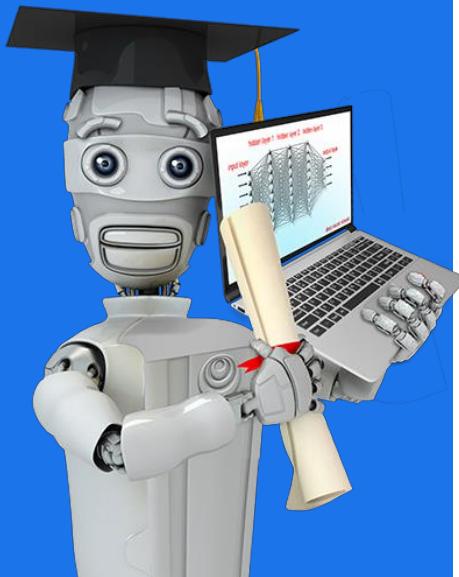


Stanford  
ONLINE

DeepLearning.AI



# Machine Learning

---

Welcome!

Re: Urgent Information :)

External

Spam ×

Congratulations!  
You've won  
a million dollars!



Compose

Mail

Inbox

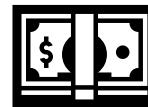
Starred

Snoozed

Sent

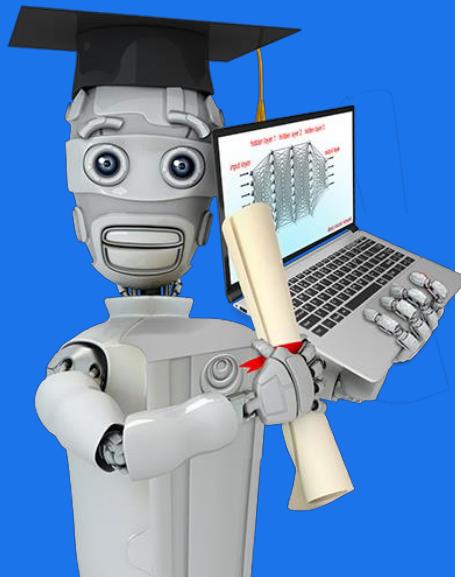
Drafts

42



Stanford  
ONLINE

DeepLearning.AI



# Machine Learning

---

Applications of  
Machine Learning

Stanford  
ONLINE

DeepLearning.AI



# Machine Learning Overview

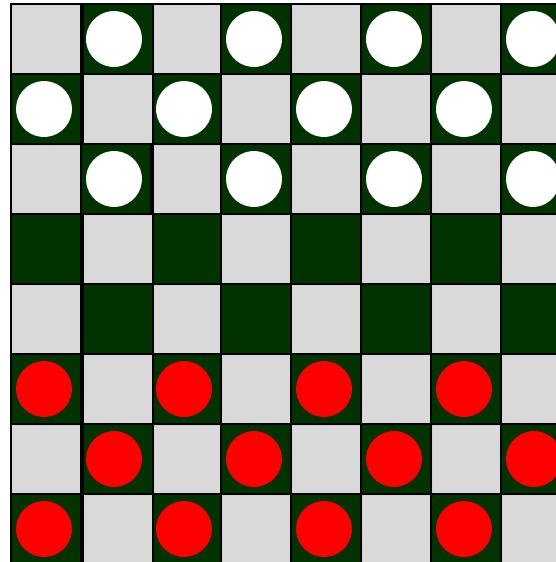
---

What is  
Machine Learning?

# Machine learning

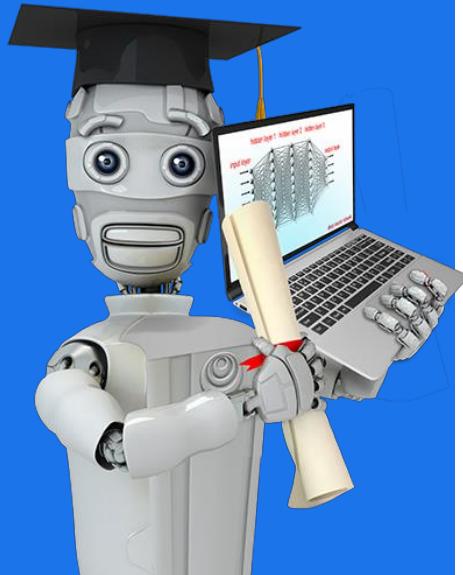
“Field of study that gives computers the ability to learn without being explicitly programmed.”

Arthur Samuel (1959)



Stanford  
ONLINE

DeepLearning.AI

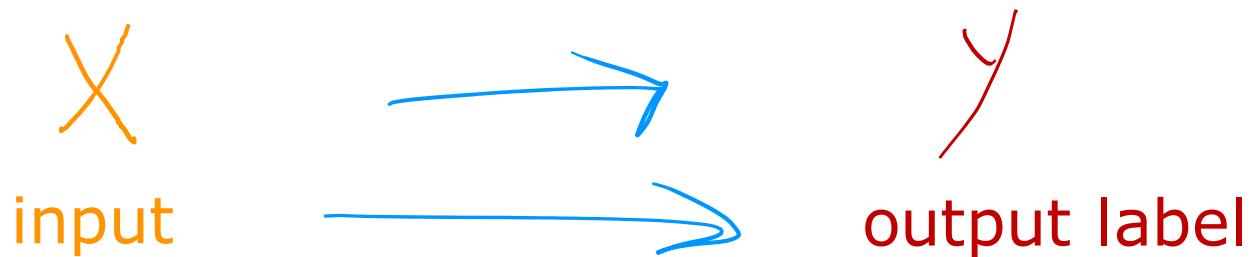


# Machine Learning Overview

---

## Supervised Learning Part 1

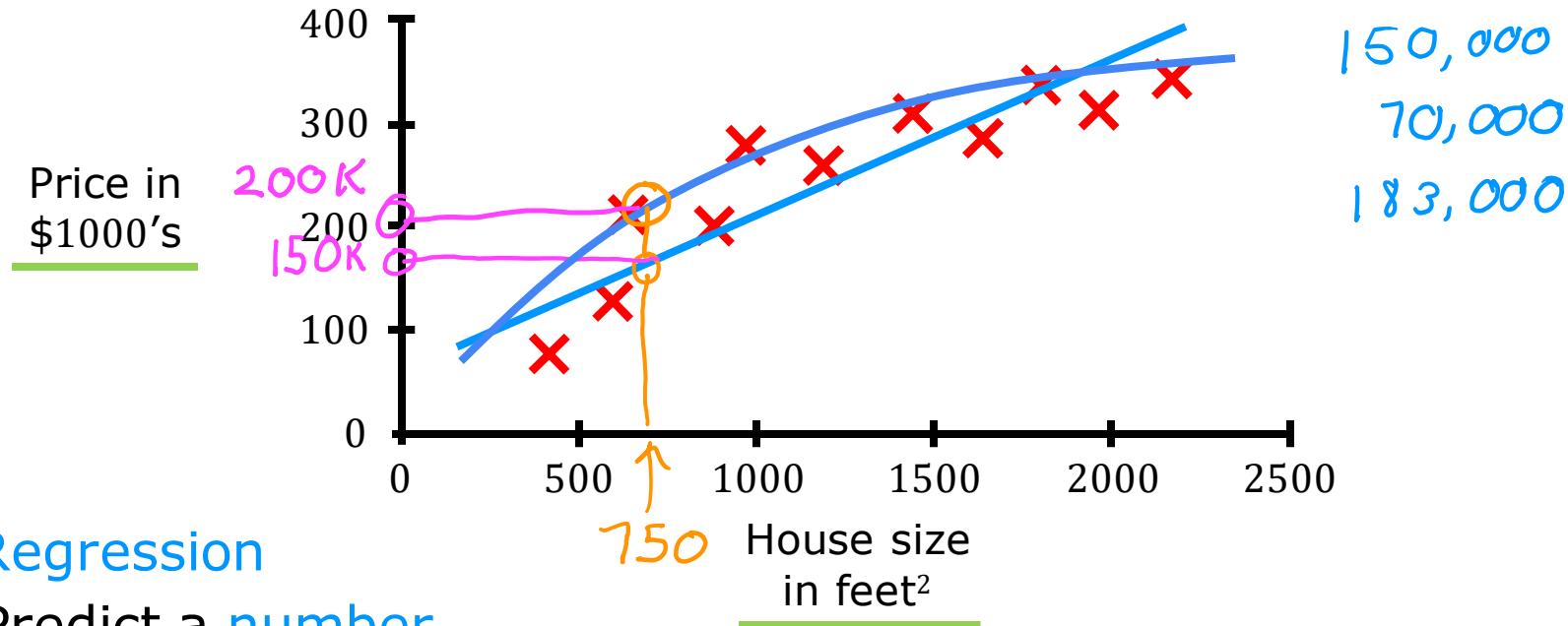
# Supervised learning



Learns from being given “right answers”

Input (X)	Output (Y)	Application
email	spam? (0/1)	spam filtering
audio	text transcripts	speech recognition
English	Spanish	machine translation
ad, user info	click? (0/1)	online advertising
image, radar info	position of other cars	self-driving car
image of phone	defect? (0/1)	visual inspection

# Regression: Housing price prediction



Stanford  
ONLINE

DeepLearning.AI



# Machine Learning Overview

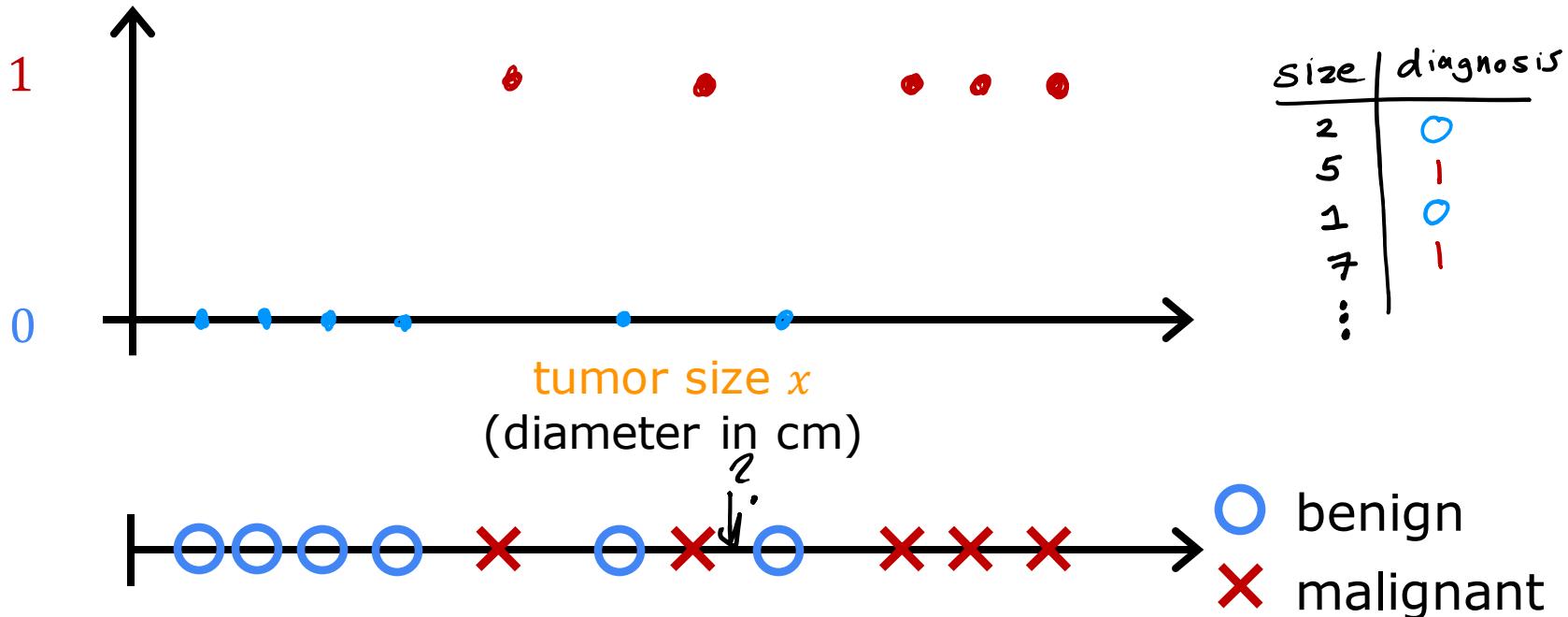
---

## Supervised Learning Part 2

# Classification: Breast cancer detection



malignant      benign

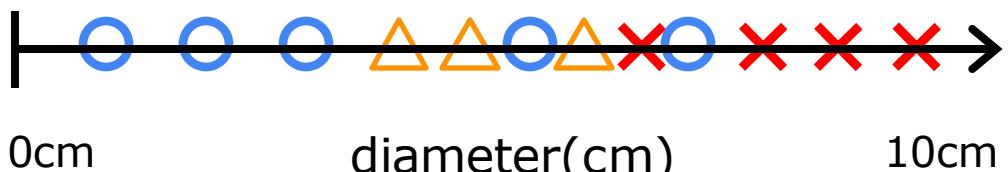


# Classification: Breast cancer detection

○ benign

✗ malignant type 1

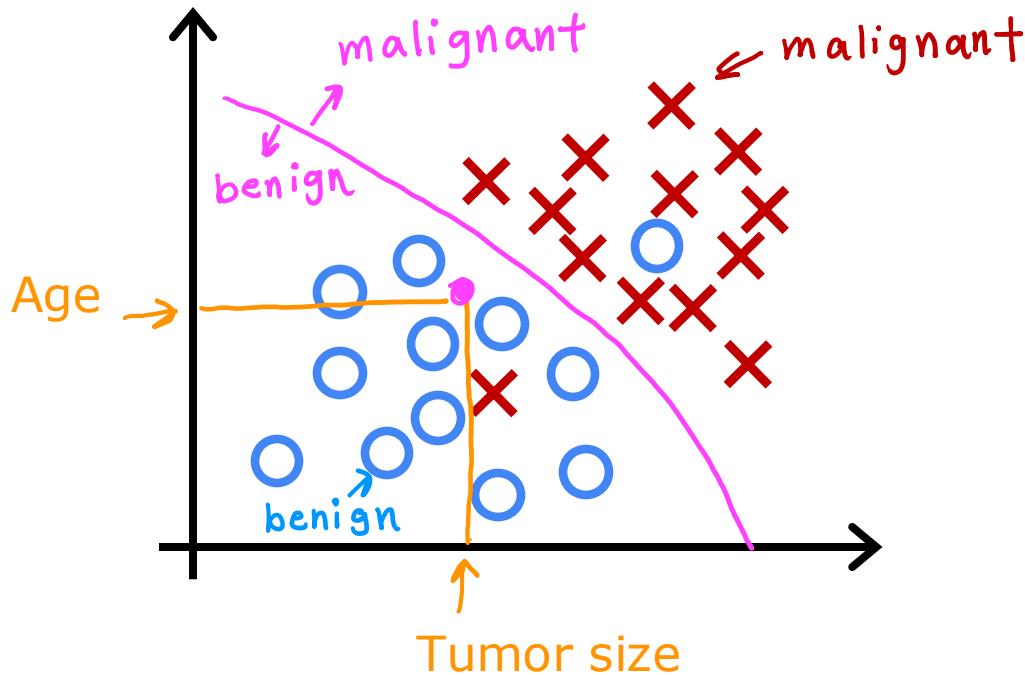
△ malignant type 2



Classification  
predict categories    cat   dog    benign malignant    0, 1, 2

small number of possible outputs

# Two or more inputs



# Supervised learning

Learns from being given “right answers”

## Regression

Predict a number

infinitely many possible outputs

## Classification

predict categories

small number of possible outputs



Stanford  
ONLINE

DeepLearning.AI

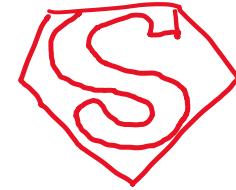


# Machine Learning Overview

---

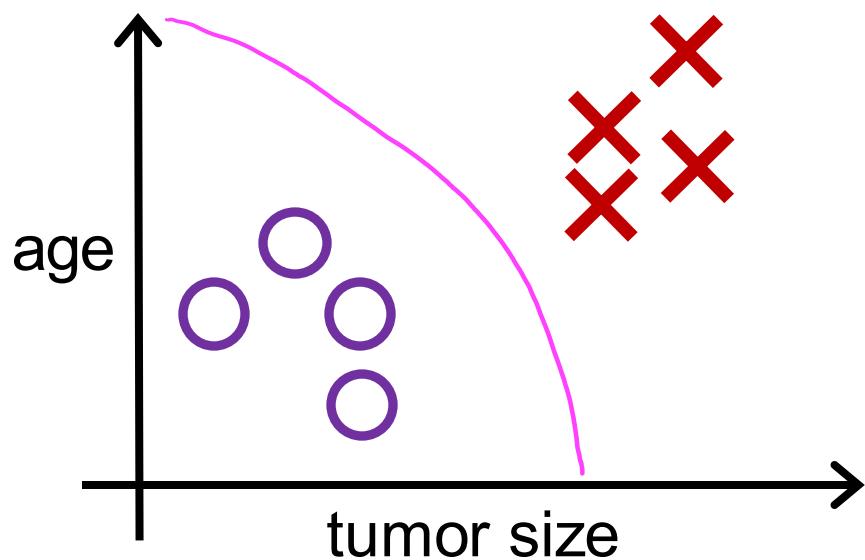
## Unsupervised Learning Part 1

Previous: Supervised learning

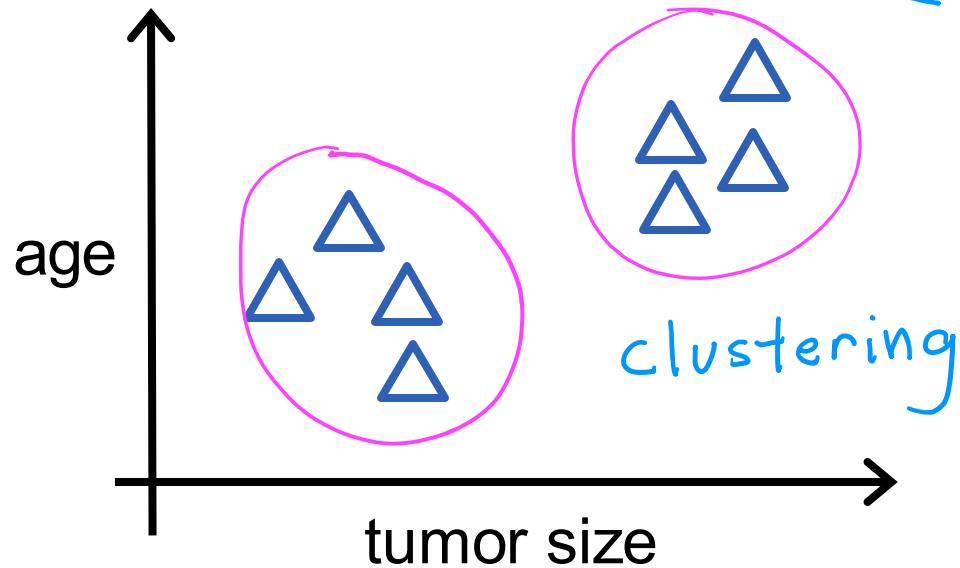


Now: Unsupervised learning

Supervised learning  
Learn from data **labeled**  
with the “**right answers**”



Unsupervised learning  
Find something interesting  
in **unlabeled** data.



# Clustering: Google news



Giant **panda** gives birth to rare **twin** cubs at Japan's oldest **zoo**

USA TODAY · 6 hours ago



- Giant **panda** gives birth to **twin** cubs at Japan's oldest **zoo**

CBS News · 7 hours ago

- Giant **panda** gives birth to **twin** cubs at Tokyo's Ueno **Zoo**

WHBL News · 16 hours ago

- A Joyful Surprise at Japan's Oldest **Zoo**: The Birth of **Twin Pandas**

The New York Times · 1 hour ago

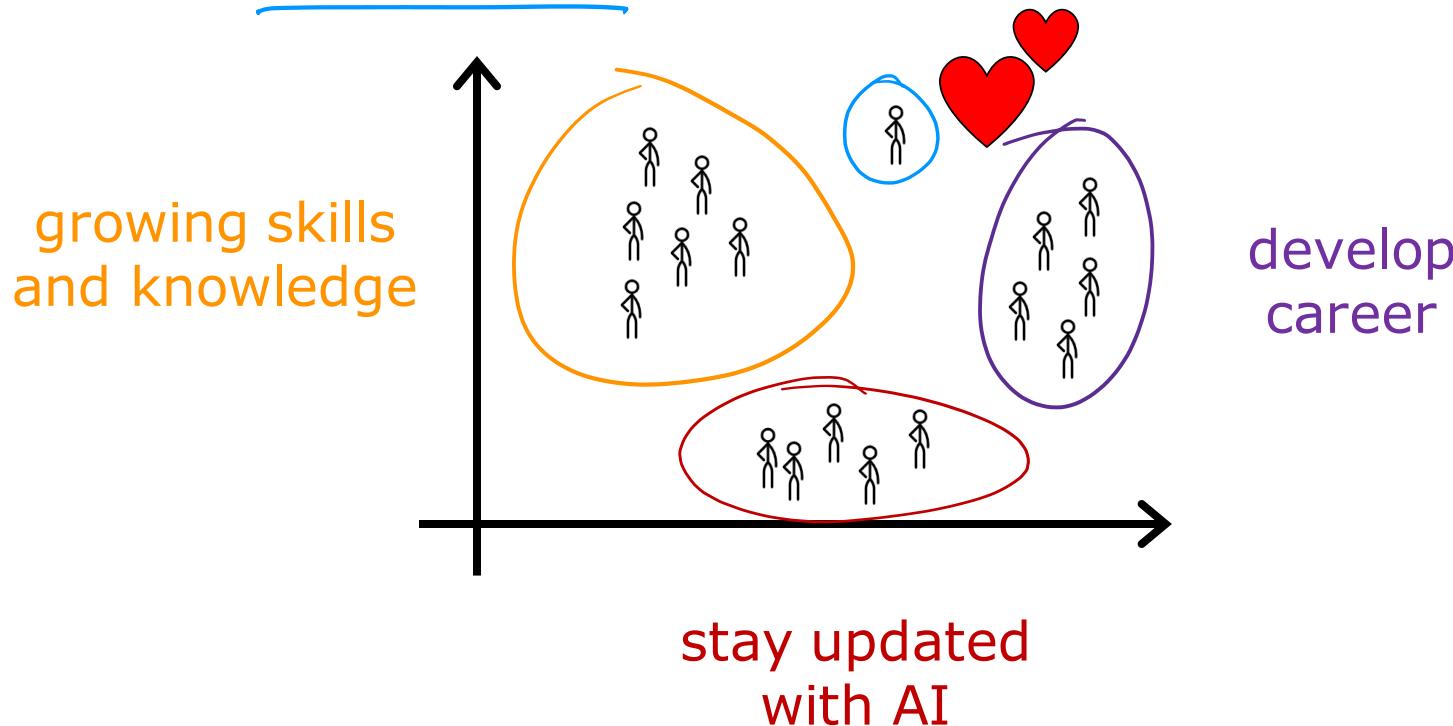
- **Twin** Panda **Cubs** Born at Tokyo's Ueno **Zoo**

PEOPLE · 6 hours ago

View Full Coverage

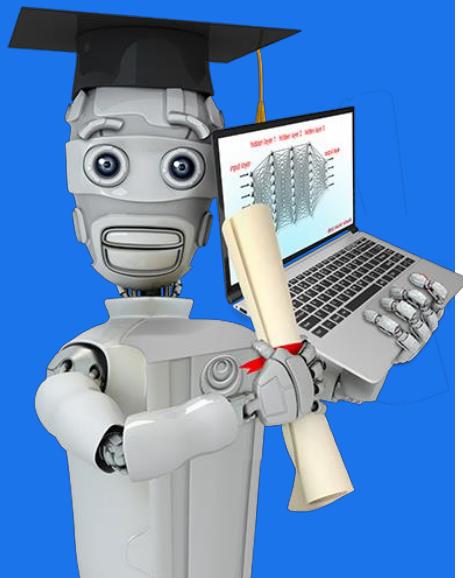


# Clustering: Grouping customers



Stanford  
ONLINE

DeepLearning.AI



# Machine Learning Overview

---

## Unsupervised Learning Part 2

# Unsupervised learning

Data only comes with inputs  $x$ , but not output labels  $y$ .  
Algorithm has to find **structure** in the data.

## Clustering

Group similar data points together.

## Dimensionality reduction

Compress data using fewer numbers.

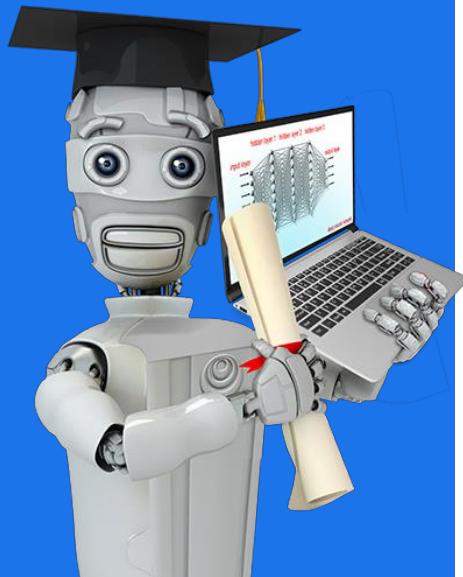
## Anomaly detection

Find unusual data points.



Stanford  
ONLINE

DeepLearning.AI

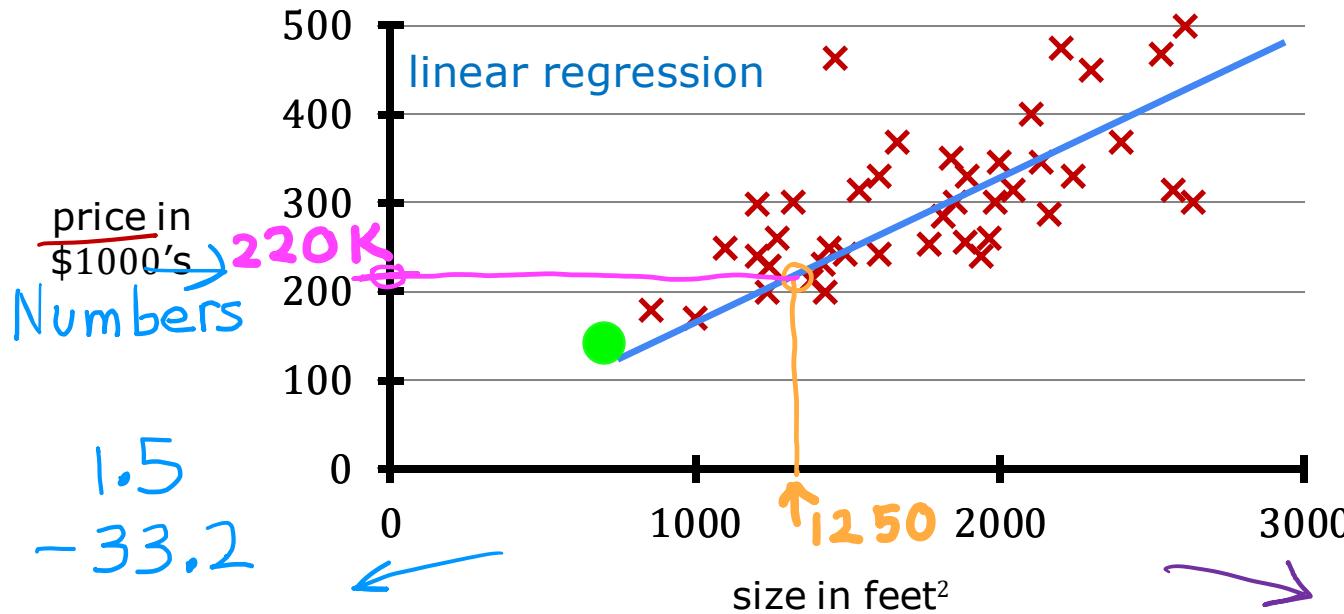


# Linear Regression with One Variable

---

## Linear Regression Model Part 1

# House sizes and prices



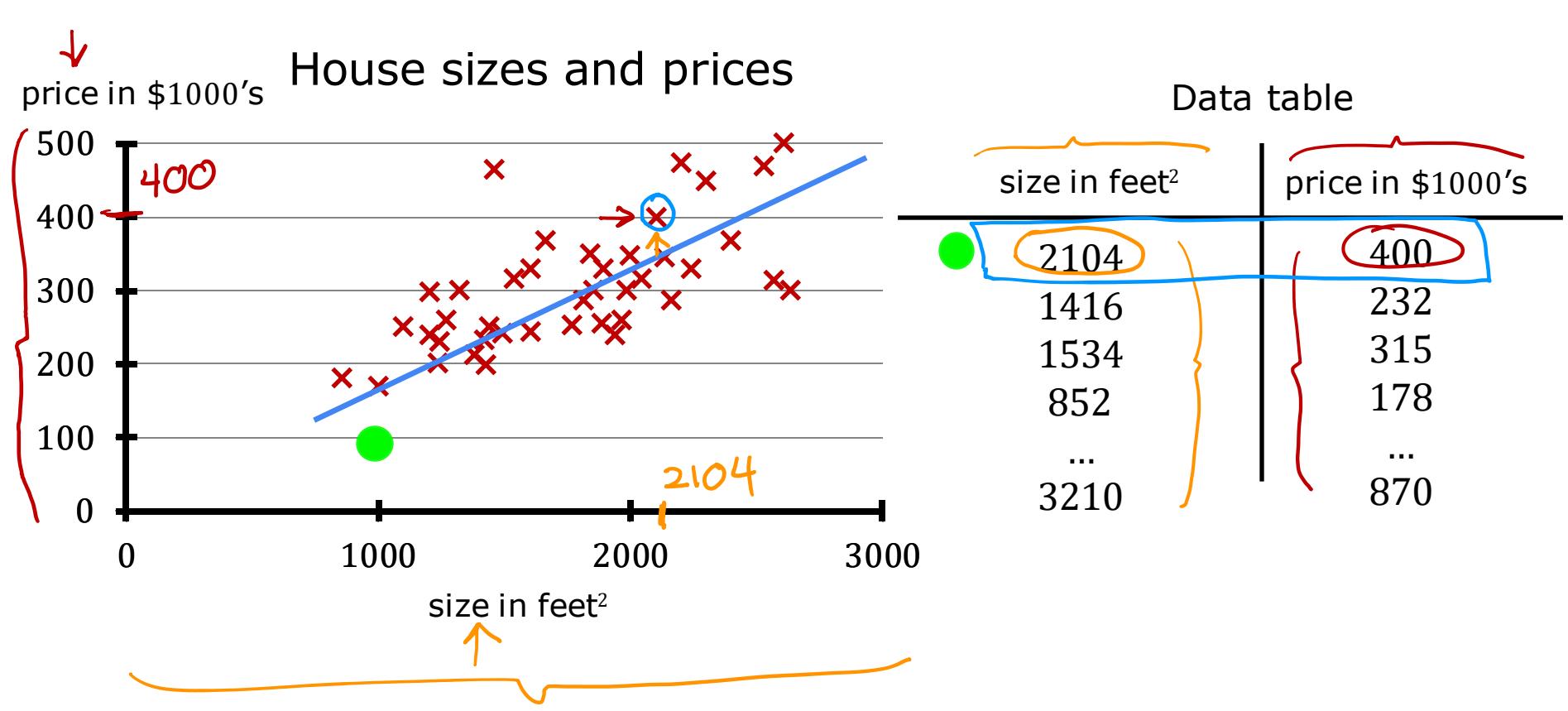
Regression model  
Predicts numbers  
Infinitely many possible outputs

Supervised learning model  
Data has "right answers"

Classification model  
Predicts categories  
Small number of possible outputs

categories  
cat } 2  
dog }

disease  10



# Terminology

Training set: Data used to train the model

$x$

size in feet<sup>2</sup>

(1)

2104

(2)

1416

(3)

1534

(4)

852

...

(47)

3210

$$x^{(1)} = 2104$$

$$(x^{(1)}, y^{(1)}) = (2104, 400)$$

$$x^{(2)} = 1416$$

$$X^{(2)} \neq X^2 \text{ not exponent}$$

	$x$	$y$
1	2104	400
2	1416	232
3	1534	315
4	852	178
...	...	...
47	3210	870

$$m = 47$$

Notation:

$x$  = "input" variable  
feature

$y$  = "output" variable  
"target" variable

$m$  = number of training examples

$(x, y)$  = single training example

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

$(x^{(i)}, y^{(i)})$  =  $i^{\text{th}}$  training example  
index  $(1^{\text{st}}, 2^{\text{nd}}, 3^{\text{rd}} \dots)$

Stanford  
ONLINE

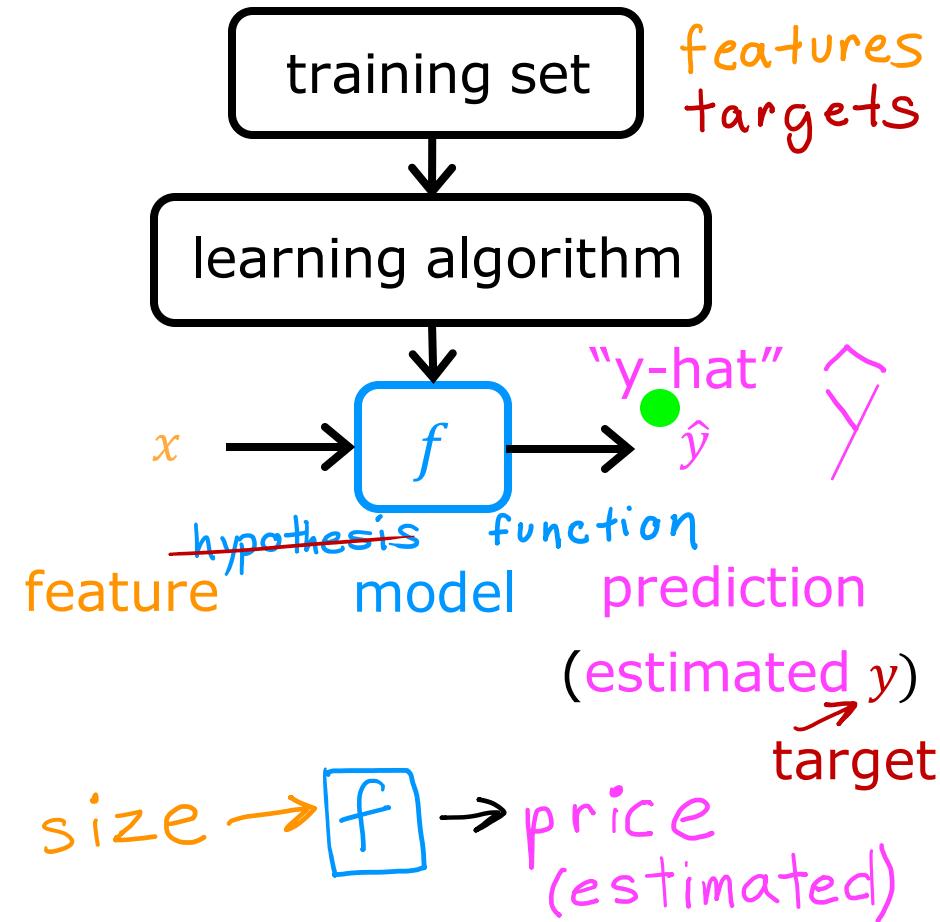
DeepLearning.AI



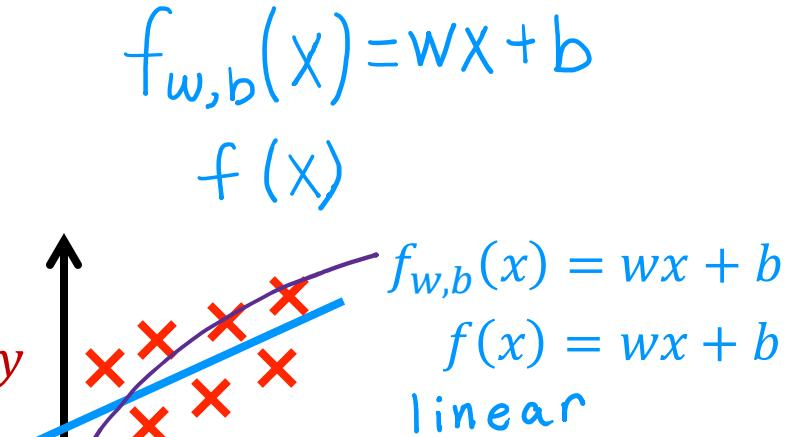
# Linear Regression with One Variable

---

## Linear Regression Model Part 2



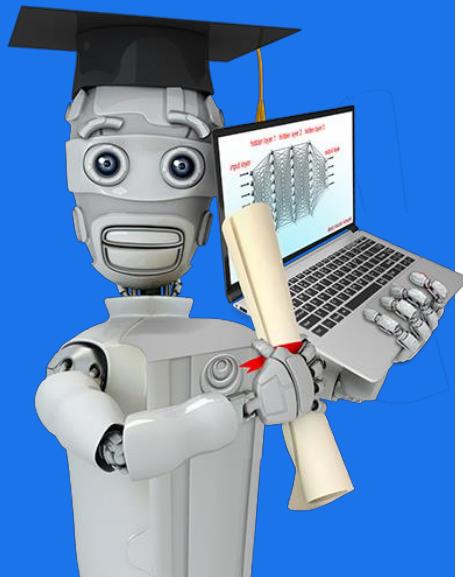
How to represent  $f$ ?



single feature  $x$   
Linear regression with one variable.  
size  
Univariate linear regression.  
one variable

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with One Variable

---

## Cost Function

# Training set

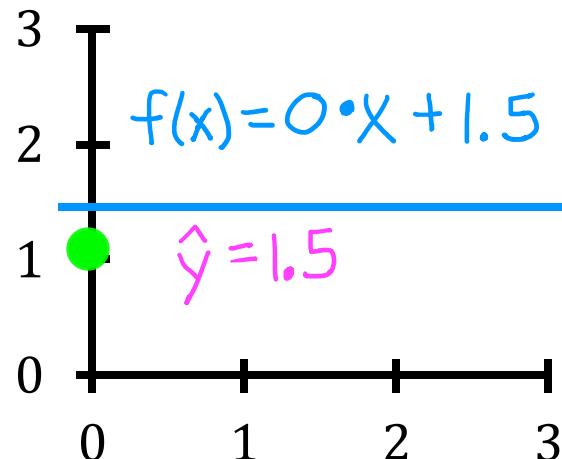
features	targets
size in feet <sup>2</sup> ( $x$ )	price \$1000's ( $y$ )
2104	460
1416	232
1534	315
852	178
...	...

$$\text{Model: } f_{w,b}(x) = wx + b$$

$w, b$ : parameters  
coefficients  
weights

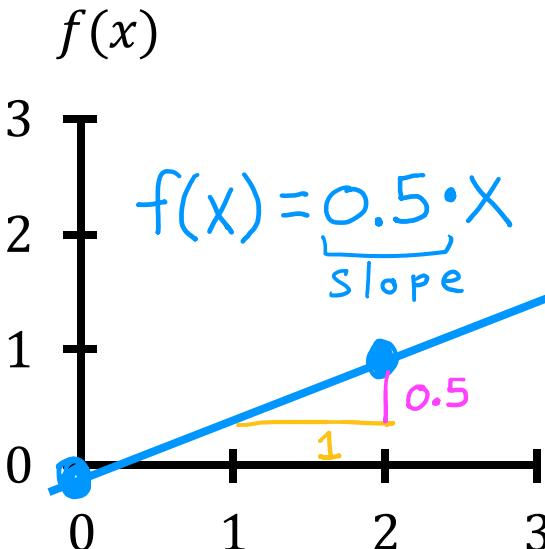
What do  $w, b$  do?

$$f_{w,b}(x) = wx + b$$

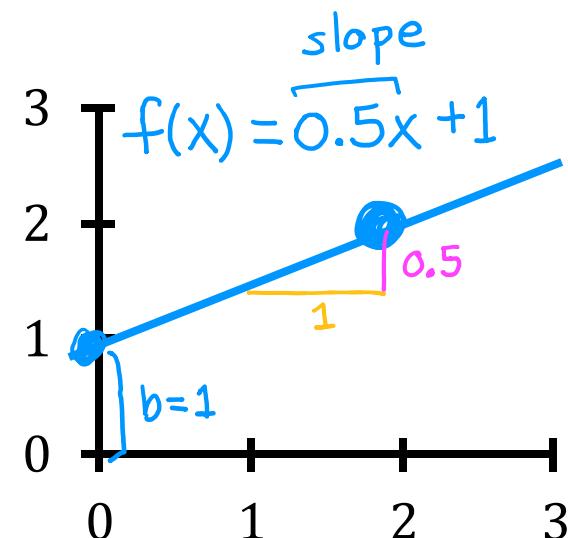


$$\rightarrow w = 0$$
$$\rightarrow b = 1.5$$

*(y-intercept)*

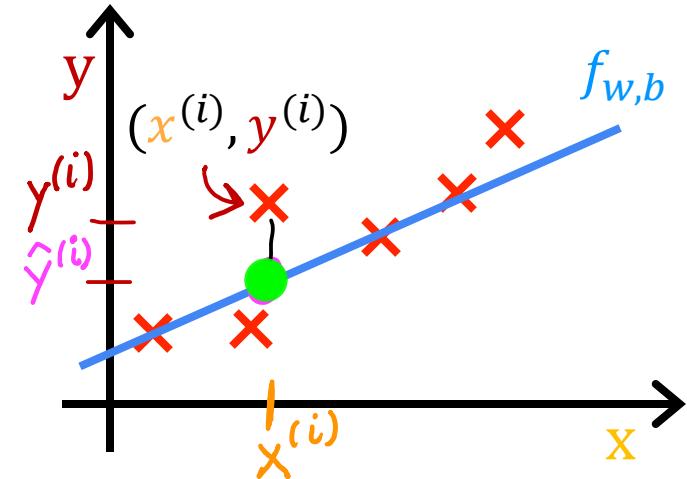


$$\rightarrow w = 0.5$$
$$\rightarrow b = 0$$



$$\rightarrow w = 0.5$$
$$\rightarrow b = 1$$

## Cost function: Squared error cost function



$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

$$f_{w,b}(x^{(i)}) = w x^{(i)} + b$$

$$\bar{J}(w, b) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

$m$  = number of training examples

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

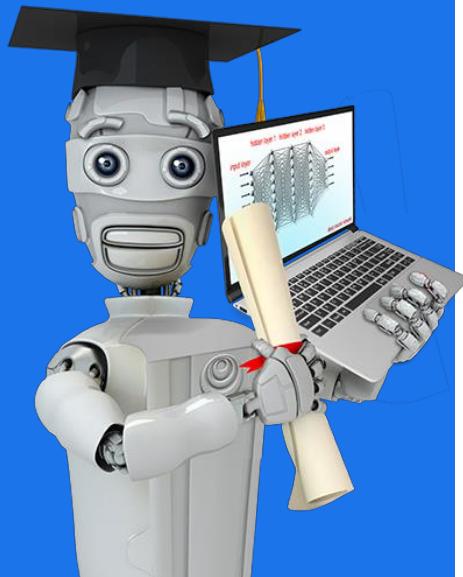
intuition (next!)

Find  $w, b$ :

$\hat{y}^{(i)}$  is close to  $y^{(i)}$  for all  $(x^{(i)}, y^{(i)})$ .

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with One Variable

---

Cost Function  
Intuition

model:

$$\underline{f_{w,b}(x) = wx + b}$$

parameters:

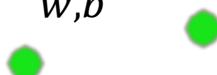
$$\underline{w, b}$$

cost function:

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

goal:

$$\underset{w,b}{\text{minimize}} J(w, b)$$



simplified

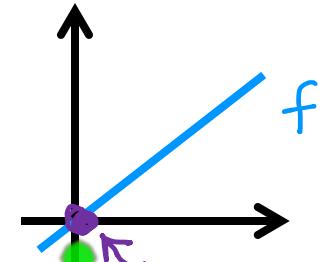
$$f_w(x) = \underline{wx}$$

$$b = \emptyset$$

$$w$$

$$\underline{J(w)} = \frac{1}{2m} \sum_{i=1}^m (\underline{f_w(x^{(i)})} - y^{(i)})^2$$

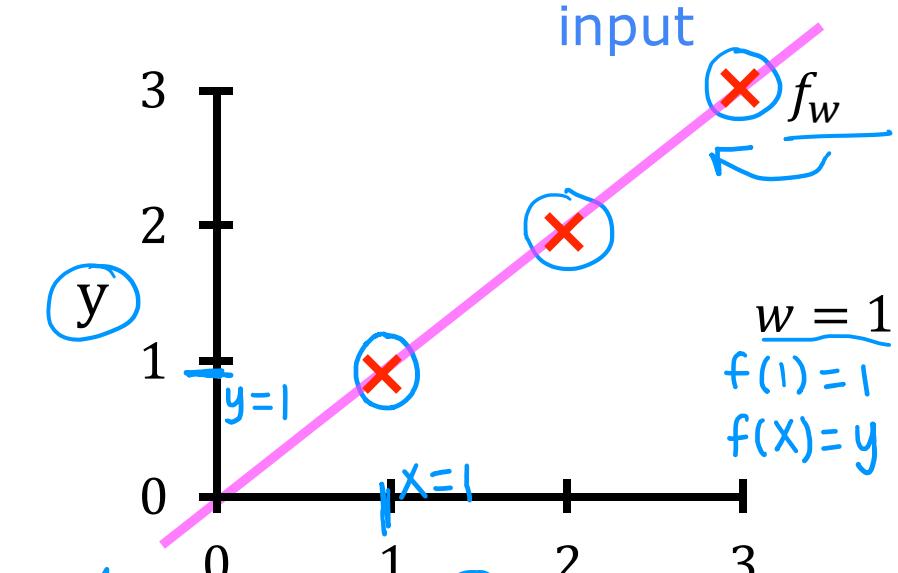
$$\underset{w}{\text{minimize}} \underline{J(w)}$$



$$\omega x^{(i)}$$

$\rightarrow f_w(x)$

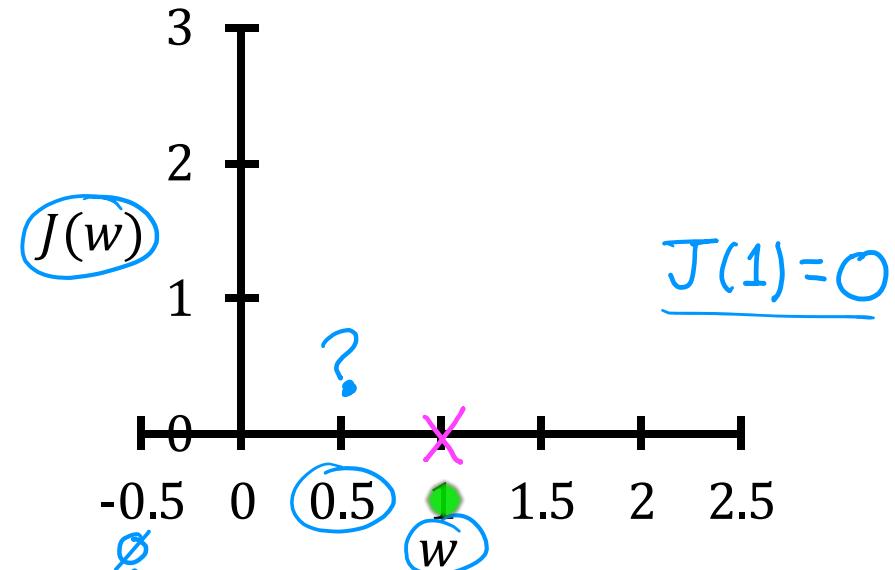
(for fixed  $w$ , function of  $x$ )



$$J(w) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (wx^{(i)} - y^{(i)})^2$$

$J(w)$

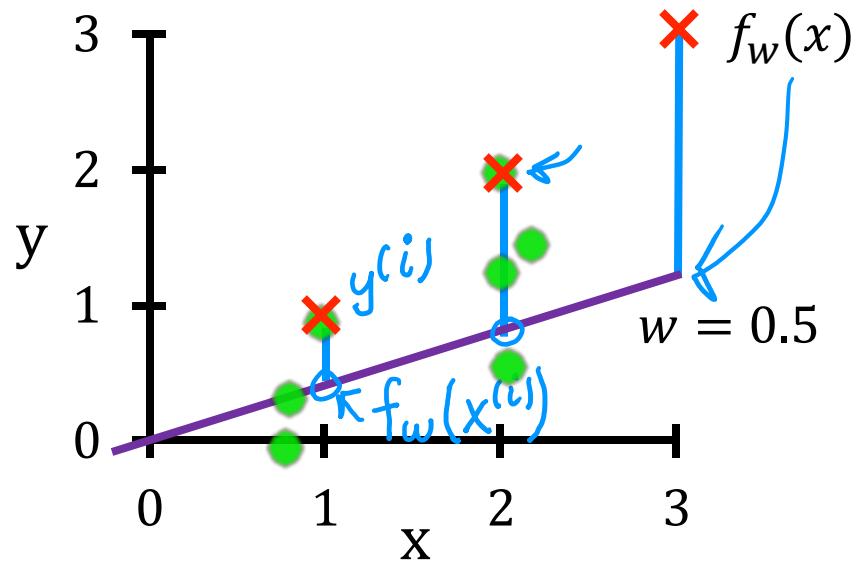
(function of  $w$ )  
parameter



$$= \frac{1}{2m} (0^2 + 0^2 + 0^2) = 0$$

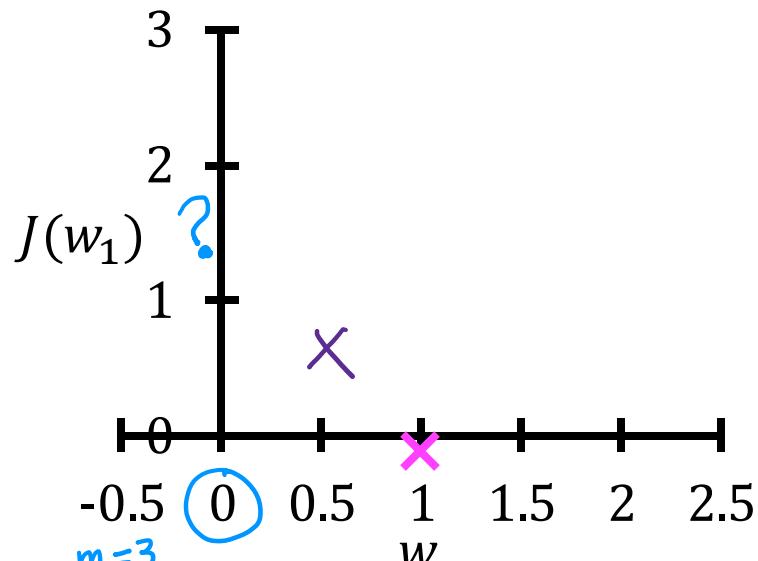
$f_w(x)$

(function of  $x$ )

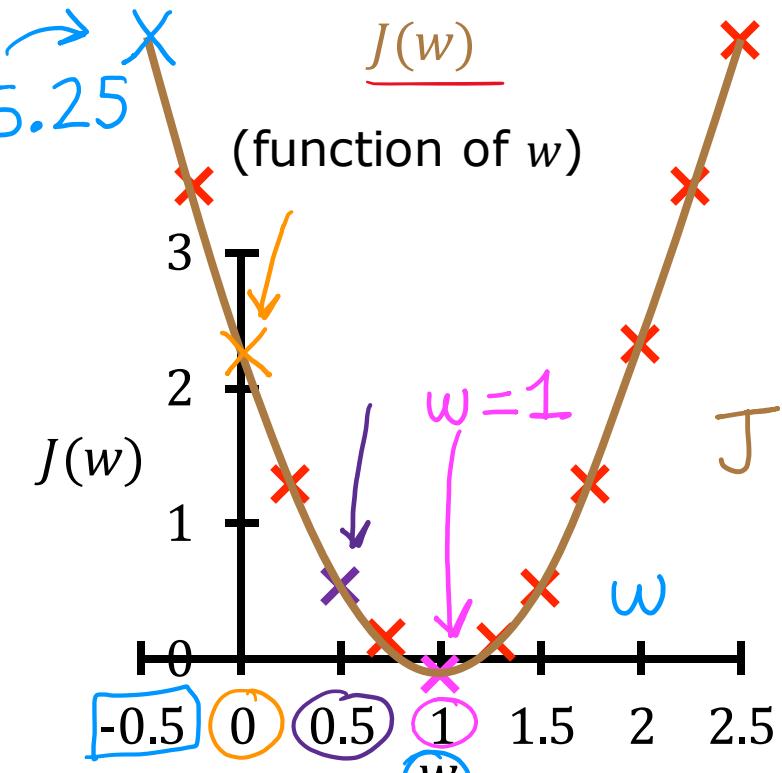
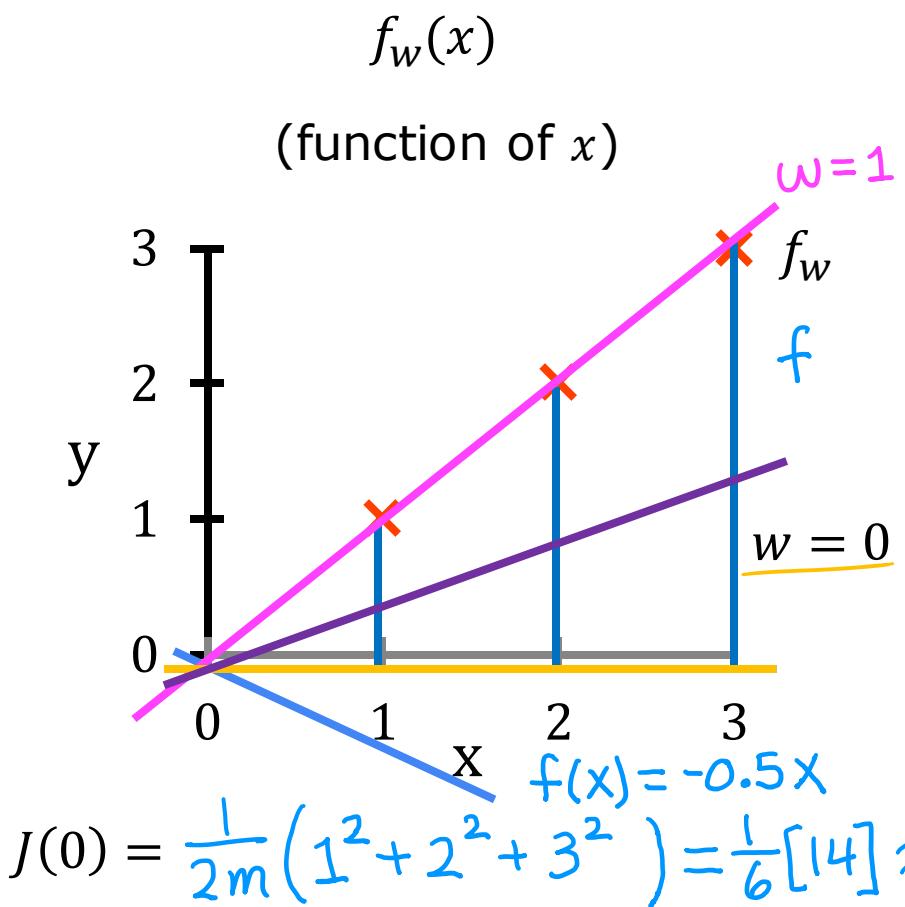


$J(w)$

(function of  $w$ )



$$J(0.5) = \frac{1}{2m} \left[ (0.5-1)^2 + (1-2)^2 + (1.5-3)^2 \right] = \frac{1}{2 \times 3} [3.5] = \frac{3.5}{6} \approx 0.58$$

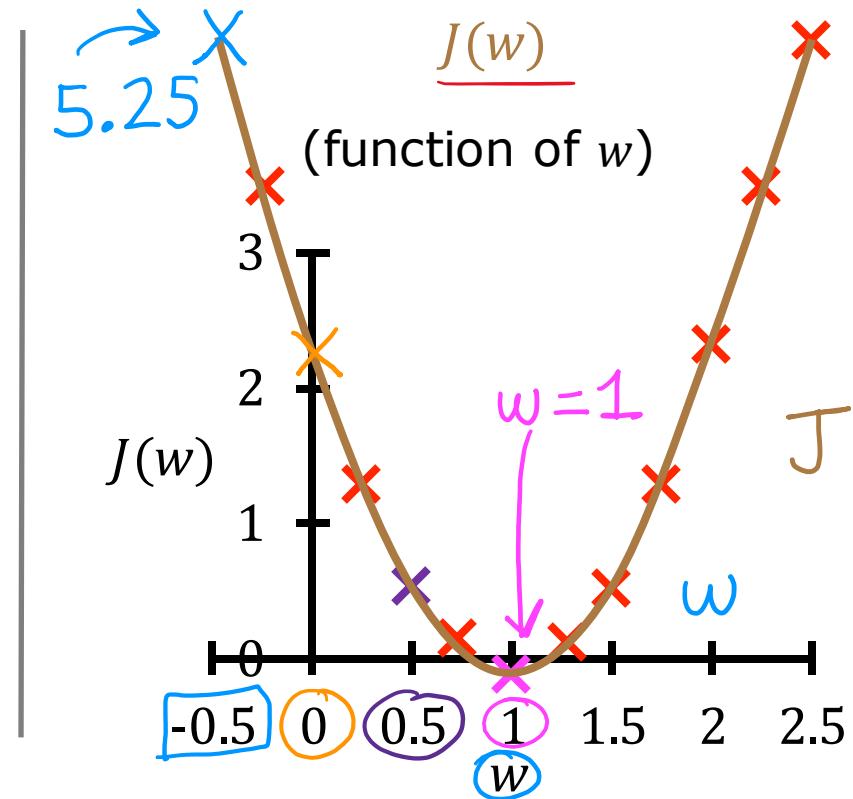


goal of linear regression:

$$\underset{w}{\text{minimize}} J(w)$$

general case:

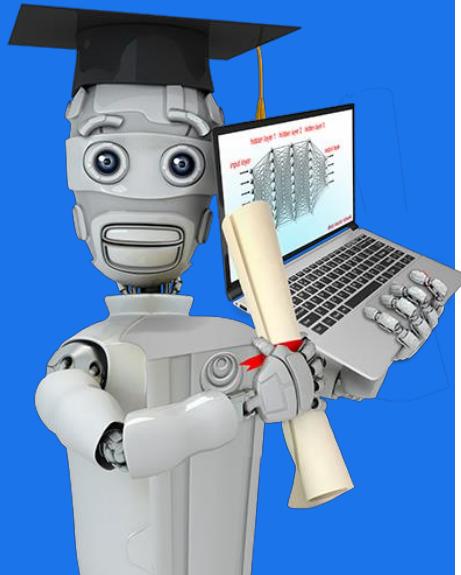
$$\underset{w,b}{\text{minimize}} J(w, b)$$



choose  $w$  to minimize  $J(w)$

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with One Variable

---

Visualizing  
the Cost Function

Model

$$f_{w,b}(x) = wx + b$$

Parameters

$w, b$

~~before:  $b=0$~~

Cost Function

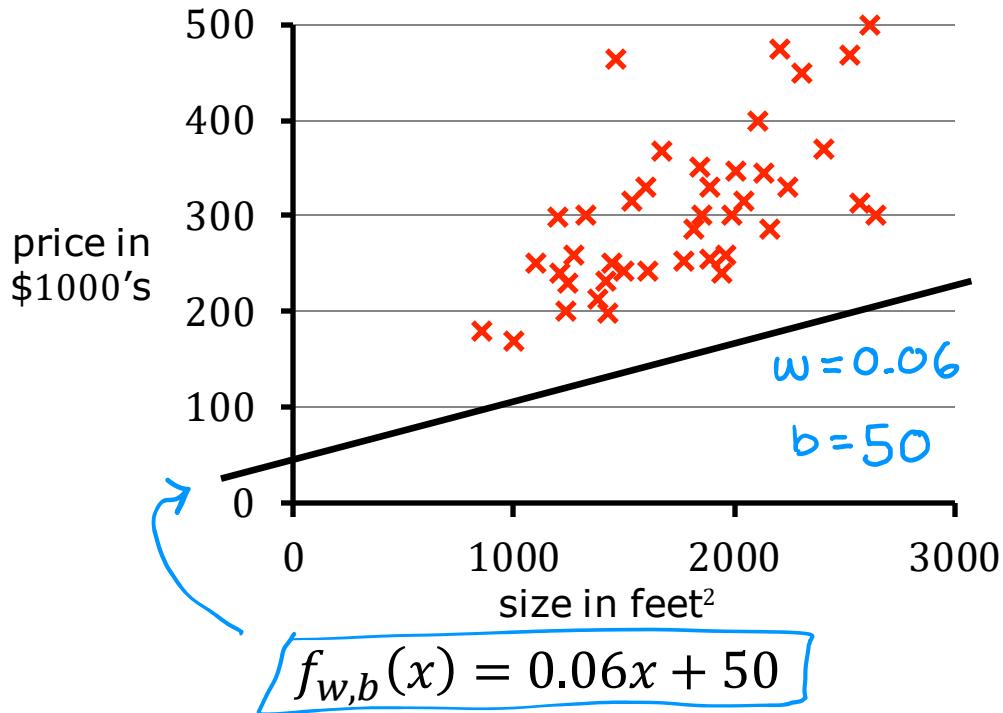
$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Objective

$$\underset{w,b}{\text{minimize}} J(w, b)$$

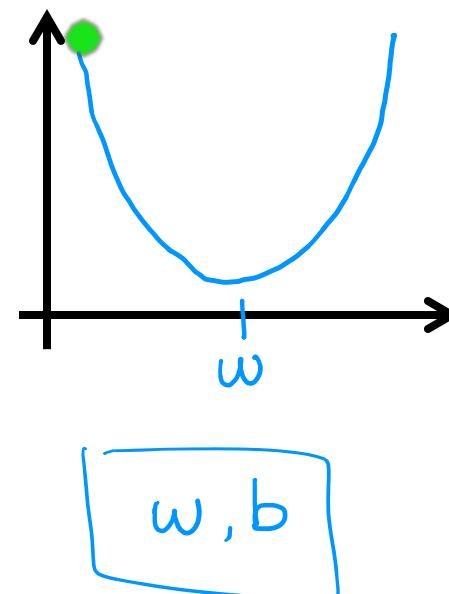
$$\underline{f_{w,b}}$$

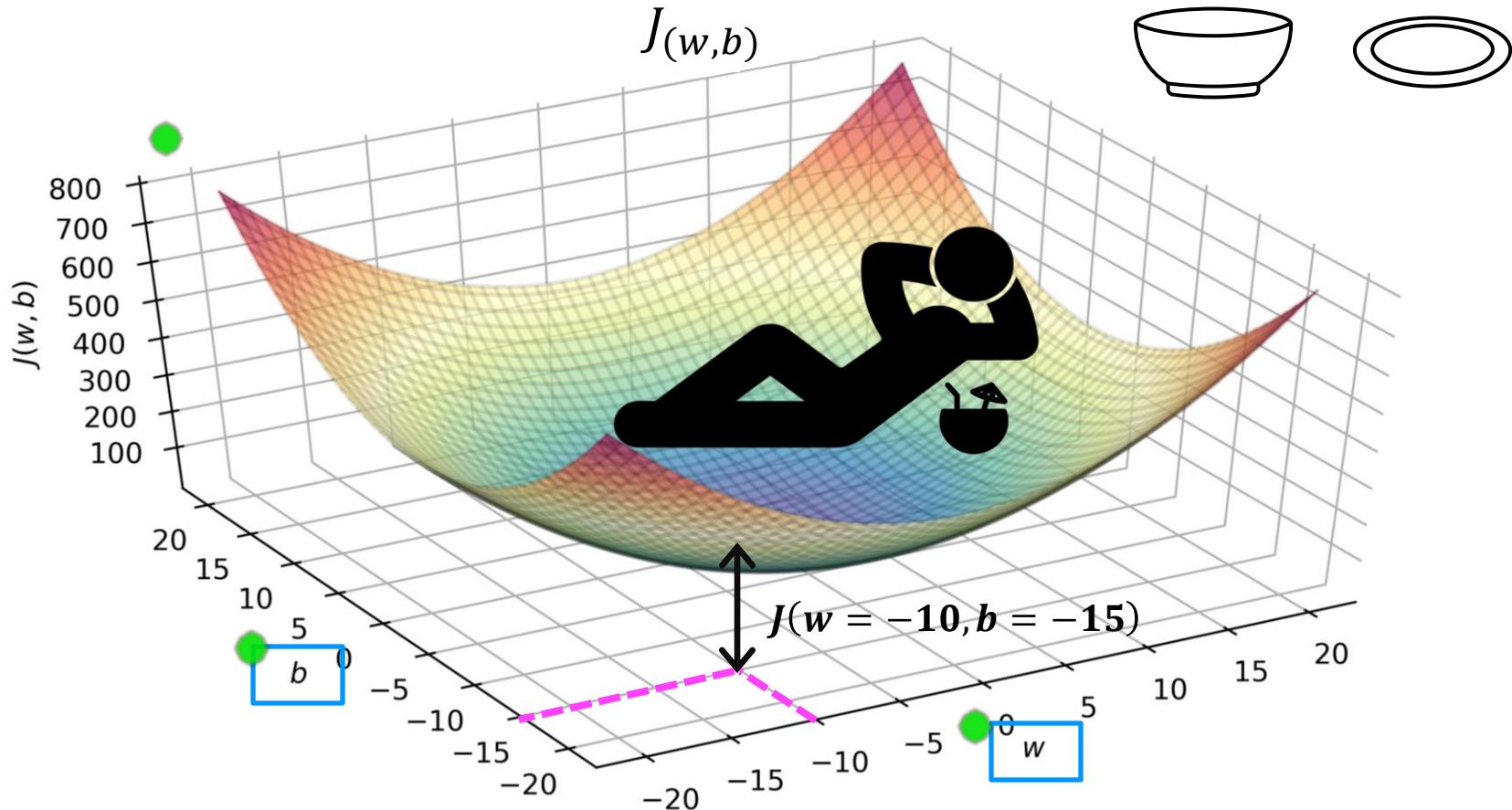
(function of  $x$ )



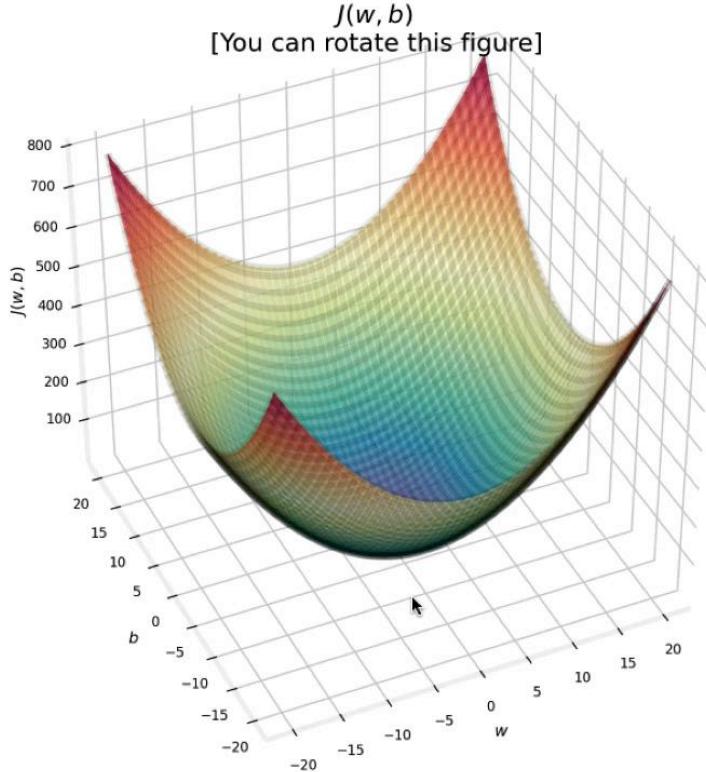
$$\underline{J}$$

(function of  $w, b$ )





# 3D surface plot



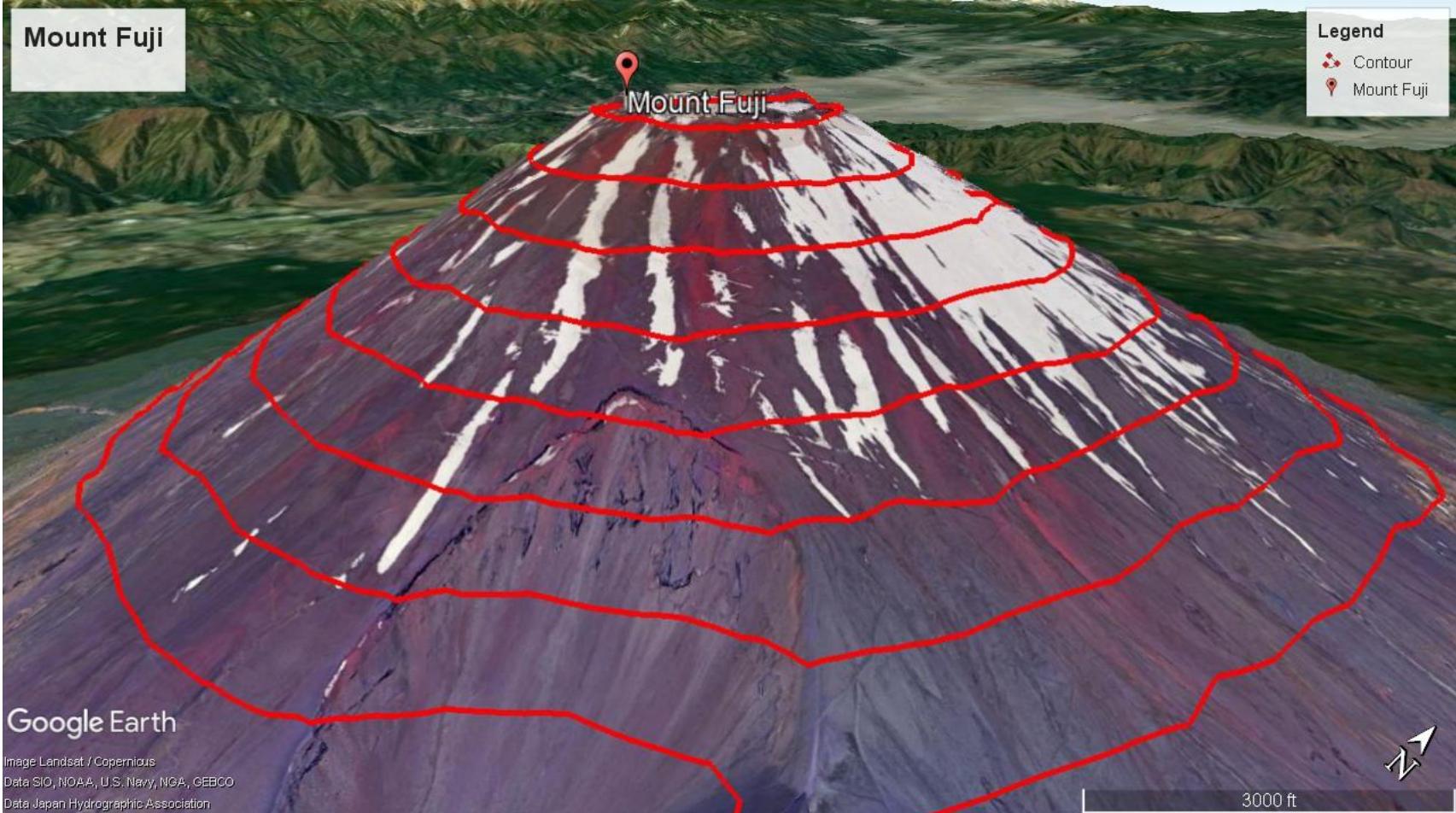
Alternative  
contour plot

# Mount Fuji

Mount Fuji

## Legend

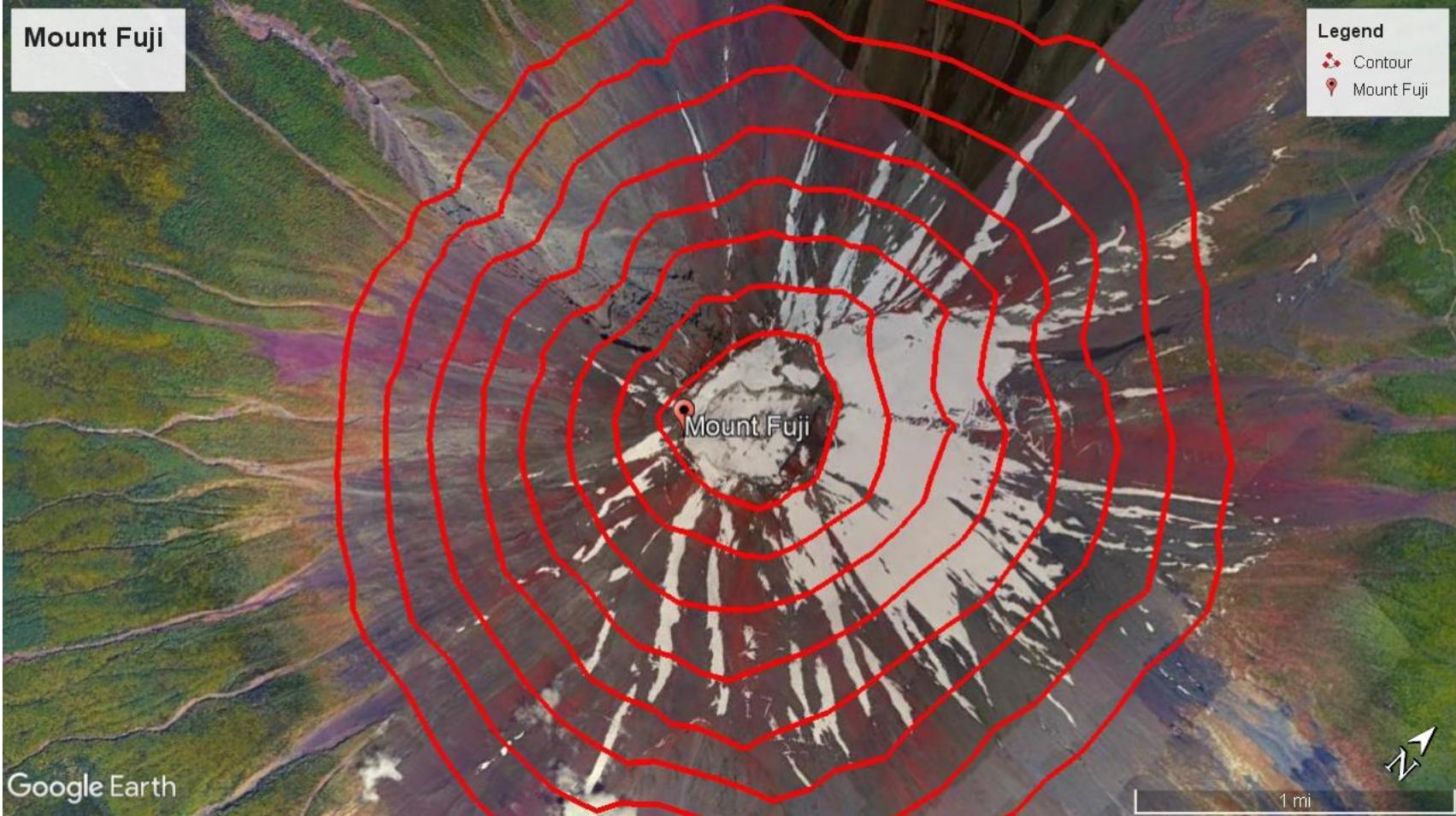
- Contour
- Mount Fuji

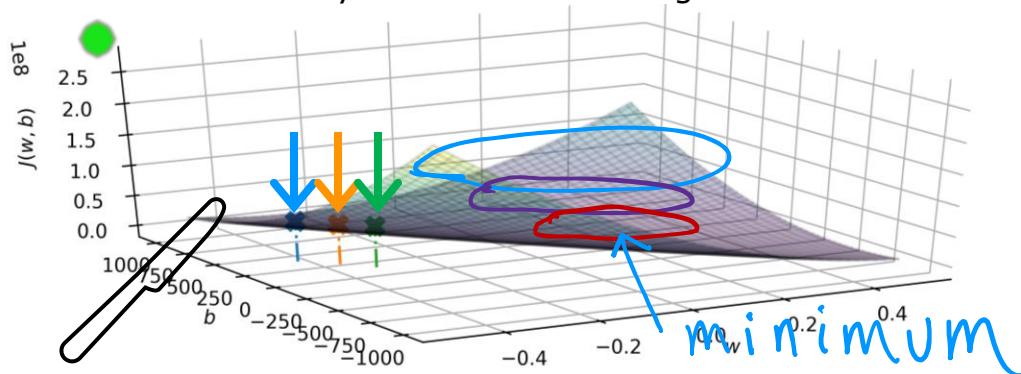
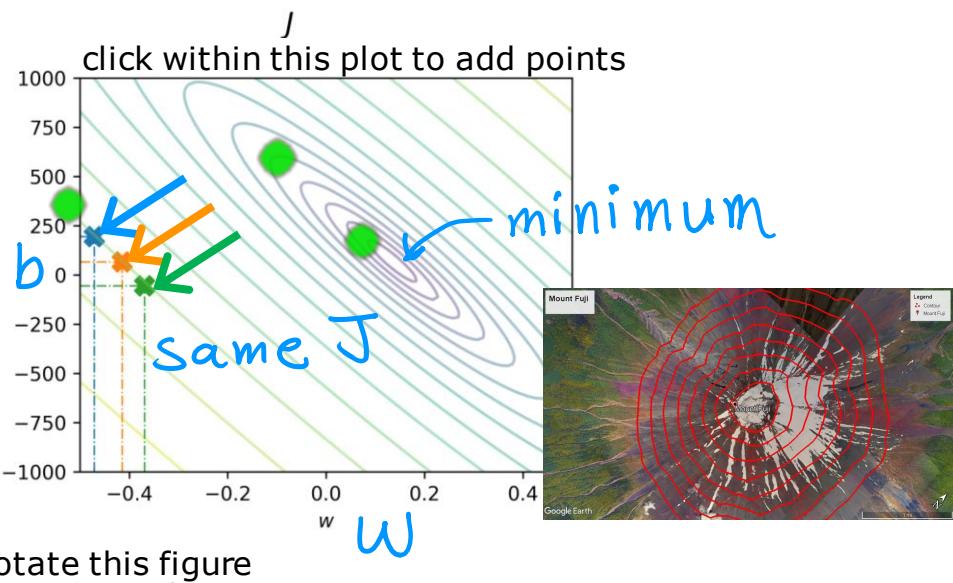
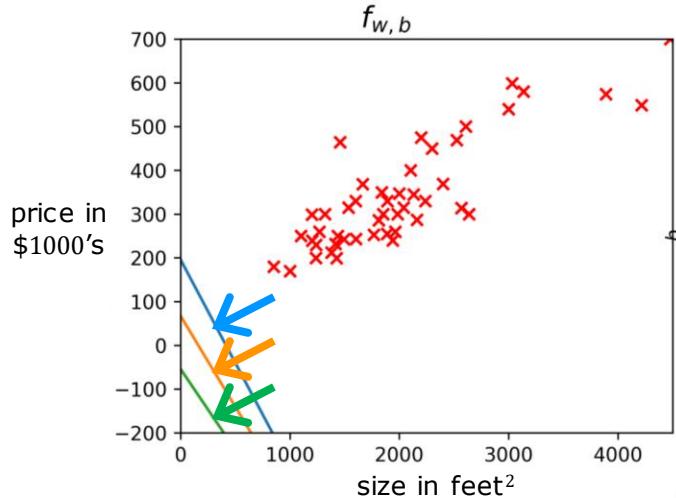


# Mount Fuji

Legend

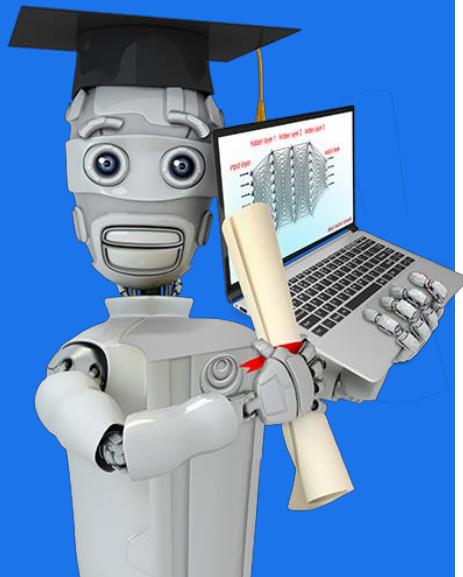
- Contour
- Mount Fuji





Stanford  
ONLINE

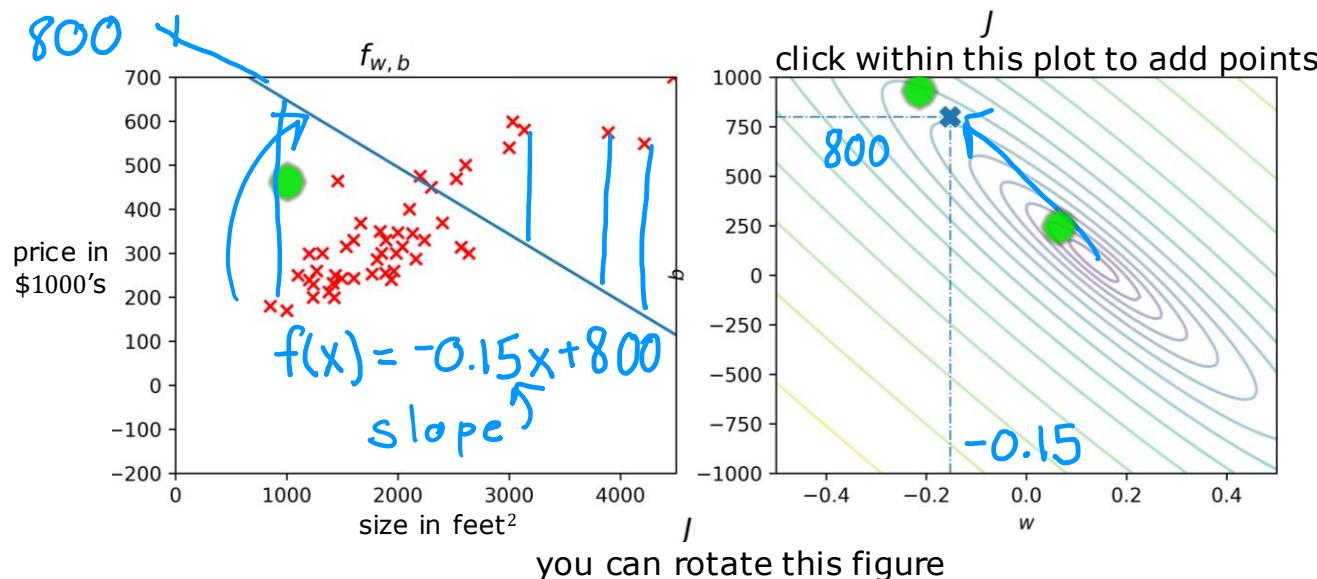
DeepLearning.AI



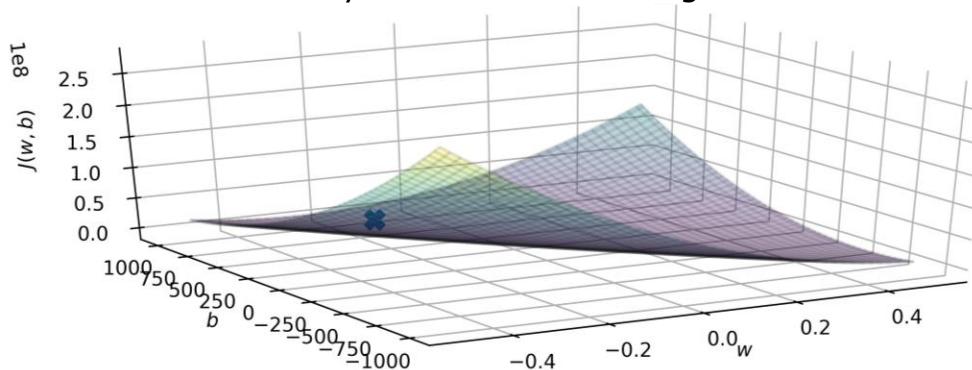
# Linear Regression with One Variable

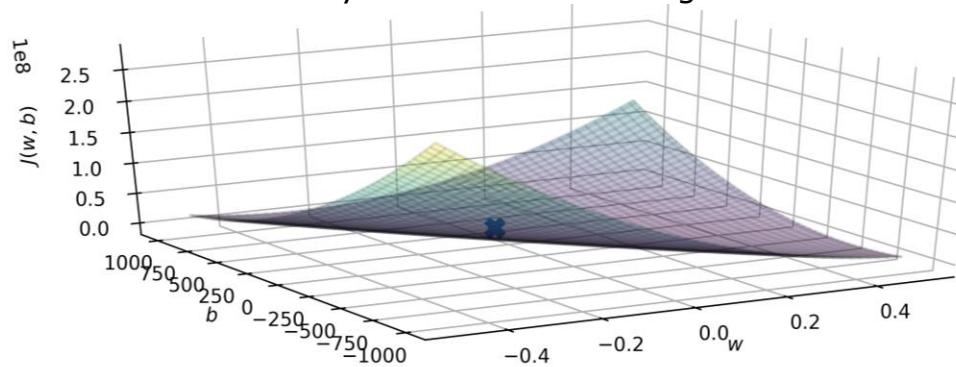
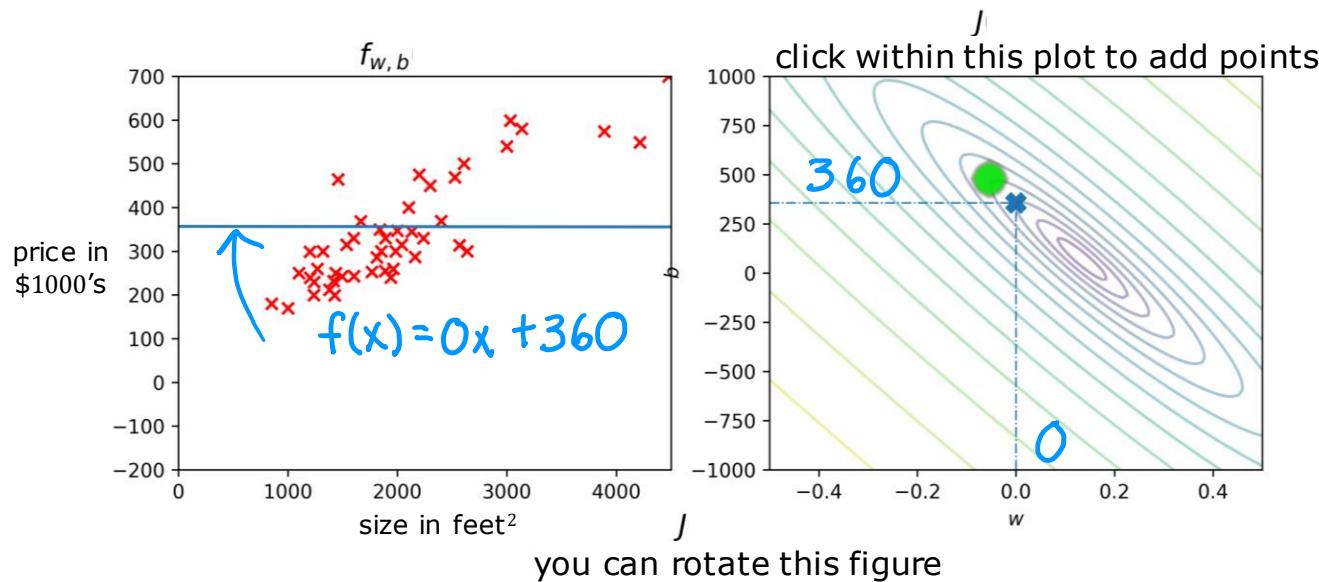
---

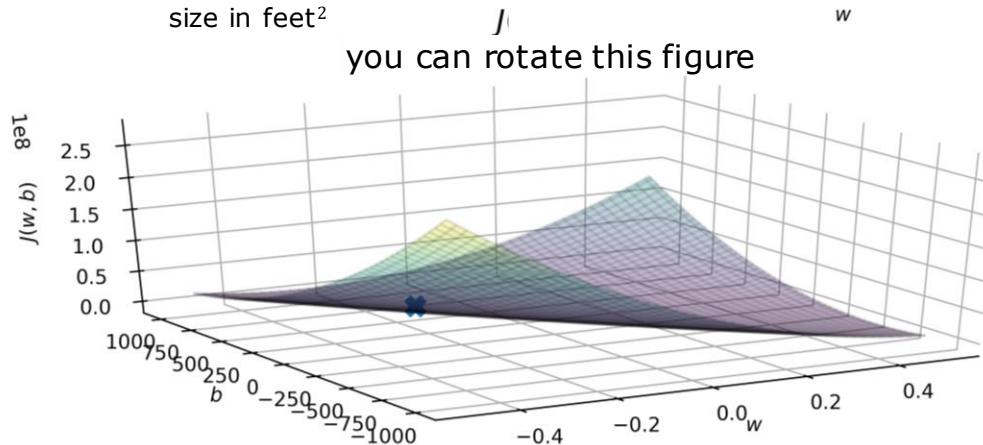
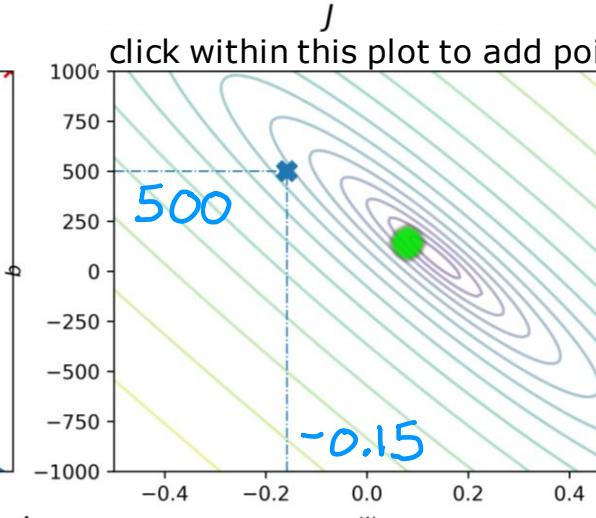
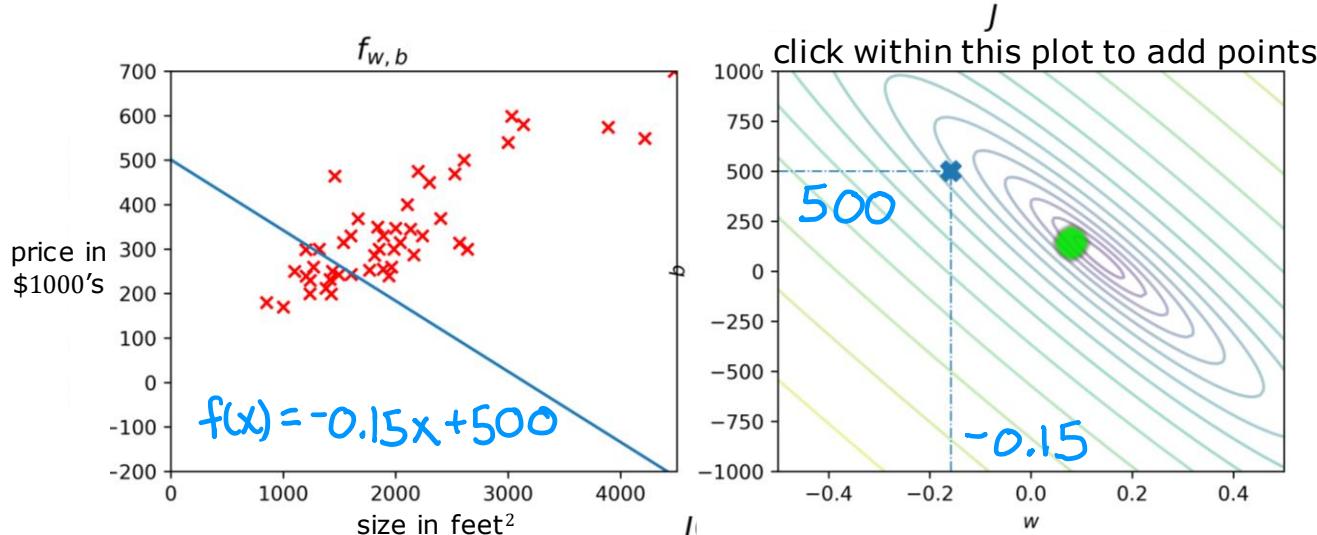
## Visualization examples

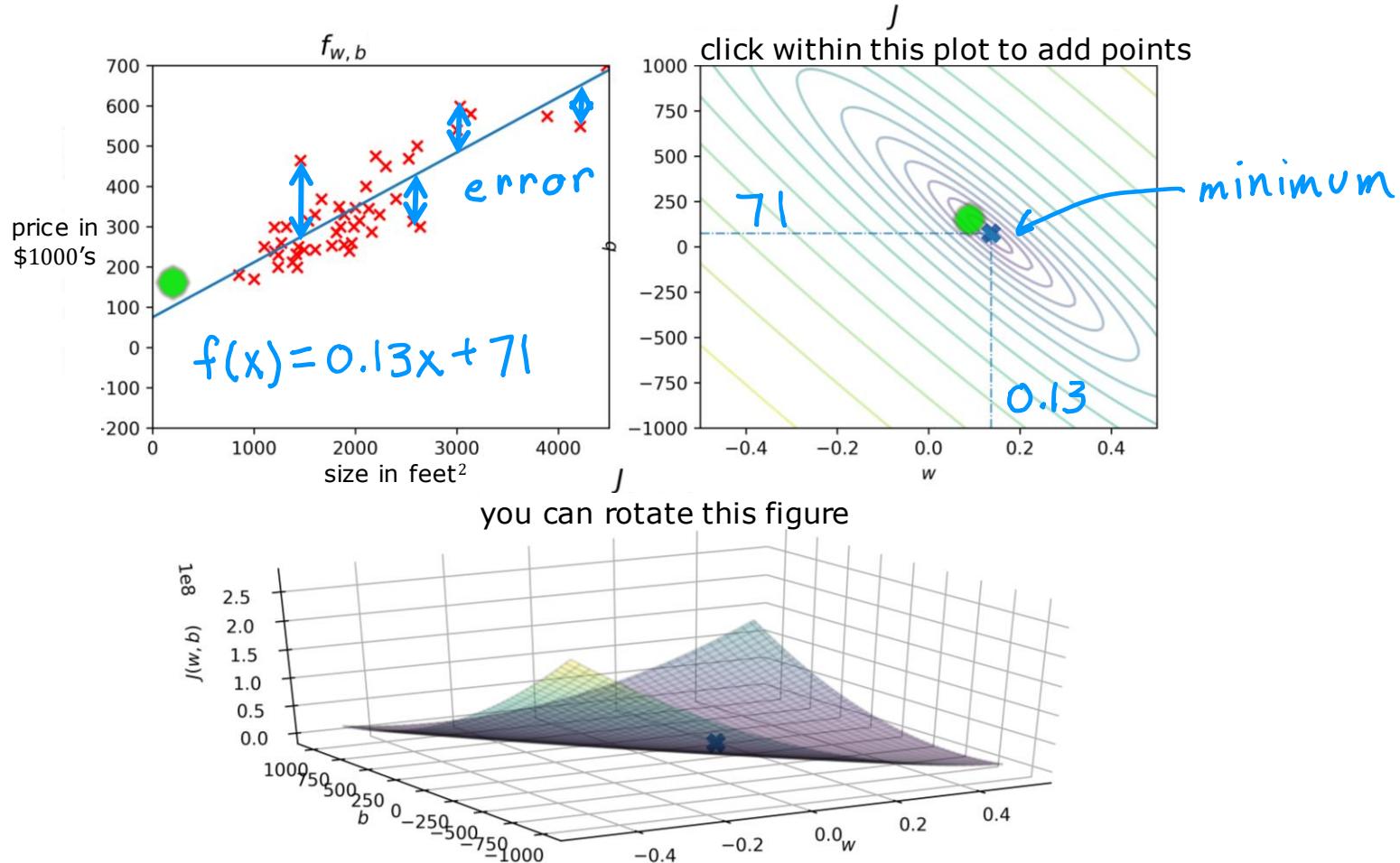


you can rotate this figure









Stanford  
ONLINE

DeepLearning.AI



# Training Linear Regression

---

## Gradient Descent

Have some function  $\underline{J(w, b)}$  for linear regression  
or any function

Want  $\min_{w, b} \underline{J(w, b)}$   $\min_{w_1, \dots, w_n, b} \underline{J(w_1, w_2, \dots, w_n, b)}$

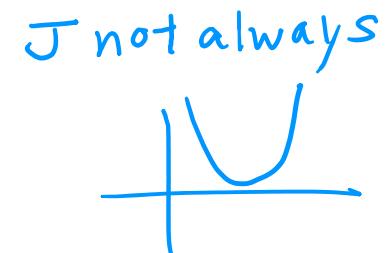
Outline:

Start with some  $\underline{w, b}$  (set  $w=0, b=0$ )

Keep changing  $w, b$  to reduce  $J(w, b)$

Until we settle at or near a minimum

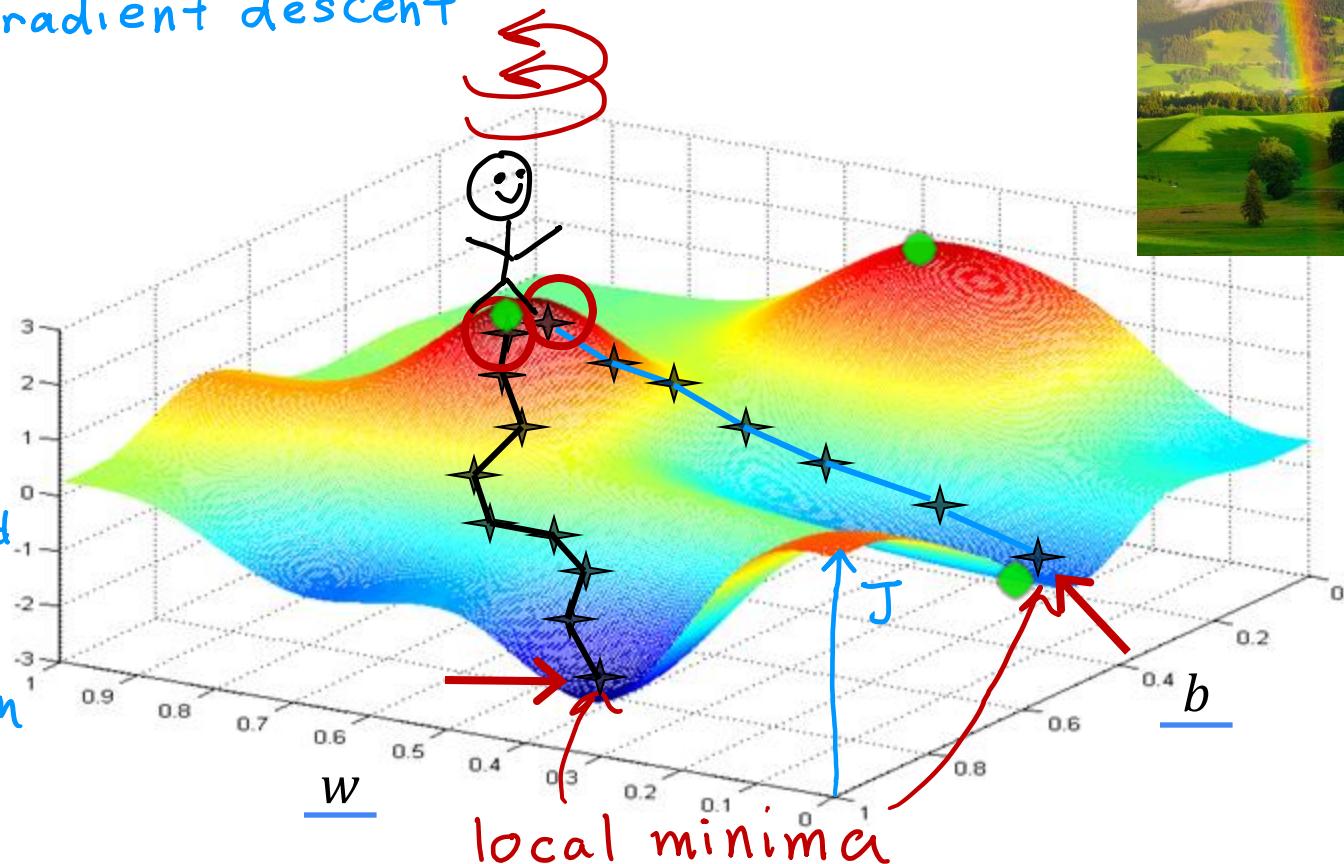
may have >1 minimum



gradient descent

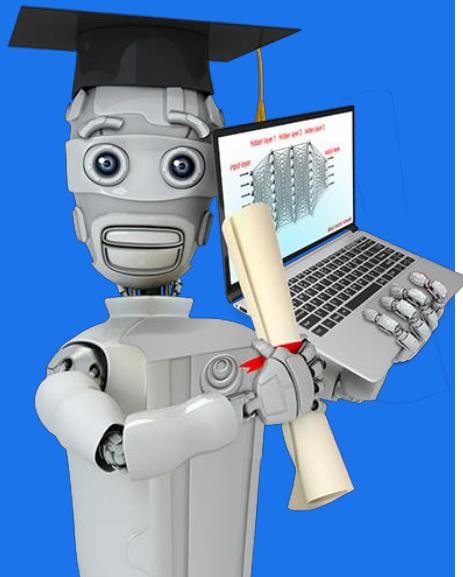
$$J(w, b)$$

not squared  
error cost  
not linear  
regression



Stanford  
ONLINE

DeepLearning.AI



# Training Linear Regression

---

## Implementing Gradient Descent

# Gradient descent algorithm

Repeat until convergence

$$\left\{ \begin{array}{l} w = w - \alpha \frac{\partial}{\partial w} J(w, b) \\ b = b - \alpha \frac{\partial}{\partial b} J(w, b) \end{array} \right.$$

Learning rate  
Derivative

Simultaneously  
update w and b

Assignment

$$a = c$$

$$a = a + 1$$

Code

Truth assertion

$$a = c$$

$$a = a + 1$$

Math

$$a == c$$

Correct: Simultaneous update

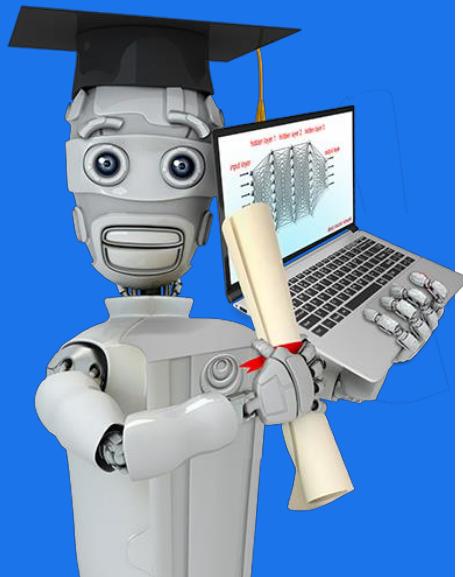
$$\left. \begin{array}{l} tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w, b) \\ tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w, b) \\ w = tmp\_w \\ b = tmp\_b \end{array} \right\}$$

Incorrect

$$\left. \begin{array}{l} tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w, b) \\ w = tmp\_w \\ tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w, b) \\ b = tmp\_b \end{array} \right\}$$

Stanford  
ONLINE

DeepLearning.AI



# Training Linear Regression

---

## Gradient Descent Intuition

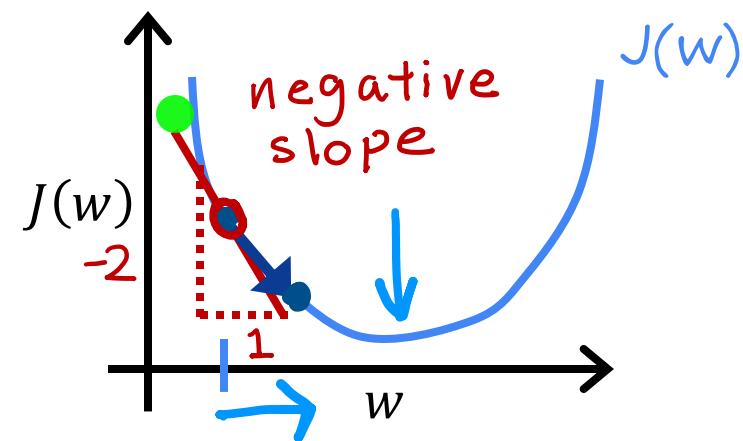
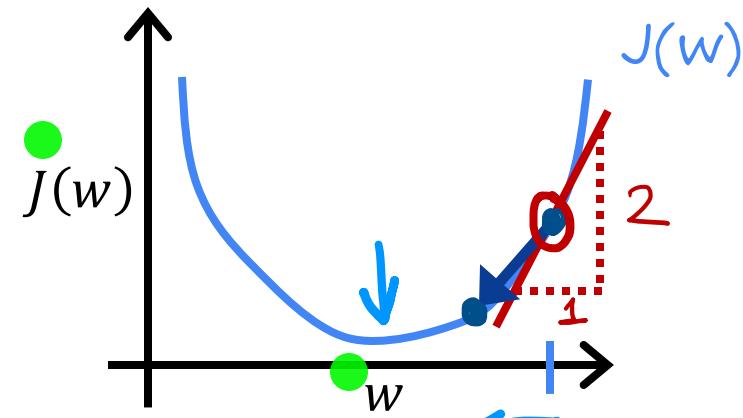
# Gradient descent algorithm

- repeat until convergence {  
  learning rate  $\alpha$   
     $w = w - \alpha \frac{\partial}{\partial w} J(w, b)$  *derivative*  
     $b = b - \alpha \frac{\partial}{\partial b} J(w, b)$

$$J(w)$$

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

$$\min_w J(w)$$



$$w = w - \alpha \frac{\frac{d}{dw} J(w)}{> 0}$$

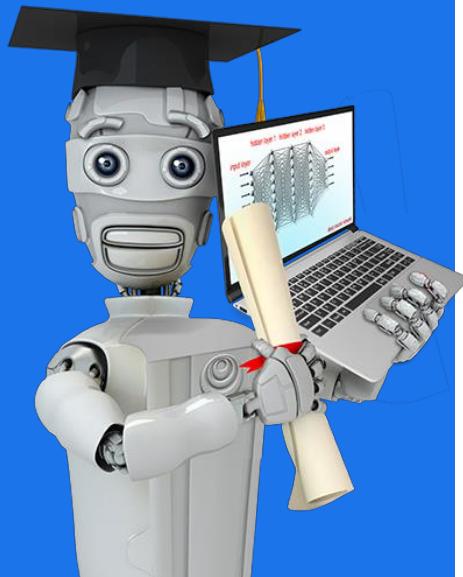
$w = w - \underline{\alpha} \cdot (\text{positive number})$

$$\frac{d}{dw} J(w) < 0$$

$w = w - \alpha \cdot (\text{negative number})$

Stanford  
ONLINE

DeepLearning.AI



# Training Linear Regression

---

## Learning Rate

$$w = w - \alpha \frac{d}{dw} J(w)$$

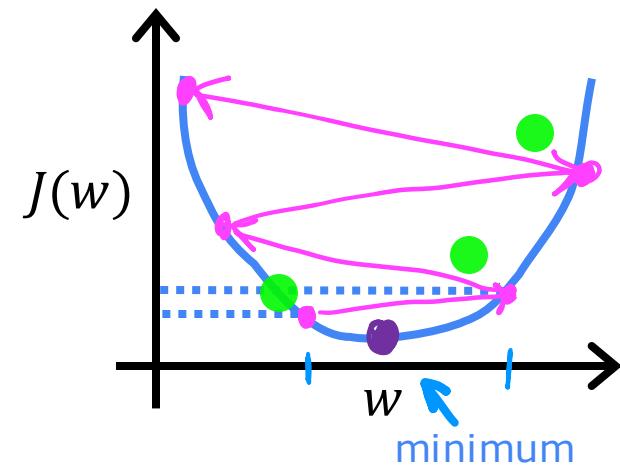
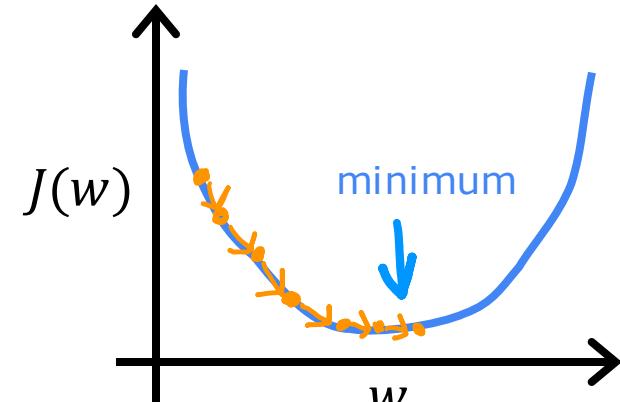
If  $\alpha$  is too small...

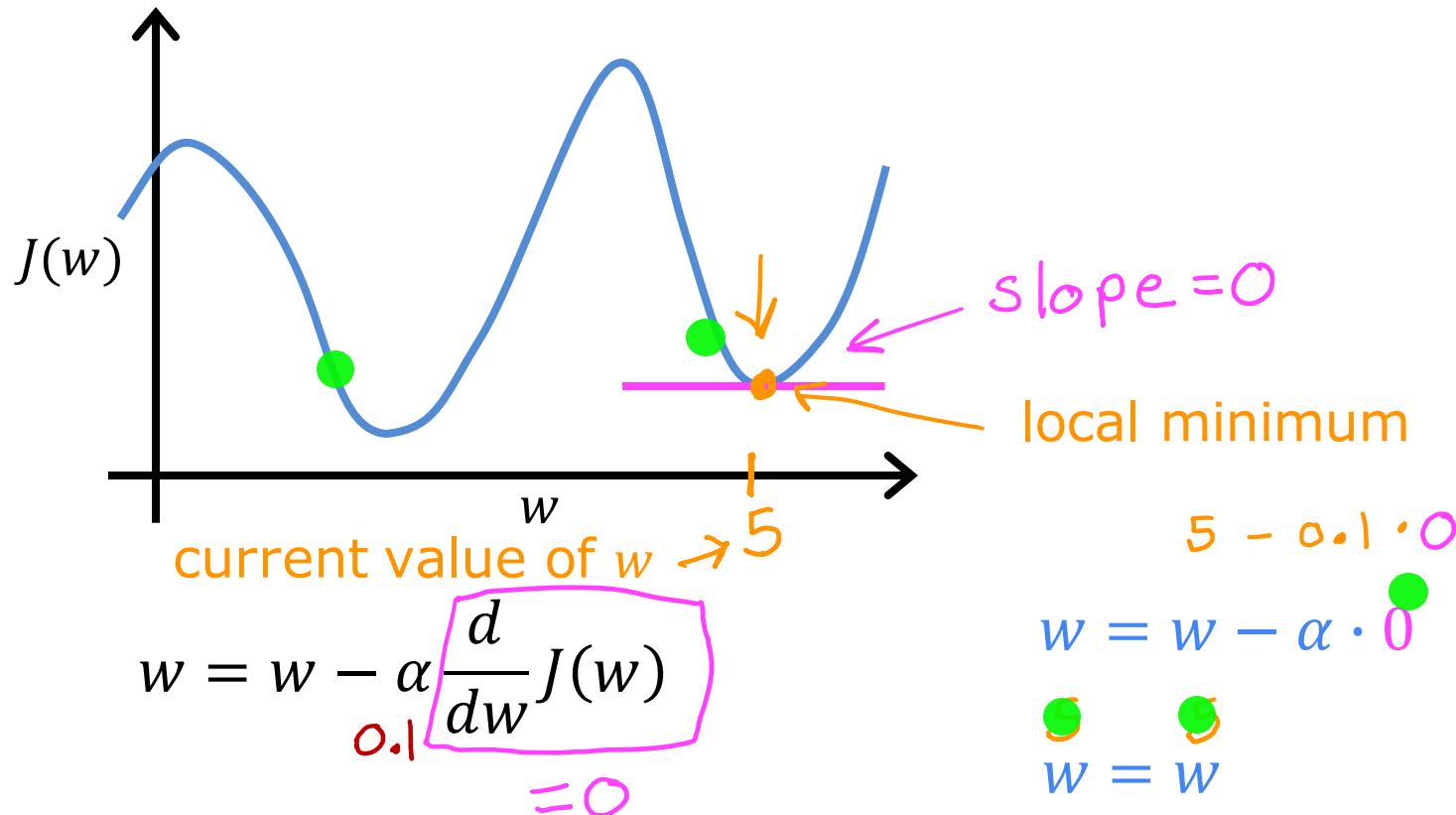
Gradient descent may be slow.

If  $\alpha$  is too large...

Gradient descent may:

- Overshoot, never reach minimum
- Fail to converge, diverge





Can reach local minimum with fixed learning rate

$w = w - \alpha \frac{d}{dw} J(w)$

smaller

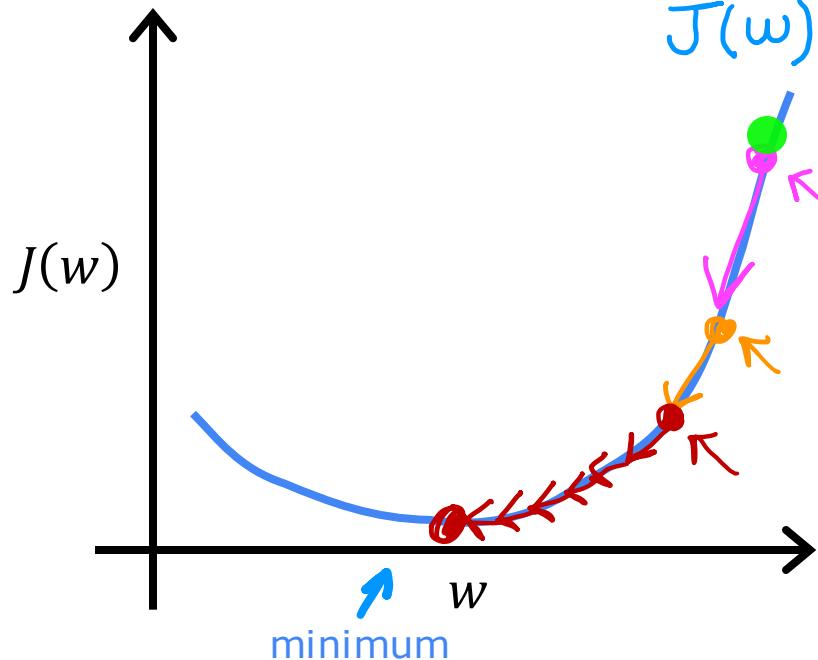
not as large

large

Near a local minimum,

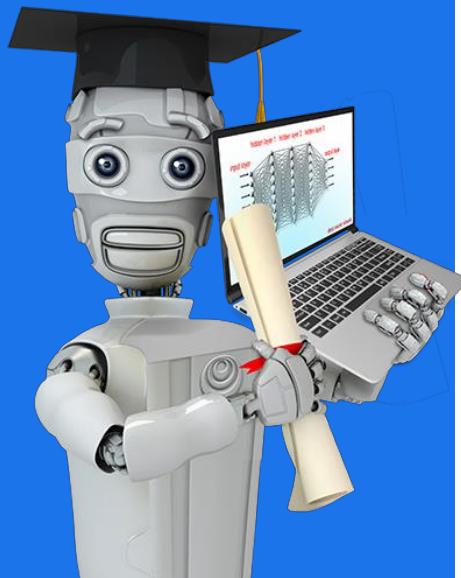
- Derivative becomes smaller
- Update steps become smaller

Can reach minimum without decreasing learning rate  $\alpha$



Stanford  
ONLINE

DeepLearning.AI



# Training Linear Regression

---

Gradient Descent  
for Linear Regression

## Linear regression model

$$f_{w,b}(x) = wx + b$$

## Cost function

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

## Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

}

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

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

next slide  
is optional!

(Optional)

$$\frac{\partial}{\partial w} J(w, b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^m (\underline{wx^{(i)} + b} - y^{(i)})^2$$

$$= \cancel{\frac{1}{2m}} \sum_{i=1}^m (\underline{wx^{(i)} + b} - y^{(i)}) \cancel{2x^{(i)}} = \boxed{\frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}}$$

$$\frac{\partial}{\partial b} J(w, b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^m (\underline{wx^{(i)} + b} - y^{(i)})^2$$

$$= \cancel{\frac{1}{2m}} \sum_{i=1}^m (\underline{wx^{(i)} + b} - y^{(i)}) \cancel{2} = \boxed{\frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})}$$

no  $x^{(i)}$

# Gradient descent algorithm

repeat until convergence {

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

}

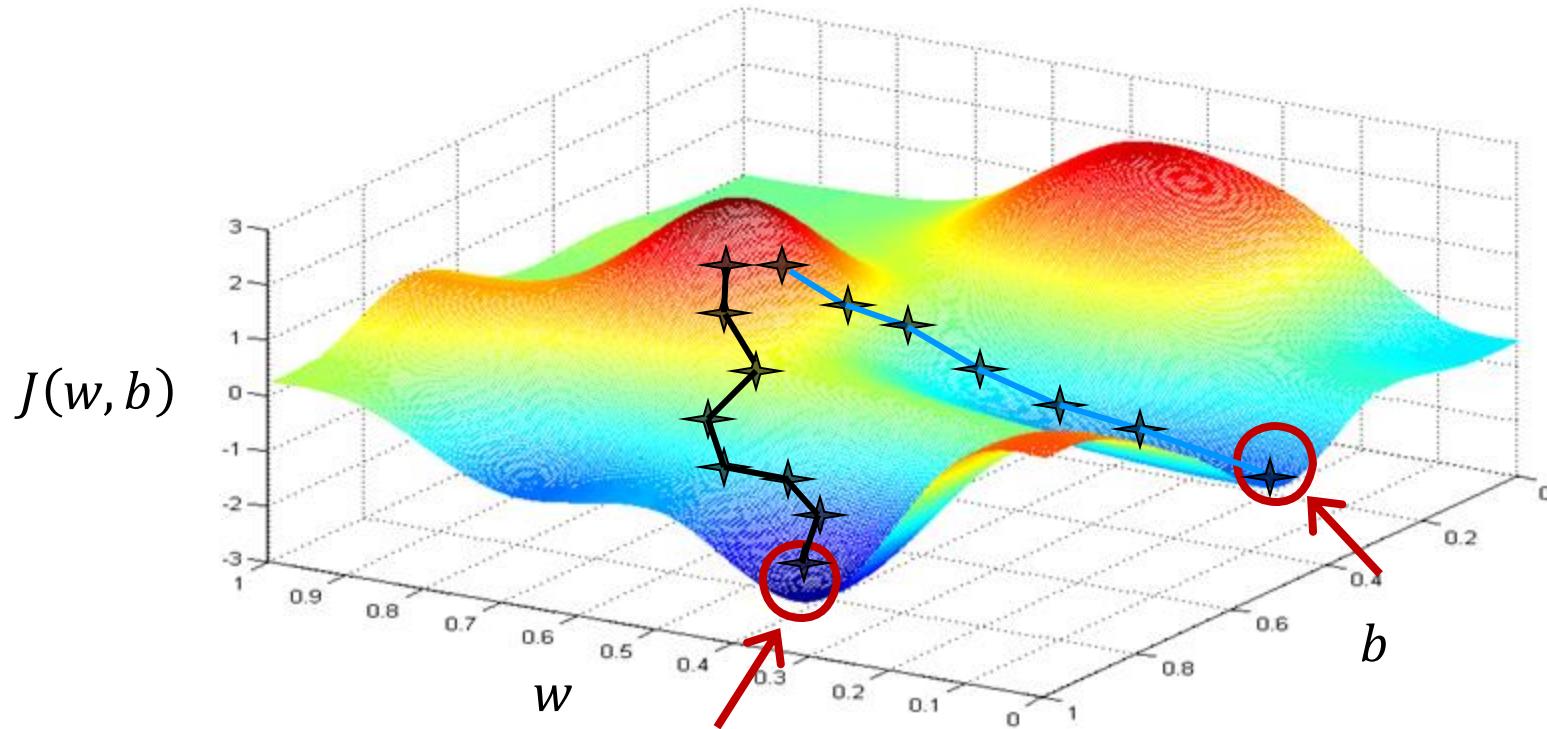
Update w and b simultaneously

$\frac{\partial}{\partial w} J(w, b)$

$\frac{\partial}{\partial b} J(w, b)$

$f_{w,b}(x^{(i)}) = wx^{(i)} + b$

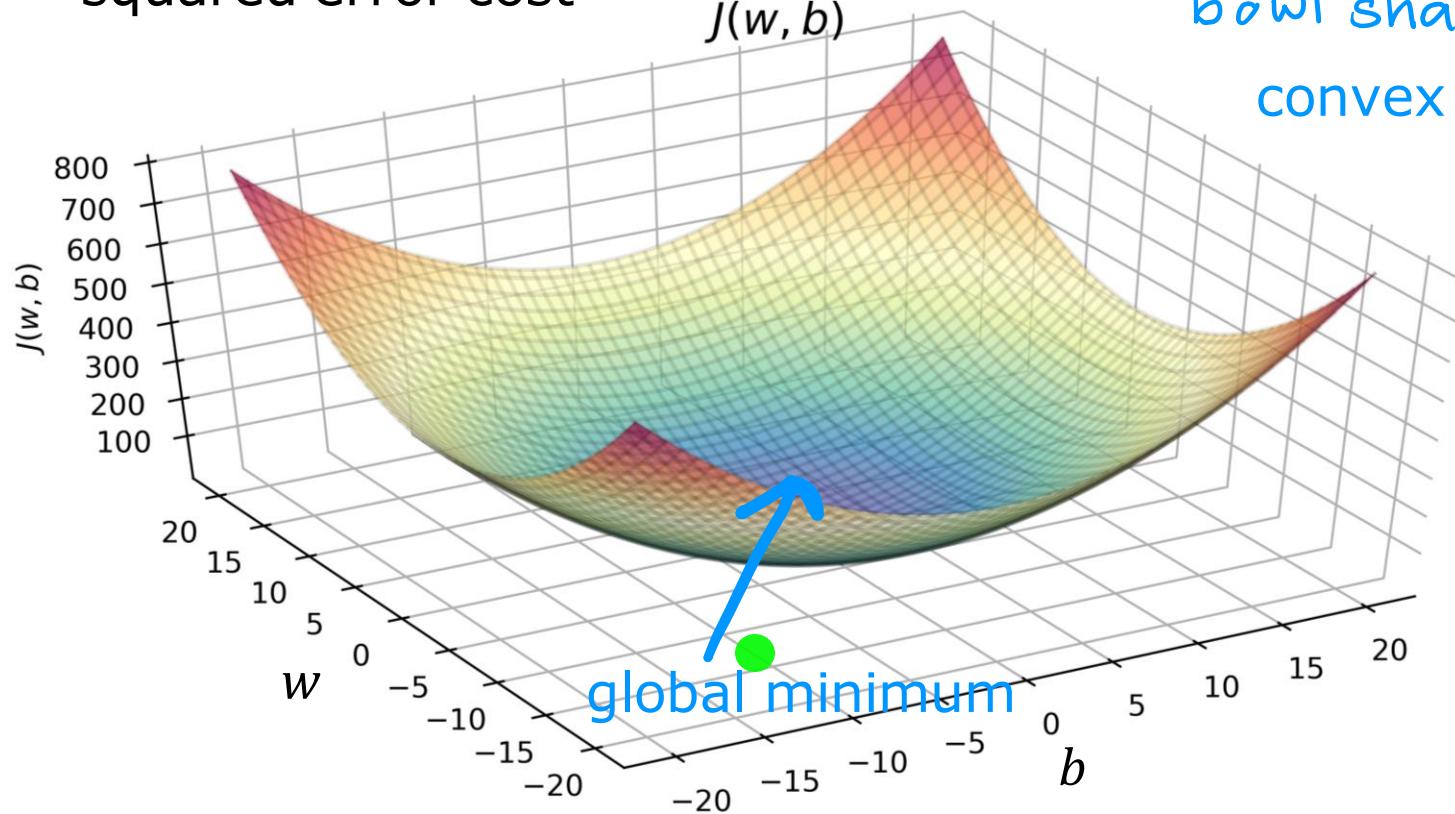
# More than one local minimum



squared error cost

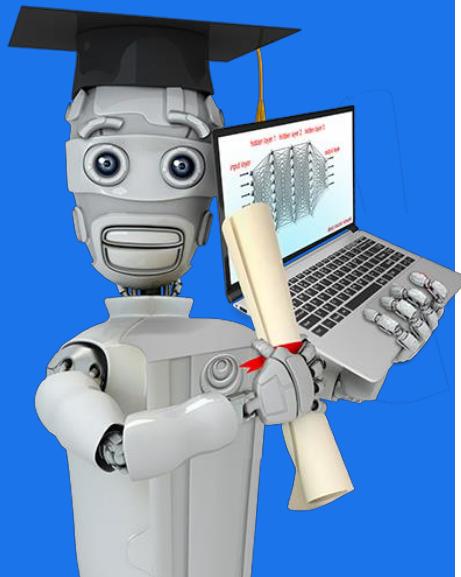
$$J(w, b)$$

bowl shape  
convex function



Stanford  
ONLINE

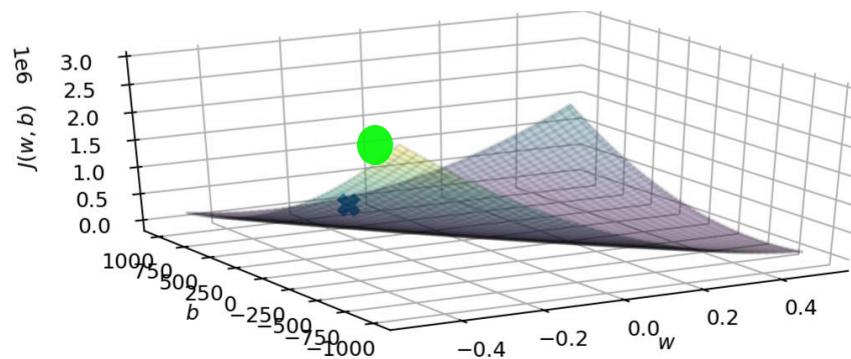
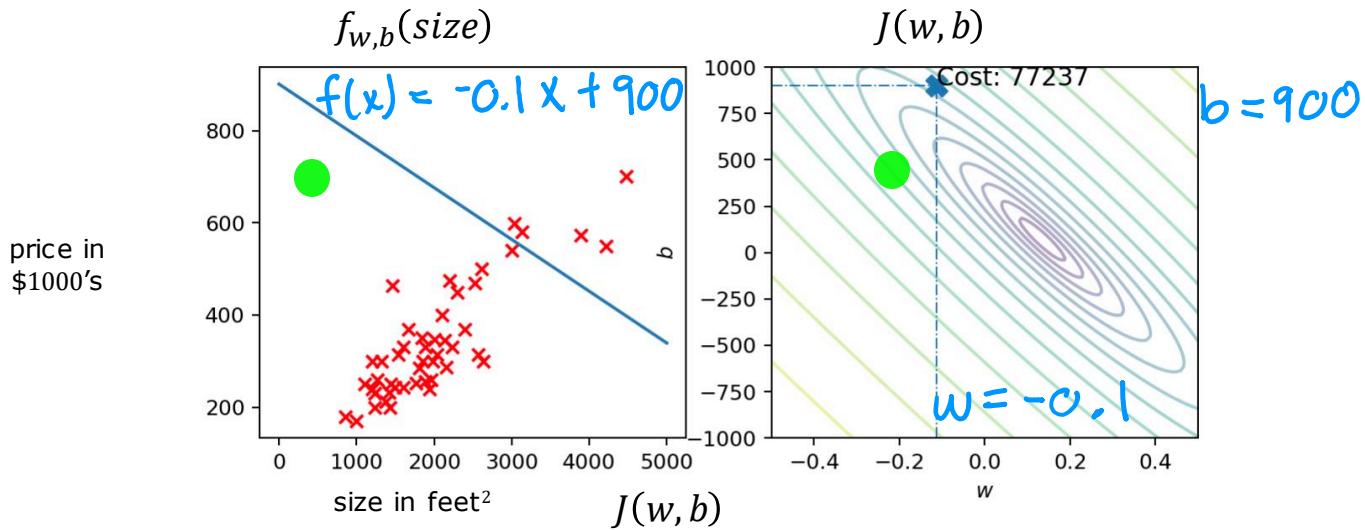
DeepLearning.AI

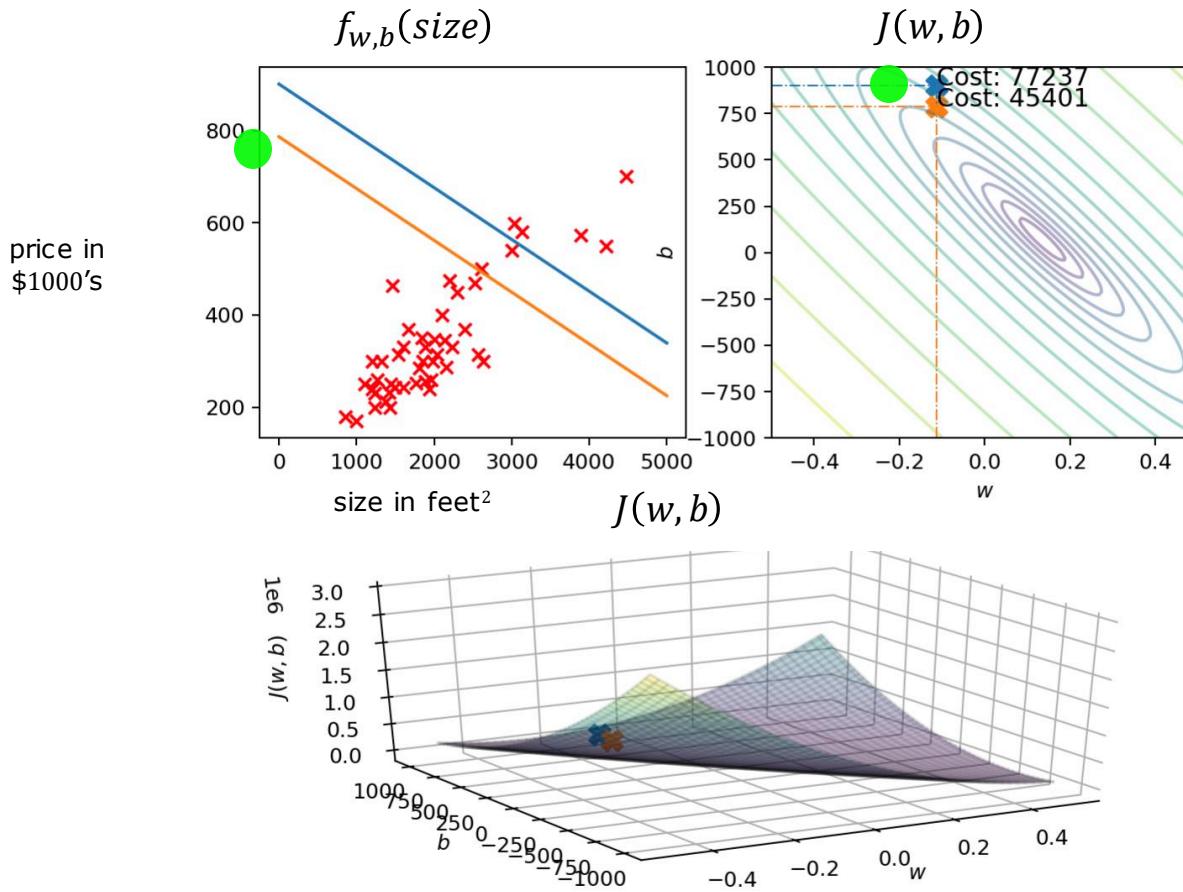


# Training Linear Regression

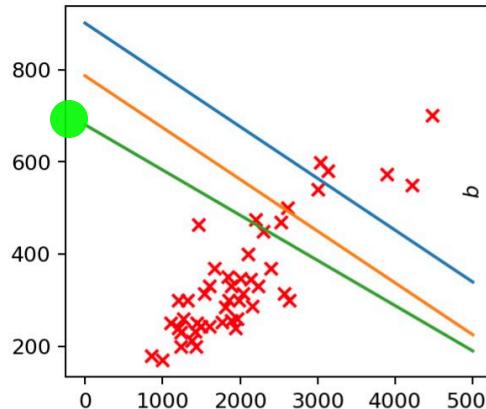
---

Running  
Gradient Descent

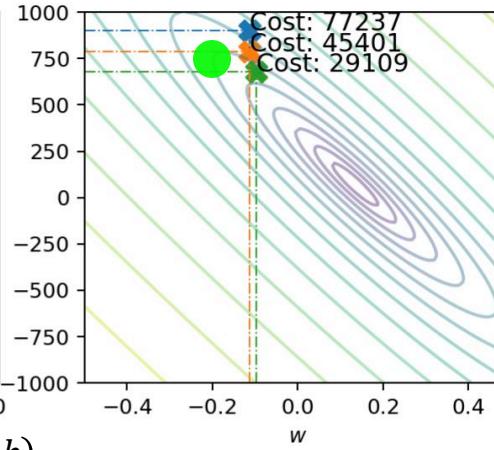




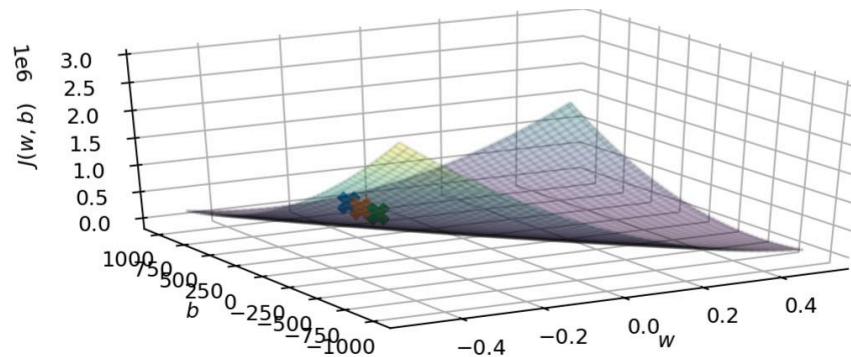
$f_{w,b}(\text{size})$

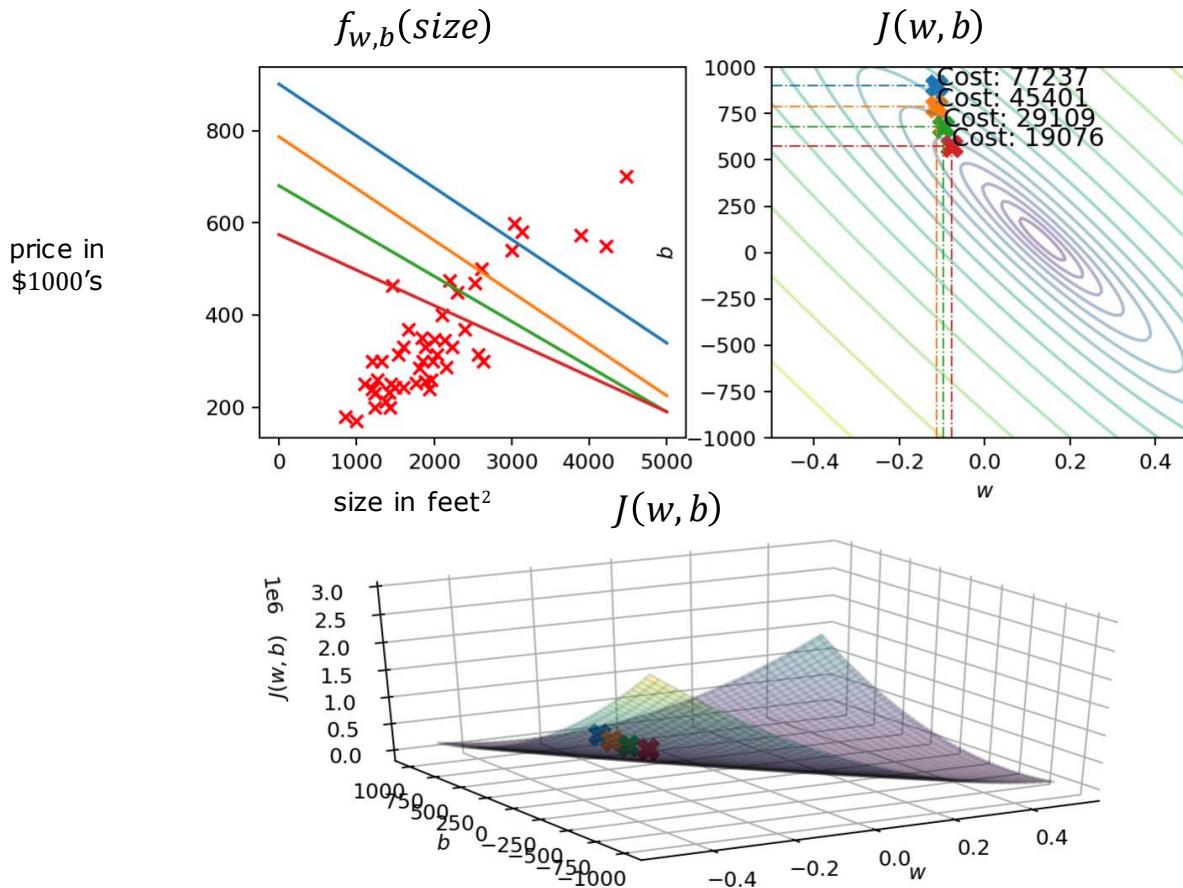


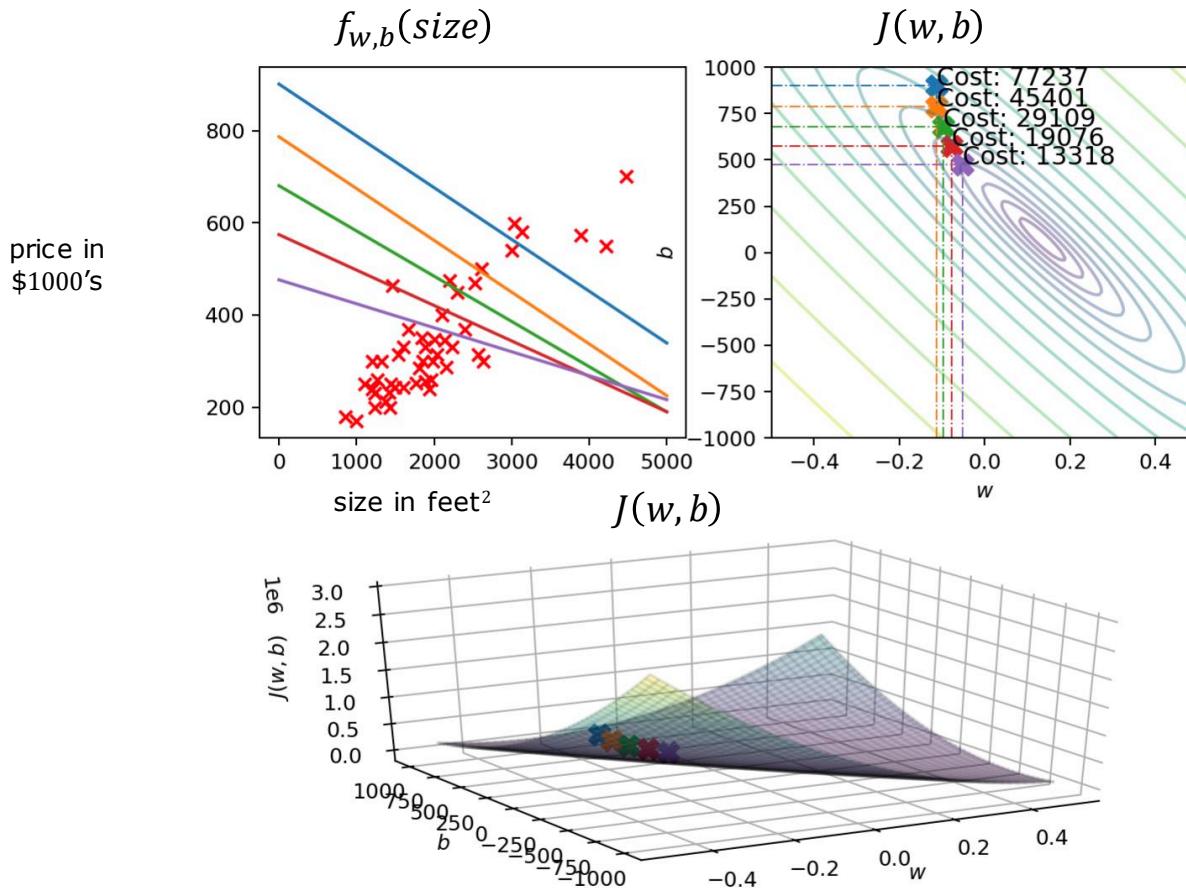
$J(w, b)$

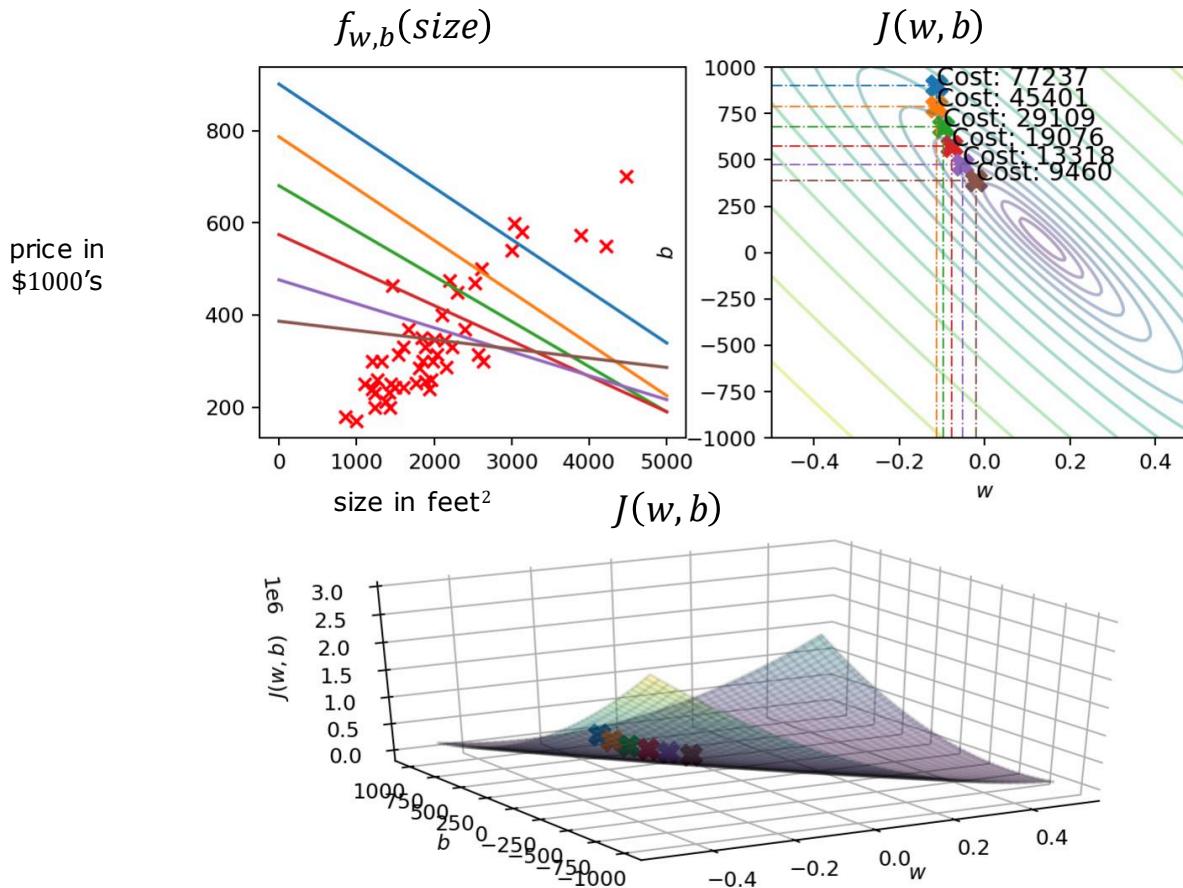


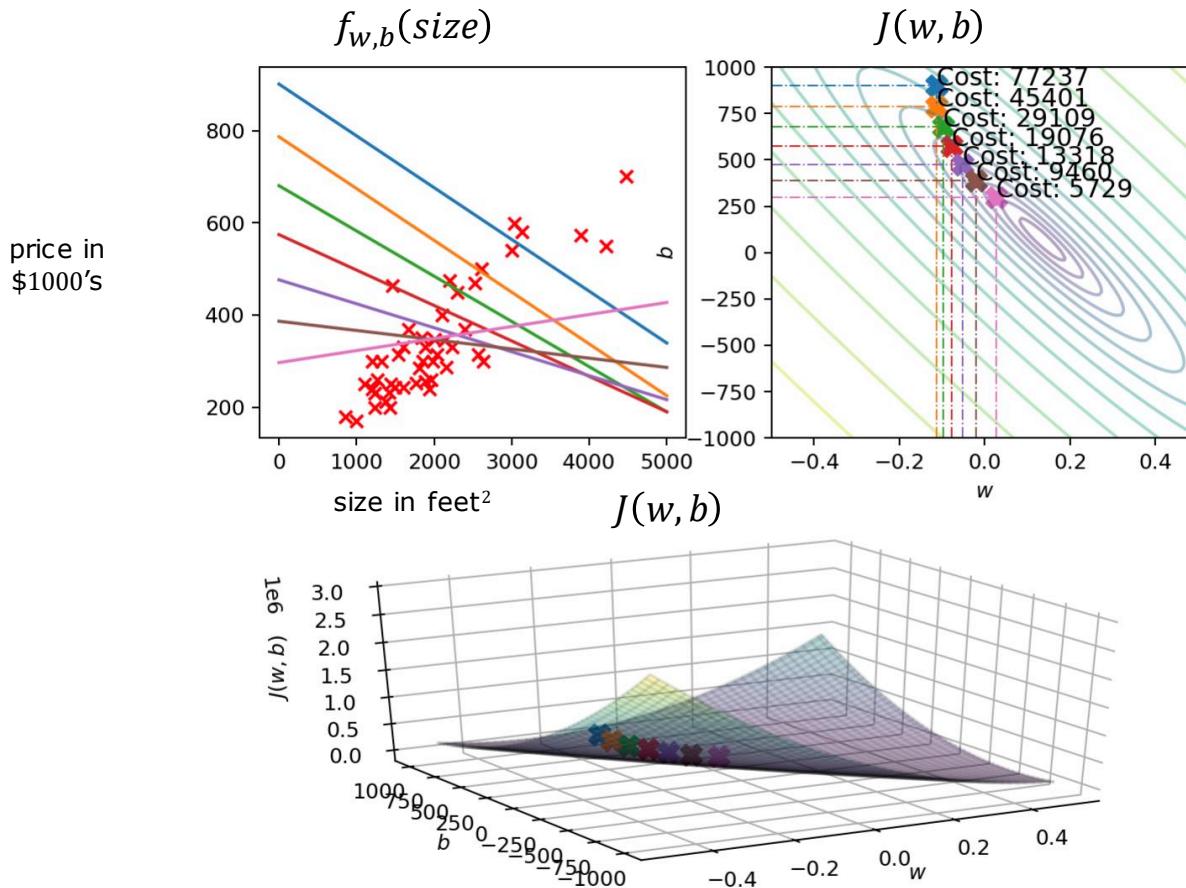
$J(w, b)$



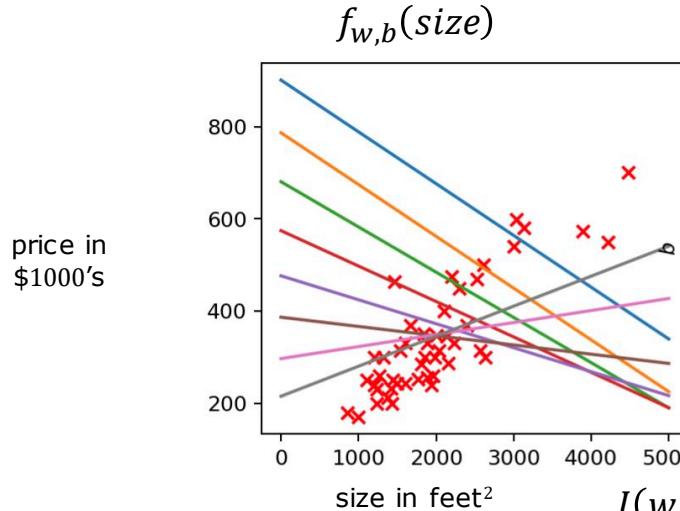




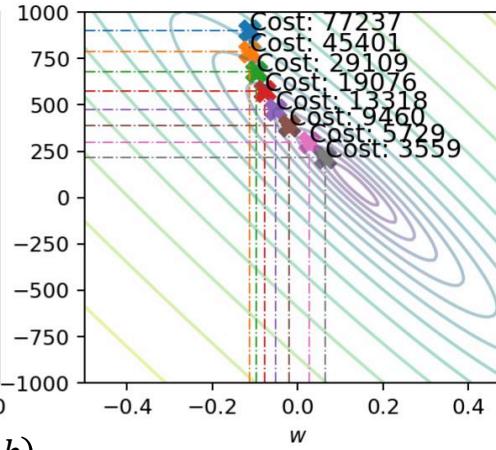




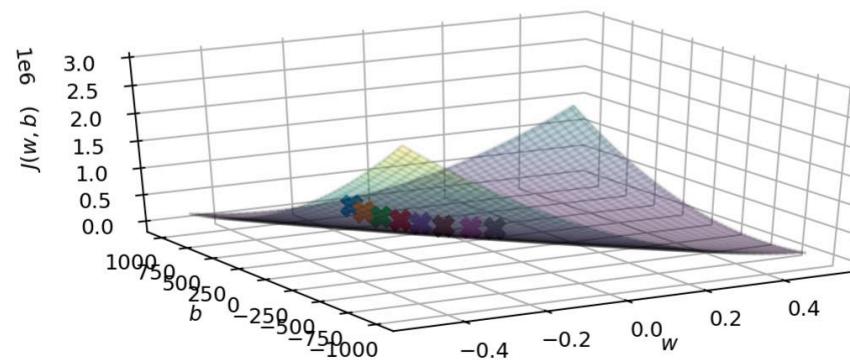
$$f_{w,b}(\text{size})$$

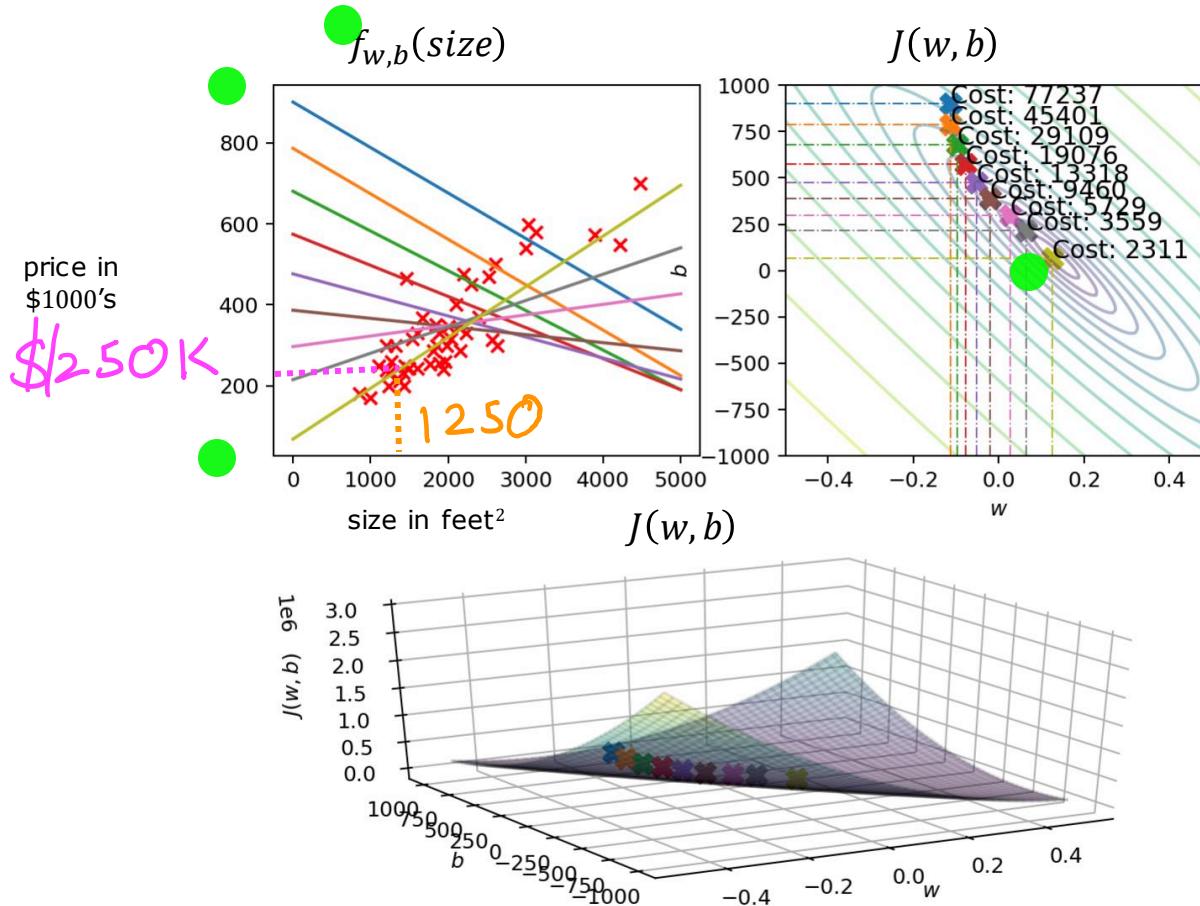


$$J(w, b)$$



$$J(w, b)$$

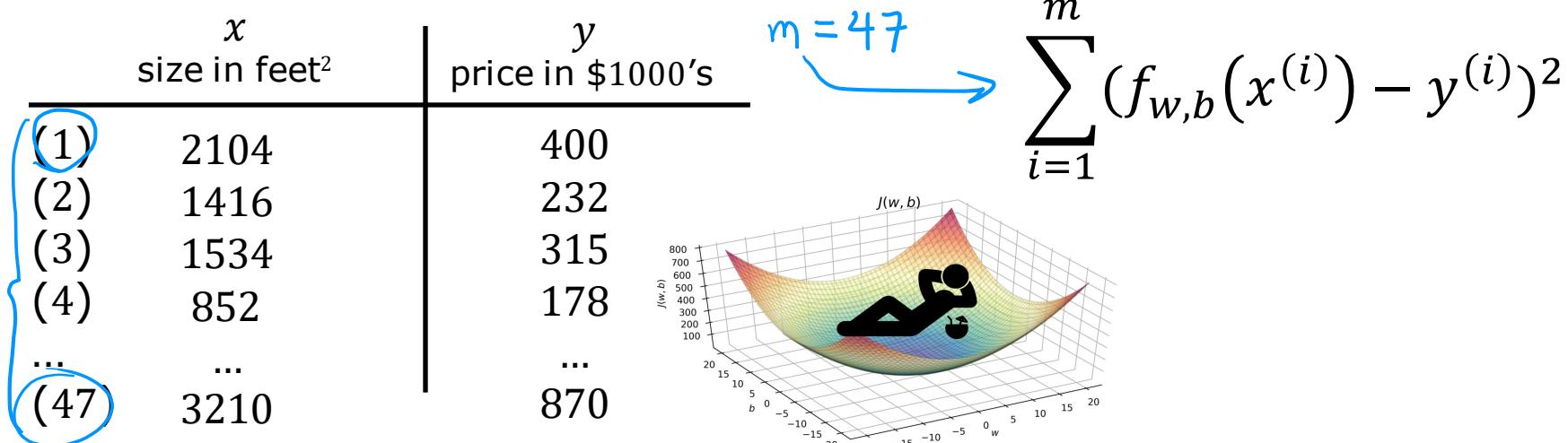




# “Batch” gradient descent

“Batch”: Each step of gradient descent uses all the training examples.

other gradient descent: subsets



# Copyright Notice

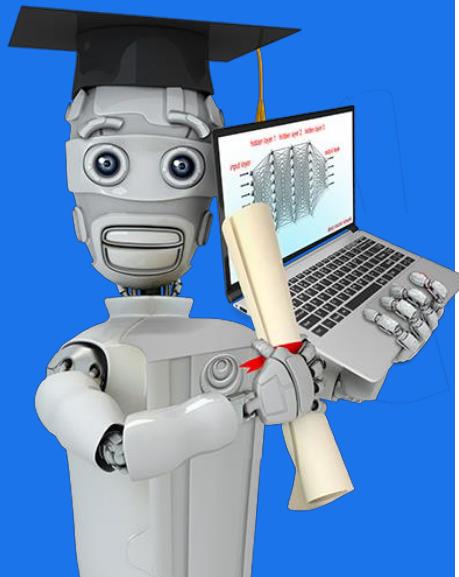
These slides are distributed under the Creative Commons License.

[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>

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with Multiple Variables

## Multiple Features

# Multiple features (variables)

One  
feature



Size in feet <sup>2</sup> ( $x$ )	Price (\$) in 1000's ( $y$ )
2104	400
1416	232
1534	315
852	178
...	...

$$f_{w,b}(x) = wx + b$$

# Multiple features (variables)

Size in feet <sup>2</sup>	Number of bedrooms	Number of floors	Age of home in years	Price (\$) in \$1000's
$x_1$	$x_2$	$x_3$	$x_4$	
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$i=2$

$x_j = j^{th}$  feature

•  $n$  = number of features

$\vec{x}^{(i)}$  = features of  $i^{th}$  training example

$x_j^{(i)}$  = value of feature  $j$  in  $i^{th}$  training example

$j=1\dots 4$   
 $n=4$

$$\vec{x}^{(2)} = [1416 \ 3 \ 2 \ 40]$$

$$x_3^{(2)} = 2$$

# Model:

Previously:  $f_{w,b}(x) = wx + b$

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

example

$$f_{w,b}(x) = 0.1 \underset{\text{size}}{\uparrow} x_1 + 4 \underset{\text{\#bedrooms}}{\uparrow} x_2 + 10 \underset{\text{\#floors}}{\uparrow} x_3 + -2 \underset{\text{years}}{\uparrow} x_4 + 80 \underset{\text{base price}}{\uparrow}$$

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

$$f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

$$\vec{w} = [w_1 \ w_2 \ w_3 \dots w_n]$$

b is a number

parameters  
of the model

vector  $\vec{x} = [x_1 \ x_2 \ x_3 \dots x_n]$

$$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n + b$$

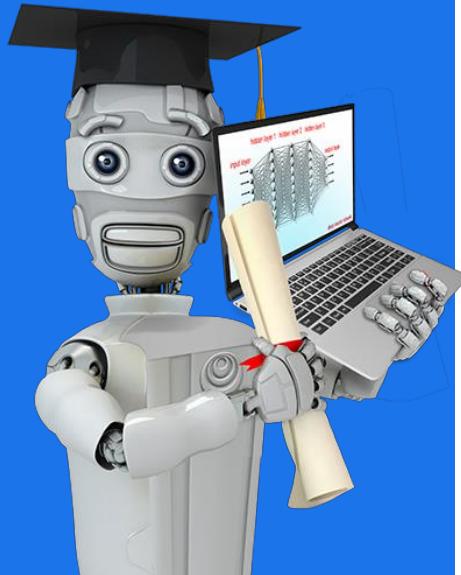
dot product

multiple linear regression

(not multivariate regression)

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with Multiple Variables

---

## Vectorization Part 1

## Parameters and features

$$\vec{w} = [w_1 \ w_2 \ w_3] \quad n=3$$

$b$  is a number

$$\vec{x} = [x_1 \ x_2 \ x_3]$$

linear algebra: count from 1

$$w[0] \quad w[1] \quad w[2]$$

```
w = np.array([1.0, 2.5, -3.3])
```

```
b = 4
```

$$x[0] \ x[1] \ x[2]$$

```
x = np.array([10, 20, 30])
```

code: count from 0

## Without vectorization $n=100,000$

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

```
f = w[0] * x[0] +  
    w[1] * x[1] +  
    w[2] * x[2] + b
```

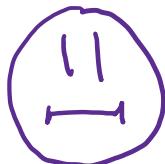


## Without vectorization

$$f_{\vec{w},b}(\vec{x}) = \left( \sum_{j=1}^n w_j x_j \right) + b \quad \sum_{j=1}^n \rightarrow j=1 \dots n \\ 1, 2, 3$$

$$\text{range}(0, n) \rightarrow j=0 \dots n-1$$

```
f = 0  
for j in range(0, n):  
    f = f + w[j] * x[j]  
f = f + b
```



## Vectorization

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

```
f = np.dot(w, x) + b
```



Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with Multiple Variables

---

## Vectorization Part 2

## Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

$t_0 \quad f + w[0] * x[0]$

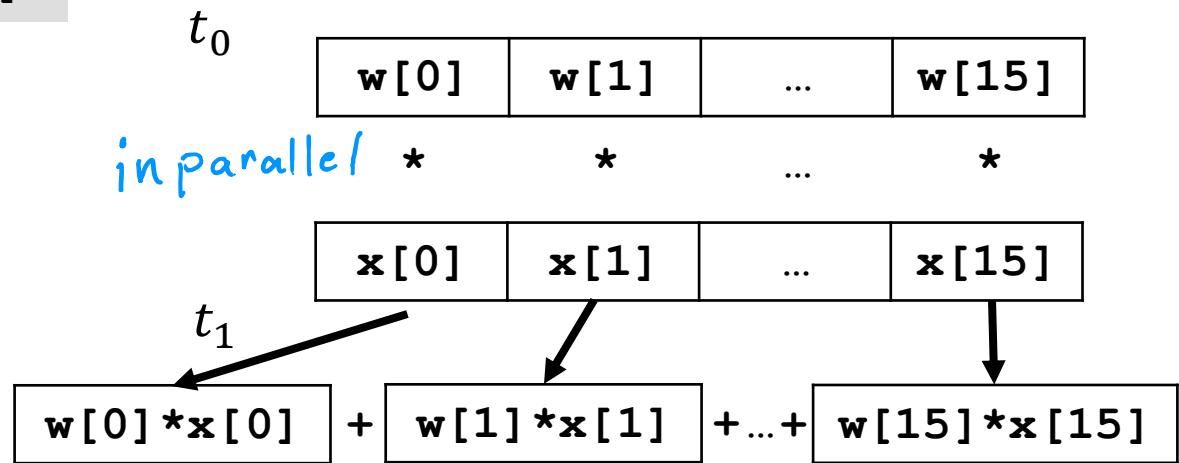
$t_1 \quad f + w[1] * x[1]$

$\dots$

$t_{15} \quad f + w[15] * x[15]$

## Vectorization

```
np.dot(w,x)
```



efficient → scale to large datasets

Gradient descent       $\vec{w} = (w_1 \quad w_2 \quad \dots \quad w_{16})$  ~~b~~ parameters  
derivatives       $\vec{d} = (d_1 \quad d_2 \quad \dots \quad d_{16})$

```
w = np.array([0.5, 1.3, ... 3.4])
```

```
d = np.array([0.3, 0.2, ... 0.4])
```

compute  $w_j = w_j - \underbrace{0.1d_j}_{\text{learning rate } \alpha}$  for  $j = 1 \dots 16$

### Without vectorization

$$w_1 = w_1 - 0.1d_1$$

$$w_2 = w_2 - 0.1d_2$$

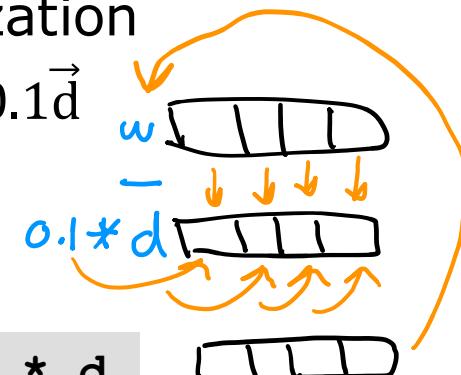
:

$$w_{16} = w_{16} - 0.1d_{16}$$

```
for j in range(0,16):  
    w[j] = w[j] - 0.1 * d[j]
```

### With vectorization

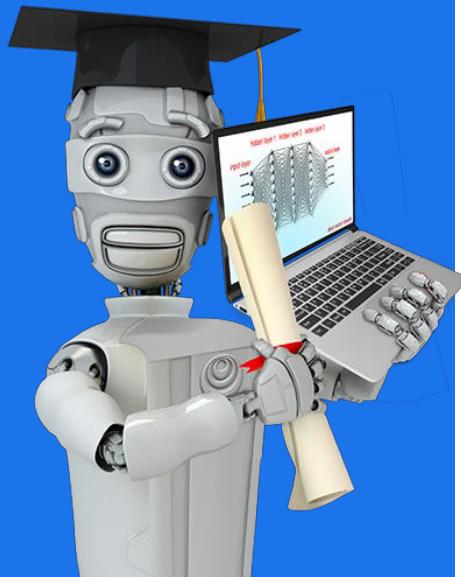
$$\vec{w} = \vec{w} - 0.1\vec{d}$$



```
w = w - 0.1 * d
```

Stanford  
ONLINE

DeepLearning.AI



# Linear Regression with Multiple Variables

---

## Gradient Descent for Multiple Regression

## Previous notation

Parameters

$$w_1, \dots, w_n$$

$$\bullet \quad b$$

Model  $f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + \dots + w_n x_n + b$

Cost function  $J(\underbrace{w_1, \dots, w_n}_\bullet, b)$

Gradient descent

```
repeat {  
     $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\underbrace{w_1, \dots, w_n}_\bullet, b)$   
     $b = b - \alpha \frac{\partial}{\partial b} J(\underbrace{w_1, \dots, w_n}_\bullet, b)$   
}
```

## Vector notation

$\vec{w}$  *vector of length n*  
 $\vec{w} = [w_1 \dots w_n]$   
 $b$  still a number  
 $f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$   
 $J(\vec{w}, b)$  dot product

```
repeat {  
     $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$   
     $b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$   
}
```

# Gradient descent

One feature

repeat {

$$\underline{w} = w - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\underline{w}, b}(x^{(i)}) - y^{(i)}) \underline{x}^{(i)}$$

$\hookrightarrow \frac{\partial}{\partial \underline{w}} J(\underline{w}, b)$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\underline{w}, b}(x^{(i)}) - y^{(i)})$$

simultaneously update  $w, b$

}

$n$  features ( $n \geq 2$ )

repeat {

$$\begin{aligned} j &= 1 & \underline{w}_1 &= w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\overrightarrow{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \underline{x}_1^{(i)} \\ &\vdots & & \hookrightarrow \frac{\partial}{\partial \underline{w}_1} J(\overrightarrow{w}, b) \\ j &= n & w_n &= w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\overrightarrow{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)} \end{aligned}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\overrightarrow{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

simultaneously update  
 $w_j$  (for  $j = 1, \dots, n$ ) and  $b$

}

# An alternative to gradient descent

## → Normal equation

- Only for linear regression
- Solve for  $w$ ,  $b$  without iterations

## Disadvantages

- Doesn't generalize to other learning algorithms.
- Slow when number of features is large ( $> 10,000$ )

## What you need to know

- Normal equation method may be used in machine learning libraries that implement linear regression.
- Gradient descent is the recommended method for finding parameters  $w, b$



Stanford  
ONLINE

DeepLearning.AI



# Practical Tips for Linear Regression

---

## Feature Scaling Part 1

# Feature and parameter values

$$\widehat{\text{price}} = w_1 x_1 + w_2 x_2 + b$$

↓      ↓  
size   #bedrooms

$x_1$ : size (feet<sup>2</sup>)  
range: 300 – 2,000

$x_2$ : # bedrooms  
range: 0 – 5



House:  $x_1 = 2000$ ,  $x_2 = 5$ ,  $\text{price} = \$500k$

one training example

size of the parameters  $w_1, w_2$ ?

•  $w_1 = 50, w_2 = 0.1, b = 50$

$$\widehat{\text{price}} = \underbrace{50 * 2000}_{100,000K} + \underbrace{0.1 * 5}_{0.5K} + \underbrace{50}_{50K}$$

$$\widehat{\text{price}} = \$100,050.5K = \$100,050,500$$

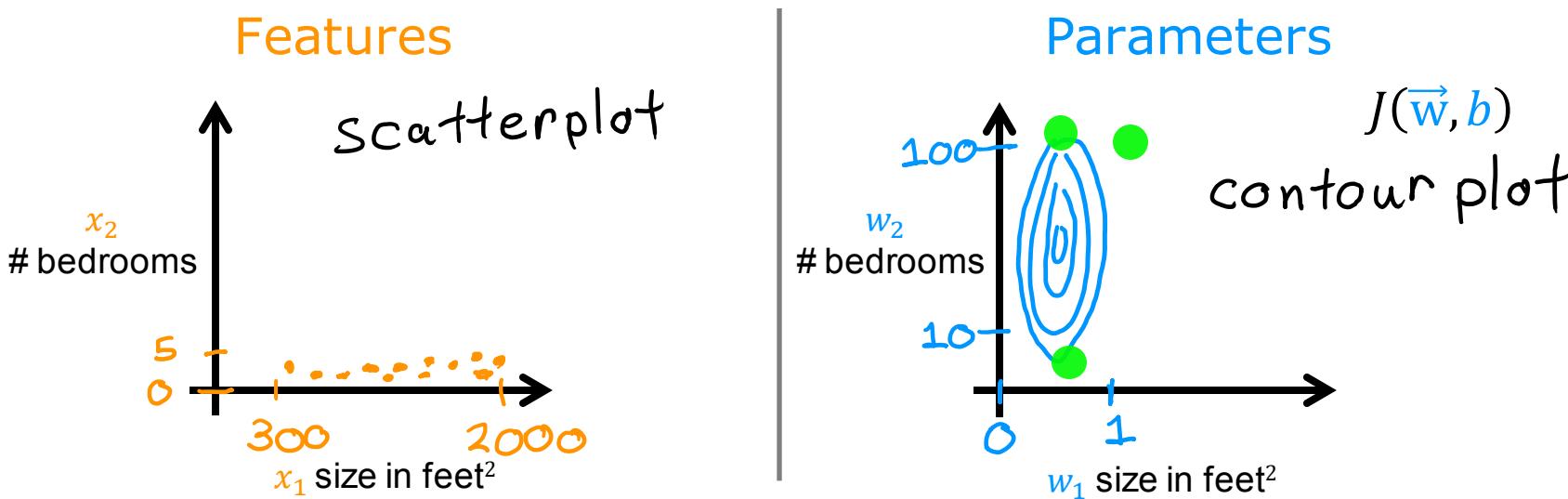
→  
 $w_1 = 0.1, w_2 = 50, b = 50$

$$\widehat{\text{price}} = \underbrace{0.1 * 2000}_{200K} + \underbrace{50 * 5}_{250K} + \underbrace{50}_{50K}$$

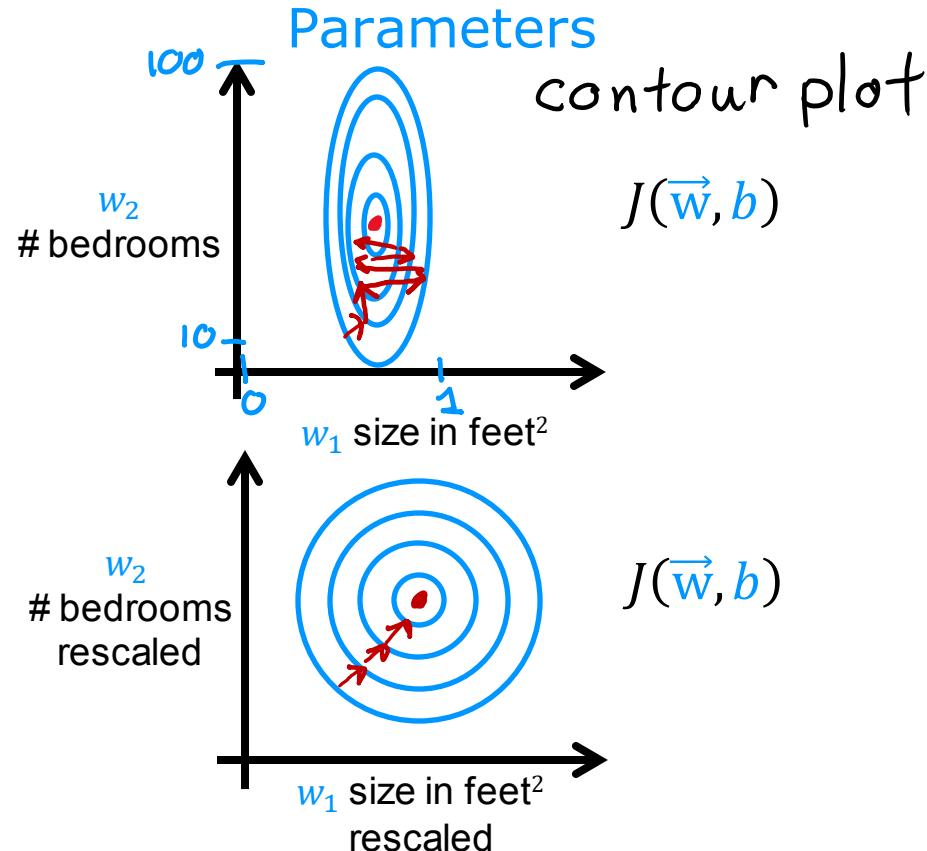
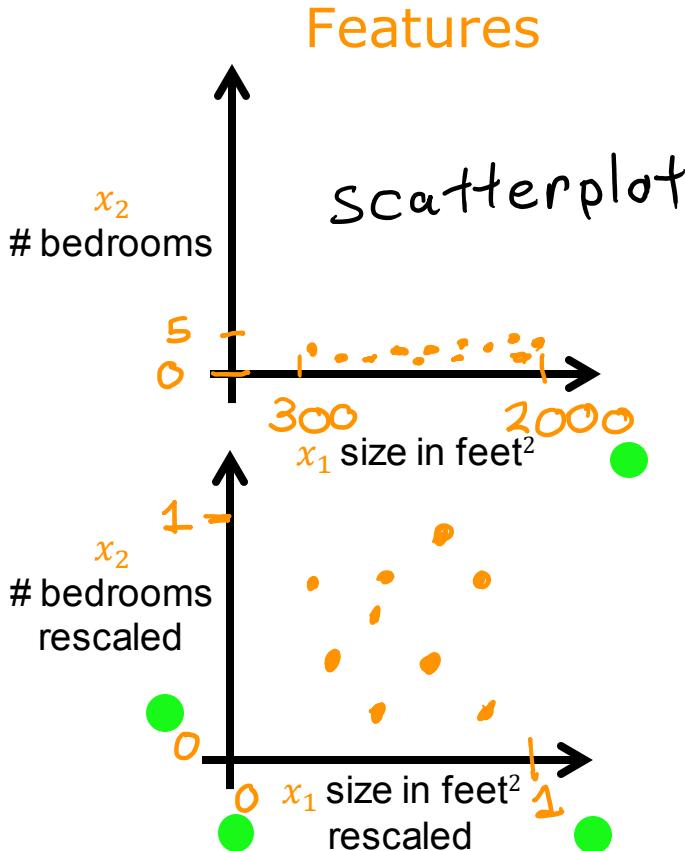
$$\widehat{\text{price}} = \$500k \text{ more reasonable}$$

# Feature size and parameter size

	size of feature $x_j$	size of parameter $w_j$
size in feet <sup>2</sup>	↔	↔
#bedrooms	↔	↔↔

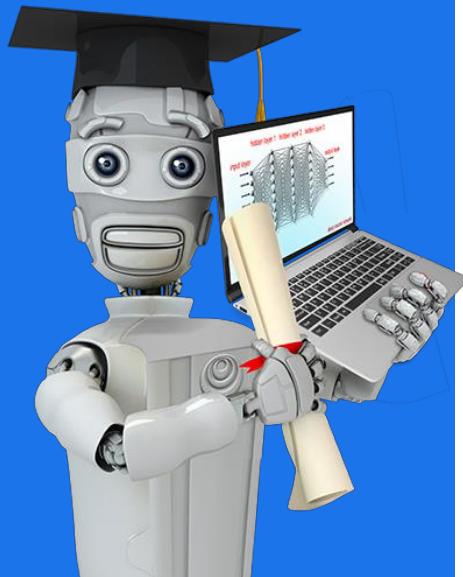


# Feature size and gradient descent



Stanford  
ONLINE

DeepLearning.AI

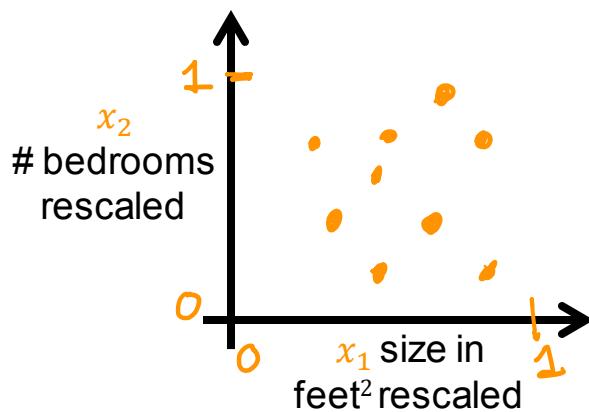
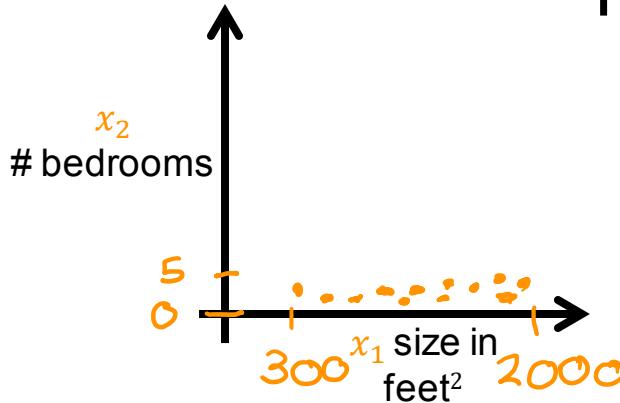


# Practical Tips for Linear Regression

---

## Feature Scaling Part 2

# Feature scaling



$$300 \leq x_1 \leq 2000$$

$$x_{1,scaled} = \frac{x_1}{2000}$$

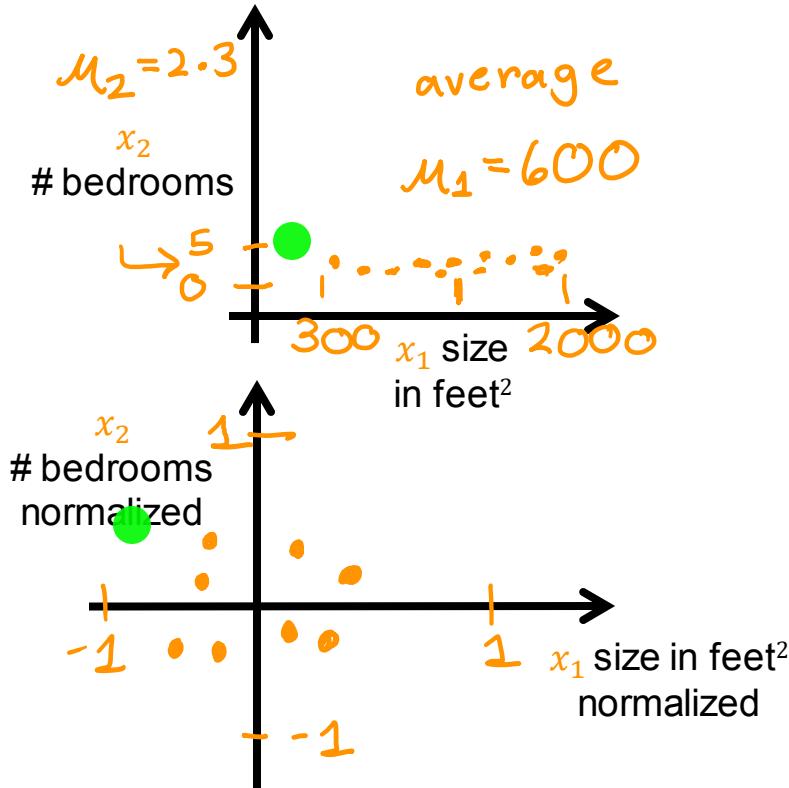
$$0.15 \leq x_{1,scaled} \leq 1$$

$$0 \leq x_2 \leq 5$$

$$x_{2,scaled} = \frac{x_2}{5}$$

$$0 \leq x_{2,scaled} \leq 1$$

# Mean normalization



$$300 \leq x_1 \leq 2000$$

$$x_1 = \frac{x_1 - \mu_1}{2000 - 300}$$

max-min

$$-0.18 \leq x_1 \leq 0.82$$

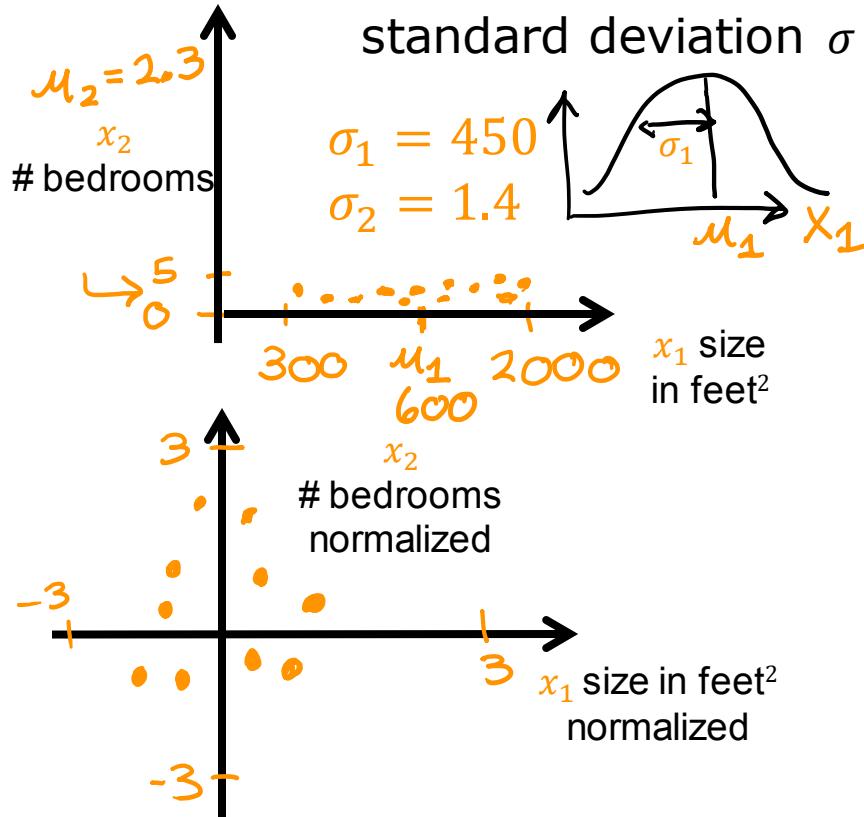
$$0 \leq x_2 \leq 5$$

$$x_2 = \frac{x_2 - \mu_2}{5 - 0}$$

max-min

$$-0.46 \leq x_2 \leq 0.54$$

# Z-score normalization



$$300 \leq x_1 \leq 2000$$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

$$-0.67 \leq x_1 \leq 3.1$$

$$0 \leq x_2 \leq 5$$

$$x_2 = \frac{x_2 - \mu_2}{\sigma_2}$$

$$-1.6 \leq x_2 \leq 1.9$$

# Feature scaling

aim for about  $-1 \leq x_j \leq 1$  for each feature  $x_j$   
 $-3 \leq x_j \leq 3$   
 $-0.3 \leq x_j \leq 0.3$

$\left. \begin{matrix} -3 \leq x_j \leq 3 \\ -0.3 \leq x_j \leq 0.3 \end{matrix} \right\}$  acceptable ranges

$$0 \leq x_1 \leq 3$$

Okay, no rescaling

$$-2 \leq x_2 \leq 0.5$$

Okay, no rescaling

$$-100 \leq x_3 \leq 100$$

too large  $\rightarrow$  rescale

$$-0.001 \leq x_4 \leq 0.001$$

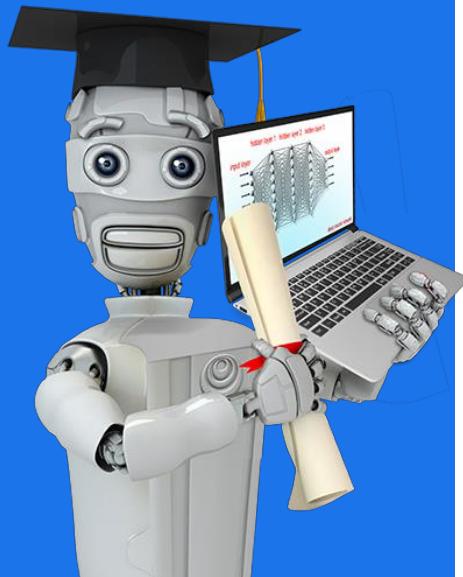
too small  $\rightarrow$  rescale

$$98.6 \leq x_5 \leq 105$$

too large  $\rightarrow$  rescale

Stanford  
ONLINE

DeepLearning.AI



# Practical Tips for Linear Regression

---

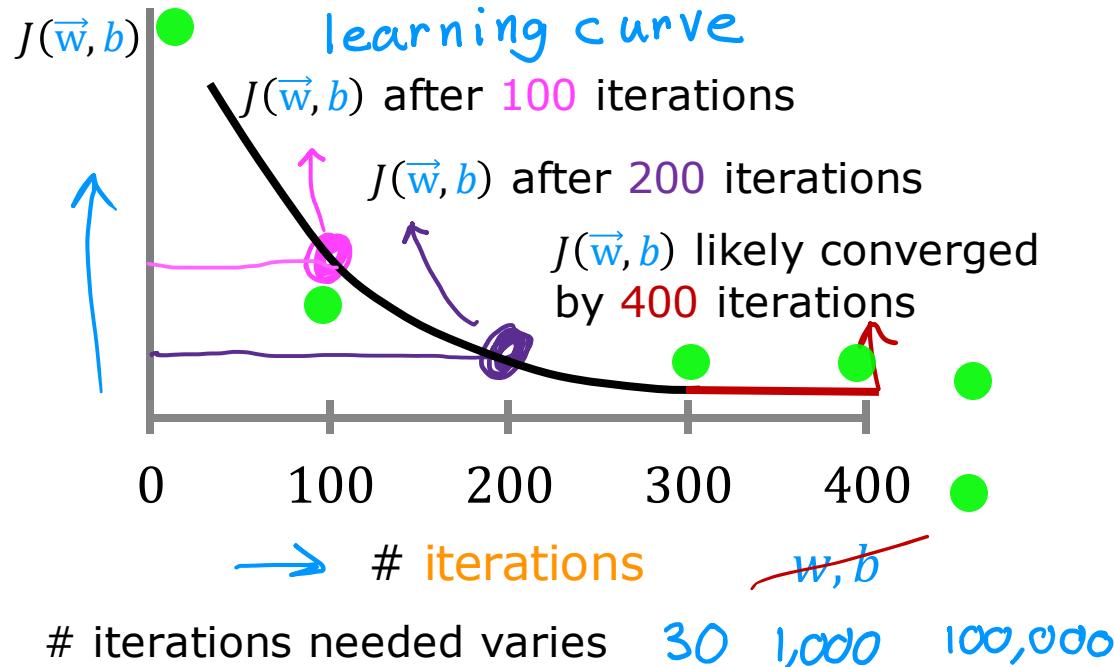
## Checking Gradient Descent for Convergence

# Gradient descent

$$\left\{ \begin{array}{l} w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b) \\ b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b) \end{array} \right.$$

# Make sure gradient descent is working correctly

objective:  $\min_{\vec{w}, b} J(\vec{w}, b)$   $J(\vec{w}, b)$  should **decrease** after every iteration



Automatic convergence test

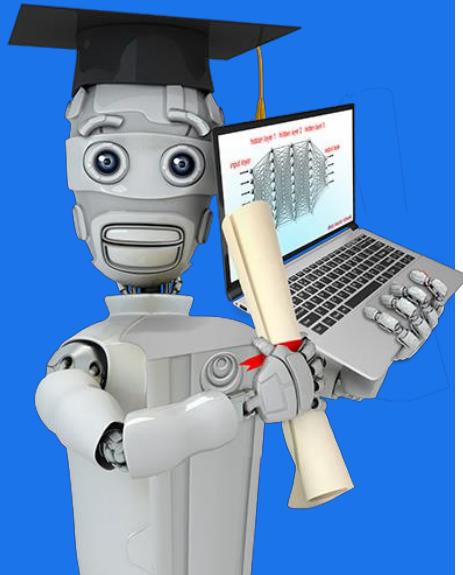
Let  $\varepsilon$  “epsilon” be  $10^{-3}$ .  
**0.001**

If  $J(\vec{w}, b)$  decreases by  $\leq \varepsilon$  in one iteration,  
declare convergence.

(found parameters  $\vec{w}, b$   
to get close to  
global minimum)

Stanford  
ONLINE

DeepLearning.AI

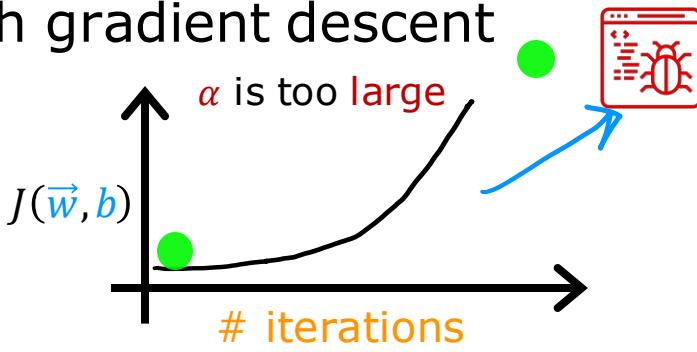
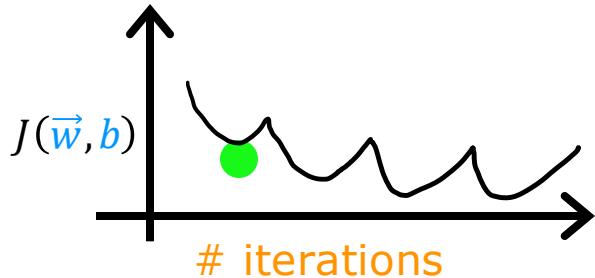


# Practical Tips for Linear Regression

---

## Choosing the Learning Rate

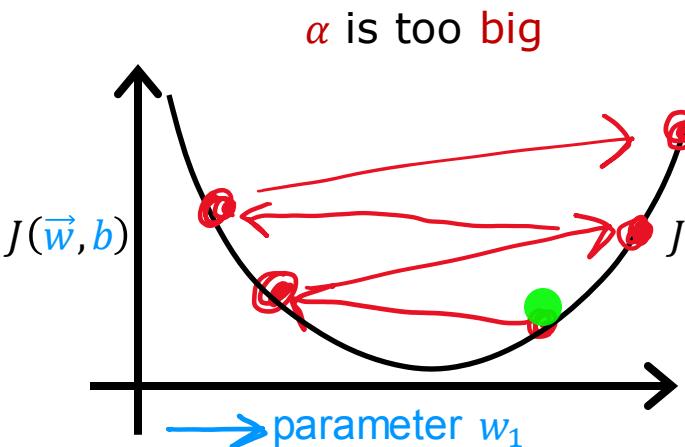
# Identify problem with gradient descent



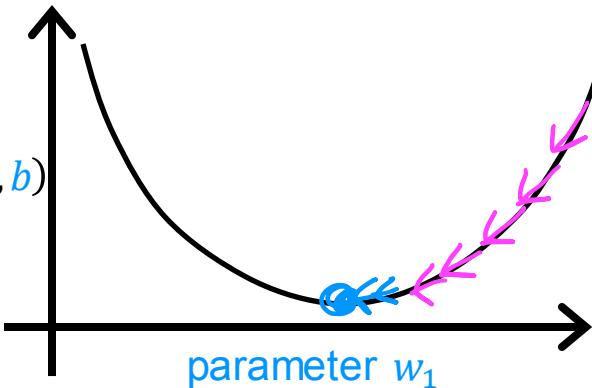
or learning rate is too large

$w_1 = w_1 + \alpha d_1$  ||  
use a minus sign  
 $w_1 = w_1 - \alpha d_1$  ||

## Adjust learning rate



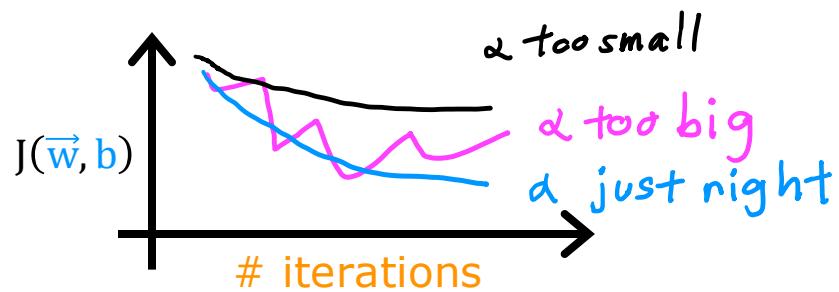
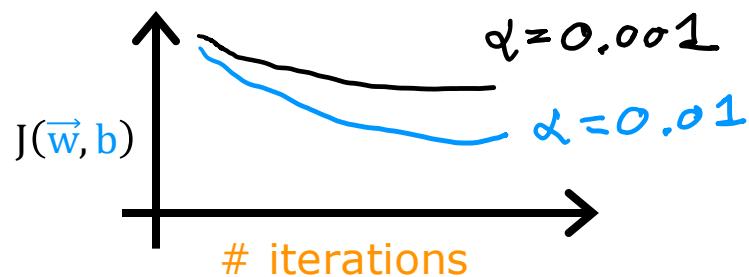
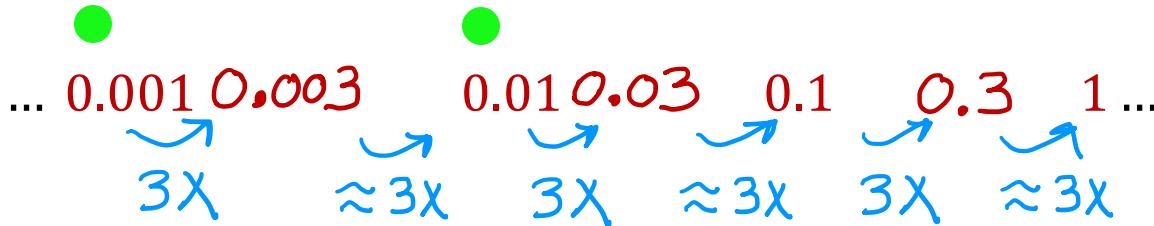
Use smaller  $\alpha$



With a small enough  $\alpha$ ,  $J(\vec{w}, b)$  should decrease on every iteration

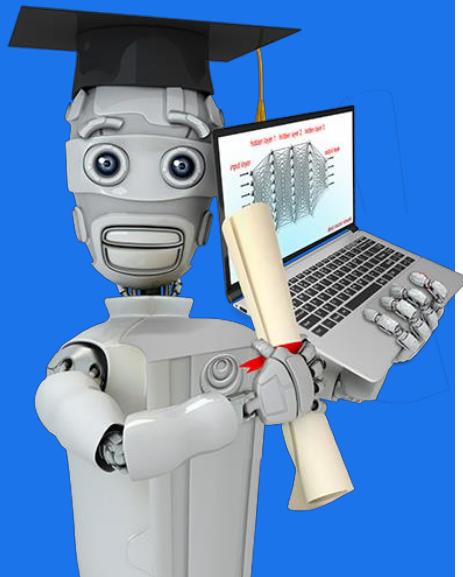
If  $\alpha$  is too small, gradient descent takes a lot more iterations to converge

Values of  $\alpha$  to try:



Stanford  
ONLINE

DeepLearning.AI



# Practical Tips for Linear Regression

---

## Feature Engineering

# Feature engineering

$$f_{\vec{w}, b}(\vec{x}) = w_1 \underline{x_1} + w_2 \underline{x_2} + b$$

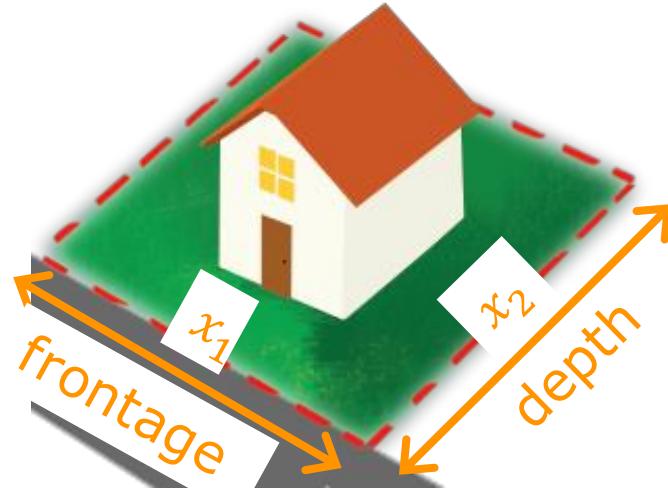
frontage      depth

$$\text{area} = \text{frontage} \times \text{depth}$$

$$x_3 = x_1 x_2$$

new feature

$$f_{\vec{w}, b}(\vec{x}) = \underline{w_1} \underline{x_1} + \underline{w_2} \underline{x_2} + \underline{w_3} \underline{x_3} + b$$



Feature engineering:  
Using **intuition** to design  
**new features**, by  
transforming or combining  
original features.

Stanford  
ONLINE

DeepLearning.AI

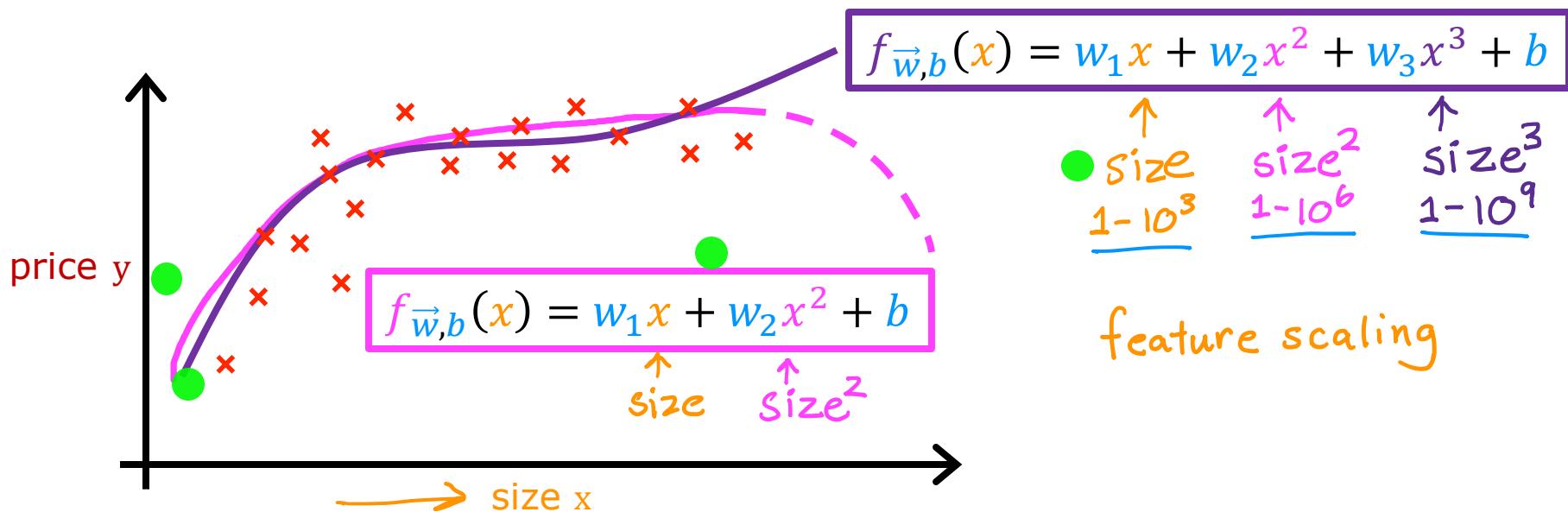


# Practical Tips for Linear Regression

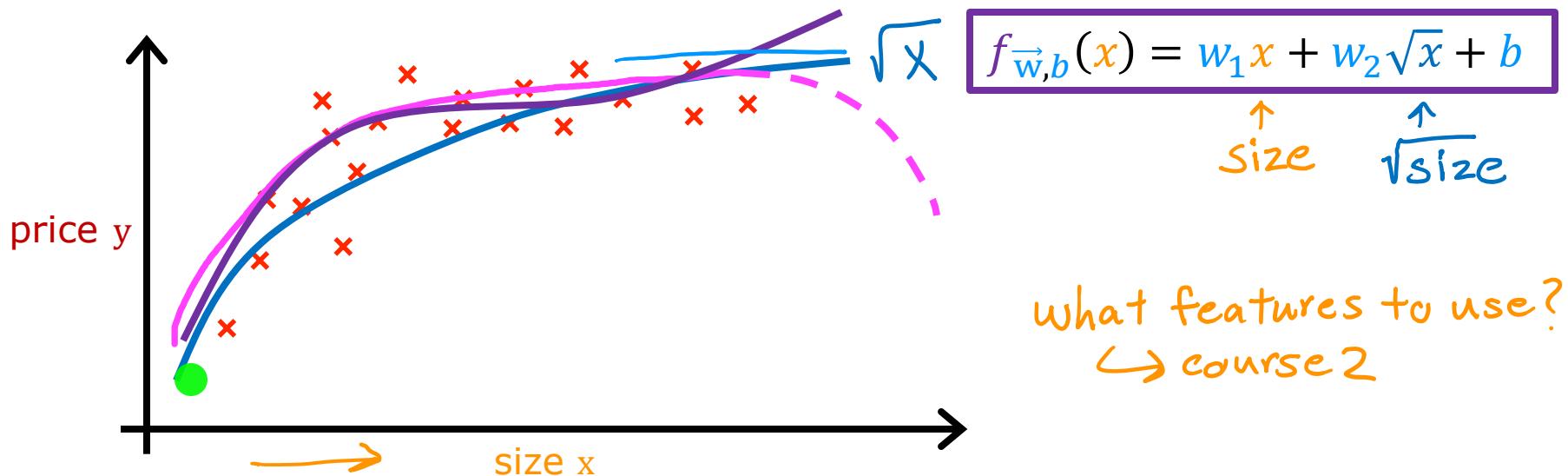
---

## Polynomial Regression

# Polynomial regression



# Choice of features



# Copyright Notice

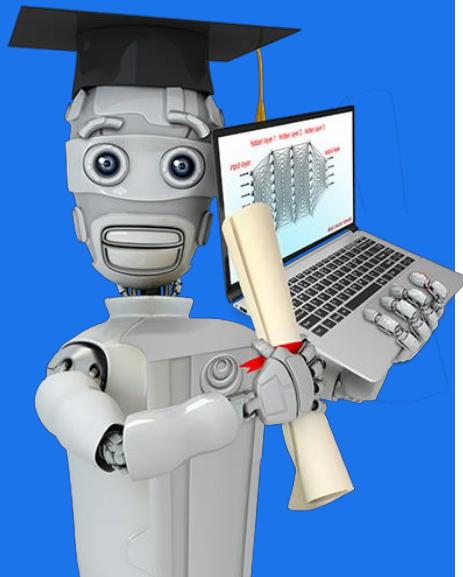
These slides are distributed under the Creative Commons License.

[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>

Stanford  
ONLINE

DeepLearning.AI



# Classification

---

## Motivations

# Classification

Question

Is this email spam?

Is the transaction fraudulent?

Is the tumor malignant?

Answer " $y$ "

no	yes
no	yes
no	yes

$y$  can only be one of **two** values

"binary classification"

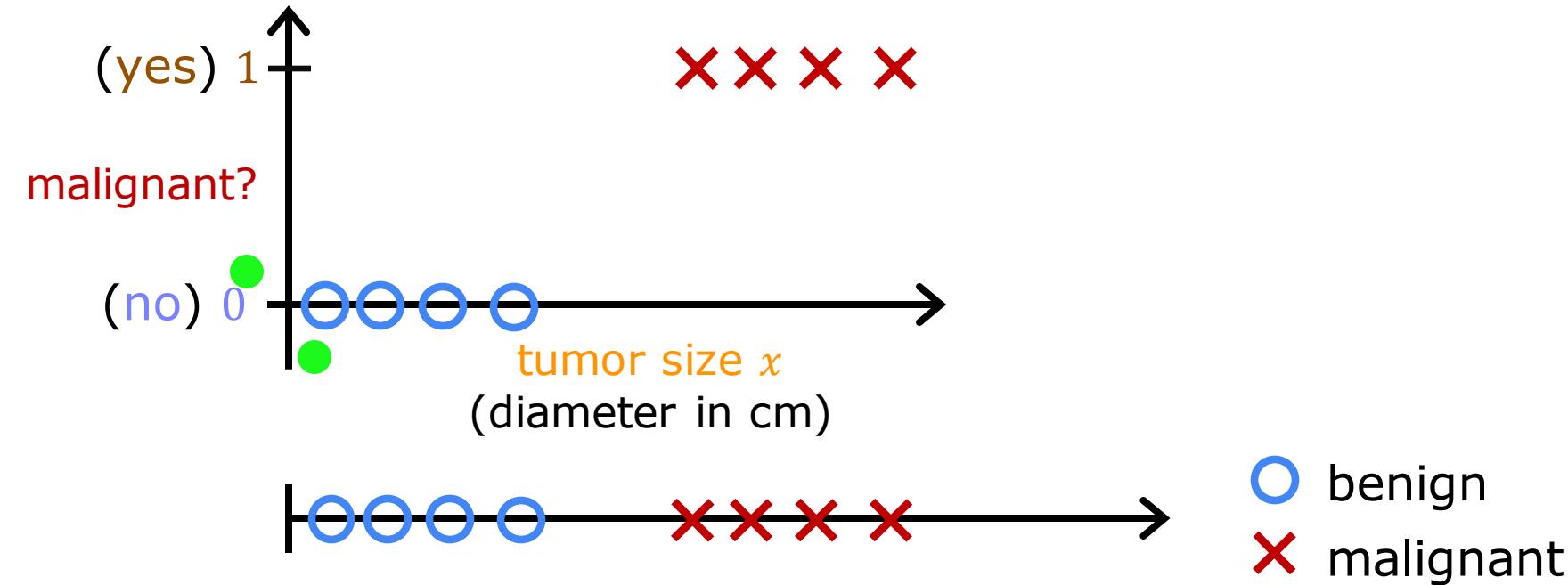
class = category

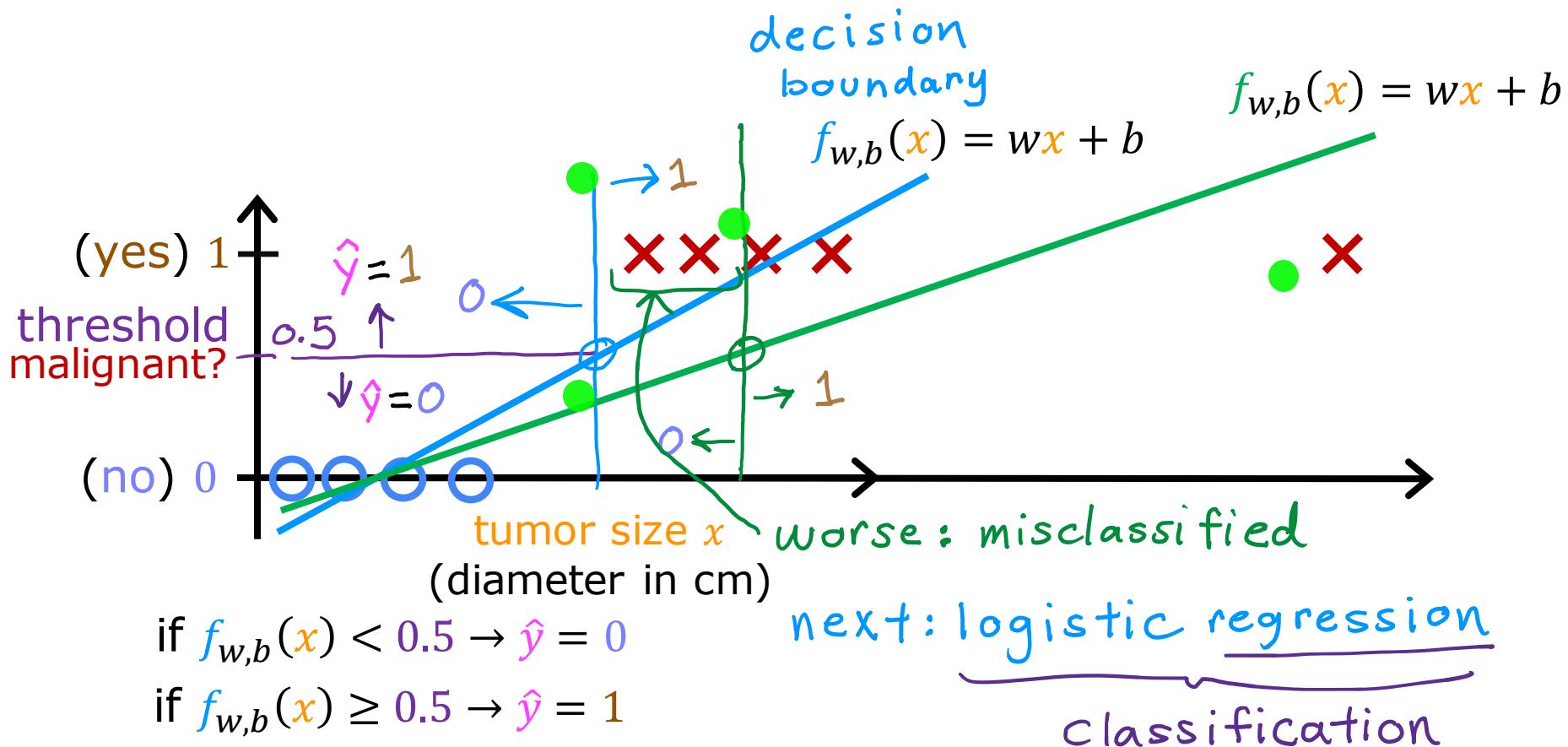
"negative class"  
 $\neq$  "bad"  
absence

false      true  
0            1

useful for  
classification

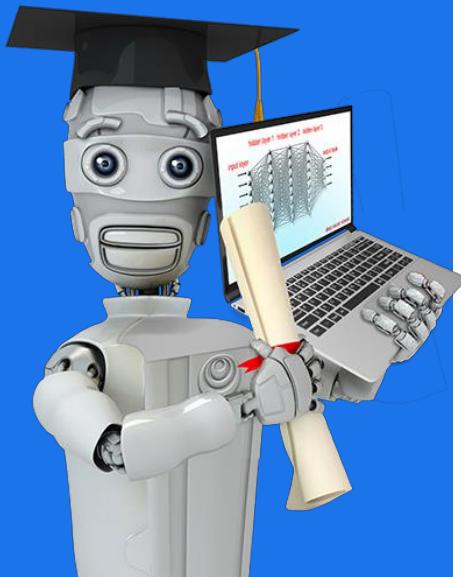
"positive class"  
 $\neq$  "good"  
presence





Stanford  
ONLINE

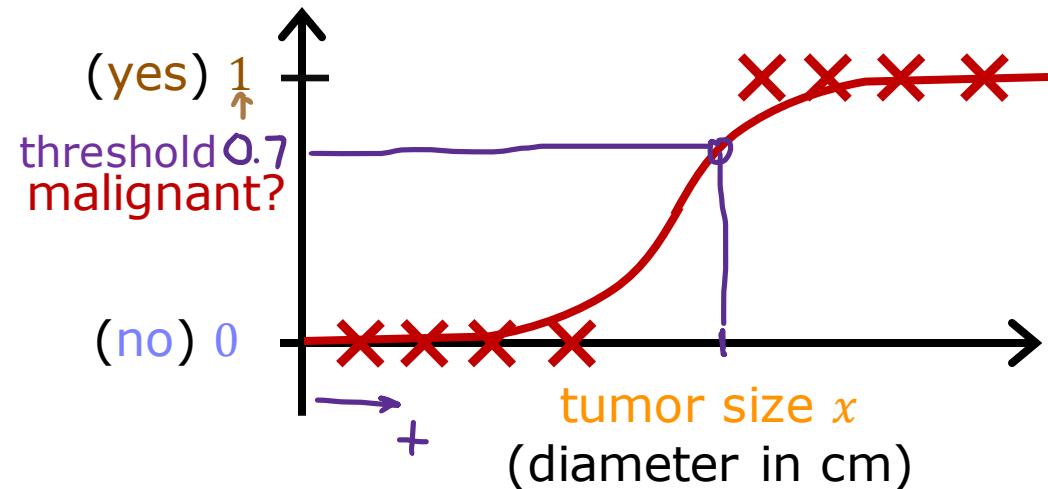
DeepLearning.AI



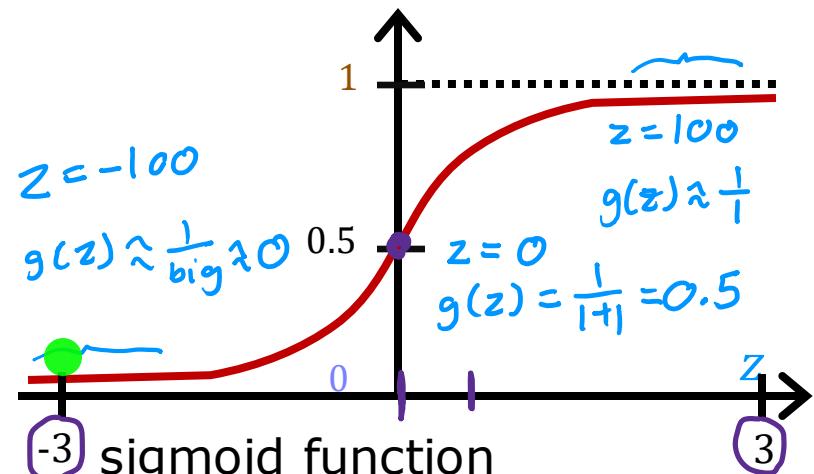
# Classification

---

## Logistic Regression



Want outputs between 0 and 1

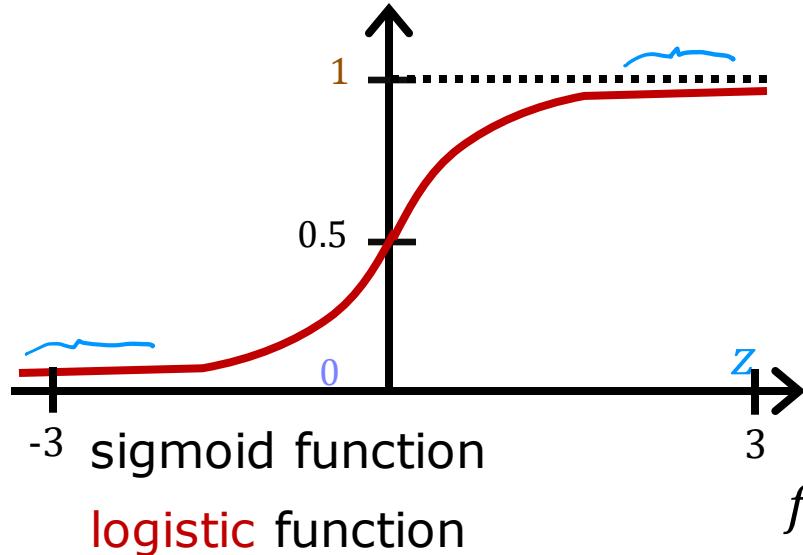


logistic function

outputs between 0 and 1

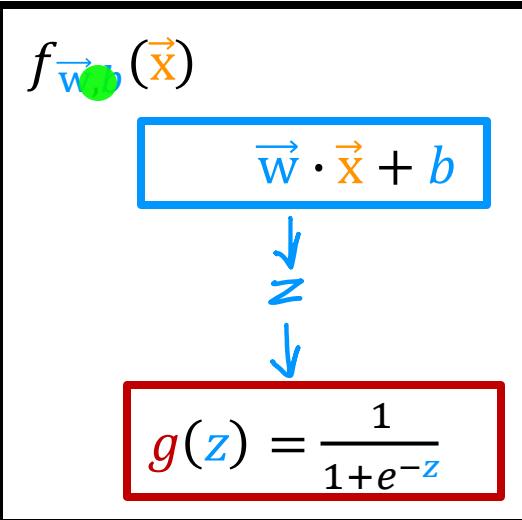
$$g(z) = \frac{1}{1+e^{-z}} \quad 0 < g(z) < 1$$

Want outputs between 0 and 1



outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \quad 0 < g(z) < 1$$



$$f_{\vec{w}, b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_z) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

"logistic regression"

$$e \approx 2.7$$

# Interpretation of logistic regression output

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

“probability” that class is 1

$$f_{\vec{w}, b}(\vec{x}) = P(y = 1 | \vec{x}; \vec{w}, b)$$

Probability that  $y$  is 1,  
given input  $\vec{x}$ , parameters  $\vec{w}, b$

Example:

$x$  is “tumor size”

$y$  is 0 (not malignant)  
or 1 (malignant)

$$P(y = 0) + P(y = 1) = 1$$

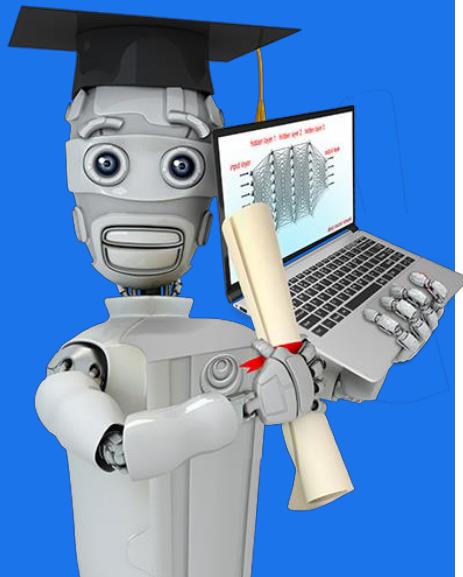
$$f_{\vec{w}, b}(\vec{x}) = 0.7$$

70% chance that  $y$  is 1



Stanford  
ONLINE

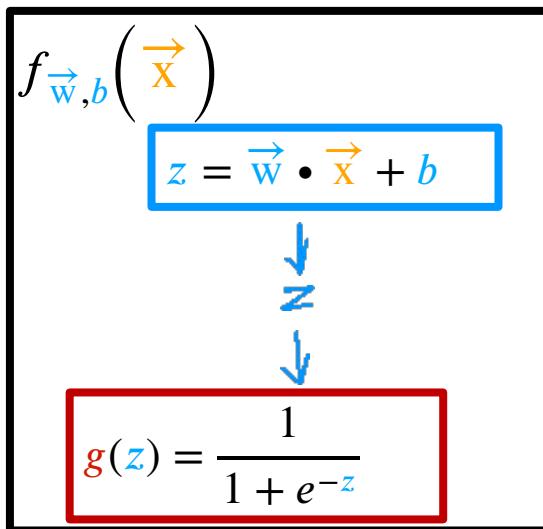
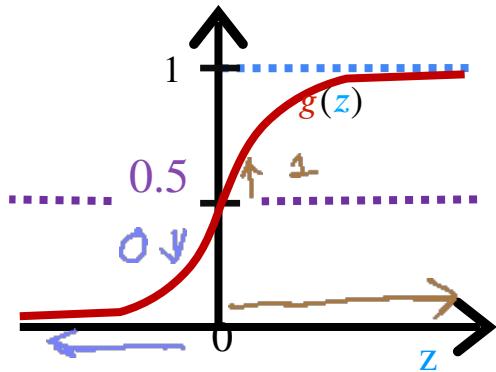
DeepLearning.AI



# Classification

---

## Decision Boundary



$$f_{\vec{w}, b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x}}_z + \bar{b}) = P(y=1 | x; \vec{w}, b)$$

0 or 1? threshold

Is  $f_{\vec{w}, b}(\vec{x}) \geq \underline{0.5}$ ?

Yes:  $\hat{y} = 1$

No:  $\hat{y} = 0$

When is

$$f_{\vec{w}, b}(\vec{x}) \geq 0 \quad g(z) \geq 0.5$$

$$z \geq 0$$

$$z < 0$$

$$\vec{w} \cdot \vec{x} + b \geq 0$$

$$\vec{w} \cdot \vec{x} + b < 0$$

$$\hat{y} = 1$$

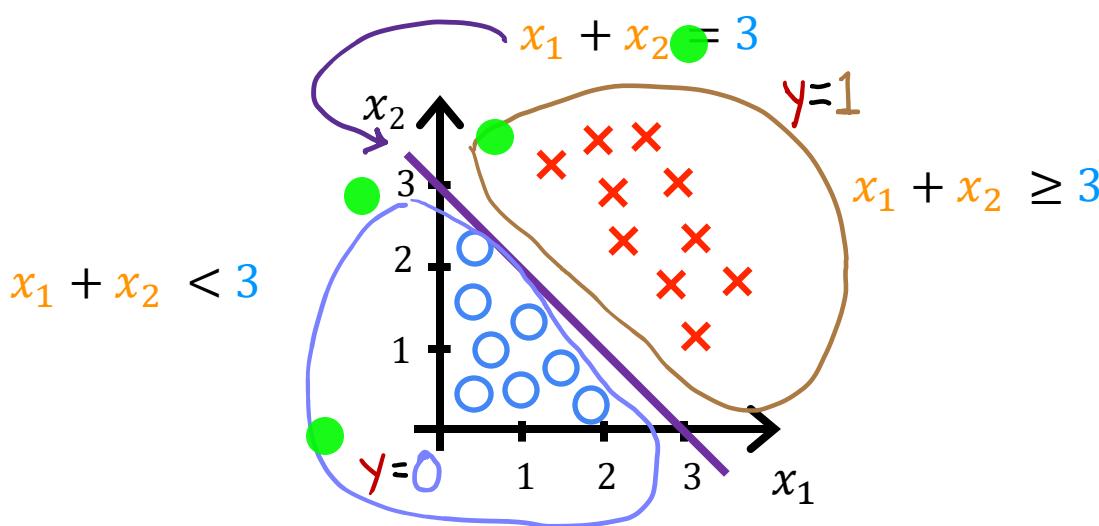
$$\hat{y} = 0$$

# Decision boundary

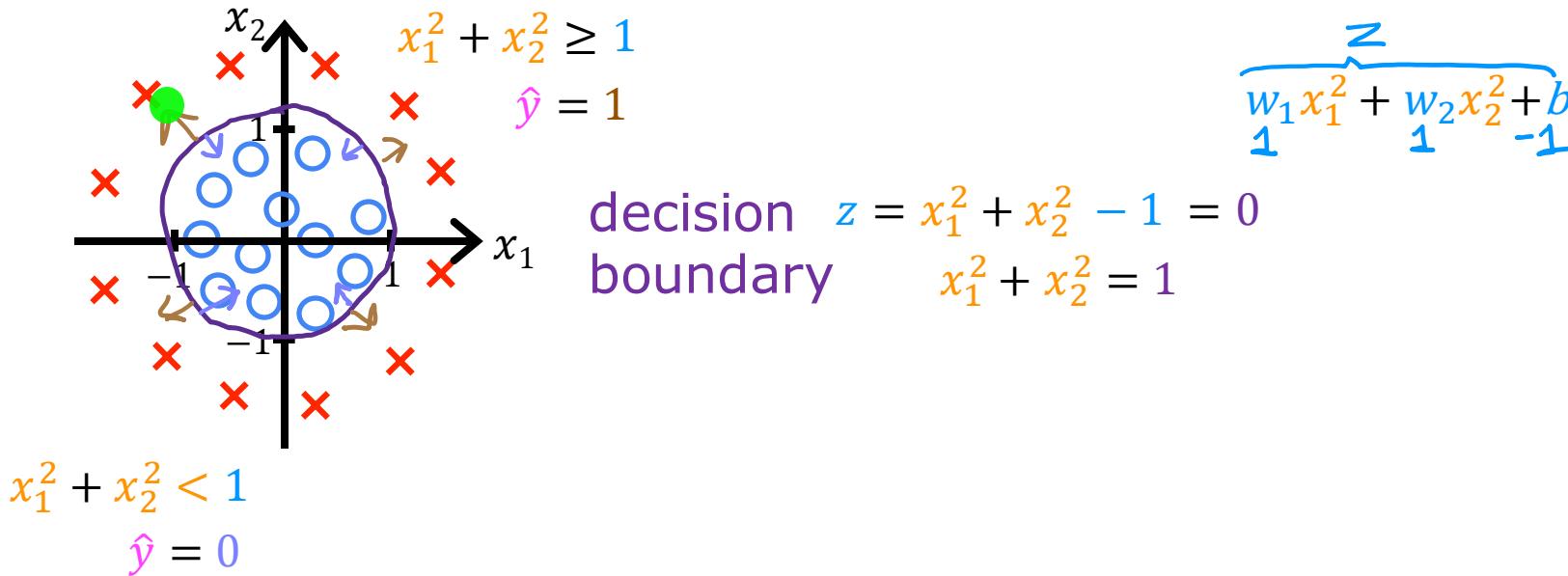
$$f_{\vec{w}, b}(\vec{x}) = g(z) = g(\underbrace{w_1 x_1 + w_2 x_2}_{1} + \underbrace{b}_{-3})$$

Decision boundary  $z = \vec{w} \cdot \vec{x} + b = 0$

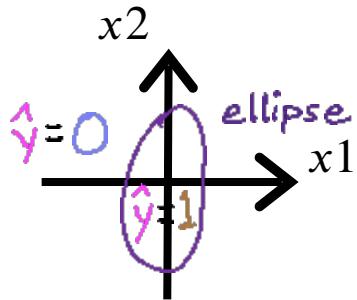
$$z = x_1 + x_2 - 3 = 0$$



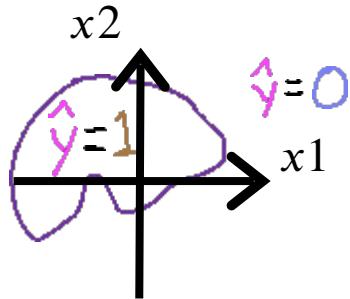
# Non-linear decision boundaries



# Non-linear decision boundaries



$$f_{\vec{w}, b}(\vec{x}) = g(z) = g(w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2 + w_6 x_1^3 + \dots + b)$$



Stanford  
ONLINE

DeepLearning.AI



# Cost Function

---

## Cost Function for Logistic Regression

# • Training set

	tumor size (cm) $x_1$	...	patient's age $x_n$	malignant? $y$	$i = 1, \dots, m$ ↪ training examples
$i=1$	10		52	1	target $y$ is 0 or 1
:	2		73	0	
:	5		55	0	
$i=m$	12		49	1	
	...		...	...	

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

How to choose  $\vec{w} = [w_1 \ w_2 \ \cdots \ w_n]$  and  $b$ ?

# Squared error cost

*cost*

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

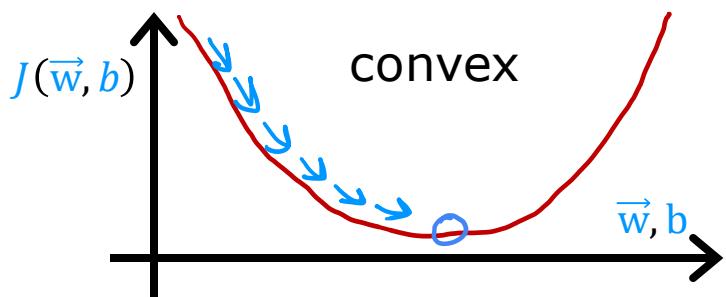
average of training set

*loss*

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})$$

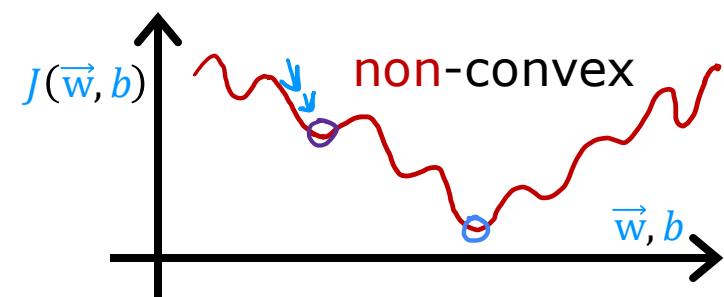
linear regression

$$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



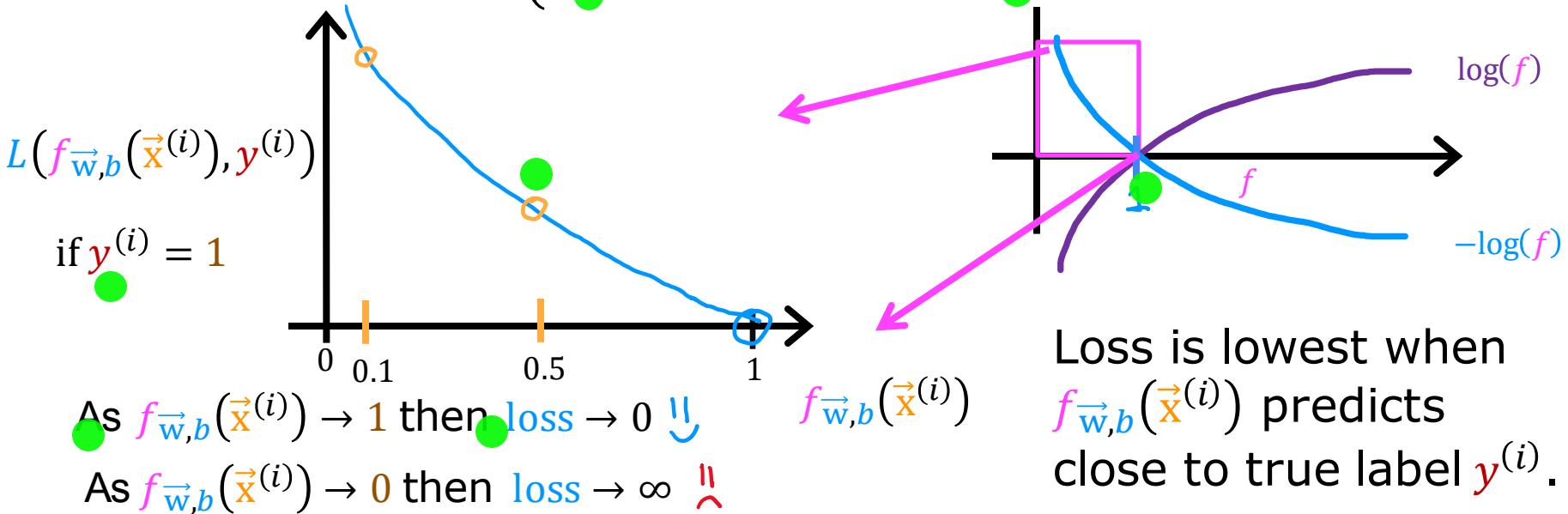
logistic regression

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$



# Logistic loss function

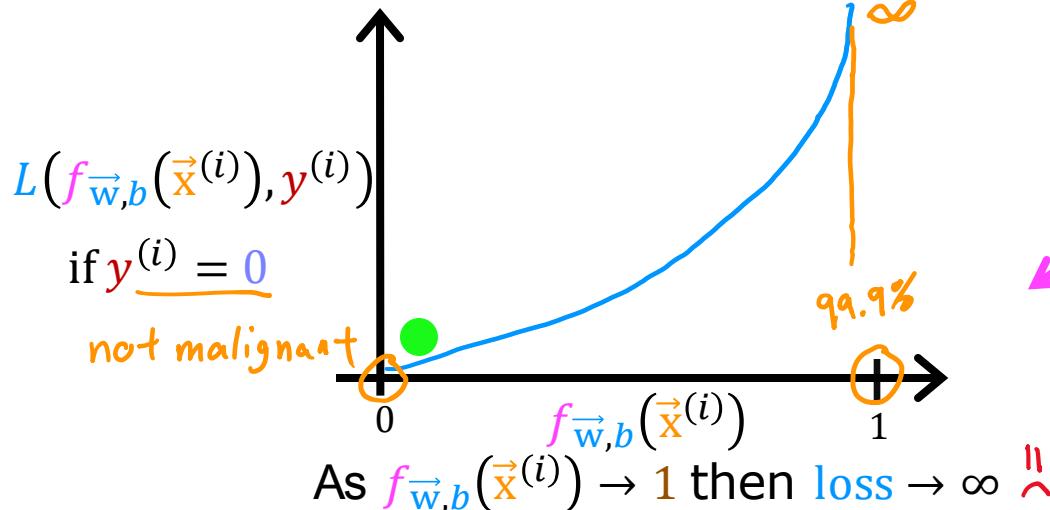
$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$



# Logistic loss function

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

As  $f_{\vec{w}, b}(\vec{x}^{(i)}) \rightarrow 0$  then  $\text{loss} \rightarrow 0$  



The further prediction  $f_{\vec{w}, b}(\vec{x}^{(i)})$  is from target  $y^{(i)}$ , the higher the loss.

# Cost

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m \underbrace{L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})}_{\text{loss}}$$

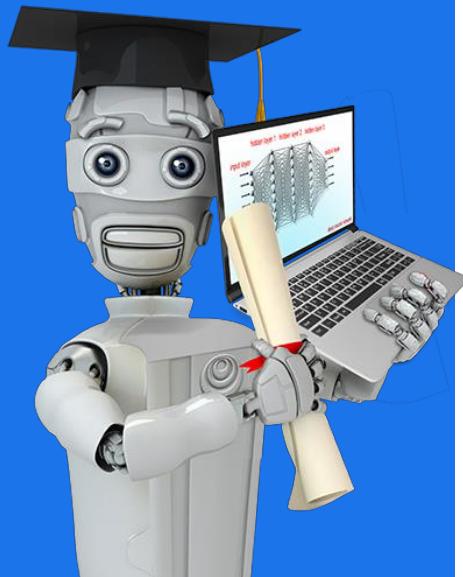
$$= \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

*Convex*  
*can reach a global minimum*

find  $w, b$  that minimize cost  $J$

Stanford  
ONLINE

DeepLearning.AI



# Cost Function

---

Simplified Cost  
Function for Logistic  
Regression

# Simplified loss function

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))$$

if  $y^{(i)} = 1$ :

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \underbrace{-1 \log(f(x))}_{\Theta}$$

# Simplified loss function

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))$$

if  $y^{(i)} = 1$ :

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -1 \log(f(\vec{x}))$$

if  $y^{(i)} = 0$ :

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = - (1 - 0) \log(1 - f(\vec{x}))$$

# Simplified cost function

$$\text{loss} \quad L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))$$

$$\text{cost} \quad J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m [L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})]$$

$$= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

↑ convex  
(single global minimum)

maximum likelihood  
(don't worry about it!)

Stanford  
ONLINE

DeepLearning.AI



# Gradient Descent

---

## Gradient Descent Implementation

# Training logistic regression

Find  $\vec{w}, b$

Given new  $\vec{x}$ , output  $f_{\vec{w}, b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$

$$P(y=1|\vec{x}; \vec{w}, b)$$

# Gradient descent

*cost*

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \right]$$

repeat {  
   $j = 1 \dots n$   
   $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$   
   $b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$   
}  
} simultaneous updates

$$\frac{\partial}{\partial w_j} J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \underline{x_j^{(i)}}$$
$$\frac{\partial}{\partial b} J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

# Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

} simultaneous updates

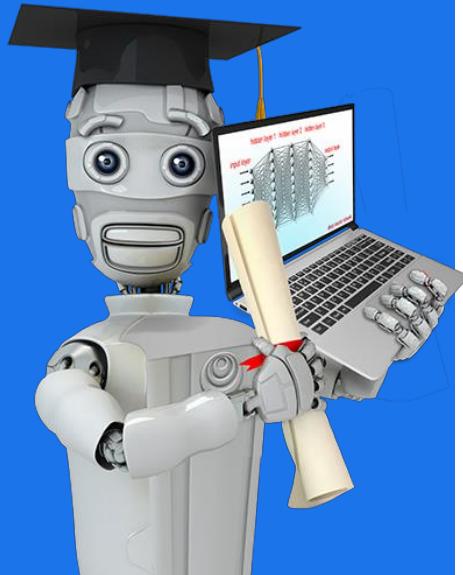
- Same concepts:
- Monitor gradient descent (learning curve)
  - Vectorized implementation
  - Feature scaling

Linear regression       $f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Logistic regression       $f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{(-\vec{w} \cdot \vec{x} + b)}}$

Stanford  
ONLINE

DeepLearning.AI

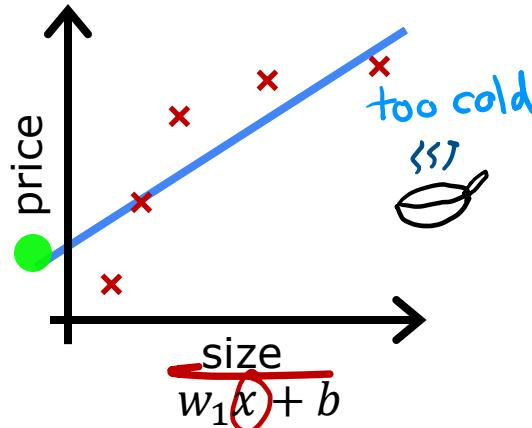


# Regularization to Reduce Overfitting

---

## The Problem of Overfitting

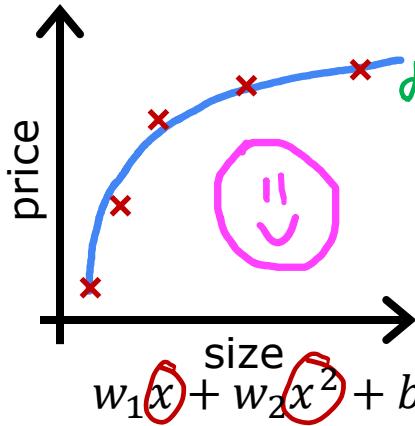
# Regression example



underfit

- Does not fit the training set well

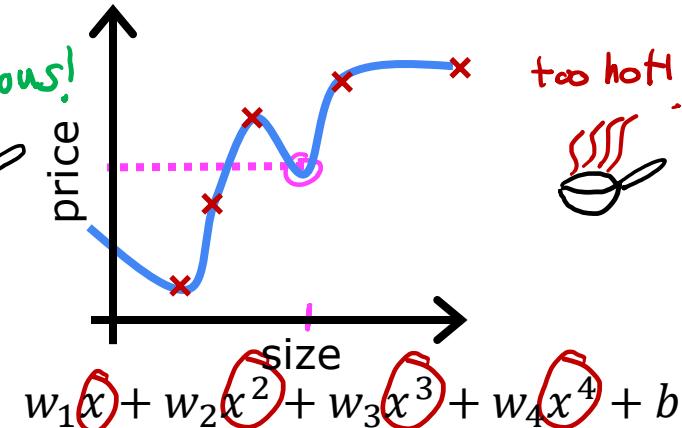
high bias



just right

- Fits training set pretty well

generalization

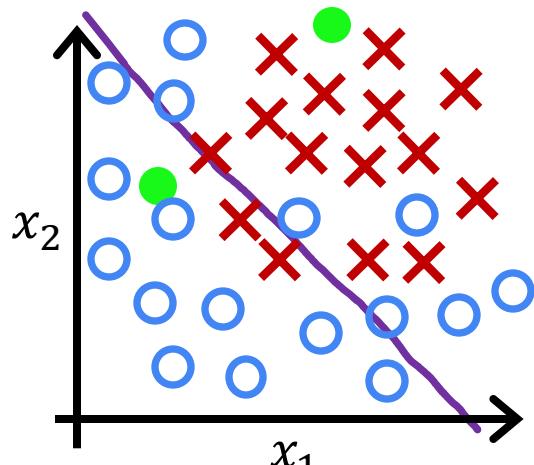


overfit

- Fits the training set extremely well

high variance

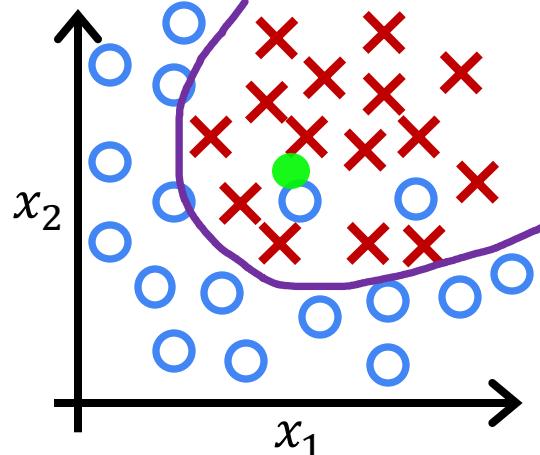
# Classification



$$z = w_1 x_1 + w_2 x_2 + b$$

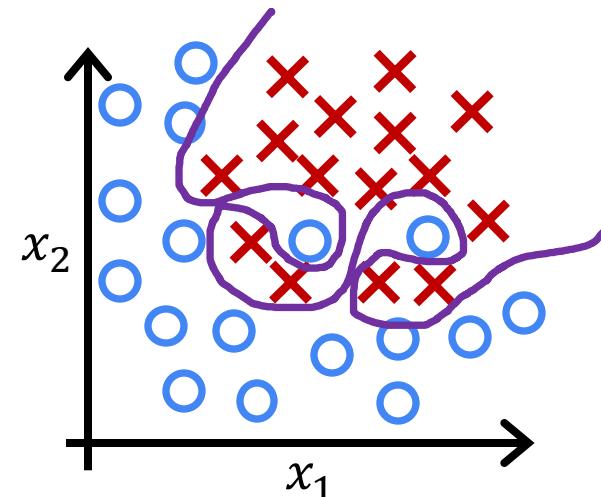
$$f_{\vec{w}, b}(\vec{x}) = g(z)$$

$g$  is the sigmoid function  
underfit      high bias



$$\begin{aligned} z = & w_1 x_1 + w_2 x_2 \\ & + w_3 x_1^2 + w_4 x_2^2 \\ & + w_5 x_1 x_2 + b \end{aligned}$$

just right

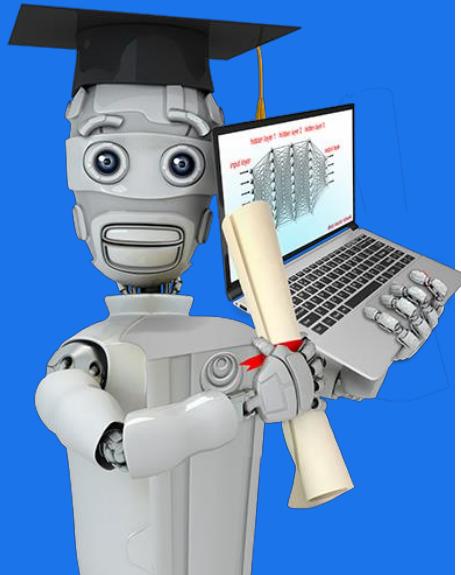


$$\begin{aligned} z = & w_1 x_1 + w_2 x_2 \\ & + w_3 x_1^2 + w_4 x_2^2 \\ & + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 \\ & + \dots + b \end{aligned}$$

Overfit

Stanford  
ONLINE

DeepLearning.AI

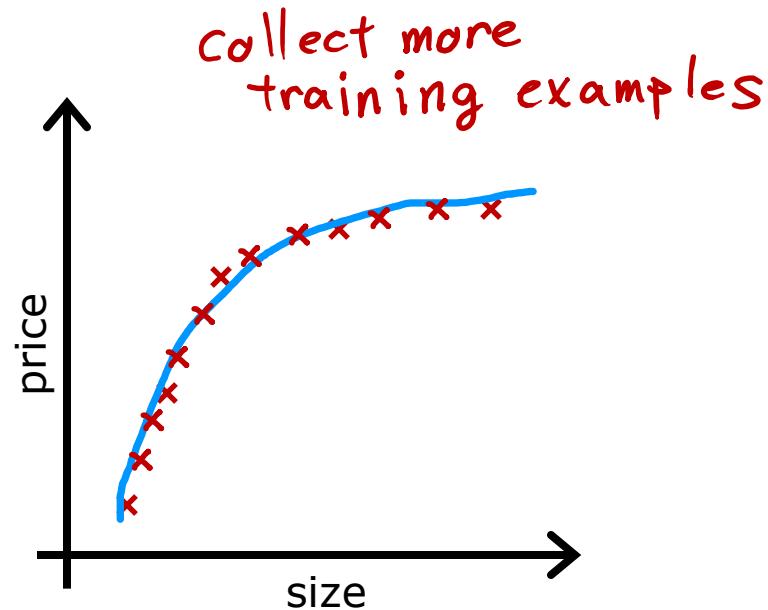
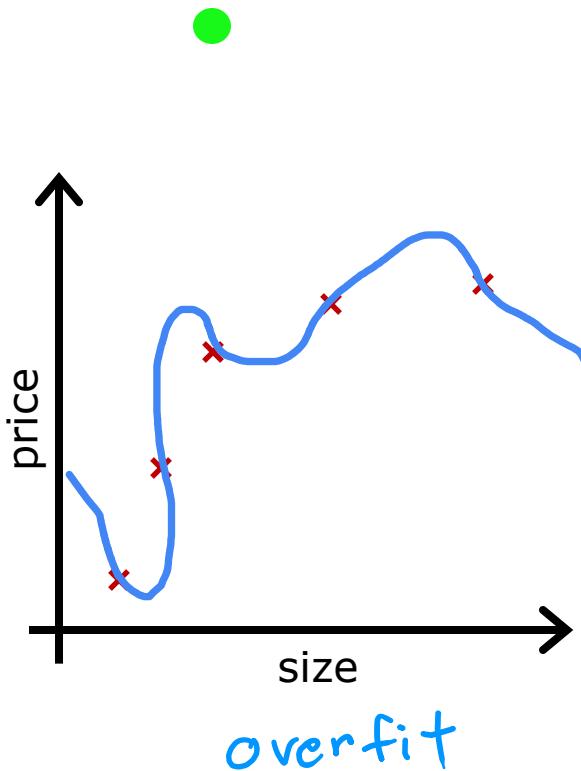


# Regularization to Reduce Overfitting

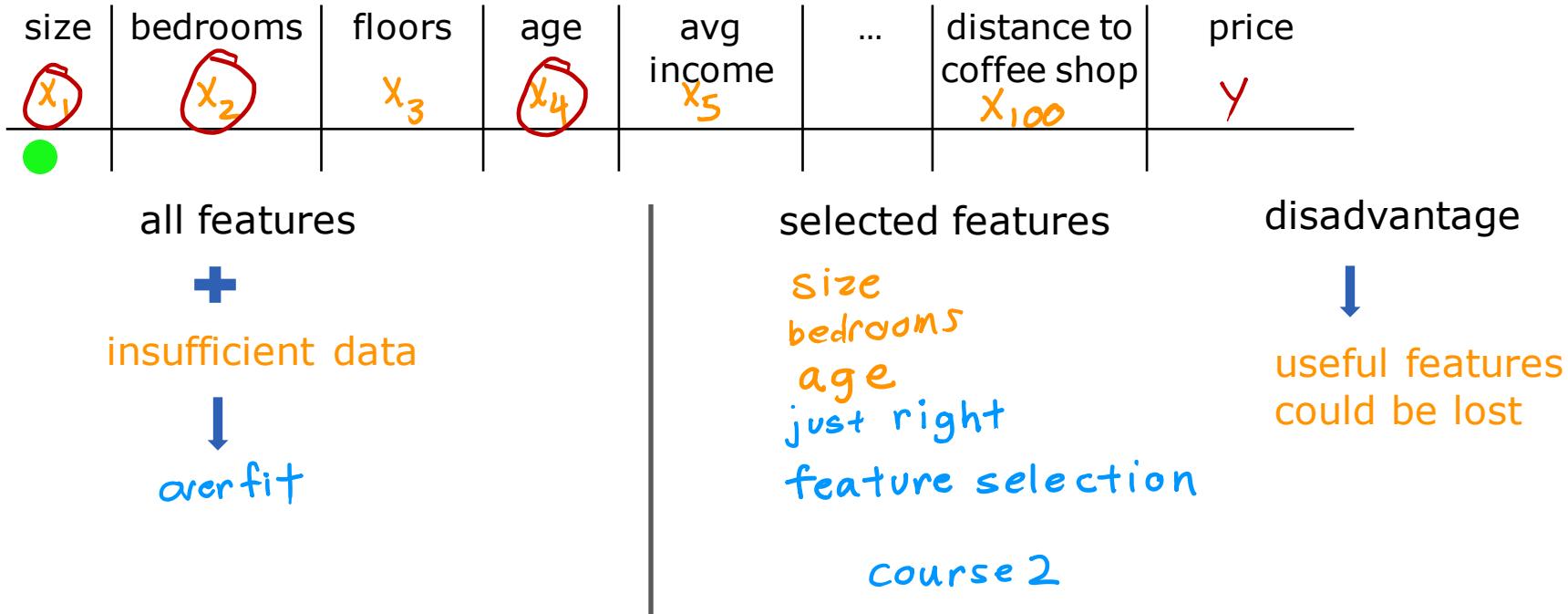
---

## Addressing Overfitting

# Collect more training examples

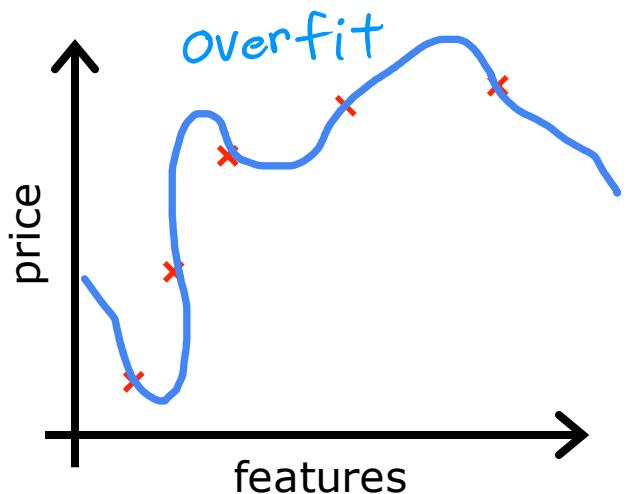


# Select features to include/exclude



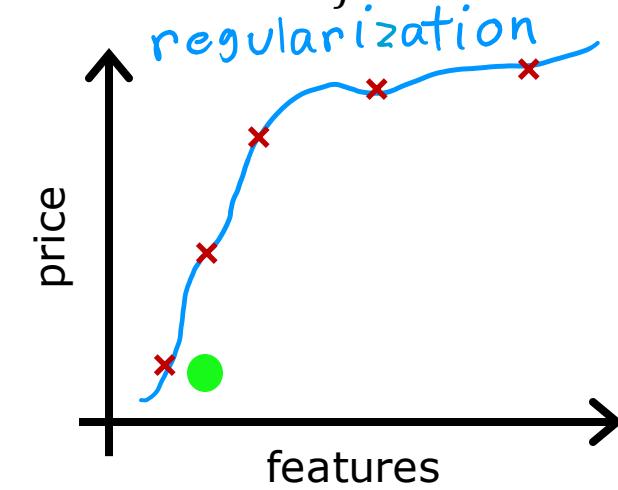
# Regularization

Reduce the size of parameters  $w_j$



$$f(x) = 28x - 385x^2 + 39x^3 - \cancel{174x^4} + 100$$

large values for  $w_j$       eliminate feature



$$f(x) = 13x - 0.23x^2 + 0.000014x^3 - \cancel{0.0001x^4} + 10$$

small values for  $w_j$

# Addressing overfitting

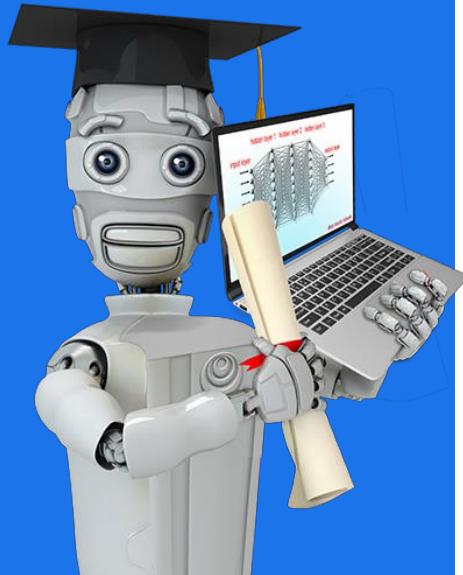
## Options

1. Collect more data
  2. Select features
    - Feature selection *in course 2*
  3. Reduce size of parameters
    - “Regularization” *next videos!*
- 



Stanford  
ONLINE

DeepLearning.AI

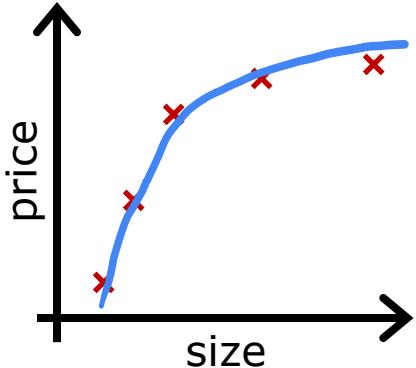


# Regularization to Reduce Overfitting

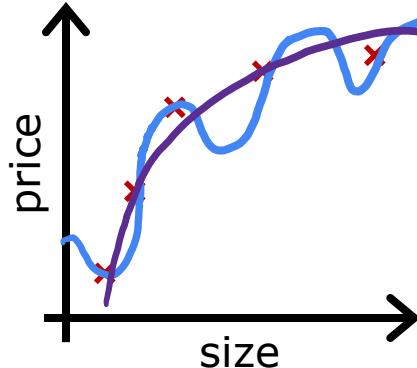
---

## Cost Function with Regularization

# Intuition



$$w_1x + w_2x^2 + b$$



$$w_1x + w_2x^2 + \cancel{w_3x^3} + \cancel{w_4x^4} + b$$

$\approx 0$        $\approx 0$

make  $w_3, w_4$  really small ( $\approx 0$ )

$$\min_{\vec{w}, b} \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + 1000 \underbrace{w_3^2}_{0.001} + 1000 \underbrace{w_4^2}_{0.002}$$

# Regularization

small values  $w_1, w_2, \dots, w_n, b$

simpler model

$$w_3 \approx 0$$

less likely to overfit

$$w_4 \approx 0$$

size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	...	$x_{100}$	$y$

$n$  features       $n = 100$

$w_1, w_2, \dots, w_{100}, b$

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \underbrace{\frac{\lambda}{2m} \sum_{j=1}^n w_j^2}_{\text{"lambda"} \quad \text{regularization parameter}} + \underbrace{\frac{\lambda}{2m} b^2}_{\lambda > 0}$$

can include or exclude  $b$

# Regularization

$$\min_{\vec{w}, b} J(\vec{w}, b) = \min_{\vec{w}, b} \left[ \underbrace{\frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2}_{\text{mean squared error}} + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right]$$

fit data

regularization term

Keep  $w_j$  small

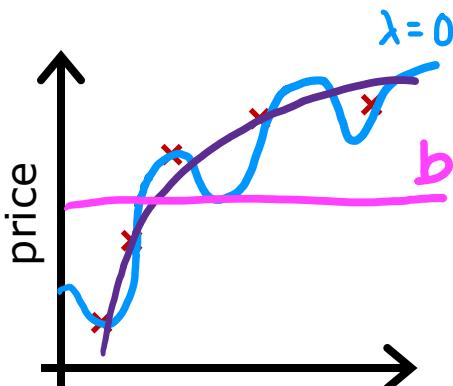
$\lambda$  balances both goals

choose  $\lambda = 10^{10}$

$$f_{\vec{w}, b}(\vec{x}) = \underbrace{w_1 x}_\approx + \underbrace{w_2 x^2}_\approx + \underbrace{w_3 x^3}_\approx + \underbrace{w_4 x^4}_\approx + b$$

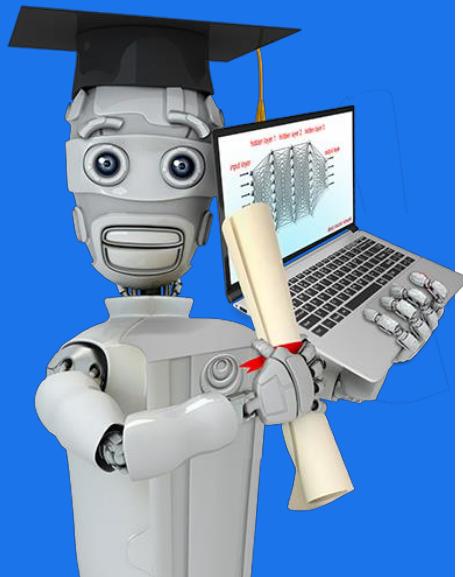
$$f(x) = b$$

Choose  $\lambda$



Stanford  
ONLINE

DeepLearning.AI



# Regularization to Reduce Overfitting

---

## Regularized Linear Regression

# Regularized linear regression

$$\min_{\vec{w}, b} J(\vec{w}, b) = \min_{\vec{w}, b} \left[ \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$j = 1, \dots, n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

} simultaneous update

$$\begin{aligned} &= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j \\ &= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \end{aligned}$$

don't have to regularize  $b$

# Implementing gradient descent

repeat {

- $w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m \left[ (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update  $j = 1, \dots, n$



# Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m \left[ (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update  $j = 1, \dots, n$

$$w_j = \underbrace{w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \left( 1 - \alpha \frac{\lambda}{m} \right)} - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}}_{\text{usual update}}$$

shrink  $w_j$

$$\alpha \frac{\lambda}{m}$$
$$0.01 \frac{1}{50} = 0.0002$$
$$w_j \left( 1 - 0.0002 \right)$$
$$0.9998$$

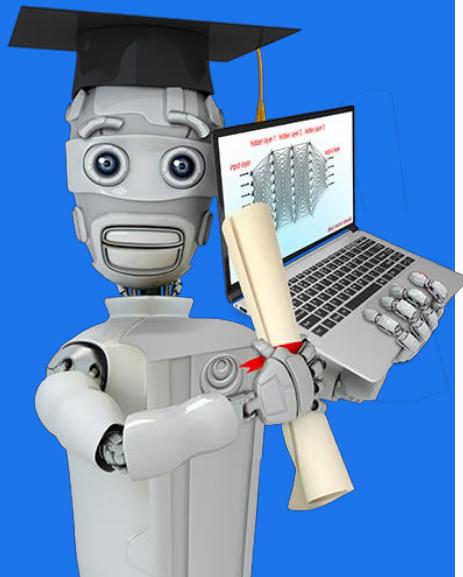


# How we get the derivative term (optional)

$$\begin{aligned} \frac{\partial}{\partial w_j} J(\vec{w}, b) &= \frac{\partial}{\partial w_j} \left[ \frac{1}{2m} \sum_{i=1}^m \underbrace{(\vec{w} \cdot \vec{x}^{(i)}) + b - y^{(i)}}_{f(\vec{x})}^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right] \\ &= \frac{1}{2m} \sum_{i=1}^m \left[ (\vec{w} \cdot \vec{x}^{(i)} + b - y^{(i)}) \cancel{\times} x_j^{(i)} \right] + \frac{\lambda}{2m} \cancel{\times} w_j \quad \text{No } \sum_{j=1}^n \\ &= \frac{1}{m} \sum_{i=1}^m \left[ \underbrace{(\vec{w} \cdot \vec{x}^{(i)} + b - y^{(i)})}_{f(\vec{x})} x_j^{(i)} \right] + \frac{\lambda}{m} w_j \\ &= \frac{1}{m} \sum_{i=1}^m \left[ (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \end{aligned}$$

Stanford  
ONLINE

DeepLearning.AI

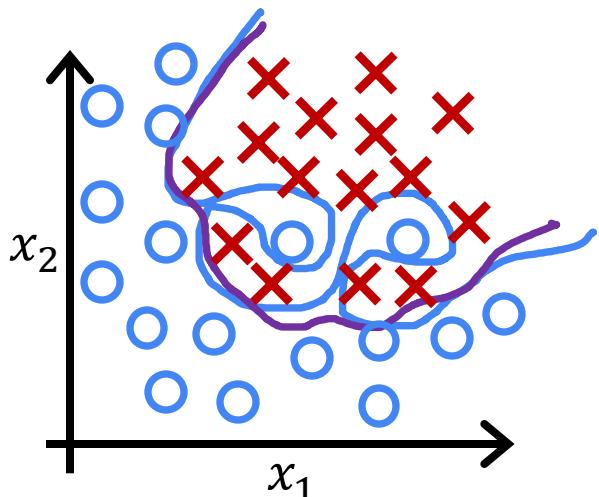


# Regularization to Reduce Overfitting

---

## Regularized Logistic Regression

# Regularized logistic regression



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + \dots + b$$
$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-z}}$$

## Cost function

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

$\min_{\vec{w}, b} J(\vec{w}, b) \rightarrow w_j \downarrow$

# Regularized logistic regression

$$\min_{\vec{w}, b} J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

j = 1...n

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

}

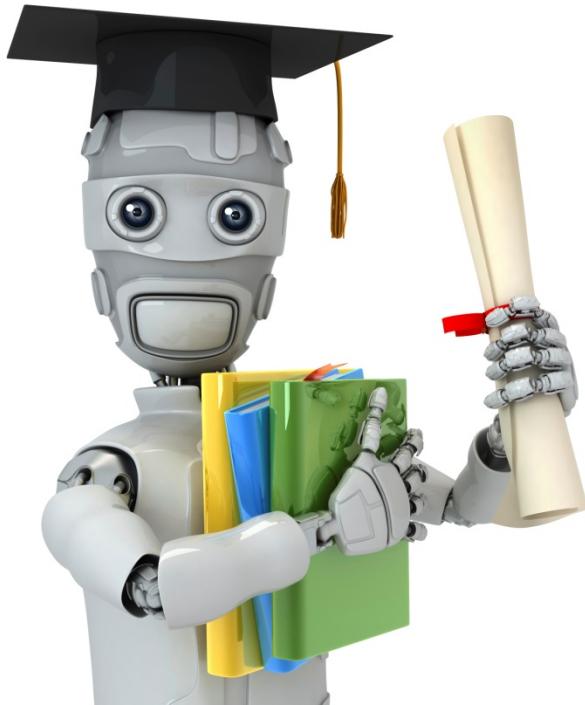
Looks same as  
for linear regression!

$$= \frac{1}{m} \sum_{i=1}^m \left[ (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j$$

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

logistic regression

don't have to  
regularize b



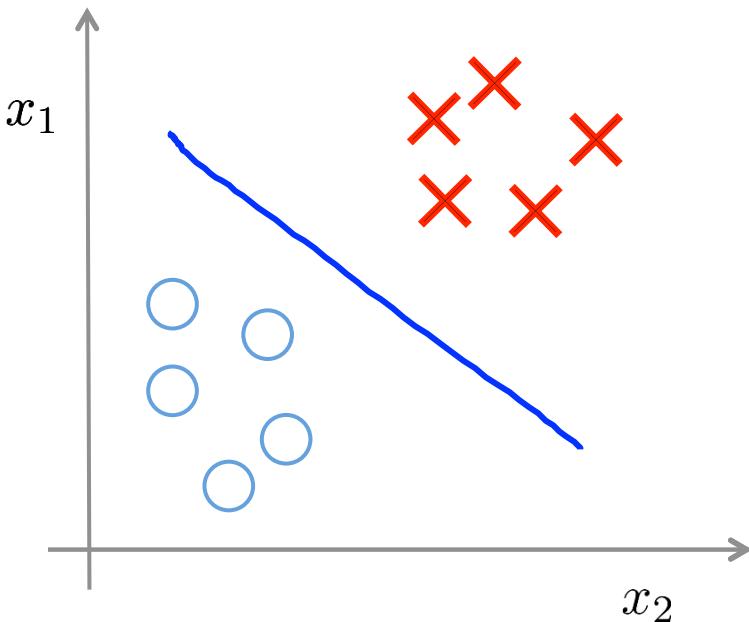
Machine Learning

# Clustering

---

## Unsupervised learning introduction

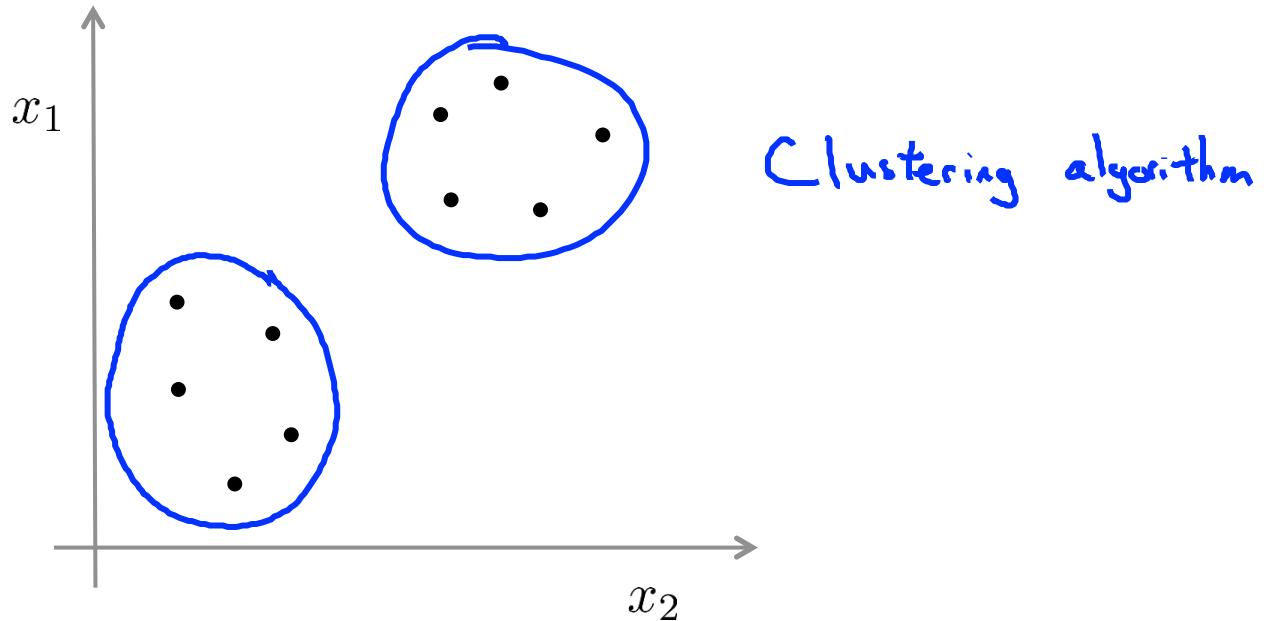
# Supervised learning



Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$

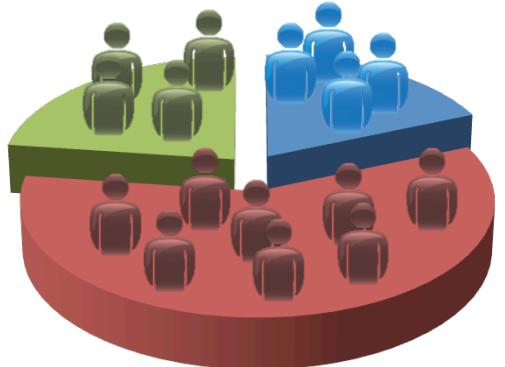


# Unsupervised learning

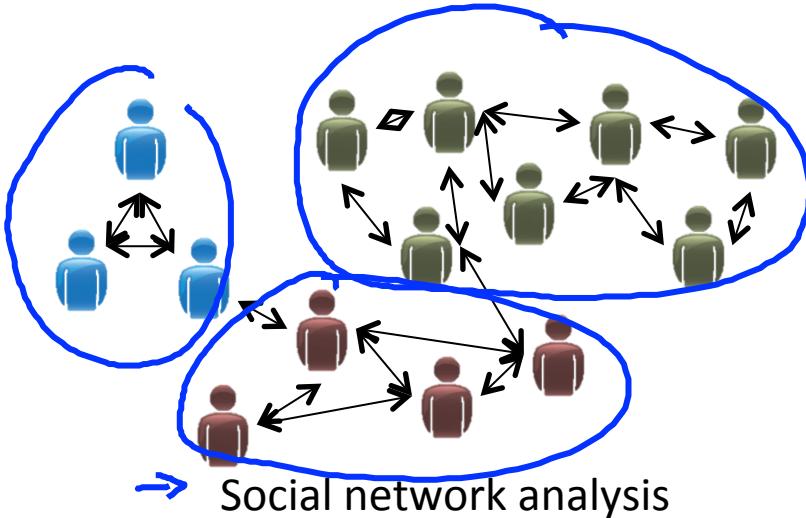


Training set:  $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \dots, \underline{x}^{(m)}\}$   $\leftarrow$

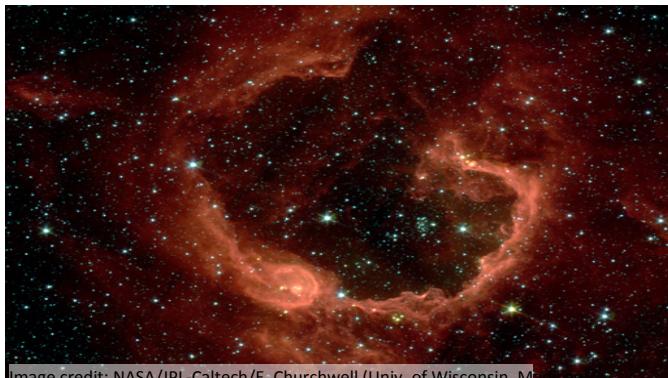
# Applications of clustering



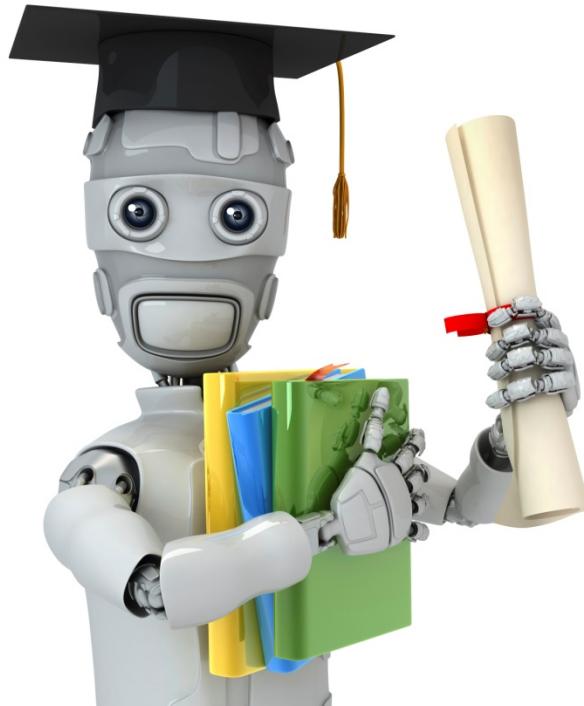
→ Market segmentation



Organize computing clusters



→ Astronomical data analysis

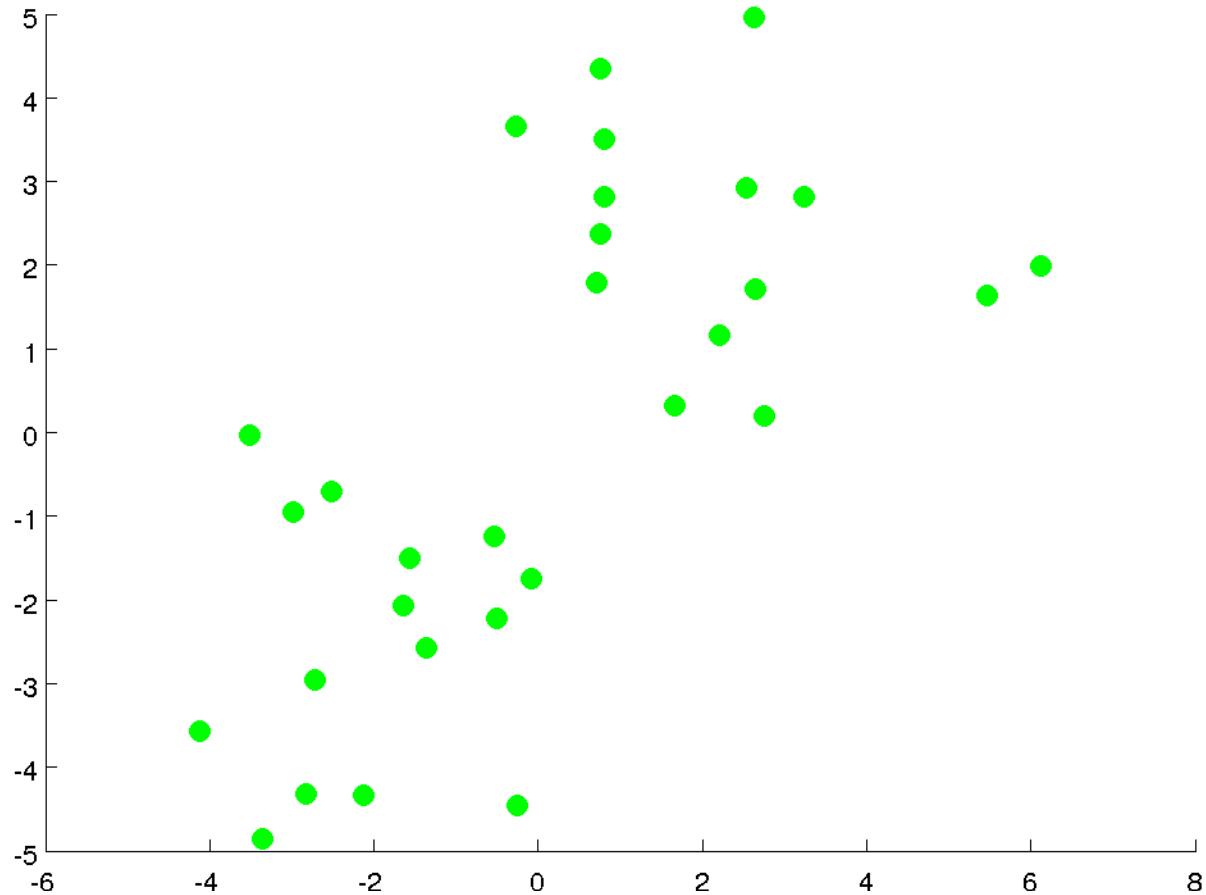


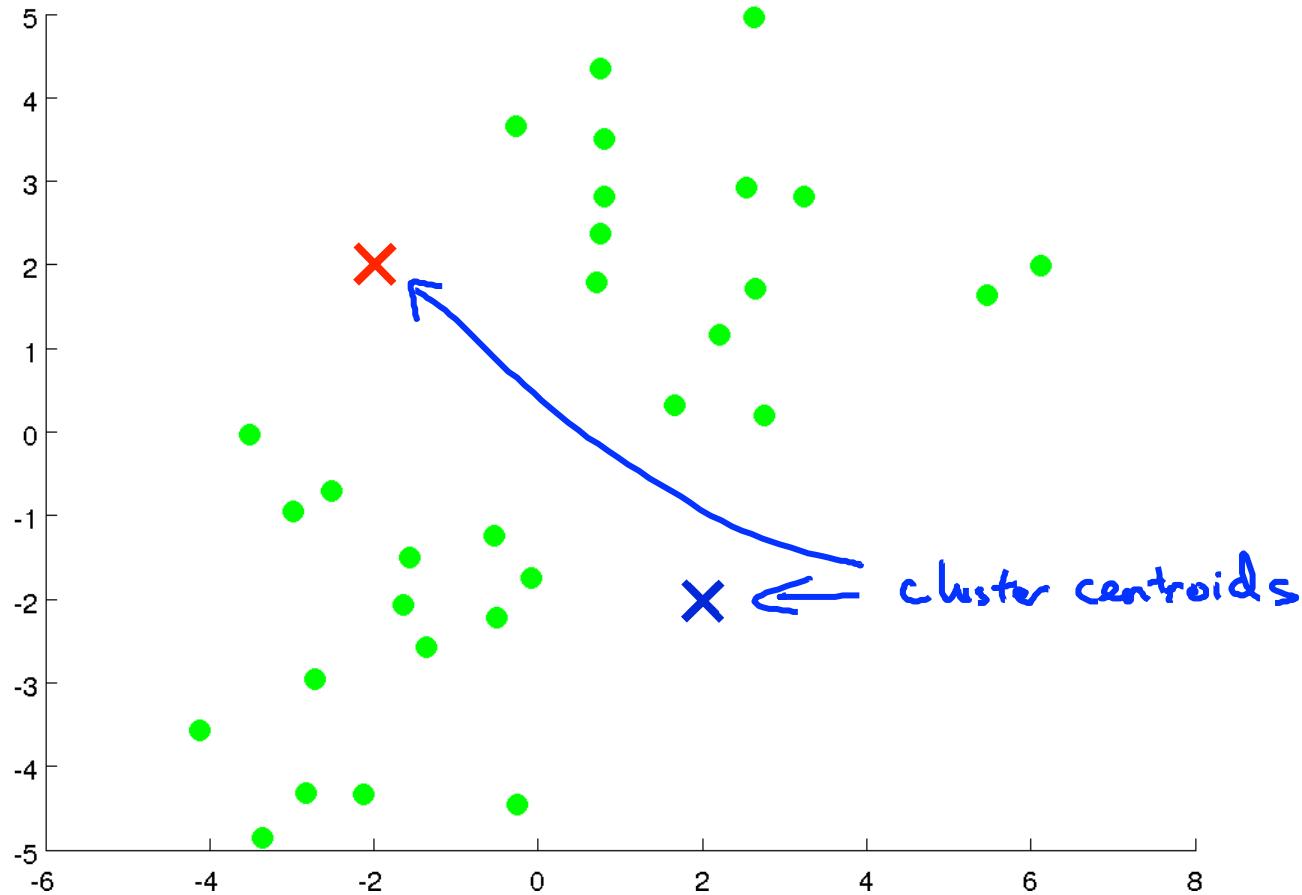
Machine Learning

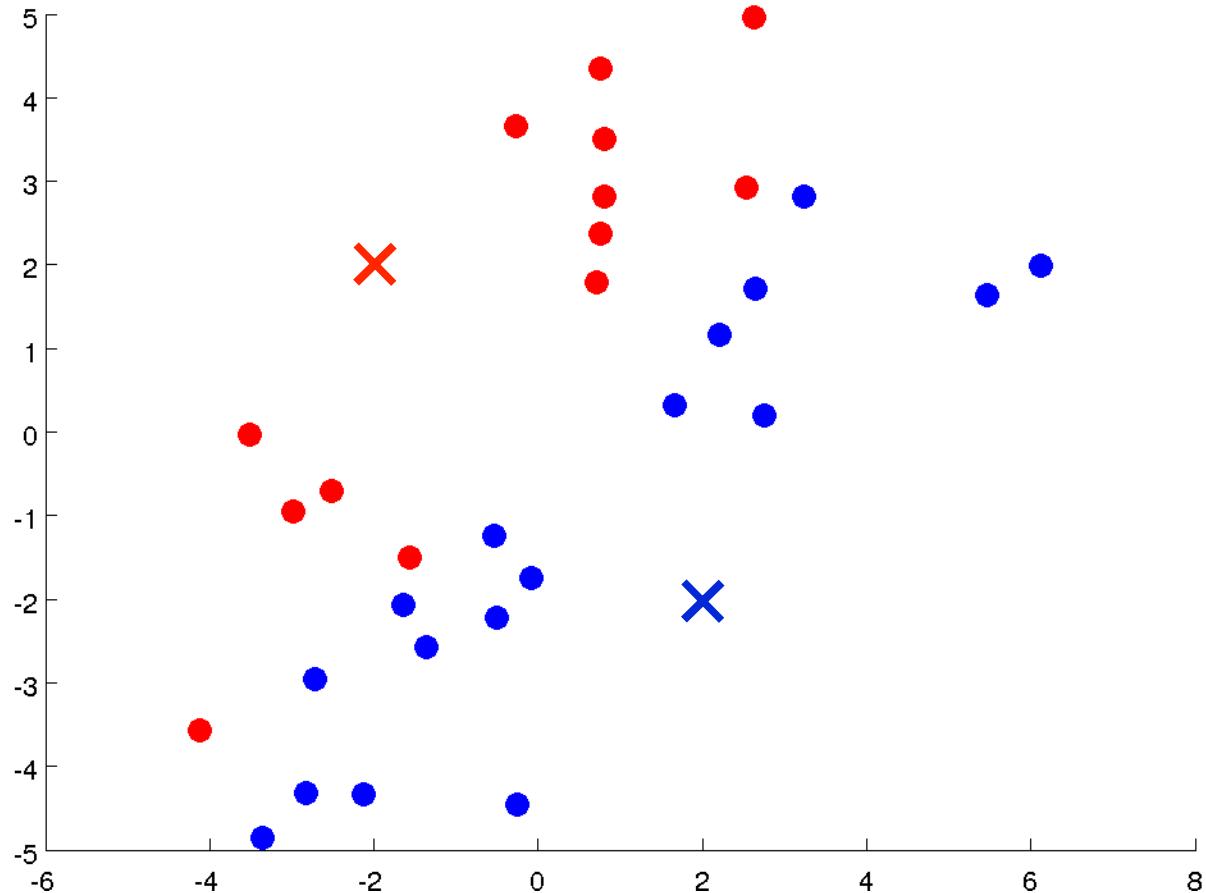
# Clustering

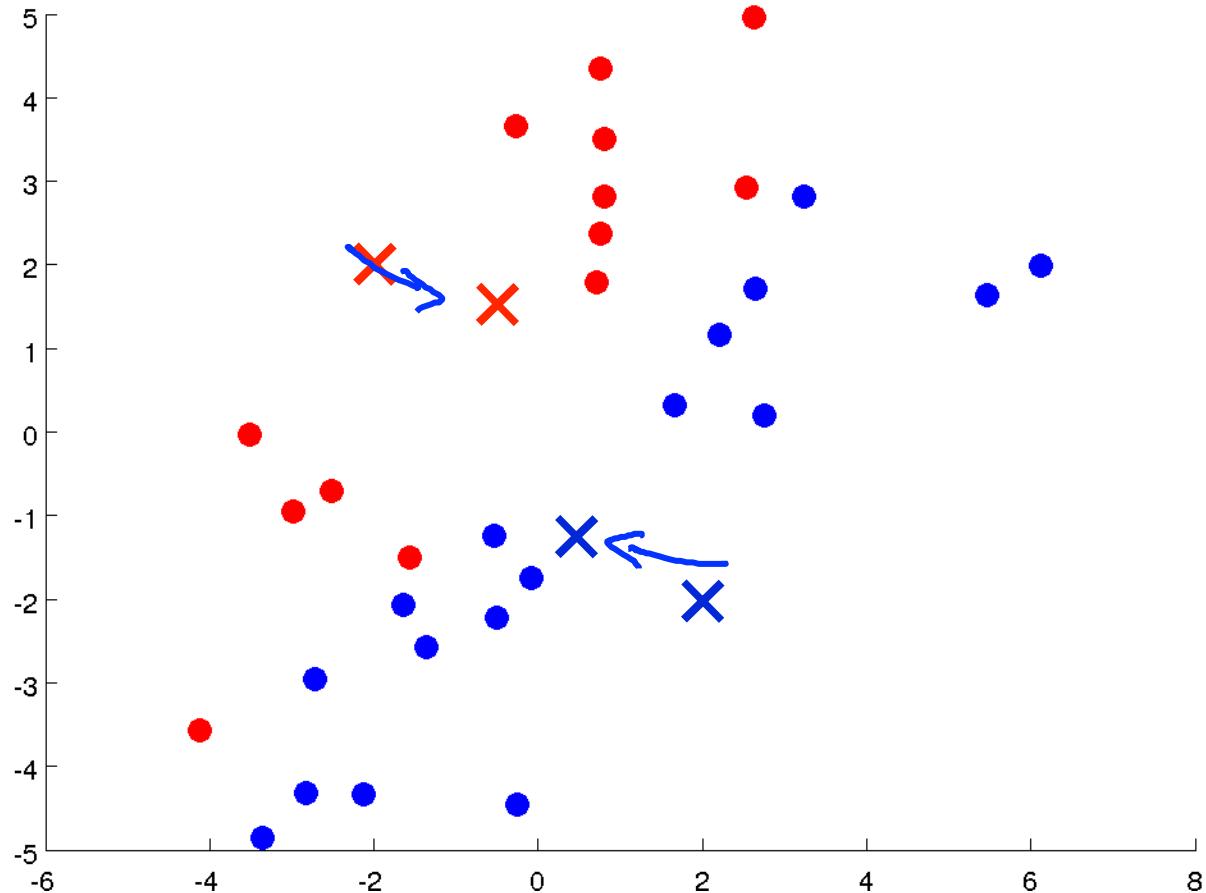
---

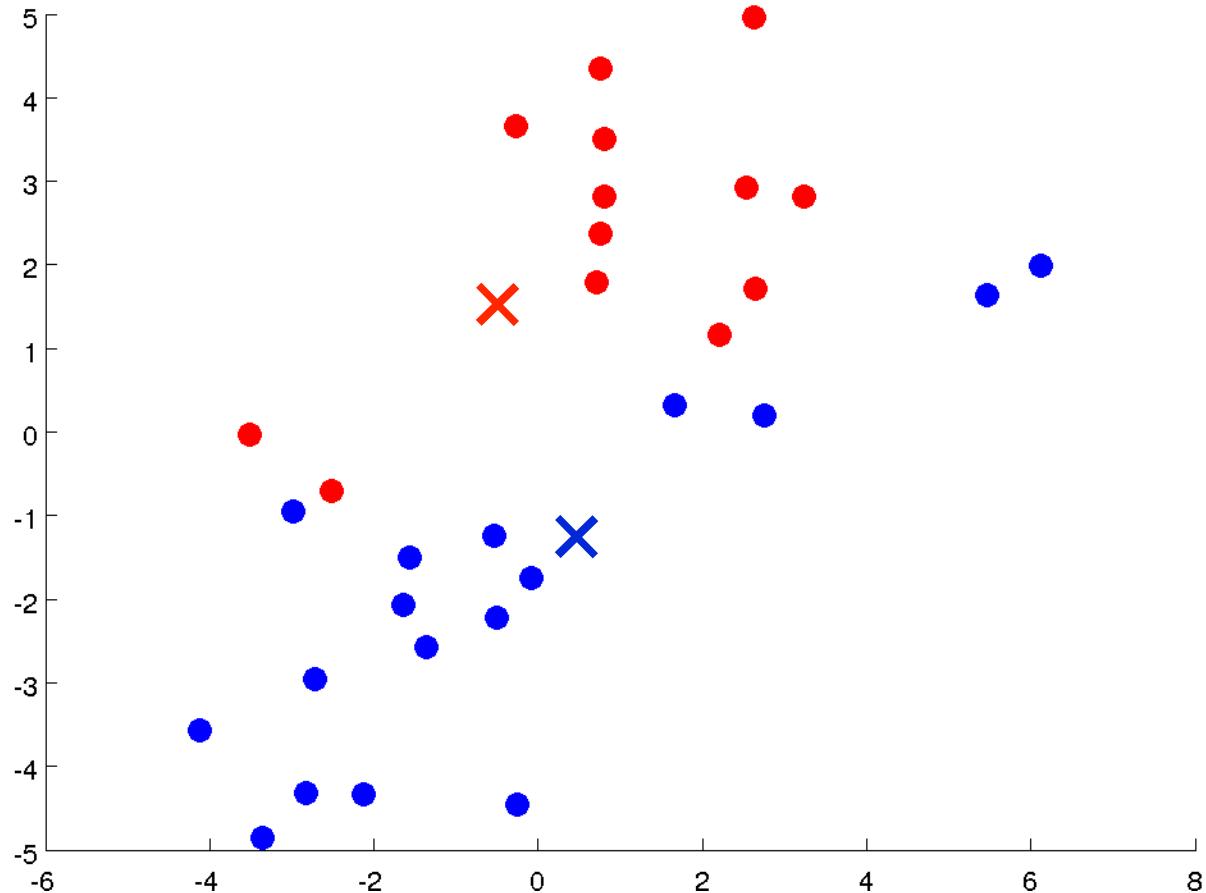
K-means  
algorithm

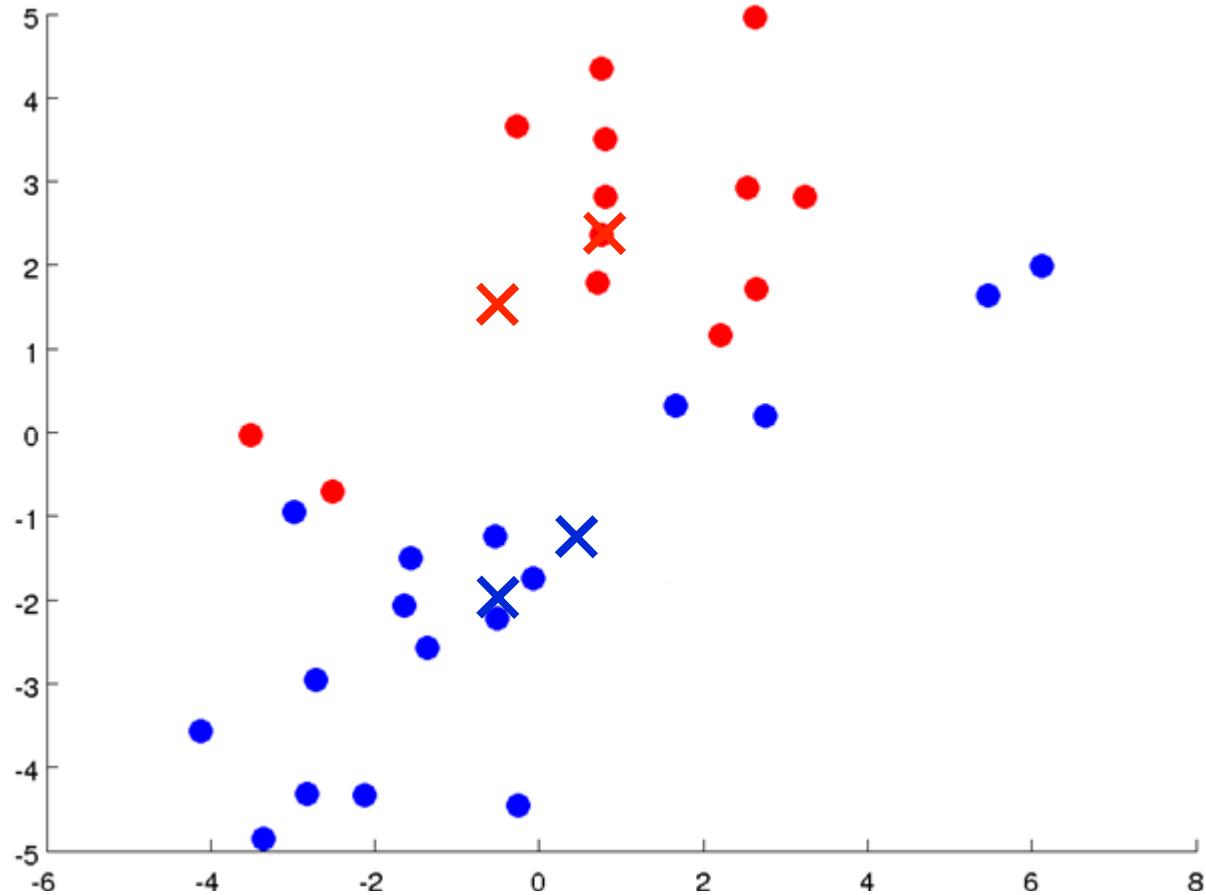


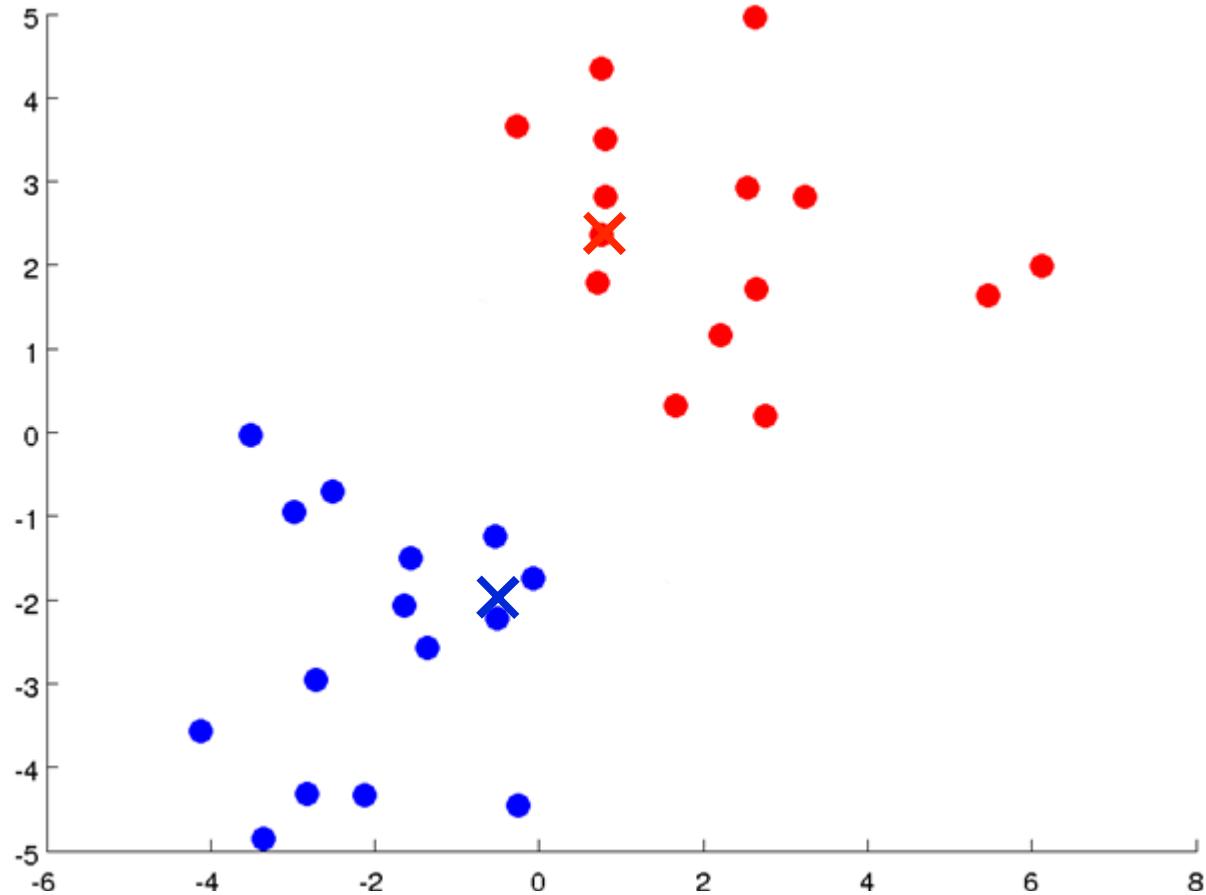


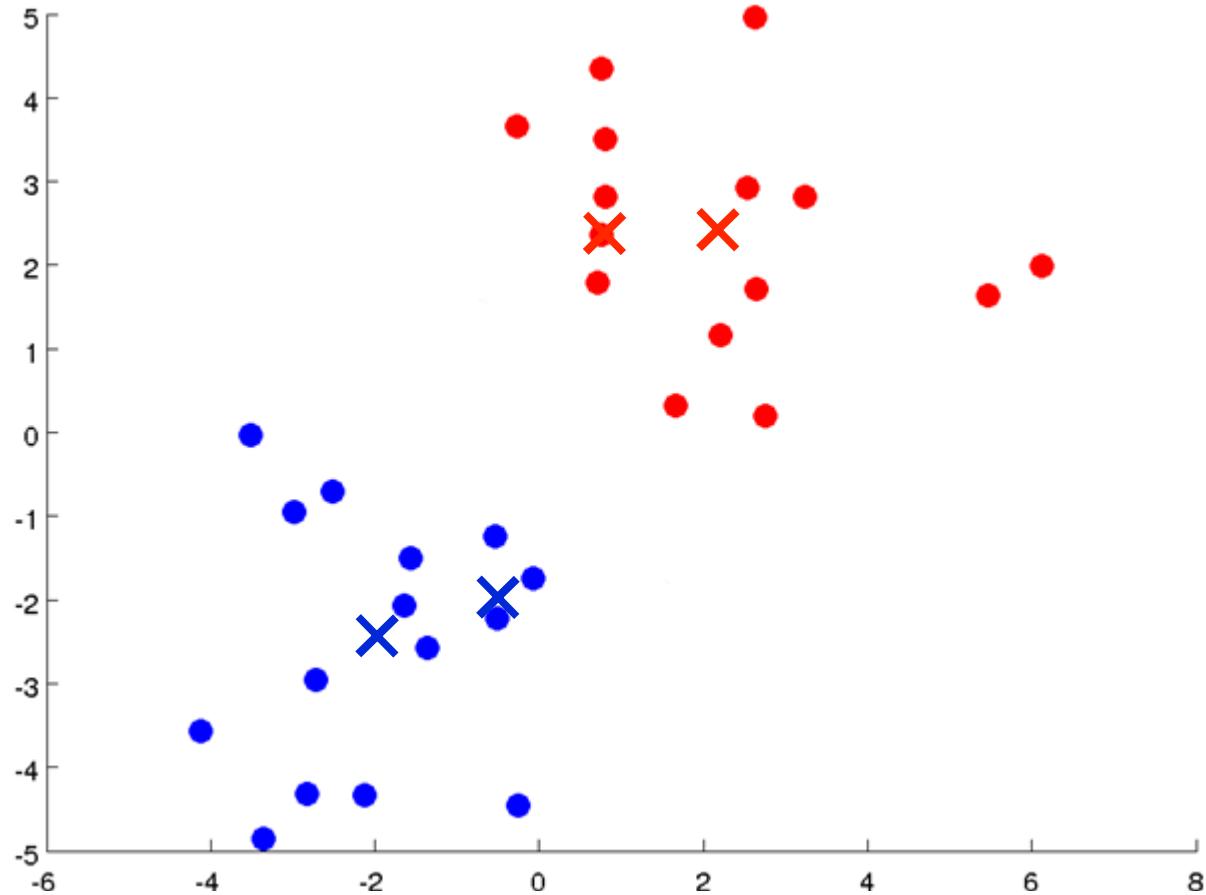


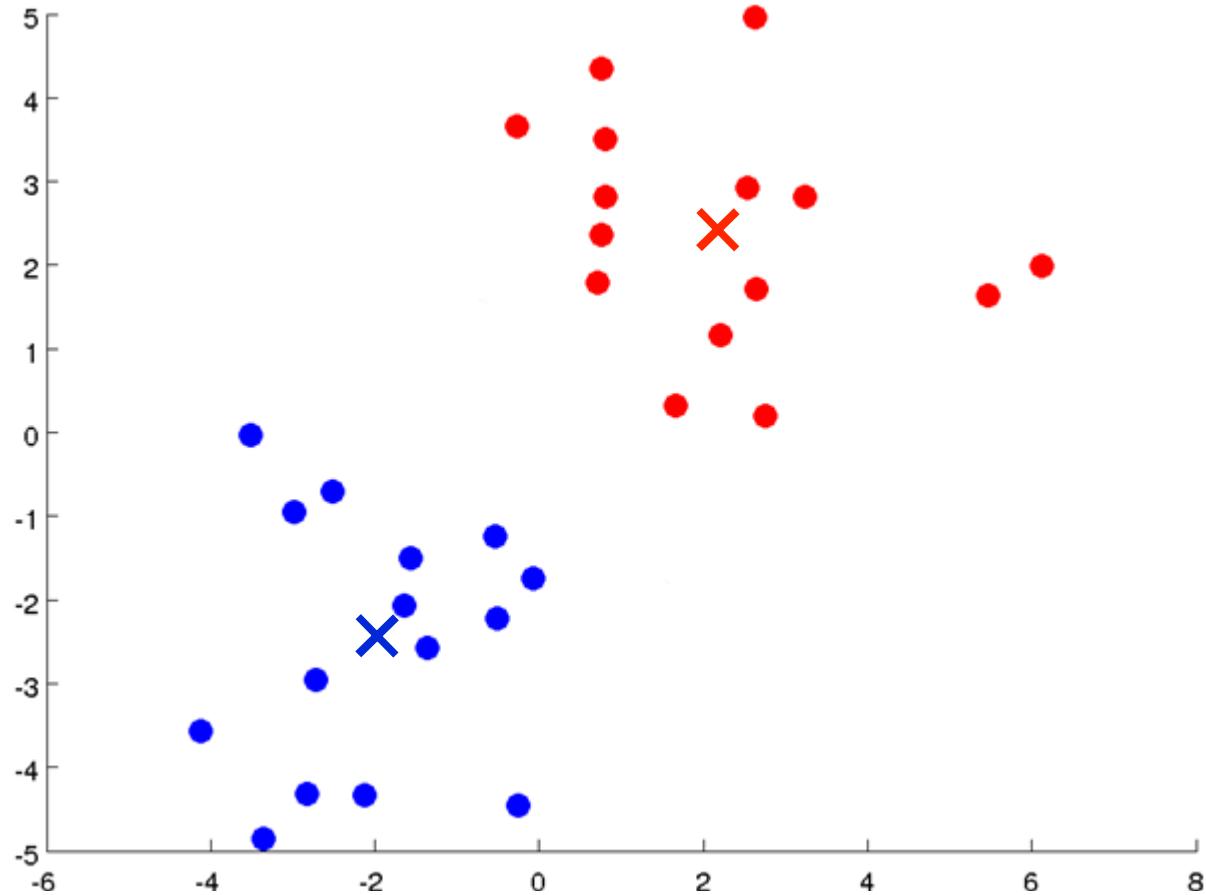












# K-means algorithm

Input:

- $K$  (number of clusters) 
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  

$x^{(i)} \in \mathbb{R}^n$  (drop  $x_0 = 1$  convention)

## K-means algorithm

$$\mu_1 \quad \mu_2$$

Randomly initialize  $K$  cluster centroids  $\underline{\mu}_1, \underline{\mu}_2, \dots, \underline{\mu}_K \in \mathbb{R}^n$

Repeat {

Cluster  
assignment  
step

for  $i = 1$  to  $m$

$\underline{c}^{(i)}$  := index (from 1 to  $K$ ) of cluster centroid  
closest to  $x^{(i)}$

$$\min_k \|\underline{x}^{(i)} - \underline{\mu}_k\|^2$$

for  $k = 1$  to  $K$

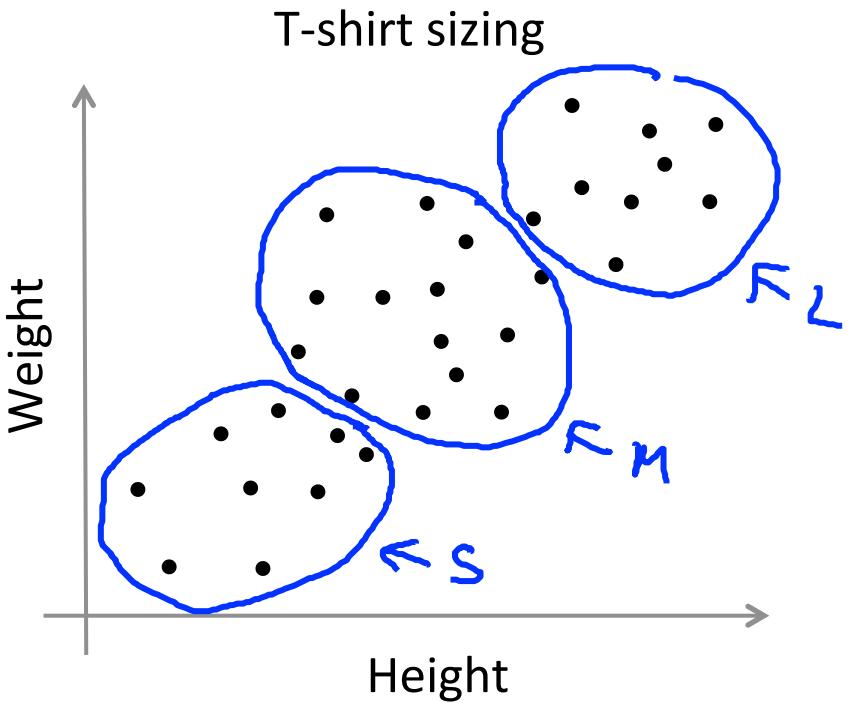
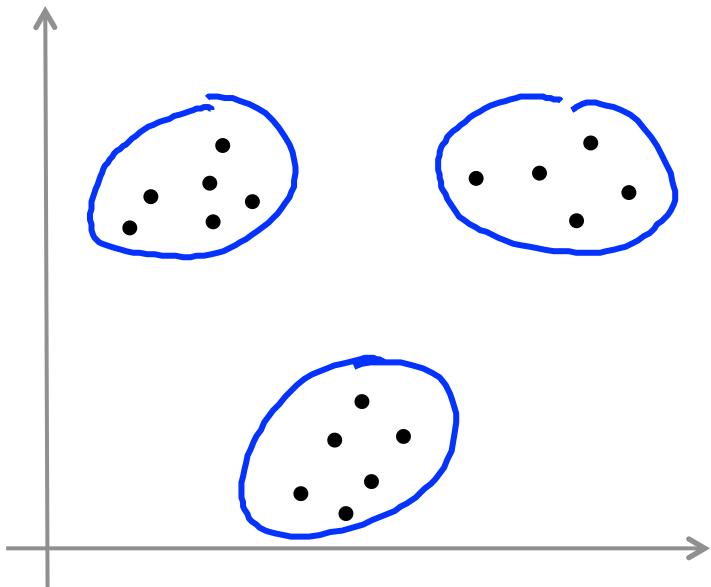
$\rightarrow \underline{\mu}_k$  := average (mean) of points assigned to cluster  $k$   
 $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$   $\rightarrow c^{(1)}=2, c^{(2)}=2, c^{(3)}=2, c^{(4)}=2$

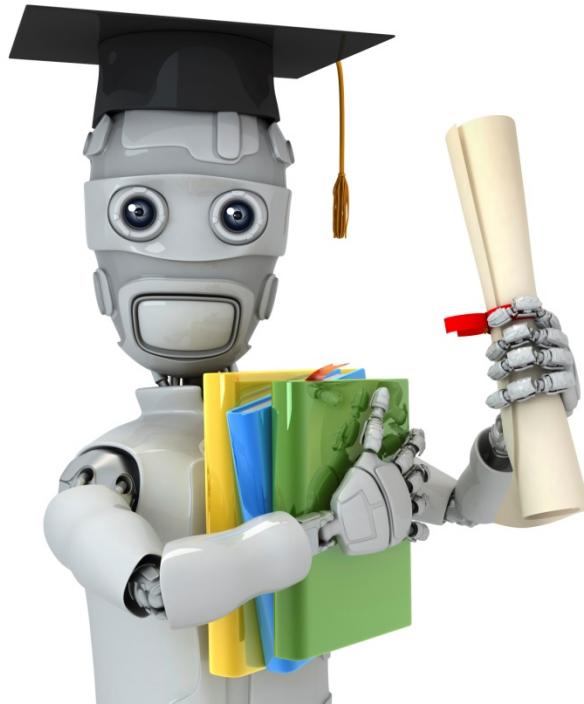
$$\underline{\mu}_2 = \frac{1}{4} \left[ \underline{x}^{(1)} + \underline{x}^{(2)} + \underline{x}^{(3)} + \underline{x}^{(4)} \right] \in \mathbb{R}^n$$

Move  
centroid

## K-means for non-separated clusters

S, M, L





Machine Learning

# Clustering Optimization objective

---

## K-means optimization objective

- $c^{(i)}$  = index of cluster ( $1, 2, \dots, K$ ) to which example  $x^{(i)}$  is currently assigned
- $\mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )  $K$   
 $k \in \{1, 2, \dots, K\}$
- $\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned  $x^{(i)} \rightarrow S$   
 $c^{(i)} = s$   
 $\mu_{c^{(i)}} = \mu_s$

Optimization objective:

$$\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

min  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$       Distortion

# K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {      [Cluster assignment step]  
                Minimize  $J(\dots)$  wrt  $[c^{(1)}, c^{(2)}, \dots, c^{(n)}] \leftarrow$   
                (holding  $\mu_1, \dots, \mu_K$  fixed)

for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid  
closest to  $x^{(i)}$

move  
centroid

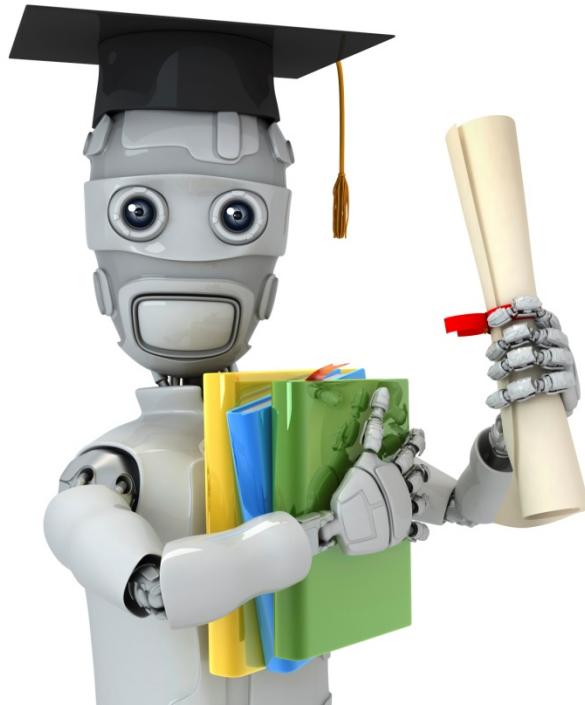
for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

minimize  $J(\dots)$  wrt

$[\mu_1, \dots, \mu_K]$



Machine Learning

# Clustering

---

## Random initialization

## K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

    for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid  
        closest to  $x^{(i)}$

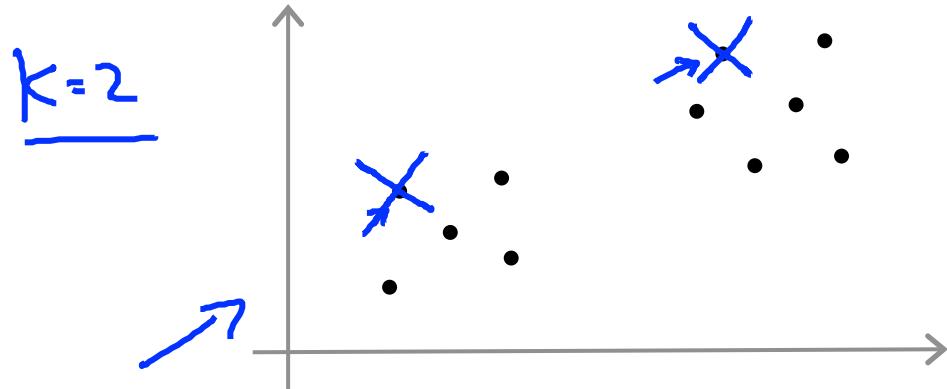
    for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

## Random initialization

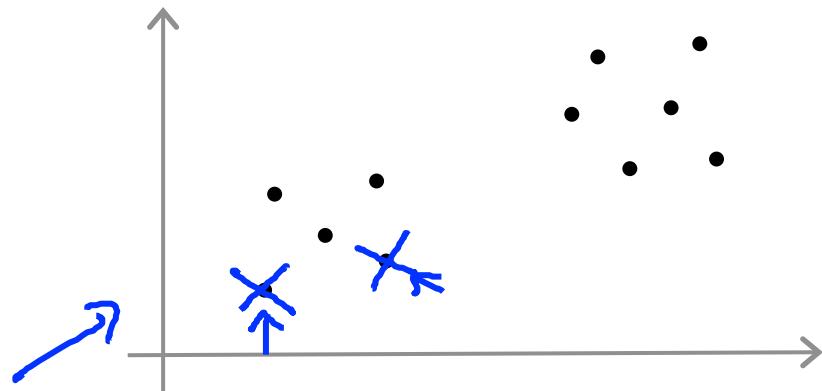
Should have  $K < m$



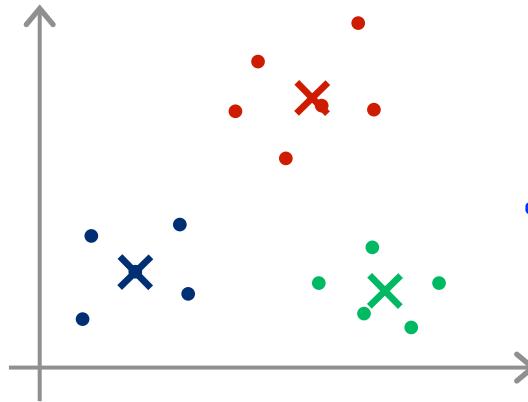
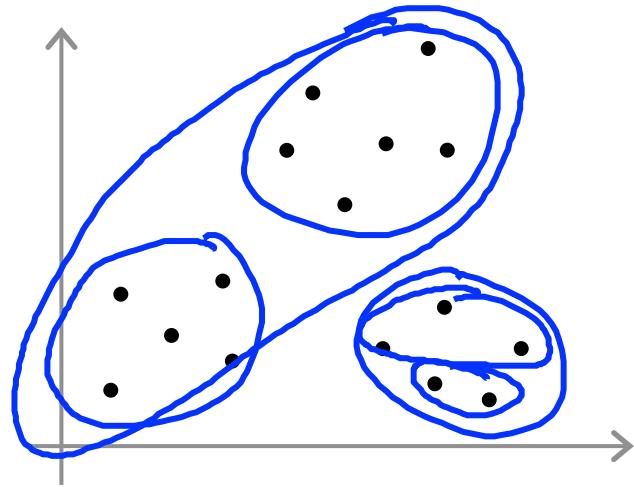
Randomly pick  $K$  training examples.

Set  $\mu_1, \dots, \mu_K$  equal to these  $K$  examples.

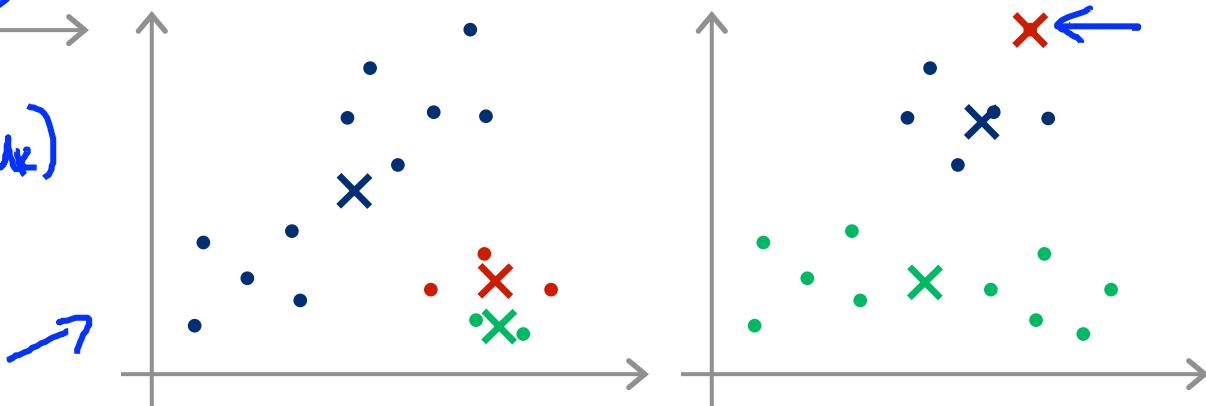
$$\begin{aligned}\mu_1 &= x^{(1)} \\ \mu_2 &= x^{(2)} \\ &\vdots\end{aligned}$$



## Local optima



$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k)$$



## Random initialization

For i = 1 to 100 {

    Randomly initialize K-means.

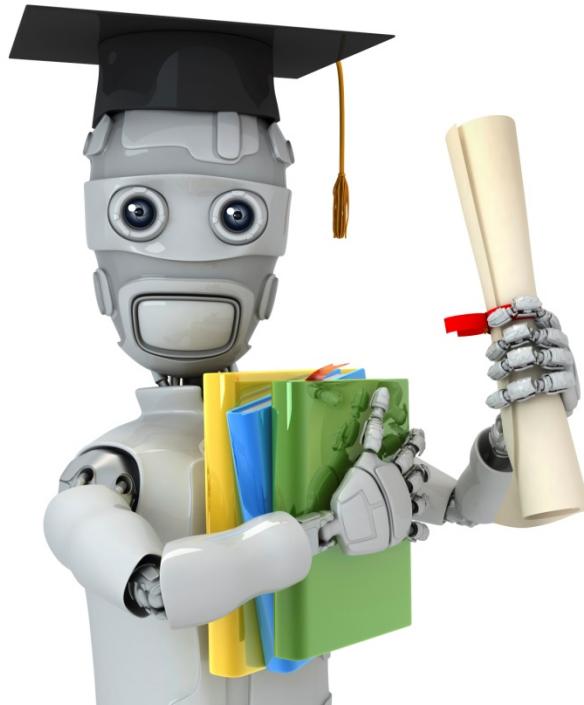
    Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ .

    Compute cost function (distortion)

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

}

Pick clustering that gave lowest cost  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

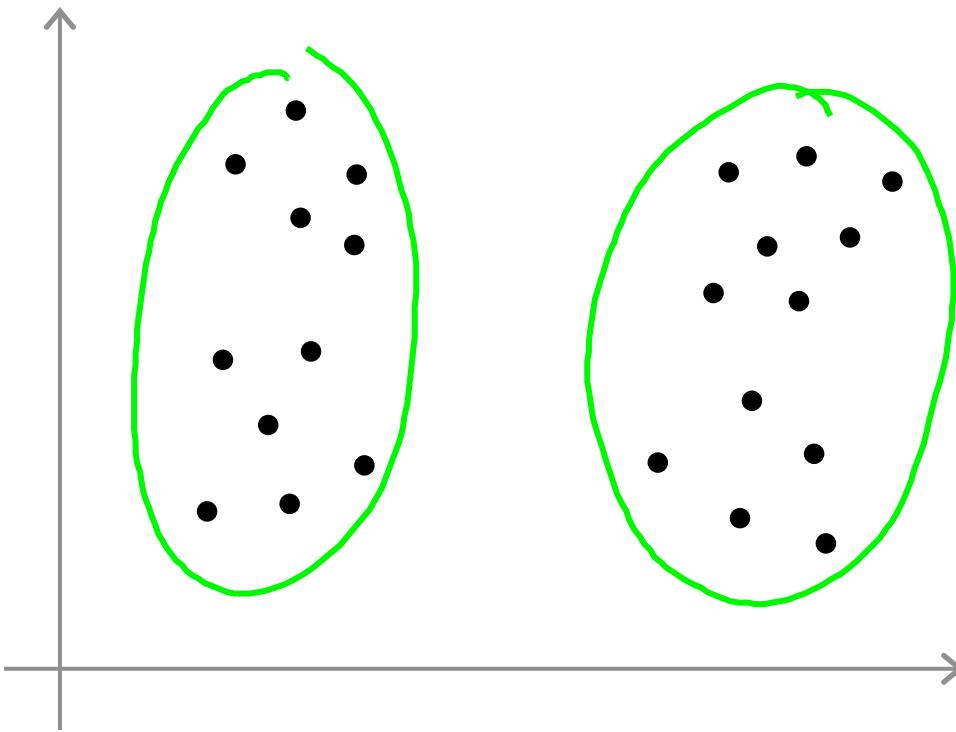


Machine Learning

# Clustering

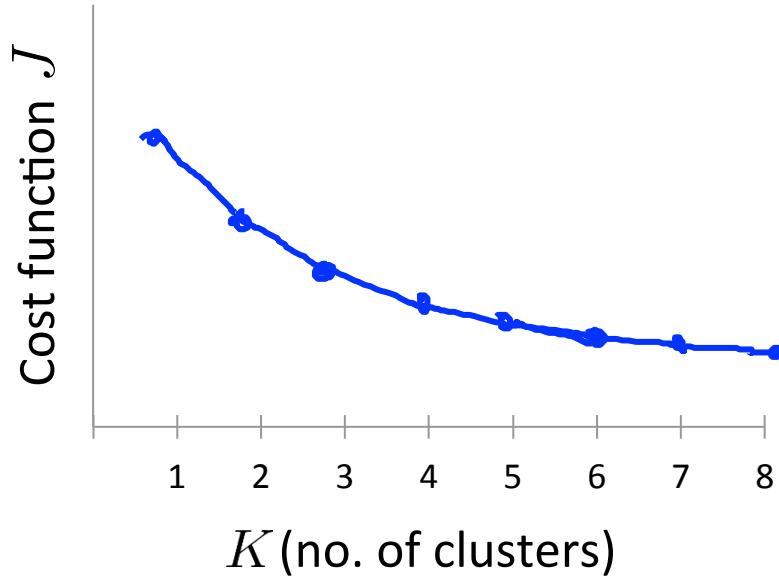
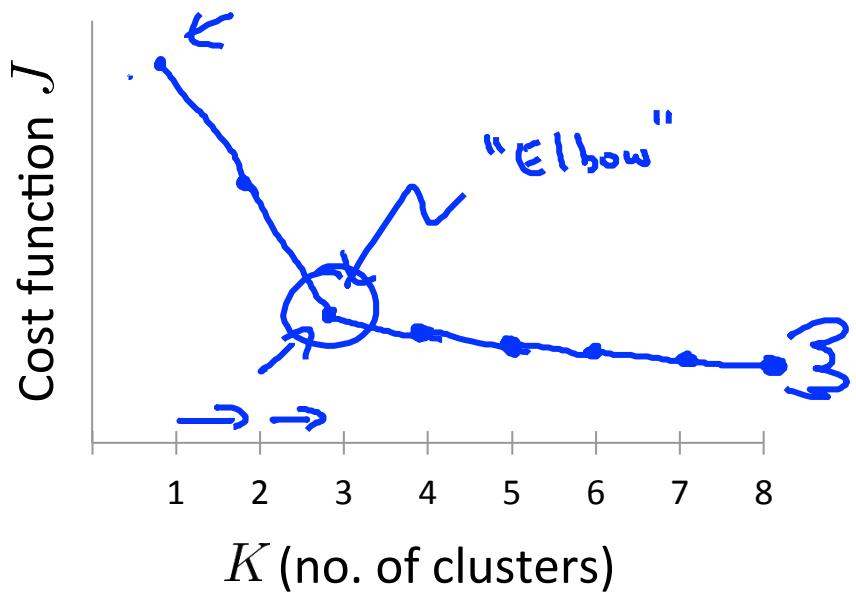
## Choosing the number of clusters

# What is the right value of K?



# Choosing the value of K

Elbow method:

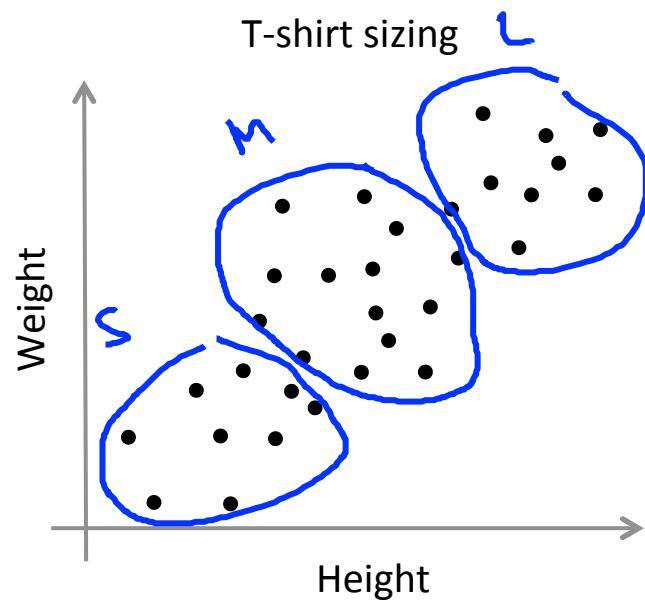


## Choosing the value of K

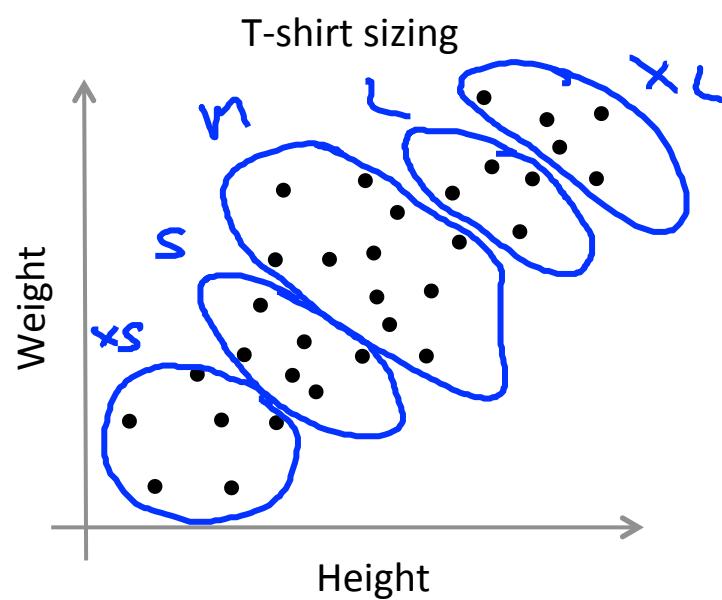
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

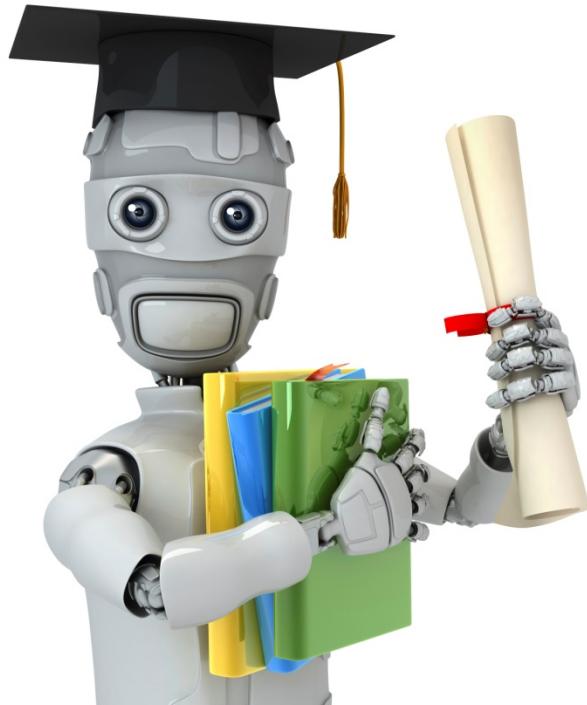
$K=3$       S, M, L

E.g.



$K=5$       XS, S, M, L, XL





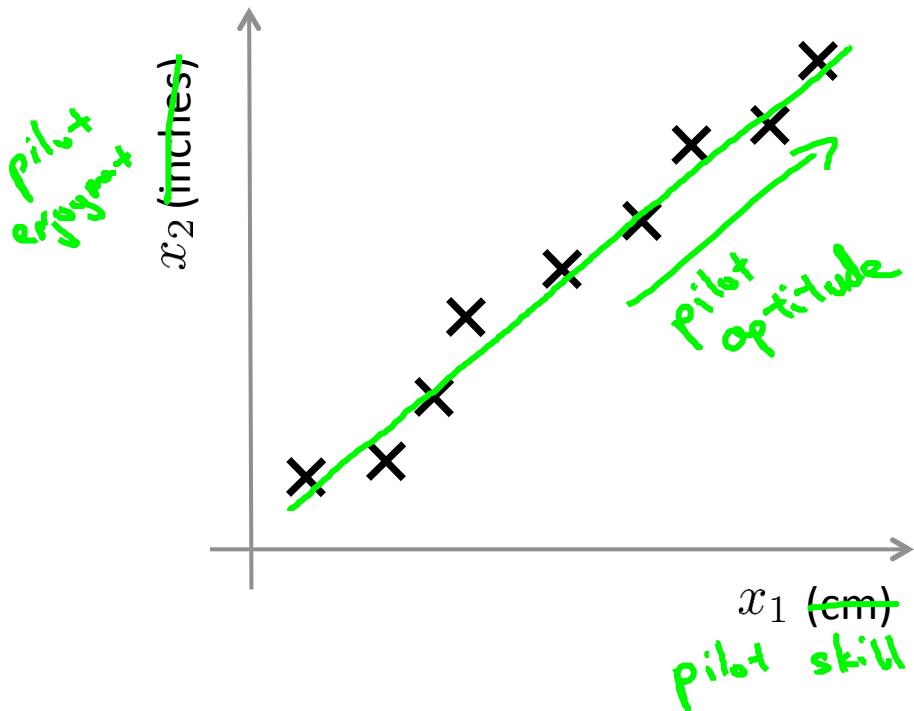
Machine Learning

# Dimensionality Reduction

---

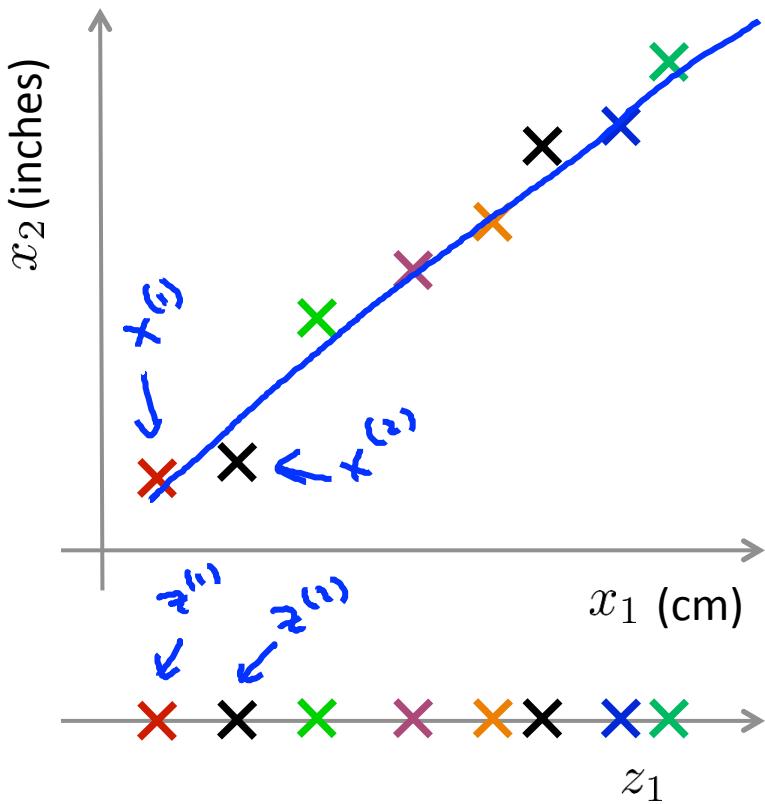
Motivation I:  
Data Compression

# Data Compression



Reduce data from  
2D to 1D

# Data Compression



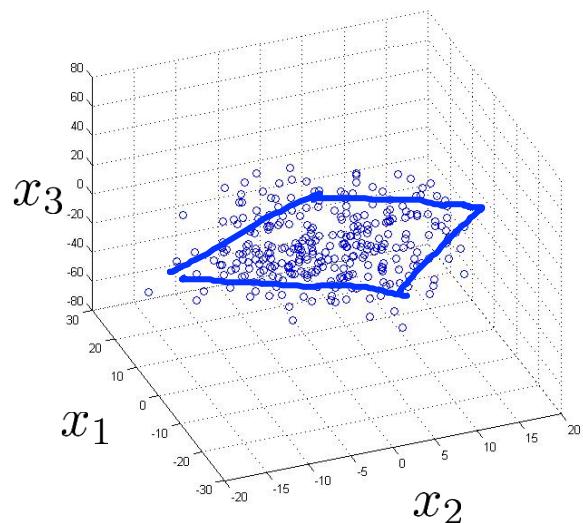
Reduce data from  
2D to 1D

$$\begin{aligned}x^{(1)} \in \mathbb{R}^2 &\rightarrow z^{(1)} \in \mathbb{R} \\x^{(2)} \in \mathbb{R}^2 &\rightarrow z^{(2)} \in \mathbb{R} \\&\vdots \\x^{(m)} \in \mathbb{R}^2 &\rightarrow z^{(m)} \in \mathbb{R}\end{aligned}$$

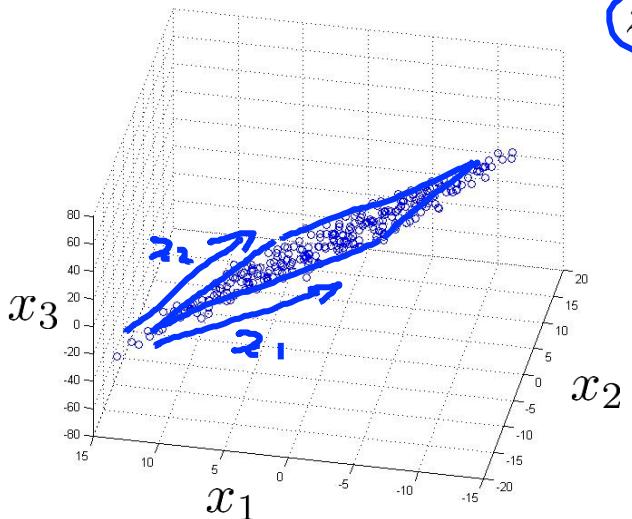
# Data Compression

10000  $\rightarrow$  1000

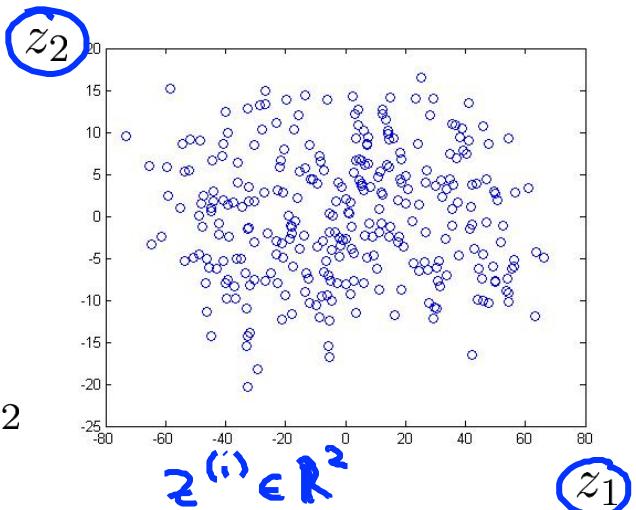
Reduce data from 3D to 2D



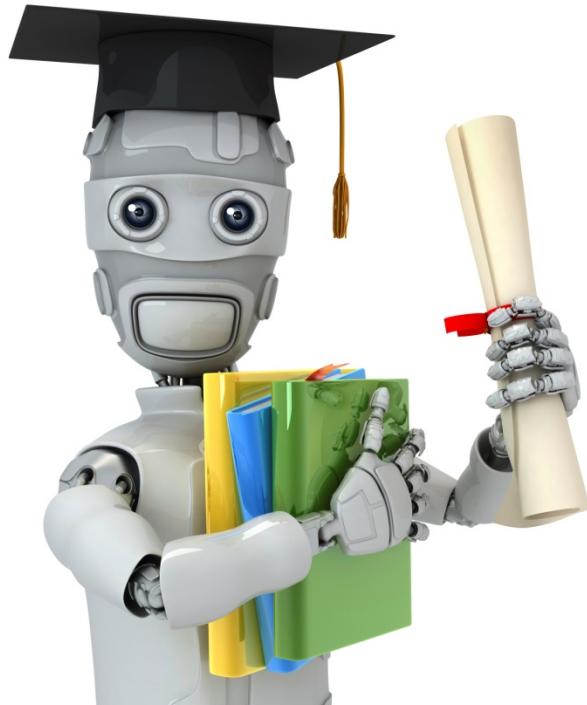
$$x^{(1)} \in \mathbb{R}^3$$



$$z^{(1)} \in \mathbb{R}^2$$



$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \tilde{z}^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix}$$



Machine Learning

# Dimensionality Reduction

---

## Motivation II: Data Visualization

# Data Visualization

$$x \in \mathbb{R}^{50}$$

$$x^{(i)} \in \mathbb{R}^{50}$$

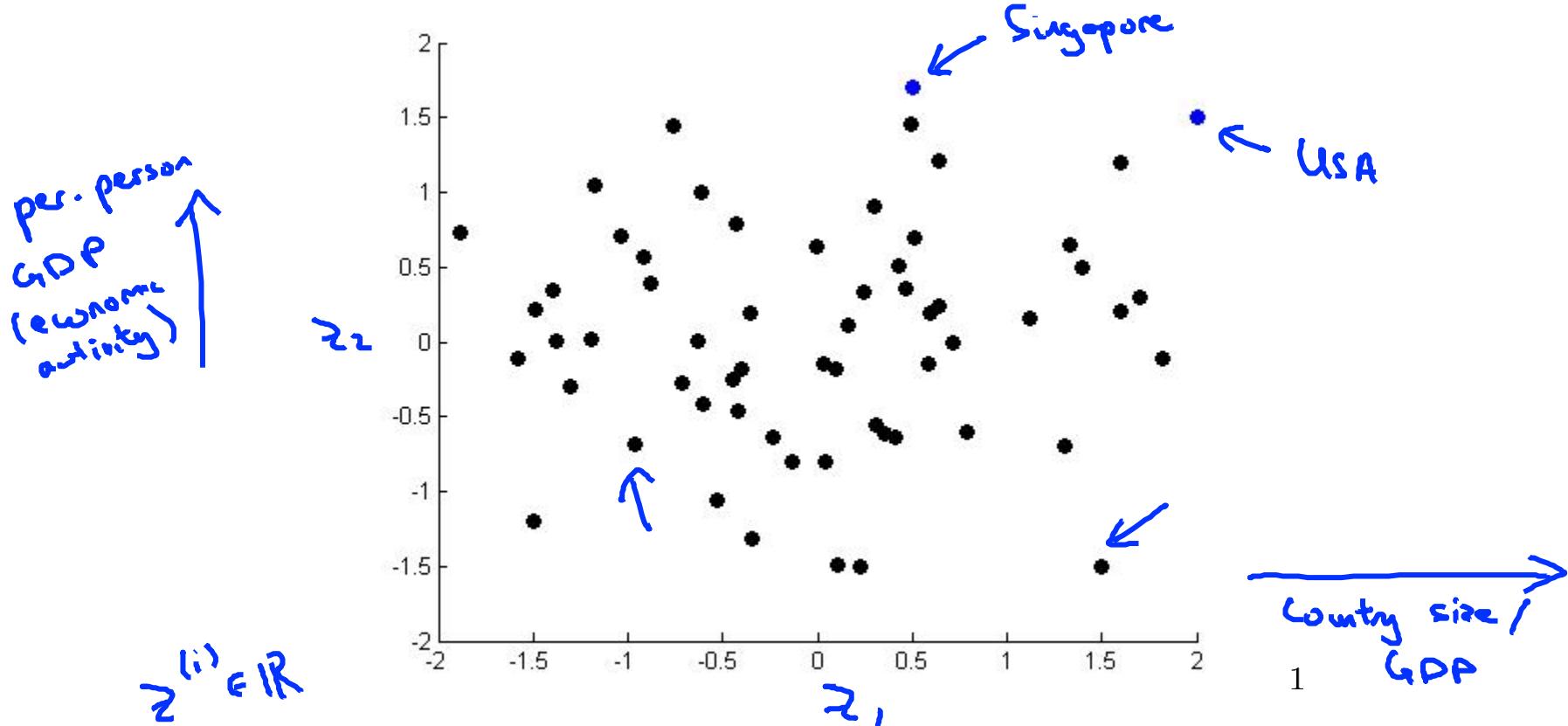
$x_6$

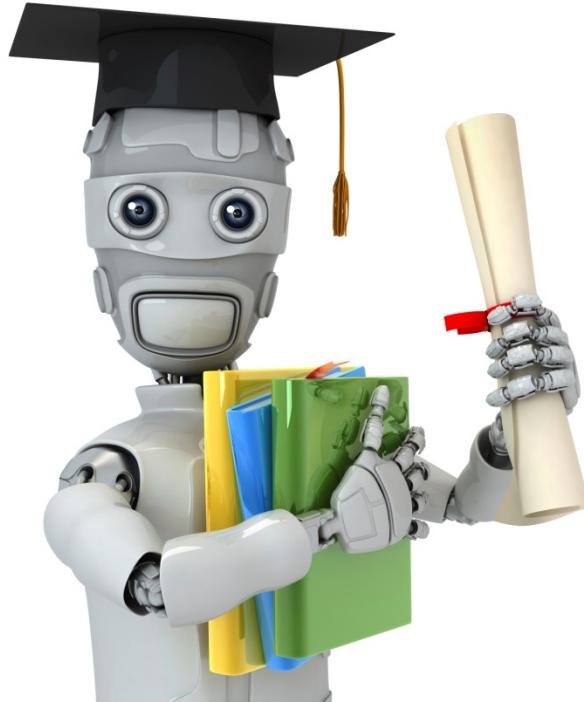
Country	$x_1$ GDP (trillions of US\$)	$x_2$ Per capita GDP (thousands of intl. \$)	$x_3$ Human Develop- ment Index	$x_4$ Life expectancy	$x_5$ Poverty Index (Gini as percentage)	Mean household income (thousands of US\$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	...

# Data Visualization

Country	$z_1$	$z_2$	$z^{(i)} \in \mathbb{R}^2$
Canada	1.6	1.2	
China	1.7	0.3	Reduce data
India	1.6	0.2	from 500
Russia	1.4	0.5	to 2D
Singapore	0.5	1.7	
USA	2	1.5	
...	...	...	

# Data Visualization





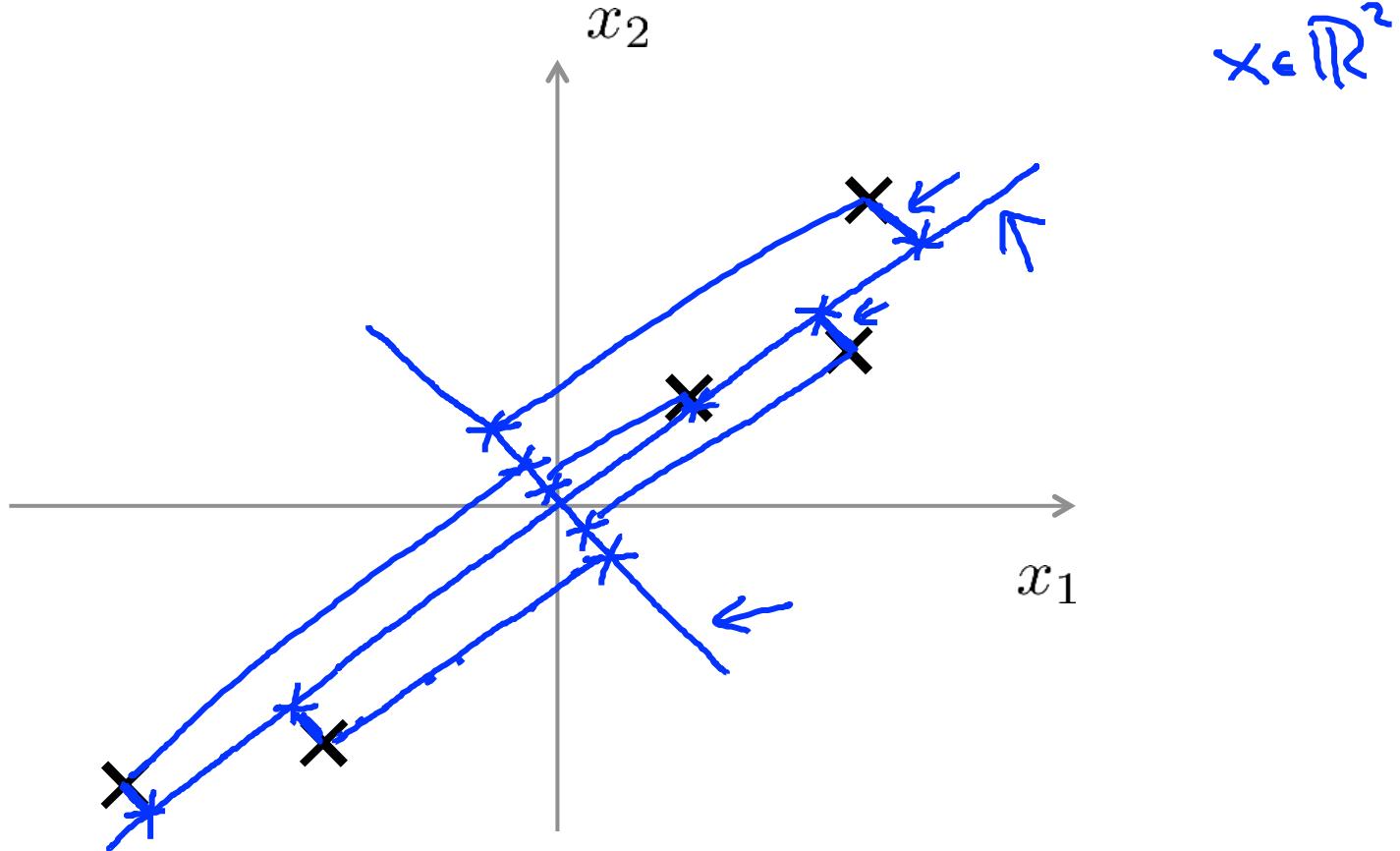
Machine Learning

# Dimensionality Reduction

---

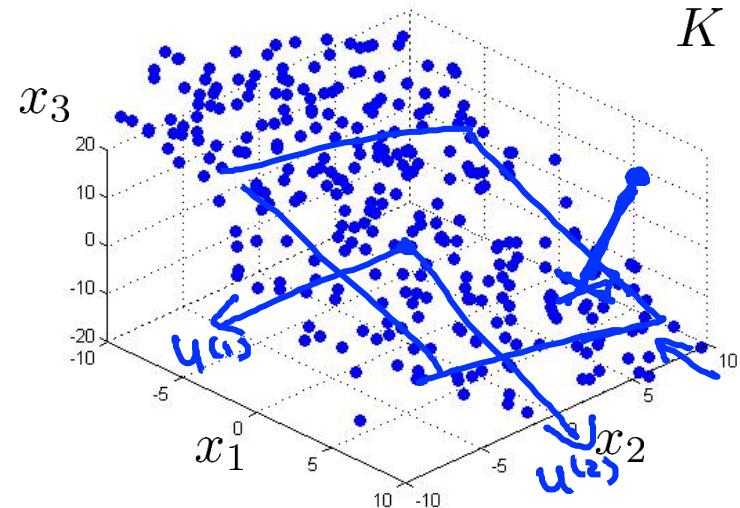
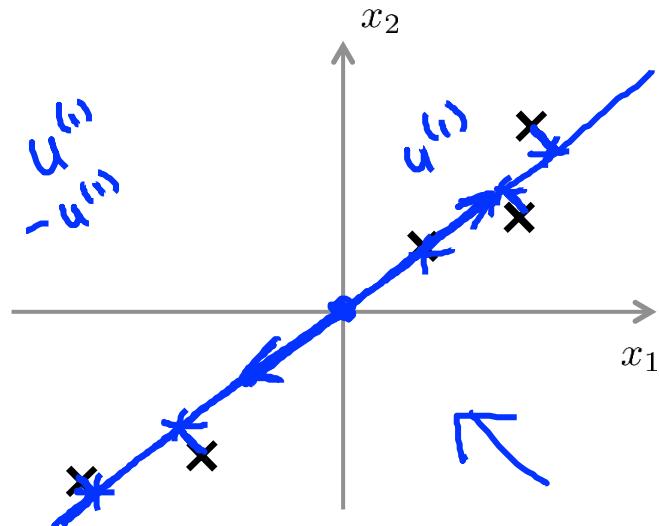
Principal Component  
Analysis problem  
formulation

# Principal Component Analysis (PCA) problem formulation



## Principal Component Analysis (PCA) problem formulation

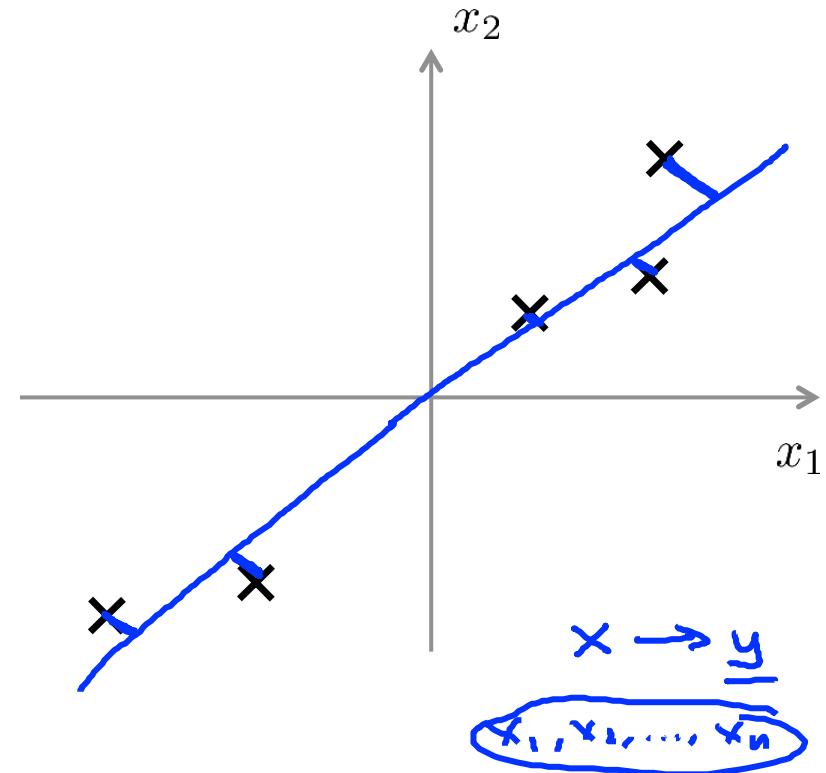
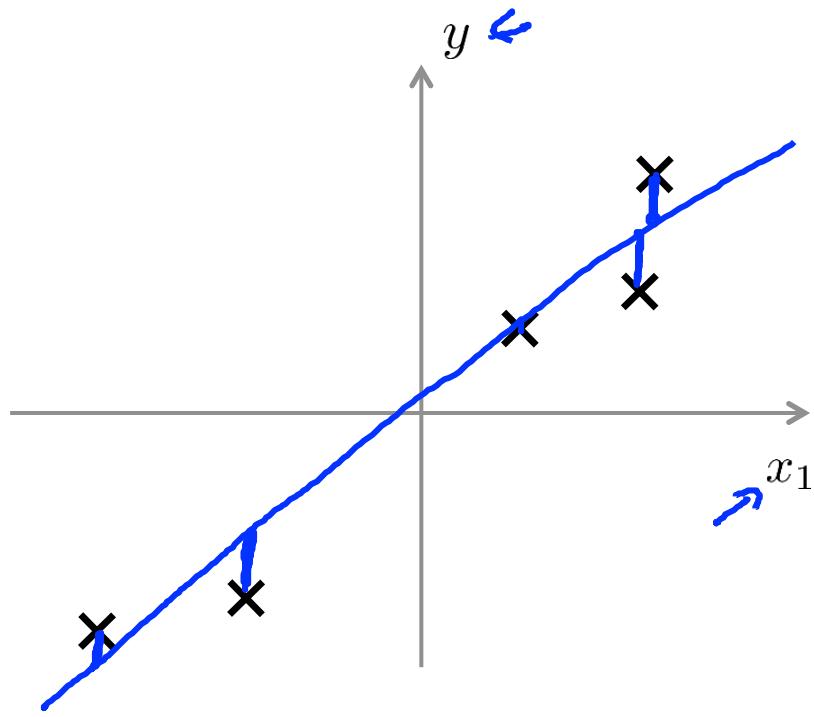
$$3D \rightarrow 2D \\ K = 2$$



Reduce from 2-dimension to 1-dimension: Find a direction (a vector  $\underline{u^{(1)} \in \mathbb{R}^n}$ ) onto which to project the data so as to minimize the projection error.

Reduce from  $n$ -dimension to  $k$ -dimension: Find  $k$  vectors  $\underline{u^{(1)}, u^{(2)}, \dots, u^{(k)}}$  onto which to project the data, so as to minimize the projection error.

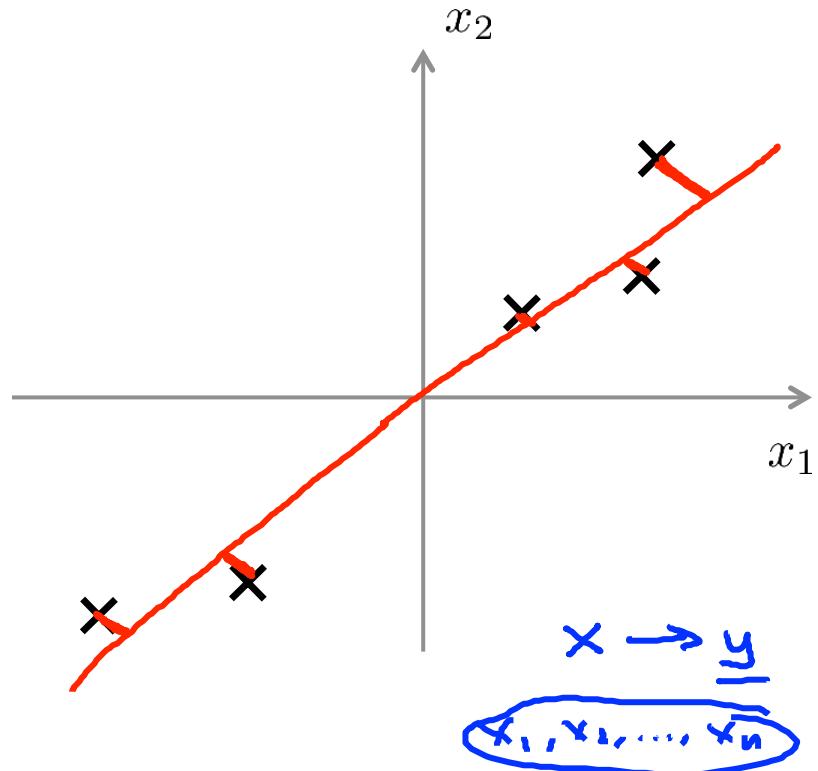
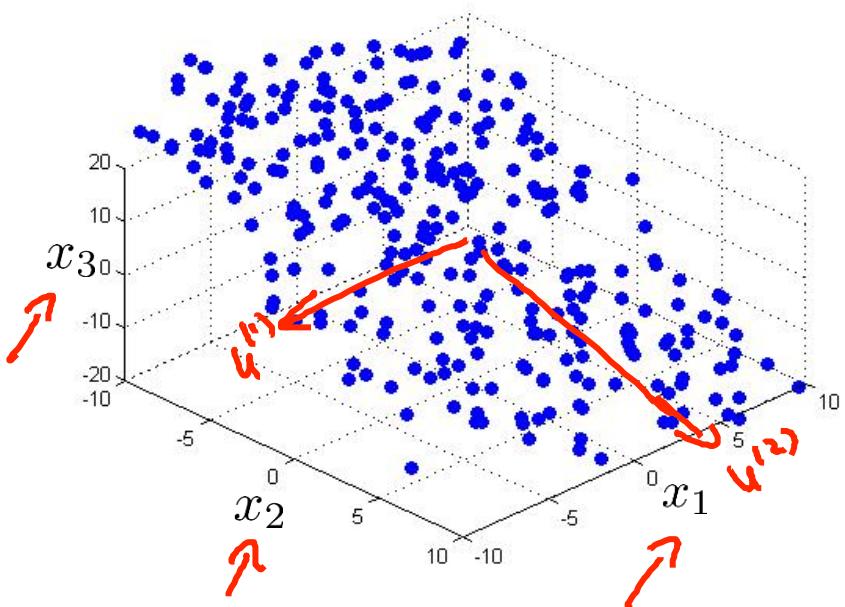
# PCA is not linear regression

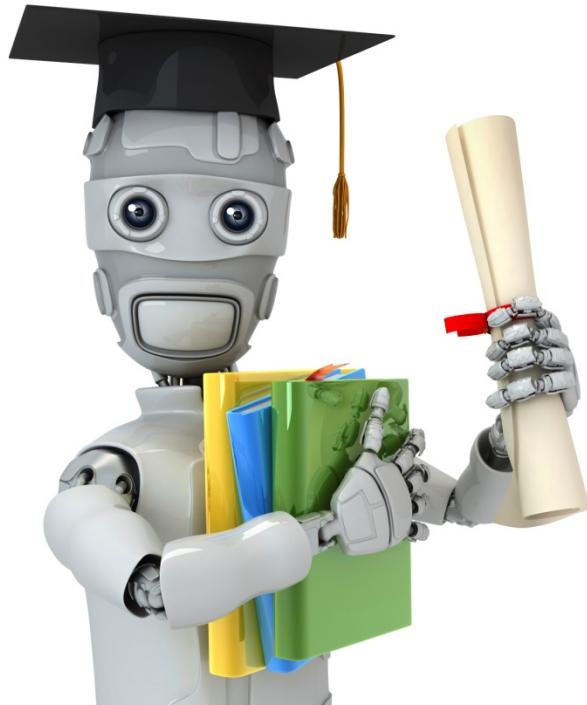


$x \rightarrow y$

$x_1, x_2, \dots, x_n$

# PCA is not linear regression





Machine Learning

# Dimensionality Reduction

---

Principal Component  
Analysis algorithm

## Data preprocessing

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  

Preprocessing (feature scaling/mean normalization):

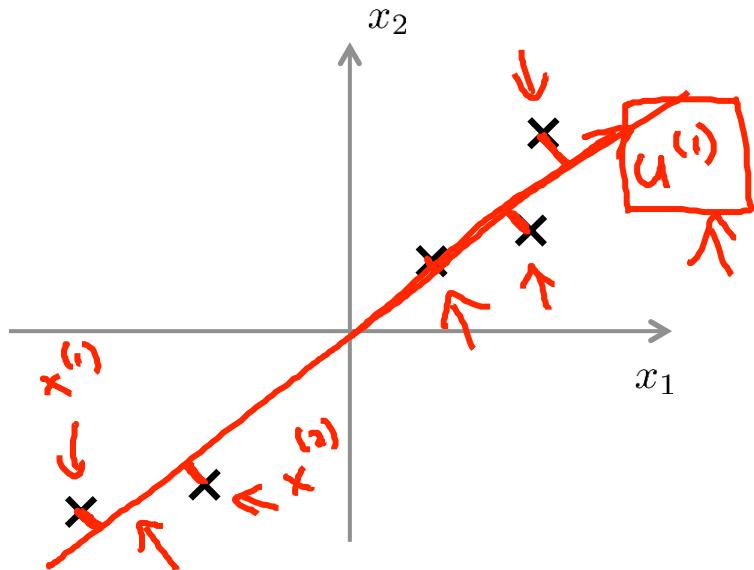
$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

Replace each  $x_j^{(i)}$  with  $\underline{x_j - \mu_j}$ .

If different features on different scales (e.g.,  $x_1$  = size of house,  $x_2$  = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

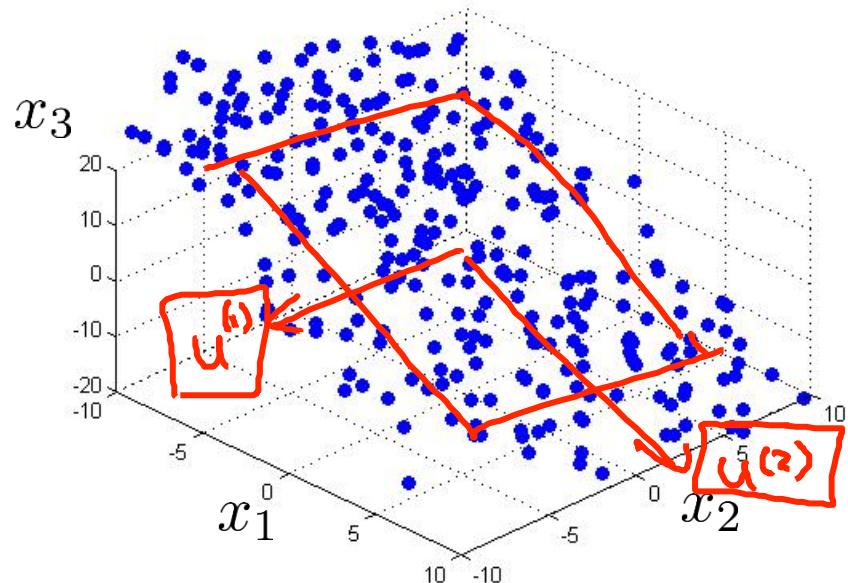
# Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D

$$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \underline{\mathbb{R}}$$

Diagram illustrating the projection of data points onto the first principal component. Data points  $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}$  are shown as black 'x' marks. Their projections onto the line, labeled  $t_{z^{(1)}}(x)$ , are shown as red 'x' marks. A red box labeled  $z^{(1)}$  represents the resulting 1D vector.



Reduce data from 3D to 2D

$$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$$
$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

# Principal Component Analysis (PCA) algorithm

Reduce data from  $n$ -dimensions to  $\underline{k}$ -dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)}) (x^{(i)})^T$$

$n \times 1$        $1 \times n$

Sigma

Compute "eigenvectors" of matrix  $\Sigma$ :

$$\rightarrow [U, S, V] = \underline{\text{svd}}(\text{Sigma}) ;$$

→ Singular value decomposition  
eig(Sigma)

$n \times n$  matrix.

$$U = \begin{bmatrix} | & | & | & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \dots & u^{(m)} \\ | & | & | & & | \end{bmatrix}$$

$k$

$U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \dots, u^{(k)}$

# Principal Component Analysis (PCA) algorithm

From  $[U, S, V] = \text{svd}(\Sigma)$ , we get:

$$\rightarrow U = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$\underbrace{\phantom{u^{(1)} \quad u^{(2)} \quad \dots \quad u^{(n)}}_k}$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

$$z \in \mathbb{R}^k \quad z^{(i)} = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T$$

$\underbrace{\phantom{u^{(1)} \quad u^{(2)} \quad \dots \quad u^{(k)}}_{n \times k}}$

U<sub>reduce</sub>

$$x^{(i)} = \begin{bmatrix} (u^{(1)})^T \\ \vdots \\ (u^{(k)})^T \end{bmatrix} \quad \underbrace{\phantom{(u^{(1)})^T \quad \vdots \quad (u^{(k)})^T}_{k \times n}}_{k \times 1} \quad \underbrace{\phantom{x^{(i)}}_{n \times 1}}$$

# Principal Component Analysis (PCA) algorithm summary

- After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

$$X = \begin{bmatrix} \vdots & & \vdots \\ x^{(1)\top} & \cdots & x^{(m)\top} \end{bmatrix}$$
$$\text{Sigma} = (1/m) * X' * X;$$

$$\rightarrow [U, S, V] = \text{svd}(\text{Sigma});$$

$$\rightarrow U_{\text{reduce}} = U(:, 1:k);$$

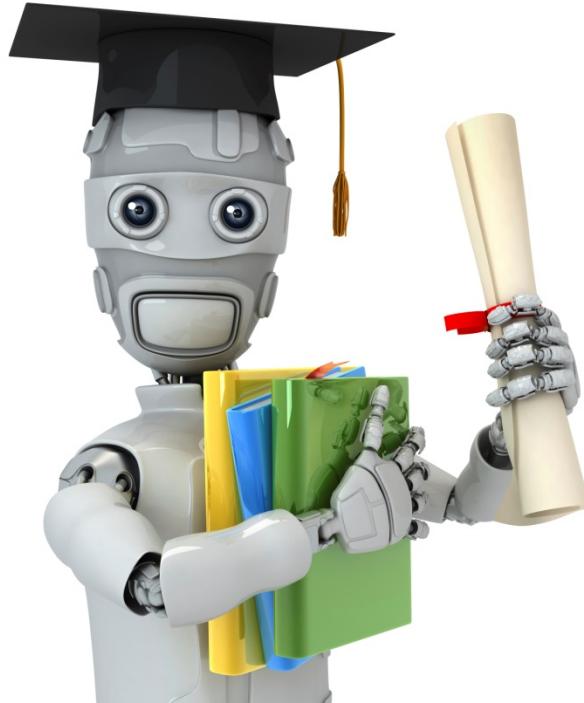
$$\rightarrow z = U_{\text{reduce}}' * x;$$

↑

↑

$$x \in \mathbb{R}^n$$

$$x \neq 1$$



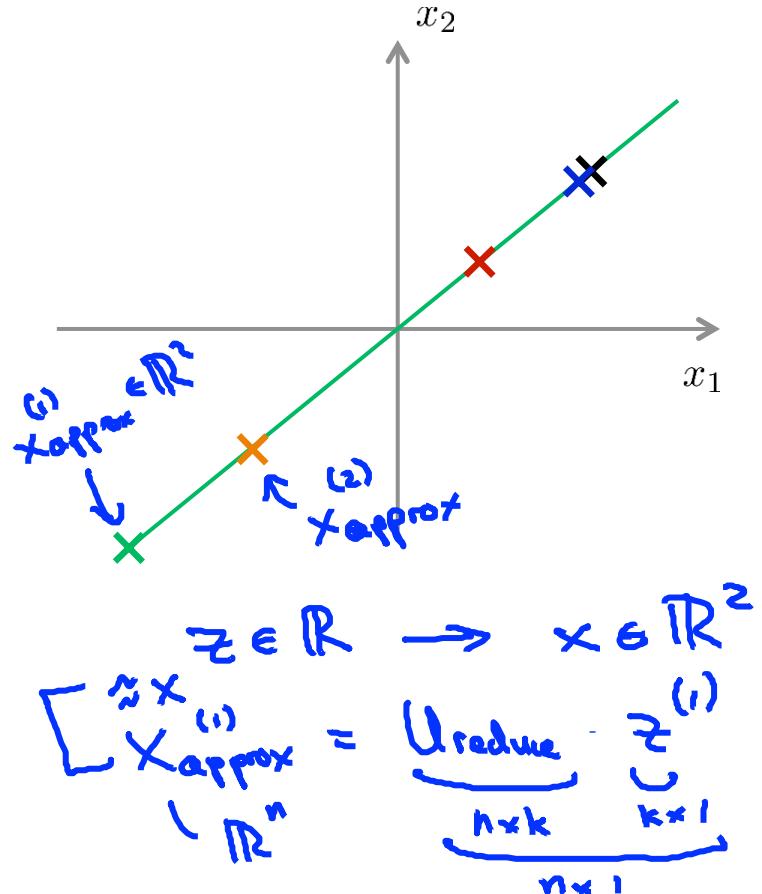
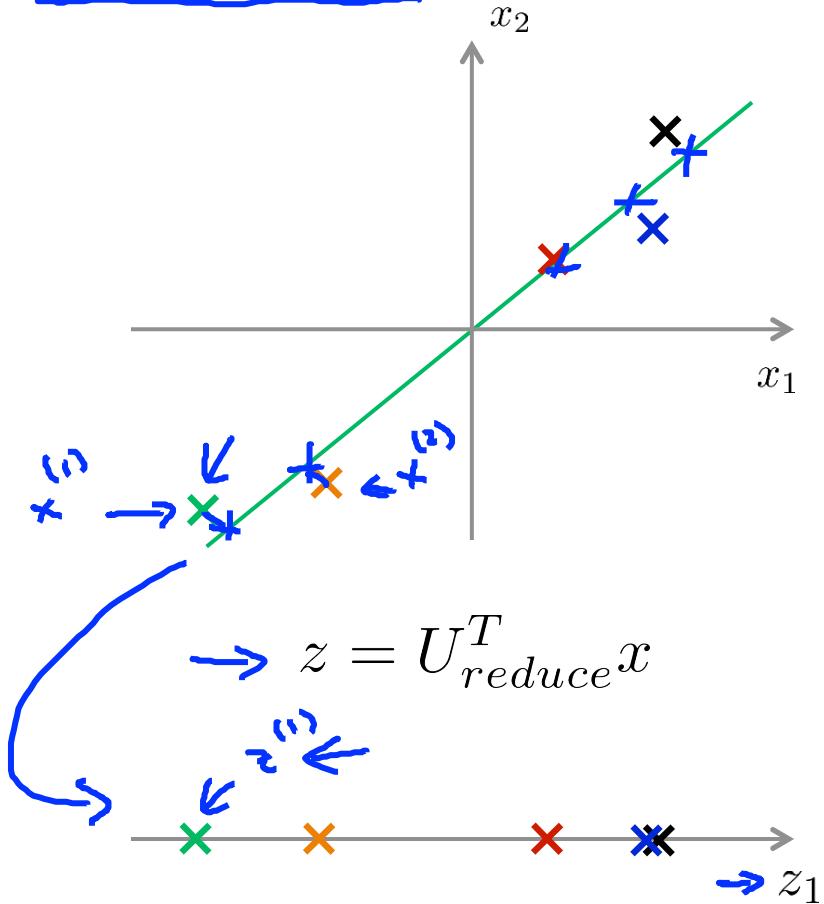
Machine Learning

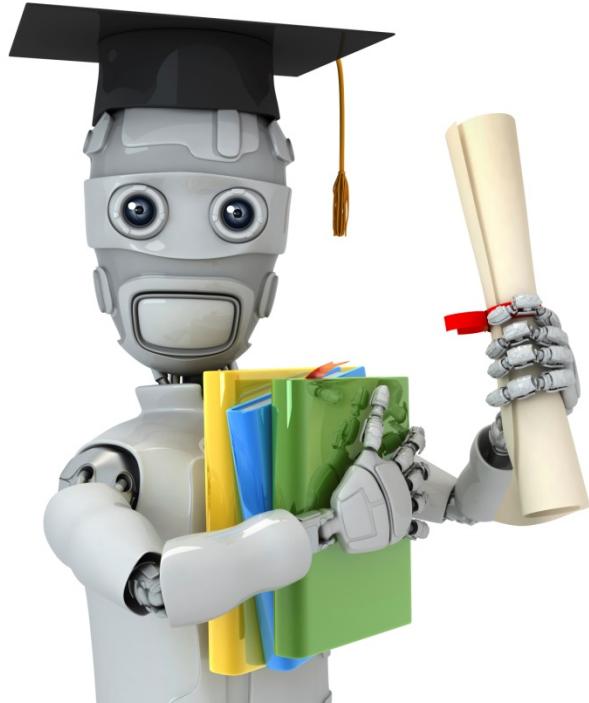
# Dimensionality Reduction

---

Reconstruction from  
compressed  
representation

## Reconstruction from compressed representation





Machine Learning

# Dimensionality Reduction

---

Choosing the number of principal components

## Choosing $k$ (number of principal components)

Average squared projection error:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose  $k$  to be smallest value so that

$$\rightarrow \frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

$$\frac{(1\%)}{\frac{0.05}{5\%}} \quad (10\%)$$

$\rightarrow$  "99% of variance is retained"  
~~95%~~ 90%

# Choosing $k$ (number of principal components)

Algorithm:

Try PCA with  $k = 1$   $\xrightarrow{k=2}$   $\xrightarrow{k=3}$   $\xrightarrow{k=4}$  ...

Compute  $U_{reduce}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

$$\rightarrow [U, S, V] = svd(\Sigma)$$

$$\rightarrow \Sigma =$$

For given  $k$

$$1 - \frac{\sum_{i=1}^k \sigma_{ii}}{\sum_{i=1}^m \sigma_{ii}} \leq 0.01$$

$$\rightarrow \frac{\sum_{i=1}^k \sigma_{ii}}{\sum_{i=1}^m \sigma_{ii}} \geq 0.99$$

# Choosing $k$ (number of principal components)

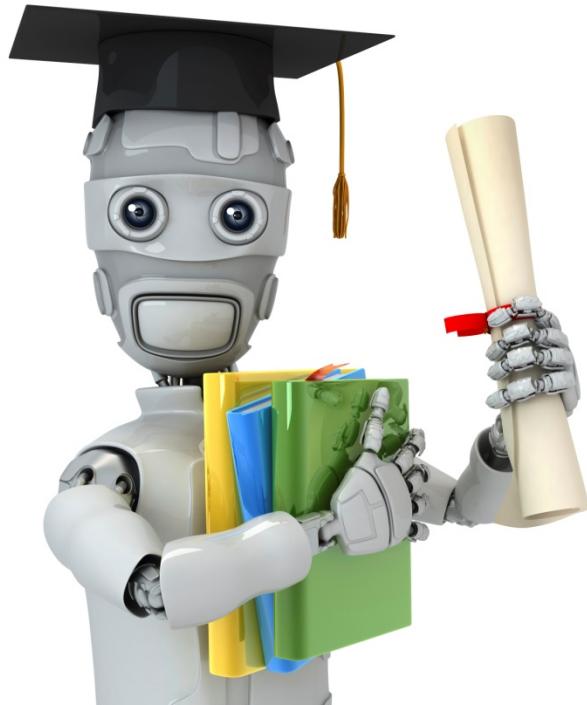
→  $[U, S, V] = \text{svd}(\Sigma)$

Pick smallest value of  $k$  for which

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

$k=100$

(99% of variance retained)



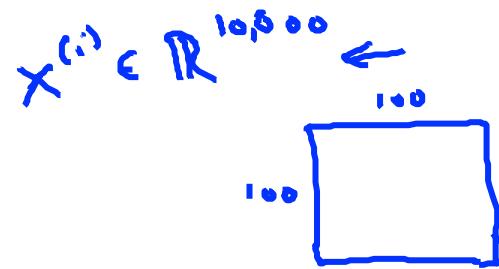
Machine Learning

# Dimensionality Reduction

---

## Advice for applying PCA

## Supervised learning speedup



→  $(\underline{x}^{(1)}, y^{(1)}), (\underline{x}^{(2)}, y^{(2)}), \dots, (\underline{x}^{(m)}, y^{(m)})$

Extract inputs:

Unlabeled dataset:  $\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(m)} \in \mathbb{R}^{10000}$

↓ PCA

$\underline{z}^{(1)}, \underline{z}^{(2)}, \dots, \underline{z}^{(m)} \in \mathbb{R}^{1000}$

$U_{reduce}$

New training set:

$(\underline{z}^{(1)}, y^{(1)}), (\underline{z}^{(2)}, y^{(2)}), \dots, (\underline{z}^{(m)}, y^{(m)})$

$$h_\theta(z) = \frac{1}{1 + e^{-\theta^T z}}$$

Note: Mapping  $x^{(i)} \rightarrow z^{(i)}$  should be defined by running PCA

only on the training set. This mapping can be applied as well to the examples  $x_{cv}^{(i)}$  and  $x_{test}^{(i)}$  in the cross validation and test sets

# Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm ←

Choose  $k$  by % of variance retain

- Visualization

$k=2$  or  $k=3$

## Bad use of PCA: To prevent overfitting

→ Use  $z^{(i)}$  instead of  $x^{(i)}$  to reduce the number of features to  $k < n$ .

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

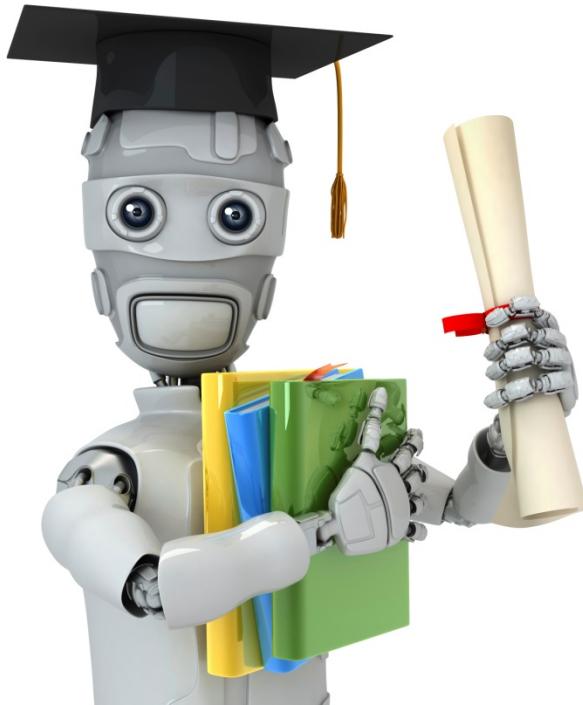
$$\rightarrow \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2}$$

## PCA is sometimes used where it shouldn't be

Design of ML system:

- - Get training set  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- - ~~Run PCA to reduce  $x^{(i)}$  in dimension to get  $z^{(i)}$~~
- - Train logistic regression on  $\{(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})\}$
- - Test on test set: Map  $x_{test}^{(i)}$  to  $z_{test}^{(i)}$ . Run  $h_\theta(z)$  on  $\{(z_{test}^{(1)}, y_{test}^{(1)}), \dots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

- How about doing the whole thing without using PCA?
- Before implementing PCA, first try running whatever you want to do with the original/raw data  $x^{(i)}$ . Only if that doesn't do what you want, then implement PCA and consider using  $\underline{z^{(i)}}$ .



Machine Learning

# Anomaly detection

---

Problem  
motivation

## Anomaly detection example

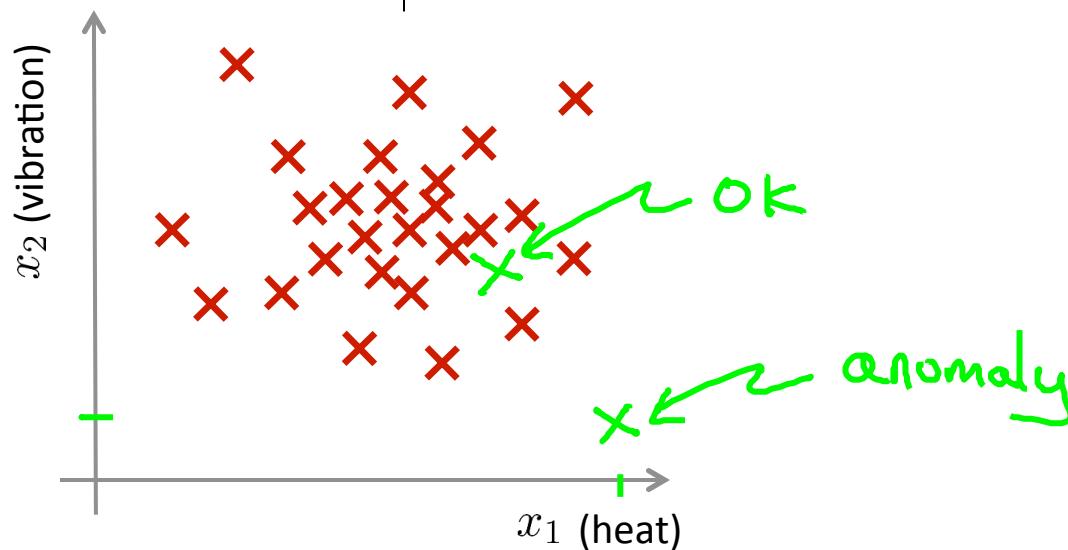
Aircraft engine features:

- $x_1$  = heat generated
- $x_2$  = vibration intensity

...

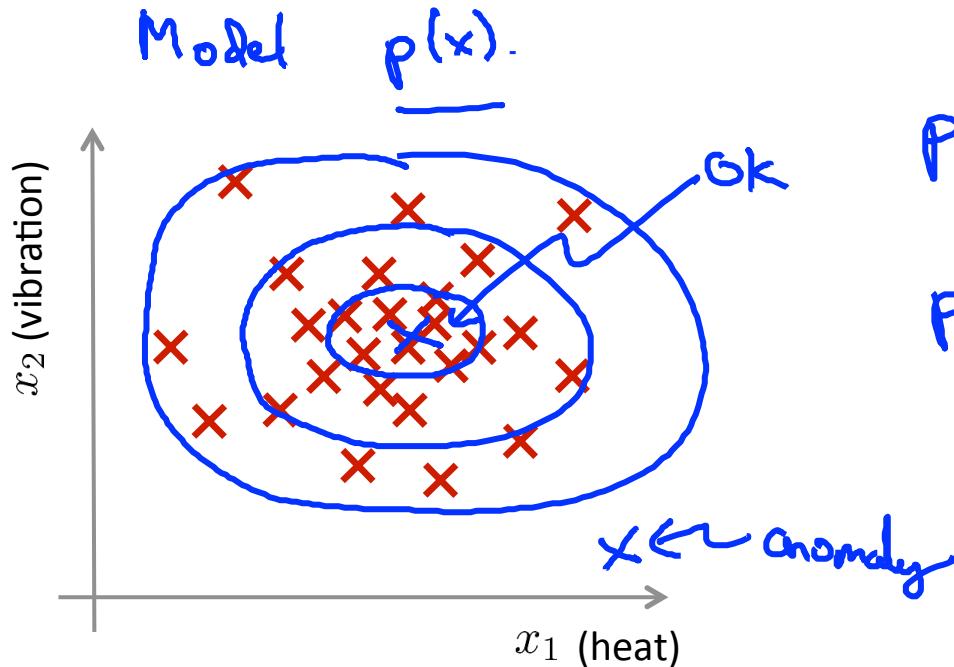
Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

New engine:  $x_{test}$



# Density estimation

- Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
- Is  $x_{test}$  anomalous?



$p(x_{test}) < \varepsilon \rightarrow \text{flag anomaly}$

$p(x_{test}) \geq \varepsilon \rightarrow \text{OK}$

## Anomaly detection example

→ Fraud detection:

→  $x^{(i)}$  = features of user  $i$ 's activities

→ Model  $p(x)$  from data.

→ Identify unusual users by checking which have  $p(x) < \varepsilon$

$$\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \quad p(x)$$

→ Manufacturing

→ Monitoring computers in a data center.

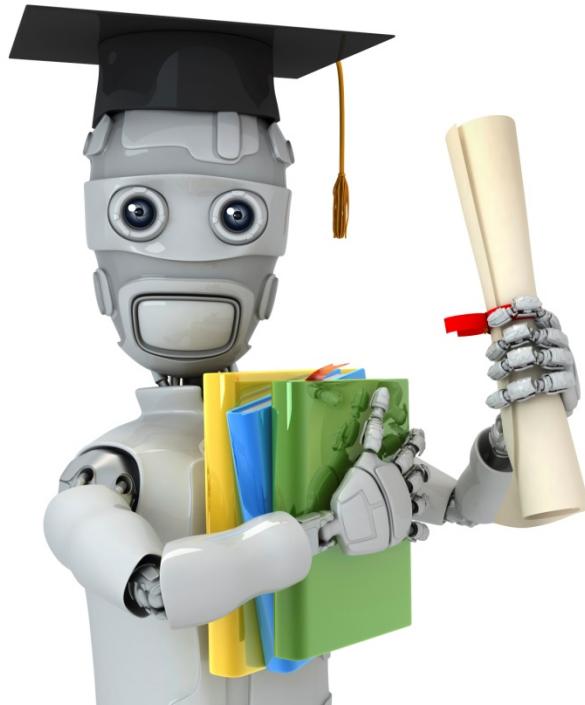
→  $x^{(i)}$  = features of machine  $i$

$x_1$  = memory use,  $x_2$  = number of disk accesses/sec,

$x_3$  = CPU load,  $x_4$  = CPU load/network traffic.

...

$$p(x) < \varepsilon$$



Machine Learning

# Anomaly detection

---

## Gaussian distribution

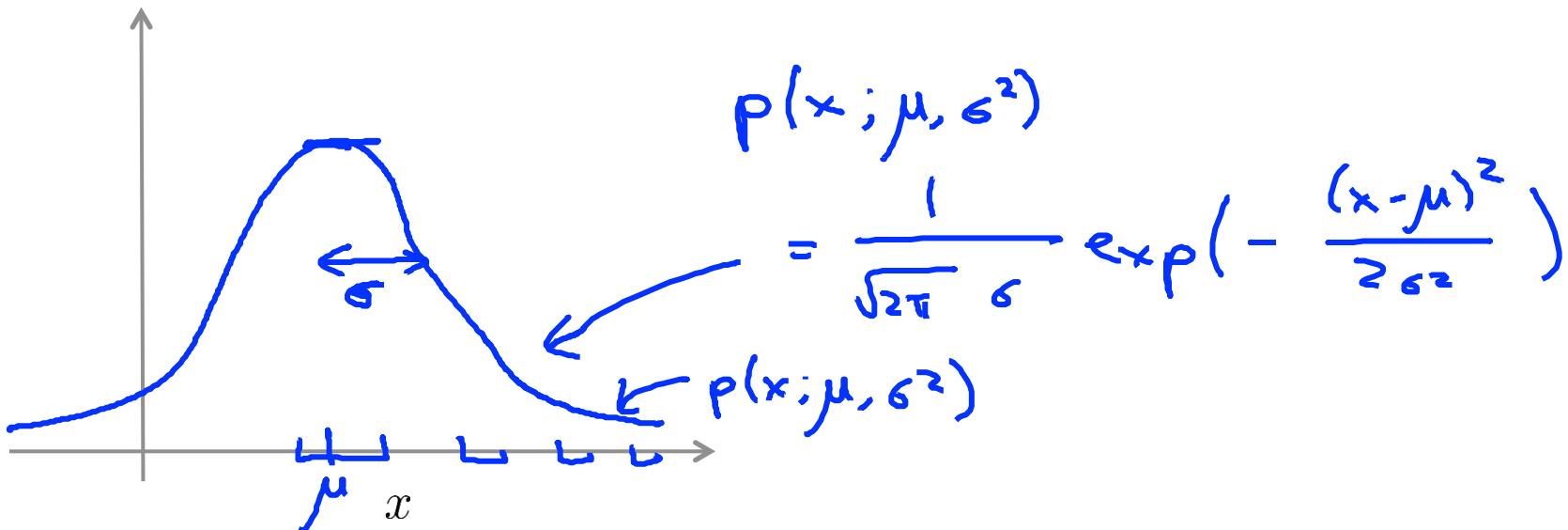
## Gaussian (Normal) distribution

Say  $x \in \mathbb{R}$ . If  $x$  is a distributed Gaussian with mean  $\mu$ , variance  $\sigma^2$ .

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

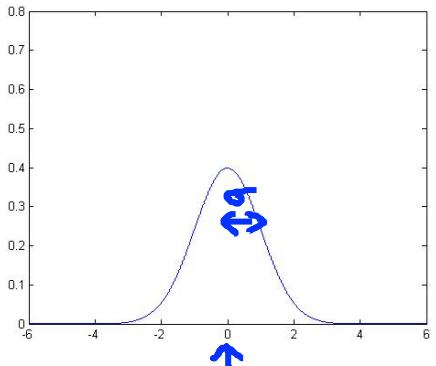
↖ "distributed as"

$\sigma$  standard deviation

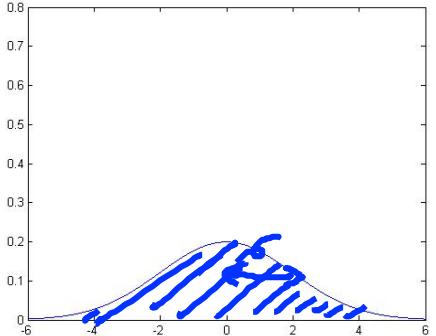


# Gaussian distribution example

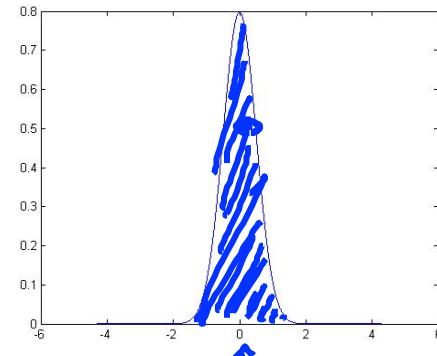
$$\rightarrow \mu = 0, \sigma = 1$$



$$\rightarrow \mu = 0, \sigma = 2$$

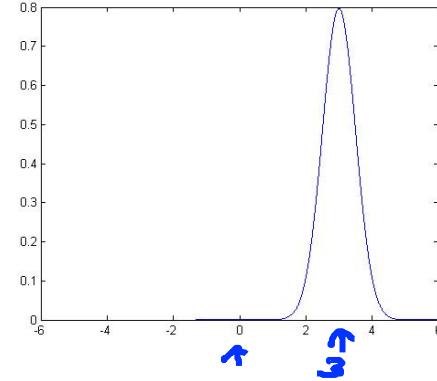


$$\rightarrow \mu = 0, \sigma = 0.5$$



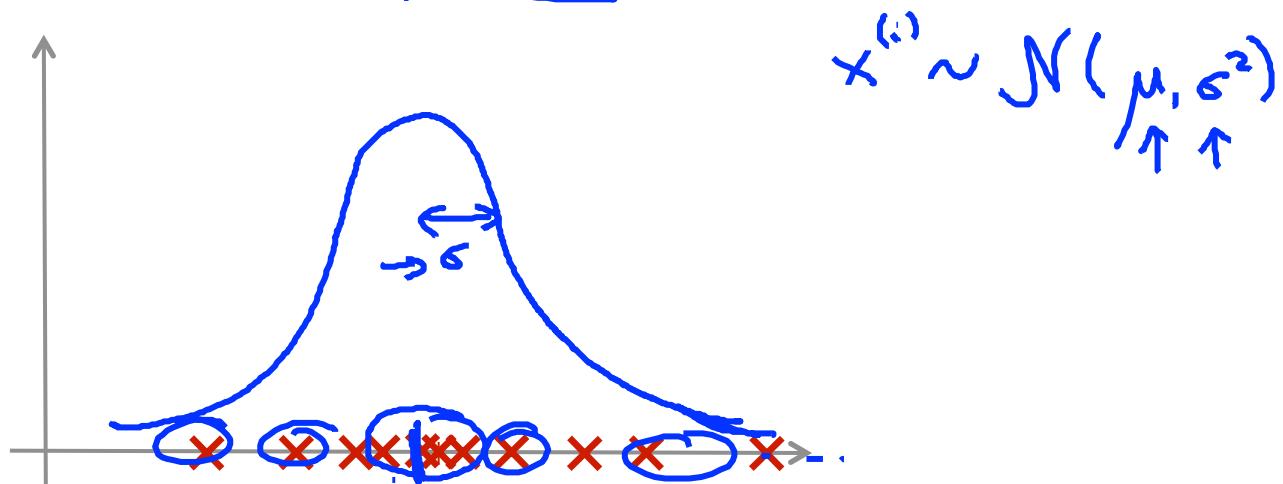
$$\zeta^2 = 0.25$$

$$\rightarrow \mu = 3, \sigma = 0.5$$



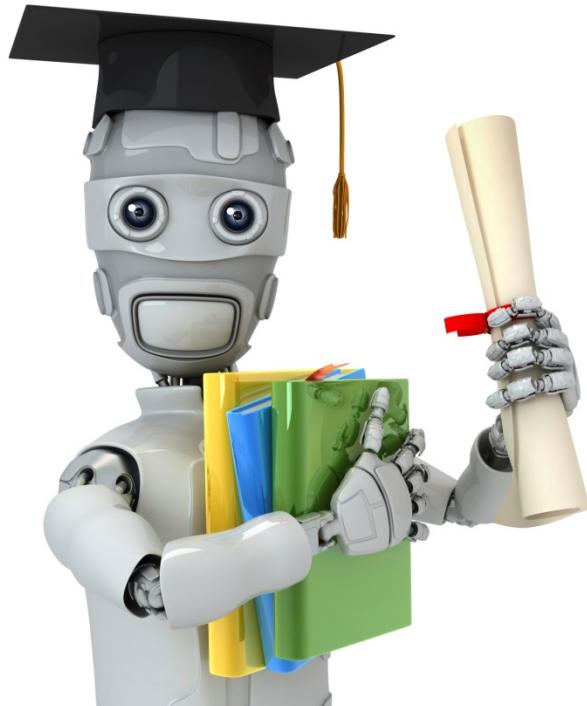
## Parameter estimation

→ Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$   $x^{(i)} \in \mathbb{R}$



$$\Rightarrow \hat{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\Rightarrow \hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2$$



Machine Learning

# Anomaly detection

---

# Algorithm

## → Density estimation

→ Training set:  $\{x^{(1)}, \dots, x^{(m)}\}$

Each example is  $x \in \mathbb{R}^n$

→  $p(x)$

$$= \boxed{p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) p(x_3; \mu_3, \sigma_3^2) \dots p(x_n; \mu_n, \sigma_n^2)}$$

$$= \boxed{\prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)}$$

$$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

$$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$$

$$\sum_{i=1}^n i = 1+2+3+\dots+n$$

$$\prod_{i=1}^n i = 1 \times 2 \times 3 \times \dots \times n$$

# Anomaly detection algorithm

- 1. Choose features  $x_i$  that you think might be indicative of anomalous examples.

$$\{x^{(1)}, \dots, x^{(m)}\}$$

- 2. Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\rightarrow \boxed{\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}}$$

$$p(x_j; \mu_j, \sigma_j^2)$$

$$\rightarrow \boxed{\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2}$$

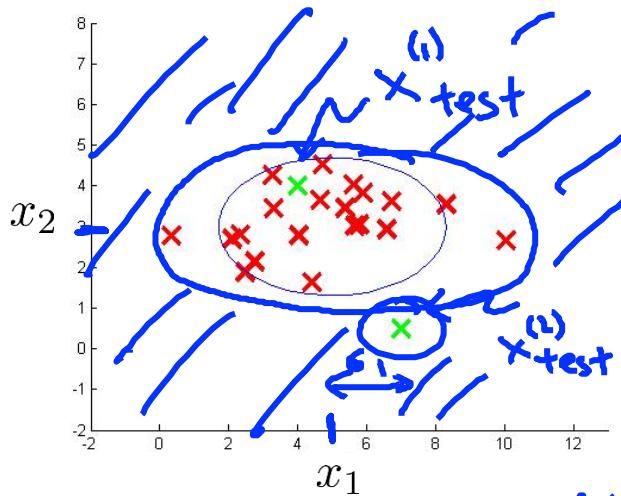
$$\mu_1, \mu_2, \dots, \mu_n$$
$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

- 3. Given new example  $x$ , compute  $p(x)$ :

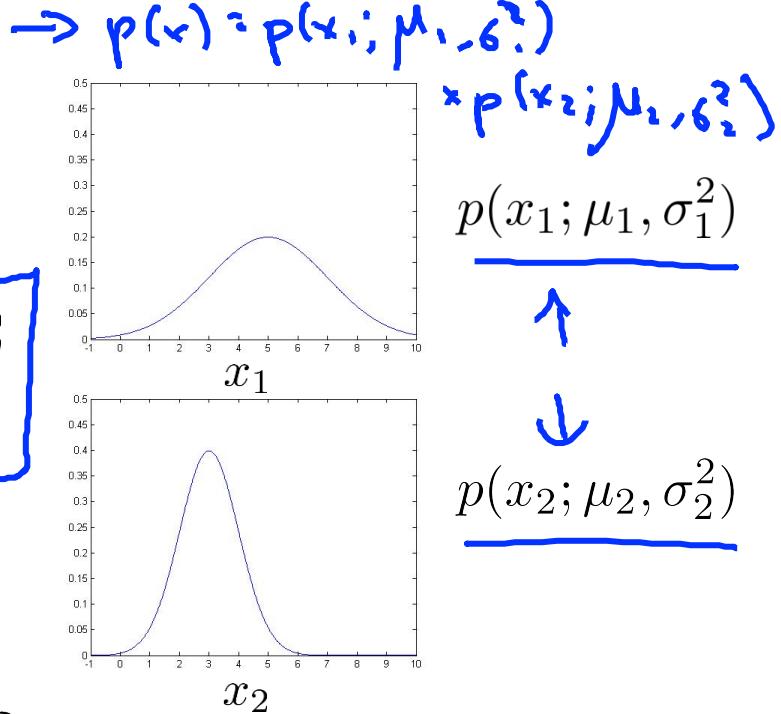
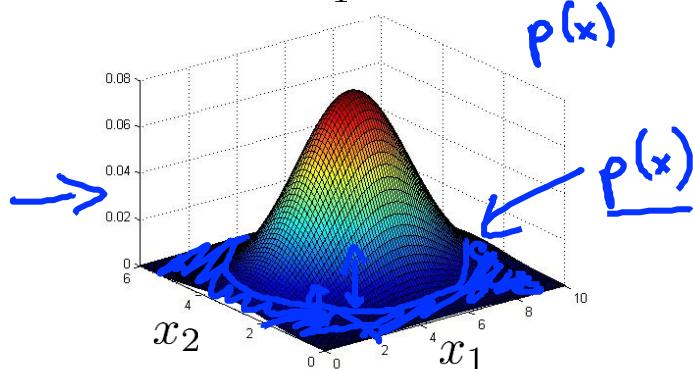
$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $\underline{p(x) < \varepsilon}$

# Anomaly detection example

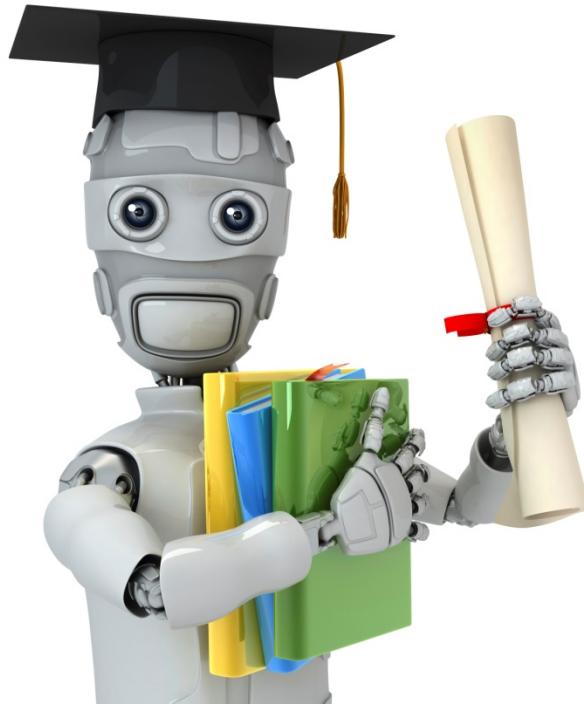


$$\begin{aligned} \mu_1 &= 5, \underline{\sigma_1} = 2 \\ \mu_2 &= 3, \underline{\sigma_2} = 1 \end{aligned}$$



$$\underline{\varepsilon = 0.02}$$

$$\begin{aligned} p(x_{test}^{(1)}) &= 0.0426 &> \underline{\varepsilon} \\ p(x_{test}^{(2)}) &= 0.0021 &< \underline{\varepsilon} \end{aligned}$$



Machine Learning

# Anomaly detection

---

Developing and  
evaluating an anomaly  
detection system

## The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

- Assume we have some labeled data, of anomalous and non-anomalous examples. ( $y = 0$  if normal,  $y = 1$  if anomalous).
- Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)
- Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
- Test set:  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

$$y=1$$

## Aircraft engines motivating example

- 10000 good (normal) engines
- 20 flawed engines (anomalous) 2 - 50 y = 1
- Training set: 6000 good engines ( $y = 0$ )  $p(x) = p(x_1; \mu_1, \sigma^2_1) \dots p(x_n; \mu_n, \sigma^2_n)$
- CV: 2000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )
- Test: 2000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

Alternative:

Training set: 6000 good engines

→ CV: 4000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

→ Test: 4000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

## Algorithm evaluation

- Fit model  $p(x)$  on training set  $\{x^{(1)}, \dots, x^{(m)}\}$
- On a cross validation/test example  $x$ , predict

$(x_{\text{test}}^{(i)}, y_{\text{test}}^{(i)})$



$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

$y=0$

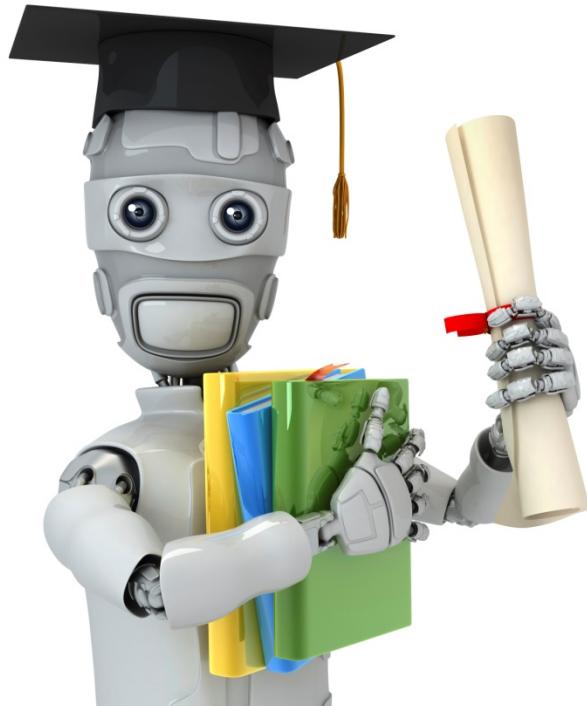
Possible evaluation metrics:

- - True positive, false positive, false negative, true negative
- - Precision/Recall
- -  $F_1$ -score

CV

Test set

Can also use cross validation set to choose parameter  $\varepsilon$



Machine Learning

# Anomaly detection

---

Anomaly detection  
vs. supervised  
learning

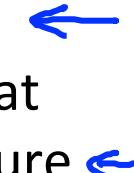
## Anomaly detection

vs.

## Supervised learning

- Very small number of positive examples ( $y = 1$ ). (0-20 is common).
- Large number of negative ( $y = 0$ ) examples. 
- Many different “types” of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;
- future anomalies may look nothing like any of the anomalous examples we've seen so far.

Large number of positive and negative examples. 

Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set. 

Spam 

## Anomaly detection

- • Fraud detection  $y=1$
- • Manufacturing (e.g. aircraft engines)
- • Monitoring machines in a data center

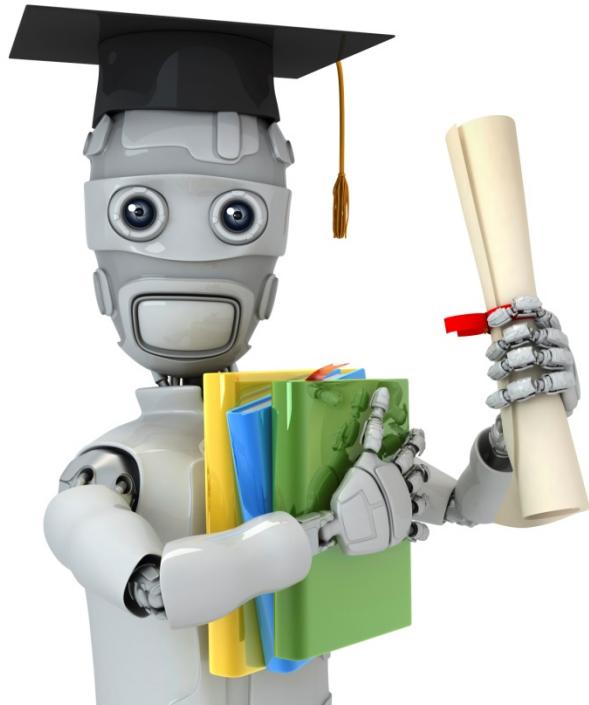
vs.

## Supervised learning

- Email spam classification ←
- Weather prediction (sunny/rainy/etc).
- Cancer classification ←

:

:



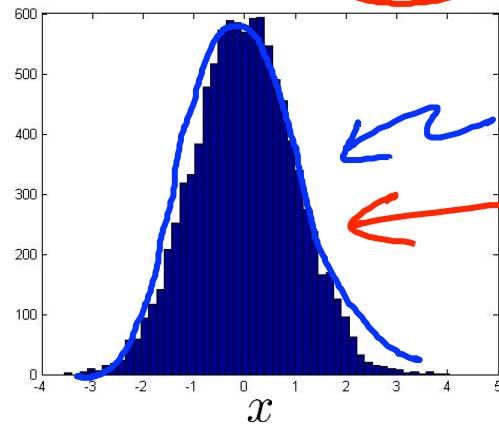
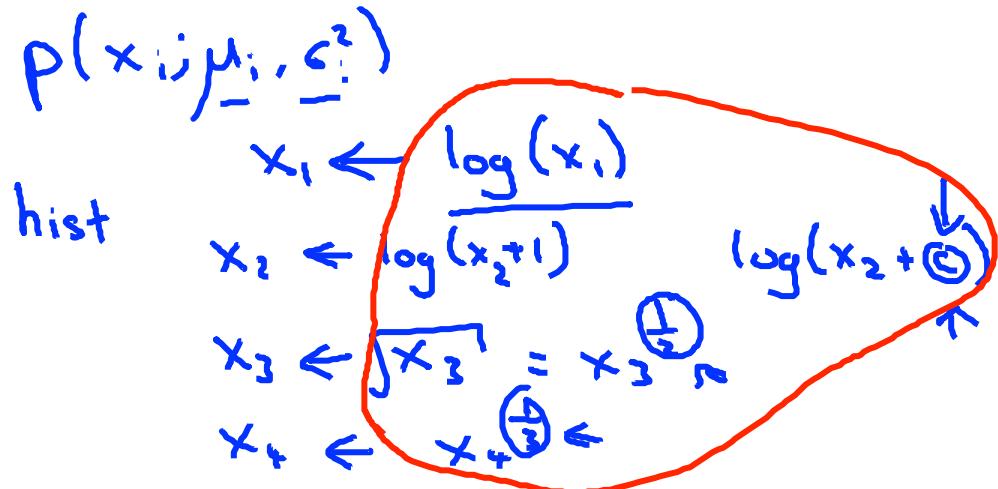
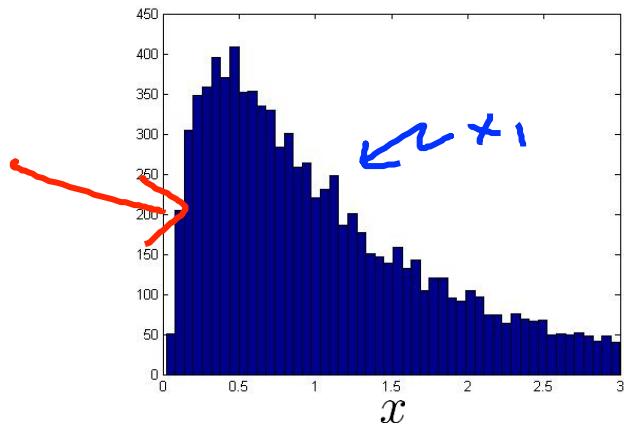
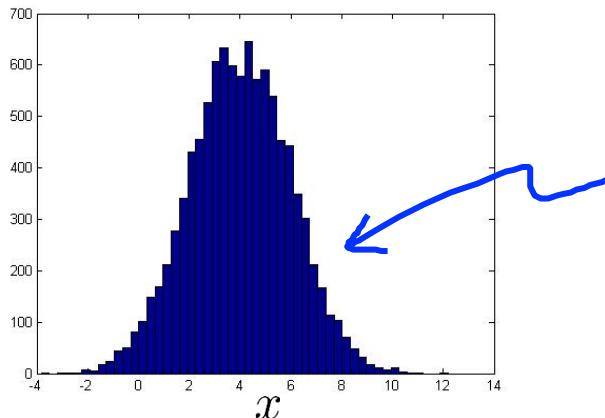
Machine Learning

# Anomaly detection

---

Choosing what features to use

# Non-gaussian features

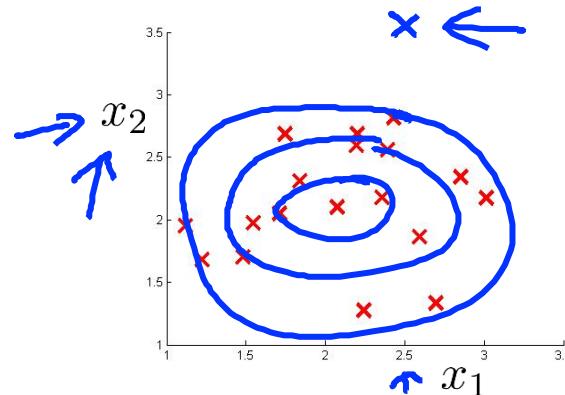
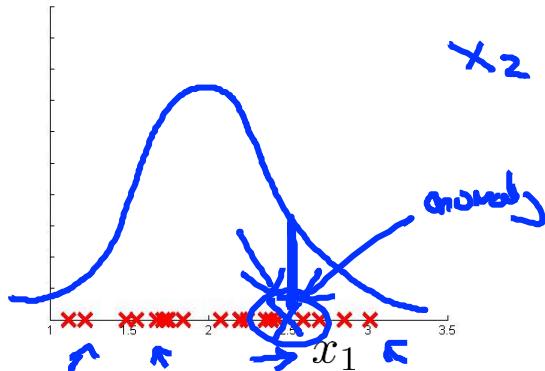


## → Error analysis for anomaly detection

Want  $p(x)$  large for normal examples  $x$ .  
 $p(x)$  small for anomalous examples  $x$ .

Most common problem:

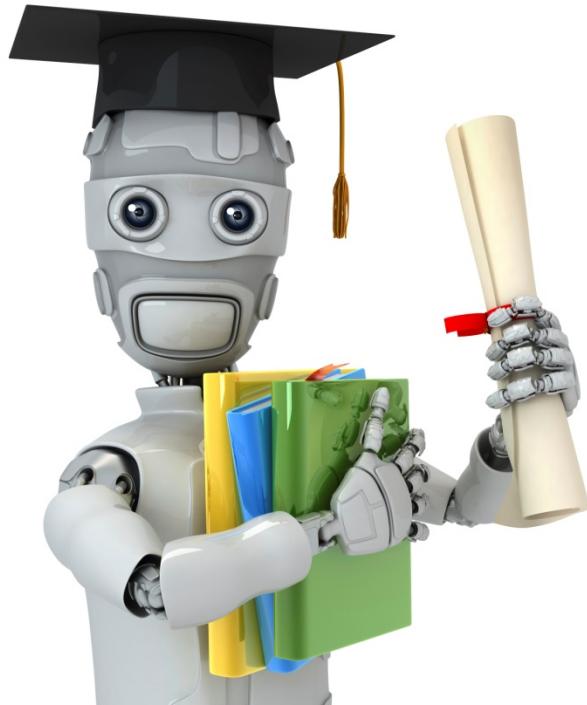
$p(x)$  is comparable (say, both large) for normal and anomalous examples



- Monitoring computers in a data center
- Choose features that might take on unusually large or small values in the event of an anomaly.
  - $x_1$  = memory use of computer
  - $x_2$  = number of disk accesses/sec
  - $x_3$  = CPU load ←
  - $x_4$  = network traffic ←

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

$$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$



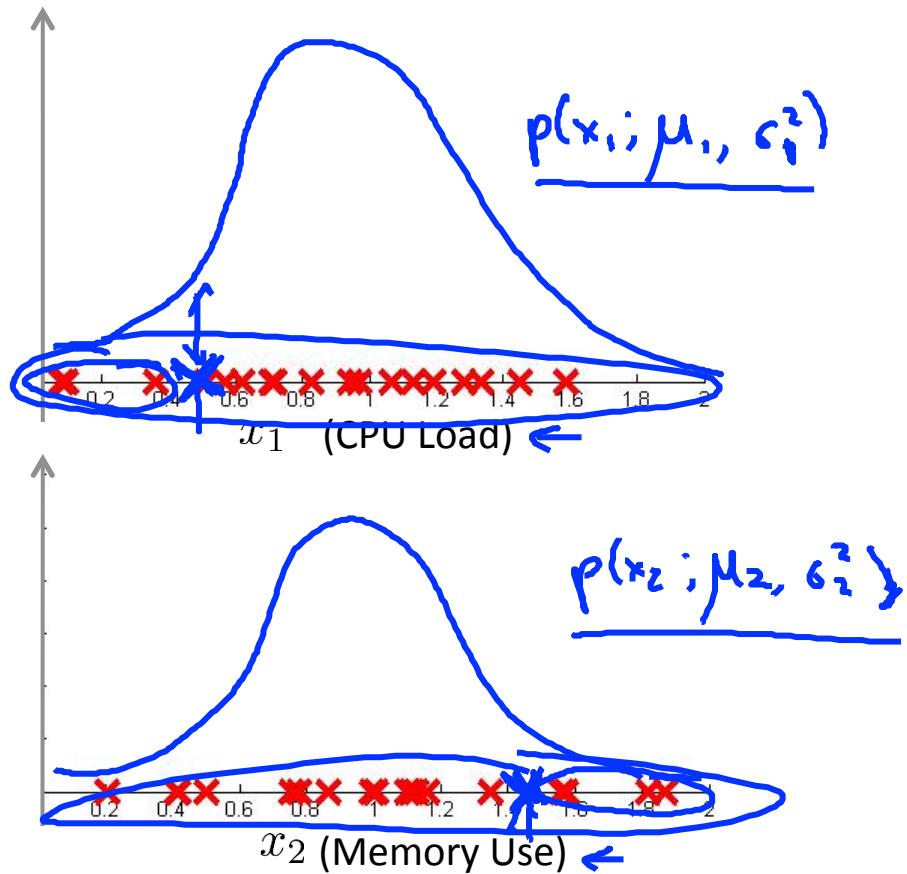
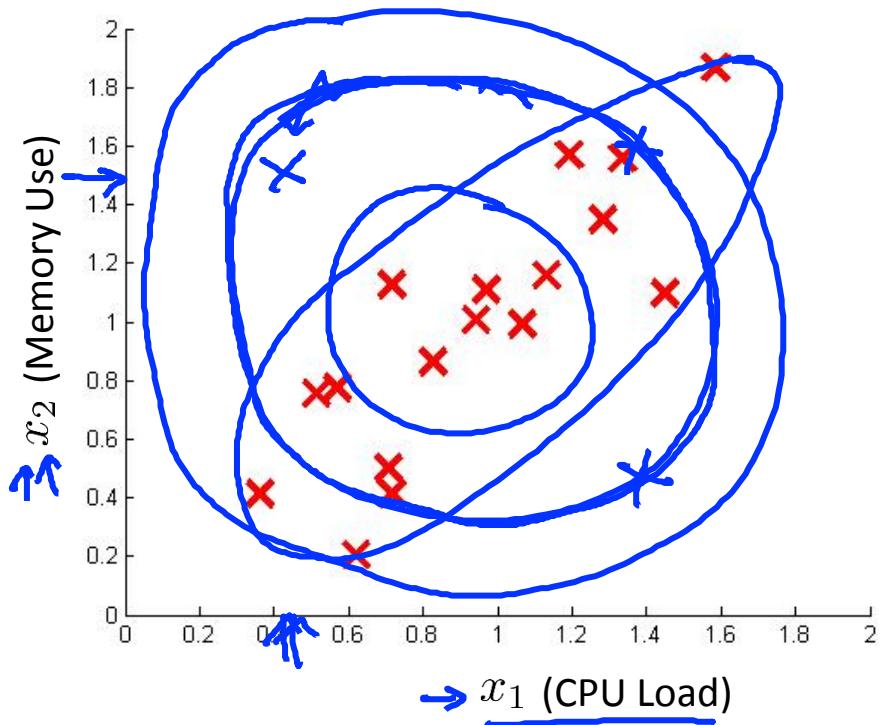
Machine Learning

# Anomaly detection

---

Multivariate  
Gaussian distribution

# Motivating example: Monitoring machines in a data center



# Multivariate Gaussian (Normal) distribution

→  $x \in \mathbb{R}^n$ . Don't model  $p(x_1), p(x_2), \dots$ , etc. separately.  
Model  $p(x)$  all in one go.  
Parameters:  $\mu \in \mathbb{R}^n$ ,  $\Sigma \in \mathbb{R}^{n \times n}$  (covariance matrix)

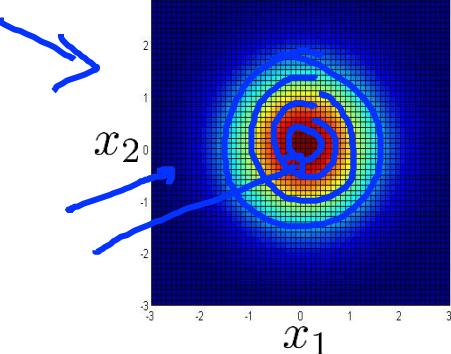
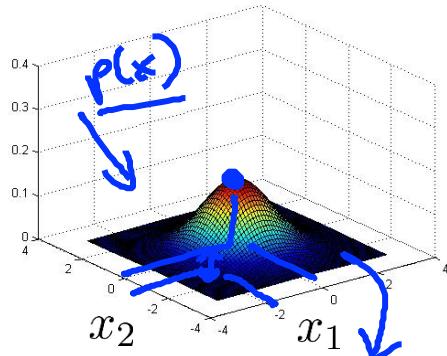
$$p(x; \mu, \Sigma) =$$

$$\frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

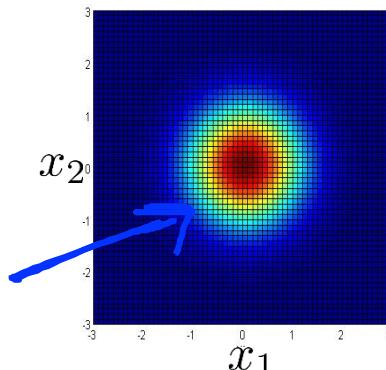
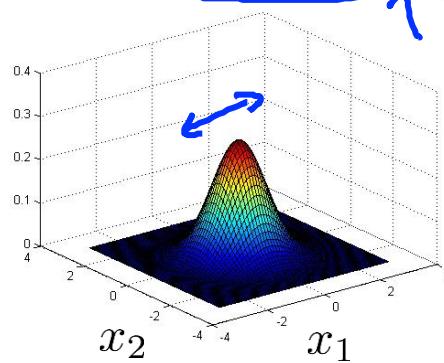
$|\Sigma| = \text{determinant of } \Sigma \quad | \det(\text{Sigma})$

# Multivariate Gaussian (Normal) examples

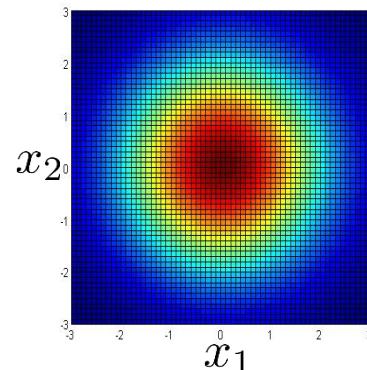
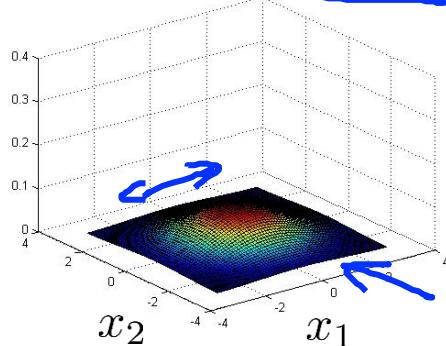
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

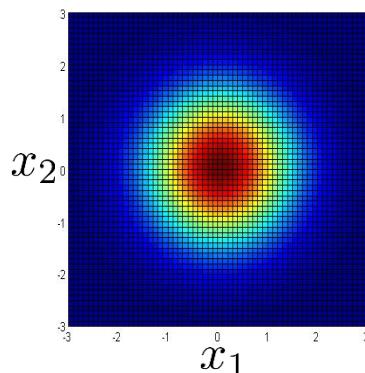
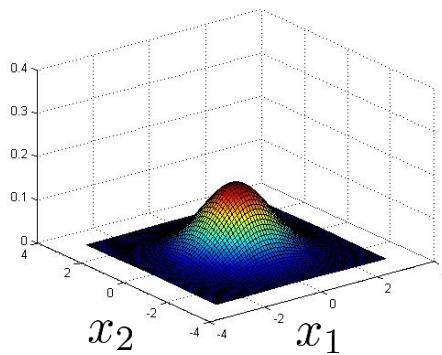


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

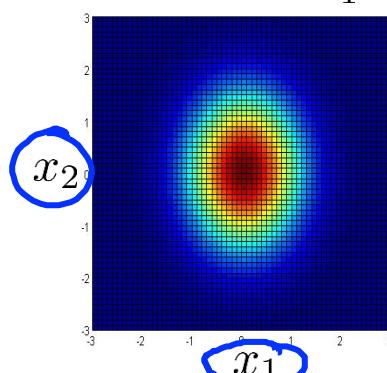
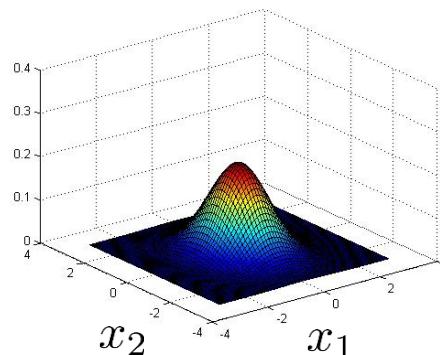


# Multivariate Gaussian (Normal) examples

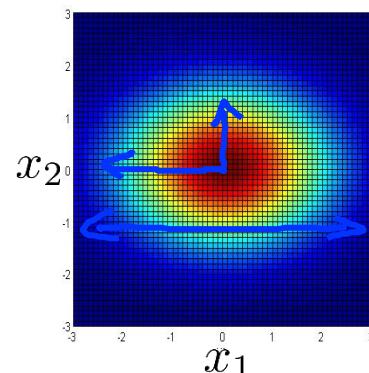
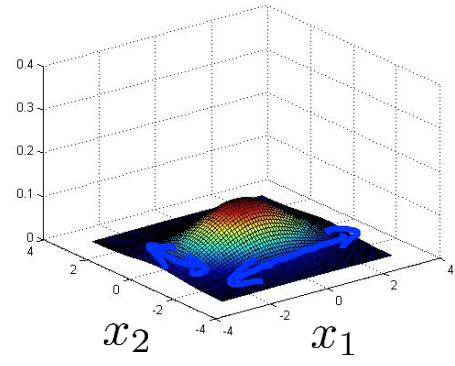
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$$

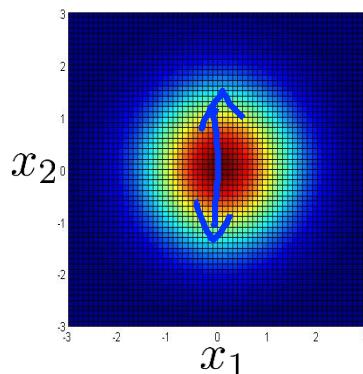
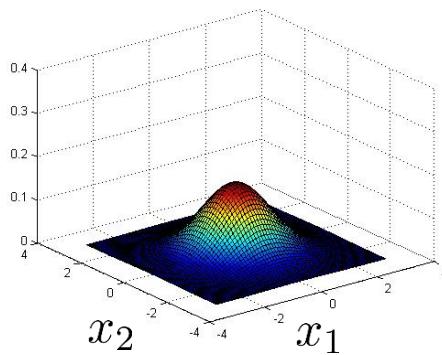


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

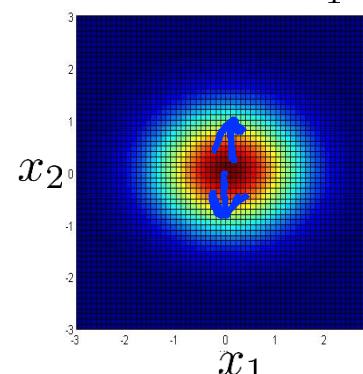
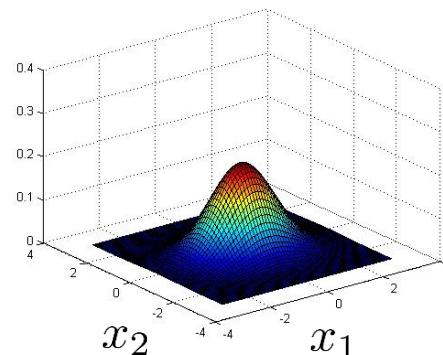


# Multivariate Gaussian (Normal) examples

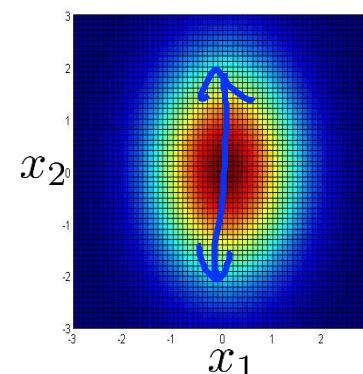
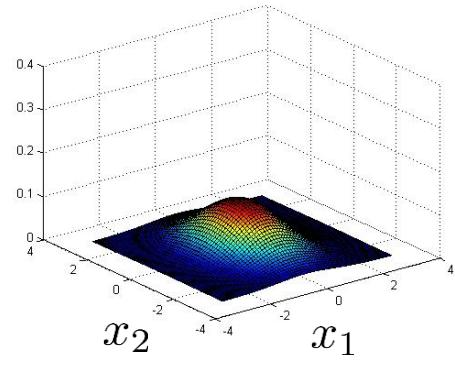
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$$

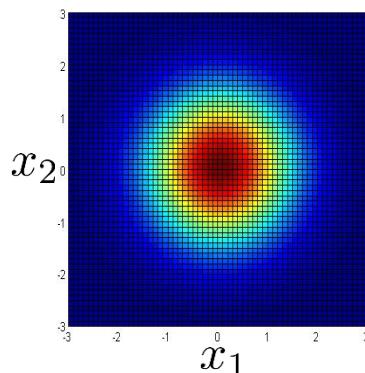
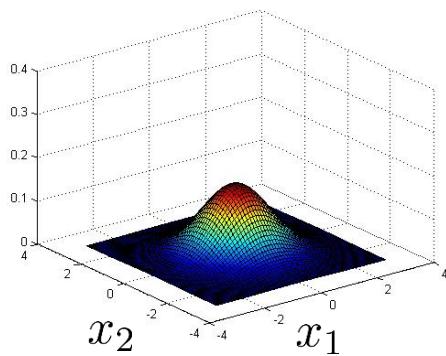


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

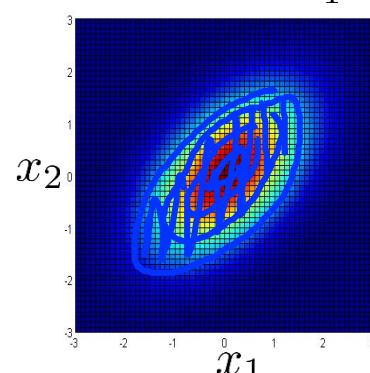
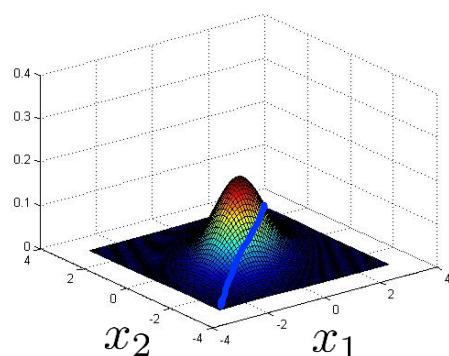


# Multivariate Gaussian (Normal) examples

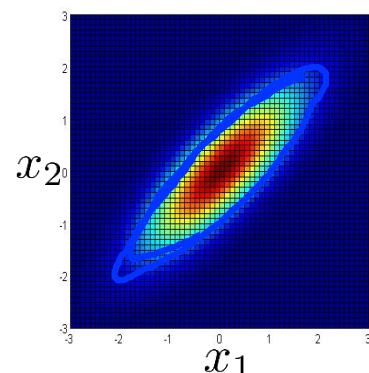
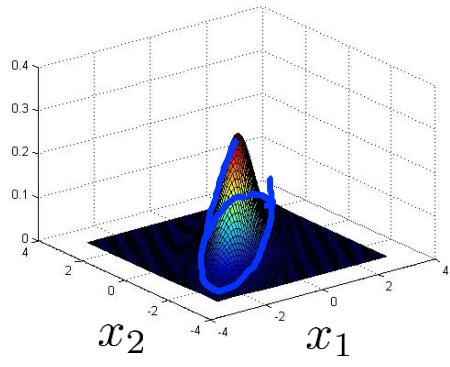
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

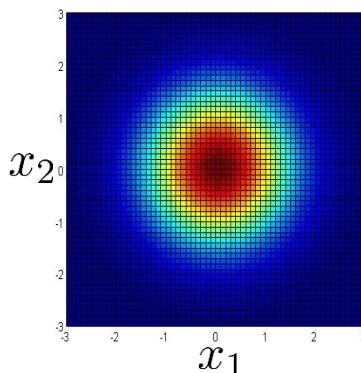
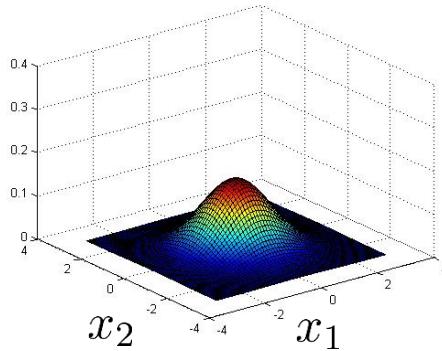


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

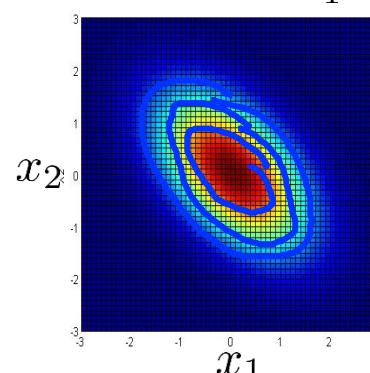
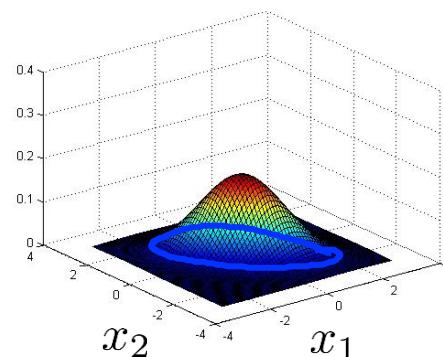


# Multivariate Gaussian (Normal) examples

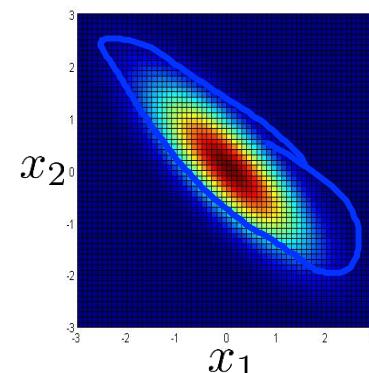
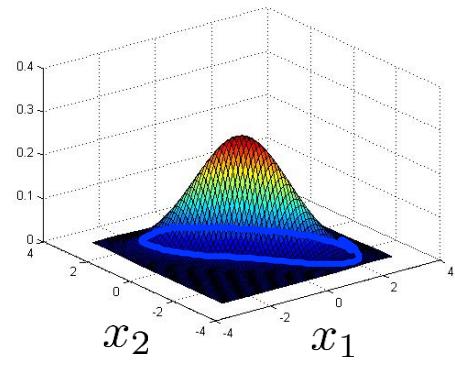
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

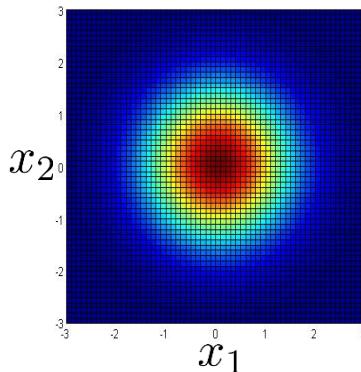
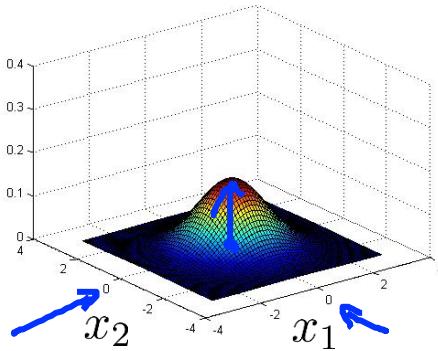


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$

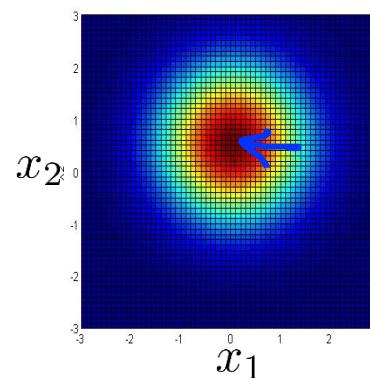
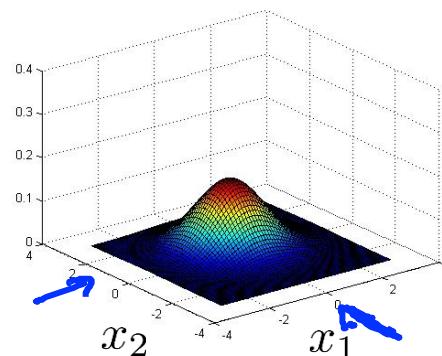


# Multivariate Gaussian (Normal) examples

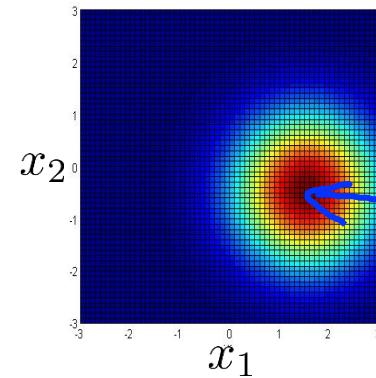
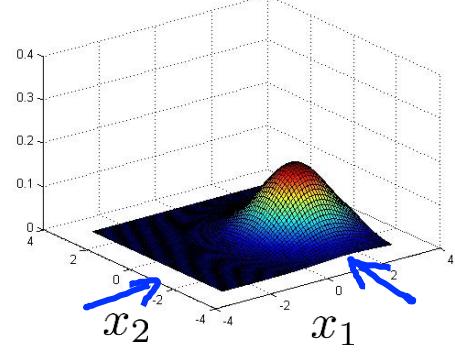
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

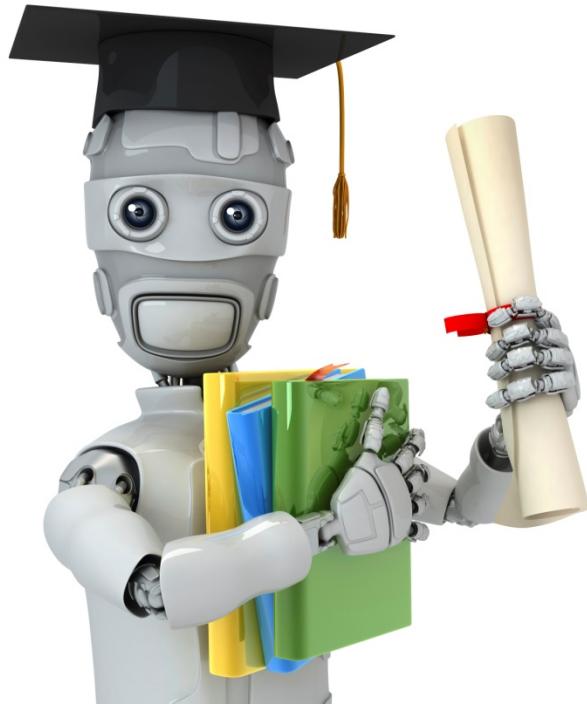


$$\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$





Machine Learning

# Anomaly detection

---

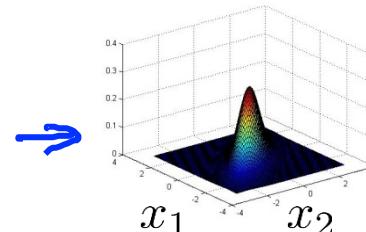
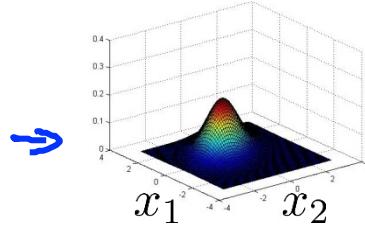
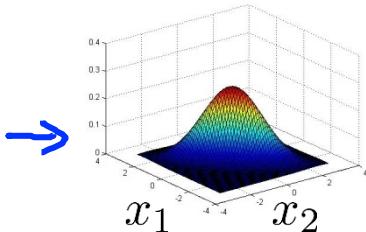
Anomaly detection using  
the multivariate  
Gaussian distribution

# Multivariate Gaussian (Normal) distribution

Parameters  $\mu, \Sigma$

$$\mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$$

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$



Parameter fitting:

Given training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$x \in \mathbb{R}^n$$

$$\rightarrow \boxed{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\rightarrow \boxed{\Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

# Anomaly detection with the multivariate Gaussian

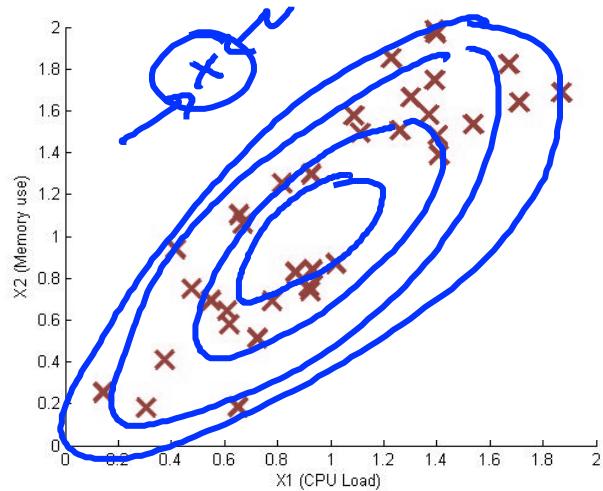
1. Fit model  $p(x)$  by setting

$$\left[ \begin{array}{l} \mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \\ \Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T \end{array} \right]$$

2. Given a new example  $x$ , compute

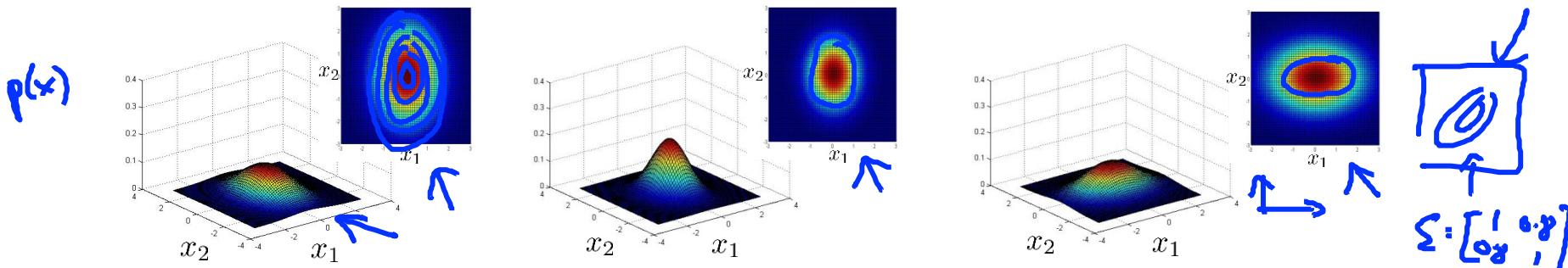
$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

Flag an anomaly if  $\underline{p(x) < \varepsilon}$



## Relationship to original model

Original model:  $p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$



Corresponds to multivariate Gaussian

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

where

$$\Sigma = \begin{bmatrix} \dots & & \\ & \dots & \\ \vdots & & \end{bmatrix}$$

## → Original model

$$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$$

Manually create features to capture anomalies where  $x_1, x_2$  take unusual combinations of values.

$$\rightarrow x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$$

- Computationally cheaper (alternatively, scales better to large  $n=10,000, n=100,000$ )
  - OK even if  $m$  (training set size) is small

## vs. → Multivariate Gaussian

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

→ Automatically captures correlations between features

$$\Sigma \in \mathbb{R}^{n \times n}$$

$$\underline{\Sigma^{-1}}$$

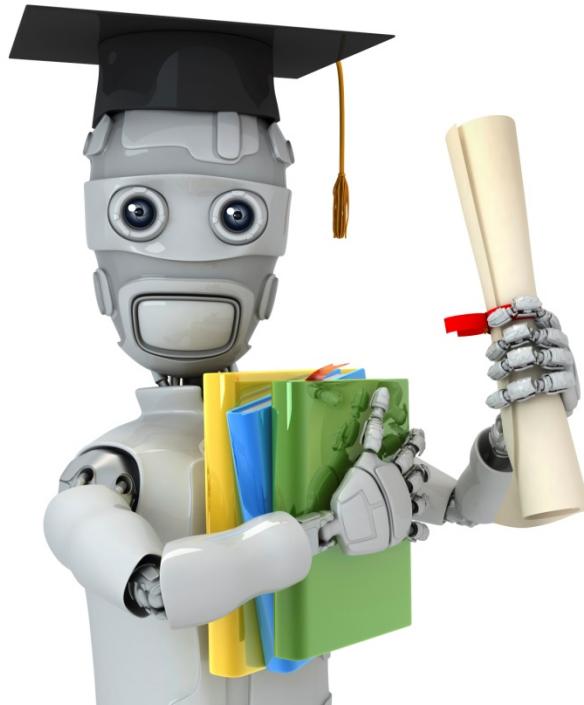
Computationally more expensive

$$\rightarrow \Sigma \sim \frac{n^2}{2}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 + x_5 \\ x_5 \end{bmatrix}$$

Must have  $m > n$  or else  $\Sigma$  is non-invertible.

$$\underline{m \geq n}$$



Machine Learning

# Recommender Systems

---

## Problem formulation

## Example: Predicting movie ratings

→ User rates movies using ~~one to five stars~~  
~~zero~~

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	6
Romance forever	5	?	0	0
Cute puppies of love	?	4	0	?
Nonstop car chases	5	0	5	4
Swords vs. karate	5	0	?	?

$$n_u = 4$$

$$n_m = 5$$

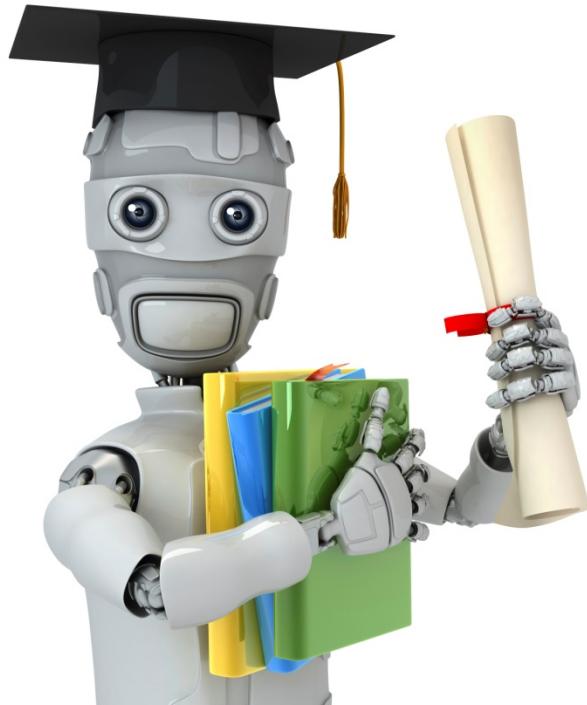


→  $n_u$  = no. users

→  $n_m$  = no. movies

→  $r(i, j) = 1$  if user  $j$  has rated movie  $i$   
 $y^{(i,j)}$  = rating given by user  $j$  to movie  $i$   
(defined only if  $r(i, j) = 1$ )

$$0, \dots, 5$$



Machine Learning

# Recommender Systems

---

Content-based  
recommendations

## Content-based recommender systems

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$n_u = 4, n_m = 5$	$x_0 = 1$	$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$
Love at last	1	5	5	0	0		
Romance forever	2	5	?	?	0		
Cute puppies of love	3	?	4	0	?		
Nonstop car chases	4	0	0	5	4		
Swords vs. karate	5	0	0	5	?		

For each user  $j$ , learn a parameter  $\theta^{(j)} \in \mathbb{R}^3$ . Predict user  $j$  as rating movie  $(\theta^{(j)})^T x^{(i)}$  stars.

$$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \leftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$$

$$(\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$$

## Problem formulation

- $r(i, j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
- $y^{(i,j)}$  = rating by user  $j$  on movie  $i$  (if defined)
- $\theta^{(j)}$  = parameter vector for user  $j$
- $x^{(i)}$  = feature vector for movie  $i$
- For user  $j$ , movie  $i$ , predicted rating:  $(\theta^{(j)})^T(x^{(i)})$
- $m^{(j)}$  = no. of movies rated by user  $j$

To learn  $\theta^{(j)}$ :

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i : r(i,j)=1} \left( (\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$\theta^{(j)} \in \mathbb{R}^{n+1}$$

## Optimization objective:

To learn  $\theta^{(j)}$  (parameter for user  $j$ ):

$$\rightarrow \min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

To learn  $\theta^{(1)}$ ,  $\theta^{(2)}$ , ...,  $\theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$\Theta^{(1)}, \dots, \Theta^{(n_u)}$

## Optimization algorithm:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$J(\theta^{(1)}, \dots, \theta^{(n_u)})$

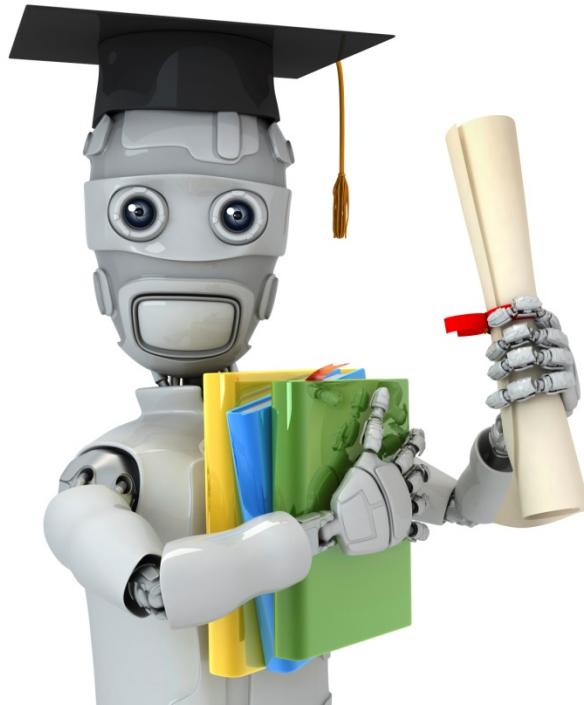
Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

~~$m^{(j)}$~~

$$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$$



Machine Learning

# Recommender Systems

---

## Collaborative filtering

# Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9



# Problem motivation

$x^{(1)}$

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)	$x_o = 1$
<del>Love at last</del>	5	5	0	0	1.0	0.0	
Romance forever	5	?	?	0	?	?	
Cute puppies of love	?	4	0	?	?	?	
Nonstop car chases	0	0	5	4	?	?	
Swords vs. karate	0	0	5	?	?	?	

$x^{(2)}$

$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

$\theta^{(j)}$

$(\theta^{(1)})^T x^{(1)} \approx 5$

$(\theta^{(2)})^T x^{(1)} \approx 5$

$(\theta^{(3)})^T x^{(1)} \approx 0$

$(\theta^{(4)})^T x^{(1)} \approx 0$

$x^N = \begin{bmatrix} 1 \\ 1.0 \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$

$x^{(1)}$

Andrew Ng

# Optimization algorithm

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(i)}$ :

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2$$

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

# Collaborative filtering

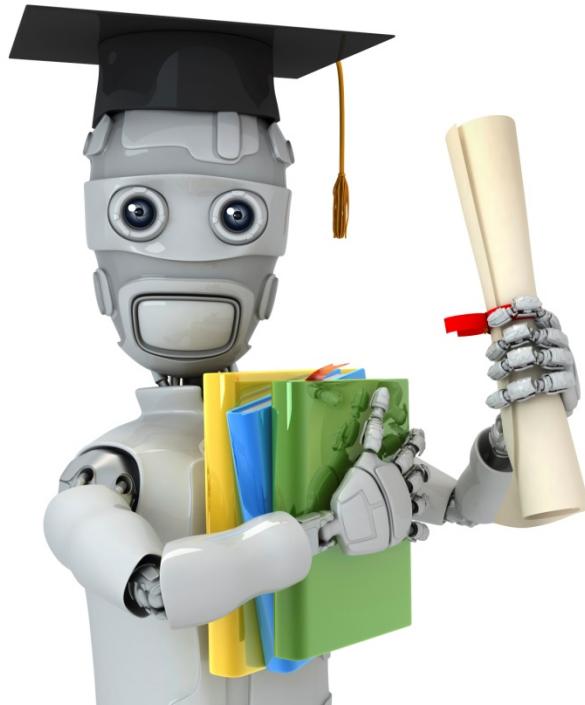
Given  $x^{(1)}, \dots, x^{(n_m)}$  (and movie ratings),  
can estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$

$$\begin{matrix} r^{(i,j)} \\ y^{(i,j)} \end{matrix}$$



Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ ,  
can estimate  $x^{(1)}, \dots, x^{(n_m)}$

Guess  $\Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \dots$



Machine Learning

# Recommender Systems

---

Collaborative  
filtering algorithm

## Collaborative filtering optimization objective

Given  $x^{(1)}, \dots, x^{(n_m)}$ , estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$(i,j) : r(i,j) = 1$$

$$x \in \mathbb{R}^n$$

$$\theta \in \mathbb{R}^n$$

$$x_1 = 1$$

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , estimate  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Minimizing  $x^{(1)}, \dots, x^{(n_m)}$  and  $\theta^{(1)}, \dots, \theta^{(n_u)}$  simultaneously:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$\min_{x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$



## Collaborative filtering algorithm

~~$x \in \mathbb{R}^n$ ,  $\theta \in \mathbb{R}^n$~~

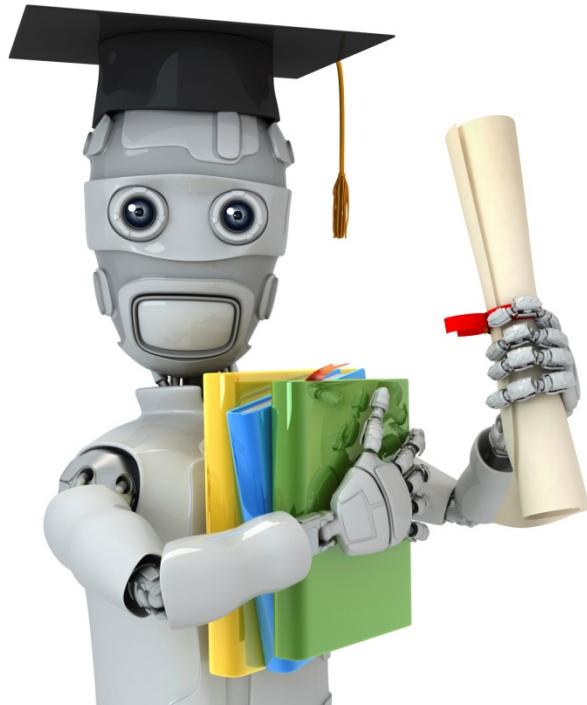
- 1. Initialize  $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$  to small random values.
- 2. Minimize  $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$  using gradient descent (or an advanced optimization algorithm). E.g. for every  $j = 1, \dots, n_u, i = 1, \dots, n_m$  :

$$x_k^{(i)} := x_k^{(i)} - \alpha \left( \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$
$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

$\frac{\partial}{\partial x_k^{(i)}} J(\dots)$

- 3. For a user with parameters  $\underline{\theta}$  and a movie with (learned) features  $\underline{x}$ , predict a star rating of  $\underline{\theta}^T \underline{x}$ .

$$(\underline{\theta}^{(i)})^T (\underline{x}^{(i)})$$



Machine Learning

# Recommender Systems

---

Vectorization:  
Low rank matrix  
factorization

# Collaborative filtering

$$n_m = 5$$

$$n_u = 4$$

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$



$y^{(i,j)}$

## Collaborative filtering

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$$\mathbf{x} \Theta^T \leftarrow (\Theta^{(1)})^T (x^{(1)})$$

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T (x^{(1)}) \\ (\theta^{(1)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(1)})^T (x^{(n_m)}) \end{bmatrix} \quad \begin{bmatrix} (\theta^{(2)})^T (x^{(1)}) \\ (\theta^{(2)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(2)})^T (x^{(n_m)}) \end{bmatrix} \quad \dots \quad \begin{bmatrix} (\theta^{(n_u)})^T (x^{(1)}) \\ (\theta^{(n_u)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(n_u)})^T (x^{(n_m)}) \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(n_m)})^T \end{bmatrix} \quad \Theta = \begin{bmatrix} -(\Theta^{(1)})^T \\ -(\Theta^{(2)})^T \\ \vdots \\ -(\Theta^{(n_u)})^T \end{bmatrix}$$

→ Low rank matrix factorization

## Finding related movies

For each product  $i$ , we learn a feature vector  $\underline{x}^{(i)} \in \mathbb{R}^n$ .

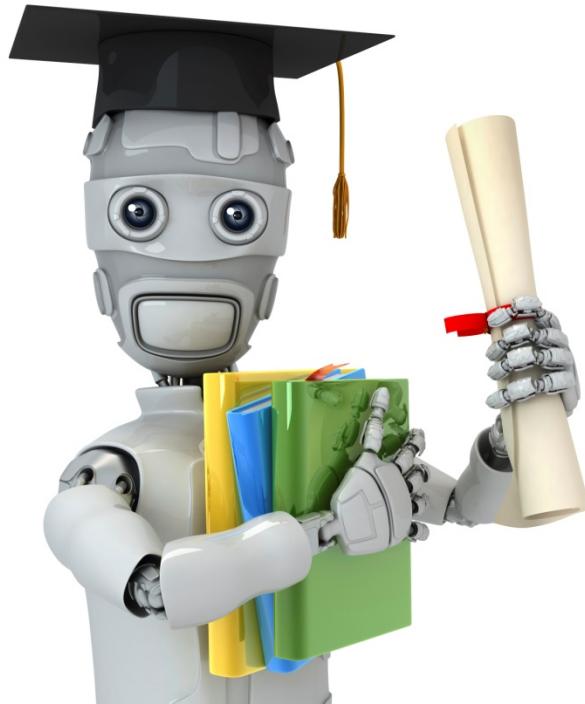
$\rightarrow x_1 = \text{romance}$ ,  $x_2 = \text{action}$ ,  $x_3 = \text{comedy}$ ,  $x_4 = \dots$

How to find movies  $j$  related to movie  $i$ ?

small  $\|x^{(i)} - x^{(j)}\|$   $\rightarrow$  movie  $j$  and  $i$  are "similar"

5 most similar movies to movie  $i$ :

Find the 5 movies  $j$  with the smallest  $\|x^{(i)} - x^{(j)}\|$ .



Machine Learning

# Recommender Systems

---

Implementational  
detail: Mean  
normalization

# Users who have not rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

↓

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} \frac{1}{2} \sum_{(i,j): r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$n=2$

$$\underline{\Theta}^{(s)} \in \mathbb{R}^2$$

$$\underline{\Theta}^{(s)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\frac{\lambda}{2} [(\underline{\Theta}_1^{(s)})^2 + (\underline{\Theta}_2^{(s)})^2] \leftarrow$$

$$(\underline{\Theta}^{(s)})^T \underline{x}^{(i)} = 0$$

## Mean Normalization:

$$Y = \begin{bmatrix} \rightarrow & 5 & 5 & 0 & 0 & ? & -2.5 \\ \rightarrow & 5 & ? & ? & 0 & ? & -2.5 \\ Y = & ? & 4 & 0 & ? & ? & -2 \\ \rightarrow & 0 & 0 & 5 & 4 & ? & \vdots \\ \rightarrow & 0 & 0 & 5 & 0 & ? & \vdots \end{bmatrix}$$

$$\mu = \begin{bmatrix} \rightarrow & 2.5 \\ \rightarrow & 2.5 \\ \rightarrow & 2 \\ \rightarrow & 2.25 \\ \rightarrow & 1.25 \end{bmatrix} \rightarrow \underline{Y} =$$

$$\begin{bmatrix} \circled{2.5} & \circled{2.5} & \circled{-2.5} & \circled{-2.5} & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

$$\rightarrow (\underline{\theta}^{(s)})^T (\underline{x}^{(i)}) + \underline{\mu_i}$$

$\downarrow$   
learn  $\underline{\theta}^{(s)}, \underline{x}^{(i)}$

User 5 (Eve):

$$\underline{\theta}^{(s)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\underbrace{(\underline{\theta}^{(s)})^T (\underline{x}^{(i)})}_{\sim 0} + \boxed{\underline{\mu_i}}$$

# Copyright Notice

These slides are distributed under the Creative Commons License.

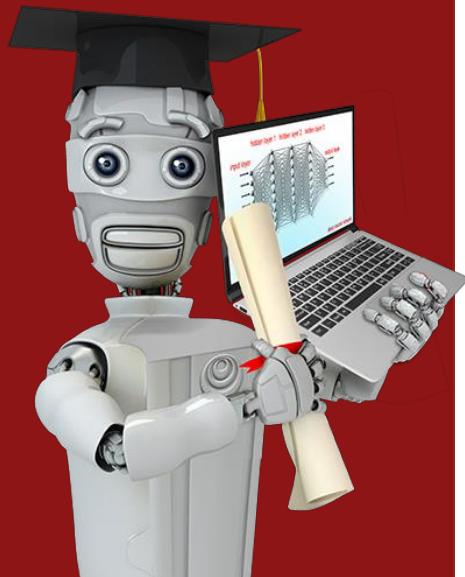
[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

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



Stanford  
ONLINE

# Advanced Learning Algorithms



## Welcome!

# Advanced learning algorithms

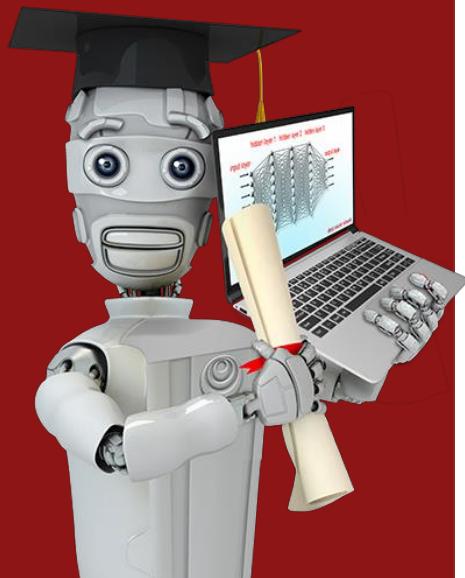
Neural Networks 

inference (prediction)

training

Practical advice for building machine learning systems 

Decision Trees 



## Neural Networks Intuition

# Neurons and the brain

# Neural networks

Origins: Algorithms that try to mimic the brain.

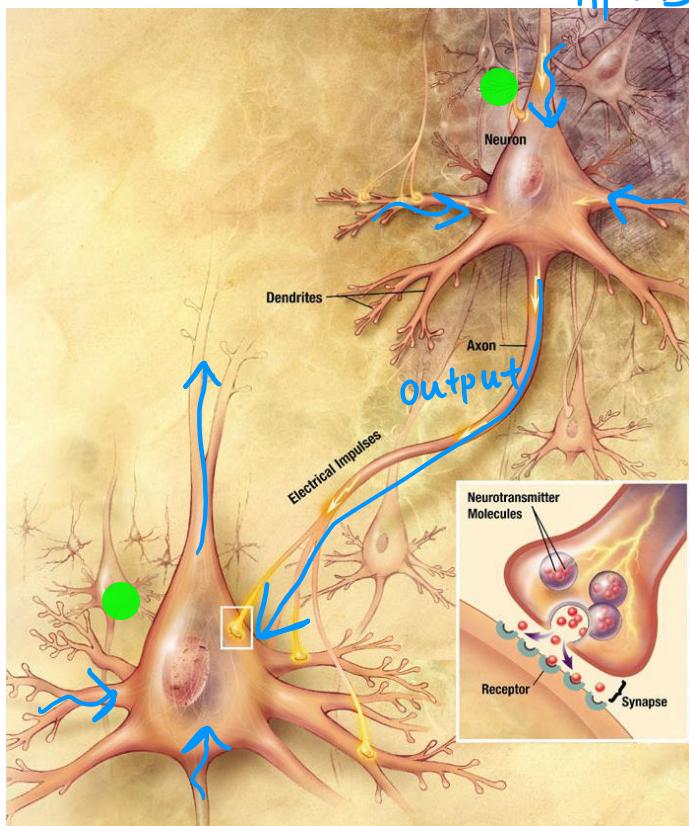


Used in the 1980's and early 1990's.  
Fell out of favor in the late 1990's.

Resurgence from around 2005.

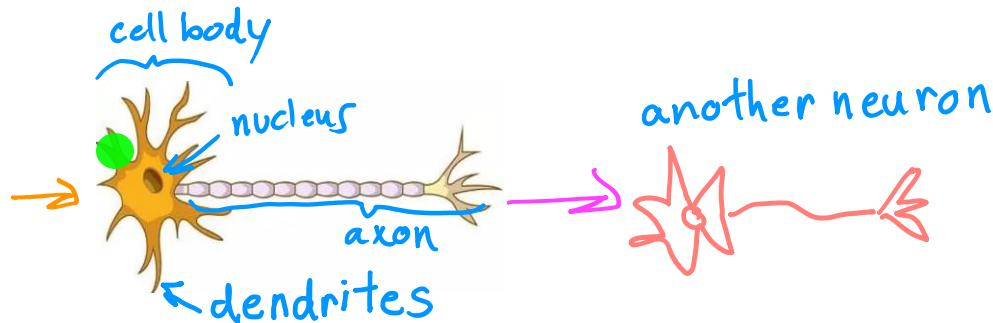
speech → images → text (NLP) → ...

# Neurons in the brain



## Biological neuron

inputs                    outputs



## Simplified mathematical model of a neuron

inputs                    outputs

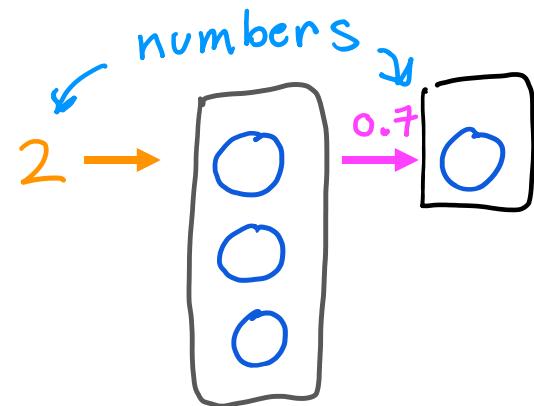
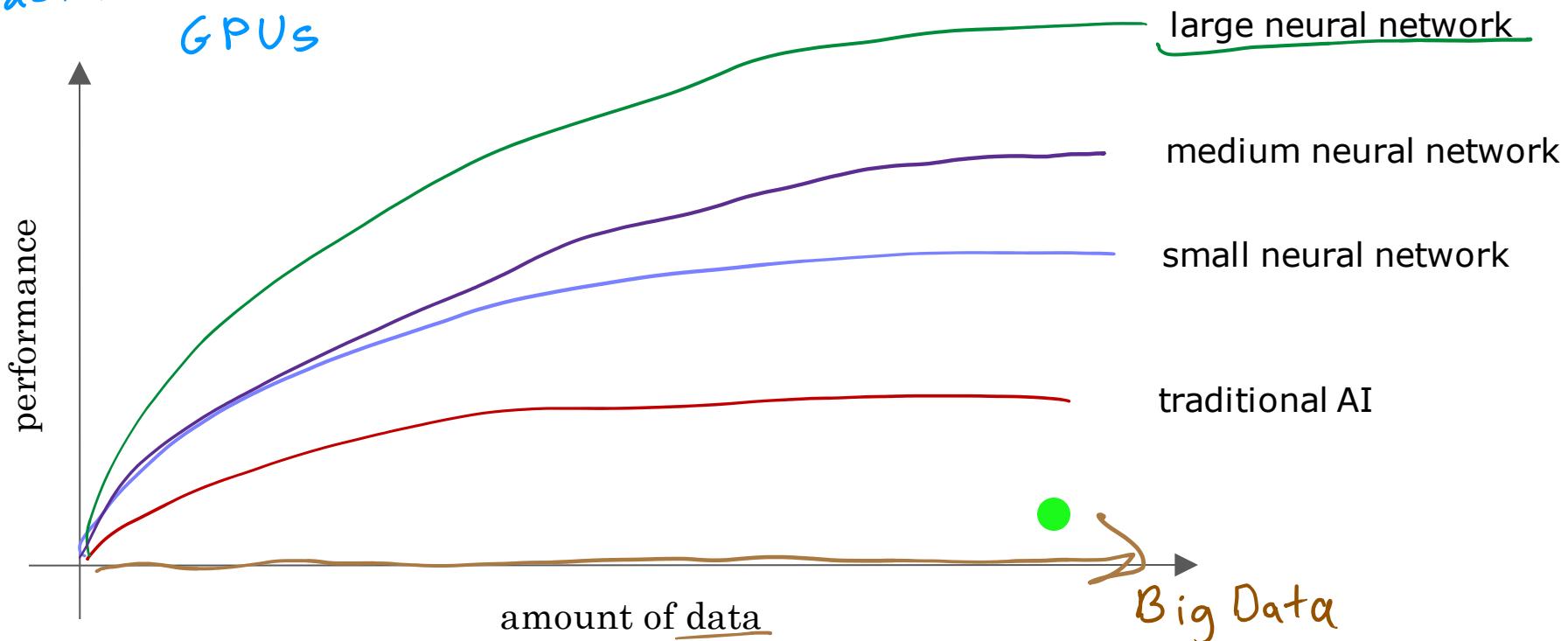
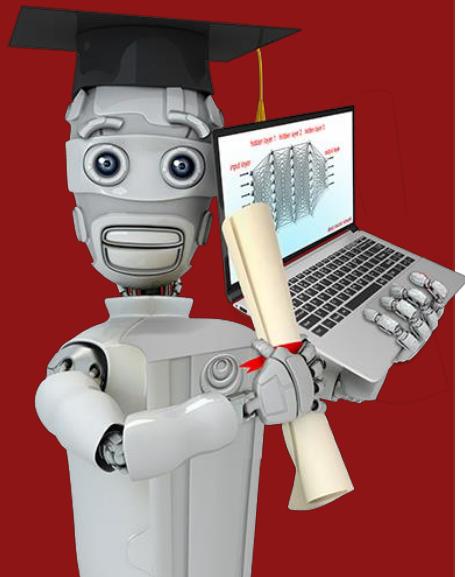


image source: <https://biologydictionary.net/sensory-neuron/>

Faster computer processors  
GPUs

# Why Now?

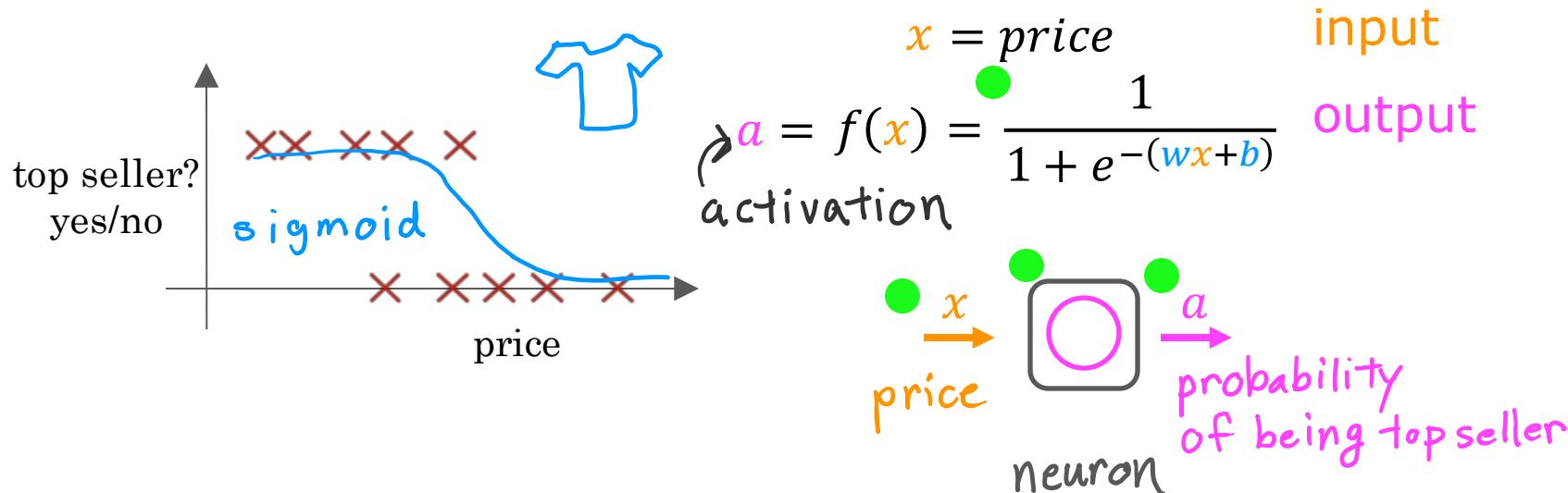




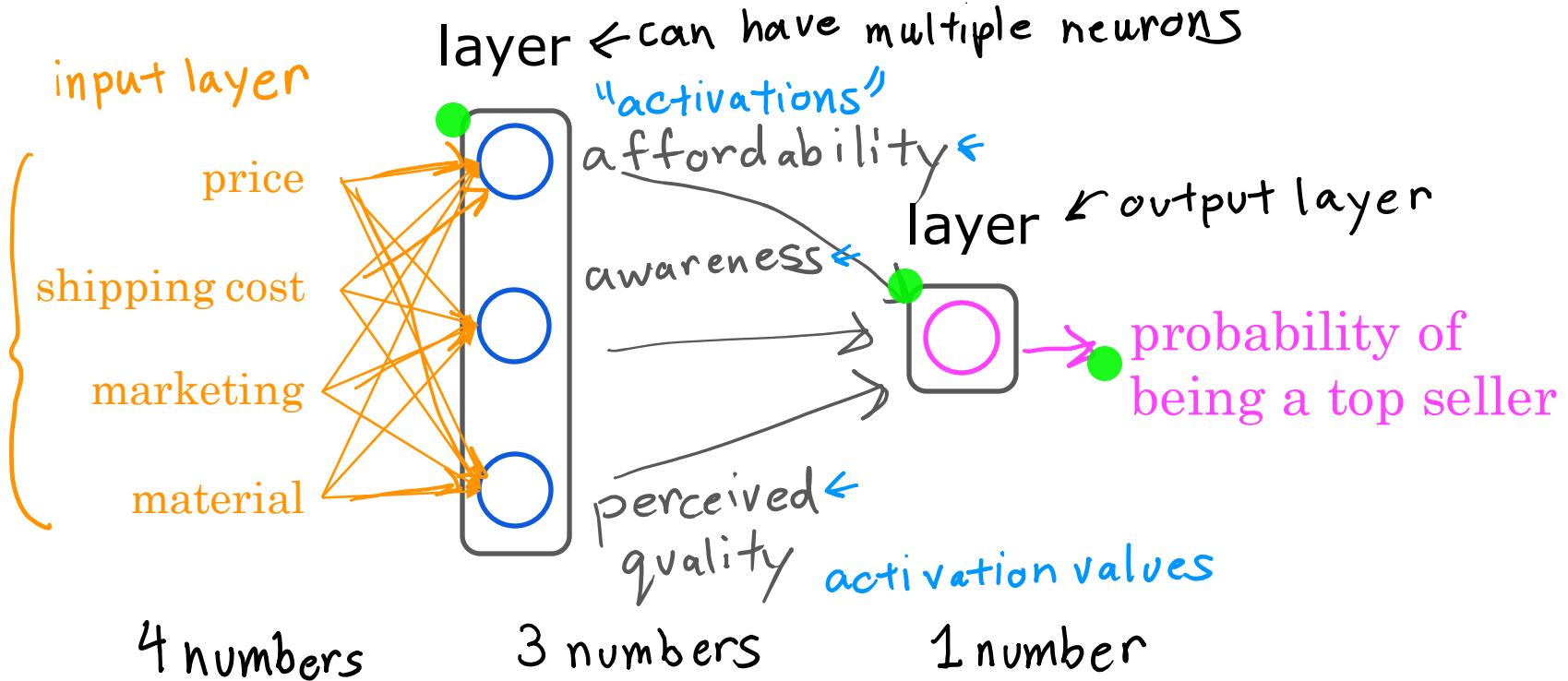
## Neural Network Intuition

# Demand Prediction

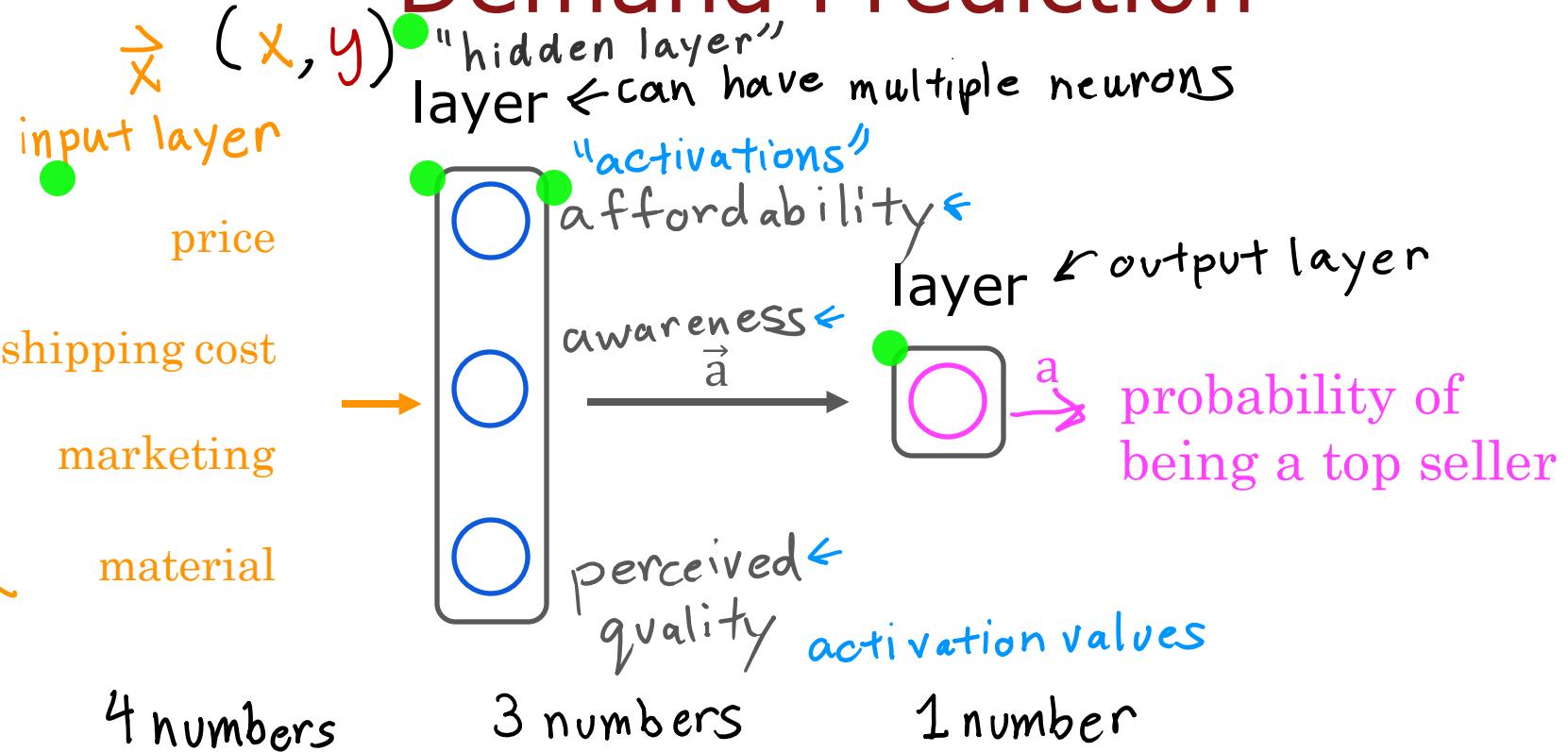
# Demand Prediction



# Demand Prediction



# Demand Prediction



# Demand Prediction

•  $\vec{x} \rightarrow (x, y)$

input layer

price

shipping cost

marketing

material

4 numbers

"hidden layer"  
layer can have multiple neurons

"activations"

affordability

awareness

$\vec{a}$

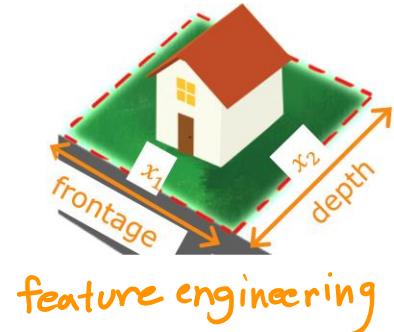
layer

output layer

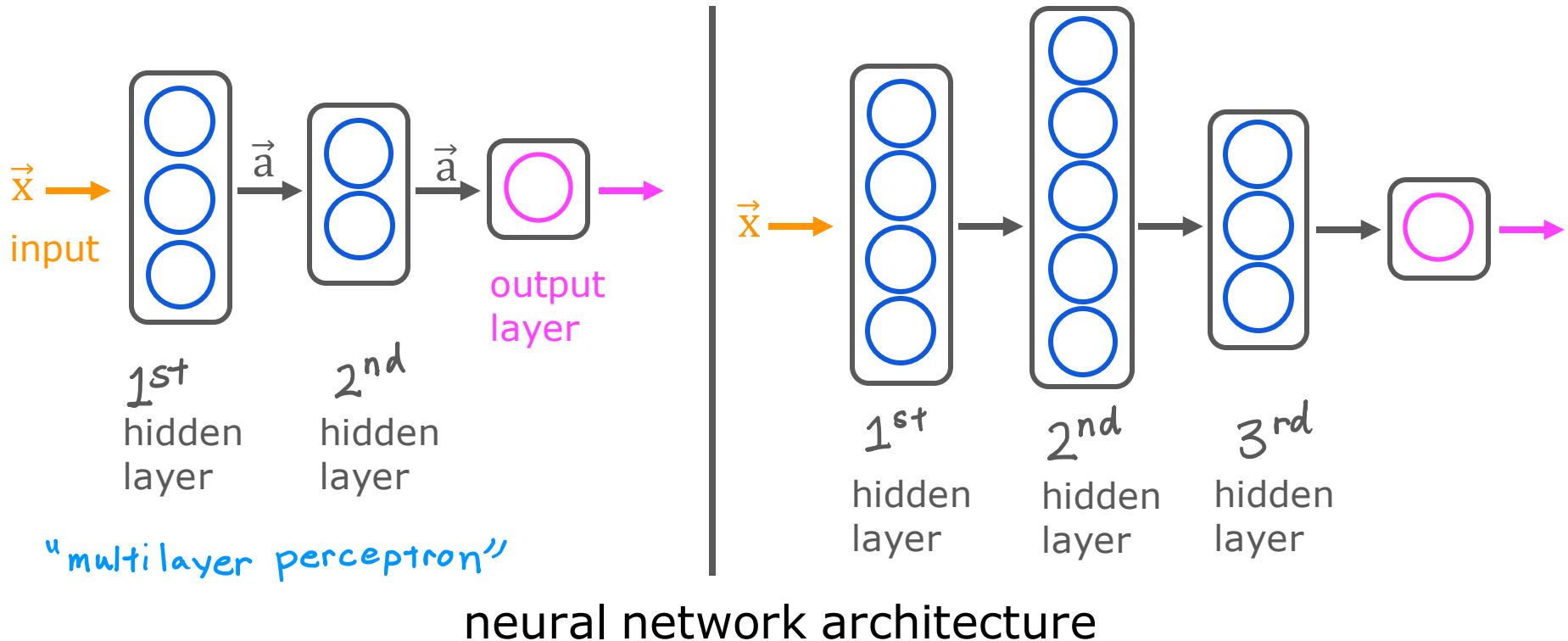
perceived  
quality

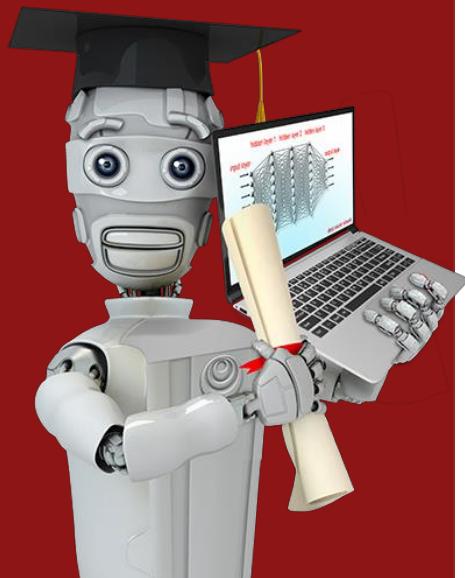
activation values

1 number



# Multiple hidden layers

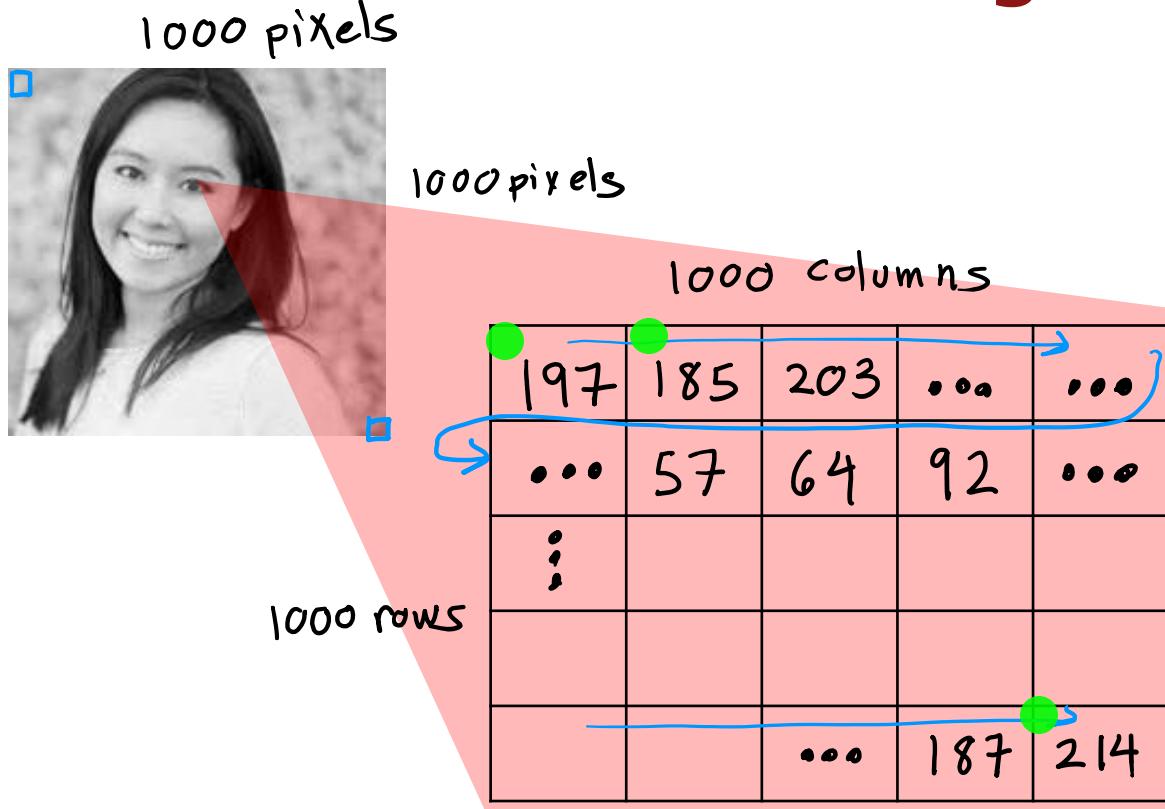




# Neural Networks Intuition

**Example:**  
**Recognizing Images**

# Face recognition

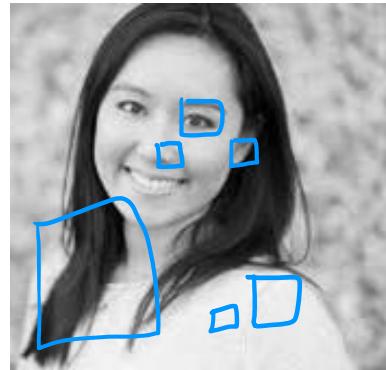


197
185
203
⋮
57
64
92
⋮
187
214

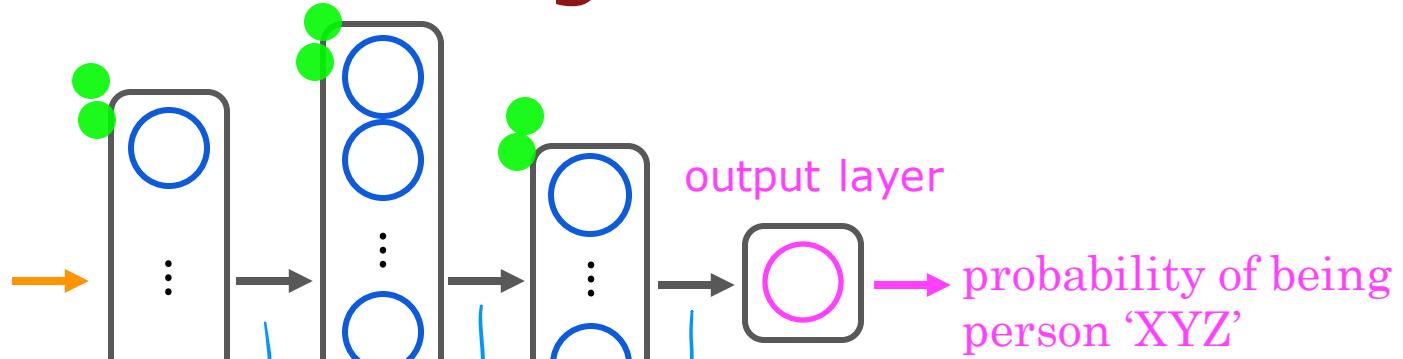
$\vec{x} =$

1 million

# Face recognition



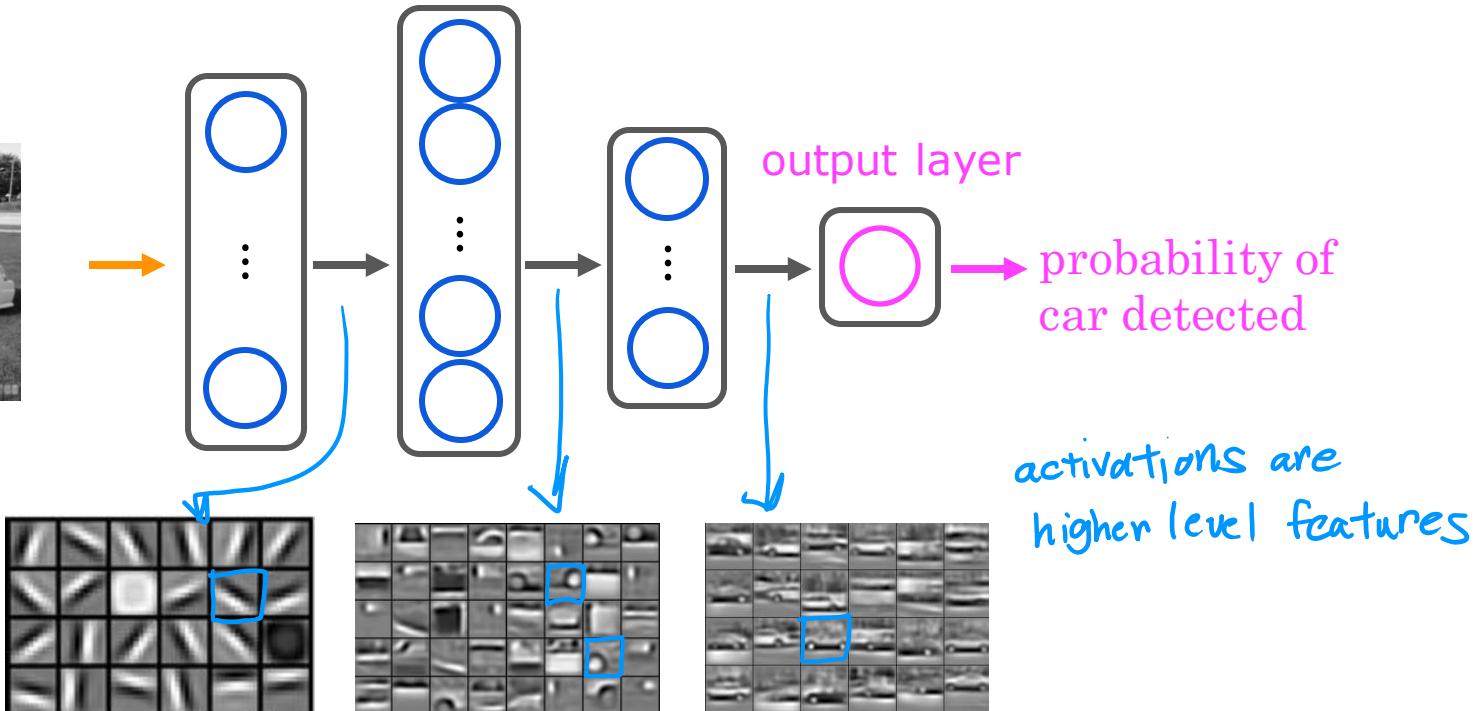
$\vec{x}$   
input



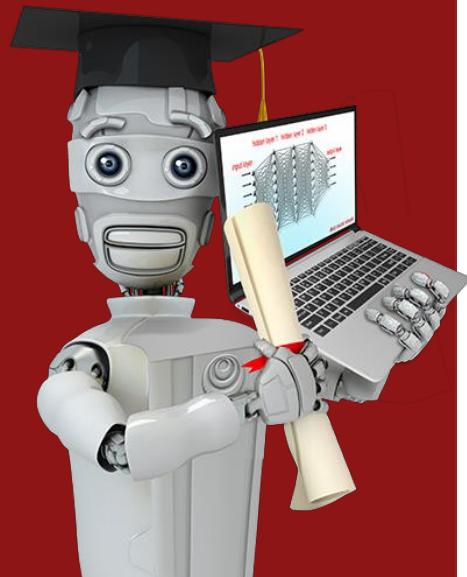
activations are  
higher level features

source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

# Car classification



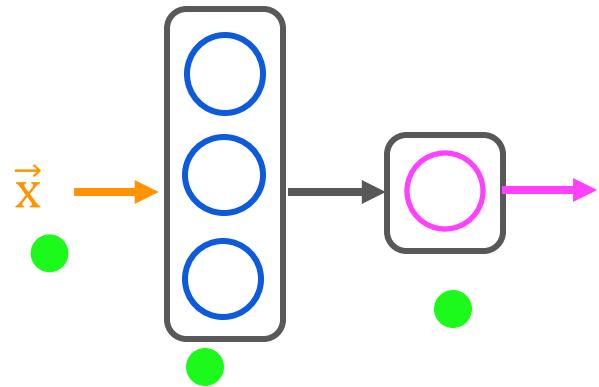
source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng



## Neural network model

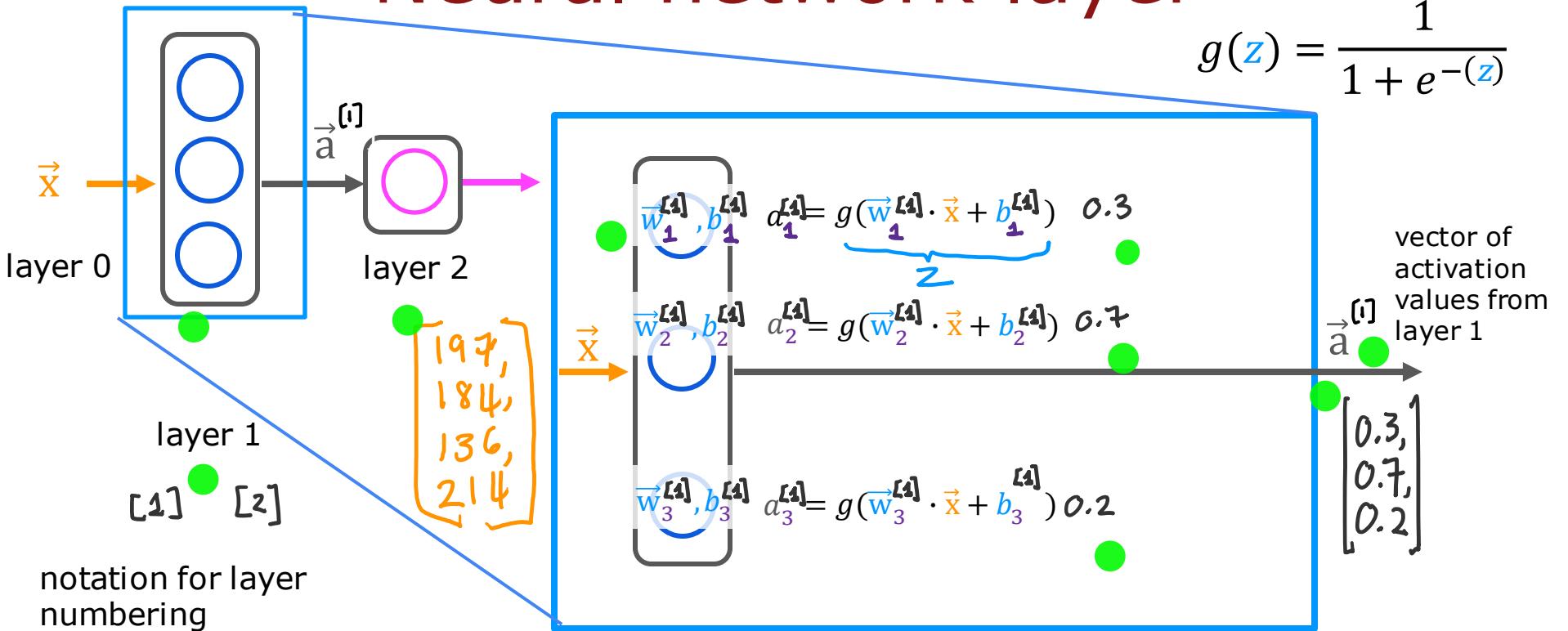
## Neural network layer

# Neural network layer



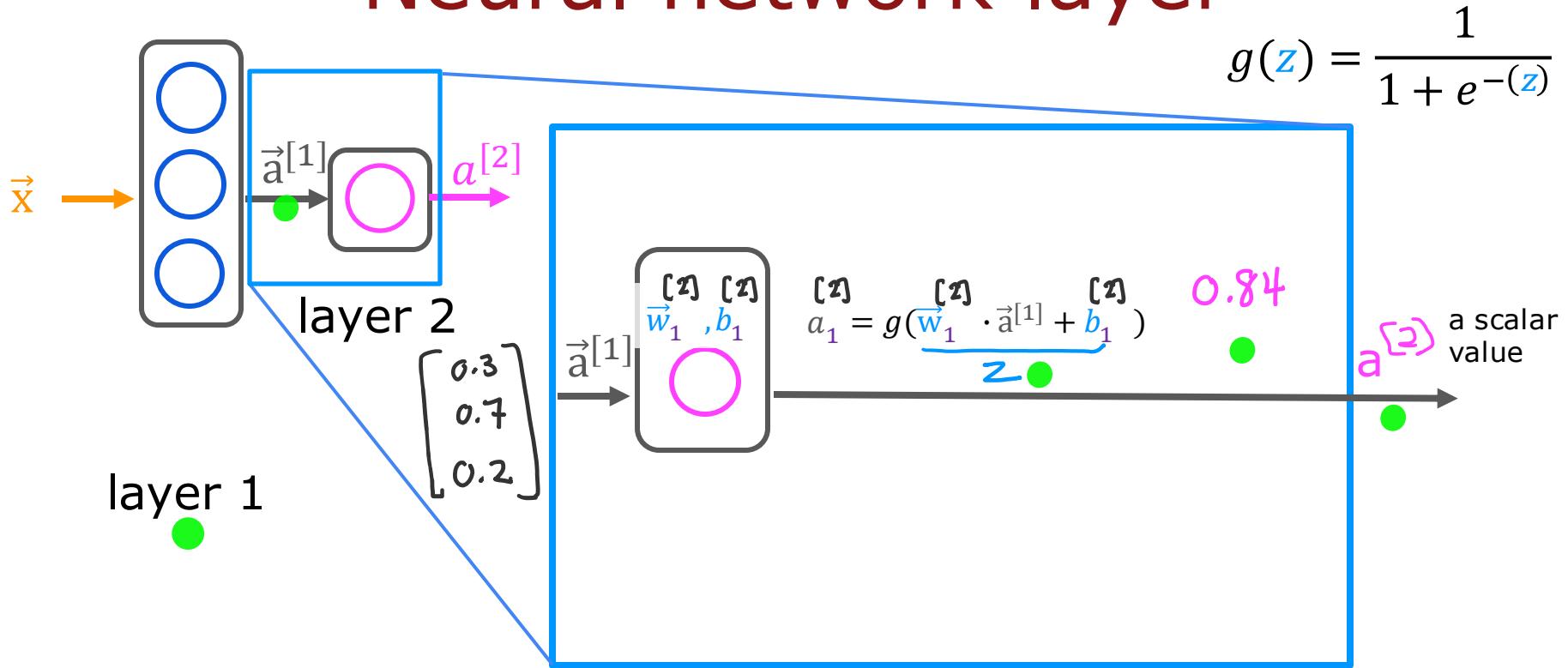
# Neural network layer

$$g(z) = \frac{1}{1 + e^{-(z)}}$$

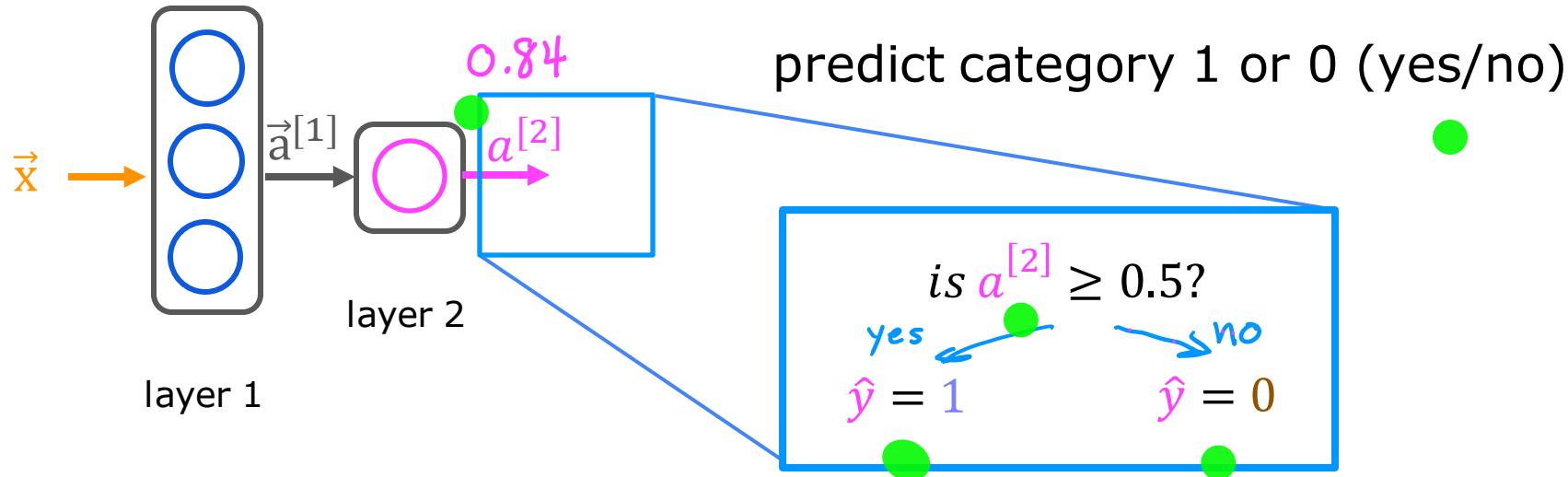


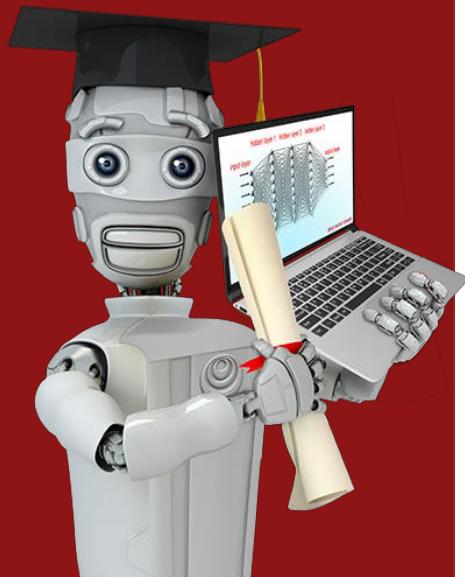
notation for layer  
numbering

# Neural network layer



# Neural network layer

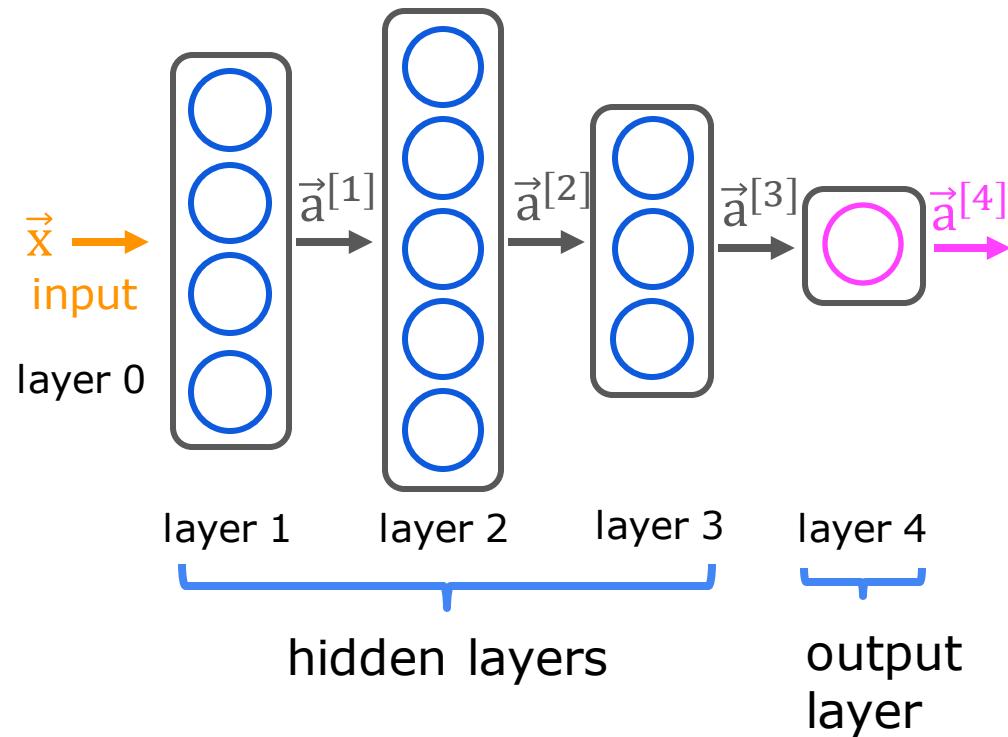




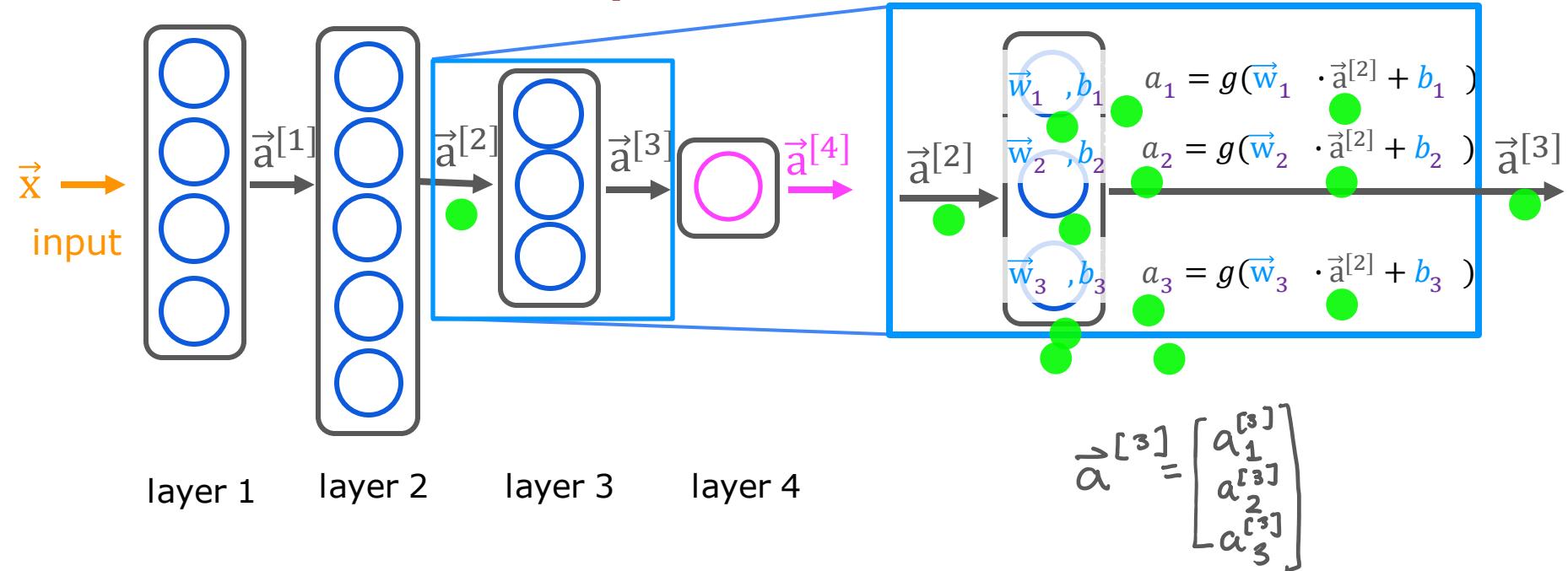
## Neural Network Model

More complex neural networks

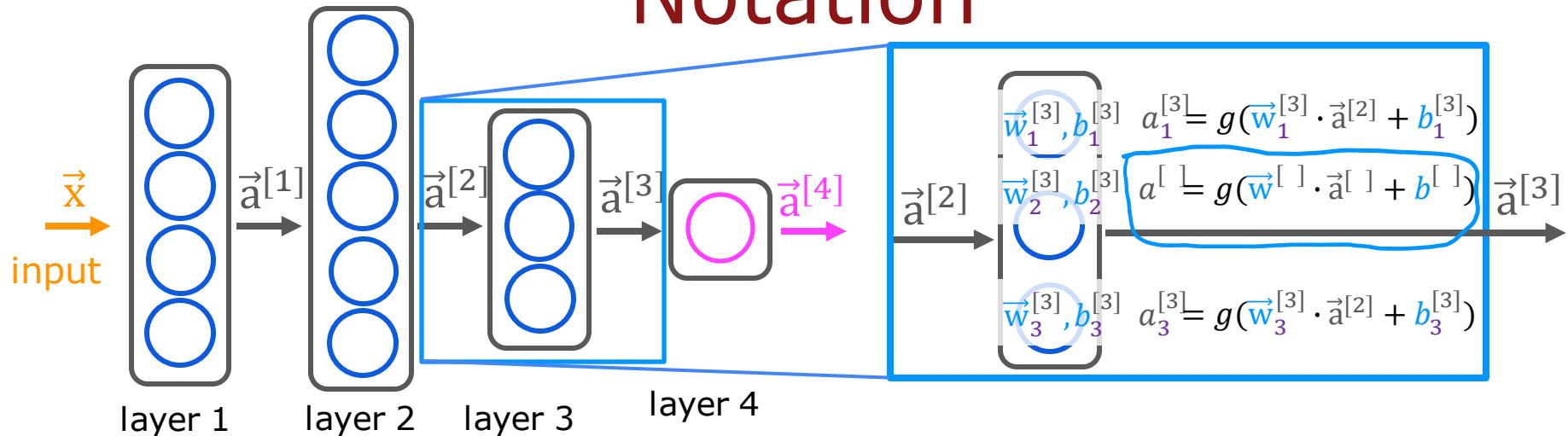
# More complex neural network



# More complex neural network

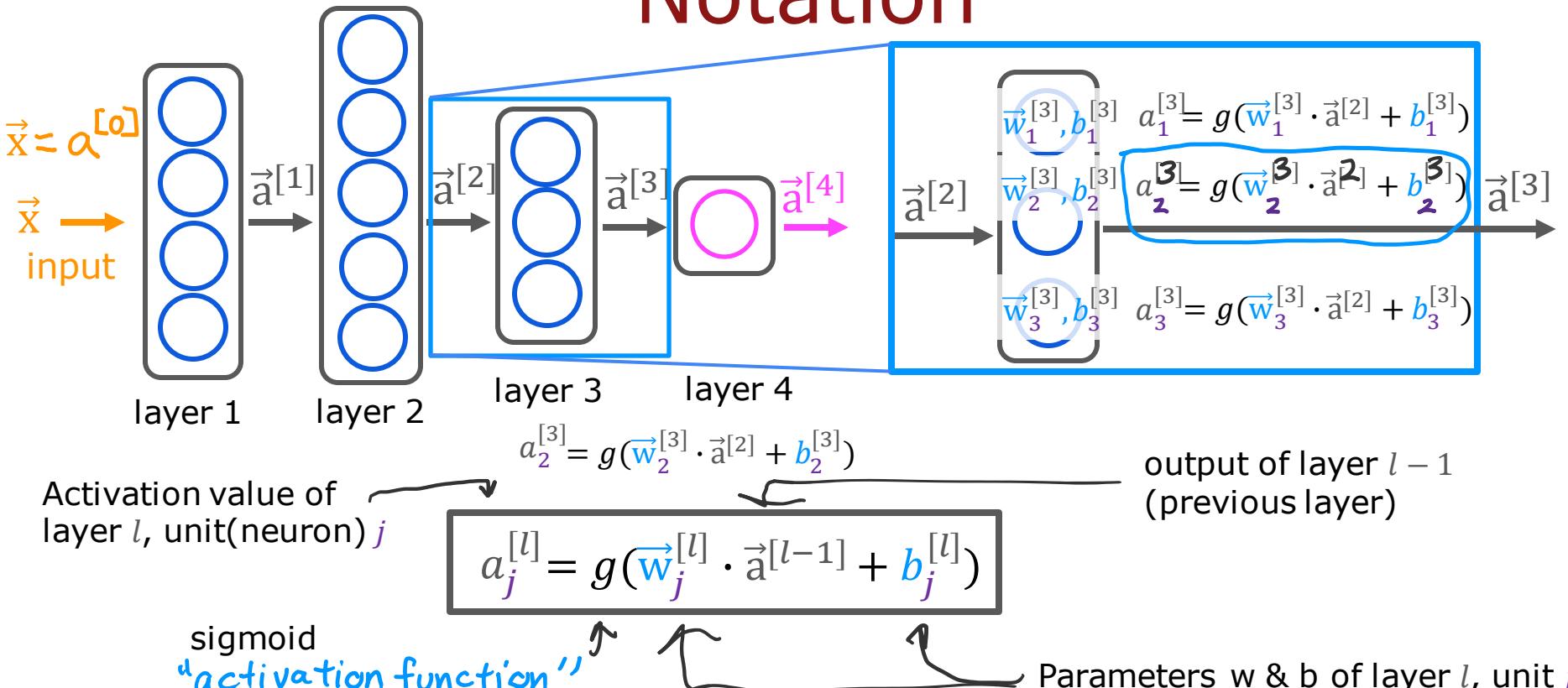


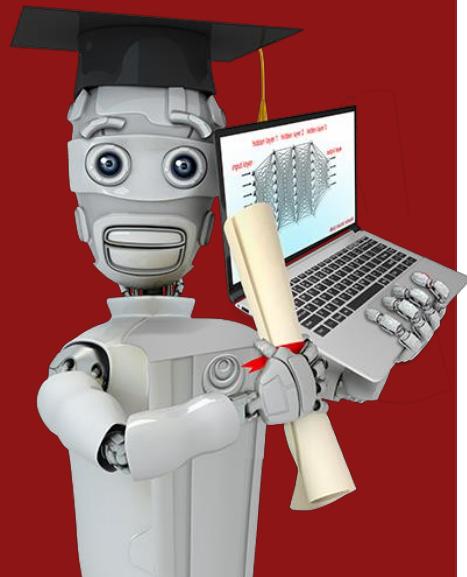
# Notation



Question:  
Can you fill in the superscripts and  
subscripts for the second neuron?

# Notation

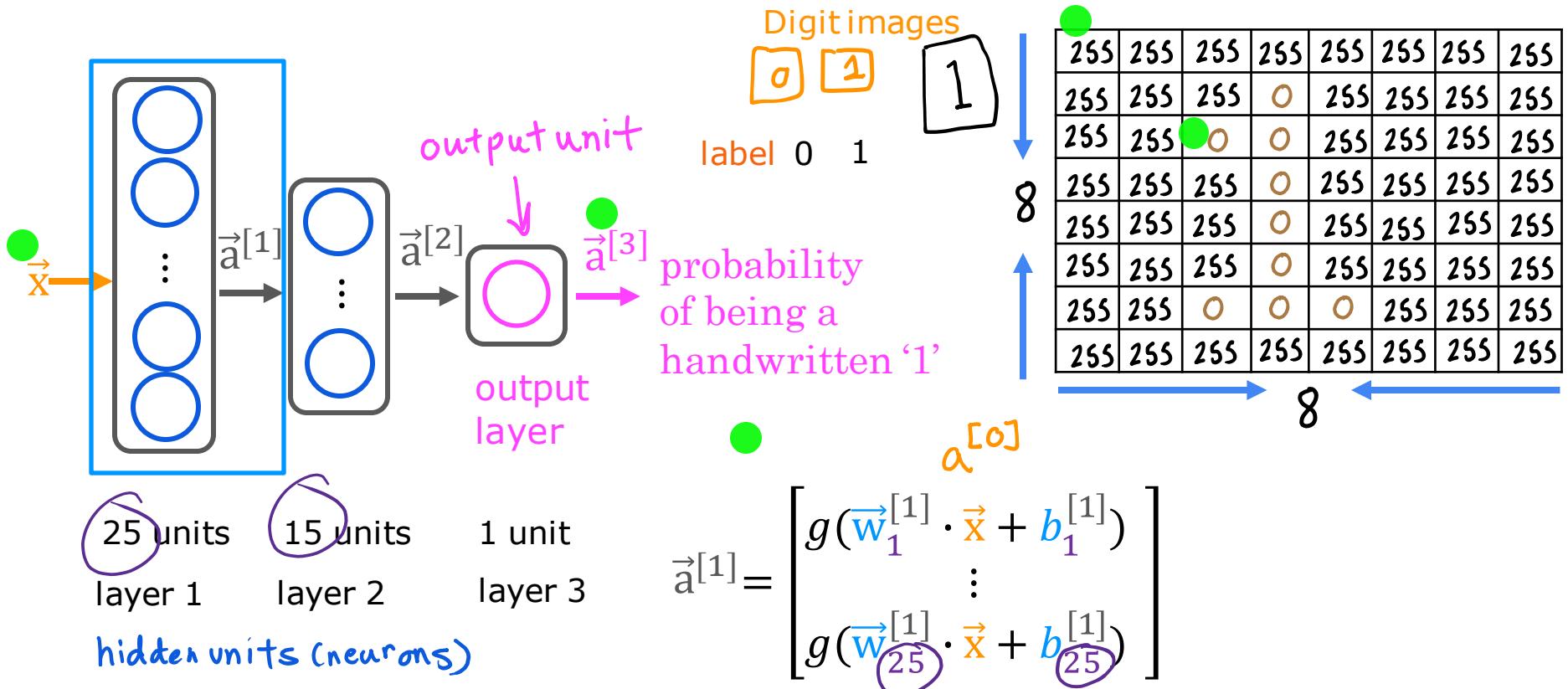




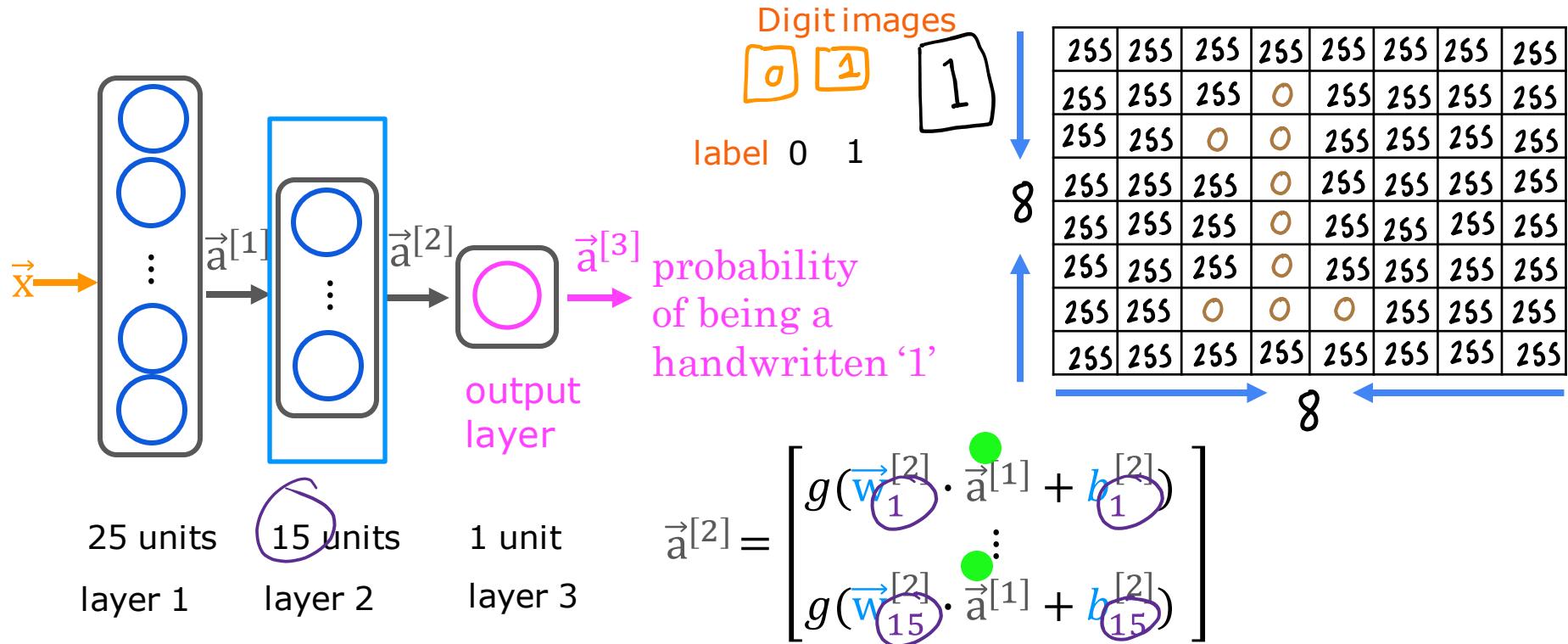
## Neural Network Model

Inference: making predictions  
(forward propagation)

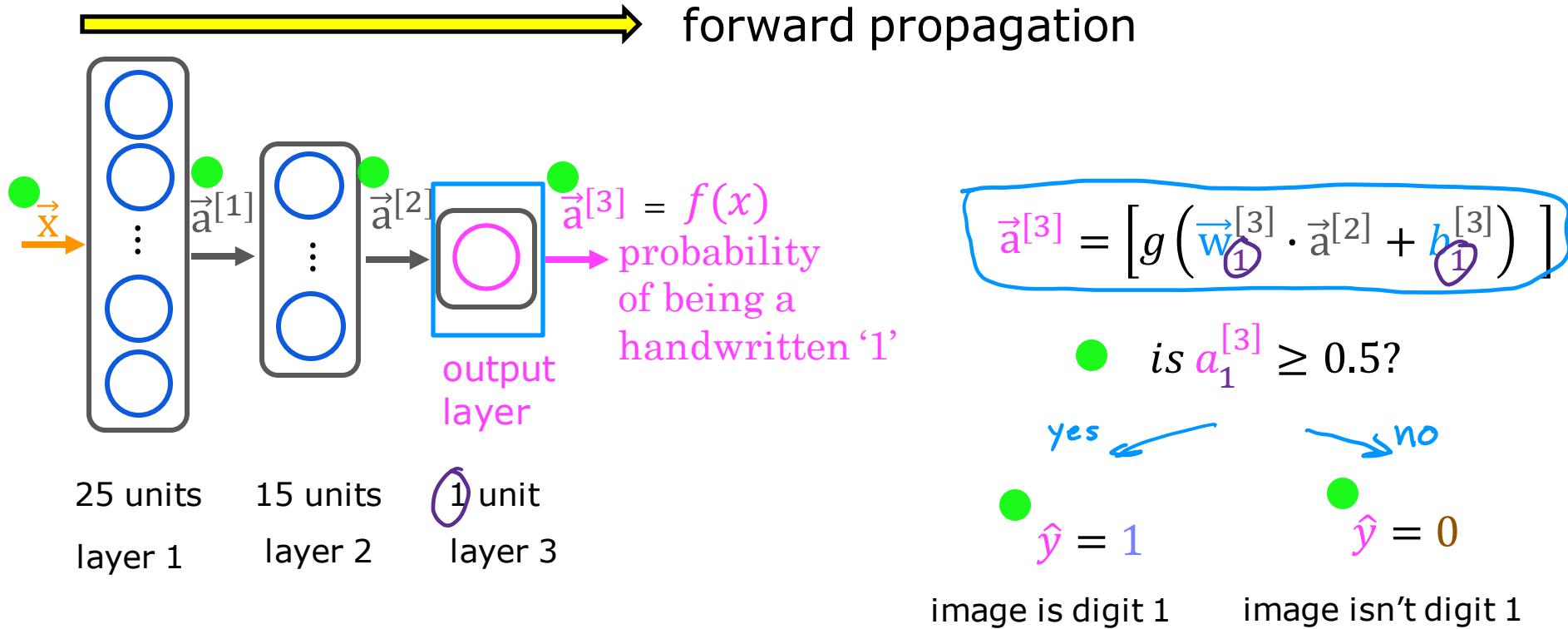
# Handwritten digit recognition

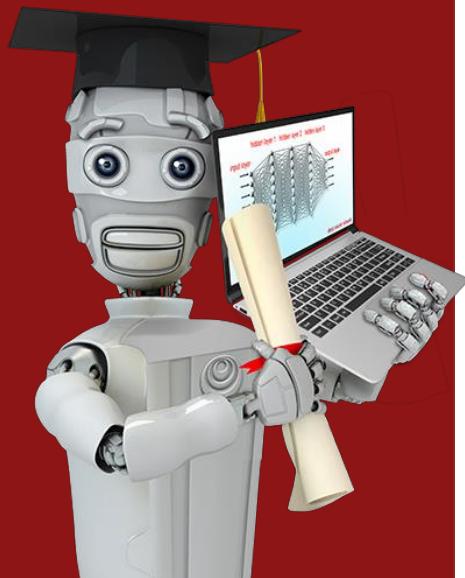


# Handwritten digit recognition



# Handwritten digit recognition

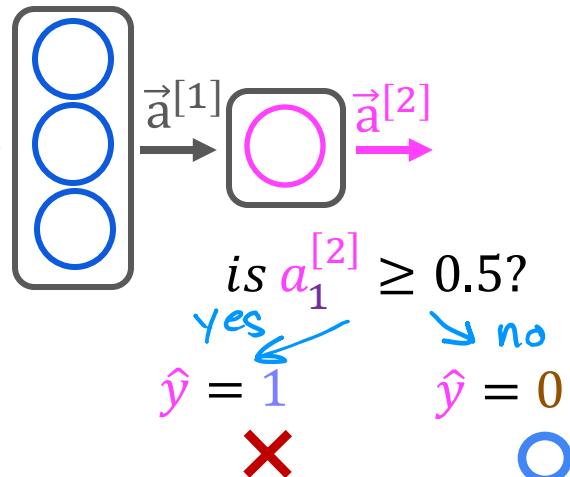
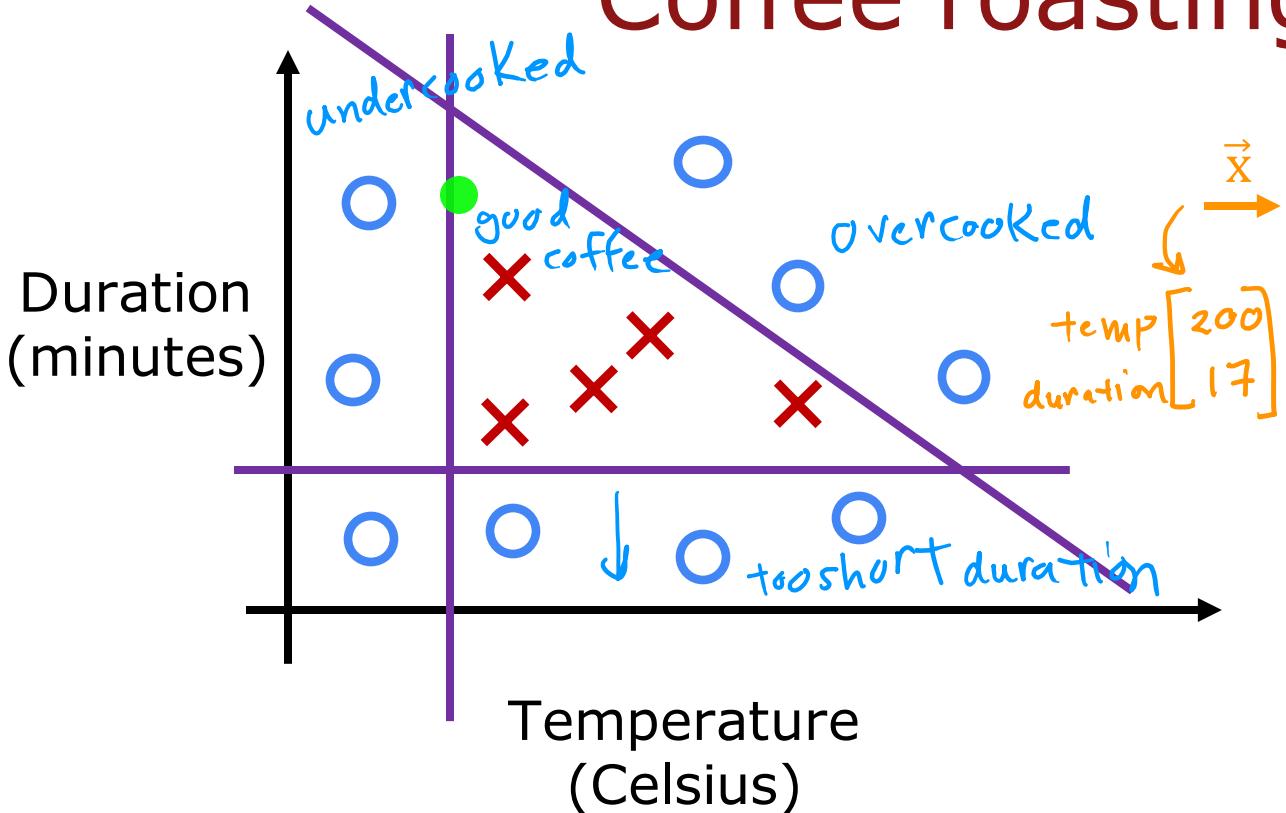


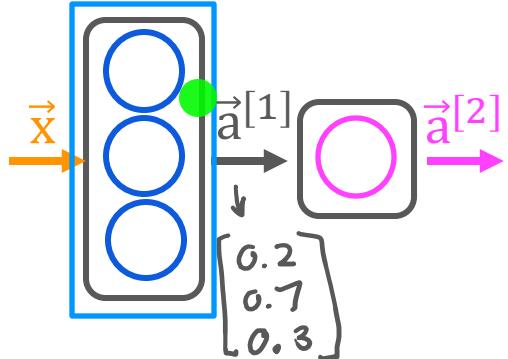


## TensorFlow implementation

## Inference in Code

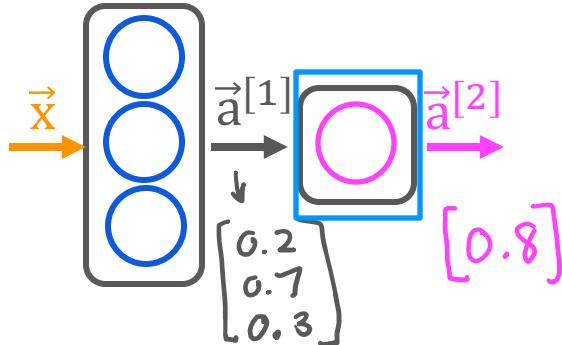
# Coffee roasting





```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

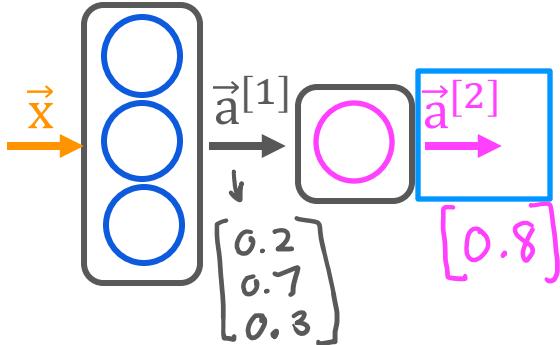
# Build the model using TensorFlow



```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

# Build the model using TensorFlow



```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

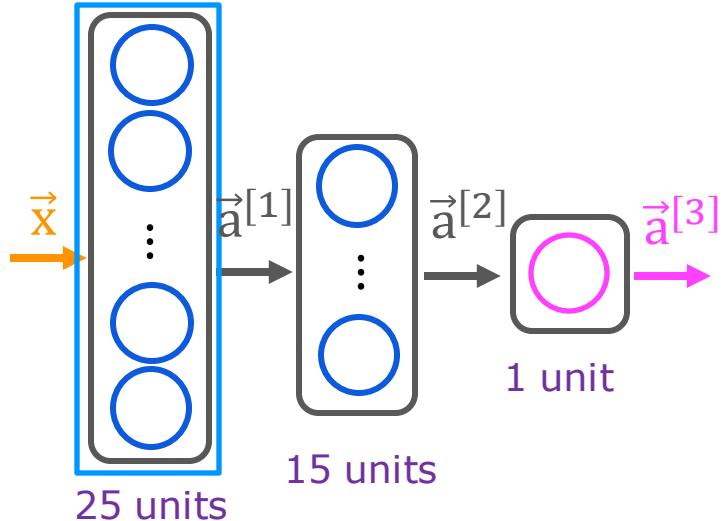
```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

is  $a_1^{[2]} \geq 0.5?$

yes  $\hat{y} = 1$  no  $\hat{y} = 0$

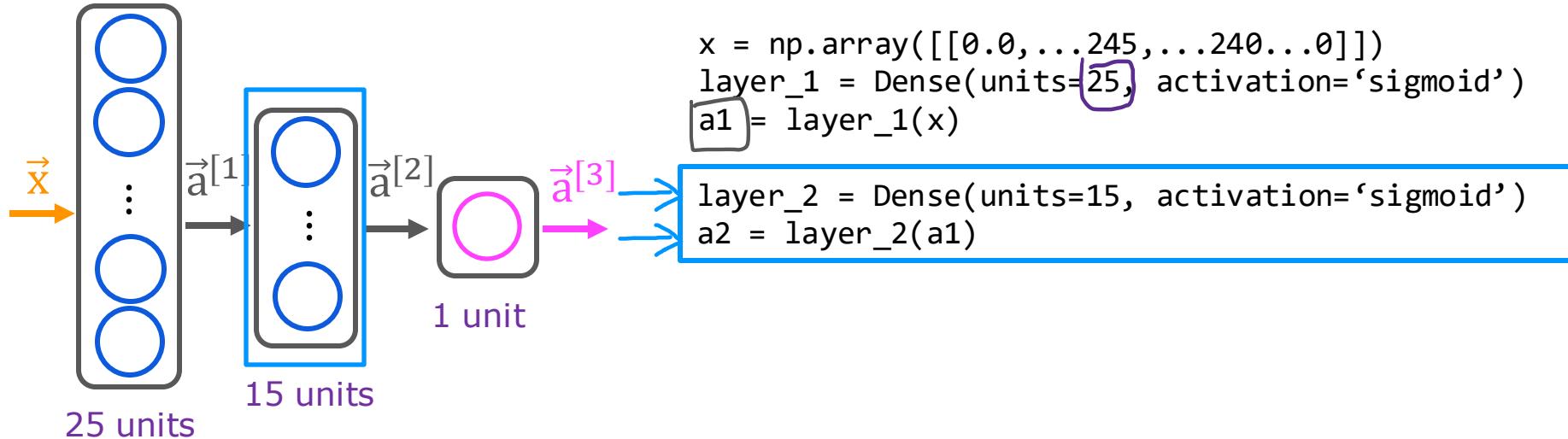
```
if a2 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

# Model for digit classification

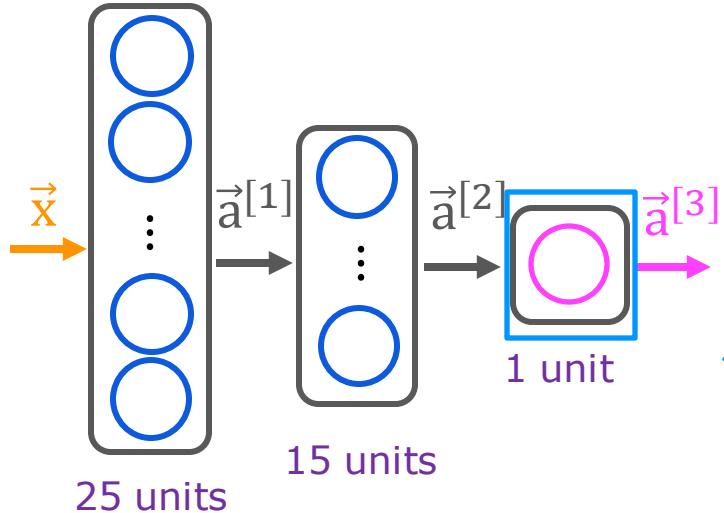


```
x = np.array([[0.0, ..., 245, ..., 240, ..., 0]])  
layer_1 = Dense(units=25, activation='sigmoid')  
a1 = layer_1(x)
```

# Model for digit classification



# Model for digit classification

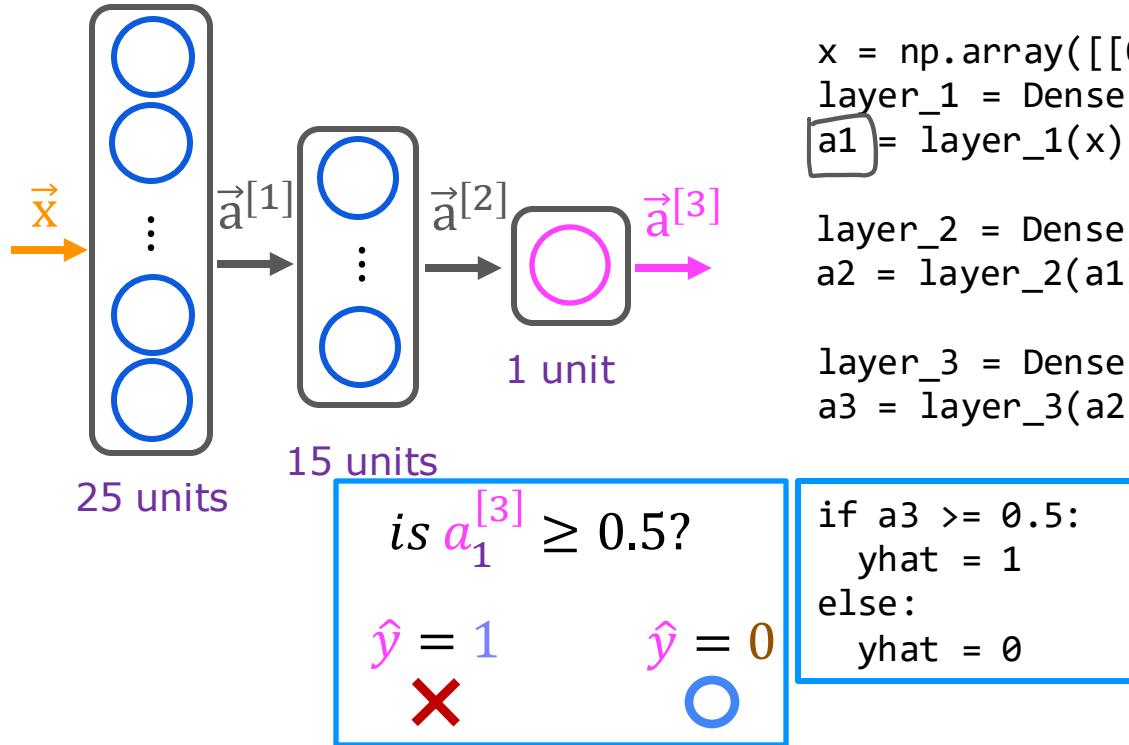


```
x = np.array([[0.0, ..., 245, ..., 240, ..., 0]])  
layer_1 = Dense(units=25, activation='sigmoid')  
a1 = layer_1(x)
```

```
layer_2 = Dense(units=15, activation='sigmoid')  
a2 = layer_2(a1)
```

```
layer_3 = Dense(units=1, activation='sigmoid')  
a3 = layer_3(a2)
```

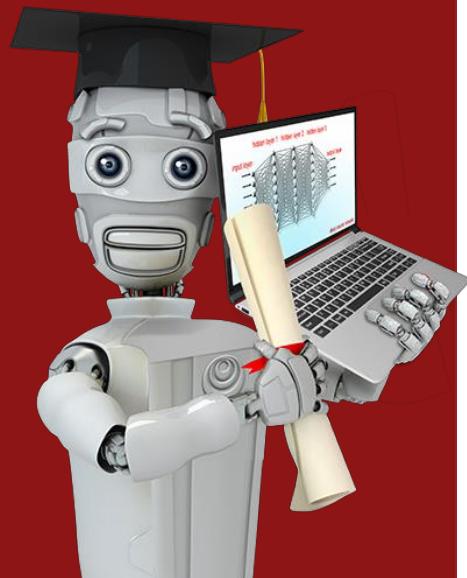
# Model for digit classification



```
x = np.array([[0.0, ...245,...240...0]])  
layer_1 = Dense(units=25, activation='sigmoid')  
a1 = layer_1(x)
```

```
layer_2 = Dense(units=15, activation='sigmoid')  
a2 = layer_2(a1)
```

```
layer_3 = Dense(units=1, activation='sigmoid')  
a3 = layer_3(a2)
```



## TensorFlow implementation

# Data in TensorFlow

# Feature vectors

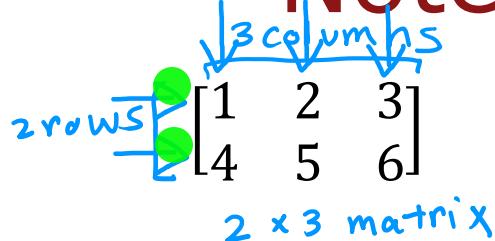
temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

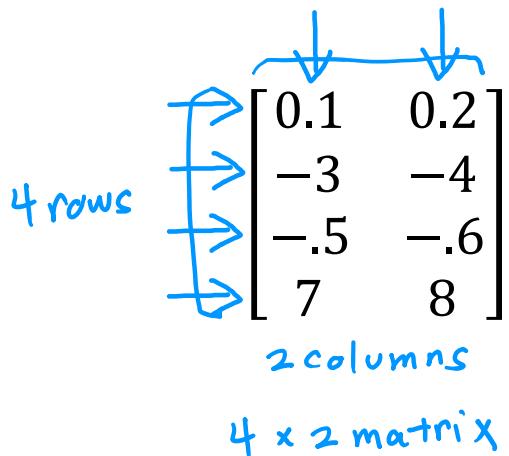
x = np.array([[200.0, 17.0]]) ←

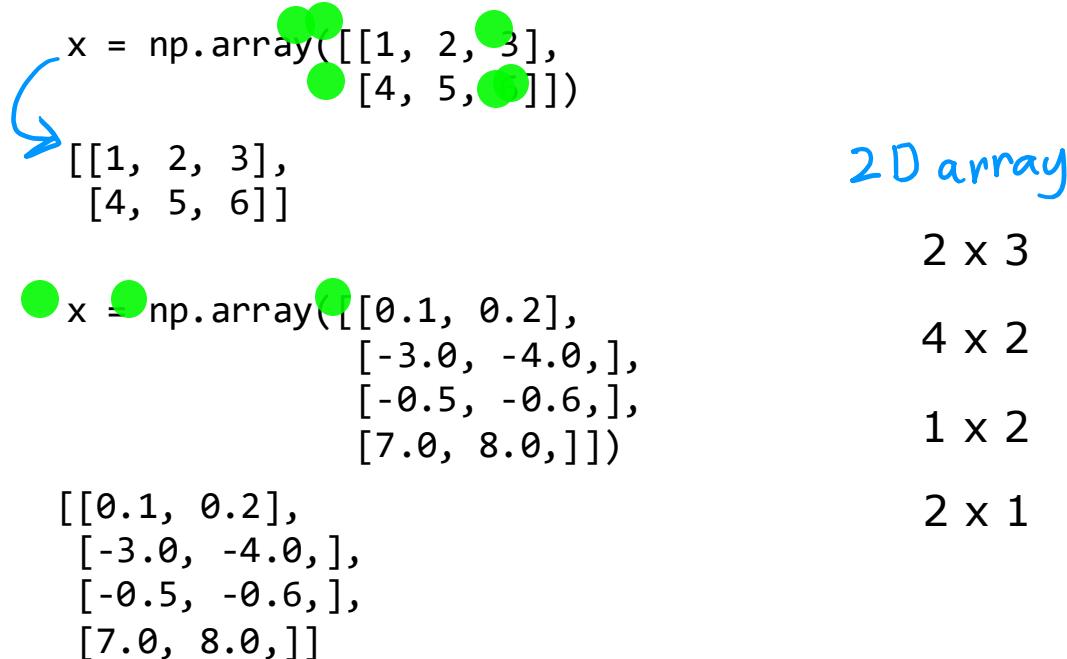
[[200.0, 17.0]]

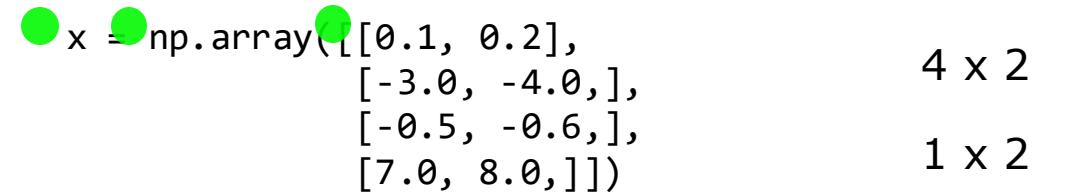
why?

# Note about numpy arrays

  
2 rows  
3 columns  
 $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$   
 $2 \times 3$  matrix

  
4 rows  
2 columns  
 $\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -0.5 & -0.6 \\ 7 & 8 \end{bmatrix}$   
 $4 \times 2$  matrix

  
 $x = np.array([[1, 2, 3], [4, 5, 6]])$   
 $[[1, 2, 3], [4, 5, 6]]$   
2D array  
 $2 \times 3$

  
 $x = np.array([[0.1, 0.2], [-3.0, -4.0], [-0.5, -0.6], [7.0, 8.0]])$   
 $[[0.1, 0.2], [-3.0, -4.0], [-0.5, -0.6], [7.0, 8.0]]$   
 $4 \times 2$

  
 $x = np.array([[-0.5, -0.6], [7.0, 8.0]])$   
 $[-0.5, -0.6], [7.0, 8.0]$   
 $1 \times 2$

  
 $x = np.array([[7.0, 8.0]])$   
 $[7.0, 8.0]$   
 $2 \times 1$

# Note about numpy arrays

x = np.array([[200, 17]]) → [200 17] 1 x 2

x = np.array([200], [17]) → [200 17] 2 x 1

→ x = np.array([200, 17])

1D  
"Vector"

# Feature vectors

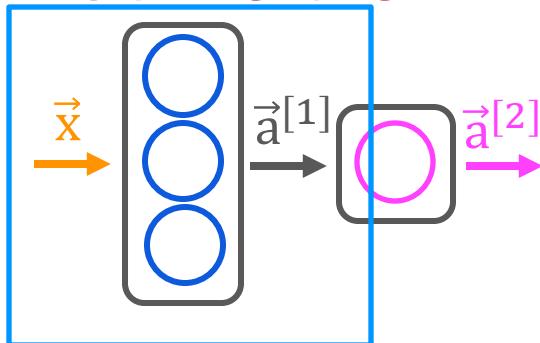
temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

`x = np.array([[200.0, 17.0]])` ←

`[[200.0, 17.0]]`

↓      ↓      1 x 2  
→ [200.0    17.0]

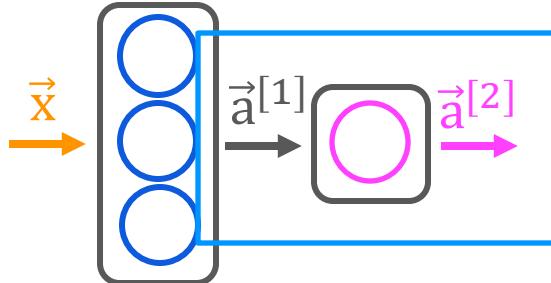
# Activation vector



```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
→ [0.2, 0.7, 0.3] 1 x 3 matrix
→ tr.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)
→ a1.numpy()
```

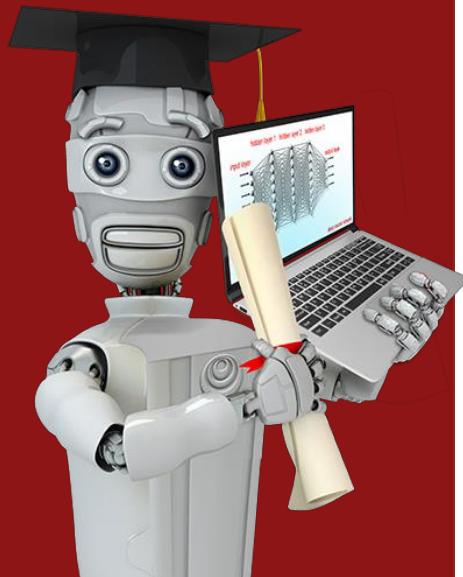
```
array([[1.4661001, 1.125196 , 3.2159438]], dtype=float32)
```

# Activation vector



```
→ layer_2 = Dense(units=1, activation='sigmoid')
→ a2 = layer_2(a1)
→ [[0.8]] ←
→ tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
→ a2.numpy()
→ array([[0.8]], dtype=float32)
```

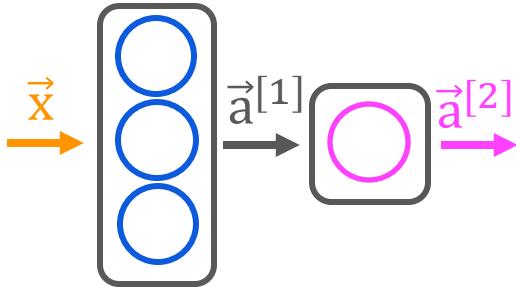
$1 \times 1$



## TensorFlow implementation

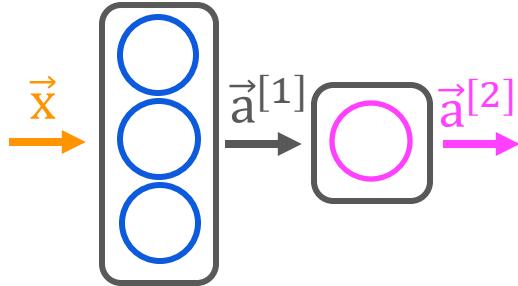
### Building a neural network

# What you saw earlier

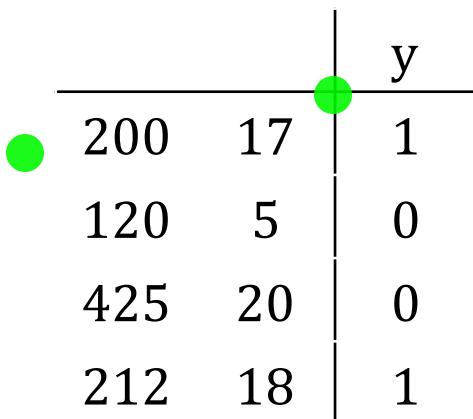


```
→ x = np.array([[200.0, 17.0]])  
→ layer_1 = Dense(units=3, activation="sigmoid")  
→ a1 = layer_1(x)  
  
→ layer_2 = Dense(units=1, activation="sigmoid")  
→ a2 = layer_2(a1)
```

# Building a neural network architecture



```
→ layer_1 = Dense(units=3, activation="sigmoid")  
→ layer_2 = Dense(units=1, activation="sigmoid")  
→ model = Sequential([layer_1, layer_2])
```



```
x = np.array([[200.0, 17.0],  
              [120.0, 5.0],  
              [425.0, 20.0],  
              [212.0, 18.0]])
```

$4 \times 2$

```
y = np.array([1,0,0,1])
```

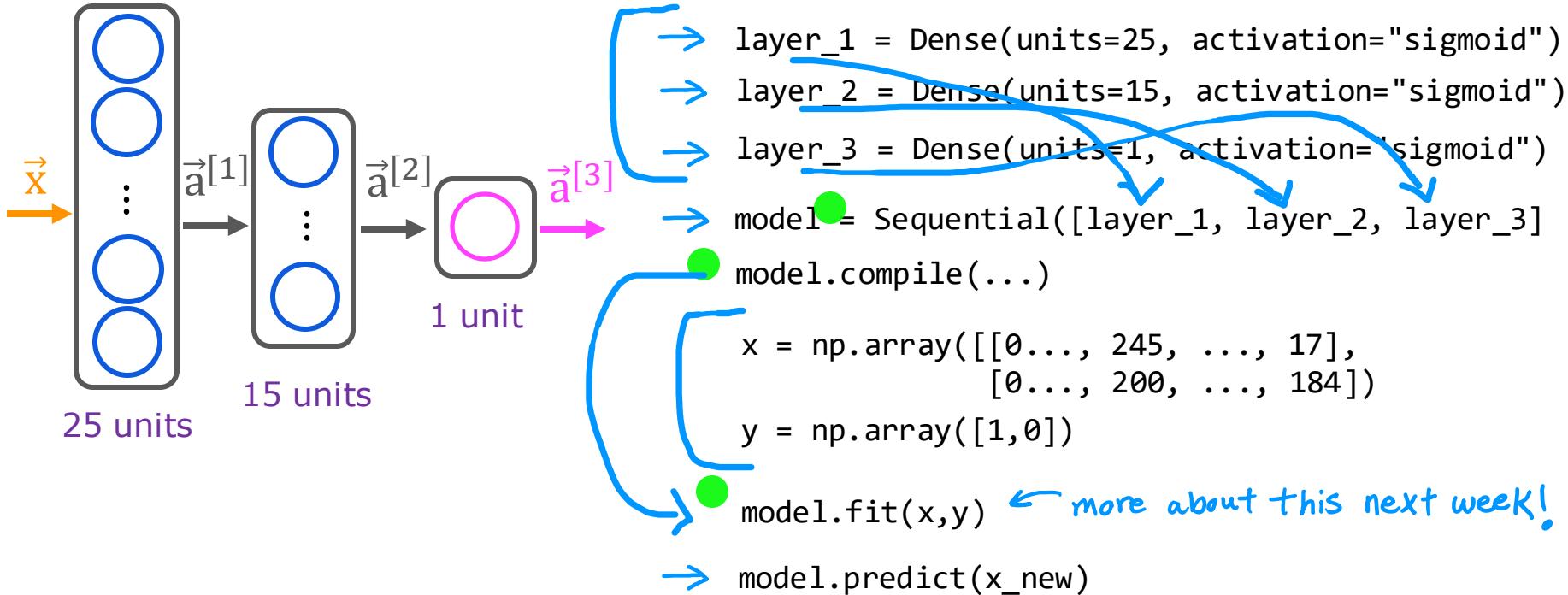
```
model.compile(...)
```

```
model.fit(x,y)
```

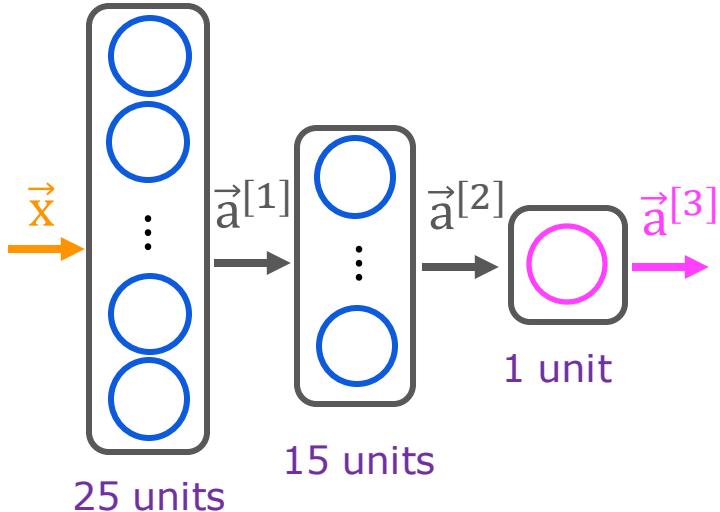
```
→ model.predict(x_new)
```

more about this next week!

# Digit classification model



# Digit classification model



```
model = Sequential([
    Dense(units=25, activation="sigmoid"),
    Dense(units=15, activation="sigmoid"),
    Dense(units=1, activation="sigmoid")])

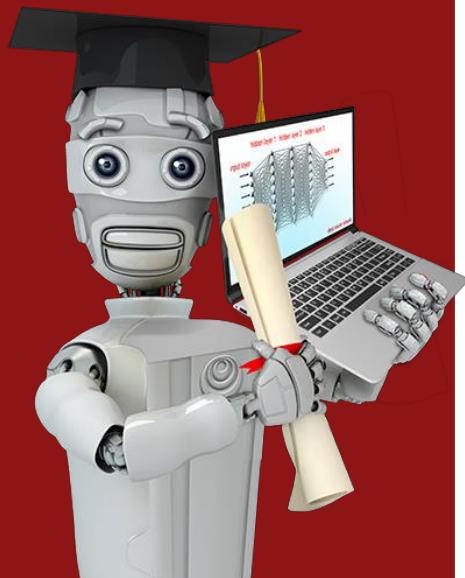
model.compile(...)

x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])

y = np.array([1,0])

model.fit(x,y)

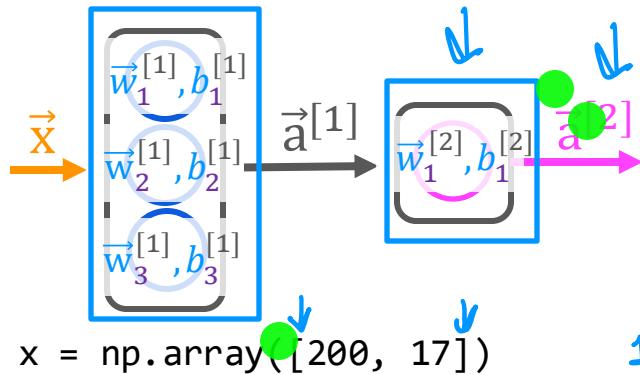
model.predict(x_new)
```



# Neural network implementation in Python

## Forward prop in a single layer

# forward prop (coffee roasting model)



$x = \text{np.array([200, 17])}$

$$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$$

$w1\_1 = \text{np.array([1, 2])}$

$b1\_1 = \text{np.array([-1])}$

$z1\_1 = \text{np.dot}(w1\_1, x) + b$

$a1\_1 = \text{sigmoid}(z1\_1)$

$$a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$$

$w1\_2 = \text{np.array([-3, 4])}$

$b1\_2 = \text{np.array([1])}$

$z1\_2 = \text{np.dot}(w1\_2, x) + b$

$a1\_2 = \text{sigmoid}(z1\_2)$

$$a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$$

$w1\_3 = \text{np.array([5, -6])}$

$b1\_3 = \text{np.array([2])}$

$z1\_3 = \text{np.dot}(w1\_3, x) + b$

$a1\_3 = \text{sigmoid}(z1\_3)$

$$a1 = \text{np.array([a1\_1, a1\_2, a1\_3])}$$

$$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

$w2\_1 = \text{np.array([-7, 8])}$

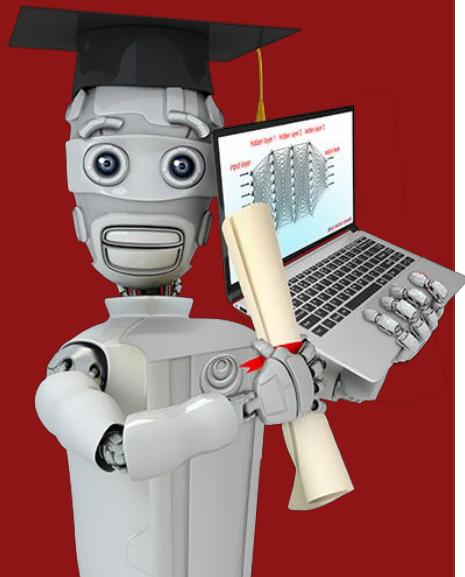
$b2\_1 = \text{np.array([3])}$

$z2\_1 = \text{np.dot}(w2\_1, a1) + b2\_1$

$a2\_1 = \text{sigmoid}(z2\_1)$

$$\vec{w}_1^{[2]} \quad w2\_1$$

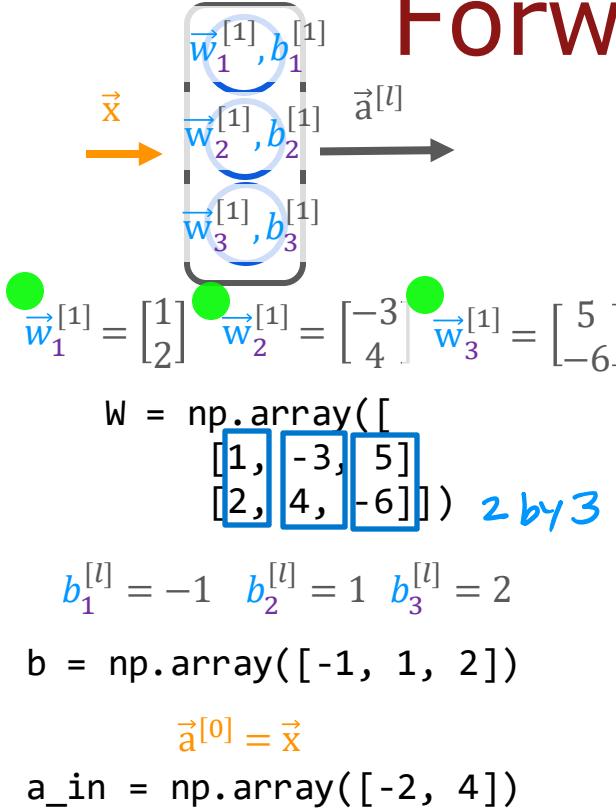
1D arrays



# Neural network implementation in Python

General implementation of  
forward propagation

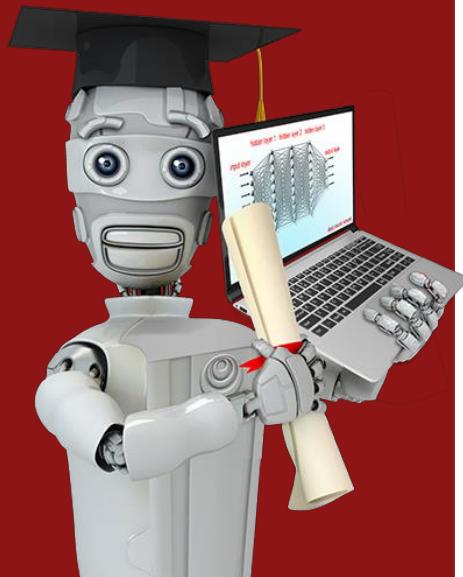
# Forward prop in NumPy



```
def dense(a_in, W, b, g):
    units = W.shape[1] [0,0,0]
    a_out = np.zeros(units)
    for j in range(units): 0,1,2
        w = W[:,j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

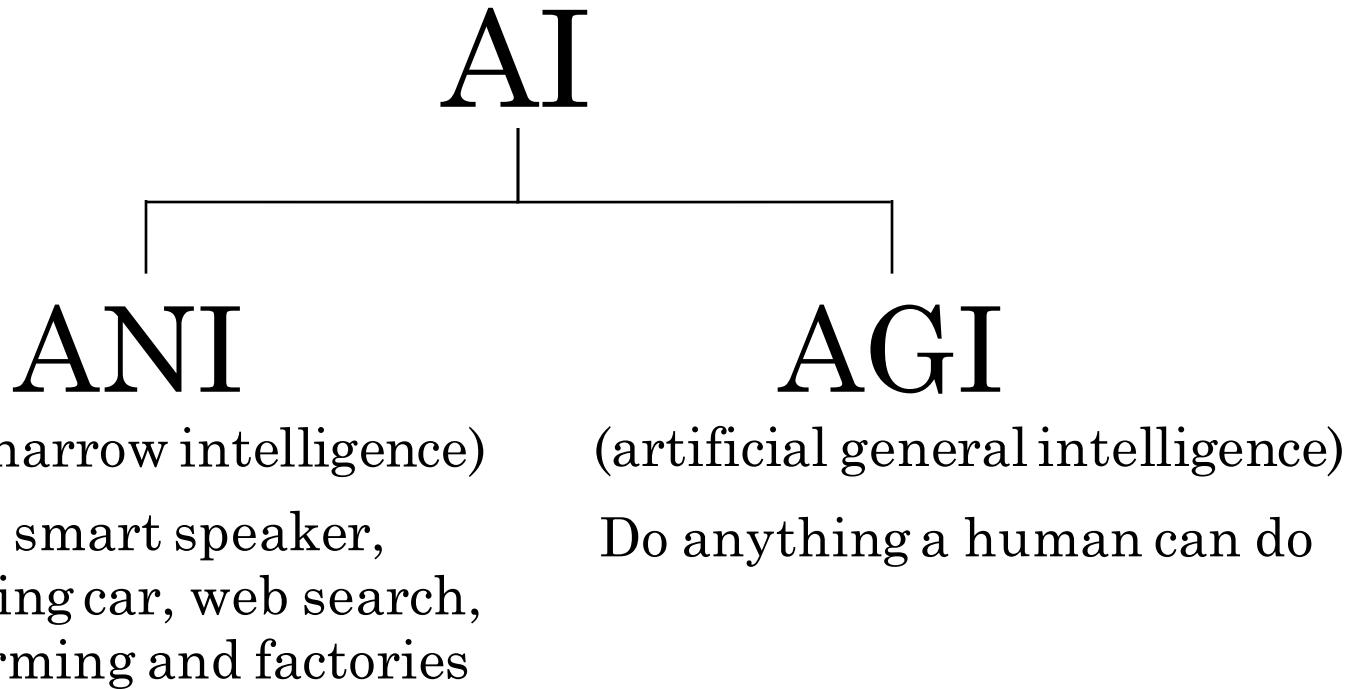
```
def sequential(x):
    a1 = dense(x, w1, b1, g)
    a2 = dense(a1, w2, b2, g)
    a3 = dense(a2, w3, b3, g)
    a4 = dense(a3, w4, b4, g)
    f_x = a4
    return f_x
```

capital W refers to a matrix



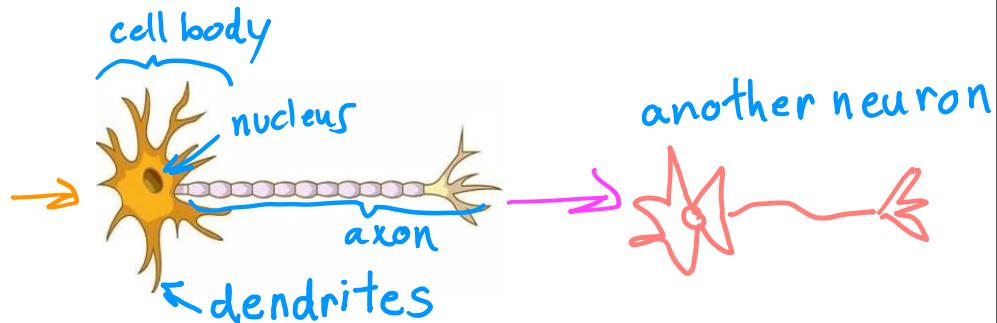
# Speculations on artificial general intelligence (AGI)

Is there a path to AGI?



## Biological neuron

inputs                    outputs



## Simplified mathematical model of a neuron

inputs                    outputs

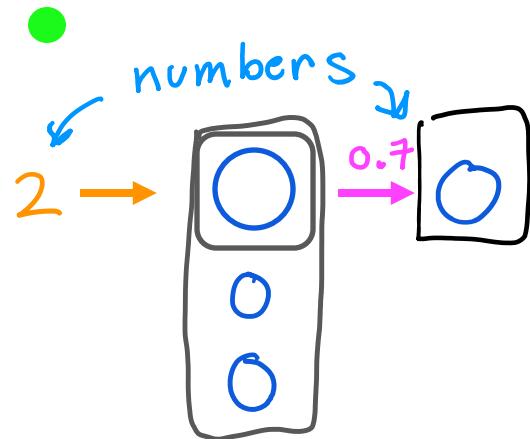
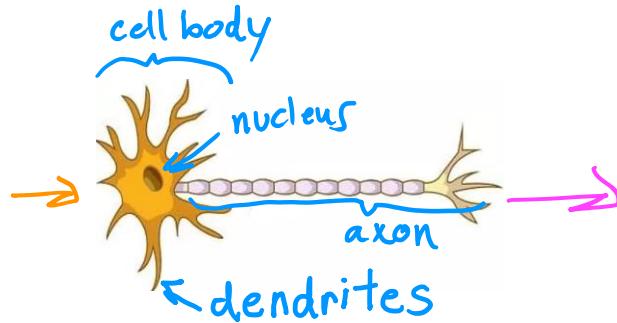


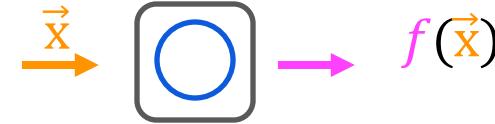
image source: <https://biologydictionary.net/sensory-neuron/>

# Neural network and the brain

Can we mimic the human brain?

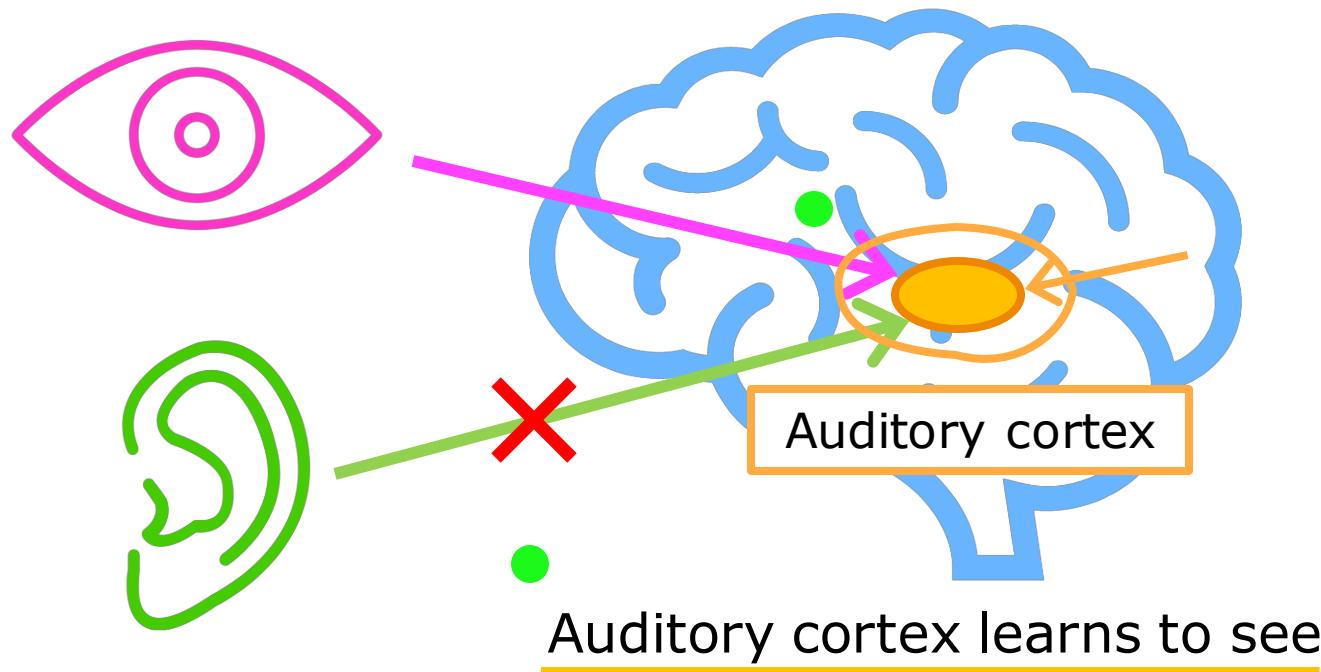


vs



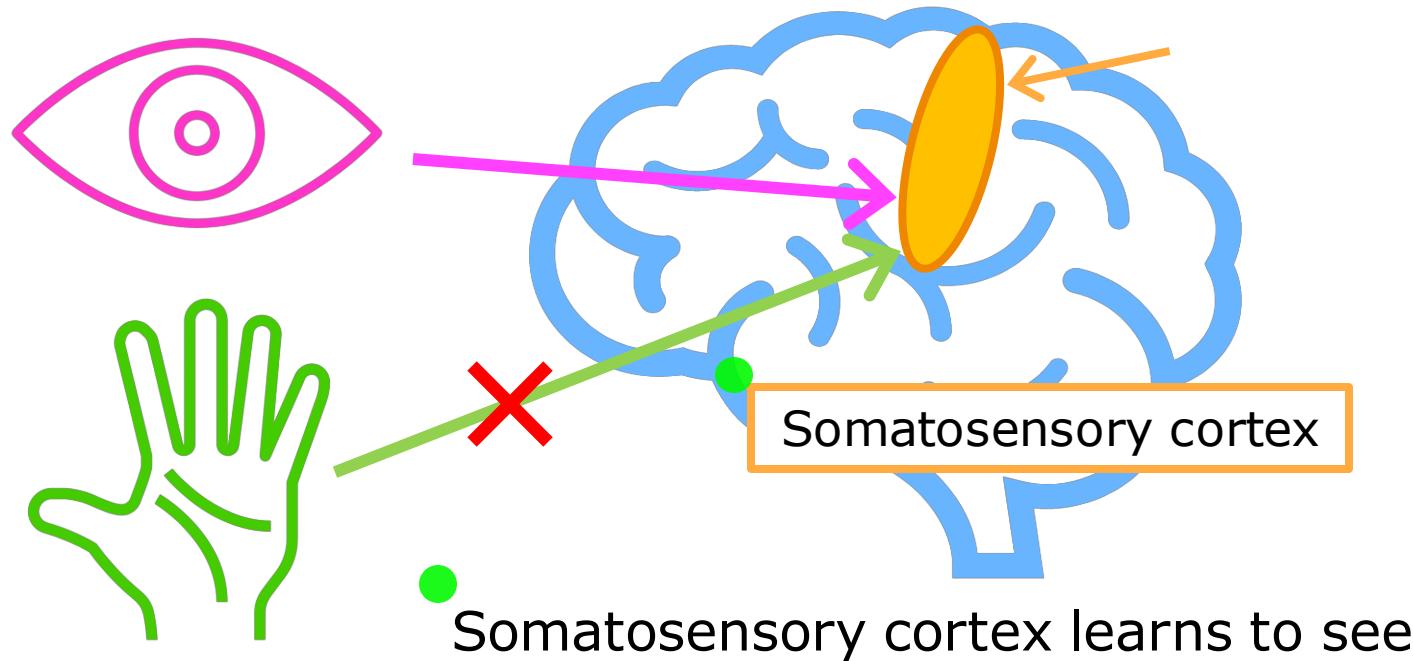
We have (almost) no idea how the brain works

# The “one learning algorithm” hypothesis



[Roe et al., 1992]

# The “one learning algorithm” hypothesis



[Metin & Frost, 1989]

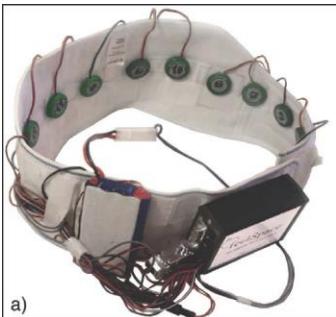
# Sensor representations in the brain



Seeing with your tongue



Human echolocation (sonar)

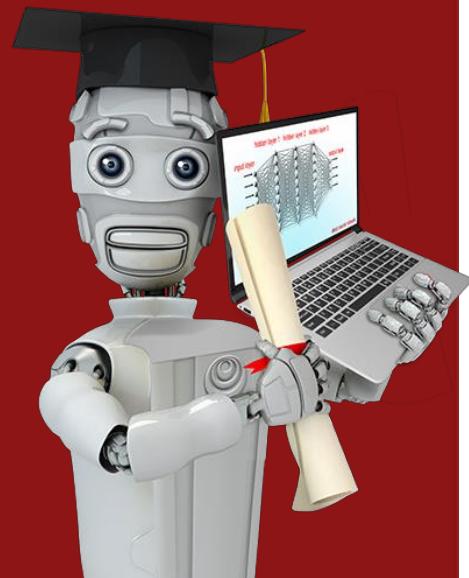


Haptic belt: Direction sense

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]



Implanting a 3<sup>rd</sup> eye



## Vectorization (optional)

**How neural networks are implemented efficiently**

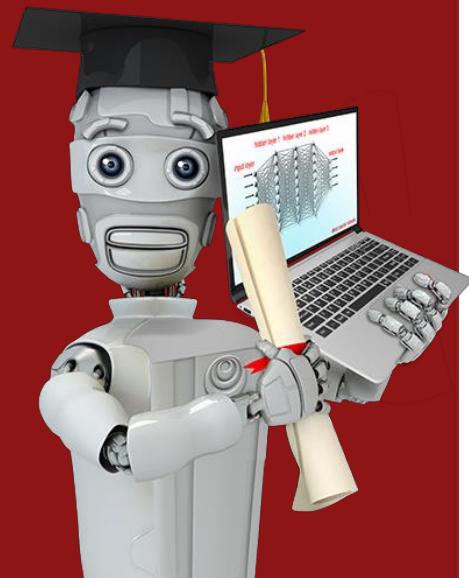
# For loops vs. vectorization

```
x = np.array([200, 17])  
W = np.array([[1, -3, 5],  
             [-2, 4, -6]])  
b = np.array([-1, 1, 2])  
  
def dense(a_in,W,b):  
    a_out = np.zeros(units)  
    for j in range(units):  
        w = W[:,j]  
        z = np.dot(w,x) + b[j]  
        a[j] = g(z)  
    return a
```

vectorized

```
X = np.array([[200, 17]]) 2Darray  
W = np.array([[1, -3, 5],  
             [-2, 4, -6]]) same  
B = np.array([[-1, 1, 2]]) 1x3 2Darray  
  
def dense(A_in,W,B): all 2Darrays  
    Z = np.matmul(A_in,W) + B  
    A_out = g(Z) matrix multiplication  
    return A_out  
  
[[1,0,1]]
```

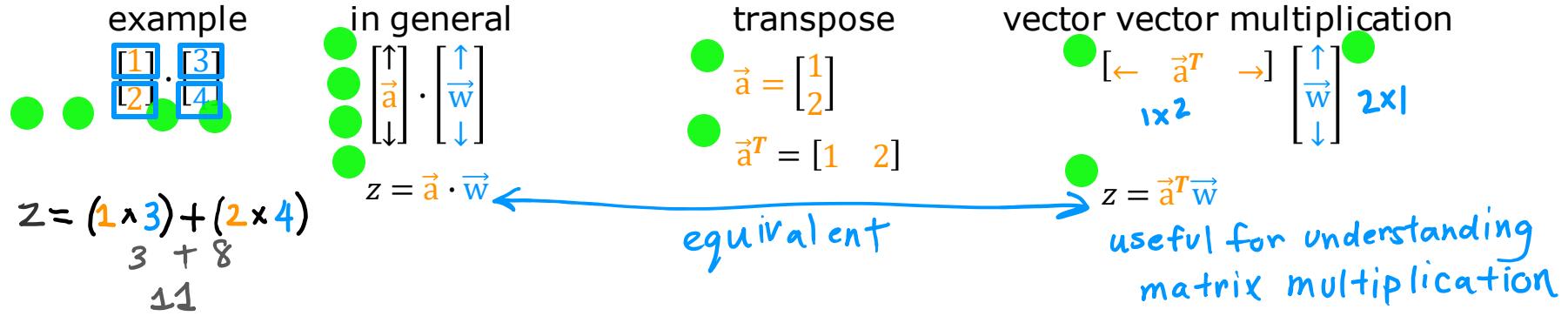
[1,0,1]



## Vectorization (optional)

# Matrix multiplication

# Dot products



# Vector matrix multiplication

$$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\vec{a}^T = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

$$Z = \vec{a}^T W \quad [ \leftarrow \vec{a}^T \rightarrow ] \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

**1 by 2**

$$Z = \begin{bmatrix} \vec{a}^T \vec{w}_1 & \vec{a}^T \vec{w}_2 \end{bmatrix}$$
$$(1 * 3) + (2 * 4) \quad (1 * 5) + (2 * 6)$$
$$3 + 8 \quad 5 + 12$$
$$11 \quad 17$$

$$Z = [11 \quad 17]$$

# matrix matrix multiplication

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 1 & 2 \\ -1 & -2 \end{bmatrix}$$

**rows**

$$\mathbf{A}^T = \begin{bmatrix} 1 & 2 \\ 2 & -2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

**Columns**

$$\mathbf{W} = \begin{bmatrix} \vec{w}_1 \\ \vec{w}_2 \end{bmatrix}$$
$$\mathbf{Z} = \mathbf{A}^T \mathbf{W} = \begin{bmatrix} \leftarrow \stackrel{\rightarrow^T}{\vec{a}_1} \rightarrow \stackrel{\uparrow}{\vec{w}_1} & \leftarrow \stackrel{\rightarrow^T}{\vec{a}_2} \rightarrow \stackrel{\uparrow}{\vec{w}_2} \\ \downarrow & \downarrow \end{bmatrix}$$

$$\begin{array}{c} \text{row1 col1} \\ \text{row2 col1} \\ (\cdot 1 \times 3) + (-2 \times 4) \\ -3 + -8 \\ -11 \end{array} = \begin{bmatrix} \stackrel{\rightarrow^T}{\vec{a}_1} \vec{w}_1 & \stackrel{\rightarrow^T}{\vec{a}_1} \vec{w}_2 \\ \stackrel{\rightarrow^T}{\vec{a}_2} \vec{w}_1 & \stackrel{\rightarrow^T}{\vec{a}_2} \vec{w}_2 \end{bmatrix} \begin{array}{c} \text{row1 col2} \\ \text{row2 col2} \\ (-1 \times 5) + (-2 \times 6) \\ -5 + -12 \\ -17 \end{array}$$
$$= \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix}$$

general rules for  
matrix multiplication  
↳ next video!

## Vectorization (optional)

# Matrix multiplication rules



# Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \\ 0.1 & 0.2 & 1 \end{bmatrix} \vec{a}_1^\top \quad \vec{a}_2^\top \quad \vec{a}_3^\top$$
$$W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad \vec{w}_1 \quad \vec{w}_2 \quad \vec{w}_3 \quad \vec{w}_4$$
$$Z = A^T W = \boxed{\quad}$$

# Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix}$$
$$w = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 0 \end{bmatrix}$$
$$z = A^T w = \begin{bmatrix} \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \end{bmatrix}$$

$$\vec{a}_1^T \vec{w}_1 = (1 \times 3) + (2 \times 4) = 11$$

3 by 4 matrix

row 3 column 2

$$\vec{a}_3^T \vec{w}_2 = (0.1 \times 5) + (0.2 \times 6) = 1.7$$

0.5 + 1.2

row 2 column 3?

$$\vec{a}_2^T \vec{w}_3 = (-1 \times 7) + (-2 \times 8) = -23$$

-7 + -16

# Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

$$\vec{a}_1^T \vec{w}_1 = (1 \times 3) + (2 \times 4) = 11$$

3 by 4 matrix

row 3 column 2

$$\vec{a}_3^T \vec{w}_2 = (0.1 \times 5) + (0.2 \times 6) = 1.7$$

0.5 + 1.2

row 2 column 3?

$$\vec{a}_2^T \vec{w}_3 = (-1 \times 7) + (-2 \times 8) = -23$$

-7 + -16

# Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

$3 \times 2$        $2 \times 4$

can only take dot products  
of vectors that are same length

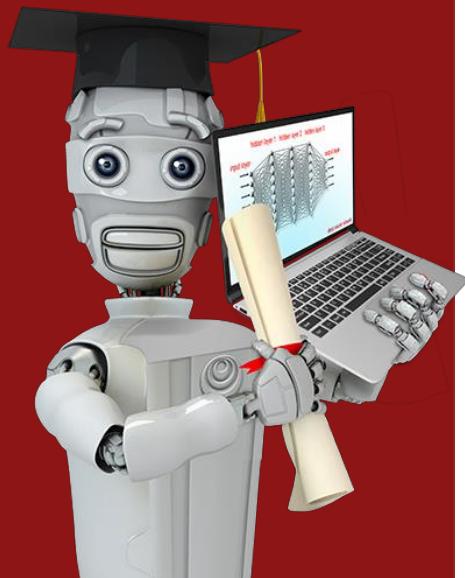
$[0.1 \ 0.2]$   
length 2

$\begin{bmatrix} 5 \\ 6 \end{bmatrix}$   
length 2

3 by 4 matrix  
↳ same # rows as  $A^T$   
same # columns as  $W$

## Vectorization (optional)

# Matrix multiplication code



# Matrix multiplication in NumPy

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & -1 & 0.1 \\ -1 & -2 & 0.2 \\ 0.1 & 0.2 & 1 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

```
A=np.array([1,-1,0.1],  
          [2,-2,0.2]))
```

```
W=np.array([3,5,7,9],  
          [4,6,8,0])
```

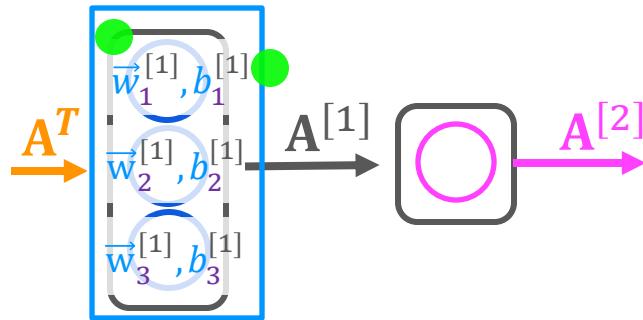
Z = np.matmul(AT,W)  
or  
Z = AT @ W

```
AT=np.array([1,2],  
           [-1,-2],  
           [0.1,0.2])
```

AT=A.T  
transpose

result  
[[11,17,23,9],  
 [-11,-17,-23,-9],  
 [1.1,1.7,2.3,0.9]]

# Dense layer vectorized



$$A^T = [200 \quad 17]$$

$$W = \begin{bmatrix} 1 & -3 & 5 \\ -2 & 4 & -6 \end{bmatrix}$$

$$\vec{b} = \begin{bmatrix} -1 & 1 & 2 \\ 1 & 3 \end{bmatrix}$$

$$Z = A^T W + B$$

$$\begin{bmatrix} 165 \\ -531 \\ 900 \end{bmatrix}$$
$$z_1^{[1]} \quad z_2^{[1]} \quad z_3^{[1]}$$

$$A = g(Z)$$

$$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

A

```
AT = np.array([[200, 17]])
```

```
W = np.array([[1, -3, 5],  
[-2, 4, -6]])
```

```
b = np.array([[-1, 1, 2]])
```

a-in

```
def dense(AT,W,b,g):  
    z = np.matmul(AT,W) + b  
    a_out = g(z)
```

a-in

return a\_out

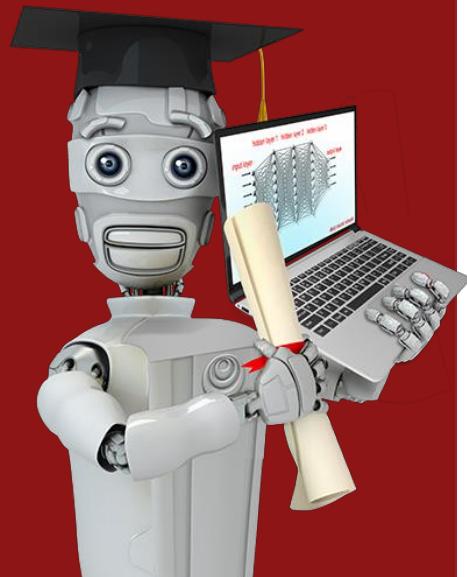
```
[[1,0,1]]
```

# 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>

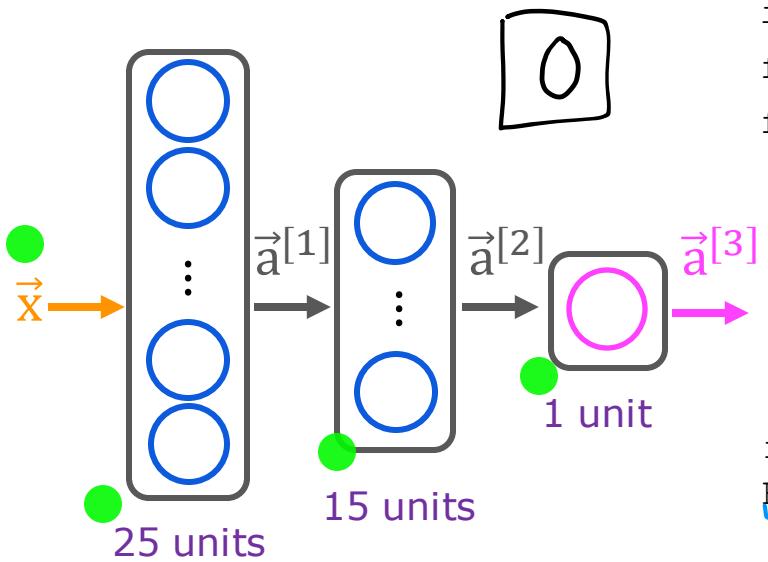


# Neural Network Training

---

TensorFlow  
implementation

# Train a Neural Network in TensorFlow



Given set of  $(x, y)$  examples

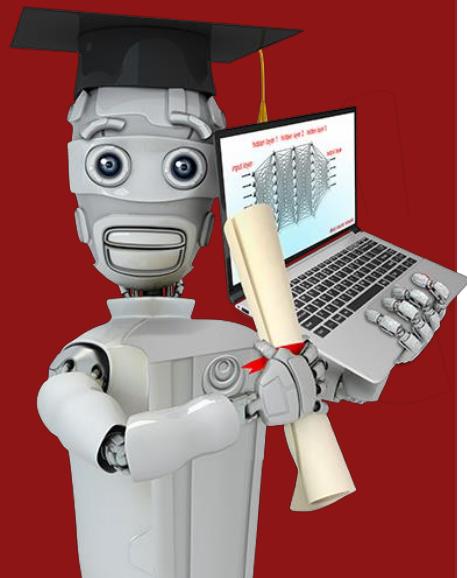
How to build and train this in code?

```
import tensorflow as tf  
from tensorflow.keras import Sequential  
from tensorflow.keras.layers import Dense  
model = Sequential([  
    Dense(units=25, activation='sigmoid'),  
    Dense(units=15, activation='sigmoid'),  
    Dense(units=1, activation='sigmoid')])  
from tensorflow.keras.losses import  
BinaryCrossentropy
```

model.compile(loss=BinaryCrossentropy())

model.fit(X, Y, epochs=100)

③  
epochs: number of steps  
in gradient descent



# Neural Network Training

---

## Training Details

# Model Training Steps

TensorFlow

①

specify how to  
compute output  
given input  $x$  and  
parameters  $w, b$   
(define model)

$$f_{\vec{w}, b}(\vec{x}) = ?$$

②

specify loss and cost

$$L(f_{\vec{w}, b}(\vec{x}), y) \quad 1 \text{ example}$$

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})$$

③

Train on data to  
minimize  $J(\vec{w}, b)$

logistic regression

$$\begin{aligned} z &= np.dot(w, x) + b \\ f_x &= 1 / (1 + np.exp(-z)) \end{aligned}$$

logistic loss

$$\begin{aligned} \text{loss} &= -y * np.log(f_x) \\ &- (1-y) * np.log(1-f_x) \end{aligned}$$

$$\begin{aligned} w &= w - \alpha * dj_dw \\ b &= b - \alpha * dj_db \end{aligned}$$

neural network

```
model = Sequential([  
    Dense(...),  
    Dense(...),  
    Dense(...)])
```

binary cross entropy

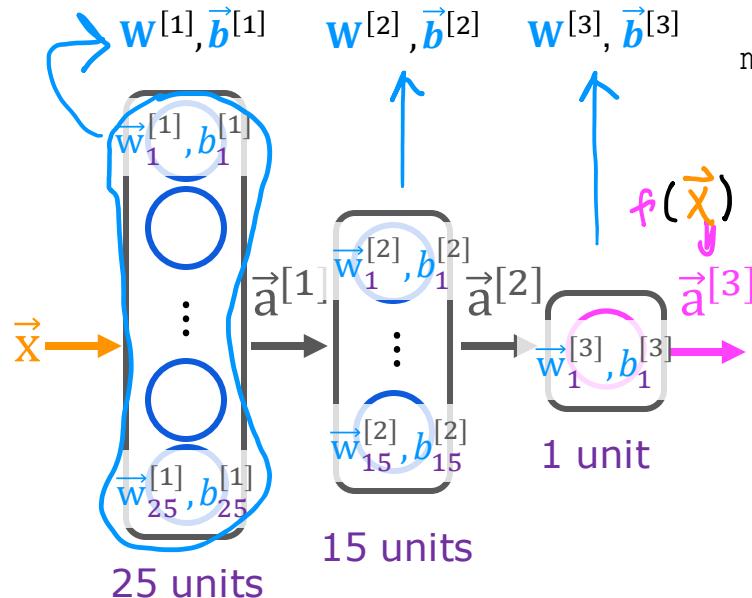
```
model.compile(  
    loss=BinaryCrossentropy())
```

```
model.fit(X, y, epochs=100)
```

# 1. Create the model

define the model

$$f(\vec{x}) = ?$$



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid')
])
```

## 2. Loss and cost functions

Mnist digit classification problem

binary classification

$$L(f(\vec{x}), y) = -y \log(f(\vec{x})) - (1 - y) \log(1 - f(\vec{x}))$$

compare prediction vs. target

logistic loss

also known as binary cross entropy

```
model.compile(loss= BinaryCrossentropy())
```

regression

(predicting numbers  
and not categories)

```
model.compile(loss= MeanSquaredError())
```

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^m L(f(\vec{x}^{(i)}), y^{(i)})$$

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$      $\vec{b}^{[1]}, \vec{b}^{[2]}, \vec{b}^{[3]}$

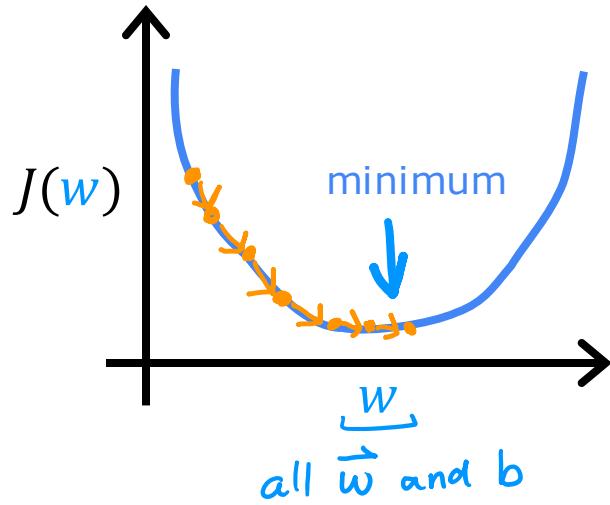
$f_{\mathbf{W}, \mathbf{B}}(\vec{x})$

```
from tensorflow.keras.losses import  
BinaryCrossentropy
```



```
from tensorflow.keras.losses import  
MeanSquaredError
```

# 3. Gradient descent



```
repeat {  
     $w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$   
     $b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$   
}  
} Compute derivatives  
for gradient descent  
using "back propagation"
```

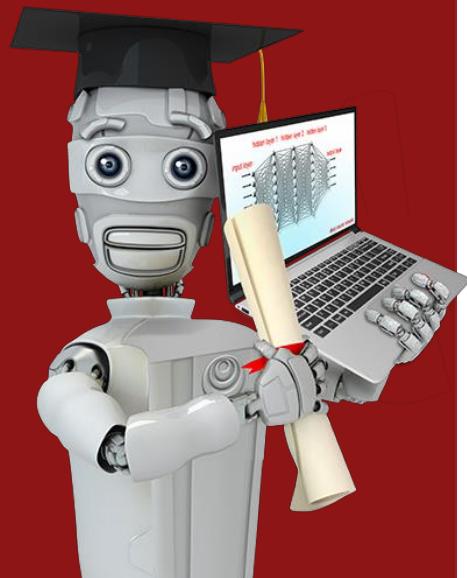
`model.fit(X, y, epochs=100)`

# Neural network libraries

Use code libraries instead of coding "from scratch"



Good to understand the implementation  
(for tuning and debugging).

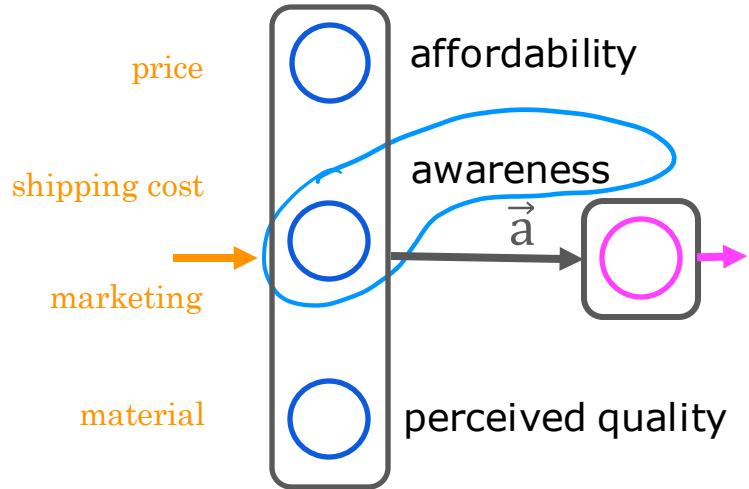


# Activation Functions

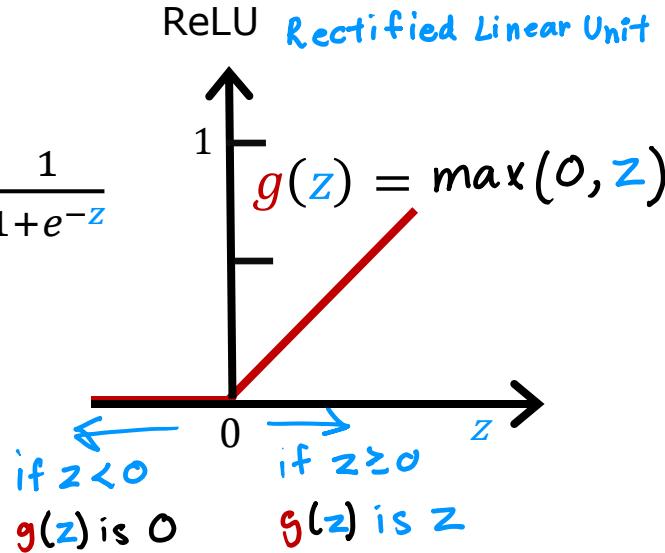
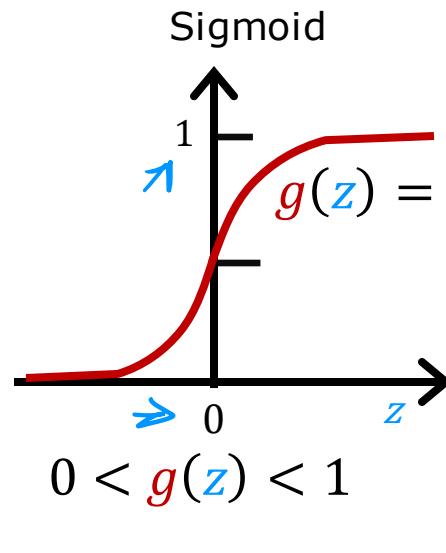
---

Alternatives to the  
sigmoid activation

# Demand Prediction Example



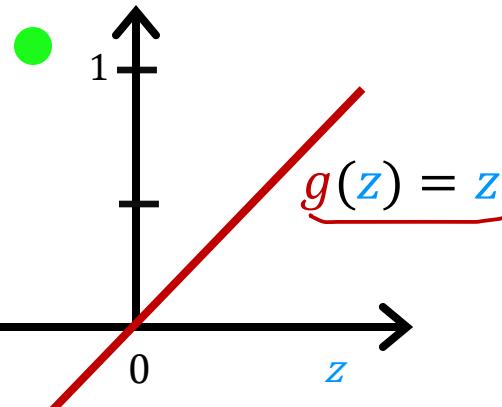
$$a_2^{[1]} = g(\overrightarrow{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$$



# Examples of Activation Functions

"No activation function"

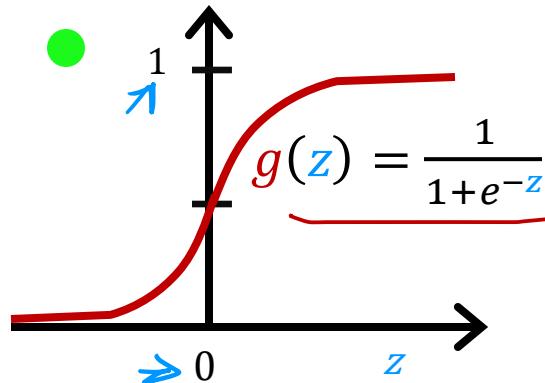
Linear activation function



$$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$$

$$a_2^{[1]} = g(\overrightarrow{w}_2^{[1]} \cdot \vec{x} + \overrightarrow{b}_2^{[1]})$$

Sigmoid

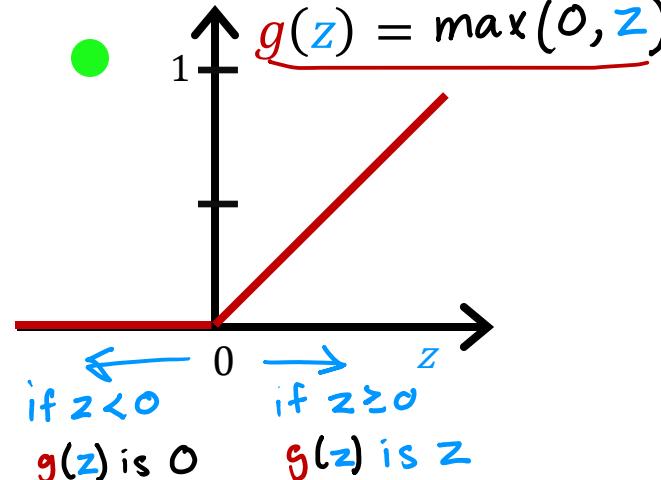


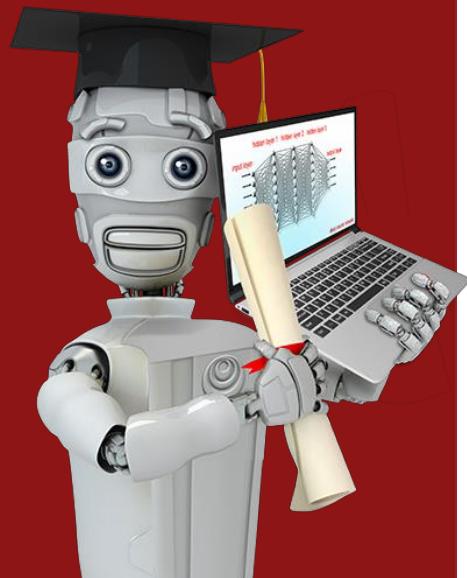
$$0 < g(z) < 1$$

Later: softmax activation

ReLU

Rectified Linear Unit



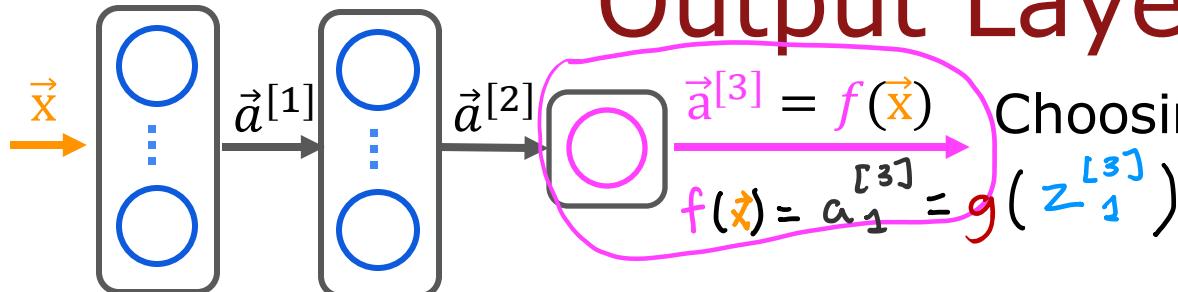


# Activation Functions

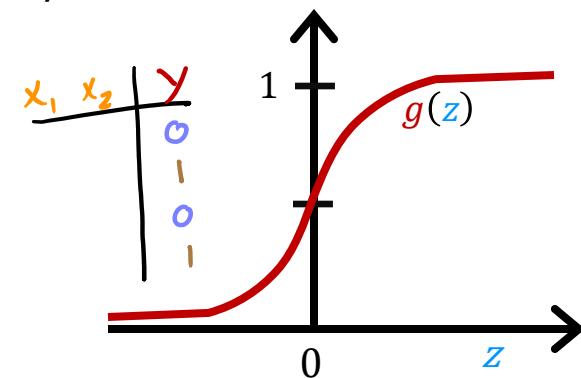
---

## Choosing activation functions

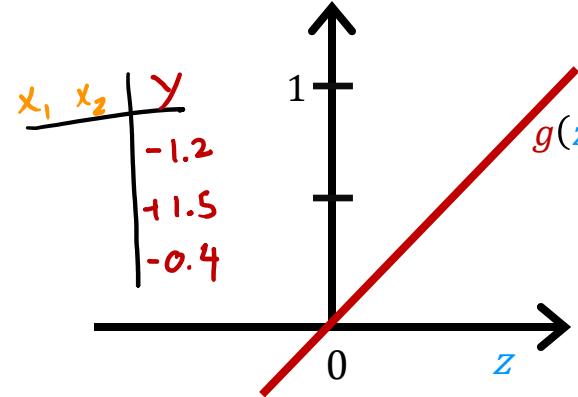
# Output Layer



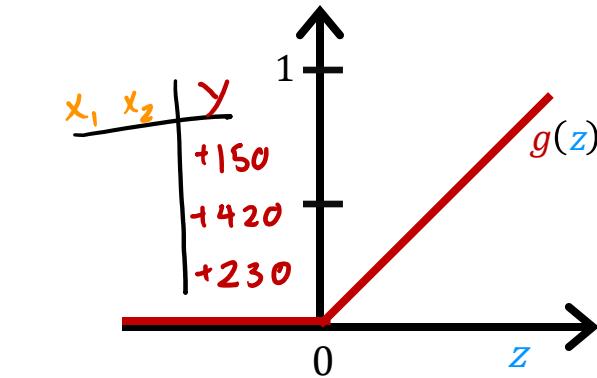
Binary classification  
Sigmoid  
 $y=0/1$



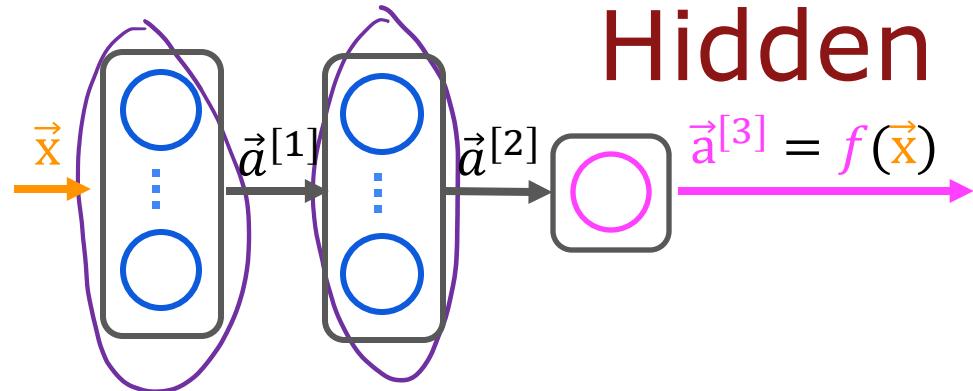
Regression  
Linear activation function  
 $y = +/-$



Regression  
ReLU  
 $y = 0 \text{ or } +$

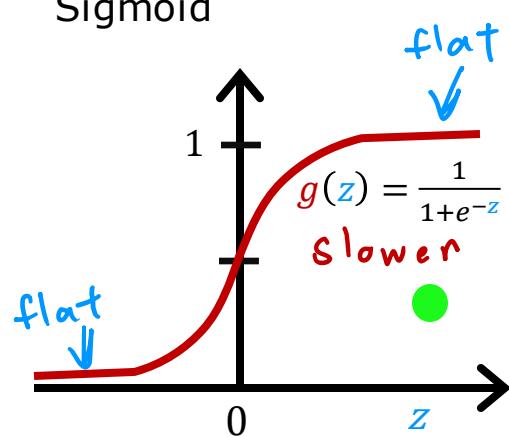


# Hidden Layer

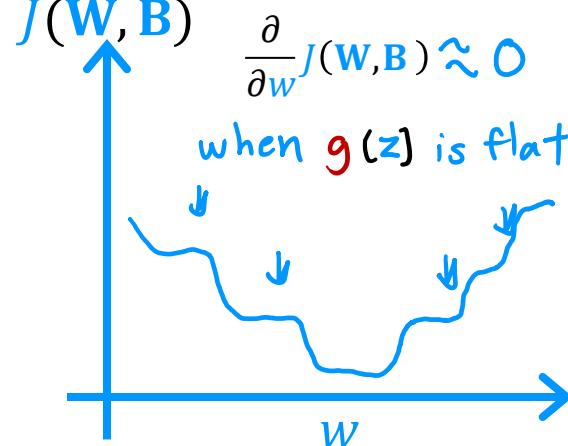


Choosing  $g(z)$  for hidden layer

Sigmoid

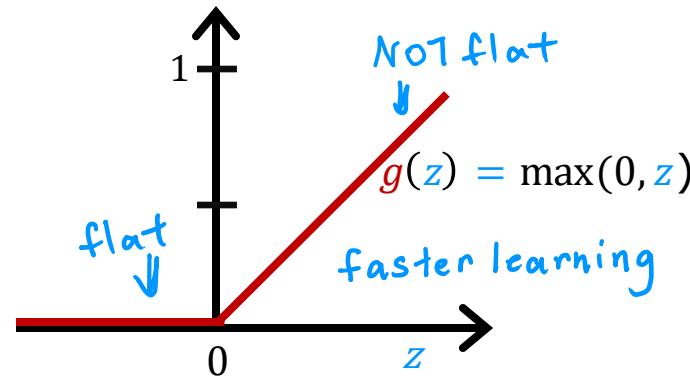


$J(W, B)$

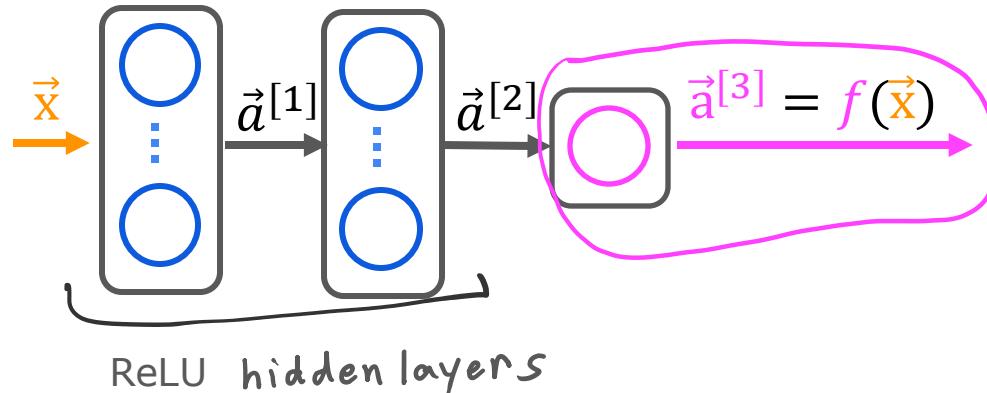


ReLU

faster

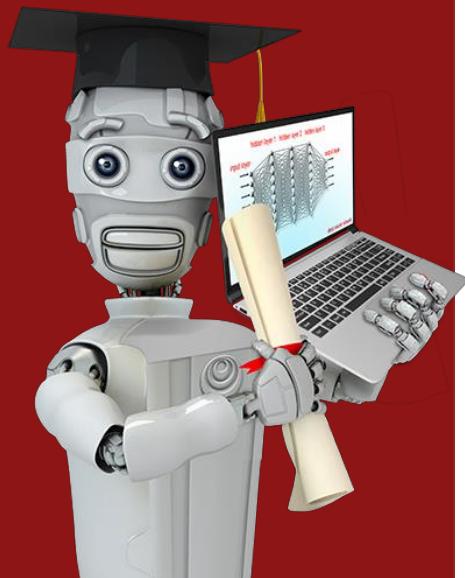


# Choosing Activation Summary



```
from tf.keras.layers import Dense  
model = Sequential([  
    Dense(units=25, activation='relu'), layer1  
    Dense(units=15, activation='relu'), layer2  
    Dense(units=1, activation='sigmoid') layer3  
])  
or 'linear'  
or 'relu'
```

binary classification  
activation='sigmoid'  
regression  $y$  negative/  
positive  
activation='linear'  
regression  $y \geq 0$   
activation='relu'

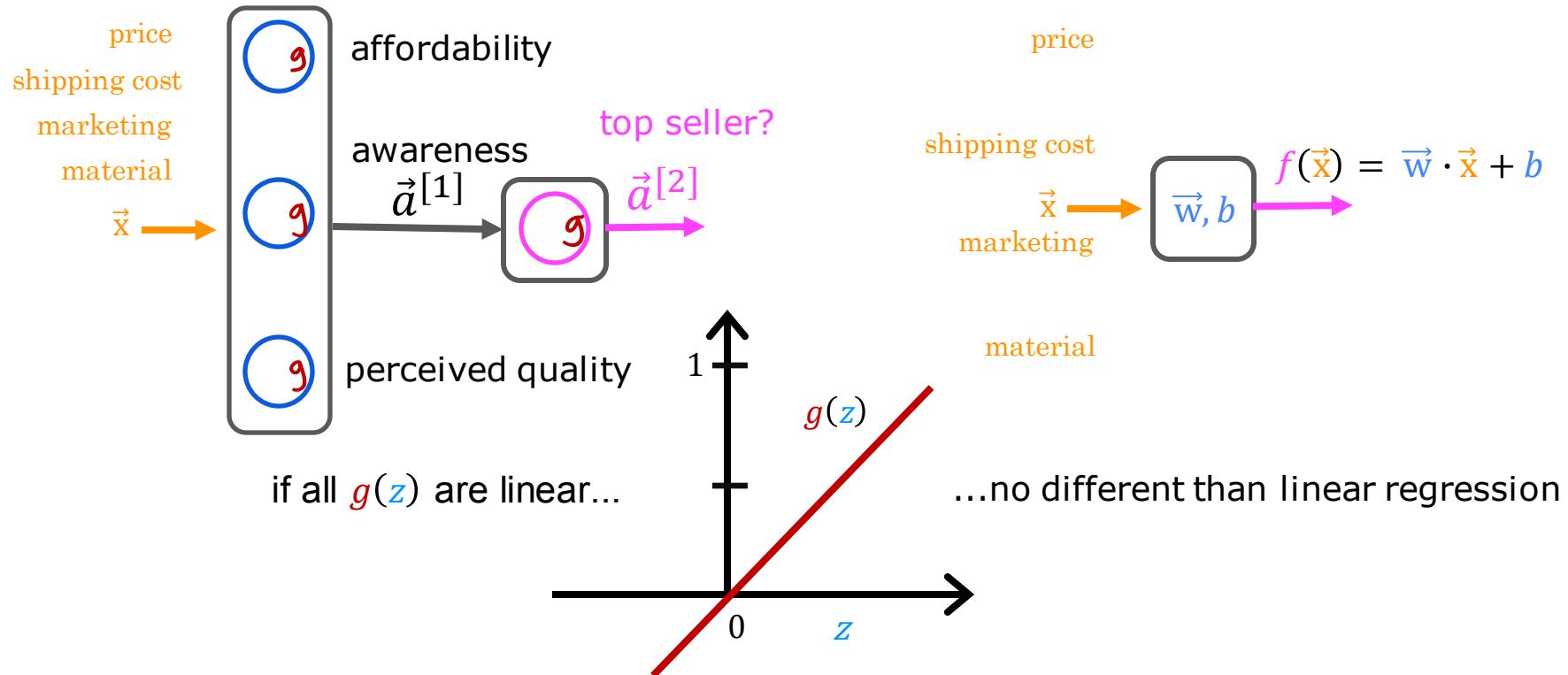


# Activation Functions

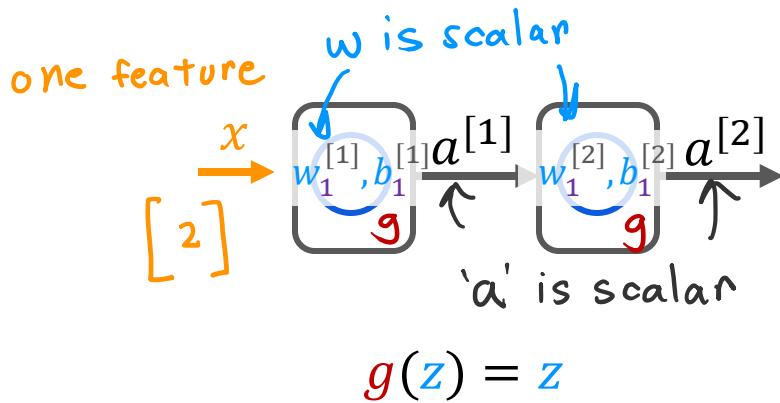
---

Why do we need activation functions?

# Why do we need activation functions?



# Linear Example

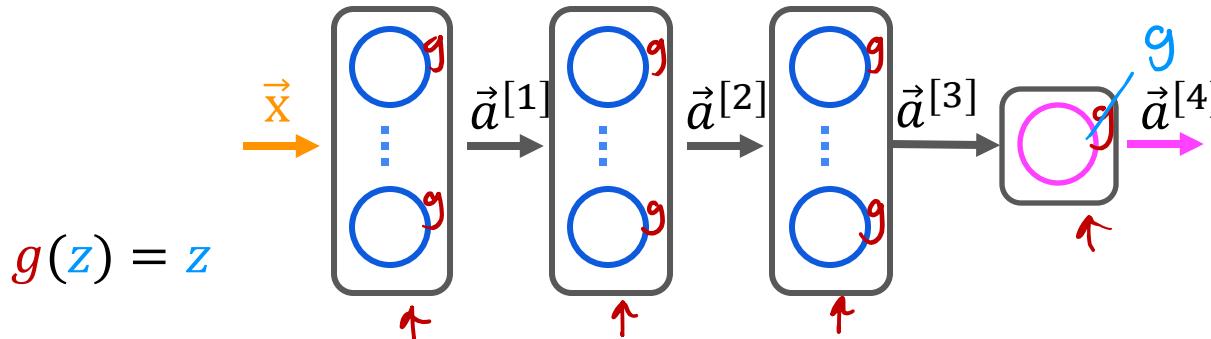


$$\begin{aligned} a^{[1]} &= \underbrace{w_1^{[1]} x}_{\downarrow} + b_1^{[1]} \\ a^{[2]} &= w_1^{[2]} a^{[1]} + b_1^{[2]} \\ &= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]} \\ \vec{a}^{[2]} &= (\underbrace{\vec{w}_1^{[2]} \vec{w}_1^{[1]}}_{\omega}) x + \underbrace{w_1^{[2]} b_1^{[1]}}_{b} + b_1^{[2]} \end{aligned}$$

$$\vec{a}^{[2]} = w x + b$$

$$f(x) = w x + b \text{ linear regression}$$

# Example



$$g(z) = z$$

$$\vec{a}^{[4]} = \vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]}$$

all linear (including output)  
↳ equivalent to linear regression

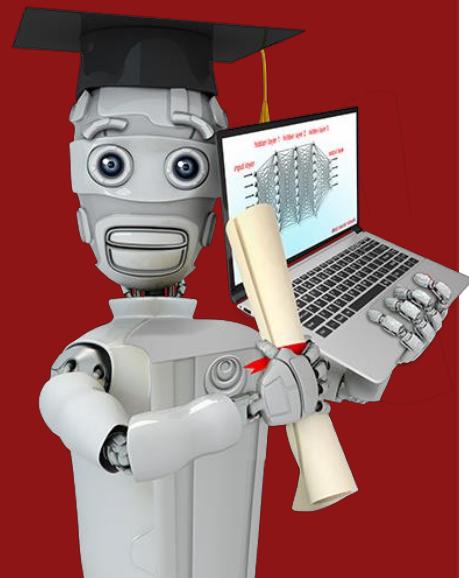
$$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]}\cdot\vec{a}^{[3]}+b_1^{[4]})}}$$

output activation is sigmoid  
(hidden layers still linear)  
↳ equivalent to logistic regression

Don't use linear activations in hidden layers (use ReLU)

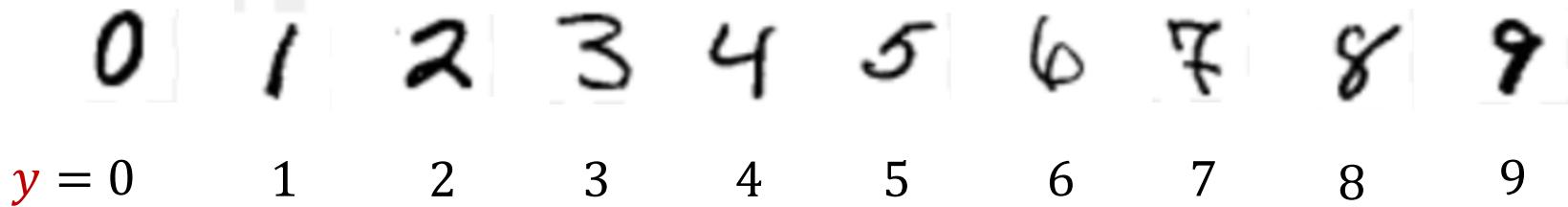
# Multiclass Classification

---



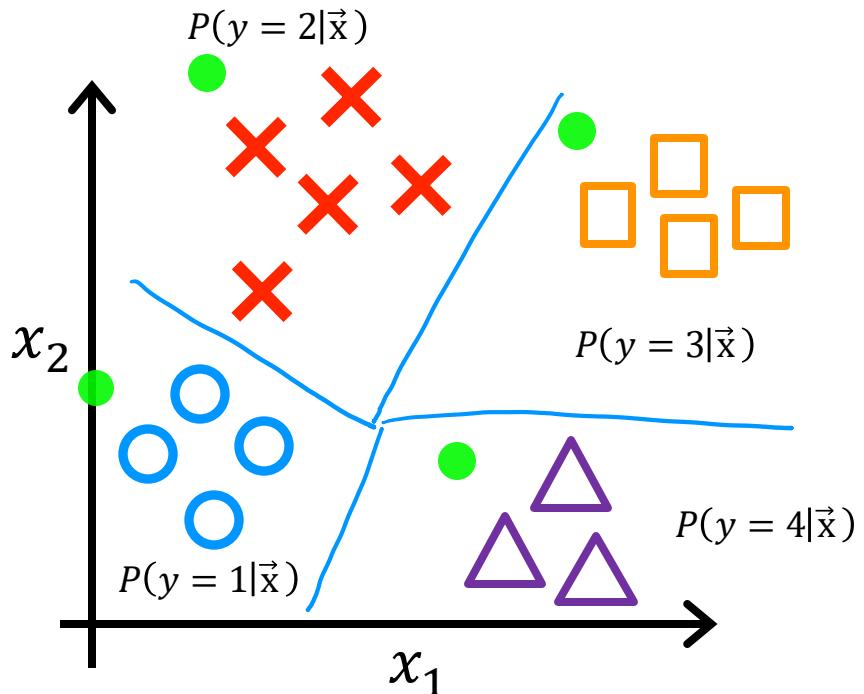
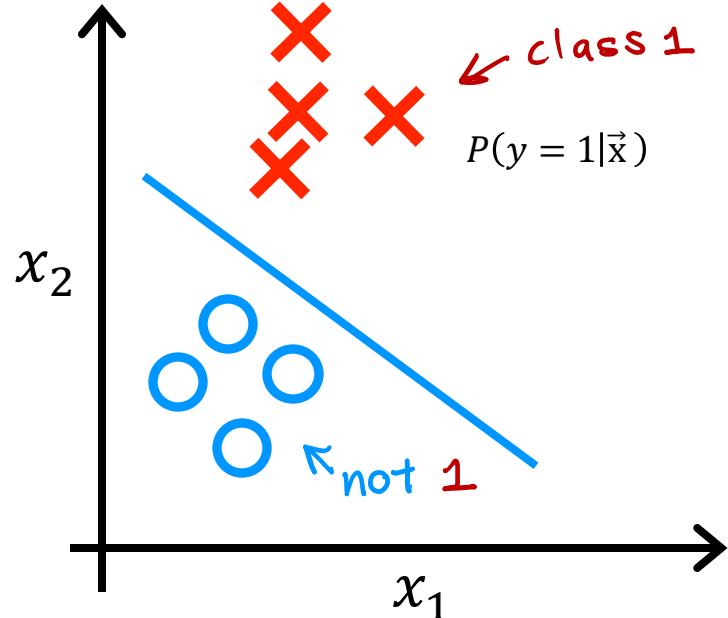
## Multiclass

# MNIST example



multiclass classification problem:  
target  $y$  can take on more than two possible values

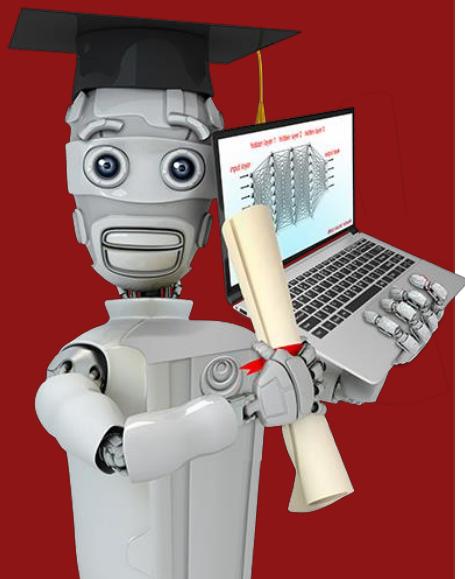
# Multiclass classification example



# Multiclass Classification

---

## Softmax



## Logistic regression (2 possible output values)

$$z = \vec{w} \cdot \vec{x} + b$$

**✗**  $a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$

**○**  $a_2 = 1 - a_1 = P(y=0|\vec{x})$

## Softmax regression (N possible outputs) $y=1, 2, 3, \dots, N$

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \dots, N$$

parameters  $w_1, w_2, \dots, w_N$   
 $b_1, b_2, \dots, b_N$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^N e^{z_k}} = P(y=j|\vec{x})$$

Note:  $a_1 + a_2 + \dots + a_N = 1$

## Softmax regression (4 possible outputs) $y=1, 2, 3, 4$

**✗**  $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

**✗** **○** **□** **△**  
 $= P(y=1|\vec{x})$  **0.30**

**○**  $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
 $= P(y=2|\vec{x})$  **0.20**

**□**  $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$

$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
 $= P(y=3|\vec{x})$  **0.15**

**△**  $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$

$$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
 $= P(y=4|\vec{x})$  **0.35**

# Cost

## Logistic regression

$$z = \vec{w} \cdot \vec{x} + b$$

$$a_1 = g(z) = \frac{1}{1 + e^{-z}} = P(y = 1 | \vec{x})$$

$$a_2 = 1 - a_1 = P(y = 0 | \vec{x})$$

$$\text{loss} = -y \underbrace{\log a_1}_{\text{if } y=1} - (1-y) \underbrace{\log(1-a_1)}_{\text{if } y=0}$$

$$J(\vec{w}, b) = \text{average loss}$$

## Softmax regression

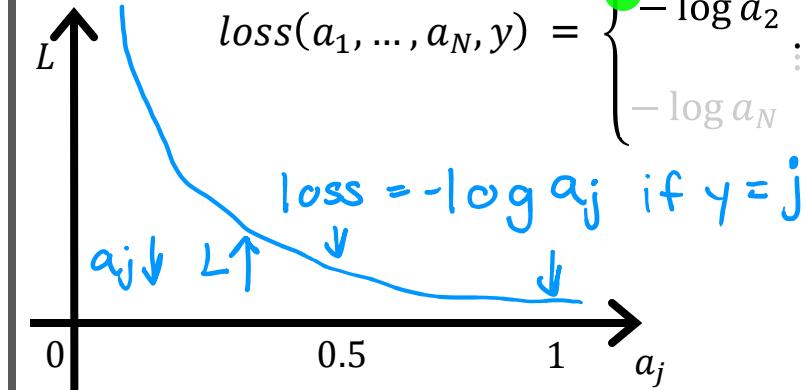
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1 | \vec{x})$$

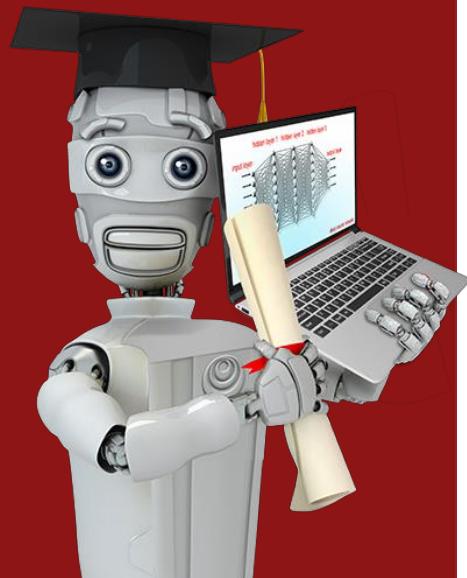
$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = N | \vec{x})$$

### Crossentropy loss

$$\text{loss}(a_1, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \vdots & \vdots \\ -\log a_N & \text{if } y = N \end{cases}$$



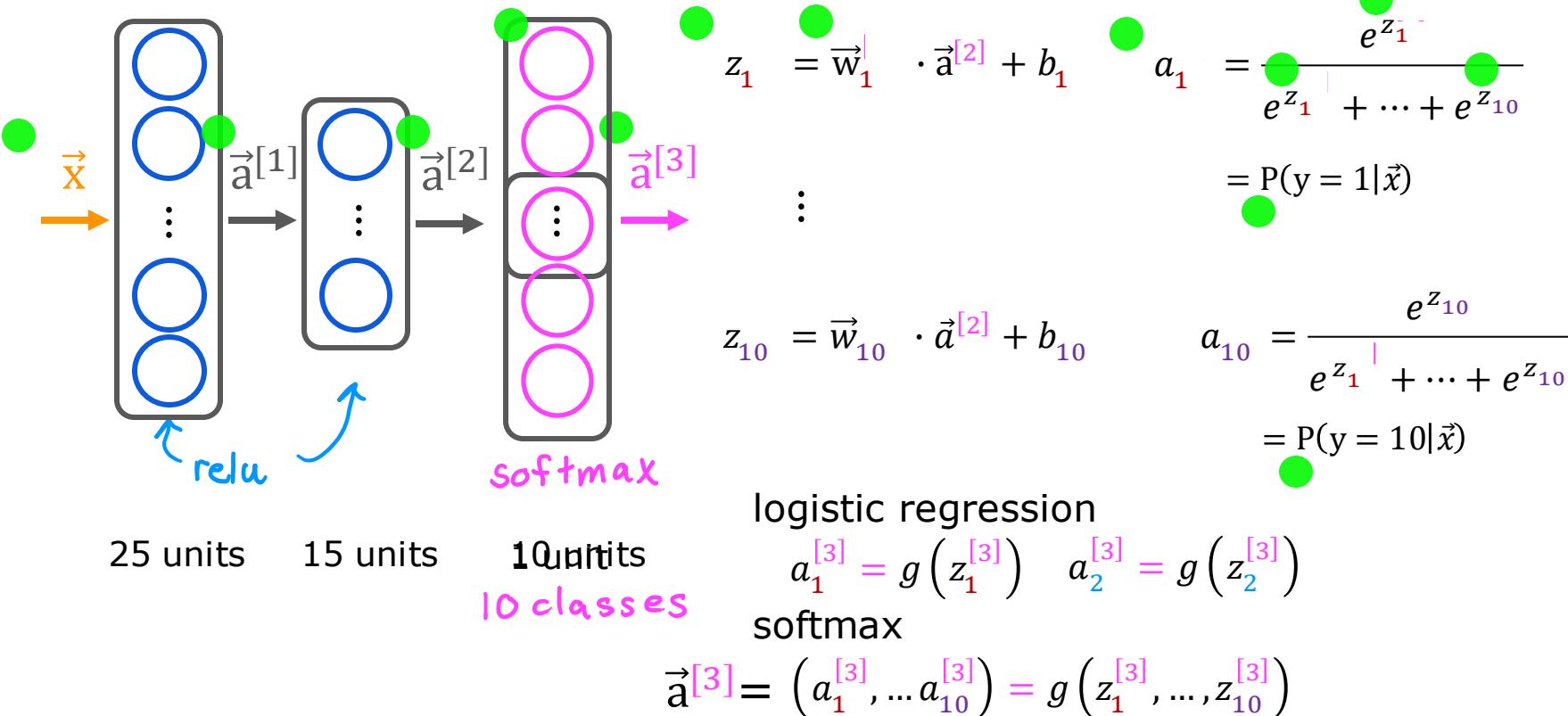


# Multiclass Classification

---

Neural Network with  
Softmax output

# Neural Network with Softmax output



# MNIST with softmax

①

specify the model

$$f_{\vec{w}, b}(\vec{x}) = ?$$

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax')
])
```

②

specify loss and cost

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

```
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
model.fit(X, Y, epochs=100)
```

③

Train on data to  
minimize  $J(\vec{w}, b)$

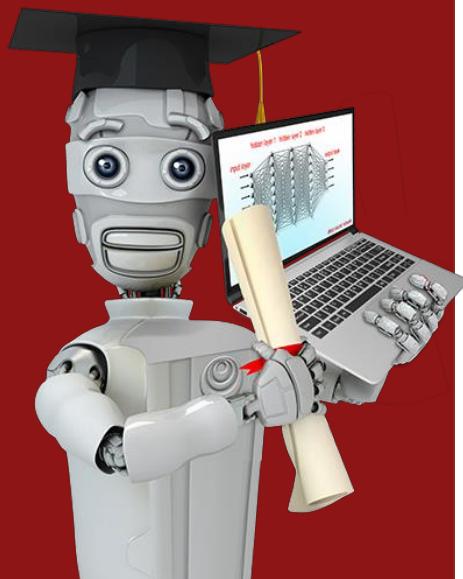
Note: better (recommended) version later.

*Don't use the version shown here!*

# Multiclass Classification

---

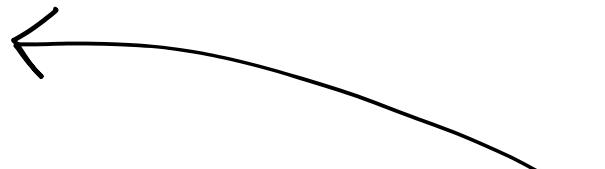
Improved implementation  
of softmax



# Numerical Roundoff Errors

option 1

$$x = \frac{2}{10,000}$$



option 2

$$x = \underbrace{\left(1 + \frac{1}{10,000}\right)} - \underbrace{\left(1 - \frac{1}{10,000}\right)} =$$

# Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

$$1 + \frac{1}{10,000} \quad 1 - \frac{1}{10,000}$$

model = Sequential([

Dense(units=25, activation='relu')

Dense(units=15, activation='relu') **'linear'**

Dense(units=10, activation='sigmoid')

~~model.compile(loss=BinaryCrossEntropy())~~

Logistic regression:

$$a = g(z) = \frac{1}{1 + e^{-z}}$$

Original loss

$$\text{loss} = -y \log(a) - (1 - y) \log(1 - a)$$

~~model.compile(loss=BinaryCrossEntropy(from\_logits=True))~~

More accurate loss (in code)

$$\text{loss} = -y \log\left(\frac{1}{1 + e^{-z}}\right) - (1 - y) \log\left(1 - \frac{1}{1 + e^{-z}}\right)$$

**logit: z**

# More numerically accurate implementation of softmax

Softmax regression

$$(a_1, \dots, a_{10}) = g(z_1, \dots, z_{10})$$

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ \vdots \\ -\log a_{10} & \text{if } y = 10 \end{cases}$$

More Accurate

$$L(\vec{a}, y) = \begin{cases} -\log \frac{e^{z_1}}{e^{z_1} + \dots + e^{z_{10}}} & \text{if } y = 1 \\ \vdots \\ -\log \frac{e^{z_{10}}}{e^{z_1} + \dots + e^{z_{10}}} & \text{if } y = 10 \end{cases}$$

```
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax')])
```

'linear'

~~model.compile(loss=SparseCategoricalCrossEntropy())~~

model.compile(loss=SparseCrossEntropy(from\_logits=True))

# MNIST (more numerically accurate)

model

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='linear') ])
```

loss

```
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(..., loss=SparseCategoricalCrossentropy(from_logits=True))
```

fit

```
model.fit(X, Y, epochs=100)
```

predict

```
logits = model(X) ← not  $a_1 \dots a_{10}$ 
f_x = tf.nn.softmax(logits) is  $z_1 \dots z_{10}$ 
```

# logistic regression (more numerically accurate)

model

```
model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='linear')
])
from tensorflow.keras.losses import
    BinaryCrossentropy
```

loss

```
model.compile(..., BinaryCrossentropy(from_logits=True)) )
model.fit(X,Y,epochs=100)
```

fit

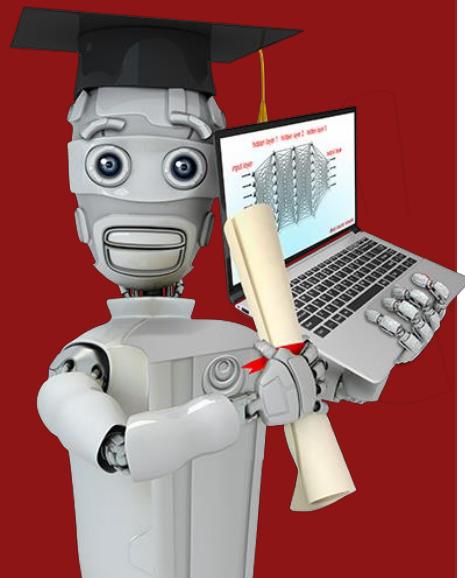
```
logit = model(X)
```

$z$

predict

```
f_x = tf.nn.sigmoid(logit)
```

$\underbrace{\hspace{1cm}}$



# Multi-label Classification

---

Classification with  
multiple outputs  
(Optional)

# Multi-label Classification



Is there a car?

$$\begin{array}{ll} \text{yes} & y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \end{array}$$

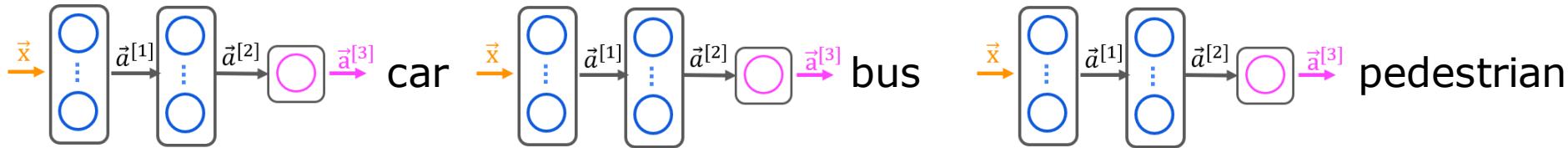
Is there a bus?

$$\begin{array}{ll} \text{no} & y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{array}$$

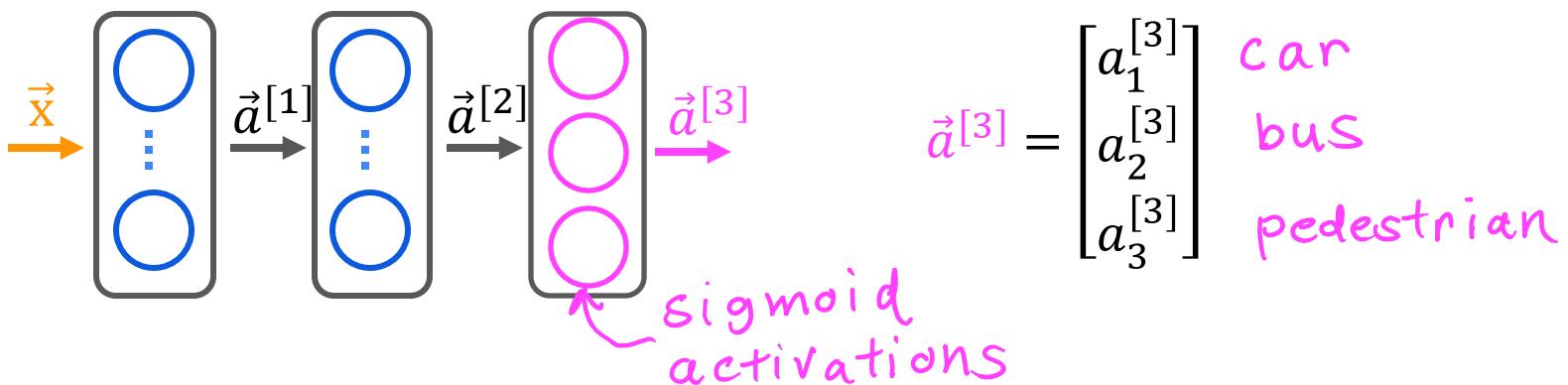
Is there a pedestrian?

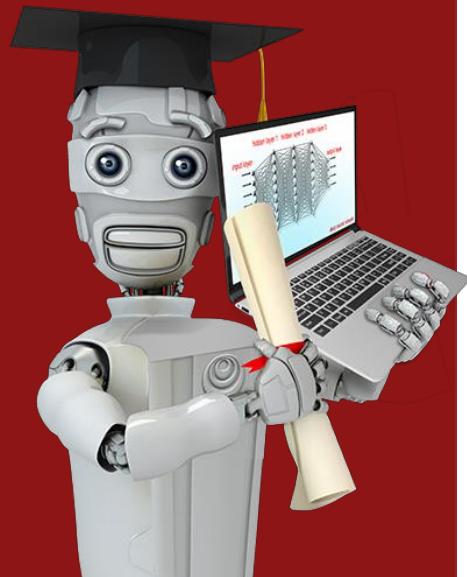
$$\begin{array}{ll} \text{yes} & y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

# Multiple classes



Alternatively, train one neural network with three outputs





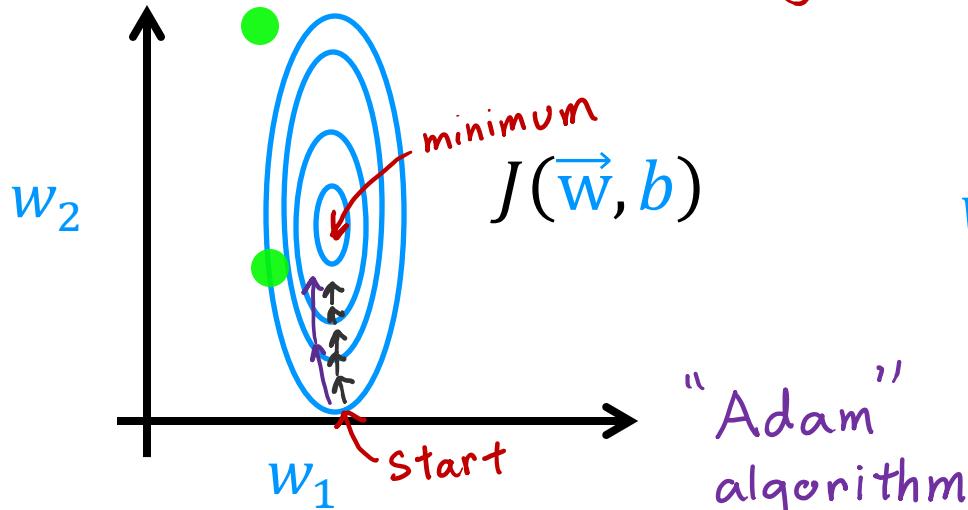
## Additional Neural Network Concepts

# Advanced Optimization

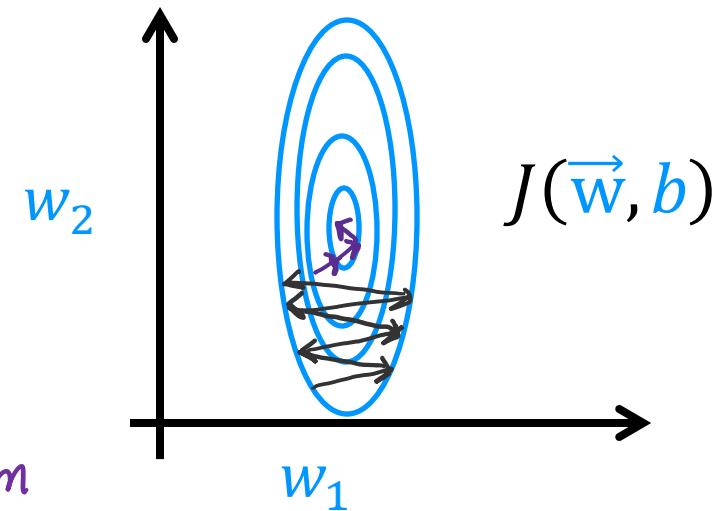
# Gradient Descent

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

↑  
learning rate



Go faster – increase  $\alpha$



Go slower – decrease  $\alpha$

# Adam Algorithm Intuition

Adam: Adaptive Moment estimation    *not just one  $\alpha$*

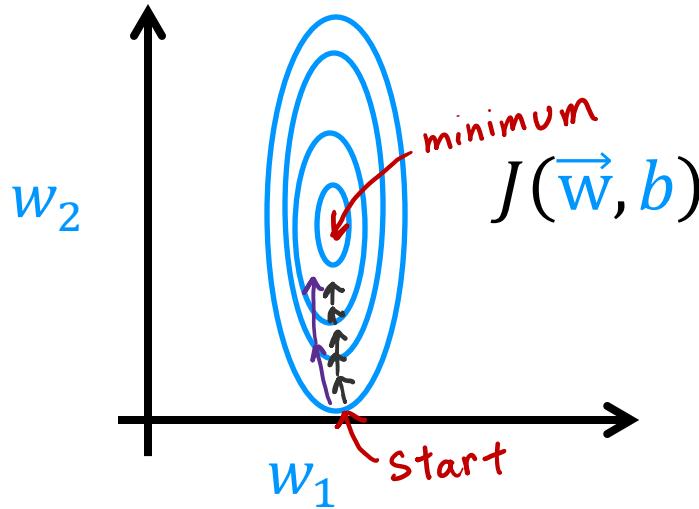
$$w_1 = w_1 - \underbrace{\alpha_1}_{\text{red}} \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

⋮

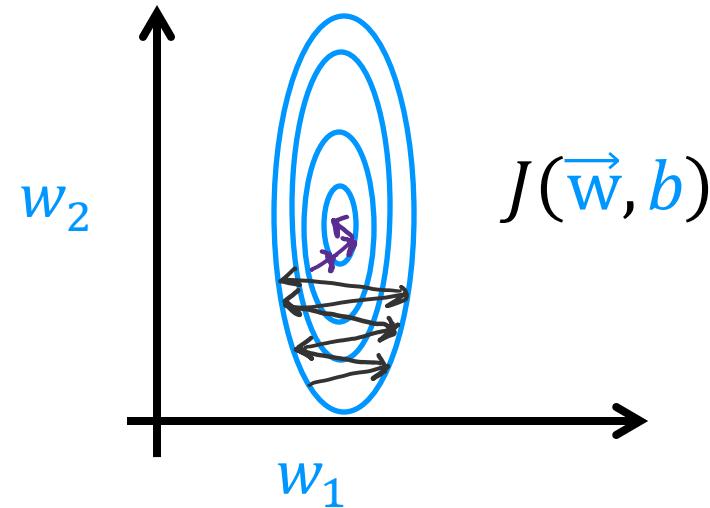
$$w_{10} = w_{10} - \underbrace{\alpha_{10}}_{\text{red}} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$

$$b = b - \underbrace{\alpha_{11}}_{\text{red}} \frac{\partial}{\partial b} J(\vec{w}, b)$$

# Adam Algorithm Intuition



If  $w_j$  (or  $b$ ) keeps moving in same direction, increase  $\alpha_j$ .



If  $w_j$  (or  $b$ ) keeps oscillating, reduce  $\alpha_j$ .

# MNIST Adam

model

```
model = Sequential([
    tf.keras.layers.Dense(units=25, activation='sigmoid')
    tf.keras.layers.Dense(units=15, activation='sigmoid')
    tf.keras.layers.Dense(units=10, activation='linear')
])
```

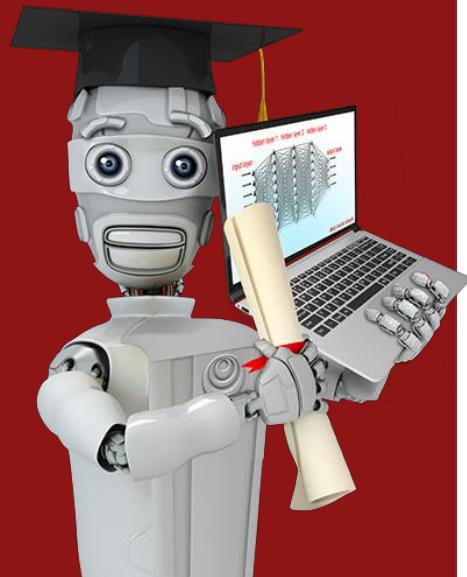
compile

$$\alpha = 10^{-3} = 0.001$$

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

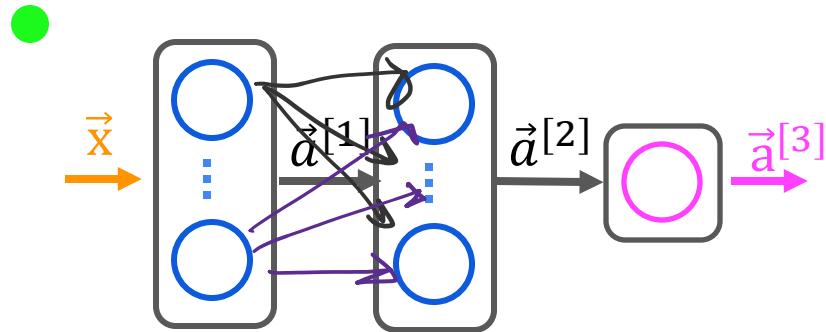
```
model.fit(X, Y, epochs=100)
```



## Additional Neural Network Concepts

### Additional Layer Types

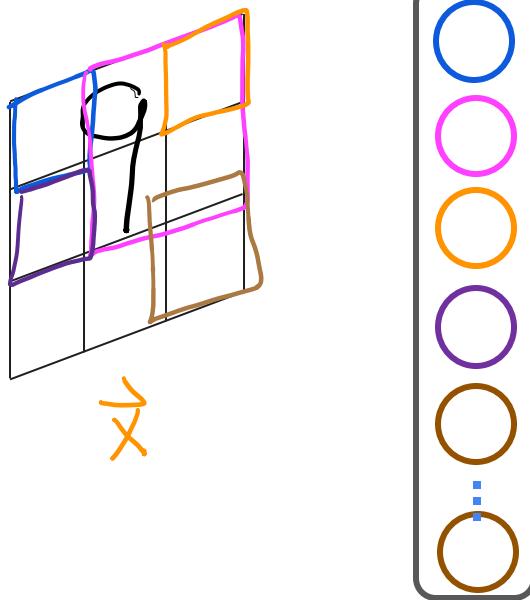
# Dense Layer



Each neuron output is a function of  
all the activation outputs of the previous layer.

$$\bullet \vec{a}_1^{[2]} = g \left( \vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]} \right)$$

# Convolutional Layer

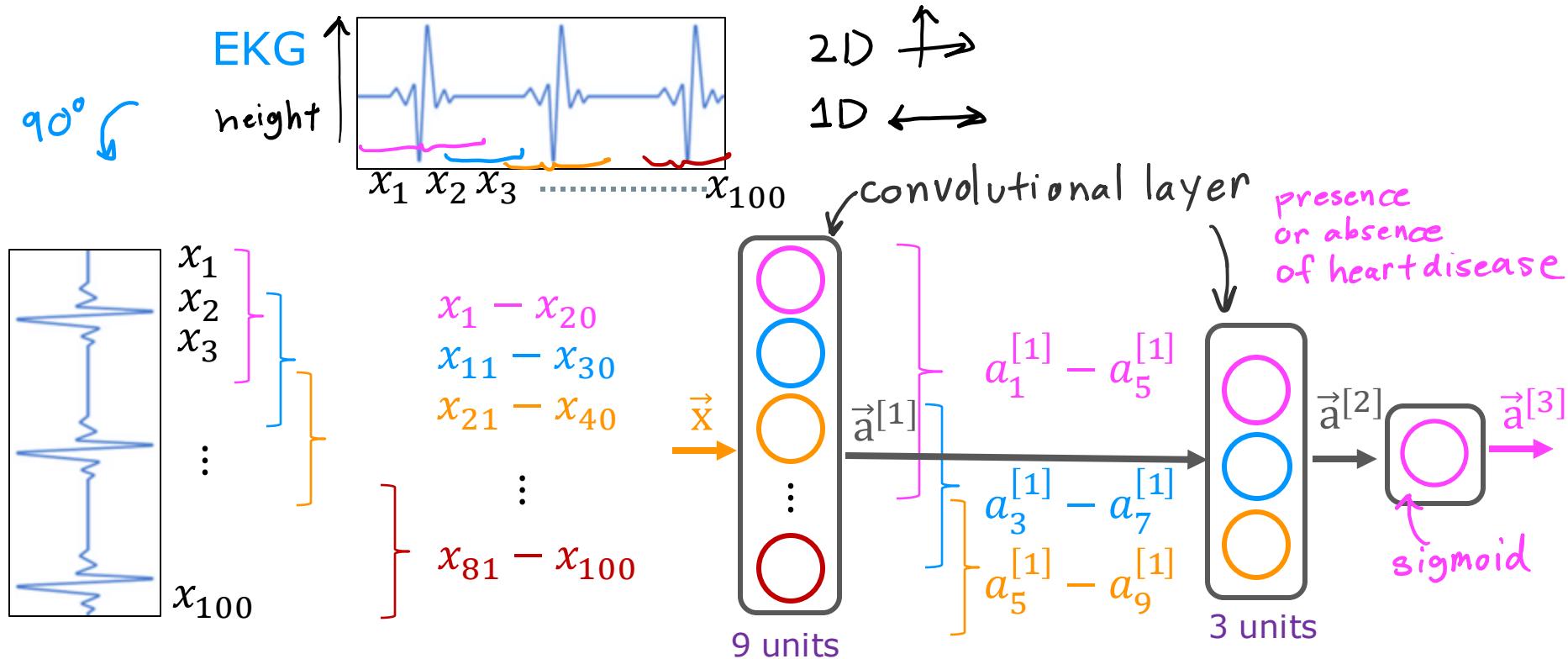


Each Neuron only looks at part of the previous layer's inputs.

Why?

- Faster computation
- Need less training data  
(less prone to overfitting)

# Convolutional Neural Network

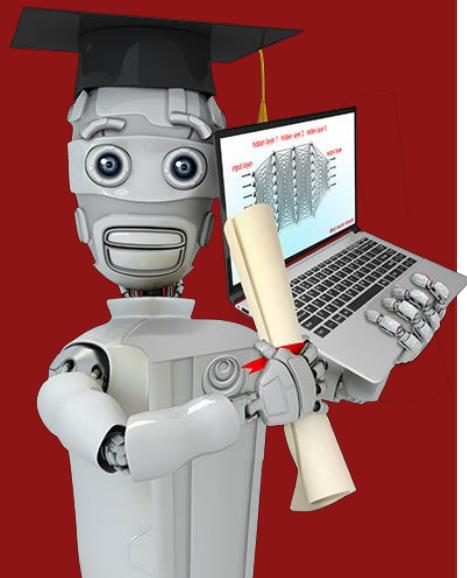


# Copyright Notice

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>



## Advice for applying machine learning

Deciding what to try next

# Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

But it makes unacceptably large errors in predictions. What do you try next?

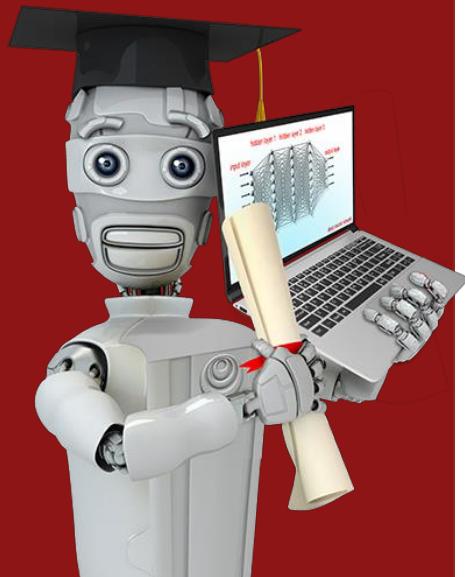
- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features ( $x_1^2, x_2^2, x_1x_2, etc$ )
- Try decreasing  $\lambda$
- Try increasing  $\lambda$

# Machine learning diagnostic

Diagnostic: A test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

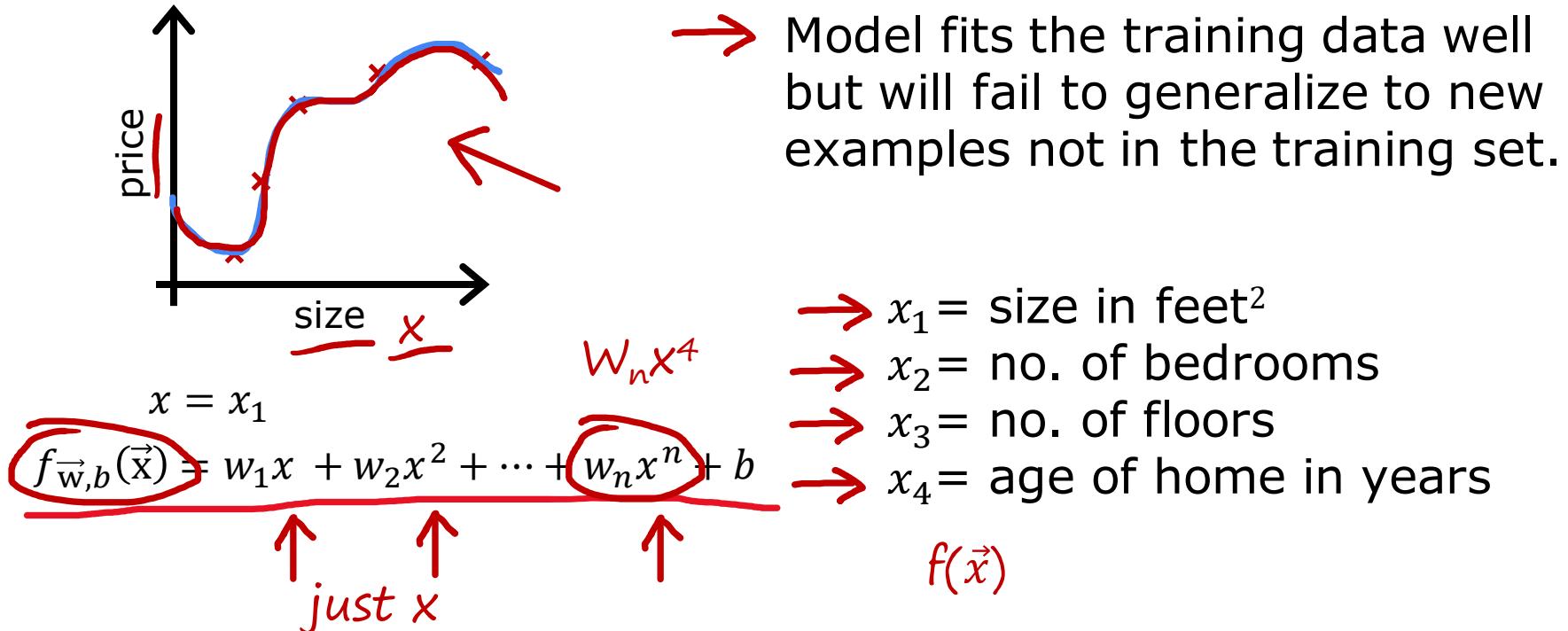
Diagnostics can take time to implement but doing so can be a very good use of your time.

## Evaluating and choosing models



# Evaluating a model

# Evaluating your model



# Evaluating your model

Dataset:

	size	price	
70%	2104	400	$(x^{(1)}, y^{(1)})$
	1600	330	$(x^{(2)}, y^{(2)})$
	2400	369	$\vdots$
	1416	232	$(x^{(m_{train})}, y^{m_{train}})$
	3000	540	
	1985	300	
	1534	315	
30%	1427	199	$(x_{test}^{(1)}, y_{test}^{(1)})$
	1380	212	$\vdots$
	1494	243	$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Diagram illustrating the splitting of a dataset into training and test sets. A red bracket labeled "training set" groups the first 7 rows (70% of the data). A red bracket labeled "test set" groups the last 3 rows (30% of the data). The training set is mapped to a sequence of pairs  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m_{train})}, y^{m_{train}})$ . The test set is mapped to a sequence of pairs  $(x_{test}^{(1)}, y_{test}^{(1)}), (x_{test}^{(2)}, y_{test}^{(2)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$ .

$m_{train}$  = no. training examples = 7

$m_{test}$  = no. test examples = 3

# Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function  $J(\vec{w}, b)$

$$\rightarrow J(\vec{w}, b) = \min_{\vec{w}, b} \left[ \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left( f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^n w_j^2 \right]$$

Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left( f_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)} \right)^2$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left( f_{\vec{w}, b}(\vec{x}_{train}^{(i)}) - y_{train}^{(i)} \right)^2$$

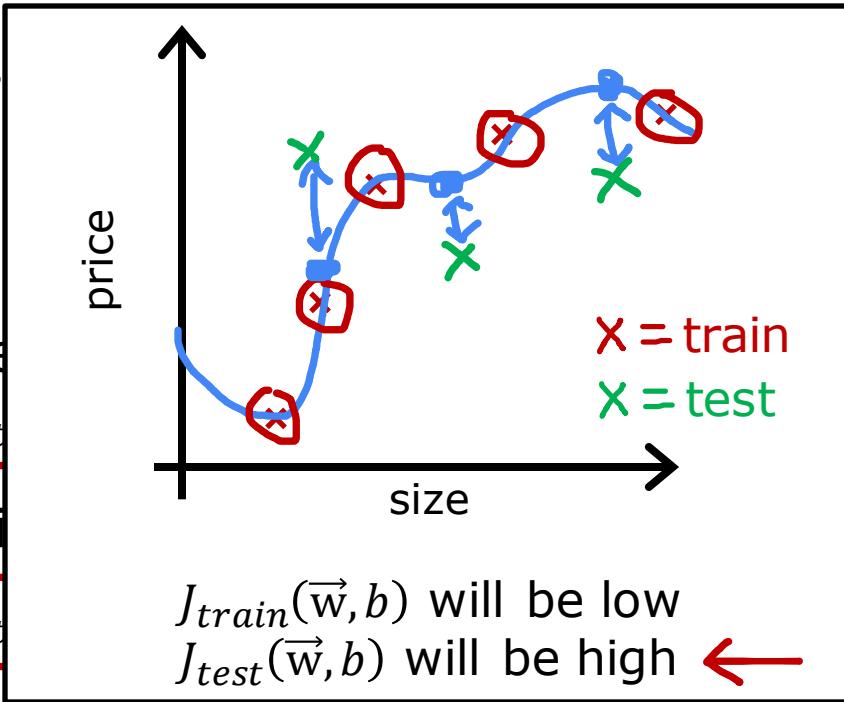
# Train/test procedure for linear regression (with squared error cost)

Fit parameters

$$\rightarrow J(\vec{w}, b) = \min_{\vec{w}, b}$$

Compute test error

Compute training error



$b)$

$$ain \sum_{j=1}^n w_j^2]$$

$$(i) \left( est \right)^2$$

$$- y_{train}^{(i)} \right)^2 \Big]$$

# Train/test procedure for classification problem

O / |

Fit parameters by minimizing  $J(\vec{w}, b)$  to find  $\vec{w}, b$

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[ y_{test}^{(i)} \log(f_{\vec{w}, b}(\vec{x}_{test}^{(i)})) + (1 - y_{test}^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}_{test}^{(i)})) \right]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[ y_{train}^{(i)} \log(f_{\vec{w}, b}(\vec{x}_{train}^{(i)})) + (1 - y_{train}^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}_{train}^{(i)})) \right]$$

# Train/test procedure for classification problem

O / 1

Fit parameters by minimizing  $J(\vec{w}, b)$  to find  $\vec{w}, b$

E.g.,

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

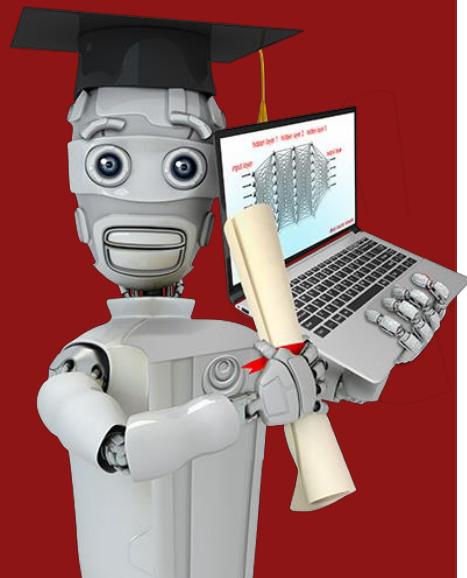
fraction of the test set and the fraction of the train set that the algorithm has misclassified.

$$\hat{y} = \begin{cases} 1 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

count  $\hat{y} \neq y$

$J_{test}(\vec{w}, b)$  is the fraction of the test set that has been misclassified.

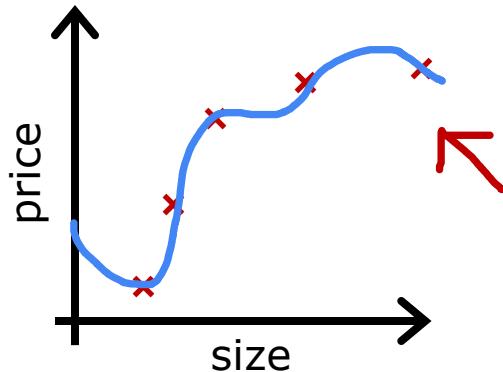
$J_{train}(\vec{w}, b)$  is the fraction of the train set that has been misclassified.



## Evaluating and choosing models

Model selection and training/cross validation/test sets

# Model selection (choosing a model)



$$f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

Once parameters  $\vec{w}, b$  are fit to the training set, the training error  $J_{train}(\vec{w}, b)$  is likely lower than the actual generalization error.

$J_{test}(\vec{w}, b)$  is better estimate of how well the model will generalize to new data than  $J_{train}(\vec{w}, b)$ .

# Model selection (choosing a model)

$d=1$

$$1. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + b$$

$d=2$

$$2. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + b$$

$d=3$

$$3. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + b$$

$\vdots$

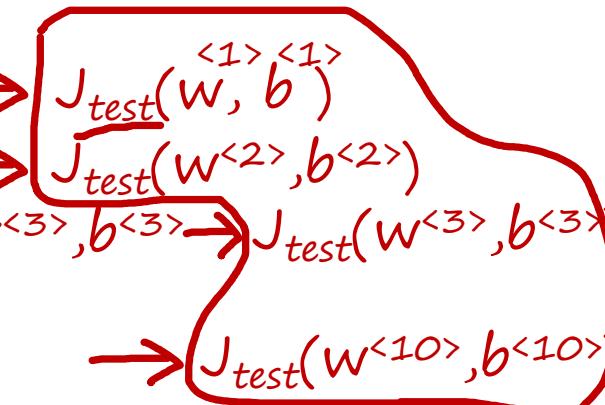
$d=10$

$$10. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + \dots + w_{10} x^{10} + b$$

$$\xrightarrow{w, b^{<1>} \atop w, b^{<2>}} \xrightarrow{w, b^{<1>} \atop w, b^{<2>}}$$

$$\xrightarrow{w^{<3>}, b^{<3>}} \xrightarrow{w^{<3>}, b^{<3>}}$$

$$\xrightarrow{w^{<10>}, b^{<10>}}$$



Choose  $w_1 x_1 + \dots + w_5 x^5 + b$   $d=5$

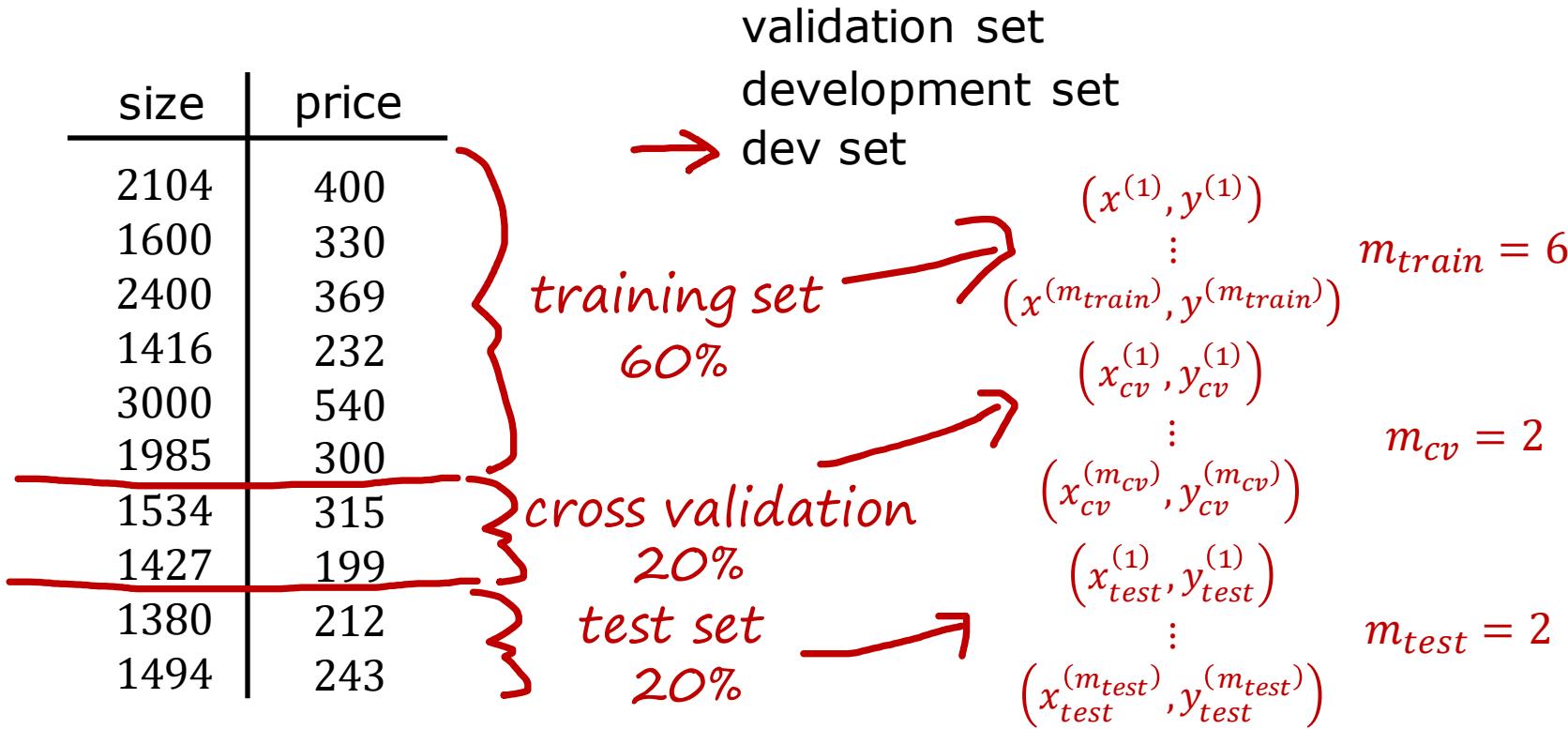
$$J_{test}(w^{<5>}, b^{<5>})$$

How well does the model perform? Report test set error  $J_{test}(w^{<5>}, b^{<5>})$ ?

The problem is  $J_{test}(w^{<5>}, b^{<5>})$  is likely to be an optimistic estimate of generalization error. Ie: An extra parameter  $d$  (degree of polynomial) was chosen using the test set.

$w, b$

# Training/cross validation/test set



# Training/cross validation/test set

Training error: 
$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[ \sum_{i=1}^{m_{train}} (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 \right]$$

Cross validation error: 
$$J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}} \left[ \sum_{i=1}^{m_{cv}} (f_{\vec{w}, b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)})^2 \right]$$
 (validation error,  
dev error)

Test error: 
$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[ \sum_{i=1}^{m_{test}} (f_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)})^2 \right]$$

# Model selection

$d=1$

$$1. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + b$$

$d=2$

$$2. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + b$$

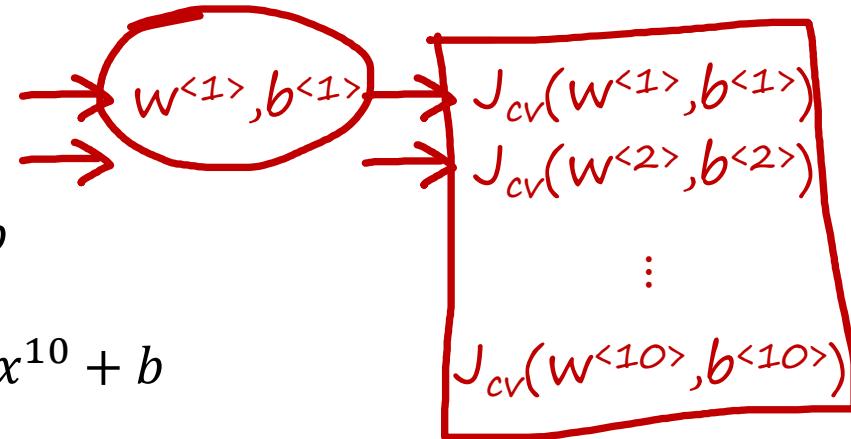
$d=3$

$$3. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + b$$

:

$d=10$

$$10. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + \dots + w_{10} x^{10} + b$$



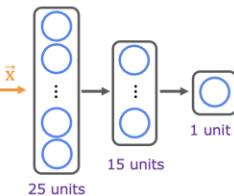
→ Pick  $w_1 x_1 + \dots + w_4 x^4 + b$

$(J_{cv}(w^{<4>}, b^{<4>}))$

Estimate generalization error using test the set:  $J_{test}(w^{<4>}, b^{<4>})$

# Model selection – choosing a neural network architecture

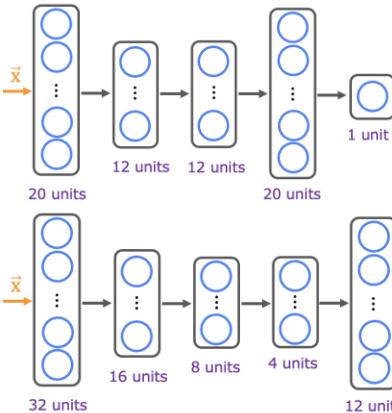
1.



$$w^{(1)}, b^{(1)}$$

$$\underline{J_{cv}(\mathbf{W}^{(1)}, \mathbf{B}^{(1)})}$$

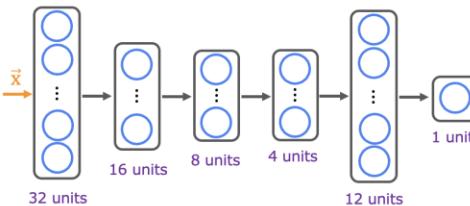
→ 2.



$$w^{(2)}, b^{(2)}$$

$$\textcircled{J_{cv}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})}$$

3.



$$w^{(3)}, b^{(3)}$$

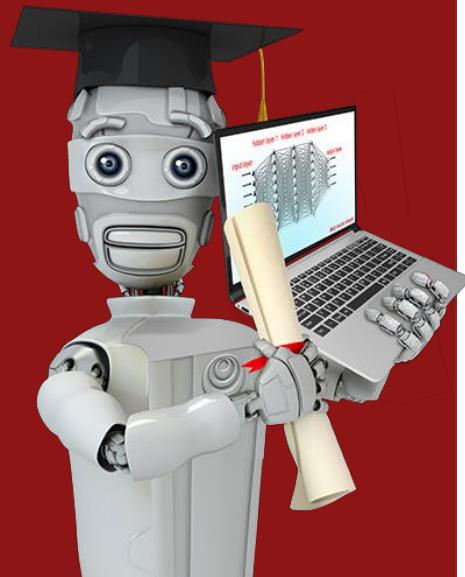
$$J_{cv}(\mathbf{W}^{(3)}, \mathbf{B}^{(3)})$$

Train, CV

Pick  $\mathbf{W}^{(2)}, \mathbf{B}^{(2)}$

Estimate generalization error using the test set:

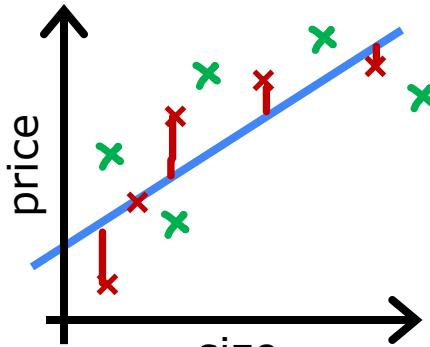
$$\boxed{J_{test}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})}$$



## Bias and variance

Diagnosing bias and variance

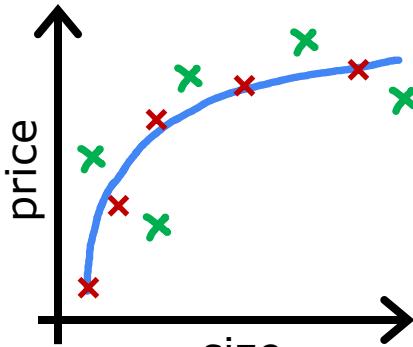
# Bias/variance



$$f_{\vec{w},b}(x) = w_1x + b$$

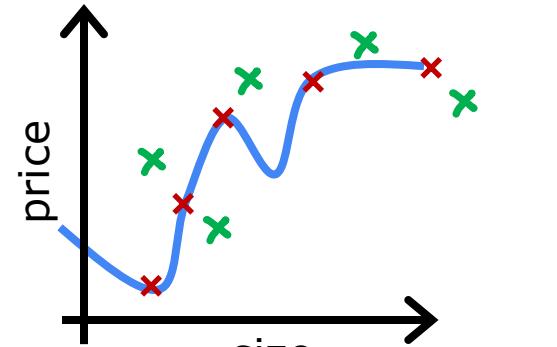
→ High bias  
(underfit)

$d = 1$   $J_{train}$  is high  
 $J_{cv}$  is high



$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$$

"Just right"



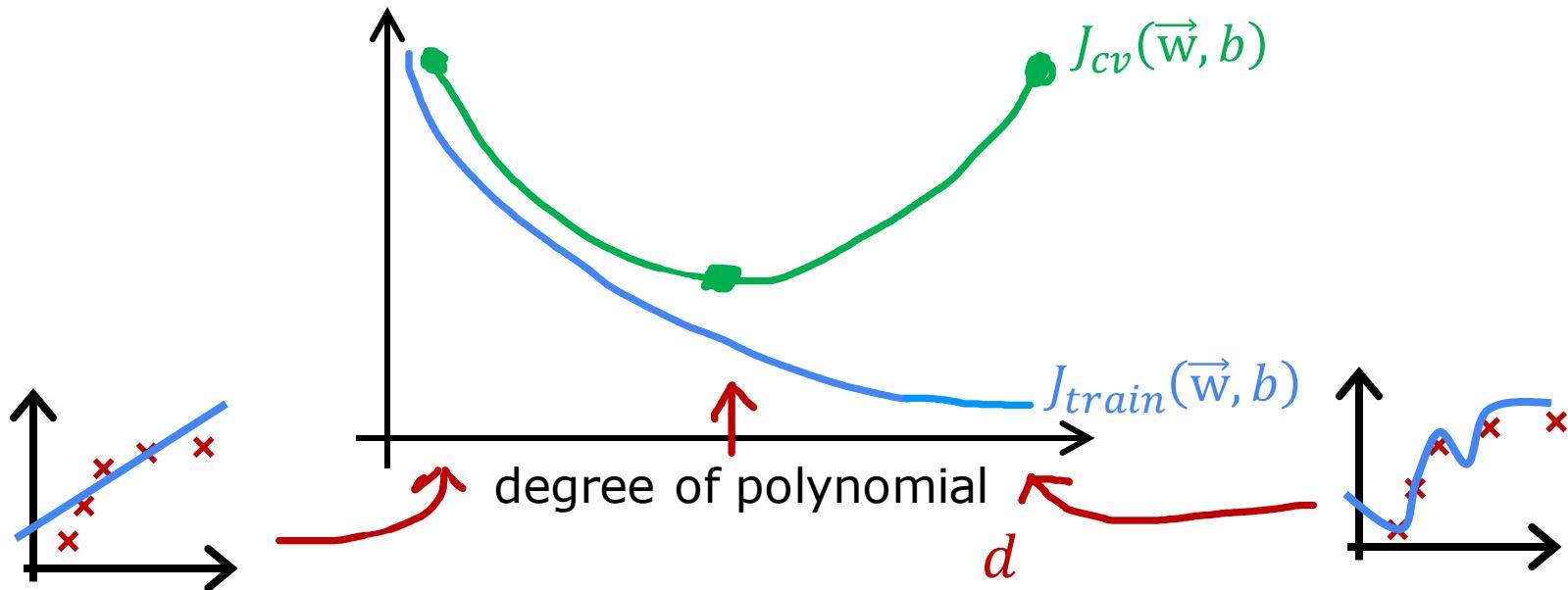
$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

High variance  
(overfit)

$d = 2$   $J_{train}$  is low  
 $J_{cv}$  is low

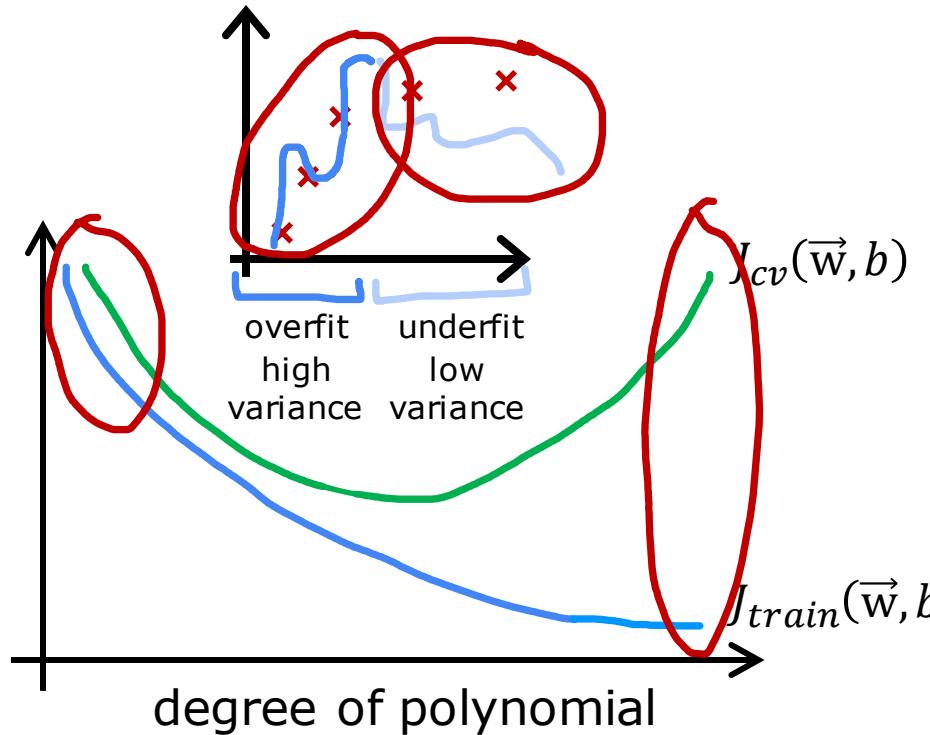
$d = 4$   $J_{train}$  is low  
 $J_{cv}$  is high

# Understanding bias and variance



# Diagnosing bias and variance

How do you tell if your algorithm has a bias or variance problem?



High bias (underfit)

$J_{train}$  will be high

( $J_{train} \approx J_{cv}$ )

High variance (overfit)

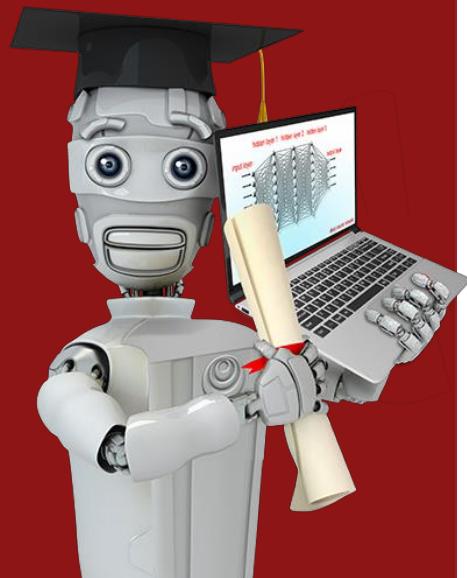
$J_{cv} \gg J_{train}$

( $J_{train}$  may be low)

High bias and high variance

$J_{train}$  will be high

and  $J_{cv} \gg J_{train}$



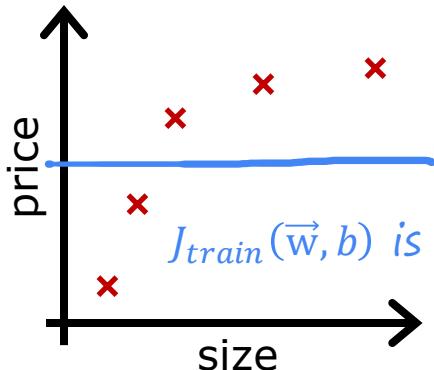
## Bias and variance

Regularization and  
bias/variance

# Linear regression with regularization

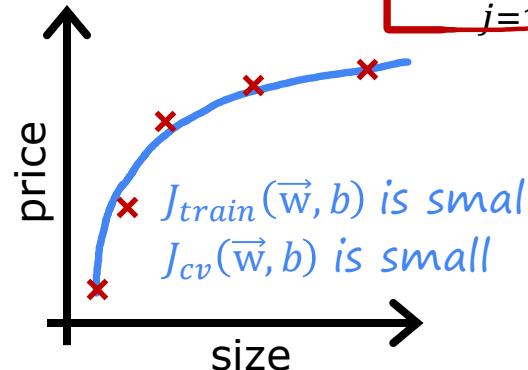
Model:  $f_{\vec{w}, b}(x) = \underbrace{w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b}_m$

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n w_j^2$$



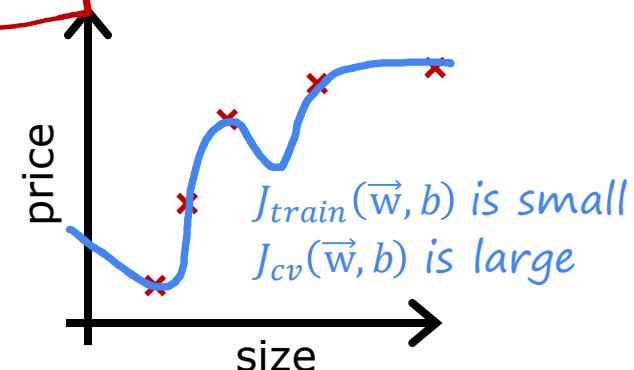
Large  $\lambda$   
High bias (underfit)

$$\lambda = 10,000 \quad w_1 \approx 0, w_2 \approx 0$$
$$f_{\vec{w}, b}(\vec{x}) \approx b$$



Intermediate  $\lambda$

$$\lambda$$

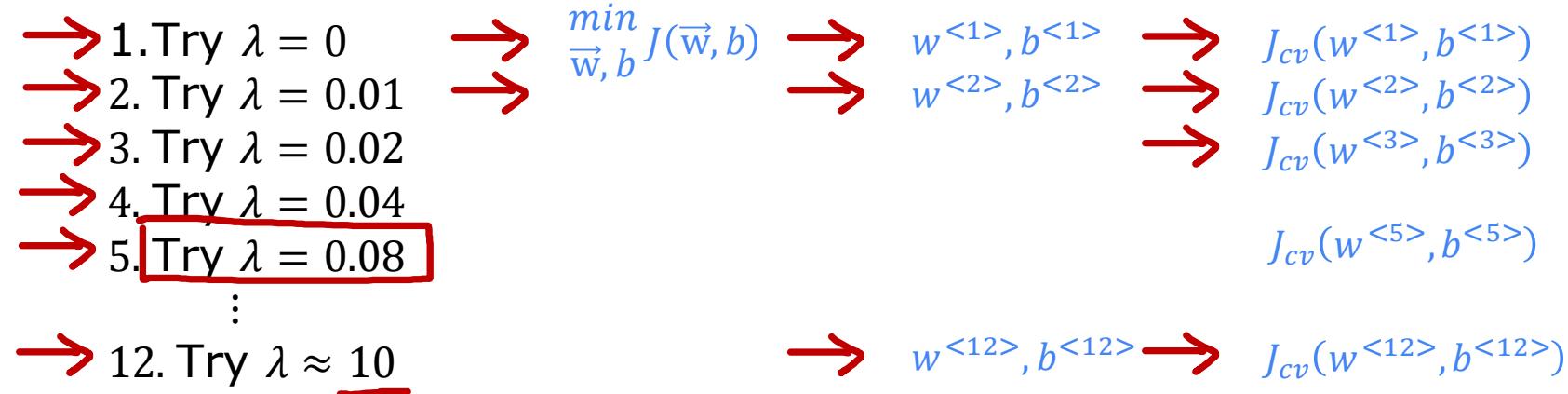


Small  $\lambda$   
High variance (overfit)

$$\lambda = 0$$

# Choosing the regularization parameter $\lambda$

Model:  $f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

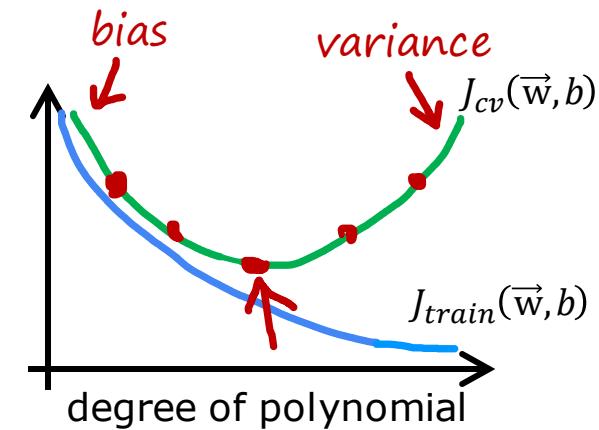
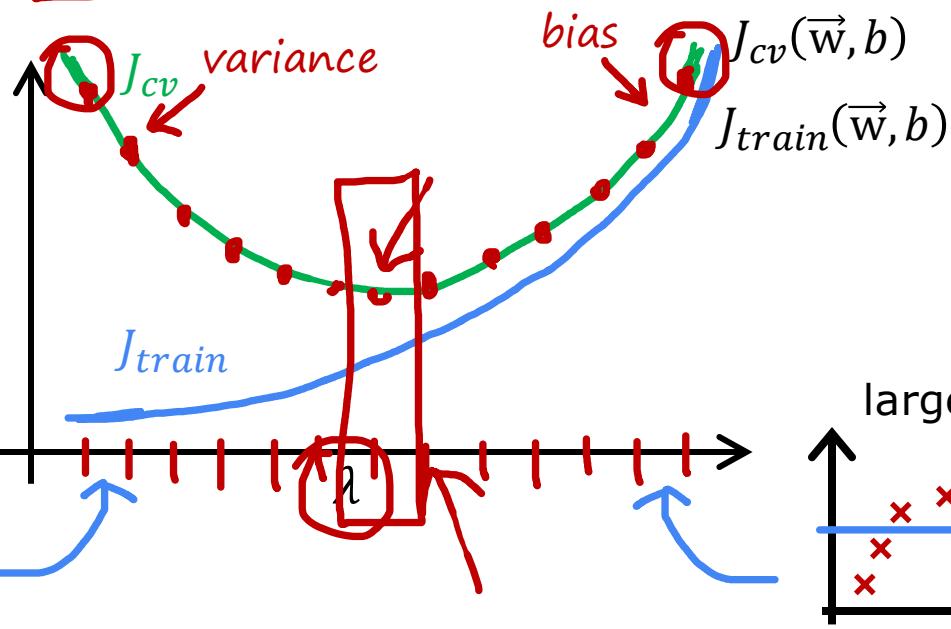


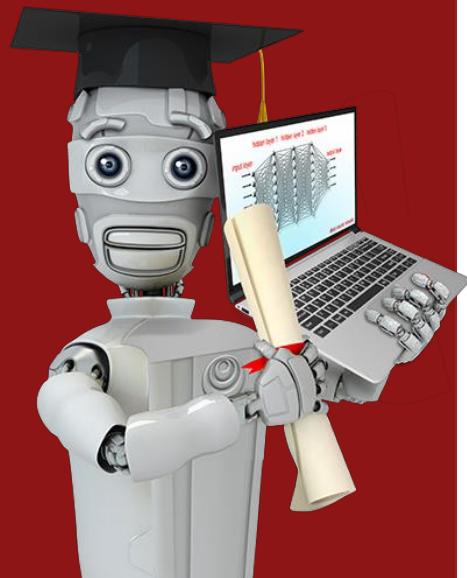
Pick  $w^{<5>}, b^{<5>}$

Report test error:  $J_{test}(w^{<5>}, b^{<5>})$

# Bias and variance as a function of regularization parameter $\lambda$

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$





## Bias and variance

Establishing a baseline level of performance

# Speech recognition example



Human level performance

: 10.6% 0.2%

: 10.8% 4.0%

: 14.8%

Training error  $J_{train}$

Cross validation error  $J_{cv}$

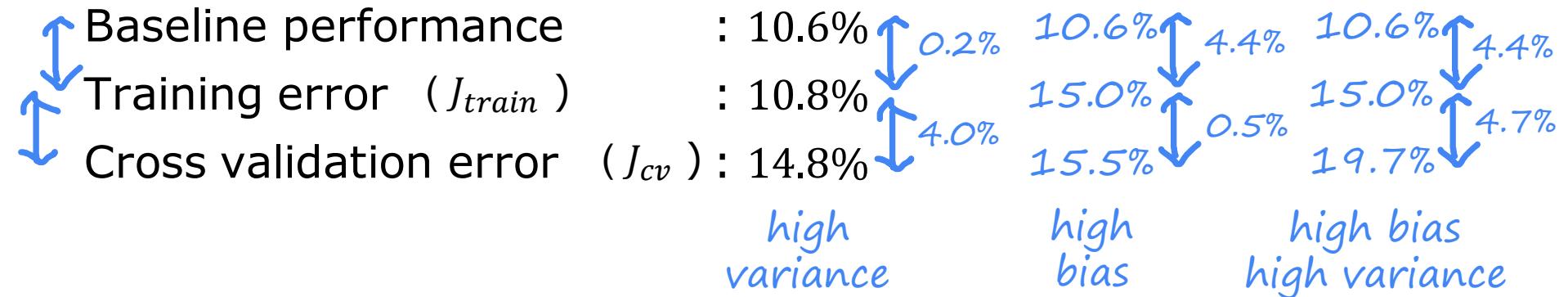


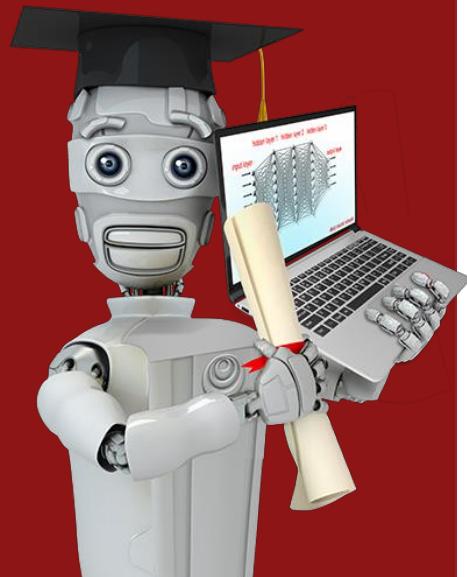
# Establishing a baseline level of performance

What is the level of error you can reasonably hope to get to?

- Human level performance
- Competing algorithms performance
- Guess based on experience

# Bias/variance examples

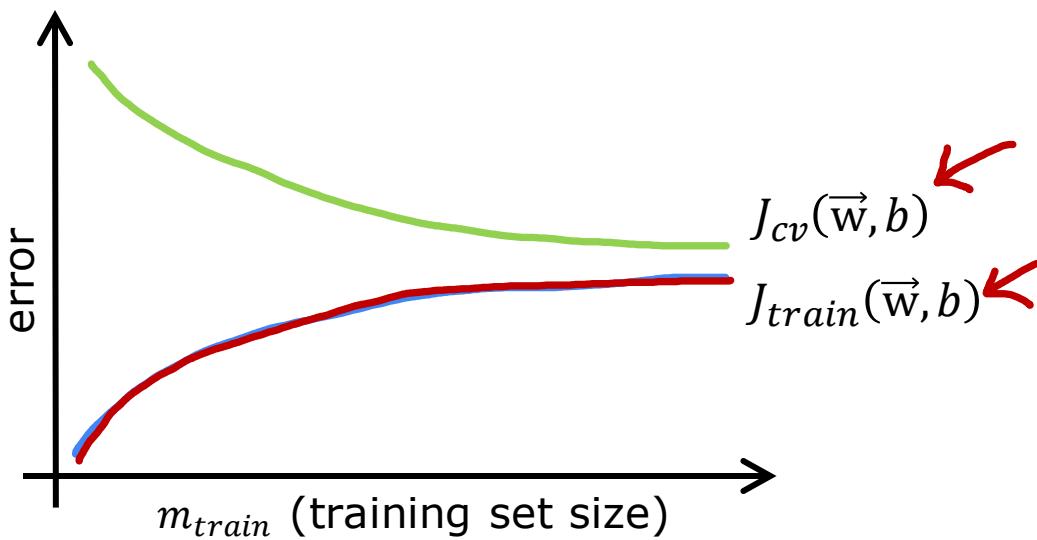




## Bias and variance

# Learning curves

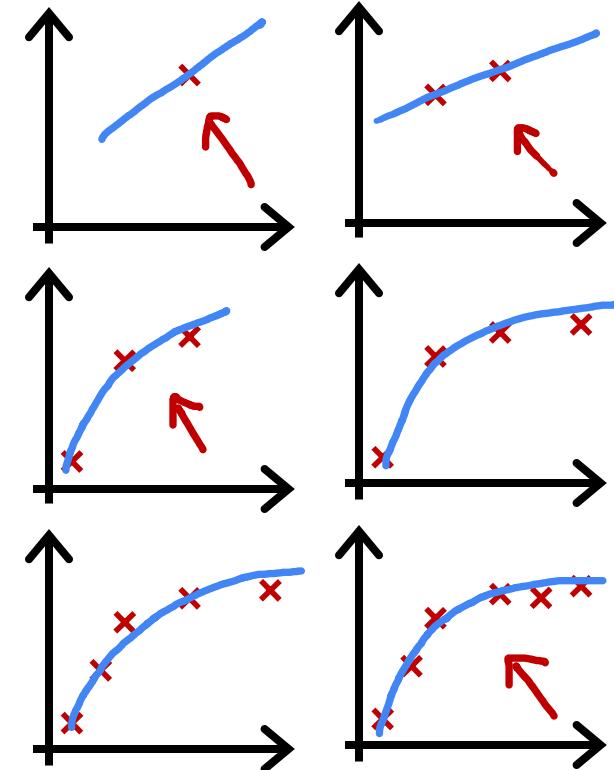
# Learning curves



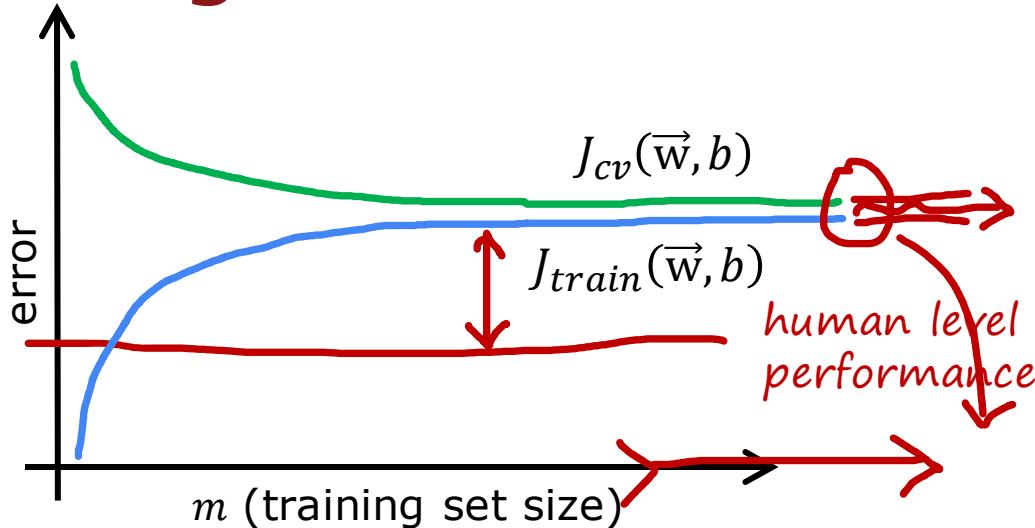
$J_{train}$  = training error

$J_{cv}$  = cross validation error

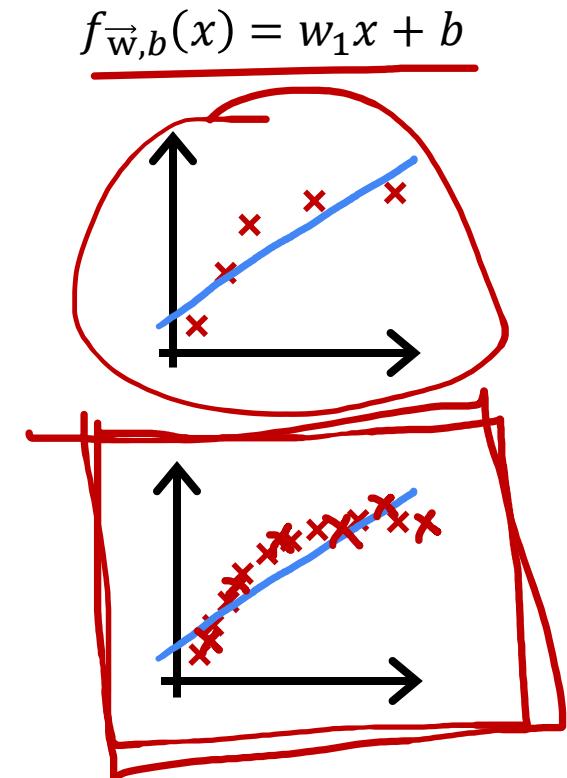
$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$



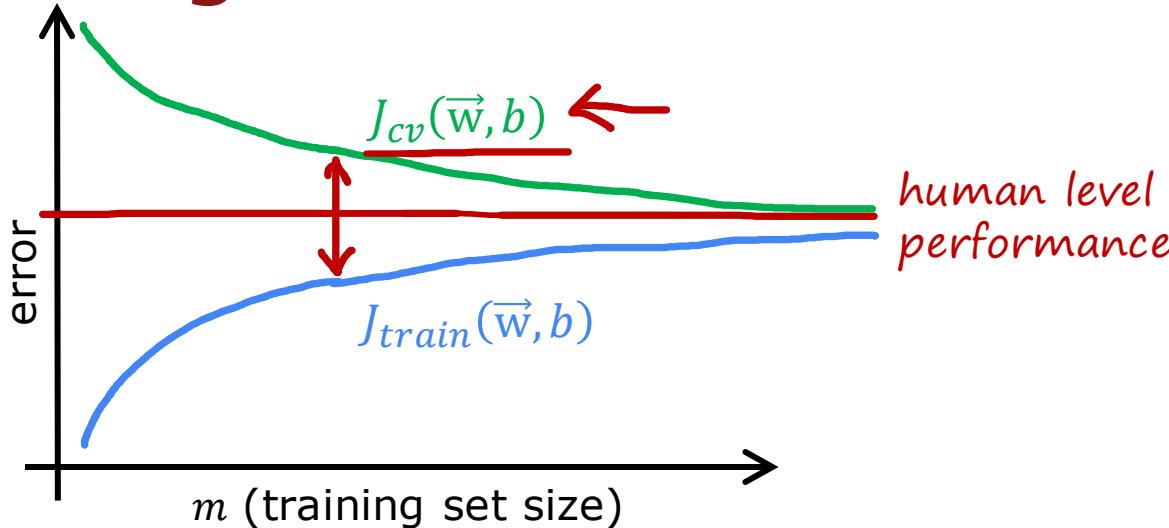
# High bias



if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

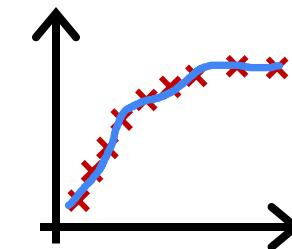
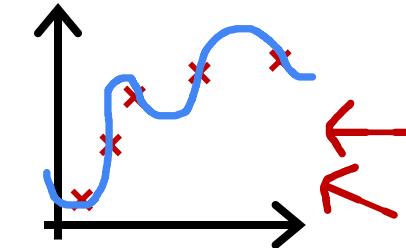


# High variance



if a learning algorithm suffers  
from high variance, getting  
more training data is likely to  
help.

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b \quad (\text{with small } \lambda)$$





## Bias and variance

Deciding what to try next  
revisited

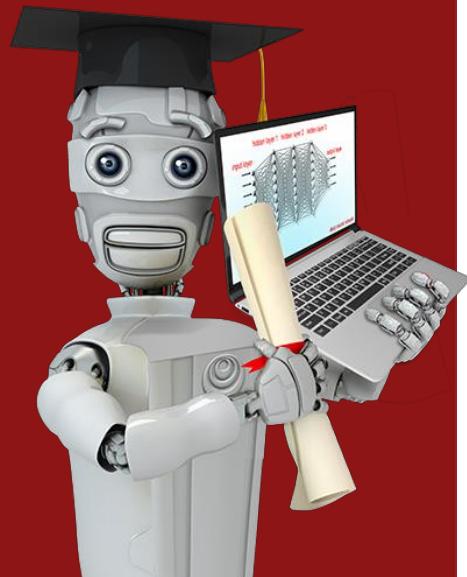
# Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(\vec{w}, b) = \underbrace{\frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2}_{\text{Large errors}} + \underbrace{\frac{\lambda}{2m} \sum_{j=1}^n w_j^2}_{\text{Large } \lambda}$$

But it makes unacceptably large errors in predictions. What do you try next?

- Get more training examples fixes high variance
- Try smaller sets of features  $x, x^2, \cancel{x}, \cancel{x^3}, \cancel{x^4}, \dots$  fixes high variance
- Try getting additional features ← fixes high bias
- Try adding polynomial features  $(x_1^2, x_2^2, x_1 x_2, \text{etc})$  fixes high bias
- Try decreasing  $\lambda$  ← fixes high bias
- Try increasing  $\lambda$  ← fixes high variance



## Bias and variance

**Bias/variance and  
neural networks**

# The bias variance tradeoff

$$f_{\vec{w}, b}(x) = w_1 x + b$$

Simple model

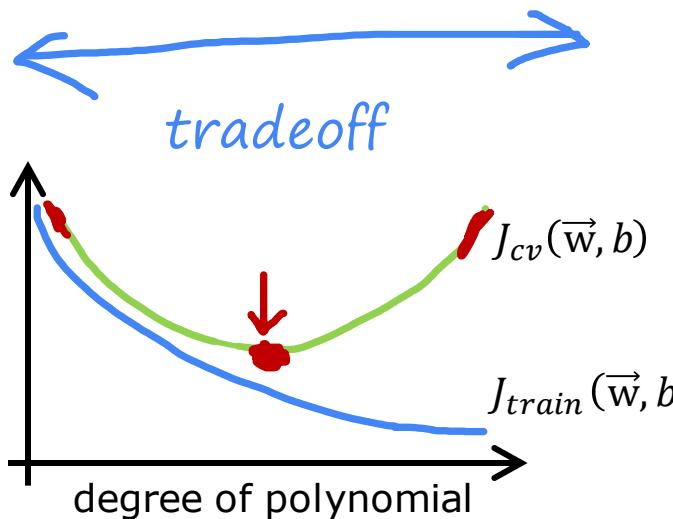
High bias

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

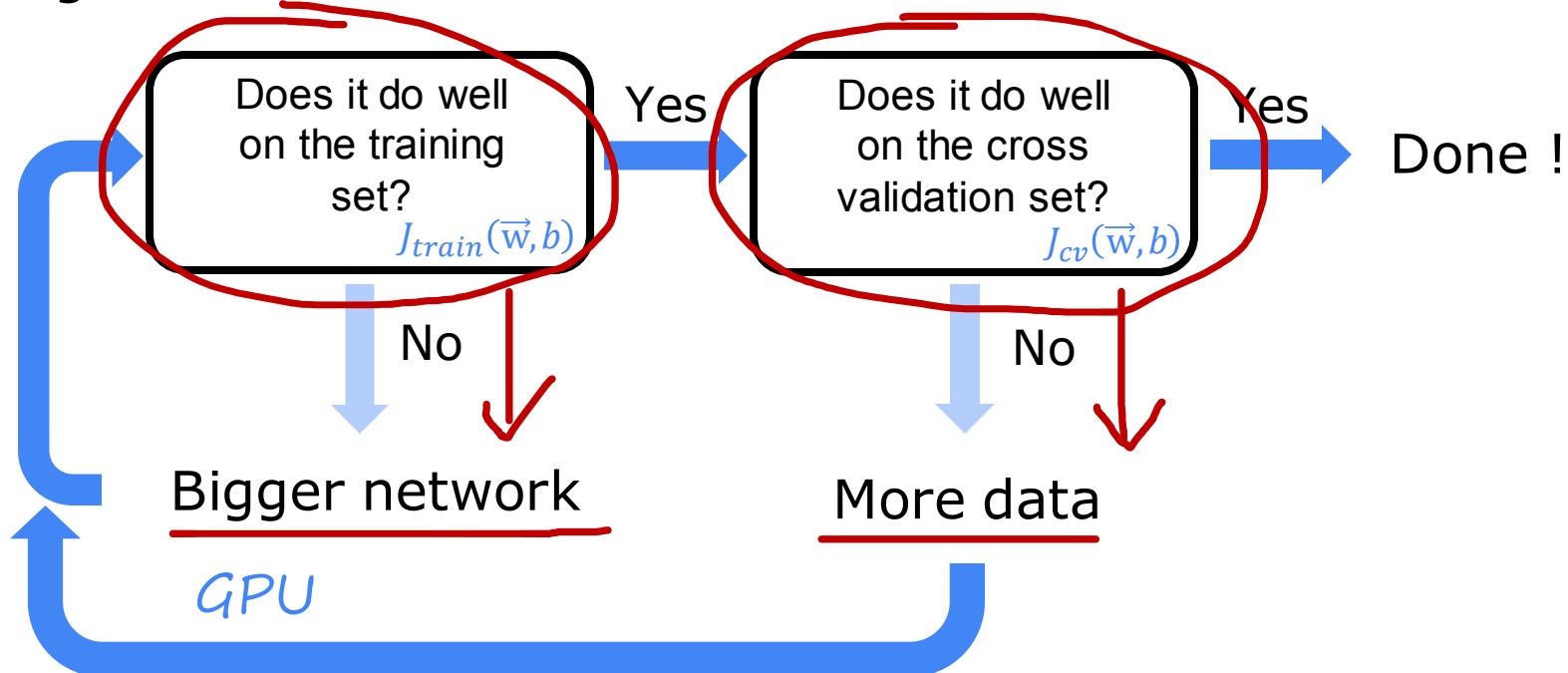
Complex model

High variance

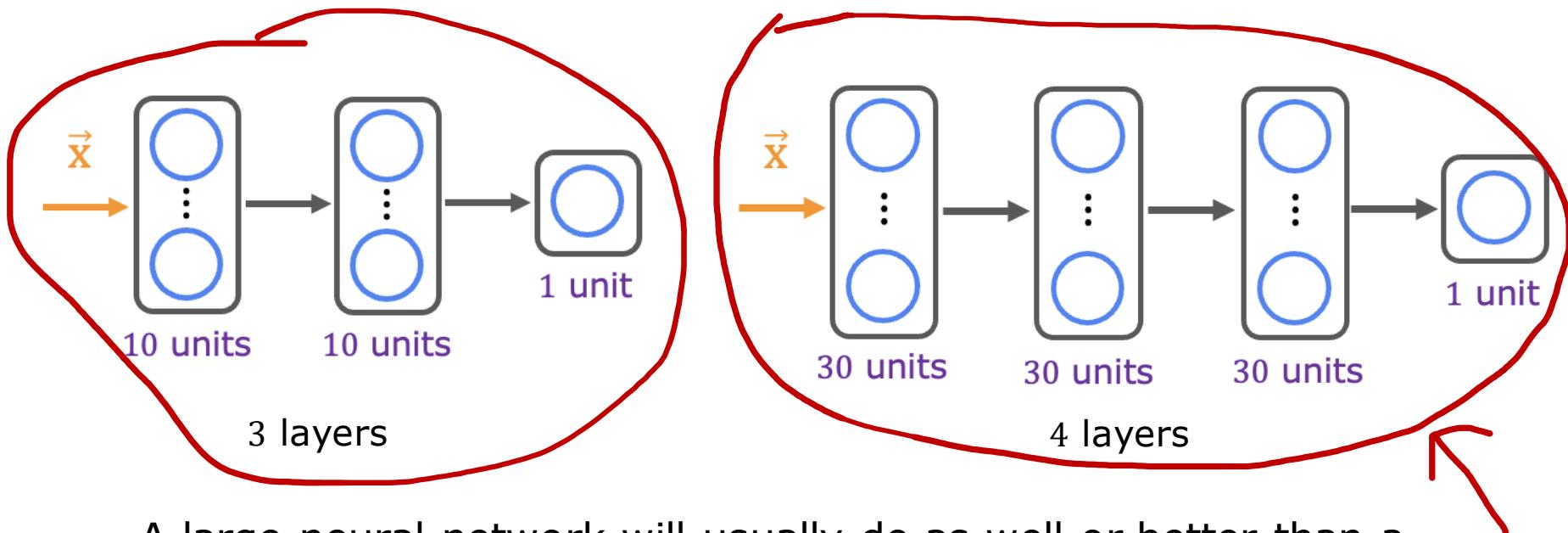


# Neural networks and bias variance

Large neural networks are low bias machines



# Neural networks and regularization



A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

# Neural network regularization

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^m L(f(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{\text{all weights } \mathbf{W}} (w^2)$$

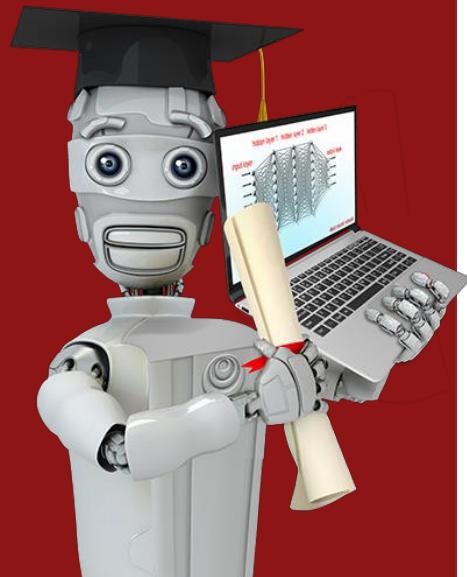
## Unregularized MNIST model

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

## Regularized MNIST model

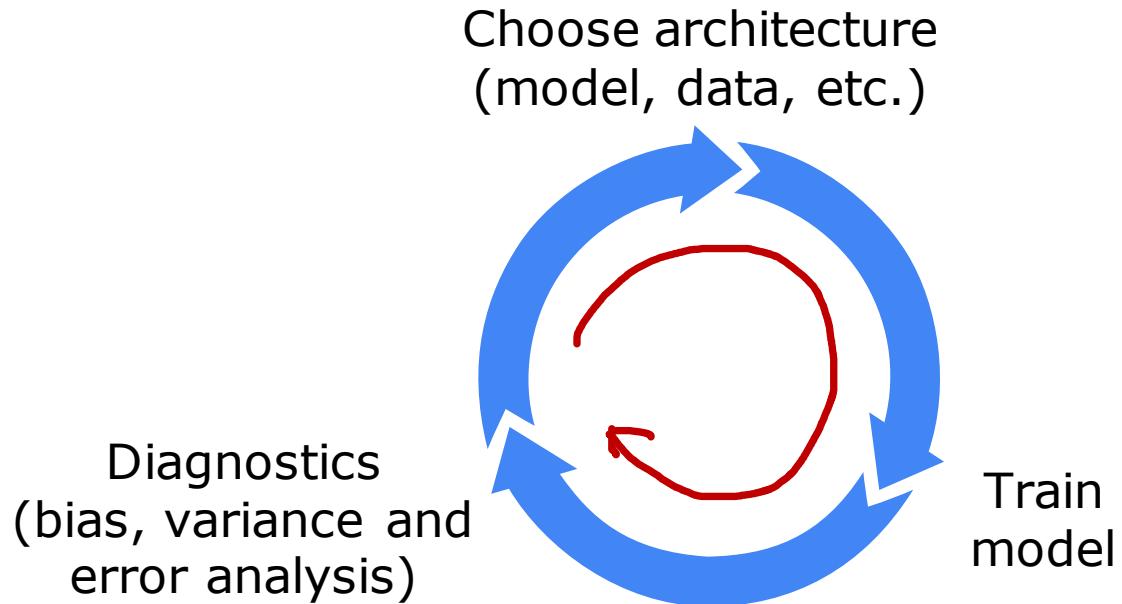
```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

# Machine learning development process



**Iterative loop of  
ML development**

# Iterative loop of ML development



# Spam classification example

From: cheapsales@buystufffromme.com  
To: Andrew Ng  
Subject: Buy now!

Deal of the week! Buy now!  
Rolex w4tchs - \$100  
Med1cine (any kind) - £50  
Also low cost M0rgages  
available.

From: Alfred Ng  
To: Andrew Ng  
Subject: Christmas dates?

Hey Andrew,  
Was talking to Mom about plans  
for Xmas. When do you get off  
work. Meet Dec 22?  
Alf

# Building a spam classifier

Supervised learning:  $\underline{\vec{x}} = \text{features of email}$   
 $\underline{y} = \text{spam (1) or not spam (0)}$

Features: list the top 10,000 words to compute  $x_1, x_2, \dots, x_{10,000}$

$$\vec{x} = \begin{bmatrix} 0 & a \\ 1 & andrew \\ 2 & buy \\ 1 & deal \\ 0 & discount \\ \vdots & \vdots \end{bmatrix}$$

From: `cheapsales@buystufffromme.com`  
To: Andrew Ng  
Subject: Buy now!

Deal of the week! Buy now!  
Rolex w4tchs - \$100  
Medicine (any kind) - £50  
Also low cost M0rgages available.

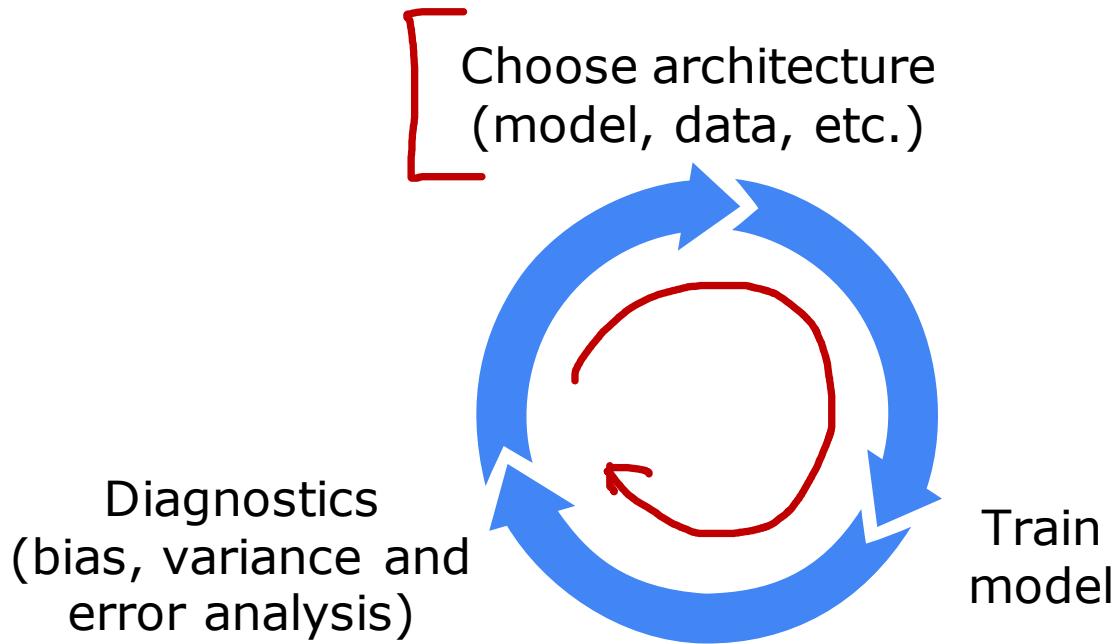
# Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., “Honeypot” project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body.  
E.g., should “discounting” and “discount” be treated as the same word.
- Design algorithms to detect misspellings.  
E.g., w4tches, med1cine, m0rtgage.

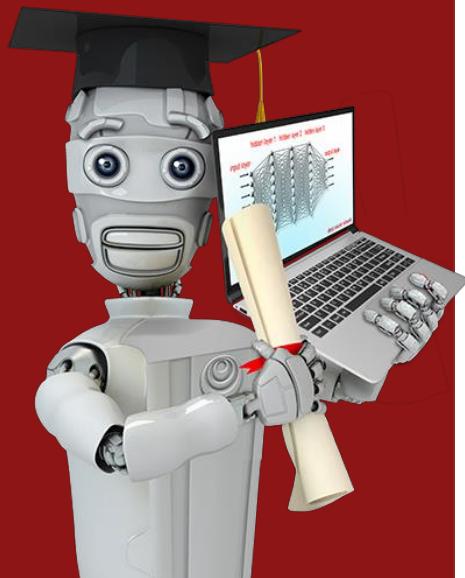


# Iterative loop of ML development



# Machine learning development process

## Error analysis



# Error analysis

$m_{cv} = \frac{500}{5000}$  examples in cross validation set.

Algorithm misclassifies 100 of them.

Manually examine 100 examples and categorize them based on common traits.

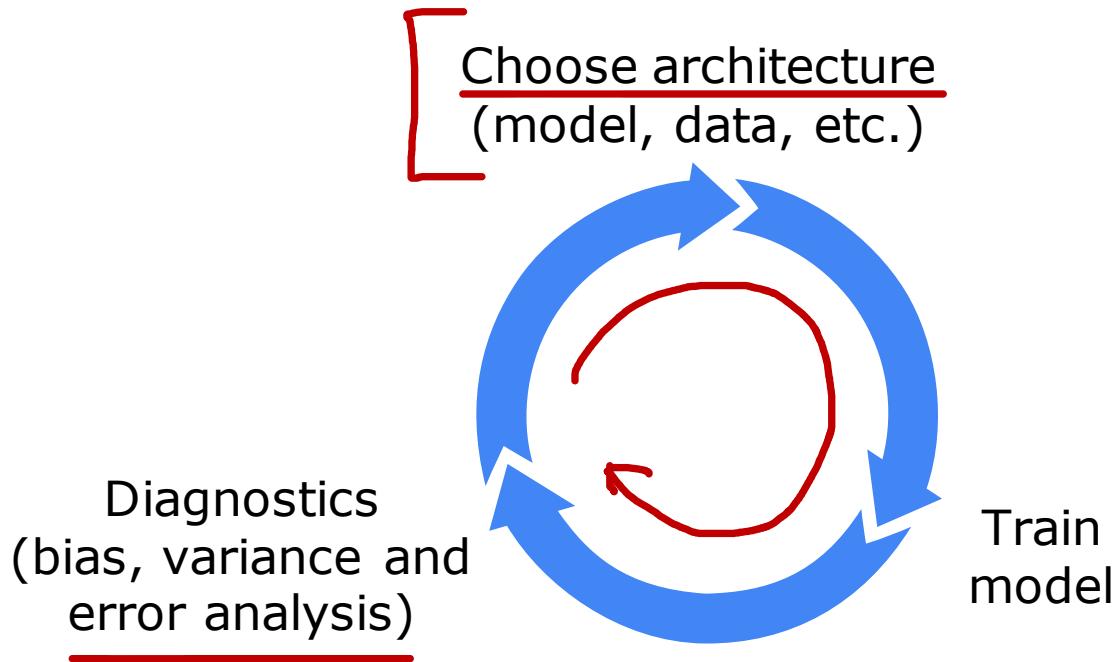
- Pharma: 21 → more data features
- Deliberate misspellings (w4tches, med1cine): 3
- Unusual email routing: 7
- Steal passwords (phishing): 18 → more data features
- Spam message in embedded image: 5

# Building a spam classifier

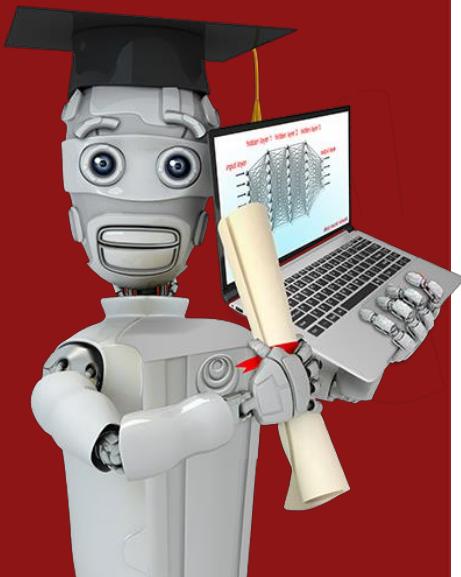
How to try to reduce your spam classifier's error?

- Collect more data. E.g., “Honeypot” project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body.  
E.g., should “discounting” and “discount” be treated as the same word.
- Design algorithms to detect misspellings.  
E.g., w4tches, med1cine, m0rtgage.

# Iterative loop of ML development



# Machine learning development process



## Adding data

# Adding data

- Add more data of everything. E.g., “Honeypot” project.
- Add more data of the types where error analysis has indicated it might help.

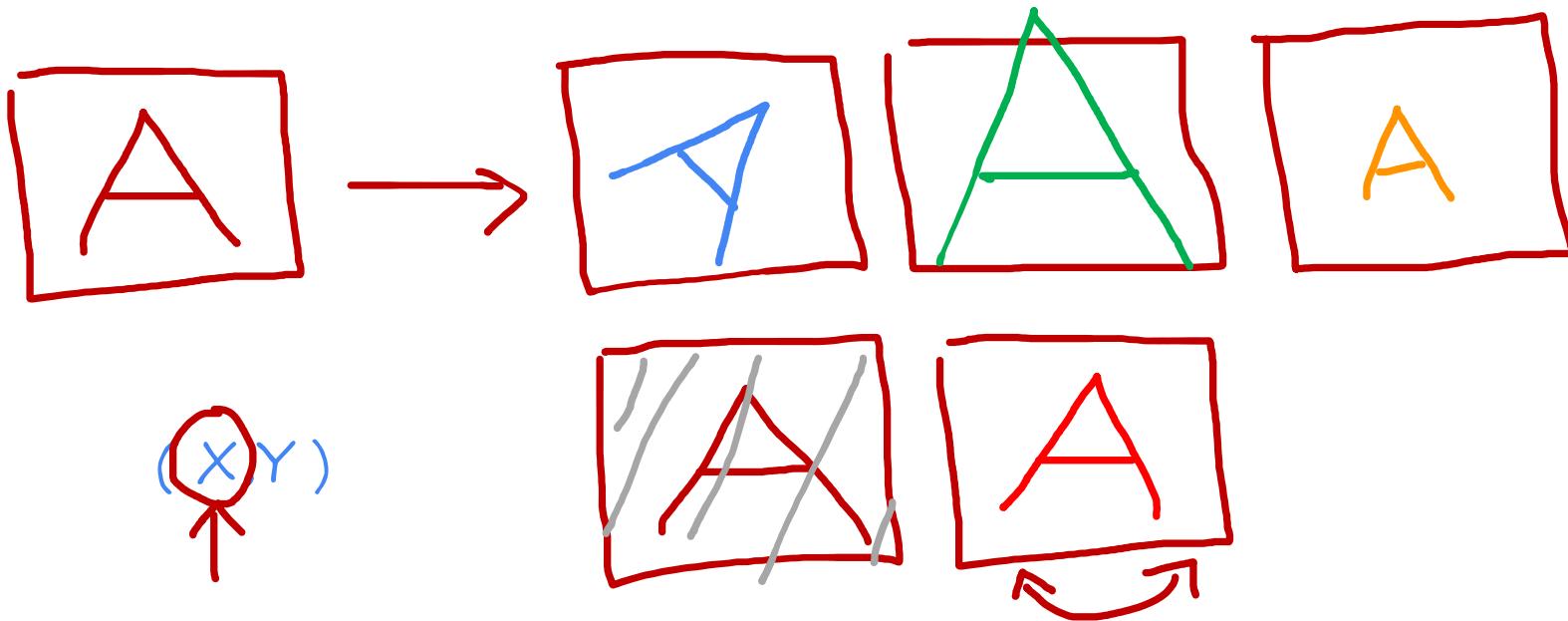
*Pharma spam*

E.g., Go to unlabeled data and find more examples of Pharma related spam.

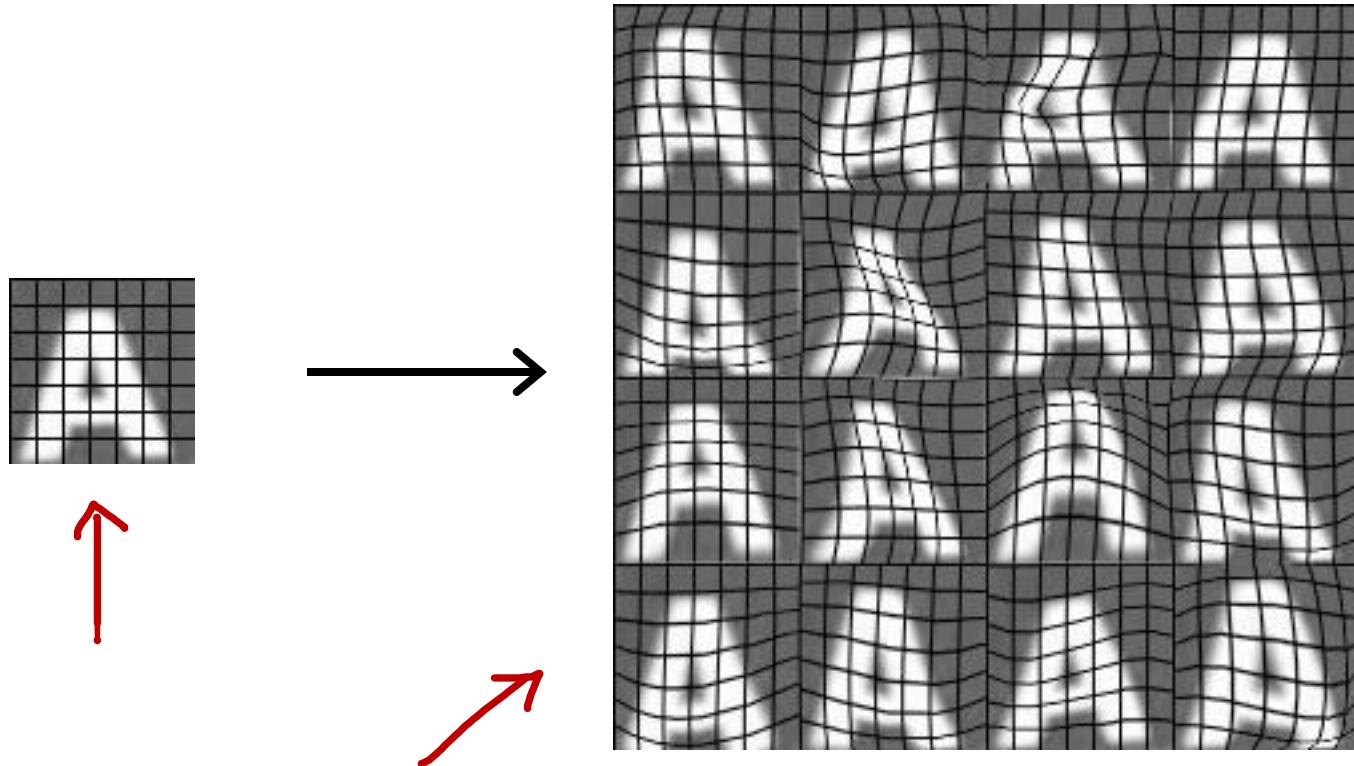
( X,Y )

# Data augmentation

Augmentation: modifying an existing training example to create a new training example.

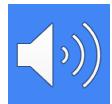


# Data augmentation by introducing distortions



# Data augmentation for speech

Speech recognition example



Original audio (voice search: "What is today's weather?")



+ Noisy background: Crowd



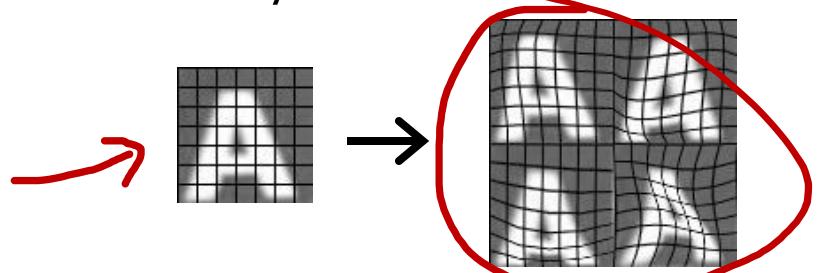
+ Noisy background: Car



+ Audio on bad cellphone connection

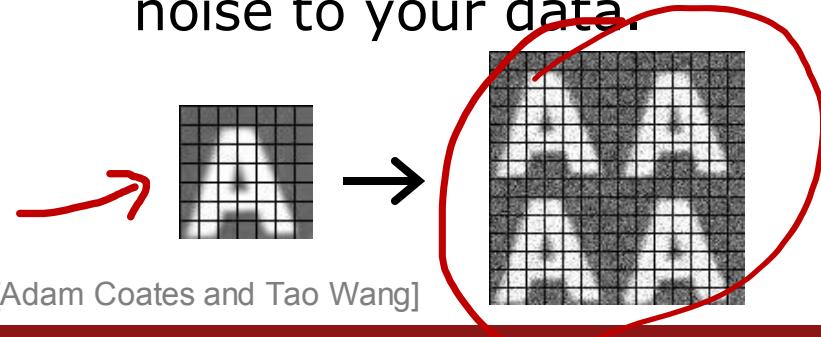
# Data augmentation by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



Audio:  
Background noise,  
bad cellphone connection

Usually does not help to add purely random/meaningless noise to your data.



$x_i$ =intensity (brightness) of pixel  $i$   
 $x_i \leftarrow x_i + \text{random noise}$

[Adam Coates and Tao Wang]

# Data synthesis

Synthesis: using artificial data inputs to create a new training example.

# Artificial data synthesis for photo OCR



[<http://www.publicdomainpictures.net/view-image.php?image=5745&picture=times-square>]

# Artificial data synthesis for photo OCR



Real data

[Adam Coates and Tao Wang]

Abcdefg

Abcdefg

A  
bc  
def  
g

A  
B  
c  
d  
e  
f  
g

A  
b  
c  
d  
e  
f  
g

# Artificial data synthesis for photo OCR



Real data



Synthetic data

[Adam Coates and Tao Wang]

# Engineering the data used by your system

Conventional  
model-centric  
approach:

$$\text{AI} = \text{Code} + \text{Data}$$

(algorithm/model)



Work on this

Data-centric  
approach:

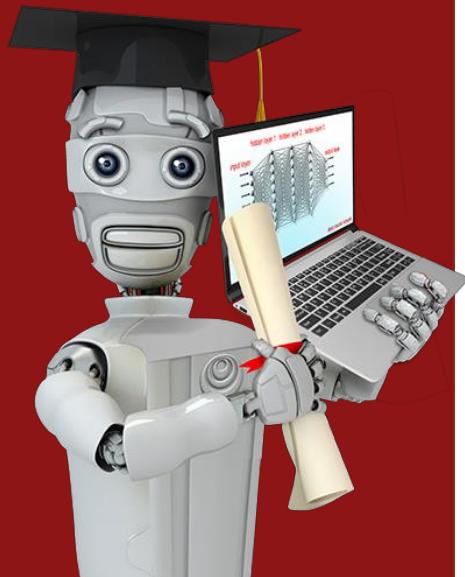
$$\text{AI} = \text{Code} + \text{Data}$$

(algorithm/model)



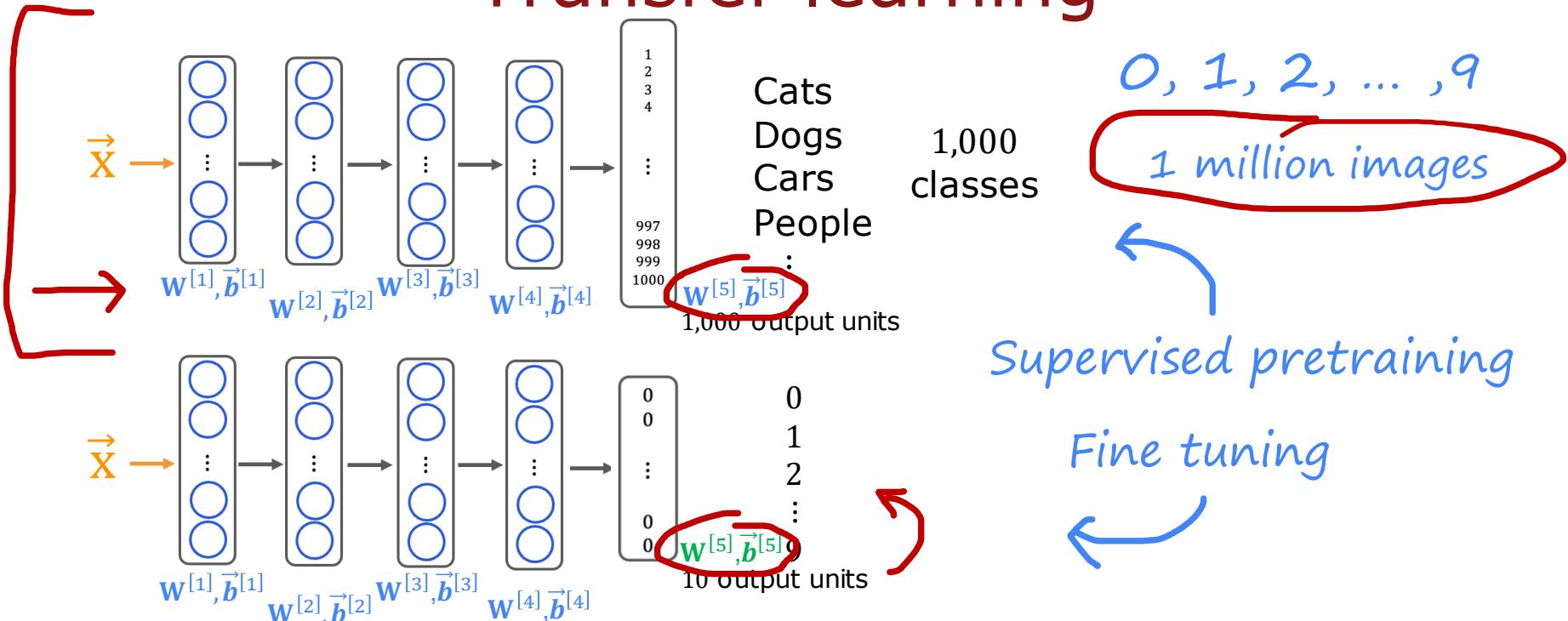
Work on this

# Machine learning development process



**Transfer learning: using data  
from a different task**

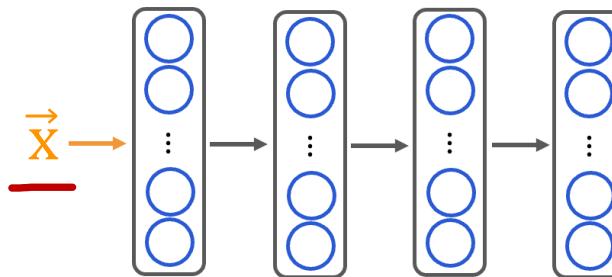
# Transfer learning



Option 1: only train **output layers** parameters.

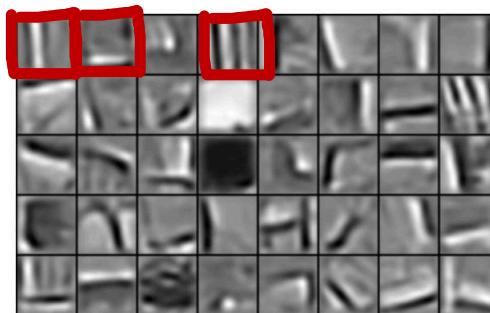
Option 2: **train all parameters**.

# Why does transfer learning work?

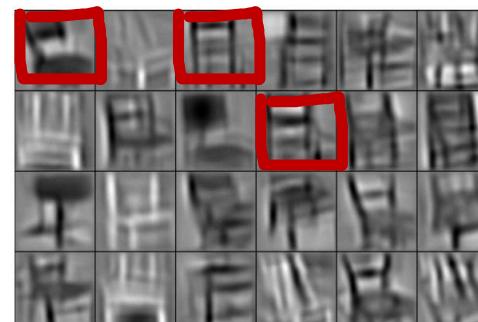


detects  
edges      detects  
corners      detects  
curves/basic shapes

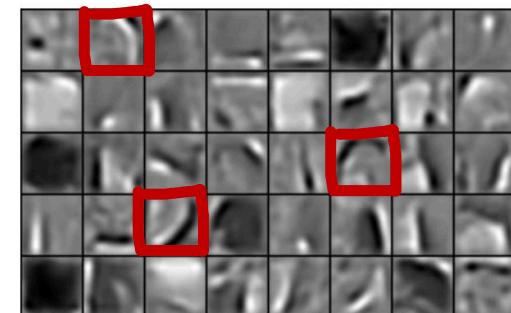
*use the same input type*



Edges



Corners

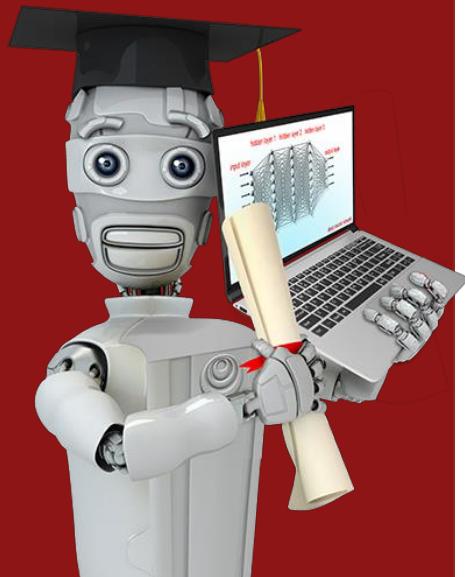


Curves / basic shapes

# Transfer learning summary

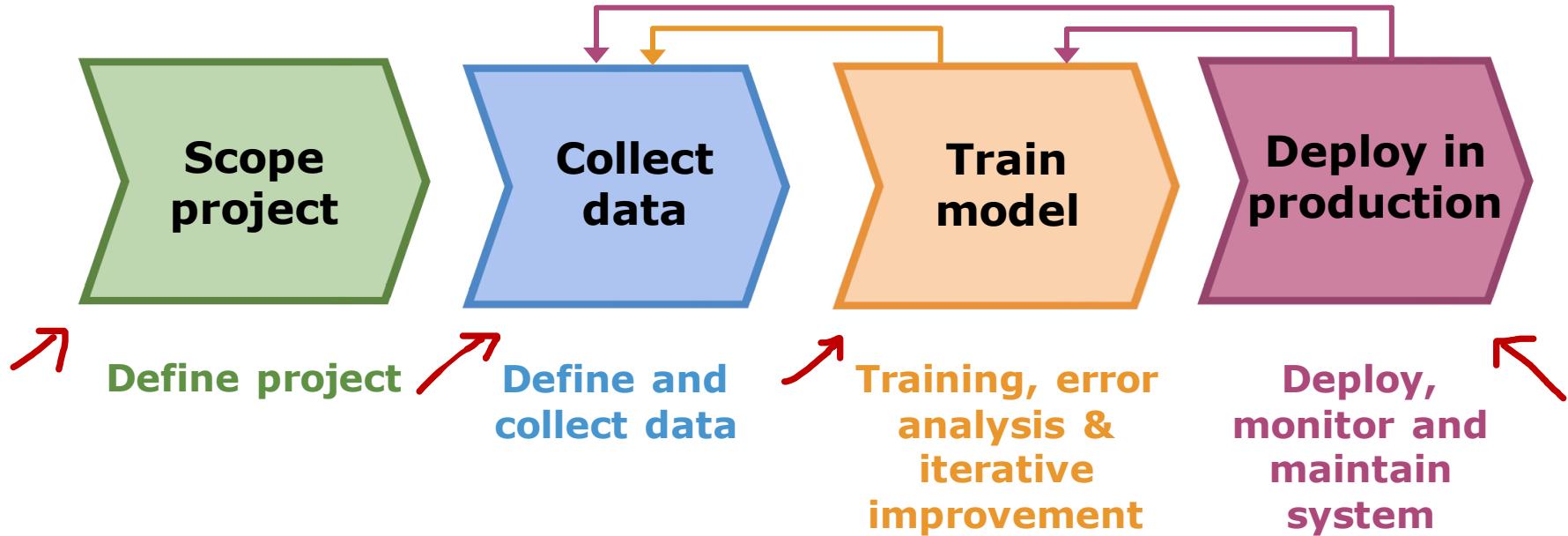
- 1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). *1M*
- 2. Further train (fine tune) the network on your own data. *1000*  
*50*

# Machine learning development process

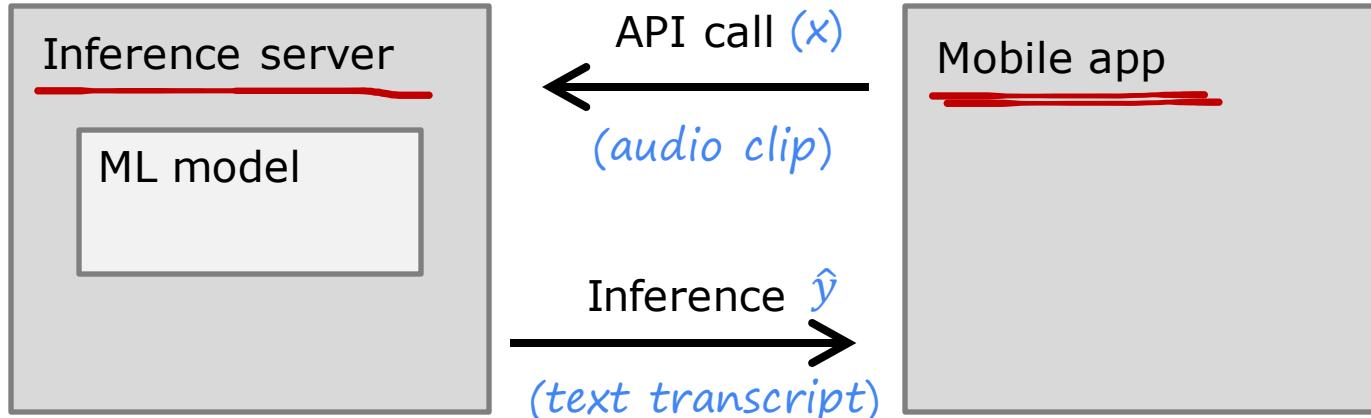


Full cycle of a  
machine learning project

# Full cycle of a machine learning project



# Deployment

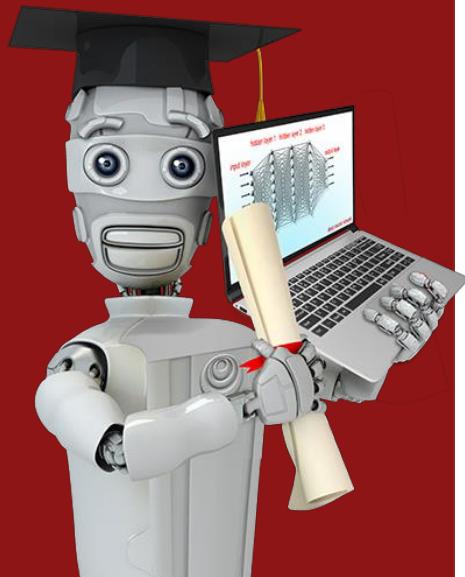


→ Software engineering may be needed for:

- Ensure reliable and efficient predictions
- Scaling
- Logging
- System monitoring
- Model updates

MLOps  
machine learning  
operations

# Machine learning development process



## Fairness, bias, and ethics

# Bias

Hiring tool that discriminates against women.

Facial recognition system matching dark skinned individuals to criminal mugshots.

Biased bank loan approvals.

Toxic effect of reinforcing negative stereotypes.

# Adverse use cases

## Deepfakes

Spreading toxic/incendiary speech through optimizing for engagement.

Generating fake content for commercial or political purposes.

Using ML to build harmful products, commit fraud etc.

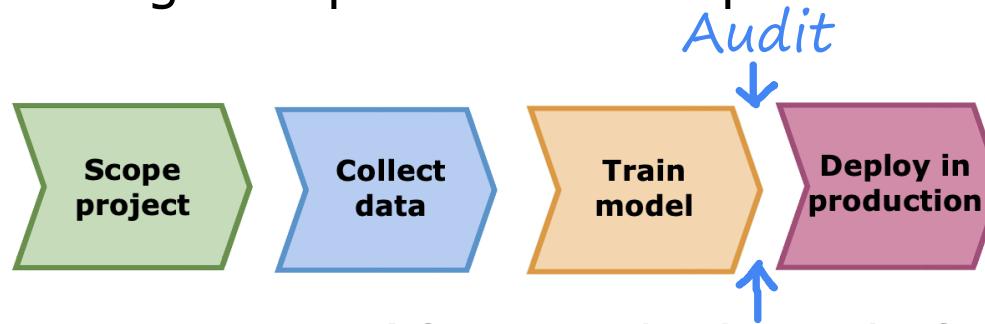
Spam vs anti-spam : fraud vs anti-fraud.

# Guidelines

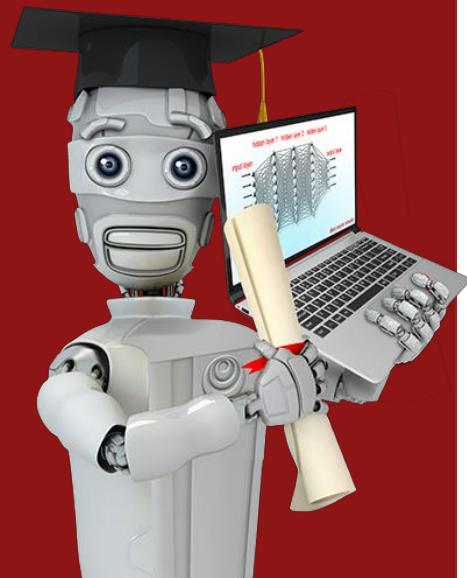
Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.

Carry out literature search on standards/guidelines for your industry.

Audit systems against possible harm prior to deployment.



Develop mitigation plan (if applicable), and after deployment, monitor for possible harm.



## Skewed datasets (optional)

Error metrics for  
skewed datasets

# Rare disease classification example

Train classifier  $f_{\vec{w}, b}(\vec{x})$

( $y = 1$  if disease present,  
 $y = 0$  otherwise)

Find that you've got 1% error on test set  
(99% correct diagnoses)

Only 0.5% of patients have the disease

print ("y=0")

99.5% accuracy

0.5% error

1%

1.2%

# Precision/recall

$y = 1$  in presence of rare class we want to detect.

Actual Class		
		1      0
Predict -ed Class	1	True positive 15
	0	False positive 5

		False negative 10
Predict -ed Class	1	True negative 70
	0	False negative 10

↓      25

↓      75

`print("y=0")`

## Precision:

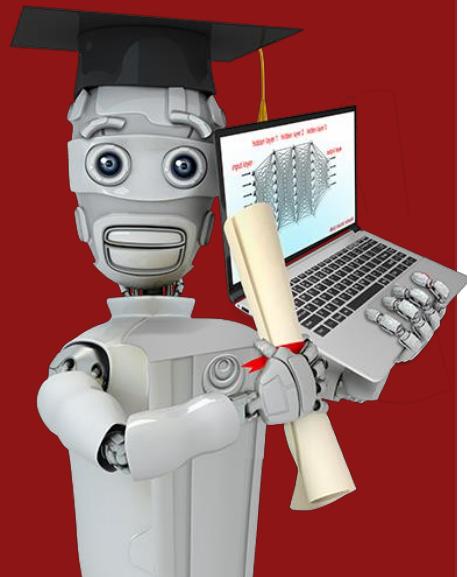
(of all patients where we predicted  $y = 1$ , what fraction actually have the rare disease?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15+5} = 0.75$$

## Recall: ←

(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$



## Skewed datasets (optional)

Trading off precision  
and recall

# Trading off precision and recall

Logistic regression:  $0 < f_{\vec{w}, b}(\vec{x}) < 1$

- Predict 1 if  $f_{\vec{w}, b}(\vec{x}) \geq \cancel{0.5}$  ~~0.7~~ ~~0.9~~ ~~0.3~~
- Predict 0 if  $f_{\vec{w}, b}(\vec{x}) < \cancel{0.5}$  ~~0.7~~ ~~0.9~~ ~~0.3~~

$$\text{precision} = \frac{\text{true positives}}{\text{total predicted positive}}$$
$$\text{recall} = \frac{\text{true positives}}{\text{total actual positive}}$$

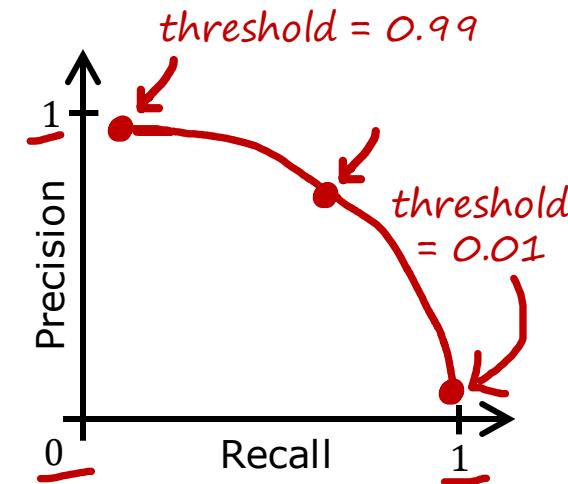
Suppose we want to predict  $y = 1$  (rare disease) only if very confident.

→ higher precision, lower recall.

Suppose we want to avoid missing too many cases of rare disease (when in doubt predict  $y = 1$ )

→ lower precision, higher recall.

More generally predict 1 if:  $f_{\vec{w}, b}(\vec{x}) \geq \underline{\text{threshold}}$ .



# F1 score

How to compare precision/recall numbers?

	Precision (P)	Recall (R)	Average	$F_1$ score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.501	0.0392

`print("y=1")`

~~Average =  $\frac{P+R}{2}$~~

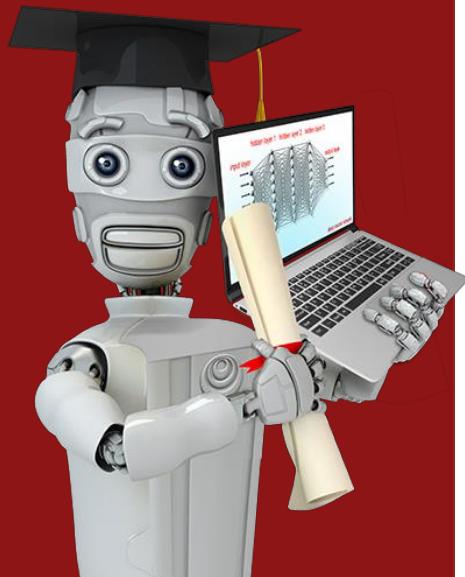
$F1 \text{ score} = \frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right) = 2 \frac{PR}{P+R}$

# 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>



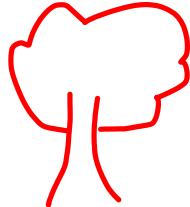
## Decision Trees

# Decision Tree Model

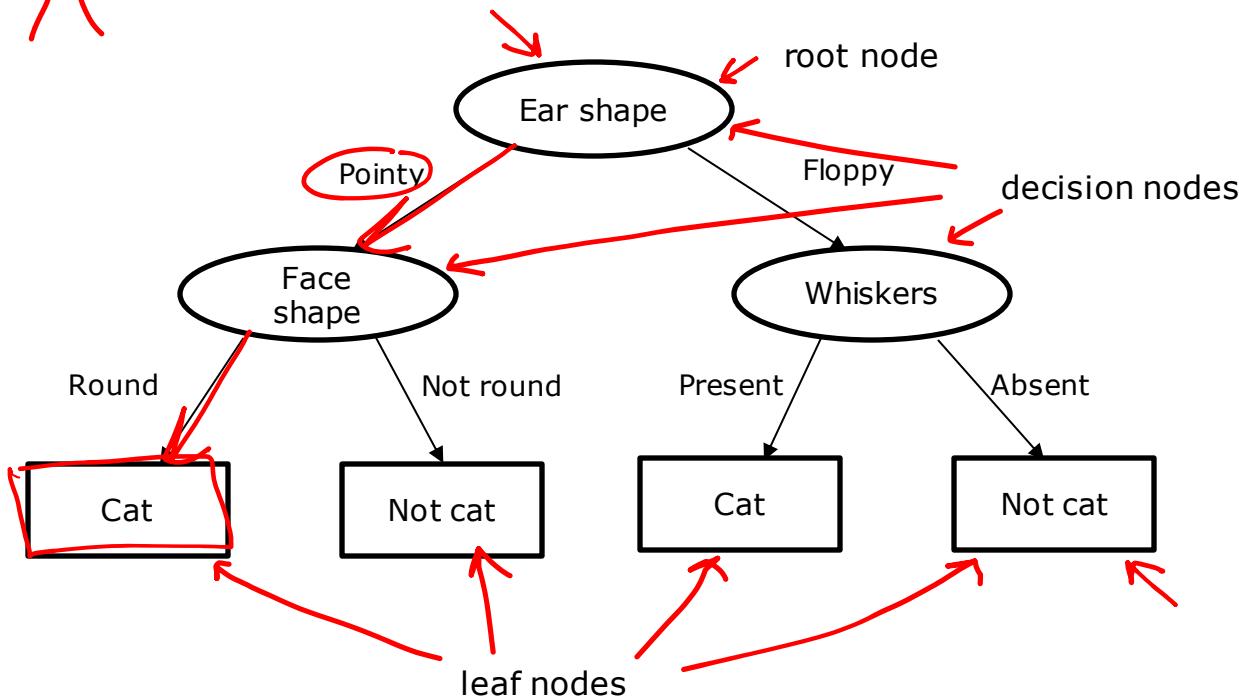
# Cat classification example

	Ear shape ( $x_1$ )	Face shape( $x_2$ )	Whiskers ( $x_3$ )	Cat
	Pointy ↗	Round ↗	Present ↗	1
	Floppy ↗	Not round ↗	Present	1
	Floppy	Round	Absent ↗	0
	Pointy	Not round	Present	0
	Pointy	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0

**Categorical (discrete values)**      X      y



# Decision Tree

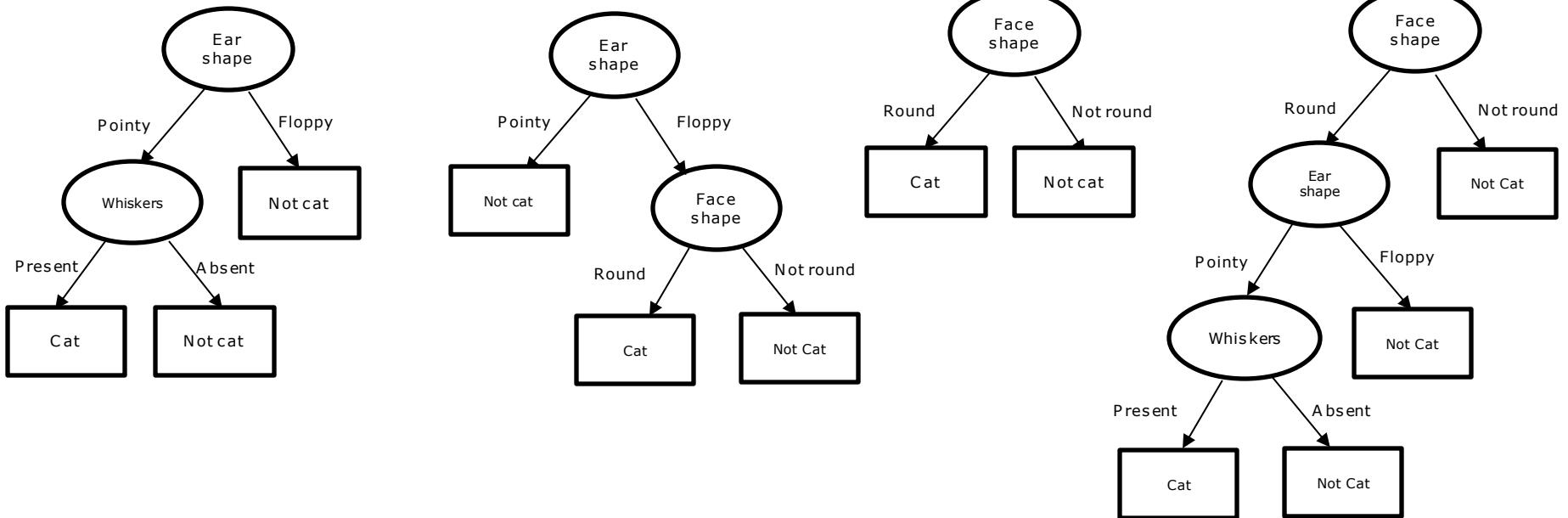


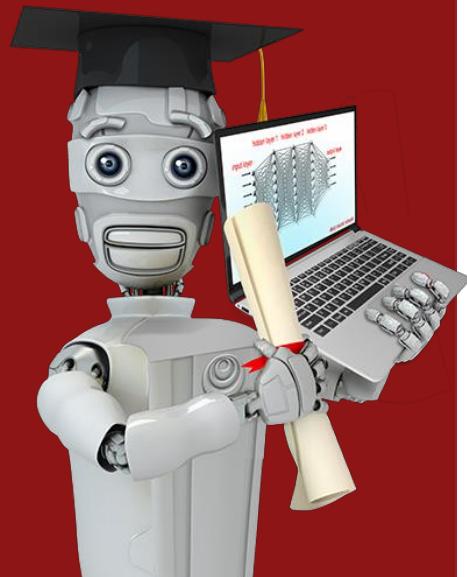
New test example



Ear shape: Pointy  
Face shape: Round  
Whiskers: Present

# Decision Tree





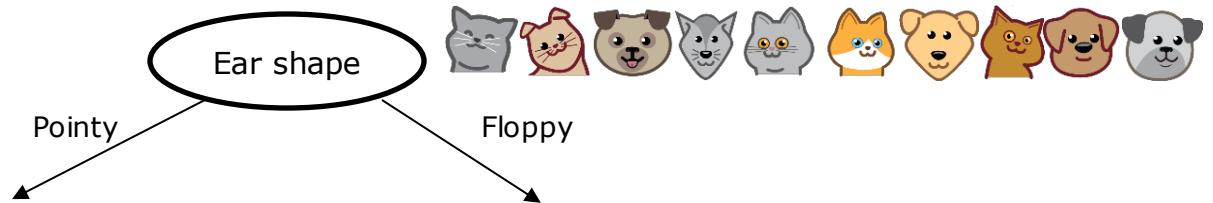
## Decision Trees

# Learning Process

# Decision Tree Learning



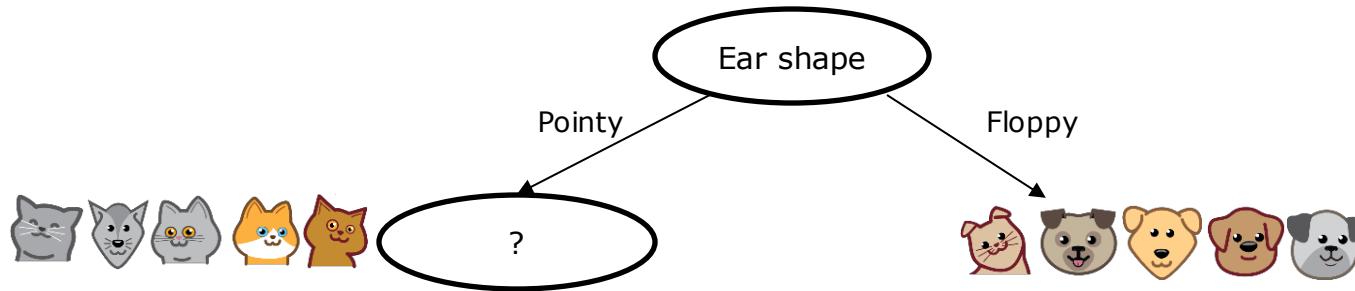
# Decision Tree Learning



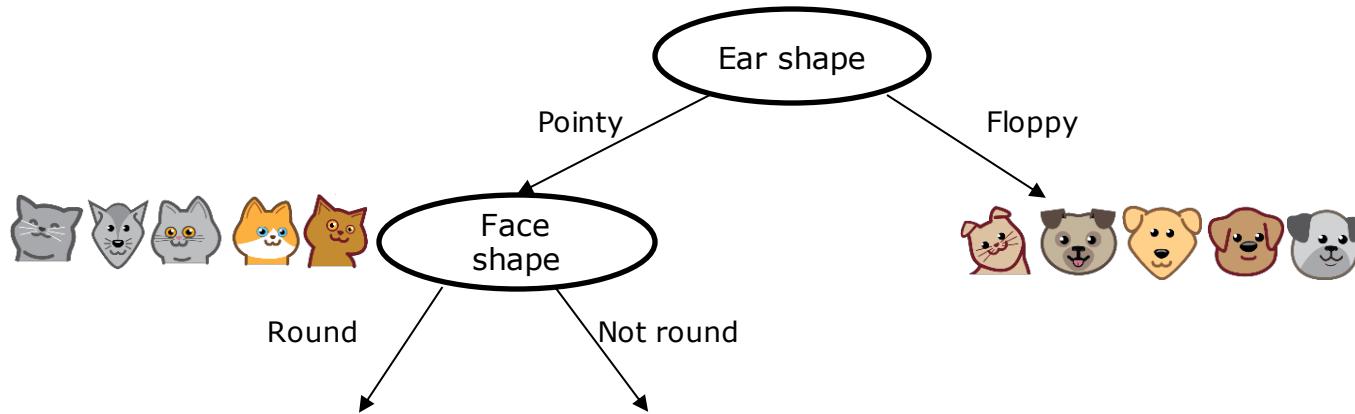
# Decision Tree Learning



# Decision Tree Learning

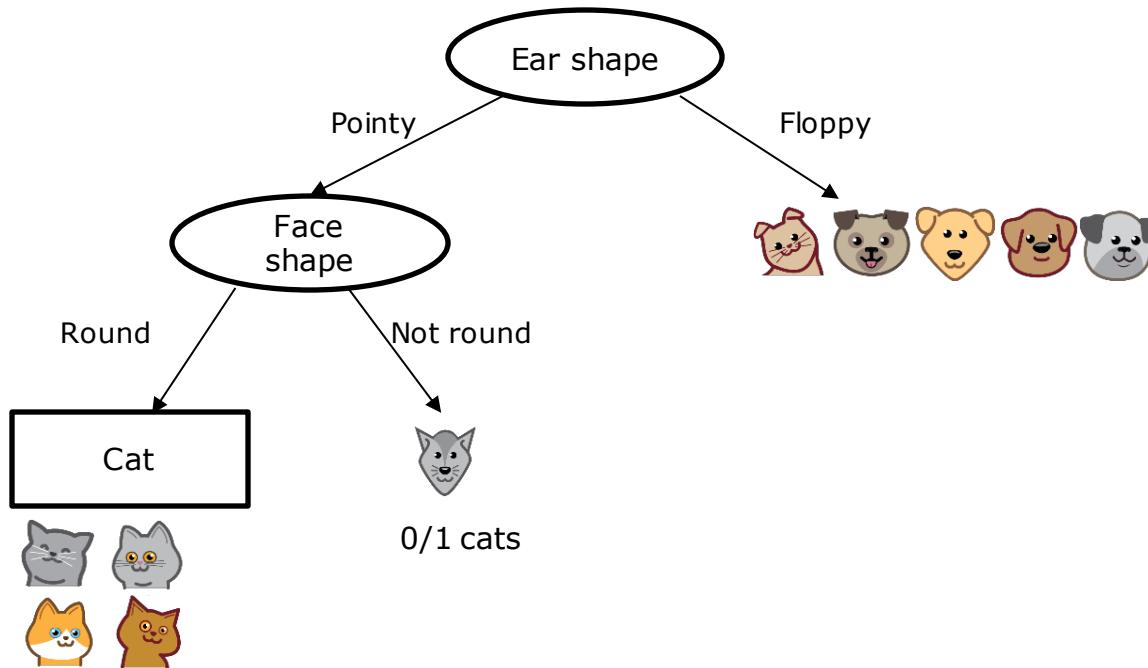


# Decision Tree Learning

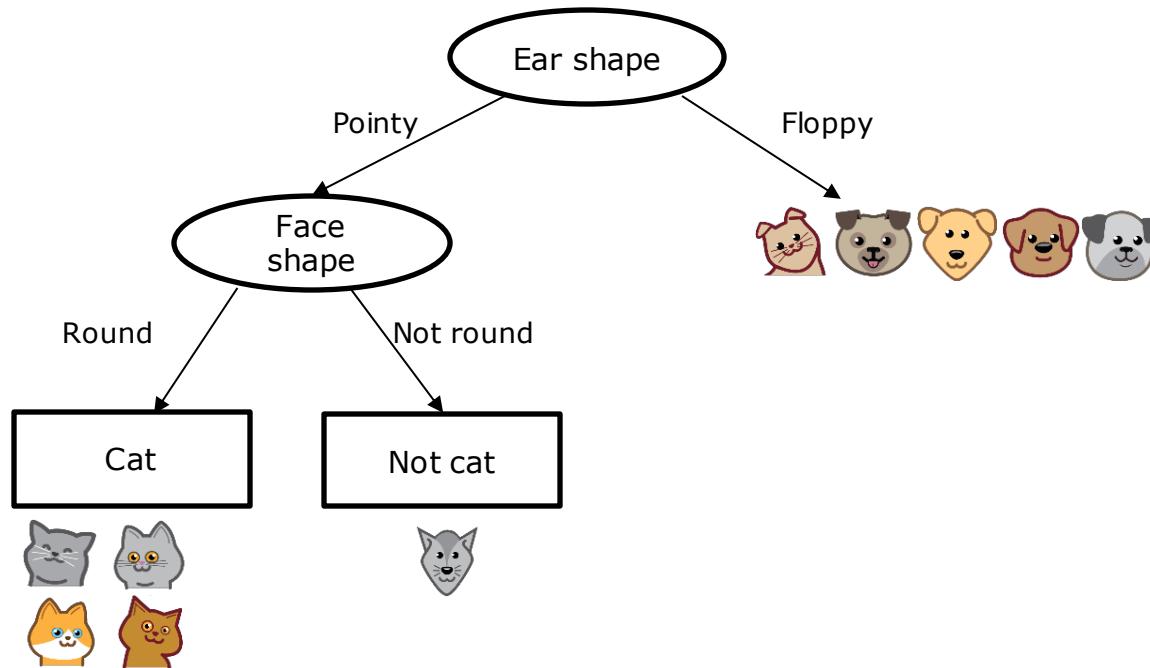


4/4 cats

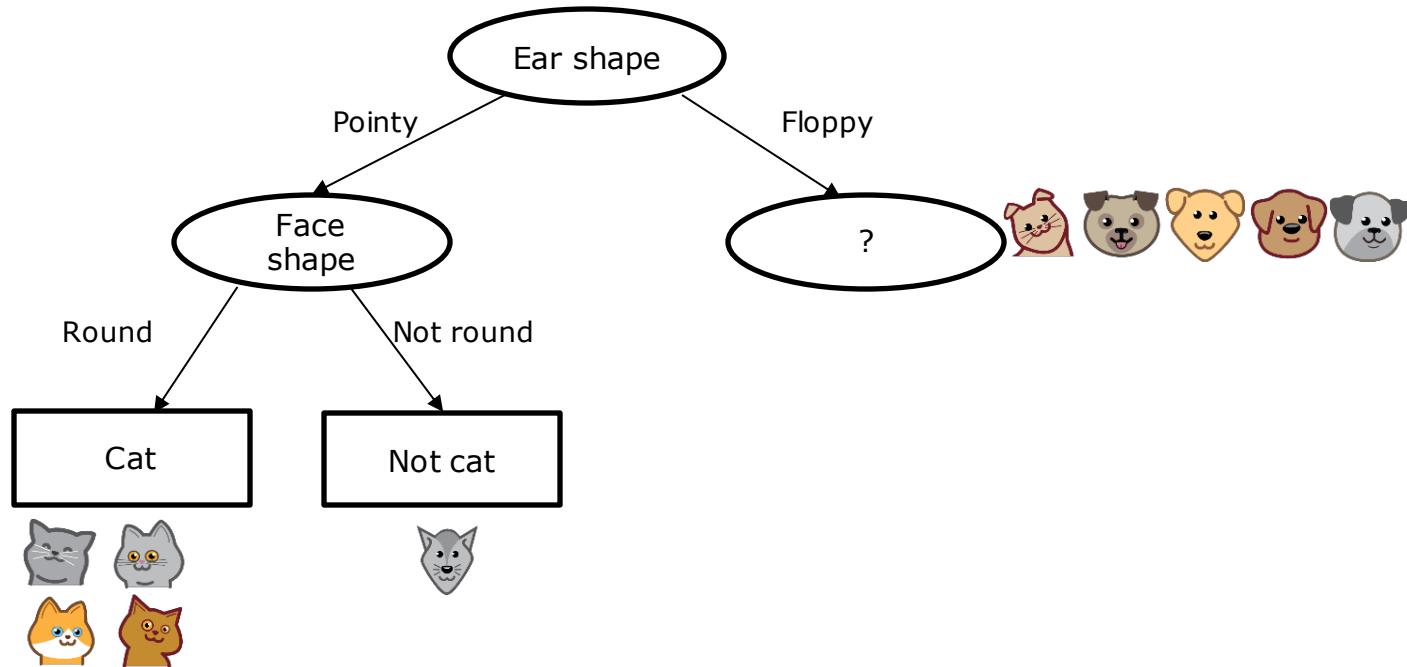
# Decision Tree Learning



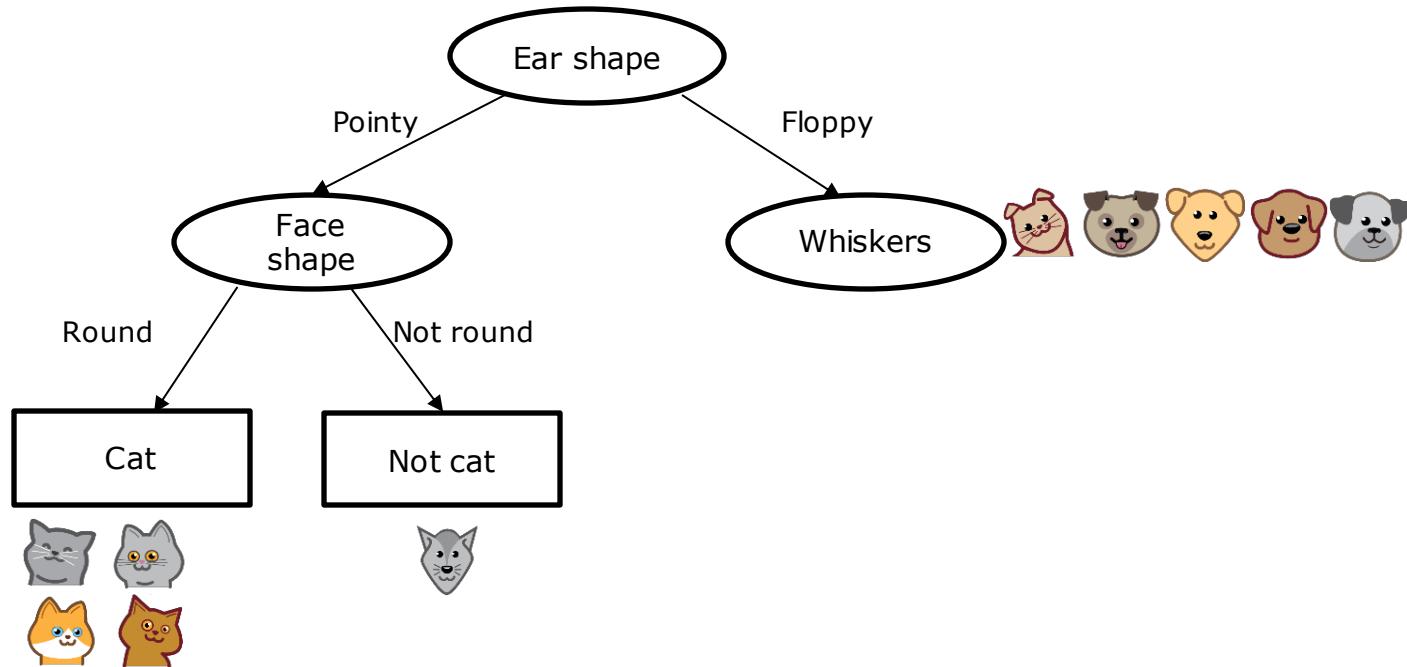
# Decision Tree Learning



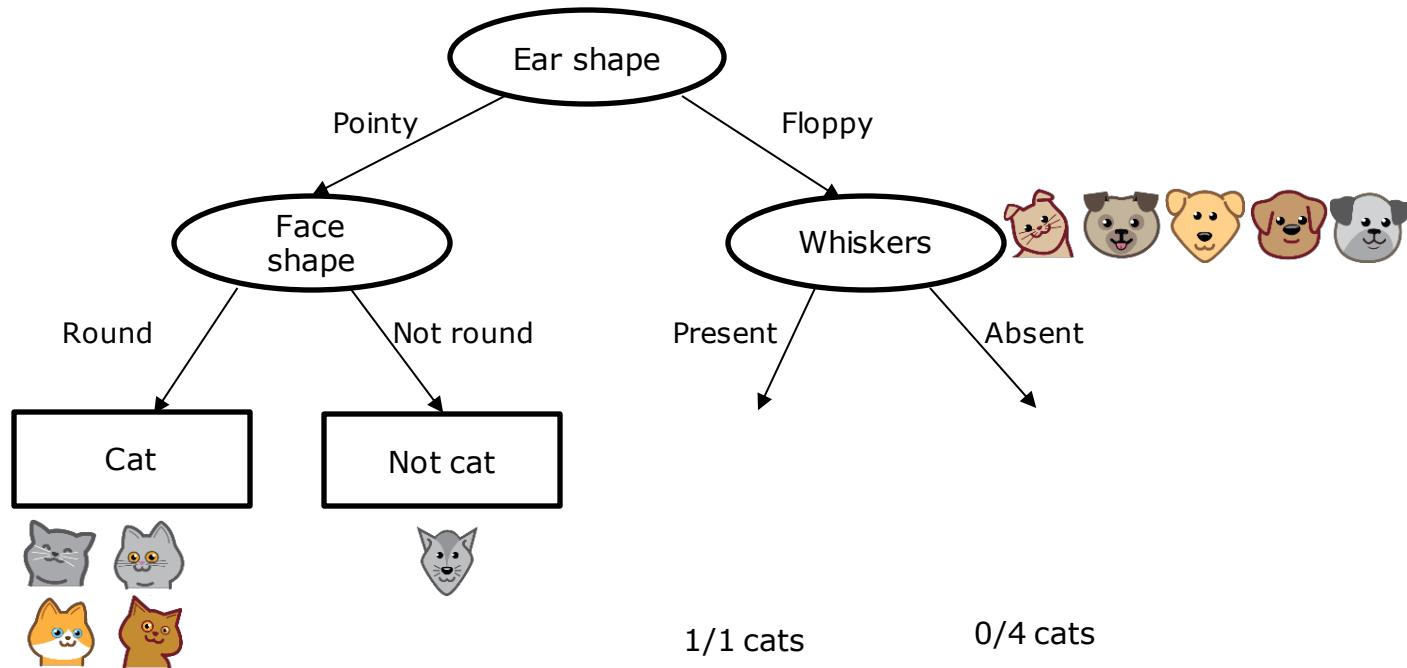
# Decision Tree Learning



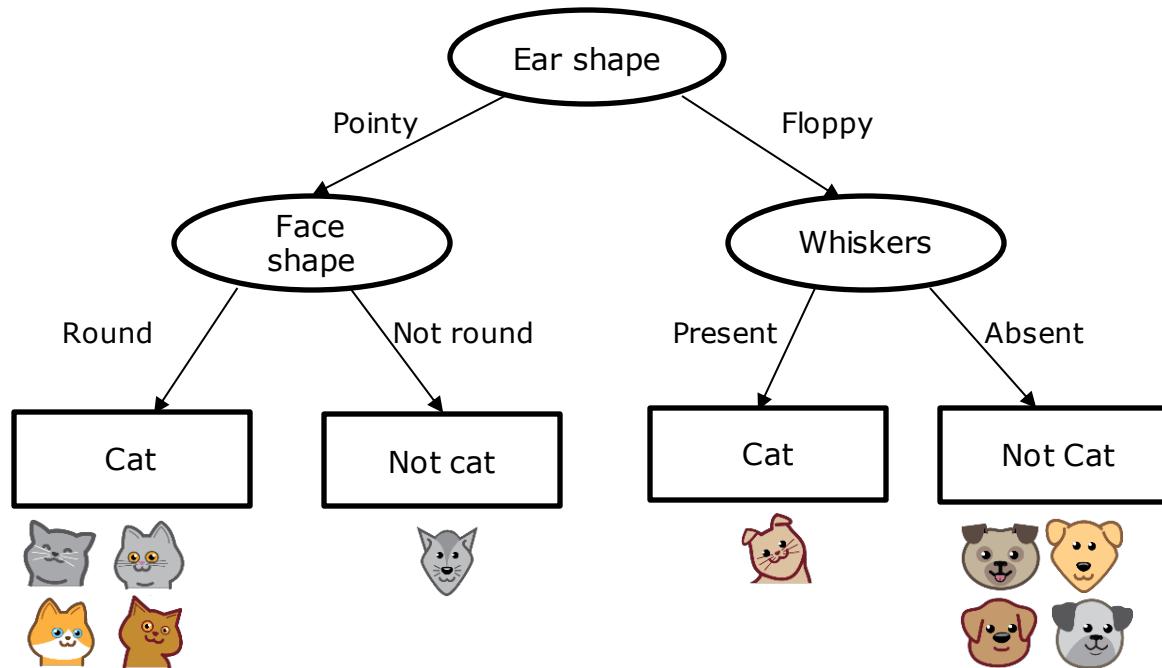
# Decision Tree Learning



# Decision Tree Learning



# Decision Tree Learning



# Decision Tree Learning

**Decision 1:** How to choose what feature to split on at each node?

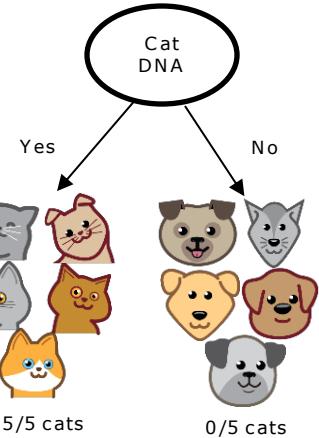
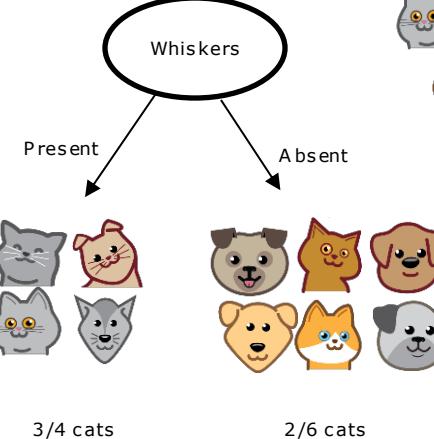
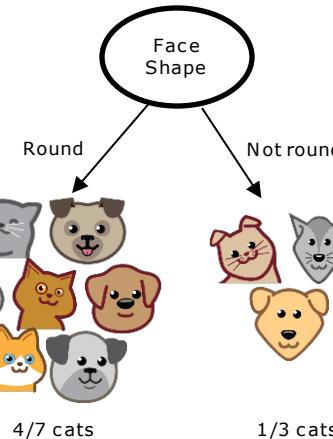
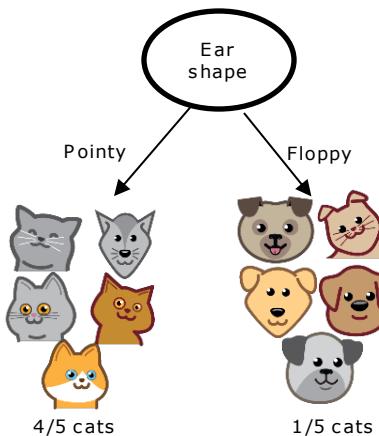
?

Maximize purity (or minimize impurity)

# Decision Tree Learning

**Decision 1:** How to choose what feature to split on at each node?

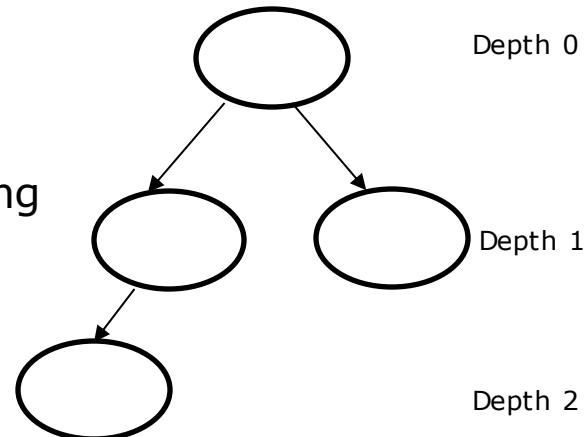
Maximize purity (or minimize impurity)



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

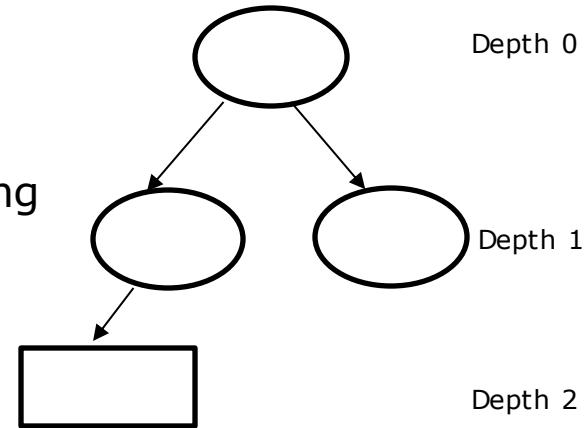
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

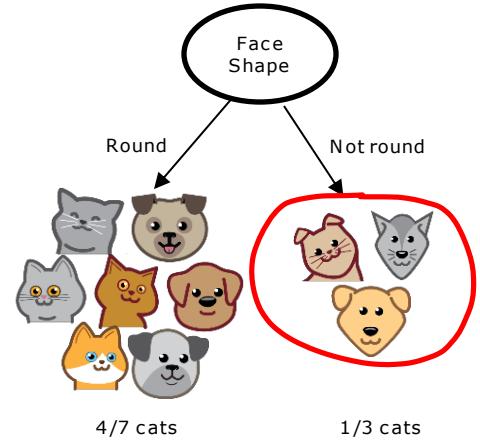
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

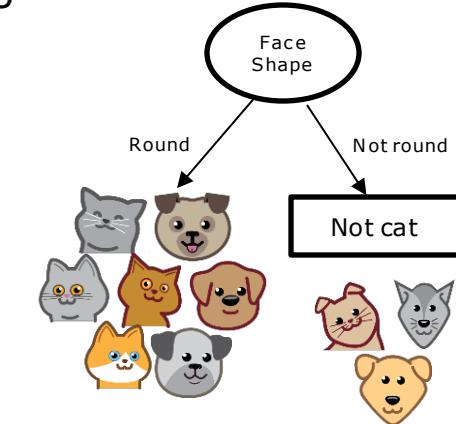
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



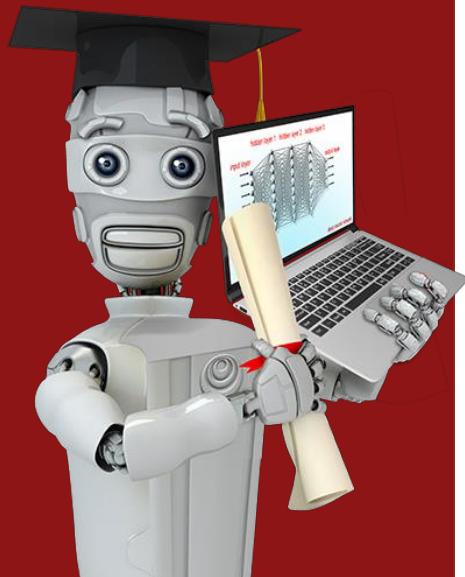
# Decision Tree Learning

**Decision 2:** When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



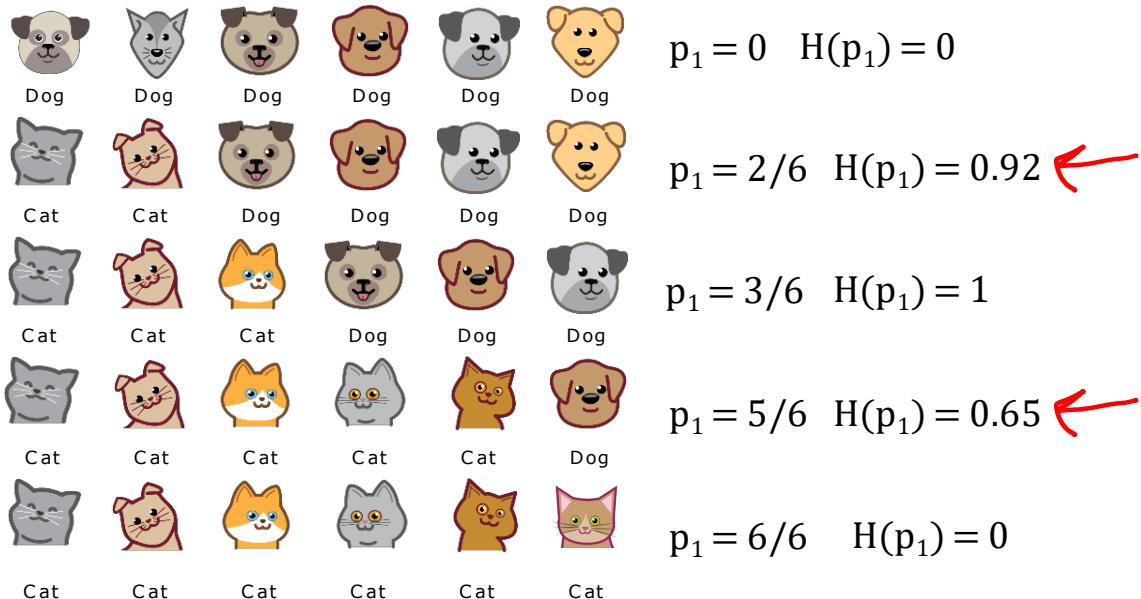
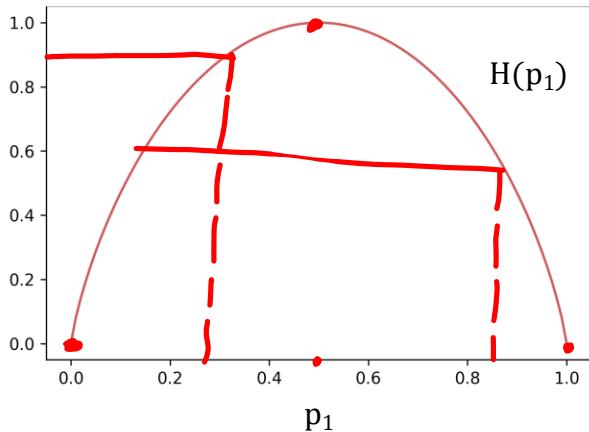
## Decision Tree Learning



# Measuring purity

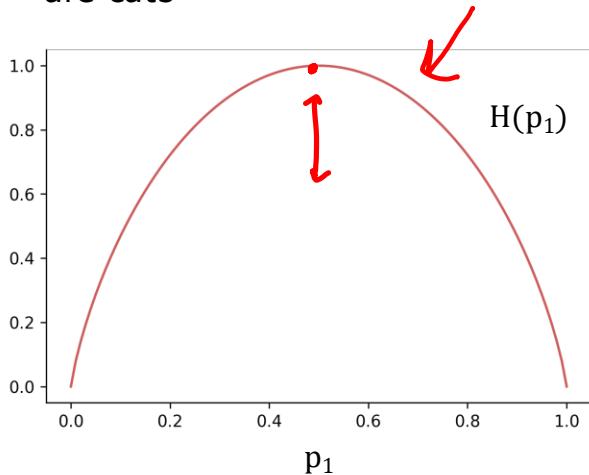
# Entropy as a measure of impurity

$p_1$  = fraction of examples that are cats



# Entropy as a measure of impurity

$p_1$  = fraction of examples that are cats



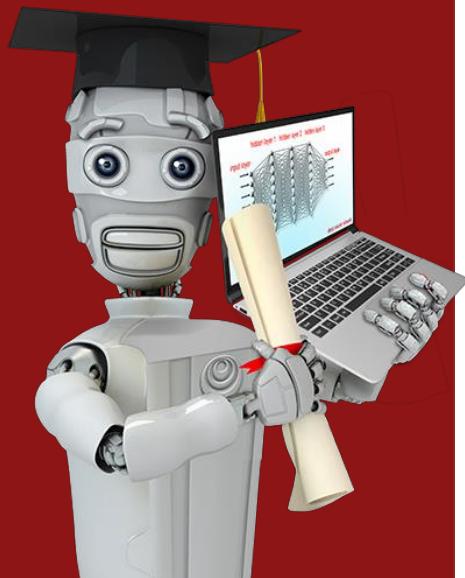
$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

$$= -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$



Note: " $0 \log(0)$ " = 0



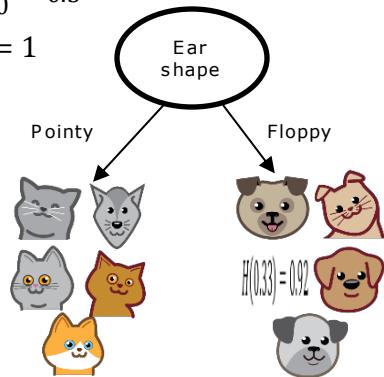
# Decision Tree Learning

## Choosing a split: Information Gain

# Choosing a split

$$p_1 = \frac{5}{10} = 0.5$$

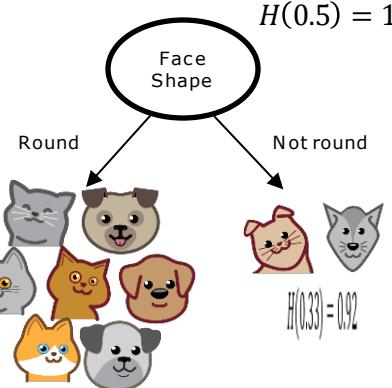
$$H(0.5) = 1$$



$$p_1 = \frac{4}{5} = 0.8 \quad p_1 = \frac{1}{5} = 0.2$$

$$H(0.8) = 0.72 \quad H(0.2) = 0.72$$

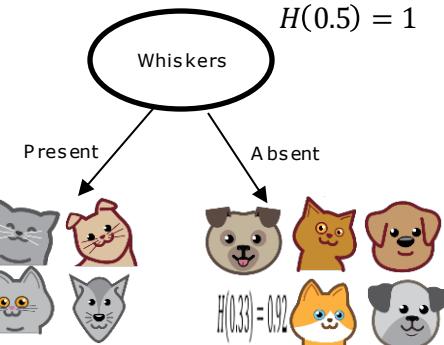
$$H(0.5) - \left( \frac{5}{10} H(0.8) + \frac{5}{10} H(0.2) \right) \\ = 0.28$$



$$p_1 = \frac{4}{7} = 0.57 \quad p_1 = \frac{1}{3} = 0.33$$

$$H(0.57) = 0.99 \quad H(0.33) = 0.92$$

$$H(0.5) - \left( \frac{7}{10} H(0.57) + \frac{3}{10} H(0.33) \right) \\ = 0.03$$

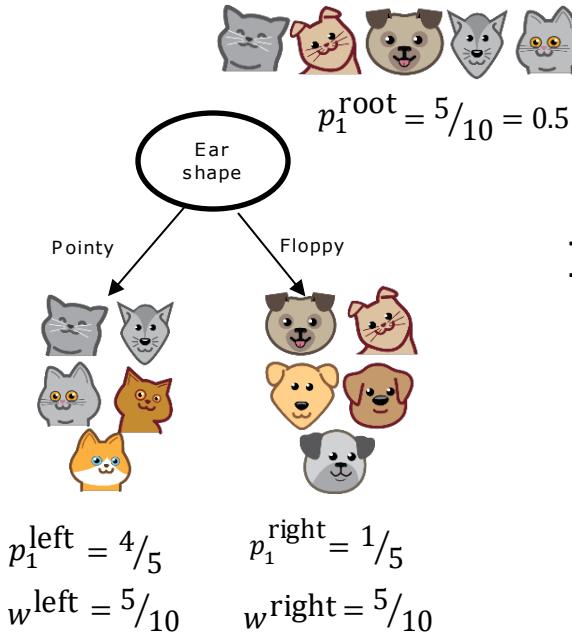


$$p_1 = \frac{3}{4} = 0.75 \quad p_1 = \frac{2}{6} = 0.33$$

$$H(0.75) = 0.81 \quad H(0.33) = 0.92$$

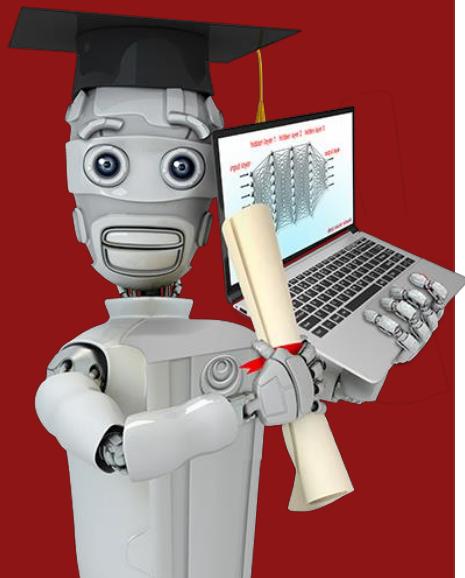
$$H(0.5) - \left( \frac{4}{10} H(0.75) + \frac{6}{10} H(0.33) \right) \\ = 0.12$$

# Information Gain



Information gain

$$= H(p_1^{\text{root}}) - \left( w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}) \right)$$



## Decision Tree Learning

Putting it together

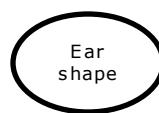
# Decision Tree Learning

- Start with all examples at the root node
- Calculate information gain for all possible features, and pick the one with the highest information gain
- Split dataset according to selected feature, and create left and right branches of the tree
- Keep repeating splitting process until stopping criteria is met:
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a maximum depth
  - Information gain from additional splits is less than threshold
  - When number of examples in a node is below a threshold

# Recursive splitting



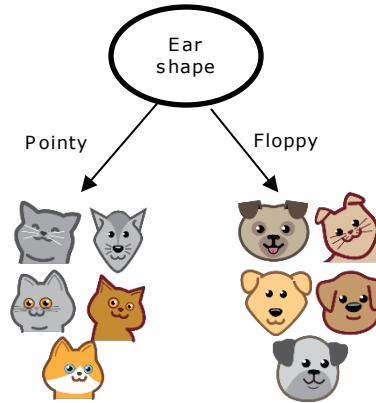
# Recursive splitting



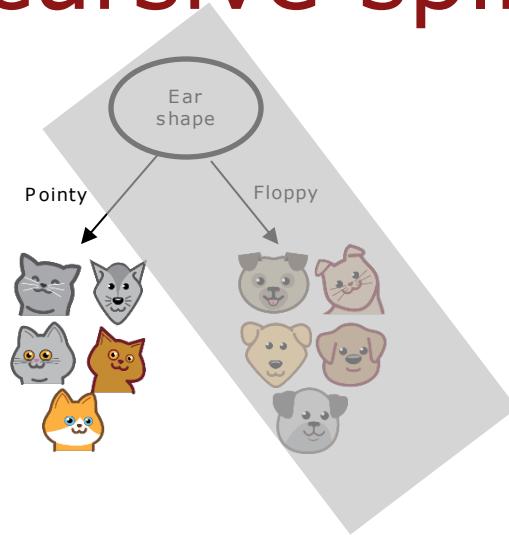
# Recursive splitting



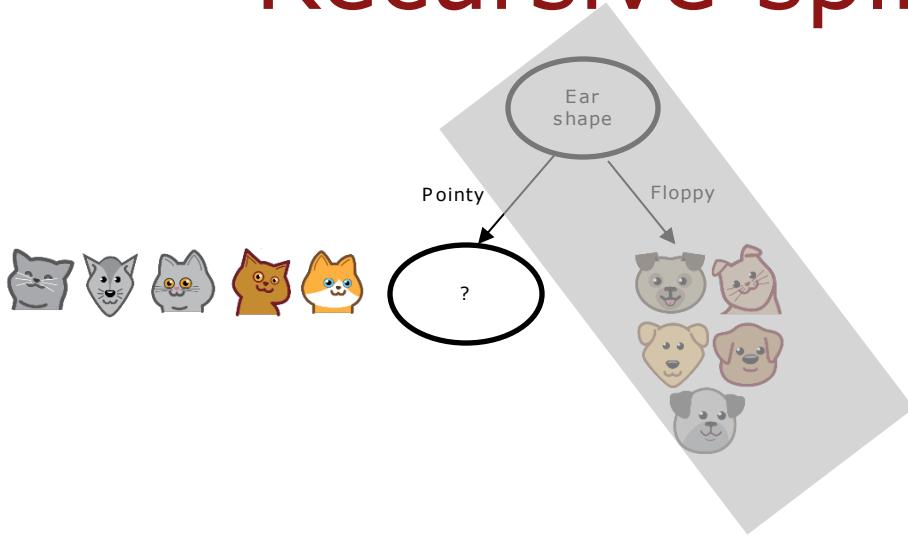
# Recursive splitting



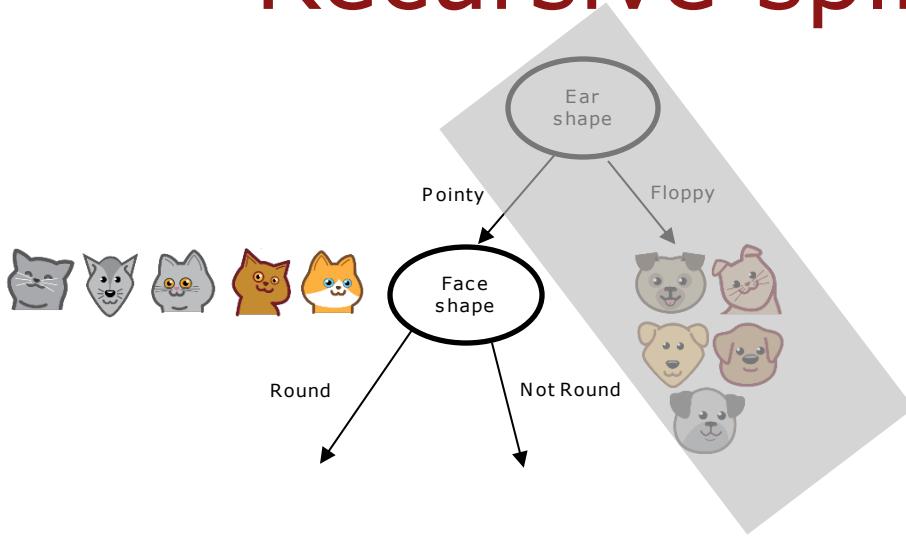
# Recursive splitting



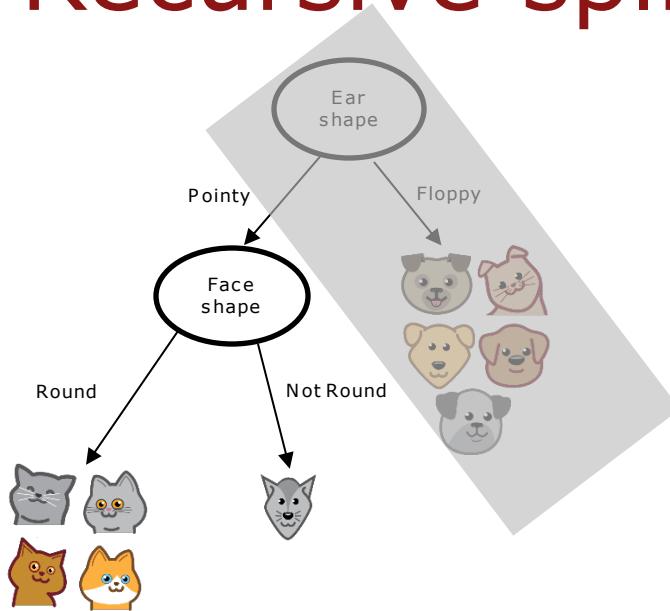
# Recursive splitting



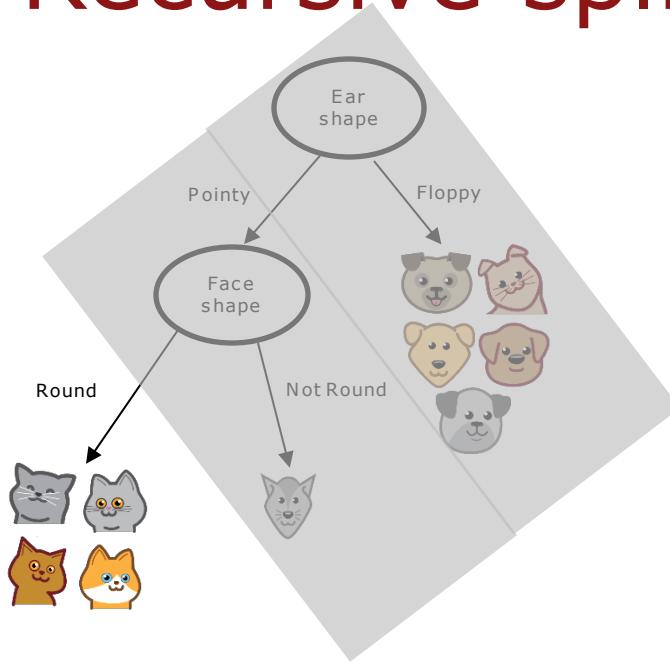
# Recursive splitting



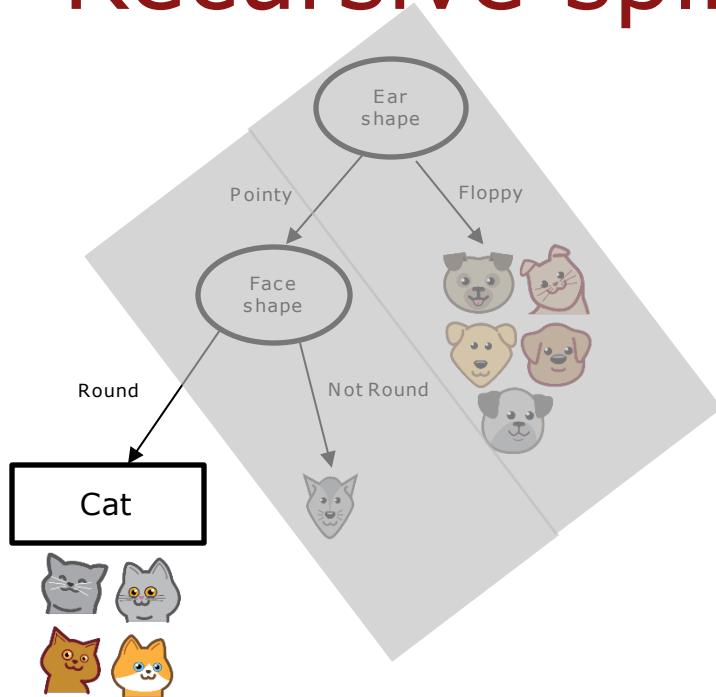
# Recursive splitting



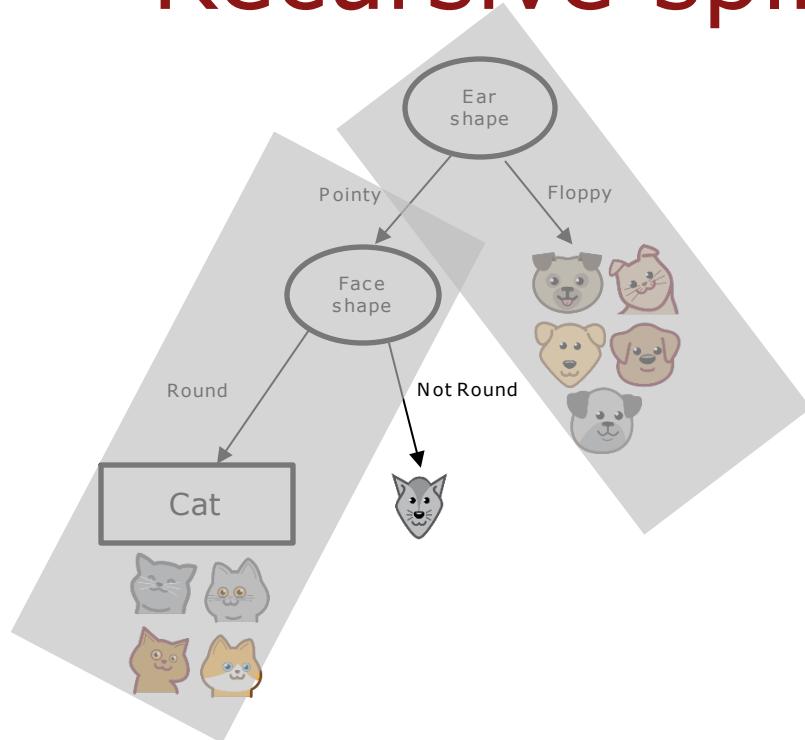
# Recursive splitting



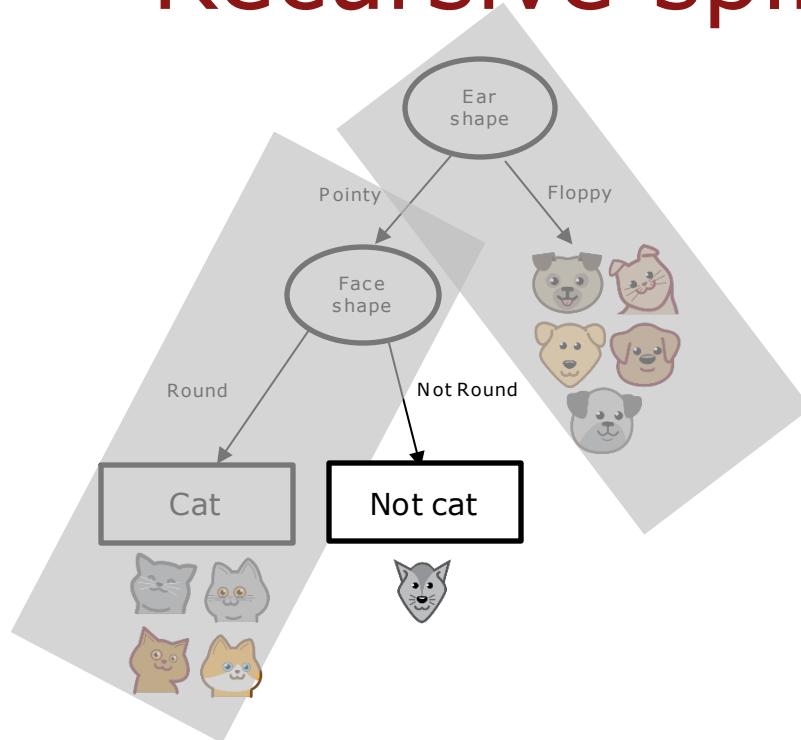
# Recursive splitting



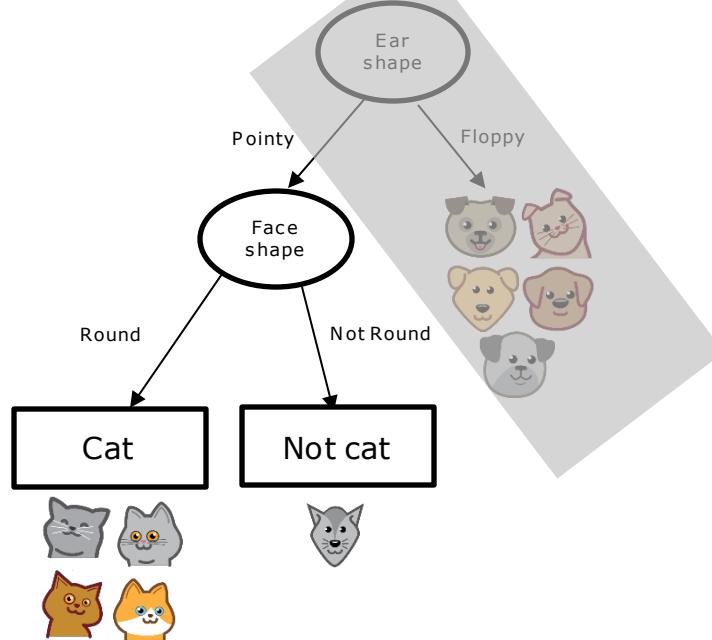
# Recursive splitting



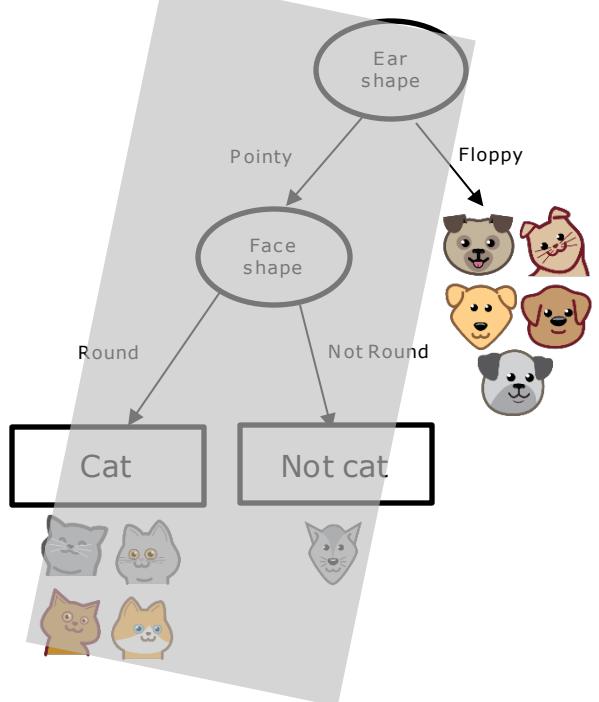
# Recursive splitting



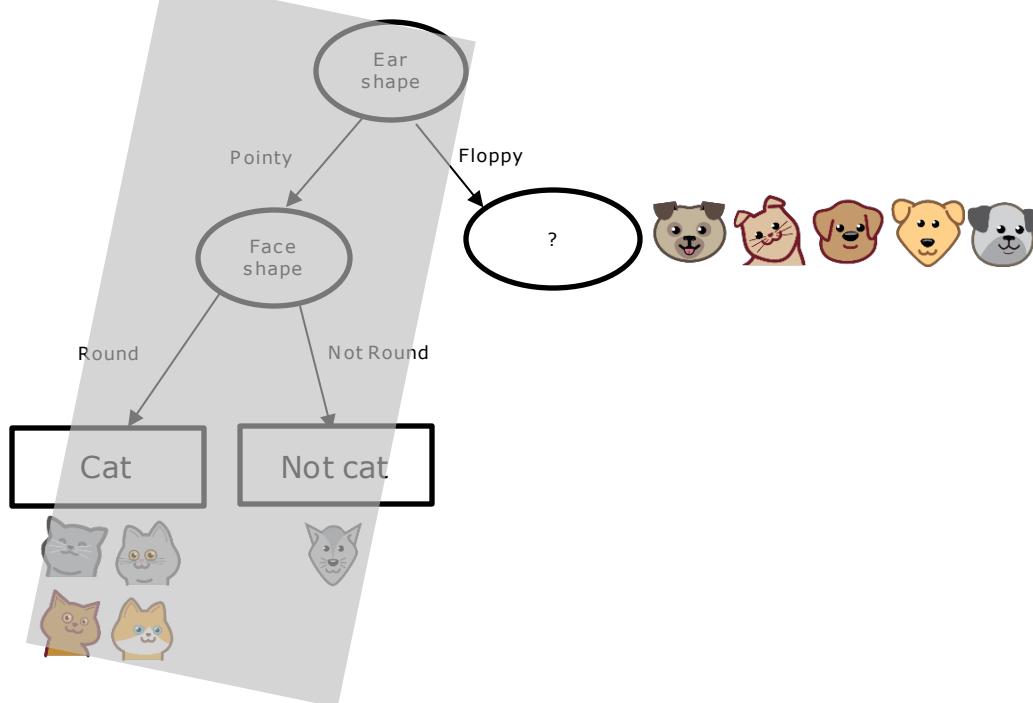
# Recursive splitting



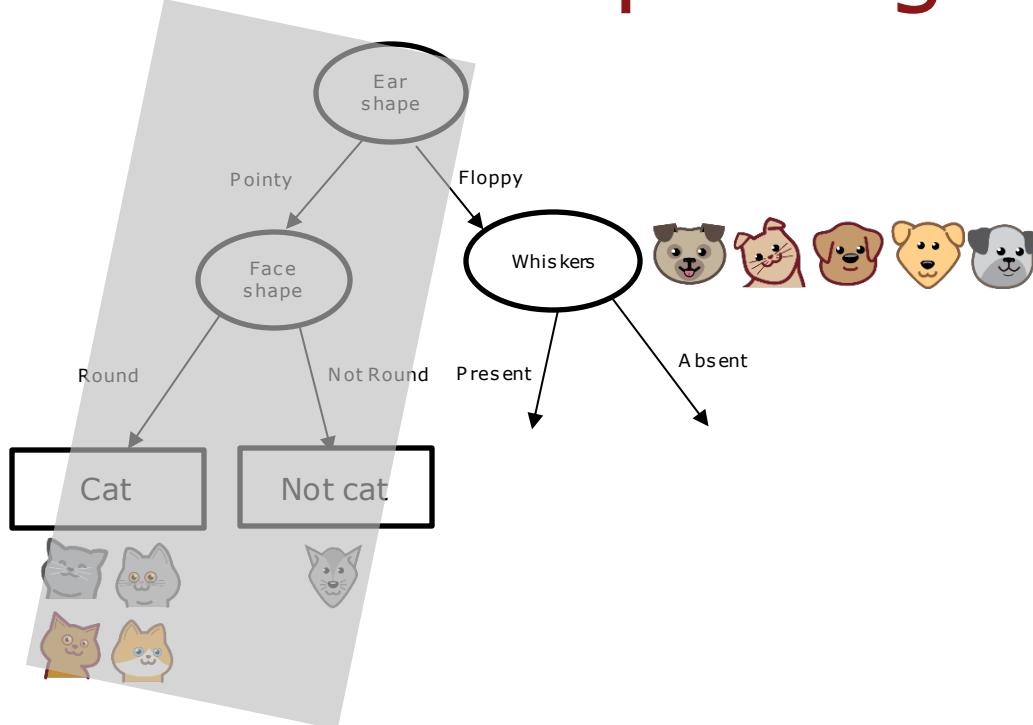
# Recursive splitting



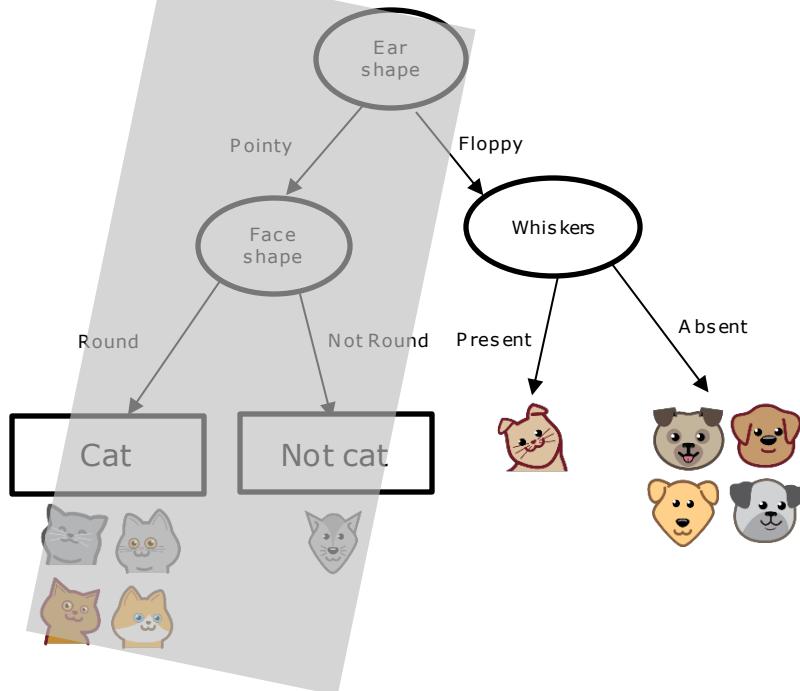
# Recursive splitting



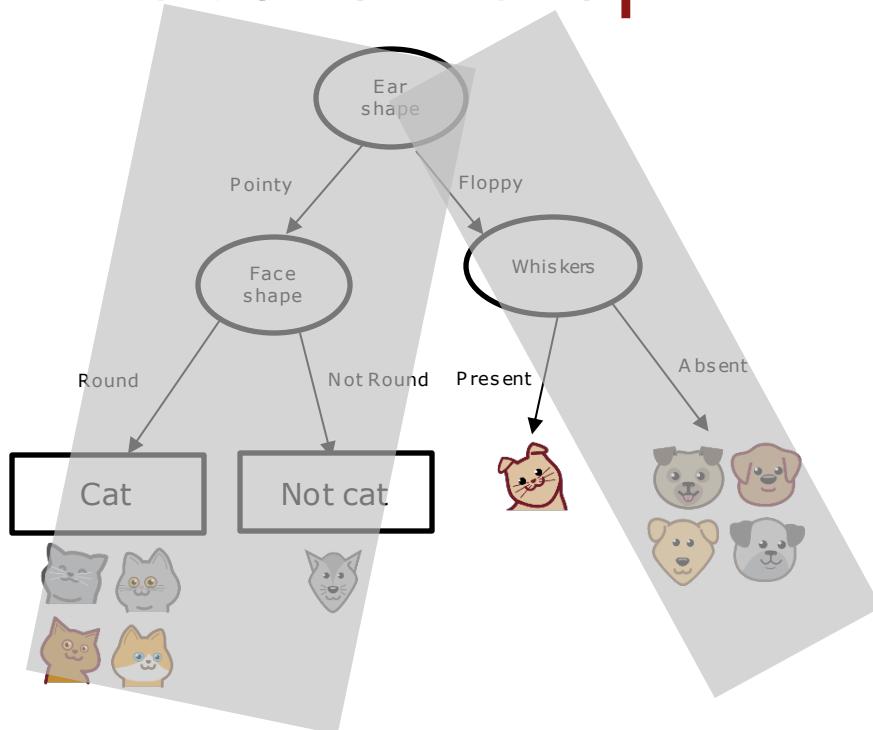
# Recursive splitting



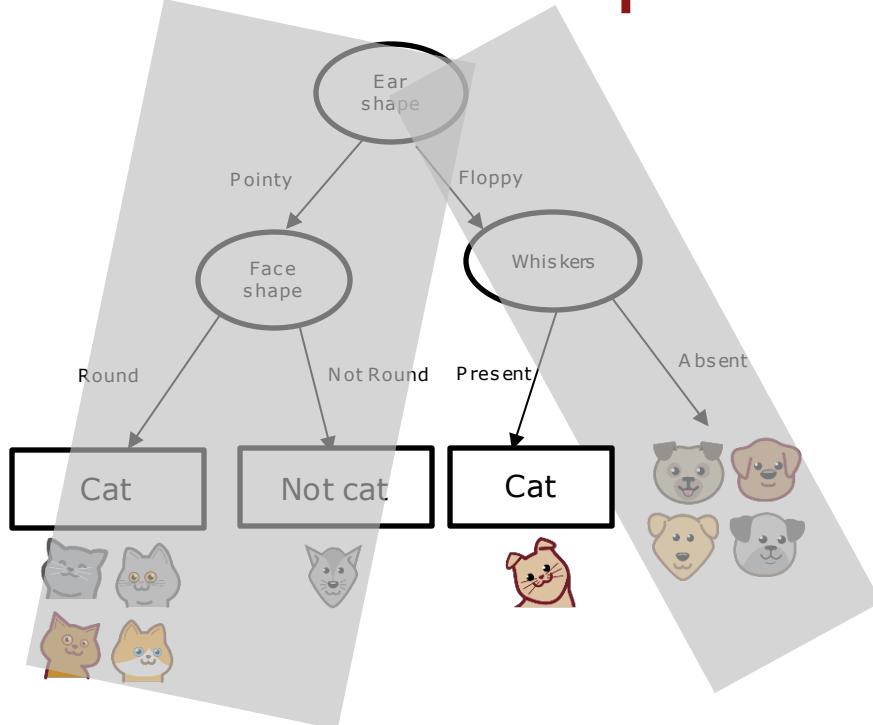
# Recursive splitting



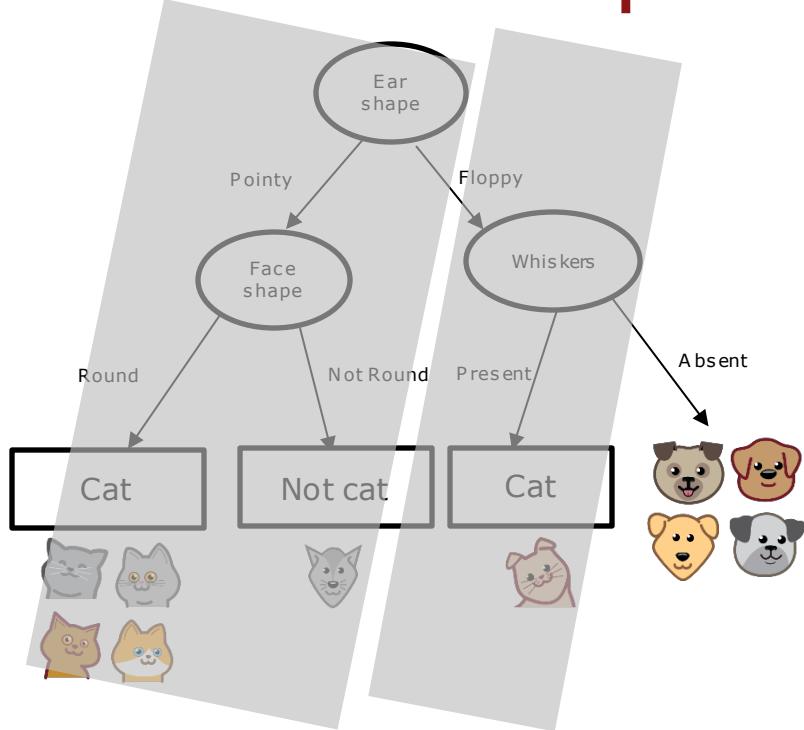
# Recursive splitting



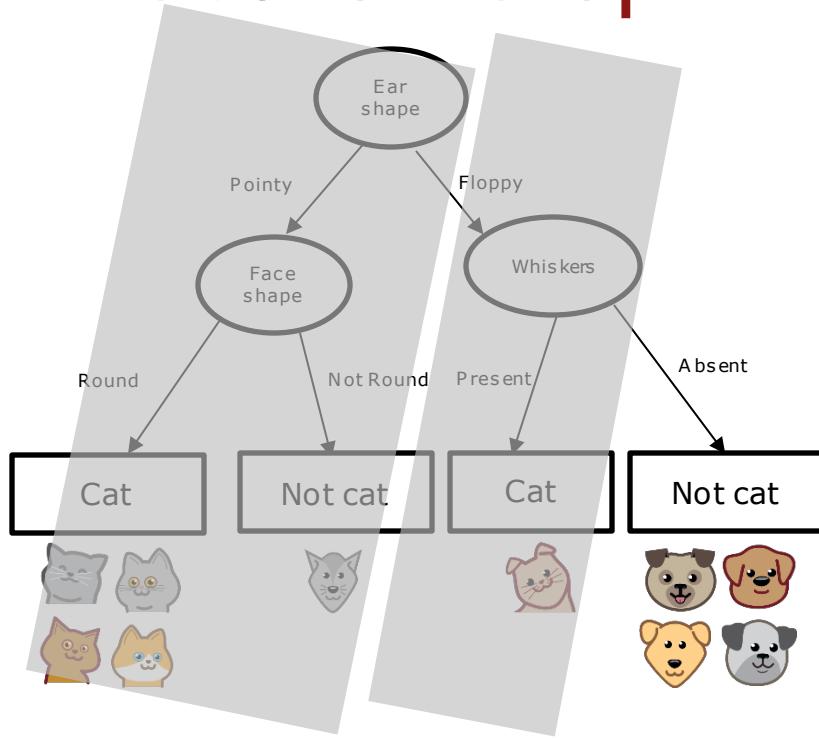
# Recursive splitting



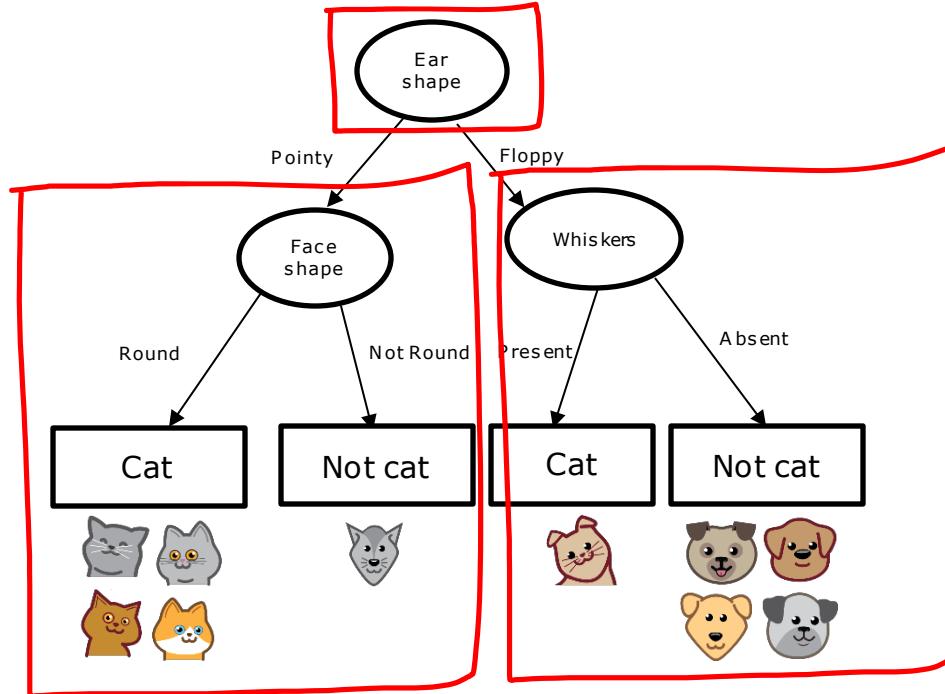
# Recursive splitting



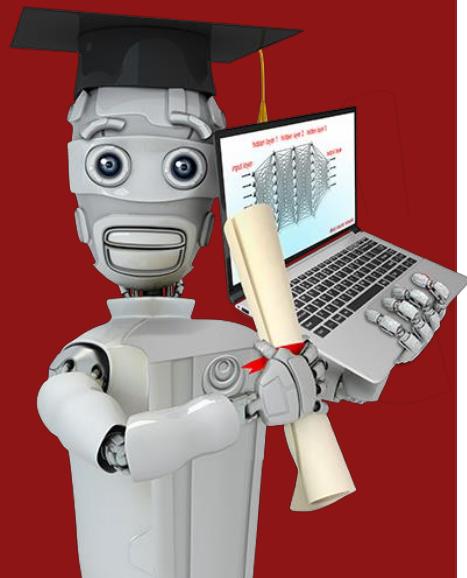
# Recursive splitting



# Recursive splitting



Recursive algorithm

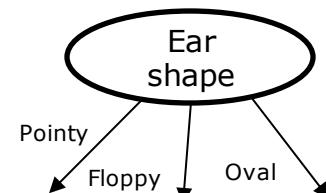


# Decision Tree Learning

Using one-hot encoding of categorical features

# Features with three possible values

	Ear shape ( $x_1$ )	Face shape ( $x_2$ )	Whiskers ( $x_3$ )	Cat ( $y$ )
	Pointy ↗	Round	Present	1
	Oval	Not round	Present	1
	Oval ↗	Round	Absent	0
	Pointy	Not round	Present	0
	Oval	Round	Present	1
	Pointy	Round	Absent	1
	Floppy ↗	Not round	Absent	0
	Oval	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0



3 possible values

# One hot encoding

Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat	
	Pointy	1	0	0	Round	Present	1
	Oval	0	0	1	Not round	Present	1
	Oval	0	0	1	Round	Absent	0
	Pointy	1	0	0	Not round	Present	0
	Oval	0	0	1	Round	Present	1
	Pointy	1	0	0	Round	Absent	1
	Floppy	0	1	0	Not round	Absent	0
	Oval	0	0	1	Round	Absent	1
	Floppy	0	1	0	Round	Absent	0
	Floppy	0	1	0	Round	Absent	0

# One hot encoding

If a categorical feature can take on  $k$  values,  
create  $k$  binary features (0 or 1 valued).

# One hot encoding

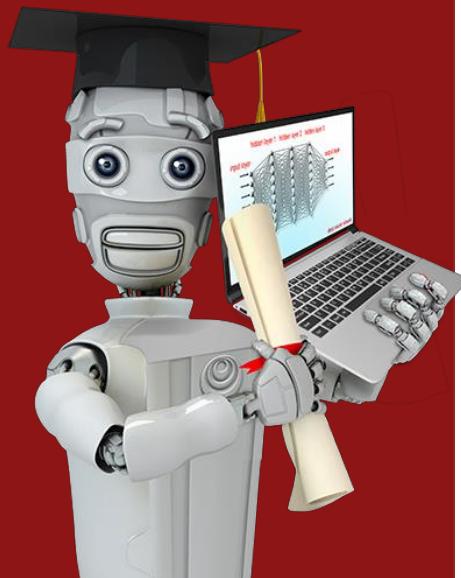
Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat	
	Pointy	1	0	0	Round	Present	1
	Oval	0	0	1	Not round	Present	1
	Oval	0	0	1	Round	Absent	0
	Pointy	1	0	0	Not round	Present	0
	Oval	0	0	1	Round	Present	1
	Pointy	1	0	0	Round	Absent	1
	Floppy	0	1	0	Not round	Absent	0
	Oval	0	0	1	Round	Absent	1
	Floppy	0	1	0	Round	Absent	0
	Floppy	0	1	0	Round	Absent	0

# One hot encoding and neural networks

	Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
	1	0	0	Round 1	Present 1	1
	0	0	1	Not round 0	Present 1	1
	0	0	1	Round 1	Absent 0	0
	1	0	0	Not round 0	Present 1	0
	0	0	1	Round 1	Present 1	1
	1	0	0	Round 1	Absent 0	1
	0	1	0	Not round 0	Absent 0	1
	0	0	1	Round 1	Absent 0	1
	0	1	0	Round 1	Absent 0	1
	0	1	0	Round 1	Absent 0	1

# Decision Tree Learning

Continuous valued features

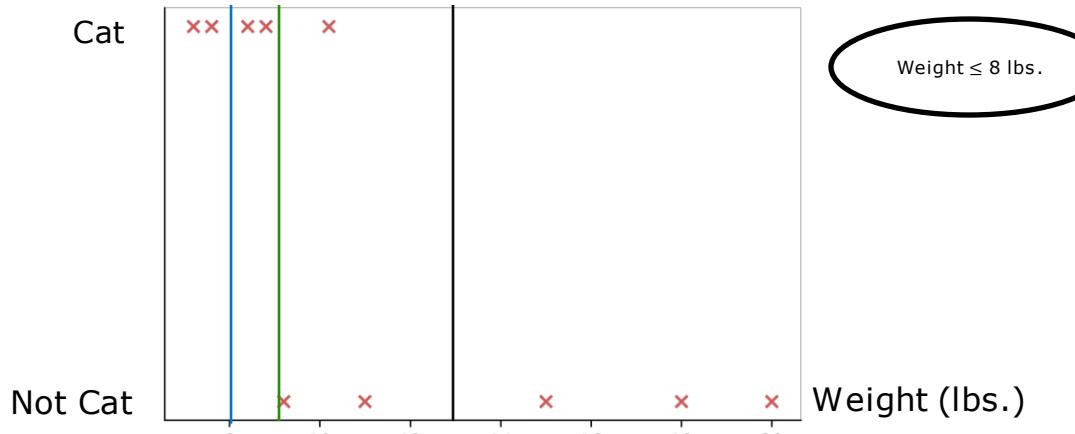


# Continuous features



Ear shape	Face shape	Whiskers	Weight (lbs.)	Cat	
	Pointy	Round	Present	7.2	1
	Floppy	Not round	Present	8.8	1
	Floppy	Round	Absent	15	0
	Pointy	Not round	Present	9.2	0
	Pointy	Round	Present	8.4	1
	Pointy	Round	Absent	7.6	1
	Floppy	Not round	Absent	11	0
	Pointy	Round	Absent	10.2	1
	Floppy	Round	Absent	18	0
	Floppy	Round	Absent	20	0

# Splitting on a continuous variable



Weight  $\leq 8$  lbs.

Weight  $\leq 9$  lbs.

Yes

No



$$H(0.5) - \left( \frac{2}{10} H\left(\frac{2}{2}\right) + \frac{8}{10} H\left(\frac{3}{8}\right) \right) = 0.24$$

$$H(0.5) - \left( \frac{4}{10} H\left(\frac{4}{4}\right) + \frac{6}{10} H\left(\frac{1}{6}\right) \right) = 0.61$$

$$H(0.5) - \left( \frac{7}{10} H\left(\frac{5}{7}\right) + \frac{3}{10} H\left(\frac{0}{3}\right) \right) = 0.40$$

# Decision Tree Learning

## Regression Trees (optional)



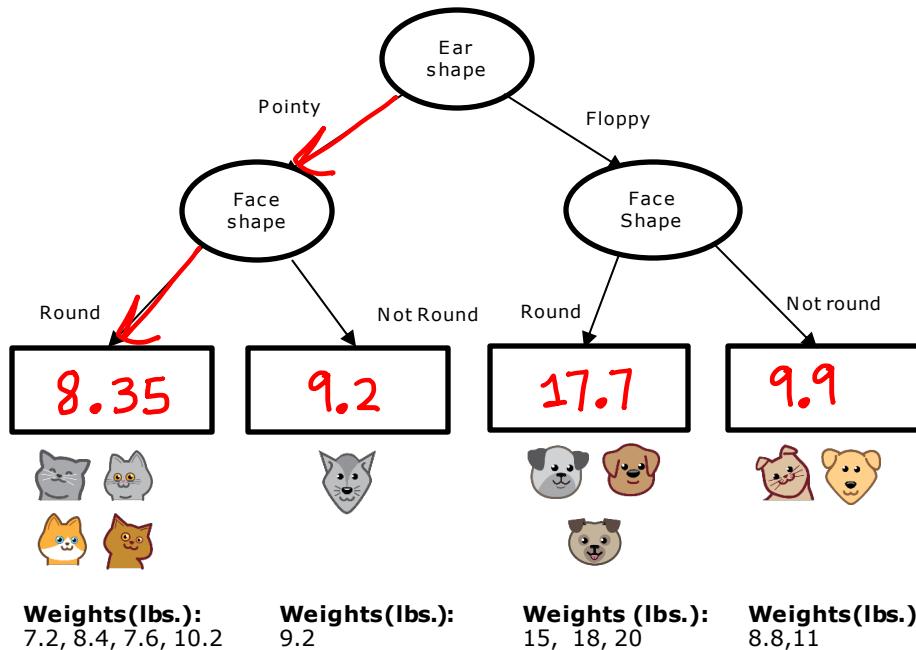
# Regression with Decision Trees: Predicting a number

	Ear shape	Face shape	Whiskers	Weight (lbs.)
	Pointy	Round	Present	7.2
	Floppy	Not round	Present	8.8
	Floppy	Round	Absent	15
	Pointy	Not round	Present	9.2
	Pointy	Round	Present	8.4
	Pointy	Round	Absent	7.6
	Floppy	Not round	Absent	11
	Pointy	Round	Absent	10.2
	Floppy	Round	Absent	18
	Floppy	Round	Absent	20

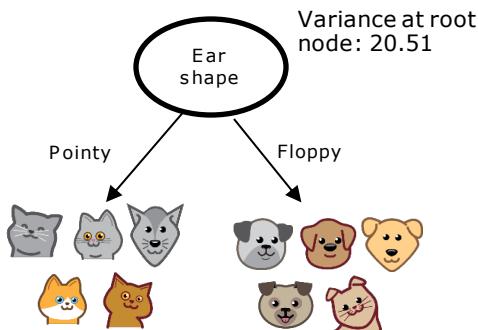
**X**

**y**

# Regression with Decision Trees



# Choosing a split

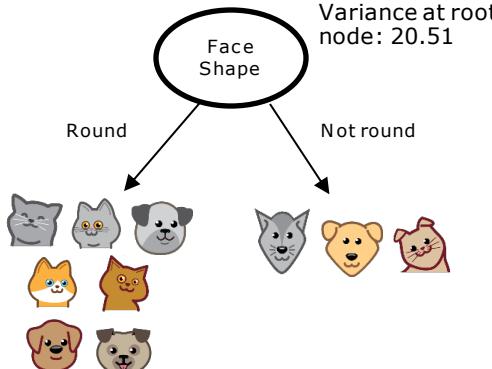


Weights: 7.2,  
9.2, 8.4, 7.6, 10.2

Variance: 1.47

$$w^{\text{left}} = \frac{5}{10} \quad w^{\text{right}} = \frac{5}{10}$$

$$20.51 - \left( \frac{5}{10} * 1.47 + \frac{5}{10} * 21.87 \right) \\ = 8.84$$

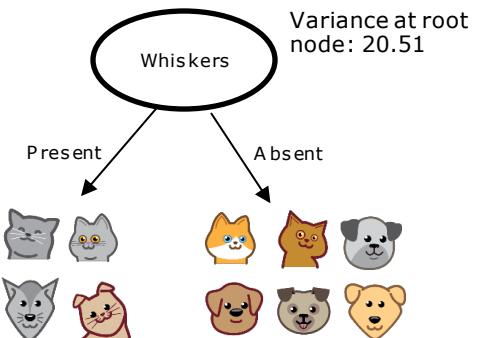


Weights: 7.2, 15,  
8.4, 7.6, 10.2, 18, 20

Variance: 27.80

$$w^{\text{left}} = \frac{7}{10} \quad w^{\text{right}} = \frac{3}{10}$$

$$20.51 - \left( \frac{7}{10} * 27.80 + \frac{3}{10} * 1.37 \right) \\ = 0.64$$

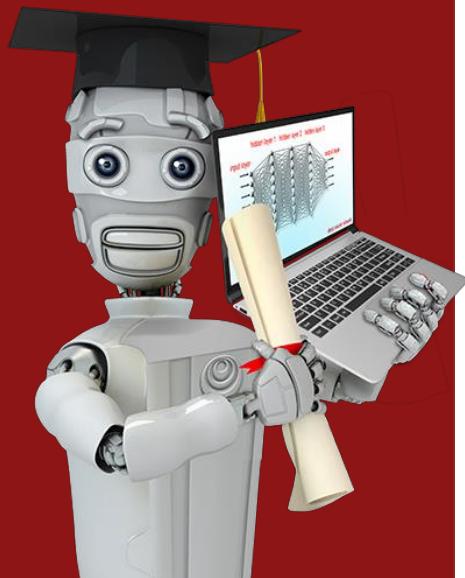


Weights: 7.2, 8.8,  
9.2, 8.4

Variance: 0.75

$$w^{\text{left}} = \frac{4}{10} \quad w^{\text{right}} = \frac{6}{10}$$

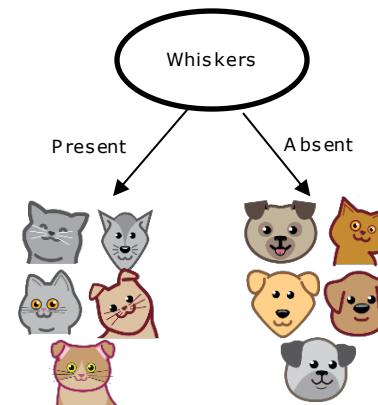
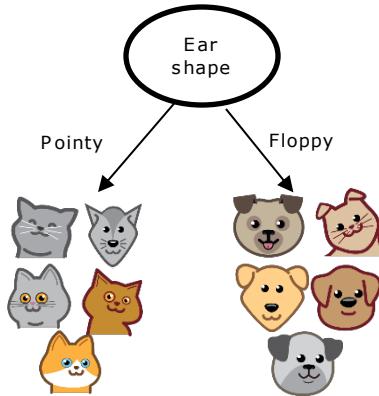
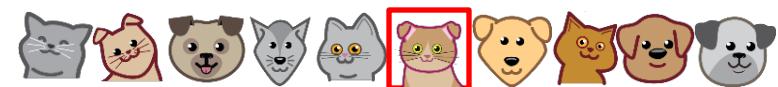
$$20.51 - \left( \frac{4}{10} * 0.75 + \frac{6}{10} * 23.32 \right) \\ = 6.22$$



## Tree ensembles

Using multiple decision trees

# Trees are highly sensitive to small changes of the data

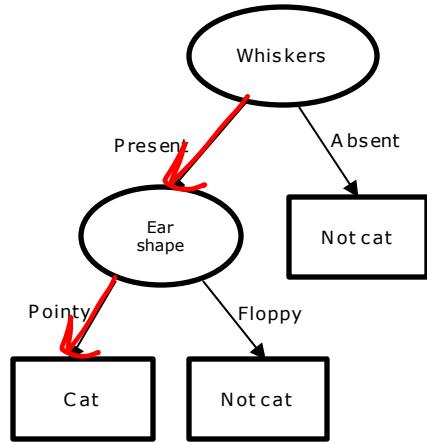


# Tree ensemble

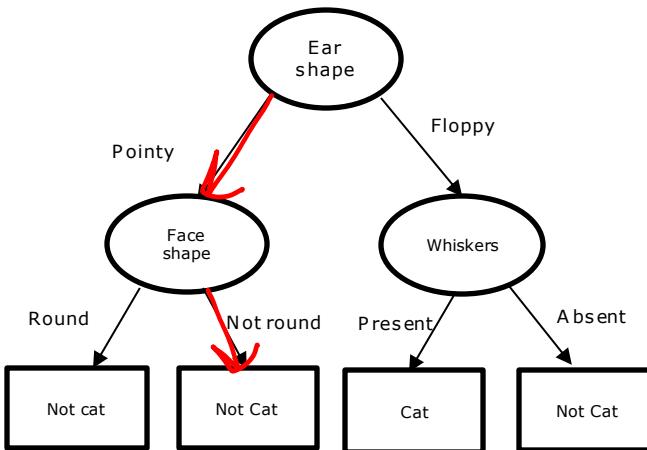
New test example



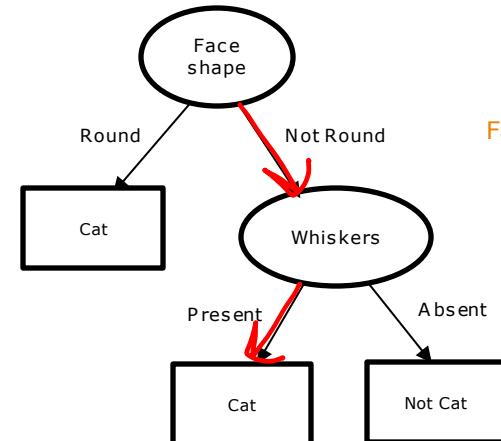
Ear shape: Pointy  
Face shape: Not Round  
Whiskers: Present



Prediction: Cat

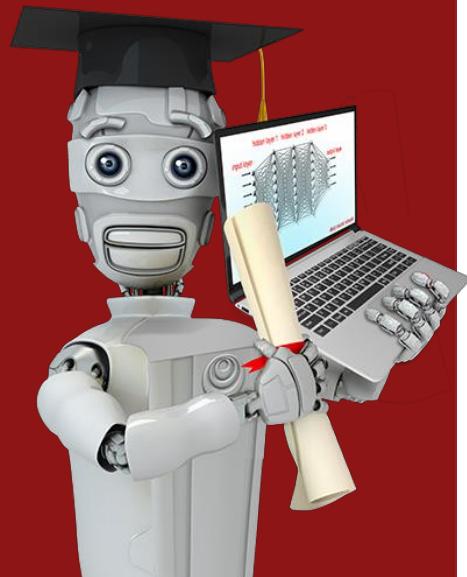


Prediction: Not cat



Prediction: Cat

Final prediction: Cat



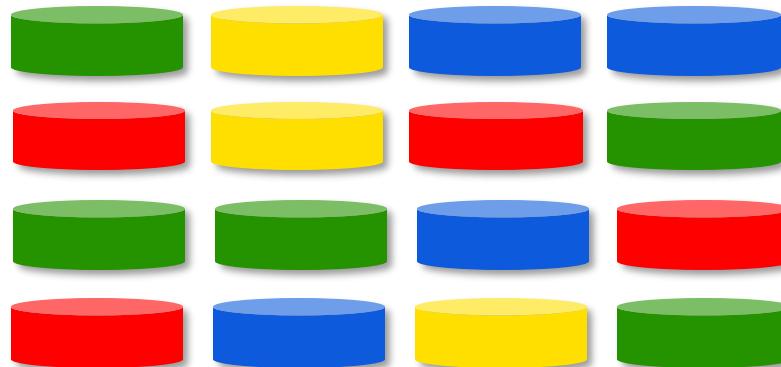
## Tree ensembles

### Sampling with replacement

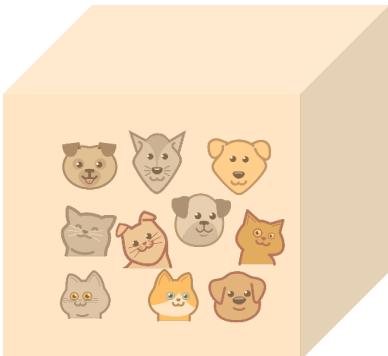
# Sampling with replacement



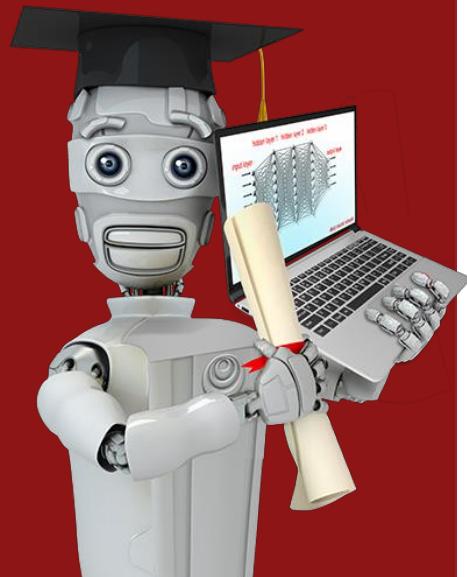
Sampling with replacement:



# Sampling with replacement



Ear shape	Face shape	Whiskers	Cat	
	Pointy	Round	Present	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Not round	Present	0
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Round	Absent	1



## Tree ensembles

# Random forest algorithm

# Generating a tree sample

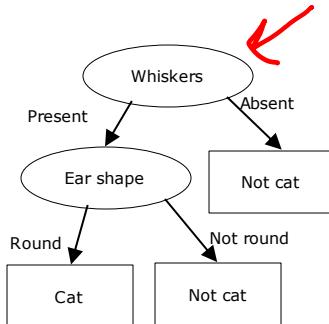
Given training set of size  $m$

For  $b = 1$  to  $B$

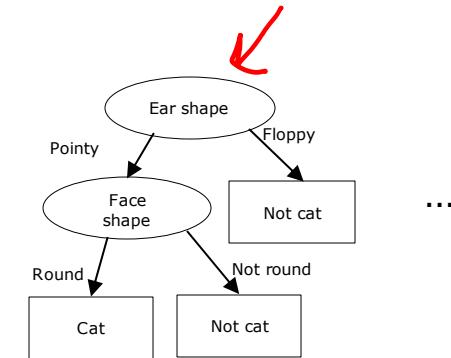
    Use sampling with replacement to create a new training set of size  $m$

    Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Pointy	Round	Absent	Yes
Floppy	Not Round	Absent	No
Floppy	Not Round	Absent	No
Pointy	Round	Absent	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



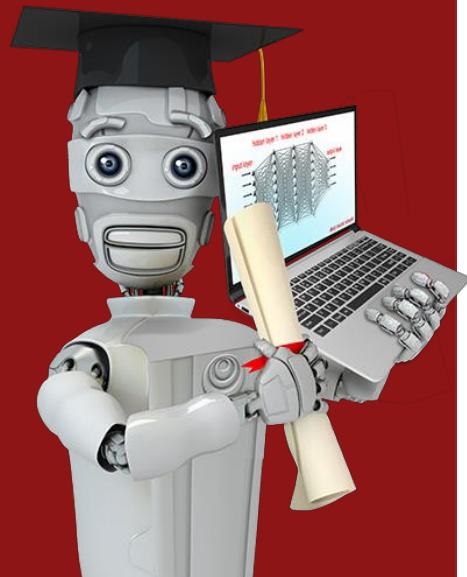
Bagged decision tree

# Randomizing the feature choice

At each node, when choosing a feature to use to split, if  $n$  features are available, pick a random subset of  $k < n$  features and allow the algorithm to only choose from that subset of features.

$$k = \sqrt{n}$$

Random forest algorithm



## Tree ensembles

# XGBoost

# Boosted trees intuition

Given training set of size  $m$

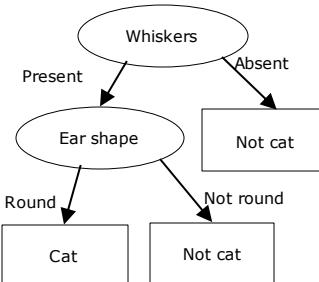
For  $b = 1$  to  $B$ :

Use sampling with replacement to create a new training set of size  $m$

But instead of picking from all examples with equal ( $1/m$ ) probability, make it more likely to pick examples that the previously trained trees misclassify

Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Prediction
Pointy	Round	Present	Cat ✓
Floppy	Not Round	Present	Not cat ✗
Floppy	Round	Absent	Not cat ✓
Pointy	Not Round	Present	Not cat ✓
Pointy	Round	Present	Cat ✓
Floppy	Round	Absent	Not cat ✗
Floppy	Not Round	Absent	Not cat ✓
Pointy	Round	Absent	Not cat ✗
Floppy	Round	Absent	Not cat ✓
Floppy	Not Round	Absent	Not cat ✗

1, 2, ..., b-1 b

# XGBoost (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

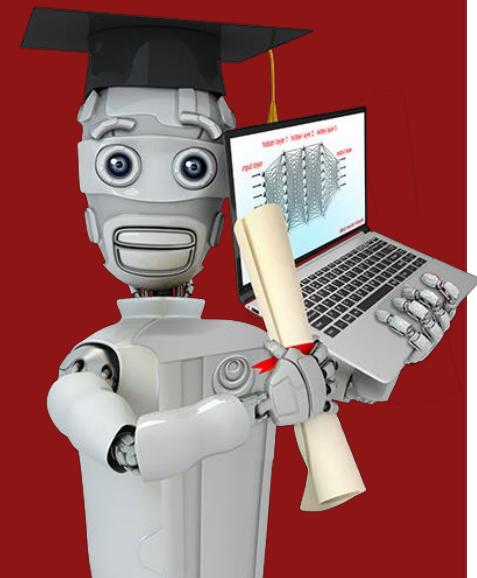
# Using XGBoost

## Classification

```
→from xgboost import XGBClassifier  
  
→model = XGBClassifier()  
  
→model.fit(X_train, y_train)  
→y_pred = model.predict(X_test)
```

## Regression

```
from xgboost import XGBRegressor  
  
model = XGBRegressor()  
  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)
```



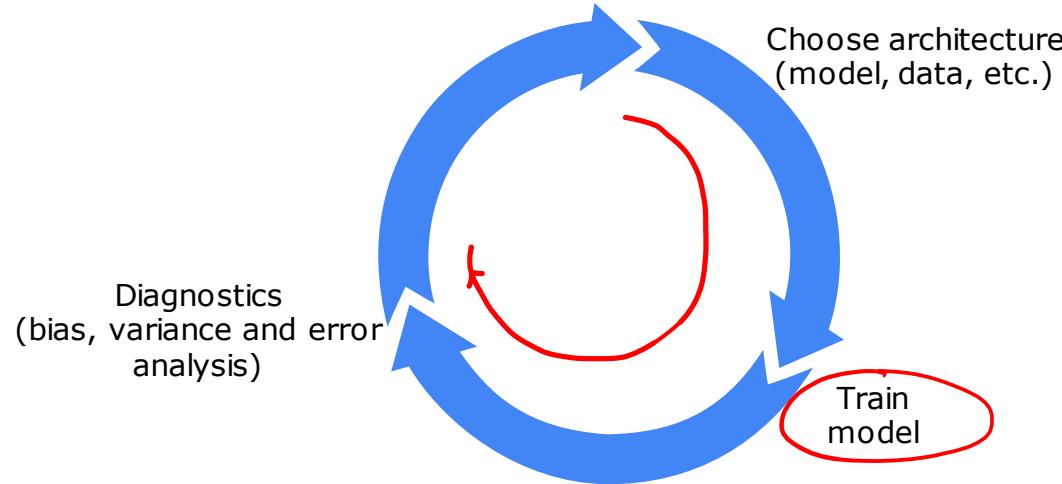
## Conclusion

**When to use decision trees**

# Decision Trees vs Neural Networks

## Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast



# Decision Trees vs Neural Networks

## Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human interpretable

## Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks