

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

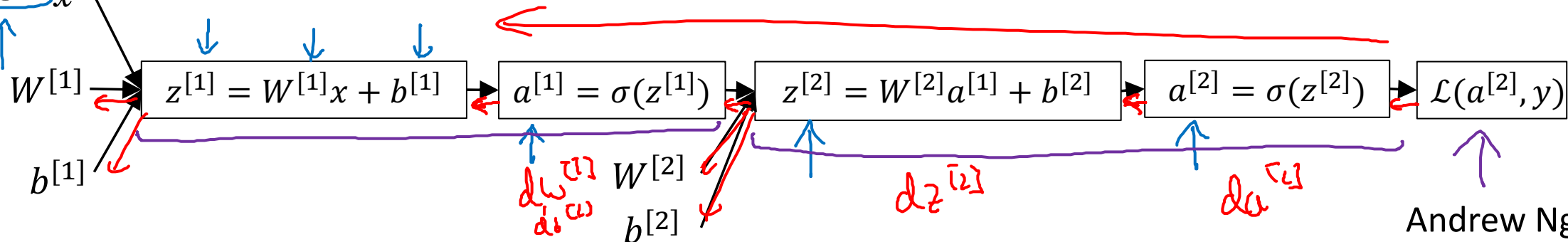
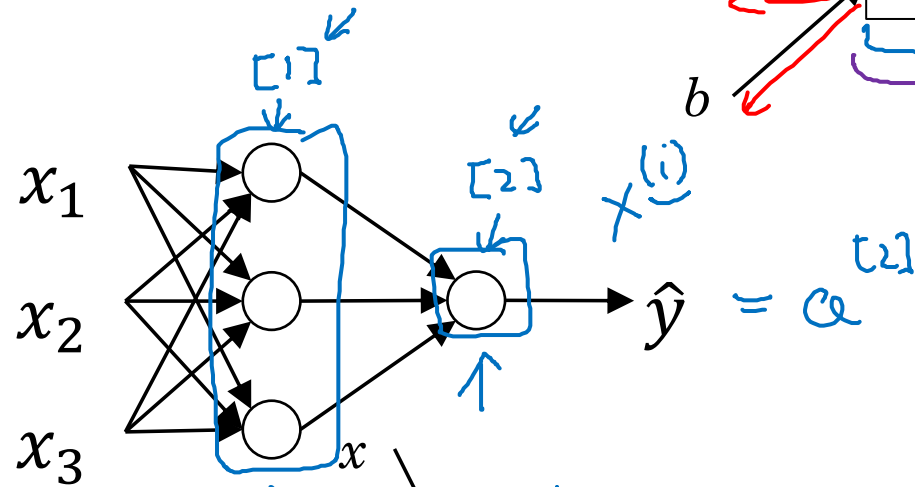
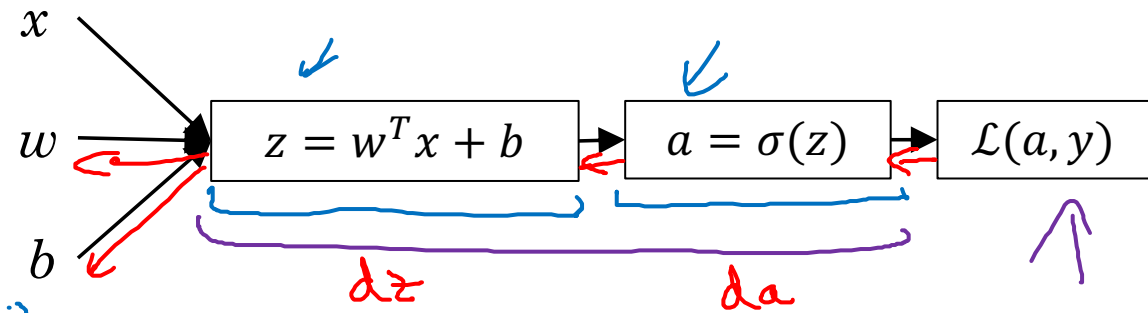
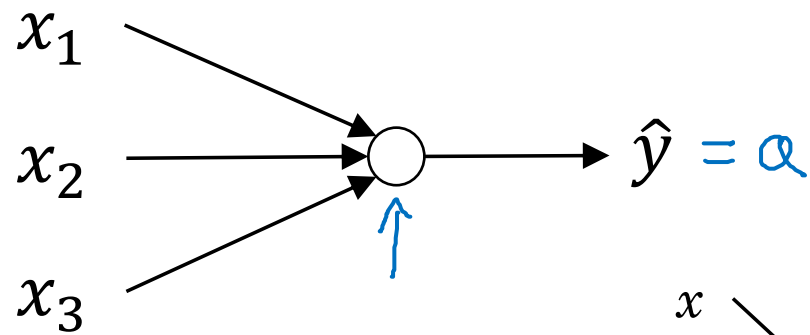


deeplearning.ai

One hidden layer
Neural Network

Neural Networks Overview

What is a Neural Network?



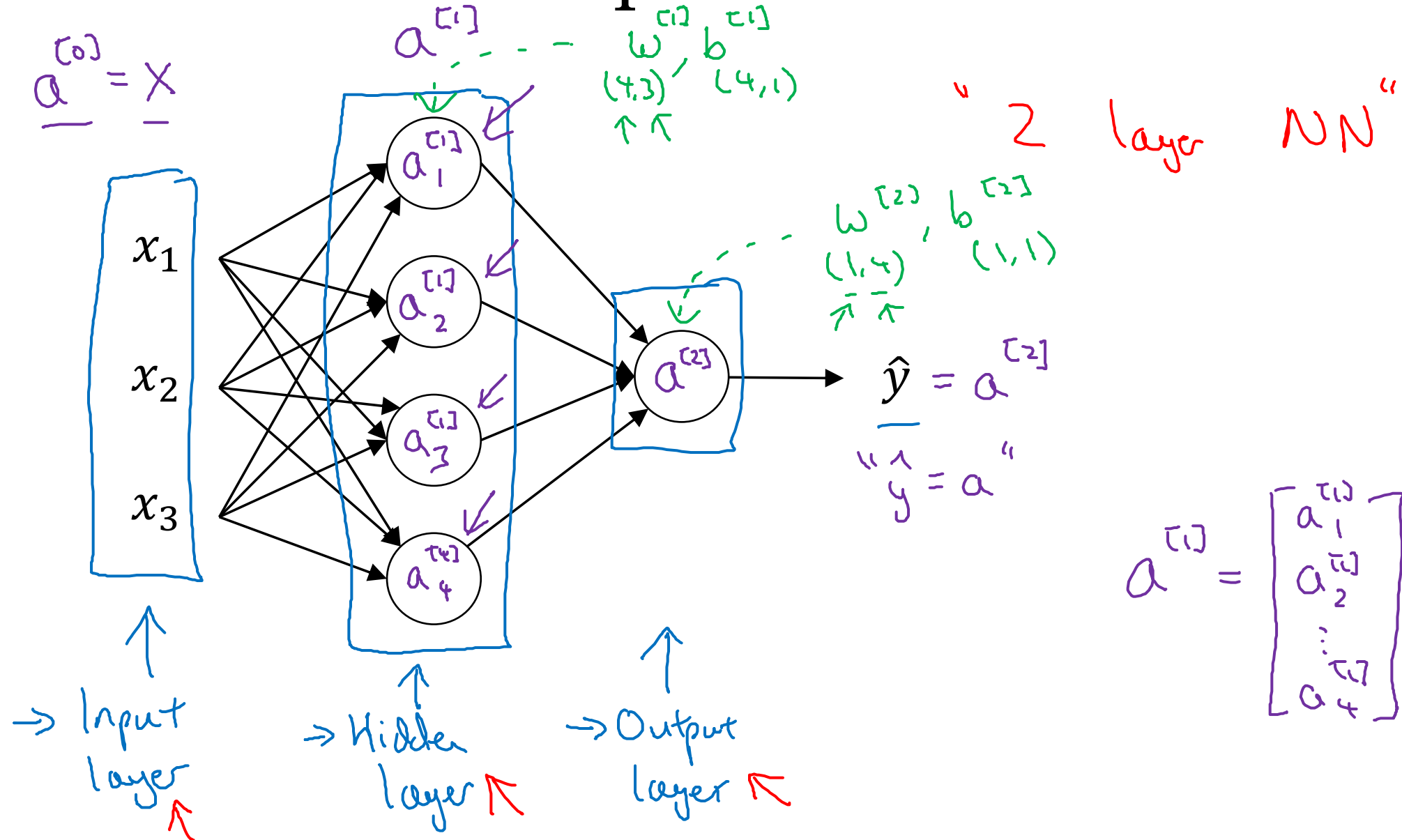


deeplearning.ai

One hidden layer
Neural Network

Neural Network
Representation

Neural Network Representation



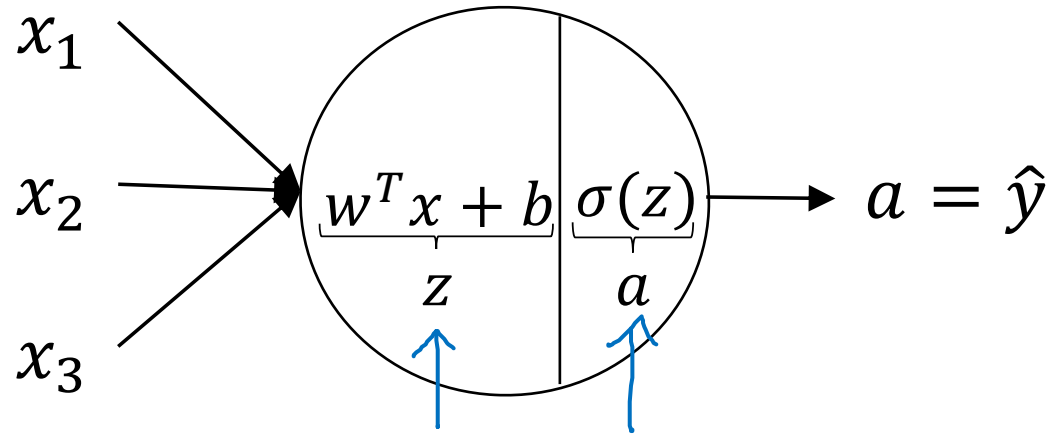


deeplearning.ai

One hidden layer Neural Network

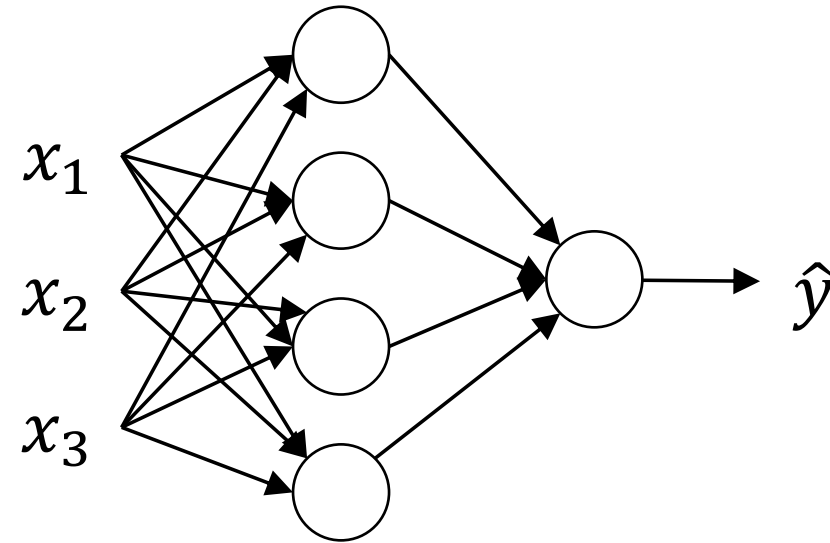
Computing a
Neural Network's
Output

Neural Network Representation

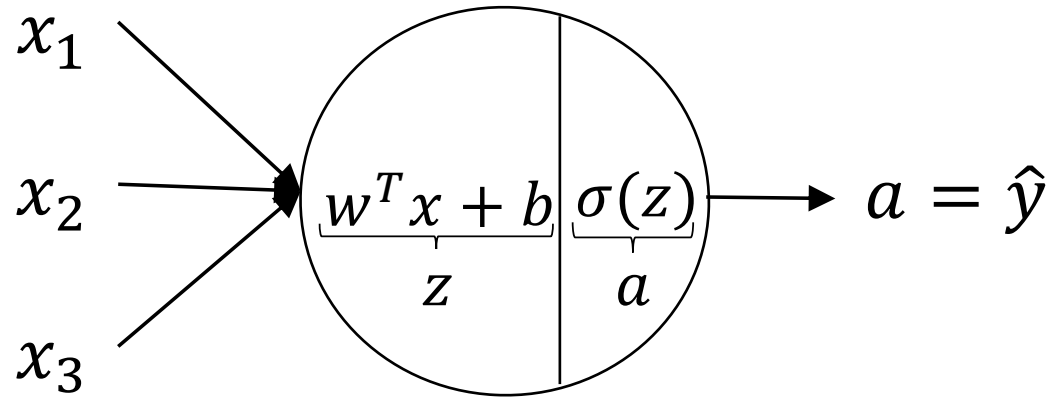


$$z = w^T x + b$$

$$a = \sigma(z)$$

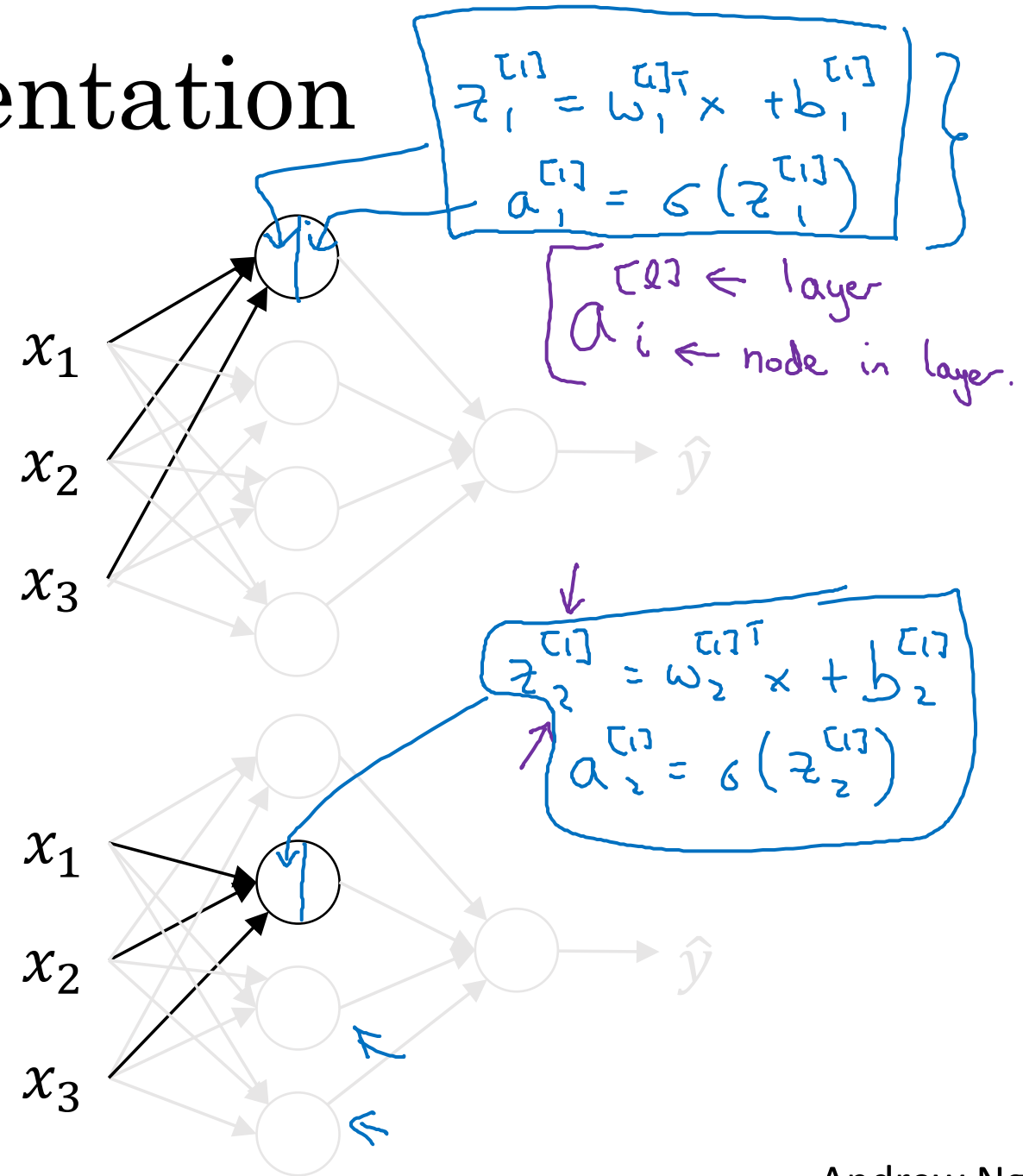


Neural Network Representation

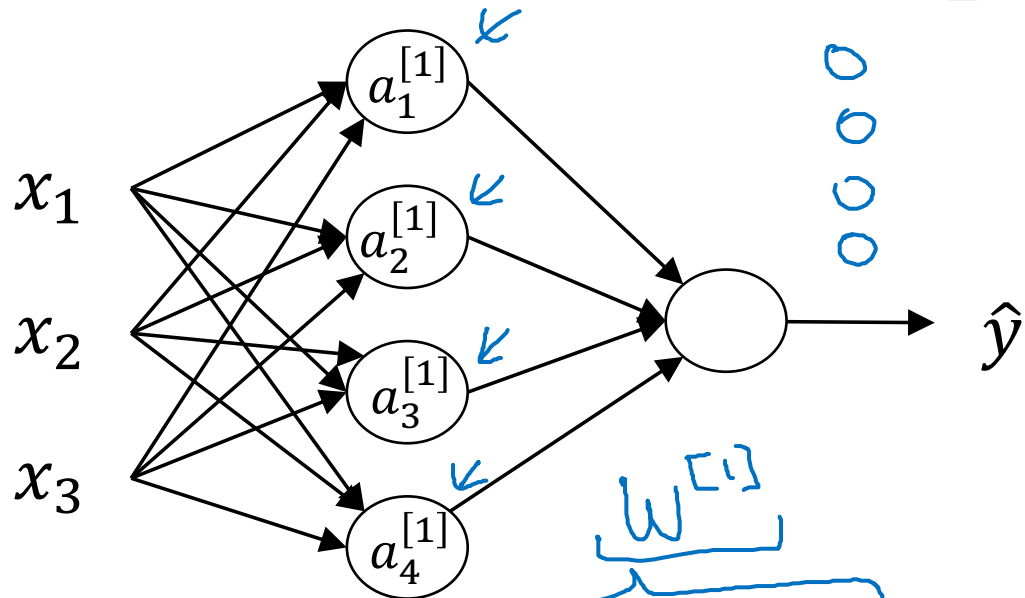


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation

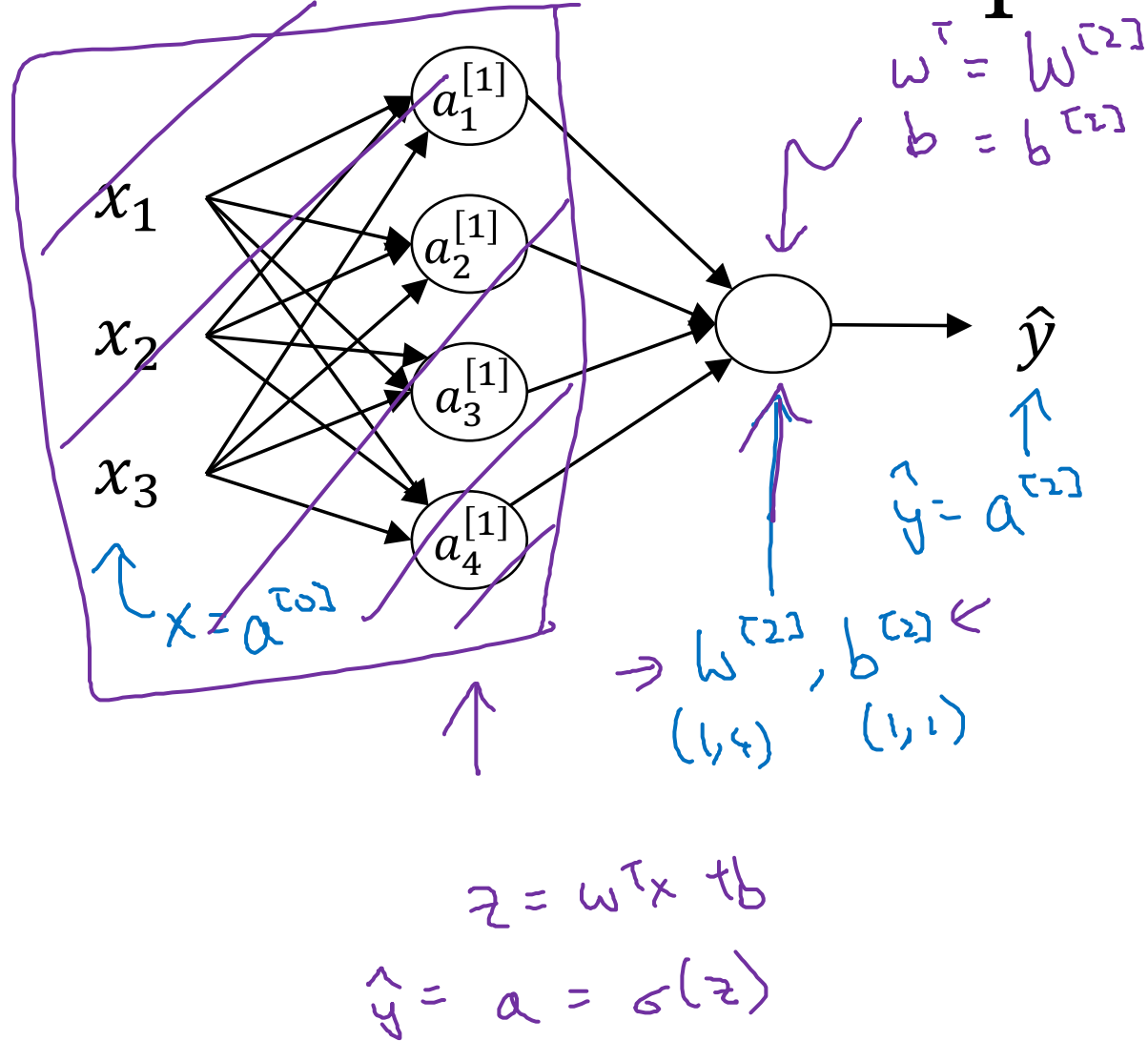


$$\begin{aligned} z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} & a_1^{[1]} &= \sigma(z_1^{[1]}) \\ z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} & a_2^{[1]} &= \sigma(z_2^{[1]}) \\ z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]} & a_3^{[1]} &= \sigma(z_3^{[1]}) \\ z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]} & a_4^{[1]} &= \sigma(z_4^{[1]})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

$$\begin{aligned} \rightarrow z^{[1]} &= \begin{bmatrix} -w_1^{[1]T} \\ -w_2^{[1]T} \\ -w_3^{[1]T} \\ -w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} \rightarrow w_1^{[1]T} x + b_1^{[1]} \\ \rightarrow w_2^{[1]T} x + b_2^{[1]} \\ \rightarrow w_3^{[1]T} x + b_3^{[1]} \\ \rightarrow w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} \\ \rightarrow a^{[1]} &= \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]}) \end{aligned}$$

Dimensions: $W^{[1]} (4,3)$, $b^{[1]} (4,1)$, $z^{[1]} (4,1)$, $a^{[1]} (4,1)$.

Neural Network Representation learning



Given input x :

$$\begin{aligned} \rightarrow z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\ &\quad (4,1) \quad (4,3) \quad (3,1) \quad (4,1) \\ \rightarrow a^{[1]} &= \sigma(z^{[1]}) \\ &\quad (4,1) \quad (4,1) \\ \rightarrow z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ &\quad (1,1) \quad (1,4) \quad (4,1) \quad (1,1) \\ \rightarrow a^{[2]} &= \sigma(z^{[2]}) \\ &\quad (1,1) \quad (1,1) \end{aligned}$$

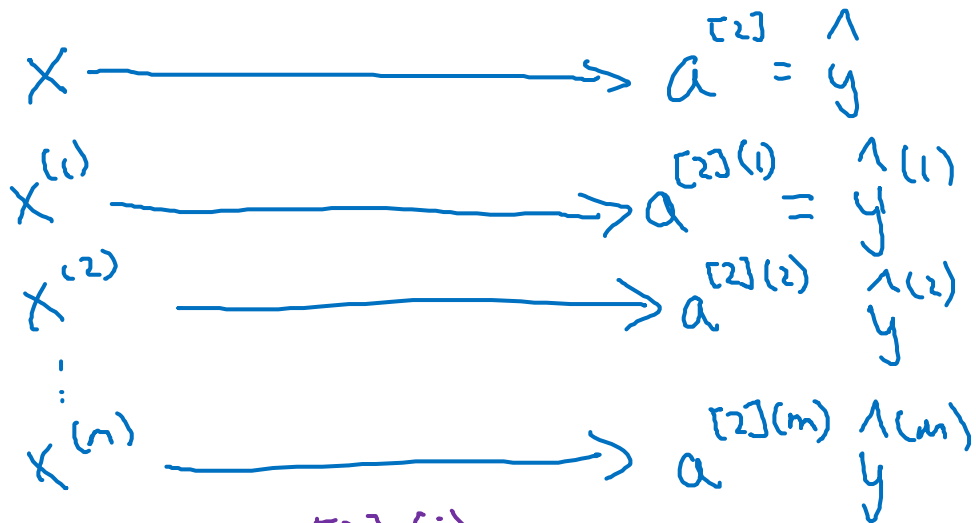
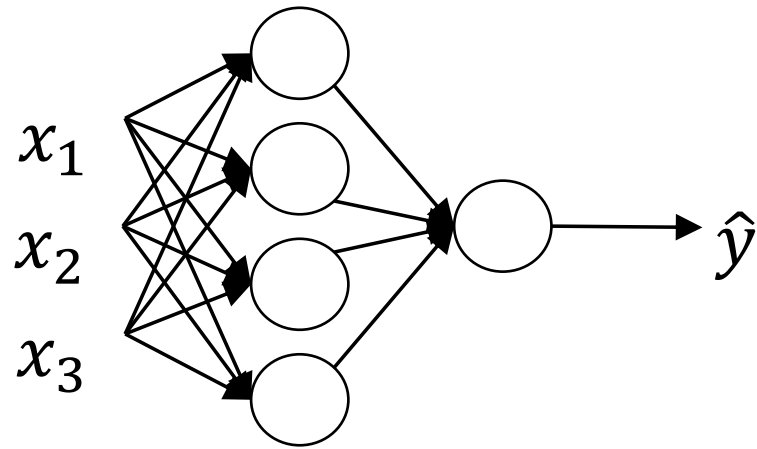


deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



$a^{[2](i)}$
 $\nwarrow \nearrow$ example i
 layer 2

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

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

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

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

for $i = 1$ to m ,

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

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

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

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

Vectorizing across multiple examples

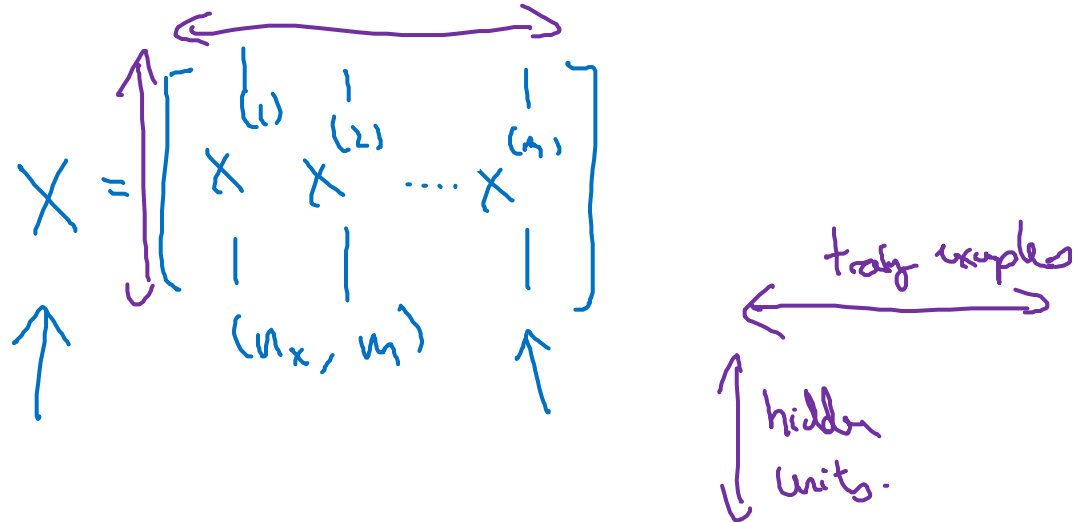
for $i = 1$ to m :

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

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

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

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

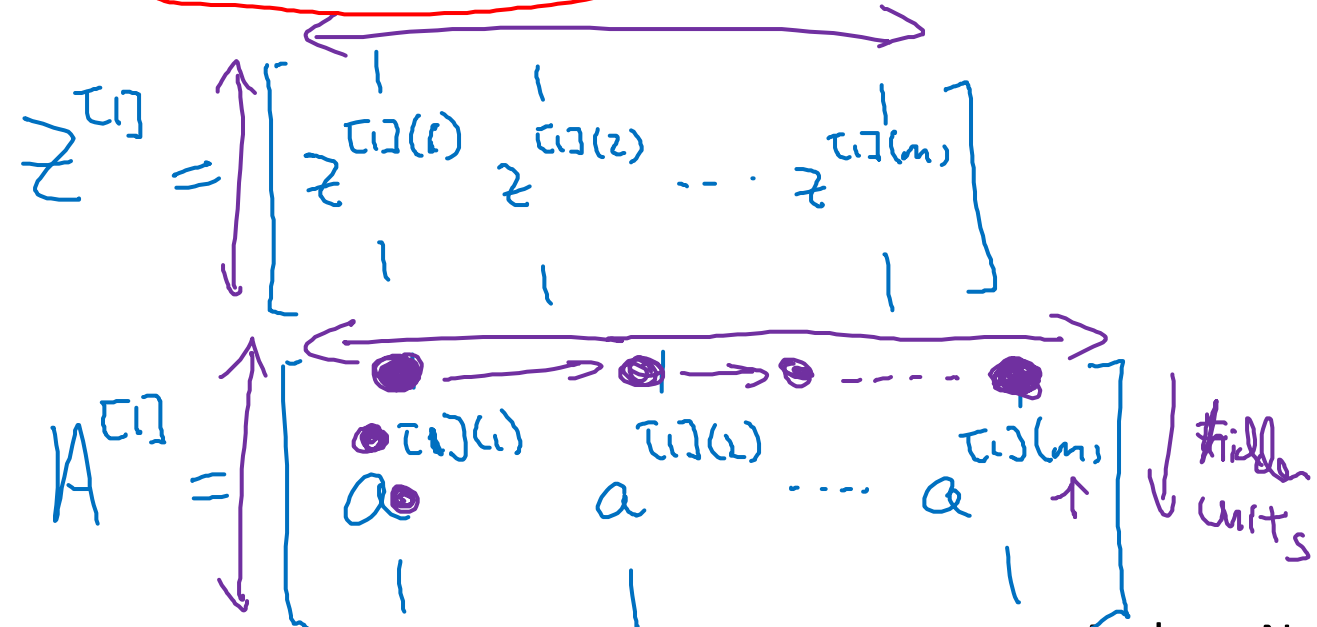


$$z^{[1]} = W^{[1]}X + b^{[1]}$$

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

$$\rightarrow z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

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





deeplearning.ai

One hidden layer Neural Network

Explanation
for vectorized
implementation

Justification for vectorized implementation

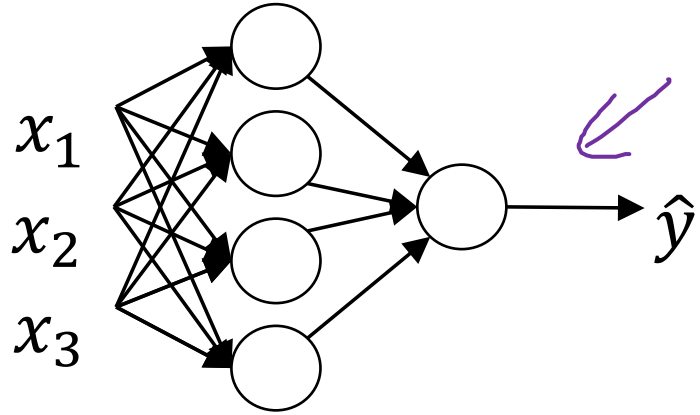
$$z^{1} = \underbrace{w^{[1]} x^{(1)}}_{\text{purple}} + \cancel{b^{[1]}}_{\text{red}}, \quad z^{[1](2)} = \underbrace{w^{[1]} x^{(2)}}_{\text{green}} + \cancel{b^{[1]}}_{\text{red}}, \quad z^{[1](3)} = \underbrace{w^{[1]} x^{(3)}}_{\text{yellow}} + \cancel{b^{[1]}}_{\text{red}}$$

$$w^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \quad w^{[1]} x^{(1)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \quad w^{[1]} x^{(2)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \quad w^{[1]} x^{(3)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

$$w^{[1]} \begin{bmatrix} | & | & | & \dots \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots \\ | & | & | & \dots \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet & \dots \\ \bullet & \bullet & \bullet & \dots \\ \bullet & \bullet & \bullet & \dots \\ \bullet & \bullet & \bullet & \dots \end{bmatrix} = \begin{bmatrix} | & | & | & \dots \\ z^{1} & z^{[1](2)} & z^{[1](3)} & \dots \\ | & | & | & \dots \end{bmatrix} = z^{[1]}$$

$\hat{X} \quad w^{[1]} x^{(i)} = z^{[1](i)} \quad \begin{matrix} \uparrow \\ + b^{[1]} \end{matrix} \quad \begin{matrix} \uparrow \\ + b^{[1]} \end{matrix} \quad \begin{matrix} \uparrow \\ + b^{[1]} \end{matrix}$

Recap of vectorizing across multiple examples



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

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

for $i = 1$ to m

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

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

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

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

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

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

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

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

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

$$W^{[1]}A^{[0]} + b^{[1]}$$

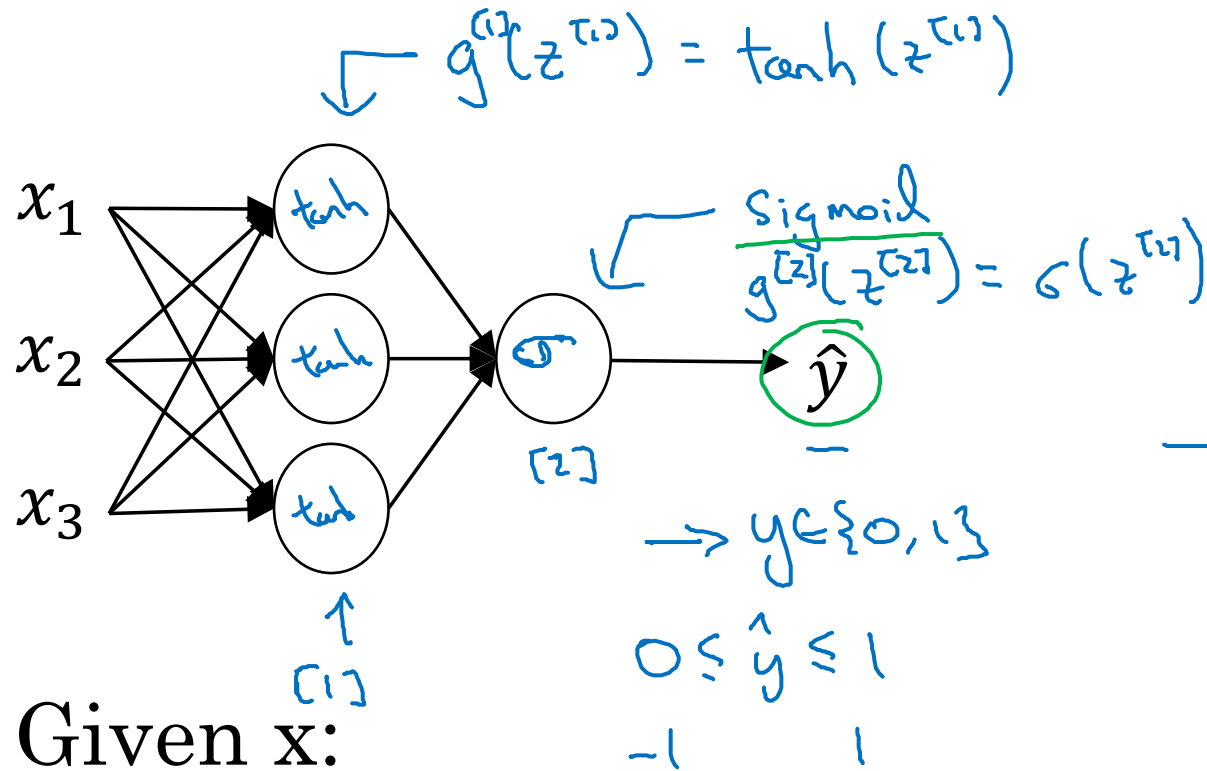


deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions



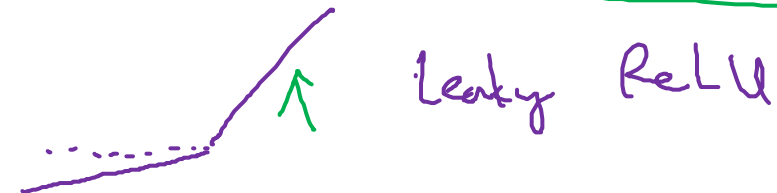
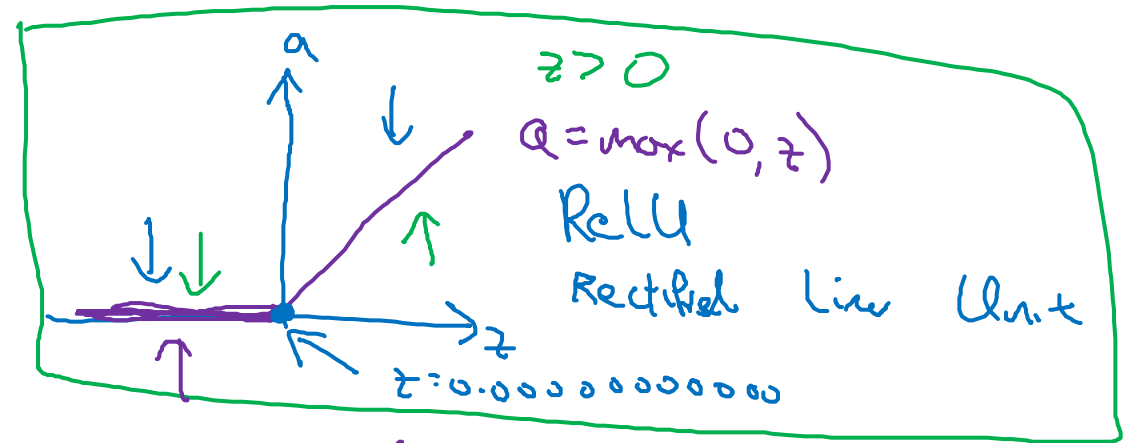
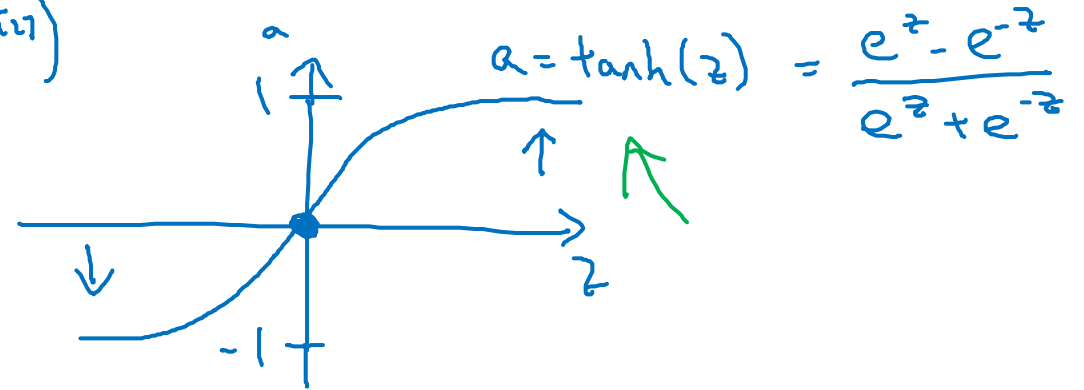
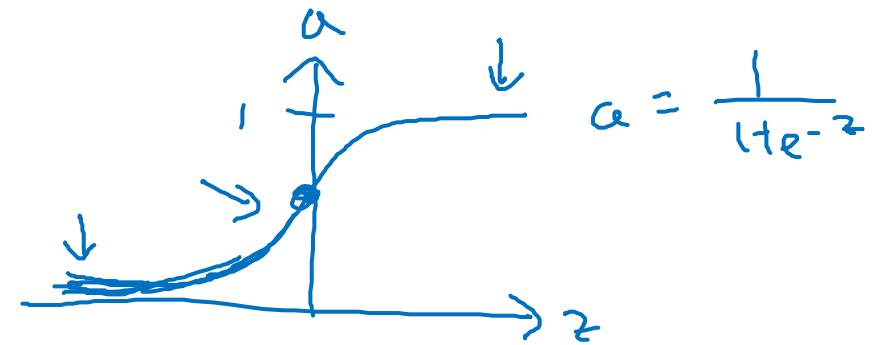
Given x :

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

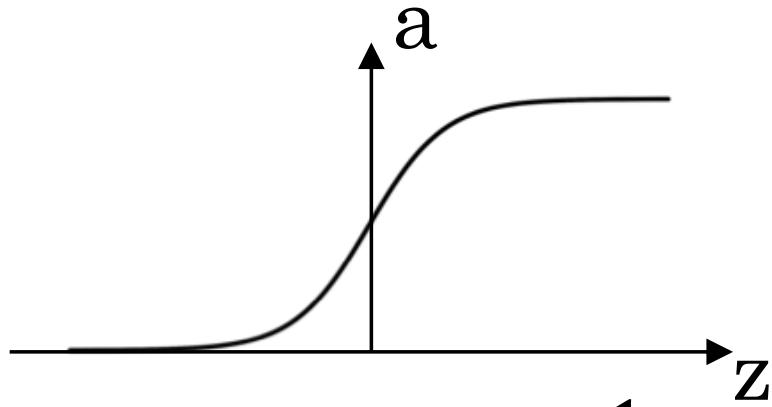
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})} g^{(1)}(z^{(1)})$$

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

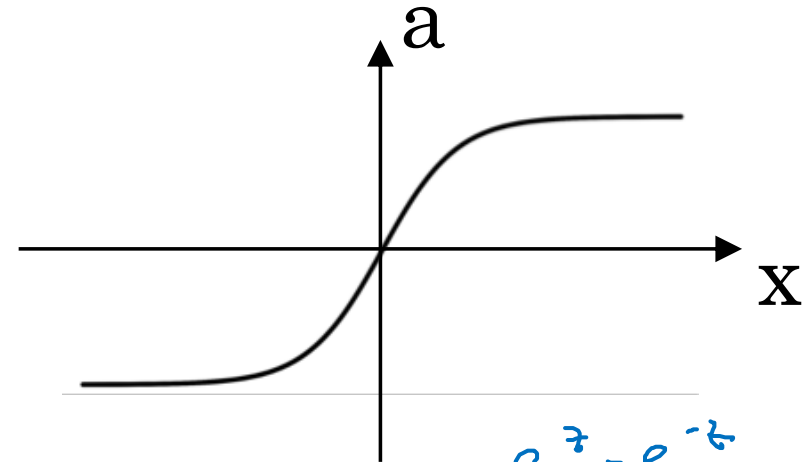
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})} g^{(2)}(z^{(2)})$$



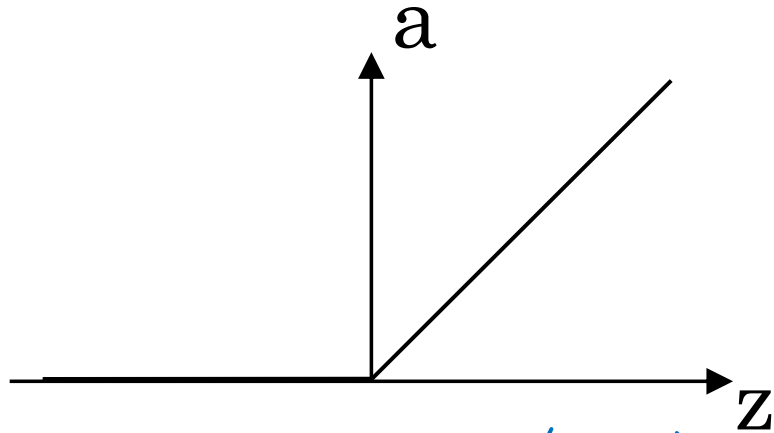
Pros and cons of activation functions



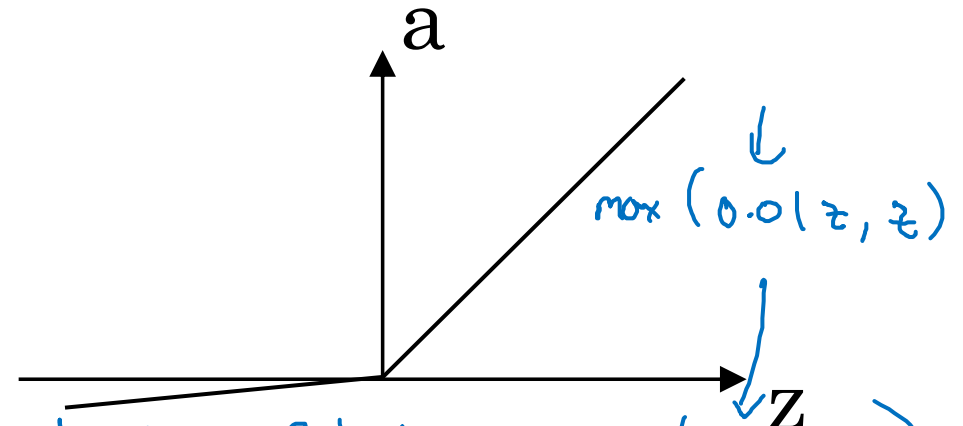
sigmoid: $a = \frac{1}{1 + e^{-z}}$



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



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



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

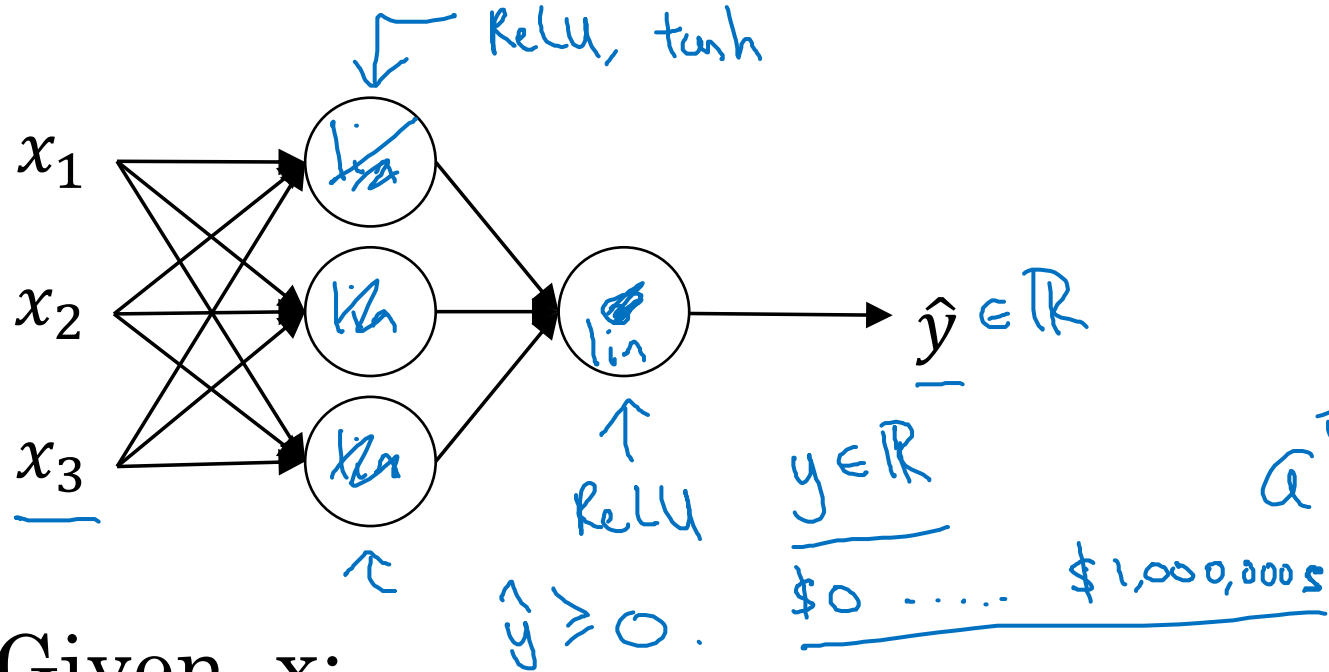


deeplearning.ai

One hidden layer Neural Network

Why do you
need non-linear
activation functions?

Activation function



Given x :

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

$g(z) = z$
"linear activation function"

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

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

$$\begin{aligned} &= \underbrace{\left(W^{[2]} W^{[1]} \right)}_{w'} x + \underbrace{\left(W^{[2]} b^{[1]} + b^{[2]} \right)}_{b'} \\ &= \underline{w'x + b'} \end{aligned}$$

$$g(z) = z$$

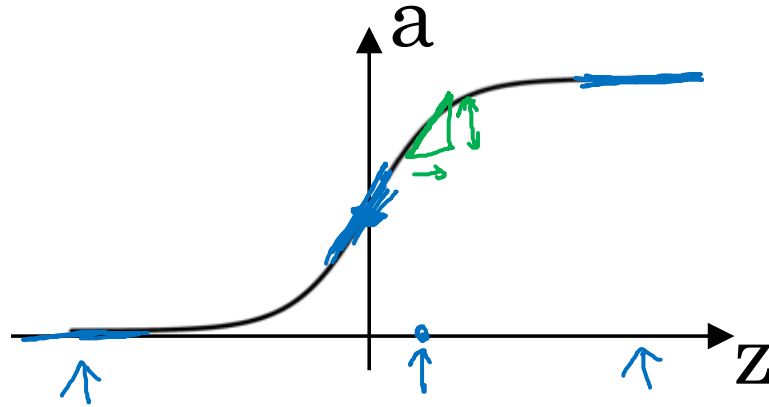


deeplearning.ai

One hidden layer
Neural Network

Derivatives of
activation functions

Sigmoid activation function



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

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

$$\begin{aligned} \boxed{g'(z)} &= \boxed{\frac{d}{dz} g(z)} = \text{slope of } g(z) \text{ at } z \\ &= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right) \\ &= g(z) (1 - g(z)) \leftarrow \\ &= \boxed{a(1-a)} \quad \left| \begin{array}{l} g'(z) = a(1-a) \\ \uparrow \\ a \end{array} \right. \end{aligned}$$

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

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

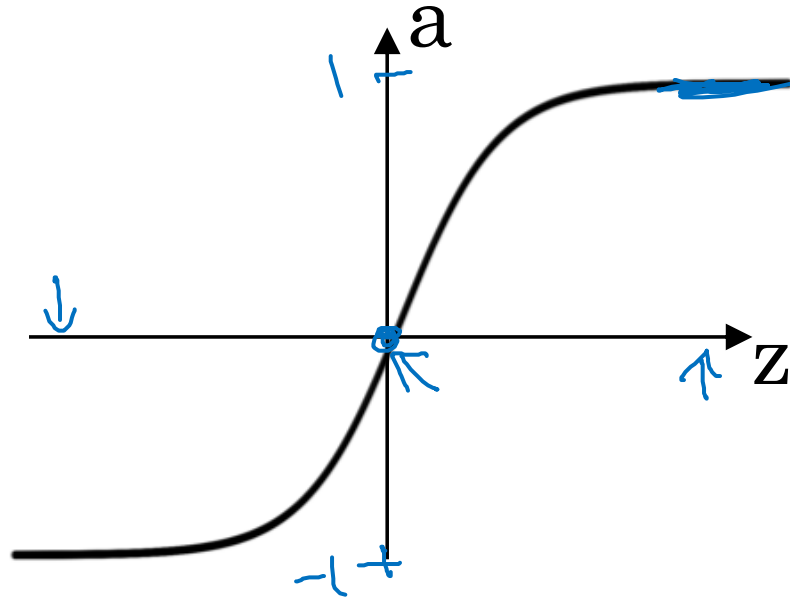
$$z = -10 \quad g(z) \approx 0$$

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

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

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

Tanh activation function



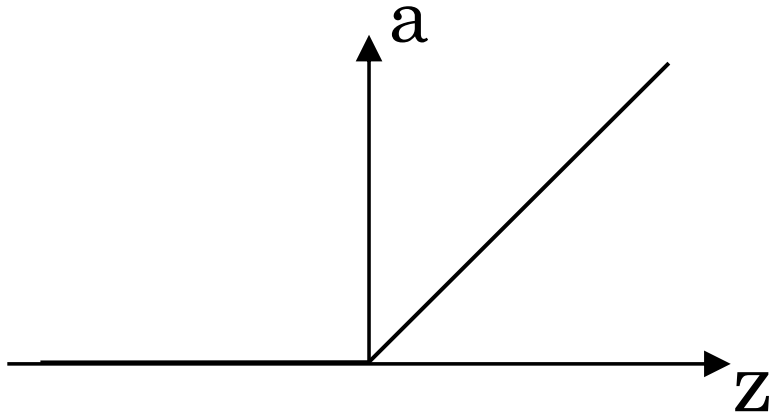
$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z = \underline{1 - (\tanh(z))^2} \leftarrow$$

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

$$\left| \begin{array}{ll} z=10 & \tanh(z) \approx 1 \\ & g'(z) \approx 0 \\ z=-10 & \tanh(z) \approx -1 \\ & g'(z) \approx 0 \\ z=0 & \tanh(z) = 0 \\ & g'(z) = 1 \end{array} \right.$$

ReLU and Leaky ReLU



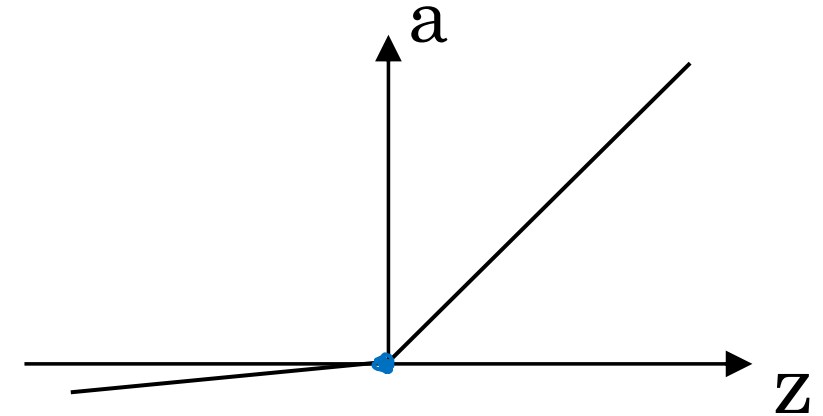
ReLU

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

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

~~$undefined$ if $z = 0$~~

$z = 0.0000...0$



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

Gradient descent for neural networks

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

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

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

\uparrow \uparrow \uparrow
 $a^{[2]}$

Gradient descent:

→ Repeat {

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

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

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

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

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

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:

$$dz^{[2]} = A^{[2]} - y \leftarrow$$

$$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$

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

$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} \times \underbrace{g^{[1]'}(z^{[1]})}_{\text{element-wise product}} \quad (n^{[1]}, m)$$

$$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$$

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

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

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

$$(n^{[2]}) \leftarrow$$

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



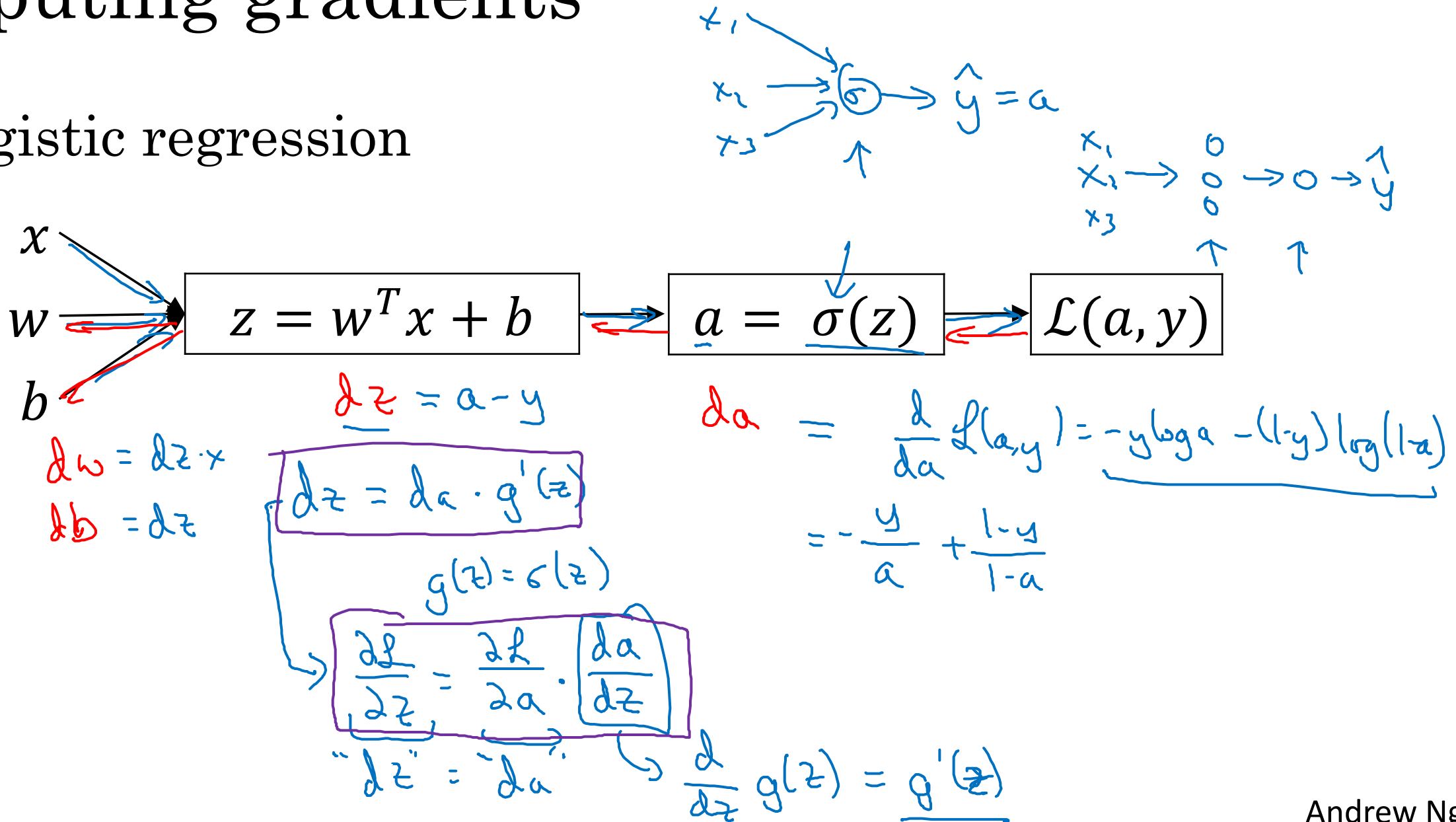
deeplearning.ai

One hidden layer
Neural Network

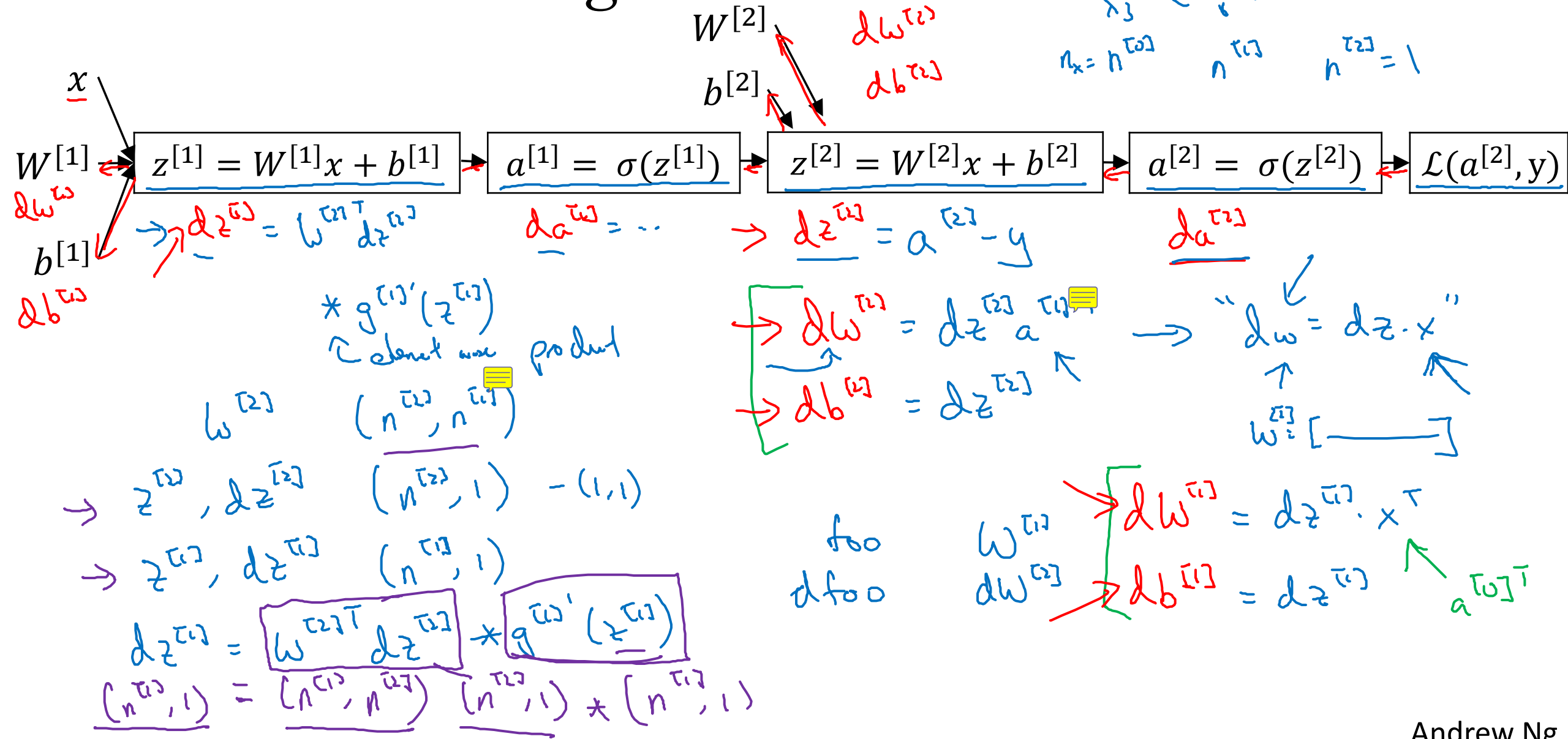
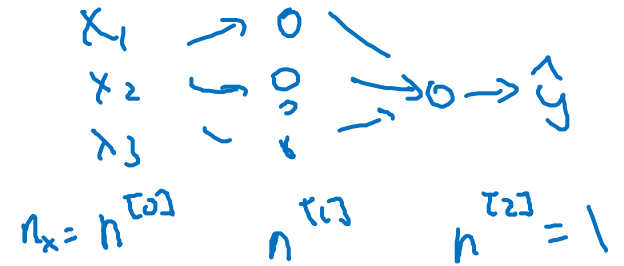
Backpropagation
intuition (Optional)

Computing gradients

Logistic regression



Neural network gradients



Summary of gradient descent

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

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

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

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

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

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

Vectorized Implementation:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[1]} = \begin{bmatrix} z^{1} \\ z^{[1](2)} \\ \dots \\ z^{[1](n)} \end{bmatrix}$$
$$z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$

Summary of gradient descent

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

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

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

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

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

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

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

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

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

$$\underbrace{dZ^{[1]}}_{(n^{[1]}, m)} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(Z^{[1]})}_{(n^{[1]}, m)}$$

↙ elementwise product

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

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

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

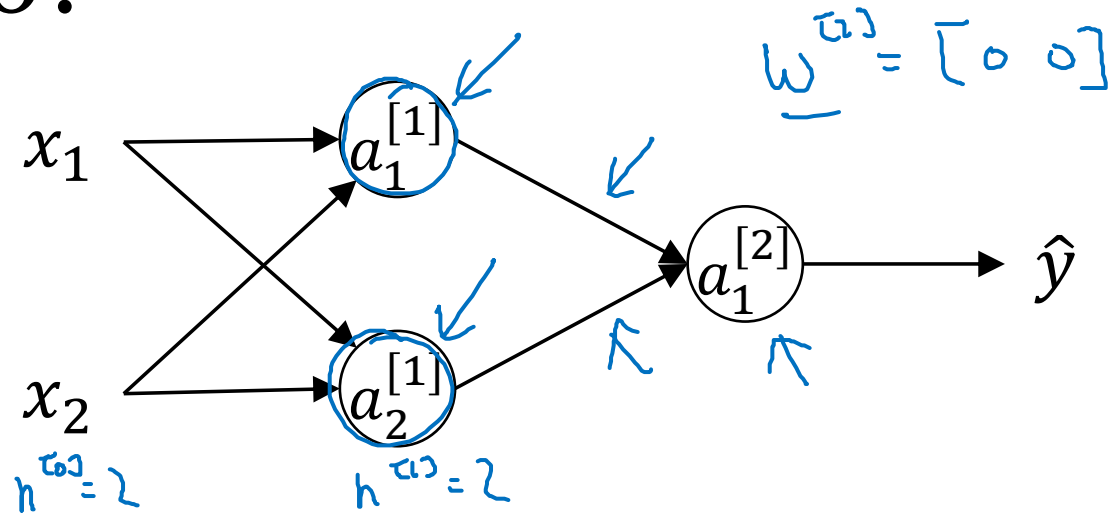


deeplearning.ai

One hidden layer
Neural Network

Random Initialization

What happens if you initialize weights to zero?



$$\underline{w}^{(1)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$a_1^{(1)} = a_2^{(1)}$$

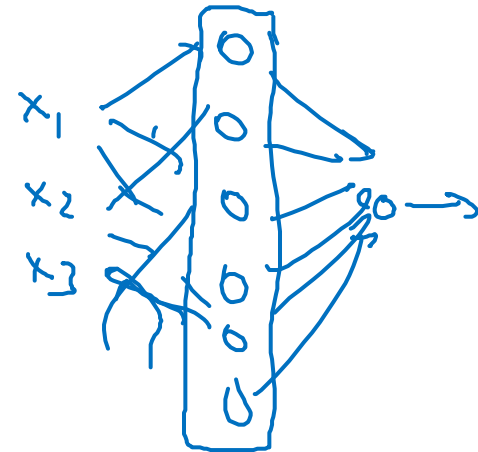
$$\underline{dw} = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

$$\underline{b}^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$dz_1^{(1)} = dz_2^{(1)}$$

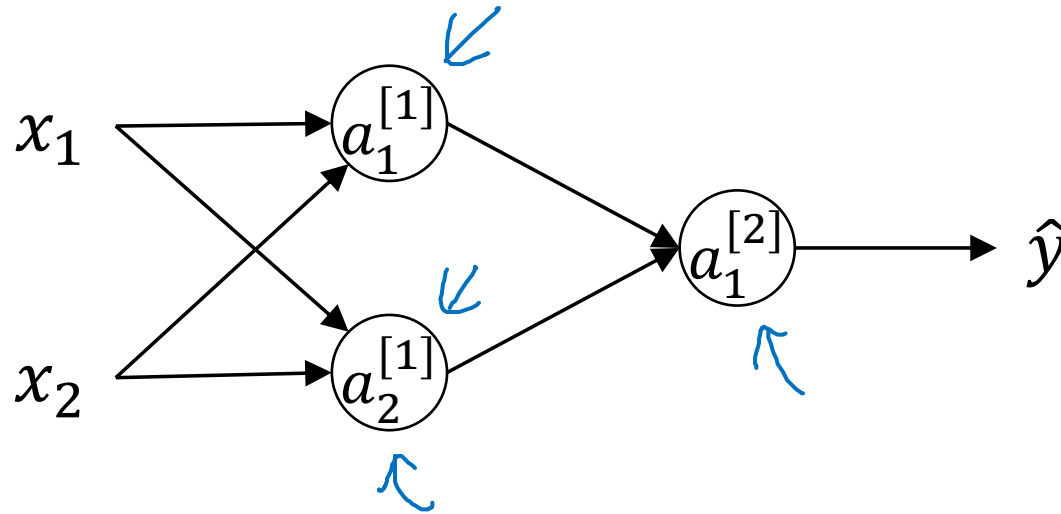
$$\underline{w}^{(1)} = \underline{w}^{(1)} - \underline{dw}$$

Symmetric

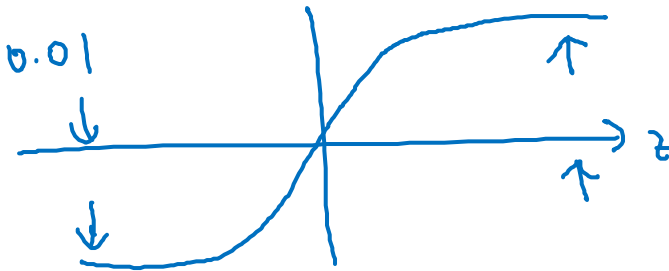


$$\underline{w}^{(1)} = \begin{bmatrix} \dots & \cdot \\ \dots & \cdot \end{bmatrix}$$

Random initialization



→ $w^{[1]} = \text{np.random.randn}(2,2) * \frac{0.01}{100?}$
 $b^{[1]} = \text{np.zeros}(2,1)$
 $w^{[2]} = \text{np.random.randn}(1,2) * 0.01$
 $b^{[2]} = 0$



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