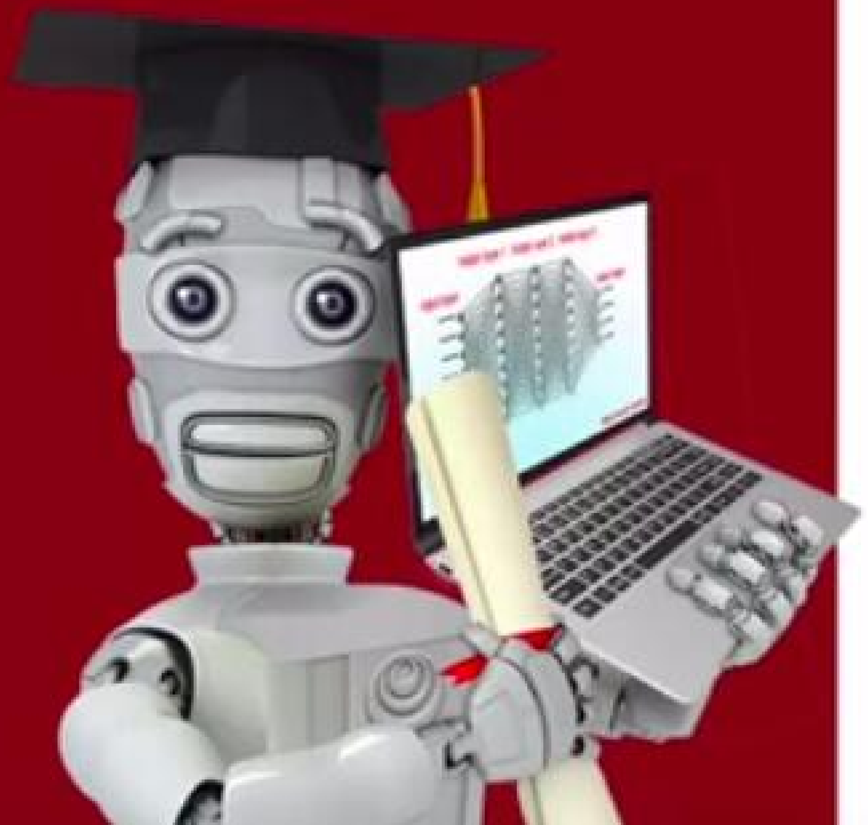
DeepLearning.AI

Stanford
ONLINE



Neural Networks Intuition

Neurons and the brain

Neural networks

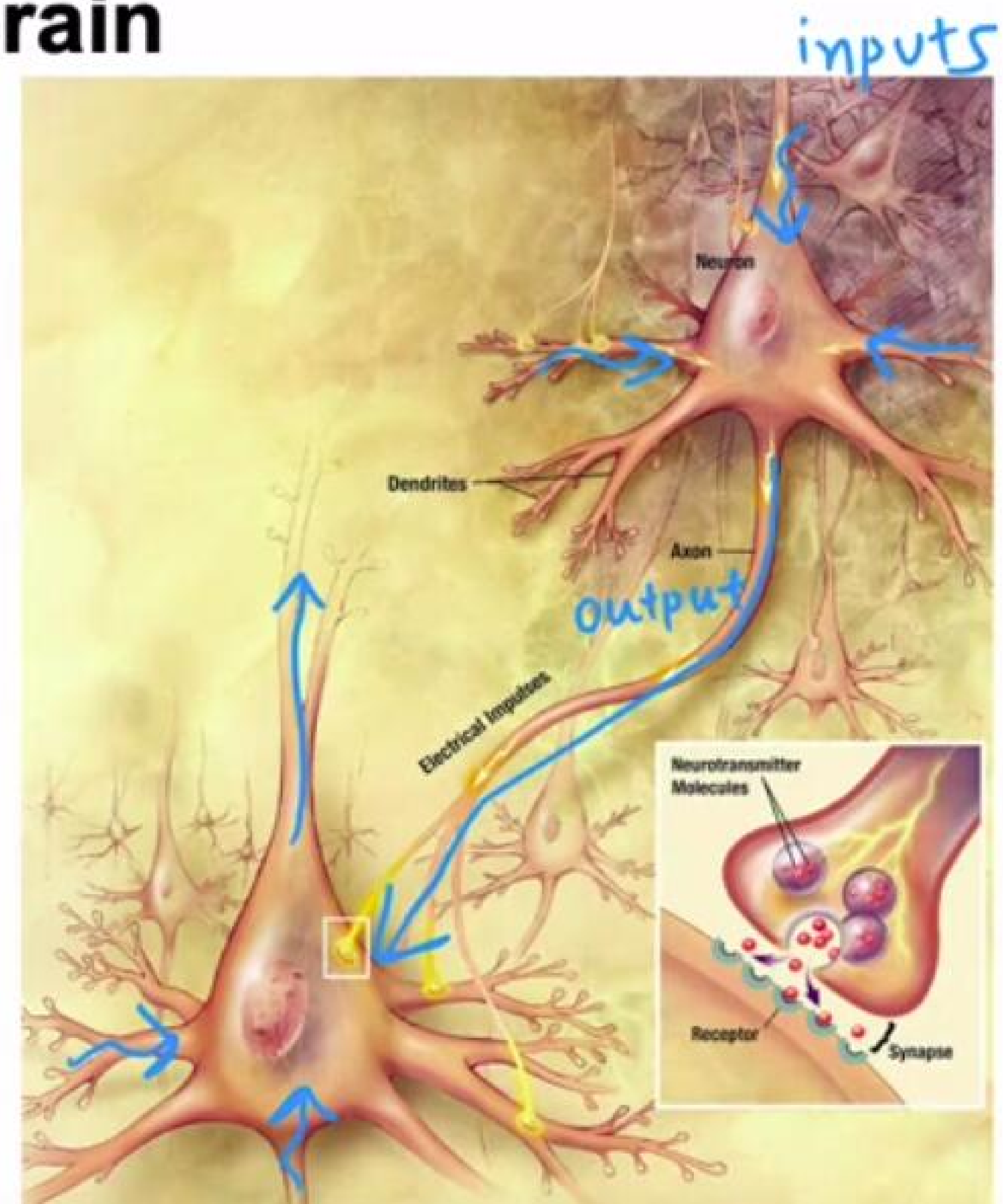
Origins: Algorithms that try to mimic the brain.

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

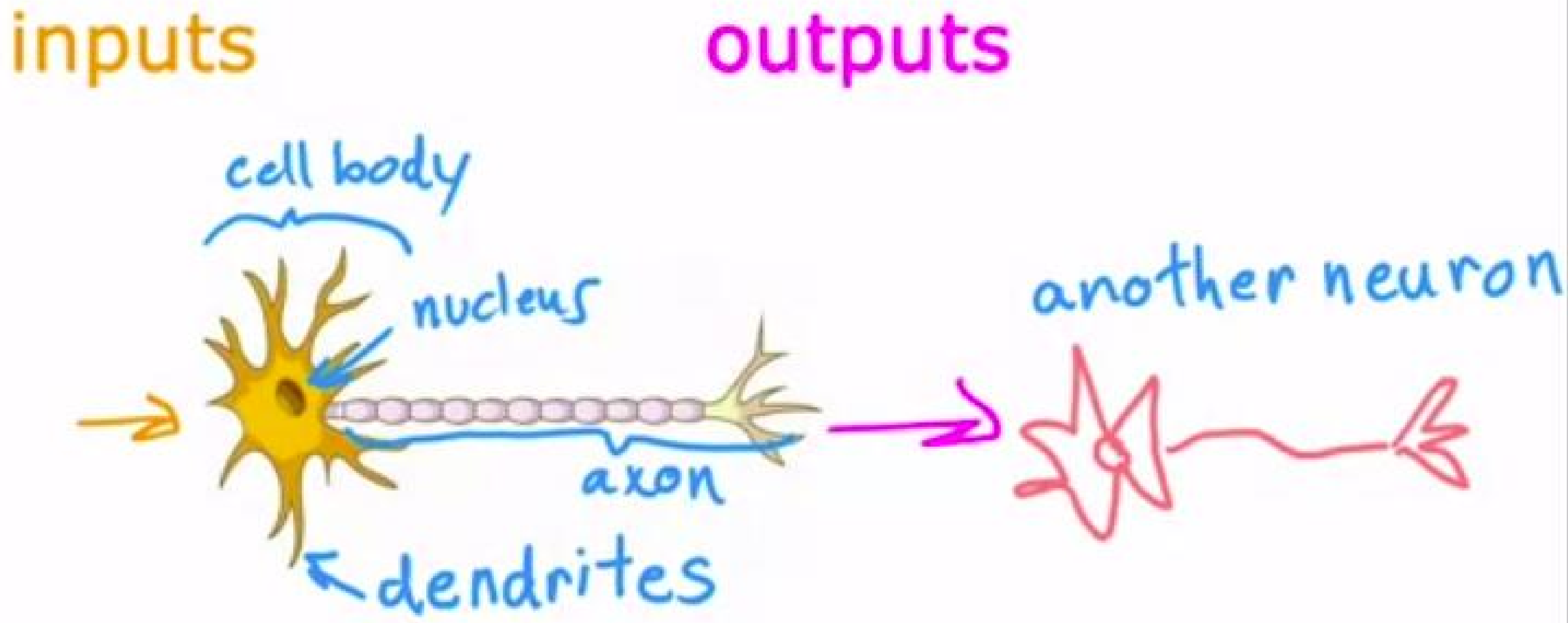
Resurgence from around 2005.

speech → images → text (NLP) → ...

Neurons in the brain



Biological neuron



Simplified mathematical model of a neuron

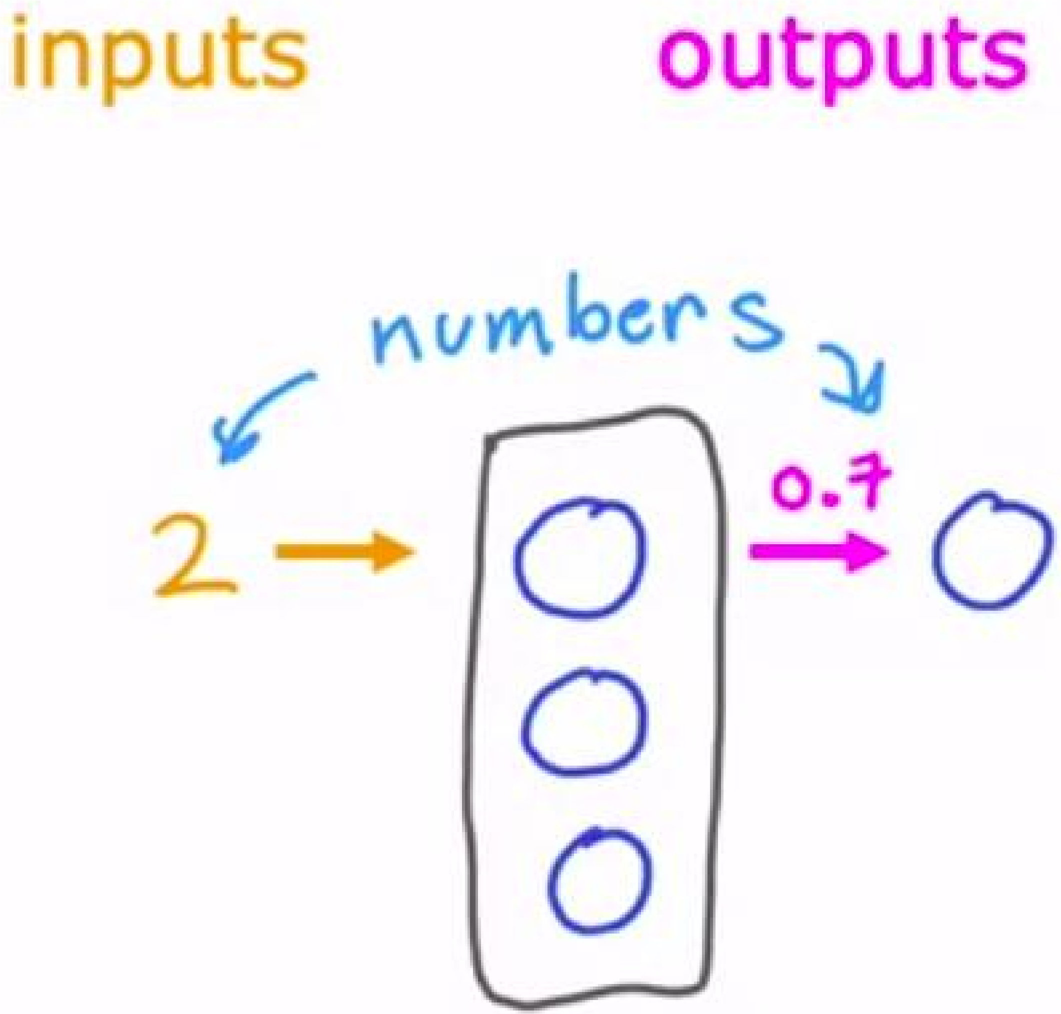
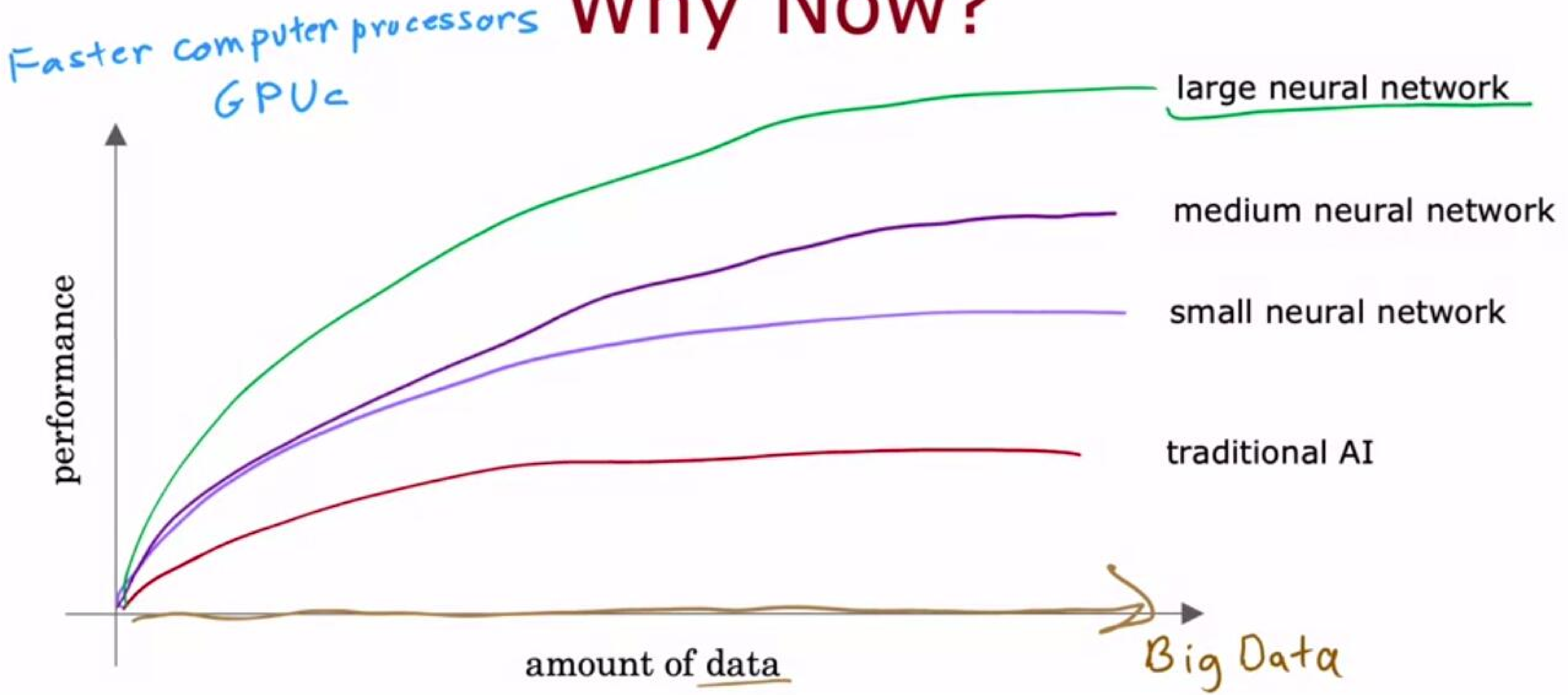


image source: https://biologydictionary.net/sensory-neuron/

Why Now?



Faster computer processors
GPUs
large neural network
medium neural network
small neural network
traditional AI
performance
amount of data
Big Data

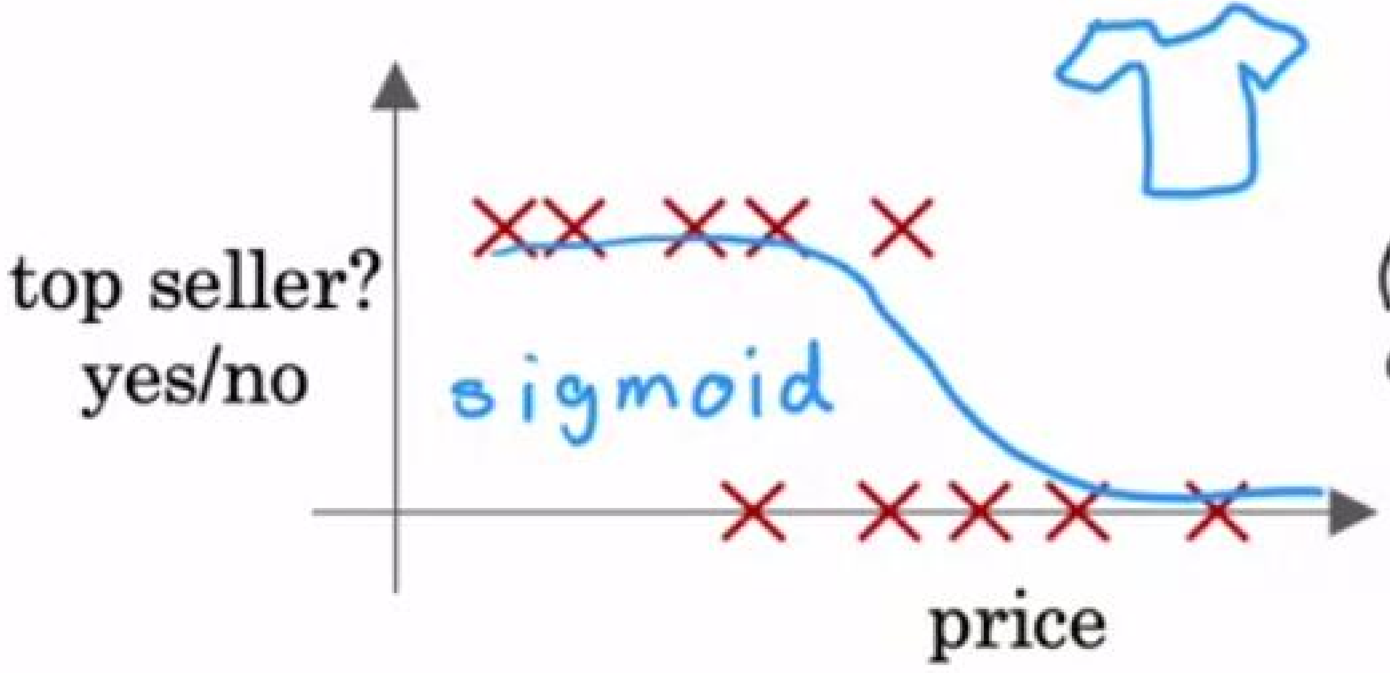
DeepLearning.AI

Stanford
ONLINE

Neural Network Intuition

Demand Prediction

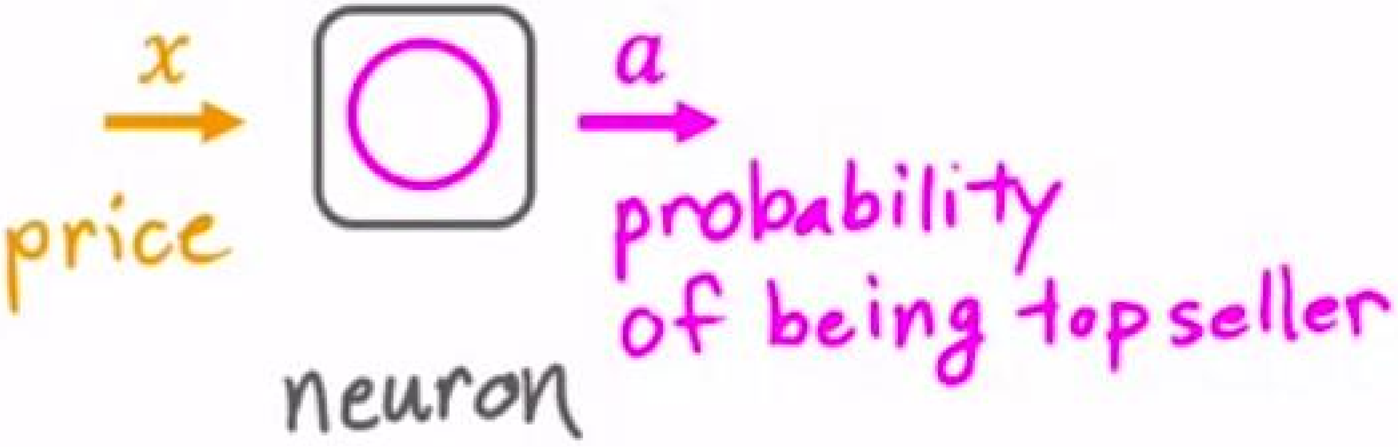
Demand Prediction



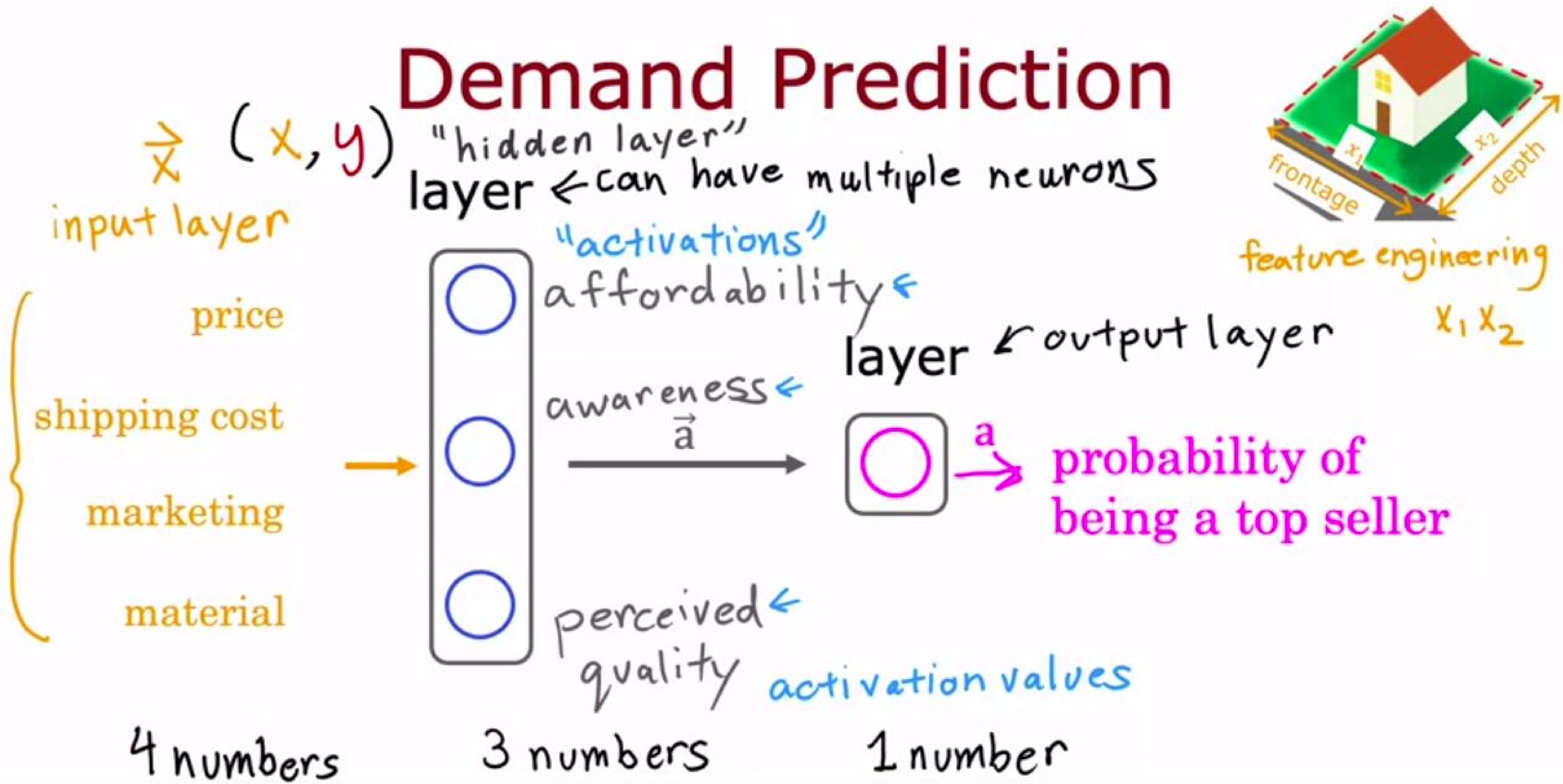
$x = price$ input

$a = f(x) = \dfrac{1}{1 + e^{-(wx+b)}}$ output

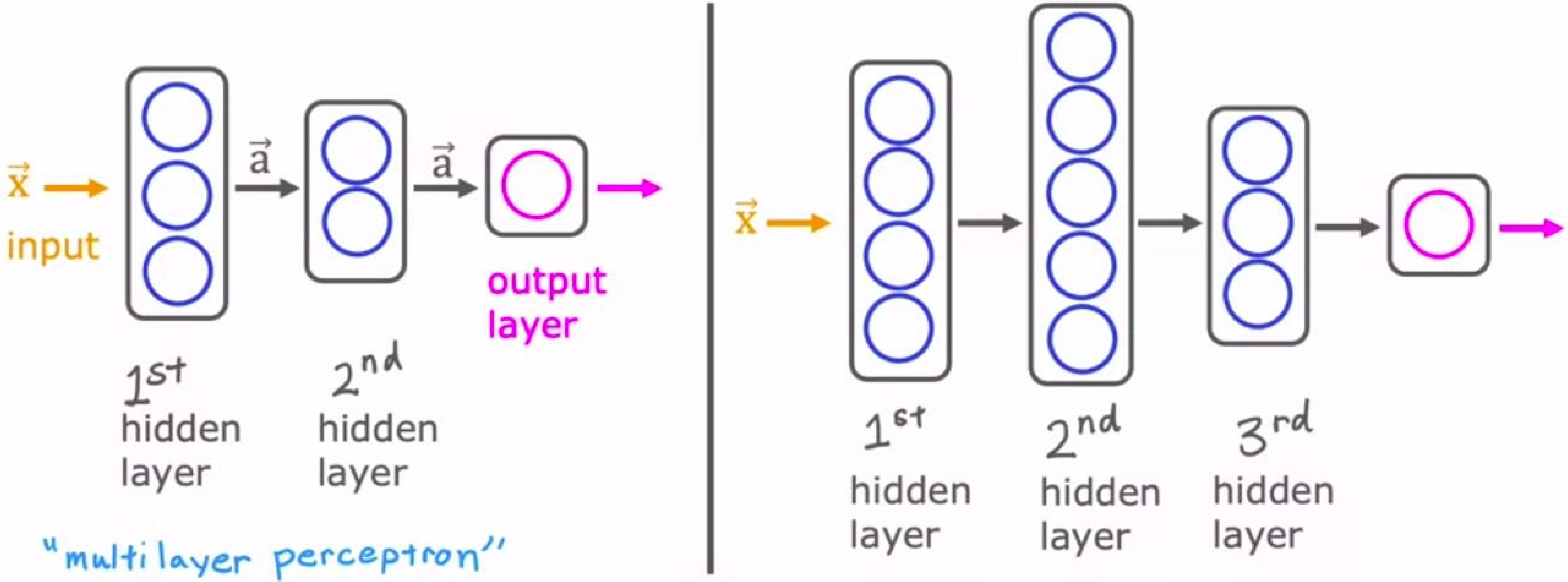
activation



Demand Prediction



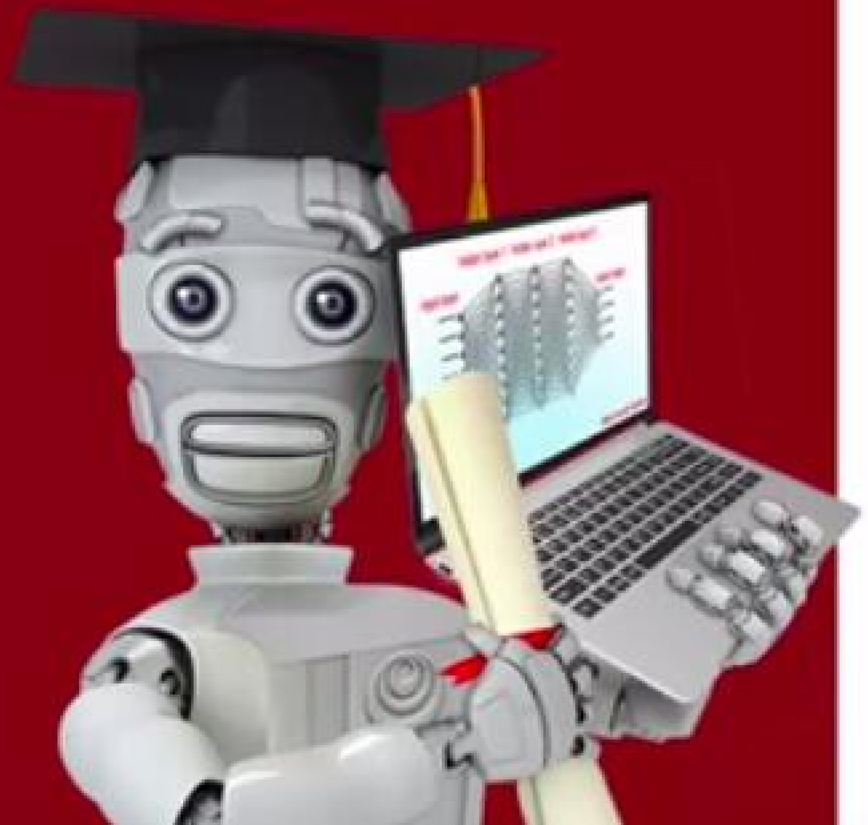
Multiple hidden layers



neural network architecture

DeepLearning.AI

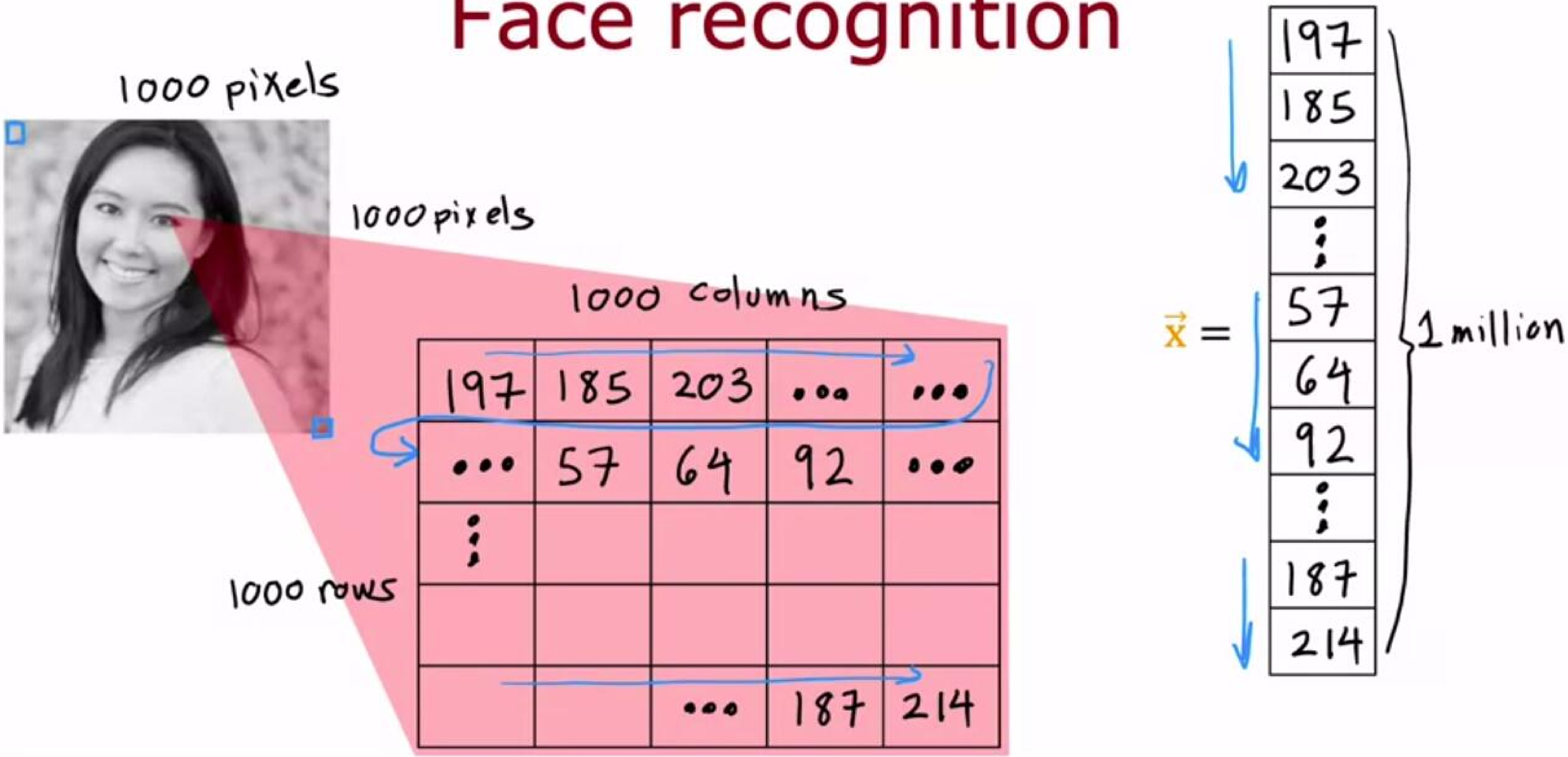
Stanford
ONLINE



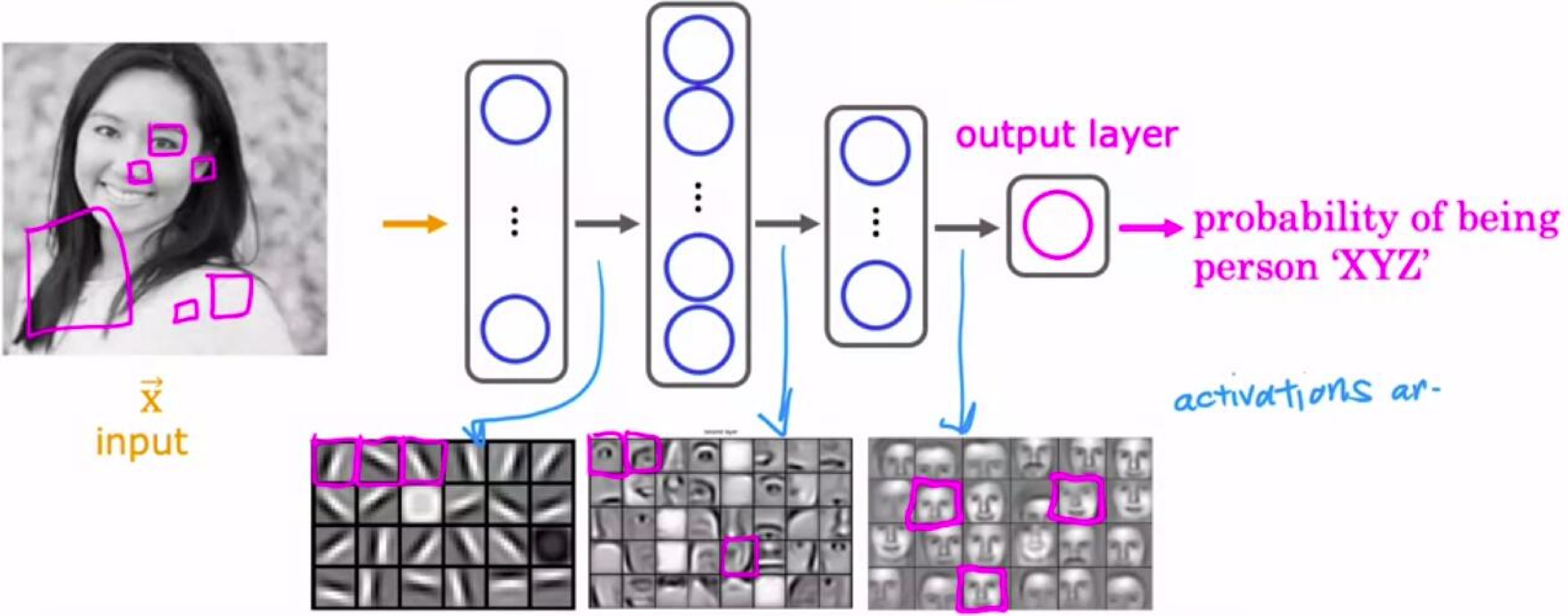
Neural Networks Intuition

Example: Recognizing Images

Face recognition

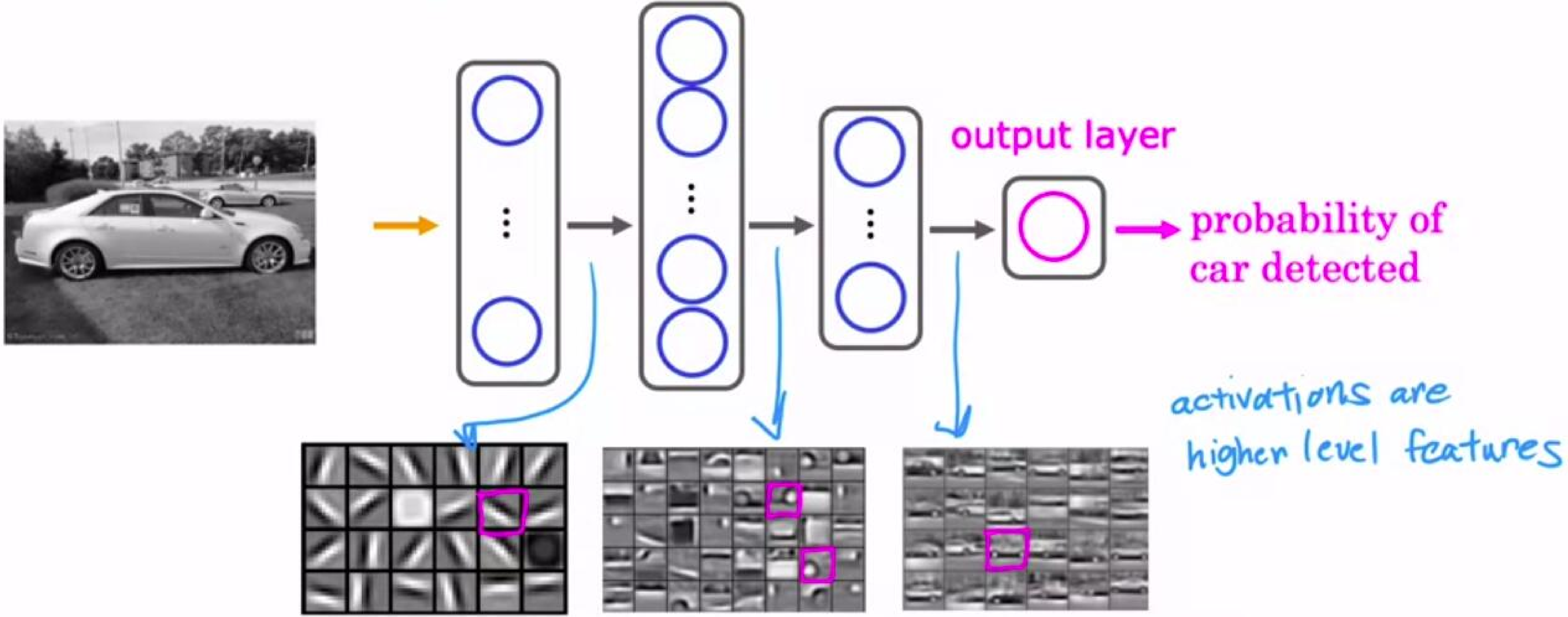


Face recognition



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

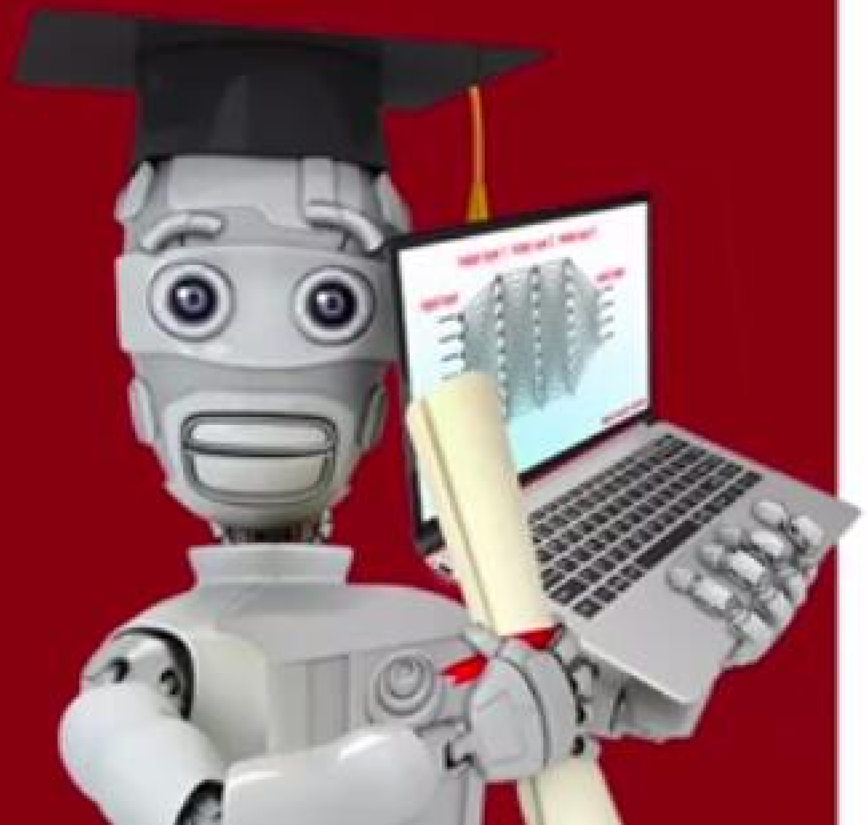
Car classification



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

DeepLearning.AI

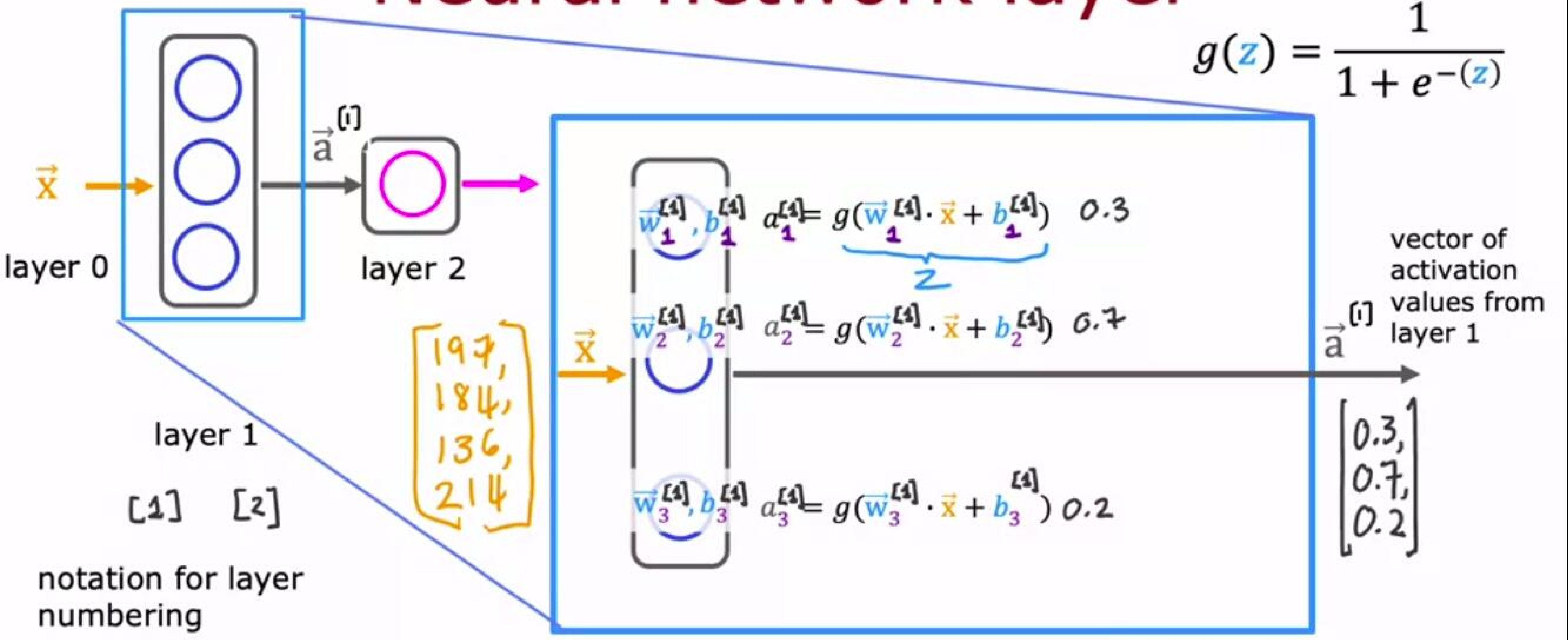
Stanford
ONLINE



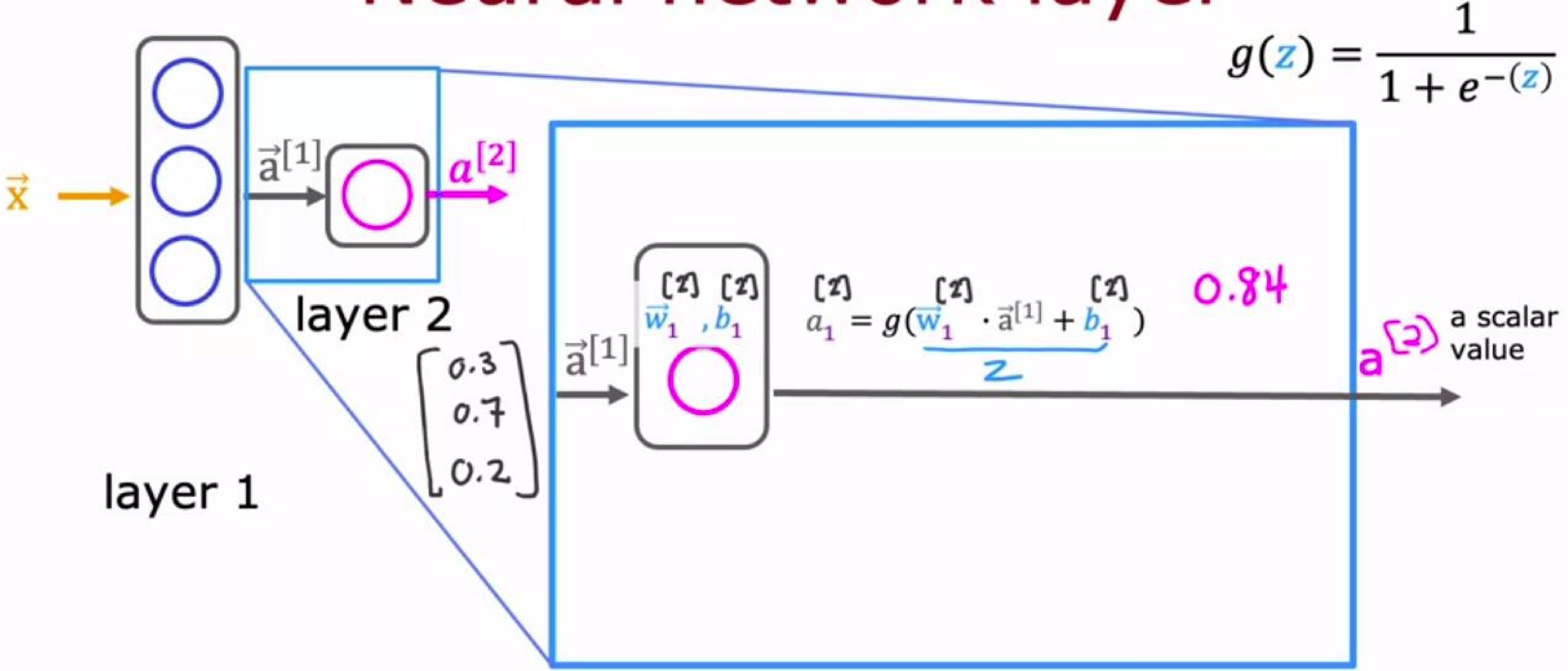
Neural network model

Neural network layer

Neural network layer

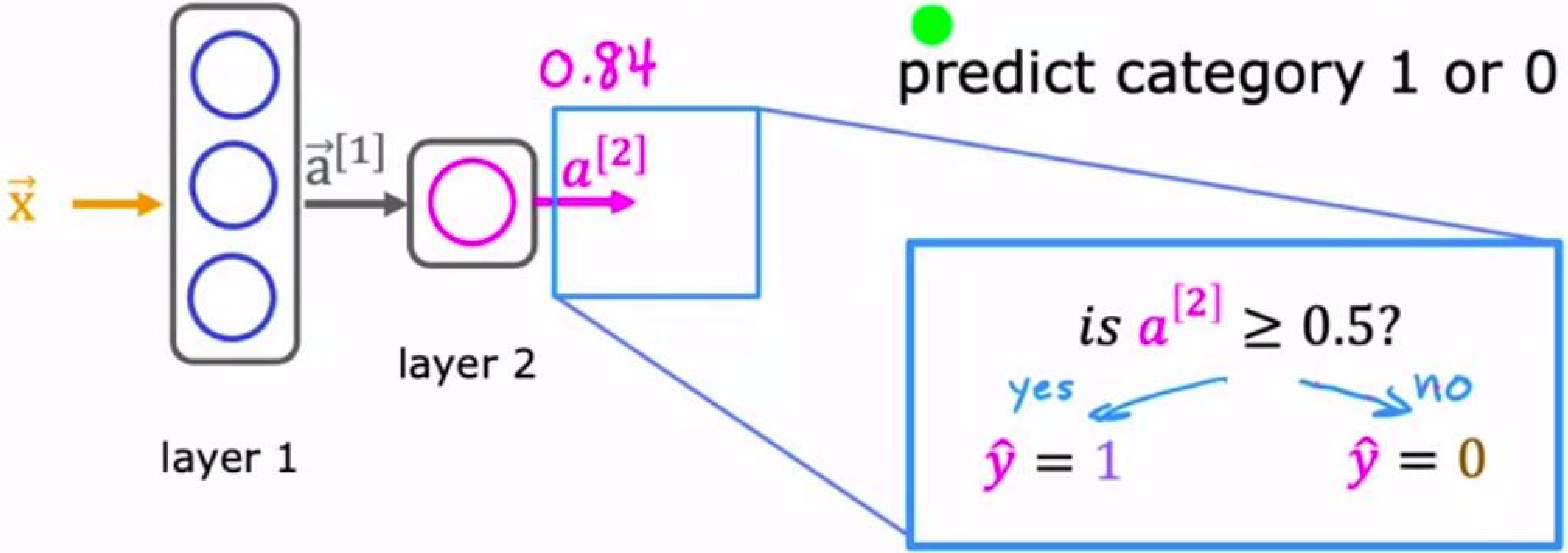


Neural network layer



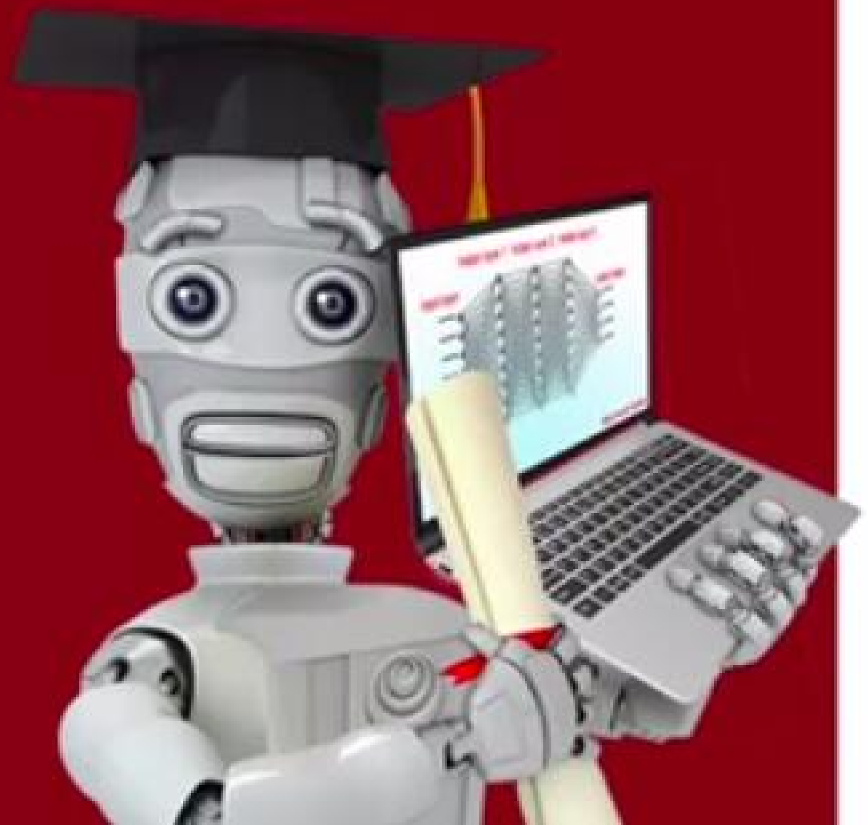
Neural network layer

predict category 1 or 0 (yes/no)



DeepLearning.AI

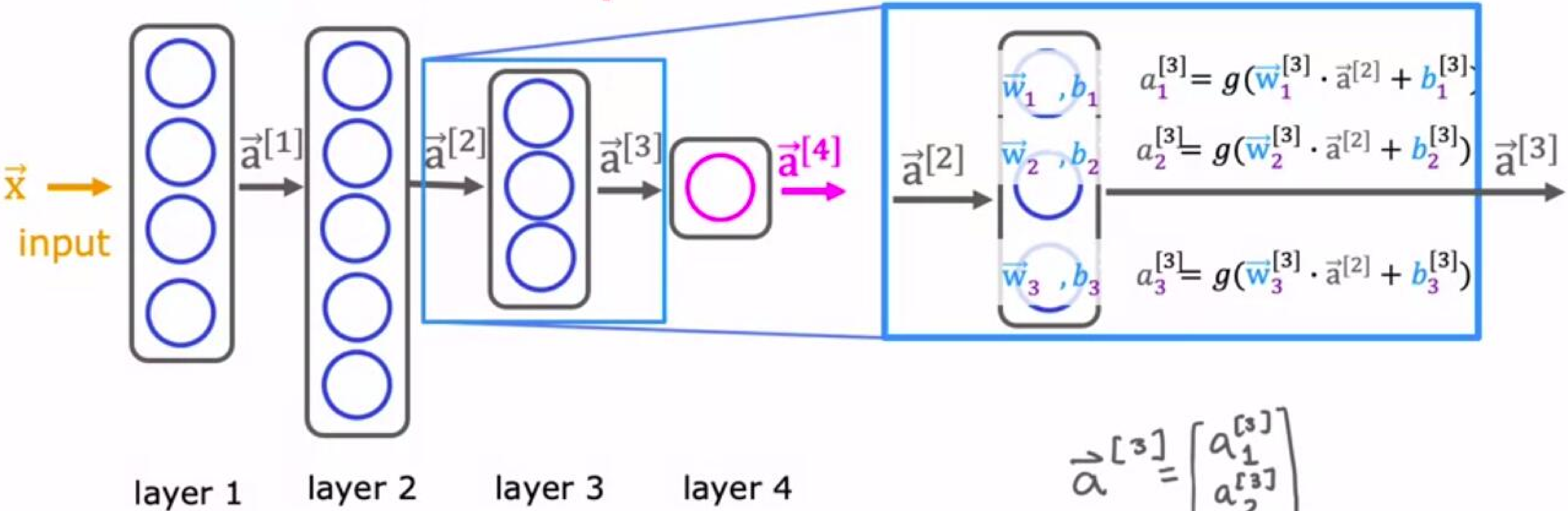
Stanford
ONLINE



Neural Network Model

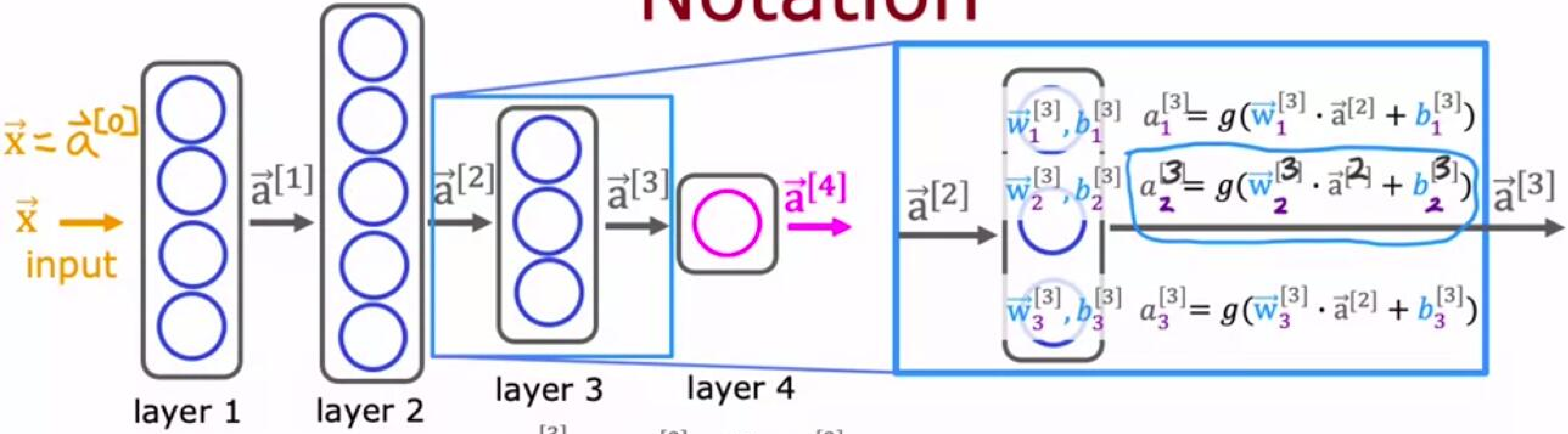
More complex neural networks

More complex neural network



$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$$

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)

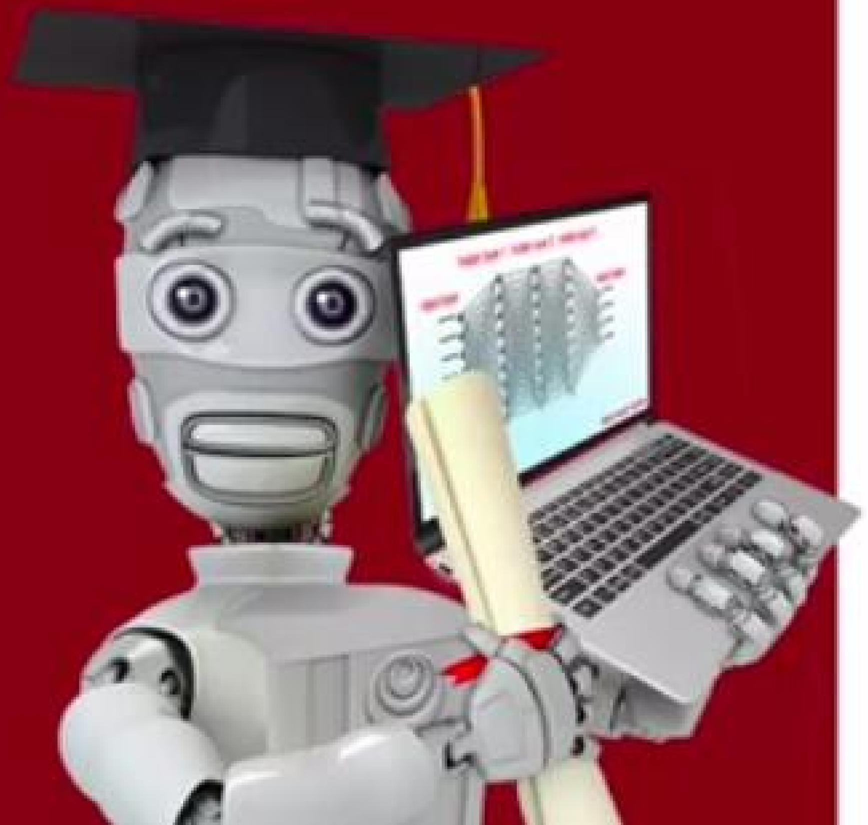
$$\boxed{a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})}$$

sigmoid
"activation function"

Parameters w & b of layer l, unit j

DeepLearning.AI

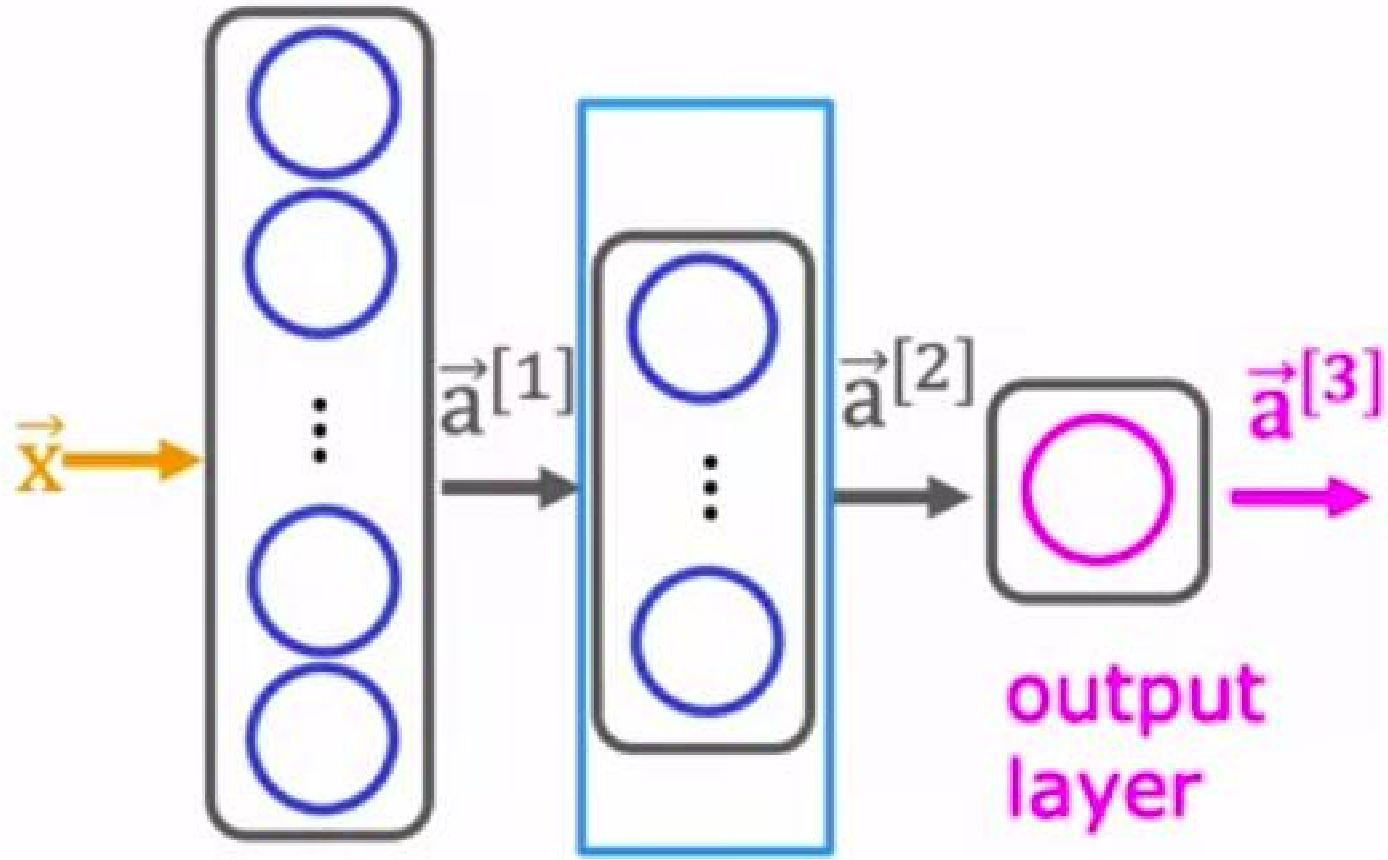
Stanford
ONLINE



Neural Network Model

Inference: making predictions (forward propagation)

Handwritten digit recognition



probability of being a handwritten '1'

25 units	15 units	1 unit
layer 1	layer 2	layer 3

Digit images

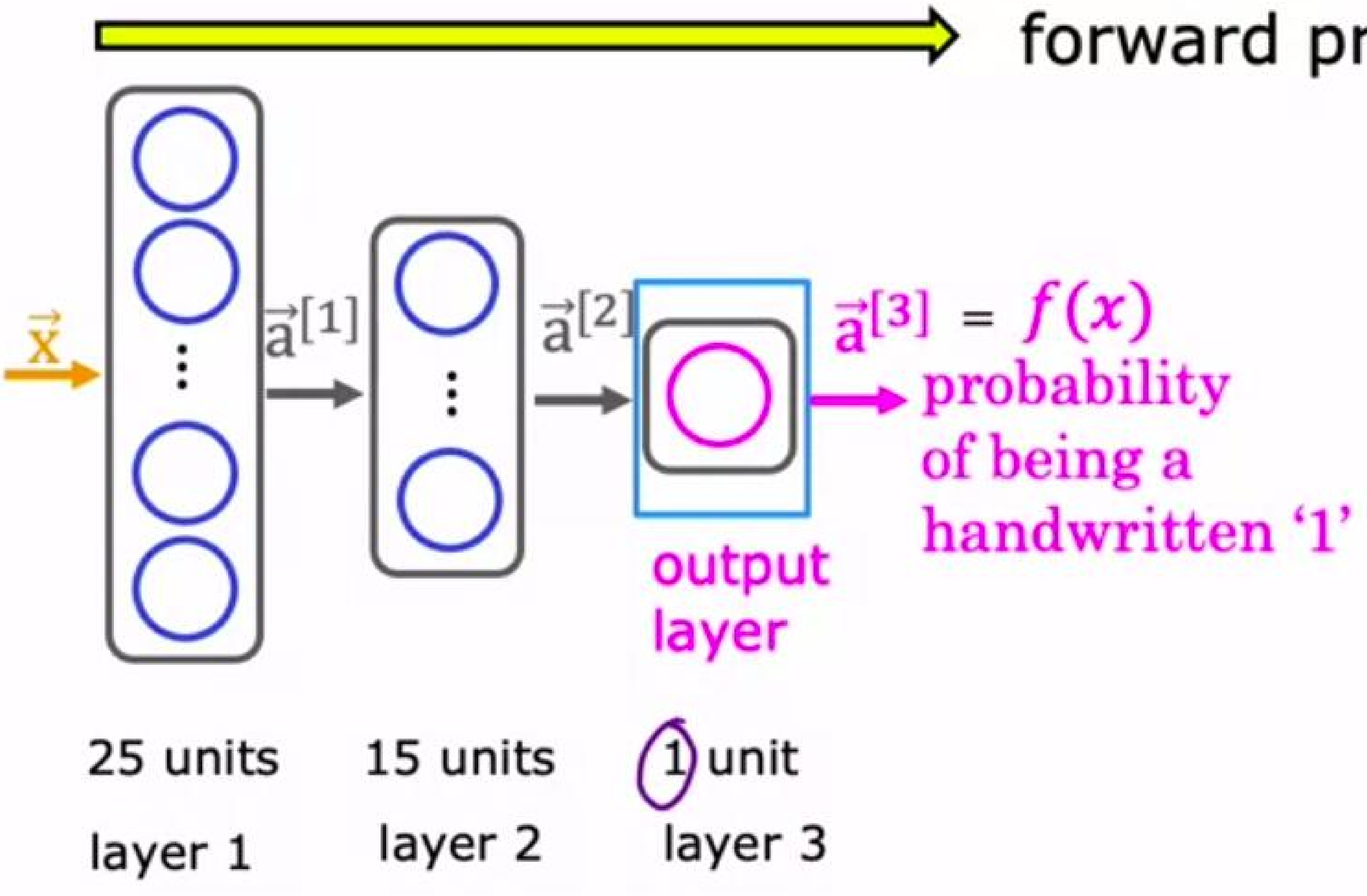
label 0 1

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

8 × 8

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

Handwritten digit recognition



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

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

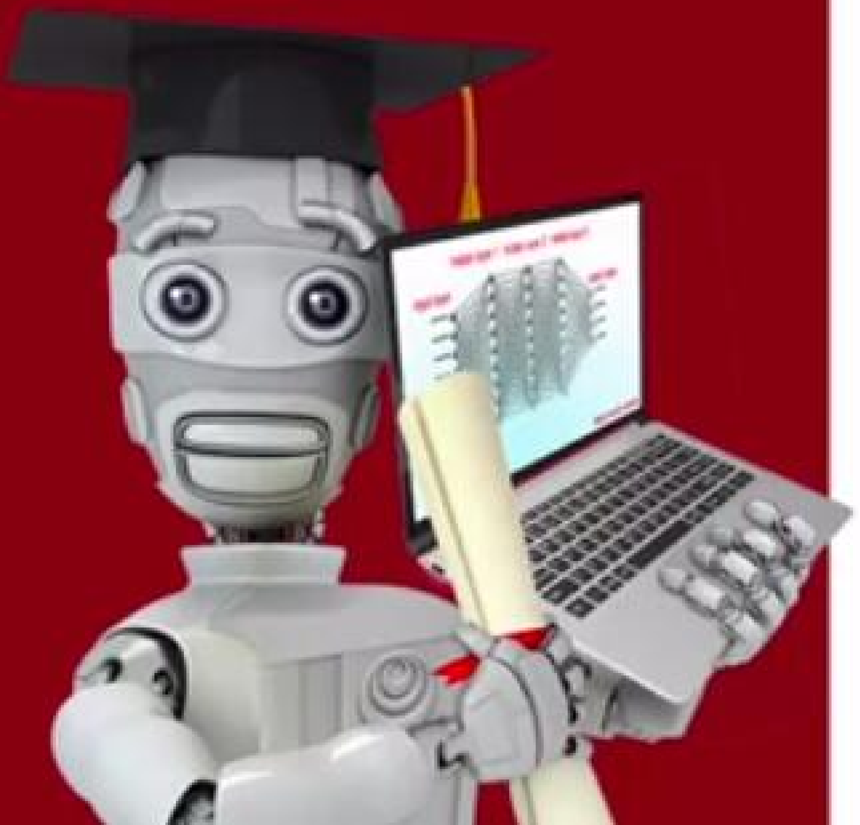
yes — no

$\hat{y} = 1$ — $\hat{y} = 0$

image is digit 1 — image isn't digit 1

DeepLearning.AI

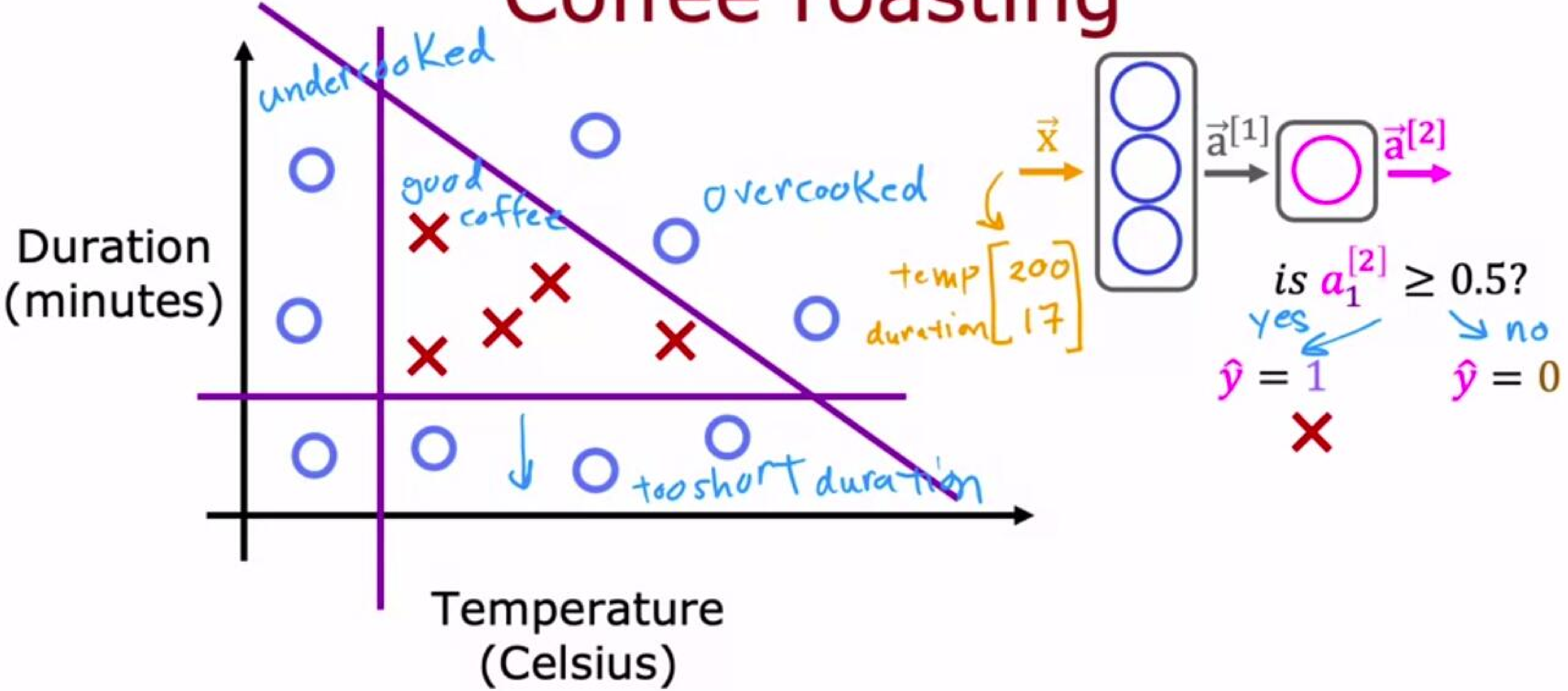
Stanford
ONLINE



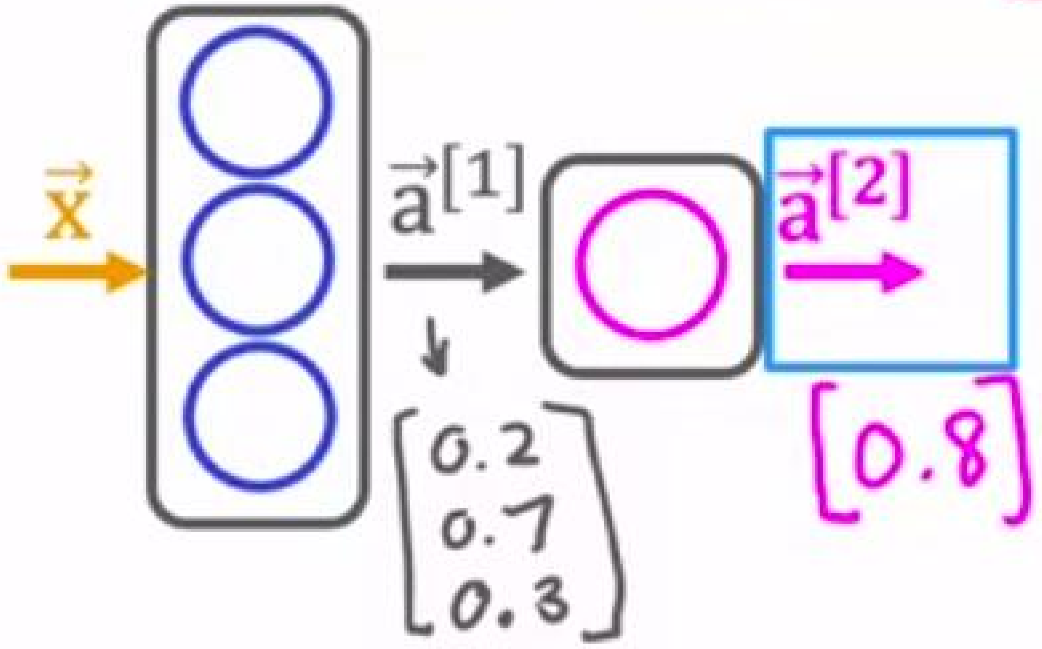
TensorFlow implementation

Inference in Code

Coffee roasting



Build the model using TensorFlow



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

yes → $\hat{y} = 1$

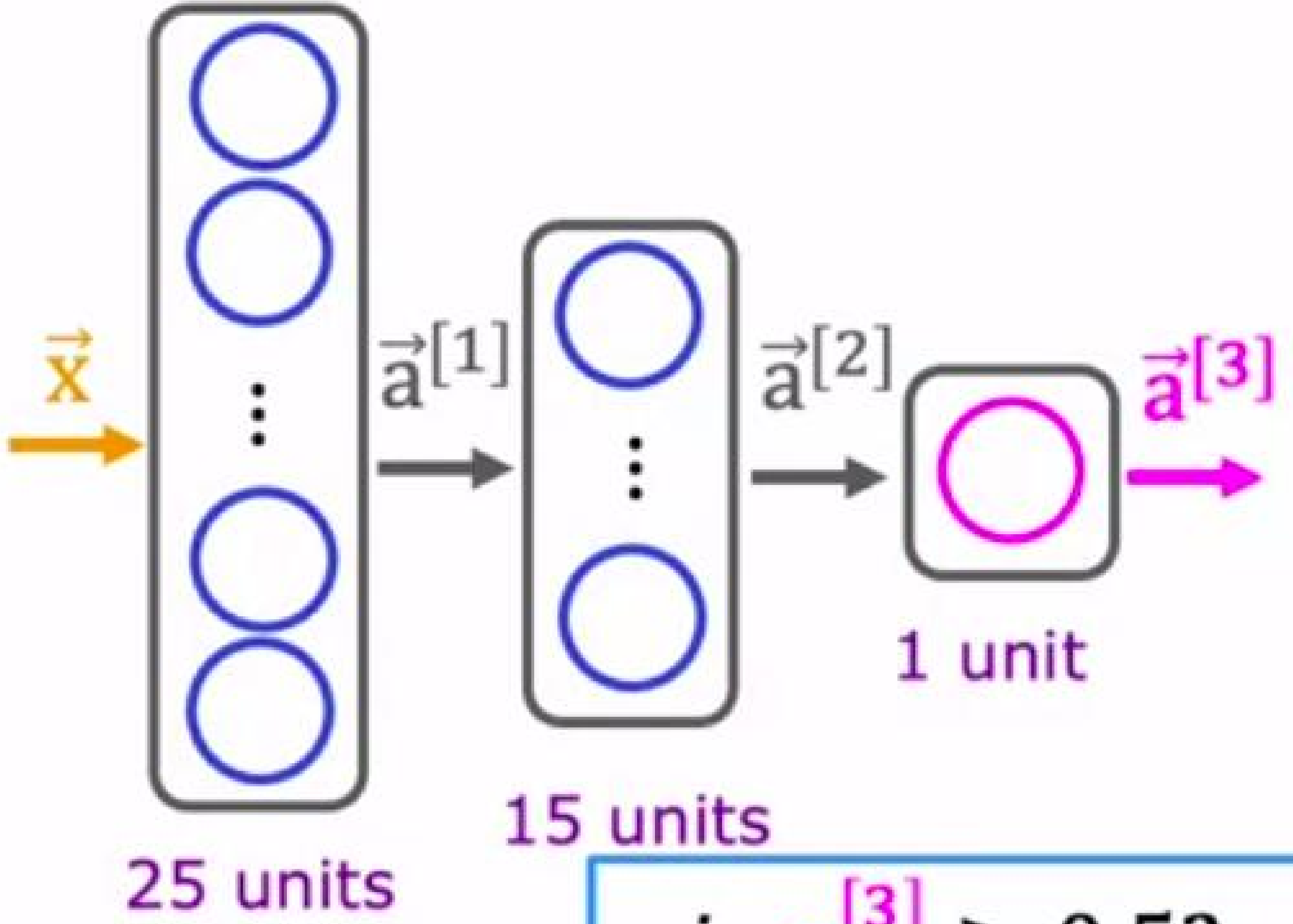
no → $\hat{y} = 0$

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

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

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

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)
```

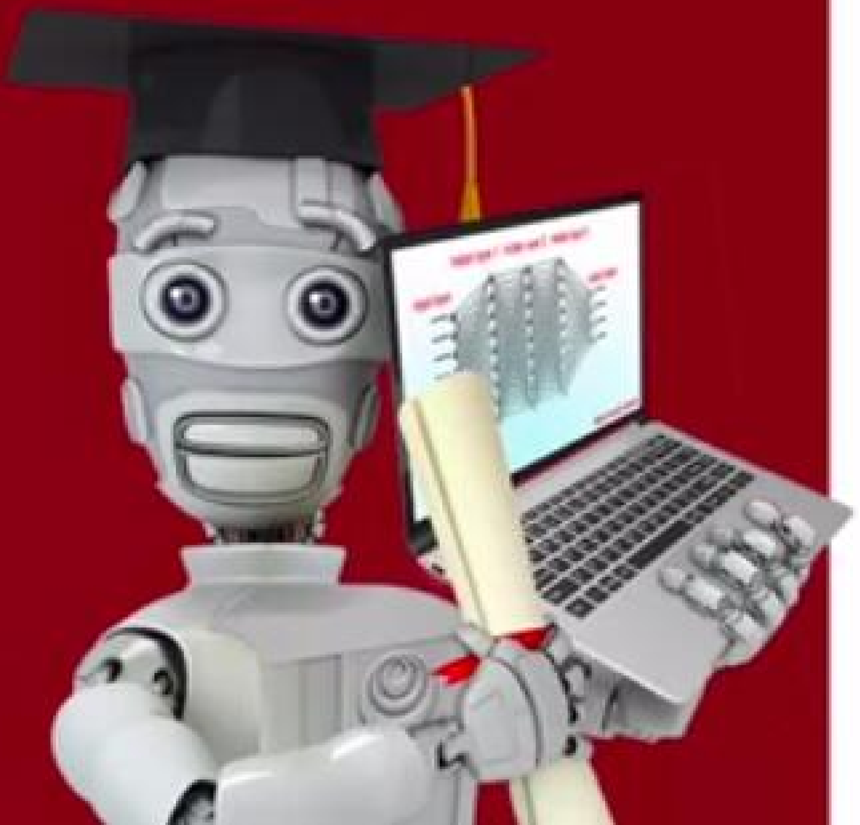
is $a_1^{[3]} \geq 0.5$?

$\hat{y} = 1$ $\hat{y} = 0$

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

DeepLearning.AI

Stanford ONLINE



TensorFlow implementation

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

$$\begin{array}{cc} & \text{3 columns} \\ \text{2 rows} & \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\end{array}$$

2 × 3 matrix

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

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

$$\begin{array}{cc} \text{4 rows} & \begin{bmatrix}0.1 & 0.2\\ -3 & -4\\ -.5 & -.6\\ 7 & 8\end{bmatrix}\end{array}$$

2 columns

4 × 2 matrix

```
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

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
...	...	...

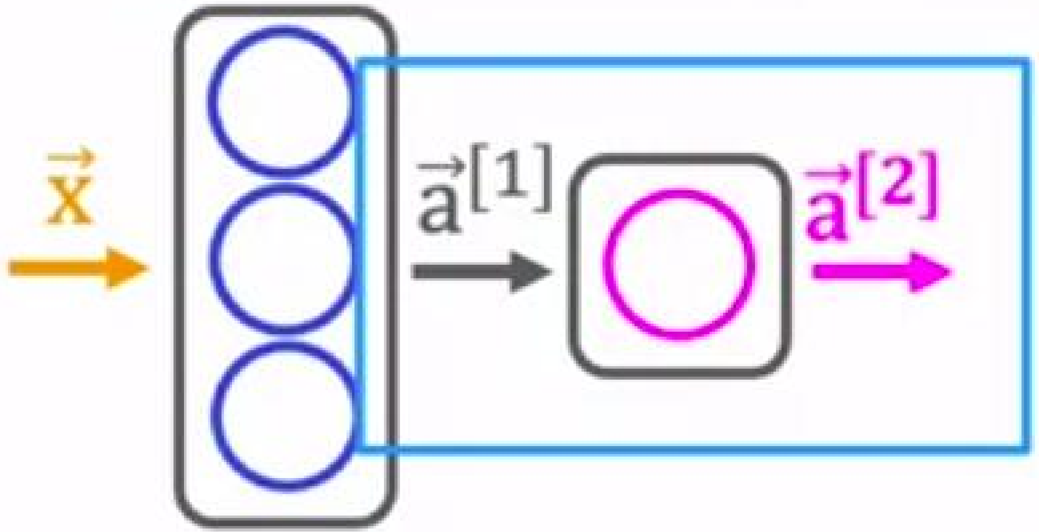
```
x = np.array([[200.0, 17.0]])
```

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

1 x 2

[200.0 17.0]

Activation vector




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

[[0.8]]

1 x 1

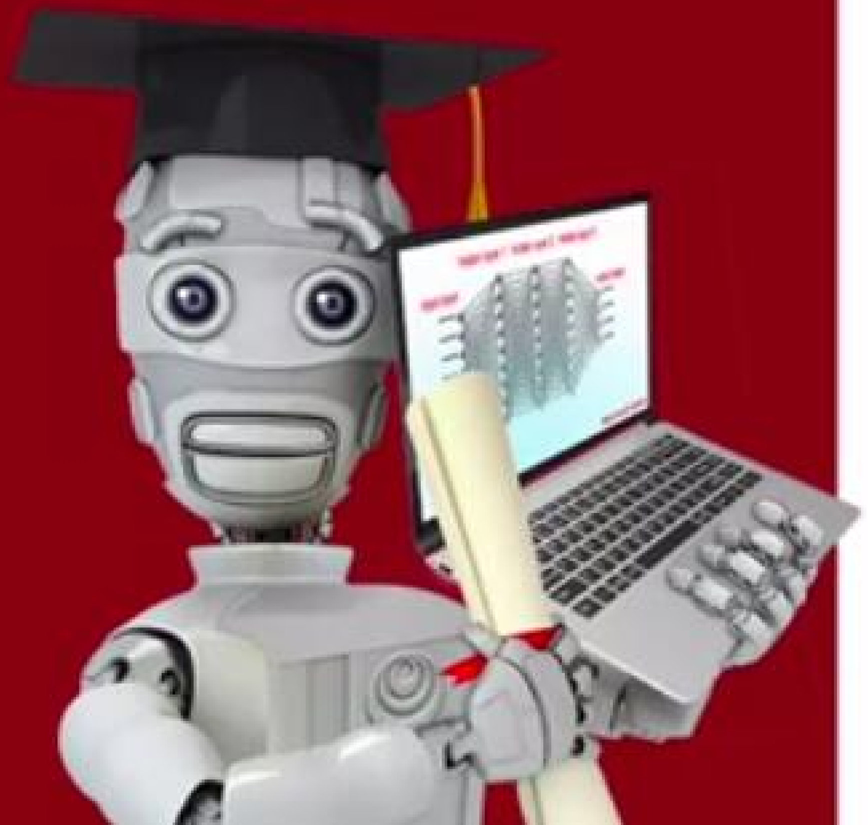
```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
a2.numpy()
```

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

DeepLearning.AI

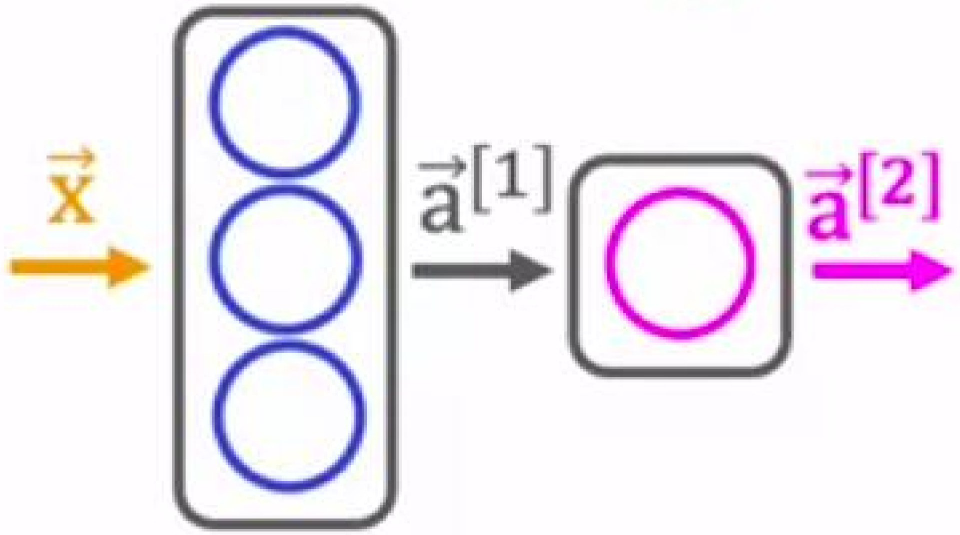
Stanford
ONLINE



TensorFlow implementation

Building a neural network

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])
```

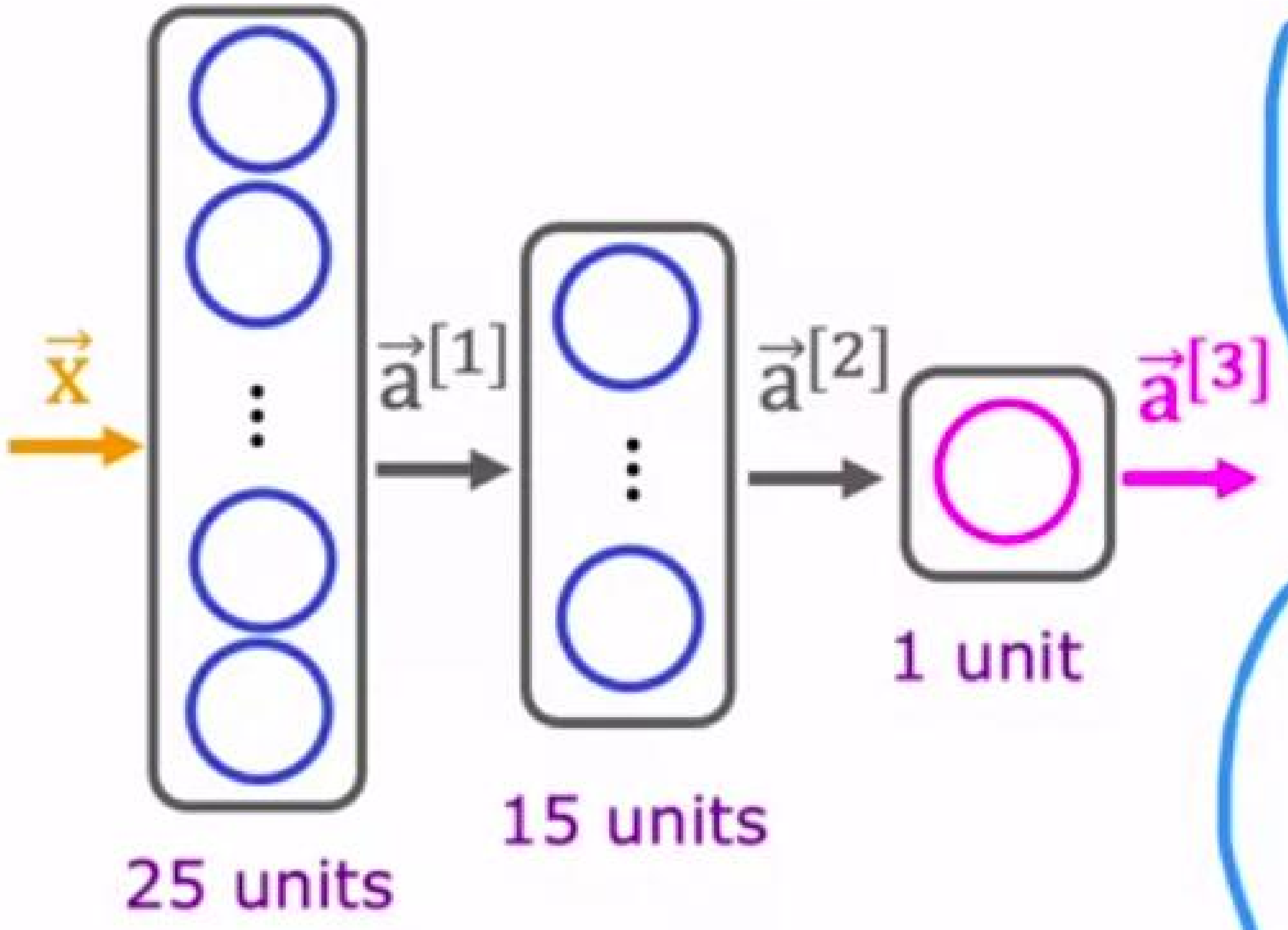
```
model.compile(...)
```

more about this next week!

```
model.fit(x,y)
```

```
model.predict(x_new)
```

Digit classification model



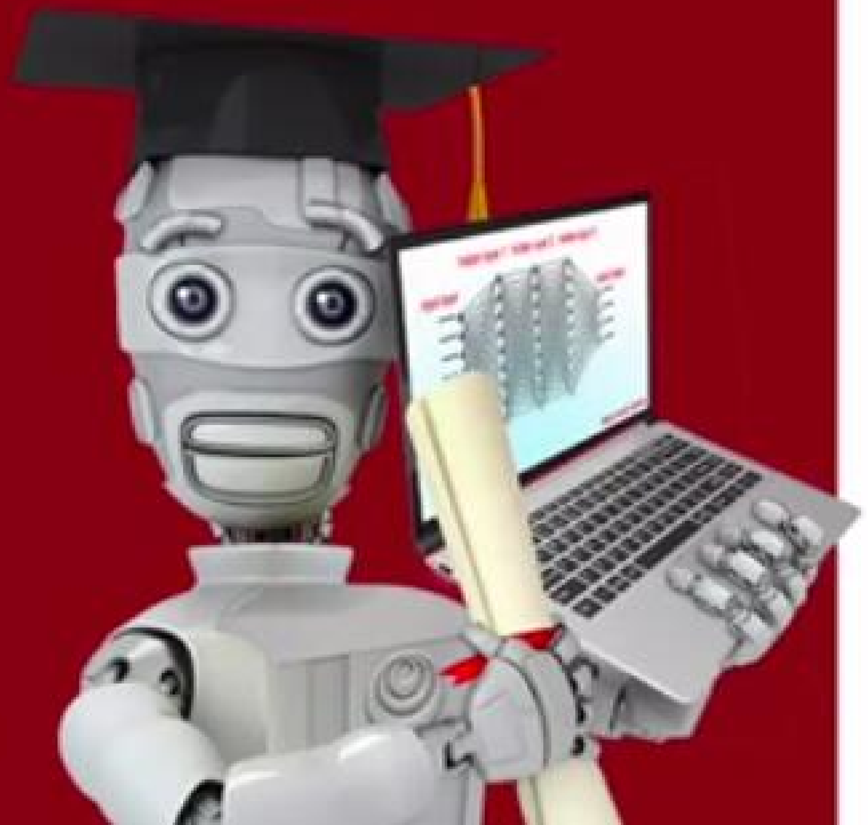

```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)

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!

DeepLearning.AI

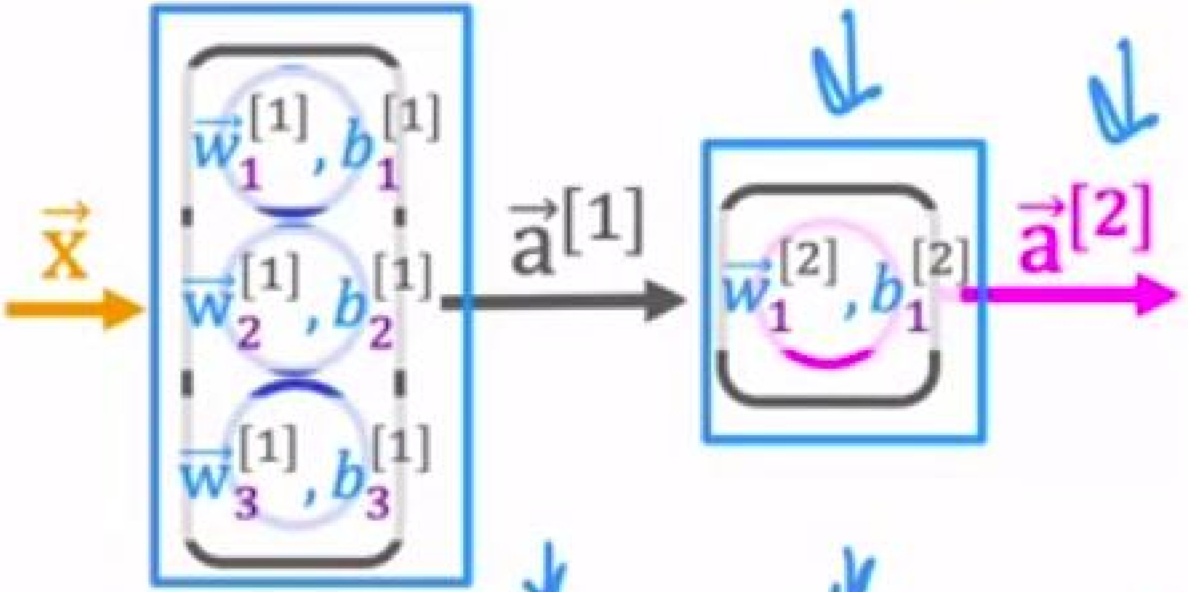
Stanford
ONLINE

Neural network implementation in Python



Forward prop in a single layer

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)
```

$w_1^{[2]}$ $\underbrace{\texttt{w2\_1}}$

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

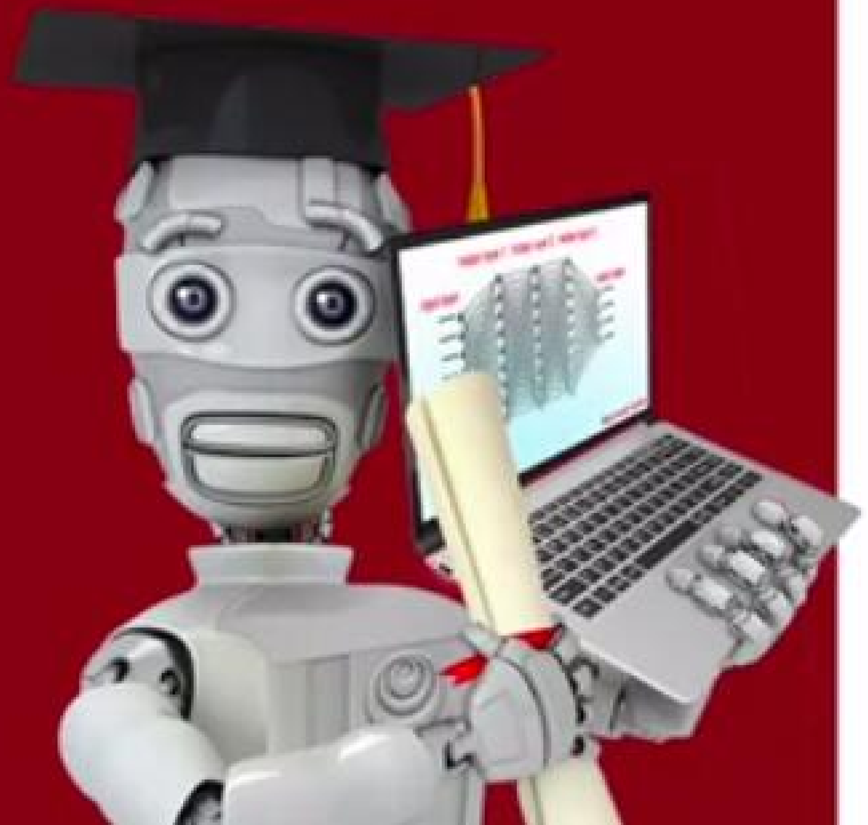
```
w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b1_2
a1_2 = sigmoid(z1_2)
```

```
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])
```

DeepLearning.AI

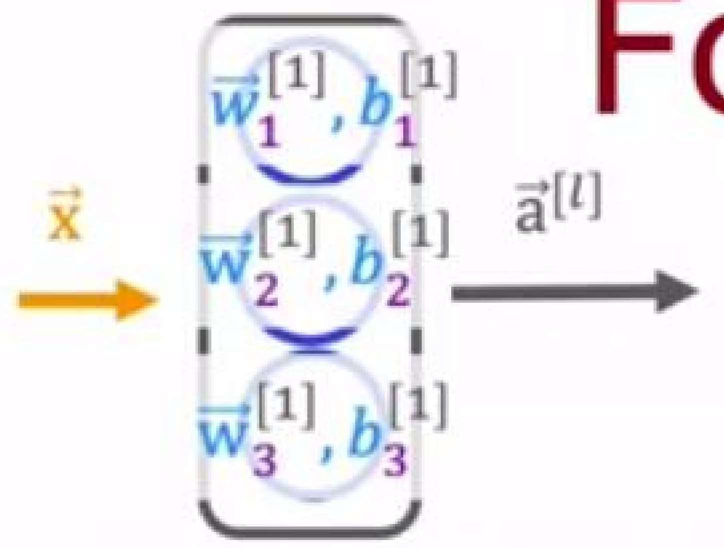
Stanford
ONLINE



Neural network implemenation in Python

General implementation of forward propagation

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):
  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
```

W

$a^{[1]}$

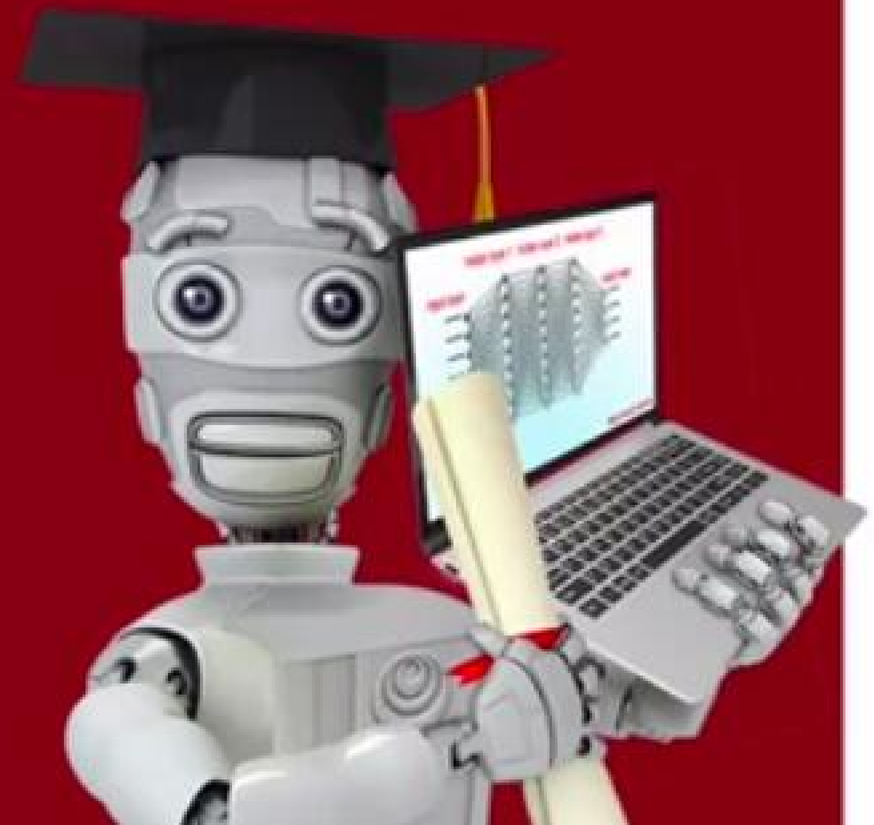
$a^{[2]}$

capital W refers to a matrix

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]
```

```
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

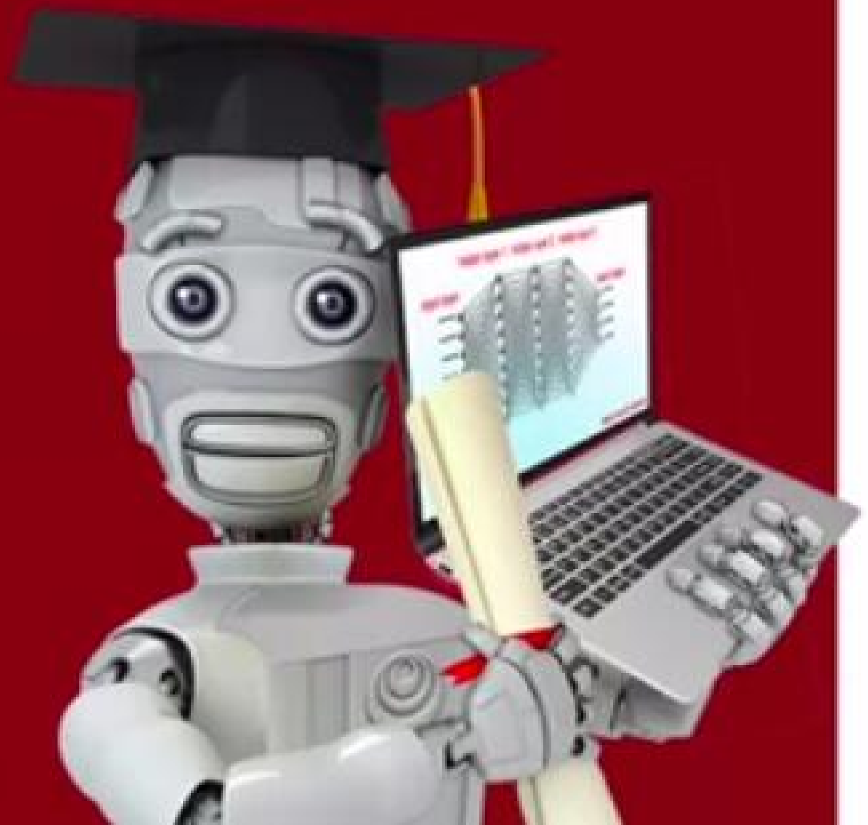
vectorized

matrix multiplication

DeepLearning.AI

Stanford ONLINE

Vectorization (optional)



Matrix multiplication

Dot products

example

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

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

$$3 + 8$$

$$11$$

in general

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

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

equivalent

transpose

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

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

vector vector multiplication

$$\underset{1\times2}{\begin{bmatrix}\leftarrow & \vec{a}^T & \rightarrow\end{bmatrix}} \begin{bmatrix}\uparrow\\ \vec{w}\\ \downarrow\end{bmatrix} 2\times1$$

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

useful for understanding matrix multiplication

Vector matrix multiplication

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

$\vec{a}^T = \begin{bmatrix} \boxed{1 \quad 2} \end{bmatrix}$ $\quad W = \begin{bmatrix} \boxed{\begin{matrix} 3 \\ 4 \end{matrix}} & \boxed{\begin{matrix} 5 \\ 6 \end{matrix}} \end{bmatrix}$ $\quad Z = \vec{a}^T W \quad [\leftarrow \;\; \vec{a}^T \;\; \rightarrow] \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$

1 by 2

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

$(1 * 3) + (2 * 4)$ $\qquad$ $(1 * 5) + (2 * 6)$

$3 + 8$ $\qquad$ $5 + 12$

11 $\qquad$ 17

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

matrix matrix multiplication

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

$$A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} \leftarrow & \vec{a}_1^T & \rightarrow \\ \leftarrow & \vec{a}_2^T & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

rows   columns

row1 col1   row1 col2

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

row2 col1   row2 col2

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

$-3 \quad + \quad -8$

-11

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

$-5 \quad + \quad -12$

-17

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

general rules for
matrix multiplication
↳ next video!

DeepLearning.AI

Stanford ONLINE

Vectorization (optional)

Matrix multiplication rules

Matrix multiplication rules

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

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

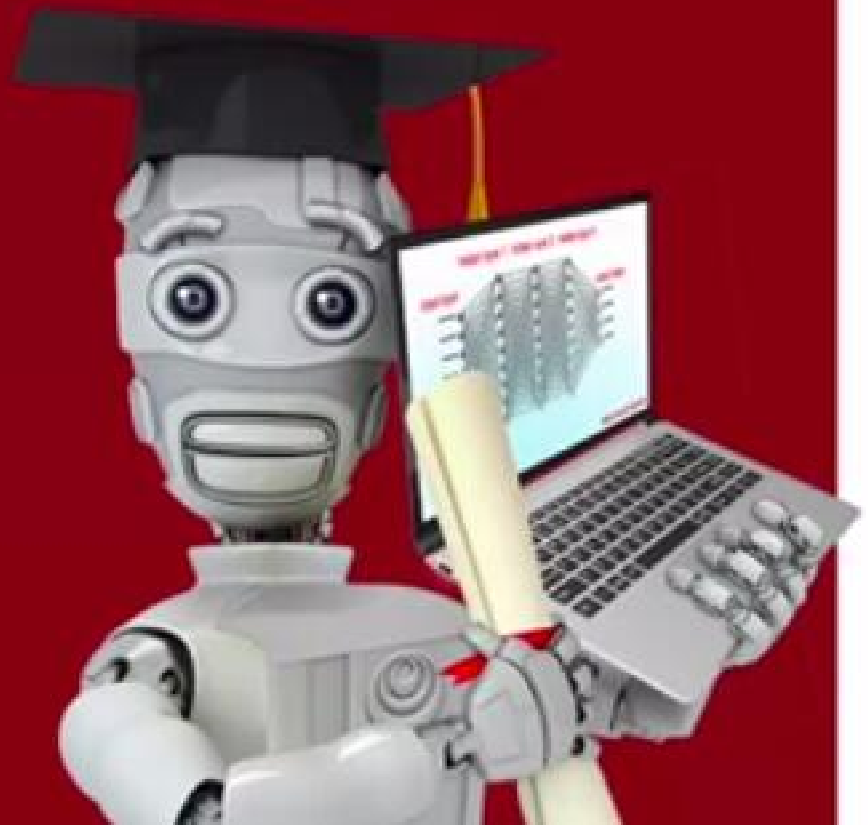
3 by 4 matrix

↳ same # rows as A^T

same # columns as W

DeepLearning.AI

Stanford ONLINE



Vectorization (optional)

Matrix multiplication code

Matrix multiplication in NumPy

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

```
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]])
```

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

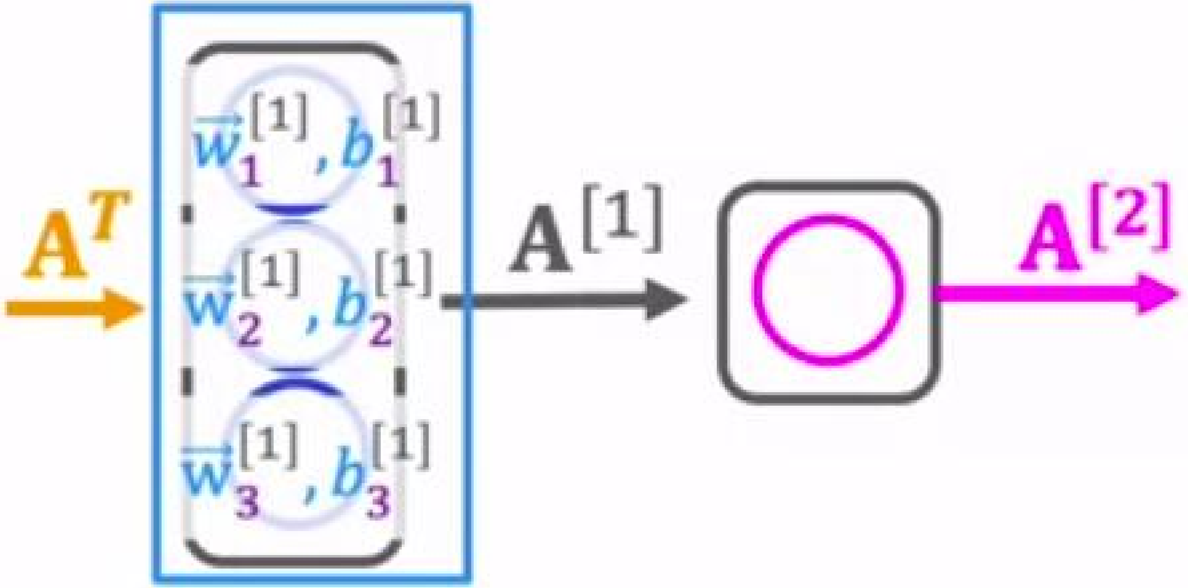
or

```
Z = AT @ W
```

result

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

Dense layer vectorized



$A^T = [200 \quad 17]$

1 x 2

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

2 x 3

$B = [-1 \quad 1 \quad 2]$

1 x 3

$Z = A^T W + B$

$$\begin{array}{ccc} [\boxed{165} & \boxed{-531} & \boxed{900}] \\ z_1^{[1]} & z_2^{[1]} & z_3^{[1]} \end{array}$$

$A = g(Z)$

$[1 \quad 0 \quad 1]$

A

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

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

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

a_in

```
def dense(AT,W,b):
  z = np.matmul(AT,W) + b
  a_out = g(z)
  return a_out
```

a_in

```
[[1,0,1]]
```