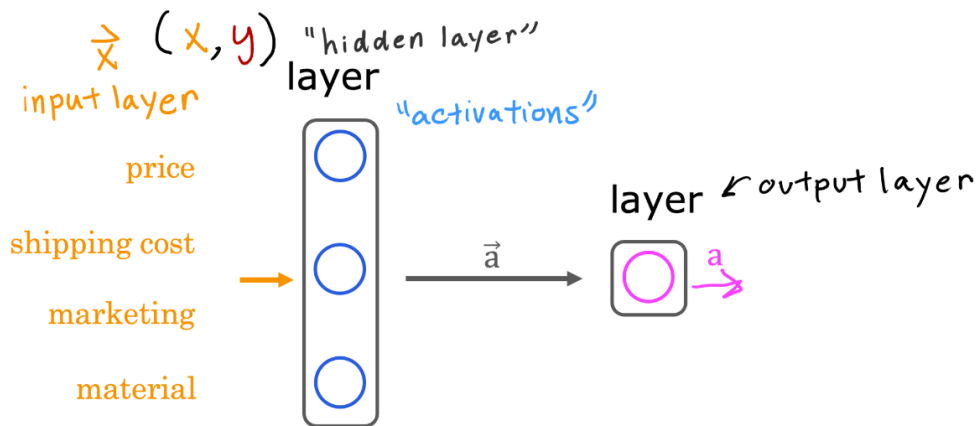


1.

1 point



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

- ☒ neurons
- ☒ layers
- ☐ axon
- ☒ activation function

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

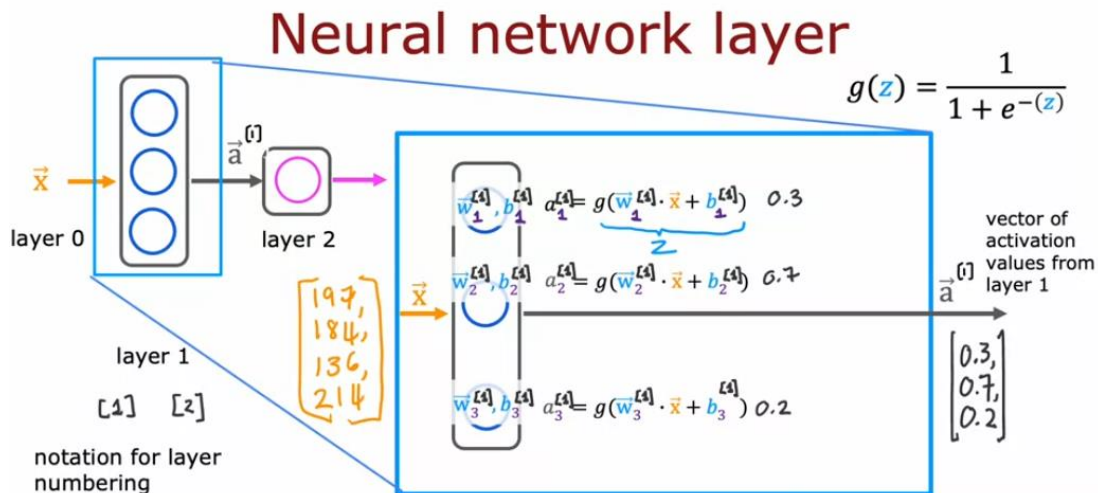
1 point

- ☒ True
- ☐ False

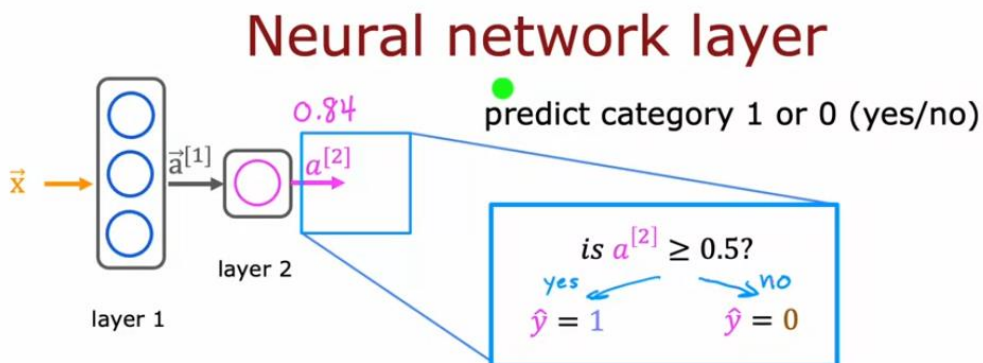
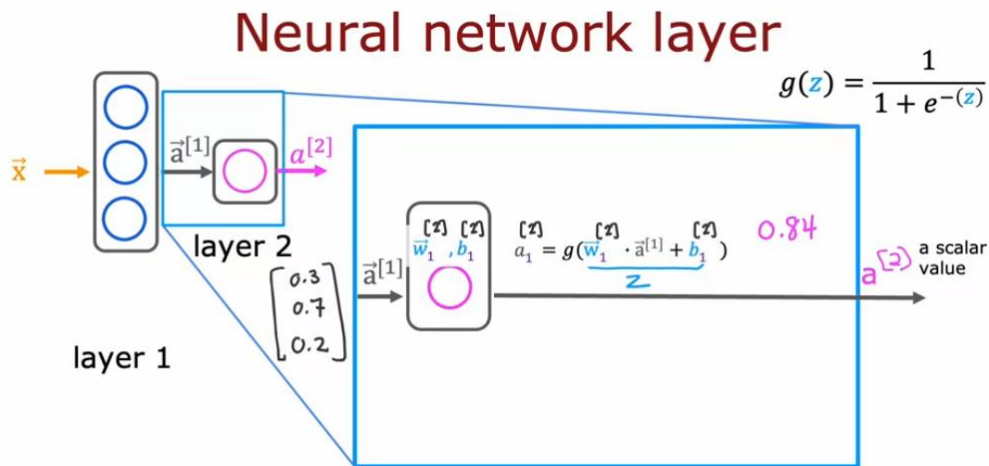
Coursera Honor Code [Learn more](#)

Windows'u Etkinleştir
Windows'u etkinleştirmek için Ayarlar'a gidin.

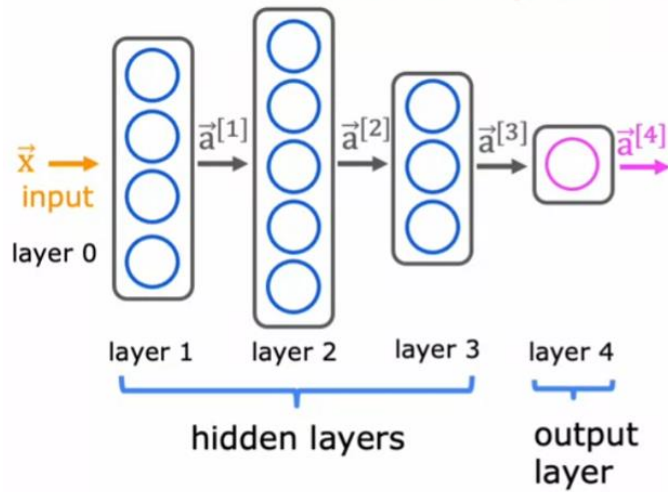
☒ I, **Şaban Kara**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.



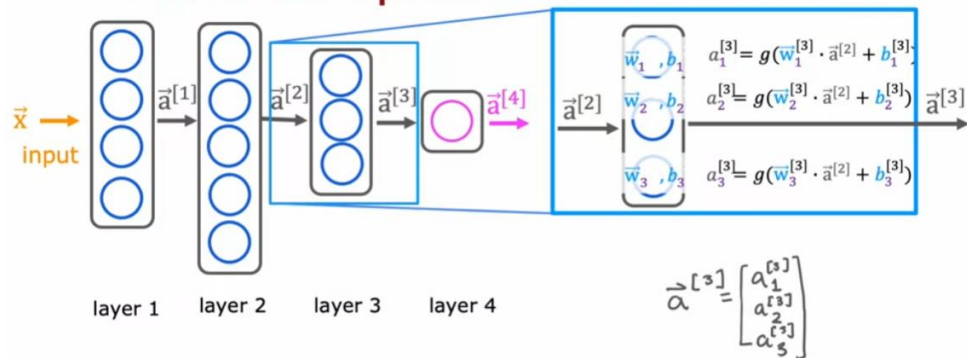
A üstü parantez sayı hangi katmana ait olduğunu gösterir.



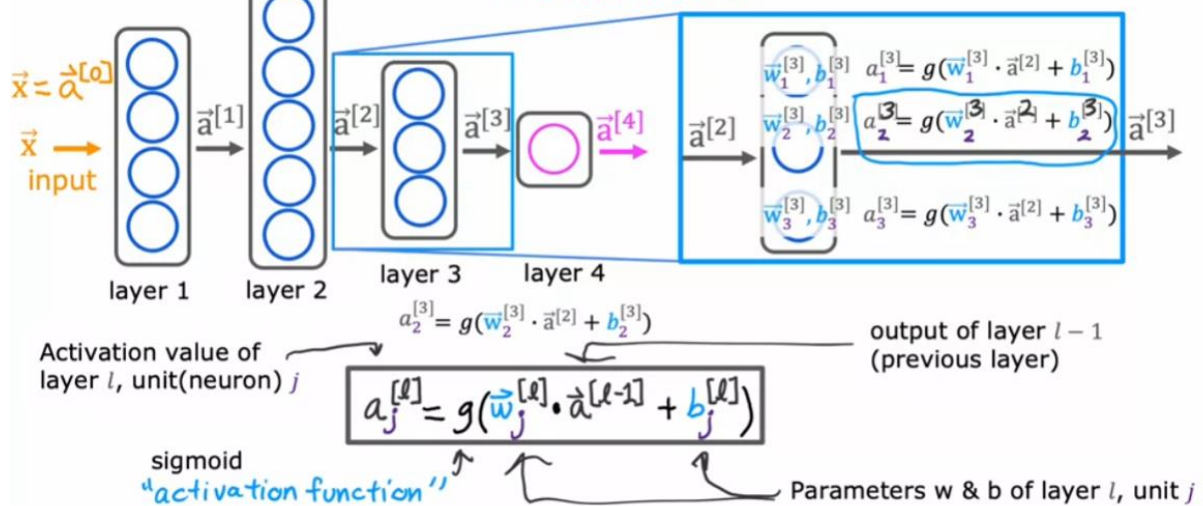
More complex neural network



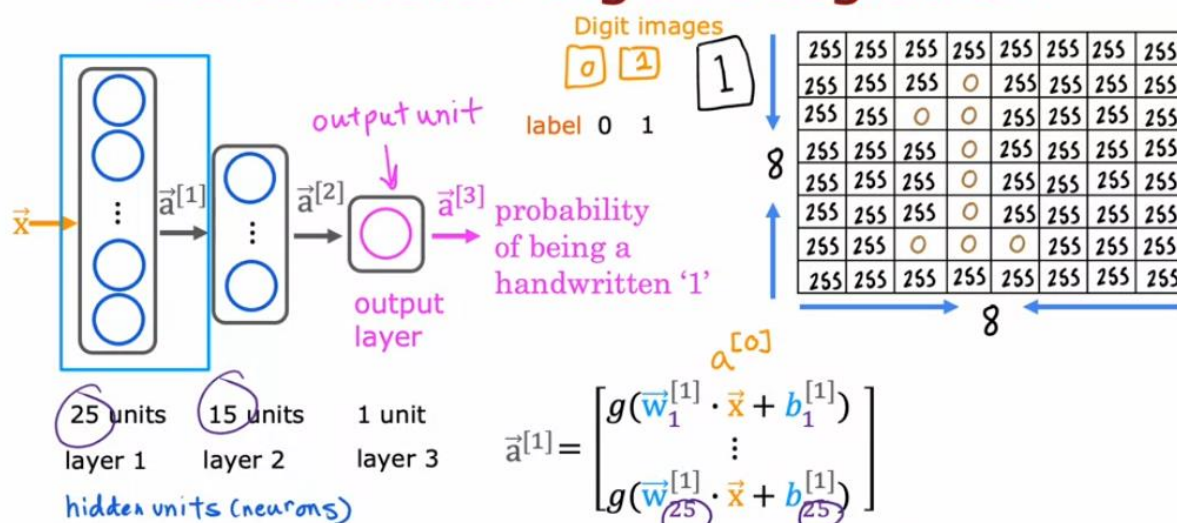
More complex neural network



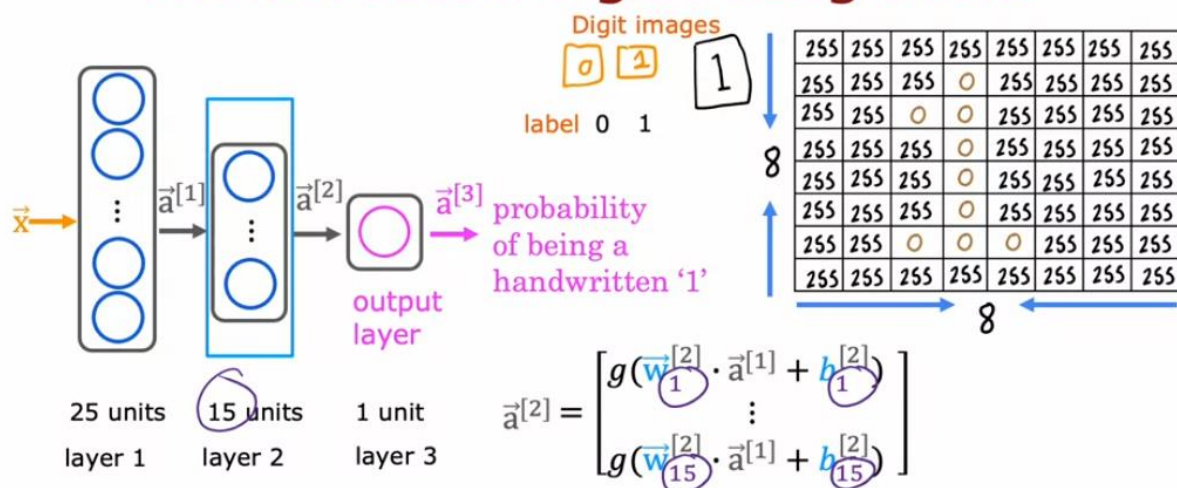
Notation



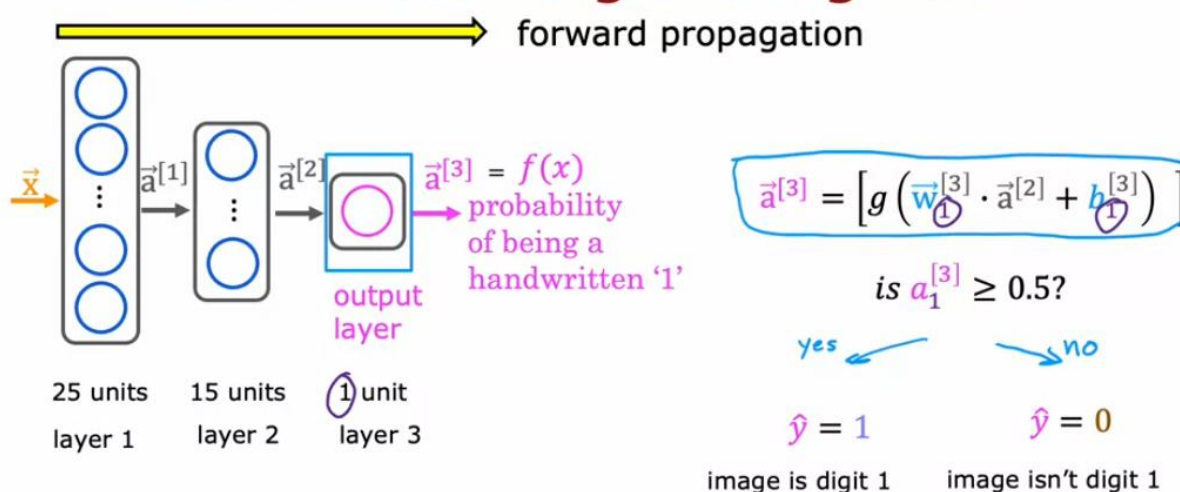
Handwritten digit recognition



Handwritten digit recognition

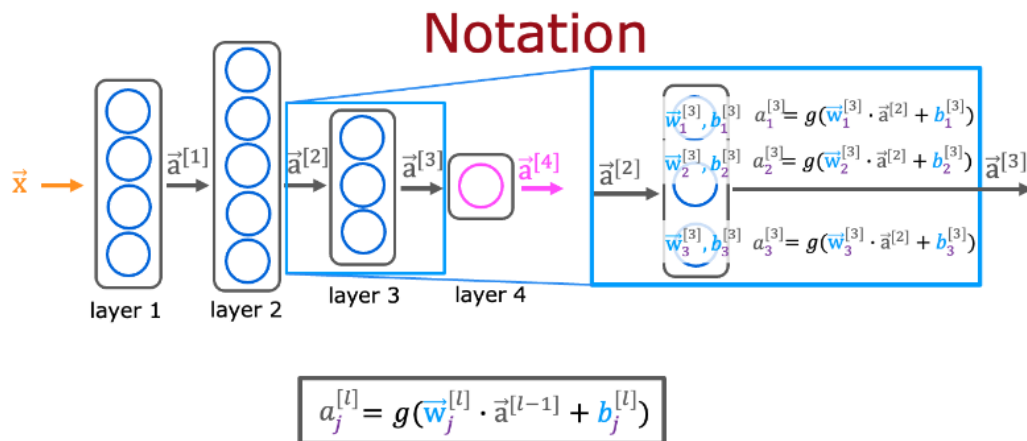


Handwritten digit recognition



1.

1 point



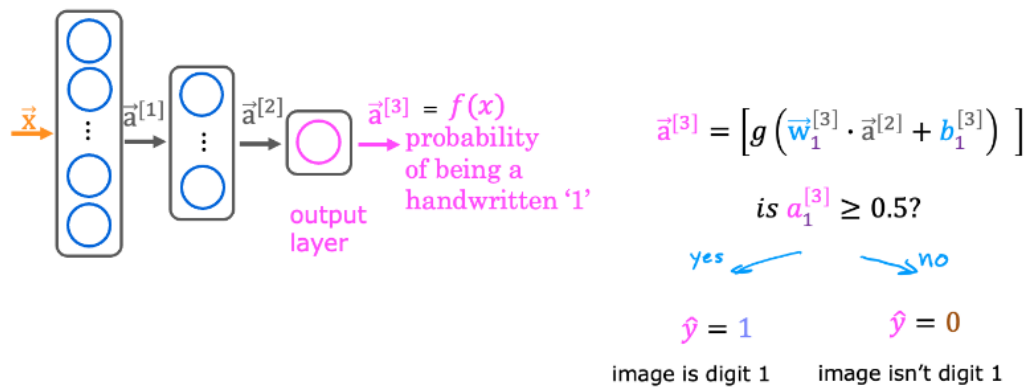
For a neural network, what is the expression for calculating the activation of the third neuron in layer 2? Note, this is different from the question that you saw in the lecture video.

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

2.

1 point

Handwritten digit recognition



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

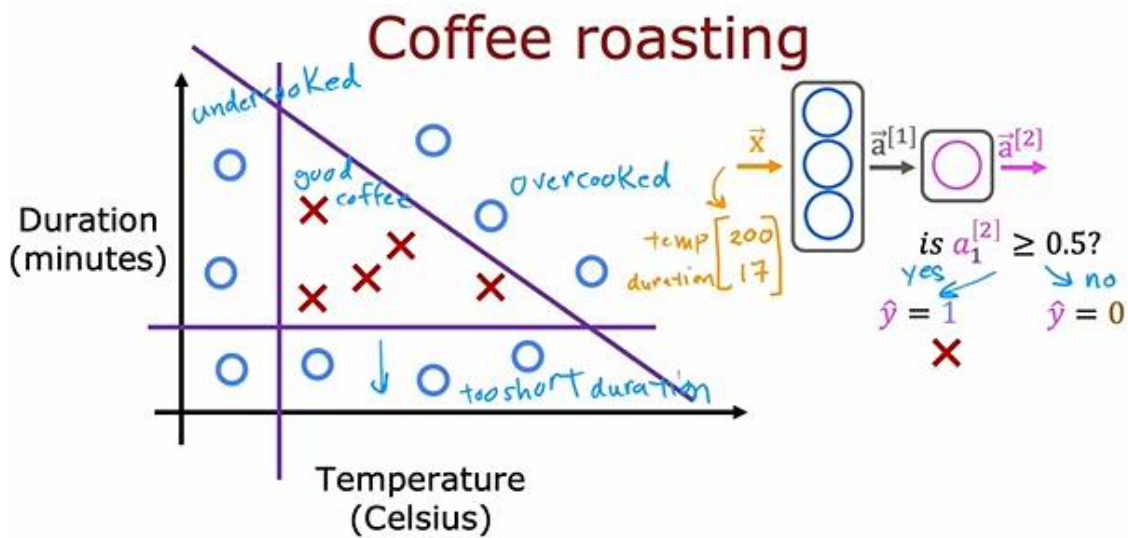
- ☐ A vector of several numbers, each of which is either exactly 0 or 1
☐ A vector of several numbers that take values between 0 and 1
☐ A number that is either exactly 0 or 1, comprising the network's prediction
☒ The estimated probability that the input image is of a number 1, a number that ranges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

Coursera Honor Code [Learn more](#)

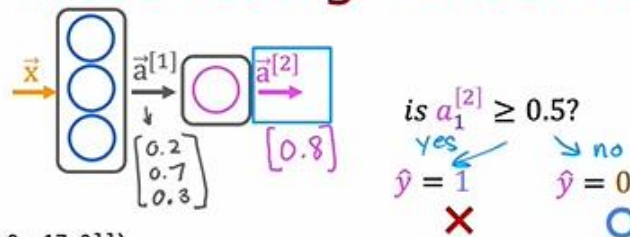


I, **Şaban Kara**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Inference in Code



Build the model using TensorFlow

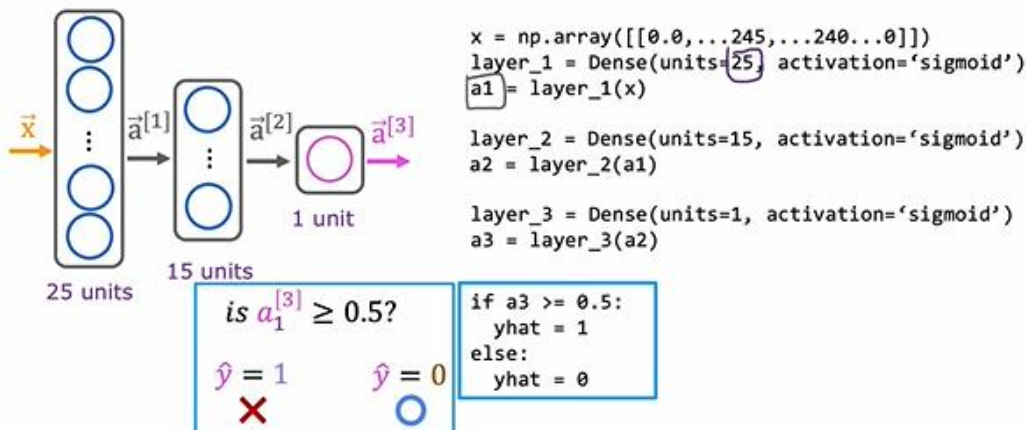


```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

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

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

Model for digit classification



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

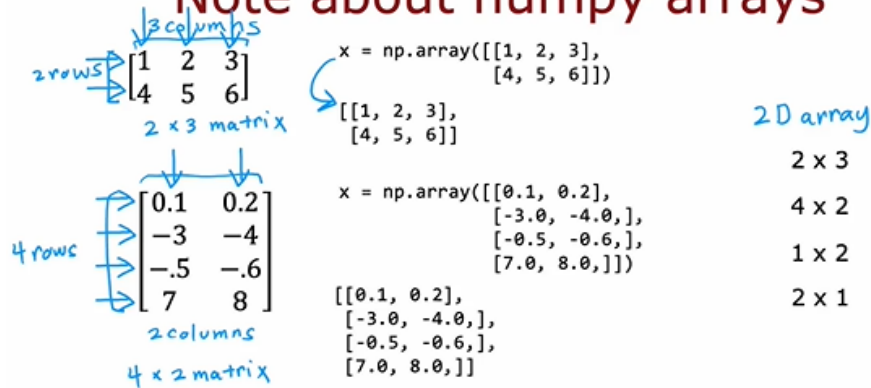
```
layer_2 = Dense(units=15, activation='sigmoid')
a2 = layer_2(a1)
```

```
layer_3 = Dense(units=1, activation='sigmoid')
a3 = layer_3(a2)
```

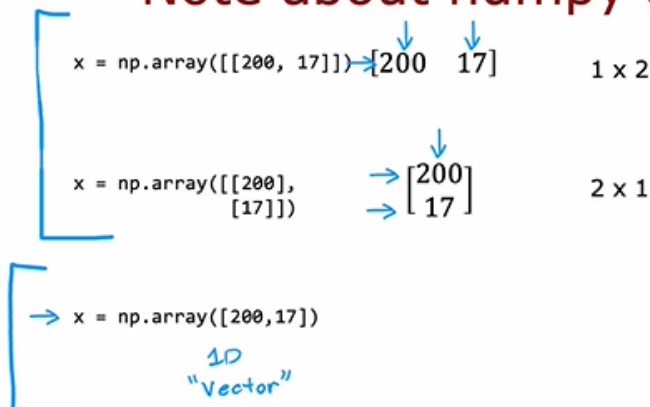
```
if a3 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

Data in Tensorflow

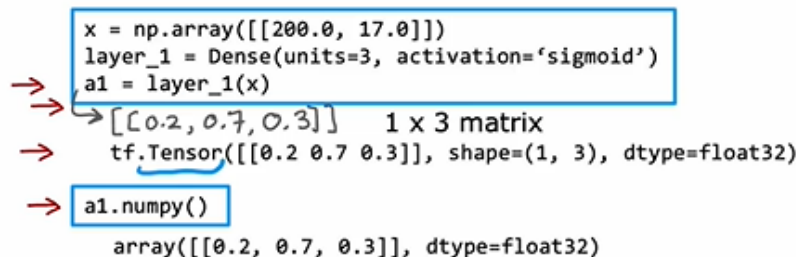
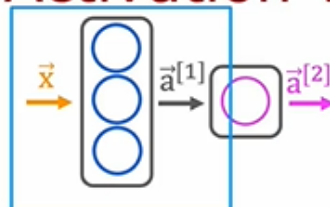
Note about numpy arrays



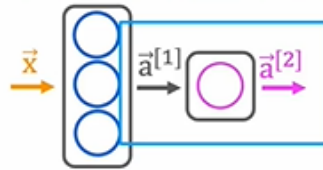
Note about numpy arrays



Activation vector



Activation vector



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

↖ $[[0.8]]$ ← 1×1

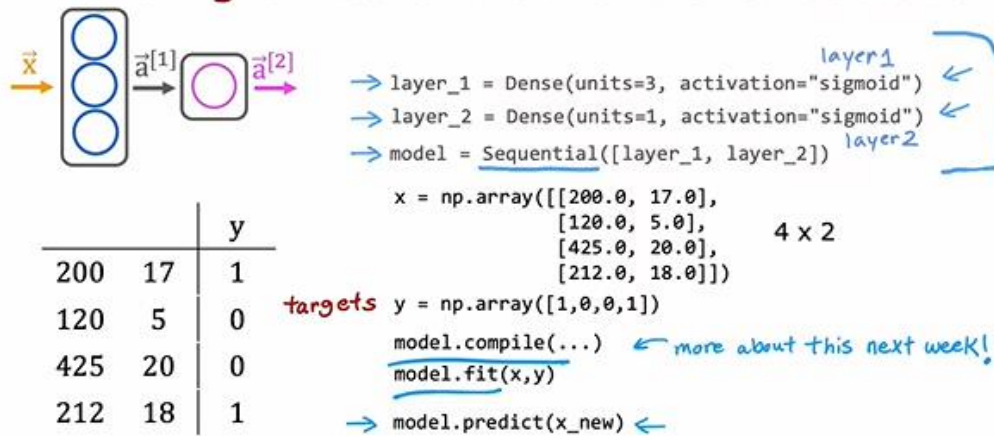
```
→ tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
→ a2.numpy()
```

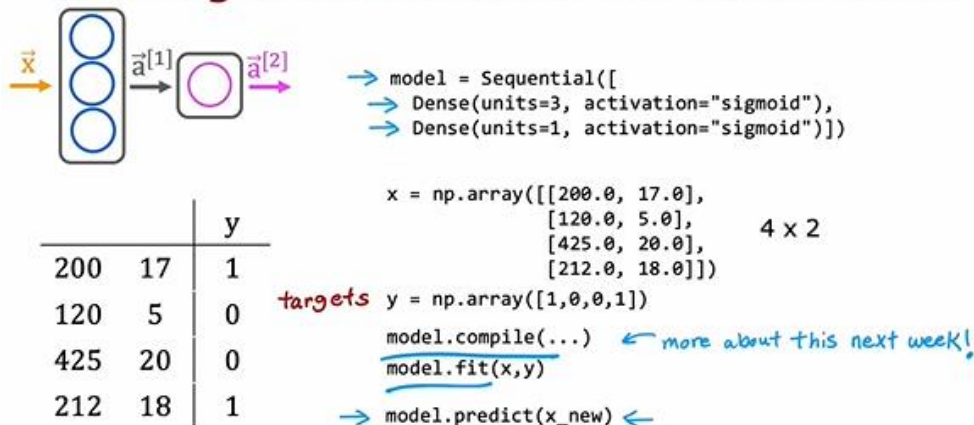
```
→ array([[0.8]], dtype=float32)
```


Building a Neural Network

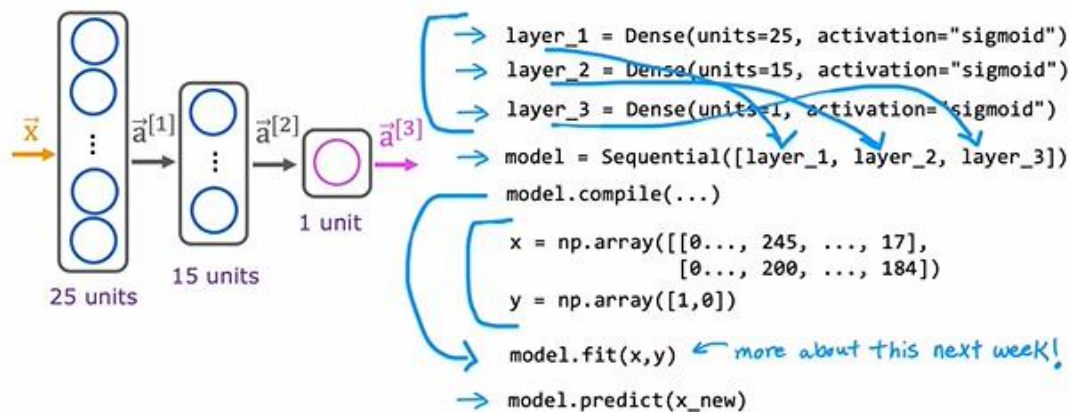
Building a neural network architecture



Building a neural network architecture



Digit classification model



1. For the the following code:

1 point

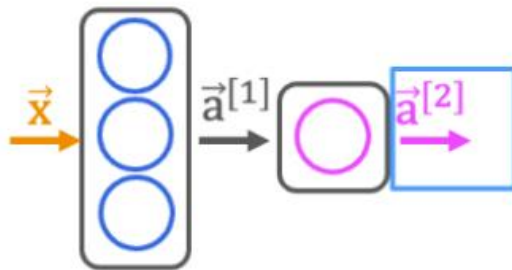
```
model = Sequential([
    Dense(units=25, activation="sigmoid"),
    Dense(units=15, activation="sigmoid"),
    Dense(units=10, activation="sigmoid"),
    Dense(units=1, activation="sigmoid")])
```

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)
```

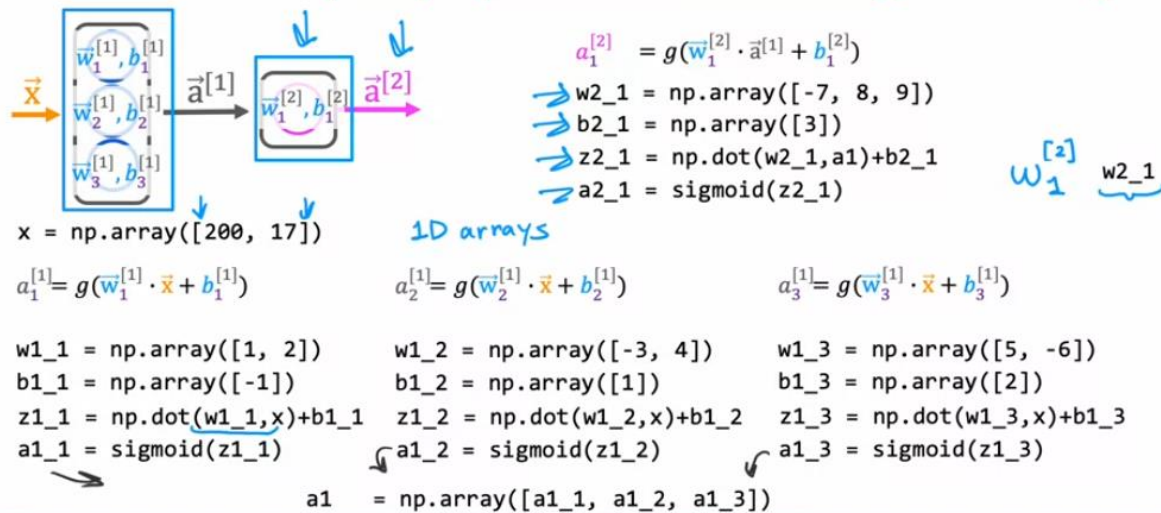
```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

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')`

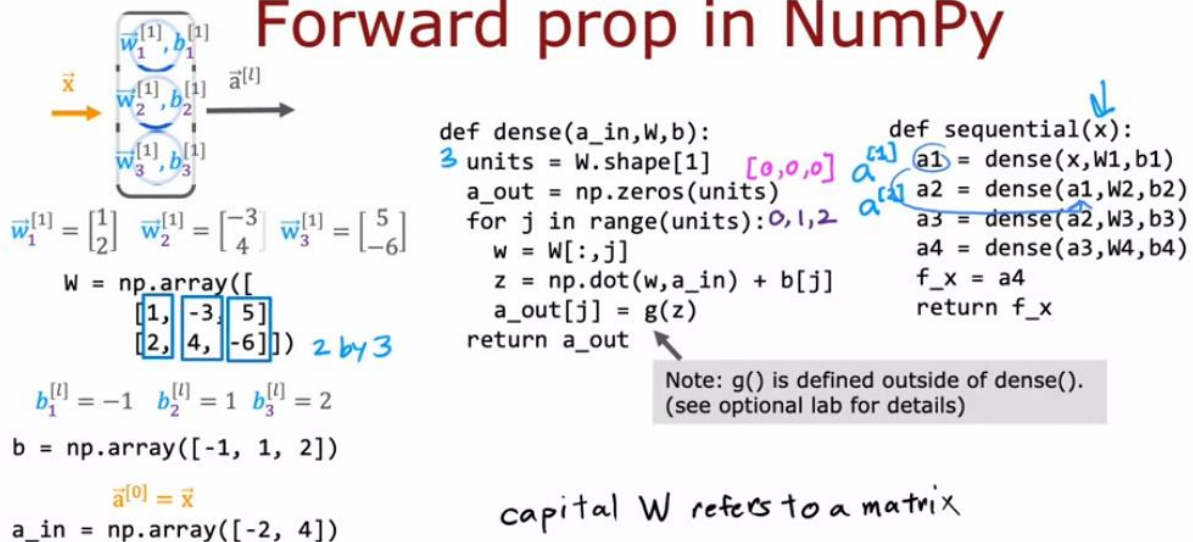
Forward prop in a single layer

forward prop (coffee roasting model)



General implementation of forward propagation

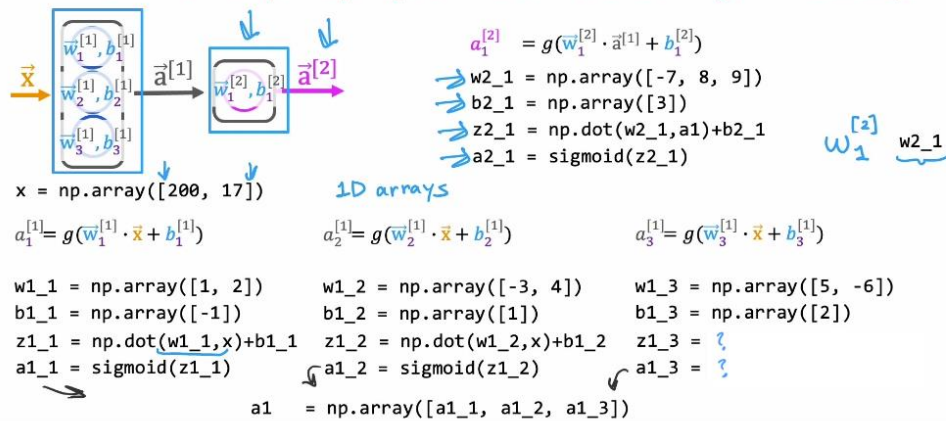
Forward prop in NumPy



1.

1 point

forward prop (coffee roasting model)



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)
```

☒

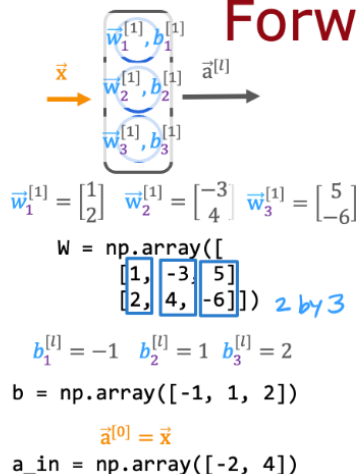
```
z1_3 = np.dot(w1_3, x) + b1_3
```

```
a1_3 = sigmoid(z1_3)
```

2.

1 point

Forward prop in NumPy



```
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
```

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.

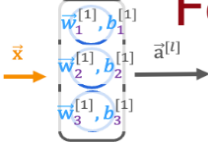
☒

In the columns of W.

3.

1 point

Forward prop in NumPy



\vec{x}

$\vec{a}^{[l]}$

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

$W = \text{np.array}(\begin{bmatrix} 1, -3, 5 \\ 2, 4, -6 \end{bmatrix})$ *2 by 3*

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

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

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

$a_{in} = \text{np.array}([-2, 4])$

```
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

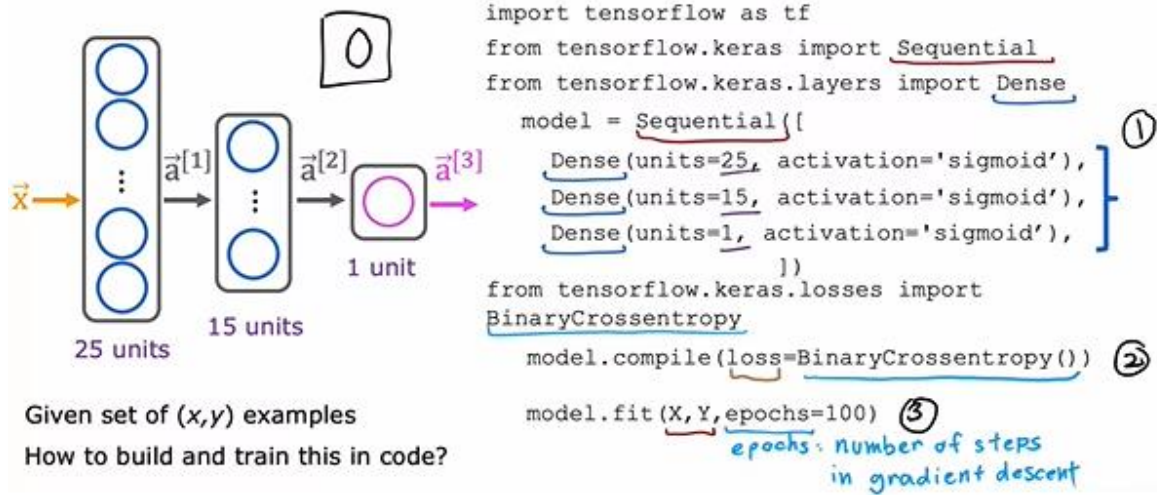
Coursera Honor Code [Learn more](#)



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

Train a Neural Network in TensorFlow



Training Details

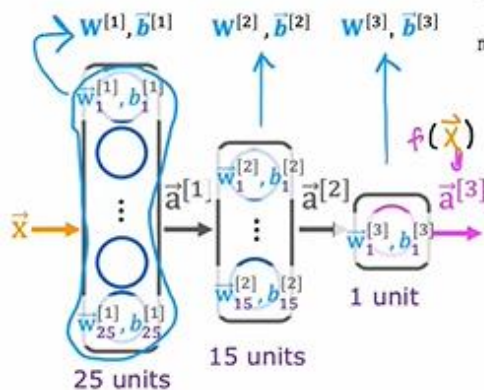
Model Training Steps TensorFlow

<p>① specify how to compute output given input x and parameters w, b (define model)</p> $f_{\bar{w}, b}(\bar{x}) = ?$ <p>② specify loss and cost</p> $L(f_{\bar{w}, b}(\bar{x}), y) \quad \text{1 example}$ $J(\bar{w}, b) = \frac{1}{m} \sum_{i=1}^m L(f_{\bar{w}, b}(\bar{x}^{(i)}), y^{(i)})$ <p>③ Train on data to minimize $J(\bar{w}, b)$</p>	<p>logistic regression</p> <pre>z = np.dot(w, x) + b f_x = 1 / (1 + np.exp(-z))</pre> <p>logistic loss</p> <pre>loss = -y * np.log(f_x) -(1-y) * np.log(1-f_x)</pre> <pre>w = w - alpha * dj_dw b = b - alpha * dj_db</pre>	<p>neural network</p> <pre>model = Sequential([Dense(...), Dense(...), Dense(...)])</pre> <p>binary cross entropy</p> <pre>model.compile(loss=BinaryCrossentropy())</pre> <pre>model.fit(X, y, epochs=100)</pre>
--	---	--

1. Create the model

define the model

$$f(\bar{x}) = ?$$



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

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

2. Loss and cost functions

handwritten digit classification problem → binary classification

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

Compare prediction vs. target

logistic loss

also known as binary cross entropy

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

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

model.compile(loss= BinaryCrossentropy())

regression

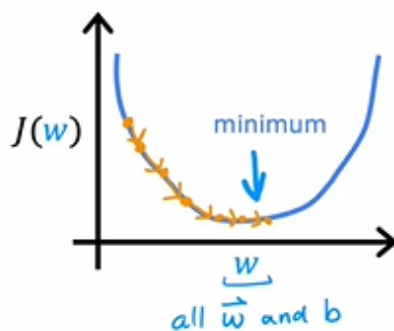
(predicting numbers and not categories) mean squared error

model.compile(loss= MeanSquaredError())

from tensorflow.keras.losses import
BinaryCrossentropy **K** Keras

from tensorflow.keras.losses import
MeanSquaredError

3. Gradient descent



repeat {

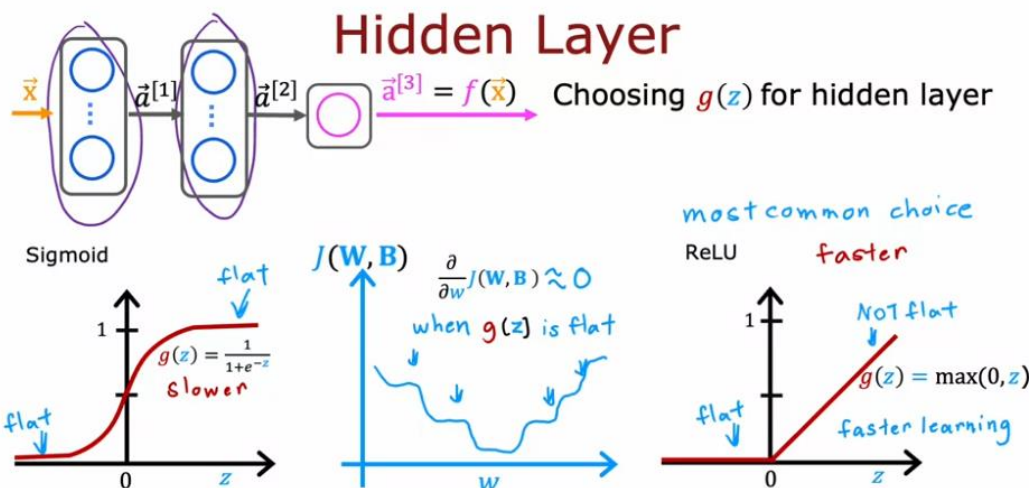
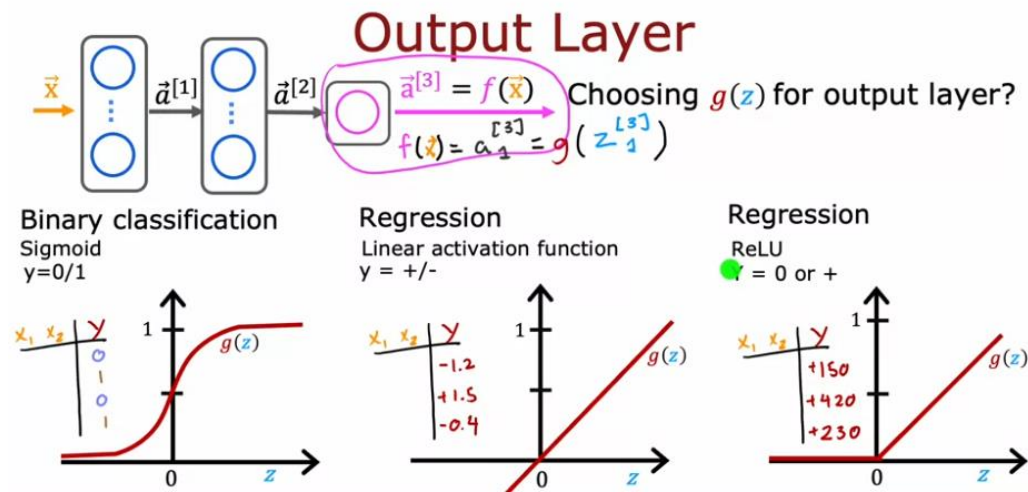
$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

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

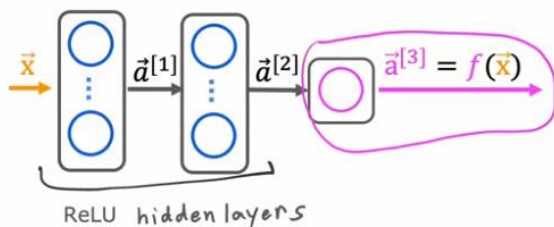
} Compute derivatives
for gradient descent
using "backpropagation"

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

Choosing activation function



Choosing Activation Summary



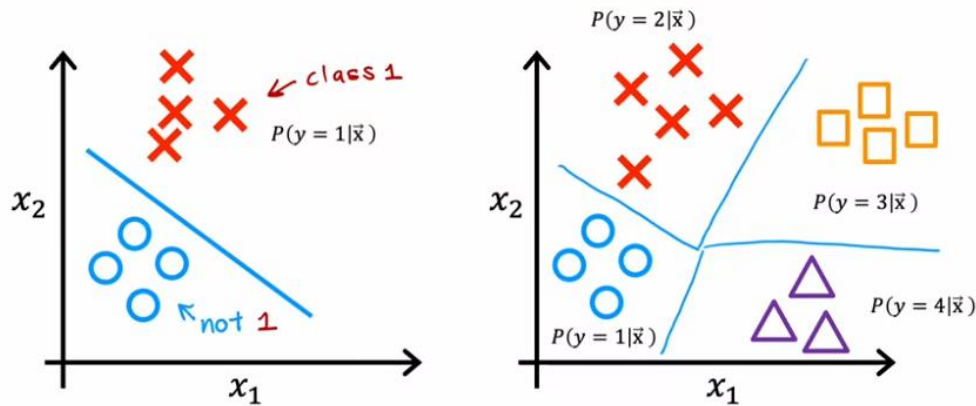
binary classification
 activation='sigmoid'
 regression y negative/
 activation='linear' positive
 regression $y \geq 0$
 activation='relu'

```
from tf.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu'), layer1
    Dense(units=15, activation='relu'), layer2
    Dense(units=1, activation='sigmoid') layer3
])
```

or 'linear'
or 'relu'

Multiclass

Multiclass classification example



Softmax

Logistic regression
(2 possible output values)
 $z = \vec{w} \cdot \vec{x} + b$

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

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

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

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

parameters w_1, w_2, \dots, w_N
 b_1, b_2, \dots, b_N

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

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

Softmax regression (4 possible outputs) $y=1, 2, 3, 4$

$$\times z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \quad a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1|\vec{x}) \quad 0.30$$

$$\circ z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \quad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=2|\vec{x}) \quad 0.20$$

$$\square z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \quad a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=3|\vec{x}) \quad 0.15$$

$$\triangle z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4|\vec{x}) \quad 0.35$$

Cost

Logistic regression

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

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

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

$$\text{loss} = -y \log a_1 - (1-y) \log(1-a_1)$$

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

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

Softmax regression

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

$$\vdots$$

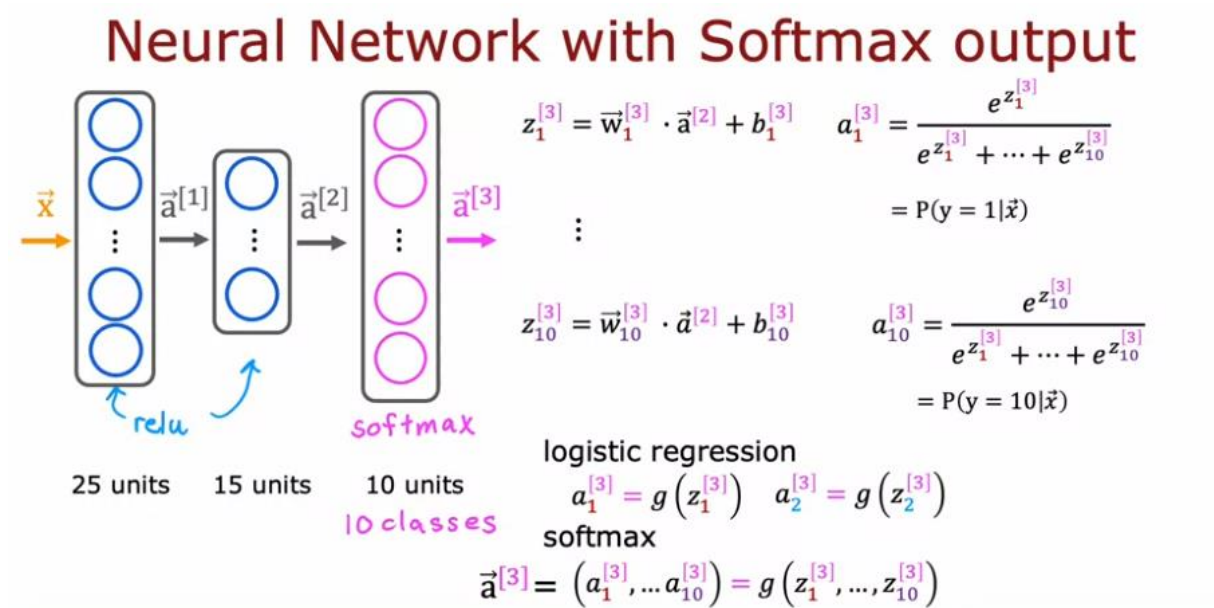
$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y=N|\vec{x})$$

Crossentropy loss

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

$\text{loss} = -\log a_j \text{ if } y=j$

Neural Network with Softmax output



MNIST with softmax

① specify the model

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

② specify loss and cost

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

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

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax')
])
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
model.fit(X,Y, epochs=100)
```

Note: better (recommended) version later.

Don't use the version shown here!

Improved implementation of softmax

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

$$\text{loss} = -y \log(\hat{a}) - (1 - y) \log(1 - \hat{a})$$

More accurate loss (in code)

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

logit: z

$1 + \frac{1}{10,000}$ $1 - \frac{1}{10,000}$
`model = Sequential([`
`Dense(units=25, activation='relu'),`
`Dense(units=15, activation='relu'), 'linear'`
`Dense(units=10, activation='sigmoid')`
`])`
~~`model.compile(loss=BinaryCrossEntropy())`~~
`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

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

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

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

MNIST (more numerically accurate)

```
model    import tensorflow as tf
         from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Dense
         model = Sequential([
             Dense(units=25, activation='relu'),
             Dense(units=15, activation='relu'),
             Dense(units=10, activation='linear') ])
loss     from tensorflow.keras.losses import
         SparseCategoricalCrossentropy
         model.compile(..., loss=SparseCategoricalCrossentropy(from_logits=True) )
fit      model.fit(X,Y,epochs=100)
predict  logits = model(X) ← not  $a_1 \dots a_{10}$ 
         f_x = tf.nn.softmax(logits)      is  $z_1 \dots$ 
```

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

Multi-label Classification



Is there a car? yes
Is there a bus? no
Is there a pedestrian yes

$$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

no
no
yes

$$y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

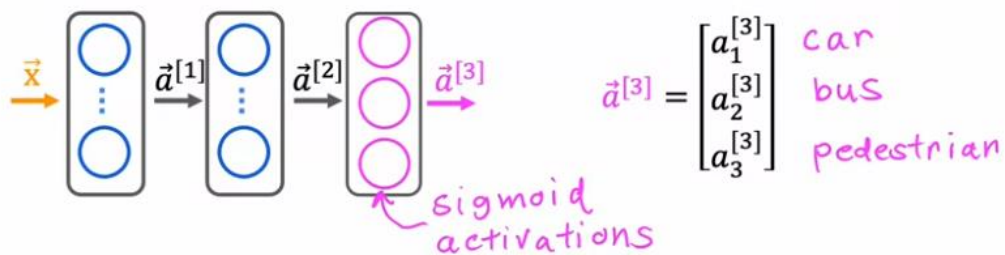
yes
yes
no

$$y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Multi-label Classification



Alternatively, train one neural network with three outputs

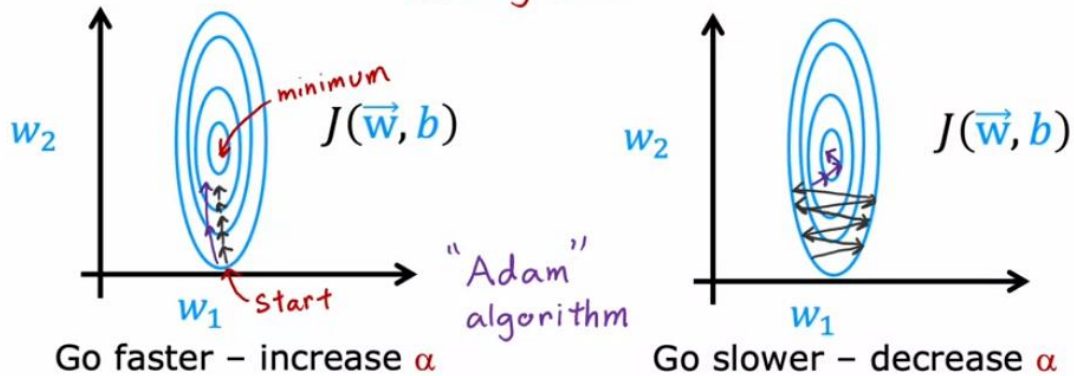


Advanced Optimization

Gradient Descent

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

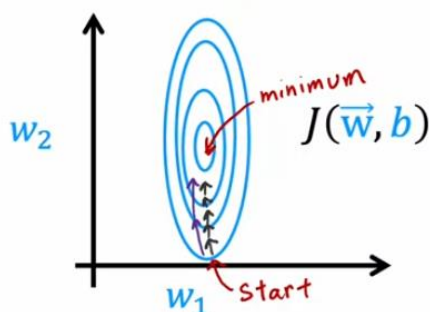
learning rate



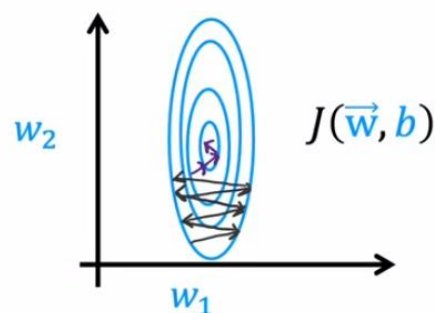
Adam Algorithm Intuition

Adam: Adaptive Moment estimation *not just one α*

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



If w_j (or b) keeps moving in same direction, increase α_j .



If w_j (or b) keeps oscillating, reduce α_j .

MNIST Adam

model

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

compile

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

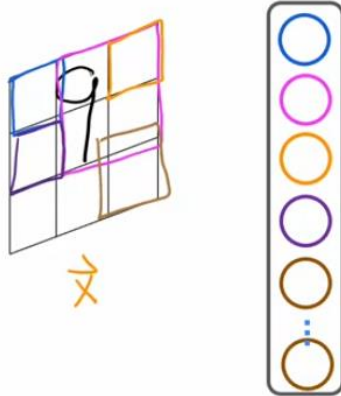
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),  
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

Additional Layer Types

Convolutional Layer



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

Why?

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

Convolutional Neural Network

