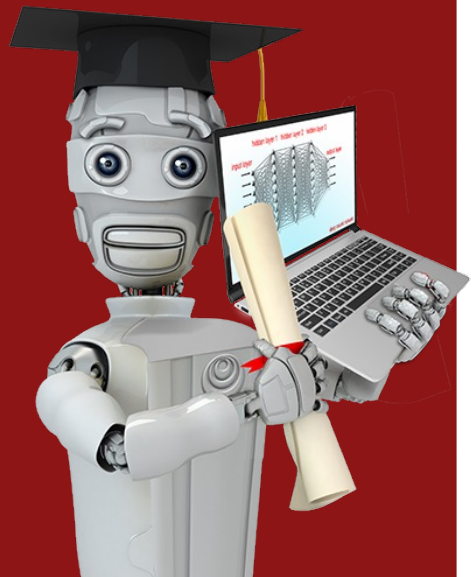
Copyright Notice

DeepLearning.AI

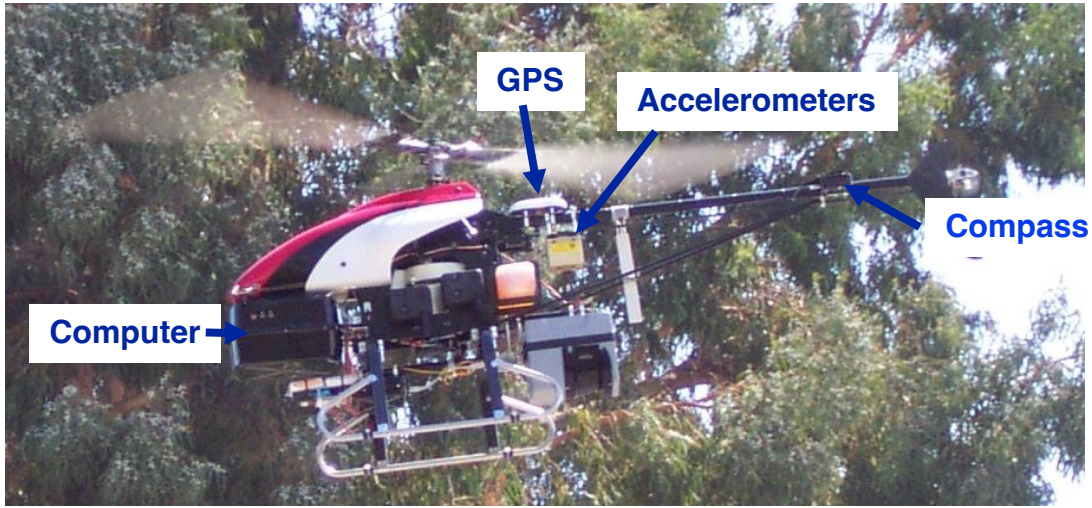
Stanford ONLINE



Reinforcement Learning Introduction

What is Reinforcement Learning?

Autonomous Helicopter



How to fly it?

Autonomous Helicopter

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

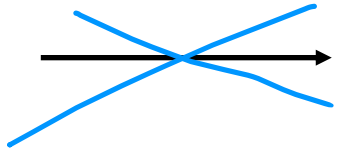
→ For more videos: http://heli.stanford.edu.

Reinforcement Learning

position of helicopter → 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

Robotic Dog Example

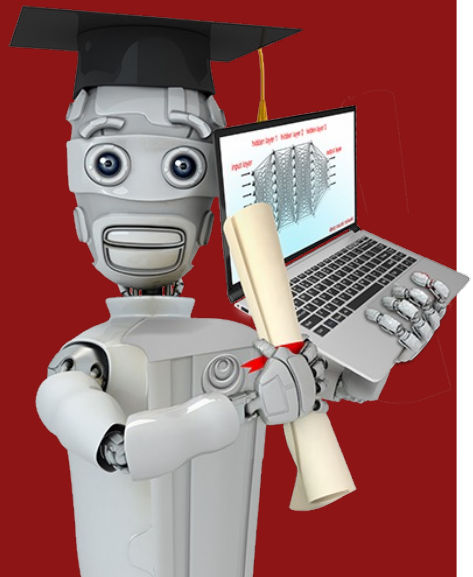
[Thanks to Zico Kolter]

Applications

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

DeepLearning.AI

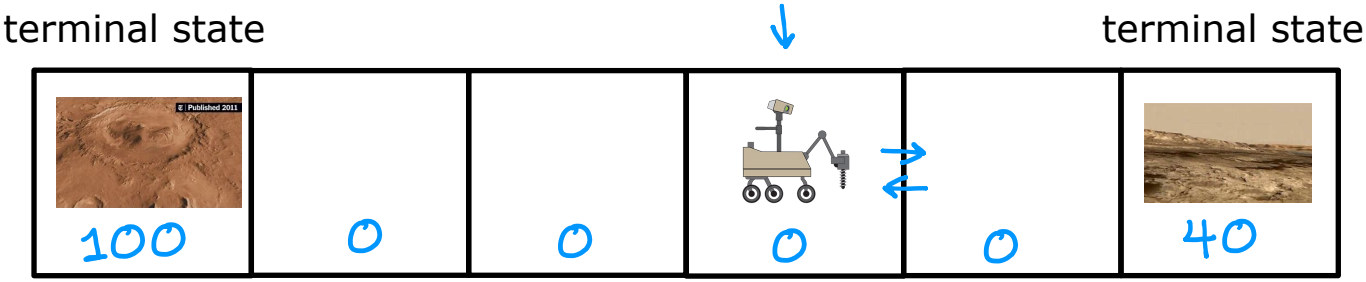
Stanford ONLINE

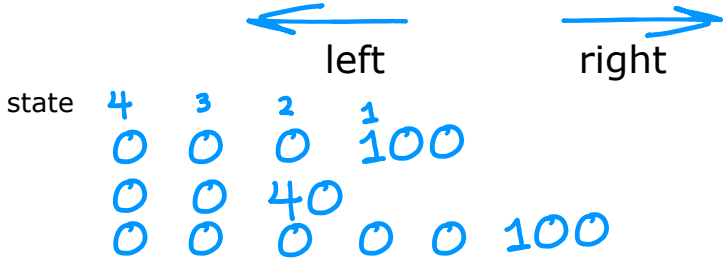


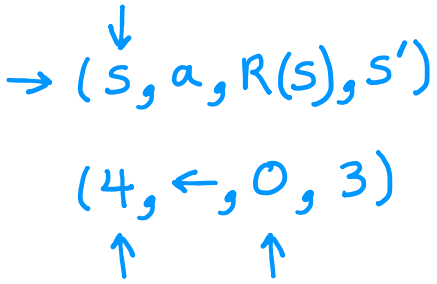
Reinforcement Learning formalism

Mars rover example

Mars Rover Example



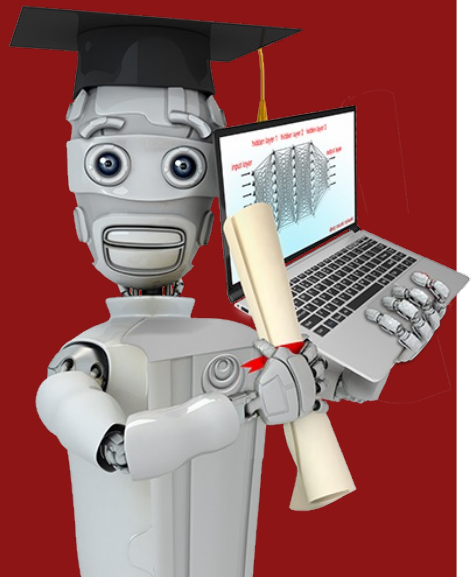




[Credit: Jagriti Agrawal, Emma Brunskill]

DeepLearning.AI

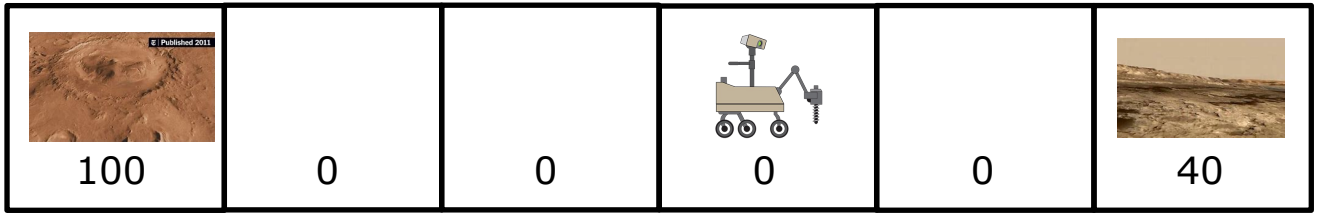
Stanford ONLINE



Reinforcement Learning formalism

The Return in reinforcement learning

Return



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

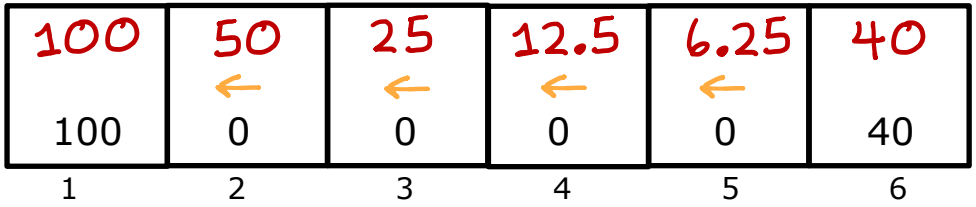
Return $= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$ (until terminal state)

Discount Factor $\gamma = 0.9 \quad 0.99 \quad 0.999$

$\gamma = 0.5$

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

Example of Return

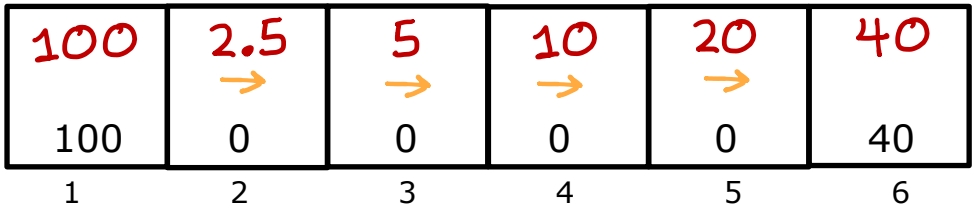


← return

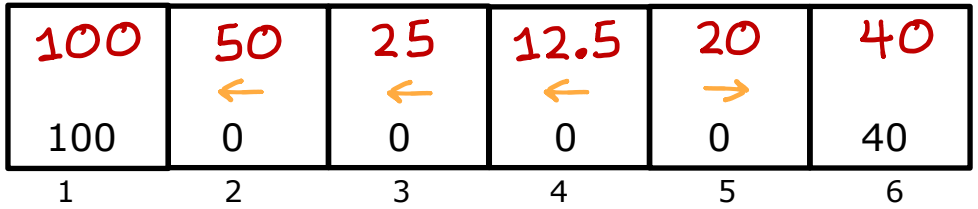
← reward

$\gamma = 0.5$

The return depends on the actions you take.



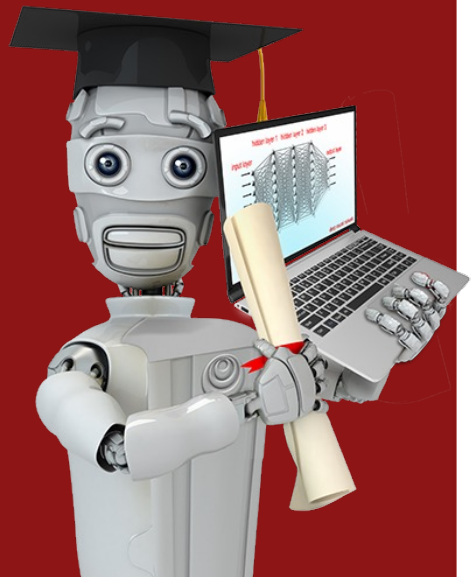
$0 + (0.5)0 + (0.5)^2 40 = 10$



$0 + (0.5)40 = 20$

DeepLearning.AI

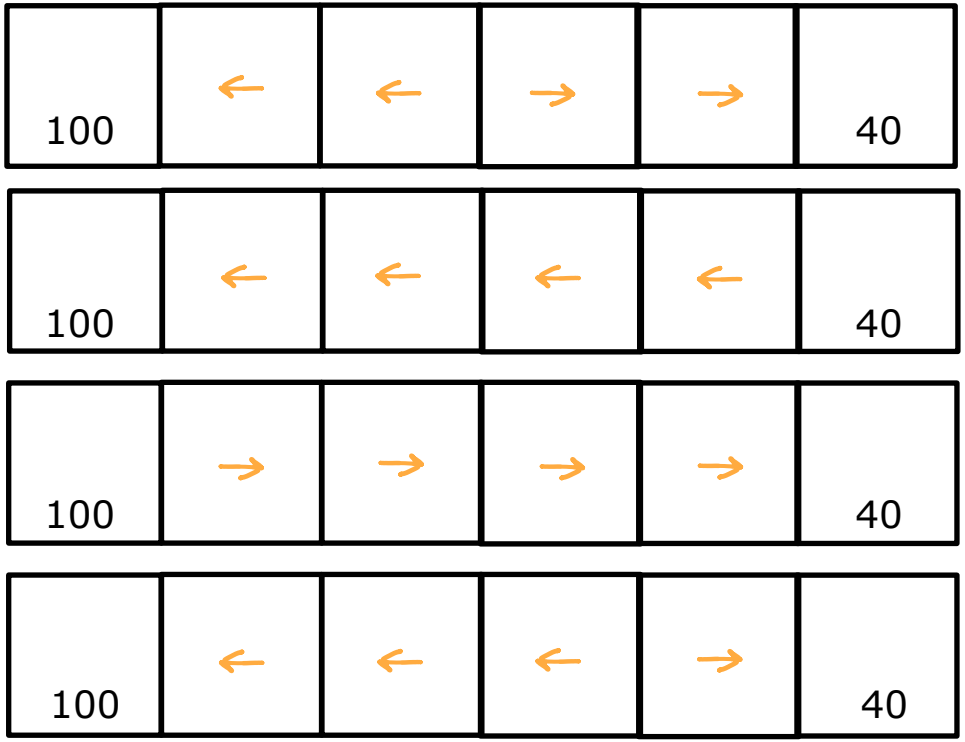
Stanford ONLINE



Reinforcement Learning formalism

Making decisions: Policies in reinforcement learning

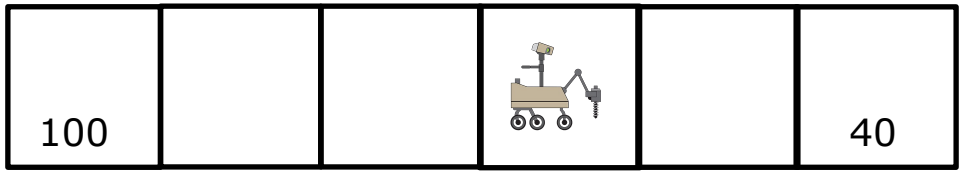
Policy



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

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*.

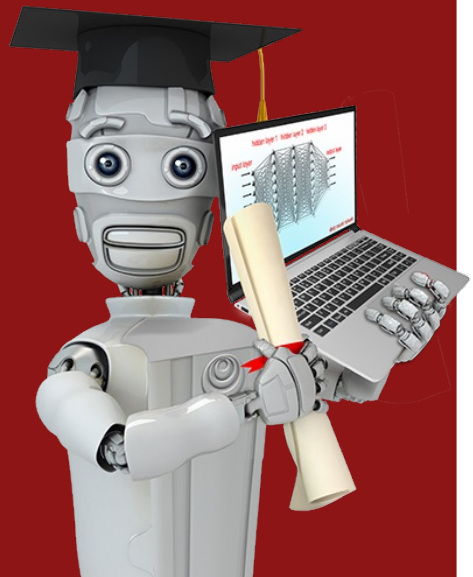
The goal of reinforcement learning



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

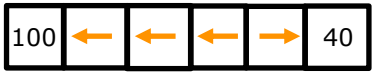
DeepLearning.AI

Stanford ONLINE

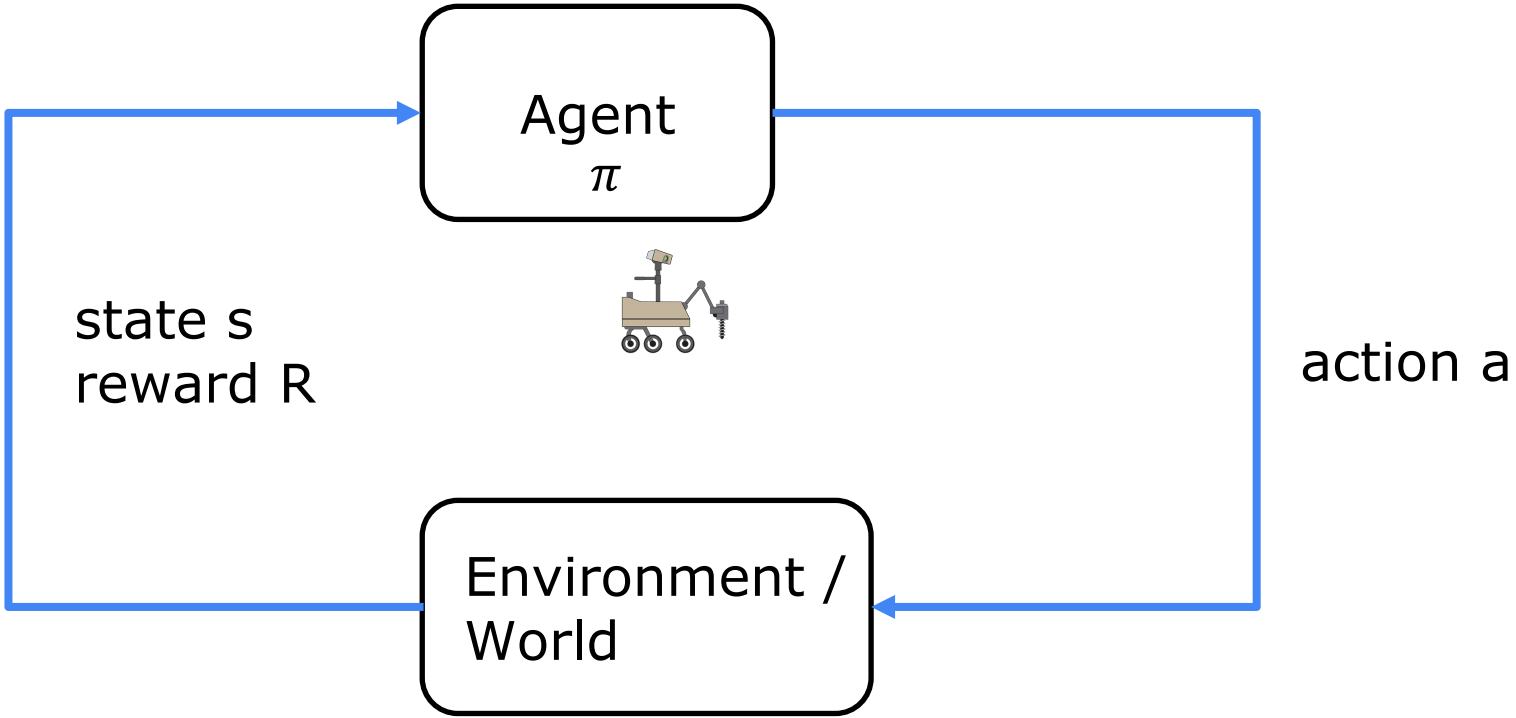


Reinforcement Learning formalism

Review of key concepts

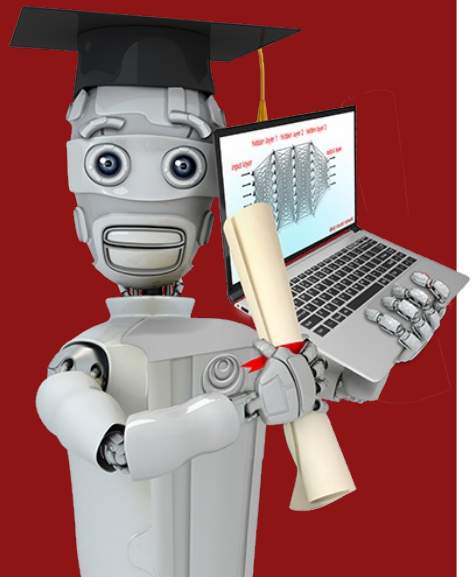
	Mars rover 	Helicopter 	Chess 
states	6 states	position of helicopter	pieces on board
actions	← →	how to move control stick	possible move
rewards	100, 0, 40	+1, −1000	+1, 0, −1
discount factor γ	0.5	0.99	0.995
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$
policy π		Find $\pi(s) = a$	Find $\pi(s) = a$

Markov Decision Process (MDP)



DeepLearning.AI

Stanford ONLINE



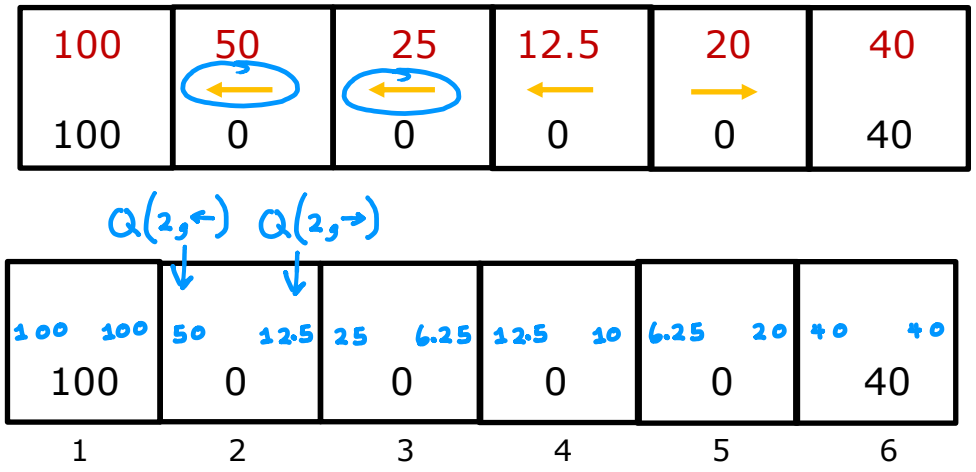
State-action value function

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.

$Q(s,a)$



← return
← action
← reward

$Q(2,\rightarrow) = 12.5$

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

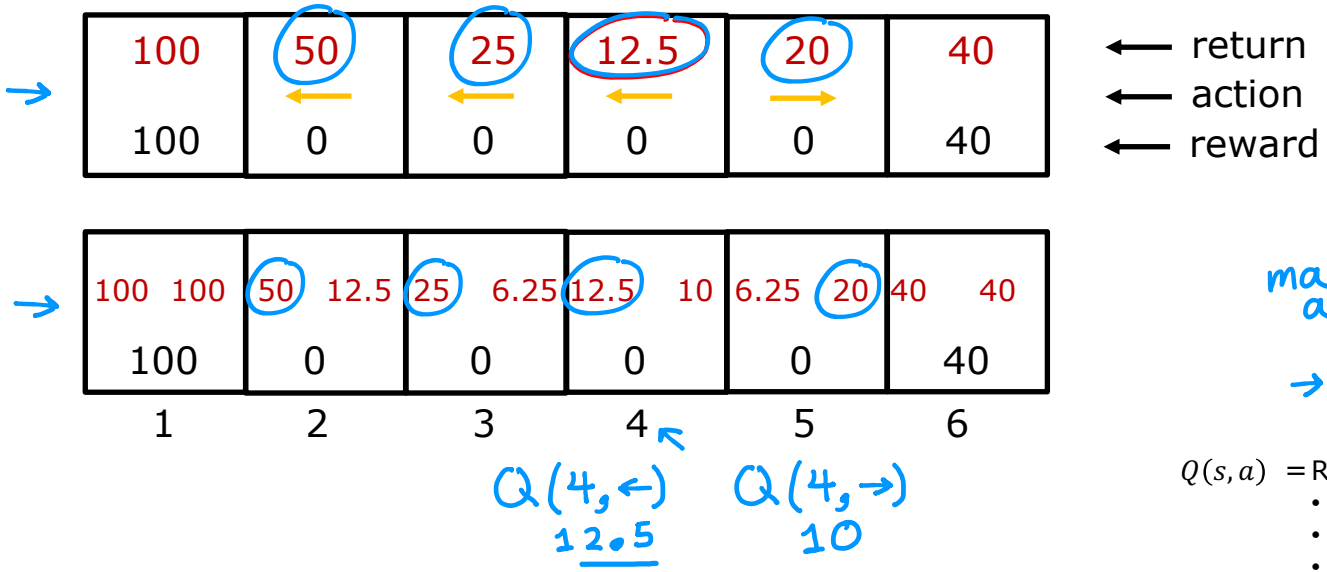
$Q(2,\leftarrow) = 50$

$0+(0.5)100$

$Q(4,\leftarrow) = 12.5$

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

Picking actions



← return
← action
← reward

$\max_a Q(s, a)$

$\rightarrow \pi(s) = a$

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

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

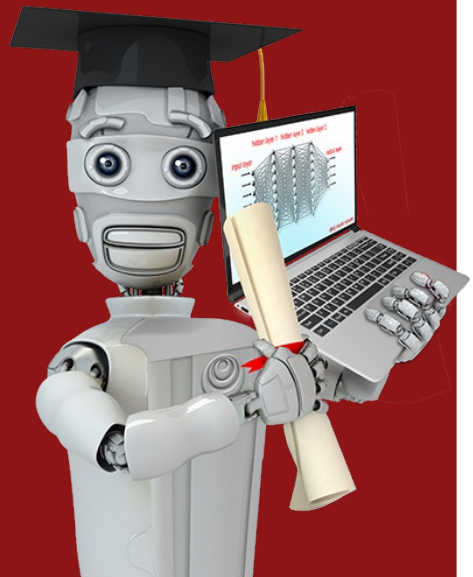
The best possible action in state s is the action a that gives $\max_a Q(s, a)$.

Q^*

Optimal Q function

DeepLearning.AI

Stanford ONLINE



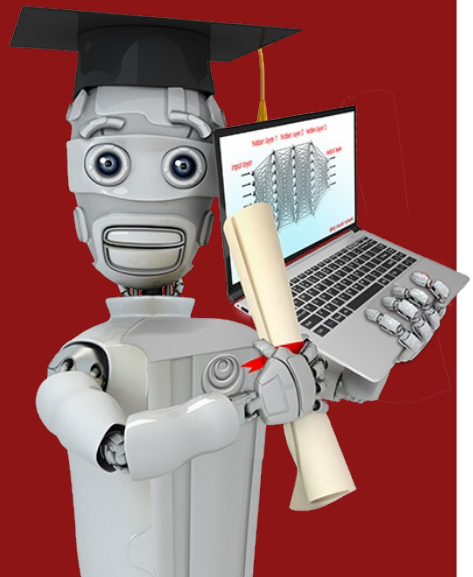
State-action value function

State-action value function example

Jupyter Notebook

DeepLearning.AI

Stanford ONLINE

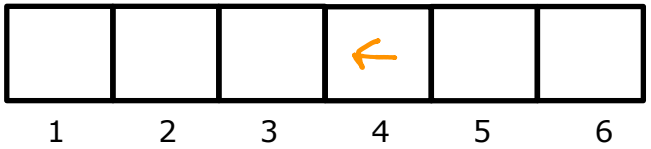


State-action value function

Bellman Equation

Bellman Equation

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



s : current state
a : current action

$R(s)$ = reward of current state

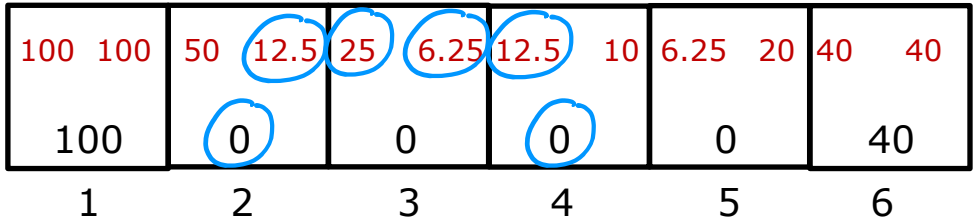
s' : state you get to after taking action a
a' : action that you take in state s'

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

Bellman Equation

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

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



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

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

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

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

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

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

Reward you get right away

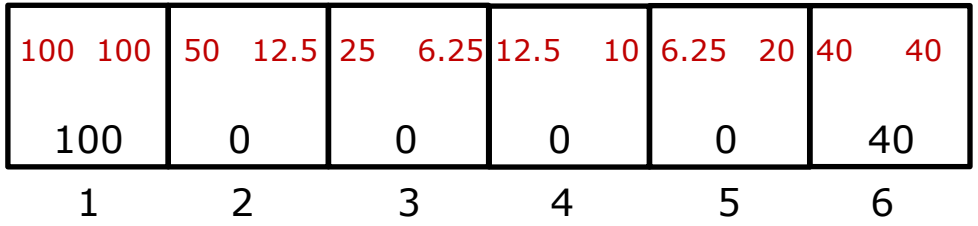
Return from behaving optimally starting from state s'.

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

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

Explanation of Bellman Equation

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



$Q(4,\leftarrow)$

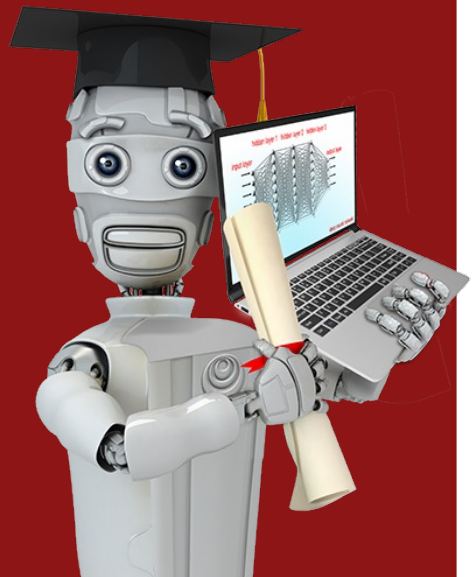
$= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

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

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

DeepLearning.AI

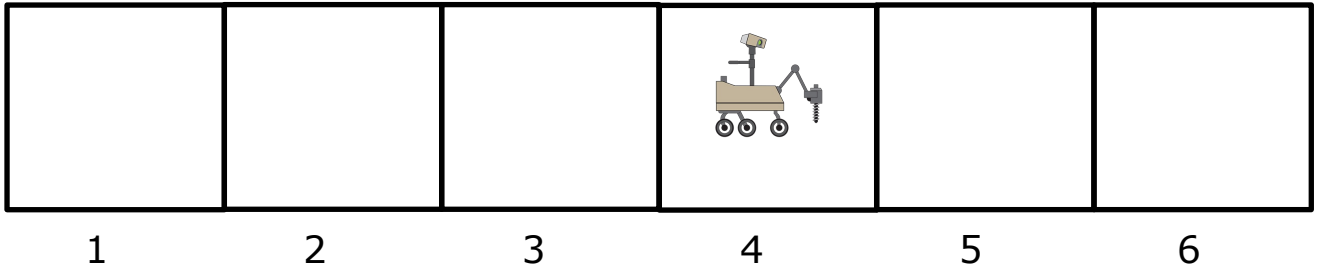
Stanford ONLINE



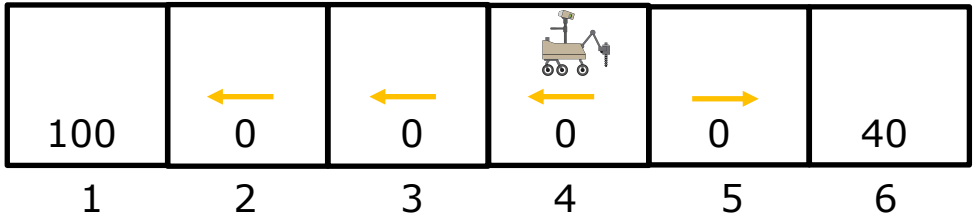
State-action value function

Random (stochastic) environment (Optional)

Stochastic Environment



Expected Return



0 0 0 100 0 100
0 0 0 0
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:

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

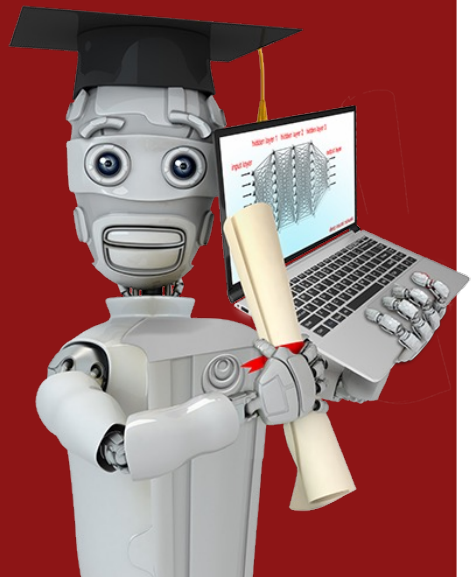
3 ←

2 or 4

Jupyter Notebook

DeepLearning.AI

Stanford ONLINE

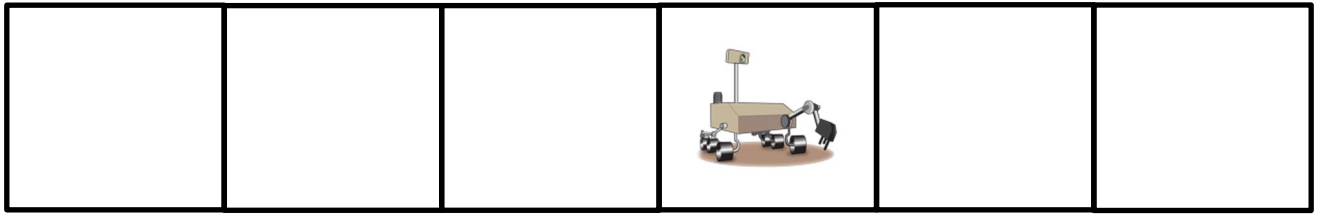


Continuous State Spaces

Example of continuous state applications

Discrete vs Continuous State

Discrete State: $S = 1, 2, 3, 4, 5, 6$



Continuous State:

$$S = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \leftarrow 0\text{-}360^\circ$$

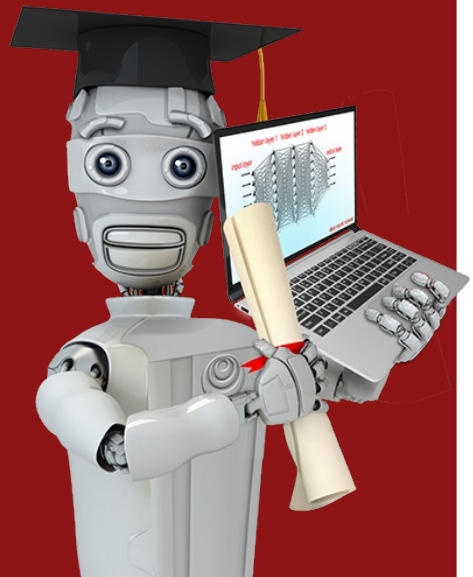
Autonomous Helicopter

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

roll
pitch
yaw

DeepLearning.AI

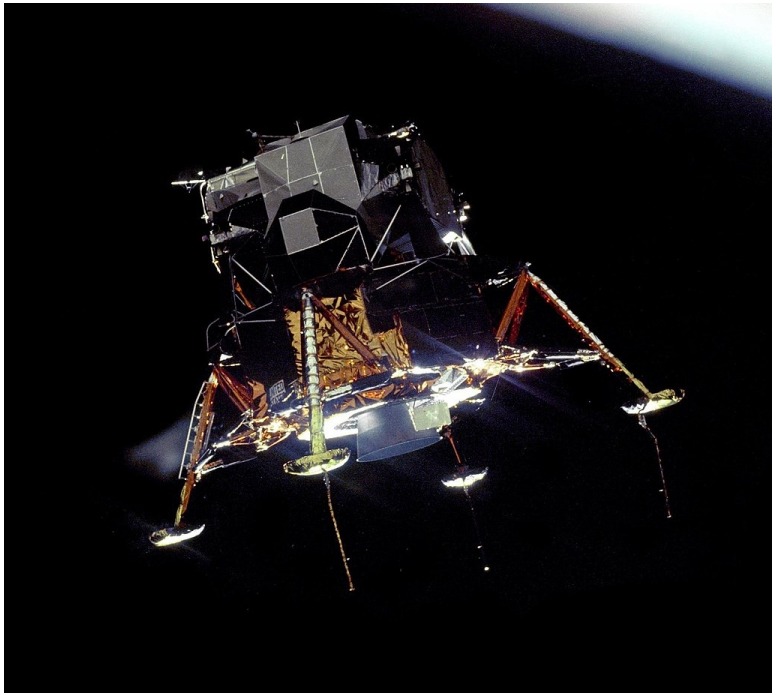
Stanford ONLINE

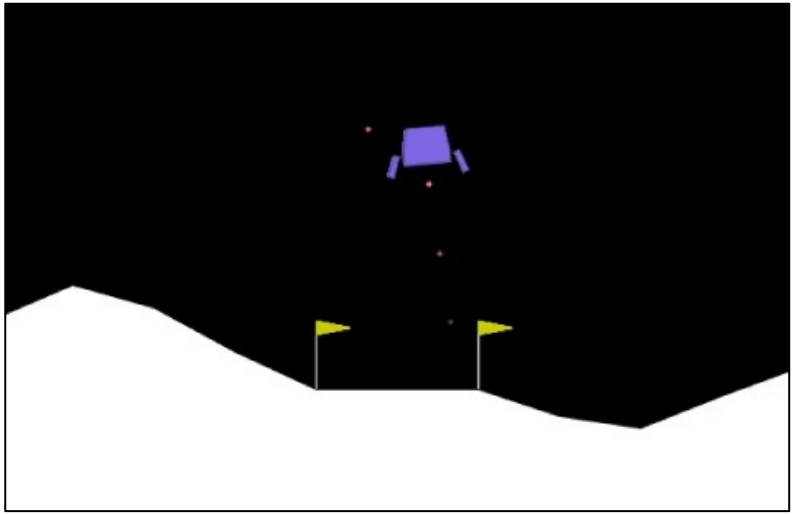


Continuous State Spaces

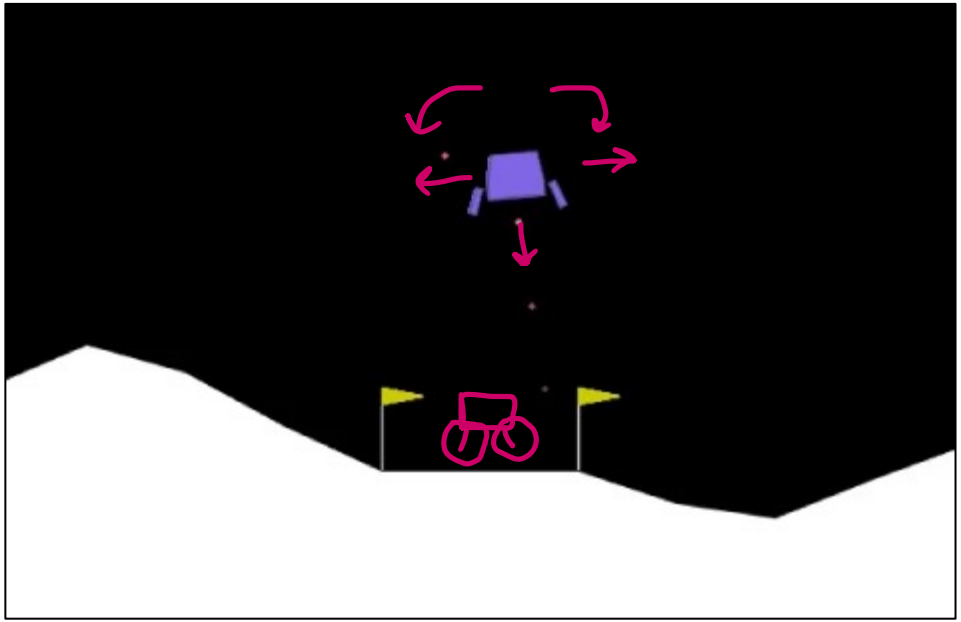
Lunar Lander

Lunar Lander





Lunar Lander



actions:

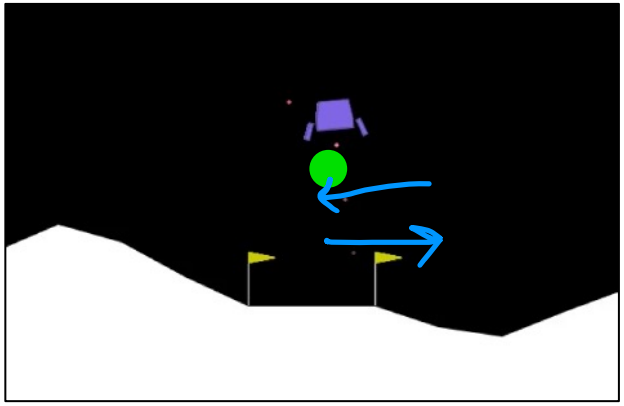
- do nothing
- left thruster
- main thruster
- right thruster

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

0 or 1

Reward Function

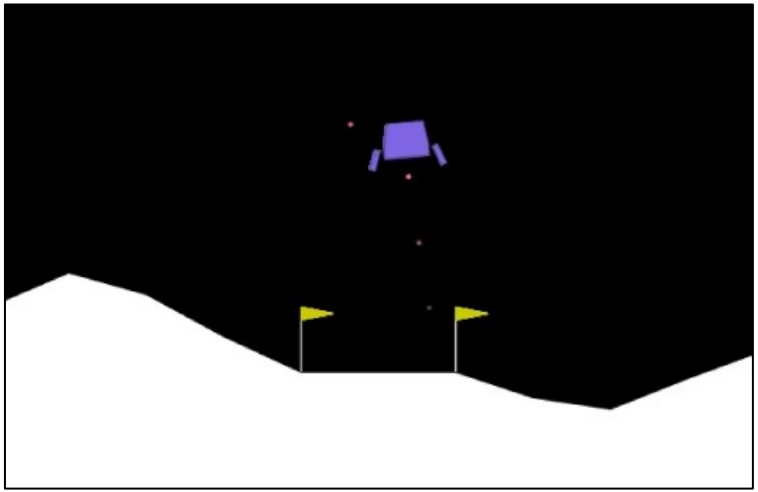
- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03



Lunar Lander Problem

Learn a policy π that, given

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

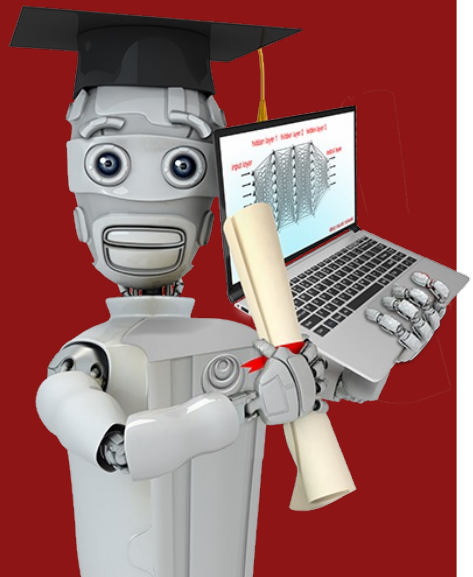


picks action $a = \pi(s)$ so as to maximize the return.

$$\gamma = 0.985$$

DeepLearning.AI

Stanford ONLINE

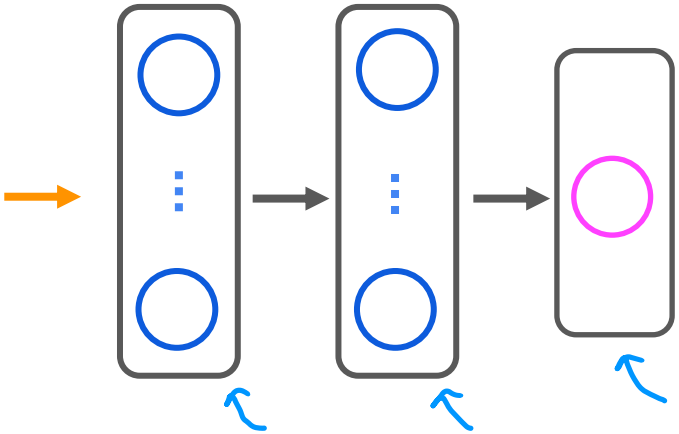


Continuous State Spaces

Learning the state-value function

Deep Reinforcement Learning

$$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$Q(s,a)$

y

$x \rightarrow y$

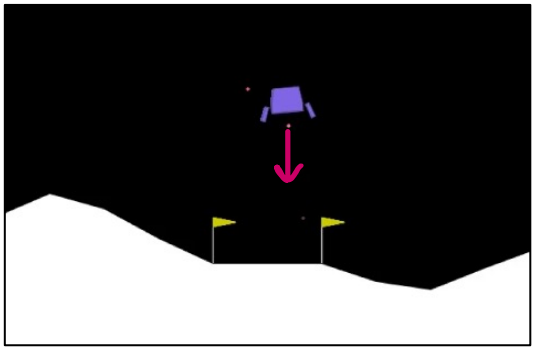
12 inputs

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 $Q(s,a)$



Bellman Equation

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

x $\quad$ y

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

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

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

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

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

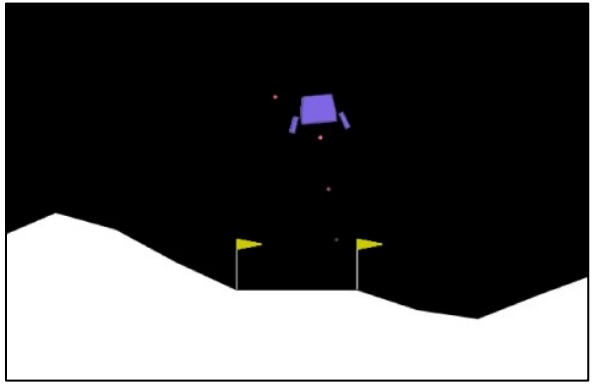
$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)})$ ←

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

x	y
$x^{(1)} = (s^{(1)}, a^{(1)})$	$y^{(1)}$
$x^{(2)} = (s^{(2)}, a^{(2)})$	$y^{(2)}$
$x^{(10,000)}$	$y^{(10,000)}$

8 $\quad$ 4

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

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

Set $Q = Q_{new}$.

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

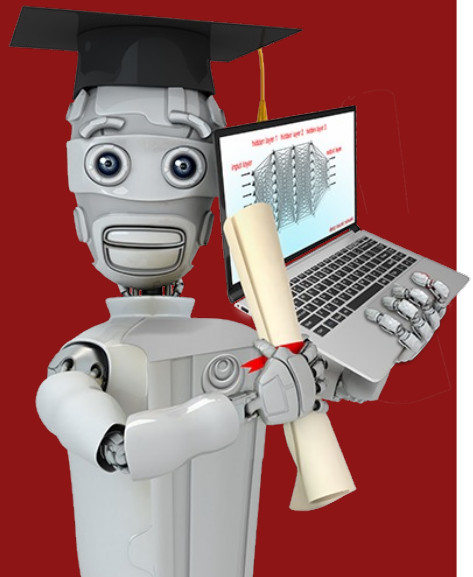
x, y

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

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

DeepLearning.AI

Stanford ONLINE

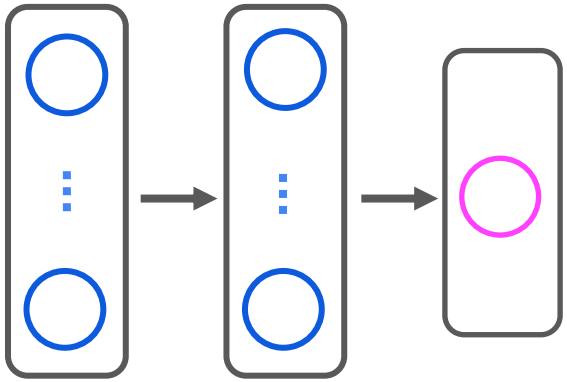


Continuous State Spaces

Algorithm refinement: Improved neural network architecture

Deep Reinforcement Learning

$$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$Q(s, a)$

12 inputs

64 units

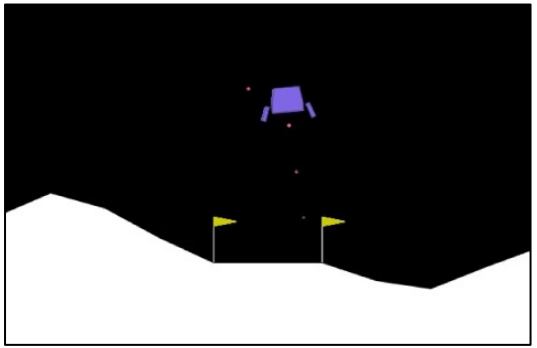
64 units

1 unit

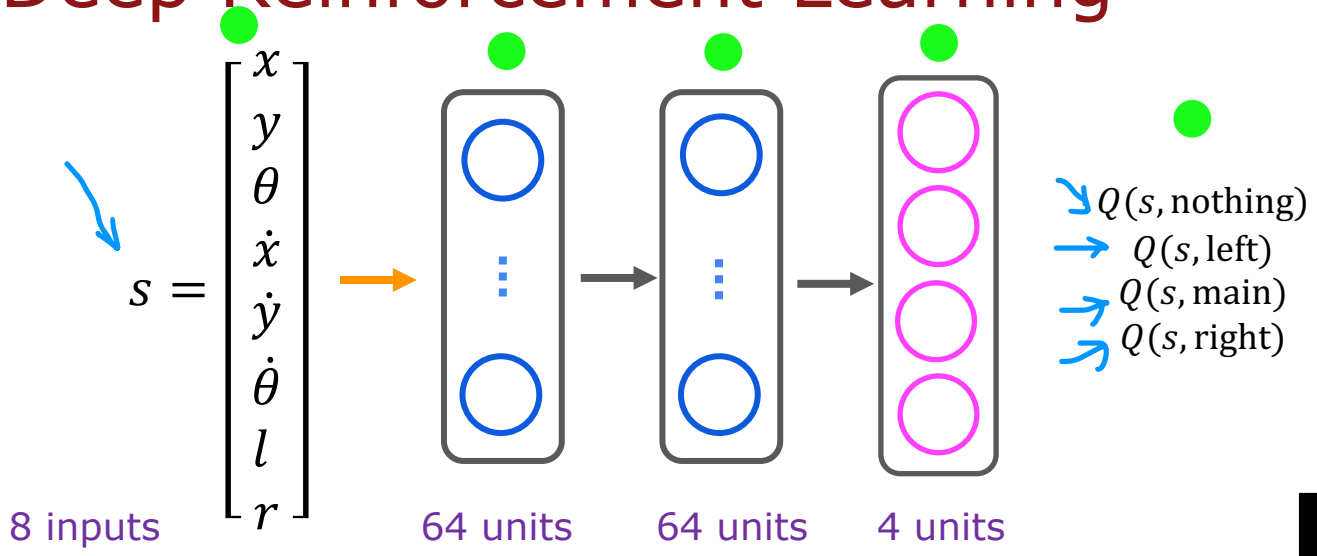
In a state s, use neural network to compute

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

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

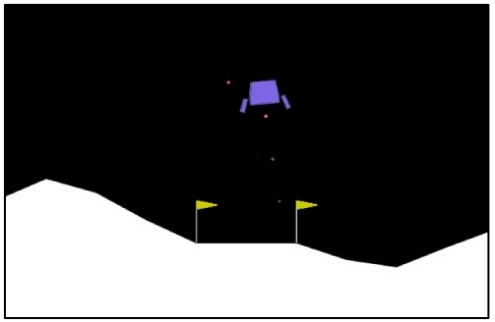


Deep Reinforcement Learning



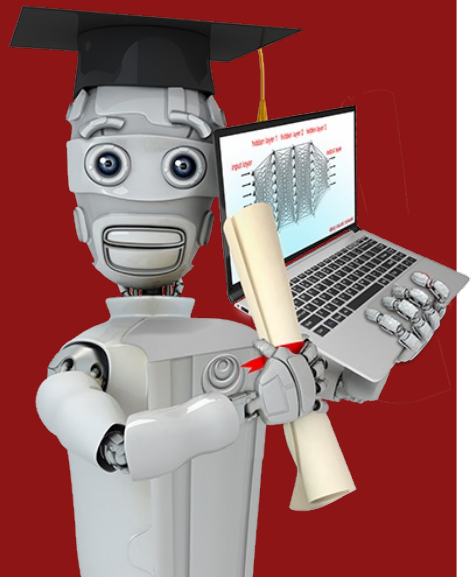
In a state s, input s to neural network.

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



DeepLearning.AI

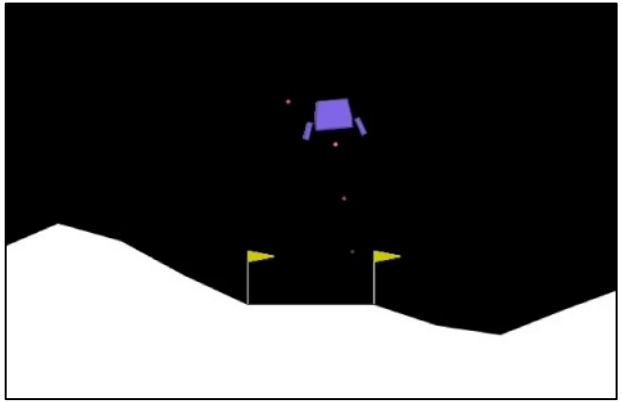
Stanford ONLINE



Continuous State Spaces

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

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

Option 1: → will never try other actions ↳ 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 ($\varepsilon = 0.05$)

greedy 95% of the time

exploring 5% of the time

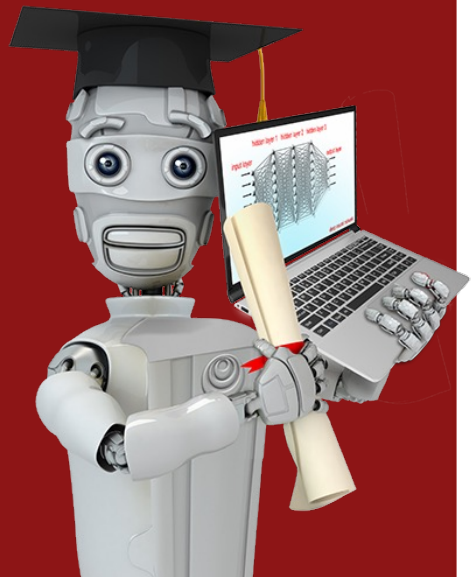
start ε high

gradually decrease

1.0 → 0.01

DeepLearning.AI

Stanford ONLINE



Continuous State Spaces

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

How to choose actions while still learning?

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

100,000,000

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

$m = 100,000,000$

$m' = 1,000$

repeat {

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

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

}

Mini-batch

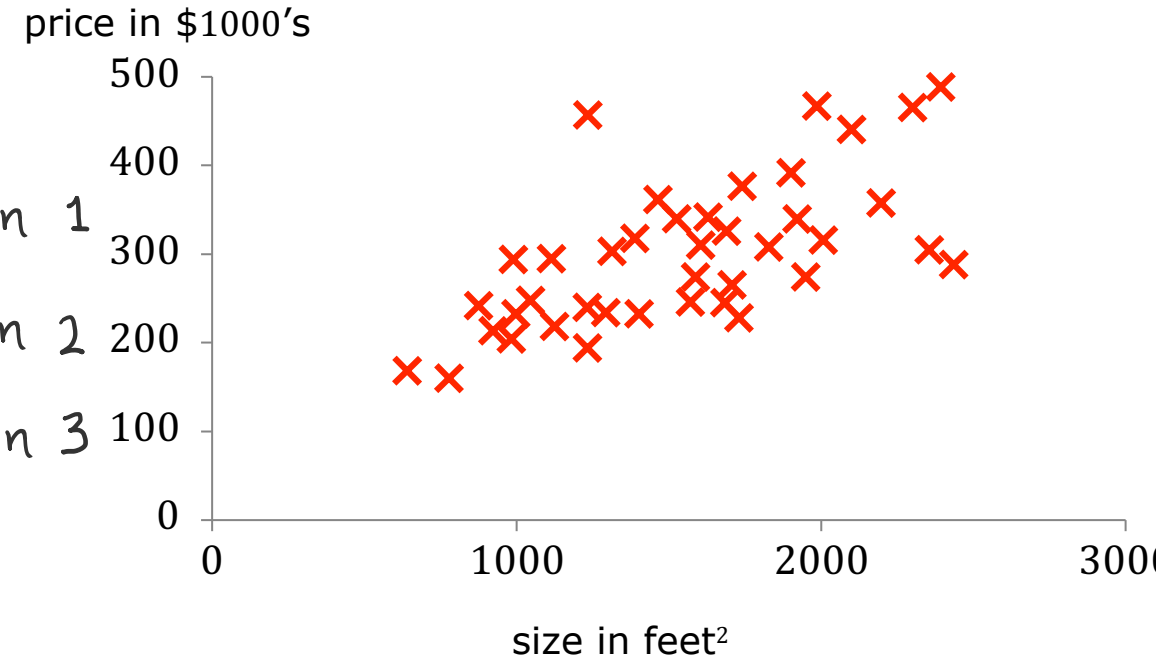
x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

batch

iteration 1

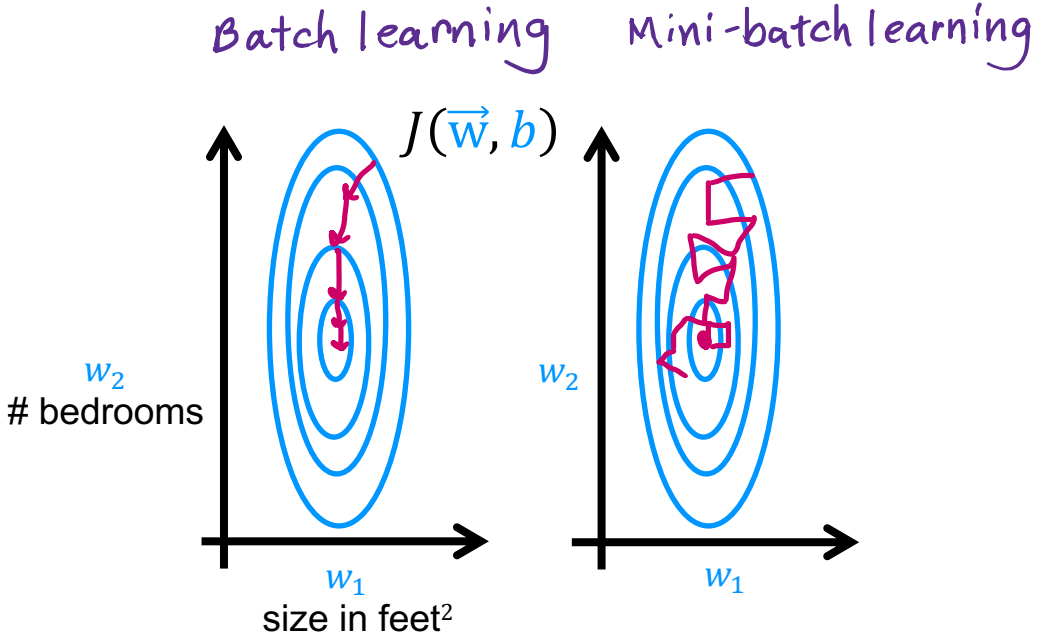
iteration 2

iteration 3

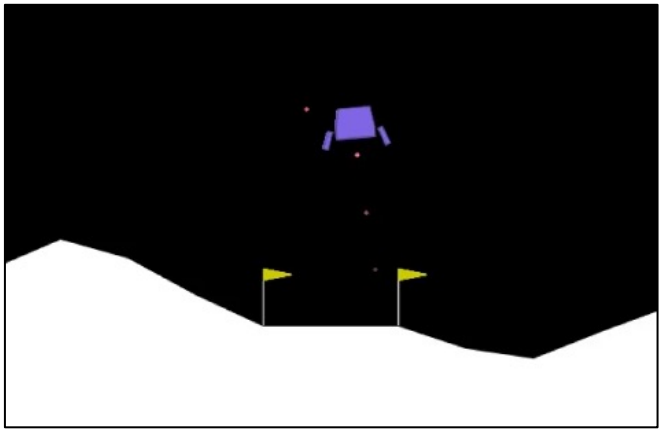


Mini-batch

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:

Create training set of ~~10,000~~ 1,000 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 $Q = Q_{new}$.

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

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

Soft Update

Set $Q = Q_{new}$.

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

$Q(s,a)$

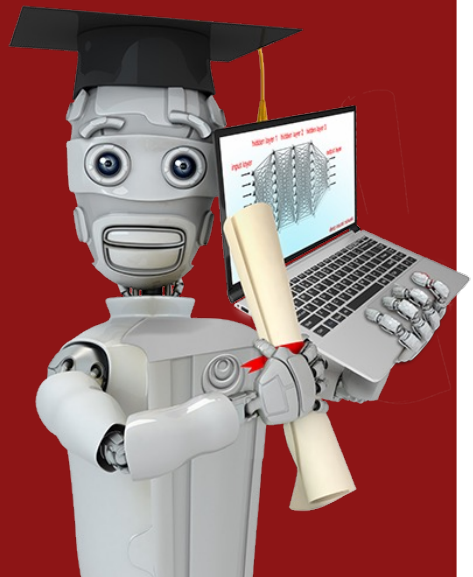
$W = 0.01 W_{new} + 0.99 W$

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

$w = 1 w_{new} + 0 w$

DeepLearning.AI

Stanford ONLINE



Continuous State Spaces

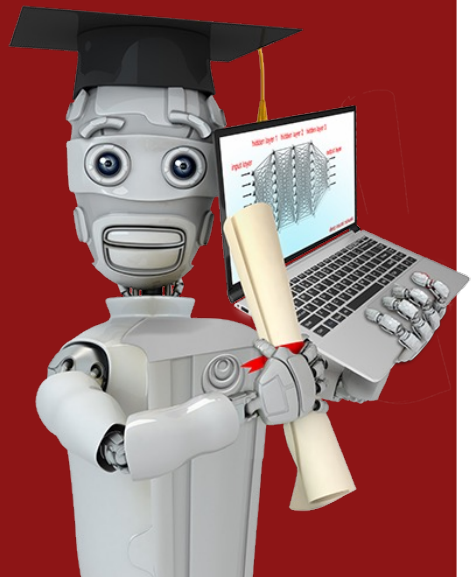
The state of reinforcement learning

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.

DeepLearning.AI

Stanford
ONLINE



Conclusion

Summary and
Thank you

Courses

- Supervised Machine Learning: Regression and Classification

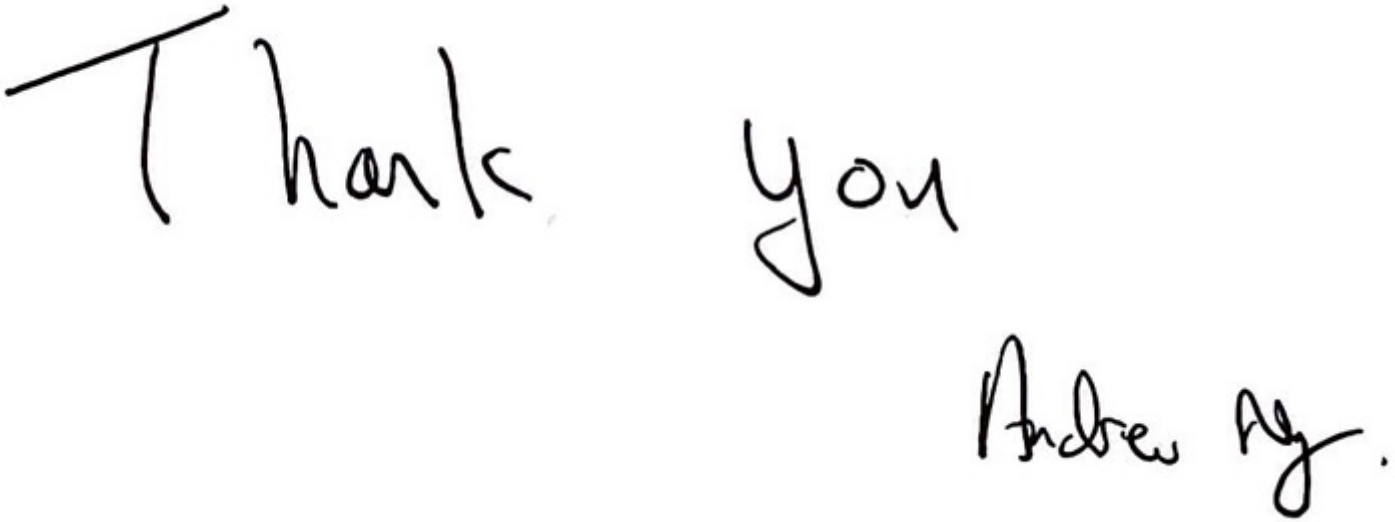
  Linear regression, logistic regression, gradient descent

- 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.