

Week 3

Reinforcement Learning

[Why use Reinforcement Learning?](#)

[Overview of how Reinforcement Learning Works](#)

[Return](#)

[Example](#)

[Goal of Reinforcement Learning - Policy](#)

[State-action Value Function](#)

[Q-learning function](#)

[Bellman Equation](#)

[Random Stochastic Environment](#)

[Continuous State Spaces](#)

[Discrete vs Continuous states](#)

[Deep Reinforcement Learning](#)

[Creating training set for Reinforcement Learning](#)

[Learning Algorithm](#)

[Algorithm Refinement](#)

[How to choose actions while still learning?](#)

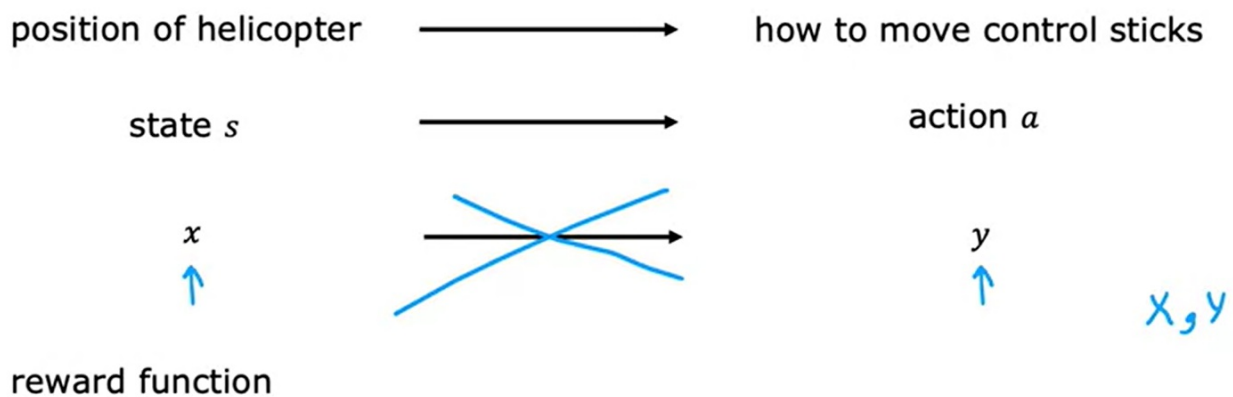
[Limitations of Reinforcement Learning](#)

Reinforcement Learning

Why use Reinforcement Learning?

- Supervised learning is not a good approach → Very difficult to get use a dataset to get an ideal action → Many ambiguities/nuances on what is the right action to take
- You don't need to know the exact actions to take → Robot will automatically learn based on the reward function

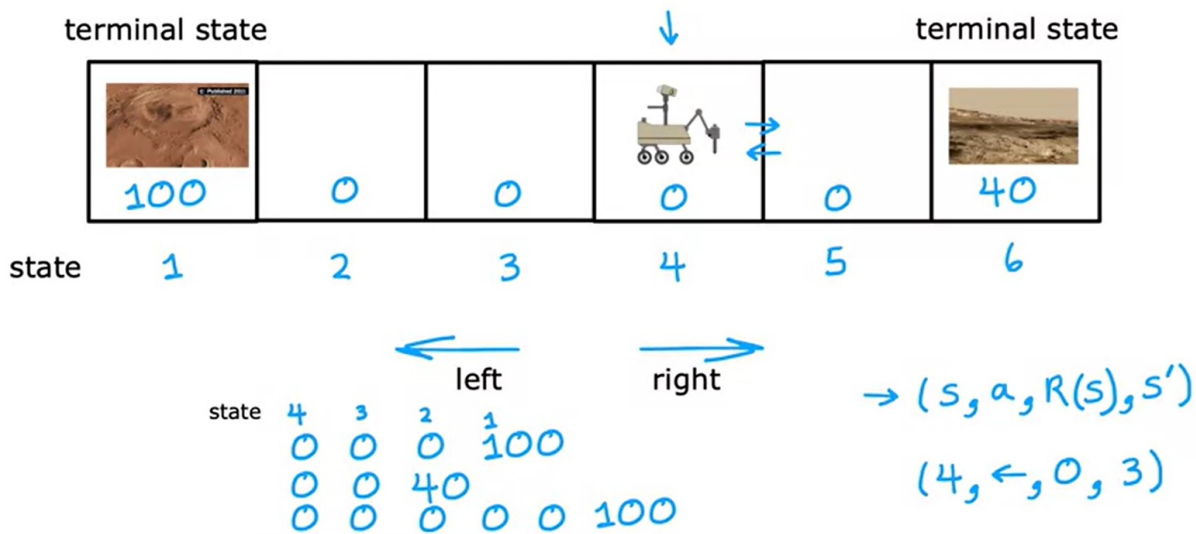
Reinforcement Learning



Overview of how Reinforcement Learning Works

- Robot will be aware of:
(state, action, reward, new state)

Mars Rover Example

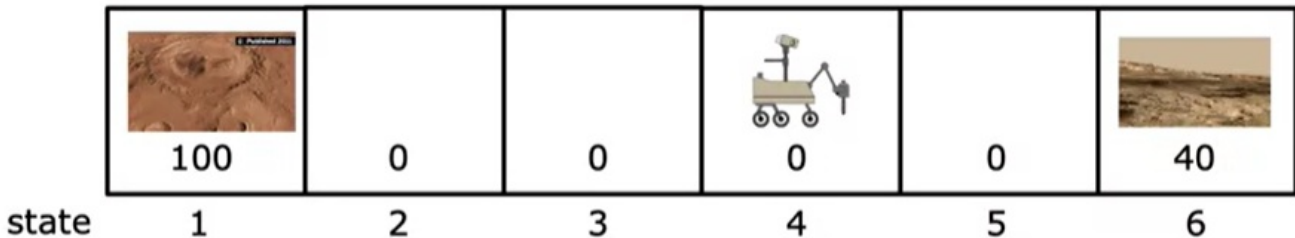


[Credit: Jagriti Agrawal, Emma Brunskill]

Return

- Idea: Penalize the additional states taken to reach the terminal state by adding a discount factor $(\gamma)(\gamma)(\gamma)$
- high γ = agent seeks long-term rewards

Return



$$\text{Return} = 0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 0.729 \times 100 = 72.9$$

$$\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \text{ (until terminal state)}$$

Discount Factor $\gamma = 0.9 \quad 0.99 \quad 0.999$
 $\gamma = 0.5$

$$\text{Return} = 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 = 12.5$$

Example

- Reward at each state (depending on the action) if discount factor γ is **0.5**
- Keep in mind that a state can have **negative** reward

Example of Return

100	50	25	12.5	6.25	40
100	0	0	0	0	40
1	2	3	4	5	6

← return

$$\gamma = 0.5$$

← reward

The return depends on the actions you take.

100	2.5	5	10	20	40
100	0	0	0	0	40
1	2	3	4	5	6

$$0 + (0.5)0 + (0.5)^2 40 = 10$$


100	50	25	12.5	20	40
100	0	0	0	0	40
1	2	3	4	5	6

$$0 + (0.5)40 = 20$$

Goal of Reinforcement Learning - Policy

- Create a policy that maps a state to an action

The goal of reinforcement learning

100					40
-----	--	--	---	--	----

Find a policy π that tells you what action ($a = \pi(s)$) to take in every state (s) so as to maximize the return.

State-action Value Function

Q-learning function

- $Q(s,a)$ represents the **maximum potential reward** possible if a particular action a is taken at state s , assuming that steps taken after are **optimal** (which depends on the policy).

Picking actions

→	100	50	25	12.5	20	40	← return
	100	0	0	0	0	40	← action
		←	←	←	→		← reward

→	100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
	100		0		0		0		0		40	
	1	2	3	4	5	6						

$Q(4, \leftarrow) = 12.5$ $Q(4, \rightarrow) = 10$

$$\max_a Q(s, a)$$

$$\pi(s) = a$$

$Q(s, a)$ = Return if you

- start in state s .
- take action a (once).
- then behave optimally after that.

The best possible return from state s is $\max_a Q(s, a)$.

The best possible action in state s is the action a that gives $\max_a Q(s, a)$.

Bellman Equation

Notations:

$Q(s, a)$ = Return if you

- start in state s .
- take action a (once).
- then behave optimally after that.



$$R(1)=100 \quad R(2)=0$$

s : current state
 a : current action

$R(s)$ = reward of current state

s' : state you get to after taking action a
 a' : action that you take in state s'

- Bellman equation is a recursive function
- $\max_a Q(s', a)$ represents the return from optimal behavior from s'

Explanation of Bellman Equation

$Q(s, a)$ = Return if you

- start in state s .
- take action a (once).
- then behave optimally after that.

$s \rightarrow s'$

→ The best possible return from state s' is $\max_{a'} Q(s', a')$

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

Reward you get right away Return from behaving optimally starting from state s' .

$$Q(s, a) = R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \dots]$$

Random Stochastic Environment

- Misstep probability → The next state can be different from the optimal based on probability
- Maximize average expected return

Expected Return

Goal of Reinforcement Learning:

Choose a policy $\pi(s) = a$ that will tell us what action a to take in state s so as to maximize the expected return.

Bellman Equation:

$$Q(s, a) = R(s) + \gamma E[\max_{a'} Q(s', a')]$$

(Handwritten annotations: an arrow points from '3' to $R(s)$, and an arrow points from '2 or 4' to $\max_{a'} Q(s', a')$)

Continuous State Spaces

Discrete vs Continuous states

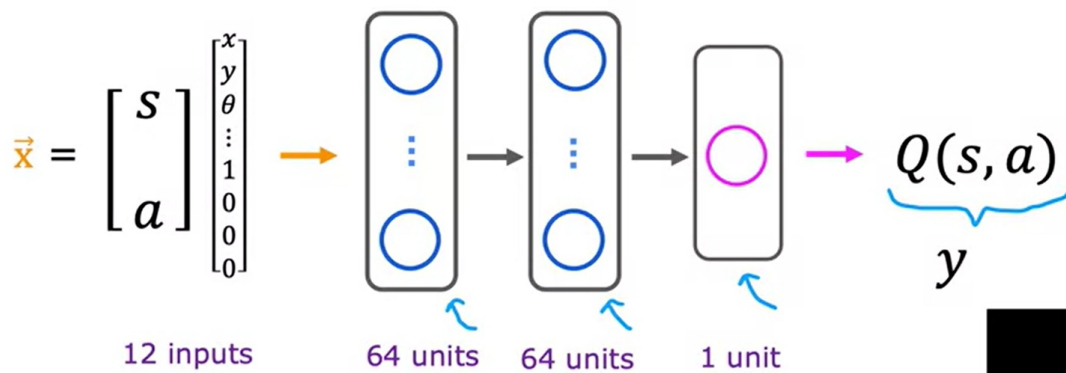
- Instead of simply moving from state 1 - 6, think about helicopters moving x, y, z , rotation, which is continuous

Deep Reinforcement Learning

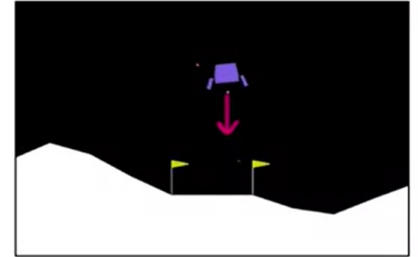
- In the example below, the 12 inputs come from:
 - 8 states
 - 4 q-learning functions (1 if the function is used, 0 otherwise) → actions

- How do we train a neural network to output $Q(s,a)$? Use Bellman Equation to create a training set of x and y .

Deep Reinforcement Learning



In a state s , use neural network to compute
 $Q(s, \text{nothing}), Q(s, \text{left}), Q(s, \text{main}), Q(s, \text{right})$
 Pick the action a that maximizes $Q(s, a)$



Creating training set for Reinforcement Learning

- Note that $s'1s'^11$ is not fixed → Q is from a random guess

Bellman Equation

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

x (under s, a) y (under $R(s)$)

$$f_{w,b}(x) \approx y$$

$$(s, a, R(s), s')$$

$$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)}) \leftarrow$$

$$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)}) \leftarrow$$

$$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)}) \leftarrow$$

$$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$$

$$y^{(2)} = R(s^{(2)}) + \gamma \max_{a'} Q(s'^{(2)}, a')$$

x	y
$x^{(1)} = (s^{(1)}, a^{(1)})$	$y^{(1)}$
$x^{(2)} = (s^{(2)}, a^{(2)})$	$y^{(2)}$
$x^{(10,000)}$	$y^{(10,000)}$

Learning Algorithm

- As you run this algorithm for many iterations, the accuracy will improve

Learning Algorithm

Initialize neural network randomly as guess of $Q(s, a)$.

Repeat {

Take actions in the lunar lander. Get $(s, a, R(s), s')$.

Store 10,000 most recent $(s, a, R(s), s')$ tuples.

Replay Buffer

Train neural network:

Create training set of 10,000 examples using

$$x = (s, a) \text{ and } y = R(s) + \gamma \max_{a'} Q(s', a')$$

Train Q_{new} such that $Q_{new}(s, a) \approx y$.

Set $Q = Q_{new}$.

$$f_{w, B}(x) \approx y$$

$$x, y$$

$$x', y'$$

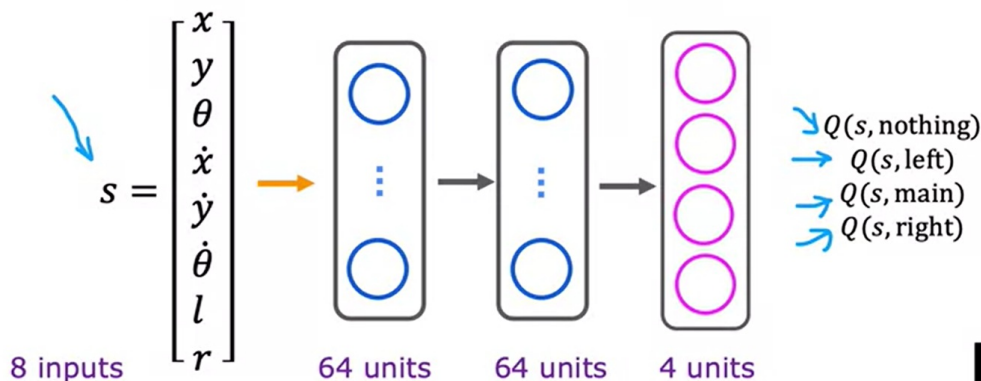
$$\vdots$$

$$x^{10000}, y^{10000}$$

Algorithm Refinement

- Inefficient → The original requires you to carry out inference 4 times in the neural network for some state
- Train neural network to output all 4 values simultaneously
 - 8 inputs only (for each state)
 - Given a state s , we only need to run the inference once!
 - Bellman Equation is also more efficient due to the 4 Q functions outputted in each inference

Deep Reinforcement Learning



In a state s , input s to neural network.

Pick the action a that maximizes $Q(s, a)$. $R(s) + \gamma \max_{a'} Q(s', a')$

How to choose actions while still learning?

- Option 1 can be bad as it can be influenced easily by the random $Q()$
 - E.g $Q(s, \text{main})$ generated could be low, and using option 1, main thrusters will never be used
- Hence an exploration step is needed to find an alternate $Q()$ which can sometimes be a good option to take
- ϵ -greedy policy ($\epsilon = 0.05$)

How to choose actions while still learning?

In some state s

Option 1:

Pick the action a that maximizes $Q(s, a)$.

Option 2:

- With probability 0.95, pick the action a that maximizes $Q(s, a)$. Greedy, "Exploitation"
- With probability 0.05, pick an action a randomly. "Exploration"

ϵ -greedy policy ($\epsilon = 0.05$)
0.95

$Q(s, \text{main})$ is low
↑
 a

Start ϵ high
 $1.0 \rightarrow 0.01$
Gradually decrease

Limitations of Reinforcement Learning

Limitations of Reinforcement Learning

- Much easier to get to work in a simulation than a real robot!
- Far fewer applications than supervised and unsupervised learning.
- But ... exciting research direction with potential for future applications.