

Reinforcement Learning

[Machine Learning - Coursera](#)

Position is given -> How to move the control sticks?

state s -> action a

Supervised Learning?

$x \rightarrow y$

for x 's we will find y 's from our knowledge?

it is too ambiguous for it, flying is unpredictable.

how to make a dog behave? Good dog! Bad dog!

you have to tell it what to do, rather than how to do it.

reward/punishment

Robotic Dog Example

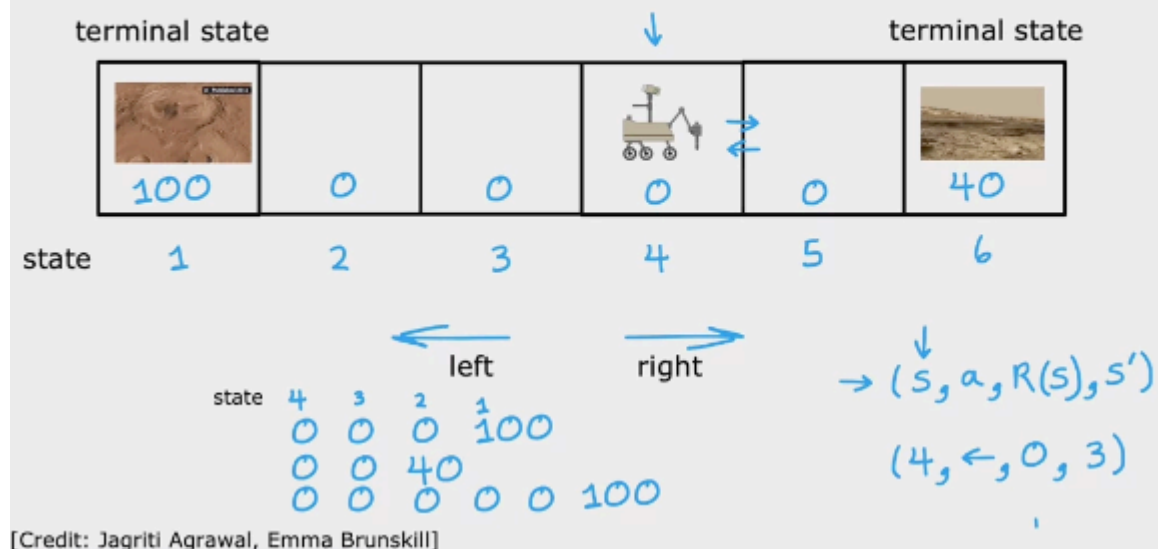


[Thanks to Zico Kolter]

Financial trading?

terminal state when gets to the one of the states, day ends.

Mars Rover Example



The Return in Reinforcement Learning

discount factor of $r = (ex) 0.9$

Return = $R + r.R + r^2.R + \dots$

Finance: Interest rate, time value of money.

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

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




Policies in reinforcement learning

can choose always go *direction*

state \rightarrow (policy) \rightarrow action

goal of RL: find a policy (π) that tells you what action to take in every state so as to maximize the return.

states, actions, rewards, discount factor, return, policy(π)

	Mars rover 	Helicopter 	Chess 						
states	6 states	position of helicopter	pieces on board						
actions	\longleftrightarrow	how to move control stick	possible move						
rewards	100, 0, 40	+1, -1000	+1, 0, -1						
discount factor γ	0.5	0.99	0.995						
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$						
policy π	<table border="1"><tr><td>100</td><td>\leftarrow</td><td>\leftarrow</td><td>\leftarrow</td><td>\rightarrow</td><td>40</td></tr></table>	100	\leftarrow	\leftarrow	\leftarrow	\rightarrow	40	Find $\pi(s) = a$ $\uparrow \quad \uparrow$	Find $\pi(s) = a$ $\uparrow \quad \uparrow$
100	\leftarrow	\leftarrow	\leftarrow	\rightarrow	40				

Markov Decision Process (MDP)

future depends on where you are now, not on how you got here.

State Action Value Function (Q-function)

Picking actions

→

100	50	25	12.5	20	40
100	0	0	0	0	40

→

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

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

← return
← action
← reward

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

$$\pi(s) = a$$

$Q(s, a)$ = 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:

Bellman Equation

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

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



$R(1)=100$ $R(2)=0$... $R(6)=40$

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

$R(s)$ = reward of current state

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

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' .

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

Discrete vs Continuous State

how would you represent the position of the helicopter/truck/...

pos, pitch, angle. with a vector:

Autonomous Helicopter



$$s = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \omega \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}$$

LUNAR LANDER

actions:

do nothing

left thruster

main thruster

right thruster

s=

[

x

y

velocity

angle

angular velocity

/ sitting on ground?

r sitting on ...?

]

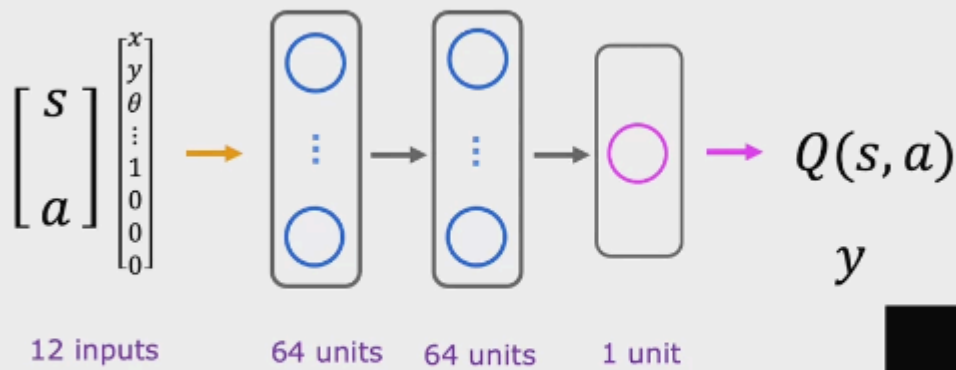
custom reward function:

crush -100

soft land +100

leg grounded +, fire main engine ---, side thrust -, get to landing pad ++

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 $Q(s, a)$



Create a supervised learning data for the NN output (Learning the $Q(s, a)$ function)

Bellman Equation

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

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

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

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

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

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

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

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

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

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 $Q = Q_{new}$.

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

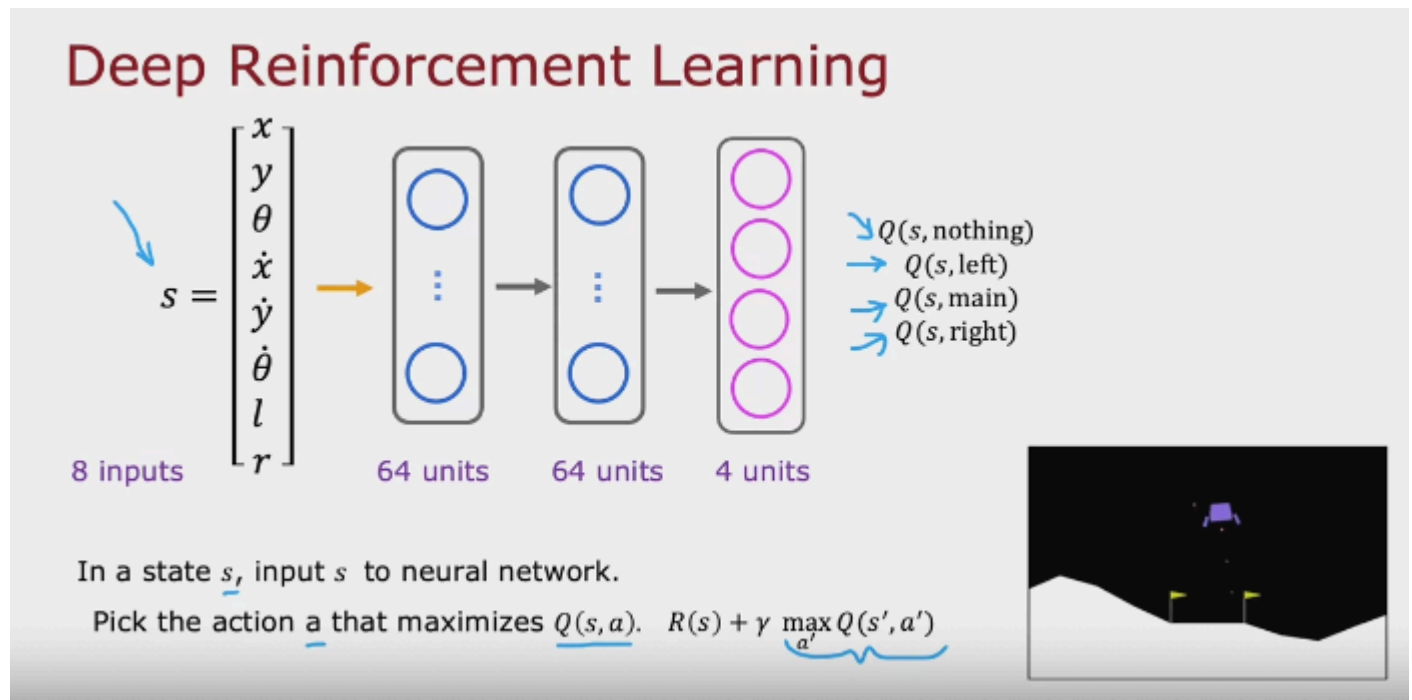


$$x, y$$

$$x^{(1)}, y^{(1)}$$

$$\vdots$$

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



Algorithm refinement: ϵ -greedy policy

How to choose actions while we are still learning?

at some probability, pick an action a randomly (%5)

epsilon greedy, epsilon = 0.05

Some strategies may never be tried by the NN in some cases, (stuck by chance)

start ϵ high and gradually decrease (we know more, possibility of being stuck is lower.)

Algorithm refinement: Mini-batch and soft updates

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 model:

Create training set of 1,000 examples using

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

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

Set $Q = Q_{new}$.

$x^{(1)}, y^{(1)}$

\vdots

$x^{(1000)}, y^{(1000)}$

Limitations

much easier in simulations rather than real world scenarios.

fewer applications compared to (un)supervised learning.

potential for future.