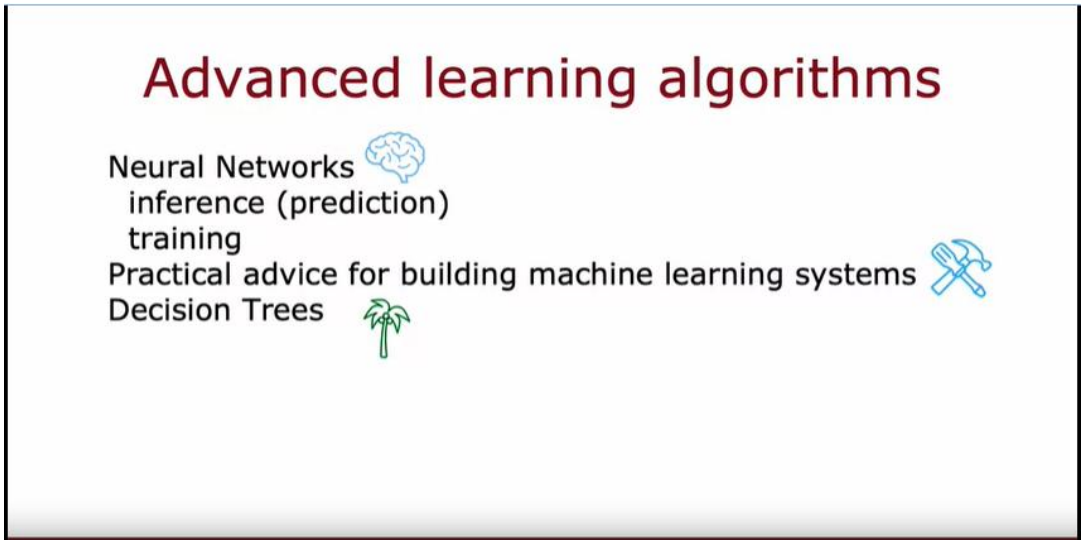
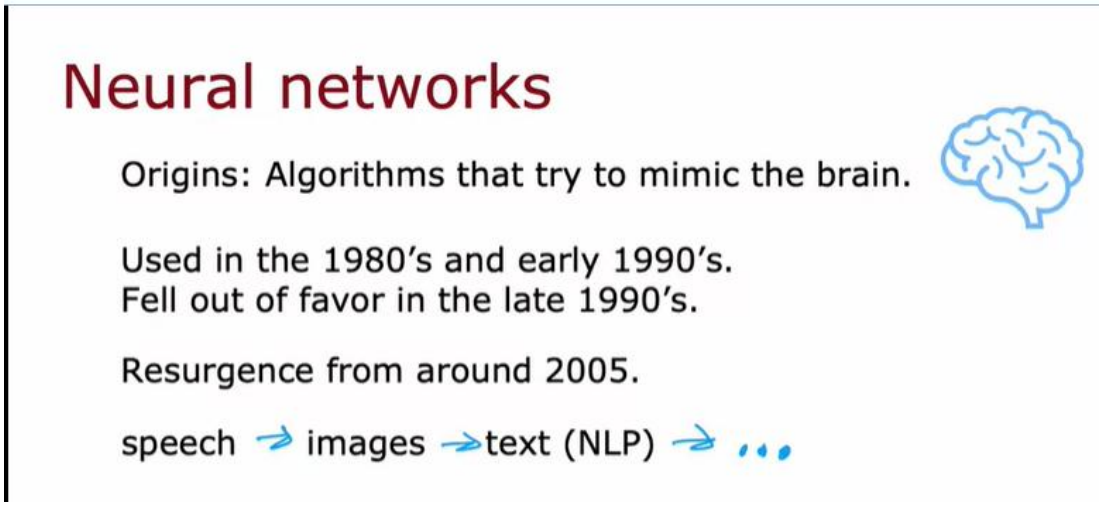
Supervised Machine Learning: **Index**

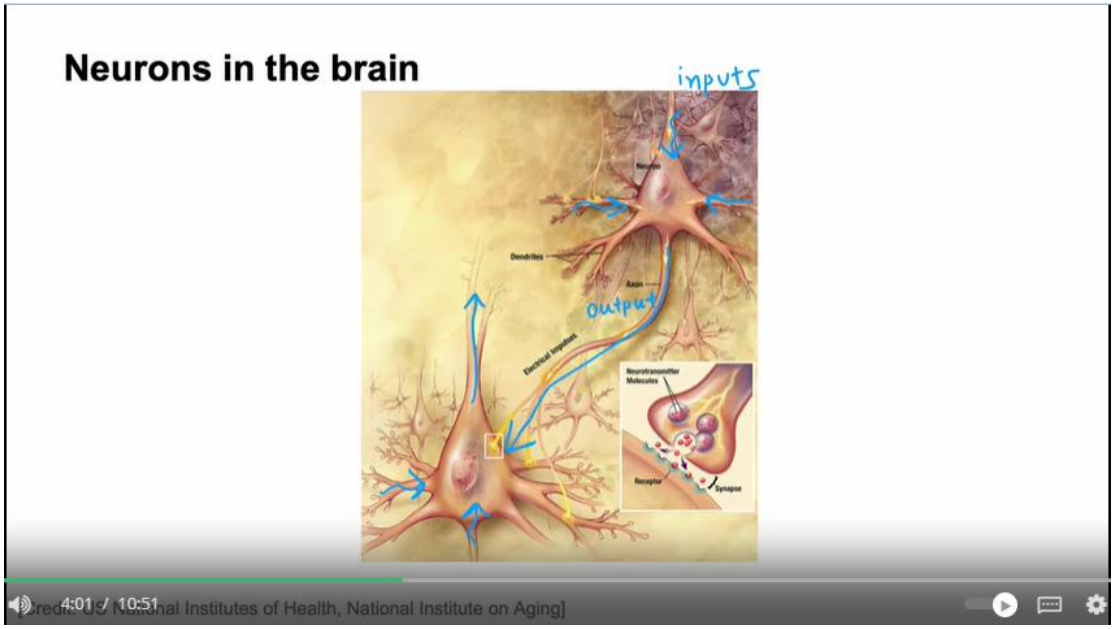
Part (Week) 1 : Neural Networks

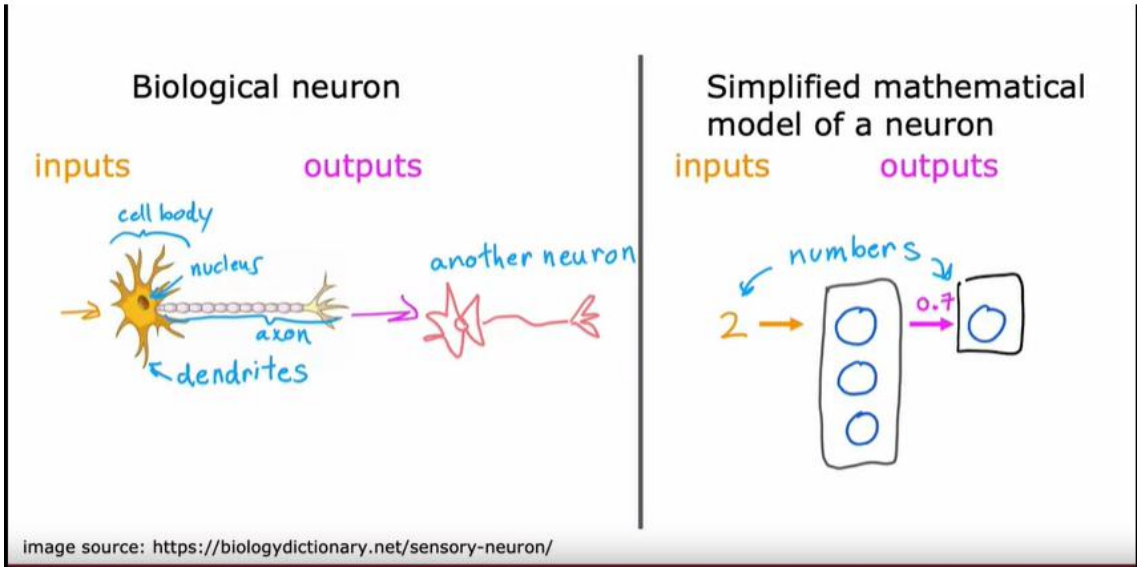
2.1 - Neural networks intuition : 2.1.1 - Welcome :

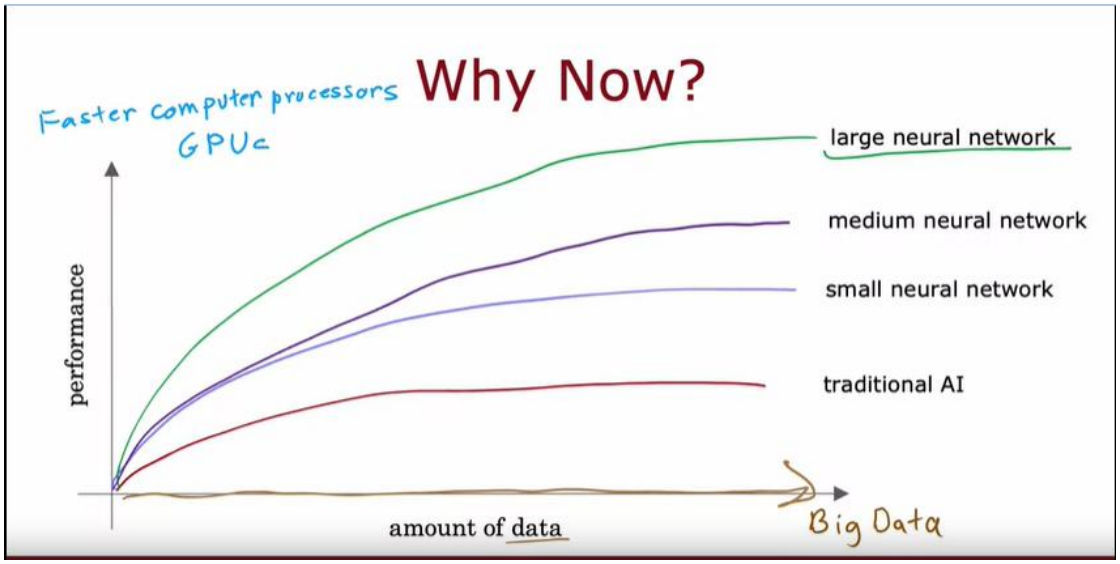


2.1 - Neural networks intuition : 2.1.2 - Neurons and the brain :

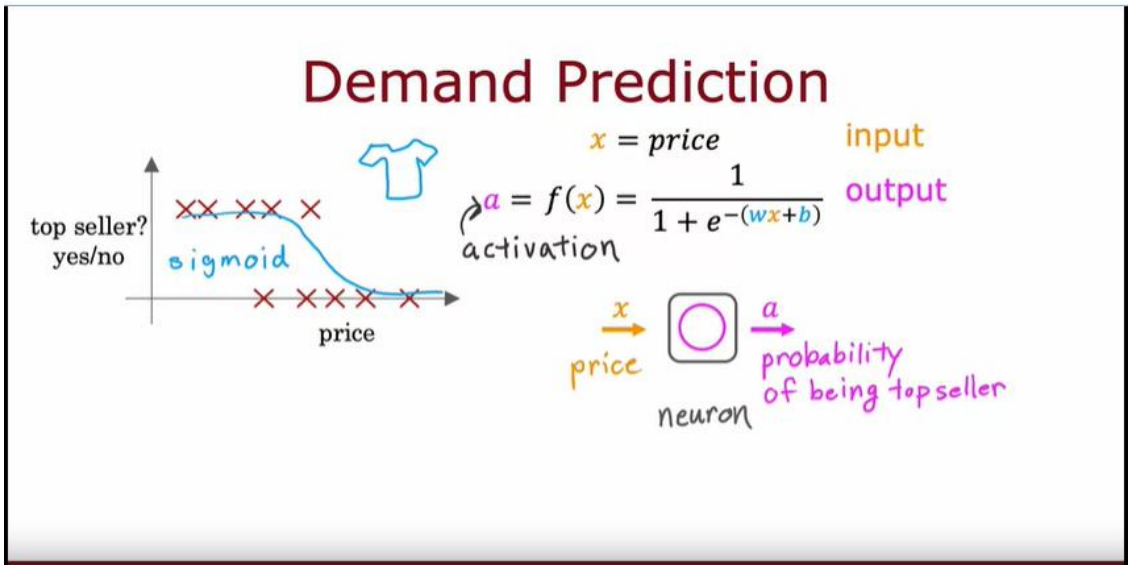


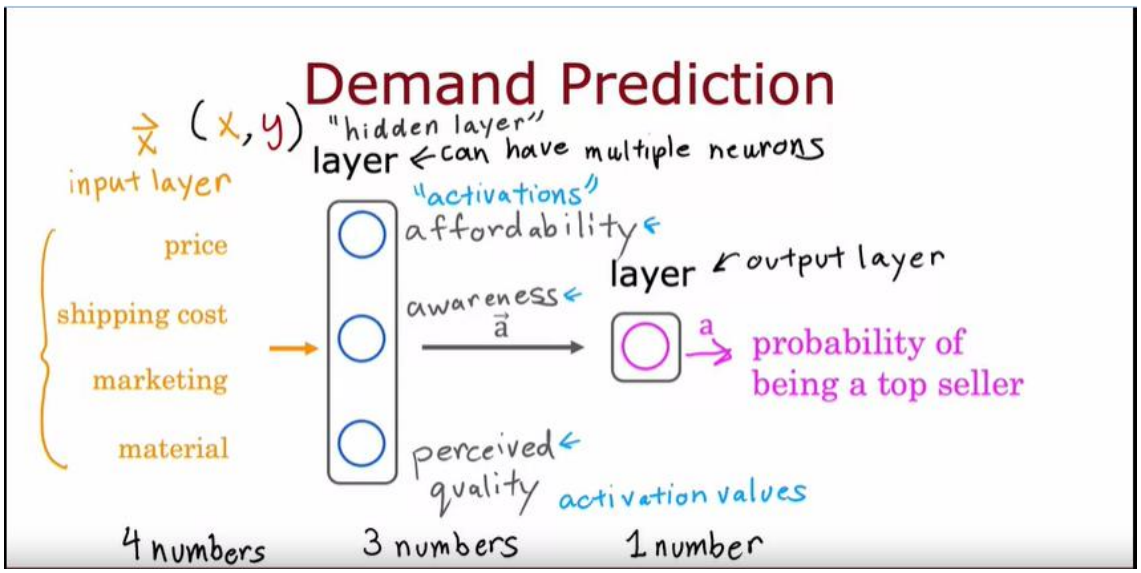


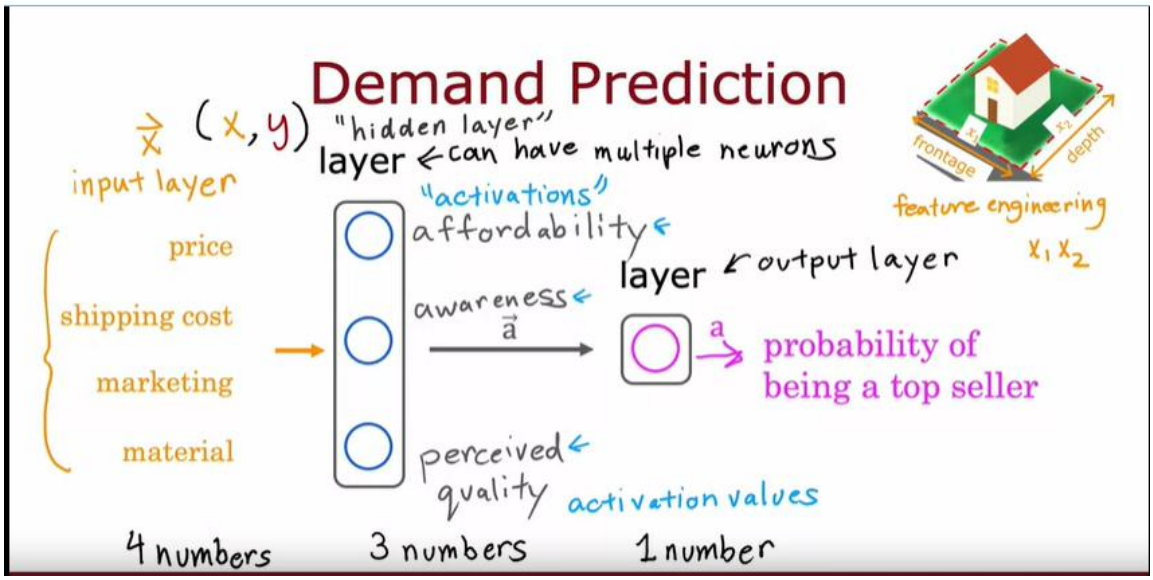


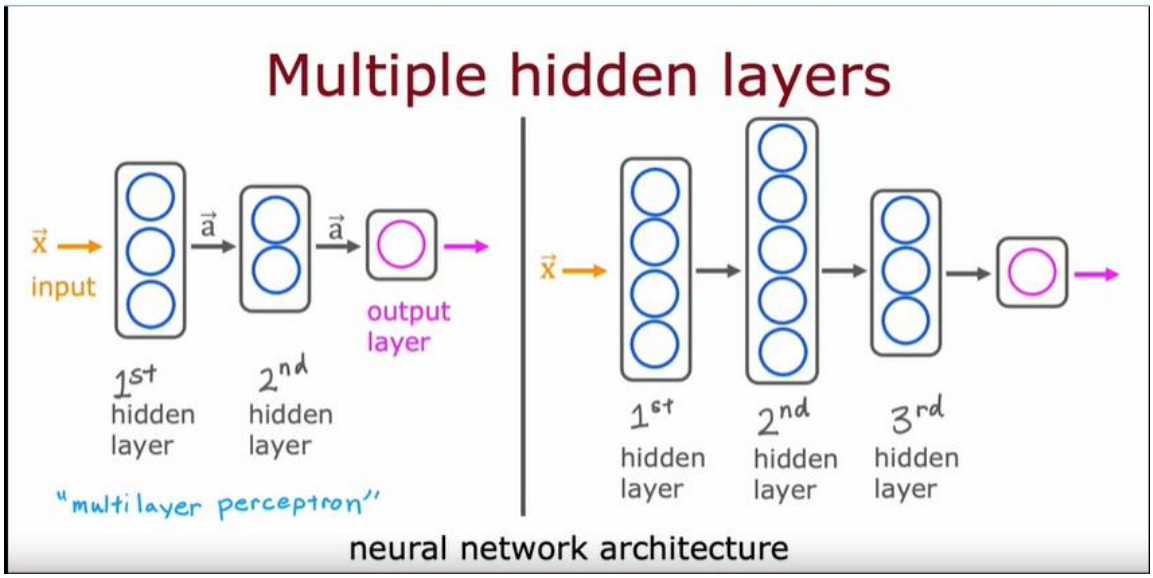


2.1 - Neural networks intuition : 2.1.3 - Demand Prediction :



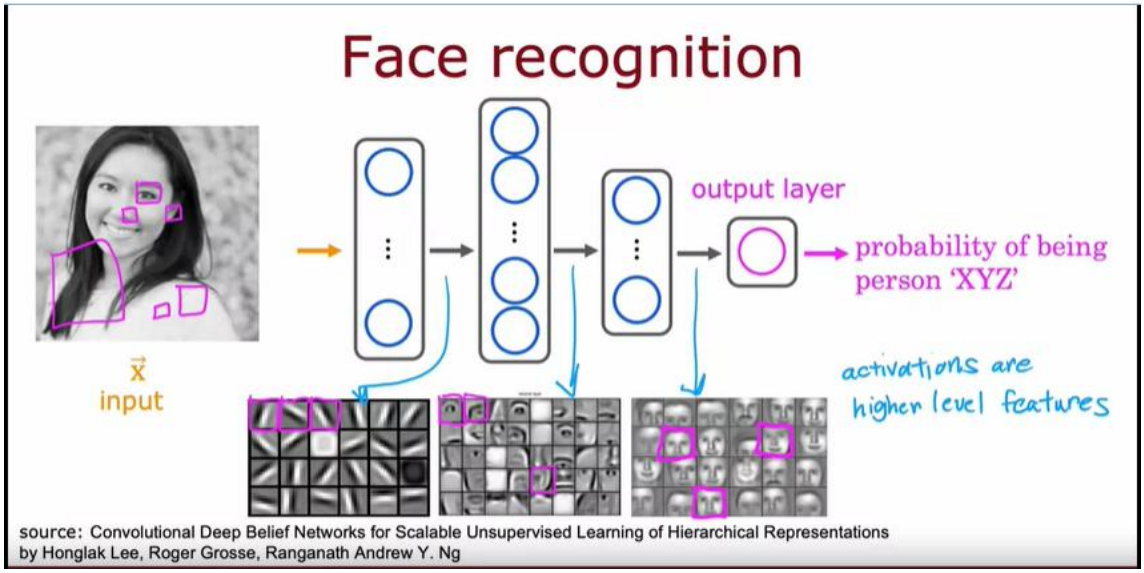


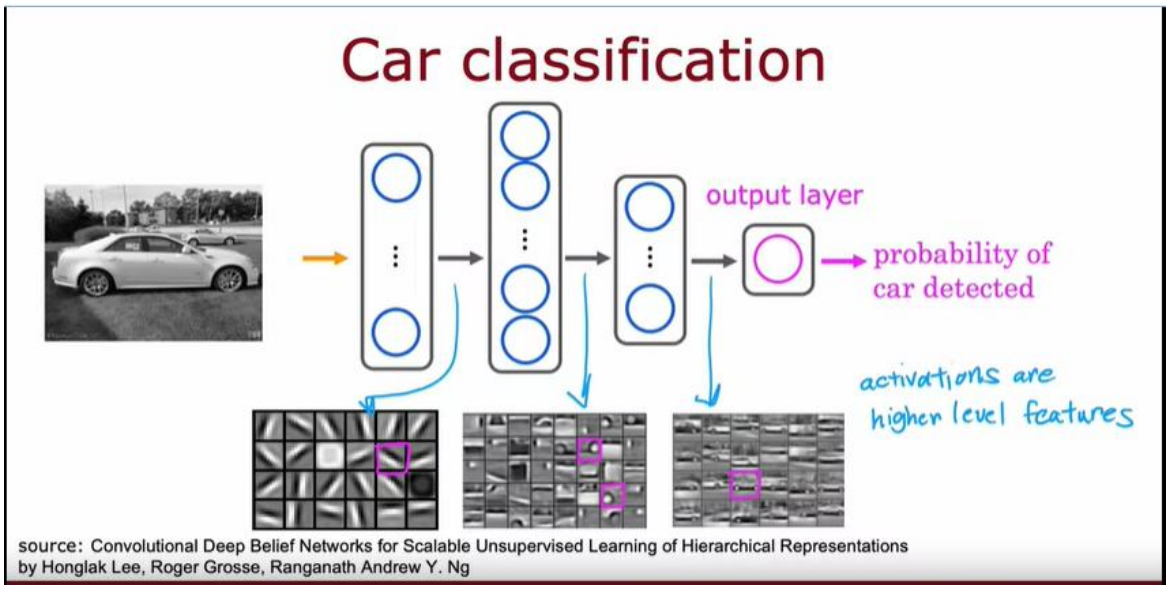




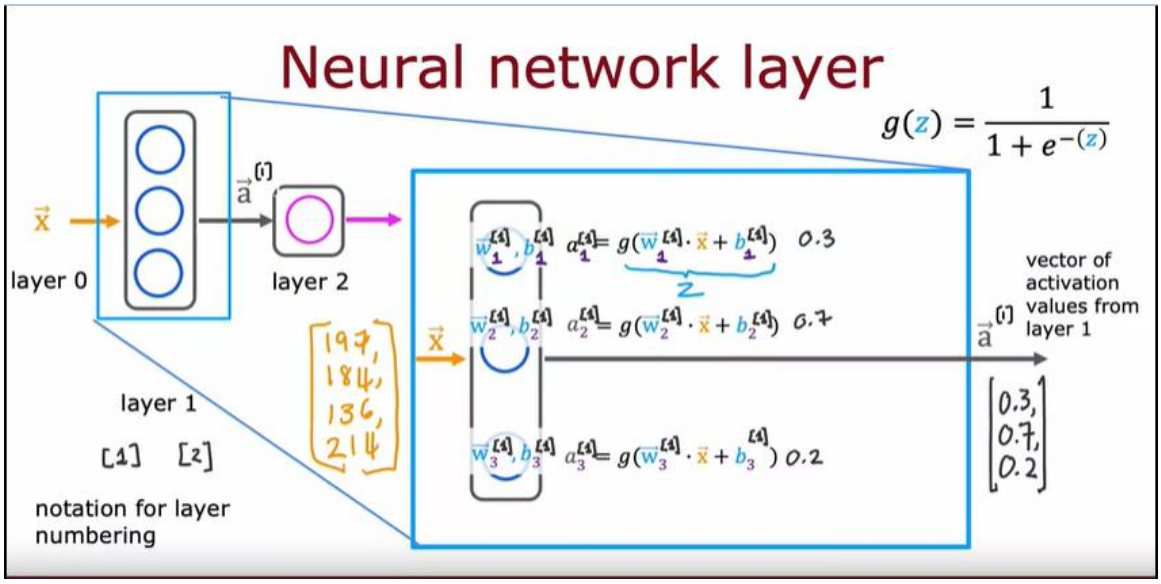
2.1 - Neural networks intuition : 2.1.4 - Example: Recognizing Images :

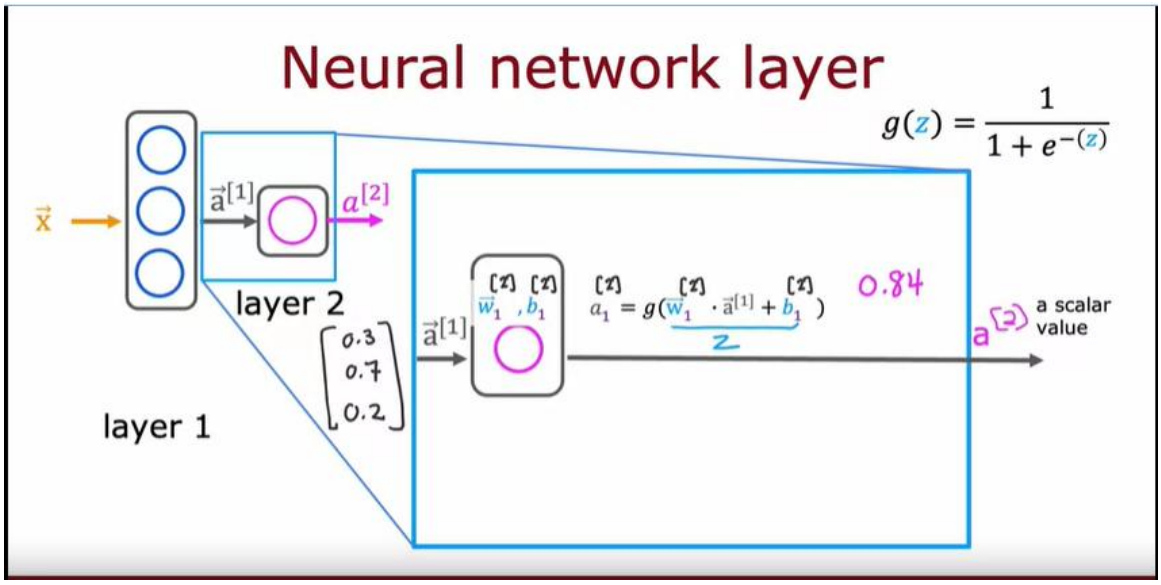
Pixel intensity --> 0 - 255

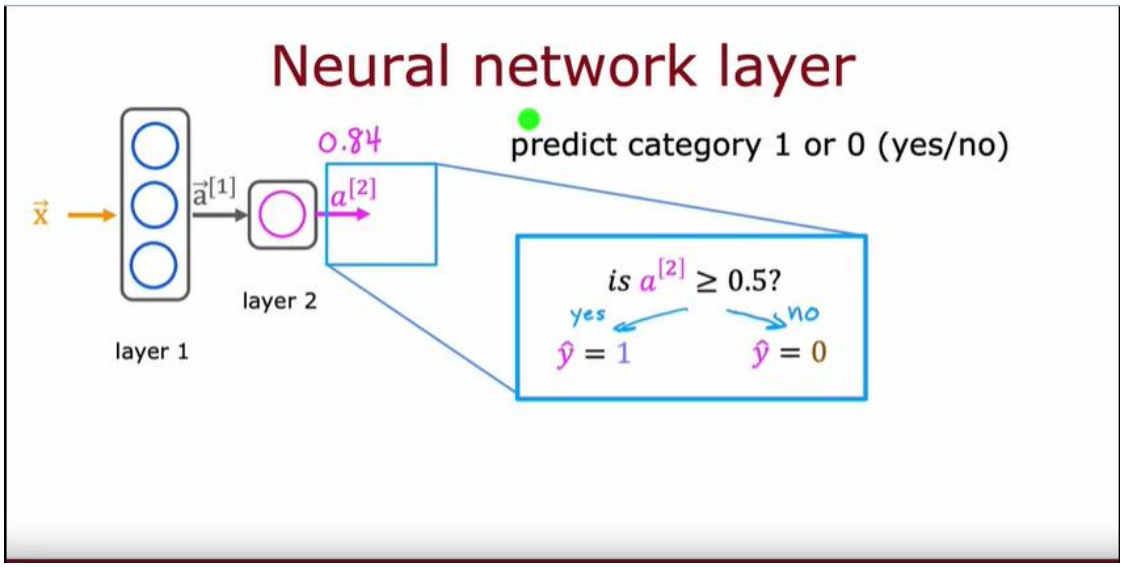




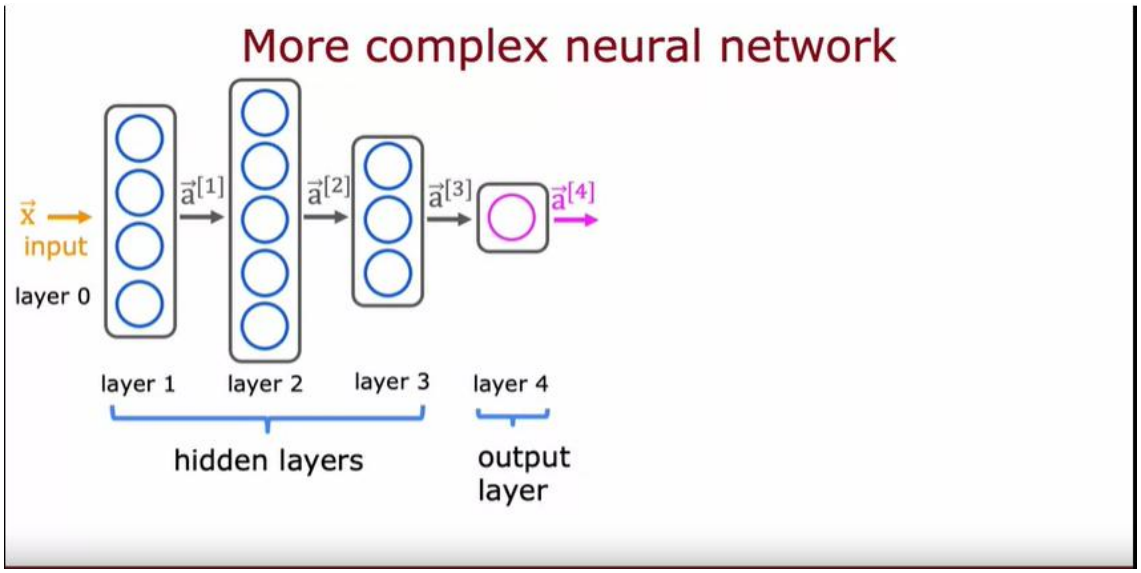
2.2 - Neural network model : 2.2.1 - Neural network layer :

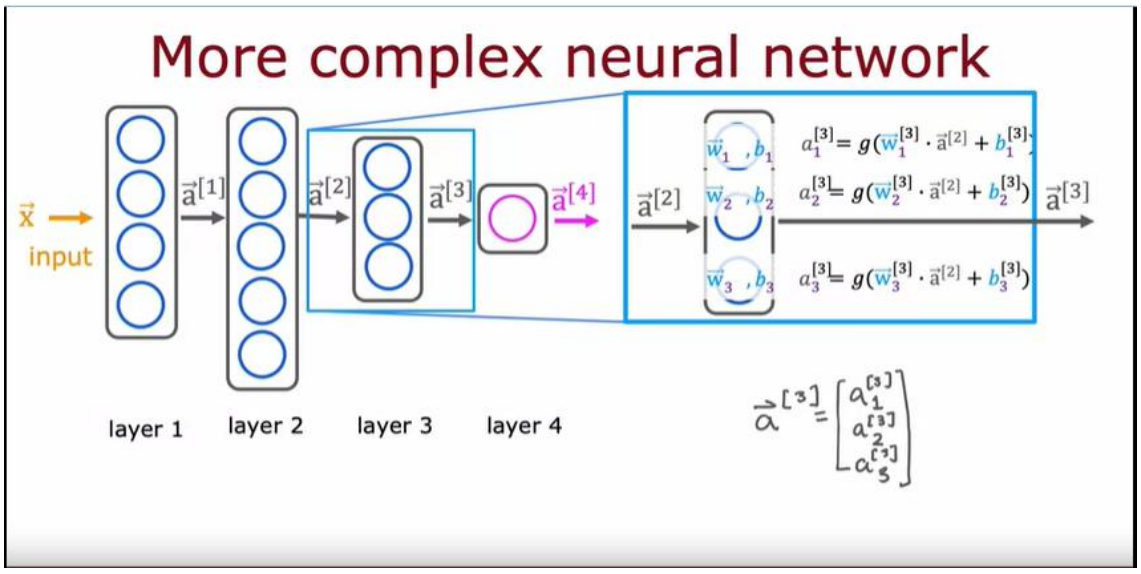


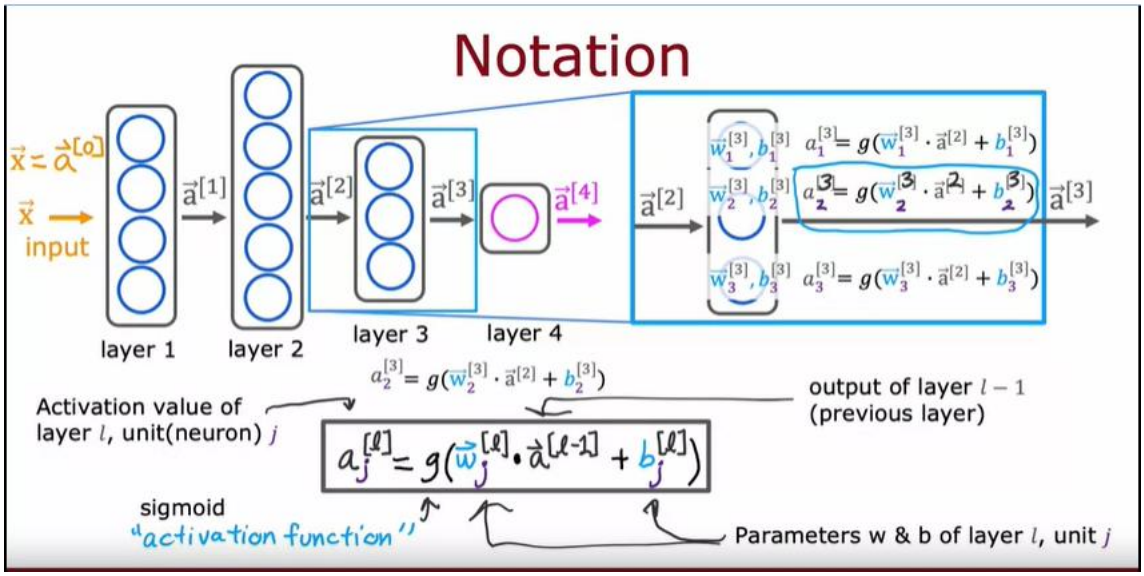




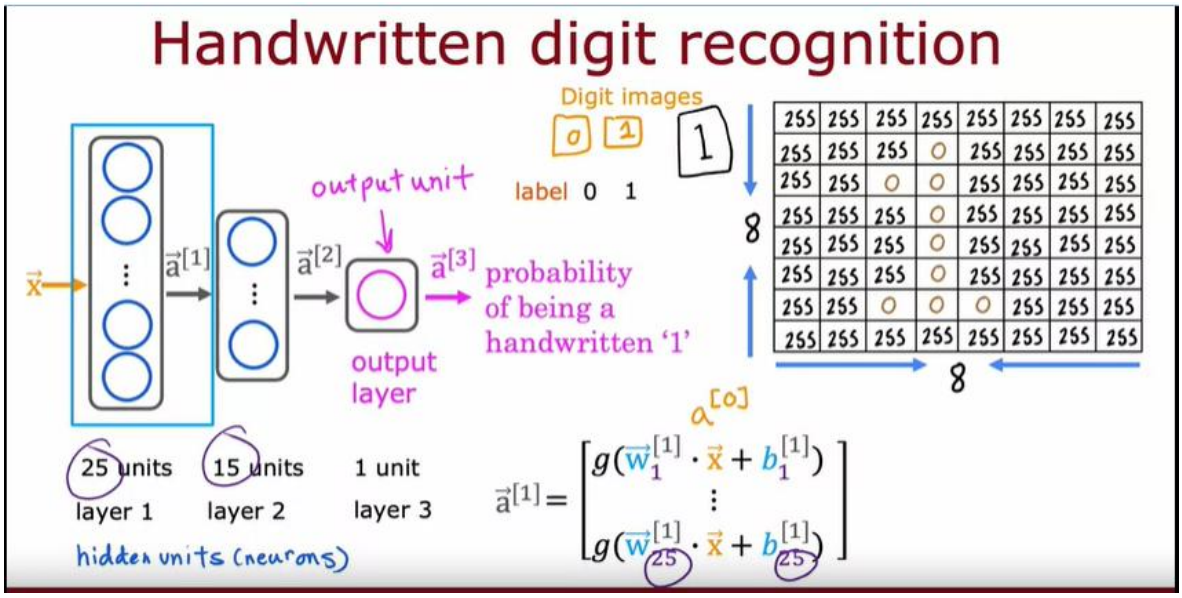
2.2 - Neural network model : 2.2.2 - More complex neural networks :

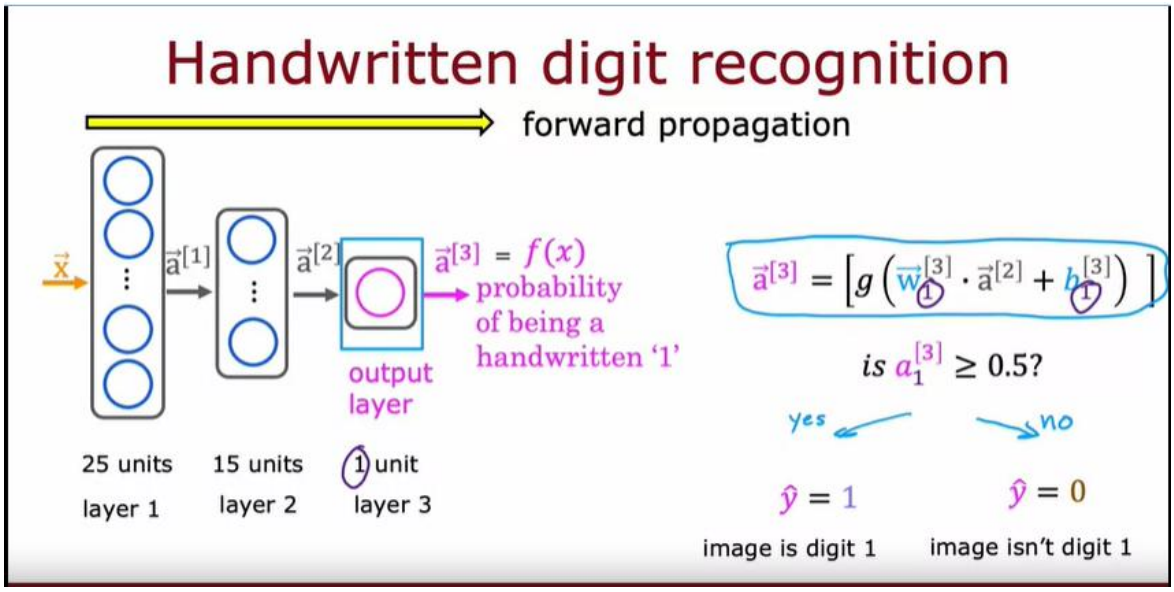






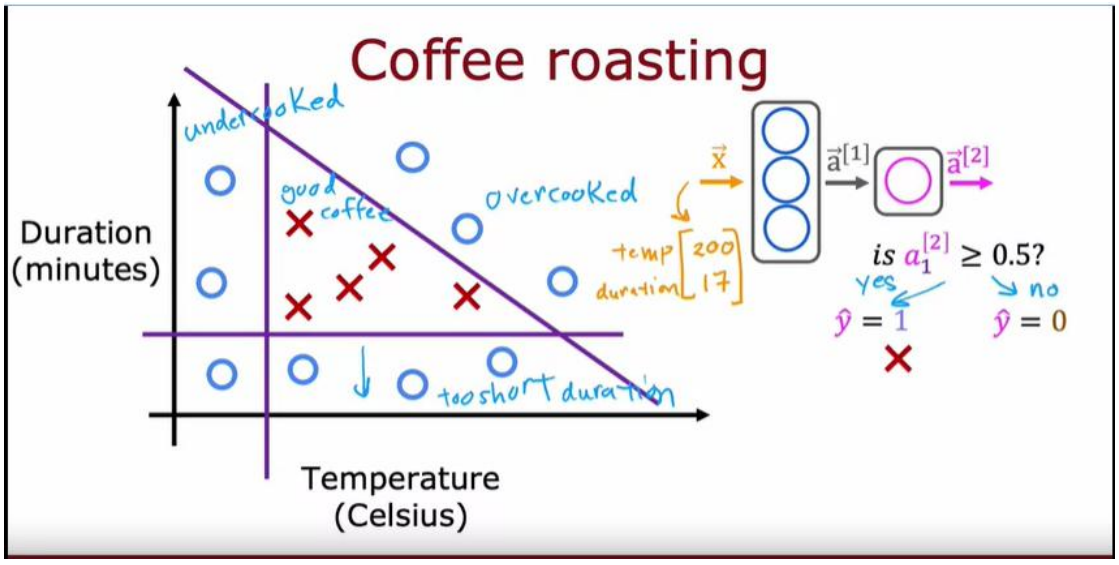
2.2 - Neural network model : 2.2.3 - Inference: making predictions (forward propagation) :

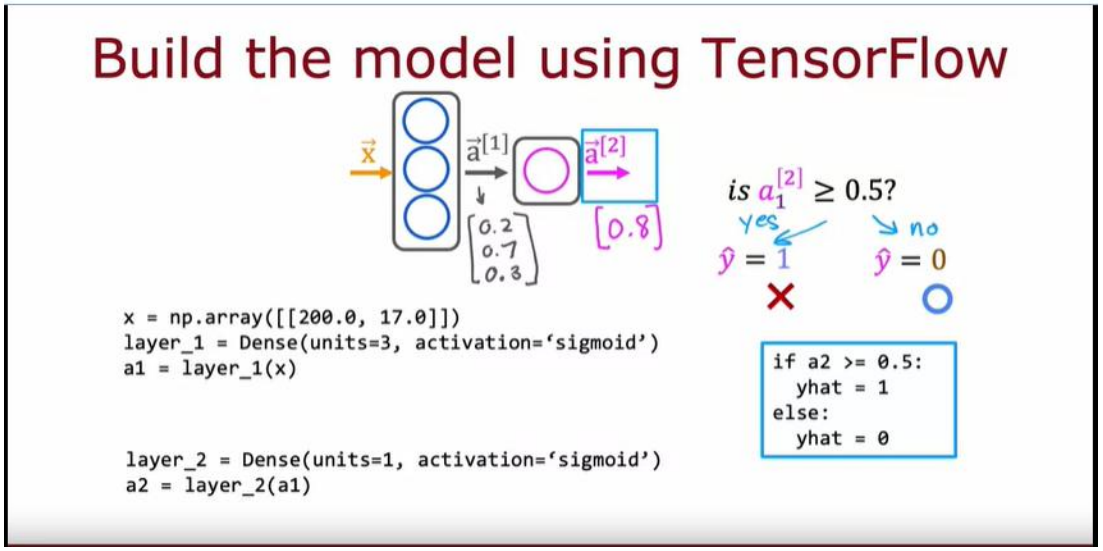


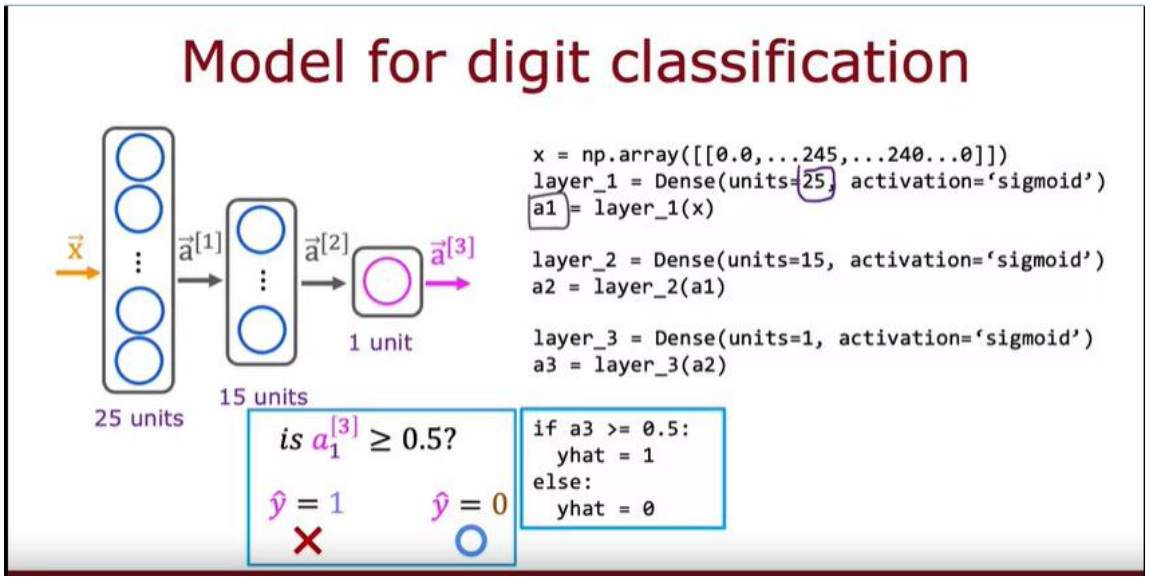


2.3 - TensorFlow implementation : 2.3.1 - Inference in Code :

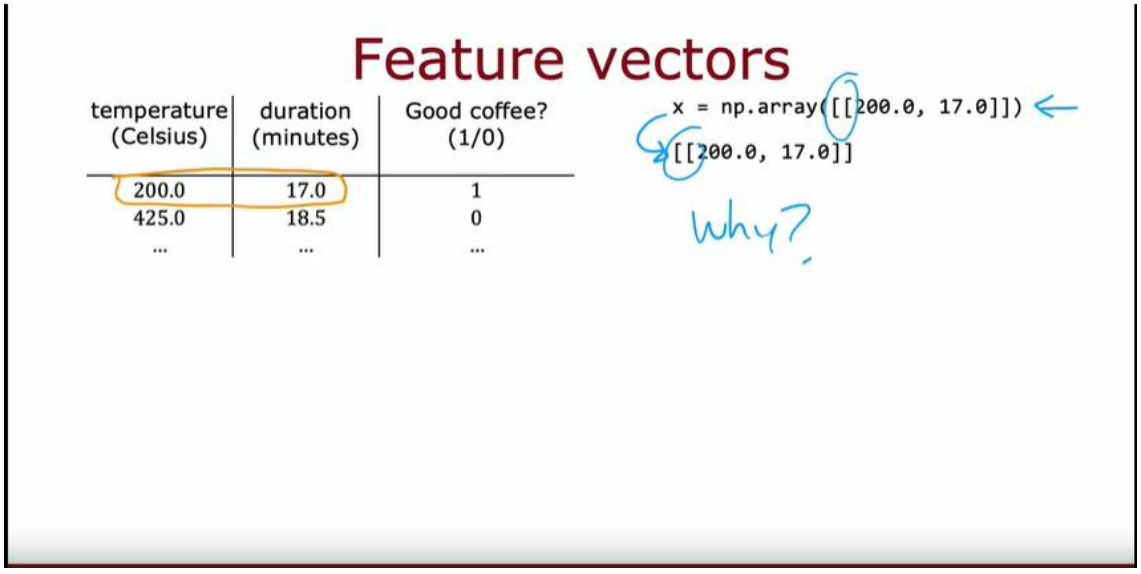
- TensorFlow and PyTorch are the leading frameworks to implementing deep learning algorithms.
- One of the remarkable things about neural networks is the same algorithm can be applied to so many different applications.

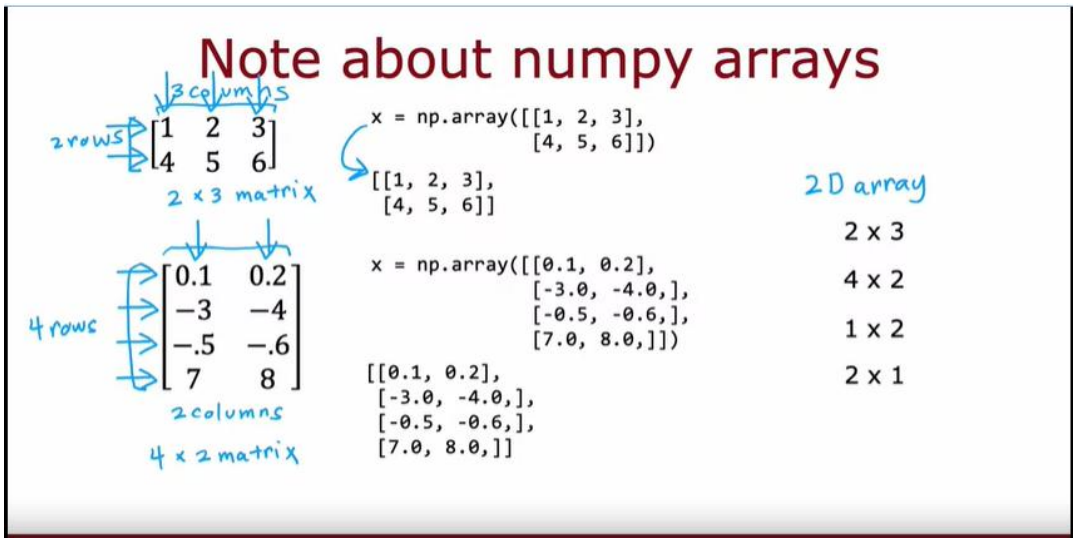


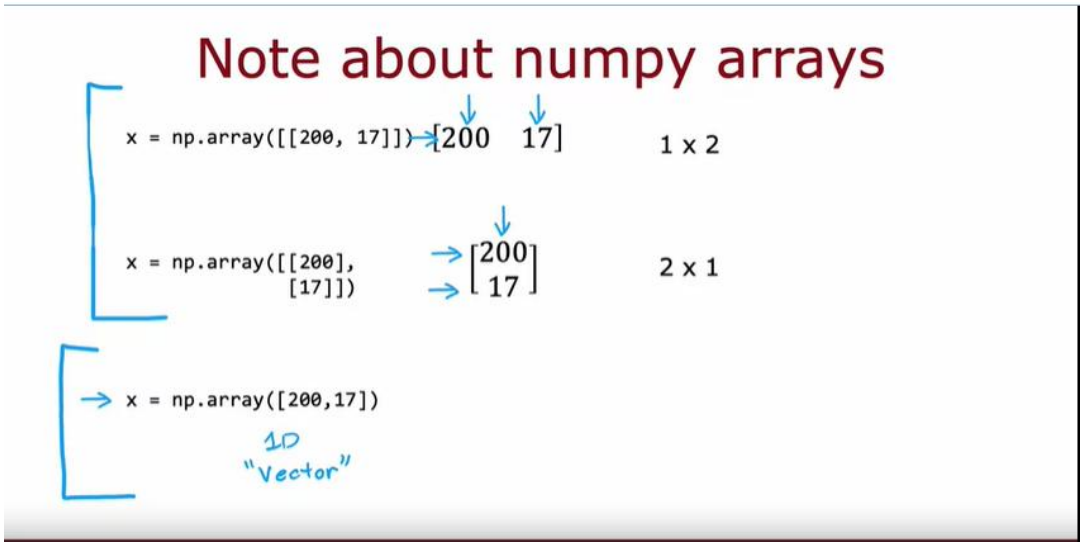




2.3 - TensorFlow implementation : 2.3.2 - Data in TensorFlow :

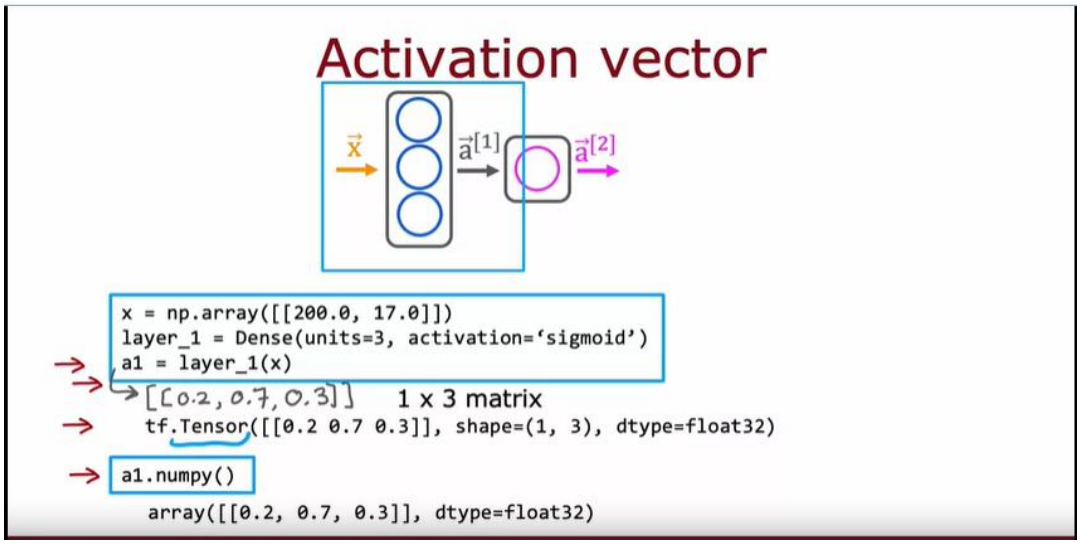


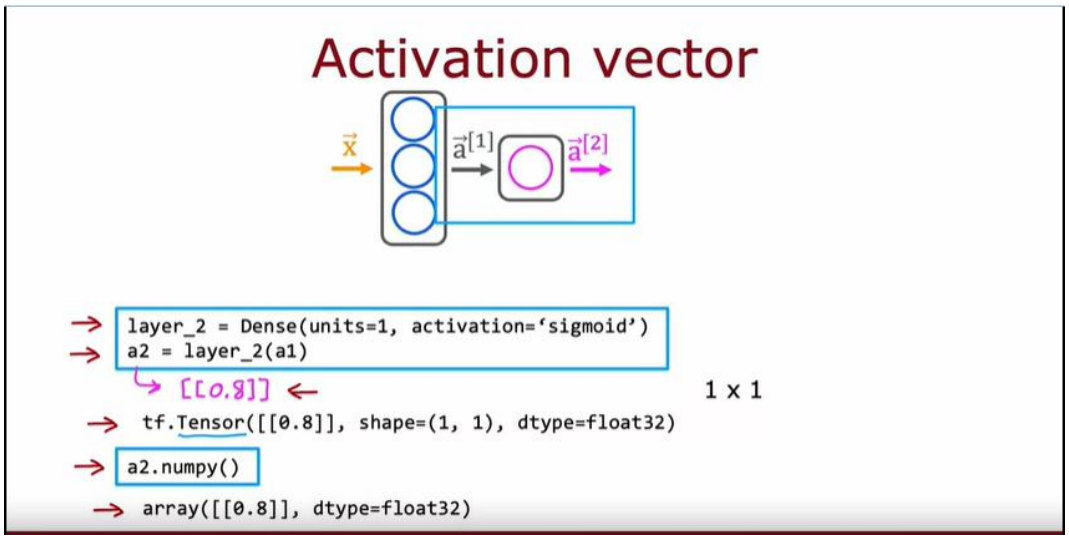




what is the tensor?

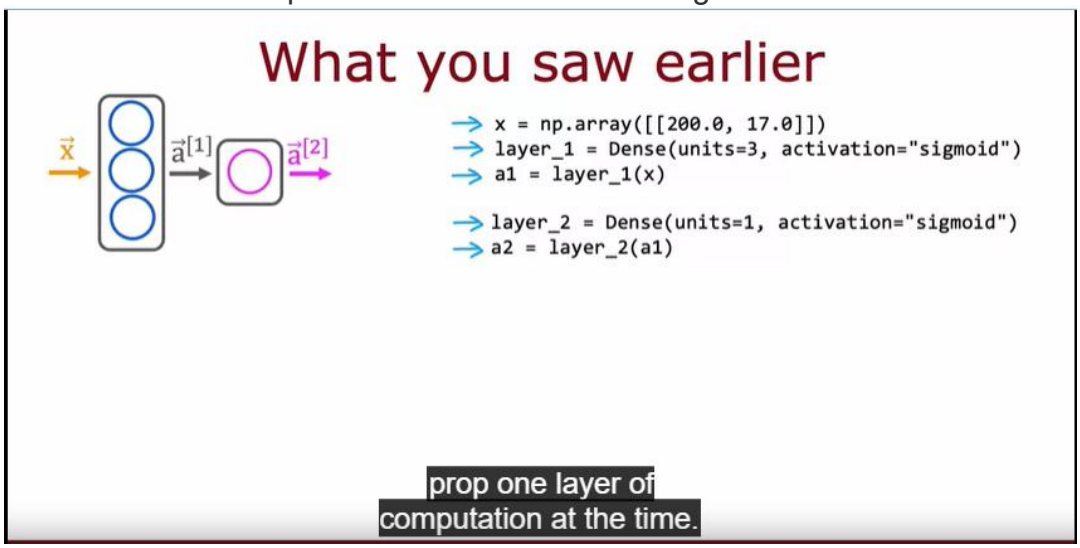
A tensor here is a data type that the TensorFlow team had created in order to store and carry out computations on matrices efficiently.



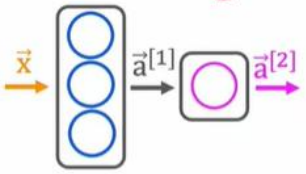


When you pass a NumPy array into TensorFlow, TensorFlow likes to convert it to its own internal format : The tensor and then operate efficiently using tensors. And when you read the data back out you can keep it as a tensor or convert it back to a NumPy array.

2.3 - TensorFlow implementation : 2.3.3 - Building a neural network :



Building a neural network architecture




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

layer1, layer2

		y
200	17	1
120	5	0
425	20	0
212	18	1

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

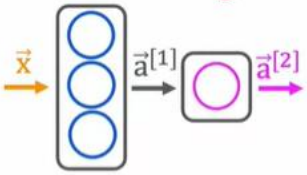
4 x 2

targets

```
y = np.array([1,0,0,1])
model.compile(...)
model.fit(x,y)
model.predict(x_new)
```

← more about this next week!

Building a neural network architecture




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

		y
200	17	1
120	5	0
425	20	0
212	18	1

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

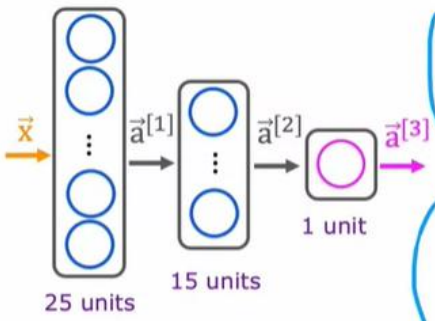
4 x 2

targets

```
y = np.array([1,0,0,1])
model.compile(...)
model.fit(x,y)
model.predict(x_new)
```

← more about this next week!

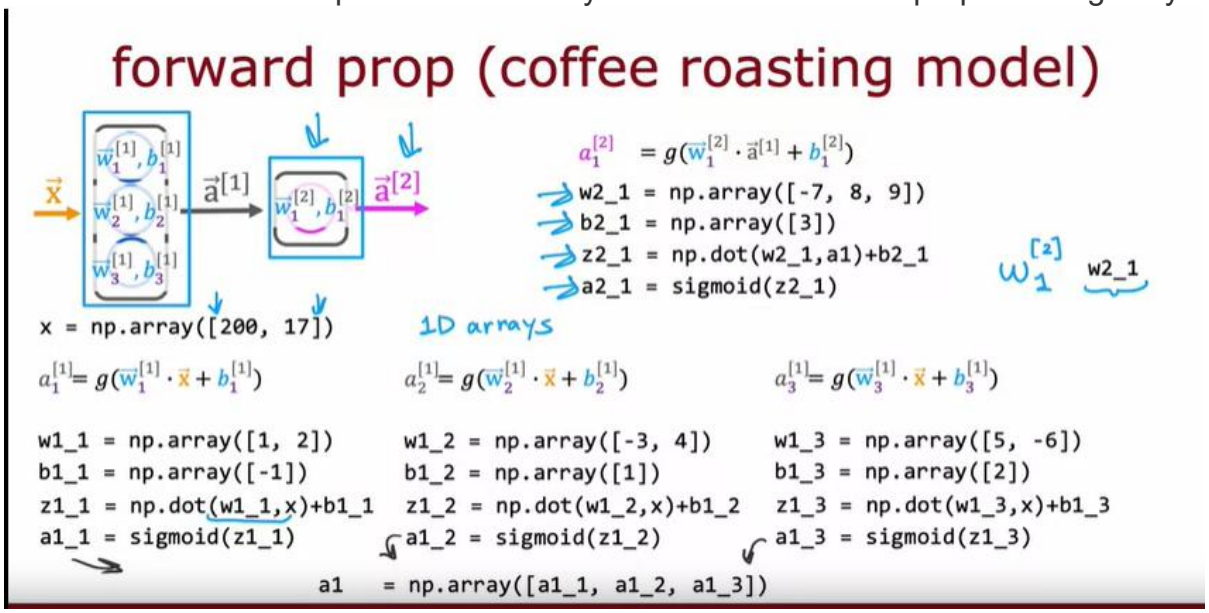
Digit classification model



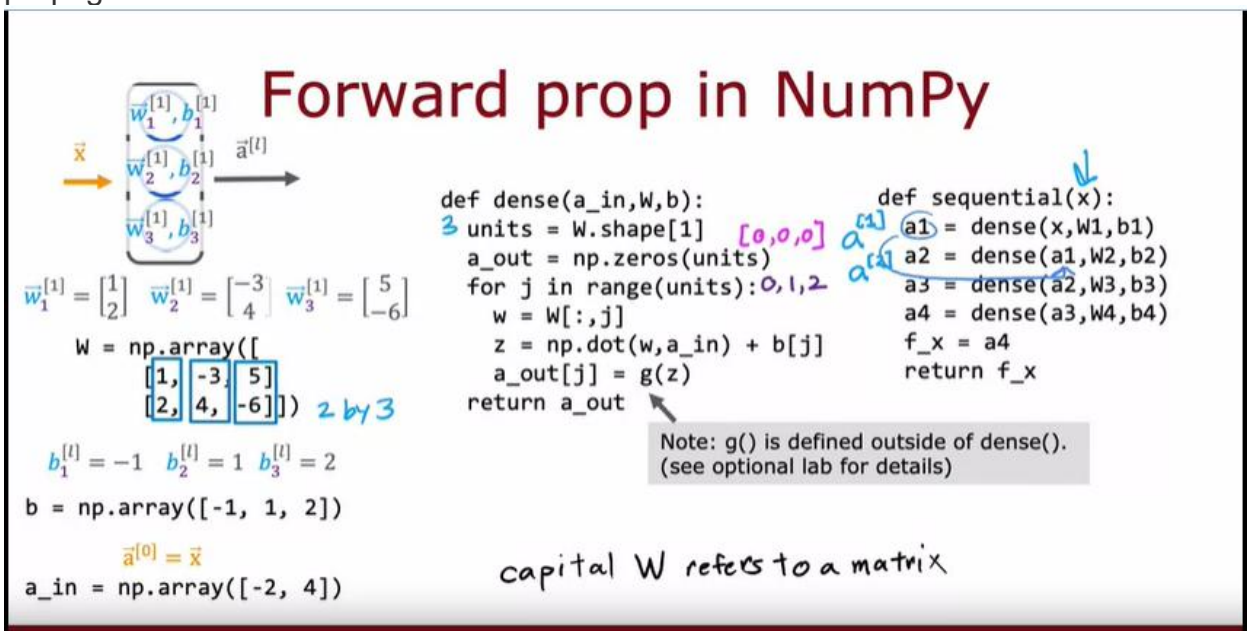

```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])
model.fit(x,y)
model.predict(x_new)
```

← more about this next week!

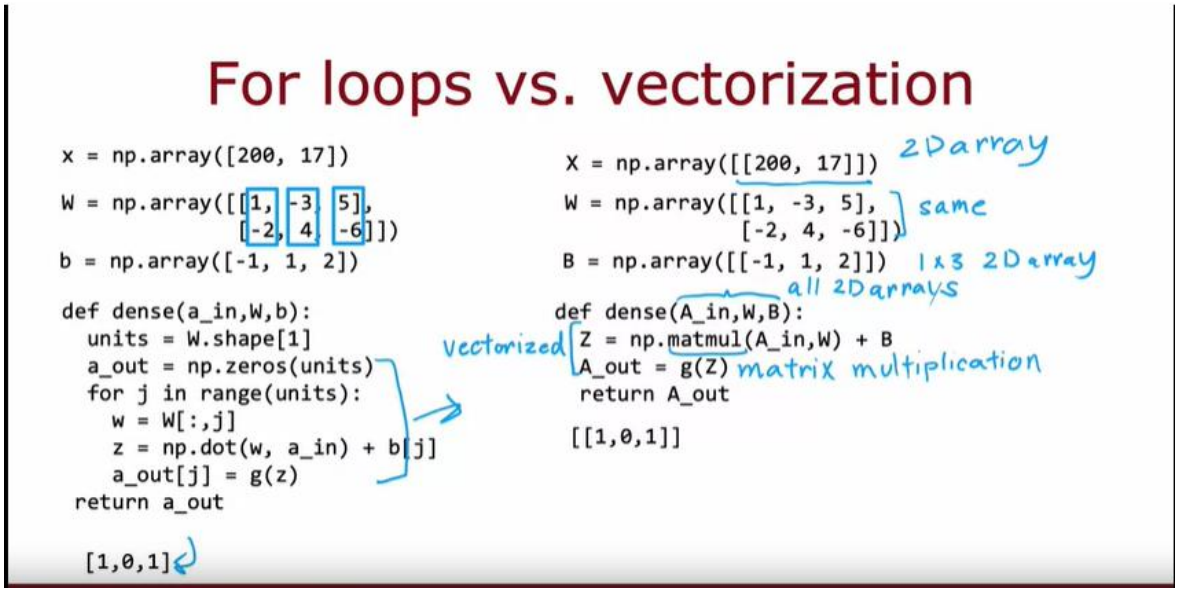
2.4 - Neural network implementation in Python : 2.4.1 - Forward prop in a single layer :



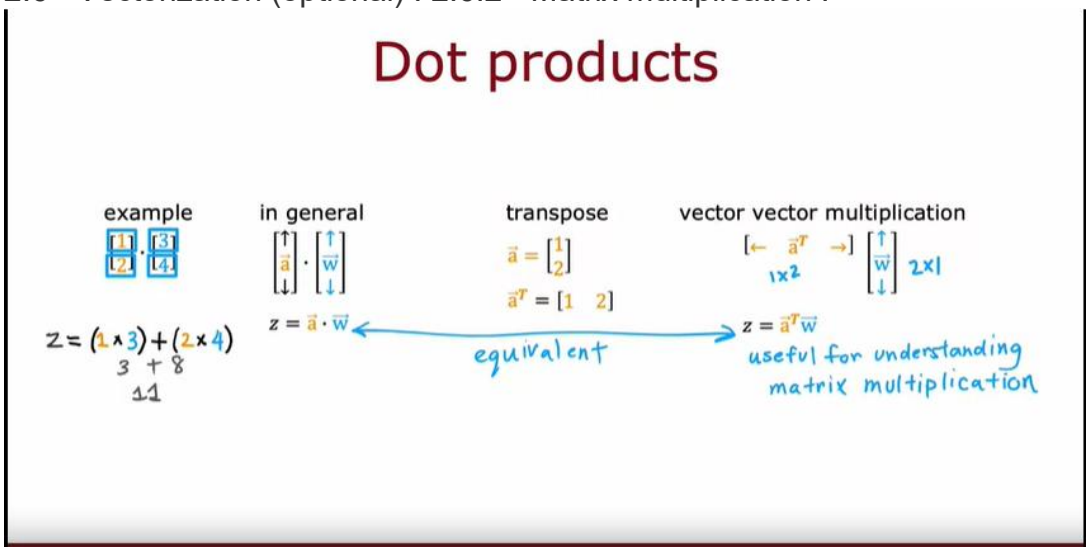
2.4 - Neural network implementation in Python : 2.4.2 - General implementation of forward propagation :

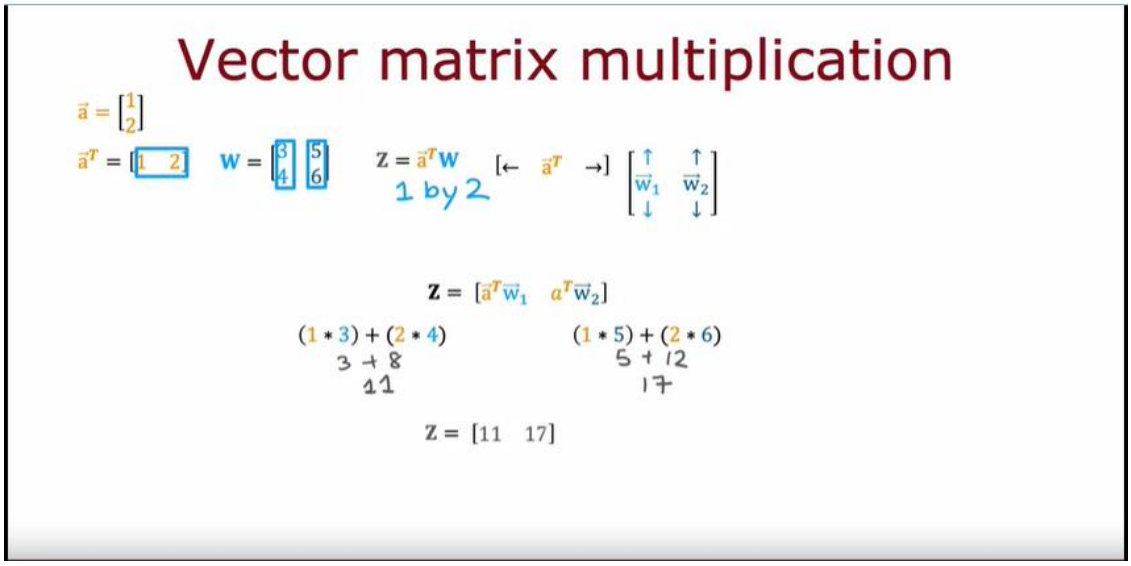


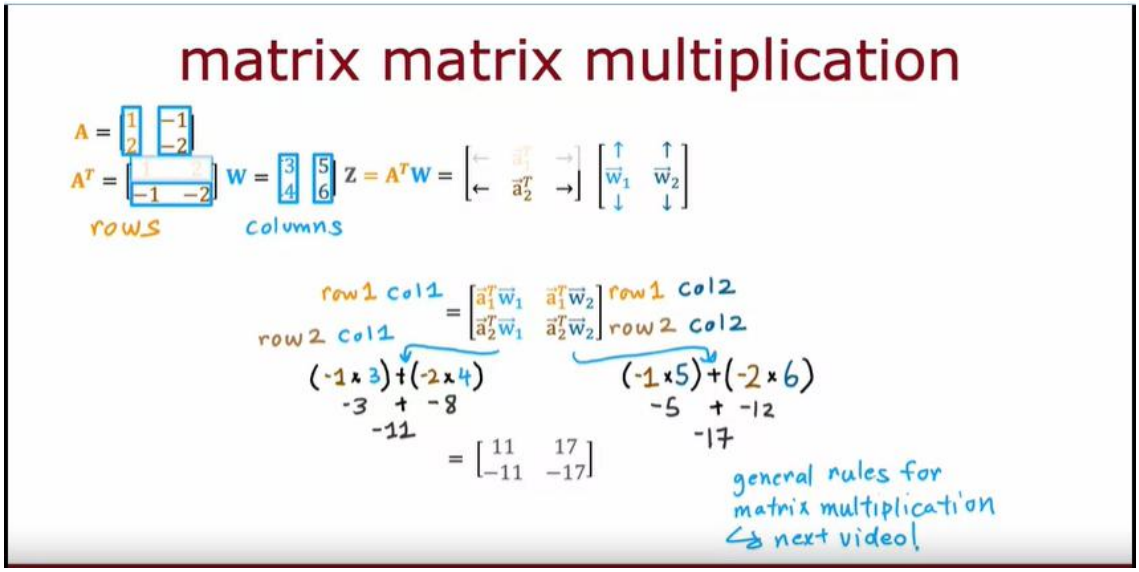
2.6 - Vectorization (optional) : 2.6.1 - How neural networks are implemented efficiently :



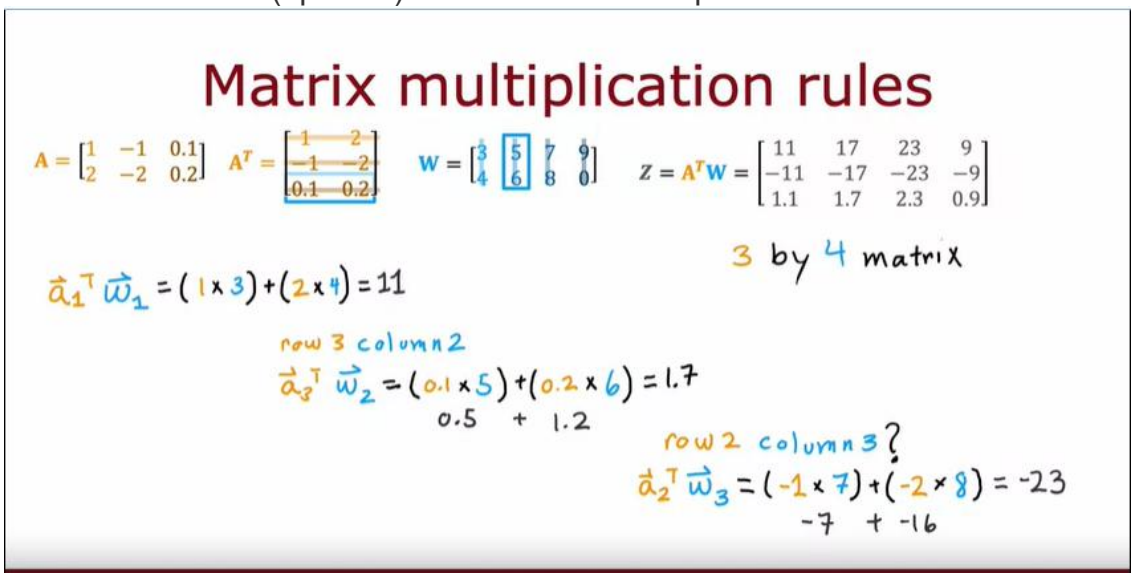
2.6 - Vectorization (optional) : 2.6.2 - Matrix multiplication :

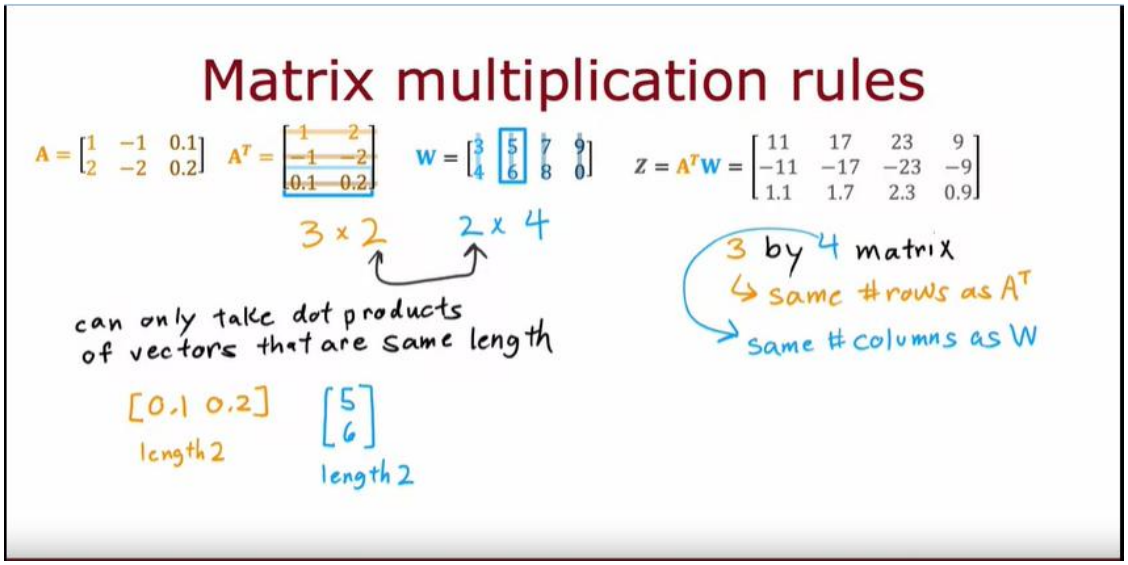




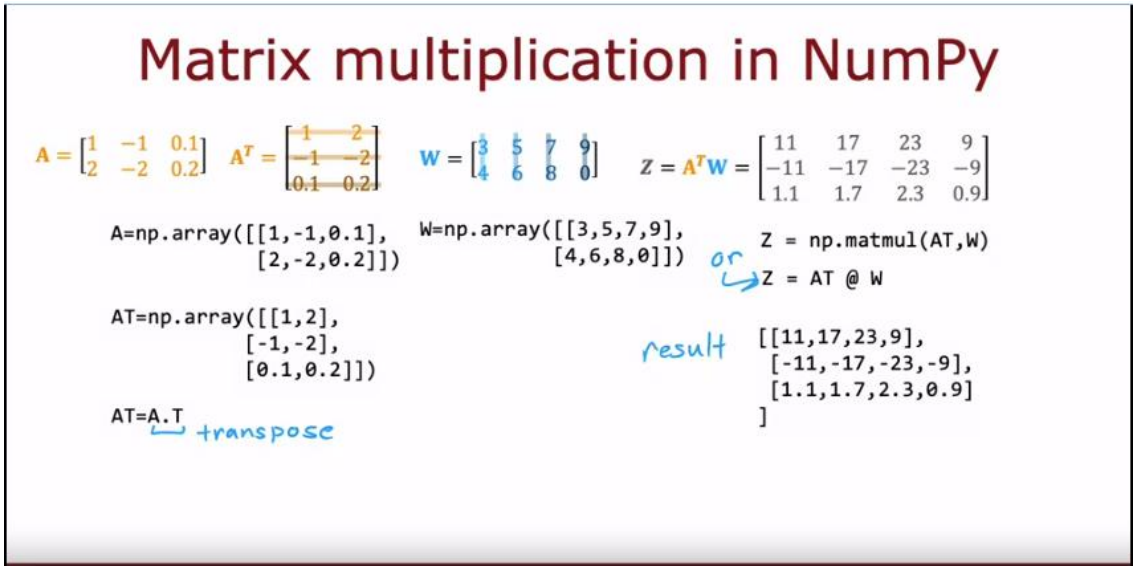


2.6 - Vectorization (optional) : 2.6.3 - Matrix multiplication rules :

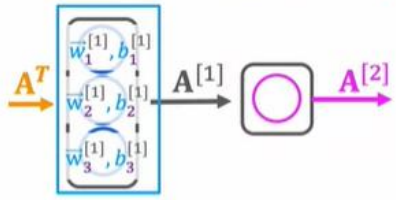




2.6 - Vectorization (optional) : 2.6.4 - Matrix multiplication code :



Dense layer vectorized



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

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

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

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

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

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

$$A = g(Z)$$

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

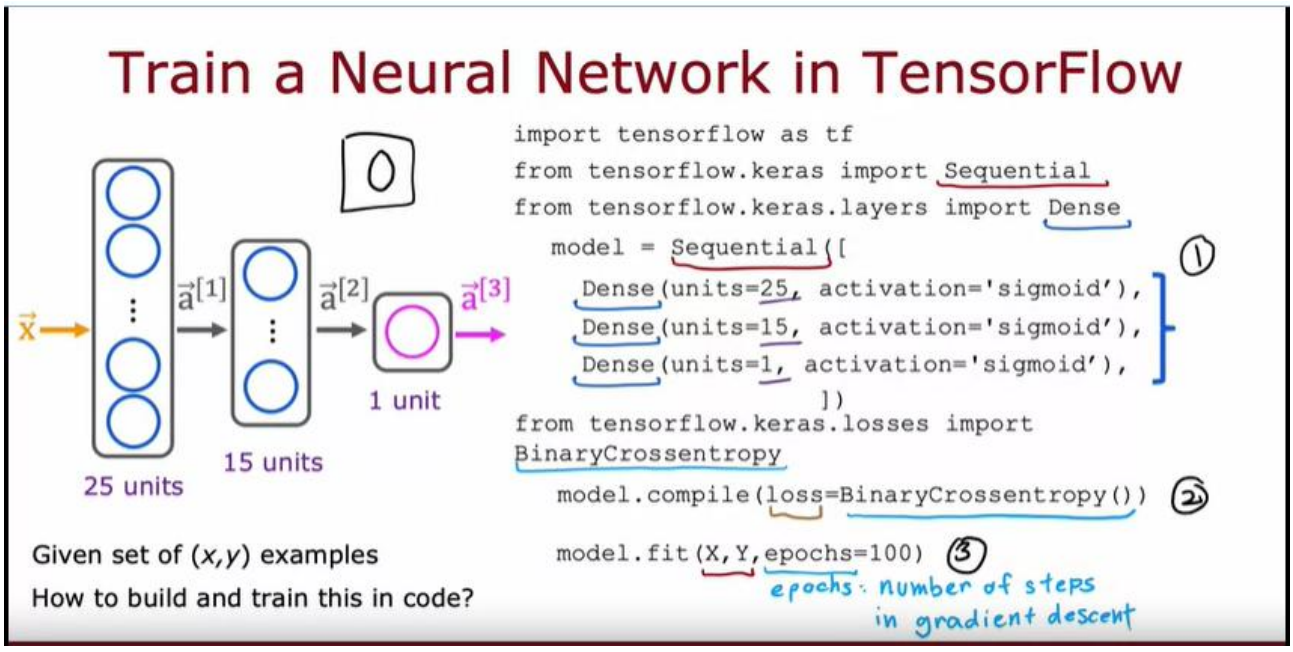
A

```
AT = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([[-1, 1, 2]])
          a_in
def dense(AT,W,b):
  z = np.matmul(AT,W) + b
  a_out = g(z)    a_in
  return a_out
```

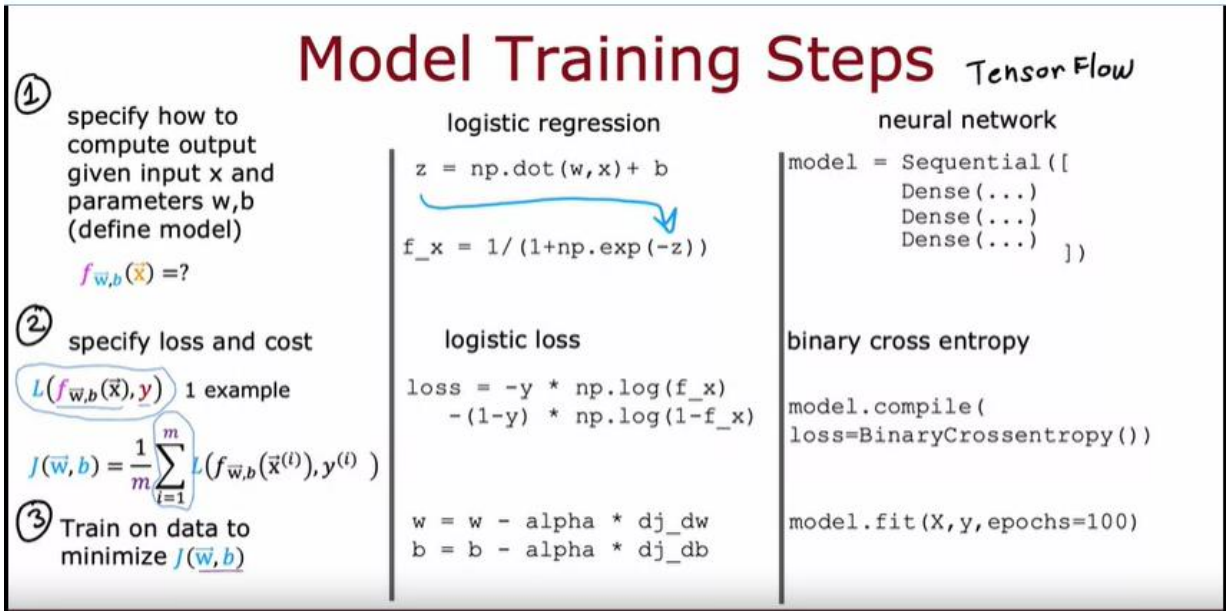
[[1,0,1]]

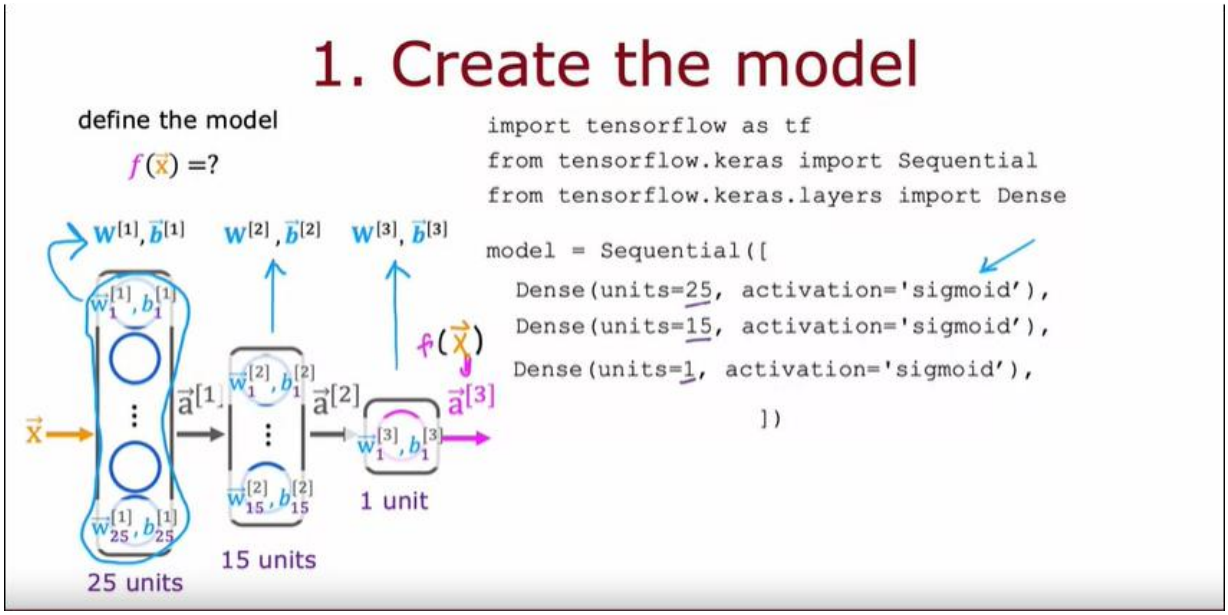
Part (Week) 2 : Neural Network Training

2.1 - Neural Network Training : 2.1.1 - TensorFlow implementation :

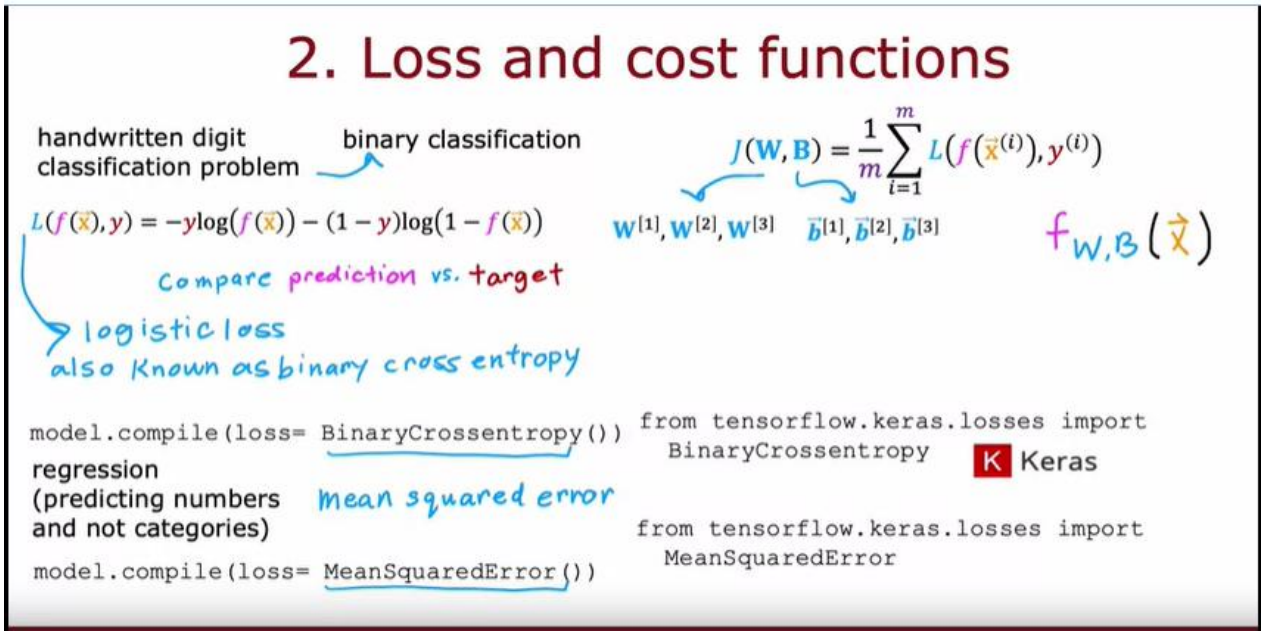


2.1 - Neural Network Training : 2.1.2 - Training Details. Duration:

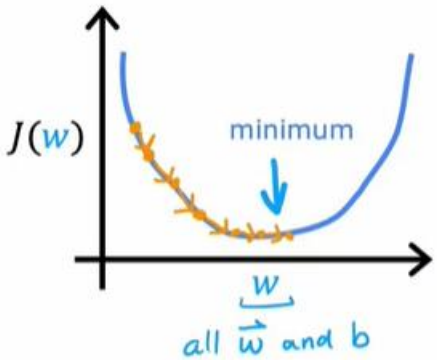




By the way, I don't always remember the names of all the loss functions in the TensorFlow, but I just do a quick web search myself to find the right name and then I plug that into my code.



3. Gradient descent



repeat {

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

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

} Compute derivatives for gradient descent using "back propagation"

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

Neural network libraries

Use code libraries instead of coding "from scratch"

TensorFlow PyTorch

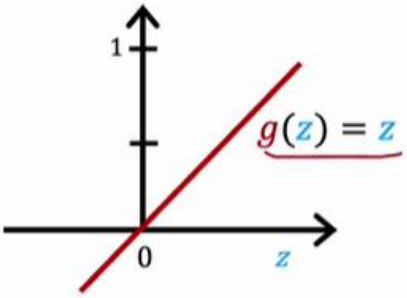
Good to understand the implementation (for tuning and debugging).

2.2 - Activation Functions : 2.2.1 - Alternatives to the sigmoid activation:

Examples of Activation Functions

"No activation function"

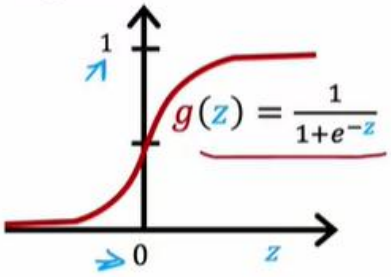
Linear activation function



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

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

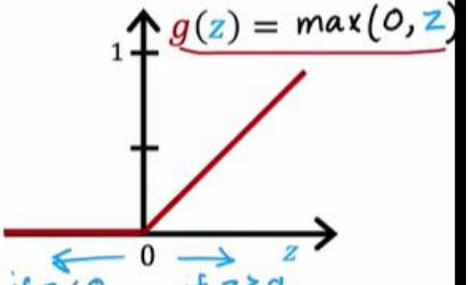
Sigmoid



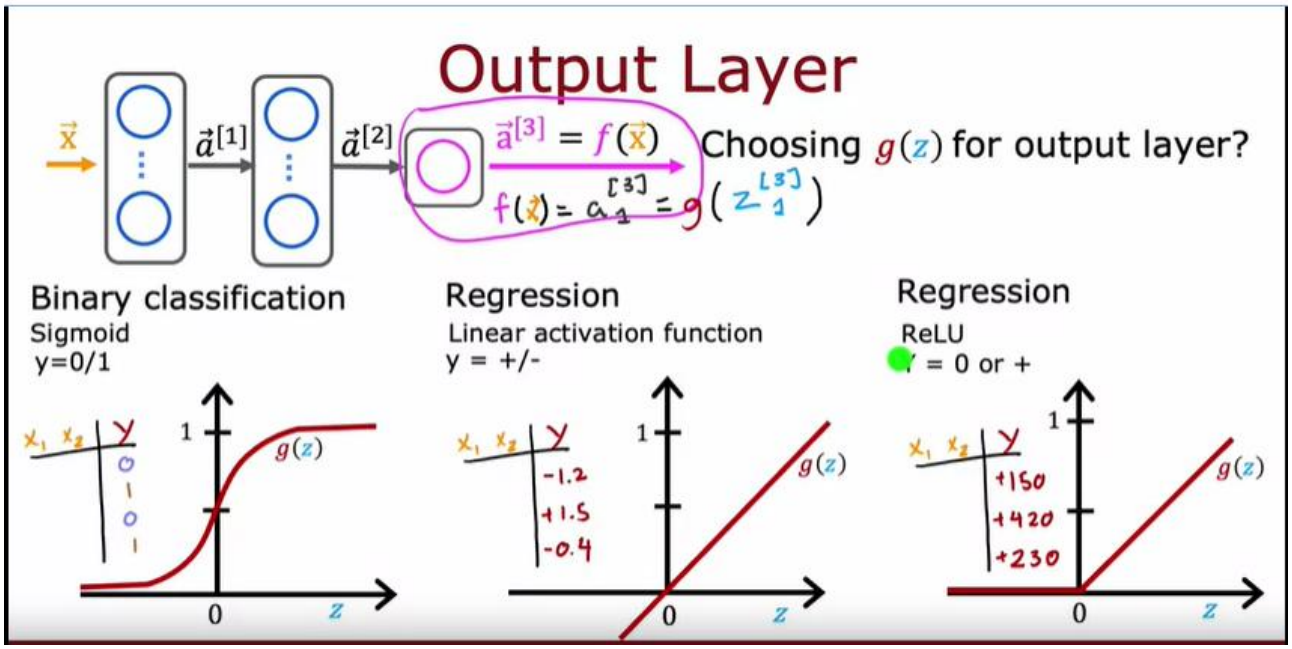
$0 < g(z) < 1$

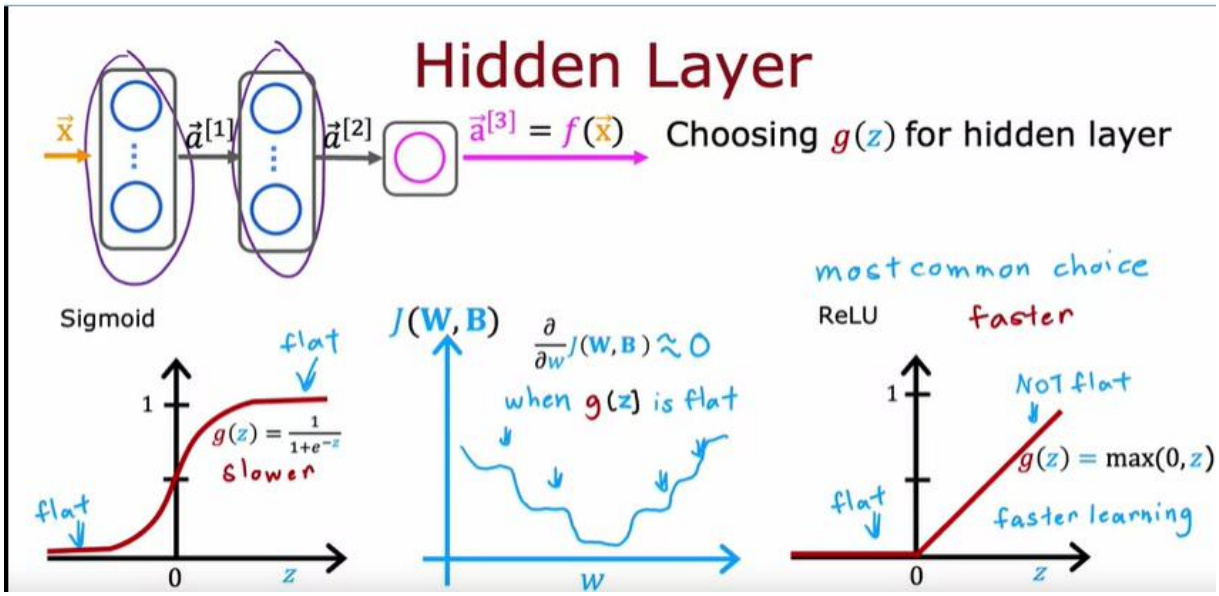
Later: softmax activation

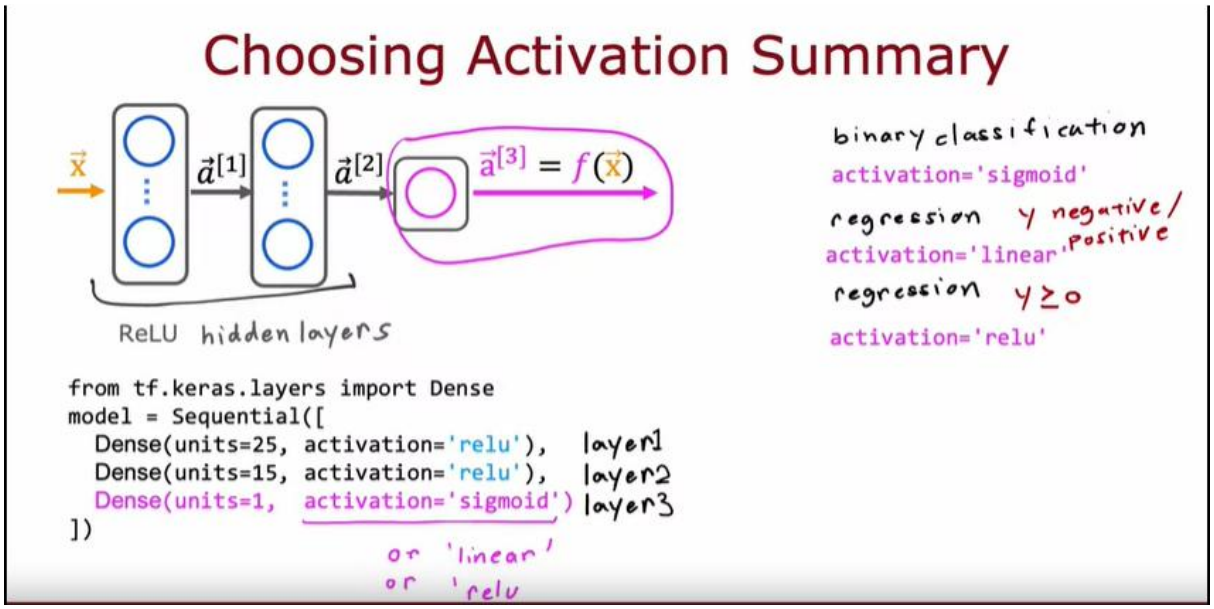
ReLU Rectified Linear Unit



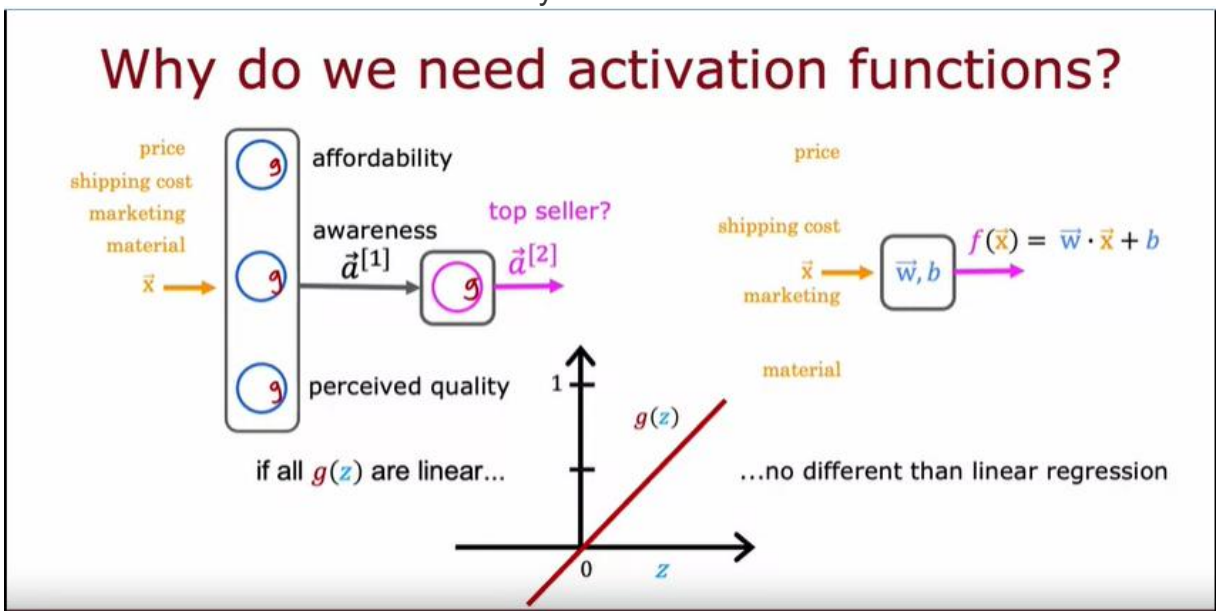
2.2 - Activation Functions : 2.2.2 - Choosing activation functions:

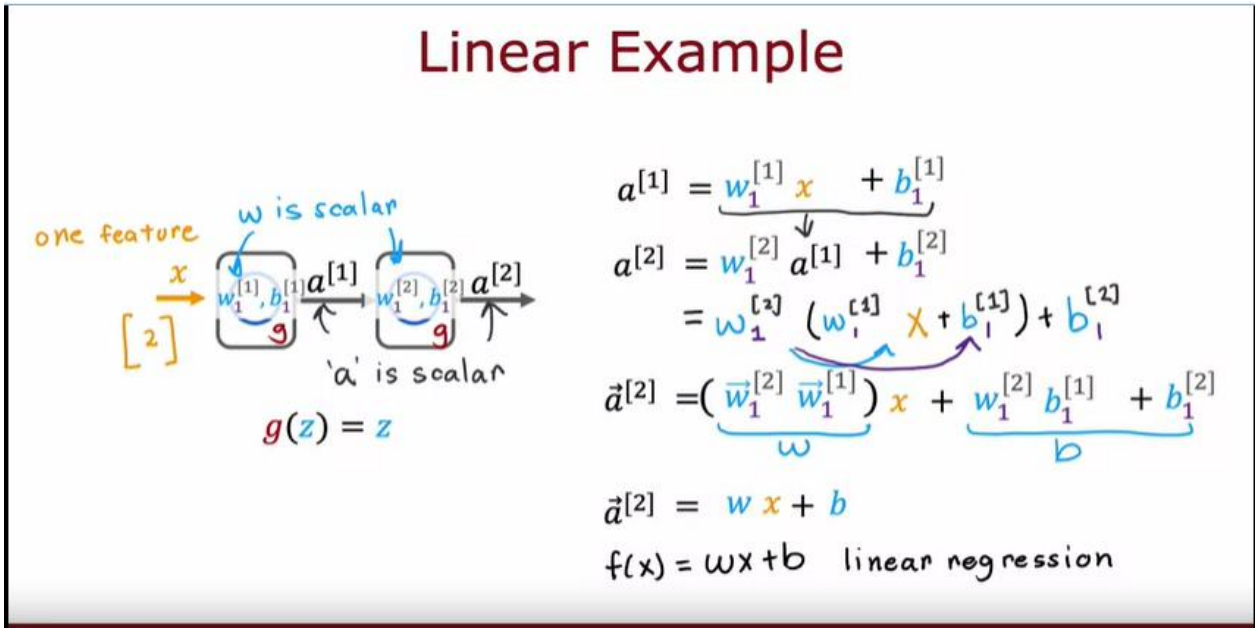


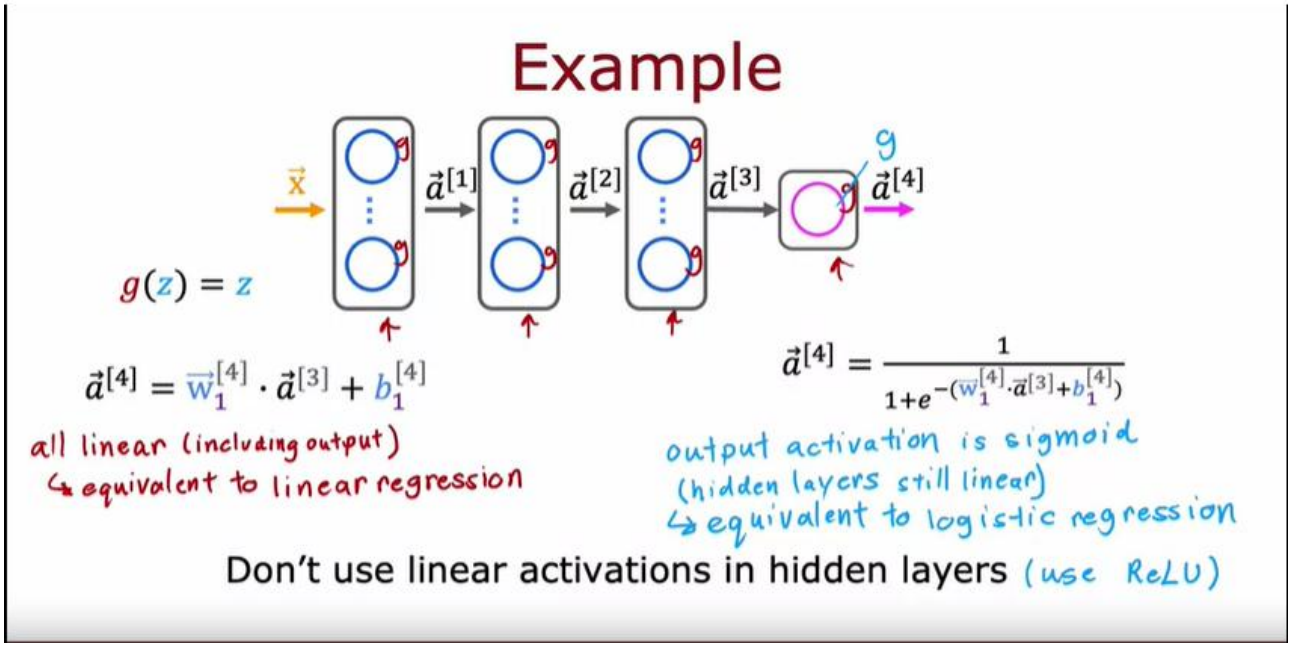




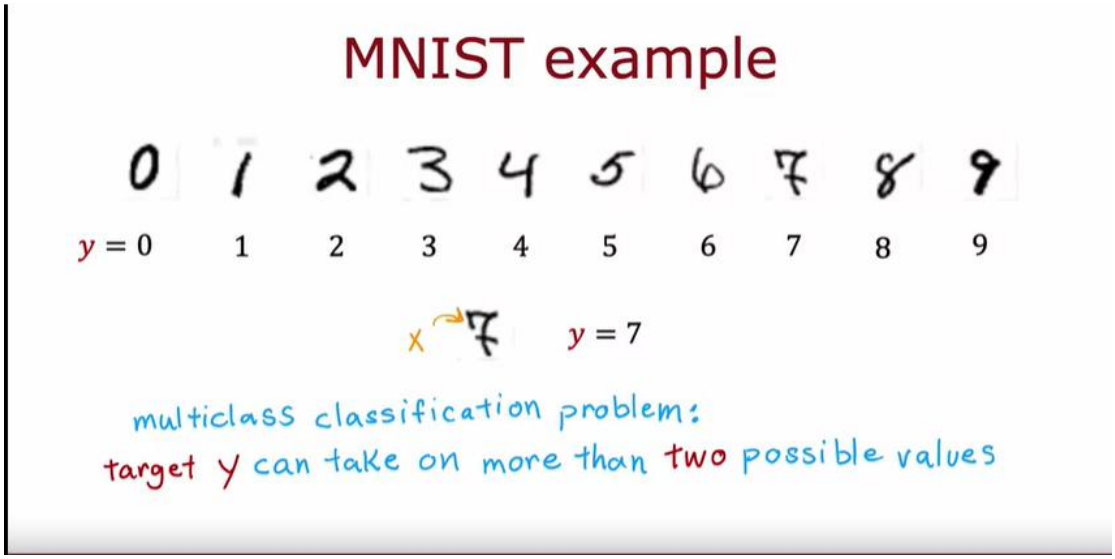
2.2 - Activation Functions : 2.2.3 - Why do we need activation functions? :



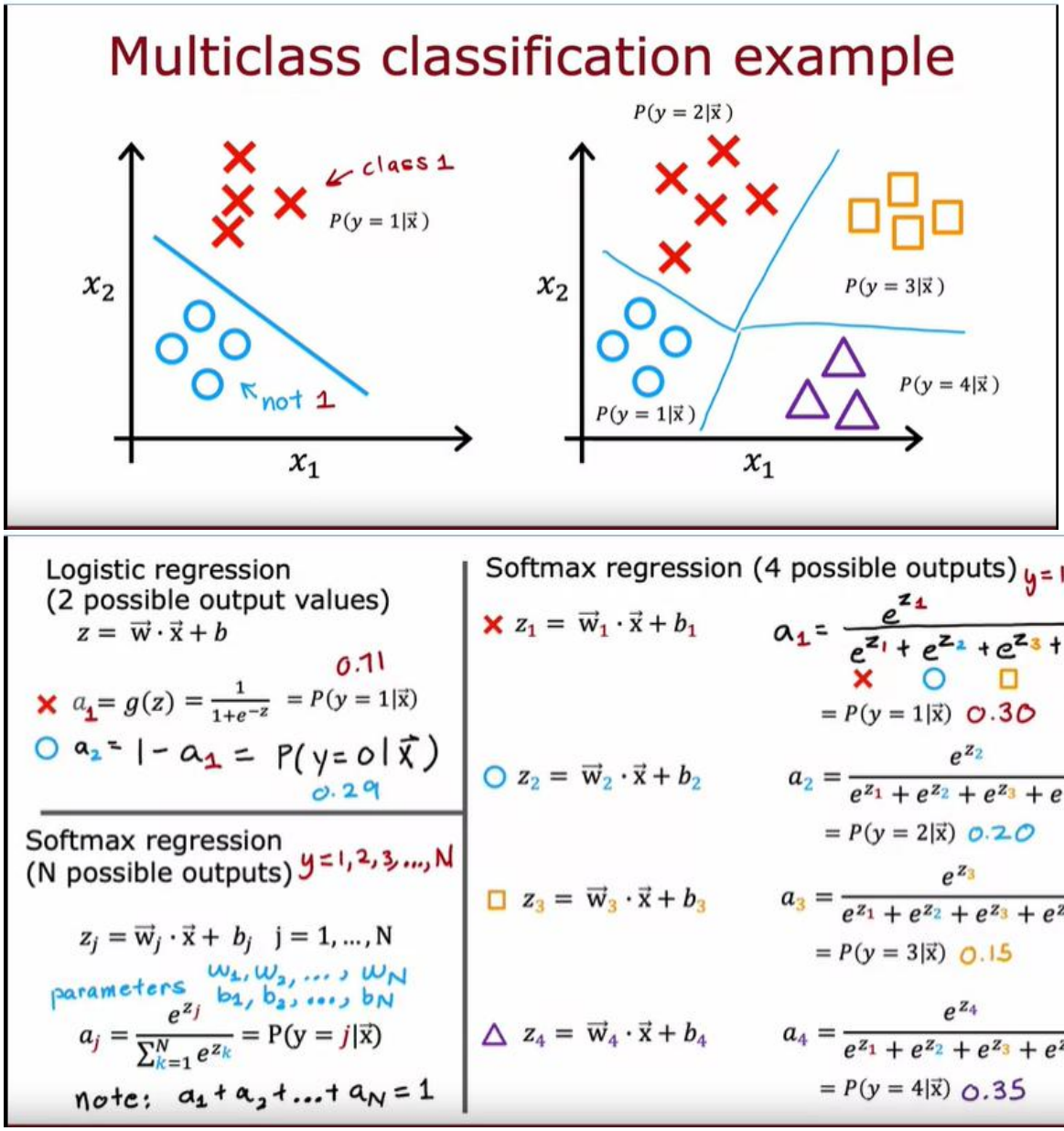


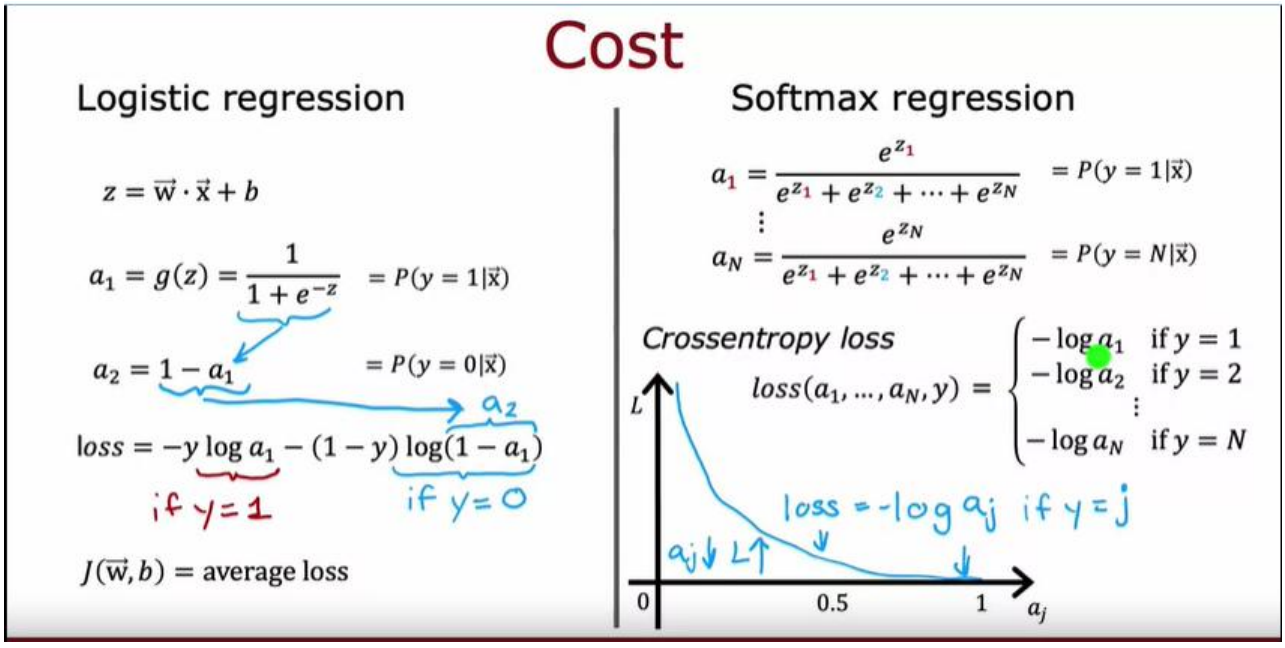


2.3 - Multiclass Classification : 2.3.1 - Multiclass :

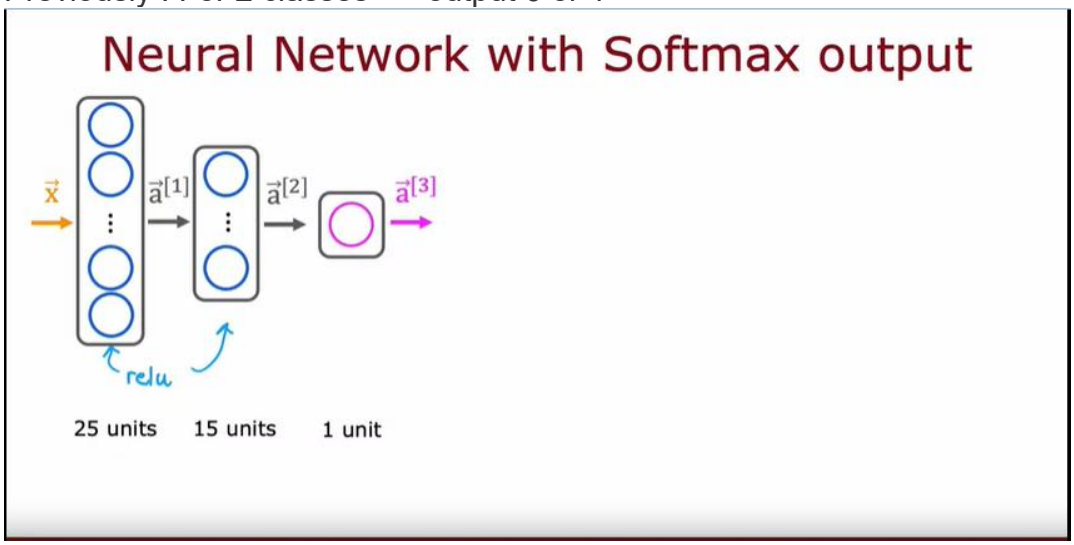


2.3 - Multiclass Classification : 2.3.2 - Softmax :

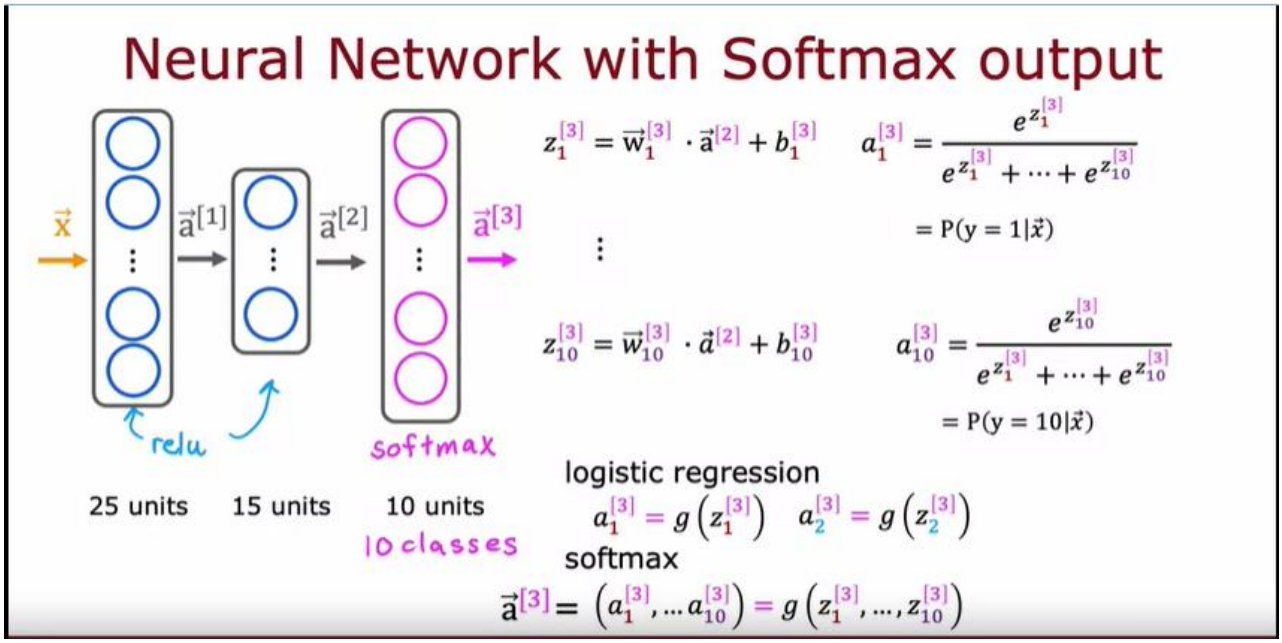




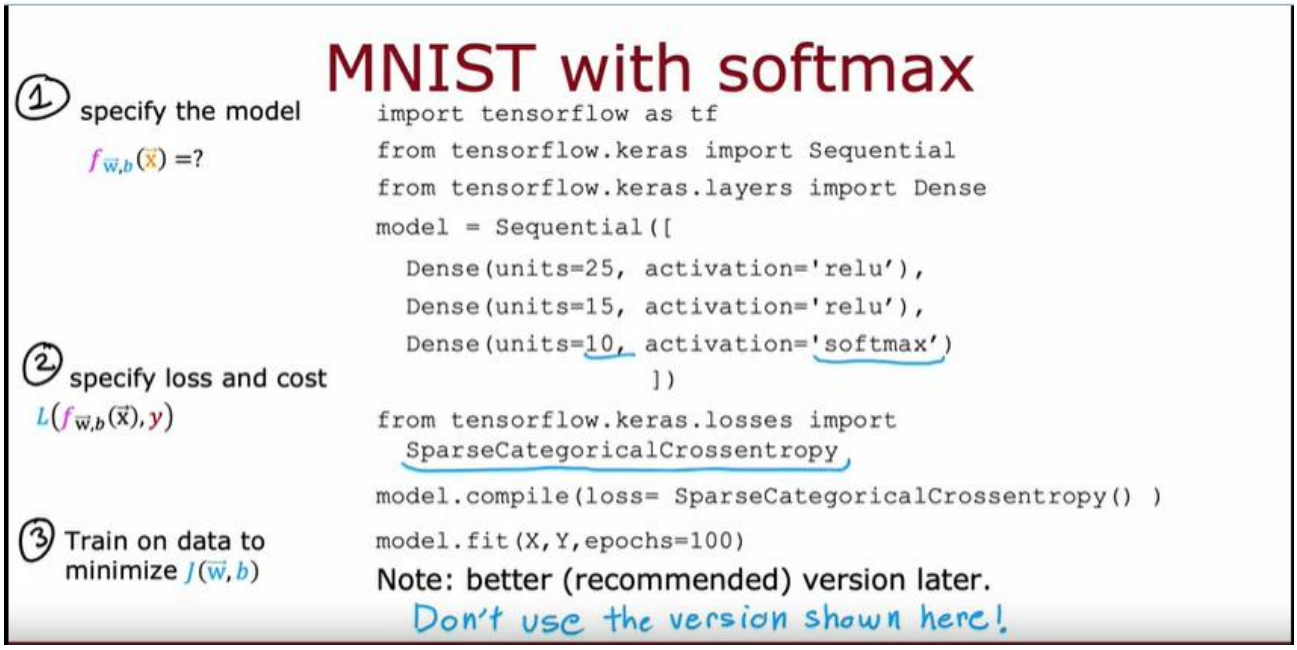
2.3 - Multiclass Classification : 2.3.3 - Neural Network with Softmax output :
Previously : For 2 classes --> output 0 or 1



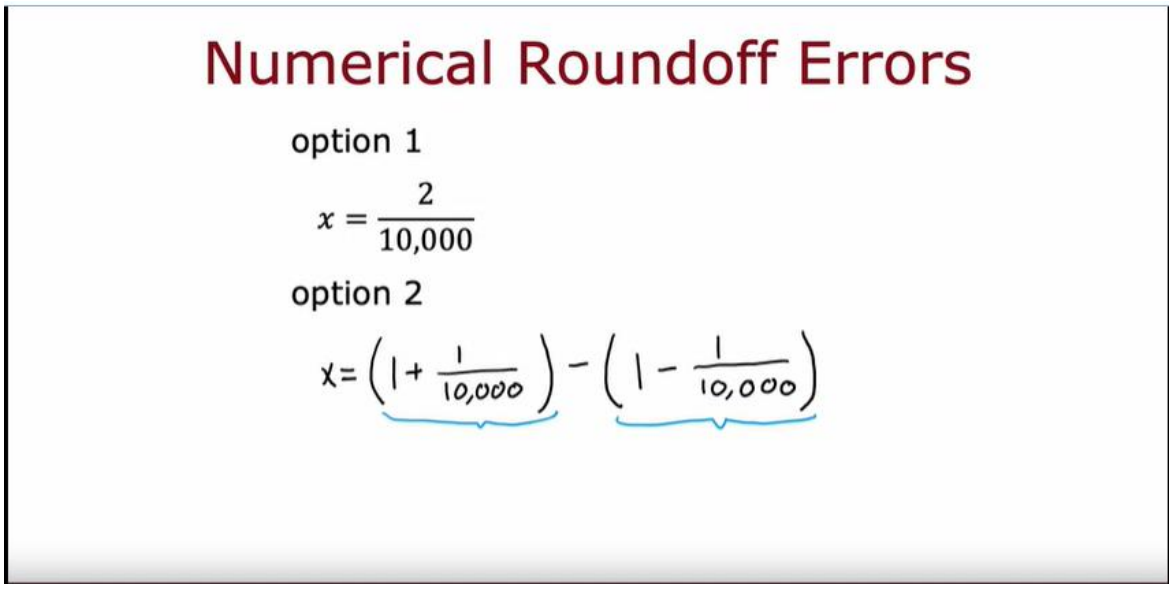
Now : For 10 classes --> output 0 or 9

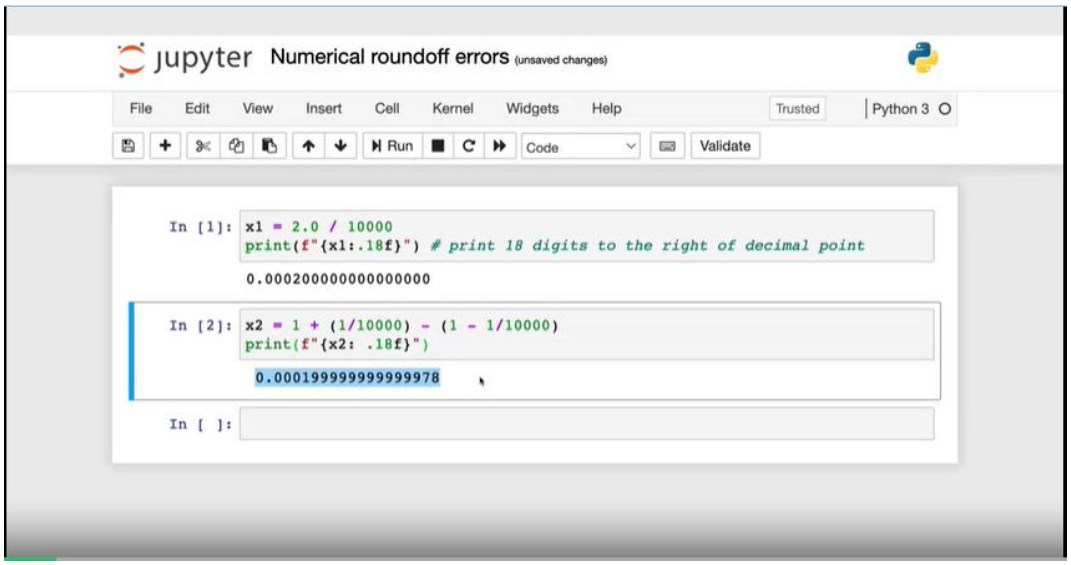


whereas for logistic regression we had the BinaryCrossentropy function, here we're using the SparseCategoricalCrossentropy function. And what sparse categorical refers to is that you're still classified y into categories. So it's categorical. This takes on values from 1 to 10. And sparse refers to that y can only take on one of these 10 values. So each image is either 0 or 1 or 2 or so on up to 9. You're not going to see a picture that is simultaneously the number two and the number seven so sparse refers to that each digit is only one of these categories.

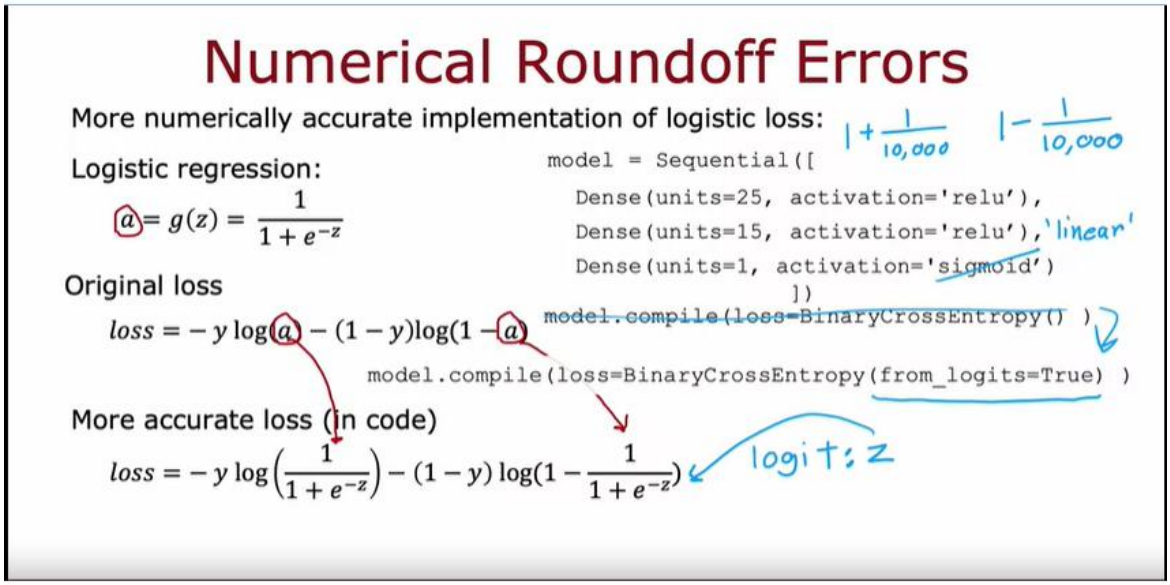


2.3 - Multiclass Classification : 2.3.4 - Improved implementation of softmax :

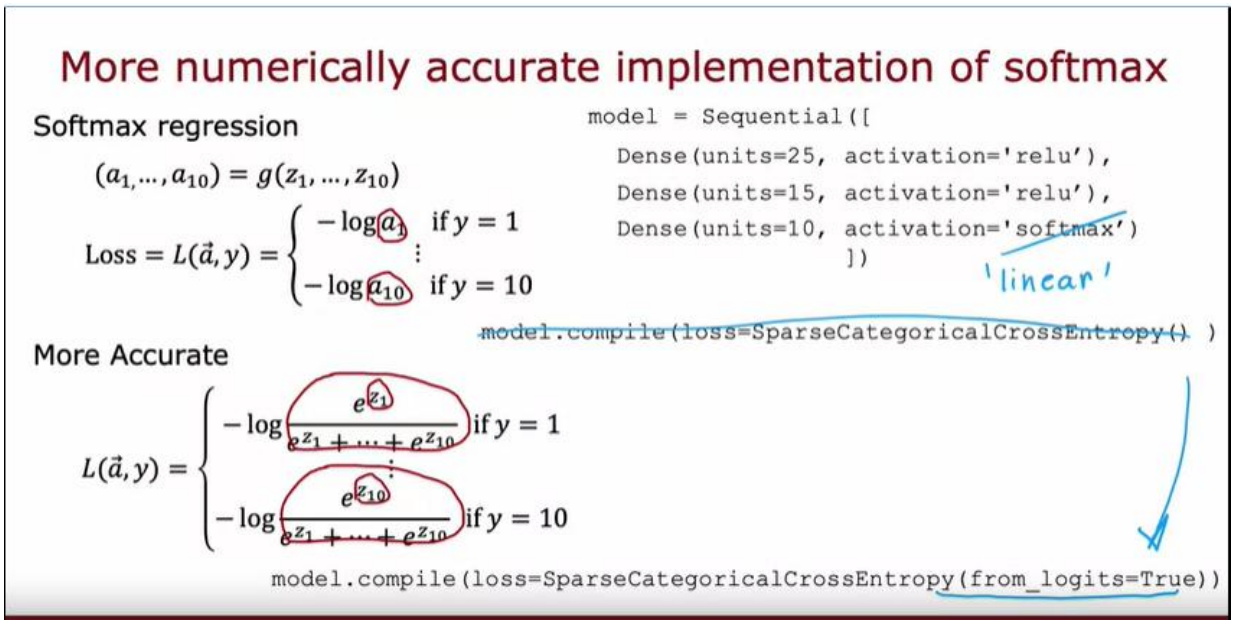




Both the codes work well for Logistic Regression. Numerical Round Off errors play vital role in softmax regression



Numerically More Accurate :



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)

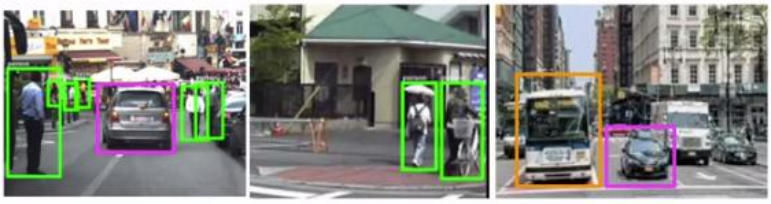
model
```
model = Sequential([
  Dense(units=25, activation='sigmoid'),
  Dense(units=15, activation='sigmoid'),
  Dense(units=1, activation='linear')
                ])
from tensorflow.keras.losses import
  BinaryCrossentropy
```
loss
```
model.compile(..., BinaryCrossentropy(from_logits=True))
model.fit(X,Y,epochs=100)
```
fit
```
logit = model(X)
```
z

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

2.3 - Multiclass Classification : 2.3.5 - Classification with multiple outputs (Optional) : Multi-Label Classification :

Multi-label Classification

$\vec{x}$

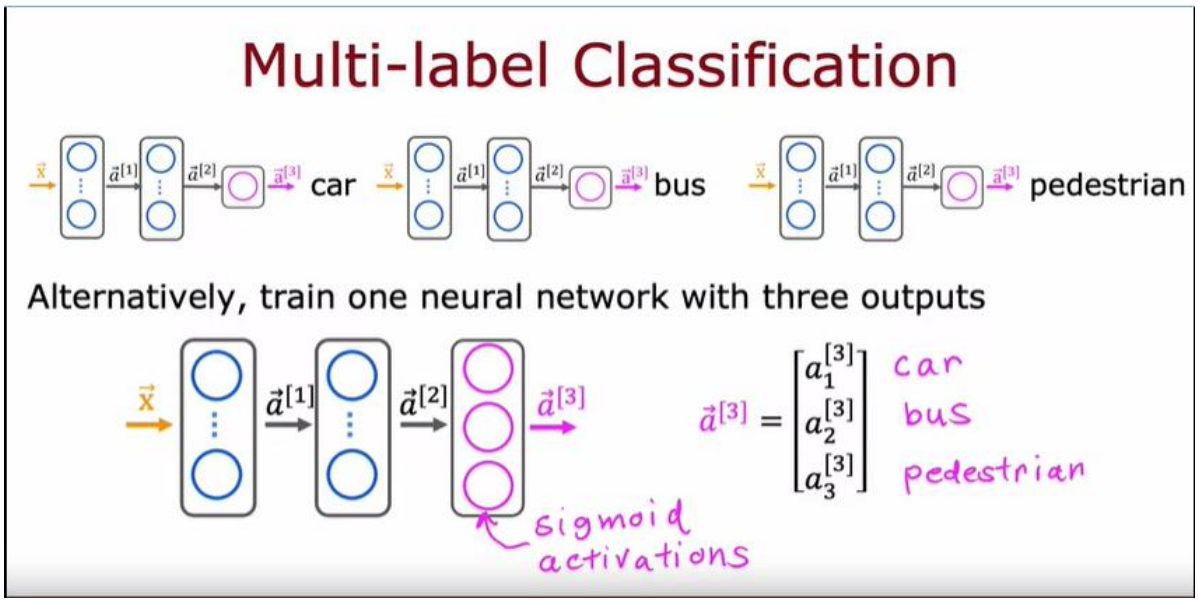


Is there a car?
Is there a bus?
Is there a pedestrian

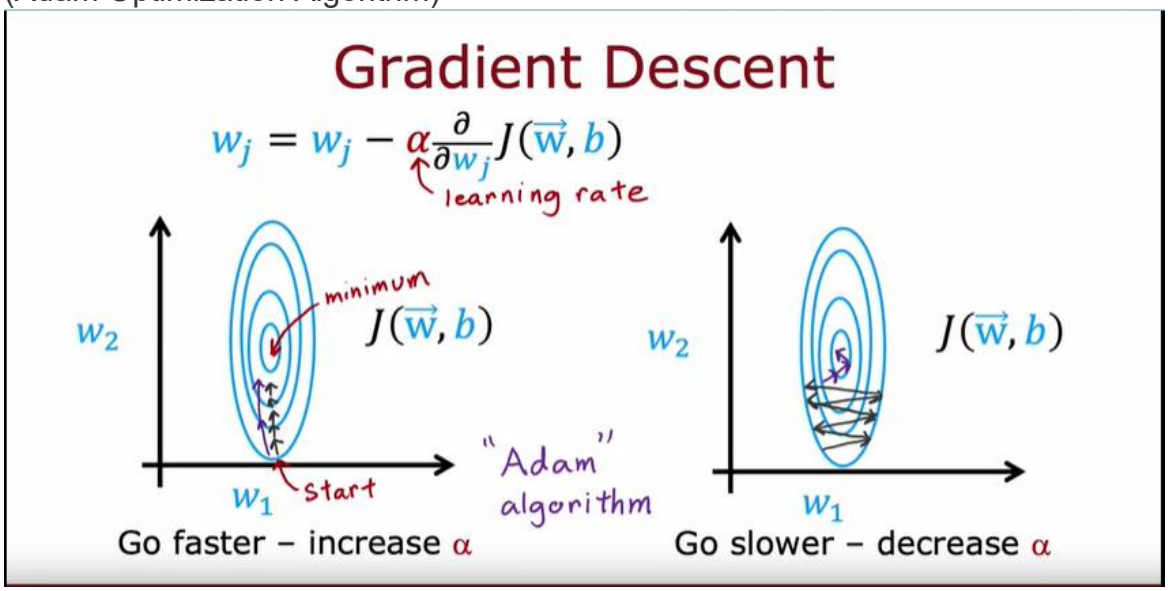
yes, no, yes $y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

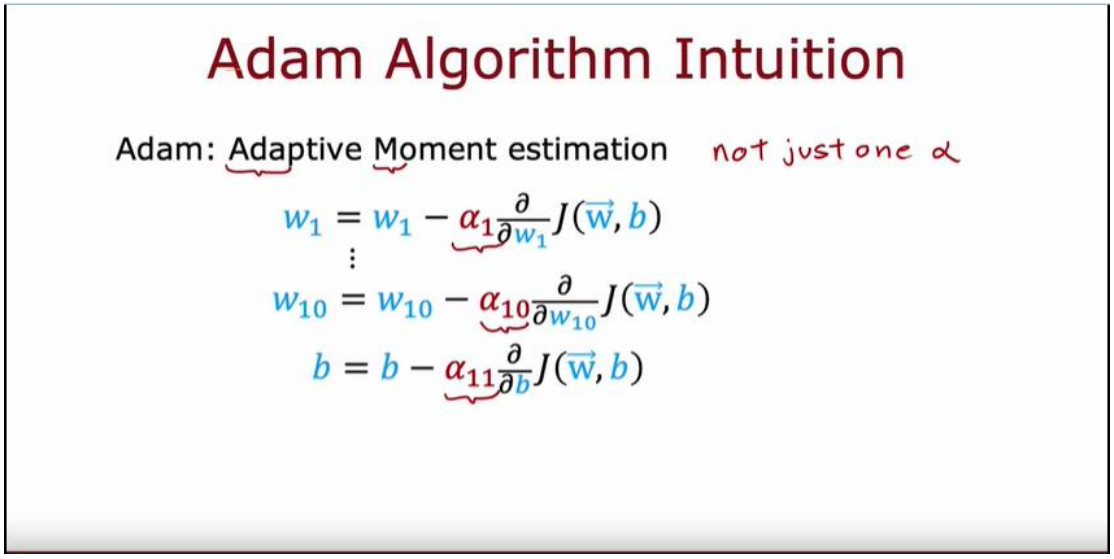
no, no, yes $y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

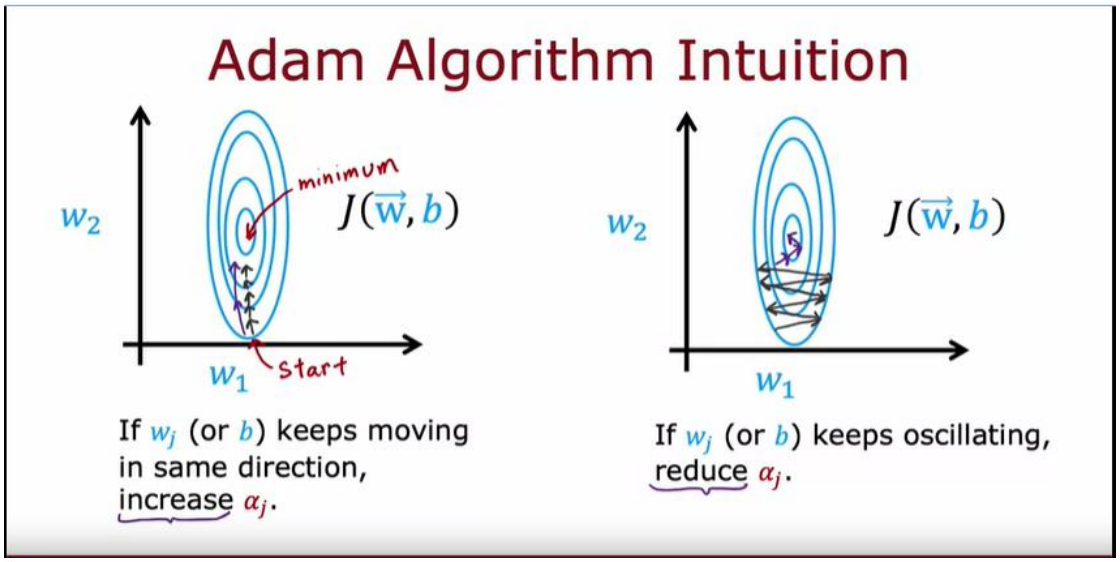
yes, yes, no $y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

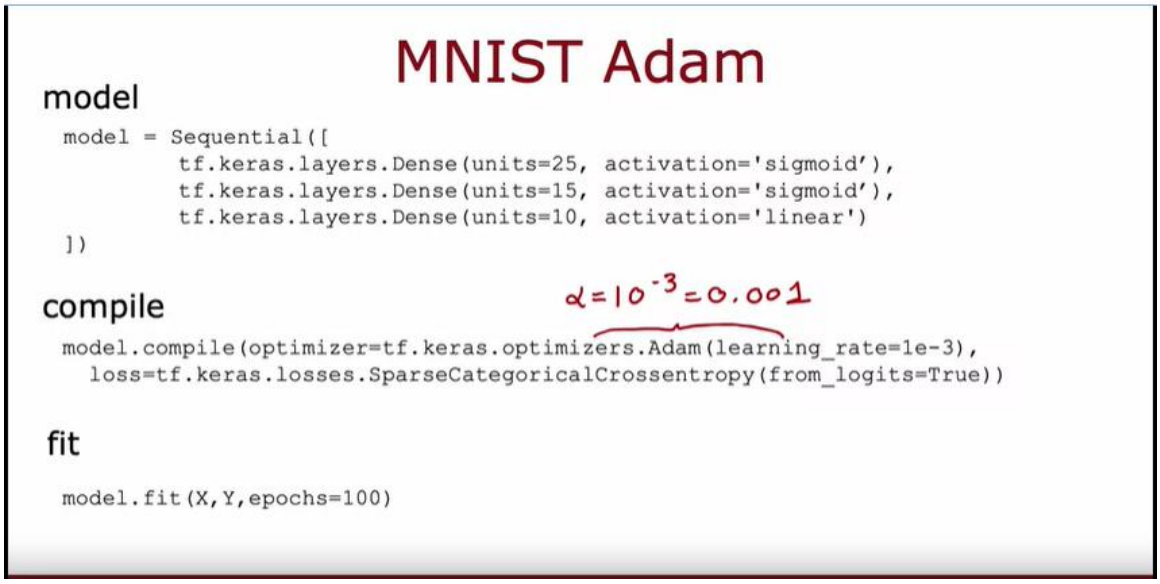


2.4 - Additional Neural Network Concepts : 2.4.1 - Advanced Optimization : (Adam Optimization Algorithm)

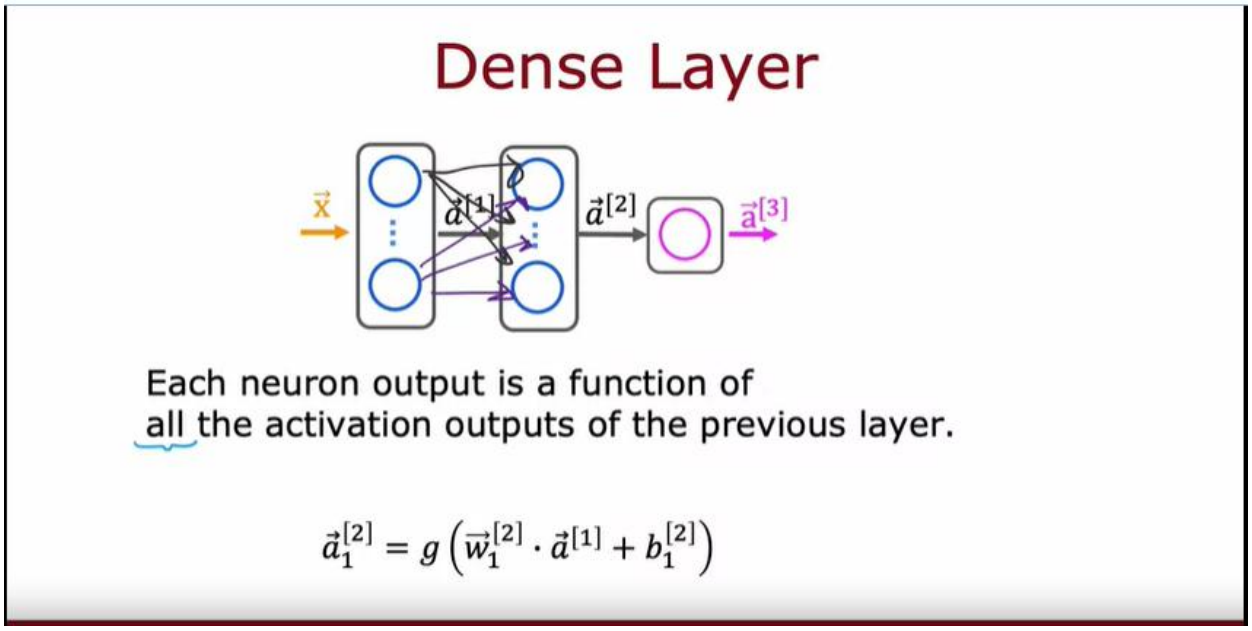


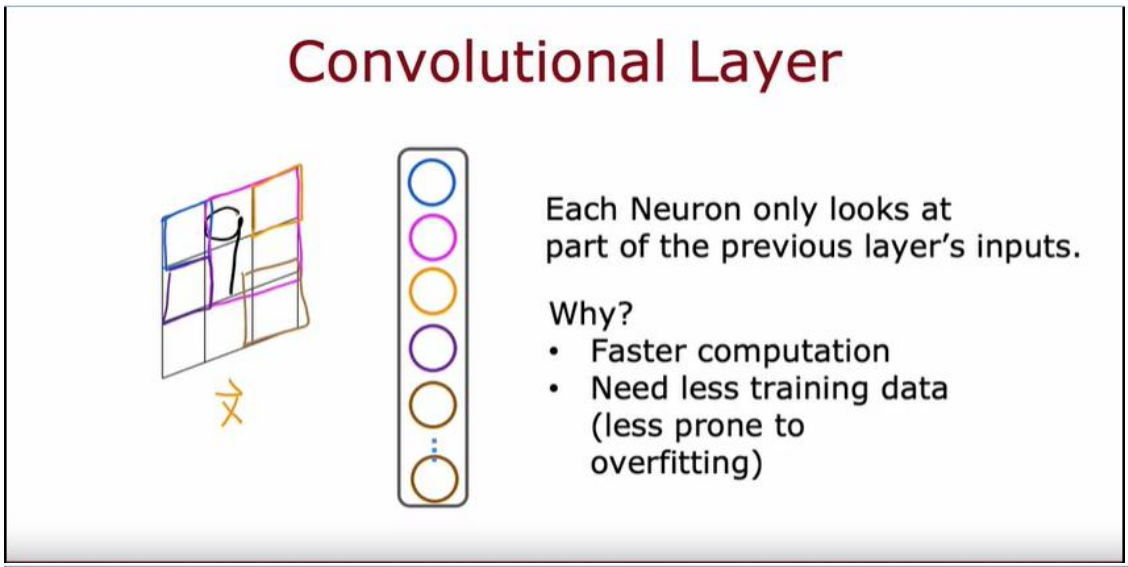


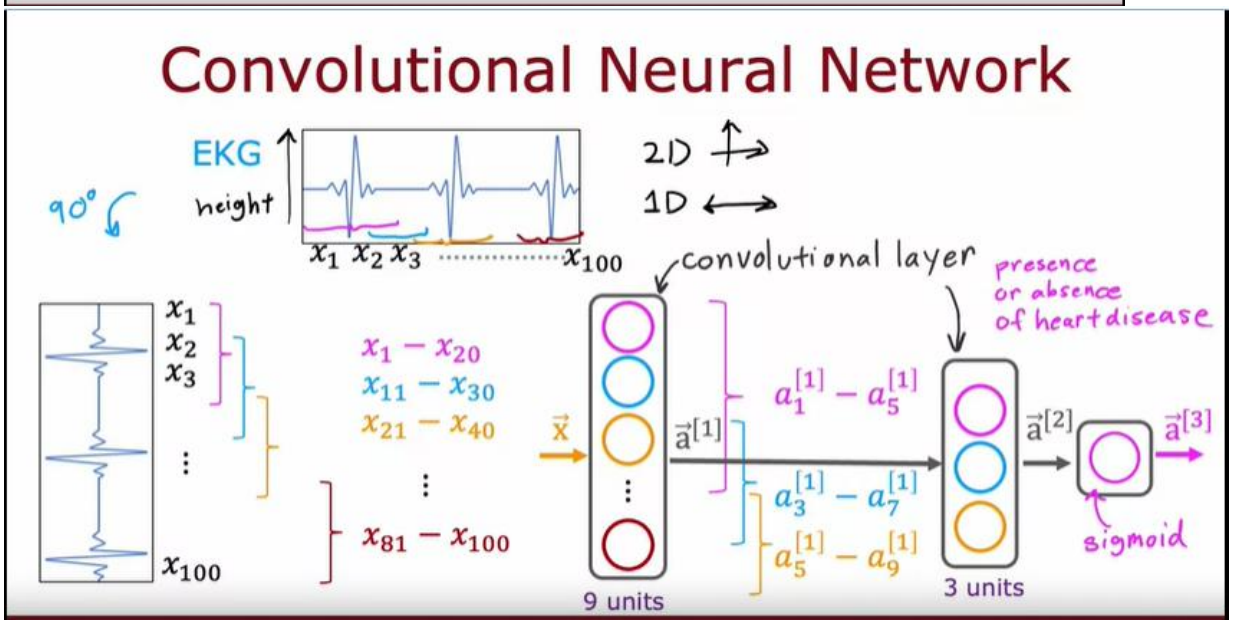




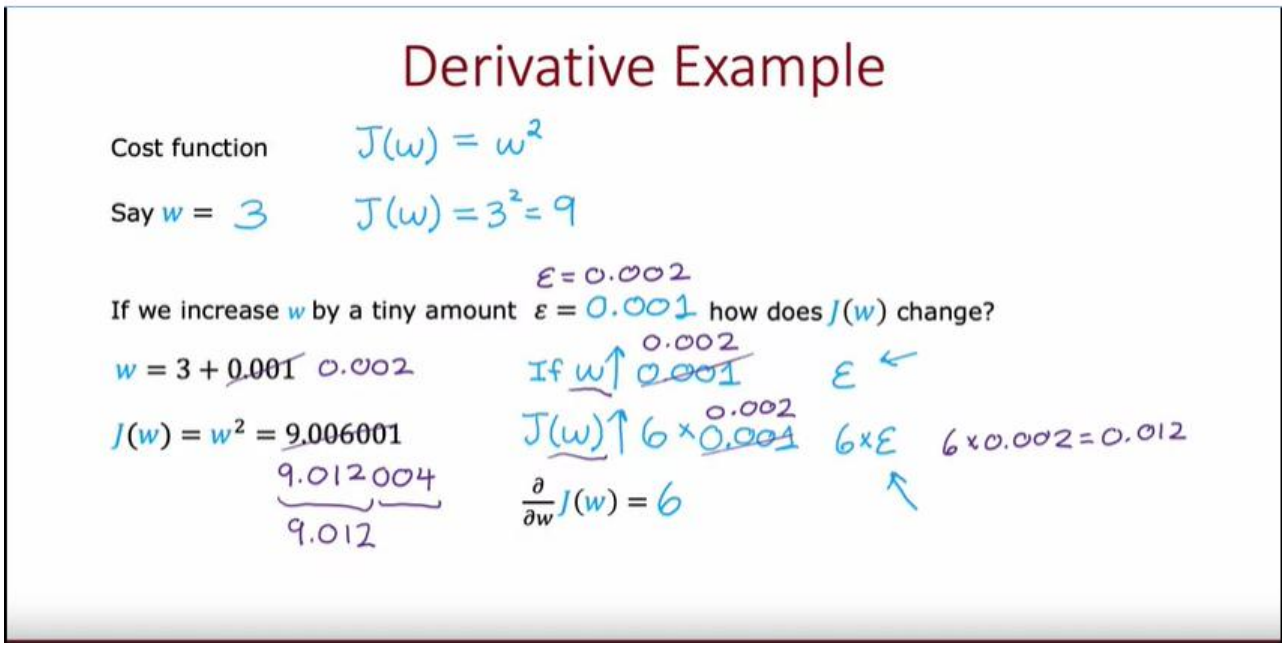
2.4 - Additional Neural Network Concepts : 2.4.2 - Additional Layer Types :







2.5 - Back Propagation (Optional) : 2.5.1 - What is a derivative? :



Informal Definition of Derivative

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

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

Gradient descent

repeat {

learning rate

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

}

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

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

you can use Python to compute these derivatives yourself using a nifty Python package called SymPy

More Derivative Examples

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

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

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

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

$J(w) = w^2$

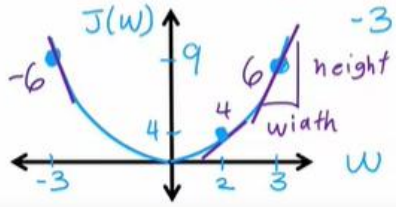
↓0.006

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

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

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

-3+0.001



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

Even More Derivative Examples

$w = 2$

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

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

$J(w) = w^3 = 8$

$J(w) = w = 2$

$J(w) = \frac{1}{w} = \frac{1}{2} = 0.5$

```
In [2]: J, w = sympy.symbols('J,w')

In [3]: J = w**2
        J
Out[3]: w^2

In [5]: dJ_dw = sympy.diff(J,w)
        dJ_dw
Out[5]: 2w

In [6]: dJ_dw.subs([(w,2)])
Out[6]: 4

In [ ]:
```

```
In [2]: J, w = sympy.symbols('J,w')

In [7]: J=w**3 #J = w**2
        J
Out[7]: w^3

In [8]: dJ_dw = sympy.diff(J,w)
        dJ_dw
Out[8]: 3w^2

In [9]: dJ_dw.subs([(w,2)])
Out[9]: 12

In [ ]:
```

```
In [2]: J, w = sympy.symbols('J,w')

In [10]: J=w #J=w**3 #J = w**2
         J
Out[10]: w

In [11]: dJ_dw = sympy.diff(J,w)
         dJ_dw
Out[11]: 1

In [12]: dJ_dw.subs([(w,2)])
Out[12]: 1

In [ ]:
```

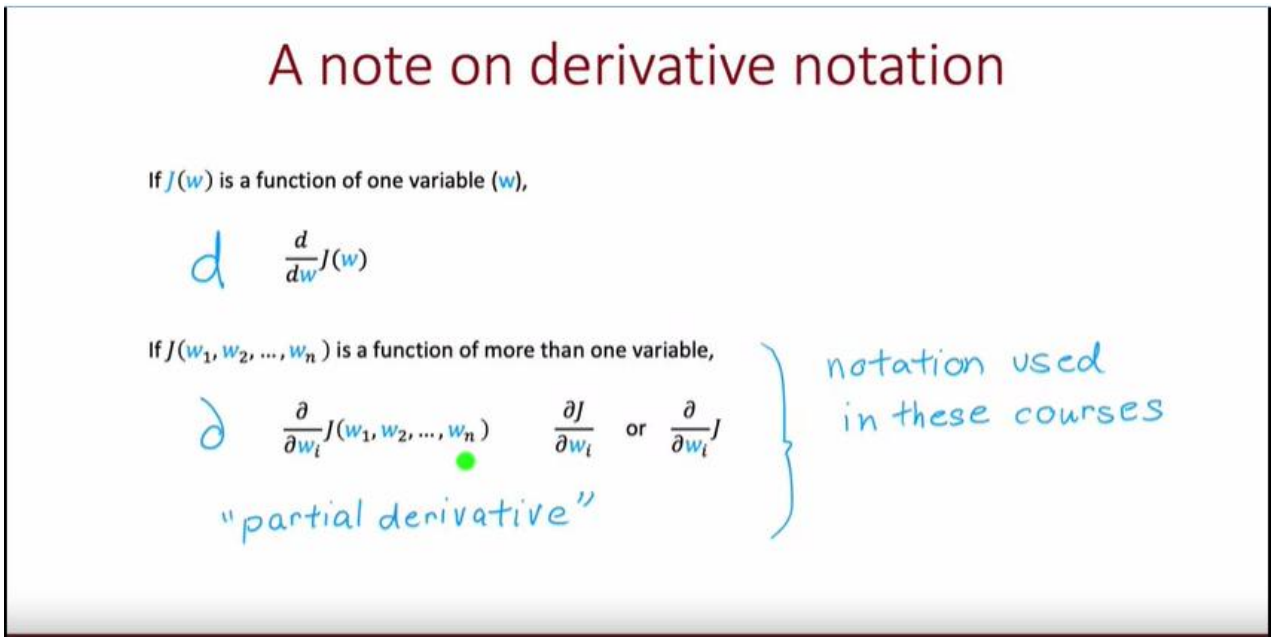
```
In [1]: import sympy

In [2]: J, w = sympy.symbols('J,w')

In [13]: J=1/w #J=w #J=w**3 #J = w**2
         J
Out[13]: 1/w

In [14]: dJ_dw = sympy.diff(J,w)
         dJ_dw
Out[14]: -1/w^2

In [15]: dJ_dw.subs([(w,2)])
Out[15]: -1/4

In [ ]:
```

Even More Derivative Examples

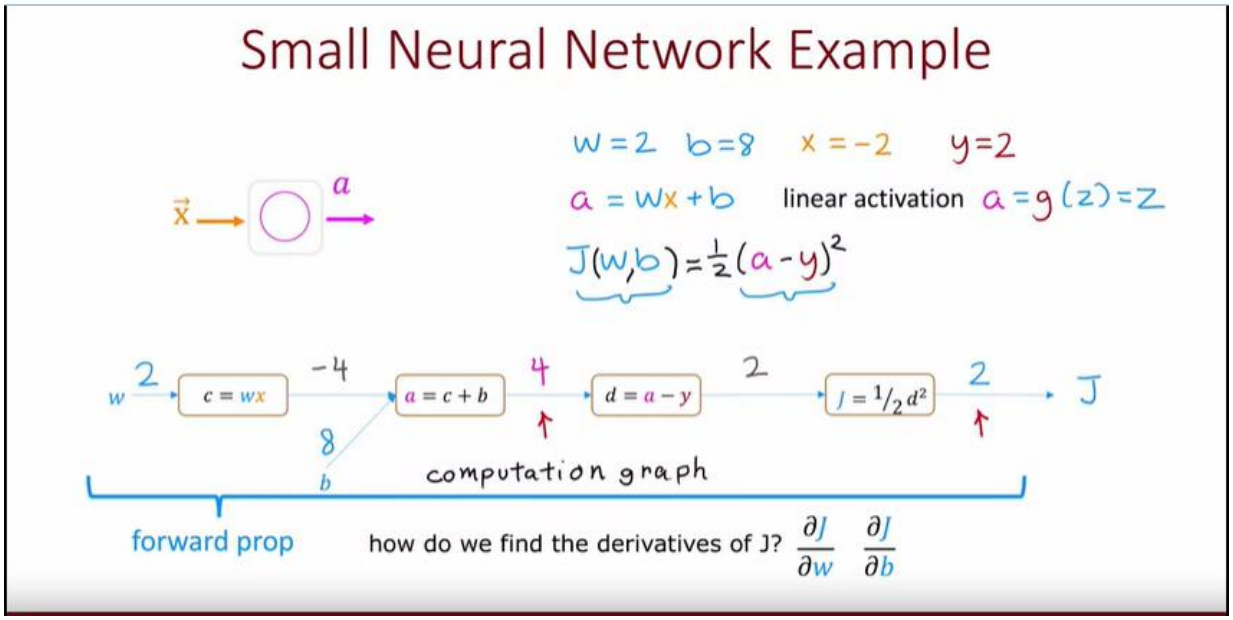
$w = 2$

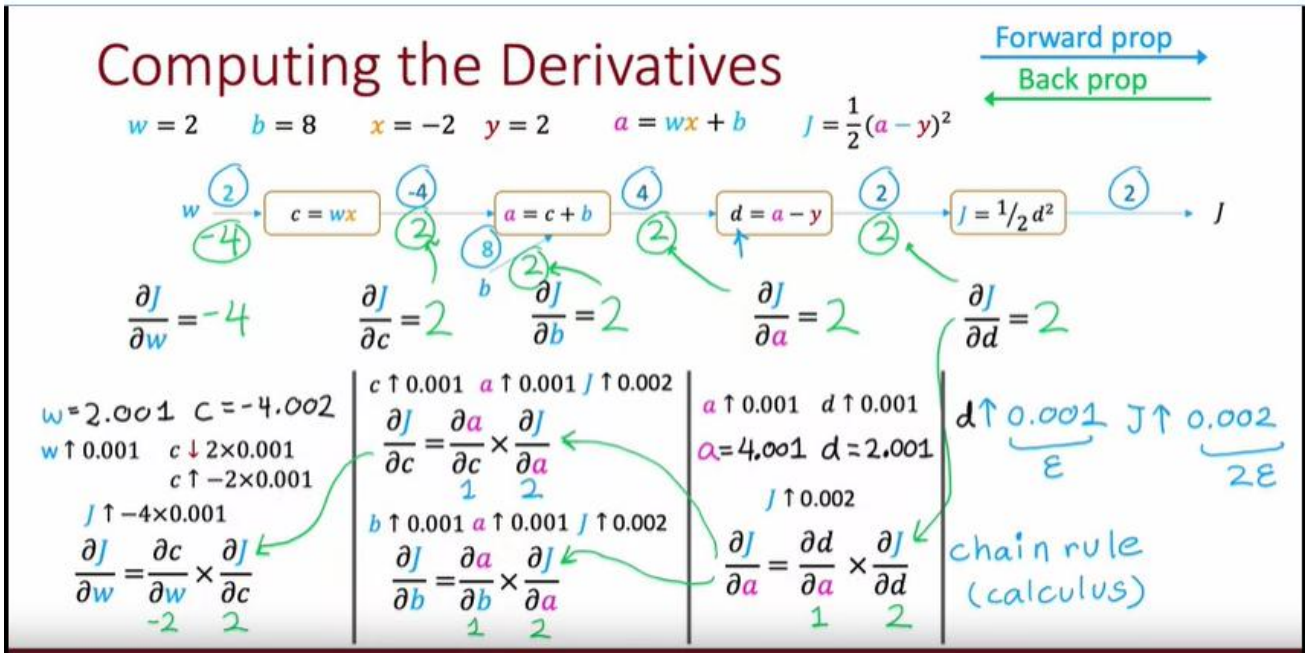
$J(w) = w^2 = 4$	$\frac{\partial}{\partial w} J(w) = 2w = 4$	$w \uparrow 0.001$ (ε)	$J(w) = 4.004001$ $J(w) \uparrow 4 \times \varepsilon$
$J(w) = w^3 = 8$	$\frac{\partial}{\partial w} J(w) = 3w^2 = 12$	$w \uparrow \varepsilon$	$J(w) = 8.012006$ $J(w) \uparrow 12 \times \varepsilon$
$J(w) = w = 2$	$\frac{\partial}{\partial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

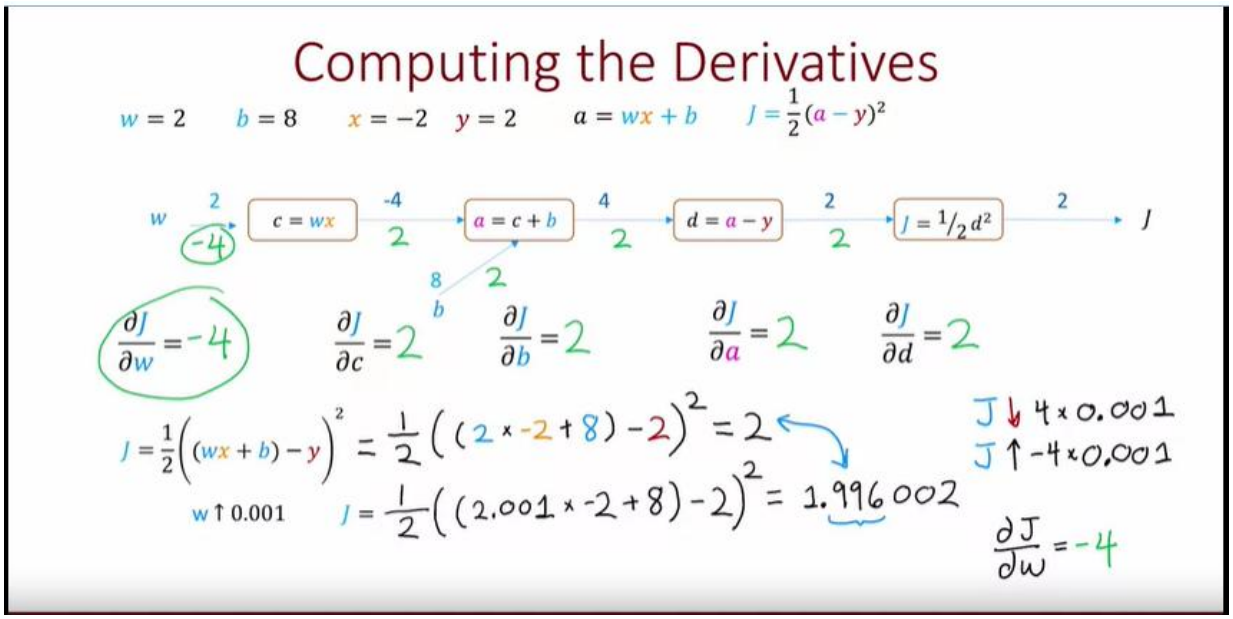
$\frac{\partial}{\partial w} J(w)$ $\quad w \uparrow \varepsilon$ $\quad J(w) \uparrow k \times \varepsilon$

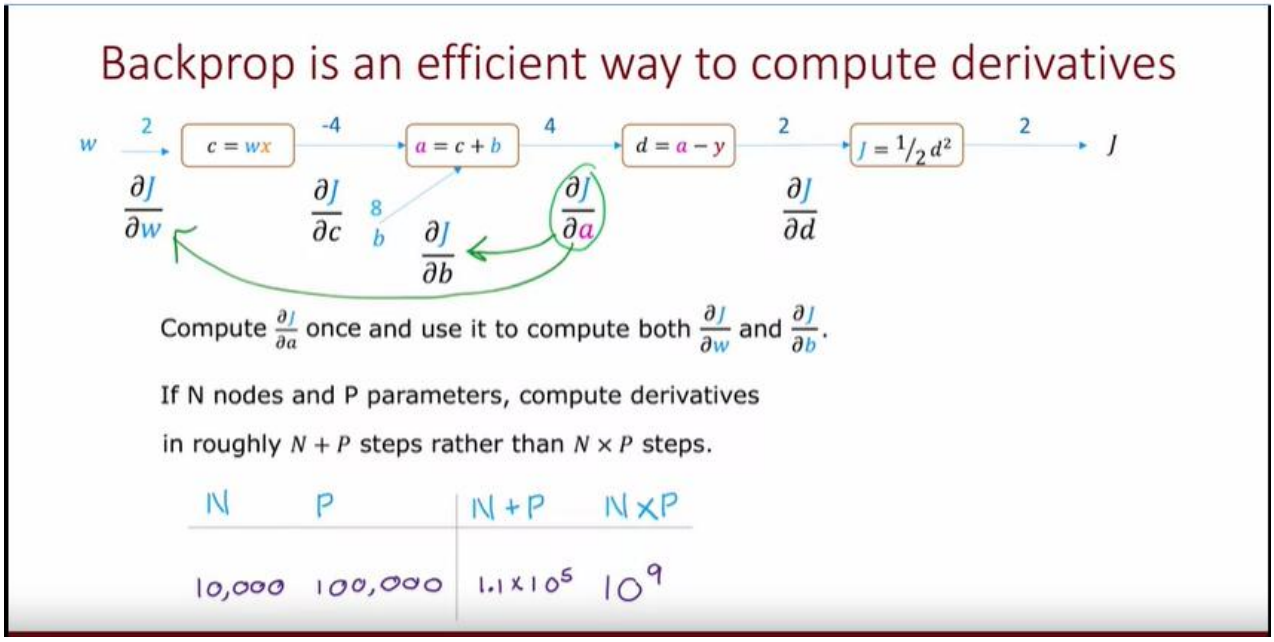


2.5 - Back Propagation (Optional) : 2.5.2 - Computation graph :

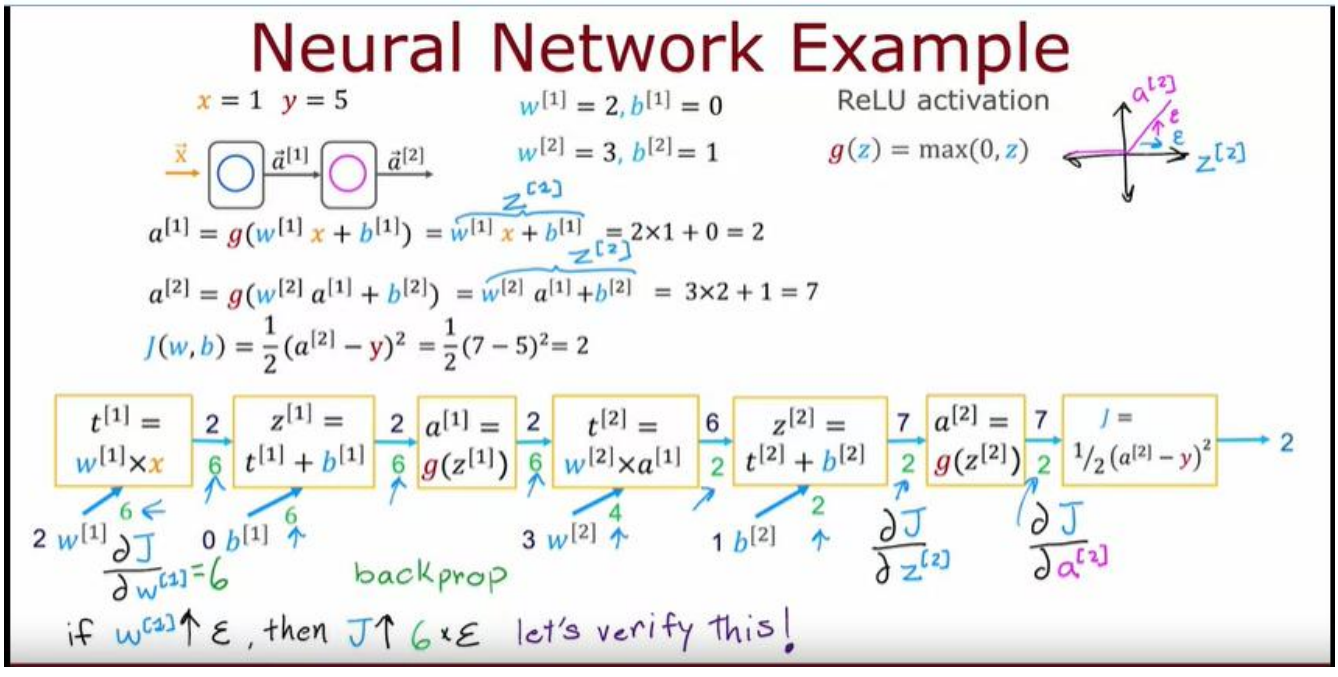








2.5 - Back Propagation (Optional) : 2.5.3 - Larger neural network example :



Neural Network Example

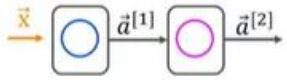
$x = 1 \quad y = 5$

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

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

ReLU activation

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



$a^{[1]} = g(w^{[1]} x + b^{[1]}) = w^{[1]} x + b^{[1]} = 2 \times 1 + 0 = 2$ → 2.001

$a^{[2]} = g(w^{[2]} a^{[1]} + b^{[2]}) = w^{[2]} a^{[1]} + b^{[2]} = 3 \times 2 + 1 = 7$ → 2.001 → 7.003

$J(w,b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(7-5)^2 = 2$ → 7.003 → $\frac{(2.003)^2}{2}$ → 2.006005

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

$\frac{\partial J}{\partial w^{[1]}} \quad \frac{\partial J}{\partial b^{[1]}} \quad \frac{\partial J}{\partial w^{[2]}} \quad \frac{\partial J}{\partial b^{[2]}}$

N nodes

P parameters

$w_1, b_1, w_2, b_2 \ldots$

inefficient way

N x P

efficient way (backprop)

N + P

Part (Week) 3 : Machine Learning

3.1 - Advice for applying machine learning
- 3.1.1 - Deciding what to try next. Duration: 3 minutes3 min
- 3.1.2 - Evaluating a model. Duration: 10 minutes10 min
- 3.1.3 - Model selection and training/cross validation/test sets. Duration: 13 minutes13 min
- Lab: Optional Lab: Model Evaluation and Selection. Duration: 30 minutes30 min
- Practice quiz: Advice for applying machine learning

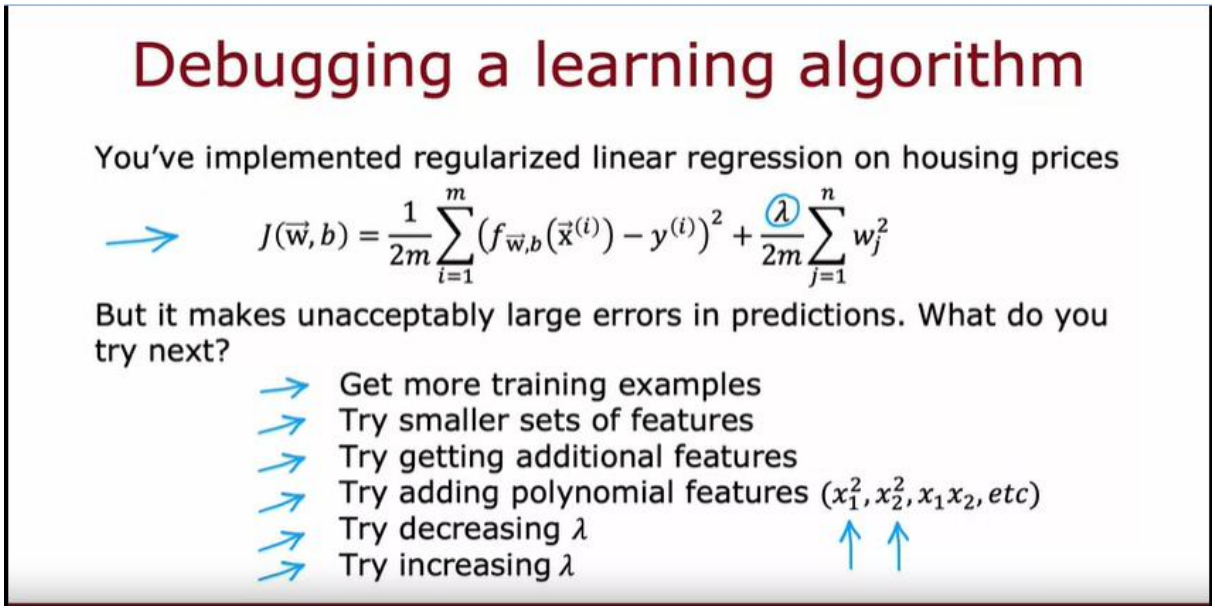
3.2 - Bias and variance
- 3.2.1 - Diagnosing bias and variance. Duration: 11 minutes11 min
- 3.2.2 - Regularization and bias/variance. Duration: 10 minutes10 min
- 3.2.3 - Establishing a baseline level of performance. Duration: 9 minutes9 min
- 3.2.4 - Learning curves. Duration: 11 minutes11 min
- 3.2.5 - Deciding what to try next revisited. Duration: 8 minutes8 min
- 3.2.6 - Bias/variance and neural networks. Duration: 10 minutes10 min
- Lab: Optional Lab: Diagnosing Bias and Variance. Duration: 30 minutes30 min
- Practice quiz: Bias and variance

3.3 - Machine learning development process
- 3.3.1 - Iterative loop of ML development. Duration: 7 minutes7 min
- 3.3.2 - Error analysis. Duration: 8 minutes8 min
- 3.3.3 - Adding data. Duration: 14 minutes14 min
- 3.3.4 - Transfer learning: using data from a different task. Duration: 11 minutes11 min
- 3.3.5 - Full cycle of a machine learning project. Duration: 8 minutes8 min
- 3.3.6 - Fairness, bias, and ethics. Duration: 9 minutes9 min
- Practice quiz: Machine learning development process

3.4 - Skewed datasets (optional)
- 3.4.1 - Error metrics for skewed datasets. Duration: 11 minutes11 min
- 3.4.2 - Trading off precision and recall. Duration: 11 minutes11 min
- Practice Lab: Advice for applying machine learning

3.4 - Skewed datasets (optional) : 3.4.2 - Trading off precision and recall :

3.1 - Advice for applying machine learning : 3.1.1 - Deciding what to try next : :



3.1 - Advice for applying machine learning : 3.1.1 - Deciding what to try next :

Machine learning diagnostic

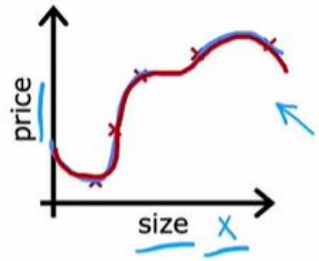
Diagnostic:

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.

3.1 - Advice for applying machine learning : 3.1.2 - 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.

$x = x_1$

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

$w_4 x^4$

just x

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

$f(\vec{x})$

Evaluating your model

Dataset:

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

70%

30%

training set → $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ⋮, $(x^{(m_{train})}, y^{(m_{train})})$

m_{train} = no. training examples = 7

test set → $(x_{test}^{(1)}, y_{test}^{(1)})$, ⋮, $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. test examples = 3

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

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

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

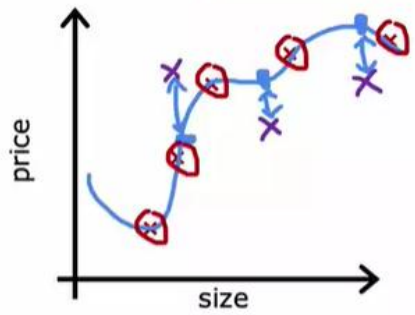
Compute test error:

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

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

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

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

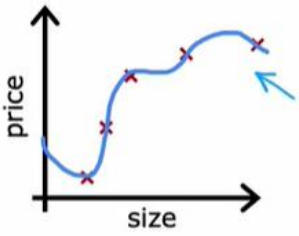
count $\hat{y} \neq y$

$J_{test}(\vec{w}, b)$ 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.

3.1 - Advice for applying machine learning : 3.1.3 - 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.

$J_{test}(\vec{w}, b)$ is better estimate of how well the model will generalize to new data compared to $J_{train}(\vec{w}, b)$.

$x = x_1$

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

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% → $(x^{(1)}, y^{(1)}) \ \vdots \ (x^{(m_{train})}, y^{(m_{train})})$ $m_{train} = 6$

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

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

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

⋮ ⋮ ⋮

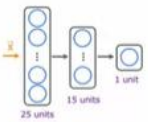
$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b$ $J_{cv}(w^{<10>}, b^{<10>})$

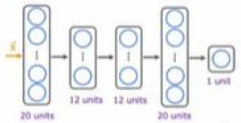
→ Pick $w_1 x + \cdots + w_4 x^4 + b$ $(J_{cv}(w^{<4>}, b^{<4>}))$

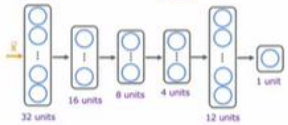
Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$

It's considered best practice in machine learning that if you have to make decisions about your model, such as fitting parameters or choosing the model architecture, such as neural network architecture or degree of polynomial if you're fitting a linear regression, to make all those decisions only using your training set and your cross-validation set, and to not look at the test set at all while you're still making decisions regarding your learning algorithm. It's only after you've come up with one model as your final model to only then evaluate it on the test set and because you haven't made any decisions using the test set, that ensures that your test set is a fair and not overly optimistic estimate of how well your model will generalize to new data.

Model selection – choosing a neural network architecture

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

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

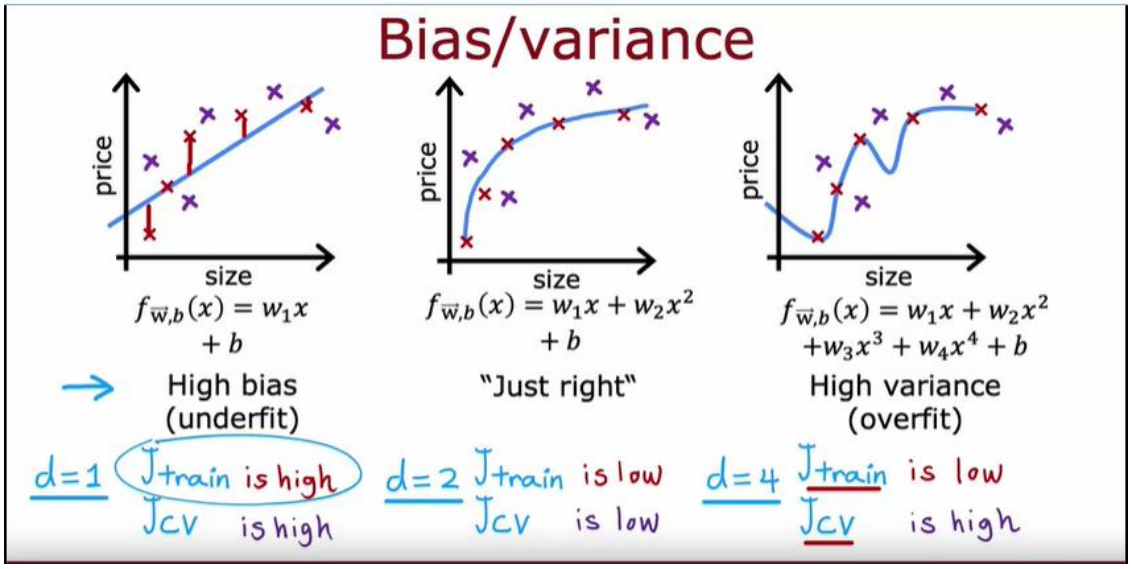
3.  $w^{<3>}, b^{<3>}$ $J_{cv}(w^{<3>}, b^{<3>})$

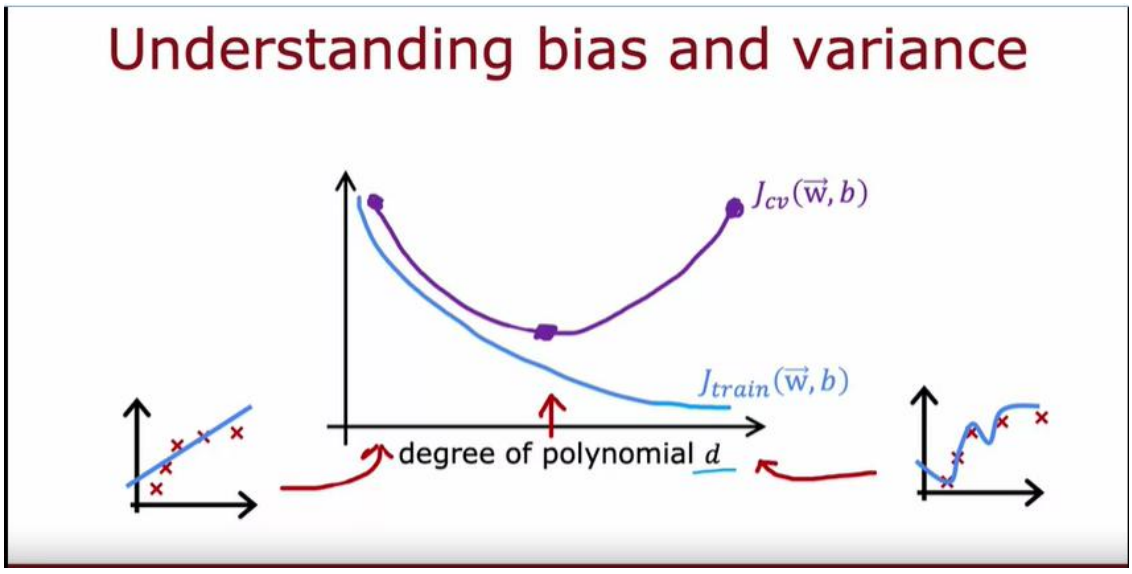
train, CV

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

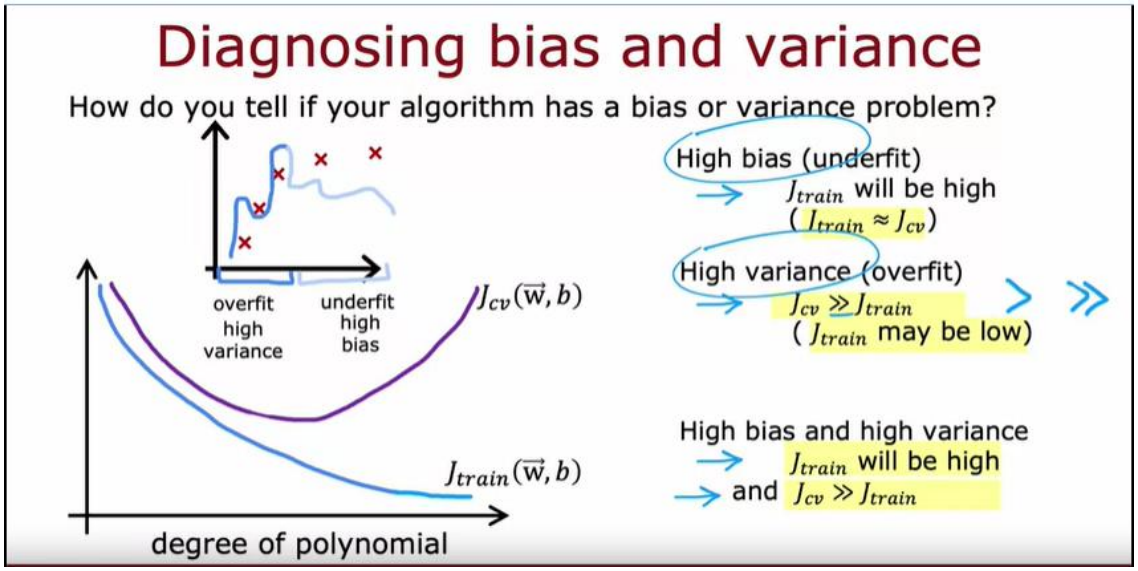
Estimate generalization error using the test set: $J_{test}(w^{<2>}, b^{(<2>)})$

3.2 - Bias and variance : 3.2.1 - Diagnosing bias and variance :

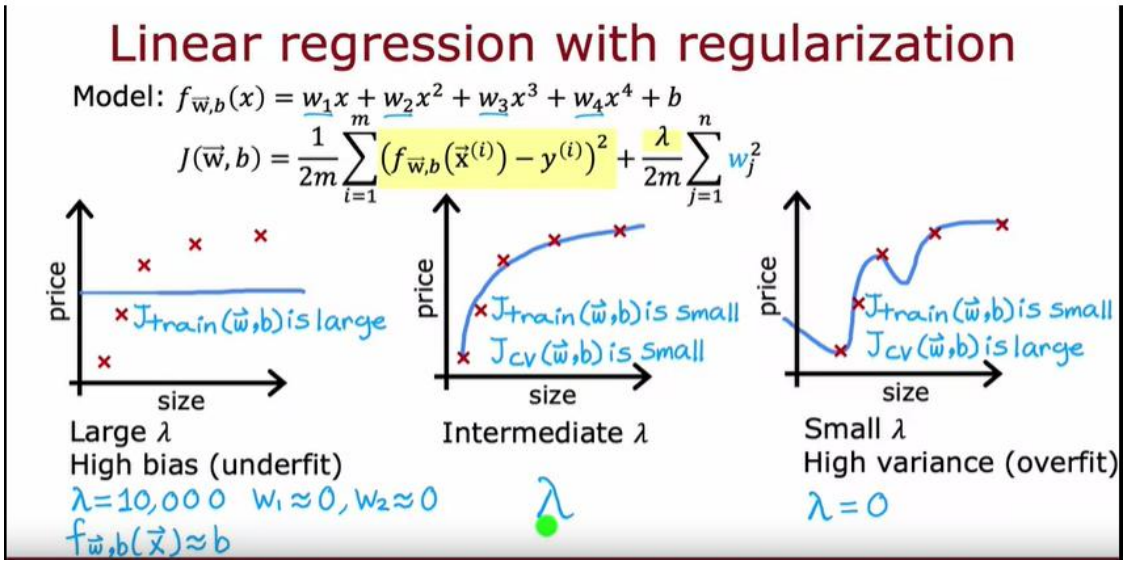




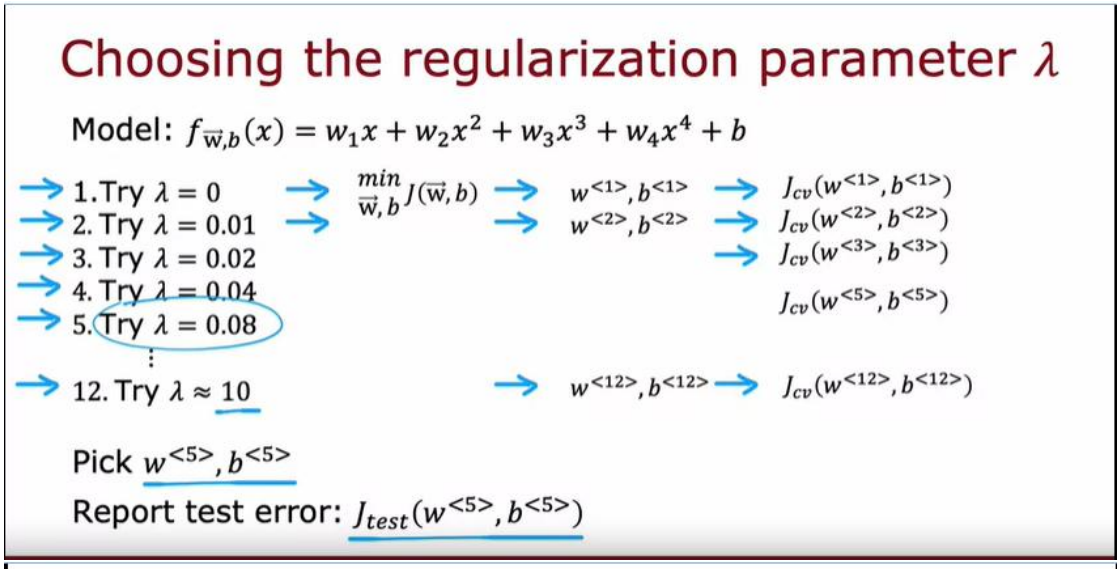
In some cases, it is possible to simultaneously have high bias and have high-variance. You won't see this happen that much for linear regression, but it turns out that if you're training a neural network, there are some applications where unfortunately you have high bias and high variance.

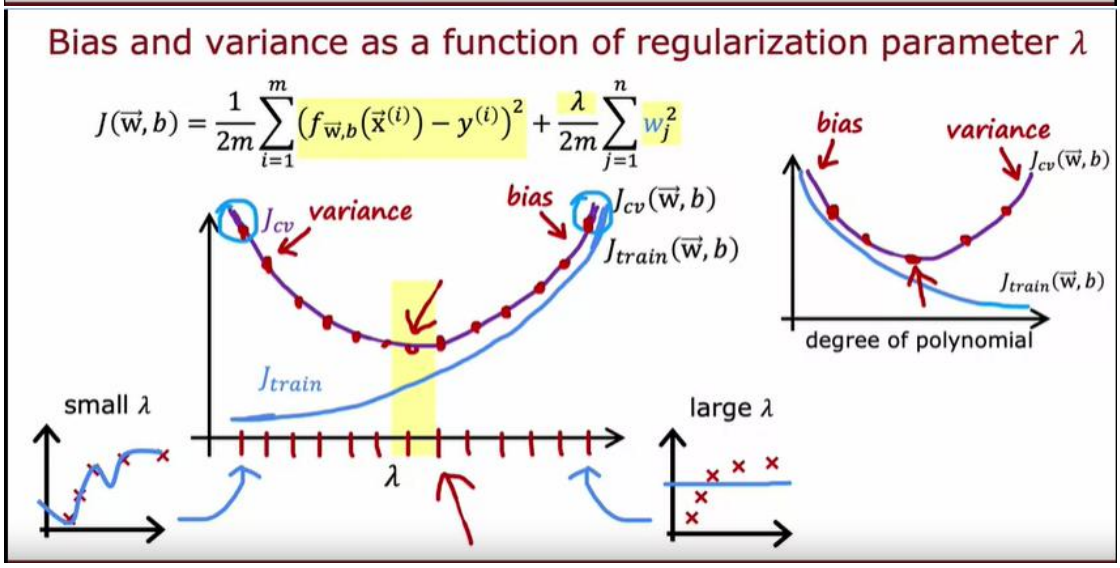


3.2 - Bias and variance : 3.2.2 - Regularization and bias/variance :

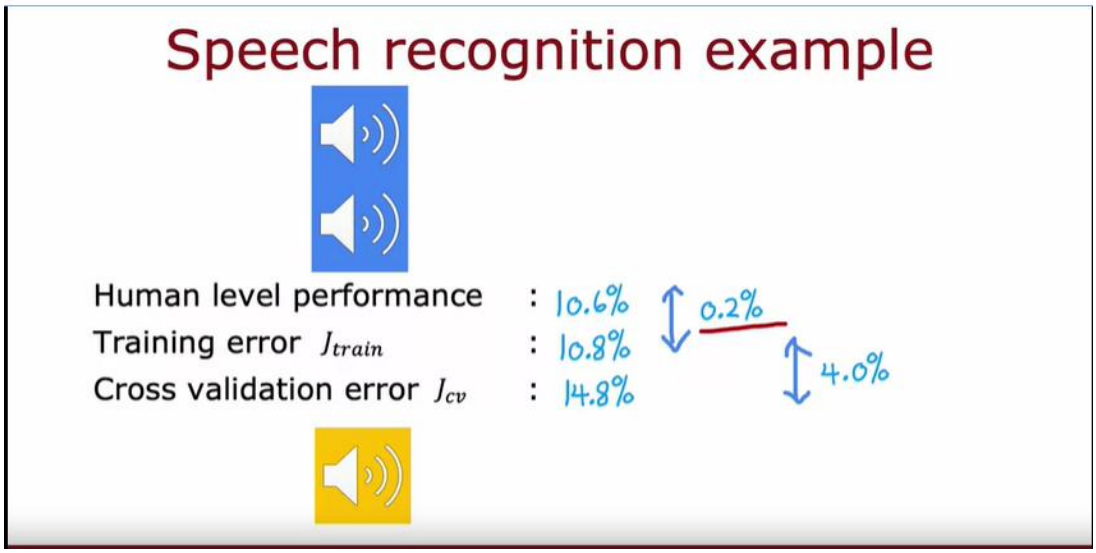


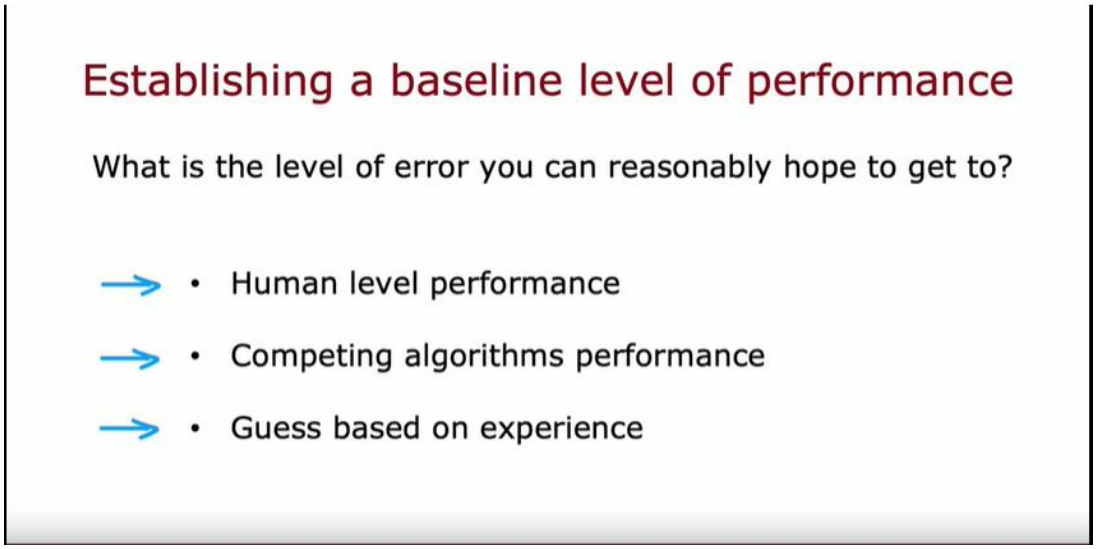
The problem we're addressing is if you're fitting a fourth-order polynomial, so that's the model and you're using regularization, how can you choose a good value of Lambda?

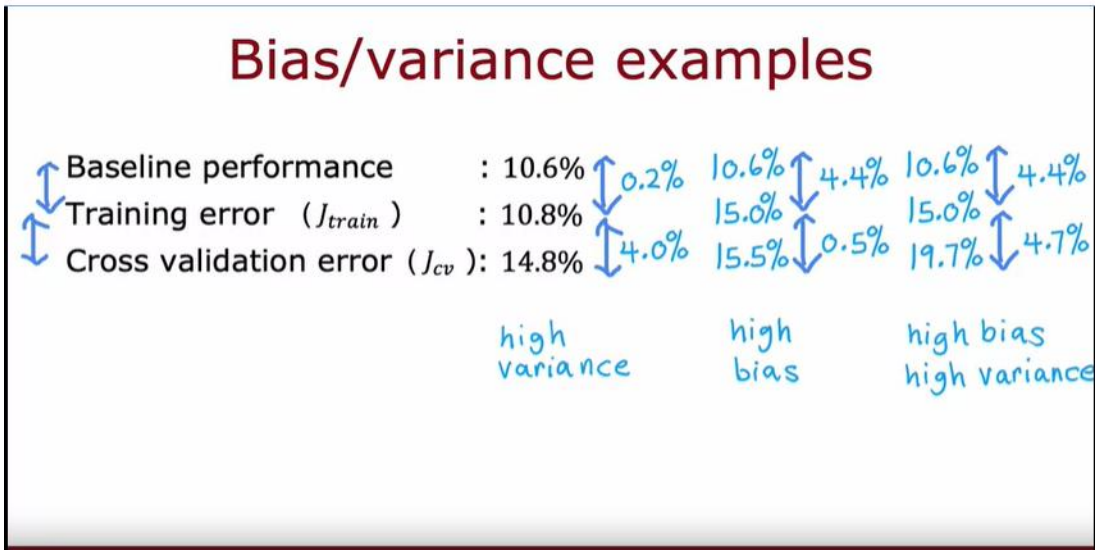




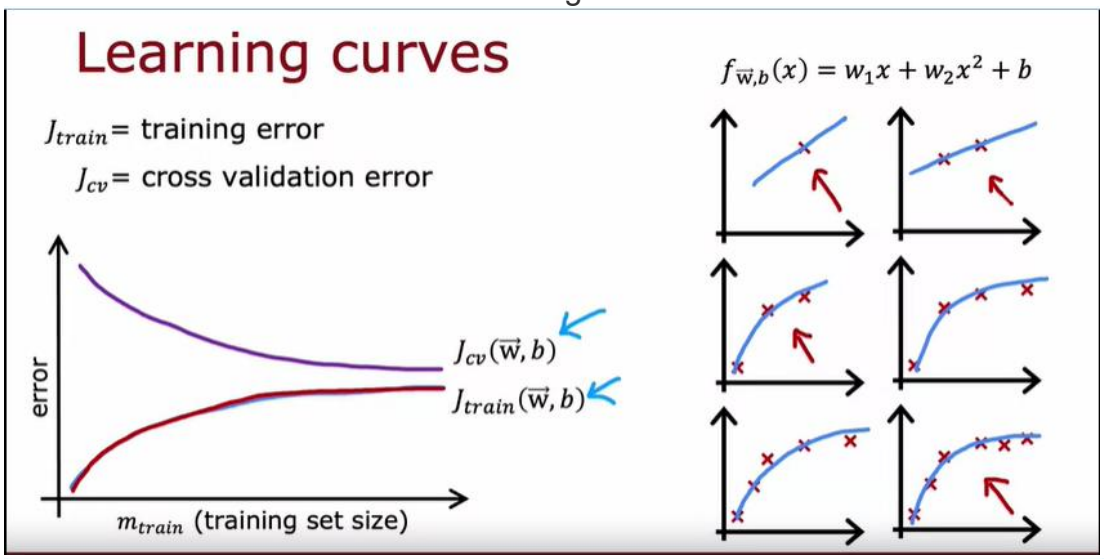
3.2 - Bias and variance : 3.2.3 - Establishing a baseline level of performance :

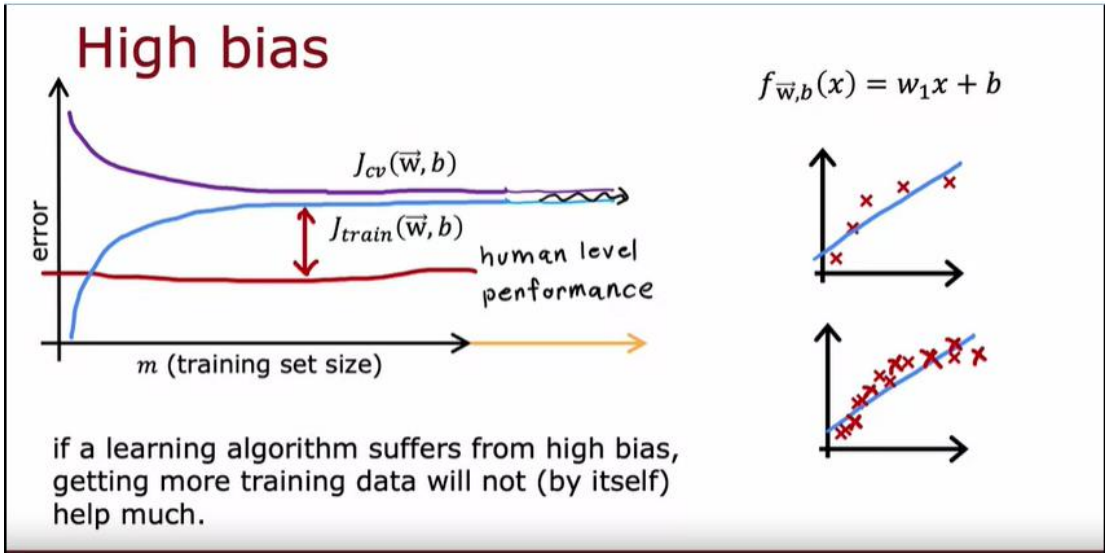


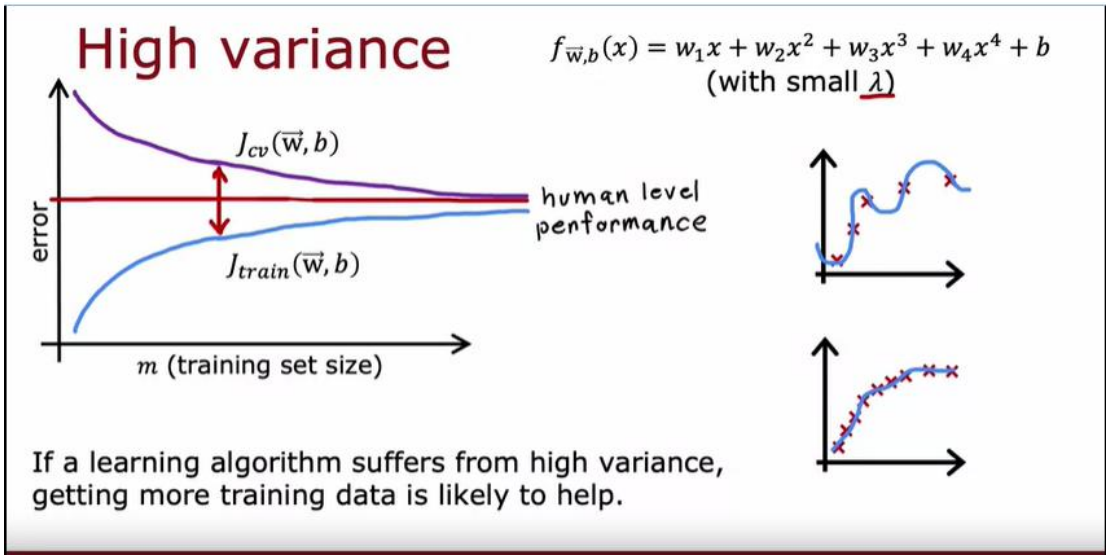




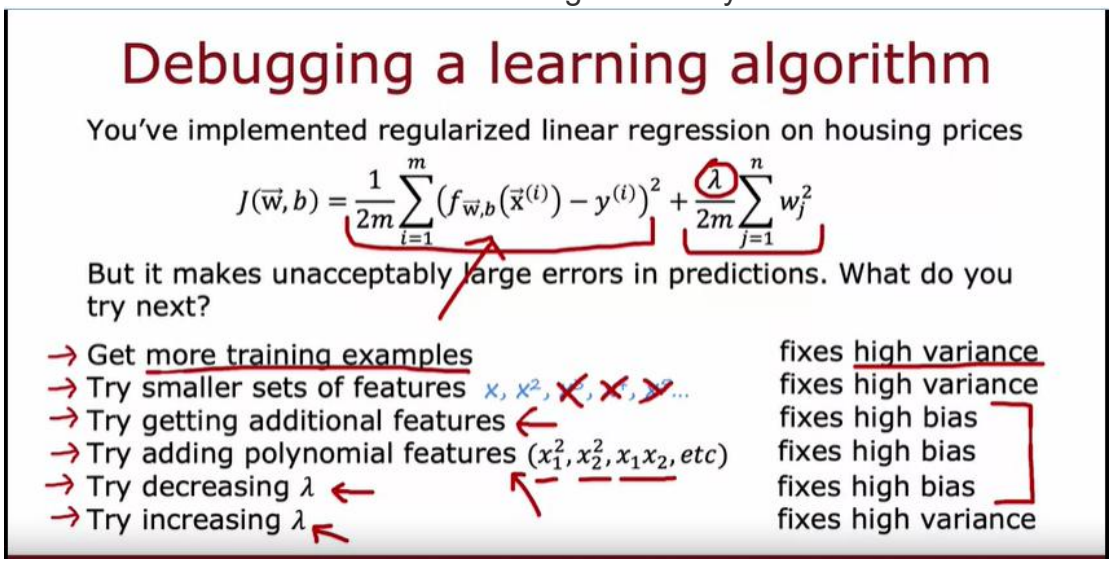
3.2 - Bias and variance : 3.2.4 - Learning curves :



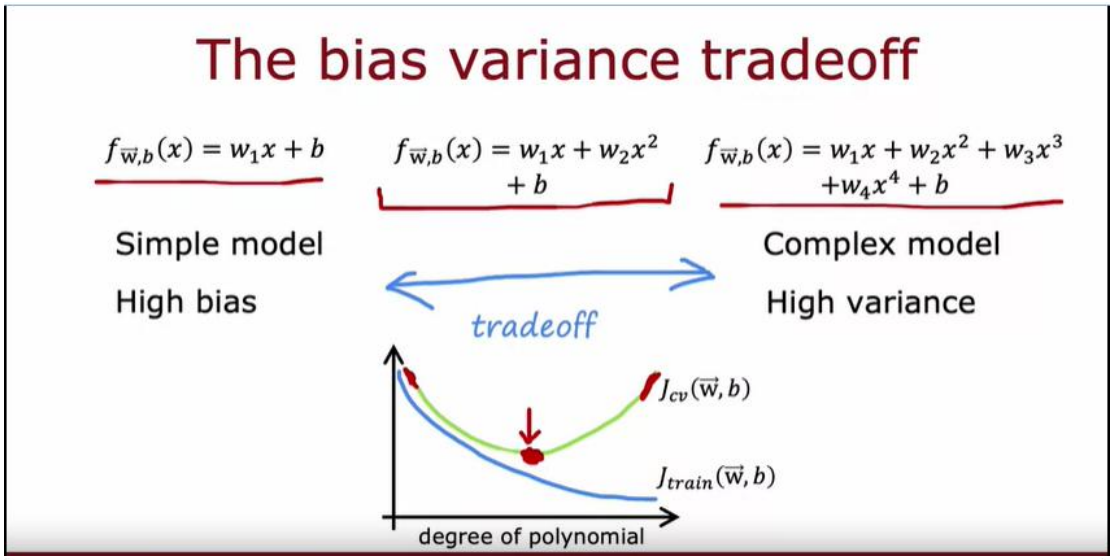


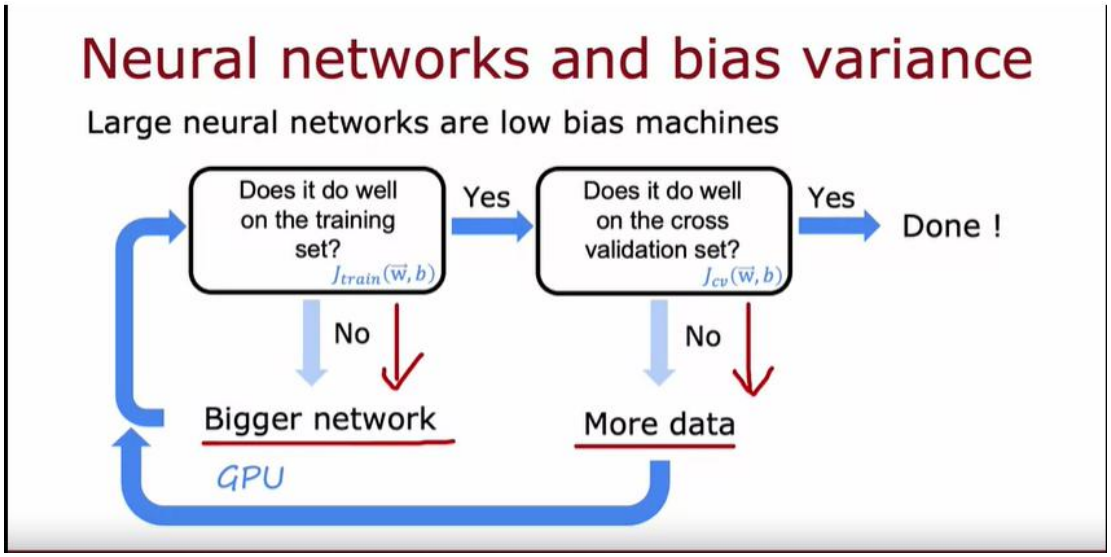


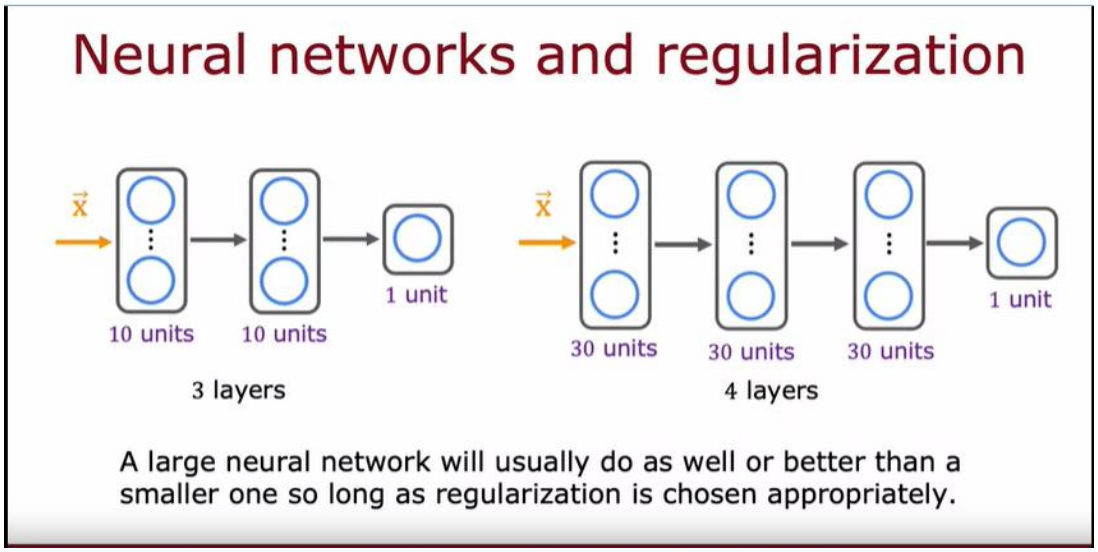
3.2 - Bias and variance : 3.2.5 - Deciding what to try next revisited :

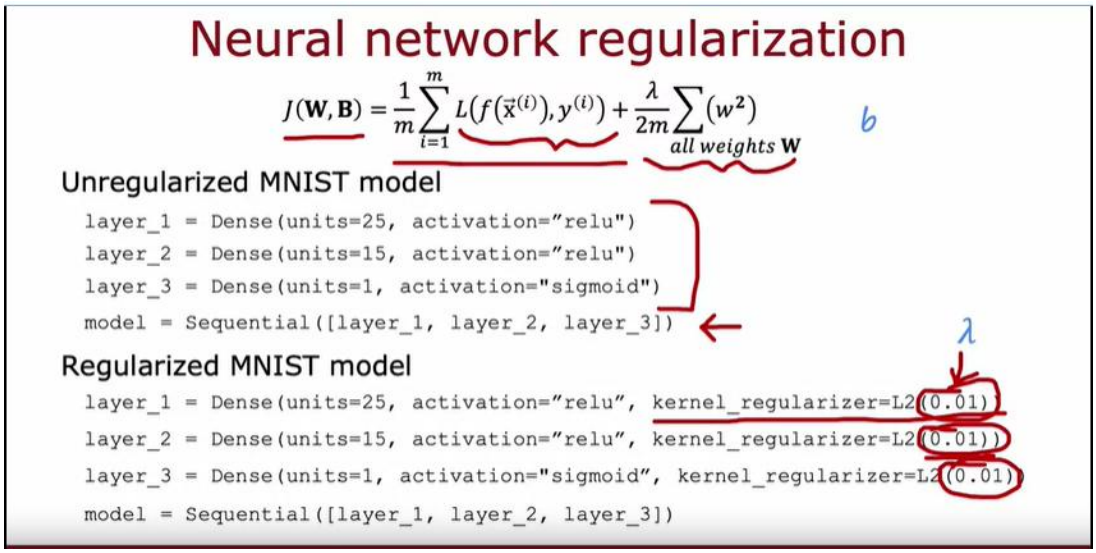


3.2 - Bias and variance : 3.2.6 - Bias/variance and neural networks :

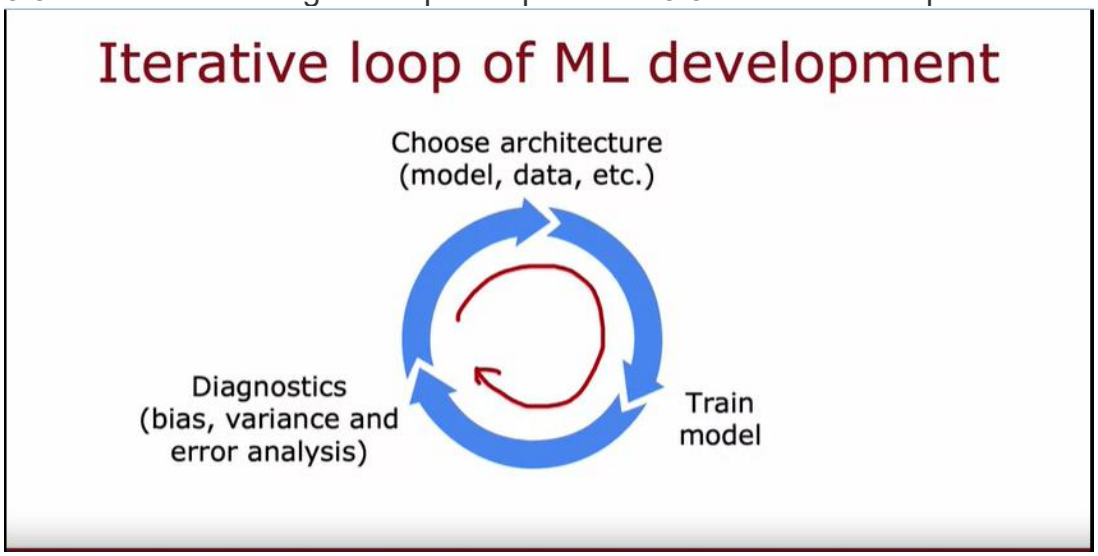


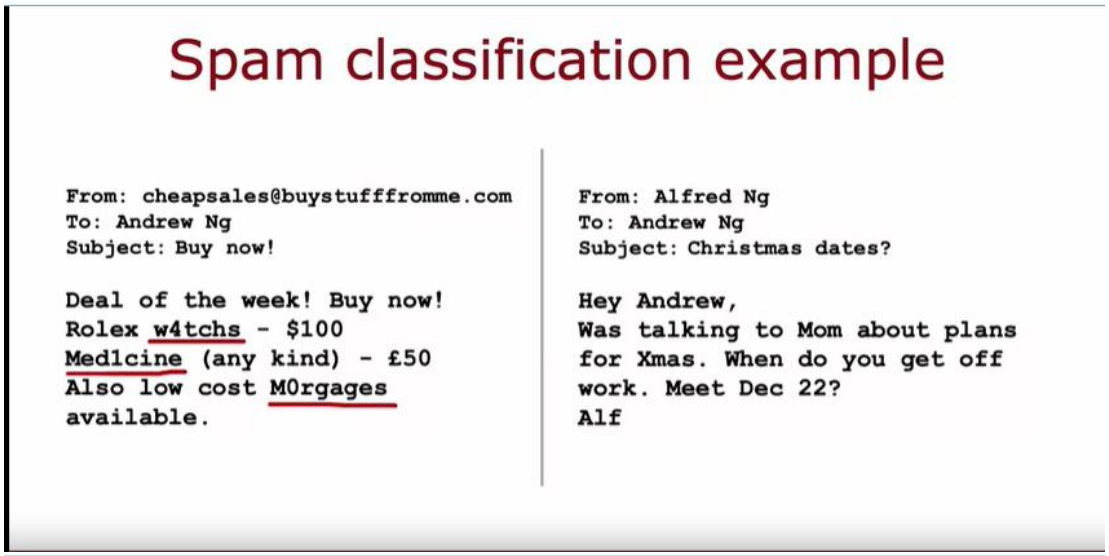


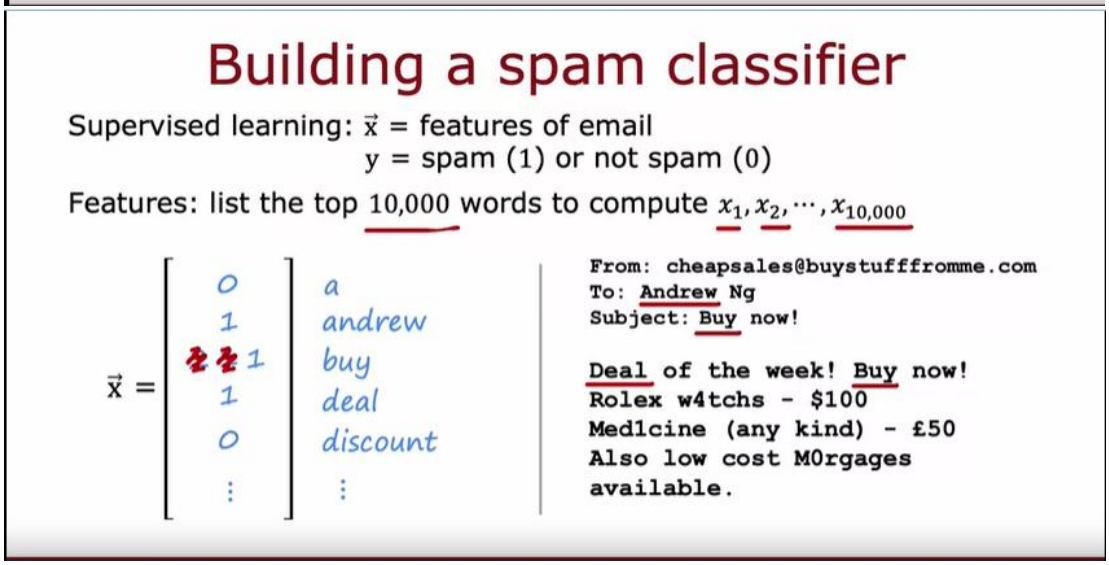




3.3 - Machine learning development process : 3.3.1 - Iterative loop of ML development :

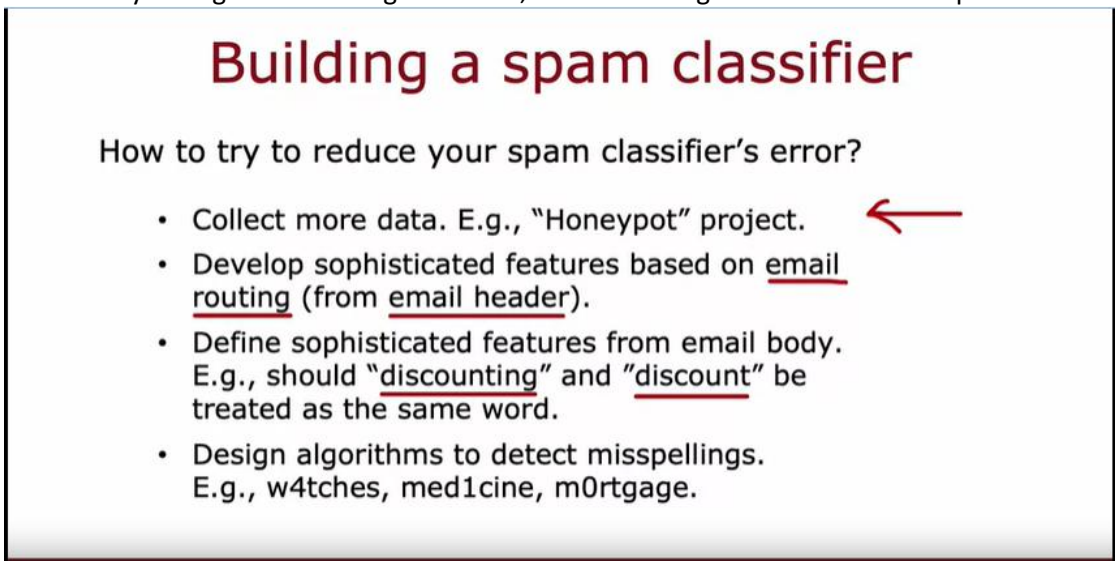


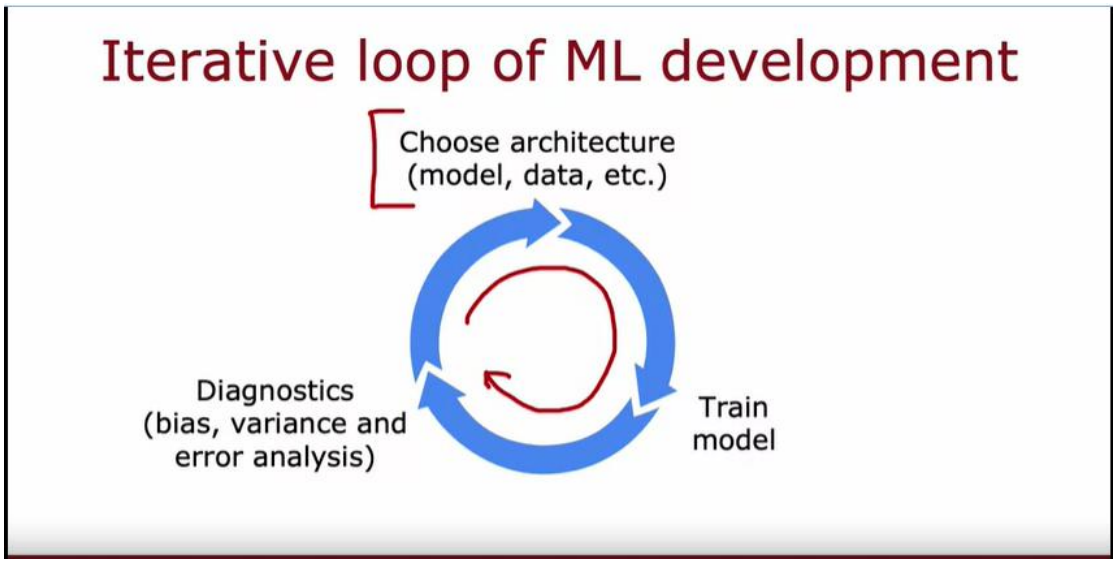




Honeypot projects: These are very large-scale projects that create a large number of fake email addresses and tries to deliberately to get these fake email addresses into the hands of spammers so that when they send spam email to these fake emails well we know these are spam email messages and so this is a way to get a lot of spam data.

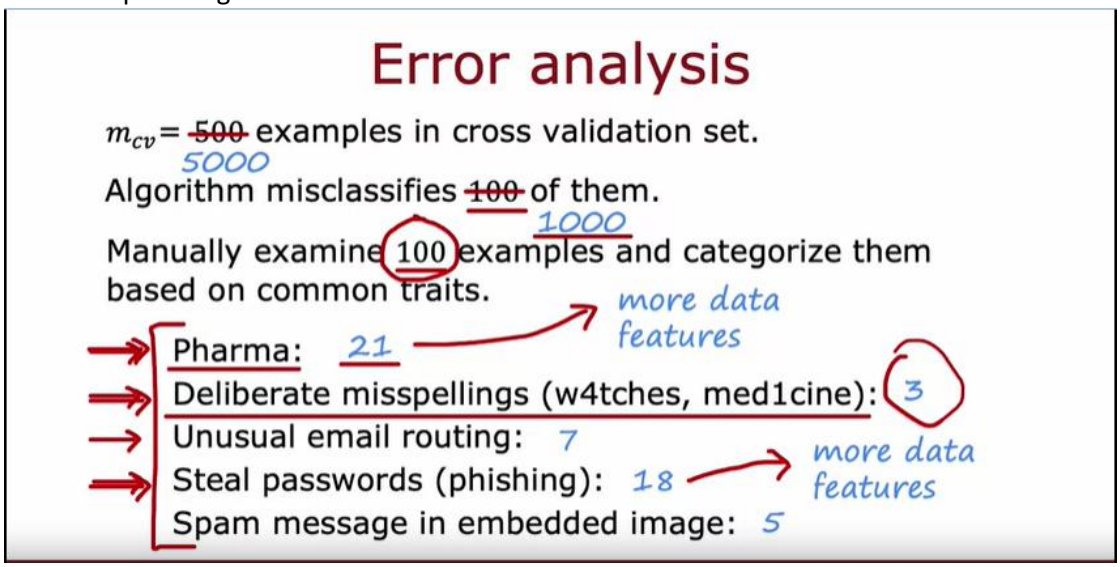
- Choosing the more promising path forward can speed up your project easily 10 times compared to if you were to somehow choose some of the less promising directions.
- For example, we've already seen that if your algorithm has high bias rather than high variance, then spending months and months on a honeypot project may not be the most fruitful direction.
- But if your algorithm has high variance, then collecting more data could help a lot.

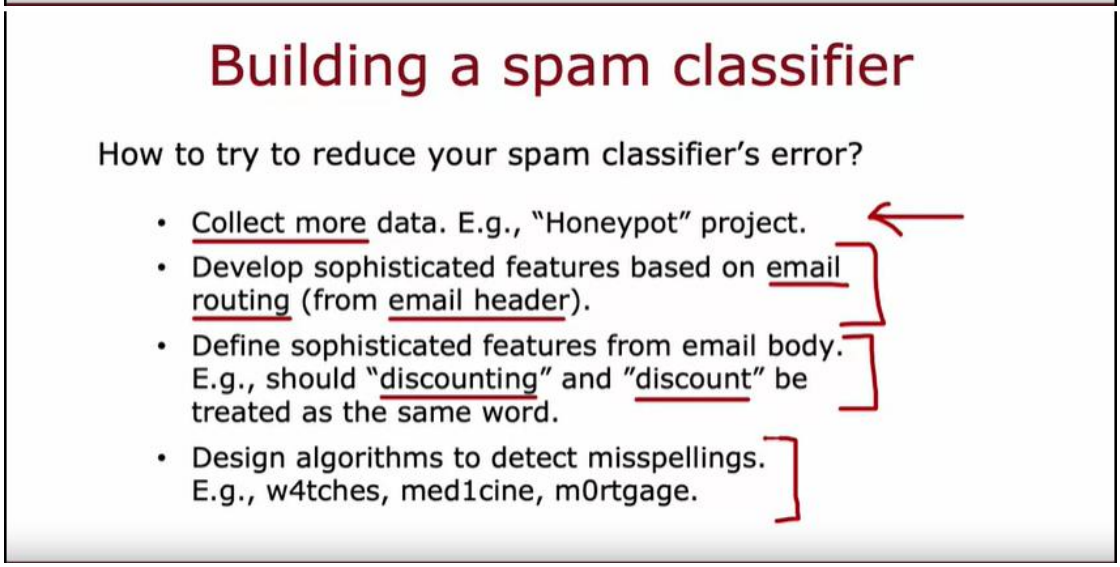


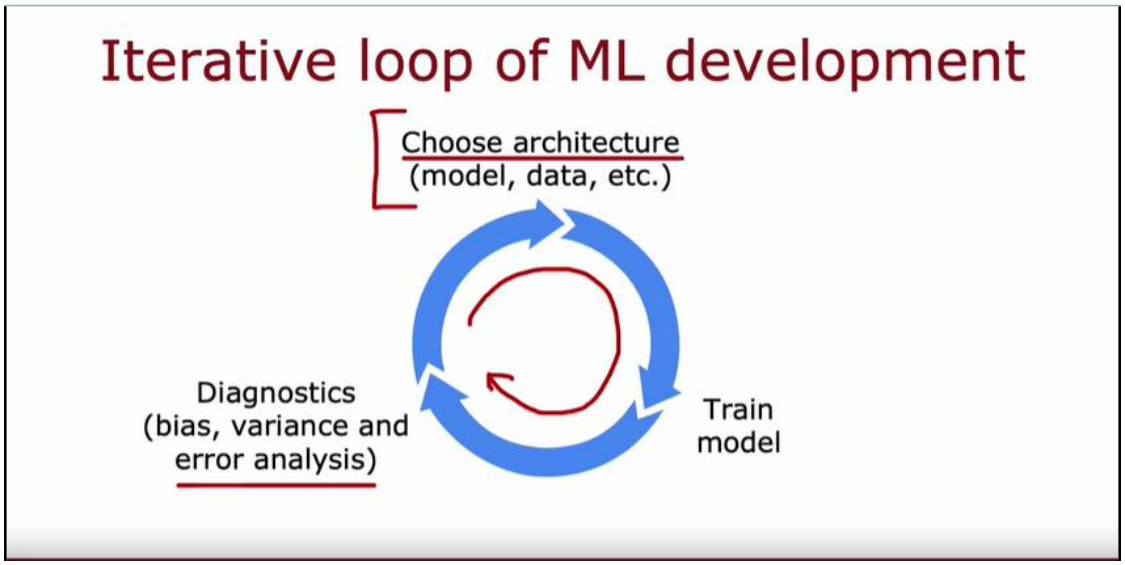


3.3 - Machine learning development process : 3.3.2 - Error analysis :

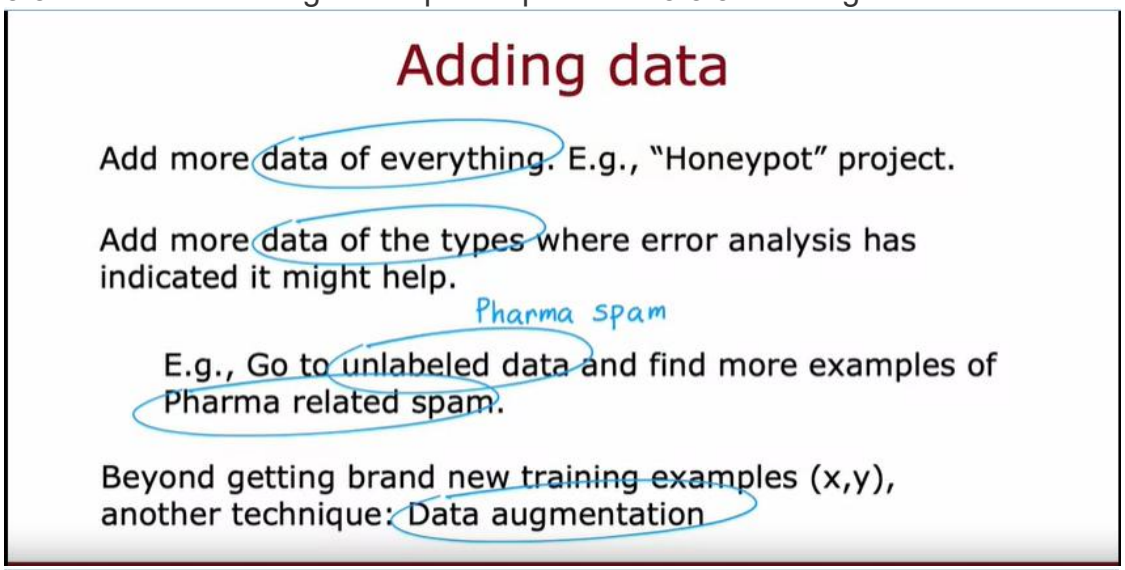
- Spam is sometimes also, instead of writing the spam message in the email body they instead create an image and then writes to spam the message inside an image that appears in the email. This makes it a little bit harder for learning algorithm to figure out what's going on. Maybe some of those emails are these embedded image spam.
- **The categories for Error Analysis** can be overlapping or in other words they're not mutually exclusive. For example, there can be a pharmaceutical spam email that also has unusual routing or a password that has deliberate misspellings and is also trying to carry out the phishing attack. One email can be counted in multiple categories.

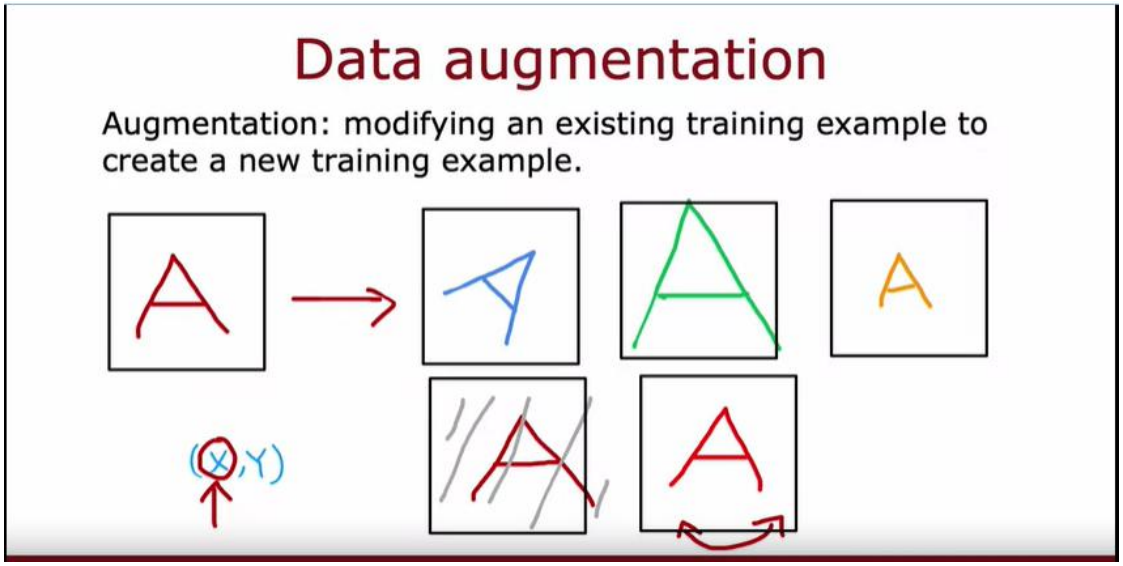




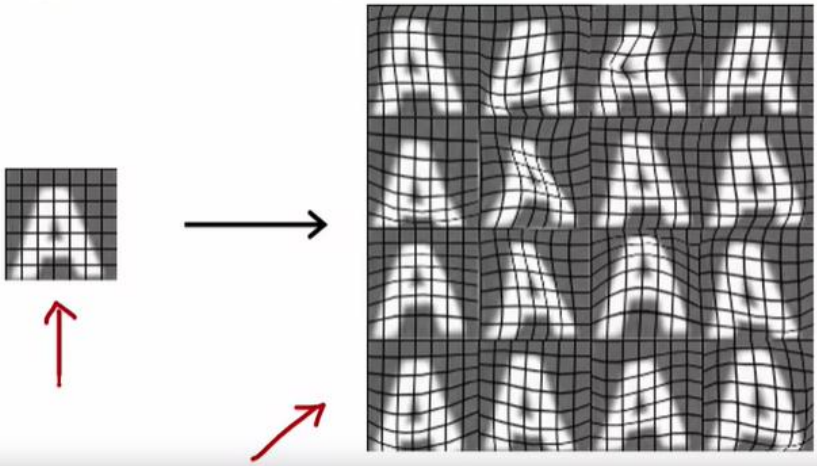


3.3 - Machine learning development process : 3.3.3 - Adding data :





Data augmentation by introducing distortions



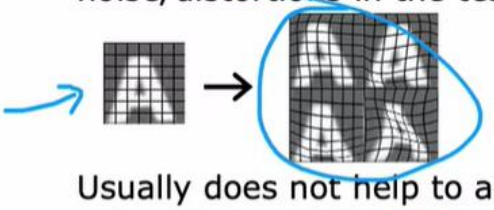
Data augmentation for speech

Speech recognition example

- Original audio (voice search: "What is today's weather?")
- + Noisy background: Crowd
- + Noisy background: Car
- + Audio on bad cellphone connection

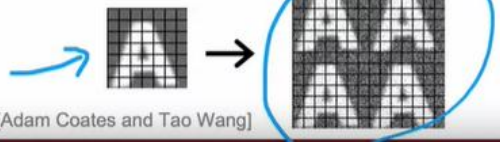
Data augmentation by introducing distortions

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.



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

[Adam Coates and Tao Wang]

Data synthesis

Synthesis: using artificial data inputs to create a new training example.

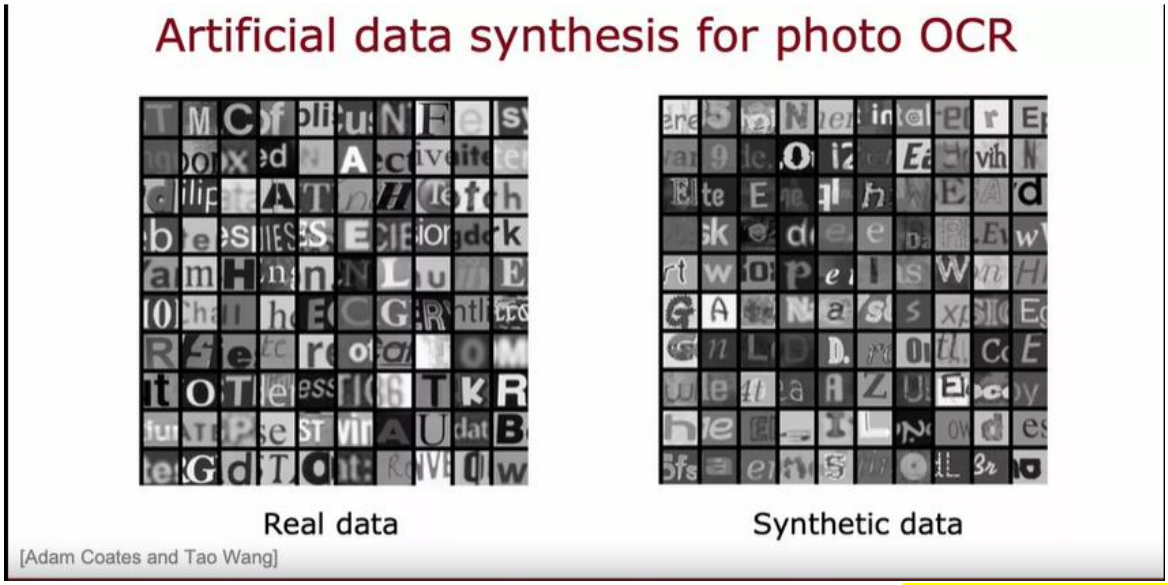
Artificial data synthesis for photo OCR

[http://www.publicdomainpictures.net/v

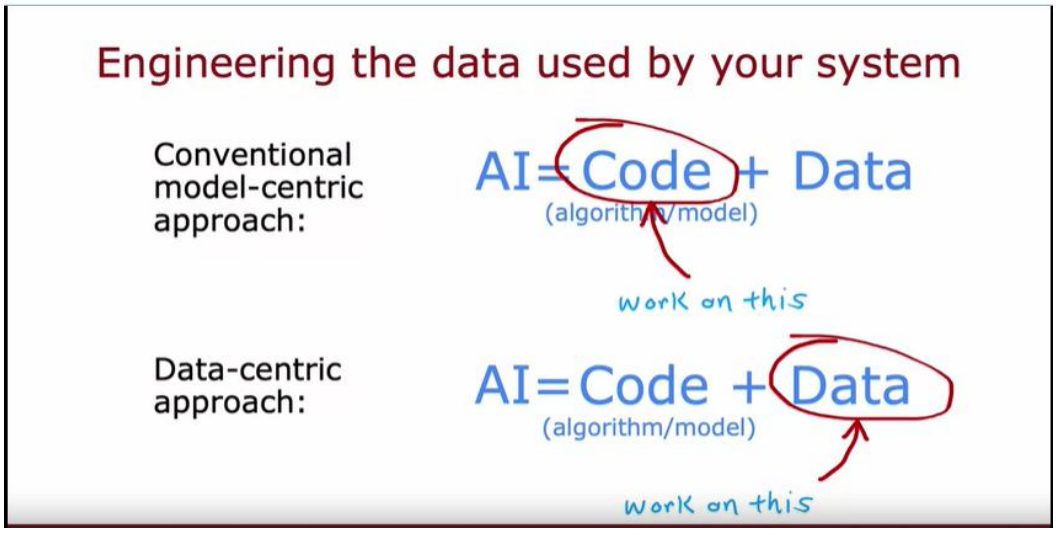
Artificial data synthesis for photo OCR

Real data

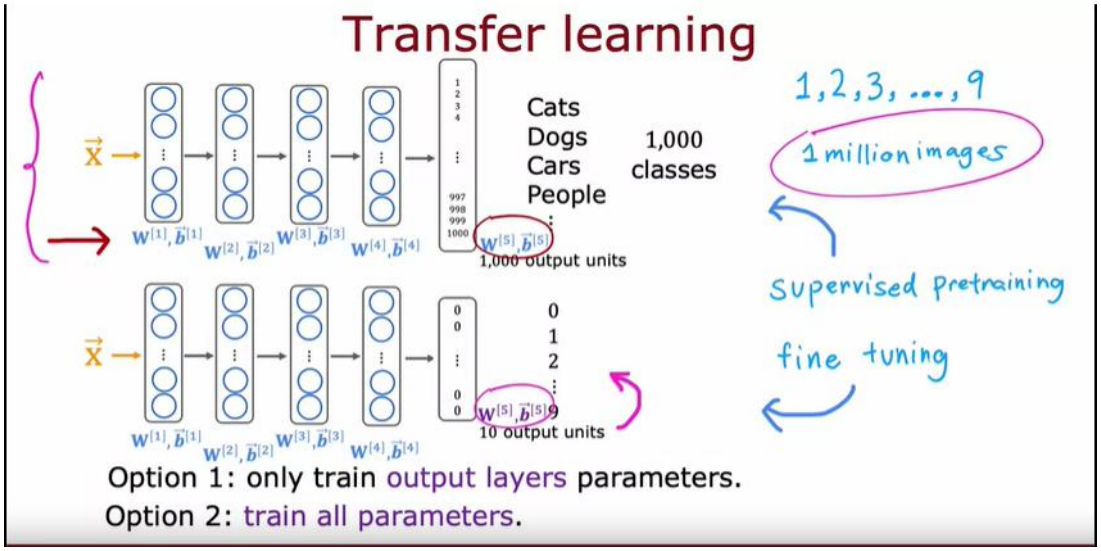
[Adam Coates and Tao Wang]



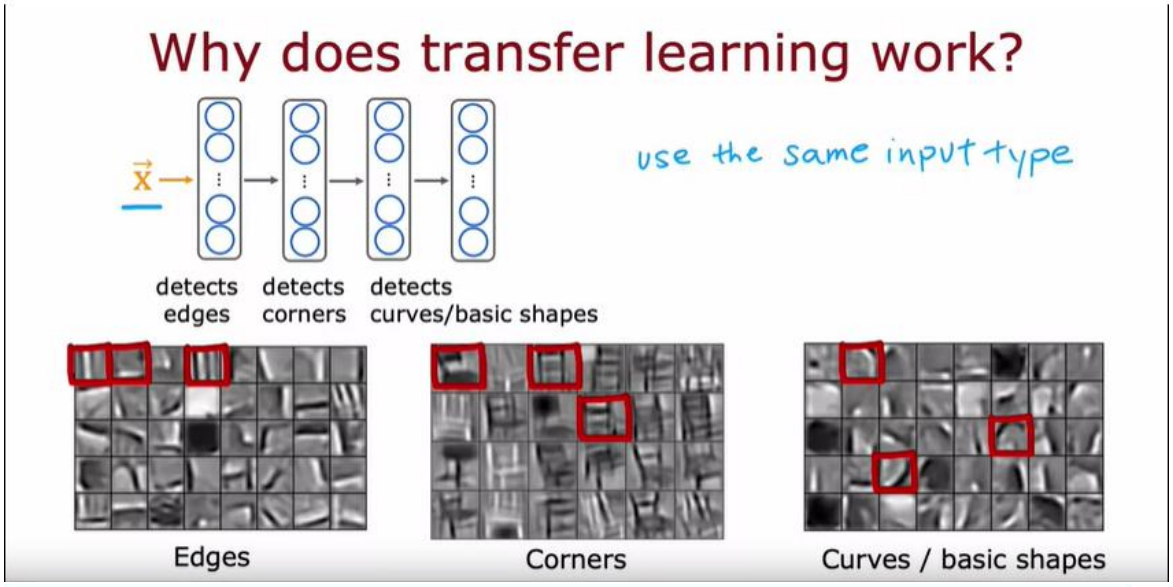
- Sometimes it can be more fruitful to spend more of your time taking a **data centric approach** in which you focus on engineering the data used by your algorithm.
- And this can be anything from collecting more data just on pharmaceutical spam. If that's what error analysis told you to do. To using data augmentation to generate more images or more audio or using data synthesis to just create more training examples.
- And sometimes that focus on the data can be an efficient way to help your learning algorithm improve its performance

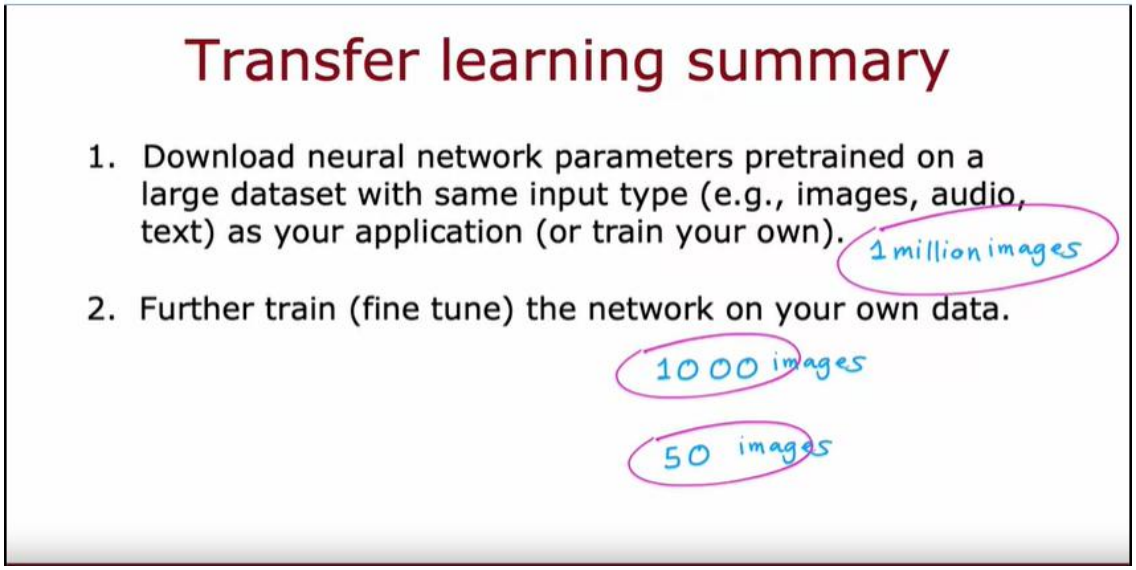


3.3 - Machine learning development process : 3.3.4 - Transfer learning: using data from a different task :



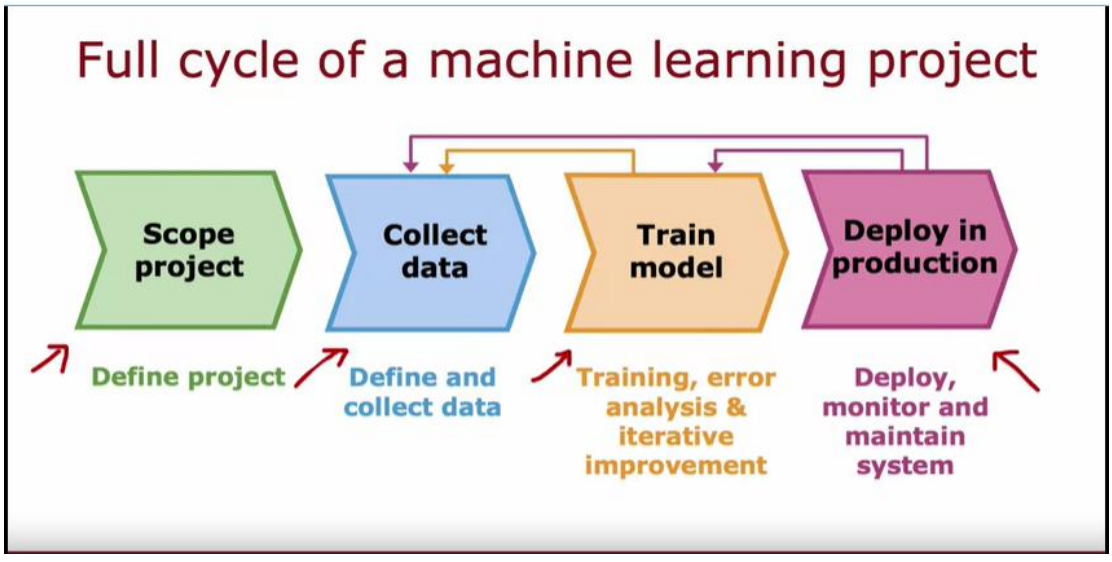
- If the final task you want to solve is a computer vision tasks, then the pre-training step also has been a neural network trained on the same type of input, namely an image of the desired dimensions.
- Conversely, if your goal is to build a speech recognition system to process audio, then a neural network pre-trained on images probably won't do much good on audio. Instead, you want a neural network pre-trained on audio data, there you then fine tune on your own audio dataset and the same for other types of applications.

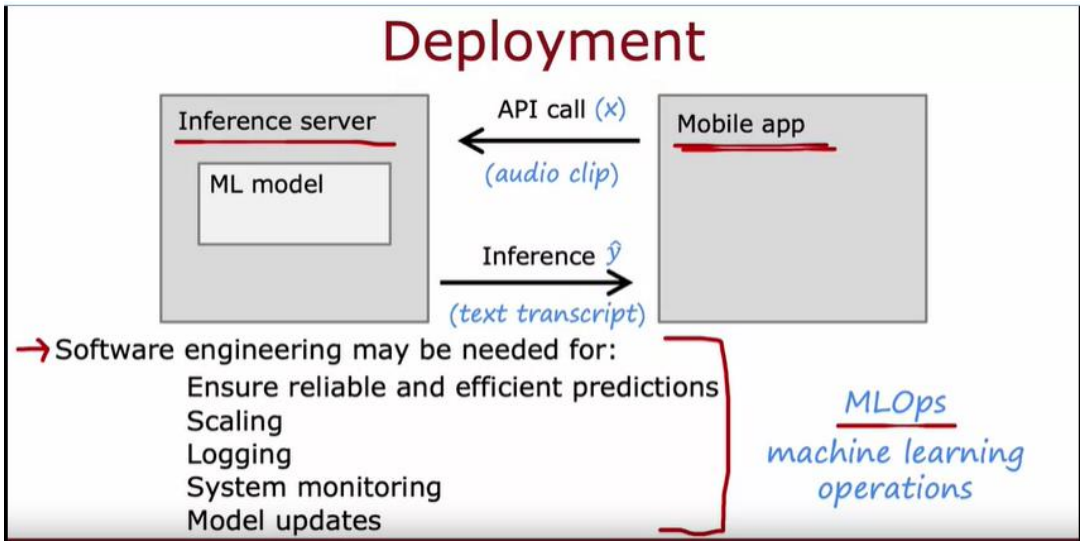




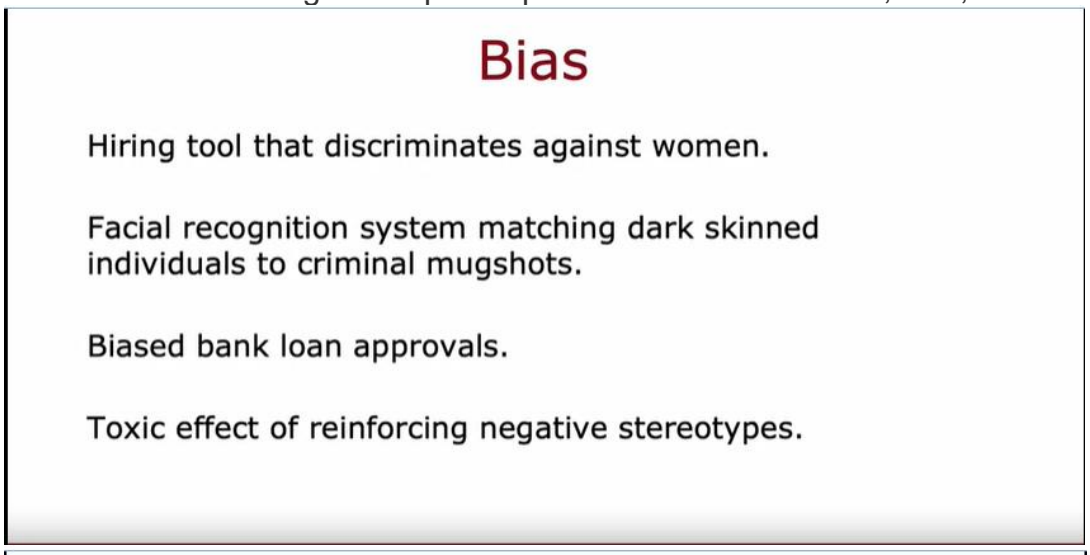
3.3 - Machine learning development process : 3.3.5 - Full cycle of a machine learning project

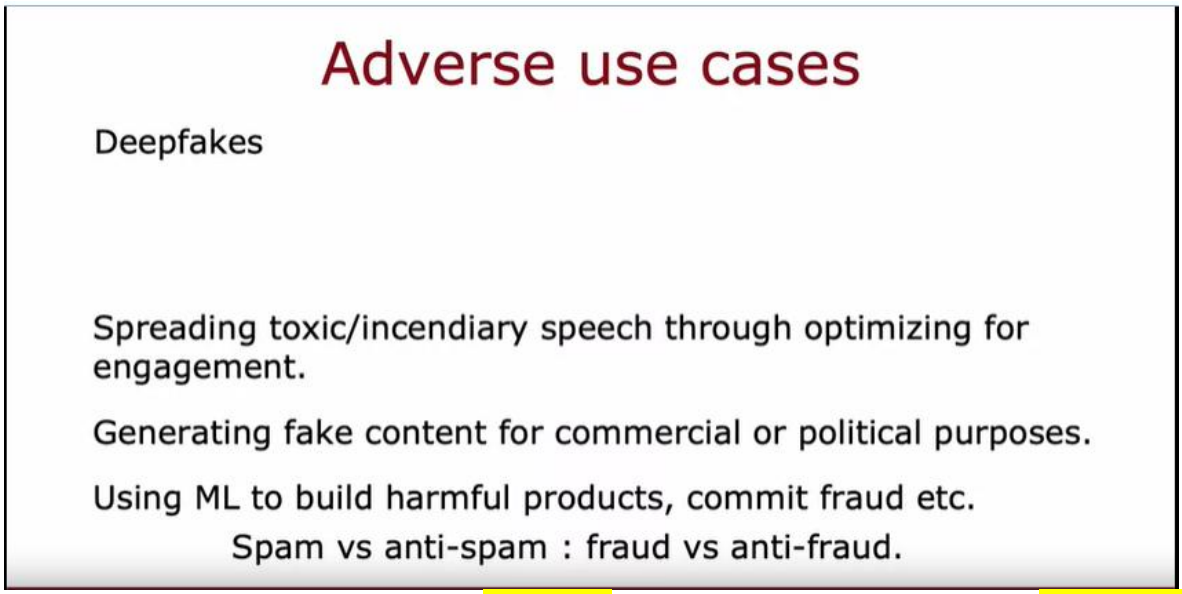
- Project Scope : Eg. speech recognition for voice search



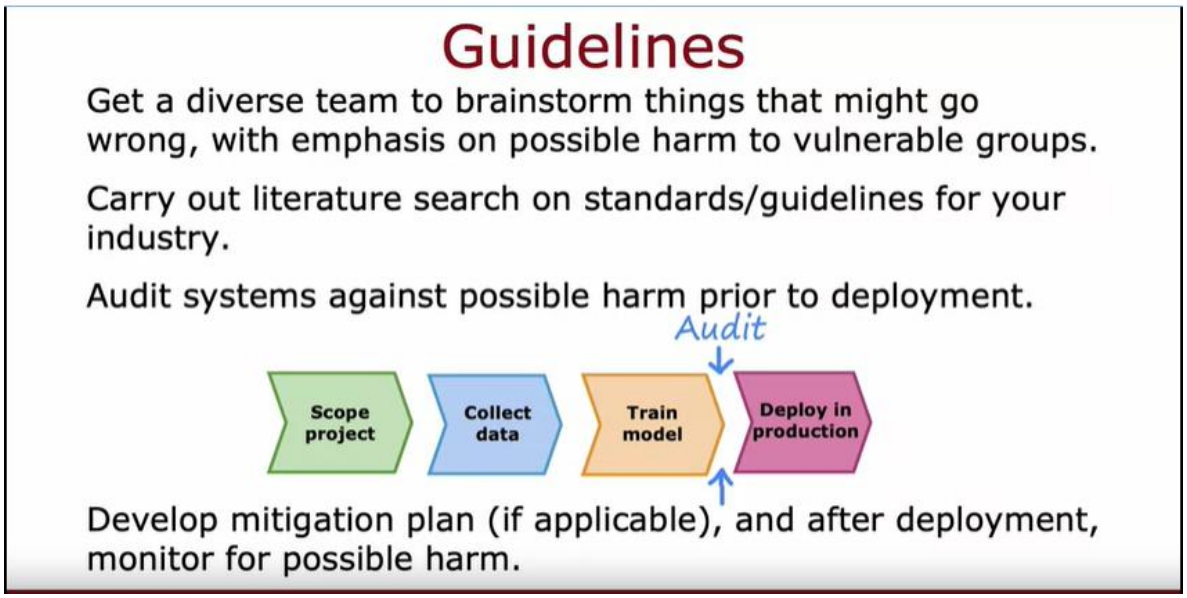


3.3 - Machine learning development process : 3.3.6 - Fairness, bias, and ethics :

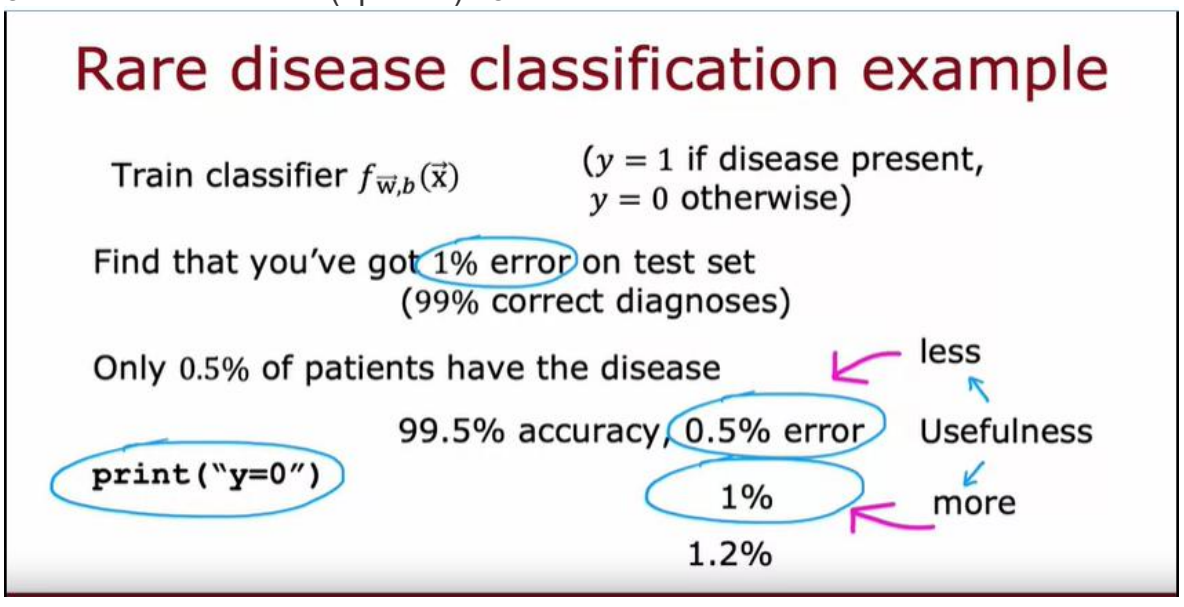


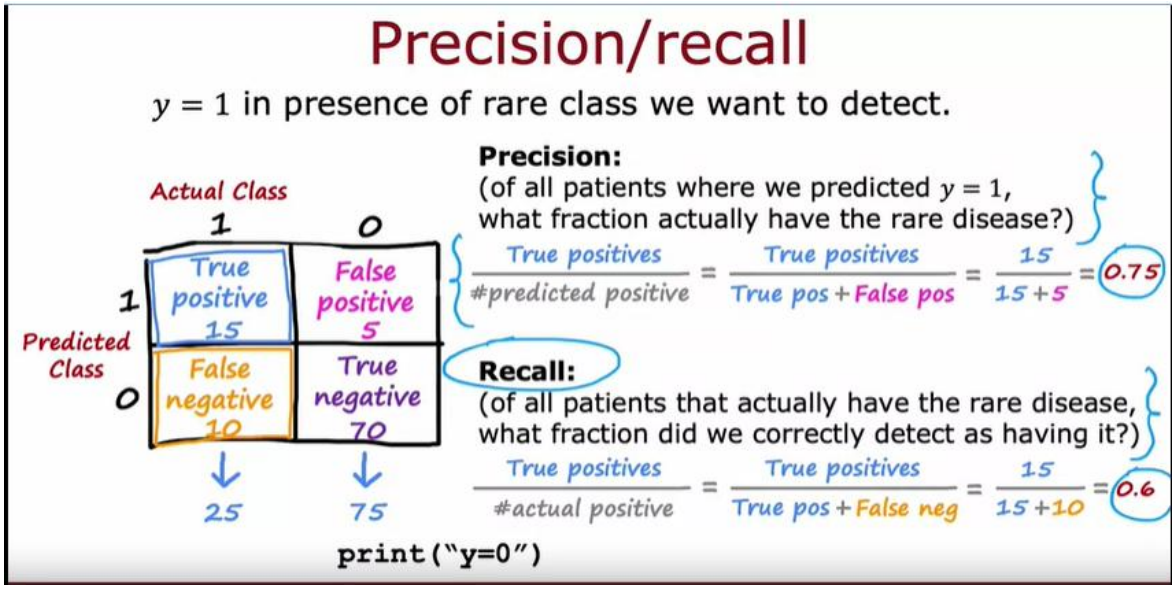


- Even after deployment to continue to monitor harm so that you can then trigger a mitigation plan and act quickly in case there is a problem that needs to be addressed.
- For example, all of the self driving car teams prior to rolling out self driving cars on the road had developed mitigation plans for what to do in case the car ever gets involved in an accident so that if the car was ever in an accident, there was already a mitigation plan that they could execute immediately rather than have a car got into an accident and then only scramble after the fact to figure out what to do.



3.4 - Skewed datasets (optional) : 3.4.1 - Error metrics for skewed datasets :

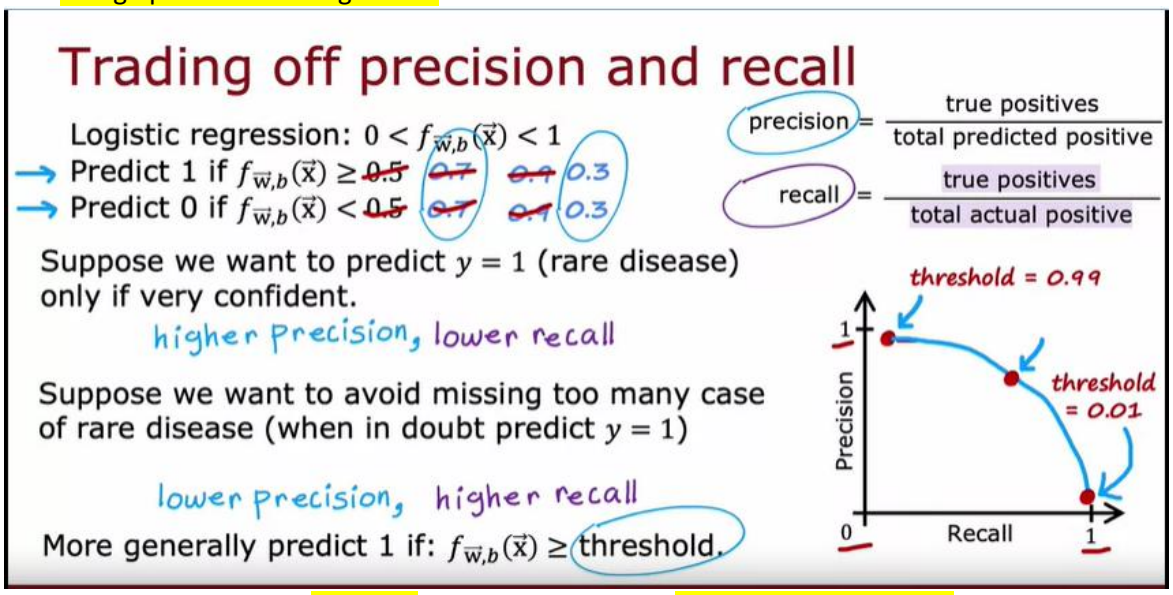




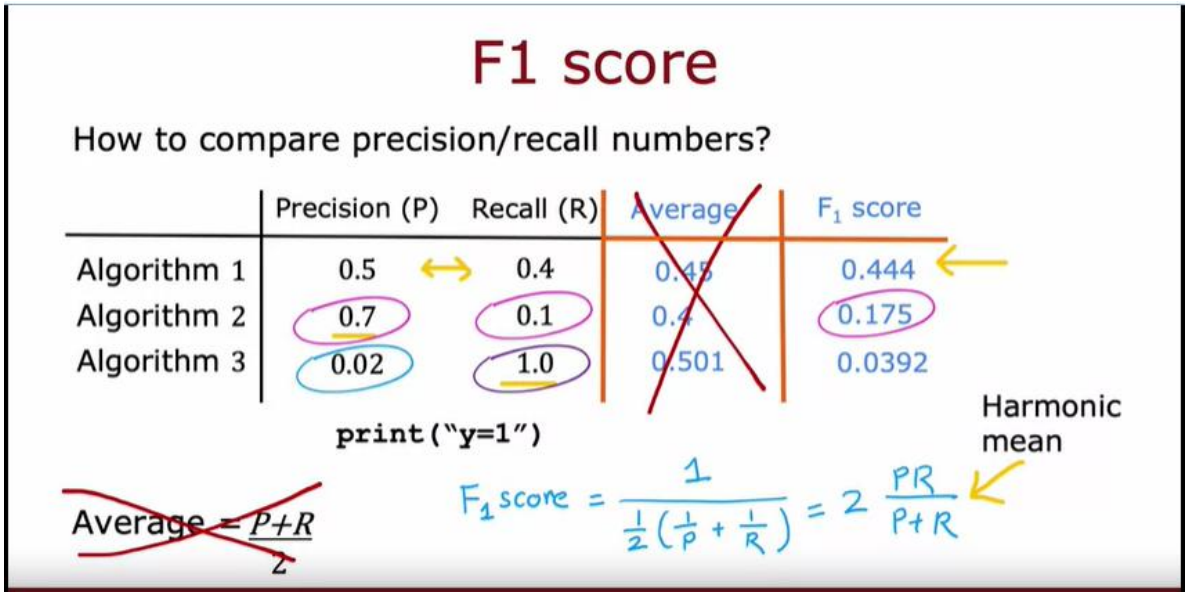
- When you have a rare class, looking at precision and recall and making sure that both numbers are decently high, that hopefully helps reassure you that your learning algorithm is actually useful.

3.4 - Skewed datasets (optional) : 3.4.2 - Trading off precision and recall :

- In the ideal case, we like for learning algorithms that have high precision and high recall.
- High precision would mean that if a diagnosis of patients have that rare disease, probably the patient does have it and it's an accurate diagnosis.
- High recall means that if there's a patient with that rare disease, probably the algorithm will correctly identify that they do have that disease.
- But it turns out that in practice there's often a trade-off between precision and recall.
- By plotting the Precision-Recall curve, you can then try to pick a threshold which corresponds to picking a point on this curve that balances, the cost of false positives and false negatives or that balances, the benefits of high precision and high recall.



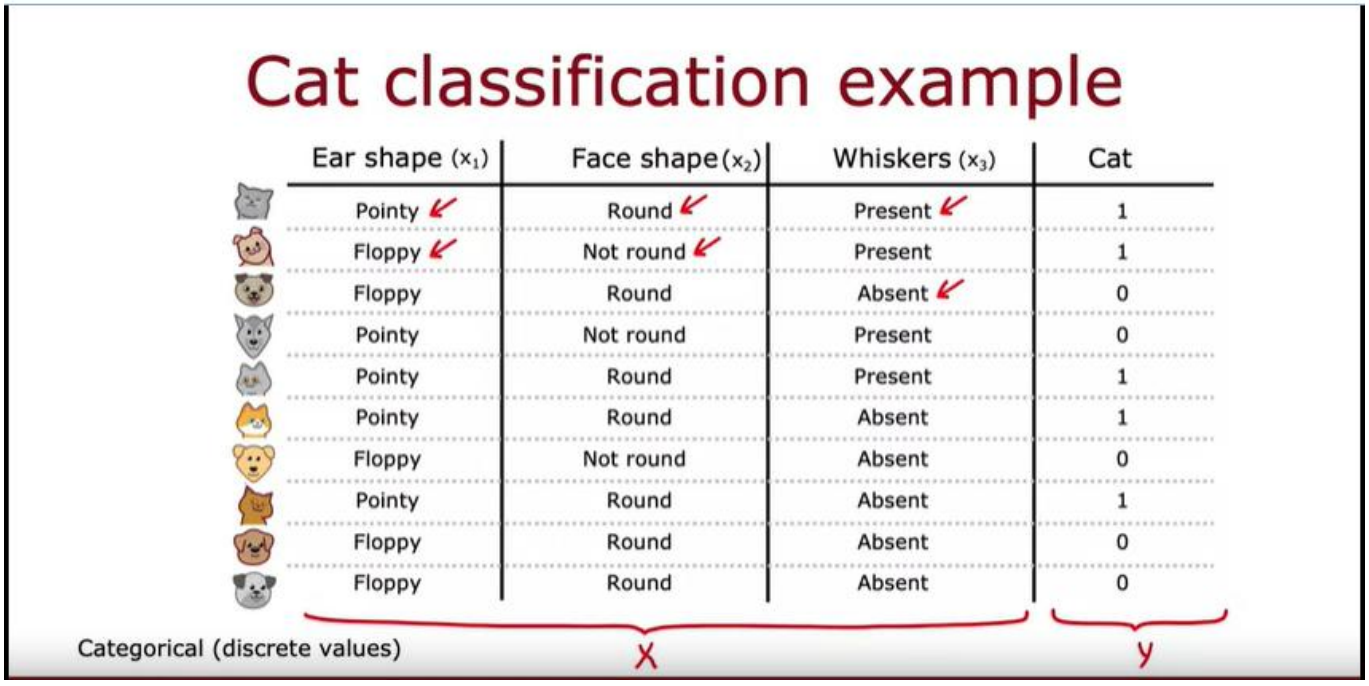
- The metric called the F1 score is sometimes used to automatically combine precision recall to help you pick the best value or the best trade-off between the two.
- F1 score is a way of combining P and R precision and recall but that gives more emphasis to whichever of these values is lower.

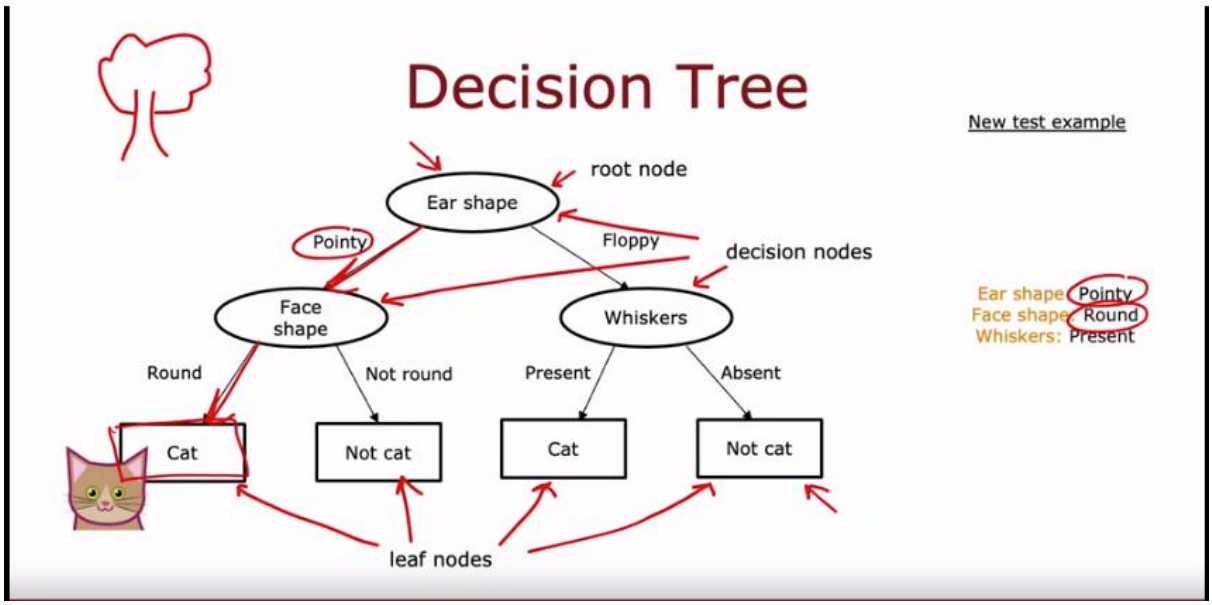

F1 score
How to compare precision/recall numbers?
Precision (P)
Recall (R)
Average
F1 score
Algorithm 1
0.5
0.4
0.45
0.444
Algorithm 2
0.7
0.1
0.4
0.175
Algorithm 3
0.02
1.0
0.501
0.0392
print("y=1")
Harmonic mean
Average = P+R / 2
F1 score = 1 / (1/2 (1/P + 1/R)) = 2 PR / P+R

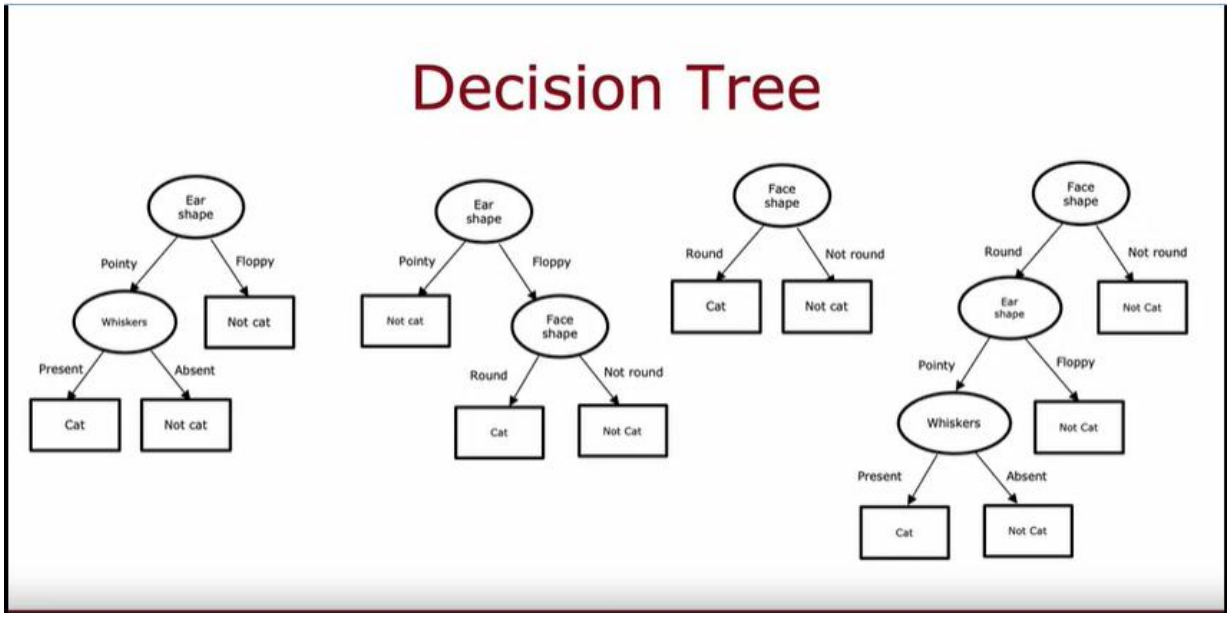
Part (Week) 4 : Decision trees

4.3 - Tree ensembles : 4.3.1 - Using multiple decision trees :

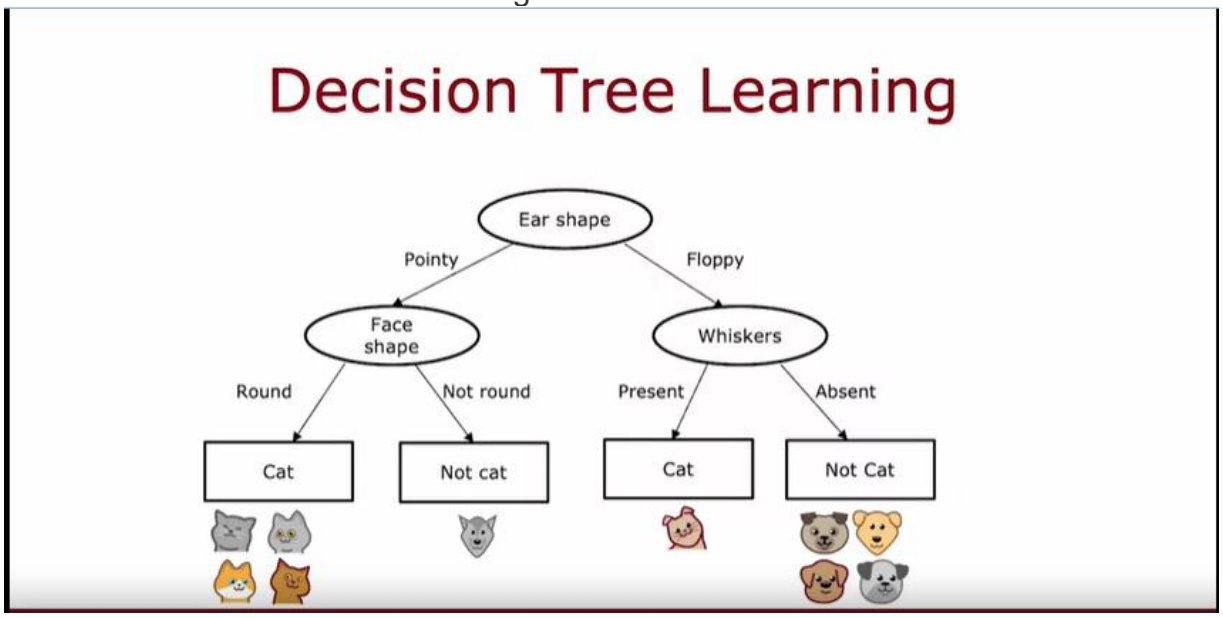
4.1 - Decision trees : 4.1.1 - Decision tree model :

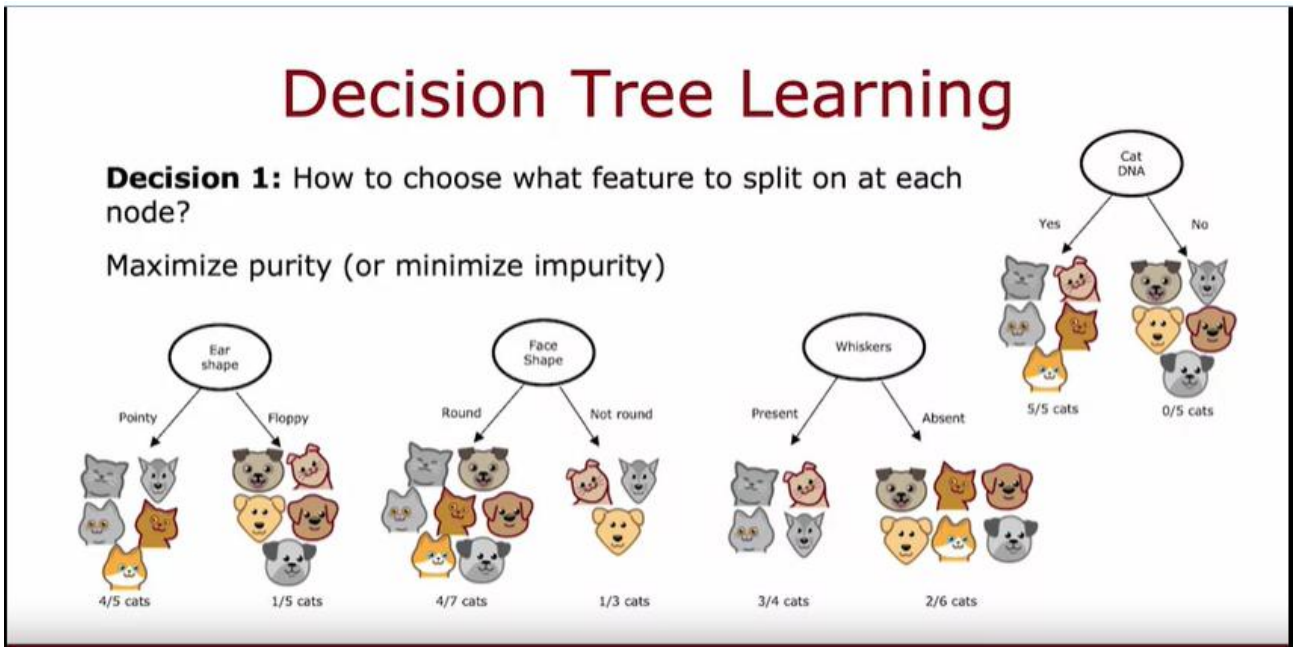




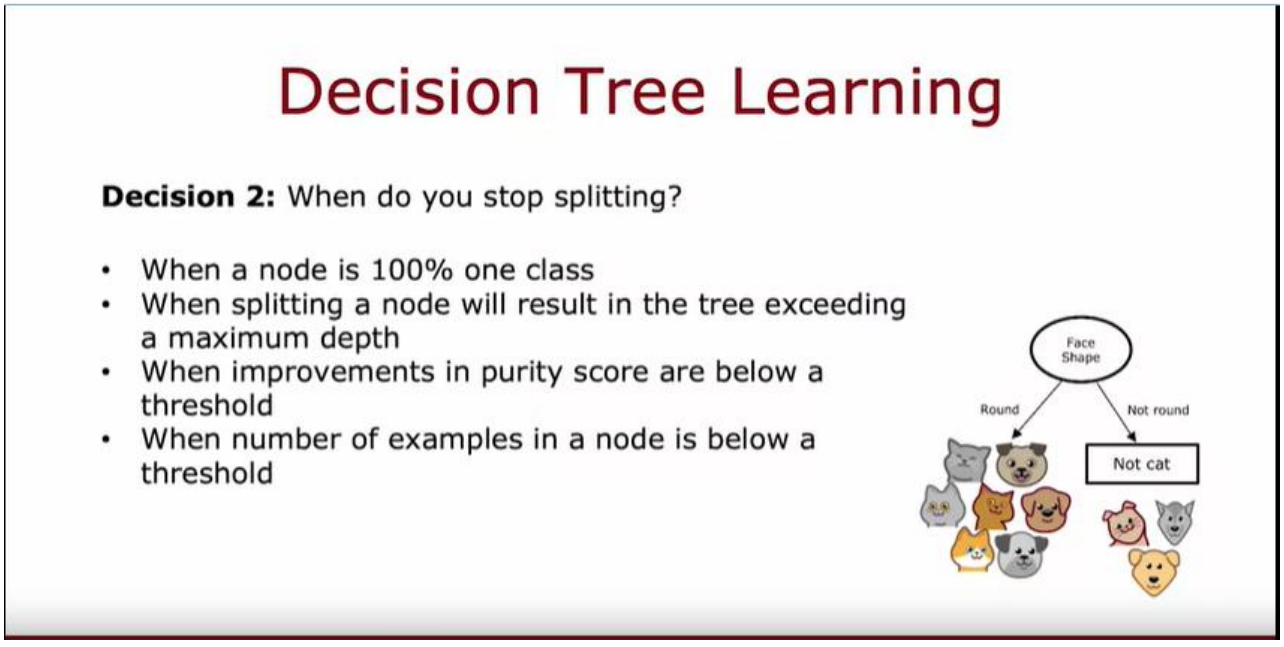


4.1 - Decision trees : 4.1.2 - Learning Process :

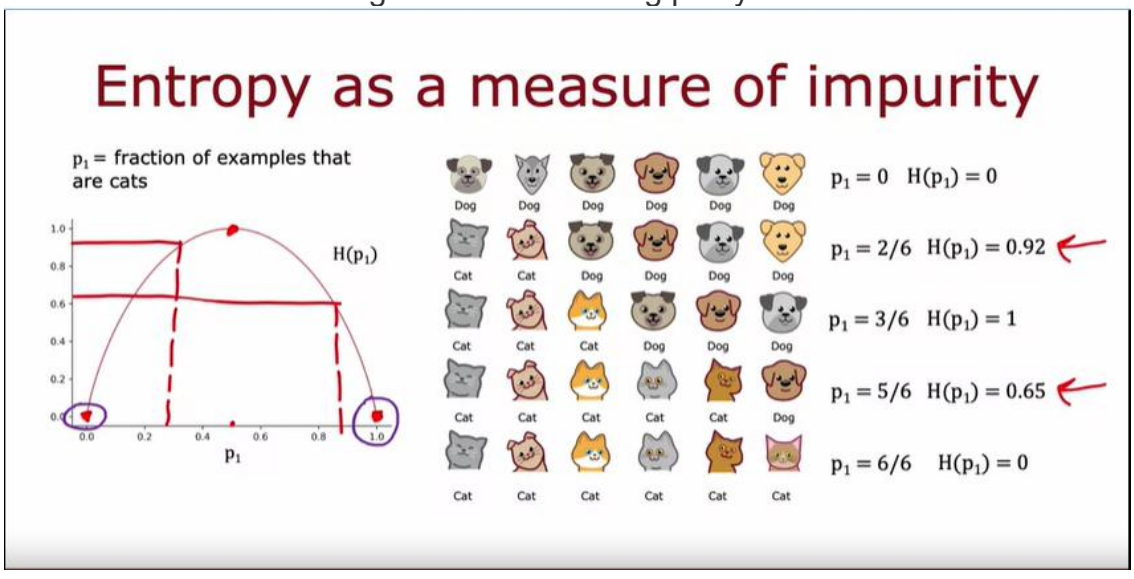


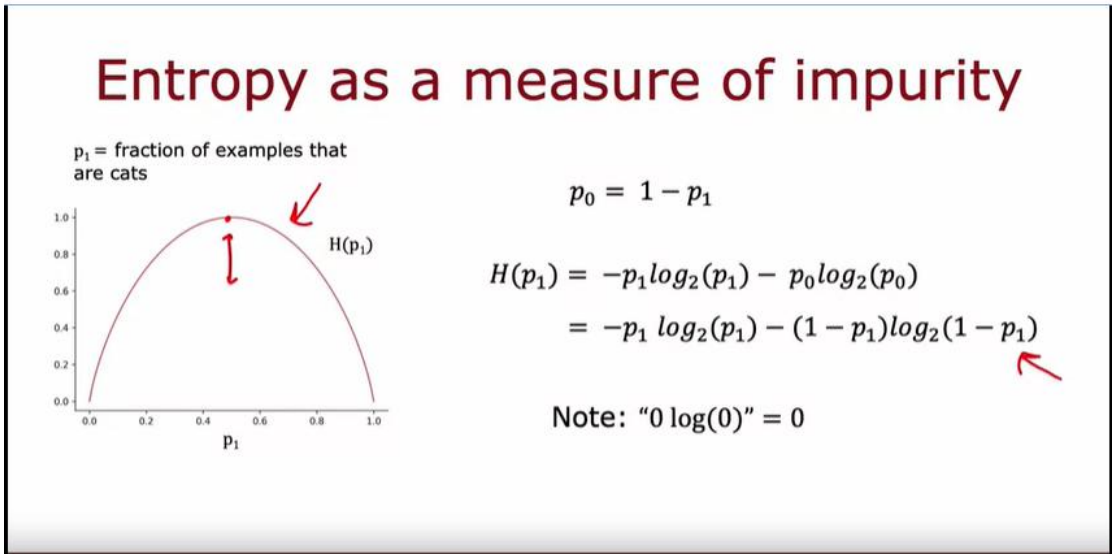


- One reason you might want to limit the depth of the decision tree is to make sure for us to tree doesn't get too big and unwieldy and second, by keeping the tree small, it makes it less prone to overfitting.



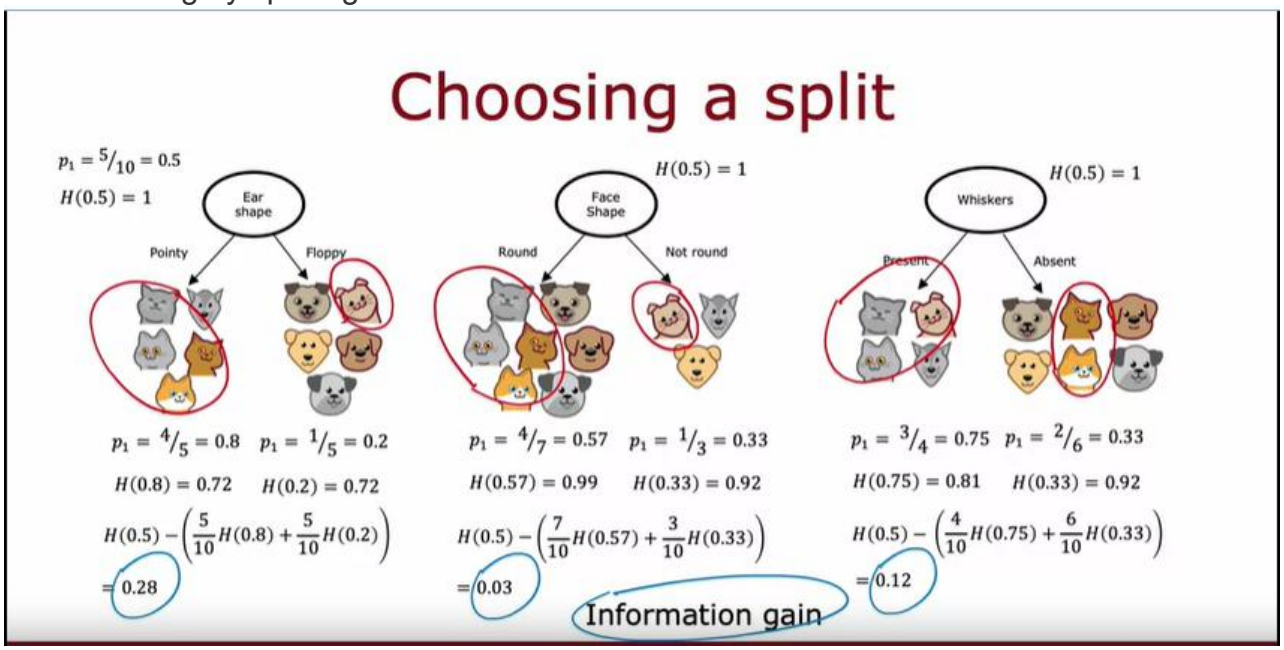
4.2 - Decision tree learning : 4.2.1 - Measuring purity :





4.2 - Decision tree learning : 4.2.2 - Choosing a split :

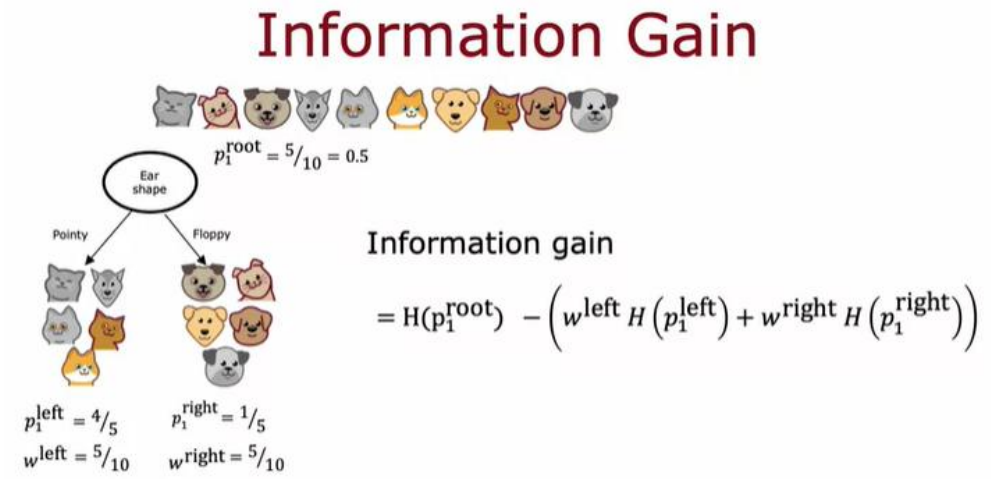
- **Information gain** measures the reduction in entropy that you get in your tree resulting from making a split.
- Avoid to carry out the split any further if the reduction in entropy is too small or below a threshold. Otherwise, you're just increasing the size of the tree unnecessarily and risking overfitting by splitting.



- weightage

w_lef= 5/10 w_rig=5/10 w_lef= 7/10 w_rig=3/10 w_lef= 4/10 w_rig=6/10

- pick the one that gives you the highest information gain

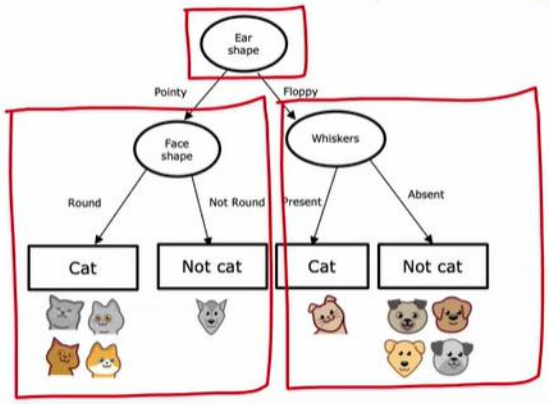


4.2 - Decision tree learning : 4.2.3 - Putting it together :

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

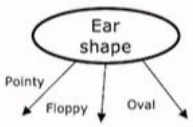


Recursive algorithm

4.2 - Decision tree learning : 4.2.4 - Using one-hot encoding of categorical features :

Features with three possible values

	Ear shape (x_1)	Face shape (x_2)	Whiskers (x_3)	Cat (y)
	Pointy	Round	Present	1
	Oval	Not round	Present	1
	Oval	Round	Absent	0
	Pointy	Not round	Present	0
	Oval	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Oval	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0



3 possible values

One hot encoding

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

One hot encoding

If a categorical feature can take on k values, create k binary features (0 or 1 valued).

One hot encoding and neural networks

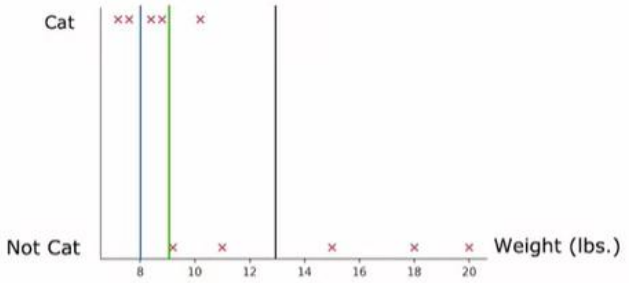
Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
1	0	0	~~Round~~ 1	~~Present~~ 1	1
0	0	1	~~Not round~~ 0	~~Present~~ 1	1
0	0	1	~~Round~~ 1	~~Absent~~ 0	0
1	0	0	~~Not round~~ 0	~~Present~~ 1	0
0	0	1	~~Round~~ 1	~~Present~~ 1	1
1	0	0	~~Round~~ 1	~~Absent~~ 0	1
0	1	0	~~Not round~~ 0	~~Absent~~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

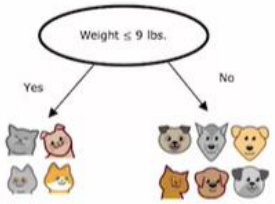
4.2 - Decision tree learning : 4.2.5 - Continuous valued features :

Continuous features

Ear shape	Face shape	Whiskers	Weight (lbs.)	Cat
Pointy	Round	Present	7.2	1
Floppy	Not round	Present	8.8	1
Floppy	Round	Absent	15	0
Pointy	Not round	Present	9.2	0
Pointy	Round	Present	8.4	1
Pointy	Round	Absent	7.6	1
Floppy	Not round	Absent	11	0
Pointy	Round	Absent	10.2	1
Floppy	Round	Absent	18	0
Floppy	Round	Absent	20	0

Splitting on a continuous variable





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

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

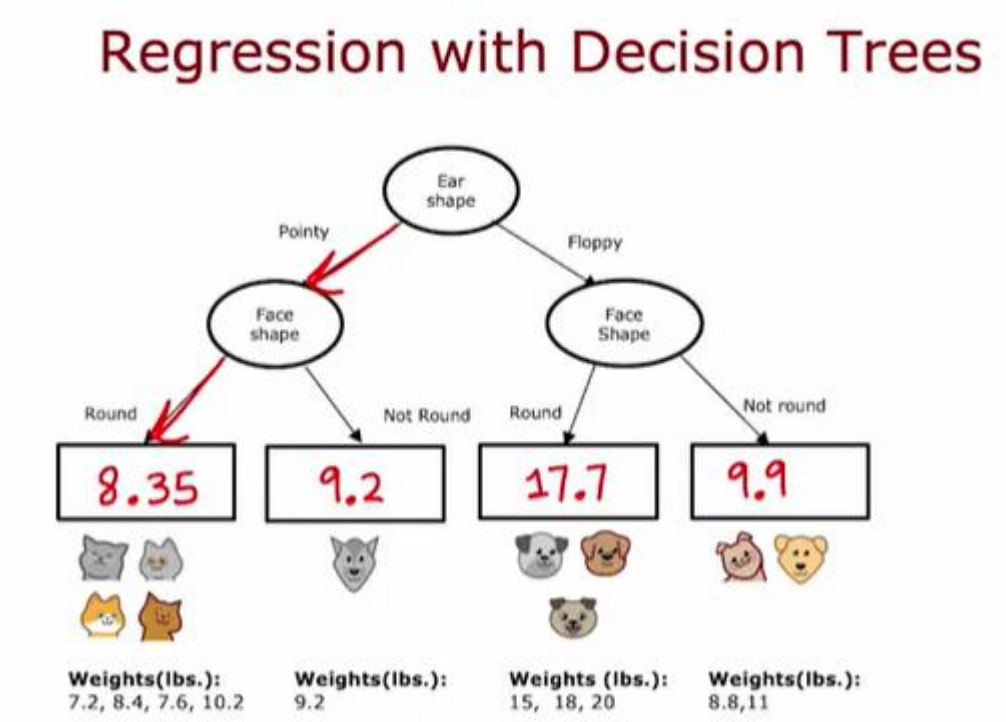
$$H(0.5) - \left(\frac{7}{10}H\left(\frac{5}{7}\right) + \frac{3}{10}H\left(\frac{0}{3}\right)\right) = 0.40$$

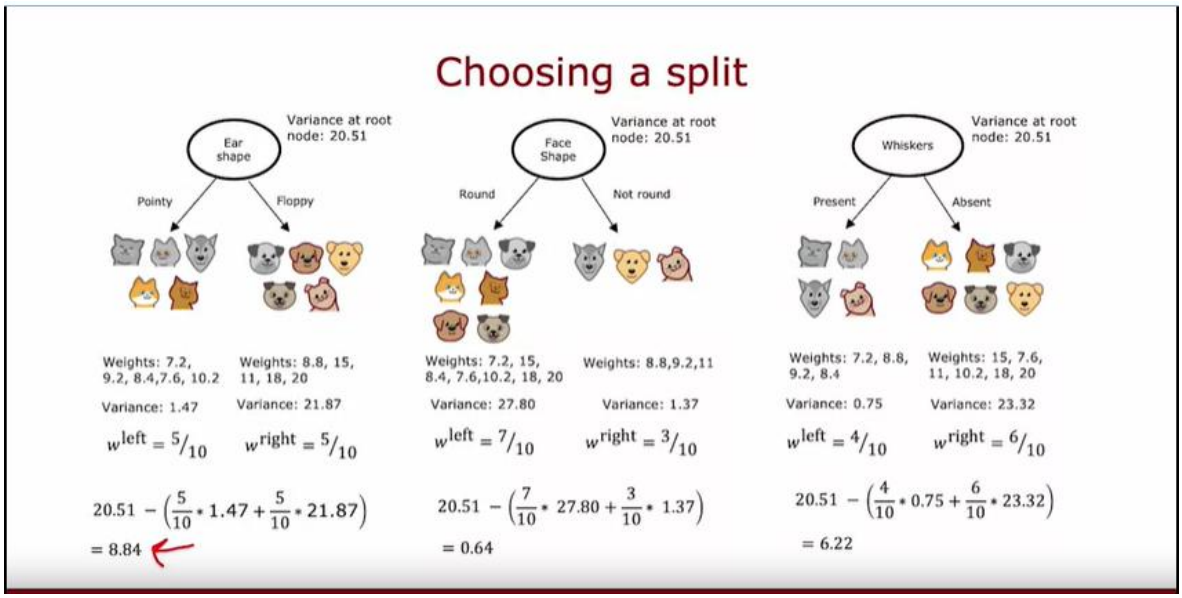
4.2 - Decision tree learning : 4.2.6 - Regression Trees (optional) :

Regression with Decision Trees: Predicting a number

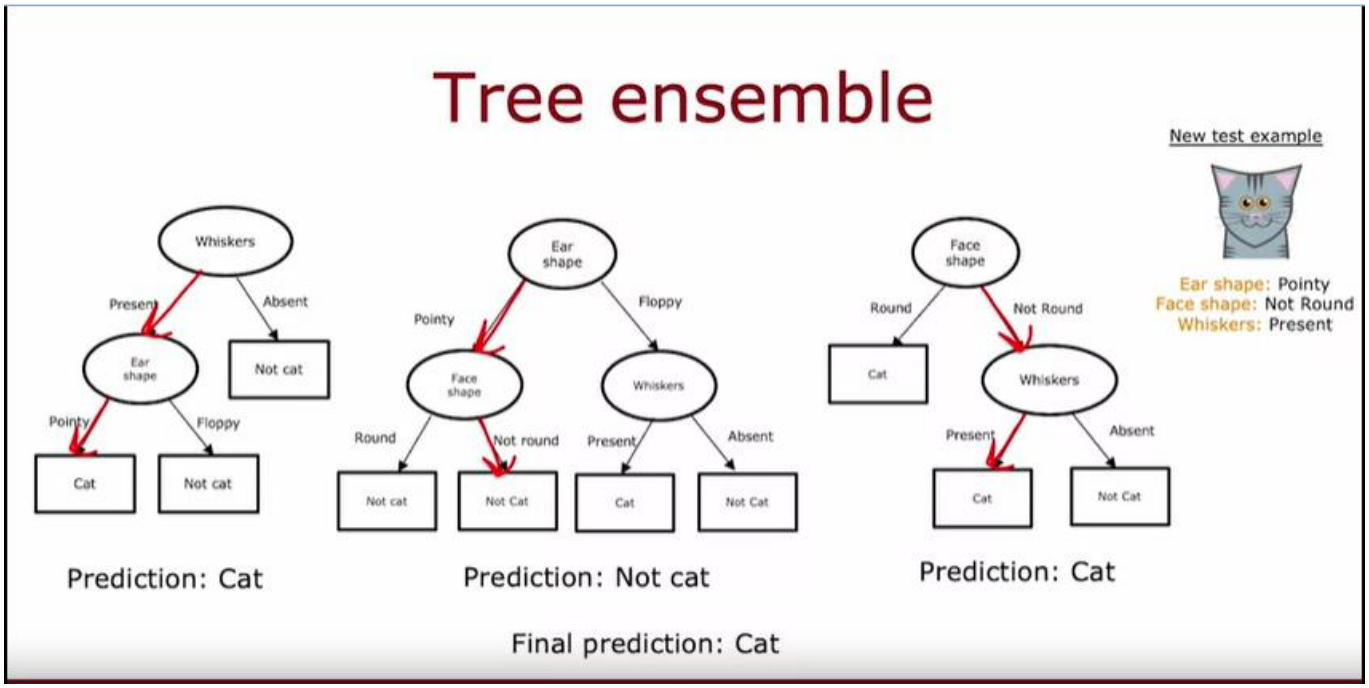
Ear shape	Face shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not round	Present	8.8
Floppy	Round	Absent	15
Pointy	Not round	Present	9.2
Pointy	Round	Present	8.4
Pointy	Round	Absent	7.6
Floppy	Not round	Absent	11
Pointy	Round	Absent	10.2
Floppy	Round	Absent	18
Floppy	Round	Absent	20

X y





4.3 - Tree ensembles : 4.3.1 - Using multiple decision trees :


Tree ensemble
New test example
Ear shape: Pointy
Face shape: Not Round
Whiskers: Present
Whiskers
Present
Absent
Ear shape
Not cat
Pointy
Floppy
Cat
Not cat
Prediction: Cat
Ear shape
Pointy
Floppy
Face shape
Whiskers
Round
Not round
Present
Absent
Not cat
Not Cat
Cat
Not Cat
Prediction: Not cat
Face shape
Round
Not Round
Cat
Whiskers
Present
Absent
Cat
Not Cat
Prediction: Cat
Final prediction: Cat