terminal;

end

return Q

Source: Reinforcement Learning 101 by Sriramanth Tenneti, 2020, Analytics Vidhya



Tabular Q-learning: on-policy or off-policy?

Recall:

- **On-policy:** using the same policy for acting and updating.
- **Off-policy:** using a different policy for acting (inference) and updating (training).

In Q-learning: We are using the epsilon-greedy policy for acting (acting policy):

Choose action A_t using policy derived from Q (e.g., ϵ -greedy)

This acting policy is different from the greedy policy that is used to select the best next-state action value to update our Q-value (updating policy).

$$\gamma \max_a Q(S_{t+1}, a)$$

This is why we say that Q Learning is an off-policy algorithm.

On-policy value-based learning: Sarsa

With Sarsa, another value-based algorithm, the epsilon-greedy policy selects the next state-action pair, not a greedy policy.

Algorithm 13: Sarsa

Input: policy π , positive integer $num_episodes$, small positive fraction α , GLIE $\{\epsilon_i\}$

Output: value function Q ($\approx q_\pi$ if $num_episodes$ is large enough)

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

for $i \leftarrow 1$ **to** $num_episodes$ **do**

$\epsilon \leftarrow \epsilon_i$

 Observe S_0

 Choose action A_0 using policy derived from Q (e.g., ϵ -greedy)

$t \leftarrow 0$

repeat

 Take action A_t and observe R_{t+1}, S_{t+1}

 Choose action A_{t+1} using policy derived from Q (e.g., ϵ -greedy)

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

$t \leftarrow t + 1$

until S_t is terminal;

end

return Q

Source: Temporal Difference Methods by Xray, 2020, zhuanlan.zhihu

On-policy vs Off-policy value-based learning: Recap



- **Off-policy:** using a different policy for acting (inference) and updating (training).

Choose action A_t using policy derived from Q (e.g., ϵ -greedy) Epsilon Greedy Policy
Take action A_t and observe R_{t+1}, S_{t+1}
 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
Greedy Policy

Source: Hugging Face Deep RL Course 2018

- **On-policy:** using the same policy for acting and updating.

Choose action A_0 using policy derived from Q (e.g., ϵ -greedy) Epsilon Greedy Policy
 $t \leftarrow 0$
repeat
 Take action A_t and observe R_{t+1}, S_{t+1}
 Choose action A_{t+1} using policy derived from Q (e.g., ϵ -greedy)
 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$



Tabular Q-learning: Quiz!

- Q1: Consider an agent that has a state made of two discrete variables. The first variable is in the set $\{0, 1, 2, 3\}$ while the second variable is binary. If the agent has 4 possible actions, how many states does the agent have?

	6
	8
	10
	12

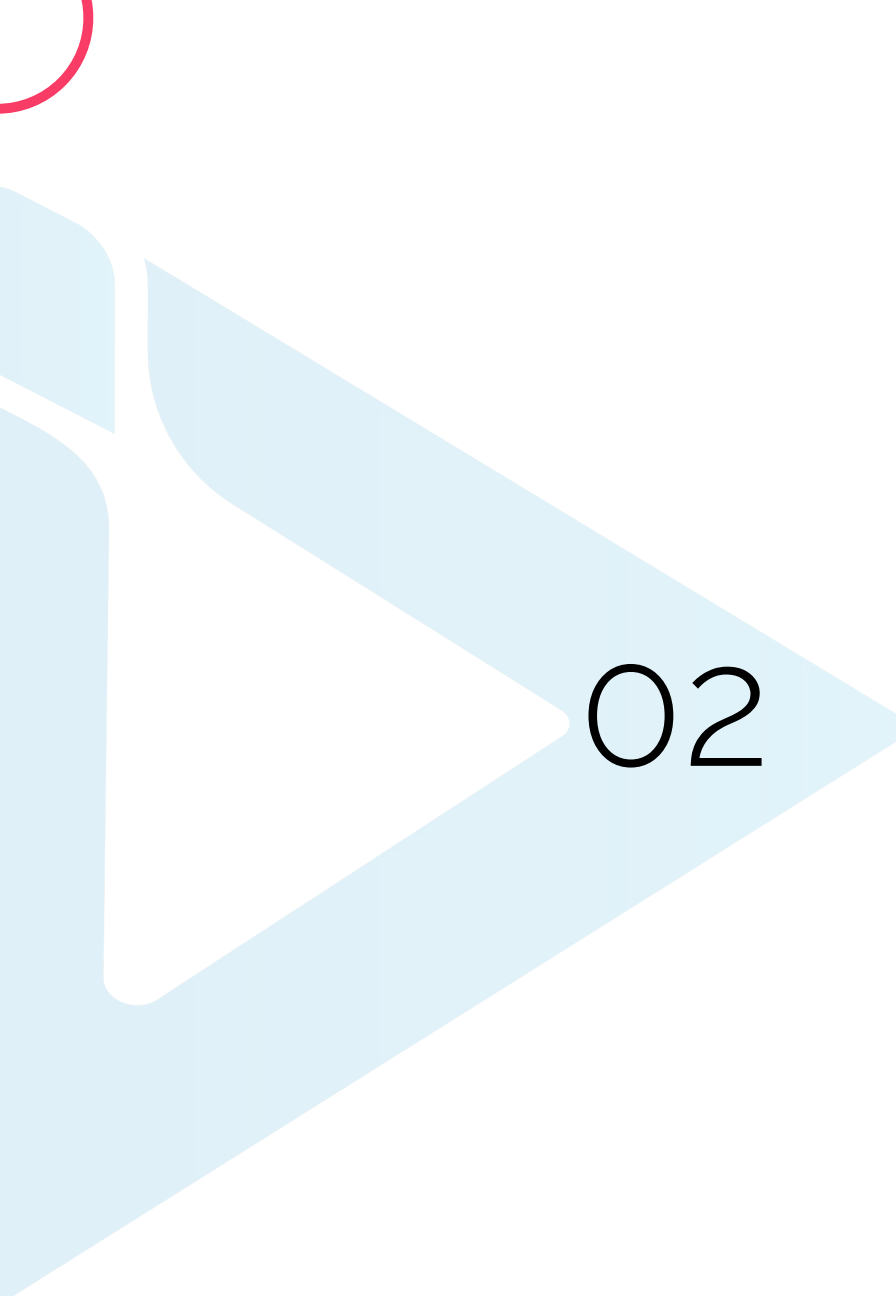


Tabular Q-learning: Quiz!

- Q2: Consider an agent that has a state made of two discrete variables. The first variable is in the set $\{0, 1, 2, 3\}$ while the second variable is binary. If the agent has 4 possible actions, what is the dimension of the q table (rows, columns)?

	(4, 6)
	(4, 8)
	(6, 4)
	(8, 4)





02



Deep Q-learning




From Tabular Q-learning → to Deep Q-learning

Using the Q-table to estimate the Q-values is fine for small discrete environments.

However, when the environment has numerous states or is continuous, as in most cases, a Q-table is impractical and not feasible.

Example:

- Consider a state made of four continuous variables:
 - $\text{speed} \in [0,1]$, $\text{angle1} \in [0,1]$, $\text{angle2} \in [0,1]$, $\text{acceleration} \in [0,1]$
- How many states?
 - discretization step of 0.01 → 100 for each state
 - Number of states: $100^4 = 10000000$ states !!



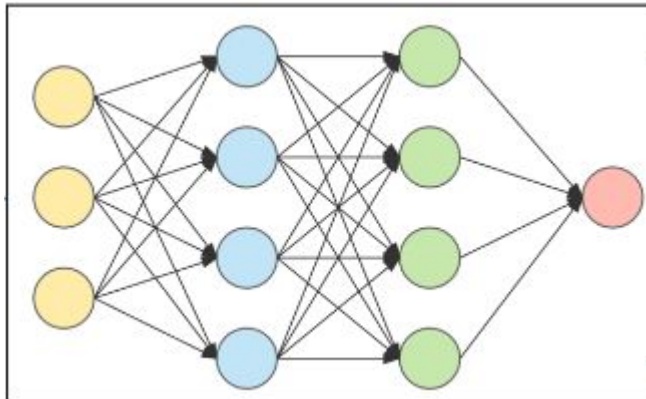
	→	←	↑	↓
Start	0	1	0	0
Idle	2	0	0	3
Hole	0	2	0	0
End	1	0	0	0

Source: An Introduction to Q-Learning,
2022, Datacamp

From Tabular Q-learning → to Deep Q-learning

How overcome the Q-table burden?

- Instead of a table, use any function that maps the state and action onto the Q value
- Most popular solution is to use a deep neural network as a function approximator to approximate the Q-table: This is known as Deep Q-learning (see the [DQN paper](#))



Source: A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python, 2019, Analytics Vidhya

Deep RL = RL + Neural Network

Did you know:

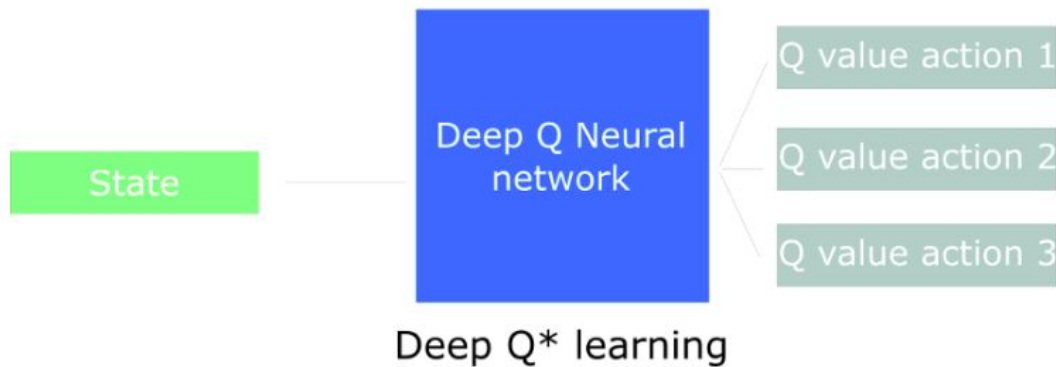
In 2013, DeepMind published a paper entitled "Playing Atari with Deep Reinforcement Learning" (DQN paper) that outlined their new approach to an old algorithm, which gave them enough performance to play six of seven Atari 2600 games at record levels!

Deep Q-learning

Idea of DQN is to use a neural network with parameters θ , to estimate the Q-values :

$$Q(s,a; \theta) \approx Q^*(s,a)$$

Inspired by supervised learning, the deep Q neural network learns to provide reliable estimates of the Q-values based on the interactions with the environment. The learned Q-function is then used by an agent to select actions.



- The input is the state
- The prediction is the Q value for each action
- Desirable action: action with the largest Q value

Source: An introduction to Deep Q-Learning by Thomas Simonini, 2018, freeCodeCamp

Note that DQN is only applicable to environments with discrete action spaces.

Standard Deep Q-learning

The Q-learning algorithm is then modified as follows:

1. Initialize $Q(s, a)$ with some initial approximation.
2. By interacting with the environment, obtain the tuple (s, a, r, s') .
3. Calculate loss: $\mathcal{L} = (Q(s, a) - r)^2$ if the episode has ended, or
 $\mathcal{L} = \left(Q(s, a) - \left(r + \gamma \max_{a' \in A} Q_{s', a'} \right) \right)^2$ otherwise.
4. Update $Q(s, a)$ using the **stochastic gradient descent (SGD)** algorithm, by minimizing the loss with respect to the model parameters.
5. Repeat from step 2 until converged.

Zoom on the loss function:

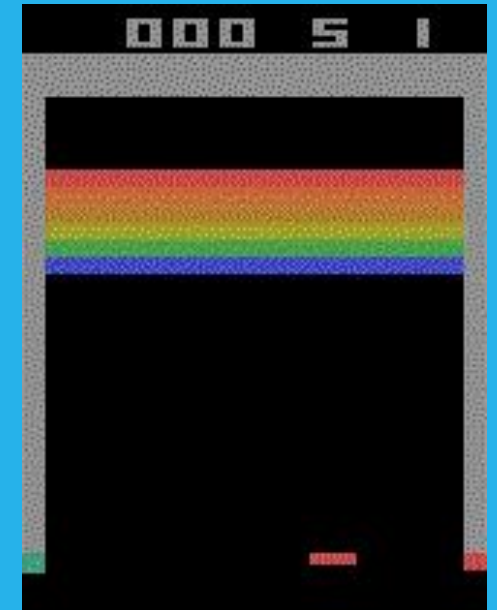
$$\mathcal{L} = \left(Q(s, a) - \underbrace{\left(r + \gamma \max_{a' \in A} Q_{s', a'} \right)}_{\substack{\text{TD error:} \\ \text{difference between target and prediction}}} \right)^2$$

y:
This term is usually denoted y and called
TD target (or simply target)

Tabular vs Deep Q-learning: Quiz!

- Q1: you want to build an RL agent that plays the breakout Atari game? Knowing that the agent receives raw pixel data as input (210, 160, 3). The action can take only one of the following actions: NOOP, FIRE, RIGHT, LEFT. Choosing between tabular and deep Q-learning, which of the following statements seem reasonable?

<input type="checkbox"/>	The action space is discrete so tabular Q learning seems fine
<input type="checkbox"/>	The action space is small so deep Q learning seems fine
<input type="checkbox"/>	The state space is small so tabular Q learning seems fine
<input type="checkbox"/>	The state space is large so deep Q learning seems fine

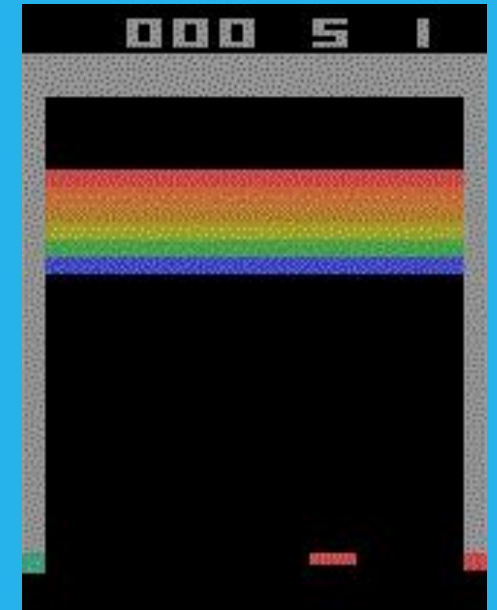


Source: Deep Q-Learning for Atari Breakout by Jacob Chapman and Mathias Lechner , 2020, Keras

Tabular vs Deep Q-learning: Quiz!

- Q1: you want to build an RL agent that plays the breakout Atari game? Knowing that the agent receives raw pixel data as input (210, 160, 3). The action can take only one of the following actions: NOOP, FIRE, RIGHT, LEFT. Choosing between tabular and deep Q-learning, which of the following statements seem reasonable?

<input type="checkbox"/>	The action space is discrete so tabular Q learning seems fine
<input type="checkbox"/>	The action space is small so deep Q learning seems fine
<input type="checkbox"/>	The state space is small so tabular Q learning seems fine
<input checked="" type="checkbox"/>	The state space is large so deep Q learning seems fine



Source: Deep Q-Learning for Atari Breakout by Jacob Chapman and Mathias Lechner, 2020, Keras

Standard Deep Q-learning: What could go wrong?

- At each time step, we learn from a tuple (s, a, r, s') and then throw this experience \Rightarrow our neural network tends to forget the previous experiences as it overwrites with new experiences \rightarrow **Risk of forgetting previous experiences and no data efficiency**
- It is more efficient to make use of previous experience, by learning with it multiple times.



Source: An introduction to Deep Q-Learning by Thomas Simonini, 2018, freeCodeCamp

Standard Deep Q-learning: What could go wrong?

- Another limitation is that the data used for SGD update are highly correlated: these data samples are very close to each other, as they belong to the same episode (we know that future states and rewards depend on previous states and actions)→**Need to reduce correlation between experiences**

Episode 1

s1,a1,r2	s2,a2,r3	s3,a3,r4	s4,a4,r5	s5,a5,r6	s6,a6,r7	s7,a7,r8
----------	----------	----------	----------	----------	----------	----------

Episode 2

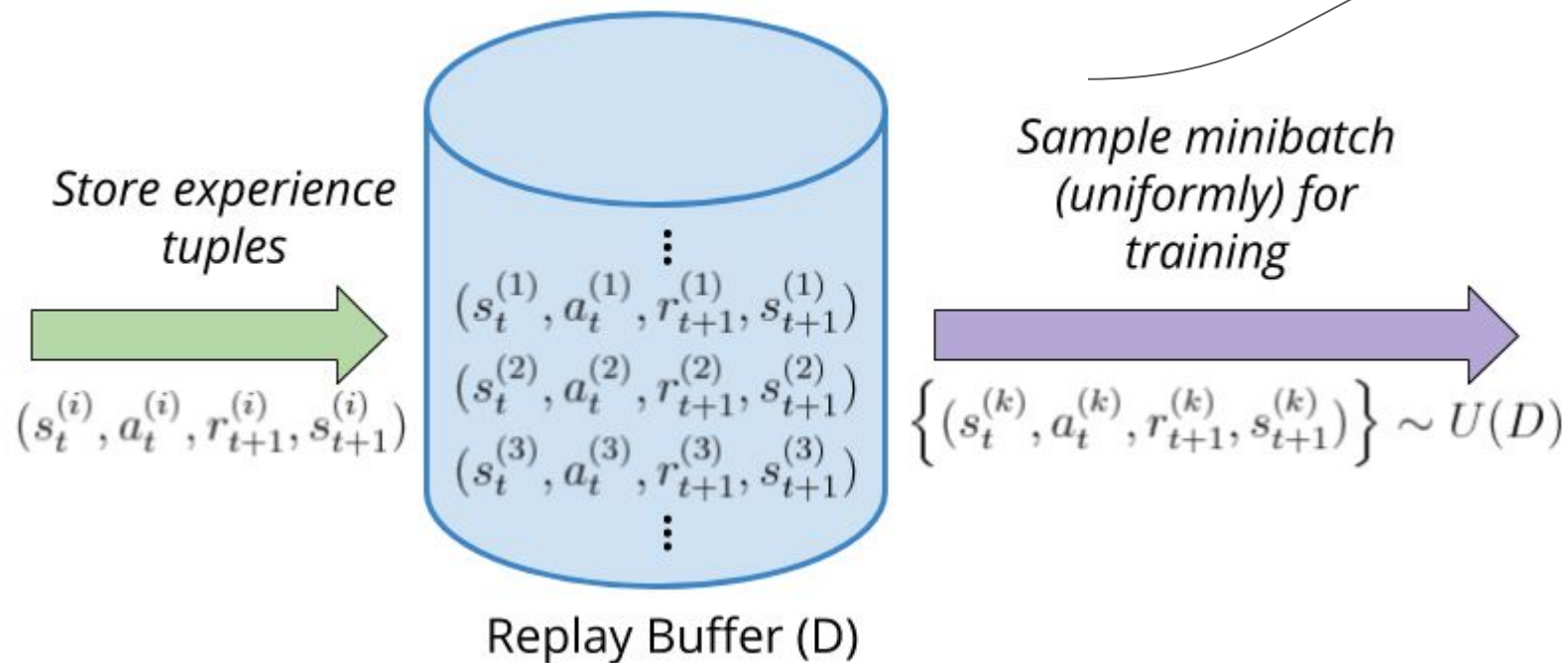
s1,a1,r2	s2,a2,r3	s3,a3,r4
----------	----------	----------

Episode 3

s1,a1,r2	s2,a2,r3	s3,a3,r4	s4,a4,r5
----------	----------	----------	----------

Deep Q-learning with Experience Replay

- One solution to overcome the data inefficiency and the highly correlated data samples: **Experience Replay buffer**



The minibatch contains experiences from different episodes and different policies. This has two advantages:

- break the correlations between samples
- data efficiency by using each transition in many updates

- If memory is full, the oldest experience is discarded to make space for the latest one.

Deep Q-learning with Experience Replay



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N
Initialize action-value function Q with random weights
for episode = 1, M **do**
 Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
 for $t = 1, T$ **do**
 With probability ϵ select a random action a_t
 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
 Execute action a_t in emulator and observe reward r_t and image x_{t+1}
 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}
 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}
 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
 end for
end for

Source: Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Deep Q-learning: The Instability issue

Recall that the parameters θ of the Q network are updated by performing gradient descent in the direction that minimizes the loss wrt model parameters θ .

$$L(\theta) = (Q(s,a;\theta) - y)^2 \text{ with } y = r + \gamma \max_{a'} Q(s',a';\theta)$$

The TD target is estimated with the same neural network with parameters θ , whose parameters are being updated: **Instability**



Source: presentermedia.com, 2023

we are getting closer to our target but also moving our target! It's like chasing our own tail!



Deep Q-learning: Improving Stability

To alleviate the instability, one trick is to use a snapshot of the network parameters from a few iterations ago instead of the last iteration for generating the target. This copy is called the **target network** (symbolically denoted with a hat).

The update rule for the network weights are modified as follows:

Before (Unstable): $\theta := \theta + \alpha \overbrace{(r + \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))}^{\text{computed with } \theta} \nabla_{\theta} Q(s, a; \theta)$

After: $\theta := \theta + \alpha \underbrace{(r + \max_{a'} \hat{Q}(s', a'; \theta^-))}_{\text{Target-network}} - Q(s, a; \theta) \nabla_{\theta} Q(s, a; \theta)$

computed with θ^-

The weights of the target network are updated after every T steps:

$$\theta^- := \theta$$

Deep Q-learning: Connecting the dots!



Putting it all together:

- Initialize replay memory with fixed capacity
 - Initialize action-value function q with random weights w
 - Initialize target action-value weights w^-
 - For number of episodes:
 - Observe state S
 - Choose action A_t from state S_t using policy π ϵ -greedy($q(S,A,w)$)
 - Take action A_t , observe reward R_{t+1} and next state S_{t+1}
 - Store experience tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory
- Obtain random minibatch of tuples $(s_j, a_j, r_{j+1}, s_{j+1})$ from replay memory
- Set target $y_j = r_j + \gamma \max_a Q(s_{j+1}, a, w^-)$
- Update: $\Delta w = \alpha (y_j - Q(s_j, a_j, w)) \nabla_w Q(s_j, a_j, w)$
- Every C steps, reset $w^- = w$

Deep Q-learning: Connecting the dots!



```
Initialize network  $Q$ 
Initialize target network  $\hat{Q}$ 
Initialize experience replay memory  $D$ 
Initialize the Agent to interact with the Environment
while not converged do
    /* Sample phase
     $\epsilon \leftarrow$  setting new epsilon with  $\epsilon$ -decay
    Choose an action  $a$  from state  $s$  using policy  $\epsilon$ -greedy( $Q$ )
    Agent takes action  $a$ , observe reward  $r$ , and next state  $s'$ 
    Store transition  $(s, a, r, s', done)$  in the experience replay memory  $D$ 

    if enough experiences in  $D$  then
        /* Learn phase
        Sample a random minibatch of  $N$  transitions from  $D$ 
        for every transition  $(s_i, a_i, r_i, s'_i, done_i)$  in minibatch do
            if  $done_i$  then
                |  $y_i = r_i$ 
            else
                |  $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$ 
            end
        end
        Calculate the loss  $\mathcal{L} = 1/N \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$ 
        Update  $Q$  using the SGD algorithm by minimizing the loss  $\mathcal{L}$ 
        Every  $C$  steps, copy weights from  $Q$  to  $\hat{Q}$ 
    end
end
```

Source: Deep Q-network (DQN)-II, by Jordi Torres, 2020, Towards Data Science

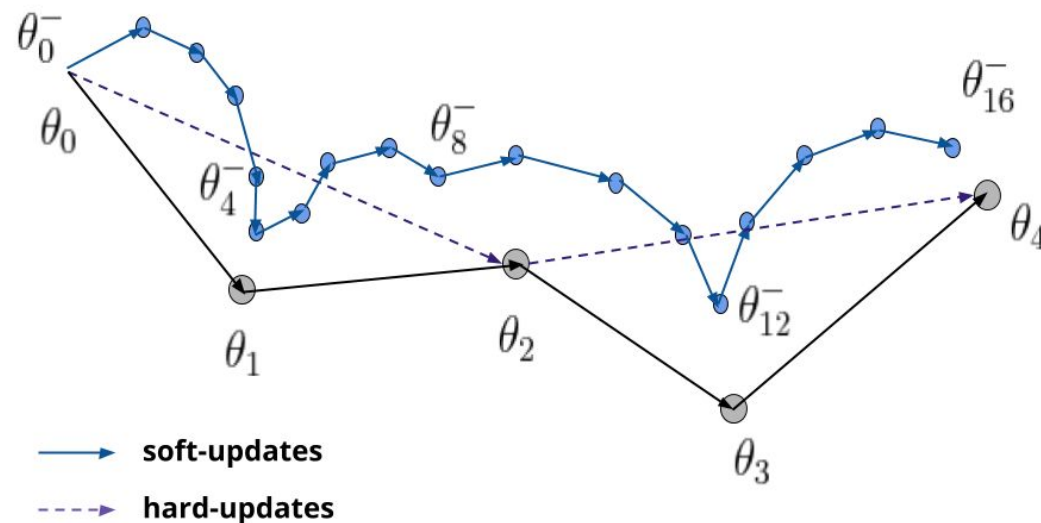
Deep Q-learning: Soft updates

- When the weights of the Q network are allocated entirely to the target network after T time steps, this is called **hard updates**:

$$\text{After T time steps: } \theta^- := \theta$$

- When T is relatively large, which is usually the case (in the order of thousands of steps), learning can be slowed down significantly. This is because any change in the Q function is propagated only after the target network update (i.e., after T time steps). These “jumpy” updates could also result in learning instability.
- To remediate this, **soft-updates** can be applied instead. The idea is to apply smoother weight updates to the target network instead of periodical integral update ($\tau < 1$):

$$\theta^- := \tau \theta + (1 - \tau) \theta^-$$



Source: Udacity Deep RL project 1, Gregor, 2018, wpumacay

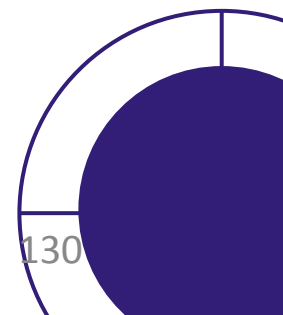


03





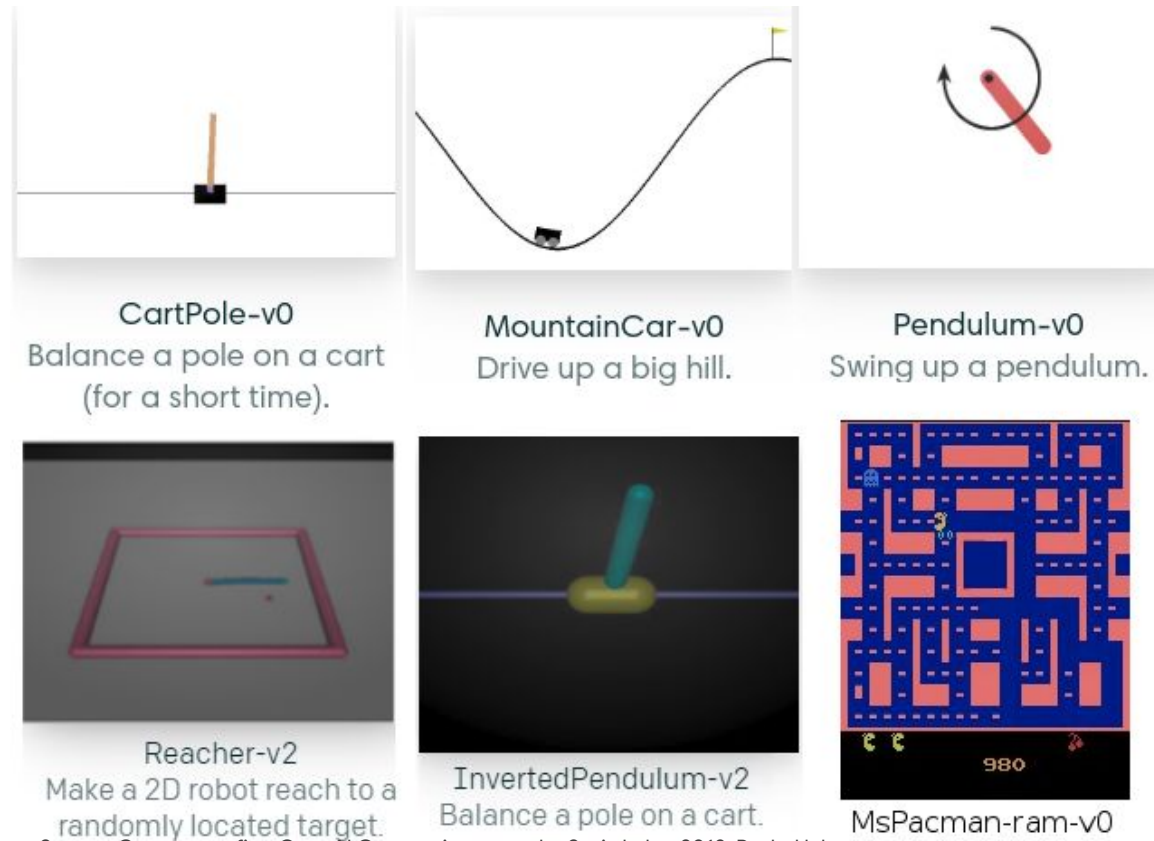
Gymnasium library



Gymnasium library

Gymnasium is a project that provides an API for all single agent reinforcement learning environments, and includes implementations of common environments: cartpole, pendulum, mountain-car, mujoco, atari, and more.

The API contains four key functions: [make](#), [reset](#), [step](#) and [render](#), that this basic usage will introduce you to.



Gymnasium library

The **environment** is represented in Gymnasium by the Env class, which has the following members:

- **reset():** This resets the environment to its initial state, returning the initial observation.

```
observation, info = env.reset(seed=42)
```

- **step():** allows the agent to take an action in the environment. In gymnasium, if the environment has terminated, this is returned by `step`. Similarly, we may also want the environment to end after a fixed number of timesteps, in this case, the environment issues a truncated signal. If either of terminated or truncated are true then reset should be called next to restart the environment.

```
observation, reward, terminated, truncated, info = env.step(action)
```

- **render():** This method allows to visualize the agent in action.

Gymnasium library

The **environment** contains also:

- **action_space**: This is the field of the Space class, providing a specification for allowed actions in the environment. It can be discrete, continuous or a combination of both.
- **observation_space** : This field has the same Space class, but specifies the observations provided by the environment. It can be discrete much like action spaces.



04





DQN improvements



DQN Improvements

- Dueling DQN :

Paper: <https://arxiv.org/abs/1511.06581>

$$Q(s,a) = V(s) + A(s,a)$$

Network will have two separate paths for value of state distribution and advantage distribution. On the output, both paths will be summed together, providing the final value probability distributions for actions. $V(s)$: the value of being at that state

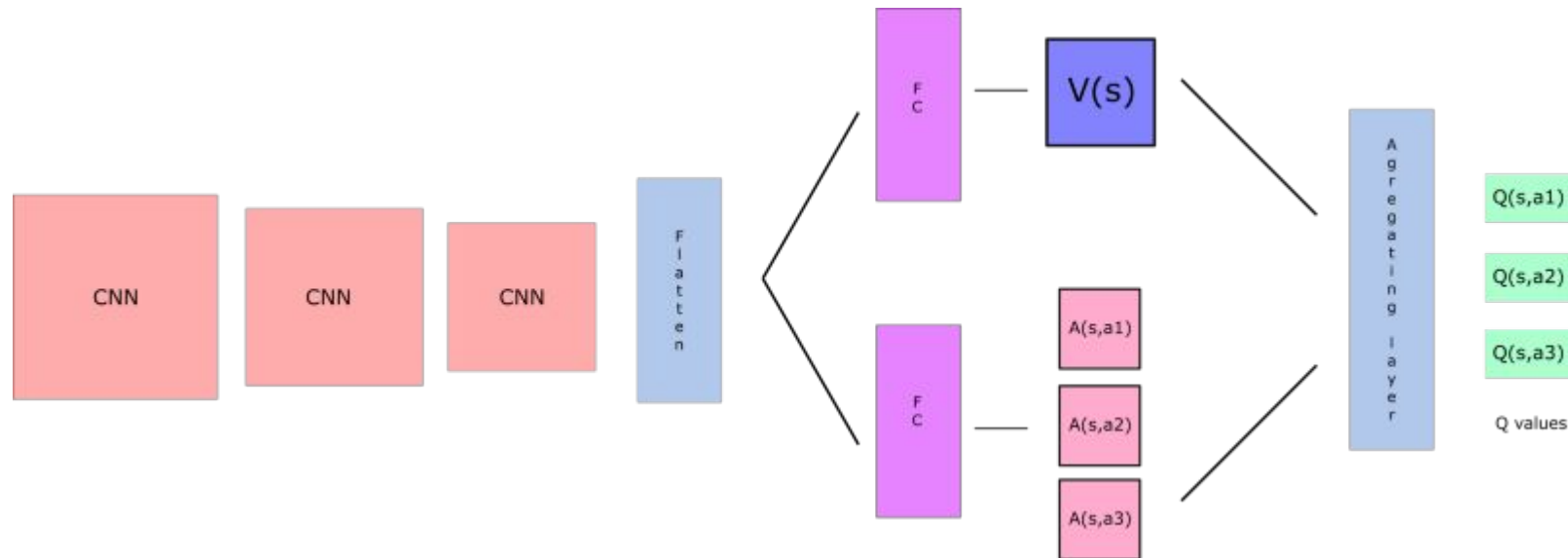
$A(s,a)$: the advantage of taking that action at that state (how much better is to take this action versus all other possible actions at that state).

DQN Improvements

- Dueling DQN:

By decoupling the estimation, intuitively our DDQN can learn which states are (or are not) valuable without having to learn the effect of each action at each state (since it's also calculating $V(s)$).

This is particularly useful for states where their actions do not affect the environment in a relevant way.



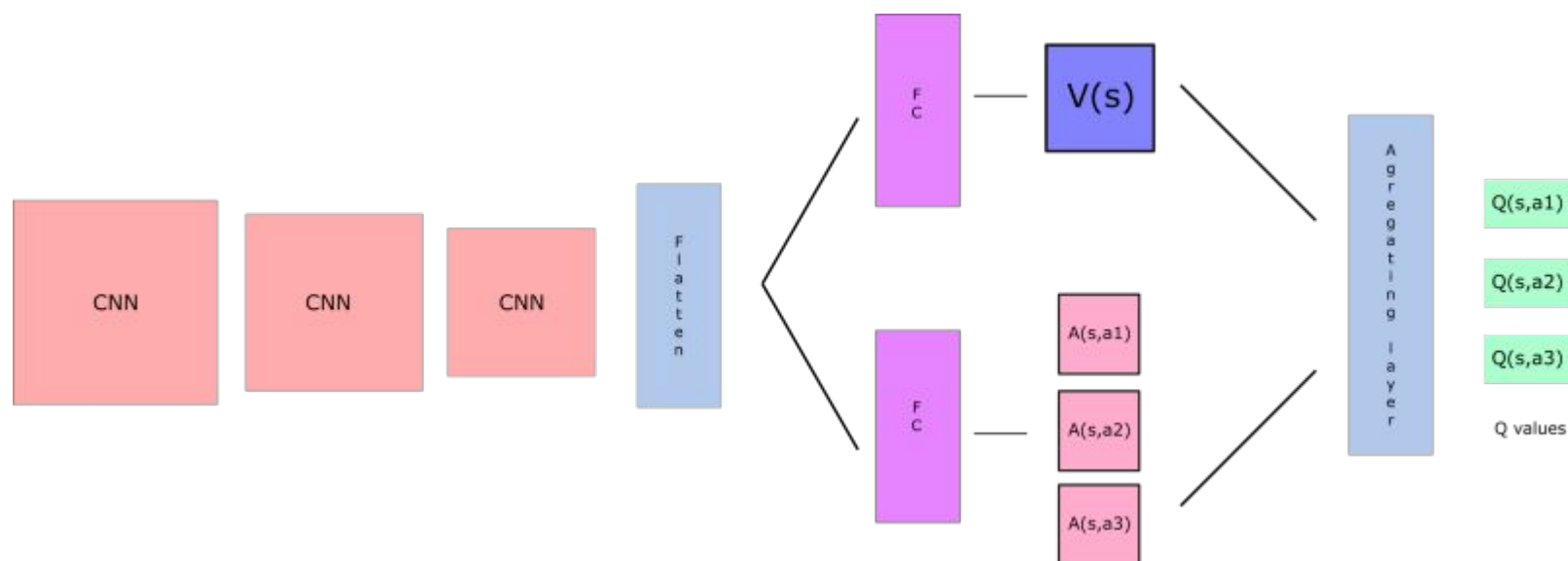
[Source:](#) Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed..., by Thomas Simonini, 2018, FreeCodeCamp

DQN Improvements

- Dueling DQN :

Paper: <https://arxiv.org/abs/1511.06581>

$$Q(s,a) = V(s) + A(s,a)$$



Source: Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed..., by Thomas Simonini, 2018, FreeCodeCamp

DQN Improvements



- Double DQN:

Paper: <https://arxiv.org/abs/1509.06461>

Problem addressed: Deep Q-Learning tends to overestimate action values.

⇒ Harmful to training performance.

⇒ Can lead to suboptimal policies.

Especially in early stages

Basic TD target:

$$Q(s,a) = r + \gamma \max_{a'} Q'(s',a')$$

Proposed TD target:

Choosing actions for the next state with the actual network but taking values of Q from target network

$$Q(s,a) = r + \gamma \max_{a'} Q'(s', \operatorname{argmax}_a Q(s_{t+1},a))$$

DQN Improvements



- Double DQN:

Paper: <https://arxiv.org/abs/1509.06461>

Problem addressed: Deep Q-Learning tends to overestimate action values.

Remember how we calculate the TD target

$$\begin{aligned} Q_{\text{tar}}^{\pi_{\theta}}(s, a) &= r + \gamma \max_{a'} Q^{\pi_{\theta}}(s', a') \\ &= r + \max(Q^{\pi_{\theta}}(s', a'_1), Q^{\pi_{\theta}}(s', a'_2), \dots, Q^{\pi_{\theta}}(s', a'_n)) \end{aligned}$$

Q target

reward of taking that action at that state

max q value among all possible actions from next state

If the max q value contain any errors, then it will be positively biased and the resulting Q-values will be overestimated. **We are not sure that the chosen action is the best action because:**

- An agent may not fully explore the environment
- The environment may be noisy

DQN Improvements



- Double DQN:

Overestimation in the face of uncertainty can be useful, e.g: at the beginning of training it can be helpful to overestimate $Q^\pi(s, a)$ for unvisited or rarely visited (s, a) pairs because this increases the likelihood that these states will be visited, allowing an agent to gain experience about how good or bad they are.

However, DQN overestimates $Q^\pi(s, a)$ for the (s, a) pairs that have been visited often. This becomes a problem if an agent does not explore (s, a) uniformly. Then the overestimation of $Q^\pi(s, a)$ will also be nonuniform and this may incorrectly change the rank of actions as measured by $Q^\pi(s, a)$. Under these circumstances, the a an agent thinks is best in s is in fact not the best action to take.

Solution: when we compute the Q target, we use two networks to decouple the action selection from the target Q value generation.

- use our DQN network to select what is the best action to take for the next state (the action with the highest Q value).
- use our target network to calculate the target Q value of taking that action at the next state.

$$Q_{\text{tar:DoubleDQN}}^\pi(s, a)r + \gamma Q^{\pi_\varphi}(s', \max_{a'} Q^{\pi_\theta}(s', a'))$$

DQN Improvements



- Noisy networks:

Paper: <https://arxiv.org/pdf/1706.10295.pdf>

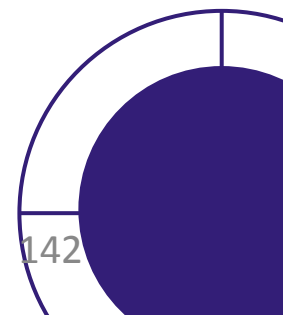
Problem addressed: exploration of the environment.

Independent Gaussian noise: random value drawn from normal distribution

Factorized Gaussian noise: keeping only two random vectors: one with the size of input and another with size of the output of the layer.

05

General advices



How do I frame my task in RL



- Follow the MDP formalism.
- Start with simplified version of your task until you see signs of life.
- Simplify the feature space. Once it starts working, make the task harder until you solve the full task.
- Simplify the reward function. Formulate so it can give you FAST feedback to know whether you're doing the right thing or not.

How can I diagnose my RL agent behaviour?

⇒ Sanity checks

- Sensitivity to the change in EVERY hyper parameter is considered as bad sign ⇒ non robustness.
- Look at the episode return min/max/stddev/mean, max is important not just mean
- Look at the episode length (sometimes more informative than return.
- Health indicators differ from one class of algorithm to another. Policy gradients VS Q-learning



How can I diagnose my RL agent behaviour

Q-learnings



- Metrics:
 - How to measure if your agent is converging to some locally optimal policy \Rightarrow epsilon-greedy
 - \Rightarrow Epsilon schedules are important
 - Learning rate schedule are helpful. This should be decreasing over time.
 - TD-error is decreasing
 - The action-values estimates should increase as the cumulative reward increases.

How do I evaluate my RL agent

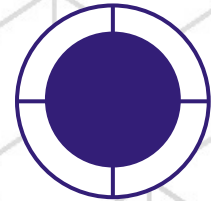
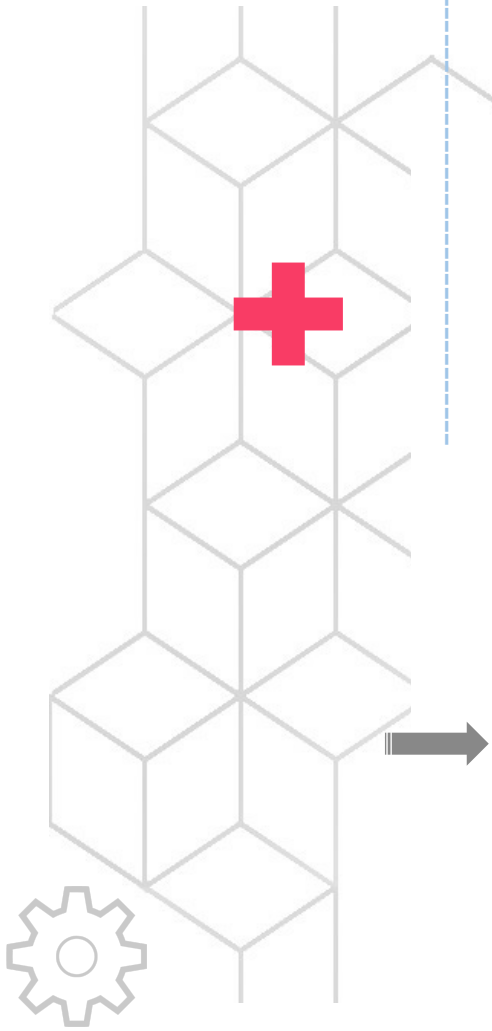


- Use a separate test environment to evaluate the performance of your agent at a given time.
- Evaluate your agent for n test episodes and average the reward per episode to have a good estimate. (n between 5 and 20)



Course #3

Policy Based methods



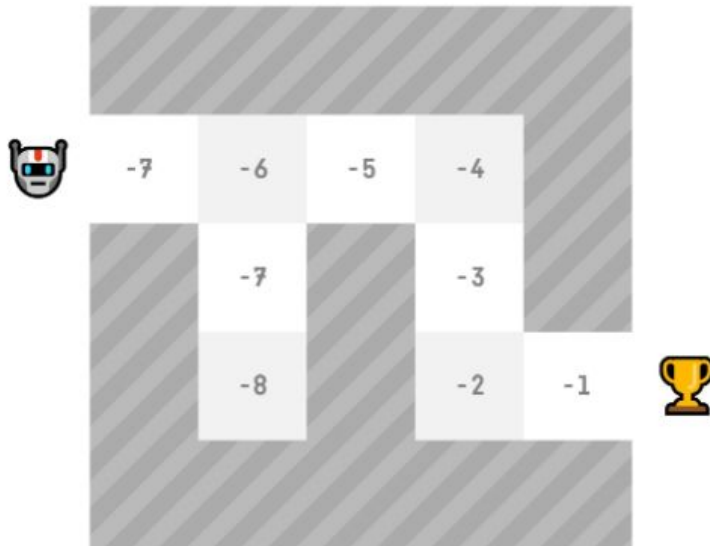
Recall: Values and policy

Recall: The ultimate goal of an RL agent is to find a policy π that achieves a lot of reward over the long run. We find this policy through training. To train the agent,

$$v(s) / Q(s, a)$$

Value-Based Learning

Teach the agent to learn which state is **more valuable** and then take the actions that **leads to the more valuable states**

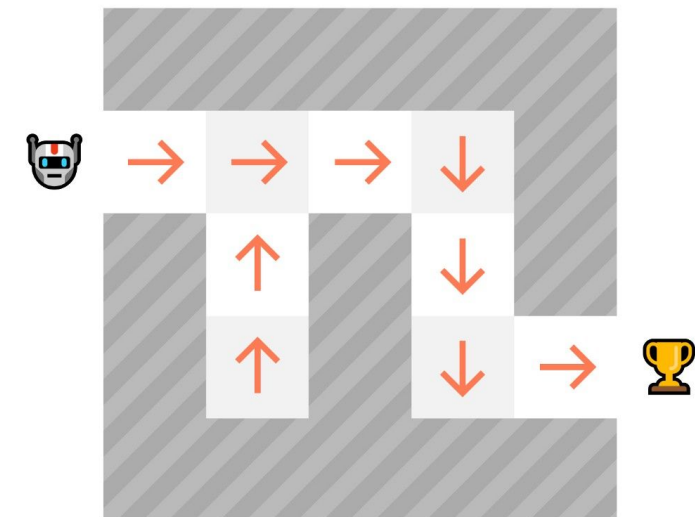


Source: Hugging Face Deep RL Course 2018

$$\pi(a|s)$$

Policy-Based Learning

The agent learns a policy function directly without passing through a value function. The agent learns which action to take, given the state is in.



Source: Hugging Face Deep RL Course 2018

Policy representation

- Recall that in Q-learning, the Q function was parameterized by a neural network that returns the values of actions as scalars. These values then dictate to us how to behave as we select the action with the largest value.
- In policy-based methods, we learn a policy function directly by parameterizing it:

$$\pi(a_t|s_t) \rightarrow \pi(a_t|s_t; \theta)$$

- Learning the policy means that we are going to look for the parameters θ that maximize a certain objective function $J(\theta)$, which is a performance measure with respect to parameter θ .



Policy representation: Deterministic Policies

- **Deterministic Policy:**

$$\pi : s \rightarrow a$$

Instead of sampling from the action probabilities, the agent need only choose the greedy action.
The last layer of the neural network representing the policy is the action to be taken.



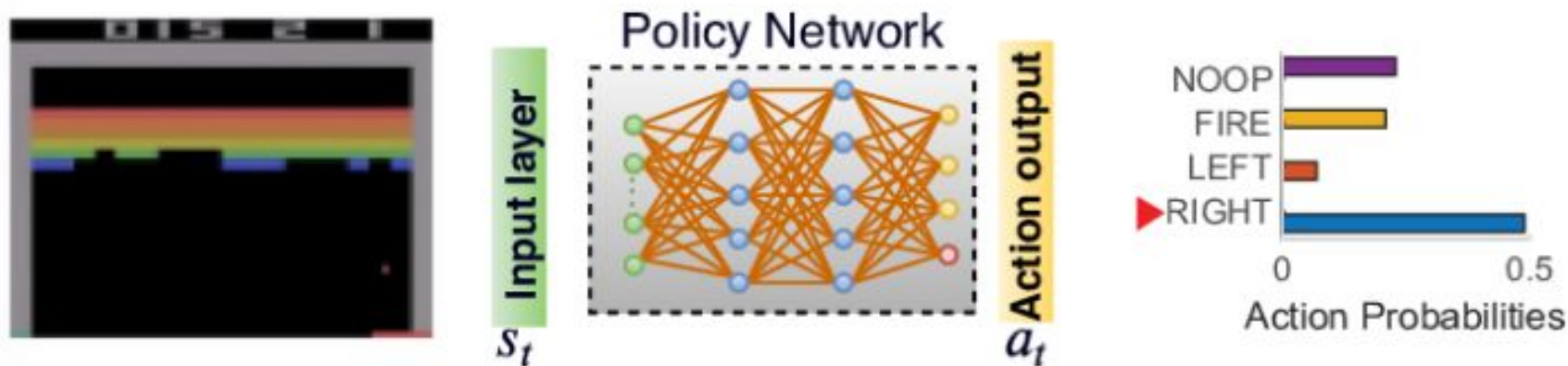
Policy representation: Stochastic Policies

- **Stochastic Policy:**
 - Categorical policies:

$$\pi(a_i|s_t; \theta) = P[a_t|s_t] = \text{softmax}(a_i)$$
$$a_i = \max_i \pi(a_i|s_t; \theta)$$

The agent passes the current environment state as input to the network, which returns action probabilities. Then, the agent samples from those probabilities to select an action.

Example: Atari game



Source: modified from Qu, X., Sun, Z., Ong, Y. S., Gupta, A., & Wei, P. (2020). Minimalistic attacks: How little it takes to fool deep reinforcement learning policies. IEEE Transactions on Cognitive and Developmental Systems, 13(4), 806-817.

Policy representation: Stochastic Policies

- **Stochastic Policy:**

- Gaussian policies: used mostly with continuous action spaces. The policy is a sample from a Gaussian distribution.

$$\pi(a_t|s_t; \theta) \sim N(\mu(s_t), \sigma^2(s_t))$$

The mean μ and standard deviation of the normal distribution are both functions of the state features.

Quiz!

The neural network that approximates the policy takes the environment state as input. The output layer returns the probability that the agent should select each possible action. Which of the following is a valid activation function for the output layer??

A: linear (i.e; no activation function)

B: Softmax

C: ReLu

Quiz!

For continuous action spaces, the neural network has one node for each action entry (or index). For example, consider the action space of the bipedal walker environment, shown in the figure below.

Actions

Type: Box(4) - Torque control(default) / Velocity control - Change inside
`/envs/box2d/bipedal_walker.py` line 363

Num	Name	Min	Max
0	Hip_1 (Torque / Velocity)	-1	+1
1	Knee_1 (Torque / Velocity)	-1	+1
2	Hip_2 (Torque / Velocity)	-1	+1
3	Knee_2 (Torque / Velocity)	-1	+1

Quiz!

→ In this case, any action is a vector of four numbers, so the output layer of the policy network will have four nodes.

→ Every entry in the action must be a number between -1 and 1

Which of the following describes a valid output layer for the policy?

A: output layer with ReLu activation function.

B: output layer with softmax activation function

C: output layer with tanh activation function

Why policies may be more attractive than values?

- **Simplicity:** The policy is all what we are looking for when we are solving a RL problem. Therefore, whenever the agent is in a given state, it is more straightforward to use the policy directly to decide its next move instead of computing and/or storing the value of a state or action and then select the action that maximizes these values as in value-based methods. Doing this extra work of computing the Q or V values could be tedious especially for large action spaces. Why do the extra work?

**Policy-based
learning**



Go Right



**Value-based
learning**

Please wait, I am still
calculating Q value, only
41891 actions left...

Why policies may be more attractive than values?

- **Stochastic policies:** An extra benefit of policy-based methods is that they can learn a stochastic policy while value functions can't. One advantage of a stochastic policy is that it can capture the uncertainty/stochasticity of the environment. With a stochastic policy, the same state could lead to different actions (it is possible to have more than one action to choose from in a certain situation).
- For example: In a poker game, the agent may not take the same action in response to the same hand since the probability of winning or losing depends on the opponent's hand and how the betting has proceeded.



Source: PokerListings, 2023

Why policies may be more attractive than values?

- **High dimensional or continuous action spaces:** In Q-learning for example, to be able to decide on the best action to take having $Q(s,a)$ we need to solve a small optimization problem finding a , which maximizes $Q(s,a)$. In the case of Atari with several discrete actions this wasn't a problem: we just approximated values of all actions and took the action with the largest Q .

But, if we have a large number of possible actions or an infinite possibility of actions? This optimization problem becomes hard as Q is usually represented by nonlinear NN, so finding the argument that maximizes the function's values can be tricky. In such cases, it's more feasible to avoid values and work with the policy directly.

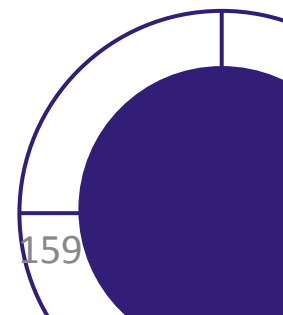


Source: Hugging Face Deep RL Course 2018

For a self-driving car,, you can have a near infinite choice of actions (it can turn left by turning the wheel at 15° , 17.2° , 20° , 21.1° , 21.2° , honk, turn right at 20° , etc...)

01

Policy gradients methods



159

Policy gradients: A subclass of policy-based methods



Policy gradient methods are a **subclass of policy-based methods** that estimate the weights of a policy through **gradient-ascent**.

In Policy-gradient methods, we optimize the parameter θ **directly** by performing gradient ascent on the objective function $J(\theta)$, which is the performance measure.

Note that there are other classes of policy-based methods where we optimize the parameter θ **indirectly** by maximizing the local approximation of the objective function with techniques like **hill climbing, simulated annealing or evolution strategies**.

Policy gradients: Big picture

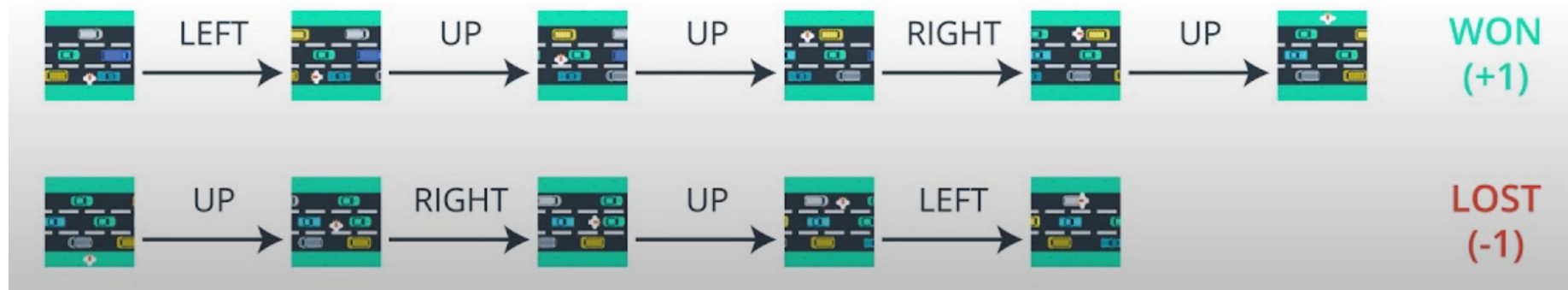
We just learned that policy-gradient methods aim to find parameters θ that maximize the expected return.

Question: How we're going to change our network parameters using the expected return to improve the policy?

Answer: The idea is that we're going to let the agent interact during an episode. And if we win the episode, \Rightarrow We can change the network weights a bit to make it more likely to select the actions it selected while in those states in the future.

If the agent has lost the game, we update the network weights so that it is less likely to repeat these decisions in the future.

So, eventually, for each state-action pair, we want to increase the $P(a | s)$: the probability of taking that action at that state. Or decrease if we lost.



Source: Deep Reinforcement learning nanodegree program, Udacity 2022

Policy gradients: Big picture

The Policy-gradient algorithm (simplified) looks like this:

Training Loop:

Collect an **episode with the π** (policy).

Calculate the return (sum of rewards).

Update the weights of the π :

If **positive return** → **increase** the probability of each (state, action) pairs taken during the episode.

If **negative return** → **decrease** the probability of each (state, action) taken during the episode

In policy-based methods, the optimization is most of the time **on-policy** since for each update, we only use data (trajectories) collected by our most recent version of π_θ

Policy gradients: More formally

Trajectory: sequence of states and actions.

$$\tau = s_0, a_0, s_1, a_1, \dots, s_H, a_H$$

A trajectory could correspond to a full episode or a part of the episode.

Horizon is the length of a trajectory, denoted by H .

$R(\tau)$ is the sum of discounted rewards from that trajectory.

$$R(\tau) = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{H-1} r_H$$

Policy gradients: More formally

Policy-gradient is an optimization problem: we want to find the parameters θ that **maximize** our objective function $J(\theta)$. So, we need to use **gradient-ascent**. Our step for gradient ascent is:

$$\theta \leftarrow \theta + \alpha * \nabla_{\theta} J(\theta)$$

α is the step size that is generally allowed to decay over time. We can repeatedly apply this update rule in the hopes that θ converges to the value that maximize $J(\theta)$.

Policy gradient: Quiz!

- Q1: Why do we use gradient ascent instead of gradient descent to optimize $J(\theta)$?

	We want to minimize $J(\theta)$ and gradient ascent gives us the gives the direction of the steepest increase of $J(\theta)$
	We want to maximize $J(\theta)$ and gradient ascent gives us the gives the direction of the steepest increase of $J(\theta)$

Policy gradients: More formally

Objective Function: gives us the performance of the agent given a trajectory and it outputs the expected return (called also expected cumulative reward).

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

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

Trajectory (read Tau)
Sequence of states and actions

Return: cumulative reward

Gamma: discount rate

Policy gradients: More formally

Objective Function: gives us the performance of the agent given a trajectory and it outputs the expected return (called also expected cumulative reward).

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

- The expected return can be calculated as a weighted average as follows:

$$J(\theta) = \sum_{\tau} \underbrace{P(\tau; \theta)}_{\text{Probability of the trajectory (depends on } \theta \text{ since it defines the policy that it uses to select the actions of the trajectory which as an impact of the states visited).}} \underbrace{R(\tau)}_{\text{Cumulative return from trajectory}}$$

We calculate the expected return $J(\theta)$ by summing for all trajectories, the probability of taking that trajectory given θ and the return of this trajectory.

Policy gradients: More formally

If we develop further the definition of the objective function provided earlier, the objective function can be expressed in terms of the policy as follows:

$$J(\theta) = \sum_{\tau} P(\tau; \theta) R(\tau)$$
$$P(\tau; \theta) = \left[\prod_{t=0}^{\infty} \underbrace{P(s_{t+1} | s_t, a_t)}_{\text{Environment dynamics (state distribution)}} \underbrace{\pi_{\theta}(a_t | s_t)}_{\text{Probability of taking that action } a_t \text{ at state } s_t} \right]$$

There are two problems with using the expression above for computing the derivative of $J(\theta)$

- **Problem 1:** We can't calculate the “true” gradient of $J(\theta)$ as it involves calculating the probability of **each possible trajectory** → computationally expensive. Instead, we would rather use sample-based estimate based on the experience collected from some trajectories.
- **Problem 2:** The expression above involves the knowledge of state distribution (i.e., environment dynamics). But, this may not be known especially if our focus is on model-free reinforcement learning.

Policy gradients: More formally

How overcome these two problems and find an estimate of the gradient of the objective function?



Good news: Policy gradient theorem!

This theorem will help us in deriving a differentiable expression for the objective function that does not involve the use of the state distribution. The policy gradient theorem states that:

For any differentiable policy and for any policy objective function, the policy gradient is:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau)]$$

Source: Hugging Face Deep RL Course 2018

Policy gradients: REINFORCE



REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

```
1 Loop forever (for each episode):
2   Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot|\cdot, \theta)$ 
3   Loop for each step of the episode  $t = 0, 1, \dots, T - 1$ :
4      $G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$  ( $G_t$ )
5      $\theta \leftarrow \theta + \alpha G \nabla \ln \pi(A_t|S_t, \theta)$ 
```

Source: Reinforcement Learning-An Introduction, a book by Richard Sutton

Policy gradients: REINFORCE

More into the gradient of the objective function:

$$\nabla_{\theta} J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^m \sum_{t=0} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) R(\tau^{(i)})$$

Estimation of the gradient (given we use some trajectories to estimate the gradient)

Scaling factor inversely proportional to the number of trajectories (m)

Probability of the agent to select action at from state st

Cumulative return of i-th trajectory

The gradient is equal to the gradient of the log-probability of the action taken and it is scaled by the return :

- If return $R(\tau)$ is high: it will push up the probabilities of the state-action combinations
- otherwise, it will push down the probabilities of the state-action combinations

⇒ In other words, we are trying to increase the probability of the actions that have given us good total reward and decrease the probability of actions with bad final outcomes.

Policy gradients: REINFORCE



Difference from Q-learning:

- No explicit exploration is needed.
- In Q-learning: epsilon-greedy strategy. Now, with probabilities returned by the network, the exploration is performed automatically. In the beginning, the network is initialized with random weights and the network returns uniform probability distribution. This distribution corresponds to random agent behaviour.
- No replay buffer is used. PG belong to the on-policy methods class. We can't train on data obtained by an old policy.
- NO target network is needed.

Policy gradients: REINFORCE limitations



- The update process is very **inefficient**. We run the policy once, update once, and then throw away the trajectory.
- **Correlation between samples:** training samples in a single episode are usually highly correlated, which is bad for SGD training. For DQN, this was solved by considering a replay buffer. But, this solution is not applicable to the policy gradient family because these methods are on-policy. To solve this, the idea is, instead of communicating with one environment, we use several parallel environments and use their transitions as training data.
- **Local optimum and exploration issues:** Even with the policy represented as a probability distribution, there is a risk that the agent converges to some local optimal policy and stops exploring the environment. In DQN, this was solved by epsilon-greedy action selection. In policy-gradient methods, one solution for this is the use of **entropy bonus**.

Policy gradients: The Entropy Bonus

To prevent the agent from being stuck in a local optimum, we first compute the entropy of the policy:

$$H(\pi) = - \sum \pi(a|s) \log \pi(a|s)$$

The entropy is a measure of uncertainty. It is positive and high when all actions have the same probability. The entropy become minimal if the agent has 1 probability for one action and 0 for all others (i.e., when the agent is 100% sure about an action)

Once the entropy is computed, it is then subtracted from the loss function in order to punish the agent for being too certain about the action to take. Note that the loss function is simply the negative of the objective function.

⇒ this introduces new hyperparameter called `entropy_beta`. It is the scale of the entropy bonus in the loss function expression.

Policy gradients: REINFORCE limitations

- **High gradients variance:** The gradient formula is proportional to the discounted reward while the range of this reward is heavily dependent on the environment.

For example, in the cartpole, if the pole is held for 5 steps, the reward (undiscounted) is five. But, if we hold it for 100 steps, the total reward is 100. So, there is a large difference between these two scenarios. We need to do something about this, otherwise the training could become unstable.

The simplest way for handling this is to subtract a value called baseline $B(s_t)$ from the return. This baseline $B(s_t)$ can be any function as long as it does not depend on the action. Some possible choices of the baseline are:

- A constant value, which is normally the mean of the discounted rewards
- The moving average of the discounted reward
- The value of the state $V(s)$



Policy gradients: REINFORCE with baseline

REINFORCE with Baseline (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

 Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (G_t)$$

$$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$$

$$\theta \leftarrow \theta + \alpha^{\theta} \delta \nabla \ln \pi(A_t | S_t, \theta)$$

Source: Reinforcement Learning-An Introduction, a book by Richard Sutton

Learning material



- Reinforcement Learning-An Introduction, a book by Richard Sutton and his doctoral advisor Andrew Barto. An online draft of the book is available [here](#)
- Teaching [material](#) from David Silver including video lectures is a great introductory course on RL
- Technical [tutorial](#) on RL by Pieter Abbeel and John Schulman (Open AI/ Berkeley AI Research Lab).
- Reinforcement learning hands-on (Second edition), a book with [tutorials](#) by Maxim Iapan.
- Huggingface deep RL course
- Deep Reinforcement Learning nanodegree on [Udacity](#).
- [Andrej Karpathy's Deep Reinforcement Learning: Pong from Pixels](#) is a great introduction to build motivation and intuition.



THANK YOU!



17/02/2025

