Week3

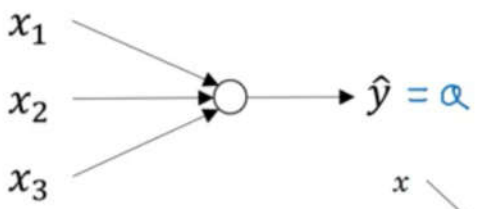
06 May 2020 18:44

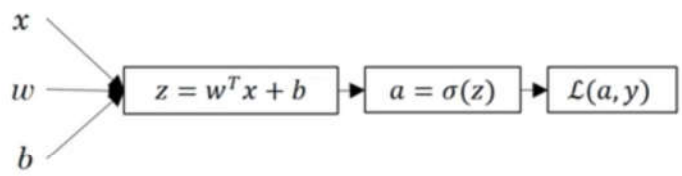
deeplearning.ai

# One hidden layer Neural Network

# Neural Networks Overview

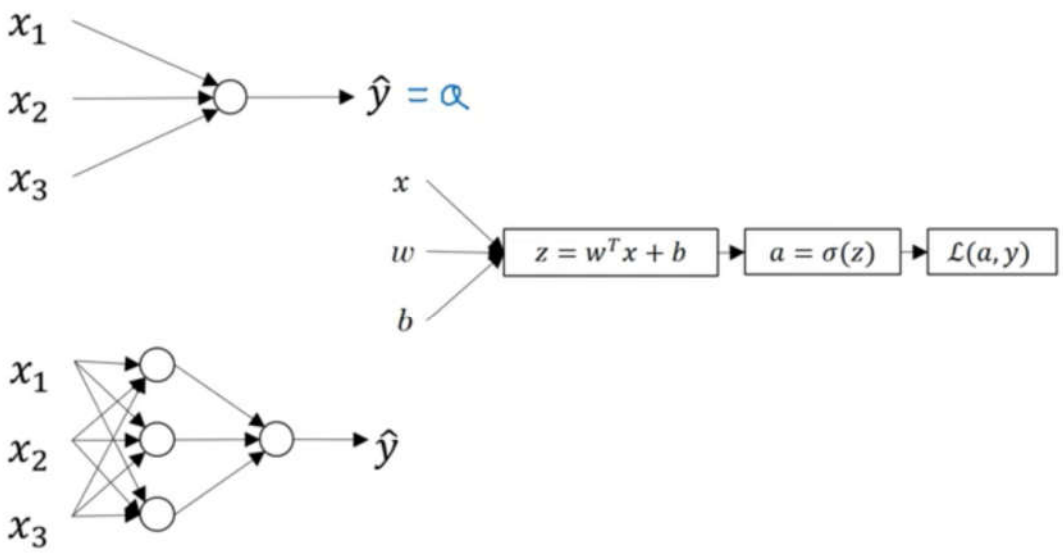
## What is a Neural Network?




Andrew Ng

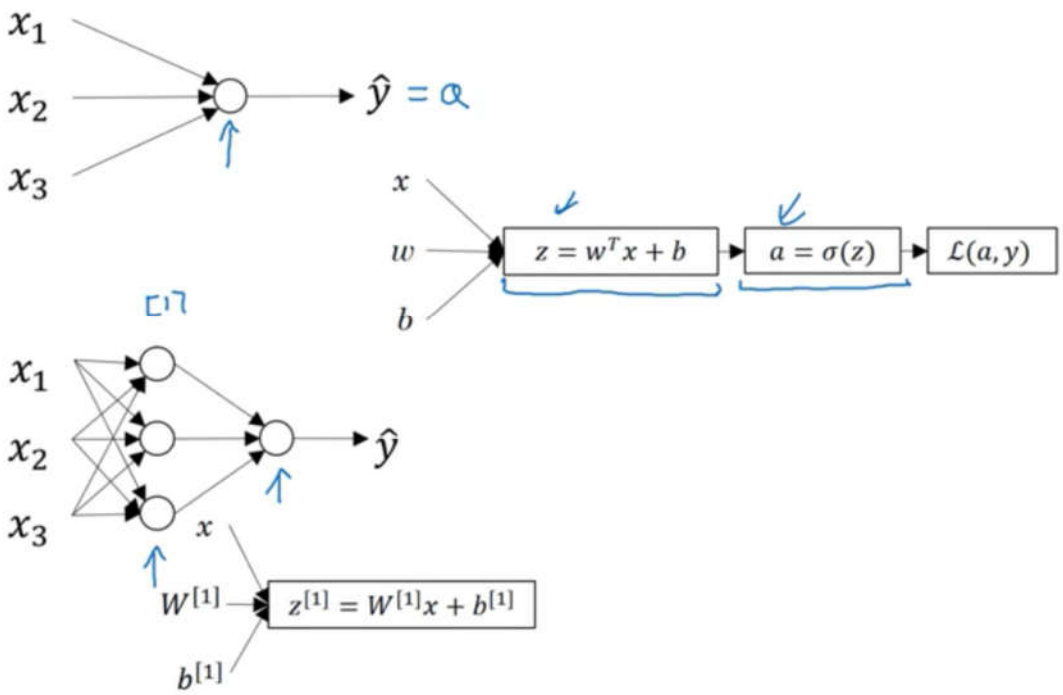
# What is a Neural Network?




0:55 / 4:26

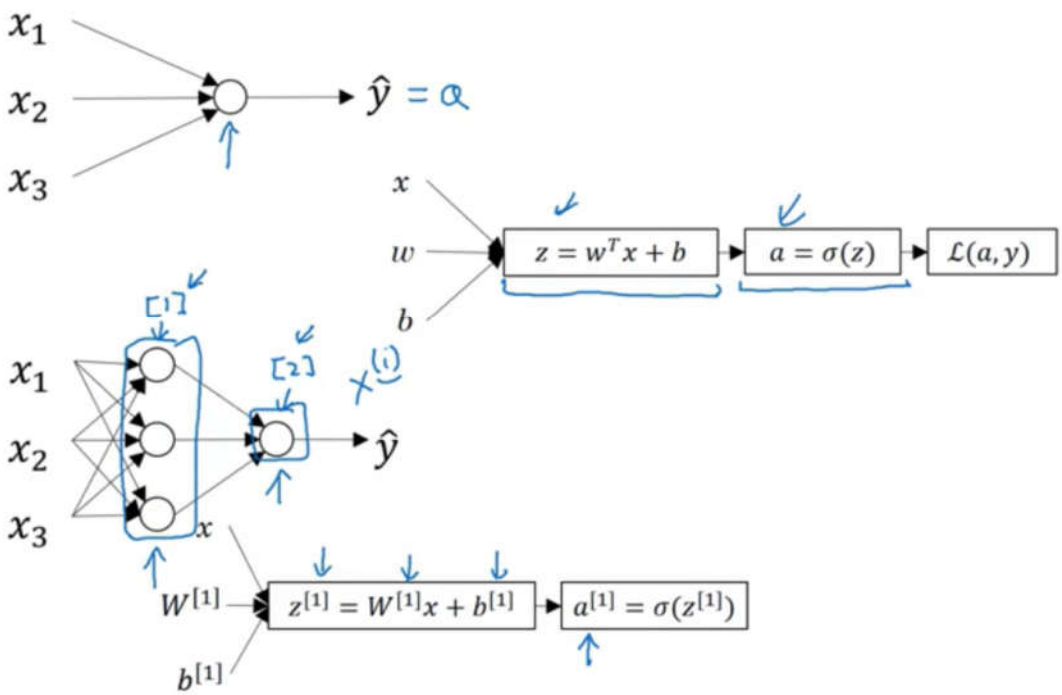
Andrew Ng

# What is a Neural Network?



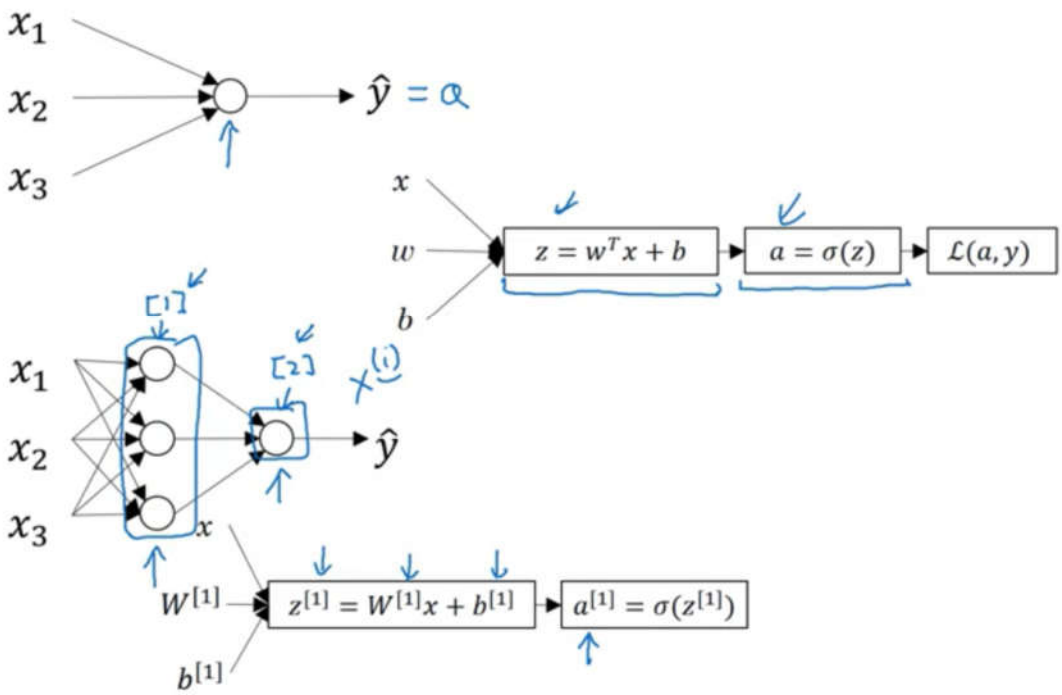

Andrew Ng

# What is a Neural Network?



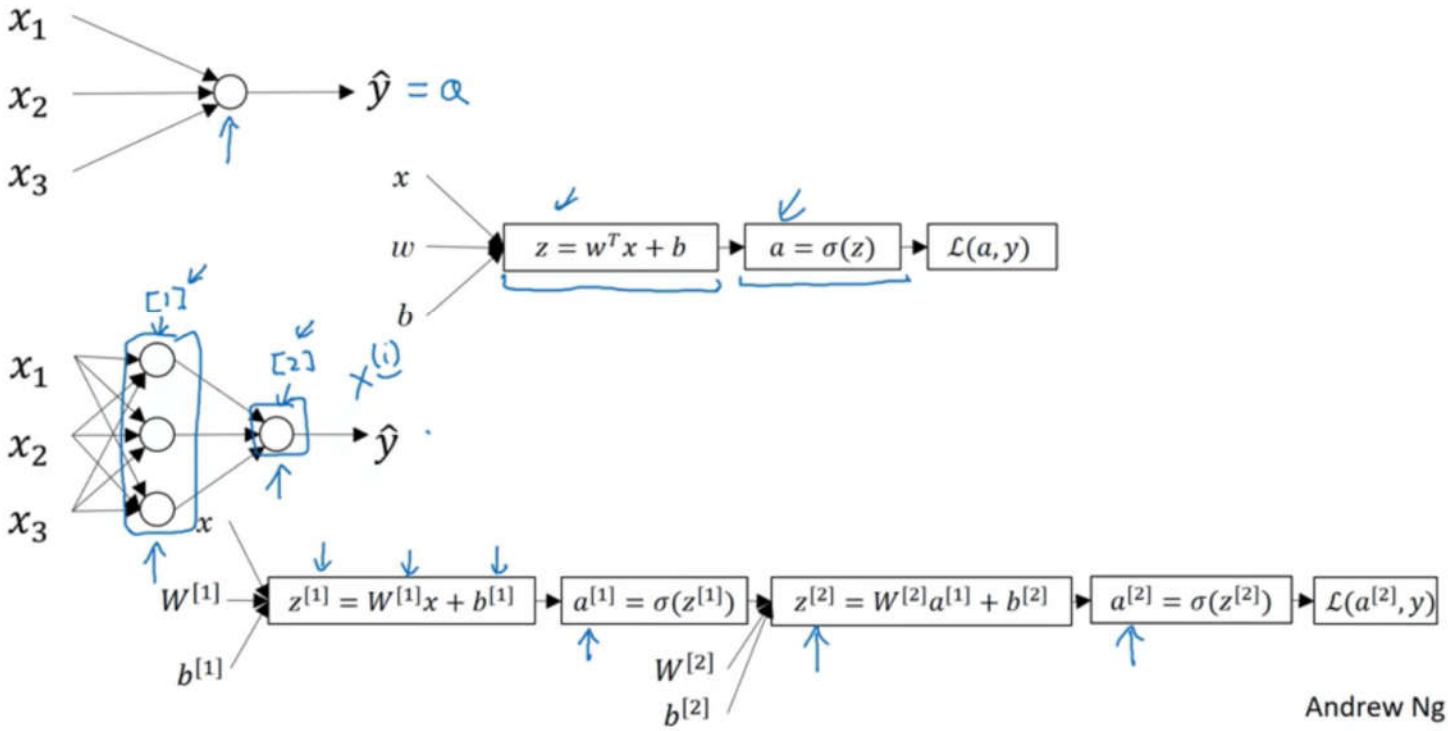

Andrew Ng

# What is a Neural Network?

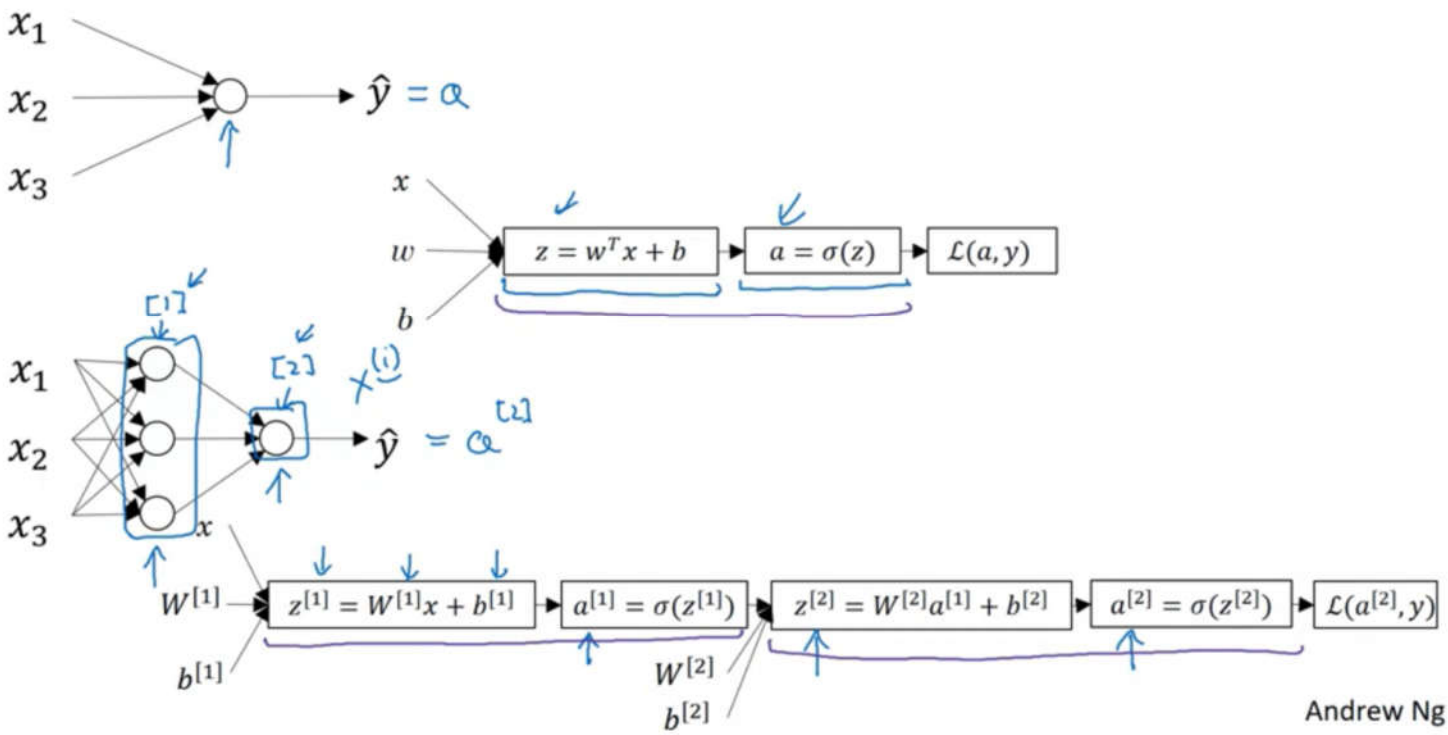



Andrew Ng

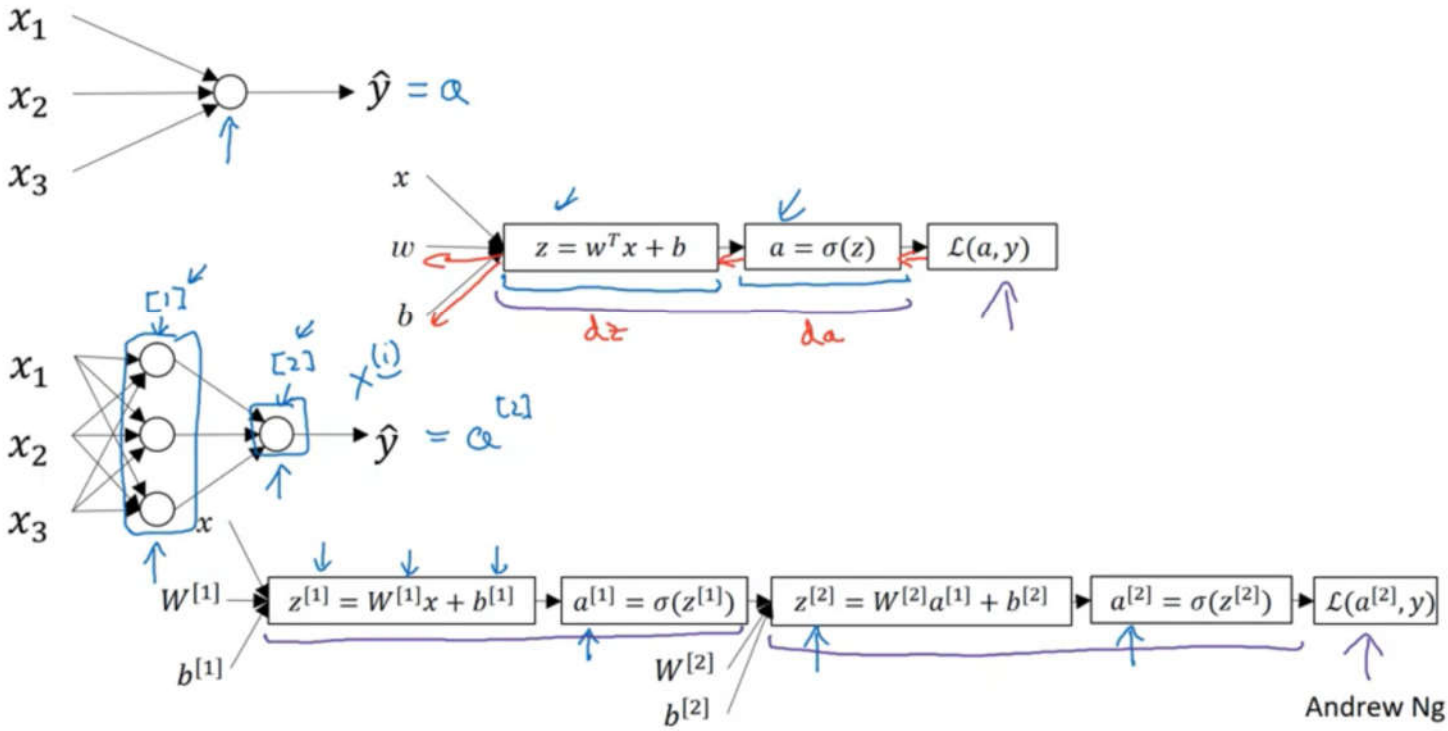
# What is a Neural Network?



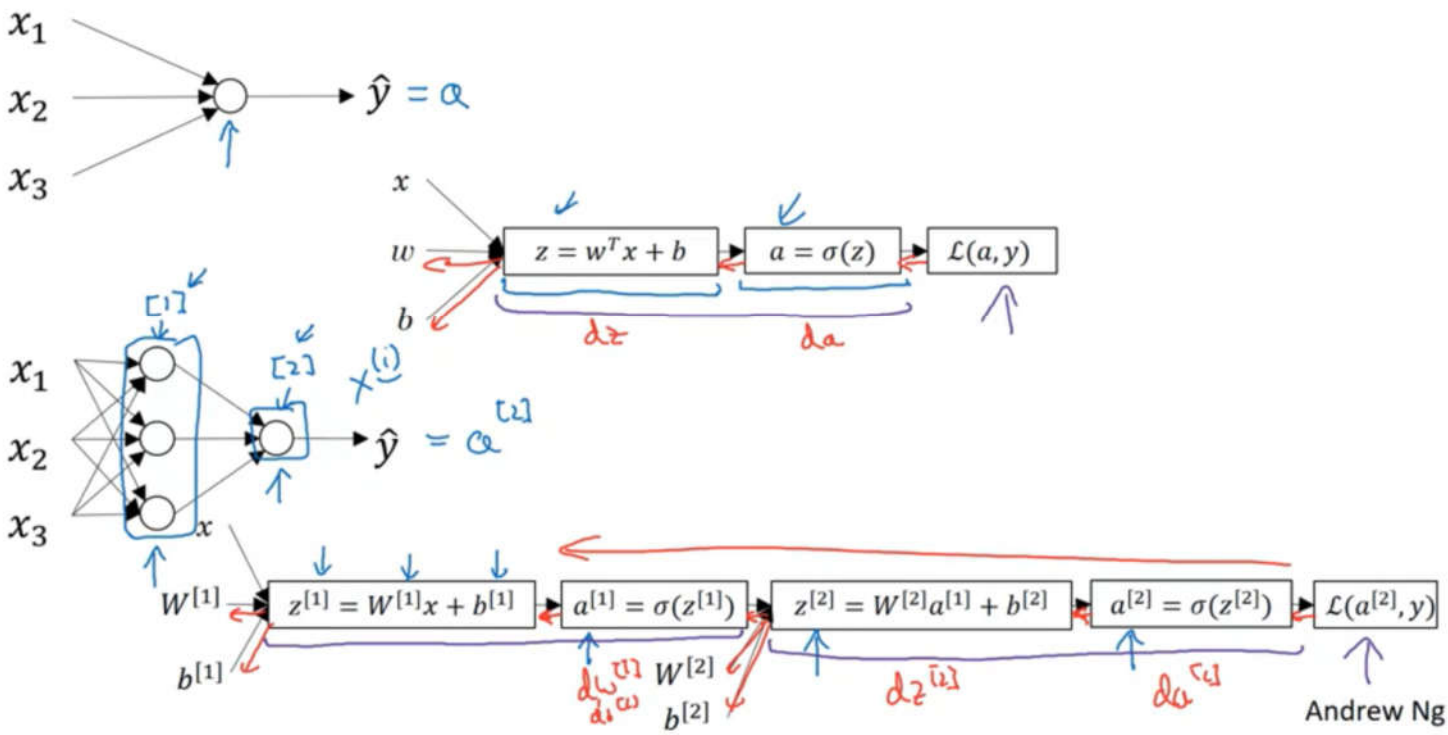

# What is a Neural Network?



# What is a Neural Network?




# What is a Neural Network?



Press Esc to exit full screen

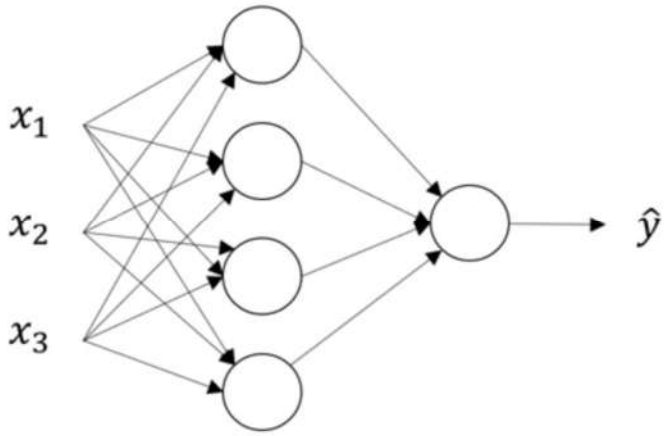
deeplearning.ai

# One hidden layer Neural Network

## Neural Network Representation

0:02 / 5:14

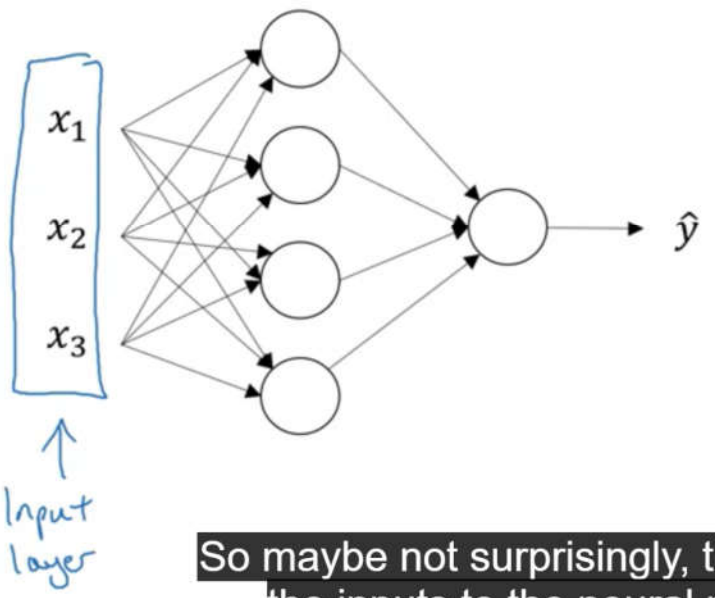
## Neural Network Representation




Here's a picture of a neural network.

Andrew Ng

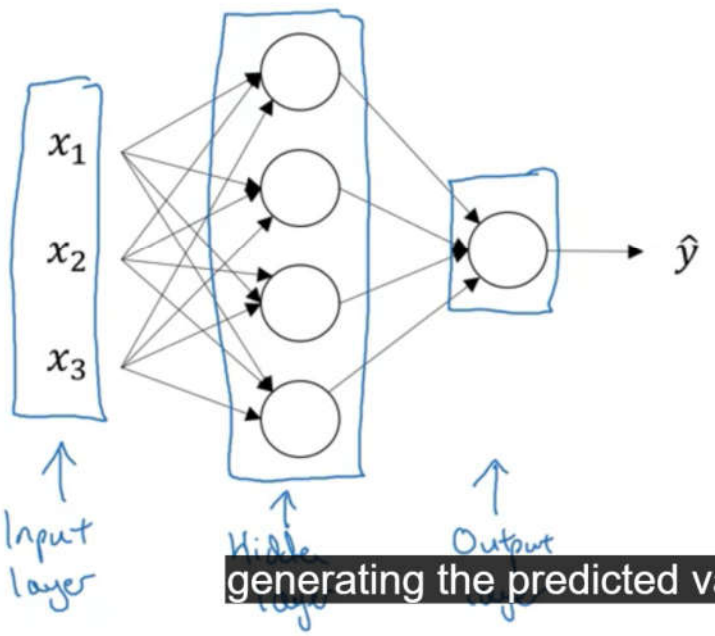
# Neural Network Representation




So maybe not surprisingly, this contains the inputs to the neural network.

Andrew Ng

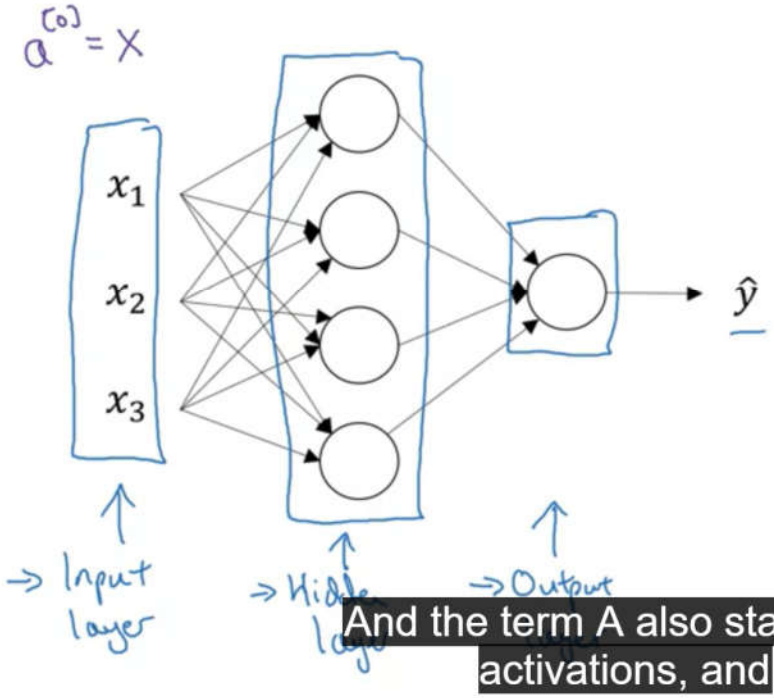
# Neural Network Representation




generating the predicted value y hat.

Andrew Ng

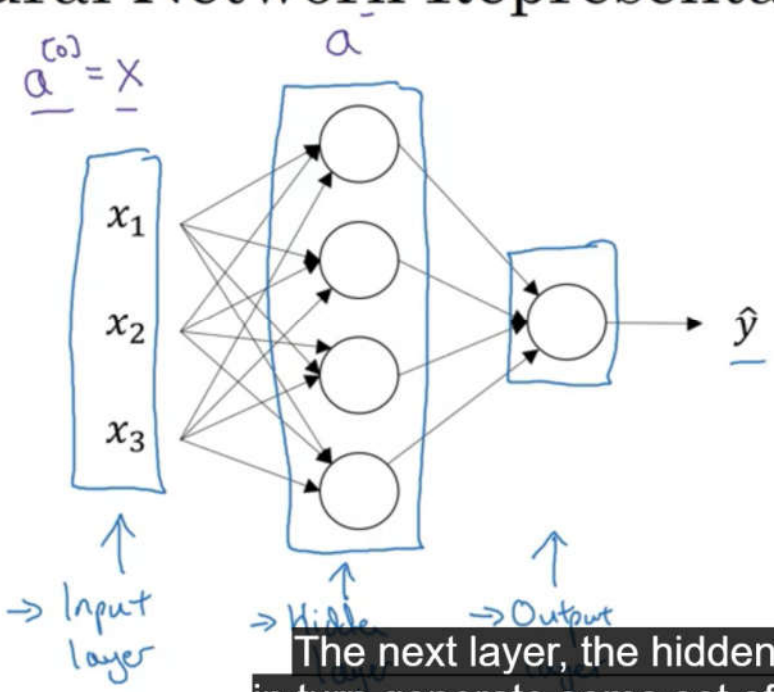
# Neural Network Representation




And the term A also stands for activations, and

Andrew Ng

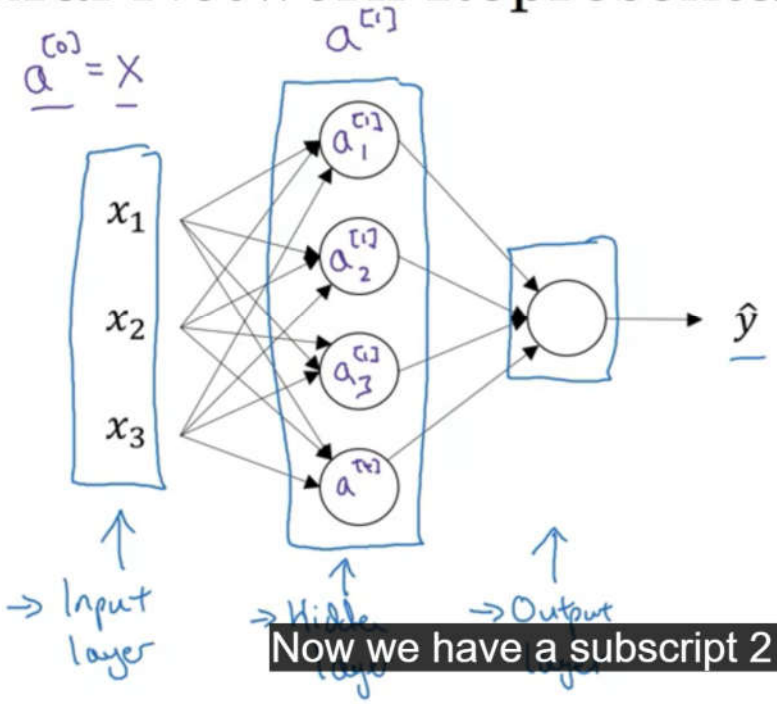
# Neural Network Representation




The next layer, the hidden layer, will in turn generate some set of activations,

Andrew Ng

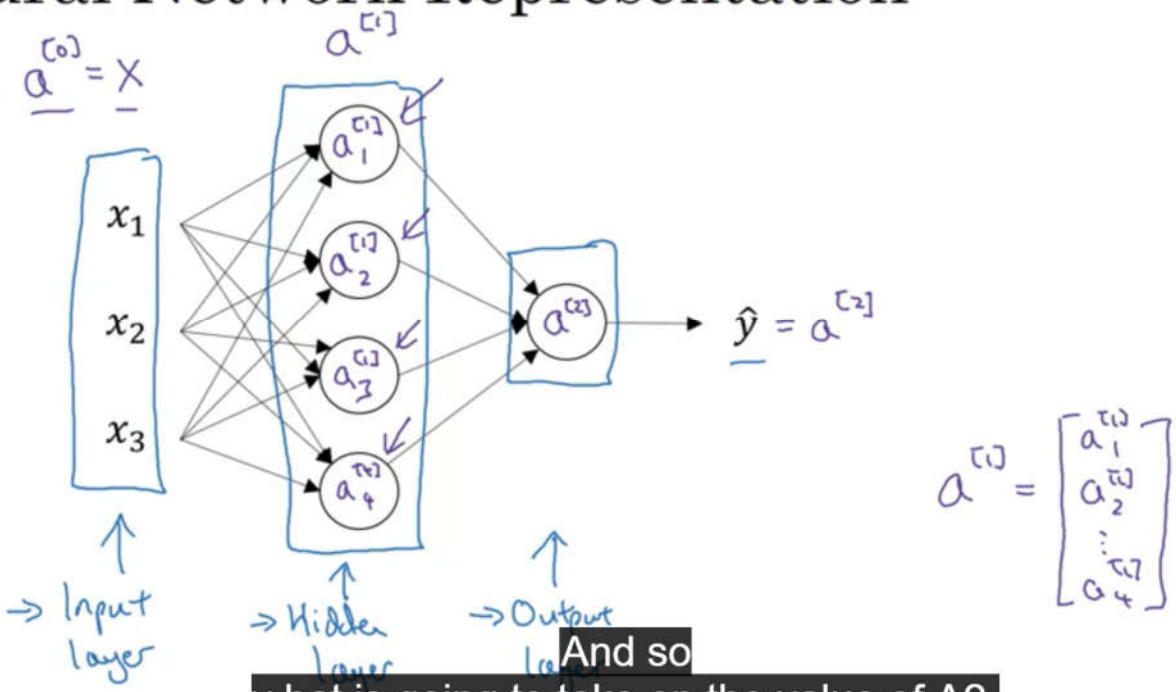
# Neural Network Representation




Now we have a subscript 2 and so on.

Andrew Ng

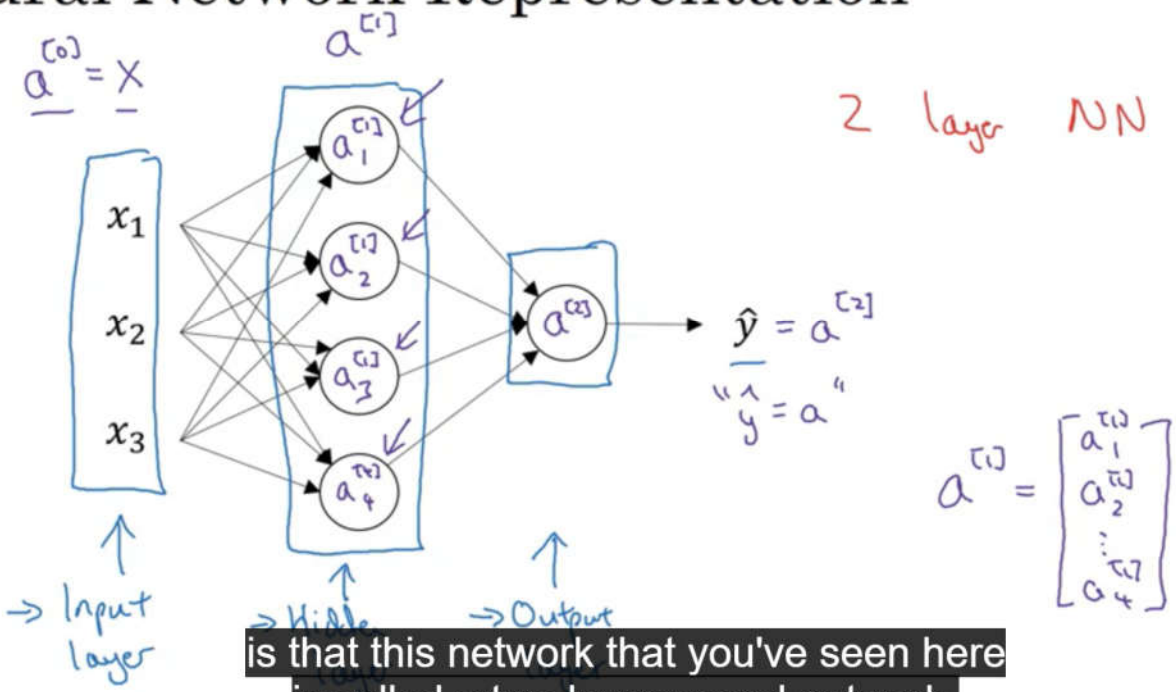
# Neural Network Representation




And so
y hat is going to take on the value of A2.

Andrew Ng

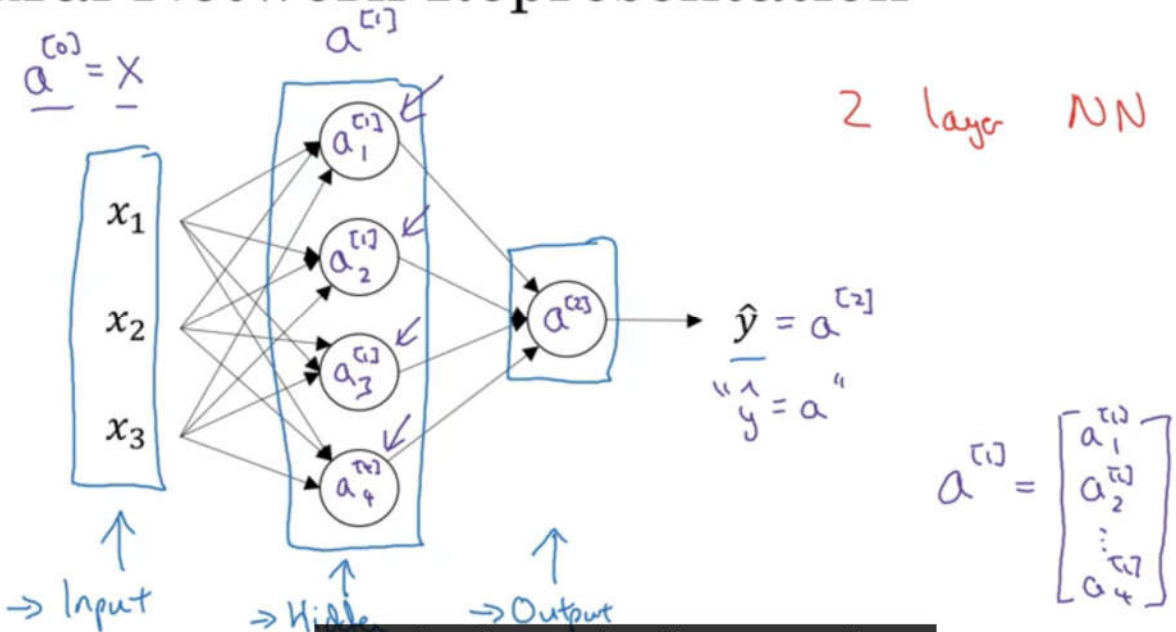
# Neural Network Representation




is that this network that you've seen here
is called a two layer neural network.

Andrew Ng

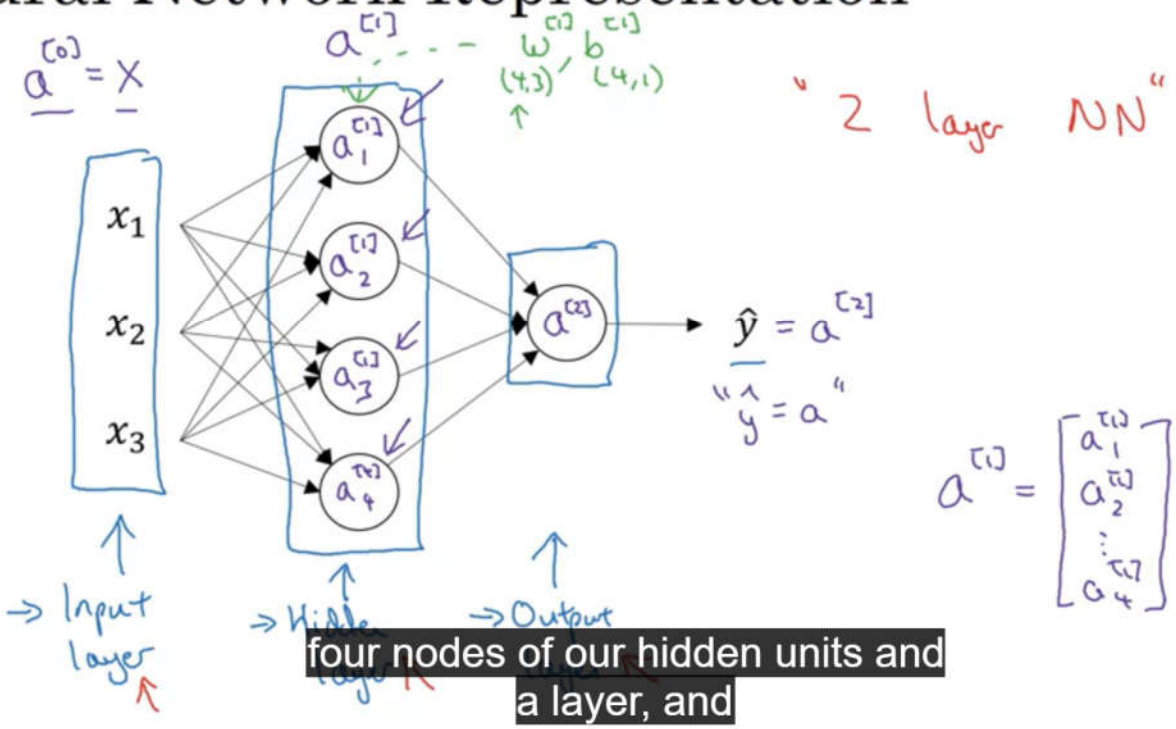
# Neural Network Representation




technically maybe there are three
layers in this neural network.

Andrew Ng

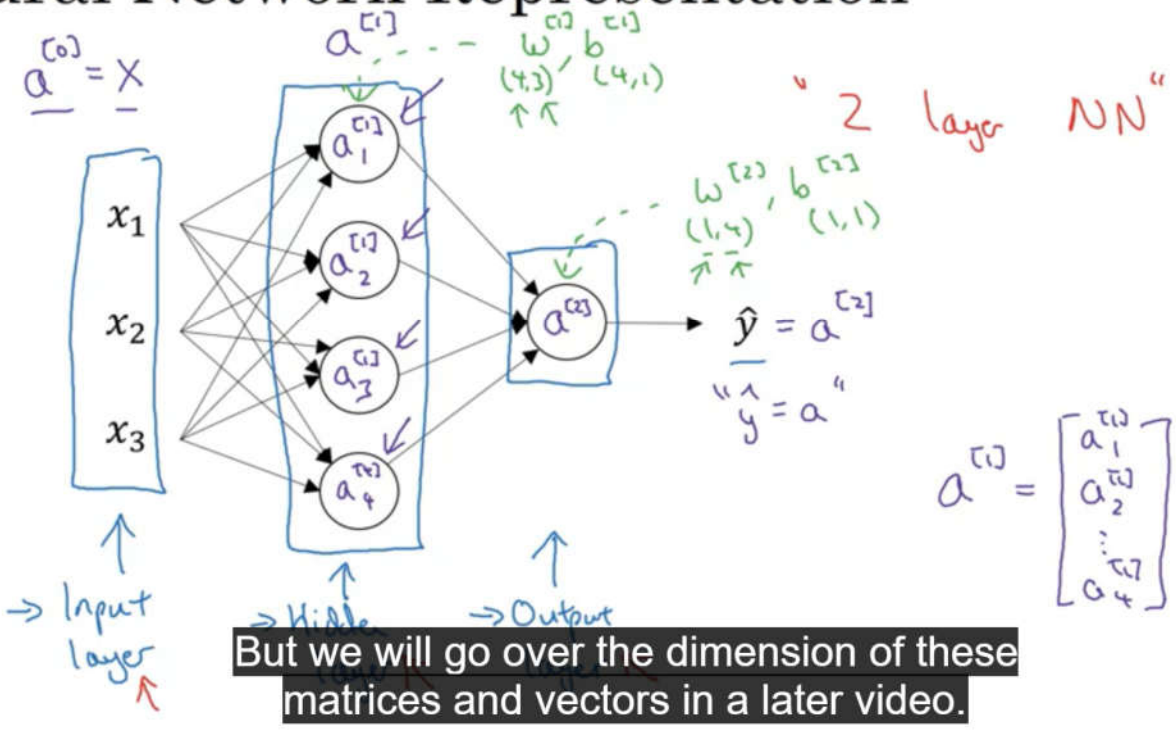
# Neural Network Representation




four nodes of our hidden units and a layer, and

Andrew Ng

# Neural Network Representation




But we will go over the dimension of these matrices and vectors in a later video.

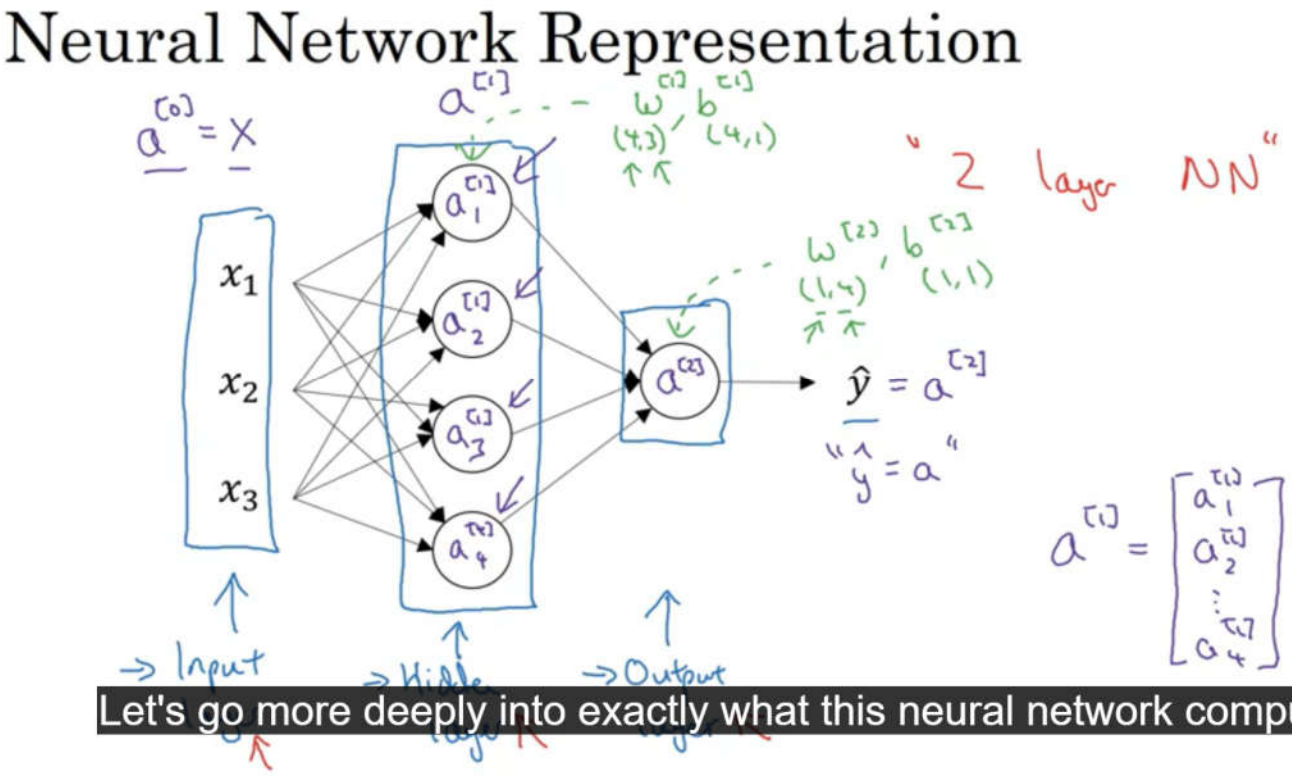
Andrew Ng

deeplearning.ai

# One hidden layer Neural Network

## Computing a Neural Network's Output

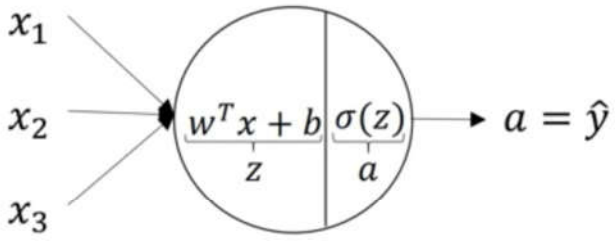
In the last video, you saw what a single hidden layer neural network looks like.




Let's go more deeply into exactly what this neural network computes.

Andrew Ng

# Neural Network Representation



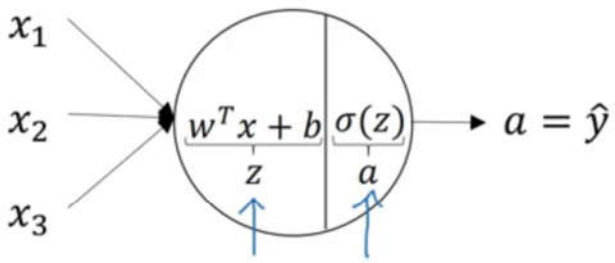

$z = w^T x + b$

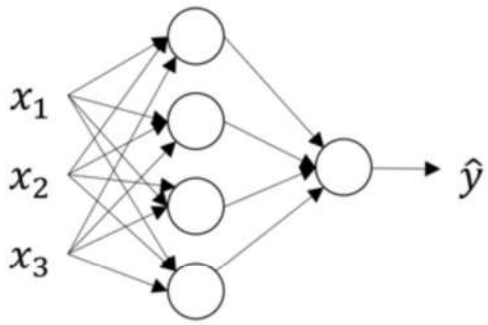
$a = \sigma(z)$

Now, we've said before that logistic regression,

Andrew Ng

# Neural Network Representation



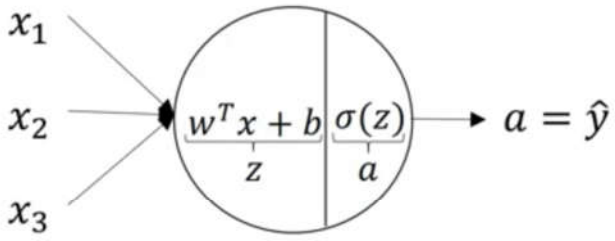

$z = w^T x + b$

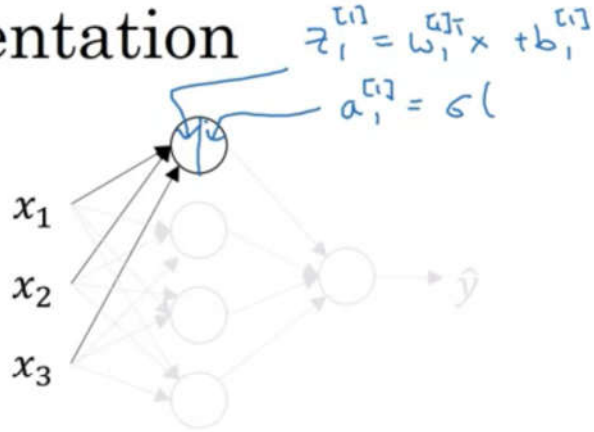
$a = \sigma(z)$

So, a neural network just does this a lot more times.

Andrew Ng

# Neural Network Representation



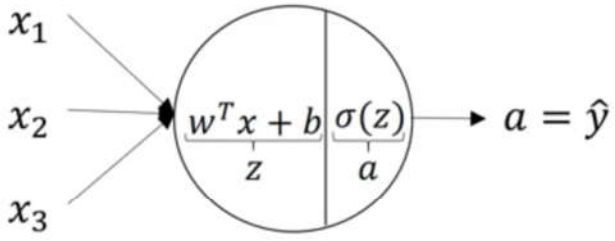

$z = w^T x + b$

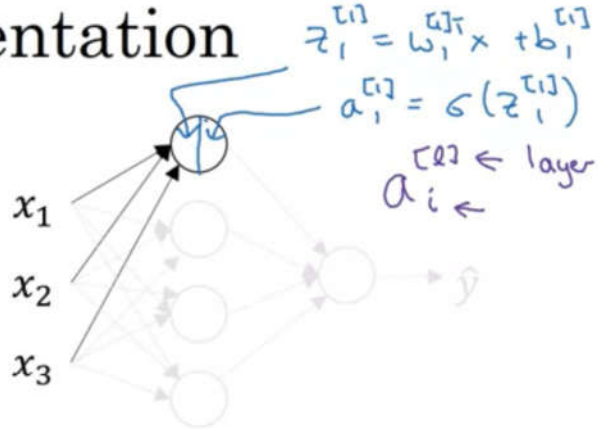
$a = \sigma(z)$

is it computes a_[1]_1 equals sigmoid of z_[1]_1, like so.

Andrew Ng

# Neural Network Representation



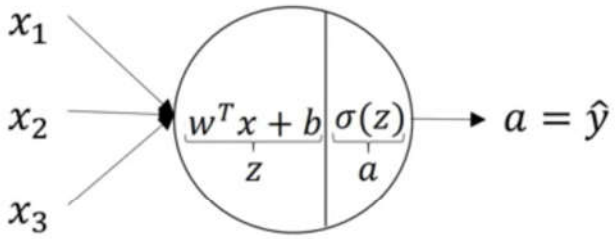

$z = w^T x + b$

$a = \sigma(z)$

and the i subscript here,

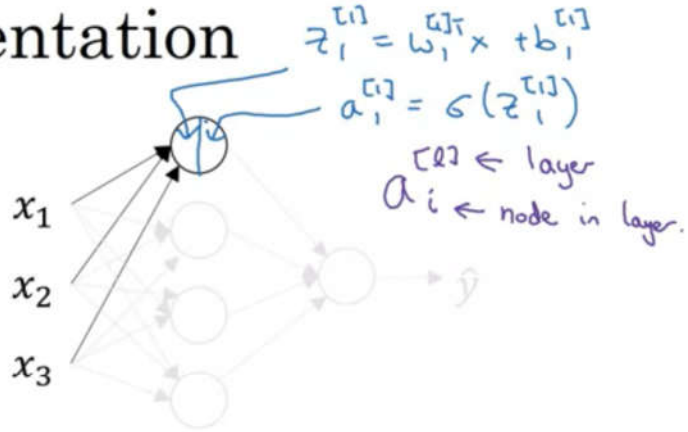
Andrew Ng

# Neural Network Representation




$z = w^T x + b$

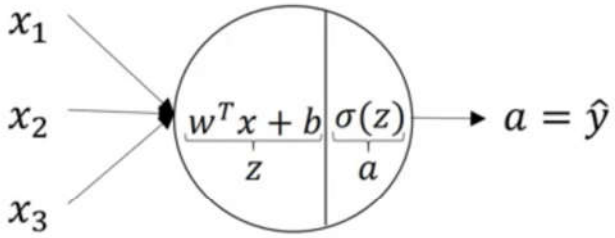
$a = \sigma(z)$




So, the node we'll be looking at is layer one,

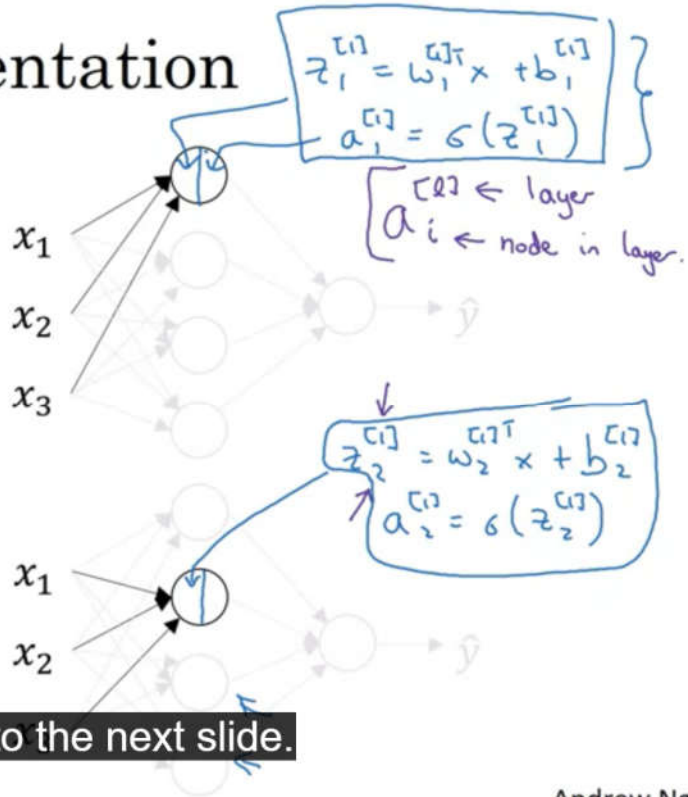
Andrew Ng

# Neural Network Representation




$z = w^T x + b$

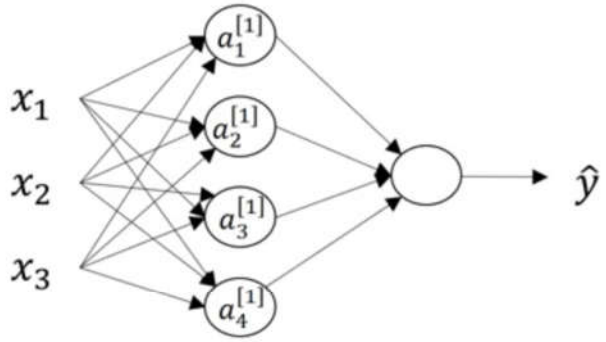
$a = \sigma(z)$




and let's copy them to the next slide.

Andrew Ng

# Neural Network Representation



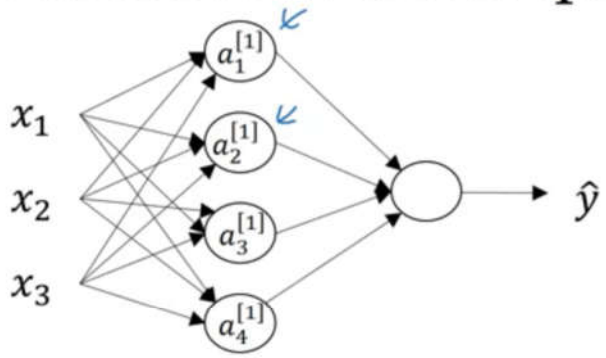

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

and here's the second equations that we've worked out

Andrew Ng

# Neural Network Representation




$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

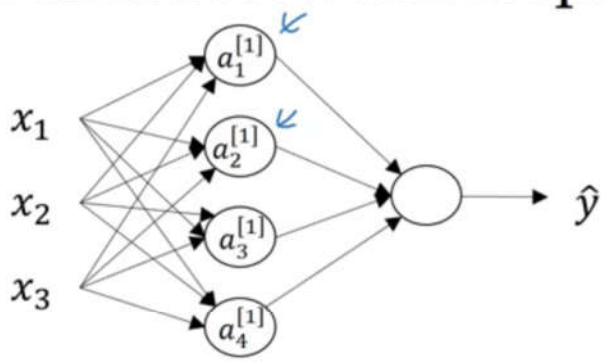
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

If you then go through and write out the corresponding equations

Andrew Ng

# Neural Network Representation



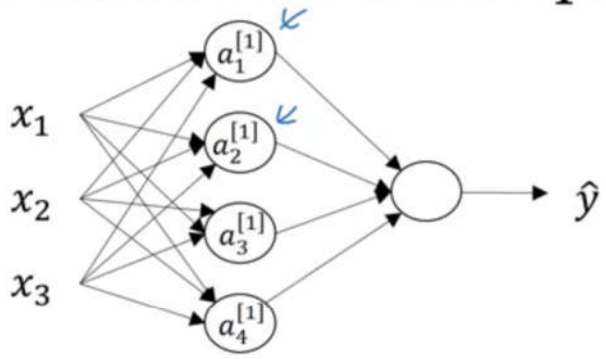

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

for the third and fourth hidden units, you get the following.

Andrew Ng

# Neural Network Representation

$(w_1^{[1]})^T x$




$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

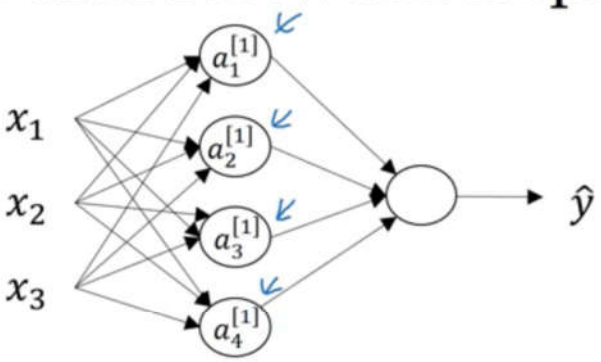
$$\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}$$

So, by stacking those four w vectors together,

Andrew Ng

# Neural Network Representation

$(w_1^{[1]})^T x$



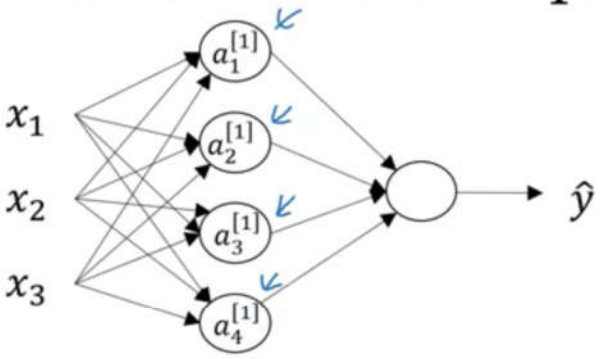

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$\underset{(4,3)}{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

Andrew Ng

# Neural Network Representation

$(w_1^{[1]})^T x$

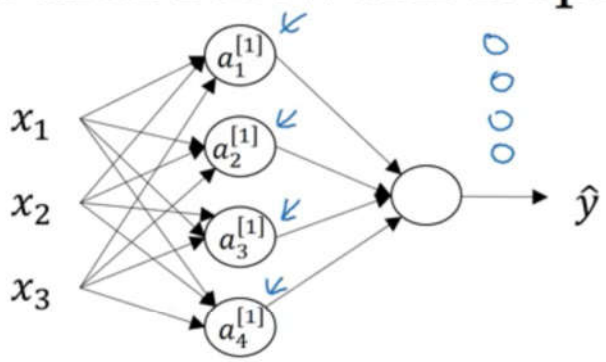



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$\underset{(4,3)}{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Andrew Ng

# Neural Network Representation




$(w_1^{[1]})^T x$

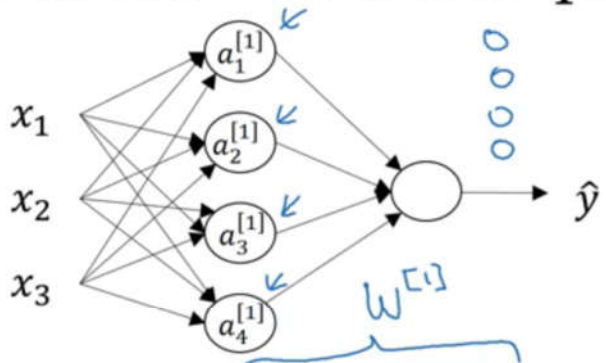
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

5:26 / 9:57

Andrew Ng

# Neural Network Representation



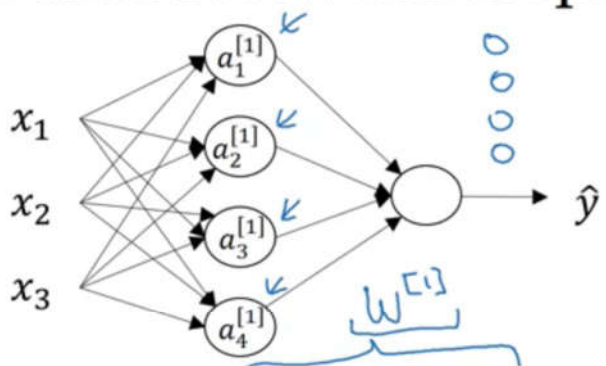

$(w_1^{[1]})^T x$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \overbrace{\underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{(4,3)}}^{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}\ (4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Andrew Ng

# Neural Network Representation




$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$(w_1^{[1]})^T x$ $\quad a^{[1]}$

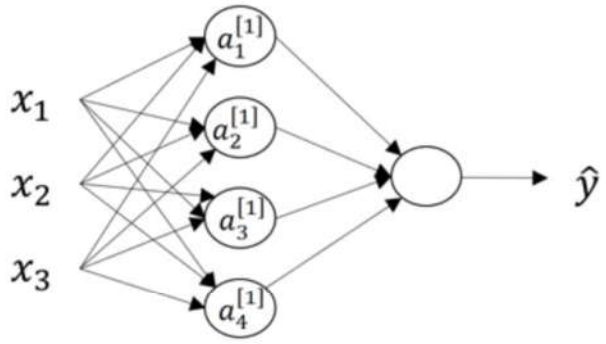
$$\rightarrow z^{[1]} = \underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix} \rightarrow w_1^{[1]T} x + b_1^{[1]} \\ \rightarrow w_2^{[1]T} x + b_2^{[1]} \\ \rightarrow w_3^{[1]T} x + b_3^{[1]} \\ \rightarrow w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$(4,3)$ $\quad (4,1)$

$$\rightarrow a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Andrew Ng

# Neural Network Representation learning




Given input x:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

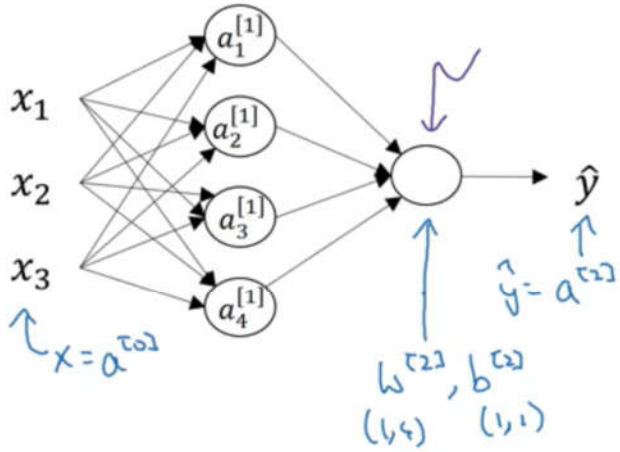
$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Andrew Ng

# Neural Network Representation learning




Given input x:

$$\rightarrow \underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}} \underset{(3,1)}{\overset{a^{[0]}}{\cancel{x}}} + \underset{(4,1)}{b^{[1]}}$$

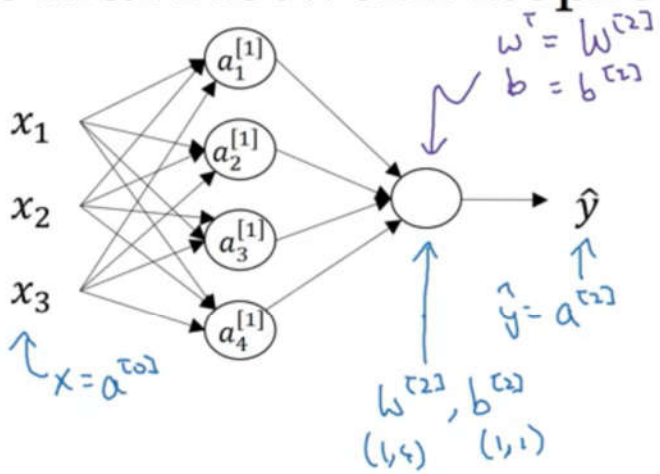
$$\rightarrow \underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

$$\rightarrow \underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\rightarrow \underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

Andrew Ng

# Neural Network Representation learning




Given input x:

$$\rightarrow \underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}} \underset{(3,1)}{\overset{a^{[0]}}{\cancel{x}}} + \underset{(4,1)}{b^{[1]}}$$

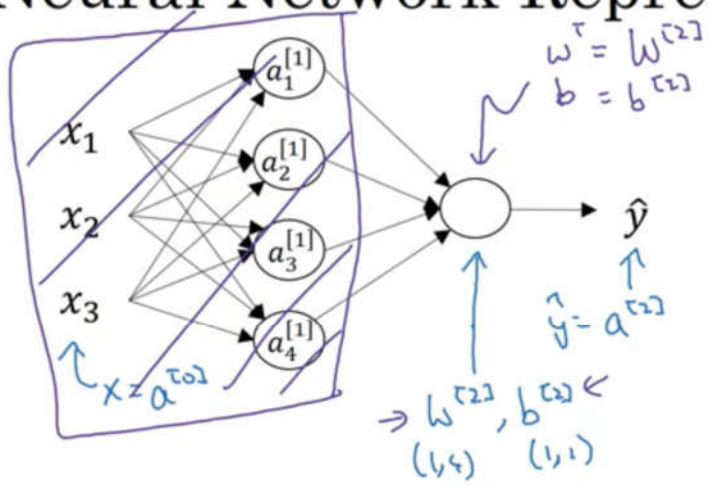
$$\rightarrow \underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

$$\rightarrow \underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\rightarrow \underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

Andrew Ng

# Neural Network Representation learning




$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

Given input x:

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$
(4,1) (4,1)

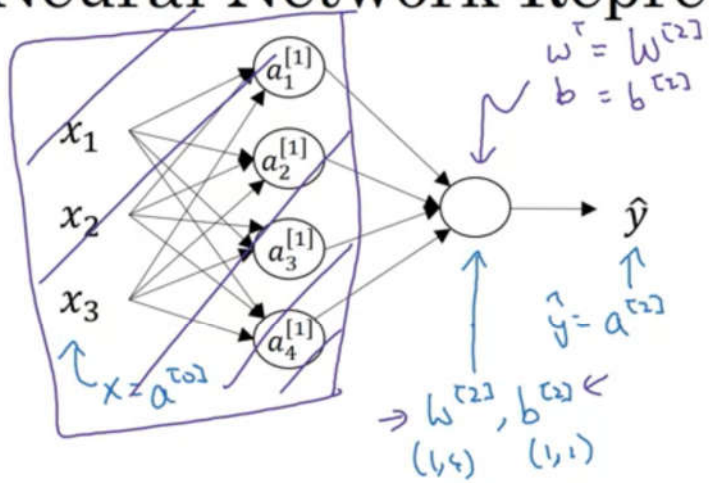
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

equals sigmoid of z.

Andrew Ng

# Neural Network Representation learning




$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

Given input x:

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$
(4,1) (4,1)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

is to computer this output is just these four equations.

Andrew Ng

Press Esc to exit full screen

deeplearning.ai

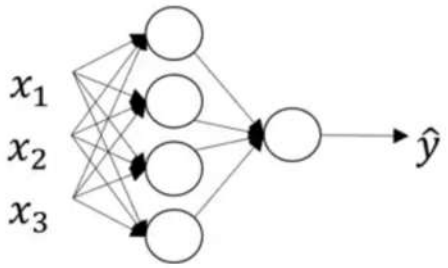
# One hidden layer Neural Network

## Vectorizing across multiple examples

In the last video, you saw how to compute the prediction on a neural network,

0:00 / 9:05

## Vectorizing across multiple examples

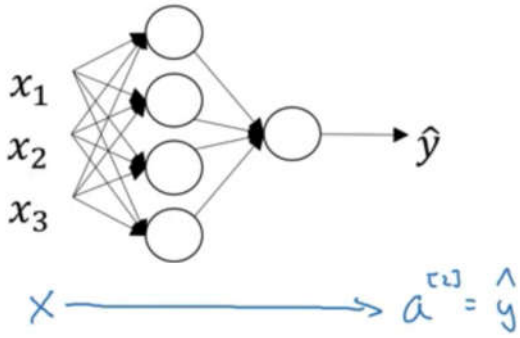



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

These were the four equations we have from the previous video of how you compute z1,

Andrew Ng

# Vectorizing across multiple examples

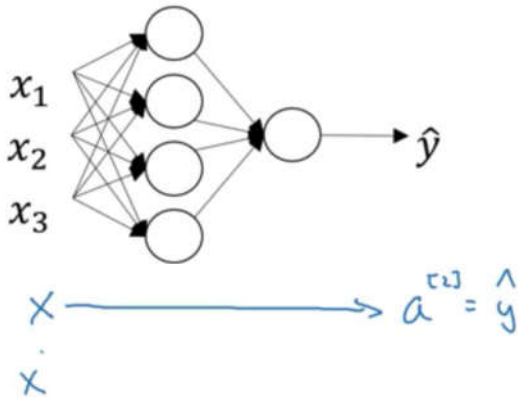



$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

you can use them to generate a2 =y hat for
a single training example.

Andrew Ng

# Vectorizing across multiple examples

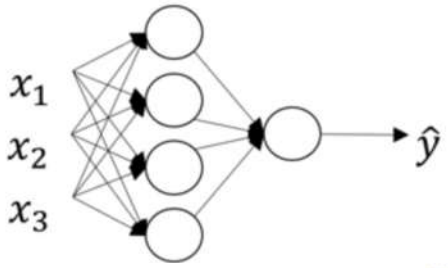



$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

Now if you have m training examples,
you need to repeat this process for

Andrew Ng

# Vectorizing across multiple examples




$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

$X \longrightarrow a^{[2]} = \hat{y}$

$X^{(1)} \longrightarrow = \hat{y}^{(1)}$

$X^{(2)} \longrightarrow \hat{y}^{(2)}$

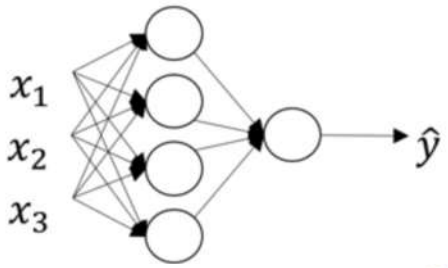
$\vdots$

$X^{(m)} \longrightarrow \hat{y}^{(m)}$

And so in all these activation function notation as well,

Andrew Ng

# Vectorizing across multiple examples




$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

$X \longrightarrow a^{[2]} = \hat{y}$

$X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$X^{(2)} \longrightarrow a^{[2](2)} \; \hat{y}^{(2)}$

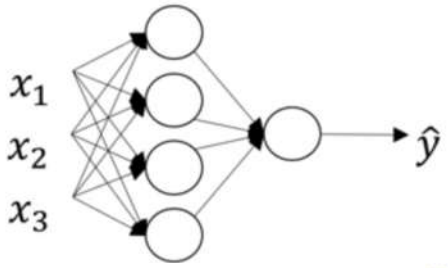
$\vdots$

$X^{(m)} \longrightarrow a^{[2](m)} \; \hat{y}^{(m)}$

and a(2)(m), so

Andrew Ng

# Vectorizing across multiple examples




$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

$$X \longrightarrow a^{[2]} = \hat{Y}$$

$$X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$X^{(2)} \longrightarrow a^{[2](2)} \quad \hat{y}^{(2)}$$

$$\vdots$$

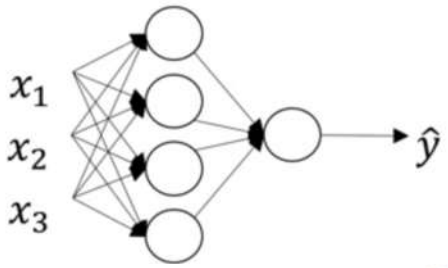
$$X^{(m)} \longrightarrow a^{[2](m)} \quad \hat{y}^{(m)}$$

$$a^{[2](i)}$$

The round bracket i refers to training example i,

Andrew Ng

# Vectorizing across multiple examples




$$\begin{cases} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

$$X \longrightarrow a^{[2]} = \hat{Y}$$

$$X^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$X^{(2)} \longrightarrow a^{[2](2)} \quad \hat{y}^{(2)}$$

$$\vdots$$

$$X^{(m)} \longrightarrow a^{[2](m)} \quad \hat{y}^{(m)}$$

$$a^{[2](i)}$$

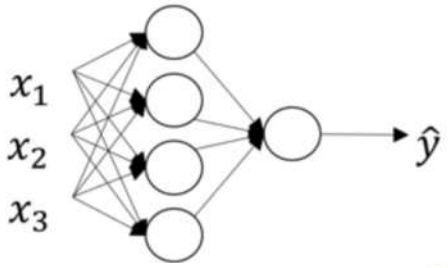
example

layer 2

and the square bracket 2 refers to layer 2, okay.

Andrew Ng

# Vectorizing across multiple examples




$$\left\{\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \end{aligned}\right\}$$

$$X \longrightarrow a^{[2]} = \hat{Y}$$

$$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$x^{(2)} \longrightarrow a^{[2](2)} \quad \hat{y}^{(2)}$$

$$\vdots$$

$$x^{(m)} \longrightarrow a^{[2](m)} \quad \hat{y}^{(m)}$$

$a^{[2](i)}$ ← example $i$, layer 2

for $i = 1$ to $m$,

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

Andrew Ng

# Vectorizing across multiple examples

Press Esc to exit full screen

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

4:01 / 9:05

Andrew Ng

# Vectorizing across multiple examples

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$
$$(n_x, m)$$

Andrew Ng

# Vectorizing across multiple examples

```
for i = 1 to m:
```

$$\boxed{z^{[1](i)}} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$
$$(n_x, m)$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Andrew Ng

# Vectorizing across multiple examples

```
for i = 1 to m:
```

$$\boxed{z^{[1](i)}} = W^{[1]}x^{(i)} + b^{[1]}$$
$$\boxed{a^{[1](i)}} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$
$$\uparrow \quad (n_x, m) \quad \uparrow$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Andrew Ng

# Vectorizing across multiple examples

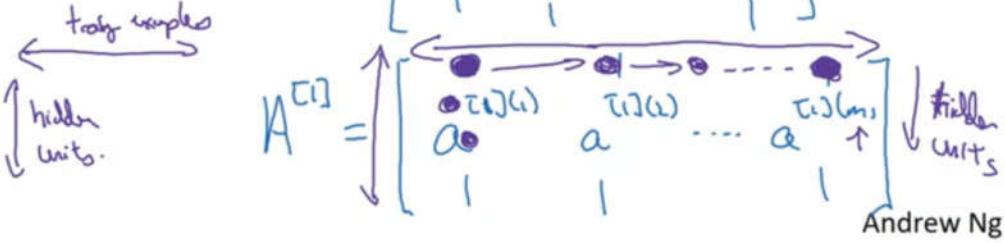
```
for i = 1 to m:
```

$$\boxed{z^{[1](i)}} = W^{[1]}x^{(i)} + b^{[1]}$$
$$\boxed{a^{[1](i)}} = \sigma(z^{[1](i)})$$
$$\boxed{z^{[2](i)}} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$\boxed{a^{[2](i)}} = \sigma(z^{[2](i)})$$

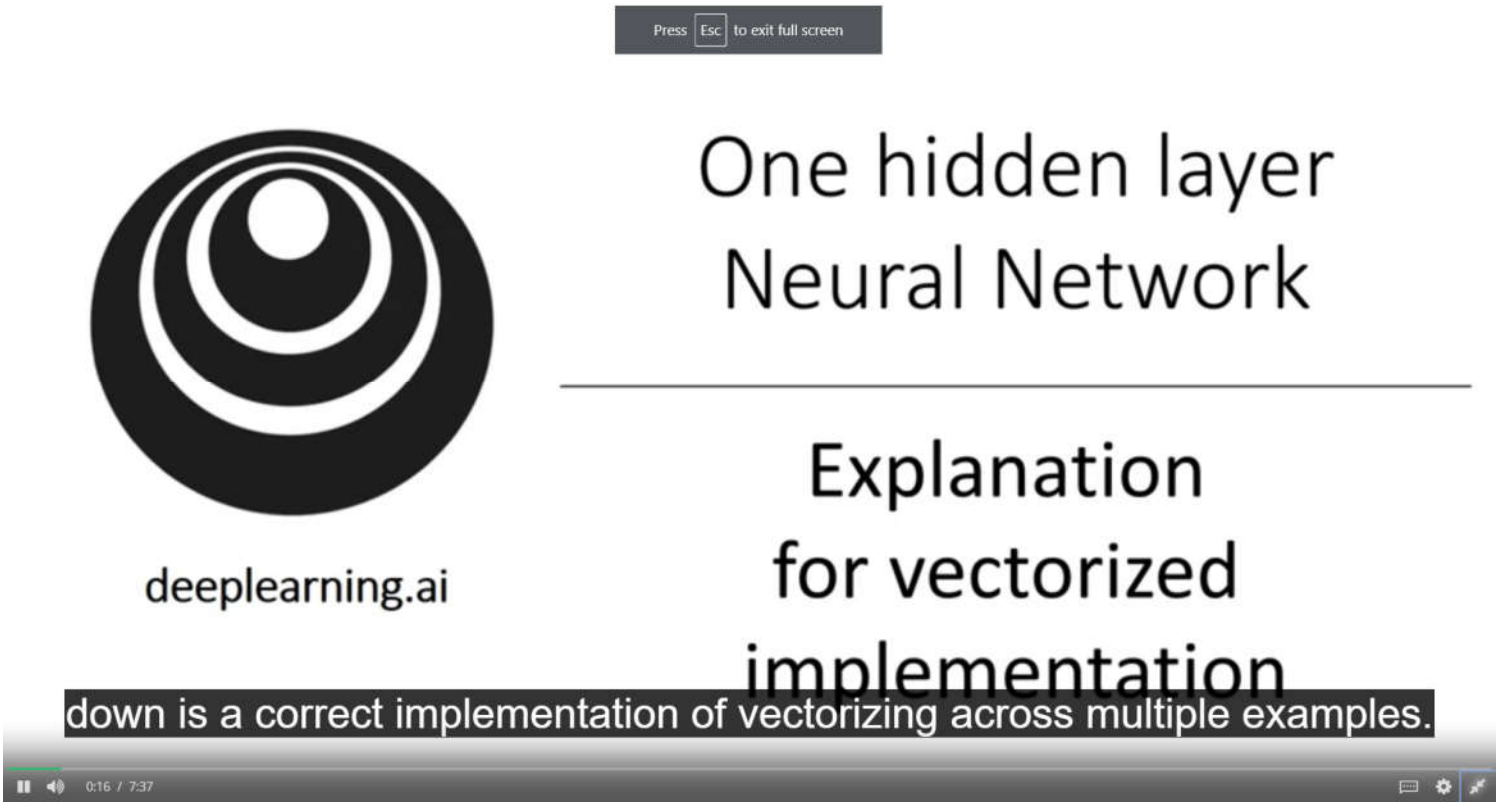
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$\rightarrow A^{[1]} = \sigma(Z^{[1]})$$
$$\rightarrow Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$\rightarrow A^{[2]} = \sigma(Z^{[2]})$$

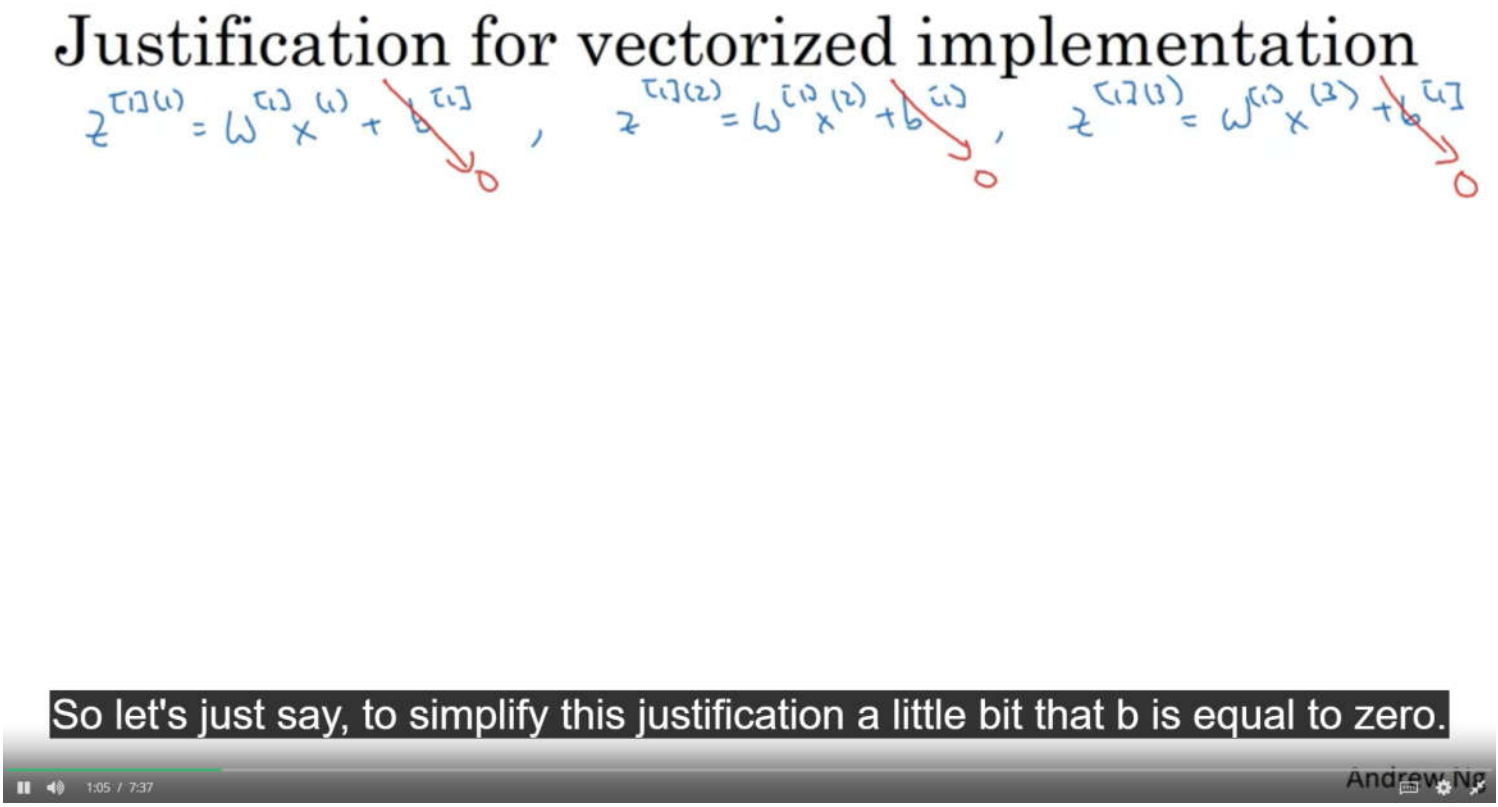
$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$
$$\uparrow \quad (n_x, m) \quad \uparrow$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$




Andrew Ng


Press Esc to exit full screen
One hidden layer Neural Network
deeplearning.ai
Explanation for vectorized implementation
down is a correct implementation of vectorizing across multiple examples.
0:16 / 7:37


Justification for vectorized implementation
$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}^{\,0}$ , $z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}^{\,0}$ , $z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}^{\,0}$
So let's just say, to simplify this justification a little bit that b is equal to zero.
1:05 / 7:37
Andrew Ng

# Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}$$

$$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

gives you some third column vector, that's this z13.

Andrew Ng

# Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}$$

$$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \cdots \\ | & | & | \end{bmatrix}}_{X} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \cdots \\ | & | & | \end{bmatrix} = Z^{[1]}$$

And so, this is just our matrix capital Z1.

Andrew Ng

# Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}^{\,0}, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}^{\,0}, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}^{\,0}$$

$$W^{[1]} = \begin{bmatrix} \text{—} \\ \text{—} \\ \text{—} \\ \text{—} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{X} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

$$W^{[1]}x^{(1)} = z^{[1](1)}$$

Andrew Ng

# Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}^{\,0}, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}^{\,0}, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}^{\,0}$$

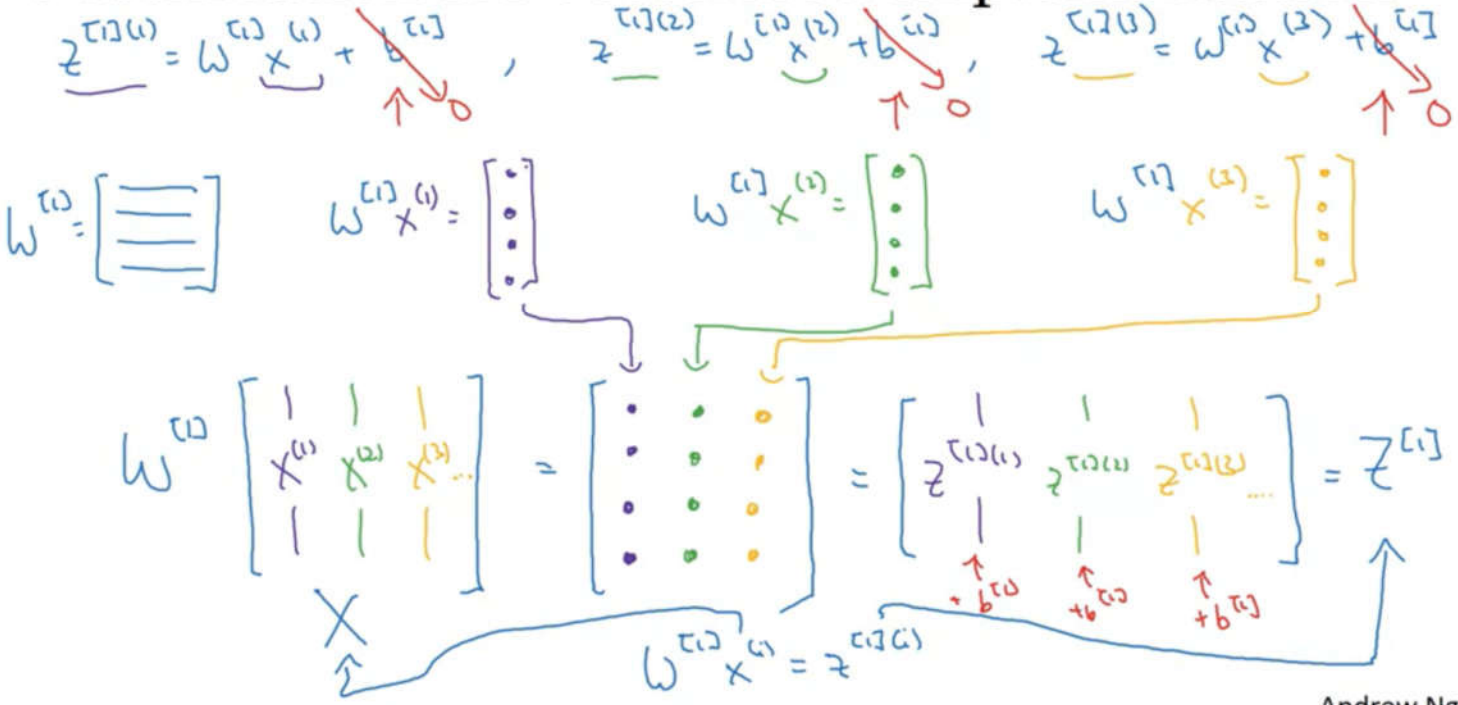
$$W^{[1]} = \begin{bmatrix} \text{—} \\ \text{—} \\ \text{—} \\ \text{—} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{X} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

$$W^{[1]}x^{(1)} = z^{[1](1)}$$

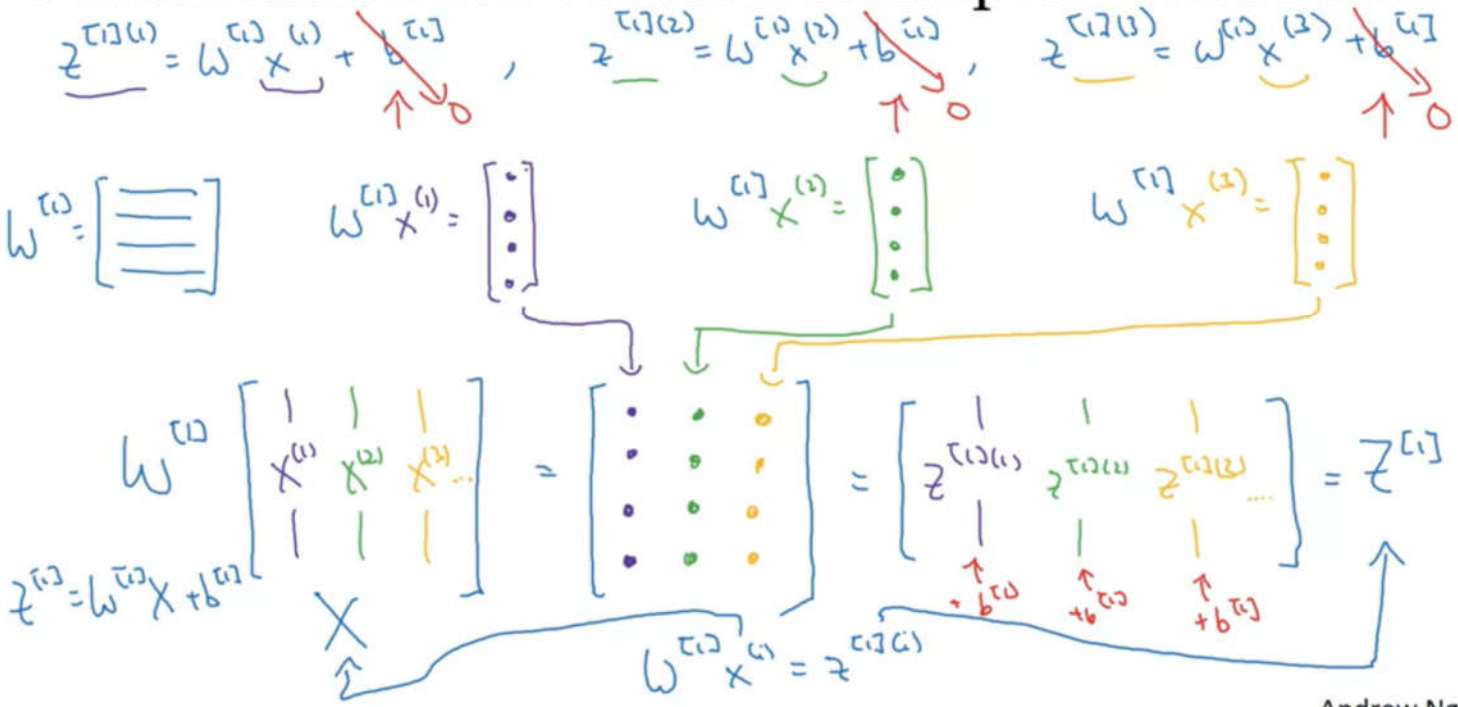
Andrew Ng

# Justification for vectorized implementation



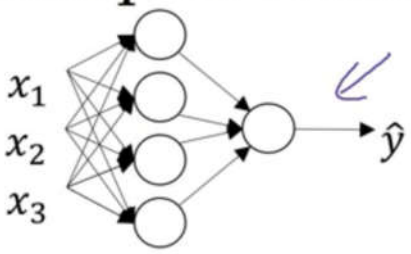

Andrew Ng

# Justification for vectorized implementation




Andrew Ng

# Recap of vectorizing across multiple examples




$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

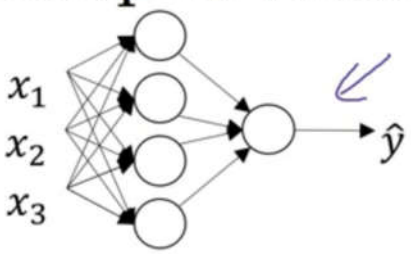
```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

# Recap of vectorizing across multiple examples




$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

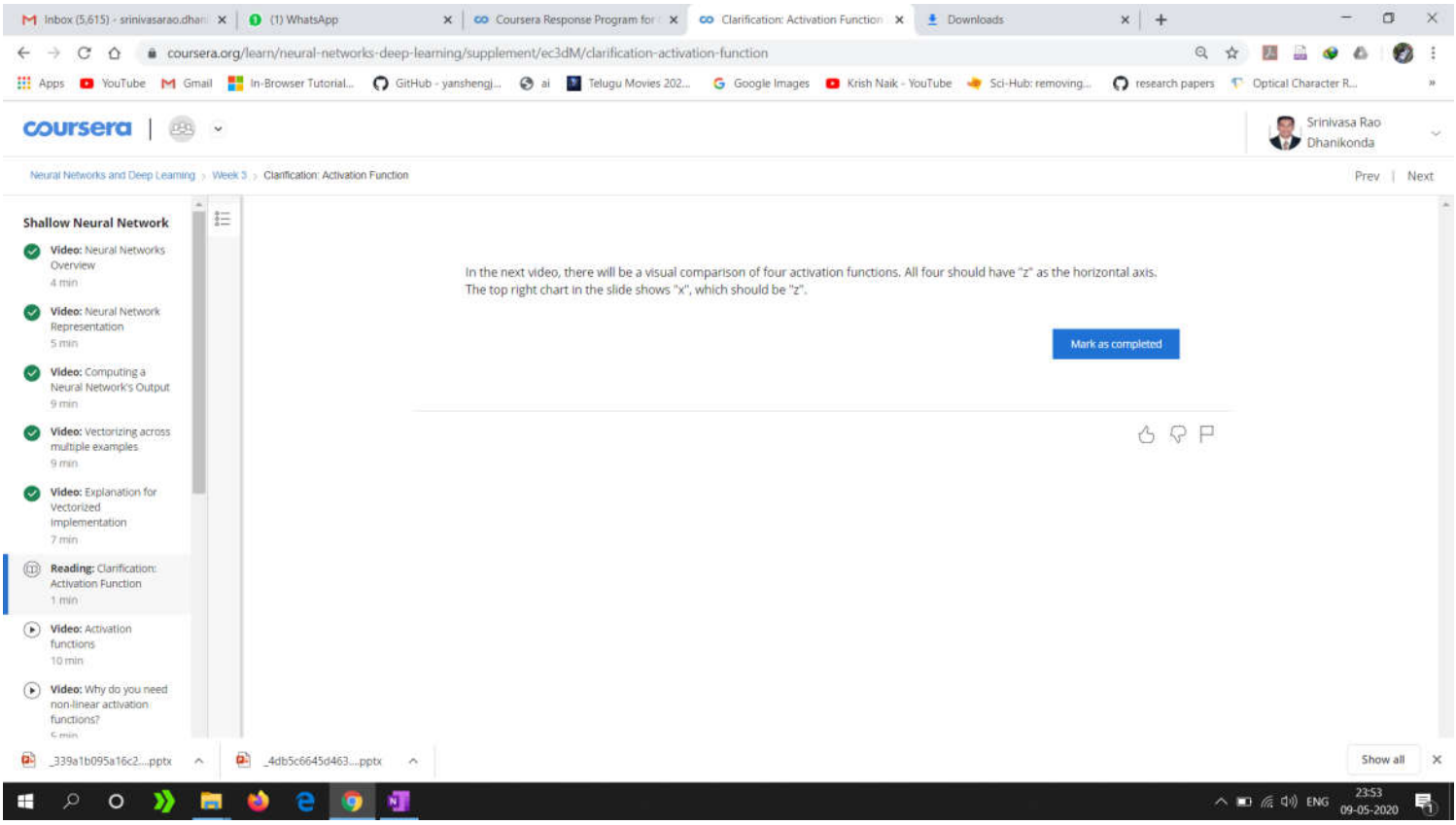
```
for i = 1 to m
```

$$\boxed{z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$A^{[0]}$ $\quad X = a^{[0]} \quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \leftarrow W^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

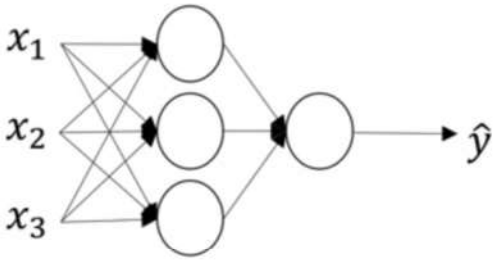



deeplearning.ai

# One hidden layer Neural Network

# Activation functions

# Activation functions

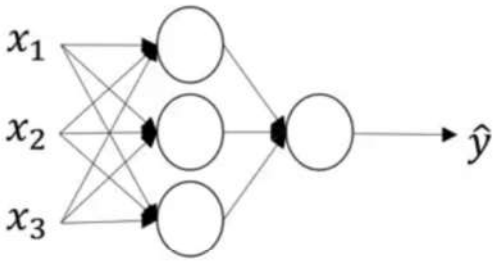



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

0:28 / 10:56 Andrew Ng

# Activation functions



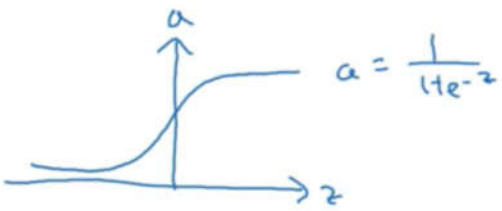

Given x:

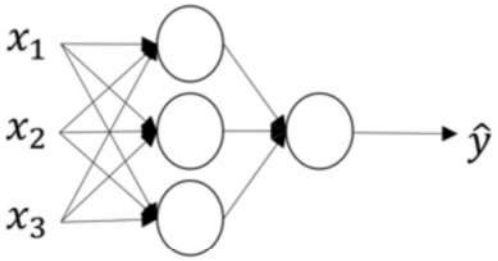
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

So that sigmoid is called an activation function.

0:31 / 10:56 Andrew Ng

# Activation functions



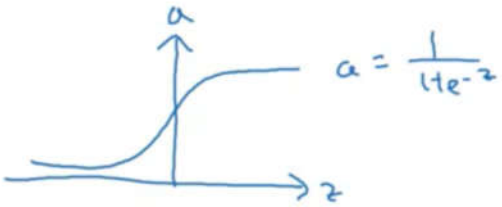

Given x:

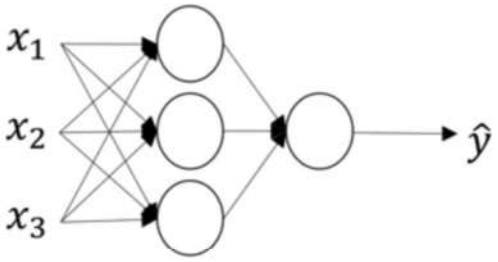
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

So in the more general case,
we can have a different function,

Andrew Ng

# Activation functions



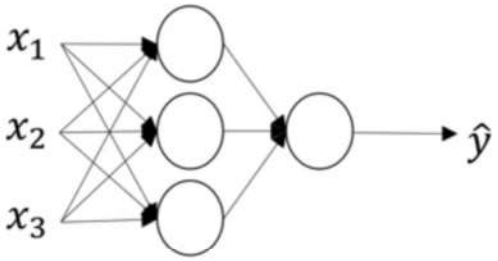

Given x:

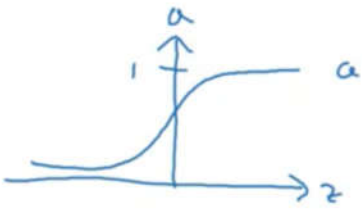
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]}) \quad g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

where g could be a nonlinear function
that may not be the sigmoid function.

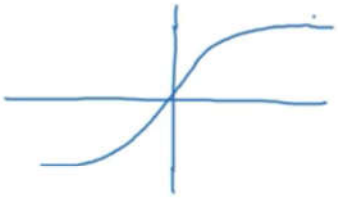
Andrew Ng

# Activation functions




$a = \frac{1}{1+e^{-z}}$



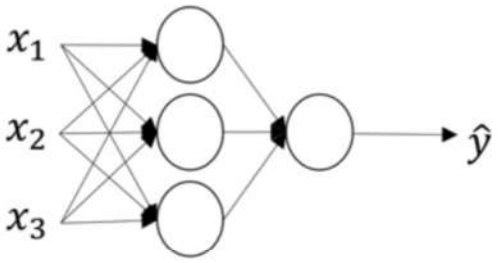
Given x:

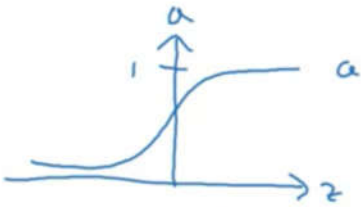
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}\ g(z^{[2]})$$

is the tangent function or
the hyperbolic tangent function.

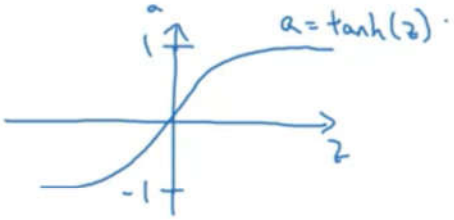
Andrew Ng

# Activation functions




$a = \frac{1}{1+e^{-z}}$




$a = \tanh(z)$

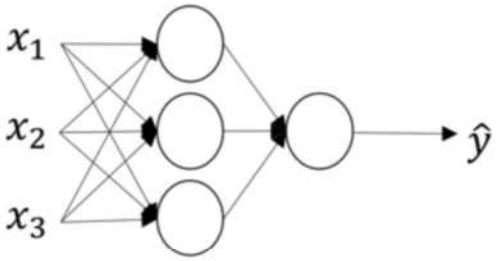
Given x:

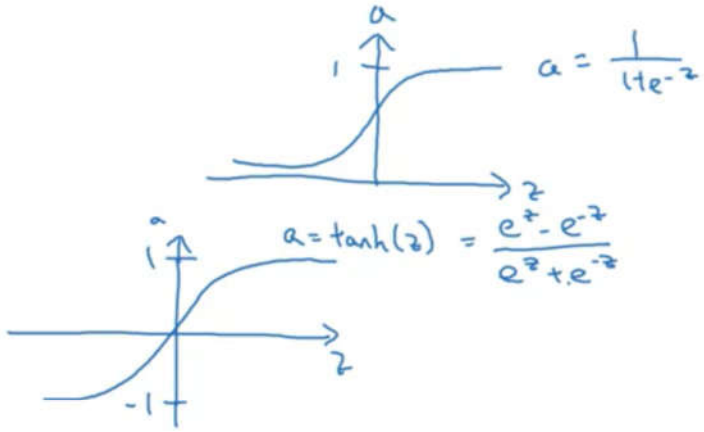
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}\ g(z^{[2]})$$

The formula for the tanh function is e to

Andrew Ng

# Activation functions



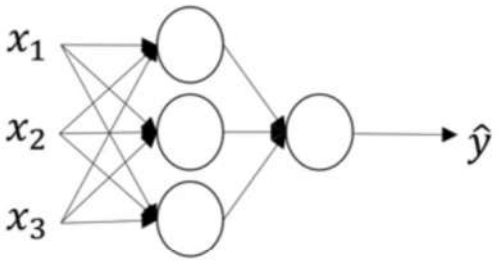

Given x:

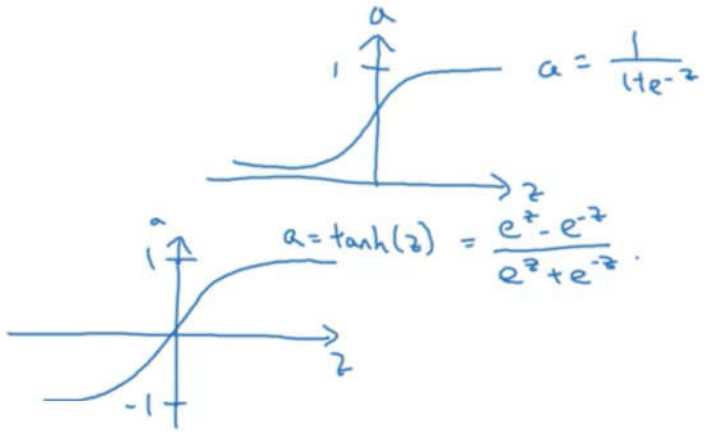
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} =$$

And is actually mathematically,
a shifted version of the sigmoid function.

Andrew Ng


# Activation functions



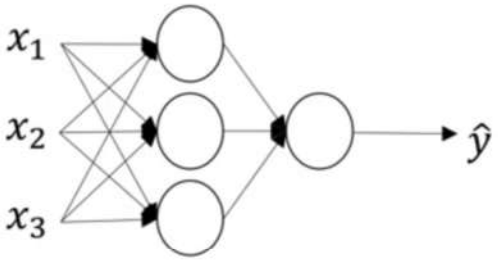
Given x:

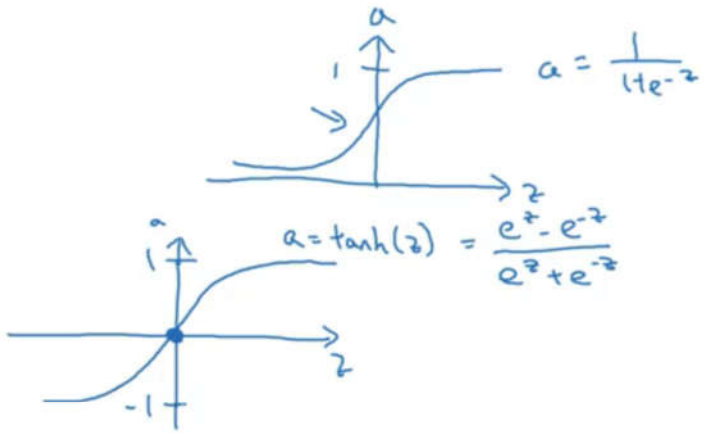
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} =$$

And is actually mathematically,
a shifted version of the sigmoid function.

Andrew Ng

# Activation functions




Given x:

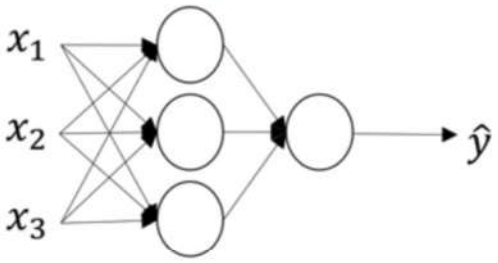
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}$$

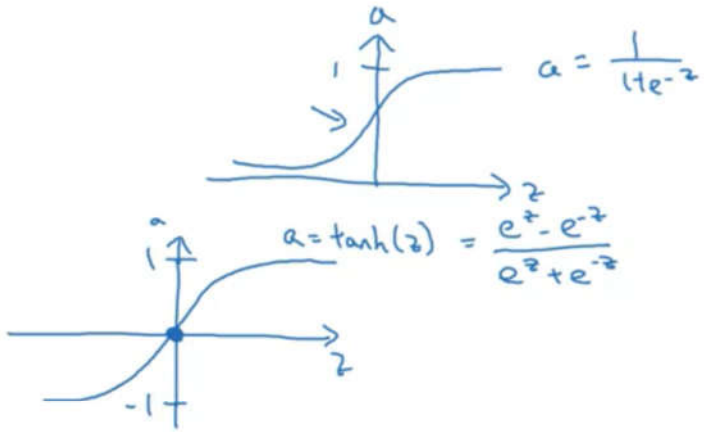
that it now crosses a zero zero point and
v scale, so it goes 15 minus 1 and plus 1.

Andrew Ng

# Activation functions

$g(z^{[1]}) = \tanh(z^{[1]})$




Given x:

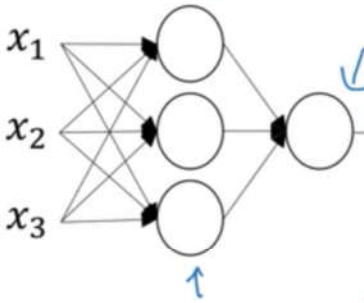
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}$$

this almost always works
better than the sigmoid

Andrew Ng

# Activation functions

$g(z^{[1]}) = \tanh(z^{[1]})$



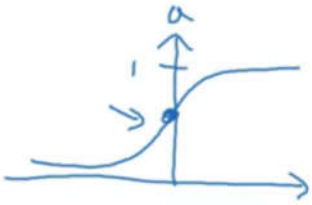

Sigmoid

$g(z^{[2]}) = \sigma(z^{[1]})$

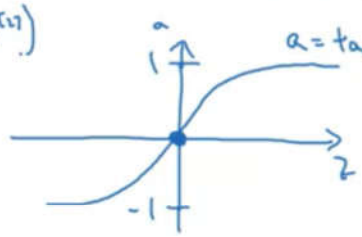
$\rightarrow y \in \{0, 1\}$

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

$-1 \quad 1$




$a = \frac{1}{1+e^{-z}}$




$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Given x:

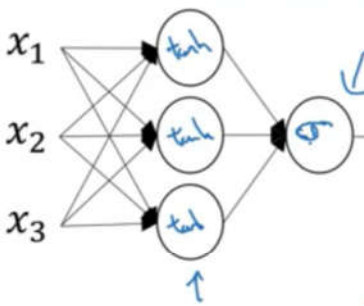
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g(z^{[2]})$$

So g of z 2 here is equal to sigma of z 2.

Andrew Ng

# Activation functions

$g(z^{[1]}) = \tanh(z^{[1]})$



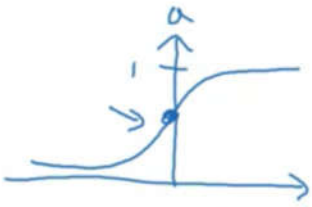

Sigmoid

$g(z^{[2]}) = \sigma(z^{[1]})$

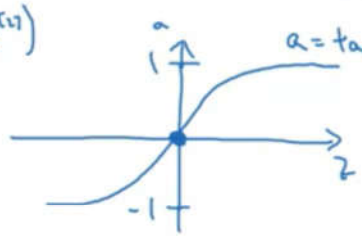
$\rightarrow y \in \{0, 1\}$

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

$-1 \quad 1$




$a = \frac{1}{1+e^{-z}}$




$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

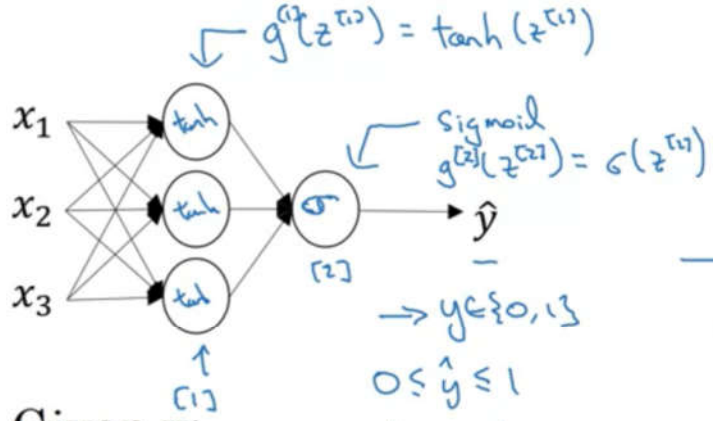
Given x:

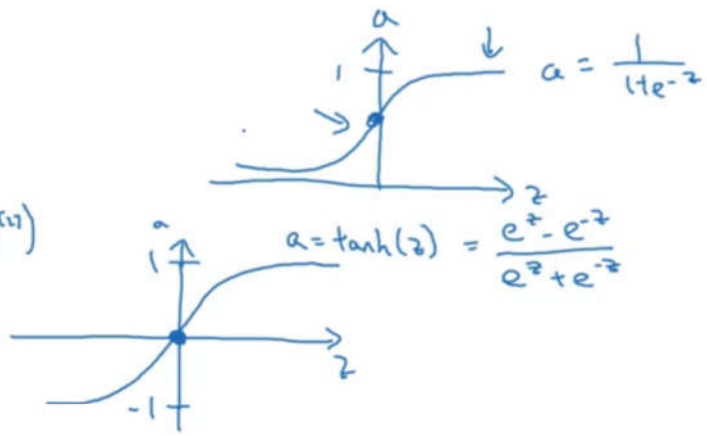
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g(z^{[2]})$$

So deactivation functions can be different for different layers.

Andrew Ng

# Activation functions



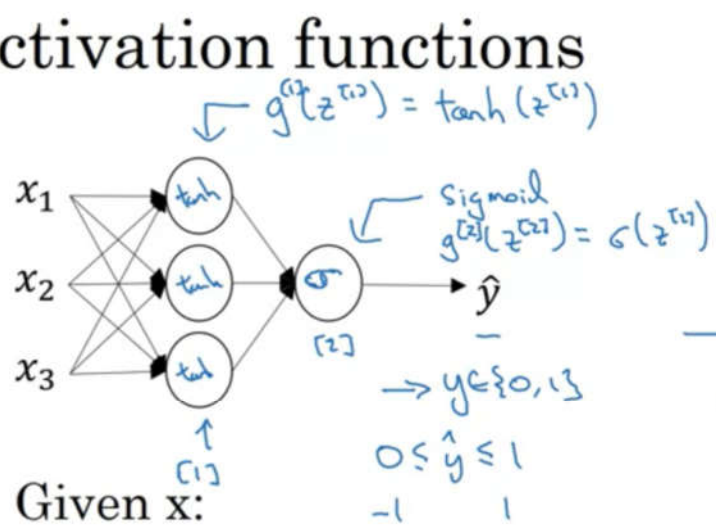

Given x:

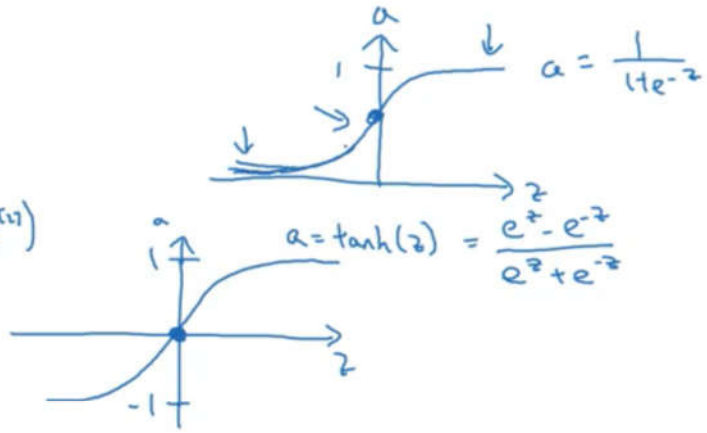
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}$$

the tanh function is that if z is either very large or very small,

Andrew Ng

# Activation functions



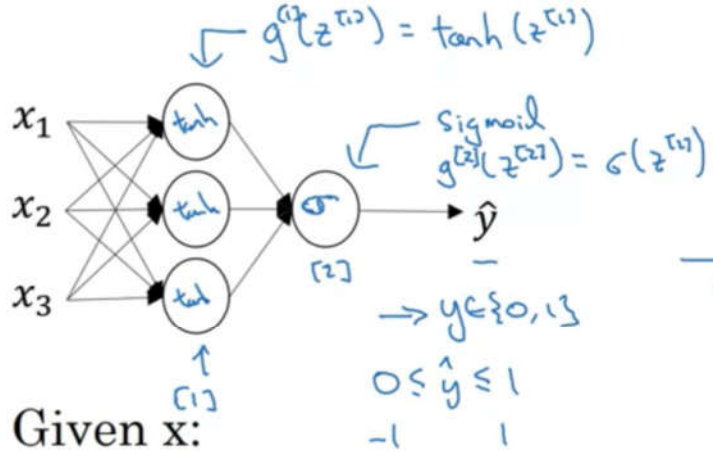

Given x:

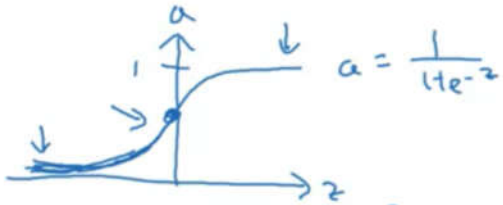
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} =$$

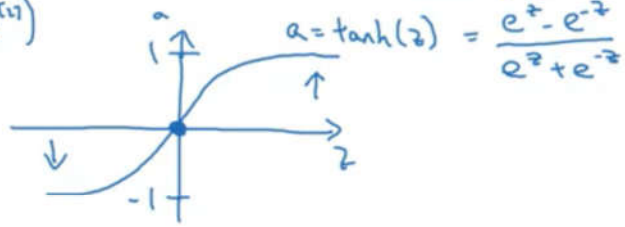
then the gradient or the derivative or the slope of this function becomes very small.

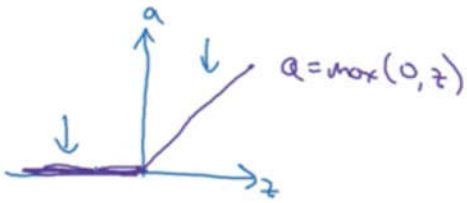
Andrew Ng

# Activation functions



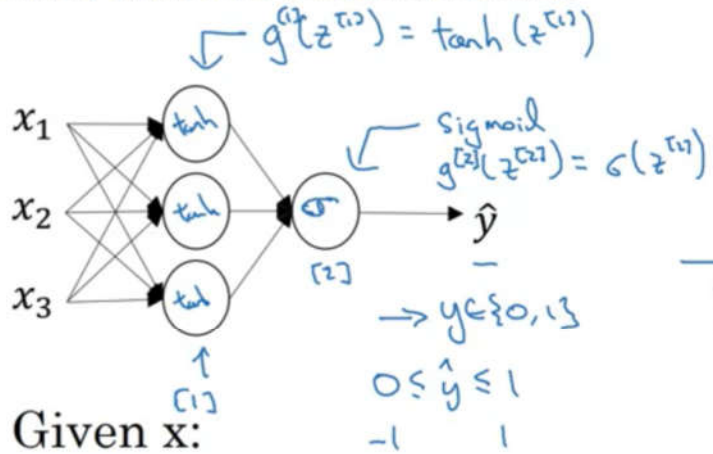

Given x:

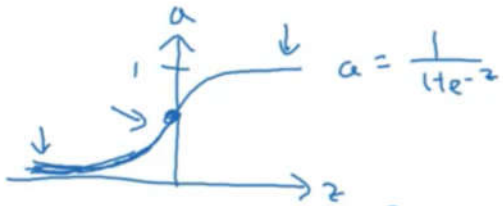
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}$$

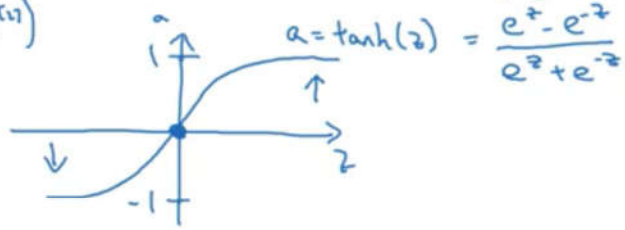
And the derivative or
the slope is 0, when z is negative.

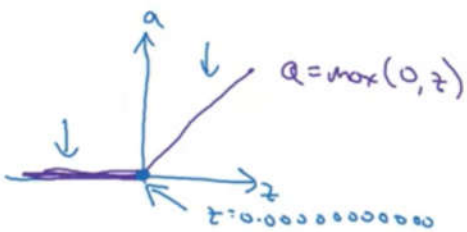
Andrew Ng

# Activation functions



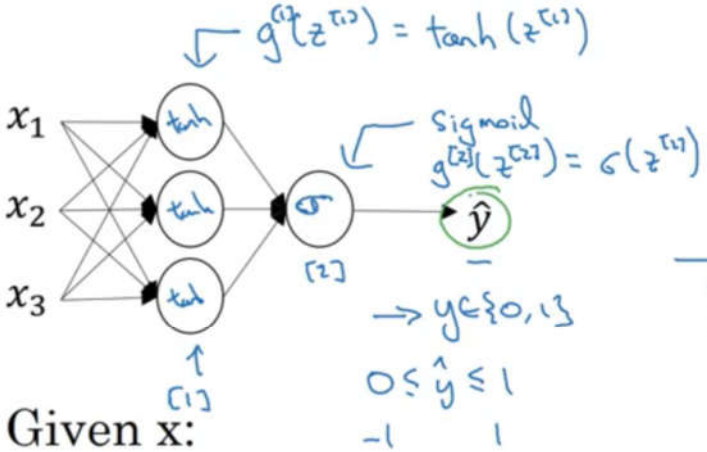

Given x:

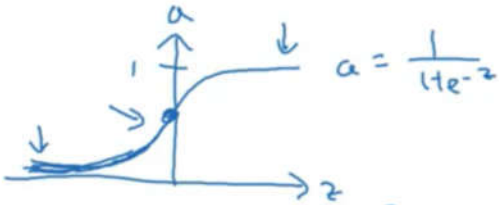
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{\sigma(z^{[2]})}$$

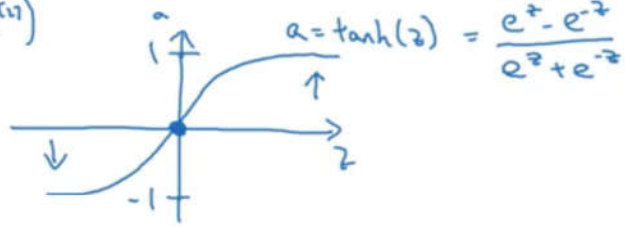
here are some rules of thumb for
choosing activation functions.

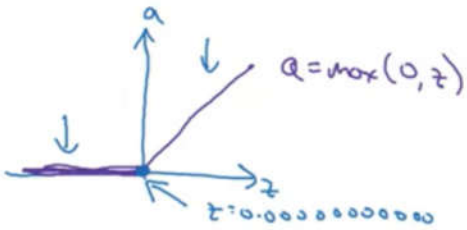
Andrew Ng

# Activation functions




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

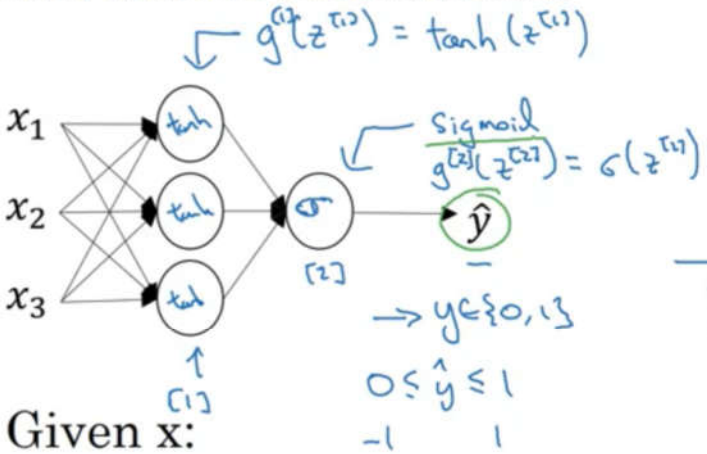
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

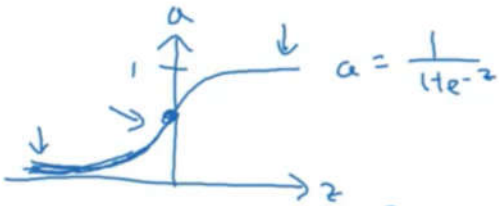
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}$$

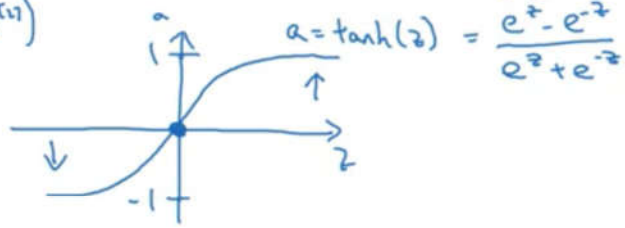
If your output is 0, 1 value,
if you're using binary classification,

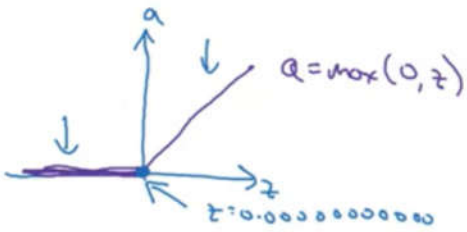
Andrew Ng

# Activation functions




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

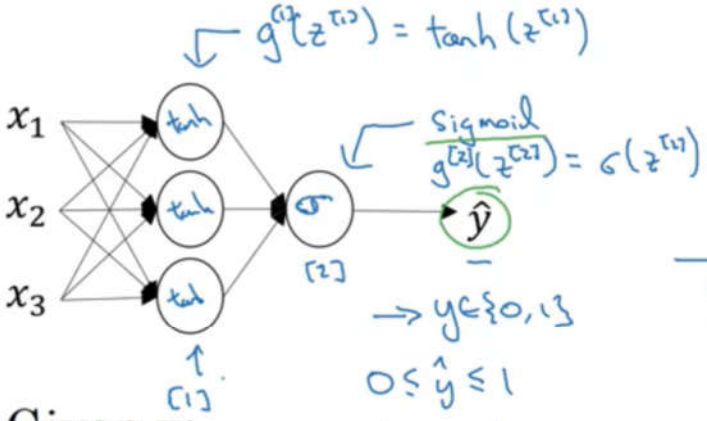
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

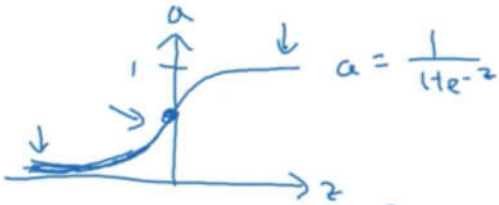
$$\rightarrow a^{[2]} =$$

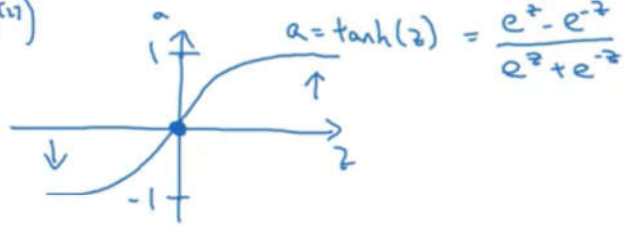
then the sigmoid activation function is a
very natural choice for the output layer.

Andrew Ng

# Activation functions



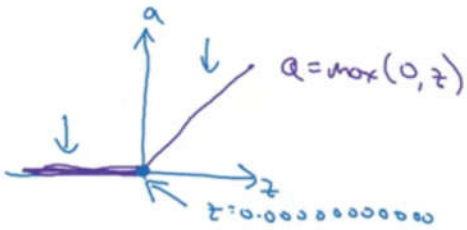

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

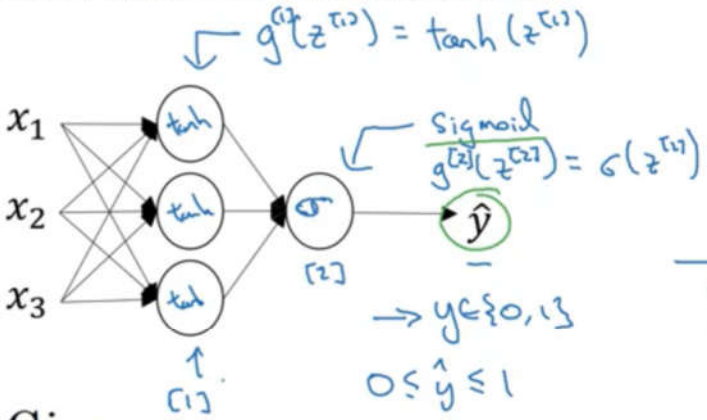
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

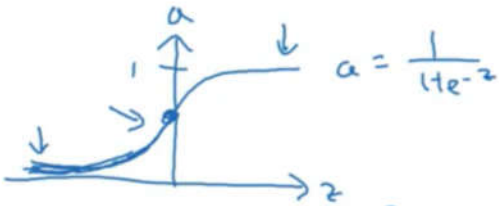


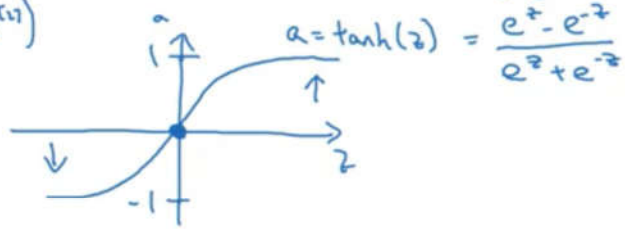

And then for all other unit's ReLU,

Andrew Ng


# Activation functions



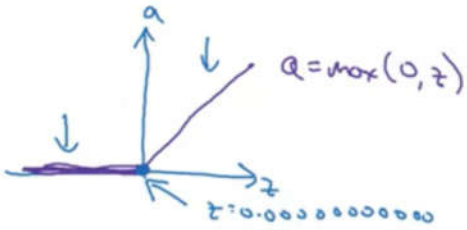

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

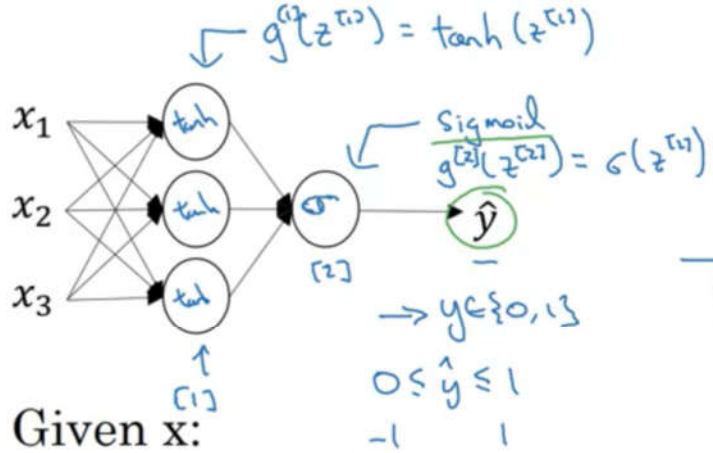
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

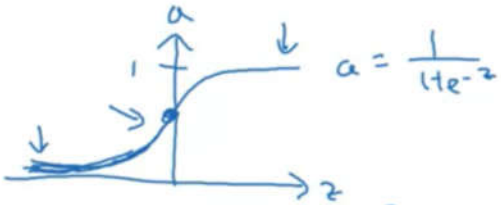


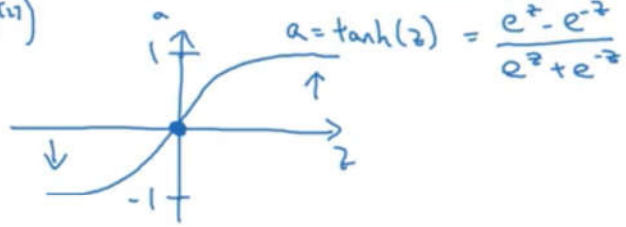

And then for all other unit's ReLU,

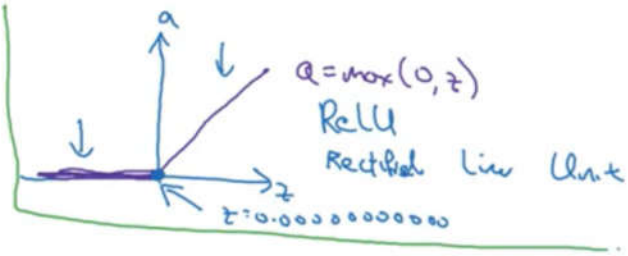
Andrew Ng

# Activation functions



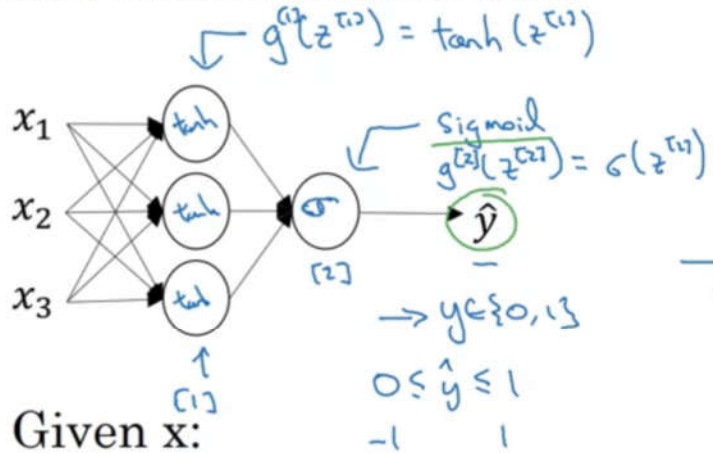

Given x:

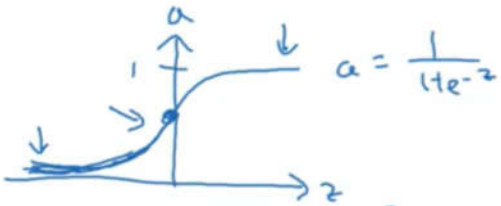
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}$$

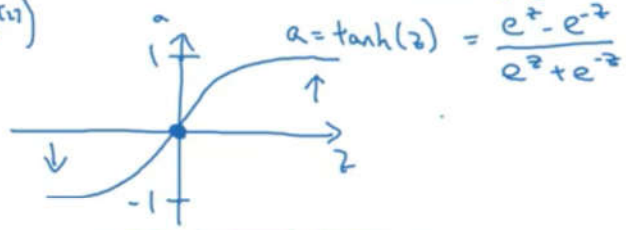
Is increasingly the default choice of activation function.

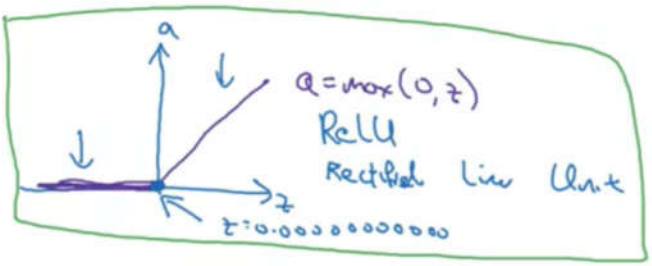
Andrew Ng

# Activation functions



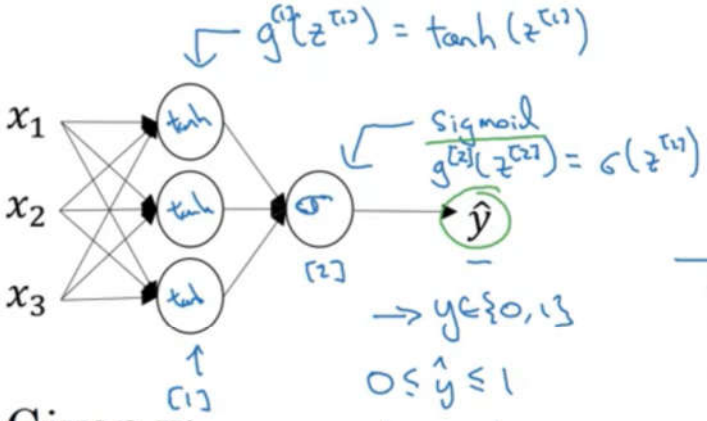

Given x:

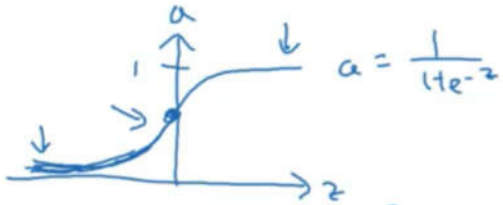
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}$$

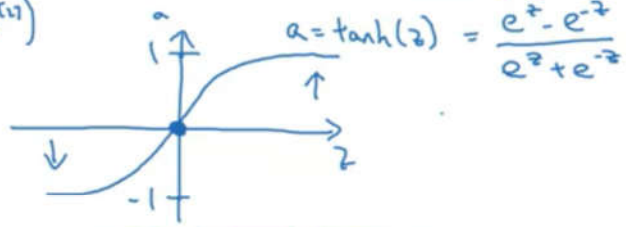
So if you're not sure what to use for your hidden layer,

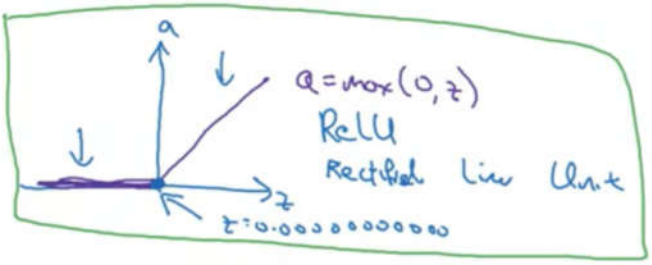
Andrew Ng

# Activation functions



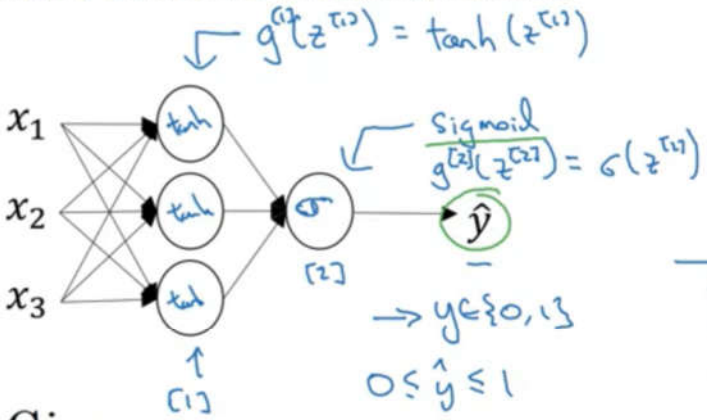

Given x:

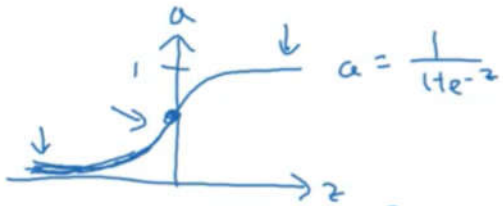
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g$$

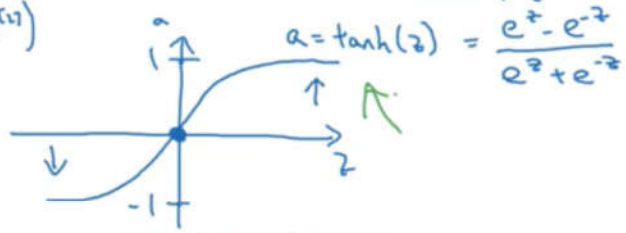
I would just use the ReLU activation function.

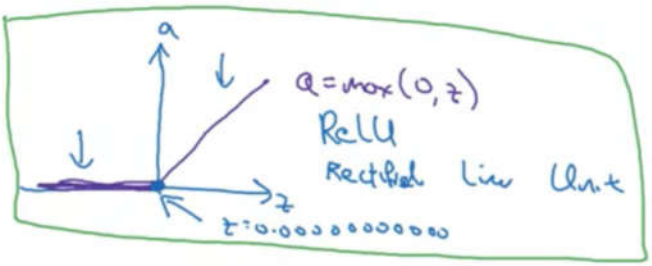
Andrew Ng

# Activation functions



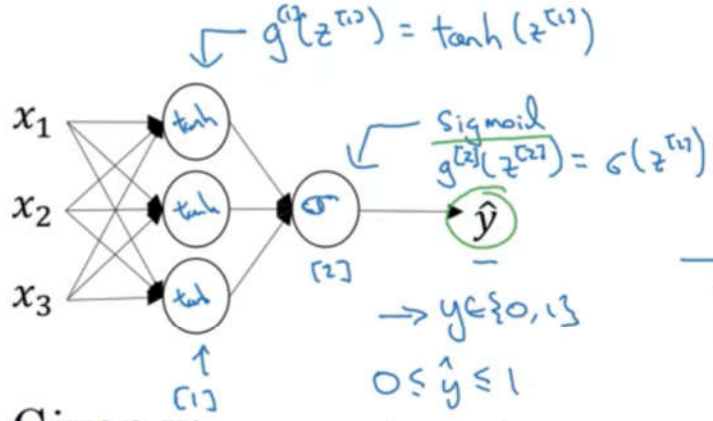

Given x:

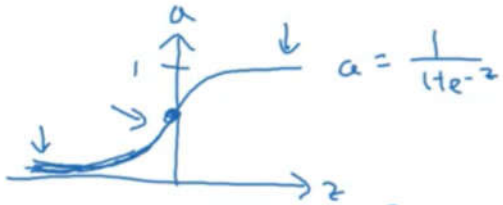
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z}$$

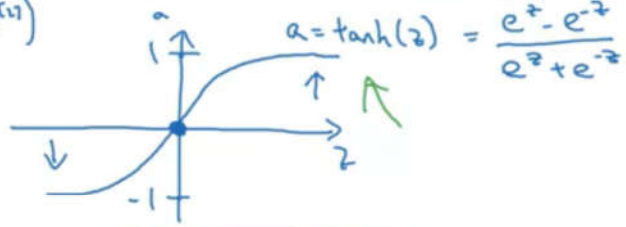
Although sometimes people also use the tanh activation function.

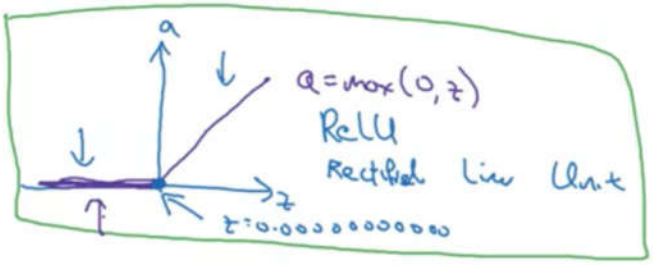
Andrew Ng

# Activation functions



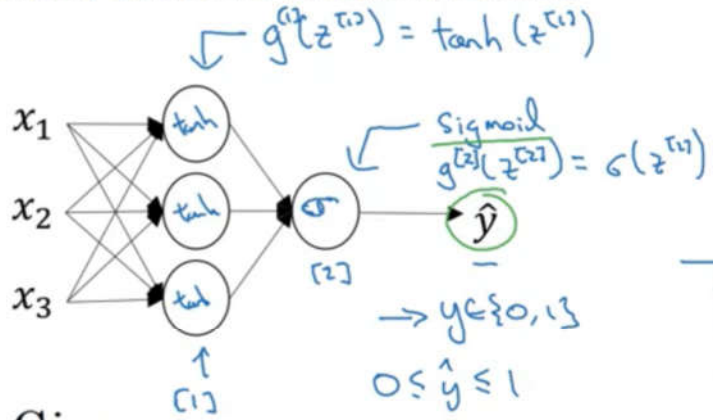

Given x:

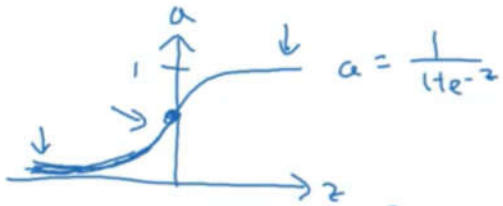
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

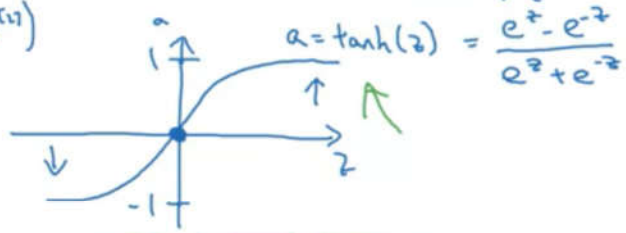
One disadvantage of the ReLU is that the derivative is equal to zero,

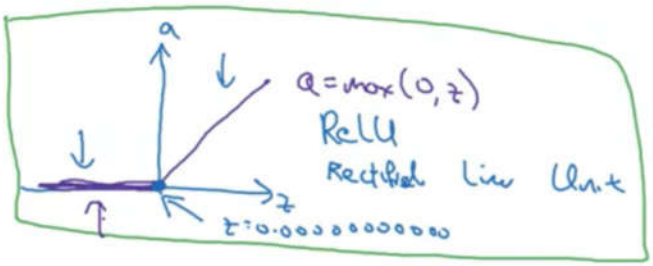
Andrew Ng


# Activation functions



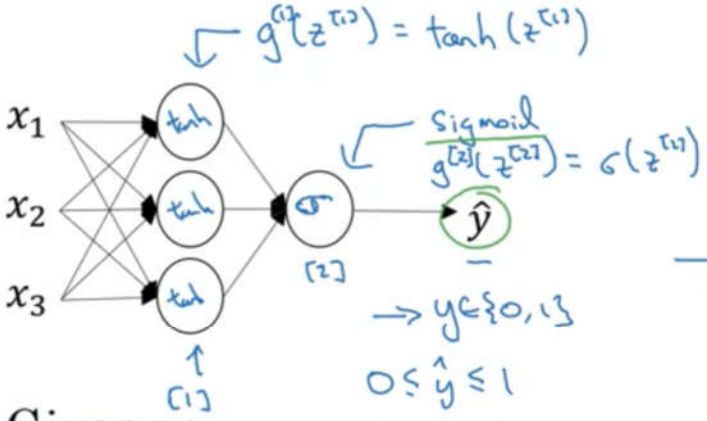
Given x:

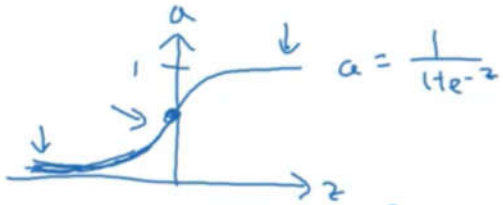
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

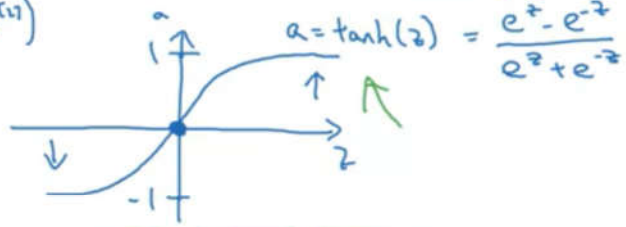

when z is negative.

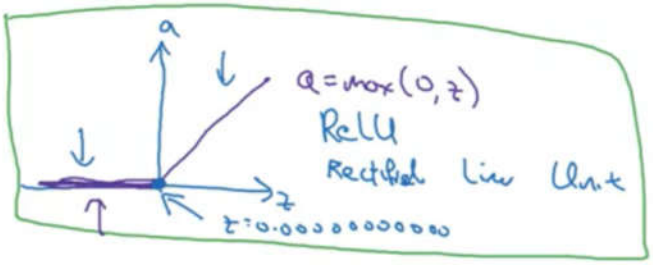

Andrew Ng

# Activation functions




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

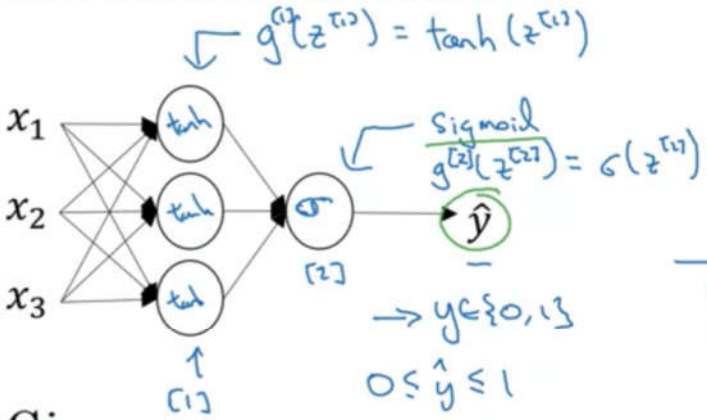
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

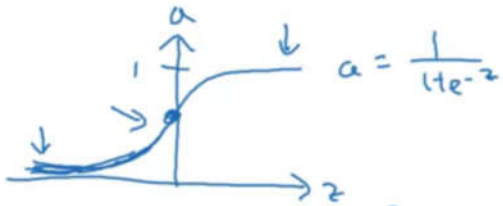
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

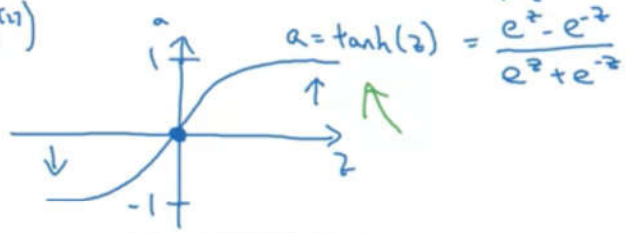
But there is another version of the ReLU called the leaky ReLU.

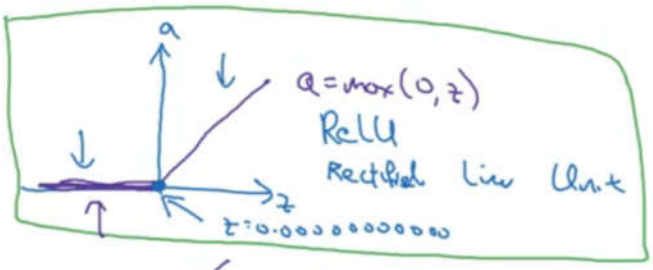
Andrew Ng

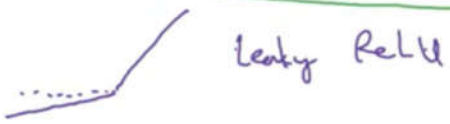
# Activation functions




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

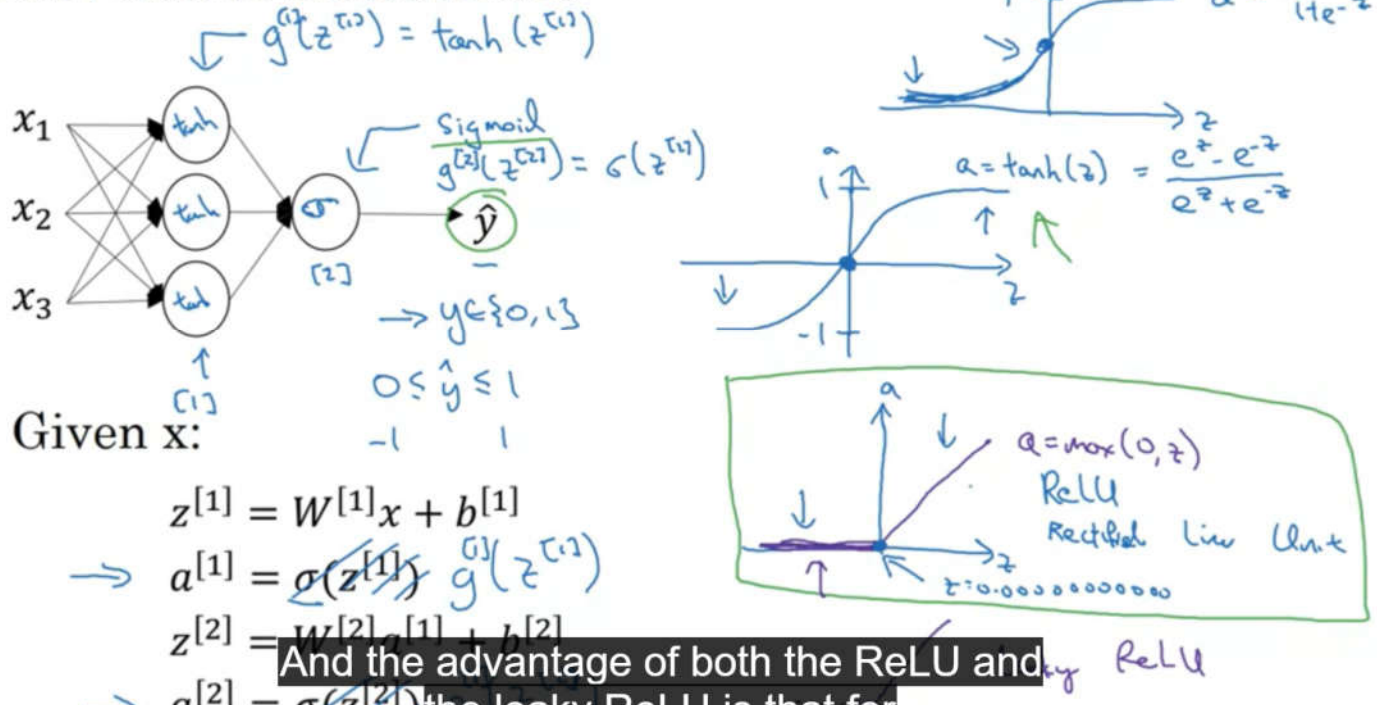
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$

Andrew Ng

# Activation functions



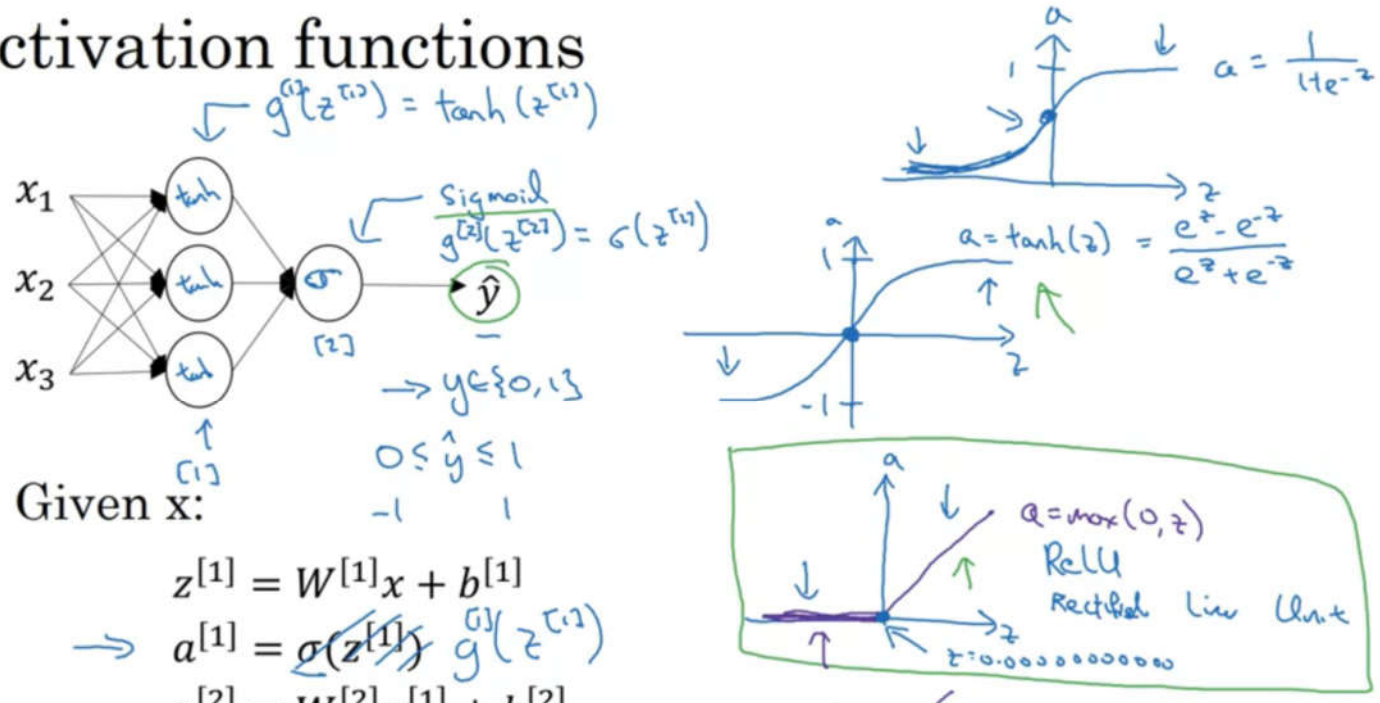

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}$$

And the advantage of both the ReLU and the leaky ReLU is that for

Andrew Ng

# Activation functions



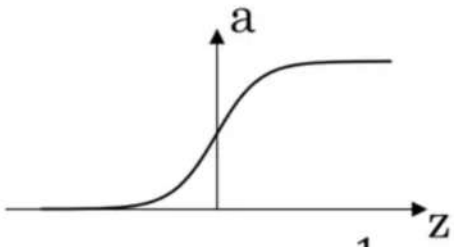

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}$$

And so in practice, using the ReLU activation function,

Andrew Ng

# Pros and cons of activation functions

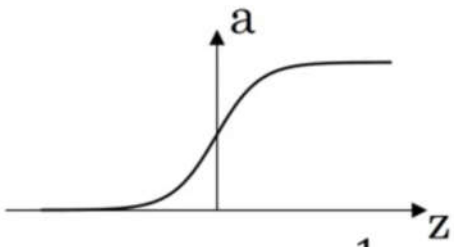



sigmoid: $a = \frac{1}{1+e^{-z}}$

So let's just quickly recap the pros and cons of different activation functions.

Andrew Ng

# Pros and cons of activation functions

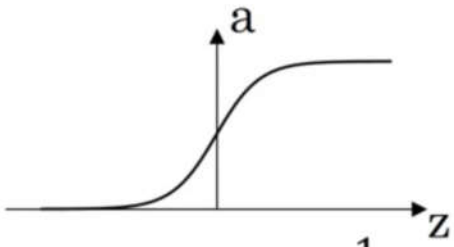



sigmoid: $a = \frac{1}{1+e^{-z}}$

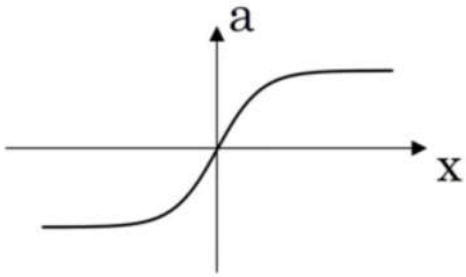
if you are doing binary classification, or maybe almost never use this.

Andrew Ng

# Pros and cons of activation functions



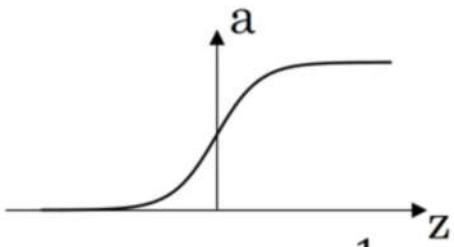

sigmoid: $a = \dfrac{1}{1+e^{-z}}$



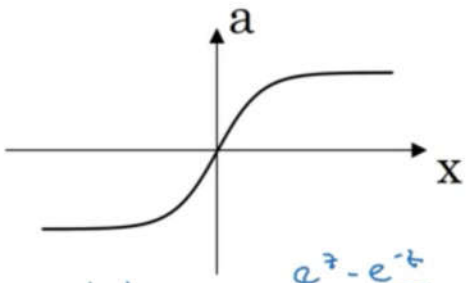

And the reason I almost never use this is because the tanh

Andrew Ng

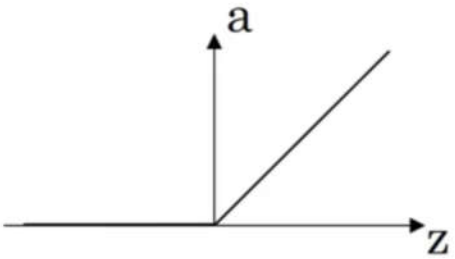
# Pros and cons of activation functions




sigmoid: $a = \dfrac{1}{1+e^{-z}}$



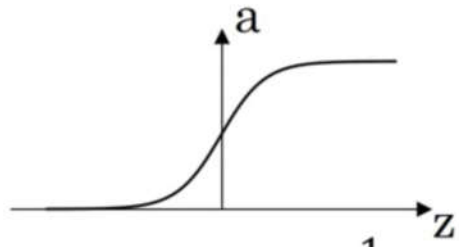

tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



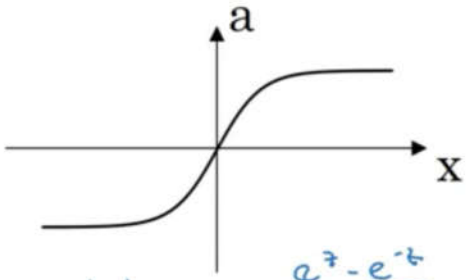

And then the default,

Andrew Ng

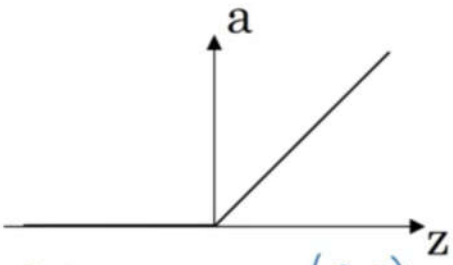
# Pros and cons of activation functions



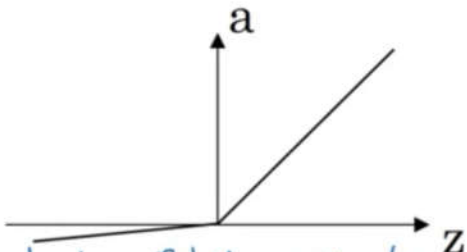

sigmoid: $a = \dfrac{1}{1+e^{-z}}$




tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



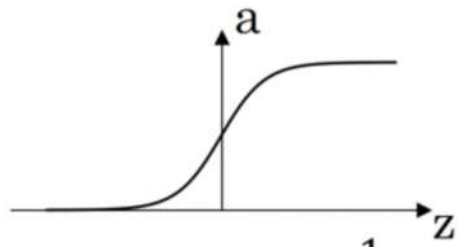

ReLU $a = \max(0, z)$



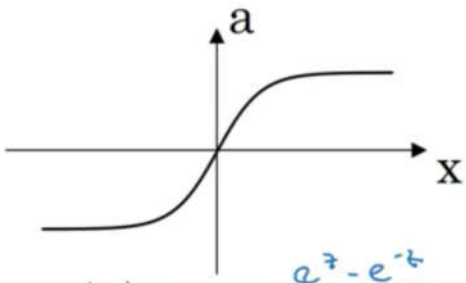

Leaky ReLU $a = \max(0.01z, z)$

Andrew Ng

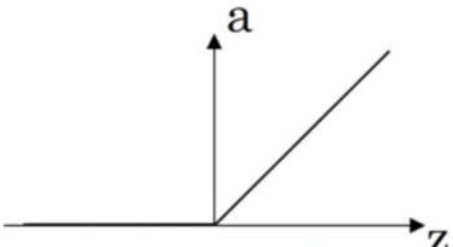
# Pros and cons of activation functions




sigmoid: $a = \dfrac{1}{1+e^{-z}}$

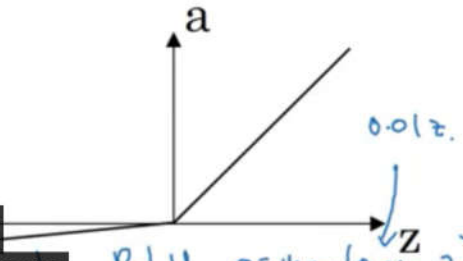



tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$




ReLU $a = \max(0, z)$

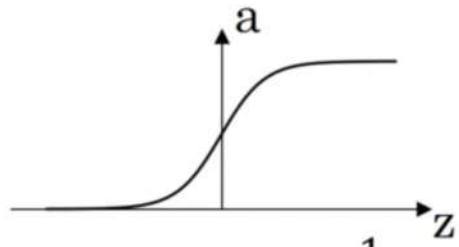
And you might say,
why is that constant 0.01?



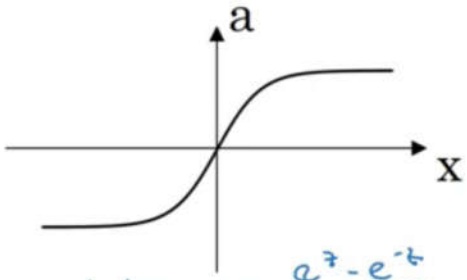

Leaky ReLU $a = \max(0.01z, z)$

Andrew Ng

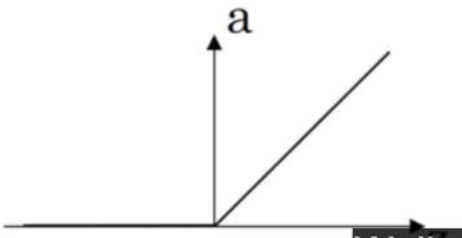
# Pros and cons of activation functions



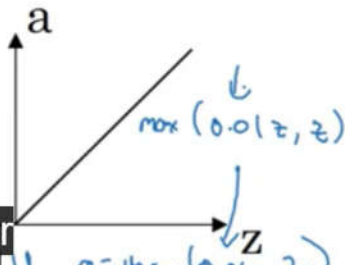

sigmoid: $a = \frac{1}{1+e^{-z}}$




tanh: $a = \frac{e^{z}-e^{-z}}{e^{z}+e^{z}}$




ReLU $a = \max(0, z)$




$\max(0.01z, z)$

Leaky ReLU $a = \max(0.01z, z)$

Well, you can also make that another parameter of the learning algorithm.

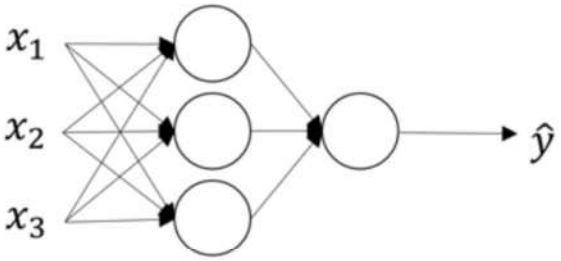
Andrew Ng

deeplearning.ai

# One hidden layer Neural Network

# Why do you need non-linear activation functions?

# Activation function



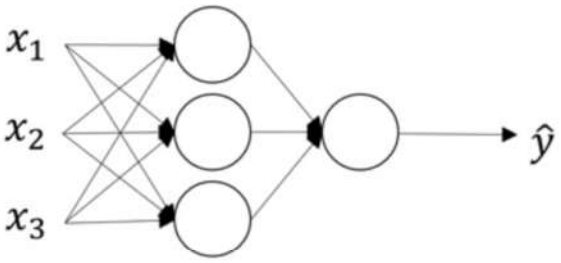

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

So, here's the four prop equations for the neural network.

Andrew Ng

# Activation function



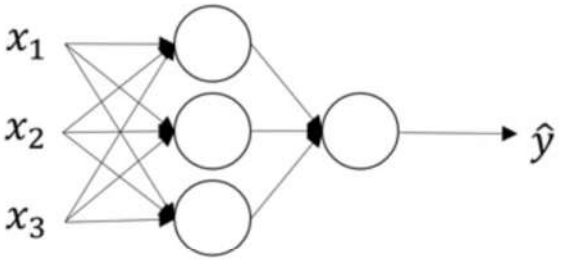

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})} \; z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

And set a1 equals z1.

Andrew Ng

# Activation function




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

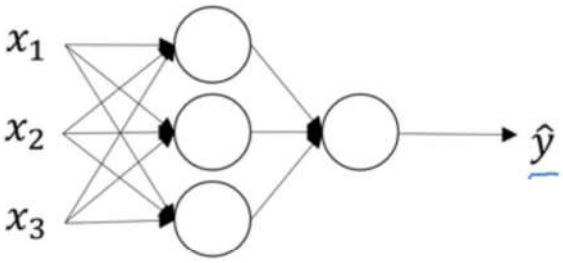
$g(z) = z$

"linear activation function"

Maybe a better name for it would be the identity activation function

Andrew Ng

# Activation function




Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}\ z^{[2]}$$

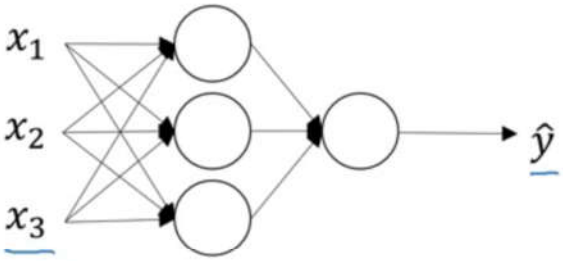
$g(z) = z$

"linear activation function"

0:49 / 5:35

Andrew Ng

# Activation function




$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}\ z$$

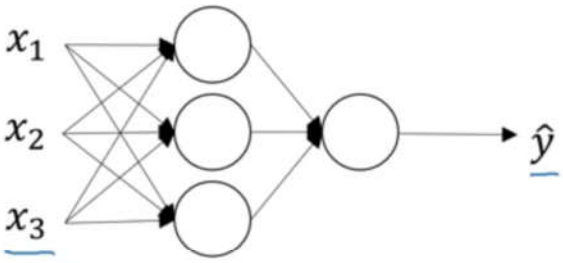
$g(z) = z$

"linear activation

Then if you take this definition of a1 and

Andrew Ng

# Activation function




$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}\underbrace{(W^{[1]}x + b^{[1]})}_{a^{[1]}} + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}\ z$$

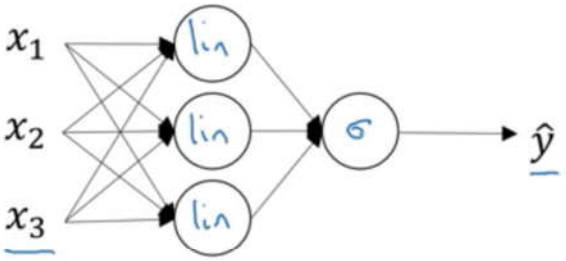
$g(z) = z$

"linear activation

SO this is just equal to w' x + b'.

Andrew Ng

# Activation function




$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$

$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$

$$= W'x + b'$$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

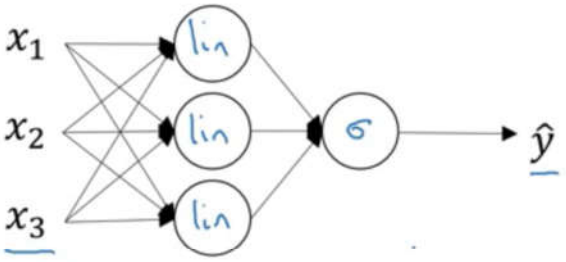
$g(z) = z$

"linear activation

a linear activation function here and a sigmoid function here, then this model is

Andrew Ng

# Activation function




$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$

$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$

$$= W'x + b'$$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

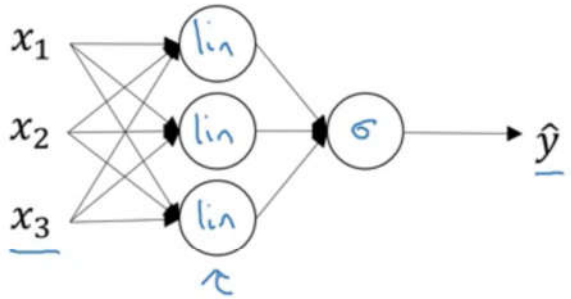
$g(z) = z$

"linear activation

no more expressive than standard logistic regression without any hidden layer.

Andrew Ng

# Activation function




$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$

$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$

$$= W'x + b'$$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

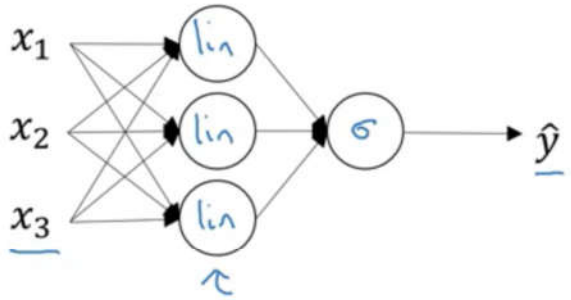
$g(z) = z$

"linear activation

But the take home is that a linear hidden layer is more or less useless

Andrew Ng


# Activation function



$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$

$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$

$$= W'x + b'$$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

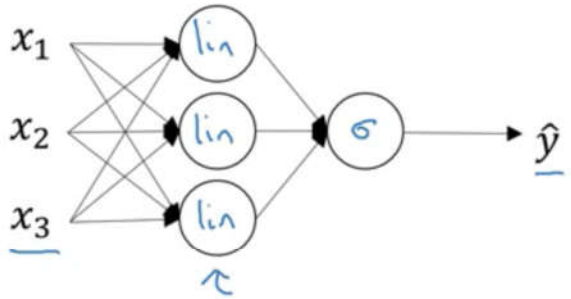
$g(z) = z$

"linear activation


because the composition of two linear functions is itself a linear function.


Andrew Ng

# Activation function




$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$
$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$
$$= W'x + b'$$

Given x:

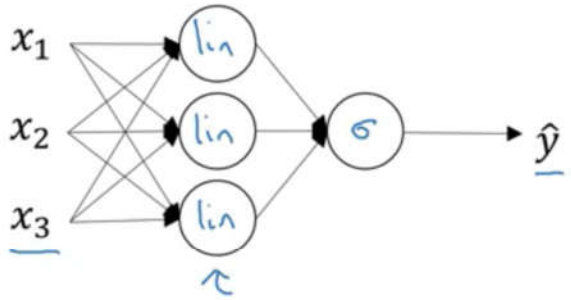
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$
"linear activation

So unless you throw a non-linear [INAUDIBLE] in there, then you're not computing more

Andrew Ng


# Activation function



$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$
$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$
$$= W'x + b'$$

Given x:

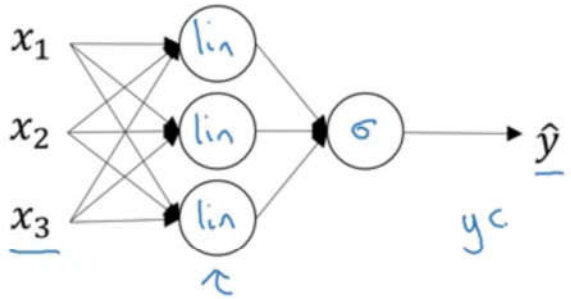
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$
"linear activation


There is just one place where you might use a linear activation function.


Andrew Ng

# Activation function




$y \in$

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})} \quad z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})} \quad z^{[2]}$$

$g(z) = z$

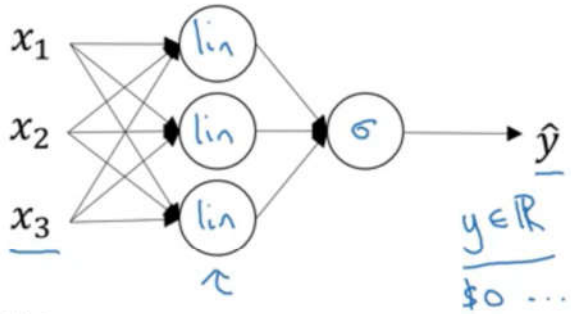
"linear activation"

$g(z) = z$

So if y is a real number.

Andrew Ng

# Activation function




$y \in \mathbb{R}$

\$0 ..... \$

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})} \quad z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$

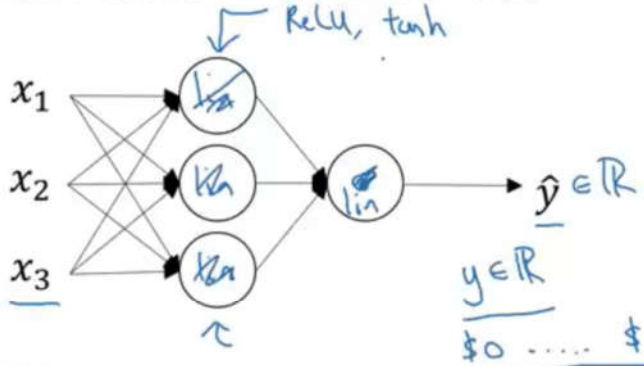
"linear activation"

$g(z) = z$

\$0 is the price of house up to however expensive, right, houses get, I guess.

Andrew Ng

# Activation function




$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}\ z^{[2]}$

$g(z) = z$

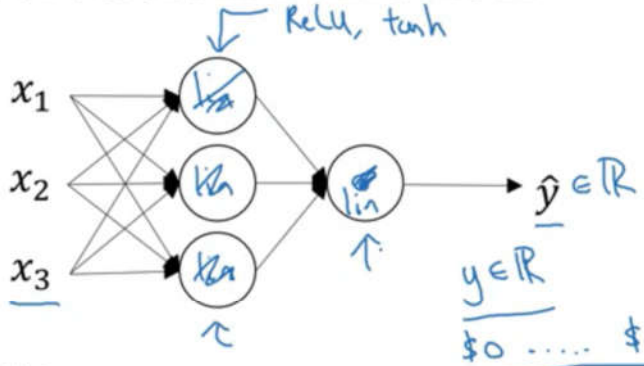
"linear activation"

$g(z) = z$

They could use ReLU or tanh or
Leaky ReLU or maybe something else.

Andrew Ng

# Activation function




$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}\ z^{[2]}$

$g(z) = z$

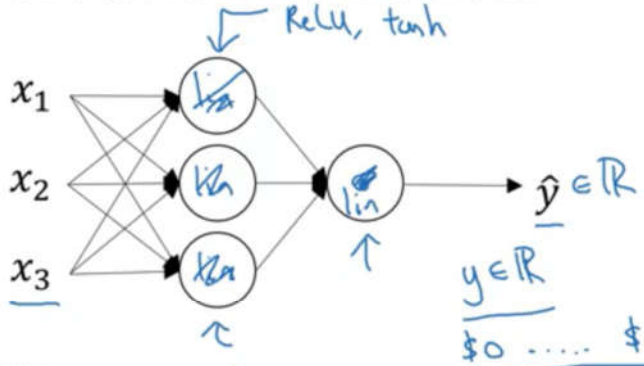
"linear activation"

$g(z) = z$

is usually in the output layer.

Andrew Ng

# Activation function




$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$

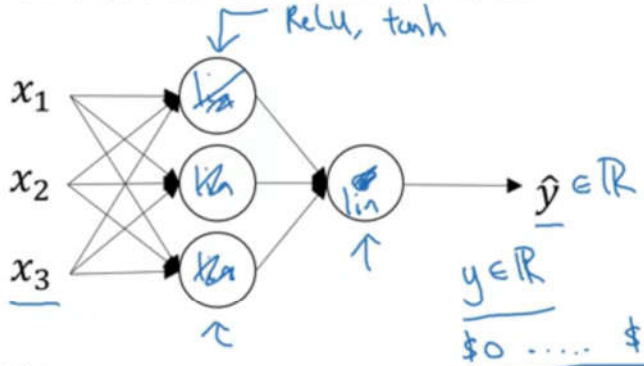
"linear activation

$g(z) = z$

But other than that, using a linear activation function in the hidden layer

Andrew Ng


# Activation function




$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$

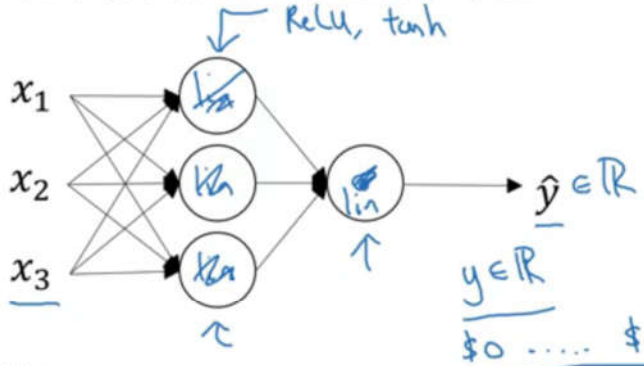
"linear activation

$g(z) = z$


except for some very special circumstances relating to compression that we're

Andrew Ng

# Activation function




$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

$= W'x + b'$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})}$$

$g(z) = z$

"linear activation

$g(z) = z$

going to talk about using the linear activation function is extremely rare.

Andrew Ng

deeplearning.ai

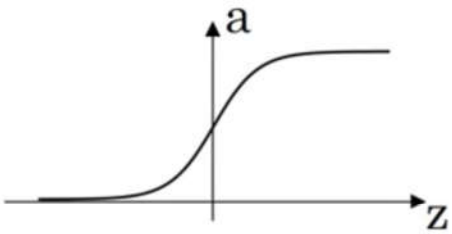
# One hidden layer Neural Network

# Derivatives of activation functions

When you implement back propagation for your neural network,

0:00 / 7:57

# Sigmoid activation function

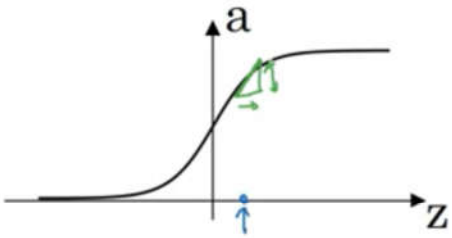



$$g(z) = \frac{1}{1+e^{-z}}$$

Here's the familiar Sigmoid activation function.

Andrew Ng

# Sigmoid activation function

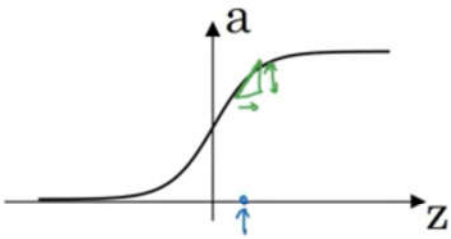



$$g(z) = \frac{1}{1+e^{-z}}$$

the height over width of this lower triangle here.

Andrew Ng

# Sigmoid activation function




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

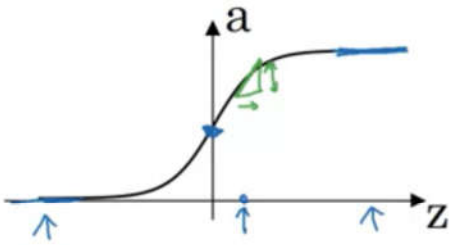
$\frac{d}{dz} g(z)$ = slope of $g(x)$ at $z$

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

$= g(z)\left(1 - g(z)\right)$

Andrew Ng

# Sigmoid activation function




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

$\frac{d}{dz} g(z)$ = slope of $g(x)$ at $z$

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

$= g(z)\left(1 - g(z)\right) \leftarrow$

$z = 10.$ $g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10$ $g(z) \approx 0$

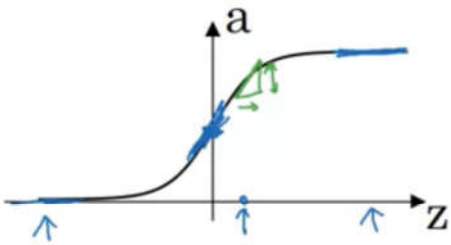
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0$ $g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

Andrew Ng

# Sigmoid activation function




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

$$g'(z) = \boxed{\frac{d}{dz} g(z)} = \text{slope of } g(x) \text{ at } z$$

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

$$= g(z)\left(1 - g(z)\right) \leftarrow$$

So, g prime of z in calculus,

$z = 10.\quad g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10\quad g(z) \approx 0$

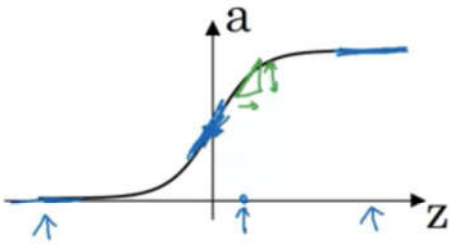
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0\quad g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

Andrew Ng

# Sigmoid activation function




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

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

$$g'(z) = \boxed{\frac{d}{dz} g(z)} = \text{slope of } g(x) \text{ at } z$$

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

$$= \underbrace{g(z)\left(1 - g(z)\right)}_{= a\,(1-a)} \leftarrow \quad \Big|\; g'(z) = a(1-a)$$

$z = 10.\quad g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10\quad g(z) \approx 0$

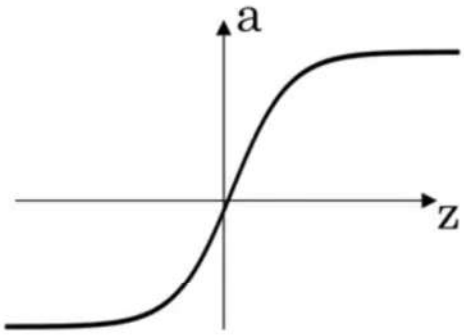
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0\quad g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

Andrew Ng

# Tanh activation function

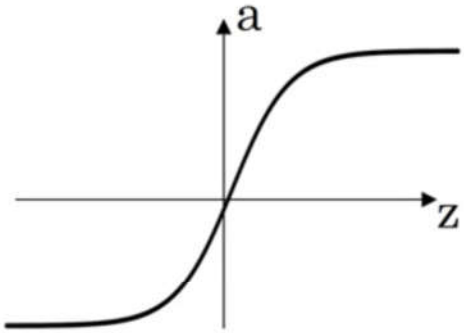



$$g(z) = \tanh(z)$$

Let's now look at the Tanh activation function.

Andrew Ng

# Tanh activation function




$$g(z) = \tanh(z)$$

$$= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

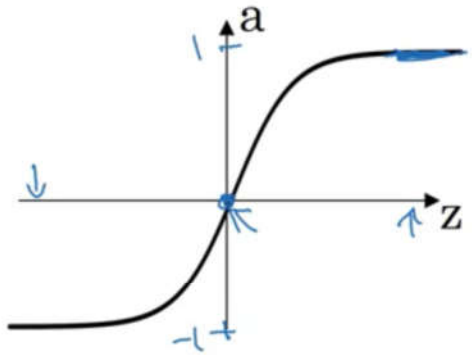
$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= 1 - (\tanh(z))^2$$

the shorthand we have previously when we call this g prime of z again.

Andrew Ng

# Tanh activation function




$$g(z) = \tanh(z)$$

$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= 1 - (\tanh(z))^2$$

$a = g(z)$, $g'(z) = 1 - a^2$

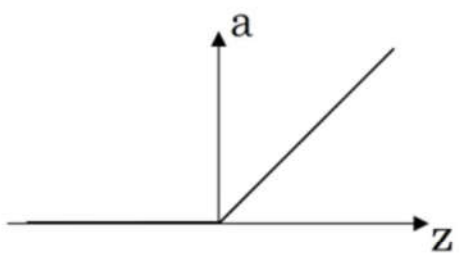
$z = 10$ $\tanh(z) \approx 1$
$g'(z) \approx 0$

$z = -10$ $\tanh(z) \approx -1$
$g'(z) \approx 0$

$z = 0$ $\tanh(z) = 0$
$g'(z) = 1$

Andrew Ng

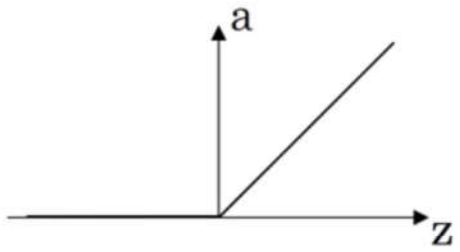
# ReLU and Leaky ReLU




ReLU

Andrew Ng

# ReLU and Leaky ReLU




ReLU

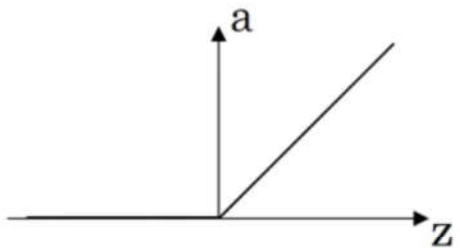
$g(z) = \max(0, z)$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \cancel{\text{undef}} & \cancel{\text{if } z = 0} \end{cases}$$

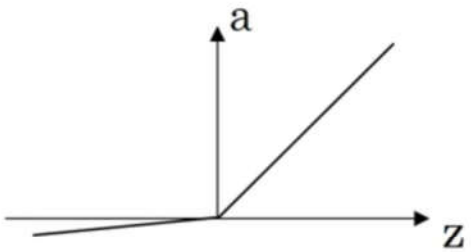
$z = 0.0000\ldots 0$

Andrew Ng

# ReLU and Leaky ReLU




ReLU




Leaky ReLU

$g(z) = \max(0, z)$

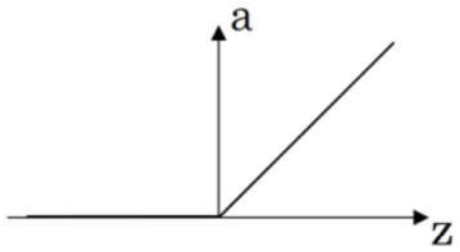
$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \cancel{\text{undef}} & \cancel{\text{if } z = 0} \end{cases}$$

$z = 0.0000\ldots 0$

7:01 / 7:57

Andrew Ng

# ReLU and Leaky ReLU

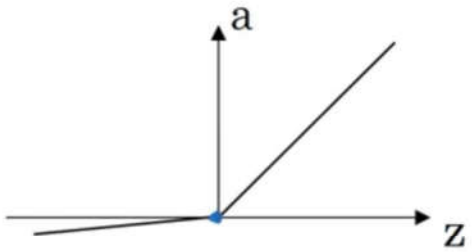



ReLU

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

$z = 0.0000\ldots0$




Leaky ReLU

$g(z) = \max(0.01z, z)$

$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$

Andrew Ng

Press Esc to exit full screen

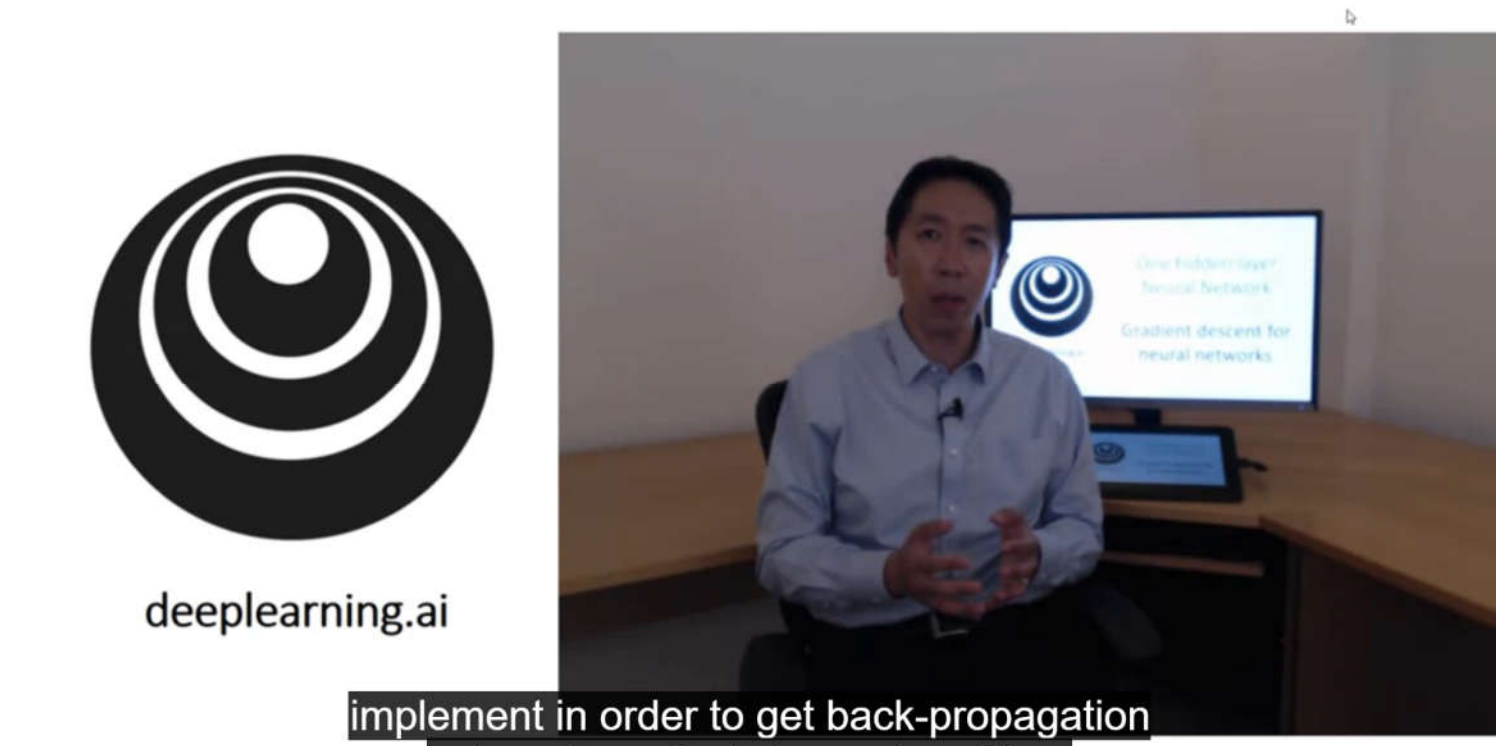
deeplearning.ai

# One hidden layer Neural Network

# Gradient descent for neural networks

All right. I think this'll be an exciting video.

0:01 / 9:57




# Gradient descent for neural networks

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$n_x = n^{[0]}$, $n^{[1]}$, $\underline{n^{[2]} = 1}$

Cost function: $J(W^{[1]}, b^{[1]}, \underline{W^{[2]}}, \underline{b^{[2]}}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$

$\hat{y} \leftarrow a^{[2]}$

Gradient descent:

Repeat {

Compute predicts $(\hat{y}^{(i)}, i = 1, \ldots, m)$

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$

$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$

$W^{[2]} := \ldots$, $b^{[2]} := \ldots$

# Gradient descent for neural networks

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

$(n^{[1]}, n^{[0]}) \quad (n^{[1]}, 1) \quad (n^{[2]}, n^{[1]}) \quad (n^{[2]}, 1)$

$n_x = n^{[0]}, \quad n^{[1]}, \quad \underline{n^{[2]} = 1}$

Cost function: $J(W^{[1]}, b^{[1]}, \underline{W^{[2]}}, \underline{b^{[2]}}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$

$\hat{y} \leftarrow a^{[2]}$

Gradient descent:

→ Repeat {

→ Compute predicts $(\hat{y}^{(i)}, i = 1, \ldots, m)$

$\underline{dW^{[1]}} = \frac{\partial J}{\partial W^{[1]}}, \quad \underline{db^{[1]}} = \frac{\partial J}{\partial b^{[1]}}, \ldots$

$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$

$b^{[1]} := b^{[1]}$

the DW1, DB1 as well as the derivatives DW2 and DB2.

# Formulas for computing derivatives

Forward propagation:

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

5:16 / 9:57

# Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \underline{np.sum}(dZ^{[2]}, \underline{axis=1}, \underline{keepdims=True})$$

$$Y = [y^{(1)}\ y^{(2)} \cdots, y^{(m)}]$$

$(n^{[1]}, )$

$\downarrow (n^{[2]}, 1)$

# Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]}) \leftarrow$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \underline{np.sum}(dZ^{[2]}, \underline{axis=1}, \underline{keepdims=True})$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$Y = [y^{(1)}\ y^{(2)} \cdots, y^{(m)}]$$

$(n^{[1]}, )$

$\downarrow (n^{[2]}, 1)$

# Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

$$Y = [y^{(1)}\ y^{(2)} \cdots, y^{(m)}]$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \underbrace{np.sum}(dZ^{[2]}, \underbrace{axis=1}, \underbrace{keepdims=True})$$

$(n^{[2]}, 1)$

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

element-wise product

and so this times there is an element-wise product of two matrices.

# Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

$$Y = [y^{(1)}\ y^{(2)} \cdots, y^{(m)}]$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \underbrace{np.sum}(dZ^{[2]}, \underbrace{axis=1}, \underbrace{keepdims=True})$$

$(n^{[2]}, 1)$

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

element-wise product

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$

# Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

$$Y = [y^{(1)}\ y^{(2)} \cdots, y^{(m)}]$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T} \quad (n^{[2]}, n^{[1]})$$

$$db^{[2]} = \frac{1}{m} \underline{np.sum}(dZ^{[2]}, \underline{axis=1}, \underline{keepdims=True}) \quad (n^{[2]}, 1)$$

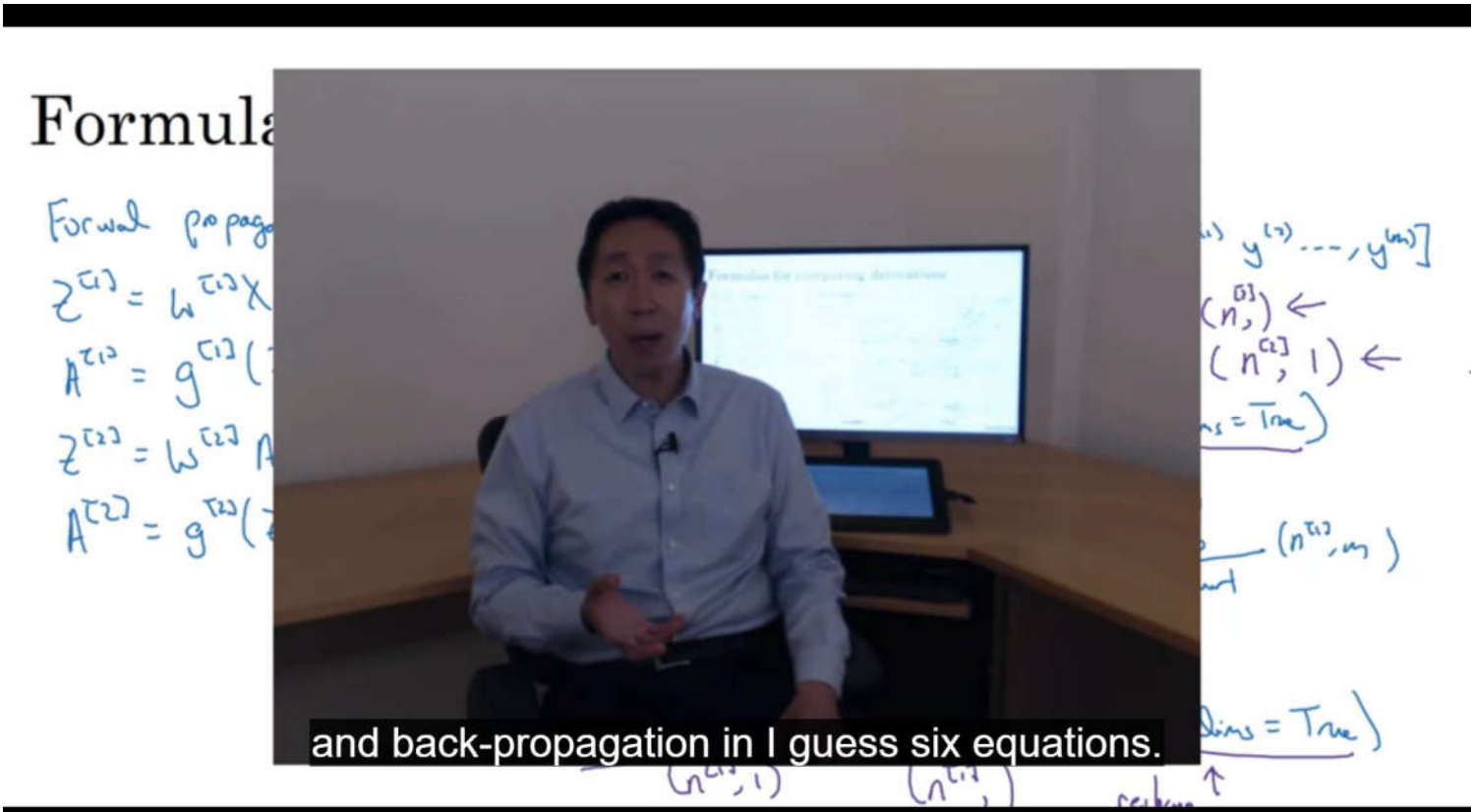
$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{} \quad (n^{[1]}, m)$$

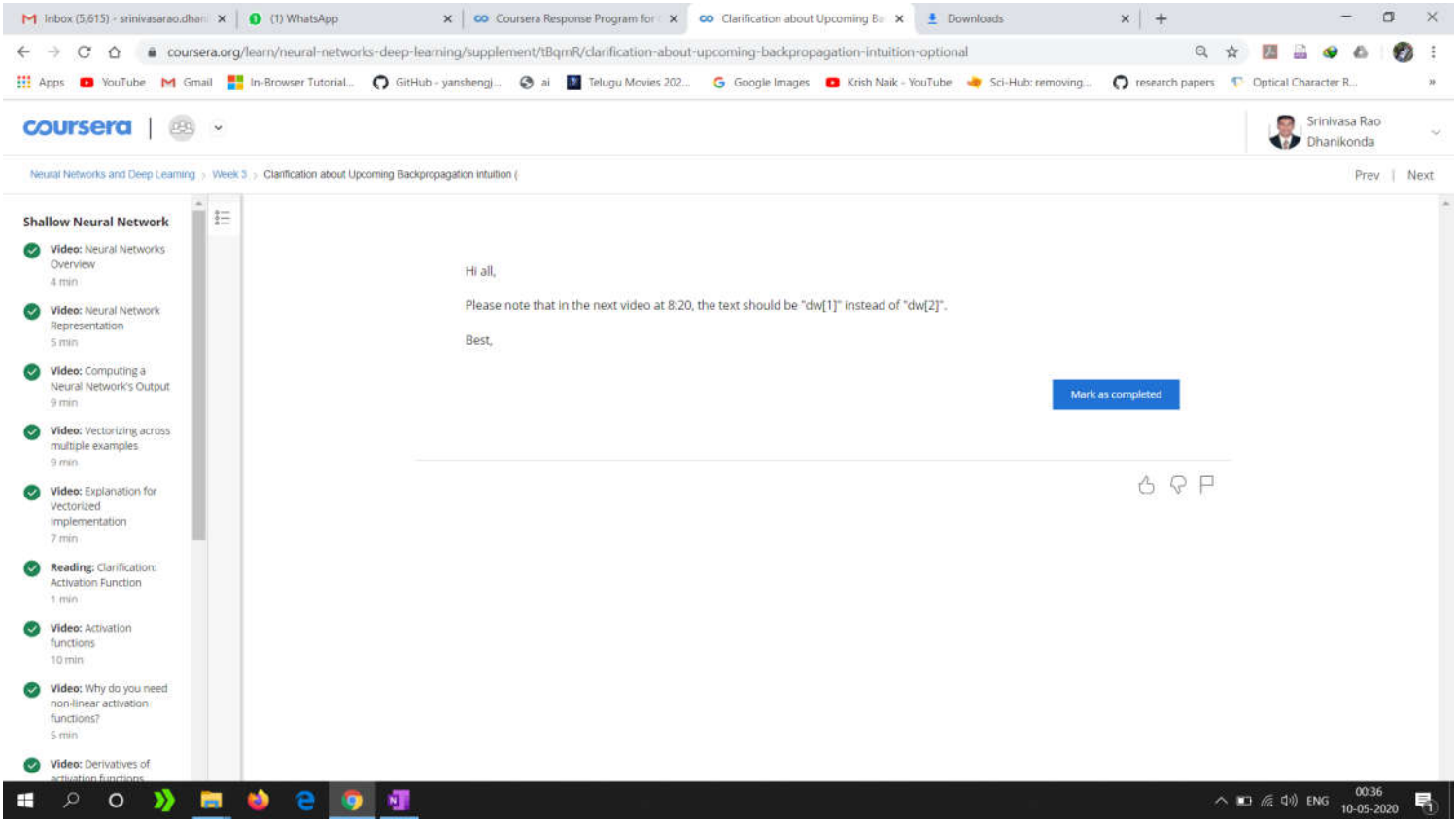
element-wise product

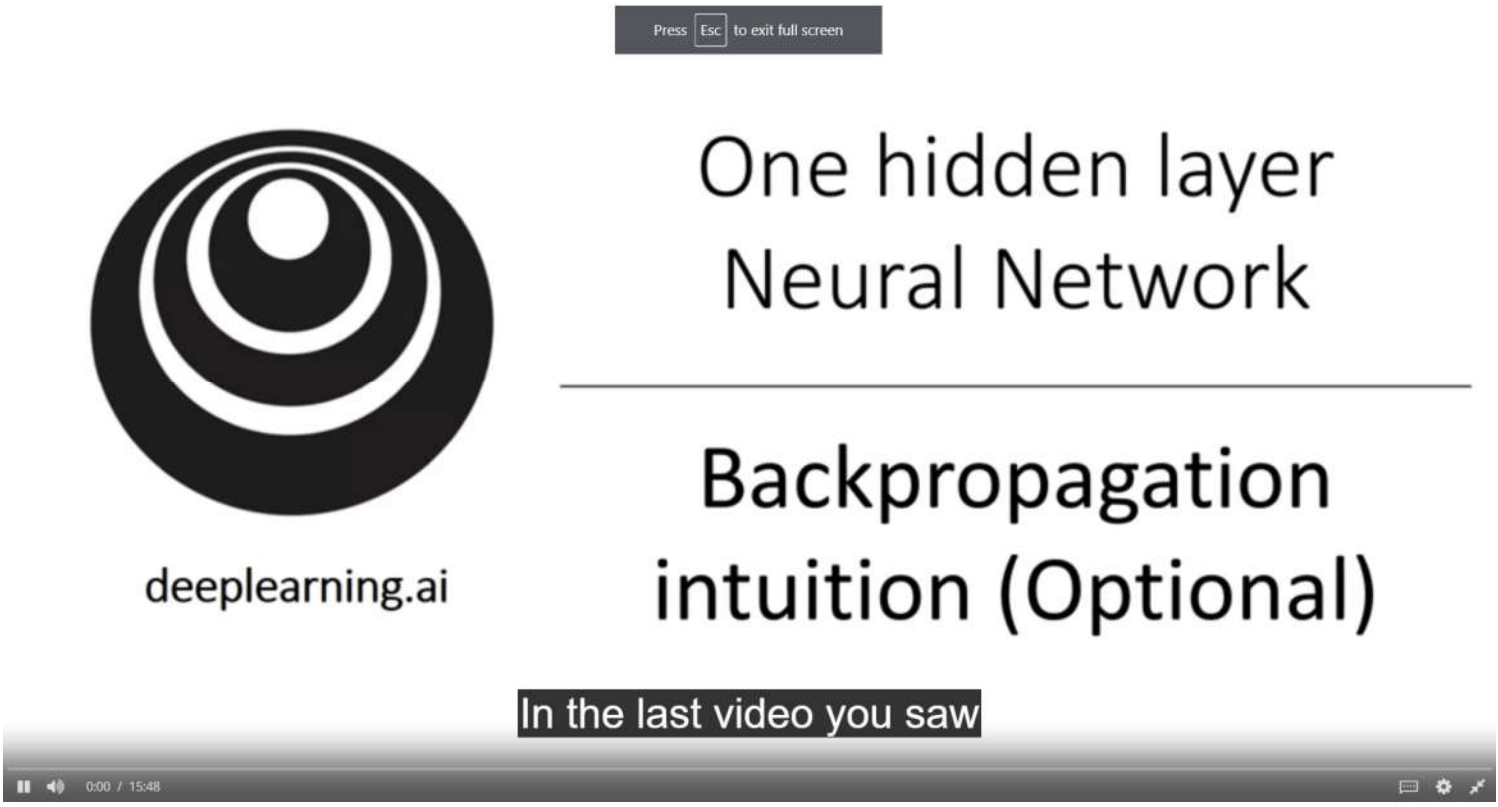
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

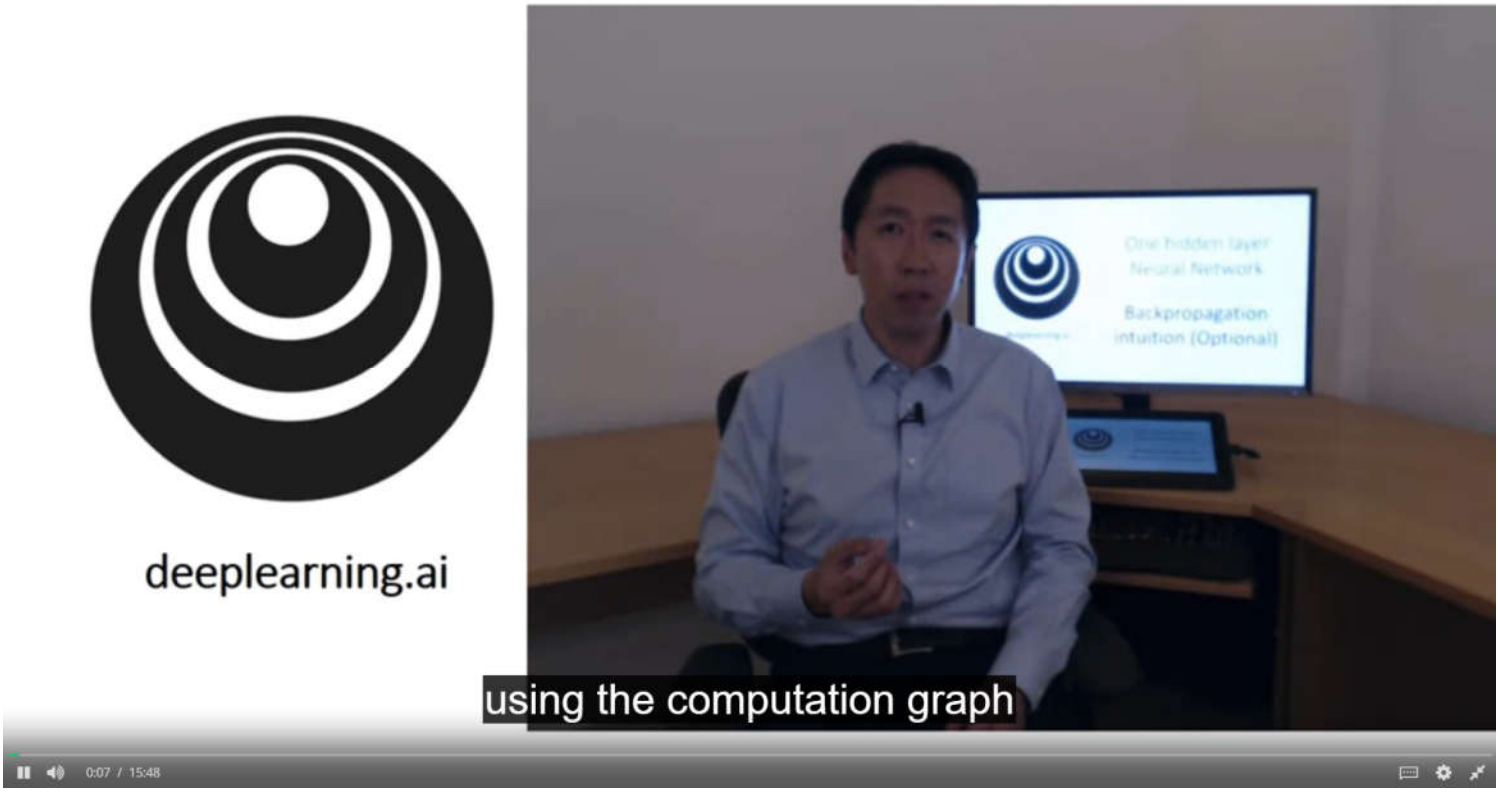
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, \underline{keepdims=True})$$

$(n^{[1]}, 1)$ $(n^{[1]},)$ reshape


Formula
and back-propagation in I guess six equations.

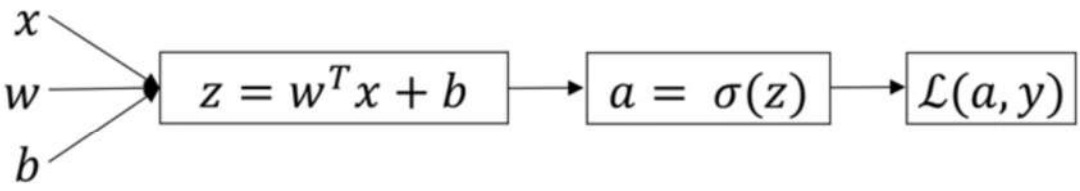

coursera.org/learn/neural-networks-deep-learning/supplement/tBqmR/clarification-about-upcoming-backpropagation-intuition-optional
coursera
Srinivasa Rao Dhanikonda
Neural Networks and Deep Learning › Week 3 › Clarification about Upcoming Backpropagation intuition (
Prev | Next
Shallow Neural Network
Video: Neural Networks Overview 4 min
Video: Neural Network Representation 5 min
Video: Computing a Neural Network's Output 9 min
Video: Vectorizing across multiple examples 9 min
Video: Explanation for Vectorized Implementation 7 min
Reading: Clarification: Activation Function 1 min
Video: Activation functions 10 min
Video: Why do you need non-linear activation functions? 5 min
Video: Derivatives of activation functions
Hi all,
Please note that in the next video at 8:20, the text should be "dw[1]" instead of "dw[2]".
Best,
Mark as completed


Press Esc to exit full screen
One hidden layer
Neural Network
deeplearning.ai
Backpropagation
intuition (Optional)
In the last video you saw
0:00 / 15:48


deeplearning.ai
using the computation graph
0:07 / 15:48

# Computing gradients

## Logistic regression

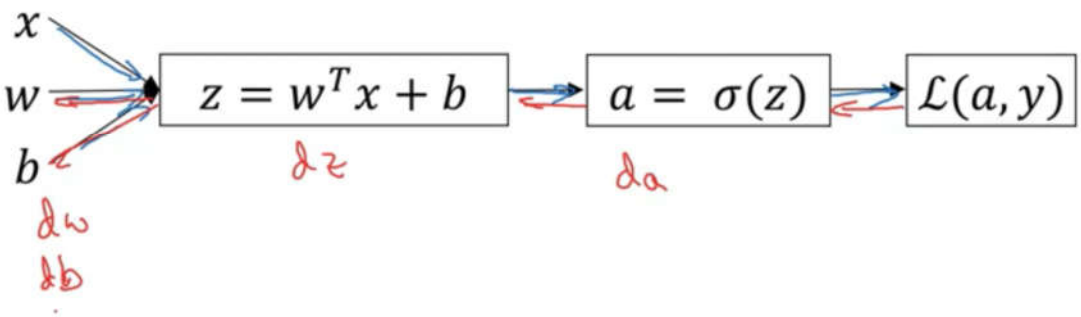



So recall that when
we talked about

Andrew Ng

# Computing gradients

## Logistic regression

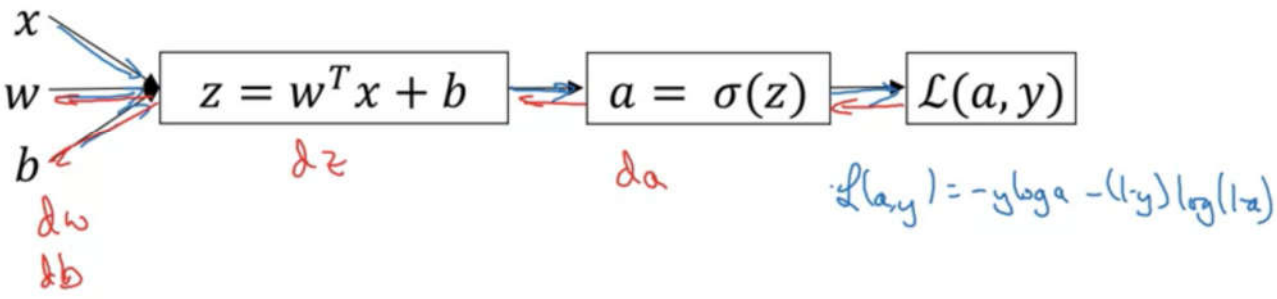



and then go on to
compute dw and db.

Andrew Ng

# Computing gradients

## Logistic regression

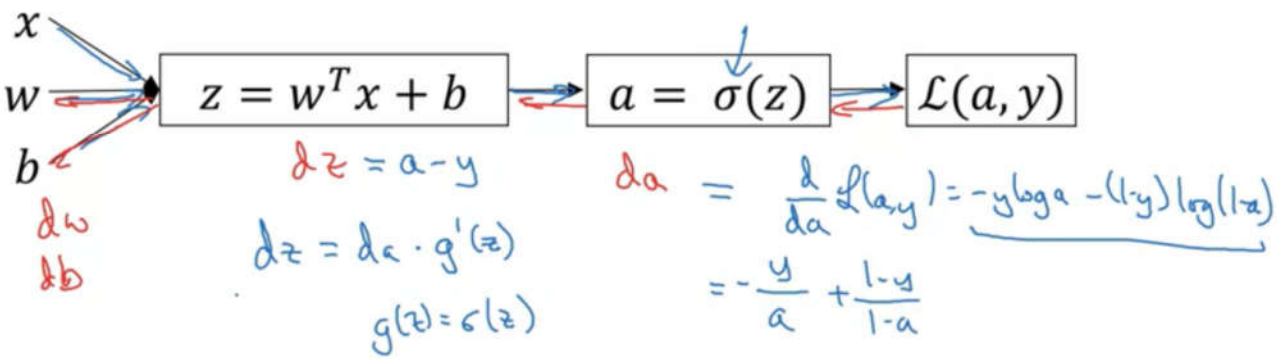



So if you're familiar with

Andrew Ng

# Computing gradients

## Logistic regression

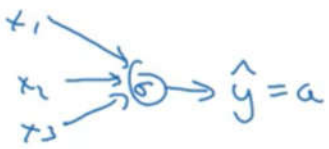


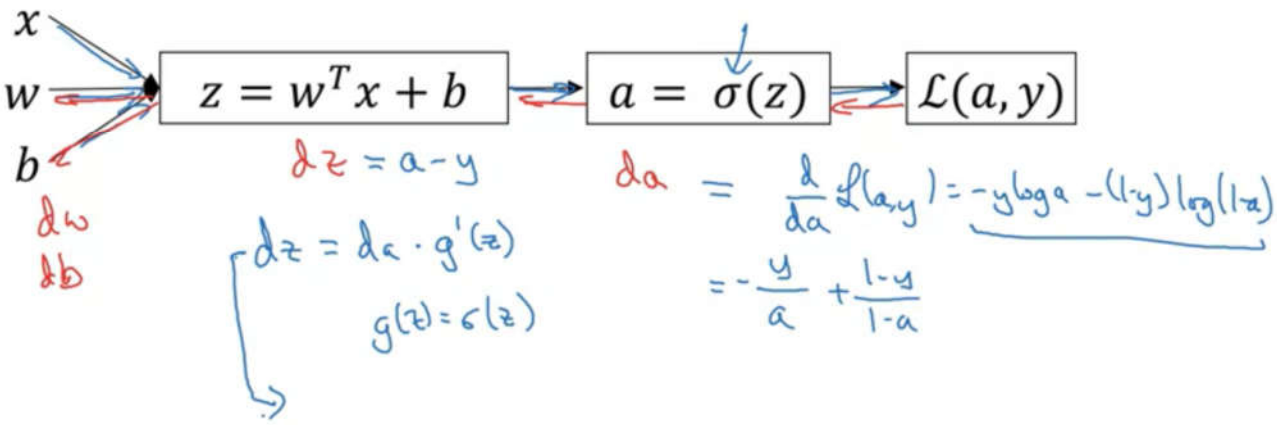

As in aside only for those of you familiar

Andrew Ng

# Computing gradients

## Logistic regression



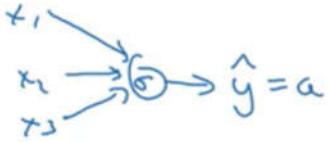



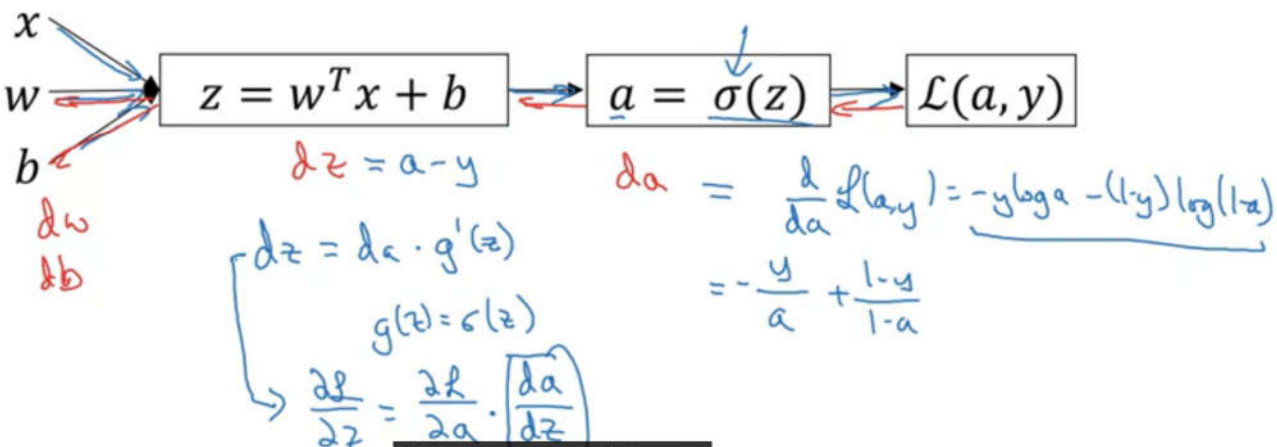

with the chain rule of calculus,

Andrew Ng

# Computing gradients

## Logistic regression



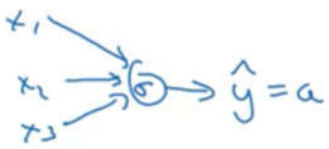



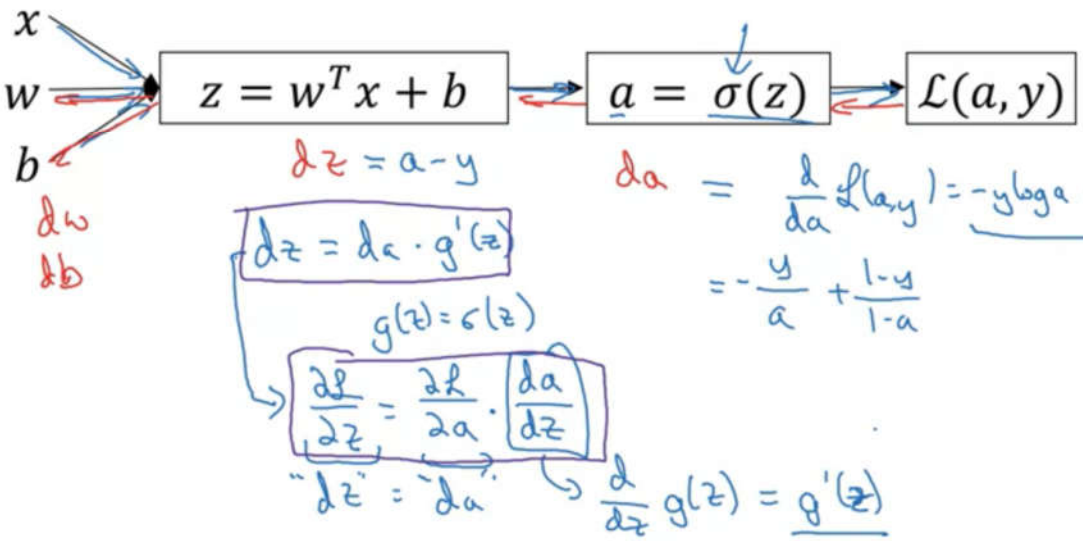

this is equal to d,

Andrew Ng

# Computing gradients

## Logistic regression

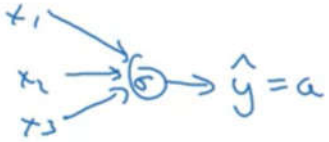


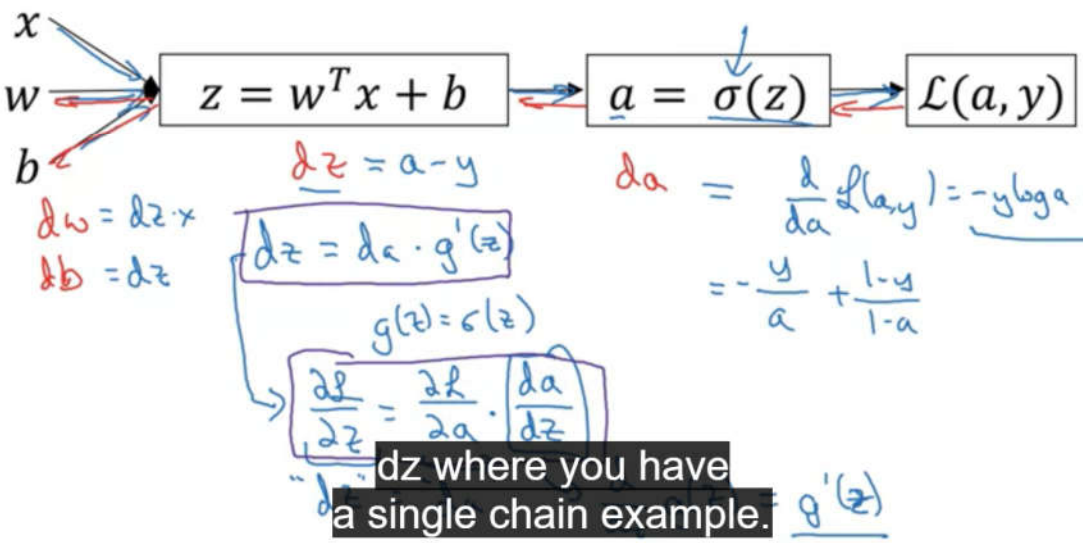




Andrew Ng

# Computing gradients

## Logistic regression

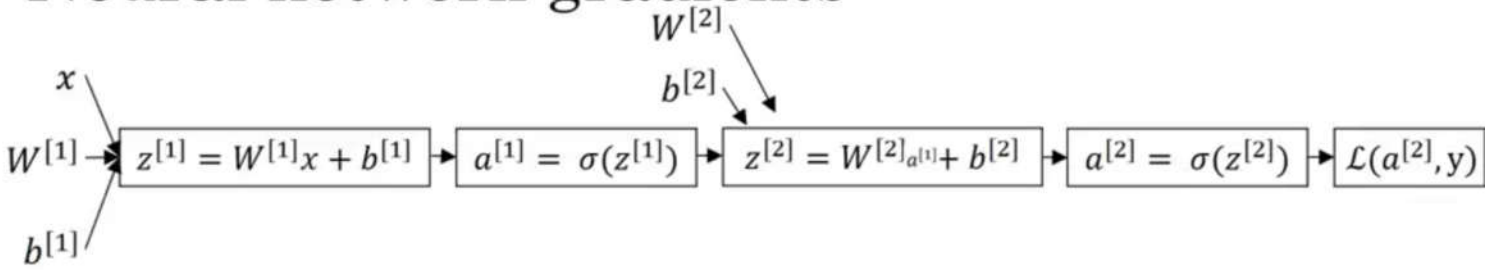



dz where you have
a single chain example.

Andrew Ng

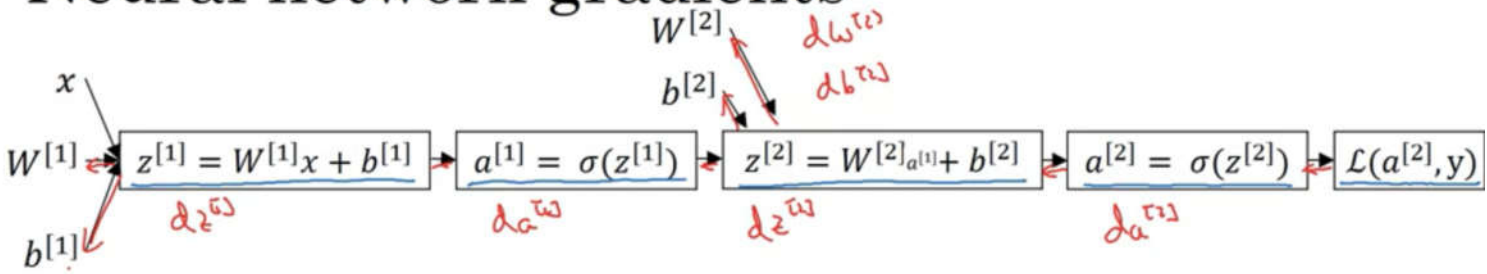
# Neural network gradients




a hidden layer, and
an output layer

Andrew Ng

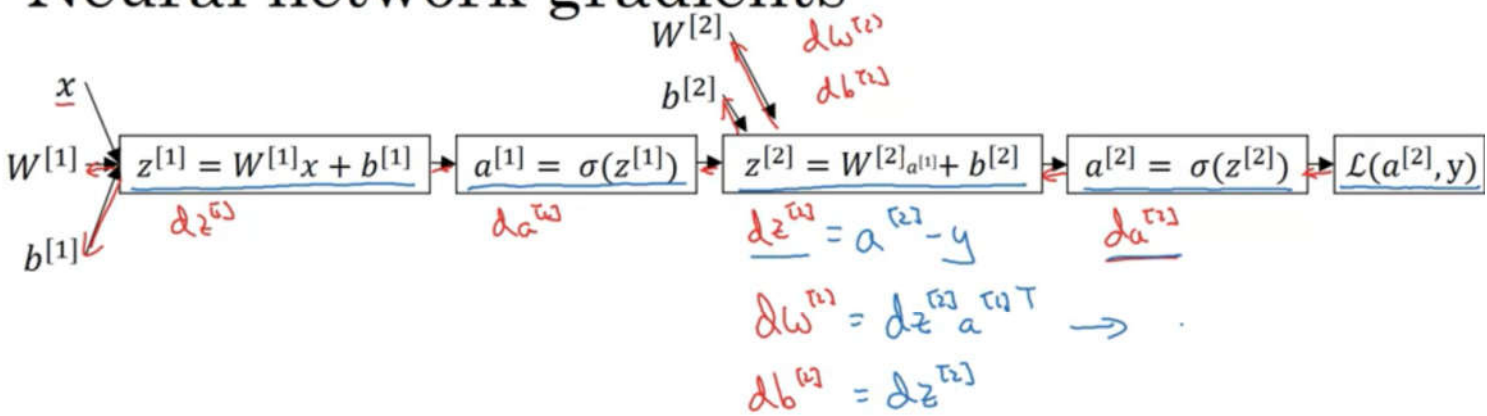
# Neural network gradients




go backwards to compute
da_1, dz_1 and so on.

Andrew Ng

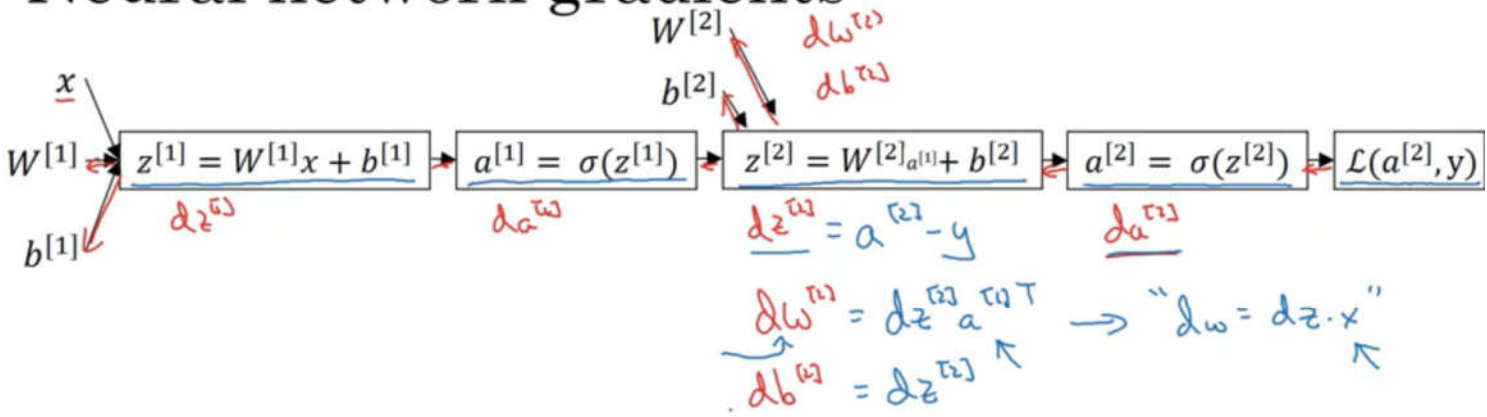
# Neural network gradients




for logistic regression
where we had that

Andrew Ng

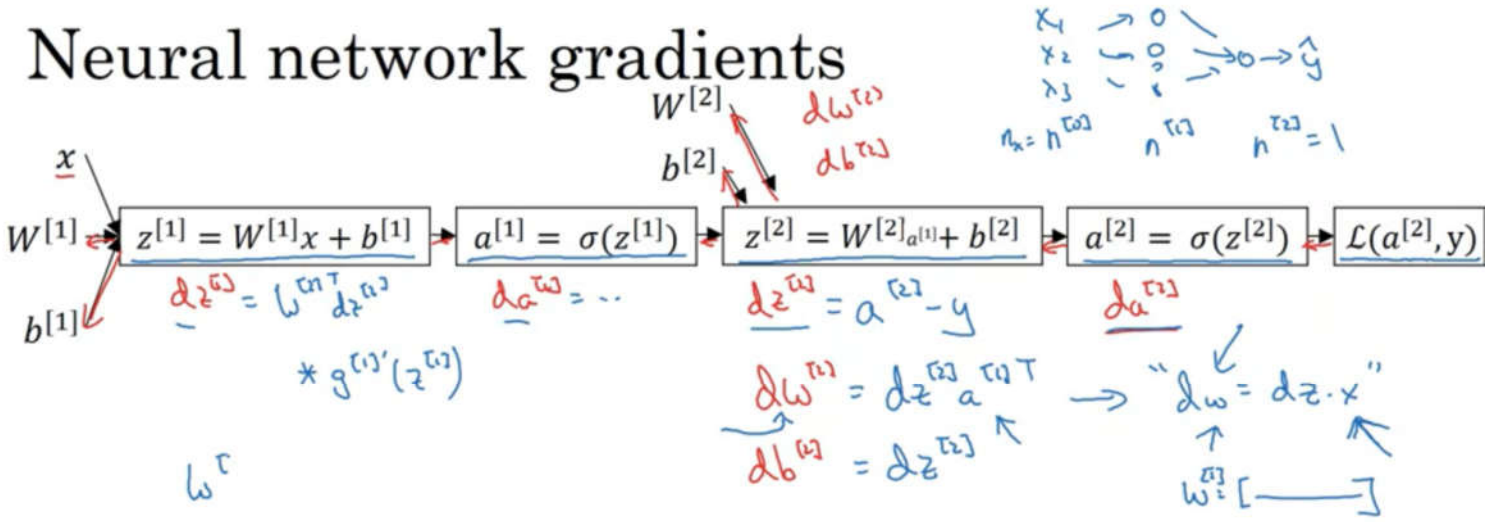
# Neural network gradients




the relationship between
the capital matrix W

Andrew Ng

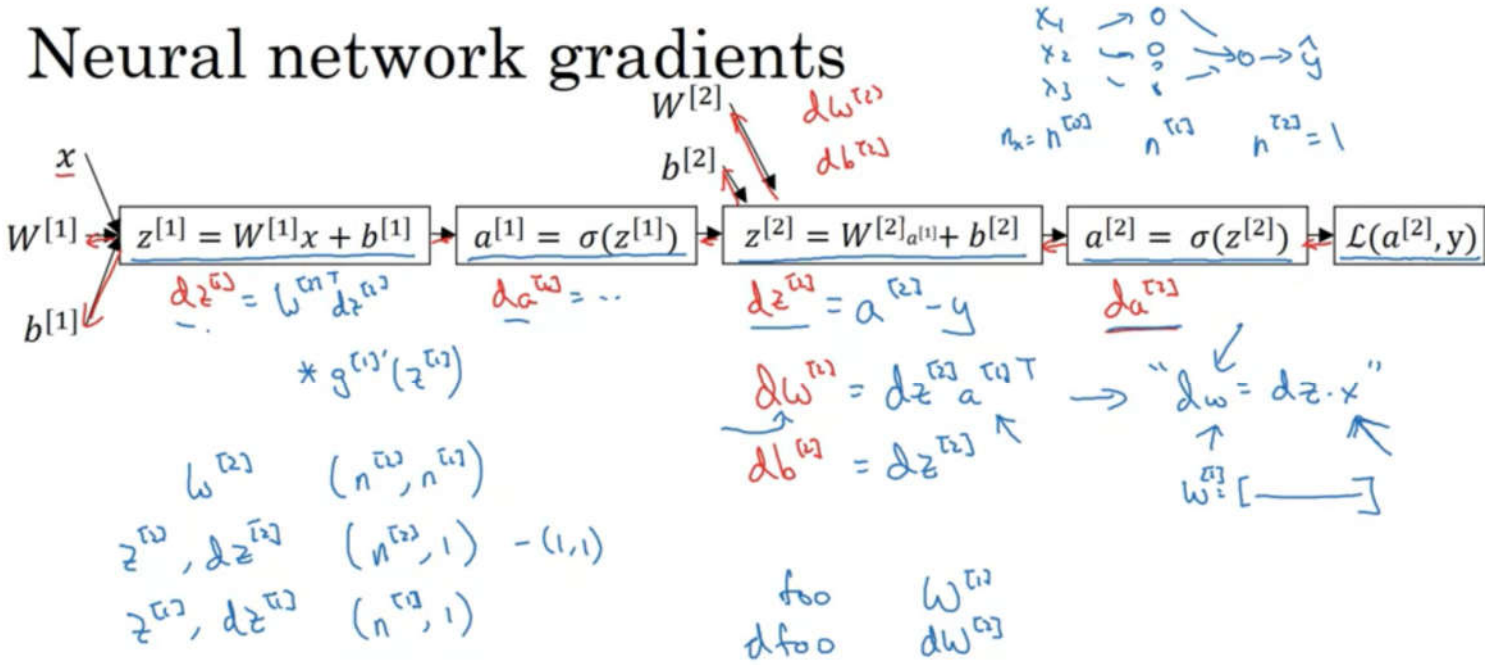
# Neural network gradients




the matrix W_2 is n_2
by n_1 dimensional,

Andrew Ng

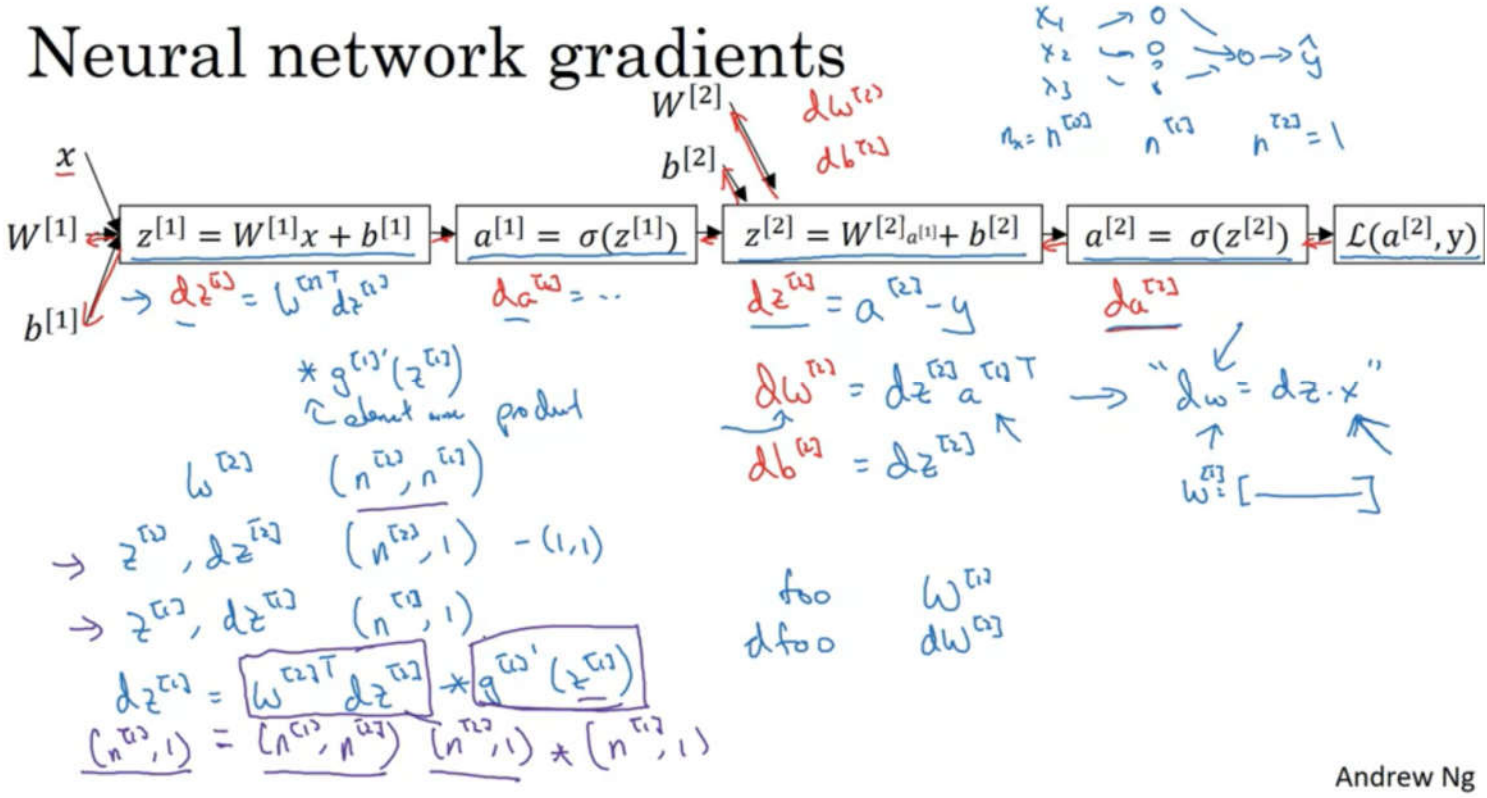
# Neural network gradients




So to make sure that the
dimensions of this all match up,

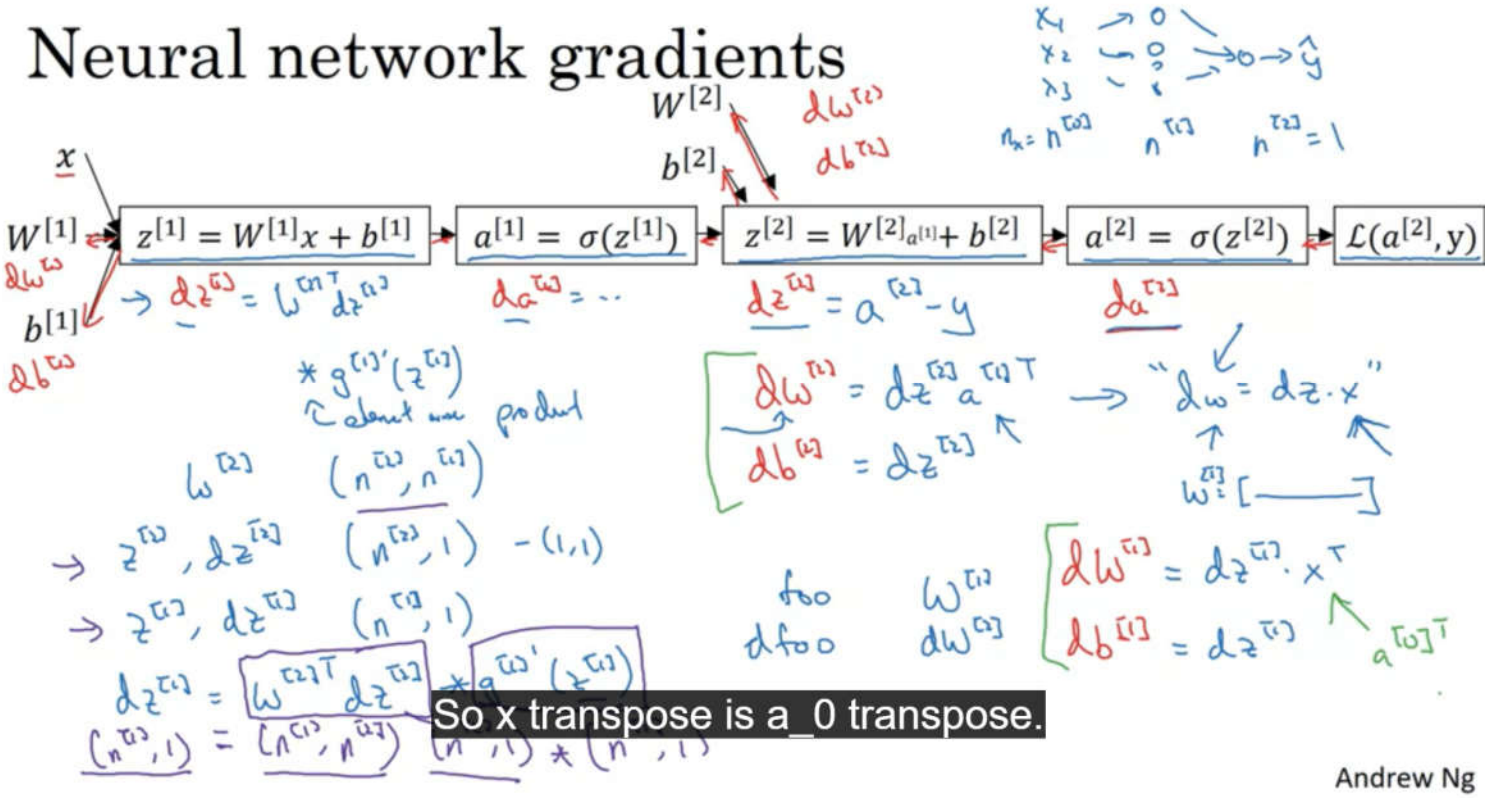
Andrew Ng

# Neural network gradients




Andrew Ng

# Neural network gradients




So x transpose is a_0 transpose.

Andrew Ng

# Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

So far we have derived
back-propagation for

Andrew Ng

# Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

Vectorized Implementation:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

of a matrix makes
all this workout.

Andrew Ng

# Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$\underset{(n^{[1]},1)}{dz^{[1]}} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$J(\ldots) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

elementwise product

$$\underset{(n^{[1]},m)}{dZ^{[1]}} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

14:20 / 15:48 Andrew Ng

# One hidden layer Neural Network

## Random Initialization

deeplearning.ai

When you change your neural network,

0:00 / 7:57

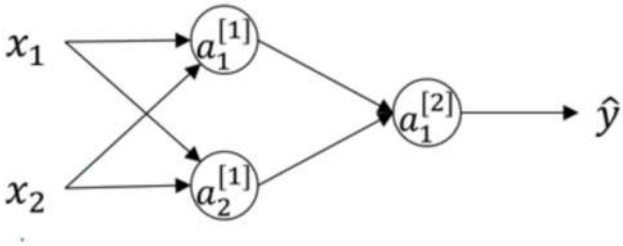
deeplearning.ai

deeplearning.ai

deeplearning.ai
One hidden layer
Neural Network
Random Initialization
But for a neural network of initialize
the weights to parameters to all zero and

deeplearning.ai
One hidden layer
Neural Network
Random Initialization
then applied gradient descent,
it won't work.

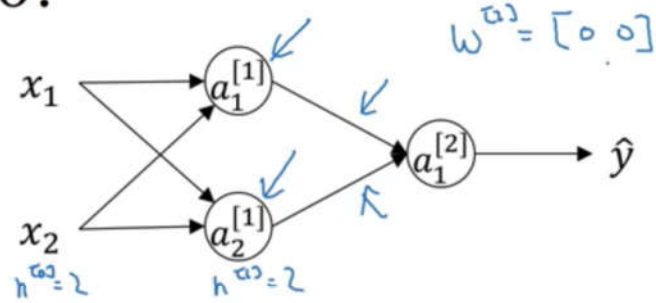
# What happens if you initialize weights to zero?




So you have here two input features, so

Andrew Ng

# What happens if you initialize weights to zero?



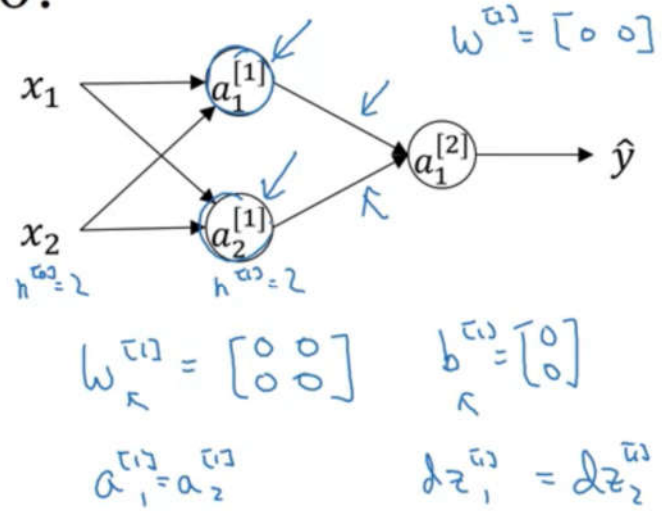

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]} \qquad dz_1^{[1]} = dz_2^{[1]}$$

But if you initialize
the neural network this way,

Andrew Ng

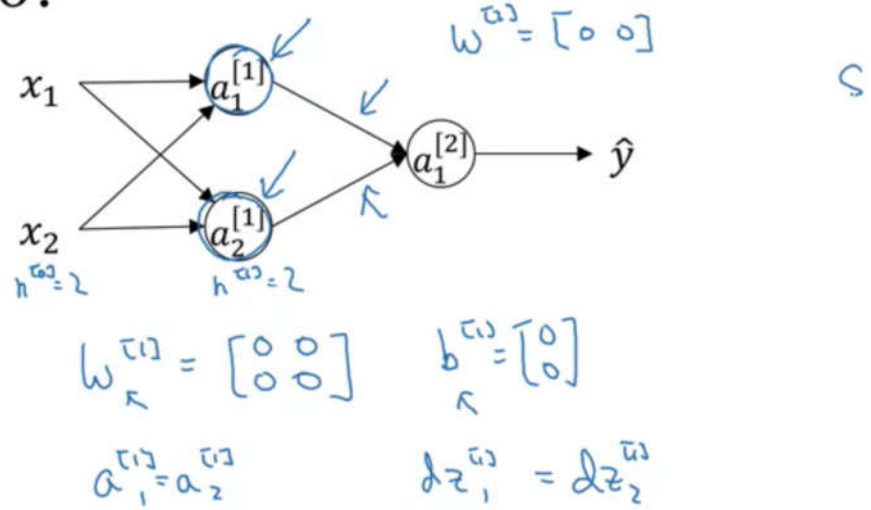
# What happens if you initialize weights to zero?




then this hidden unit and
this hidden unit are completely identical.

Andrew Ng

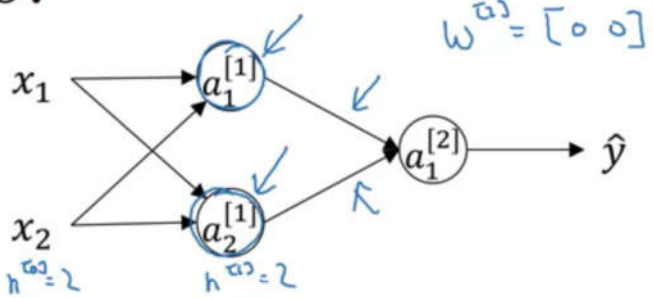
# What happens if you initialize weights to zero?




Sometimes you say they're
completely symmetric,

Andrew Ng

# What happens if you initialize weights to zero?




$W^{[1]} = [0 \ 0]$

Symmetric

$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

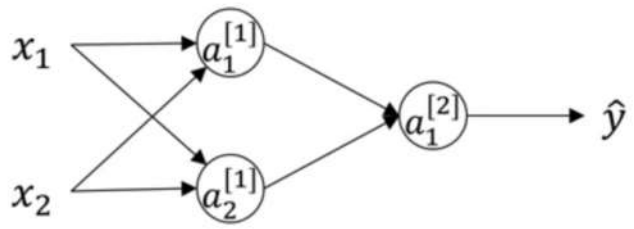
$a_1^{[1]} = a_2^{[1]}$ $\quad dz_1^{[1]} = dz_2^{[1]}$

$W^{[1]} = \begin{bmatrix} \cdots & \cdot \\ \cdots & \cdot \end{bmatrix}$

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\quad W^{[1]} = W^{[1]} - \alpha dW$

Andrew Ng

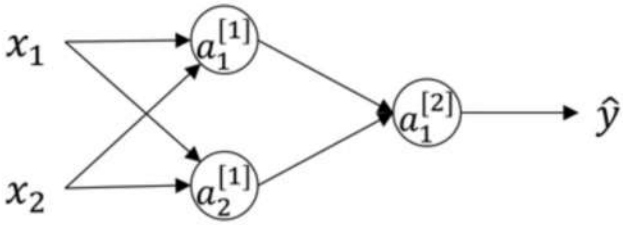
# Random initialization




The solution to this is to initialize your parameters randomly.

Andrew Ng

# Random initialization

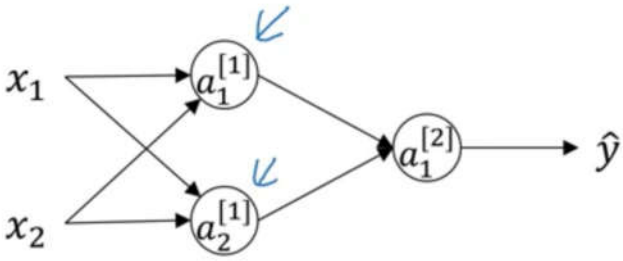



$W^{[1]} = np.random.randn((2,2)) * 0.01$

So you initialize it to
very small random values.

Andrew Ng

# Random initialization



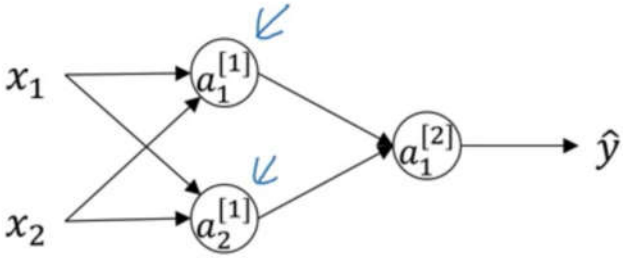

$\rightarrow W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

And so you no longer have this
symmetry breaking problem.

Andrew Ng

# Random initialization




$\rightarrow W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

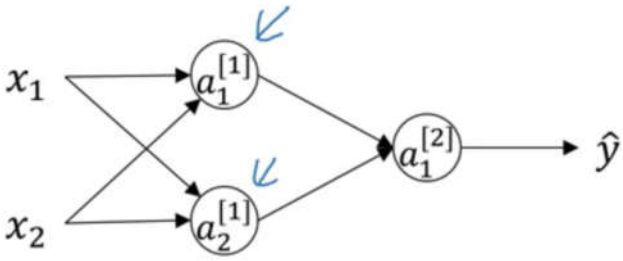
$W^{[2]} = \ldots$

$b^{[2]} = 0$

So you might be wondering, where did this constant come from and why is it 0.01?

Andrew Ng

# Random initialization




$\rightarrow W^{[1]} = np.random.randn((2,2)) * 0.01$

1.

$b^{[1]} = np.zero((2,1))$

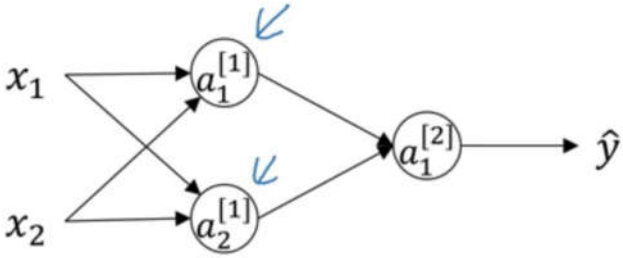
$W^{[2]} = \ldots$

$b^{[2]} = 0$

Why not put the number 100 or 1000?

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * \frac{0.01}{100?}$

$b^{[1]} = np.zero((2,1))$

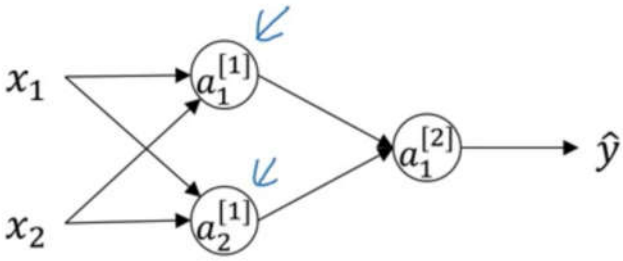
$W^{[2]} = \ldots$

$b^{[2]} = 0$

the weights to very small random values.

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * \frac{0.01}{100?}$

$b^{[1]} = np.zero((2,1))$

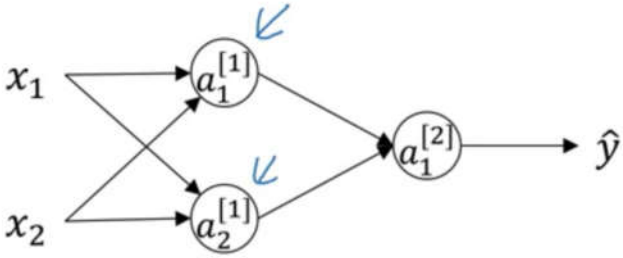
$W^{[2]} = \ldots$

$b^{[2]} = 0$

Because if you are using a tanh or sigmoid activation function, or

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * \frac{0.01}{100?}$

$b^{[1]} = np.zero((2,1))$

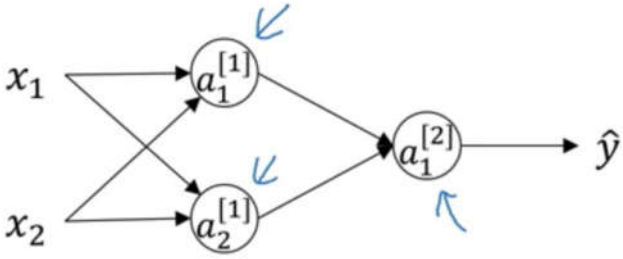
$W^{[2]} = \ldots$

$b^{[2]} = 0$

the other sigmoid,
even just at the output layer.

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * \frac{0.01}{100?}$

$b^{[1]} = np.zero((2,1))$

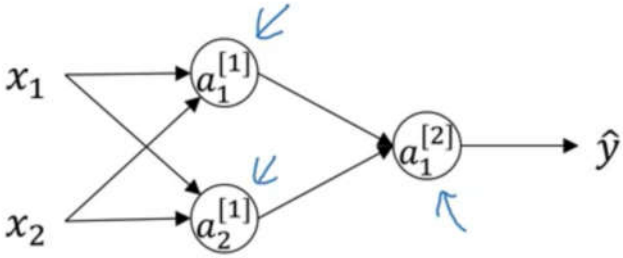
$W^{[2]} = \ldots$

$b^{[2]} = 0$

then when you compute
the activation values,

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

$W^{[2]} = \ldots$

$b^{[2]} = 0$

$100?$

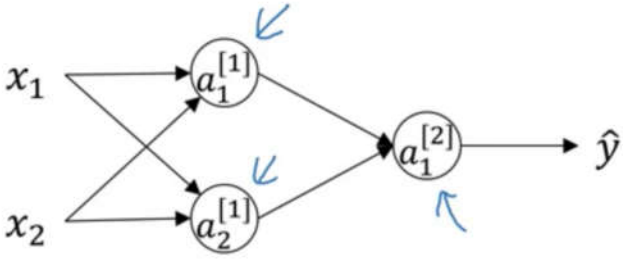
$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

values of z will be either very large or very small.

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

$W^{[2]} = \ldots$

$b^{[2]} = 0$

$100?$

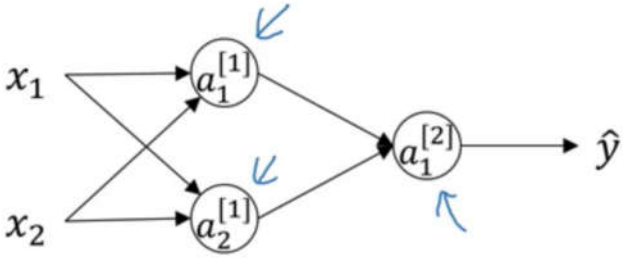
$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

even at the very start of training, with very large values of z.

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

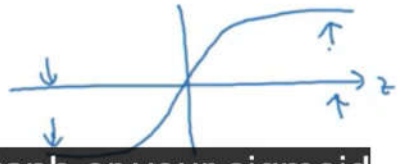
$W^{[2]} = \ldots$

$b^{[2]} = 0$

100?

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

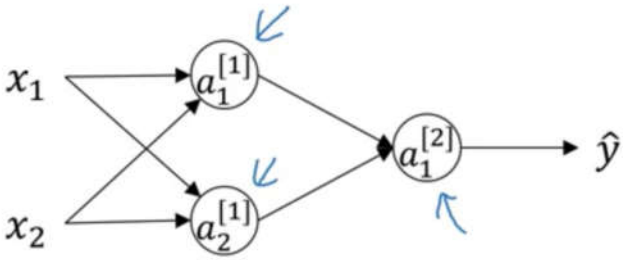



Which causes your tanh or your sigmoid activation function to be saturated,

6:13 / 7:57

Andrew Ng

# Random initialization




$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zero((2,1))$

$W^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$

100?

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

I guess this would be 1 by 2 in this example, times 0.01.

Andrew Ng