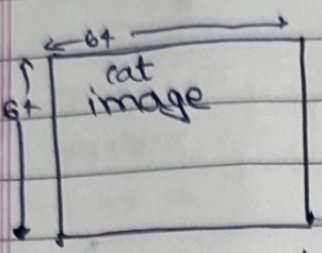
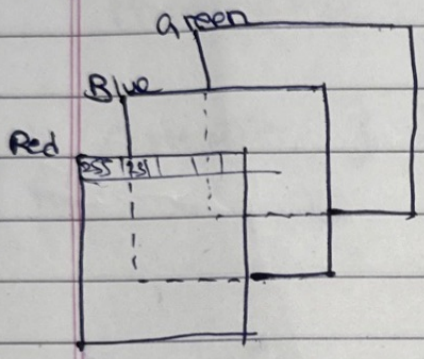
5-8-2024 WEEK-2

BINARY CLASSIFICATION :-



→ 1(Cat) vs 0 (non cat)
y → output

64 by 64 pixel



$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \end{bmatrix}$ 64×64×3 → 12288

x = 12288

matrices of 64 by 64 pixel

Notation :-

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

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

M = Mtrain Mtest = # test examples

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

much simpler

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

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

~~$Y \in R$~~

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

Y.shape = $(1, m)$

$w \Rightarrow n_x$ dimensional vector
$b \Rightarrow$ a real number

Date .....
Page .....

LOGISTIC REGRESSION :-

Given x, want $\hat{y} = P(y=1|x)$

$0 \leq \hat{y} \leq 1$

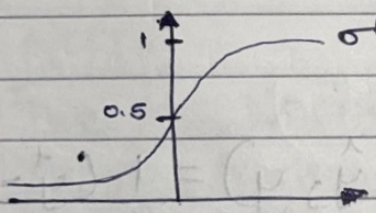
input feature vector x

$x \in R^{n_x}$, $b \in R$

Parameters: $w \in R^{n_x}$, $b \in R$

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

sigmoid function



$z = w^T x + b$

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

if z large $\sigma(z) \approx \frac{1}{1+0} = 1$

if z small $\sigma(z) \approx \frac{1}{1+\text{big}} = \frac{1}{\text{big}} \approx 0$

$x_0 = 1$, $x \in R^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \\ \theta_{n_x} \end{bmatrix}$ $\left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\}$ b ← , $\left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\}$ w ←

LOGISTIC REGRESSION COST FUNCTION

→ $\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

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

$x^{(i)}$
$y^{(i)}$
$z^{(i)}$ } i-th example

Loss (error) function: $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

Loss function L is a function we will need to define to measure how good our output $\hat{y}$ is when the true label is y

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

if $y = 1$ then

$L(\hat{y}, y) = -\log \hat{y} + 0$
$= -\log \hat{y}$

want $\log \hat{y}$ large, want $\hat{y}$ large

if $y=0$:

$L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large

want $\hat{y}$ small

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

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

NOTE:-

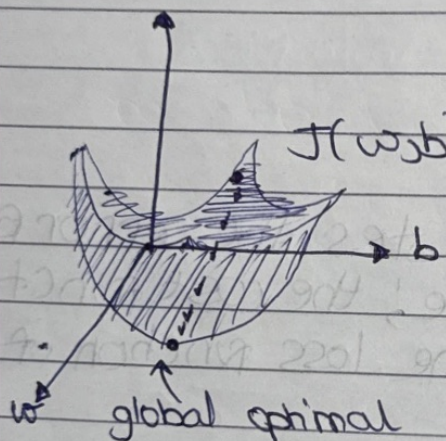
The loss function computes the error for a single training example; the cost function is the average of the loss function of the entire training set.

GRADIENT DESCENT:-

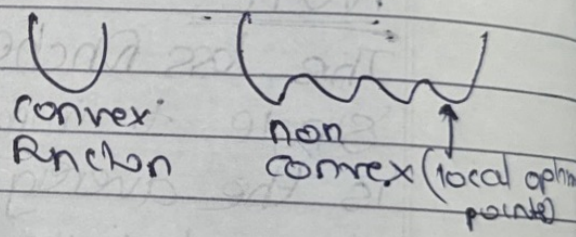
Recap: $\hat{y} = \sigma(wx + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

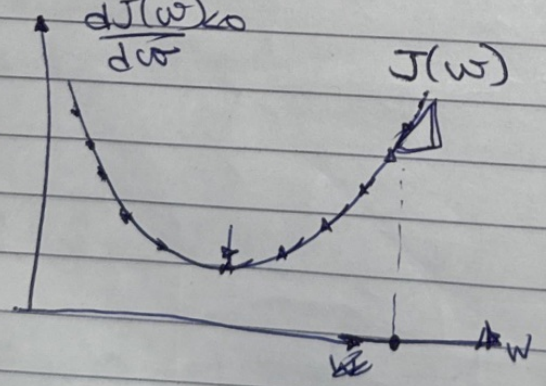
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

want to find w, b that minimize $J(w,b)$



convex function





Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$ ← learning rate

}

$dw \Rightarrow$ derivative term

Date
Page

$J(w,b)$ $\quad w := w - \alpha \frac{dJ(w,b)}{dw}$

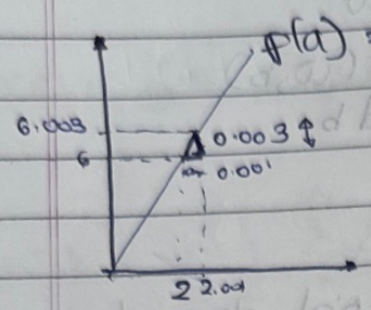
$b := b - \alpha \frac{dJ(w,b)}{db}$

$\frac{\partial J(w,b)}{\partial w_K}$

$\partial \rightarrow$ partial derivative

$d \rightarrow$ only 1

DERIVATIVES



$a = 2 \quad f(a) = 6$

$a = 2.001 \quad f(a) = 6.003$

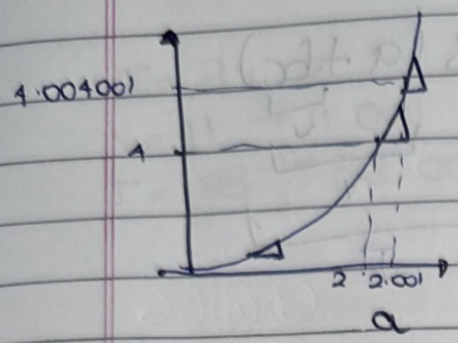
$$\text{slope} = \frac{\text{height}}{\text{width}} = \frac{0.003}{0.001} = 3$$

$a = 5 \quad f(a) = 15$

$a = 5.001 \quad f(a) = 15.003$

slope = 3

MORE DERIVATIVE EXAMPLES



$f(a) = a^2$

different slope

$\therefore$ slope $= f'(a) = 2a$

infinitesimal → very very small

$f(a) = a^3$

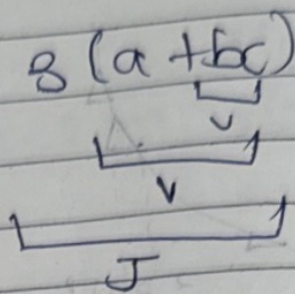
$f'(a) = 3a^2$

$f(a) = \log(a)$

$f'(a) = 1/a$

COMPUTATION GRAPH :-

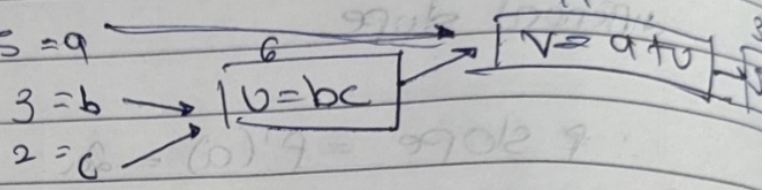
$J(a,b,c) = 3(a+bc)$



$u = bc$

$v = a + u$

$J = 3v$



one-step of backward propogation on a computation graph yields derivative of final output variable

DERIVATIVES WITH COMPUTATION GRAPH :-

$J = 3v$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$f(a) = 3a$

$\frac{df(a)}{da} = \frac{df}{da} = 3$

$J = 3v$

$\frac{dJ}{dv} = 3$

$a = 5 \rightarrow 5.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{da}$

chain rule

$\frac{dJ}{dv} = 3 \quad \frac{dv}{da} = 1$

d. Final output variable

d. variable

$\frac{dJ}{db} = \frac{dJ}{dv} \cdot \frac{dv}{db}$

$= 3 \times 2$

$= 6$

$b = 3 \rightarrow 3.001 \quad c = 2$

$v = bc = 6 \rightarrow 6.002$

$\frac{dv}{db} = 2$

$\frac{dJ}{dv} = 3$

$\frac{dJ}{dc} = \frac{dJ}{dv} \cdot \frac{dv}{dc} = 9$

3×3

LOGISTIC REGRESSION GRADIENT DESCENT

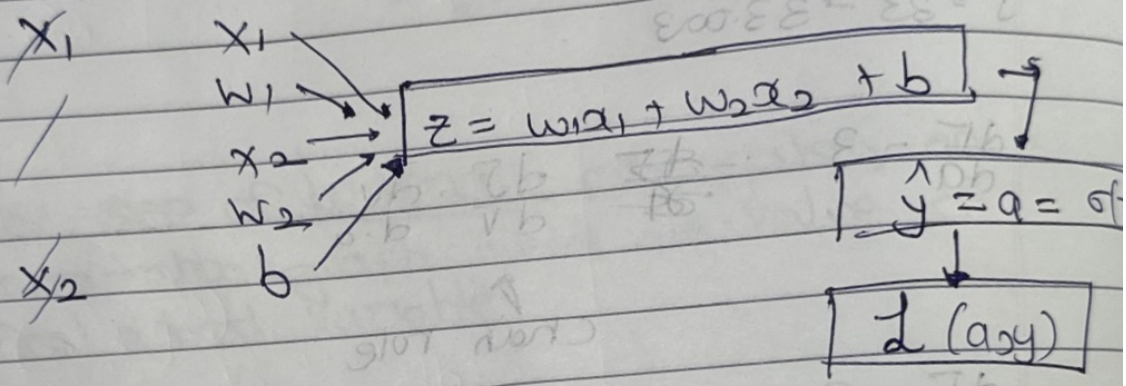
$\rightarrow z = w^T x + b$

$\hat{y} = a = \sigma(z)$

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

a = output of logistic regression

y = grand truth table



$$\text{"da"} = \frac{d\mathcal{L}(a, y)}{da}$$

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

$$dz = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a, y)}{dz}$$

$$= a - y \qquad a(1-a)$$

$$= \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$$

$$\frac{dL}{dw_1} = "dw_1" = x_1 dz$$

$$dw_2 = x_2 dz$$

$$db = dz$$

$$w_1 := w_1 - \alpha dw_1$$
$$w_2 := w_2 - \alpha dw_2$$
$$b := b - \alpha db$$

GRADIENT DESCENT ON M EXAMPLES:-

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

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

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

$$dw_1^{(i)} - (x^{(i)}, y^{(i)})$$

$(x^{(i)}, y^{(i)})$
$dw_1, dw_2, db^{(i)}$

$J=0, dw_1=0, dw_2=0, db=0$
for $i=1$ to m
$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$J += y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})$

$dw_1 += x_1^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$
$db += dz^{(i)}$
$n=2$

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

$$dw_1 = \frac{\partial J}{\partial w_1}$$

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

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

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

VECTORISATION:-

Q] what is vectorization?

$z = w^T x + b$ $\quad w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $\quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$

$w \in R^{n_x}$

$x \in R^{n_x}$

Non-vectorized	vectorized
z = 0 for i in region(n-x): z += w[i] * x[i] z += b	z = np.dot(w, x) + b $w^T x$

GPU & SIMD
CPU single instruction multiple data

MORE VECTORISATION EXAMPLES:-

Whenever possible, avoid explicit for-loops

$u = Av$

$u_i = \sum_j A_{ij} v_j$ ← NON-VECTORIZED

```
u = np.zeros((n,1))
    for i....
      for j....
        u[i] += A[i][j] * v[j]
```

```
u = np.dot(A, v)
```

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

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

np.log(v)
np.abs(v)
np.maximum(v, 0)

v**2    1/v
```

```
→ u = np.zeros((n,1))
→ for i in range(n):
→     u[i] = math.exp(v[i])
```

VECTORISING LOGISTIC REGRESSION:-

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

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

$z^{(3)} = w^T x^{(3)} + b$

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

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

(n_x, m)

$n_x \times m$

R

$$\begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix} = w^T X + \begin{bmatrix} b & b & \cdots & b \end{bmatrix}$$

$1 \times m$

$$= \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots \end{bmatrix}$$

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

$(1,1)$ R

Broadcasting

VECTORIZING LOGISTIC REGRESSION'S GRADIENT OUTPUT:-

$dz^{(1)} = a^{(1)} - y^{(1)}$

$dz^{(2)} = a^{(2)} - y^{(2)}$

$dz = [dz^{(1)} \; dz^{(2)} \; dz^{(3)} \; \dots \; dz^{(m)}]$

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

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

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

$dw = 0$	$db = 0$
$dw += x^{(1)} dz^{(1)}$	$db += dz^{(1)}$
$dw += x^{(2)} dz^{(2)}$	$db += dz^{(2)}$
⋮	⋮
	$db += dz^{(m)}$
$dw /= m$	$db /= m$

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

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

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

$$= \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

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

$n \times 1$

Implementing Logistic Regression

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

For $i=1$ to m:

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

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

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

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

$\begin{bmatrix} dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)} \\ dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)} \end{bmatrix}$ $dw \mathrel{+}= x^{(i)} * dz^{(i)}$

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

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

$db = db/m$

for iteration in range (1000):

$$z = w^T X + b$$
$$= np.dot(w.T, X) + b$$
$$A = \sigma(z)$$
$$dz = A - Y$$
$$dw = \frac{1}{m} X\, dz^T$$
$$db = \frac{1}{m} np.sum(dz)$$

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

~~BROADCASTING~~ BROADCASTING IN PYTHON:-

Calories from carbs, protens, fats in 100g of different foods:

	Apples	Beef	Egg	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
	59cal			

$\frac{56}{59} \times 100 \Rightarrow 94.4\%$

Calculate % of calories for carbs, Protein, fat
Can you do this without an explicit for loop?

```
cal = A.sum(axis=0)
percentage = 100*A/(cal.reshape(1,4))
```

Broadcasting example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \quad 100 = \begin{bmatrix} 14 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

(2) $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$

(3) $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 \\ 200 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$

GENERAL PRINCIPLE :-

$(m,n) \begin{matrix} + \\ - \\ * \\ / \end{matrix} (1,n) \rightsquigarrow (m,n)$

$(m,1) \rightsquigarrow (m,n)$

$(m,1) + R$

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

$[1\ 2\ 3] + 100 = [101\ \ 102\ \ 103]$

MATLAB / OCTAVE → bsx function

A NOTE ON PYTHON NUMPY VECTORS:

rank 1 array

```
a = np.random.randn(5)
a.shape = (5,)
"rank 1 array"
```

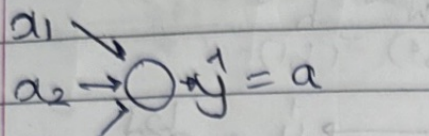
} Don't use

```
a = np.random.randn(5,1) → a.shape = (5,1)
a = np.random.randn(1,5) → a.shape = (1,5)
assert(a.shape == (5,1))
```

QUICK TOUR OF JUPYTER/PYTHON NOTEBOOKS

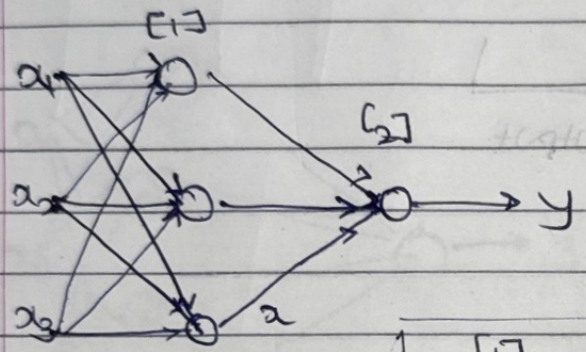
8/8/2024 WEEK-3 :- SHALLOW NUERAL NETWORK

NUERAL NETWORK OVERVIEW:-



a
w
b

$z = w^T x + b \rightarrow a = \sigma(z) = \mathcal{L}(a, y)$



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

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

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

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

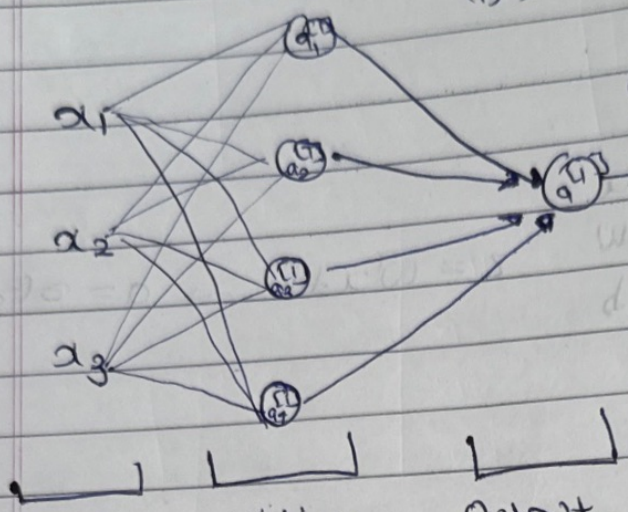
$\mathcal{L}(a^{[2]}, y)$

[1] ← layer

(i) ← ith example

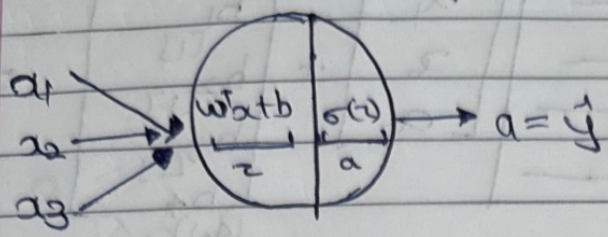
NUERAL NETWORK REPRESENTATION :-

2 layer NN



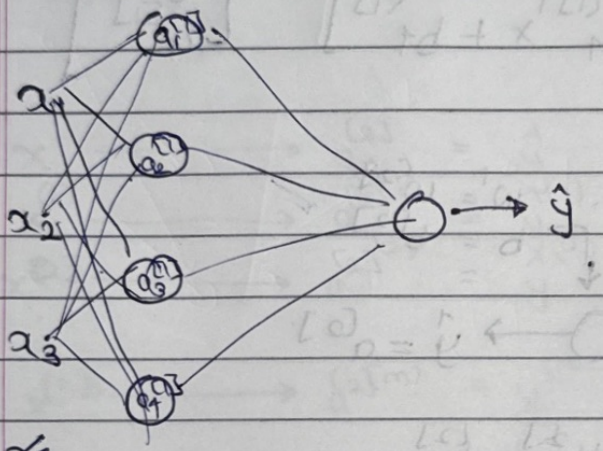
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

COMPUTING A NUERAL NETWORKS OUTPUT:-



$z = w^T x + b$

$a = \sigma(z)$



$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$

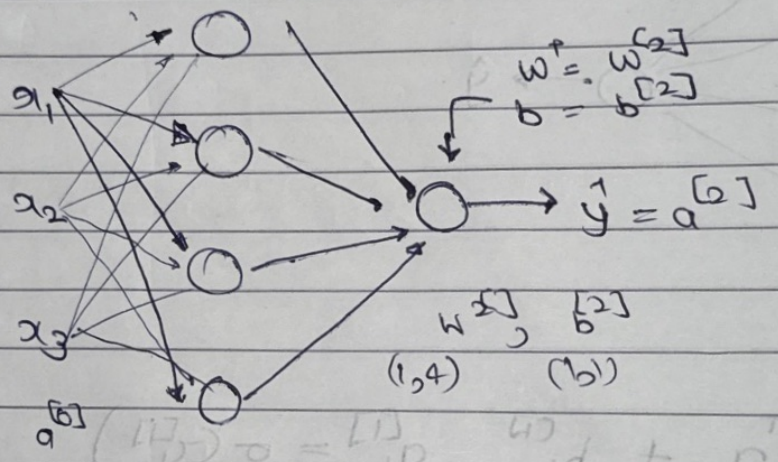
$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$

$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$

$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$

$$z^{[1]} = \begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \\ - & w_4^{[1]T} & - \end{bmatrix}_{4\times(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}_{(4,1)} =$$

$$= \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$



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

$$\hat{y} = a = \sigma(z)$$

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

(4,1) (4,3) (3,1) (4,1)

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

(4,1) (4,1)

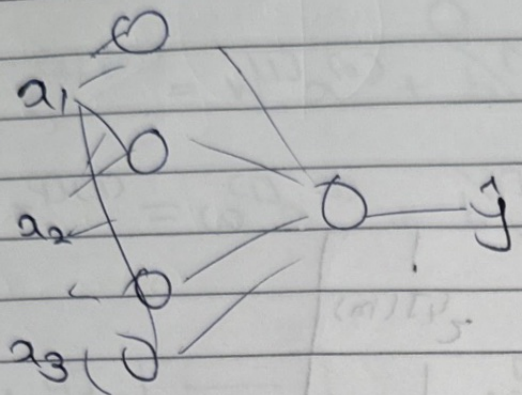
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

(1,1) (1,4) (4,1) (1,1)

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

(1,1) (1,1)

VECTORISING ACROSS MULTIPLE EXAMPLES:-



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

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

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

$\vdots$

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

for i = 1 to m

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

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

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

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

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

(n_x, m)

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

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

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

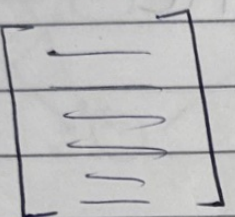
$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

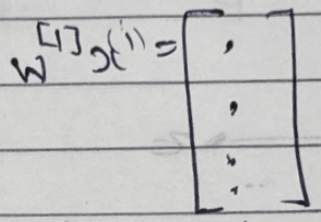
EXPLAINATION FOR VECTORISATION IMPLEMENTATION

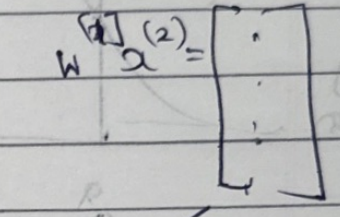
$$z^{[1](1)} = W^{[1]} x^{(1)} + b^{[1]}$$

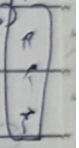
$$z^{[1](2)} = W^{[1]} x^{(2)} + b^{[1]}$$

$$z^{[1](3)} = W^{[1]} x^{(3)} + b^{[1]}$$

$W^{[1]} =$ 

$W^{[1]} x^{(1)} =$ 

$W^{[1]} x^{(2)} =$ 

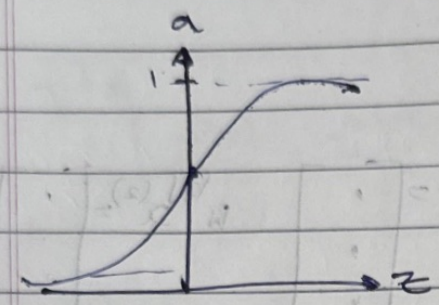
$W^{[1]} x^{(3)} =$ 

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

ACTIVATION FUNCTIONS

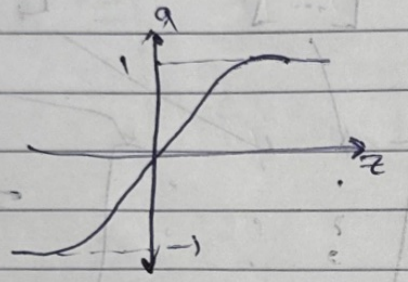
Sigmoid is called activation function

$g(z) \rightarrow$ non-sigmoid



only for binary classification

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

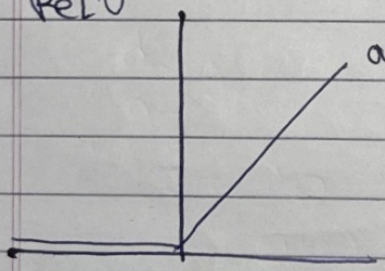


strictly superior

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

ReLU

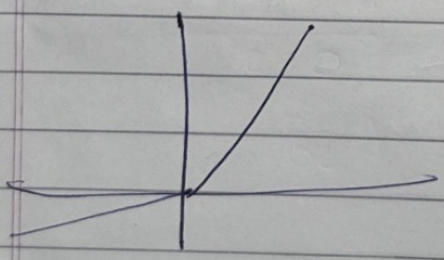
most commonly used



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

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

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

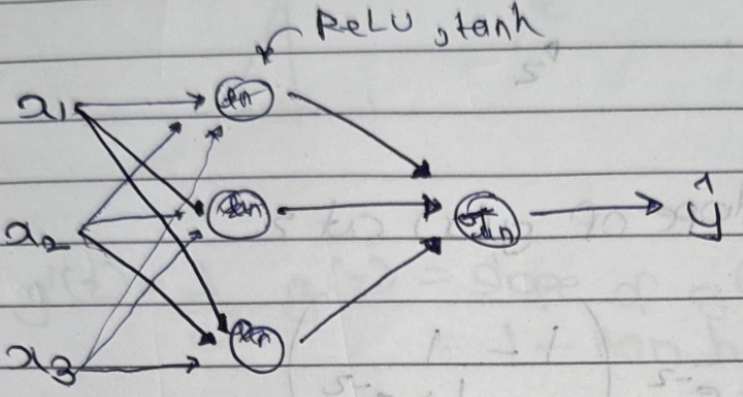


$$\max(0.01z, z)$$

NN

WHY DO YOU NEED NON-LINEAR ACTIVATION FUNCTION?

Activation function



Given x:

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

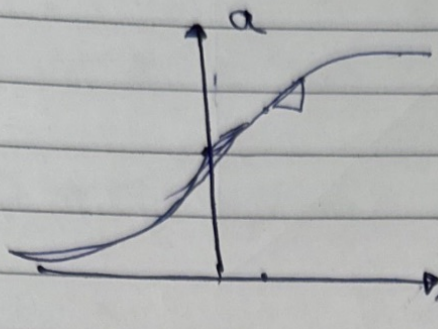
$g(z) = z$ linear activation function

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

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

$$= W'x + b'$$

DERIVATIVES OF ACTIVATION FUNCTIONS



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

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

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

$$g(z)(1-g(z))$$

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

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

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

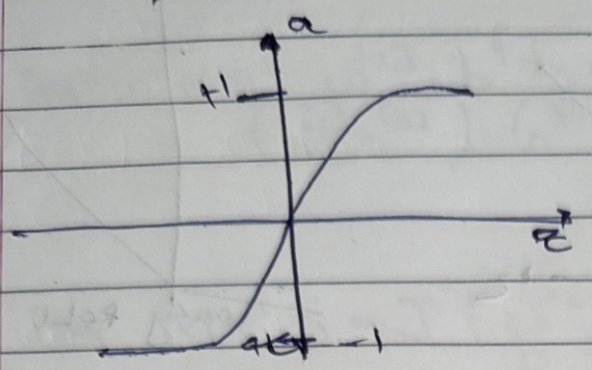
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) = 0$

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

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

Date
Page

tanh activation function



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

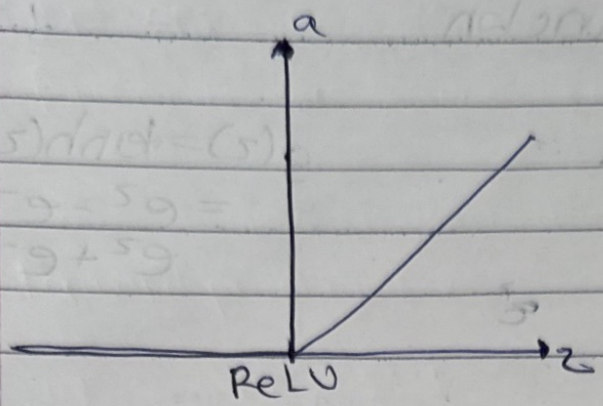
$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$
$$= 1 - (\tanh(z))^2$$

$z = 10 \quad \tanh(z) \approx 1$
$g'(z) \approx 0$

$z = -10 \quad \tanh(z) \approx -1$
$g'(z) \approx 0$

$z = 0 \quad \tanh(z) = 0$
$g'(z) = 1$

ReLU and Leaky ReLU



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

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

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

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

GRADIENT DESCENT FOR NUERAL NETWORKS:-

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$ $\quad n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

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

Cost function $= J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

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

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

Gradient descent

Repeat {

compute prediction $(\hat{y}^{(i)}, i = 1 \ldots m)$

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

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

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

formulas for computing derivatives

Forward propogation:

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

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

Back Propogation:

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

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

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

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

element product

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

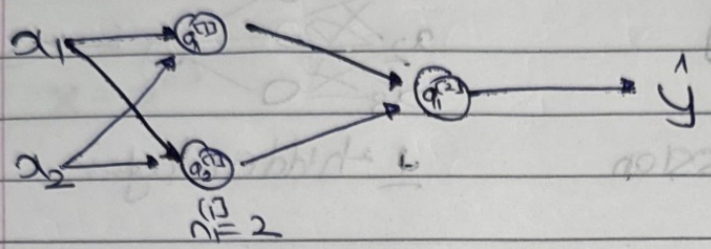
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$

Date
Page

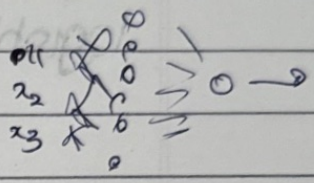
RANDOM INITIALIZATION:-

what happens if you initialize weights to zero?

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



Symmetric



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

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

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

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix} \qquad w^{[1]} = \begin{bmatrix} - & - \\ - & - \end{bmatrix}$$

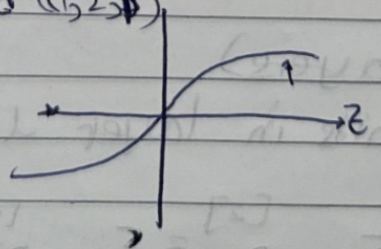
$w^{[1]} = np.random.randn((2,2)) \times 0.01$ (initialization)

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

$w^{[2]} = np.random.rand((1,2))$

$b^{[2]} = 0$

We like initializing with small values



0.01 constant works for shallow NN

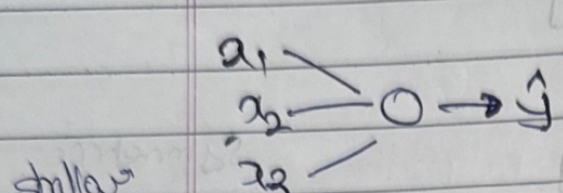
$$w^{[1]} = w^{[1]} - \alpha dw^{[1]}$$

$$b^{[1]} = b^{[1]} - \alpha db^{[1]}$$

8/8/2024 WEEK-4 :- DEEP NUERAL NETWORK

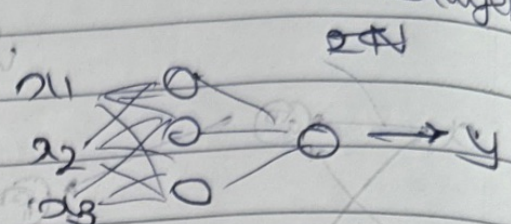
DEEP L-LAYER NUERAL NETWORK

what is deep nueral network?

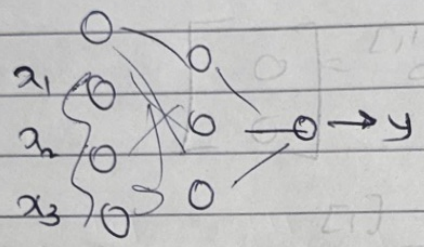


shallow

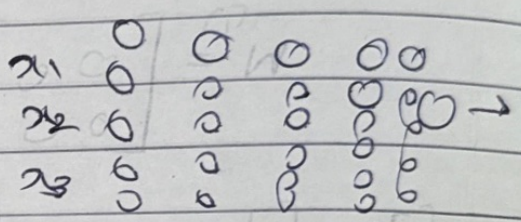
logistic regression



1 hidden layer

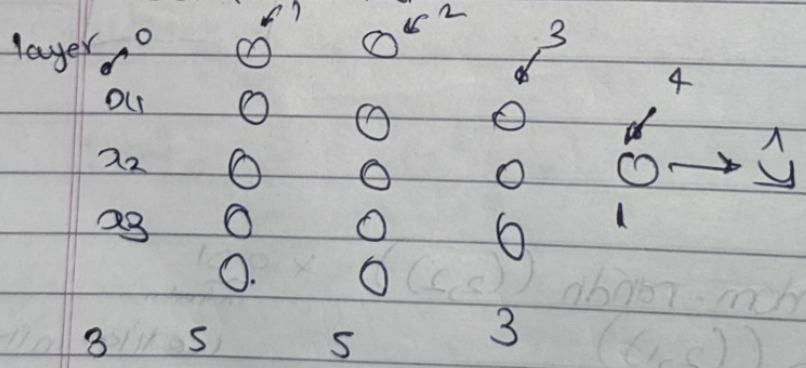


2 hidden layers



5 hidden layer (deep)

Deep NN notation



L = 4

4 layer NN

L = 4 (layers)

$n^{[l]}$ = units in layer l

$n^{[1]} = 5$ $n^{[2]} = 5$ $n^{[3]} = 3$ $n^{[4]} = 1 = n^{[L]}$

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

Date
Page

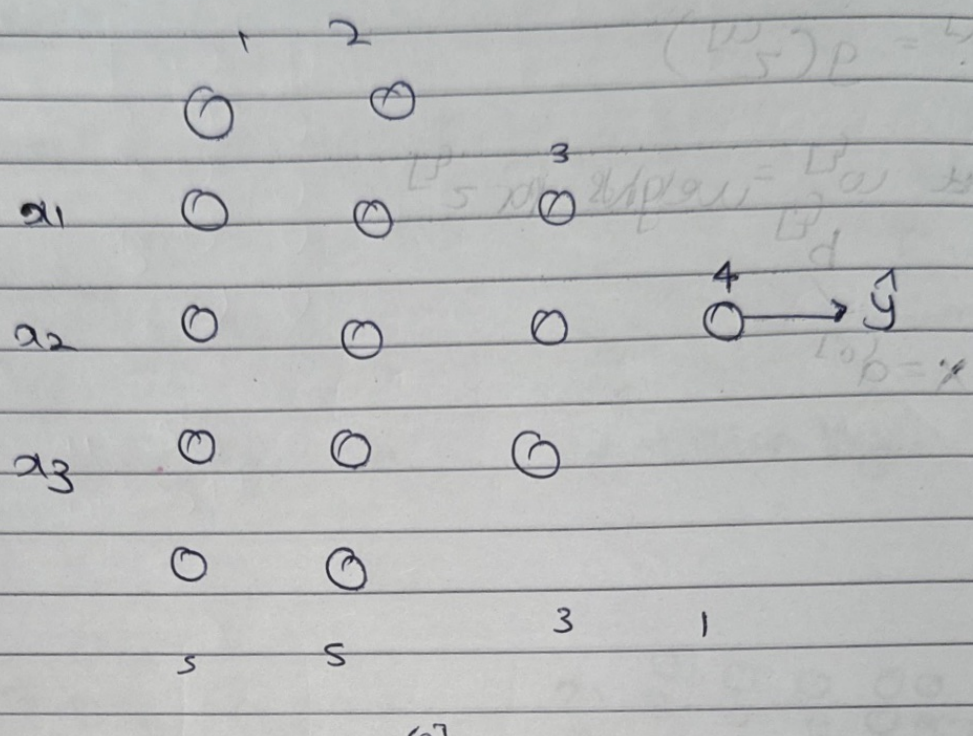
$a^{[l]}$ = activations in layer l

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

$w^{[l]}$, $b^{[l]}$ = weights for $z^{[l]}$

$x = a^{[0]}$

FORWARD PROPOGATION IN A DEEP NETWORK



$a^{[0]}$

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

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

$z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$
$a^{[3]} = g^{[3]}(z^{[3]}) = \hat{y}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$
$a^{[l]} = g^{[l]}(z^{[l]})$

Date
Page

Vectorized:

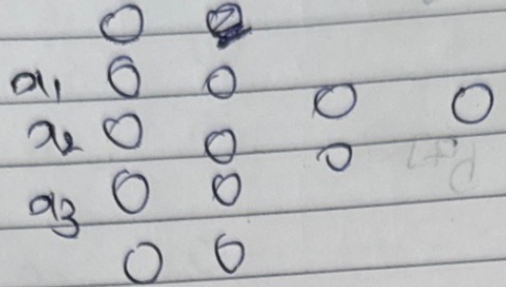
$$z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$

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

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

Date
Page

GETTING YOUR MATRIX DIMENSIONS RIGHT

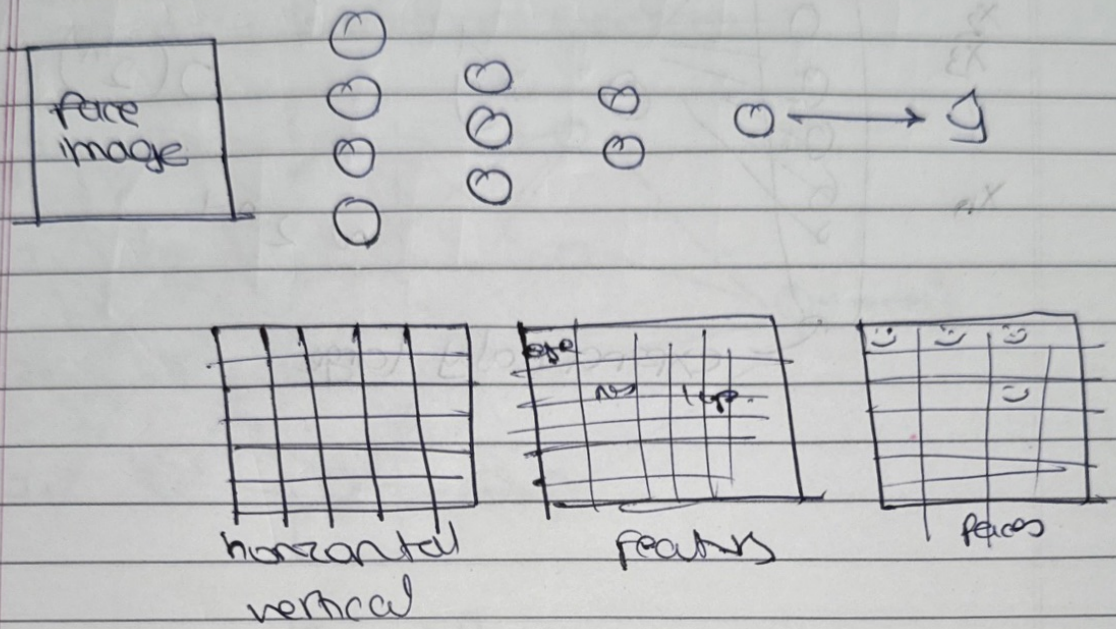


$W_1 = (5, 3)$

$b_1 = (5, 1)$

⋮

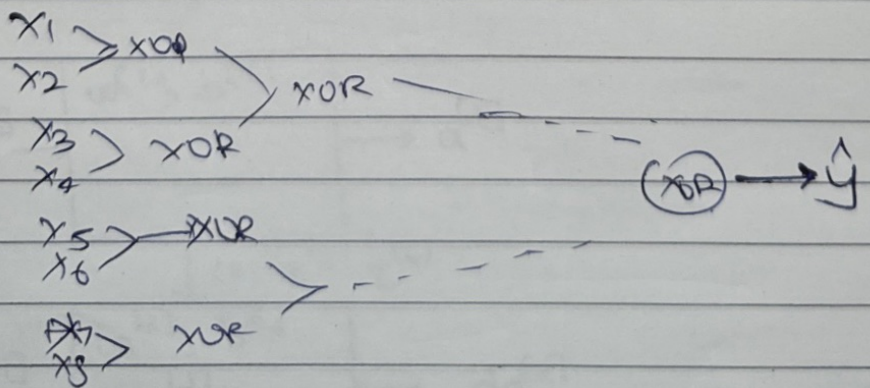
WHY DEEP REPRESENTATIONS:-

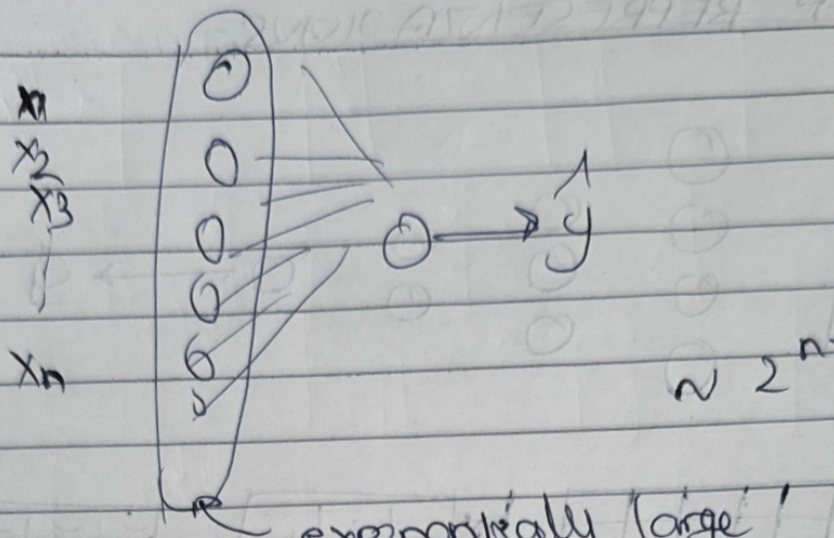


CIRCUIT THEORY & DEEP LEARNING:-

Informally; There are functions you can compute with a small L-layer deep neural network that shallow networks require exponentially more hidden units to compute

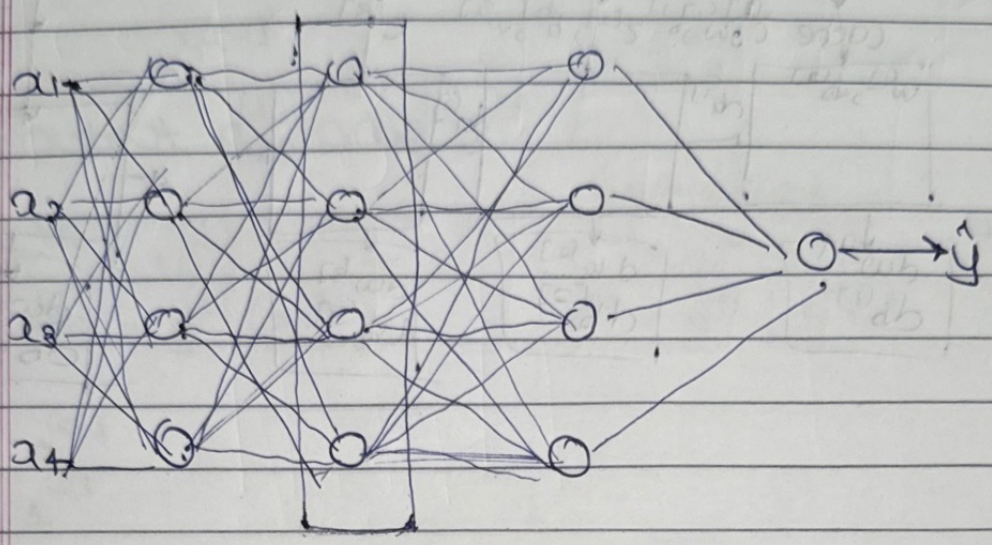
$y = x_1 \text{ XOR } x_2 \text{ XOR } x_3 \text{ ... XOR } x_n$




x_1
x_2
x_3
x_n
$\hat{y}$
$O(2^n)$
$\sim 2^{n-1}$
exponentially large!!

BUILDING BLOCKS OF DEEP NUERAL NETWORKS

forward & backward functions



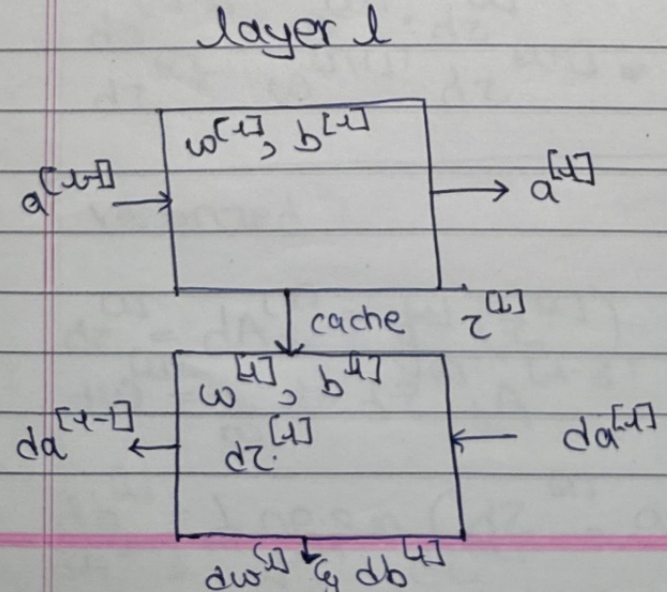
layer l: $w^{[l]}$, $b^{[l]}$

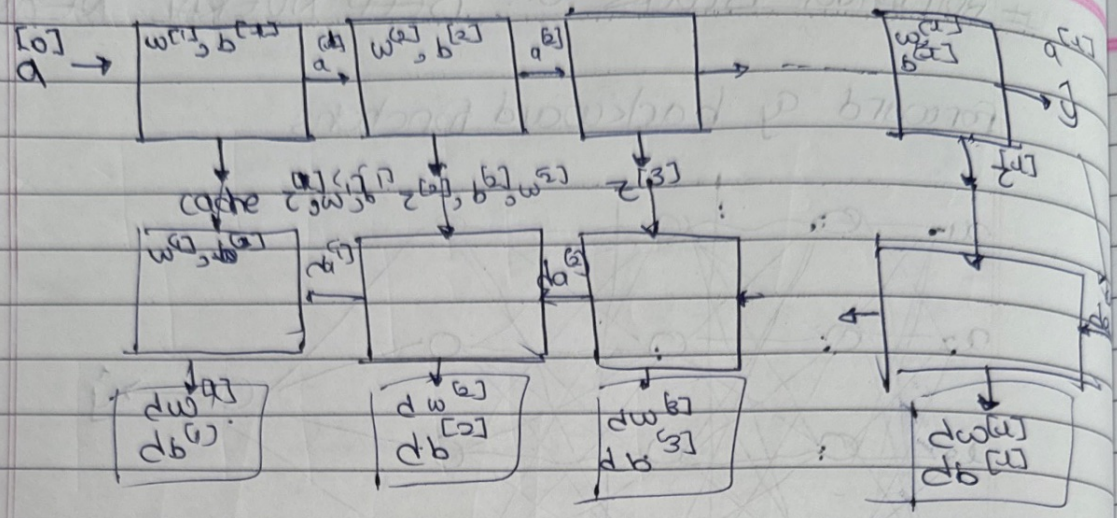
→ forward: Input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

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

→ Backward: Input $da^{[l]}$ cache($z^{[l]}$), output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$





FORWARD AND BACKWARD PROPOGATION :-

forward propogation for layer l

→ input $a^{[l-1]}$ ← $w^{[l]}, b^{[l]}$

→ output $a^{[l]}$, cache $(z^{[l]})$

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

Vectorized

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

$a^{[0]}$
$A^{[0]}$

$x = A^{0} \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow$

Backward propogation for layer l

→ Input $da^{[l]}$

→ Output $da^{[l-1]}$, ~~dw~~ $dW^{[l]}$, $db^{[l]}$

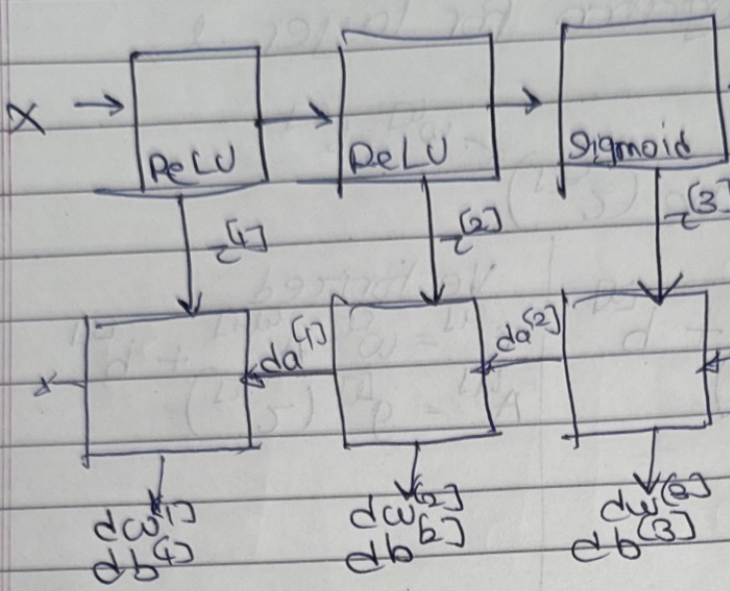
$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$
$$dz^{[l]} = w^{[l+1]T} \cdot dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$$

---

vectorized

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Summary



$$da^{[3]} = -\frac{y}{a} + \frac{(1-y)}{(1-a}$$

$$dA^{[1]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \right.$$

$$\left. -\frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})} \right)$$

PARAMETERS VS HYPERPARAMETERS:

Parameters:- $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \ldots\ldots$

Hyperparameters:- Learning rate α