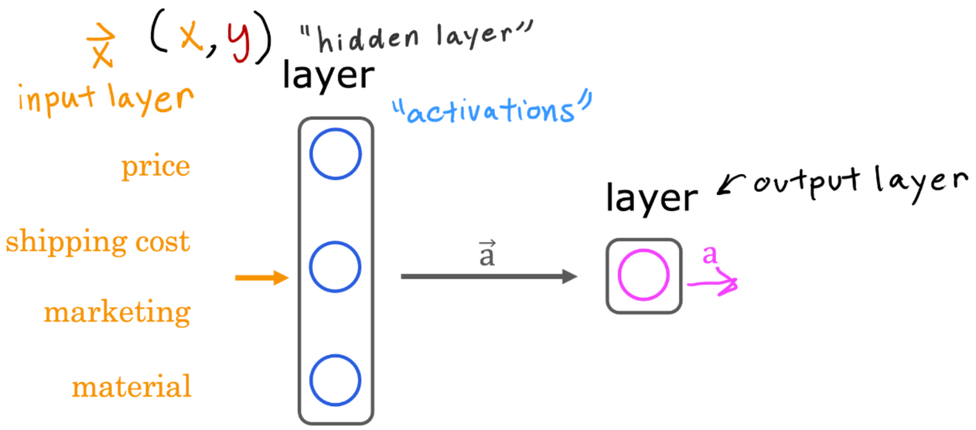
1. 1 point



Which of these are terms used to refer to components of an artificial neural network? (hint: three of these are correct)

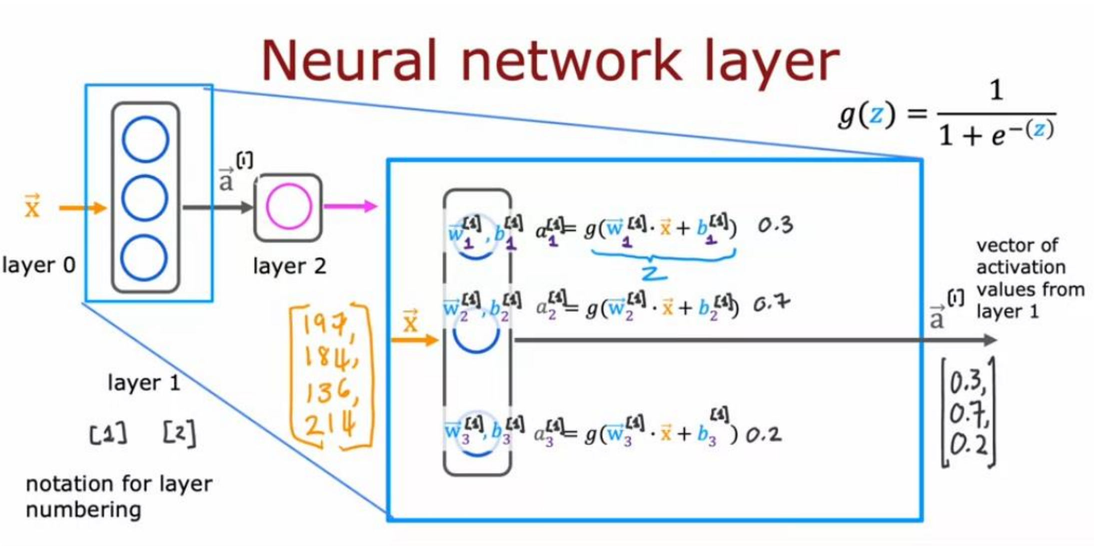
- [x] neurons
- [x] layers
- [ ] axon
- [x] activation function

2. True/False? Neural networks take inspiration from, but do not very accurately mimic, how neurons in a biological brain learn. 1 point

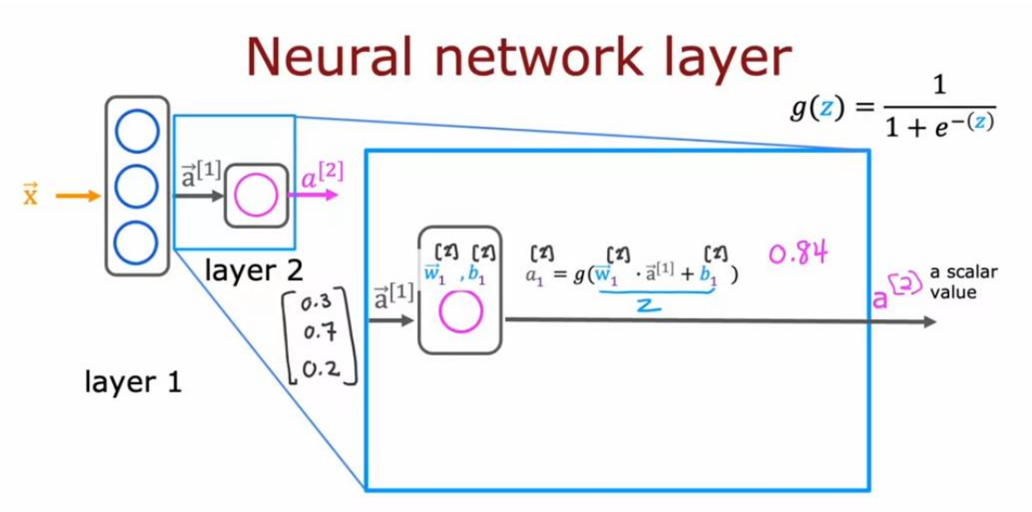
- (●) True
- ( ) False

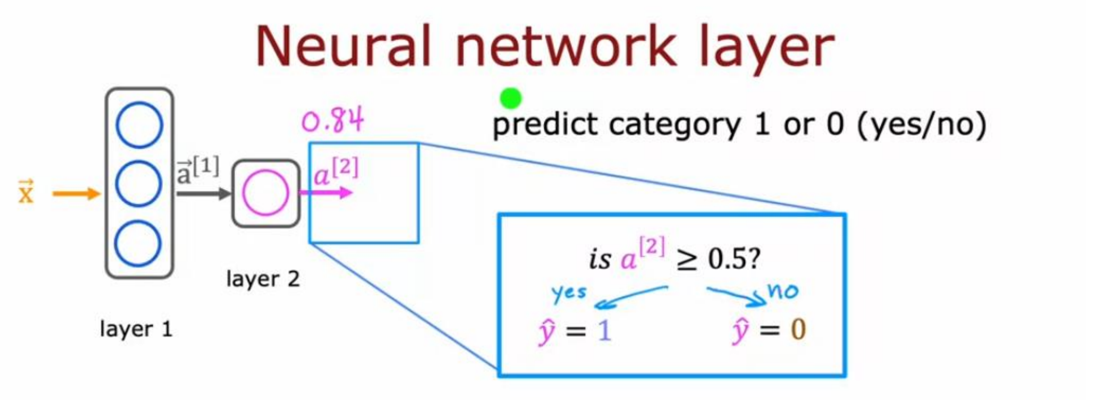
Coursera Honor Code Learn more

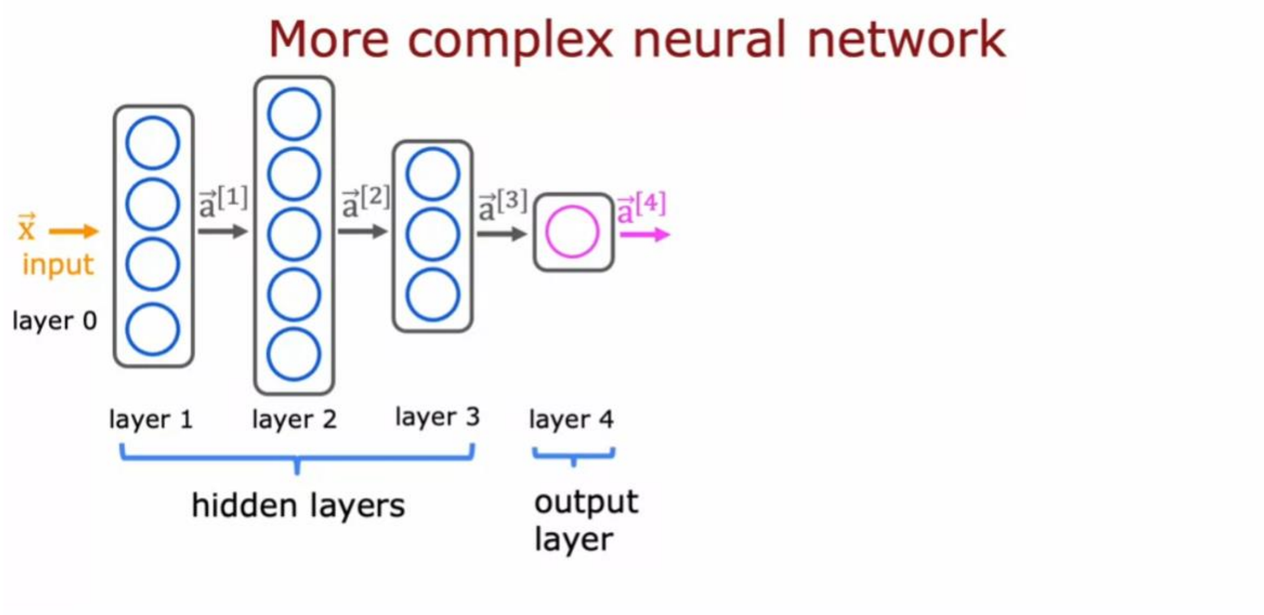
- [x] I, **Şaban Kara**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

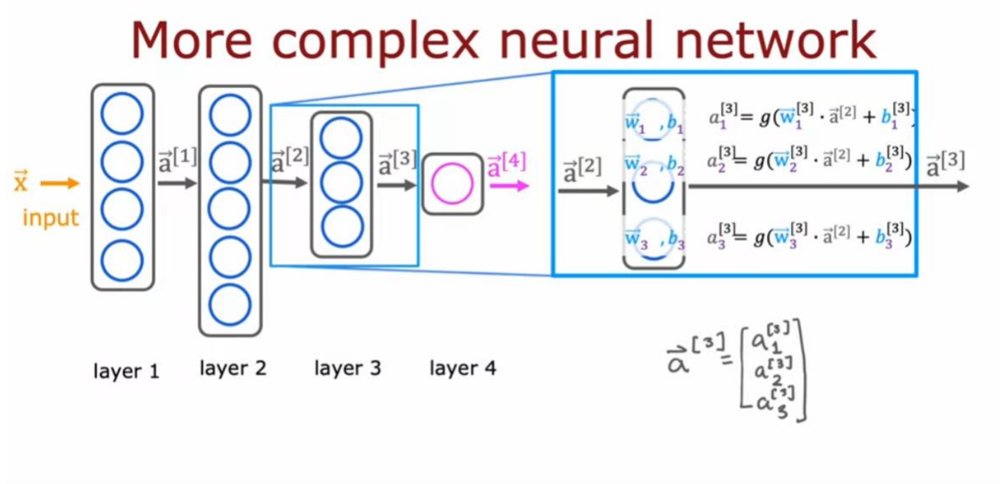


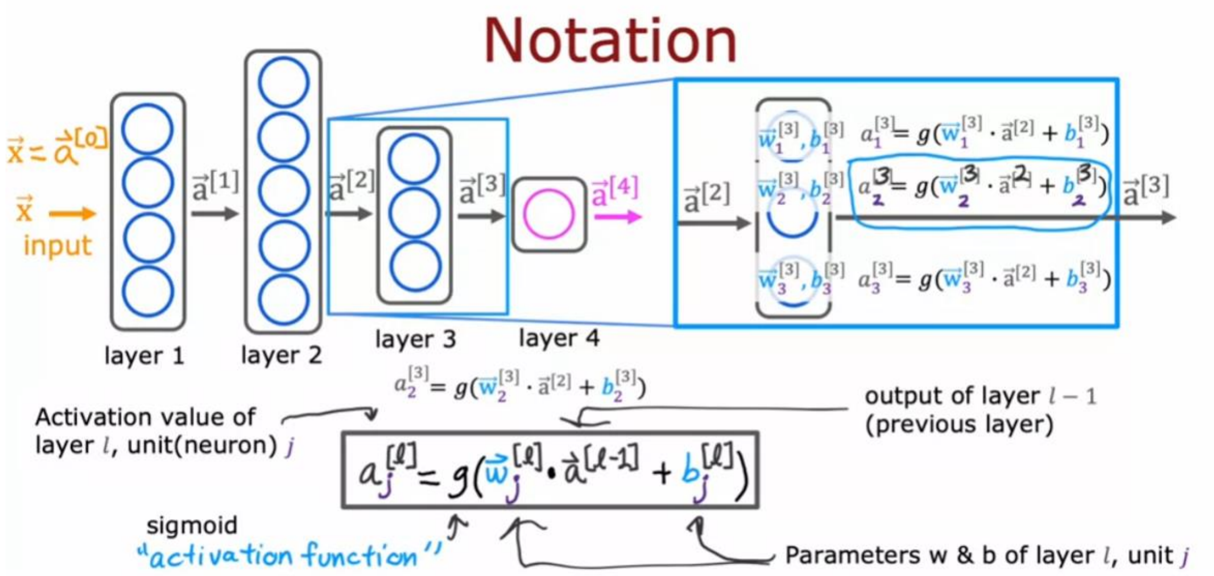
A üstü parantez sayı hangi katmana ait olduğunu gösterir.



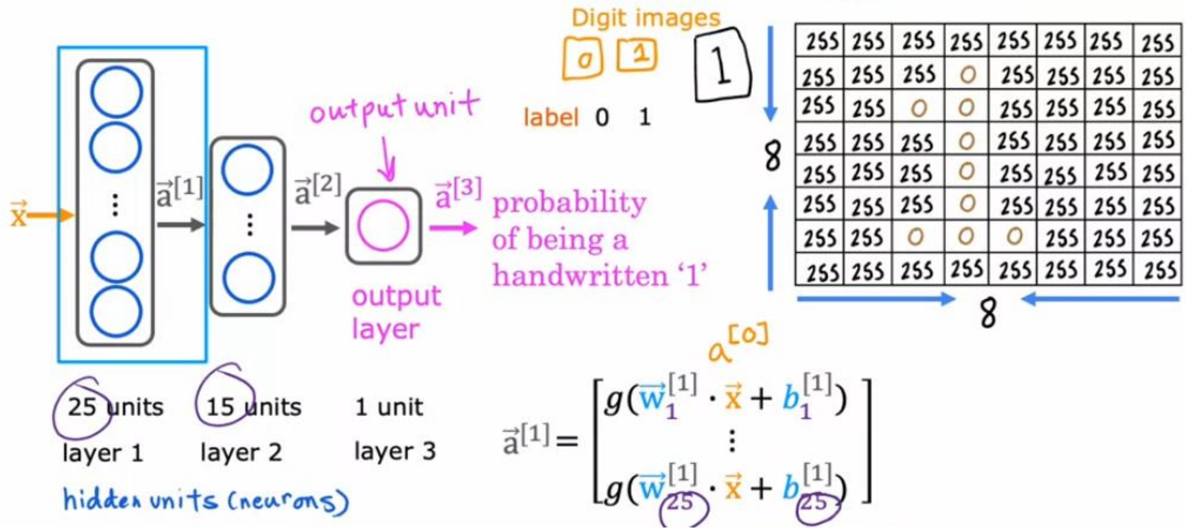



More complex neural network
$\vec{x}$ input
layer 0
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
$\vec{a}^{[4]}$
layer 1
layer 2
layer 3
layer 4
hidden layers
output layer


More complex neural network
$\vec{x}$ input
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
$\vec{a}^{[4]}$
$\vec{w}_1, b_1$
$\vec{w}_2, b_2$
$\vec{w}_3, b_3$
$a_1^{[3]} = g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$
$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
$a_3^{[3]} = g(\vec{w}_3^{[3]} \cdot \vec{a}^{[2]} + b_3^{[3]})$
layer 1
layer 2
layer 3
layer 4
$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$


Notation
$\vec{x} = \vec{a}^{[0]}$
$\vec{x}$ input
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
$\vec{a}^{[4]}$
$\vec{w}_1^{[3]}, b_1^{[3]}$
$\vec{w}_2^{[3]}, b_2^{[3]}$
$\vec{w}_3^{[3]}, b_3^{[3]}$
$a_1^{[3]} = g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$
$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
$a_3^{[3]} = g(\vec{w}_3^{[3]} \cdot \vec{a}^{[2]} + b_3^{[3]})$
layer 1
layer 2
layer 3
layer 4
$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
Activation value of layer l, unit(neuron) j
output of layer $l-1$ (previous layer)
$a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$
sigmoid
"activation function"
Parameters w & b of layer l, unit j

Handwritten digit recognition

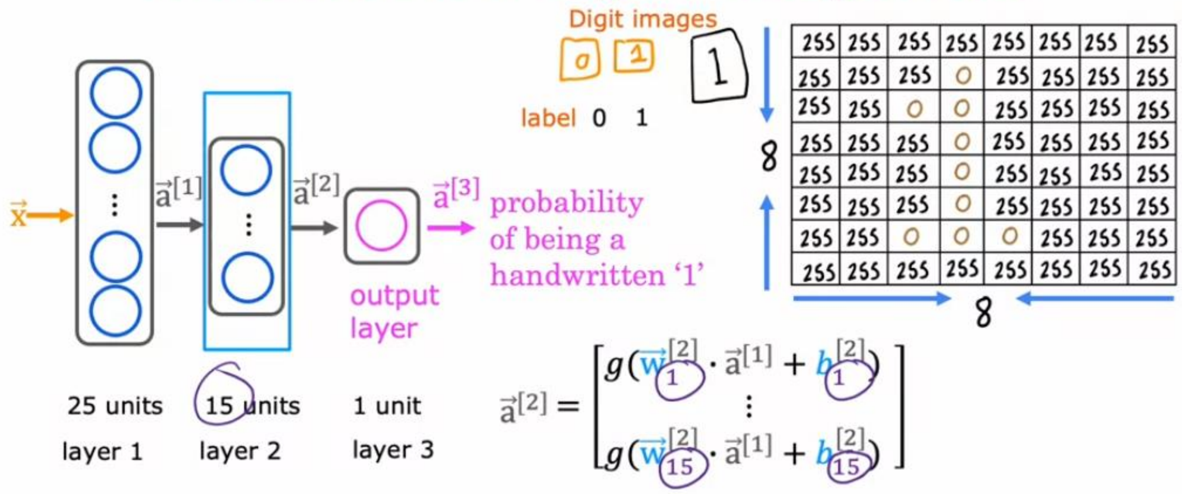


255	255	255	255	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	0	255	255	255
255	255	255	255	255	255	255	255

$a^{[0]}$

$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \\ \vdots \\ g(\vec{w}_{25}^{[1]} \cdot \vec{x} + b_{25}^{[1]}) \end{bmatrix}$$

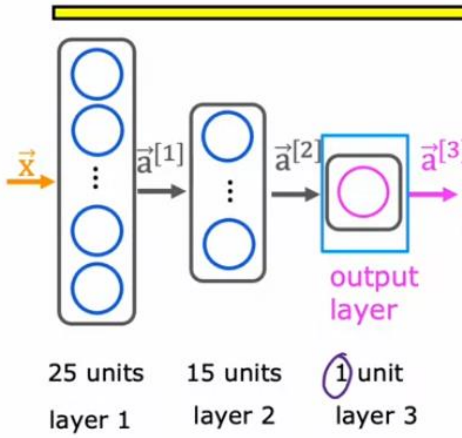
Handwritten digit recognition



$$\vec{a}^{[2]} = \begin{bmatrix} g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) \\ \vdots \\ g(\vec{w}_{15}^{[2]} \cdot \vec{a}^{[1]} + b_{15}^{[2]}) \end{bmatrix}$$

Handwritten digit recognition

forward propagation



$$\vec{a}^{[3]} = \left[g\left(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}\right) \right]$$

is $a_1^{[3]} \geq 0.5$?

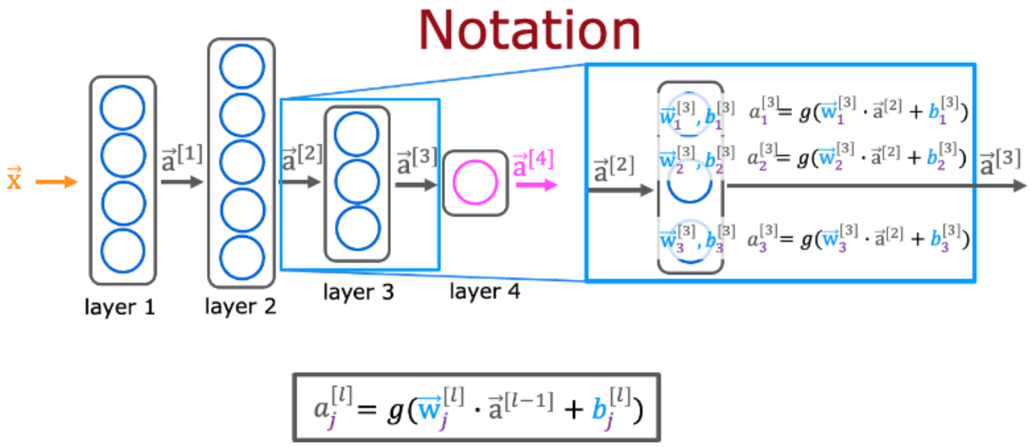
yes / no

$\hat{y} = 1$ — image is digit 1

$\hat{y} = 0$ — image isn't digit 1

1.

1 point

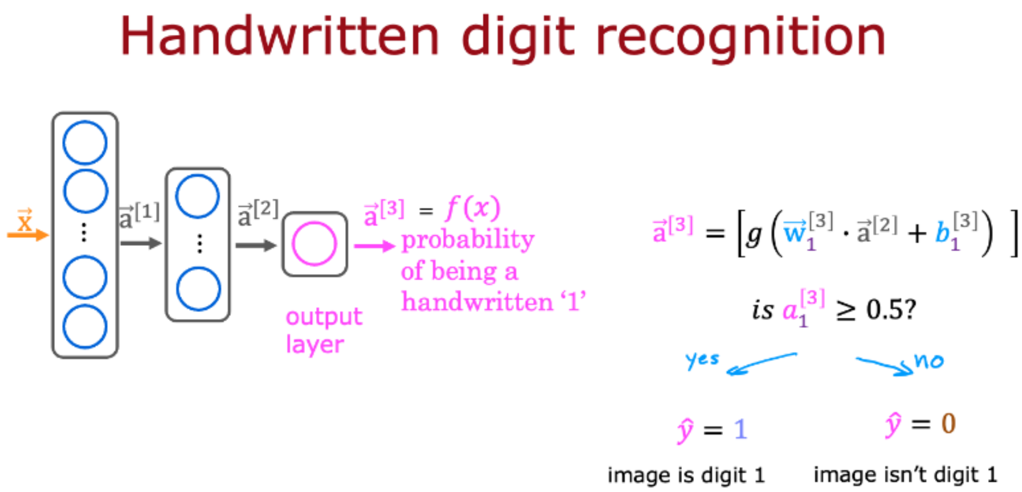


For a neural network, what is the expression for calculating the activation of the third neuron in layer 2? Note, this is different from the question that you saw in the lecture video.

- ○ $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[2]} + b_3^{[2]})$
- ◉ $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[1]} + b_3^{[2]})$
- ○ $a_3^{[2]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
- ○ $a_3^{[2]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[1]} + b_2^{[3]})$

2.

1 point



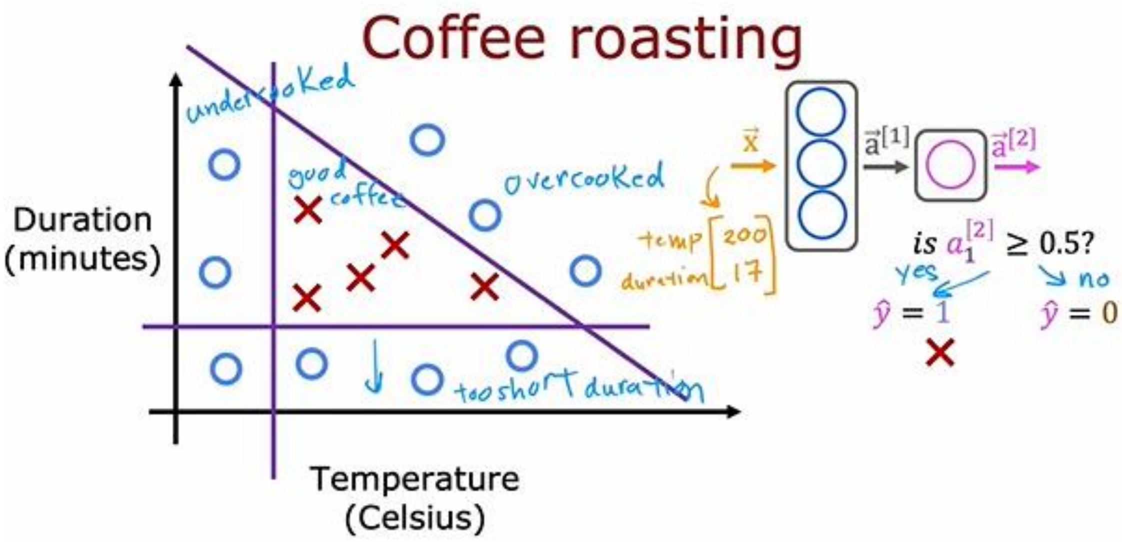
For the handwriting recognition task discussed in lecture, what is the output $a_1^{[3]}$?

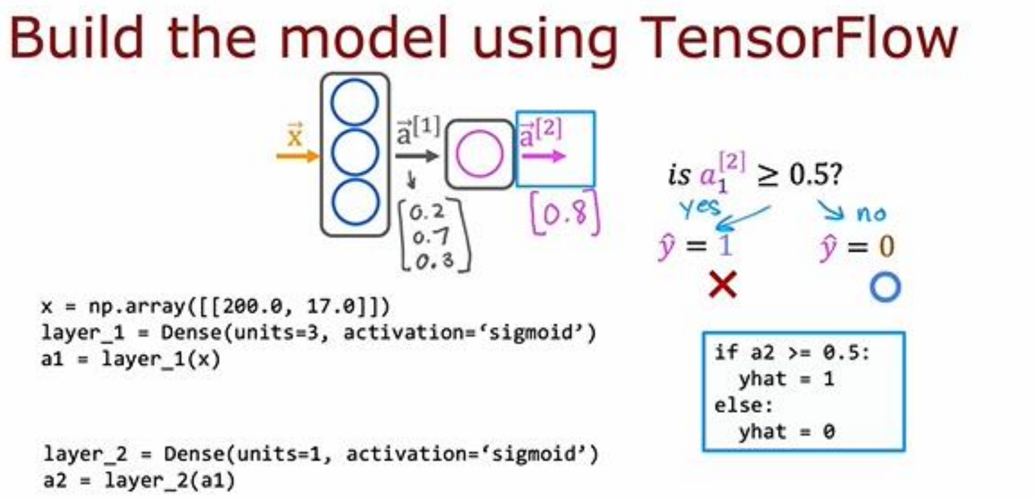
- ○ A vector of several numbers, each of which is either exactly 0 or 1
- ○ A vector of several numbers that take values between 0 and 1
- ○ A number that is either exactly 0 or 1, comprising the network's prediction
- ◉ The estimated probability that the input image is of a number 1, a number that ranges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

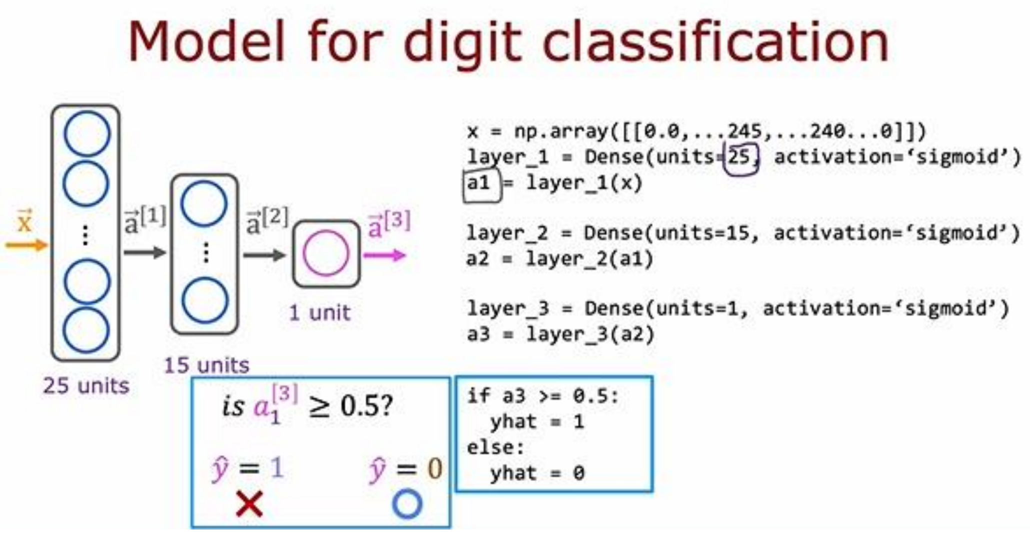
Coursera Honor Code [Learn more](#)

☐ I, **Şaban Kara**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

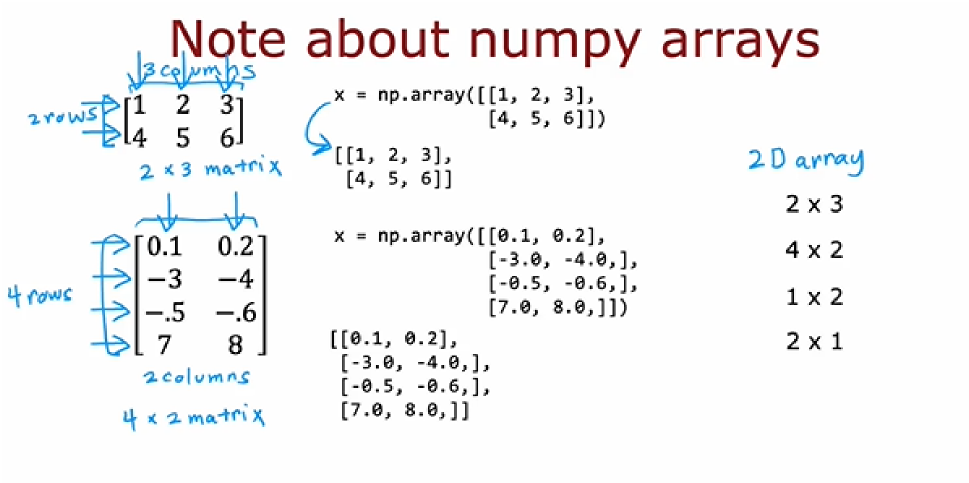
Inference in Code

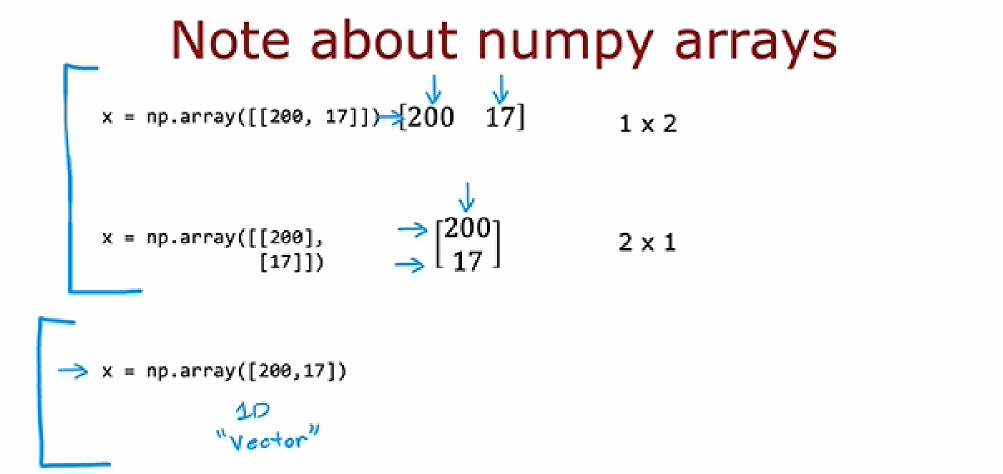


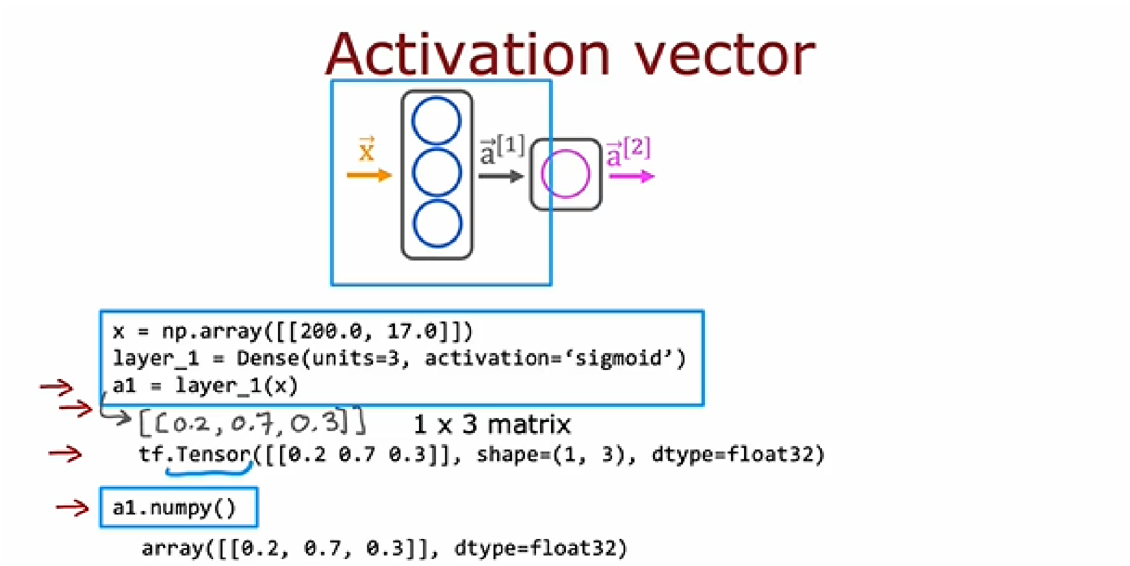




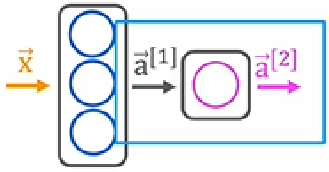
Data in Tensorflow







Activation vector




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

[[0.8]]

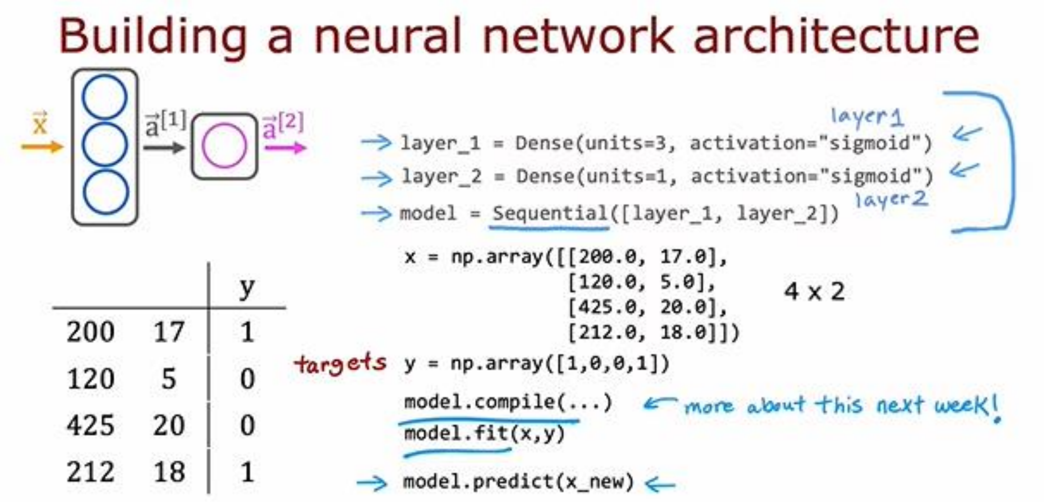
1 x 1

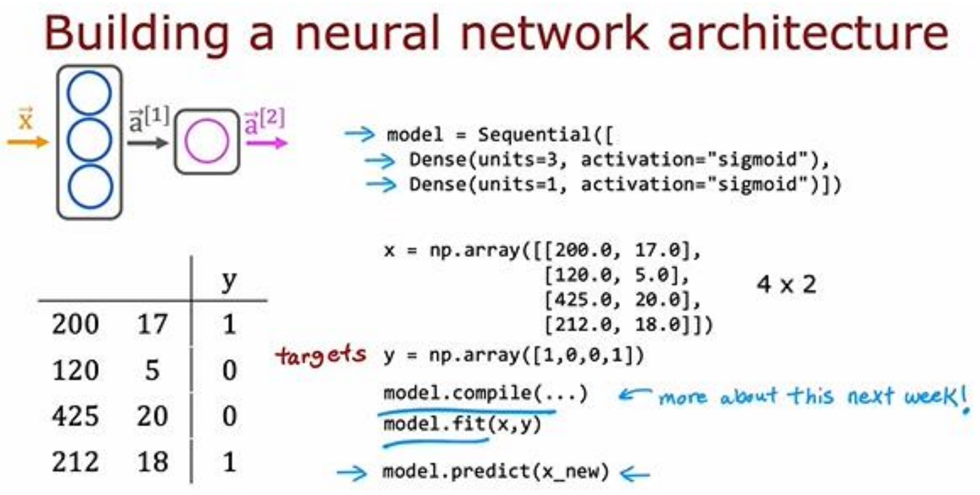
```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

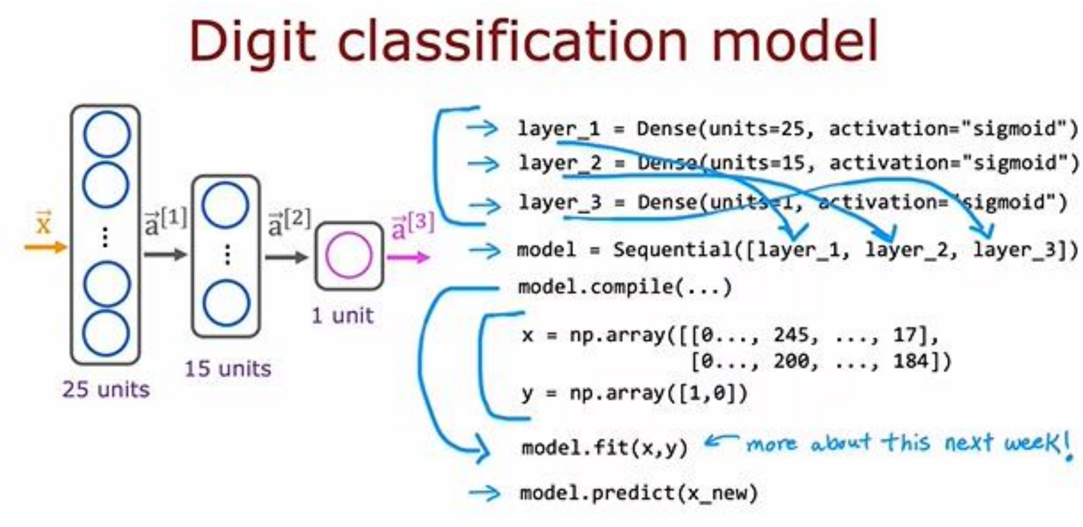
```
a2.numpy()
```

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

Building a Neural Network







1. For the the following code: 1 point

model = Sequential([

Dense(units=25, activation="sigmoid"),

Dense(units=15, activation="sigmoid"),

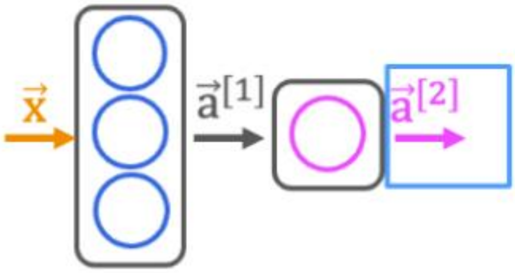
Dense(units=10, activation="sigmoid"),

Dense(units=1, activation="sigmoid")])

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

- ○ 3
- ◉ 4
- ○ 25
- ○ 5

2. 1 point



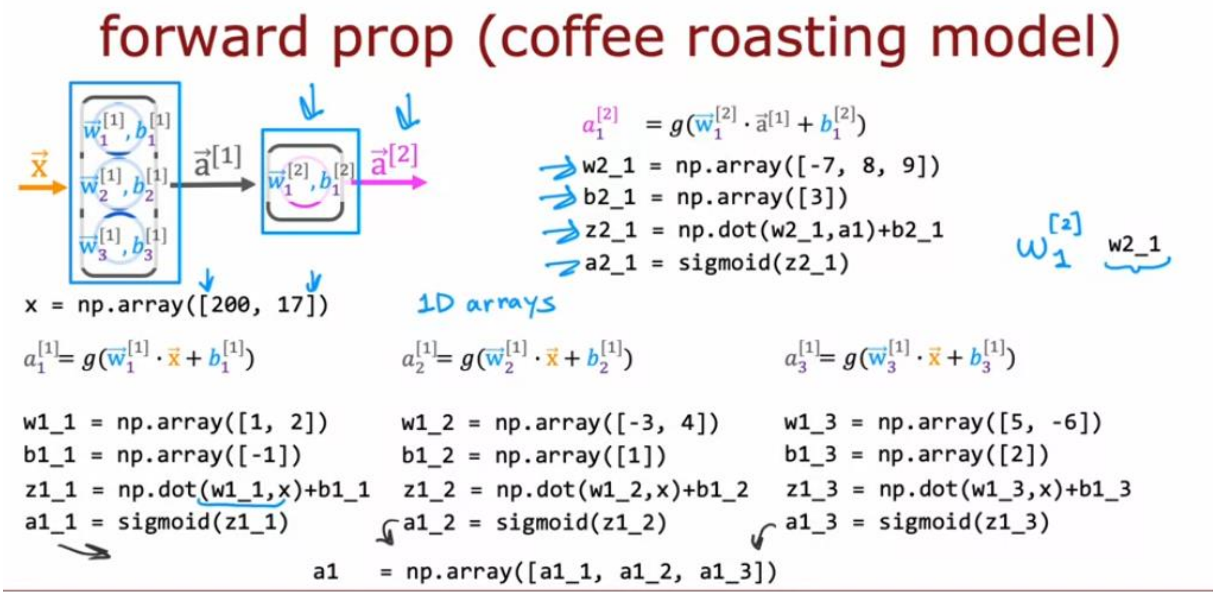

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)


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

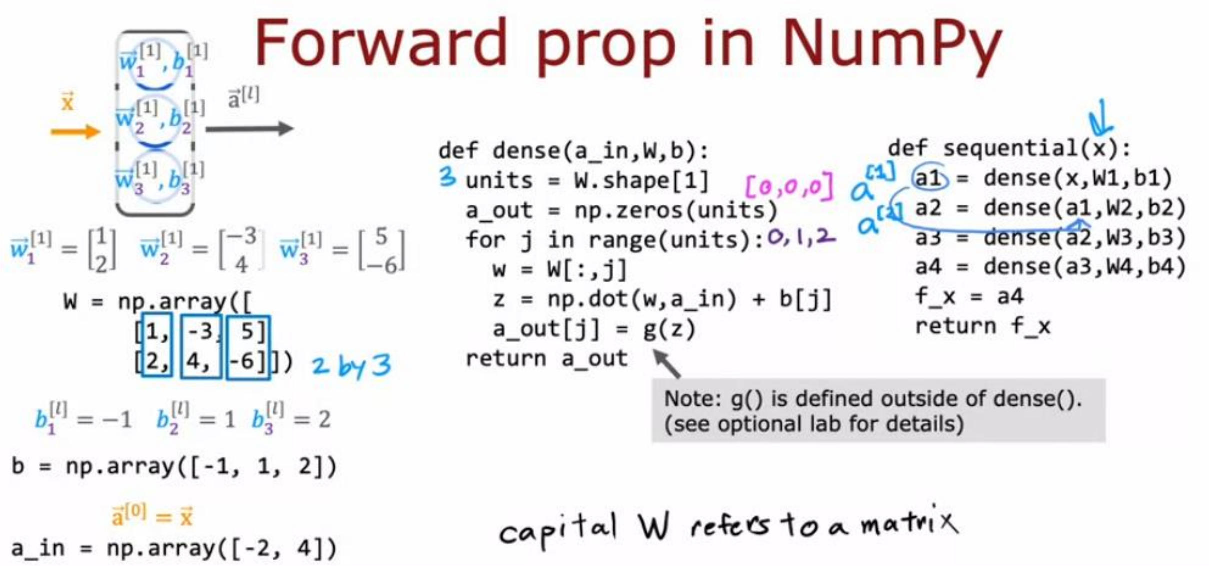
How do you define the second layer of a neural network that has 4 neurons and a sigmoid activation?

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

Forward prop in a single layer

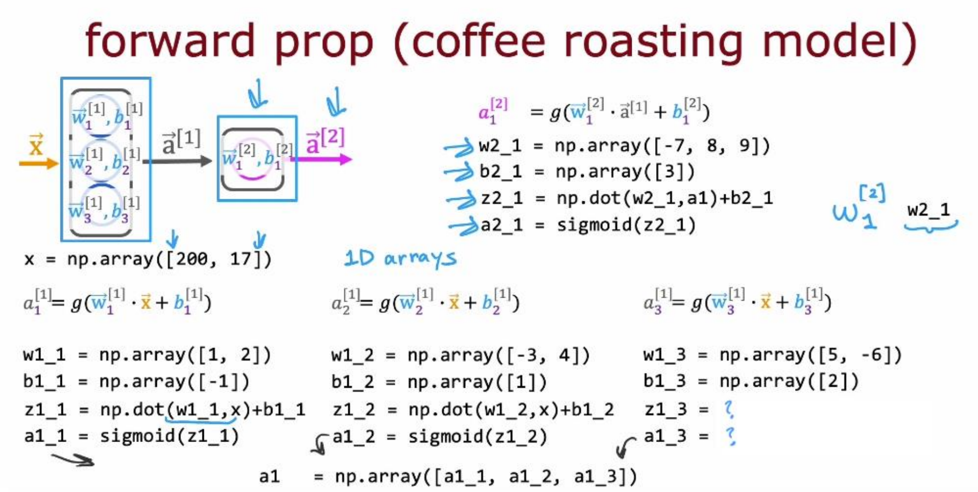


Genearal implemantation of forward propagation



1.

1 point



According to the lecture, how do you calculate the activation of the third neuron in the first layer using NumPy?

- [ ] layer_1 = Dense(units=3, activation='sigmoid')

  a_1 = layer_1(x)

- [ ] z1_3 =w1_3 * x + b

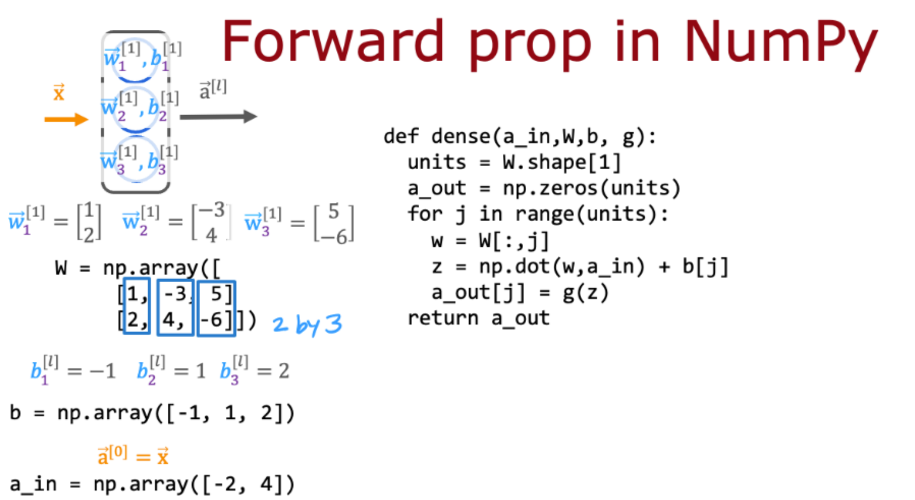
  a1_3 = sigmoid(z1_3)

- [x] z1_3 = np.dot(w1_3, x) + b1_3

  a1_3 = sigmoid(z1_3)

2.

1 point



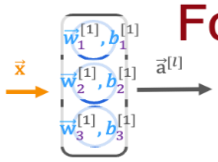
According to the lecture, when coding up the numpy array W, where would you place the w parameters for each neuron?

- [ ] In the rows of W.
- [x] In the columns of W.

3.

1 point

Forward prop in NumPy



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

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

2 by 3

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

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

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

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

```
def dense(a_in,W,b, g):
  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
```

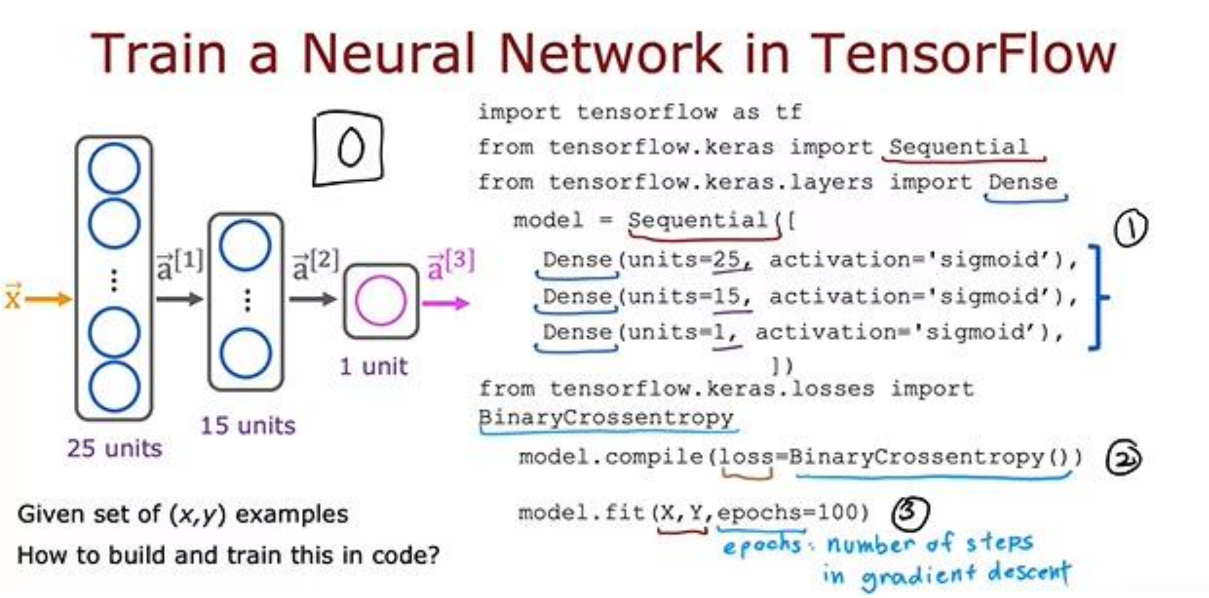
For the code above in the "dense" function that defines a single layer of neurons, how many times does the code go through the "for loop"? Note that W has 2 rows and 3 columns.

- ○ 2 times
- ○ 6 times
- ◉ 3 times
- ○ 5 times

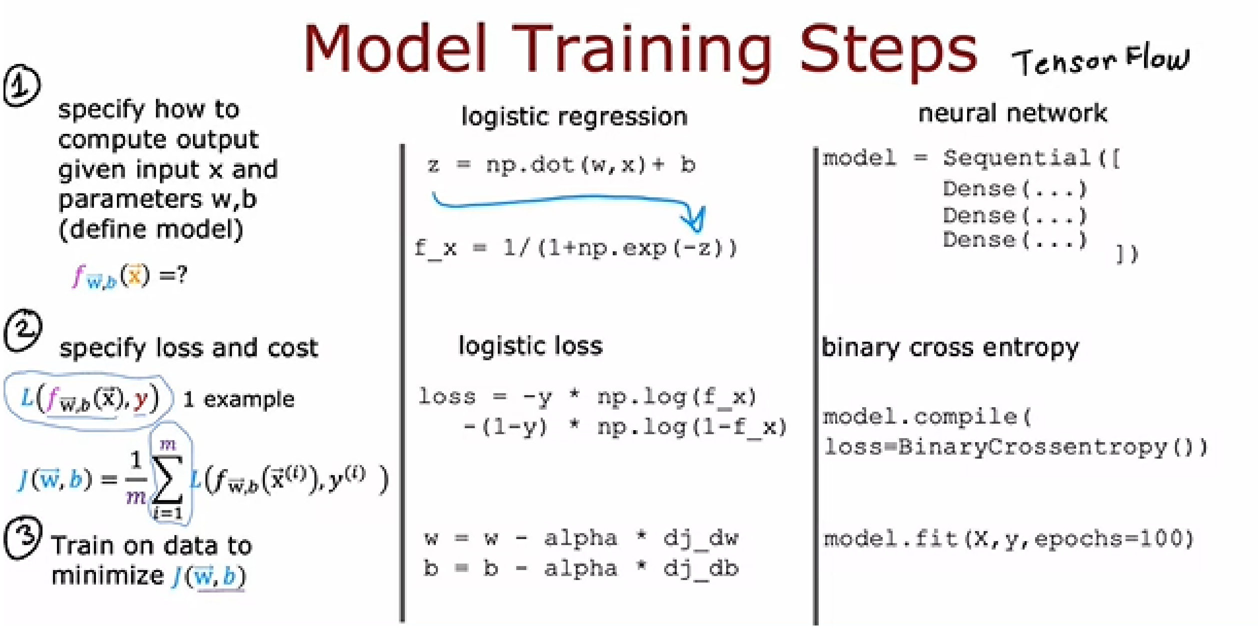
Coursera Honor Code Learn more

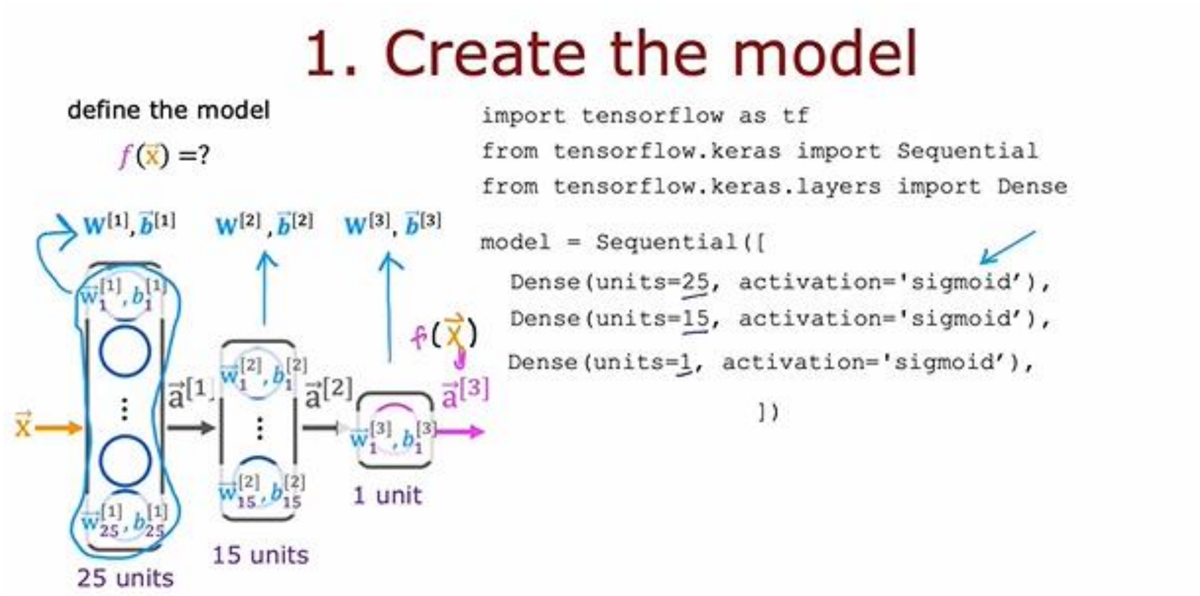
☑ I, **Şaban Kara**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Tensorflow Implementation



Training Details





2. Loss and cost functions

handwritten digit classification problem → binary classification

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

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

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$ $\vec{b}^{[1]}, \vec{b}^{[2]}, \vec{b}^{[3]}$ $f_{W,B}(\vec{x})$

compare prediction vs. target

logistic loss
also known as binary cross entropy

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

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

Keras

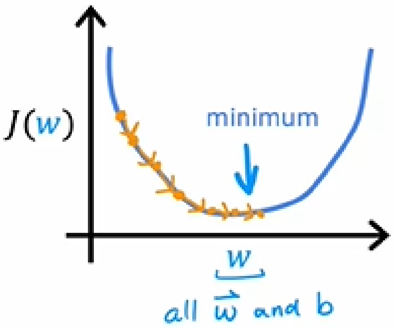
regression
(predicting numbers
and not categories)

mean squared error

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

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

3. Gradient descent



repeat {

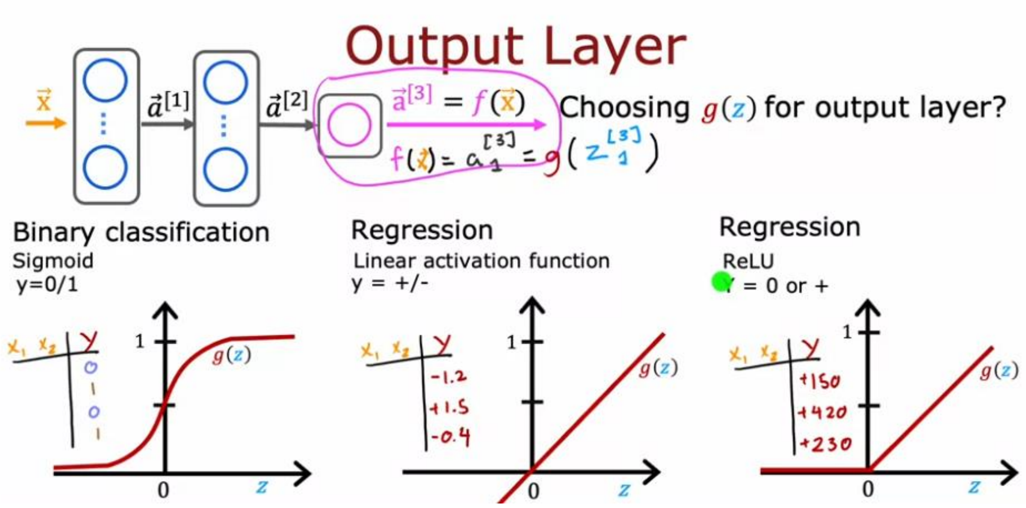
$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

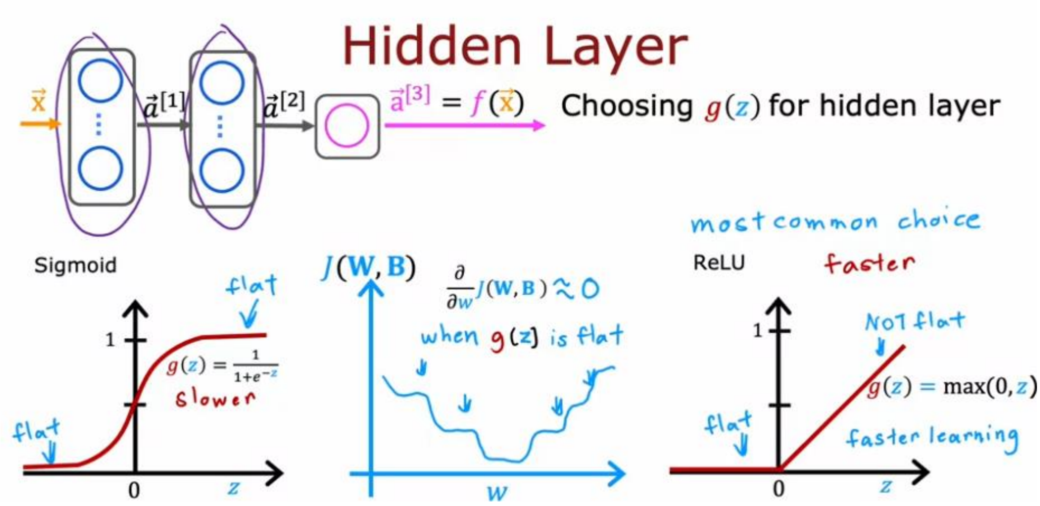
$$b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial bj} J(\vec{w}, b)$$

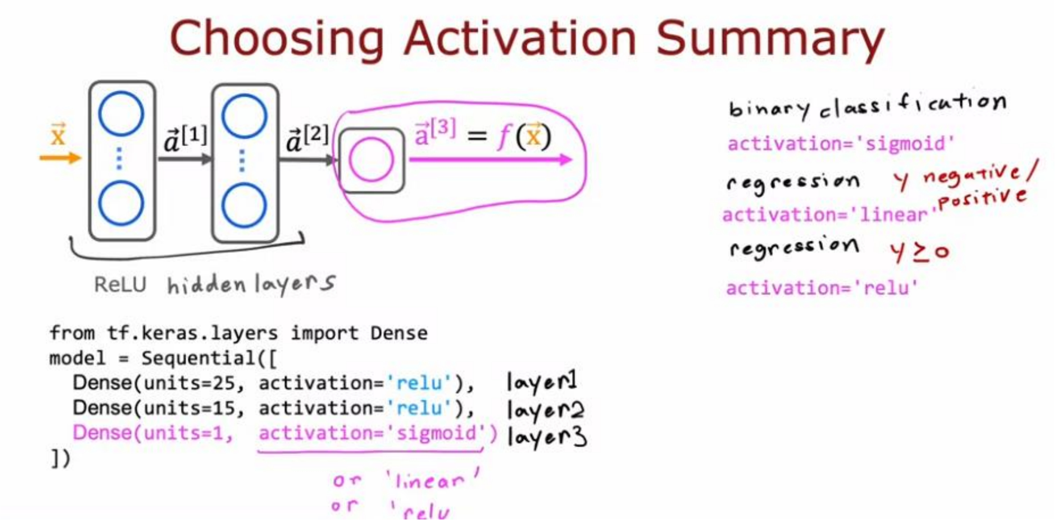
} Compute derivatives
for gradient descent
using "back propagation"

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

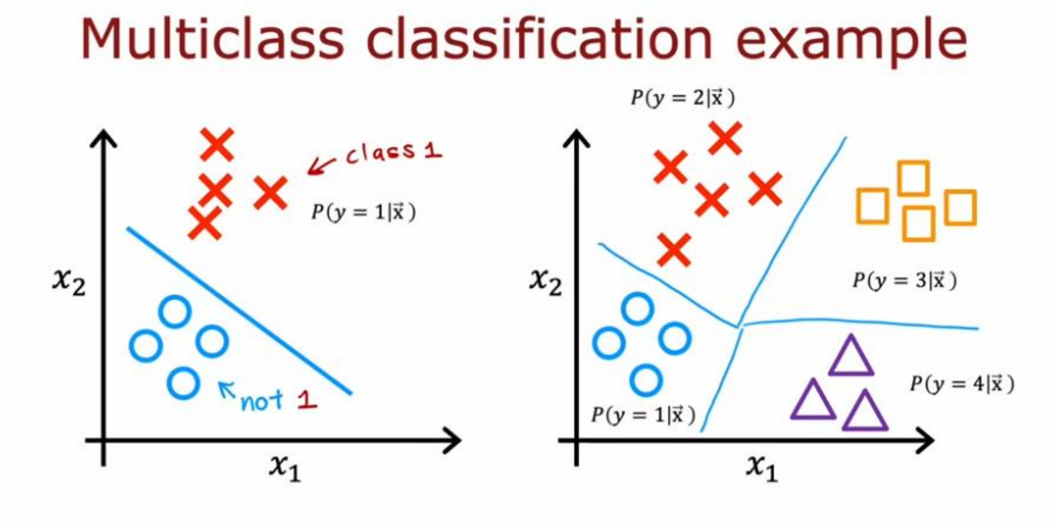
Choosing activation function



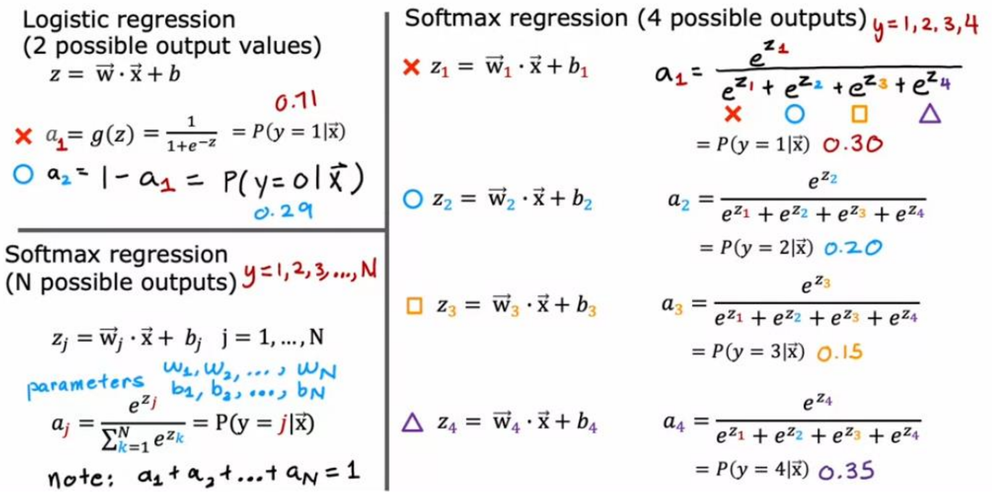


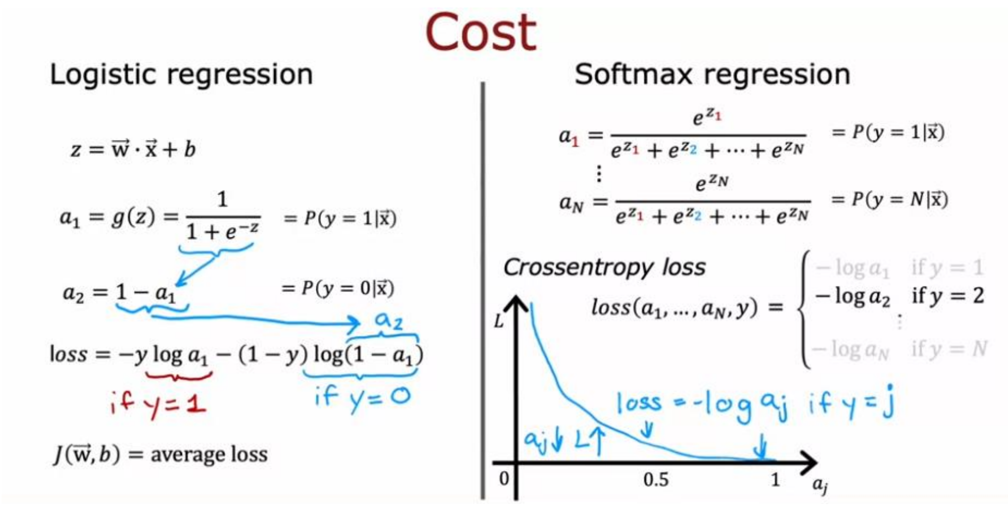


Multiclass

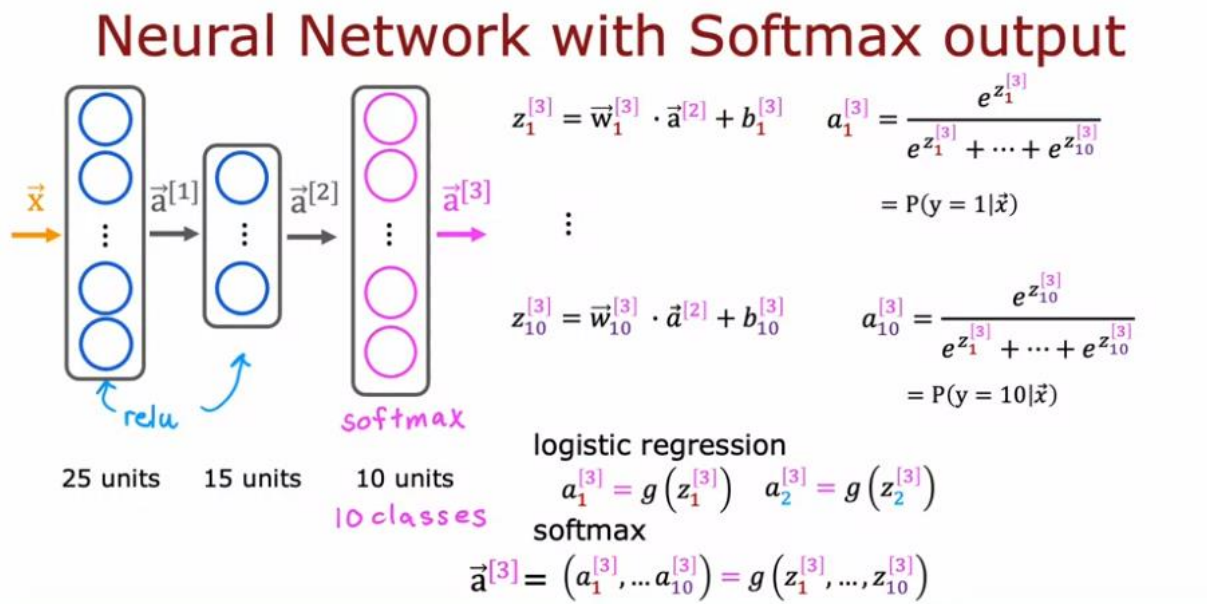


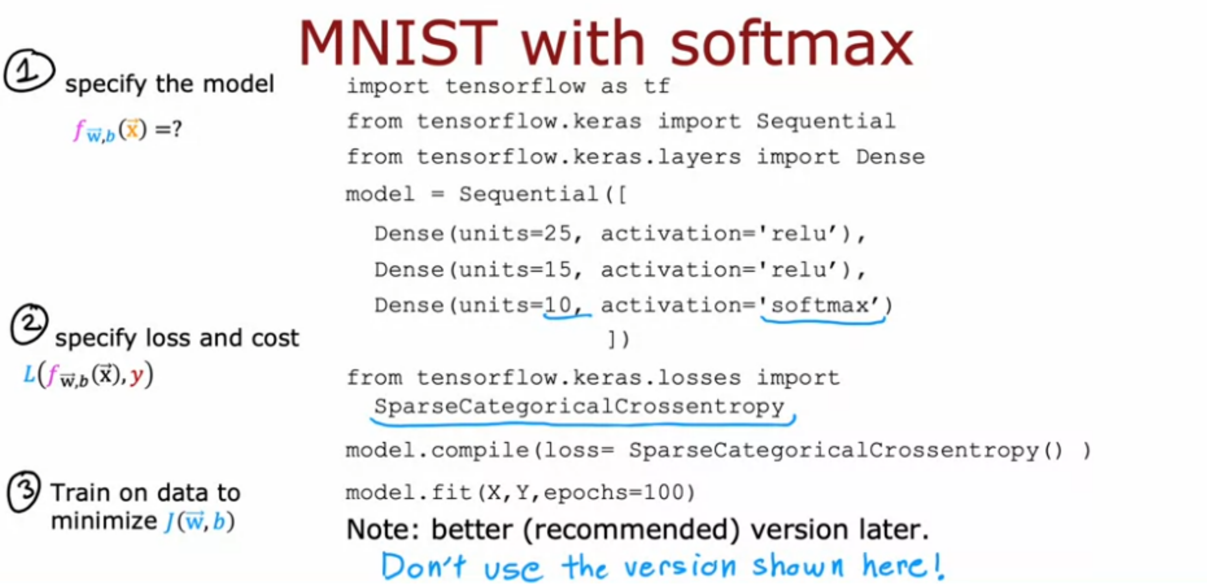
Softmax





Neural Network with Softmax output





Improved implementation of softmax

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss: $1+\frac{1}{10{,}000}$ $\quad 1-\frac{1}{10{,}000}$

Logistic regression:

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

Original loss

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

More accurate loss (in code)

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

logit: z

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=10, activation='sigmoid')  'linear'
  ])
model.compile(loss=BinaryCrossEntropy() )
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

More numerically accurate implementation of softmax

Softmax regression

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

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

More Accurate

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

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=10, activation='softmax')
  ])                                    'linear'
model.compile(loss=SparseCategoricalCrossEntropy() )
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

MNIST (more numerically accurate)

model

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

loss

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

fit

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

predict

```
logits = model(X)
```

not $a_1 \ldots a_{10}$
is $z_1 \ldots$

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

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

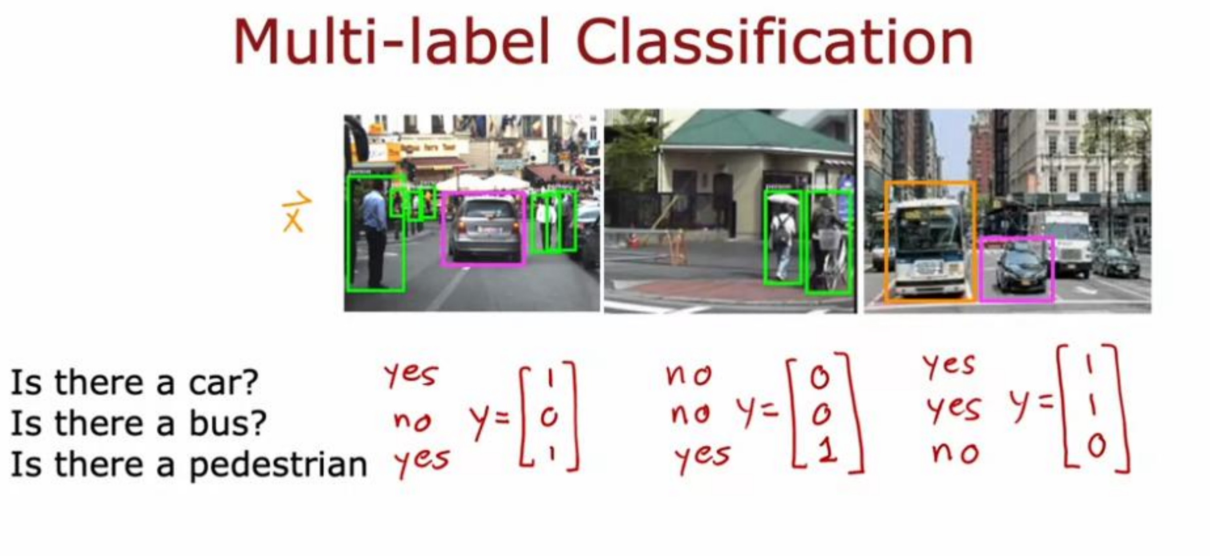
```
logit = model(X)
```

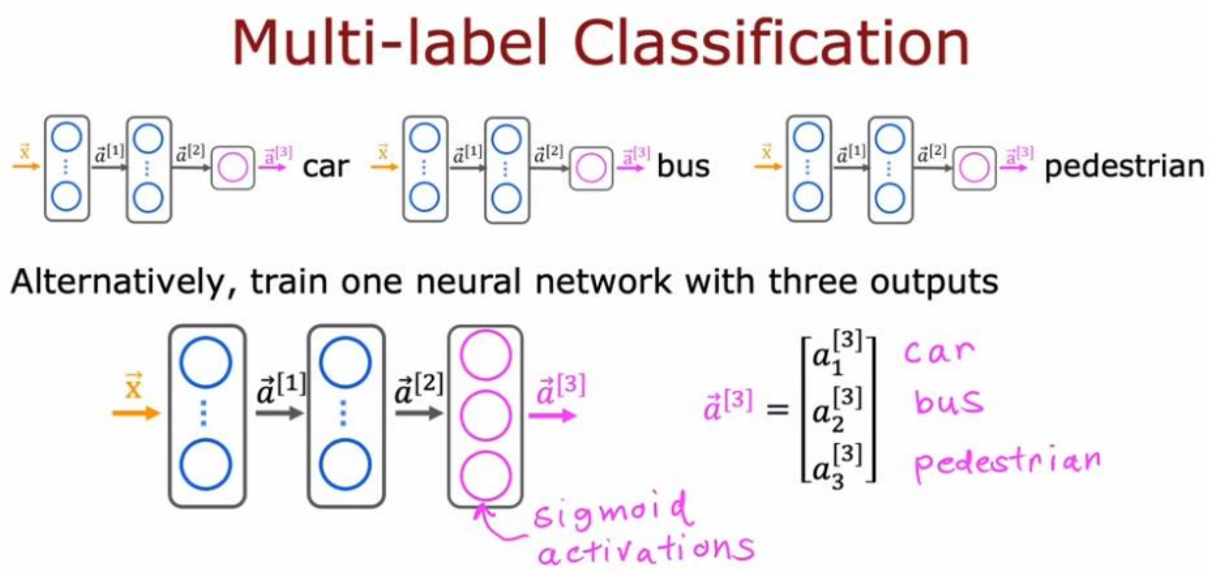
z

predict

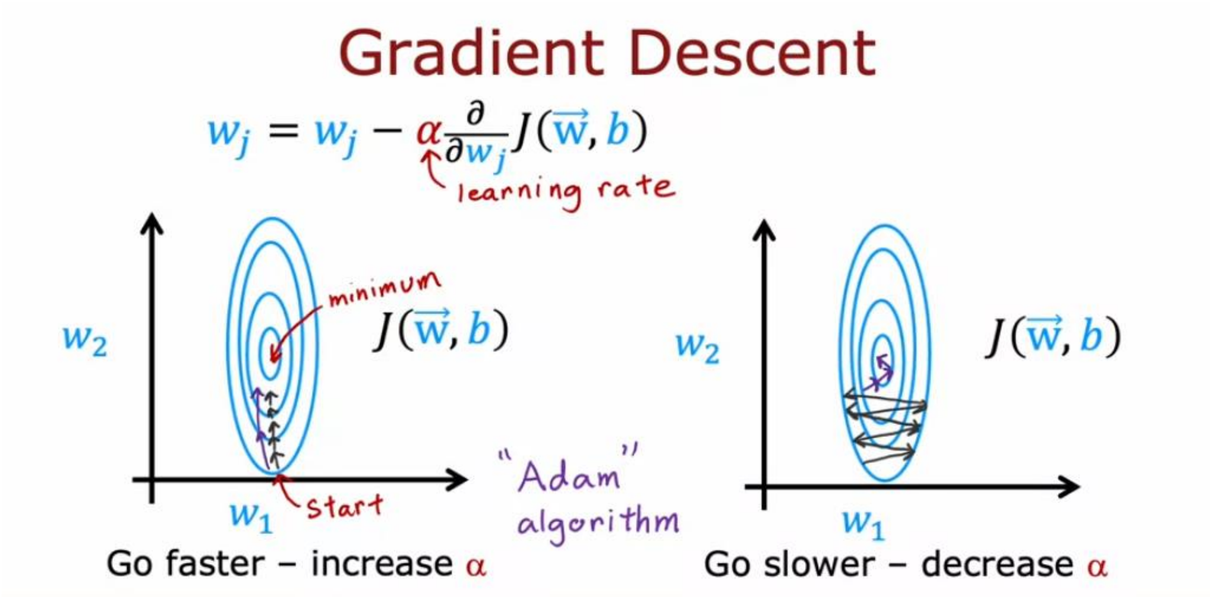
```
f_x = tf.nn.sigmoid(logit)
```

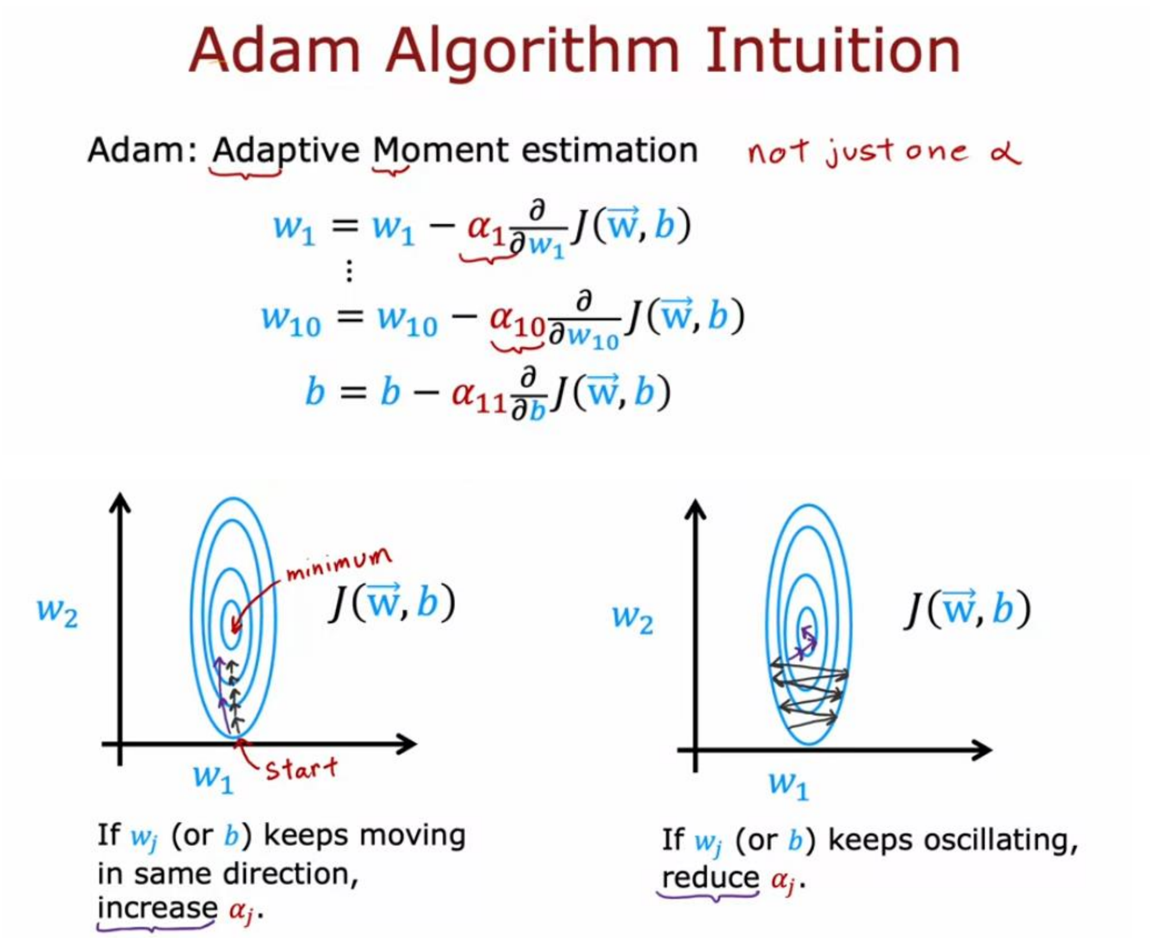
Classification with multiple outputs





Advanced Optimization





MNIST Adam

model

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

compile

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

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

fit

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

Additional Layer Types