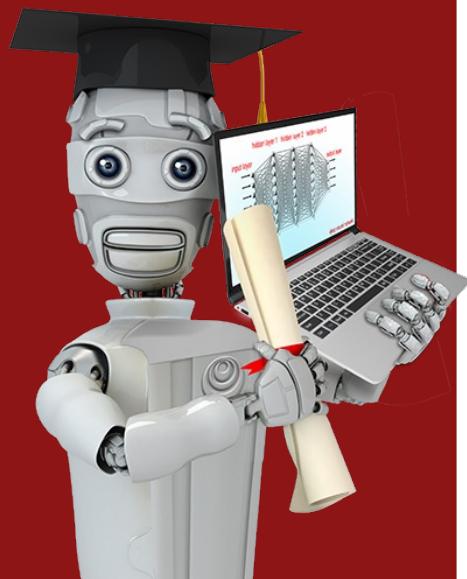


Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

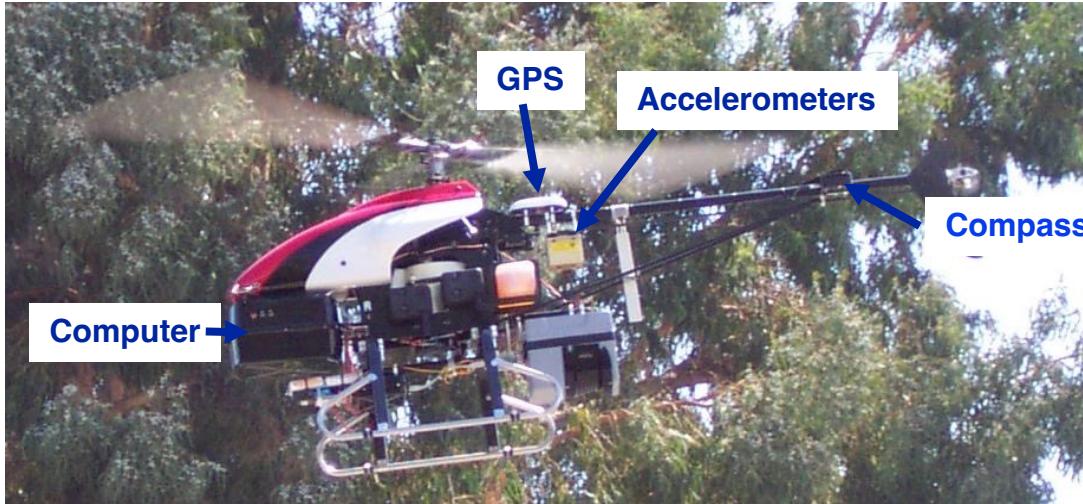
For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



Reinforcement Learning Introduction

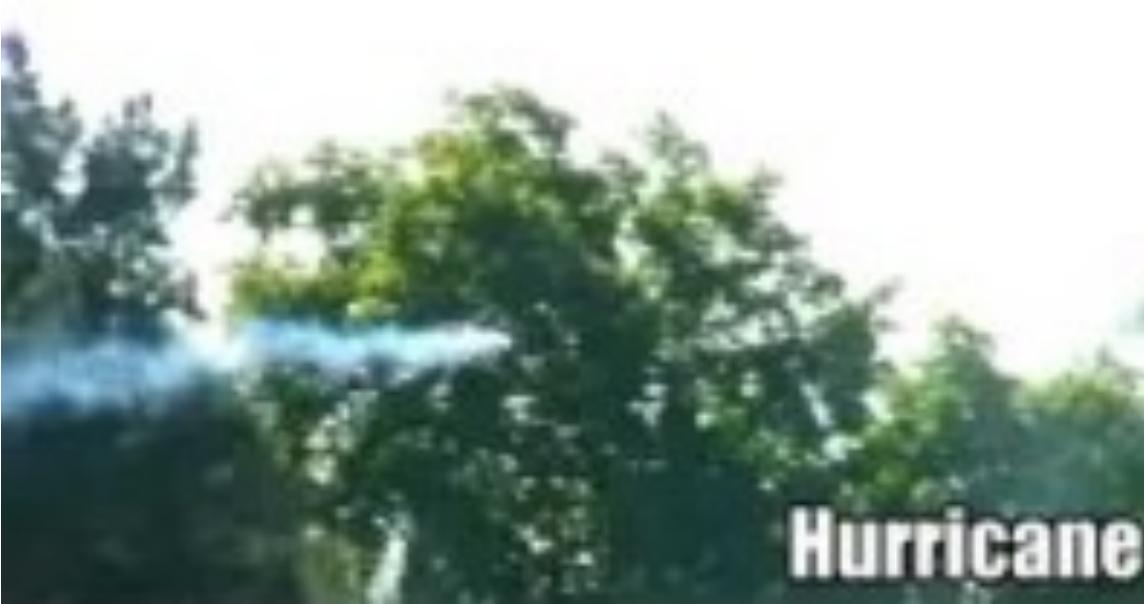
What is Reinforcement Learning?

Autonomous Helicopter



How to fly it?

Autonomous Helicopter



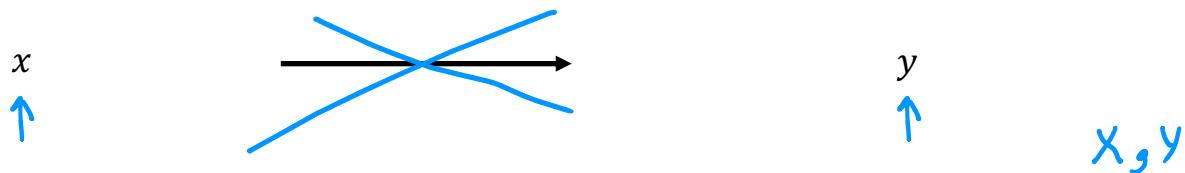
[Thanks to Pieter Abbeel, Adam Coates and Morgan Quigley]

→ For more videos: <http://heli.stanford.edu>.

Reinforcement Learning

position of helicopter \longrightarrow how to move control sticks

state s \longrightarrow action a



reward function

positive reward : helicopter flying well $+1$

negative reward : helicopter flying poorly -1000

Robotic Dog Example

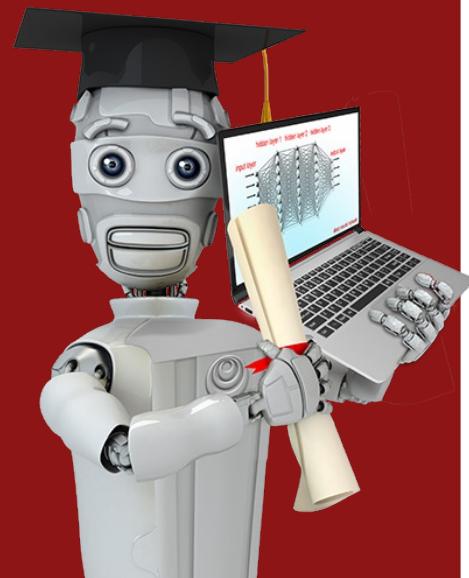


[Thanks to Zico Kolter]

Applications

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



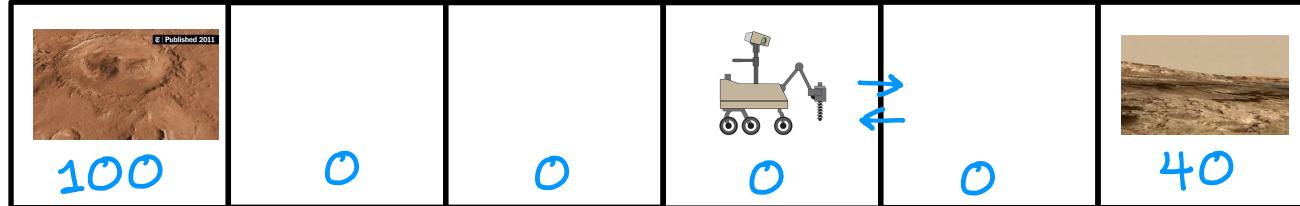


Reinforcement Learning formalism

Mars rover example

Mars Rover Example

terminal state



state

1

2

3

4

5

6

state

4

3

2

1

100

0

0

40

0

0

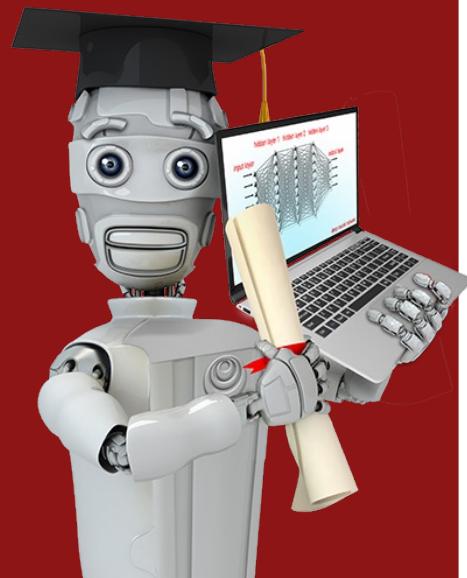
right

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

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

left

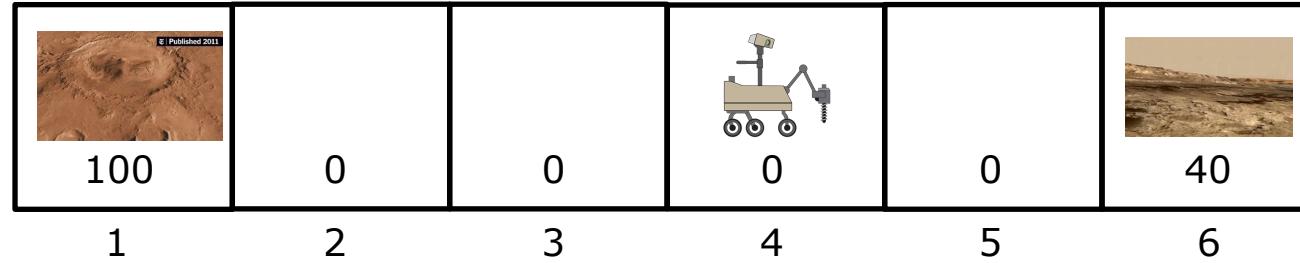
[Credit: Jagriti Agrawal, Emma Brunskill]



Reinforcement Learning formalism

The Return in reinforcement learning

Return



$$\text{Return} = 0 + (0.9)0 + (0.9)^20 + (0.9)^3100 = 0.729 \times 100 = 72.9$$

$$\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \quad (\text{until terminal state})$$

$$\text{Discount Factor} \quad \gamma = 0.9 \quad 0.99 \quad 0.999$$

$$\gamma = 0.5$$

$$\text{Return} = 0 + (0.5)0 + (0.5)^20 + (0.5)^3100 = 12.5$$

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

$$\gamma = 0.5$$

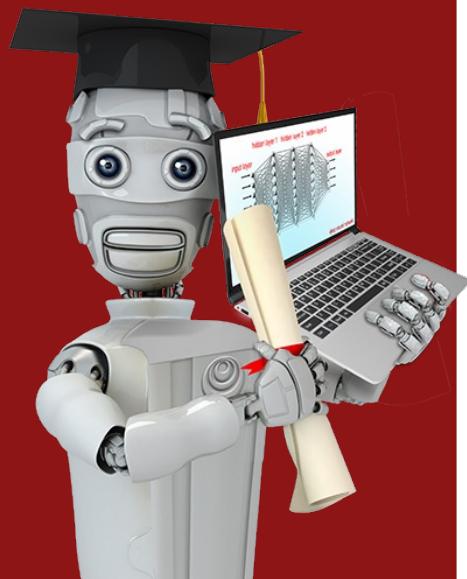
The return depends on the actions you take.

100	2.5	5	10	20	40
100	0	0	0	0	40
1	2	3	4	5	6

$$0 + (0.5)0 + (0.5)^2 40 = 10$$

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

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

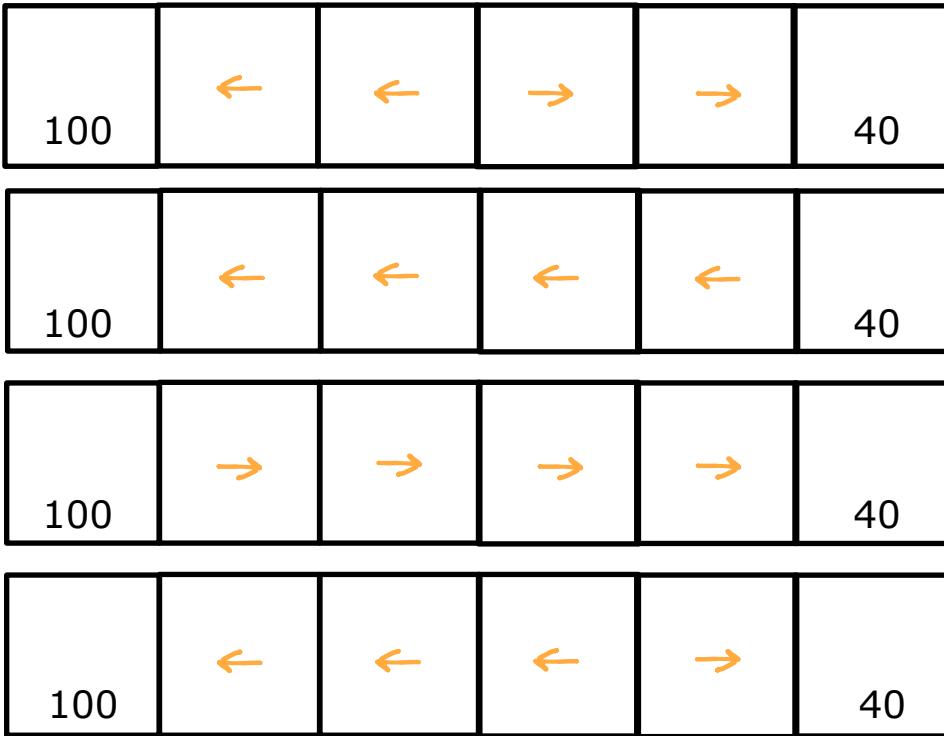


Reinforcement Learning formalism

Making decisions: Policies in reinforcement learning

Policy

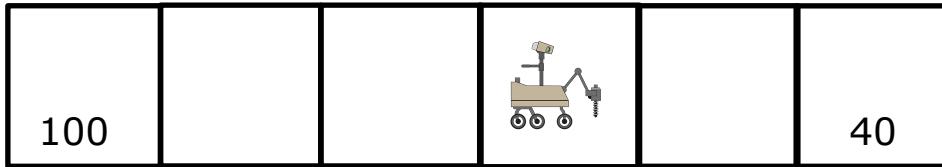
state s policy π action a



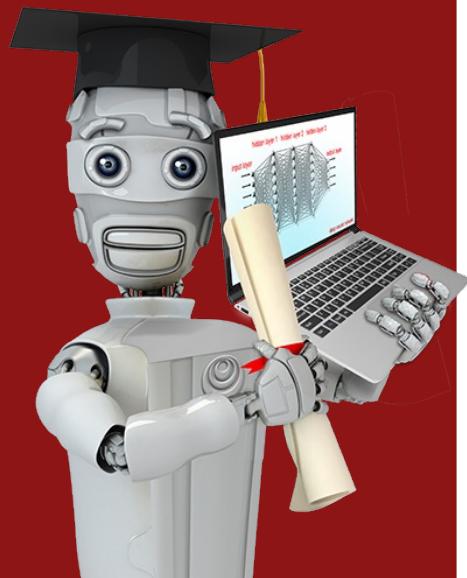
$$\begin{aligned}\pi(s) &= a \\ \pi(2) &= \leftarrow \\ \pi(3) &= \leftarrow \\ \pi(4) &= \leftarrow \\ \pi(5) &= \rightarrow\end{aligned}$$

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



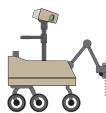
Find a policy π that tells you what action ($a = \pi(s)$) to take in every state (s) so as to maximize the return.



Reinforcement Learning formalism

Review of key concepts

Mars rover



Helicopter



Chess



- states
- actions
- rewards
- discount factor γ
- return
- policy π

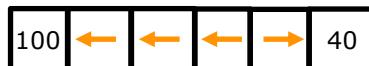
6 states



100, 0, 40

0.5

$$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$$



position of helicopter

how to move
control stick

+1, -1000

0.99

$$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$$

Find $\pi(s) = a$ 

pieces on board

possible move

+1, 0, -1

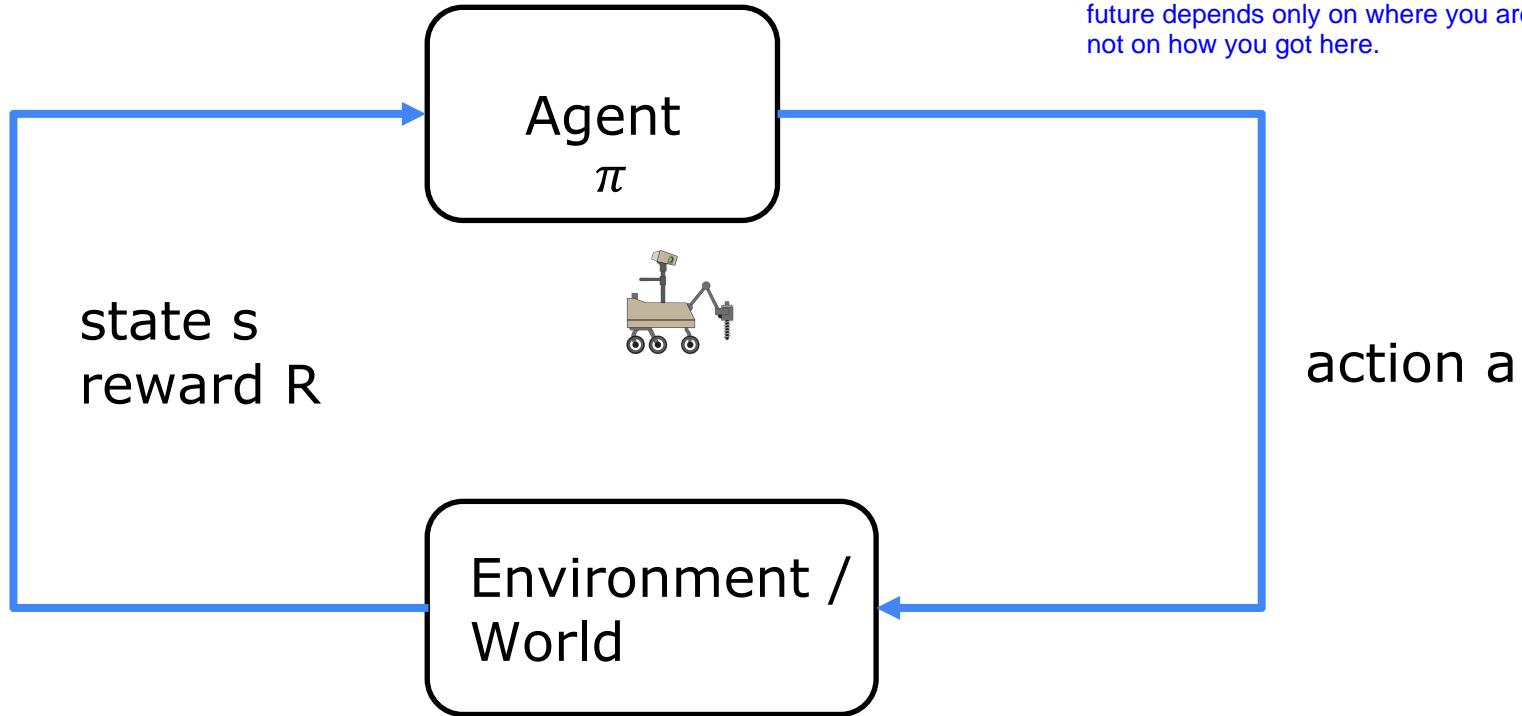
0.995

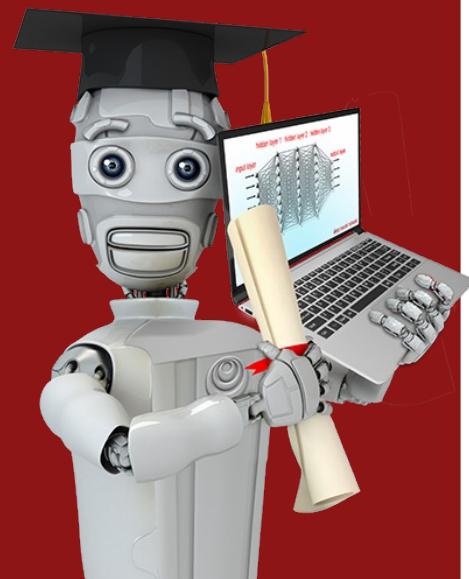
$$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$$

Find $\pi(s) = a$ 

Markov Decision Process (MDP)

in a MDP, The term Markov in the MDP or Markov decision process refers to that the future only depends on the current state and not on anything that might have occurred prior to getting to the current state. In other words, in a Markov decision process, the future depends only on where you are now, not on how you got here.





State-action value function

State-action value function definition

State action value function (Q-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

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

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

1 2 3 4 5 6

$$Q(2, \rightarrow) = 12.5$$

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

$$Q(2, \leftarrow) = 50$$

$$0 + (0.5) 100$$

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

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

Picking actions

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

← return
← action
← reward

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	40	40	40
1	2	3	4	5	6						

$$\begin{array}{l} Q(4, \leftarrow) \\ \underline{12.5} \end{array} \quad \begin{array}{l} Q(4, \rightarrow) \\ 10 \end{array}$$

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

$\rightarrow \pi(s) = a$

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

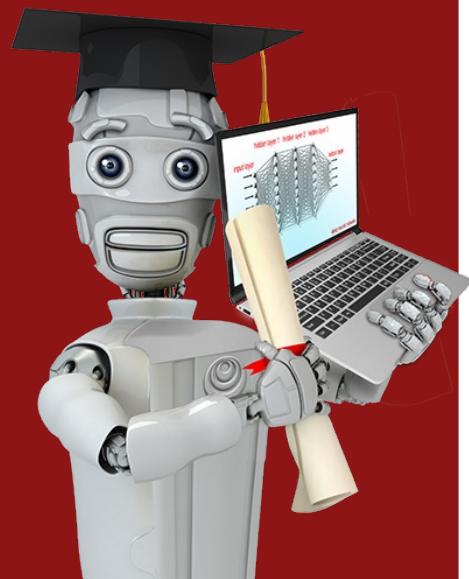
- start in state s .
- take action a (once).
- then behave optimally after that.

The best possible return from state s is $\max_a Q(s, a)$.

Q^*

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

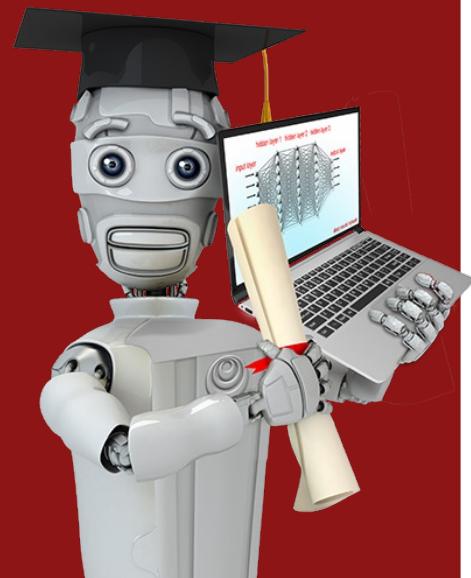
Optimal Q function



State-action value function

State-action value function example

Jupyter Notebook



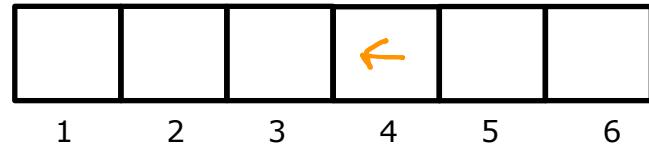
State-action value function

Bellman Equation

Bellman Equation

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

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



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

s : current state

$R(s)$ = reward of current state

a : current action

s' : state you get to after taking action a

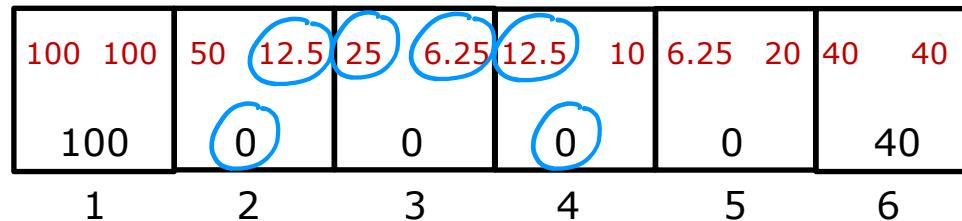
a' : action that you take in state s'

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

Bellman Equation

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

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



$$\begin{aligned} s &= 2 \\ a &= \rightarrow \\ s' &= 3 \end{aligned}$$

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

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

$$\begin{aligned} s &= 4 \\ a &= \leftarrow \\ s' &= 3 \end{aligned}$$

Explanation of Bellman Equation

$\{ Q(s, a) = \text{Return if you } \leftarrow$

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

$\rightarrow \text{The best possible return from state } s' \text{ is } \max_a Q(s', a')$ $s \rightarrow s'$

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

Reward you get right away Return from behaving optimally starting from state s' .

$$Q(s, a) = R_1 + r [R_2 + r^2 R_3 + r^3 R_4 + \dots]$$

$R_1 + r R_2 + r^2 R_3 + r^3 R_4 + \dots$

↑ ↑ ↑

Explanation of Bellman Equation

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

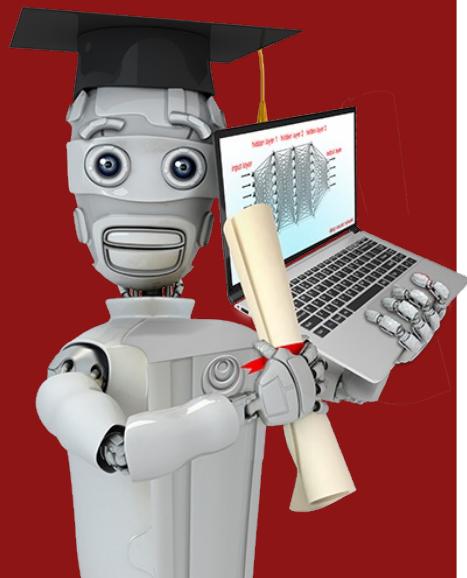
100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	0	40	40	40
1	2	3	4	5	6						

$Q(4, \leftarrow)$

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

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

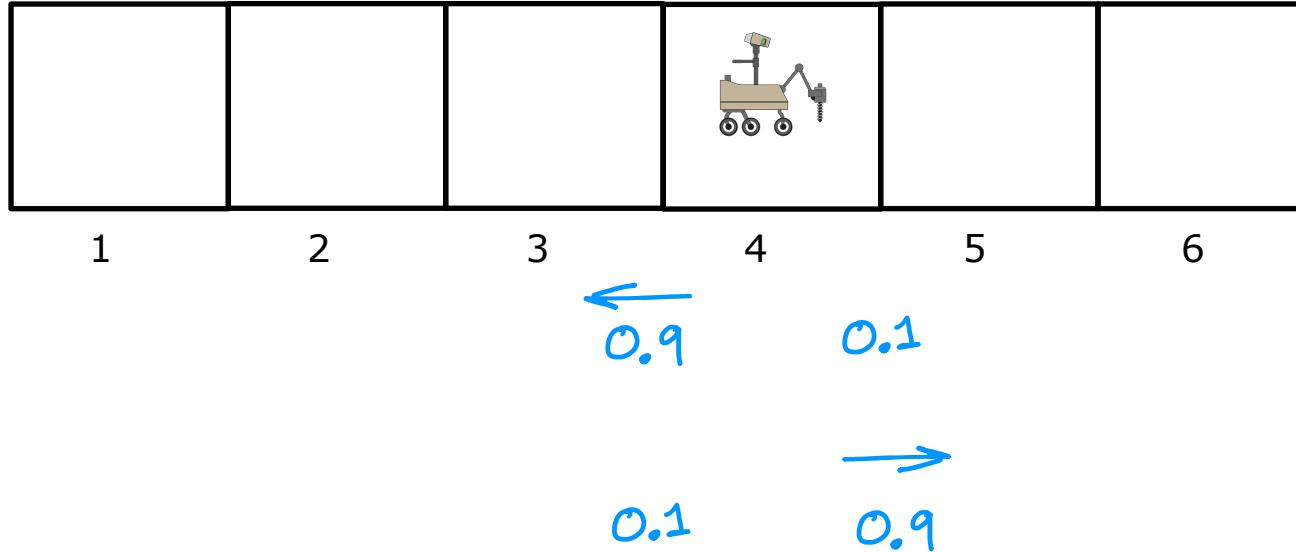
$$= R(4) + (0.5) \max_{a'} Q(3, a')$$



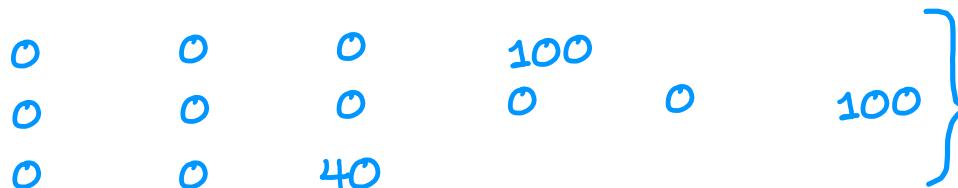
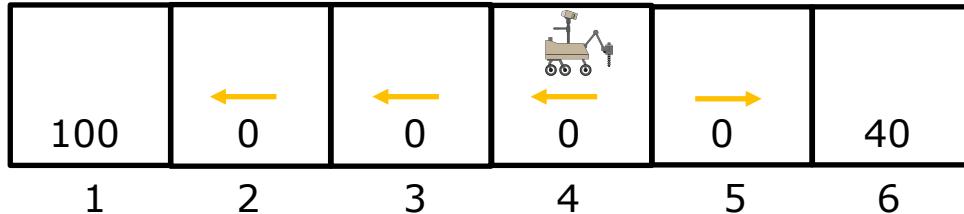
State-action value function

Random (stochastic)
environment (Optional)

Stochastic Environment



Expected Return



Expected Return = Average($R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots$)
= $E[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots]$

Expected Return

Goal of Reinforcement Learning:

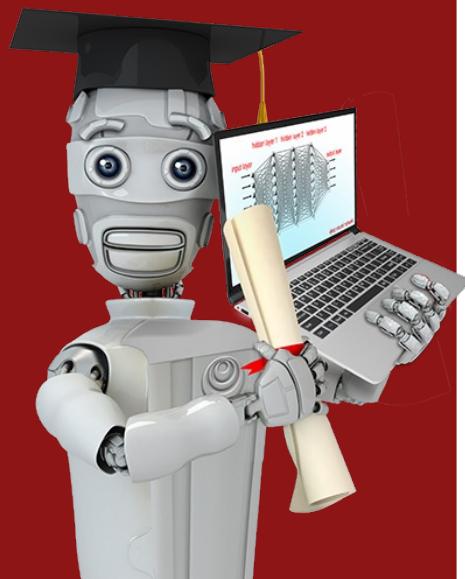
Choose a policy $\pi(s) = a$ that will tell us what action a to take in state s so as to maximize the expected return.

Bellman
Equation:

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

The diagram illustrates the Bellman equation with blue annotations. It shows the components of the equation: $Q(s, a)$, $R(s)$, γ , $E[\max_{a'} Q(s', a')]$, and the individual terms 3 , 2 , and 4 . Blue arrows point from the labels to their corresponding parts in the equation. A blue bracket groups the term $R(s) + \gamma E[\max_{a'} Q(s', a')]$. Another blue bracket groups the term $\max_{a'} Q(s', a')$. A third blue bracket groups the numbers 3 , 2 , and 4 .

Jupyter Notebook



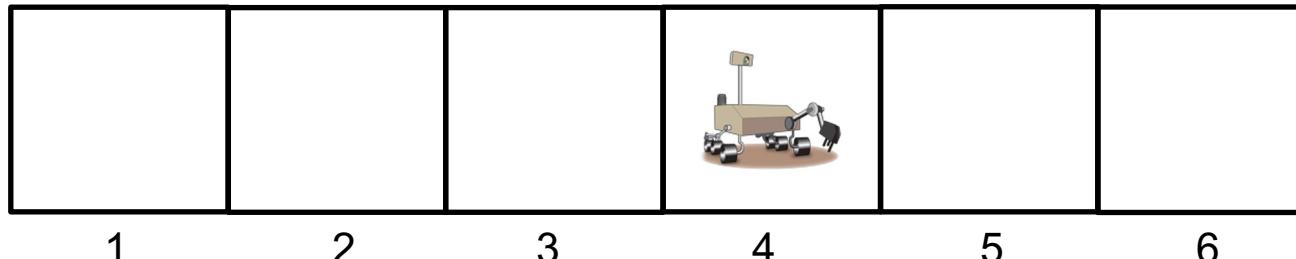
Continuous State Spaces

Example of continuous state applications

Discrete vs Continuous State

Discrete State:

$$S = 1, 2, 3, 4, 5, 6$$



Continuous State:



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

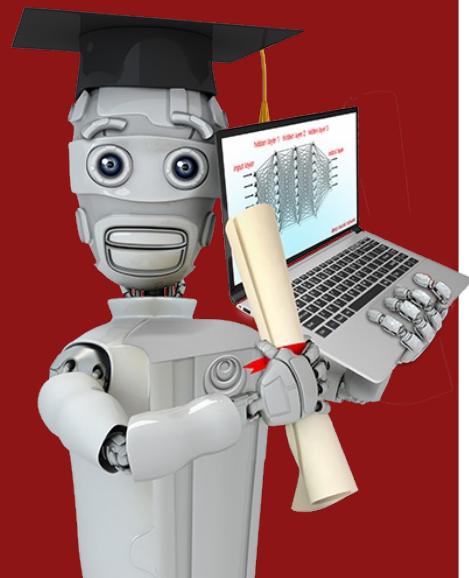
A handwritten note next to the vector indicates $0 - 360^\circ$ with an arrow pointing to the angle θ .

Autonomous Helicopter



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

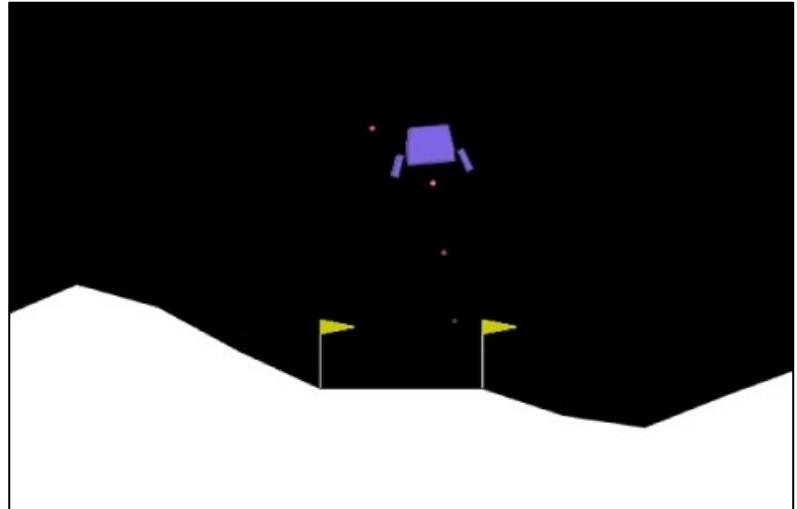
roll
pitch
yaw



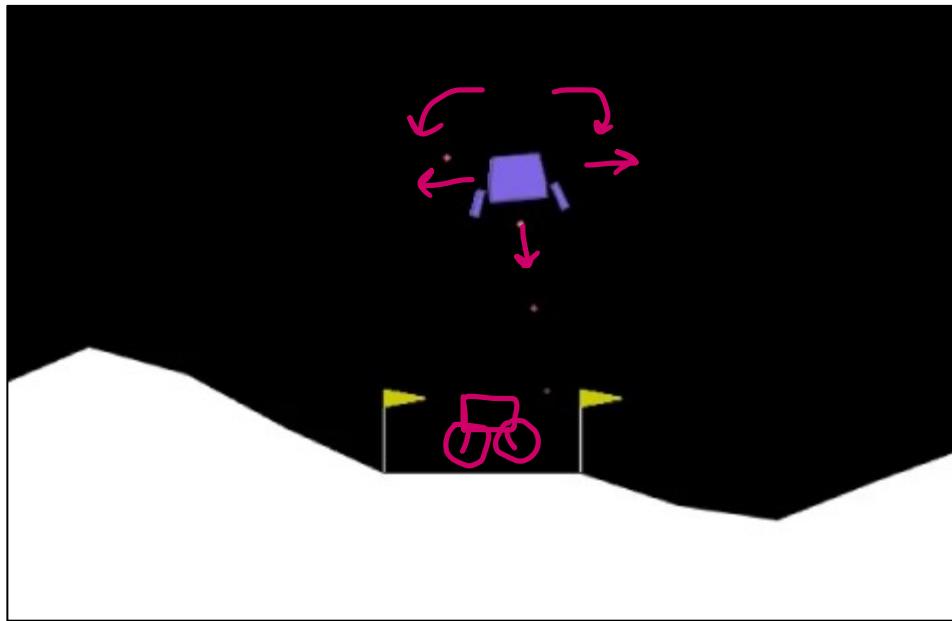
Continuous State Spaces

Lunar Lander

Lunar Lander



Lunar Lander



actions:

do nothing

left thruster

main thruster

right thruster

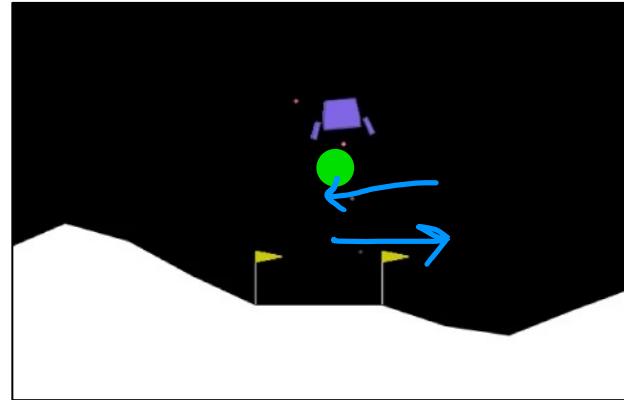
$s =$

0 or 1

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

Reward Function

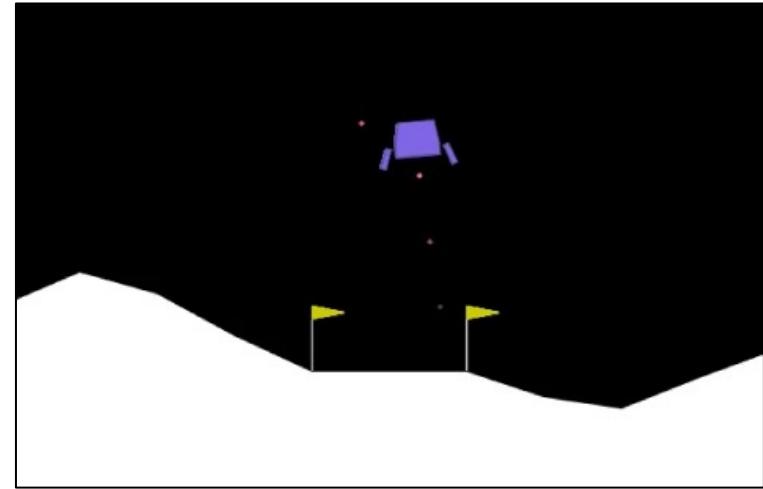
- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03



Lunar Lander Problem

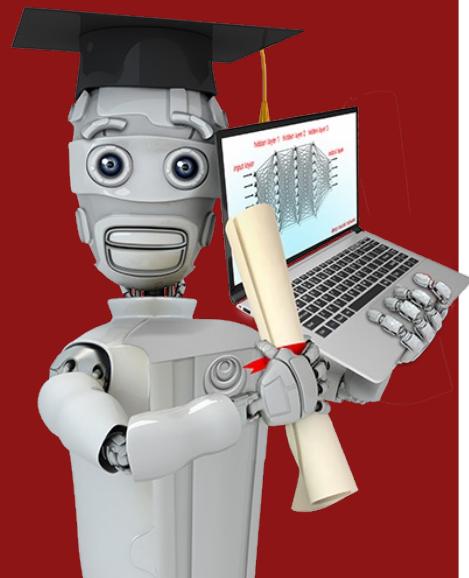
- Learn a policy π that, given

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



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

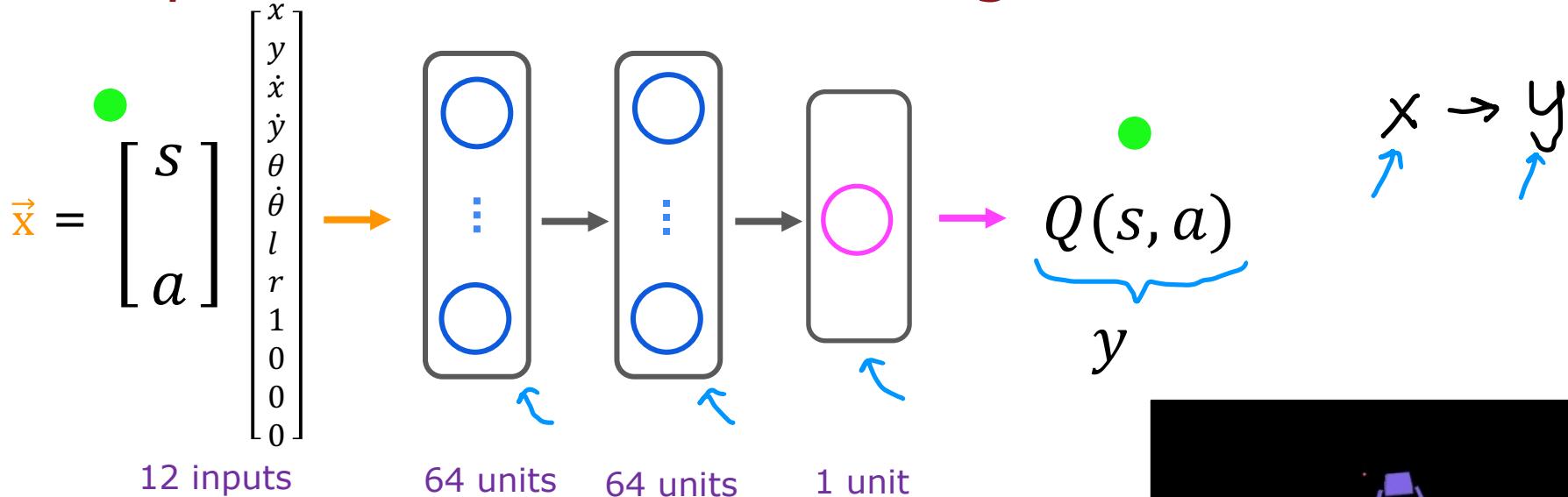
$$\gamma = 0.985$$



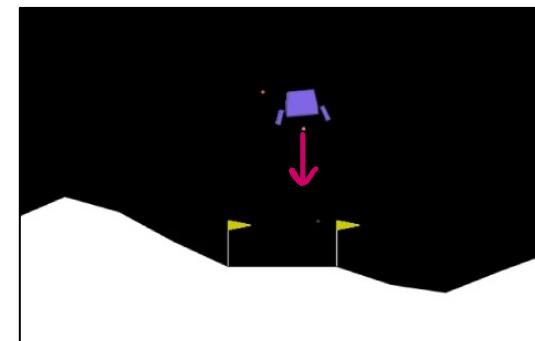
Continuous State Spaces

Learning the state-value function

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 Equation

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

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

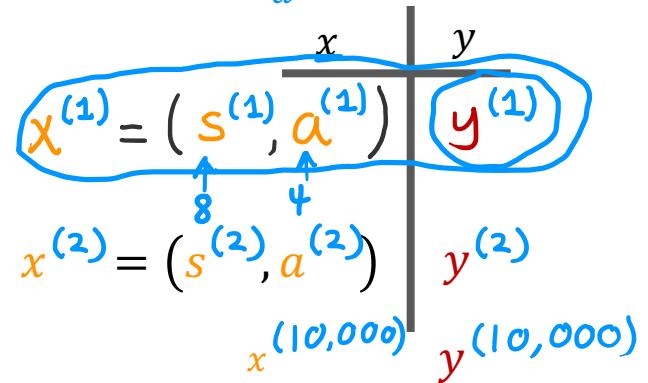
$$(s, a, R(s), s')$$

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

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

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

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



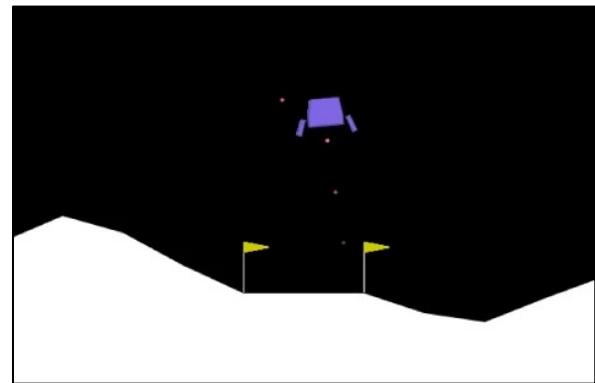
Learning Algorithm

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

Repeat {

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

Store 10,000 most recent $\underline{(s, a, R(s), s')}$ tuples.



replay buffer

Train neural network:

Create training set of 10,000 examples using

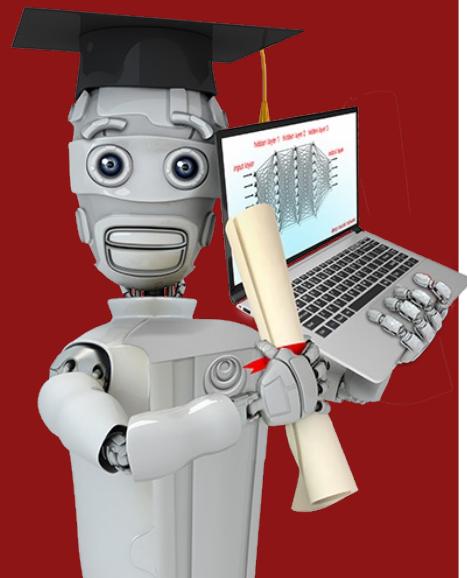
$$x = \underline{(s, a)} \text{ and } y = R(s) + \gamma \max_{a'} Q(s', a')$$

Train \underline{Q}_{new} such that $\underline{Q}_{new}(s, a) \approx y$. $f_{w, \beta}(x) \approx y$

Set $\underline{Q} = \underline{Q}_{new}$.

$x^{(1)}, y^{(1)}$
 x, y ;
 $x^{(100000)}, y^{(100000)}$

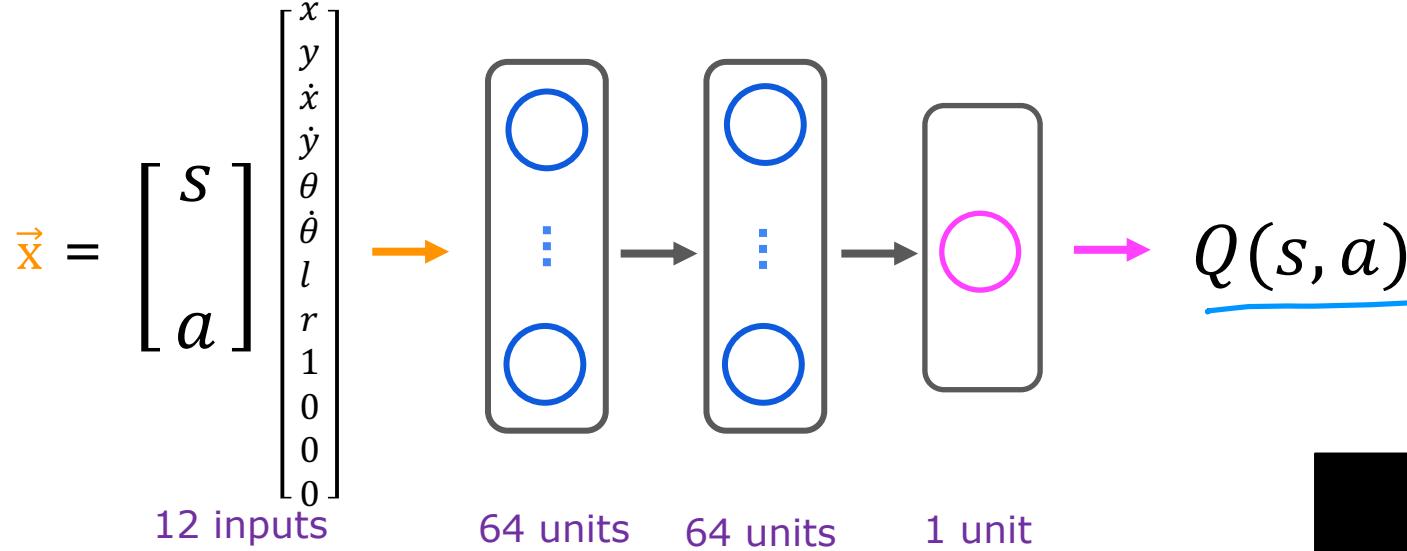
[Mnih et al., 2015, [Human-level control through deep reinforcement learning](#)]



Continuous State Spaces

Algorithm refinement:
Improved neural network
architecture

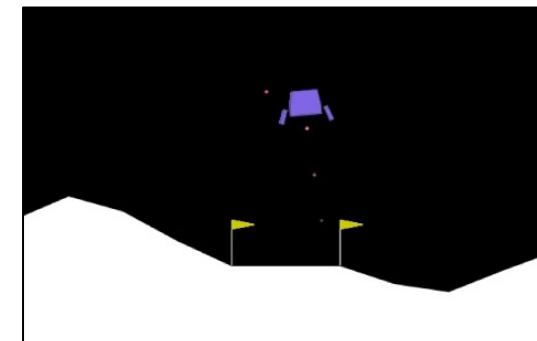
Deep Reinforcement Learning



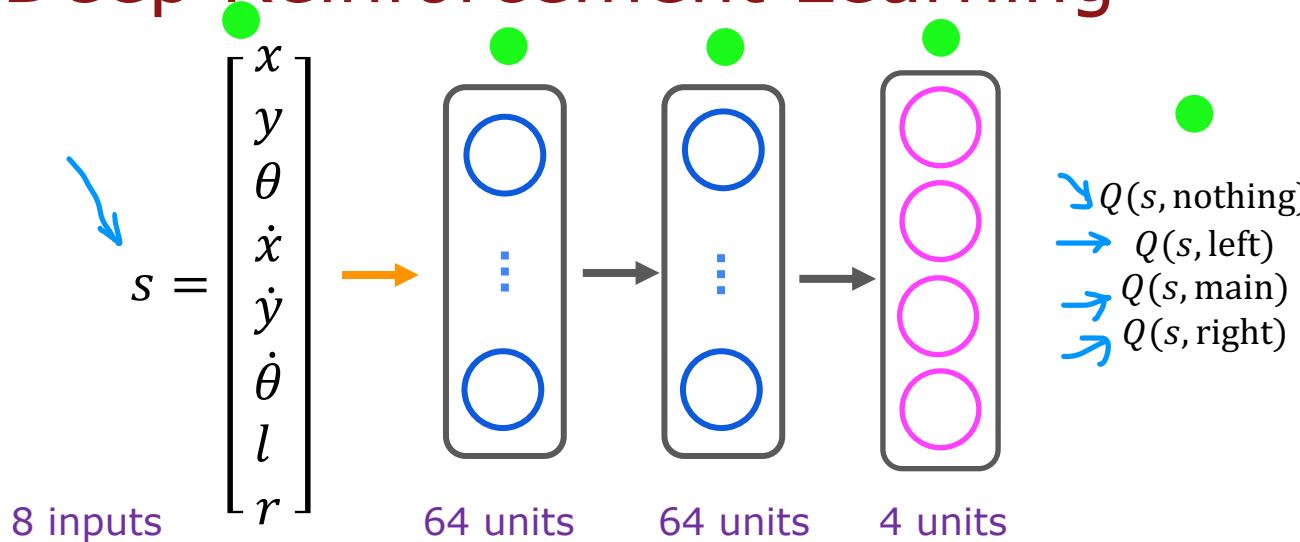
In a state s , use neural network to compute

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

Pick the action a that maximizes $\underline{Q(s, a)}$

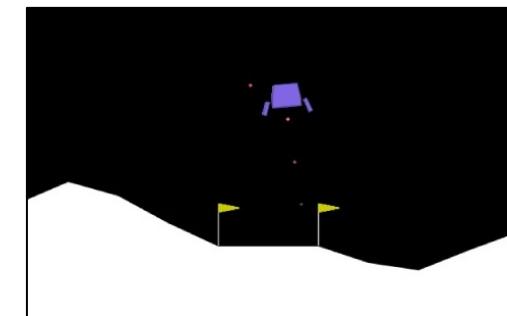


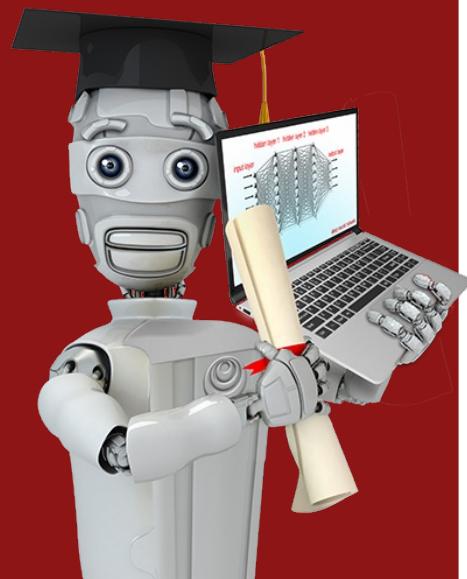
Deep Reinforcement Learning



In a state \underline{s} , input \underline{s} to neural network.

Pick the action \underline{a} that maximizes $\underline{Q}(s, a)$. $R(s) + \gamma \max_{a'} Q(s', a')$





Continuous State Spaces

Algorithm refinement:
 ϵ -greedy policy

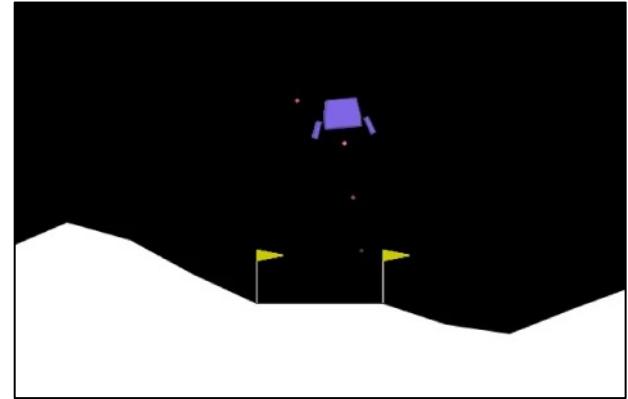
Learning Algorithm

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

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

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

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

what if init $Q(s, \text{main})$ to be low?

Option 1: \rightarrow will never try other actions \hookrightarrow will never fire main thrusters!

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**

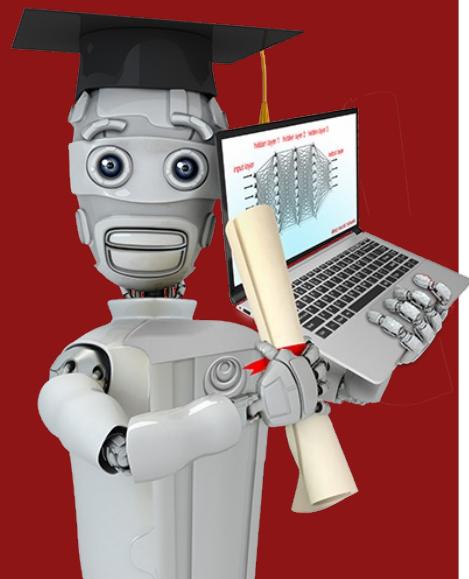
With probability 0.05, pick an action a randomly. **exploration**

ϵ -greedy policy ($\epsilon = 0.05$)

greedy 95% of the time

exploring 5% of the time

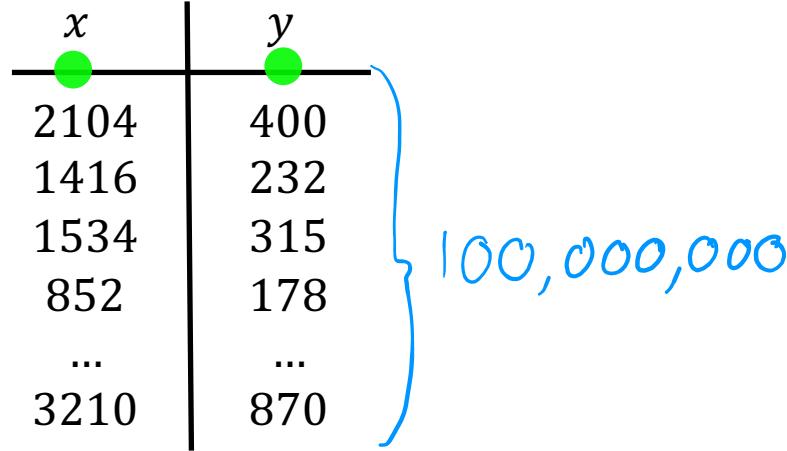
start ϵ high
gradually decrease
 $1.0 \rightarrow 0.01$



Continuous State Spaces

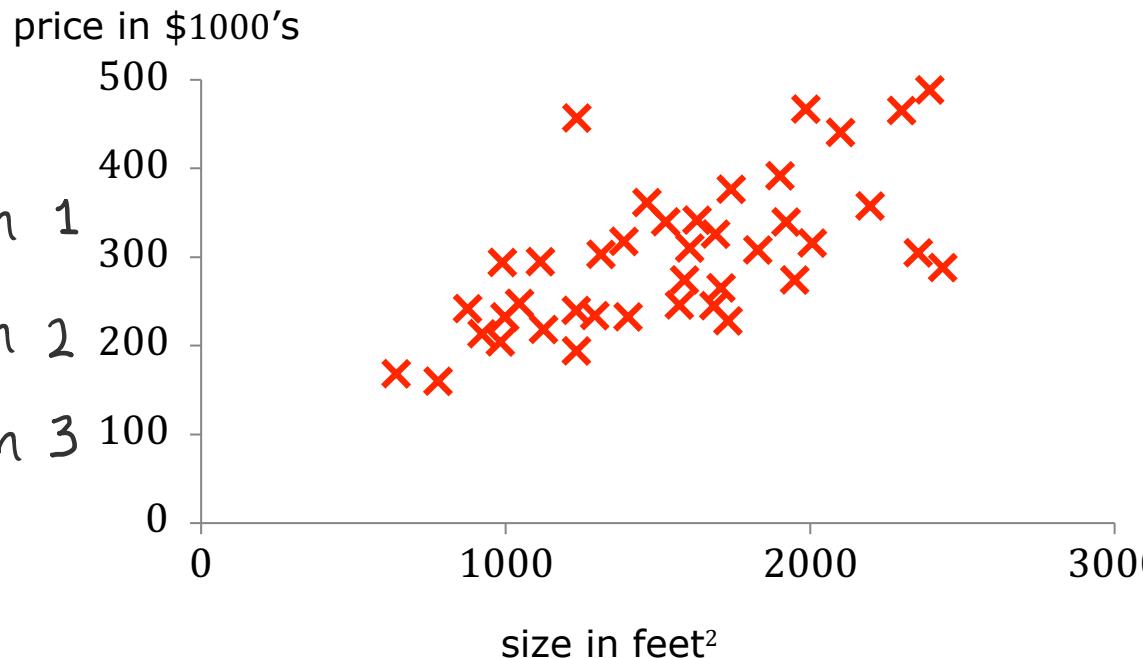
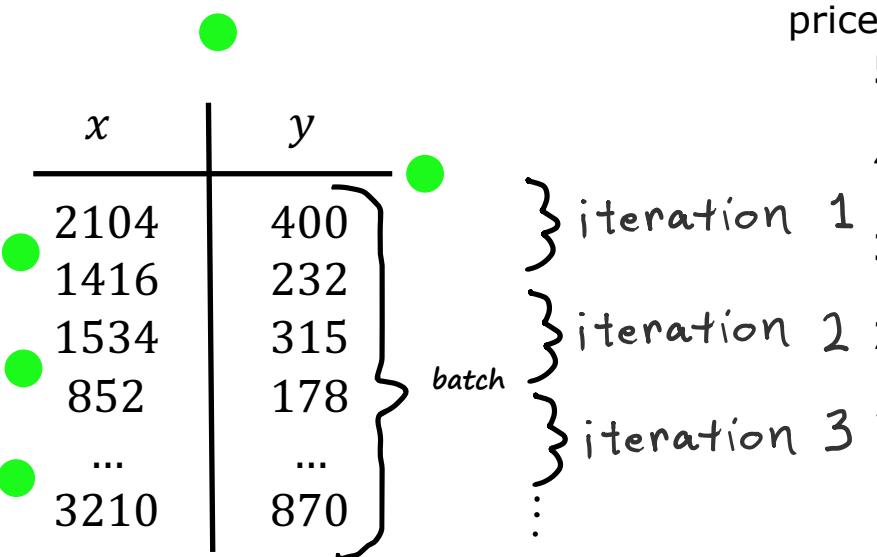
Algorithm refinement:
Mini-batch and soft update
(optional)

How to choose actions while still learning?



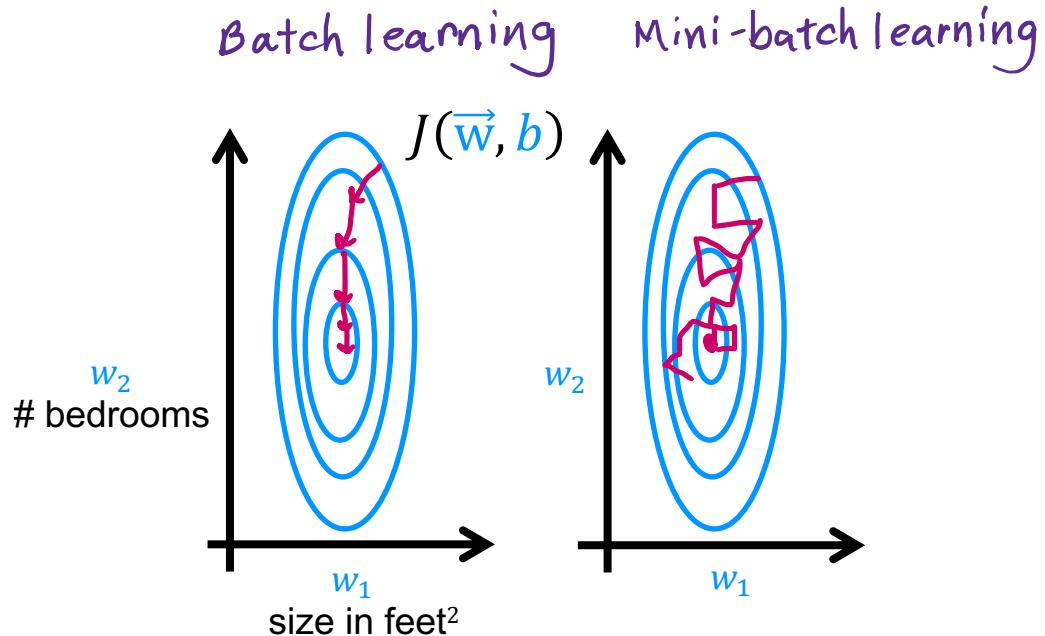
repeat {
 $J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$
 $m = 100,000,000$
 $m' = 1,000$
 $w = w - \alpha \frac{\partial}{\partial w} \left[\frac{1}{2m'} \sum_{i=1}^{m'} (f_{w,b}(x^{(i)}) - y^{(i)})^2 \right]$
 $b = b - \alpha \frac{\partial}{\partial b} \left[\frac{1}{2m'} \sum_{i=1}^{m'} (f_{w,b}(x^{(i)}) - y^{(i)})^2 \right]$
}

Mini-batch



Mini-batch

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



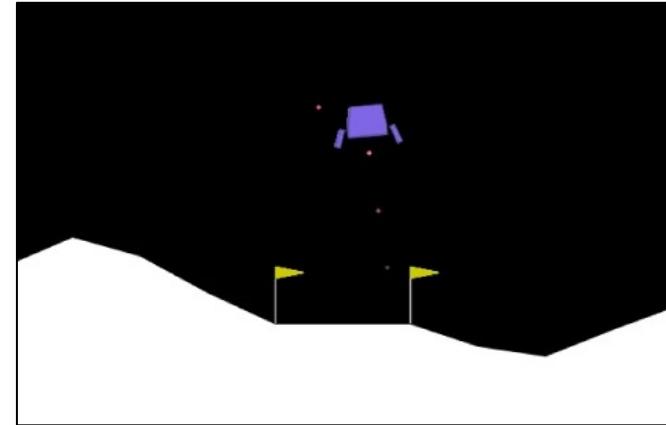
Learning Algorithm

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

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

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

Set $Q = Q_{new}$.

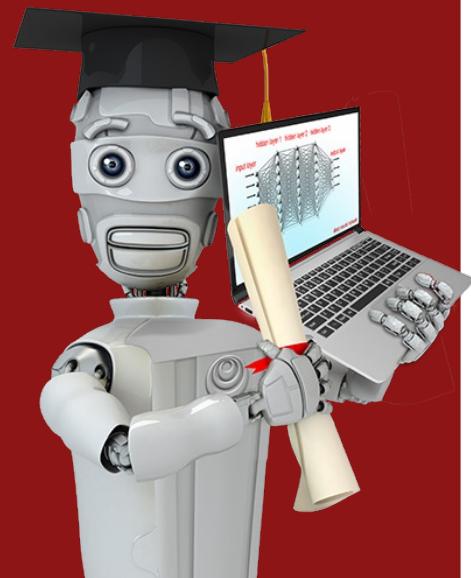
$x^{(1)}, y^{(1)}$
⋮
 $x^{(1000)}, y^{(1000)}$

Soft Update

Set $Q = Q_{new}$. \leftarrow $Q(s, a)$

\uparrow \uparrow
 W, B W_{new}, B_{new}

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

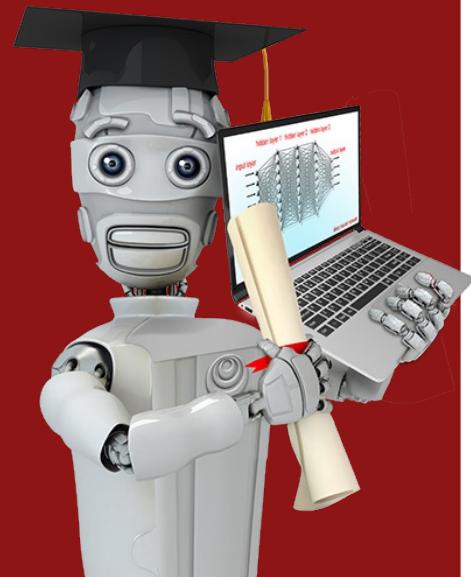


Continuous State Spaces

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.



Conclusion

Summary and
Thank you

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

Thank you

Andrew Ng.