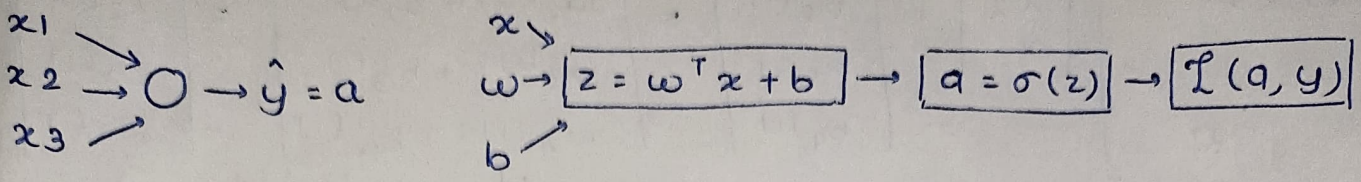
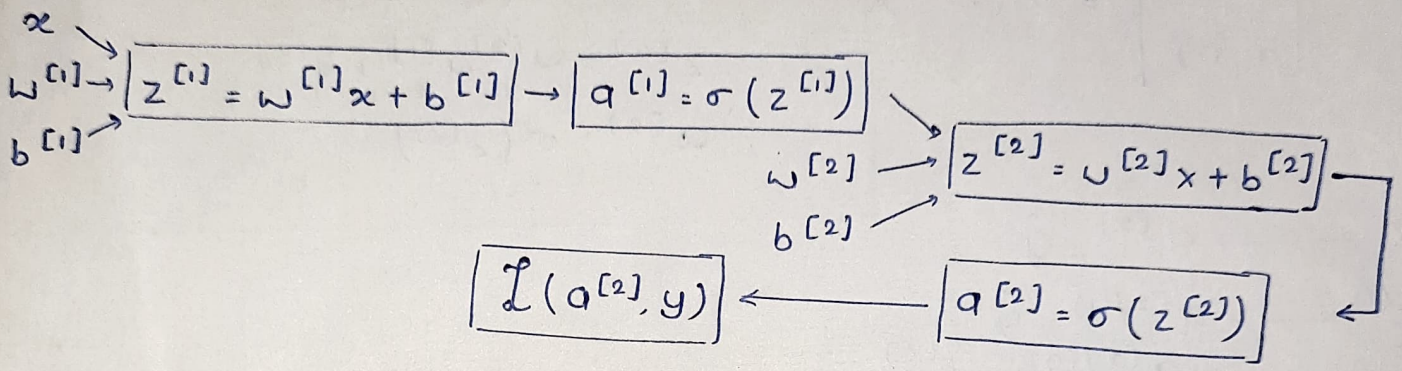
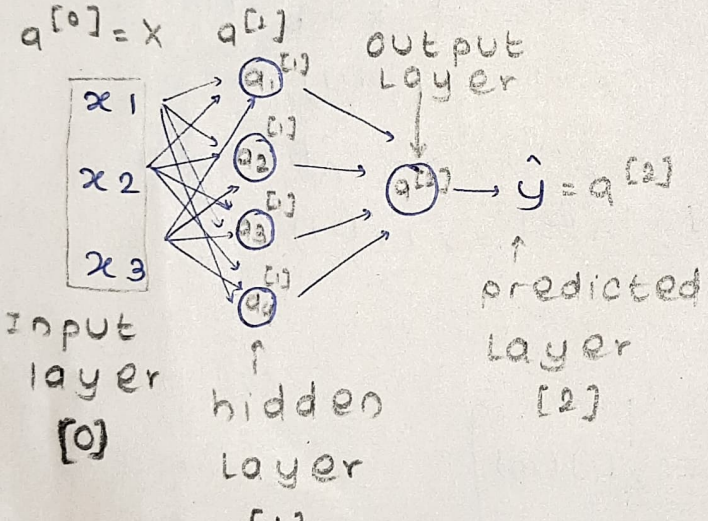
• NEURAL NETWORKS OVERVIEW



- [i] → ith layer
- (i) → ith training example



• NEURAL NETWORK REPRESENTATION:



- Hidden layer: values not observed in training set

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

Hidden layer: $w^{[1]}$, $b^{[1]}$

Nodes → (4, 3) ; (4, 1)

↑ inputs ↑ input

output: $w^{[2]}$, $b^{[2]}$

(1, 4) ; (1, 1)

Node: $\underbrace{w^T x + b}_{z}$ and $\underbrace{\sigma(z)}_{a}$ — w^T: a vector transpose

Layer 1 → node 1: $z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$ & $a_1^{[1]} = \sigma(z_1^{[1]})$

node 2: $z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$ & $a_2^{[1]} = \sigma(z_2^{[1]})$

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

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

Given input x: $x = a^{[0]}$

$(4,1) \rightarrow z^{[1]}_{(4,1)} = \underset{(4,3)}{w^{[1]}} \underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$

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

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

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

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

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

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

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

for $i = 1$ to m:

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

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

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

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

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

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

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

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

$$Z^{[1]} = \begin{bmatrix} | & | & | & & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots & z^{[1](m)} \\ | & | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & | & & | \\ a^{[1](1)} & a^{[1](2)} & a^{[1](3)} & \cdots & a^{[1](m)} \\ | & | & | & & | \end{bmatrix}$$

↕ input features

↔ training examples

° VECTORIZED IMPLEMENTATION:

$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}$; $z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}$; $z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$

$$w^{[1]} = \begin{bmatrix} — \\ — \\ — \\ — \end{bmatrix} \quad w^{[1]} x^{(1)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad w^{[1]} x^{(2)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad w^{[1]} x^{(3)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

$$X = \begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix} \Rightarrow w^{[1]} X = \begin{bmatrix} \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

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

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

$\Rightarrow \boxed{z^{[i]} = w^{[i]} x + b^{[i]}}$

* x stacked in column vector $\Rightarrow$ z stacked in col. vec (input) (output)

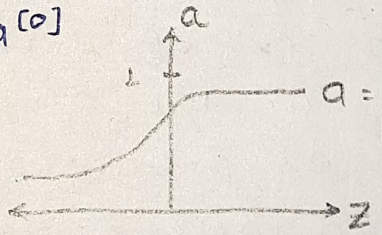
° ACTIVATION FUNCTION: (sigmoid is one choice)

$z^{[1]} = w^{[1]} x + b^{[1]}$ ... $x = a^{[0]}$

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

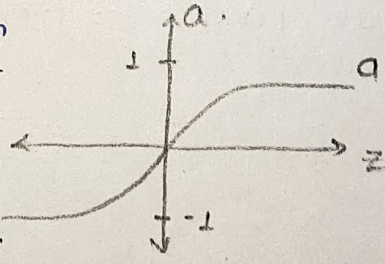
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

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



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

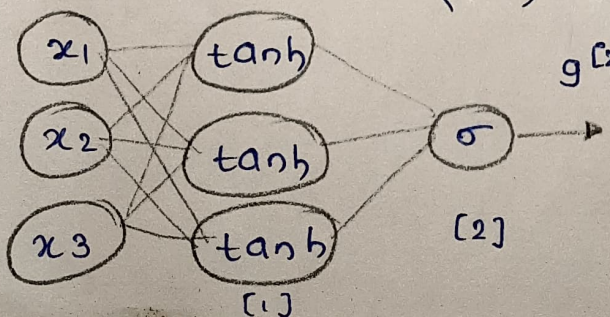
NOTE: For hidden layers $g(z^{[i]}) = \tanh(z^{[i]})$ works better than $\sigma(z^{[i]})$ as mean is closer to zero.



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

Exception: σ activation function works better for output layer in binary classification $\boxed{0 \le \hat{y} \le 1}$

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

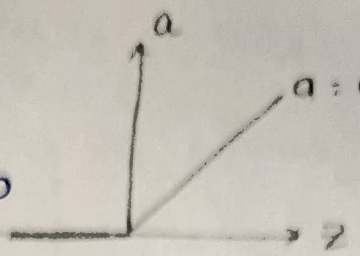


$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

° limitation:

If $z >>> 0$ or $z <<< 0$ then slope ≈ 0 slows gradient descent.

Better choice = ReLU (Rectified Linear Unit)



RULES:

(i) For binary classification
output layer ... $\hat{y} \in \{0,1\}$
use : SIGMOID FUNCTION $0 \leq \hat{y} \leq 1$

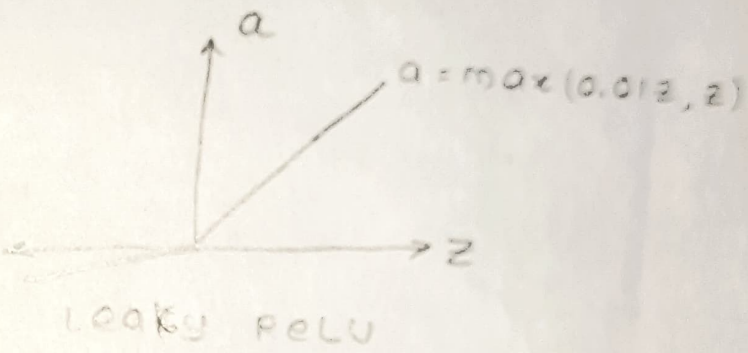
(ii) For all other cases ... activation fx? = ReLU
when z = -ve, slope = 0
* Leaky ReLU better than ReLU

NOTE: (ReLU) faster than sigmoid / tanh

σ : 0 to 1

tanh : -1 to 1

ReLU ⇒ introduces
non-linearity
$\hat{y} \geq 0$



* Linear / identity activation function : $g(z) = z$
output = linear fx? of input
- only useful for ML on a regression problem
(when $y \in \mathbb{R}$ ... ex. prediction of housing price)
Here : hidden layers = ReLU or tanh
output layer = linear activation function

• DERIVATIVES OF ACTIVATION FUNCTIONS:

sigmoid:

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

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

slope of $g(z)$ at z $= g(z)(1-g(z)) = g'(z)$

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

If: (i) $z >>>$ +ve no ... $g'(z) = 0$

(ii) $z <<<<$ -ve no ... $g'(z) = 0$

(iii) $z = 0$ .... $g'(z) = \frac{1}{4}$

tanh:

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

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

$$= 1 - [\tanh(z)]^2$$

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

If: (i) $z >>>$ +ve no ... $g'(z) = 0$

(ii) $z <<<$ -ve no ... $g'(z) = 0$

(iii) $z = 0$ .... $g'(z) = 1$

ReLU:

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

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

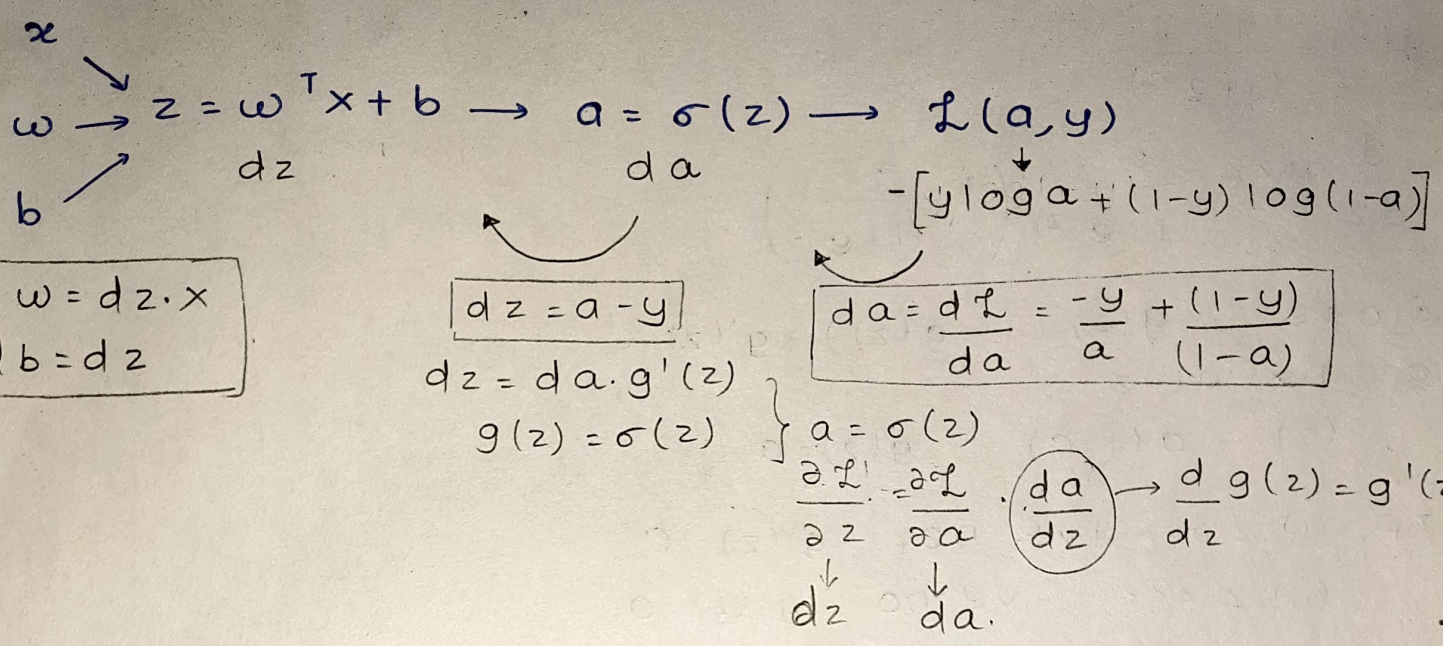
* Technically undefined for $\boxed{z = 0}$

Leaky ReLU: $g(z) = \max(0.01z, z)$

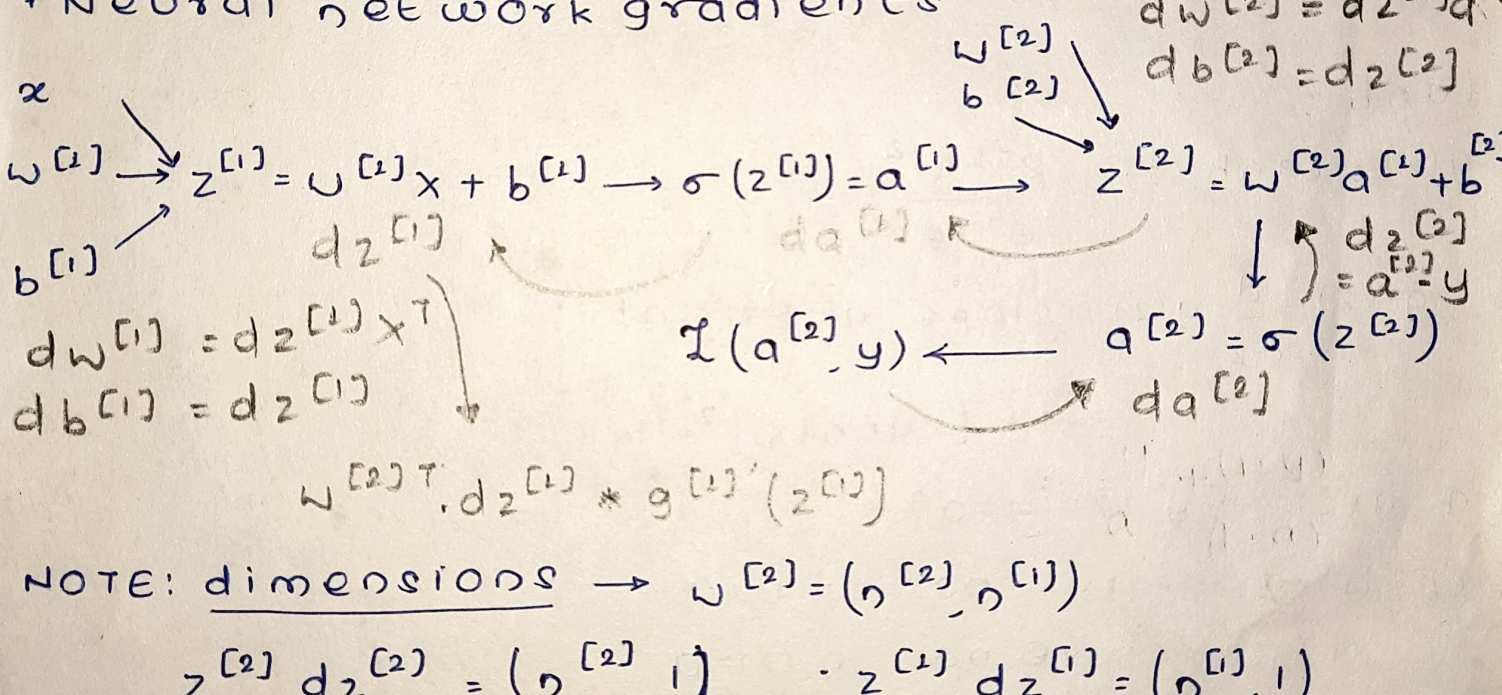
$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

• COMPUTING GRADIENTS: $a = \hat{y}$ $\sigma(z) = \frac{1}{1+e^{-z}}$

→ Logistic Regression



* Neural network gradients



NOTE: dimensions → $w^{[2]} = (n^{[2]}, n^{[1]})$

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

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

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

IMP * ⇒ element wise product

np.dot(a, b) ⇒ Matrix multiplication.

• summary of Gradient descent

axis=0
Horizontal

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

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

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

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

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

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

vectorized

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

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

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

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

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

for backprop

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

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

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

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

↑ vertical

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

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

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

↑ vertical

- If all weights are initialized to zero, all hidden units compute the same function

  * Large z slows down learning

- **RANDOM INITIALIZATION**

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

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

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

$b^{[2]} = 0$

NOTE: $W^{[k]}$ dimensions $\Rightarrow$ (no. of neurons in k^{th} layer, no. of inputs of layer)

$b^{[k]}$ dimensions $\Rightarrow$ (no. of neurons in k^{th} layer, 1)