

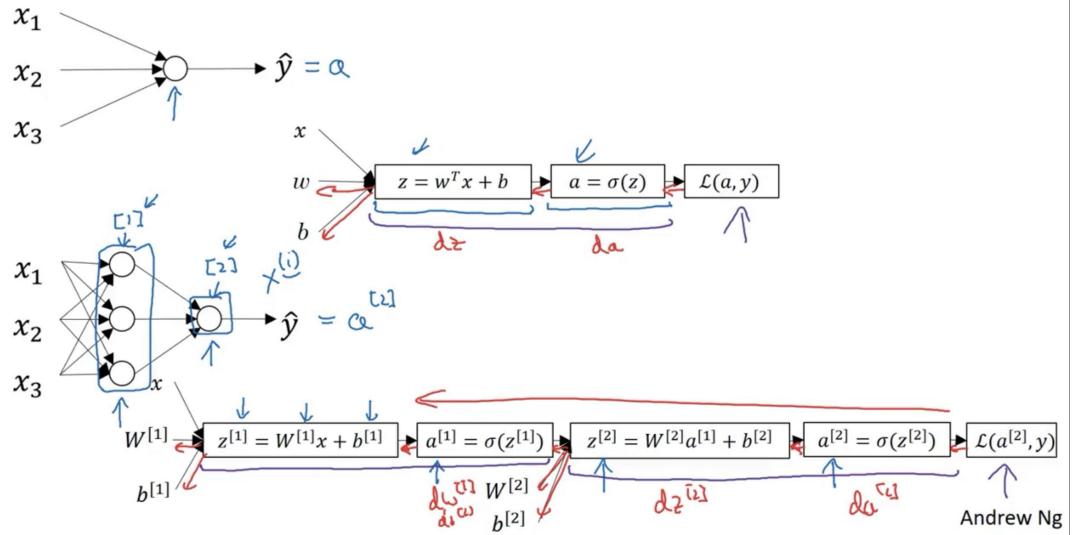
Week 3 - Shallow neural networks

Wednesday, August 5, 2020 4:28 PM

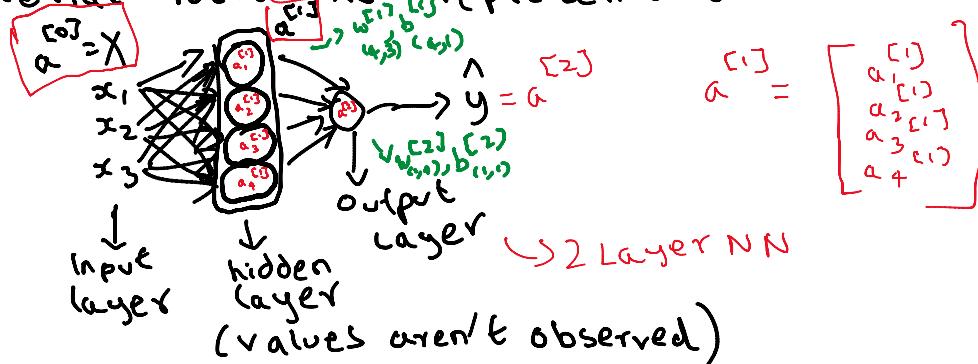
① Neural Networks Overview

① Neural Network with hidden layer

What is a Neural Network?



② Neural Networks representation



③ Computing a Neural Network's output

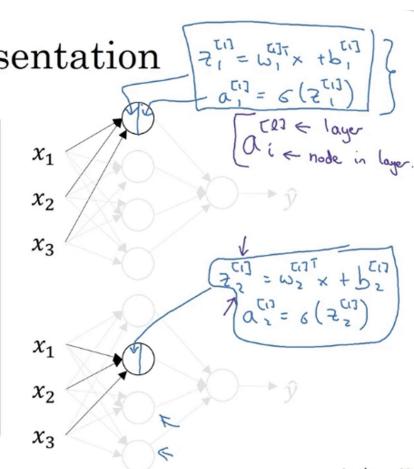
Neural Network Representation

$$x_1 \\ x_2 \\ x_3$$

$$w^T x + b$$

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

$$a = \sigma(z)$$



Andrew Ng

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

vectorize

$$z^{[1]} = \begin{bmatrix} -w_1^{[1]T} \\ -w_2^{[1]T} \\ -w_3^{[1]T} \\ -w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$w^{[1]T} \quad \quad \quad b^{[1]}$

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

- NN representation learning

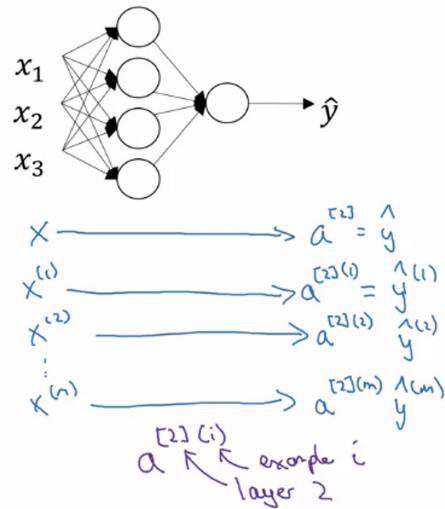
Given input x :

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

hidden layer

④ Vectorizing across multiple examples

Vectorizing across multiple examples



$$\left\{ \begin{array}{l} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{array} \right. \quad \rightarrow \boxed{\begin{array}{l} \text{for } i = 1 \text{ to } m, \\ z^{[1](i)} = w^{[1]}x^{(i)} + b^{[1]} \\ a^{[1](i)} = \sigma(z^{[1](i)}) \\ z^{[2](i)} = w^{[2]}a^{[1](i)} + b^{[2]} \\ a^{[2](i)} = \sigma(z^{[2](i)}) \end{array}}$$

Andrew Ng

Vectorizing across multiple examples

for $i = 1$ to m :

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

$$X = \begin{bmatrix} & 1 & & 1 \\ & X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ & | & | & & | \\ (n_x, m) & | & | & & | \end{bmatrix} \quad \begin{array}{l} \text{rows for samples} \\ \text{columns for hidden units} \end{array}$$

$$\begin{aligned} z^{[1]} &= w^{[1]}X + b^{[1]} \\ \Rightarrow A^{[1]} &= \sigma(z^{[1]}) \\ \Rightarrow z^{[2]} &= w^{[2]}A^{[1]} + b^{[2]} \\ \Rightarrow A^{[2]} &= \sigma(z^{[2]}) \\ z^{[1]} &= \begin{bmatrix} z^{1} & z^{[1](2)} & \dots & z^{[1](m)} \end{bmatrix} \\ A^{[1]} &= \begin{bmatrix} a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \end{bmatrix} \quad \begin{array}{l} \text{# hidden units} \\ \text{# hidden units} \end{array} \end{aligned}$$

Andrew Ng

⑤ Explanation for vectorised implementation

Justification for vectorized implementation

$$z^{(1)} = w^{(1)}x + b^{(1)}, \quad z^{(2)} = w^{(2)}x^{(1)} + b^{(2)}, \quad z^{(3)} = w^{(3)}x^{(2)} + b^{(3)}$$

$$w^{(1)} = \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}, \quad w^{(1)}x = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad w^{(1)}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad w^{(1)}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$z^{(1)} = w^{(1)}x + b^{(1)} \quad \text{and} \quad z^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} z^{(1)(1)} \\ z^{(1)(2)} \\ z^{(1)(3)} \end{bmatrix} = z^{(1)}$$

$$w^{(1)}x^{(1)} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} z^{(1)(1)} \\ z^{(1)(2)} \\ z^{(1)(3)} \end{bmatrix} = z^{(1)}$$

$$z^{(1)(1)} = w^{(1)(1)}x^{(1)(1)} + b^{(1)(1)}, \quad z^{(1)(2)} = w^{(1)(2)}x^{(1)(2)} + b^{(1)(2)}, \quad z^{(1)(3)} = w^{(1)(3)}x^{(1)(3)} + b^{(1)(3)}$$

Andrew Ng

Recap of vectorizing across multiple examples

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

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

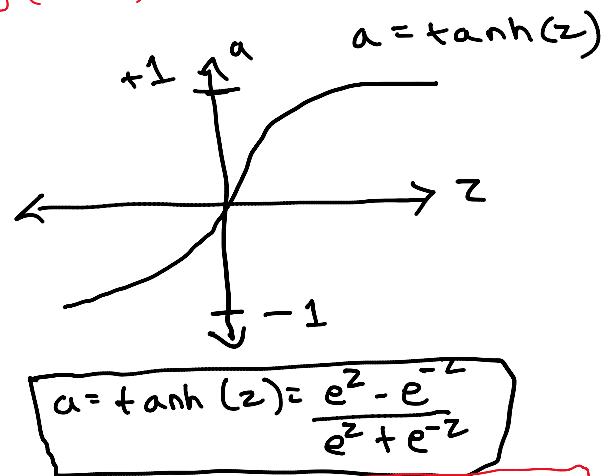
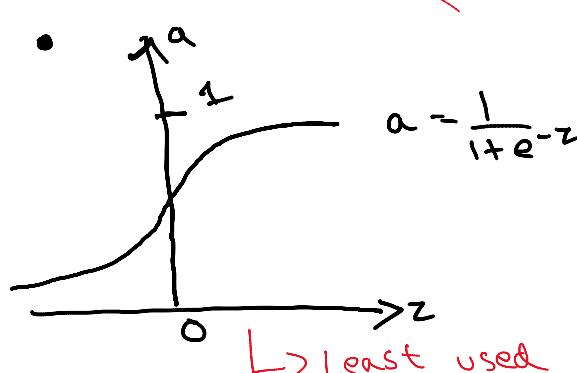
$\text{for } i = 1 \text{ to } m$
 $\rightarrow z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$
 $\rightarrow a^{[1](i)} = \sigma(z^{[1](i)})$
 $\rightarrow z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$
 $\rightarrow a^{[2](i)} = \sigma(z^{[2](i)})$

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

Andrew Ng

⑥ Activation functions

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



→ Downside of sigmoid & tanh is if z is very small or very large, it slows down

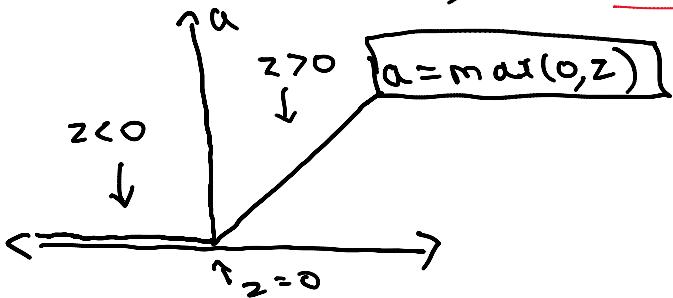
tanh is if z is very small or very large, it slows down gradient descent

for every layer except output layer/binary classifi.

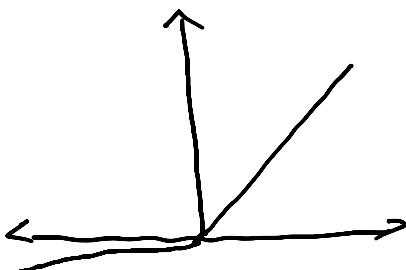
- ReLU (Rectified Linear Unit)

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

→ Default/most used



- Leaky ReLU



⑦ Need for activation functions

- If $g(z) = z \rightarrow$ "Linear activation function" (or) "Identity function"

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

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

$$\Rightarrow a^{[2]} = w^{[2]}(w^{[1]}x + b^{[1]}) + b^{[2]}$$

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

$$\Rightarrow w'x + b' \rightarrow \text{Linear function}$$

output isn't interesting or useful

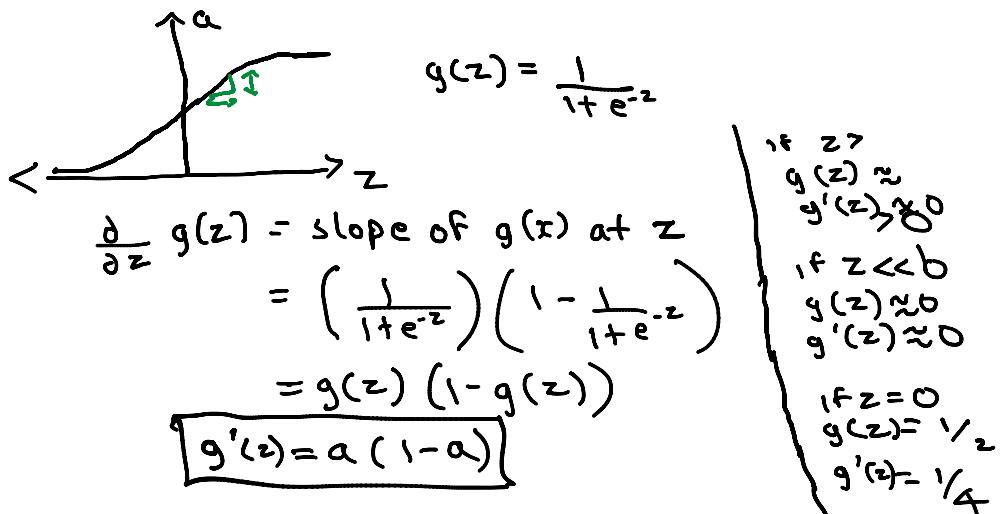
- Linear activation functions are useful only in special cases like house price prediction

⑧ Derivatives of activation functions

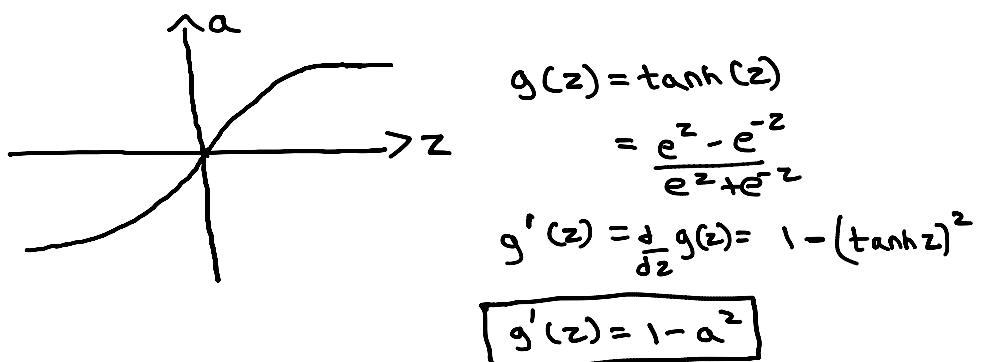
- Sigmoid

$\uparrow a$

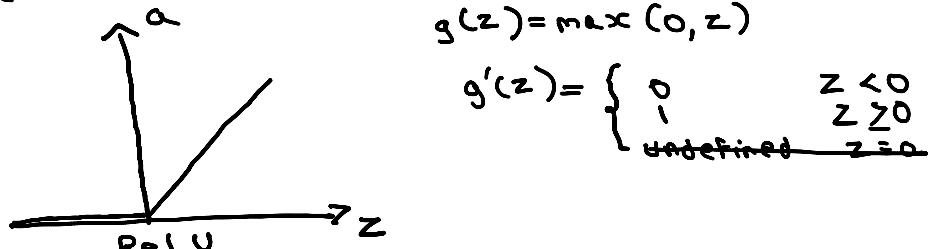
- Sigmoid



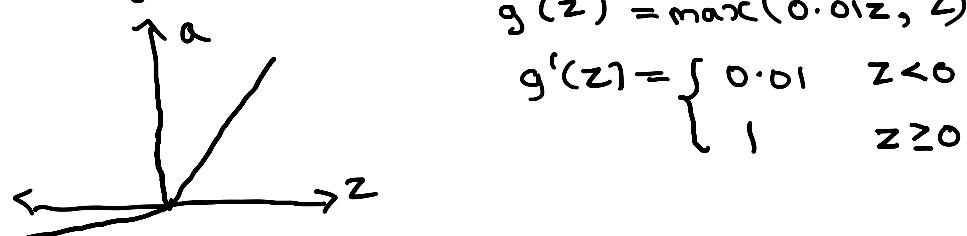
- tanh



- ReLU



- Leaky ReLU



9 Gradient descent for Neural Networks (1 hidden layer)

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

$n_x = n^{[0]} \downarrow \quad n^{[1]} \downarrow \quad n^{[2]} \downarrow$
 Input features hidden unit Output feature (this case)
 $n^{[2]} = 1$

$$\therefore w^{[1]}_{n^{[0]}, 1}, b^{[1]}_{n^{[1]}, 1}, w^{[2]}_{n^{[1]}, n^{[2]}}, b^{[2]}_{n^{[2]}, 1}$$

$$\therefore \omega_{(n^{[1]}, n^{[0]})}^{[1]}, b_{(n^{[1]}, 1)}^{[1]}, \omega_{(n^{[2]}, n^{[1]})}^{[2]}, b_{(n^{[2]}, 0)}^{[2]}$$

- Cost function:

$$J(\omega^{[1]}, b^{[1]}, \omega^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i)$$

- Gradient descent

Repeat {

compute prediction ($\hat{y}^{[i]}$, $i=1.., m$)

$$\delta w^{[1]} = \frac{\partial J}{\partial w^{[1]}}, \delta b^{[1]} = \frac{\partial J}{\partial b^{[1]}}, \dots$$

$$w^{[1]} = w^{[1]} - \alpha \delta w^{[1]}$$

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

$$\begin{aligned} w^{[2]} \\ b^{[2]} \end{aligned} = \dots \quad \}$$

- Formulas for computing derivatives

i. Forward propagation

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

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

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

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

ii. Back propagation

$$\delta z^{[2]} = A^{[2]} - y$$

$$\delta w^{[2]} = \frac{1}{m} \delta z^{[2]} A^{[1]T}$$

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

$$\delta z^{[1]} = \underbrace{W^{[2]T} \delta z^{[2]} * g'^{[1]}(z^{[1]})}_{\substack{(n^{[1]}, m) \times (n^{[2]}, m) \\ \text{elementwise product}}} \quad \begin{array}{l} \text{to output } (n, 1) \\ \text{instead of } (n,) \end{array}$$

$$\delta w^{[1]} = \frac{1}{m} \delta z^{[1]} x^T$$

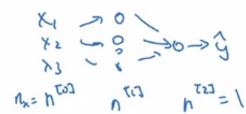
$$\delta b^{[1]} = \frac{1}{m} \text{np.sum}(\delta z^{[1]}, \text{axis}=1, \text{keepdims=True})$$

(10) Backpropagation Intuition

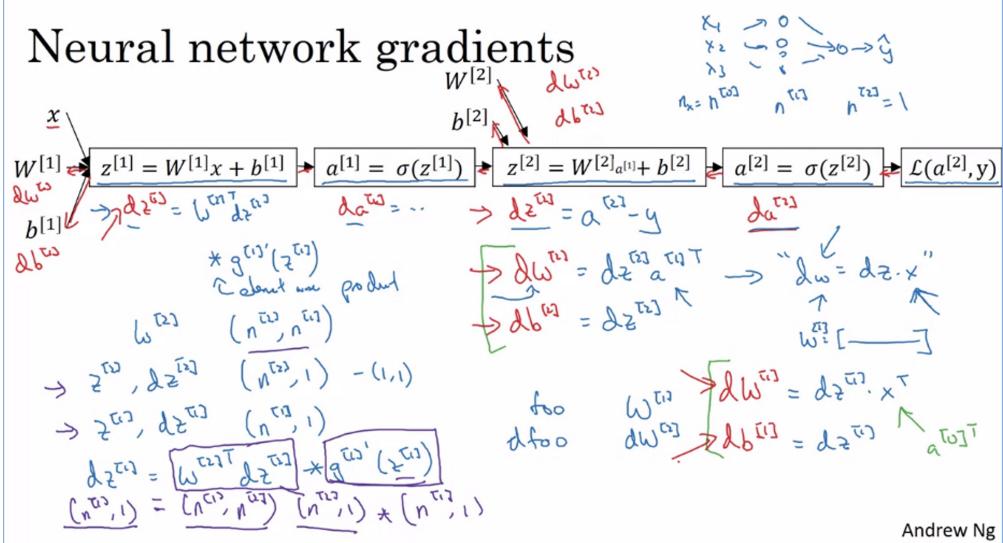
Neural network gradients

x

$$\begin{matrix} W^{[2]} & \downarrow \\ b^{[2]} & \downarrow \end{matrix} \quad \begin{matrix} \delta w^{[2]} \\ \delta b^{[2]} \end{matrix}$$



Neural network gradients



Andrew Ng

Summary of gradient descent

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

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

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

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

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

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

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

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

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

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

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

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

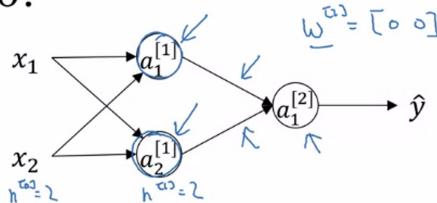
$$\mathcal{J}(\cdot) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(y_i, \hat{y}_i)$$

Andrew Ng

11

Random Initialisation

What happens if you initialize weights to zero?

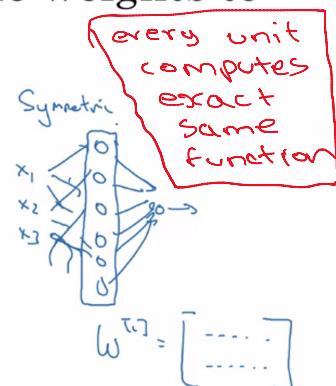


$$w^{[1]} = [0 \ 0]$$

$$a_1^{[1]} = a_2^{[1]} \quad d\hat{y} = d\hat{y}$$

$$\Delta w = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

$$w^{[2]} = w^{[1]} - \Delta w$$



Andrew Ng

• Random initialisation

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

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

Small values
preferred to avoid saturation of function

$w^{[1]} = np.random.rand(2, 2) * 0.01$ to avoid saturation of function

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

$w^{[2]} = \dots$ ↳ initialising b to zero doesn't affect learning

$b^{[2]} = 0$