Deep - L layer N.N

→ ~~Shaw~~ Shallow Vs Deep is a matter of depth ie no of hidden layers

→ Logistic reg. is a 1 layer N.N

→

Forward propagation in Deep Network

For a single training eg :-
for 1st hidden layer
→ $z^{[1]} = W^{[1]}x + b^{[1]}$
→ $a^{[1]} = g^{[1]}(z^{[1]})$

for 2nd hidden layer
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = g^{[2]}(z^{[2]})$
.... and so on

→ In O/P layer
$z^{[last]} = W^{[last]}a^{[last-1]} + b^{[last]}$
$a^{[last]} = g(z^{[last]}) = \hat{y}$

General rule : $z^{[l]} = W^{(l)}a^{[l-1]} + b^{[l]}$
$a^{[l]} = g^{[l]}(z^{[l]})$

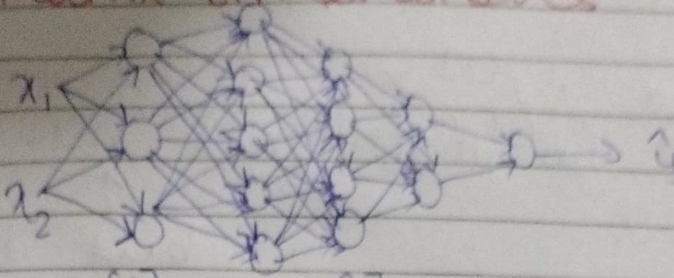
If vectorized :

for $l=1$ to last ↑ no way to eq do without for loop

$Z^{[1]} = W^{[1]}X + b^{[1]}$ $(X = A^{[0]})$
$A^{[1]} = g^{[1]}(Z^{[1]})$
$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
$A^{[2]} = g^{[2]}(Z^{[2]})$
... and so on

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

Matrix Dimensions



$n^{[0]} = 2$
$n^{[1]} = 3$
$n^{[2]} = 5$
$n^{[3]} = 4$
$n^{[4]} = 2$
$n^{[5]} = 1$

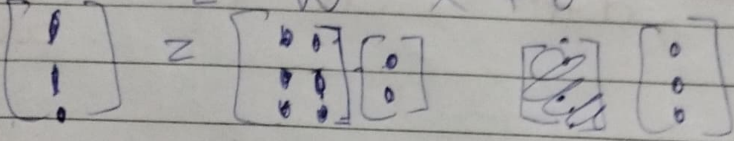
$z^{[1]} = W^{[1]} x + b$

$z^{[1]} = (3,1) = (n^{[1]}, 1)$ $\quad b = (n^{[1]}, 1) = (3,1)$

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

$W = (3,2) = (n^{[1]}, n^{[0]})$

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



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

$(5\times 1) \quad (5\times 3) \quad (3,1) \quad (5,1)$

→ $W^{[l]} = (n^{[l]}, n^{[l-1]})$

→ dw shld be same dimension as W

ie $dw^{[l]} = (n^{[l]}, n^{[l-1]})$

→ $db^{[l]} = [n^{(l)} - 1]$ — same as b

→ $a^{[l]} = g^{[l]}(z^{[l]})$

$\therefore$ z & a shld be same dimension $= (n^{[l]}, 1)$

For m examples

$$\underset{(n^{[1]},m)}{Z^{[1]}} = \underset{(n^{[1]},n^{[0]})}{W^{[1]}}\ \underset{(n^{[0]},m)}{X} + \underset{(n^{[1]},1)}{b^{[1]}}$$

but by python broadcasting it is duplicated to $(n^{[1]}, m)$

$\therefore Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$ (same as Z & A)

Why deep Networks work well

⇒ Detects simple things like edges first and then builds them into more complex network things.

☆ Circuit theory and deep learning

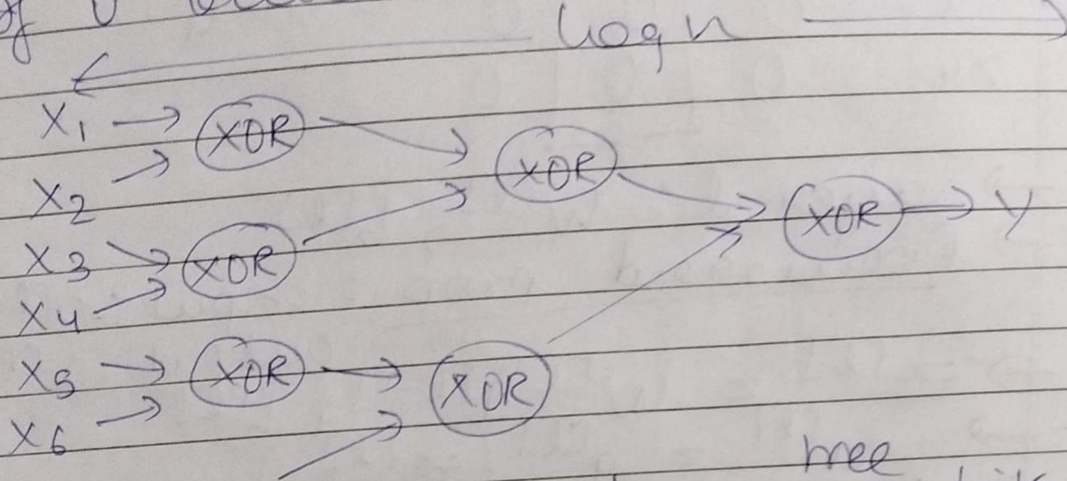
There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute

→ Circuit theory pertains the thinking about what types of functions you can compute with different

AND, OR and NOT gates.

→ If we r trying to compute x_1 XOR, x_2 XOR .... x_n XOR if we have n features

If u build XOR tree like this:

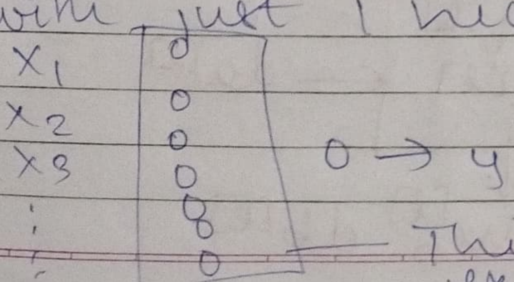


We can build a ~~ckt~~ tree like this

→ To compute XOR the depth of network will be on the order log n

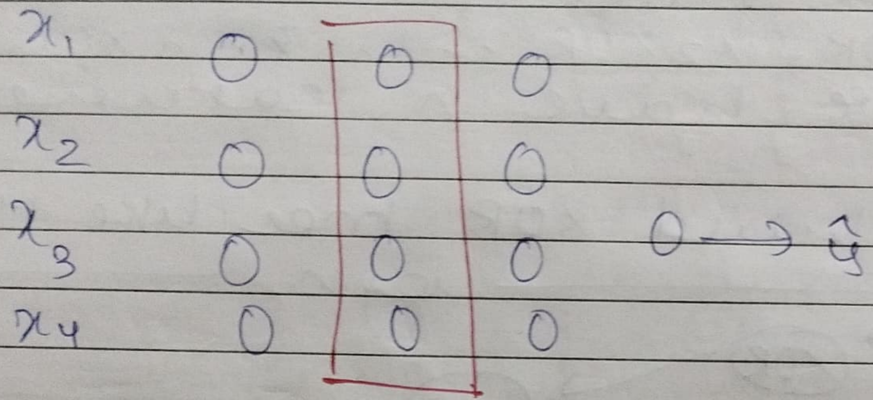
→ no of nodes/ckt components/gates in the network is not that large. We dont need that many gates in order to compute exclusive OR

→ If we are forced to compute with just 1 hidden layer



This layer will be exponentially large (2^{n-1})

Building blocks of deep neural Networks



→ Layer l : $W^{[l]}$, $b^{[l]}$

☆→ **Forward prop** : Input : $a^{[l-1]}$
Output $a^{[l]}$

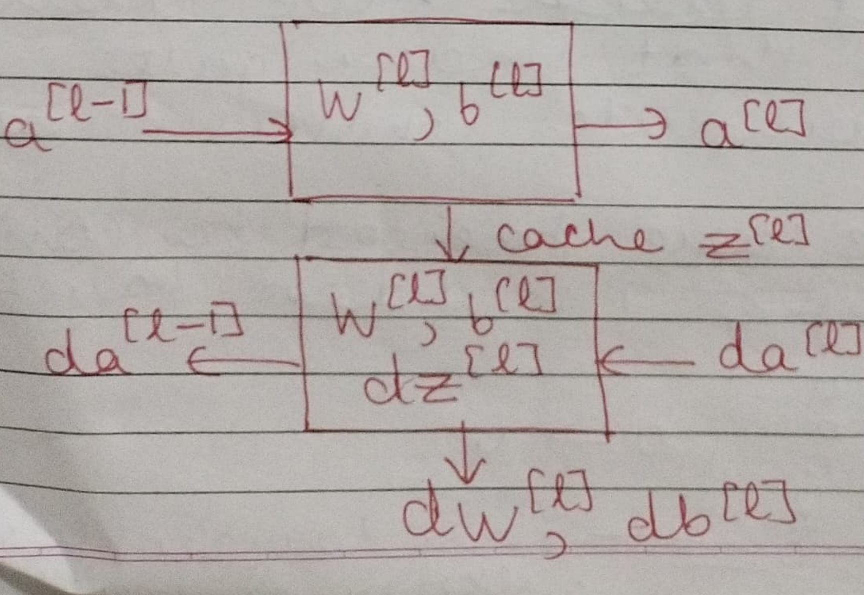
→ $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

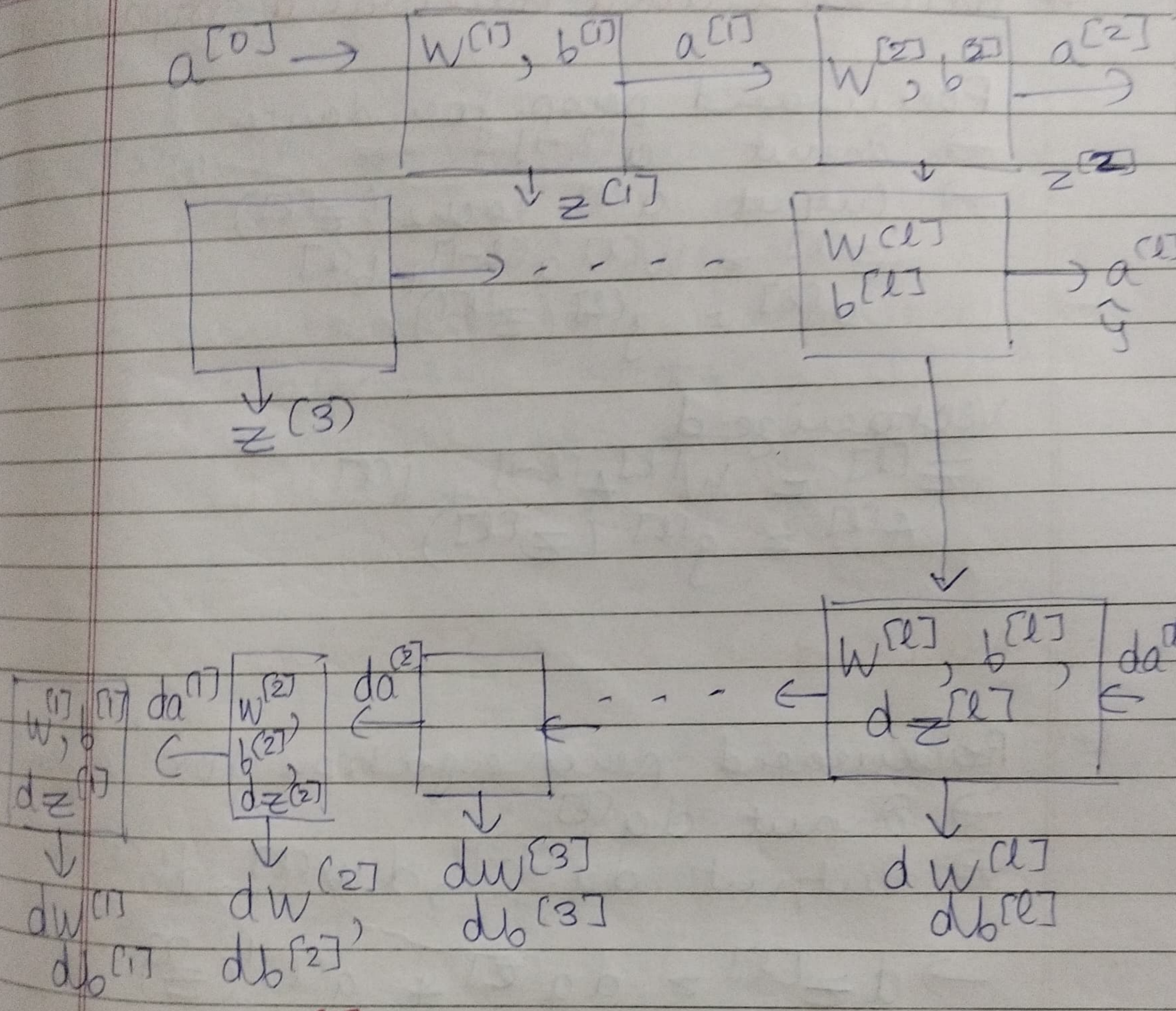
→ $a^{[l]} = g^{[l]}(z^{[l]})$

→ also cache $z^{[l]}$ as storing $z^{[l]}$ is useful for back propagation

☆→ **Backward prop** : Input $da^{[l]}$, cache
Output $da^{[l-1]}$

Layer l





$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$

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

→ It is useful to store $z^{[1]}$, $W^{[1]}$ & $b^{[1]}$ .... in cache during forward prop to be used in backward prop.

Forward and Backward Propagation

Forward prop. for layer l
→ Input $a^{[l-1]}$
→ Output $a^{[l]}$, cache ($z^{[l]}$) — $W^{[l]}, b^{[l]}$

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

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

Backward propagation for layer l
→ Input $da^{[l]}$
→ Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

→ $dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$
element wise multiplication

→ $dw^{[l]} = dz^{[l]} a^{[l-1]T}$
→ $db^{[l]} = dz^{[l]}$
→ $da^{[l-1]} = W^{[l]T} dz^{[l]}$

Vectorized:
$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]})$
$dW^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$

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

PAGE No.
DATE / /

$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

→ In ~~are~~ forward prop for layer 0 we pass in the parameters x
→ Similarly for backward prop. we pass $da^{[l]}$ which is

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

Parameters Vs Hyperparameters

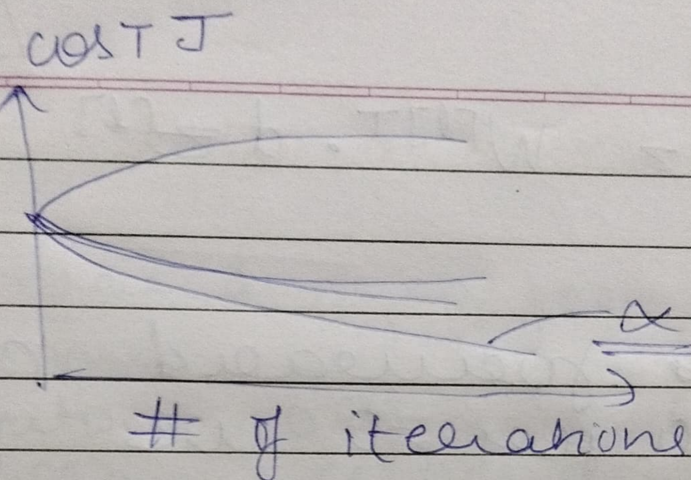
Parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]} \ldots$

Hyperparameters :
→ learning ~~dat~~ rate : α
→ iterations
→ hidden layers l
→ hidden units $n^{[1]}, n^{[2]}, \ldots$
choice of activation functions

control the parameters

These determine final value of parameters

☆ Applying deep learning is an empirical process
→ We keep trying diff values of ~~cost function~~ α and observe cost function

→ 

→ U just have to try out diff values of hyperparameters and see what works best
It is an empirical process