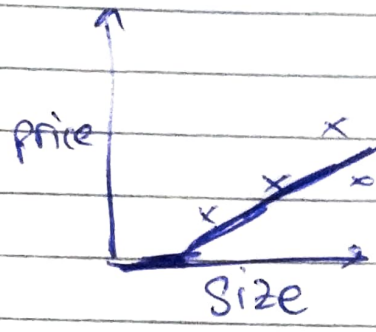
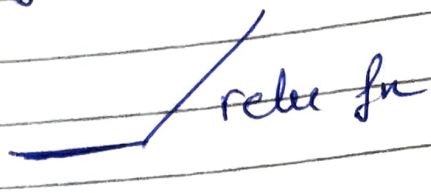
NN & DL

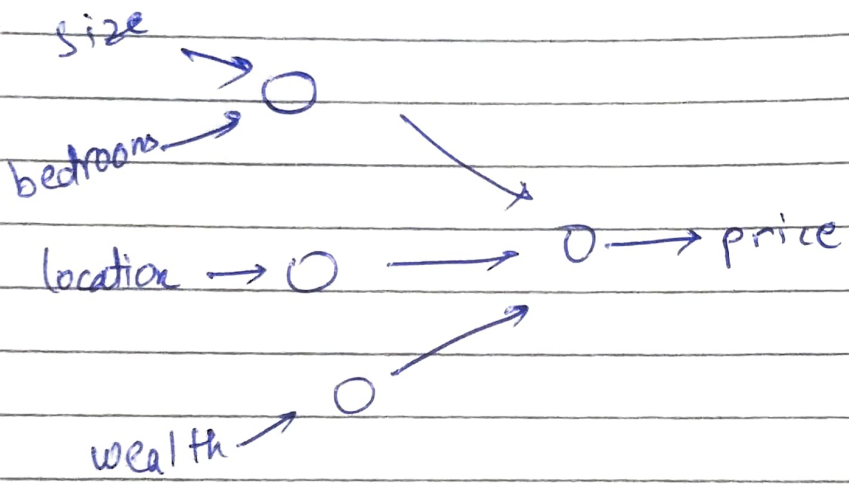
Intro to Deep Learning



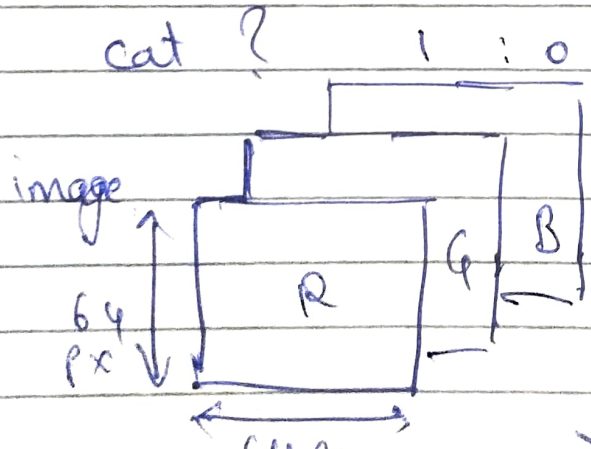


Size → price

↳ neuron



Binary Classification - Logistic Reg as NN



dimensions of input vector: 64 × 64 × 3

X → y

Notation

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

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

split

m_{train} $\quad m_{test}$

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

m

Logistic Reg

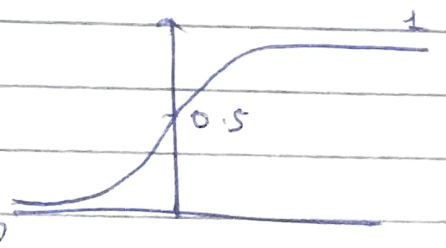
Given x, we want $\hat{y} = P(y = 1 \mid x)$

$x \in \mathbb{R}^{n_x}$

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

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

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



Alternative notation of assigning

$x_0 = 1$

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

does not work very well in case of NNs

* Cost function

we want $\hat{y}^{(i)} \approx y^{(i)}$

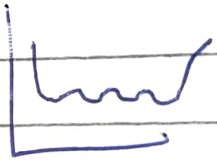
Logistic regression's cost fn -

if mse is considered

ie. $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

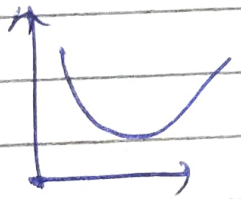
gives a curve with local minima which doesn't work well with gradient descent

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



Instead we use as Loss fn [Applied on single training ex]

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



- $y \in \{0, 1\}$
- $\hat{y} \in (0, 1)$

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

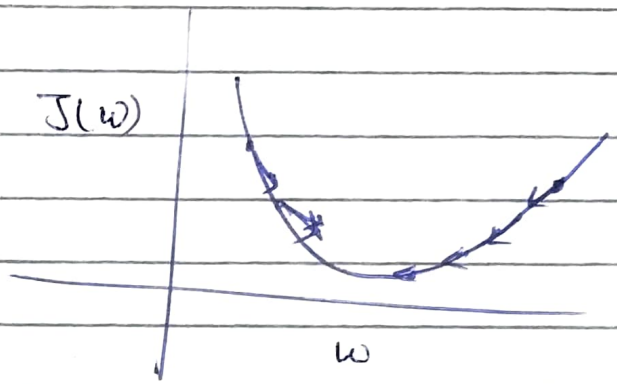
$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

* Gradient Descent

Repeat {

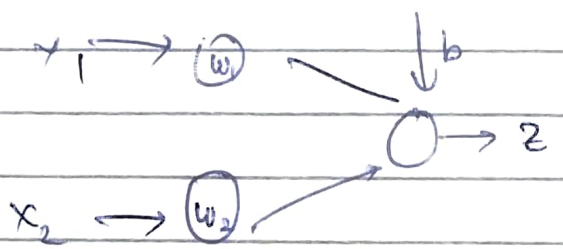
$$w := w - \alpha \frac{dJ(w)}{dw}$$

}



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

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



$$z = w_1 x_1 + w_2 x_2 + b$$

$$a = \sigma(z)$$

$$L(a, y)$$

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

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

$$= a - y$$

$$\frac{\partial L}{\partial w_1} = dw_1 = x_1 \, dz$$

$$dw_2 = x_2 \, dz$$

$$db = dz$$

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

* LR on m examples

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

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

Initialize

$J = 0 \quad dw_1 = 0 \quad dw_2 = 0 \quad db = 0$

Vectorization

- $z = w^T x + b$

  avoid explicit for loops whenever possible

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

$dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$

$dz = [dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}]$

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

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

implementation:

```
for i in range(epoch):
    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)

$w := w - \alpha\, dw$

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

$\hat{y}$ If $y=1$ $p(y|x) = \hat{y}$

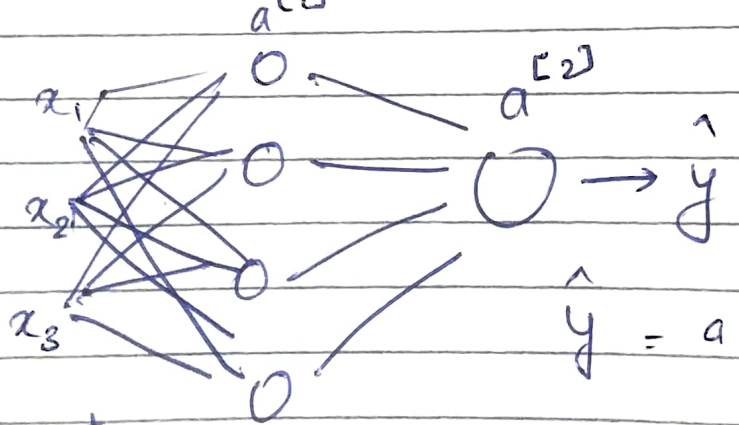
$y=0$ $p(y|x) = 1-\hat{y}$

$p(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$

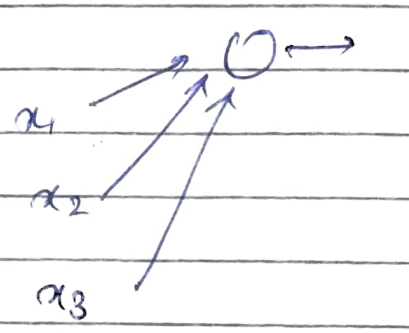
If $y=1$: $p(y|x) = \hat{y}\ (1-\hat{y})^{0}$

$y=0$: $p(y|x) = \hat{y}$

Shallow Neural Network



What a single node does -



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

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

Similarly

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

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

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

Vectorize this:

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

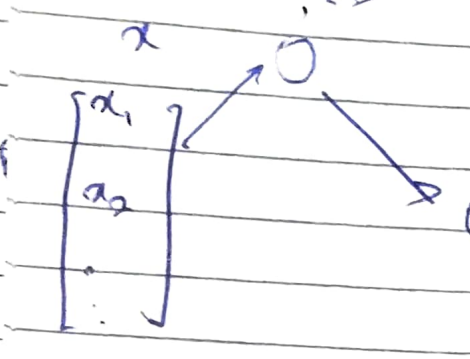
Shape: $[4, 3]$ $[3, 1]$

$$\downarrow$$

$$\sigma([\;\;])$$

$$\downarrow$$

$$a$$



for i=1 to m:

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

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

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

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

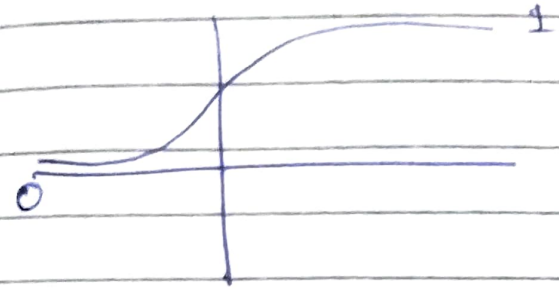
σ is not necessarily the sigmoid fn it can be any activation fn.

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

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

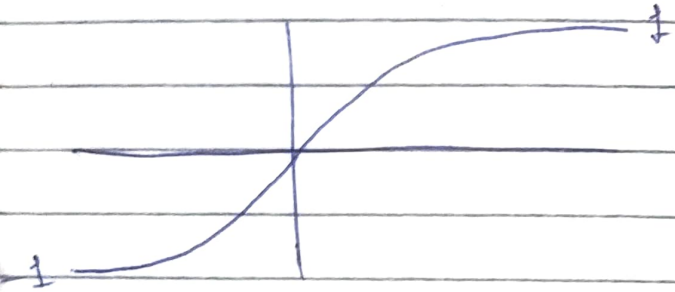
$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Activation fns:



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

used in binary classification

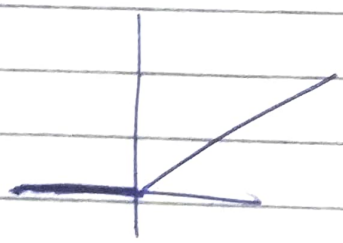


$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

is much superior than sigmoid

these often pose problems for G.D. as for large positive or negative values as the slope is almost close to 0. $\frac{dy}{dx} \longrightarrow 0$

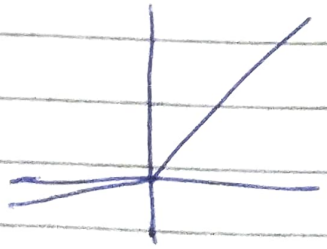
ReLU



$a = \max(0, z)$

Rectified Linear Unit

Leaky ReLU



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

can be any small number

DATE: / /

Why use non linear Activation fns?

Given x:

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

$$a^{[1]} = g^{[1]}(z^{[1]}) \rightarrow \text{make it } z^{[1]}$$

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

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

Let $g(x) = x$

then

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

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

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

$$\downarrow \qquad\qquad \downarrow$$

$$w'' \qquad\qquad b''$$

$$= w'' x + b''$$

back to the ~~previous~~ layer 1 form of eqn

Hence the hidden layer is useless

→ might just have these w'' & b'' in the input layer

Derivatives

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

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

GD for NN

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

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

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

~~Repeat {~~

~~formulas for computing derivatives~~

Forward propagation:

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

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

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

$$A^{[2]} = g^{[2]}(z^{[2]})$$

Back Propagation

$$dz^{[2]} = A^{[2]} - Y$$

$$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$

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

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

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

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

$dw^{[1]}$ instead of $dw[2]$

Initialize w to small positive values

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

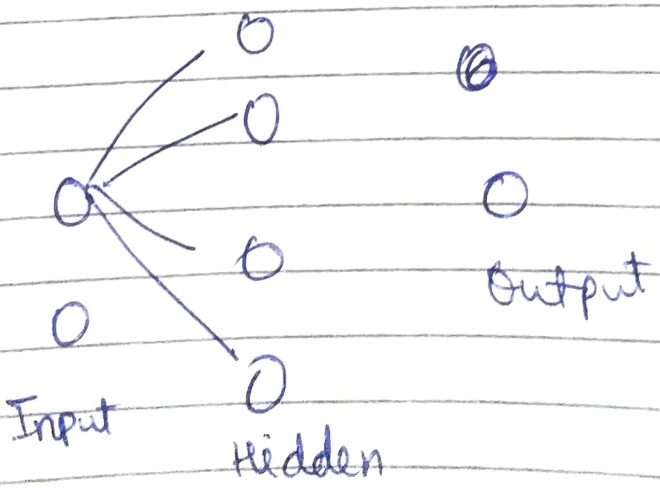
$b^{[1]} = np.zeros((2,2))$

If weight values are large you may end up on the flat sections of tanh & sigmoid so Hence gradient descent will be very slow.

Programming Assignment

$x = 2 \times 400$

$y = 1 \times 400$

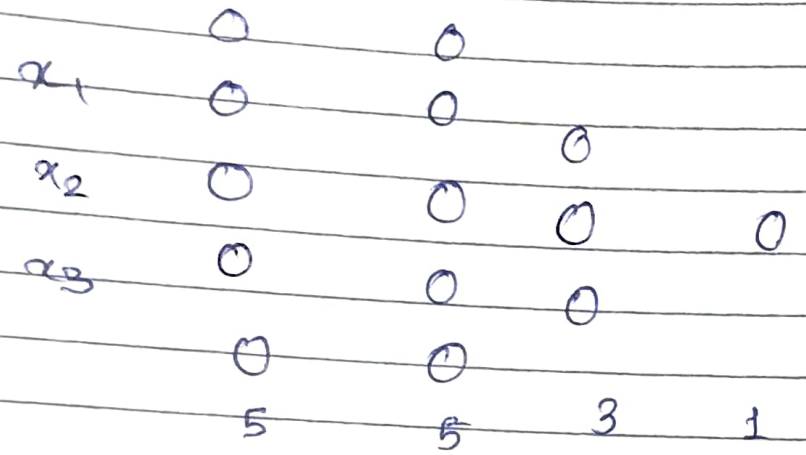


$w_1 = 4 \times 2$ $\quad b_1 = 4 \times 1$

$w_2 = 1 \times 4$ $\quad b_2 = 1 \times 1$

Deep L layer NN

Notations



$L = 4$ (has 4 layers)

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

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

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

$w^{[l]}$, $b^{[l]}$: weights & bias for layer l

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

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

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

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

6:35 $a^{[l]} = g^{[l]}(z^{[l]})$ $dim(a) = (n^{[l]}, 1)$

$$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$$

$$a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$$

generalized to for $l = 1$ to 4

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

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

$$dim(w^{[l]}) = (n^{[l]}, n^{[l-1]})$$

Why deep?

Ex speech recognition

wave → low level features (pitch etc) → Phonetic sounds → words from phonetic sounds

Analogous to many layer logic gate circuits.

Forward & Backward fns

x_1	0	0	0	
x_2	0	0	0	$0 \rightarrow y$
x_3	0	0	0	
x_4	0	0	0	

↓

Consider layer l

Forward: input: $a^{[l-1]}$

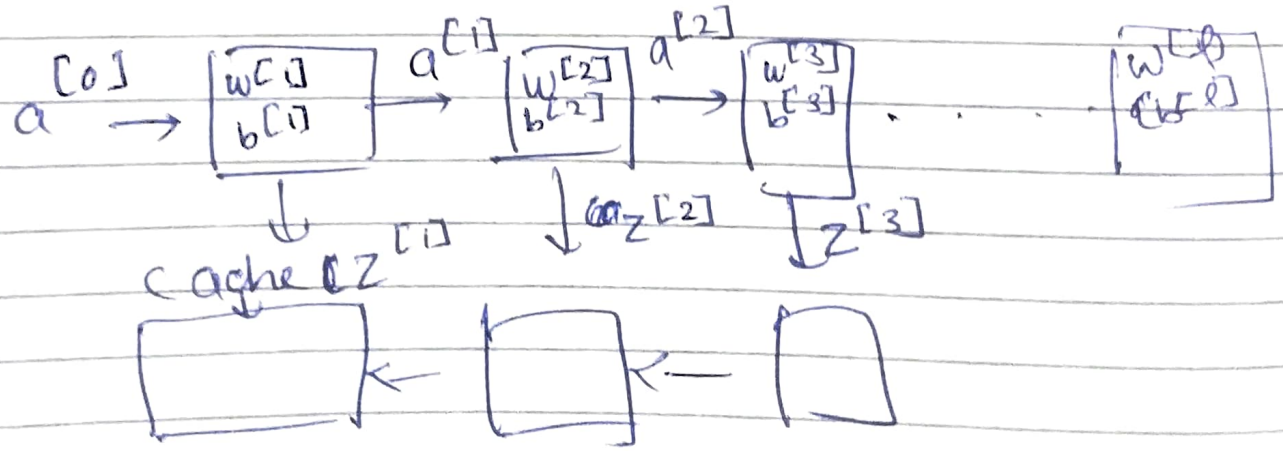
output: $a^{[l]}$, cache($z^{[l]}, w^{[l]}, b^{[l]}$)

Backward: input: $da^{[l]}$

output: $da^{[l-1]}$

$dw^{[l]}$

$db^{[l]}$



$dw^{[l]} = dz^{[l]} * a^{[l-1]T}$

2:30

B.P.

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

$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot a^{[l-1]T}$

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

$da^{[l-1]} = dw^{[l]T} \cdot dz^{[l]}$

#Parameters & Hyperparameters

↓ W & b

↘ α, iterations, hidden layer, hidden units, choice of activation fns.