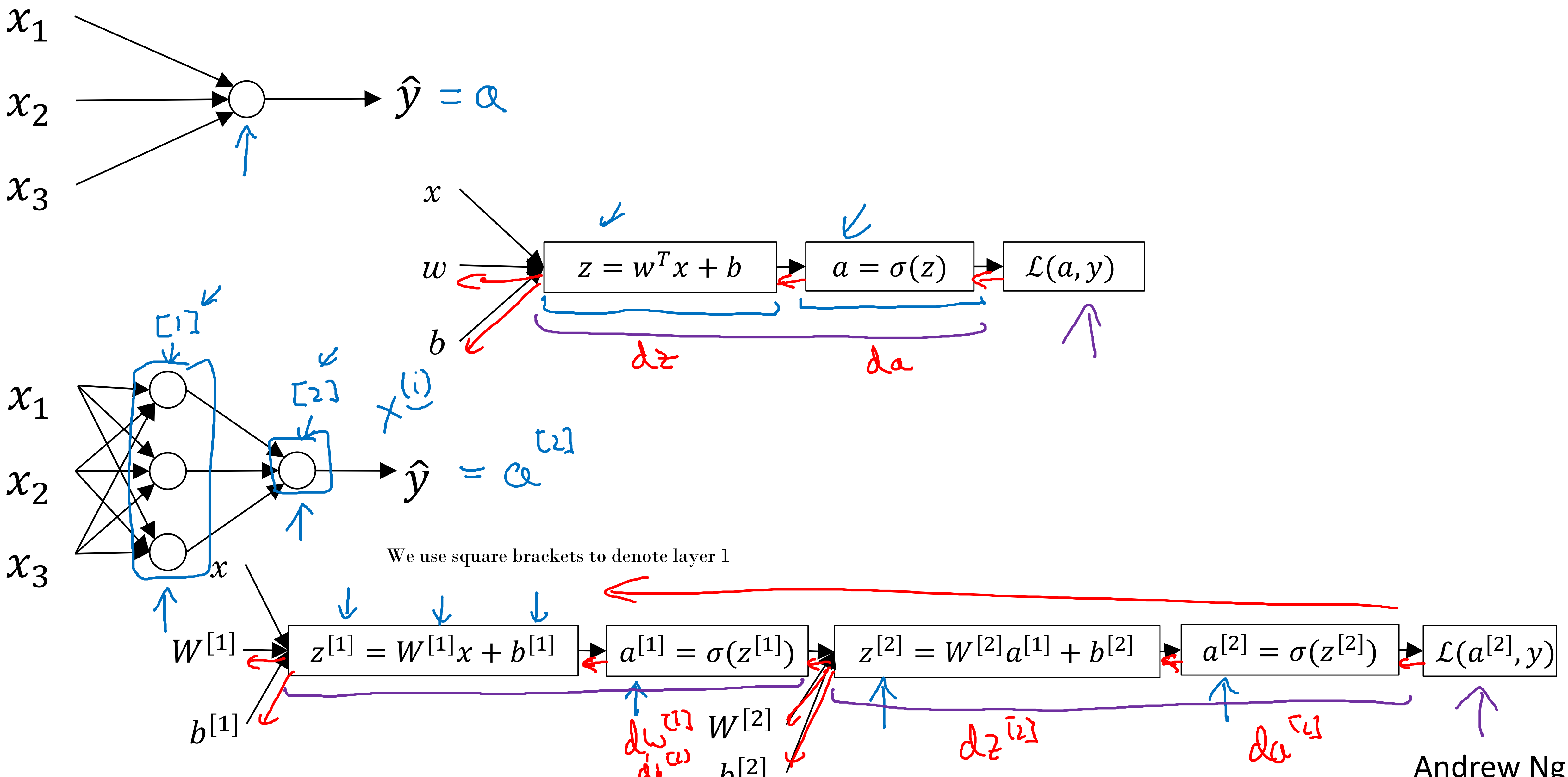
deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?



We use square brackets to denote layer 1

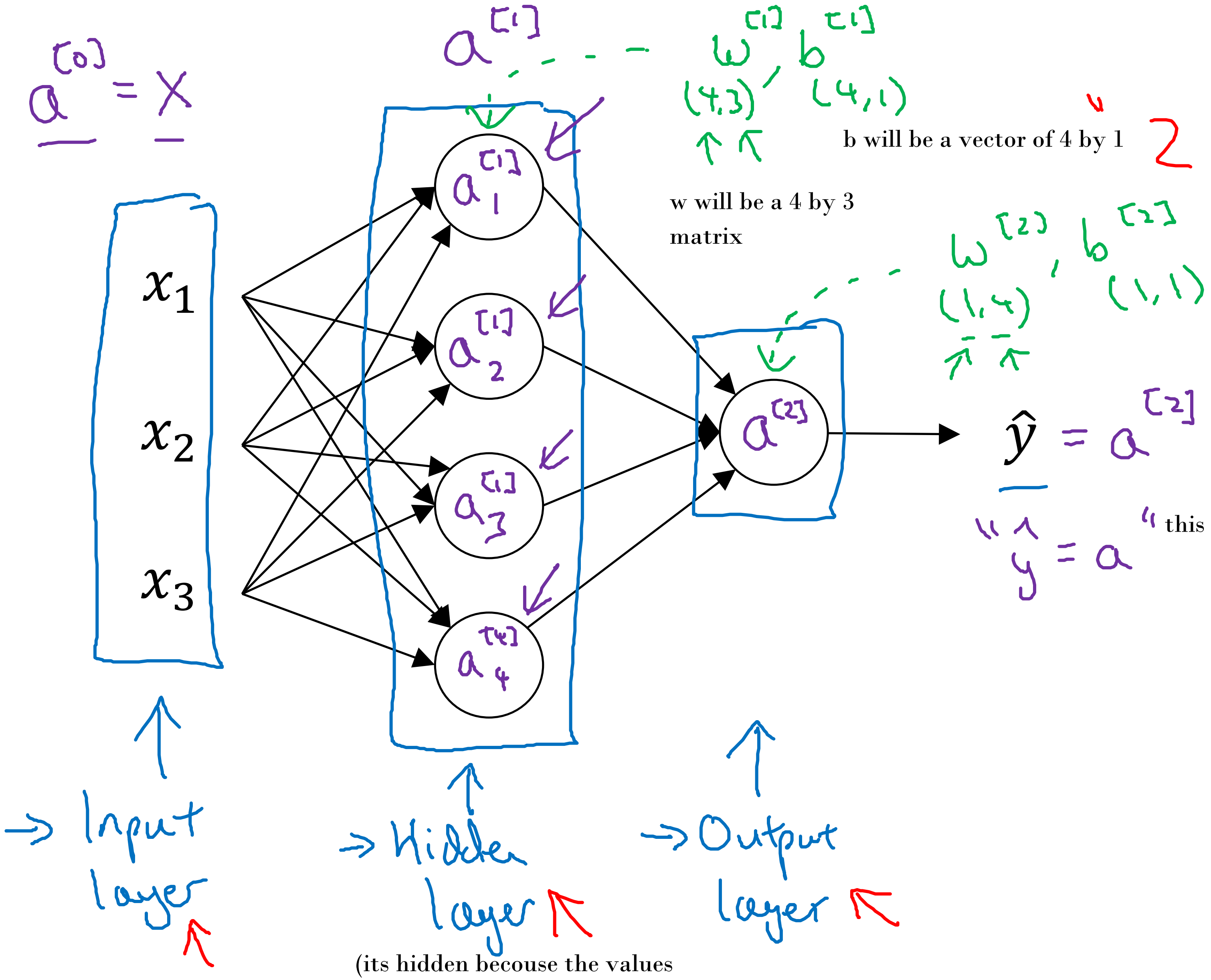
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



b will be a vector of 4 by 1

w will be a 4 by 3 matrix

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

“2 layer NN”

we dont count imput layer (input layer is layer 0)

“$\hat{y} = a$” this was for logistic regression as we had only one layer

$$a^{[1]} = \begin{bmatrix} a^{[1]}_1 \\ a^{[1]}_2 \\ \vdots \\ a^{[1]}_4 \end{bmatrix}$$

this is the hidden layer

(its hidden becouse the values of this are not in training set)

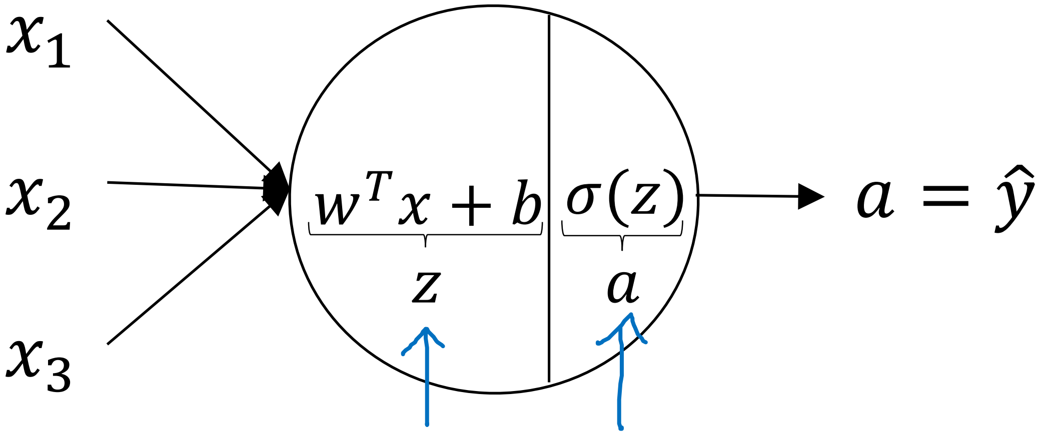
Andrew Ng

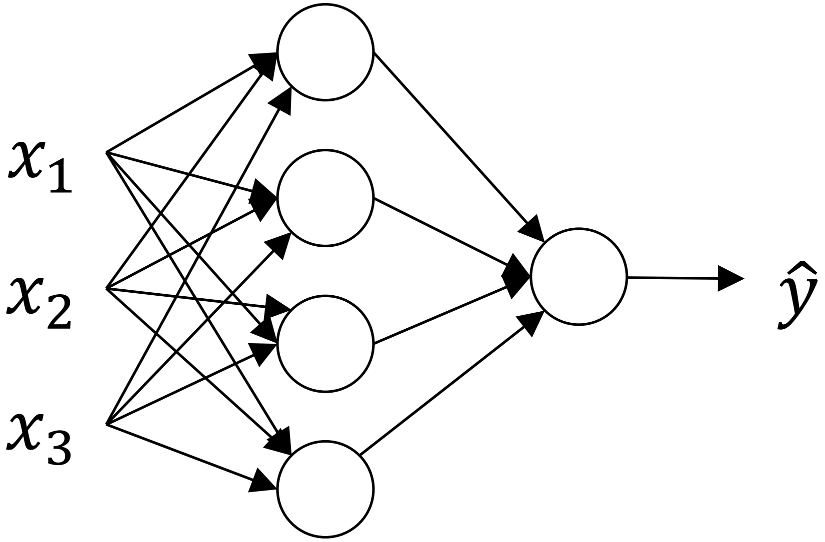
deeplearning.ai

One hidden layer Neural Network

Computing a Neural Network's Output

Neural Network Representation

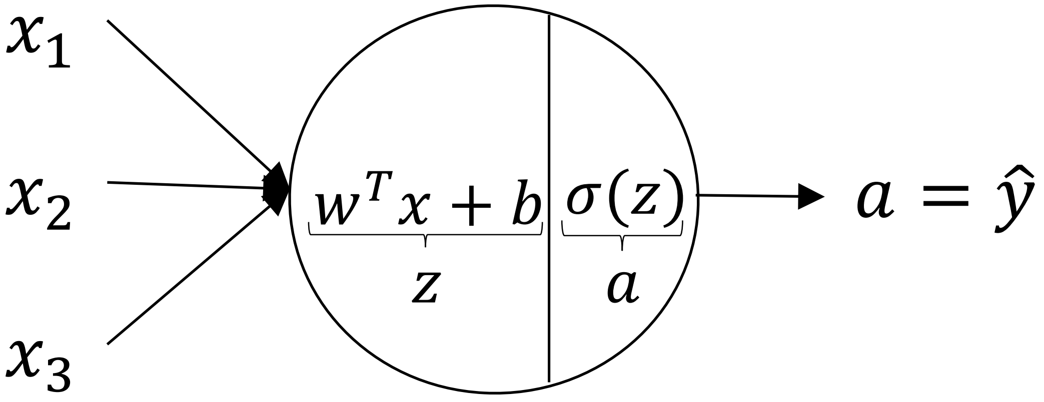




$$z = w^Tx + b$$

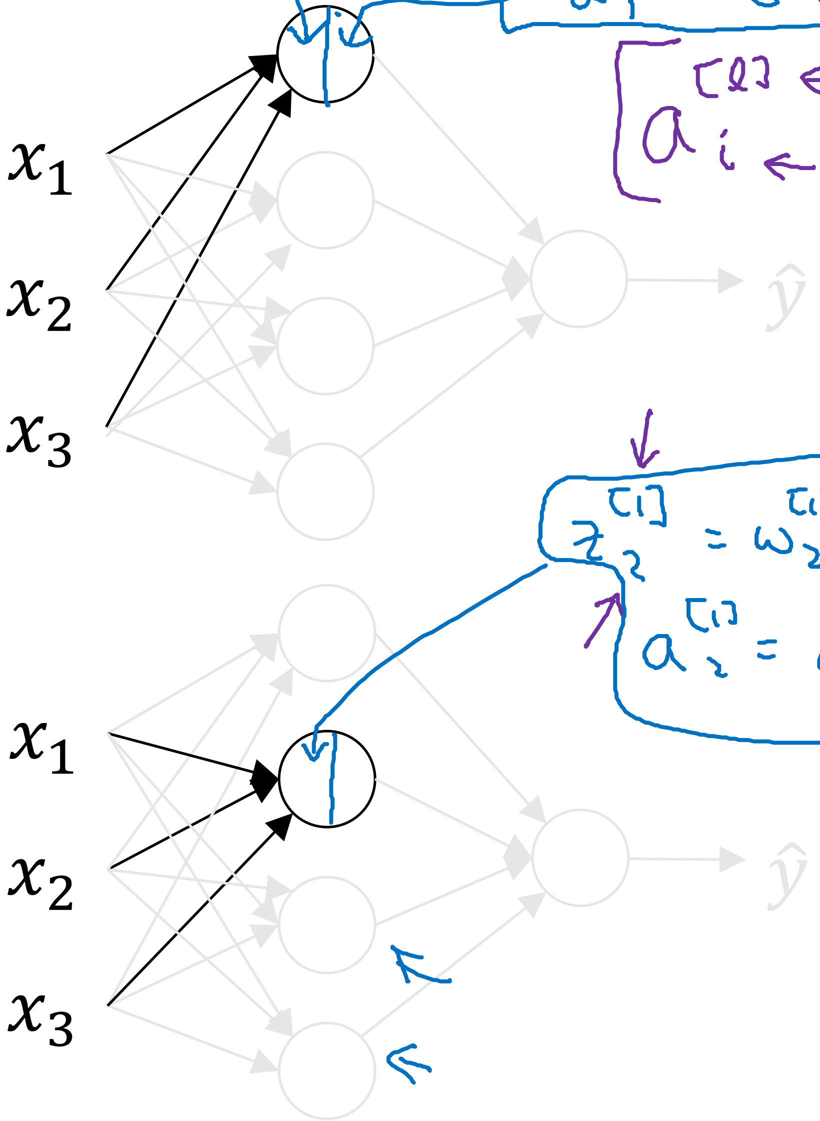
$$a = \sigma(z)$$

Neural Network Representation



$$z = w^T x + b$$

$$a = \sigma(z)$$



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

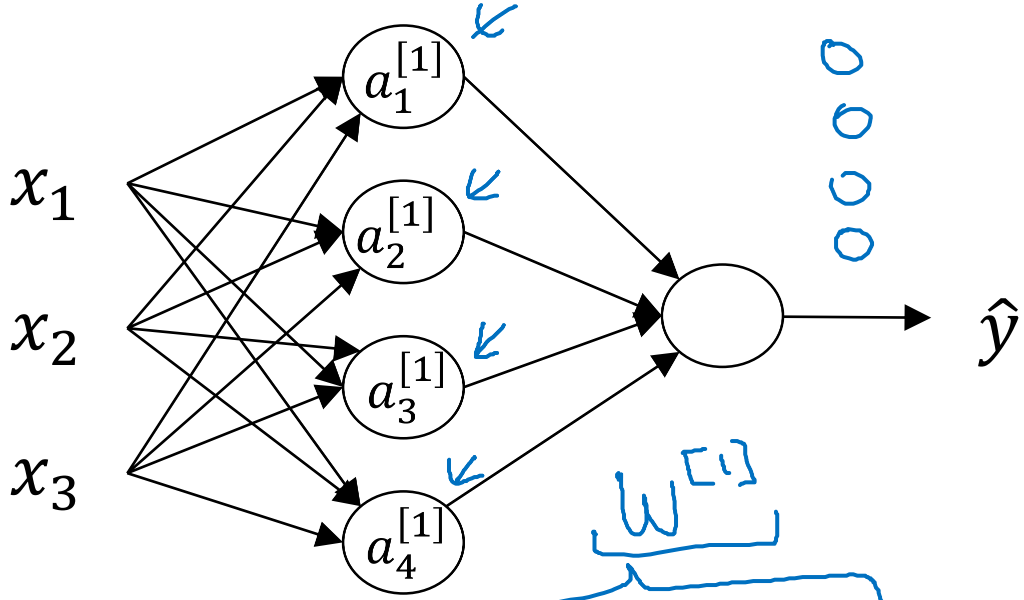
$a_i^{[l]}$ ← layer, ← node in layer.

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

Andrew Ng

Neural Network Representation



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

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

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

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

$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

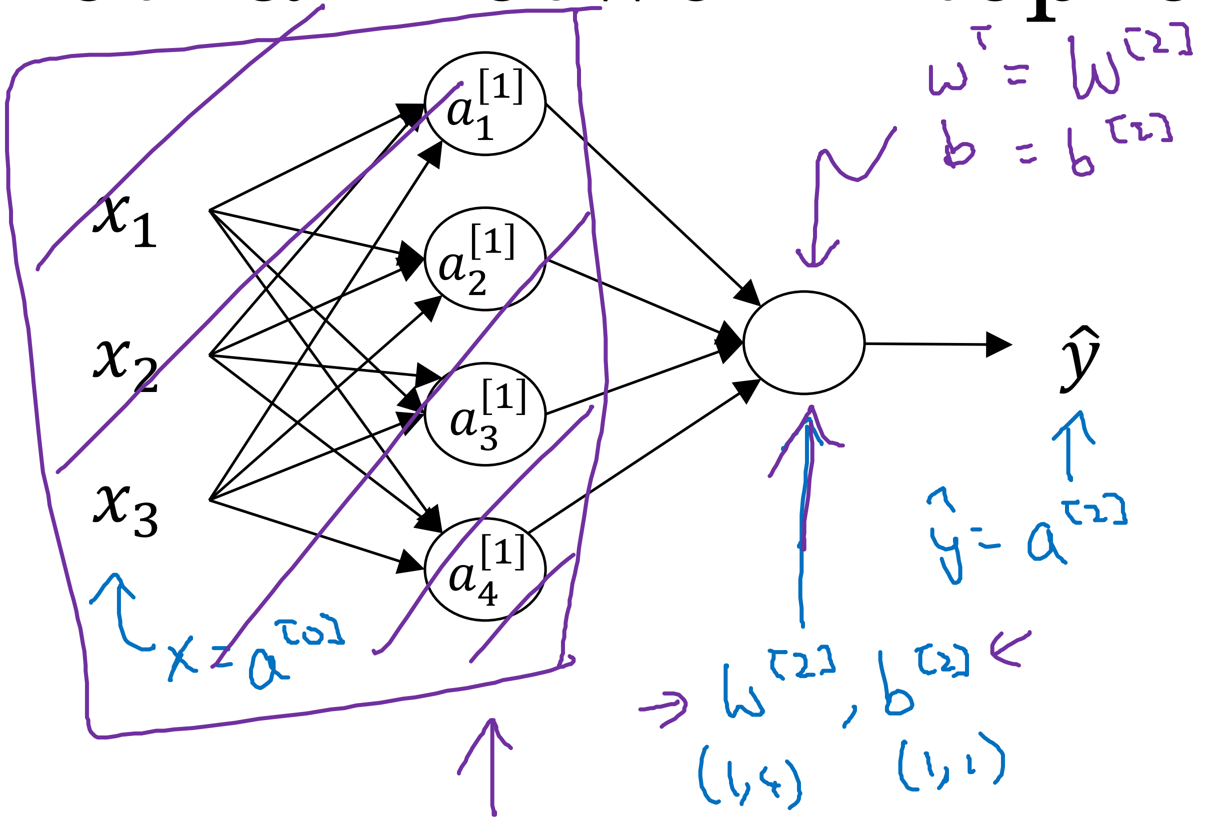
$$\rightarrow z^{[1]} = \overbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}^{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix}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

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

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

Andrew Ng

Neural Network Representation learning



$$z = w^T x + b$$

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

Given input x:

you can replace x with a[0] as its the layer 0 (its same)

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

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

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

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

Andrew Ng

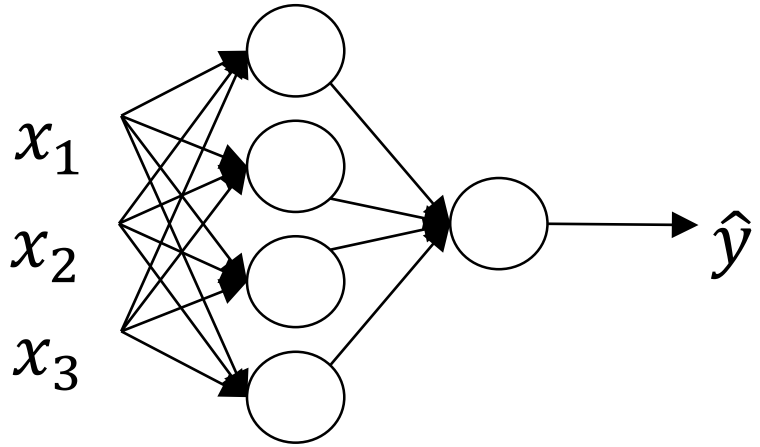
deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples

We saw previewsly how to calculate the output with one training example, now we will see for m training examples.



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

This where the four equations on how u compute it for just one training example

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

repeat for n training examples

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

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

example i

layer 2

To compute the four above eq for m training examples

we want to get rid of that for loop

```
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

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

This is how we vectorize it!!!

$$\boxed{\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \end{aligned}}$$

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

(n_x, m)

nx number of features

m is number of training examples

training examples

hidden units.

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

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

hidden units

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

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

to semplify we assume b = 0

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

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

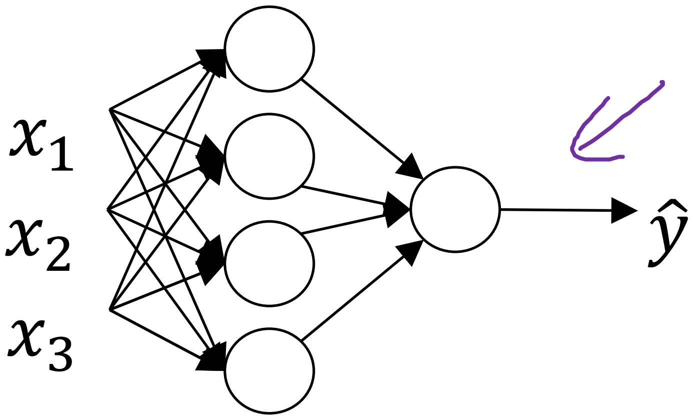
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

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

here we have just justified that the above is a correct vectorization.

Andrew Ng

Recap of vectorizing across multiple examples



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

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

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

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

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

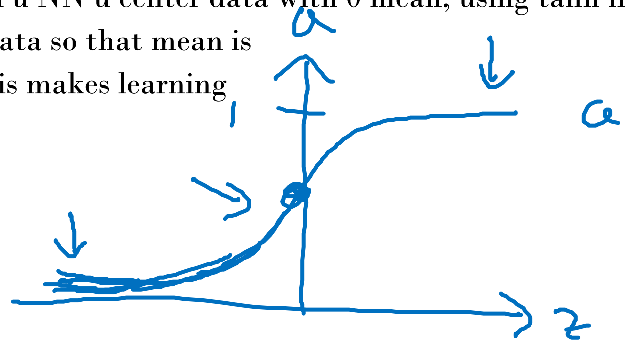
One hidden layer Neural Network

deeplearning.ai

Activation functions

Activation functions

tanh is centred at 0, when u train u NN u center data with 0 mean, using tanh instead of sigmund has the effect of centering your data so that mean is closer to 0 rather then 0.5 and this makes learning a bit easyier.



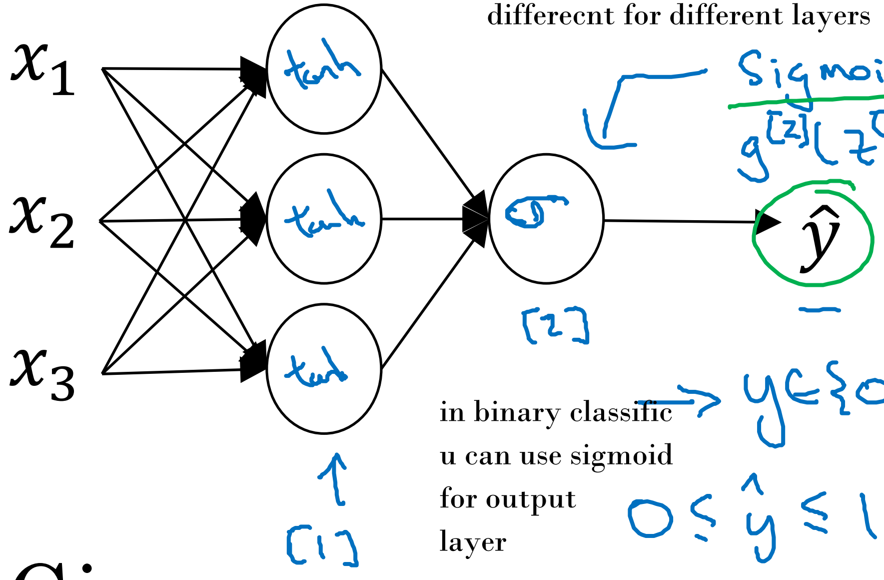
$a = \frac{1}{1+e^{-z}}$

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

use tanh for hidden layer and signoid for output layer. Activat fct can be differecnt for different layers

Sigmoid

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

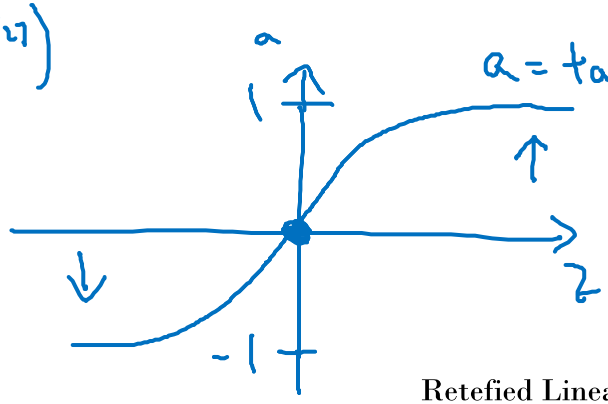


in binary classific u can use sigmoid for output layer → $y \in \{0,1\}$

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

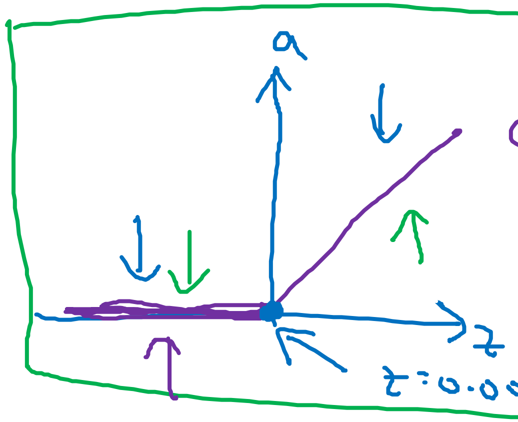
-1 1

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



this works better than the sigmuid fct

Retefied Linear Unit (very popular) good for hidden layer



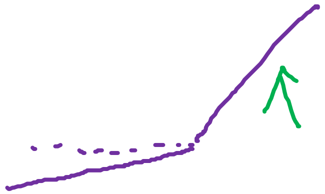
$z > 0$

$a = \max(0, z)$

ReLU

Rectified Linear Unit

NOT CLEAR WHY THIS IS BETTER!!!



Leaky ReLU

this works better but its not used in practice.

Given x:

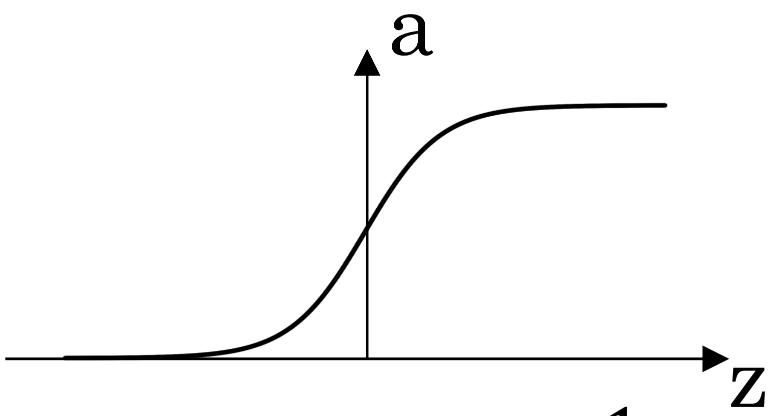
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

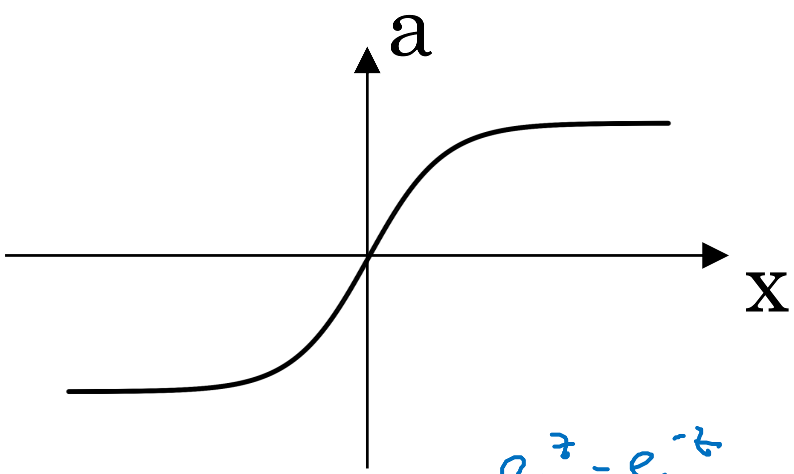
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

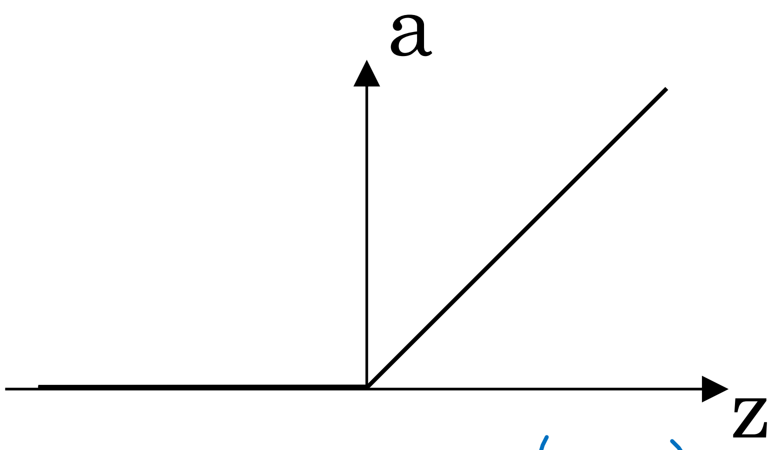
Pros and cons of activation functions



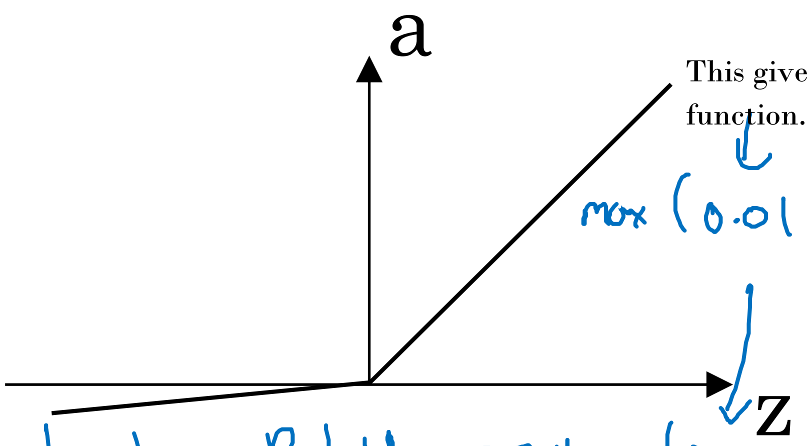
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



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



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



This gives that bent on the function.

$\max(0.01z, z)$

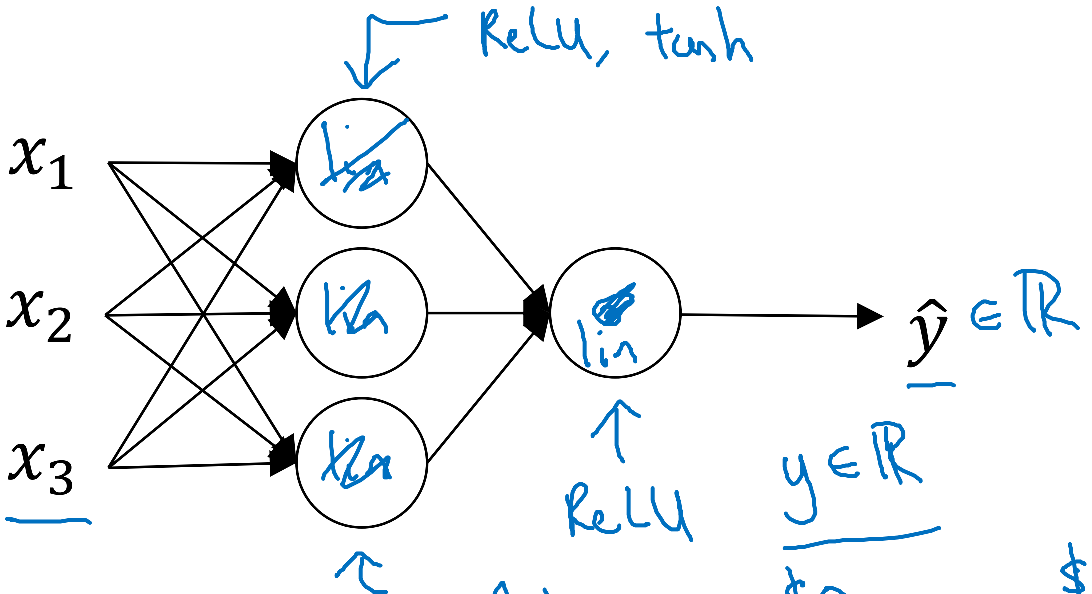
Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



$y \in \mathbb{R}$

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

hidden linear act fct layer is usles.

$\hat{y} \geq 0.$

Given x:

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

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

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

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

for a prob like housing price predict
you can use hidden relu and output layer a
lin activat function

$g(z) = z$

"linear activation function"

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

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

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

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

$$= W'x + b'$$

$g(z) = z$

if u use a linear activation function, no matter how many layers
all its doing is computing a linear activation fct.

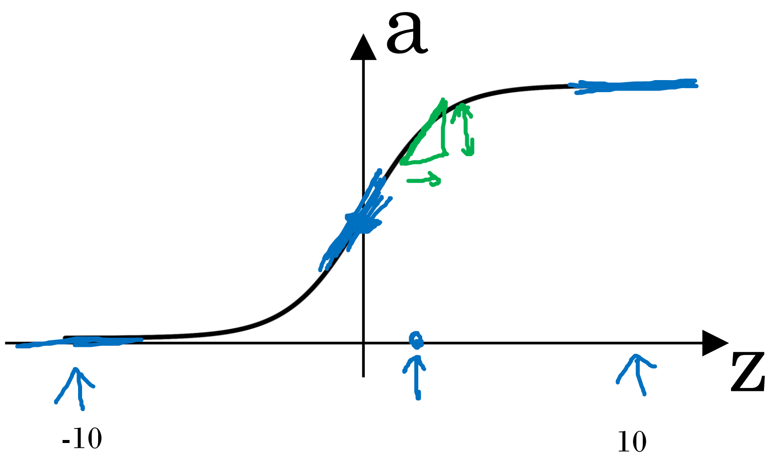
deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

When you compute backpropagation u need to compute derivative or slope of the activation function

Sigmoid activation function



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

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

Thats check that is the slope of the sigmoid fct:

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

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

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

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

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

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

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

short hand denotation

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

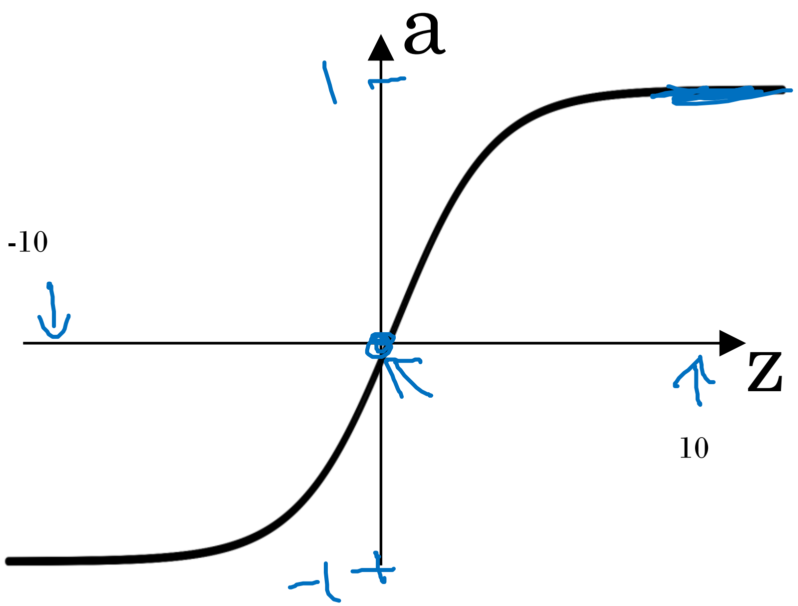
$$= \underbrace{g(z)\,(1-g(z))} \leftarrow$$

$$= \boxed{a\,(1-a)}$$

$$g'(z) = a(1-a)$$
$$\uparrow$$

a is simoid fct

Tanh activation function



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

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

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

$$= \underline{1 - (\underline{\tanh}(z))^2} \leftarrow$$

$$a = g(z), \quad \boxed{\underset{\uparrow}{g'}(z) = 1 - \underset{\uparrow}{a^2}}$$

$z = 10 \quad \tanh(z) \approx 1$

$g'(z) \approx 0$

$z = -10 \quad \tanh(z) \approx -1$

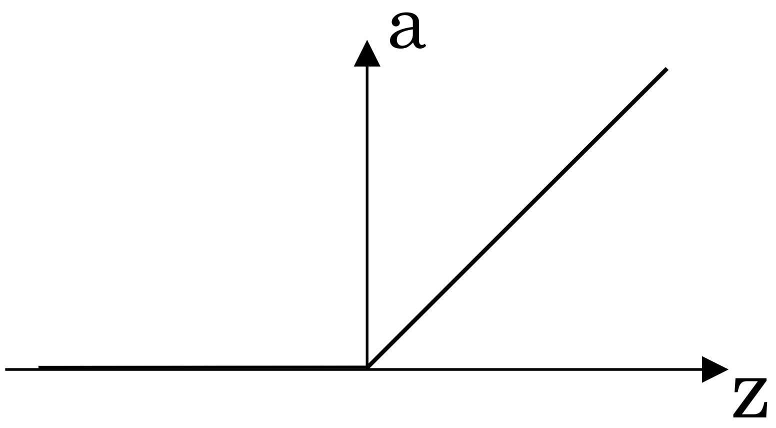
$g'(z) \approx 0$

$z = 0 \quad \tanh(z) = 0$

$g'(z) = 1$

if you have computed the value of a u can very quickly compute the derivative with that formula

ReLU and Leaky ReLU



ReLU

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

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

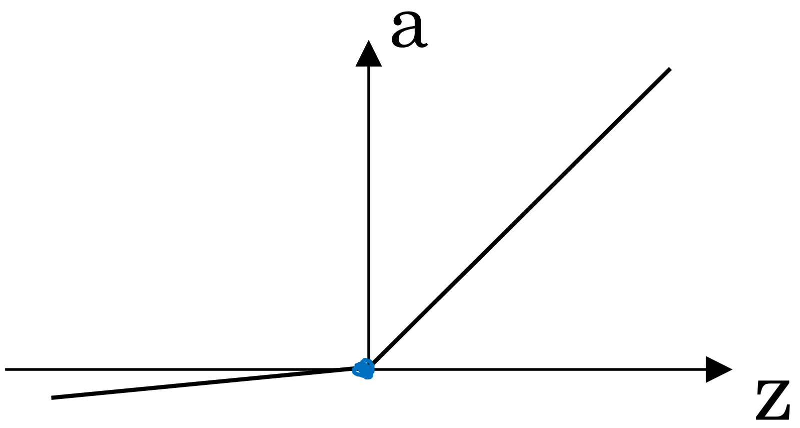
~~undefd if z = 0~~

$z = 0.0000\ldots\ldots 0$

its undefined for z= 0 but u can set it to 1 or 0

it does not matter

This works better but its not much used.



Leaky ReLU

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

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

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Here we will see how to implement gradient descent for the NN with one hidden layer, next we gone see why these equations work.

Gradient descent for neural networks

Parameters: $\underset{(n^{[1]}, n^{[0]})}{W^{[1]}}, \underset{(n^{[1]}, 1)}{b^{[1]}}, \underset{(n^{[2]}, n^{[1]})}{W^{[2]}}, \underset{(n^{[2]}, 1)}{b^{[2]}}$

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

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

This is the cost function and we are assuming we are doing binary classification.

Gradient descent:

$\rightarrow$ Repeat {

we saw previewsly how to compute these predictions $\rightarrow$ Compute predicts $(\hat{y}^{(i)}, i = 1 \dots, m)$

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

the key is to compute these derivative terms.

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

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

$W^{[2]} := \dots \qquad b^{[2]} := \dots$

}

VERY IMPORTANT FOR IMPLEMENTATION OF A NN

Formulas for computing derivatives

This is not trivial!!!

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:

$dZ^{[2]} = A^{[2]} - Y$ ↙

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

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

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

this prevents python from outputing $(n^{[2]},)$ ← it avoid to output this funny rank array

$(n^{[2]}, 1)$ ←

summing horizontaly in that matrix

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

↖ element-wise product

elementwise product of two matrixes

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

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

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

Andrew Ng

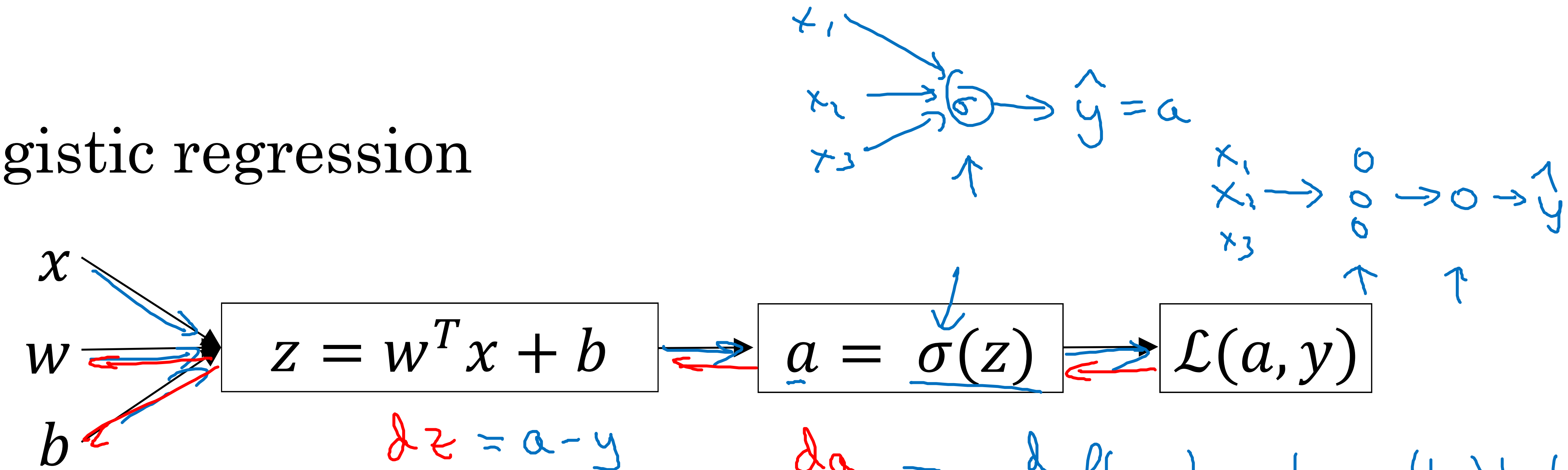
deeplearning.ai

One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

Logistic regression



$dz = a - y$

$da = \frac{d}{da}\mathcal{L}(a,y) = \underbrace{-y\log a - (1-y)\log(1-a)}$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$dw = dz \cdot x$

$db = dz$

$\boxed{dz = da \cdot g'(z)}$

$g(z) = \sigma(z)$

$\boxed{\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \boxed{\frac{da}{dz}}}$

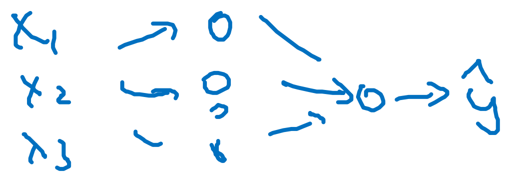
"dz" = "da"

$\frac{d}{dz} g(z) = \underline{g'(z)}$

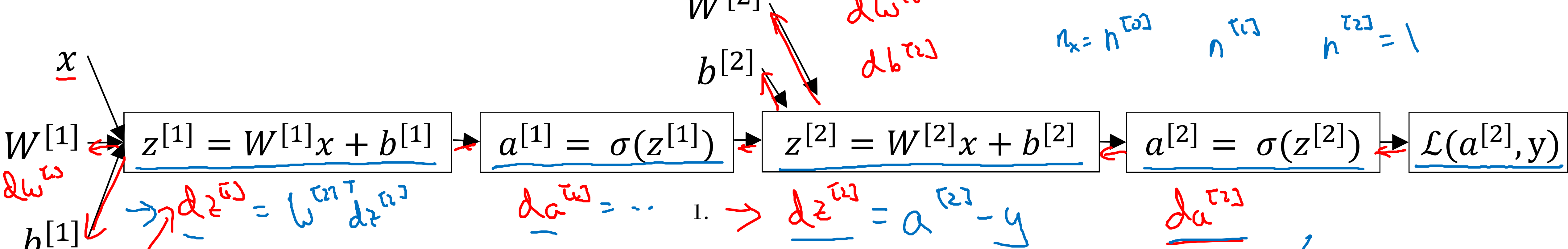
We did not explained but it turns out its like this.

This is the reason (chain rule)

Neural network gradients



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



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

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

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

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

$\mathcal{L}(a^{[2]}, y)$

4. $dz^{[1]} = W^{[2]T} dz^{[2]}$ $* \, g^{[1]'}(z^{[1]})$

element wise product

$da^{[1]} = \ldots$

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

$da^{[2]}$

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

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

" $dw = dz \cdot x$ "

$w^{[1]}: [\_\_\_\_\_]$

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

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

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

$$dz^{[1]} = \boxed{W^{[2]T} dz^{[2]}} * \boxed{g^{[1]'}(z^{[1]})}$$

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

foo

dfoo

$W^{[1]}$

$dW^{[2]}$

5. $dW^{[1]} = dz^{[1]} \cdot x^T$

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

$a^{[0]T}$

There you have 6 eq to implement successfully a NN.

Summary of gradient descent

So here are the 6 equation. They are just for one training example.

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

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

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

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

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

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

Here we vectorize the implementation for m training examples.

Vectorized Implementation:

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

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

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

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

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

Summary of gradient descent

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

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

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

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

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

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

Here are the vectorized implementation for m training examples:

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

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

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

There is this extra 1/m becouse the cost fct is the above

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

elementwise product

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

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

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

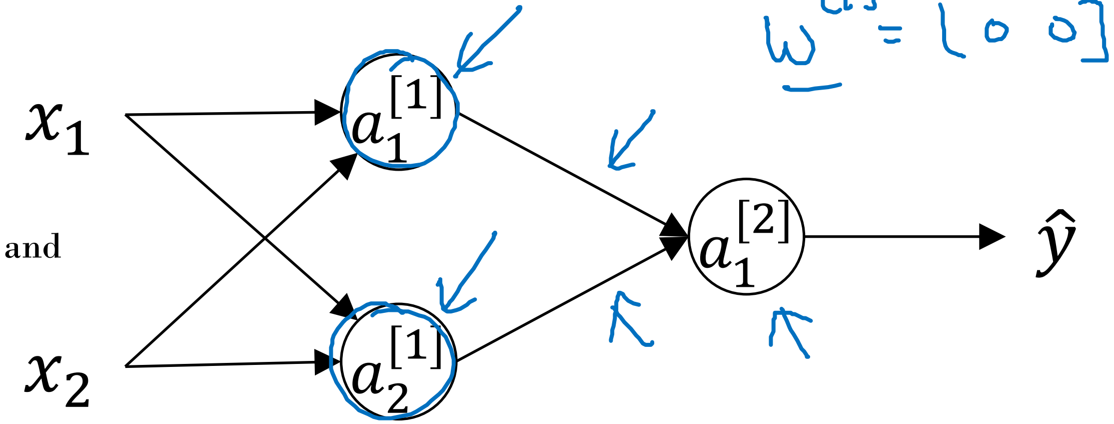
One hidden layer Neural Network

Random Initialization

deeplearning.ai

What happens if you initialize weights to zero?

For a logistic reg is fine to initialize weights to 0 but for a neural netwrok it wont work.

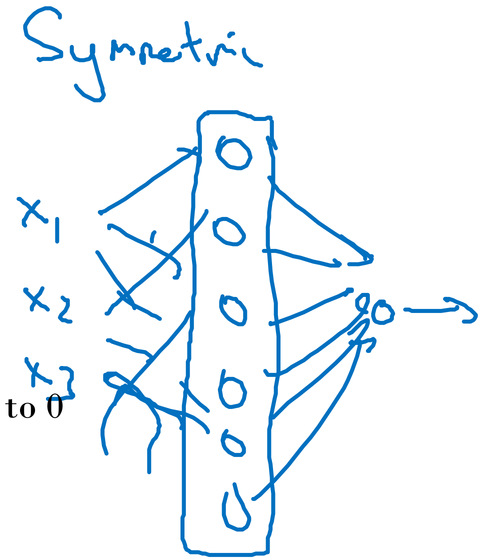


$W^{[2]} = [0 \quad 0]$

two features and two hidden layers.

$n^{[0]} = 2$ $n^{[1]} = 2$

This is also called the Symetry breaking problem.



say we initialize by 0 $W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

and say also b is 0 0, and this is ok to initialize to 0 but not for W.

For any example that you give it you have: $a_1^{[1]} = a_2^{[1]}$ Then when you compute backprop $dz_1^{[1]} = dz_2^{[1]}$

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$

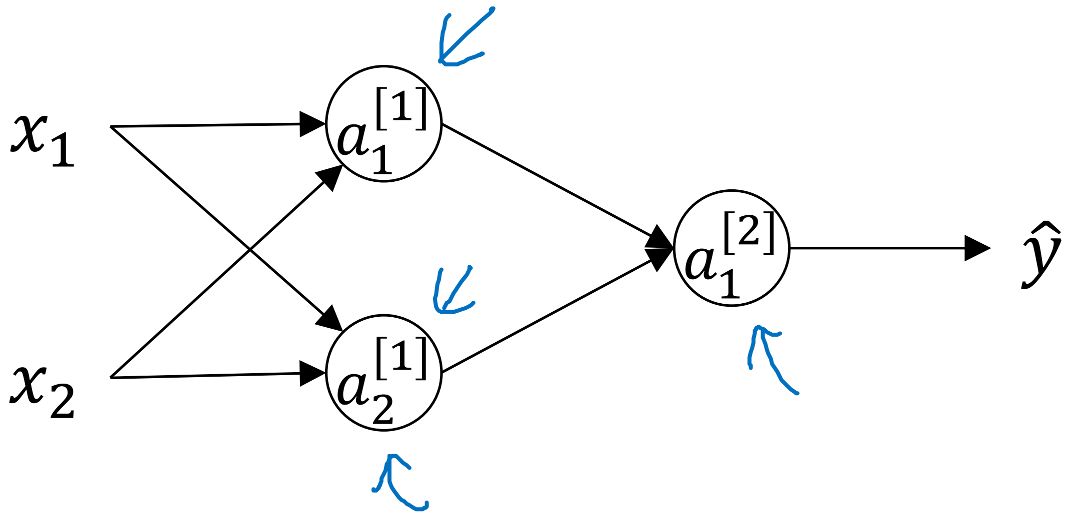
$W^{[1]} = W^{[1]} - \alpha dW$

It turns out that after every iteration the hidden units will be the same.

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

it will end up with first row equal to second row.

Random initialization



we pref to init weight to very small random numbers. In certain cases u might want to choose a different constant, we will talk about this next week.

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

100 ?

this is fine to init with 0

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

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

$b^{[2]} = 0$

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

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