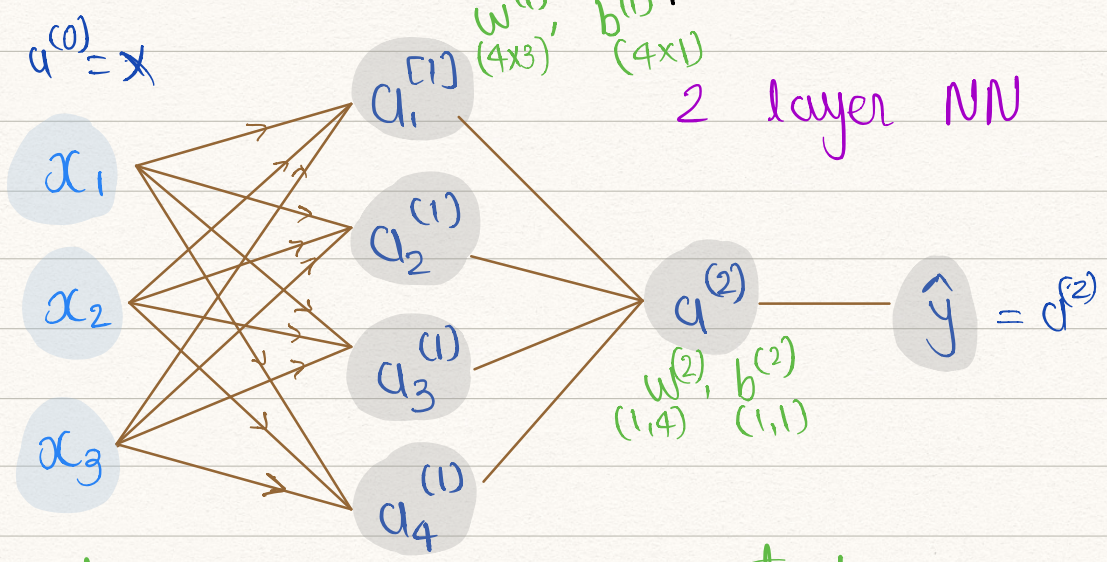
Neural Network Representation:



2 layer NN

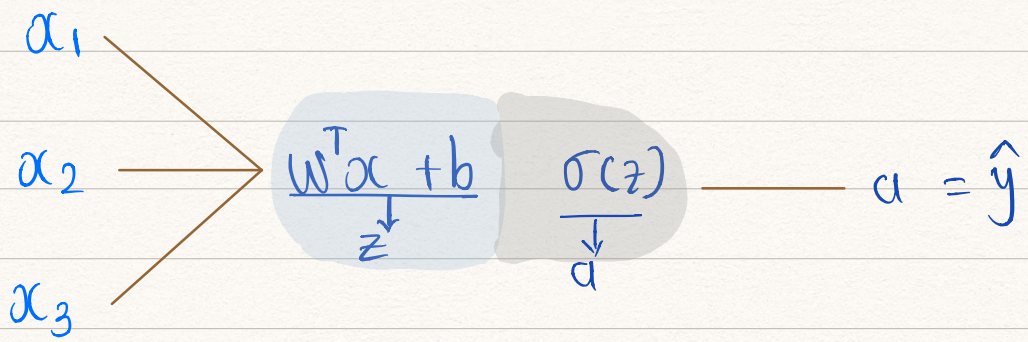
input layer (0) | hidden layer (1) | output layer (2)

$$a^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_4^{(1)} \end{bmatrix}$$

$\hat{y} = a^{(2)}$ because in logistic reg. we only get single o/p from N.N.

$w^{(1)} \longrightarrow (4 \times 3) \longrightarrow$ 4 output unit
3 input unit

Computing NN's Output:



$a_i^{(l)}$ → layer number
→ node in layer

$$z_1^{(1)} = W_1^{(1)T} x + b_1^{(1)}$$
$$a_1^{(1)} = \sigma(z_1^{(1)})$$
} layer 1, node 1 computation

$$z_2^{(1)} = W_2^{(1)T} x + b_2^{(1)}$$
$$a_2^{(1)} = \sigma(z_2^{(1)})$$
} layer 1, node 2 computation

Vectorize the whole process:

$$\begin{bmatrix} — W_1^{(1)T} — \\ — W_2^{(1)T} — \\ — W_3^{(1)T} — \\ — W_4^{(1)T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \\ z_4^{(1)} \end{bmatrix} = z^{(1)}$$

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

given input x:

$$\underset{(4\times1)}{z^{(1)}} = \underset{(4\times3)}{W^{(1)}} \underset{(3\times1)}{x} + \underset{(4,1)}{b^{(1)}} \qquad (x = a^{(0)})$$

$$\underset{(4,1)}{a^{(1)}} = \underset{(4,1)}{\sigma(z^{(1)})}$$

Algorithm:

for $i = 1$ to m:

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

} 4 lines of for loop

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

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

→ training examples.

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

↓ hidden units

Explaination for vectorized implementation:

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

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

$$w^{[1]} \begin{bmatrix} | & | & \\ x^{(1)} & x^{(2)} & \cdots \\ | & | & \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \\ \cdot & \cdot & \cdots \\ \cdot & \cdot & \\ \cdot & \cdot & \end{bmatrix} = \begin{bmatrix} | & | & \\ z^{[1](1)} & z^{[1](2)} & \cdots \\ | & | & \end{bmatrix}$$

$+b^{[1]}$ $+b^{[1]}$

$W^{[1]} \cdot X$ $\qquad Z^{[1]}$

↓ Set of all training examples

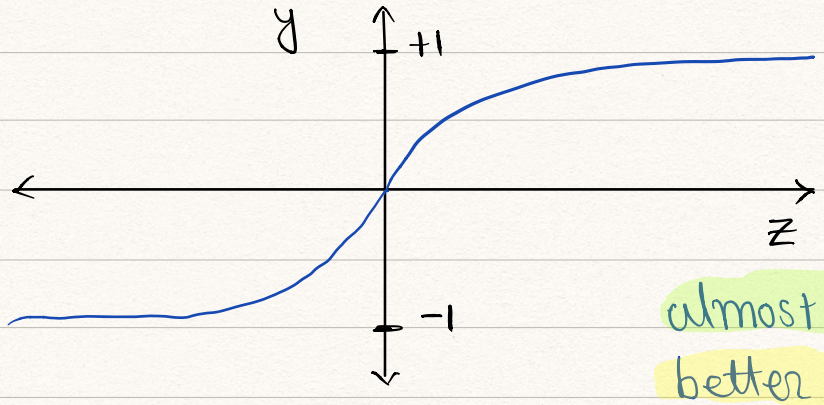
$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \end{aligned}\right\} \text{vectorization of 4 lines in for loop}$$

Activation Functions:

Sigmoid Function:

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

tanh Function:



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

almost always works better than sigmoid

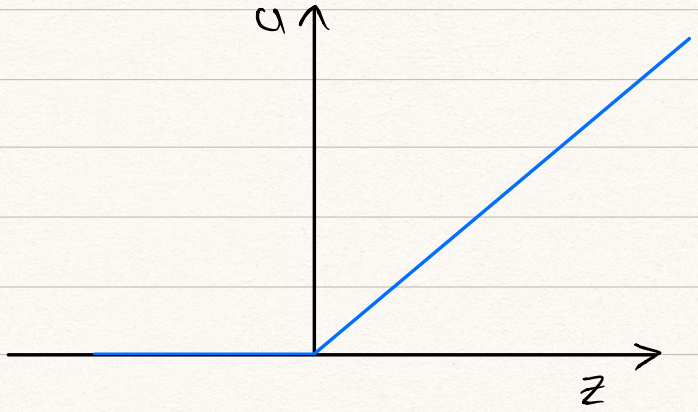
- have close to zero mean
- centering the data
- easier for further calculation

→ when you have binary classification output layer should be sigmoid.
keep hidden layers as tanh.

→ You can have different activation function for different layers.

→ When you have very large or very small i/p, slope of function is close to zero, so it can slow down gradient decent.

Rectified Linear Unit:



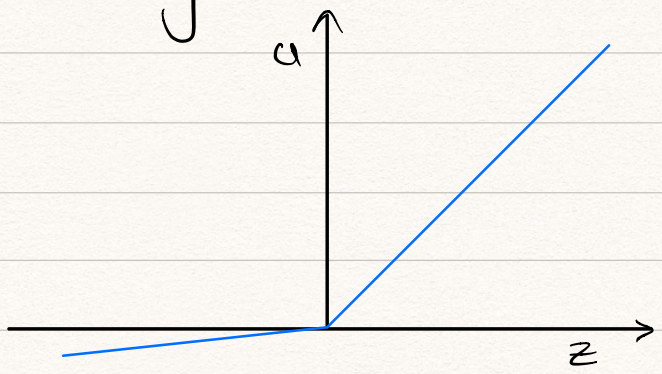
ReLU

$a = \max(0, z)$

Disadvantage:

→ Slope is 0 when i/p is negative

Leaky ReLU:



- Works better than ReLU.

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

Why do we need non linear Activation Functions?

linear activation function:

$g(z) = z$

↳ identity function.

$$a^{(1)} = z^{(1)} = w^{(1)}x + b^{(1)}$$

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

$$a^{(2)} = w^{(2)}(w^{(1)}x + b^{(1)}) + b^{(2)}$$

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

$$a^{(2)} = w'x + b'$$

No matter how many layer NN has its just doing linear activation.
No benefit of doing this.

You can use linear activation on o/p layer. not recommended at hidden layer.

Derivatives of Activation Functions:

When we implement back prop. on NN we need to compute slope or the derivative of activation function.

Sigmoid Activation Function:

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

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

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

$$= a(1-a)$$

tanh activation:

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

$$g'(z) = \frac{d}{dz}(g(z)) = 1-(\tanh(z))^2$$

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

$$= 1-a^2$$

ReLU & Leaky ReLU:

$g(z) = \max(0, z)$ → ReLU

$g'(z) = 0$ if $z < 0$

1 if $z \geq 0$

~~undefined~~ if $z = 0$

↳ due to discontinuity.

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

$g'(z) = 0.01$ $z < 0$

1 $z \geq 0$

~~undefined~~ ~~$z = 0$~~

Gradient Decent for NNs:

Parameters: $W^{(1)}$, $b^{(1)}$, $W^{(2)}$, $b^{(2)}$

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

$n_x = n^{(0)}, n^{(1)}, n^{(2)} = 1$

Cost Function: $J(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$

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

($\hat{y} = a^{(2)}$)

Repeat : { compute predictions: $(\hat{y}^{(i)}, i = 1, 2, \ldots, m)$

$dW^{(1)} = \frac{\partial J}{\partial W^{(1)}}$, $db^{(1)} = \frac{\partial J}{\partial b^{(1)}}$

$\frac{\partial}{\partial W^{(1)}}$, $\frac{\partial}{\partial b^{(1)}}$

update: $W^{(1)} = W^{(1)} - \alpha\, dW^{(1)}$

$b^{(1)} = b^{(1)} - \alpha\, db^{(1)}$

Formulas for Computing derivatives:

Forward Propogation:

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

Back Propogation:

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

$$dW^{(2)} = \frac{1}{m} dZ^{(2)} A^{(1)T}$$

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

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

$$dZ^{(1)} = \underbrace{W^{(2)T} dZ^{(2)}}_{(n^{(1)}, m)} * g^{(1)'}(Z^{(1)}) \rightarrow (n^{(1)}, m)$$

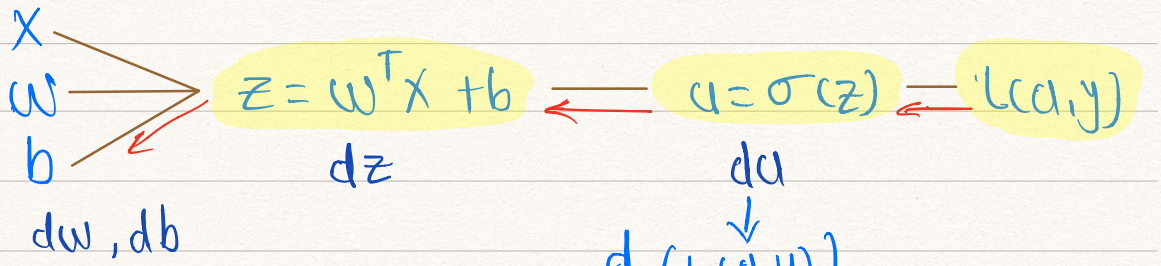
$*$ ↓ elementwise product

$$db^{(1)} = \frac{1}{m} \cdot np.sum(dZ^{(1)}, axis=1, keepdims=T)$$

$$dw^{(1)} = \frac{1}{m} dz^{(1)} \cdot x^T$$

Backpropogation Intuition:

Computing Gradient:



$$\frac{d}{da}(L(a,y))$$

$$= \frac{d}{da}(-y \log a - (1-y) \log(1-a))$$

$$da = -y/a + \frac{1-y}{1-a}$$

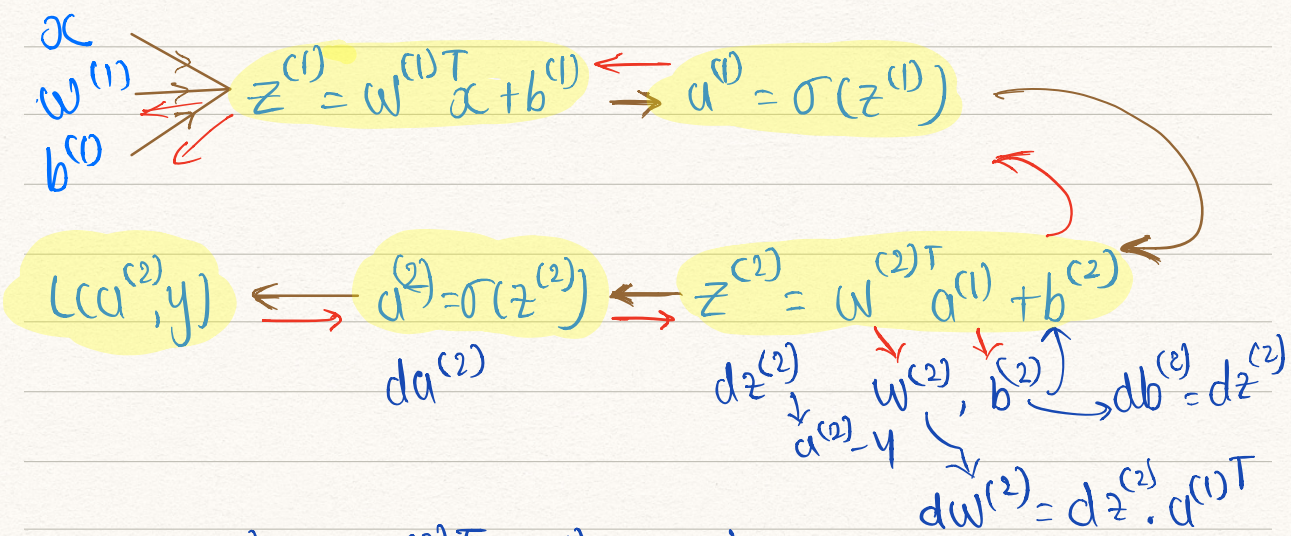
$$dz = da \cdot g'(z) \rightarrow g(z) \rightarrow \text{sigmoid } \sigma(z)$$

$$a = \sigma(z)$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \quad \frac{d}{dz}(\sigma(z))$$

$$dw = dz \cdot x$$

$$db = dz$$



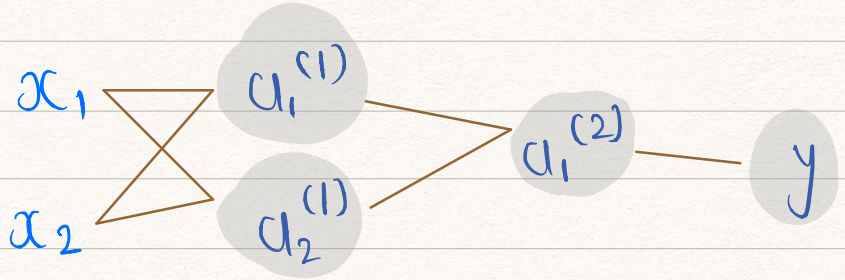
$$dz^{(1)} = w^{(2)T} dz^{(1)} * g^{(1)'}(z^{(1)})$$

$(n^{(1)}, n^{(2)})$ $(n^{(2)}, 1)$ $*$ $(n^{(1)}, 1)$ → element wise product

$w^{(2)}$ $(n^{(2)}, n^{(1)})$

Random Initialization:

What happens if you initialize weights to zero?



$$w^{(1)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{(1)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$a_1^{(1)} = a_2^{(1)}$ $\quad dz^{(1)} = dz^{(2)}$

$dw = \begin{pmatrix} u & v \\ u & v \end{pmatrix}$ $\quad W^{(1)} = W^{(1)} - \alpha\, dw$

$W^{(1)} = \begin{bmatrix} - & - & - & - \\ - & - & - & - \end{bmatrix}$ identical rows.

If you initialize with zero, hidden layer will be symmetric and we will keep on computing same function, regardless of no. of layers.

So we do...

$W^{(1)} = np.random.randn(2,2) * 0.01$ — multiply with small no.

$b^{(1)} = np.zeros((2,1))$ — it is okay to have zero

↳ we do not have symmetry breaking problem

$W^{(2)} = random$

$b^{(2)} = 0$

→ If w values are too large, z will be very big or very small. in which case we will not get good gradient decent beccause it will have dw close to zero.
(end up at flat part of activation function)