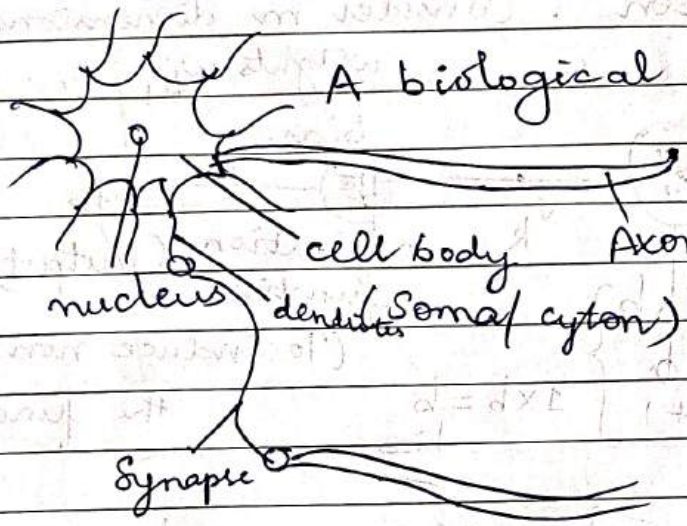
Deep Learning

Structure of neuron:

- Axon long in size.
- Dendron: for receiving information from synapse (Axon connected of another neuron)
- Cell body: for processing information



If a decision boundary can be realised to separate the distinct classes — linearly separable data.

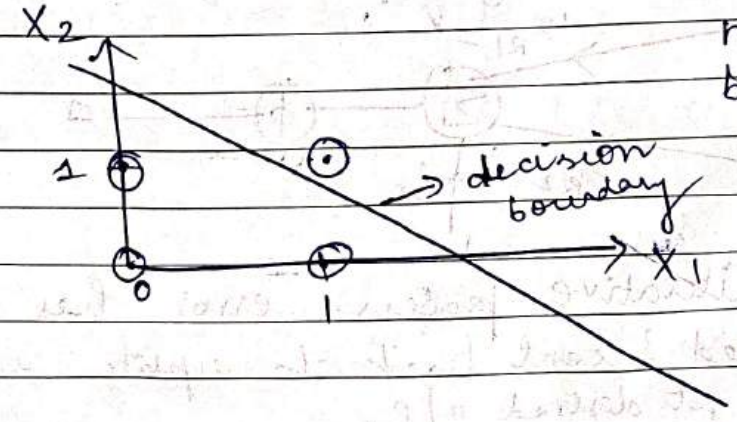
Logic Gate AND, or etc.

X_1	X_2	$X_1 X_2$
0	0	0
0	1	0
1	0	0
1	1	1

XOR gate (non linearly separable)

↓

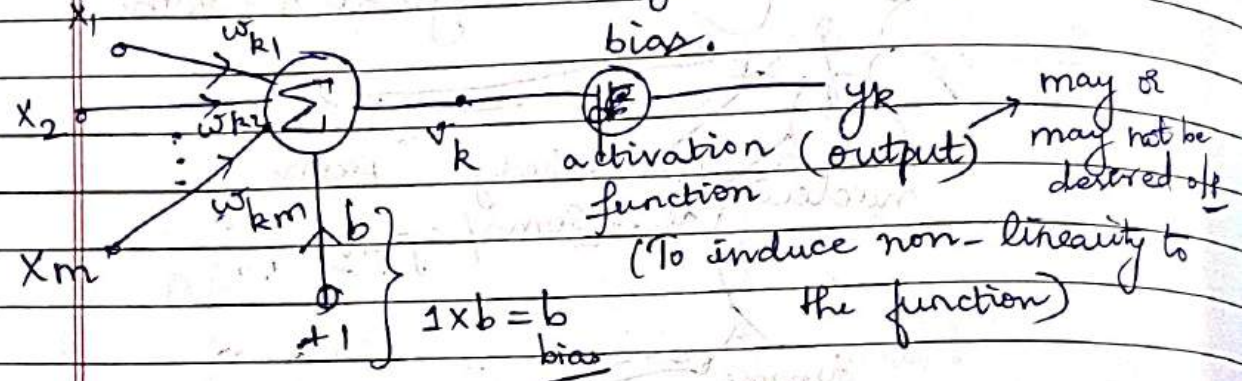
multiple decision boundaries to separate data.



A single perceptron is needed for solving linearly separable problems.

Multi-layer perceptron (consists of inputs & multiple hidden layers) to solve non linearly separable problems.

A perceptron : Consider m dimensional data with weights $w_{k1}, \cdots, w_{km}$ with bias.



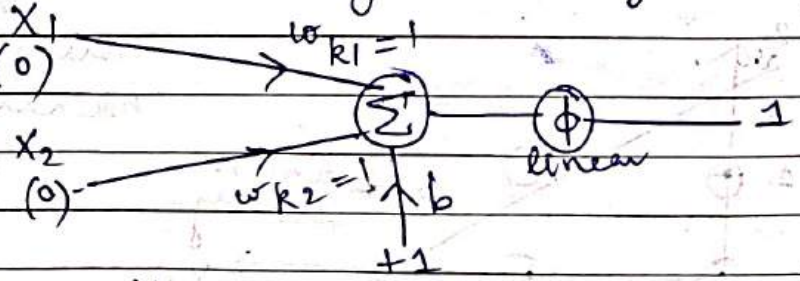
$$v_k = \sum_{j=1}^{m} w'_{kj} \cdot x_j + b \quad \text{——— (1)}$$

$$y_k = \phi(v_k) \quad \text{——— (2)}$$

$$y_k = \phi\left(\sum_{j=1}^{m} w'_{kj} \cdot x_j + b\right)$$

Consider 2 inputs X_1, X_2 with weights 1 each and a bias term 1 is induced.

for and gate (o/p by multiplication)



It is an iterative process. error has to be calculated & sent back to update weights to get desired o/p.

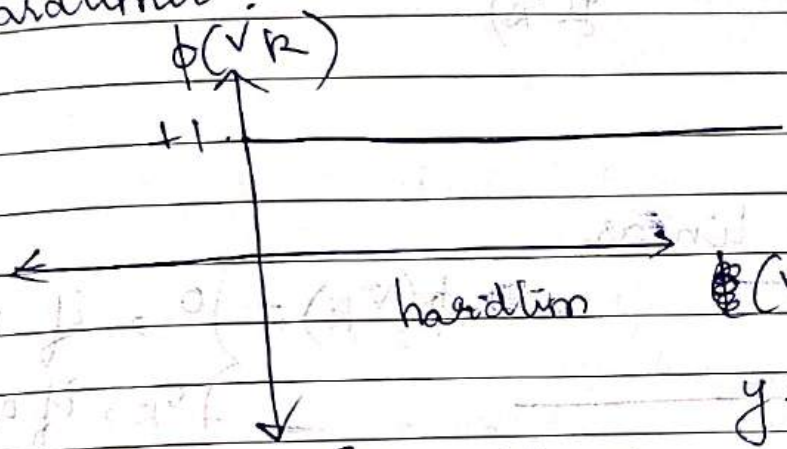
$(0 \times 1) + 1 \longrightarrow 1$

desired output = 0

So, error should be calculated.

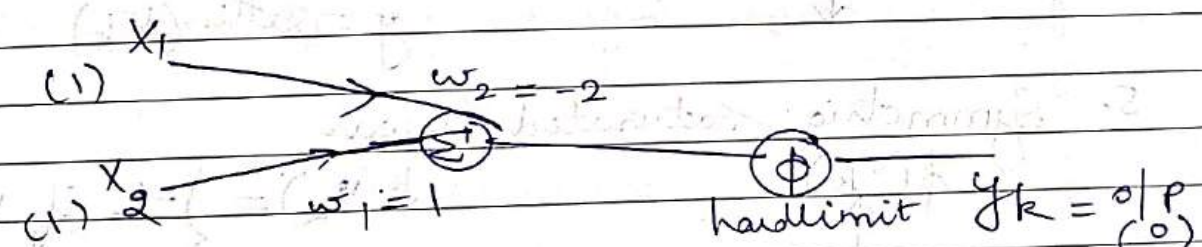
Activation function:

1. Hardlimit :-



$y = \text{hardlim}(v_k)$

$$\phi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases}$$

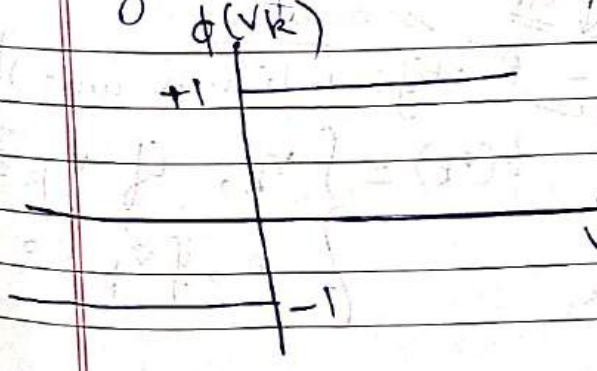


$v_k = -2(1) + 1(1) + 0$ bias

$v_k = -1$

$o/p = 0$

2. Symmetric hard limit:

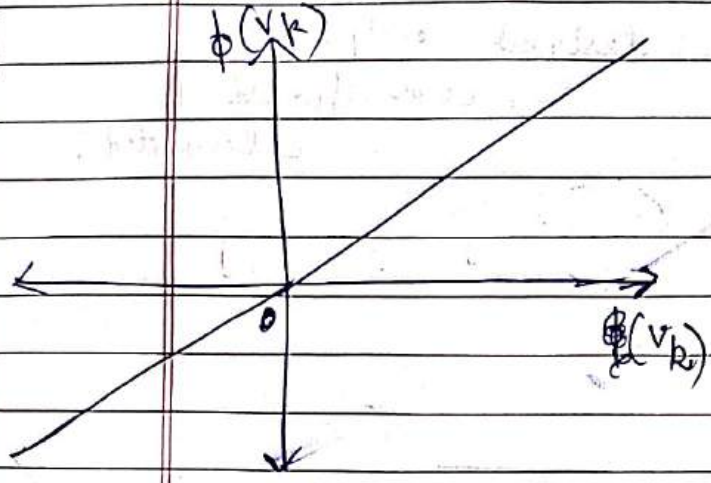


$$\phi(v_k) = \begin{cases} +1 & \text{if } v_k \geq 0 \\ -1 & \text{if } v_k < 0 \end{cases}$$

$y = \text{hardlims}(v_k)$

symmetric

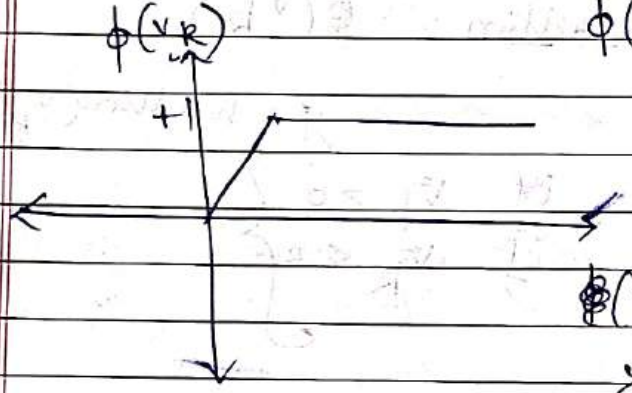
3. Linear



$$\phi(v_k) = v_k$$

$$y = lin(v_k)$$

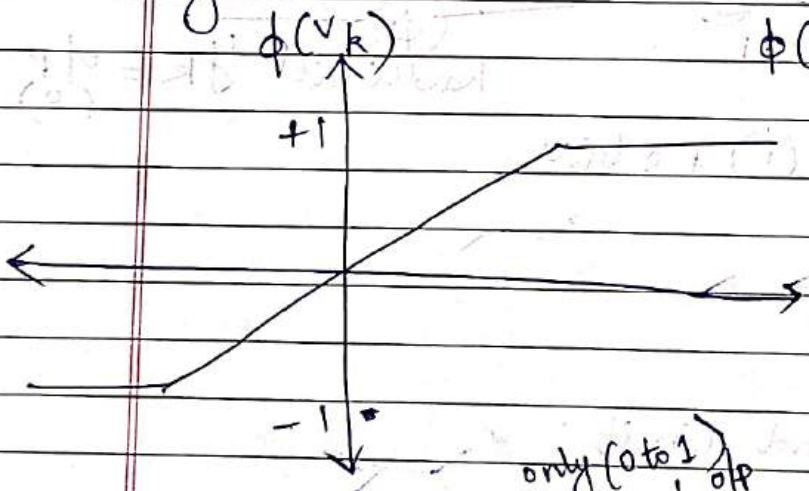
4. Saturated linear



$$\phi(v_k) = \begin{cases} 0, & \text{if } v_k < 0 \\ v_k, & \text{if } 0 \le v_k \le 1 \\ 1, & \text{if } v_k > 1 \end{cases}$$

$$y = satlin(v_k)$$

5. Symmetric saturated linear



$$\phi(v_k) = \begin{cases} -1, & \text{if } v_k < -1 \\ v_k, & \text{if } -1 \le v_k \le 1 \\ 1, & \text{if } v_k > 1 \end{cases}$$

linear

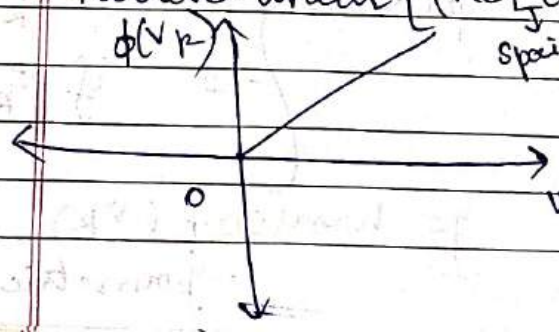
$$y = satlins(v_k)$$

symmetric saturated

6. Positive linear [(ReLU - Rectified linear unit)]

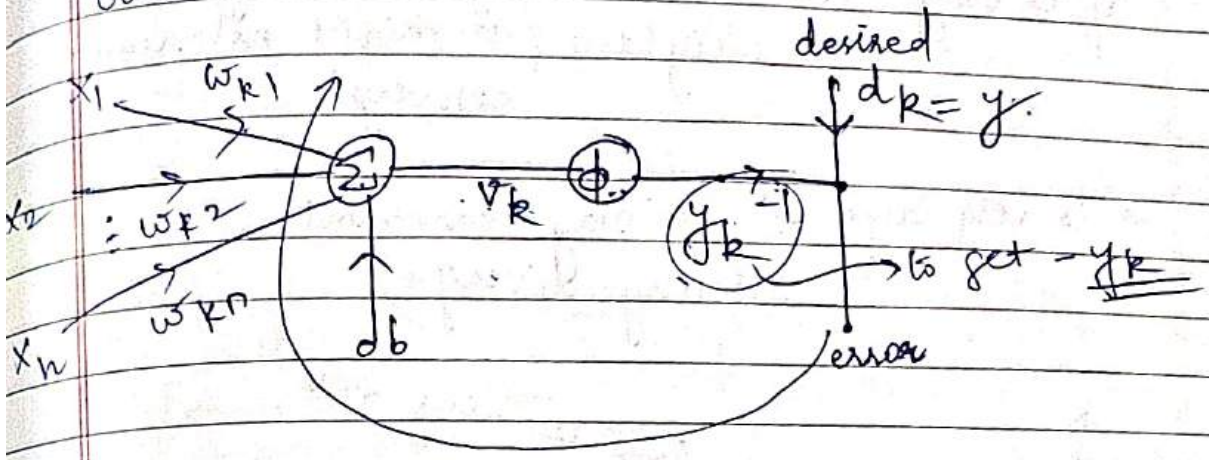
only (0 to 1) o/p

special case



$$\phi(v_k) = \begin{cases} v_k, & \text{if } v_k \ge 0 \\ 0, & \text{if } v_k < 0 \end{cases}$$

Gradient - descent:

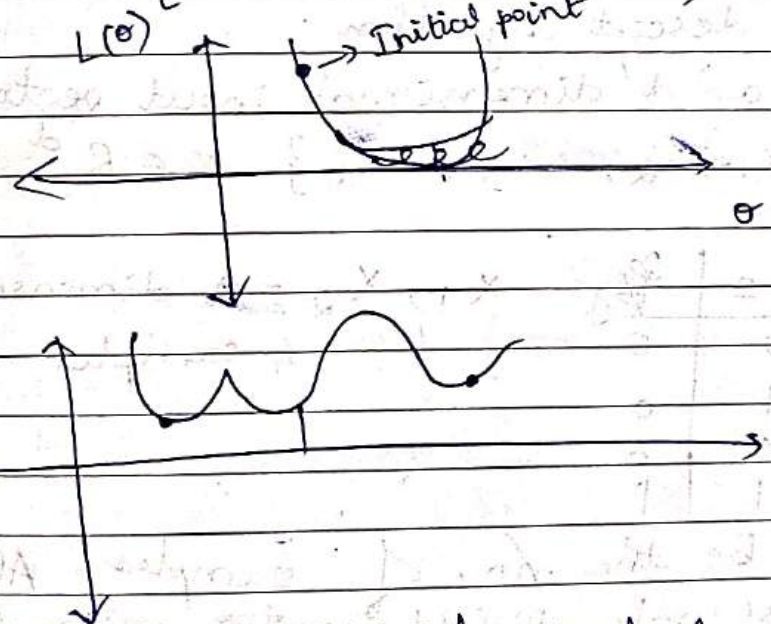


$$v_k = \sum_{i=1}^{n} w^T x + b$$

$$y_k = \phi(v_k) \qquad \text{error} = y - y_k$$

Gradient descent algorithm:-

- Start from any point on curve with initial weights and tend to reach the only global minima (not local minima).



weights are initialised randomly, so global minima is not always promised.

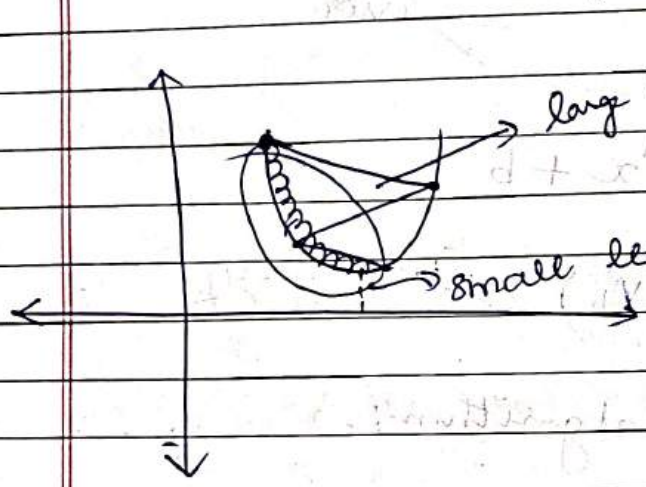
our objective is to minimize error..

$$w_{n+1} = w_n - \eta \frac{dL}{d\sigma}$$

η ↓ learning rate / step size — how it effects the performance (how it converges)

η is very small - no drastic change in weights (it might not reach convergence

η is very large - weights may overshoot (it may diverge)



Initialisation of weights and learning rate affect the model performance.

Gradient descent algorithm:

Consider a 'd' dimensional input vector

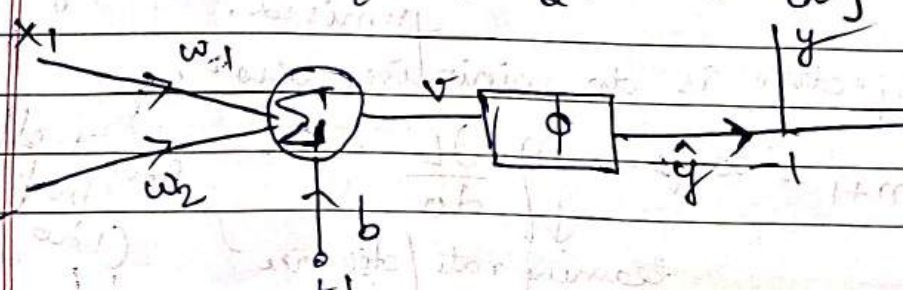
$$X = \{x_1, x_2, \ldots, x_d\} \quad X \in R^d$$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1, x_2 = 2 dimensions
4 samples

Let m be the no. of samples. Also, a d dimensional weight vector $w \in R^d$

$$w = \{w_1, w_2, \ldots, w_d\}$$



$y \rightarrow$ desired output

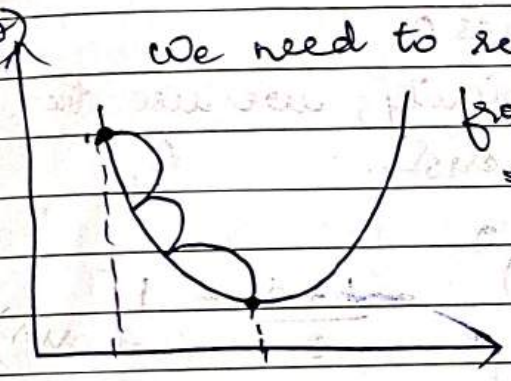
$\hat{y} \rightarrow$ obtained / predicted output.

Error $(e) = y - \hat{y}$

Loss $(w, b) = \frac{1}{2}\sum (y_i - \hat{y}_i)^2$ [obtained by partial derivative]

(weight, bias)

[We need to reduce error w.r.t. weight vector & bias] → to reduce computational intensity ⑤



We need to reach the lowest possible point from initial guess, ~~the~~ ~~reach~~ ~~the minimum~~ which is done by calculation of gradient.

(w, b) or any parameter

$$\theta_{n+1} = \theta_n - \eta \frac{\partial L(\theta)}{\partial \theta}$$

$\hat{y}_i = \phi(w x_i + b)$ (considering i^{th} sample)

In this example, $1 \leq i \leq 4$. (one sample only)

$\phi(v_k) = v_k$, linear activation fn

Let ϕ be linear activation function.

So, that the derivative = 1.

$$\hat{y}_i = w x_i + b \quad \text{——— ①}$$

$$w_{n+1} = w_n - \eta \frac{\partial L(w)}{\partial w} \text{ ——②} \quad ; \quad b_{n+1} = b_n - \eta \frac{\partial L(b)}{\partial b} \text{ ——③}$$

$$\frac{\partial L(w)}{\partial w} = \frac{\partial L(w)}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w} \quad \text{[chain rule]} \quad \text{---------- (4)}$$

Digression

error $= y - \hat{y}$.
$= 3 - 2$
$= 1$

error $= y - \hat{y}$
$= 3 - 4$
$= -1$

Net error becomes 0

To avoid this ambiguity, we use the mean squared error.

$$MSE = \frac{1^2 + (-1)^2}{2} = \frac{1+1}{2} = 1 \quad \text{(not error)}$$

That is why loss is mean squared error.

Differentiate $L(w, b) = \frac{1}{2} \sum (y_i - \hat{y}_i)^2$ w.r.t $\hat{y}_i$

$$\frac{\partial L(w)}{\partial \hat{y}_i} = \frac{1}{2} \times 2 (y_i - \hat{y}_i) \cdot \frac{\partial}{\partial \hat{y}_i} (-\hat{y}_i)$$

$$= -(y_i - \hat{y}_i) \quad \text{---------- (6)}$$

Differentiate $\hat{y}_i = w x_i + b$ w.r.t. w

$$\frac{\partial \hat{y}_i}{\partial w} = x_i \quad \text{---------- (7)}$$

$$\frac{\partial L(w)}{\partial w} = -(y_i - \hat{y}_i) x_i \quad \text{(from 4)}$$

Substitute above equation in (2)

$$w_{n+1} = w_n - \eta(-(y_i - \hat{y}_i)) x_i$$

$$\therefore w_{n+1} = w_n + \eta (y_i - \hat{y}_i) x_i \quad \text{——— (8)}$$

$$w_{n+1} = w_n + \eta \text{ error } x_i$$

$$\frac{\partial L(b)}{\partial b} = \frac{\partial L(b)}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial b} \quad \text{——— (9)}$$

$$\because L(w, b) = \frac{1}{2} \sum (y_i - \hat{y}_i)^2$$

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{1}{2} \times 2 (y_i - \hat{y}_i) \frac{\partial}{\partial \hat{y}_i} (-\hat{y}_i)$$

$$= -(y_i - \hat{y}_i) \quad \text{——— (10)}$$

Differentiate $\hat{y}_i = wx_i + b$ w.r.t. b

$$\frac{\partial \hat{y}_i}{\partial b} = 1 \quad \text{——— (11)}$$

from (9), (10), (11)

$$\frac{\partial L(b)}{\partial b} = -(y_i - \hat{y}_i) \quad \text{(from 4)}$$

$$b_{n+1} = b_n - \eta \frac{\partial L(b)}{\partial b}$$

$$b_{n+1} = b_n + \eta (\hat{y}_i - y_i) \quad \text{——— (12)}$$

For an AND gate, calculate the error and append overall error, plot decision boundary along with I/P for 100 ~~iterations~~ epochs. (use scatter plot)

i/p array

o/p array $\quad x = np.array([$

error = ( )

for no of epochs $\quad$ learning rate $\eta = 0.1$

for length (i/p)

$y = wx + b$

$y = hardlim(y)$

error

update weights

append (total error)

Decision-boundary:

$w x + b = 0$

$w_1 x_1 + w_2 x_2 + b = 0$

Plot from -0.5 to 0.5

$$x_2 = \frac{-b - w_1 x_1}{w_2}$$

AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Solve an AND problem assuming initial weights $[w_1, w_2] = [1, 2]^T$, initial bias = 1

$w_{n+1} = w_n + \eta (y_i - \hat{y}_i) x_i$

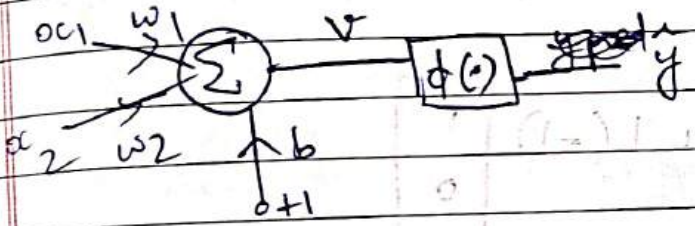
and $b_{n+1} = b_n + \eta (y_i - \hat{y}_i)$ $\eta = 1$

using gradient descent algorithm

1st iteration :-

(i) $x_1 = 0$, $x_2 = 0$, $y = 0$

x_1	x_2
0	0
0	1
1	0
1	1



$v = w_1 x_1 + w_2 x_2 + b$

$= 1(0) + 2(0) + 1$

$= 1$

$\hat{y} = \text{hardlim}(1)$

$= 1$

error = desired − obtained

$= 0 - 1$

$= -1$

$w_{n+1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 1(-1) \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$w_{n+1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$b_{n+1} = 1 + 1(-1) = 0$

$w_{new} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $b_{new} = 0$

ii) $x_1 = 0$

$x_2 = 1$

$y = 0$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(0) + 2(1) + 0$

$= 2$

$\hat{y} = \text{hardlim}(2)$

$= 1$

error = $0 - 1 = -1$

$w_{n+1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 1(-1) \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$= \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$= \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$b_{n+1} = 0 + 1(-1) = -1$

$w_{new} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $b_{new} = -1$

iii) $x_1 = 1, x_2 = 0$

$\hat{y} = 0$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(1) + 1(0) - 1$

$= 0$

$\hat{y} = \text{hardlim}(0) = 1$

error $= -1$

$w_{n+1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1(-1) \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$b_{n+1} = -1 + (1)(-1) = -2$

$w_{new} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $b_{new} = -2$

iv) $x_1 = 1, x_2 = 1$

$\hat{y} = 1$

$v = w_1 x_1 + w_2 x_2 + b$

$= 0(1) + 1(1) - 2$

$= -1$

$\hat{y} = \text{hardlim}(-1) = 0$

error $= 1 - 0 = 1$

$$w_{n+1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 1(1)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$b_{n+1} = -2 + 1(1) = -1$$

$$w_{new} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad b_{new} = -1$$

2nd iteration :-

i) $x_1 = 0, x_2 = 0, \hat{y} = 0$

$$v = w_1 x_1 + w_2 x_2 + b = 1(0) + 2(0) - 1 = -1$$

$\hat{y} = \text{hardlim}(-1) = 0$

error = 0

No need to update weights & biases.

ii) $x_1 = 0, x_2 = 1, \hat{y} = 0$

$$v = w_1 x_1 + w_2 x_2 + b$$
$$= 1(0) + 2(1) - 1$$
$$= 1$$

$\hat{y} = \text{hardlim}(1) = 1$

error = $0 - 1 = -1$

$$w_{n+1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 1(-1)\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$b_{n+1} = -1 + (1)(-1) = -2$$

$$w_{new} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad b_{new} = -2$$

iii) $x_1 = 1, x_2 = 0, \hat{y} = 0$

$$v = w_1 x_1 + w_2 x_2 + b = 1(1) + 0(1) - 2 = -1$$

$\hat{y} = \text{hardlim}(-1) = 0$ error = $0 - 0 = 0$

No need to update weights & biases.

iv) $x_1 = 1$, $x_2 = 1$, $\hat{y} = 1$

$v = w_1 x_1 + w_2 x_2 + b$
$= 1(1) + 1(1) - 2$
$= 0$

$\hat{y} = \text{hardlim}(0) = 1$

error $= 1 - 1 = 0$

No need to update weights & biases.

However, we didn't get for 2nd combinations. So, need to reiterate.

3rd iteration:

(i) $x_1 = 0$, $x_2 = 0$, $\hat{y} = 0$

$v = w_1 x_1 + w_2 x_2 + b$
$= 0(1) + 0(1) - 2$
$= -2$

$\hat{y} = \text{hardlim}(-2) = 0$

error $= 0 - 0 = 0$

No need to update weights & biases.

(ii) $x_1 = 0$, $x_2 = 1$, $\hat{y} = 0$

$v = w_1 x_1 + w_2 x_2 + b$
$= 0(1) + 1(1) - 2$
$= -1$

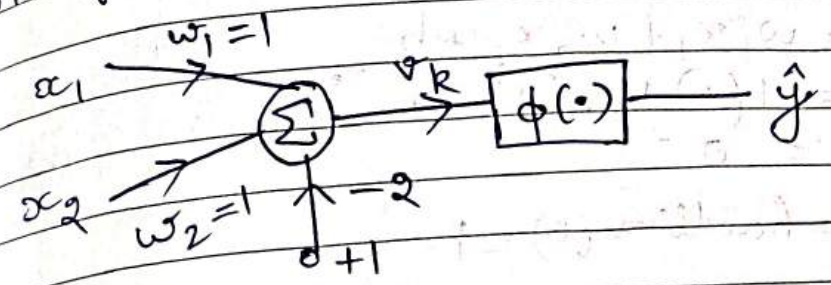
$\hat{y} = \text{hardlim}(-1) = 0$

error $= 0$

No need to update weights & biases.

classmate
Date
Page

Therefore,



$$\hat{y} = \phi(x_1 + x_2 - 2)$$

OR problem

x_1	x_2	$\hat{y}$
0	0	0
0	1	1
1	0	1
1	1	1

$[w_1, w_2] = [1, 1]^T$

bias = 0

$\eta = 1$

1st iteration:

(i) $x_1 = 0$, $x_2 = 0$, $\hat{y} = 0$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(0) + 1(0) + 0$

$= 0$

$\hat{y} = \text{hardlim}(0) = 1$

error = $0 - 1 = -1$

~~No need to update weights & bias.~~

(ii) ~~$x_1 = 0$, $x_2 = 1$, $\hat{y} = 1$~~

$$w_{n+1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1(-1)\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$b_{n+1} = 0 + 1(-1) = -1$$

(ii) $x_1 = 0, x_2 = 1, \hat{y} = 1$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(0) + 1(1) - 1$

$= 0$

$\hat{y} = \text{hardlim}(0) = 1$

error $= 0$

(iii) $x_1 = 1, x_2 = 0, \hat{y} = 1$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(1) + 1(0) - 1$

$= 0$

$\hat{y} = \text{hardlim}(0) = 1$

error $= 0$

(iv) $x_1 = 1, x_2 = 1, \hat{y} = 1$

$v = w_1 x_1 + w_2 x_2 + b$

$= 1(1) + 1(1) - 1$

$= 1$

$\hat{y} = \text{hardlim}(1) = 1$

error $= 0$

2nd iteration

(i) $x_1 = 0, x_2 = 0, \hat{y} = 0$

$v = w_1 x_1 + w_2 x_2 + b$

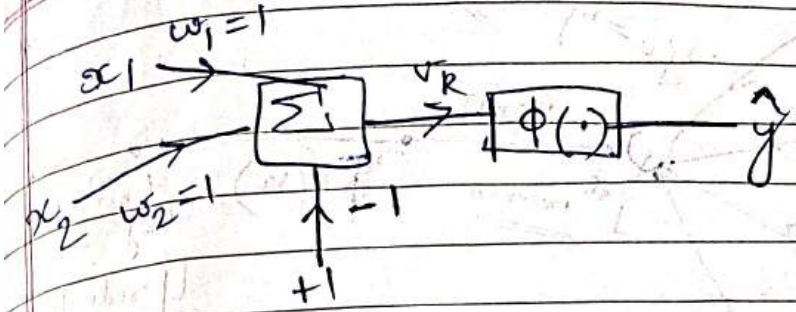
$= 1(0) + 1(0) - 1$

$= -1$

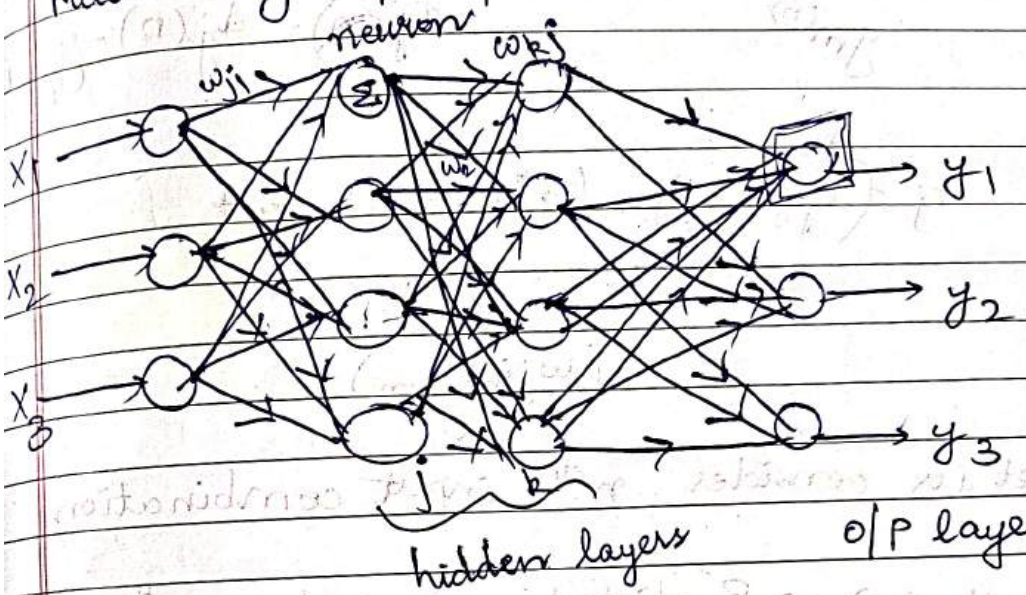
$\hat{y} = \text{hardlim}(-1)$

$= 0$

error $= 0$



Multi-layer perceptron:

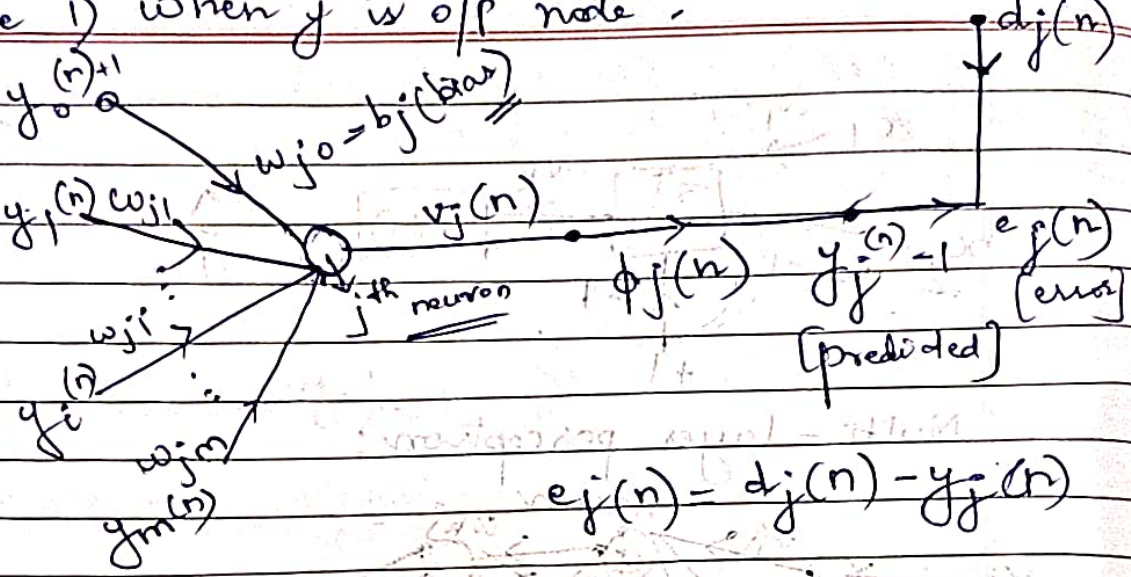


- Required for non linearly separable problems.
  Ex:- XOR, XNOR gates

Back propagation:
Weights get updated for hidden layers and input layers by propagating the gradient of loss function using the chain rule.

Cases:
- error updation for o/p neuron (If error occurs in o/p layer, it is similar to single layer perceptron)
- error updation for hidden neuron (Hidden layer receives updates weights from different layers, error from o/p layer is also received by hidden layer)

Case i) when j is o/p node.



$$e_j(n) = d_j(n) - y_j(n)$$

$$v_j = (w_{j0} \times y_0) + (w_{j1} \times y_1) + \cdots + (w_{ji} \times y_i) + \cdots + (w_{jm} \times y_m)$$

Let us consider n^{th} input combination

$y_j(n)$ → Predicted output for n^{th} input.

$d_j(n)$ → Desired output

$$\text{error} = e_j(n) = d_j(n) - y_j(n) \quad \text{——— (4)}$$

$$\text{Loss} = \mathcal{Z}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \quad \text{——— (1)}$$

(for all classes)

$$= \frac{1}{2} \sum_{j \in C} \left(d_j(n) - y_j(n)\right)^2 \quad \text{——— (2)}$$

Weight updation:

$$w_{n+1} = w_n \boxed{- \eta \frac{\partial \mathcal{Z}(n)}{\partial w_{ji}(n)}} \rightarrow \text{change in weights}$$

$$w_{n+1} = w_n + \Delta w_{ji}$$

η = learning rate

$\frac{\partial Z(n)}{\partial w_{ji}(n)}$ = Gradient of weight vector

for n^{th} iteration, error found for 0, 0 we update weights for all vectors → [we updated w_1, w_2 etc]
The entire vector is updated, so w_{ji}.

$$\Delta w_{ji} = -\eta \frac{\partial Z(n)}{\partial w_{ji}(n)} \quad \text{③}$$

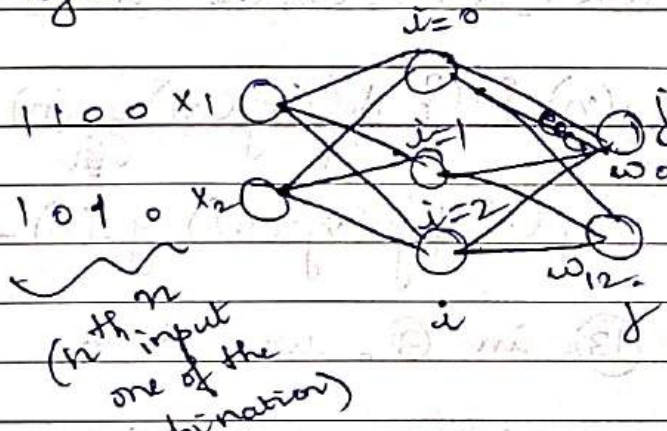
↳ $[w_{j0}, w_{j1}, \cdots, w_{ji}]$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n) \quad \text{⑤}$$

↳ Indicates from which layer of neuron connection is established

$$y_j(n) = \phi_j(v_j(n)) \quad \text{⑥}$$

Digression:



Continuation:

find $\frac{\partial Z(n)}{\partial w_{ji}(n)}$.

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$ [chain rule]

From ①, $\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n)$ ——— ⑧

From ④, $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$ ——— ⑨

$[e_j(n) = d_j(n) - y_j(n)]$

From ⑥, $y_j(n) = \phi_j(v_j(n))$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \phi'_j(v_j(n)) \quad \text{——— ⑩}$$

From ⑤, $\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$ ——— ⑪

Substituting ⑧, ⑨, ⑩, ⑪, in ⑦ we get-

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \{-1\} \phi'_j(v_j(n)) y_i(n) \quad \text{——— ⑫}$$

$$= -e_j(n) \phi'_j(v_j(n)) y_i(n) \quad \text{——— ⑬}$$

Substituting ⑬ in ②, we obtain

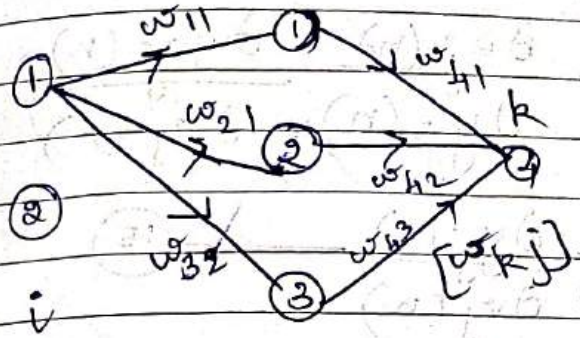
$$\Delta w_{ji}(n) = \eta e_j(n) \phi'_j(v_j(n)) y_i(n)$$

Assume local gradient 'δ'

$$\delta_j = e_j(n) \phi'_j(v_j(n)), \quad j \text{ is output node.} \quad \text{——— ⑭}$$

Therefore, $\Delta w_{ji}(n) = \eta \delta_j y_i(n)$

Digression :-

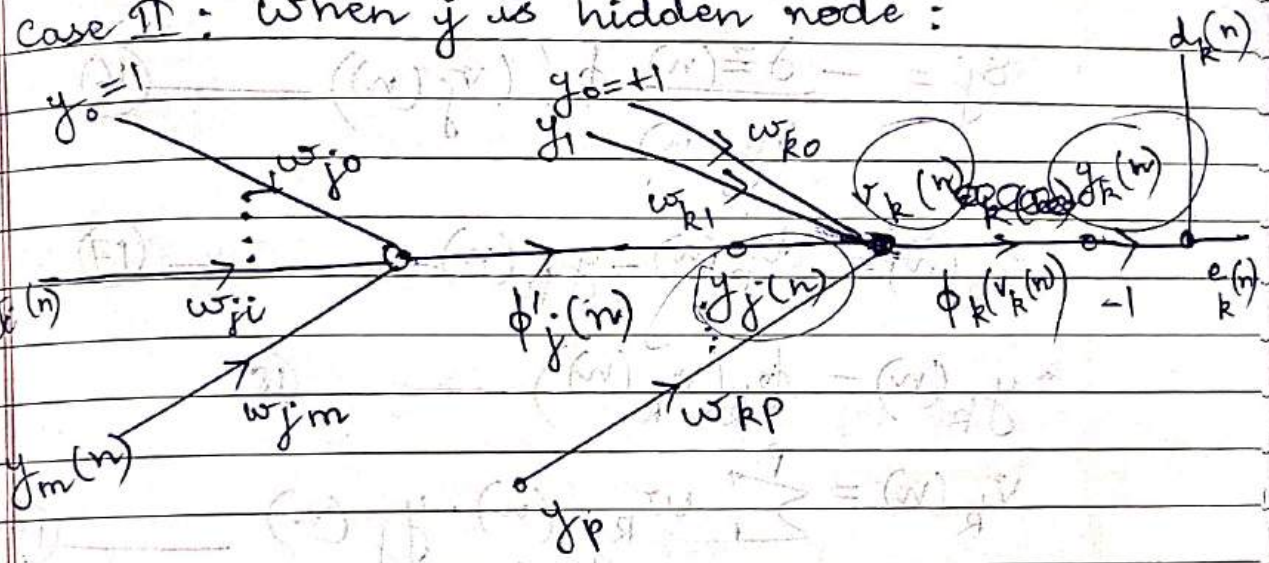


$n \rightarrow$ combination of inputs (not weight vector)

$n=1 \rightarrow$ 00 combination

$n=2 \rightarrow$ 01 etc.

Case II : When j is hidden node :



From (14), for output node we had - [previously]

$$\delta_j = e_j(n)\,\phi'_j(v_j(n))$$

{Trying to do a backward pass.}

$$\delta_j = -e_j(n)(-1)\,\phi'_j(v_j(n))$$

From (8)

$$\delta_j = \frac{-\partial \mathcal{Z}(n)}{\partial e_j(n)}(-1)\,\phi'_j(v_j(n))$$

From (9)

$$\delta_j = \frac{-\partial \mathcal{Z}(n)}{\partial e_j(n)}\,\frac{\partial e_j(n)}{\partial y_j(n)}\,\phi'_j(v_j(n))$$

From (10)

$$\delta_j = -\frac{\partial Z(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\delta_j = -\frac{\partial Z(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \quad \text{——— (15)}$$

but $\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_j'(v_j(n))$

$$\delta_j = -\frac{\partial Z(n)}{\partial y_j(n)} \phi_j'(v_j(n)) \quad \text{——— (16)}$$

$$e_k(n) = d_k(n) - y_k(n) \quad \text{——— (17)}$$

$$y_k(n) = \phi_k(v_k(n)) \quad \text{——— (18)}$$

$$v_k(n) = \sum_{j=0}^{p} w_{kj}(n) \cdot y_j(n) \quad \text{——— (19)}$$

where 2 → "constant"

$$Z(n) = \frac{1}{2} \sum_{k \in C} e_k(n) \quad \text{——— (20)}$$

(all classes or combinations)

$$\frac{\partial Z(n)}{\partial y_j(n)} = \sum e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \quad \text{——— (21)}$$

(from 20)

output to input

$$\frac{\partial Z(n)}{\partial y_j(n)} = \sum e_k(n) \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \quad \text{——— (22)}$$

From (17)

$$\frac{\partial Z(n)}{\partial y_j(n)} = -\sum e_k(n) \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \quad \text{——— (23)}$$

From (18)

$$\frac{\partial Z(n)}{\partial y_j(n)} = -\sum_k e_k(n)\, \phi'_k(v_k(n)) \frac{\partial v_k(n)}{\partial y_j(n)} \quad \text{---(24)}$$

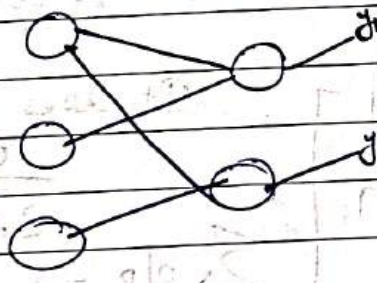
From (19)

$$\frac{\partial Z(n)}{\partial y_j(n)} = -\sum_k e_k(n)\, \phi'_k(v_k(n))\, w_{kj}(n) \quad \text{---(25)}$$

$$\therefore \frac{\partial Z(n)}{\partial y_j(n)} = -\sum_k \delta_k w_{kj}(n) \quad \text{---(26)}$$

where $\delta_k = e_k(n)\phi'_k(v_k(n))$

Digression:



O/P layer (there is no summation in because $\frac{\partial Z(n)}{\partial y_j(n)}$ it's the last layer)

However, in hidden layer, there is summation in (26) because the neuron receives error correction from different layers.

Continuation:-

Substituting (26) in (18), we get -

$$\delta_j = \phi'_j(v_j(n)) \sum_k \delta_k w_{kj}(n) \quad \text{---(27)}$$

where j is hidden node.

Note:

$\Delta w_{ji} = \eta \delta_j y_i$

$\delta_j = \phi'_j(v_j(n)) e_j$ j is o/p node.

$\delta_j = \phi'_j(v_j(n)) \sum_k \delta_k w_{kj}$ j is hidden node.

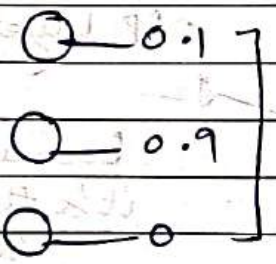
If there is one more hidden layer, do substitution again (series of steps!)

Sigmoid - binary classification $\sigma(x) = \frac{1}{1+e^{-x}}$

Softmax - multiclass classification

[Probability oriented]

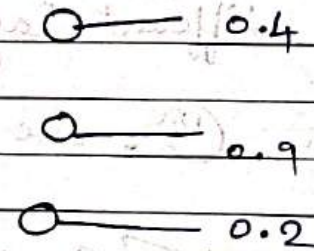
Suppose there are three classes cat, dog, zebra & we used softmax.



It does normalisation $\frac{e^x}{\sum e^x}$

$\sum$ o/p = 1

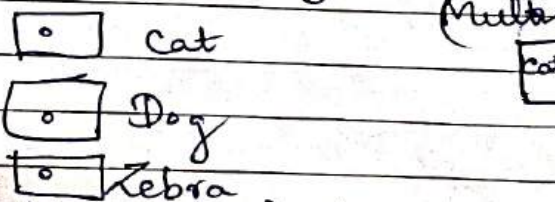
If sigmoid is used in o/p (all predicted values will lie from 0 to 1)



Therefore, it is not used in multiclass classification.

Multi-label classification:-

In same image, there are n-number of images.



Sigmoid activation

softmax gives probabilistic output.

How probabilities of softmax is calculated?

$\rightarrow 0.7 \quad = \frac{0.7}{0.7+0.2+0.1+0.4} = 7/14$

$\rightarrow 0.2 \quad = 2/14$

$\rightarrow 0.1 \quad = 1/14$

$\rightarrow 0.4 \quad = 4/14$

Activation functions:

① Log-sigmoid: [Ranges from 0 to 1]

$$y_o = \phi(v_j(n)) = \frac{1}{1+e^{-a v_j(n)}} \quad \text{——— } \star$$

$$\phi'(v_j(n)) = \frac{-1}{\{1+e^{-a v_j(n)}\}^2} \frac{\partial}{\partial v_j(n)} \left(1+e^{-a v_j(n)}\right)$$

$$\phi'(v_j(n)) = \frac{a\, e^{-a v_j(n)}}{\{1+e^{-a v_j(n)}\}^2}$$

$$\phi'(v_j(n)) = \left\{\frac{a \cdot 1}{1+e^{-a v_j(n)}}\right\} \times \left\{\frac{e^{-a v_j(n)}}{1+e^{-a v_j(n)}}\right\}$$

from $\star$

$$\phi'(v_j(n)) = a\left\{\phi(v_j(n))\right\}\left\{\frac{e^{-a v_j(n)}}{1+e^{-a v_j(n)}}\right\}$$

Adding & subtracting 1.

$$\phi'(v_j(n)) = a\left\{\phi v_j(n)\right\}\left\{1+\frac{e^{-a v_j(n)}}{1+e^{-a v_j(n)}} - 1\right\}$$

LCM

$$\phi'(v_j(n)) = a\left[\phi(v_j(n))\right]\left[1+\left[\frac{e^{-a v_j(n)} - 1 - e^{-a v_j(n)}}{1+e^{-a v_j(n)}}\right]\right]$$

$$\phi'(v_j(n)) = a\,\phi(v_j(n))\left[1 - \frac{1}{1+e^{-a v_j(n)}}\right]$$

$$\therefore \phi'(v_j(n)) = a\,\phi(v_j(n))\left[1 - \phi(v_j(n))\right]$$

——— **

$$\phi'(v_j(n)) = a\,y_0(1-y_0)$$ ——— ***

② Tan hyperbolic: [Ranges from +1 to −1]

$$y_0 = \phi(v_j(n)) = a\tanh(b\,v_j) \quad \text{——} ①$$

$$\phi'(v_j(n)) = ab\,\sec^2 h\,(b\,v_j)$$

$$\phi'(v_j(n)) = ab\,(1 - \tan^2 h\,(b\,v_j))$$

Multiply & divide by a.

$$\phi'(v_j(n)) = \frac{a^2 b}{a}\,(1 - \tan^2 h(b\,v_j))$$

$$\phi'(v_j(n)) = \frac{b}{a}\,(a^2 - a^2 \tan^2 h(b\,v_j))$$

It is of the form
$(a^2 - b^2) = (a+b)(a-b)$

$$\phi'(v_j(n)) = \frac{b}{a}\left[(a + a\tanh(b\,v_j))\,(a - a\tanh(b\,v_j))\right]$$

from ①

$$\phi'(v_j(n)) = \frac{b}{a}\left[\{a + \phi(v_j)\}\,\{a - \phi(v_j)\}\right]$$

$$\phi'(v_j(n)) = \frac{b}{a}\left[(a+y_0)(a-y_0)\right]$$ ——— *

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad \{e^{-\infty} = 0\} \quad \{e^{\infty} = -1\}$$

$x \to \infty \implies \tanh x \to 1$

$x \to -\infty \implies \tanh x \to -1$

Conclusion: → forward pass // → backward pass

Sigmoid activation function

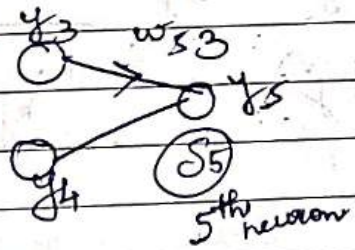
$$\Delta w_{ji} = \eta \delta_j y_i$$

$$\delta_j = \phi'(v_j(n))\, e_j(n)$$

$= y_o(1 - y_o)$ error ; j is output node.

$= y_{pred}(1 - y_{pred})(y_{desired} - y_{pred})$

Note —



$$\delta_5 = y_5(1 - y_5)(y_d - y_5)$$

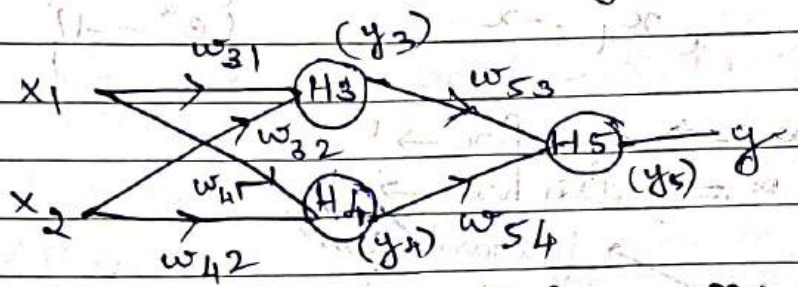
$$\Delta w_{53} = \eta \delta_5 y_3$$

$$\delta_j = \phi'(v_j(n)) \sum \delta_k w_{kj} \quad ; j \text{ is hidden node}$$

$$\delta_j = \phi(v_j)(1 - \phi(v_j)) \sum \delta_k w_{kj}$$

$$= y_o(1 - y_o) \sum \delta_k w_{kj}$$

Problem on back propagation:



$w_{31} = 0.1$ $w_{42} = 0.2$ $x_1 = 0.3$

$w_{32} = 0.4$ $w_{41} = 0.3$ $x_2 = 0.5$

$w_{53} = 0.2$ $w_{54} = 0.4$ desired o/p $y_t = 0.5$

$\eta = 1$

W.K.T. $w_{n+1} = w_n + \Delta w_{ji}$

and $\Delta w_{ji} = \eta \delta_j y_i$

where $\delta_j = y_j(1 - y_j)$ (error) ; j is o/p neuron ——— (1)

$\delta_j = y_j(1 - y_j) \sum \delta_k w_{kj}$ ——— (2) ; j is hidden neuron

Forward pass:

$H_3 = w_{31} x_1 + w_{32} x_2$

$= 0.1(0.3) + 0.4(0.5)$

$= 0.03 + 0.20$

$= 0.23$

$y_3 = \phi(H_3)$

$= \frac{1}{1+e^{-H_3}} = \frac{1}{1+e^{-0.23}} = 0.557$

$H_4 = w_{41} x_1 + w_{42} x_2$

$= 0.3(0.3) + 0.2(0.5)$

$= 0.09 + 0.1$

$= 0.19$

$$y_4 = \frac{1}{1+e^{-H_4}} = \frac{1}{1+e^{-0.19}} = 0.547$$

$$H_5 = y_3 w_{53} + y_4 w_{54}$$

$$= 0.557(0.2) + 0.547(0.4)$$

$$= 0.3302$$

$$y_5 = \frac{1}{1+e^{-H_5}} = \frac{1}{1+e^{-0.3302}} = 0.581$$

We know y_3, y_4, y_5, our o/p desired is 0.5 but we got 0.581 so we need to update the weights.

$$\text{error} = y_t - y_5 = 0.5 - 0.581$$

$$= -0.081$$

We have to minimise the error by finding local gradient & hence weight updation.

y_5 is output neuron

from (1) $\delta_5 = y_5(1-y_5)(\text{error})$

$$= 0.581\,(1-0.581)(-0.081)$$

$$= 0.581(0.419)(-0.081)$$

$$= -0.019$$

Similarly, calculate δ_3 and δ_4

H_3 and H_4 are hidden neurons.

from (2)

$$\delta_j = y_j(1-y_j)\sum \delta_k w_{kj}$$ [k is o/p neuron]

[o/p has one neuron, so no summation required]

$$\delta_3 = y_3(1-y_3)\delta_5 w_{53}$$

$$= 0.557\,(1-0.557)(-0.019)(0.2)$$

$$= -0.000937$$

$$\delta_4 = y_4(1-y_4)\,\delta_5\, w_{54}$$

$$= 0.547\,(1-0.547)(-0.019)(0.4)$$

$$= -0.00188$$

$$\therefore \Delta w_{ji} = \eta\,\delta_j\, y_i \;;\; w_{n+1} = w_n + \Delta w_{ji}$$

$$\Delta w_{53} = \eta\,\delta_5\, y_3$$

$$= 1(-0.019)\,0.557$$

$$= -0.0105$$

$$w_{53(new)} = w_{53(old)} + \Delta w_{53}$$

$$= 0.2 - 0.0105 = 0.1895$$

Similarly,

$$\Delta w_{54} = \eta\,\delta_5\, y_4$$

$$= 1 \times (-0.019)(0.547)$$

$$= -0.0103$$

$$w_{54(new)} = w_{54(old)} + \Delta w_{54}$$

$$= 0.4 - 0.0103$$

$$= 0.3897$$

In the same way, $\Delta w_{ji} = \eta\,\delta_j\, y_i$ → Previous layer

$$\Delta w_{31} = \eta\,\delta_3\, x_1$$

$$= 1\,(-0.000937)(0.3)$$

$$= -0.000281$$

$$w_{31(new)} = w_{31(old)} + \Delta w_{31}$$

$$= 0.1 - 0.000281$$

$$= 0.099720$$

$$\Delta w_{32} = \eta \delta_3 x_2$$
$$= 1 (-0.000937) 0.5$$
$$= -0.000468$$

$$w_{32(new)} = w_{32(old)} + \Delta w_{32}$$
$$= 0.4 - 0.000468 = 0.39953$$

$$\Delta w_{41} = \eta \delta_4 x_1$$
$$= 1 (-0.0018) (0.3)$$
$$= -0.000564$$

$$w_{41(new)} = w_{41(old)} + \Delta w_{41}$$
$$= 0.3 + (-0.000564)$$
$$= 0.29943$$

$$\Delta w_{42} = \eta \delta_4 x_2$$
$$= 1 (-0.00188) (0.5)$$
$$= -0.00094$$

$$(w_{42})_{new} = (w_{42})_{old} + \Delta w_{42}$$
$$= 0.2 + (-0.0094)$$
$$= 0.19905$$

Updated weights :

$w_{31} = 0.0997$ $w_{41} = 0.29943$ $w_{53} = 0.1895$

$w_{32} = 0.39953$ $w_{42} = 0.19905$ $w_{54} = 0.3897$

Note - The δ values decreases as we propagate back from output layers to hidden layers. At some point, the δ value might become zero. This is called vanishing gradient problem. Weight updation depends on

Δw_{ij}, which depends on δ, so it may not happen if gradients vanish.

More the deeper the layers become in ANN, we come across this problem.

Iteration 2 - forward pass:

$$H_3 = w_{31} x_1 + w_{32} x_2$$
$$0.0997\,(0.3) + 0.39953(0.5)$$
$$= 0.2296$$

$$y_3 = \phi(H_3) = \frac{1}{1+e^{-H_3}} = \frac{1}{1+e^{-0.2296}} = 0.557$$

$$H_4 = w_{41} x_1 + w_{42} x_2$$
$$= 0.29943(0.3) + 0.19905(0.5)$$
$$= 0.1893$$ [Same value because change in weight vector was small]

$$y_4 = \frac{1}{1+e^{-H_4}} = \frac{1}{1+e^{-0.1893}} = 0.547$$

$$H_5 = y_3 w_{53} + y_4 w_{54}$$
$$= 0.557(0.1895) + 0.547(0.3897)$$
$$= 0.3187$$

$$y_5 = \frac{1}{1+e^{-0.3187}} = 0.579$$

$$\text{error} = y_t - y_5$$
$$= 0.5 - 0.579$$
$$= -0.079$$

ich depends on δ, so it may not
adients vanish.

eeper the layers become in ANN,
across this problem.

- forward pass:

$x_1 + w_{32} x_2$
$(0.3) + 0.39953(0.5)$
96

$H_3) = \frac{1}{1+e^{-H_3}} = \frac{1}{1+e^{-0.2296}} = 0.557$

$x_1 + w_{42} x_2$

$^3(0.3) + 0.19905(0.5)$
93 [Same value because change in weight vector was small]

$\frac{}{e^{-H_4}} = \frac{1}{1+e^{-0.1893}} = 0.547$

$_{53} + y_4 w_{54}$
$+(0.1895) + 0.547(0.3897)$
87

$= 0.579$
$e^{-0.3187}$

$- y_5$
$- 0.579$
0.079

In XOR gate, forward & backward pass for all the possible combinations, then move on to next iteration.

$\delta_5 = y_5(1-y_5)$ error

$= 0.579\ (1-0.579)(+0.079)$
$= -0.0192$

$\delta_3 = y_3(1-y_3)\ \delta_5\ w_{53}$

$= 0.557\ (1-0.557)(-0.0192)(0.1895)$
$= 0.557\ (0.443)(-0.0192)(0.1895)$
$= -0.0008977$

$\delta_4 = y_4(1-y_4)\ \delta_5 w_{54}$

$= 0.547\ (1-0.547)(-0.0192)(0.3897)$
$= 0.547\ (-0.453)(-0.0192)(0.3897)$
$= -0.001854$

$\Delta w_{53} = \eta\ \delta_5 y_3$
$= -0.0192\ (0.557)$
$= -0.0106944$

$w_{53(new)} = w_{53(old)} + \Delta w_{53}$
$= 0.1895 - 0.0106944$
$= 0.1788056$

$\Delta w_{54} = \eta\ \delta_5 y_4$
$= -0.0192\ (0.547)$
$= -0.0105024$

$$w_{54}(new) = w_{54}(old) + \Delta w_{54}$$

$$= 0.3897 - 0.0105024$$

$$= 0.3791976$$

$$\Delta w_{31} = \eta \delta_3 x_1$$

$$= (-0.0008977)(0.3)$$

$$= -0.00026931$$

$$w_{31}(new) = w_{31}(old) + \Delta w_{31}$$

$$= 0.0997 - 0.00026931$$

$$= 0.09943069$$

$$\Delta w_{32} = \eta \delta_3 x_2$$

$$= (-0.0008977)(0.5)$$

$$= -0.00044885$$

$$w_{32}(new) = w_{32}(old) + \Delta w_{32}$$

$$= 0.39953 - 0.00044885$$

$$= 0.39908115$$

$$\Delta w_{41} = \eta \delta_4 x_1$$

$$= (-0.001854)(0.3)$$

$$= -0.0005562$$

$$w_{41}(new) = w_{41}(old) + \Delta w_{41}$$

$$= 0.29943 - 0.0005562$$

$$= 0.2988738$$

$$\Delta w_{42} = \eta \delta_4 x_2$$

$$= (-0.001854)(0.5)$$

$$= -0.000927$$

$$w_{42(new)} = w_{42(old)} + \Delta w_{42}$$

$$= 0.19905 - 0.000927$$

$$= 0.198123$$

Updated weights:-

$w_{31} = 0.09943069$ $w_{41} = 0.2988738$

$w_{32} = 0.39908115$ $w_{42} = 0.198123$

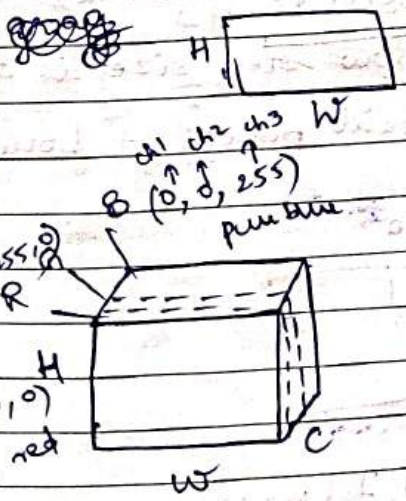
$w_{53} = 0.1788056$ $w_{54} = 0.3791976$

ANN - classifier not feature extractor

Convolution - feature extraction

Convolutional Neural networks:

A signal is 1D representation and image is 2D representation. At every time stamp, there'll be amplitudes. Images are grayscale (1 channel) & RGB (3 channels).



gray H x W x 1

RGB H x W x C → C = 3 (R, G, B)

Each grayscale image has grids with pixels of different intensities.

grayscale → 0 to 255

black (0) white (255)

For colour images, three such channels exist with values for each pixel ranging from 0 to 255.

e.g.: (10, 200, 68)

H W 3 channels

32 × 32 × 3 (Height & width will be same across all channels)

Feature map:-

We define a Kernel / filter which will run through the entire image matrix. This creates a feature map. This is the process of feature extraction.

3×3, 5×5 or 7×7 sized kernels are used in convolutions. 7×7 has larger field of view comparitively but it is computationally expensive.

! Why can't we use 2×2 kernel?

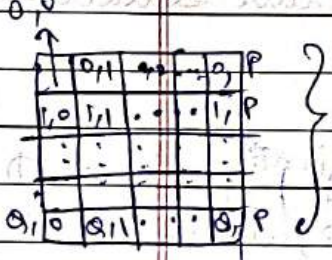
Let $I(i, j)$ → input image

$K(m, n)$ → Kernel / filter

Convolution: $O(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m, n) \cdot I(i+m, j+n)$

output image at pixel (i, j)

where $K(m, n)$ is a Kernel with dimension $M \times N$.



generalised representation for image & kernel

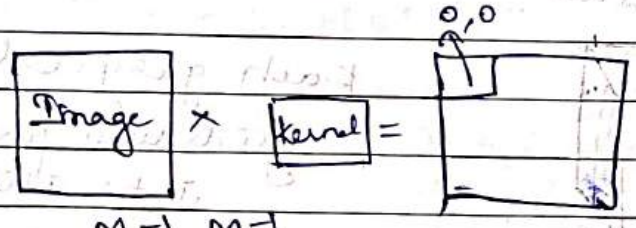
If it is image → Q = 32, P = 32 etc.

If it is Kernel → Q = 3, P = 3.

$I(i+m, j+n)$ is for stride (movement)

how step size is taken

Padding is used to avoid pixels at boundaries when Kernel is used.

Ex- 

$$O(0,0) = \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} K(m, n)\, I(m, n)$$

Assume kernel size = 3×3

M = 3, N = 3

$$O(0,0) = \sum_{m=0}^{2} \sum_{n=0}^{2} K(m,n)\, I(m,n)$$

$m=0 \Rightarrow K(0,0)\, I(0,0) + K(0,1)\, I(0,1) + K(0,2)\, I(0,2)$

$m=1 \Rightarrow + K(1,0)\, I(1,0) + K(1,1)\, I(1,1) + K(1,2)\, I(1,2)$

$m=2 \Rightarrow + K(2,0) I(2,0) + K(2,1)\, I(2,1) + K(2,2)\, I(2,2)$

Consider —

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

×

Q	R	S
T	U	V
W	X	Y

$K(m,n)$ → $M = 3$, $N = 3$

$O(0,0) = aQ + bR + cS + eT + fU + gV + iW + jX + kY$

(element-wise multiplication)

Similarly, $O(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n)\, I(i+m, j+n)$

$i=0, j=1$

$O(0,1) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n)\, I(m, 1+n)$

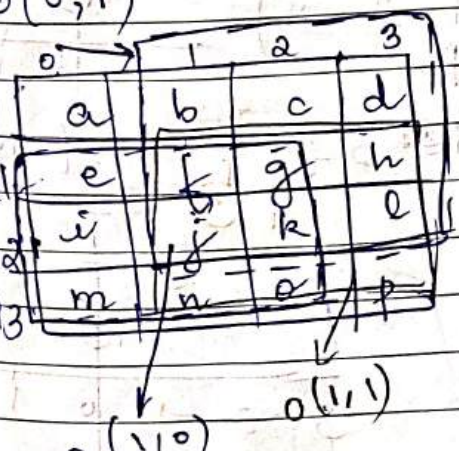
$m=0$ to 2, $n=0$ to 2

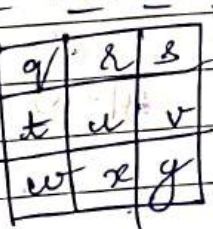
	0	1	2
0	O(0,0)	O(0,1)	
1	O(1,0)	O(1,1)	
2			

2×2

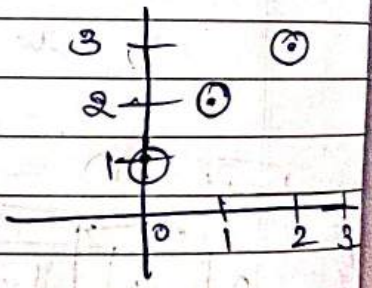
no space downwards or towards the right

(convolution)



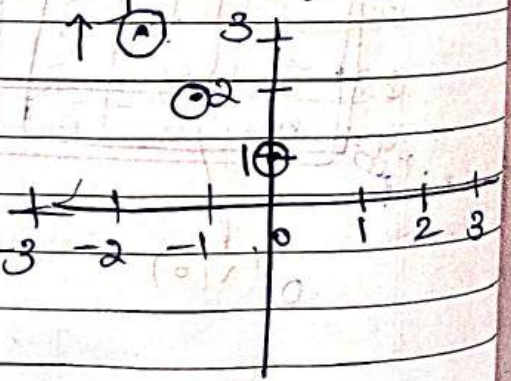


```
import numpy as np
def sigmoid(self, x):
    return 1/(1+np.exp(-x))
def sigmoid_der(x):
    return x*(1-x)

class NN:
    def __init__(self, inputs):
        self.inputs = inputs
        self.l = len(self.inputs)

class Perceptron:
    def __init__(self, learning_rate = 0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = self.sigmoid
        self.weights = None
        self.bias = None
```



$x[n] = [1 \;\; 2 \;\; 3]$

$x[-n] = [3 \;\; 2 \;\; 1]$



Ex- Image

$$O(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} K(m,n)\, I(m,n)$$

Assume kernel size = 3×3

M = 3, N =>

When an image is convolved with kernel, there is reduction in size. $4 \times 4^{\times 1}$ image $*$ $3 \times 3^{\times 1}$ kernel $\longrightarrow$ $2 \times 2 \times 1$ image.

This was the case for grayscale image (channel = 1).

$H \times W \times 1$

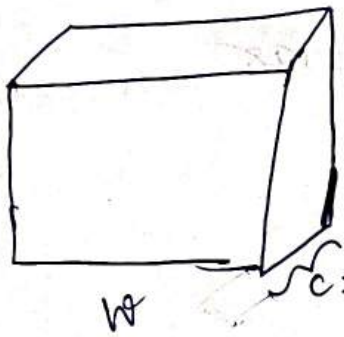
channel no. $\longrightarrow$ tensor.

For RGB channels

$I(H \times W \times C) * K(m \times n \times c) \Longrightarrow O(H_1 \times W_1 \times k)$

↳ change in no. of channels

RGB



dimensions

$I\ (H, W, 3)$

$\hookrightarrow k(m, n, 3)$

↳ no. of channels

Depth of kernel is decided by preceding layer

Convolution process is the same, but it happens for the three channels individualy & there will be a stride. Three feature maps will be obtained, upon which element wise addition occurs.

3×3 image

a_1	b_1	c_1
d_1	e_1	f_1
g_1	h_1	i_1

R

a_2	b_2	c_2
d_2	e_2	f_2
g_2	h_2	i_2

G

a_3	b_3	c_3
d_3	e_3	f_3
g_3	h_3	i_3

B

If I/p layer has "64 channels", depth of kernel also has 64 ~~channels~~

3×3 kernel | 1 Kernel with depth 3 → 1 feature map

m_1	n_1	o_1
p_1	q_1	r_1
s_1	t_1	u_1

m_2	n_2	o_2
p_2	q_2	r_2
s_2	t_2	u_2

m_3	n_3	o_3
p_3	q_3	r_3
s_3	t_3	u_3

V_1	W_1	

V_2	W_2	

V_3	W_3	

=

$V_1+V_2+V_3$ $W_1+W_2+W_3$

Same kernel cannot multiply with different channels because of different weights

To increase the no. of feature maps —
we consider more no. of kernels

one kernel (3, 3, 3) × 64 ⟶ 64 different feature maps.

(They have different combination of weights)

Consider —

Image = (H × W × C)

① convolution → (3,3) × 64 ⟶ H × W × 64

depth → C

Convolution → (3,3) × 128 ⟶ H × W × 128

depth = 64 (of previous layer)

No. of kernels is user defined.

Convolution (3, 3) × 256 ⟶ H × W × 256

depth = 128

3 × 3 × 128

Total channels = 128 × 256

①

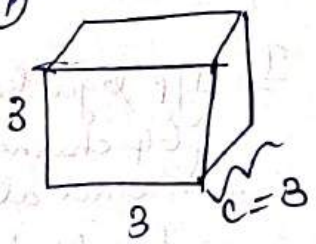


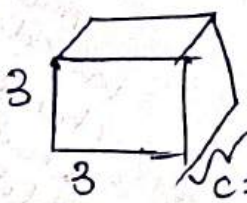
Image → H × W × 3 ... (64 kernels)

one image → multiplied with 64 kernels

↓

64 channels //

②



I/P image 8 × 8 × 64 ... {128 kernels}

Image → multiplied with 128 kernels

↓

128 channels //

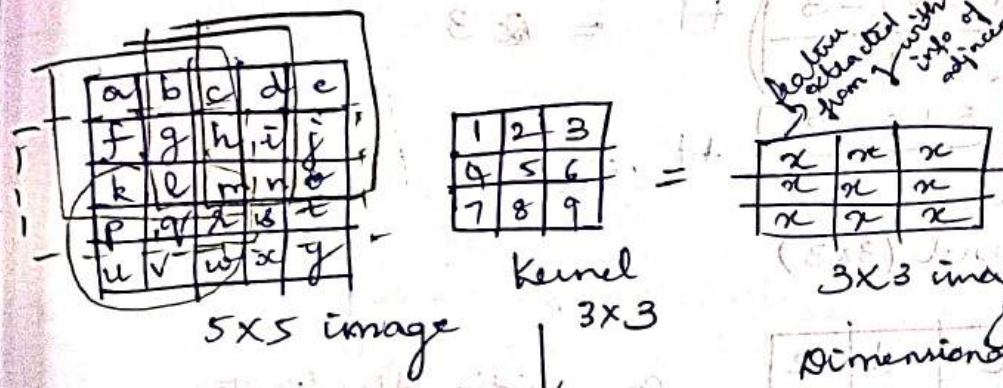
$$o(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n)\, \underbrace{I(i+m,\ j+n)}_{\text{Stride}\,(i+m,\ j+n)} \quad \{\text{stride} = 1\}$$

For a 3 × 3 kernel, stride → 1 ✓
2 ✓
3 ✓
} In general, we limit by stride 1 or 2.

By including stride,

$$o(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n)\, I(i \cdot s + m,\ j \cdot s + n)$$

(s = stride)



5×5 image

Kernel 3×3

3×3 image.

has a central pixel ⑤

Dimensionality of image reduced because it's entirely not concentrating on all pixels.

There are 4 diagonal & adjacent pixels

So, when kernel is multiplied entire concentration is on pixel g, because adjacent pictures don't vary much in information.

Moving by one step, 5 overlaps with h, similarly i is central pixel, so on

We are losing information of edge pixels hence dimensionality reduced.

How to concentrate on edge pixels? → Padding

Let's consider — Original Image : (H, W, C)

Kernel : (K × K)

output Image : (H_1, W_1, D_1)

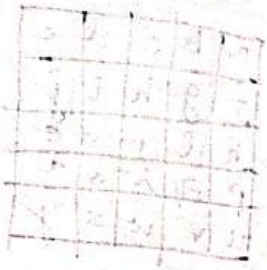
where, $H_1 = \left(\frac{H + 2P - K}{S}\right) + 1$ P → Padding, S → stride

$W_1 = \left(\frac{W + 2P - K}{S}\right) + 1$

D_1 = No. of kernels

(5×5×C Image) (3×3 kernel) $H_1 = \left[\frac{5 + 2(0) - 3}{1}\right] + 1 = 3$

$W_1 = \left[\frac{5 + 0 - 3}{1}\right] + 1 = 3$



stride → 2 kernel (3×3)

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

Convolution disadvantage —

1) less field of view [Not covering entire kernel in a single stride]

original image : 224, 224, 3

kernel → 3 × 3

stride → 1

padding → 0

$H_1 = \left(\frac{224 - 3}{1}\right) + 1 = 222$

$W_1 = \left(\frac{224 - 3}{1}\right) + 1 = 222$

o/p → 222 × 222 × 1

...w ? → Padding
...e pixels

...al Image : (H, W, C)
...l : (K × K)
...mage : (H_1, W_1, D_1)

$H_1 = \left(\frac{H + 2P - K}{S}\right) + 1$ P → Padding, S → stride

$W_1 = \left(\frac{W + 2P - K}{S}\right) + 1$

D_1 = No. of kernels

$\left[\frac{(0) - 3}{1}\right] + 1 = 3$

$\left[\frac{0 - 3}{1}\right] + 1 = 3$

(3×3)

d	e
	i
	o
s	t
	y

Convolution disadvantage –

1) less field of view [Not covering entire kernel in a single stretch]

3

$\left(\frac{224 - 3}{1}\right) + 1 = 222$

$\left(\frac{224 - 3}{1}\right) + 1 = 222$

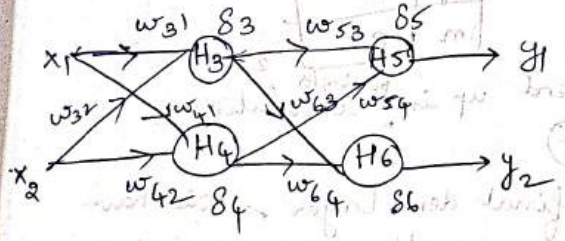
o/p → 222 × 222 × 1

When stride is increased from 1 to 2, image dimension reduced to half (lesser computations)

Padding → Valid padding
Padding → Same padding

No. of pixels to be added/padded depends on kernel size.
i.e., $\frac{k-1}{2}$ to prevent dimensionality reduction of image; where k is kernel dimension.

Convolution → flip the kernel & correlate it;
↓
DL convolution is just correlation!

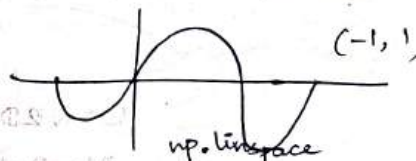


x_1	x_2	y_1	y_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Lab 3

Activation function
last : linear
intermediate : tanh

import dense layer from keras



np.linspace
$x = (-1, 1, 100)$ → samples
$y = 1 + \sin 2\pi x$

Train for 500, 1000, 15000 epochs

Pooling < average / max

- Reducing image size.

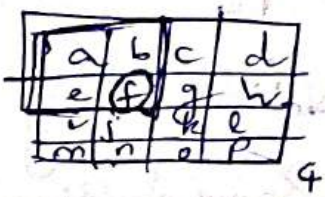
224 x 224 x 3

Conv2D (64, 3 x 3,)

MaxPool (2x2)

↓

every 4 pixels → 1 pixel



4 x 4

Avg pool

$\frac{a+b+e+f}{4}$	$\frac{c+d+g+h}{4}$
$\frac{i+j+m+n}{4}$	$\frac{k+l+o+p}{4}$

2 x 2

OR

Maxpool

f	h
n	p

2 x 2

Least possible size to end up in pooling for convolution is (7x7)

dense layer ↓ only for classification

The final dense Layer should have no. of neurons = no. of classes

Conv2D ( );

MaxPool;

flatten / global average pooling

Work on MNIST / Cifar 10

1. Download dataset from keras
2. split into x_train, y_train, x_test, y_test
3. Use categorical-cross entropy — one hot encoding
3. Use tocategorical to perform label encoding — one hot encoding

not required for sparse categorical cross entropy

Consider		R_0	R_1	R_2	R_3	
0	Image 1	1	0	0	0	$\in$ class 0
1	Image 2	0	1	0	0	$\in$ class 1
2	Image 3	0	0	1	0	$\in$ class 2
3	Image 4	0	0	0	1	$\in$ class 3

4. Normalise the image

i) Use dense layers directly & classify the images by reshape [flatten] without feature extraction.

Dense

⋮

Dense(10)

check accuracy & (model.predict) & print some images.

ii) Use convolutional layer, dense layer, maxpool for classifications.

Padding

2 types:

1. Valid padding - No pixels are pad
2. Same padding - Pixels are pad such that image size is preserved.

MSE good choice for regression (not classification)

Batch normalisation - used to maintain zero average and unit variance.

Average Pooling: Reduce the image size by half by averaging pixel values.

Maximum Pooling: Take the maximum value from a pixel range.

Global average pooling:

Binary cross entropy :-

MSE :- $J = \frac{1}{2} \sum_j (y_j - \hat{y}_j)^2$

y_j → desired, $\hat{y}_j$ → predicted

In binary classification :-

Desired values → 0 and 1

Predicted values → range from $[0, 1)$

$\hat{y} > 0.5$ (class 1)

$\hat{y} < 0.5$ (class 0)

Sigmoid activation function brings o/p in range $[0, 1)$.

Loss J :

① In case of correct classification, (y_i & $\hat{y}_i$ are same) then $J = 0$.

② For misclassification, loss J must be very high.

↓

However, in case of MSE

$$J = \frac{1}{2} \sum_j (y_j - \hat{y}_j)^2$$

Let $y_j = 0$, and obtained $\hat{y}_j = 0.99$

$\hat{y}_j = 1$

} then the penalising of loss is not much (Not much different)

③ Always $J \geq 0$ (Always true for MSE).

Binary cross entropy $BCE = -[y \ln \hat{y} + (1-y) \ln (1-\hat{y})]$

$y \rightarrow$ desired

$\hat{y} \rightarrow$ predicted

Case (i) $y=0 ; \hat{y}=0$ / $y=1 ; \hat{y}=1$ [Correct classification]

$BCE = -[0 + \ln 1]$

$= -\ln 1$

$= 0$

$y=1 \quad \hat{y}=1$

$= -[\ln 1 + 0]$

$= -\ln 1$

$= 0$ ✓

Case (ii) $y=0 ; \hat{y} \simeq 1$

$BCE = -[0 \ln 1 + (1-0) \ln (1-1)]$

$= -[0 + \ln 0]$

$= -[-\infty]$

$= \infty$

$y=0 ; \hat{y}=0.99$

$BCE = -[0 \ln 0.99 + (1-0) \ln (1-0.99)]$

$= -[\ln 0.01] = -[-4.63]$

$= 4.63$

Case (iii) clearly visible from (ii)

$[y \ln \hat{y} + (1-y) \ln (1-\hat{y})]$

$y \rightarrow$ desired

$\hat{y} \rightarrow$ predicted

$y=1 \; ; \; \hat{y}=1$ [Correct classification]

$y=1 \quad \hat{y}=1$

$= -[\ln 1 + 0]$

$= -\ln 1$

$= 0$ ✓

$+ (1-0) \ln (1-1)]$

$\ln 0]$

$0.99 + (1-0) \ln (1-0.99)]$

$0.01] \quad = -[-4.63]$

$= 4.63$

from (ii)

for N examples:

$$BCE = \frac{1}{N} \sum_{i=1}^{N} [y_i \ln \hat{y}_i + (1-y_i) \ln (1-\hat{y}_i)]$$

BCE is the modification of categorical cross entropy.

Categorical cross entropy : (Multi-class classification)
[Softmax activation function]

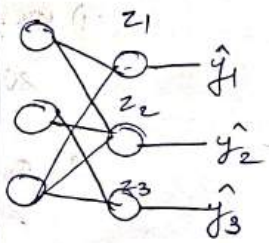
$$J = -\sum_{i=1}^{n} y_i \ln \hat{y}_i \; ; \; n \rightarrow \text{No. of classes}$$

If $n = 2$

$$= -[y_1 \ln \hat{y}_1 + y_2 \ln \hat{y}_2]$$

$y_2 = (1-y_1)$ $\hat{y}_2 = (1-\hat{y}_2)$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$



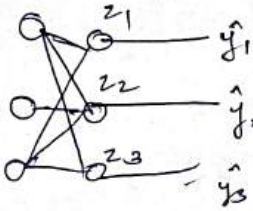
$$\hat{y}_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\hat{y}_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\hat{y}_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$\hat{y}_1 + \hat{y}_2 + \hat{y}_3 = 1$

Sigmoid ✓✓ (scores for two classes positive in case of multilabel classification)



let's say

$\hat{y}_1 = [0,1]$ 0.7 ✓✓

$\hat{y}_2 = [0,1]$ 0.2

$\hat{y}_3 = [0,1]$ 0.9 ✓✓

Summation > 1

For binary classification, we use sigmoid function & not softmax, we can use softmax too but we are increasing the complexity (no. of parameters) of network by additional neuron

Dense (1)
↓
sigmoid with BCE

but for softmax along with CCE, we needs to calculate probabilities & hence need 2 neurons.

Dense (2)

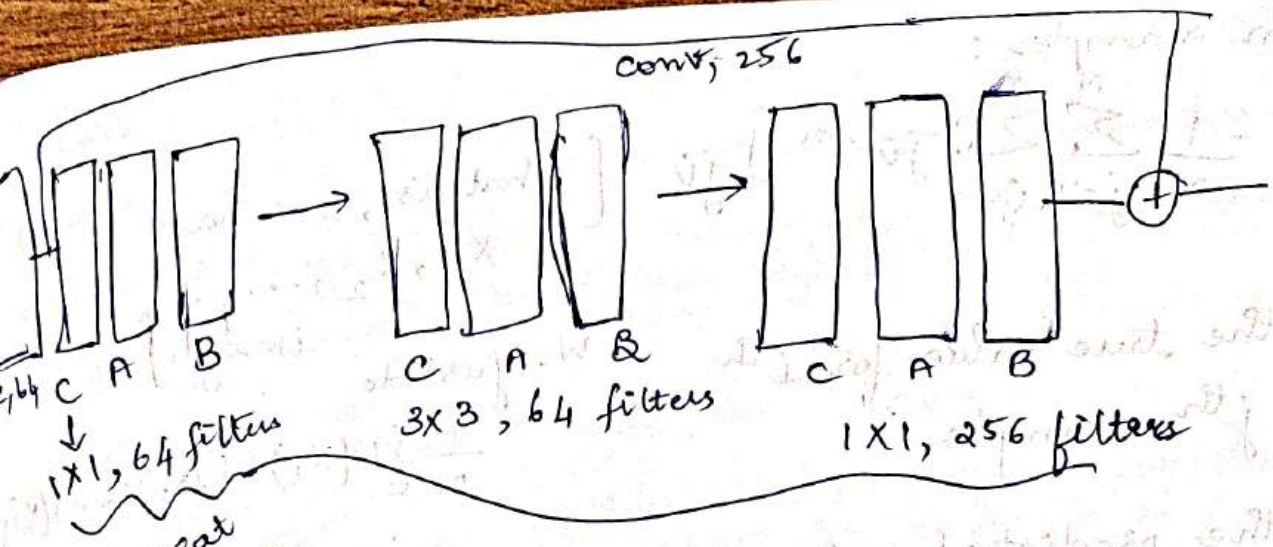
Lab Practicals

Exp-1 MNIST dataset has 10 classes, make it to 2 classes. (y=0, y=1)

1) In the last layer, use dense (1) with sigmoid activation function & loss as binary cross entropy.
2) In the last layer, use dense (2) with softmax activation function & loss as categorical cross entropy.

Exp-2: Develop a 50 layer residual network with 3X3, 64, 1 kernel size, stride kernel — CIFAR10 dataset

I/P Layer
Conv (3x3, 64, 1)
OR
Conv2D (64, 3, 1)



Conv2D (kernel size = 1, filter = filter, stride = 1)
Conv2D (kernel size = 3, filter = filter, stride = 1)
Conv2D (kernel size = 1, filter = filter, stride = 1)
} × 3

Conv2D (kernel size = 1

Deep Residual network for image recognition

Image → X ← cat / dog / horse

$y_i = [0 \ 1 \ 0]$ Image is of dog.

$\hat{y}_i = [0.1 \ 0.2 \ 0.7]$ But model predicted is as horse.

So, we are calculating categorical cross entropy.

$$L = -\sum_{i=1}^{n} y_i \ln \hat{y}_i$$

$$\therefore L = -[(0 \times \ln 0.1) + (1 \times \ln 0.2) + \{0 \times \ln(0.3)\}]$$

$$= 1.6$$

For n examples:

$$L = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{n} y_{ji} \ln \hat{y}_{ji}$$

[That is, we have $X_1, X_2, \ldots, X_n$ images]

y_{ji} is the true value for i^{th} class in j^{th} example.

We just do $\frac{1}{N}\{L(X_1)+L(X_2)+\ldots+L(X_N)\}$

$\hat{y}_{ji}$ is the predicted value for i^{th} class in j^{th} example.

Batch Normalization :

Normalization :- Since images have pixels in the range of [0, 255], we divide each pixel by 255 so that the pixels can be brought in the range of [0, 1].

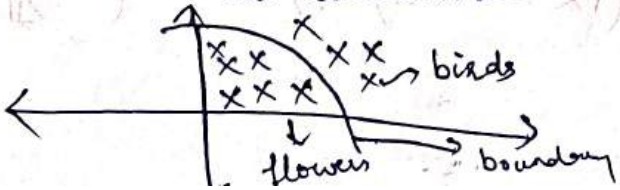
While images are being trained, they will be passed as batches to neural network.

- Consider that 100 flowers & birds have been split into 2 batches.

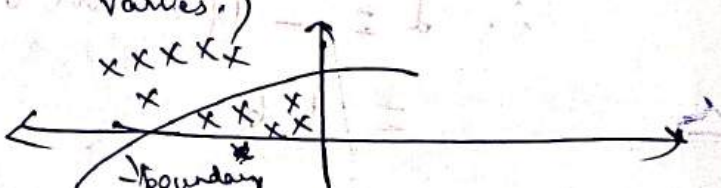
Batch 1
red rose } all red flowers
hibiscus }
green parrot
:

Batch 2
color parrot
white rose
yellow sunflower (& other yellow flowers)

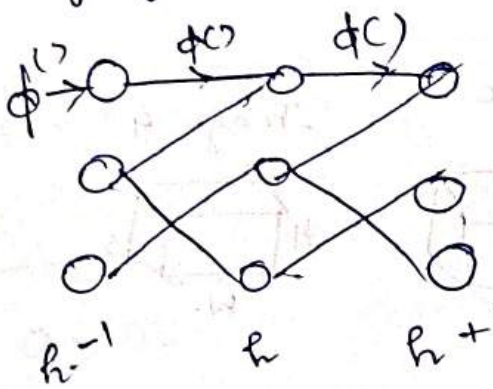
For batch 1 images, distribution —



(due to colour variation, distribution varies.)



Since samples are distributing at different places & different boundaries, model will take so much time to learn the patterns, therefore batch normalization is very much significant. (for convolutional layers)



For dense layers :-

while backpropagation, weights are updated for each layer, so. weights might differ for every layer, so when passing batch of images without normalization, there will be difference in distribution.

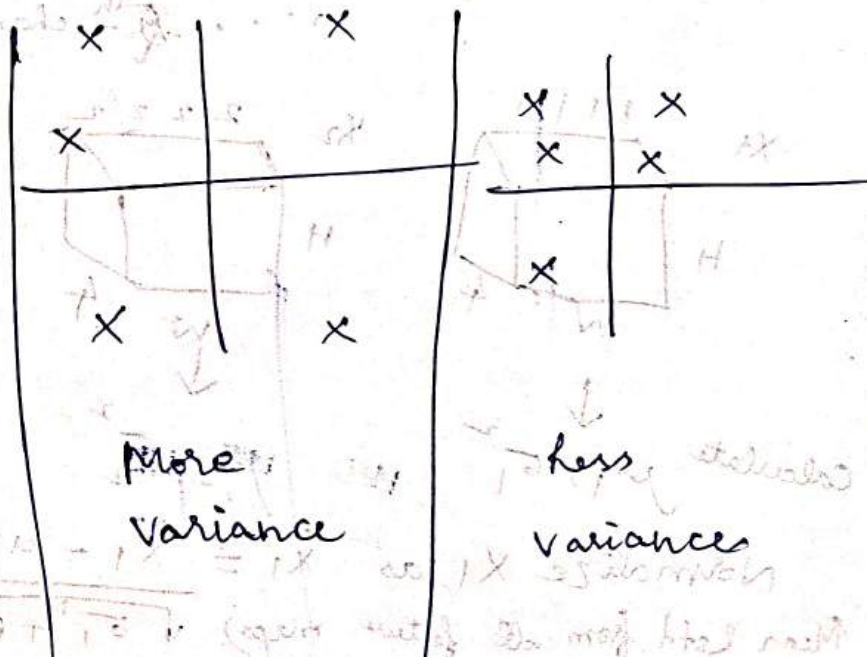
The shift in the distribution for same inputs due to multiplication with different weights for each layer is called covariance shift.

In order to suppress covariant shift, we use batch normalization where we strive to make mean zero & unit variance.

↓

(All samples will get distributed at origin)

Variance → Spread of data samples



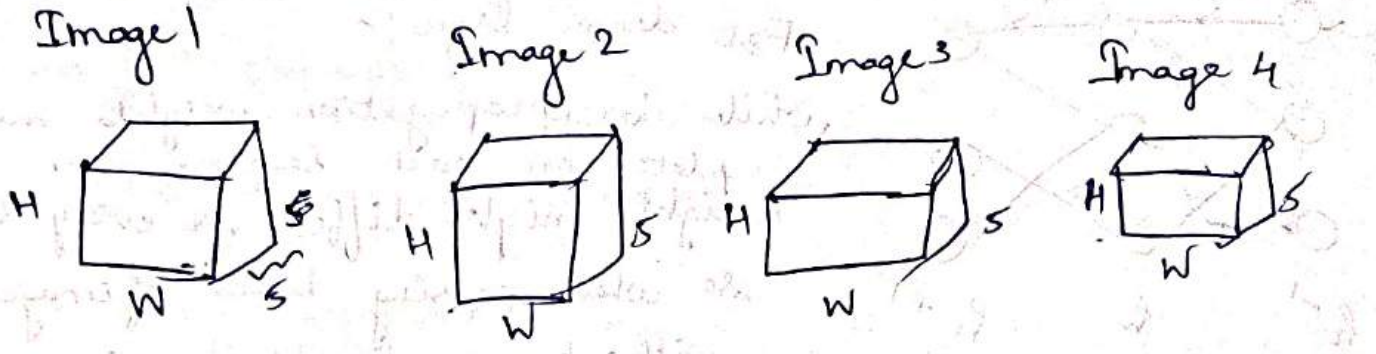
Consider:-

(H×W) Image 1 Image 2 Image 3 Image 4

Batch size = 4

No. of kernels = 5

o/p

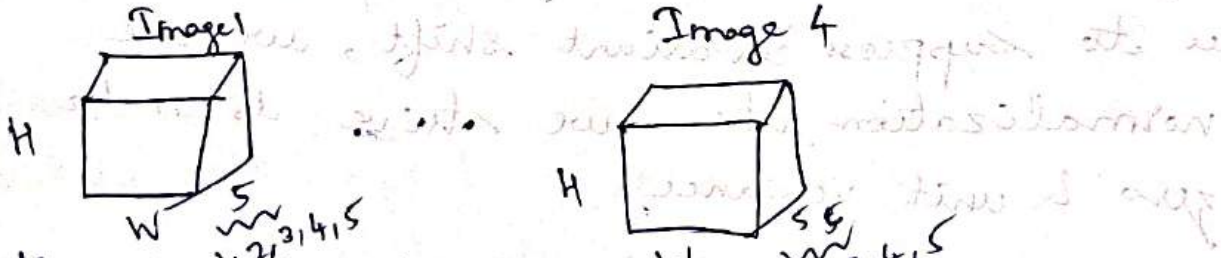


H×W×5 channels (each feature map has 5 channels → due to 5 kernels)

$B \times H \times W \times C$

That is, $X \in \mathbb{R}^{4 \times H \times W \times 5}$

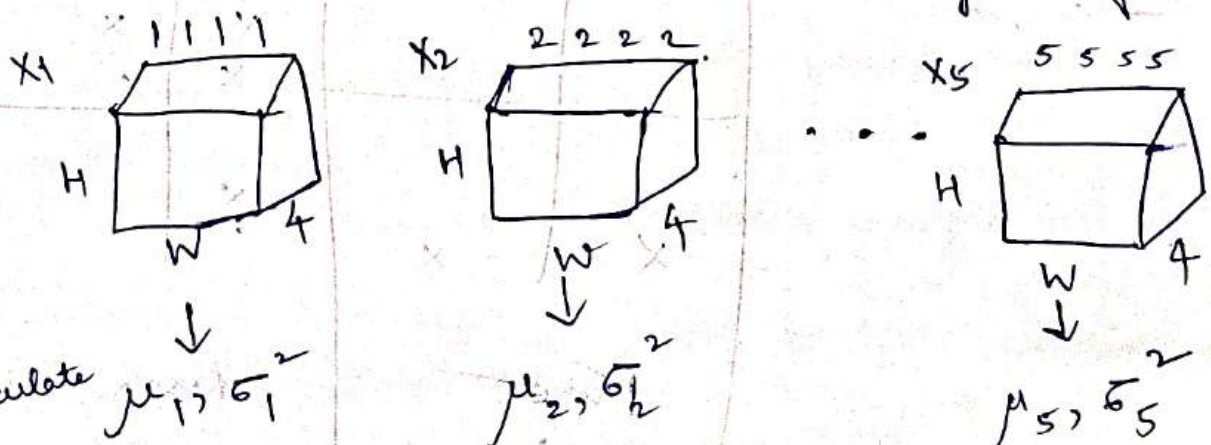
$\rightarrow X \in \mathbb{R}^{4 \times H \times W \times 5}$



Separation channelwise:- (Bring together respective feature maps)

Bring all the 1st channels together (which is due to kernel 1)

... 5th channels together from every image,



Calculate μ_1, σ_1^2 μ_2, σ_2^2 μ_5, σ_5^2

Normalize X_1 as $\hat{X}_1 = \frac{X_1 - \mu_1}{\sqrt{\sigma_1^2 + \epsilon}}$ & do for all images.

(Mean & std from all feature maps) [& for every pixel

e3 Image 4

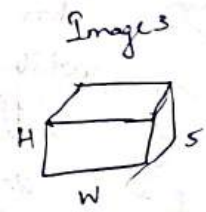


Image 3 Image 4

e map has 5 channels → due to 5 kernels)

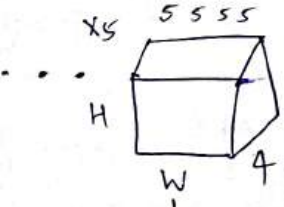
HxWxC

WX5

age 4

1, 2, 3, 4, 5
respective feature maps)
ogether (which is due to kernel 1)

els together from every image,



μ_5, σ_5^2

& do for all images.
[& for every pixel]

where $\epsilon = 10^{-5}$ because if in case standard deviation becomes zero, $\hat{x}_i \to \infty$ to avoid it, ϵ is added.

This process is called batch normalization.

Same for both dense layers & convolution layers.

1. obtain μ, σ^2 for all feature maps.
2. Do $\hat{x}$ for all pixels

Consider two channels of images -

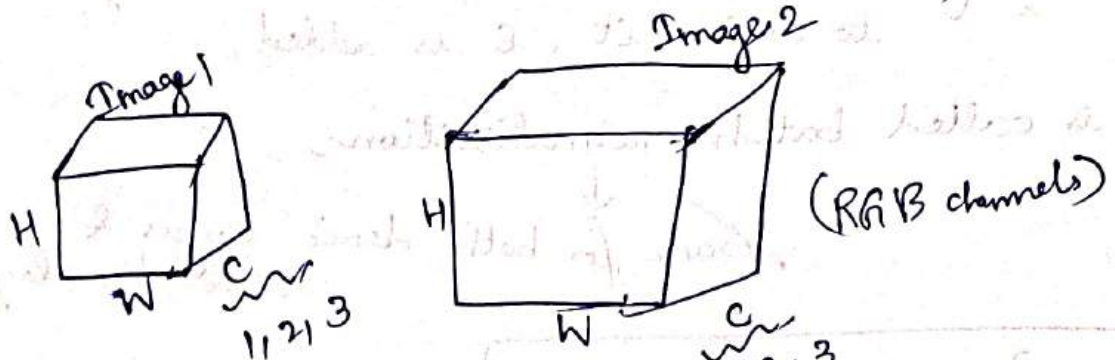
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$ batch size = 2 images each of 2X2X2

$$\mu = \frac{1+2+3+4+5+6+7+8}{8} = 4.5$$

$$\sigma^2 = \frac{(1-\mu)^2 + (2-\mu)^2 + (3-\mu)^2 + \cdots + (8-\mu)^2}{8}$$

$$\begin{bmatrix} \frac{1-\mu}{\sqrt{\sigma^2+\epsilon}} & \frac{2-\mu}{\sqrt{\sigma^2+\epsilon}} \\ \frac{3-\mu}{\sqrt{\sigma^2+\epsilon}} & \frac{4-\mu}{\sqrt{\sigma^2+\epsilon}} \end{bmatrix} \begin{bmatrix} \frac{5-\mu}{\sqrt{\sigma^2+\epsilon}} & \frac{6-\mu}{\sqrt{\sigma^2+\epsilon}} \\ \frac{7-\mu}{\sqrt{\sigma^2+\epsilon}} & \frac{8-\mu}{\sqrt{\sigma^2+\epsilon}} \end{bmatrix}$$

Batch normalisation



Bringing together of channels :-

(1,1) (2,2) (3,3)

Calculate μ_1, σ_1^2 μ_2, σ_2^2 μ_3, σ_3^2

Perform normalisation on every pixel of respective feature map.

$$\hat{X} = \frac{X - \mu_1}{\sqrt{\sigma_1^2 + \epsilon_1}}$$

Let $X \in R^{B \times H \times W \times C}$

B → total no. of images before dividing into one particular batch

N → no. of images in one batch

H → height

W → width

C → channels

$$\text{Mean} = \mu_c = \frac{1}{NBHW} \sum_{i=1}^{NB} \sum_{j=1}^{H} \sum_{k=1}^{W} x_{ijck}$$

Consider :- for channel 1 → 2 feature maps

1	2
3	4

5	6
7	8

$$\mu_c = \frac{1+2+3+4+5+6+7+8}{2 \times 2 \times 2}$$

$$\mu_c = \frac{1}{HW} \sum_{j=1}^{H} \sum_{k=1}^{W} x_{ijck} \quad \text{(Global average pooling)}$$

$$\sigma_c^2 = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} (x_i - \mu_c)^2$$

Normalization $\equiv \hat{x}_i = \frac{x_i - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$

Example:

Image 1: channel 1 $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

channel 2 $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

Image 2: channel 1 $\begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix}$

channel 2 $\begin{bmatrix} 13 & 14 \\ 15 & 16 \end{bmatrix}$

$$X = \begin{bmatrix} \text{Image 1} \begin{bmatrix} \underset{\text{channel 1}}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} & \underset{\text{channel 2}}{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}} \end{bmatrix} \\ \text{Image 2} \begin{bmatrix} \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} & \begin{bmatrix} 13 & 14 \\ 15 & 16 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\mu_1 = \frac{1+2+3+4+9+10+11+12}{2 \times 2 \times 2}$$

$$= \frac{52}{8} = \frac{26}{4} = \frac{13}{2} = 6.5$$

$$\sigma_1^2 = (1-6.5)^2 + (2-6.5)^2 + (3-6.5)^2 + (4-6.5)^2 + (9-6.5)^2 + (10-6.5)^2 + (11-6.5)^2 + (12-6.5)^2 / 8$$

$$\sqrt{\sigma_1^2} = \sqrt{17.25} = 4.815$$

$$\mu_2 = \frac{5+6+7+8+13+14+15+16}{8}$$

$$= 10.5$$

$$\sigma_2^2 = \frac{(5-10.5)^2 + (6-10.5)^2 + (7-10.5)^2 + (8-10.5)^2 + (13-10.5)^2 + (14-10.5)^2 + (15-10.5)^2 + (16-10.5)^2}{8}$$

$$\sigma_2^2 = 17.25$$

$$\sqrt{\sigma_2^2} = 4.15$$

$$\hat{X} = \begin{bmatrix} \begin{bmatrix} \frac{1-6.5}{4.15} & \frac{2-6.5}{4.15} \\ \frac{3-6.5}{4.15} & \frac{4-6.5}{4.15} \end{bmatrix} & \begin{bmatrix} \frac{5-10.5}{4.15} & \frac{6-10.5}{4.15} \\ \frac{7-10.5}{4.15} & \frac{8-10.5}{4.15} \end{bmatrix} \\ \begin{bmatrix} \frac{9-6.5}{4.15} & \frac{10-6.5}{4.15} \\ \frac{11-6.5}{4.15} & \frac{12-6.5}{4.15} \end{bmatrix} & \begin{bmatrix} \frac{13-10.5}{4.15} & \frac{14-10.5}{4.15} \\ \frac{15-10.5}{4.15} & \frac{16-10.5}{4.15} \end{bmatrix} \end{bmatrix}$$

ch1 ↓ ch2 ↓

$$\hat{X} = \begin{bmatrix} \begin{bmatrix} -1.32 & -1.08 \\ -0.84 & -0.6 \end{bmatrix} & \begin{bmatrix} -1.3253 & -1.0842 \\ -0.843 & -0.602 \end{bmatrix} \\ \begin{bmatrix} 0.6 & 0.84 \\ 1.08 & 1.32 \end{bmatrix} & \begin{bmatrix} 0.602 & 0.843 \\ 1.084 & 1.3253 \end{bmatrix} \end{bmatrix}$$

sig1 →

sig2 →

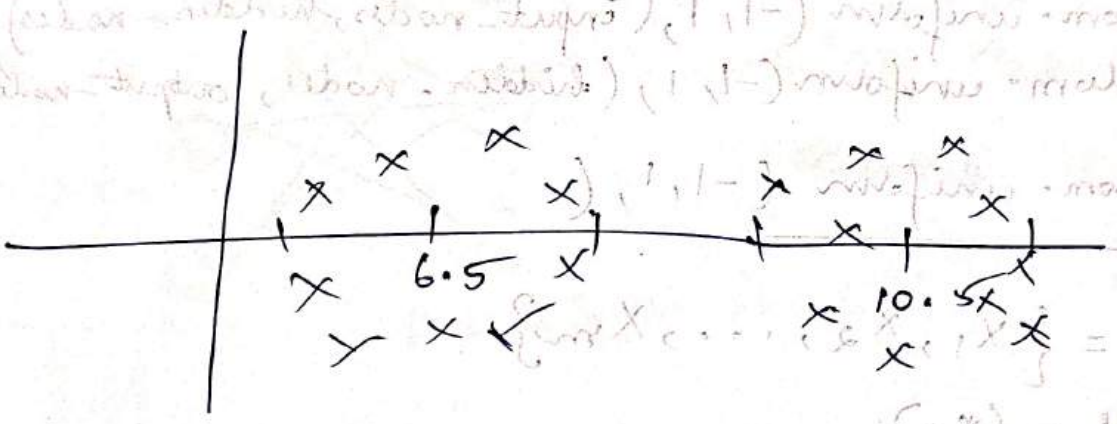
μ after normalisation $\rightarrow 0$

Variance after normalisation $\rightarrow 1$

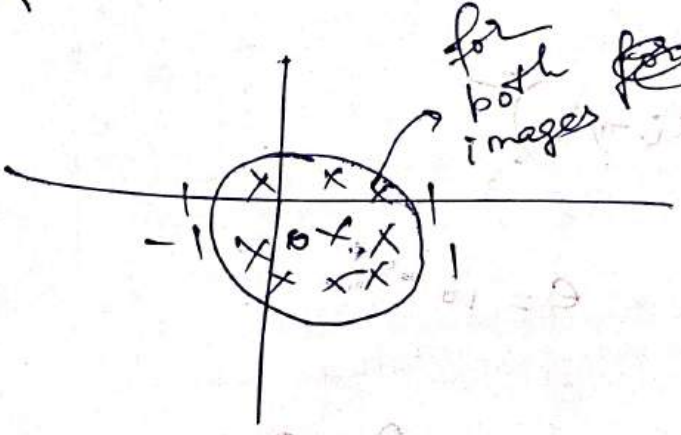
$$\sigma \rightarrow \frac{(-1.32)^2 + (-1.08)^2 + (-0.84)^2 + (-0.6)^2 + (0.6)^2 + (0.84)^2 + (1.08)^2 + (1.32)^2}{8}$$

$$= 1$$

Initially



After normalisation



Pixel distribution doesn't differ, easier training.

BP

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([0,1,1,0])
lr = 0.5
n_epochs = 10000
input_nodes = 2
hidden_nodes = 2
output_nodes = 1
w1 = np.random.uniform(-1, 1, (input_nodes, hidden_nodes))
w2 = np.random.uniform(-1, 1, (hidden_nodes, output_nodes))
b1 = np.random.uniform(-1, 1, (
```

Consider $x_i = \{x_1, x_2, \ldots, x_m\}$

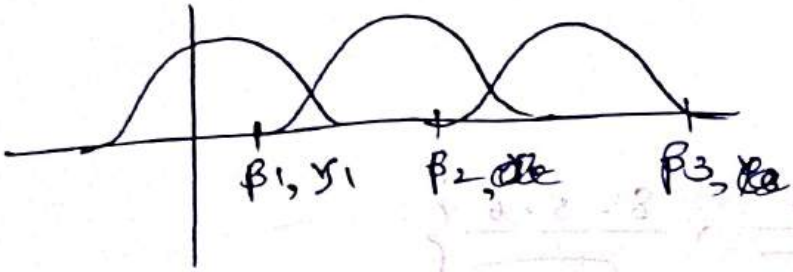
$y_i = BN_{\gamma,\beta}(x_i)$

Then, $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$

$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$

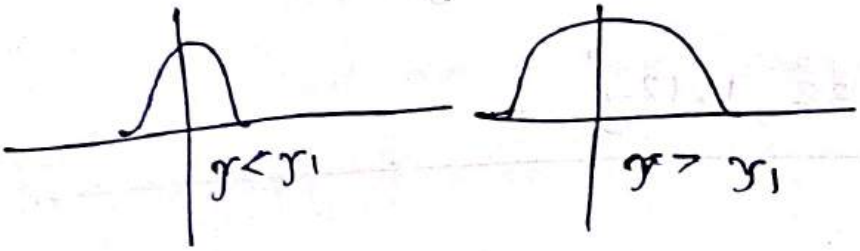
$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \epsilon = 10^{-5}$

Batch Normalisation $y_i = \gamma \hat{x}_i + \beta$

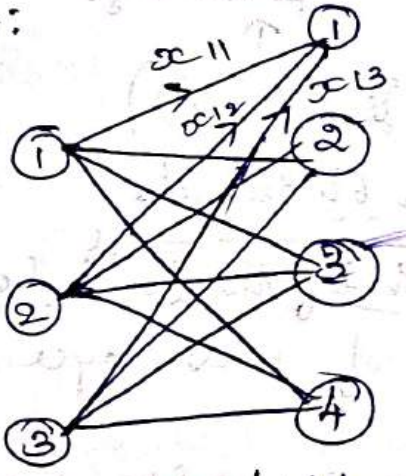


As β increases, the distribution shifts towards right

As γ increases, (or spread) variance will increase.



Consider:



hidden layer

$$X = \begin{Bmatrix} x_{11} & x_{21} & x_{31} & x_{41} \\ x_{12} & x_{22} & x_{32} & x_{42} \\ x_{13} & x_{23} & x_{32} & x_{43} \end{Bmatrix}$$

$x_{11} = 4$

$x_{12} = 5$ for 1 neuron in hidden layer

$x_{13} = 8$

$\therefore X = \{4, 5, 8\}$

$\hat{X}_i = ?$

$$\mu_x = \frac{4+5+8}{3} = \frac{17}{3} = 5.67$$

$$\sigma^2 = \frac{(4-5.67)^2 + (5-5.67)^2 + (8-5.67)^2}{3}$$

$$= \frac{(1.67)^2 + (0.67)^2 + (2.33)^2}{3} = \frac{8.66}{3}$$

$$= 2.89$$

$\sigma = 1.7$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{x_i} = \left\{ \frac{4-5.6}{\sqrt{2.89}}, \frac{5-5.6}{\sqrt{2.89}}, \frac{8-5.6}{\sqrt{2.89}} \right\}$$

$$\hat{x_i} = \{ -0.8, -0.32, 1.12 \}$$

Depthwise convolution:

$X \in \mathbb{R}^{B \times H \times W \times C}$

image = $H \times W \times 32$

Want to do Conv2D(64, (3, 3))

Kernel size

↓ with 64 kernels

Depth of this kernel will be that of prev. layer = 32

Each kernel has size 3×3 so 9 weights of 32 depth.

$\therefore$ Parameters = $(3 \times 3 \times 32) \times 64$

$= 18432$

for 1 kernels

$= (K \times K \times C_i) C_o$

no. of feature maps in prev layer

no. of feature maps we seek to apply

$C_i \rightarrow$ in channels

$C_o \rightarrow$ o/p channels

To avoid 18432 parameters, we move to depthwise convolution.

Depthwise

Kernel → K x K (No. depth)
one kernel for each channel
= K x K x 1

here, we don't decide no. of kernels.

feature map X = H x W x 32 → so directly 32 kernels will be applied.

Also, we don't introduce the parameter no. of output channels.

∴ Parameters = $K \times K \times C_{in}$

C_{in} here = 32 (No. of kernels)

= 3 x 3 x 32

= 288

Therefore, no. of parameters reduced.

Seperate multiplication with each feature map.

Advantages: Less no. of parameters

Disadvantages → Not taking cross-correlation b/w channels.
→ Intermix of channel information not considered.

Normal convolution disadvantage → more no. of parameters
High interchannel interactions there
1st channel with 1st channel & so on.

Separable convolutions:

→ Depthwise convolution + pointwise convolution
→ Incorporates cross channel interaction.

Pointwise convolution

|X| → Kernel size (1, 1) with depth = previous feature map / No. of channels in

Consider an image $H \times W \times C_{in}$

$= 64 \times 64 \times 32$

↓ separable convolution with (64 filters)

(Kernel size) $\times C_{in}$: $3 \times 3 \times 32$ (Depthwise) + $1 \times 1 \times 32 \times 64$ (Pointwise)

(1×1 kernel; depth of previous layer; 64 filters needed)

$= 288 + 2048$

$= (K \times K \times C_{in}) + (1 \times 1 \times C_{in} \times C_{out})$

(C_{in}: depth (previous no. of channels))

which results in output of $H \times W \times C_{out}$ (same padding)

Reduces parameter counts.

Gradient descent

- Vanilla gradient (or) Batch gradient descent
  - Long time for processing because at a time large data taken.
- Stochastic gradient descent
  - → At a time, one sample taken.
  - → Easy to process but is more noisy.
- Minibatch gradient descent
  - → Consider data as different batches.

Minibatch variants: Adagrad, RMS prop, Adam

Stochastic variants: SGD with momentum, (Variant RMS prop)

$\times W \times C_{in}$

$\times 64 \times 32$

↓ separable convolution with (64 filters)

(nel size) $\times C_{in}$ — 1×1 kernel — depth of previous layer

$\times 3 \times 32$ + $1 \times 1 \times 32 \times 64$ (64 filters needed)

Depthwise — Pointwise

$288 + 2048$

$\left(\text{F} \times 1 \times 1 \times C_{in} \times C_{out}\right)$

depth (previous no. of channels)

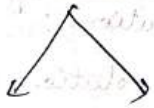
s in output of $H \times W \times C_{out}$ (same padding)

cter counts.

---

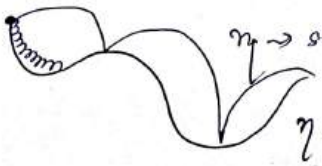
inibatch gradient descent

→ Consider data as different batches.



Adagrad RMSprop Adam

Consider



η → small it may reach local minima not global minima

η → large it may diverge.

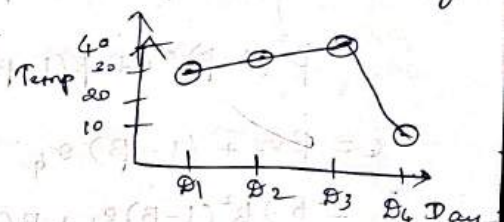
Saddle point → where slope is zero. (gradient)

No weight updation.

Due to this, we introduce momentum.

Exponential weighted average / Exponentially moving weighted average :-

Day	Temp
D1	30
D2	35
D3	40
D4	10



There is a large dip. How will models learn?

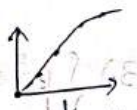
Therefore, we give weights to each day.

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

previous temperature

$v_{t-1} = 0$

$\beta = 0.9$



no sudden fall because previous experience given

$v_1 = 0.9(0) + (1-0.9)(30)$

$= 0.1 \times 30 = 3$

$v_2 = 0.9(3) + (1-0.9)(35)$

$= 2.7 + 0.1 \times 35$

$= 6.2$

$v_3 = 0.9(6.2) + (1-0.9)\,40$

$= 5.58 + 0.1 \times 40$

$= 5.58 + 4$

$= 9.58$

$v_4 = 0.9(9.58) + (1-0.9)\,10$

$= 8.622 + 1$

$= 9.622$

Algorithm :-

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$$v_{t-1} = 0$$

$$v_1 = (1-\beta)\theta_1$$

$$v_2 = \beta v_1 + (1-\beta)\theta_2$$

$$= \beta(1-\beta)\theta_1 + (1-\beta)\theta_2$$

$$v_3 = \beta v_2 + (1-\beta)\theta_3$$

$$= \beta\{\beta(1-\beta)\theta_1 + (1-\beta)\theta_2\} + (1-\beta)\theta_3$$

$$= \beta^2(1-\beta)\theta_1 + \beta(1-\beta)\theta_2 + (1-\beta)\theta_3$$

$$v_4 = \beta v_3 + (1-\beta)\theta_4$$

$$= \beta\{\beta^2(1-\beta)\theta_1 + \beta(1-\beta)\theta_2 + (1-\beta)\theta_3\} + (1-\beta)\theta_4$$

$$= \beta^3(1-\beta)\theta_1 + \beta^2(1-\beta)\theta_2 + \beta(1-\beta)\theta_3 + (1-\beta)\theta_4$$

$$= (1-\beta)\{\beta^3\theta_1 + \beta^2\theta_2 + \beta\theta_3 + \theta_4\}$$

Therefore, $v_t = (1-\beta)\{\beta^{t-1}\theta_1 + \beta^{t-2}\theta_2 + \ldots + \beta\theta_t\}$

The no. of previous values we shall consider is given by the formula → $\frac{1}{1-\beta}$ (depends on the value of β)

β small → $\beta = 0.5$ → $\frac{1}{1-\frac{1}{2}} = 2$

β large → $\beta = 0.9$ → $\frac{1}{1-0.9} = \frac{1}{0.1} = 10$

iloc[0] → first element

Here to give value of β, we incorporate previous gradients to ensure proper convergence & not

Prevent large gradient → large weight case

confusing the model with varying weight updation.

Generate time series data & plot the variations with EWMA.