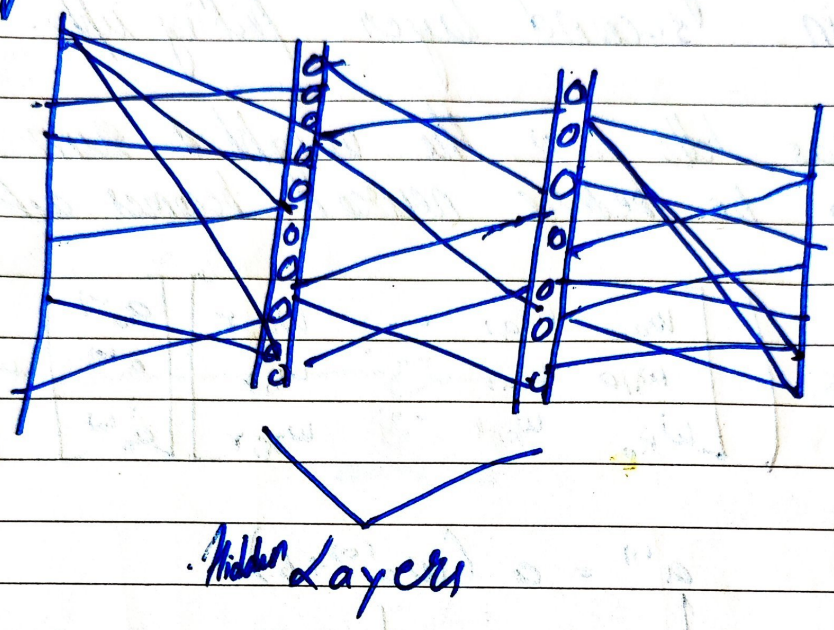
Date: __/__/____

Neural networks

- Chap-1, Deep learning.

0 — 1
↓
neuron

layers :-



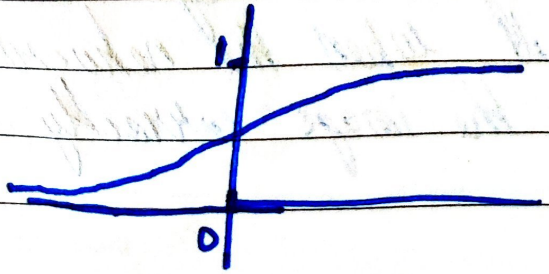
Hidden Layers

- The activation in 1 layer determines the activation in next layer
- Firing of some neurons cause some other neurons to fire.

- ~~Any general loopy kinda of fed~~
- layers can do the job of ~~dividing~~ looking for components ~~in an~~ / pattern.

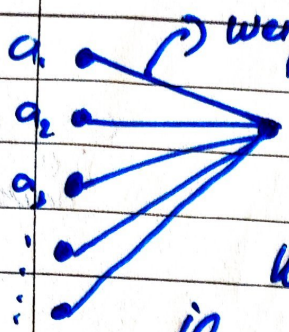
- Sigmoid function :- (σ)

$a(x) = 1/1+e^{-x}$



Range (0-1)
used for activation values.

Date: __/__/____



$\sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n) + b_0$

bias

Weight tells us what pattern this neuron in second layer picking upto.

Bias tells how the weighted sum needs to be before neuron becomes active.

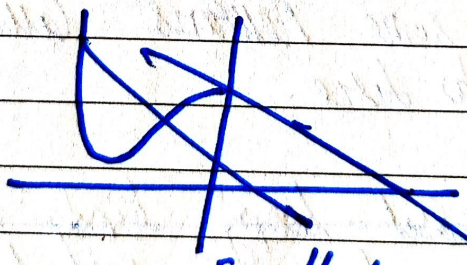
$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,r} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,r} \\ w_{k,0} & w_{k,1} & \cdots & w_{k,r} \end{bmatrix} \begin{bmatrix} a_0^{(-1)} \\ a_1^{(-1)} \\ \vdots \\ a_n^{(-1)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

$$a^{(1)} = \sigma(w a^{(0)} + b)$$

$a^{(1)}$ ↓ 1st layer

$a^{(0)}$ ↓ initial layer

~~cost~~



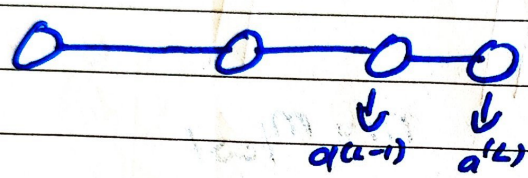
- Cost functions :- For that you add up the sq. of the differences between each these outputs activations and the value that you want them to have and this is just the cost of a single training ~~programme~~ Eg.
(The sum is small when the network confidently classifies the image correctly

Date: __/__/____

But it's large when the network doesn't really know what it's doing).
So when you calculate the avg. costs of all the training examples.

- The avg. cost is our measure for how lousy our network is.

- Consider 4 networks



(we are considering only last 2)

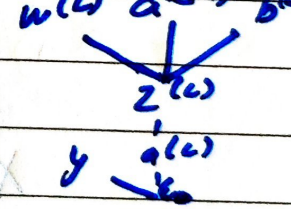
(y) → desired output

Cost → $C_0 = (a^{(L)} - y)^2$

$a^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)} = z^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$

- Computing relevent derivatives.



$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot a^{(L-1)}$$

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

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$

- Some notation :-

(x, y) $\quad x \in R^{n_x}$, $y \in \{0, 1\}$

$\downarrow$

n_x dimensional feature vector

m training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})\}$

$m = m_{train}$, $m = m_{test}$

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

$\leftarrow \quad m \quad \rightarrow$

$X \in R^{n_x \times m}$ $\qquad$ X.Shape = (n_x, m)

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

$Y \in {}^{1}R^{\times m}$

Y.Shape = $(1, m)$

* Logistic Regression cost function

$\hat{y}^{(i)} = a(w^T x^{(i)} + b)$

Date: __/__/____

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$.

Want $\hat{y}^{(i)} \approx y^{(i)}$ → true label.
↓
our output

- Loss (error) function :- (for a single training eg.)

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

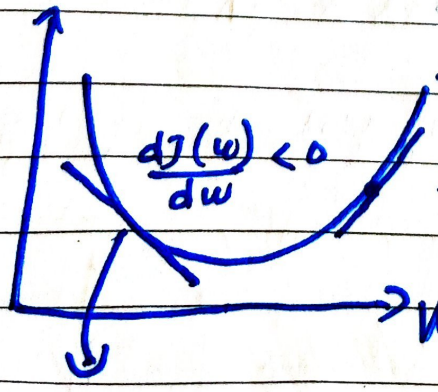
If $y=1 \Rightarrow L(\hat{y}, y) = -\log \hat{y}$

If $y=0 \Rightarrow L(y, y) = -\log(1-y)$

- Cost function : $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$
↓
used for entire training set.

$$= \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$

- Gradient descent



Here w is added

α = learning rate.

$$W := W - \alpha \frac{dJ(w)}{dw}$$

→ w is subtracted

Date: __/__/____

- Logistic regression derivatives. ⟶

$$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow L(a, y)$$

$$L = -(y \log a + (1-y) \log(1-a))$$

$$da = \frac{dL}{da}(a, y) = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$$

$$= \frac{dL}{da} a(1-a)$$

$$dz = \frac{dL}{dz} = \left[\frac{-y}{a} + \frac{1-y}{1-a}\right] a(1-a)$$

$$= [-y + ay + a - ay]$$

$$dz = \frac{dL}{dz} = a - y$$

$a = \sigma(z)$

$$\frac{da}{dz} = \sigma'(z)$$

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

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

Date: __/__/____

→ * Logistic regression on m examples.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)})$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}_{dw_1^{(i)}}$$

* $J=0;\ dw_1=0;\ dw_2=0;\ db=0$

For $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

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

$J\!+\!= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ ↓ $n=2$ (features)

$db \mathrel{+}= dz^{(i)}$

$J /= m$

$dw_1 /= m$ $\quad dw_1 = \frac{\partial J}{\partial w_1}$

$db /= m$

$dw_2 /= m$

Date: __/__/____

- Vectorization [To avoid for loops.]
  - Helps to save in deep computation, in very large data sets.

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

Non-vectorized	Vectorized
Z = 0 For i in range(n_x): Z += w^T[i] * x[i]	Z = np.dot(w, x) + b

Z += b

* Eg :-

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

If you want to calculate exponential of each element of v

$$v = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

- non-vectorized ⇒ v = np.zeros((n,1))
  For i in range(n):
  v[i] = math.exp(v[i])
- vectorized

```
import numpy as np
v = np.exp(v)
```

Date: __/__/____

$J=0$, $dw_1=0$, $dw_2=0$, $db=0$

$dw = np.zeros((n_x, 1))$

for $i=1$ to m:

$z^{(i)} = w^T x^{(i)} + b$

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

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)}[1-a^{(i)}]$

$dw_1 += x_1^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$

$n_x = 2$

$dw += x^{(i)} dz^{(i)}$

$db += dz^{(i)}$

$J = J/m$, $dw_1 = dw_1/m$, dw_2/m, $db = db/m$

$dw /= m$

- **vectorizing logistic regression**

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

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad a^{(3)} = \sigma(z^{(3)})$

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

$$[z^{(1)}\ z^{(2)}\ z^{(3)} \dots z^{(m)}] = w^T \cdot x + [b\ b \dots b]_{1\times m}$$

$$= \begin{bmatrix} w^{(T)} \text{ (row vector)} \end{bmatrix} \begin{bmatrix} x^{(1)} & x^{(2)} \dots & x^{(m)} \end{bmatrix} + [b\ b \dots]$$

$$Z = [z^{(1)}\ z^{(2)}\ z^{(3)} \dots z^{(m)}] = [w^T x^{(1)} + b \quad w^T x^{(2)} + b \dots \quad w^T x^{(m)} + b]$$

- $Z = np.dot(w.T, x) + b$
  (broadcasting in python)

~~$A = [a]$~~

$$A = [a^{(1)}\ a^{(2)}\ a^{(3)} \dots a^{(m)}] = \sigma(z)$$

* Vectorizing gradient computation.

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

$$dz = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \dots a^{(m)} - y^{(m)}]$$

~~$dw = 0$~~

Date: __/__/____

$dw = 0$

we got rid of this for loop

$dw += x^{(1)} dz^{(1)}$
$dw += x^{(2)} dz^{(2)}$
$\vdots$
$dw /= m$

$db = 0$
$db += dz^{(1)}$
$db += dz^{(2)}$
$\vdots$
$db += dz^{(m)}$
$db /= m$

For this,

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$db = \frac{1}{m}$ np.sum(dz)

$$dw = \frac{1}{m} X dz^T$$

$$= \frac{1}{m} \begin{bmatrix} x^{(1)}_1 & \cdots & x^{(m)}_1 \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + \cdots x^{m} dz^{m} \right]$$

Date: __/__/____

Modified logistic regression

For itera^n in range (1000):

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

$$z = np.dot(w.T, x) + b$$

$$A = a(z)$$

$$dz = A - y$$

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

$$db = \frac{1}{m} np.sum(dz)$$

$$w = w - \alpha dw$$

$$b = b - \alpha db$$

Broadcasting example

Calories from carbs, proteins, Fats in 100g of different foods:-

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A_{(3,4)}$

Suppose you want to calculate % of calories from Carb, [ ]

Date: __/__/____

→ means it will SMALL ↓

① cal = A.sum(axis=0) ; i.e "row vector [ ]

② percentage = 100*A/(cal) ; if (axis=1) it will sum → ie [ ] vector

* Broadcasting example.

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} \leftarrow 100 = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

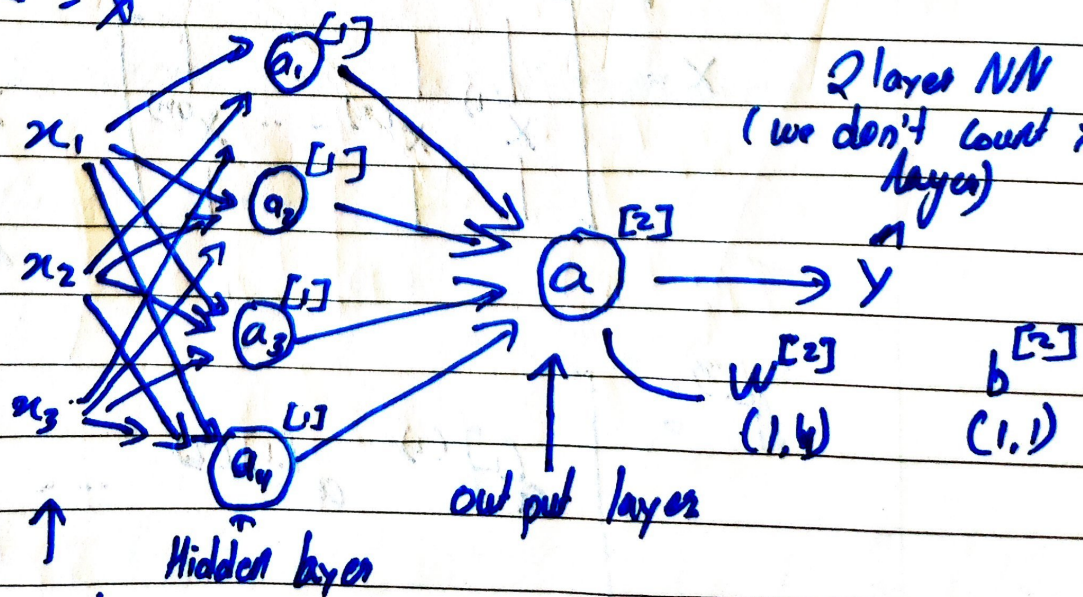
$$\begin{bmatrix}1 & 2 & 5\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix}$$

* Neural network representation

$a^{[0]} = X$



Date: __/__/____

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w^{[1]T} x + b_2^{[1]}$$
$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$a_i^{[l]}$ ← layer
↳ node in layer

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

* Vectorizing across multiple examples.

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

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

training examples

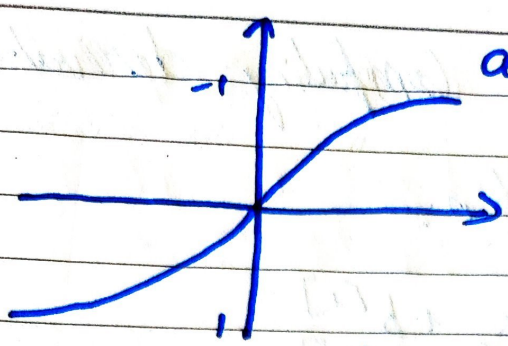
larger no.

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

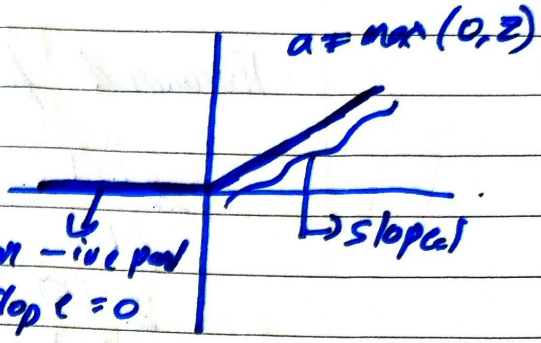
hidden units

← Training examples →

Date: __/__/____

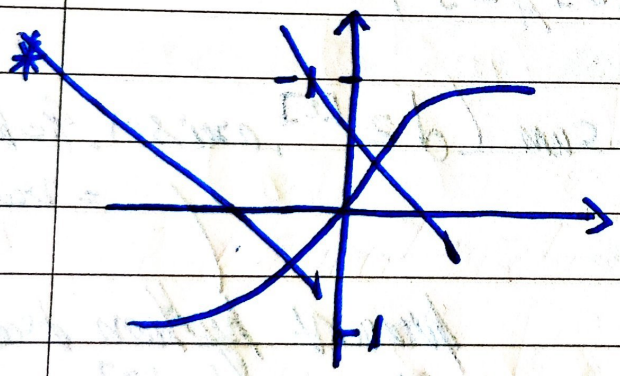
* 

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



Mostly used RELU

- Sigmoid f^n isn't a good choice in maximum cond's
- RELU is used in most

* 

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

* Gradient descent for neural networks

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

$w^{[1]}$: $(n^{[1]}, n^{[0]})$; $b^{[1]}$: $(n^{[1]}, 1)$; $w^{[2]}$: $(n^{[2]}, n^{[1]})$; $b^{[2]}$: $(n^{[2]}, 1)$

cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$

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

Formulas for computing derivatives

Forward propagation:

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

Back propagation:-

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

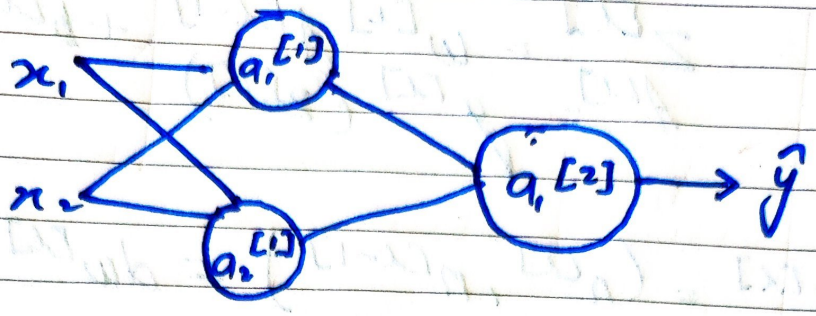
(prevent python from making $(n^{[2]},)$)

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

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

Date: __/__/____

* Random initialization



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

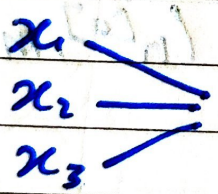
$b^{[1]} = np.zero((2,1))$

Similarly,

$W^{[2]} = \ldots\ldots$

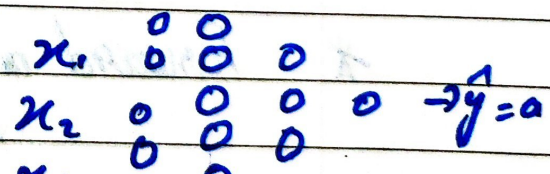
$b^{[2]} = \ldots\ldots$

* What is a deep neural network?



"Shallow"

"1 layer NN"



$L = 4$

$n^{(l)}$ = no. of units or nodes in layer l

$n^{[1]} = 5, \; n^{[2]} = 5$

$n^{[3]} = 3, \; n^{[4]} = n^{[2]} = 1$

$n^{[0]} = n_x = 3$

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

$W^{[l]}$ = weight for $z^{[l]}$

Date: __/__/____

* Forward propagation in a deep network.

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

* $W^{[l]} = (n^{[l]}, n^{[l-1]}) = dW^{[l]}$

$$b^{[l]} = (n^{[l]}, 1) = db^{[l]}$$

* Vectorized implementation.

$$Z^{[l]}, A^{[l]} = (n^{[l]}, m) = dZ^{[l]}, dA^{[l]}$$

when $l = 0$

$$A^{[0]} = X = (n^{[0]}, m)$$

* Forward and backward functions.

layer l.