eg: Autonomous Helicopter

state s ⟶ action a

if supervised learning

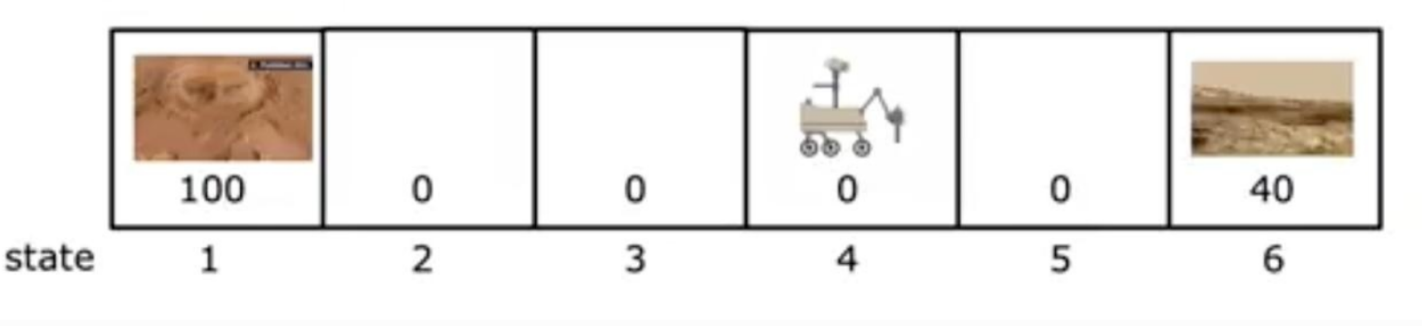
x - if this state ⟶ y

~~⟶~~ difficult to get states

Applications;

Controlling robots
Factory optimization
Financial (stock) trading
Games

eg:- Mars Rover Example;



left ⟵ ⟶ right

state

4	3	2	1	4	5	6
0	0	0	100	0	0	40

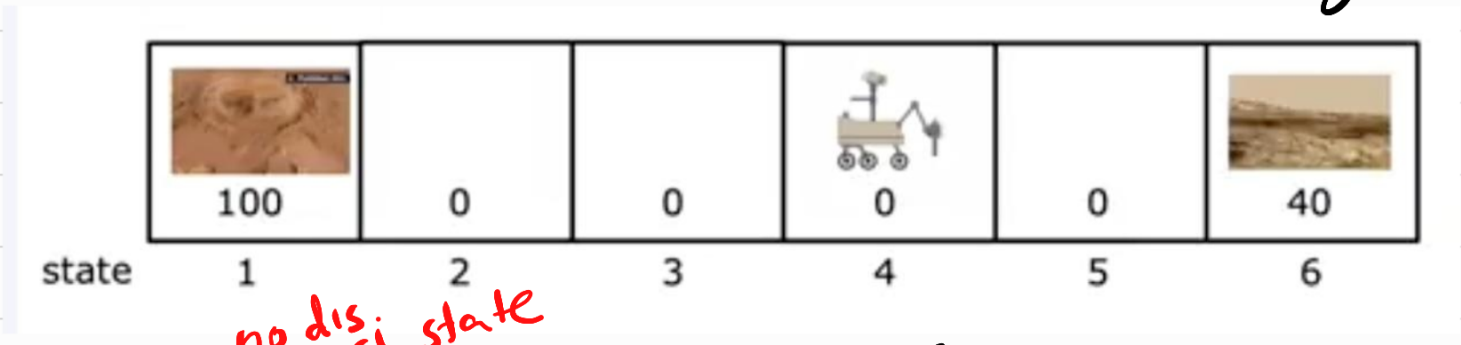
$(S, a, R(s), S')$

s or

⟵ ⟶

4	5	4	3	2	1
0	0	0	0	0	100

Return in Reinforcement Learning?



no dis. first state

Return = $0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 72.9$

discount factor γ

Return = $R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots$ (until terminal state)

(mostly γ close to 1)

if always left dis. factor = 0.5

100	50	25	12.5	6.25	40
100	0	0	0	0	40

if always right: 100 → 2.5 → 5 → 10 → 20 → 40

Both sides considered

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

Policies

State s —policy π→ action a

$\pi(2) = \leftarrow$
$\pi(3) = \leftarrow$
$\pi(4) = \rightarrow$

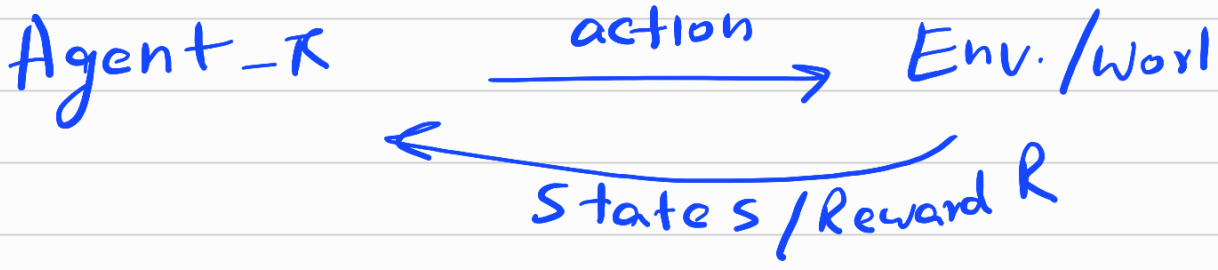
$\pi(s) \rightarrow a$

find policy to maxmize return.

	Mars rover 	Helicopter 	Chess 
states	6 states	position of helicopter	pieces on board
actions	← →	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	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$
policy π		Find $\pi(s) = a$	Find $\pi(s) = a$

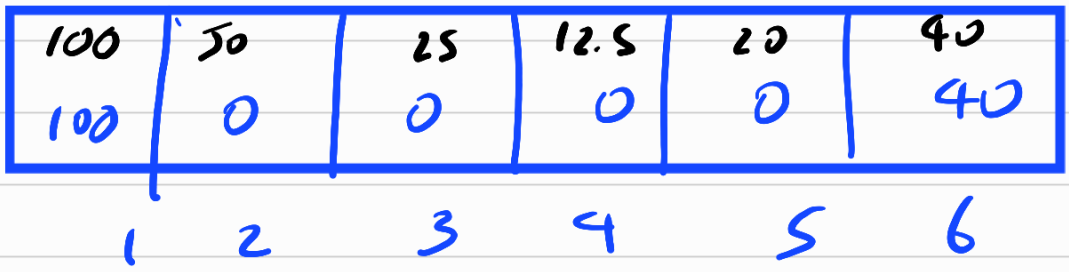
Markov Decision Process (MDP)

Future depends on current state now
(not how you got here)



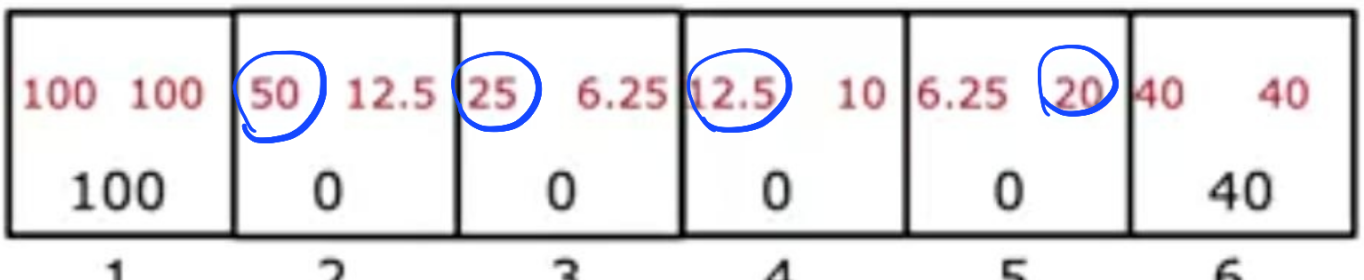
State Action Value function;
(Q-Function)

$Q(s,a)$ = Return if start state s
take action(a) once
optimal after that



$Q(4, \leftarrow) = 0 + (0.5) + 0 + 100\,(0.5)^3$

$= 12.5$



$Q(4, \leftarrow) > Q(4, \rightarrow)$

Bellman Equation

s = current state
a - current action
$R(s)$ = reward of current state
a' = action taken in s'

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

reward right away

best possible return from state

eg:-

$\gamma = 0.5$

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

$Q(2, \rightarrow)$

$s = 2$
$a = \rightarrow$
$s' = 3$

eg:-

$$Q(2, \rightarrow) = R(2) + 0.5 \max_{a'} Q(3, a')$$

$$= 0 + (0.5)(25)$$

$$= 12.5$$

② $Q(4, \leftarrow) = R(4) + 0.5 \max_{a'} Q(\cancel{4}, a')$

$= 0 + 0.5 \times (25)$

$= 12.5$

eg:- $= R(4) + 0.5 \left[0 + (0.5)0 + \underset{10^{2}}{(0.5)^{2}} \right]$

If Stochastic

Expected Return $\longrightarrow$ sequence of rewards

$= \text{Avg}\,(R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots)$

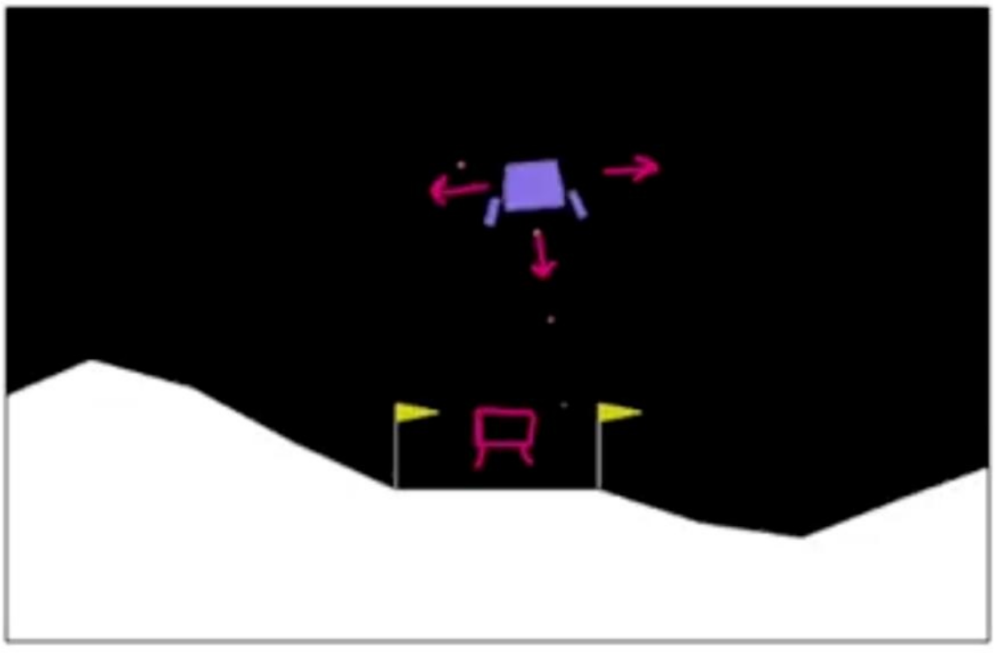
$= E\left[R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots \right]$

$$Q(s, a) = R(s) + \gamma \underset{a'}{E} \left[\max Q(s', a') \right]$$

2 & 4

for Continuos state space;

eg:- $s = \begin{vmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \omega \\ \dot{x} \end{vmatrix}$ ↓



actions:

- do nothing
- left thruster
- main thruster
- right thruster

$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$ ($\dot{x}$, $\dot{y}$: velocity; θ, $\dot{\theta}$: angle)

leg sitting on ground: left → 0 or 1 ← right

Reward Function

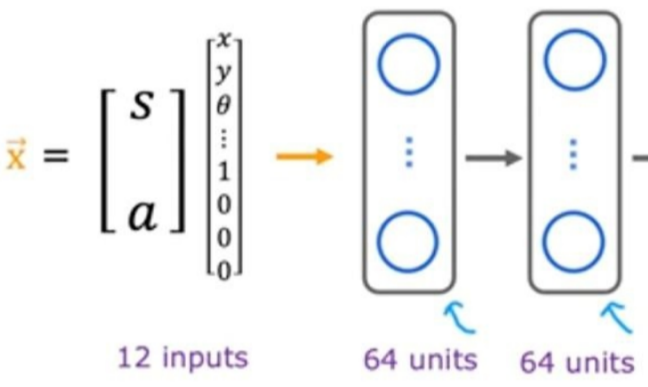
- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03

Learn Policy π given s;

pick action $a = \pi(s)$ to maximize return

$\gamma = 0.985$

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 $Q(s,a)$

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

x y

$f_{w,B}(x) \simeq y$ → using supervised learning

try different things &

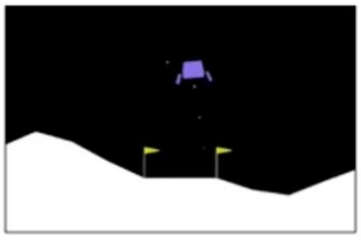
$(s^{(1)}, a^{(1)}, R(s)^{(1)}, s'^{(1)}) \rightarrow y^{(1)}$

$(s^{(2)}, \_ \_ \_ \_ \_) \rightarrow y^{(2)}$

DQN

Deep Queue Network

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

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

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

Set $Q = Q_{new}$.

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

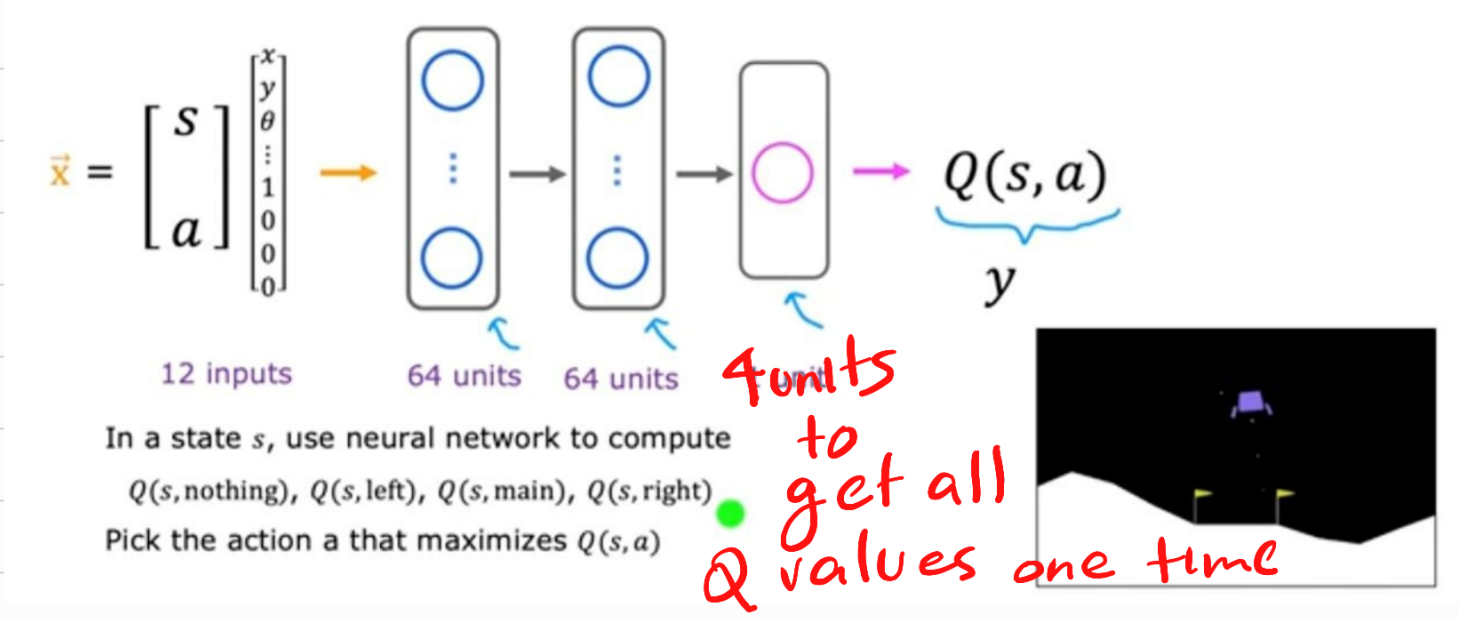
x, y $x^{(1)}, y^{(1)}$ ... x^{10000}, y^{10000}

If seperate 4 actions

→ have to run NN 4 seperate times

Output all Q values at once

only inference onetime



4 units to get all Q values one time

Q(s,nothing)

Q(s,left)

Q(s,right)

Q(s,main)

How to choose actions while still learning?

In some state s

Option 1:

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

Q(s,main) is low

↑

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 ($\varepsilon = 0.05$)

0.95

Start ε high

Gradually decrease

Mini batch and soft-updates;

making the examples

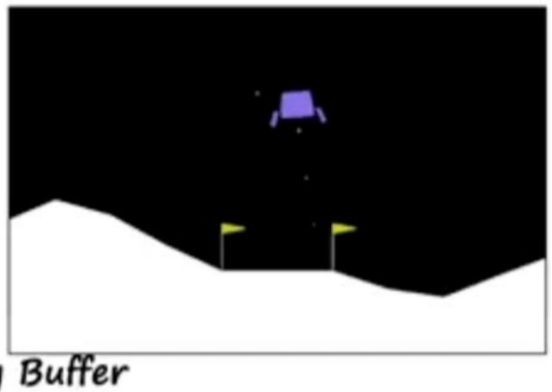
X	Y
2104	400
1416	232
- -	-
- -	- -

} mini batch 1
} mini batch 2

learning

} much faster

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:

1,000

Create training set of ~~10,000~~ 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 $Q = Q_{new}$.

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

$\vdots$

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

Soft Update ⟶ Causes R.L. algorithm to converge more likely

Set Q = Q new

W, B

Wnew, Bnew

$$W = 0.01\, W_{new} + 0.99\, W$$
$$B = 0.01\, B_{new} + 0.99\, B$$