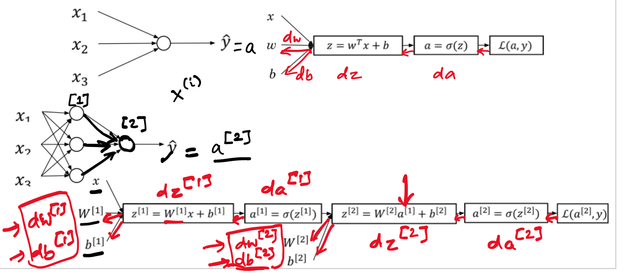
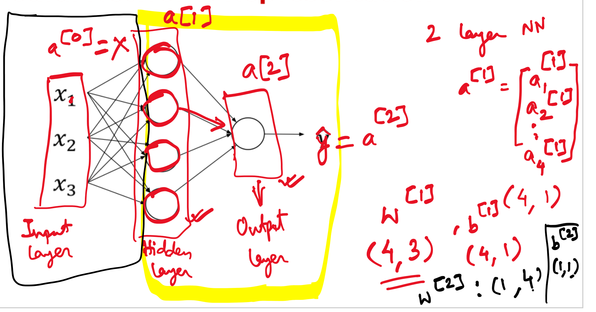
Neural Networks & Deep Learning - Part II

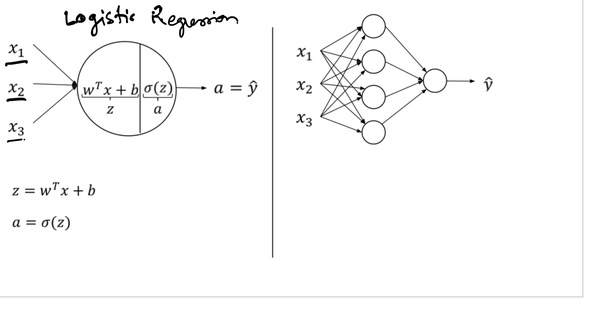
Neural Networks Overview



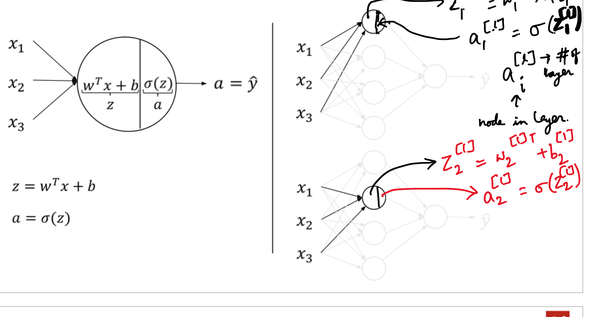
Neural Networks Representation



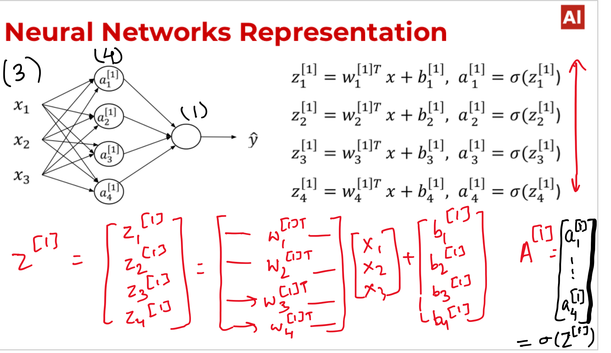
Neural Networks Representation



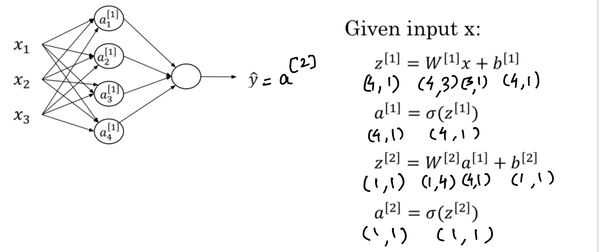
Neural Networks Representation



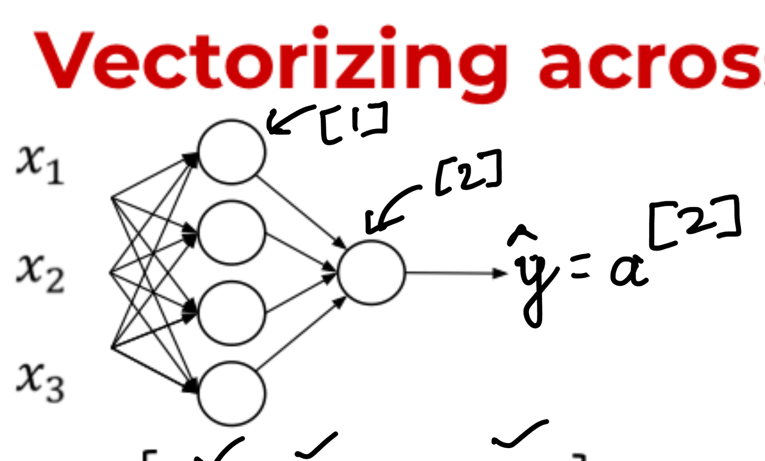
Neural Networks Representation



Neural Networks Representation Learning



Vectorizing across multiple examples



for i = 1 to m

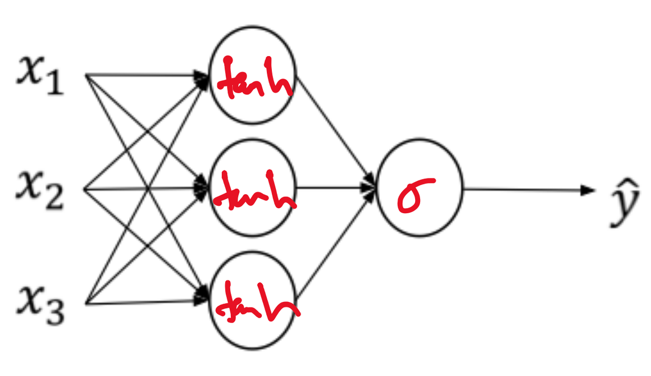
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

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

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

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

Activation functions



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

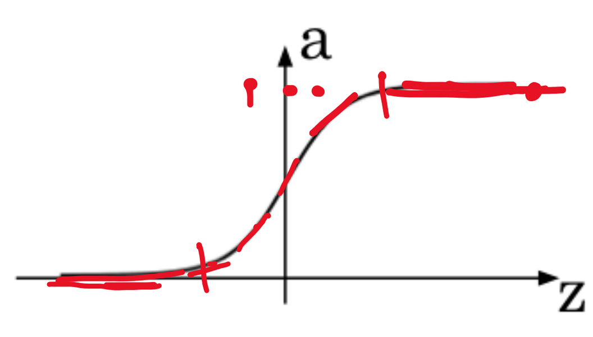
Given x:

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

Sigmoid Activation Function

$W := W - \alpha \, dw$

$g(z)$



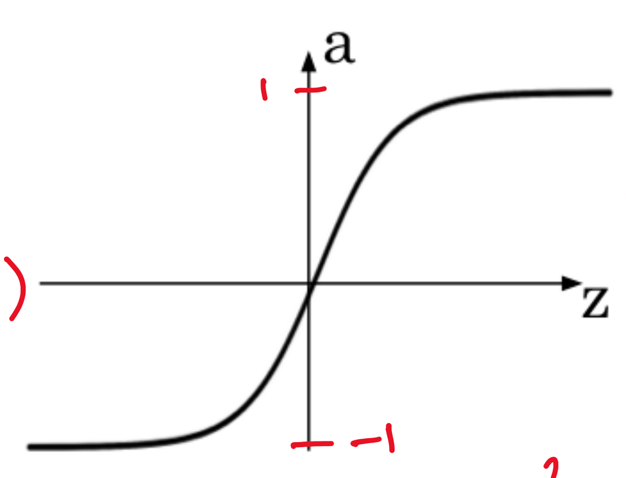
$a \in (0,1)$

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

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

for large z, $g'(z) \approx 0$
for large negative z, $g'(z) \approx 0$

Tanh Activation Function



$a \in (-1, 1)$

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

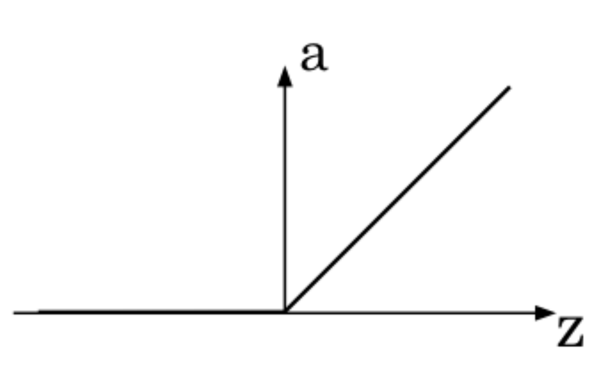
$$g'(z) = 1 - (g(z))^2$$

For $\uparrow z \Rightarrow g'(z) \approx 0$
$\downarrow z \Rightarrow g'(z) \approx 0$

"Vanishing Gradient" Problem

Tanh over Sigmoid
* Centralizes the data
* Next layer's computation is easier

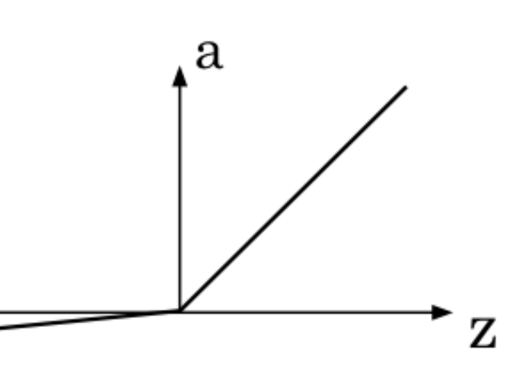
ReLU & Leaky ReLU Activation Functions



ReLU (Rectified Linear Unit)

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

$$g'(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \\ \text{undefined}, & z = 0 \end{cases}$$



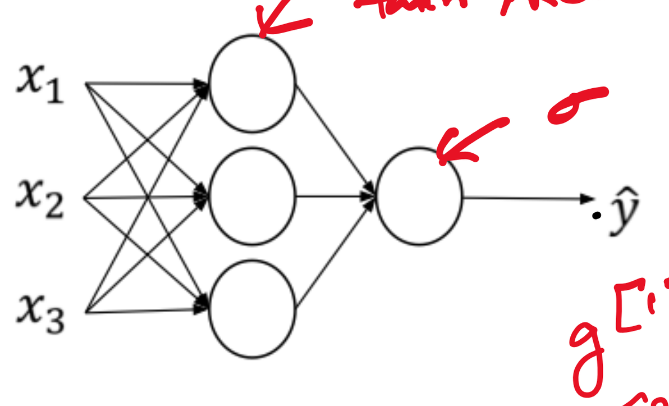
Leaky ReLU

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

$$g'(z) = \begin{cases} 0.01, & z < 0 \\ 1, & z > 0 \end{cases}$$

* Tackles the vanishing grad. problem

Why non-linear activation functions?



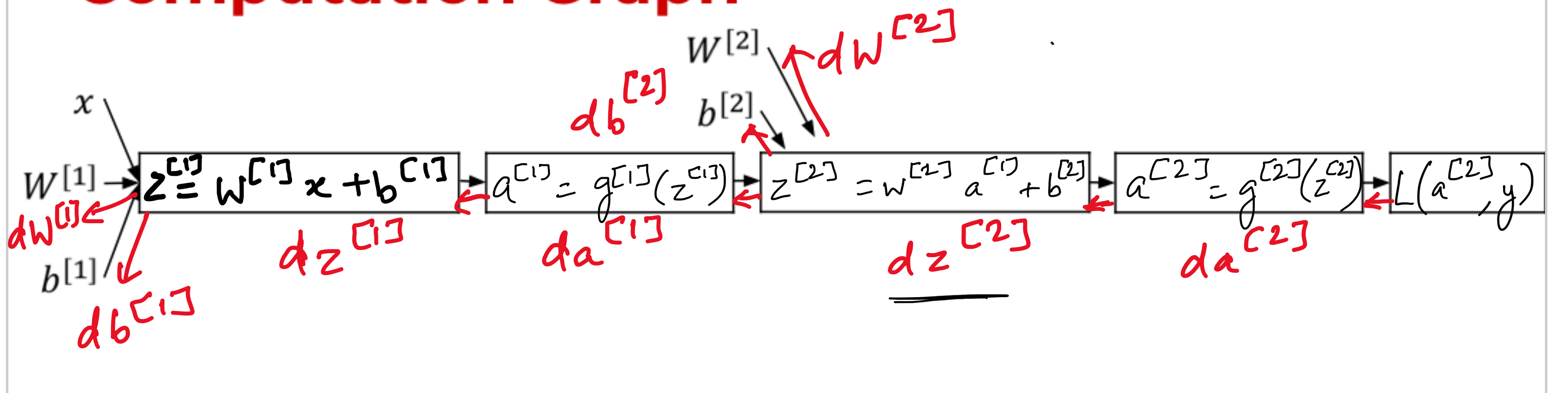
Given x:

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

$g^{[1]}(z) = z$
$g^{[2]}(z) = z$

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

Computation Graph



Gradient descent for neural networks

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

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

Gradient Descent:

Repeat {
  Compute $\hat{y}^{(i)}$
  $dW^{[1]}, db^{[1]}, dW^{[2]}, db^{[2]}$
  $W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$; $b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$
  $W^{[2]} :=$ " ; $b^{[2]} :=$ "
}

Derivative formulas

Vectorized form

$$dz^{[2]} = a^{[2]} - y \qquad dZ^{[2]} = A^{[2]} - Y$$

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

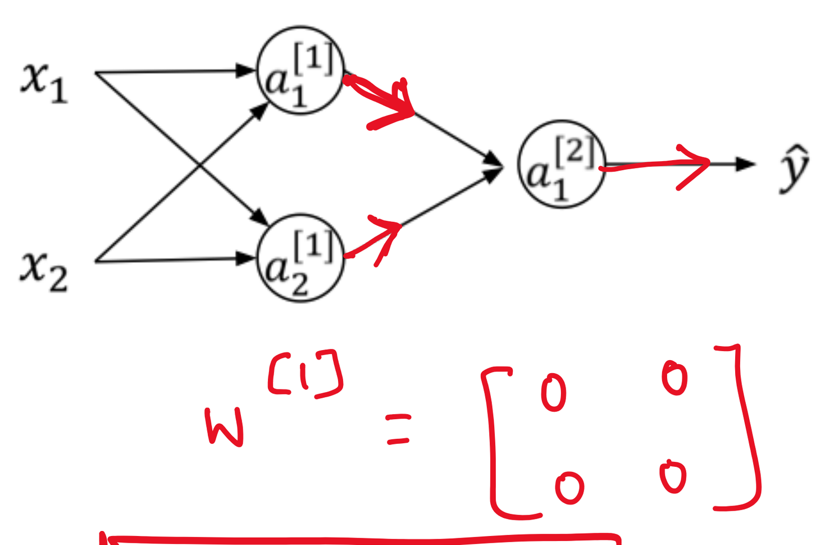
$$db^{[2]} = dz^{[2]} \qquad db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

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

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

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

Zero Initialization

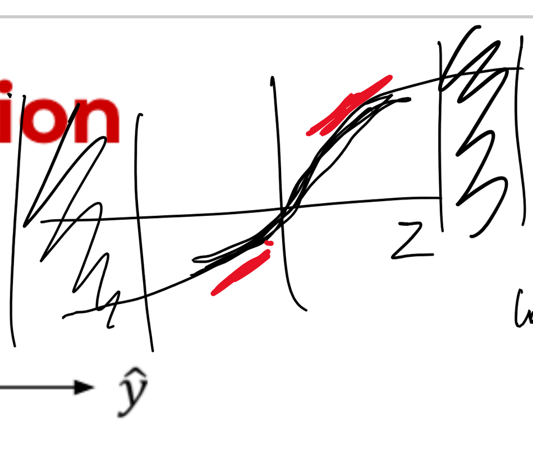


$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad W^{[2]} = [0 \ 0]$$

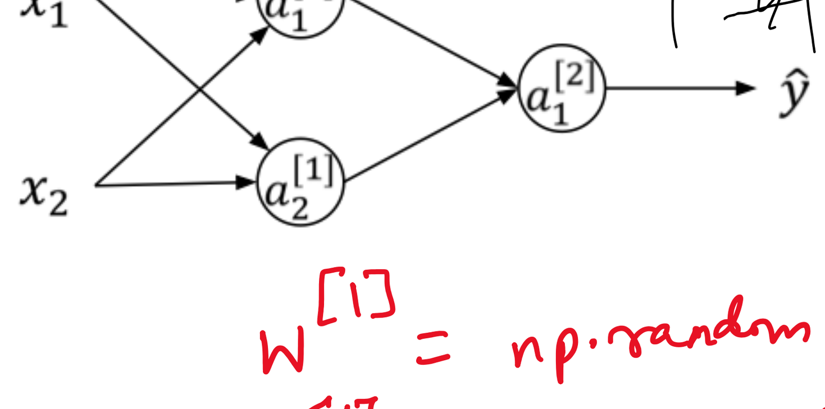
$a_1^{[1]} = a_2^{[1]}$ ✗

* All hidden units/nodes will learn the same weights (Defeats the purpose of having multiple nodes)

Random Initialization



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



$W^{[1]}$ = 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