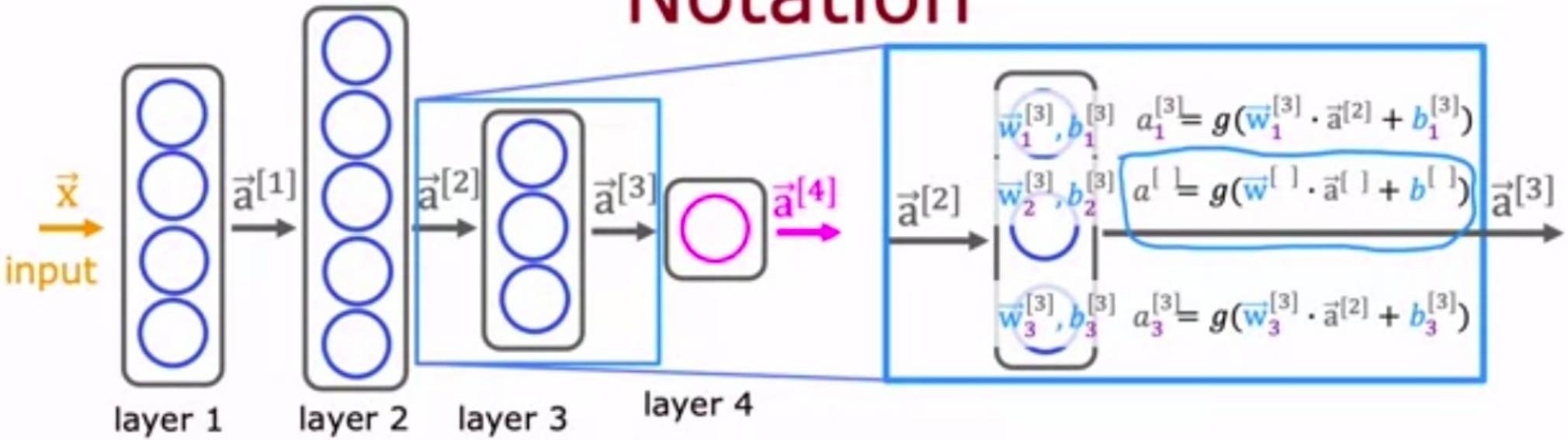
Week 1

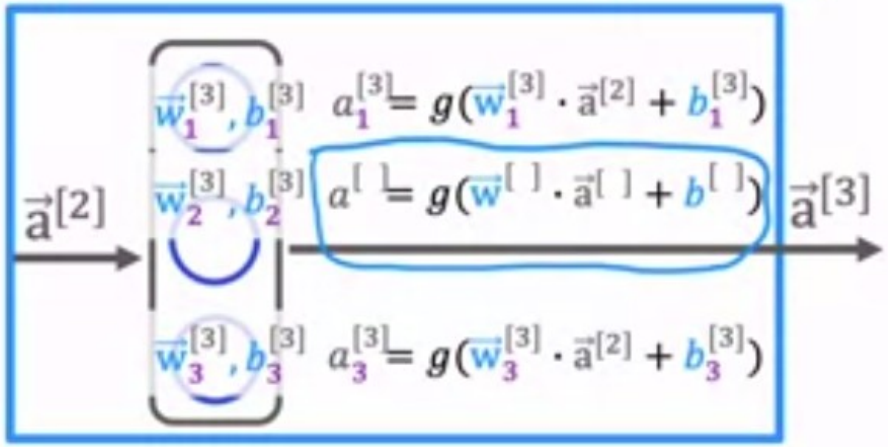
Notation



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

subscripts in this equation
and fill them in yourself?

[In Video Quiz]



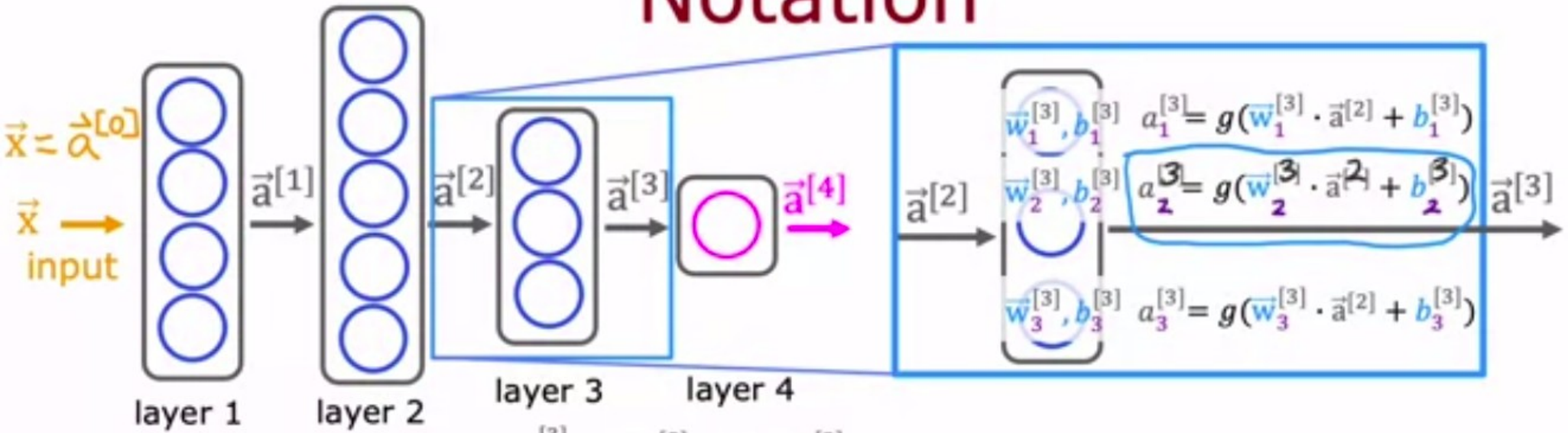
Can you fill in the superscripts and subscripts for the second neuron?

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

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

✗ $a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot a_2^{[2]} + b_2^{[3]})$ input is a vector $\vec{a}^{[2]}$ not a single number

Notation



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

Activation value of layer l, unit(neuron) j

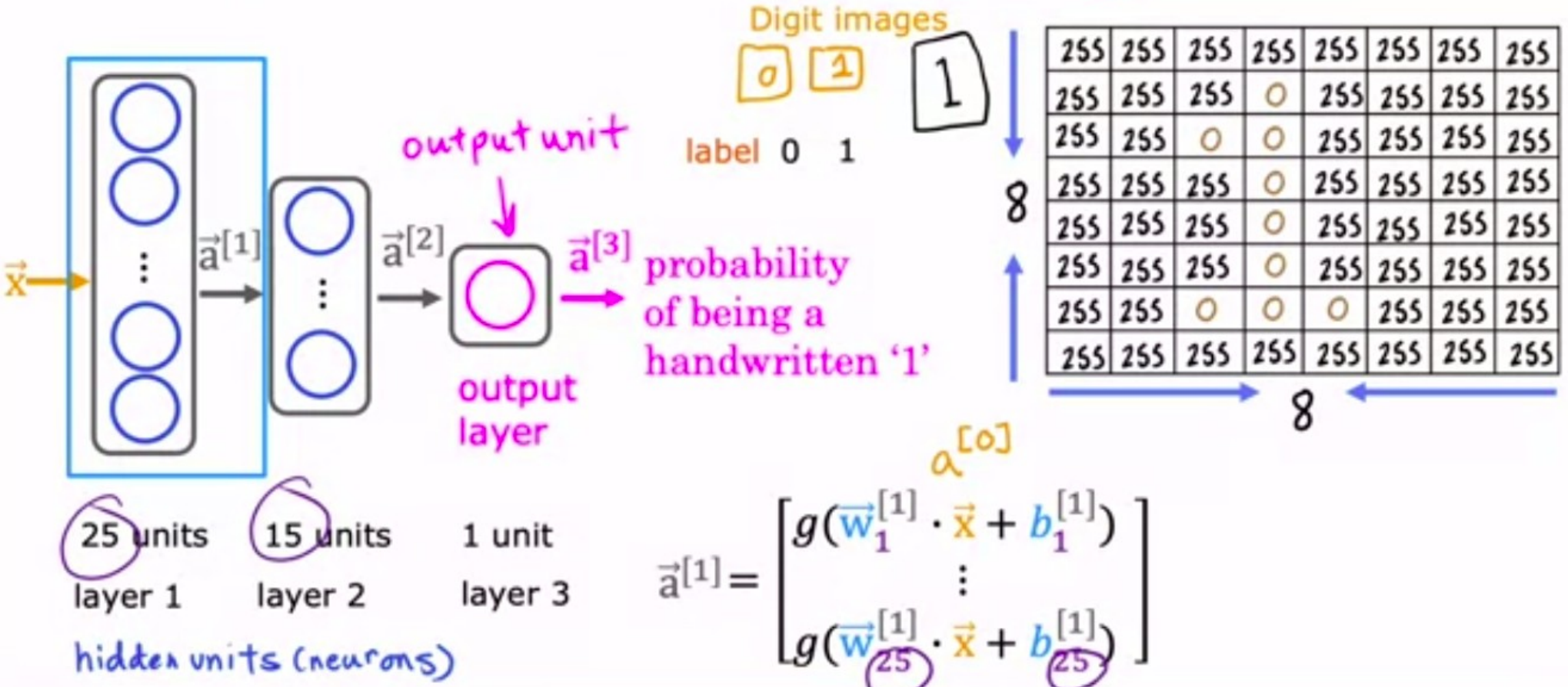
output of layer $l-1$ (previous layer)

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

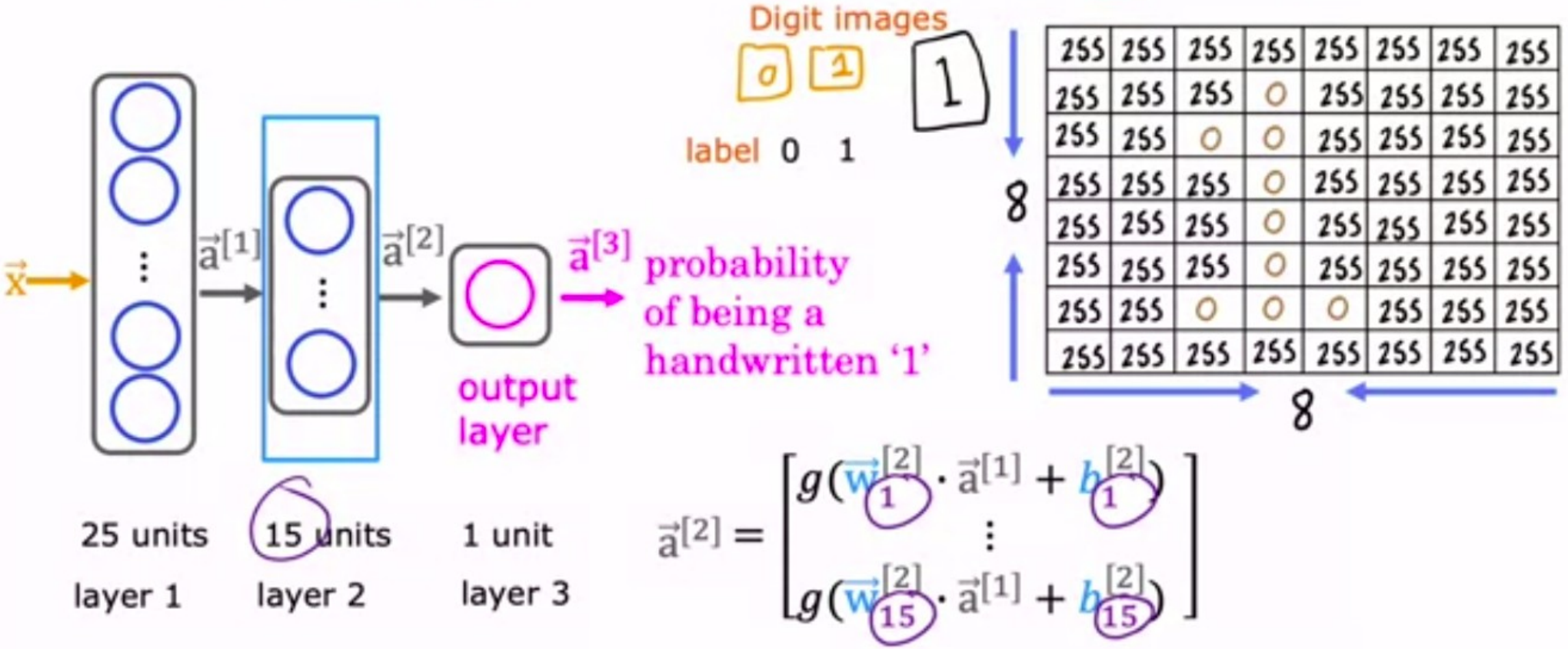
sigmoid "activation function"

Parameters w & b of layer l, unit j

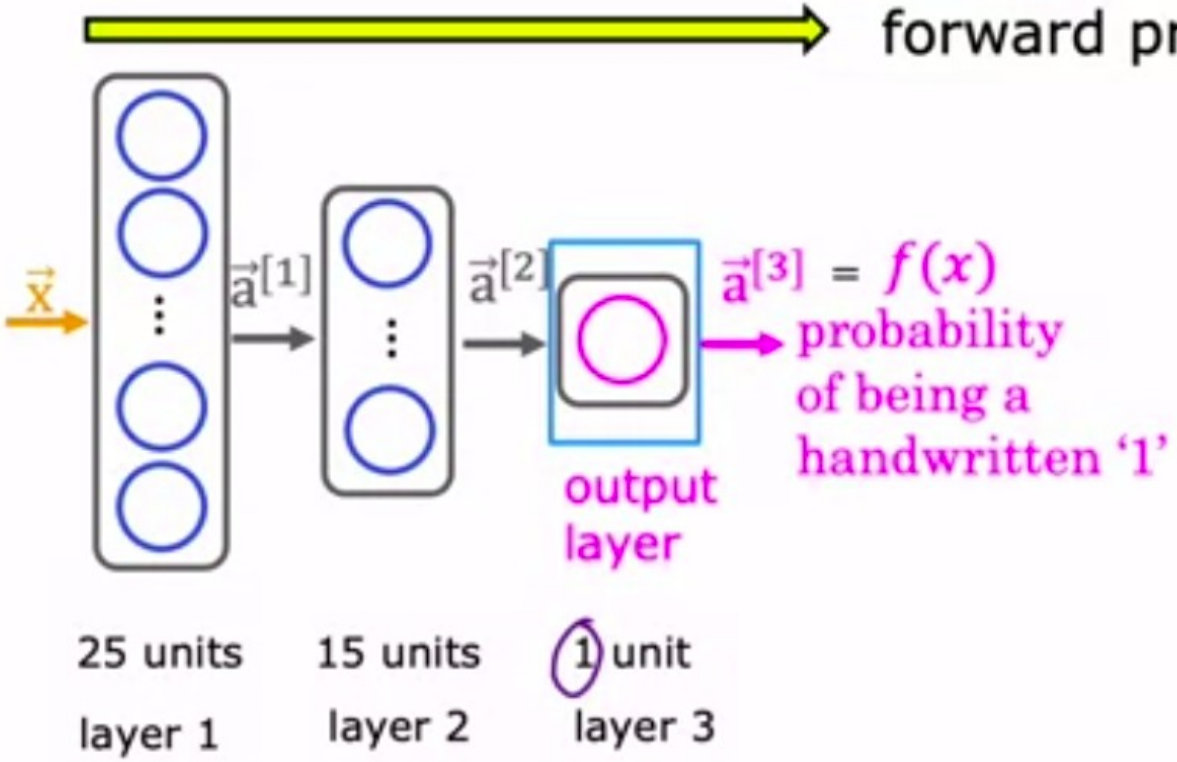
Handwritten digit recognition



Handwritten digit recognition



Handwritten digit recognition



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

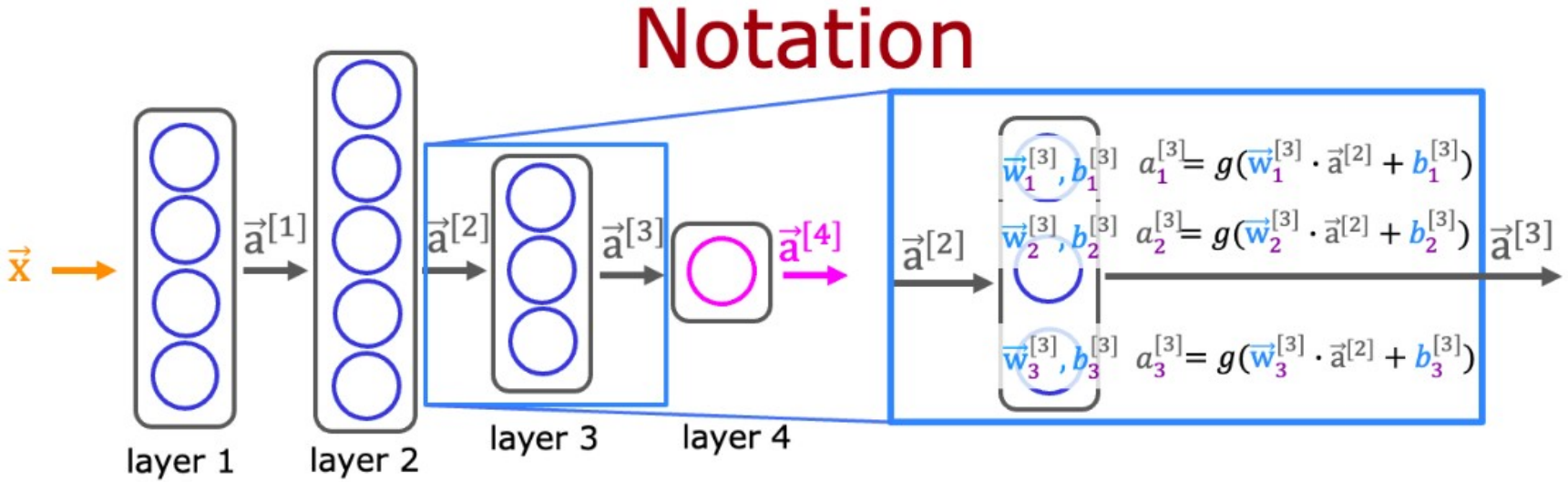
is $a_1^{[3]} \geq 0.5$?

yes → $\hat{y} = 1$ image is digit 1

no → $\hat{y} = 0$ image isn't digit 1

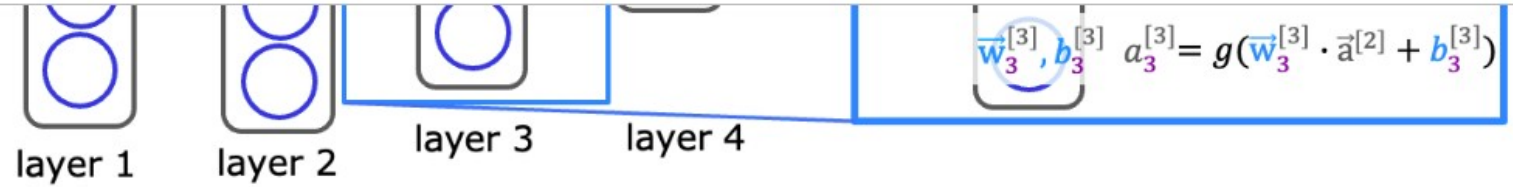
1.

1 / 1 point



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

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



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

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

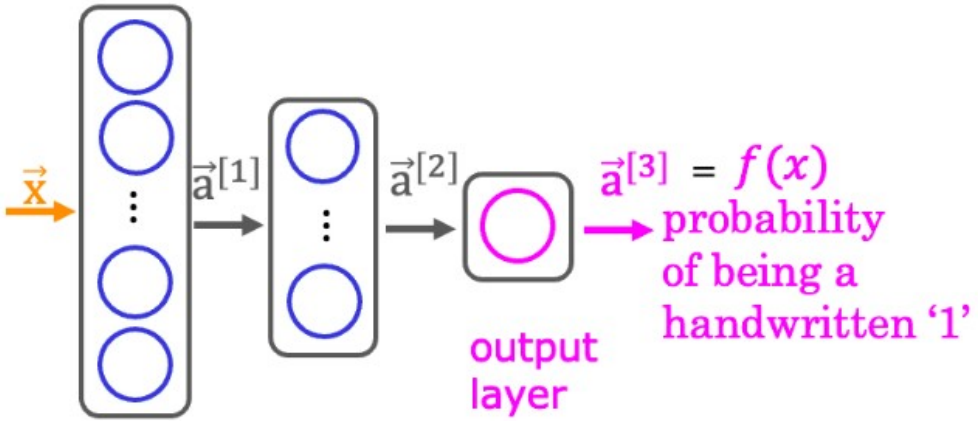
- $a_3^{[2]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[1]} + b_2^{[3]})$
- $a_3^{[2]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
- **(selected)** $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[1]} + b_3^{[2]})$
- $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[2]} + b_3^{[2]})$

Correct
Yes! The superscript [2] refers to layer 2. The subscript 3 refers to the neuron in that layer. The input to layer 2 is the activation vector from layer 1.

2.

1 / 1 point

Handwritten digit recognition



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

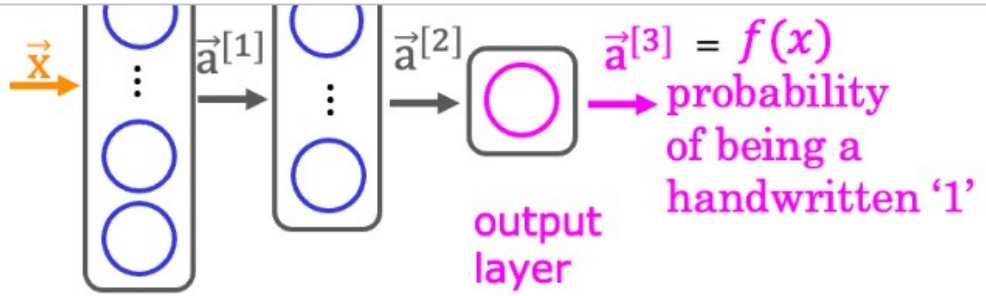
$$is\ a_1^{[3]} \geq 0.5?$$

yes → $\hat{y} = 1$: image is digit 1

no → $\hat{y} = 0$: image isn't digit 1

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

- 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



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

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

yes → $\hat{y} = 1$ image is digit 1

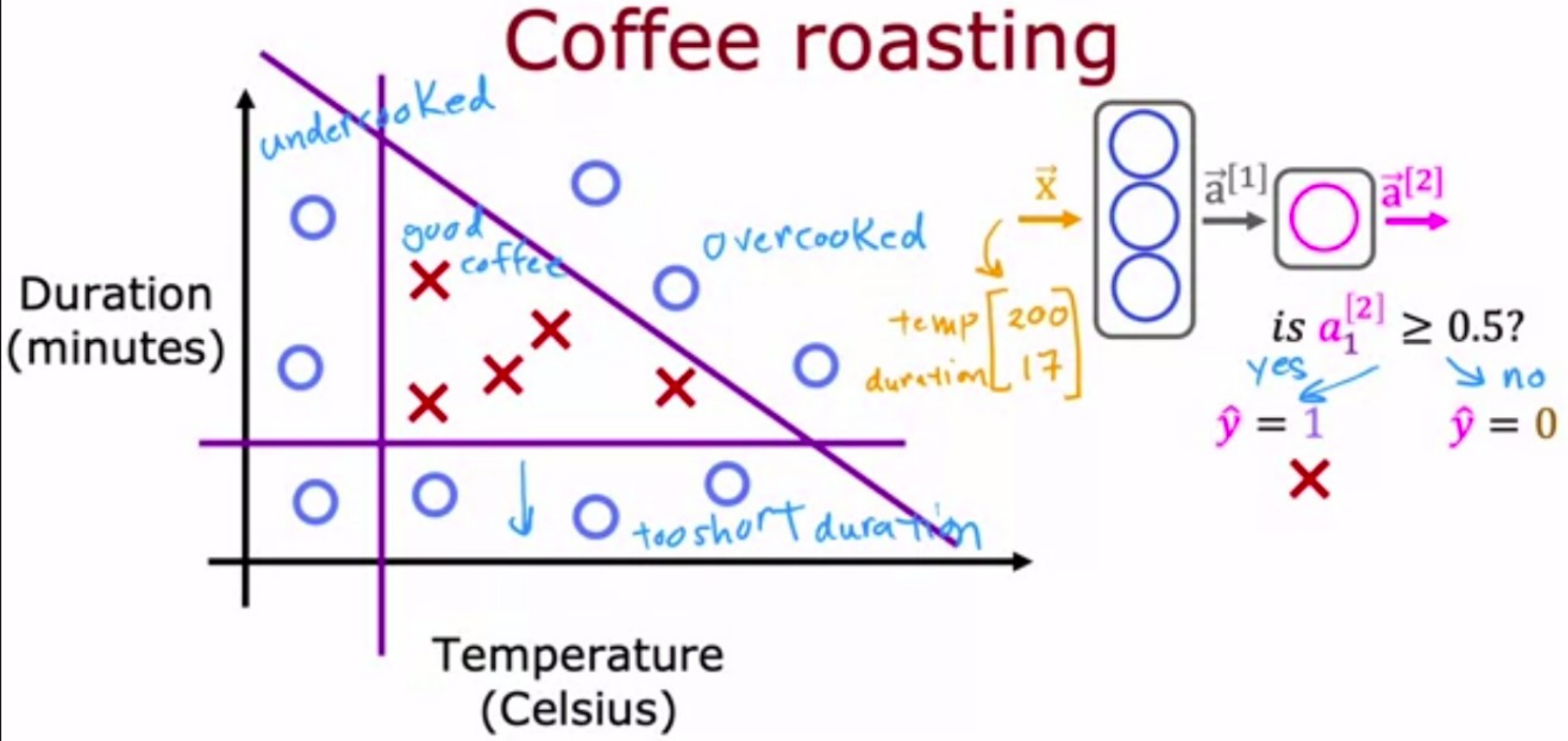
no → $\hat{y} = 0$ image isn't digit 1

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

- 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
- A vector of several numbers, each of which is either exactly 0 or 1
- (selected) The estimated probability that the input image is of a number 1, a number that ranges from 0 to 1.

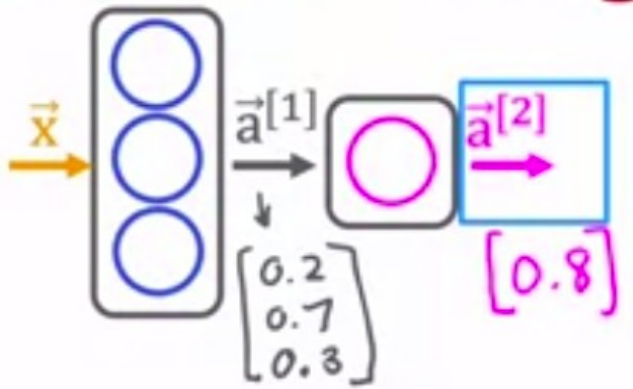
Correct
Yes! The neural network outputs a single number between 0 and 1.

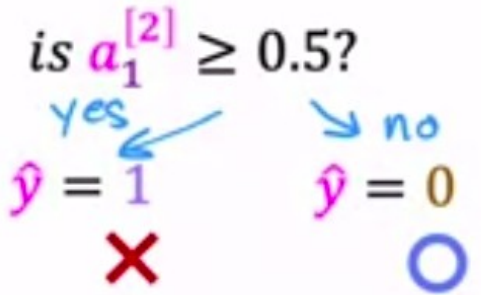
Coffee roasting



It looks like this.

Build the model using TensorFlow





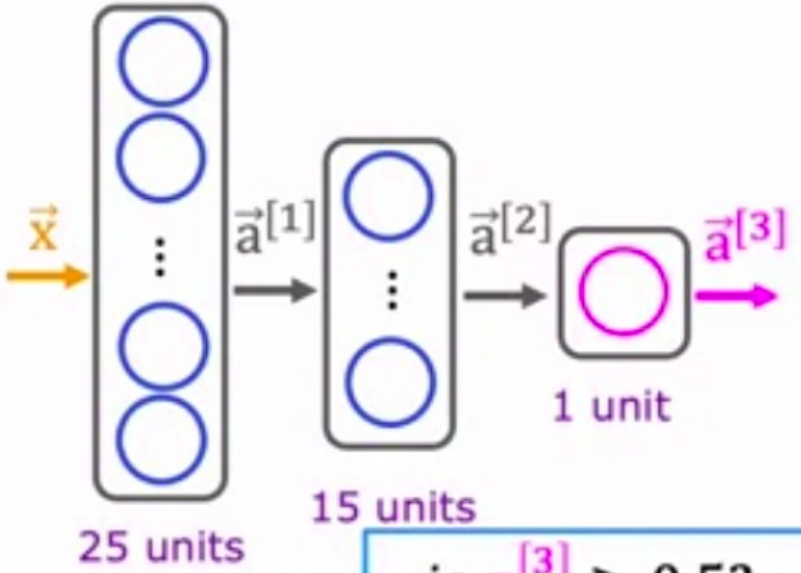

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

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

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

the handwritten digit classification problem.

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)
```

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

$\hat{y} = 1$ ✕ $\hat{y} = 0$ ○

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

a3 to come up with a binary prediction for y-hat.

Feature vectors

temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

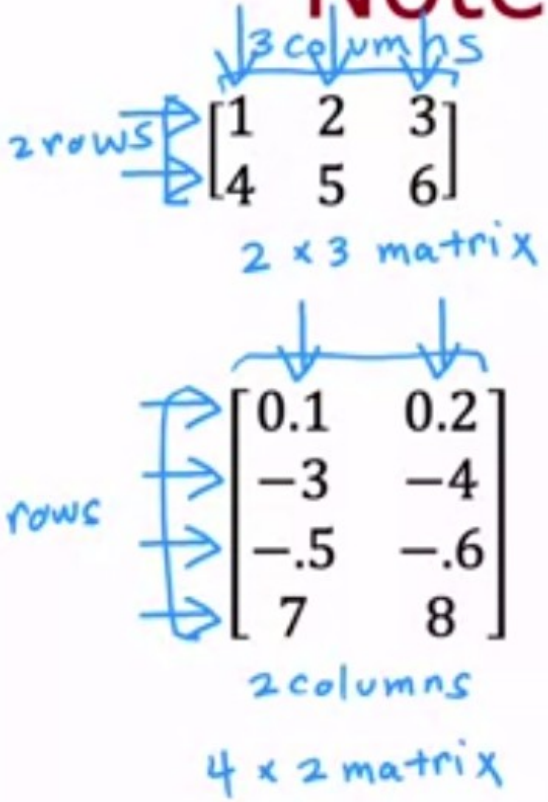
```
x = np.array([[200.0, 17.0]])
```

```
[[200.0, 17.0]]
```

Why?

you need to do with matrices and vectors
in order to implement your networks.

Note about numpy arrays




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

```
[[1, 2, 3],
 [4, 5, 6]]
```

```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0,],
              [-0.5, -0.6,],
              [7.0, 8.0,]])
```

```
[[0.1, 0.2],
 [-3.0, -4.0,],
 [-0.5, -0.6,],
 [7.0, 8.0,]]
```

2D array

2 x 3

4 x 2

1 x 2

2 x 1

And we'll see examples of these on the next slide.

Note about numpy arrays

```
x = np.array([[200, 17]])
```

$$\begin{bmatrix} 200 & 17 \end{bmatrix}$$

1 x 2

```
x = np.array([[200],
              [17]])
```

$$\begin{bmatrix} 200 \\ 17 \end{bmatrix}$$

2 x 1

```
x = np.array([200,17])
```

1D
"Vector"

it lets TensorFlow be a bit more computationally efficient internally.

Feature vectors

temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

```
x = np.array([[200.0, 17.0]])
```

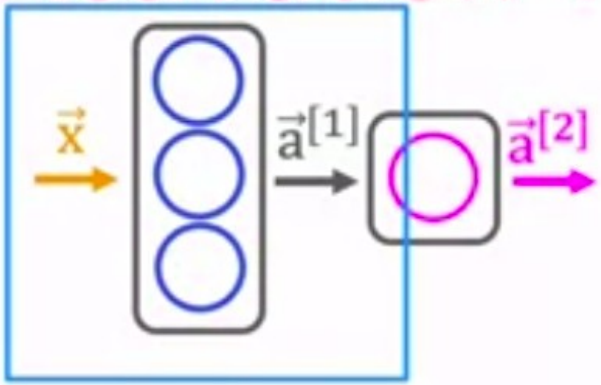
```
[[200.0, 17.0]]
```

1 x 2

→ [200.0 17.0]

And in case this seems like a lot of details and really complicated

Activation vector




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

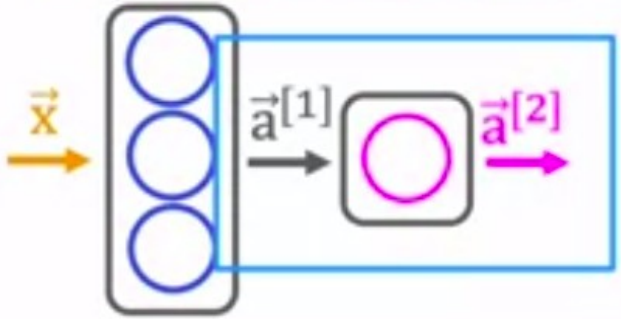
[[0.2, 0.7, 0.3]] 1 x 3 matrix

```
tf.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)
```

```
a1.numpy()
```

```
array([[0.2, 0.7, 0.3]], dtype=float32)
```

Activation vector




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

[[0.8]]

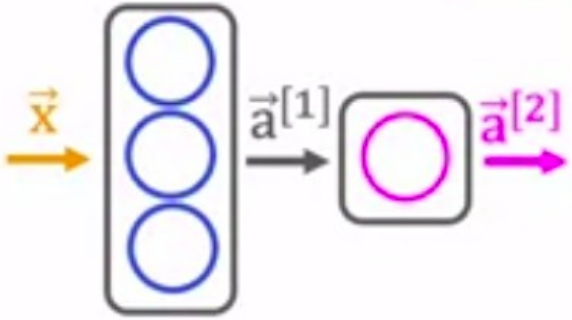
1 x 1

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

```
a2.numpy()
```

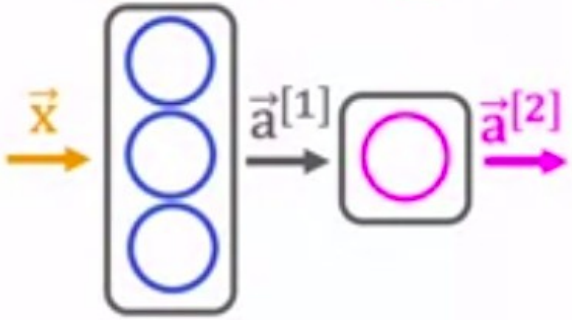
```
array([[0.8]], dtype=float32)
```

What you saw earlier




```
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)
```

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]])
```

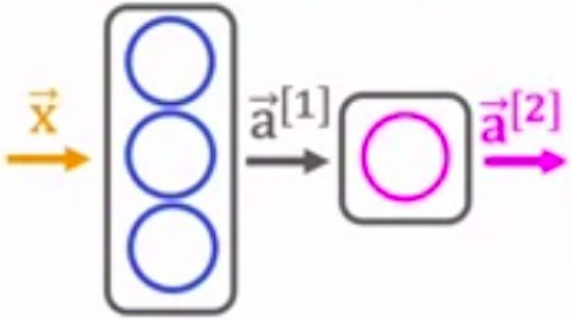
4 x 2

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]])
```

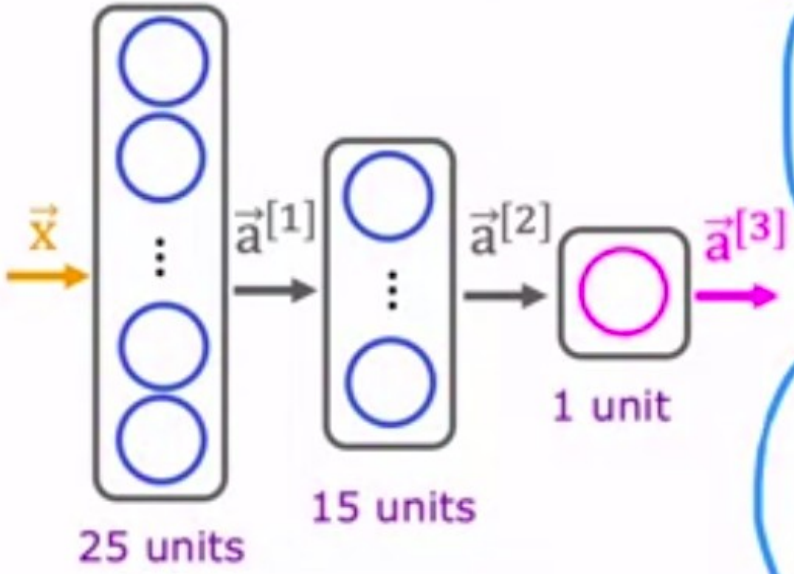
4 x 2

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!

Practice quiz: TensorFlow implementation

Graded Quiz • 10 min

Due Oct 2, 11:59 PM IST

1. For the the following code: **1 / 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?

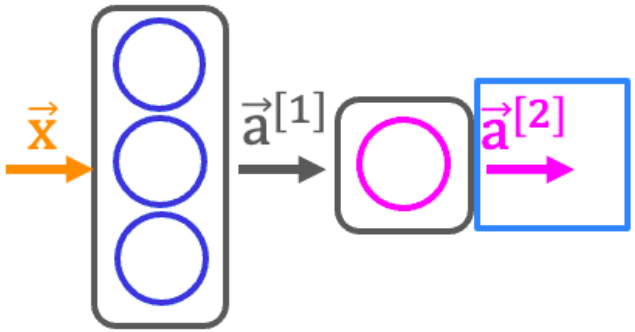
- ○ 3
- ○ 25
- ◉ 4
- ○ 5

Correct

Yes! Each call to the "Dense" function defines a layer of the neural network.

2.

1 / 1 point



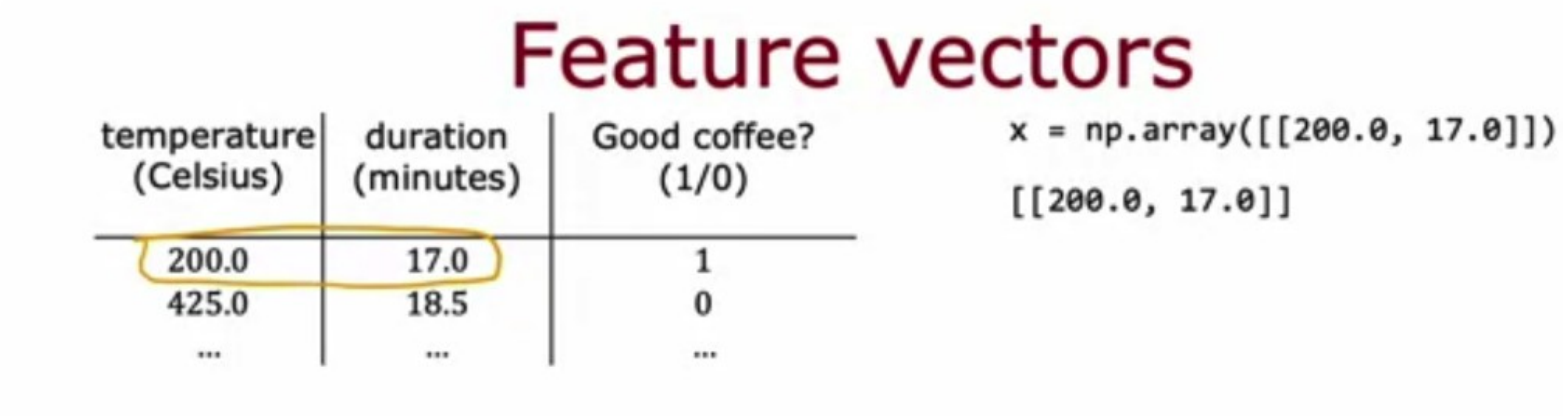

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)


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

3.

1 / 1 point



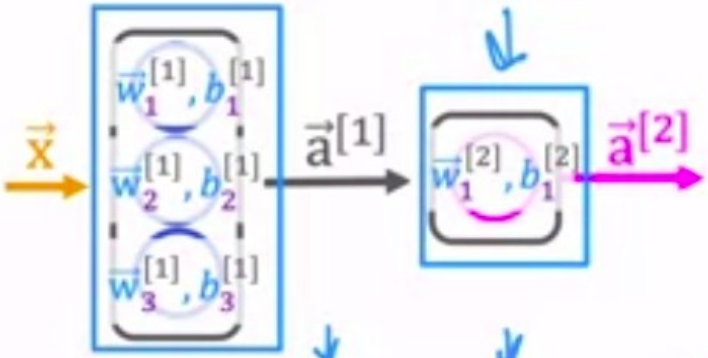
If the input features are temperature (in Celsius) and duration (in minutes), how do you write the code for the first feature vector x shown above?

- x = np.array([[200.0],[17.0]])
- x = np.array([['200.0', '17.0']])
- x = np.array([[200.0 + 17.0]])
- x = np.array([[200.0, 17.0]]) (selected)

Correct

Yes! A row contains all the features of a training example. Each column is a feature.

forward prop (coffee roasting model)



$w_1^{[2]}$ w2_1

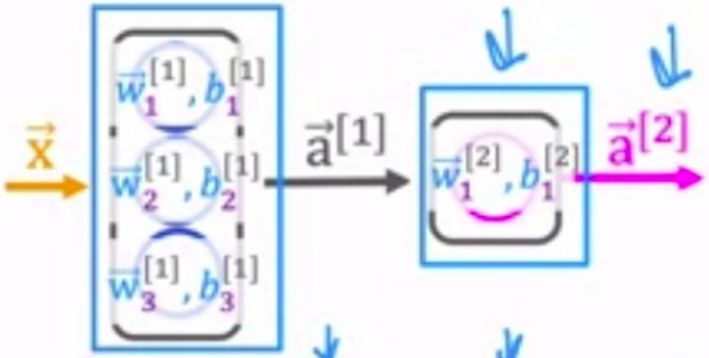
```
x = np.array([200, 17])
```

1D arrays

$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$ $a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$ $a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1,x)+b1_1
a1_1 = sigmoid(z1_1)

w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b1_2
a1_2 = sigmoid(z1_2)

w1_3 = np.array([5, -6])
b1_3 = np.array([2])
z1_3 = np.dot(w1_3,x)+b1_3
a1_3 = sigmoid(z1_3)

a1 = np.array([a1_1, a1_2, a1_3])
```

forward prop (coffee roasting model)



$$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

```
w2_1 = np.array([-7, 8, 9])
b2_1 = np.array([3])
z2_1 = np.dot(w2_1,a1)+b2_1
a2_1 = sigmoid(z2_1)
```

$w_1^{[2]}$ w2_1

```
x = np.array([200, 17])
```

1D arrays

$$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$$

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1,x)+b1_1
a1_1 = sigmoid(z1_1)
```

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

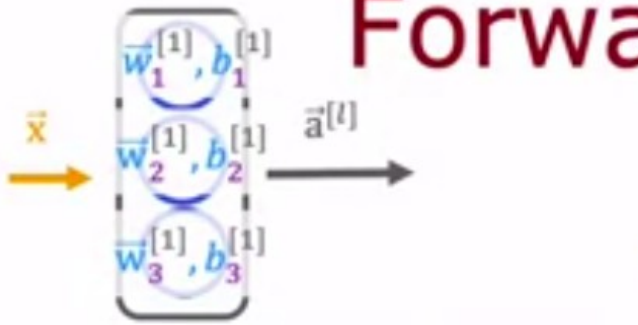
```
w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b1_2
a1_2 = sigmoid(z1_2)
```

$$a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$$

```
w1_3 = np.array([5, -6])
b1_3 = np.array([2])
z1_3 = np.dot(w1_3,x)+b1_3
a1_3 = sigmoid(z1_3)
```

```
a1 = np.array([a1_1, a1_2, a1_3])
```

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

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

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

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

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

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

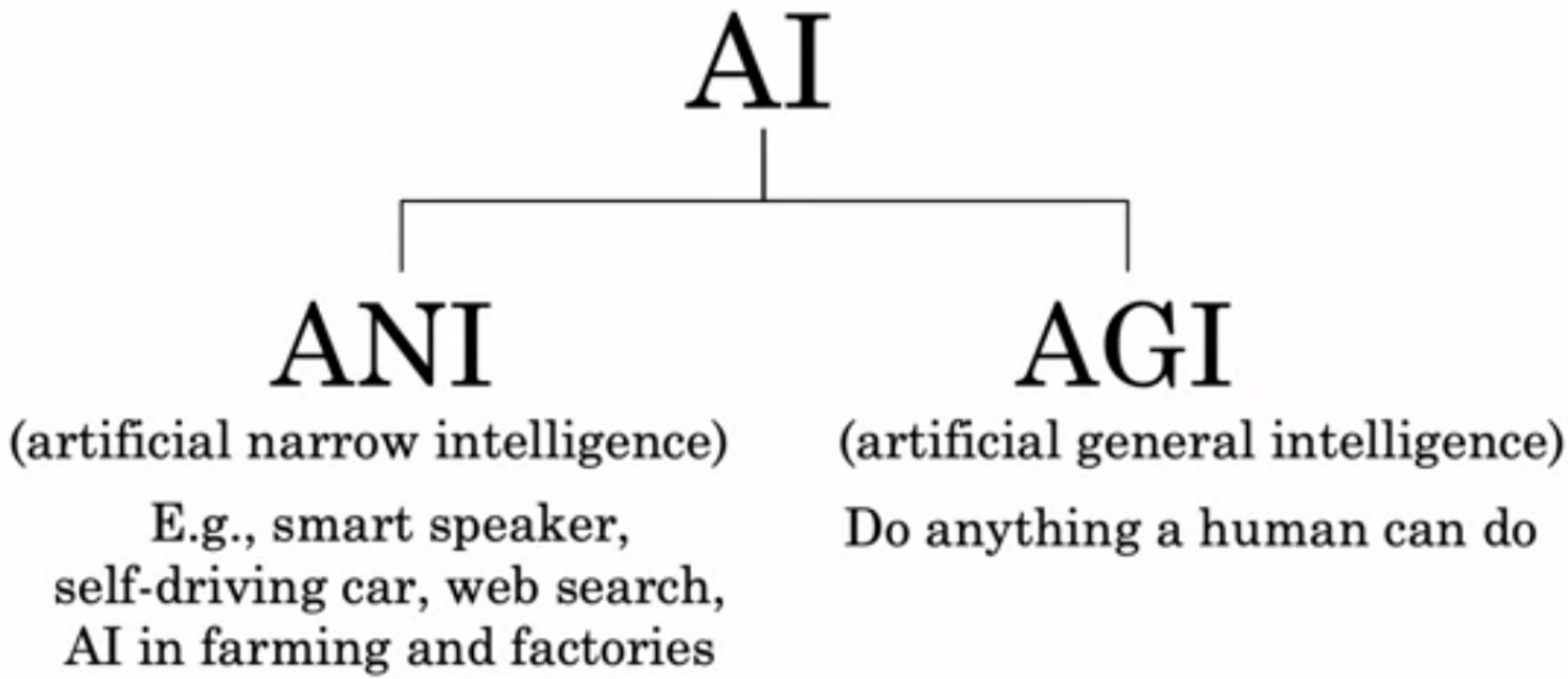
3 [0,0,0] 0,1,2

Note: g() is defined outside of dense().
(see optional lab for details)

```
def sequential(x):
    a1 = dense(x,W1,b1)
    a2 = dense(a1,W2,b2)
    a3 = dense(a2,W3,b3)
    a4 = dense(a3,W4,b4)
    f_x = a4
    return f_x
```

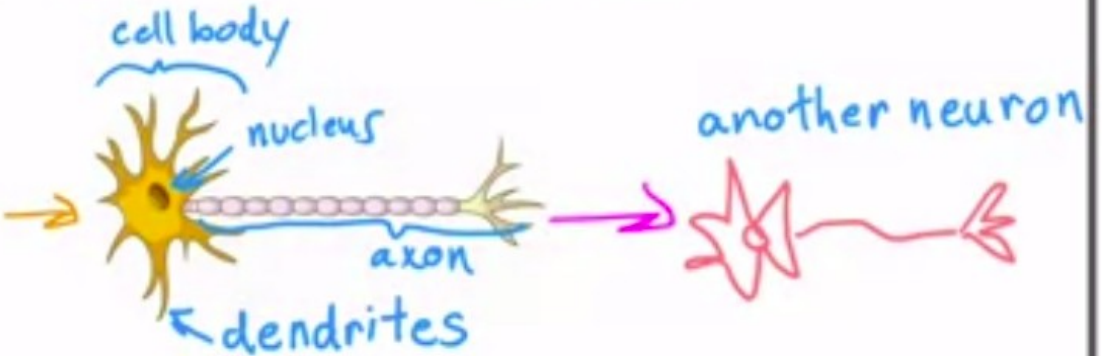
$a^{[1]}$ $a^{[2]}$

capital W refers to a matrix


AI
ANI
(artificial narrow intelligence)
E.g., smart speaker,
self-driving car, web search,
AI in farming and factories
AGI
(artificial general intelligence)
Do anything a human can do

Biological neuron

inputs outputs



Simplified mathematical model of a neuron

inputs outputs

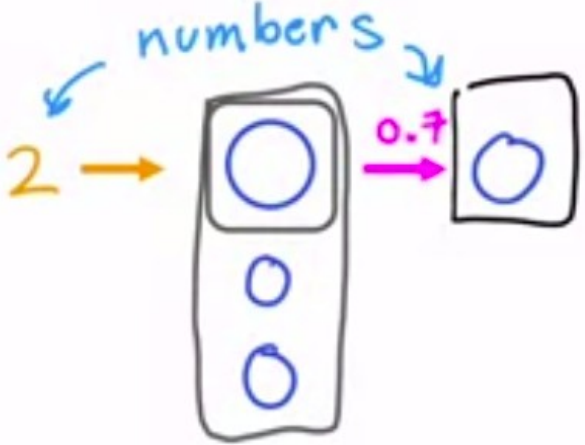
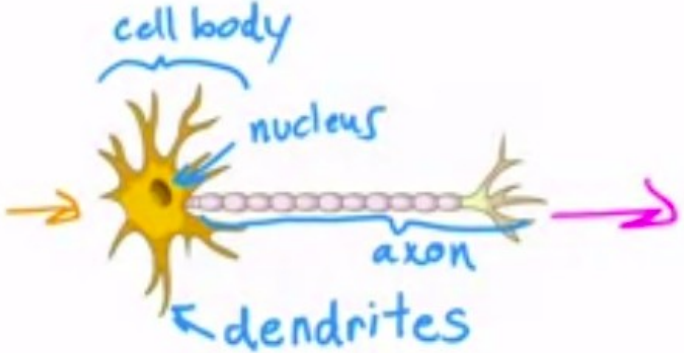


image source: https://biologydictionary.net/sensory-neuron/

Neural network and the brain

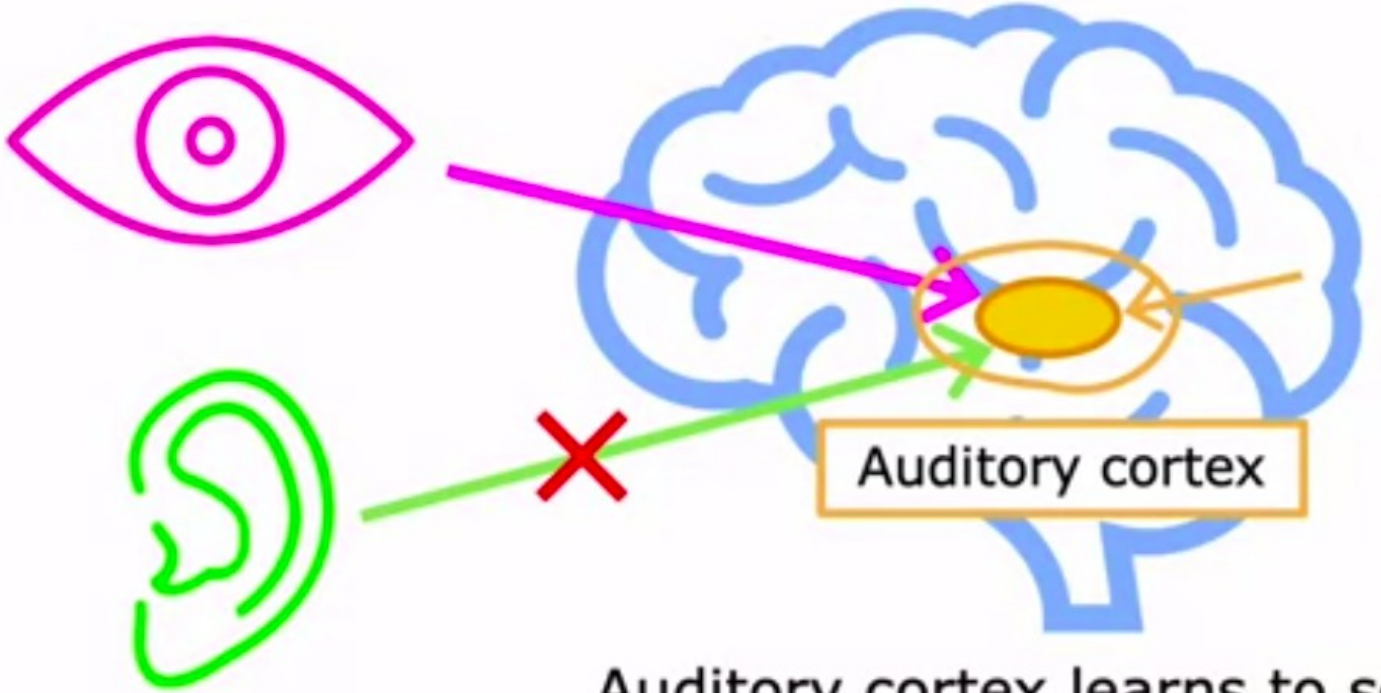
Can we mimic the human brain?



vs

We have (almost) no idea how the brain works

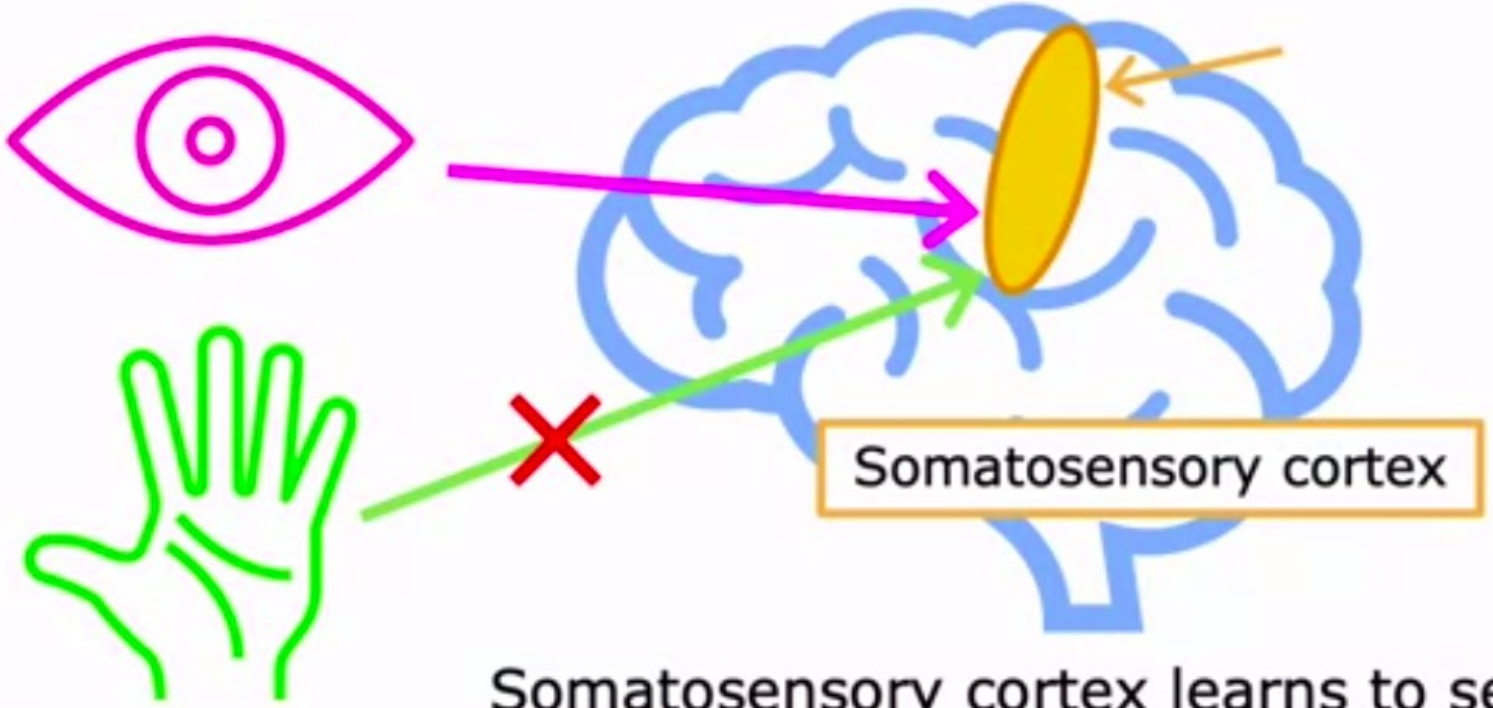
The "one learning algorithm" hypothesis



Auditory cortex learns to see

[Roe et al., 1992]

The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

[Metin & Frost, 1989]

Sensor representations in the brain

Seeing with your tongue

Human echolocation (sonar)

Haptic belt: Direction sense

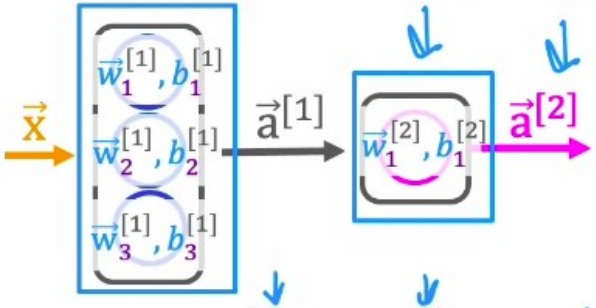
Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

1.

1 / 1 point

forward prop (coffee roasting model)



$$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

```
w2_1 = np.array([-7, 8, 9])
b2_1 = np.array([3])
z2_1 = np.dot(w2_1,a1)+b2_1
a2_1 = sigmoid(z2_1)
```

$w_1^{[2]}$ w2_1

```
x = np.array([200, 17])
```

1D arrays

$$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \qquad a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}) \qquad a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$$

```
w1_1 = np.array([1, 2])        w1_2 = np.array([-3, 4])       w1_3 = np.array([5, -6])
b1_1 = np.array([-1])          b1_2 = np.array([1])           b1_3 = np.array([2])
z1_1 = np.dot(w1_1,x)+b1_1     z1_2 = np.dot(w1_2,x)+b1_2     z1_3 = ?
a1_1 = sigmoid(z1_1)           a1_2 = sigmoid(z1_2)           a1_3 = ?

a1 = np.array([a1_1, a1_2, a1_3])
```

```
a1   = np.array([a1_1, a1_2, a1_3])
```

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

- (selected)

```
z1_3 = np.dot(w1_3, x) + b1_3
a1_3 = sigmoid(z1_3)
```

-

```
z1_3 =w1_3 * x + b
a1_3 = sigmoid(z1_3)
```

-

```
layer_1 = Dense(units=3, activation='sigmoid')
a_1 = layer_1(x)
```

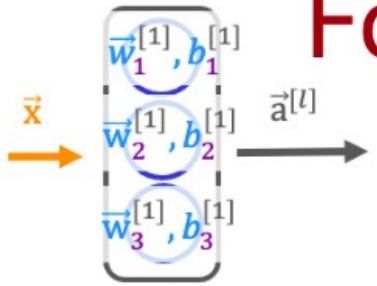
Correct

Correct. Use the numpy.dot function to take the dot product. The sigmoid function shown in lecture can be a function that you write yourself (see course 1, week 3 of this specialization), and that will be provided to you in this course.

2.

1 / 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

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

```
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
```

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

```
[[1, -3, 5]
 [2, 4, -6]])
```
2 by 3

```
    a_out[j] = g(z)
return a_out
```

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

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

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

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

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.

Correct
Correct. The w parameters of neuron 1 are in column 1. The w parameters of neuron 2 are in column 2, and so on.

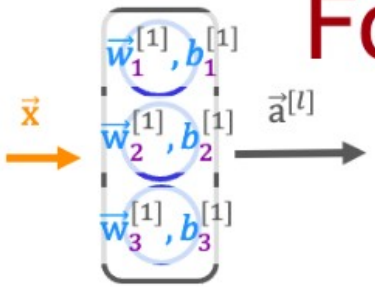
3.

1 / 1 point

3.

1 / 1 point

Forward prop in NumPy



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

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

2 by 3

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

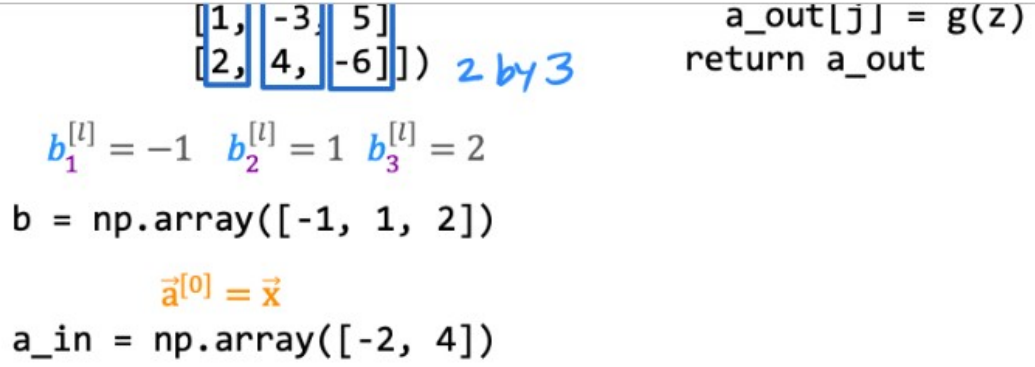
```
b = np.array([-1, 1, 2])
```

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

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

```
def dense(a_in,W,b, 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.



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.

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

Correct

Yes! For each neuron in the layer, there is one column in the numpy array W. The for loop calculates the activation value for each neuron. So if there are 5 columns in W, there are 5 neurons in the dense layer, and therefore the for loop goes through 5 iterations (one for each neuron).

DeepLearning.AI

Stanford
ONLINE

Vectorization (optional)

How neural networks are implemented efficiently

For loops vs. vectorization

```
x = np.array([200, 17])

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

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

[1,0,1]

vectorized

```
X = np.array([[200, 17]])

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

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

[[1,0,1]]

2D array

same

1x3 2D array

all 2D arrays

matrix multiplication

Dot products

example

$\begin{bmatrix}1\\2\end{bmatrix} \cdot \begin{bmatrix}3\\4\end{bmatrix}$

$z = (1 \times 3) + (2 \times 4)$

$3 + 8$

11

in general

$\begin{bmatrix}\uparrow\\ \vec{a} \\ \downarrow\end{bmatrix} \cdot \begin{bmatrix}\uparrow\\ \vec{w} \\ \downarrow\end{bmatrix}$

$z = \vec{a} \cdot \vec{w}$

transpose

$\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}$

$\vec{a}^T = \begin{bmatrix}1 & 2\end{bmatrix}$

vector vector multiplication

$\underset{1\times 2}{\begin{bmatrix}\leftarrow & \vec{a}^T & \rightarrow\end{bmatrix}} \underset{2\times 1}{\begin{bmatrix}\uparrow\\ \vec{w} \\ \downarrow\end{bmatrix}}$

$z = \vec{a}^T \vec{w}$

equivalent

useful for understanding matrix multiplication

Vector matrix multiplication

$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\vec{a}^T = [1 \quad 2]$ $\quad W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$ $\quad Z = \vec{a}^T W$

1 by 2

$[\leftarrow \quad \vec{a}^T \quad \rightarrow] \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$

$$Z = [\vec{a}^T \vec{w}_1 \quad a^T \vec{w}_2]$$

$(1 * 3) + (2 * 4)$

$3 + 8$

11

$(1 * 5) + (2 * 6)$

$5 + 12$

17

$$Z = [11 \quad 17]$$

matrix matrix multiplication

$$A = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$

rows

columns

$$Z = A^T W = \begin{bmatrix} \leftarrow & \vec{a}_1^T & \rightarrow \\ \leftarrow & \vec{a}_2^T & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

$$= \begin{bmatrix} \vec{a}_1^T \vec{w}_1 & \vec{a}_1^T \vec{w}_2 \\ \vec{a}_2^T \vec{w}_1 & \vec{a}_2^T \vec{w}_2 \end{bmatrix}$$

row 1 col 1 · row 1 col 2 · row 2 col 1 · row 2 col 2

$(-1 \times 3) + (-2 \times 4)$

$-3 + -8$

-11

$(-1 \times 5) + (-2 \times 6)$

$-5 + -12$

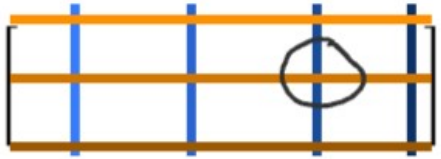
-17

$$= \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix}$$

general rules for matrix multiplication
↳ next video!

Question

$$X^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix}$$

$X^T W =$ 

Can you calculate the value at row 2, column 3?

- (-1 x 7) + (-2 x 8) = -23
- (0.1 x 5) + (0.2 x 6) = 1.7

Skip Continue

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

3 by 4 matrix

$\vec{a}_1^T \vec{w}_1 = (1 \times 3) + (2 \times 4) = 11$

row 3 column 2

$\vec{a}_3^T \vec{w}_2 = (0.1 \times 5) + (0.2 \times 6) = 1.7$

$0.5 + 1.2$

row 2 column 3?

$\vec{a}_2^T \vec{w}_3 = (-1 \times 7) + (-2 \times 8) = -23$

$-7 + -16$

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix}$$

$$Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

3 x 2 2 x 4

can only take dot products of vectors that are same length

[0.1 0.2] length 2

$\begin{bmatrix} 5 \\ 6 \end{bmatrix}$ length 2

3 by 4 matrix

↳ same # rows as A^T

same # columns as W

Matrix multiplication in NumPy

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

```
A=np.array([[1,-1,0.1],
            [2,-2,0.2]])

AT=np.array([[1,2],
             [-1,-2],
             [0.1,0.2]])

AT=A.T
```
↳ transpose

```
W=np.array([[3,5,7,9],
            [4,6,8,0]])
```

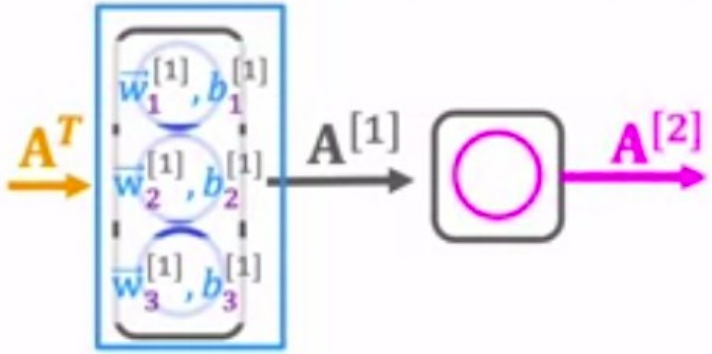
or

```
Z = np.matmul(AT,W)
Z = AT @ W
```

result

```
[[11,17,23,9],
 [-11,-17,-23,-9],
 [1.1,1.7,2.3,0.9]
]
```

Dense layer vectorized



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

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

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

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

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

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

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

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

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]]
```