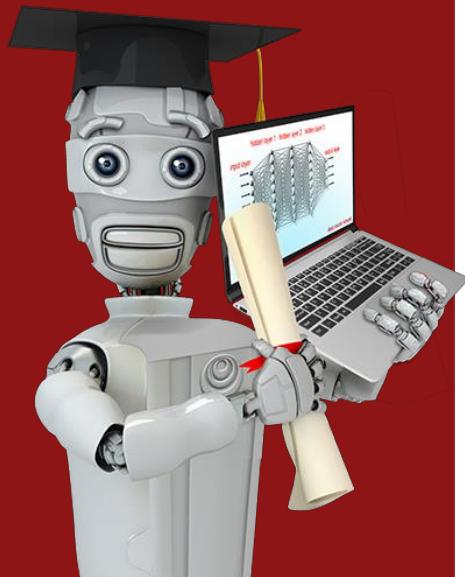




Stanford  
ONLINE

# Advanced Learning Algorithms



## Welcome!

# Advanced learning algorithms

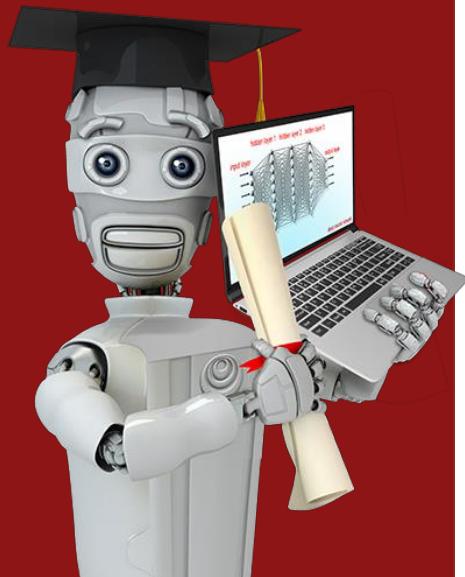
Neural Networks 

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.

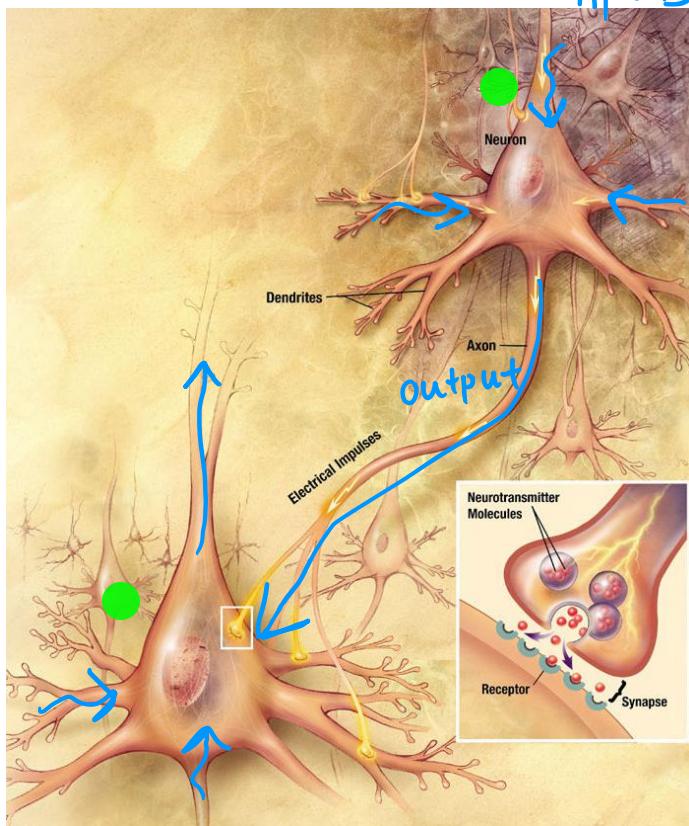


Used in the 1980's and early 1990's.  
Fell out of favor in the late 1990's.

Resurgence from around 2005.

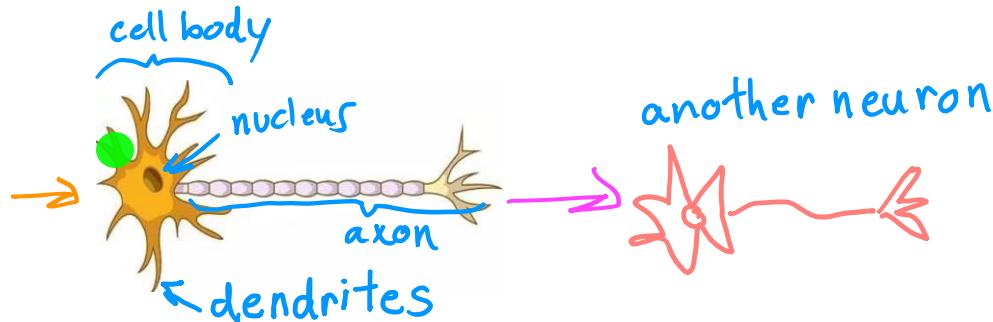
speech → images → text (NLP) → ...

# Neurons in the brain



## Biological neuron

inputs                    outputs



## Simplified mathematical model of a neuron

inputs                    outputs

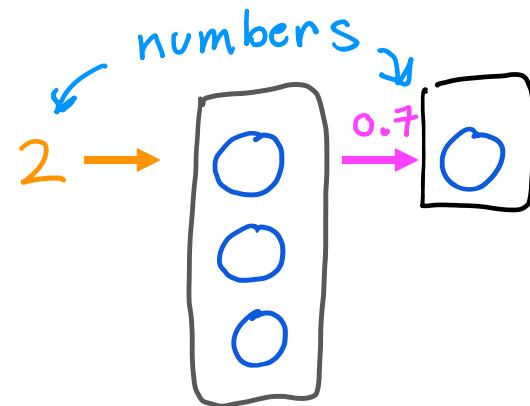
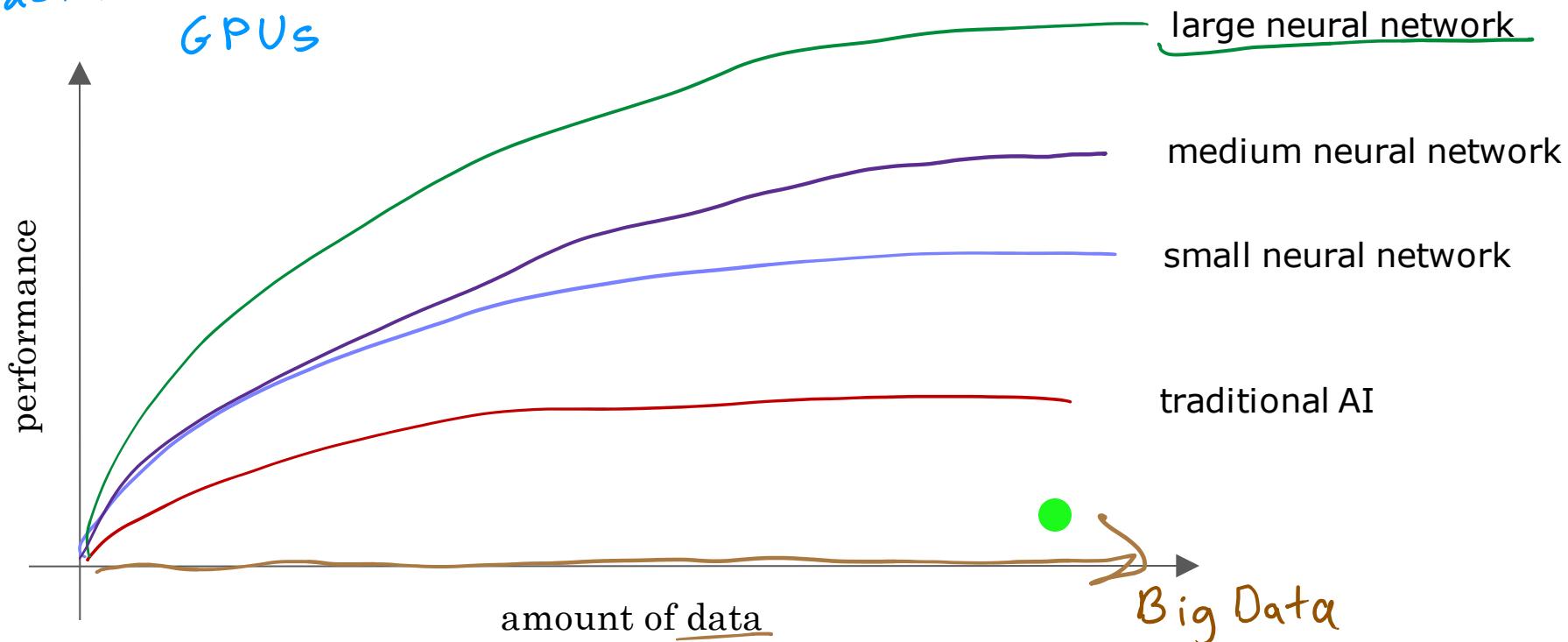
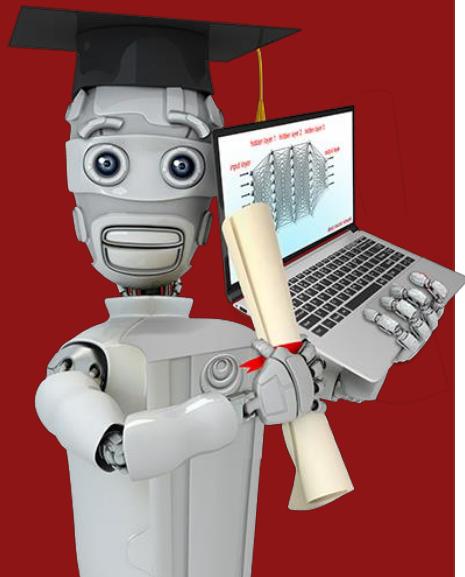


image source: <https://biologydictionary.net/sensory-neuron/>

Faster computer processors  
GPUs

# Why Now?

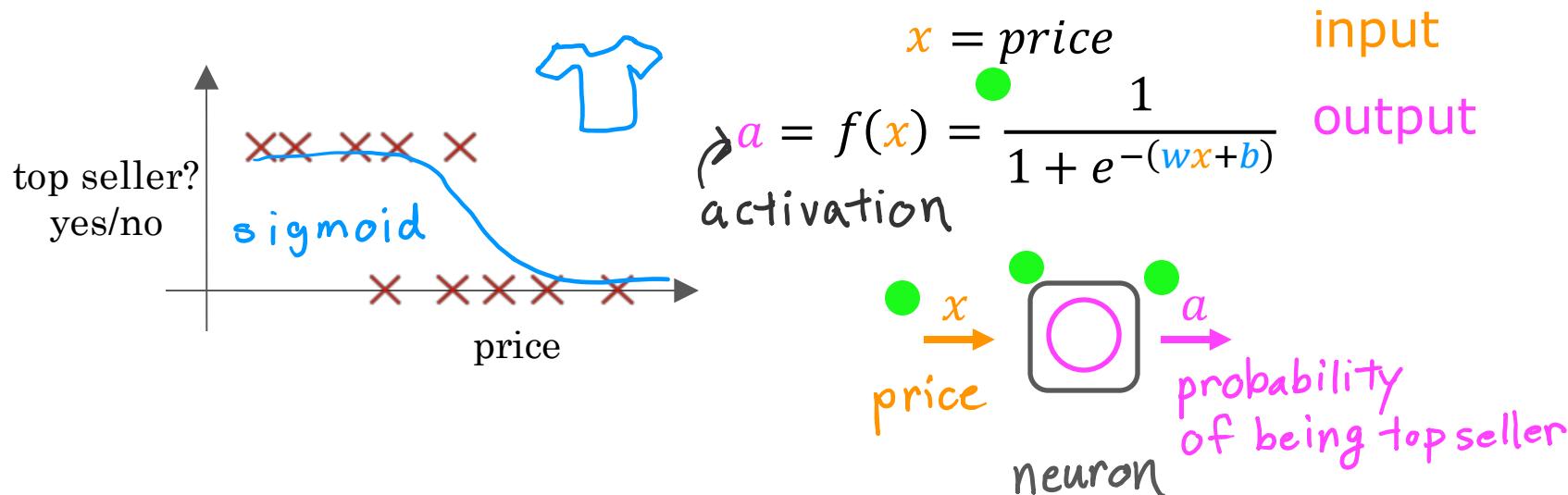




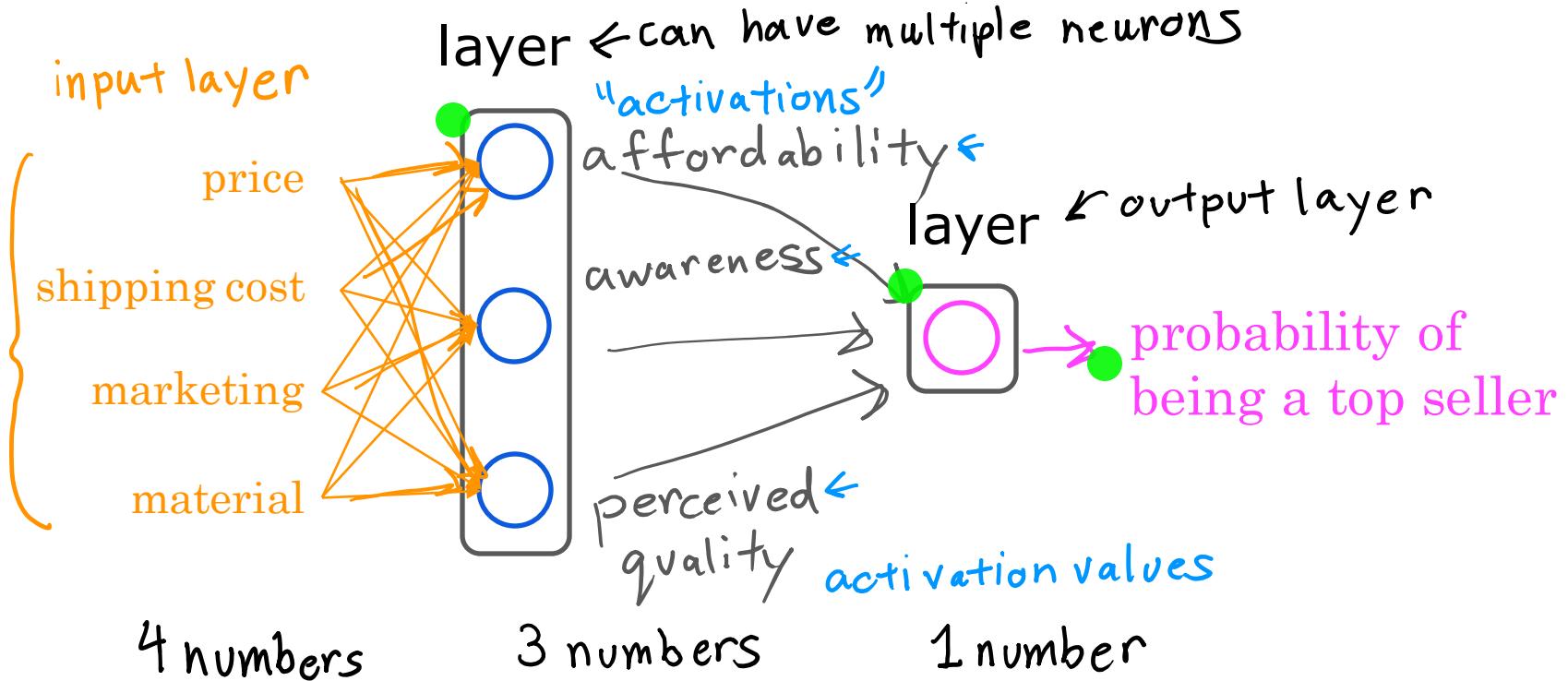
## Neural Network Intuition

# Demand Prediction

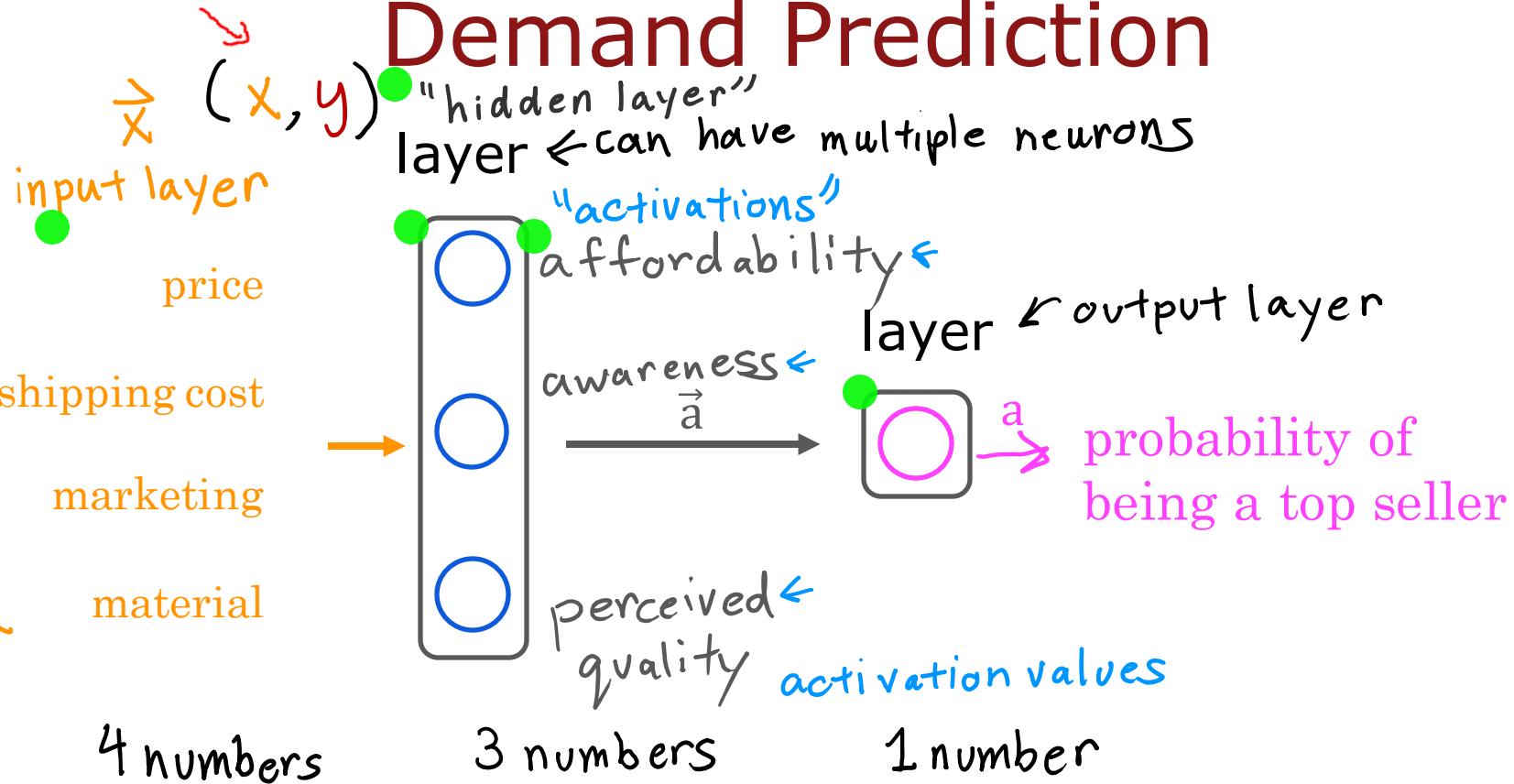
# Demand Prediction



# Demand Prediction



what we have in training set



# Demand Prediction

•  $\vec{x} \rightarrow (x, y)$

input layer

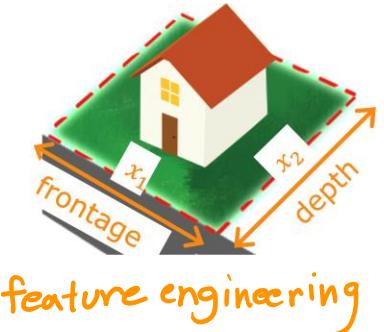
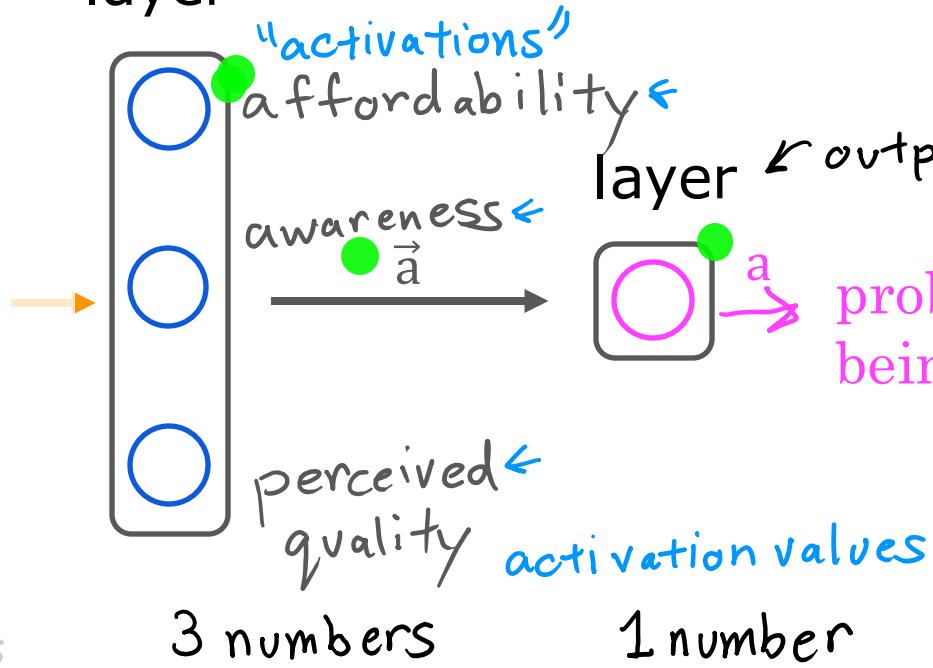
price

shipping cost

marketing

material

4 numbers



feature engineering

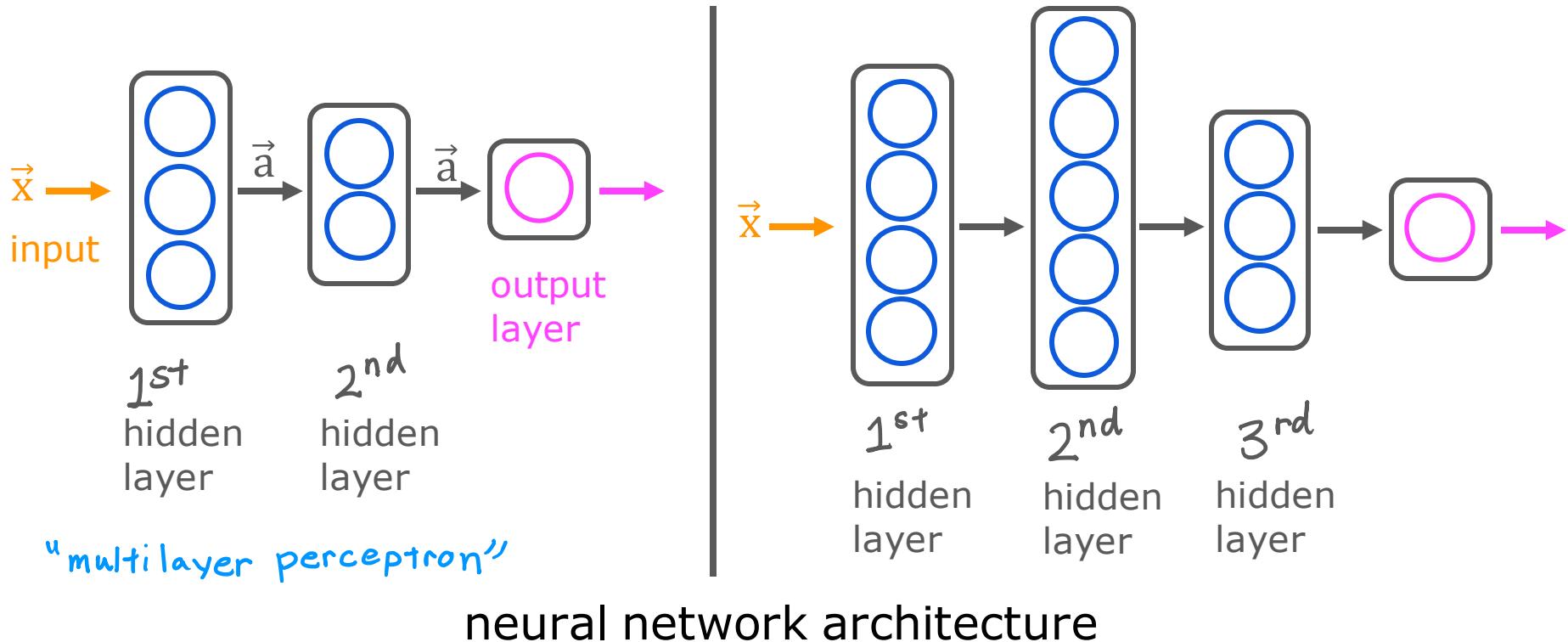
$x_1, x_2$

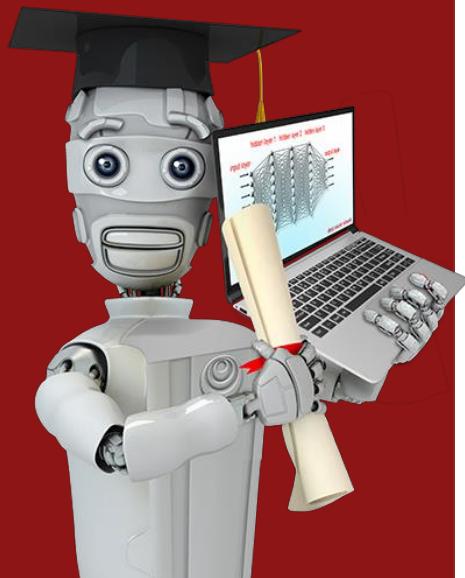
probability of  
being a top seller

neural network learn it's own  
features → affordability,  
awareness and perceived  
quality

Questions of How many hidden layers and how many neurons in each hidden layer do we want  $\rightarrow$  Question of architecture of neural networks

# Multiple hidden layers





# Neural Networks Intuition

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

# Face recognition

1000 pixels



1000 pixels

1000 rows

1000 columns

197	185	203	...	...
...	57	64	92	...
:				

pixel intensity values or pixel brightness values (0-255)

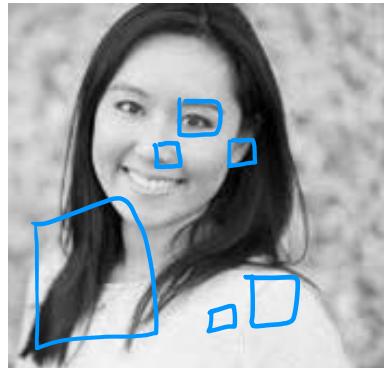
$\vec{x} =$

197
185
203
:
57
64
92
:
187
214

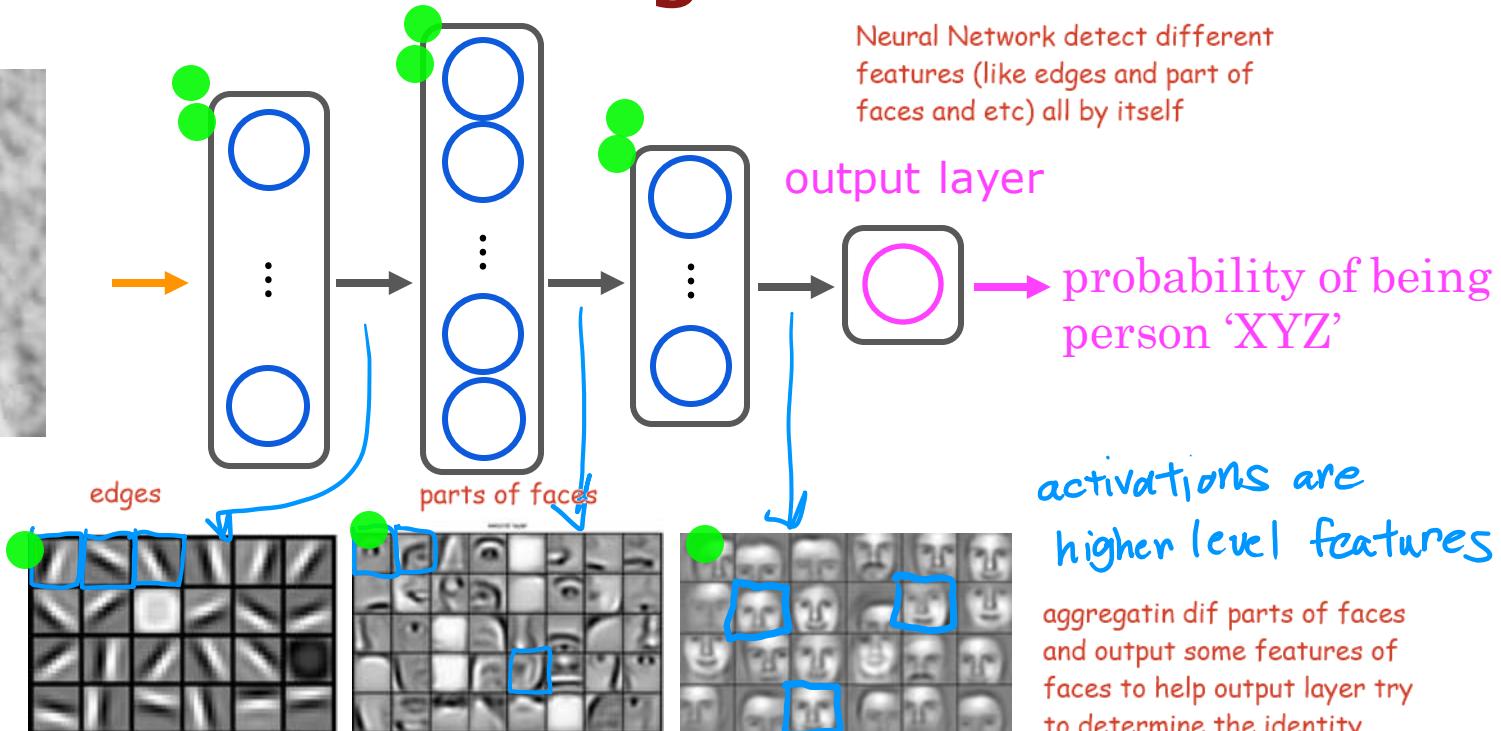
1 million

we give this vector to neural network

# Face recognition

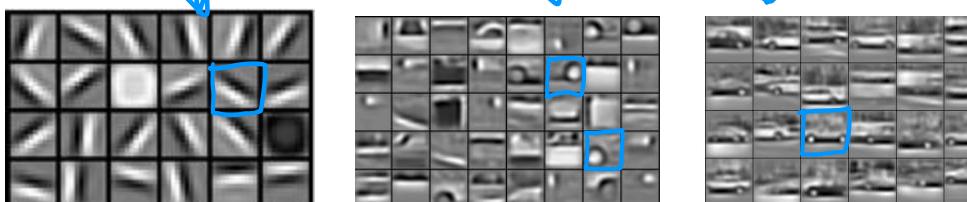
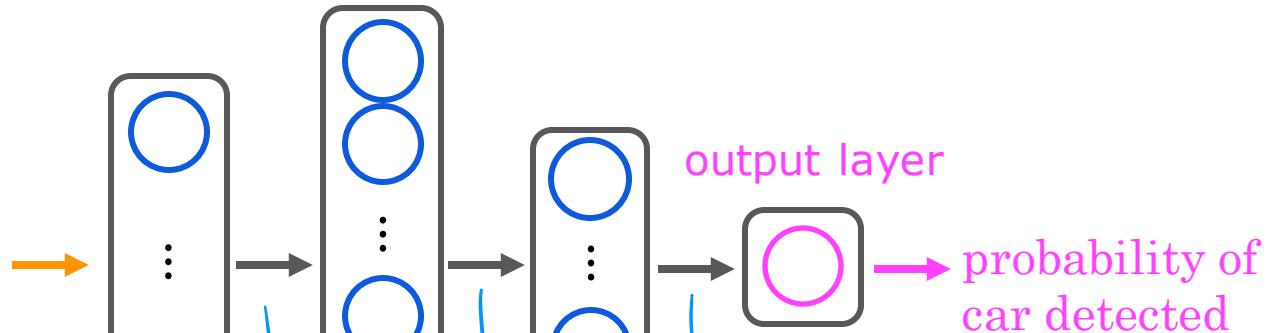


$\vec{x}$   
input



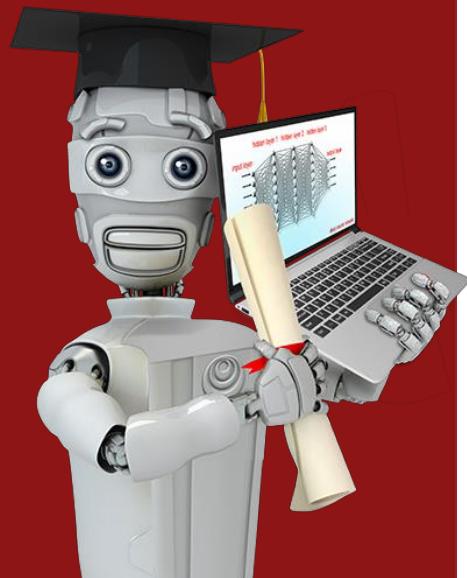
source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

# Car classification



activations are  
higher level features

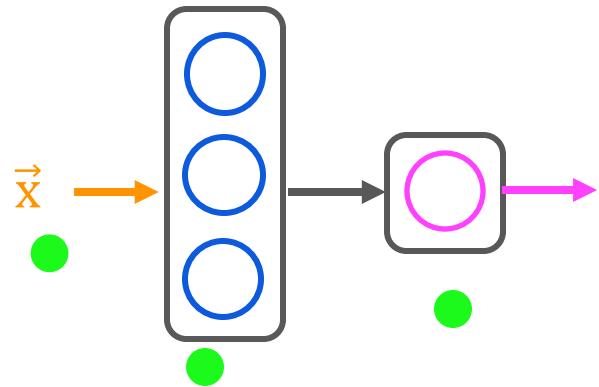
source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng



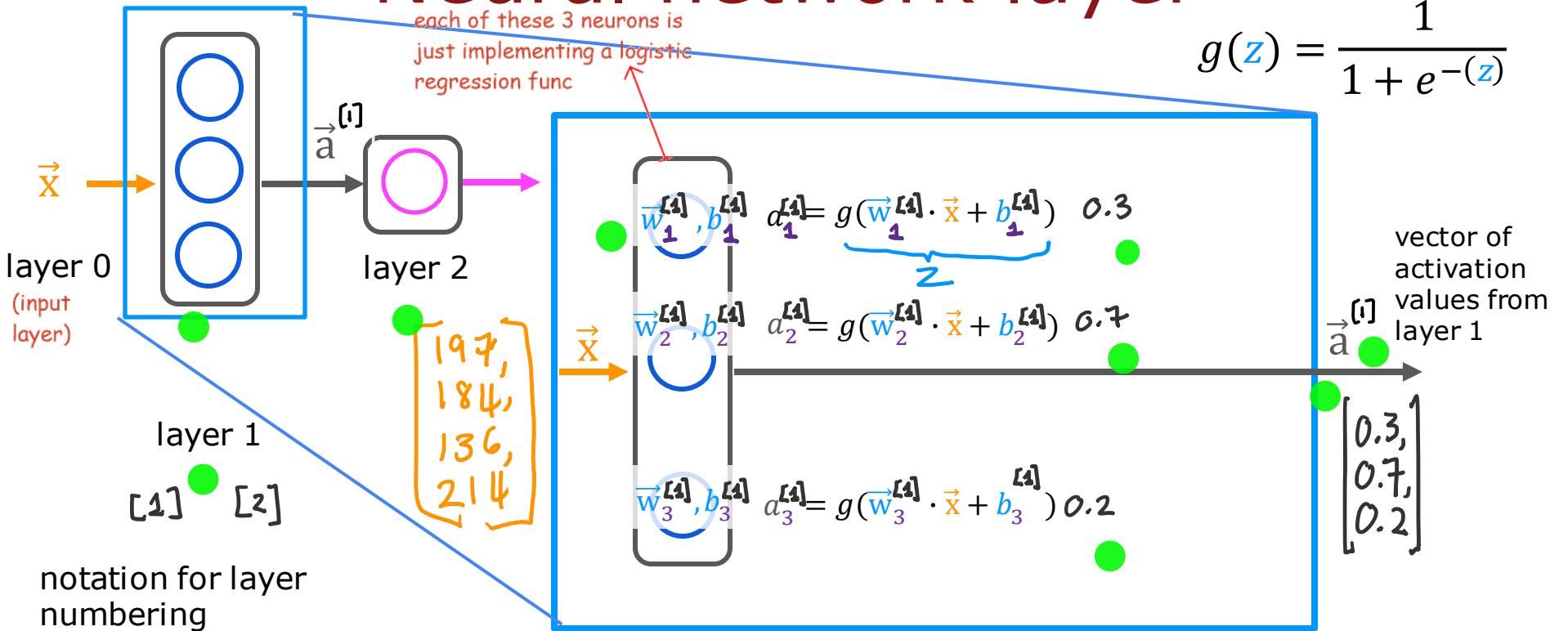
## Neural network model

## Neural network layer

# Neural network layer

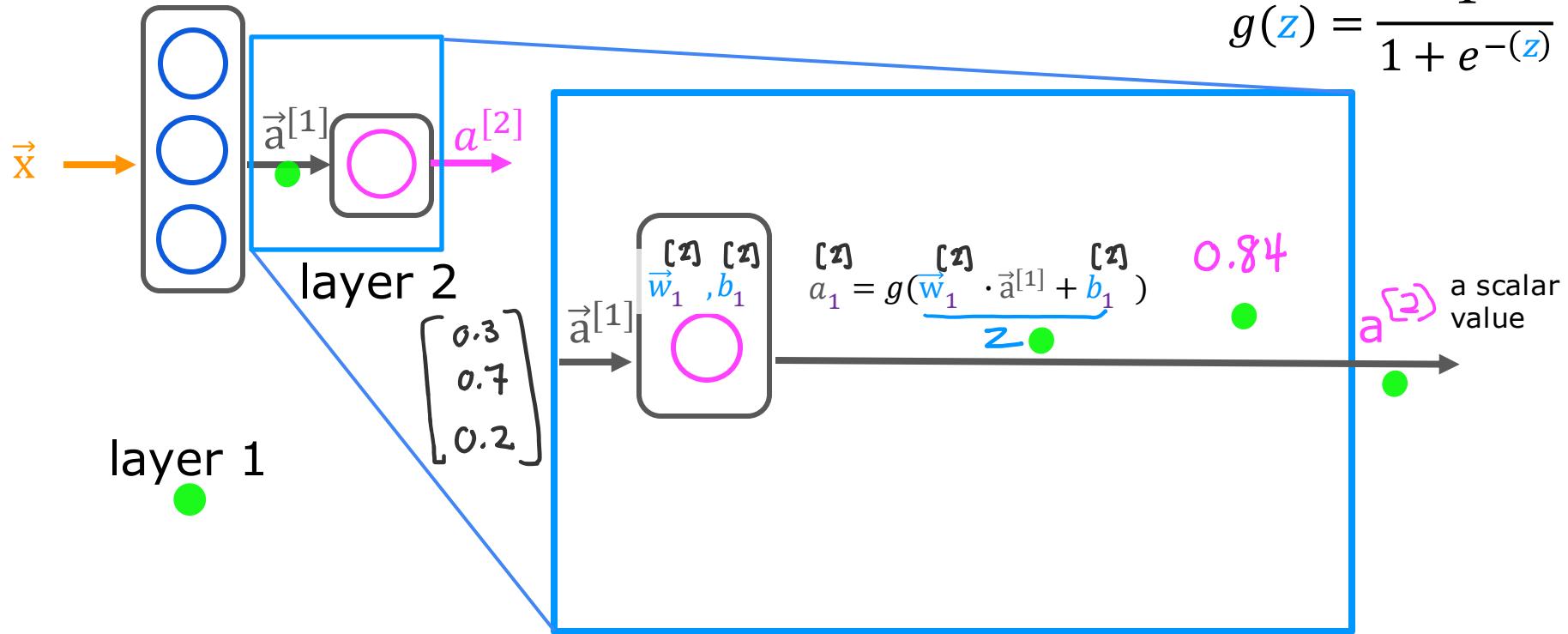


# Neural network layer

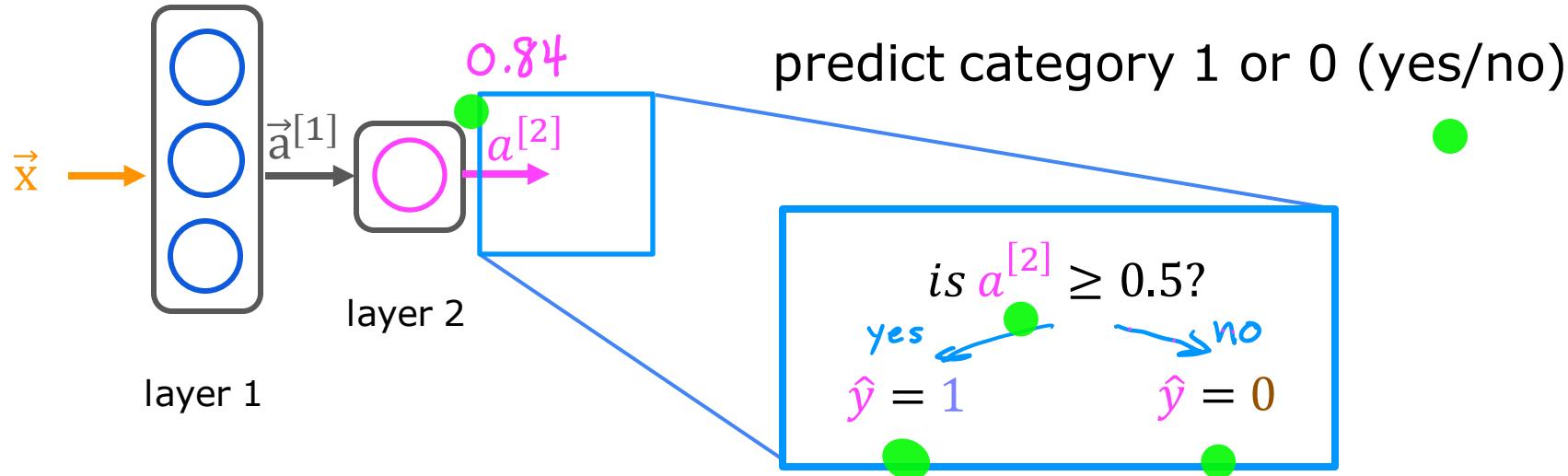


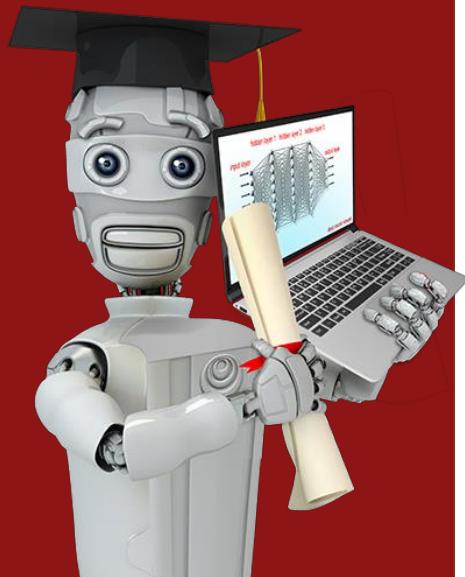
# Neural network layer

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



# Neural network layer

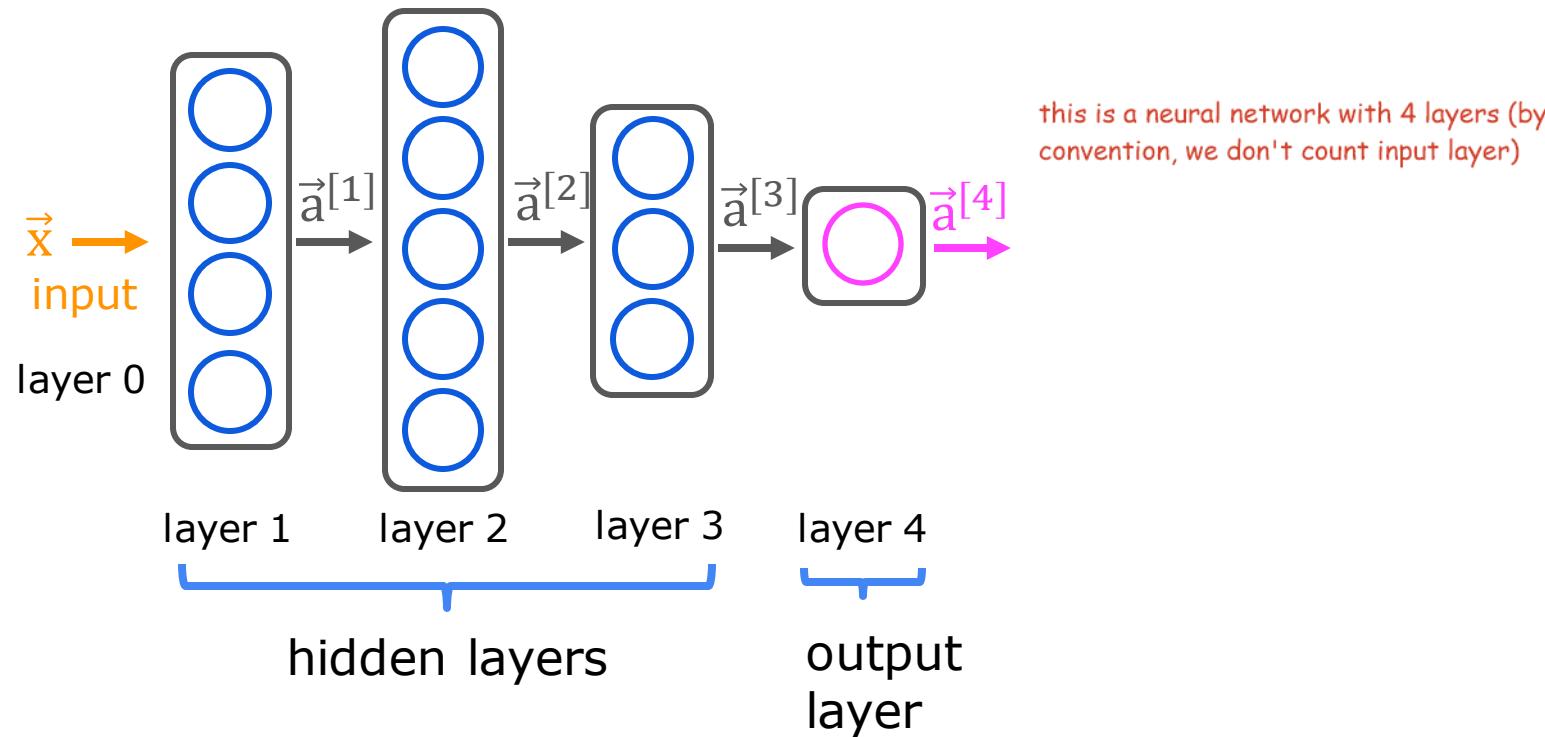




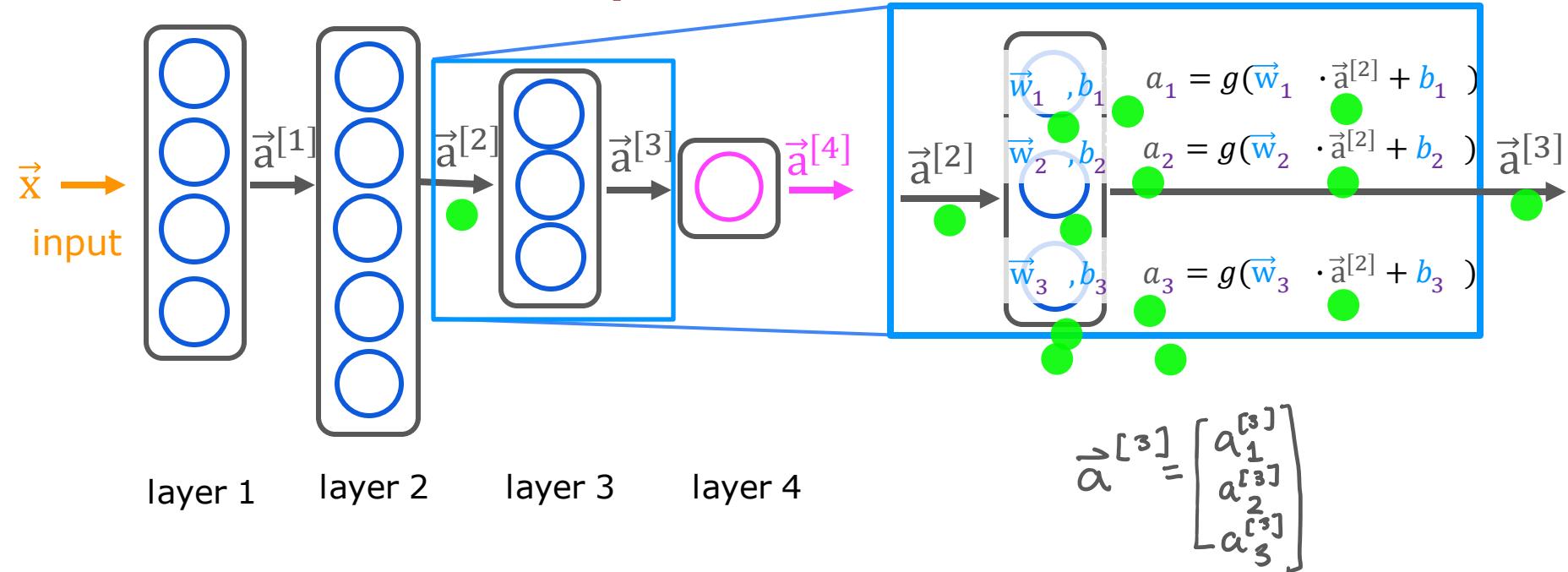
## Neural Network Model

More complex neural networks

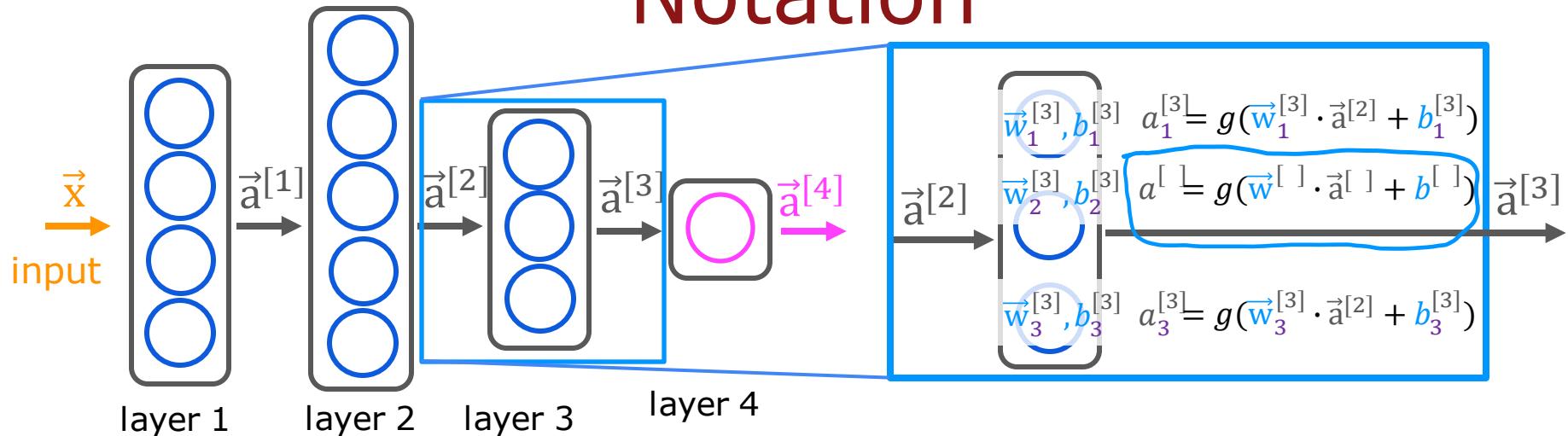
# More complex neural network



# More complex neural network



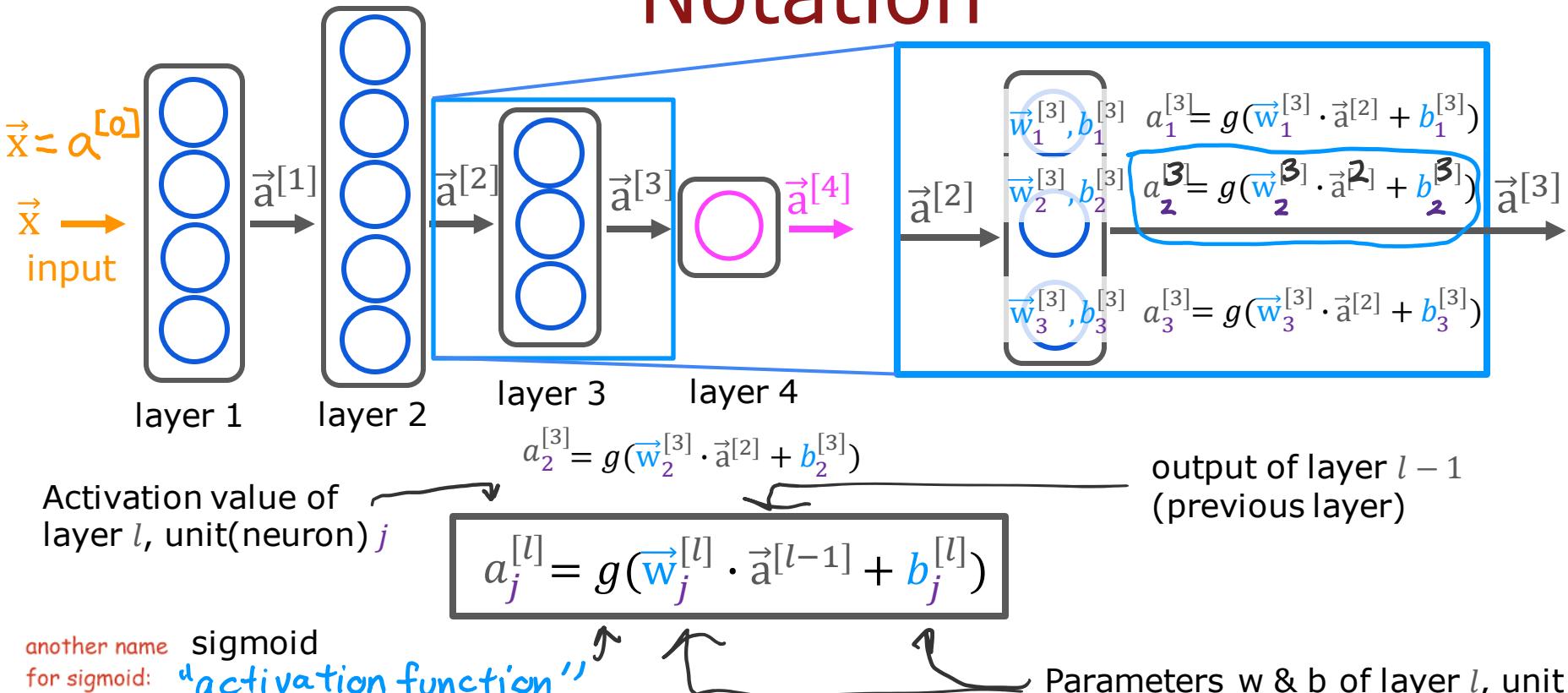
# Notation

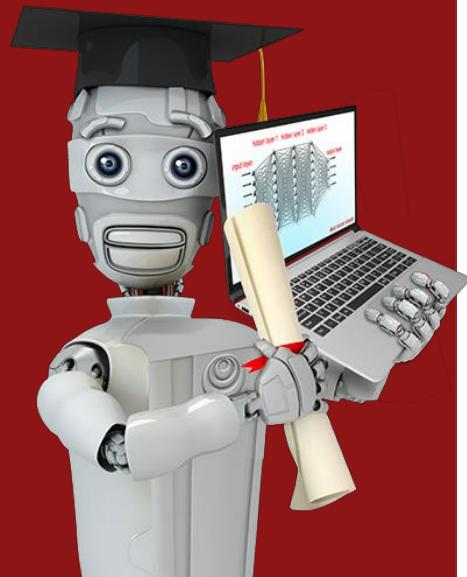


Question:  
Can you fill in the superscripts and  
subscripts for the second neuron?

activation func: func that would output activation values

# Notation

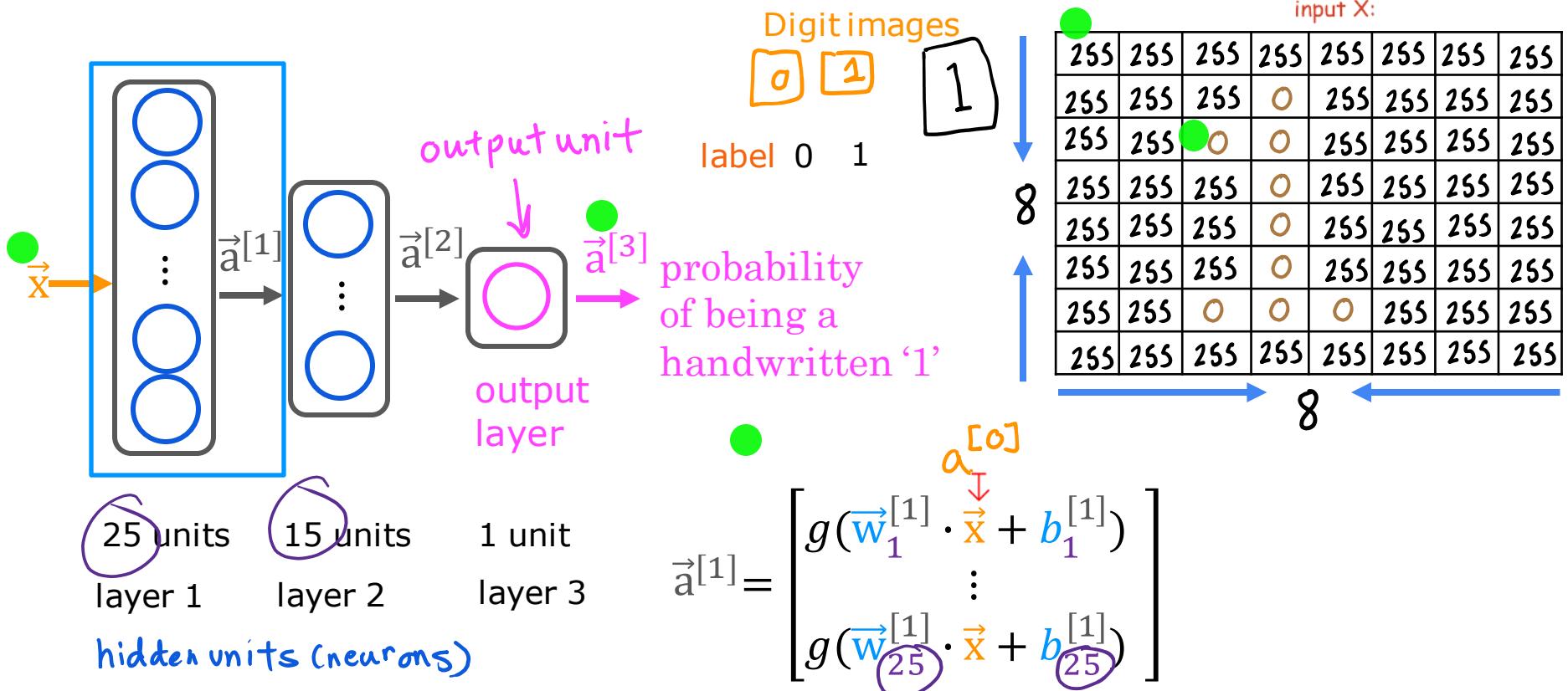




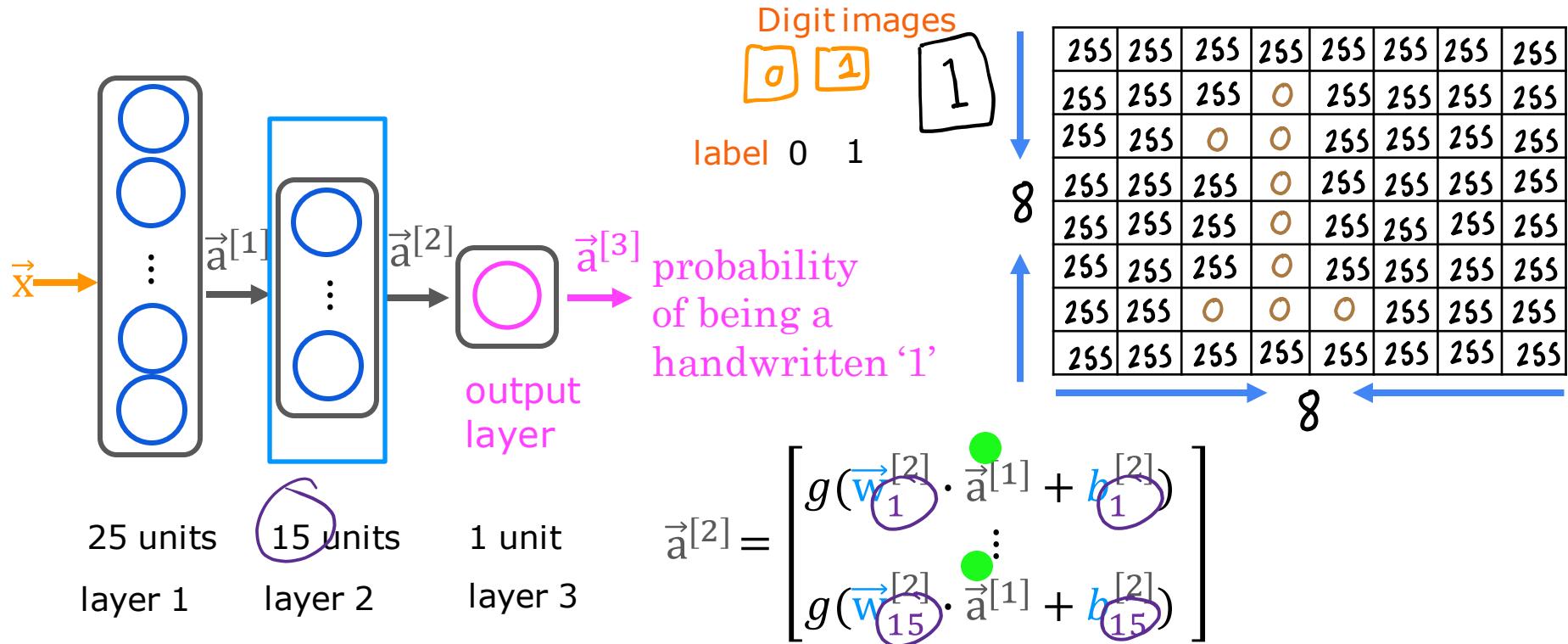
## Neural Network Model

Inference: making predictions  
(forward propagation)

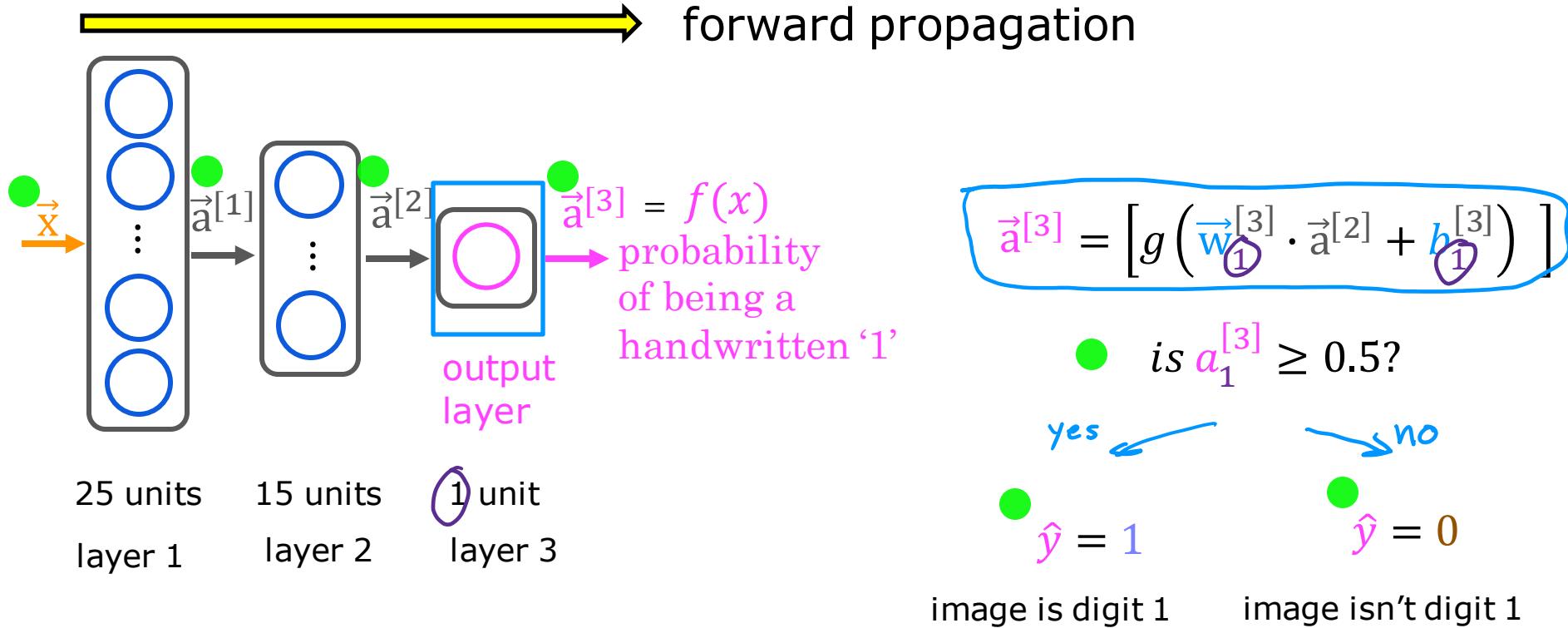
# Handwritten digit recognition

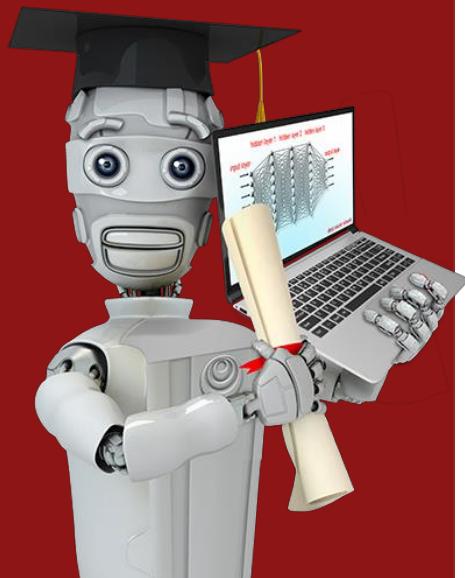


# Handwritten digit recognition



# Handwritten digit recognition

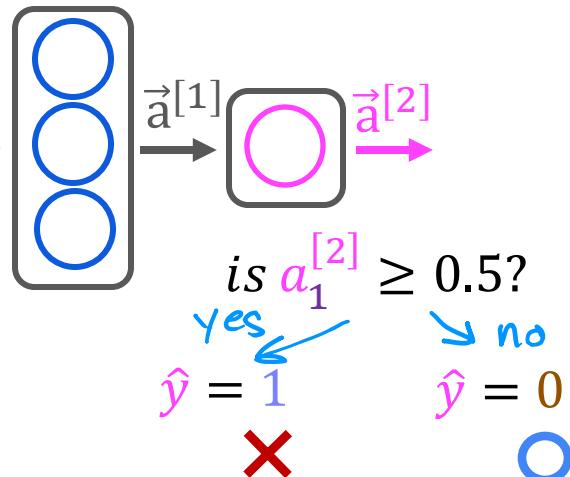
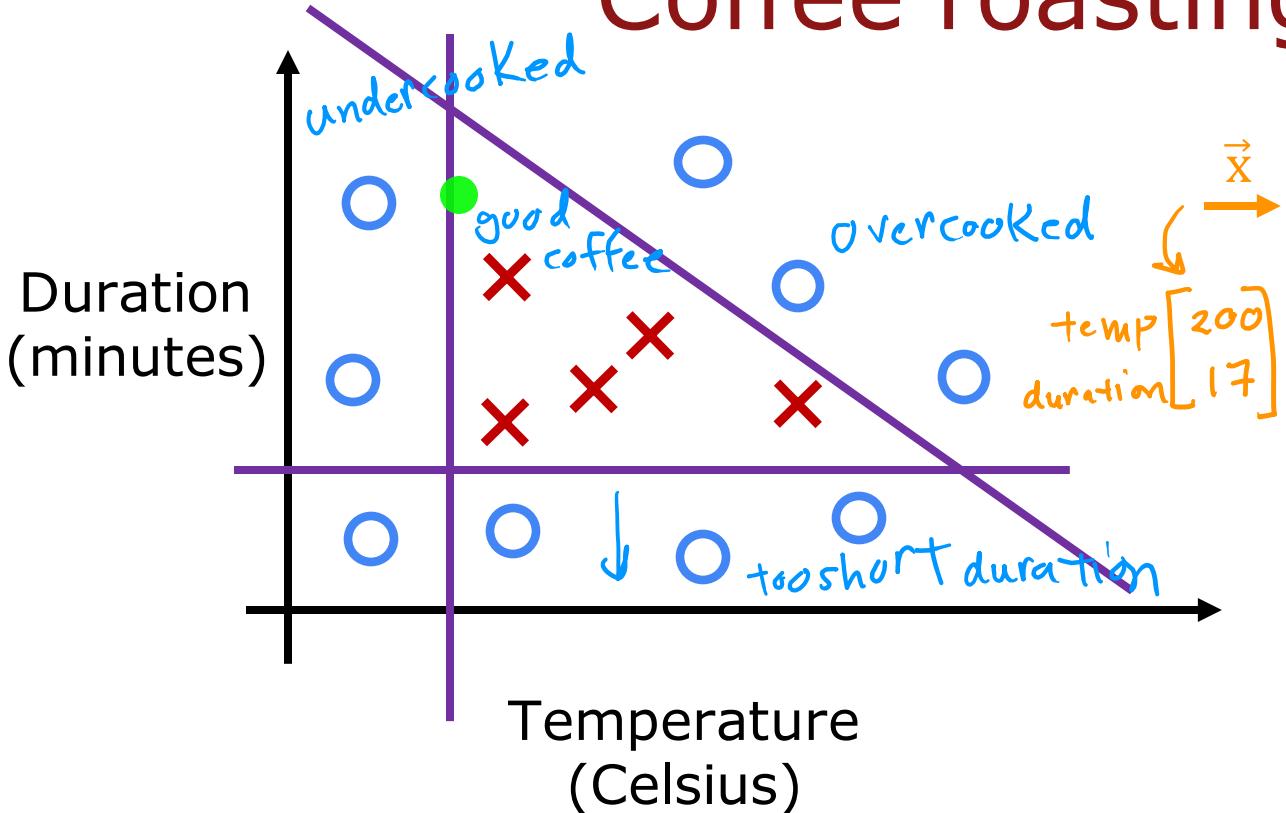




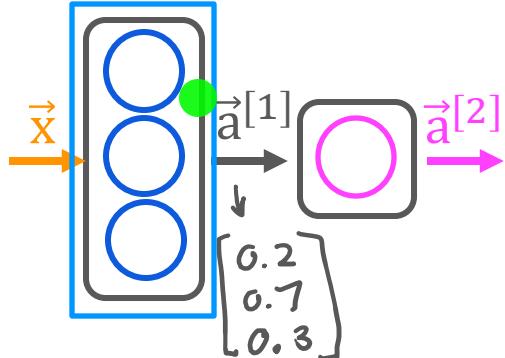
## TensorFlow implementation

## Inference in Code

# Coffee roasting

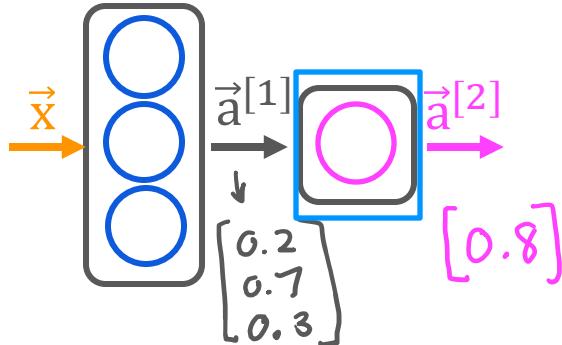


Another name for layers in neural network (that we learned so far)



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

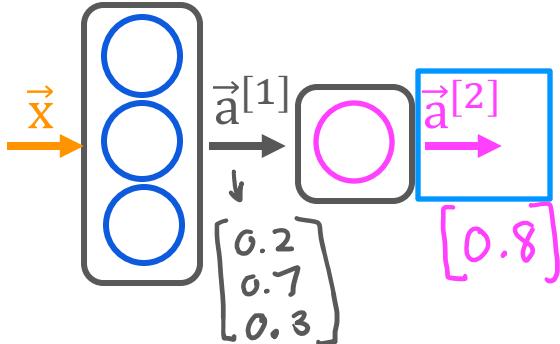
# Build the model using TensorFlow



```
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



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

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

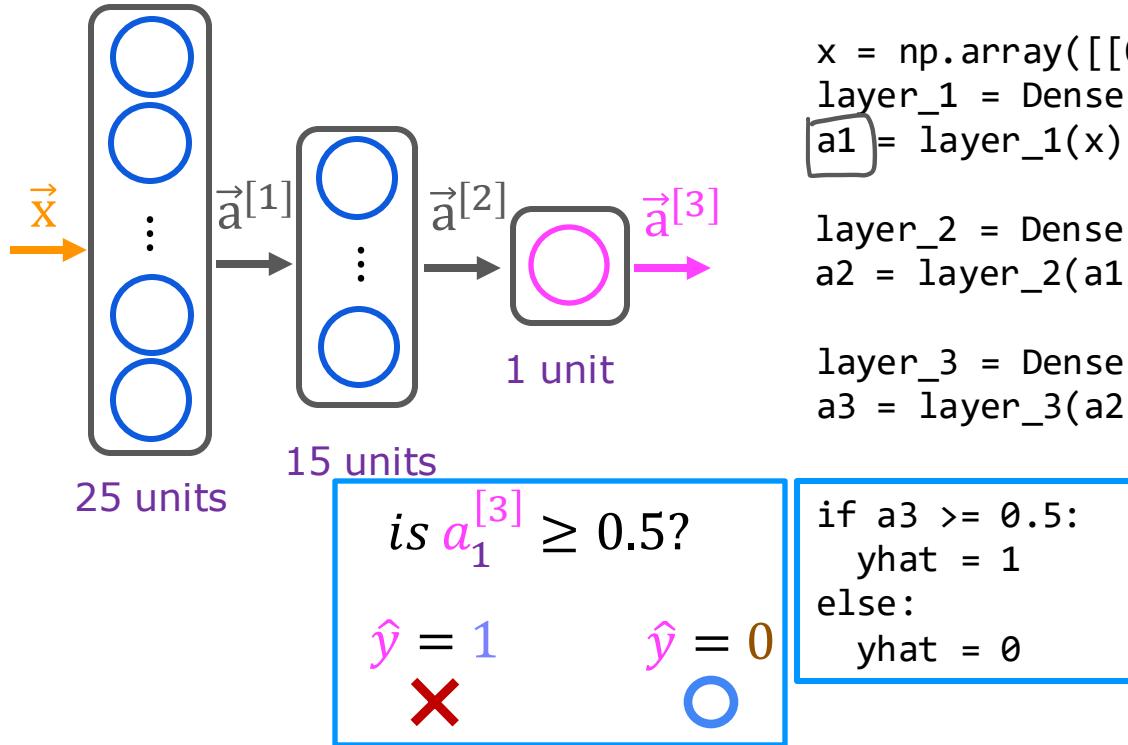
$\times$   $\circ$

```
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)
```

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

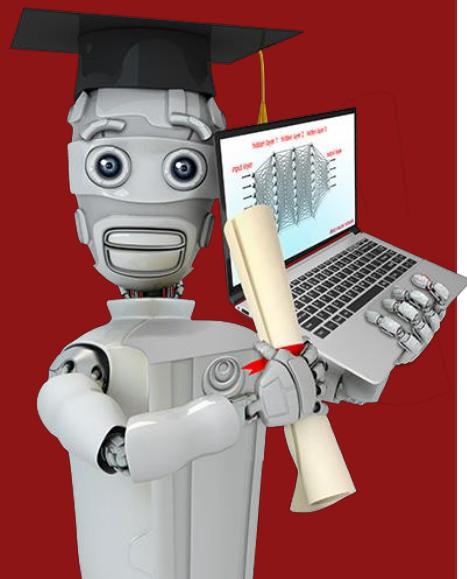
# Model for digit classification



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

```
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

3 columns  
2 rows  
[1 2 3]  
[4 5 6]  
 $2 \times 3$  matrix

4 rows  
2 columns  
[0.1 0.2]  
[-3 -4]  
[-.5 -.6]  
[7 8]  
 $4 \times 2$  matrix

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

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

[[0.1, 0.2],  
[-3.0, -4.0],  
[-0.5, -0.6],  
[7.0, 8.0]]]  
 $1 \times 2$

[[0.1, 0.2],  
[-3.0, -4.0],  
[-0.5, -0.6],  
[7.0, 8.0]]]  
 $2 \times 1$

# Note about numpy arrays

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

[200 17]

1 x 2

row vector

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

[200]  
17

2 x 1

\* we need to use 2D matrices  
for TensorFlow

column vector

```
→ 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
...	...	...

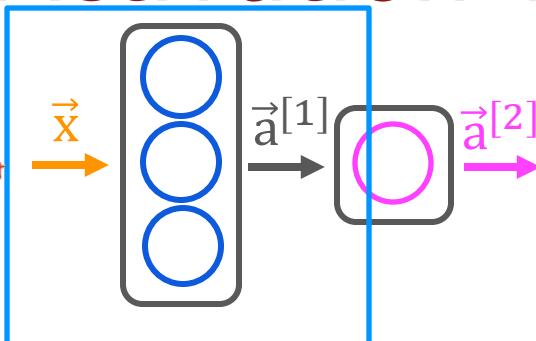
`x = np.array([[200.0, 17.0]])` ←

`[[200.0, 17.0]]`

↓      ↓      1 x 2  
→ [200.0    17.0]

# Activation vector

Think of tensor as matrix (but it's actually more general than matrix)



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

$\rightarrow$  [0.2, 0.7, 0.3]    1 x 3 matrix

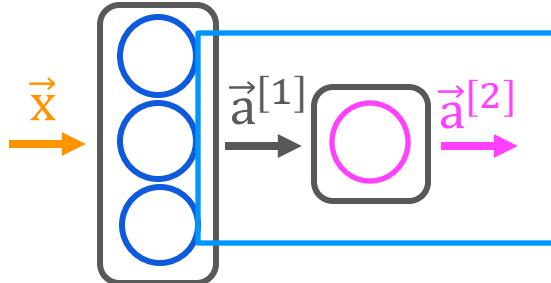
(32 bit of memory  $\rightarrow$  float32)

$\rightarrow$  tf.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)

$\rightarrow$  a1.numpy()    convert tensor to a numpy array

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

# Activation vector



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

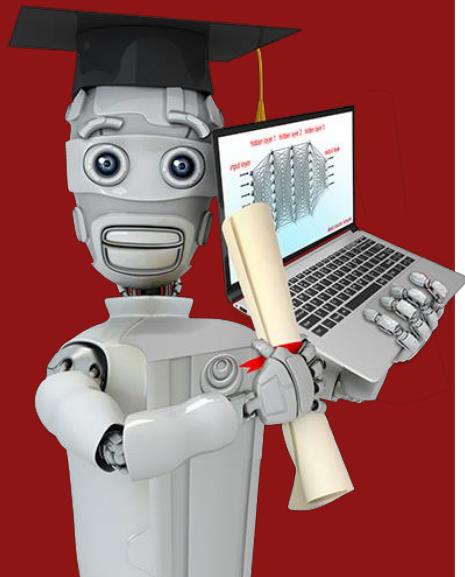
[[0.8]] ←

1 × 1

```
→ tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
→ a2.numpy()
```

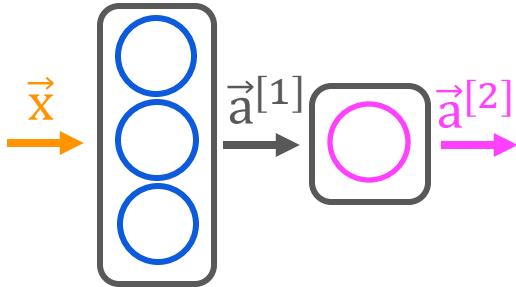
```
→ array([[0.8]], dtype=float32)
```



## TensorFlow implementation

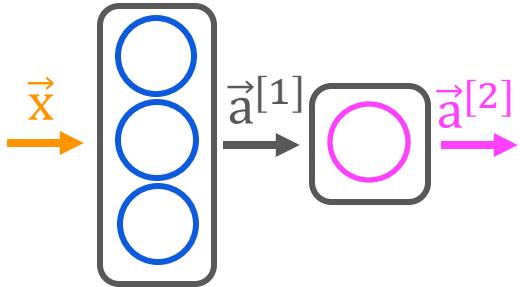
### Building a neural network

# What you saw earlier



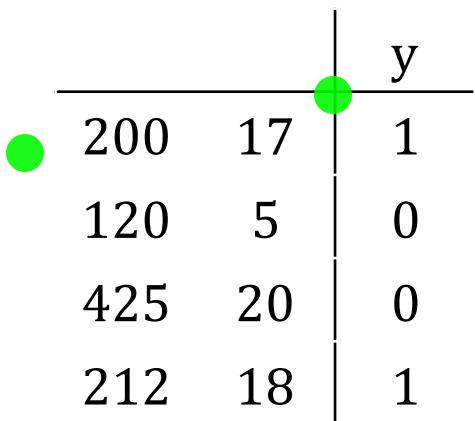
```
→ x = np.array([[200.0, 17.0]])  
→ layer_1 = Dense(units=3, activation="sigmoid")  
→ a1 = layer_1(x)  
  
→ layer_2 = Dense(units=1, activation="sigmoid")  
→ a2 = layer_2(a1)
```

# Building a neural network architecture



Would create a NN with layer\_1 and layer\_2

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



$x = \text{np.array}([[200.0, 17.0], [120.0, 5.0], [425.0, 20.0], [212.0, 18.0]])$

$4 \times 2$

$y = \text{np.array}([1,0,0,1])$

model.compile(...)

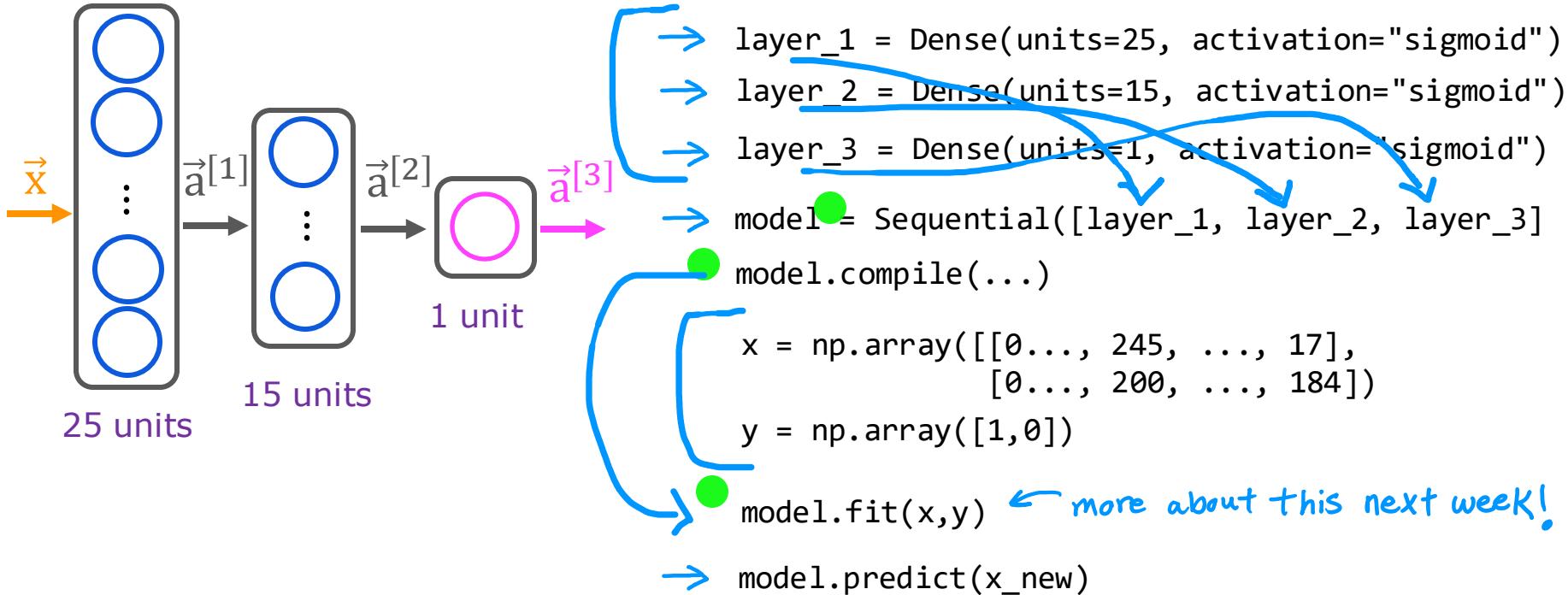
model.fit(x,y)

train on data x and y

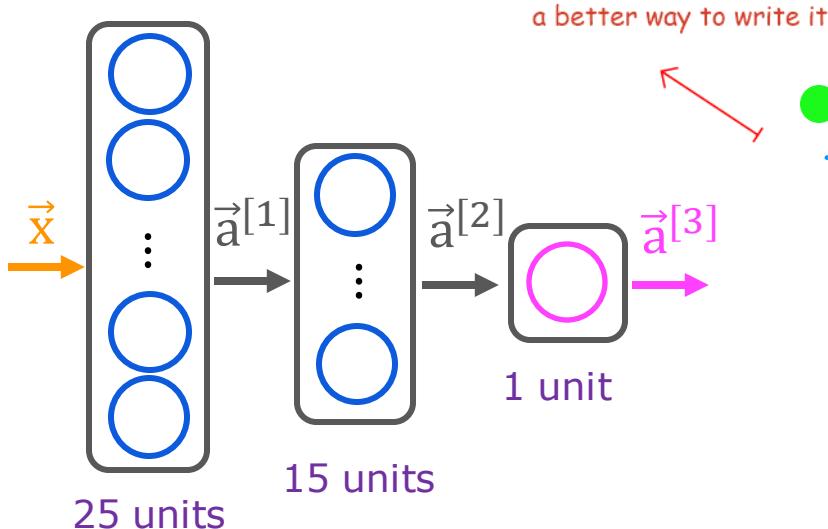
→ model.predict(x\_new)

more about this next week!

# Digit classification model



# Digit classification model



a better way to write it

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

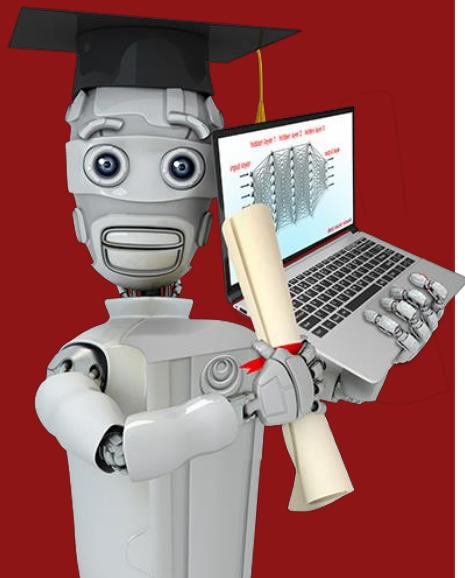
```
model.compile(...)
```

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

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

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

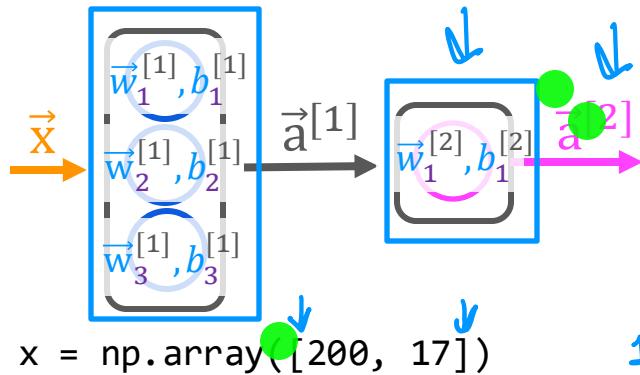
```
model.predict(x_new)
```



# Neural network implementation in Python

## Forward prop in a single layer

# forward prop (coffee roasting model)



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

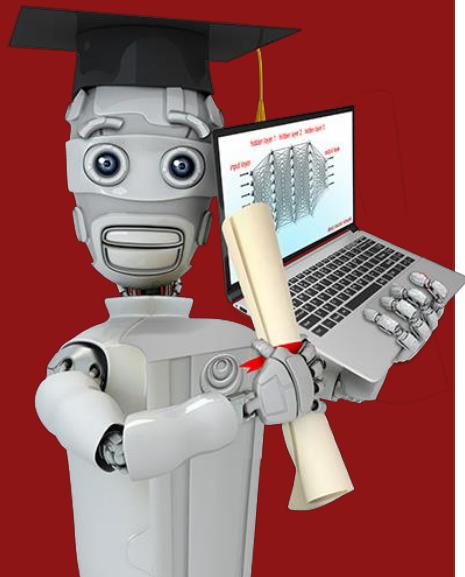
$b2\_1 = \text{np.array([3])}$

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

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

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

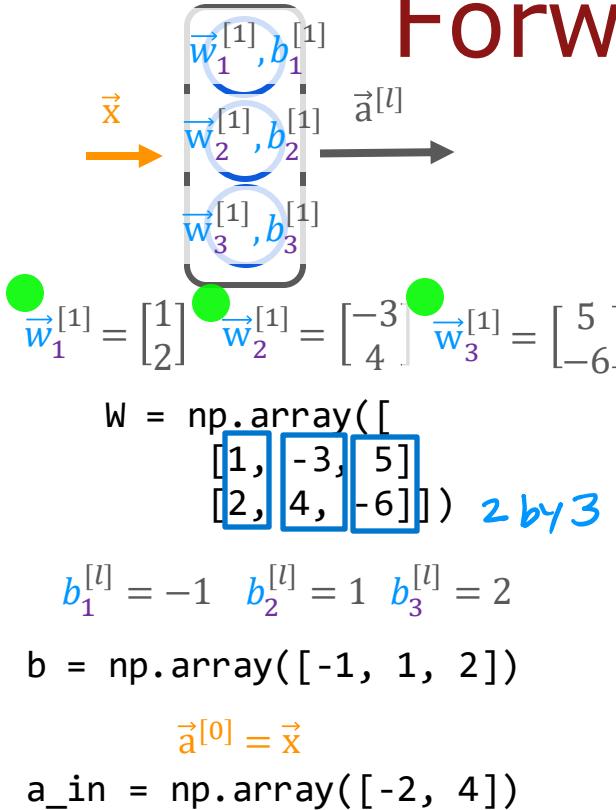
1D arrays



# Neural network implementation in Python

General implementation of  
forward propagation

# Forward prop in NumPy

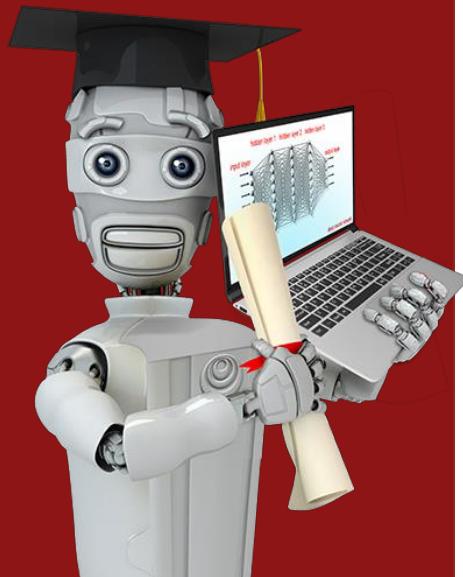


```
def dense(a_in,W,b, g):
    units = W.shape[1] [0,0,0]
    a_out = np.zeros(units)
    for j in range(units): 0,1,2
        w = W[:,j] pull out jth column of a matrix
        z = np.dot(w,a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

```
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
```

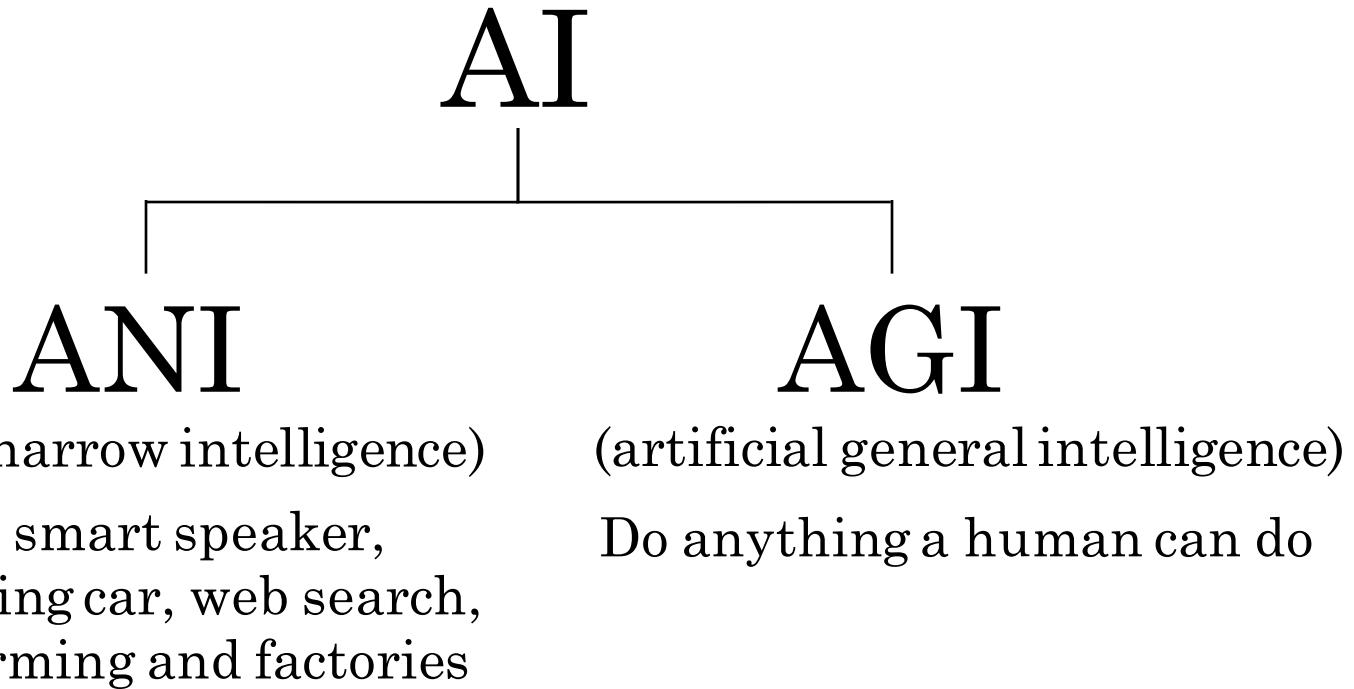
capital W refers to a matrix

Upper case  $\rightarrow$  Matrix  
 Lower case  $\rightarrow$  Vectors & Scalars



# Speculations on artificial general intelligence (AGI)

Is there a path to AGI?



## Biological neuron

inputs                    outputs

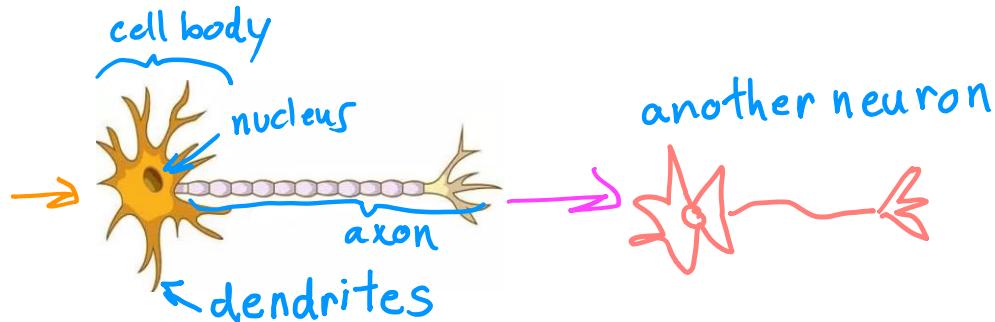
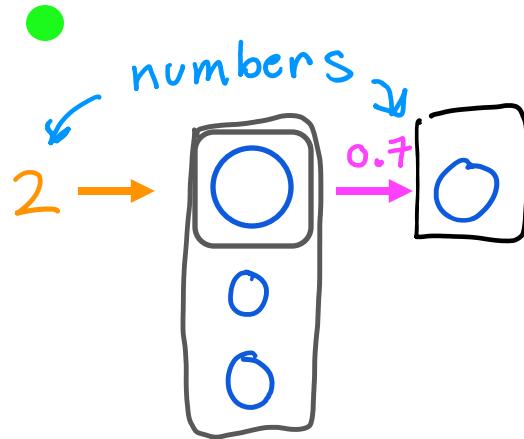


image source: <https://biologydictionary.net/sensory-neuron/>

## Simplified mathematical model of a neuron

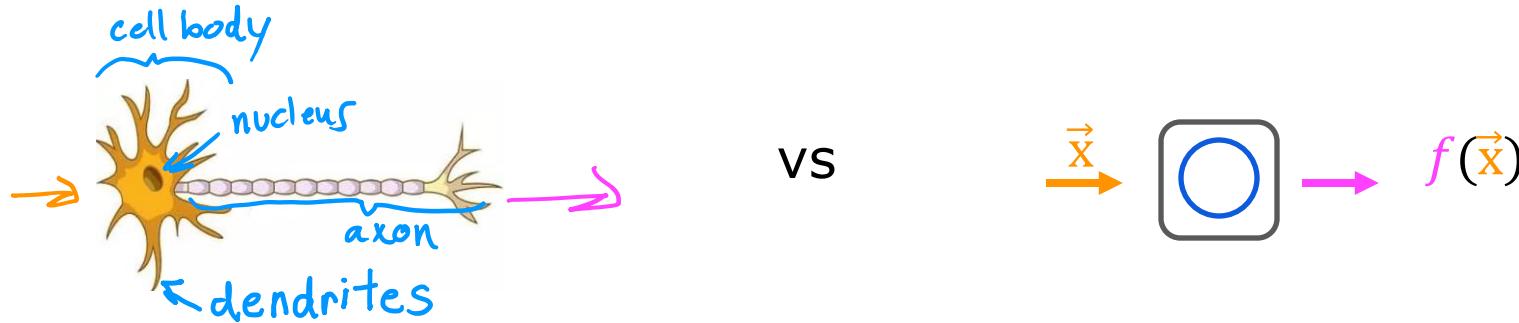
inputs                    outputs



Artificial Neurons and NNs  
are so simple and they're like  
nothing compared to Biological  
Neurons

# Neural network and the brain

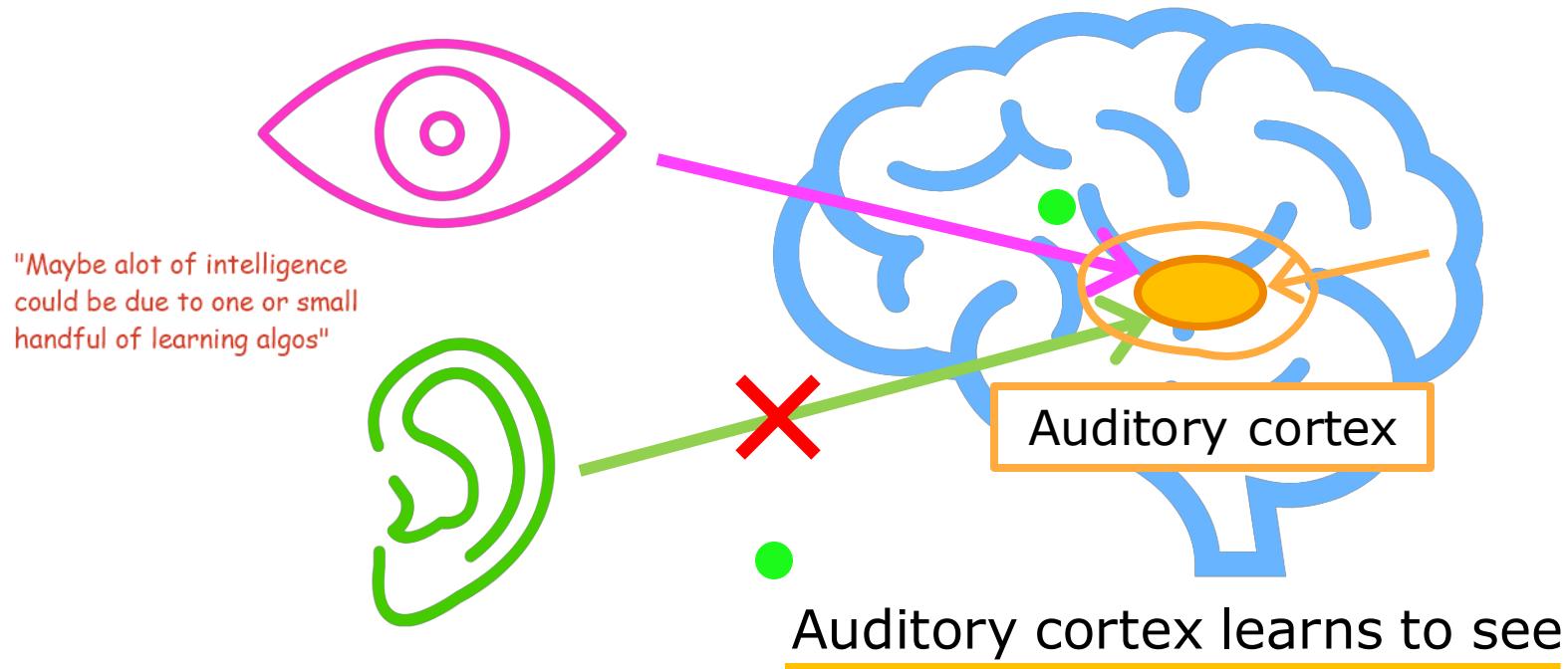
Can we mimic the human brain?



We have (almost) no idea how the brain works

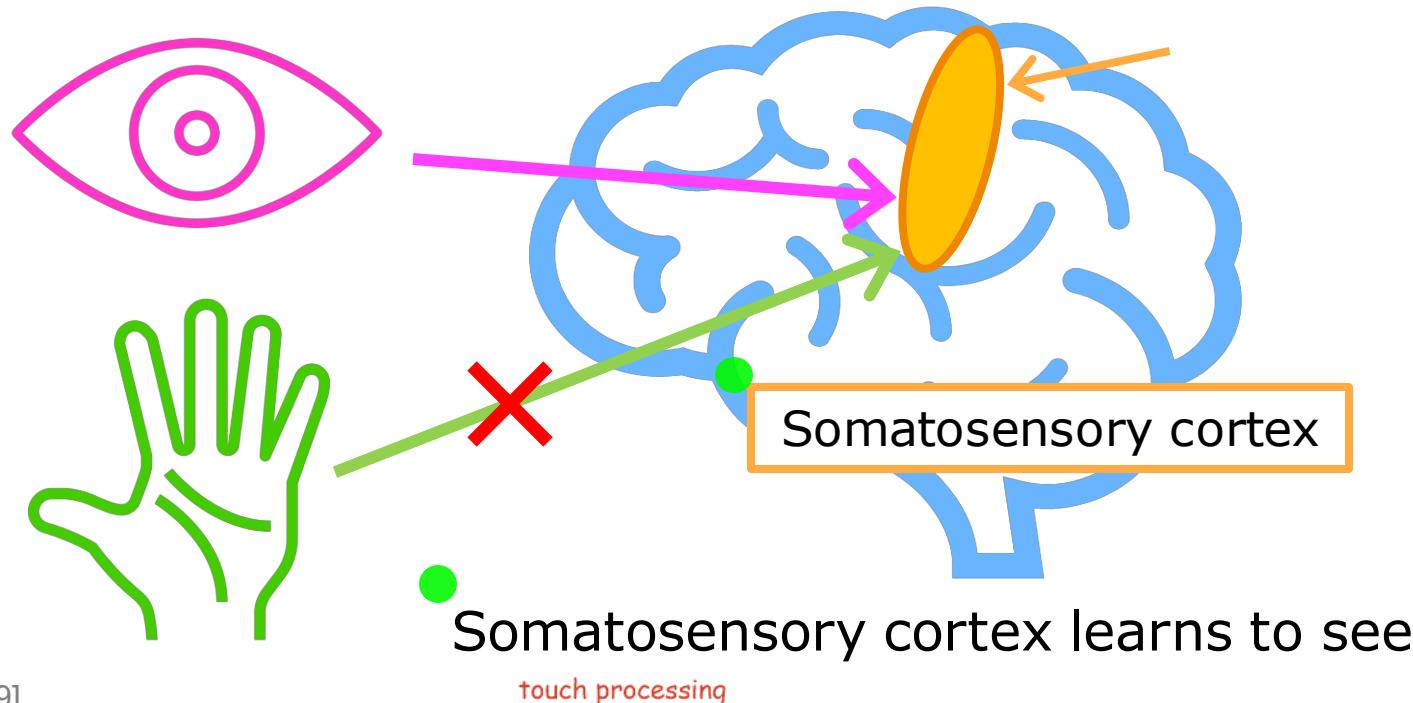
Path to AGI is very difficult path: 1- Artificial neurons are so simple compared to biological ones 2- We don't know brain and how it works!

# The "one learning algorithm" hypothesis



[Roe et al., 1992]

# The “one learning algorithm” hypothesis



[Metin & Frost, 1989]

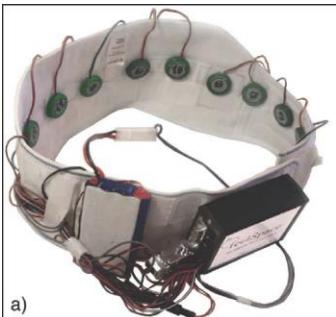
# Sensor representations in the brain



Seeing with your tongue



Human echolocation (sonar)

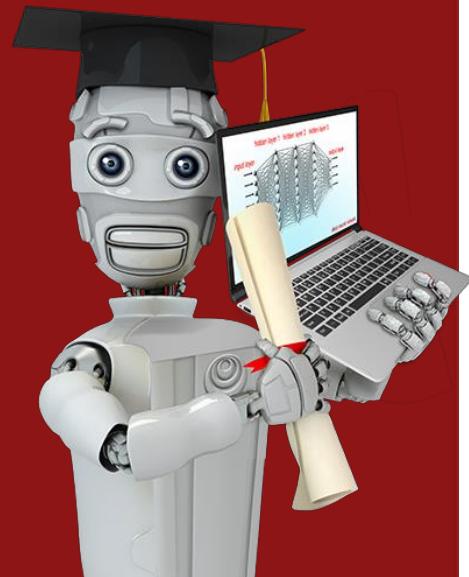


Haptic belt: Direction sense

[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**

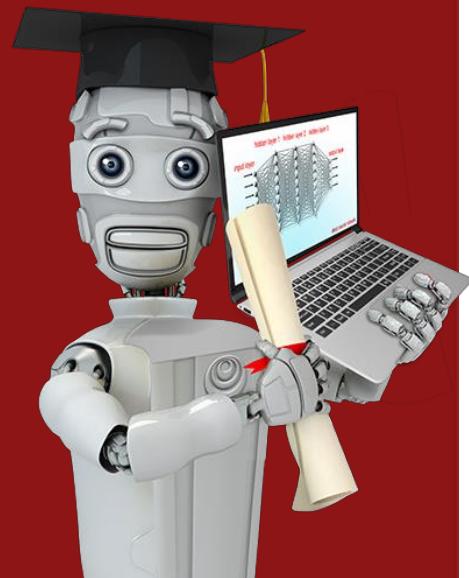
# For loops vs. vectorization

```
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]]
```

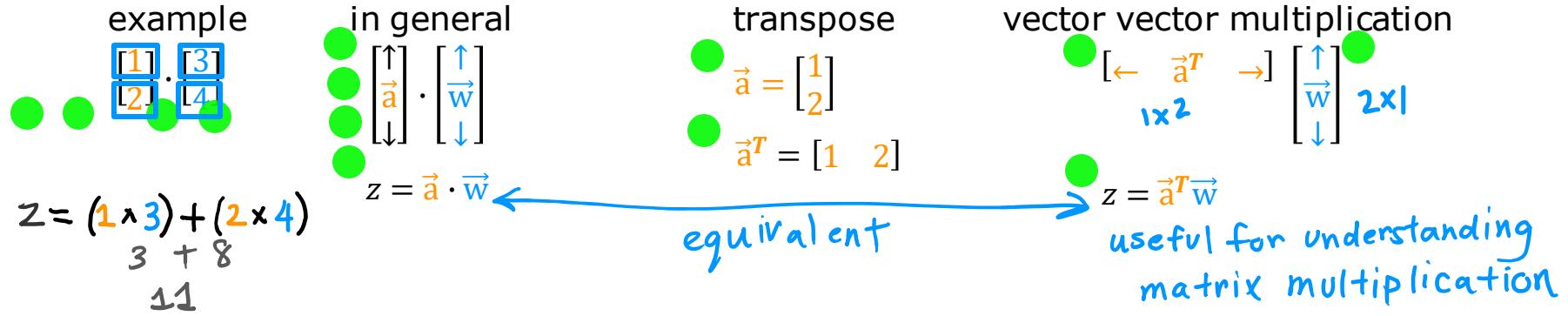
[1,0,1]



## 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**

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

**Columns**

$$\mathbf{W} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$
$$\mathbf{Z} = \mathbf{A}^T \mathbf{W} = \begin{bmatrix} \leftarrow & \xrightarrow{\mathbf{a}_1^T} & \rightarrow \\ \leftarrow & \xrightarrow{\mathbf{a}_2^T} & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \downarrow & \downarrow \end{bmatrix}$$

$\mathbf{w}_1$      $\mathbf{w}_2$

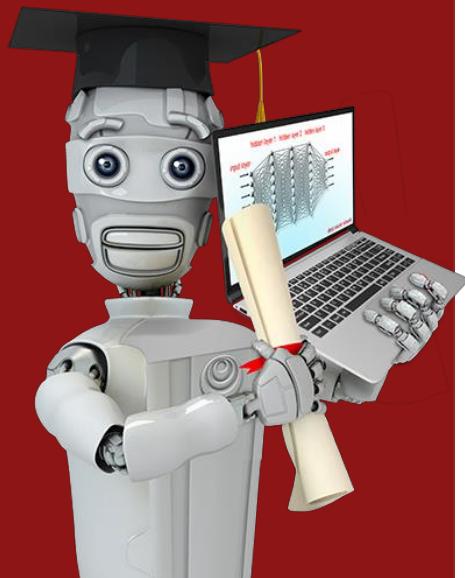
In Matrix, columns and in Transpose, rows got grouped together.

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

general rules for  
matrix multiplication  
↳ next video!

## Vectorization (optional)

# Matrix multiplication rules



# Matrix multiplication rules

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

# Matrix multiplication rules

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

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

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

$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix}$   
length 2

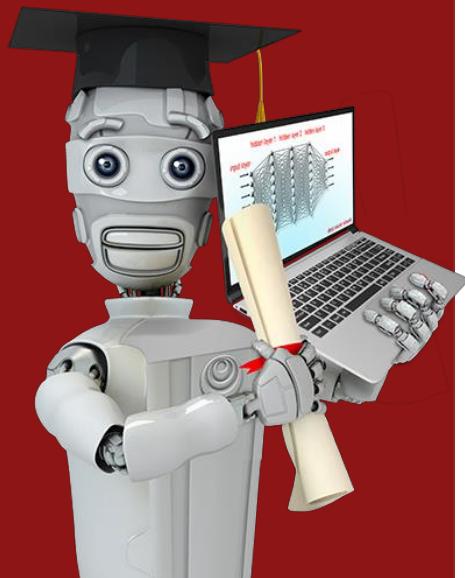
$\begin{bmatrix} 5 \\ 6 \end{bmatrix}$   
length 2

\* # columns of first matrix = # rows of second matrix

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])
```

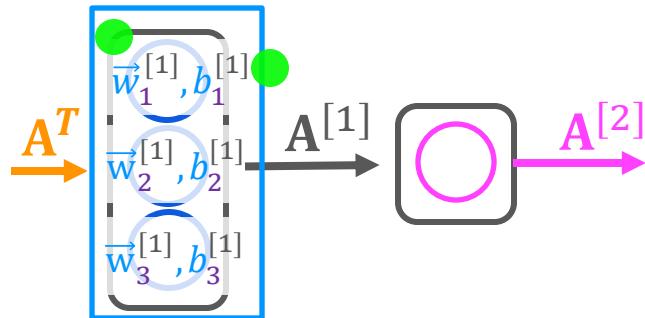
Z = np.matmul(AT,W)  
or  
Z = AT @ W

```
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]])
```

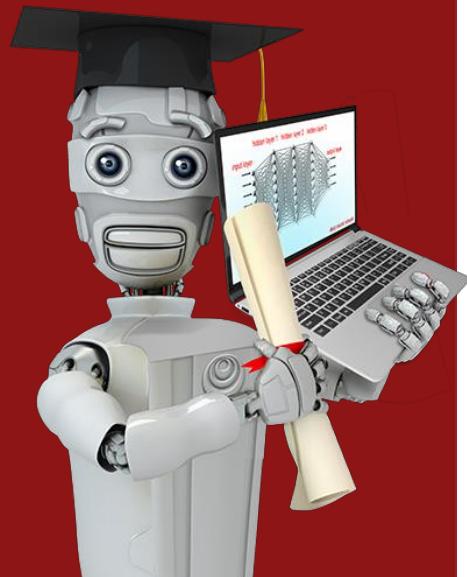
a-in

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

a-in

return a\_out

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



# Neural Network Training

---

TensorFlow  
implementation

Epoch: An epoch refers to one complete pass through the entire training dataset. If you have a dataset with 1,000 samples, and you run training for 10 epochs, the model will see each sample 10 times.

Iteration: An iteration, on the other hand, refers to one update of the model's weights. This is typically done after processing a batch of data. For example, if your dataset is split into batches of 100 samples and you have 1,000 samples, there would be 10 iterations per epoch.

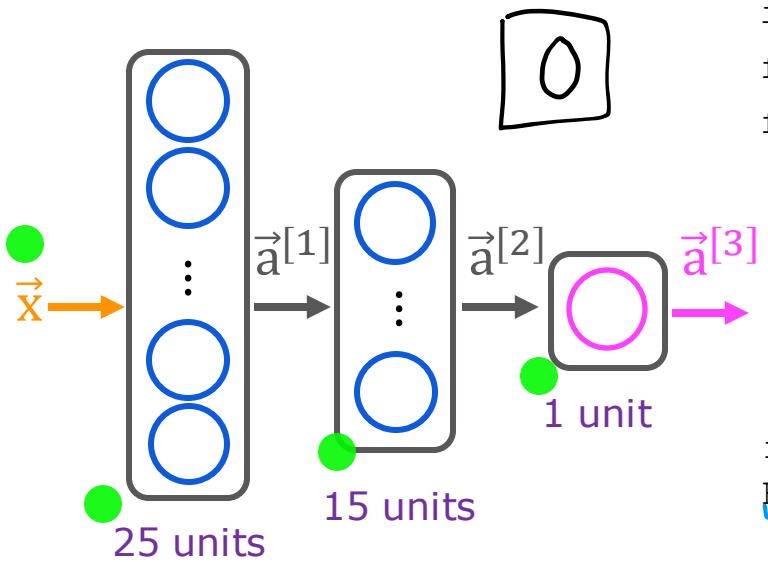
So, the number of iterations per epoch is determined by the size of your dataset and the batch size. For a given number of epochs, the total number of iterations is the number of epochs multiplied by the number of iterations per epoch.

Here's a formula to summarize:

Iterations per epoch = Total number of samples / Batch size

Total iterations = Epochs × Iterations per epoch

# Train a Neural Network in TensorFlow



Given set of  $(x, y)$  examples

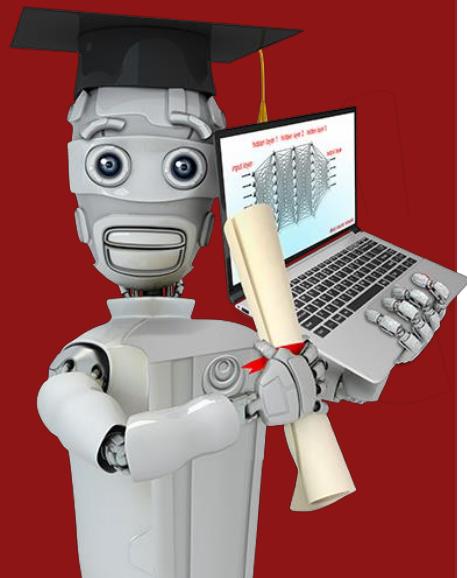
How to build and train this in code?

```
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

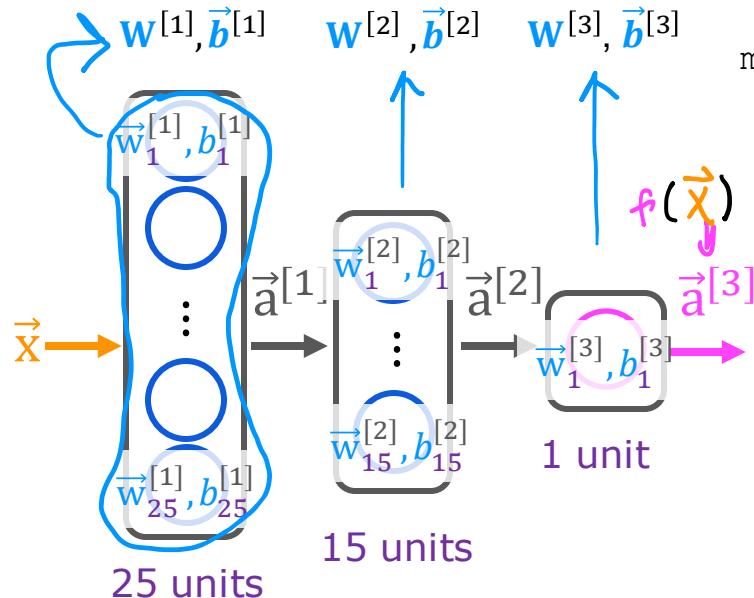
```
model.compile(  
    loss=BinaryCrossentropy())
```

```
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())
```

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

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

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

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

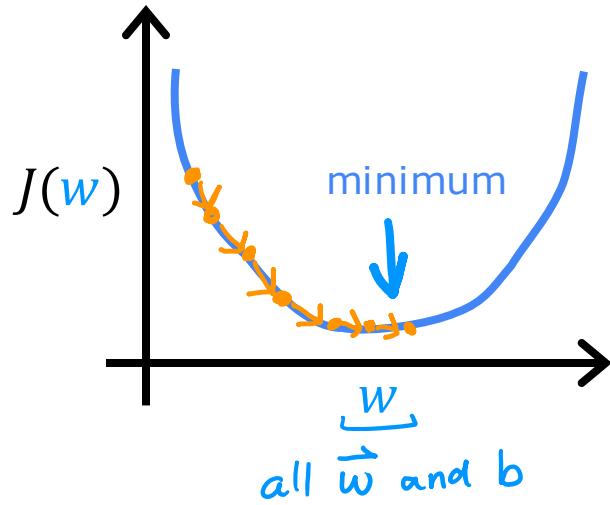
loss function

```
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"

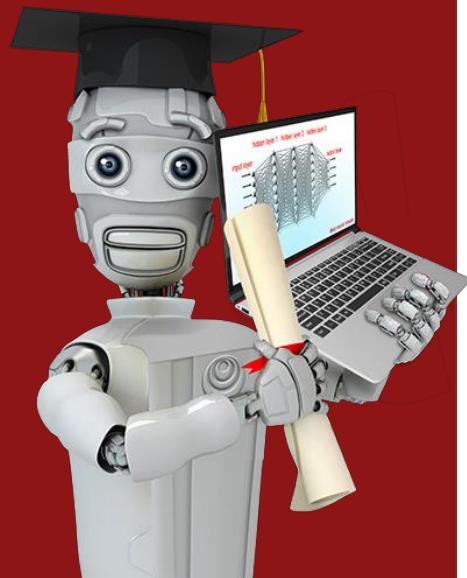
model.fit(X, y, epochs=100)

# Neural network libraries

Use code libraries instead of coding "from scratch"



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

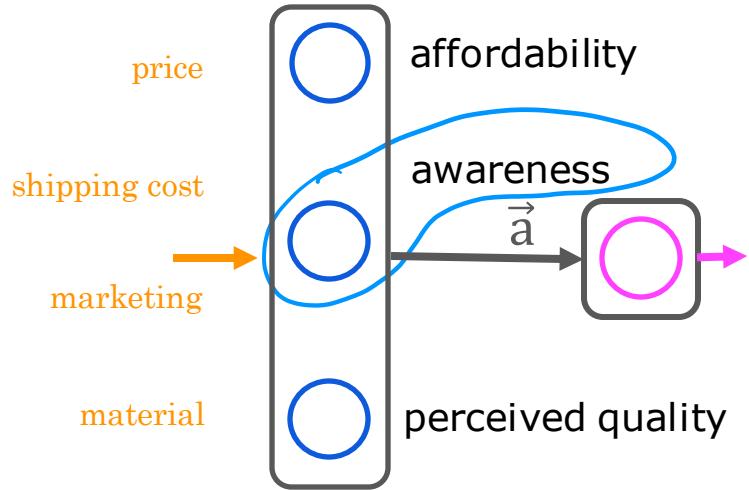


# Activation Functions

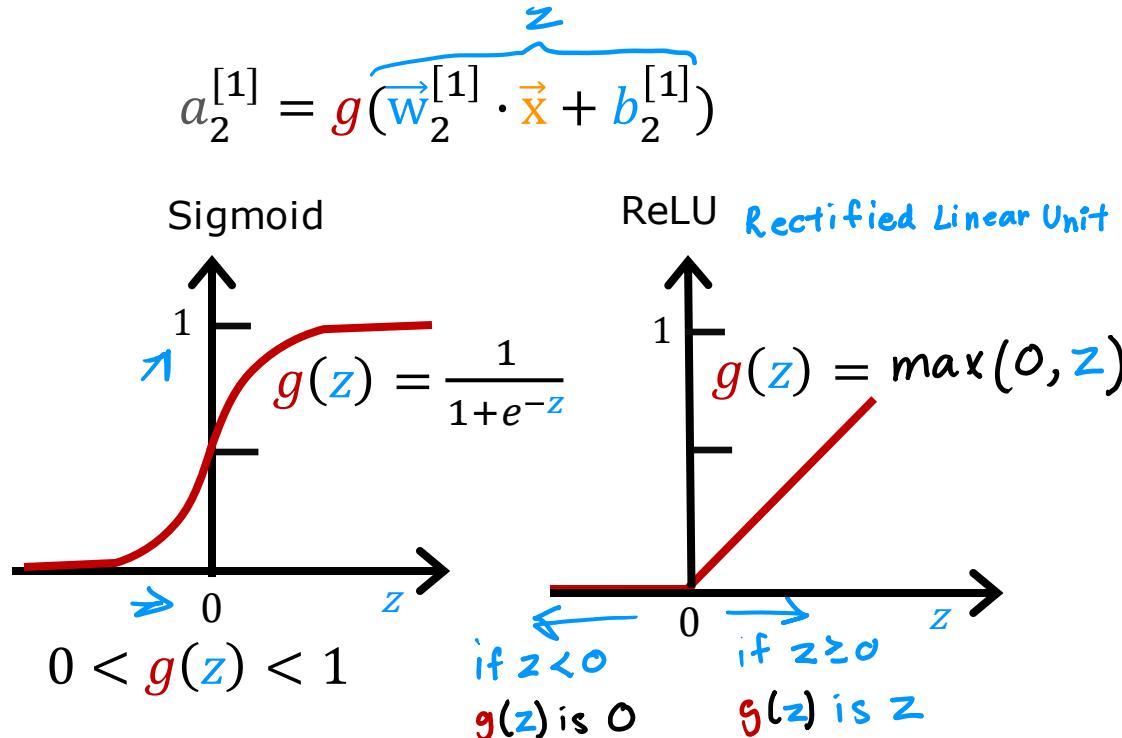
---

Alternatives to the  
sigmoid activation

# Demand Prediction Example



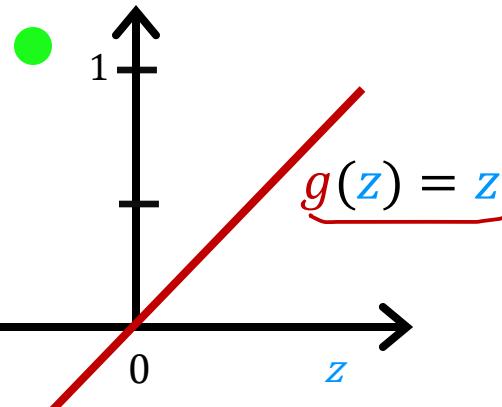
In logistic regression we assumed that awareness is either 0 or 1 but awareness has degrees to it (0- very large number)



# Examples of Activation Functions

"No activation function"

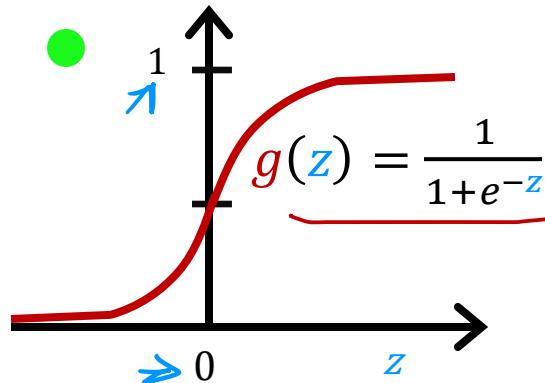
Linear activation function



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

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

Sigmoid

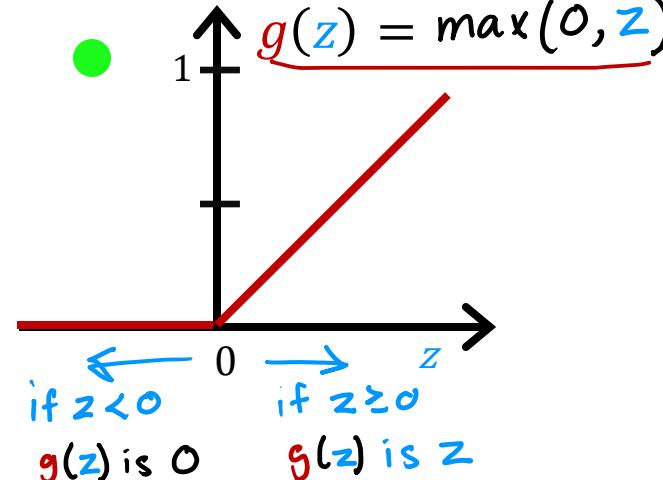


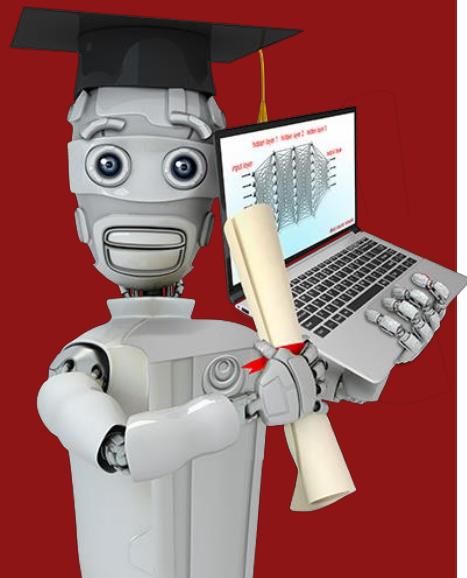
$$0 < g(z) < 1$$

Later: softmax activation

ReLU

Rectified Linear Unit



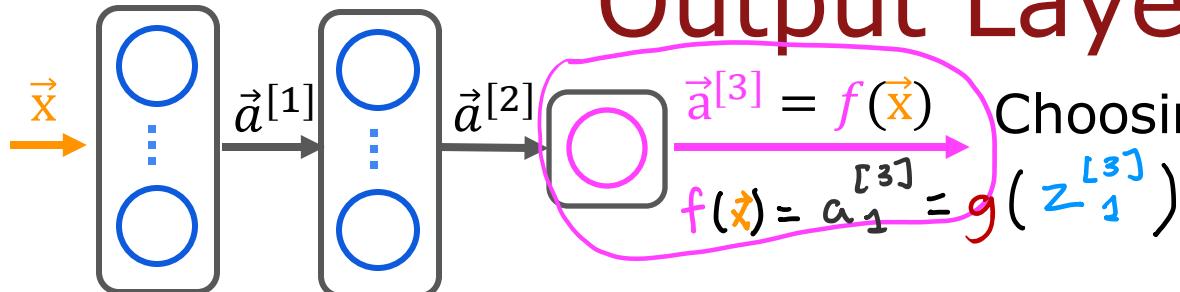


# Activation Functions

---

## Choosing activation functions

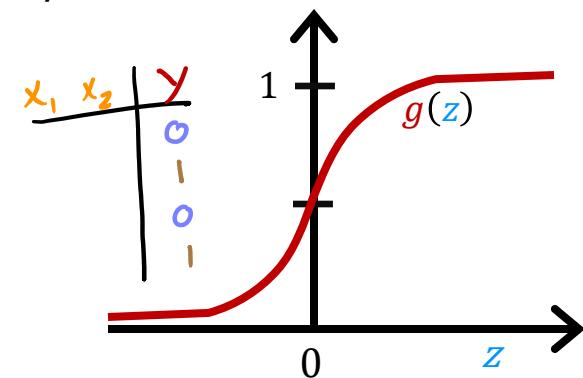
# Output Layer



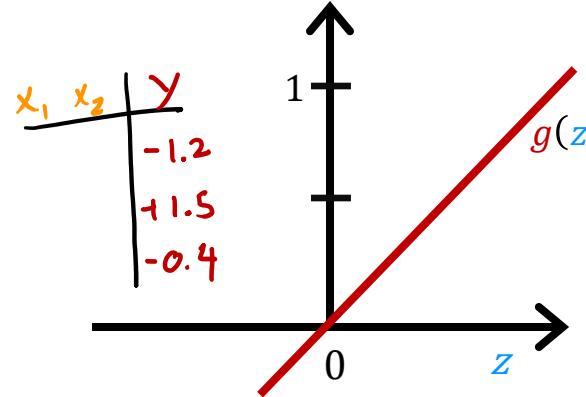
depending on  $y$ , there will be one fairly natural choice for the activation func of output layer

Choosing  $g(z)$  for output layer?

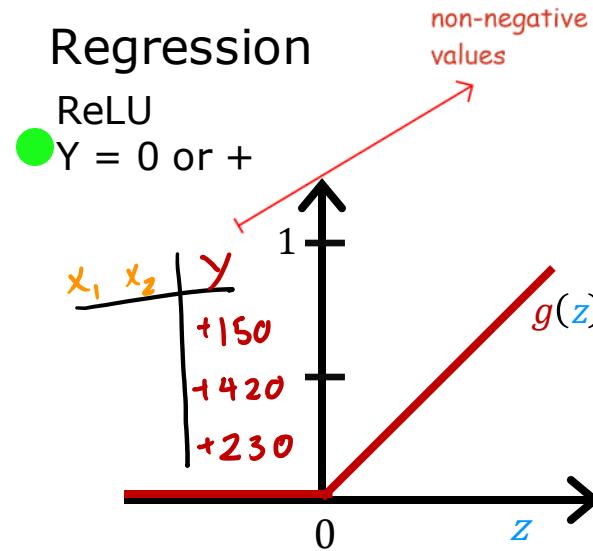
Binary classification  
Sigmoid  
 $y=0/1$



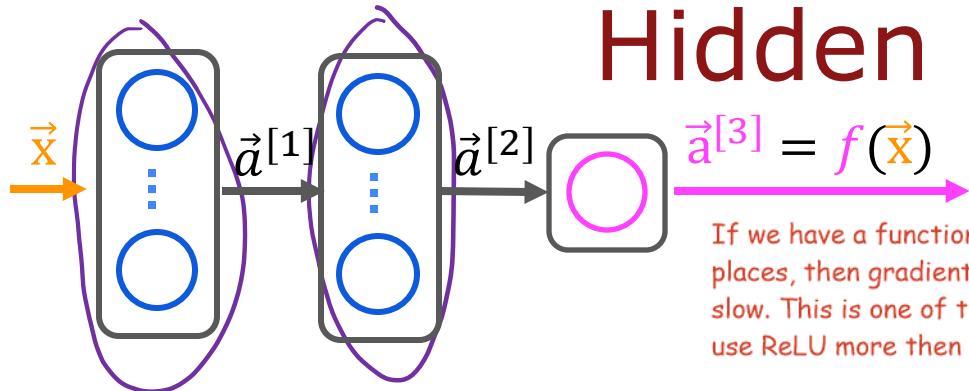
Regression  
Linear activation function  
 $y = +/-$



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



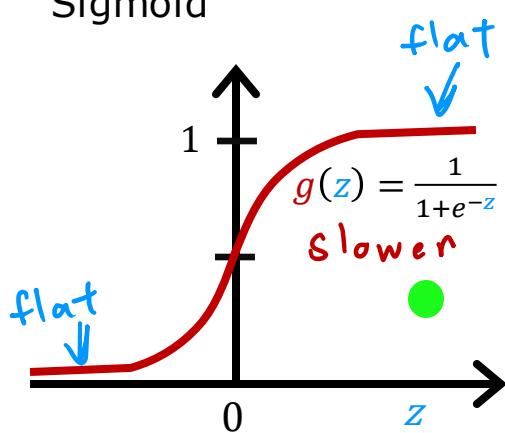
# Hidden Layer



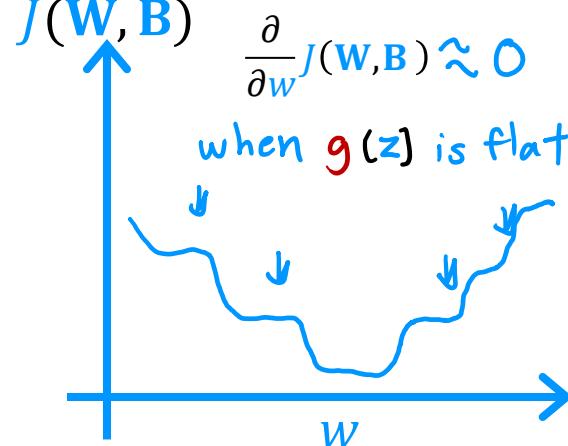
Choosing  $g(z)$  for hidden layer

If we have a function that is flat in many places, then gradient descent would be slow. This is one of the reasons that we use ReLU more than sigmoid

Sigmoid

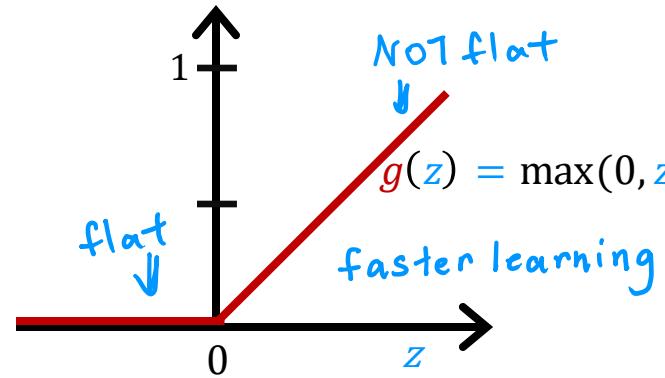


$J(W, B)$

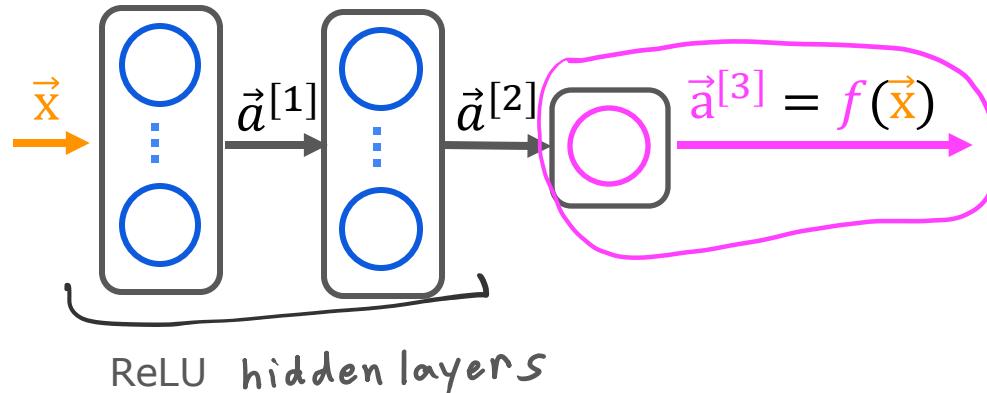


ReLU

faster

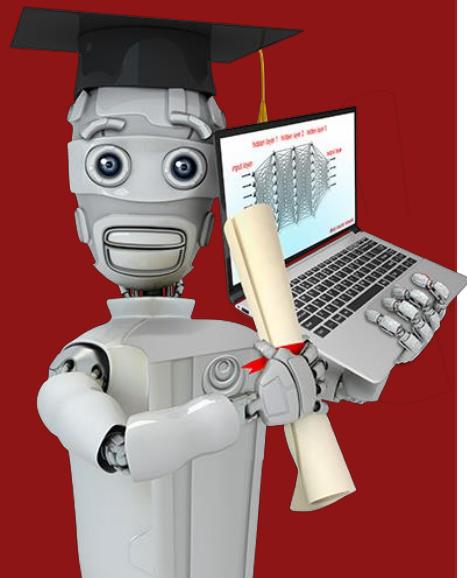


# Choosing Activation Summary



```
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'

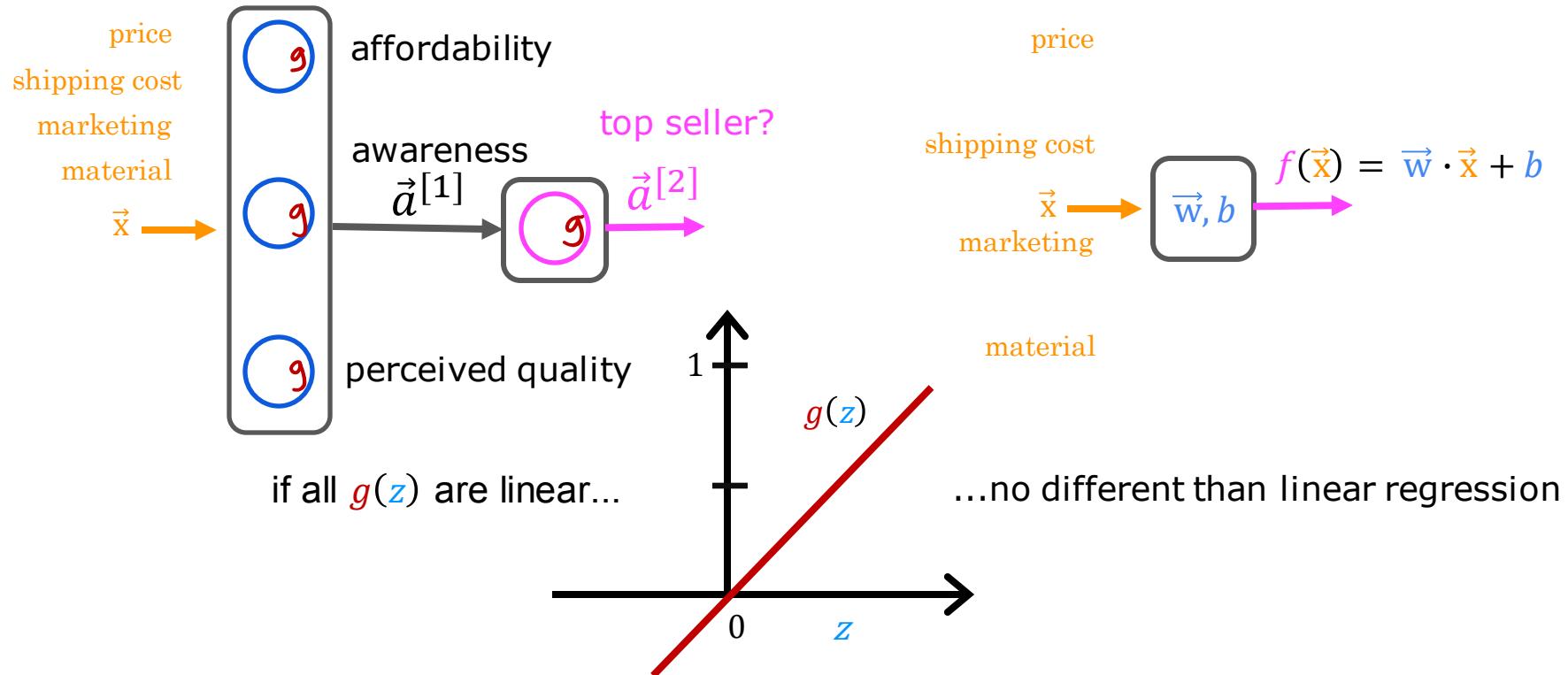


# Activation Functions

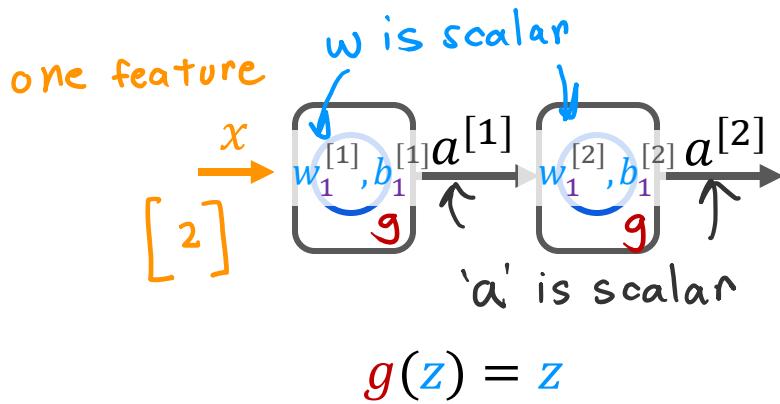
---

Why do we need activation functions?

# Why do we need activation functions?



# Linear Example



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

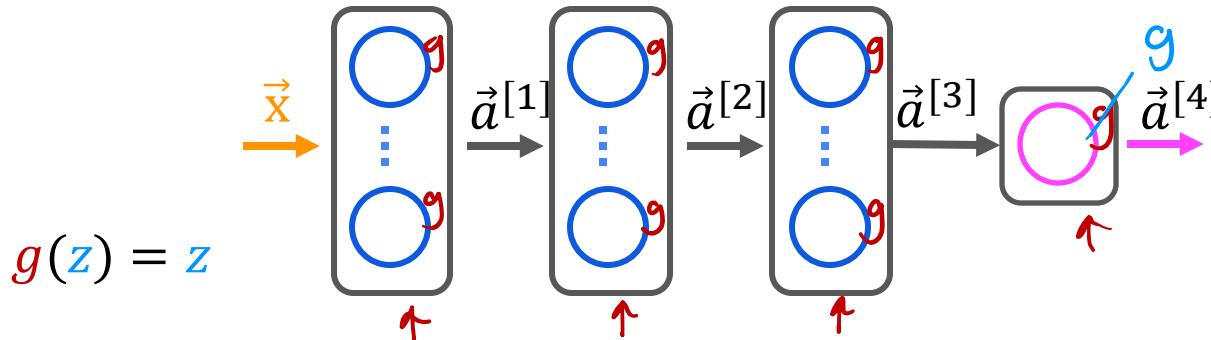
$$\vec{a}^{[2]} = w x + b$$

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

\* Comes from the fact: a linear func of a linear func is a linear func, Lol

neural network then can't compute any complex features or learn anything complex

# Example



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

all linear (including output)  
↳ equivalent to linear regression

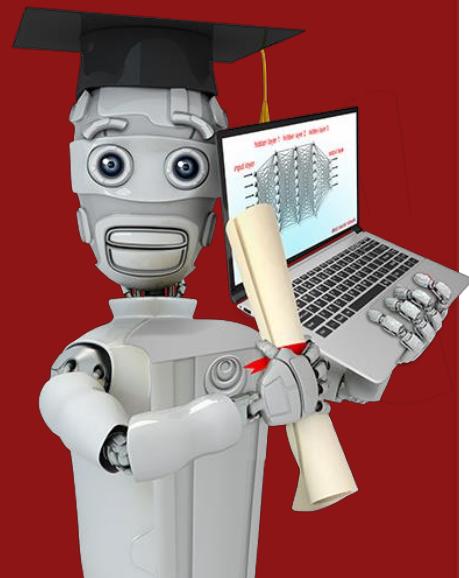
$$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]})}}$$

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

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

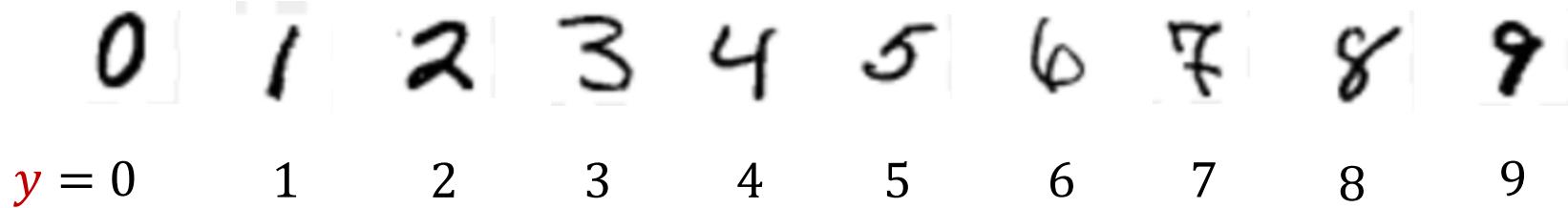
# Multiclass Classification

---



## Multiclass

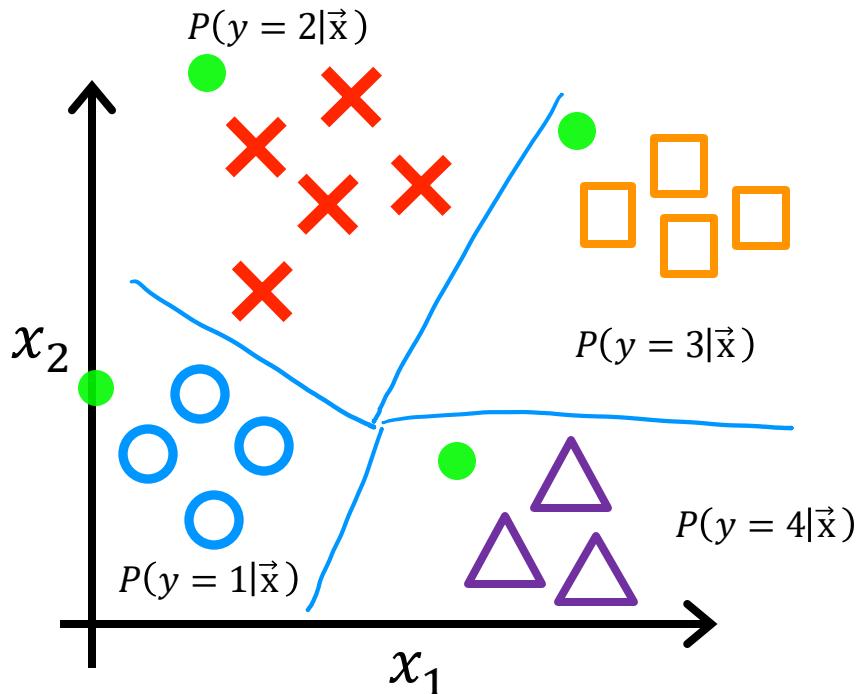
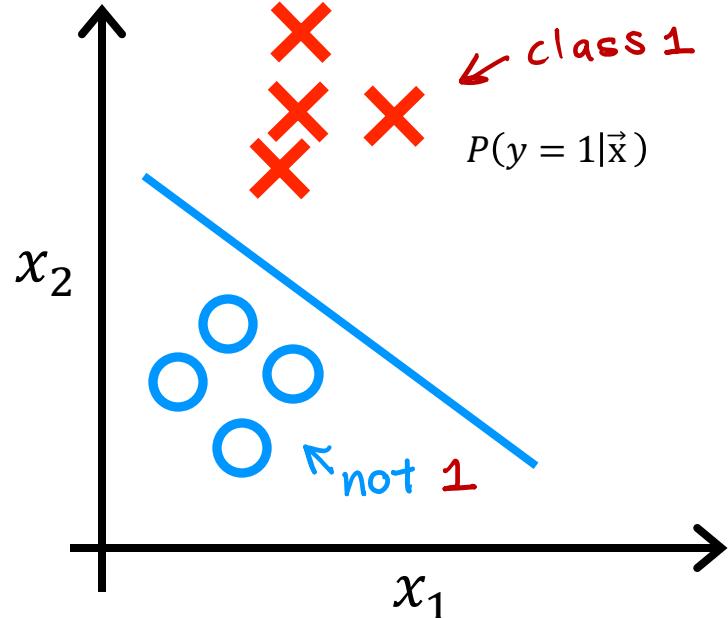
# MNIST example



$x \xrightarrow{\text{?}} 7$        $y = 7$

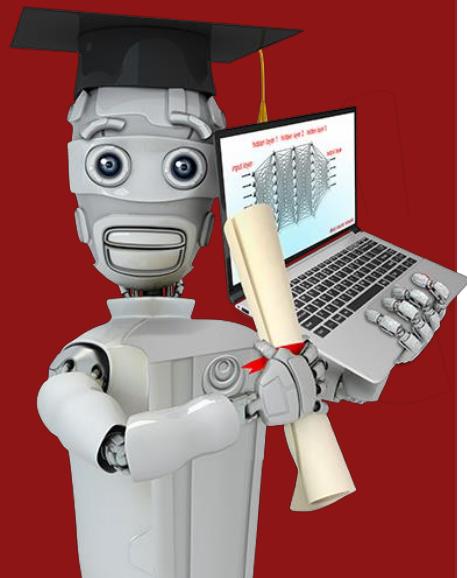
multiclass classification problem:  
target  $y$  can take on more than two possible values

# Multiclass classification example



# Multiclass Classification

---



## Softmax

*Generalization of Logistic regression*

## 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

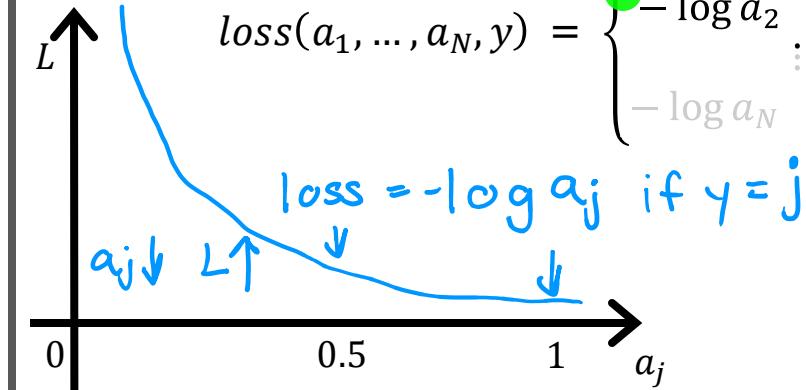
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1 | \vec{x})$$

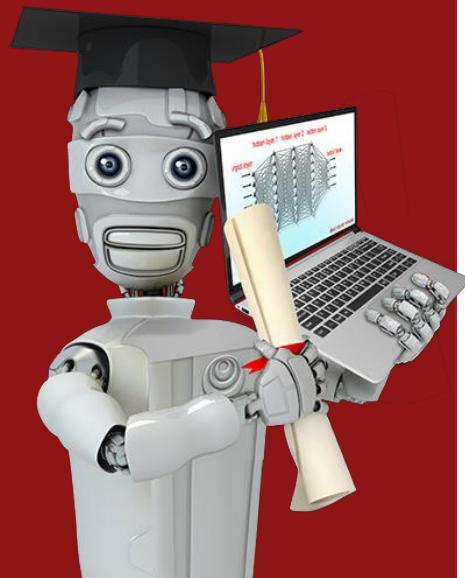
$$\vdots$$

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

### Crossentropy loss

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



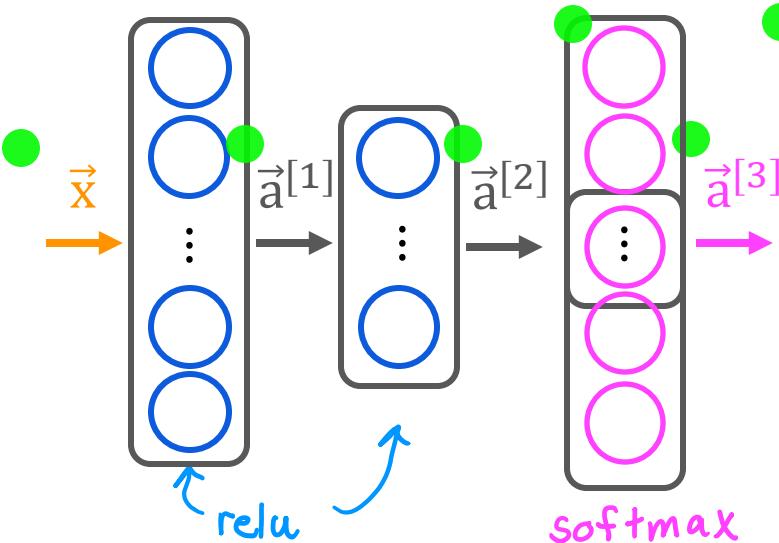


# Multiclass Classification

---

Neural Network with  
Softmax output

# Neural Network with Softmax output



25 units      15 units      10 units  
unique property of softmax: each activation value, depends on all values of z's. (unlike other activation funcs that were only dependent towards their corresponding z)

$$\vec{a}^{[1]} \rightarrow \vec{a}^{[2]} \rightarrow \vec{a}^{[3]}$$
$$z_1 = \vec{w}_1 \cdot \vec{a}^{[2]} + b_1$$
$$a_1 = \frac{e^{z_1}}{e^{z_1} + \dots + e^{z_{10}}} = P(y=1|\vec{x})$$
$$\vdots$$
$$z_{10} = \vec{w}_{10} \cdot \vec{a}^{[2]} + b_{10}$$
$$a_{10} = \frac{e^{z_{10}}}{e^{z_1} + \dots + e^{z_{10}}} = P(y=10|\vec{x})$$

logistic regression

$$a_1^{[3]} = g(z_1^{[3]}) \quad a_2^{[3]} = g(z_2^{[3]})$$

softmax

$$\vec{a}^{[3]} = (a_1^{[3]}, \dots, a_{10}^{[3]}) = g(z_1^{[3]}, \dots, z_{10}^{[3]})$$

# 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')
])
```

```
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
```

*sparse = y can only be one of  
the categories*

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

Note: better (recommended) version later.

*Don't use the version shown here!*

②

specify loss and cost

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

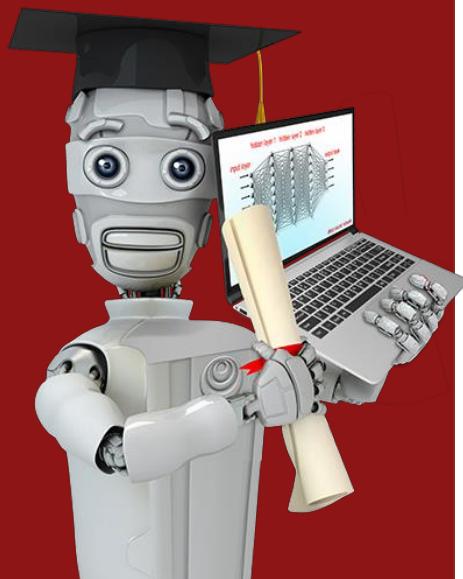
③

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

# Multiclass Classification

---

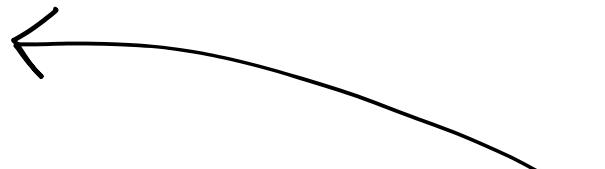
Improved implementation  
of softmax



# Numerical Roundoff Errors

option 1

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



option 2

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

option 2 has round-off error

# 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)

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

in logistic regression, each of these implementation (original and this new one) works ok. but in softmax we must use this new one

**logit: z**

effect: compute more accurately (less round-off error)

downside: less legible

# 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())~~

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

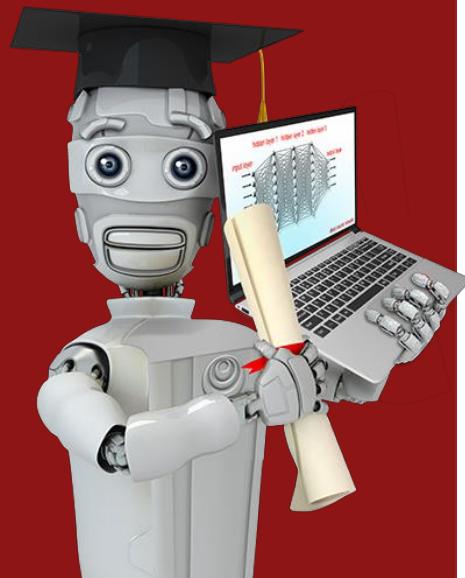
```
model.compile(..., BinaryCrossentropy(from_logits=True)) )
model.fit(X,Y,epochs=100)
```

fit

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

predict

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



# Multi-label Classification

---

Classification with  
multiple outputs  
(Optional)

# Multi-label Classification



Is there a car?

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

Is there a bus?

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

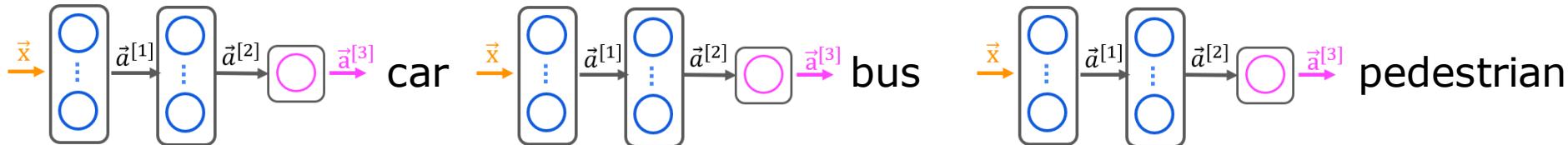
Is there a pedestrian?

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

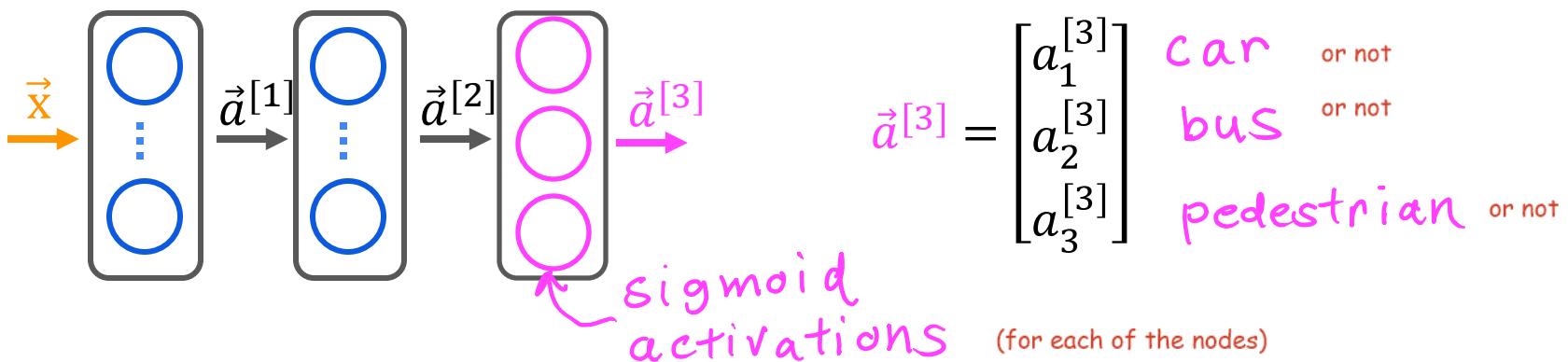
In multiclass,  $y$  was a single number that could be  $n$  different values.

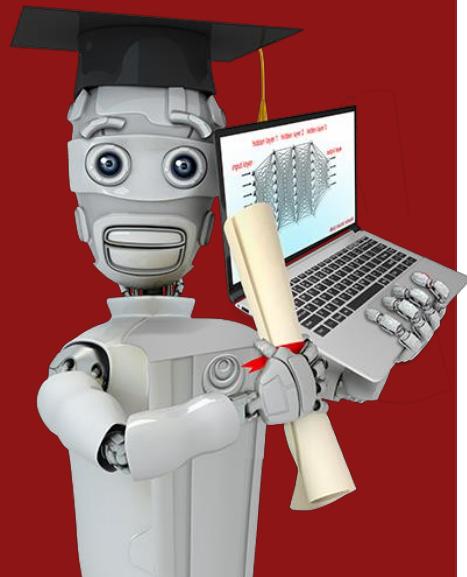
In multi-label,  $y$  can have  $n$  values (in a vector)

# Multiple classes



Alternatively, train one neural network with three outputs





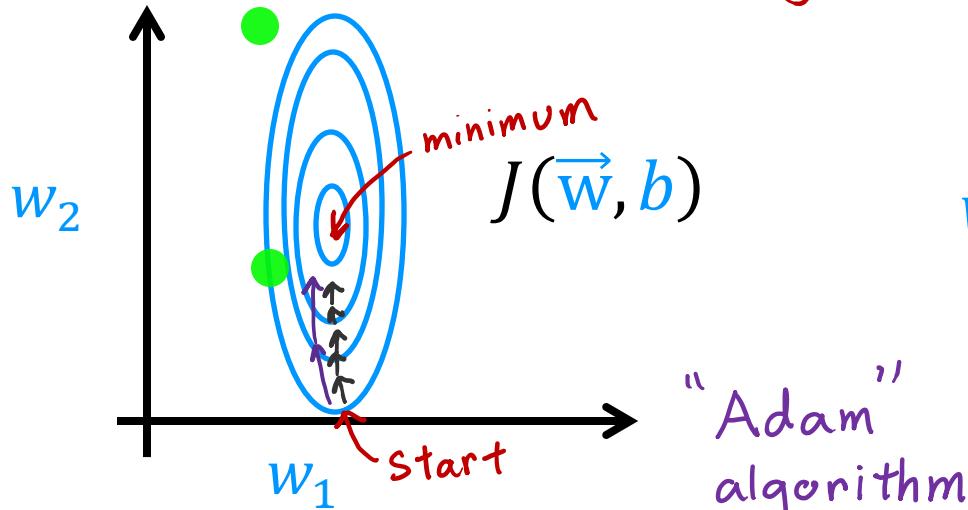
## Additional Neural Network Concepts

# Advanced Optimization

# Gradient Descent

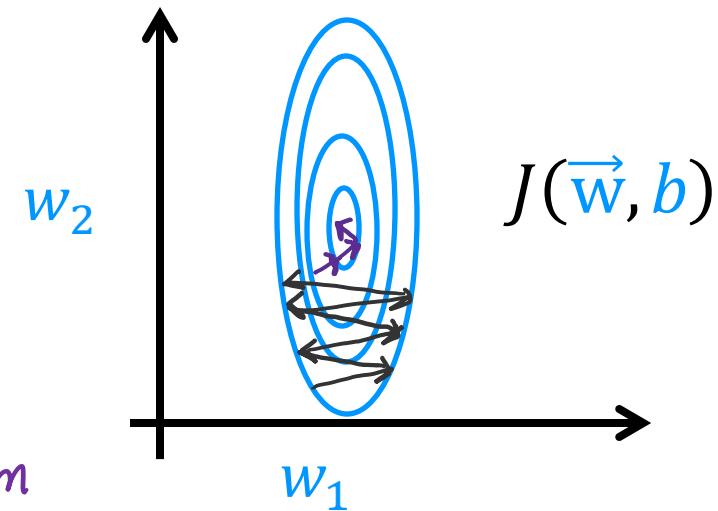
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

↑ learning rate



Go faster – increase  $\alpha$

"Adam" algorithm



Go slower – decrease  $\alpha$

# Adam Algorithm Intuition

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

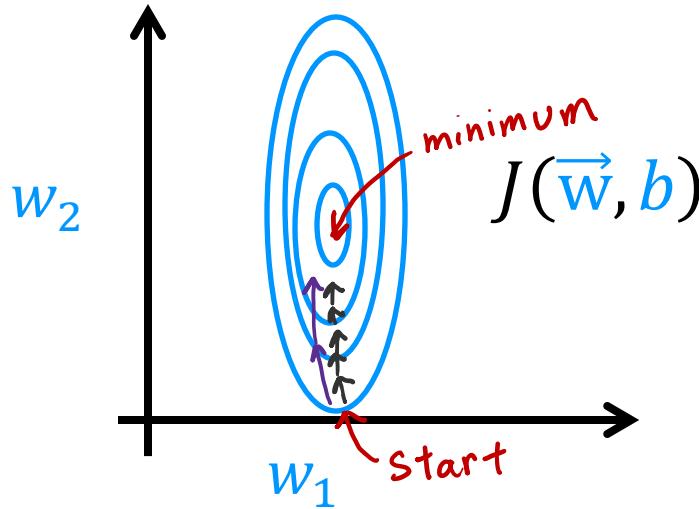
$$w_1 = w_1 - \underbrace{\alpha_1}_{\text{red}} \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

⋮

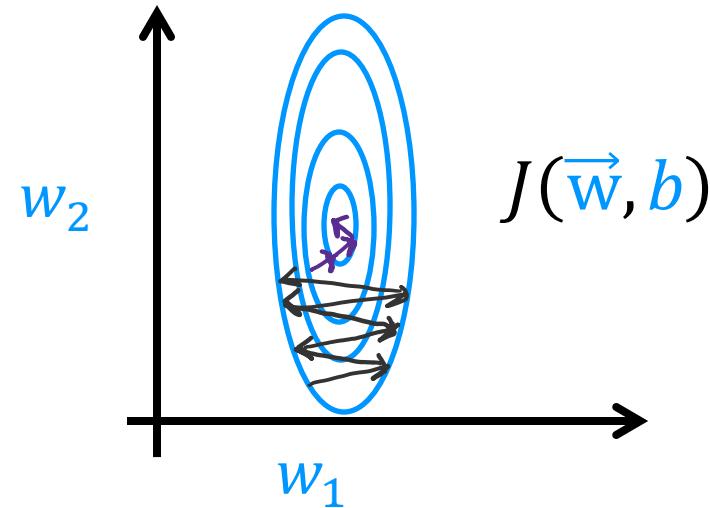
$$w_{10} = w_{10} - \underbrace{\alpha_{10}}_{\text{red}} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$

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

# 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')
])
```

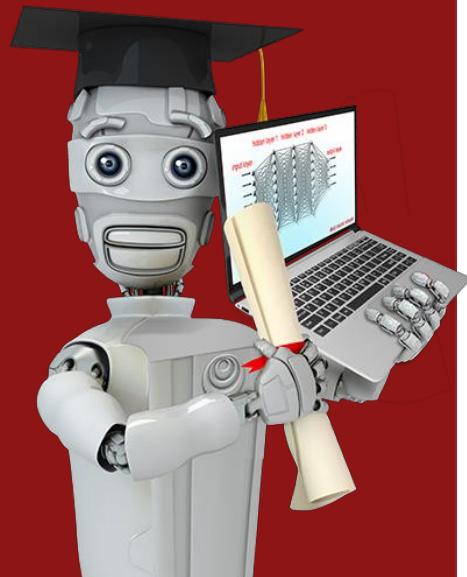
compile

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

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

fit

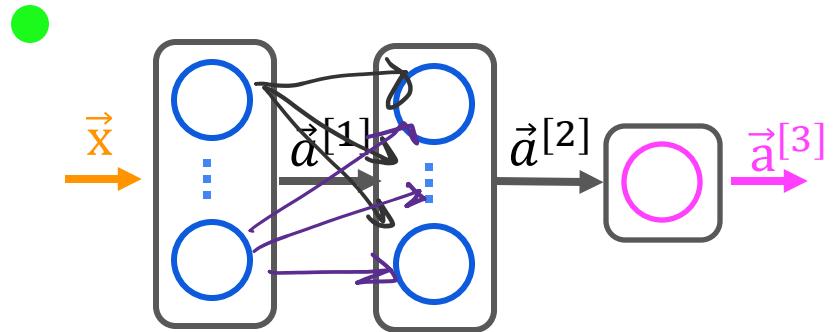
```
model.fit(X, Y, epochs=100)
```



## Additional Neural Network Concepts

### Additional Layer Types

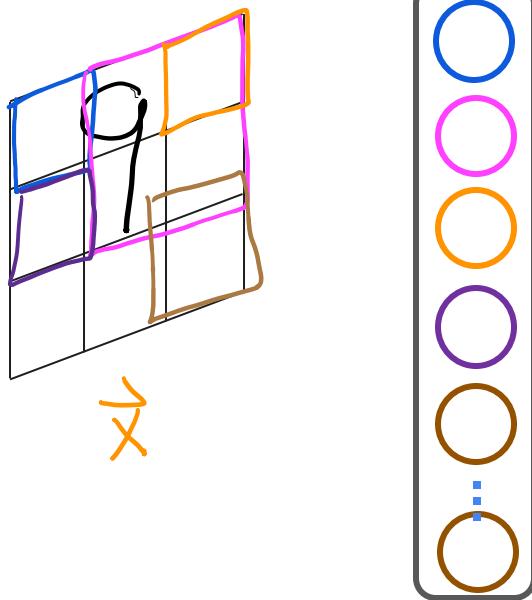
# Dense Layer



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

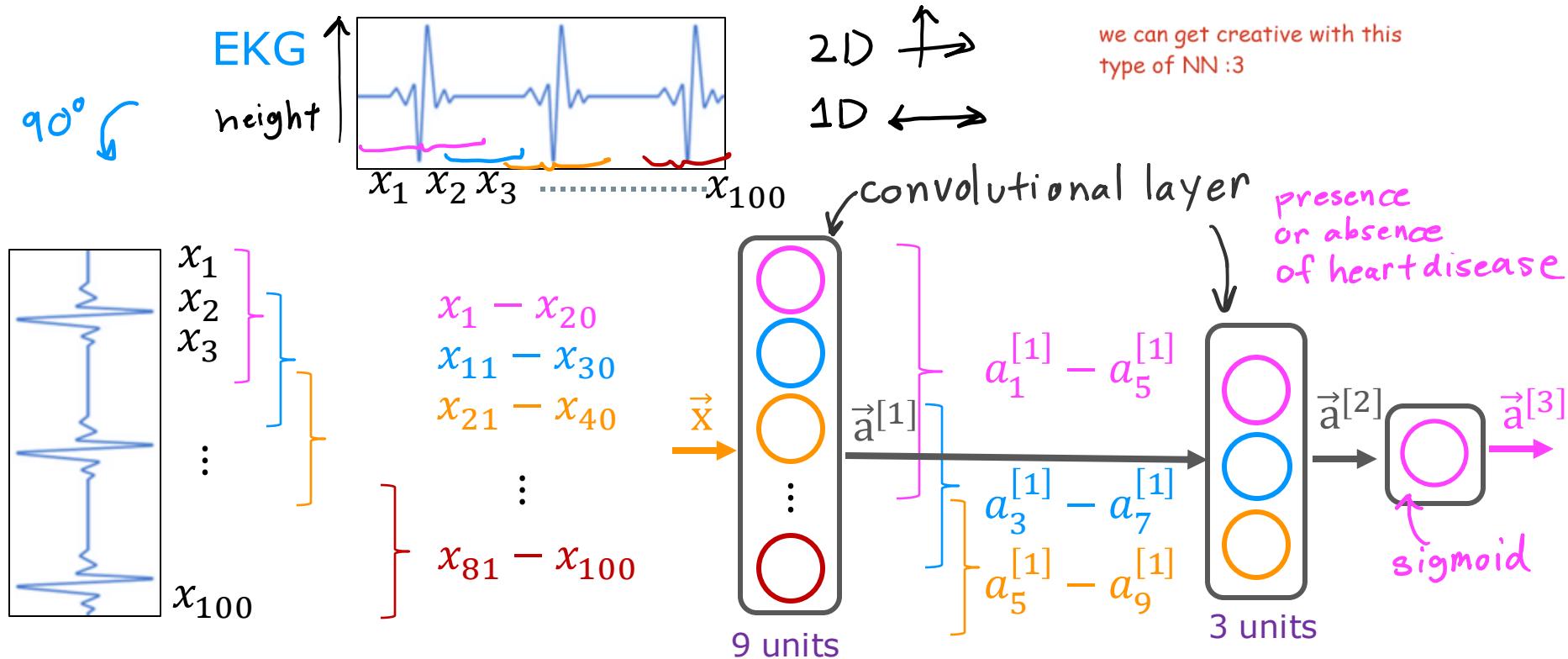


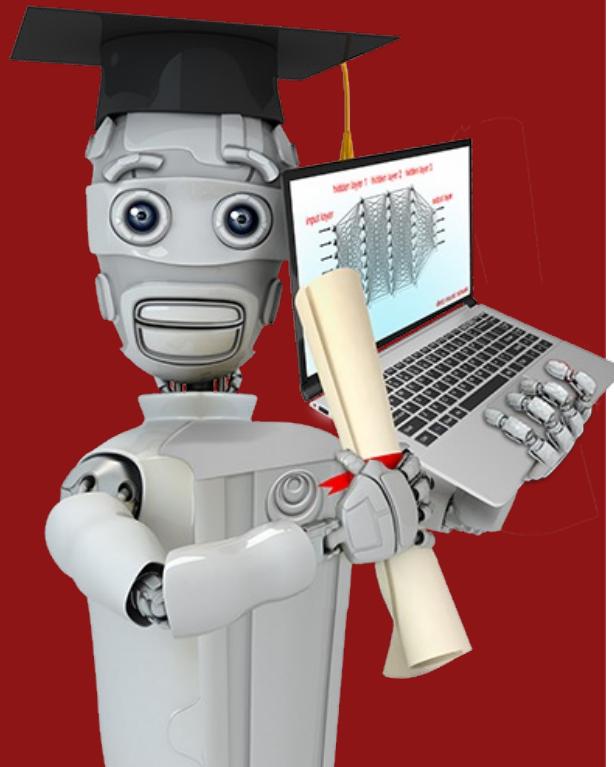
Each Neuron only looks at part of the previous layer's inputs.

Why?

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

# Convolutional Neural Network





## Backprop Intuition (Optional)

**What is a derivative?**

# Derivative Example

Cost function

$$J(w) = w^2$$

Say  $w = 3$

$$J(w) = 3^2 = 9$$

$$\epsilon = 0.002$$

If we increase  $w$  by a tiny amount  $\epsilon = 0.001$  how does  $J(w)$  change?

$$w = 3 + \cancel{0.001} \quad 0.002$$

$$J(w) = w^2 = \cancel{9.006001} \\ \underbrace{\quad\quad\quad}_{\text{9.012}} \quad \underbrace{\quad\quad\quad}_{\text{0.002}} \\ \underbrace{9.012004}_{\text{9.012}}$$

If  $w \uparrow \cancel{0.001}$        $\epsilon \leftarrow$

$J(w) \uparrow \cancel{6 \times 0.001}$        $6 \times \epsilon$        $6 \times 0.002 = 0.012$

$\frac{\partial}{\partial w} J(w) = 6$

# Informal Definition of Derivative

If  $\underbrace{w \uparrow \varepsilon}$  causes  $\underbrace{J(w) \uparrow k \times \varepsilon}$  then

$$\frac{\partial}{\partial w} J(w) = k$$

Gradient descent

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

learning rate

If derivative is small, then this update step will make a small update to  $w_j$

If the derivative is large, then this update step will make a large update to  $w_j$

# More Derivative Examples

$w = 3$

$J(w) = w^2 = 9$

$w \uparrow 0.001$

$J(w) = J(3.001) = 9.006001$

$\frac{\partial}{\partial w} J(w) = 6$

$J(w) \uparrow 6 \times 0.001$

$w = 2$

$J(w) = w^2 = 4$

$w \uparrow 0.001$

$J(w) = J(2.001) = 4.004001$

$\frac{\partial}{\partial w} J(w) = 4$

$J(w) \uparrow 4 \times 0.001$

$\downarrow 0.006$

$w = -3$

$J(w) = w^2 = 9$

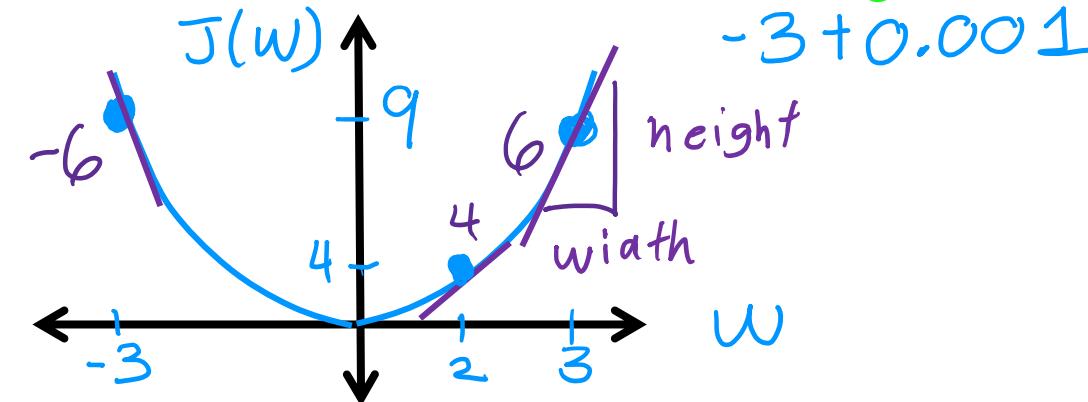
$w \uparrow 0.001$

$J(w) = J(-2.999) = 8.994001$

$\frac{\partial}{\partial w} J(w) = -6$

$J(w) \downarrow 6 \times 0.001$

$J(w) \uparrow -6 \times 0.001$



Calculus	$w$	$\frac{\partial J(w)}{\partial w}$
$\frac{\partial}{\partial w} J(w) = 2w$	3	$2 \times 3 = 6$
	$\frac{2}{3}$	$2 \times \frac{2}{3} = 4$
	-3	$2 \times -3 = -6$

# Even More Derivative Examples

$w = 2$

$J(w) = w^2 = 4$	$\frac{\partial}{\partial w} J(w) = 2w = 4$	$w \uparrow \underbrace{0.001}_{\varepsilon}$	$J(w) = 4.004001$ $J(w) \uparrow 4 \times \varepsilon$
$J(w) = w^3 = 8$	$\frac{\partial}{\partial w} J(w) = 3w^2 = 12$	$w \uparrow \varepsilon$	$J(w) = 8.012006$ $J(w) \uparrow 12 \times \varepsilon$
$J(w) = w = 2$	$\frac{\partial}{\partial w} J(w) = 1$	$w \uparrow \varepsilon$	$J(w) = 2.001$ $J(w) \uparrow 1 \times \varepsilon$
$J(w) = \frac{1}{w} = \frac{1}{2} = 0.5$	$\frac{\partial}{\partial w} J(w) = -\frac{1}{w^2} = -\frac{1}{4}$	$w \uparrow \varepsilon$ $w = \frac{1}{2.001}$	$-0.25 \times 0.001$ $0.5 - \frac{0.00025}{J(w)} = 0.49975$ $J(w) \uparrow -\frac{1}{4} \times \varepsilon$

$\frac{\partial}{\partial w} J(w) \quad w \uparrow \varepsilon \quad J(w) \uparrow k \times \varepsilon$

# A note on derivative notation

If  $J(w)$  is a function of one variable ( $w$ ),

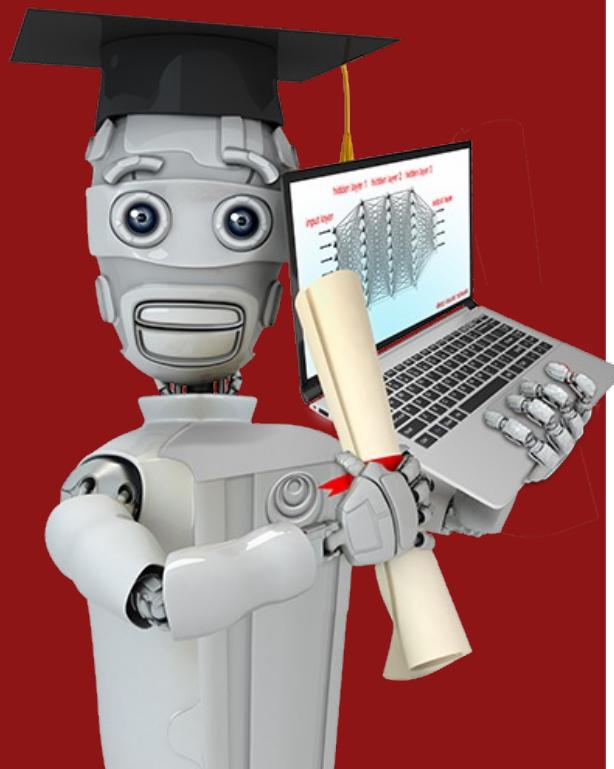
$$d \frac{d}{dw} J(w)$$

If  $J(w_1, w_2, \dots, w_n)$  is a function of more than one variable,

- $\frac{\partial}{\partial w_i} J(w_1, w_2, \dots, w_n) \quad \frac{\partial J}{\partial w_i} \quad \text{or} \quad \frac{\partial}{\partial w_i} J$

"partial derivative"

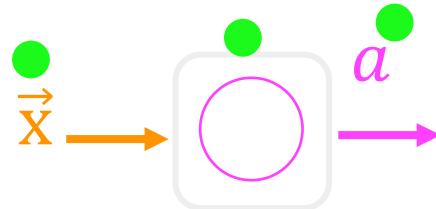
notation used  
in these courses



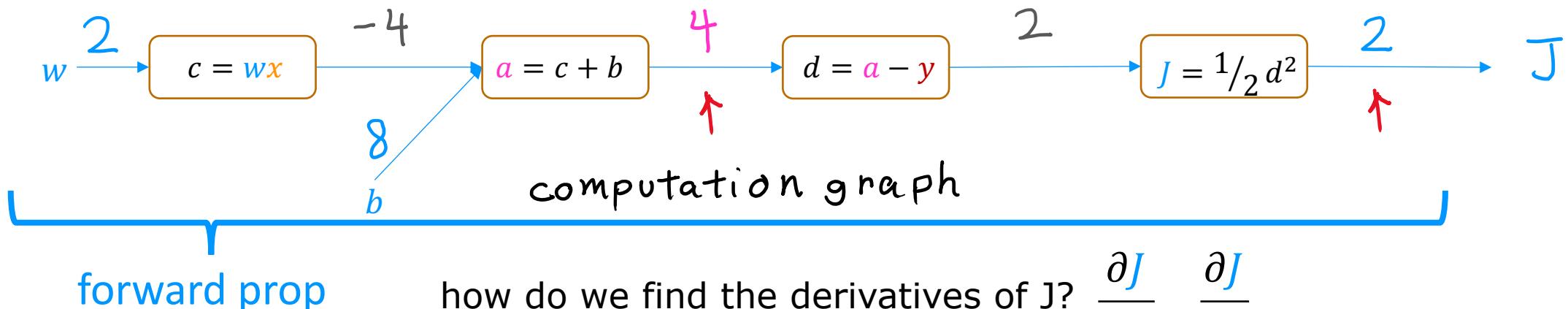
## Backprop Intuition (Optional)

## Computation Graph

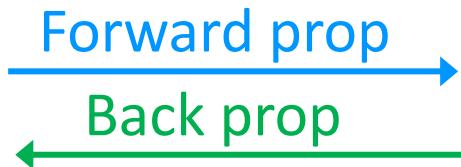
# Small Neural Network Example



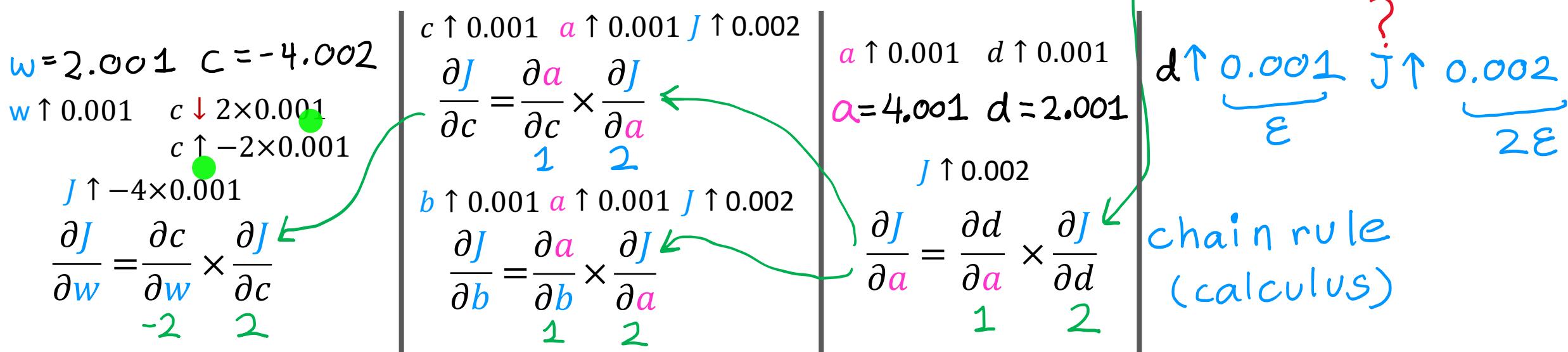
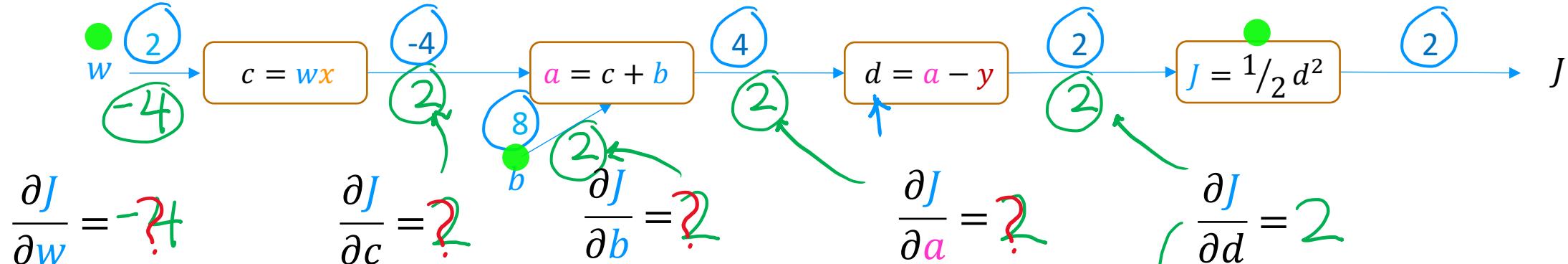
$$w=2 \quad b=8 \quad x=-2 \quad y=2$$
$$a = wx + b \quad \text{linear activation } a = g(z) = z$$
$$J(w,b) = \frac{1}{2} (a - y)^2$$



# Computing the Derivatives



$$w = 2 \quad b = 8 \quad x = -2 \quad y = 2 \quad a = wx + b \quad J = \frac{1}{2}(a - y)^2$$



# Computing the Derivatives

$w = 2$

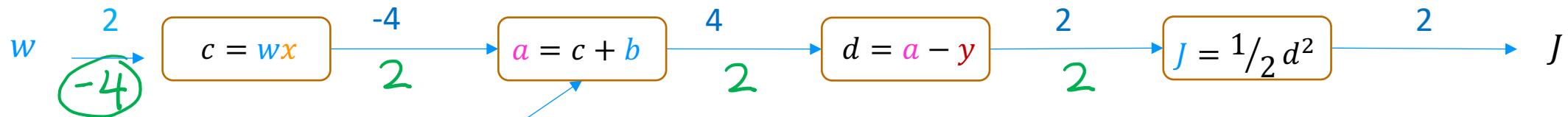
$b = 8$

$x = -2$

$y = 2$

$a = wx + b$

$J = \frac{1}{2}(a - y)^2$



$\frac{\partial J}{\partial w} = -4$

$\frac{\partial J}{\partial c} = 2$

$\frac{\partial J}{\partial b} = 2$

$\frac{\partial J}{\partial a} = 2$

$\frac{\partial J}{\partial d} = 2$

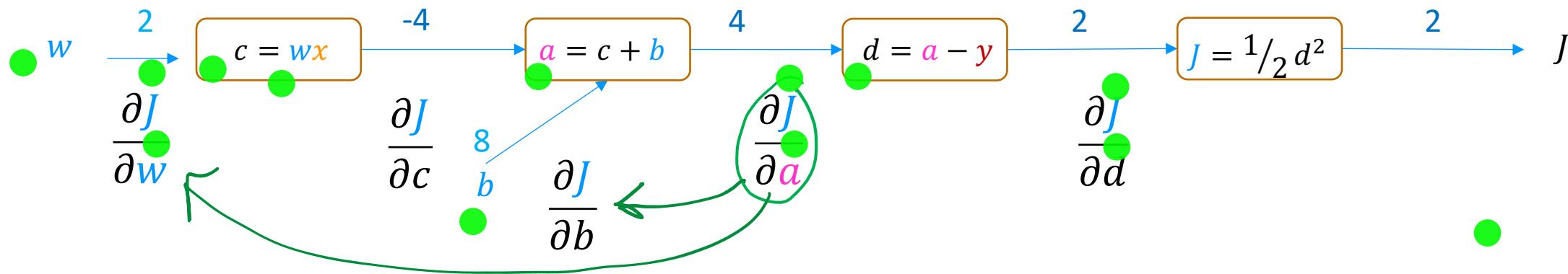
$$J = \frac{1}{2}((wx + b) - y)^2 = \frac{1}{2}((2 \times -2 + 8) - 2)^2 = 2$$

$w \uparrow 0.001 \quad J = \frac{1}{2}((2.001 \times -2 + 8) - 2)^2 = 1.\underline{996}002$

$$\begin{array}{l} J \downarrow 4 \times 0.001 \\ J \uparrow -4 \times 0.001 \end{array}$$

$\frac{\partial J}{\partial w} = -4$

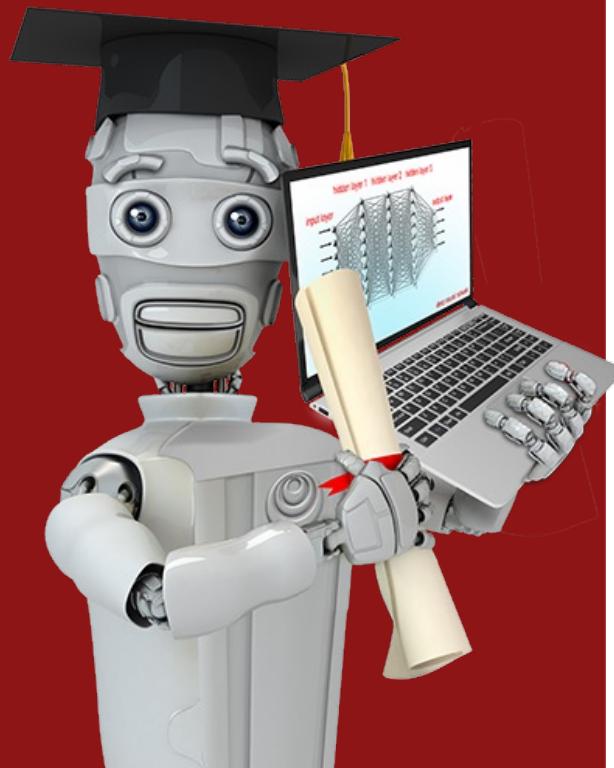
# Backprop is an efficient way to compute derivatives



Compute  $\frac{\partial J}{\partial a}$  once and use it to compute both  $\frac{\partial J}{\partial w}$  and  $\frac{\partial J}{\partial b}$ .

If N nodes and P parameters, compute derivatives  
in roughly  $N + P$  steps rather than  $N \times P$  steps.

N	P	$N + P$	$N \times P$
10,000	100,000	$1.1 \times 10^5$	$10^9$

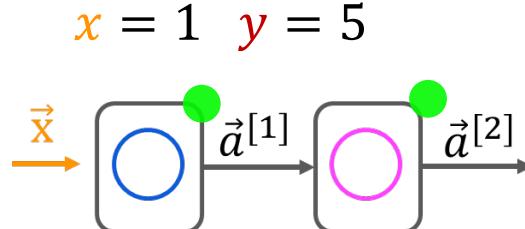


# Backprop Intuition (Optional)

## Larger Neural Network Example

# Neural Network Example

$$x = 1 \quad y = 5$$

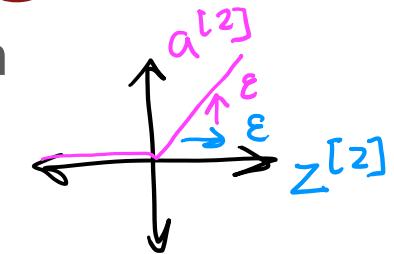


$$w^{[1]} = 2, b^{[1]} = 0$$

$$w^{[2]} = 3, b^{[2]} = 1$$

ReLU activation

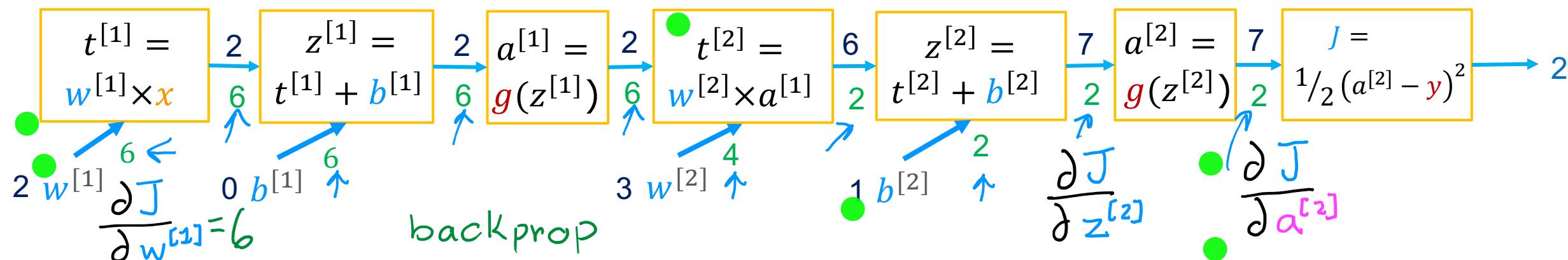
$$g(z) = \max(0, z)$$



$$a^{[1]} = g(w^{[1]} x + b^{[1]}) = \underbrace{w^{[1]} x + b^{[1]}}_{z^{[1]}} = 2 \times 1 + 0 = 2$$

$$a^{[2]} = g(w^{[2]} a^{[1]} + b^{[2]}) = \underbrace{w^{[2]} a^{[1]} + b^{[2]}}_{z^{[2]}} = 3 \times 2 + 1 = 7$$

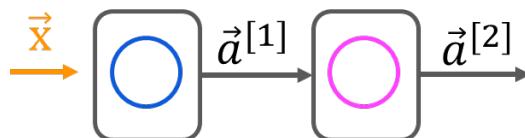
$$J(w, b) = \frac{1}{2} (a^{[2]} - y)^2 = \frac{1}{2} (7 - 5)^2 = 2$$



if  $w^{[1]} \uparrow \epsilon$ , then  $J \uparrow 6 \times \epsilon$  let's verify this!

# Neural Network Example

$$x = 1 \quad y = 5$$



$$w^{[1]} = 2, b^{[1]} = 0$$

ReLU activation

$$w^{[2]} = 3, b^{[2]} = 1$$

$$g(z) = \max(0, z)$$

$$a^{[1]} = g(w^{[1]} x + b^{[1]}) = w^{[1]} x + b^{[1]} = \cancel{2 \times 1 + 0} = \cancel{2}$$

$$a^{[2]} = g(w^{[2]} a^{[1]} + b^{[2]}) = w^{[2]} a^{[1]} + b^{[2]} = 3 \times \cancel{2} + 1 = \cancel{7.003}$$

$$J(w, b) = \frac{1}{2} (a^{[2]} - y)^2 = \frac{1}{2} (\cancel{7.003} - 5)^2 = \frac{(2.003)^2}{2} = 2.006005$$

$$w^{[1]} \uparrow 0.001 \quad J \uparrow 6 \times 0.001 \quad \frac{\partial J}{\partial w^{[1]}} = 6$$

$$\frac{\partial J}{\partial w^{[1]}}$$

$$\frac{\partial J}{\partial b^{[1]}}$$

N nodes  $\square \rightarrow \square \rightarrow \square$

inefficient way

$$N \times P$$

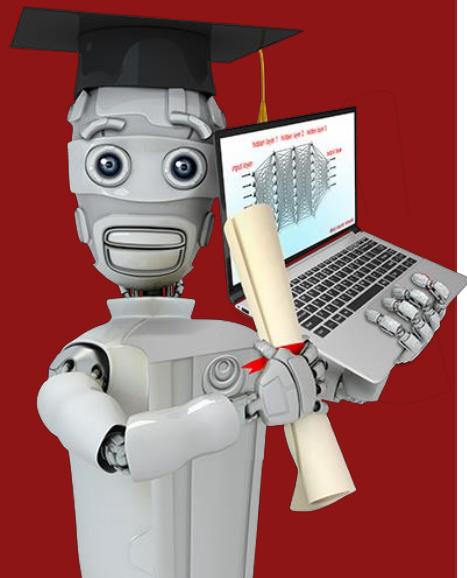
$$\frac{\partial J}{\partial w^{[2]}}$$

$$\frac{\partial J}{\partial b^{[2]}}$$

P parameters  
 $w_1, b_1, w_2, b_2 \dots$

efficient way (backprop)

$$N + P$$



## 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?

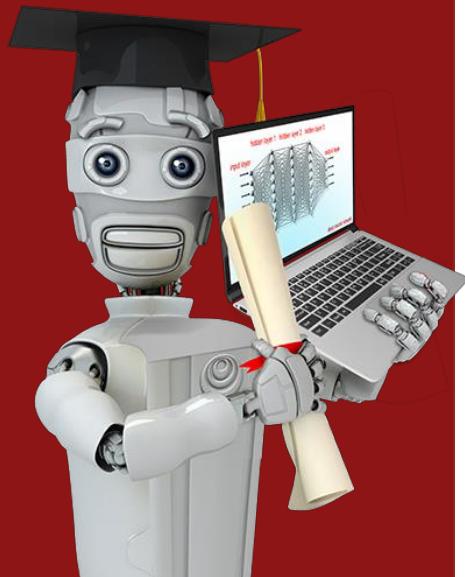
- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features ( $x_1^2, x_2^2, x_1x_2, etc$ )
- Try decreasing  $\lambda$
- Try increasing  $\lambda$

# 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.

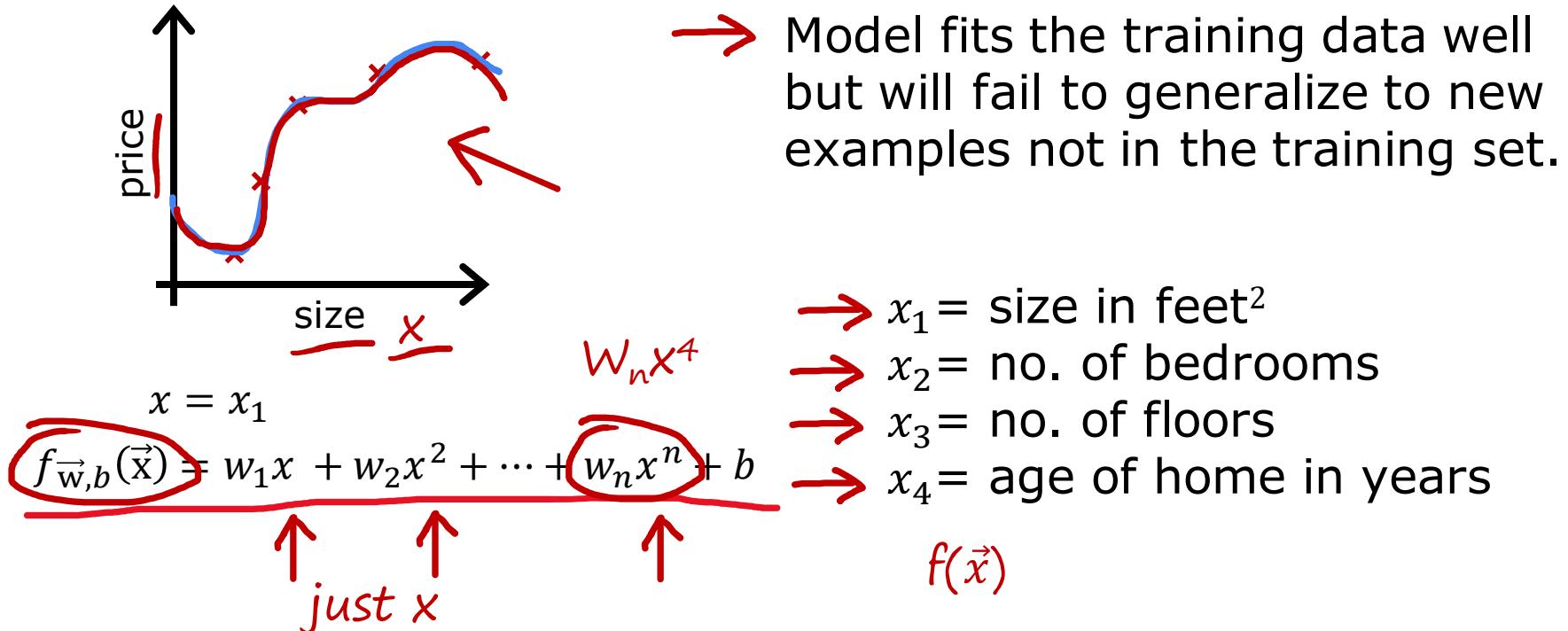
Diagnostics can take time to implement but doing so can be a very good use of your time.

## 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:

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

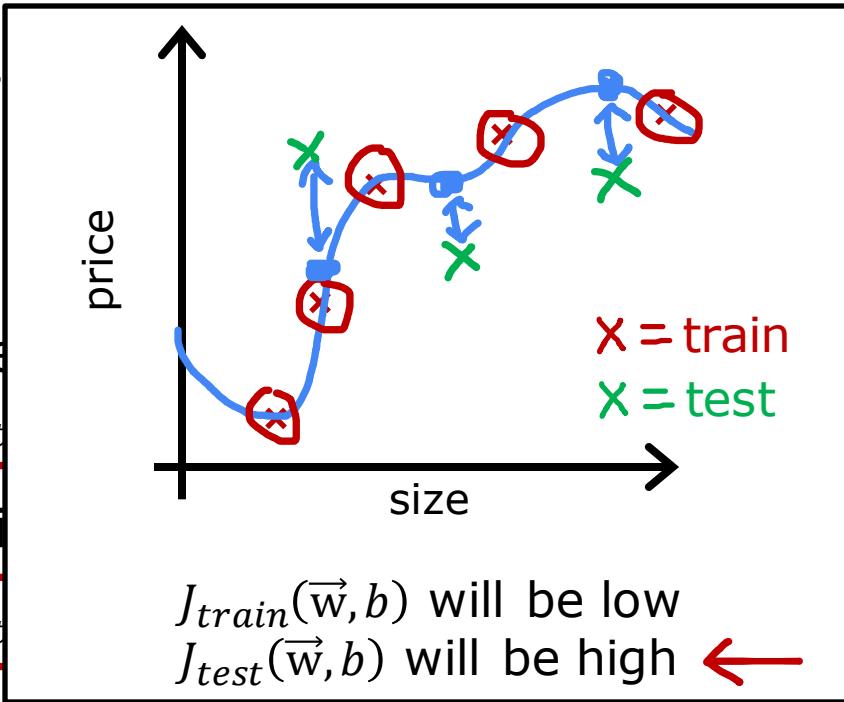
# Train/test procedure for linear regression (with squared error cost)

Fit parameters

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

Compute test error

Compute training error



$b)$

$$ain \sum_{j=1}^n w_j^2]$$

$x = \text{train}$   
 $\times = \text{test}$

$$(i) \left[ est \right]^2$$

$$- y_{train}^{(i)} \left[ \right]^2$$

$$+ \cancel{\left[ \right]} \cancel{\left[ \right]}$$

# 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

0 / /

Fit parameters by minimizing  $J(\vec{w}, b)$  to find  $\vec{w}, b$

E.g.,

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

## Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} [y_i$$

## Compute train error:

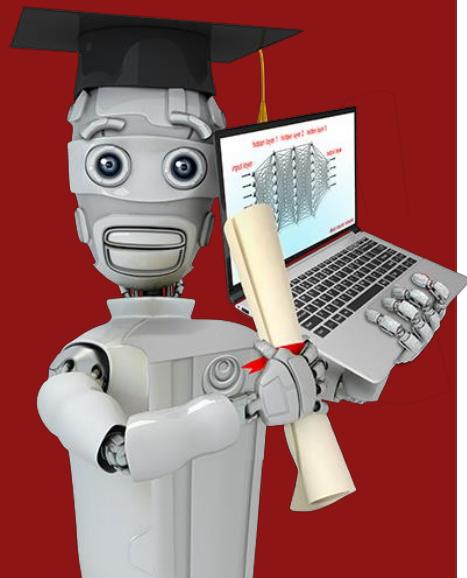
$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}}$$

fraction of the test set and the fraction of the train set that the algorithm has misclassified.

$$\hat{y} = \begin{cases} 1 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

$J_{test}(\vec{w}, b)$  is the fraction of the test set that has been misclassified.

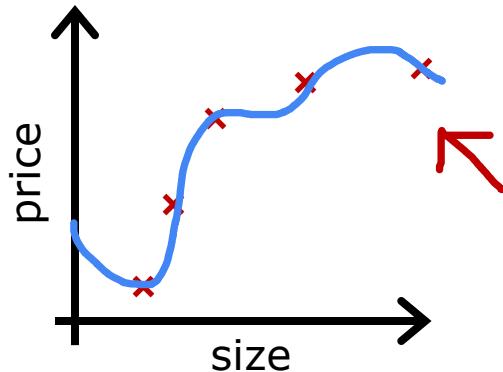
$J_{train}(\vec{w}, b)$  is the fraction of the train set that has been misclassified.



## 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)

$d=1$

$$1. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + b$$

$d=2$

$$2. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + b$$

$d=3$

$$3. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + b$$

$\vdots$

$d=10$

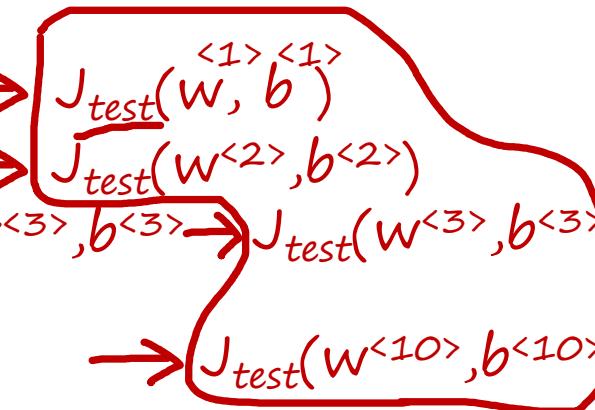
$$10. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + \dots + w_{10} x^{10} + b$$

$$\xrightarrow{w, b^{<1>} b^{<1>}}$$

$$\xrightarrow{w, b^{<2>} b^{<2>}}$$

$$\xrightarrow{w^{<3>} b^{<3>}}$$

$$\xrightarrow{w^{<10>} b^{<10>}}$$



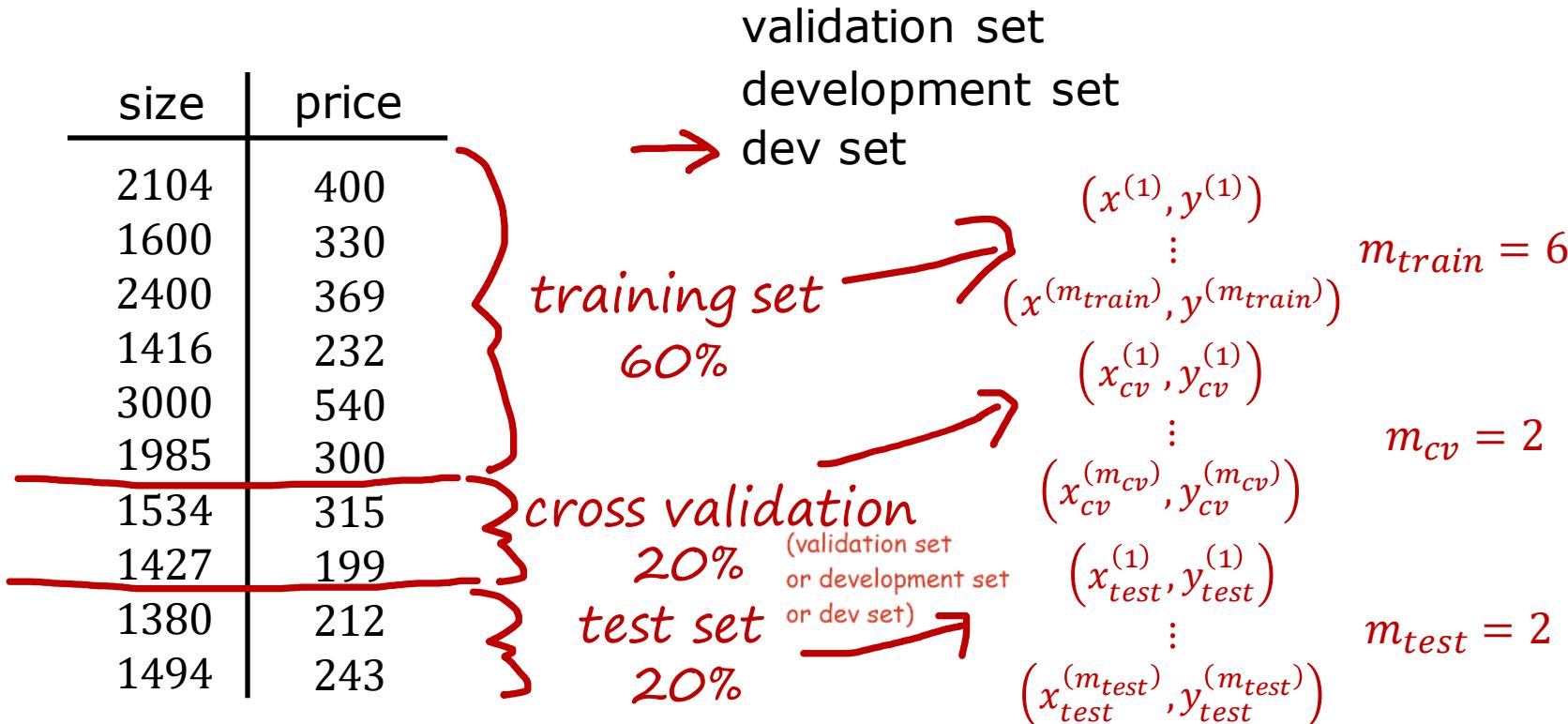
Choose  $w_1 x_1 + \dots + w_5 x^5 + b$   $d=5$

$J_{test}(w^{<5>} b^{<5>})$  is lowest so we choose this

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.

# 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

$d=1$

$$1. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + b$$

$d=2$

$$2. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + b$$

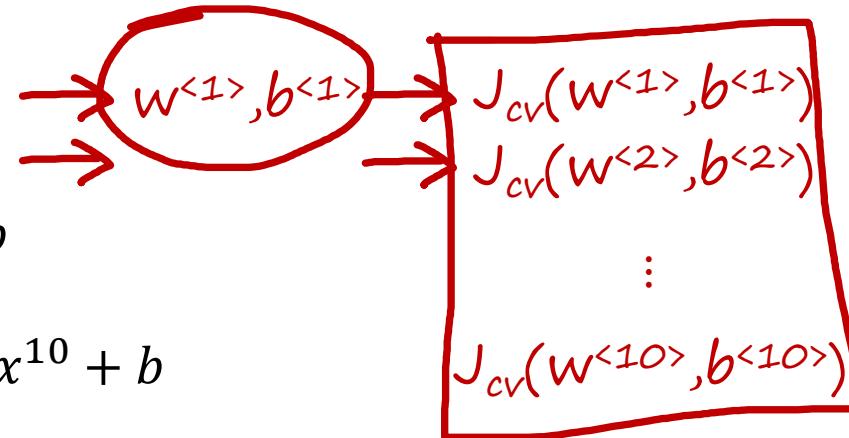
$d=3$

$$3. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + b$$

:

$d=10$

$$10. f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + w_2 x^2 + \dots + w_{10} x^{10} + b$$



→ Pick  $w_1 x_1 + \dots + w_4 x^4 + b$

$(J_{cv}(w^{<4>}, b^{<4>}))$

Estimate generalization error using test the set:  $J_{test}(w^{<4>}, b^{<4>})$

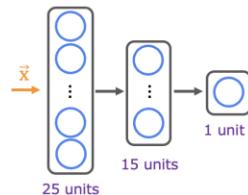
train → training set

degree of poly → cv set

generalization error → test set

# Model selection – choosing a neural network architecture

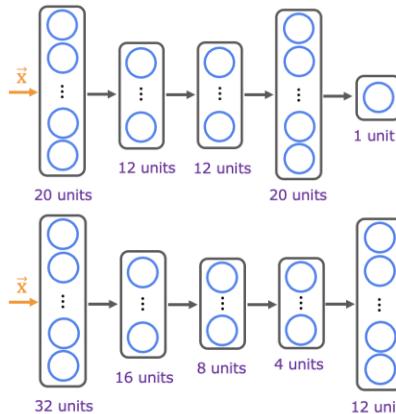
1.



$$w^{(1)}, b^{(1)}$$

$$J_{cv}(\mathbf{W}^{(1)}, \mathbf{B}^{(1)})$$

→ 2.

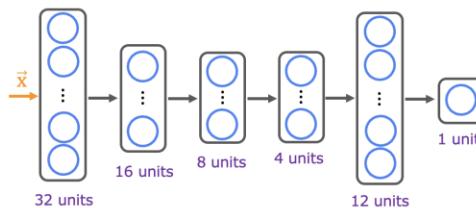


$$w^{(2)}, b^{(2)}$$

$$J_{cv}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})$$

is the one with less cv error

3.



$$w^{(3)}, b^{(3)}$$

$$J_{cv}(\mathbf{W}^{(3)}, \mathbf{B}^{(3)})$$

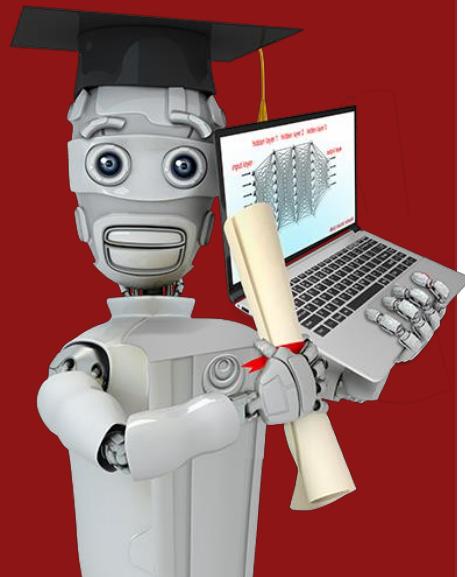
Pick  $\mathbf{W}^{(2)}, \mathbf{B}^{(2)}$

\* since we haven't fit anything using test set, then test set is a fair estimate of generalization error

Estimate generalization error using the test set:

Train, CV to fit

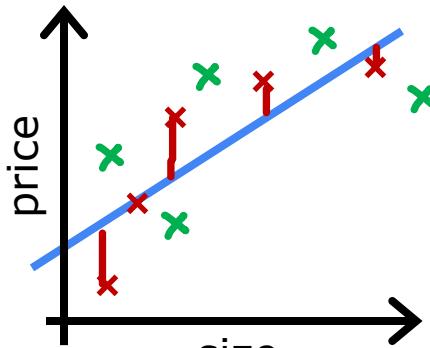
$$J_{test}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})$$



## Bias and variance

Diagnosing bias and variance

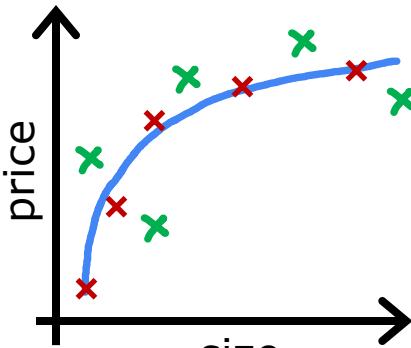
# Bias/variance



$$f_{\vec{w},b}(x) = w_1x + b$$

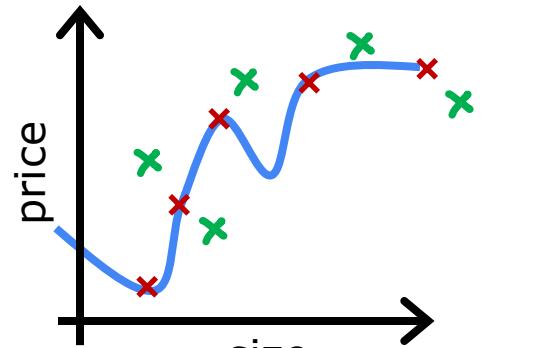
→ High bias  
(underfit)

$d = 1$   $J_{train}$  is high  
 $J_{cv}$  is high



$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$$

"Just right"



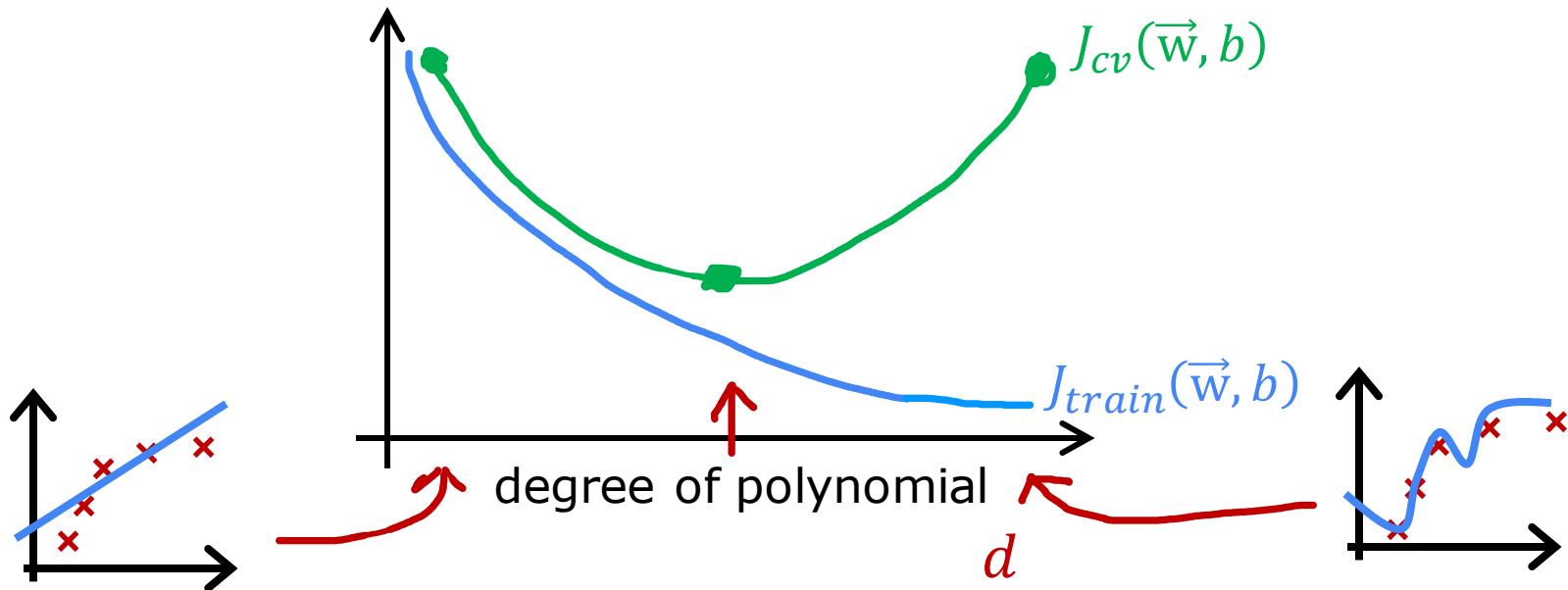
$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

High variance  
(overfit)

$d = 4$   $J_{train}$  is low  
 $J_{cv}$  is high

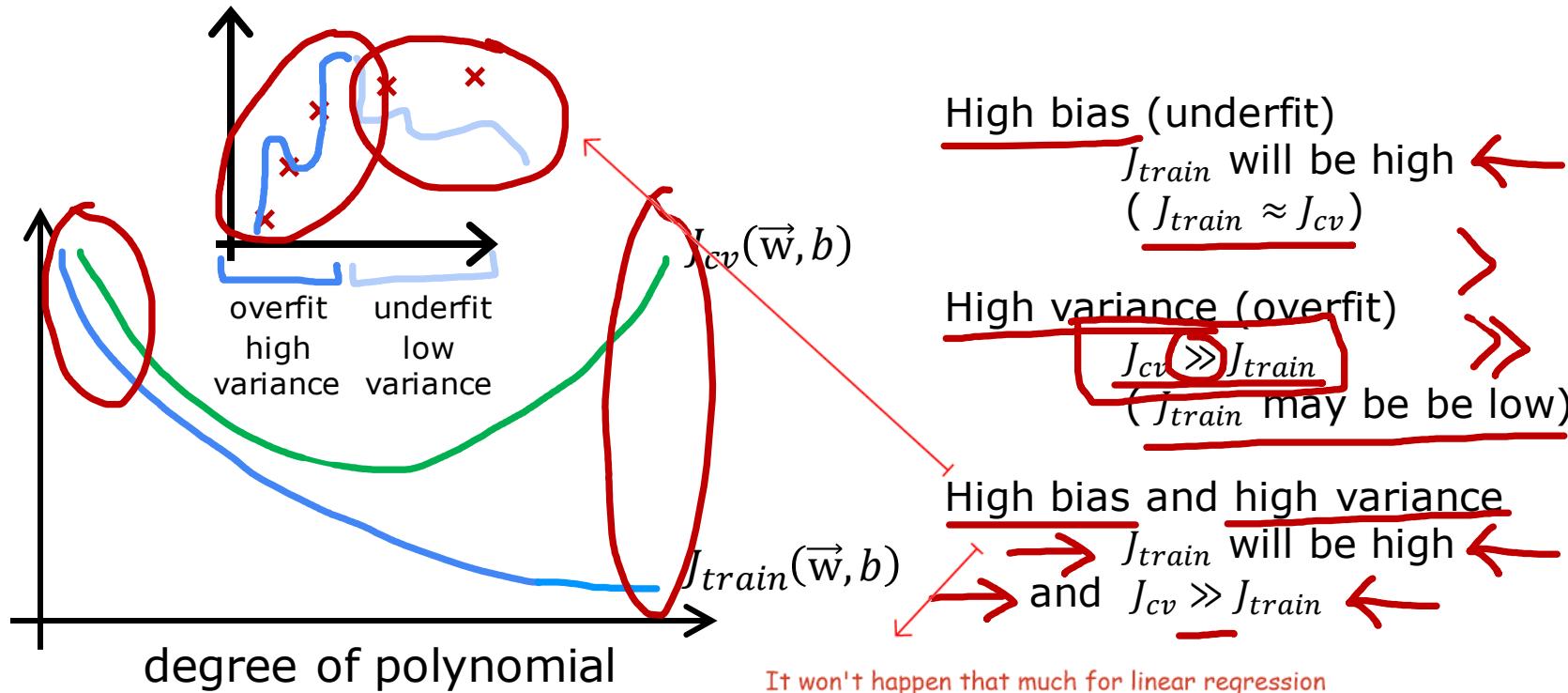
we can't always plot the data if i has more than 1 feature. So by computing  $J_{train}$  and  $J_{cv}$  we would know if it has High bias or High variance

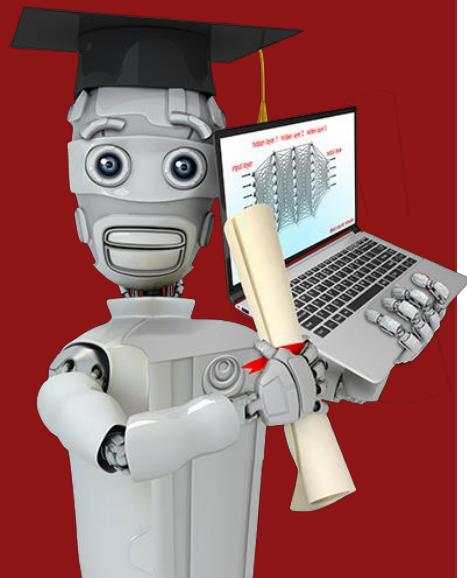
# Understanding bias and variance



# Diagnosing bias and variance

How do you tell if your algorithm has a bias or variance problem?





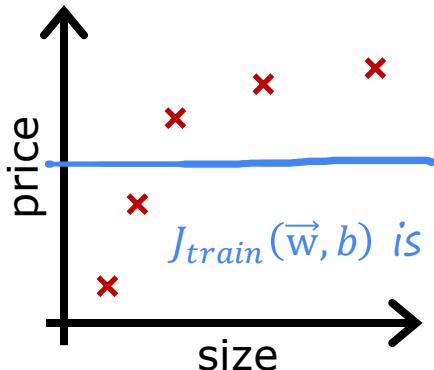
## Bias and variance

Regularization and  
bias/variance

# Linear regression with regularization

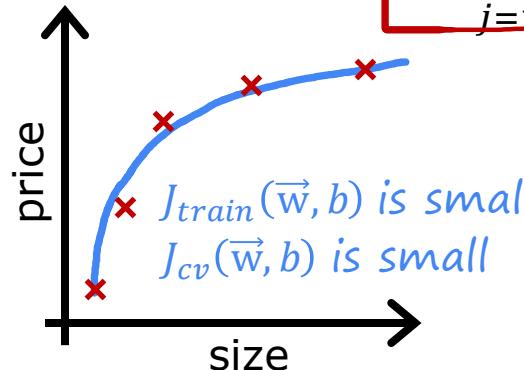
Model:  $f_{\vec{w}, b}(x) = \underbrace{w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b}_m$

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



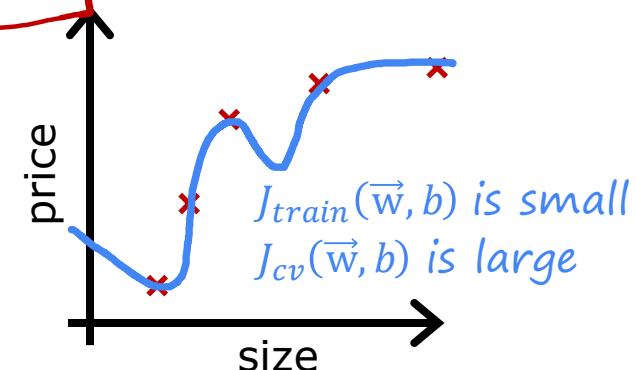
Large  $\lambda$   
High bias (underfit)

$$\lambda = 10,000 \quad w_1 \approx 0, w_2 \approx 0$$
$$f_{\vec{w}, b}(\vec{x}) \approx b$$



Intermediate  $\lambda$

$$\lambda$$

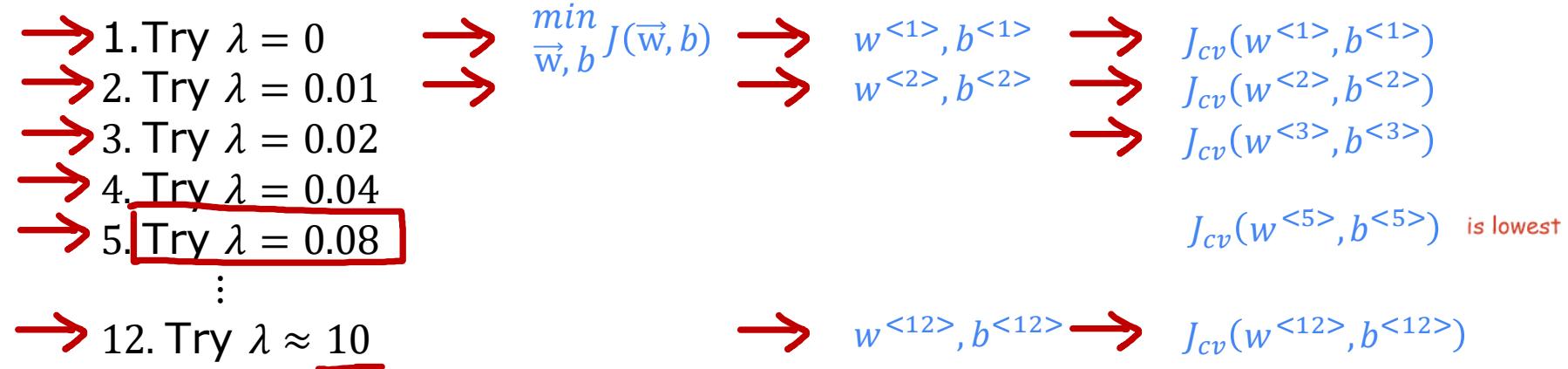


Small  $\lambda$   
High variance (overfit)

$$\lambda = 0$$

# Choosing the regularization parameter $\lambda$

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

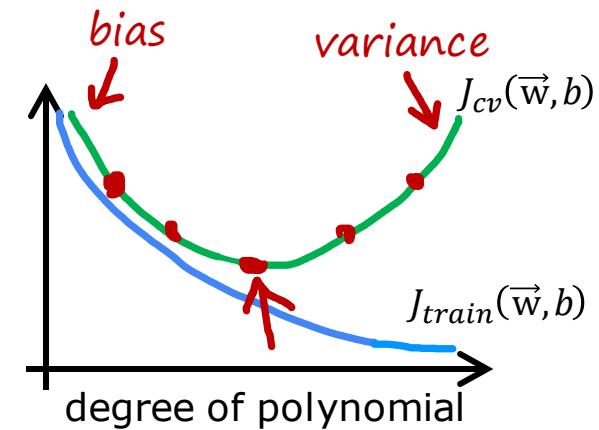
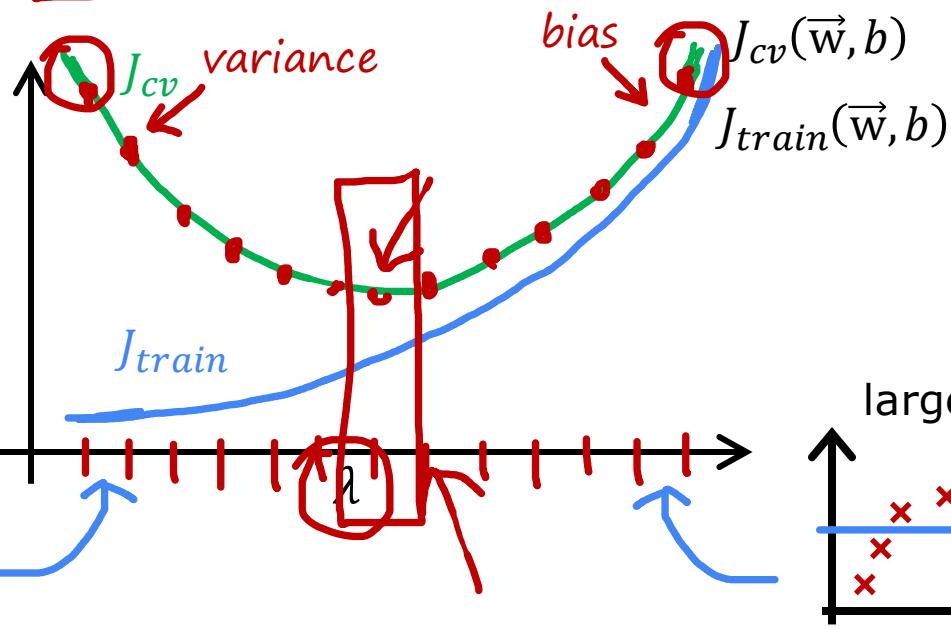


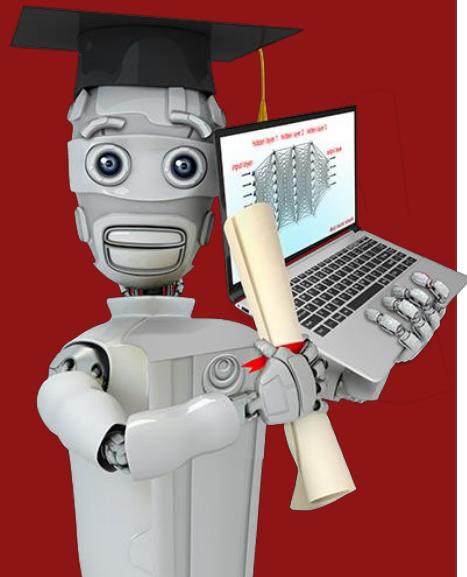
Pick  $w^{<5>}, b^{<5>}$

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% 4.0%

Training error  $J_{train}$

: 10.8%

Cross validation error  $J_{cv}$

: 14.8%



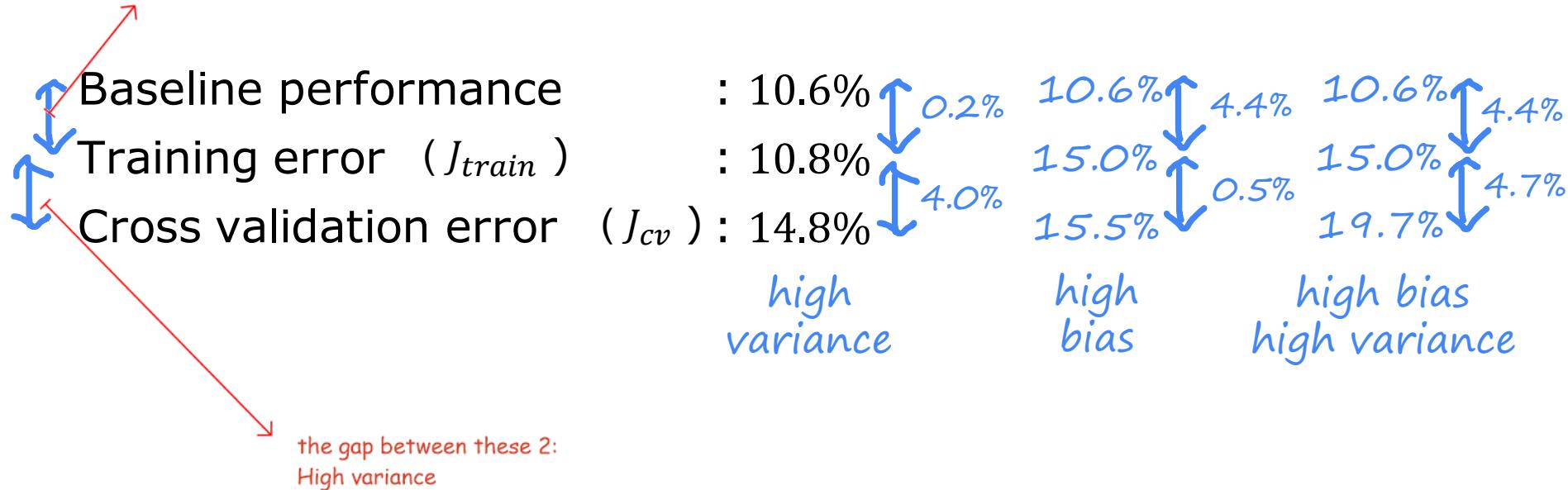
# Establishing a baseline level of performance

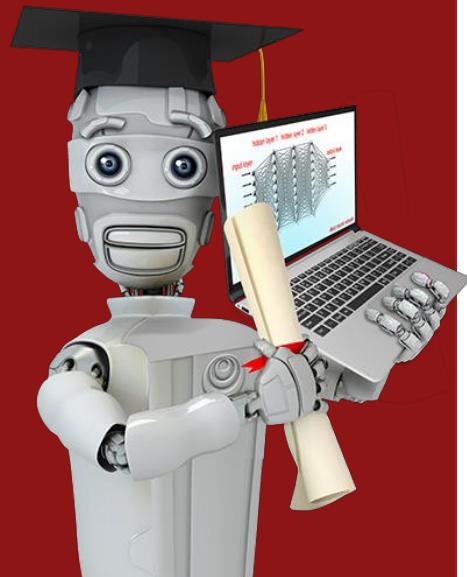
What is the level of error you can reasonably hope to get to?

- Human level performance
- Competing algorithms performance
- Guess based on experience

the gap between these  
2: High Bias

# Bias/variance examples

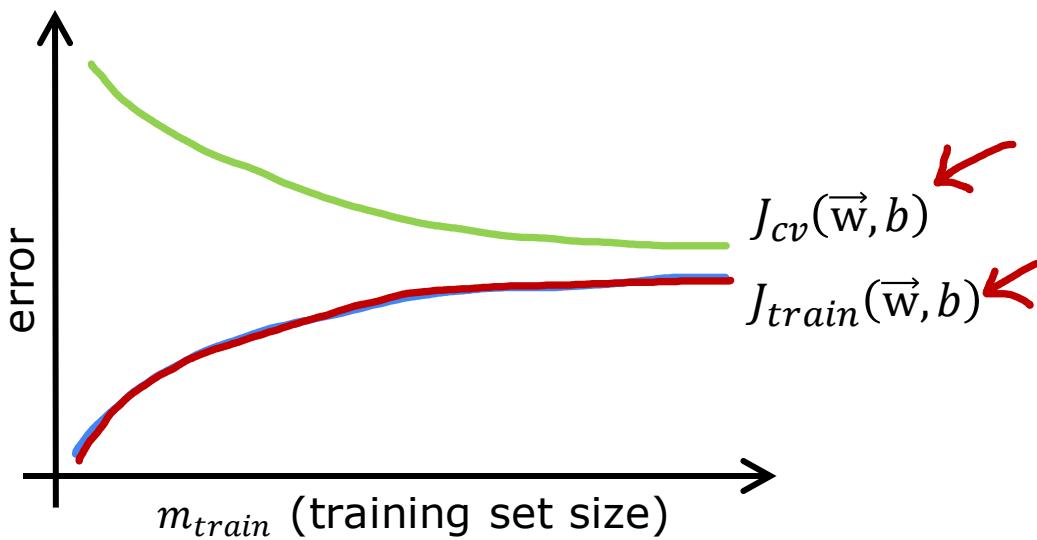




## Bias and variance

# Learning curves

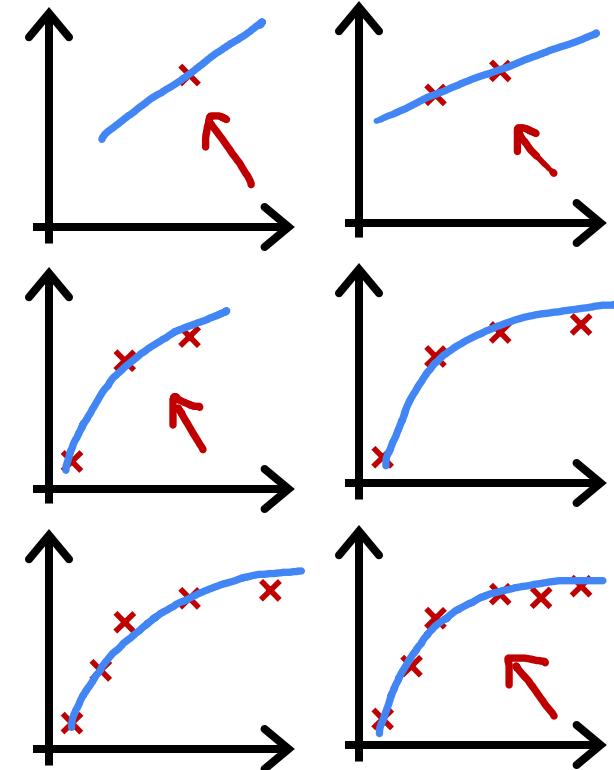
# Learning curves



$J_{train}$  = training error

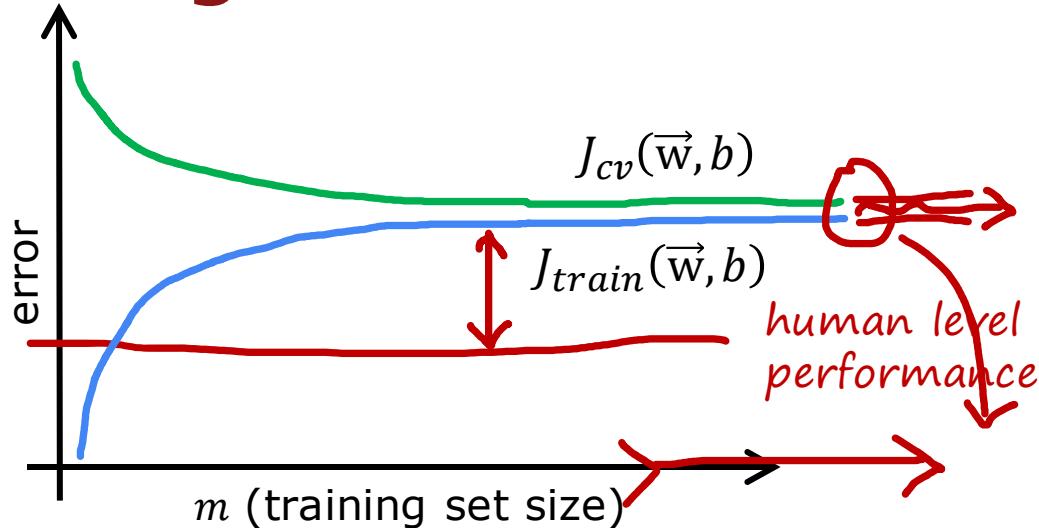
$J_{cv}$  = cross validation error

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$

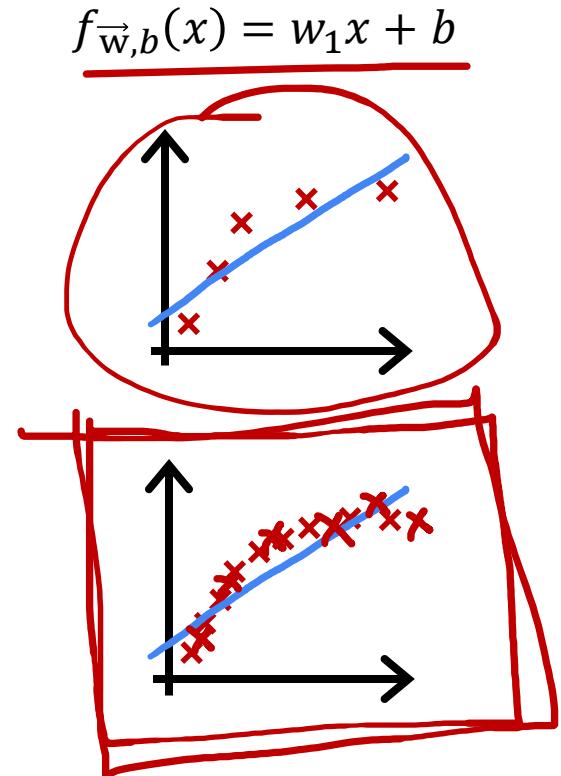


# High bias

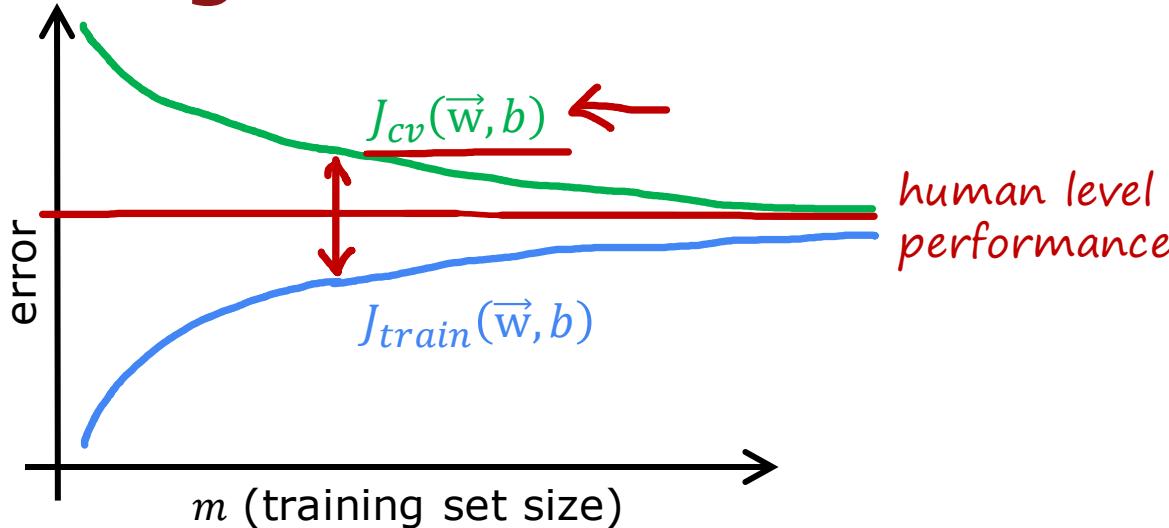
error = average training error



if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

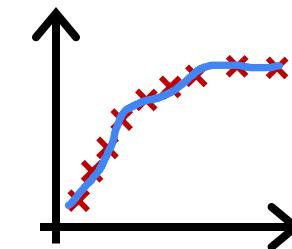
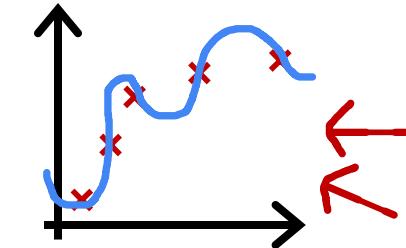


# High variance



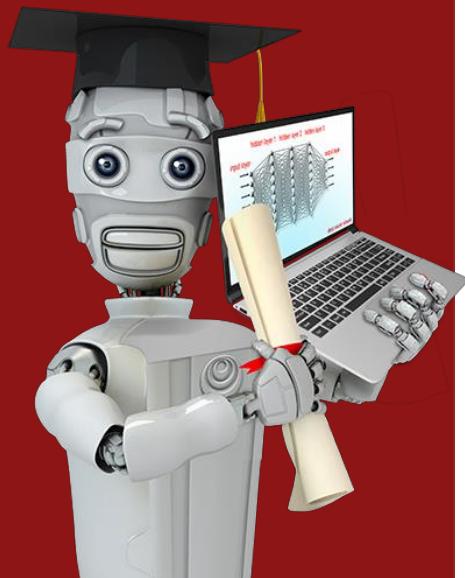
if a learning algorithm suffers from high variance, getting more training data is likely to help.

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b \quad (\text{with small } \lambda)$$



## Bias and variance

Deciding what to try next  
revisited



# Debugging a learning algorithm

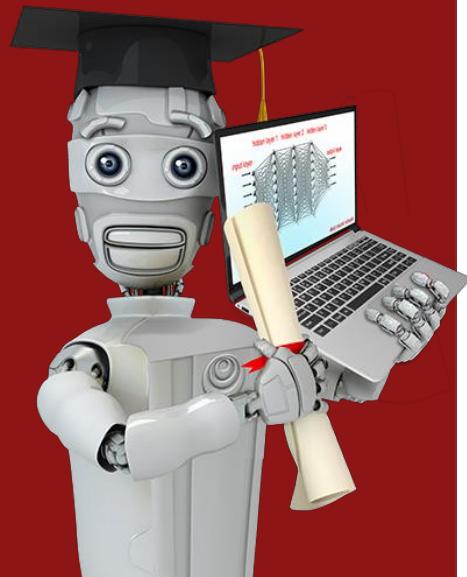
You've implemented regularized linear regression on housing prices

\* reducing training size is not  
going help with high bias. it will  
lower  $J_{train}$  but  $J_{cv}$  would  
get higher

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

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

- Get more training examples 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

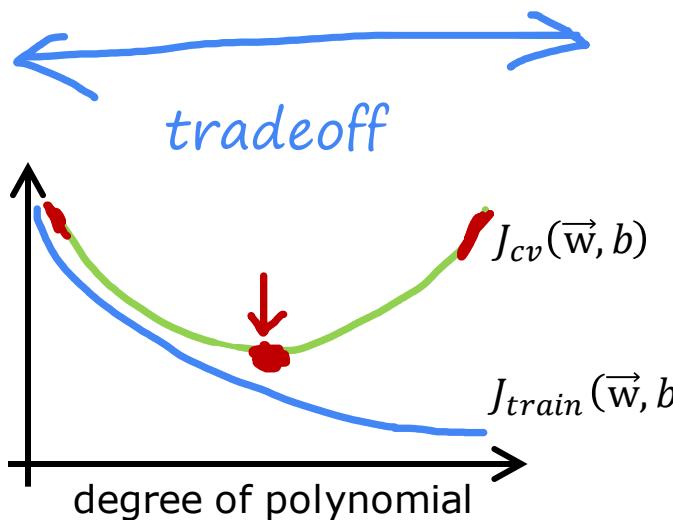
High bias

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$

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

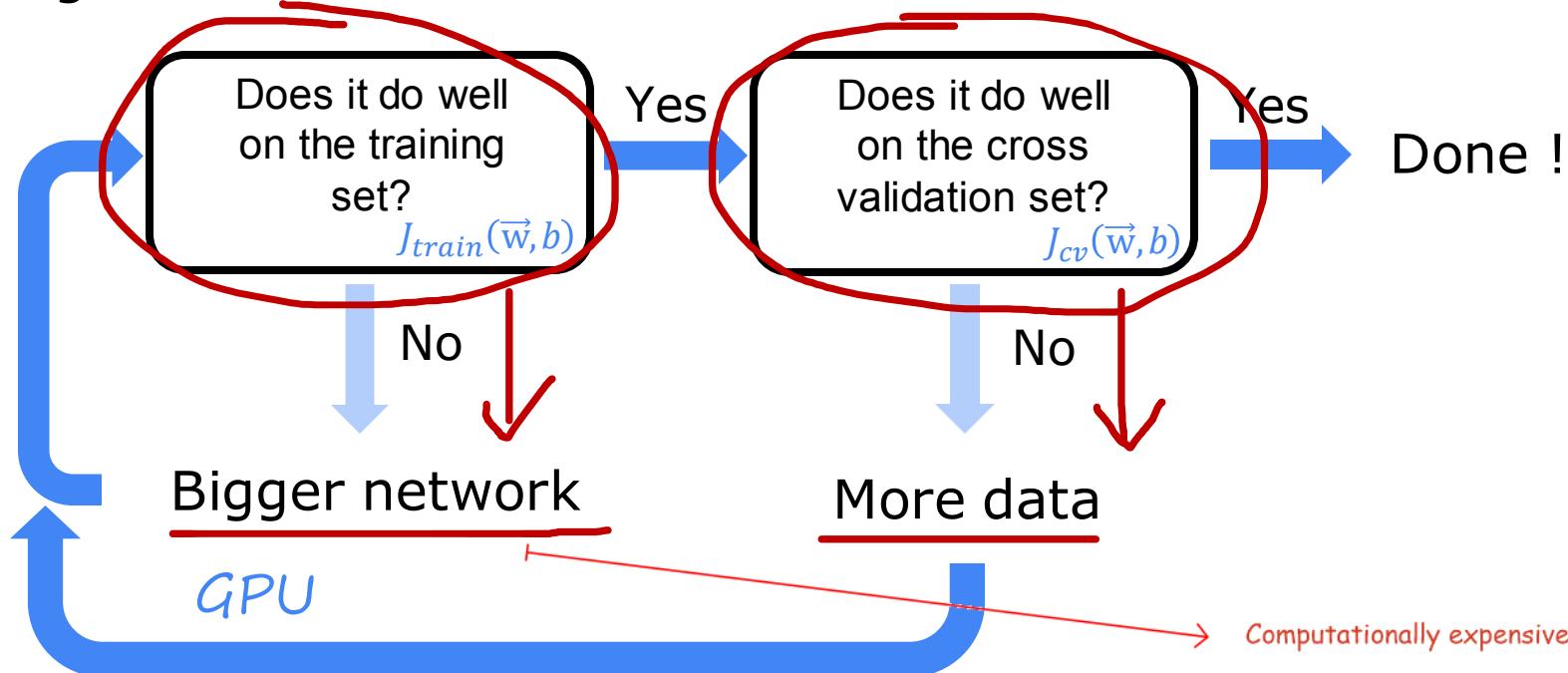
Complex model

High variance

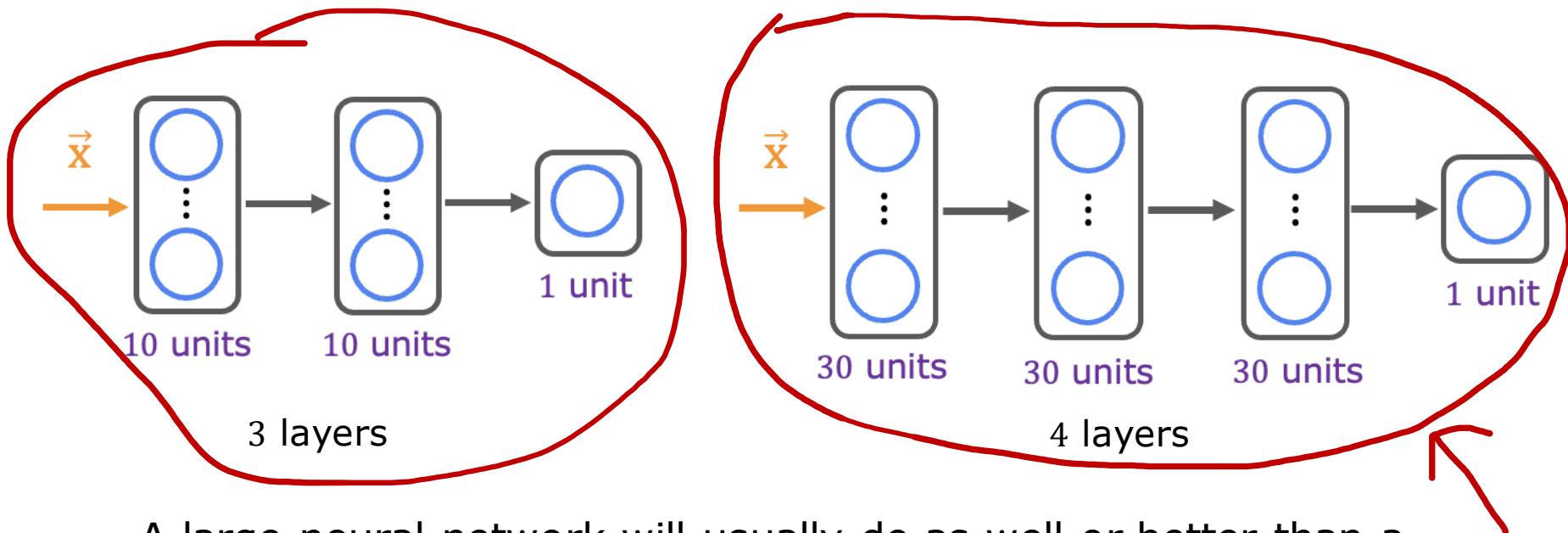


# Neural networks and bias variance

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

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^m L(f(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{all\ weights\ \mathbf{W}} (w^2)$$

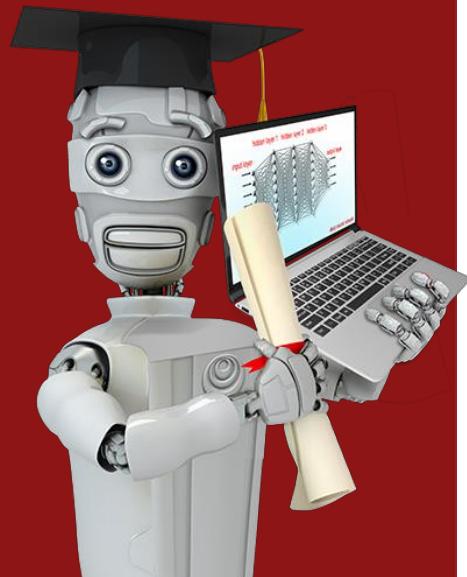
## Unregularized MNIST model

```
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

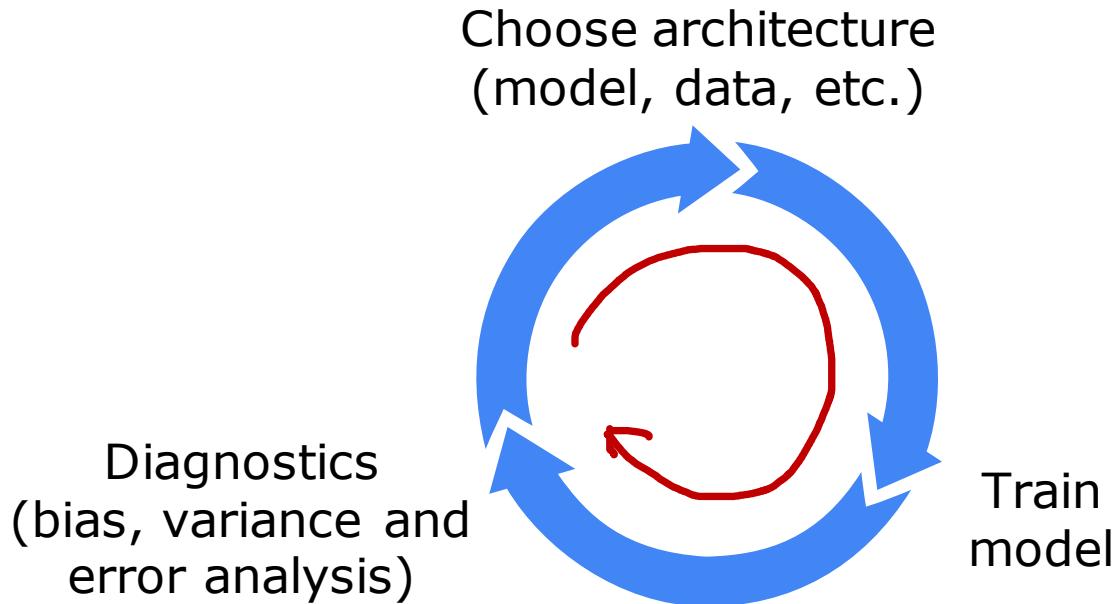
```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

# 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.

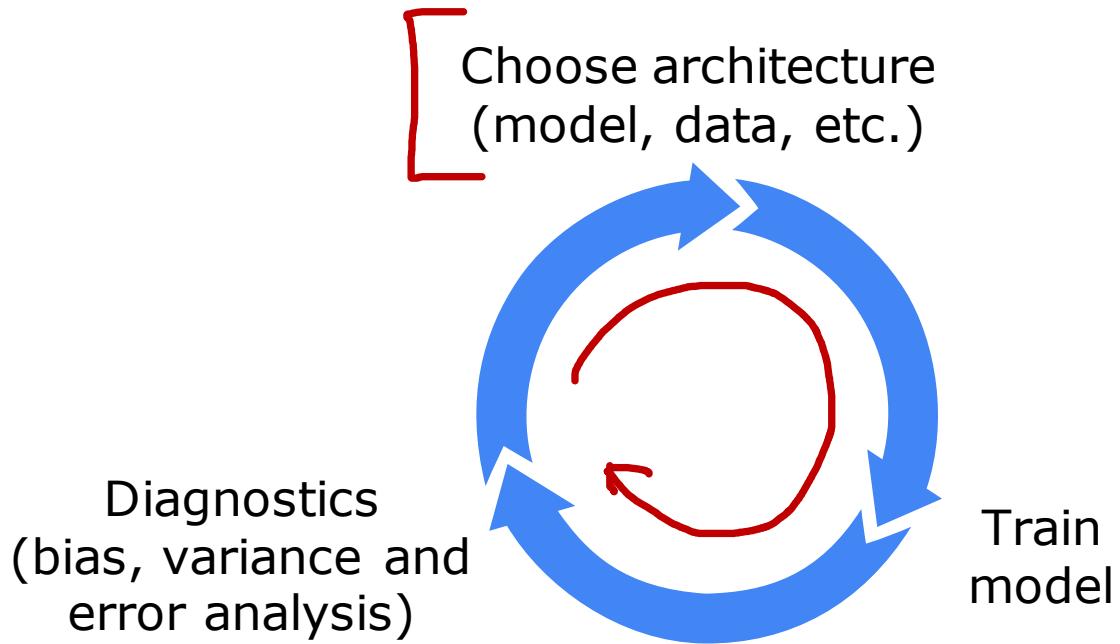
# Building a spam classifier

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.

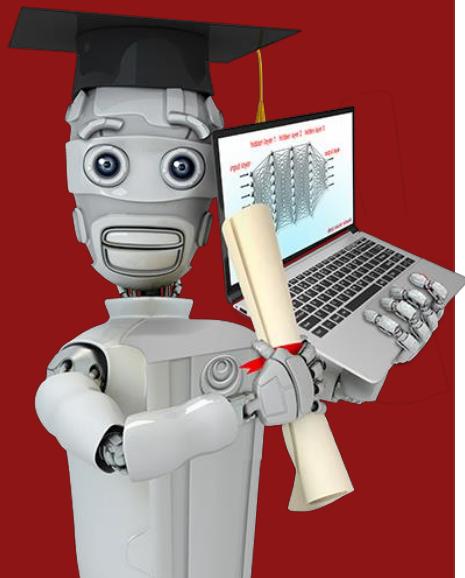


# Iterative loop of ML development



# 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.

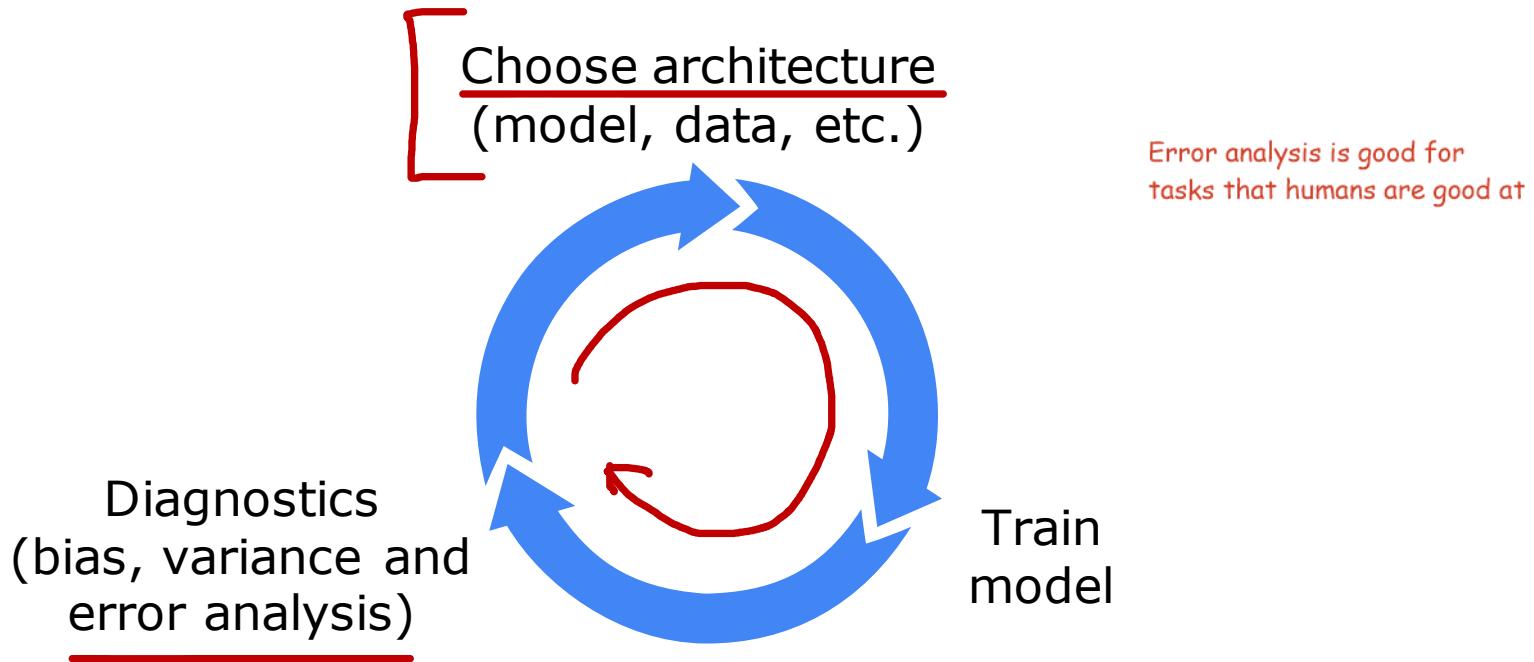
- Pharma: 21 more data features  
(especially data about pharma)
- Deliberate misspellings (w4tches, med1cine): 3
- Unusual email routing: 7
- Steal passwords (phishing): 18 more data features
- Spam message in embedded image: 5

# Building a spam classifier

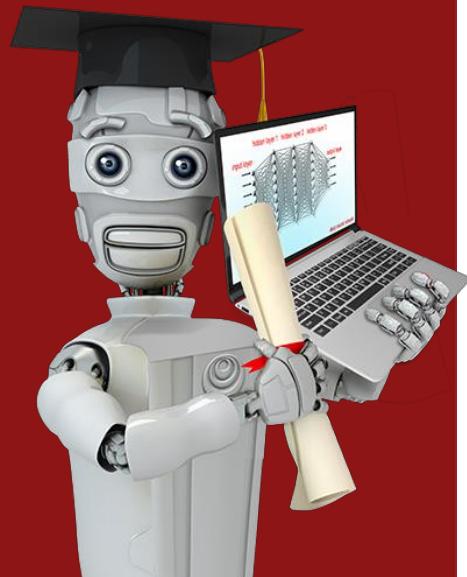
How to try to reduce your spam classifier's error?

- 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.
- 
- bias/var analysis
- error analysis
- error analysis (base on last slide this could help alot)
- error analysis

# 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.

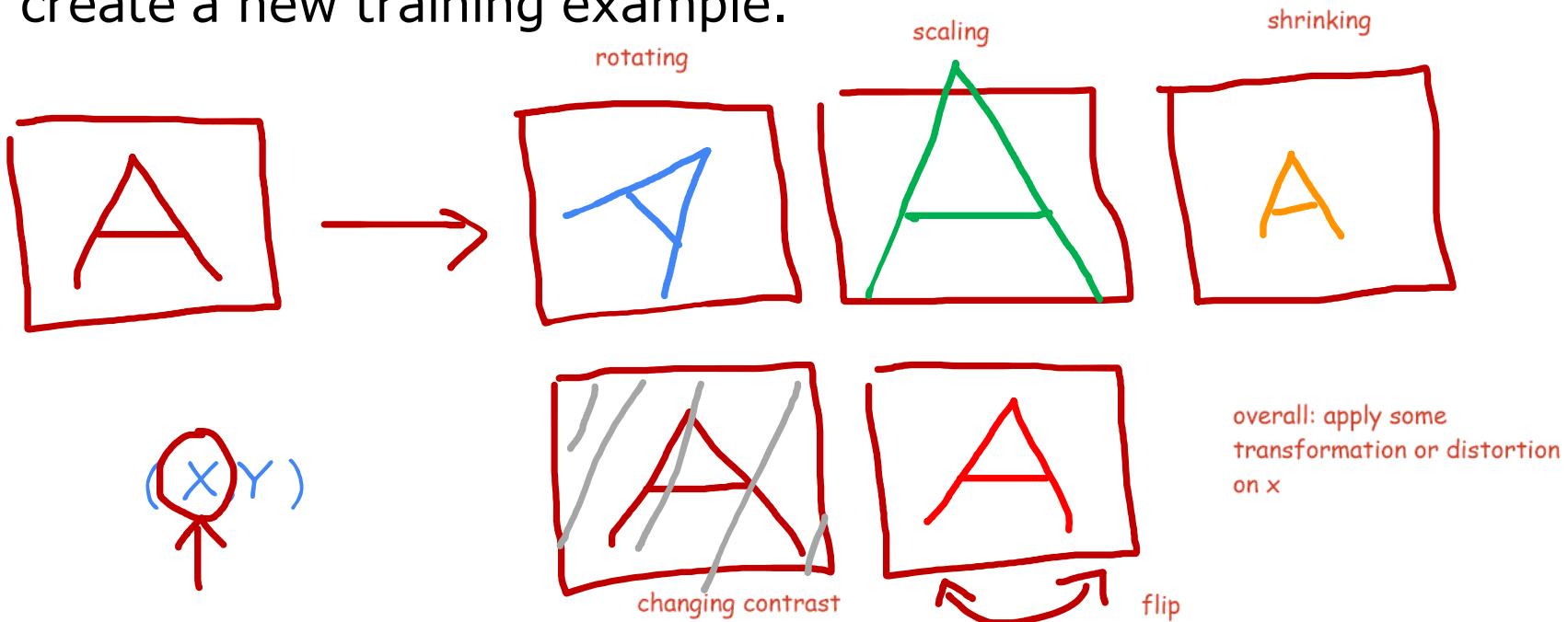
*Pharma spam*

E.g., Go to unlabeled data and find more examples of Pharma related spam.

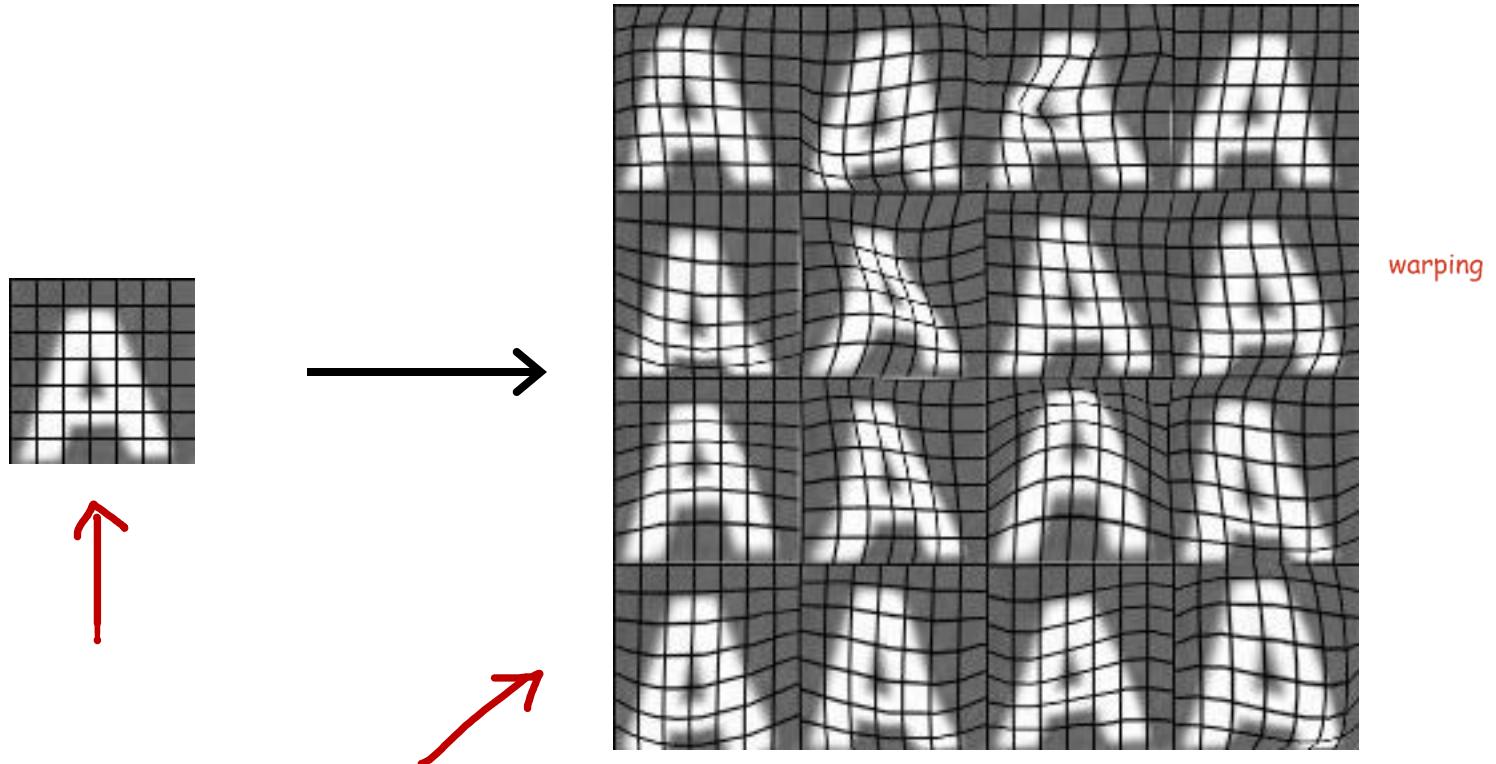
( X,Y )

# Data augmentation

Augmentation: modifying an existing training example to create a new training example.

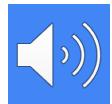


# Data augmentation by introducing distortions



# Data augmentation for speech

Speech recognition example



Original audio (voice search: "What is today's weather?")



+ Noisy background: Crowd



+ Noisy background: Car

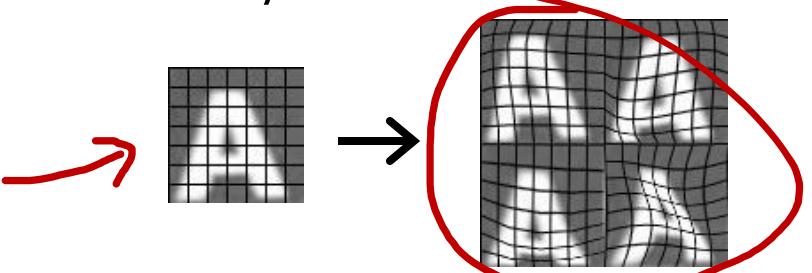


+ Audio on bad cellphone connection

# Data augmentation by introducing distortions

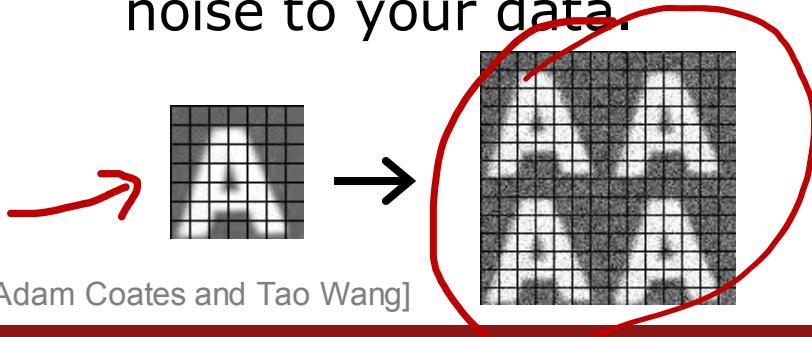
Distortion introduced should be representation of the type of noise/distortions in the test set.

(to be like what you expect to see in 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  
bc  
def  
g

# Artificial data synthesis for photo OCR



Real data



(used most for computer vision tasks)

Synthetic data

(using fonts on computer)

[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:

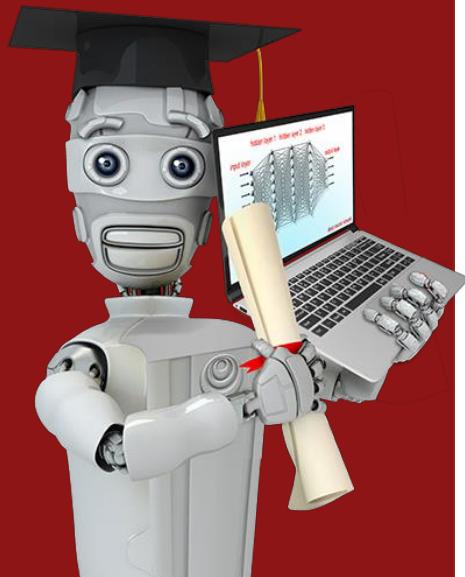
$$\text{AI} = \text{Code} + \text{Data}$$

(algorithm/model)



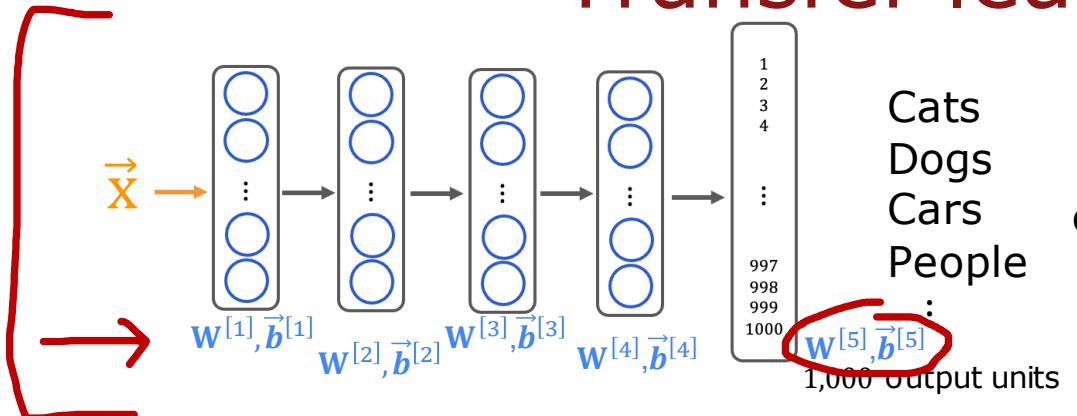
Work on this

# Machine learning development process



**Transfer learning: using data  
from a different task**

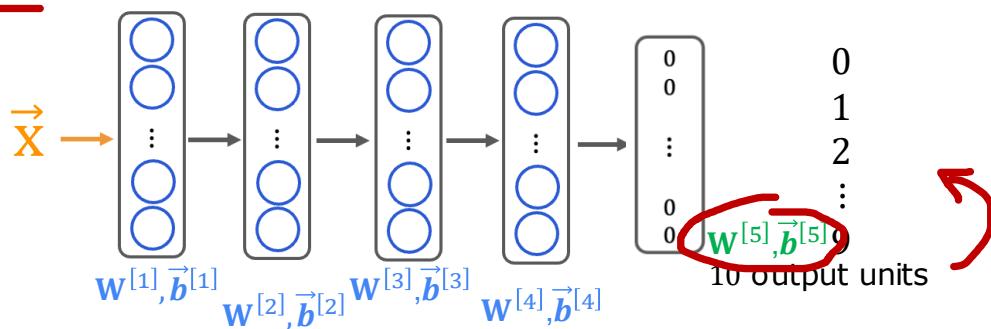
# Transfer learning



0, 1, 2, ..., 9  
1 million images

we can download  
pre-trained models from  
other people as well

Supervised pretraining



Fine tuning

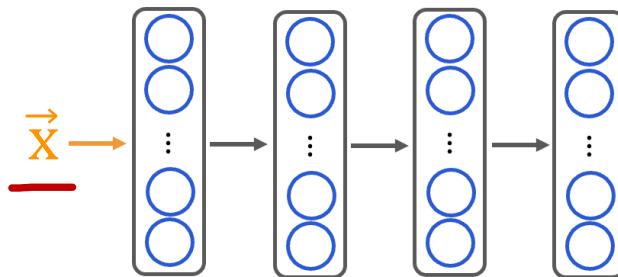
Train the upper model and  
copy its parameter ( $w[i], b[i]$ )  
to our model (the lower one)  
and then choose one of these  
options:

Option 1: only train **output layers** parameters.

good for small  
training set

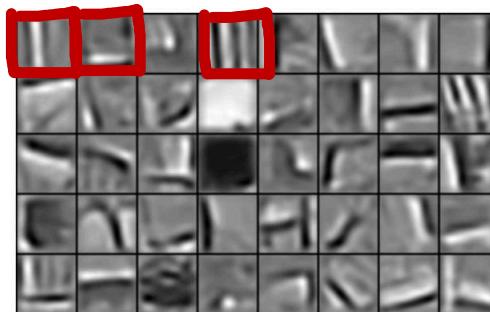
Option 2: **train all parameters.** good for large training set

# Why does transfer learning work?

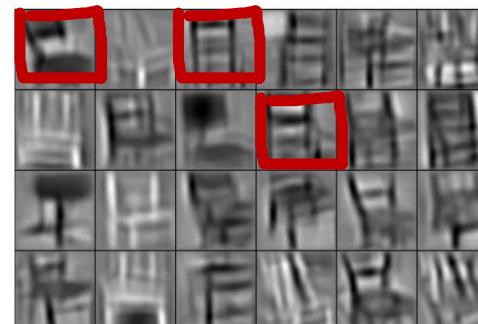


detects  
edges      detects  
corners      detects  
curves/basic shapes

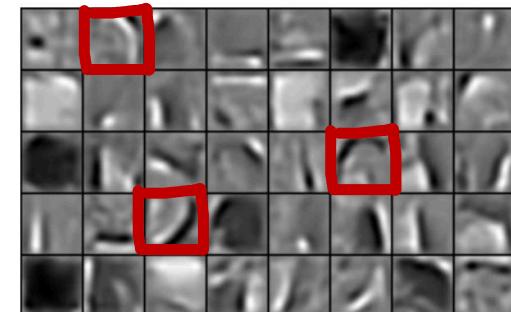
use the same input type  
(if our work is on images, use images in pretraining)



Edges



Corners

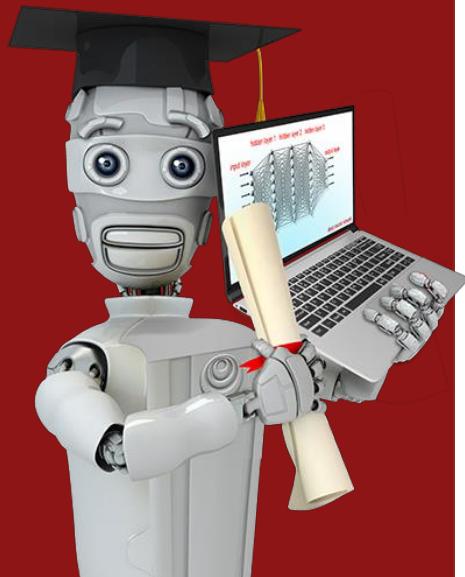


Curves / basic shapes

# Transfer learning summary

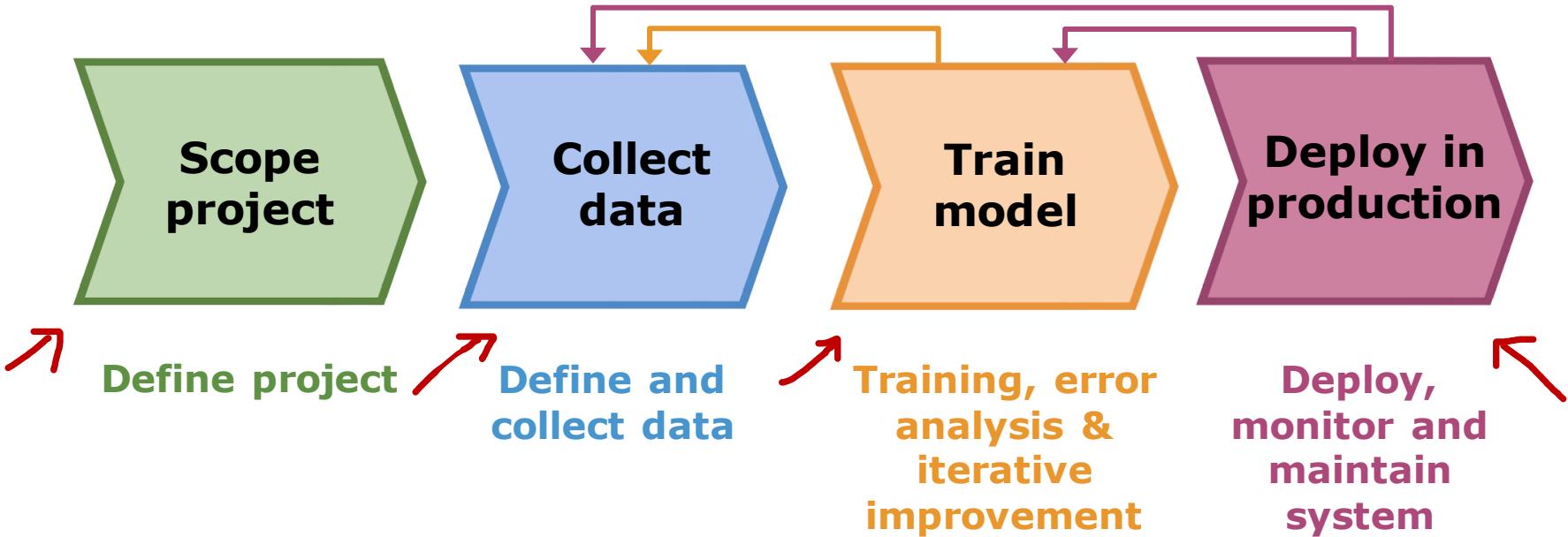
- 1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). *1M*
- 2. Further train (fine tune) the network on your own data. *1000*  
*50*

# Machine learning development process

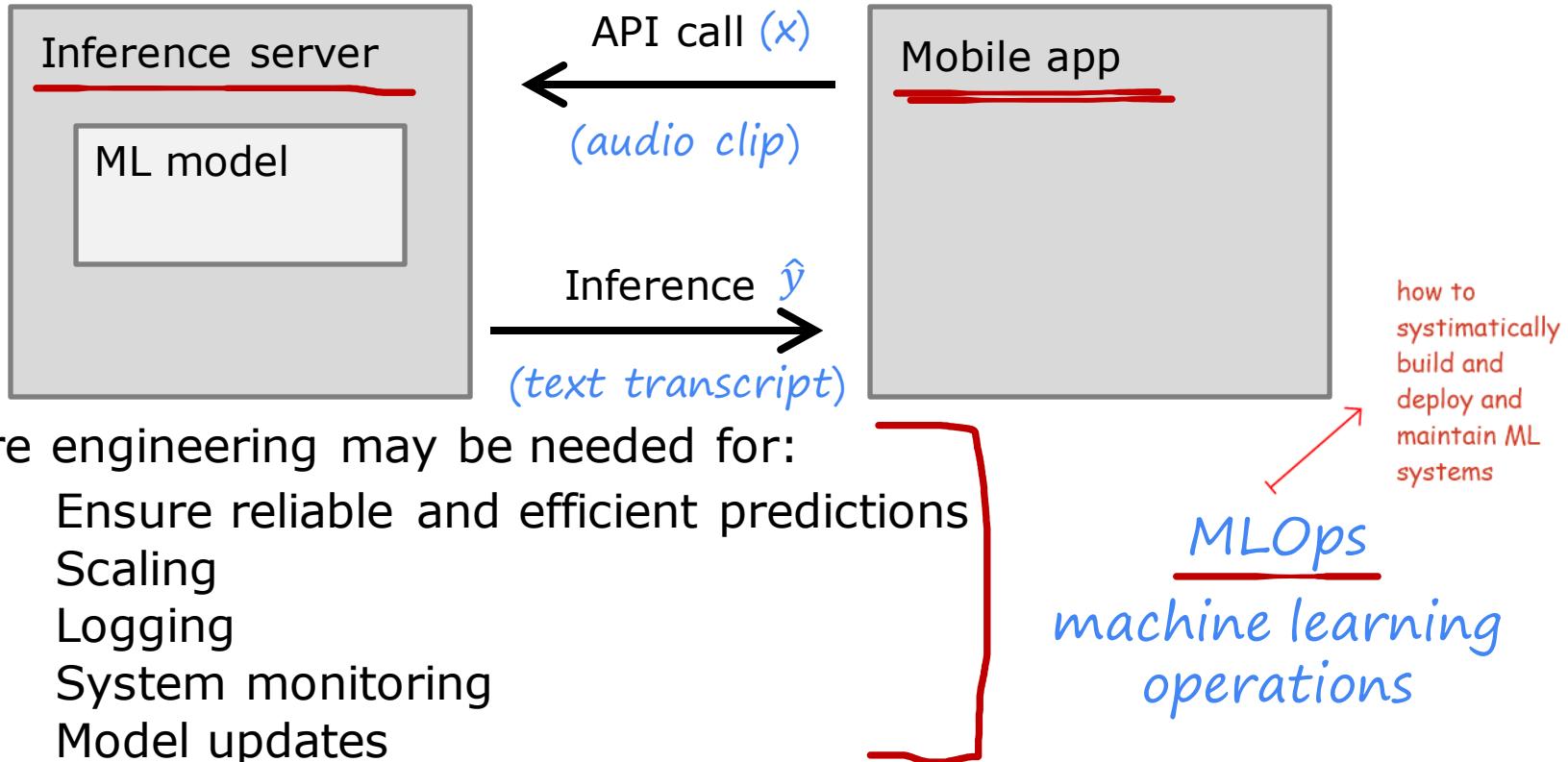


Full cycle of a  
machine learning project

# Full cycle of a machine learning project

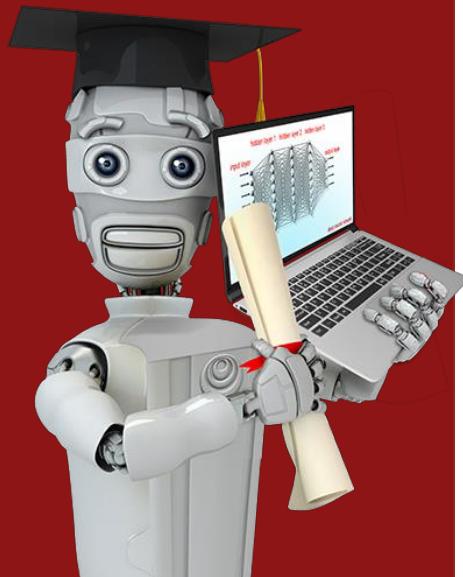


# Deployment



# 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.

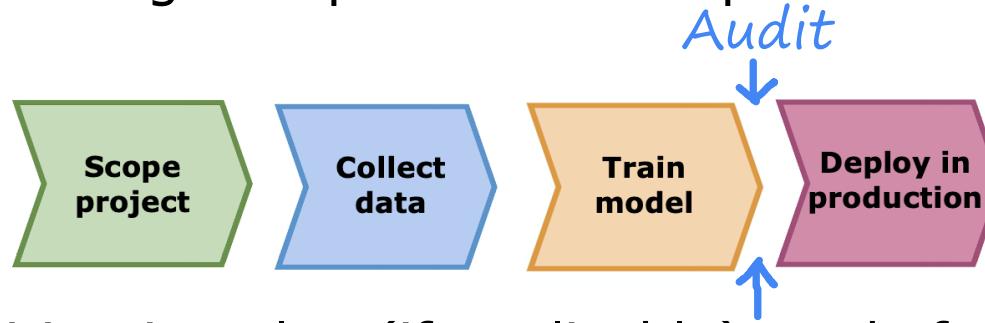
Spam vs anti-spam : fraud vs anti-fraud.

# Guidelines

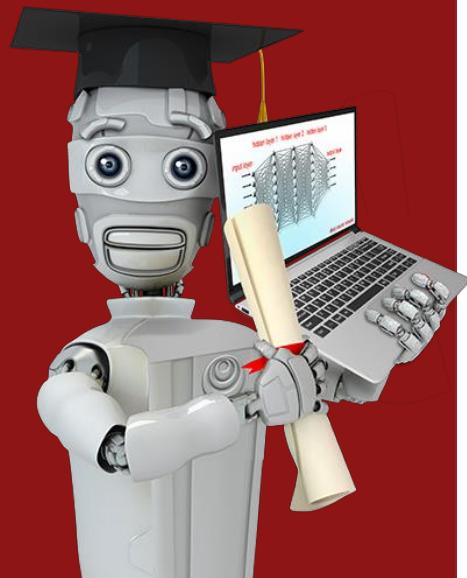
Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.

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%

any algo with Precision = 0 or  
recall = 0 is not a useful algo

# Precision/recall

recall help us to detect a learning algo  
just print(y=0) all the time or not  
what about precision? it would be 0/0

$y = 1$  in presence of rare class we want to detect.

Actual Class			
		1	0
Predict -ed Class	1	True positive 15	False positive 5
	0	False negative 10	True negative 70

confusion matrix

↓      ↓      ↓

25      75

**print("y=0")**

## Precision:

(of all patients where we predicted  $y = 1$ , what fraction actually have the rare disease?)

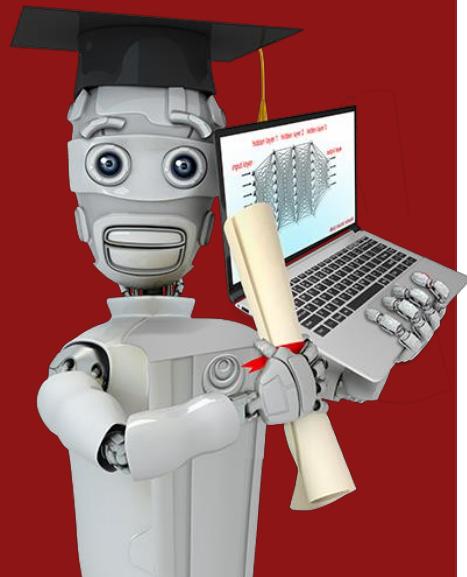
$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15 + 5} = 0.75$$

## Recall: ←

(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15 + 10} = 0.6$$

when we have a rare class (a skewed data) having high precision and recall → useful algo



## 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  $f_{\vec{w}, b}(\vec{x}) \geq 0.5$  (0.7) (0.9) 0.3)
- Predict 0 if  $f_{\vec{w}, b}(\vec{x}) < 0.5$  (0.7) (0.9) 0.3)

$$\text{precision} = \frac{\text{true positives}}{\text{total predicted positive}}$$
$$\text{recall} = \frac{\text{true positives}}{\text{total actual positive}}$$

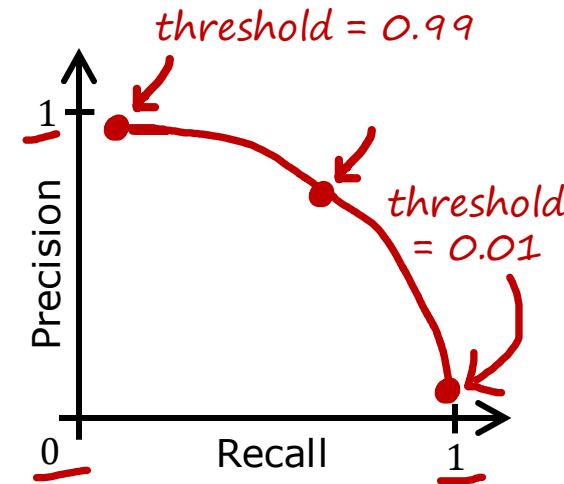
Suppose we want to predict  $y = 1$  (rare disease) only if very confident.

raising threshold → higher precision, lower recall.

Suppose we want to avoid missing too many cases of rare disease (when in doubt predict  $y = 1$ )

lowering threshold → 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?

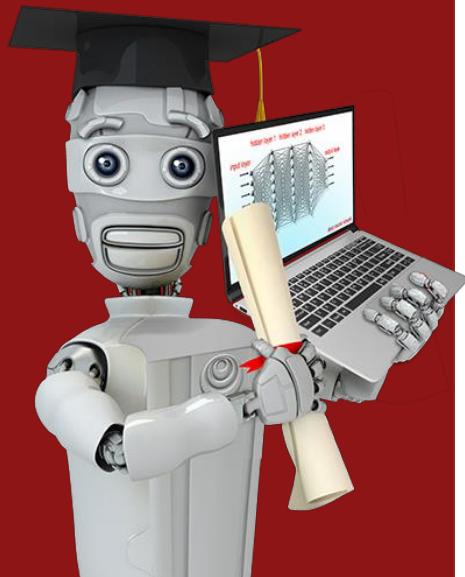
	Precision (P)	Recall (R)		$F_1$ score
Algorithm 1	0.5	0.4	<i>not a good way</i> Average 0.45 0.4 0.501	0.444
Algorithm 2	0.7	0.1		0.175
Algorithm 3	0.02	1.0		0.0392

`print("y=1")`

~~$$\text{Average} = \frac{P+R}{2}$$~~

aka harmonic mean:  
take average with  
emphasize on smaller  
values

$$F1 \text{ score} = \frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right) = 2 \frac{PR}{P+R}$$



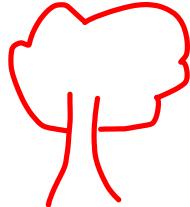
## Decision Trees

# Decision Tree Model

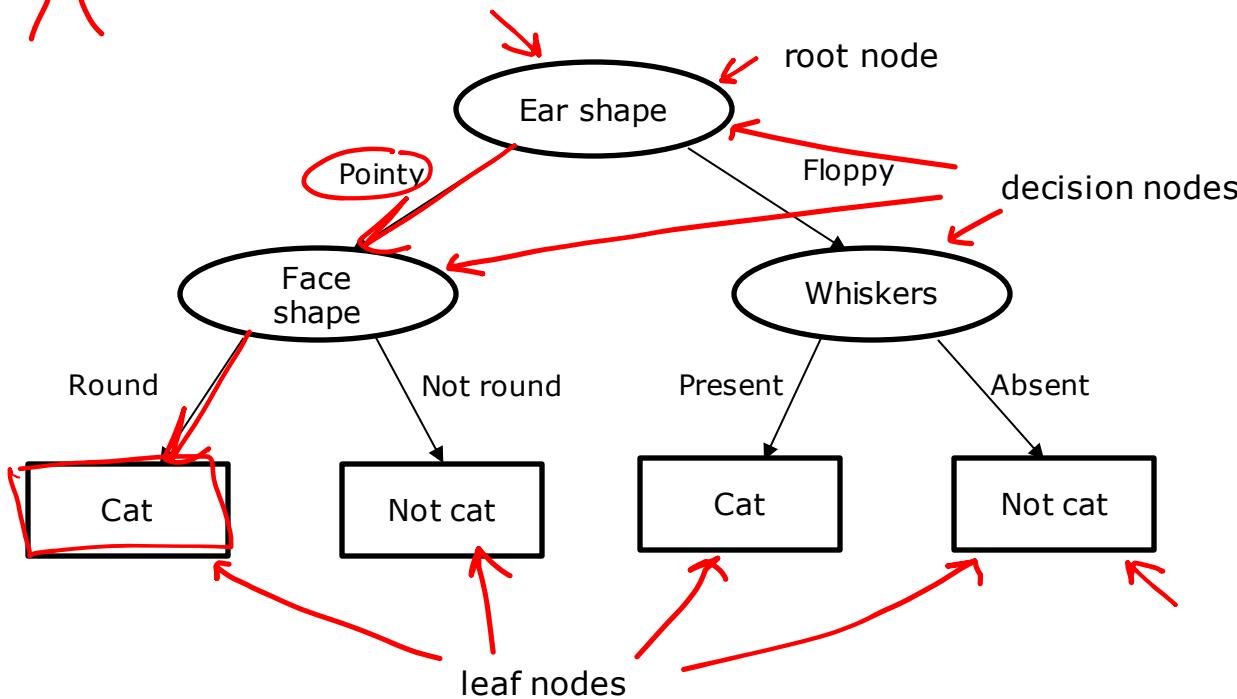
# Cat classification example

	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

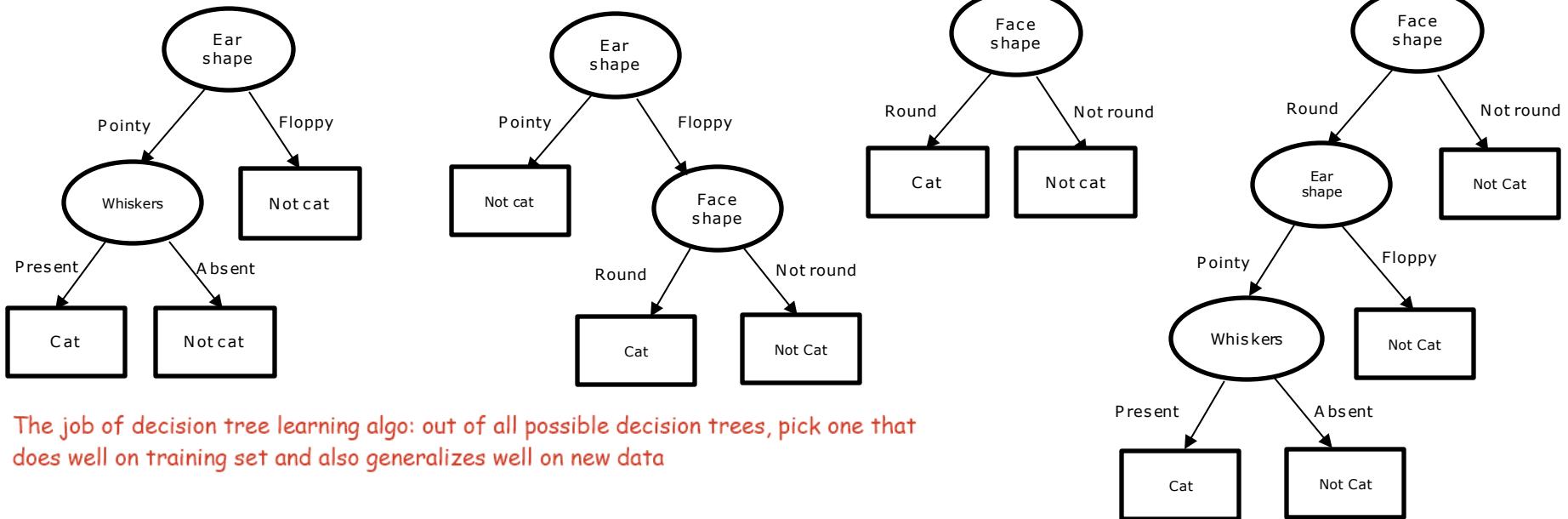


New test example

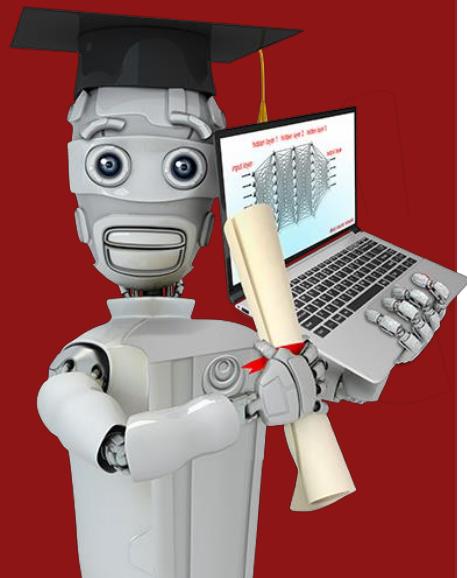


Ear shape: Pointy  
Face shape: Round  
Whiskers: Present

# Decision Tree



The job of decision tree learning algo: out of all possible decision trees, pick one that does well on training set and also generalizes well on new data



## Decision Trees

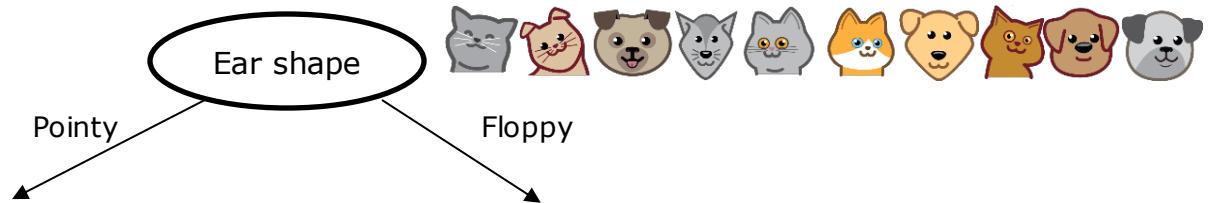
# Learning Process

# Decision Tree Learning

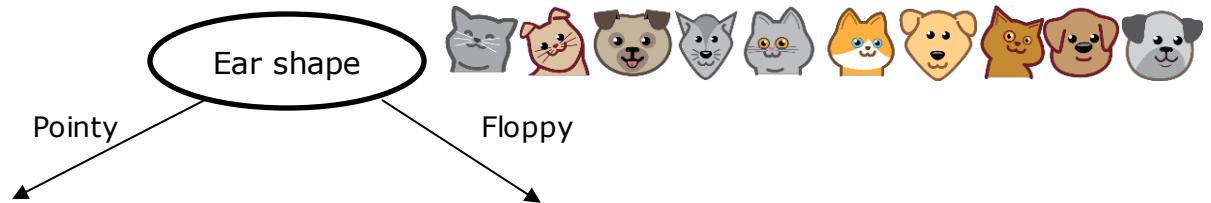
1- What feature to use at root?



# Decision Tree Learning

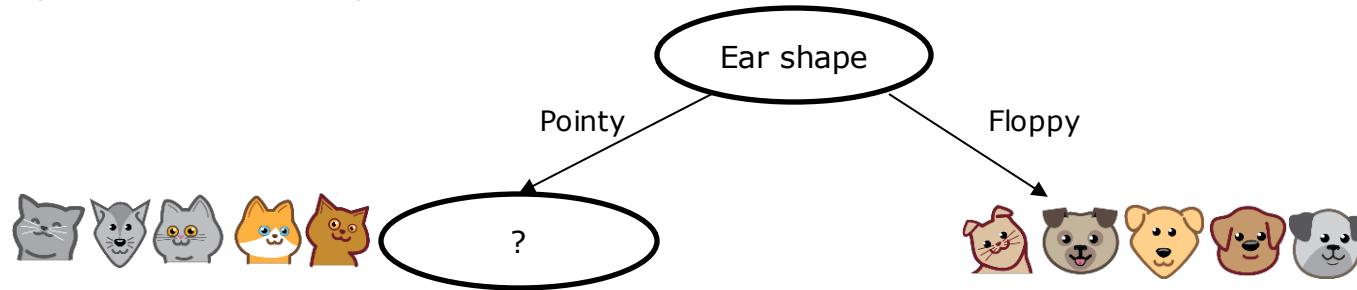


# Decision Tree Learning

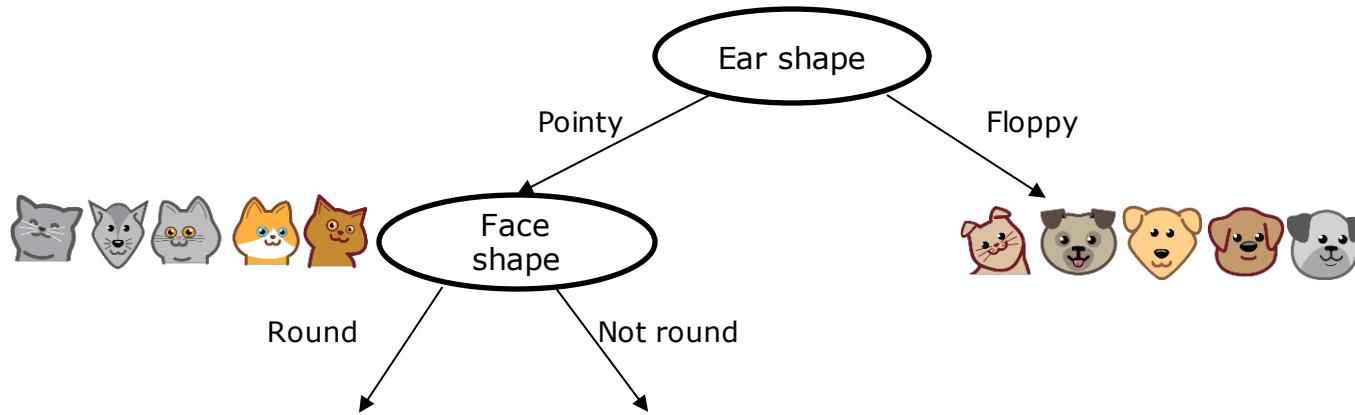


# Decision Tree Learning

2- What node to put on left branch?  
(what feature to use next)

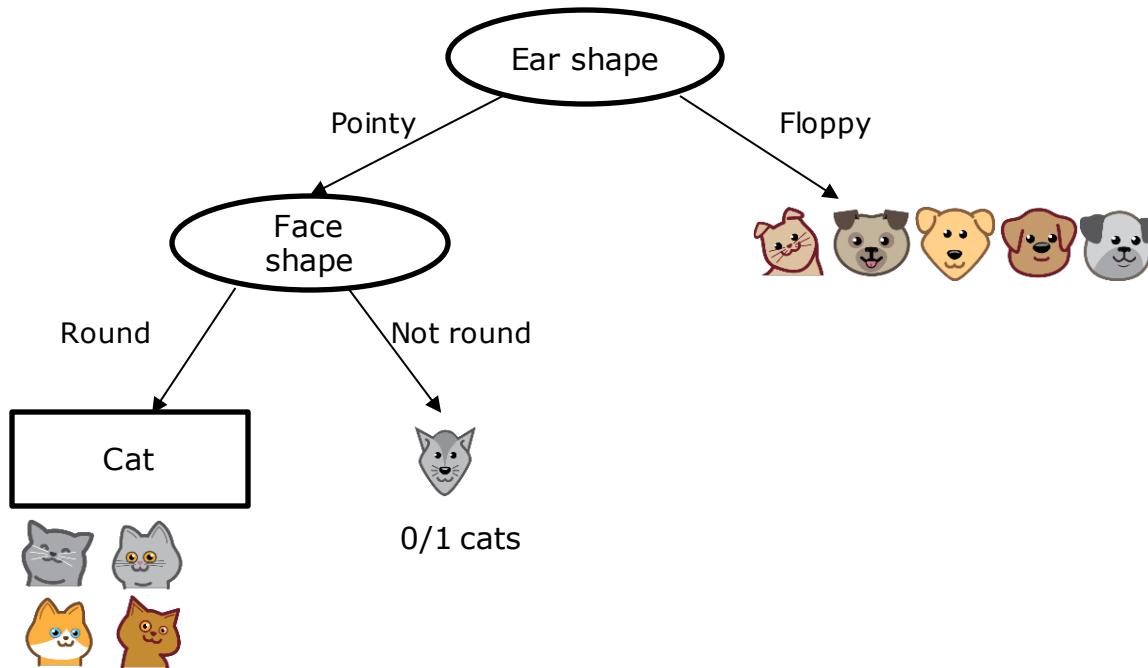


# Decision Tree Learning

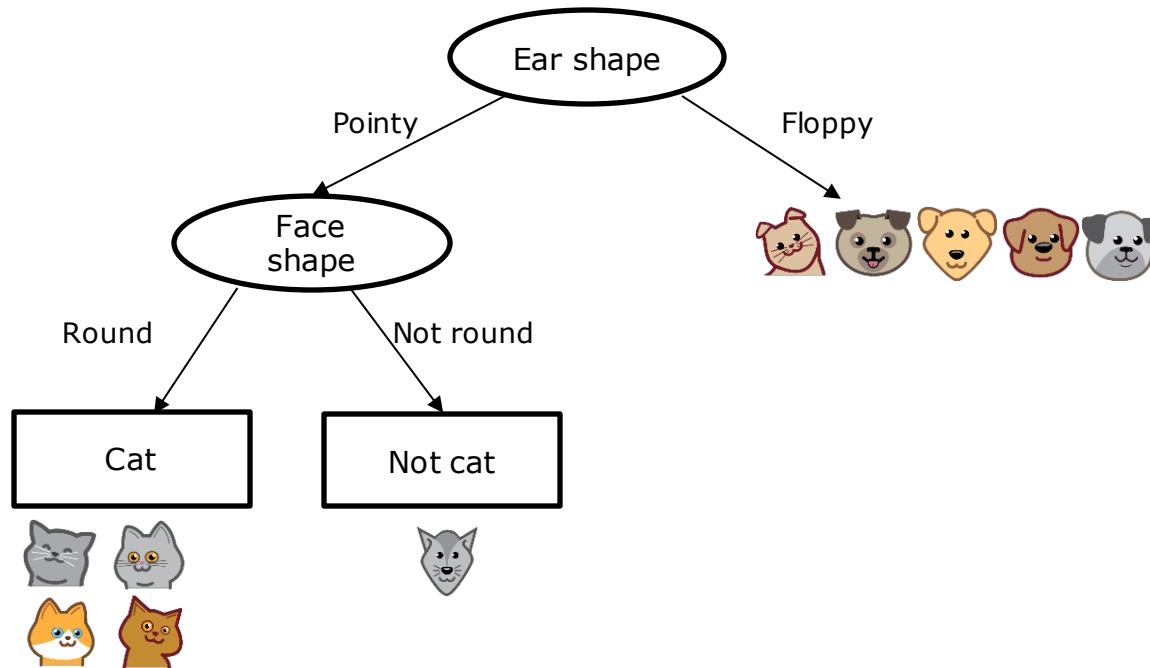


4/4 cats

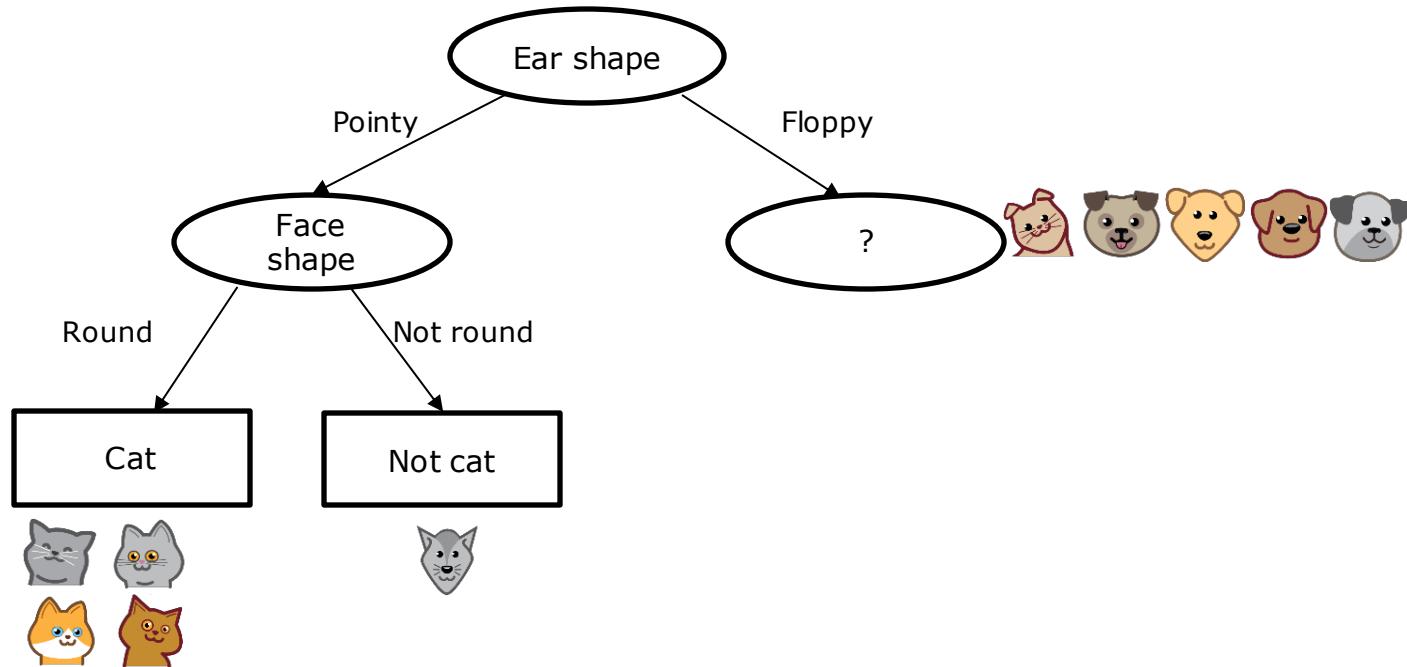
# Decision Tree Learning



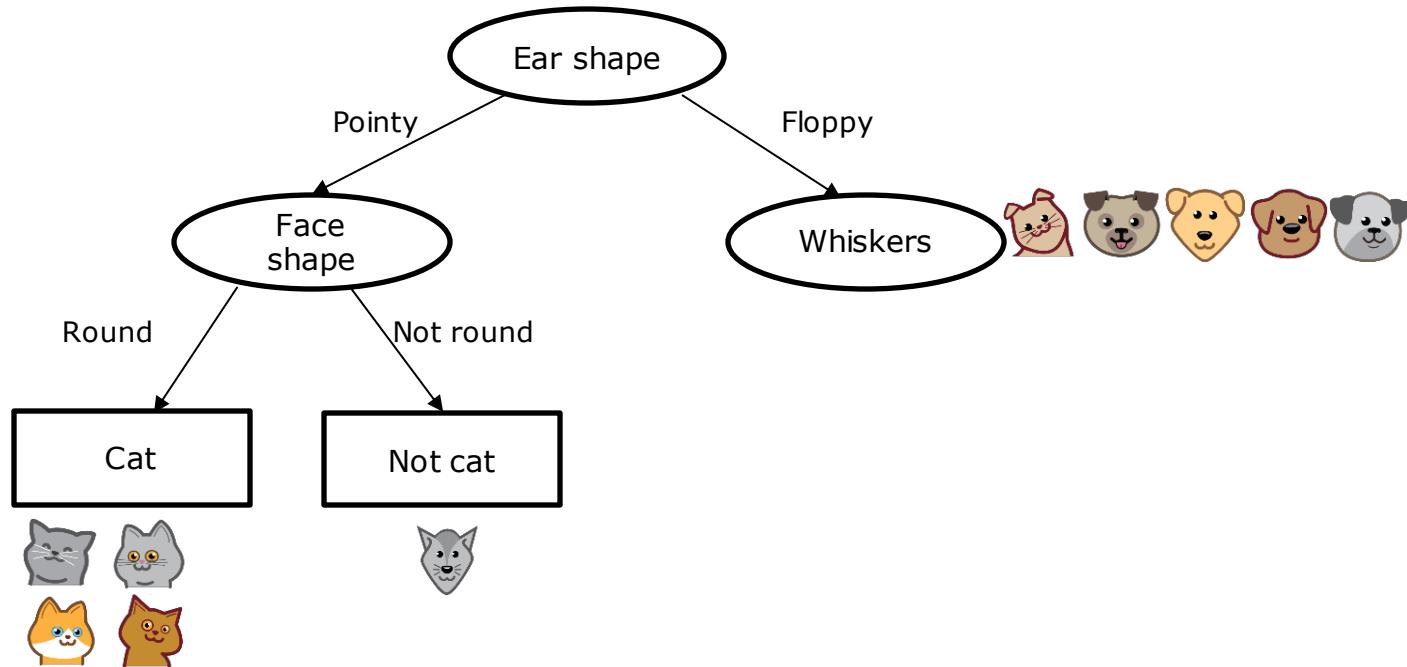
# Decision Tree Learning



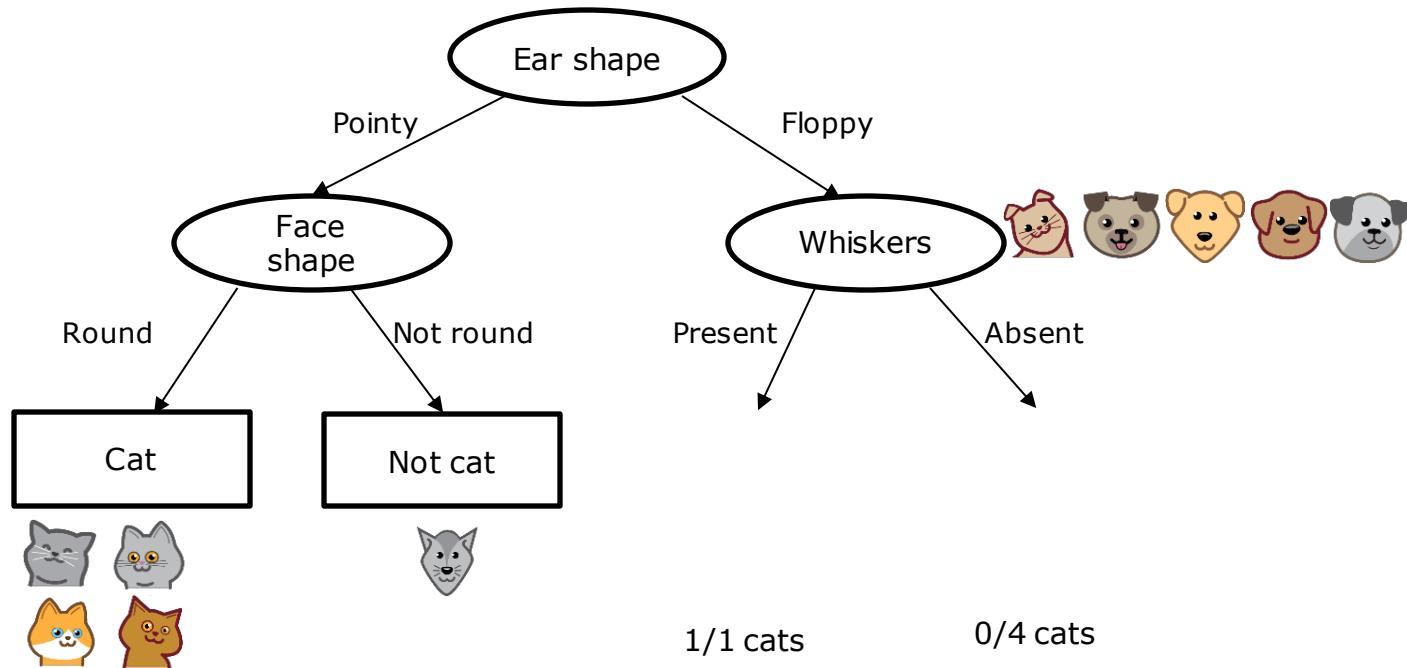
# Decision Tree Learning



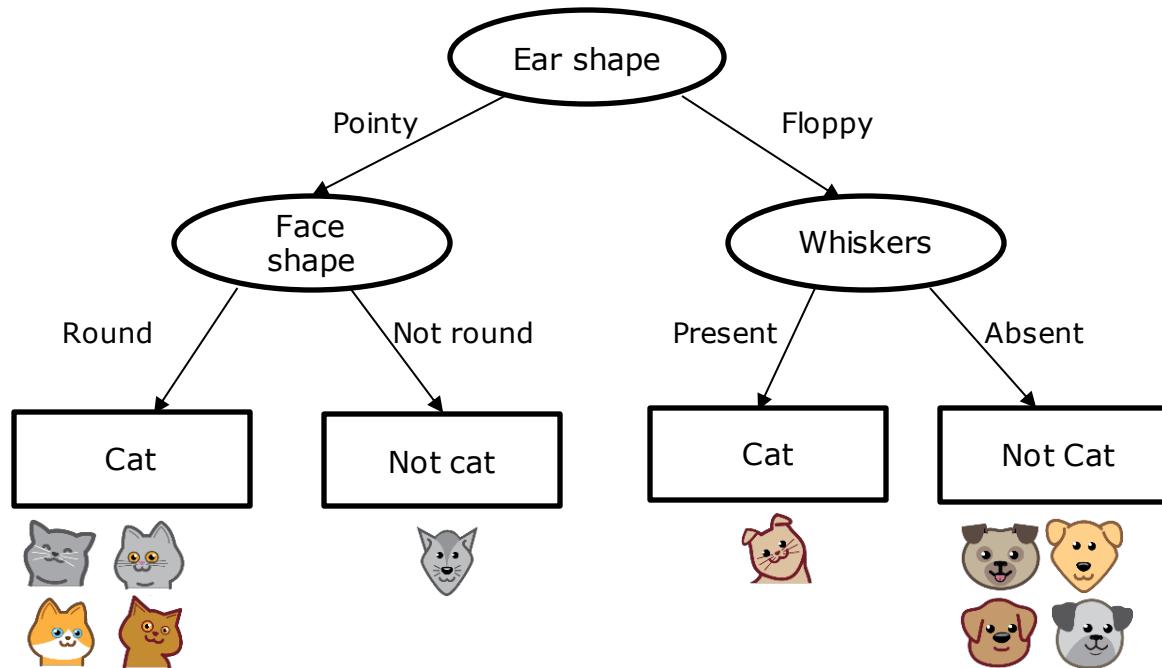
# Decision Tree Learning



# Decision Tree Learning



# Decision Tree Learning



Key decisions:

# Decision Tree Learning

**Decision 1:** How to choose what feature to split on at each node?

?

Maximize purity (or minimize impurity)

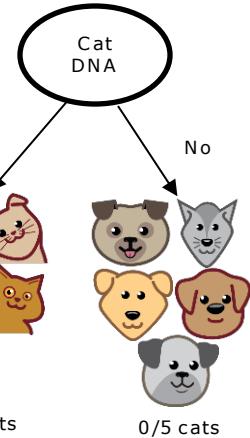
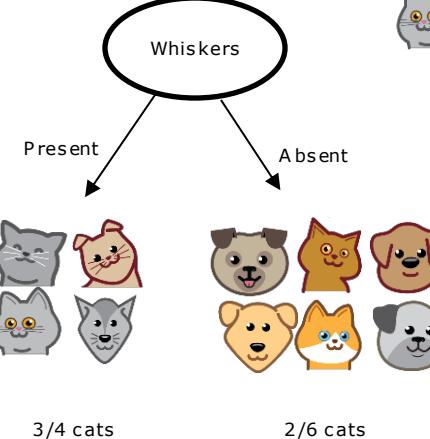
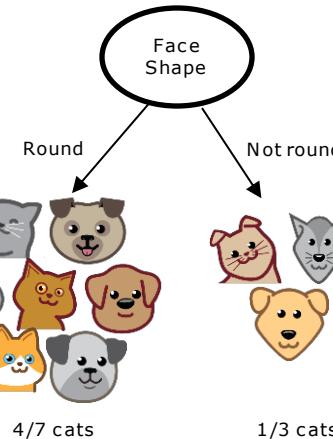
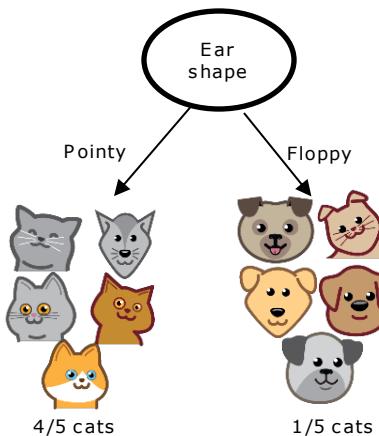


cat only or dog only

# Decision Tree Learning

**Decision 1:** How to choose what feature to split on at each node?

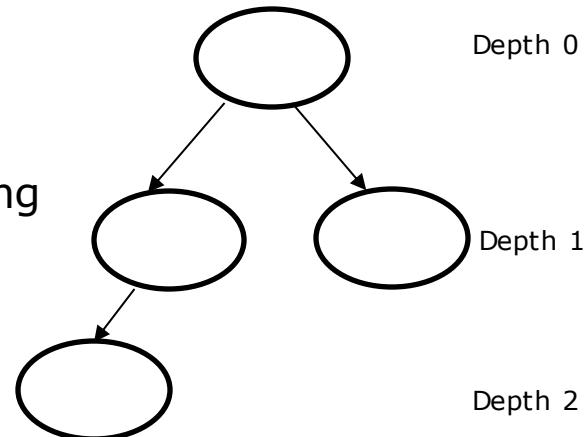
Maximize purity (or minimize impurity)



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

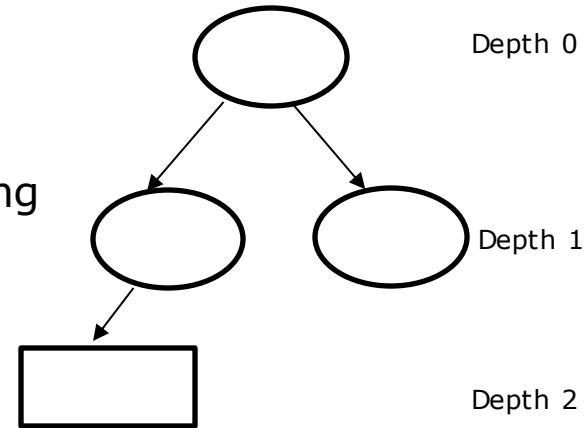
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

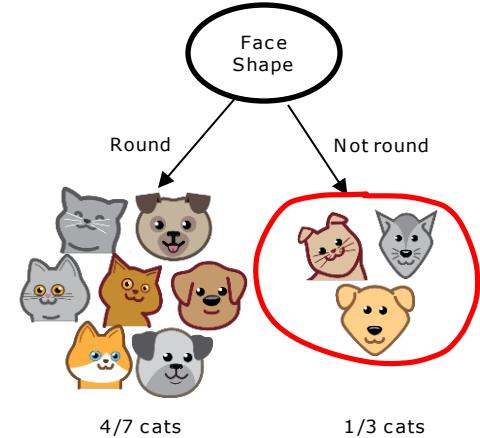
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth



# Decision Tree Learning

**Decision 2:** When do you stop splitting?

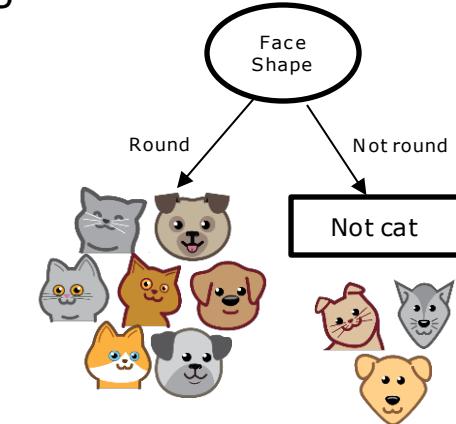
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



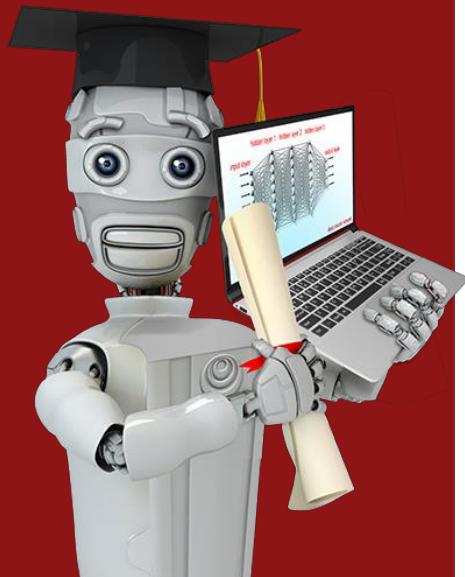
# Decision Tree Learning

**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



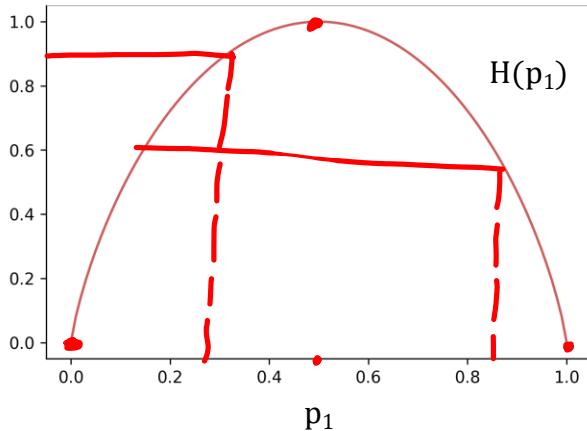
## Decision Tree Learning



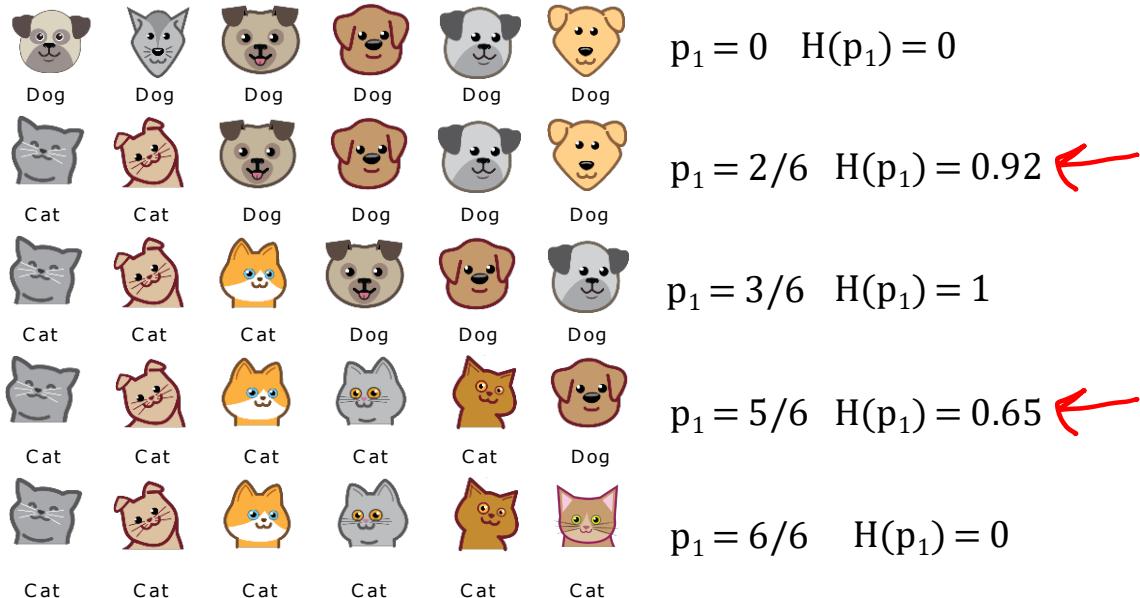
# Measuring purity

# Entropy as a measure of impurity

$p_1$  = fraction of examples that are cats

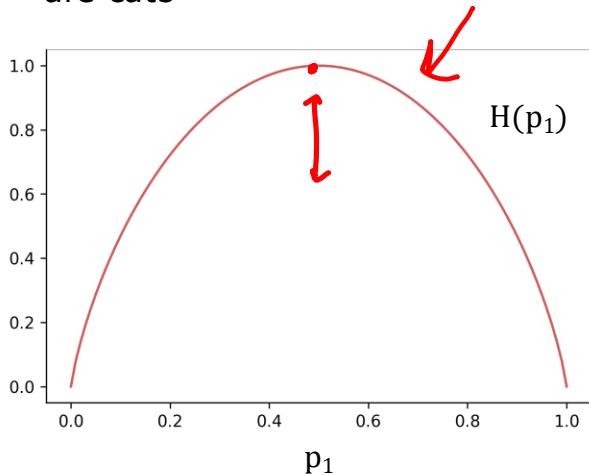


→ (Unit of disorderly)



# Entropy as a measure of impurity

$p_1$  = fraction of examples that are cats



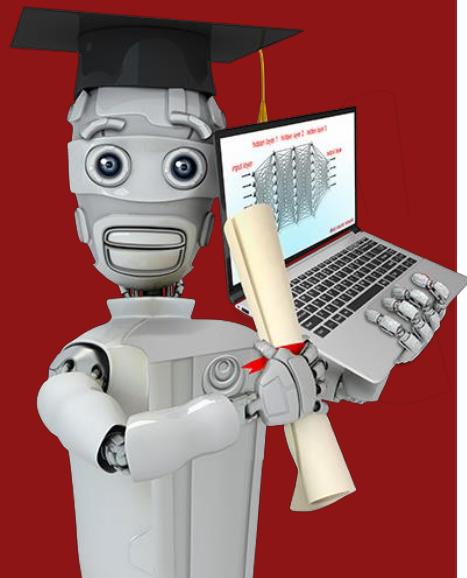
$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

$$= -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$



Note: “ $0 \log(0)$ ” = 0



# Decision Tree Learning

## Choosing a split: Information Gain

reduction of impurity:  
information gain

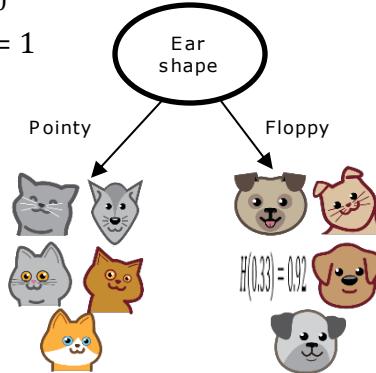
if reduction is too small  
(below a threshold), it  
doesn't worth to split ->  
reduce risk of overfitting

# Choosing a split

entropy of root node:

$$p_1 = \frac{5}{10} = 0.5$$

$$H(0.5) = 1$$



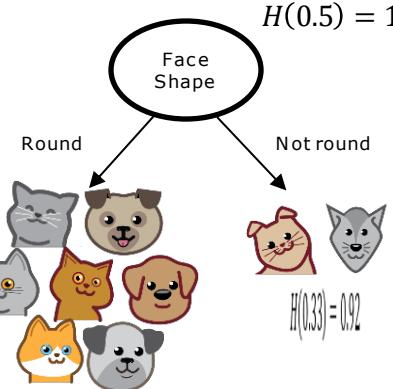
$$p_1 = \frac{4}{5} = 0.8 \quad p_1 = \frac{1}{5} = 0.2$$

$$H(0.8) = 0.72 \quad H(0.2) = 0.72$$

$$\text{reduction in entropy} \quad H(0.5) - \left( \frac{5}{10} H(0.8) + \frac{5}{10} H(0.2) \right)$$

= 0.28

entropy of root - weighted avg

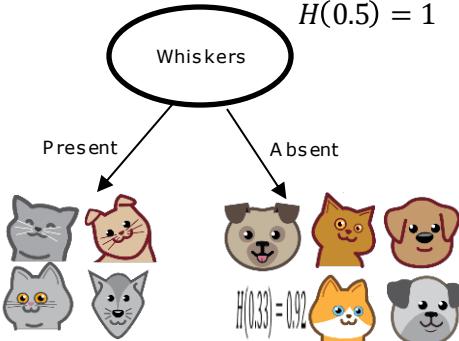


$$p_1 = \frac{4}{7} = 0.57 \quad p_1 = \frac{1}{3} = 0.33$$

$$H(0.57) = 0.99 \quad H(0.33) = 0.92$$

$$H(0.5) - \left( \frac{7}{10} H(0.57) + \frac{3}{10} H(0.33) \right)$$

= 0.03



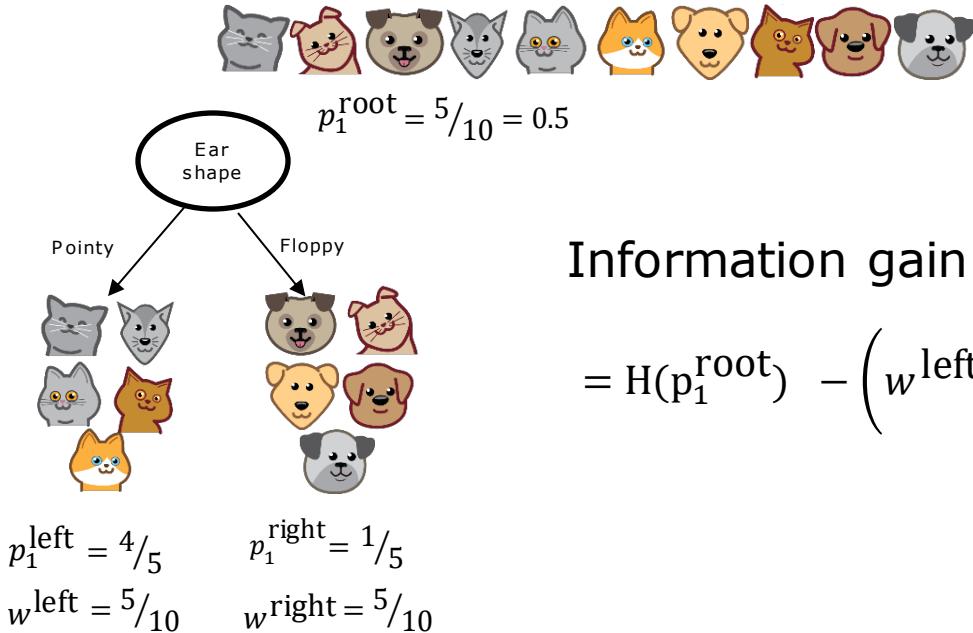
$$p_1 = \frac{3}{4} = 0.75 \quad p_1 = \frac{2}{6} = 0.33$$

$$H(0.75) = 0.81 \quad H(0.33) = 0.92$$

$$H(0.5) - \left( \frac{4}{10} H(0.75) + \frac{6}{10} H(0.33) \right)$$

= 0.12

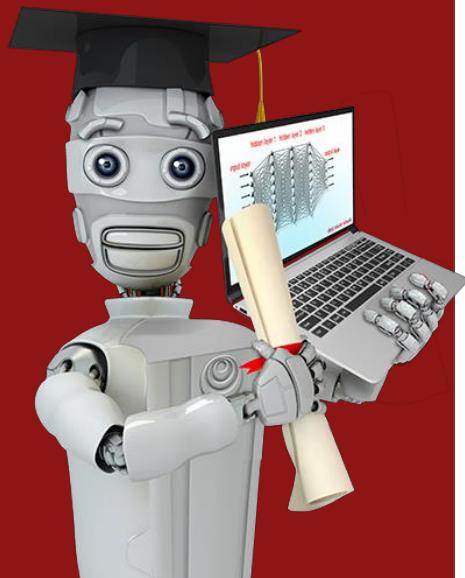
# Information Gain



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

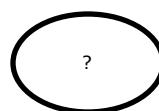


# Putting it together

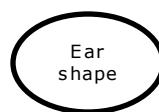
# Decision Tree Learning

- 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



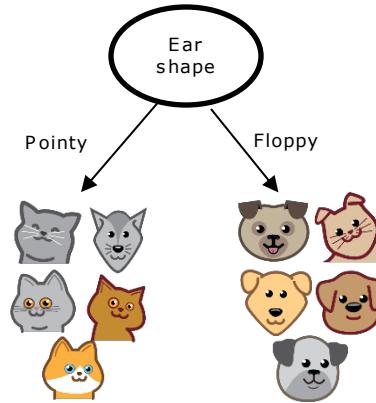
# Recursive splitting



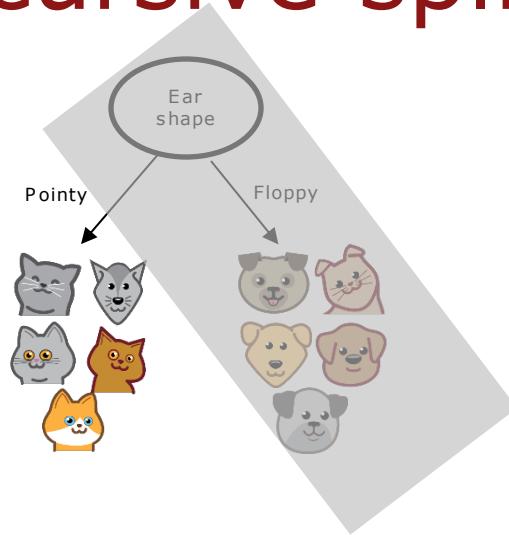
# Recursive splitting



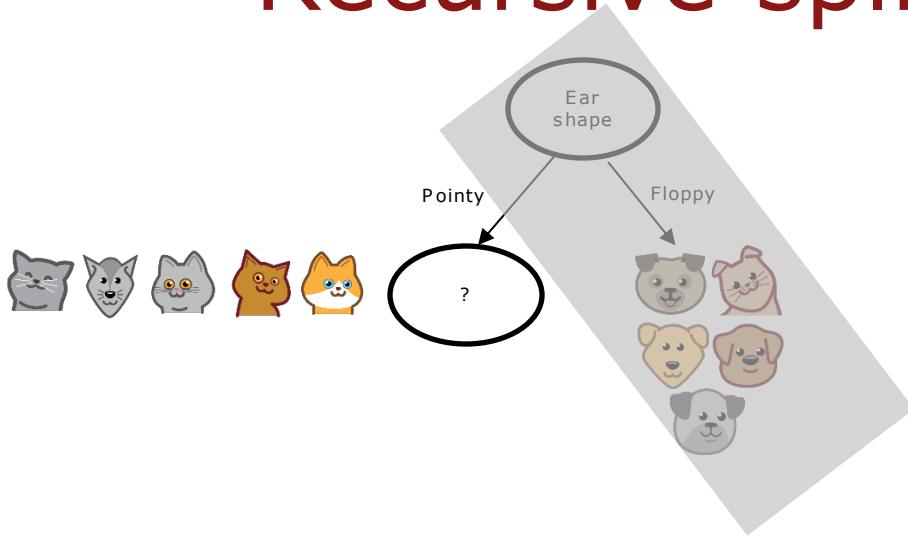
# Recursive splitting



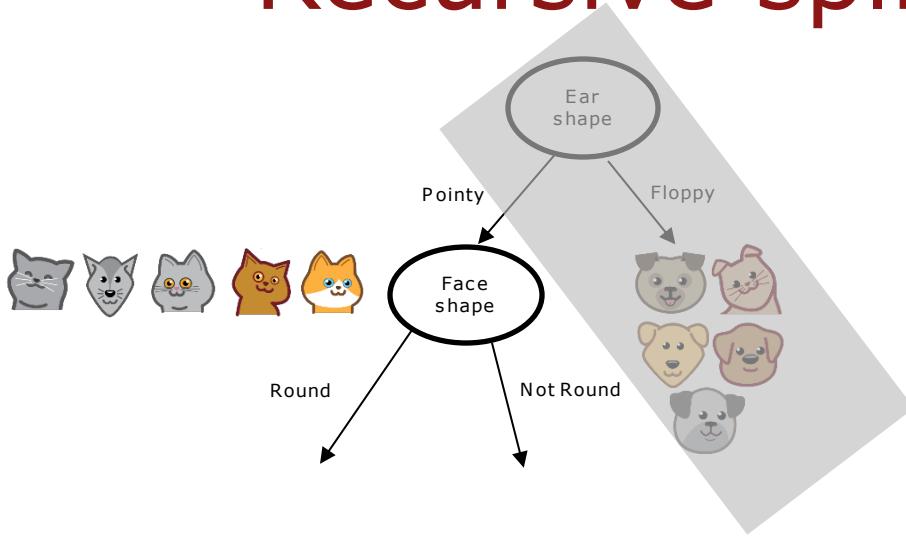
# Recursive splitting



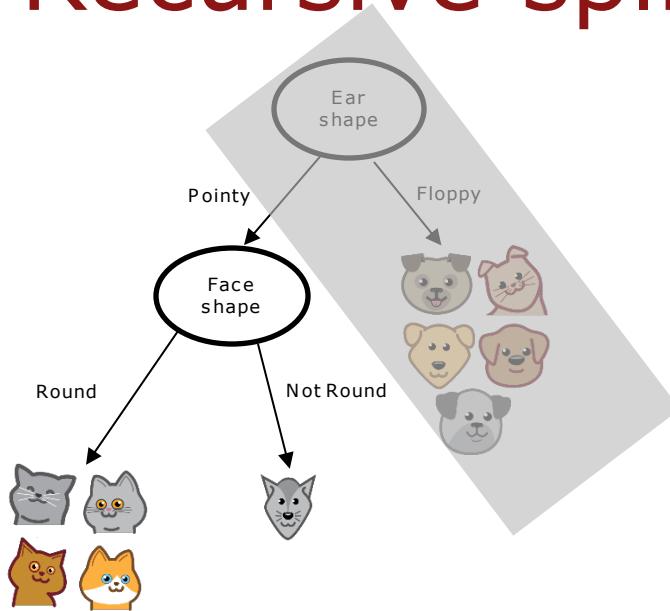
# Recursive splitting



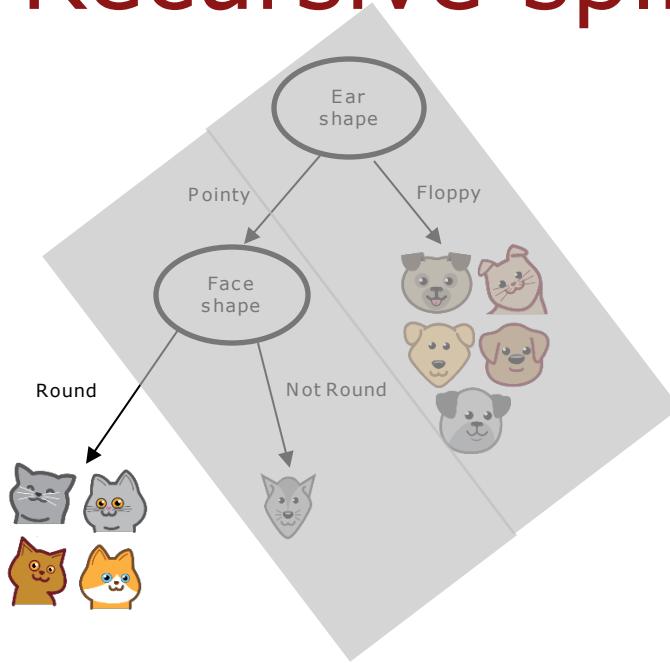
# Recursive splitting



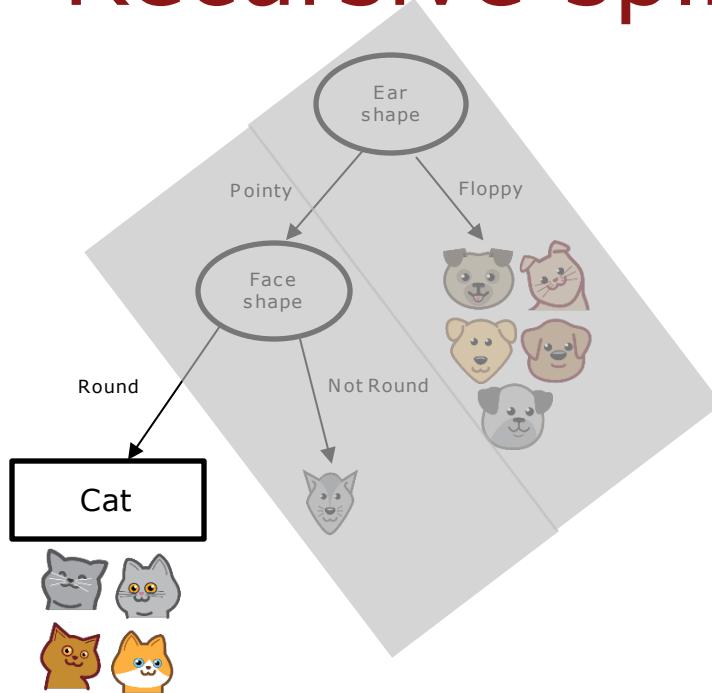
# Recursive splitting



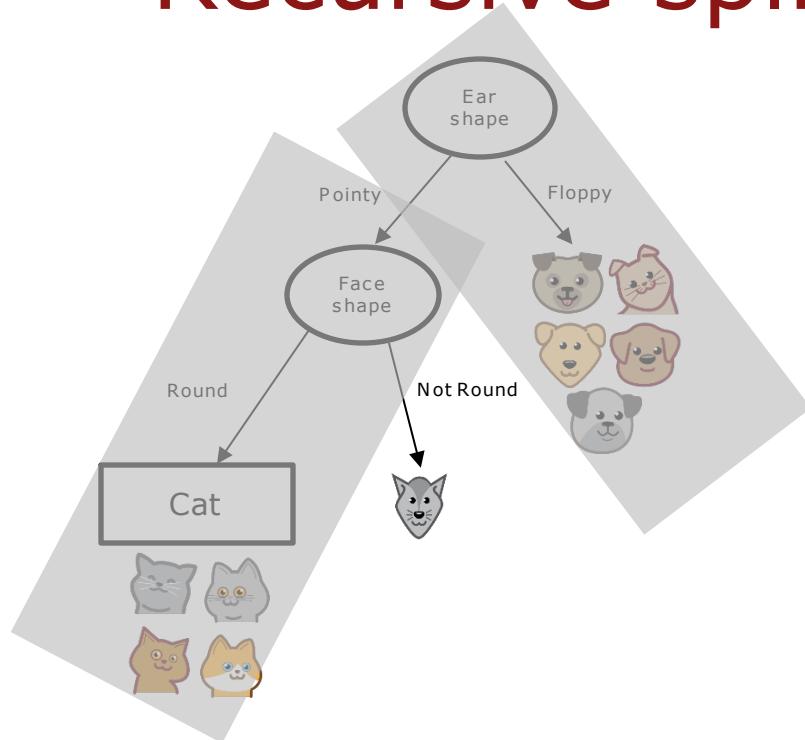
# Recursive splitting



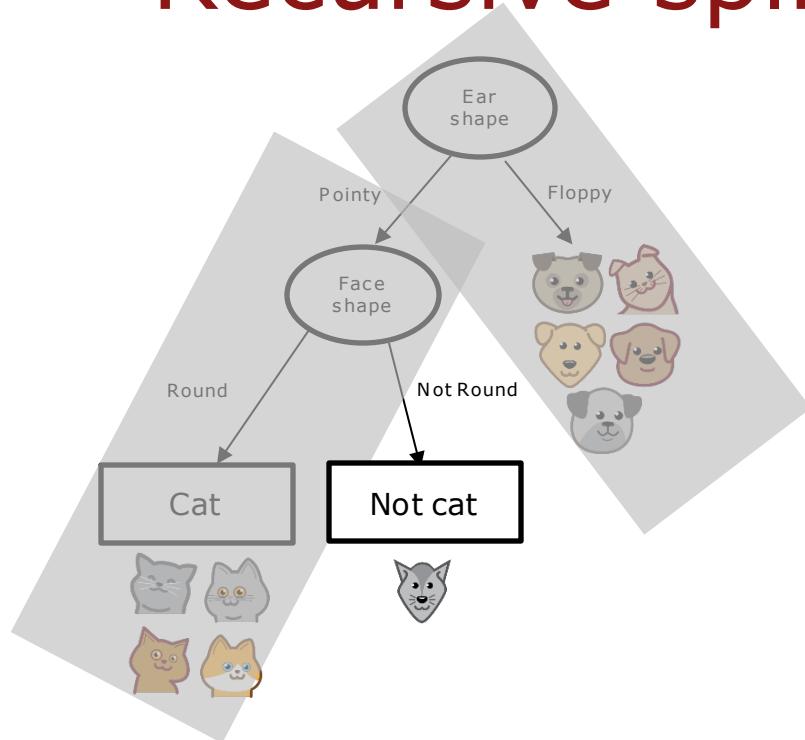
# Recursive splitting



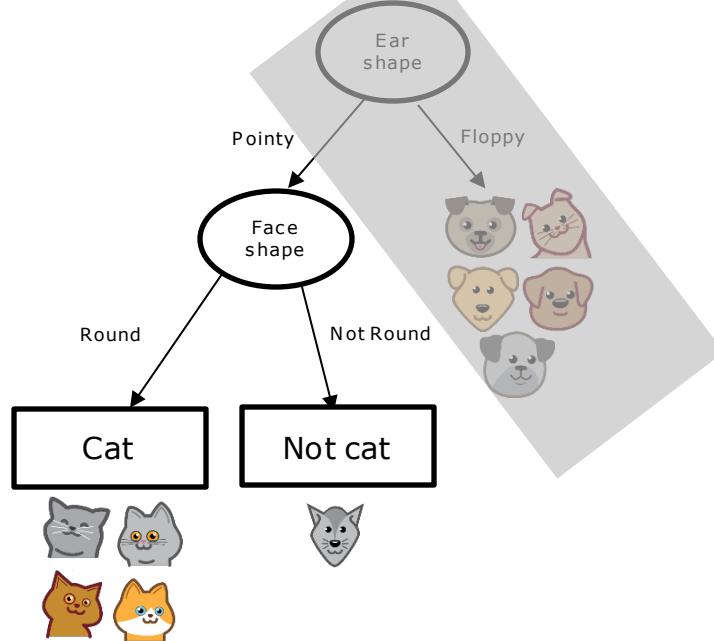
# Recursive splitting



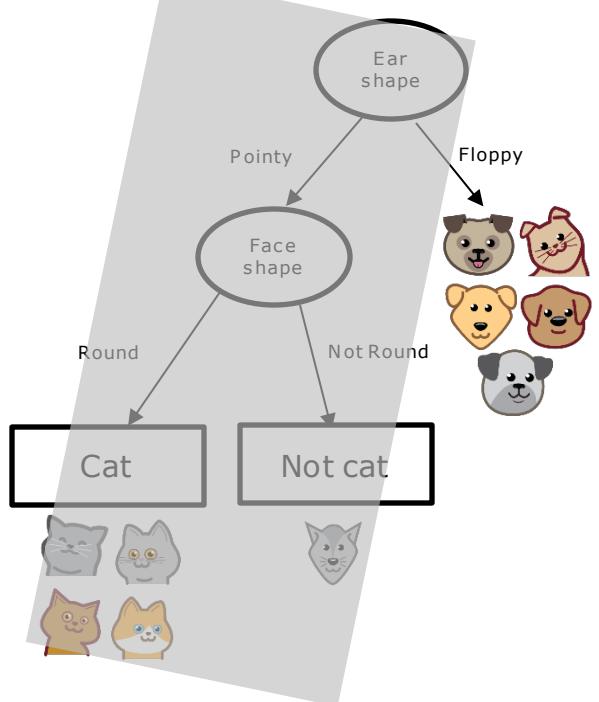
# Recursive splitting



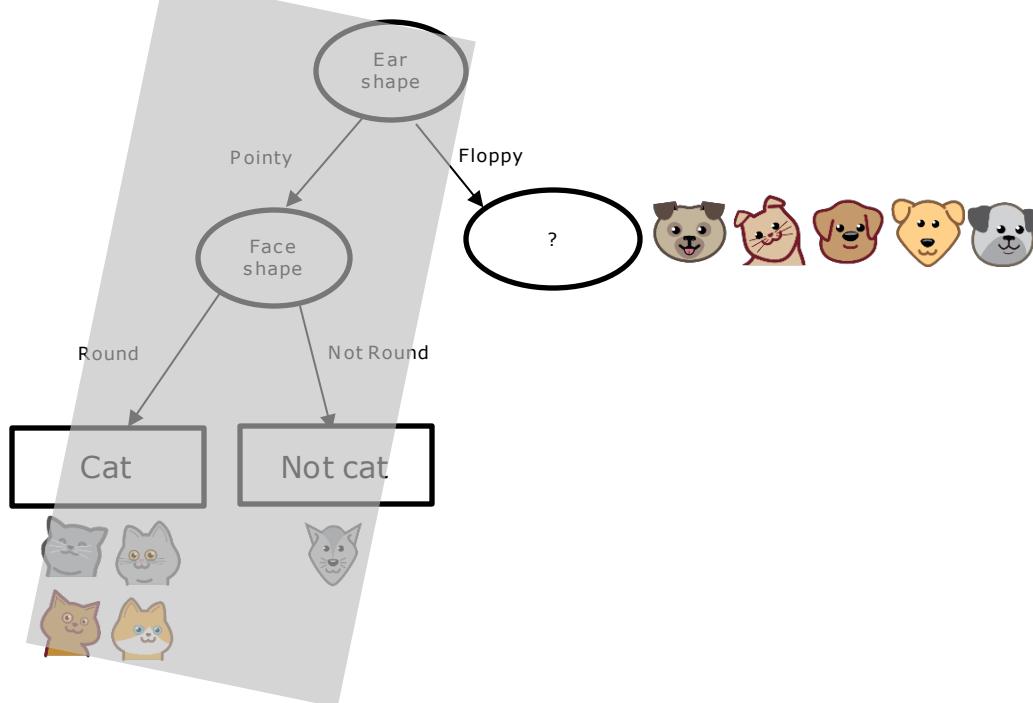
# Recursive splitting



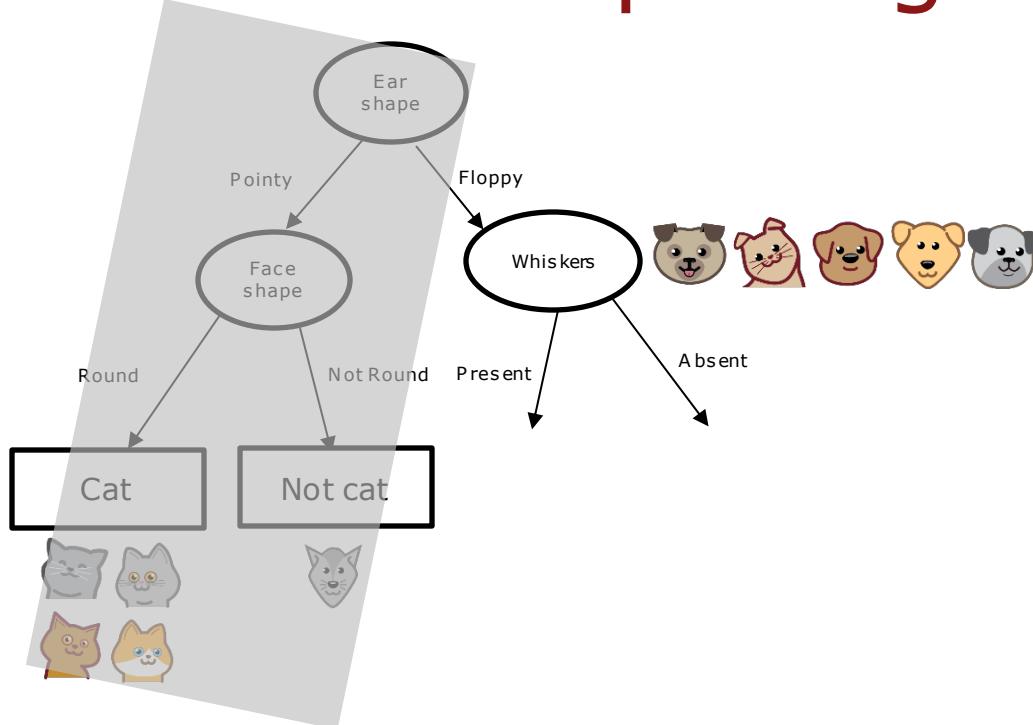
# Recursive splitting



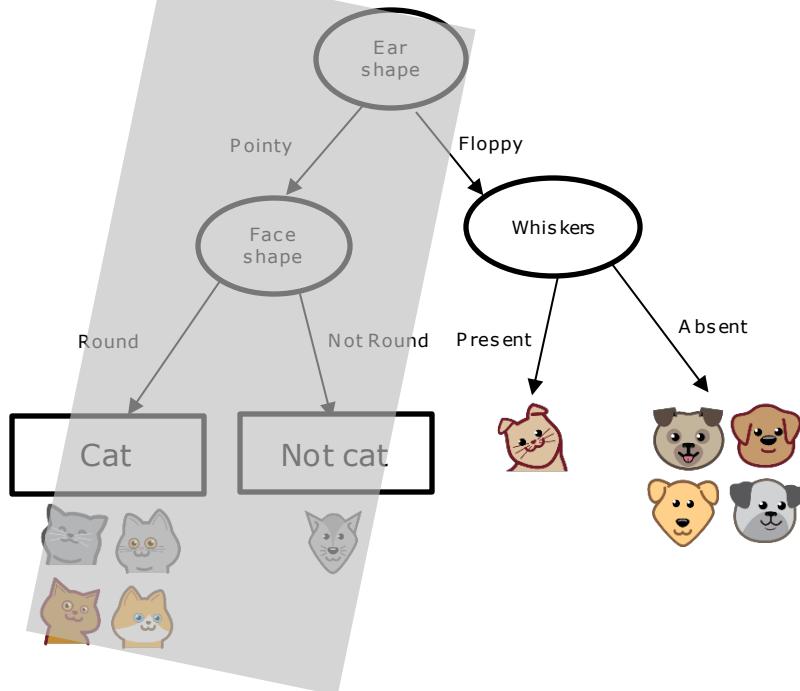
# Recursive splitting



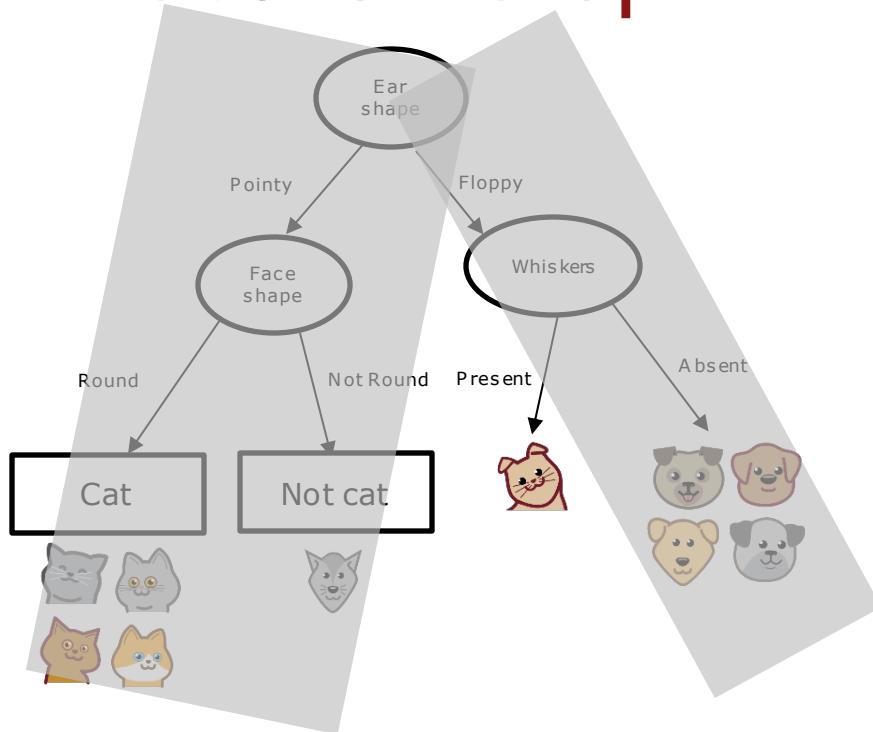
# Recursive splitting



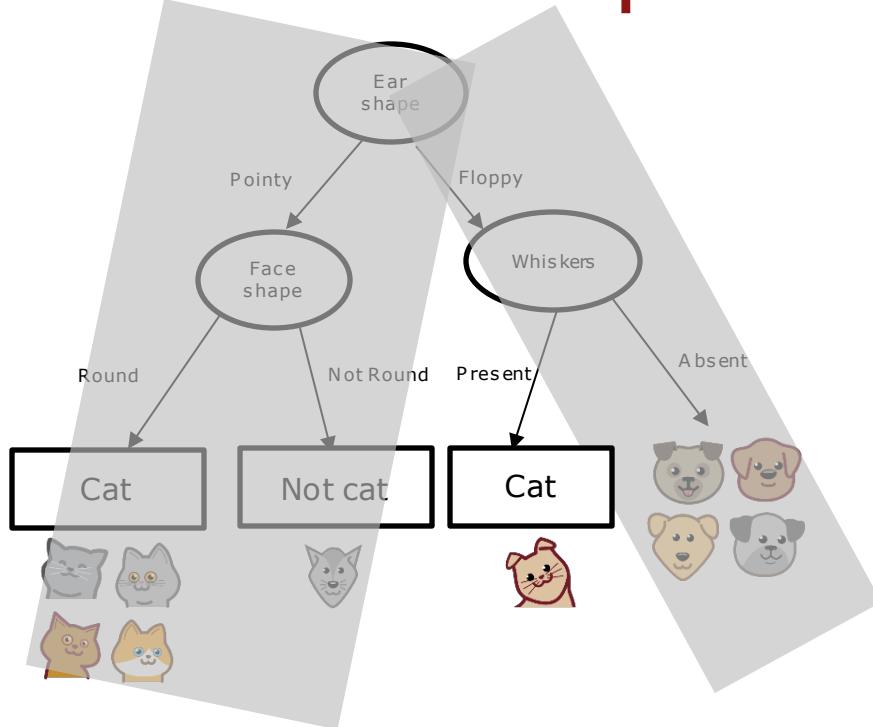
# Recursive splitting



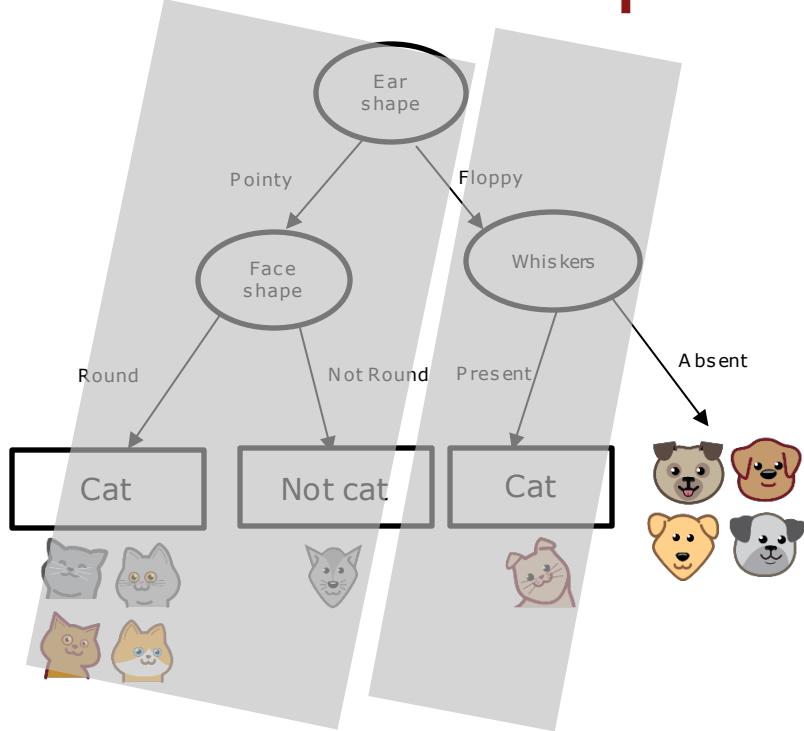
# Recursive splitting



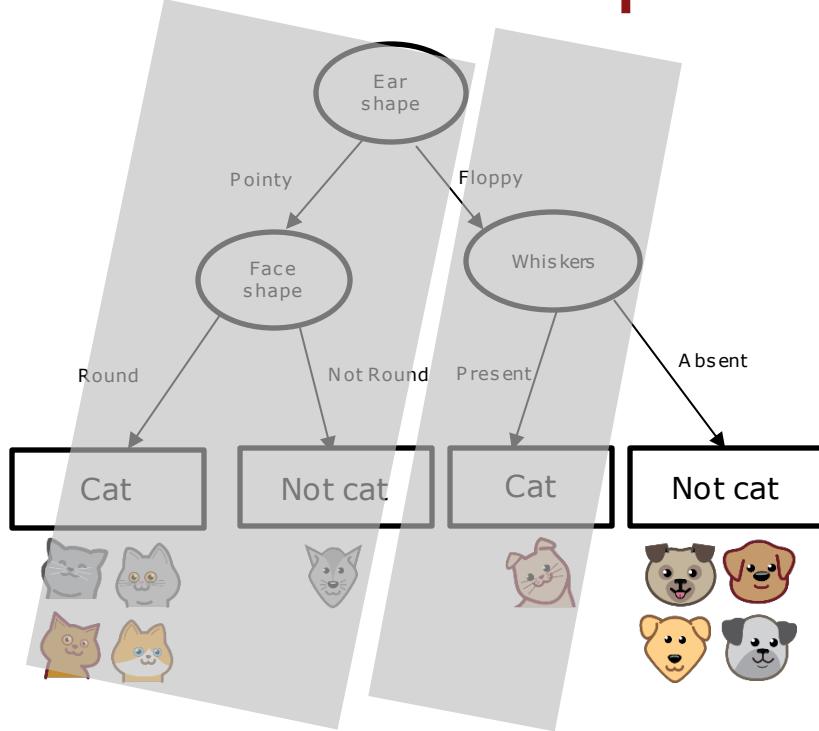
# Recursive splitting



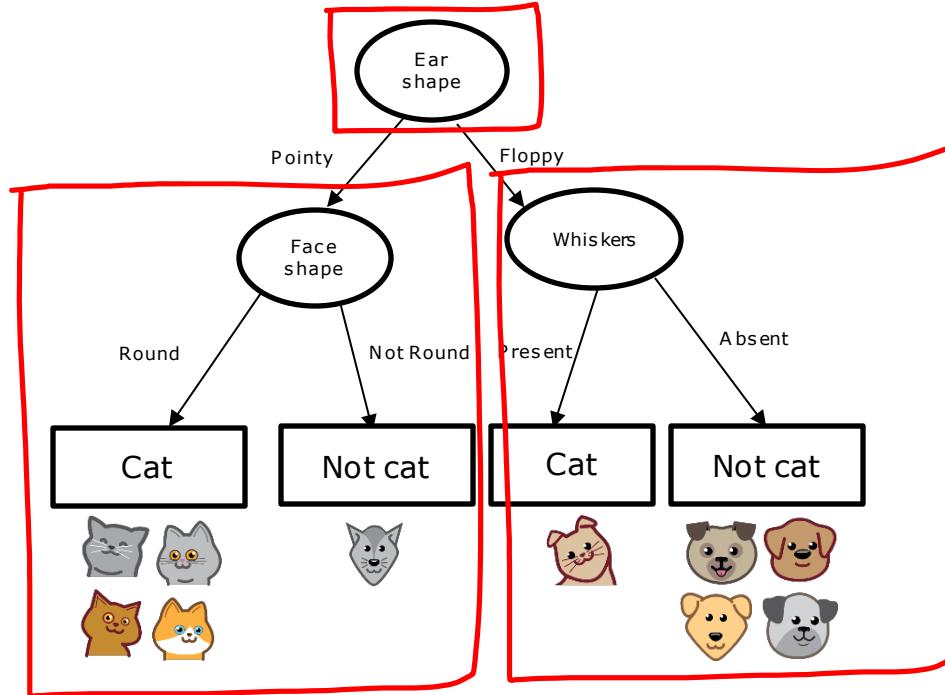
# Recursive splitting



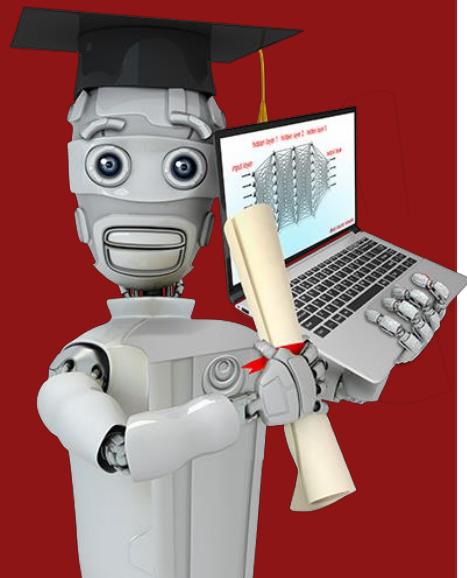
# Recursive splitting



# Recursive splitting



Recursive algorithm

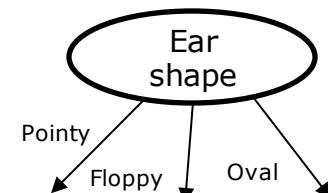


# Decision Tree Learning

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

only one of the features will take 1

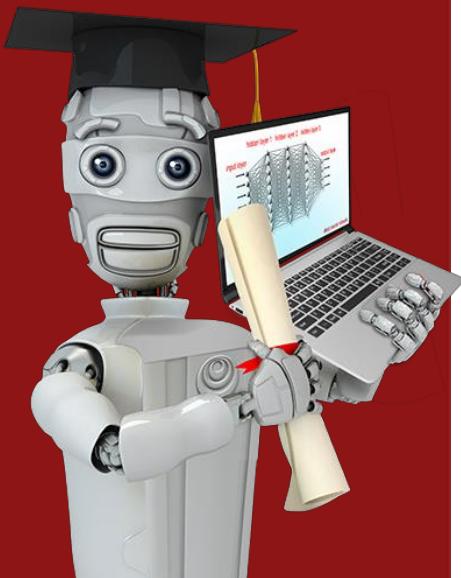
Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat	
	Pointy	1	0	0	Round	Present	1
	Oval	0	0	1	Not round	Present	1
	Oval	0	0	1	Round	Absent	0
	Pointy	1	0	0	Not round	Present	0
	Oval	0	0	1	Round	Present	1
	Pointy	1	0	0	Round	Absent	1
	Floppy	0	1	0	Not round	Absent	0
	Oval	0	0	1	Round	Absent	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

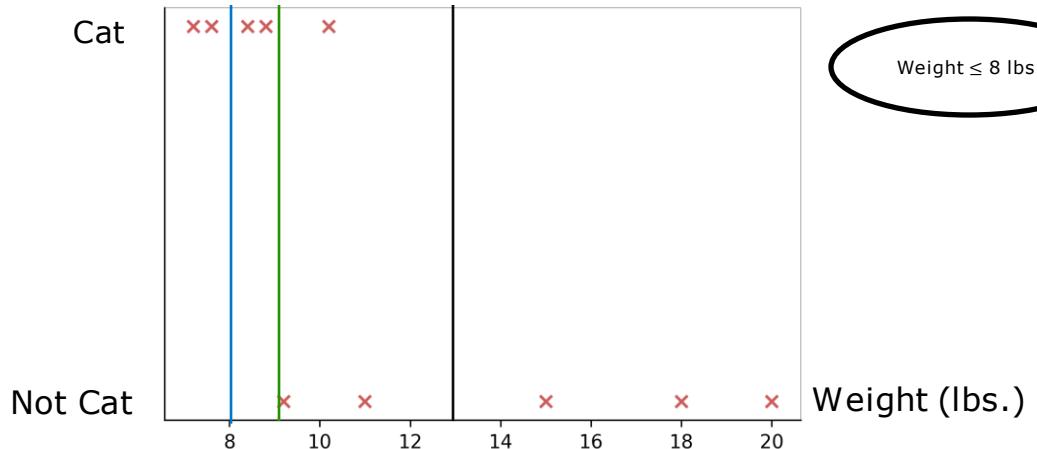


# Continuous features



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



we can sort these 10 examples base on weight and choose mid points and compute info gain. So for 10 examples we test 9 points

$$H(0.5) - \left( \frac{2}{10} H\left(\frac{2}{2}\right) + \frac{8}{10} H\left(\frac{3}{8}\right) \right) = 0.24$$

$$H(0.5) - \left( \frac{4}{10} H\left(\frac{4}{4}\right) + \frac{6}{10} H\left(\frac{1}{6}\right) \right) = 0.61$$

$$H(0.5) - \left( \frac{7}{10} H\left(\frac{5}{7}\right) + \frac{3}{10} H\left(\frac{0}{3}\right) \right) = 0.40$$

# Decision Tree Learning

## Regression Trees (optional)

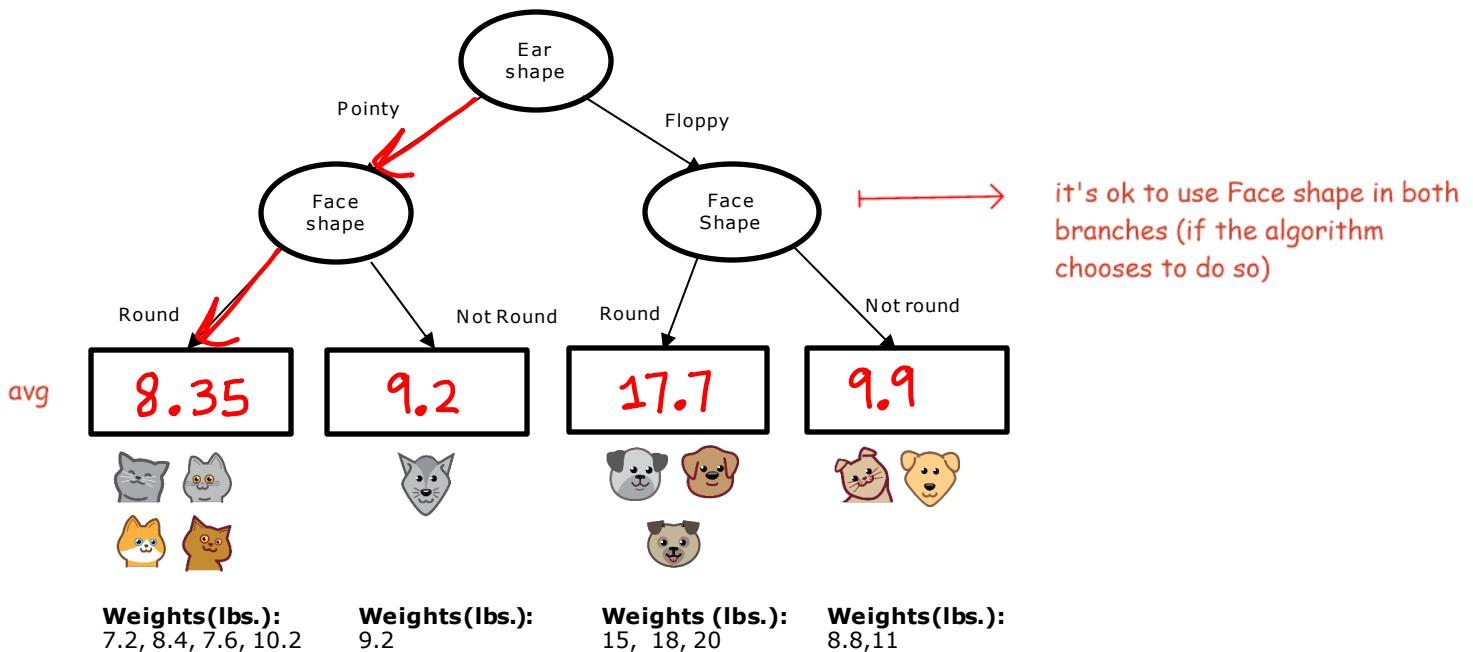


# Regression with Decision Trees: Predicting a number

	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

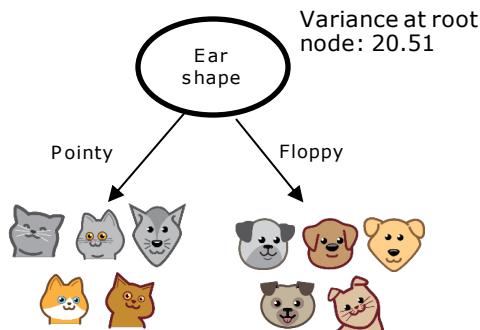
 **X**       **y**

# Regression with Decision Trees



reduce the variance of  $y$

# Choosing a split

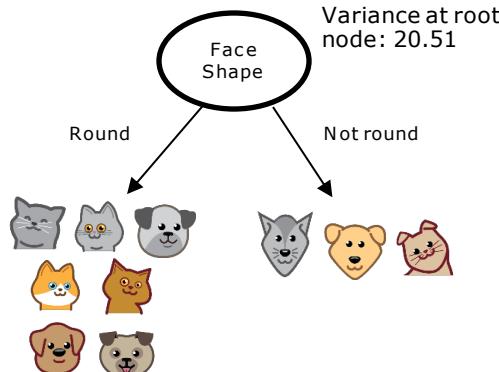


Weights: 7.2,  
9.2, 8.4, 7.6, 10.2

Variance: 1.47

$$w^{\text{left}} = \frac{5}{10} \quad w^{\text{right}} = \frac{5}{10}$$

$$20.51 - \left( \frac{5}{10} * 1.47 + \frac{5}{10} * 21.87 \right) \\ = 8.84$$

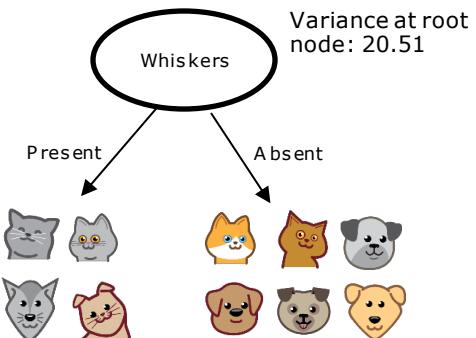


Weights: 7.2, 15,  
8.4, 7.6, 10.2, 18, 20

Variance: 27.80

$$w^{\text{left}} = \frac{7}{10} \quad w^{\text{right}} = \frac{3}{10}$$

$$20.51 - \left( \frac{7}{10} * 27.80 + \frac{3}{10} * 1.37 \right) \\ = 0.64$$

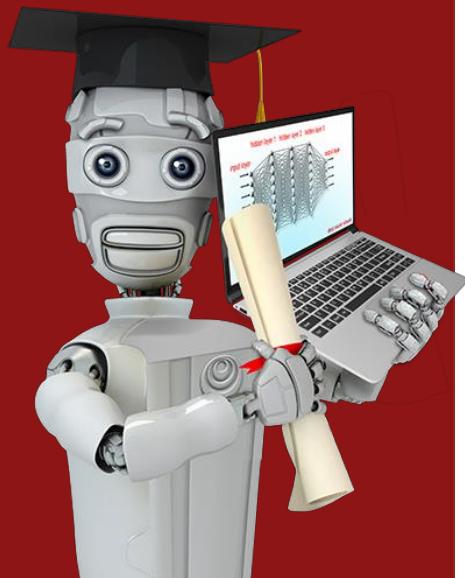


Weights: 7.2, 8.8,  
9.2, 8.4

Variance: 0.75

$$w^{\text{left}} = \frac{4}{10} \quad w^{\text{right}} = \frac{6}{10}$$

$$20.51 - \left( \frac{4}{10} * 0.75 + \frac{6}{10} * 23.32 \right) \\ = 6.22$$

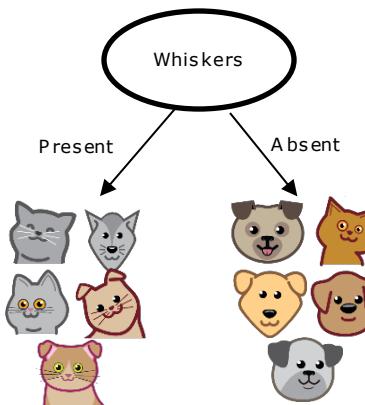
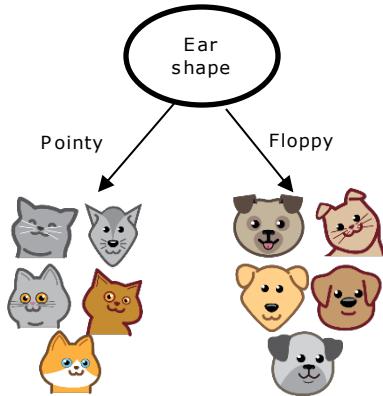
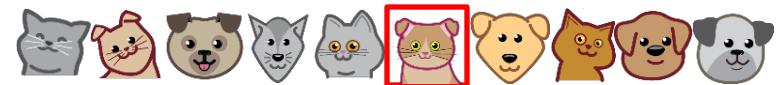


## Tree ensembles

Using multiple decision trees

# Trees are highly sensitive to small changes of the data

change one training example

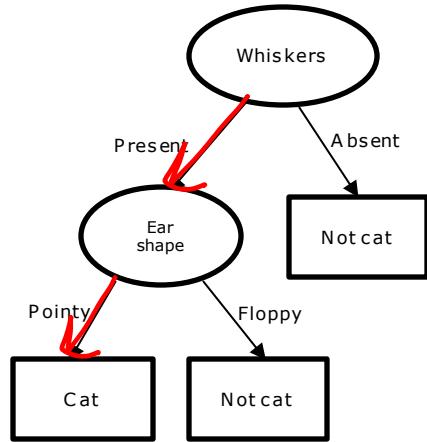


# Tree ensemble

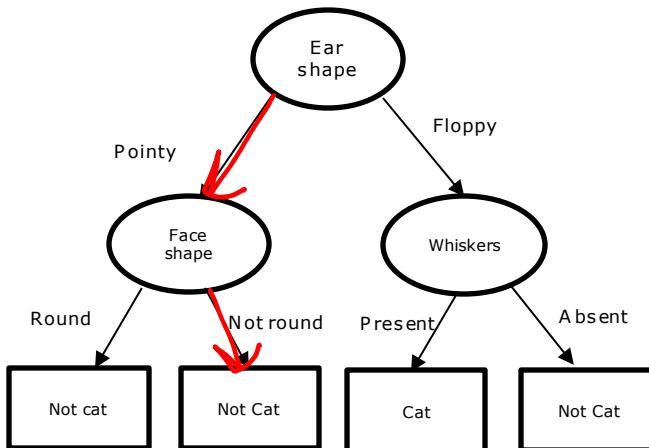
New test example



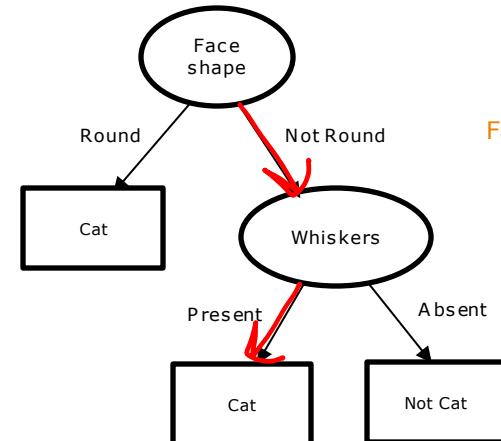
Ear shape: Pointy  
Face shape: Not Round  
Whiskers: Present



Prediction: Cat

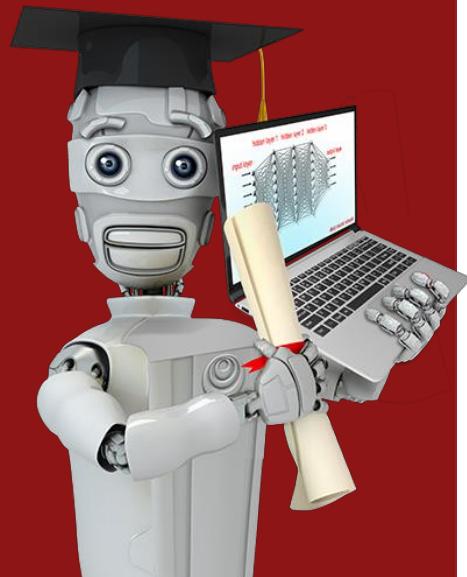


Prediction: Not cat



Prediction: Cat

Final prediction: Cat



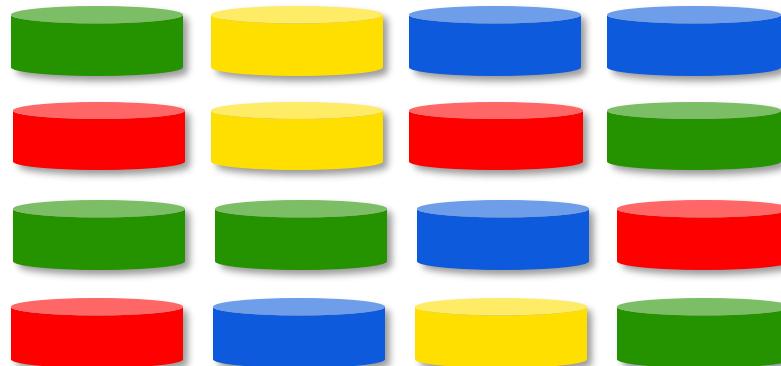
## Tree ensembles

### Sampling with replacement

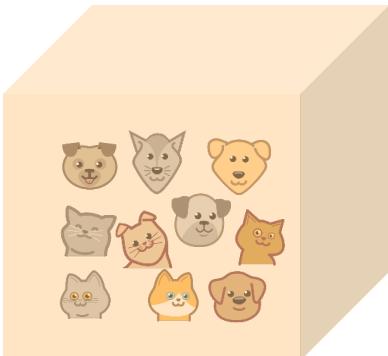
# Sampling with replacement



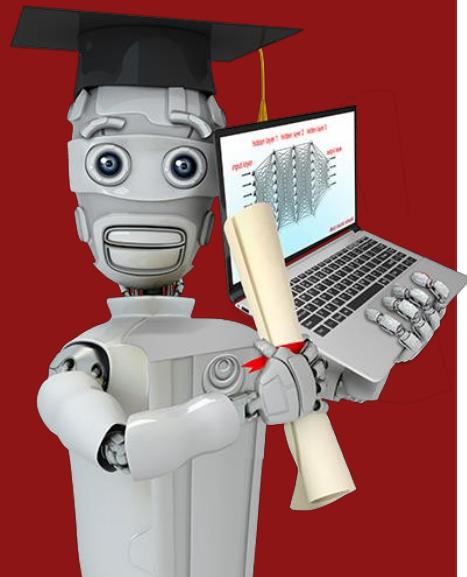
Sampling with replacement:



# 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

aka Bagged decision Tree

Given training set of size  $m$

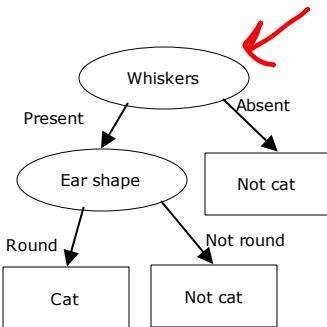
(setting  $B$  higher doesn't hurt performance  
but it doesn't get better either)

For  $b = 1$  to  $\textcircled{B}$     ( $B \rightarrow$  any value (64-128), 100 is good)

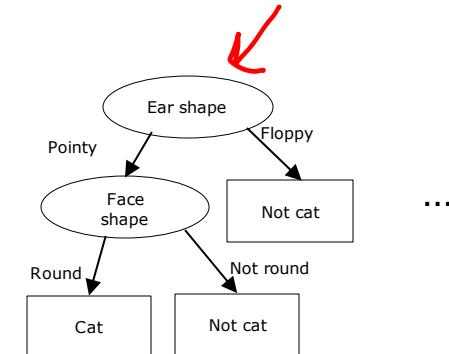
Use sampling with replacement to create a new training set of size  $m$

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	Pointy	Absent	No
Pointy	Round	Present	Yes



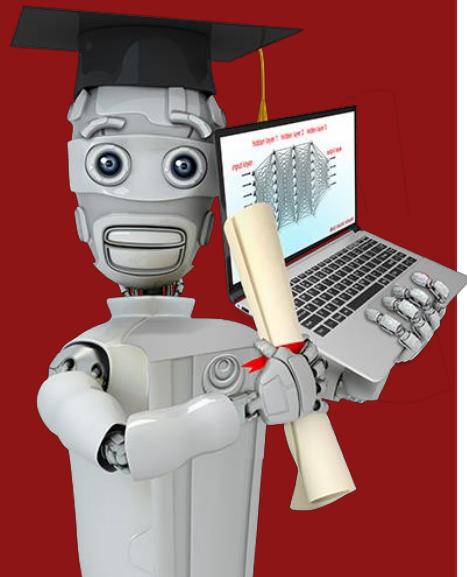
Bagged decision tree

# 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

don't do this on  $b = 1$

Given training set of size  $m$

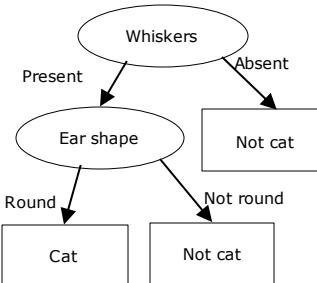
For  $b = 1$  to  $B$ :

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 ✓
Pointy	Round	Absent	Not cat ✗
Floppy	Not Round	Absent	Not cat ✓
Pointy	Round	Absent	Not cat ✗
Floppy	Round	Absent	Not cat ✓
Floppy	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

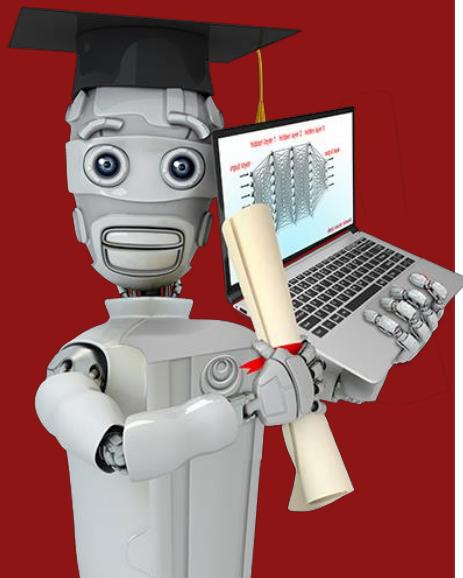
```
→from xgboost import XGBClassifier  
  
→model = XGBClassifier()  
  
→model.fit(X_train, y_train)  
→y_pred = model.predict(X_test)
```

## Regression

```
from xgboost import XGBRegressor  
  
model = XGBRegressor()  
  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)
```

# Conclusion

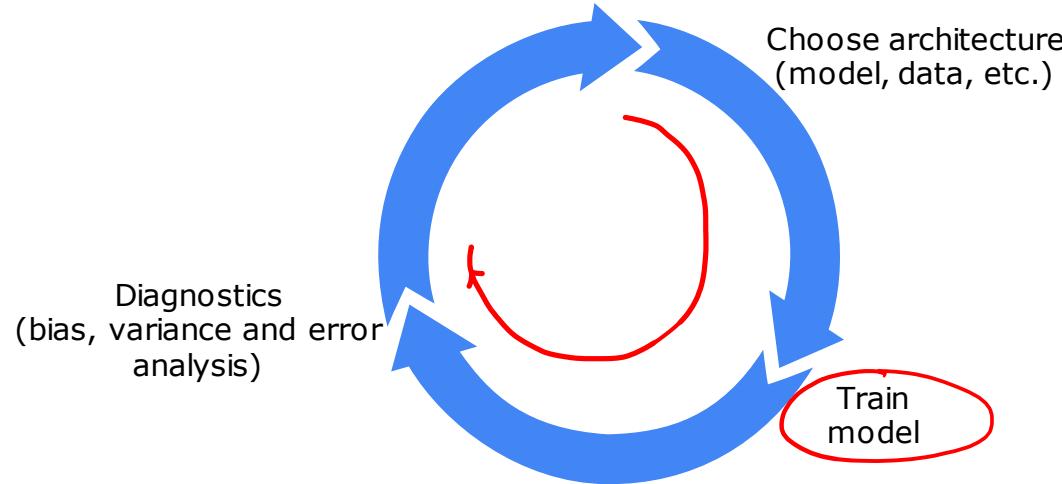
When to use decision trees



# 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

because you can train them all together but with Decision Trees, you can only train one at a time