Subject: / /

position of helicopter $\longrightarrow$ how to move control sticks

state s $\longrightarrow$ action a

Option 1: → NOT good

Use supervised learning

$x \longrightarrow y$

↳ Cannot know ideal action

Option 2:

Reinforcement learning → What to do NOT How

Reward function:

flying well $+1$

Crash -1000

Mars Rover Example:

Terminal state					Terminal state
+100 Interesting surface to analyze	0	0		0	+40 Interesting surface to analyze
State 1	2	3	4	5	6

state	4	3	2	1
	0	0	0	100

state	4	5	6
	0	0	40

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

↳ A new state

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

The Return of RL: → How to know if a particular reward is better than another.

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

$$= R_1 + \gamma R_2 + \gamma R_3 + \ldots \text{ until terminal state}$$

Subject:

A little bit less than 1

Discount factor $\gamma = 0.9$

- This way, getting reward sooner would result in higher return

Policy:

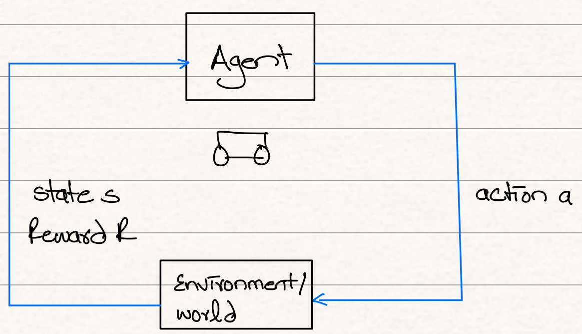
$$\underset{s}{\text{state}} \xrightarrow[\pi]{\text{Policy}} \underset{a}{\text{action}}$$

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

Markov Decision Process (MDP):

- states
- actions
- Rewards
- Discount factor γ
- Return
- policy π

- Future depends on where you are now NOT how you got here



Subject:

State action value function: (Q-function)

↳ How good is it?

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

- start in state s
- Take action a (once)
- Then behave optimally after that

Q(2,←) Q(2,→)

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

$\gamma = 0.5$

State	1	2	3	4	5	6
	+100 Interesting surface to analyze	50 ← 0	25 ← 0	12.5 ← 0	20 → 0	+40 Interesting surface to analyze

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

NOTE:
$Q = Q^*$ = Optimal Q function

Bellman equation:

s = current state
R(s) = Reward of current state
a = current action
s' = state you get after action a
a' = action you take in state s'

Subject: / /

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

Reward you get right away

Return from behaving optimally starting from state s'

Q(2,←) Q(2,→)

	1	2	3	4	5	6
Q(s,←)	100	50	25	12.5	6.25	40
Q(s,→)	100	12.5	6.25	10	20	40
R(s)	100	0	0	0	0	40

State 1 2 3 4 5 6

$\gamma = 0.5$

Example 1:

$s = 2$

$a = \rightarrow$

$s' = 3$

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

$$= 0 + (0.5)25 = 12.5$$

Example 2:

$s = 4$

$a = \leftarrow$

$s' = 3$

$$Q(4,\leftarrow) = R(4) + 0.5 \max Q(3,a')$$

$$= 0 + 0.5(25)$$

$$= 12.5$$

For terminal state,

$$Q(s,a) = R(s)$$

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

Random (stochastic) environment:

State	1	2	3	4	5	6
Action	←	←	←	←	→	
Reward	100	0	0	0	0	40

State 1 2 3 4 5 6

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

slips & goes to 4 instead

Subject:

Expected return = Return = Average$(R_1 + \gamma R_2 + \gamma^2 R_3 + \dots)$

$= E[R_1 + \gamma R_2 + \gamma^2 R_3 + \dots]$

Bellman eq:

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

Continuous state spaces:

For a truck:

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

$\dot{x}$ → How quickly is x changing

$\dot{y}$ → " y

$\dot{\theta}$ → " θ

Lunar Lander:

a = Do nothing
- left thruster ←
- Main thruster ↓
- Right thruster →

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

l, r } left or Right leg on ground (0 or 1)

Reward:

Land: 100 – 140

Crash: –100

Soft landing: +100

Leg grounded: +10

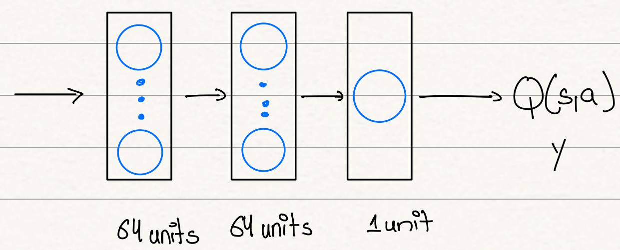
Fire main engine: –0.3

Fire side thrusters: –0.03

Subject:

Learning the state value function:

$$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



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

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

x	y

{create examples from simulation

$$(\underbrace{s, a}_{x}, \underbrace{R(s), s'}_{y})$$

$$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)})$$
$$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)})$$
$$\vdots$$
$$(s^{(m)}, a^{(m)}, R(s^{(m)}), s'^{(m)})$$

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

$$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$$
$$y^{(2)} = R(s^{(2)}) + \gamma \max_{a'} Q(s'^{(2)}, a')$$

- Initialize nueral network randomly as guess of $Q(s,a)$.
- Repeat {

  ? ⇒ ε greedy policy

  Take actions in lunar lander, get $(s, a, R(s), s')$

  Store the 10,000 more recent $(s, a, R(s), s')$ tuples

  ↳ Replay buffer

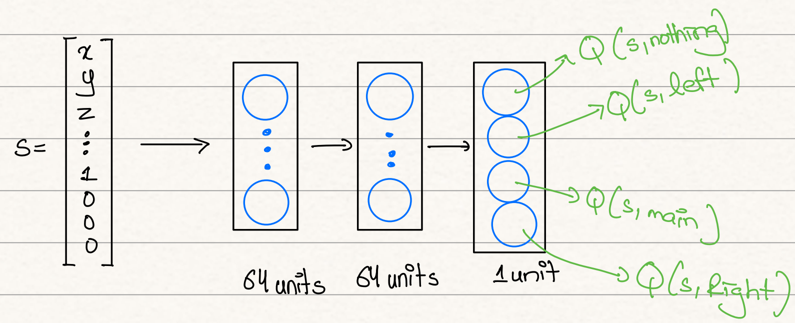
Subject:

Train nueral network:

- Create training set of 10,000 examples using
  $x = (s, a) \qquad y = R(s) + \gamma \max_{a'} Q(s', a')$
- Train Q_{new} such that $Q_{new}(s,a) \approx y$
- set $Q = Q_{new}$

}

Improved NN:



Eplison greedy policy:

How pick a while learning

Option 1:

pick a, that maxs $Q(s,a)$

Option 2:

with probability 0.95, pick a that maxs $Q(s,a)$ "Greedy, Exploitation"

with probability 0.05, pick a randomly "Exploration"

- This is better because if $Q(s, main)$ set to low in step 1 then, it will never learn to fire main thruster.

ϵ greedy policy ($\epsilon = 0.05$)

NOTE:
start ϵ high
Decrease gradually

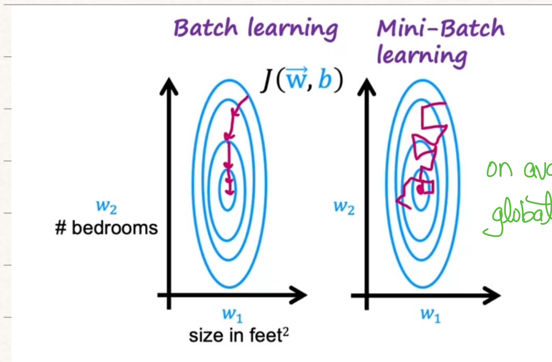
Subject:

Mini-batches & soft updates:

If m is very large, we use a mini batch.

x	y	
2104	400	mini batch 1
1416	232	
⋮	⋮	mini batch 2
3210	870	⋮

• Every iteration look at a portion of dataset



on avg, it will reach global minimum

Soft updates:

Set $Q = Q_{new}$

$w \quad B \qquad w_{new} \quad B_{new}$

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

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

99% old W

1% new B

Subject: / /

Algorithm 1: Deep Q-Learning with Experience Replay

Initialize memory buffer D with capacity N
Initialize Q-Network with random weights w
Initialize target $\hat{Q}$-Network with weights $w^{-} = w$
for episode $i = 1$ ***to*** M **do**
    Receive initial observation state S_1
    for $t = 1$ ***to*** T **do**
        Observe state S_t and choose action A_t using an ϵ-greedy policy
        Take action A_t in the environment, receive reward R_t and next state S_{t+1}
        Store experience tuple (S_t, A_t, R_t, S_{t+1}) in memory buffer D
        Every C steps perform a learning update:
        Sample random mini-batch of experience tuples (S_j, A_j, R_j, S_{j+1}) from D
        Set $y_j = R_j$ if episode terminates at step $j+1$, otherwise set $y_j = R_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a')$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; w))^2$ with respect to the Q-Network weights w
        Update the weights of the $\hat{Q}$-Network using a soft update
    end
end