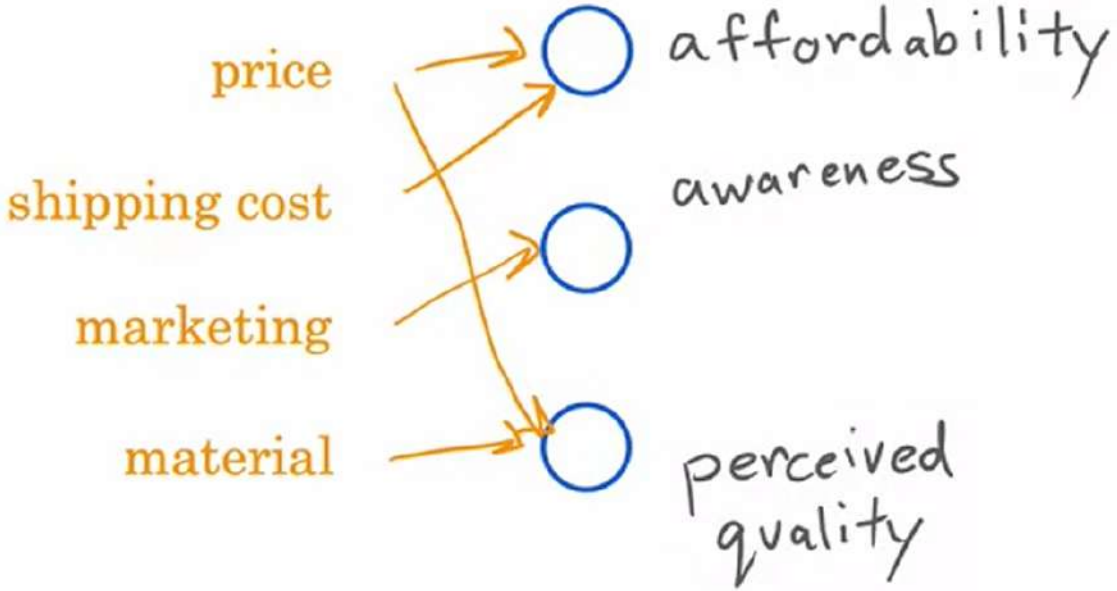
Demand Prediction



probability of being a top seller

Price is a factor here

Demand Prediction

feature engineering

x_1 x_2

$\vec{x}$ (x, y) "hidden layer" layer ← can have multiple neurons

input layer

"activations"

price

shipping cost

marketing

material

affordability ←

awareness ←

perceived ←

quality

$\vec{a}$

layer ← output layer

a

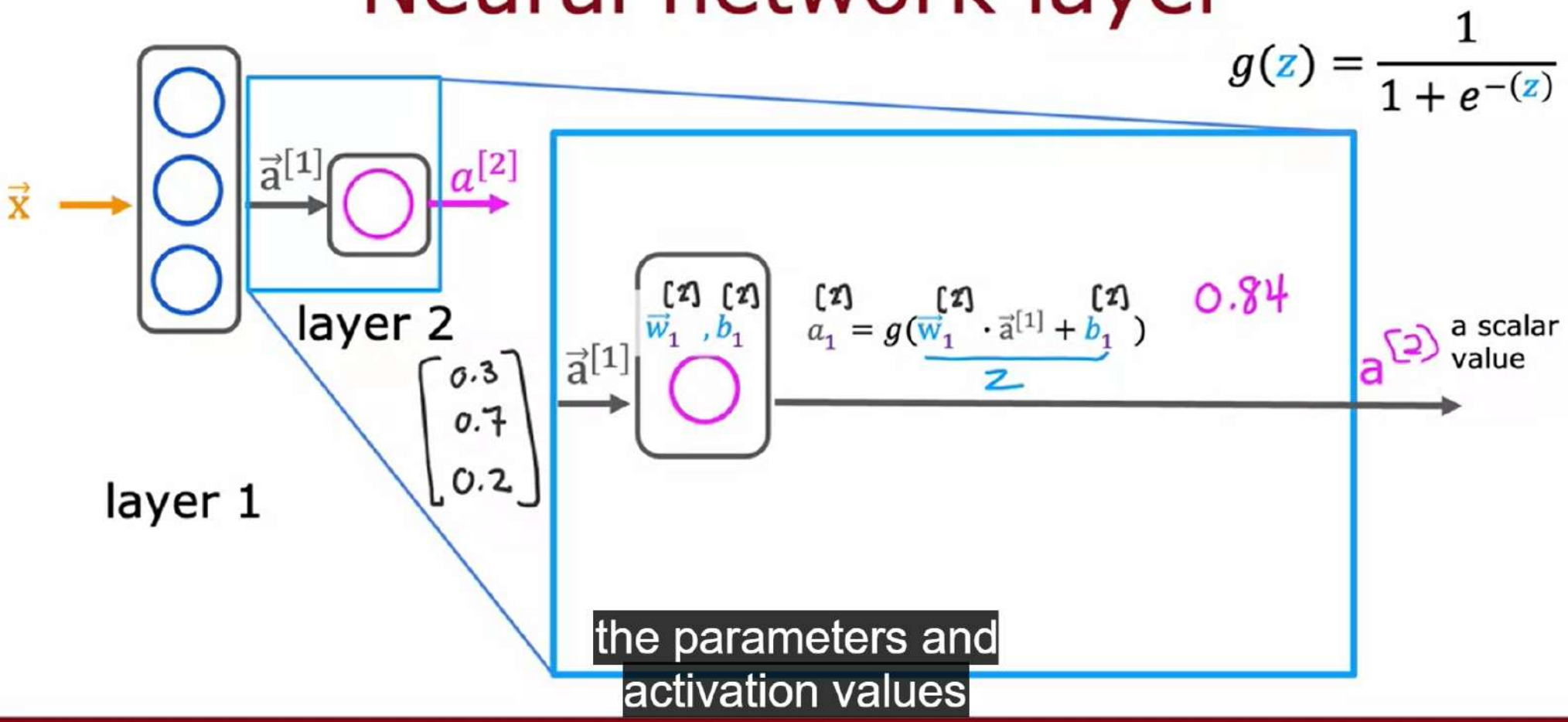
probability of being a top seller

4 numbers

3 activation values

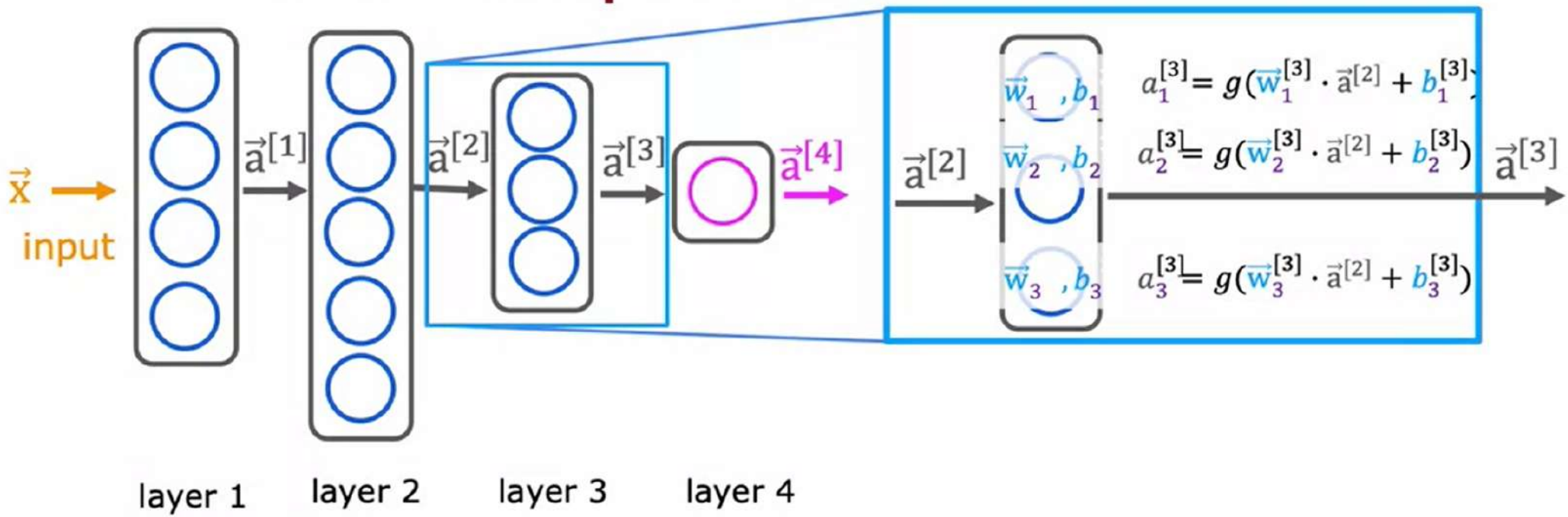
does this, the input layer has a vector of features,

Neural network layer



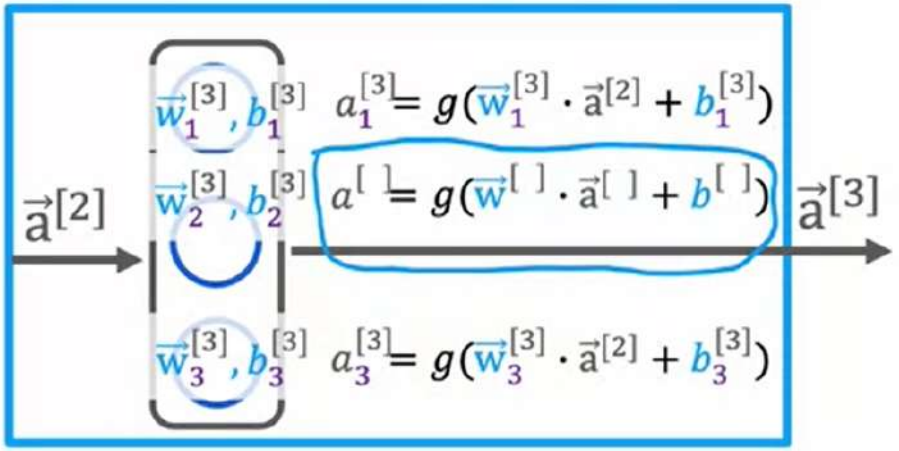
the parameters and activation values

More complex neural network



which is another vector.

[In Video Quiz]



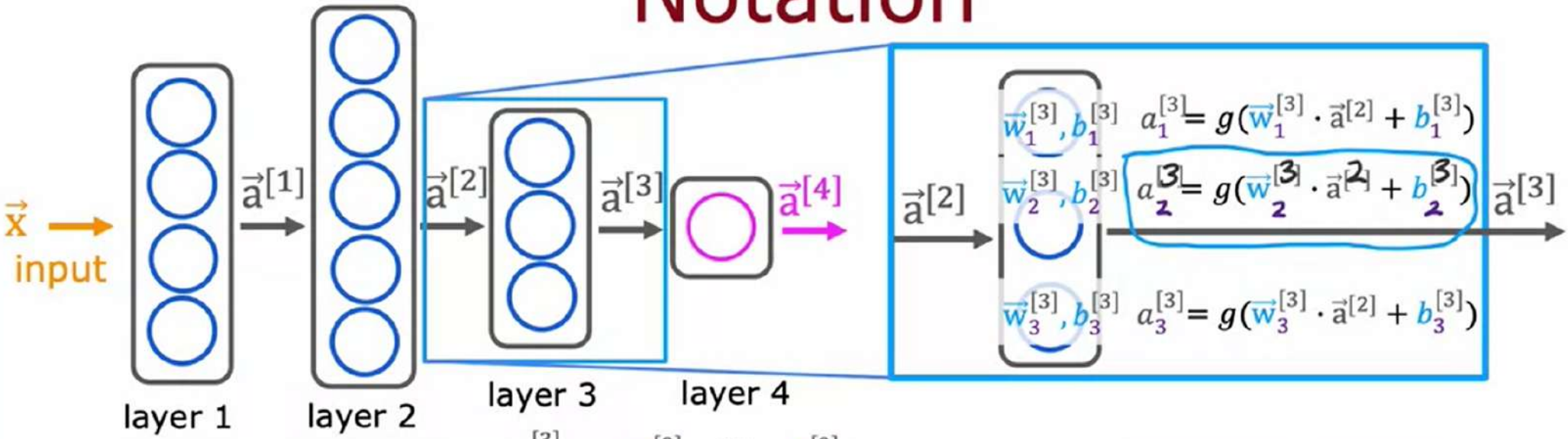
Can you fill in the superscripts and subscripts for the second neuron?

✓ $a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$

$a_2^{[3]} = g(\vec{w}_2^{[3]}$ The activation of the 2nd

$a_2^{[3]} = g(\vec{w}$ neuron at layer 3 is denoted

Notation



$$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$$

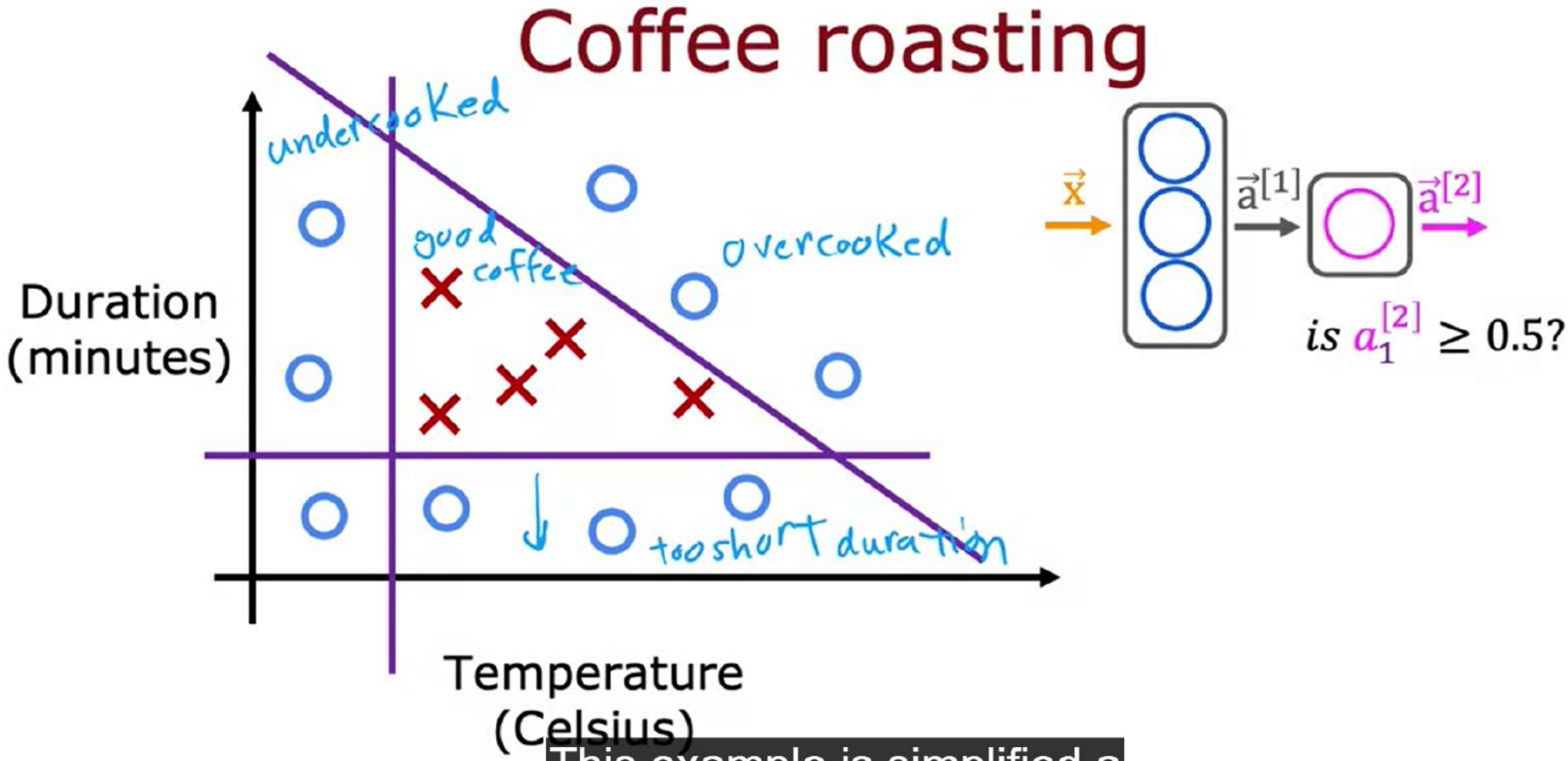
Activation value of layer l, unit(neuron) j

output of layer $l-1$ (previous layer)

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

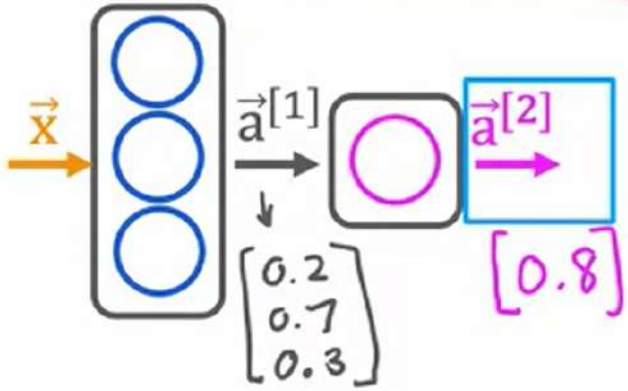
layer l and a subscript j denotes unit j.

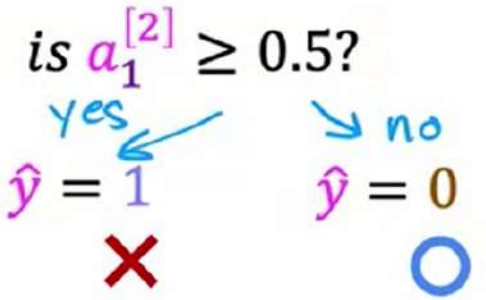
Parameters w & b of layer l, unit j



This example is simplified a

Build the model using TensorFlow





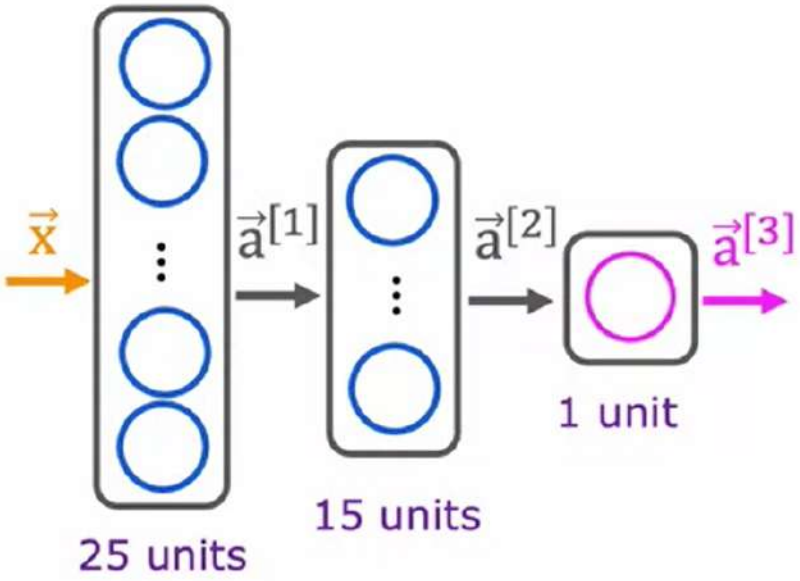

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

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

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

But these are the key steps for forward propagation in

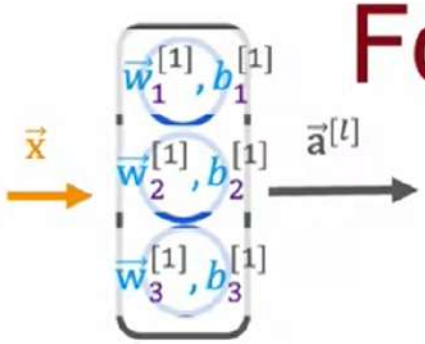
Digit classification model




```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)

x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])

model.fit(x,y)

model.predict(x_new)
```

you into a new network and same as before.

Forward prop in NumPy



$$\vec{w}_1^{[1]} = \begin{bmatrix}1\\2\end{bmatrix} \quad \vec{w}_2^{[1]} = \begin{bmatrix}-3\\4\end{bmatrix} \quad \vec{w}_3^{[1]} = \begin{bmatrix}5\\-6\end{bmatrix}$$

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

$$b_1^{[l]} = -1 \quad b_2^{[l]} = 1 \quad b_3^{[l]} = 2$$

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

$$\vec{a}^{[0]} = \vec{x}$$

```
a_in = np.array([-2, 4])
```

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

and so will pull out w_1,1.

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

[1,0,1]

vectorized

```
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):
  Z = np.matmul(A_in,W) + B
  A_out = g(Z)
  return A_out

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

2D array

same

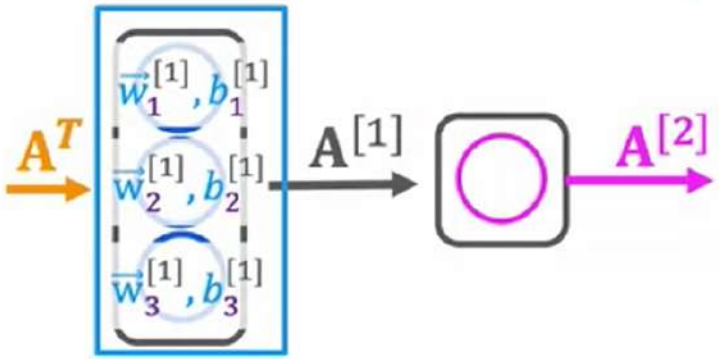
1x3 2D array

all 2D arrays

matrix multiplication

one step of forward propagation

Dense layer vectorized




```
AT = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([[-1, 1, 2]])

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

$$\mathbf{A}^T = [200 \quad 17] \qquad 1 \times 2$$

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

$$\mathbf{B} = [-1 \quad 1 \quad 2] \qquad 1 \times 3$$

$$\mathbf{Z} = \mathbf{A}^T\mathbf{W} + \mathbf{B}$$

$$[165 \quad -531 \quad 900]$$

$$z_1^{[1]} \quad z_2^{[1]} \quad z_3^{[1]}$$

$$\mathbf{A} = g(\mathbf{Z})$$

$$[1 \quad 0 \quad 1]$$

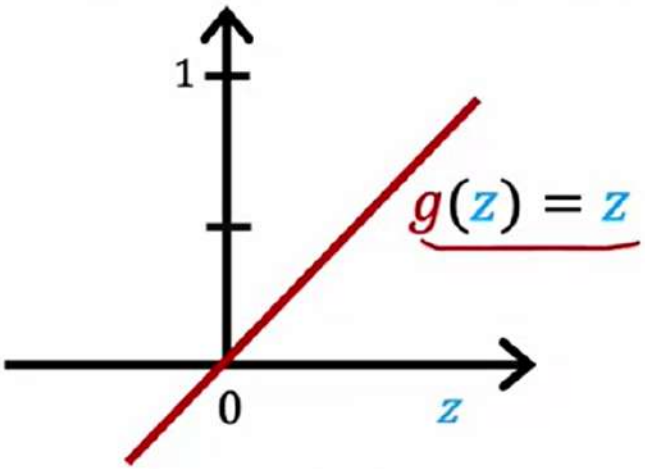
the activation function applied

Examples of Activation Functions

"No activation function"

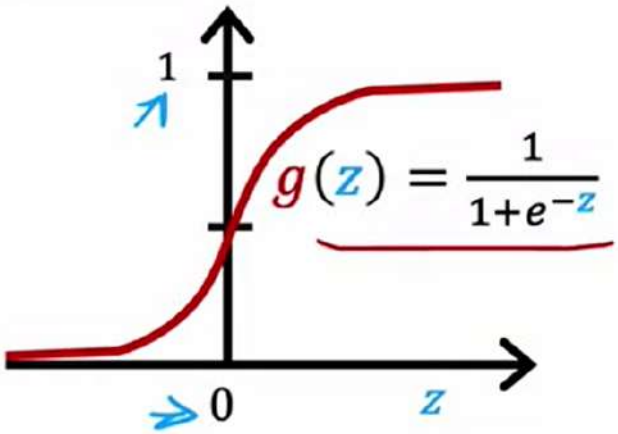
$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}^{z})$$

Linear activation function



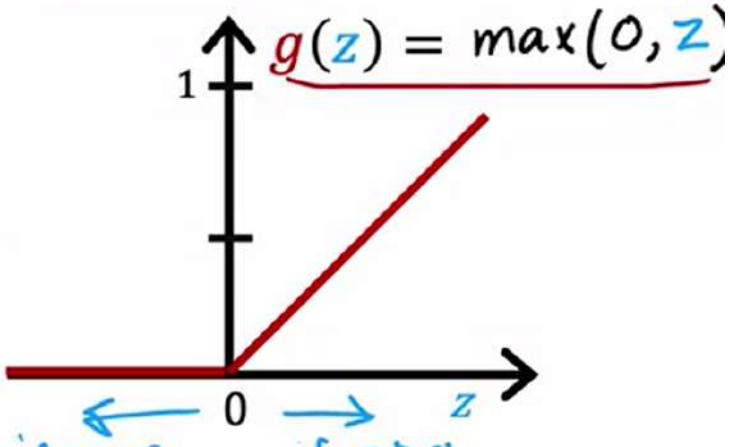
$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

Sigmoid



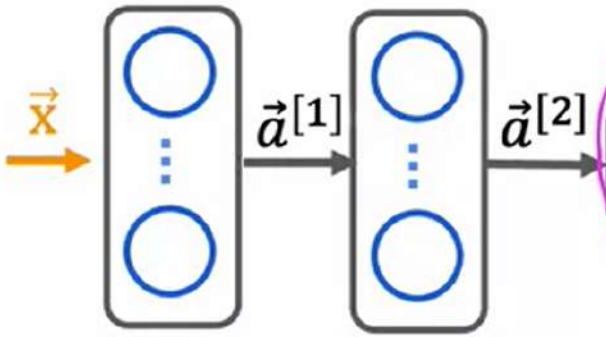
$0 < g(z) < 1$

ReLU Rectified Linear Unit



It just refers to the linear activation function.

Output Layer



$\vec{a}^{[3]} = f(\vec{x})$

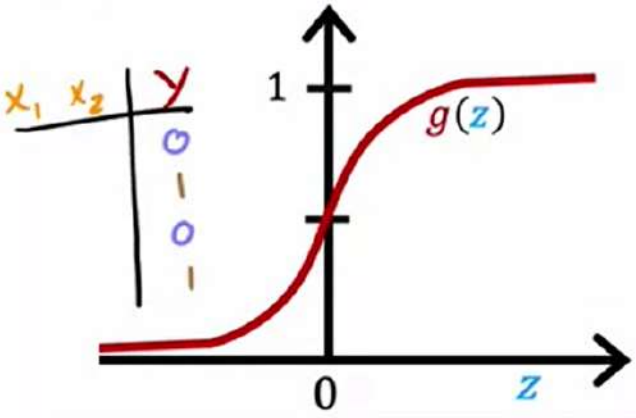
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

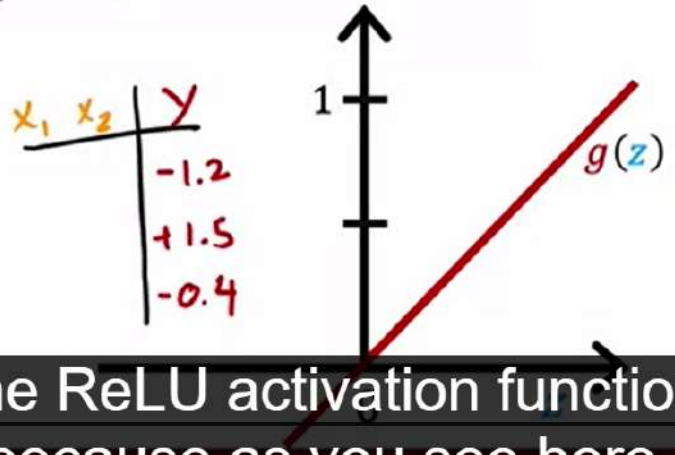
y=0/1



Regression

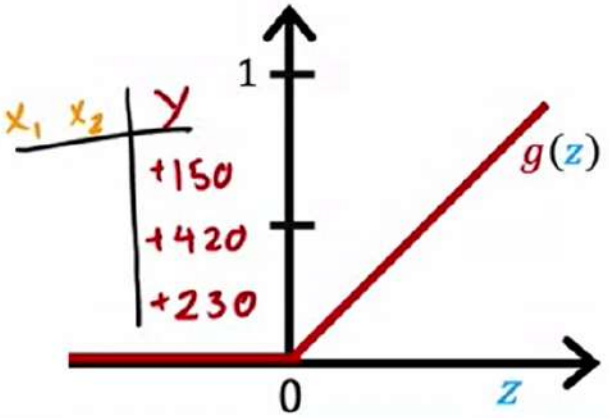
Linear activation function

y = +/-



Regression

ReLU



the ReLU activation function
because as you see here,

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

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

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

another variable k to index
the summation because

DeepLearning.AI Stanford ONLINE Andrew Ng

6:06 / 11:31

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 $y=0$

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

Softmax regression

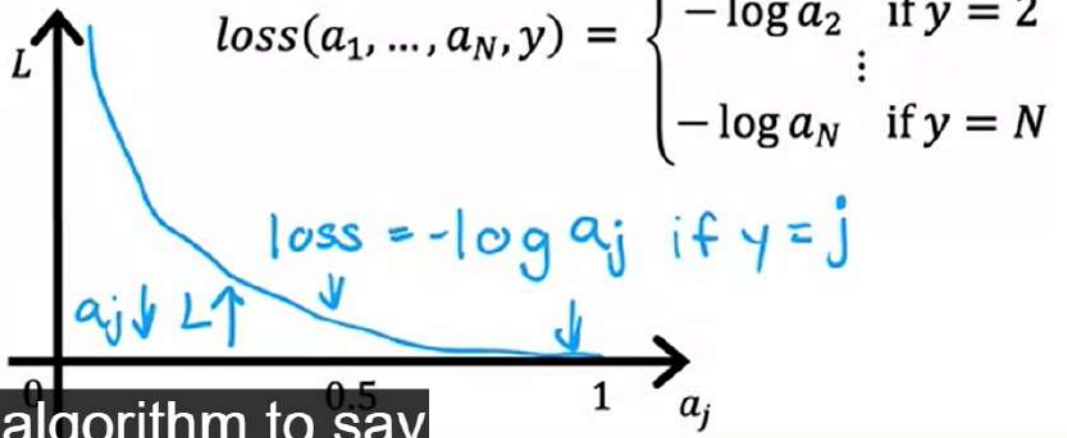
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

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



you want the algorithm to say

MNIST with softmax

1. specify the model

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

2. specify loss and cost

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

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

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

Just one important note, if you use this code exactly as I've written here, it will

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)

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

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

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

logit: z

either of these implementations
actually works okay,

More numerically accurate implementation of softmax

Softmax regression

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

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

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

'linear'

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

More Accurate

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

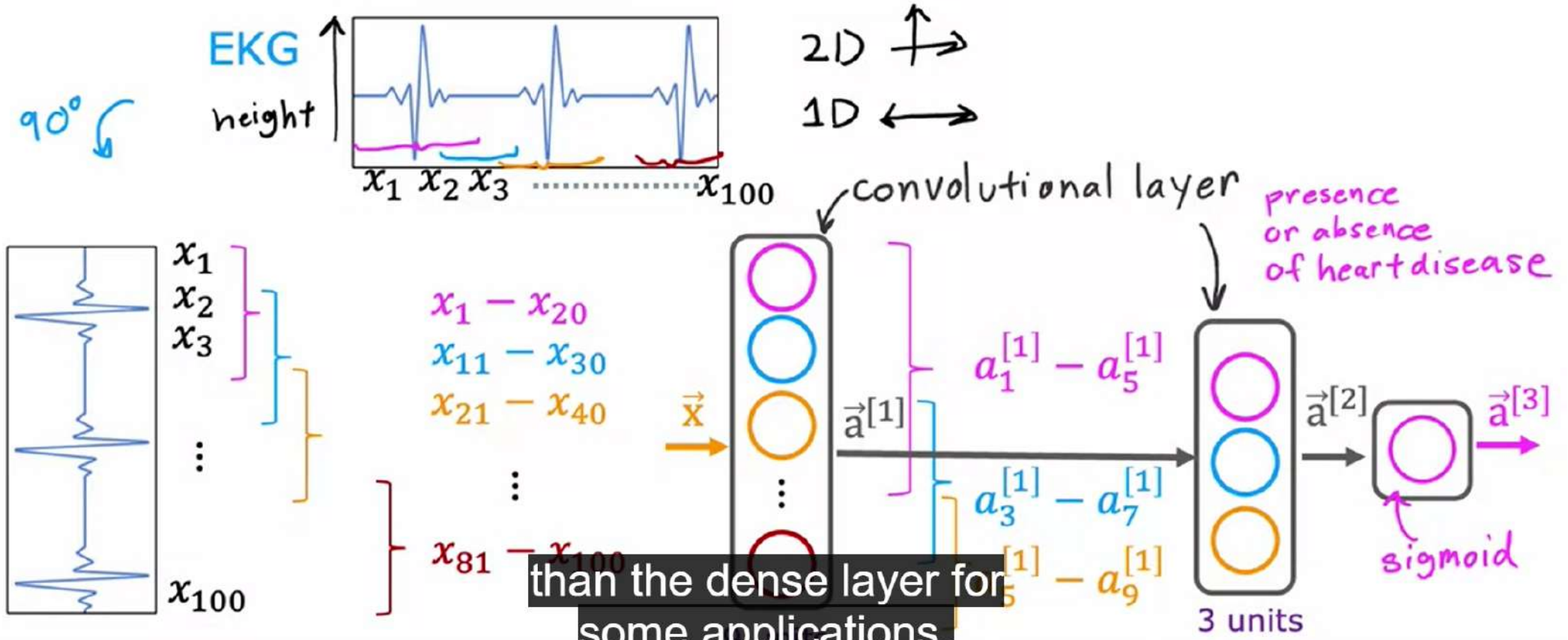
```
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

and this whole computation of

DeepLearning.AI Stanford ONLINE Andrew Ng

7:00 / 9:11

Convolutional Neural Network



than the dense layer for some applications.

Derivative Example

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

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

$\varepsilon = 0.002$

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

$w = 3 + 0.001$ 0.002

$J(w) = w^2 = 9.006001$

9.012004

9.012

If $w \uparrow$ 0.002 0.001 ε

$J(w) \uparrow 6 \times 0.001$ 0.002 $6 \times \varepsilon$ $6 \times 0.002 = 0.012$

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

That's why the
derivative of J of w

More Derivative Examples

$w = 3$ $\quad J(w) = w^2 = 9$ $\quad w \uparrow 0.001$ $\quad J(w) = J(3.001) = 9.006001$ $\quad \frac{\partial}{\partial w} J(w) = 6$

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

$w = 2$ $\quad J(w) = w^2 = 4$ $\quad w \uparrow 0.001$ $\quad J(w) = J(2.001) = 4.004001$ $\quad \frac{\partial}{\partial w} J(w) = 4$

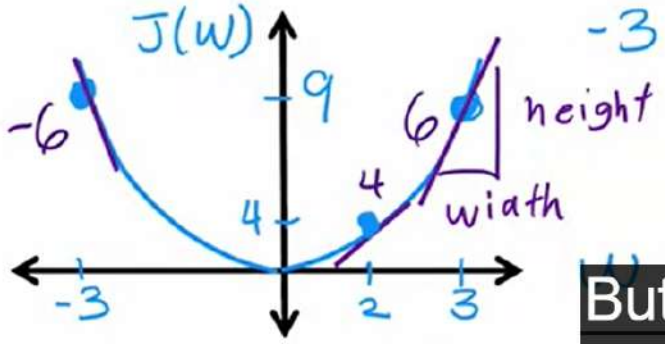
$J(w) \uparrow 4 \times 0.001$

$\downarrow 0.006$

$w = -3$ $\quad J(w) = w^2 = 9$ $\quad w \uparrow 0.001$ $\quad J(w) = J(-2.999) = 8.994001$ $\quad \frac{\partial}{\partial w} J(w) = -6$

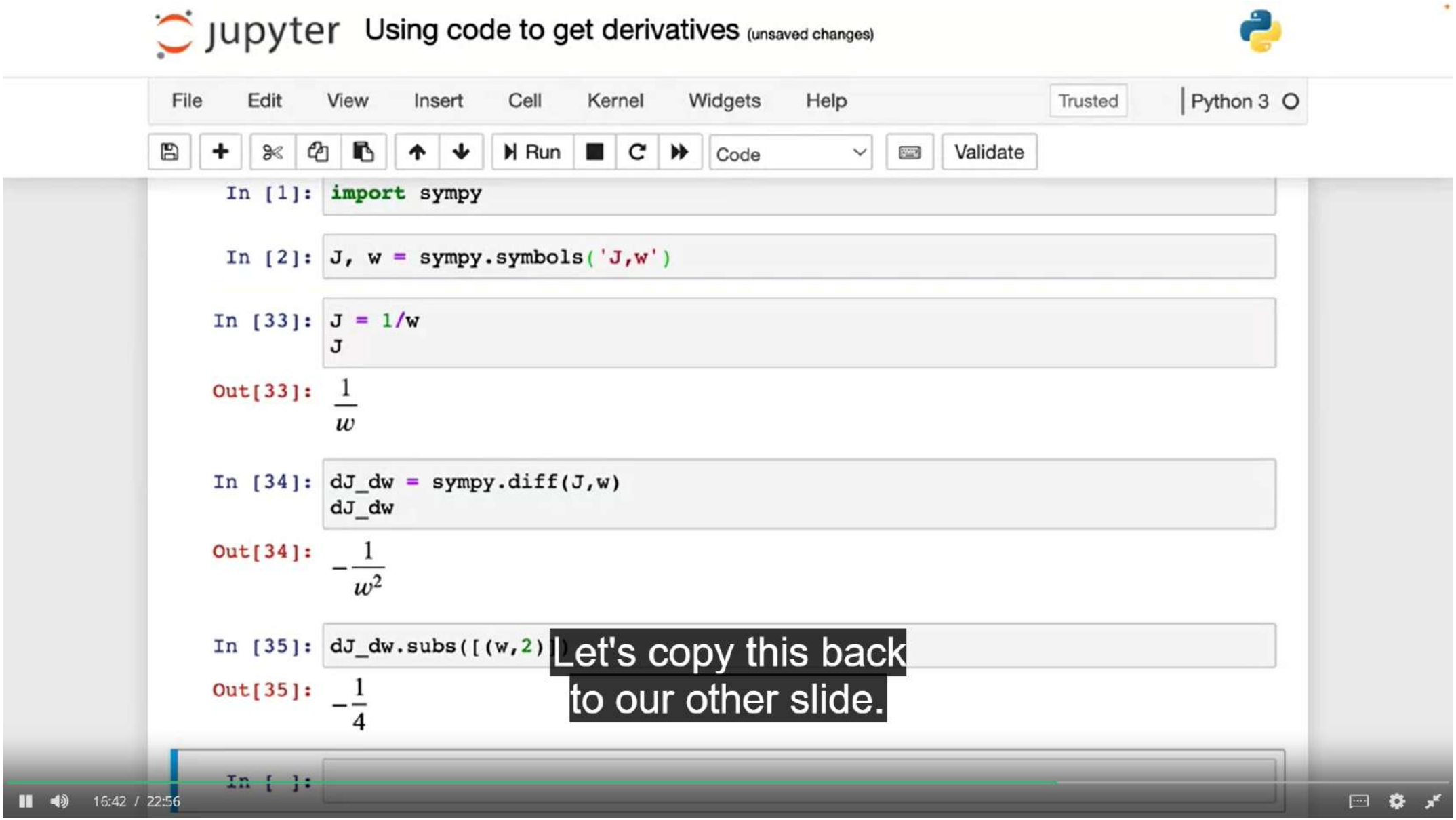
$J(w) \downarrow 6 \times 0.001$

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



$-3 + 0.001$

But if you haven't taken a calculus class before and


jupyter Using code to get derivatives (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Validate
In [1]: import sympy
In [2]: J, w = sympy.symbols('J,w')
In [33]: J = 1/w
J
Out[33]: 1/w
In [34]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[34]: -1/w^2
In [35]: dJ_dw.subs([(w,2)])
Out[35]: -1/4
In [ ]:
Let's copy this back
to our other slide.
16:42 / 22:56

Neural Network Example

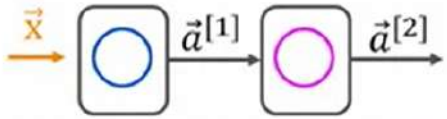
$x = 1 \quad y = 5$

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

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

ReLU activation

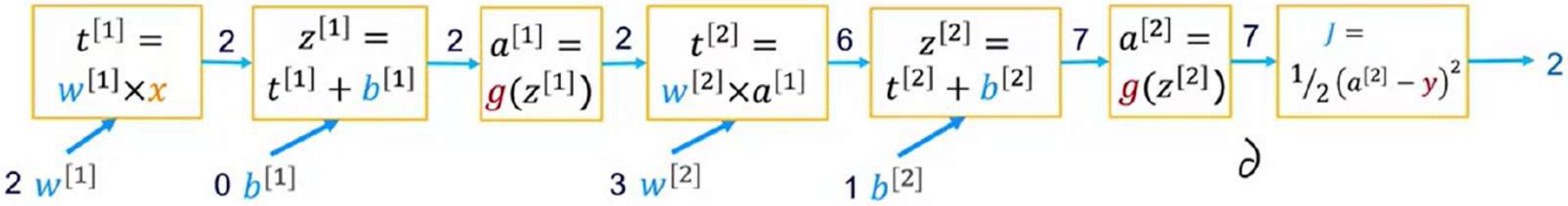
$g(z) = \max(0, z)$



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

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

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



But if you were to carry out backprop,
the first thing you do is ask,

Model selection (choosing a model)

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

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

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

⋮ ⋮

$d=10$ 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.

of the generalization error.

Model selection

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

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

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

⋮ ⋮ ⋮

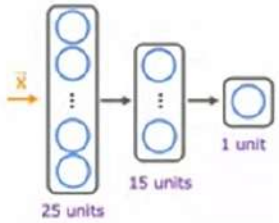
$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b$ $J_{cv}(w^{\langle 10 \rangle}, b^{\langle 10 \rangle})$

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

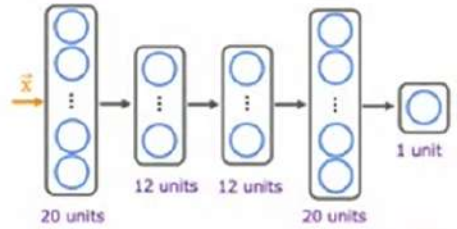
Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$

automatically make a
decision like what order

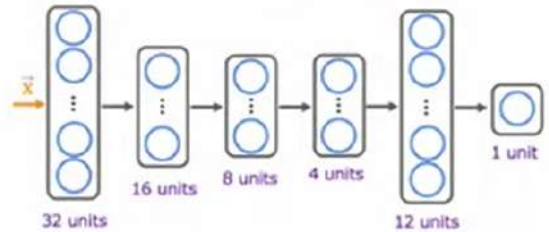
Model selection – choosing a neural network architecture

1. 

$w^{\langle 1 \rangle}, b^{\langle 1 \rangle}$ $J_{cv}(w^{\langle 1 \rangle}, b^{\langle 1 \rangle})$

2. 

$w^{\langle 2 \rangle}, b^{\langle 2 \rangle}$ $J_{cv}(w^{\langle 2 \rangle}, b^{\langle 2 \rangle})$

3. 

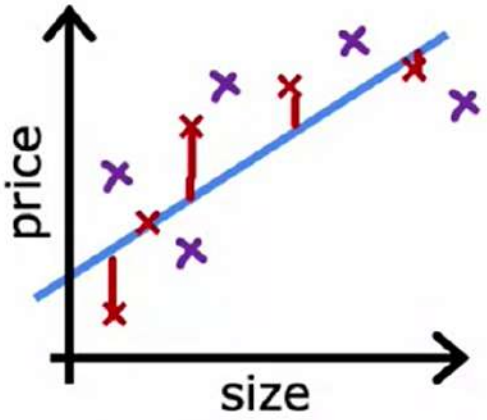
$w^{\langle 3 \rangle}, b^{\langle 3 \rangle}$ $J_{cv}(w^{\langle 3 \rangle}, b^{\langle 3 \rangle})$

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

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

You would compute this using

Bias/variance

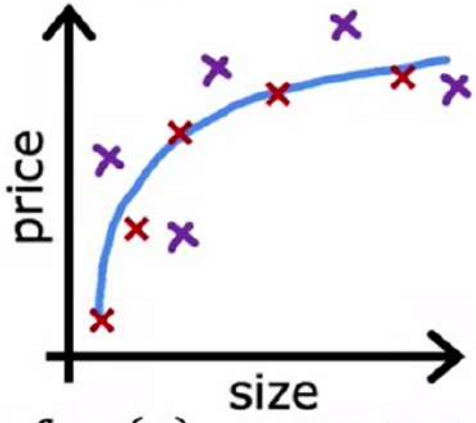


$f_{\vec{w},b}(x) = w_1 x + b$

High bias (underfit)

$d=1$ J_{train} is high

J_{cv} is high

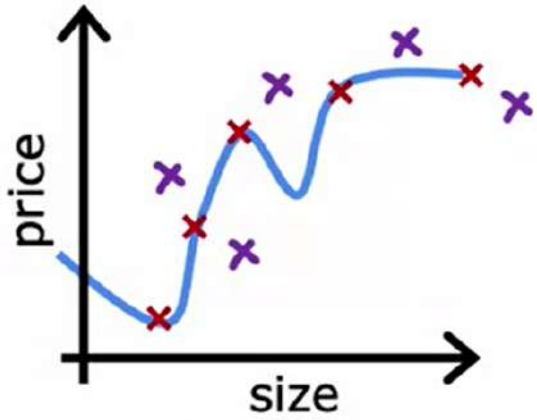


$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$

"Just right"

J_{train} is low

J_{cv} is low



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

High variance (overfit)

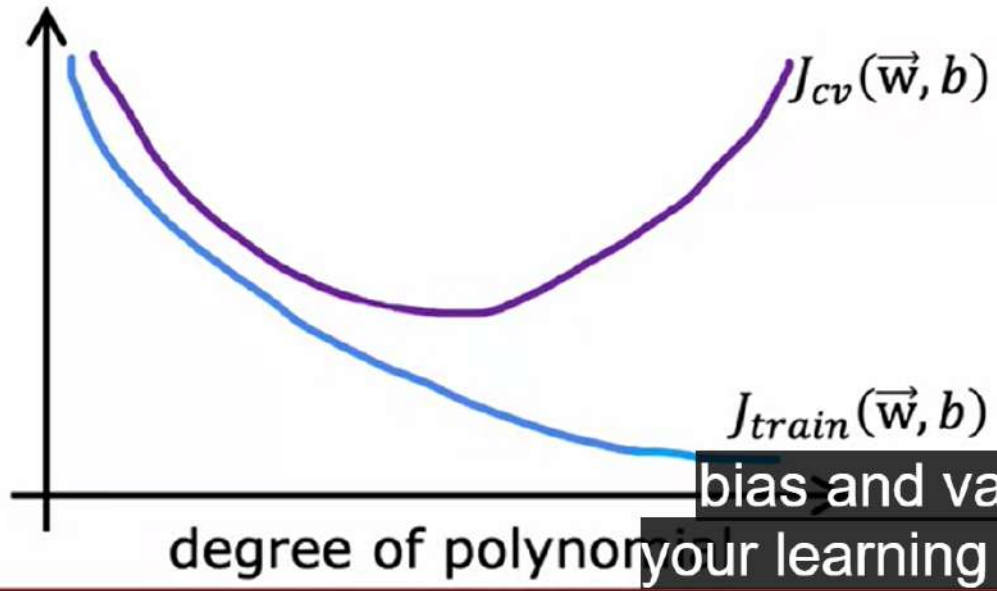
$d=4$ J_{train} is low

J_{cv} is high

but J_cv is high.

Diagnosing bias and variance

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



High bias (underfit)

J_{train} will be high

($J_{train} \approx J_{cv}$)

High variance (overfit)

$J_{cv} \gg J_{train}$

(J_{train} may be low)

High bias and high variance

J_{train} will be high

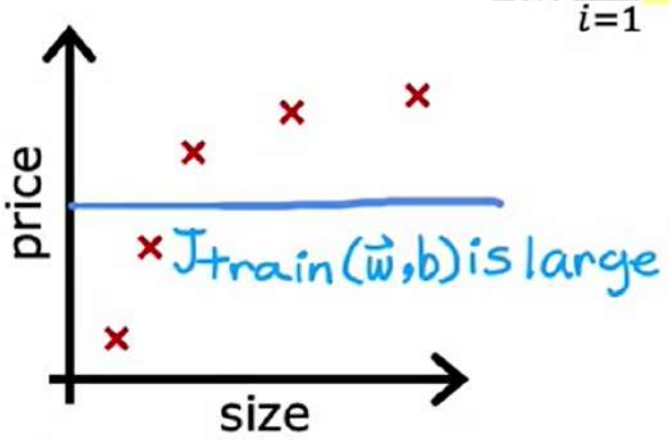
and $J_{cv} \gg J_{train}$

bias and variance in your learning algorithm?

Linear regression with regularization

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

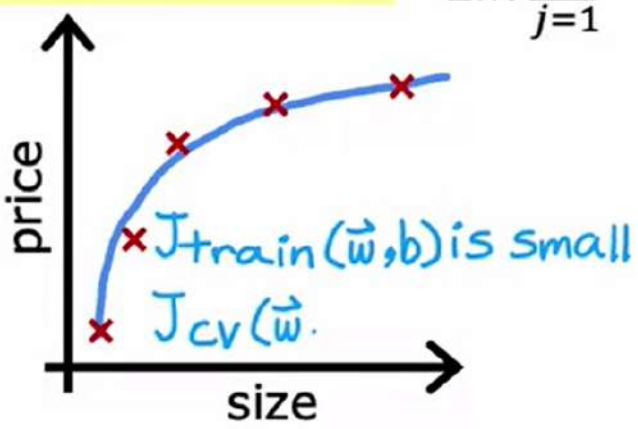
$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$



Large λ
High bias (underfit)

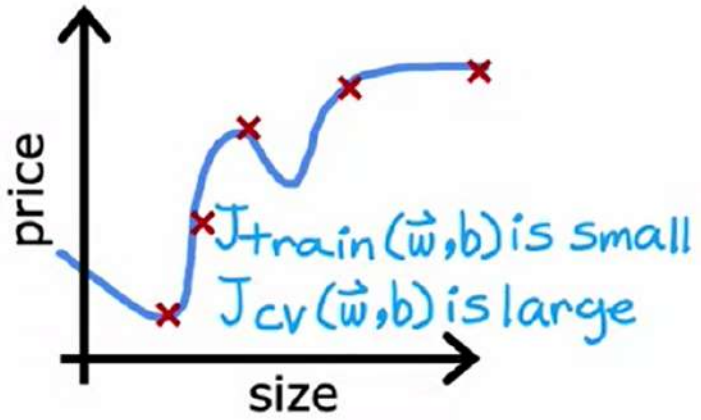
$\lambda = 10,000$ $w_1 \approx 0, w_2 \approx 0$

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



Intermediate λ

with small J_train and small J_cv.

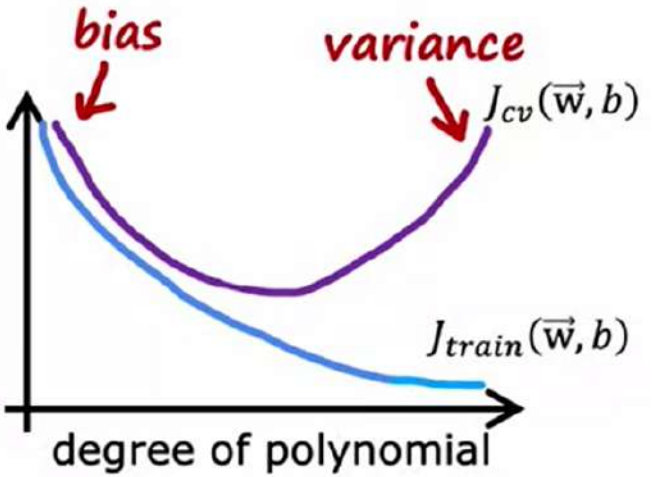


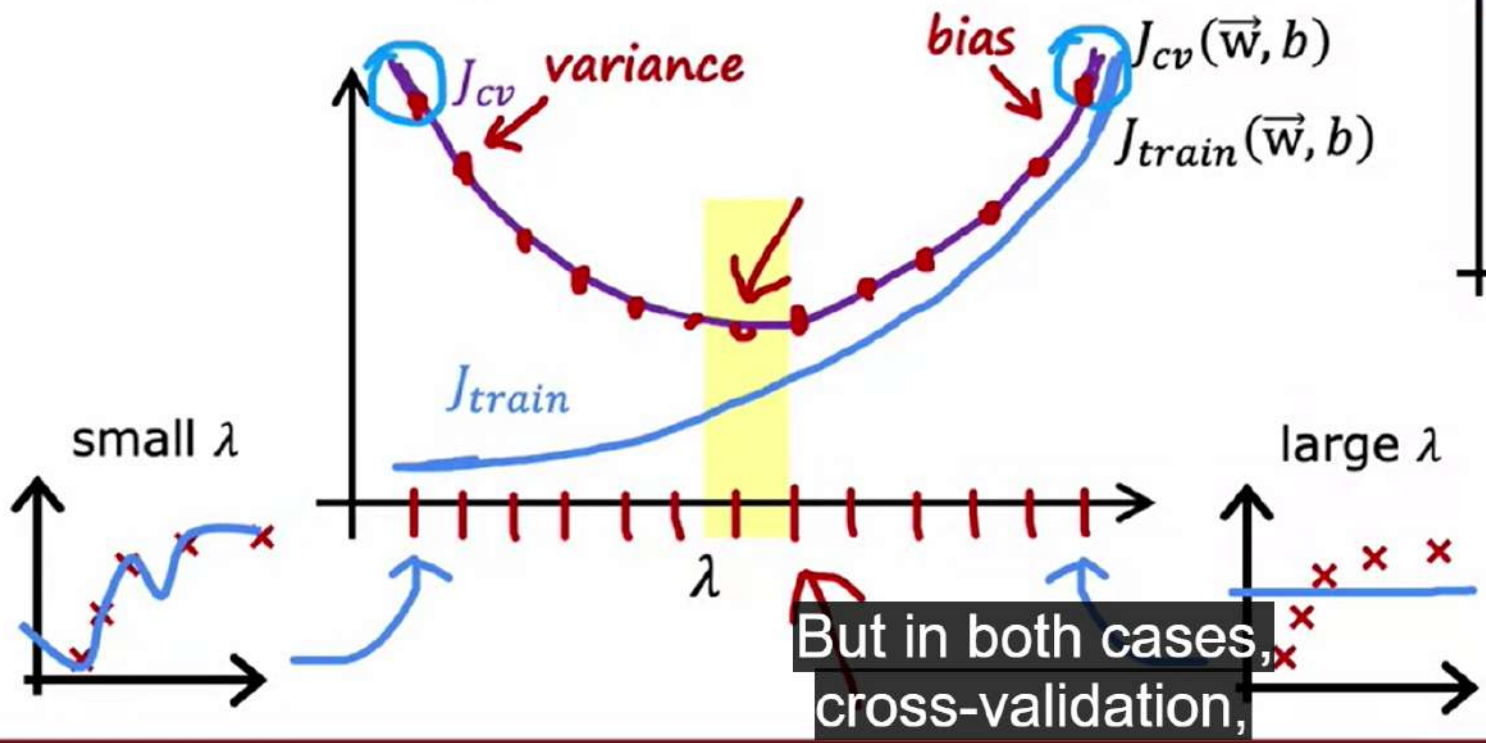
Small λ
High variance (overfit)

$\lambda = 0$

Bias and variance as a function of regularization parameter λ

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





But in both cases, cross-validation,

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

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

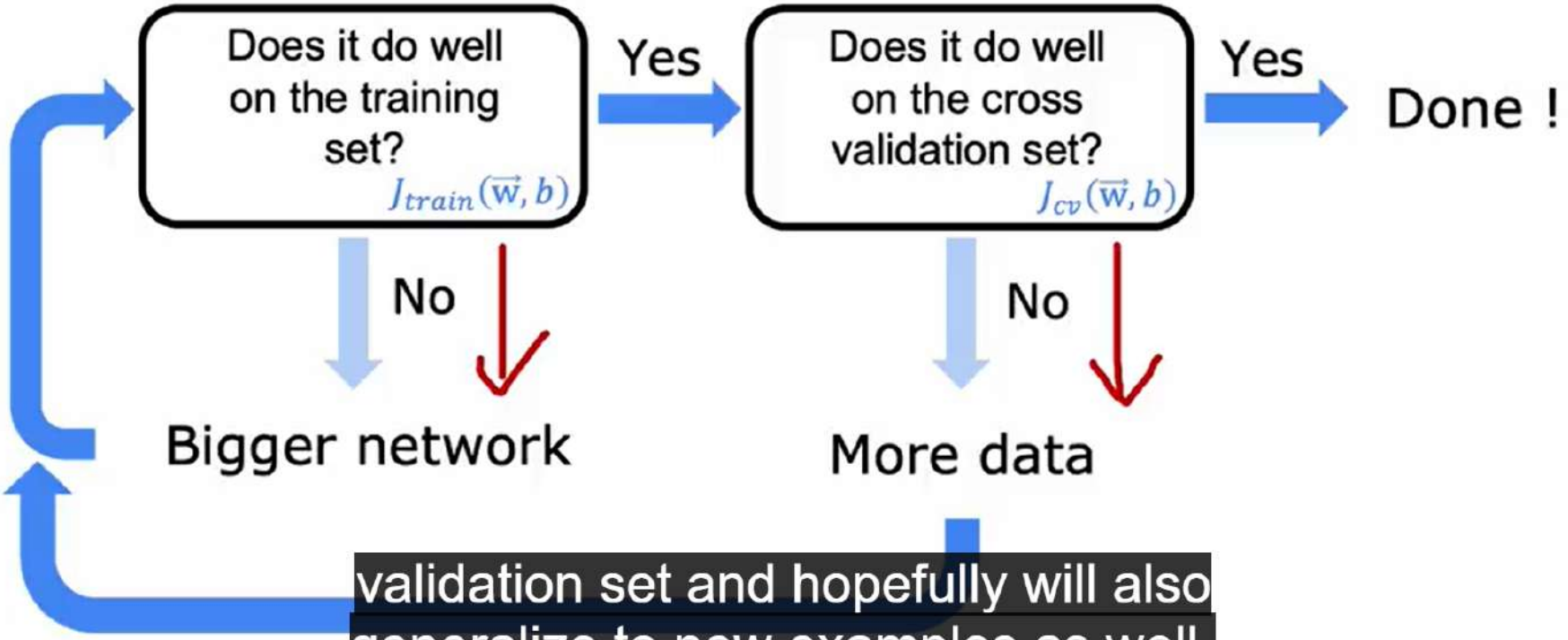
But it makes unacceptably large errors in predictions. What do you try next?

→ Get more training examples	fixes high variance
→ Try smaller sets of features $x, x^2, x^3, x^4, x^5 \ldots$	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 λ	fixes high bias
→ Try increasing λ	fixes high variance

I realized that this was a lot of stuff on this slide.

Neural networks and bias variance

Large neural networks are low bias machines



validation set and hopefully will also generalize to new examples as well.

Neural network regularization

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

b

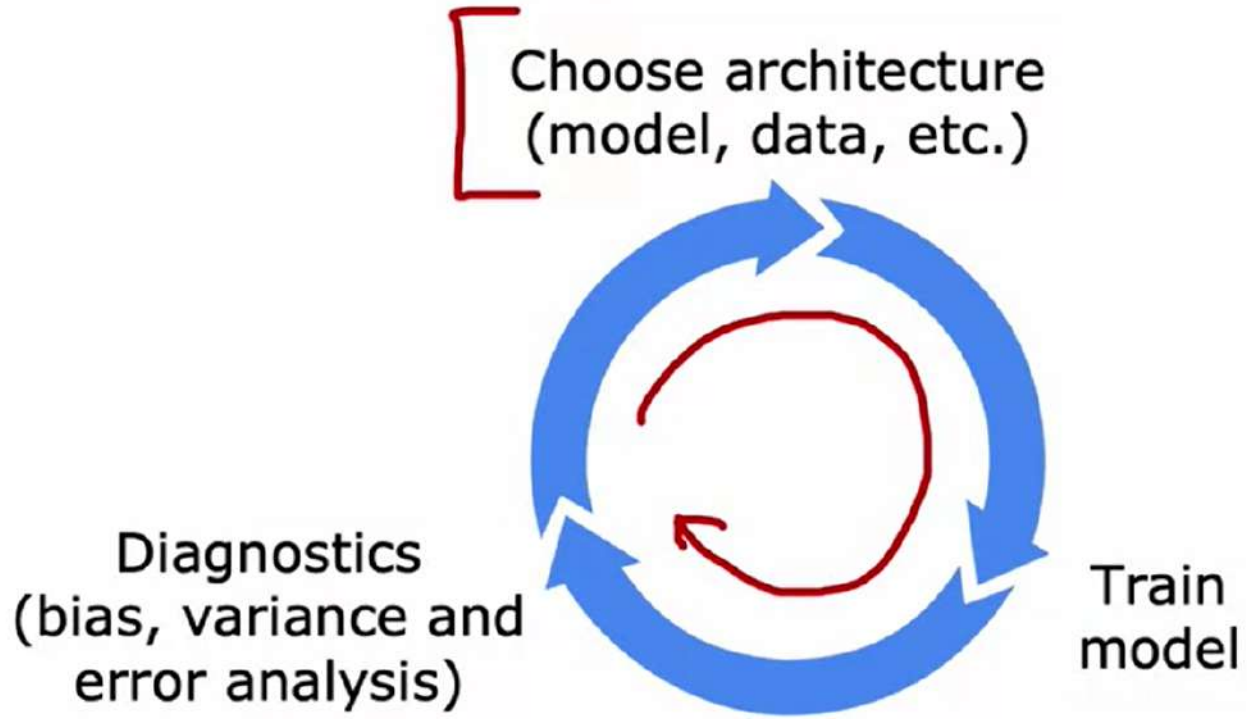
Unregularized MNIST model

```
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])
```

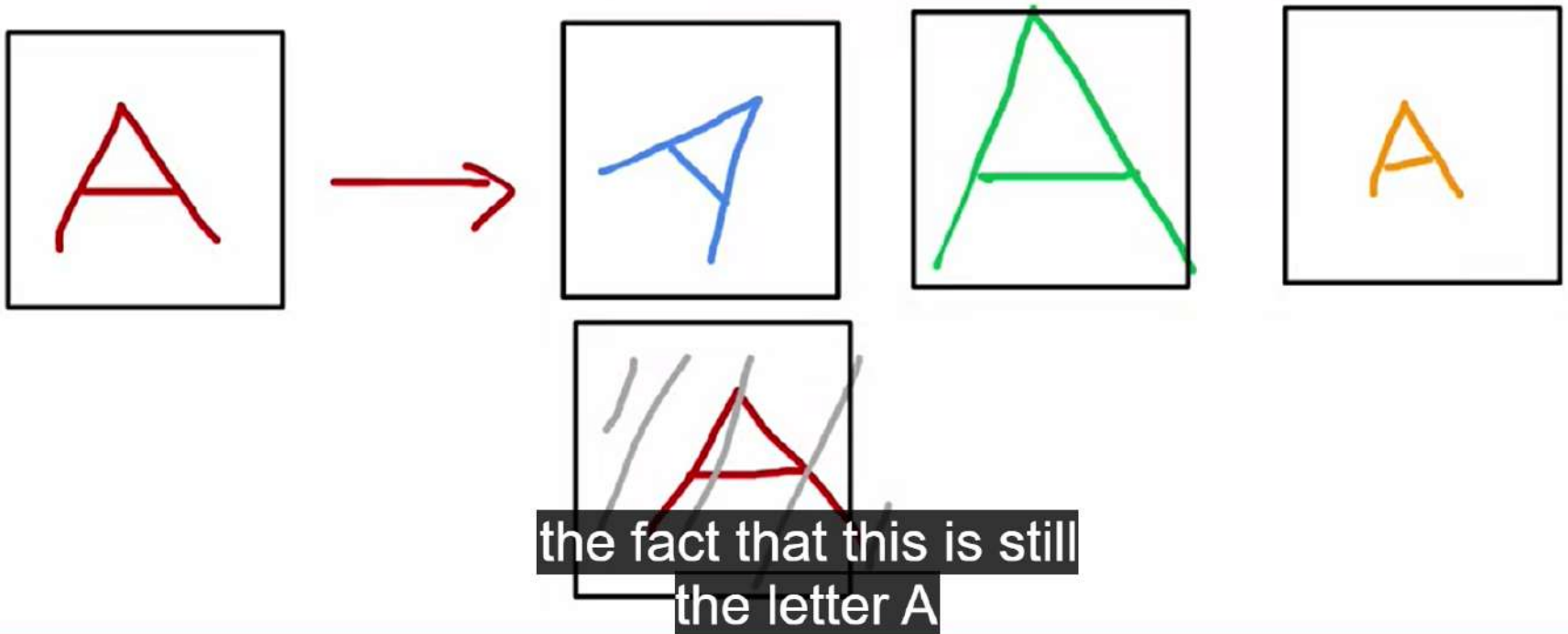
Iterative loop of ML development



variance diagnostic as well as

Data augmentation

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



the fact that this is still
the letter A

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

audio sound like you're recording it on
a bad cell phone connection like this.

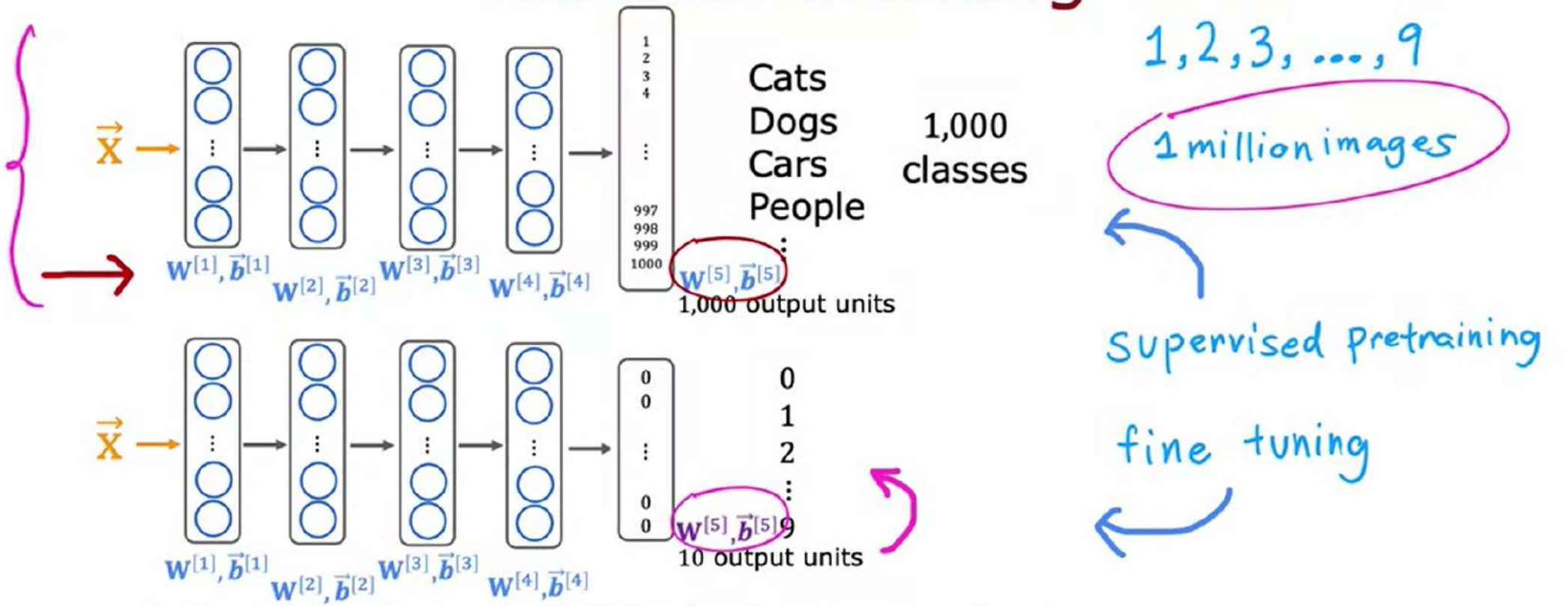
Artificial data synthesis for photo OCR

Real d etic data

So with synthetic data generation like this you can generate a very

[Adam Coates and Tao Wang]

Transfer learning



Option 1: only train output layers parameters.

Option 2: train all parameters

and just do a little bit of fine tuning to quickly

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).

1 million images

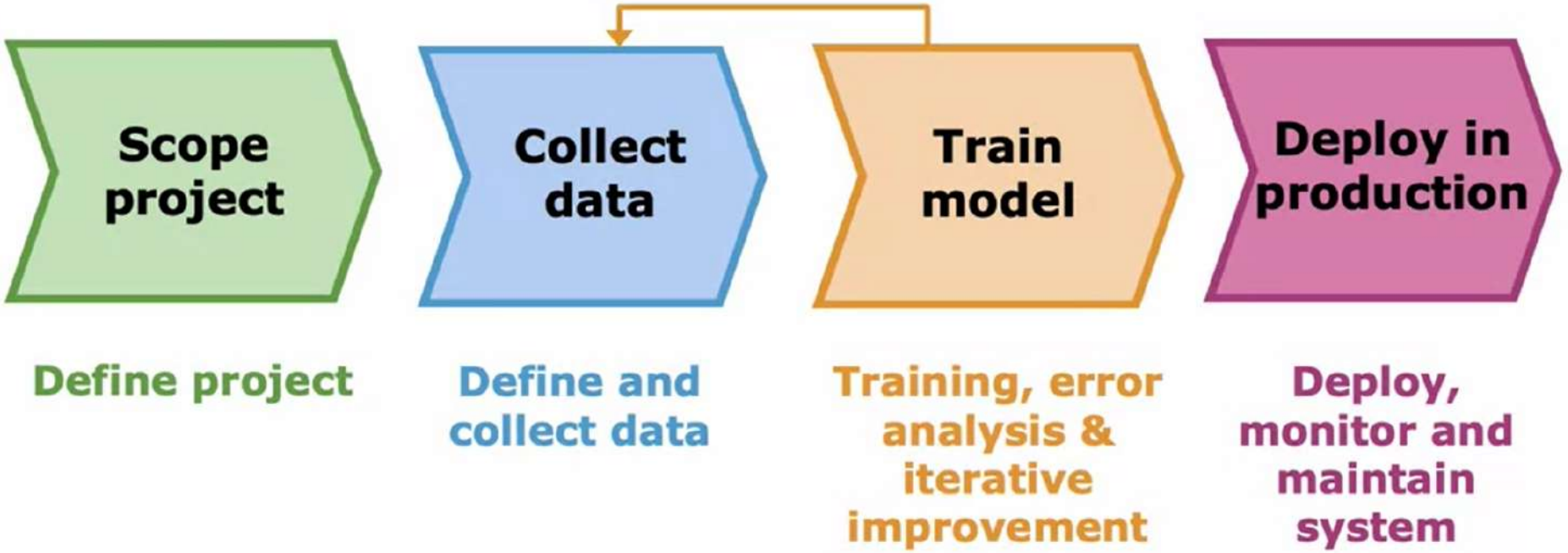
2. Further train (fine tune) the network on your own data.

1000 images

50 images

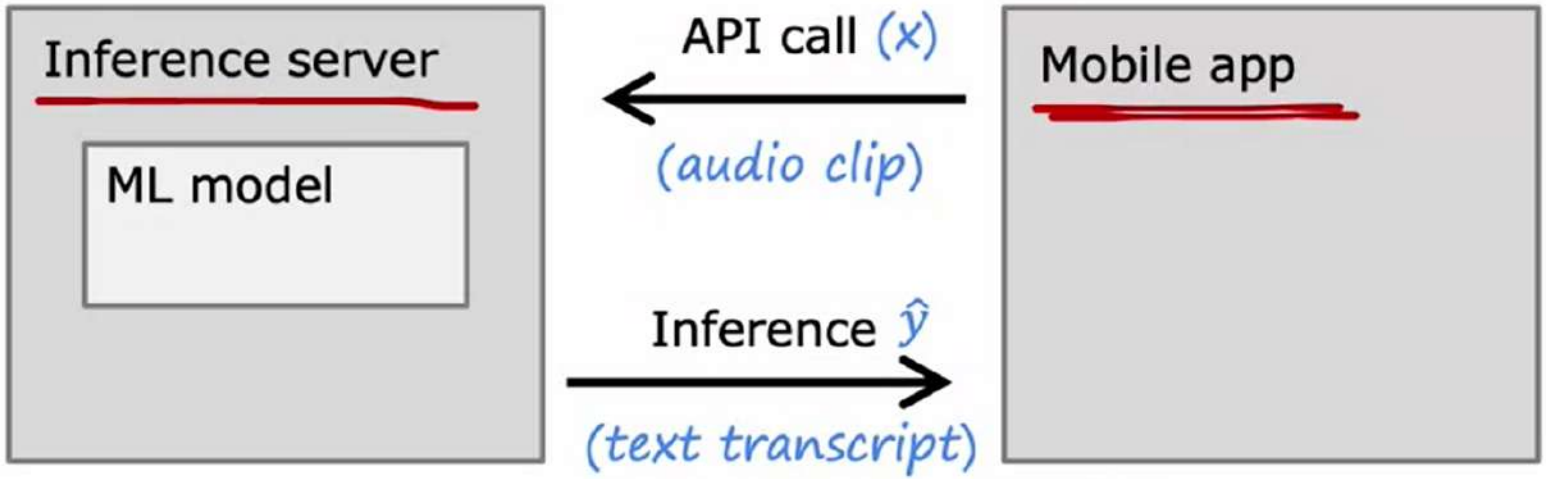
potentially much
smaller dataset.

Full cycle of a machine learning project



What that means is you make it

Deployment



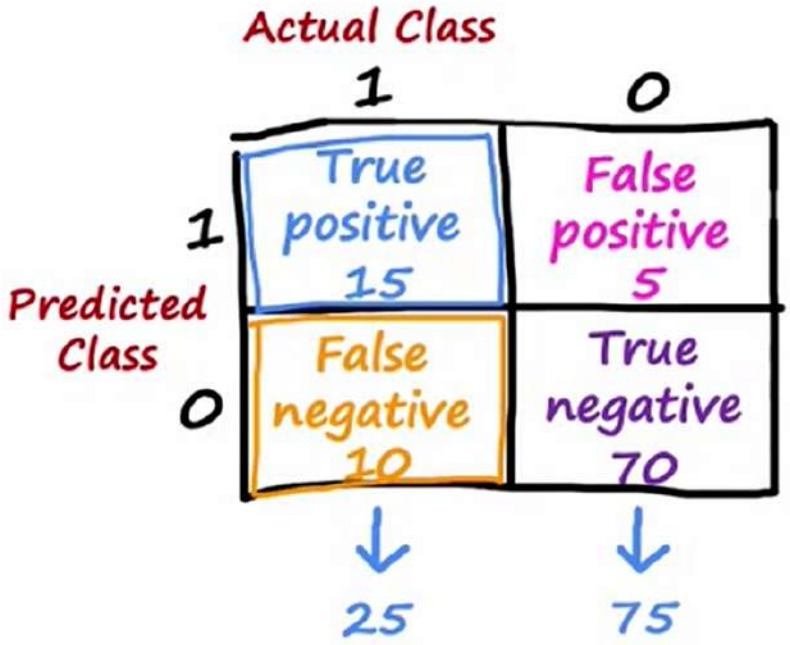
→ Software engineering may be needed for:

Ensure reliable and efficient predictions
Scaling
Logging
System monitoring
Model updat has good laws, is monitored,

MLOps
machine learning operations

Precision/recall

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



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?)

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

which is 0.6 or 60 percent.

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~~ ~~0.4~~ 0.3

→ Predict 0 if $f_{\vec{w},b}(\vec{x}) <$ ~~0.5~~ ~~0.7~~ ~~0.4~~ 0.3

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

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

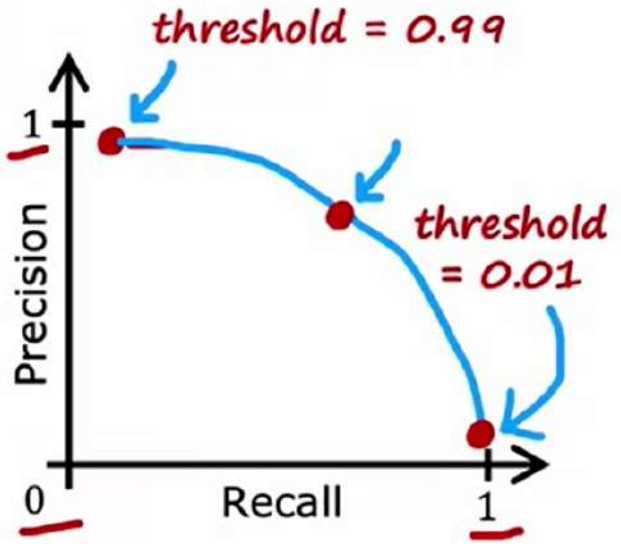
Suppose we want to predict $y = 1$ (rare disease) only if very confident.

higher precision, lower recall

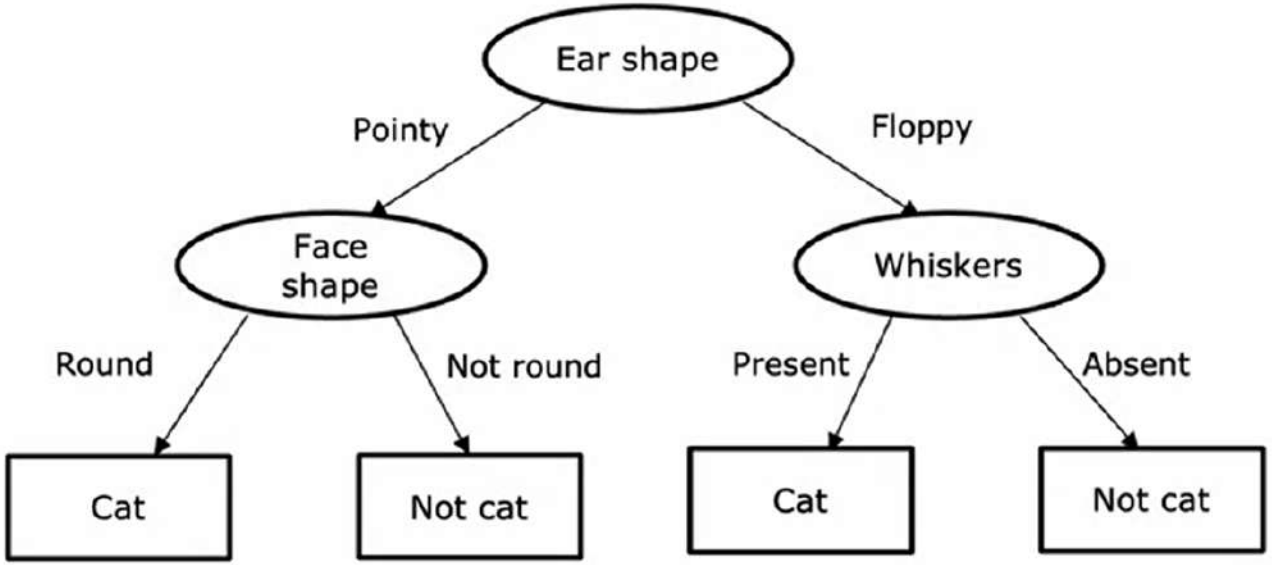
Suppose we want to avoid missing too many case of rare disease (when in doubt predict $y = 1$)

lower precision, higher recall

More generally predict 1 if: $f_{\vec{w},b}(\vec{x}) \geq$ threshold.

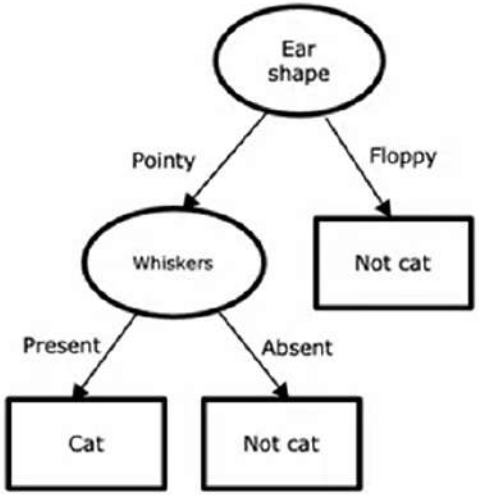


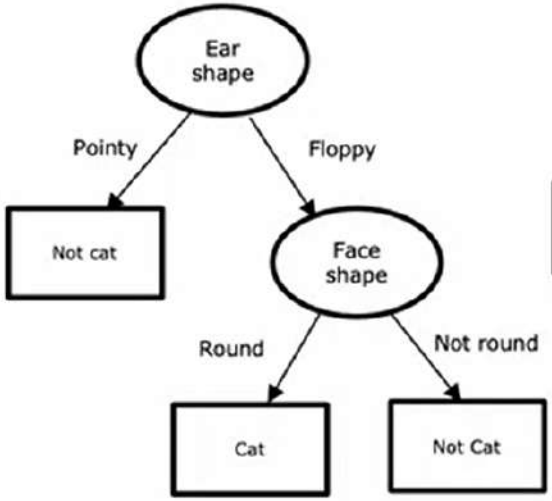
Decision Tree

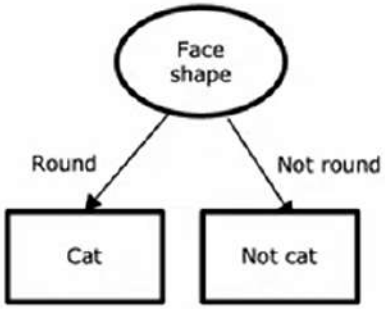


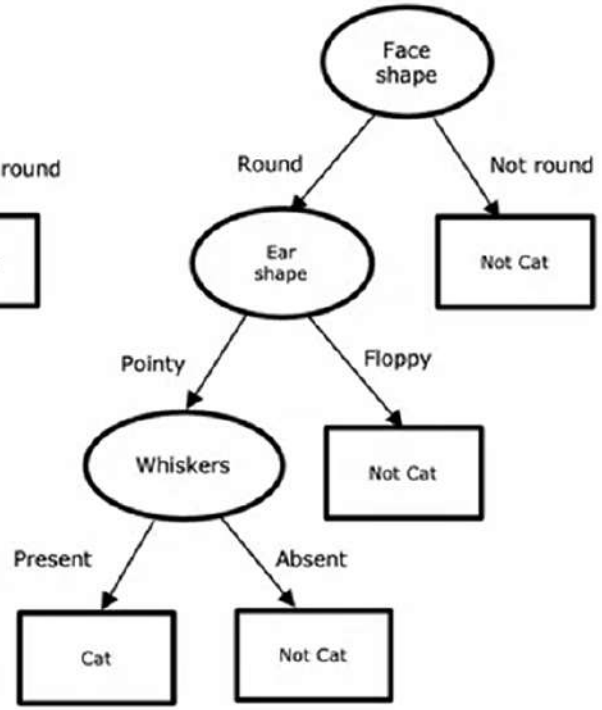
the learning algorithm
looks like a tree,

Decision Tree





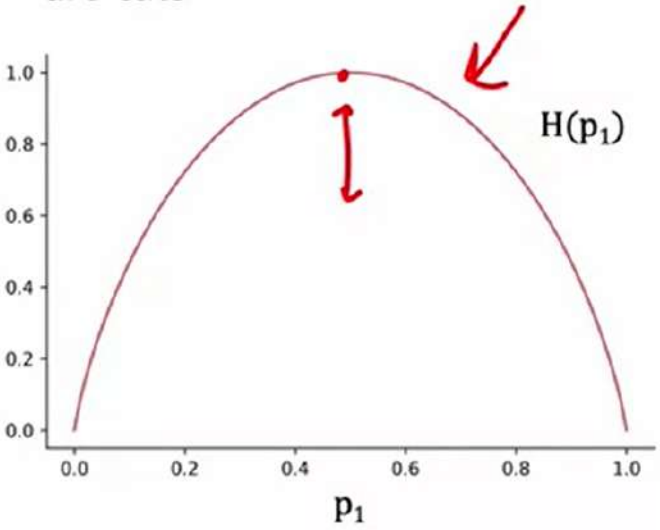




to try to pick one that

Entropy as a measure of impurity

p_1 = fraction of examples that are cats



$$p_0 = 1 - p_1$$

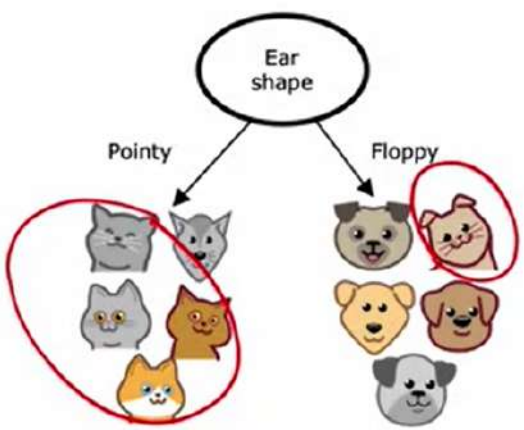
$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$

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

Note: "0 log(0)" = 0

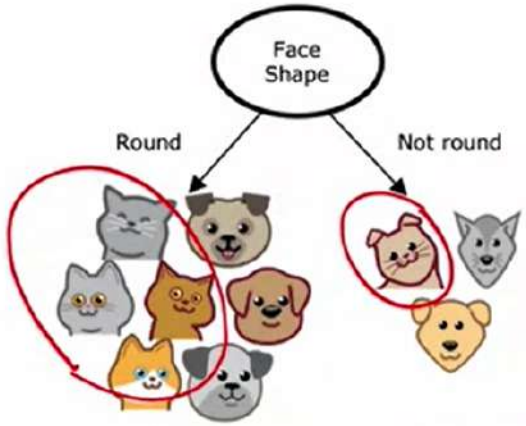
To summarize, the entropy function is

Choosing a split



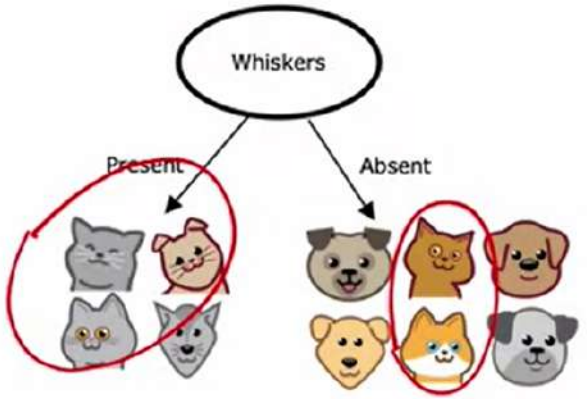
$p_1 = {}^4/_5 = 0.8$ $p_1 = {}^1/_5 = 0.2$

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



$p_1 = {}^4/_7 = 0.57$ $p_1 = {}^1/_3 = 0.33$

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

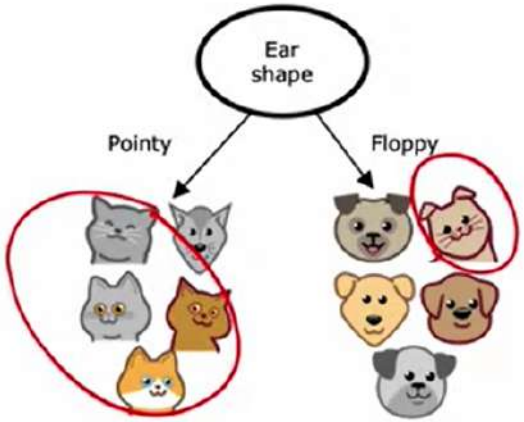


$p_1 = {}^3/_4 = 0.75$ $p_1 = {}^2/_6 = 0.33$

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

given these three options

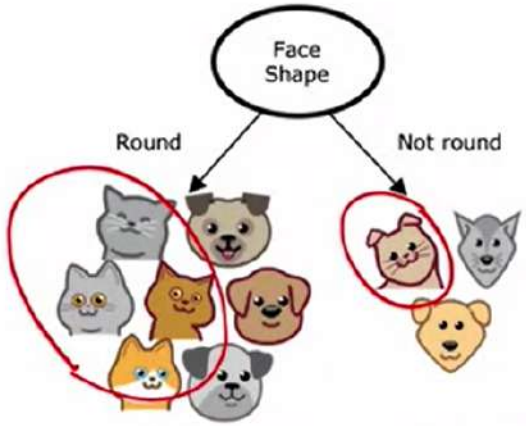
Choosing a split



$p_1 = {}^4/_5 = 0.8 \quad p_1 = {}^1/_5 = 0.2$

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

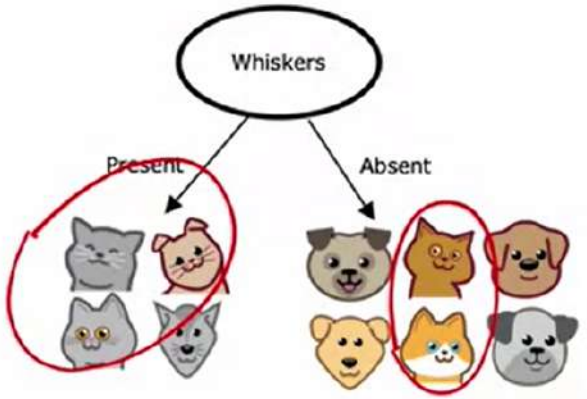
$$\left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right)$$



$p_1 = {}^4/_7 = 0.57 \quad p_1 = {}^1/_3 = 0.33$

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

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



$p_1 = {}^3/_4 = 0.75 \quad p_1 = {}^2/_6 = 0.33$

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

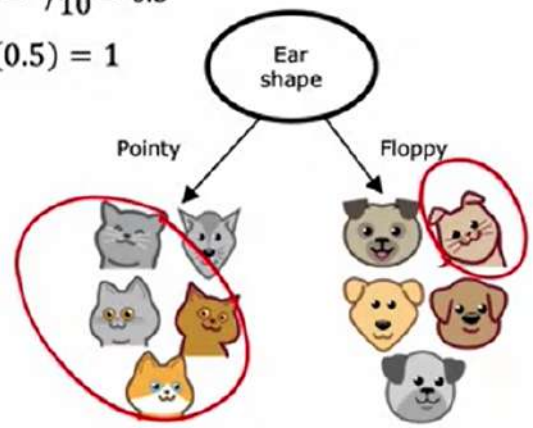
$$\left(\frac{4}{10}H(0.75) + \frac{6}{10}H(0.33)\right)$$

If we go to the root node,

Choosing a split

$p_1 = {}^{5}/_{10} = 0.5$

$H(0.5) = 1$



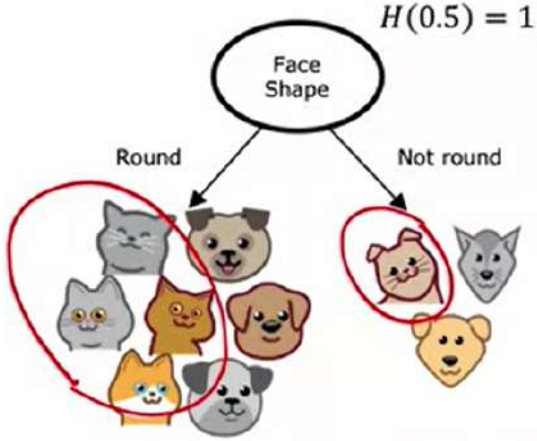
$p_1 = {}^{4}/_{5} = 0.8$ $p_1 = {}^{1}/_{5} = 0.2$

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

$$H(0.5) - \left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right)$$

$= 0.28$

$H(0.5) = 1$



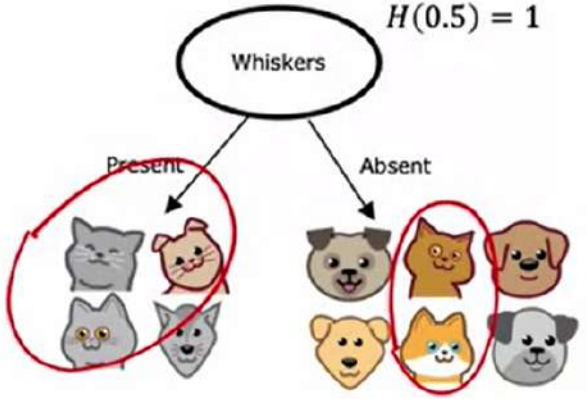
$p_1 = {}^{4}/_{7} = 0.57$ $p_1 = {}^{1}/_{3} = 0.33$

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

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

$= 0.03$

$H(0.5) = 1$



$p_1 = {}^{3}/_{4} = 0.75$ $p_1 = {}^{2}/_{6} = 0.33$

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

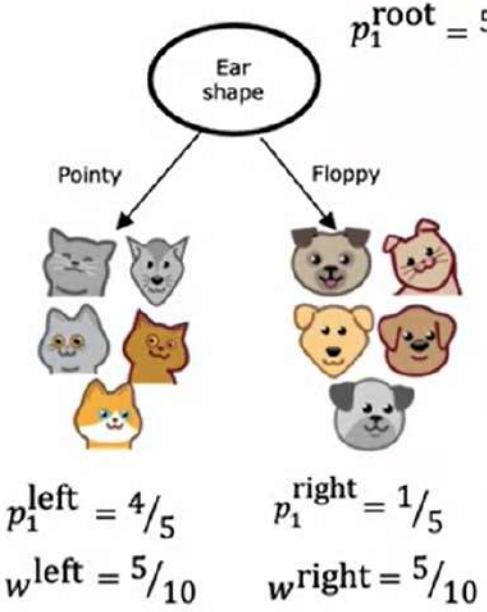
$$H(0.5) - \left(\frac{4}{10}H(0.75) + \frac{6}{10}H(0.33)\right)$$

$= 0.12$

Information gain

Information Gain

$p_1^{\text{root}} = {}^5/_{10} = 0.5$

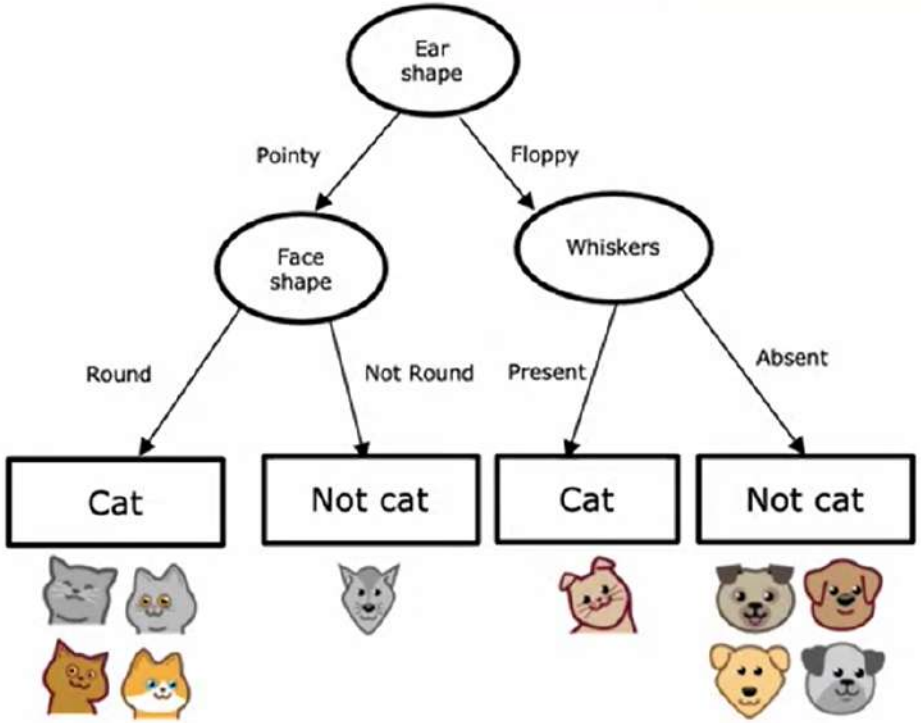


Information gain

$$= \mathrm{H}(p_1^{\text{root}}) - \left(w^{\text{left}} H\left(p_1^{\text{left}}\right) + w^{\text{right}} H\left(p_1^{\text{right}}\right) \right)$$

That will result in, hopefully,

Recursive splitting



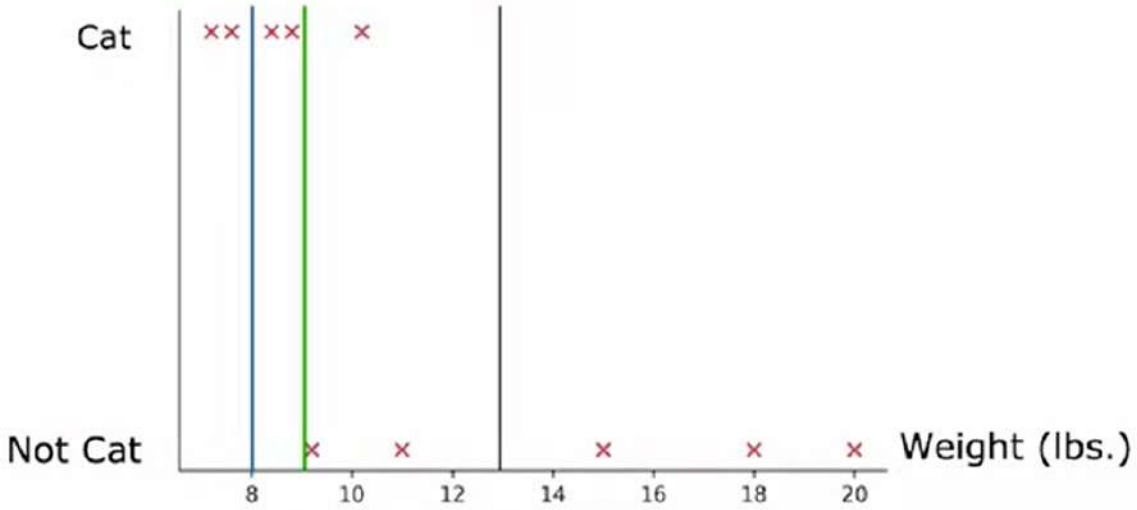
Notice that there's

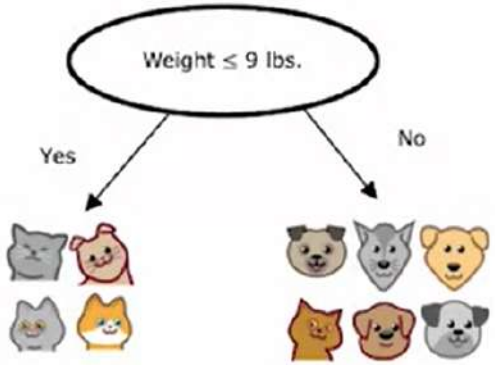
One hot encoding

	~~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

And so instead of one feature
taking on three possible values,

Splitting on a continuous variable





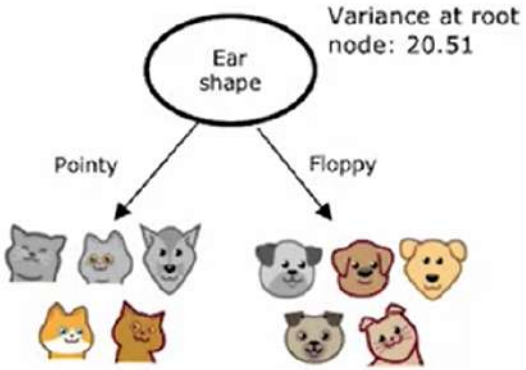
$$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$$

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

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

you can then build recursive,
li additional decision trees using

Choosing a split



Weights: 7.2, 9.2, 8.4, 7.6, 10.2

Variance: 1.47

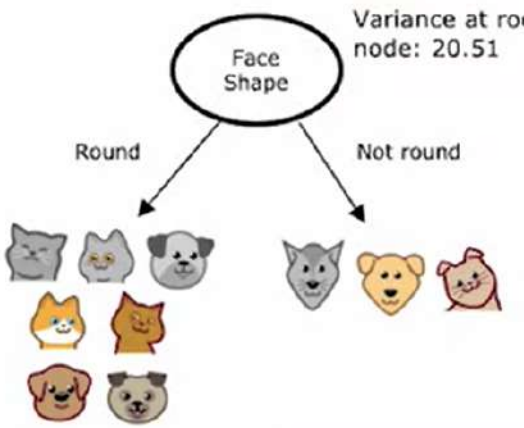
Weights: 8.8, 15, 11, 18, 20

Variance: 21.87

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

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

$$= 8.84$$



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

Variance: 27.80

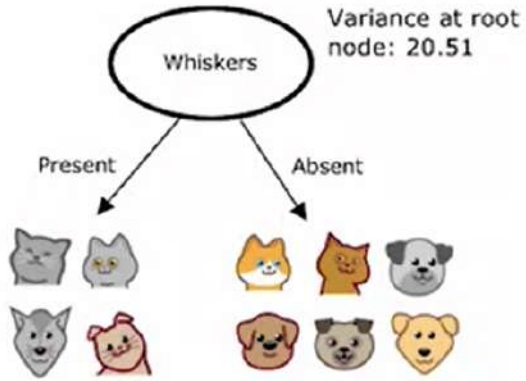
Weights: 8.8, 9.2, 11

Variance: 1.37

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

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

$$= 0.64$$



Weights: 7.2, 8.8, 9.2, 8.4

Variance: 0.75

Weights: 15, 7.6, 11, 10.2, 18, 20

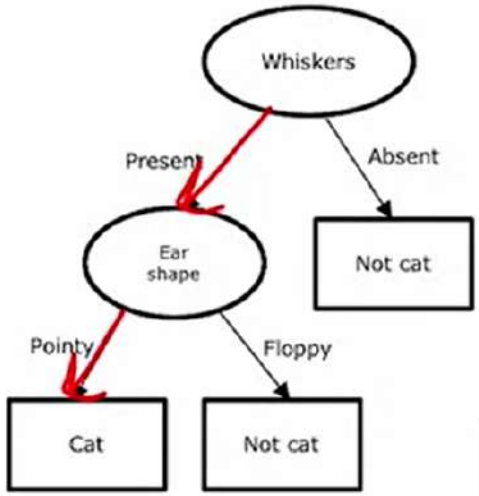
Variance: 23.32

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

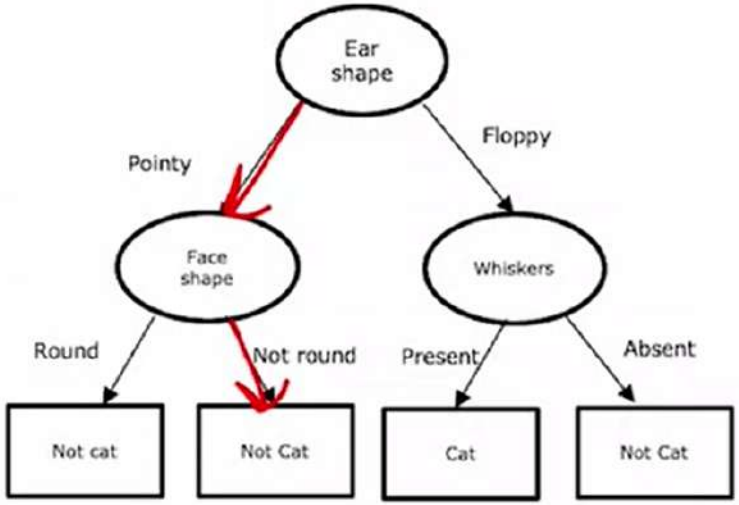
$$20.51 - \left(\frac{4}{10} * 0.75 + \frac{6}{10} * 23.32\right)$$

$$= 6.22$$

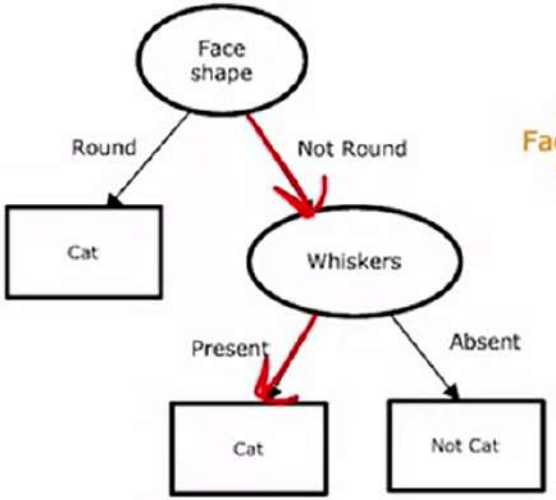
Tree ensemble



Prediction: Cat



Prediction: Not cat



Prediction: Cat

New test example

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

which happens to be the
correct prediction.

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.

So in our example we had three features available rather than picking from all end

Using XGBoost

Classification

```
from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

practitioners will use the open source libraries that implement XGBoost.

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)
```

classification, then this line
here just becomes XGBRegressor and

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

critical to getting
competitive performance.