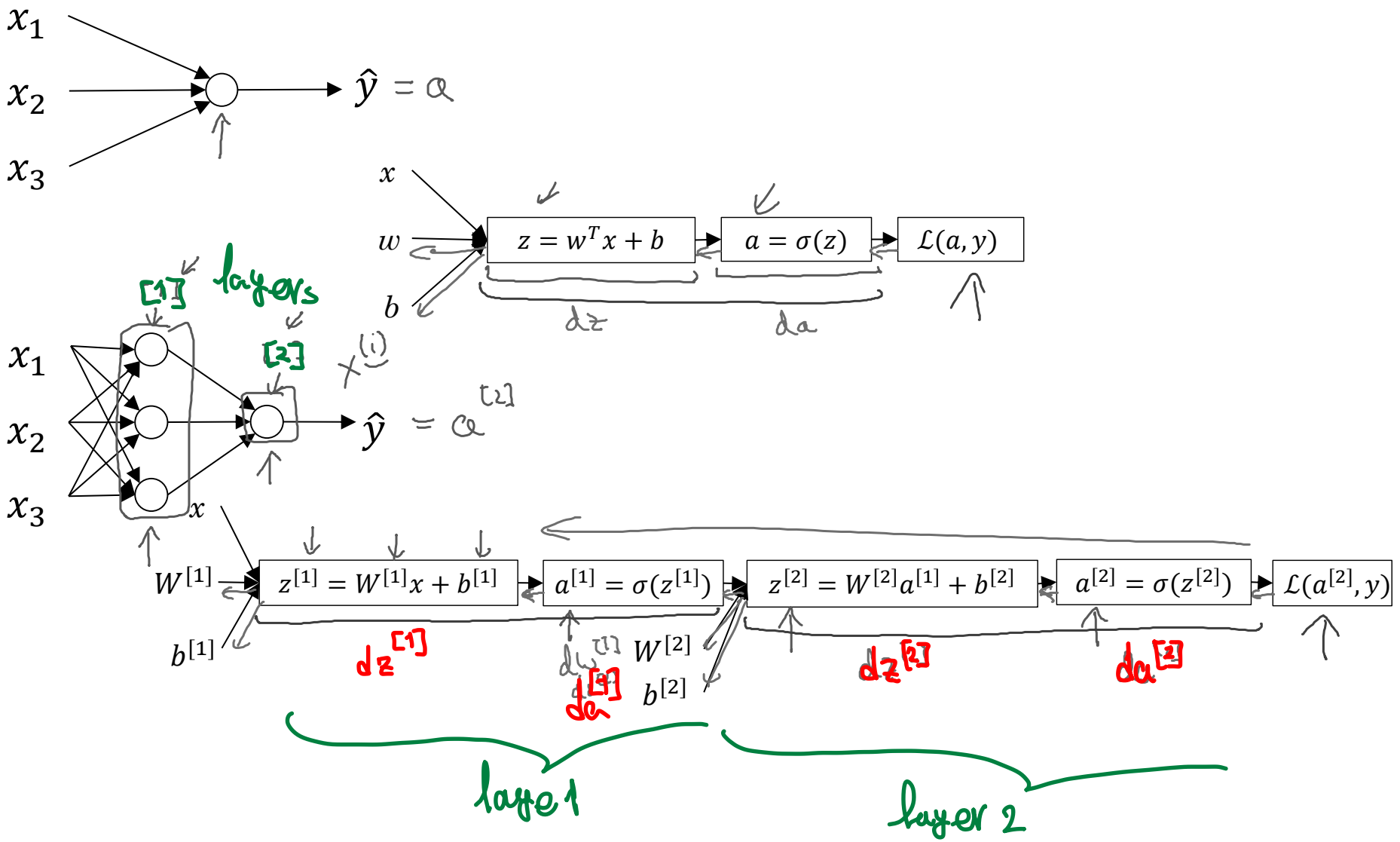
deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?

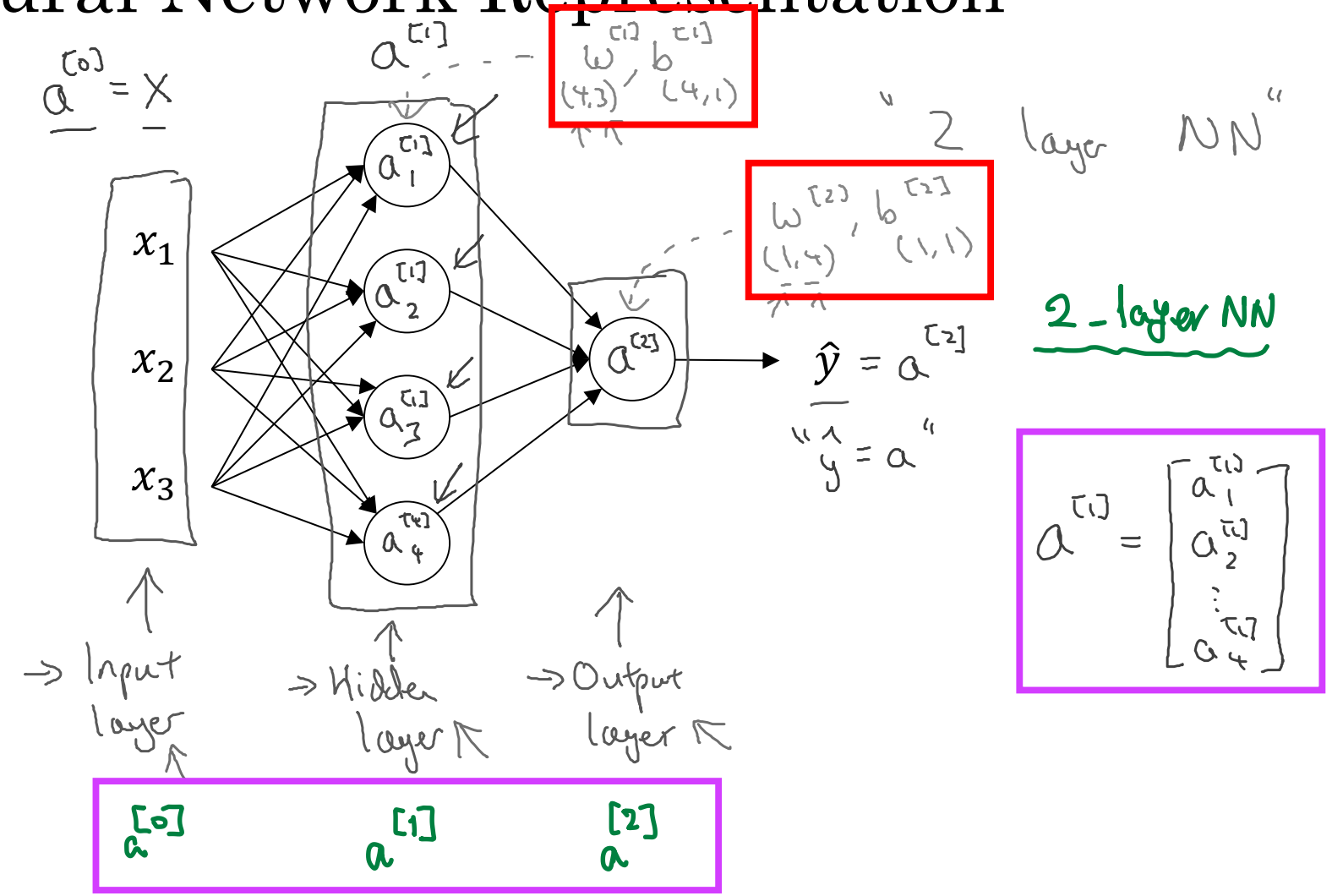


deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation

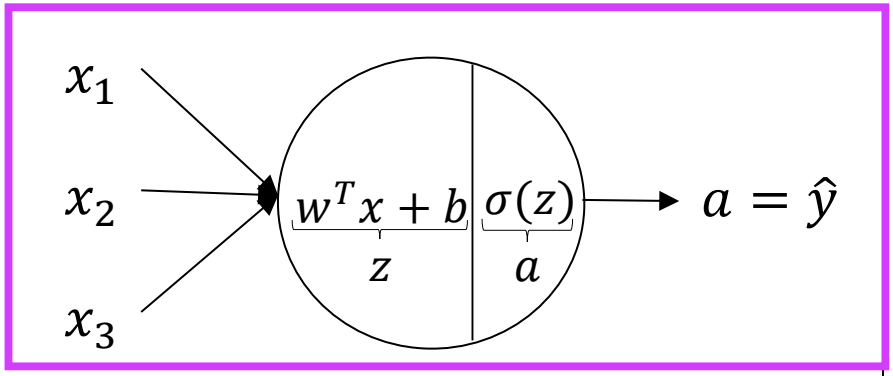


deeplearning.ai

One hidden layer Neural Network

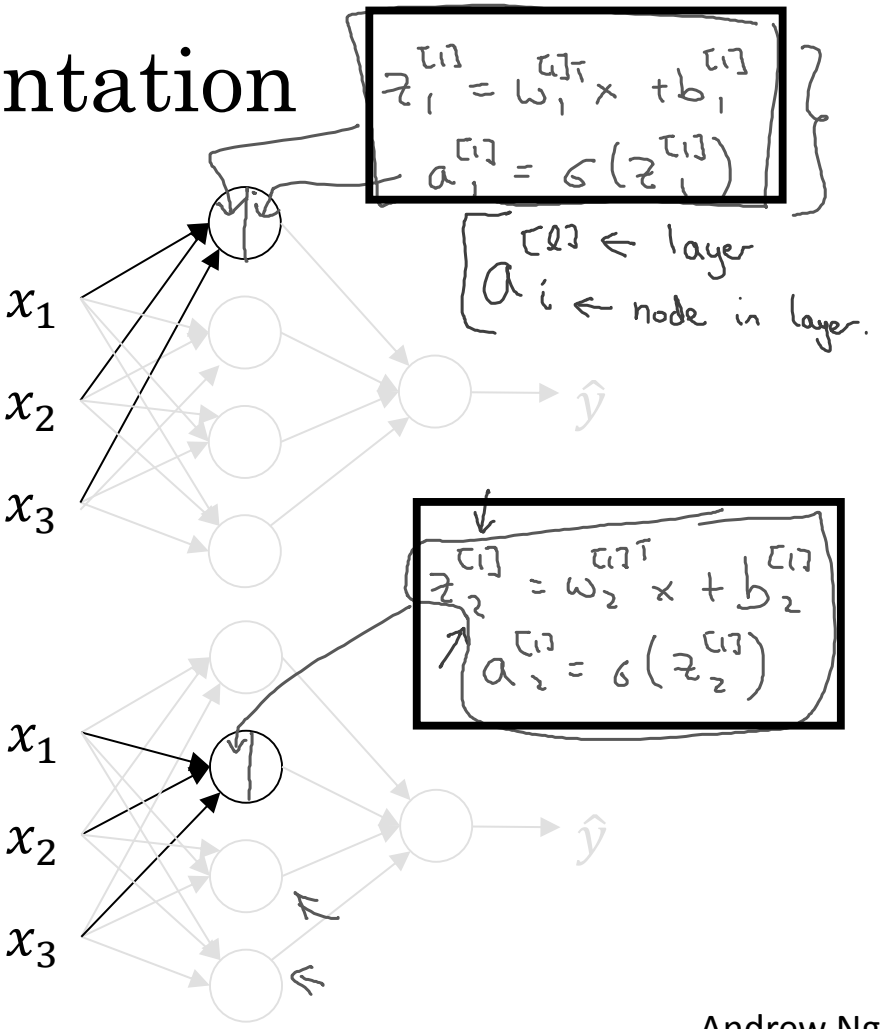
Computing a Neural Network's Output

Neural Network Representation



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

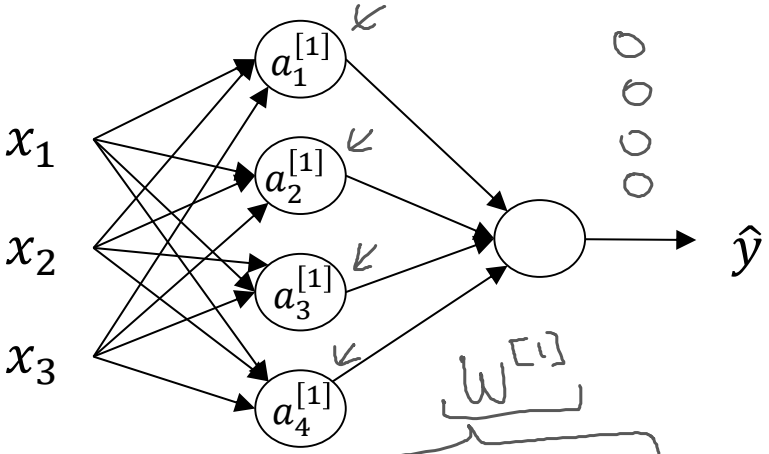
$$a = \sigma(z)$$



Andrew Ng

Neural Network Representation

$$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$



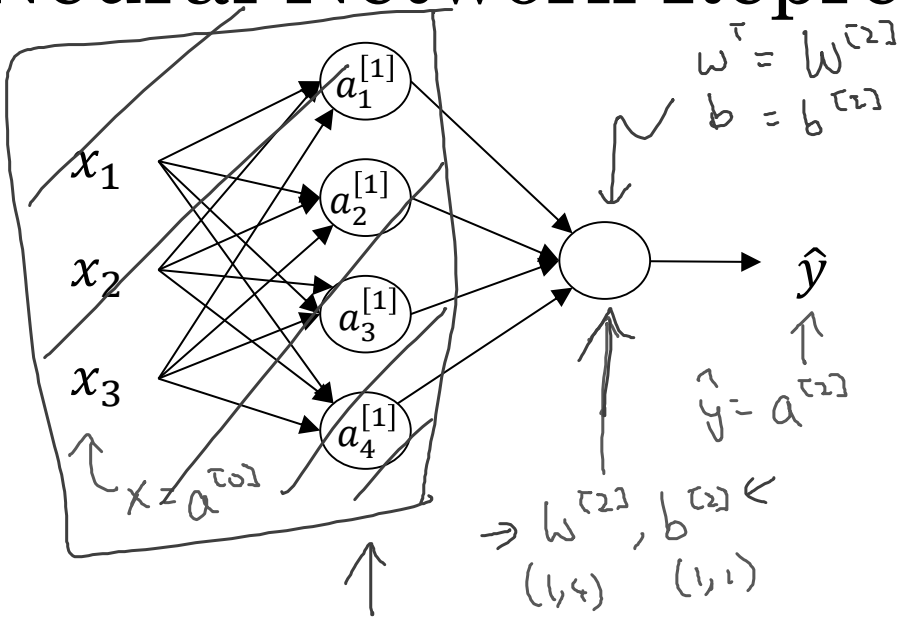
$$\begin{aligned}
z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]})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

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

$W^{[1]}$ (4, 3) $\quad b^{[1]}$ (4, 1)

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

Neural Network Representation learning



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)

4 lines of code

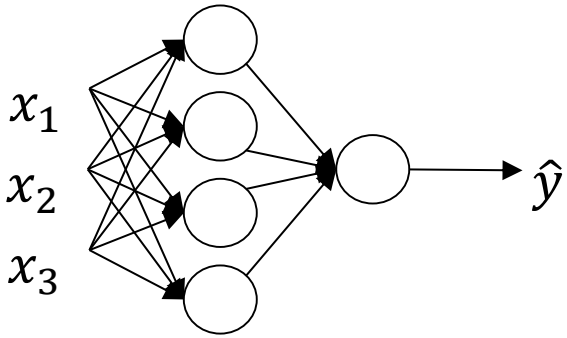
(one sample)

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

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

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

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

$\vdots$

$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[2](i)}$ ← example i, ← layer 2

for $i = 1$ to m,

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

m samples, 2 layers

درباره می خواهیم از شر loop ها بشویم ⇐ X، A، ... نیاز است

Vectorizing across multiple examples

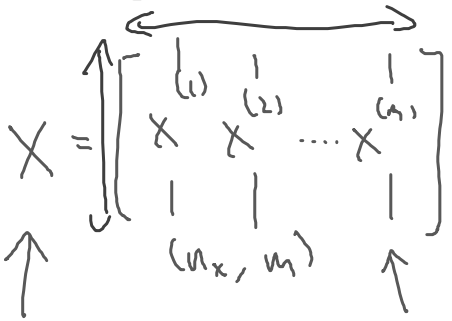
```
for i = 1 to m:
```

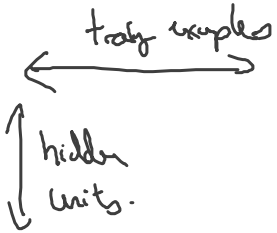
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

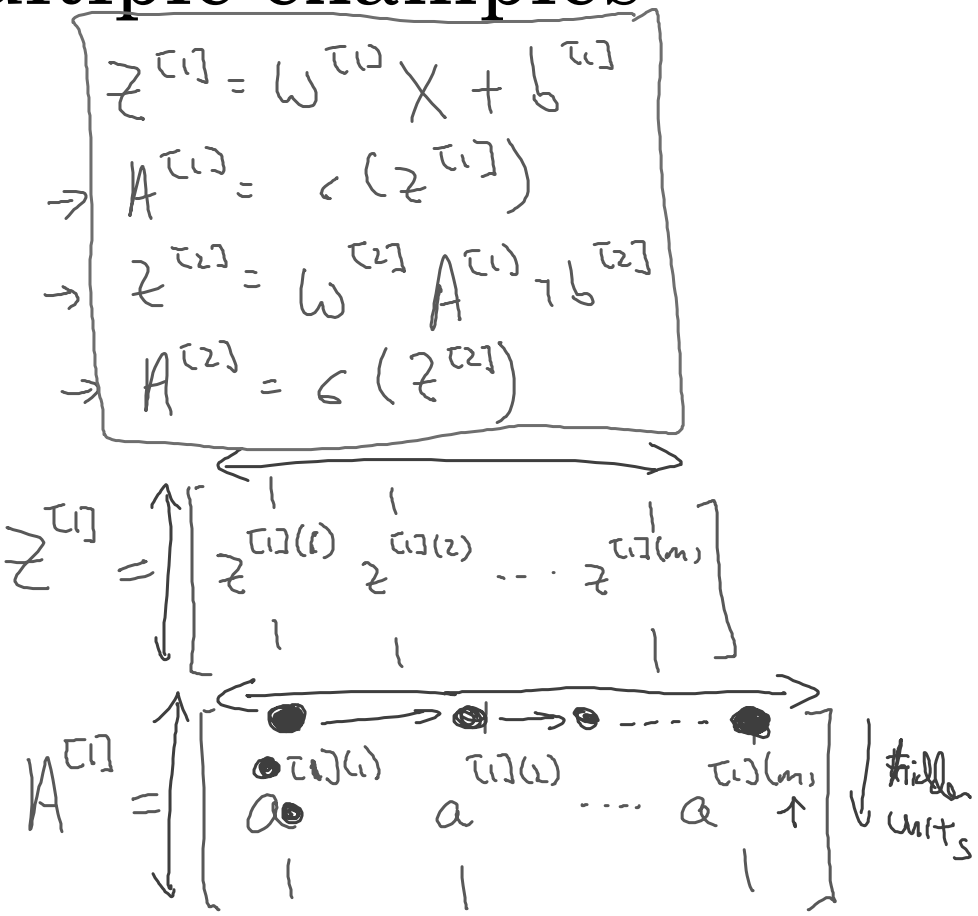
$$a^{[1](i)} = \sigma(z^{[1](i)})$$

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

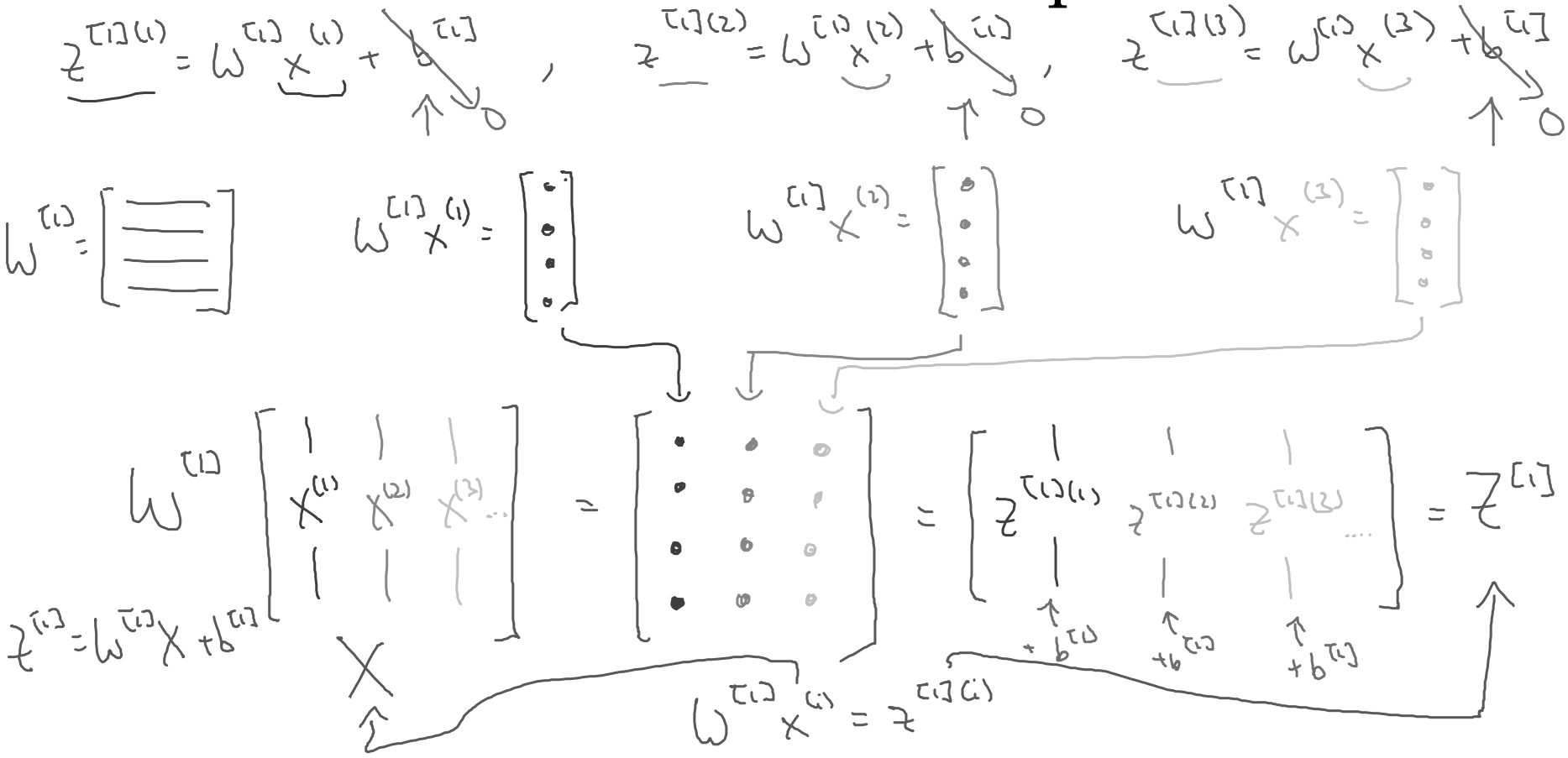
$$a^{[2](i)} = \sigma(z^{[2]}{}^{(i)})$$







Justification for vectorized implementation

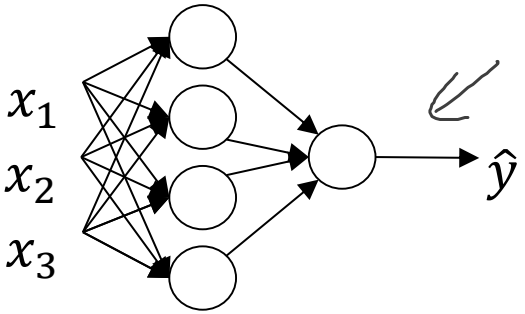


Samples

Matrix ↕ hidden units

$a^{[i](j)}_{k}$ = node k, layer i, sample j

Recap of vectorizing across multiple examples



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

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

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

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

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

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

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

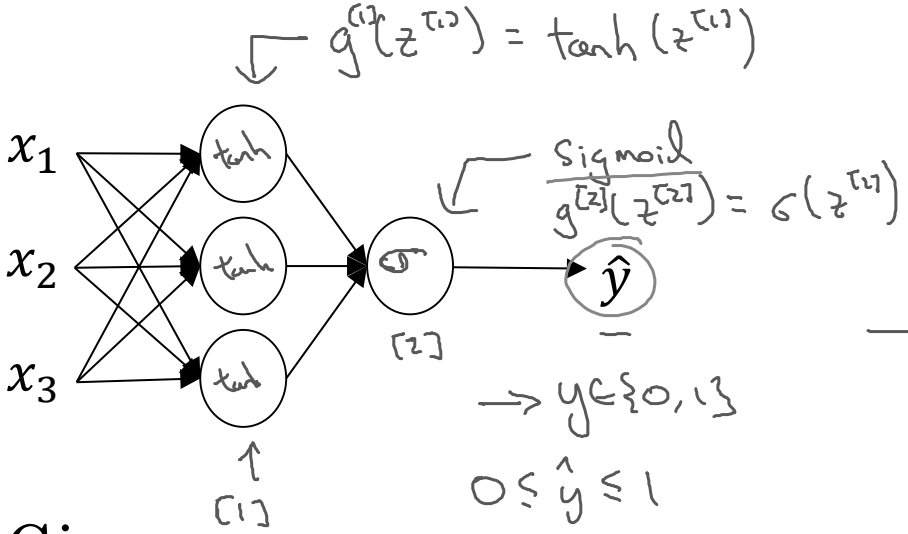
Fully vectorized version for m samples in 2 layer NN.

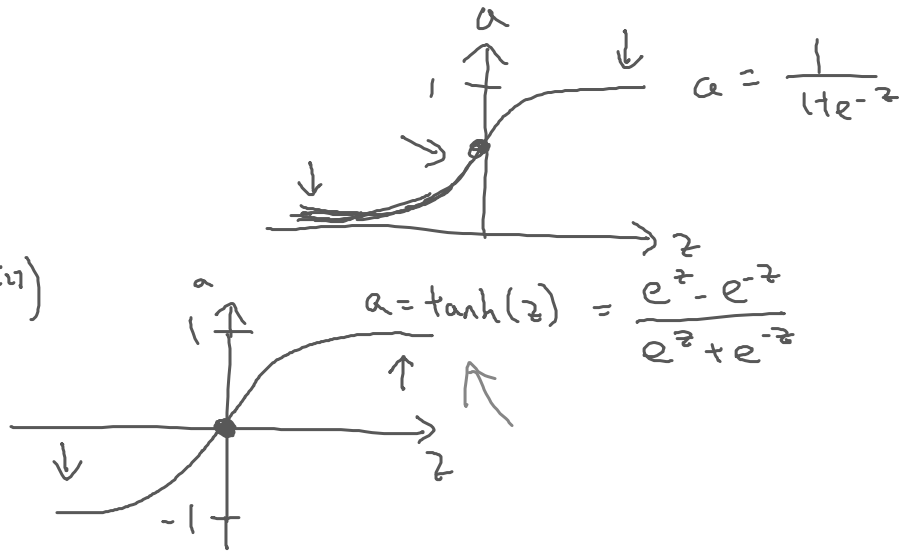
deeplearning.ai

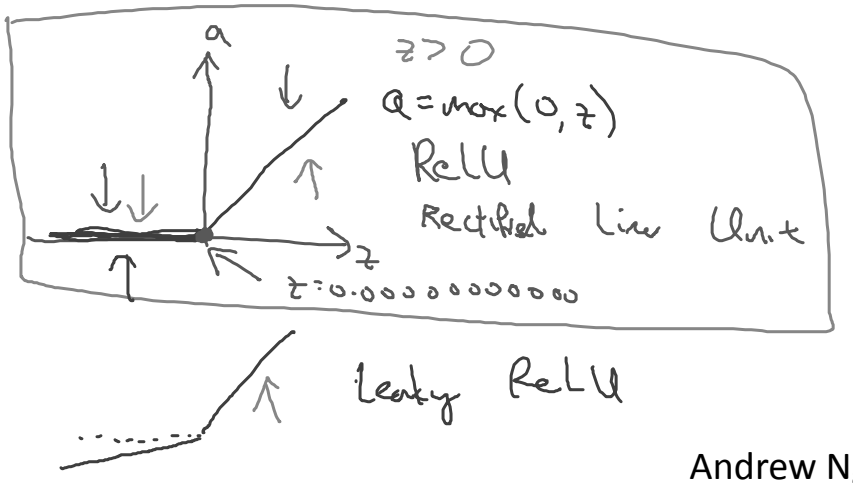
One hidden layer Neural Network

Activation functions

Activation functions







Given x:

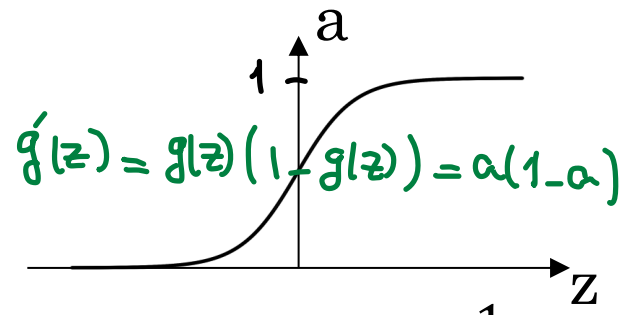
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]}) \quad g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]}) \quad g^{[2]}(z^{[2]})$$

Andrew Ng

Binary Classification: Sigmoid for output layer and ReLU for other units

$a = g(z)$

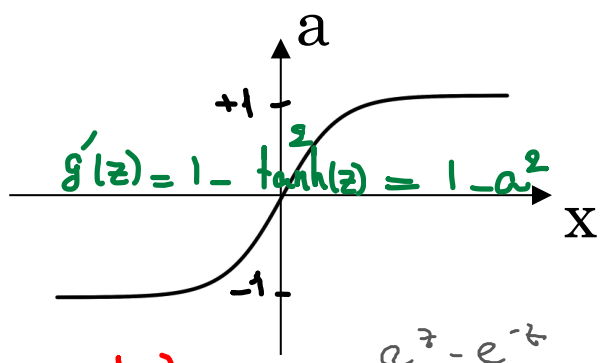
Pros and cons of activation functions



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

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

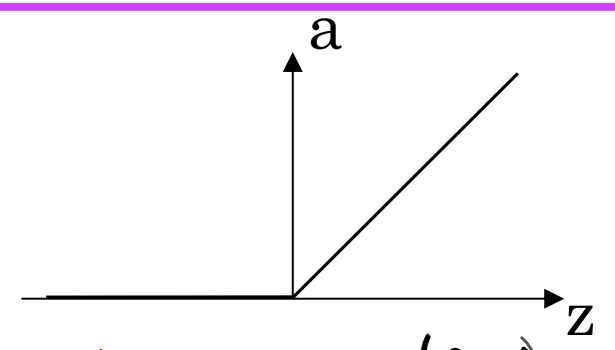
never use that



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

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

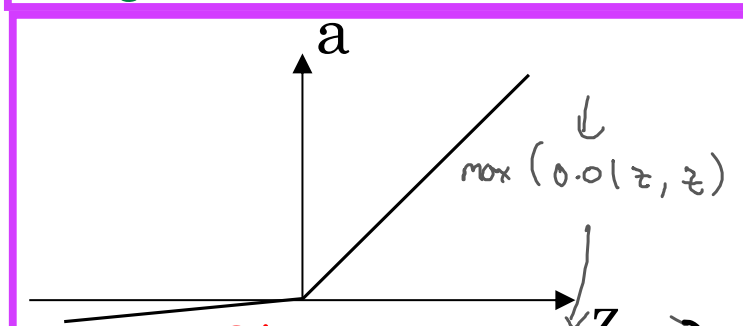
just for hidden nodes, better than Sigmoid



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

very good

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



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

better than ReLU

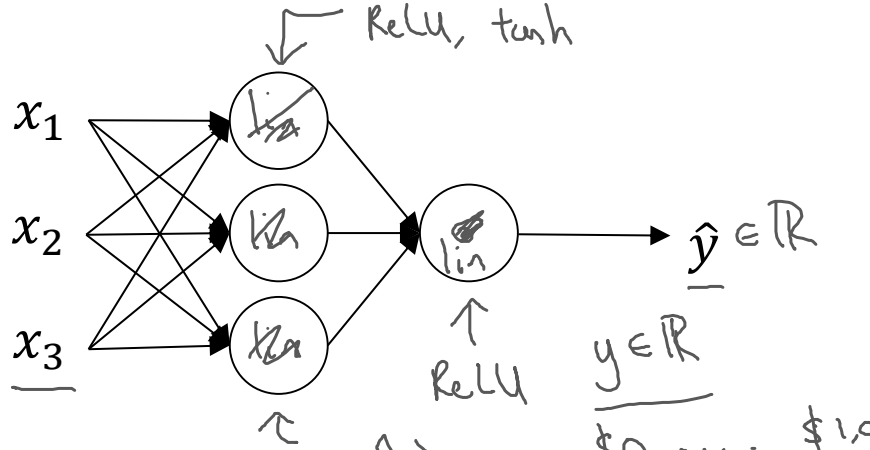
$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



$y \in \mathbb{R}$

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

Given x:

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

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

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

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

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

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

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

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

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

$$= W'x + b'$$

$g(z) = z$

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

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

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

$n_x = n^{[0]}$ (input units), $n^{[1]}$ (hidden units), $n^{[2]} = 1$ (output units)

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

Gradient descent:

→ Repeat {

one Iteration:

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

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

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

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

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

}

Formulas for computing derivatives

Forward propagation:

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

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

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

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

4 equations

Back propagation:

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

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

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

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

$*$ element-wise product

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

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

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

6 equations

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

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

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

Prevent

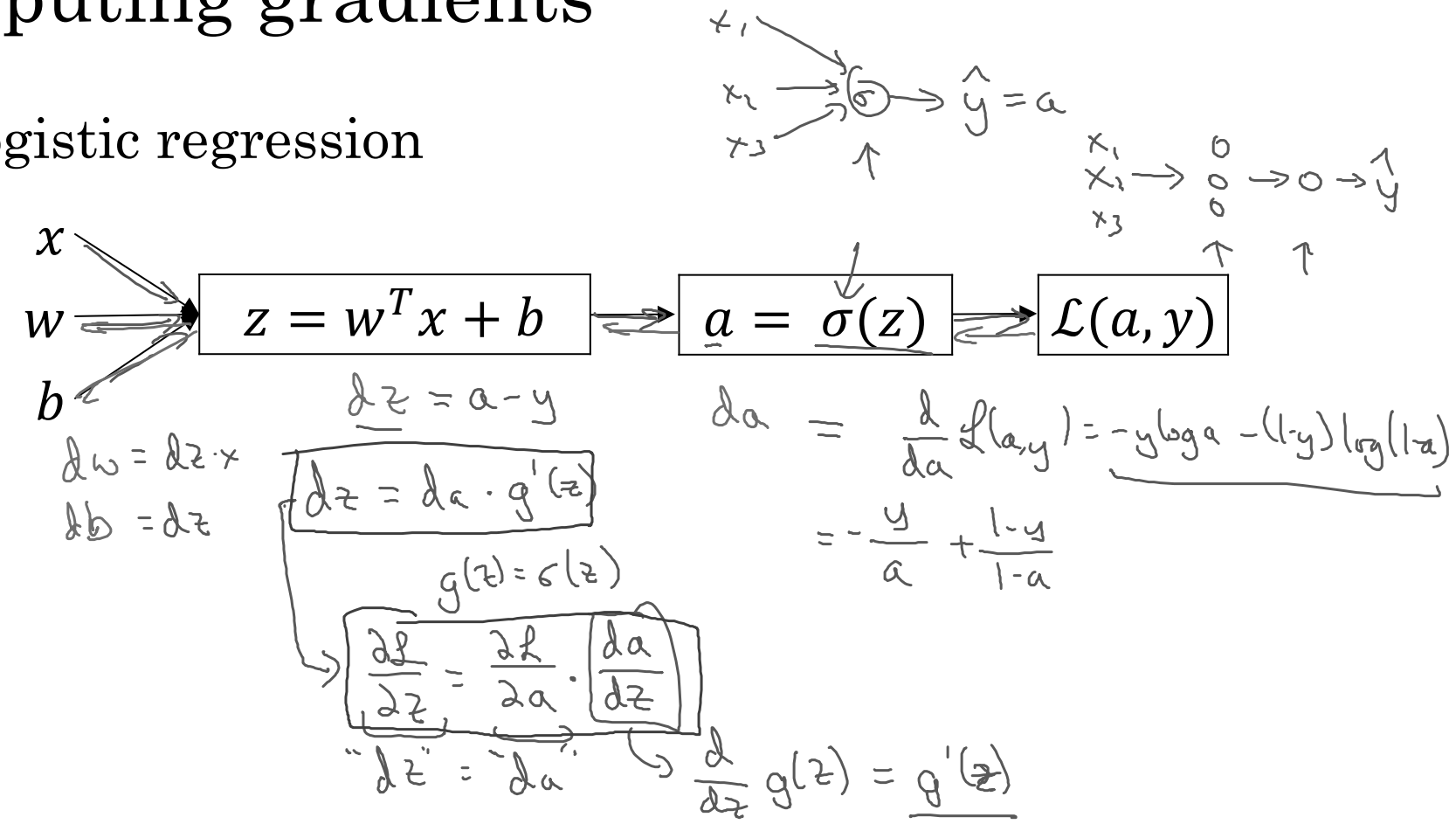
deeplearning.ai

One hidden layer Neural Network

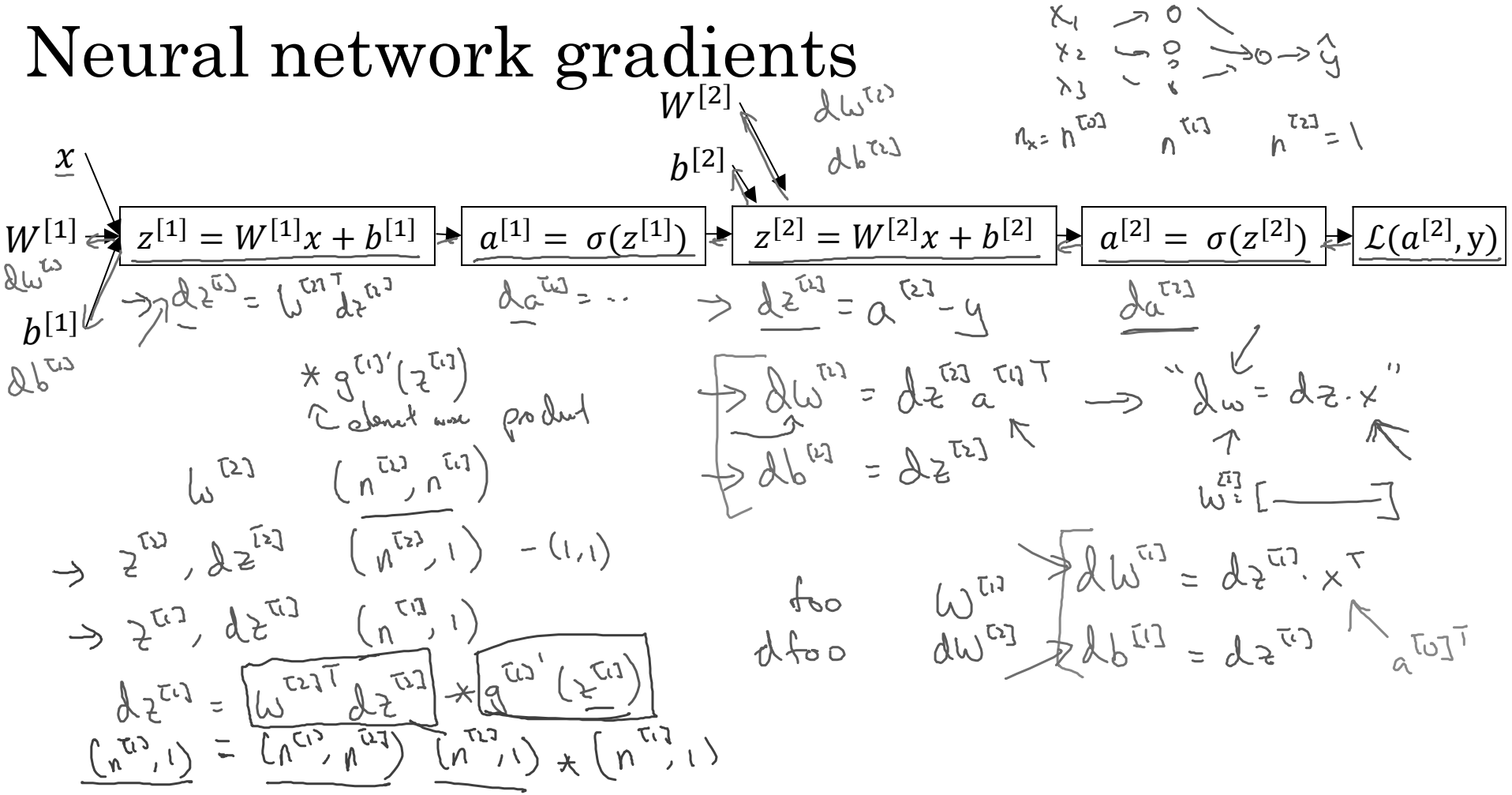
Backpropagation intuition (Optional)

Computing gradients

Logistic regression



Neural network gradients



Summary of gradient descent

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

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

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

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

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

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

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

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

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

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

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

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

$(n^{[1]}, m)$ $(n^{[1]}, m)$ $(n^{[1]}, m)$ — elementwise product

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

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

Backward Propagation Equations

deeplearning.ai

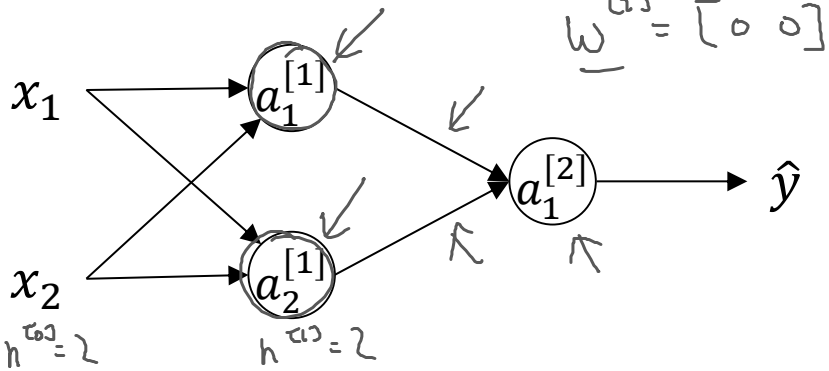
One hidden layer Neural Network

Random Initialization

Zero Initialization ⇒ Symmetry ⇒ Canceling Path ⇒ X

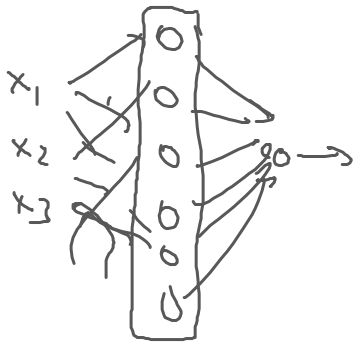
⇒ Initial Bias is required or random initialization

What happens if you initialize weights to zero?



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

Symmetric

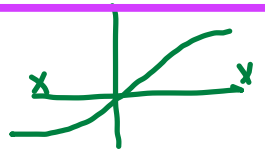


$w^{[1]} = \begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$

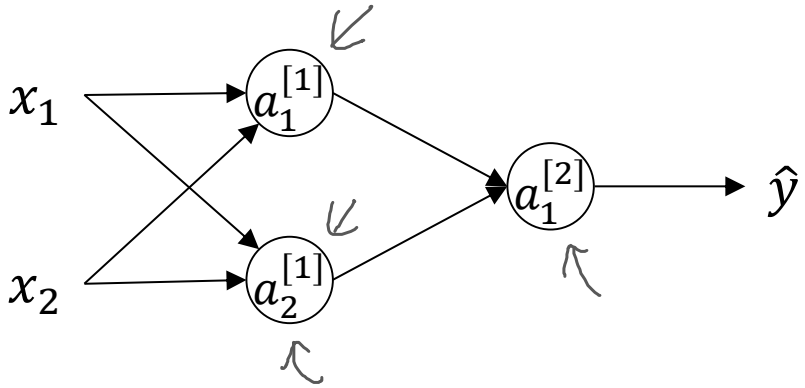
$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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

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

بهتر از مقادیر random اولیه خیلی بزرگ/کوچک نباشند چون در  saturation تولید می‌شود.

Random initialization



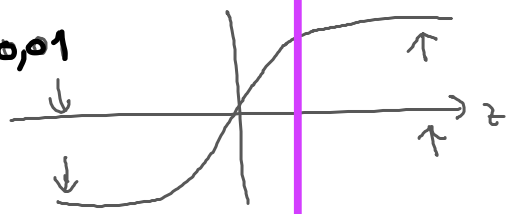
$w^{[1]} = np.random.randn((2,2)) * 0.01$

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

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

$b^{[2]} = 0$

100?



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

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

Random Initialization of Parameters