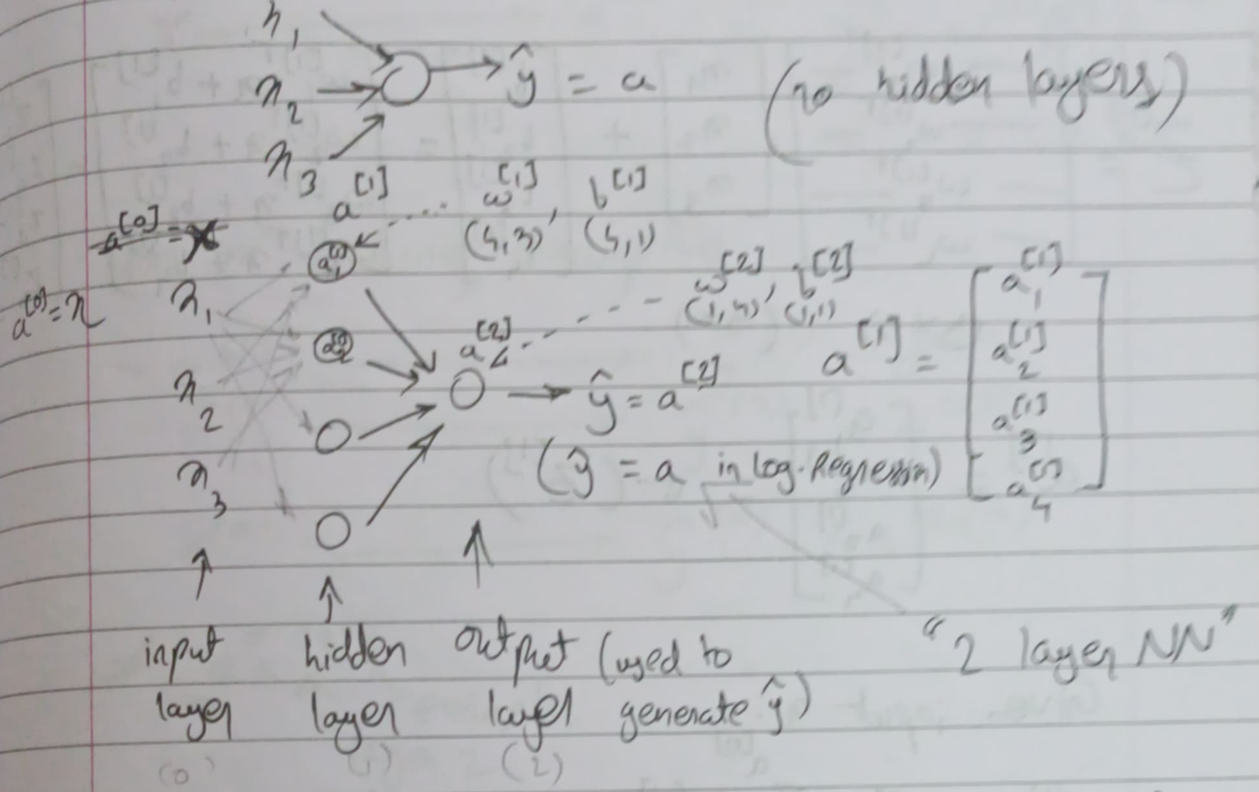
week 3 : course 1

One hidden layer Neural Network :



(no hidden layers)

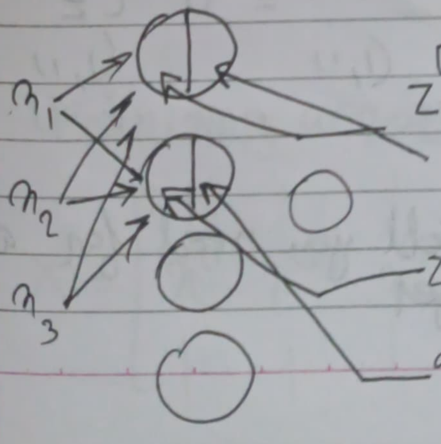
$(\hat{y} = a$ in log. Regression$)$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

"2 layer NN"

In supervised learning, whatever calculations are done in hidden layer are not seen during training phase, hence the name hidden.

Computing NN's output:



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

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

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

$[l] \leftarrow$ layer
$a_i \leftarrow$ node

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

$$Z^{[1]} = \underset{(4\times 3)}{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}} \underset{(3\times 1)}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(Z^{[1]})$$

Given input x:

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

$||^{ly}$,

$$★ \underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \underset{(4,1)}{\overset{a^{[1]} \nearrow}{x}} + \underset{(1,1)}{b^{[2]}} \quad , \quad ★ \underset{(1,1)}{a^{[2]}} = \underset{(1,1)}{\sigma(z^{[2]})}$$

4-★ lined codes is all you need for a NN with one hidden layer.

vectorizing across multiple examples :

For single training example :

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

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

for 'm' training examples :

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

$\#\ a^{[2]i}$ example i

layer 2

for i = 1 to m,

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

Vectorizing across multiple examples :

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix} \quad (n_x, m)$$

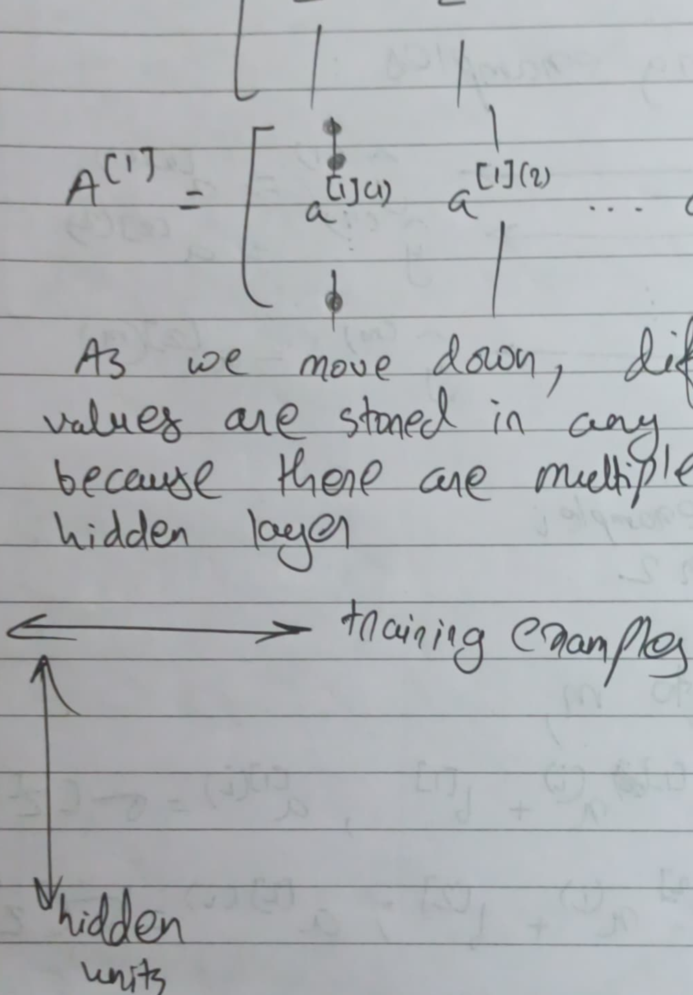
$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} \overset{A^{[0]}}{X} + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \end{aligned}\right\} \text{Vectorized form}$$

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

1st hidden unit on final traing example of 1st hidden layer

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

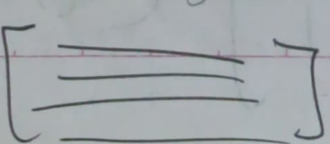
As we move down, different activation values are stored in any particular column because there are multiple nodes ~~for~~ in a hidden layer



Justification for vectorized implementation:

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

$W^{[1]} =$



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

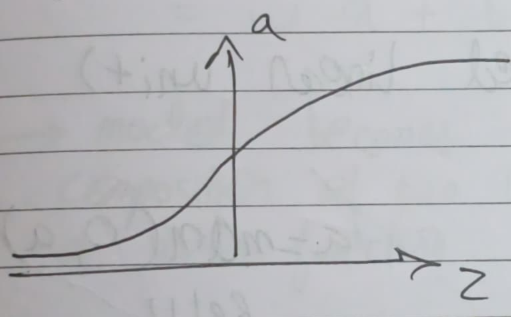
→ column vectors

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

$$= Z^{[1]}$$

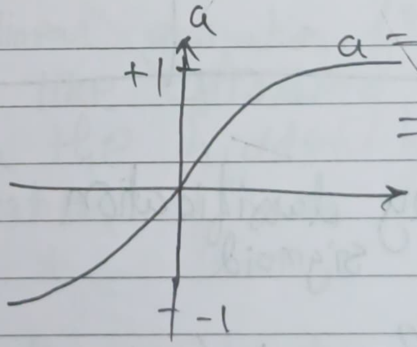
Activation function:

Sigmoid is an activation f^n



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

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



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

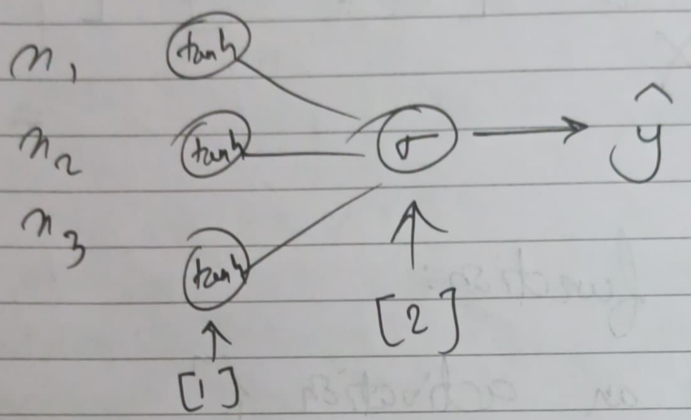
Optional

Using this function as activation f^n, the mean shifts ~~to~~ closer to zero hence when the algorithm requires to center the data, we would have zero mean.

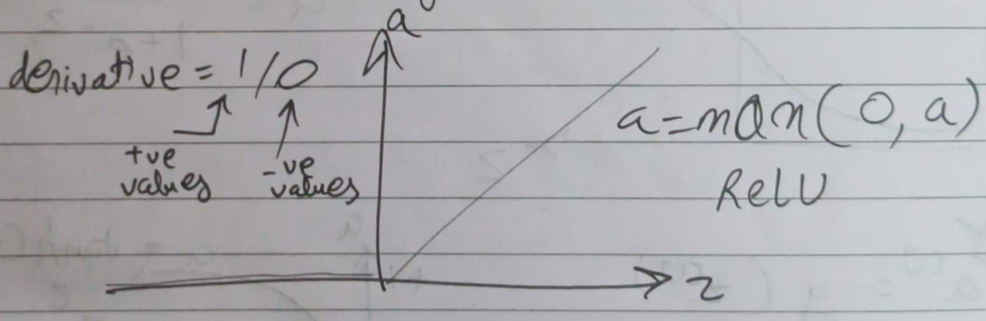
Leaky ReLU = $\max(0.01z, z)$

when we talk about binary classification, we would want $\hat{y}$ to range between 0 & 1, for which sigmoid would be preferred as compared to ~~[illegible]~~ tanh.

Also, activation f^{ns} can be different for different layers like $g^{[1]}(z^{[1]}) = \tanh$
$g^{[2]}(z^{[2]}) = \sigma$



★ ReLU (Rectified linear unit)



if Binary classification
sigmoid
else
ReLU / tanh

more preferred because fast learning as compared to others as slope = 1 in most cases

Why do you need non linear activation f^{ns}?

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

$$= z^{[1]}$$

$g(z) = z$

"linear activation f^{n}"

/ identity activation f^{n}

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

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

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

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

$$= w'x + b'$$

→ model becomes useless because the composition of two linear functions is itself a linear function.

We may use linear activation f^{n} only in output layer like estimating the house prices in which the hidden layers may contain the activation f^{ns} as ReLU or tanh.

Derivatives of activation functions:

∆ Sigmoid fⁿ

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

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

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

$$= g(z)\left(1 - g(z)\right) = a(1-a)$$

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

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

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

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

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

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

∆ Tanh

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

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

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

$g'(z) \approx 0$

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

$g'(z) \approx 0$

$z = 0$, $\tanh z \cong 0$

$g'(z) = 1$

* ReLU and Leaky ReLU

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

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

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

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

Gradient descent for NN (1 hidden layer)

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

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

Cost f^n : $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

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

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

$n_x = n^{[0]}$ (input features)

$n^{[1]}$ (hidden units)

$n^{[2]}$ (output units)

$= 1$ (in our examples)

Gradient descent :

Repeat {

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

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ...

$$w^{[1]} := w^{[1]} - \alpha \, dw^{[1]}$$
$$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$$
$$w^{[2]} :=$$
$$b^{[2]} :=$$

Formulas for computing derivatives

Formulas for forward prop :

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

Back Propogation :

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

Similar to logistic reg: $dZ^{[2]} = A^{[2]} - Y$

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

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

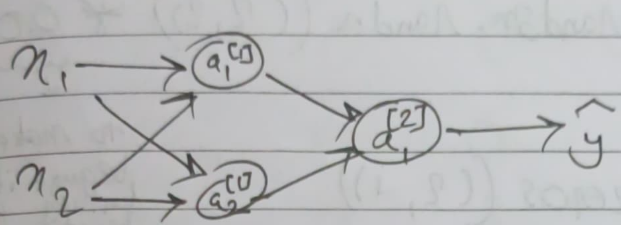
$$db^{[2]} = \frac{1}{m} \text{np.sum}(dZ^{[2]}, \text{axis}=1, \text{keepdims}=\text{true})$$

$$\underset{(n^{[1]}, m)}{dZ^{[1]}} = \underbrace{W^{[2]T} dZ^{[2]}} \underset{\uparrow \text{ element-wise Product}}{*} \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

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

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

Random Initialization



$$w^{[2]} = [0 \quad 0]$$

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

$$W^{[1]}_{(2,2)} = \cancel{(2,2)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

i.e. the hidden units of layer 1 are exacty identical (they are symmetric)

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

after which, $w^{[1]} = w^{[1]} - \alpha\, dw$

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

In summary, when initialized with zero,

it is actually pointless to have more than one hidden units in the hidden layer as all of them would be computing the same thing.

Solution to this is initialize the parameters randomly

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

to make it smaller because it has to be further given to the activation functions...

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

b can be initialized to zero as long as W is randomly initialized because it defeats the symmetry problem encountered.

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

$b^{[2]} = 0$

... like tanh and sigmoid so we may end up with value~~s~~ outputs like 1 where the slope / gradient is very small ie. gradient descent will be slow hence, learning will be slow.