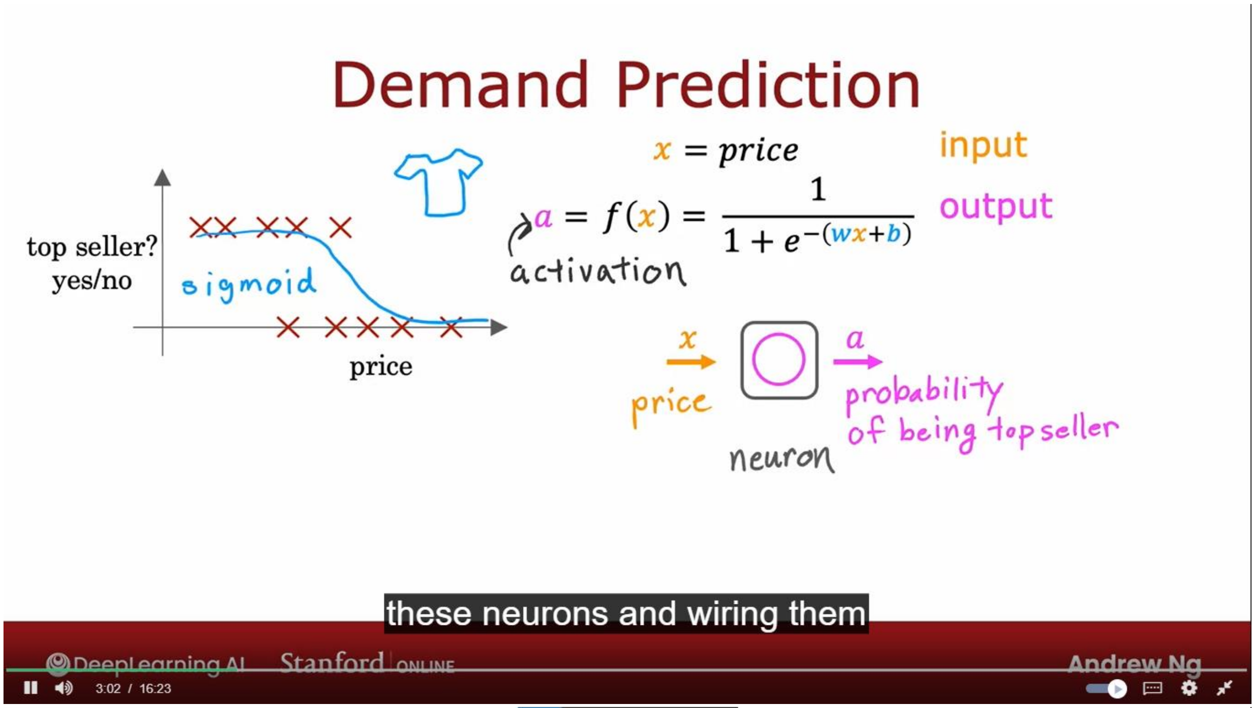
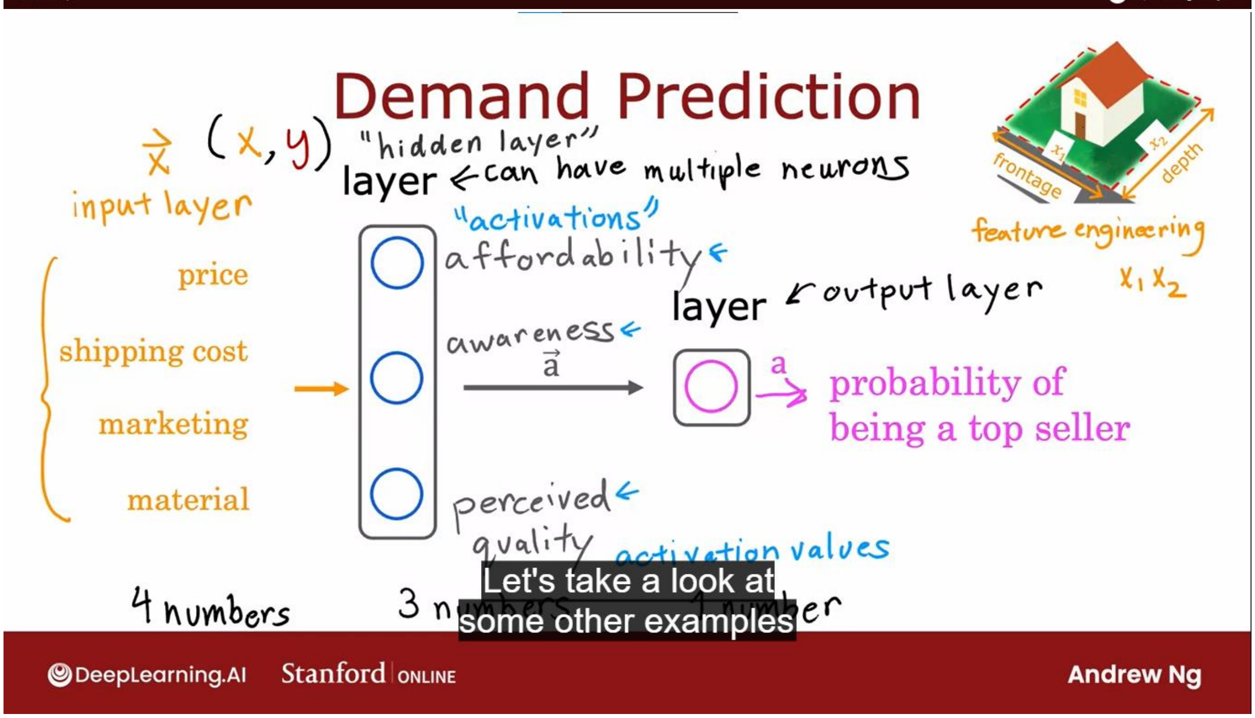
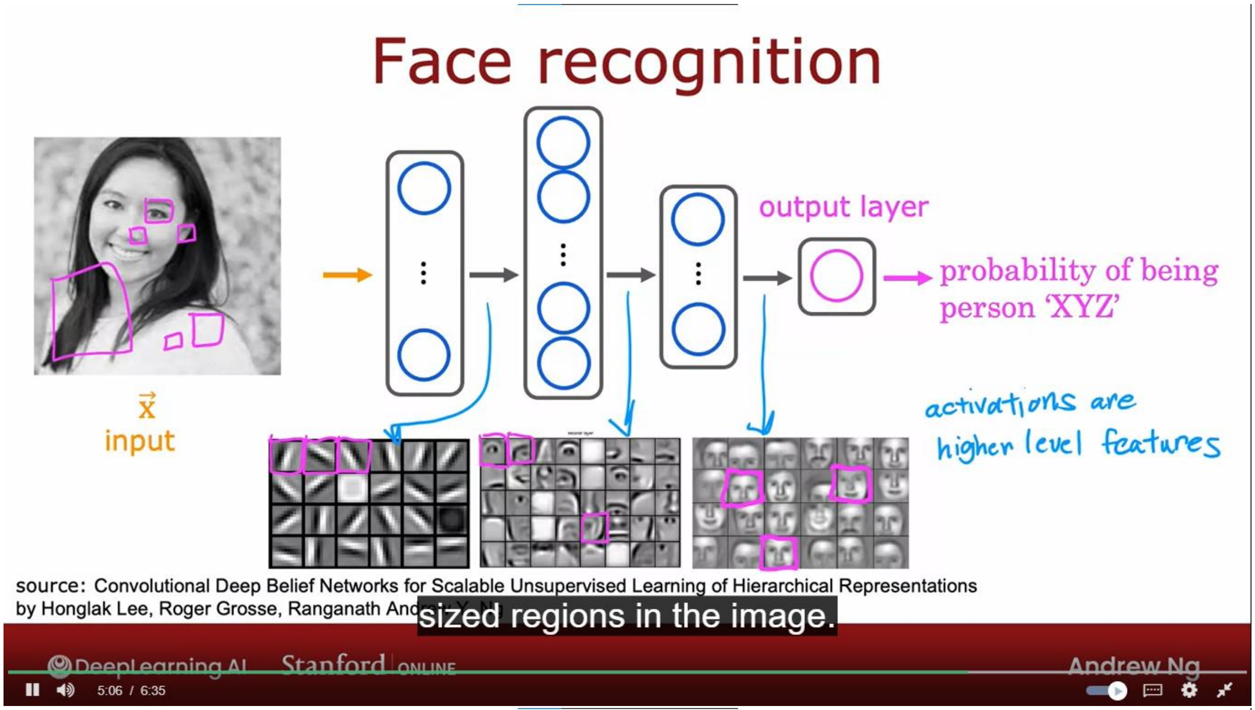
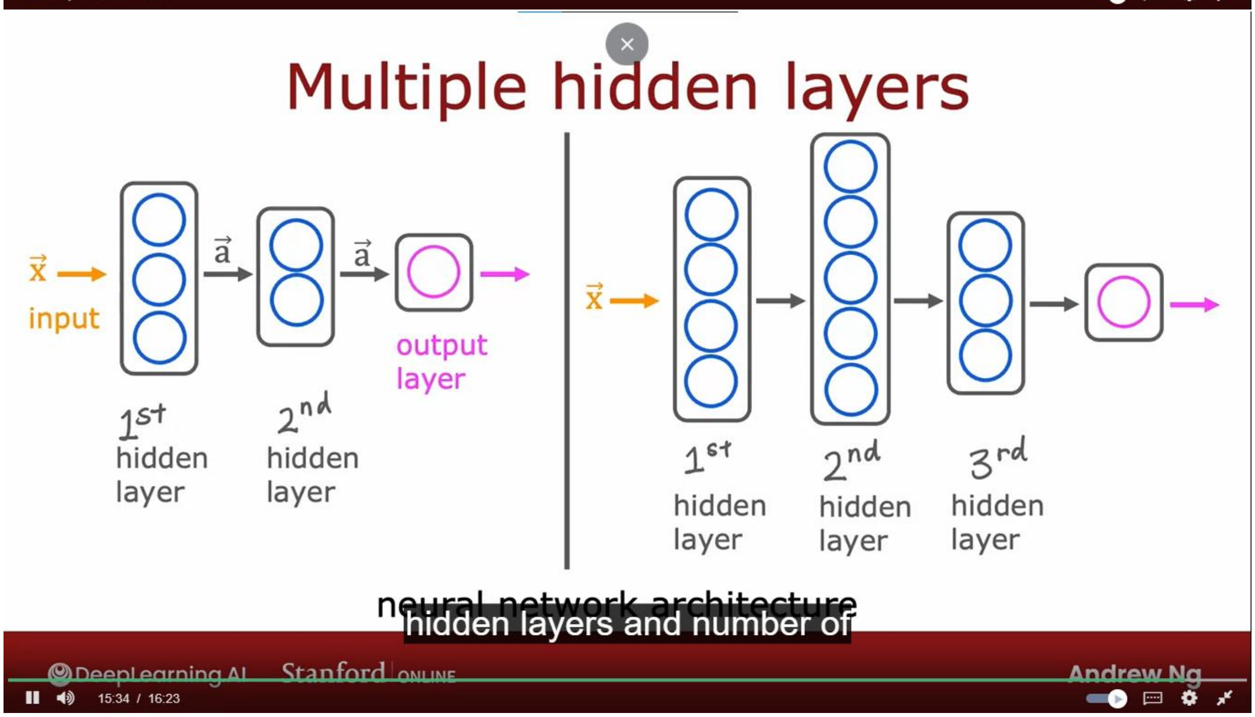
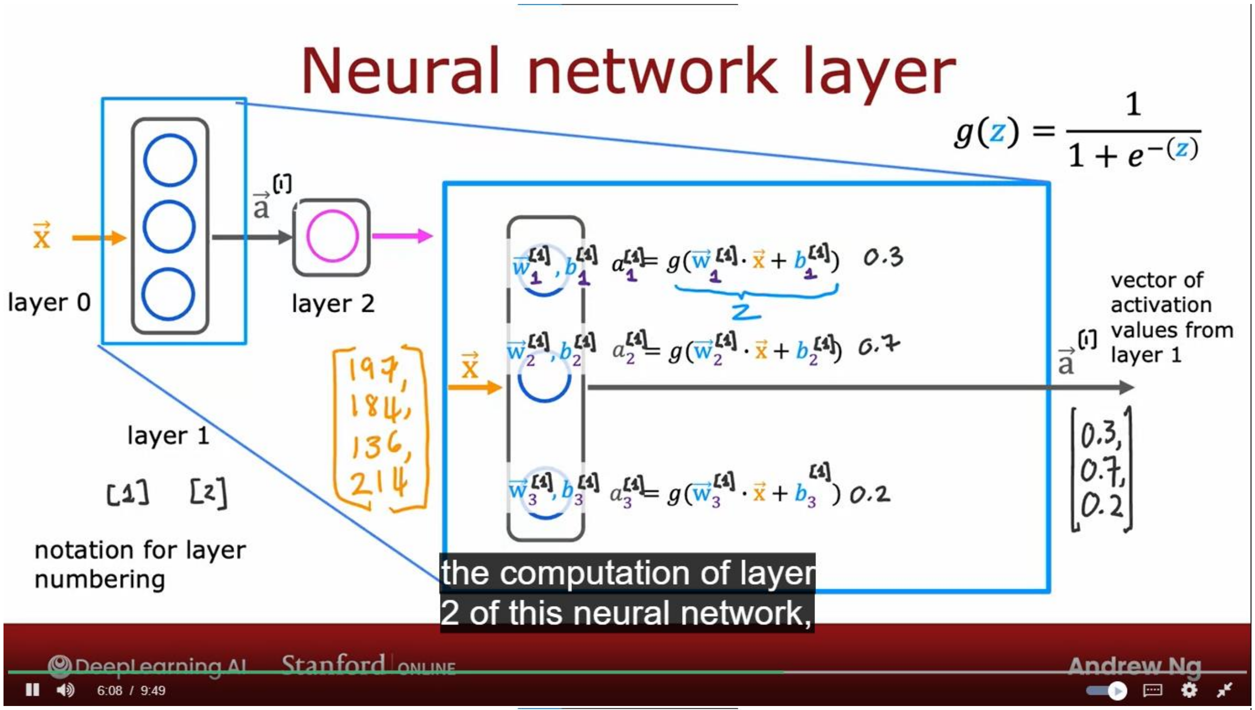
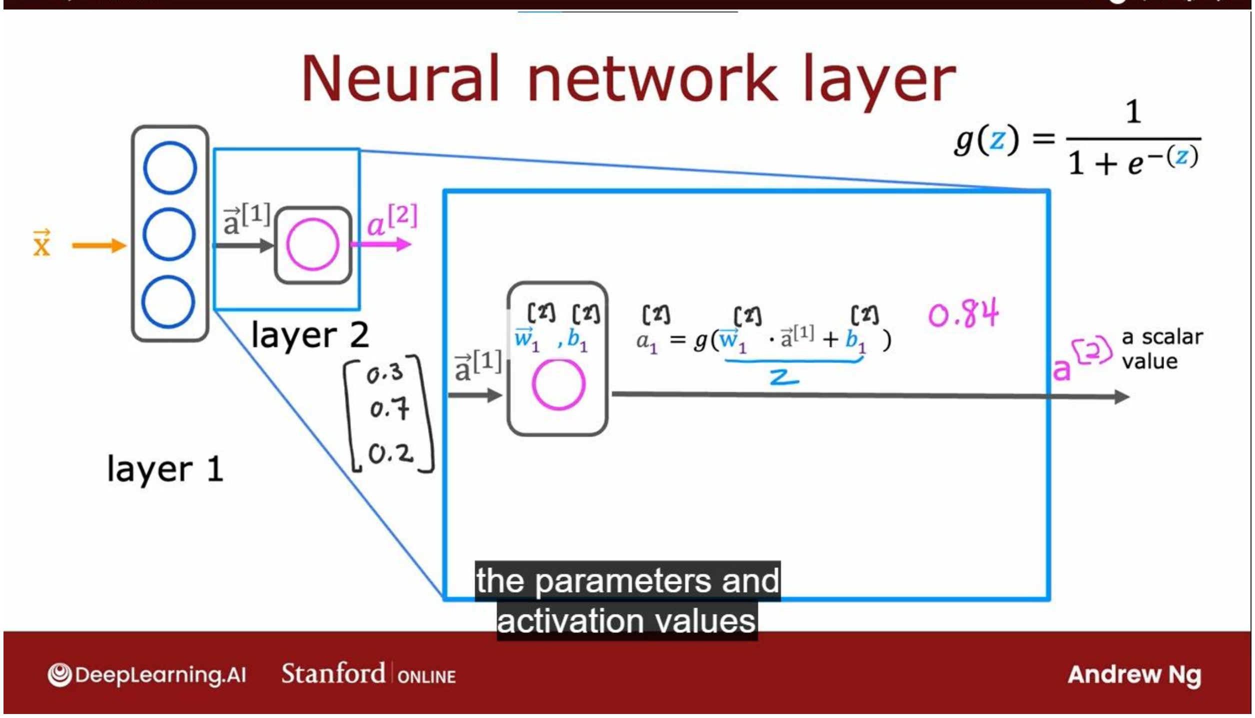

Demand Prediction
x = price
input
a = f(x) = 1 / (1 + e^-(wx+b))
output
activation
top seller?
yes/no
sigmoid
price
x
price
a
probability
of being top seller
neuron
these neurons and wiring them
DeepLearning.AI
Stanford ONLINE
Andrew Ng
3:02 / 16:23

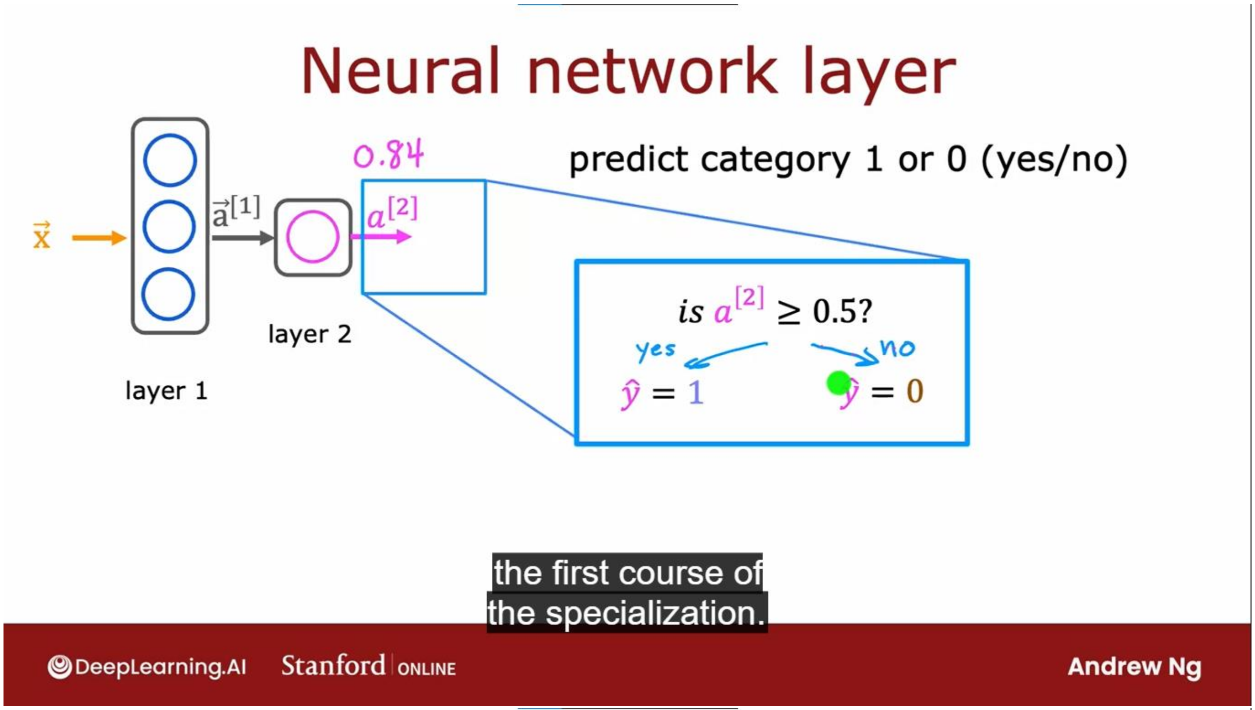

Demand Prediction
x (x, y)
"hidden layer"
layer ← can have multiple neurons
input layer
"activations"
price
shipping cost
marketing
material
affordability
awareness
a
perceived quality
activation values
layer ← output layer
a
probability of
being a top seller
frontage
depth
x1
x2
feature engineering
x1 x2
4 numbers
3 numbers
1 number
Let's take a look at
some other examples
DeepLearning.AI
Stanford ONLINE
Andrew Ng

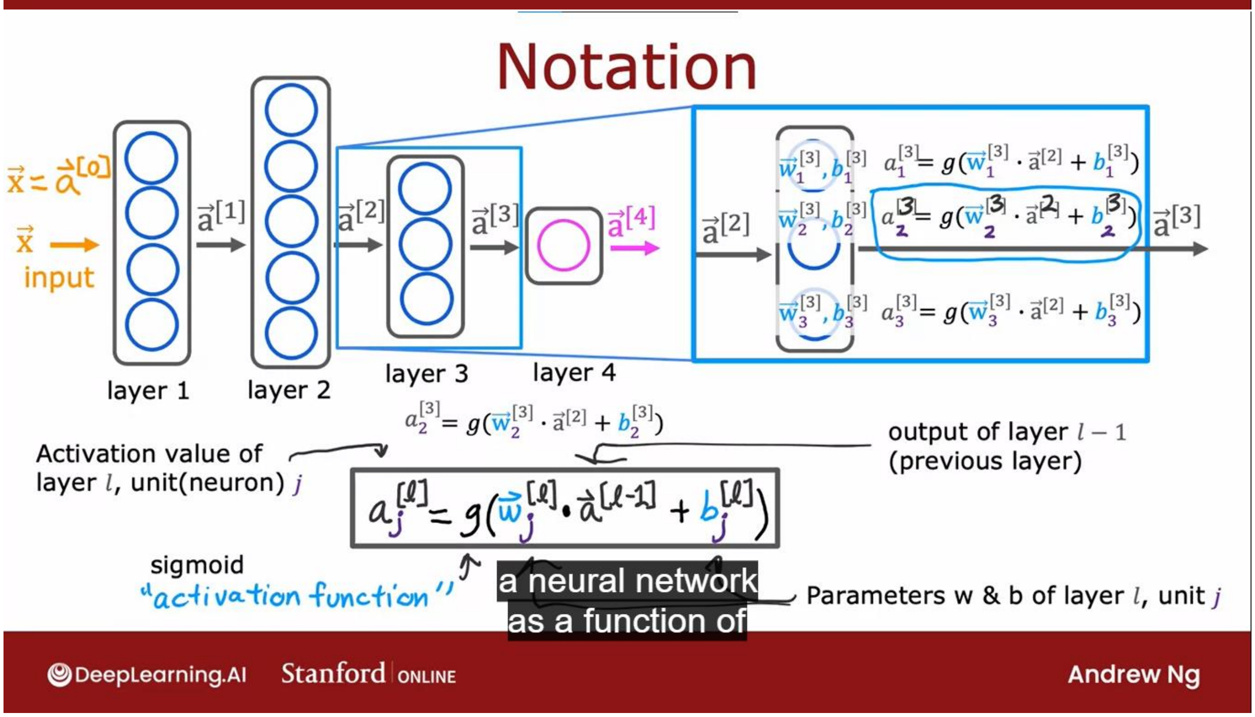

Face recognition
output layer
probability of being person 'XYZ'
activations are higher level features
$\vec{x}$
input
source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations
by Honglak Lee, Roger Grosse, Ranganath And
sized regions in the image.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:06 / 6:35

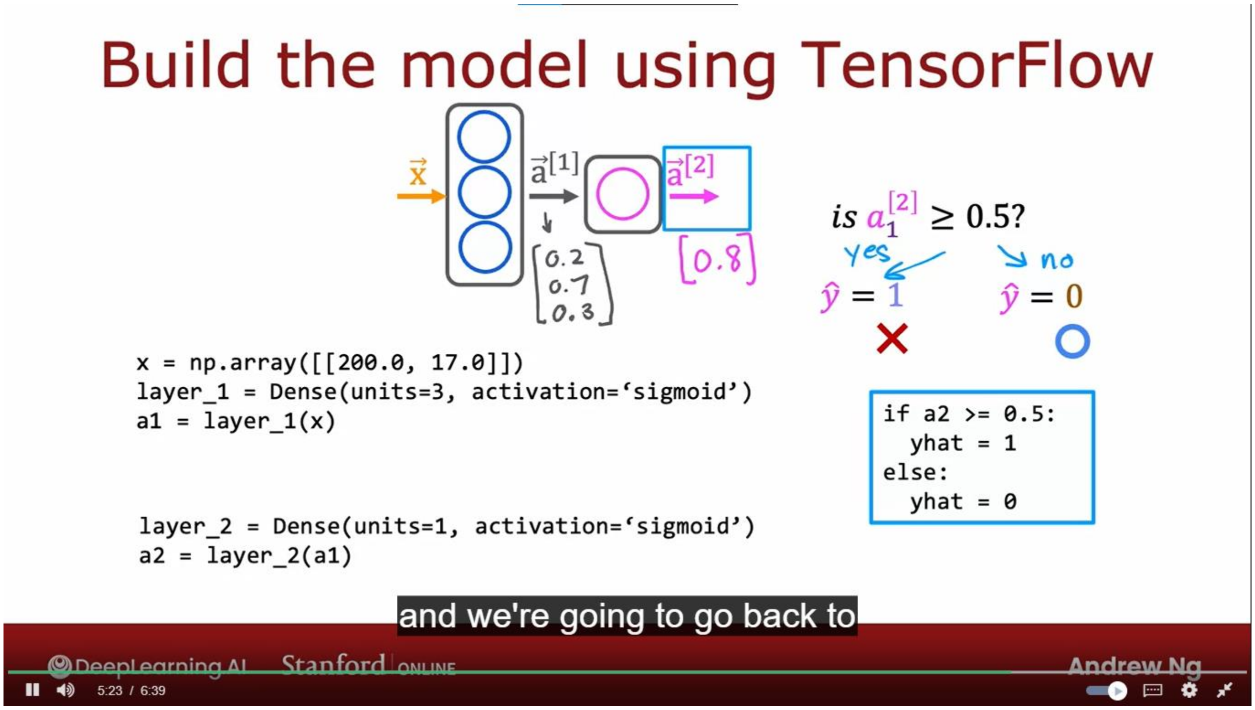

Multiple hidden layers
$\vec{x}$
input
$\vec{a}$
$\vec{a}$
output layer
1st hidden layer
2nd hidden layer
$\vec{x}$
1st hidden layer
2nd hidden layer
3rd hidden layer
neural network architecture
hidden layers and number of
DeepLearning.AI
Stanford ONLINE
Andrew Ng
15:34 / 16:23

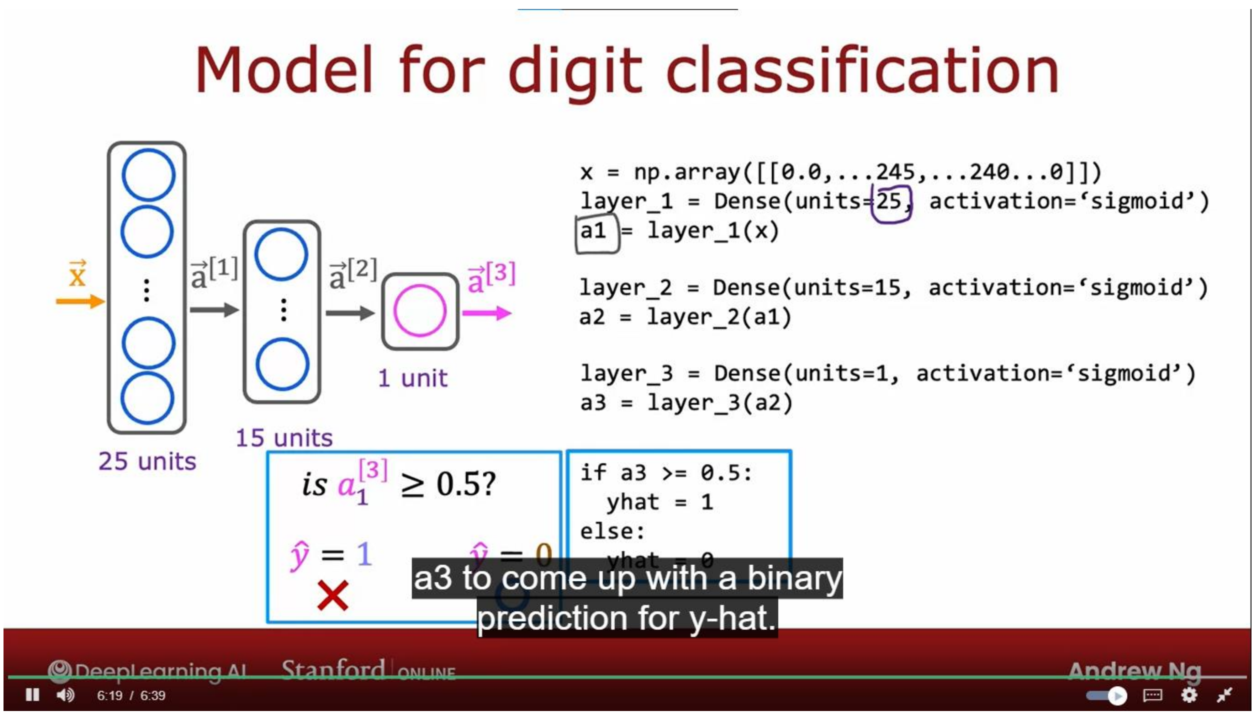

Neural network layer
$g(z) = \frac{1}{1+e^{-(z)}}$
layer 0
layer 1
layer 2
[1] [2]
notation for layer numbering
197, 184, 136, 214
$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$ 0.3
$a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$ 0.7
$a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$ 0.2
vector of activation values from layer 1
0.3, 0.7, 0.2
the computation of layer 2 of this neural network,
DeepLearning.AI
Stanford ONLINE
Andrew Ng
6:08 / 9:49

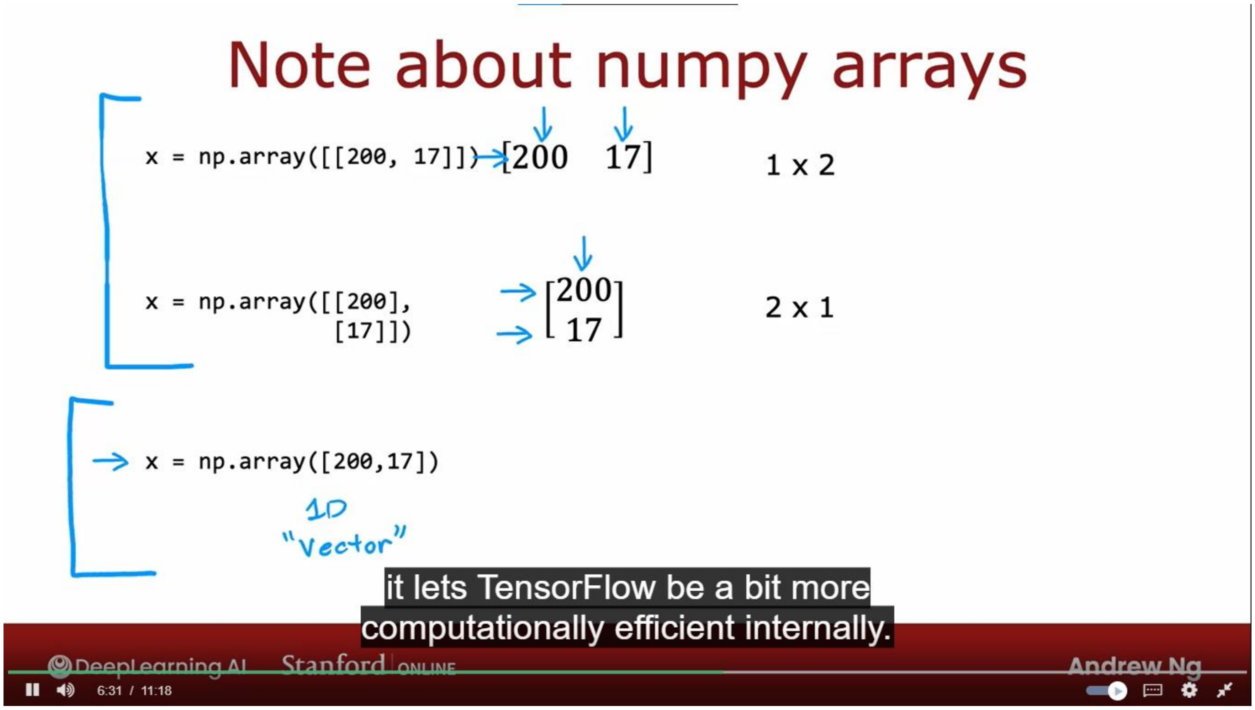

Neural network layer
$g(z) = \frac{1}{1+e^{-(z)}}$
layer 1
layer 2
$a^{[2]}$
0.3 0.7 0.2
$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$ 0.84
a scalar value
the parameters and activation values
DeepLearning.AI
Stanford ONLINE
Andrew Ng

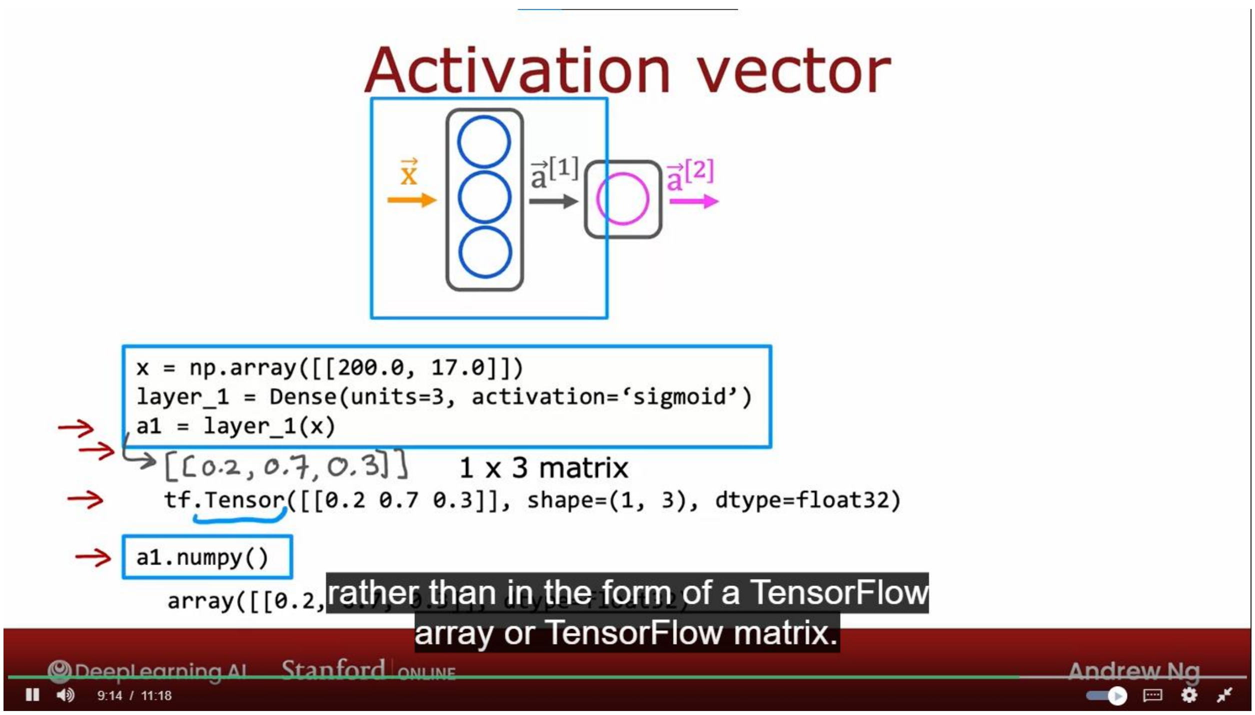

Neural network layer
predict category 1 or 0 (yes/no)
0.84
$\vec{x}$
$\vec{a}^{[1]}$
$a^{[2]}$
layer 1
layer 2
is $a^{[2]} \geq 0.5$?
yes
no
$\hat{y} = 1$
$\hat{y} = 0$
the first course of
the specialization.
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Notation
$\vec{x} = \vec{a}^{[0]}$
$\vec{x}$
input
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
$\vec{a}^{[4]}$
layer 1
layer 2
layer 3
layer 4
$\vec{w}_1^{[3]}, b_1^{[3]}$ $a_1^{[3]} = g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$
$\vec{w}_2^{[3]}, b_2^{[3]}$ $a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
$\vec{w}_3^{[3]}, b_3^{[3]}$ $a_3^{[3]} = g(\vec{w}_3^{[3]} \cdot \vec{a}^{[2]} + b_3^{[3]})$
$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$
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
a neural network
as a function of
DeepLearning.AI
Stanford ONLINE
Andrew Ng

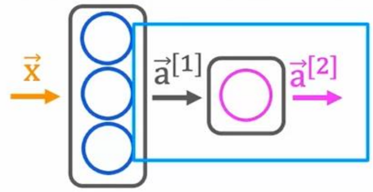

Build the model using TensorFlow
x⃗
a⃗[1]
a⃗[2]
0.2
0.7
0.3
0.8
is a1[2] ≥ 0.5?
yes
no
ŷ = 1
ŷ = 0
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
if a2 >= 0.5:
yhat = 1
else:
yhat = 0
and we're going to go back to
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:23 / 6:39


Model for digit classification
x⃗
a⃗[1]
a⃗[2]
a⃗[3]
25 units
15 units
1 unit
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 a1[3] ≥ 0.5?
ŷ = 1
ŷ = 0
if a3 >= 0.5:
yhat = 1
else:
a3 to come up with a binary
prediction for y-hat.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
6:19 / 6:39


Note about numpy arrays
x = np.array([[200, 17]])
[200 17]
1 x 2
x = np.array([[200],
[17]])
[200
17]
2 x 1
x = np.array([200,17])
1D
"Vector"
it lets TensorFlow be a bit more
computationally efficient internally.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
6:31 / 11:18


Activation vector
x
a[1]
a[2]
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,
rather than in the form of a TensorFlow
array or TensorFlow matrix.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
9:14 / 11:18

Activation vector




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

[[0.8]]

1 x 1

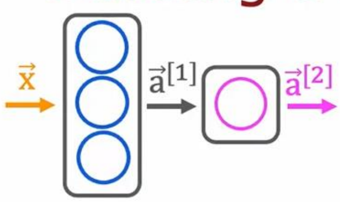
```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
a2.numpy()
```

```
array([[0.8]], dty
```

Once again you can convert
from a tensorflow tensor to

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])
```

layer

		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(...)
```

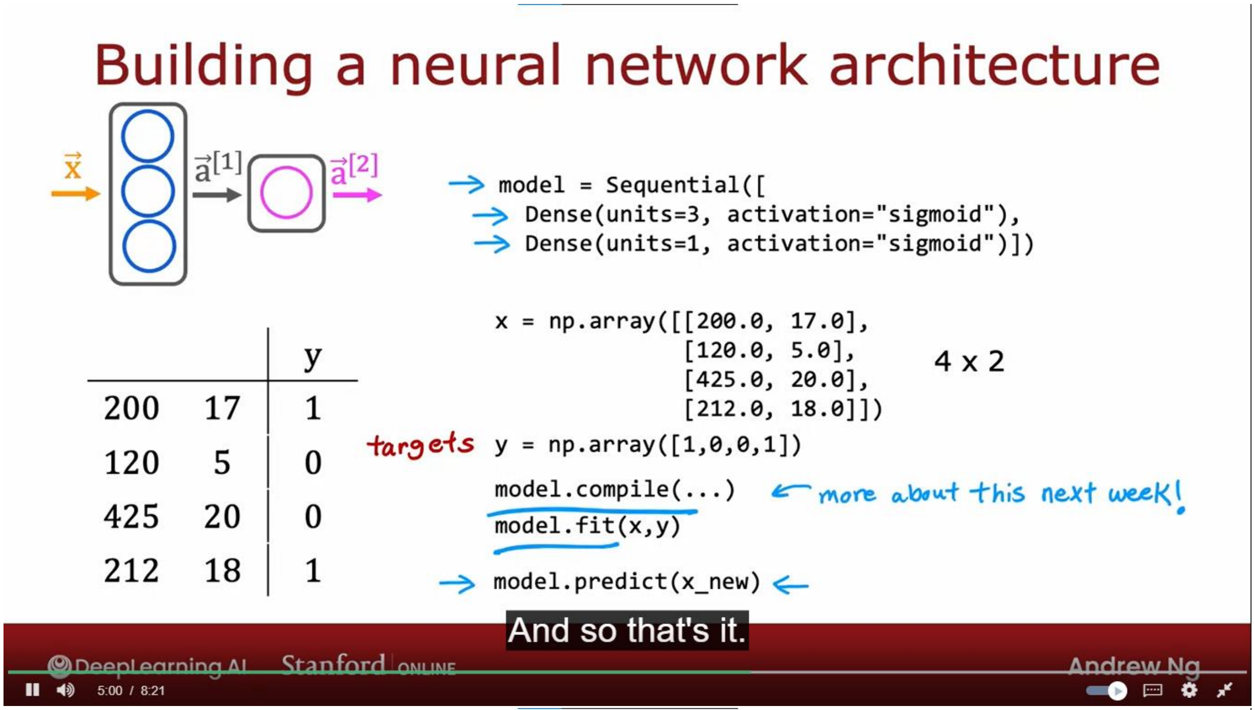
← more about this next week!

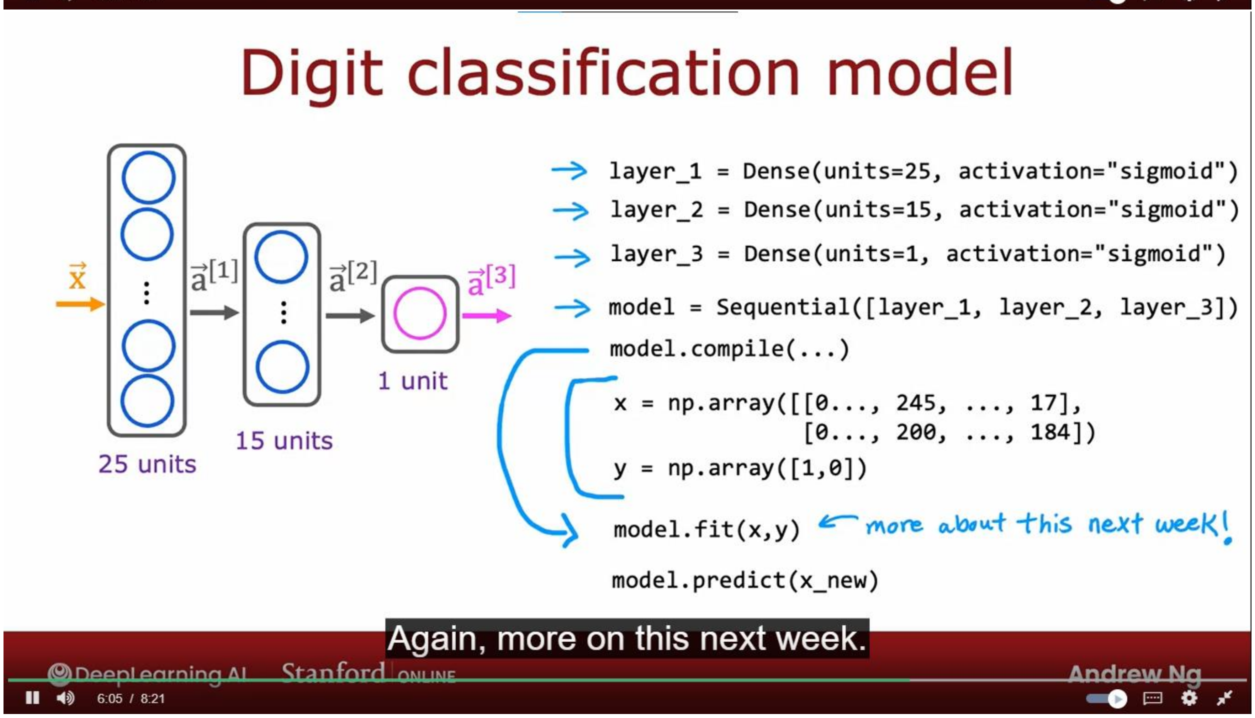
```
model.fit(x,y)

model.predict(x_new)
```

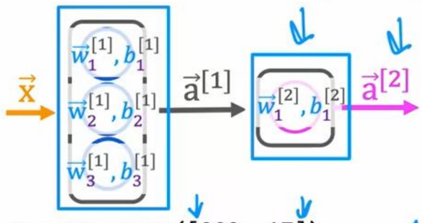
layer one and layer two as follows.

4:23 / 8:21


Building a neural network architecture
x⃗
a⃗[1]
a⃗[2]
model = Sequential([
Dense(units=3, activation="sigmoid"),
Dense(units=1, activation="sigmoid")])
x = np.array([[200.0, 17.0],
[120.0, 5.0],
[425.0, 20.0],
[212.0, 18.0]])
4 x 2
y
200 17 1
120 5 0
425 20 0
212 18 1
targets y = np.array([1,0,0,1])
model.compile(...)
more about this next week!
model.fit(x,y)
model.predict(x_new)
And so that's it.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:00 / 8:21


Digit classification model
x⃗
a⃗[1]
a⃗[2]
a⃗[3]
25 units
15 units
1 unit
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)
Again, more on this next week.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
6:05 / 8:21

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

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

```
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)
```

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

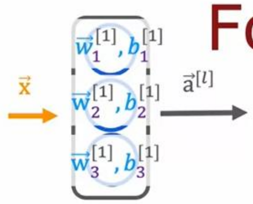
```
w1_3 = np.array([5, -6])
b1_3 = np.array([2])
z1_3 = np.dot(w1_3,x)+b1_3
a1_3 = sigmoid(z1_3)
```

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

DeepLearning.AI Stanford ONLINE Andrew Ng

4:42 / 5:06

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
```

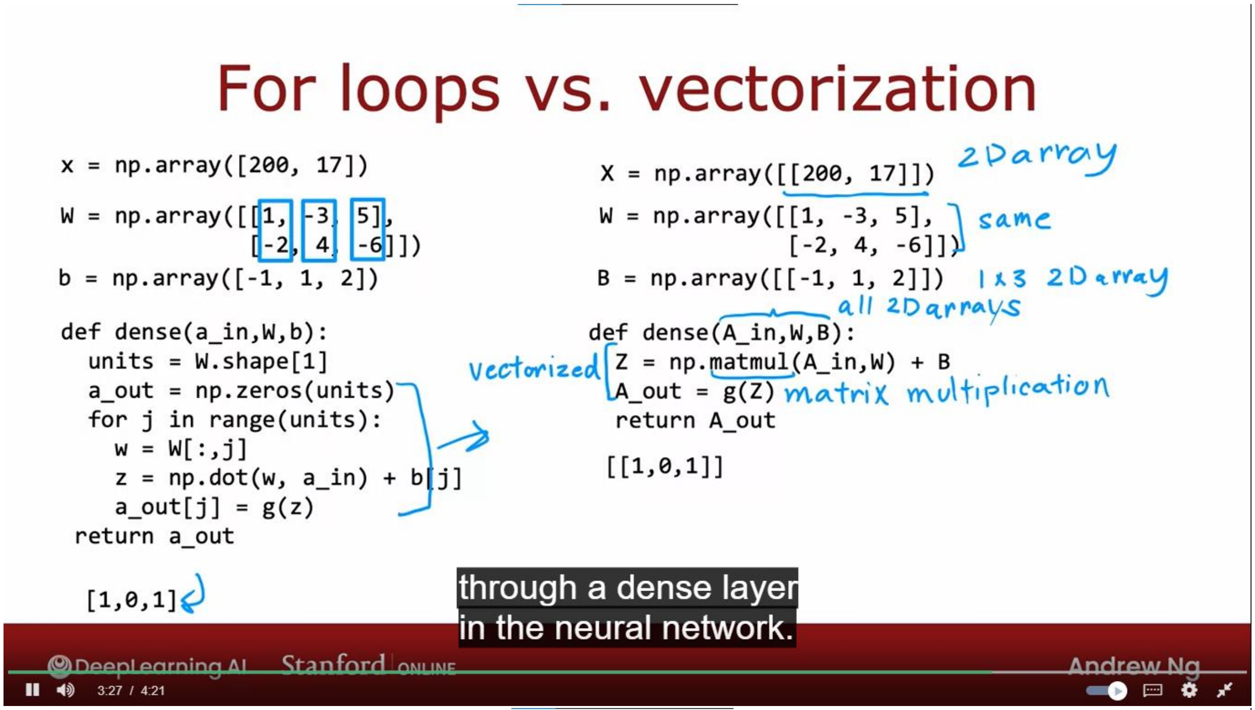
$a^{[1]}$ $a^{[2]}$

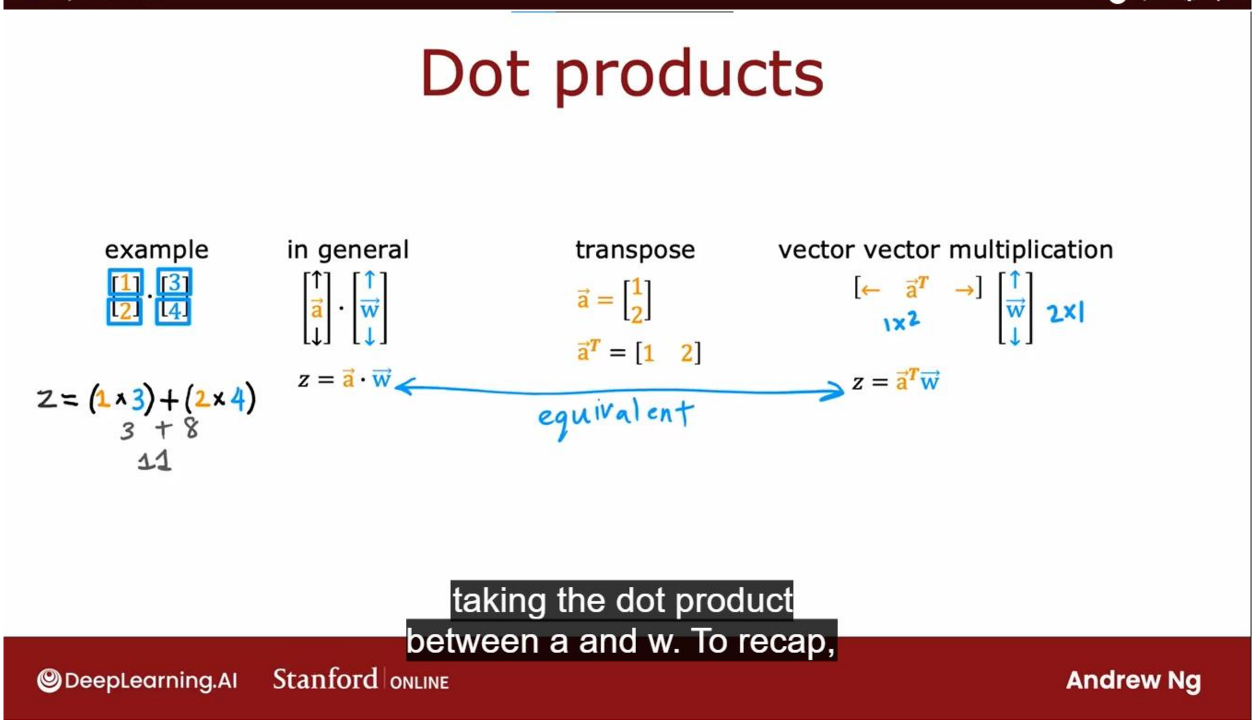
capital W refers to a matrix

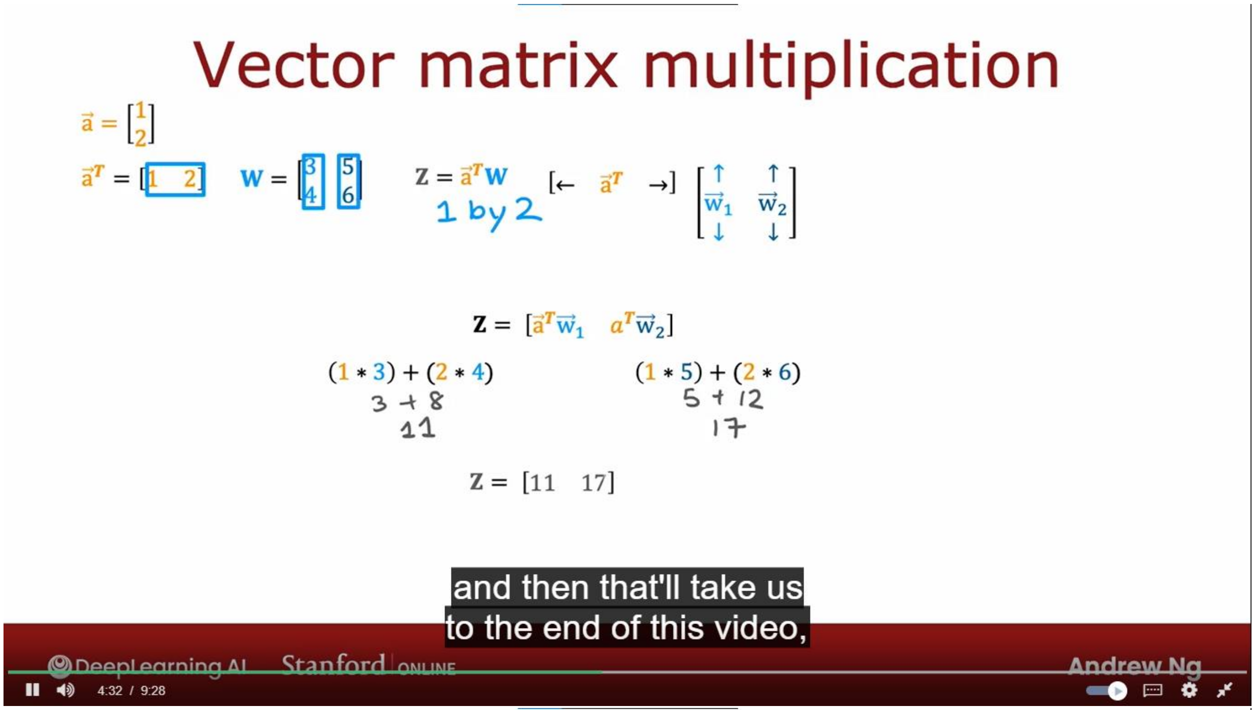
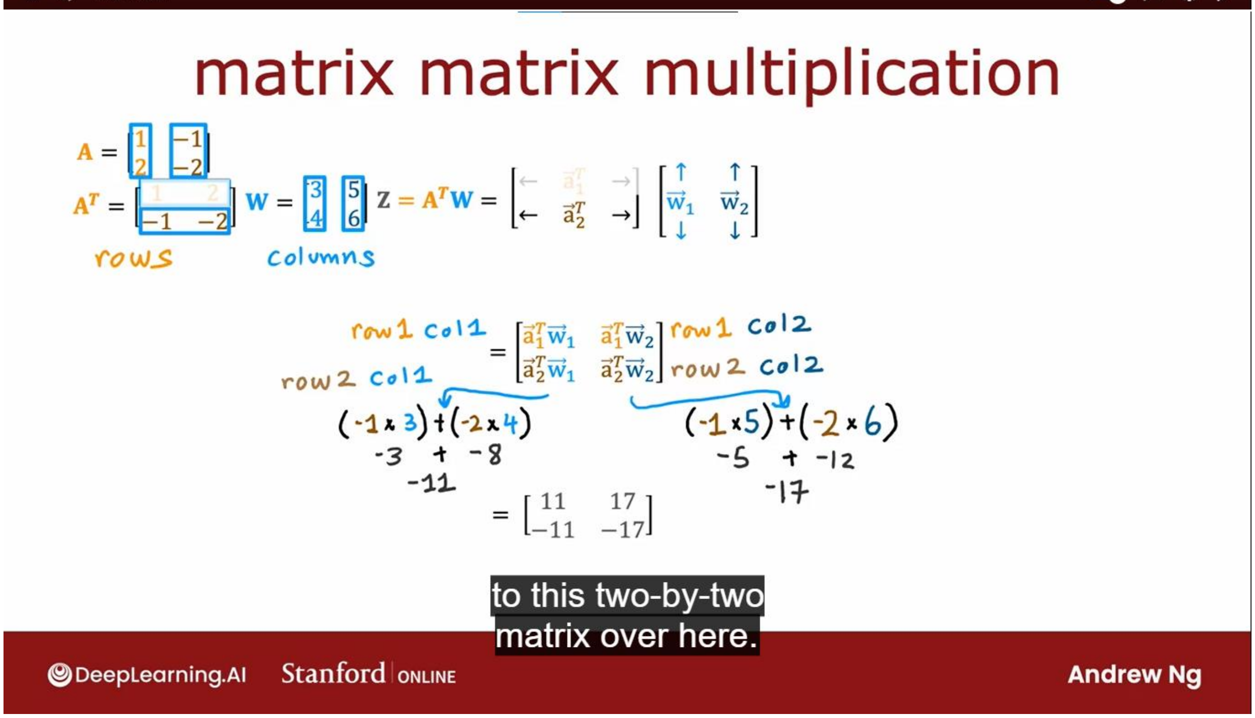
So because it's a matrix,

DeepLearning.AI Stanford ONLINE Andrew Ng

6:06 / 7:51


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]])
2D array
W = np.array([[1, -3, 5],
[-2, 4, -6]])
same
B = np.array([[-1, 1, 2]])
1x3 2D array
all 2D arrays
def dense(A_in,W,B):
vectorized
Z = np.matmul(A_in,W) + B
A_out = g(Z)
matrix multiplication
return A_out
[[1,0,1]]
through a dense layer
in the neural network.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
3:27 / 4:21


Dot products
example
in general
transpose
vector vector multiplication
$\begin{bmatrix}1\\2\end{bmatrix} \cdot \begin{bmatrix}3\\4\end{bmatrix}$
$\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}$
$\vec{a}^T = [1 \quad 2]$
1x2
2x1
$z = (1 \times 3) + (2 \times 4)$
3 + 8
11
$z = \vec{a} \cdot \vec{w}$
equivalent
$z = \vec{a}^T \vec{w}$
taking the dot product
between a and w. To recap,
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Vector matrix multiplication
$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
$\vec{a}^T = [1 \quad 2]$
$W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$
$Z = \vec{a}^T W$
1 by 2
$[\leftarrow \ \vec{a}^T \ \rightarrow] \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$
$Z = [\vec{a}^T\vec{w}_1 \quad a^T\vec{w}_2]$
$(1 * 3) + (2 * 4)$
3 + 8
11
$(1 * 5) + (2 * 6)$
5 + 12
17
$Z = [11 \quad 17]$
and then that'll take us
to the end of this video,
DeepLearning.AI
Stanford ONLINE
Andrew Ng
4:32 / 9:28

matrix matrix multiplication
$A = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$
$A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix}$
rows
$W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$
columns
$Z = A^T W = \begin{bmatrix} \leftarrow & \vec{a}_1^T & \rightarrow \\ \leftarrow & \vec{a}_2^T & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$
row1 col1
row2 col1
$= \begin{bmatrix} \vec{a}_1^T\vec{w}_1 & \vec{a}_1^T\vec{w}_2 \\ \vec{a}_2^T\vec{w}_1 & \vec{a}_2^T\vec{w}_2 \end{bmatrix}$
row1 col2
row2 col2
$(-1 \times 3) + (-2 \times 4)$
-3 + -8
-11
$(-1 \times 5) + (-2 \times 6)$
-5 + -12
-17
$= \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix}$
to this two-by-two
matrix over here.
DeepLearning.AI
Stanford ONLINE
Andrew Ng

Matrix multiplication in NumPy

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

```
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]])
```

result

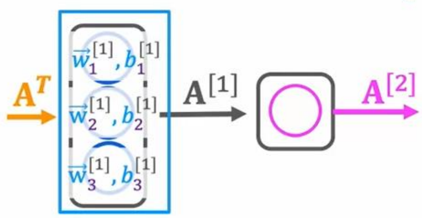
```
[[11,17,23,9],
 [-11,-17,-23,-9],
 [1.1,1.7,2.3,0.9]
]
```

```
AT=A.T
```

transpose

this rather than this @.

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

$Z = A^TW + B$

$[\boxed{165} \quad \boxed{-531} \quad \boxed{900}]$

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

$A = g(Z)$

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]]
```

the correct implementation
of the code.