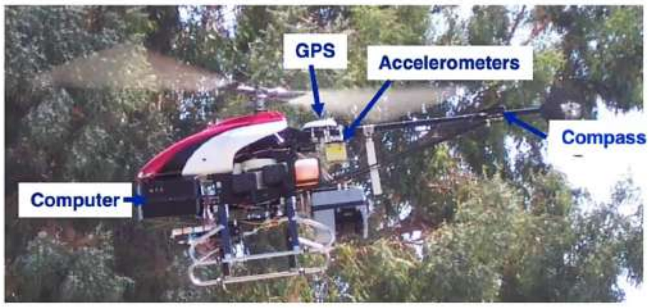
Course 3

Third week

What is Reinforcement Learning?

Autonomous Helicopter



How to fly it?

学倒飞

Autonomous Helicopter

Reinforcement Learning

position of helicopter ⟶ how to move control sticks

state s ⟶ action a

~~x ⟶ y~~

↑ ↑ x, y

reward function

positive reward : helicopter flying well $+1$

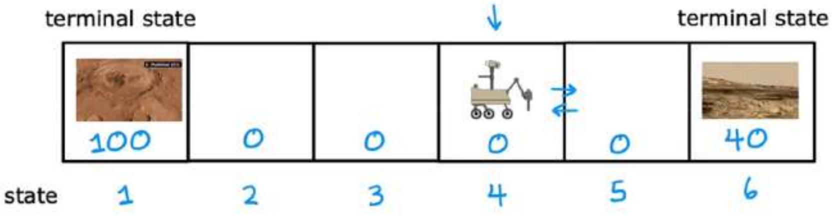
negative reward : helicopter flying poorly -1000

Applications

→ • Controlling robots
- Factory optimization
- ☆ Financial (stock) trading
- Playing games (including video games)

Mars rover example

Mars Rover Example



left right

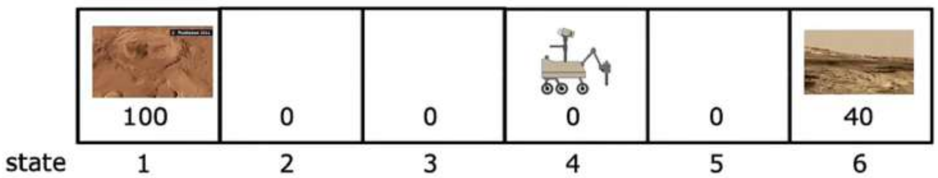
state 4 3 2 1
0 0 0 100
0 0 40
0 0 0 0 0 100

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

$(4, \leftarrow, 0, 3)$

The Return in Reinforcement Learning γ

Return



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

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

Discount Factor $\gamma = 0.9$ 0.99 0.999 → 远视

$\gamma = 0.5$

Return $= 0+(0.5)0+(0.5)^2 0+(0.5)^3 100 = 12.5$ → 短视

Example of Return

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

$\gamma = 0.5$

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

$\overset{4}{0} + (0.5)0 + (0.5)^2 40 = 10$

100	50 ←	25 ←	12.5 ←	20 →	40
100	0	0	0	0	40

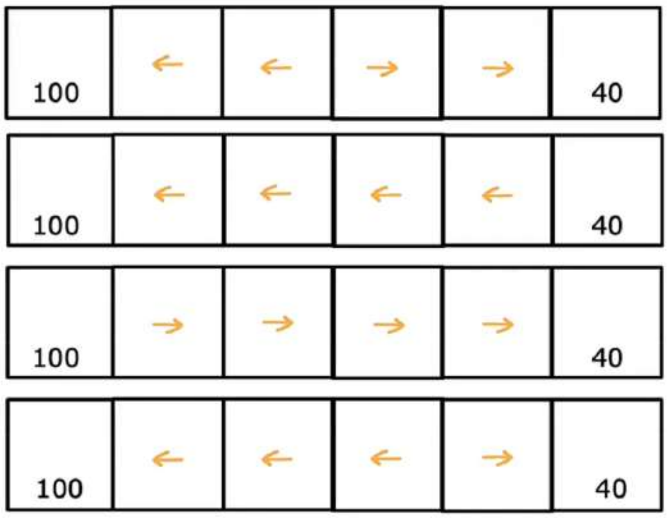
$0 + (0.5)40 = 20$

Making decisions: Policies in reinforcement Learning

Policy

state $\xrightarrow{\text{policy}}$ action

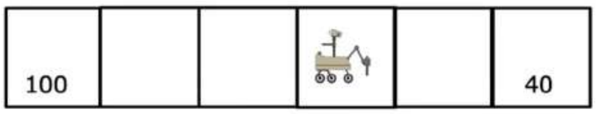
s $\quad\pi\quad$ a



$\pi(s) = a$
$\pi(2) = \leftarrow$
$\pi(3) = \leftarrow$
$\pi(4) = \leftarrow$
$\pi(5) = \rightarrow$

A policy is a function $\pi(s) = a$ mapping from states to actions, that tells you what action a to take in a given state s.

The goal of reinforcement learning

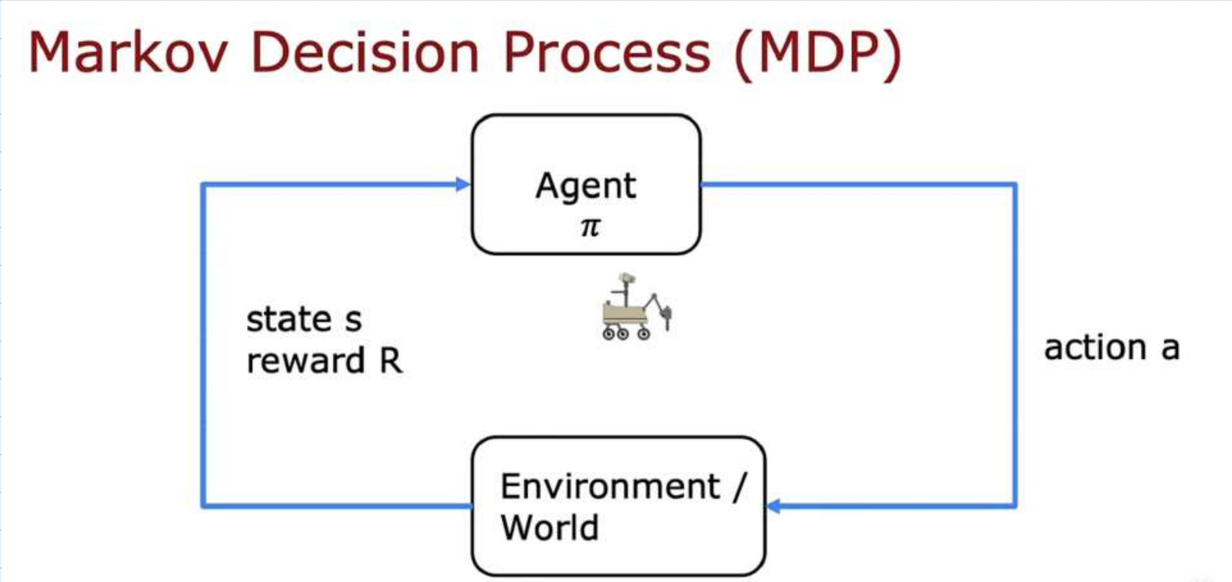


Find a policy π that tells you what action ($a = \pi(s)$) to take in every state (s) so as to maximize the return.

Review of key Concepts

	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	100 ← ← ← → 40	Find $\pi(s) = a$	Find $\pi(s) = a$

总结



State-action value function definition

State action value function

$Q(s,a)$ = Return if you
- start in state s.
- take action a (once).
- then behave optimally after that.

$Q(s,a)$

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

$Q(2,\leftarrow)$ $Q(2,\rightarrow)$

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

$Q(2,\rightarrow)=12.5$

$0+(0.5)0+(0.5)^2 0+(0.5)^3 100$

$Q(2,\leftarrow)=50$

$0+(0.5)100$

$Q(4,\leftarrow)=12.5$

$0+(0.5)0+(0.5)^2 0+(0.5)^3 100$

这通常也称为 Q 函数。

Picking actions

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

$Q(4,\leftarrow)$ $\underline{12.5}$ $Q(4,\rightarrow)$ 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 $\max_a Q(s,a)$.

Q^* Optimal Q function

State-action value function example

Optional Lab

see how γ and reward

change policy π $Q(s,a)$

Bellman Equation → help compute

Bellman Equation

$Q(s,a)$ = Return if you
- start in state s.
- take action a (once).
- then behave optimally after that.

1	2	3	4	5	6
			←		
$R(1)=100$	$R(2)=0$	...			$R(6)=40$

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

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

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

Bellman Equation

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

$Q(s,a)=R(s)$ → 在 terminal T.

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

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

$$Q(2,\rightarrow)=R(2)+0.5\max_{a'} Q(3,a')$$
$$=0+(0.5)25=12.5$$

$s=4$
$a=\leftarrow$
$s'=3$

$$Q(4,\leftarrow)=R(4)+0.5\max_{a'} Q(3,a')$$
$$=0+(0.5)25=12.5$$

$$Q(2,\rightarrow)=R(2)+\gamma \max_{a'} Q(3,a')$$
$$=0+0.5(25)$$

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 $\max_{a'} Q(s',a')$

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

Reward you get right away

Return from behaving optimally starting from state s'.

$$R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$$

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

Explanation of Bellman Equation

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

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

$Q(4,\leftarrow)$

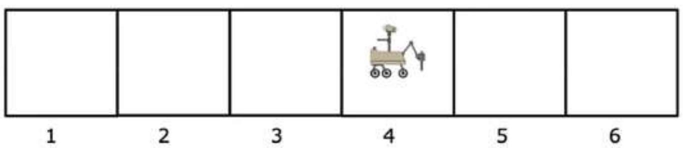
$= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

$= R(4) + (0.5)[0 + (0.5)0 + (0.5)^2 100]$

$= R(4) + (0.5) \max_{a'} Q(3,a')$

Random (stochastic) environment (Optional)

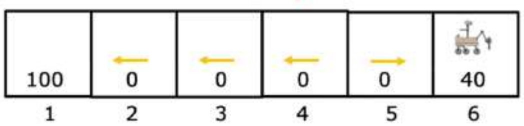
Stochastic Environment



← 0.9 0.1

→ 0.1 0.9

Expected Return



0 0 0 100

0 0 0 0 0 100

0 0 40

Expected Return = Average($R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$)

$= \mathrm{E}[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots]$

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:

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

3 ←

2 or 4

File Edit View Insert Cell Kernel Widgets Help   Trusted | Python 3

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

```
In [1]: import numpy as np
        from utils import *
```

```
In [2]: # Do not modify
        num_states = 6
        num_actions = 2
```

```
In [5]: terminal_left_reward = 100
        terminal_right_reward = 40
        each_step_reward = 0

        # Discount factor
        gamma = 0.5

        # Probability of going in the wrong direction
        misstep_prob = 0.4
```

```
In [6]: generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	32.18 ←	10.88 ←	6.15 →	13.23 →	40.0
100	0	0	0	0	40

Q(s,a)

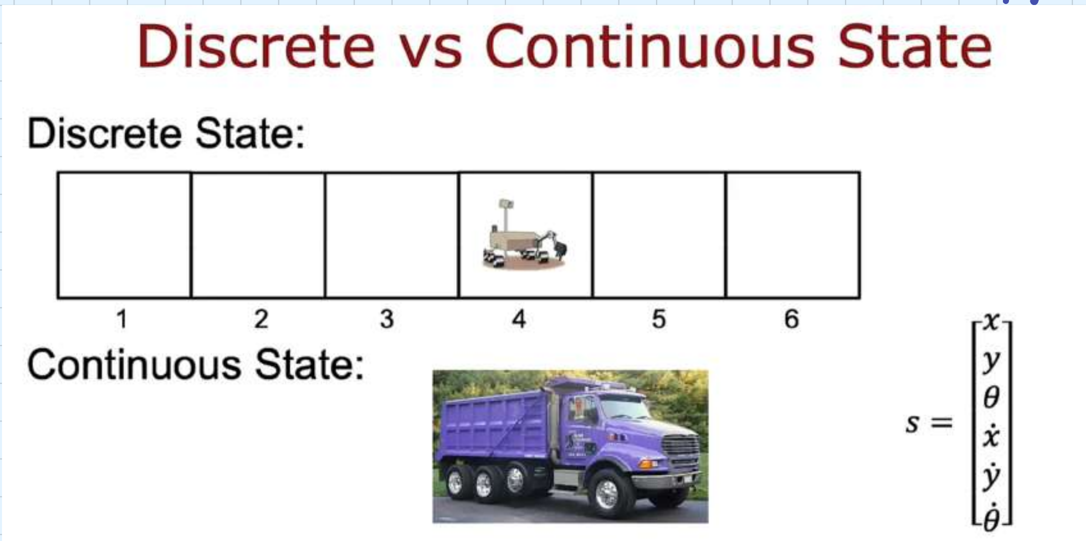
100.0	100.0	32.18	23.26	10.88	8.28	5.91	6.15	9.84	13.23	40.0	40.0
100								0		40	

你对机器人的控制程度降低了。

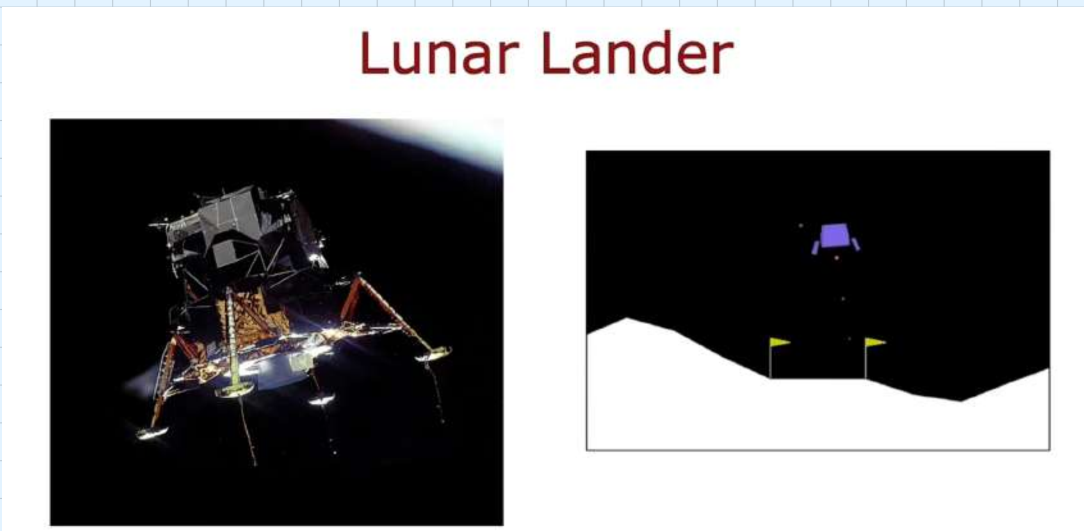
↳ 通过更改 misstep_prob 观察Q变化

更多的 S

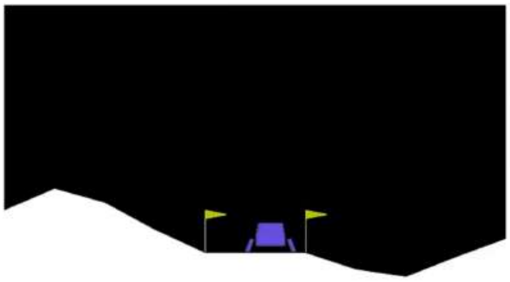
Example of Continuous state application

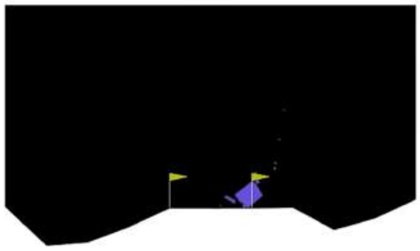


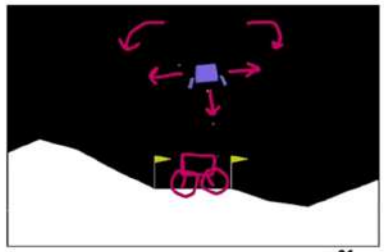
Lunar lander



if Good Policy if bad



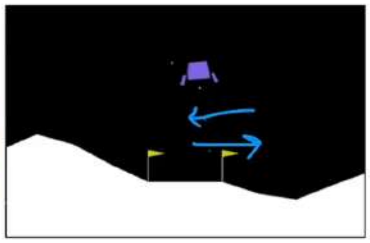




$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

Reward Function

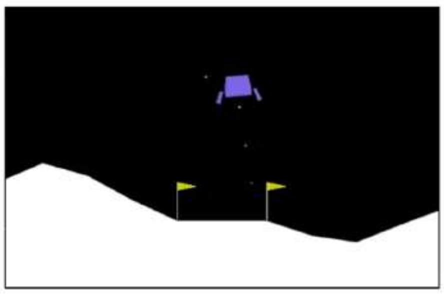
- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03



Lunar Lander Problem

Learn a policy π that, given

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



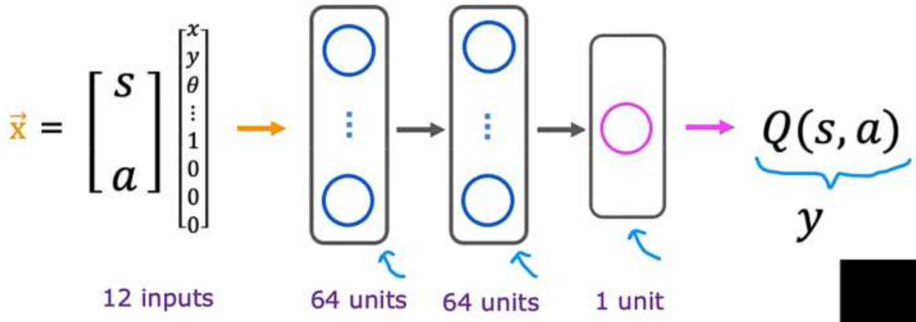
picks action $a = \pi(s)$ so as to maximize the return.

$$\gamma = 0.985$$

Learning the state-value function

DQN

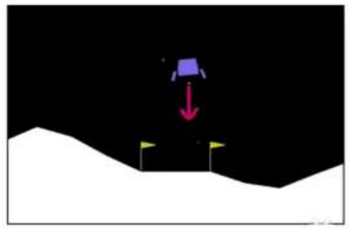
Deep Reinforcement Learning



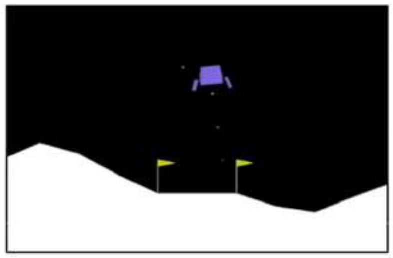
In a state s, use neural network to compute

$Q(s,\text{nothing})$, $Q(s,\text{left})$, $Q(s,\text{main})$, $Q(s,\text{right})$

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



Learning Algorithm



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

Repeat {

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

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

Train neural network:

Create training set of 10,000 examples using

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

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

Set $Q = Q_{new}$.

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

x, y

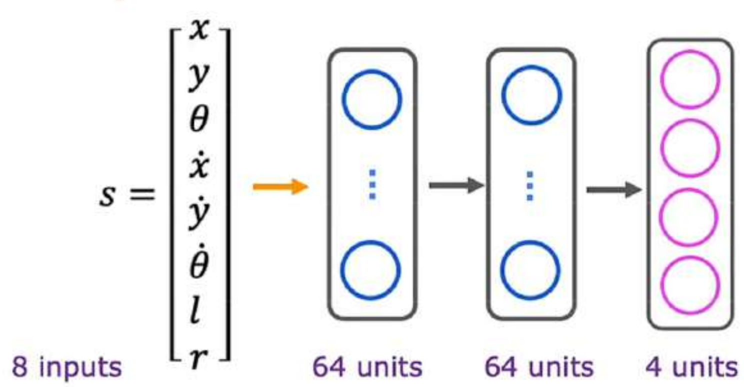
$x^{(1)}, y^{(1)}$

$\vdots$

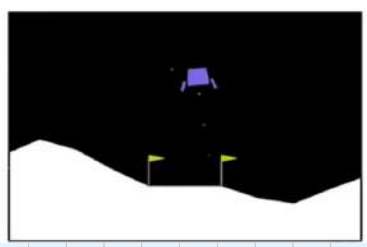
x^{10000}, y^{10000}

Algorithm refinement: Improved neural network architecture

Deep Reinforcement Learning



$Q(s,\text{nothing})$
$Q(s,\text{left})$
$Q(s,\text{main})$
$Q(s,\text{right})$
} 并行计算



In a state s, input s to neural network.

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

Algorithm refinement : ε-greedy Policy

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.

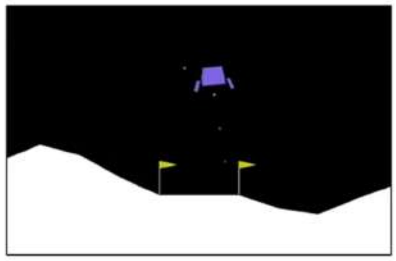
Train model:

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 $Q_{new}(s,a) \approx y$. $f_{w,b}(x) \approx y$

Set $Q = Q_{new}$.



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 $(\varepsilon = 0.05)$

0.95

Q(s,main) is low

↑

a

Start ε high

1.0 → 0.01

Gradually decrease

Algorithm refinement : Mini-batch and soft update

How to choose actions while still learning?

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

100,000,000

$$\boxed{J(w,b)} = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$m = 100{,}000{,}000$

$m' = 1{,}000$

repeat {

$$w = w - \alpha \frac{\partial}{\partial w} \boxed{\frac{1}{2m'}\sum_{i=1}^{m'}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2}$$

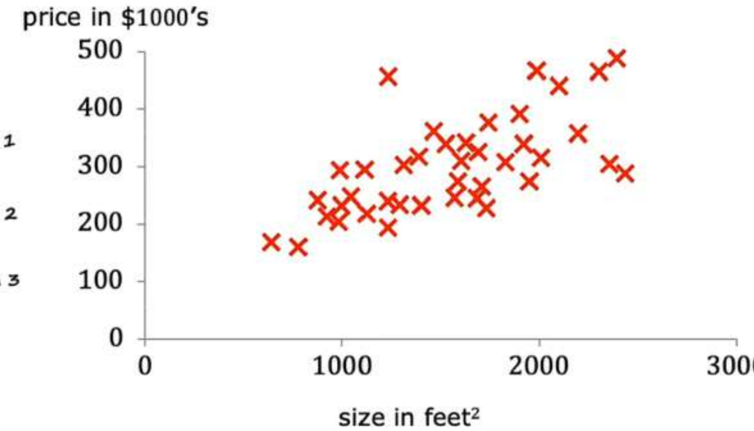
$$b = b - \alpha \frac{\partial}{\partial b} \frac{1}{2m'}\sum_{i=1}^{m'}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

}

Mini-batch

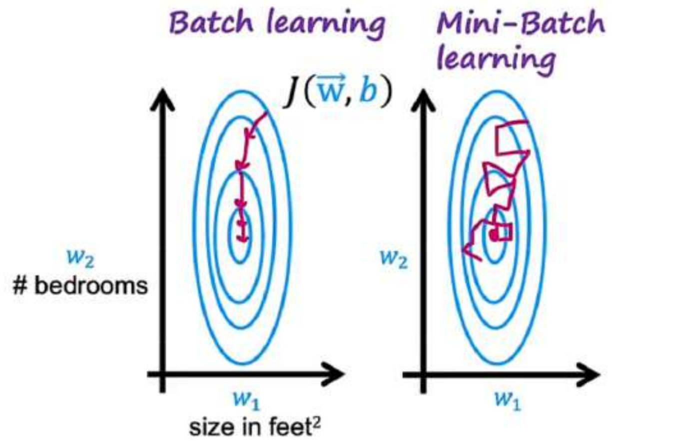
x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

mini-batch 1
mini-batch 2
mini-batch 3



Mini-batch

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870



↳ 解决 trainingset 过大的问题

Learning Algorithm

Initialize neural network randomly as guess of $\boxed{Q(s,a)}$

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

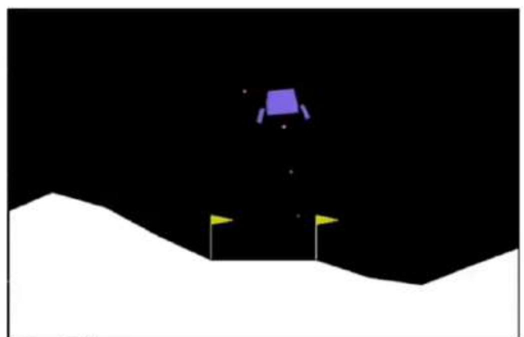
Train model:

Create training set of ~~10,000~~ 1,000 examples using

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

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

$\boxed{\text{Set } Q = Q_{new}.}$



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

⋮

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

② minibatch 会带来 Q 突然变化大

3) λ soft update

Soft Update

Set $Q = Q_{new}$. ← $Q(s, a)$

w, B $\quad$ W_{new}, B_{new}

$$W = 0.01\,W_{new} + 0.99\,W \qquad w = 1W_{new} + 0W$$

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

The state 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.

Summery of the Course

Courses

- Supervised Machine Learning: Regression and Classification
  Linear regression, logistic regression, gradient descent
- Advanced Learning Algorithms
  Neural networks, decision trees, advice for ML
- Unsupervised Learning, Recommenders, Reinforcement Learning
  Clustering, anomaly detection, collaborative filtering, content-based filtering, reinforcement learning