# Advanced Learning Algorithms

## week1

### Neural networks intuition

### Neurons and the brain

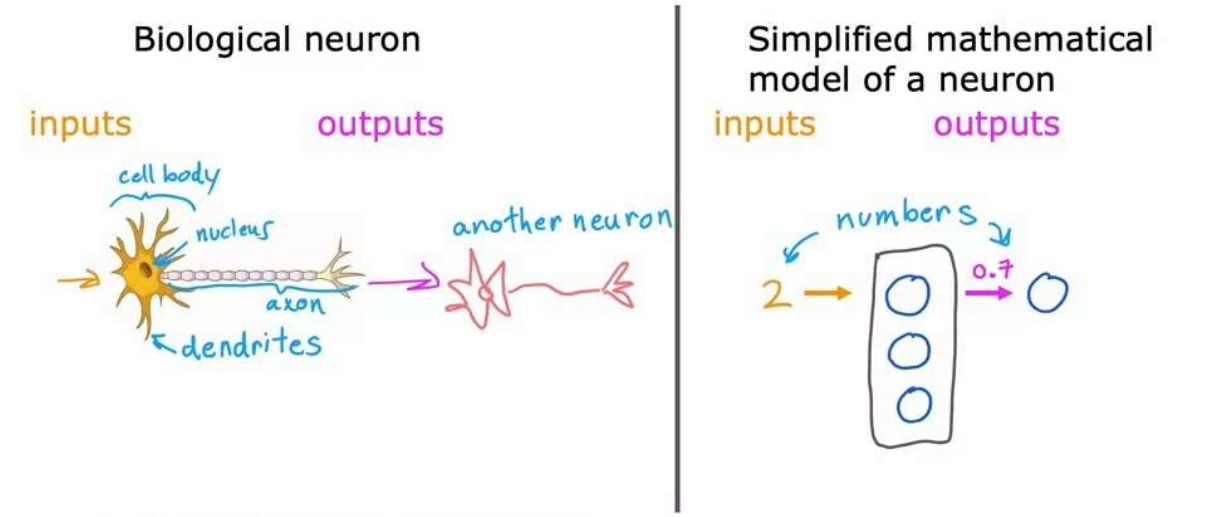
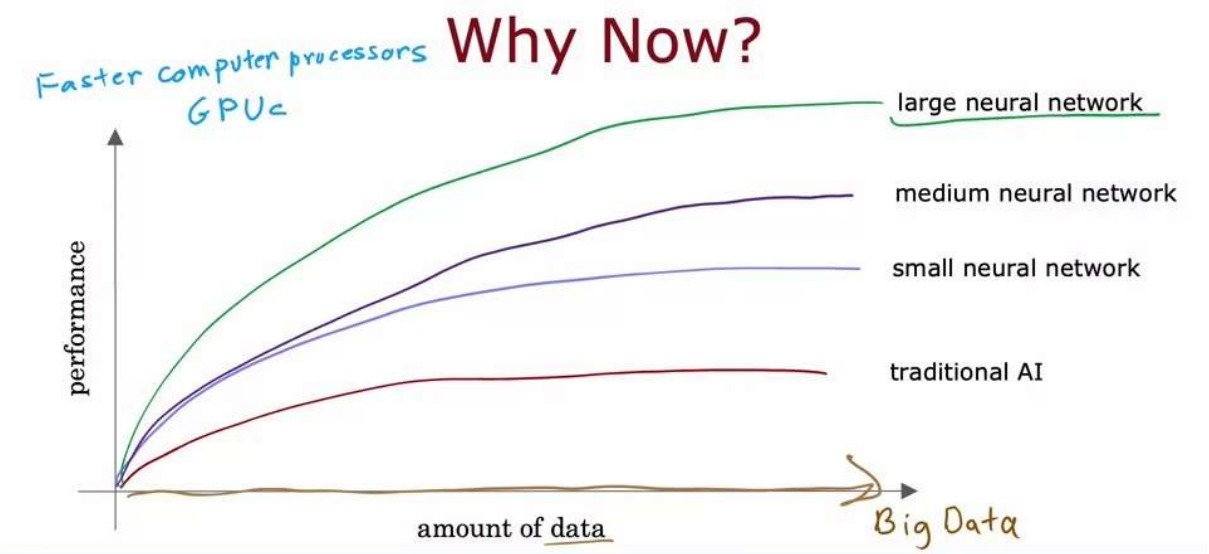
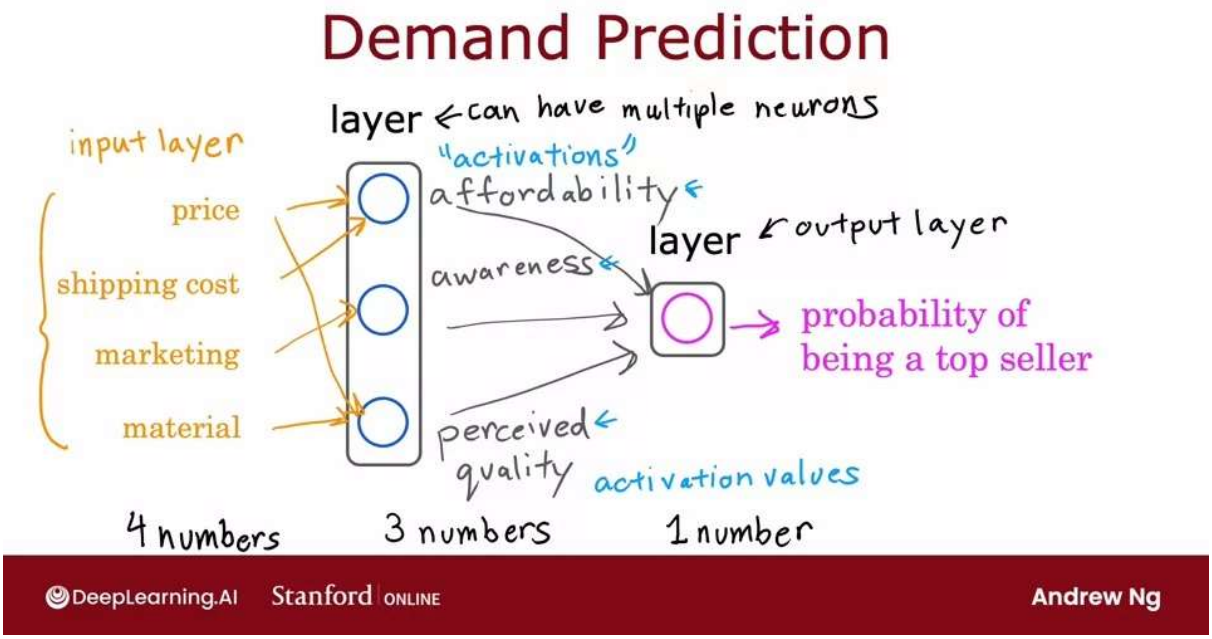


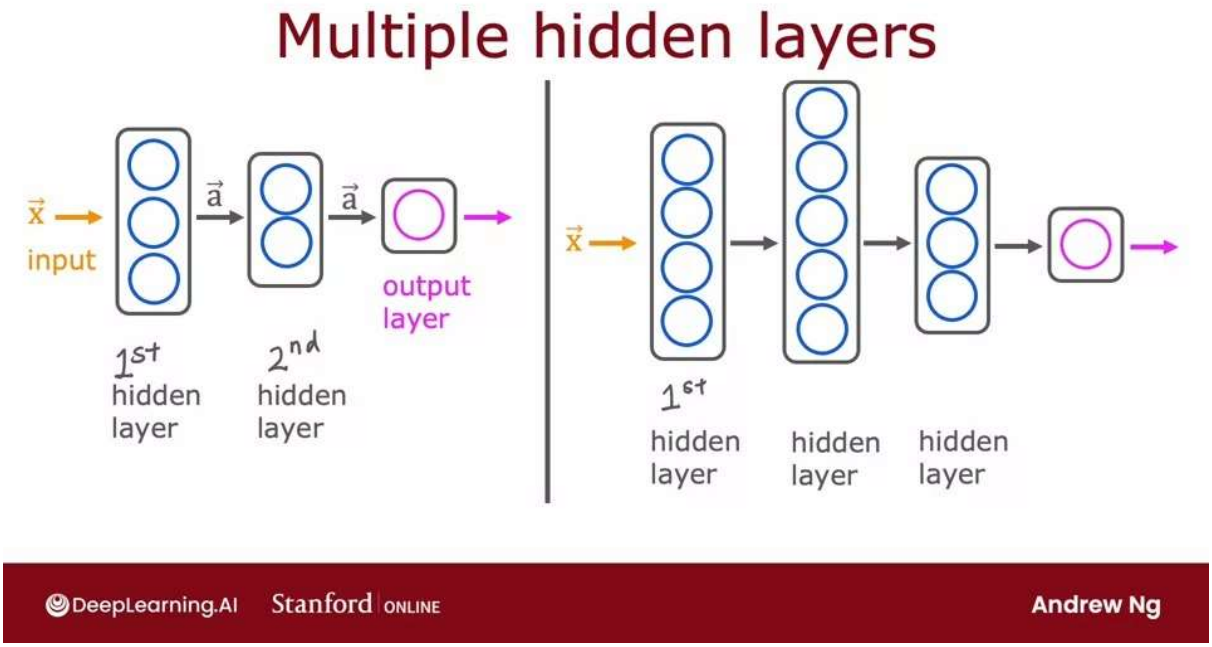

image source: https://biologydictionary.net/sensory-neuron/



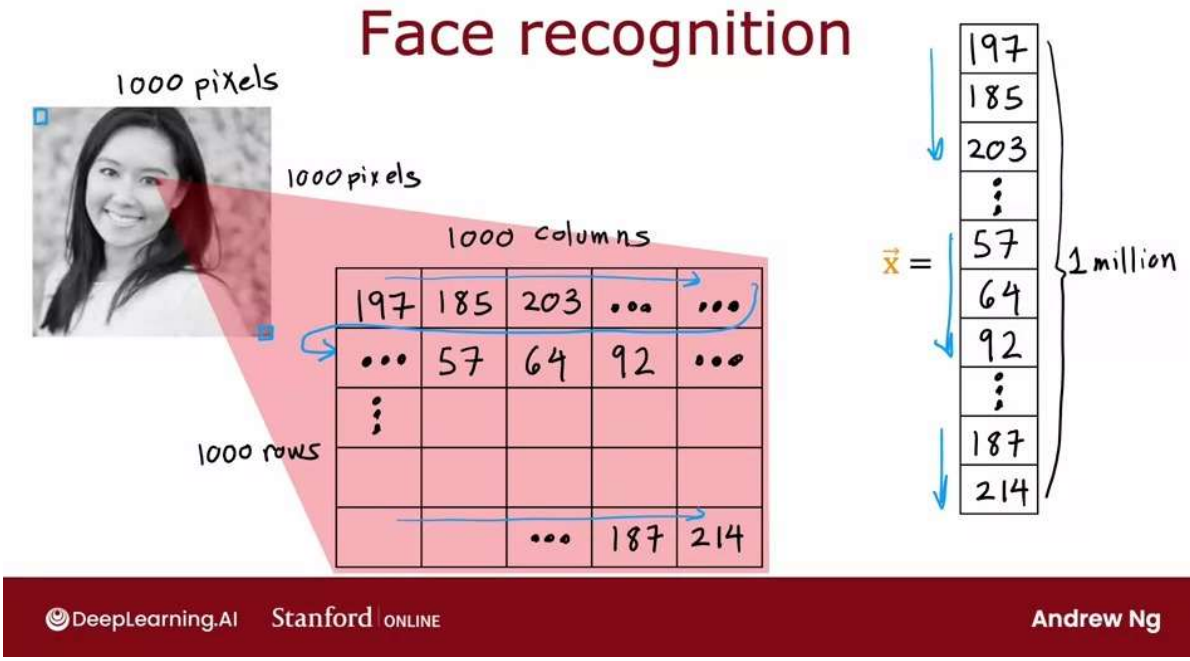


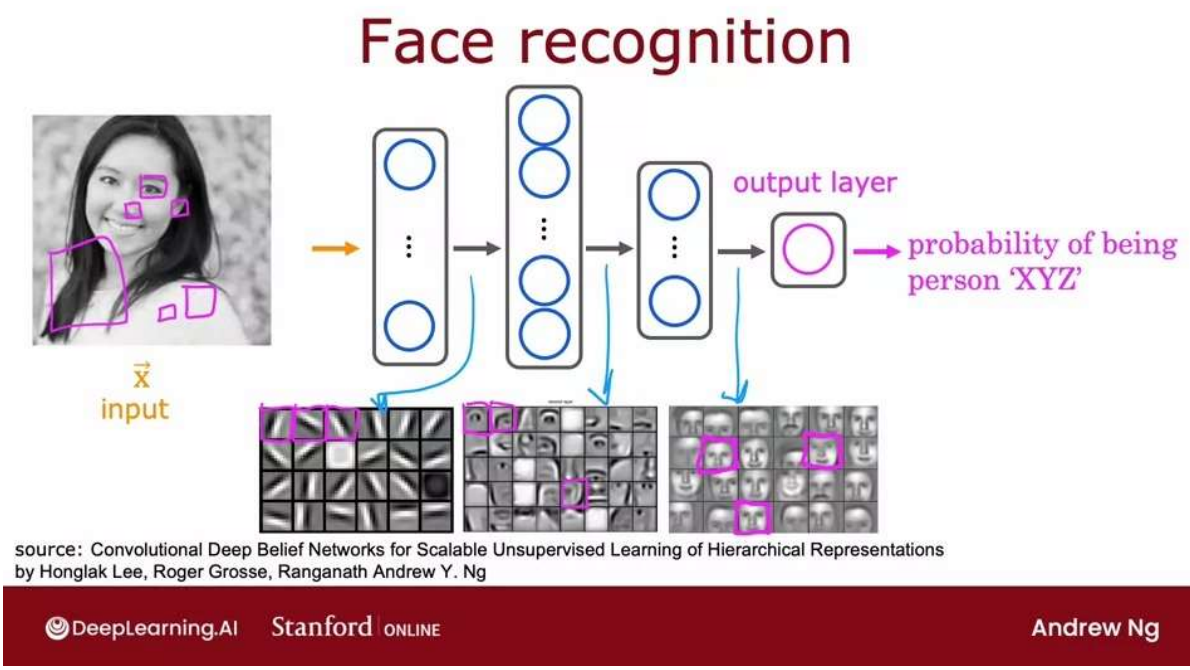
**Demand Prediction**



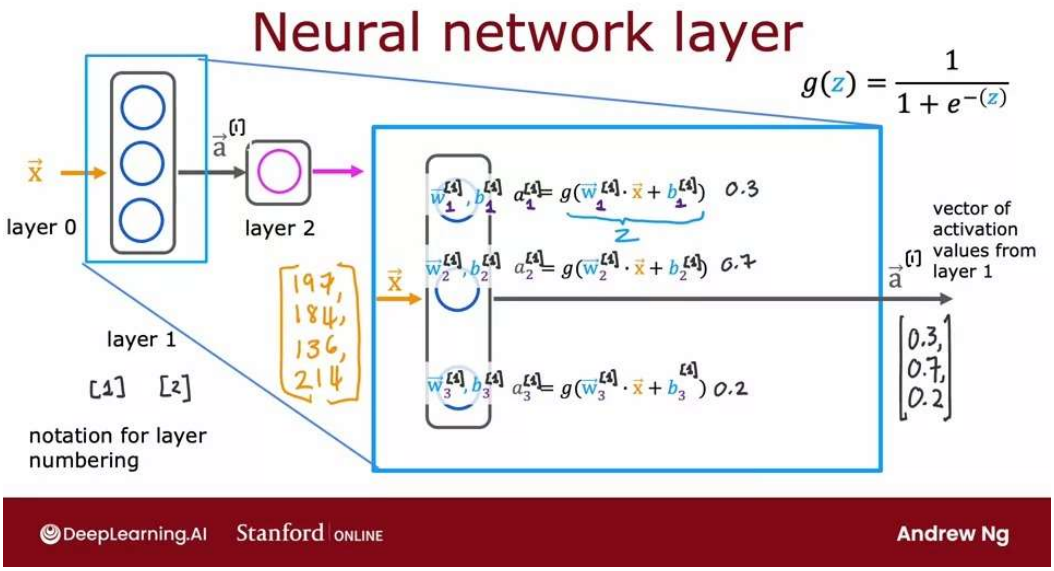
## Recognizing Images

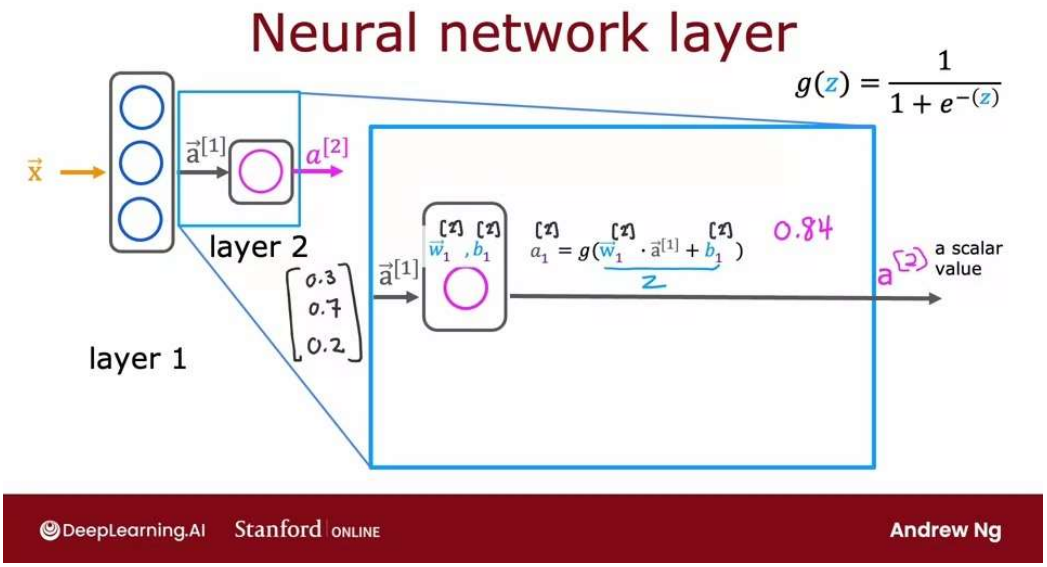


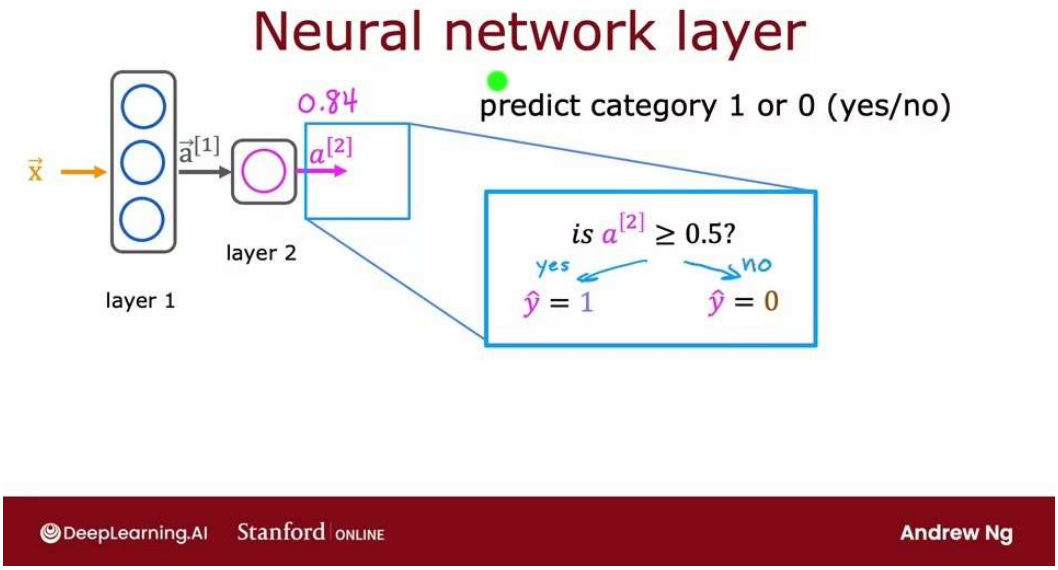




source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

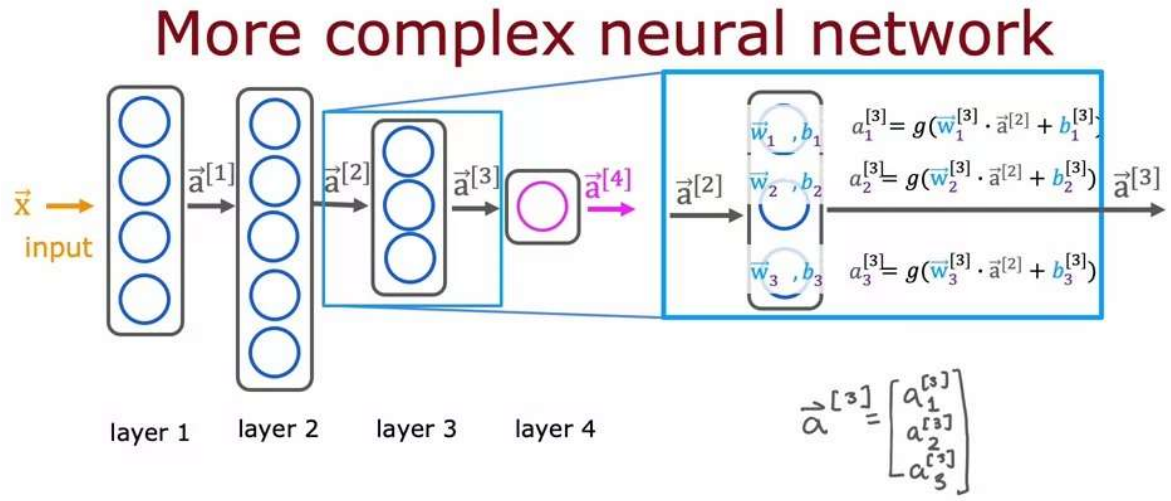
## Neural network layer

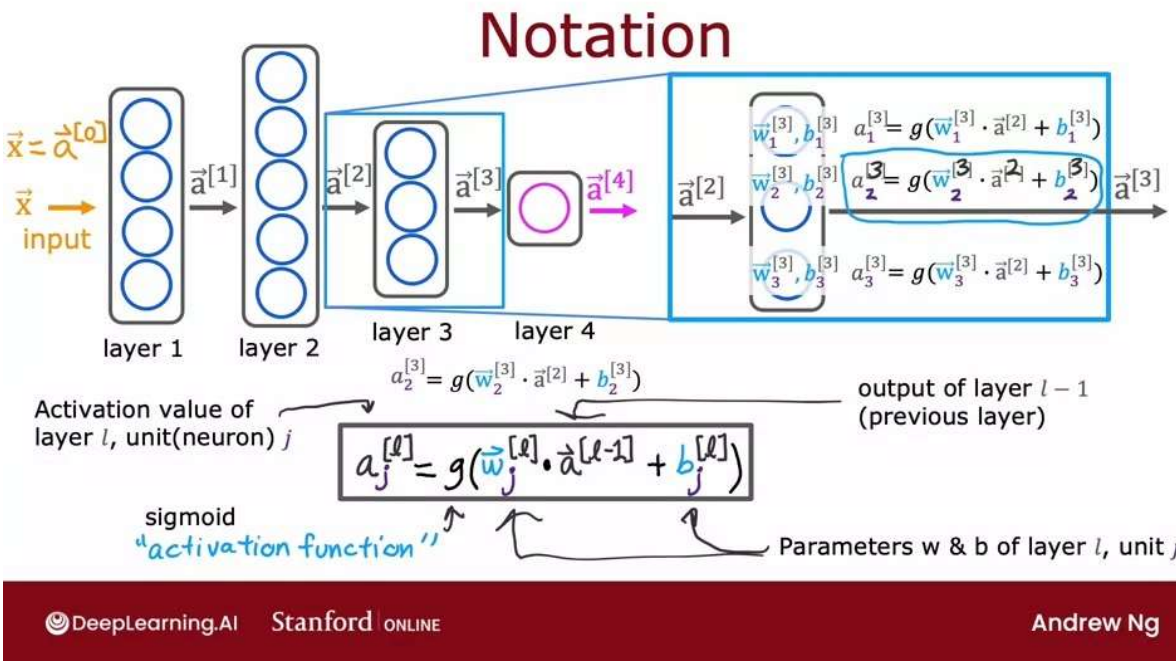


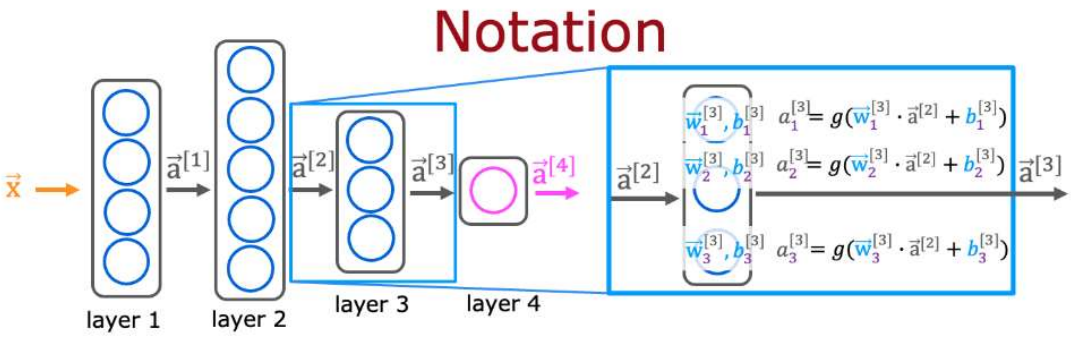



## More complex neural networks



1.

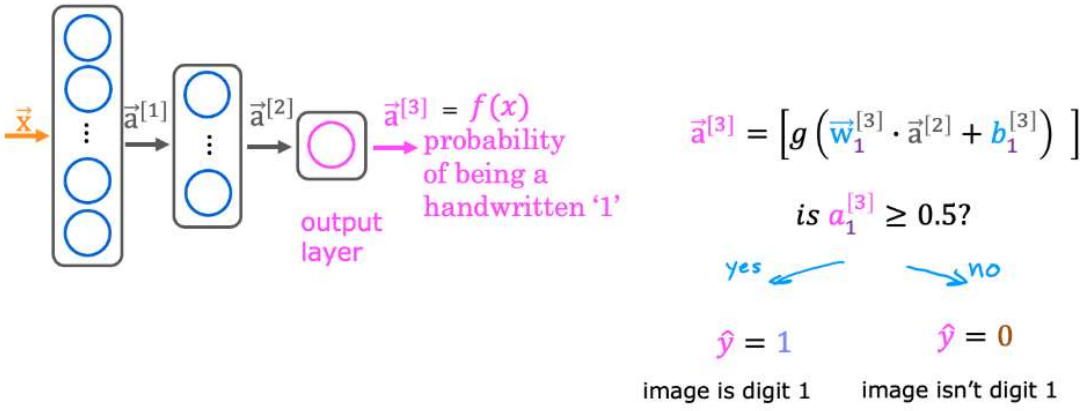



$$a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$$

For a neural network, what is the expression for calculating the activation of the third neuron in layer 2? Note, this is different from the question that you saw in the lecture video.

- ○ $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[2]} + b_3^{[2]})$
- ◉ $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[1]} + b_3^{[2]})$

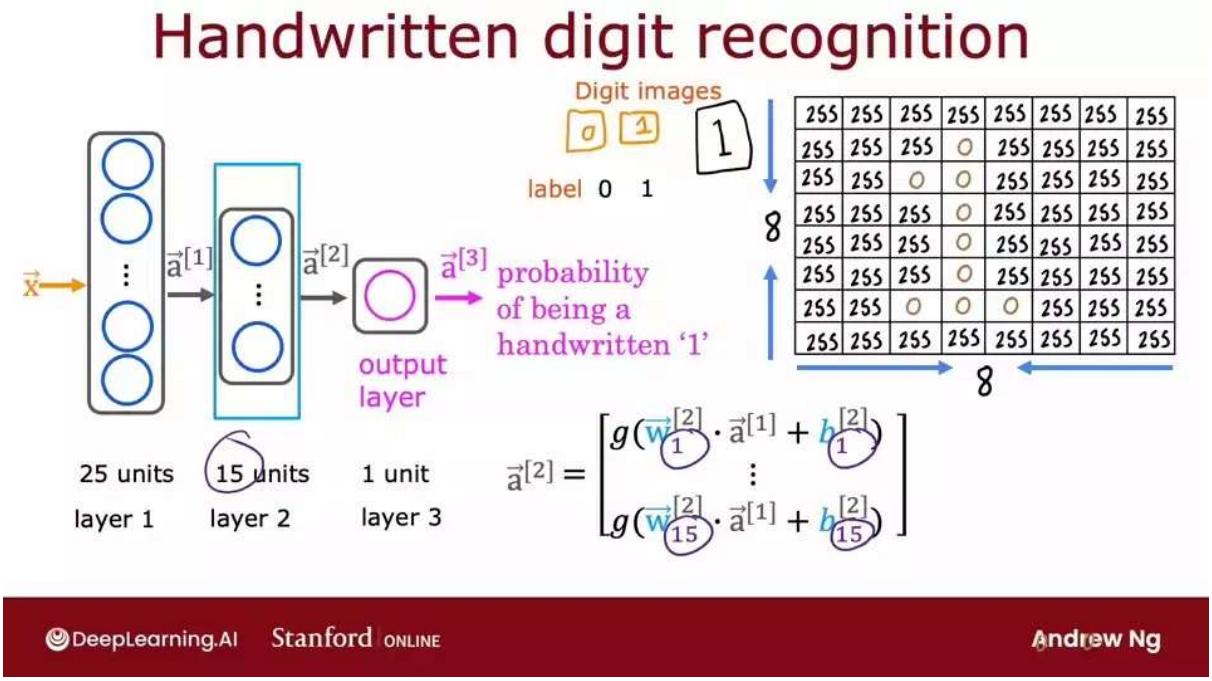
2.

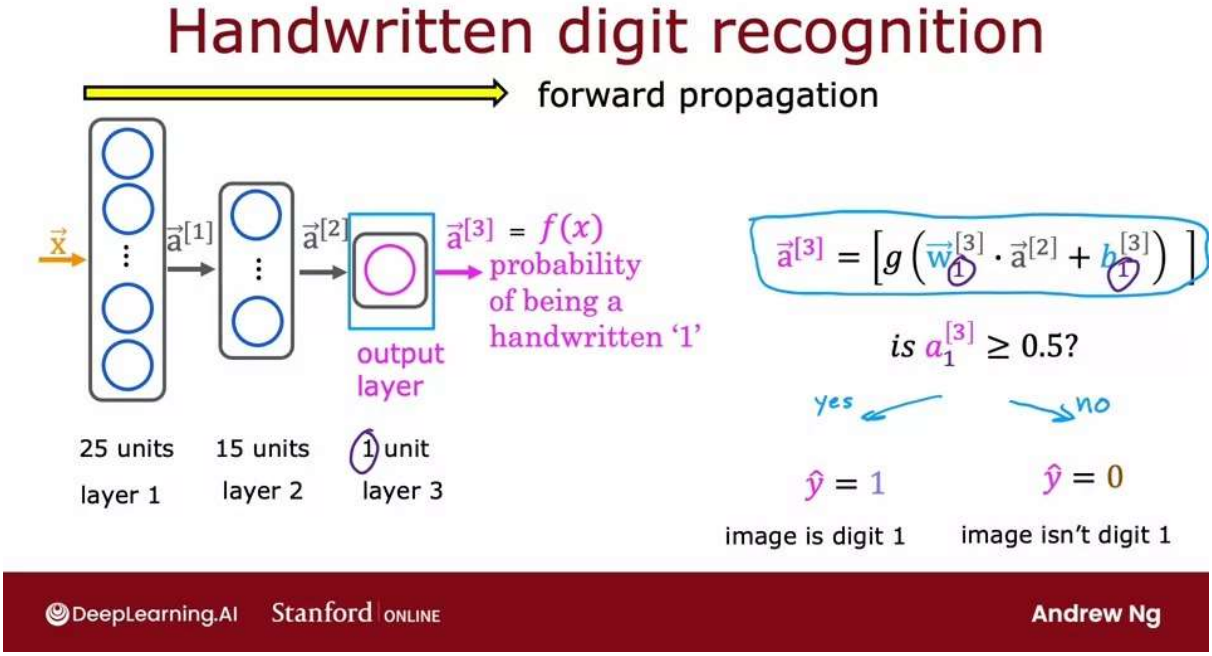



For the handwriting recognition task discussed in lecture, what is the output $a_1^{[3]}$?

- ○ A number that is either exactly 0 or 1, comprising the network's prediction
- ○ A vector of several numbers, each of which is either exactly 0 or 1
- ○ A vector of several numbers that take values between 0 and 1
- ◉ The estimated probability that the input image is of a number 1, a number that ranges from 0 to 1.

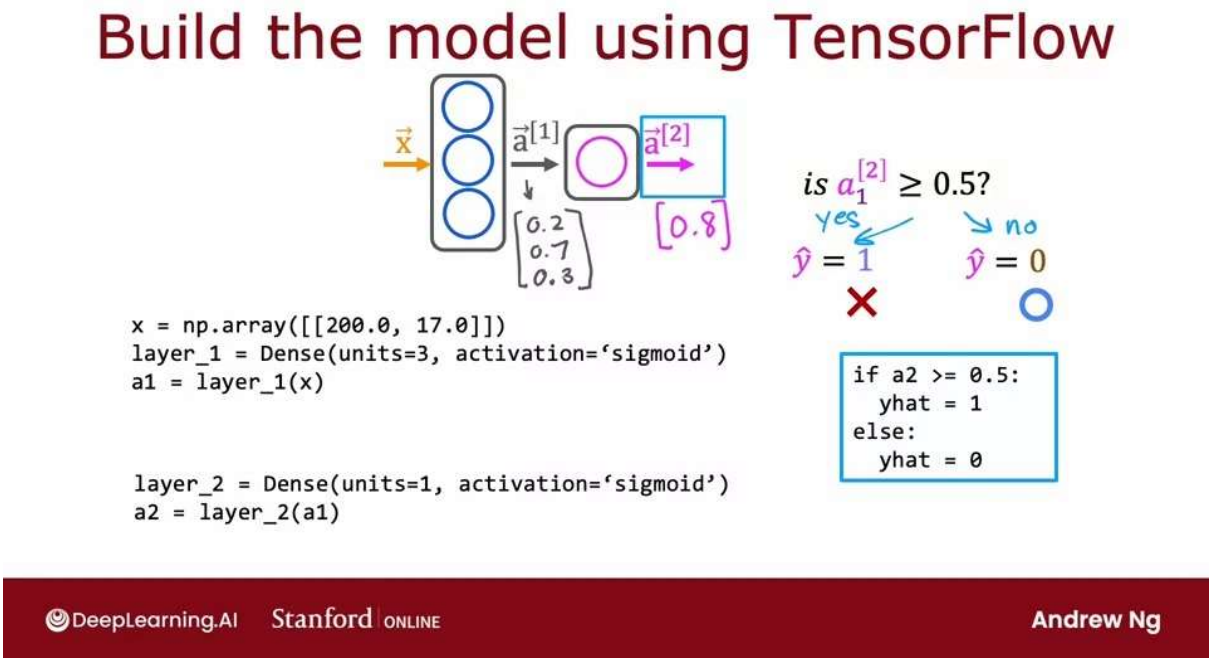
**Making predictions (forward propagation)**

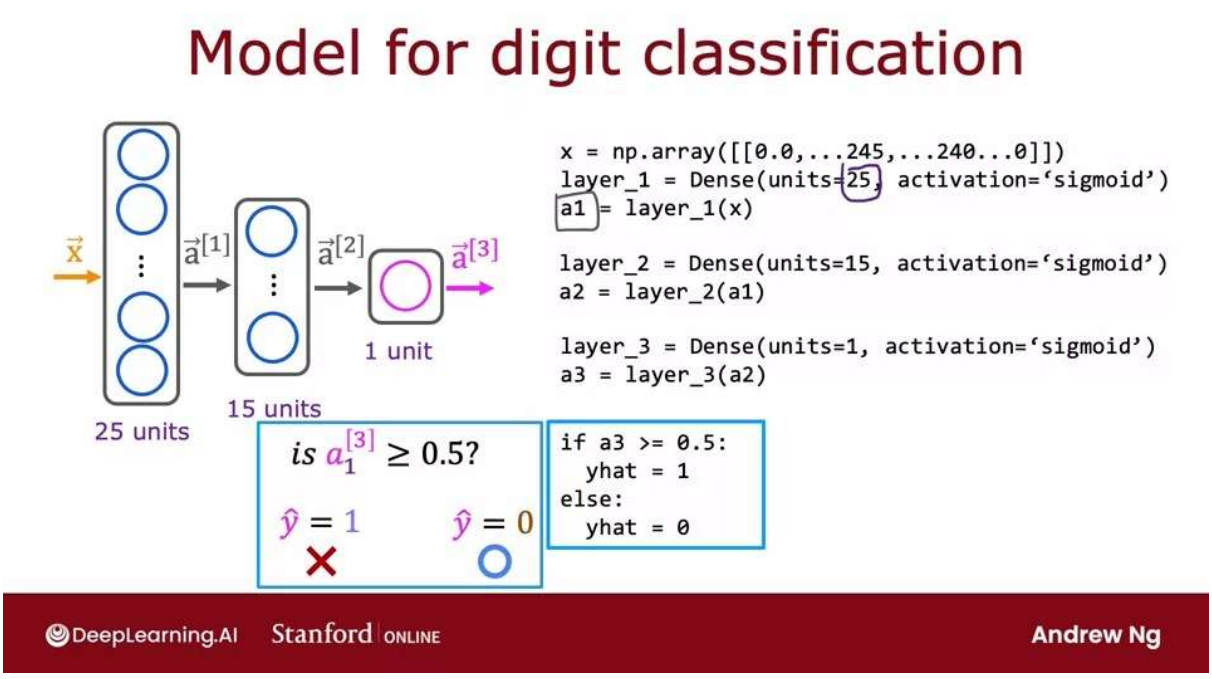


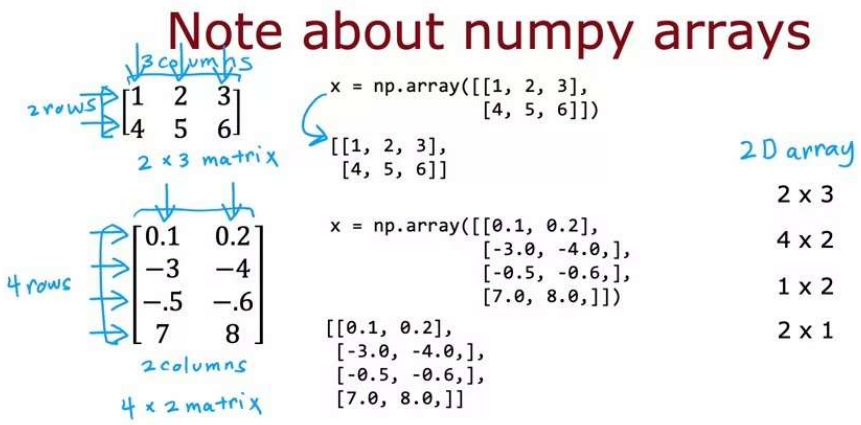
**TensorFlow implementation**

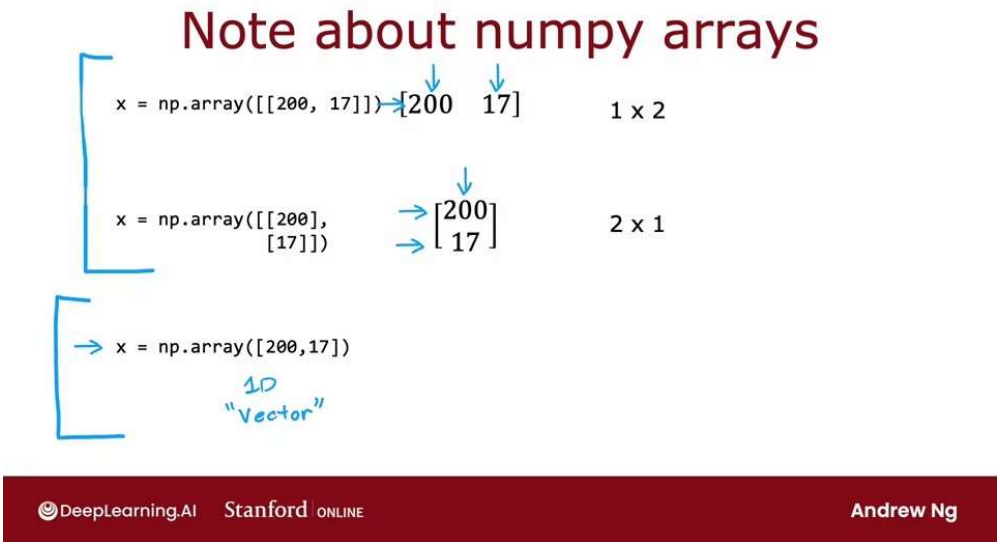
**Inference in Code**

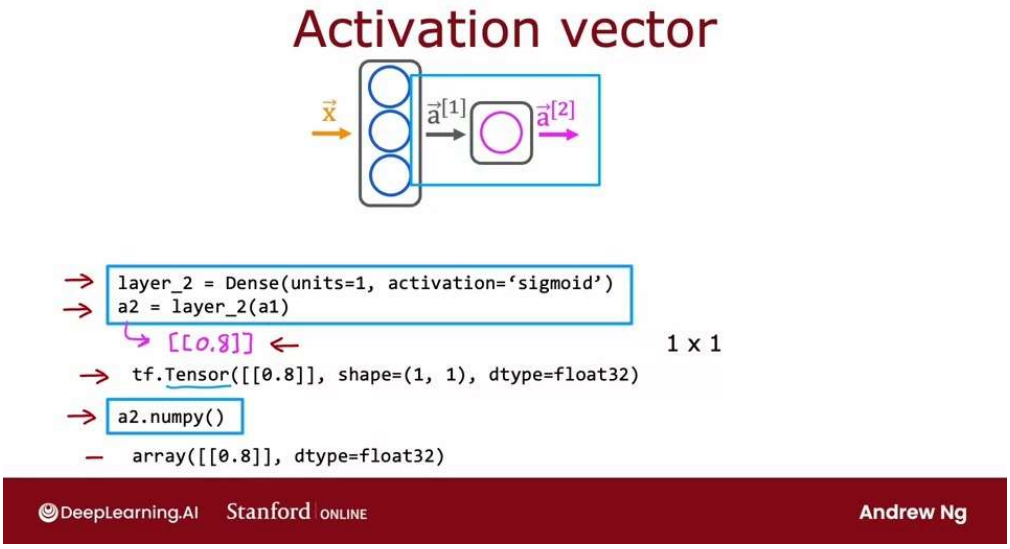


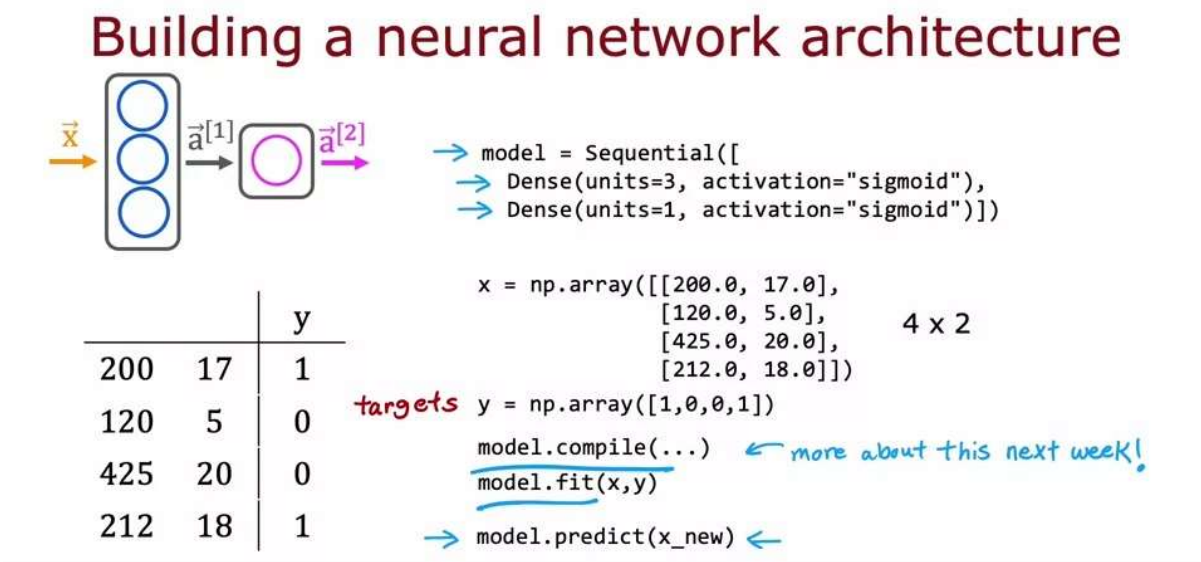
## Data in TensorFlow

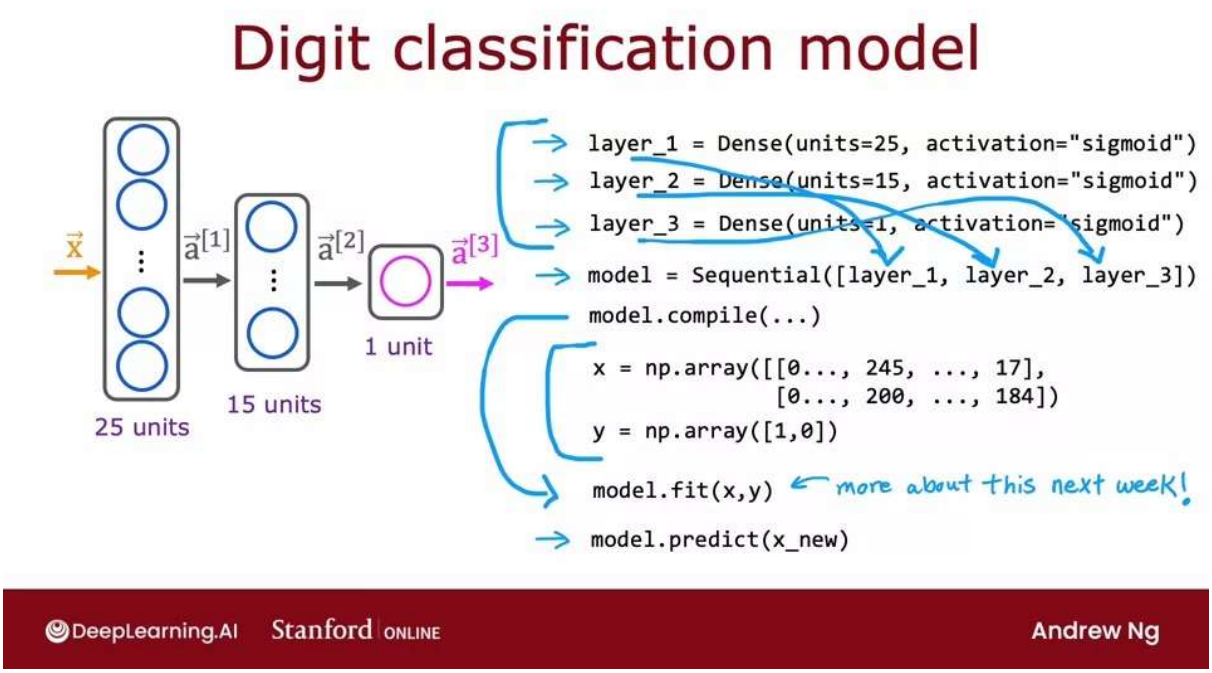


## Building a neural network



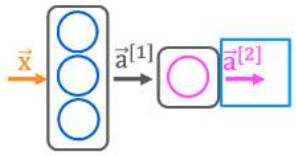



1. For the the following code:

```
model = Sequential([
 Dense(units=25, activation="sigmoid"),
 Dense(units=15, activation="sigmoid"),
 Dense(units=10, activation="sigmoid"),
 Dense(units=1, activation="sigmoid")])
```

This code will define a neural network with how many layers?

- ○ 3
- ○ 5
- ○ 25
- ◉ 4



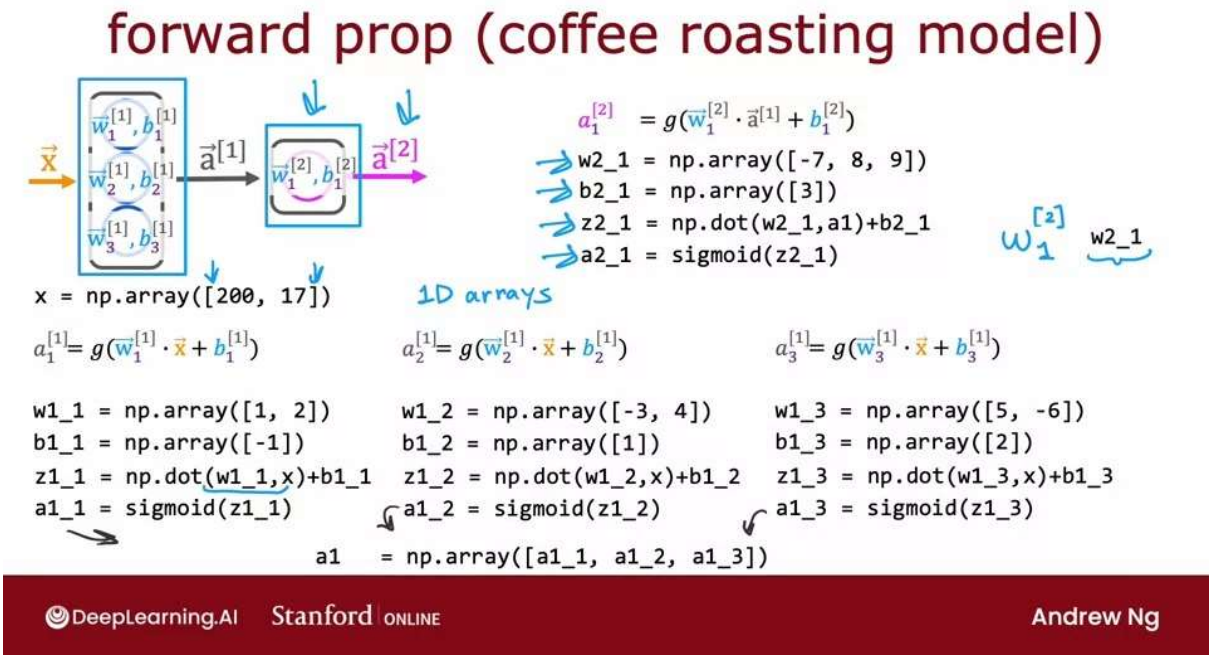

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)


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

How do you define the second layer of a neural network that has 4 neurons and a sigmoid activation?

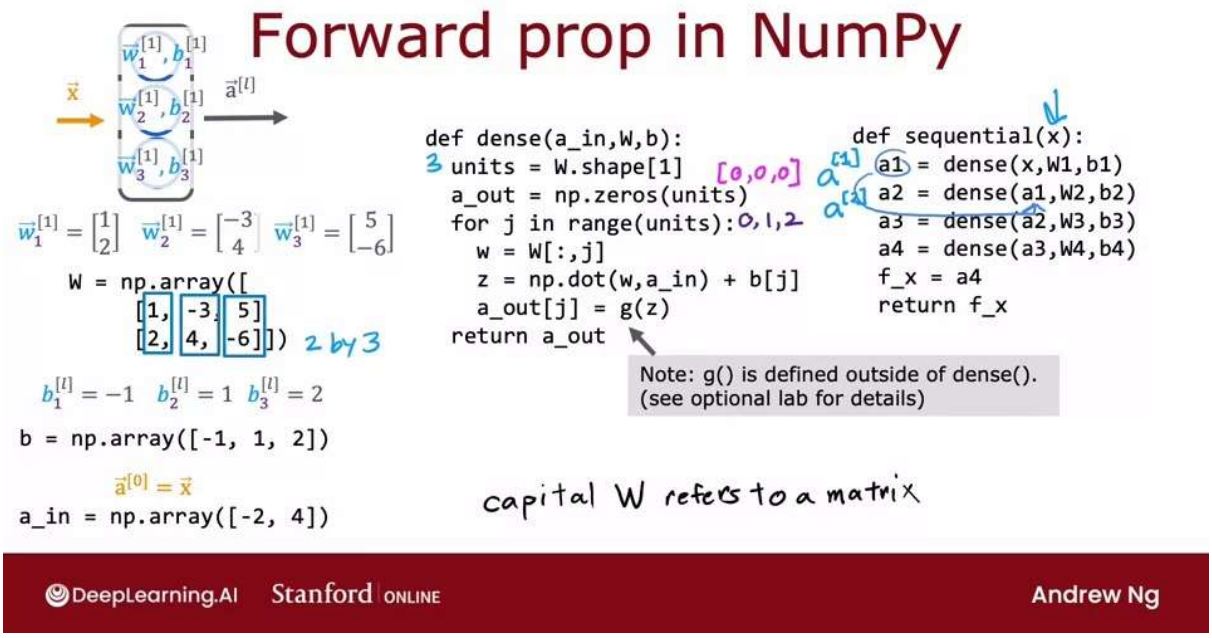
- ○ Dense(layer=2, units=4, activation = 'sigmoid')
- ◉ Dense(units=4, activation='sigmoid')
- ○ Dense(units=4)
- ○ Dense(units=[4], activation=['sigmoid'])

## Neural network implementation in Python

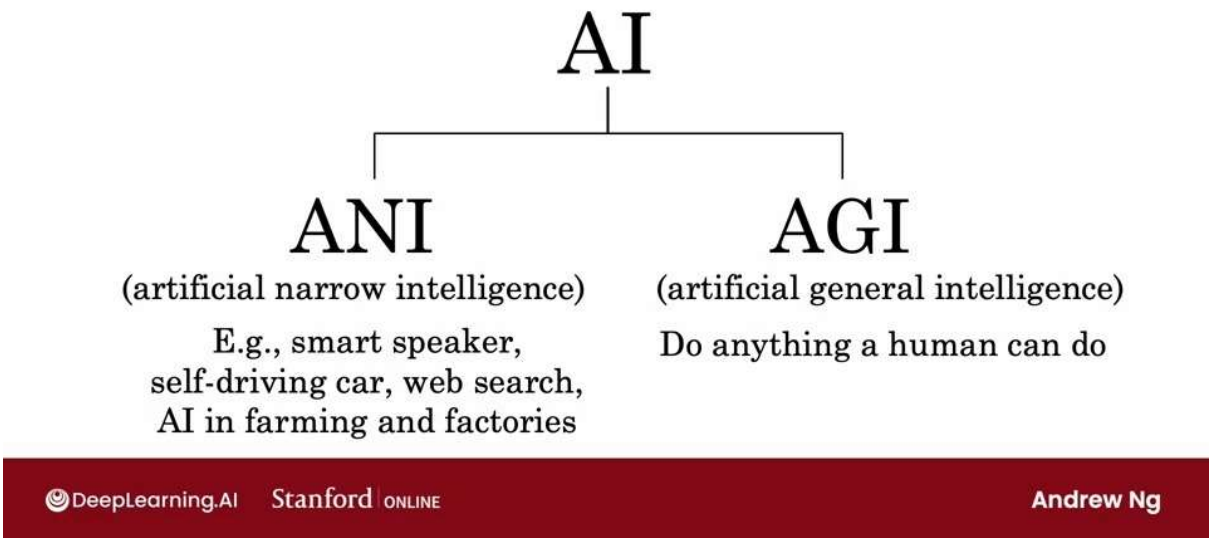
### Forward prop in a single layer

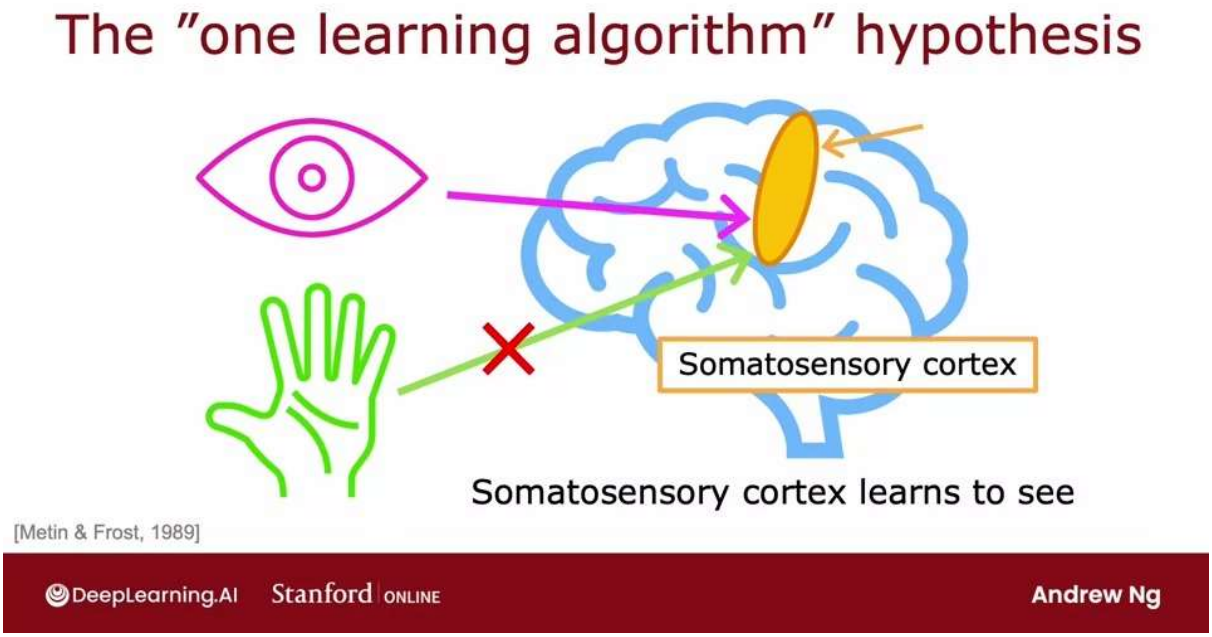



### General implementation of forward propagation

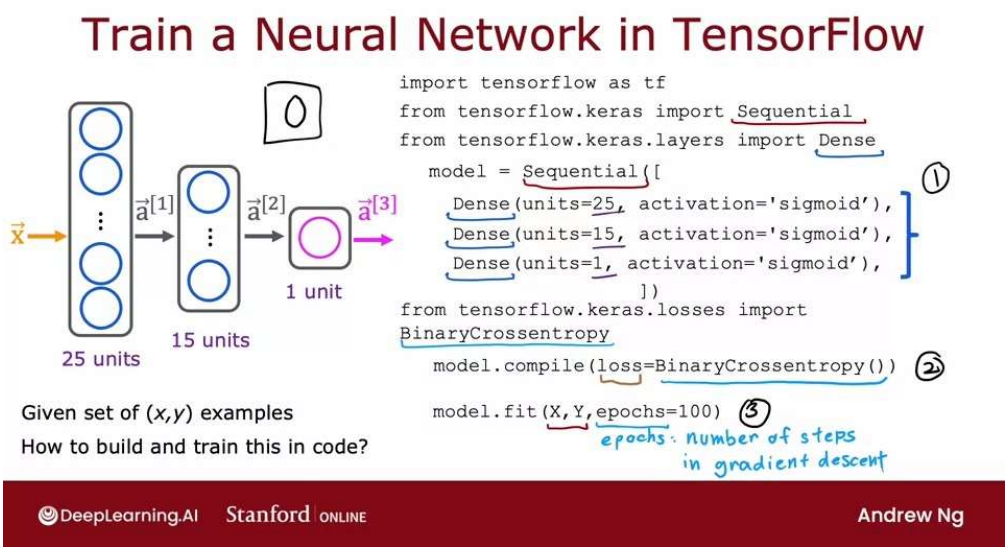


## Artificial general intelligence (AGI)

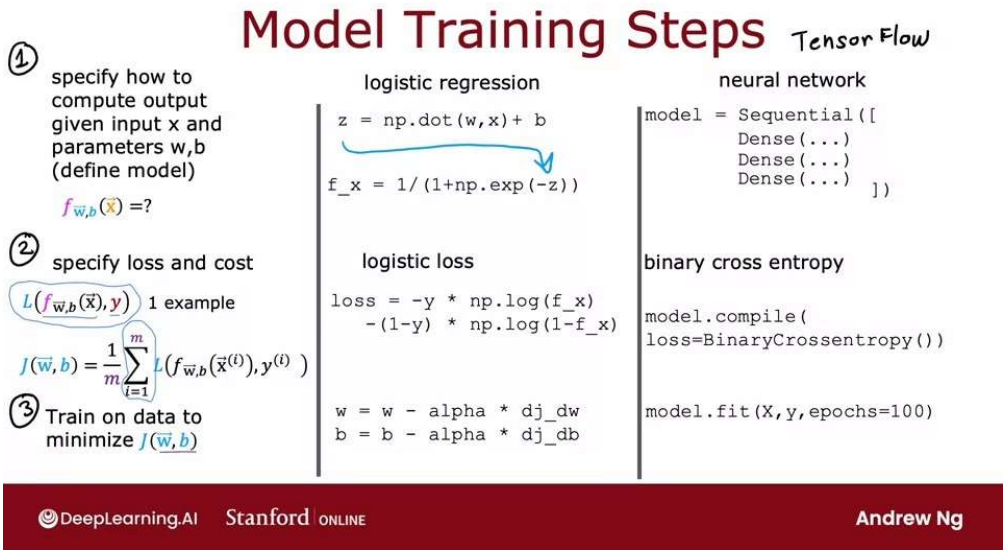


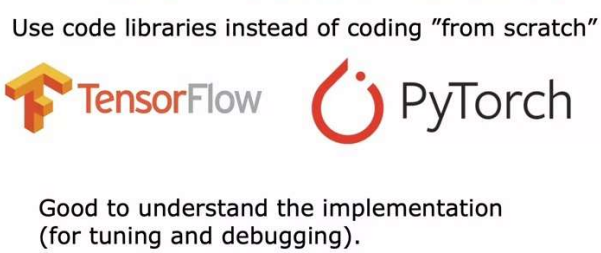



**Week 2**
**Neural Network Training**
**TensorFlow implementation**




Step1 - specify the model
Step2 - compile model
Step3 - train model

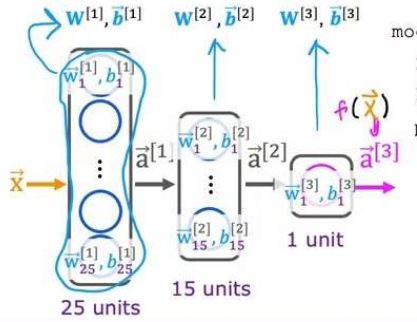


# 1. Create the model

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

handwritten 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

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

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

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

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

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

Keras

regression
(predicting numbers and not categories)

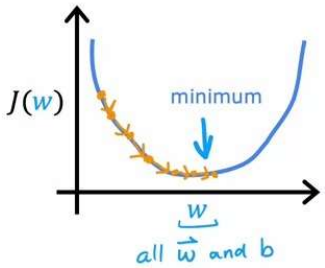
mean squared error

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

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



# 3. Gradient descent




all $\vec{w}$ and $b$

repeat {

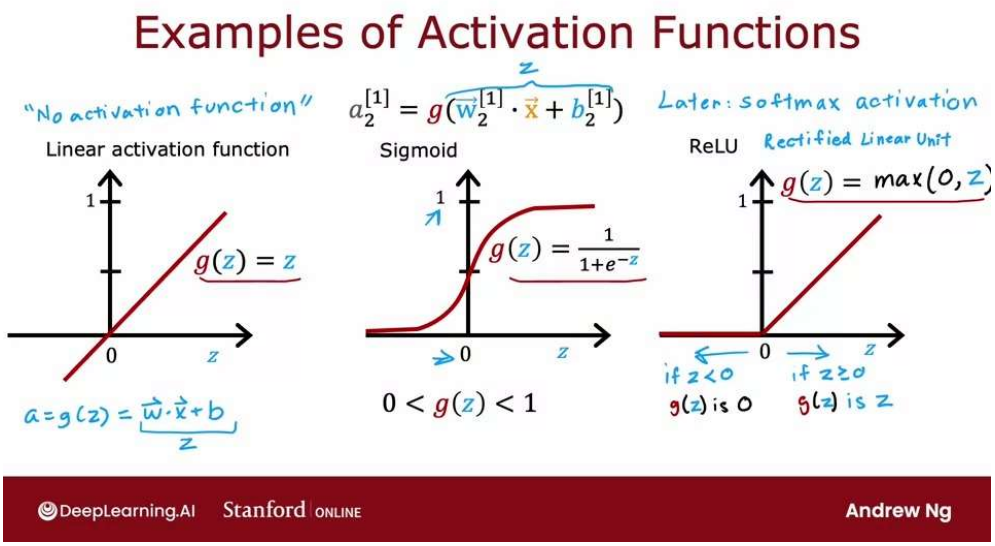
$$w_j^{[l]} = w_j^{[l]} - \alpha\frac{\partial}{\partial w_j}J(\vec{\mathbf{w}}, b)$$

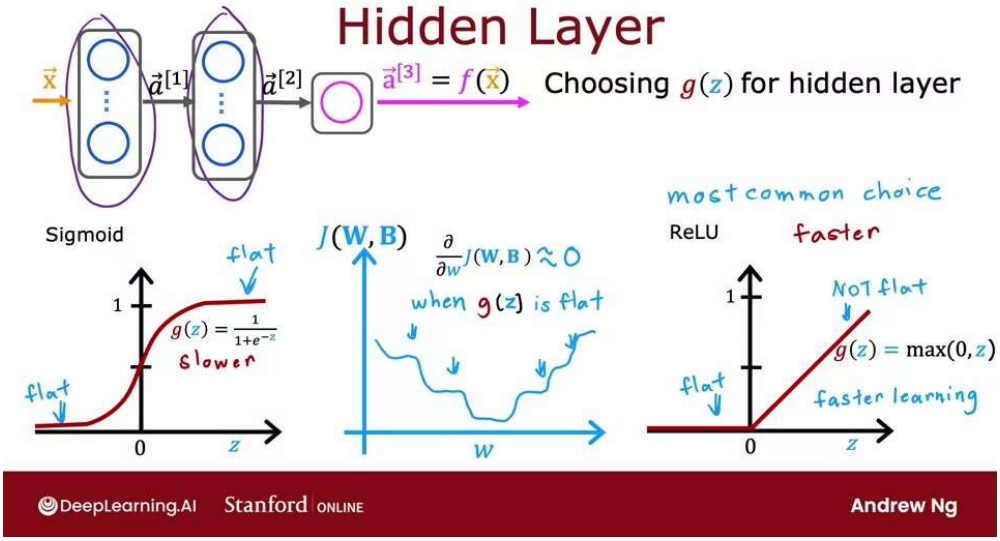
$$b_j^{[l]} = b_j^{[l]} - \alpha\frac{\partial}{\partial bj}J(\vec{\mathbf{w}}, b)$$

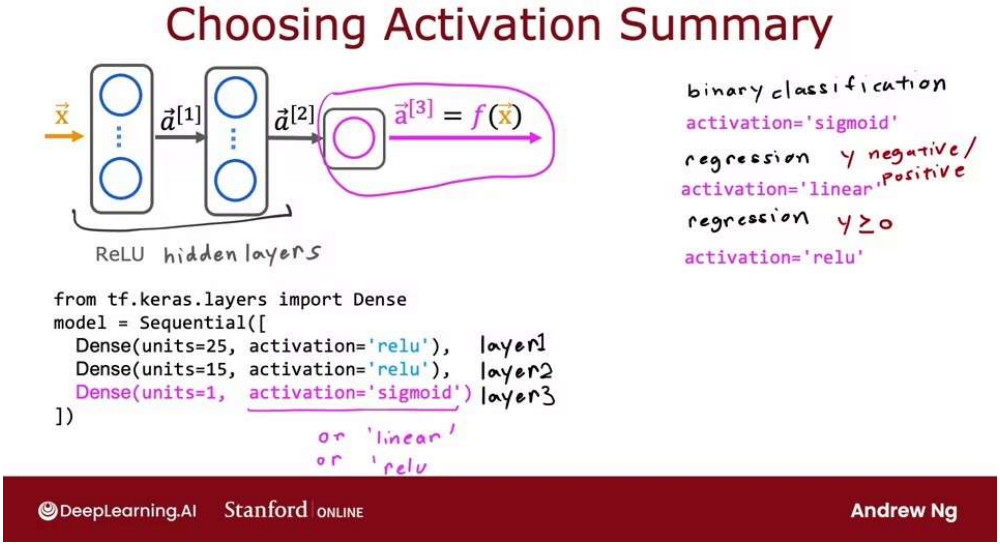
} Compute derivatives for gradient descent using "back propagation"

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

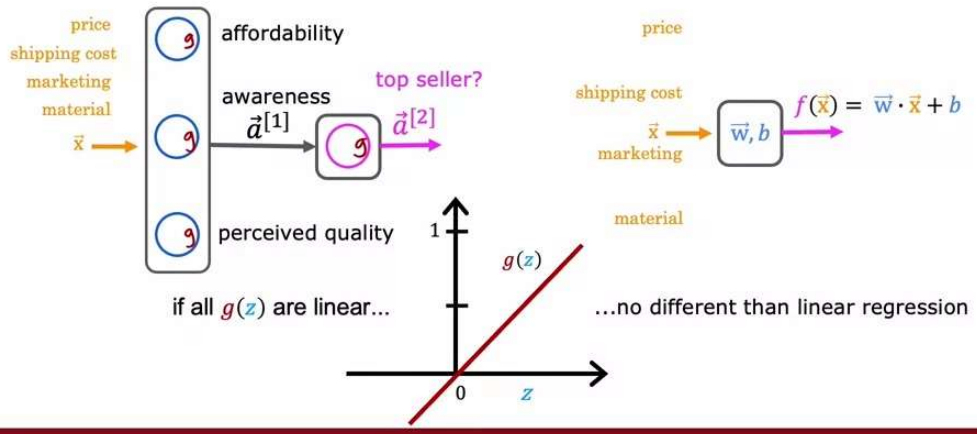
## Activation Functions



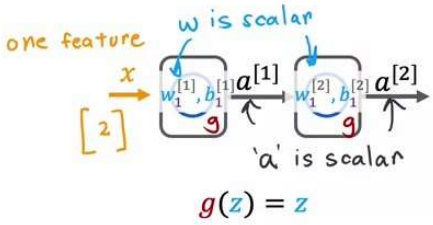



# Why do we need activation functions?






# Linear Example




$$a^{[1]} = w_1^{[1]} x + b_1^{[1]}$$

$$a^{[2]} = w_1^{[2]} a^{[1]} + b_1^{[2]}$$

$$= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]}$$

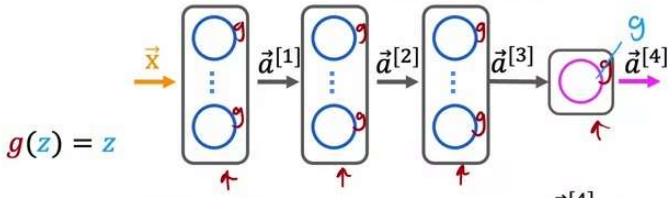
$$\vec{a}^{[2]} = (\underbrace{\vec{w}_1^{[2]} \vec{w}_1^{[1]}}_{w}) x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b}$$

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

$f(x) = wx + b$ linear regression



# Example




$g(z) = z$

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

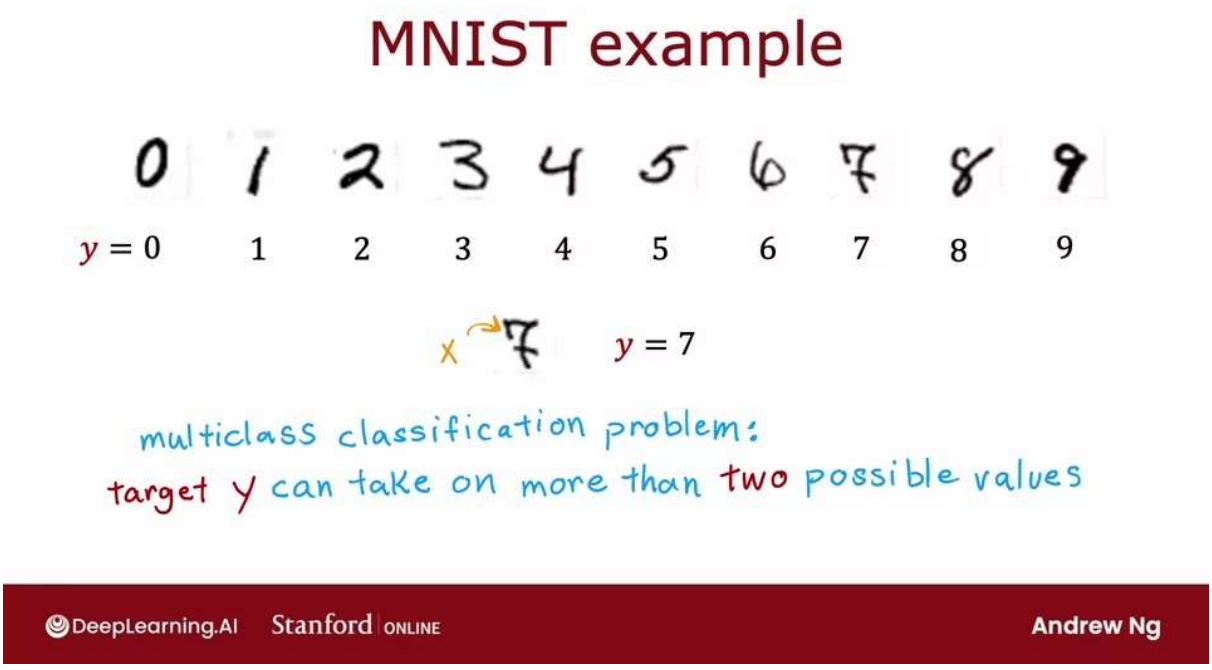
all linear (including output)
↳ equivalent to linear regression

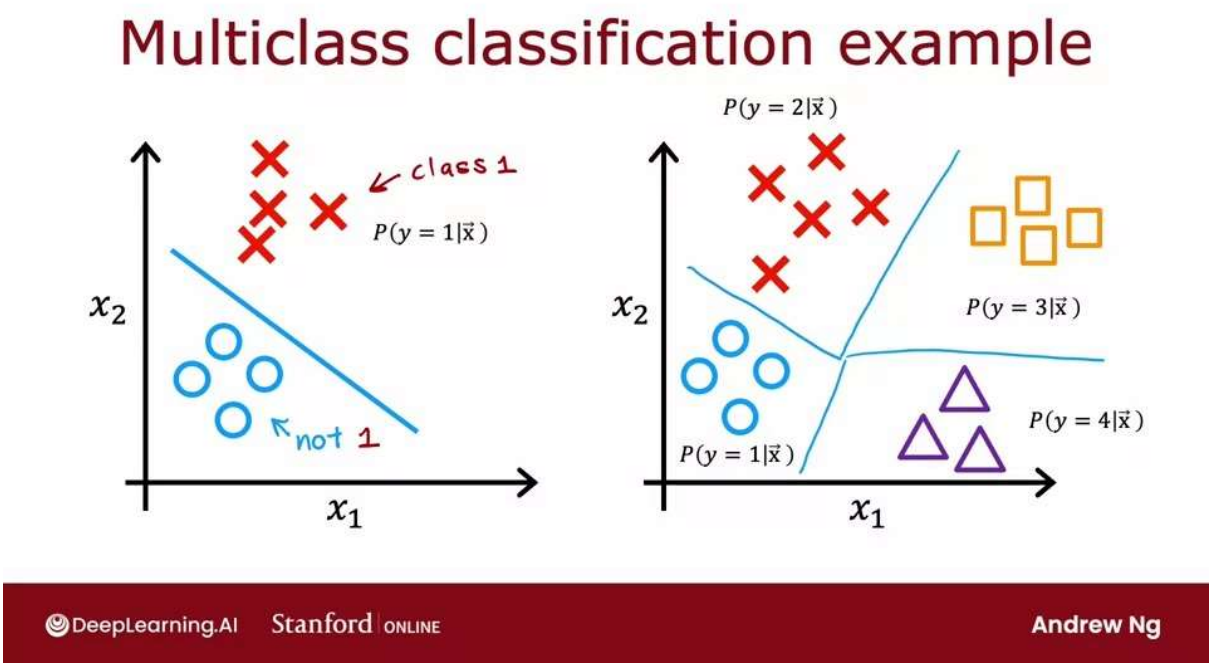
$$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]})}}$$

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

Don't use linear activations in hidden layers

**Multiclass Classification**



## Softmax

Logistic regression
(2 possible output values)

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

$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$ 0.71

$a_2 = 1 - a_1 = P(y=0|\vec{x})$ 0.29

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

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

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

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

note: $a_1 + a_2 + \ldots + a_N = 1$

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

DeepLearning.AI Stanford ONLINE Andrew Ng

# Cost

## Logistic regression

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

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

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

$loss = -y \log a_1 - (1-y) \log(1-a_1)$

if $y=1$ if $y=0$

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

## Softmax regression

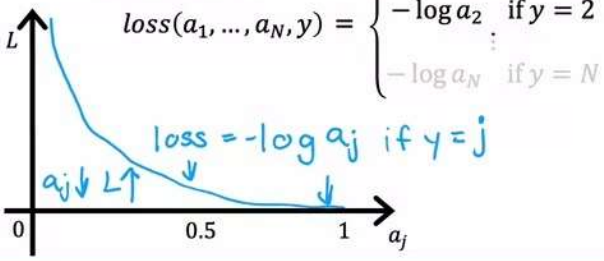
$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$

$\vdots$

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

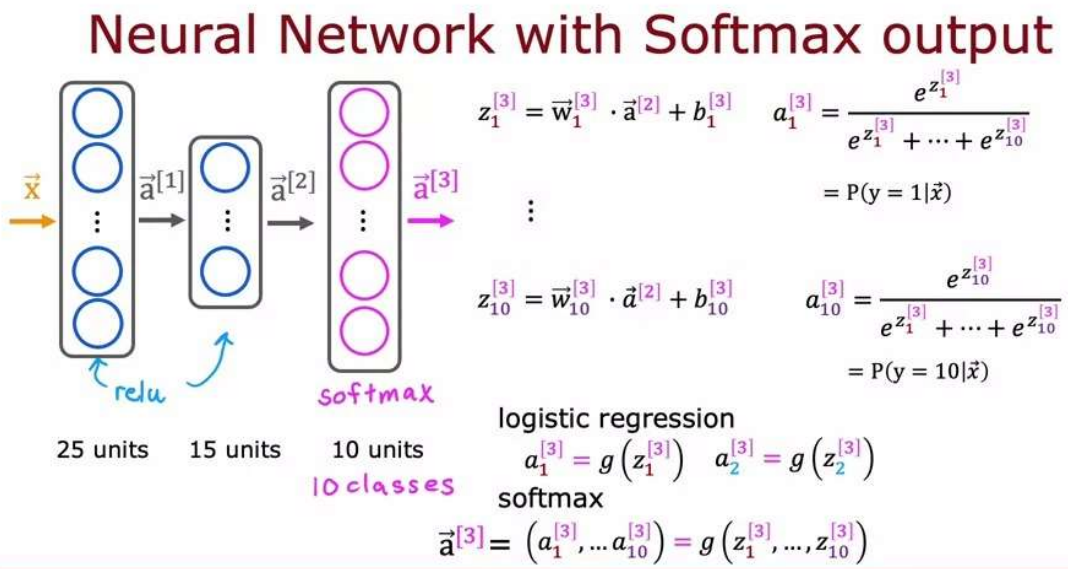
*Crossentropy loss*

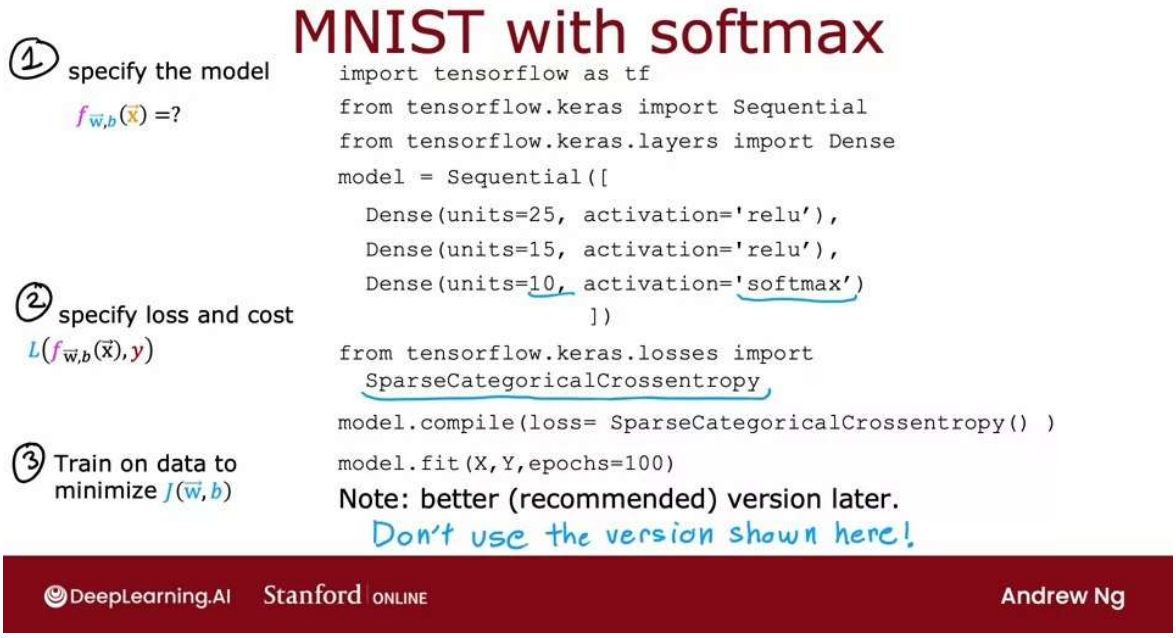
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$$




DeepLearning.AI Stanford ONLINE Andrew Ng

## Neural Network with Softmax output



## Improved implementation of softmax

# Numerical Roundoff Errors

option 1

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

option 2

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

DeepLearning.AI Stanford ONLINE Andrew Ng

# Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

$$loss = -y\log(a) - (1-y)\log(1-a)$$

More accurate loss (in code)

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

logit: z

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=1, activation='sigmoid')
                  ])
model.compile(loss=BinaryCrossEntropy() )
```

'linear'

```
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

DeepLearning.AI Stanford ONLINE Andrew Ng

# More numerically accurate implementation of softmax

Softmax regression

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

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

More Accurate

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

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

'linear'

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

```
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

DeepLearning.AI Stanford ONLINE Andrew Ng

# 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)
f_x = tf.nn.softmax(logits)
```

not $a_1 \dots a_{10}$ is $z_1 \dots z_{10}$



# logistic regression (more numerically accurate)

**model**

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

**loss**

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

**fit**

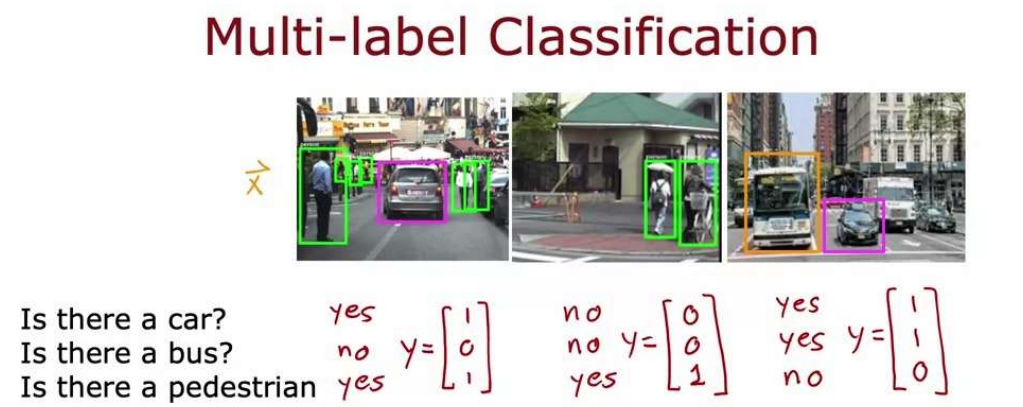
```
logit = model(X)
```

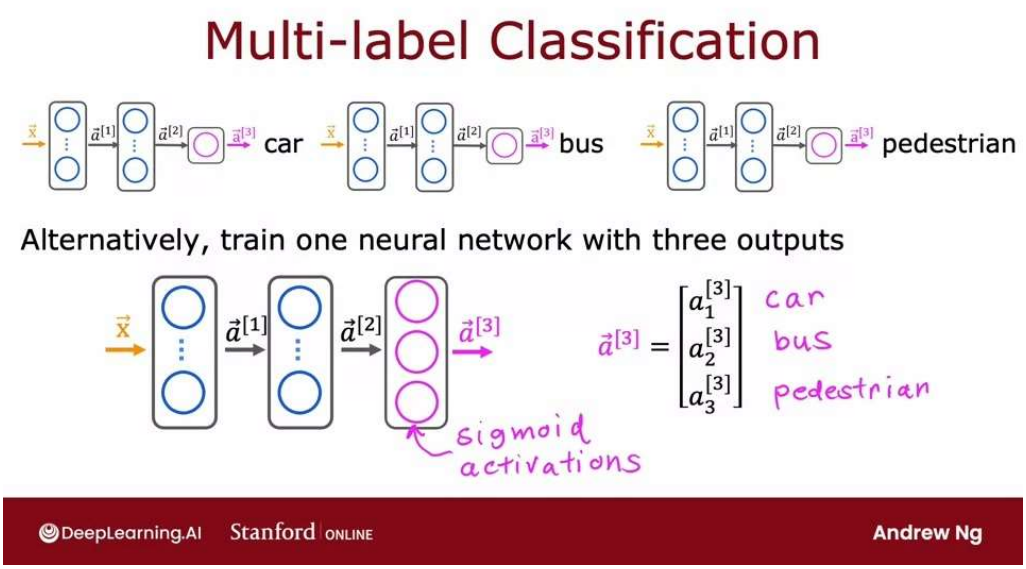
$z$

**predict**

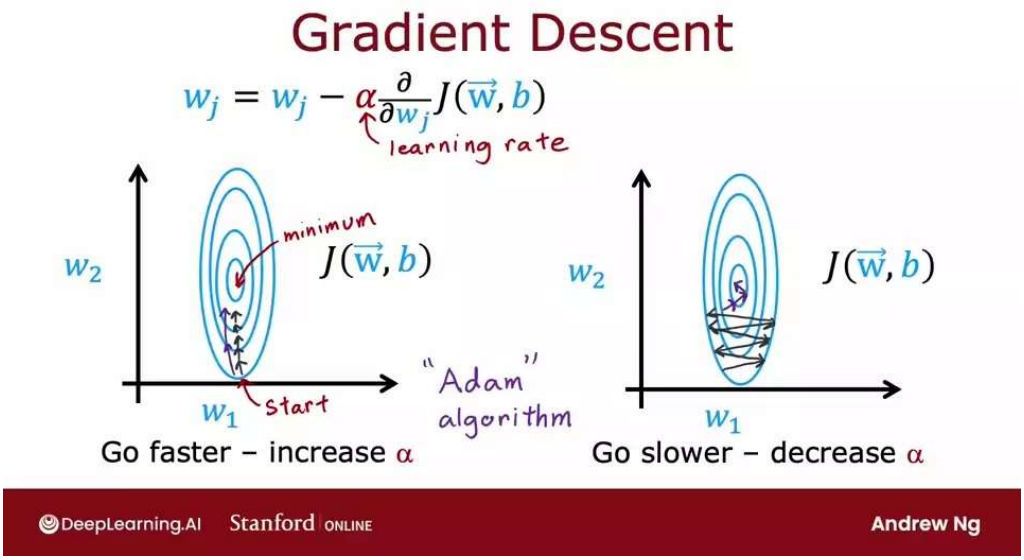
```
f_x = tf.nn.sigmoid(logit)
```

**Multilabel classification**




**Advanced Optimization**



# Adam Algorithm Intuition

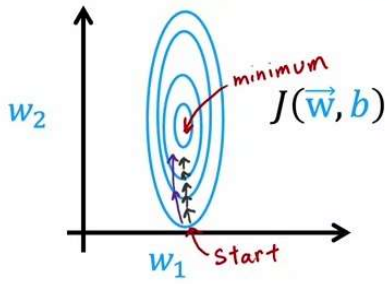
Adam: Adaptive Moment estimation

not just one $\alpha$

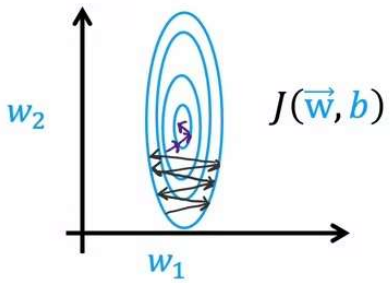
$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$
$$\vdots$$
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$



# 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

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

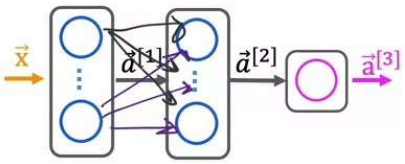
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

## Additional Layer Types

### Dense Layer

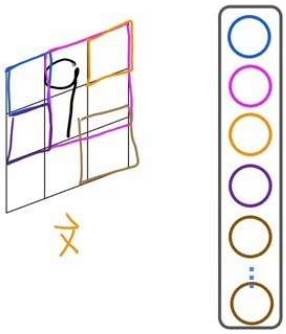



Each neuron output is a function of all the activation outputs of the previous layer.

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



### Convolutional Layer

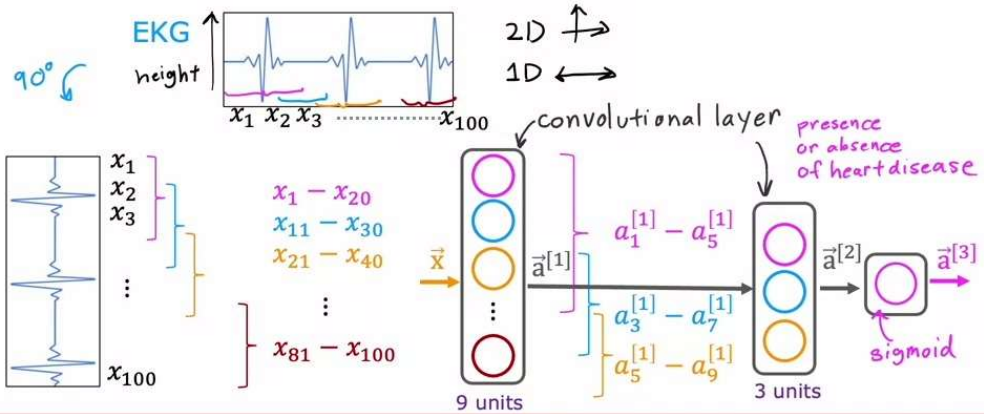



Each neuron only looks at part of the previous layer's outputs.

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



### Convolutional Neural Network



**Week 3**

**Advice for applying machine Learning**

# Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

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

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

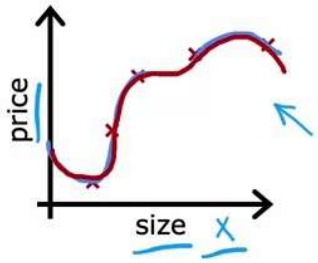
Diagnostic:

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 a model**

# Evaluating your model




Model fits the training data well but will fail to generalize to new examples not in the training set.

$$x = x_1$$

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

- $x_1$ = size in feet$^2$
- $x_2$ = no. of bedrooms
- $x_3$ = no. of floors
- $x_4$ = age of home in years



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

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

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

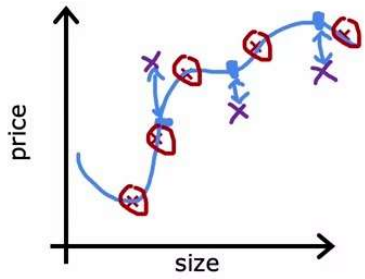
Compute test error:

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

Compute training error:

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

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




X = train

X = test

$J_{train}(\vec{w}, b)$ will be low

$J_{test}(\vec{w}, b)$ will be high



# Train/test procedure for classification problem

0/1

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

E.g.,

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

Compute test error:

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

Compute train error:

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



# Train/test procedure for classification problem

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

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

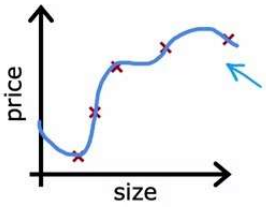
count $\hat{y} \neq y$

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

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

## Model selection and training/cross validation/test sets

# Model selection (choosing a model)




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.

$x = x_1$

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



# Model selection (choosing a model)

d=1 1. $f_{\vec{w},b}(\vec{x}) = w_1 x + b$ → $w^{<1>}, b^{<1>} \rightarrow J_{test}(w^{<1>}, b^{<1>})$

d=2 2. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + b$ → $w^{<2>}, b^{<2>} \rightarrow J_{test(w^{<2>},b^{<2>})}$

d=3 3. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b$ → $w^{<3>}, b^{<3>} \rightarrow J_{test(w^{<3>},b^{<3>})}$

⋮ ⋮

d=10 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b$ → $J_{test(w^{<10>},b^{<10>})}$

Choose $w_1 x + \cdots + w_5 x^5 + b$ d=5 $J_{test}(w^{<5>}, b^{<5>})$

How well does the model perform? Report test set error $J_{test}(w^{<5>}, b^{<5>})$?

The problem: $J_{test}(w^{<5>}, b^{<5>})$ is likely to be an optimistic estimate of generalization error (ie. $J_{test}(w^{<5>}, b^{<5>})$ < generalization error ). Because an extra parameter d (degree of polynomial) was chosen using the test set.

$w, b$ are overly optimistic estimate of generalization error on training data.



# Training/cross validation/test set

validation set
development set
dev set

| size | price |
|---|---|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| 1985 | 300 |
| 1534 | 315 |
| 1427 | 199 |
| 1380 | 212 |
| 1494 | 243 |

training set 60% → $(x^{(1)}, y^{(1)}) \ldots (x^{(m_{train})}, y^{(m_{train})})$ $m_{train} = 6$

cross validation 20% → $(x_{cv}^{(1)}, y_{cv}^{(1)}) \ldots (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$ $m_{cv} = 2$

test set 20% → $(x_{test}^{(1)}, y_{test}^{(1)}) \ldots (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$ $m_{test} = 2$

# Training/cross validation/test set

Training error: $J_{train}(\vec{w}, b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2\right]$

Cross validation error: $J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2\right]$ (validation error, dev error)

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



# Model selection

$d=1$ 1. $f_{\vec{w},b}(\vec{x}) = w_1 x + b$ $w^{<1>}, b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$

$d=2$ 2. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + b$ → $J_{cv}(w^{<2>}, b^{<2>})$

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

⋮ ⋮ ⋮

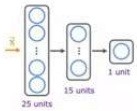
$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b$ $J_{cv}(w^{<10>}, b^{<10>})$

→ Pick $w_1 x + \cdots + w_4 x^4 + b$ $(J_{cv}(w^{<4>}, b^{<4>}))$

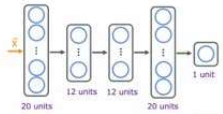
Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$



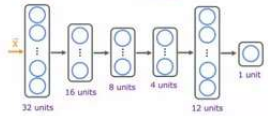
# Model selection – choosing a neural network architecture

1. 

$w^{<1>}, b^{<1>}$ $J_{cv}(w^{<1>}, b^{<1>})$

→ 2. 

$w^{<2>}, b^{<2>}$ $J_{cv}(w^{<2>}, b^{<2>})$

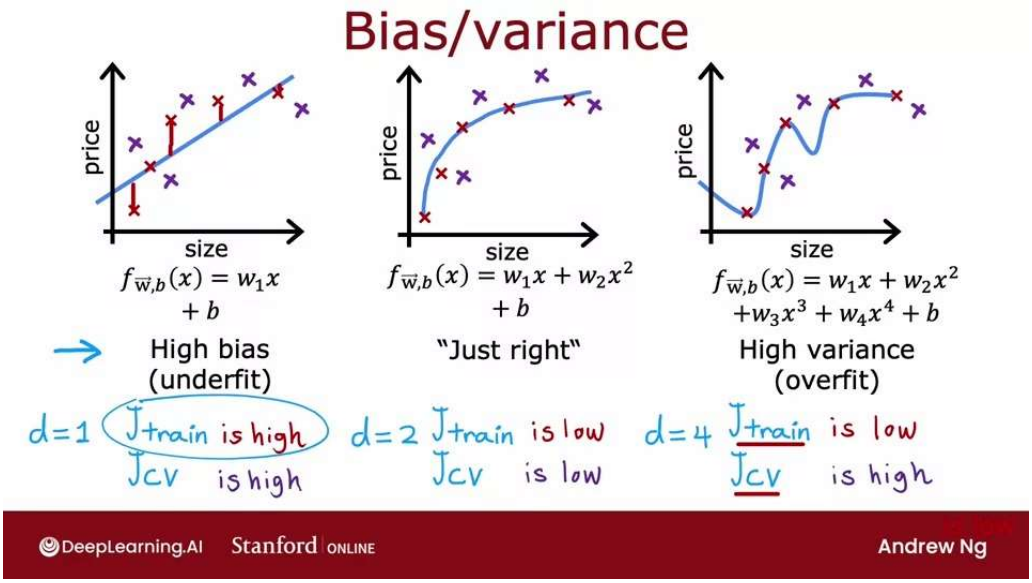
3. 

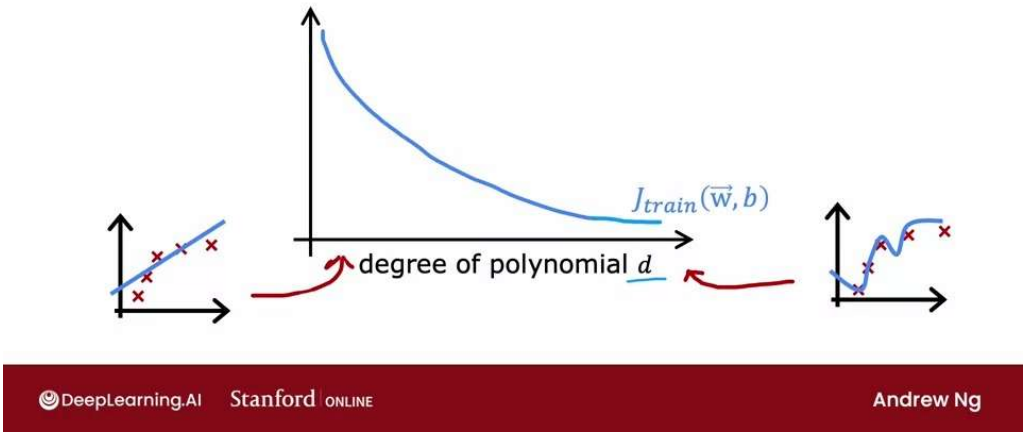
$w^{<3>}, b^{<3>}$ $J_{cv}(w^{<3>}, b^{<3>})$

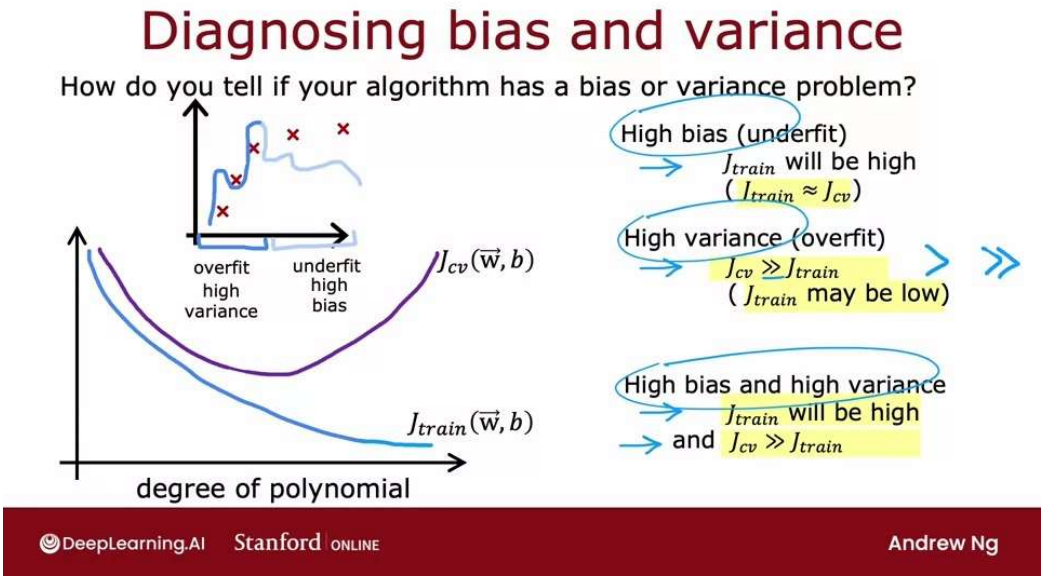
Pick $w^{<2>}, b^{<2>}$

Estimate generalization error using the test set: $J_{test}(w^{<2>}, b^{(<2>)})$

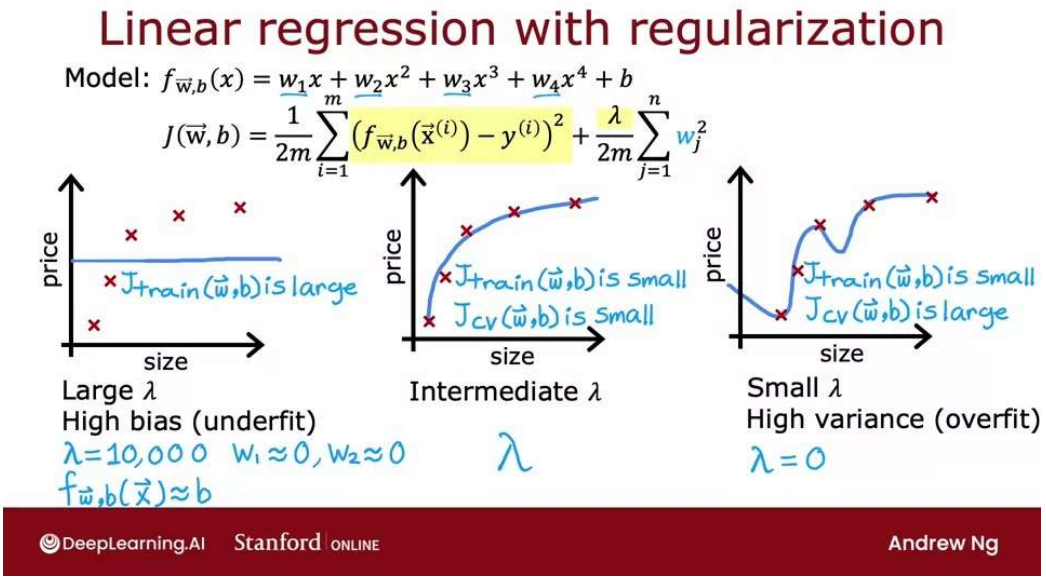
## Bias and variance

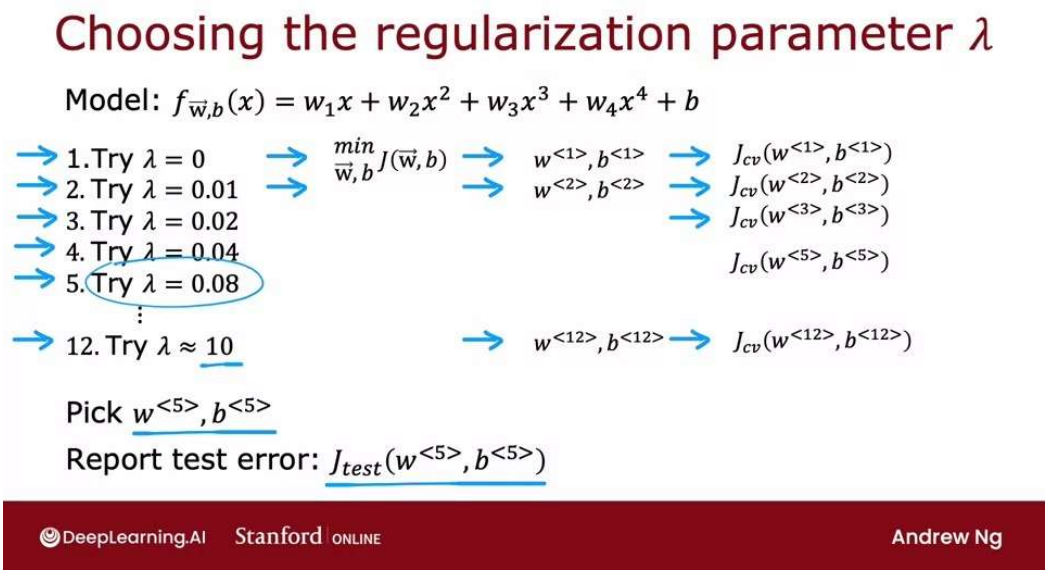


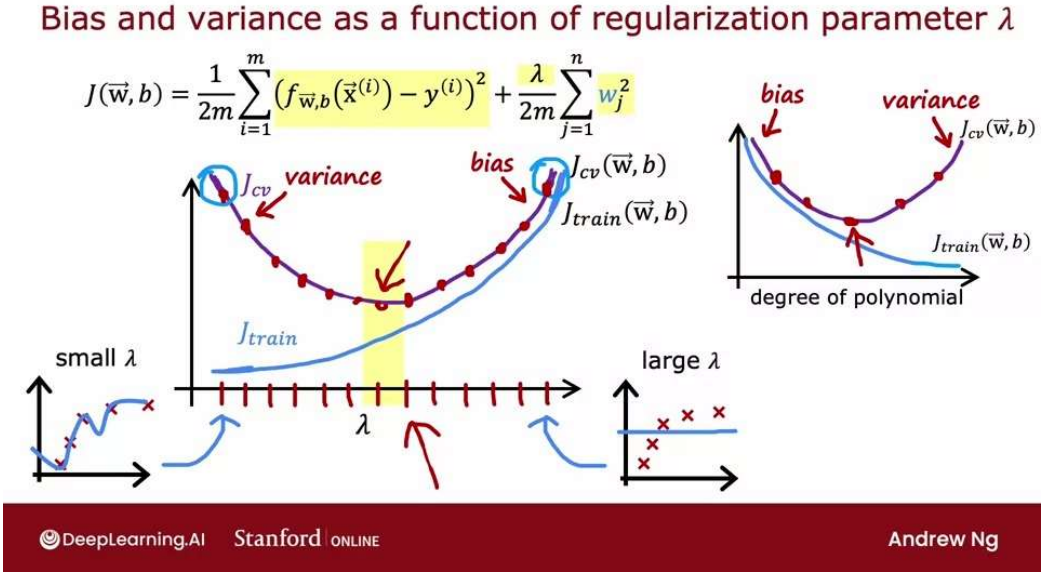



**Regularization and bias/variance**



**Baseline level of performance**

## Speech recognition example

Human level performance : 10.6%
Training error $J_{train}$ : 10.8%
Cross validation error $J_{cv}$ : 14.8%

0.2%
4.0%



## Establishing a baseline level of performance

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

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



## Bias/variance examples

Baseline performance : 10.6%
Training error ($J_{train}$) : 10.8%
Cross validation error ($J_{cv}$): 14.8%

0.2%
4.0%

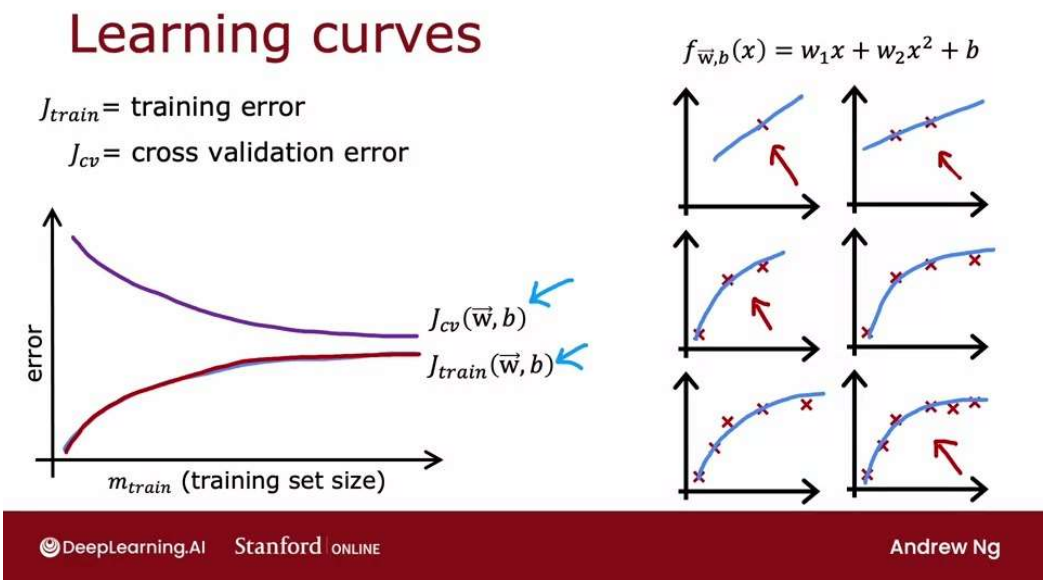
10.6%
15.0%
4.4%

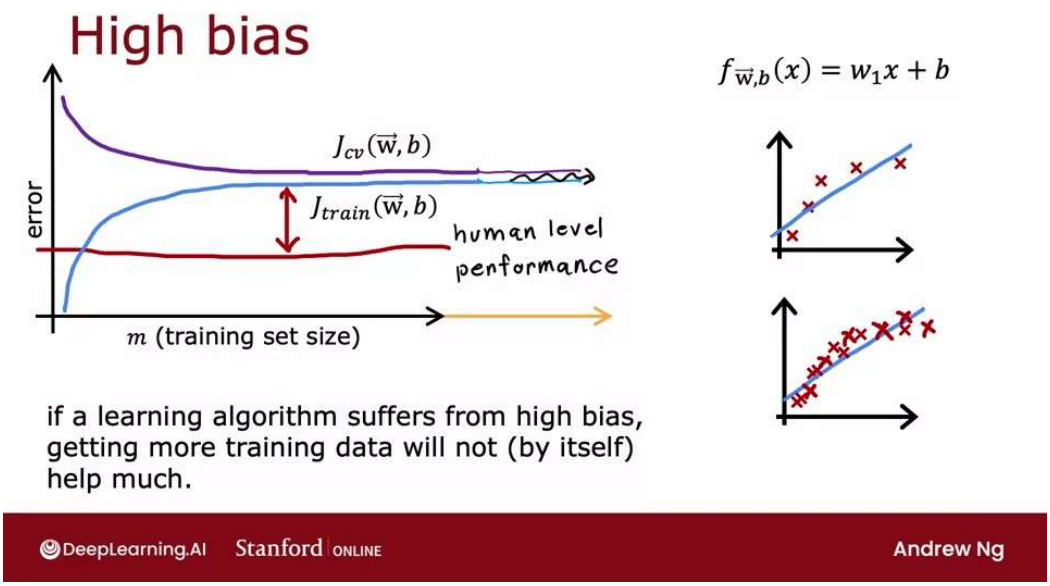
15.5%
0.5%

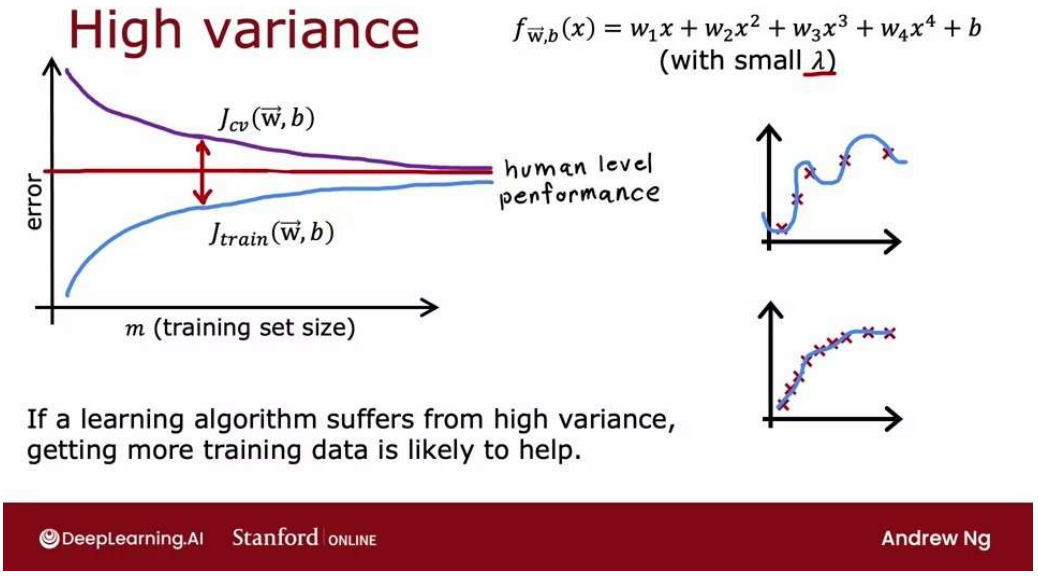
high variance

high bias

## Learning curves



# Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

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

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 $x, x^2, \ldots$ — fixes high variance
- → Try getting additional features — fixes high bias
- → Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$ — fixes high bias
- → Try decreasing $\lambda$ — fixes high bias
- → Try increasing $\lambda$ — fixes high variance



Neural networks

# The bias variance tradeoff

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

Simple model

High bias

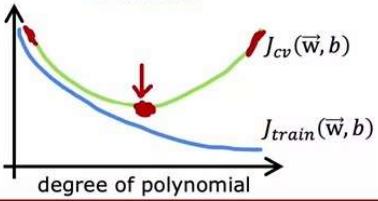
$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$

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

Complex model

High variance

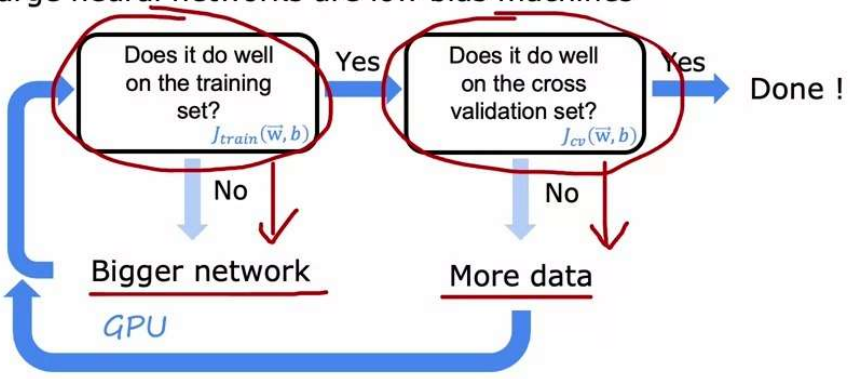
tradeoff

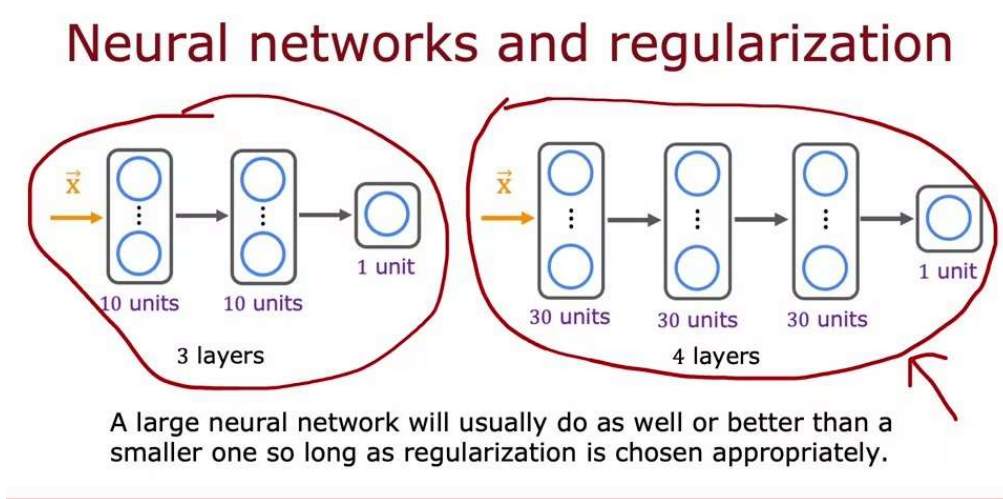


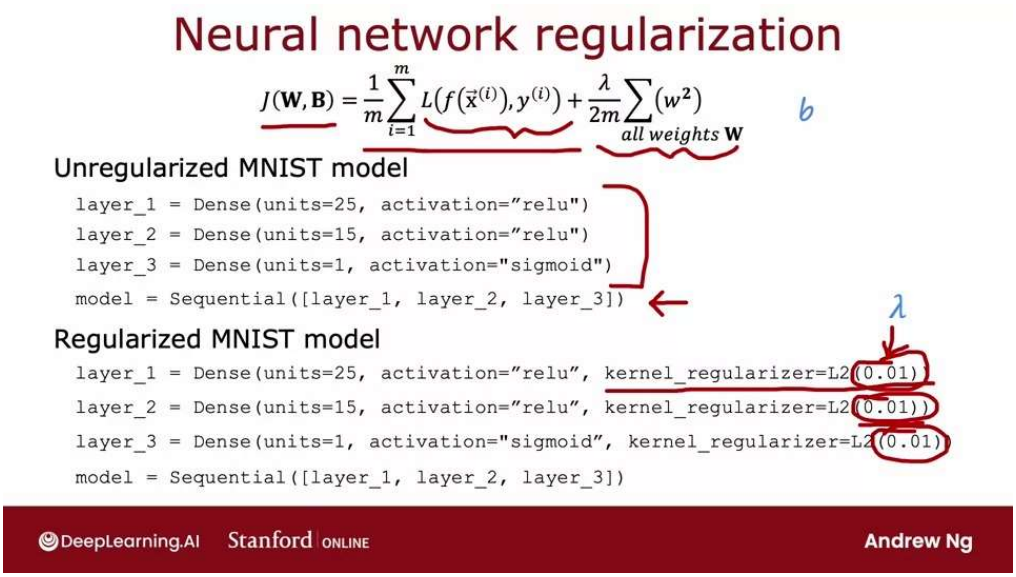



# Neural networks and bias variance

Large neural networks are low bias machines

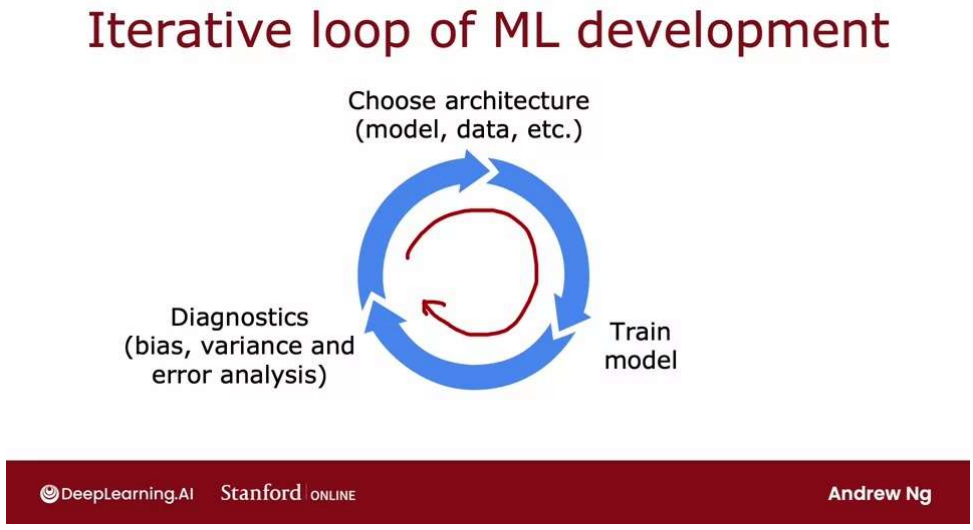





## Machine learning development process

### 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: $\vec{x}$ = features of email
$y$ = spam (1) or not spam (0)

Features: list the top 10,000 words to compute $x_1, x_2, \cdots, x_{10,000}$

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{matrix}$$

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.

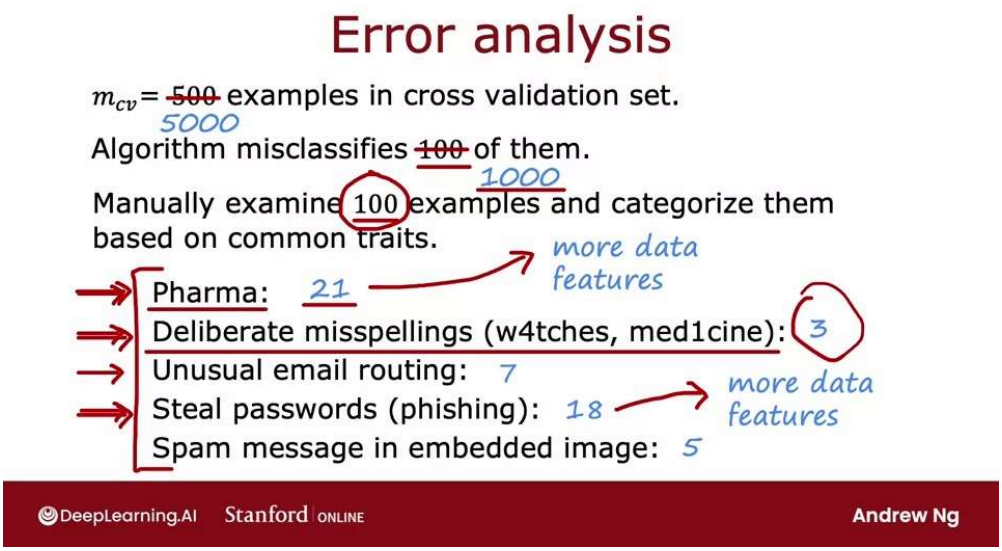


# Building a spam classifier

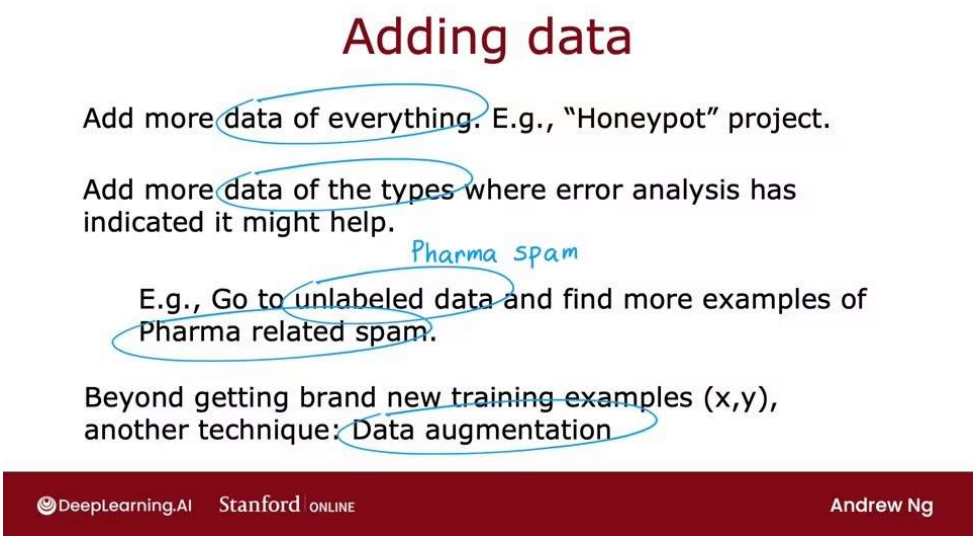
How to try to reduce your spam classifier's error?

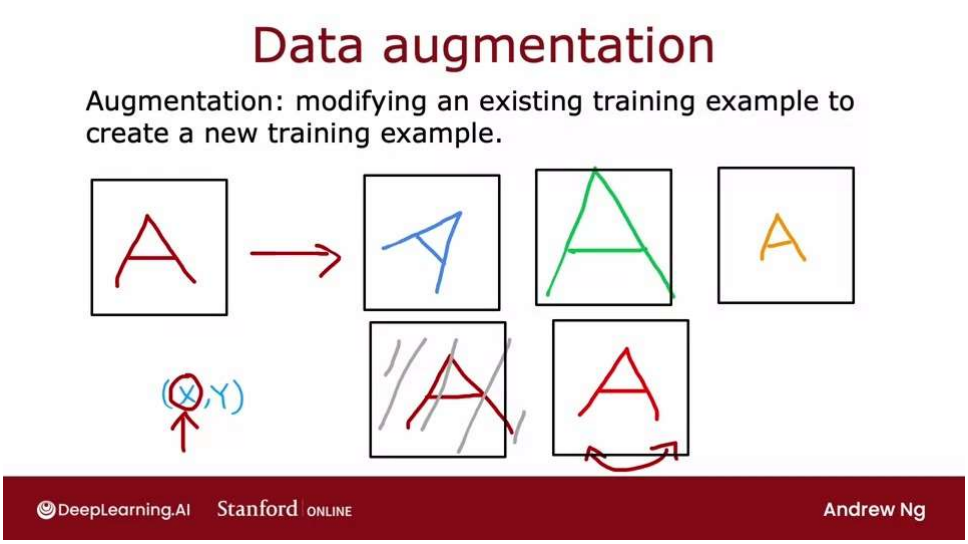
- Collect more data. E.g., "Honeypot" project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body. E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings. E.g., w4tches, med1cine, m0rtgage.

**Error analysis**

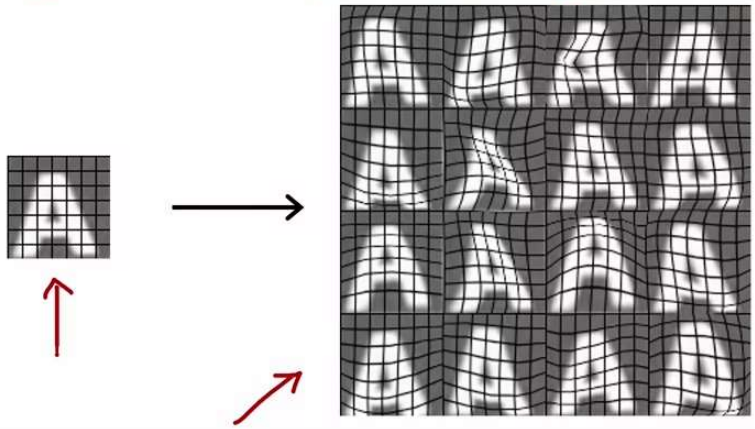



**Adding data**



## Data augmentation by introducing distortions





## Data augmentation for speech

Speech recognition example

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

\+ Noisy background: Crowd

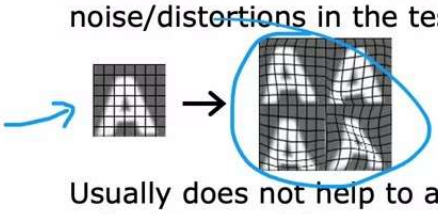
\+ Noisy background: Car

\+ Audio on bad cellphone connection



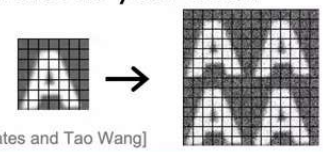
## Data augmentation by introducing distortions

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



Audio:
Background noise,
bad cellphone connection

Usually does not help to add purely random/meaningless noise to your data.



[Adam Coates and Tao Wang]

$x_i$=intensity (brightness) of pixel $i$
$x_i \leftarrow x_i$ + random noise

# Artificial data synthesis for photo OCR

Real data

Synthetic data

[Adam Coates and Tao Wang]



# Engineering the data used by your system

Conventional model-centric approach:

AI = Code + Data
(algorithm/model)

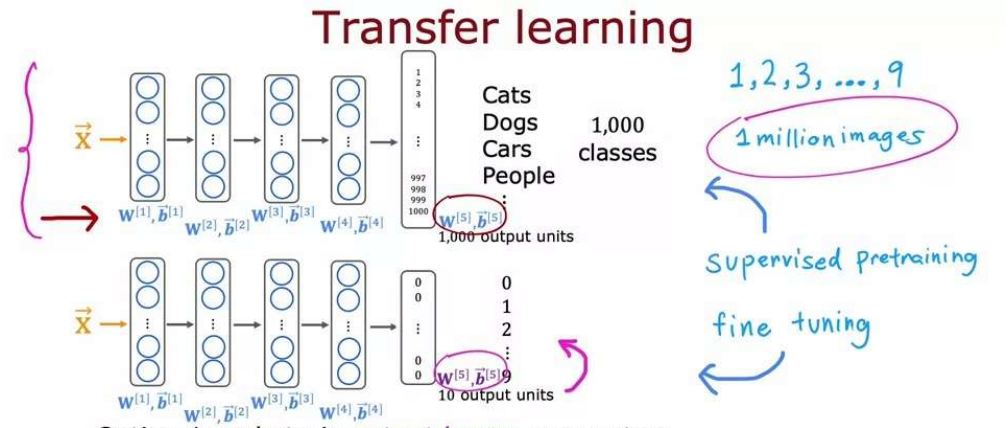
work on this

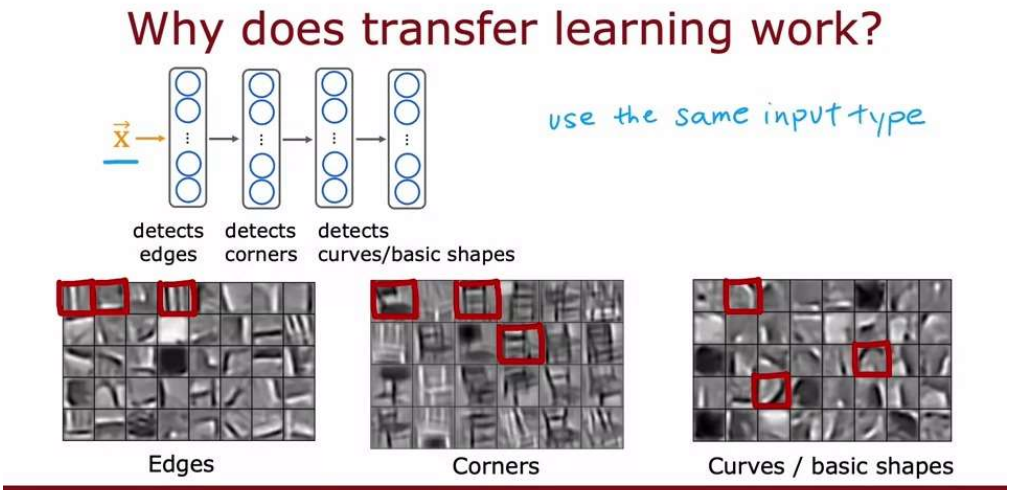
Data-centric approach:

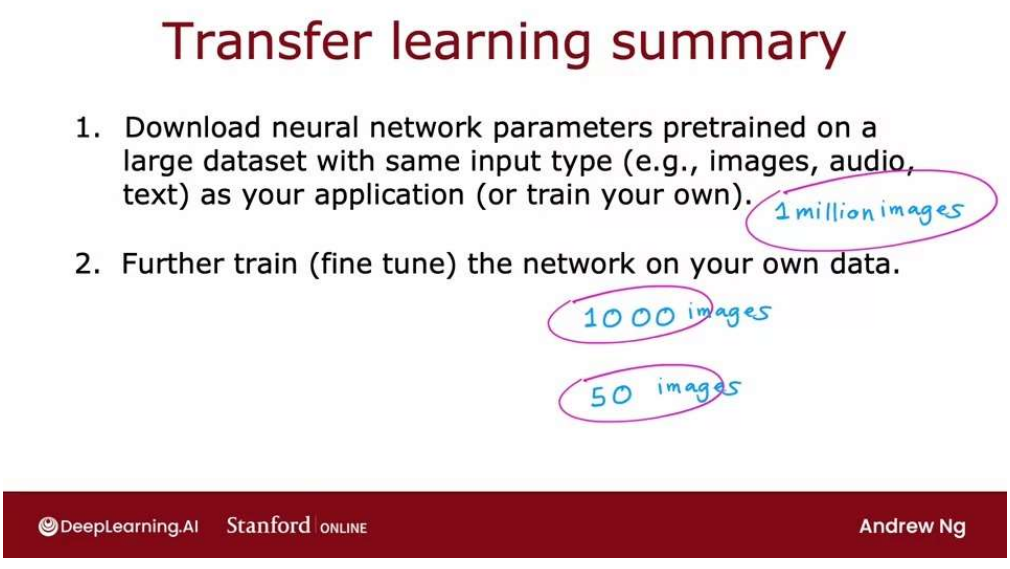
AI = Code + Data
(algorithm/model)

work on this

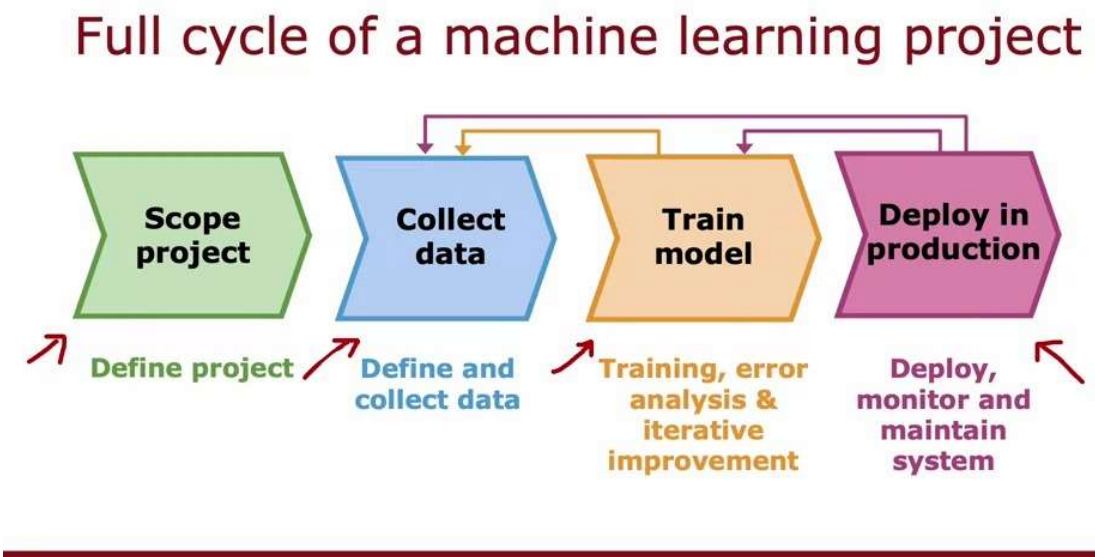
**Transfer learning**

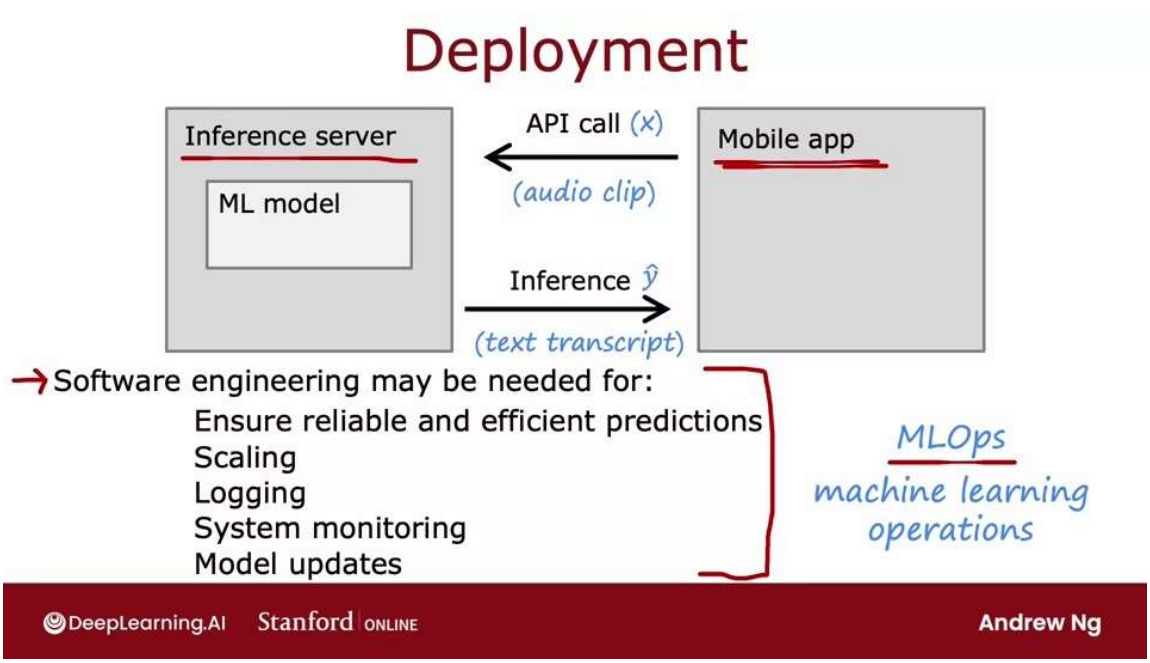


**Full cycle of a machine learning project**



**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.

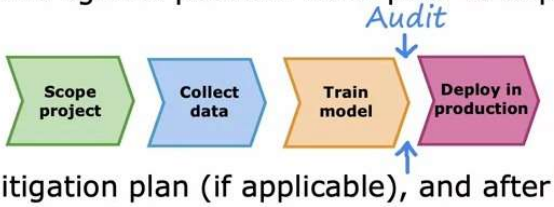


## Guidelines

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

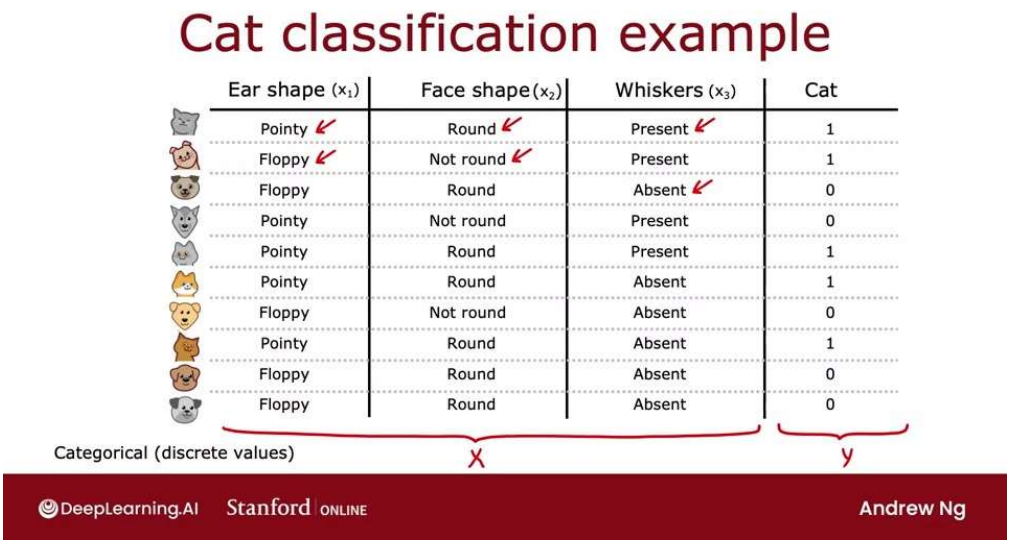
Carry out literature search on standards/guidelines for your industry.

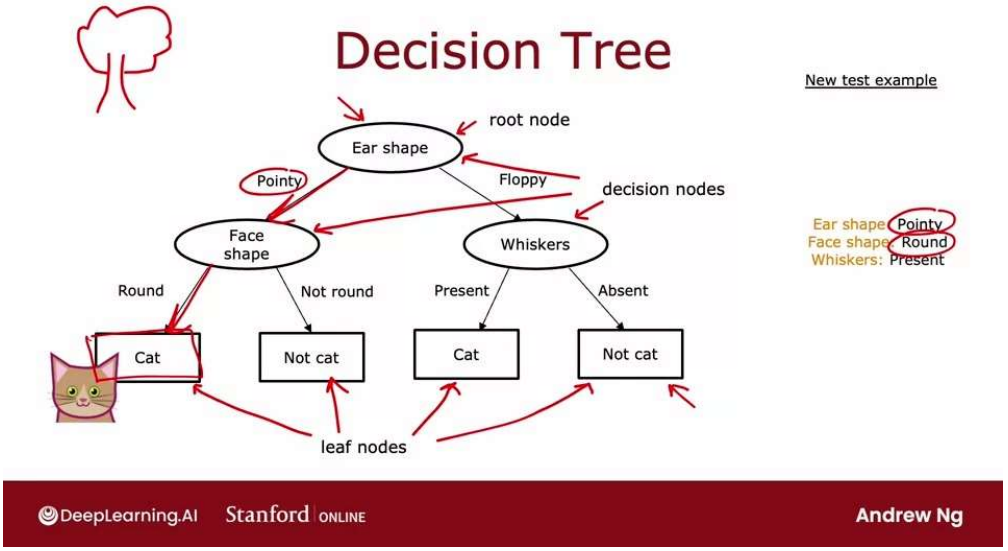
Audit systems against possible harm prior to deployment.

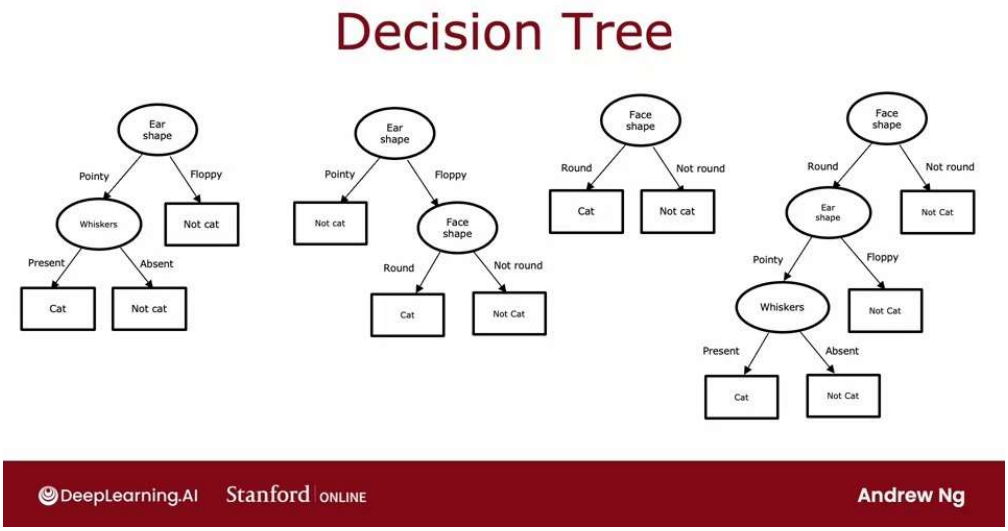



Develop mitigation plan (if applicable), and after deployment, monitor for possible harm.

## Week 4
## Decision trees

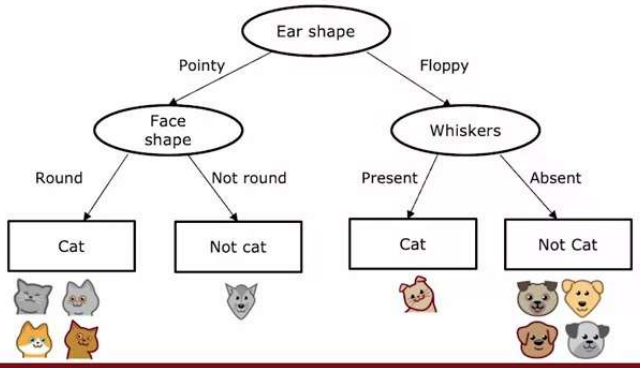


## Learning Process

# Decision Tree Learning

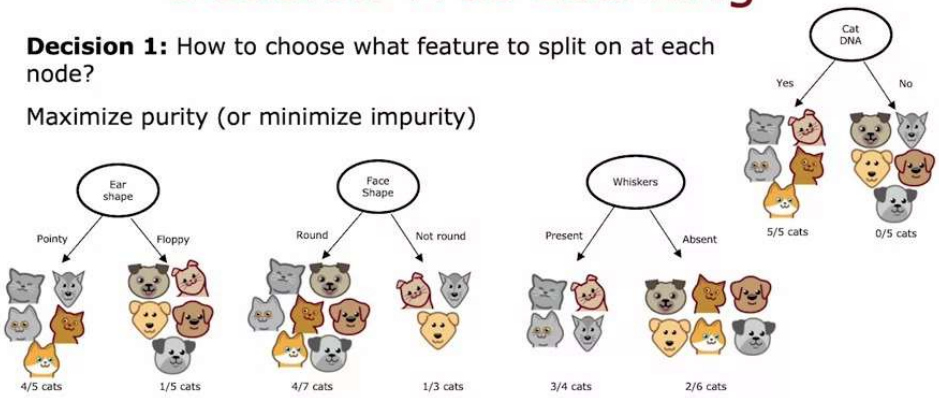



DeepLearning.AI Stanford ONLINE Andrew Ng

# Decision Tree Learning

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

Maximize purity (or minimize impurity)

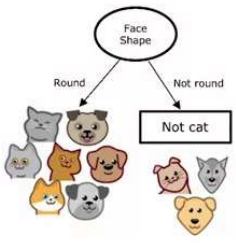



DeepLearning.AI Stanford ONLINE Andrew Ng

# 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

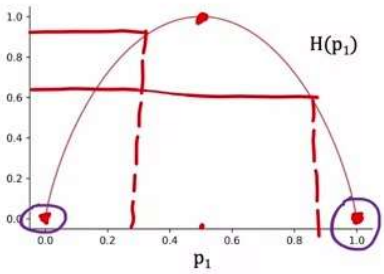



DeepLearning.AI Stanford ONLINE Andrew Ng

## Decision tree learning
### Measuring purity

# Entropy as a measure of impurity

$p_1$ = fraction of examples that are cats





$p_1 = 0 \quad H(p_1) = 0$

$p_1 = 2/6 \quad H(p_1) = 0.92$

$p_1 = 3/6 \quad H(p_1) = 1$

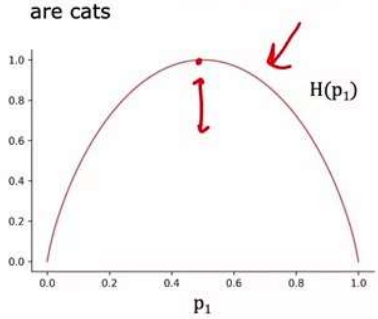
$p_1 = 5/6 \quad H(p_1) = 0.65$

$p_1 = 6/6 \quad H(p_1) = 0$



# Entropy as a measure of impurity

$p_1$ = fraction of examples that are cats

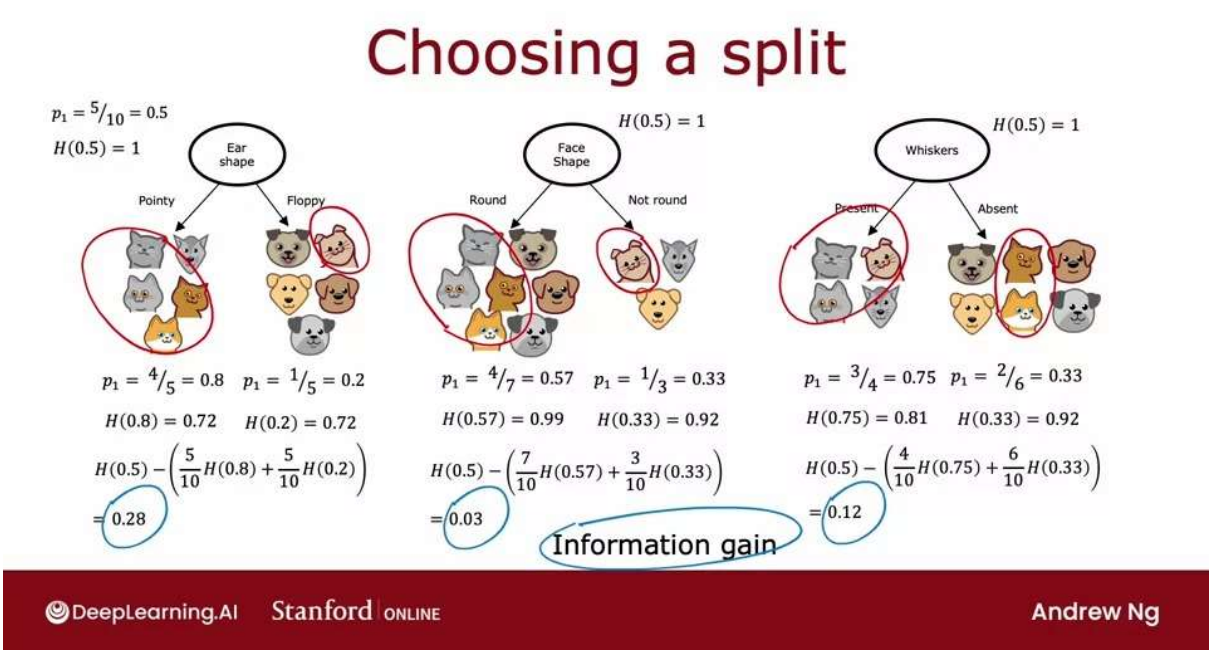


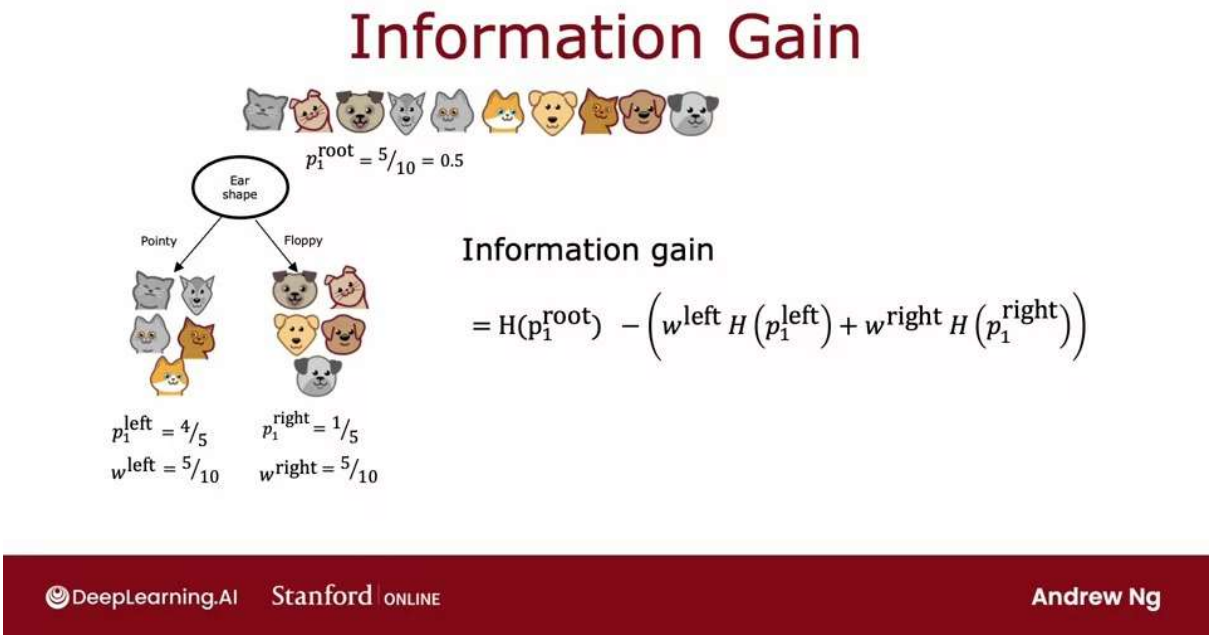

$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

Note: "0 log(0)" = 0

## Information Gain



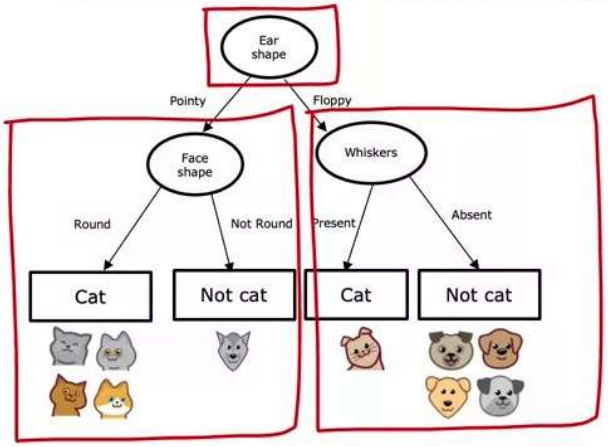
**Putting it together**

# 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

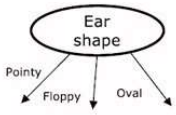



Recursive algorithm

**one-hot encoding**

## 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

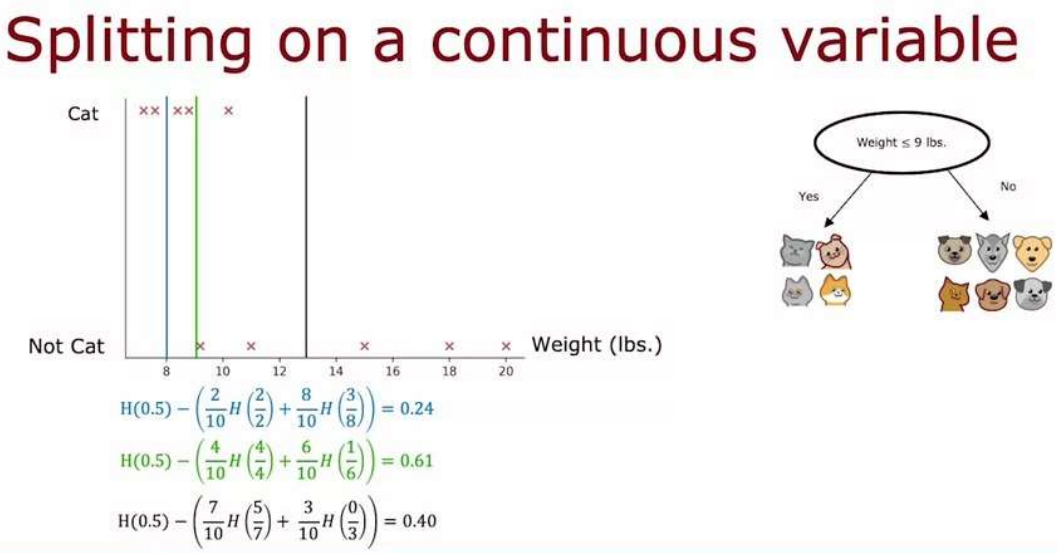
| ~~Ear shape~~ | Pointy ears | Floppy ears | Oval ears | Face shape | Whiskers | Cat |
|---|---|---|---|---|---|---|
| ~~Pointy~~ | 1 | 0 | 0 | Round | Present | 1 |
| ~~Oval~~ | 0 | 0 | 1 | Not round | Present | 1 |
| ~~Oval~~ | 0 | 0 | 1 | Round | Absent | 0 |
| ~~Pointy~~ | 1 | 0 | 0 | Not round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



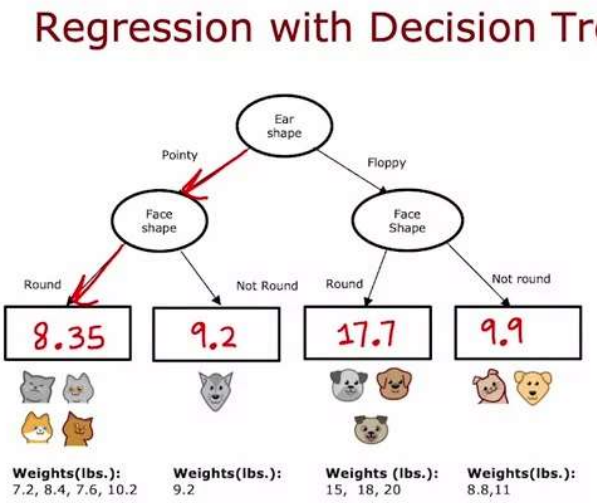
## One hot encoding and neural networks

| 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 |

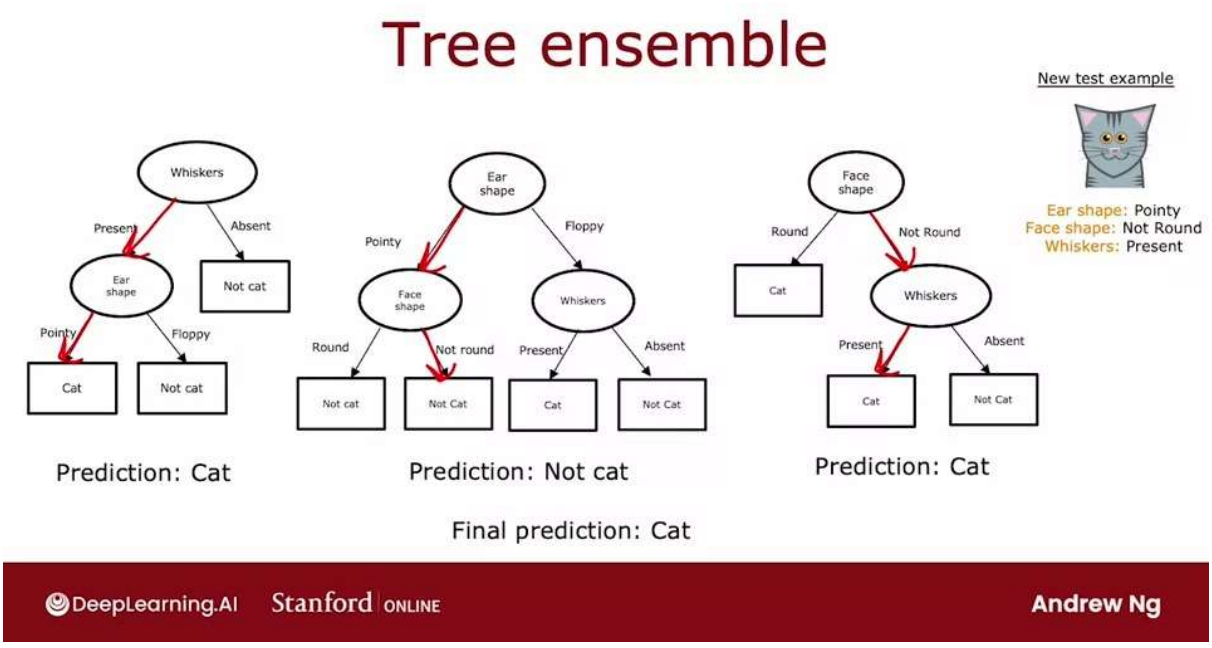
## Continuous valued features



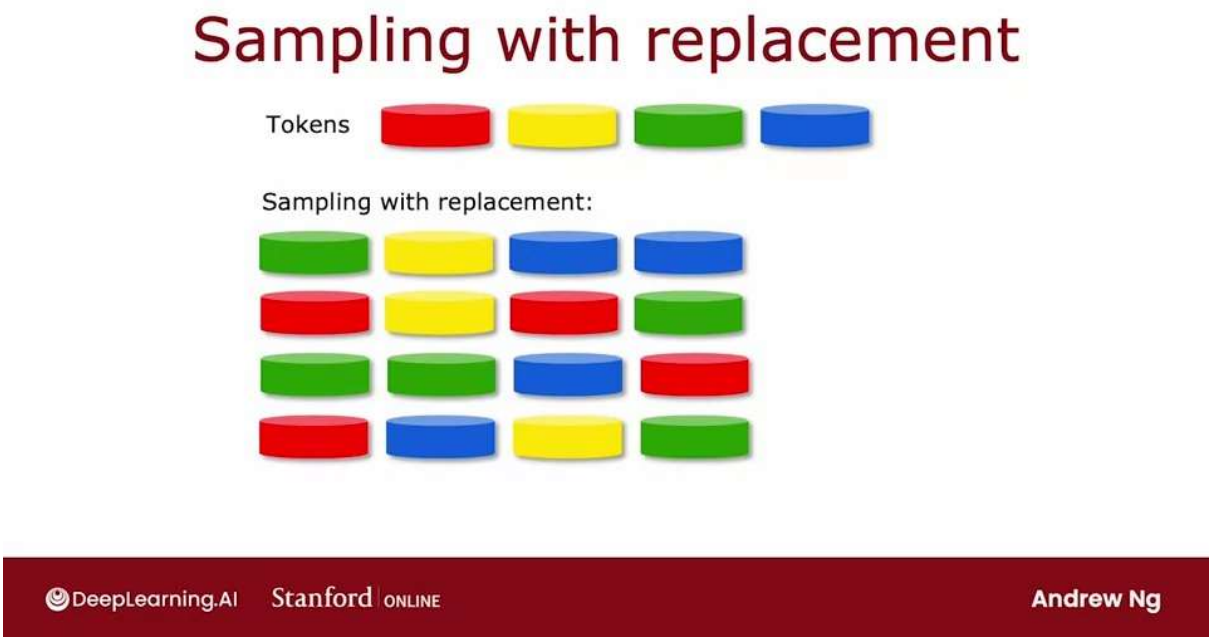

## Regression Trees

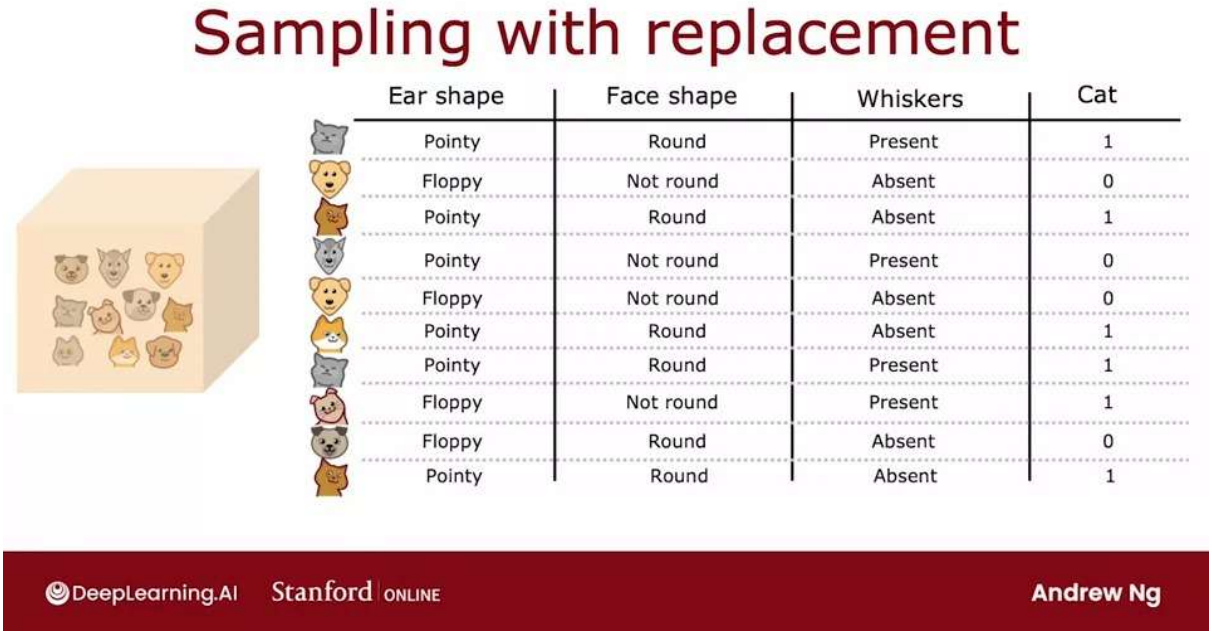


**Tree ensembles**
**multiple decision trees**

## Sampling with replacement



## Random forest algorithm

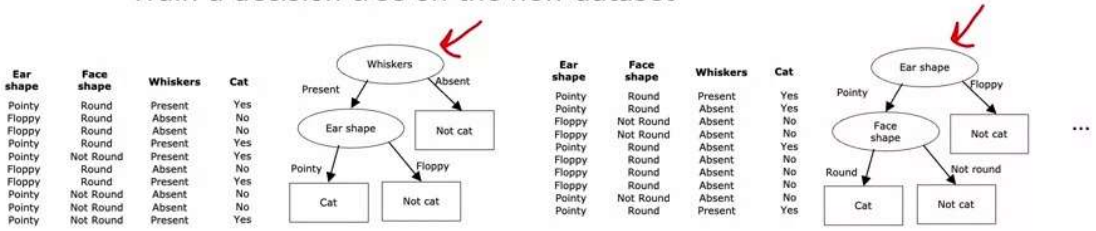
# Generating a tree sample

Given training set of size $m$

For $b = 1$ to $B$:

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

Train a decision tree on the new dataset



| Ear shape | Face shape | Whiskers | Cat |
|---|---|---|---|
| Pointy | Round | Present | Yes |
| Floppy | Round | Absent | No |
| Floppy | Round | Absent | No |
| Pointy | Round | Present | Yes |
| Pointy | Not Round | Present | Yes |
| Floppy | Round | Absent | No |
| Floppy | Round | Present | Yes |
| Pointy | Not Round | Absent | No |
| Pointy | Not Round | Absent | No |
| Pointy | Not Round | Present | Yes |



| Ear shape | Face shape | Whiskers | Cat |
|---|---|---|---|
| Pointy | Round | Present | Yes |
| Pointy | Round | Absent | Yes |
| Floppy | Not Round | Absent | No |
| Floppy | Not Round | Absent | No |
| Pointy | Round | Absent | Yes |
| Floppy | Round | Absent | No |
| Floppy | Round | Absent | No |
| Floppy | Round | Absent | No |
| Pointy | Not Round | 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

**XGBoost**

# Boosted trees intuition

Given training set of size $m$

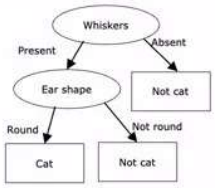
For $b = 1$ to $B$:

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

But instead of picking from all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees

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 |






# Boosted trees intuition

Given training set of size $m$

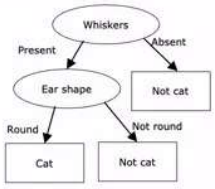
For $b = 1$ to $B$:

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

But instead of picking from all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees

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 ✓ |

# 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)
```



**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
- 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