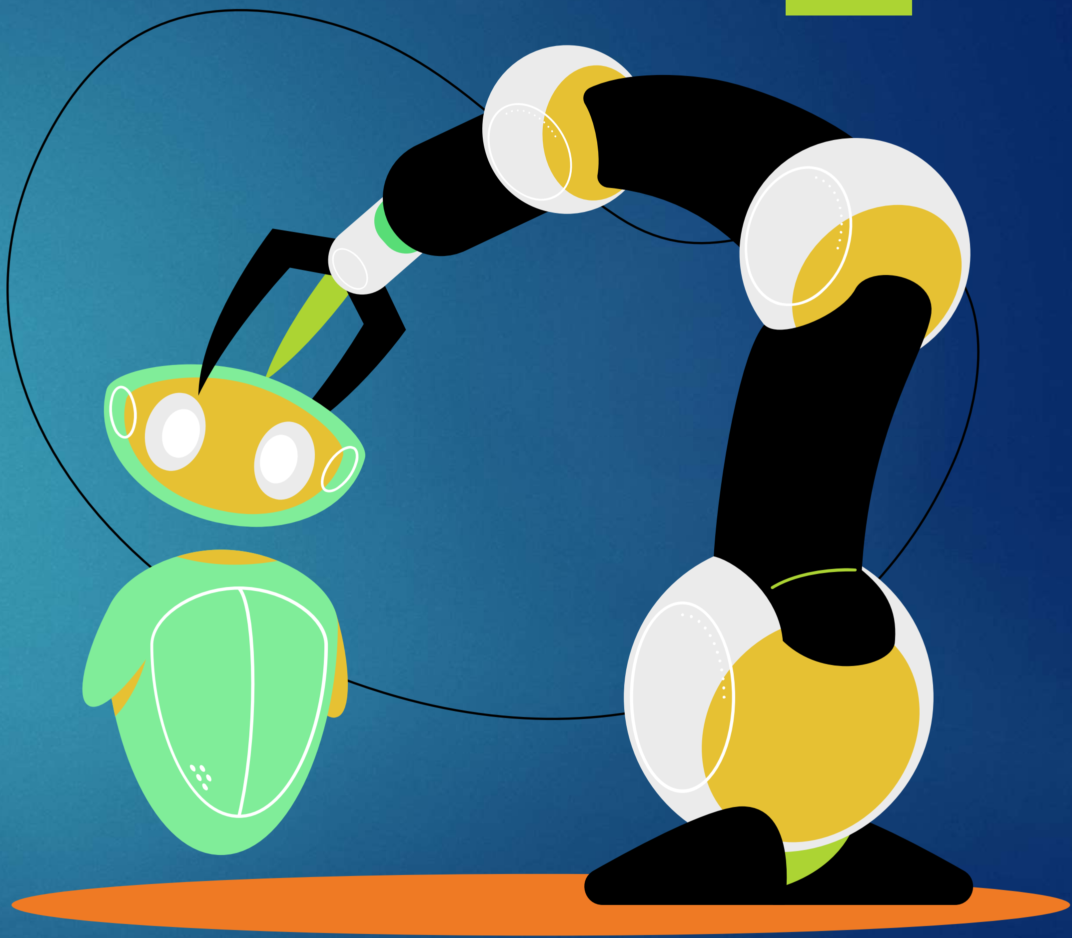
Reinforcement Learning

INSTRUCTORS: DOLPH & HOANG

MLDA @EEE

MACHINE LEARNING AND DATA ANALYTICS



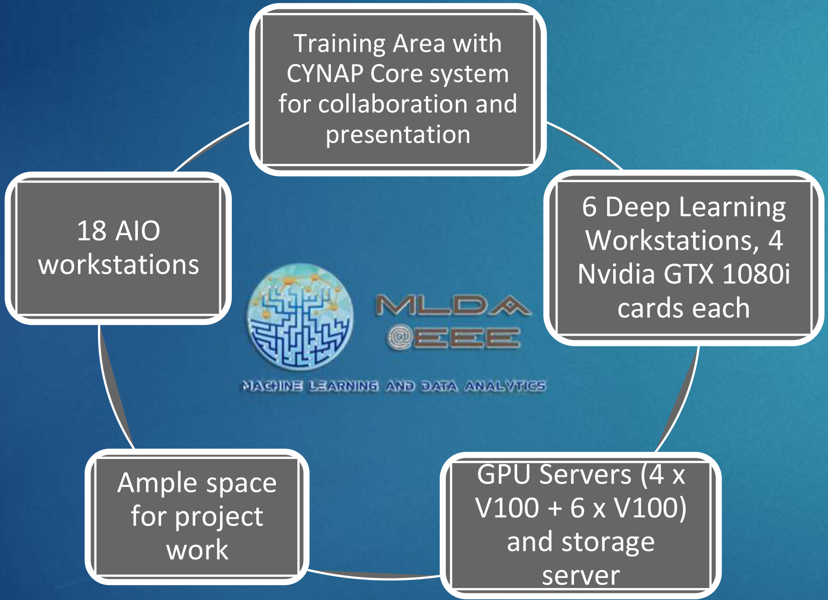
Who are we?

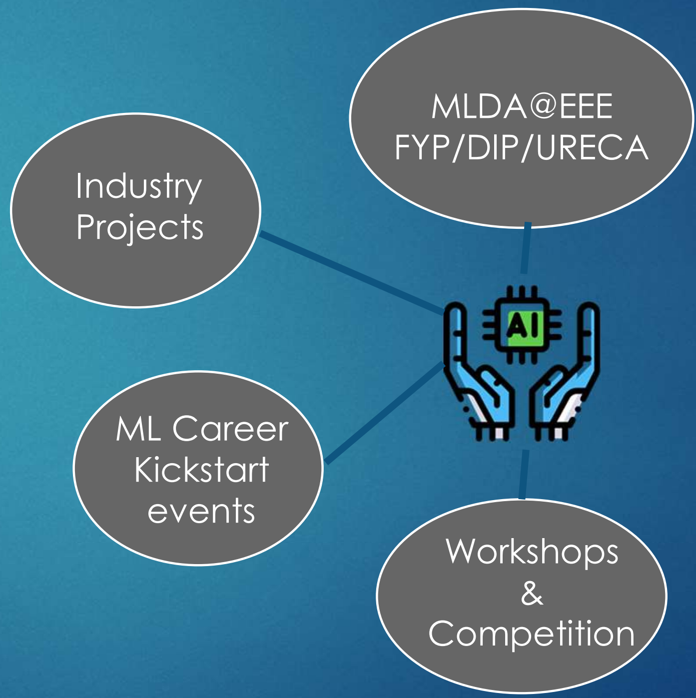
Dolph Xia
Tianyi
BCG Year 2

Hoang
CSC Year 2

Our Mission

Provide an integrated platform for NTU students to learn and implement Machine Learning, Data Science & AI, as well as facilitate connections with the industry.





- >1000 Trained ML practitioners
- >10 Academic Projects
- >30 Industry projects
- >5 competition
- >15 Industry Partners

Table of Contents

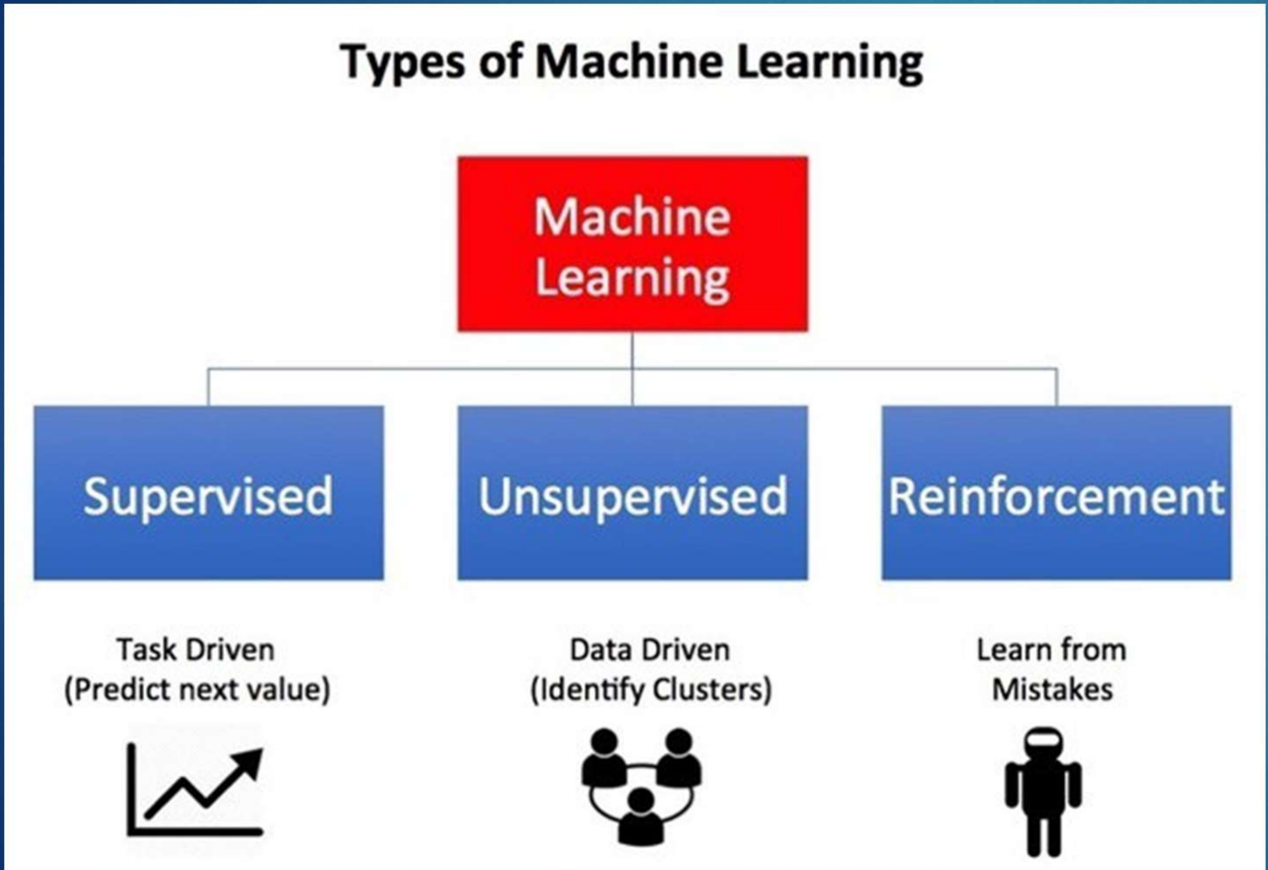
1 Introduction

What is reinforcement learning (RL)?

How you learn to play a new game?

- None/ minimal instructions
- Just try out different combinations and see how they work out

What is Reinforcement Learning?



The agent learns by interacting with the environment.

Goal: **maximize the total reward**

- **Trial and error** search for optimal actions
- Actions affect immediate and subsequent rewards. **Feedback can be delayed**

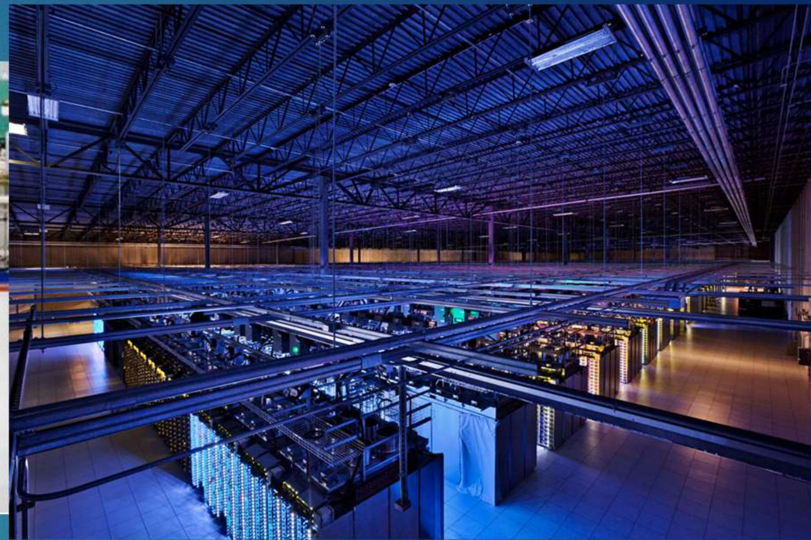
RL Application: Game

Clear goals, sequential decision making

"This indicates that reinforcement learning can yield long-term planning with large but achievable scale — without fundamental advances, contrary to our own expectations upon starting the project."

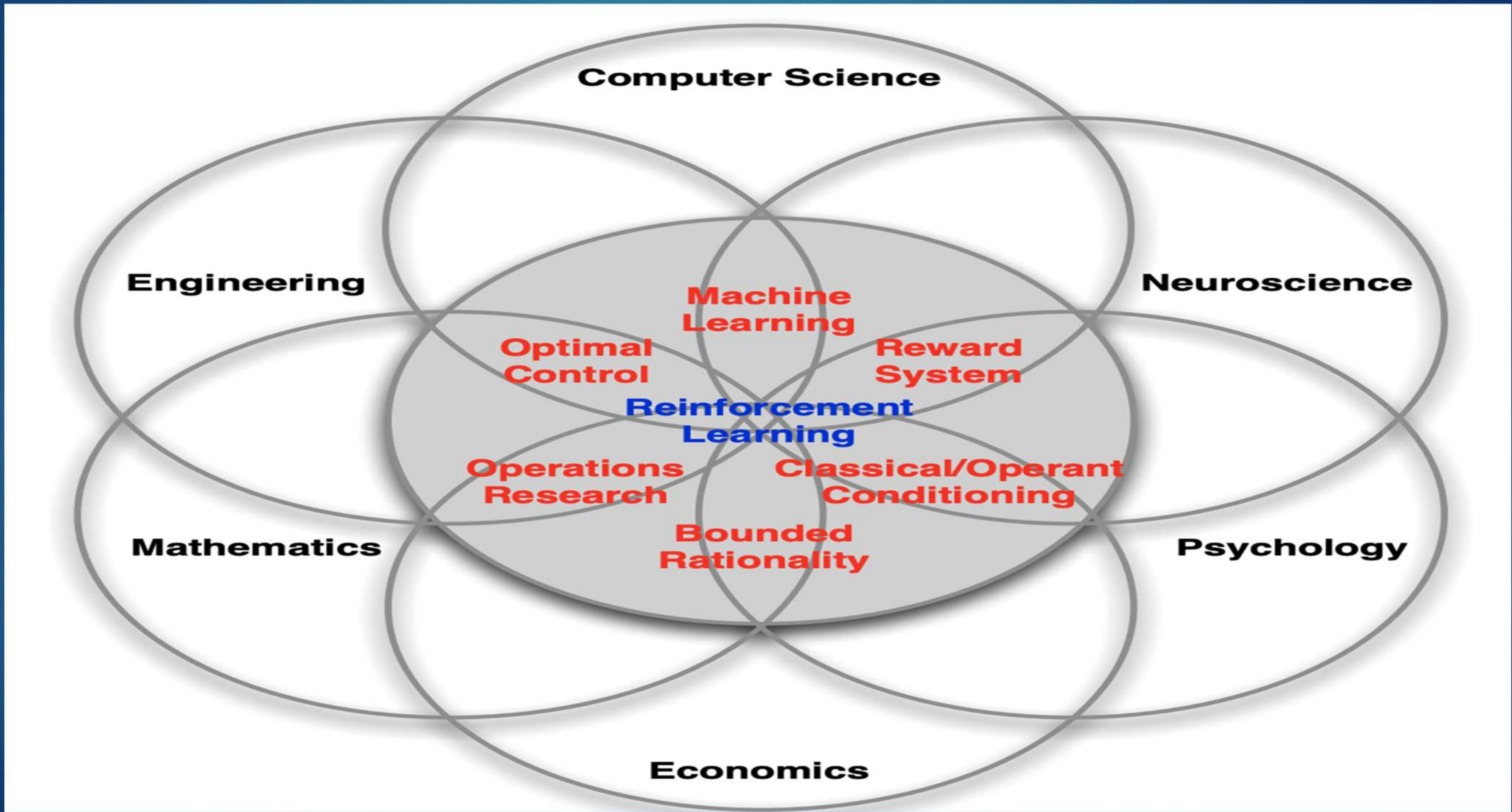
RL Application: autonomous driving

RL Application: Automated system



RL Application: robot

Many Facets of Reinforcement Learning



2

Markov Decision Process

Key Terminologies

How to model the problem?

How to decide what to do?

- The current state
- Available options
- Effect of actions on the environment

Markov Decision Process

<S,A,P,R,γ>

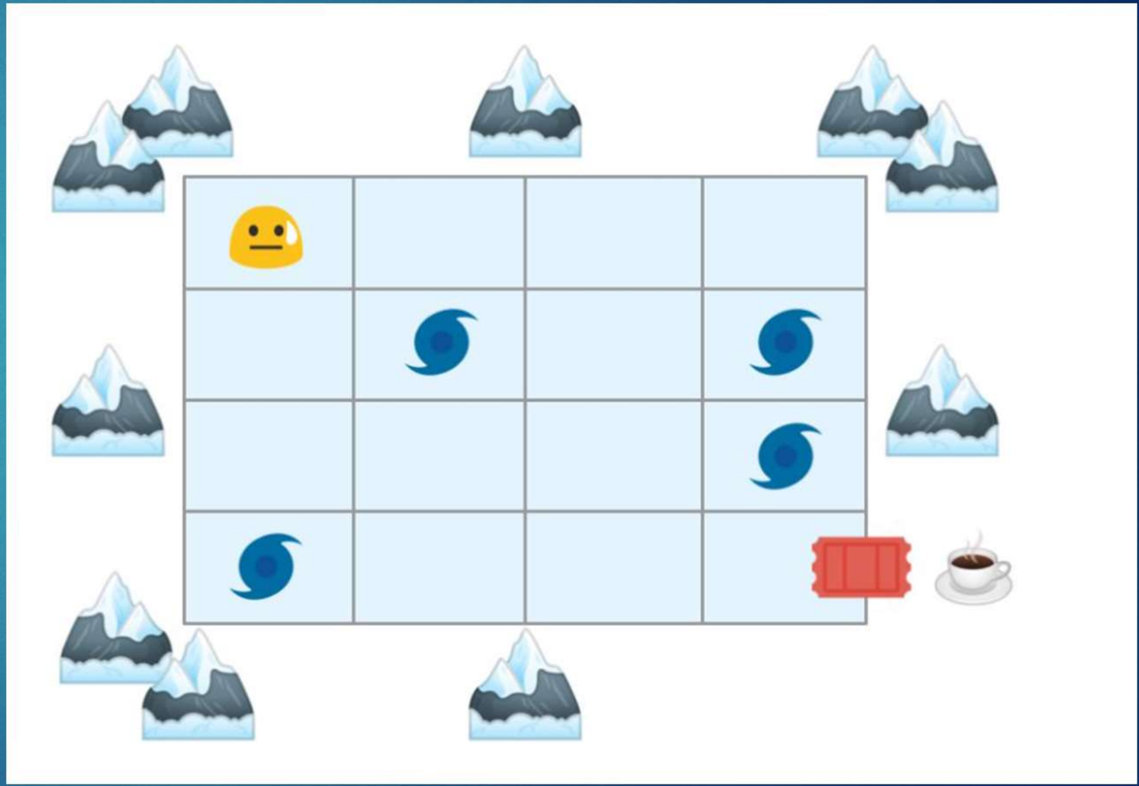
S: **State**

A: **Action**

P: **Transition**: how the env changes

R: **Reward**

γ: **discount factor**: make total reward converge



Markov Decision Process

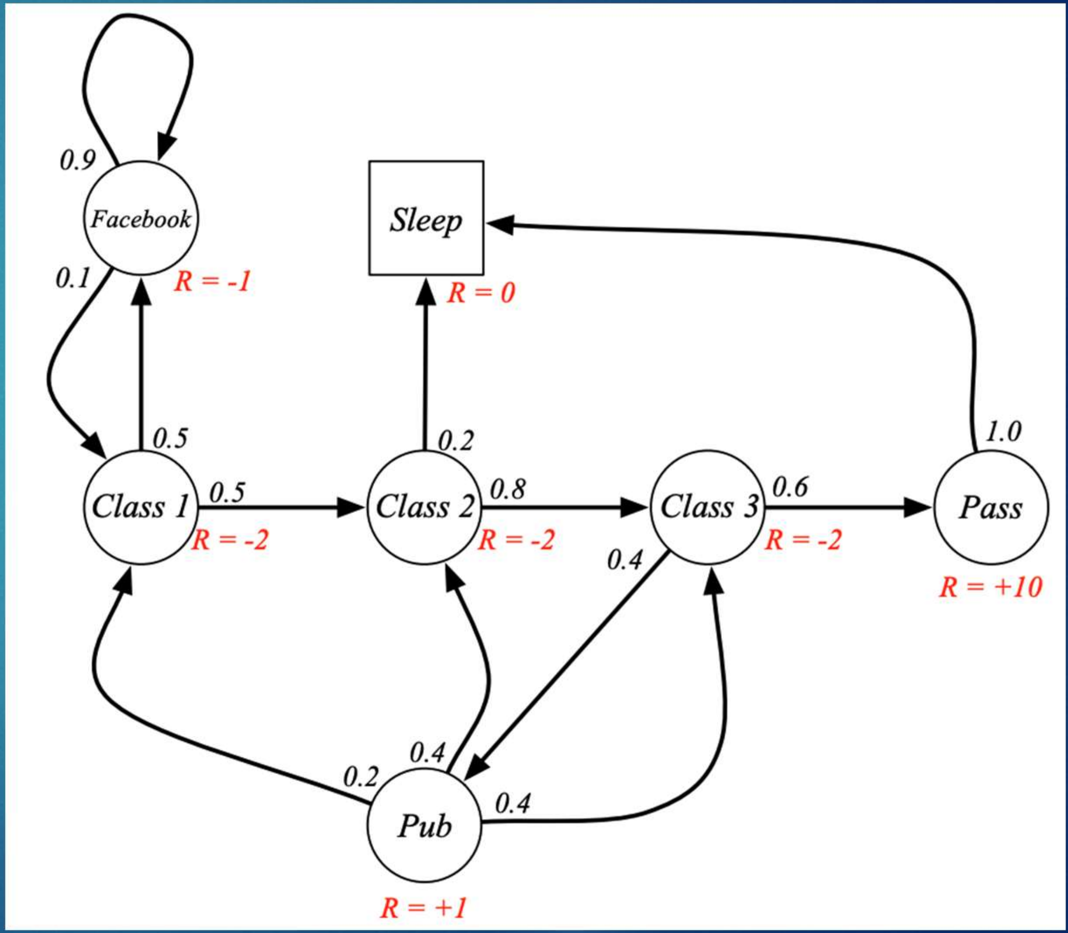
Policy π: The strategy of the agent

The action for each state

Example 1:
"always attend lectures"

Example 2:
"follow the probability shown in the picture"

Return: total discounted reward till terminal state



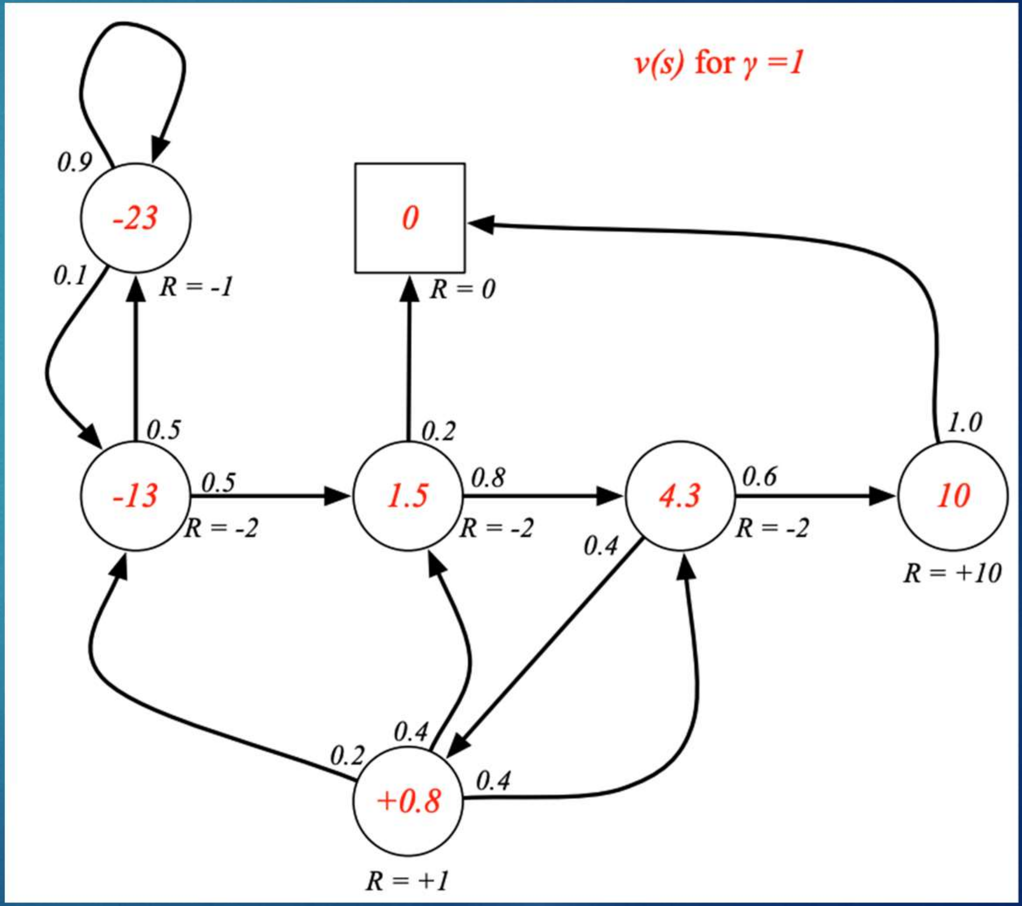
Markov Decision Process

State value function: $v_\pi(s)$

The expected return from state *s*,

under policy π

Example :
Value of 'Pass' is 10.



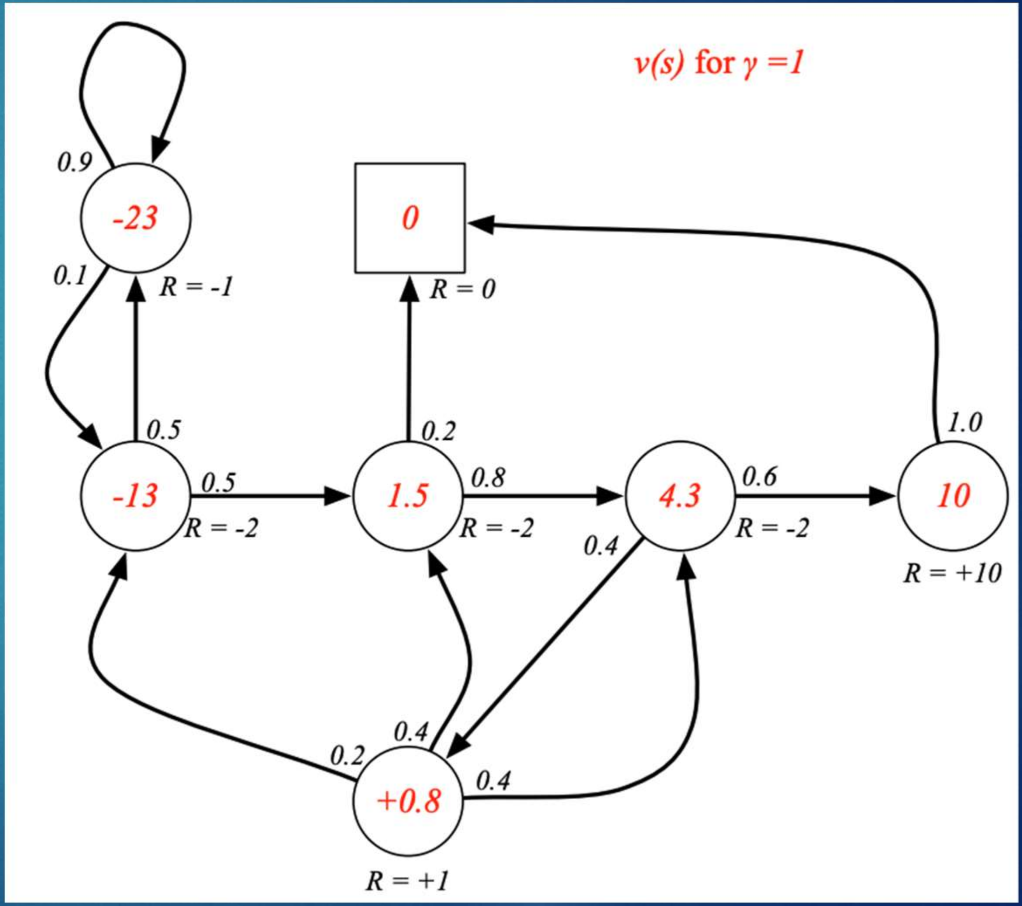
Markov Decision Process

Action value function: $q_\pi(s, a)$

The expected return from state *s*,

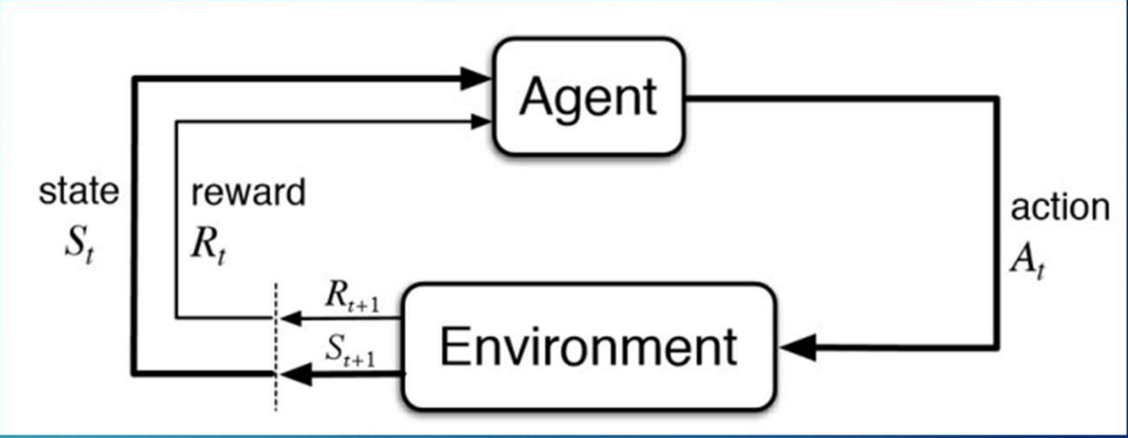
taking action *a*, under policy π

Example :
Value of 'Stay in Facebook state' is -23.



Markov Decision Process

Sequential decision making



Future states are **only dependent on the present** state

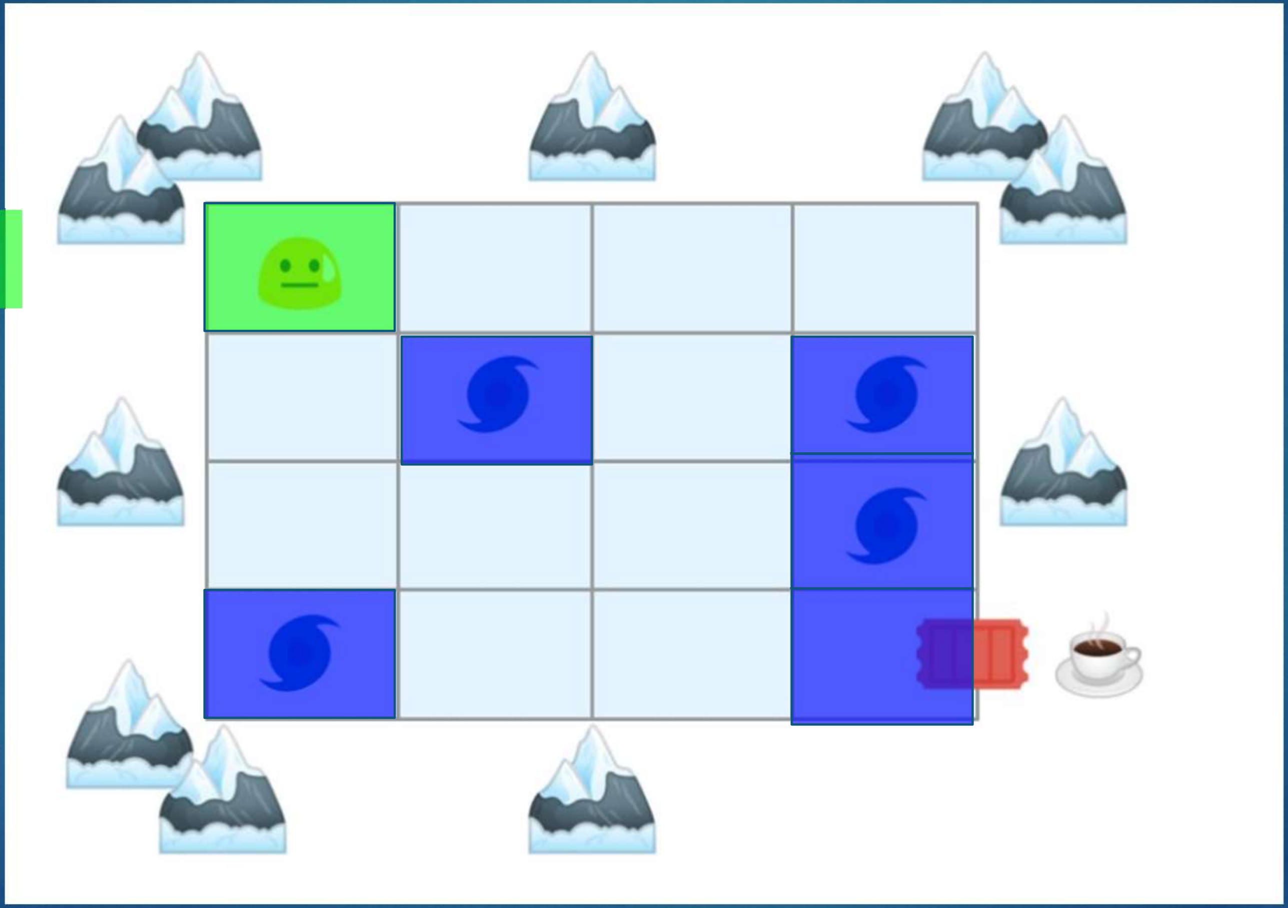
<S,A,P,R,γ>: state, action, transition, reward, discounting factor

$$\mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

An example

State: index of box (0 for start state) **Action**: 0 left 1 down 2 right 3 up

Start state



Terminal states

Return: total discounted future rewards

Policy

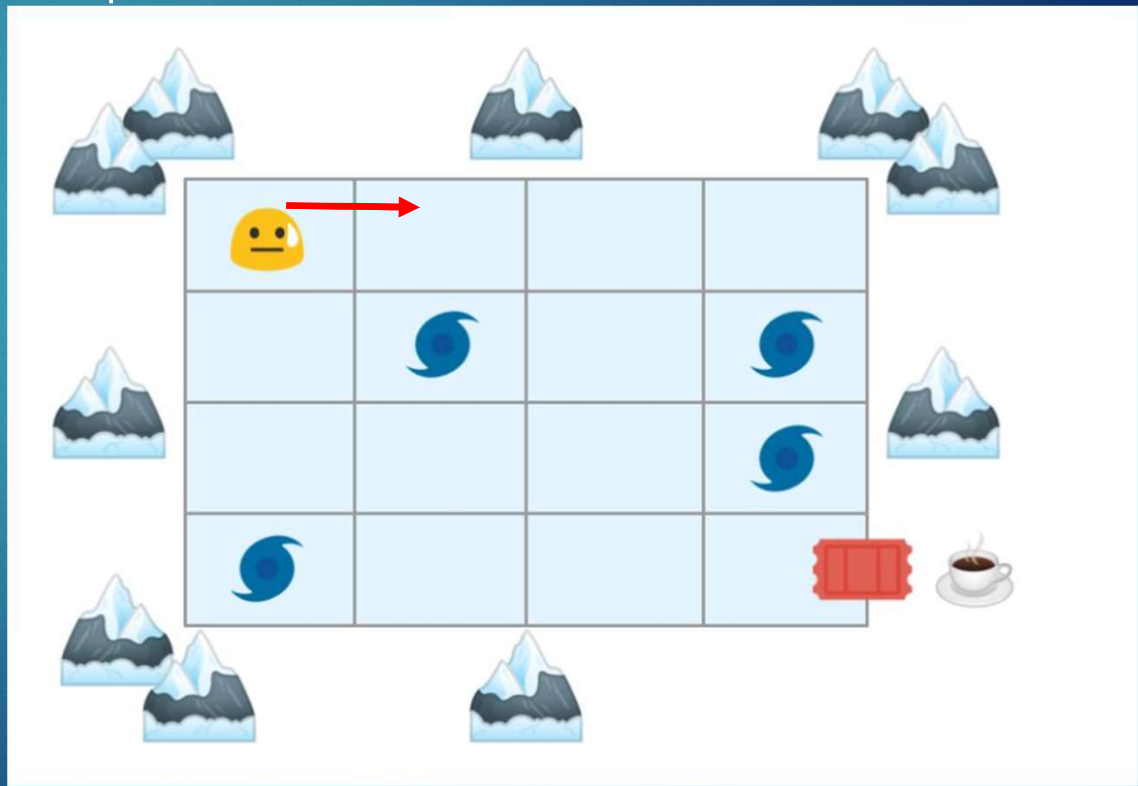
More general:
"probabilities of selecting each possible action"

A mapping from states to ~~actions~~

Example 1:
"always attend lectures"

Example 2:
"always move right"

Example 3:
"roll a 6-sided dice
and move right if get 6"

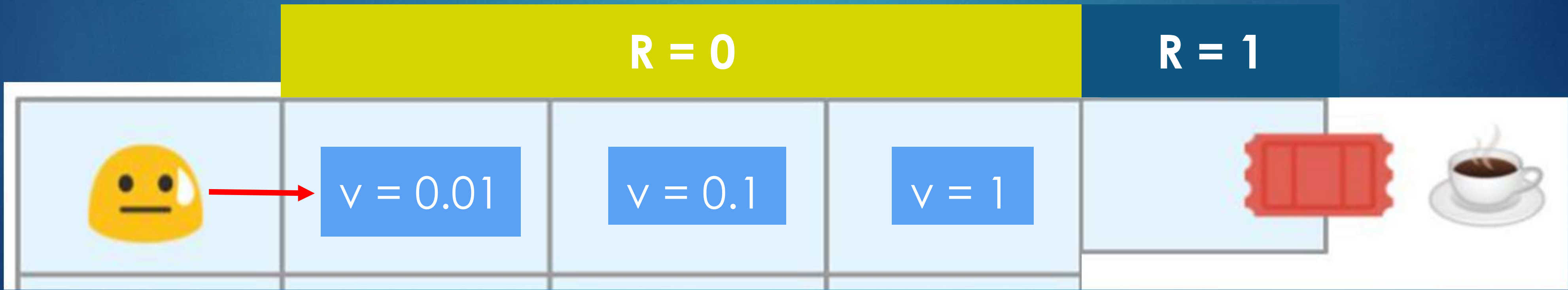


State-Value function

Expected return from **state**, **following a particular policy**.

Value of state s for policy π: $v_\pi(s)$

Example: policy "always move right", with discount factor = 0.1

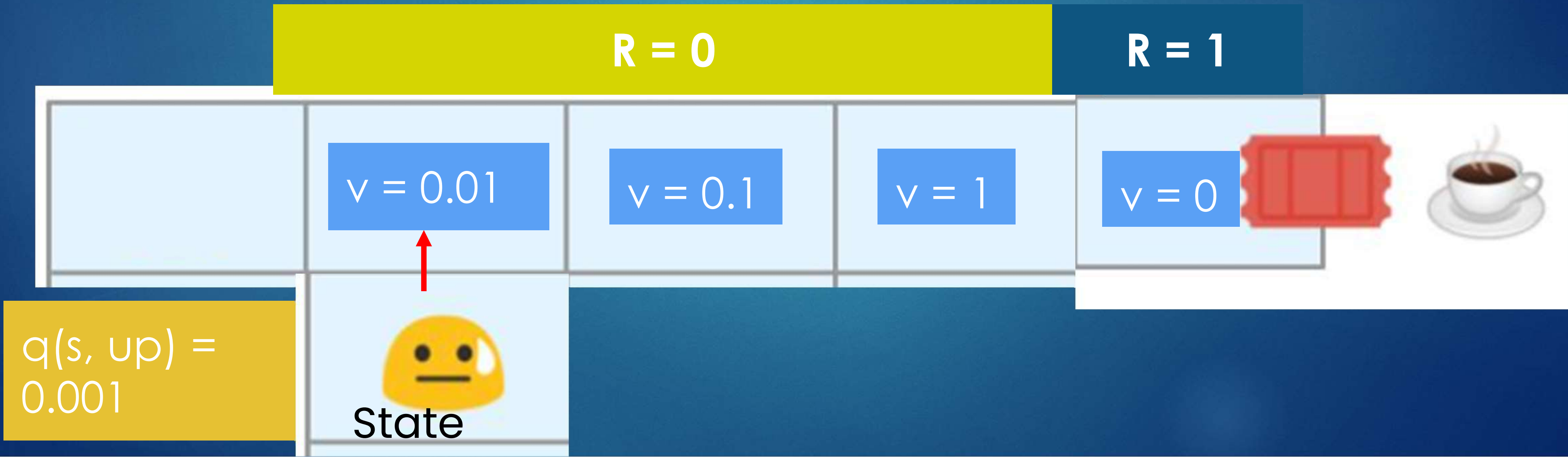


Action-Value function

Expected return from **state**, by **taking an action**, thereafter **following a particular policy**.

Action-Value of state s, action a, policy π: $q_\pi(s, a)$

Note that action may be arbitrary and doesn't have to be in line with the policy
Example: policy "always move right"



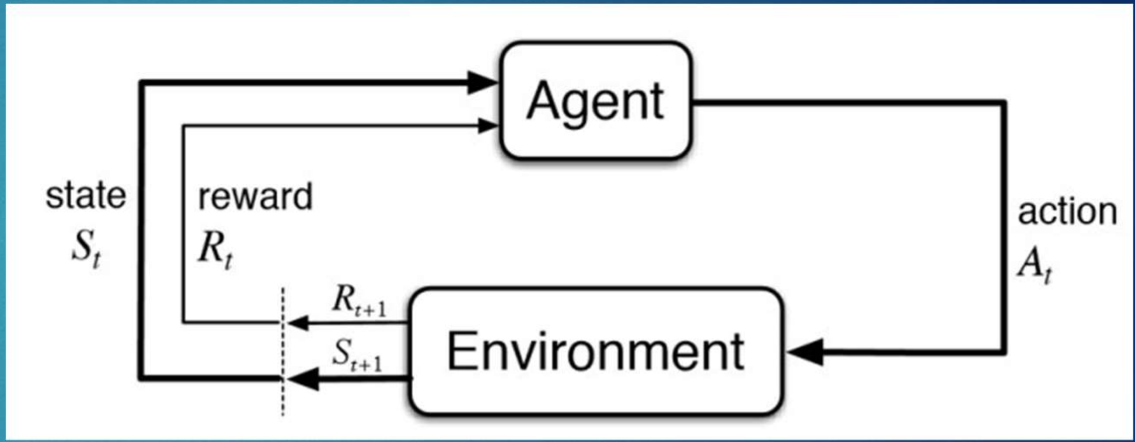
MDP summary

Return: discounted total reward

State value: expected return from state s

Action value: expected return by action a

Policy: action distribution for each state

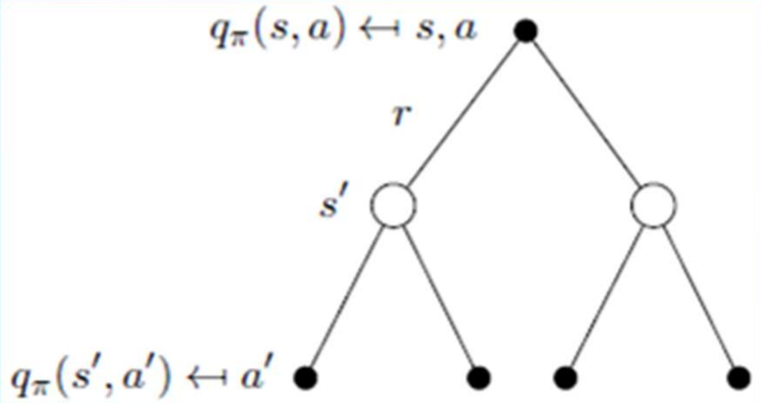


Bellman Equation

To calculate expected total discounted reward by a recursive function.

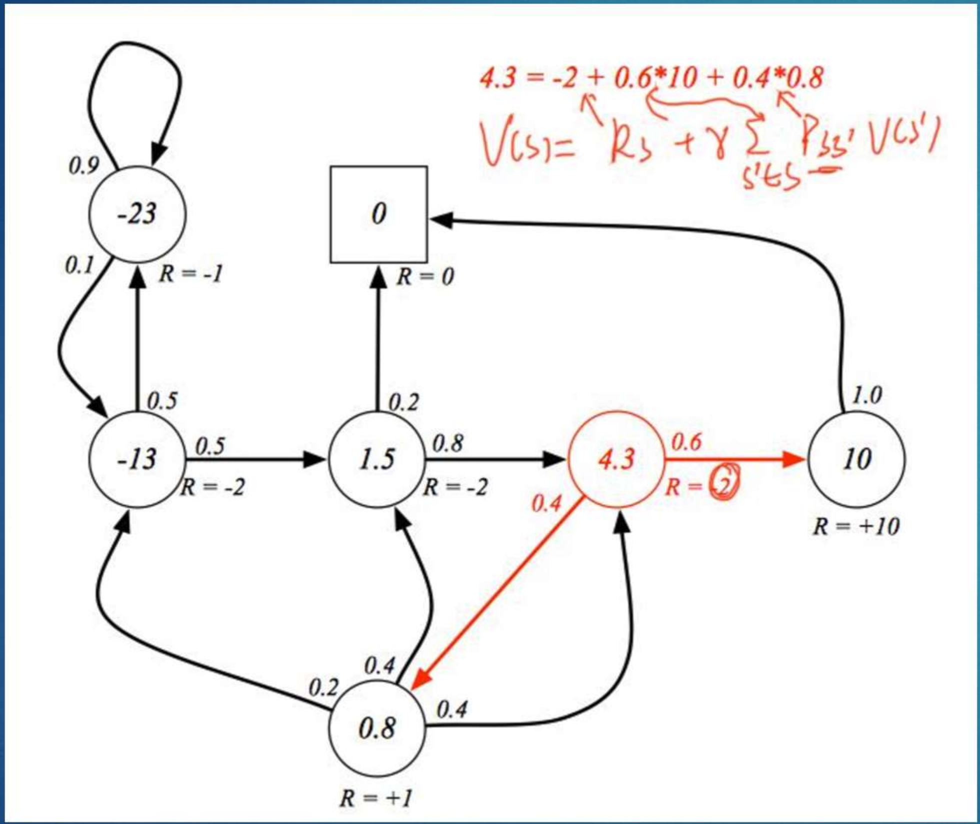
$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$
$$\quad = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$



The state-value function and action-value function can be decomposed into immediate reward plus discounted value of successor state

Bellman Equation Example



Exploitation and exploration

Motivation: find global optimal solution

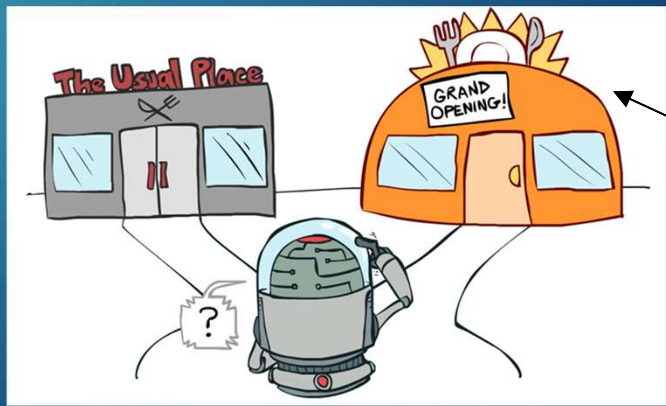
Greedy: always select best known (may miss global optimum)

Example: select a restaurant

Go to favourite restaurant or try a new restaurant

epsilon-greedy: with a small ratio, choose an action randomly rather than the currently optimal action

eps



1-eps

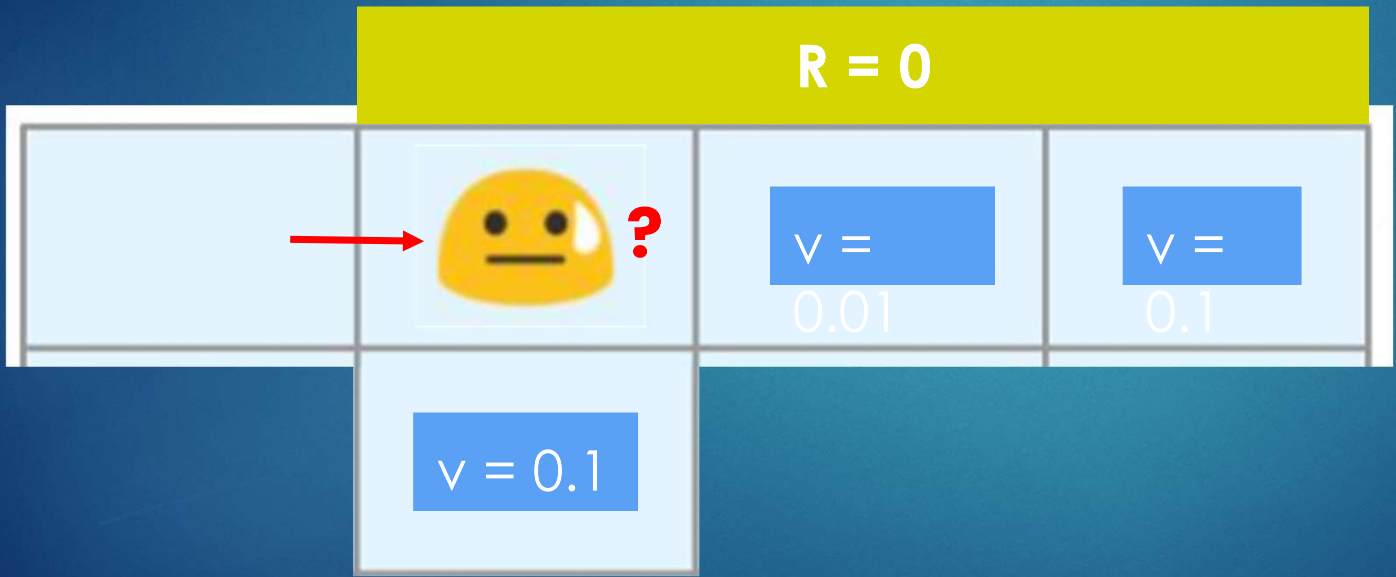
3

Algorithms

Classic techniques

How to improve policy?

Given: value function of a policy (always move right)



Policy Improvement Theorem

Eureka! 💡

If we “improve” action at one state s, with respect to v and keep everything else same, we get a better policy than π

Smaller Eureka:
The actual theorem is a bit more complicated.

Generalized Policy Iteration

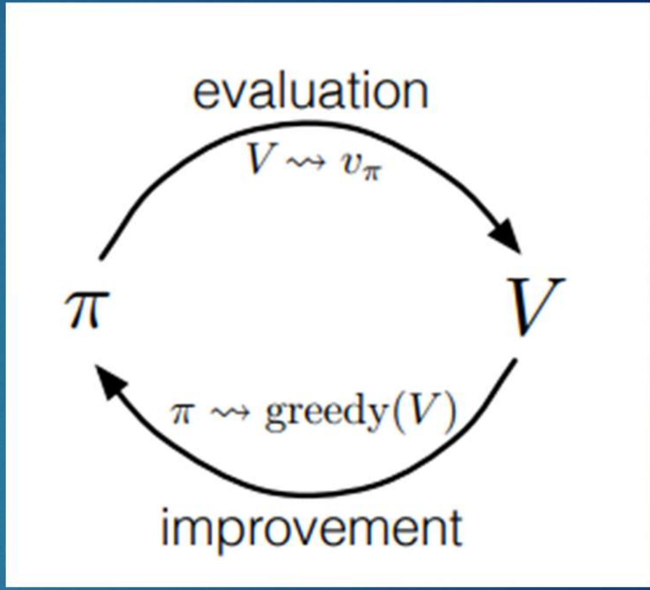
1. initialize policy

Loop:

2. Evaluate value function of policy

3. Improve policy by greedifying (to certain extent) with value function

Loop until policy is greedy with respect to its own value function



Dynamic Programming

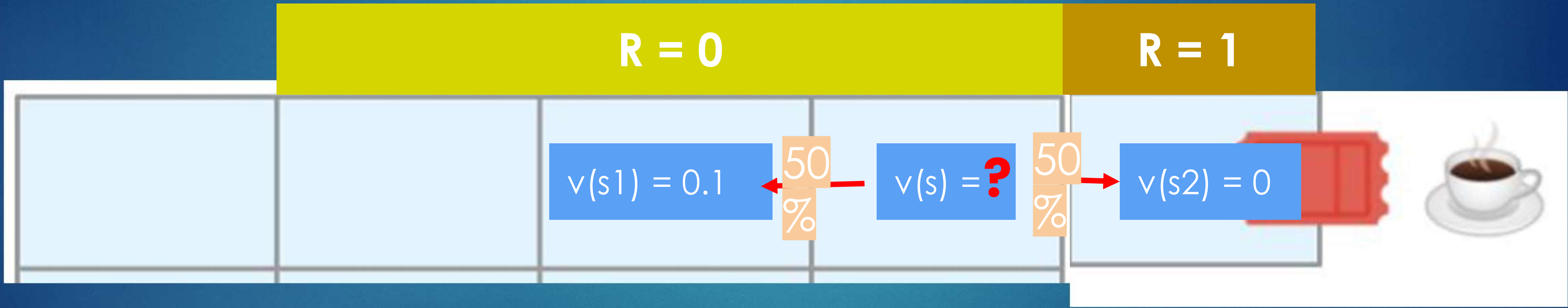
1) Assume a **complete accurate model** of the environment

If we choose to move right…

We know exactly **where** the gumdrop may end up,
with **what probability**

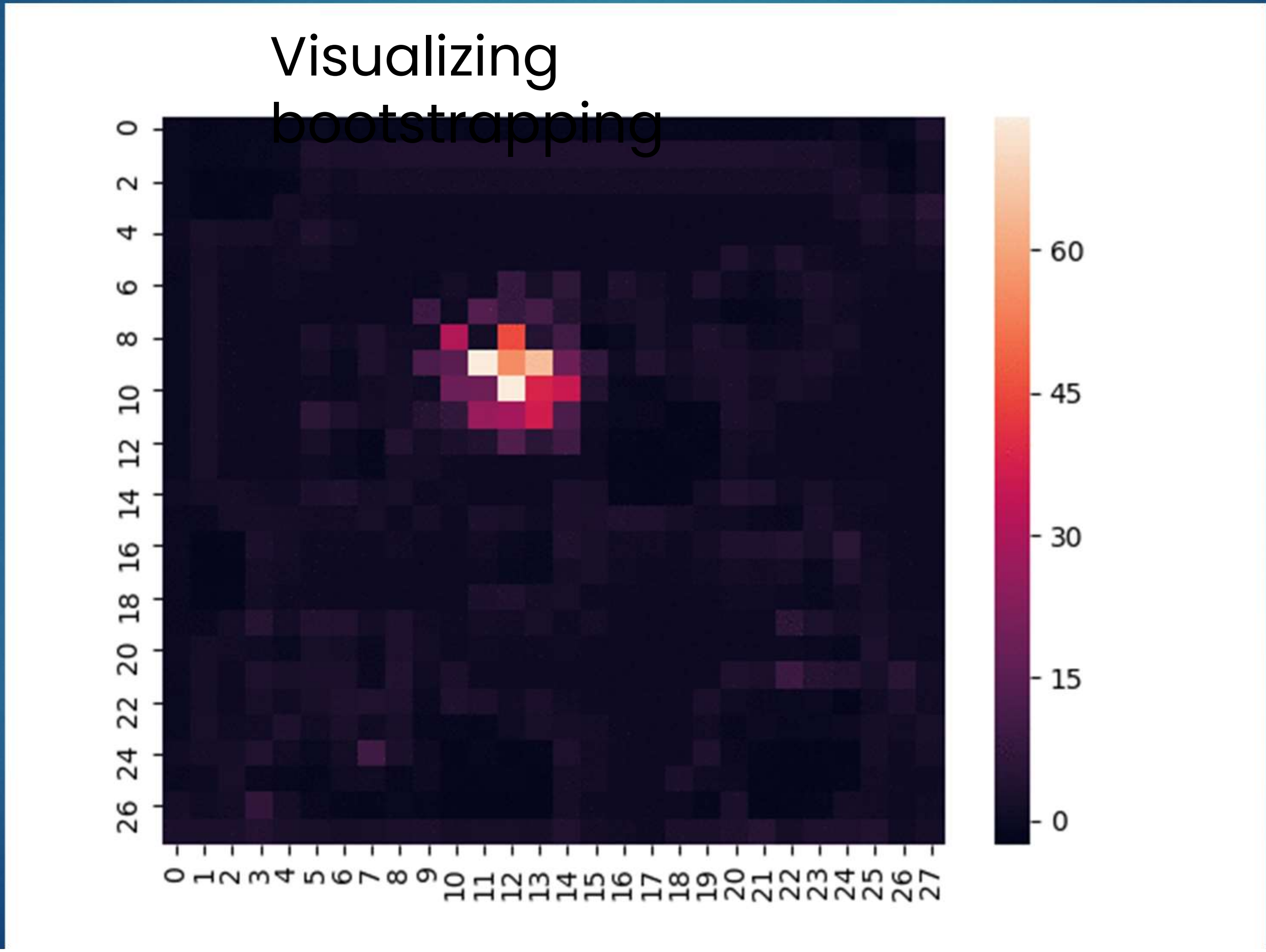
Dynamic Programming

1) Assume a **complete accurate model** of the environment
2) Update state values based on **estimated** values of other states: *bootstrapping*



$$v(s) = 50\% * (0 + v(s1)) + 50\% * (1 + v(s2))$$
$$= 0.55$$

Dynamic Programming



Dynamic Programming

Notable examples

1) Policy Iteration
2) Value Iteration

Policy Iteration - How?

1. initialize policy

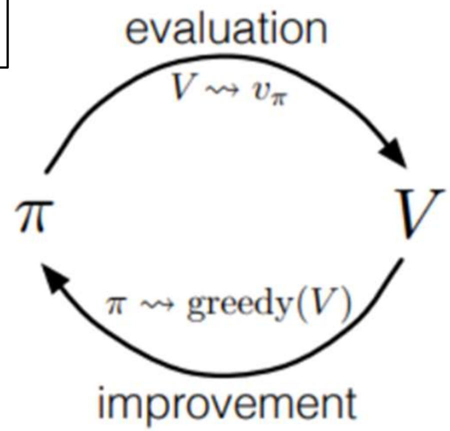
… by bootstrapping **till converge**

Loop:

2. Evaluate value function of policy

3. Improve policy by greedifying (to certain extent) with value function

Loop until policy is greedy with respect to its own value function



Policy Iteration: An Illustration

Iteration 1

Policy

→	→	→
→	→	
→	→	→

Expected Value

0	0	0
0.1	1	0
0	0	0

Iteration 2

Policy

↓	↓	→
→	→	
↑	↑	→

Expected Value

0.01	0.1	0
0.1	1	0
0.01	0.1	0

Value iteration: How?

Initialise state-value for each state

Loop until convergence:

For each state, update state-value as the highest action-value

Policy is greedy with respect to the converged state-value function

Value Iteration - An illustration

Initialise value function

0	0	0
0	0	0
0	0	0

Iteration 1

0	0	1
0	1	0
0	0	1

Iteration 2

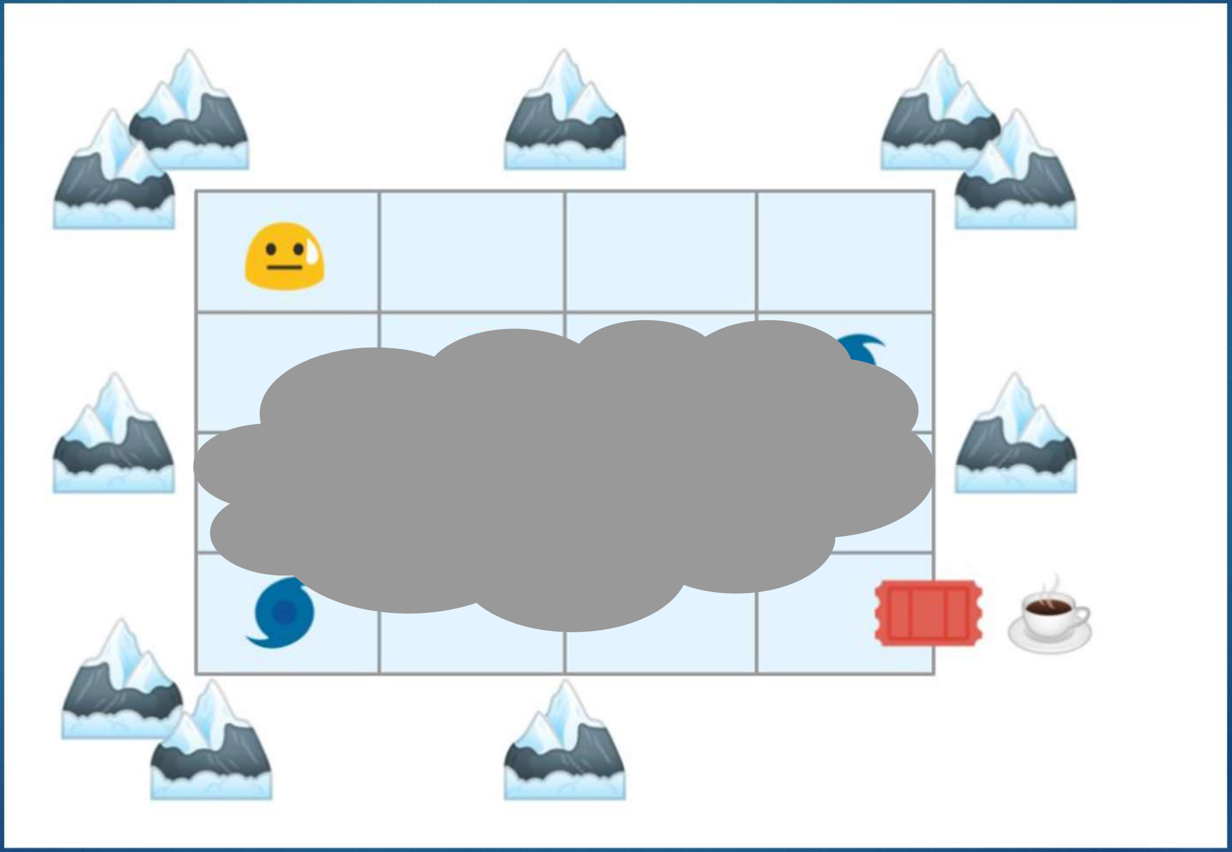
0	0.1	1
0.1	1	0
0	0.1	1

No more change after Iteration 3

↓	↓	↓
→	→	☕
↑	↑	↑

Limitations in Dynamic Programming

- Can't solve unknown environment

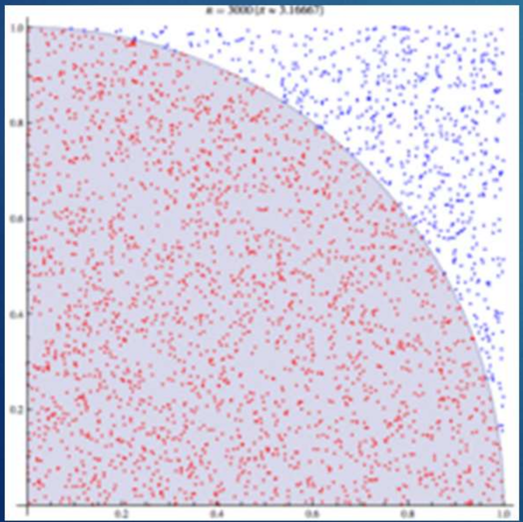


Monte Carlo

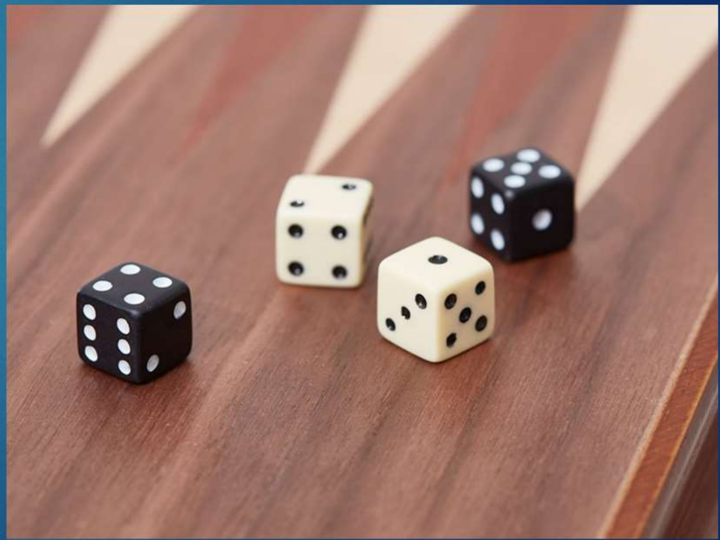
Differs from DP in how it evaluates value function

Learn value functions from experience

- Generate numerous experiences by following the policy
- Value of a state = average of the returns



Number of points	Approximate area (should be 3.14159...)
10	2.4
100	3.0
1000	3.152
10000	3.14
10000	3.14896
100000	3.14112



Q-Learning

Q learning is a value-based method of supplying information to inform which action an agent should take.

Directly approximate action-value function of **optimal policy**

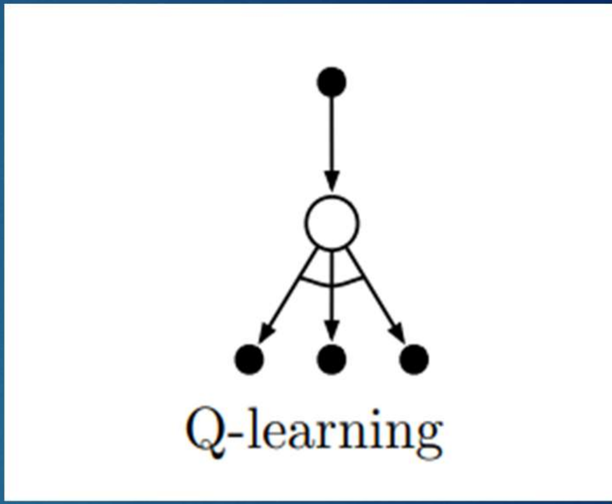
Over-simplified example:

q(today, studyRL) ← value of reward from this workshop +

+ value for studying "greedily" forever

q(today, play) ← value of reward from playing +

+ value for studying "greedily" forever



Q-Learning

Initialize all q values, store in table

Bellman equation for update

Current value

Reward

Maximum reward that can be obtained from state

$$Q(S,A) \leftarrow \boxed{Q(S,A)} + \alpha\left(\boxed{R} + \gamma \boxed{\max_{a'} Q(S',a')} - Q(S,A)\right)$$

Policy Gradient

Optimal action may not be deterministic.

Optimizing parametrized policies with respect to the expected return by gradient descent.

No value function required

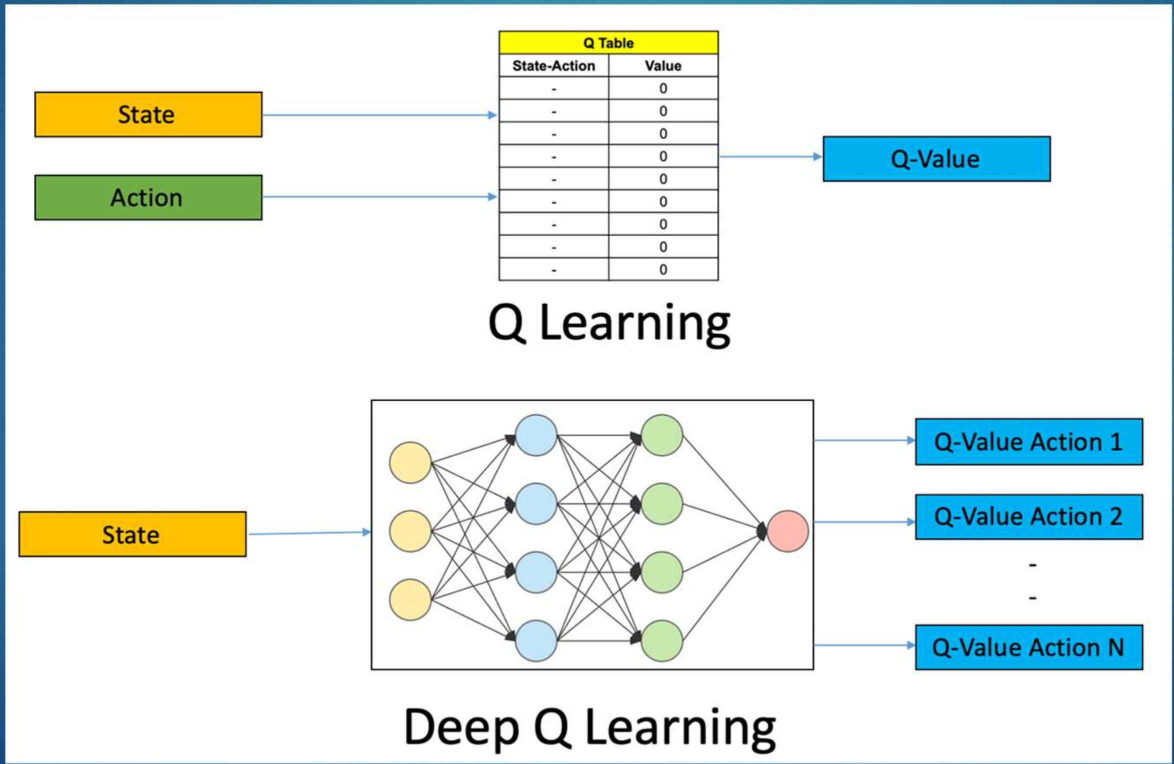
A deterministic policy can be easily exploited

Uniform random policy

Deep Reinforcement Learning

Environments can be too large for tabular representation

Environments and actions can be continuous



Break & some questions for you!

4 Hands-on

Link will be sent in Zoom chat

Question 1:

<u>What is NOT a characteristic of reinforcement learning:</u>

Option 1: It is a sequential decision making procedure.

Option 2: It is a branch of machine learning.

Option 3: Reward signals and feedbacks are always immediate.

Option 4: Time plays a crucial role in reinforcement learning.

Question 2:

<u>Which of the following is NOT true on Markov Decision Process(MDP)?</u>

Option 1: Agent is the learner and the decision maker.

Option 2: At each time step the agent takes an action.

Option 3: At each time step the environment generates a reward signal.

Option 4: All the past history states and actions are required to determine the next state.

Question 3:

Suppose discount factor γ=0.8, we observe the following sequence of rewards:

R1 = -3, R2 = 5, R3=2, R4 = 7, and R5 = 1, with T=5. What is the return G0? Hint: Work Backwards and recall that Gt = R{t+1} + γ*G{t+1}.

Option 1: 5.27

Option 2: 6.27

Option 3: 7.27

Option 4: 8.27

Question 3:

Which of the algorithms requires that the agent has complete knowledge of the environment?

Option 1: Value iteration.

Option 2: Monte Carlo.

Option 3: Q learning.

Option 4: Policy gradient.

Any questions for us?

Moving forward

- Going broad: learn other fundamental concepts and techniques
- Going deep: Deep Reinforcement Learning
- Going practical: train the models on some real games!

Where are the **links** to do all this? => They are in the feedback form 😆

Thanks!

Your feedback is extremely valuable to **reinforce our learning** as instructors, to create **better workshops for you** in the future! The **link to the additional materials** is available after submitting the feedback form.

https://forms.gle/GR4YnphhvEbmXoXv9

Credits

- https://docs.paperspace.com/machine-learning/wiki/supervised-unsupervised-and-reinforcement-learning
- https://raw.githubusercontent.com/FrancescoSaverioZuppichini/Value-Iteration-Network/master/core/gridworld_28x28/animation.gif
- Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
- https://www.davidsilver.uk/teaching/
- Mykel J. Kochenderfer (2021). Algorithms for Decision Making .MIT Press.
- https://dreager1.files.wordpress.com/2010/05/mario-vs-bowser.jpg
- https://i.insider.com/560ebbe7dd0895325c8b458e?width=1100&format=jpeg&auto=webp
- https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/