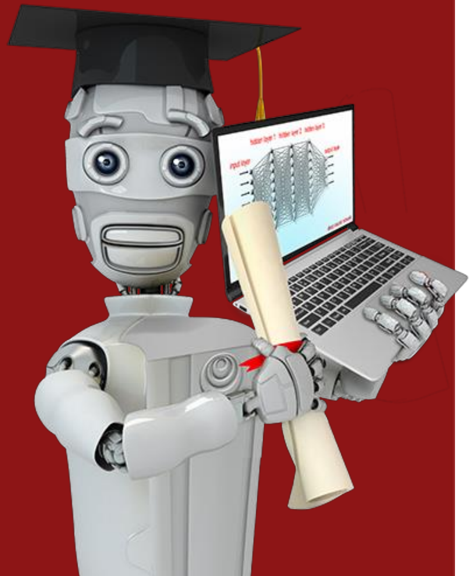
Copyright Notice

DeepLearning.AI

Stanford ONLINE



Advanced Learning Algorithms

Welcome!

Advanced learning algorithms

Neural Networks
- inference (prediction)
- training

Practical advice for building machine learning systems

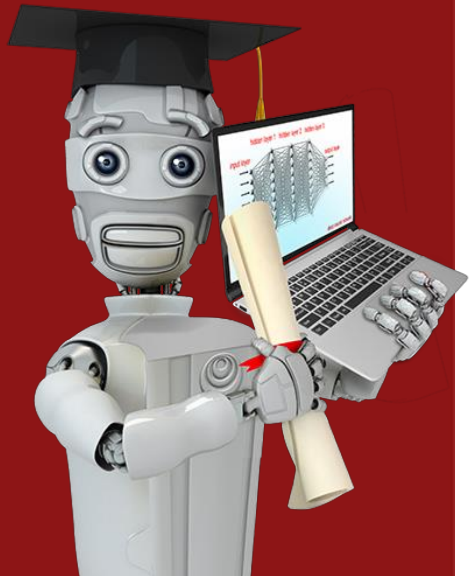
Decision Trees

DeepLearning.AI

Stanford ONLINE

Neural Networks Intuition

Neurons and the brain



Neural networks

Origins: Algorithms that try to mimic the brain.

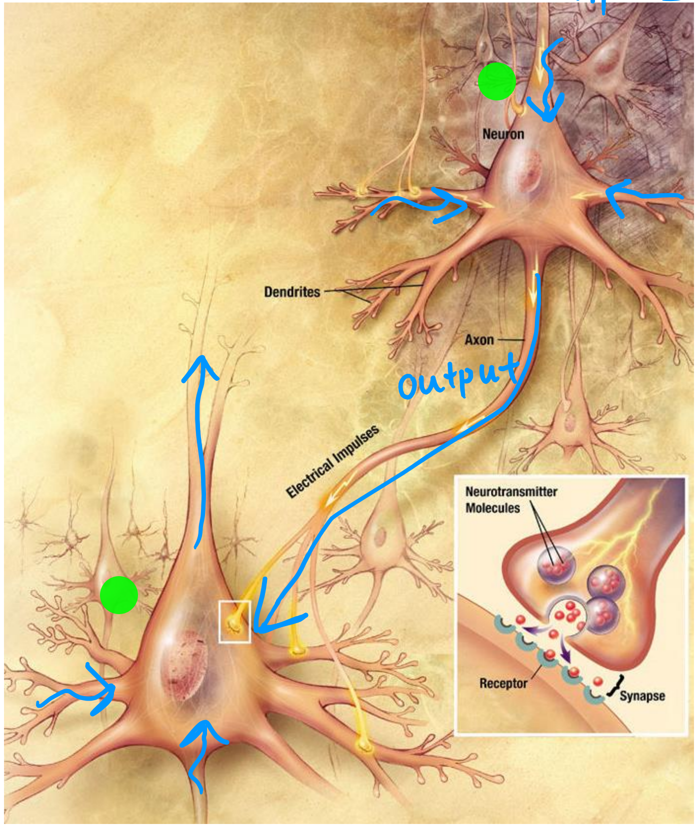
Used in the 1980's and early 1990's.
Fell out of favor in the late 1990's.

Resurgence from around 2005.

speech → images → text (NLP) → ...

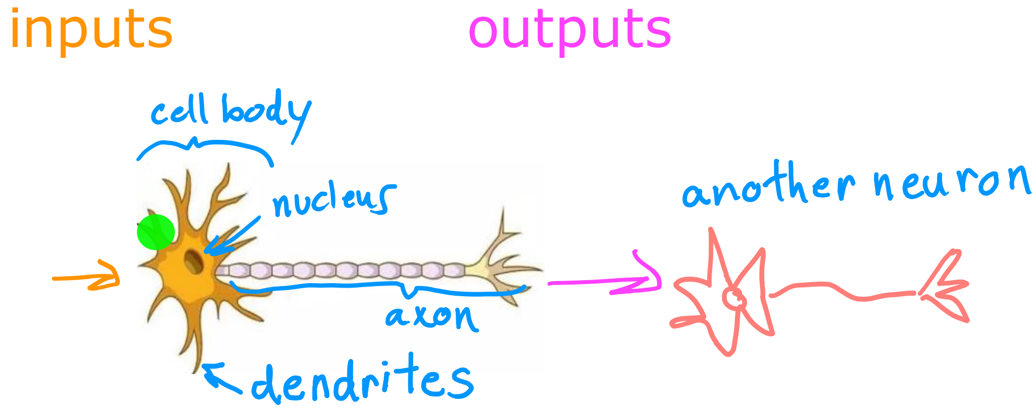
Neurons in the brain

inputs



[Credit: US National Institutes of Health, National Institute on Aging]

Biological neuron



Simplified mathematical model of a neuron

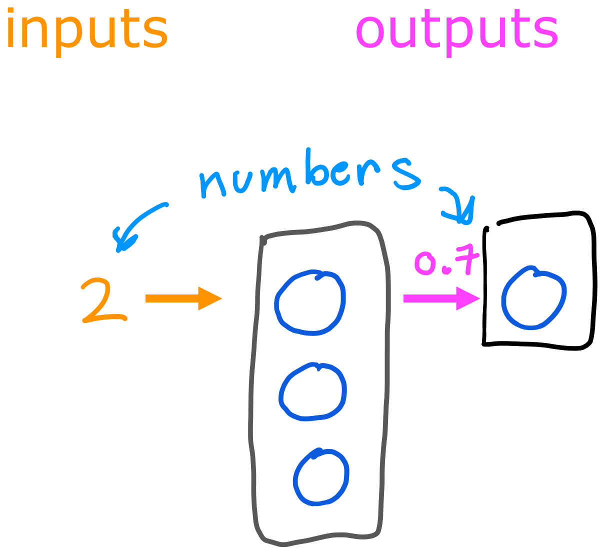
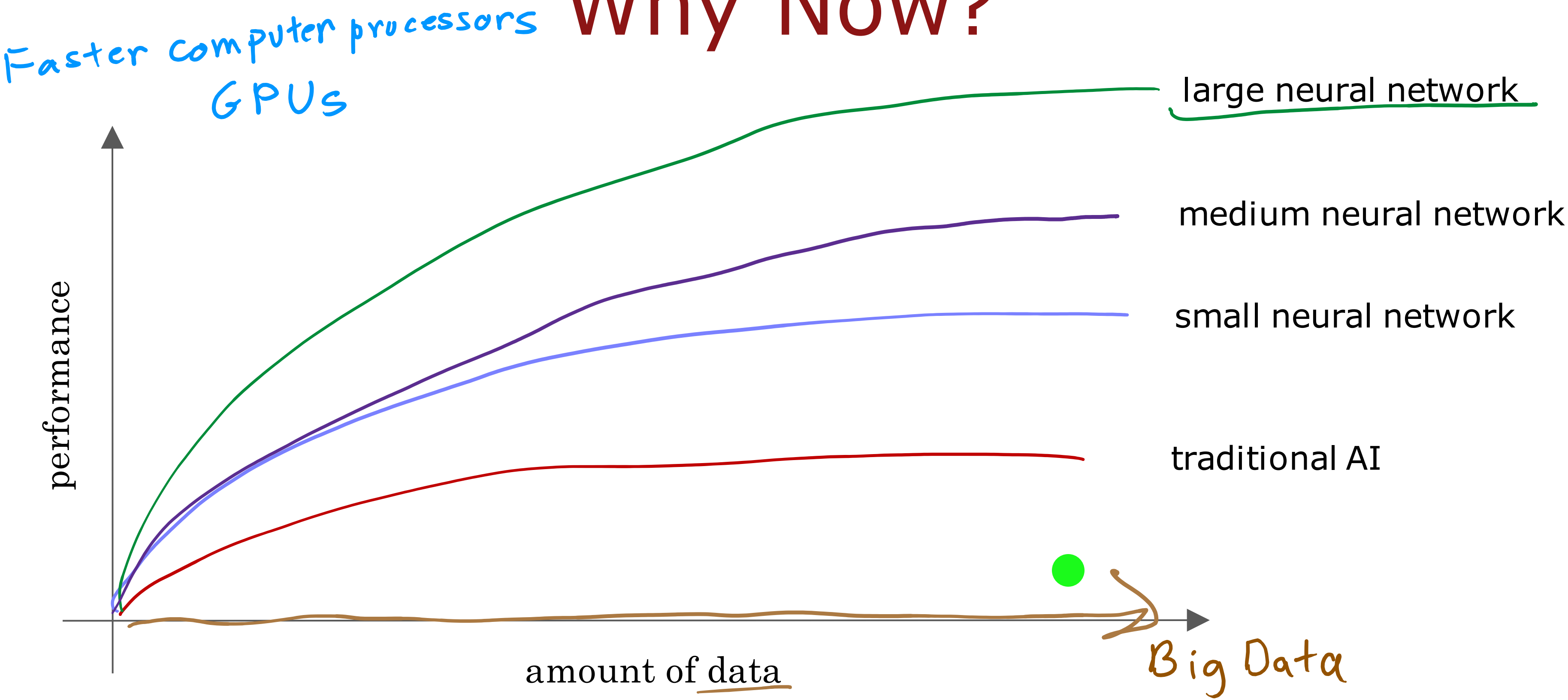


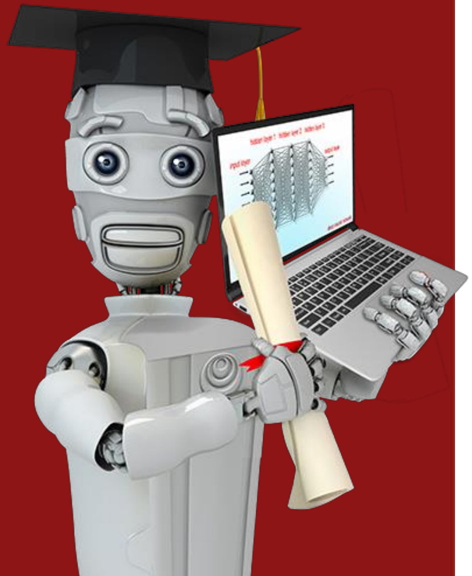
image source: https://biologydictionary.net/sensory-neuron/

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

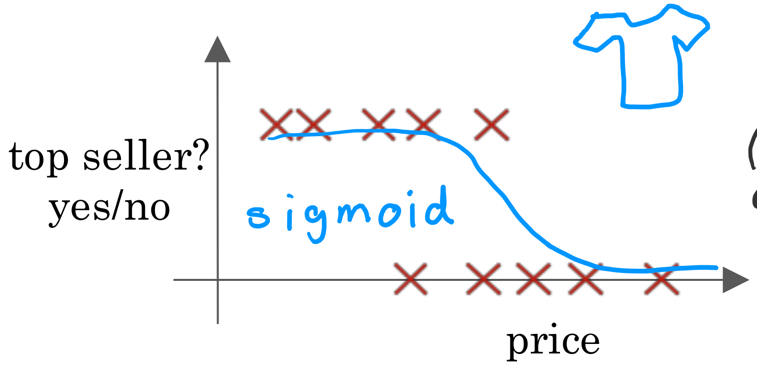
Stanford ONLINE



Neural Network Intuition

Demand Prediction

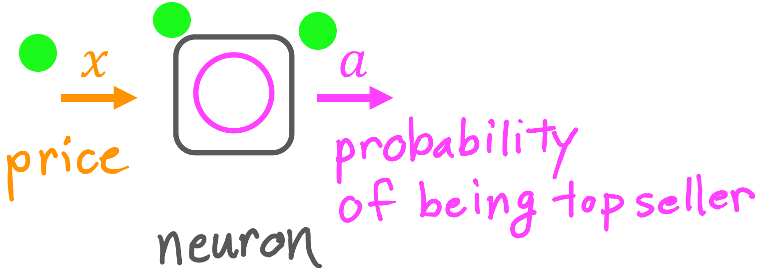
Demand Prediction



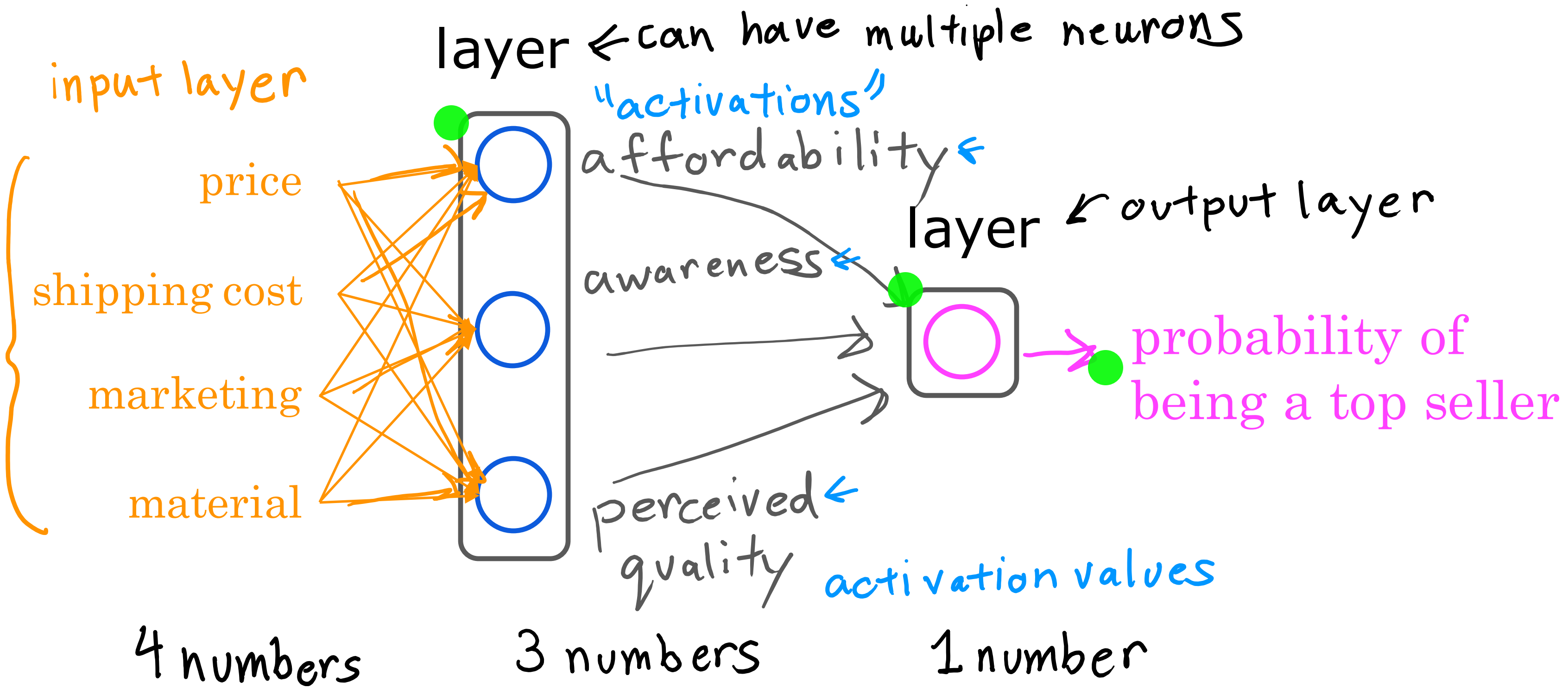
$$x = price$$ input

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

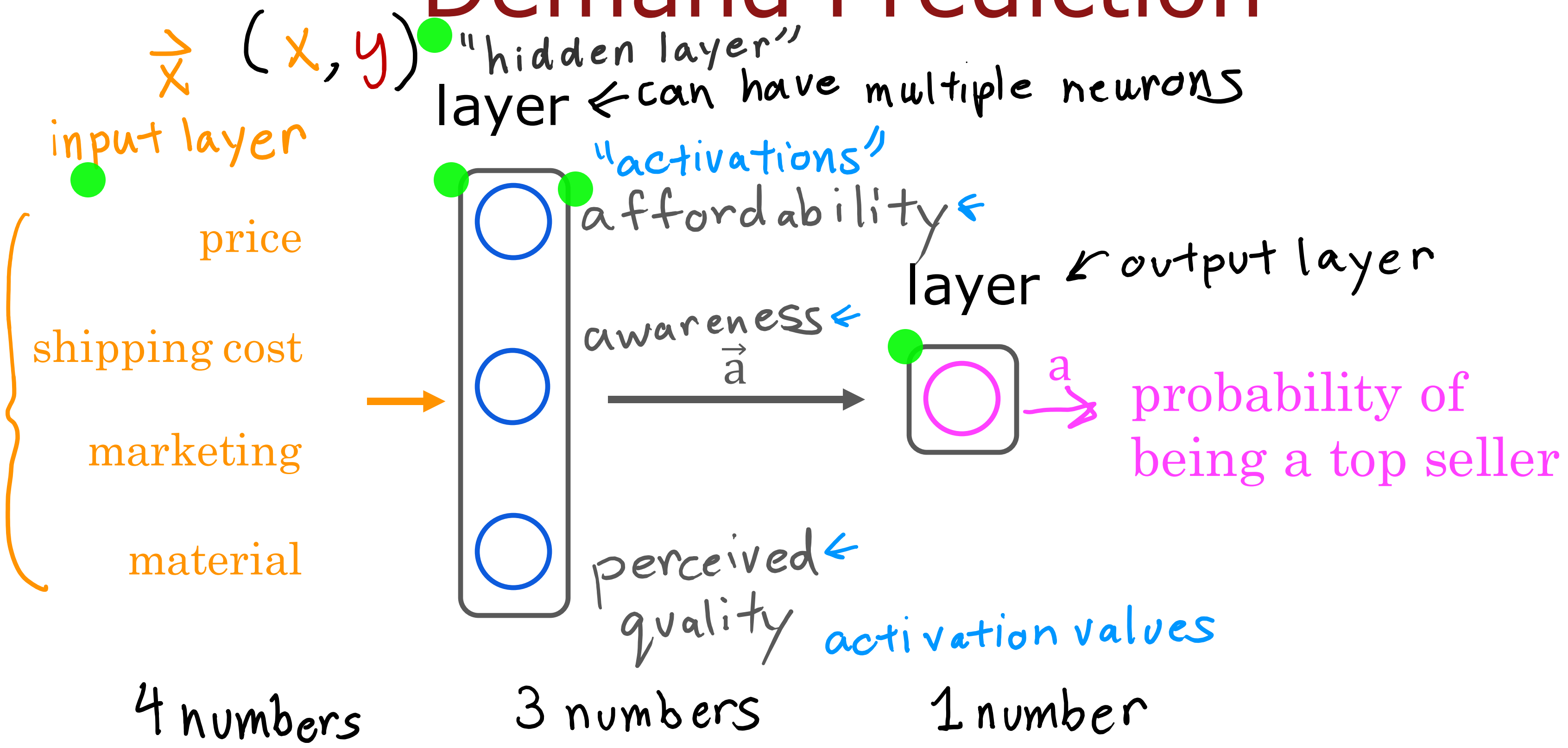
activation



Demand Prediction



Demand Prediction



Demand Prediction

feature engineering

$x_1 x_2$

$\vec{x}$ (x, y)

input layer

price

shipping cost

marketing

material

"hidden layer"

layer ← can have multiple neurons

"activations"

affordability ←

awareness ←

perceived quality ←

$\vec{a}$

layer ← output layer

a

probability of being a top seller

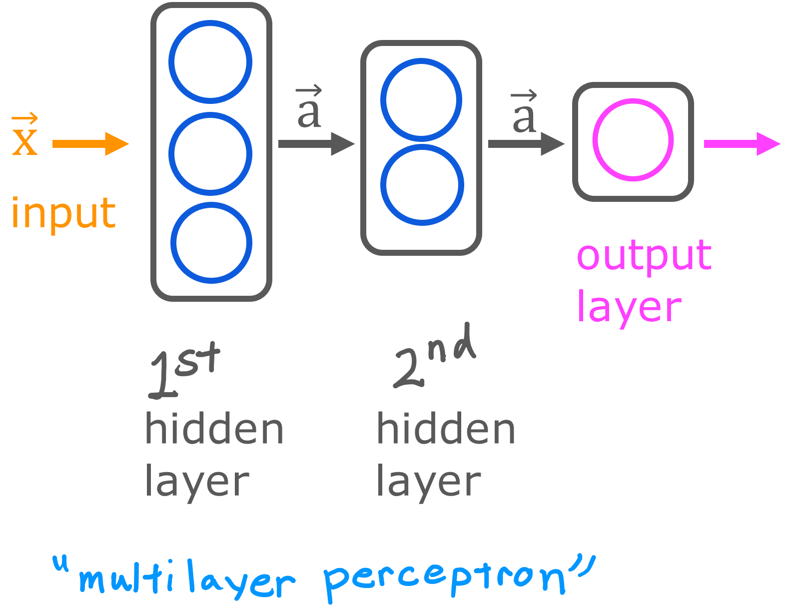
activation values

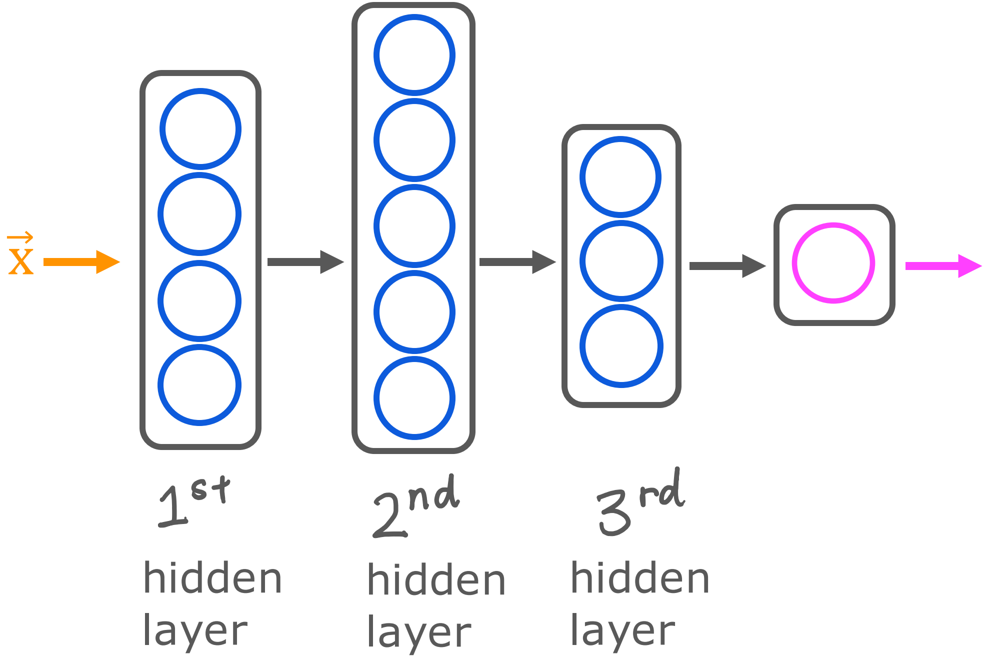
4 numbers

3 numbers

1 number

Multiple hidden layers

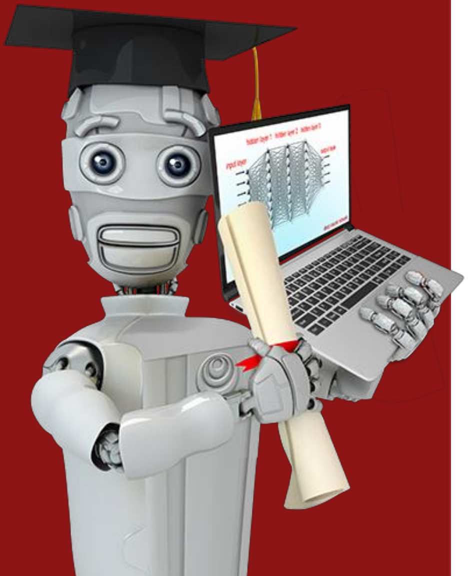




neural network architecture

DeepLearning.AI

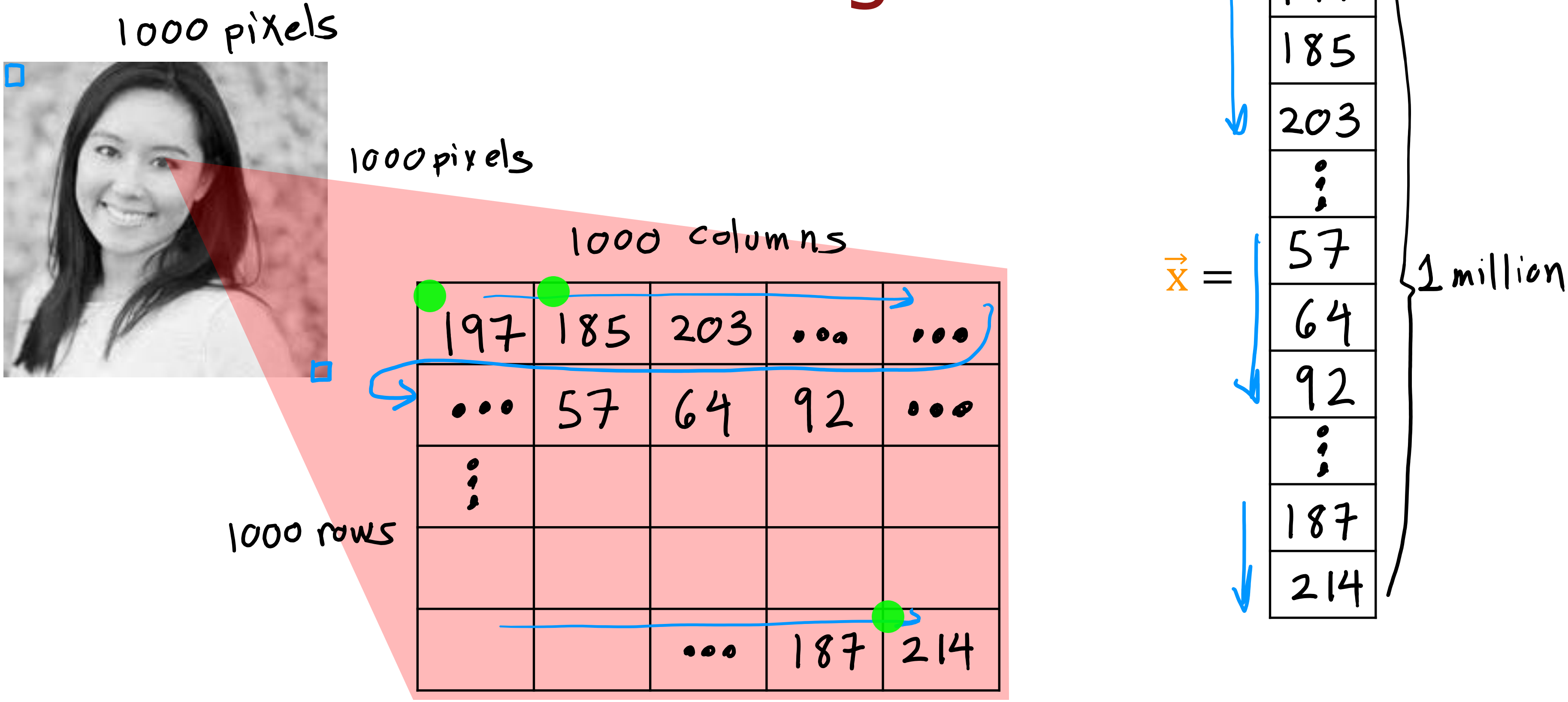
Stanford
ONLINE



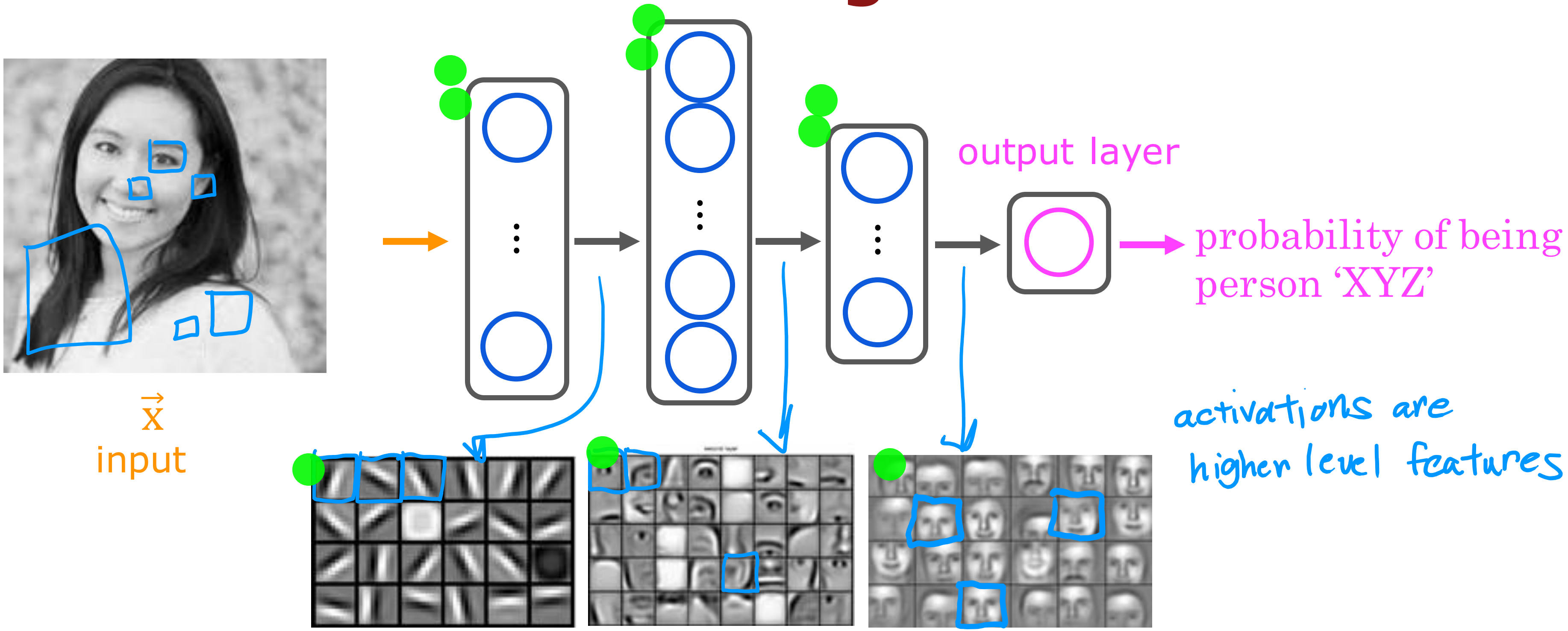
Neural Networks Intuition

Example:
Recognizing Images

Face recognition

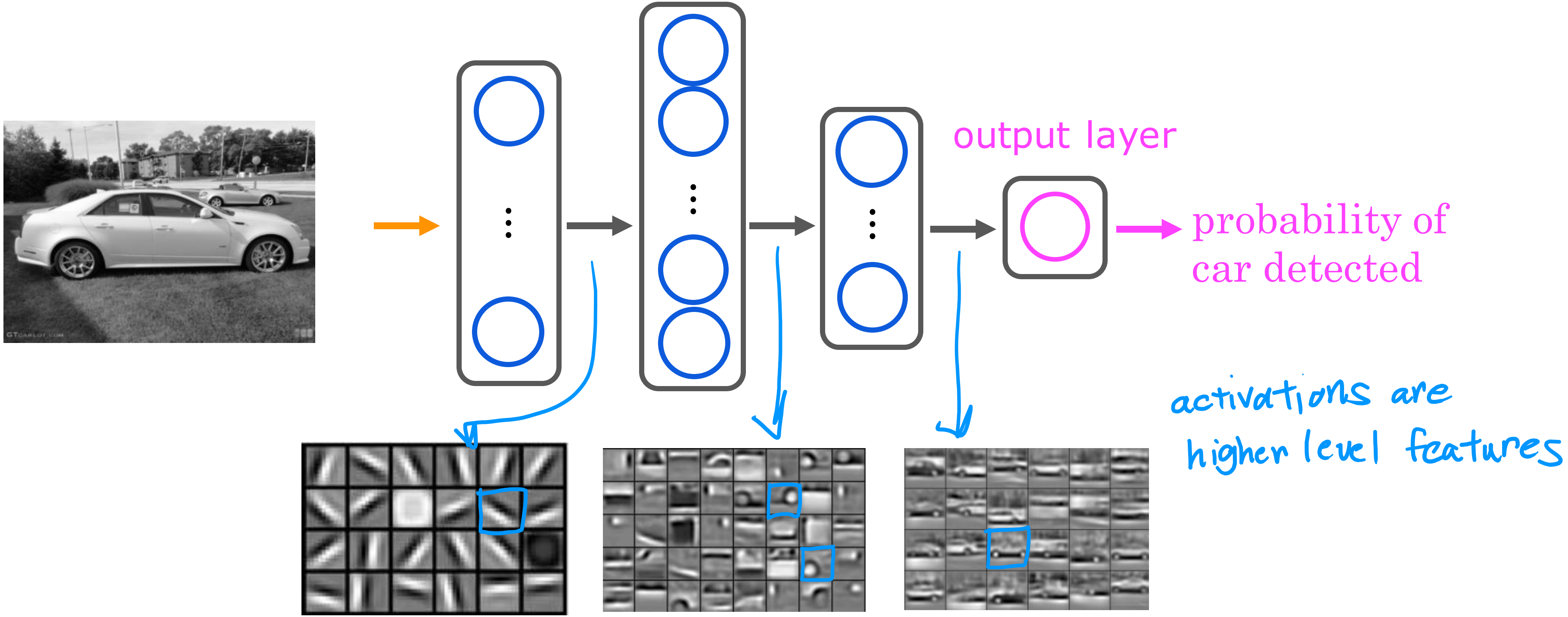


Face recognition



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

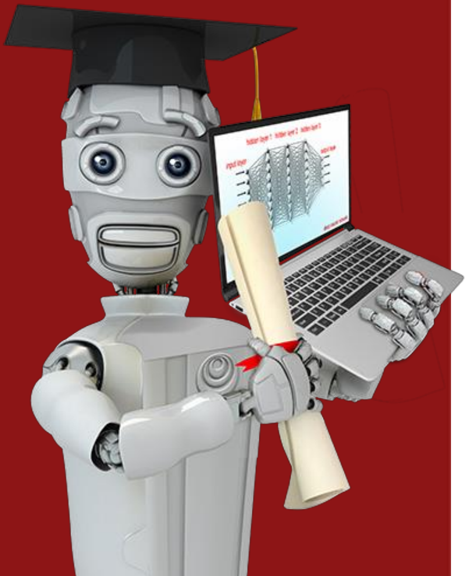
Car classification



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

DeepLearning.AI

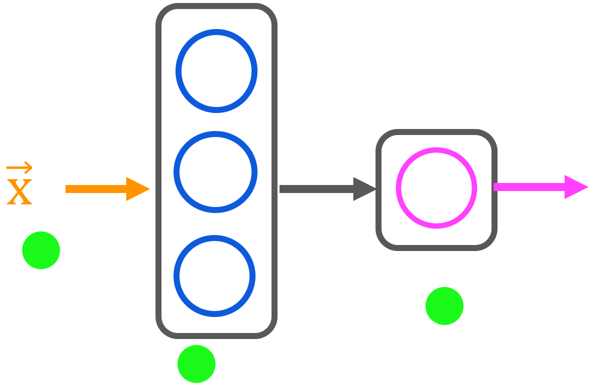
Stanford
ONLINE



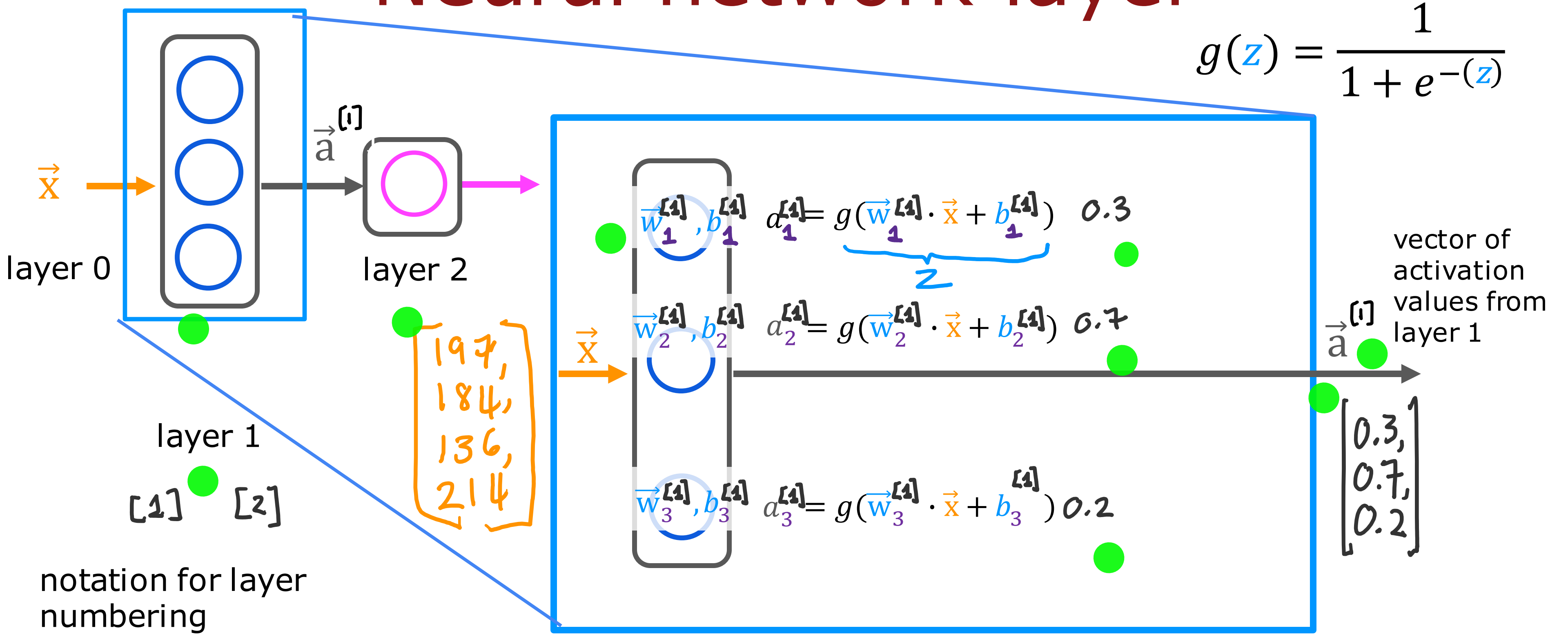
Neural network model

Neural network layer

Neural network layer

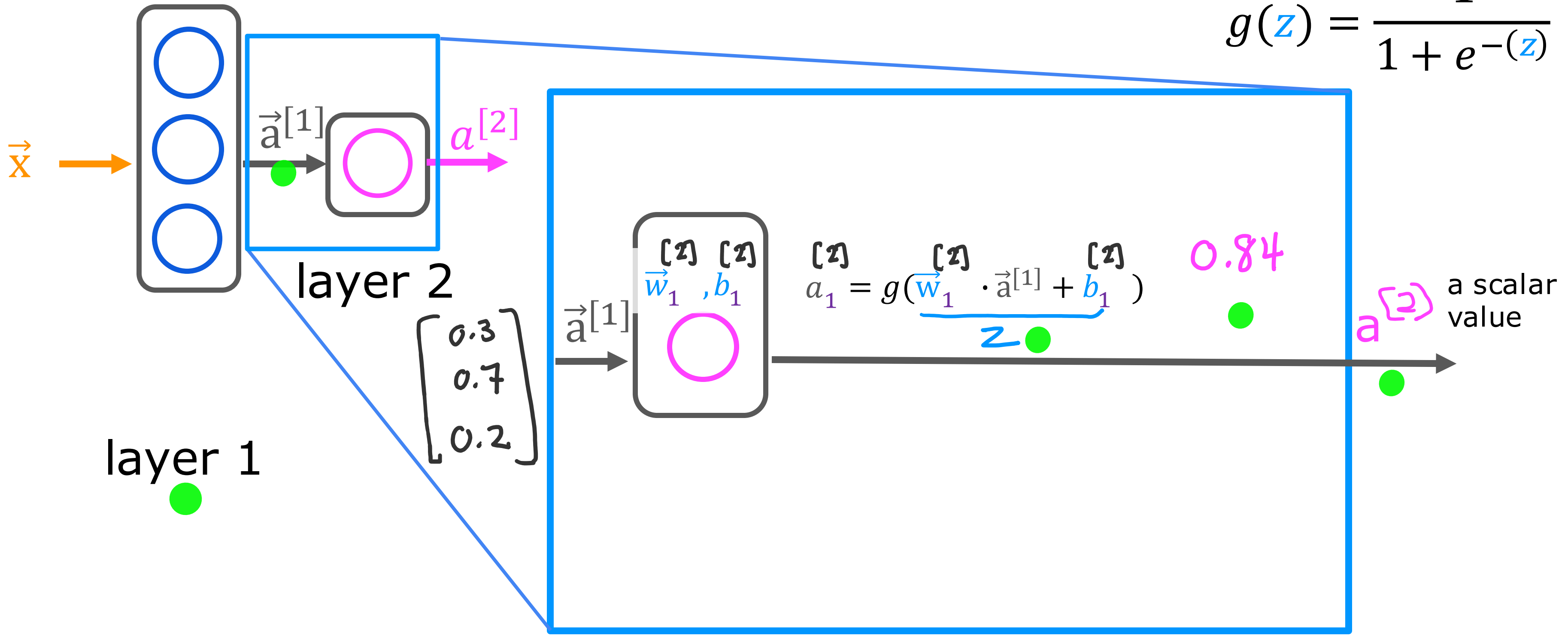


Neural network layer



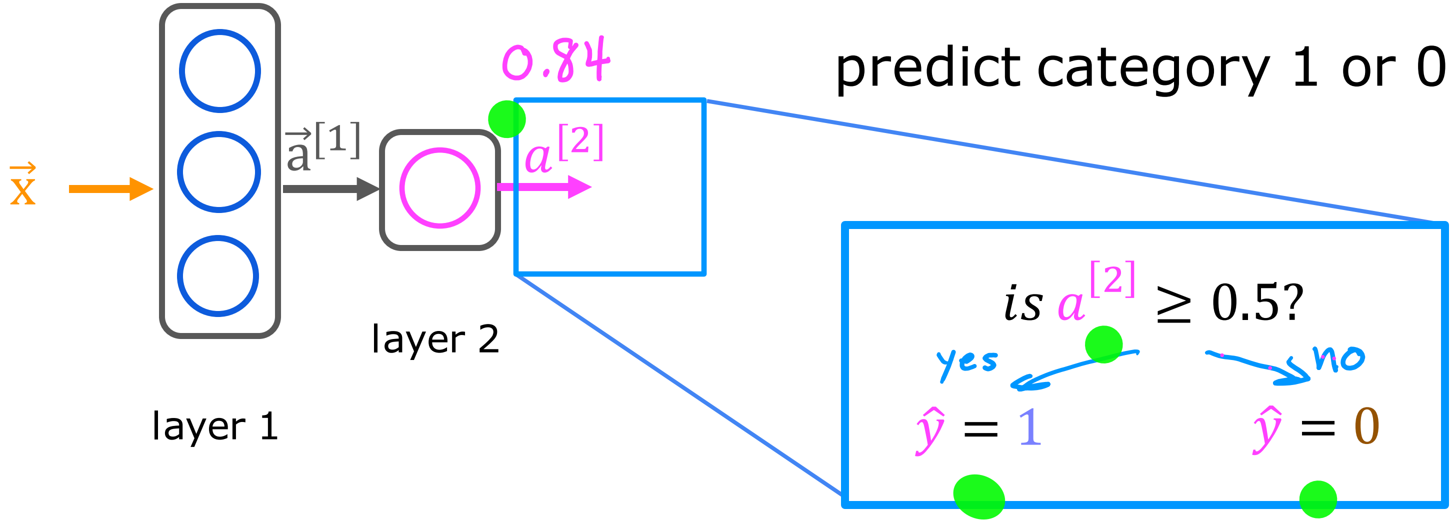
Neural network layer

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



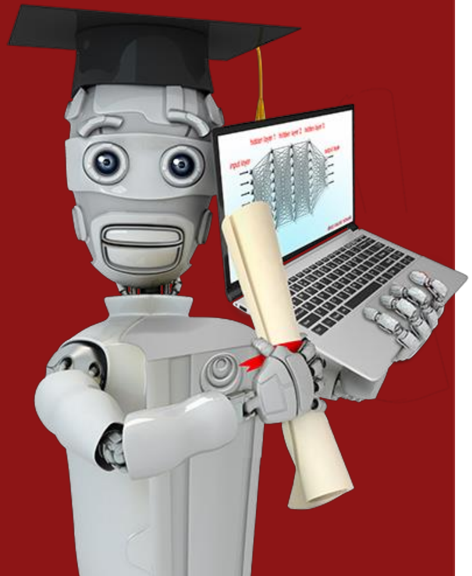
Neural network layer

predict category 1 or 0 (yes/no)



DeepLearning.AI

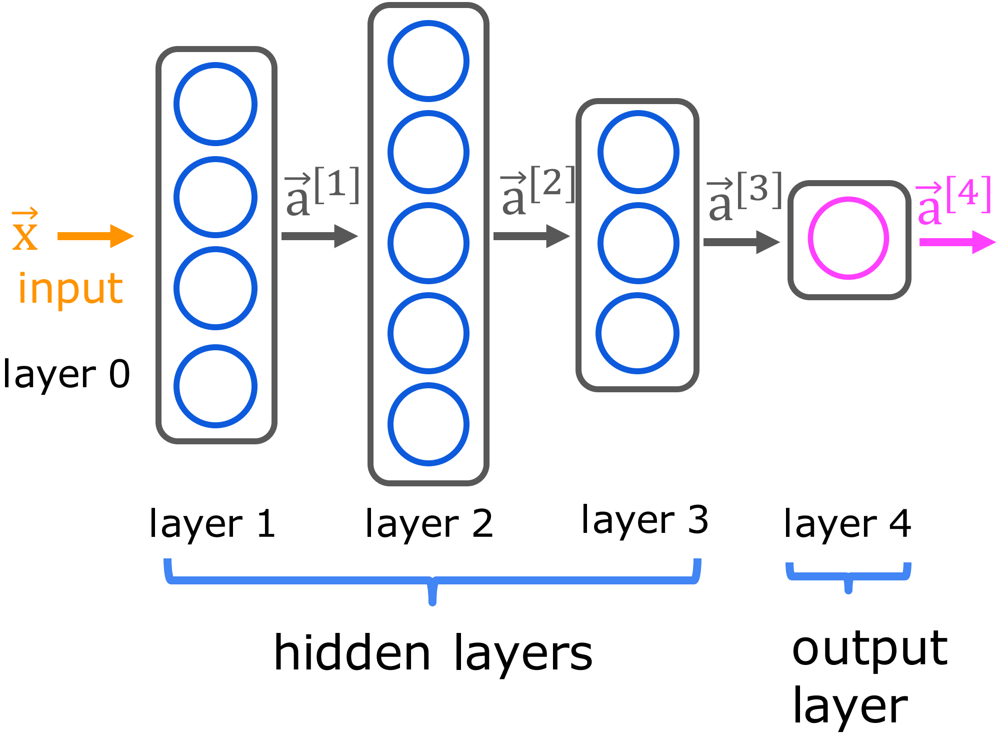
Stanford ONLINE



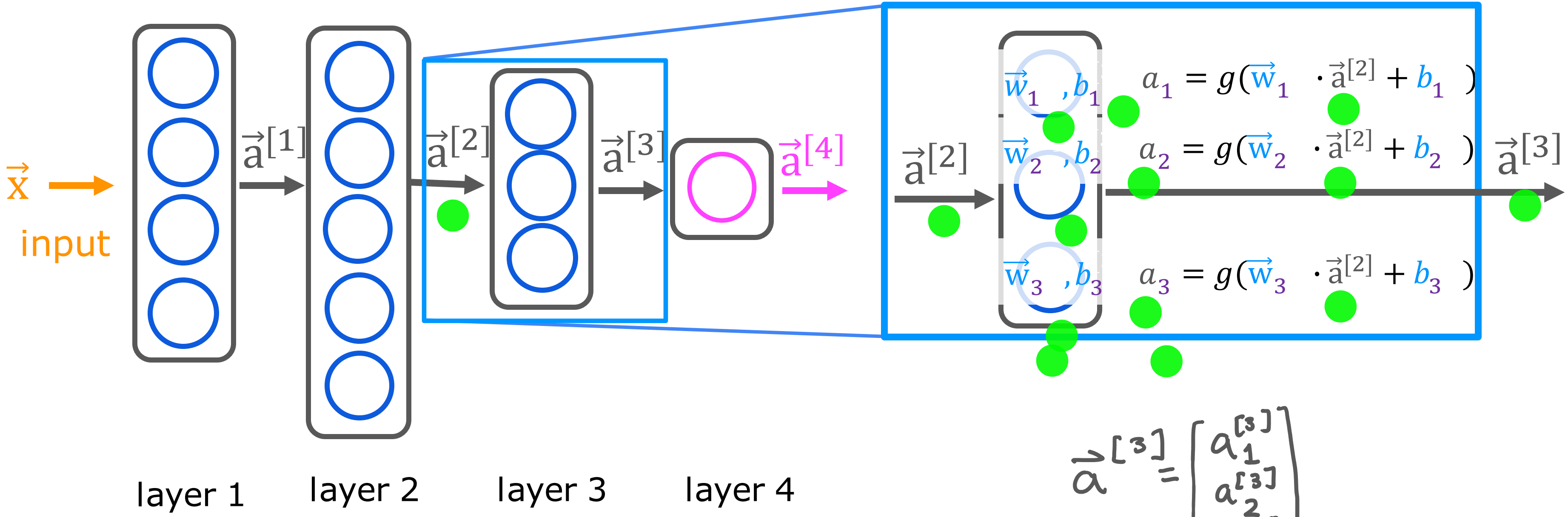
Neural Network Model

More complex neural networks

More complex neural network

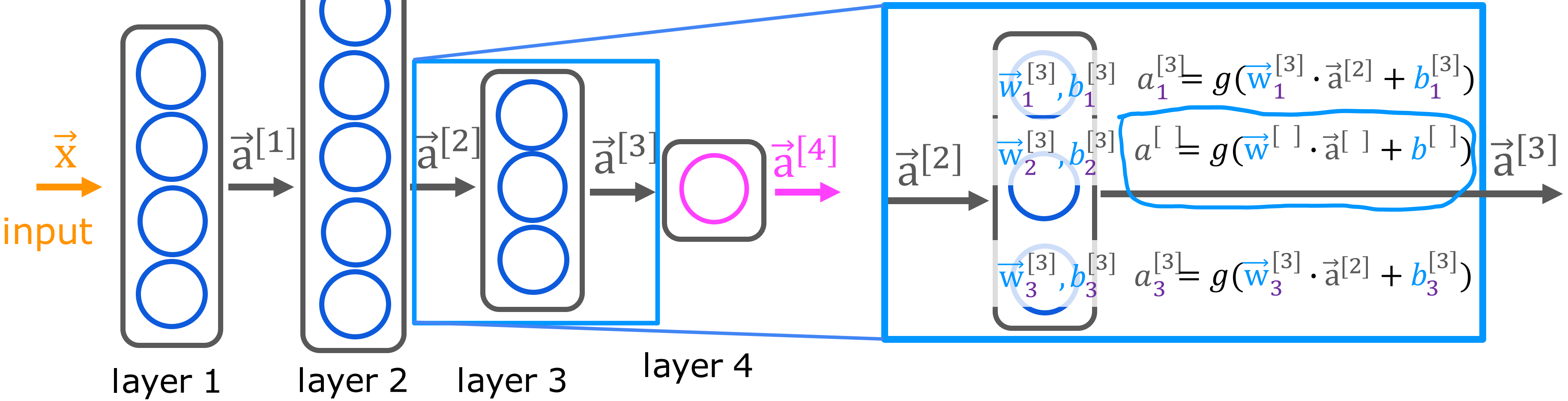


More complex neural network



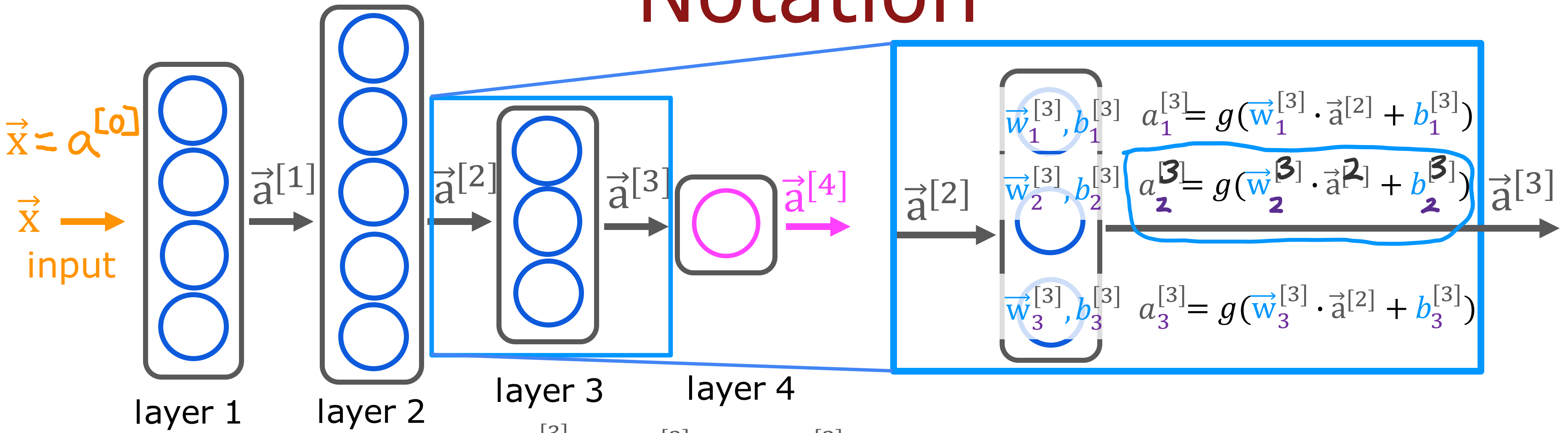
$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$$

Notation



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

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)

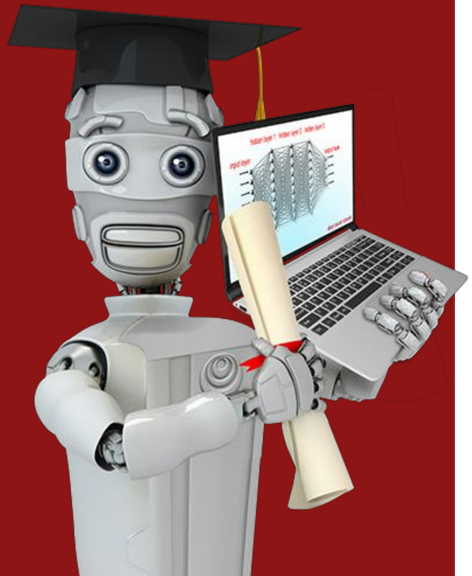
$$\boxed{a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})}$$

sigmoid
"activation function"

Parameters w & b of layer l, unit j

DeepLearning.AI

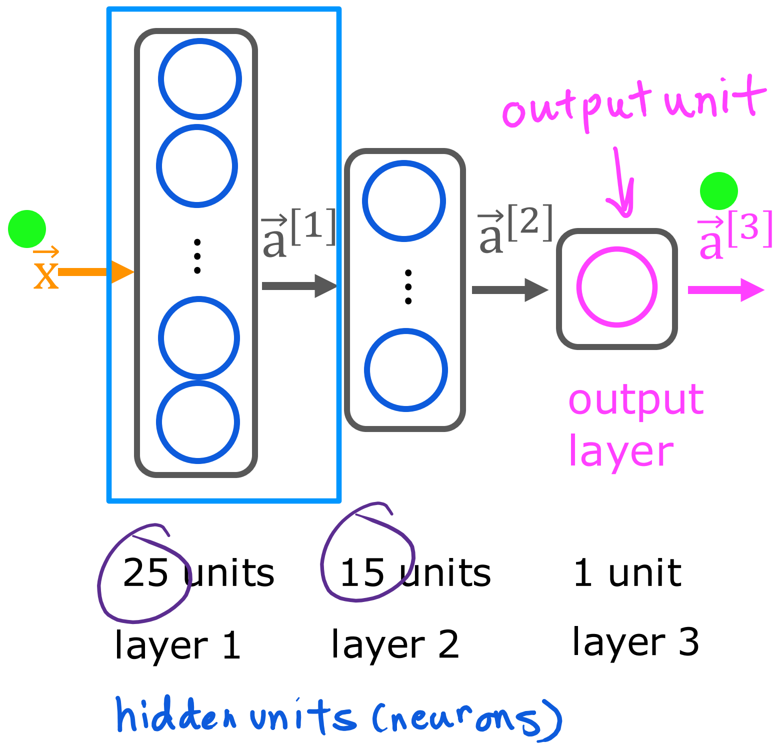
Stanford ONLINE



Neural Network Model

Inference: making predictions (forward propagation)

Handwritten digit recognition



Digit images

label 0 1

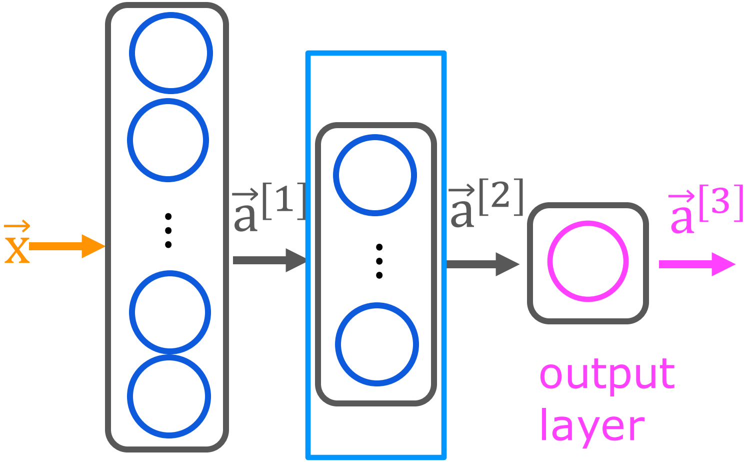
8 × 8

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

$a^{[0]}$

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

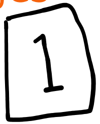
Handwritten digit recognition



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

Digit images

label 0 1



8

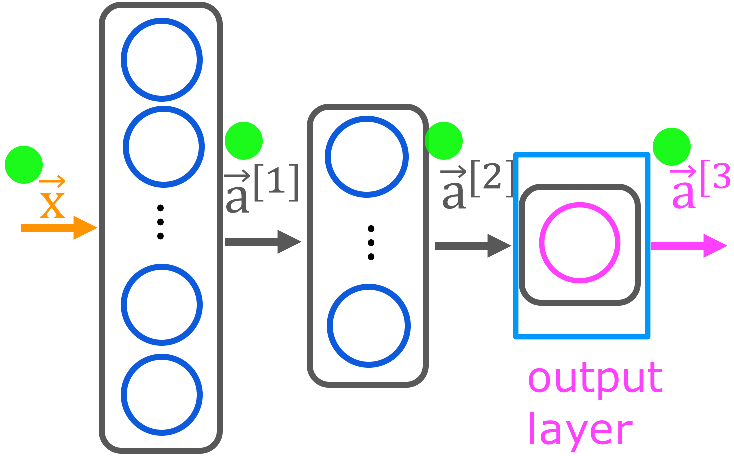
255	255	255	255	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255
255	255	0	0	0	255	255	255
255	255	255	255	255	255	255	255

8

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

Handwritten digit recognition

forward propagation



$\vec{a}^{[3]} = f(x)$ probability of being a handwritten '1'

25 units — layer 1

15 units — layer 2

1 unit — layer 3

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

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

yes

no

- $\hat{y} = 1$

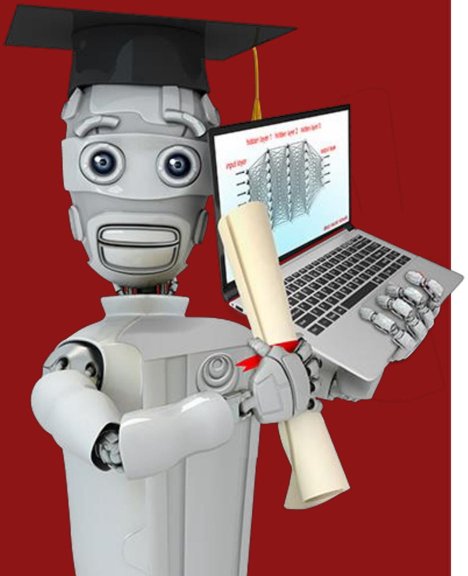
image is digit 1

- $\hat{y} = 0$

image isn't digit 1

DeepLearning.AI

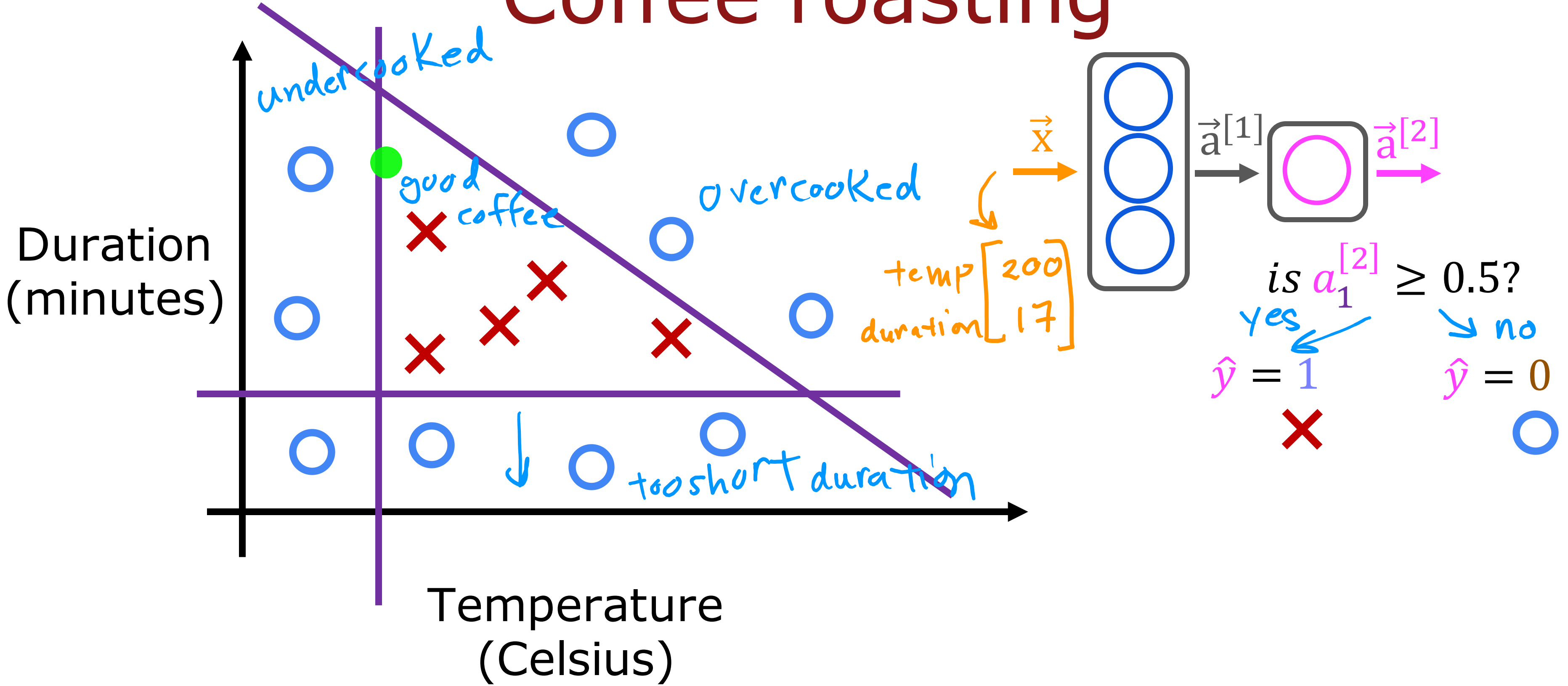
Stanford ONLINE

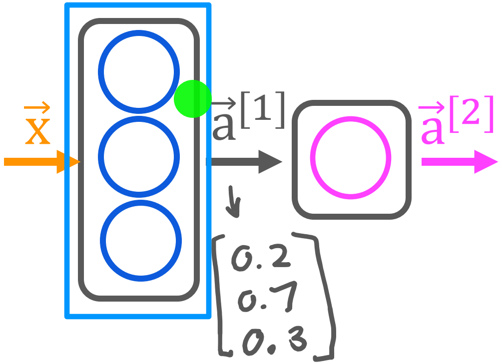


TensorFlow implementation

Inference in Code

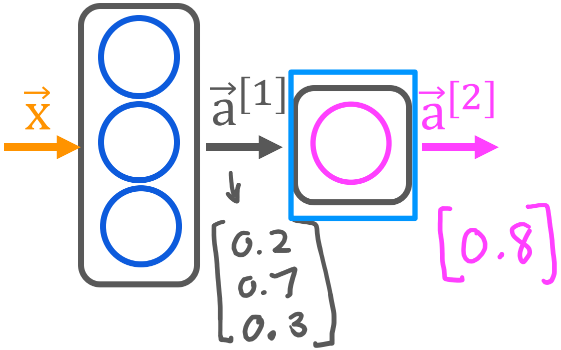
Coffee roasting






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

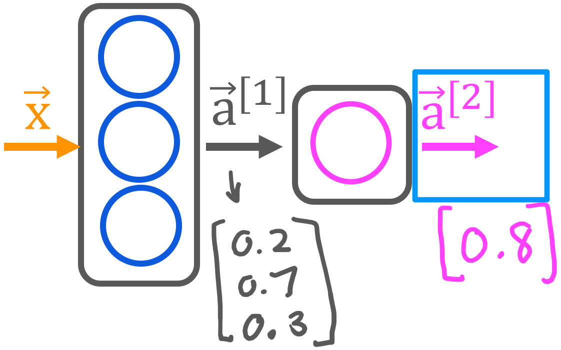
Build the model using TensorFlow




```
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



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

yes

no

$\hat{y} = 1$

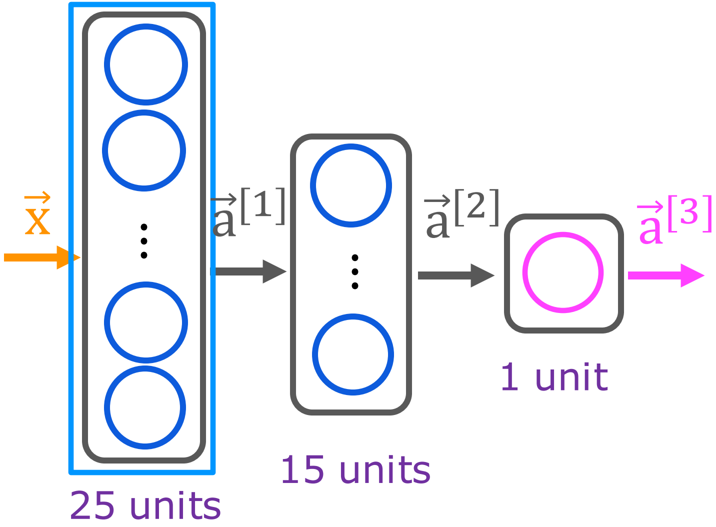
$\hat{y} = 0$

```
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
```

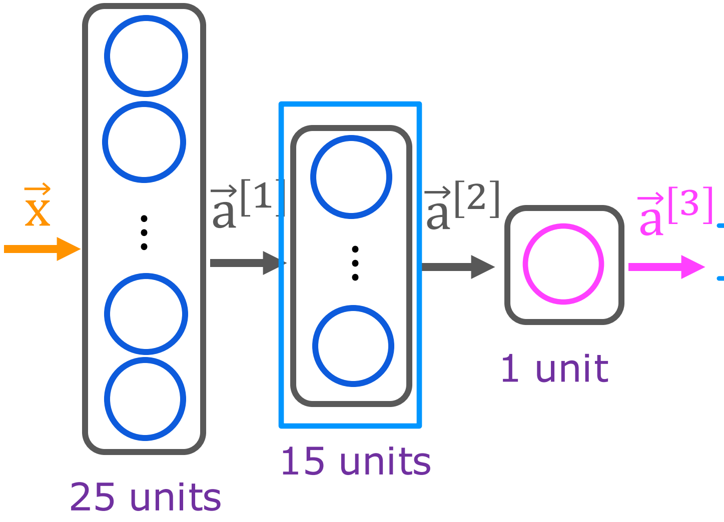
```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

Model for digit classification




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

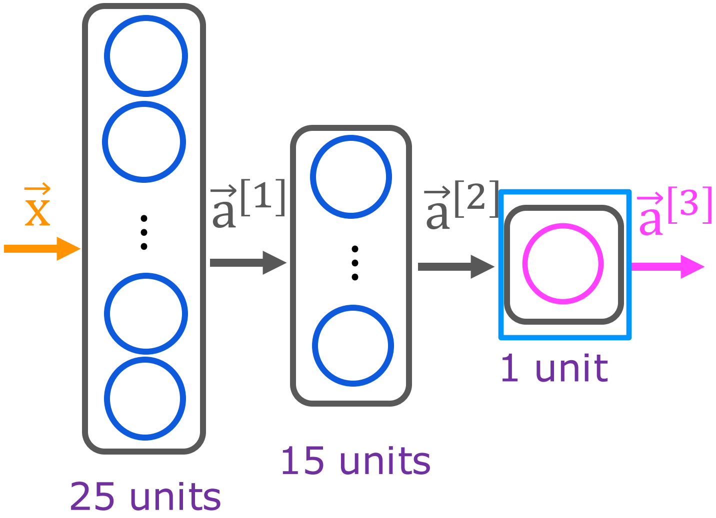
Model for digit classification




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

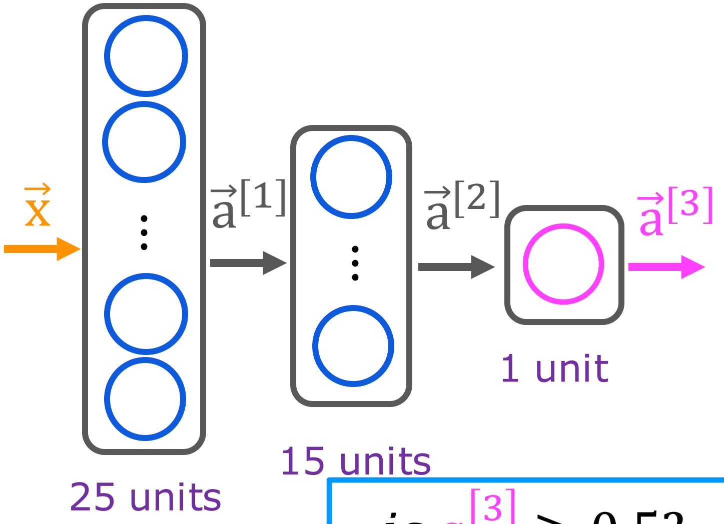
```
layer_2 = Dense(units=15, activation='sigmoid')
a2 = layer_2(a1)
```

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)
```

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)
```

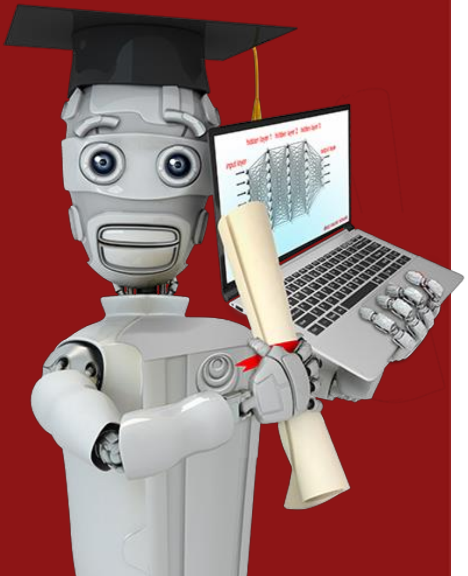
is $a_1^{[3]} \geq 0.5?$

$\hat{y} = 1$ ✕ $\hat{y} = 0$ ○

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

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 x 3 matrix

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

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

2D array

2 x 3

$$\begin{array}{cc} \text{4 rows} & \begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix} \\ & \text{2 columns} \end{array}$$

4 x 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,]]
```

4 x 2

1 x 2

2 x 1

Note about numpy arrays

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

$\rightarrow \begin{bmatrix} 200 & 17 \end{bmatrix}$ 1 x 2

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

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

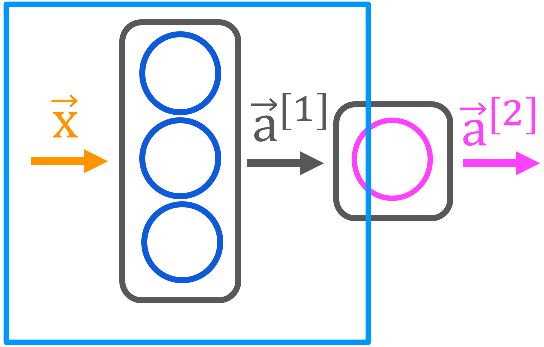
```
x = np.array([[200.0, 17.0]])
```

[[200.0, 17.0]]

1 x 2

[200.0 17.0]

Activation vector




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

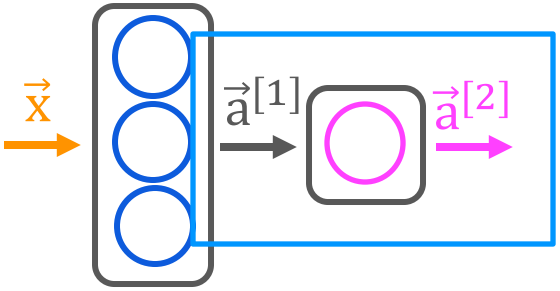
[[0.2, 0.7, 0.3]] 1 x 3 matrix

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

```
a1.numpy()
```

```
array([[1.4661001, 1.125196 , 3.2159438]], dtype=float32)
```

Activation vector




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

[[0.8]]

1 x 1

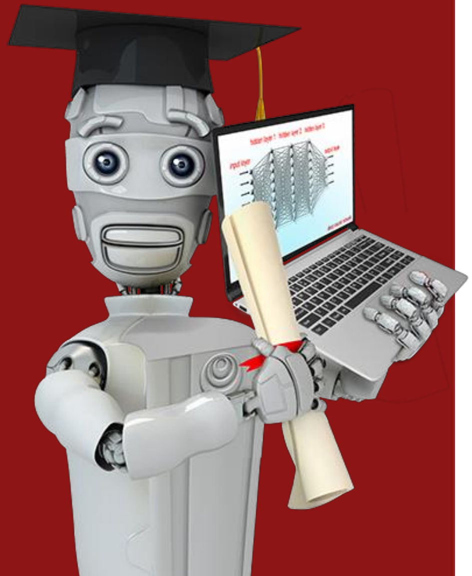
```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
a2.numpy()
```

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

DeepLearning.AI

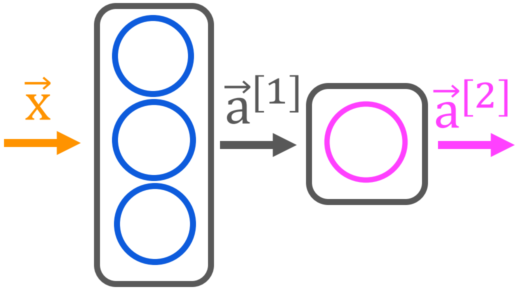
Stanford ONLINE



TensorFlow implementation

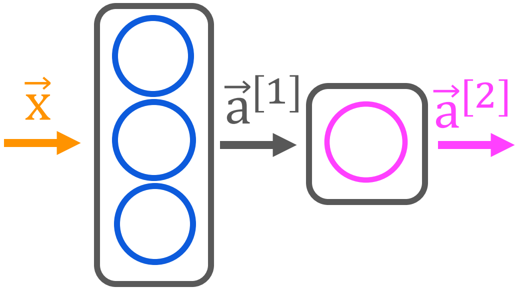
Building a neural network

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



layer1

```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
```

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]])
```

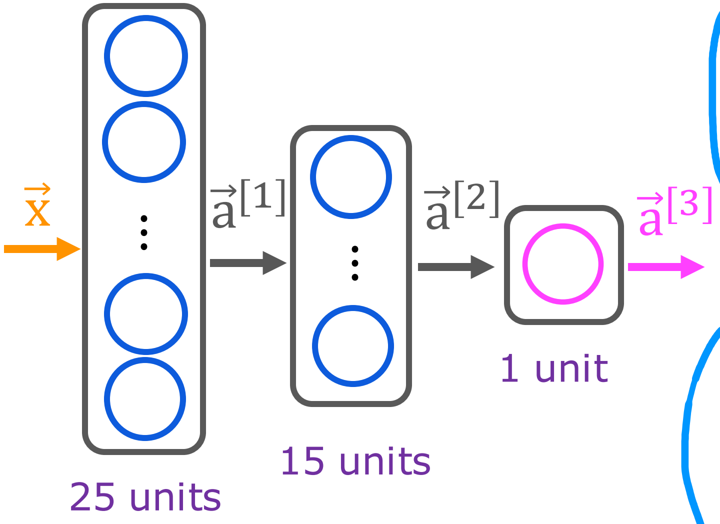
4 x 2

targets
```
y = np.array([1,0,0,1])
```

```
model.compile(...)
```
more about this next week!

```
model.fit(x,y)

model.predict(x_new)
```

Digit classification model

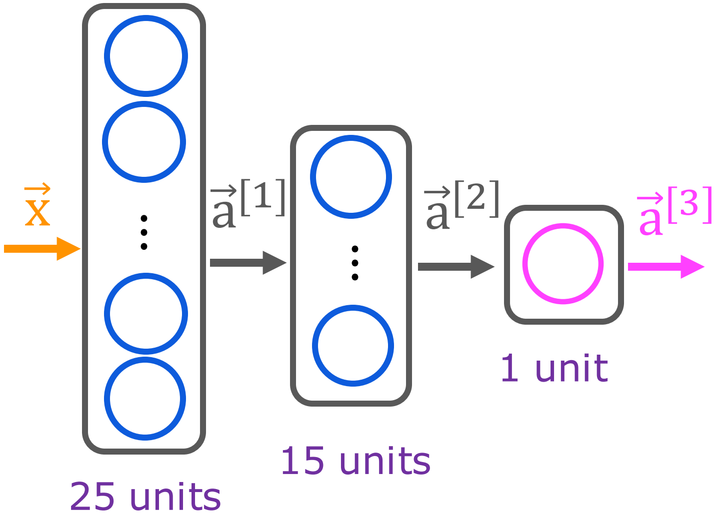



```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3]
model.compile(...)

x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])

model.fit(x,y)
```

← more about this next week!

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

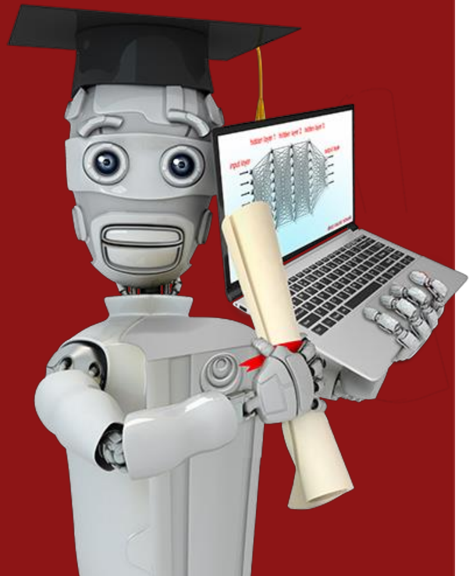
Digit classification model




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

model.compile(...)

x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])

model.fit(x,y)

model.predict(x_new)
```

DeepLearning.AI

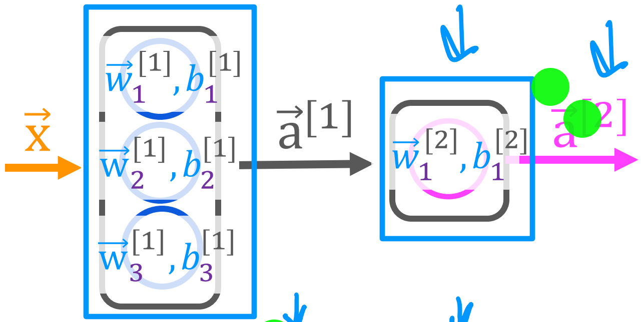
Stanford ONLINE



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])
b2_1 = np.array([3])
z2_1 = np.dot(w2_1,a1)+b2_1
a2_1 = sigmoid(z1_1)
```

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

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

1D arrays

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

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

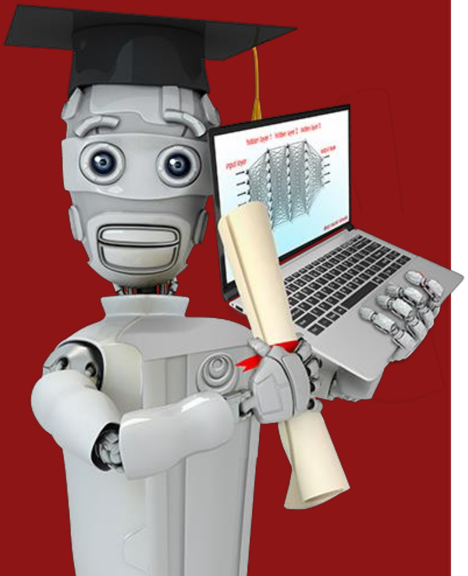
```
w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b
a1_2 = sigmoid(z1_2)
```

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

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

DeepLearning.AI

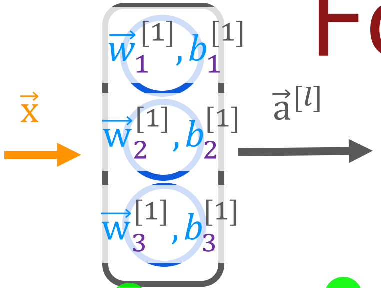
Stanford ONLINE



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, 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
```

3

[0,0,0]

0,1,2

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

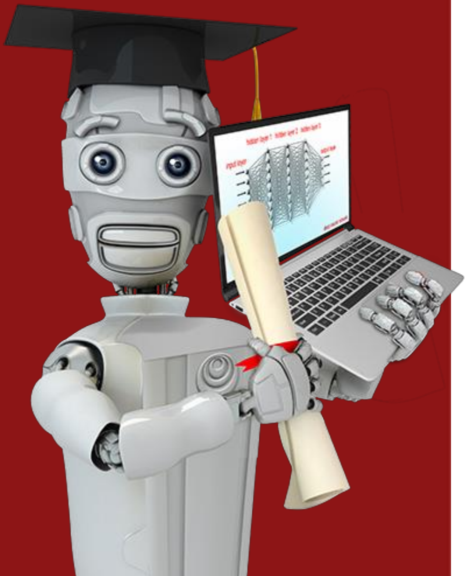
$a^{[1]}$

$a^{[2]}$

capital W refers to a matrix

DeepLearning.AI

Stanford
ONLINE



Speculations on artificial general intelligence (AGI)

Is there a path to AGI?

AI

ANI

(artificial narrow intelligence)

E.g., smart speaker,
self-driving car, web search,
AI in farming and factories

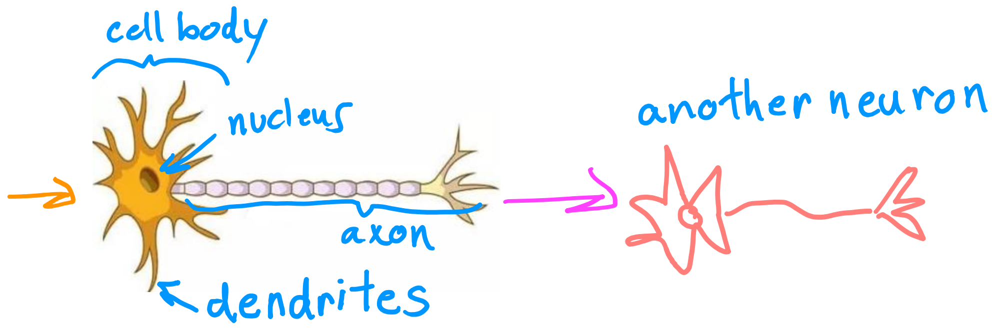
AGI

(artificial general intelligence)

Do anything a human can do

Biological neuron

inputs outputs



Simplified mathematical model of a neuron

inputs outputs

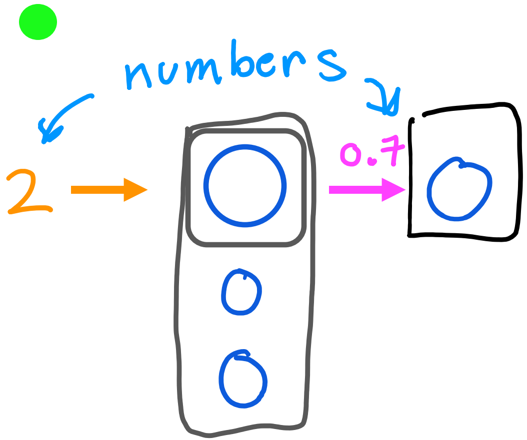
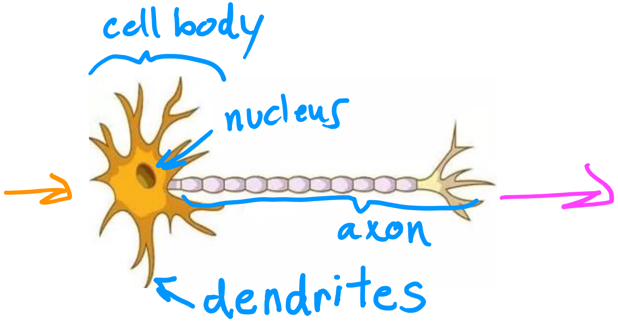


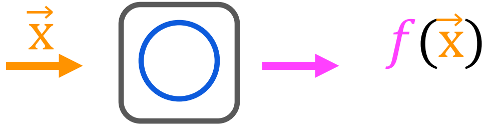
image source: https://biologydictionary.net/sensory-neuron/

Neural network and the brain

Can we mimic the human brain?

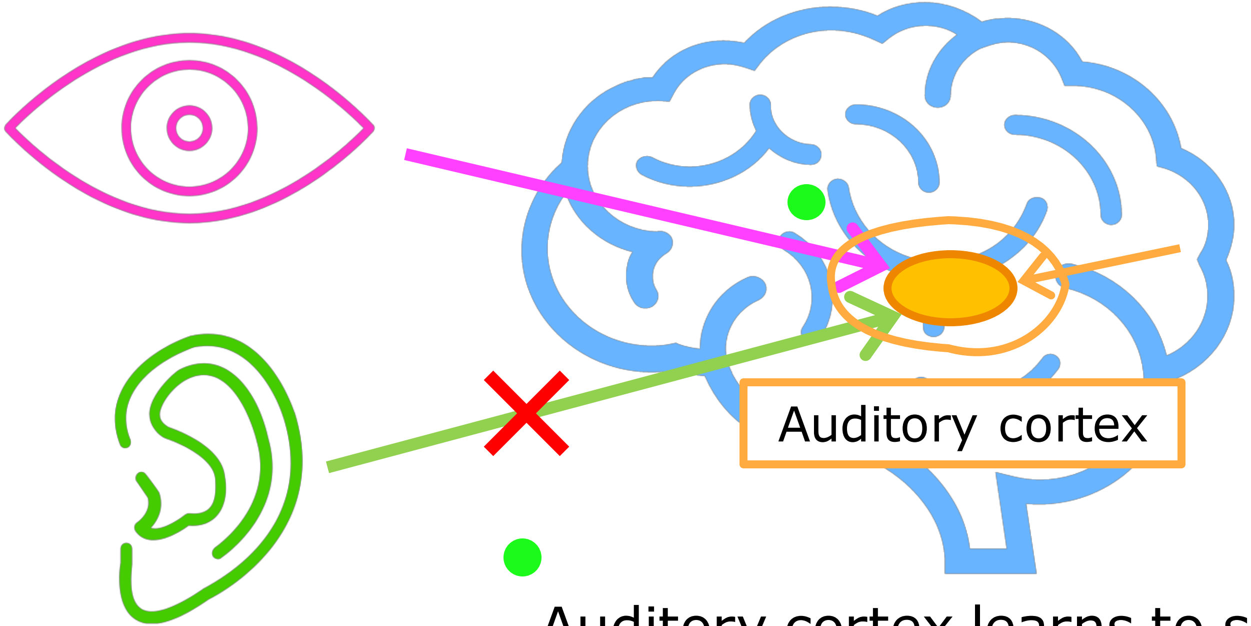


vs



We have (almost) no idea how the brain works

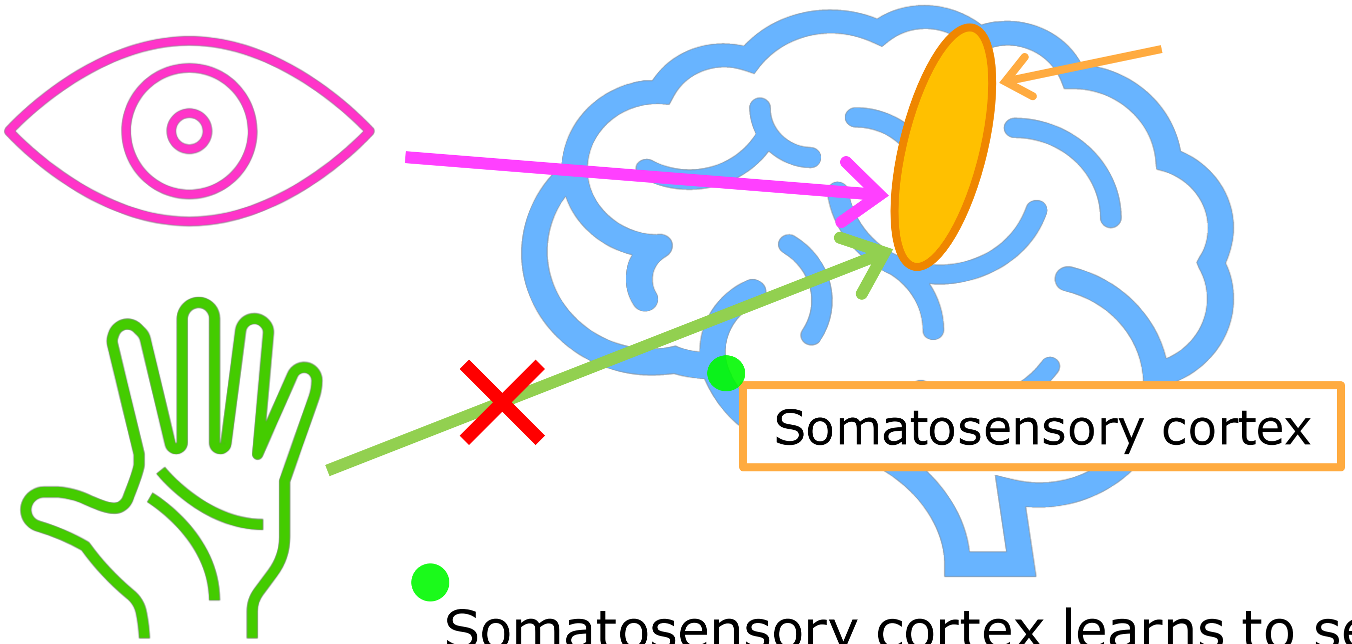
The "one learning algorithm" hypothesis



Auditory cortex learns to see

[Roe et al., 1992]

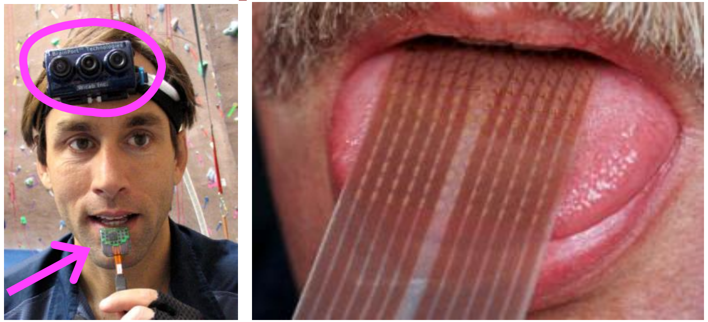
The ”one learning algorithm” hypothesis



Somatosensory cortex learns to see

[Metin & Frost, 1989]

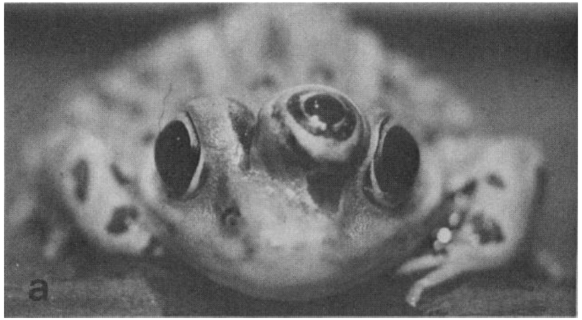
Sensor representations in the brain



Seeing with your tongue

Human echolocation (sonar)

Haptic belt: Direction sense

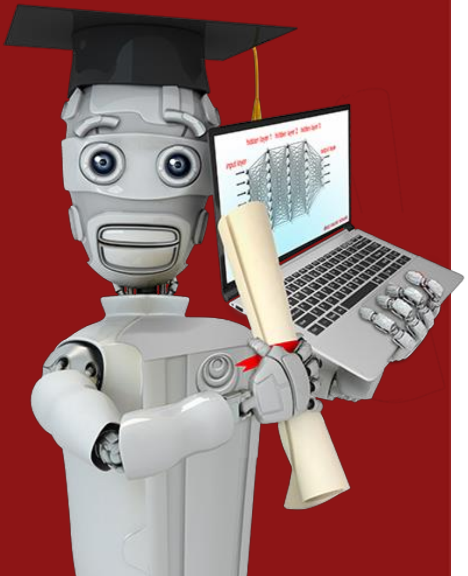


Implanting a 3rd eye

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

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):
   a_out = np.zeros(units)
   for j in range(units):
      w = W[:,j]
      z = np.dot(w,x) + b[j]
      a[j] = g(z)
  return a

  [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

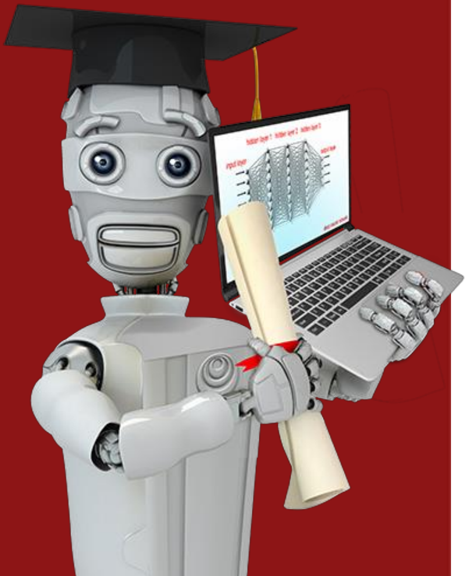
1x3 2D array

all 2D arrays

matrix multiplication

DeepLearning.AI

Stanford
ONLINE



Vectorization (optional)

Matrix multiplication

Dot products

example

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

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

$$3 + 8$$

$$11$$

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

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

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

useful for understanding matrix multiplication

Vector matrix multiplication

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

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

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

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

1 by 2

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

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

3 + 8

11

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

5 + 12

17

$\mathbf{Z} = [11 \quad 17]$

matrix matrix multiplication

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

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

rows columns

row 1 col 1 row 1 col 2

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

row 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

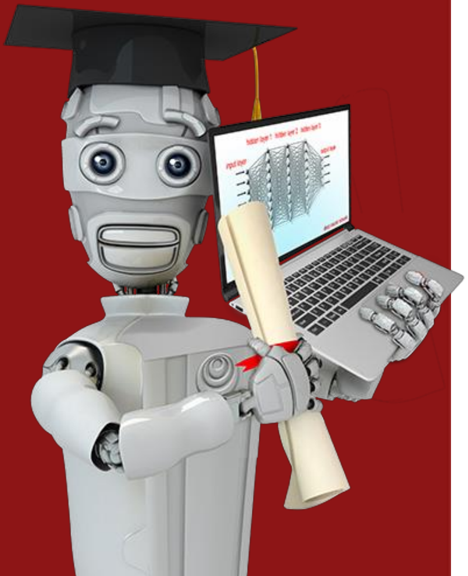
$$= \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix}$$

general rules for matrix multiplication

↳ next video!

DeepLearning.AI

Stanford
ONLINE



Vectorization (optional)

Matrix multiplication rules

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \qquad A^T = \begin{array}{cc} \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} & \begin{array}{c} \vec{a}_1^{\,T} \\ \vec{a}_2^{\,T} \\ \vec{a}_3^{\,T} \end{array} \end{array} \qquad W = \begin{array}{c} \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \\ \begin{array}{cccc} \vec{w}_1 & \vec{w}_2 & \vec{w}_3 & \vec{w}_4 \end{array} \end{array}$$

$$Z = A^T W =$$

Matrix multiplication rules

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

3 by 4 matrix

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

row 3 column 2

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

$0.5 \quad + \quad 1.2$

row 2 column 3?

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

$-7 \quad + \quad -16$

Matrix multiplication rules

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

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?

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

$-7 + -16$

Matrix multiplication rules

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

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

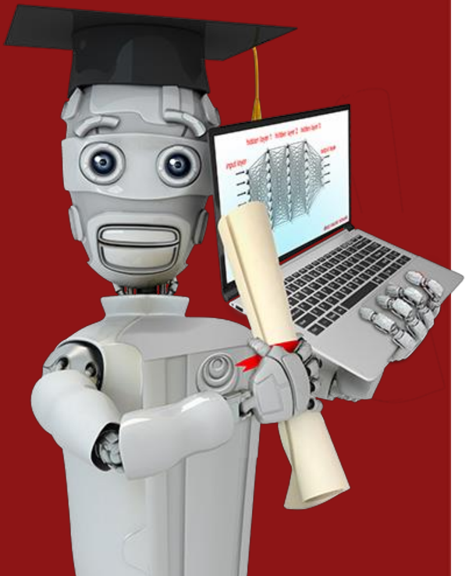
3 by 4 matrix

↳ same # rows as A^T

same # columns as W

DeepLearning.AI

Stanford ONLINE



Vectorization (optional)

Matrix multiplication code

Matrix multiplication in NumPy

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

```
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)
```

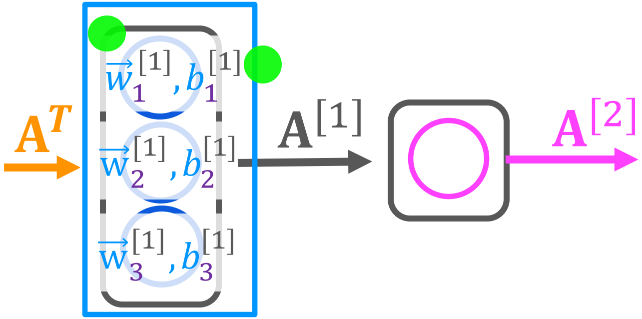
or

```
Z = AT @ W
```

result

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

Dense layer vectorized



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

1×2

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

2×3

$\vec{b} = [-1 \quad 1 \quad 2]$

1×3

$Z = A^T W + B$

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

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

$A = g(Z)$

$[1 \quad 0 \quad 1]$

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

a_in

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