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

time flying well and
hopefully to never crash.

DeepLearning.AI Stanford ONLINE Andrew Ng

5:54 / 8:48

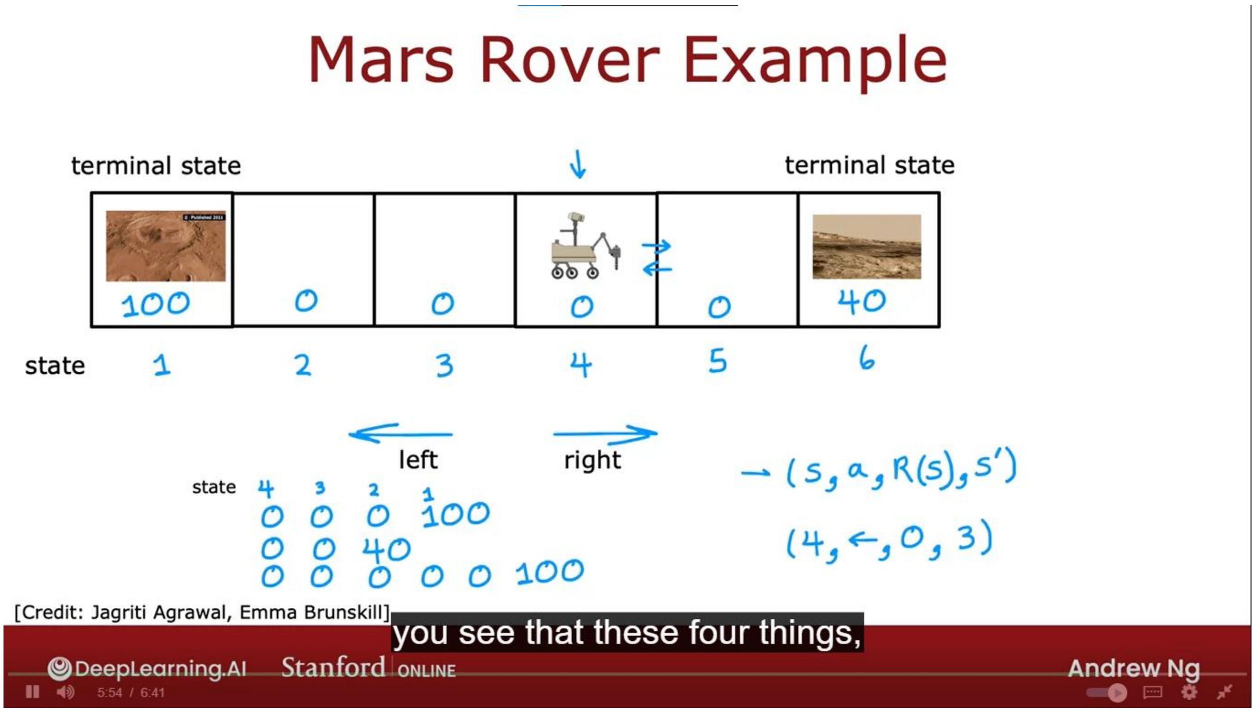
Applications

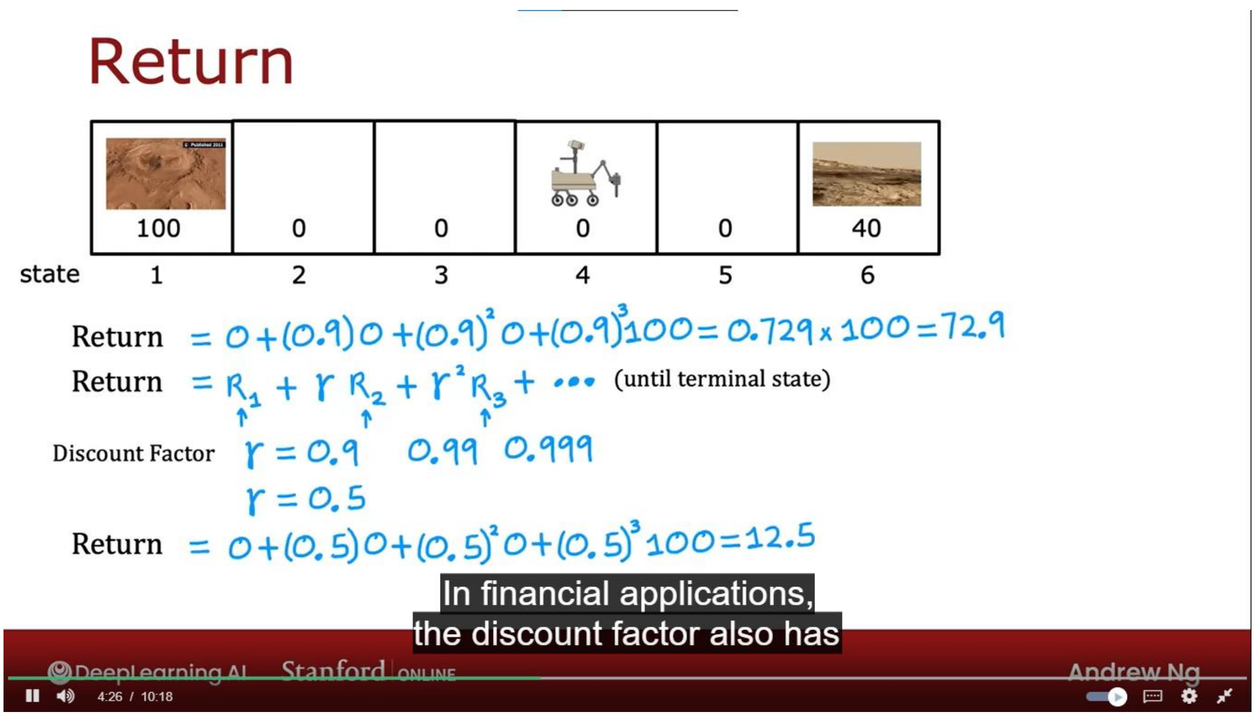
- Controlling robots
- Factory optimization
- Financial (stock) trading
- Playing games (including video games)

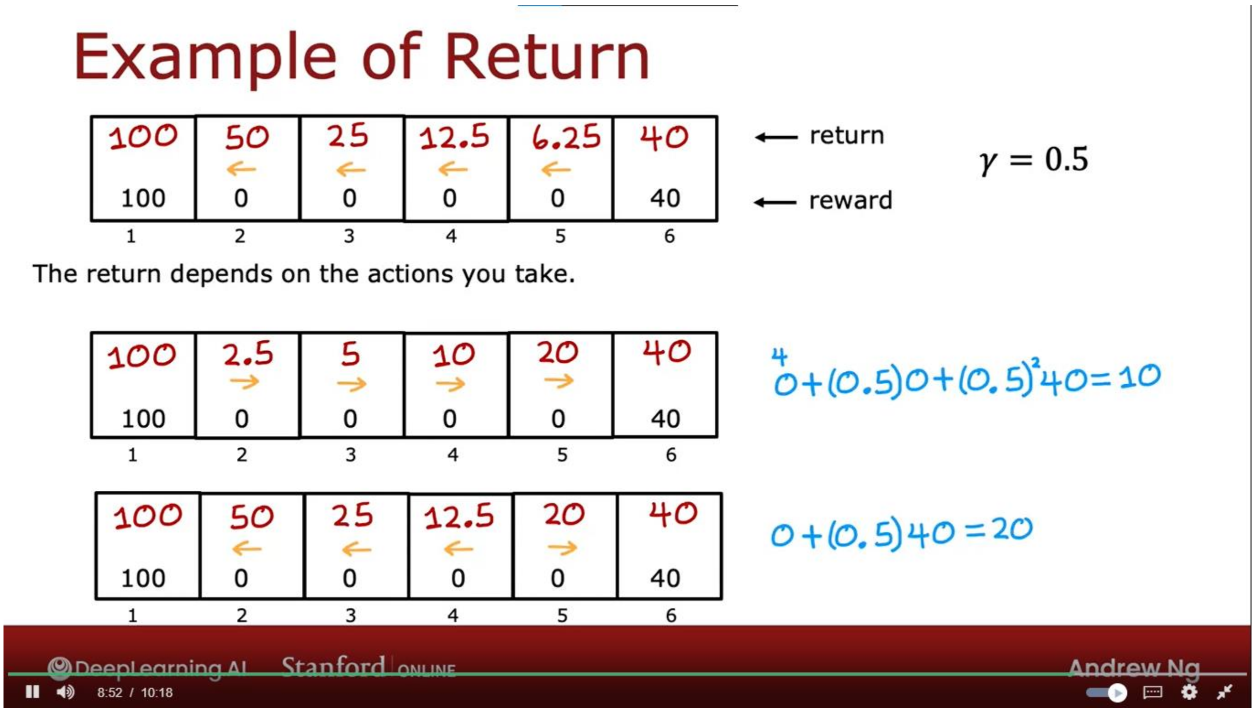
For example, one of my friends was
working on efficient stock execution.

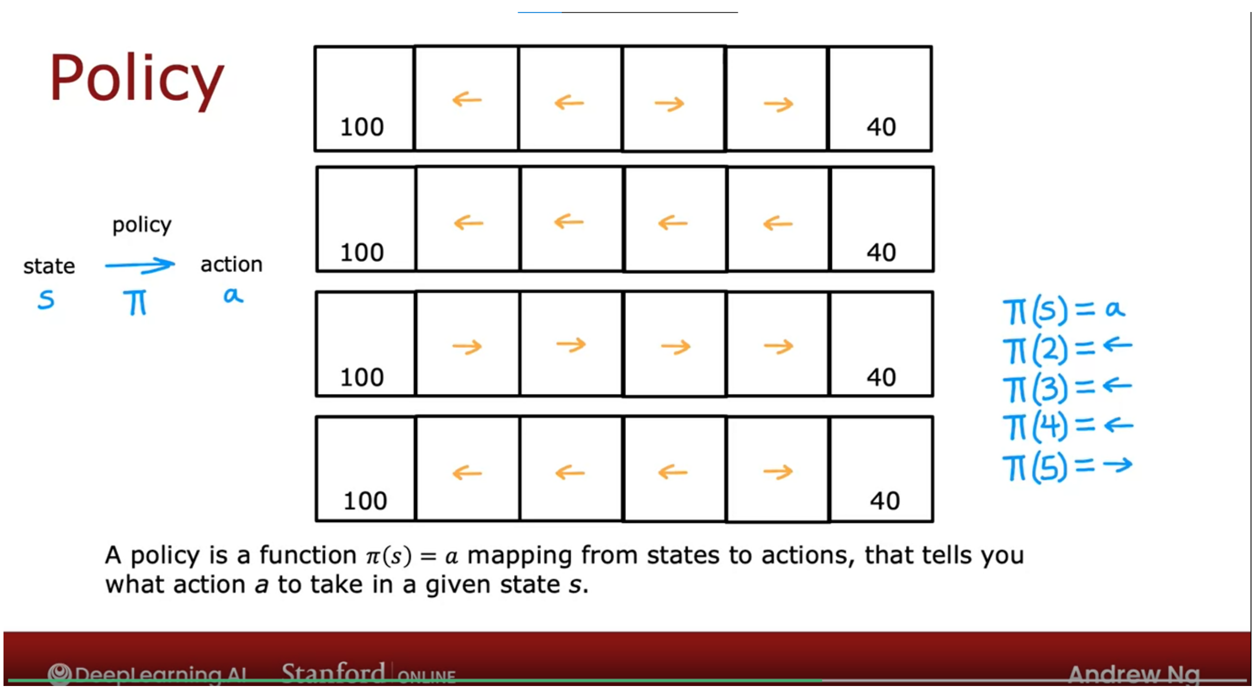
DeepLearning.AI Stanford ONLINE Andrew Ng

7:29 / 8:48

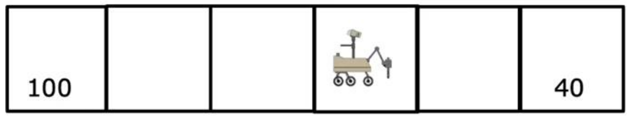

Mars Rover Example
terminal state
terminal state
100
0
0
0
0
40
state
1
2
3
4
5
6
left
right
state
4 3 2 1
0 0 0 100
0 0 40
0 0 0 0 0 100
→ (s, a, R(s), s')
(4, ←, 0, 3)
[Credit: Jagriti Agrawal, Emma Brunskill]
you see that these four things,
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:54 / 6:41


Return
100
0
0
0
0
40
state
1
2
3
4
5
6
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
$\gamma = 0.5$
Return $= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 = 12.5$
In financial applications,
the discount factor also has
DeepLearning.AI
Stanford ONLINE
Andrew Ng
4:26 / 10:18


Example of Return
100 50 25 12.5 6.25 40
100 0 0 0 0 40
1 2 3 4 5 6
← return
← reward
$\gamma = 0.5$
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
4
$0+(0.5)0+(0.5)^2 40=10$
100 50 25 12.5 20 40
100 0 0 0 0 40
1 2 3 4 5 6
$0+(0.5)40=20$
DeepLearning.AI
Stanford ONLINE
Andrew Ng
8:52 / 10:18


Policy
100 40
100 40
100 40
100 40
policy
state → action
s π a
π(s) = a
π(2) = ←
π(3) = ←
π(4) = ←
π(5) = →
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*.
DeepLearning.AI
Stanford ONLINE
Andrew Ng

The goal of reinforcement learning



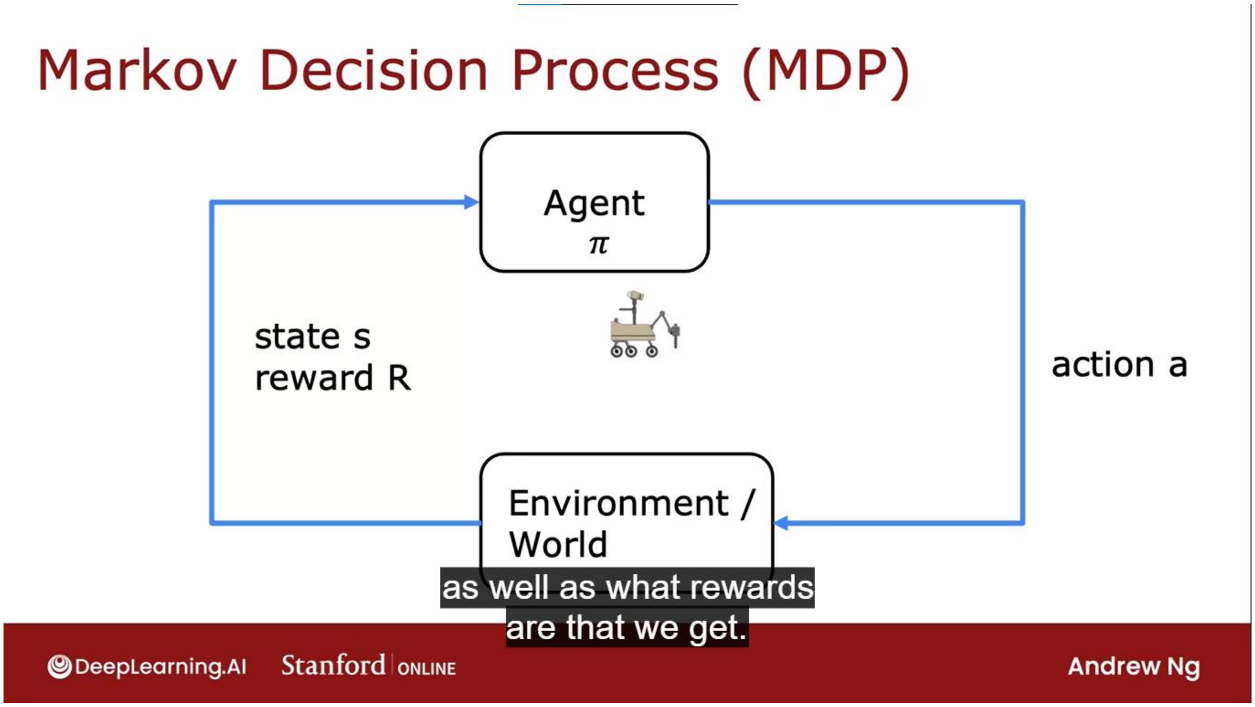
Find a policy π that tells you what action (a = $\pi(s)$) to take in every state (s) so as to maximize the return.

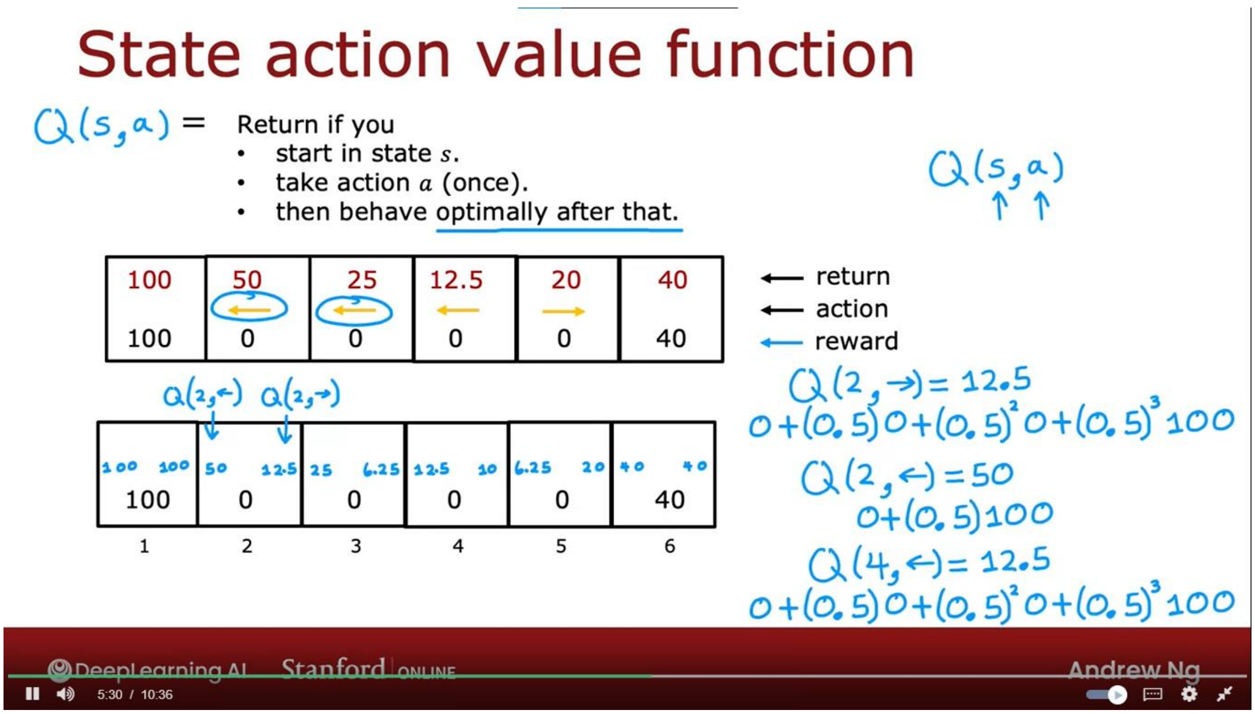
more natural
terminology but policy

	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 $\pi(s) = a$

This formalism of a reinforcement
learning application

DeepLearning.AI Stanford ONLINE Andrew Ng


Markov Decision Process (MDP)
Agent
π
state s
reward R
action a
Environment / World
as well as what rewards are that we get.
DeepLearning.AI
Stanford ONLINE
Andrew Ng


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
100 0 0 0 0 40
return
action
reward
Q(2,←) Q(2,→)
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,→) = 12.5
0+(0.5)0+(0.5)²0+(0.5)³100
Q(2,←) = 50
0+(0.5)100
Q(4,←) = 12.5
0+(0.5)0+(0.5)²0+(0.5)³100
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:30 / 10:36

Picking actions

	1	2	3	4	5	6	
return	100	50	25	12.5	20	40	← return
action		←	←	←	→		← action
reward	100	0	0	0	0	40	← reward

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

$Q(4,\leftarrow)$ $\underline{12.5}$ $\quad$ $Q(4,\rightarrow)$ 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

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
			←		

$R(1)=100$ $\quad R(2)=0$ $\cdots$ $\quad R(6)=40$

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'

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

We'll come back to see why

Bellman Equation

To exit full screen, press Esc

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

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

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

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

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

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

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

any other state action in

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

$$Q(s,a) = \underbrace{R(s)}_{\text{Reward you get right away}} + \gamma \underbrace{\max_{a'} Q(s',a')}_{\text{Return from behaving optimally starting from state } s'}$$

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

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

the total return you get

Explanation of Bellman Equation

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

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

$Q(4, \leftarrow)$

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

$= R(4) + (0.5)\underbrace{[0 + (0.5)0 + (0.5)^2 100]}$

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

Gamma times the returns from
the next state s prime.

Expected Return

100	← 0	← 0	← 0	→ 0	 40
1	2	3	4	5	6

0 0 0 100

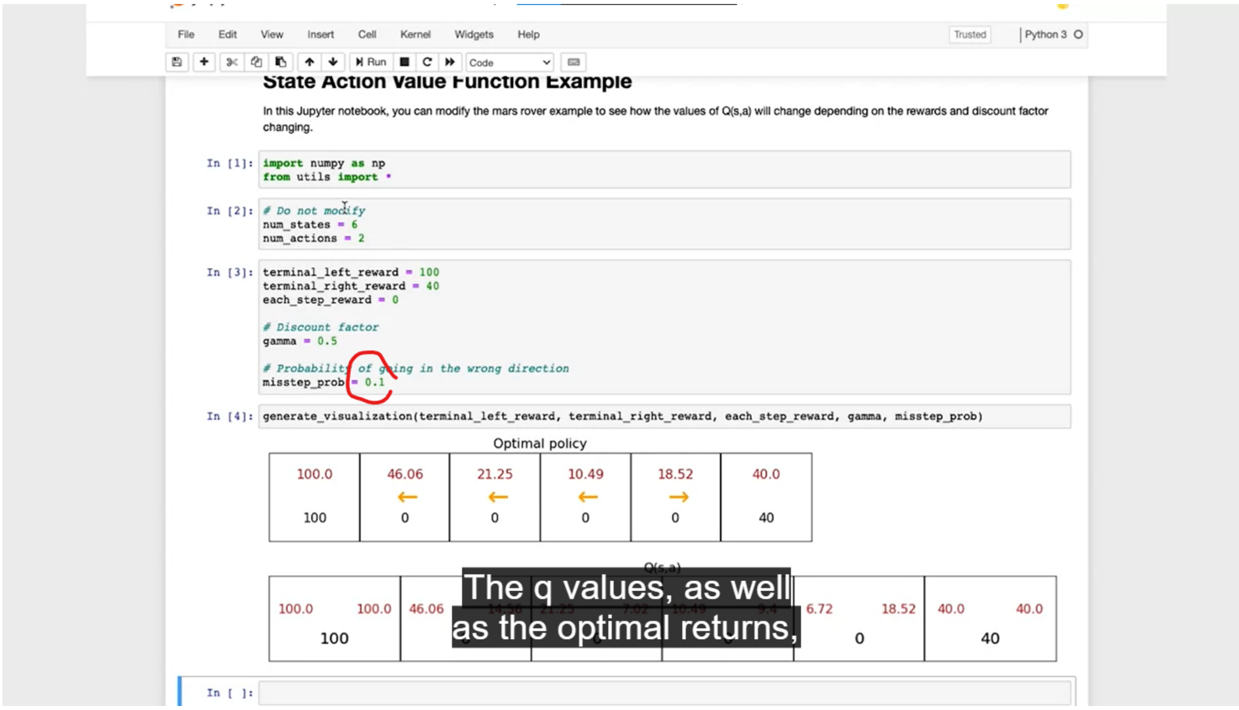
0 0 0 0 0 100

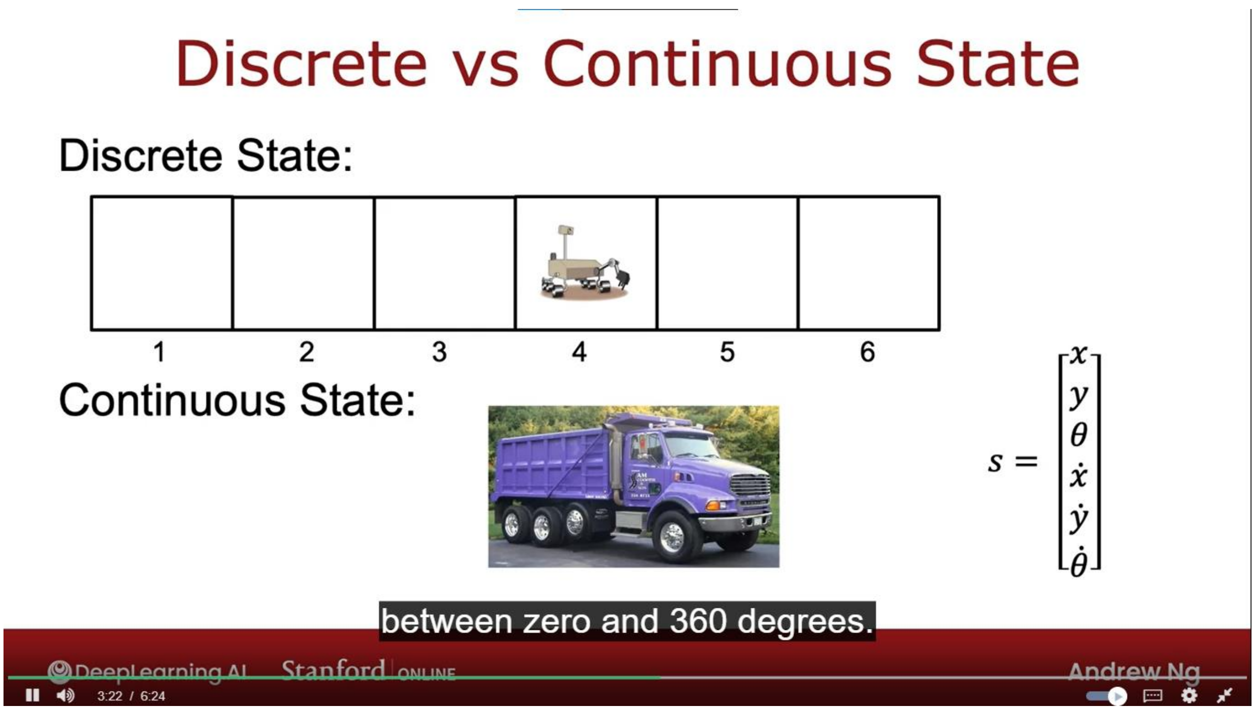
0 0 40

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

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

the average or the expected
sum of discounted rewards.


File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
State Action Value Function Example
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 [3]: 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 = 0.1
In [4]: generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
Optimal policy
100.0 46.06 21.25 10.49 18.52 40.0
100 0 0 0 0 40
Q(s,a)
100.0 100.0 46.06 6.72 18.52 40.0 40.0
100 0 40
The q values, as well as the optimal returns,
In [ ]:

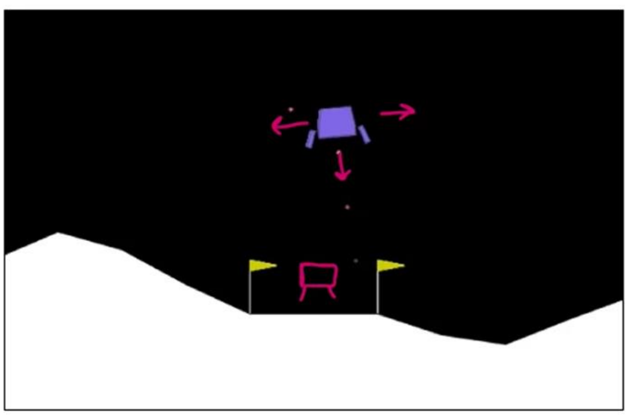

Discrete vs Continuous State
Discrete State:
1 2 3 4 5 6
Continuous State:
$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$
between zero and 360 degrees.
DeepLearning.AI Stanford ONLINE
Andrew Ng
3:22 / 6:24

Autonomous Helicopter

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

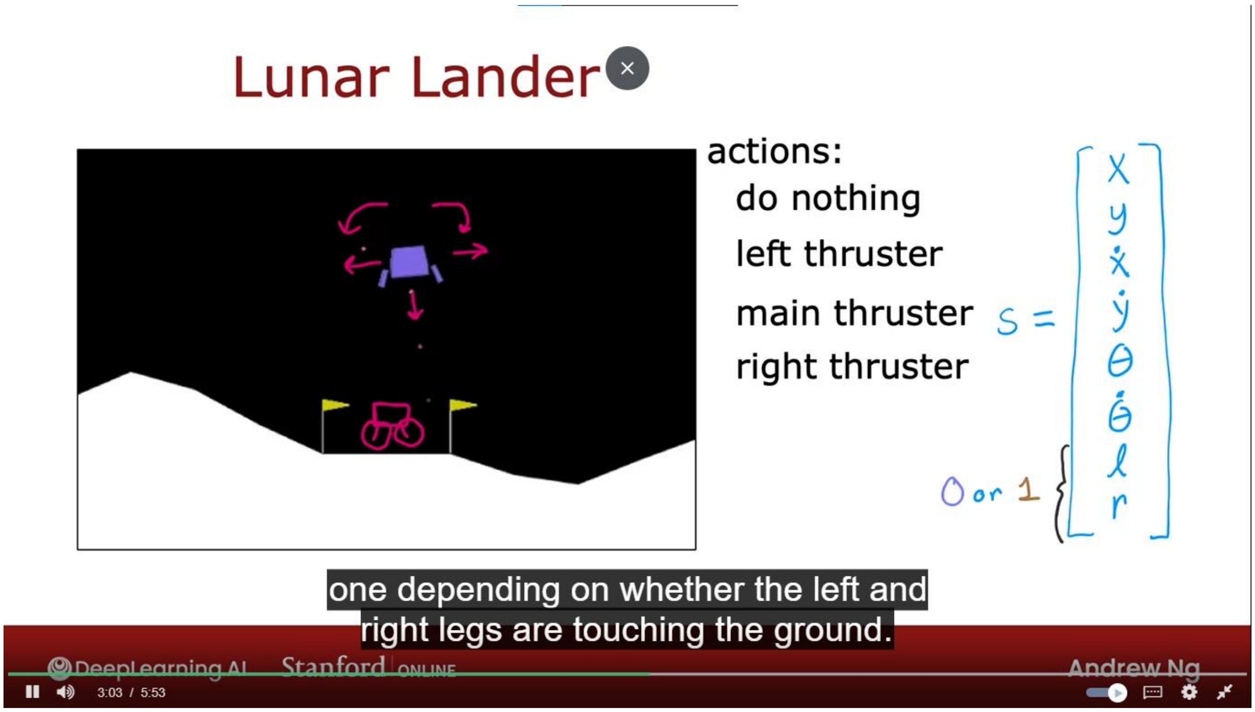
and then the row pitch,

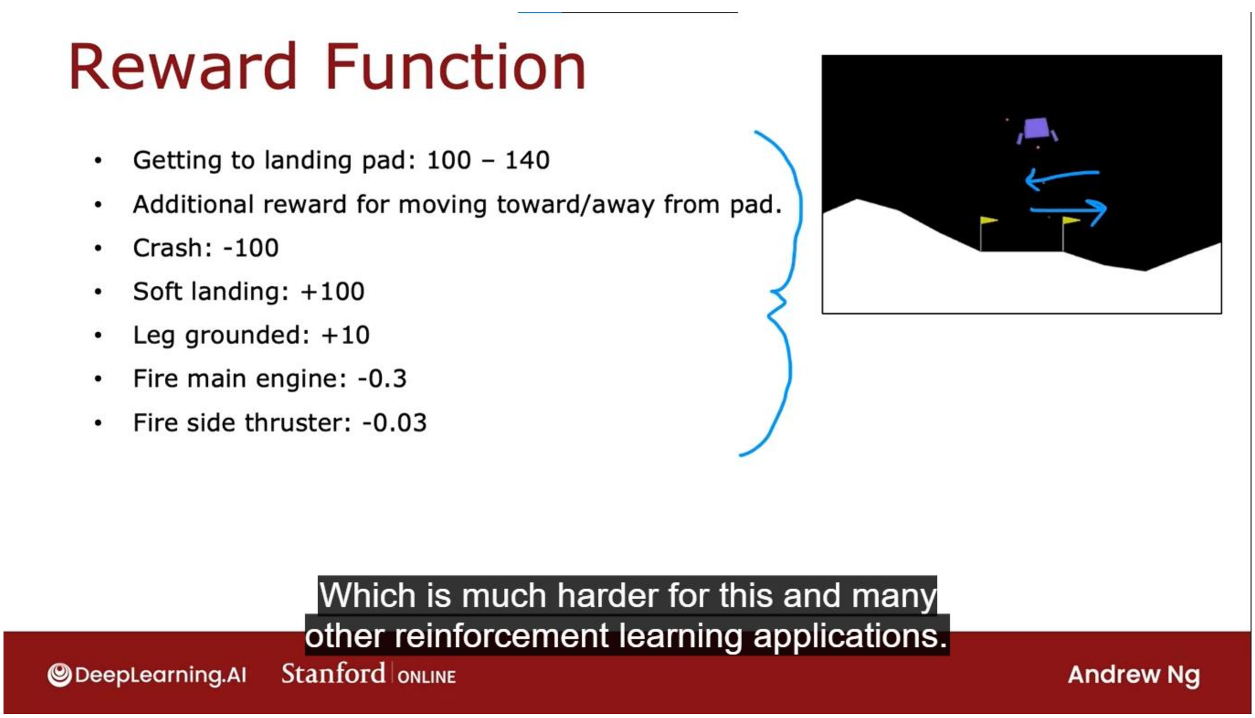
Lunar Lander

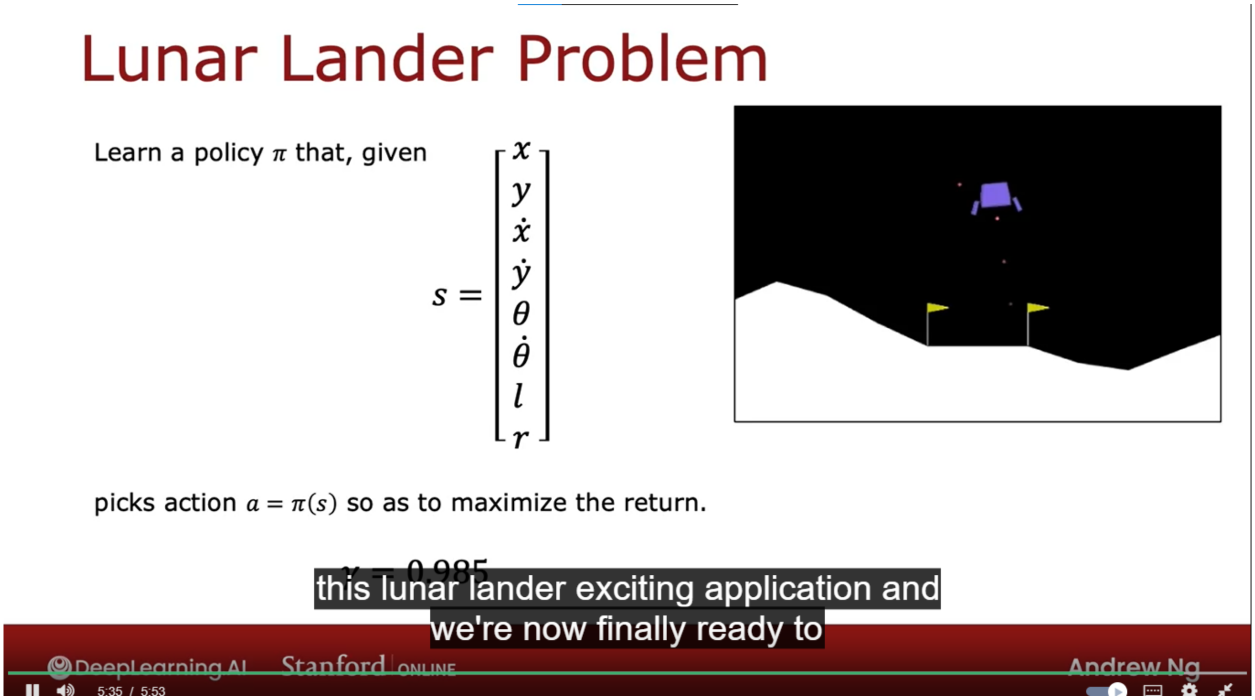


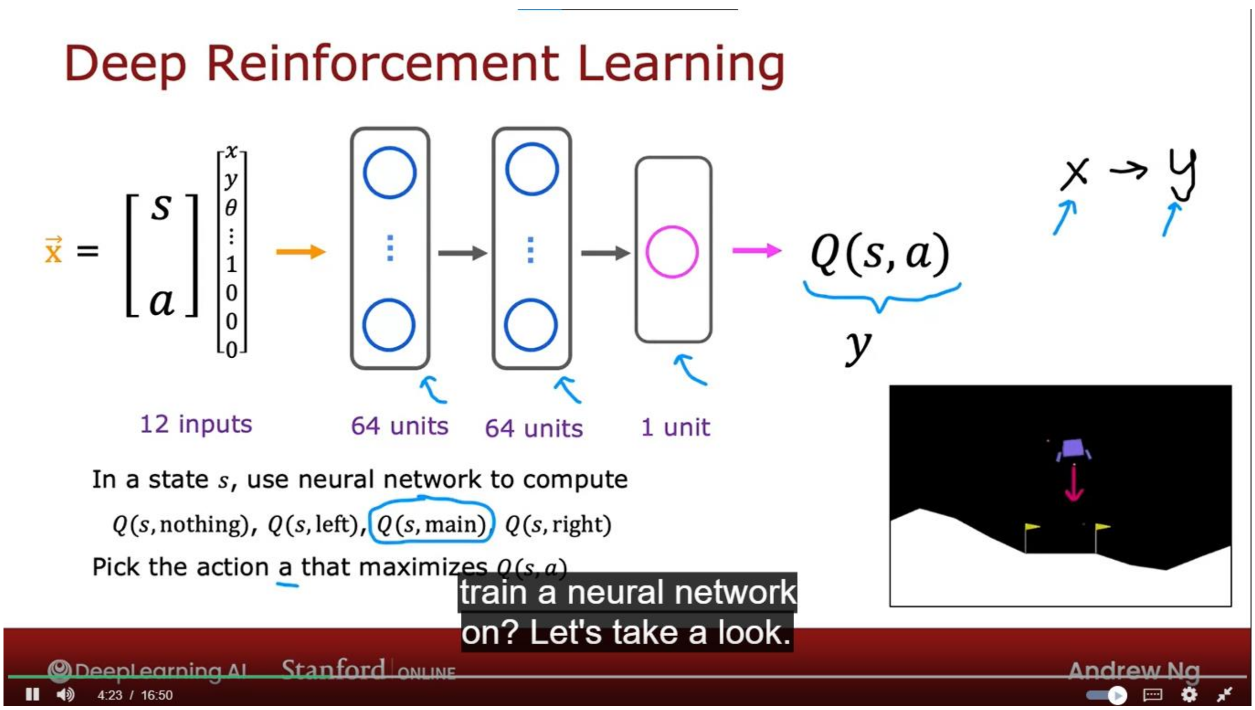
actions:
- do nothing
- left thruster
- main thruster
- right thruster

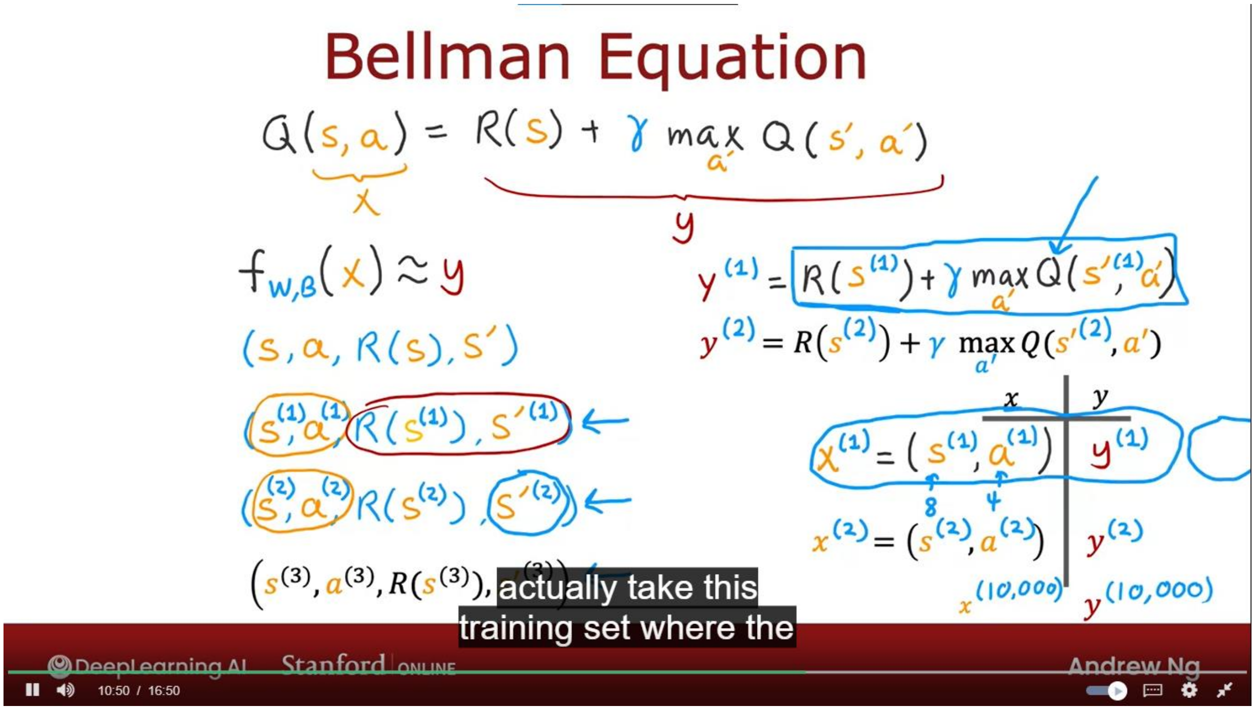
So it's the lunar lander safely between
these two flags here on the landing pad.

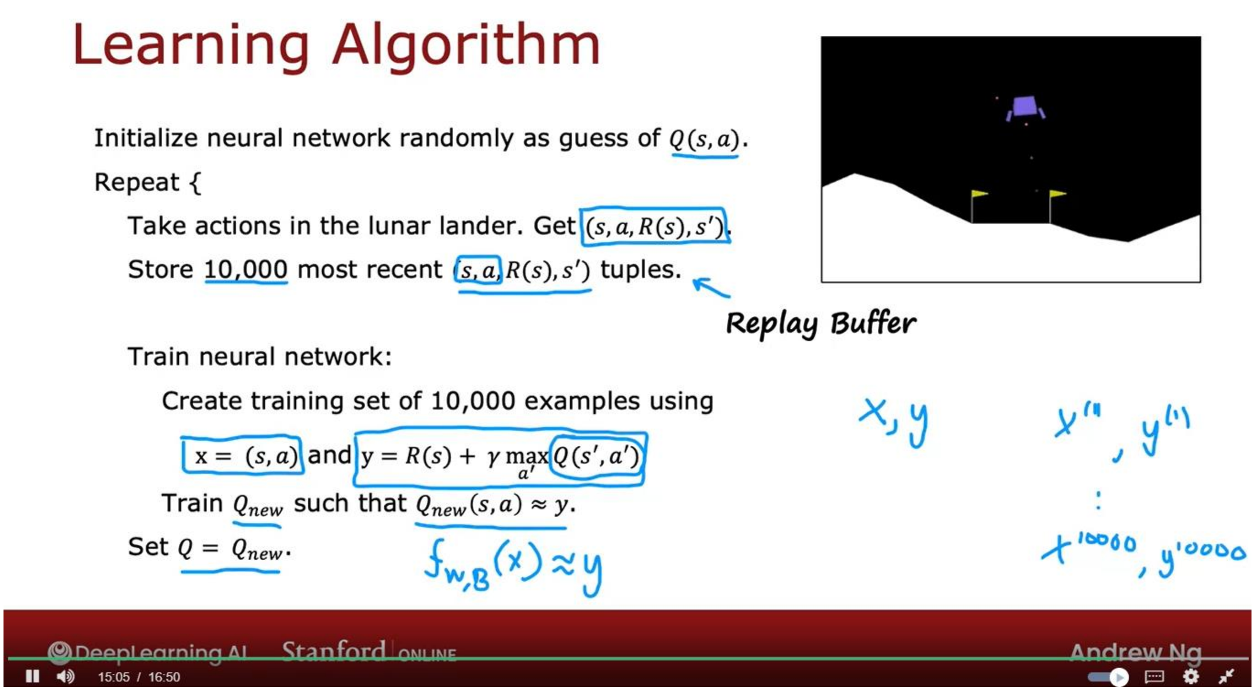

Lunar Lander
actions:
do nothing
left thruster
main thruster
right thruster
S =
x
y
ẋ
ẏ
θ
θ̇
l
r
0 or 1
one depending on whether the left and
right legs are touching the ground.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
3:03 / 5:53


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
Which is much harder for this and many
other reinforcement learning applications.
DeepLearning.AI
Stanford ONLINE
Andrew Ng


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.
$\gamma = 0.985$
this lunar lander exciting application and
we're now finally ready to
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:35 / 5:53


Deep Reinforcement Learning
$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix}$
$\begin{bmatrix} x \\ y \\ \theta \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$Q(s,a)$
y
x → y
12 inputs
64 units
64 units
1 unit
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 $Q(s,a)$
train a neural network
on? Let's take a look.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
4:23 / 16:50


Bellman Equation
Q(s, a) = R(s) + γ max_a' Q(s', a')
x
y
f_W,B(x) ≈ y
(s, a, R(s), s')
(s^(1), a^(1), R(s^(1)), s'^(1))
(s^(2), a^(2), R(s^(2)), s'^(2))
(s^(3), a^(3), R(s^(3)), s'^(3))
y^(1) = R(s^(1)) + γ max_a' Q(s'^(1), a')
y^(2) = R(s^(2)) + γ max_a' Q(s'^(2), a')
x
y
x^(1) = (s^(1), a^(1))
y^(1)
8
4
x^(2) = (s^(2), a^(2))
y^(2)
x^(10,000)
y^(10,000)
actually take this
training set where the
DeepLearning.AI
Stanford ONLINE
Andrew Ng
10:50 / 16:50


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) + γ max_a' Q(s', a')
Train Q_new such that Q_new(s, a) ≈ y.
Set Q = Q_new.
f_W,B(x) ≈ y
x, y
x^(1), y^(1)
x^(10000), y^(10000)
DeepLearning.AI
Stanford ONLINE
Andrew Ng
15:05 / 16:50

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

$Q(s, \text{main})$ is low

↑

a

Start ε high

1.0 → 0.01

Gradually decrease

in the Jupiter lab that
shows you how to do this.

DeepLearning.AI Stanford ONLINE Andrew Ng

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.

machine learning systems as well.

DeepLearning.AI Stanford ONLINE Andrew Ng

2:25 / 2:54