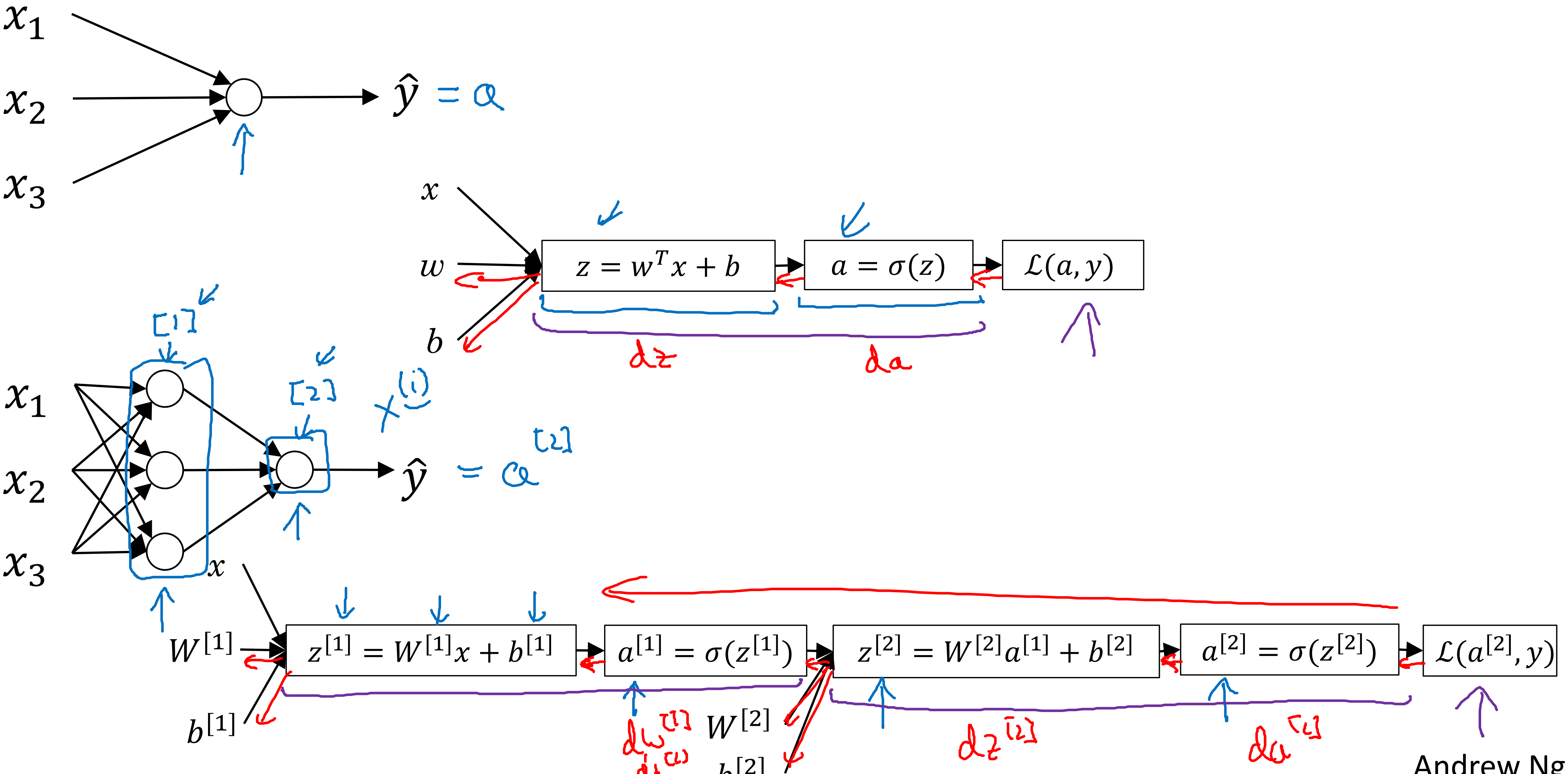
deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?



x_1

x_2

x_3

$\hat{y} = a$

x

w

b

$z = w^T x + b$

$a = \sigma(z)$

$\mathcal{L}(a, y)$

dz

da

$[1]$

$[2]$

$x^{(i)}$

x_1

x_2

x_3

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

x

$W^{[1]}$

$b^{[1]}$

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

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

$W^{[2]}$

$b^{[2]}$

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

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

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

$dw^{[1]}$

$db^{[1]}$

$dz^{[2]}$

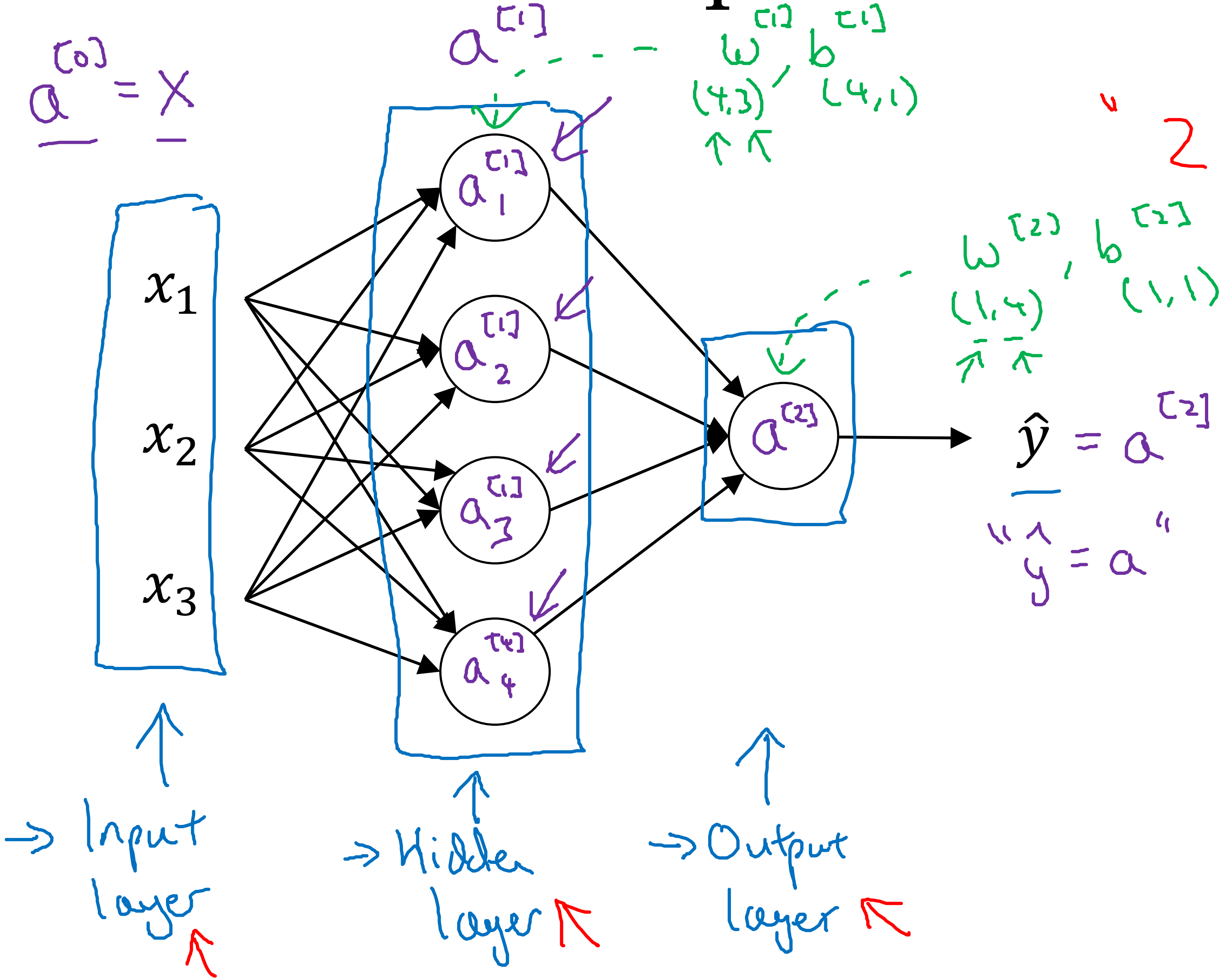
$da^{[2]}$

deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



" 2 layer NN "

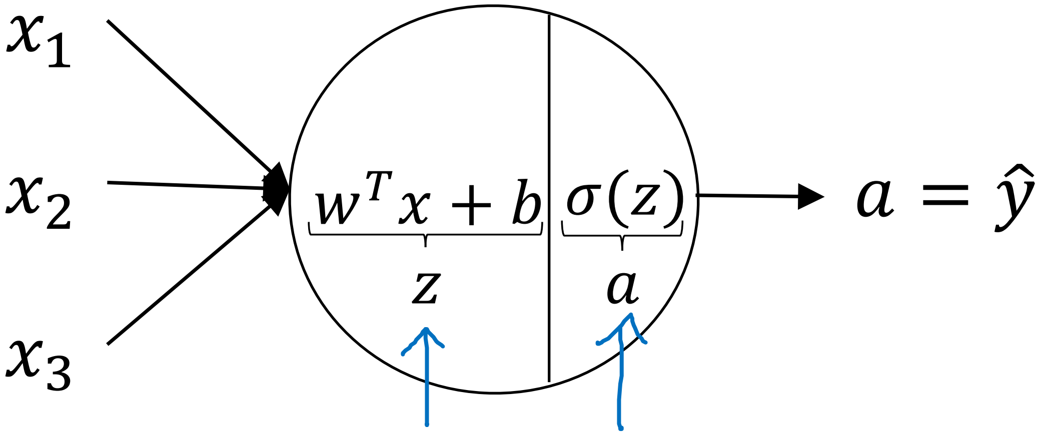
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

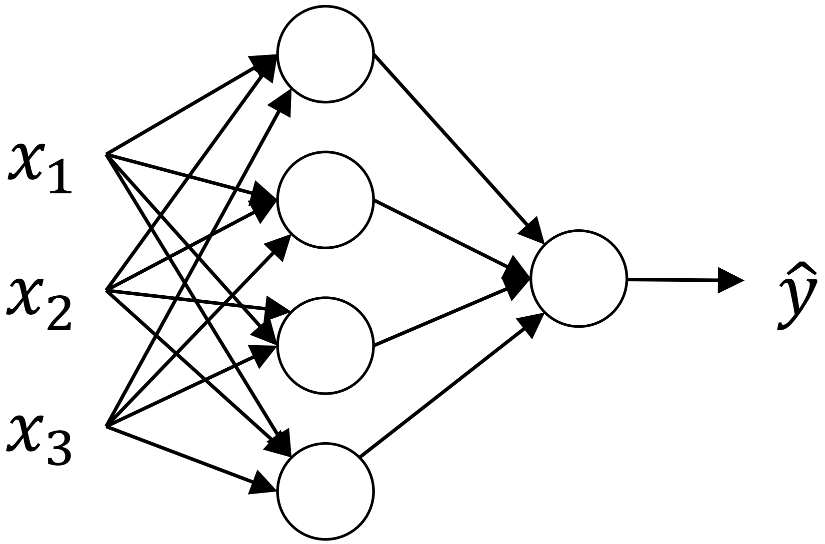
deeplearning.ai

One hidden layer Neural Network

Computing a Neural Network's Output

Neural Network Representation



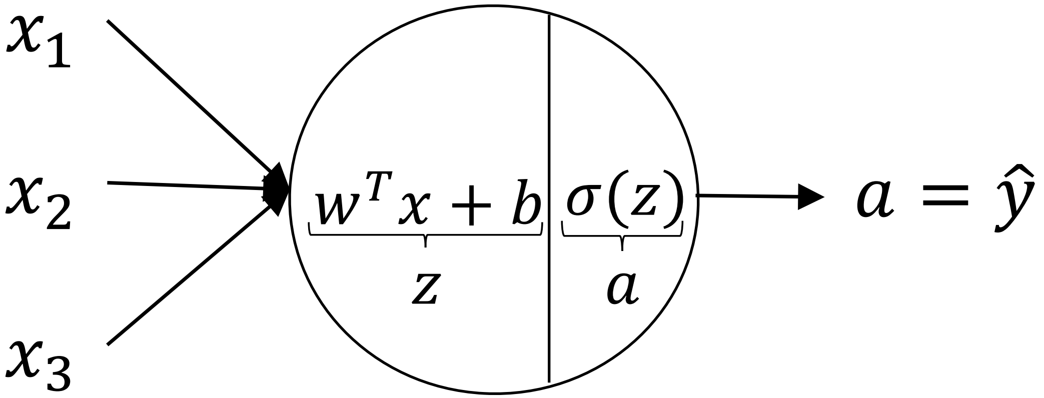


$$z = w^T x + b$$

$$a = \sigma(z)$$

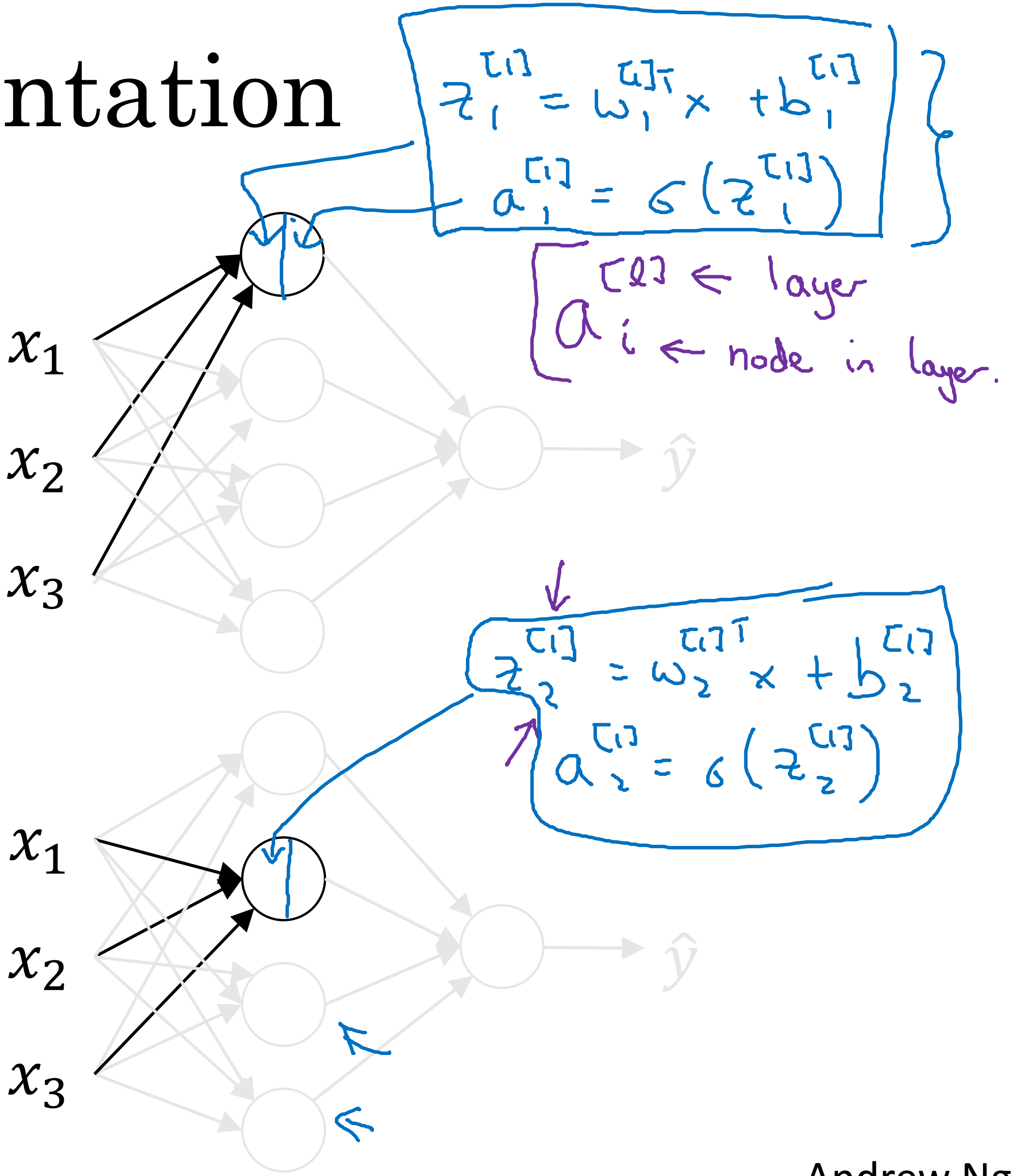
Andrew Ng

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

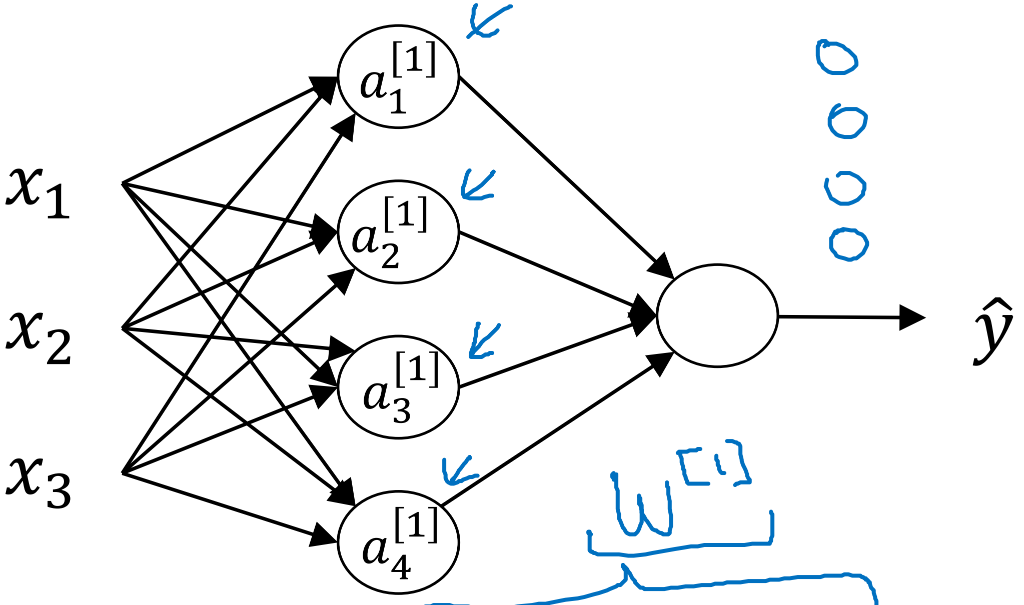
← node in layer.

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

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

Andrew Ng

Neural Network Representation



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

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

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

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

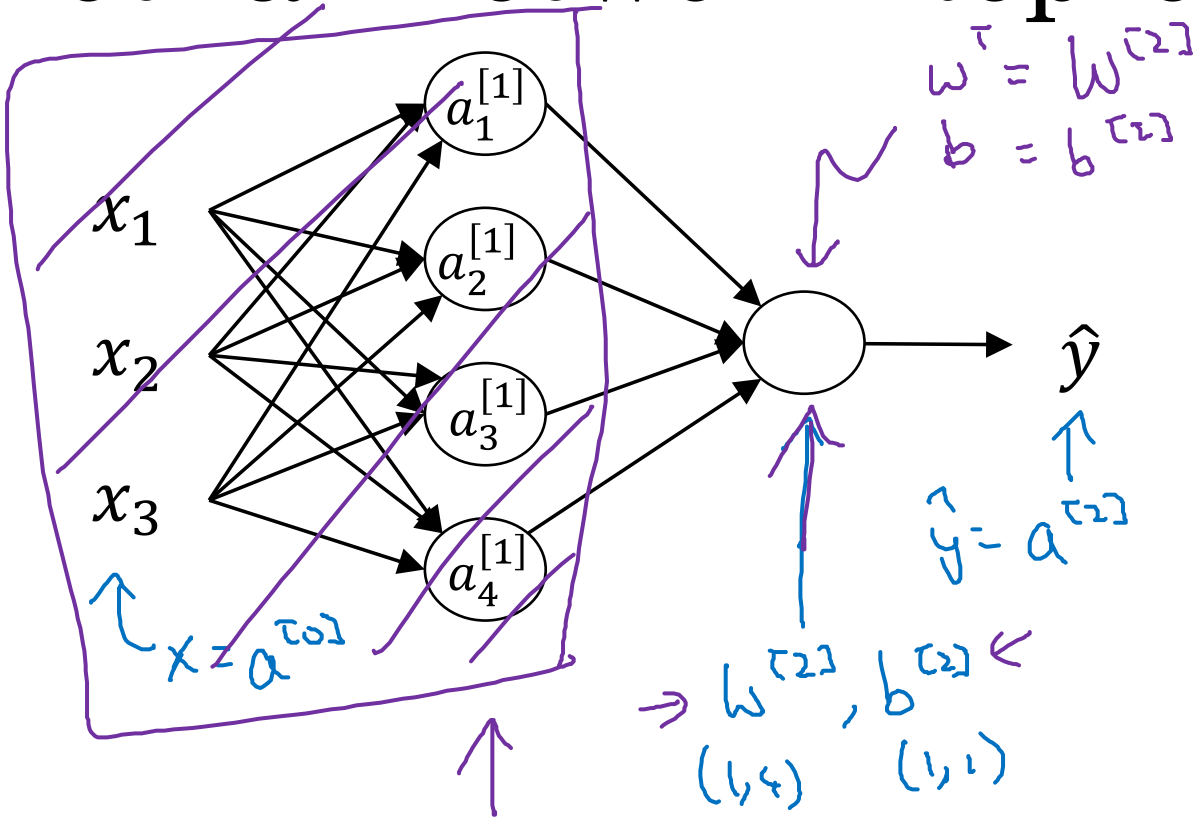
$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

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

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

Andrew Ng

Neural Network Representation learning



$z = w^T x + b$

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

Given input x:

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

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

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

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

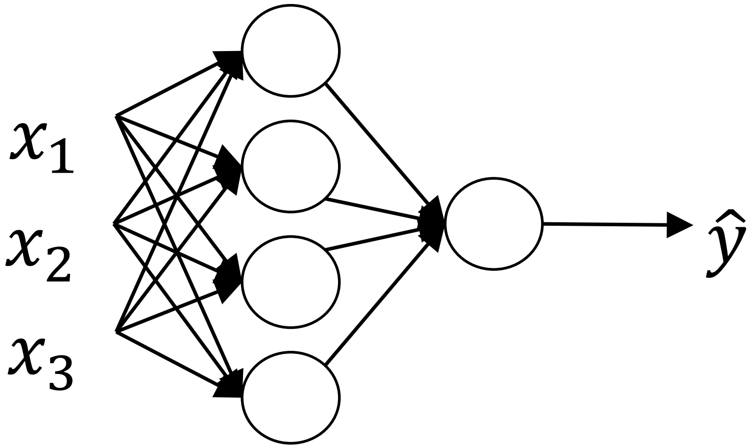
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

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

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

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

$$\vdots$$

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

$a^{[2](i)}$ ← 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

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

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

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

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

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

↔ training examples

↕ hidden units

hidden units

deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

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

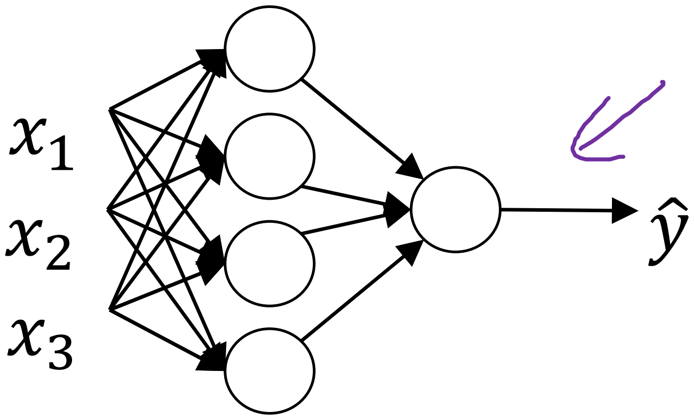
$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix}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

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

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

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

Recap of vectorizing across multiple examples



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

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

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

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

$$Z^{[1]} = W^{[1]}\overset{A^{[0]}}{\textcircled{X}} + b^{[1]} \leftarrow W^{[1]}A^{[0]} + b^{[1]}$$

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

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

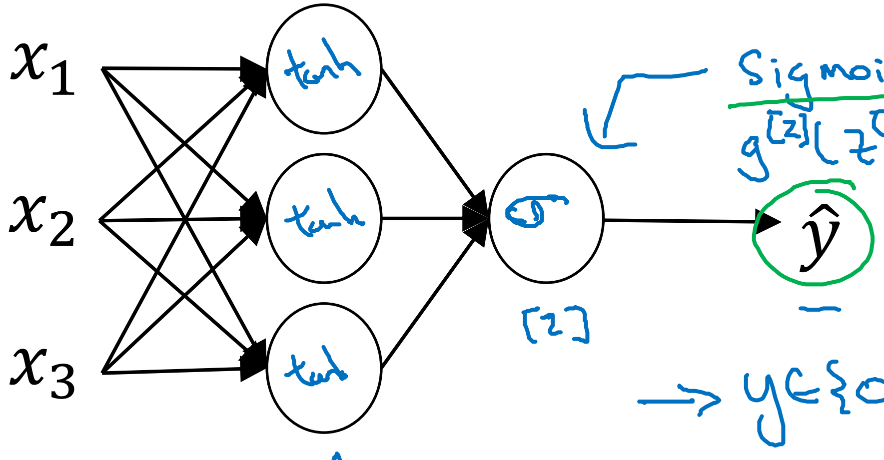
$$A^{[2]} = \sigma(Z^{[2]})$$

One hidden layer Neural Network

Activation functions

deeplearning.ai

Activation functions



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

Sigmoid

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

→ $y \in \{0, 1\}$

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

$-1 \qquad 1$

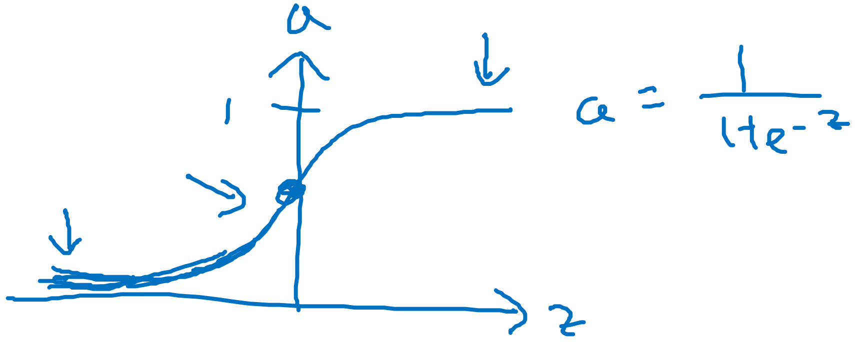
Given x:

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

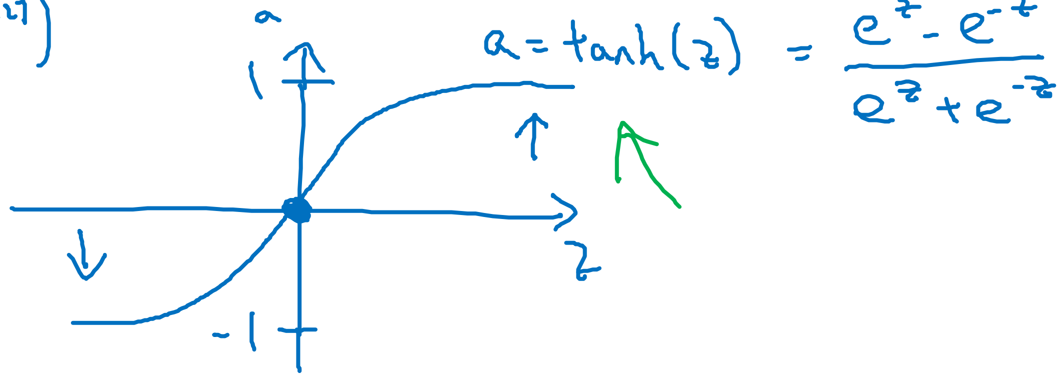
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

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

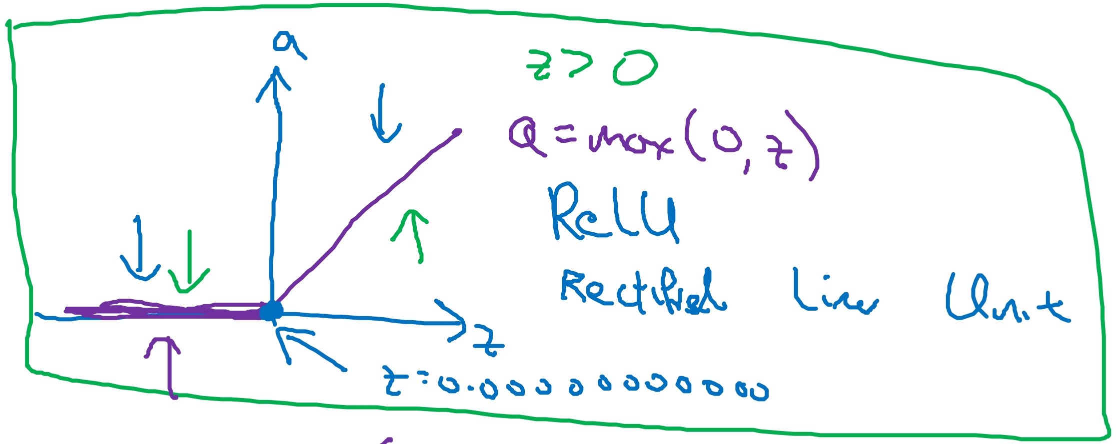
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$



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



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



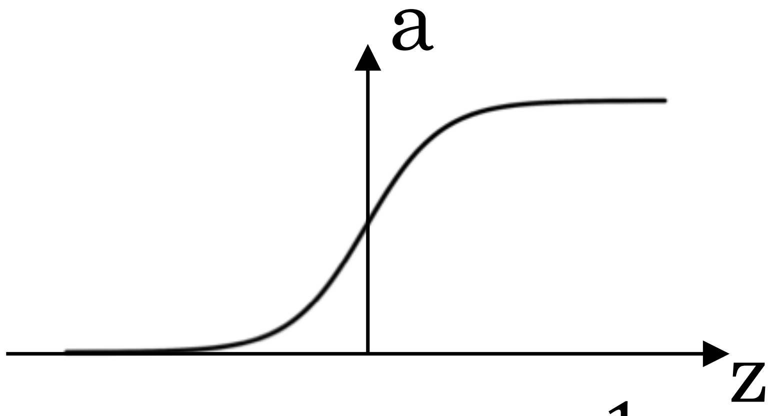
$a = \max(0, z)$

ReLU

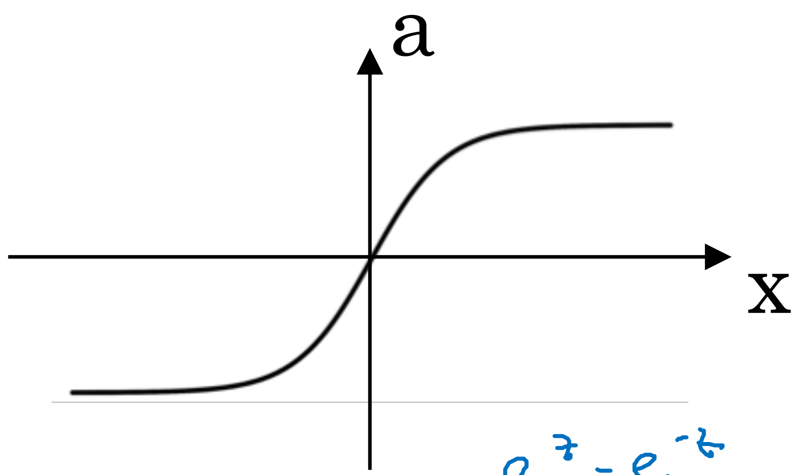
Rectified Linear Unit

Leaky ReLU

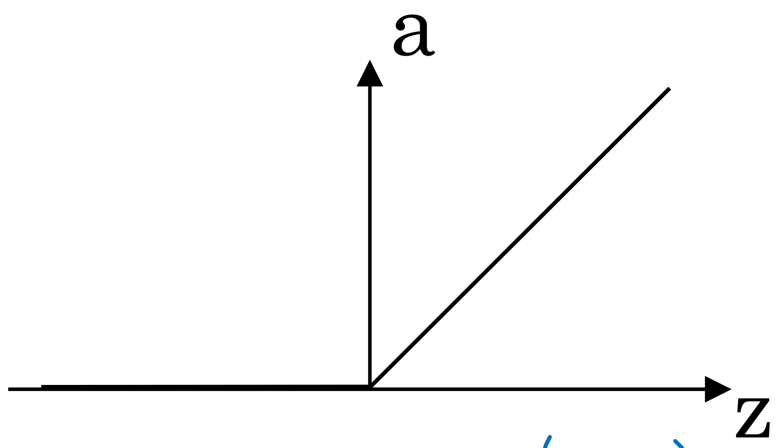
Pros and cons of activation functions



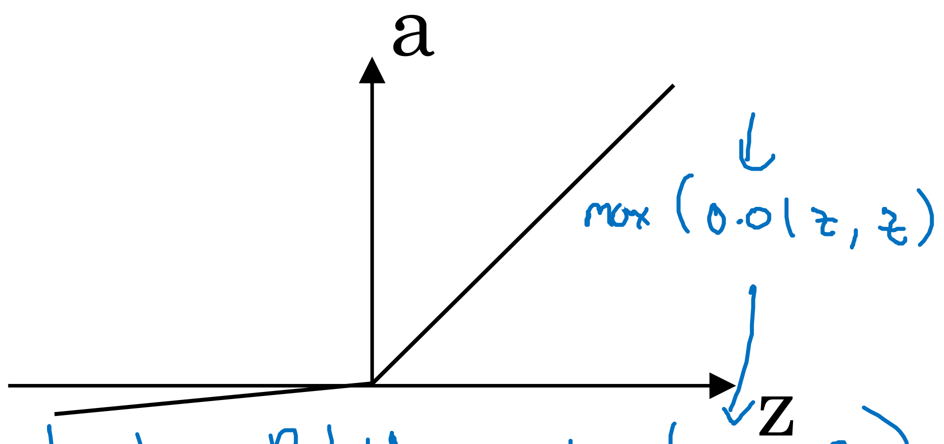
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



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



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



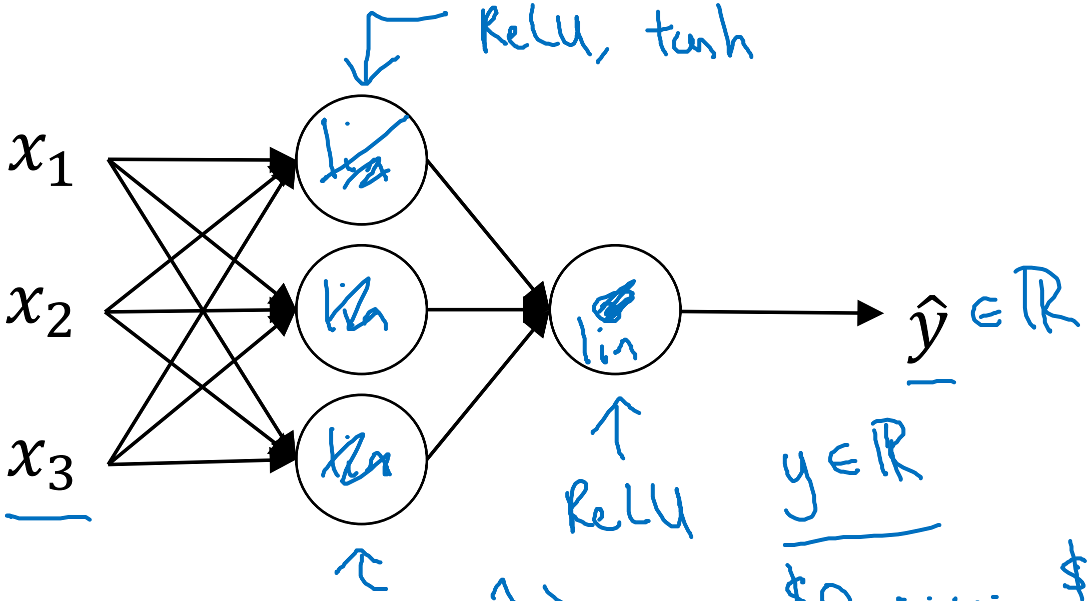
Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



$y \in \mathbb{R}$

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

$\hat{y} \geq 0.$

Given x:

→ $z^{[1]} = W^{[1]}x + b^{[1]}$

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

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

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

$g(z) = z$

"linear activation function"

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

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

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

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

$$= W'x + b'$$

$g(z) = z$

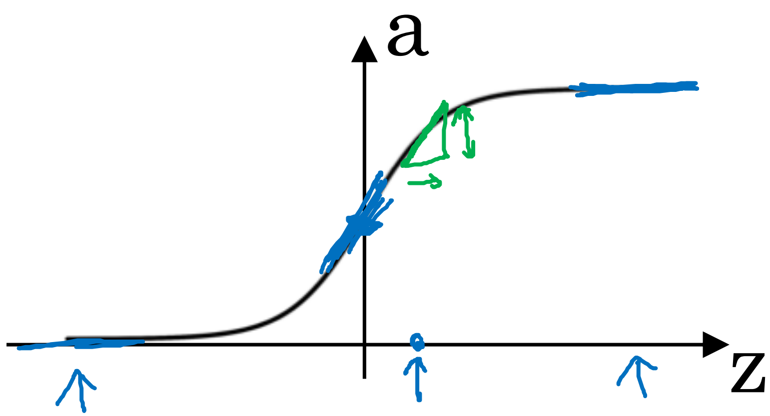
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function



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

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

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

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

$= \underbrace{g(z)\left(1 - g(z)\right)}$ ←

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

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

↑

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

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

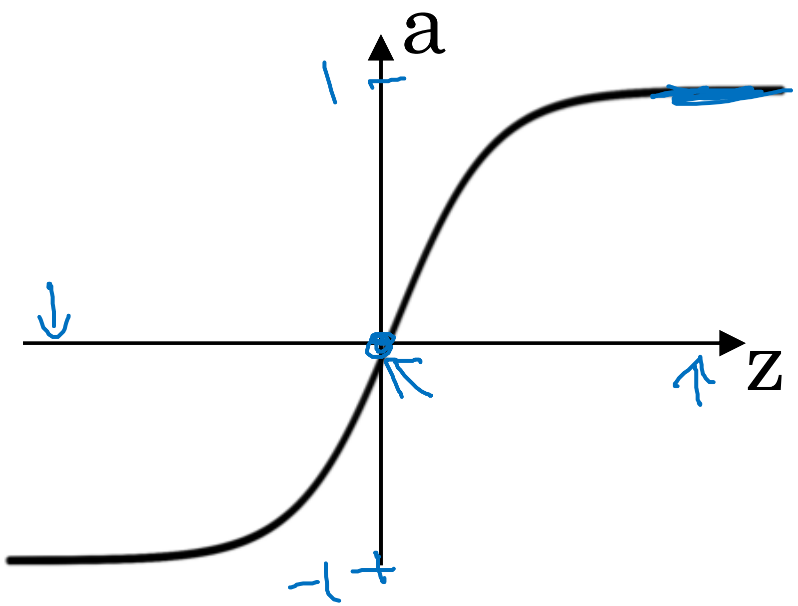
$z = -10$ $g(z) \approx 0$

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

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

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

Tanh activation function



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

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

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

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

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

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

$g'(z) \approx 0$

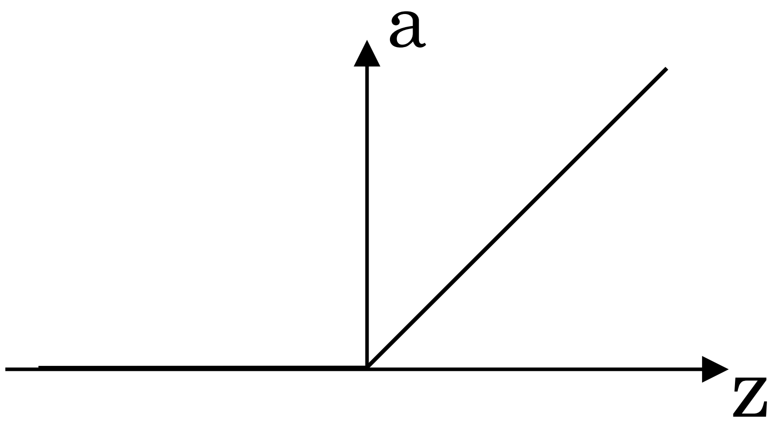
$z = -10 \quad \tanh(z) \approx -1$

$g'(z) \approx 0$

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

$g'(z) = 1$

ReLU and Leaky ReLU

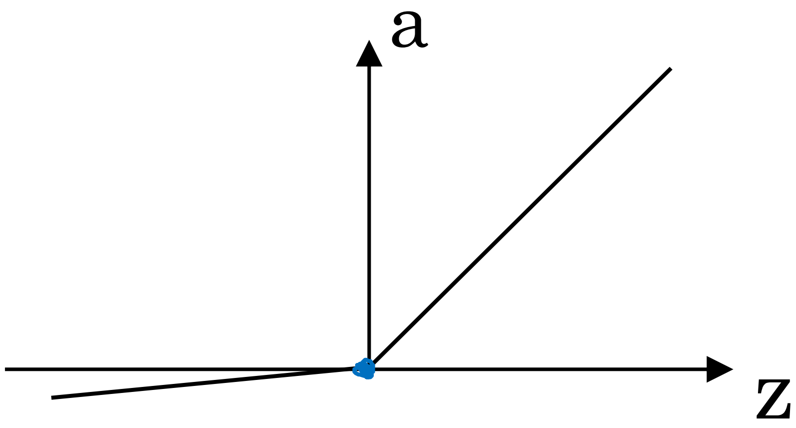


ReLU

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

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

$$z = 0.00000\ldots\ldots0$$



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

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

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

Gradient descent:

→ Repeat {

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

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

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

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

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

}

Formulas for computing derivatives

Forward propagation:

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

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

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

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

Back propagation:

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

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

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

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

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

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

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

↑ element-wise product

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

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

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

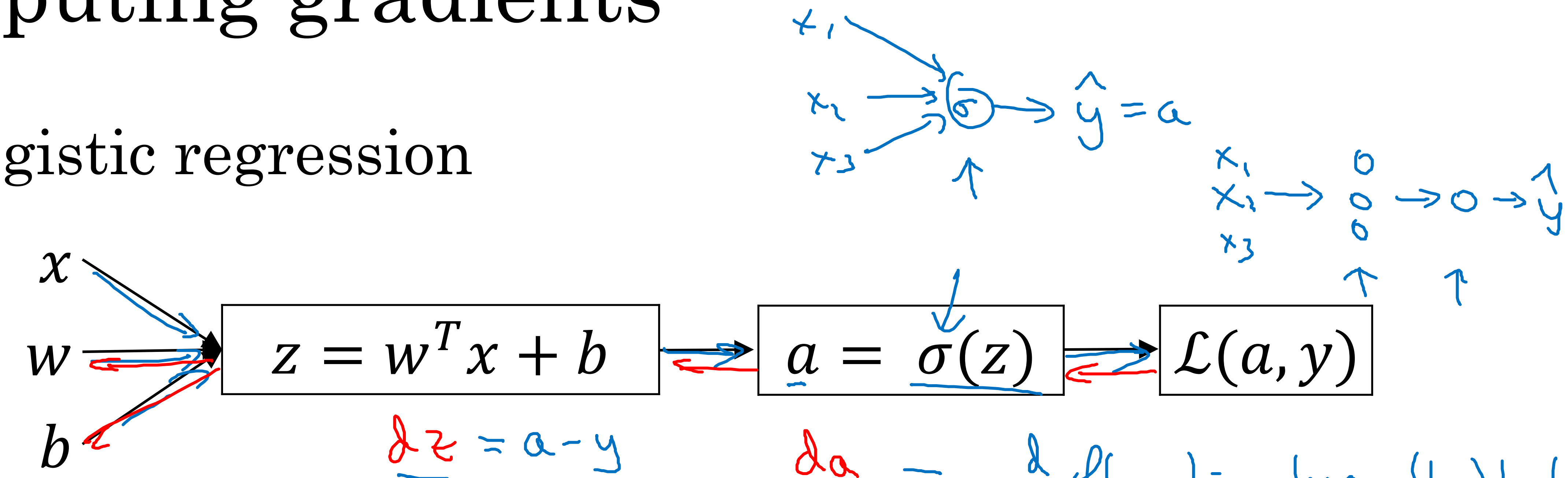
deeplearning.ai

One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

Logistic regression



x

w

b

$z = w^T x + b$

$a = \sigma(z)$

$\mathcal{L}(a, y)$

$dz = a - y$

$dw = dz \cdot x$

$db = dz$

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

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

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

"dz" = "da"

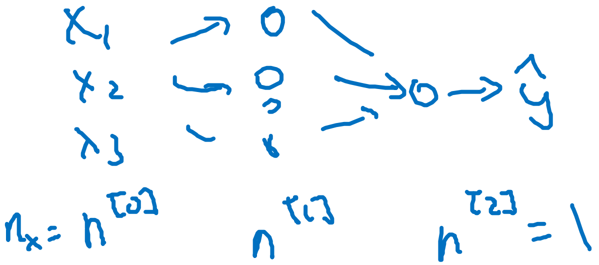
$\frac{d}{dz} g(z) = \underline{g'(z)}$

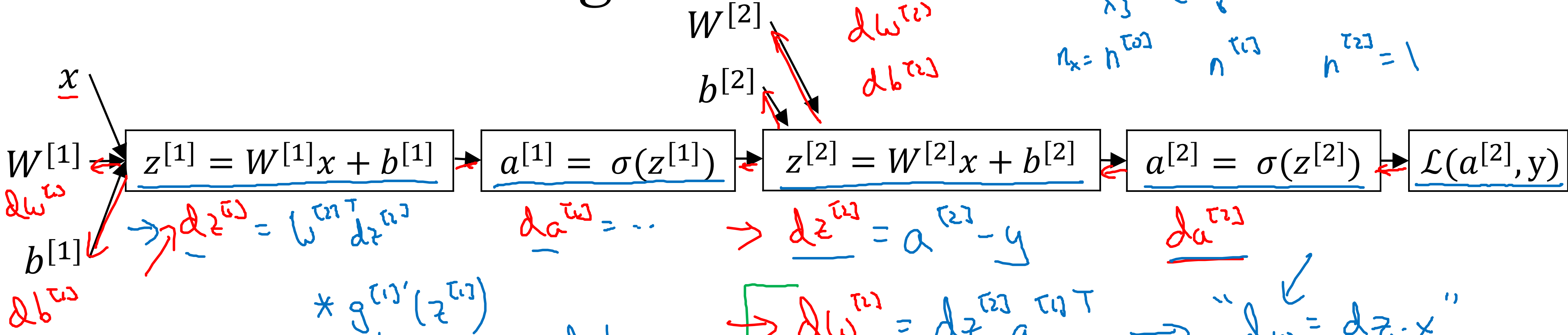
$da = \frac{d}{da} \mathcal{L}(a,y) = \underline{-y \log a - (1-y) \log(1-a)}$

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

Andrew Ng

Neural network gradients





$dz^{[1]} = w^{[2]T} dz^{[2]}$

$* g^{[1]'}(z^{[1]})$

element wise product

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

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

$da^{[2]}$

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

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

"$dw = dz \cdot x$"

$w^{[1]}$: [———]

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

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

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

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

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

foo

dfoo

$W^{[1]}$

$dW^{[2]}$

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

$a^{[0]T}$

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

Andrew Ng

Summary of gradient descent

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

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

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

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

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

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

Vectorized Implementation:

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

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

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

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

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

Summary of gradient descent

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

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

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

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

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

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

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

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

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

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

elementwise product

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

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

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

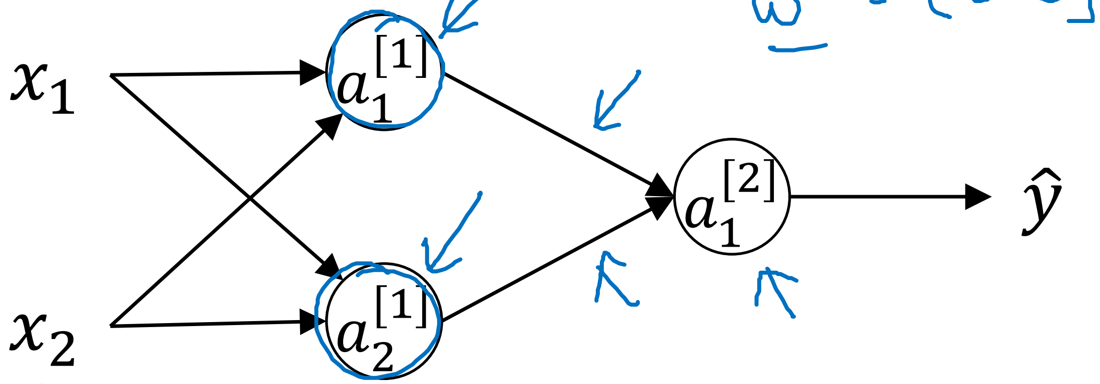
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?



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

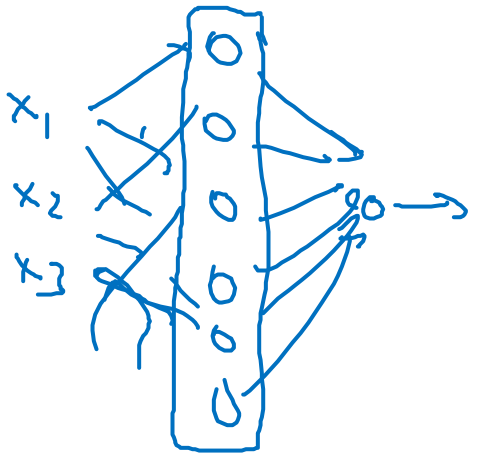
$n^{[0]} = 2$ $\quad$ $n^{[1]} = 2$

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

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

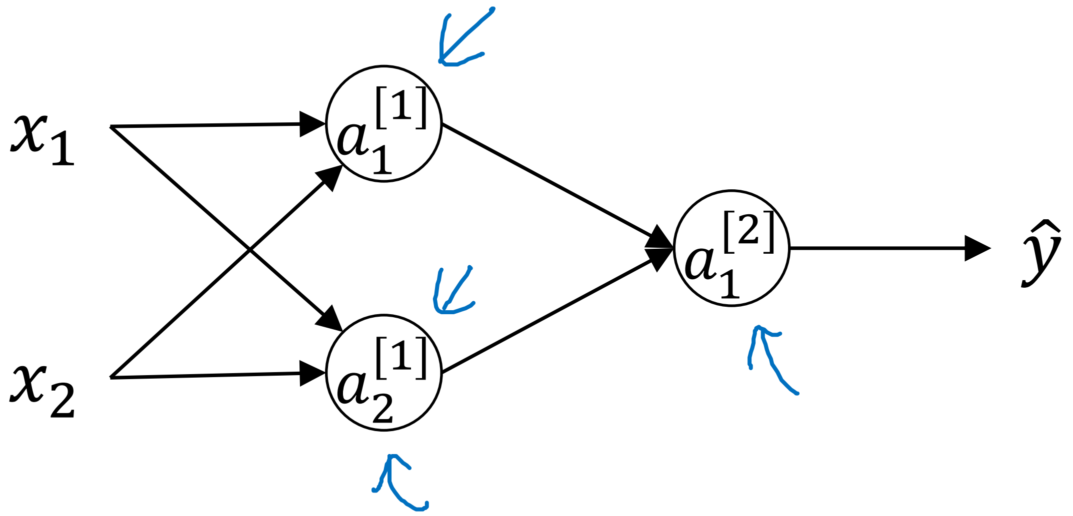
$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\quad$ $W^{[1]} = W^{[1]} - \alpha dW$

Symmetric



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

Random initialization



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

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

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

$b^{[2]} = 0$

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

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