

1. Neurons and the brain

# Neural networks

Origins: Algorithms that try to mimic the brain.



Used in the 1980's and early 1990's.  
Fell out of favor in the late 1990's.

Resurgence from around 2005.

speech → images → text (NLP) → ...

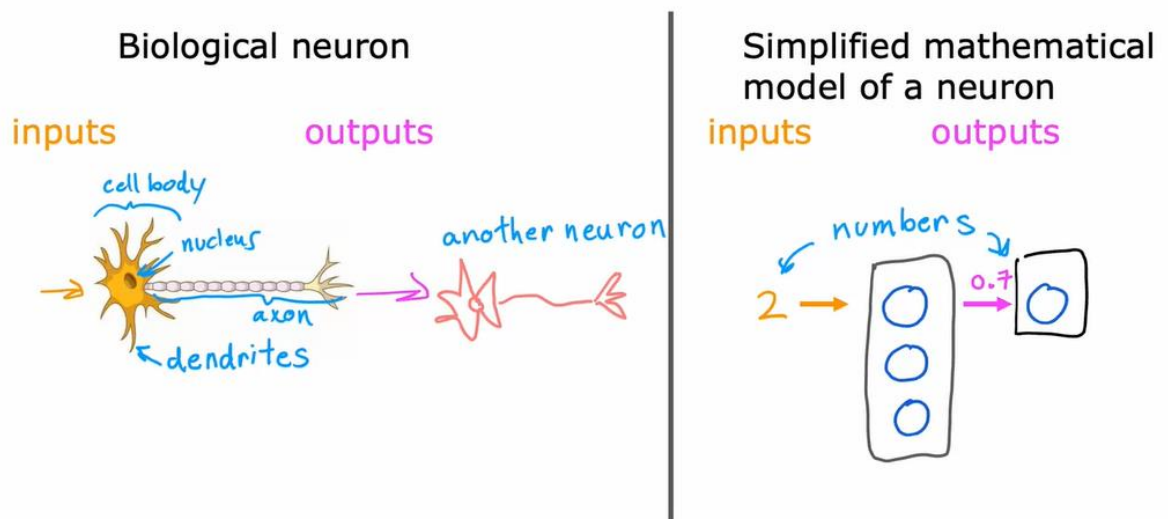
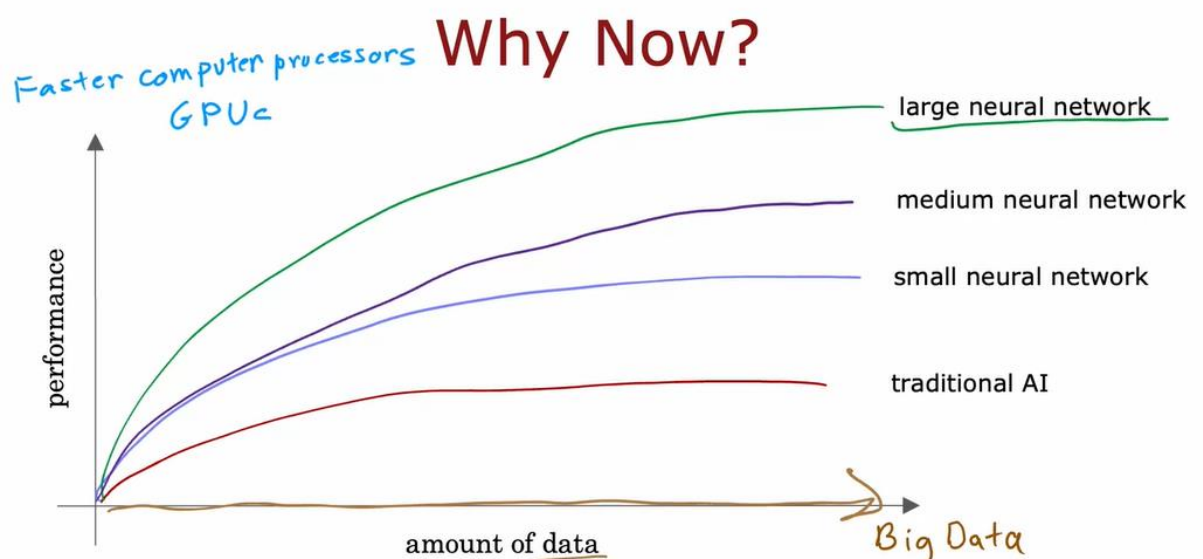
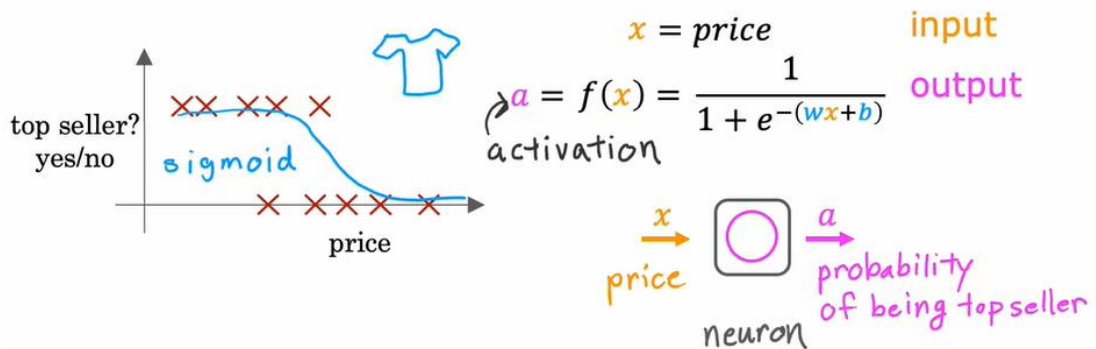


image source: <https://biologydictionary.net/sensory-neuron/>

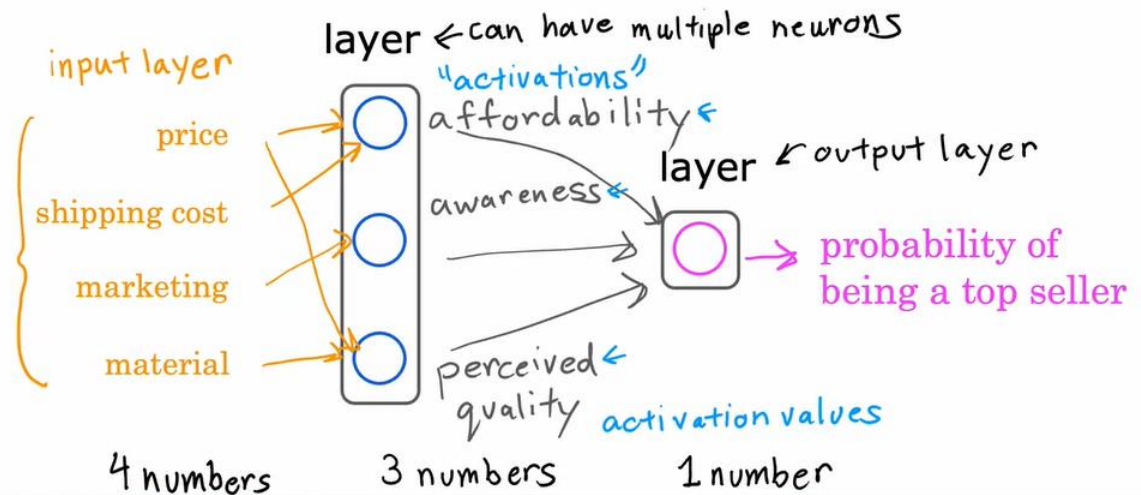


## 2. Demand prediction

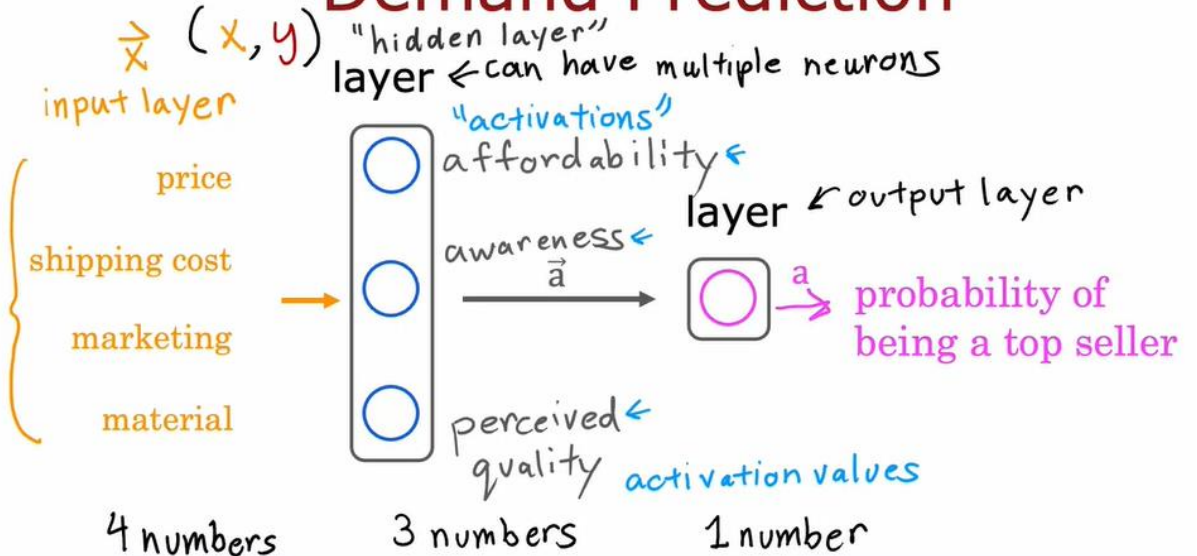
### Demand Prediction



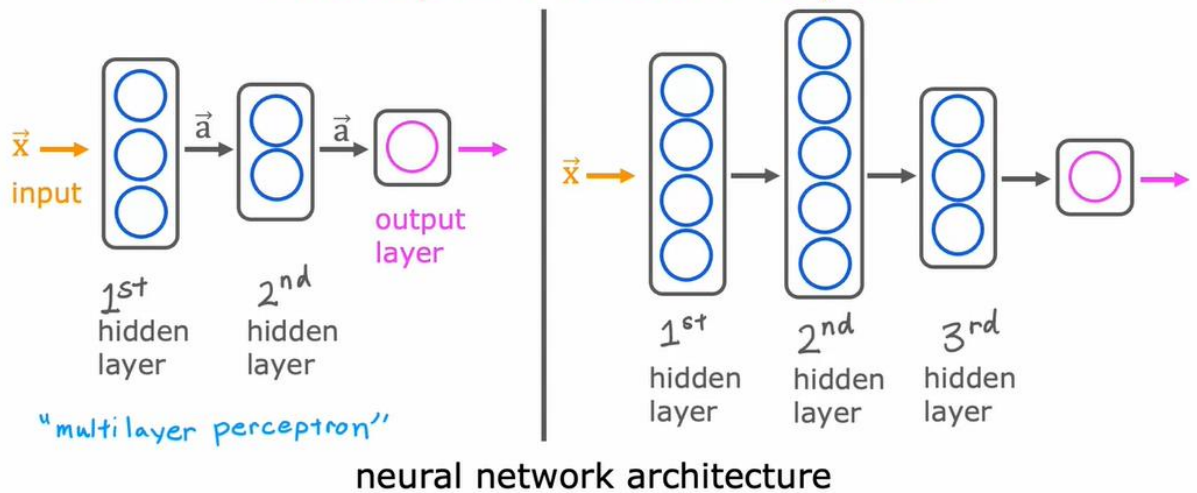
### Demand Prediction



### Demand Prediction

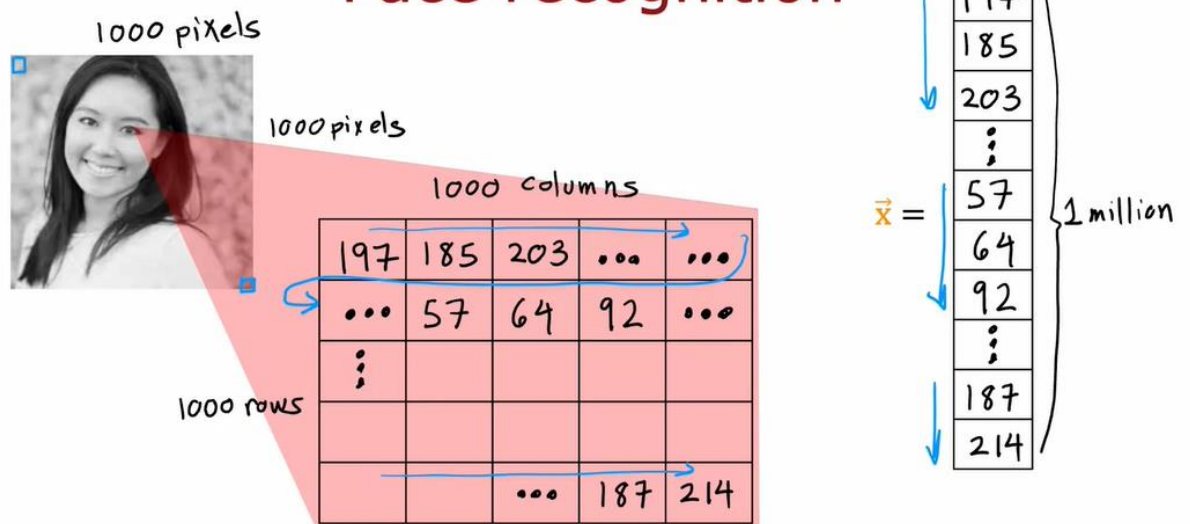


## Multiple hidden layers

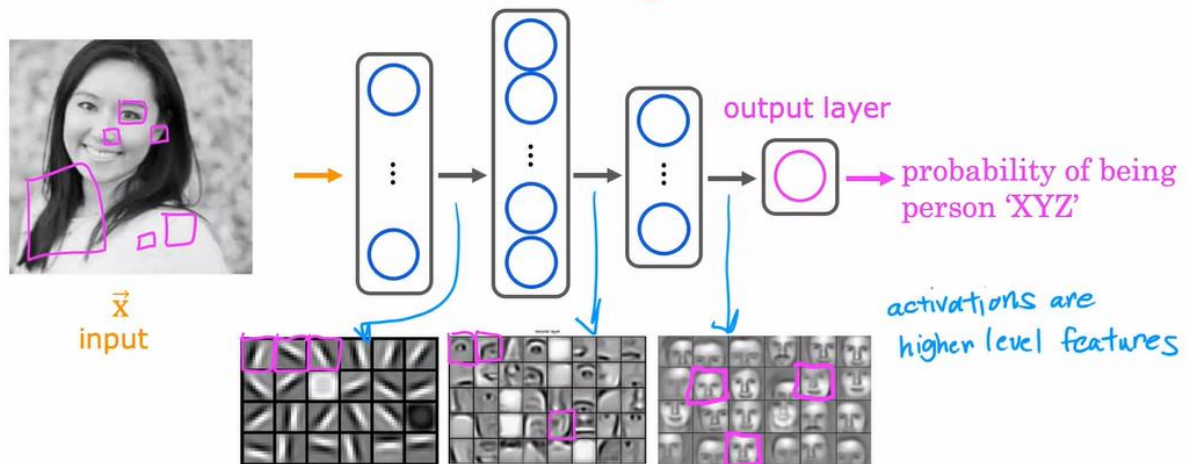


3. Example: recognizing images

## Face recognition

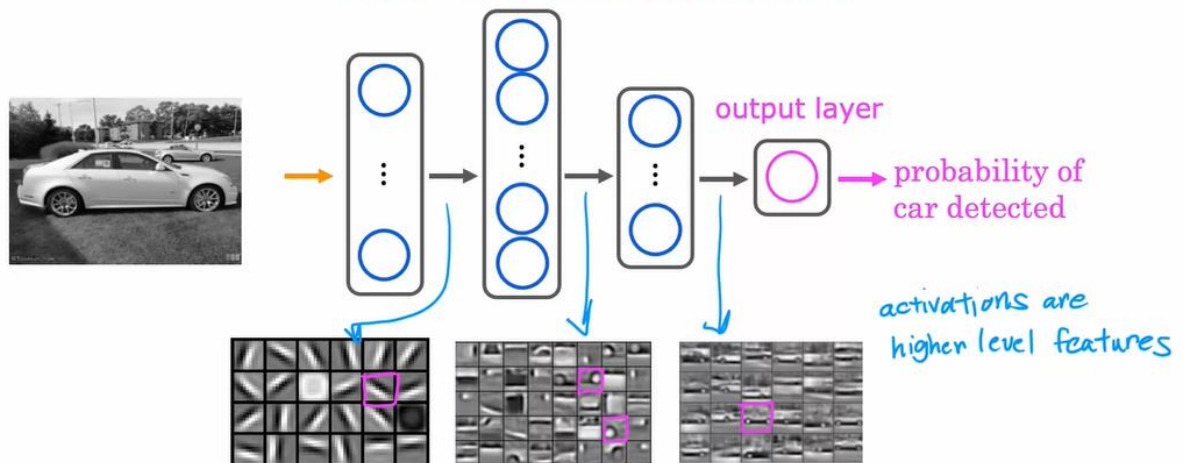


## Face recognition



source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

# Car classification



source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations  
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

## 4. Practice quiz

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

☒ activation function

✓ **Correct**

Yes, an activation is the number calculated by a neuron (and "activations" in the figure above is a vector that is output by a layer that contains multiple neurons).

☒ neurons

✓ **Correct**

Yes, a neuron is a part of a neural network

☒ layers

✓ **Correct**

Yes, a layer is a grouping of neurons in a neural network

☐ axon

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

☐ False

☒ True

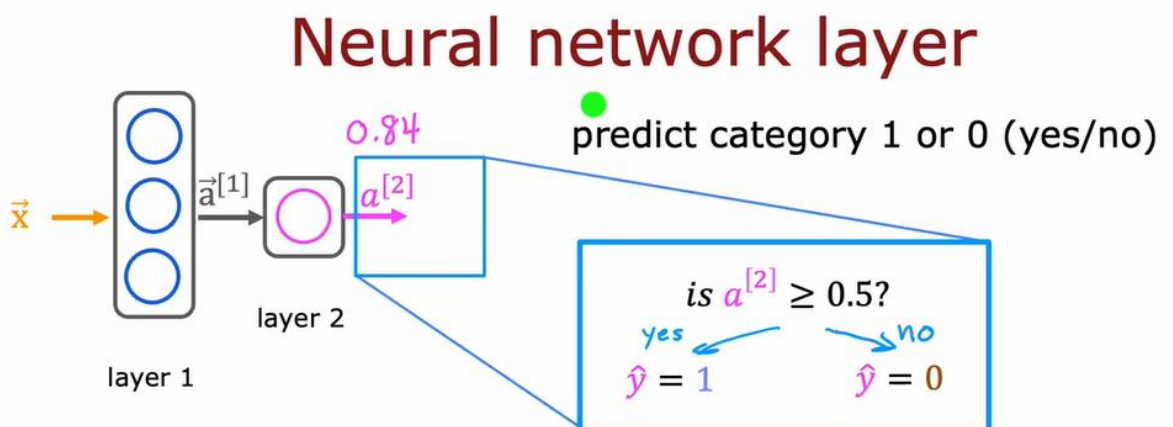
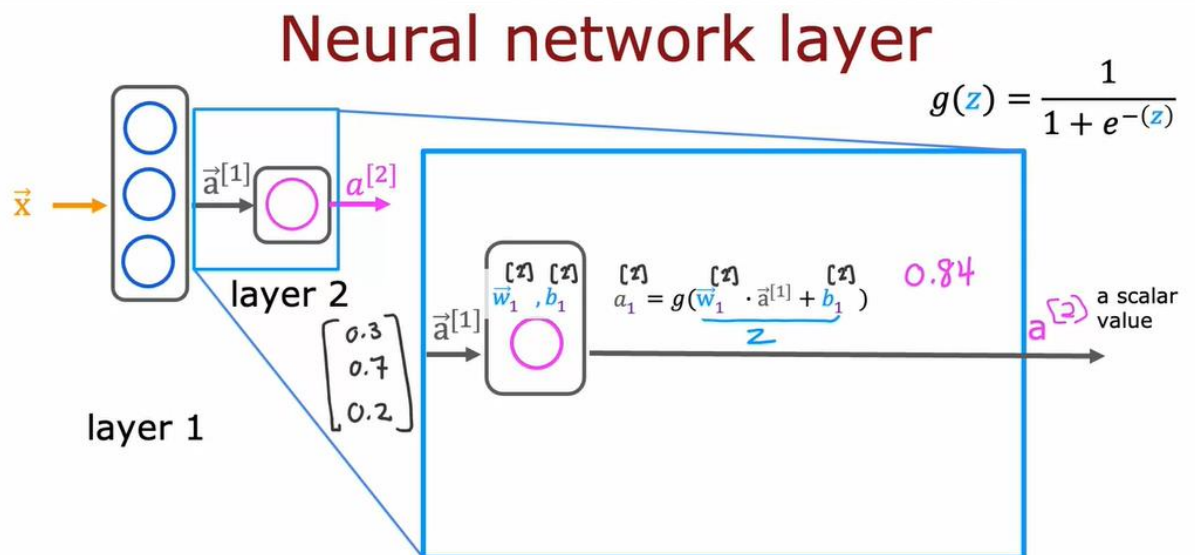
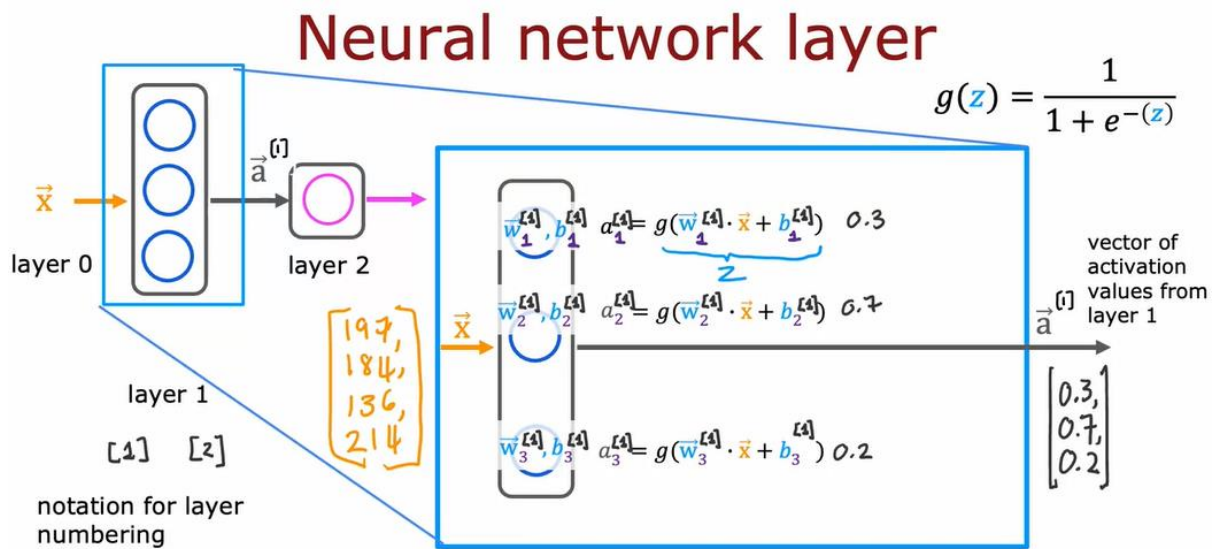
✓ **Correct**

Artificial neural networks use a very simplified mathematical model of what a biological neuron does.



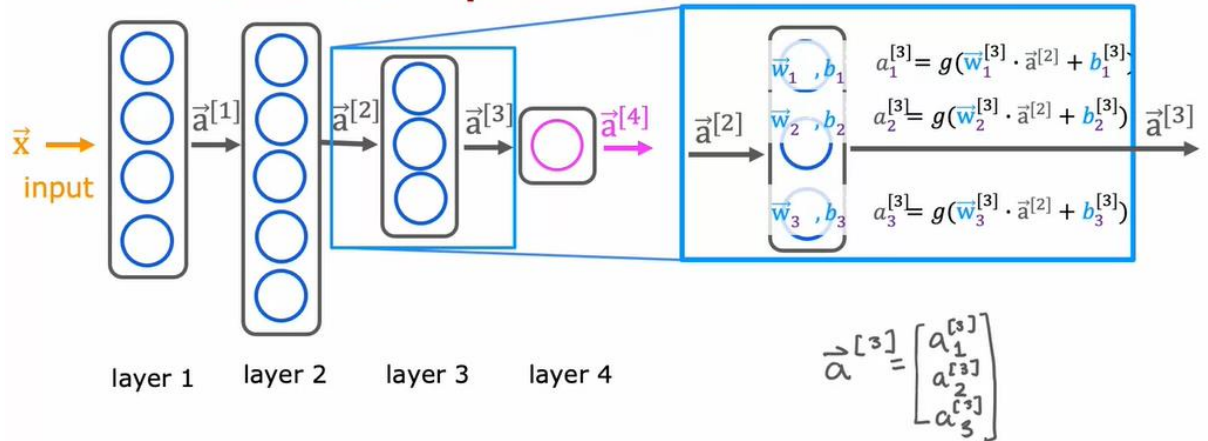
## Neural network model

### 1. Neural network layer

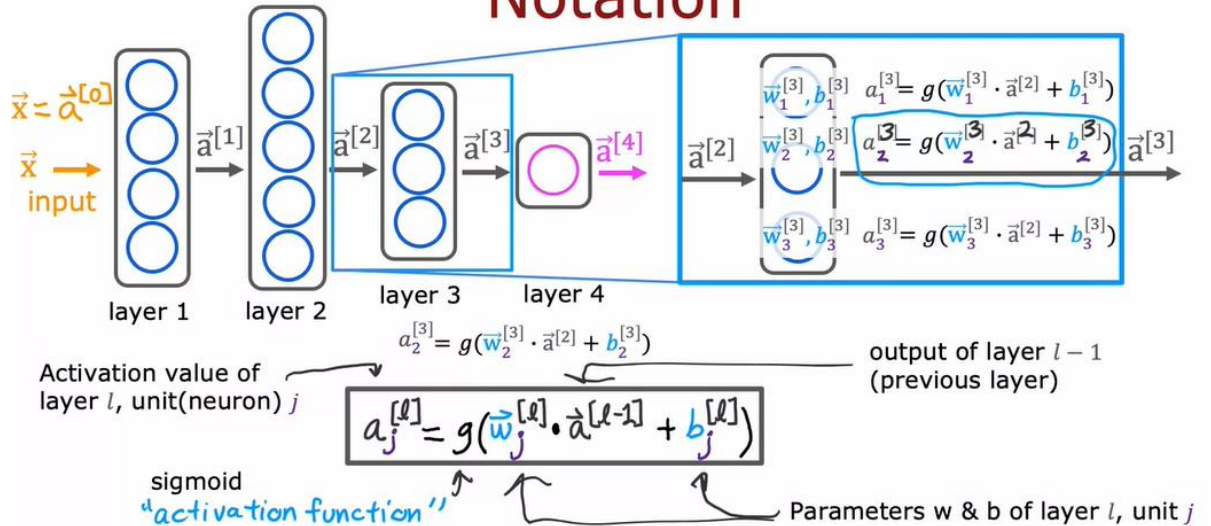


## 2. More complex neural networks

### More complex neural network

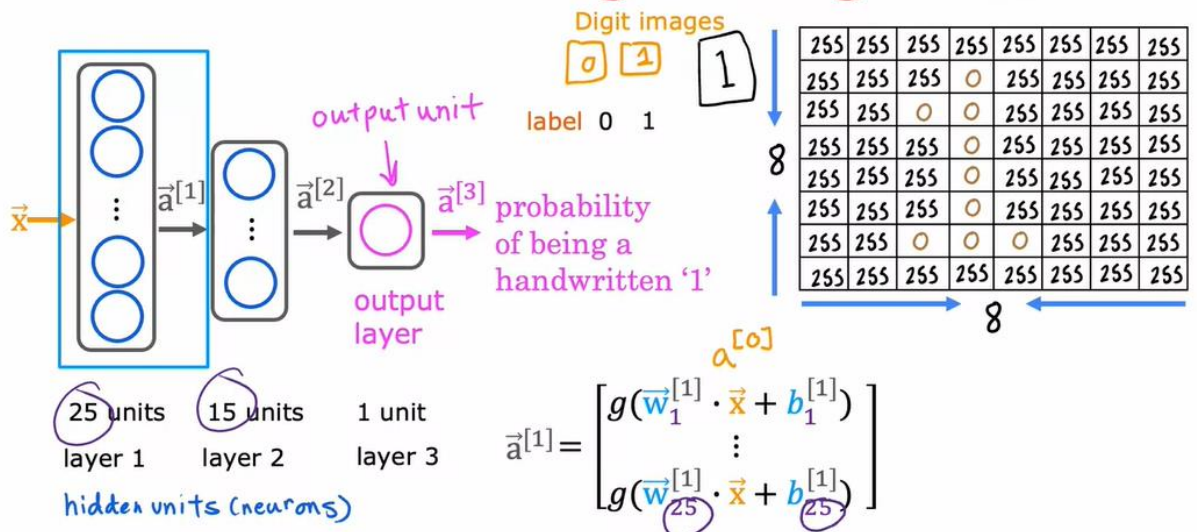


### Notation

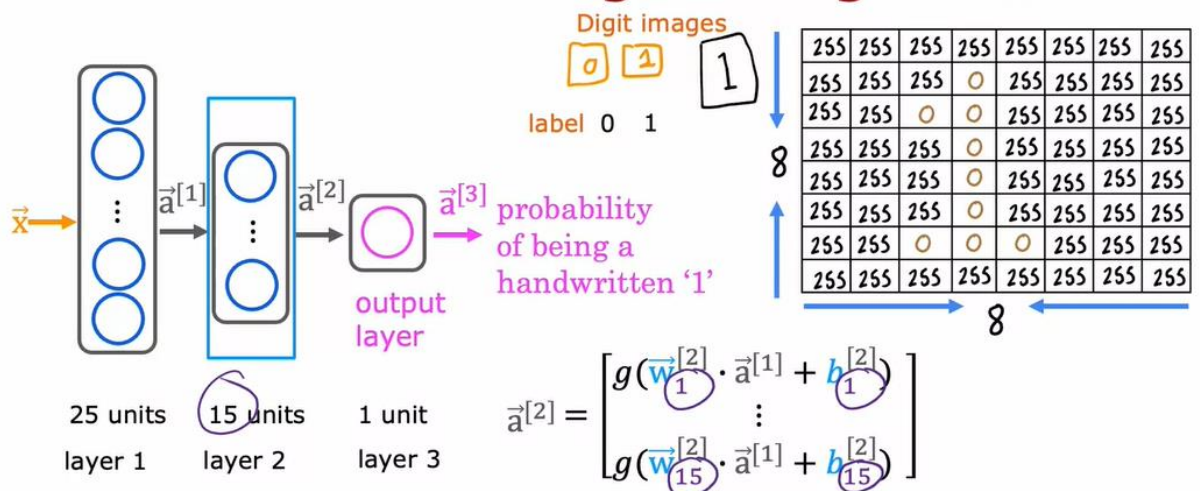


## 3. Inference: making predictions (forward propagation)

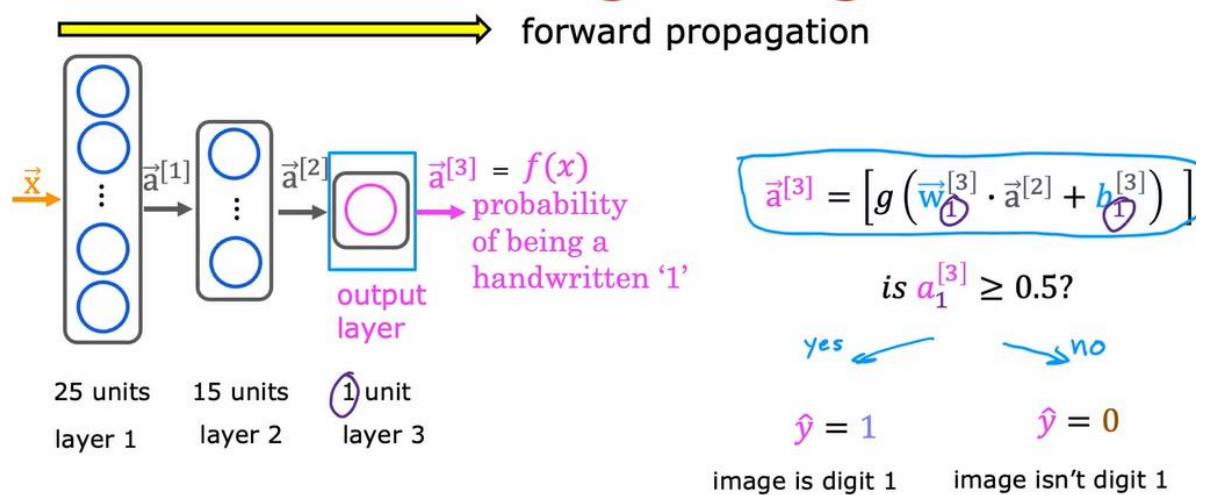
### Handwritten digit recognition



# Handwritten digit recognition



# Handwritten digit recognition



## 4. Practice quiz

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

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.

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

✓ 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.

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

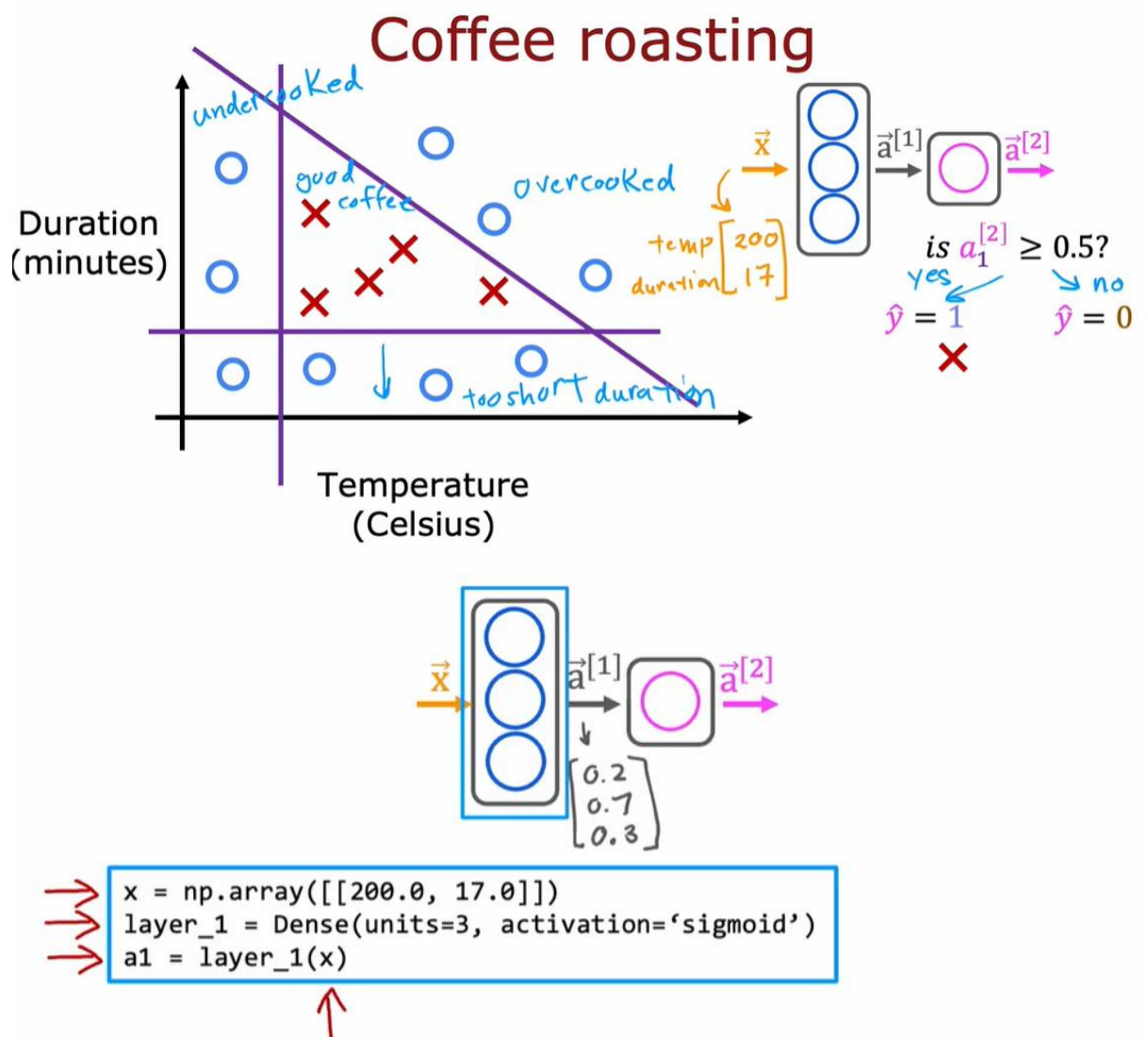
- ☐ A vector of several numbers that take values between 0 and 1
- ☐ A vector of several numbers, each of which is either exactly 0 or 1
- ☐ A number that is either exactly 0 or 1, comprising the network's prediction
- ☒ The estimated probability that the input image is of a number 1, a number that ranges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

✓ Correct

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

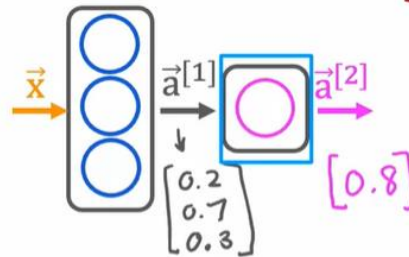
## TensorFlow implementation

### 1. Inference in code





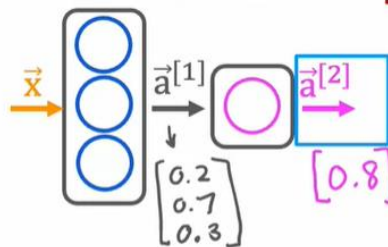
# Build the model using TensorFlow



```
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)
```

# 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)
```

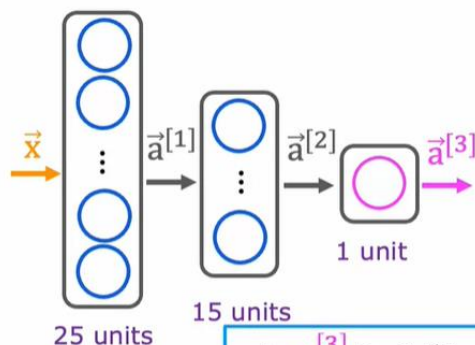
is  $a_1^{[2]} \geq 0.5$ ?

yes  $\hat{y} = 1$   $\times$

no  $\hat{y} = 0$   $\circ$

```
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)
```

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

yes  $\hat{y} = 1$   $\times$

no  $\hat{y} = 0$   $\circ$

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

## 2. Data in TensorFlow

### 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?

### Note about numpy arrays

2 rows  
 3 columns  
 $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$   
 2 x 3 matrix

4 rows  
 2 columns  
 $\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -5 & -6 \\ 7 & 8 \end{bmatrix}$   
 4 x 2 matrix

`x = np.array([[1, 2, 3],  
 [4, 5, 6]])`  
 $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

2D array

2 x 3

4 x 2

1 x 2

2 x 1

`x = np.array([[0.1, 0.2],  
 [-3.0, -4.0],  
 [-0.5, -0.6],  
 [7.0, 8.0]])`  
 $\begin{bmatrix} 0.1 & 0.2 \\ -3.0 & -4.0 \\ -0.5 & -0.6 \\ 7.0 & 8.0 \end{bmatrix}$

### 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])`

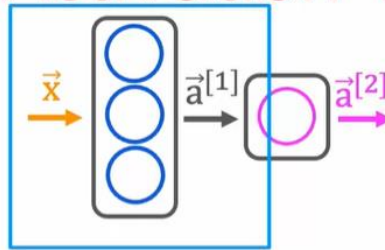
1D  
 "vector"

### 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]]`  
 ↓ ↓  
 →  $\begin{bmatrix} 200.0 & 17.0 \end{bmatrix}$  1 x 2

## Activation vector

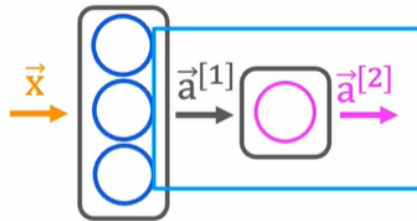


```

→ 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



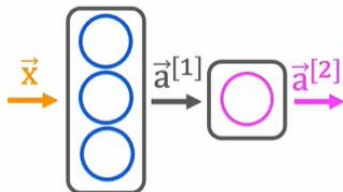
```

→ 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)

```

### 3. Building a neural network

## What you saw earlier



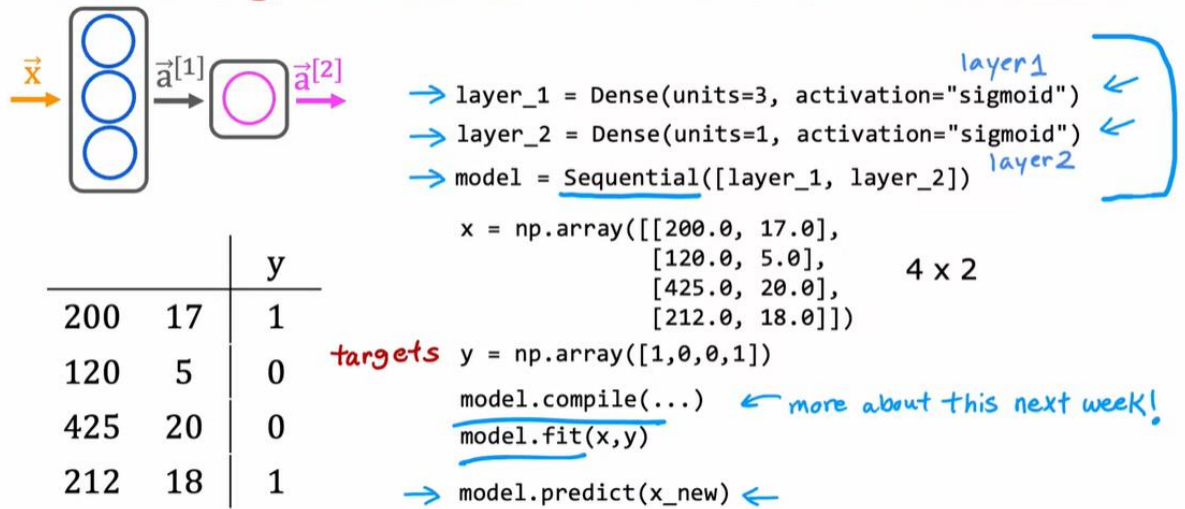
```

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

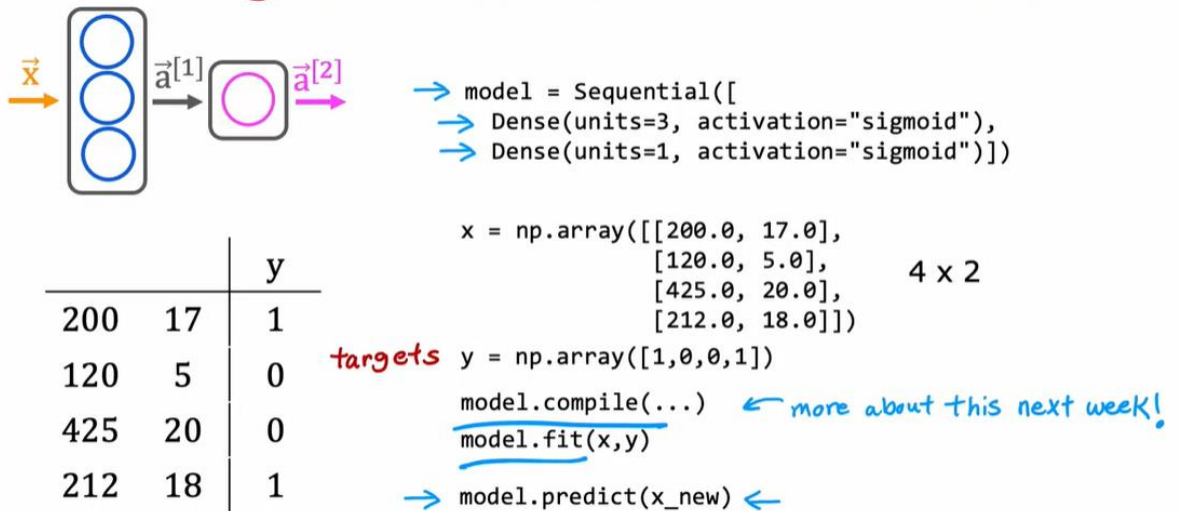
→ layer_2 = Dense(units=1, activation="sigmoid")
→ a2 = layer_2(a1)

```

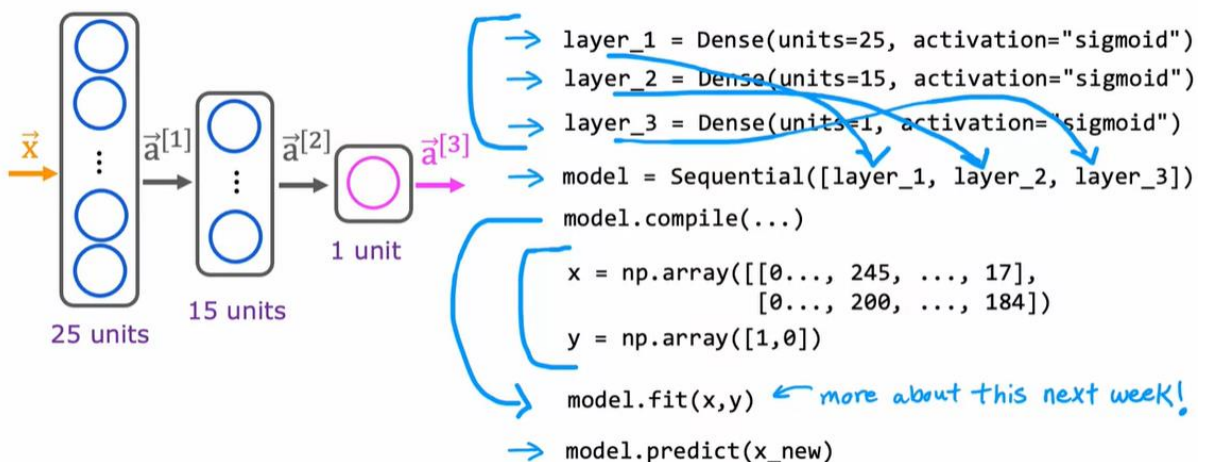
## Building a neural network architecture



## Building a neural network architecture



## Digit classification model





#### 4. Practice quiz

1. For the the following code:

```
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?

- ☐ 5
- ☐ 3
- ☒ 4
- ☐ 25



**Correct**

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

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(layer=2, units=4, activation = 'sigmoid')
- ☒ Dense(units=4, activation='sigmoid')
- ☐ Dense(units=4)



**Correct**

Yes! This will have 4 neurons and a sigmoid activation.

3.

## 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]]
```

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])`

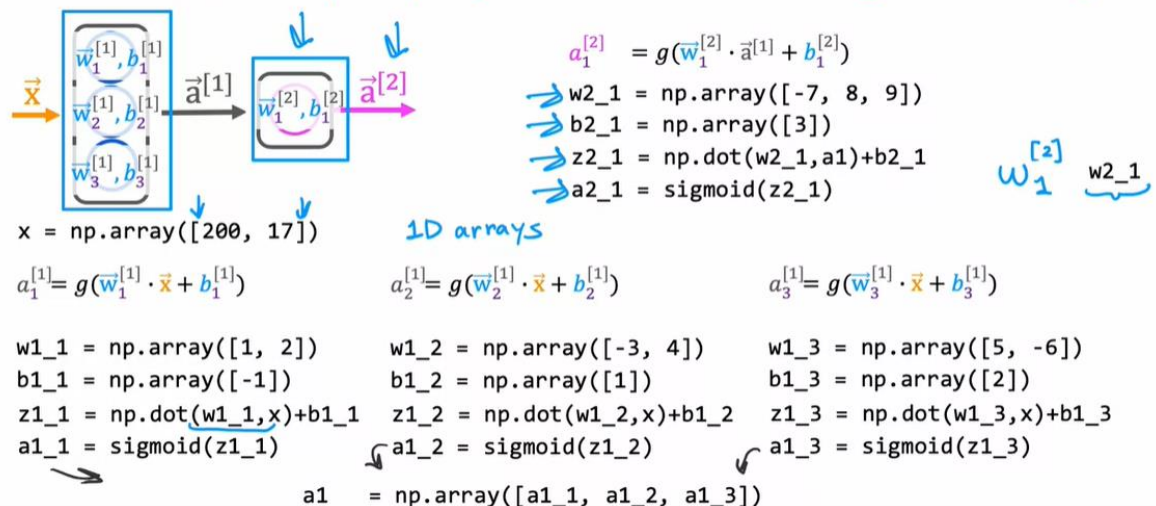
✓ Correct

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

## Neural network implementation in python

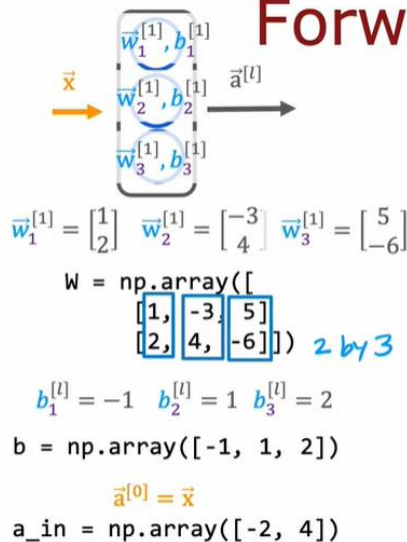
### 1. Forward prop in a single layer

## forward prop (coffee roasting model)



### 2. General implementation of forward prop

# Forward prop in NumPy



```
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

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
```

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

capital  $W$  refers to a matrix

## 3. Practice quiz

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



```
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. 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.

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



In the columns of  $W$ .



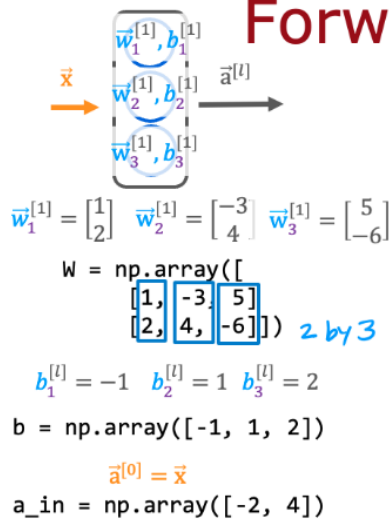
In the rows of  $W$ .



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.

## 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
```

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.

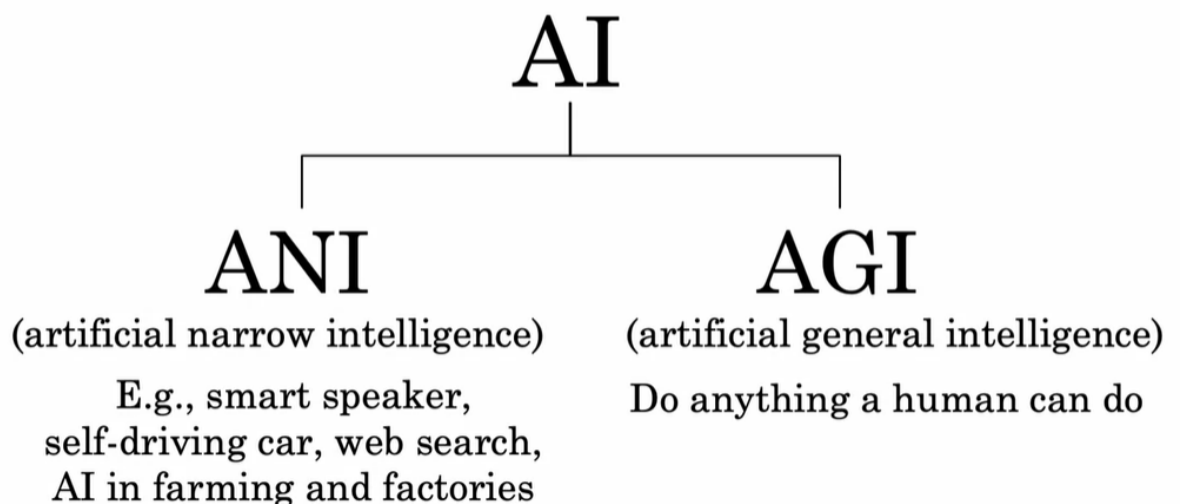
- ☐ 2 times
- ☒ 3 times
- ☐ 6 times
- ☐ 5 times

✓ 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 3 columns in  $W$ , there are 3 neurons in the dense layer, and therefore the for loop goes through 3 iterations (one for each neuron).

### Speculations on artificial general intelligence

1. Is there a path to AGI





# Neural network and the brain

Can we mimic the human brain?



We have (almost) no idea how the brain works

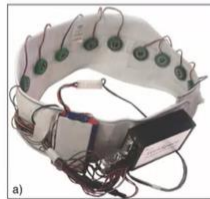
## Sensor representations in the brain



Seeing with your tongue



Human echolocation (sonar)



Haptic belt: Direction sense



Implanting a 3<sup>rd</sup> eye

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

### Vectorization

1. How neural network are implemented correctly

## 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]

```
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]]
```

vectorized

matrix multiplication

## 2. 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

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

$$\vec{a}^T = [1 \ 2]$$

vector vector multiplication

$$[\leftarrow \vec{a}^T \rightarrow] \begin{bmatrix} \uparrow \\ \vec{w} \\ \downarrow \end{bmatrix} \quad 1 \times 2 \quad 2 \times 1$$

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

equivalent

useful for understanding matrix multiplication

# Vector matrix multiplication

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

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

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

1 by 2

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

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

$$(1 \times 3) + (2 \times 4) \quad (1 \times 5) + (2 \times 6)$$

3 + 8      5 + 12

11      17

$$z = [11 \ 17]$$

# matrix matrix multiplication

$$A = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix}$$

rows

$$\vec{w} = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

columns

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

$$\begin{matrix} \text{row 1 col 1} & = & \begin{bmatrix} \vec{a}_1^T \vec{w}_1 & \vec{a}_1^T \vec{w}_2 \end{bmatrix} & \text{row 1 col 2} \\ \text{row 2 col 1} & = & \begin{bmatrix} \vec{a}_2^T \vec{w}_1 & \vec{a}_2^T \vec{w}_2 \end{bmatrix} & \text{row 2 col 2} \end{matrix}$$

$$\begin{matrix} (-1 \times 3) + (-2 \times 4) & & (-1 \times 5) + (-2 \times 6) \\ -3 + -8 & & -5 + -12 \\ -11 & & -17 \end{matrix}$$

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

general rules for matrix multiplication  
→ next video!

### 3. Matrix multiplication rules

## 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

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

3 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  $W$

### 4. Matrix multiplication code

## Matrix multiplication in NumPy

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

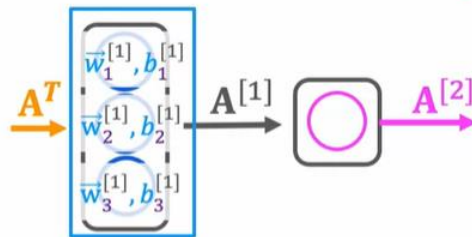
`A=np.array([[1,-1,0.1],[2,-2,0.2]])`      `W=np.array([[3,5,7,9],[4,6,8,0]])`      `Z = np.matmul(AT,W)`  
 or `Z = AT @ W`

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

`AT=A.T`      ↳ transpose

result  $\begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$

# Dense layer vectorized



$$A^T = \begin{bmatrix} 200 & 17 \end{bmatrix}$$

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

$$B = \begin{bmatrix} -1 & 1 & 2 \end{bmatrix}$$

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

$$\begin{bmatrix} 165 & -531 & 900 \end{bmatrix}$$

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

$$A = g(Z)$$

$$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

```

A
AT = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([[-1, 1, 2]])
def dense(AT,W,b):
    z = np.matmul(AT,W) + b
    a_out = g(z)
    return a_out
[[1,0,1]]

```