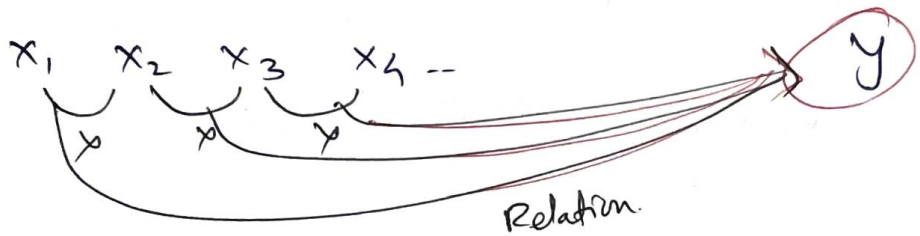
Deep Learning

objective of DL :-

To build an AI system capable enough to predict & take decision on the behalf of human being.

* Machine learning :-

Independ. Varr. → Depend. Varr



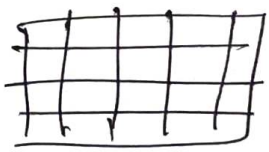
* Drawback of ML :-

↳ It is only suitable for structure Data.
(column + rows)

* [DL is applicable on Unstructure Data.]

- Text
- image
- Video
- Audio.

image :-



↳ pixel

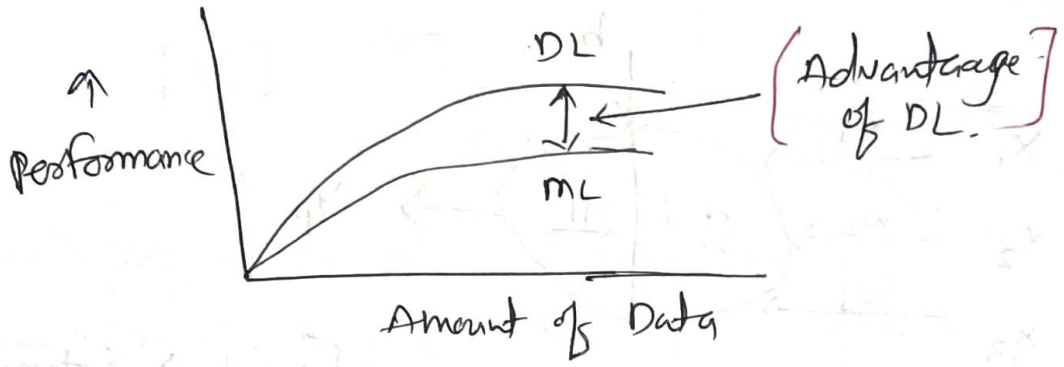
15 inches → Laptop Screen

↳ Resolution (count of Pixel)

1366 × 768

↳ 10 Lakh

* Performance graph of DL & ML :-



ML ⇒ Data preprocessing

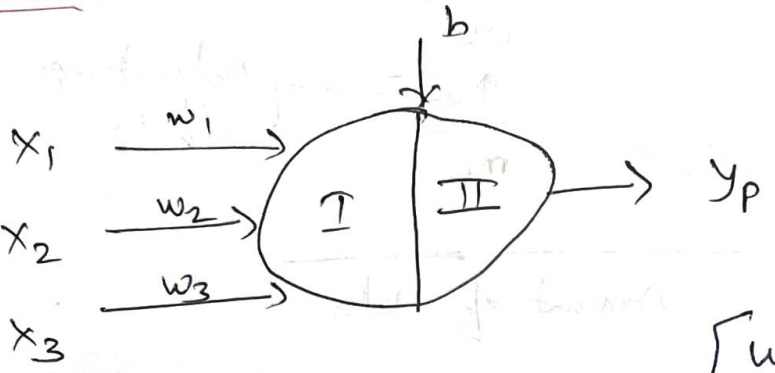
60 to 70% work

① feature engg.
- missing value Handling
- outliers detection & Handling
- Encoding
- Data balancing
- Normalization

② Feature selection techniques.

DL ⇒ [itself act as a feature selector.]

Neuron (Perceptron) :-



[w, b → External parameter]

Stage I → Summation fun^n

$$z = \sum_{i=1}^{4} w_i \cdot x_i + b$$

Stage II → Activation fun^n (sigmoid)

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

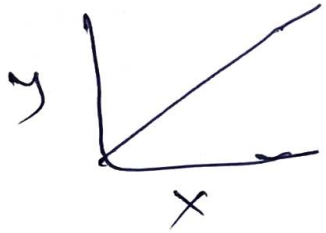
- This whole process called as perceptron & perceptron is nothing but neuron in NN.)

Sigmoid Activation fun^n :-

- It is used to add non-linearity.

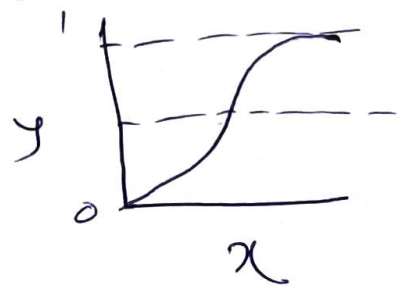
Linear fun^n

$$y = mx + c$$



Sigmoid

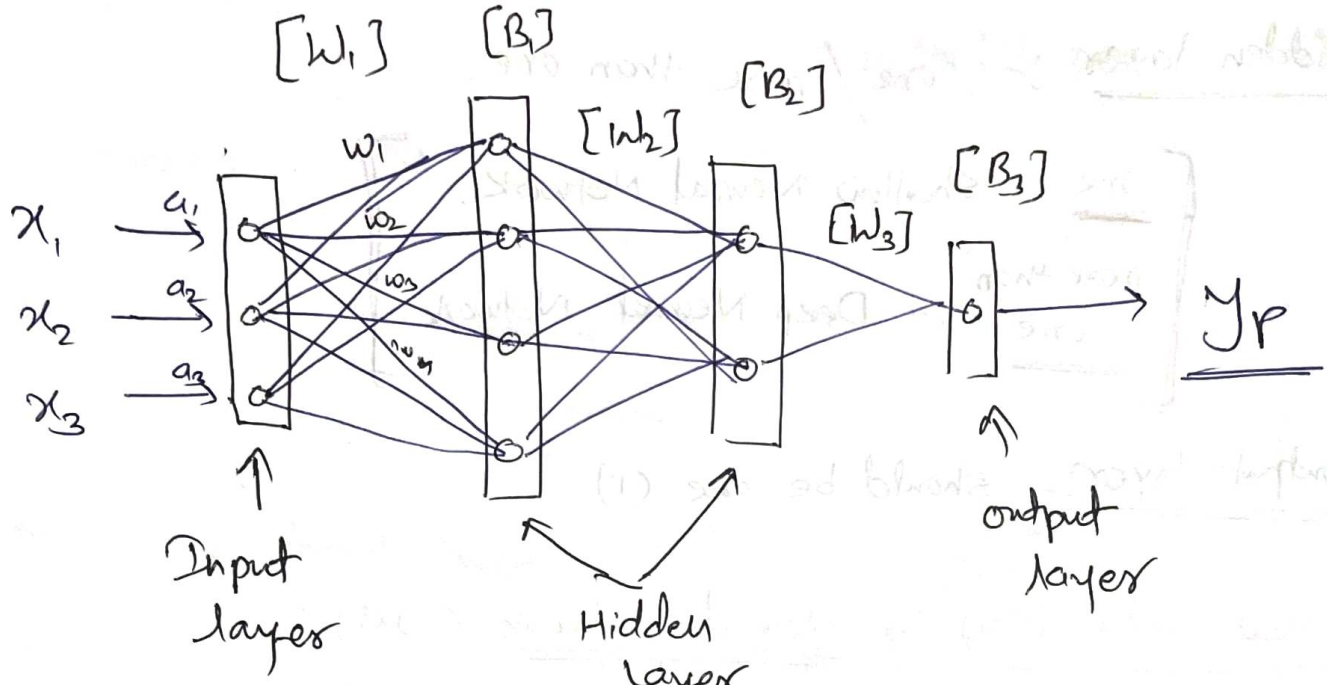
$$\sigma(y) = \frac{1}{1+e^{-y}}$$



- It gives probability value of any class.

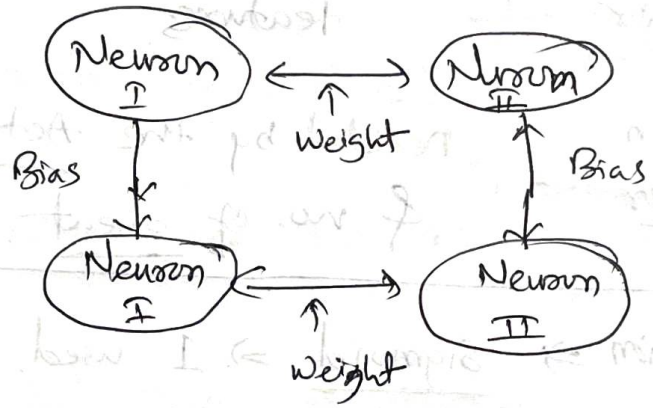
class
0 1

Structure of Neural Network (NN) :-



Weight -

Bias -



$[W_1] = [W_1, W_2, W_3, W_4, - - -, W_{12}]$

$[B_1] = [B_1, B_2, B_3, B_4]$

Input layer :- It hold the input & feed them to hidden layer

Hidden layer :- Done all the processing / learning.

output layer :- It shows the output

Input layer :- Should be one (1)

Hidden layer :- one / more than one

[one :- Shallow Neural Network
more than one :- Deep Neural Network]

output layer :- Should be one (1)

How Data flow in Neural Network (NN) :-

No. of neuron in input layer = no. of input features.

No. of neuron in output layer = Decided by the Activation fun^n & no. of event.

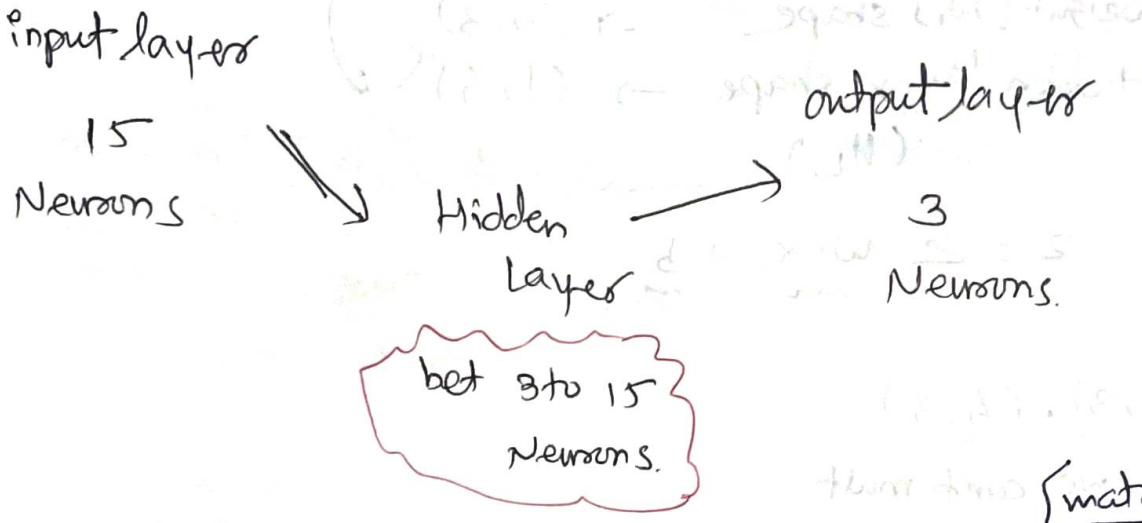
① Binary classification ⇒ Sigmoid ⇒ 1 used. (Neuron in o/p)

② multiclass classification ⇒ Softmax ⇒ (3 neuron in o/p.)

③ Regression problem ⇒ Relu ⇒ 1

④ [No. of Neuron in Hidden layer] ⇒ Not fixed.

General Thumb Rule



- shape of input layer = (1,3) (1st layer of 3 neurons)
- Shape of Hidden layer = (1,4) HL_1
- shape of Hidden layer = (1,2) HL_2
- Shape of output layer = (1,1)

[matrix.]

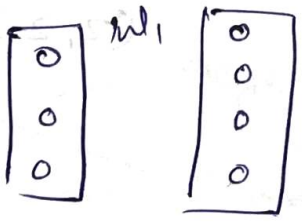
$[W_1] = (4,3)$ — Stage I

$[W_2] = (2,4)$ — Stage II

$[W_3] = (1,2)$ — Stage III

How to control Shape of the layers :—

Let's say we start from 3 Neuron & ends with 4 neurons.



W_1 → 12 weights.

Input layer shape → (1,3)

weight matrix shape ⇒ (4,3)

output layer shape → (1,4)

eg:- input layer shape → (1,3)
- weight [W_1] shape → (4,3)
Hidden layer shape → (1,4)
(H_{L_1})

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$(1,3) \cdot (4,3)$

we can't mult.

so, we Transpose the matrix.

$(1,3) \cdot (3,4) = (1,4)$

$$z = \sum_{i=1}^{M} w_i^T \cdot x_i + b$$ ← Transpose

Importance of Bias :-

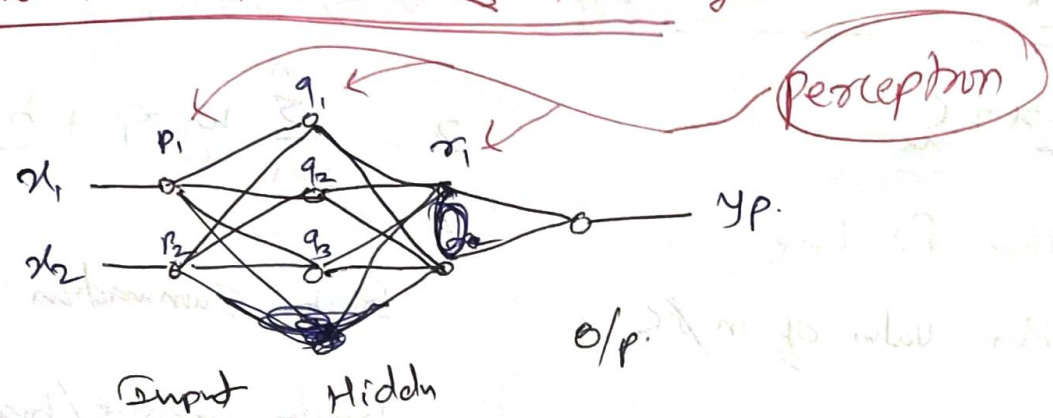
Bias ⇒ one dimensional array

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$\begin{bmatrix} 1 & 4 \\ 3 & 5 \end{bmatrix} + 2 = \begin{bmatrix} 3 & 6 \\ 5 & 7 \end{bmatrix}$$

(2,2) (2,2)

How Neural Networks learn :-



Chain Rule :-

$x_1 \to P_1 \to q_1 \to r_1 \to y_{p_1}$

$x_1 \to P_1 \to q_2 \to r_1 \to y_{p_2}$

$x_1 \to P_1 \to q_3 \to r_1 \to y_{p_3}$

$x_2 \to P_2 \to q_1 \to r_1 \to y_{p_4}$

$x_2 \to P_2 \to q_2 \to r_1 \to y_{p_5}$

$x_3 \to P_2 \to q_3 \to r_1 \to y_{p_6}$.

} 6 chain.

$x_1 \to q_1$

Stage I $\to z_1 = w_1 x_1 + b_1$

Stage II $\to \sigma(z_1) = \frac{1}{1+e^{-z_1}} = q_1$

$q_1 \to r_1$

Stage I $\to z_2 = w_2 q_1 + b_2$

Stage II $\to \sigma(z_2) = \frac{1}{1+e^{-z_2}} = y_{p_1}$

$y_a - y_{p_1} \Rightarrow$ Loss. (L)

Linear Regression

$y = mx + c$

↳ Best fit line

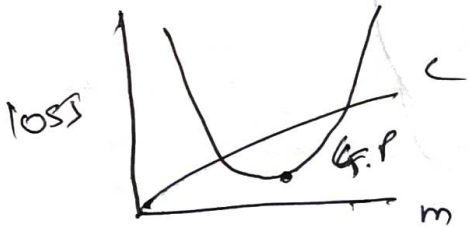
Best value of m & c

- Gradient Decent algo.

(Local minima)

$$m_{new} = m_{old} - \alpha \frac{\partial L}{\partial m}$$

$$c_{new} = c_{old} - \alpha \frac{\partial L}{\partial m}$$



Deep learning

$$z = \sum_{i=1}^{n} w_i x_i + b$$

↳ Best Summation fun^n

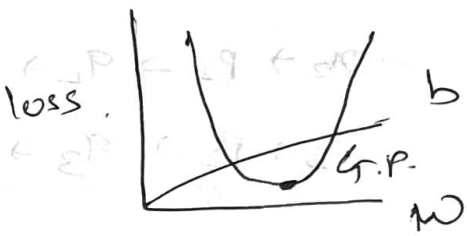
Best weight/bias.

- Gradient Decent algo.

(Local minima)

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha \frac{\partial L}{\partial b}$$



$\min [L_1, L_2, L_3, L_4, L_5, L_6]$.

minimum loss → Best w & b.

↳ Best Summation fun^n

$$z = \sum_{i=1}^{n} w_i x_i + b.$$

↳ Activation fun^n

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

① Forward propagation (F.P)
② Backward propagation (B.P)

left to right ⟶ ⇒ F.P.

right to left ⟵ ⇒ B.P.

F.P ⇒ y_p, loss

B.P ⇒ to updates the parameters

Epoch / iteration :-

1500 data points ⇒ FP + BP → 1 complete epoch

1500 data points
- 500 → FP_1 + BP_1 → Iteration I
- 500 → FP_2 + BP_2 → Iteration II
- 500 → FP_3 + BP_3 → Iteration III

Epoch.

1 Complete epoch

* Types of fun^n use in Neural Network :-

1. Optimization fun^n.
2. Activation fun^n.
3. loss fun^n.

1. Optimization fun^n :-

- To minimize / optimize error / loss
- Used to find best value for parameters.
  (w, b, learning rate (α))
- Achieve convergence / global minima point.

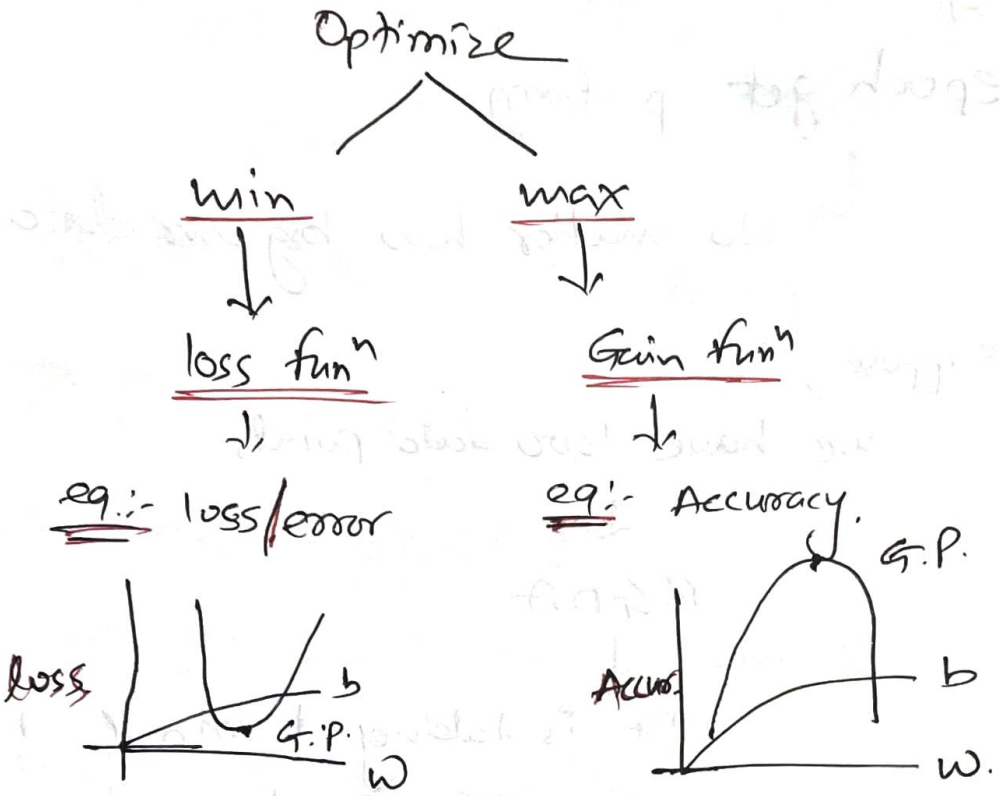
2. Activation fun^n :-

- Used to control output / neuron.
- It is governing the way neuron behaviour.

3. loss fun^n :-

- Gives a no. ($y_a - y_p$) which represent the goodness of model.
- Help to update the external parameters.
  (w, b, α)
  ↑ learning rate

① Optimization fun^n :-



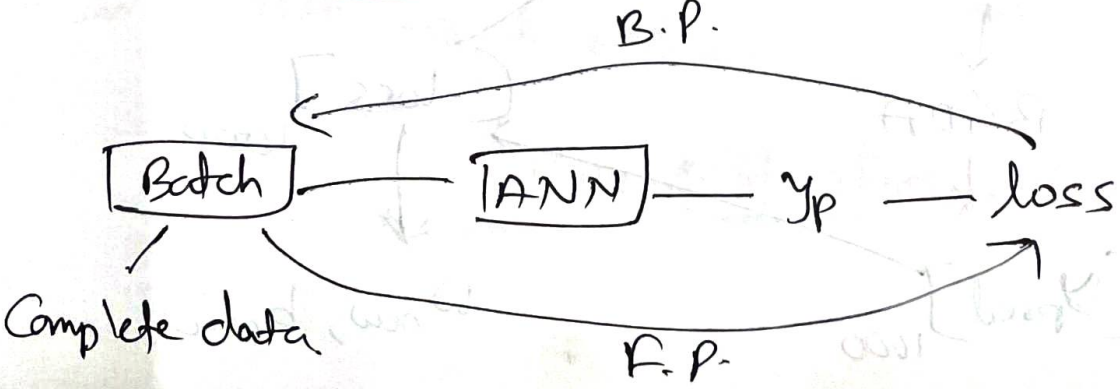
- optimization fun^n required.

taking minimum time (↓) & resources consumption to converge.

* Types of optimization fun^n :-

① weight & bias | ② learning | ③ weight, bias & learning

① Gradient Decent algorithm :-



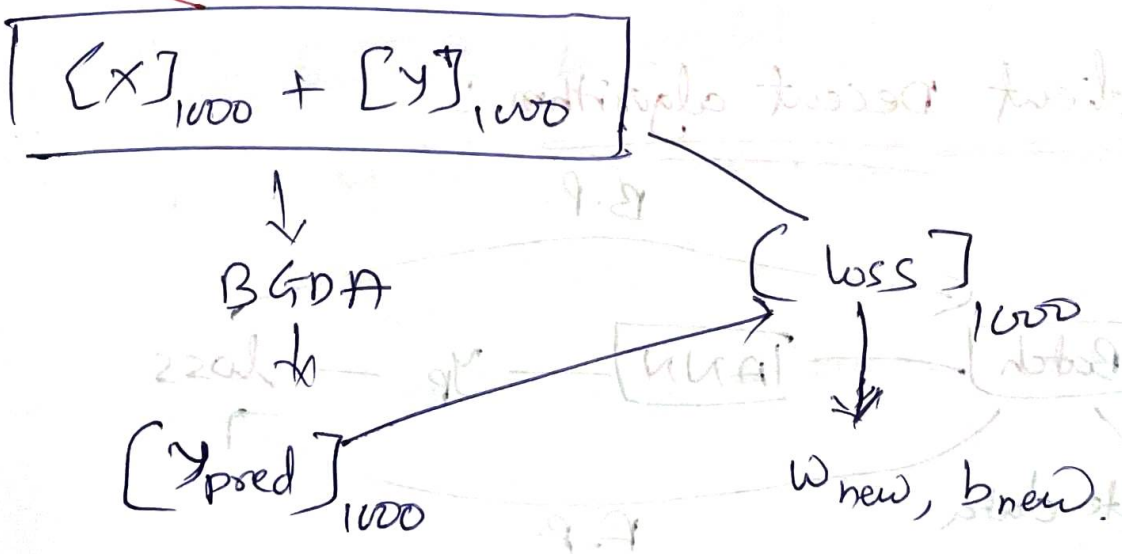
- on whole data.
  ↳
  epoch get perform
  ↳ No matter how big our data.

- ~~We~~ Suppose,
  we have 1000 data points
  ↓
  BGDA
  ↓
  It is taking 1 min & 1 GB.
  memory to achieve convergen

- Now, we have 1 lakh data
  ↓
  BGDA
  ↳ 1000 min + 100 G.B.

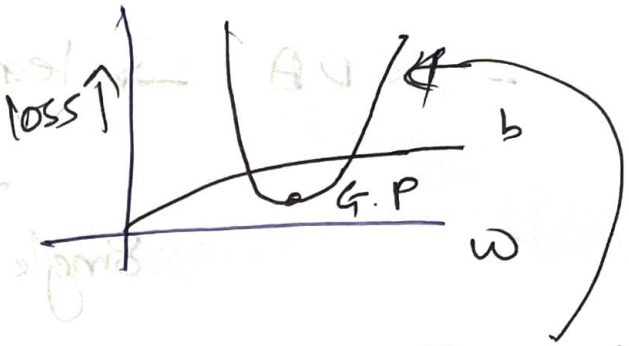
- So, BGDA is not suitable for big data size.

* How Batch Gradient Decent work in Backend. :-
(BGDA)



$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha \frac{\partial L}{\partial b}$$



Smooth learning curve

eg:- ① [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

→ (2nd table), divide 2, etc

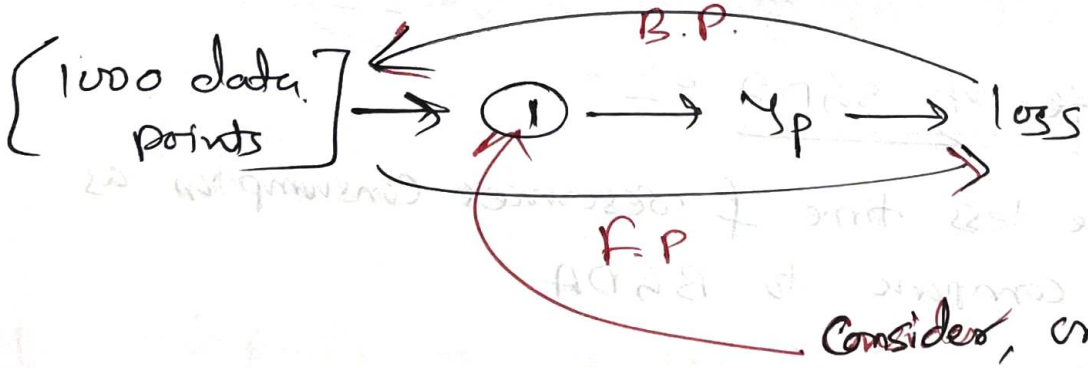
② [4, 10, 14, 8].

* Advantage of BGDA:-
  - we get smooth learning curve.

* Disadvantage of BGDA:-
  - we can't use BGDA in case of Big data.

② Stochastic Gradient Decent algorithm (SGDA):-

↳ (Randomness / Random.)



Consider, one data points at a time randomly

- In SGDA → 1 epoch

↓

'N' no. of iterations. ('N' - no. of data point)

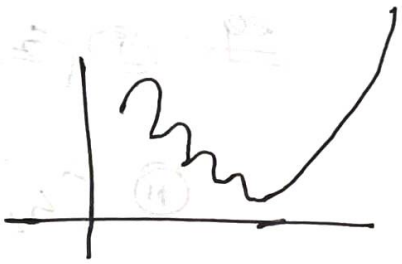
- SGDA → learns from

↓

Single data point at a time.

↓

learning curve become very noisy.



→ 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.

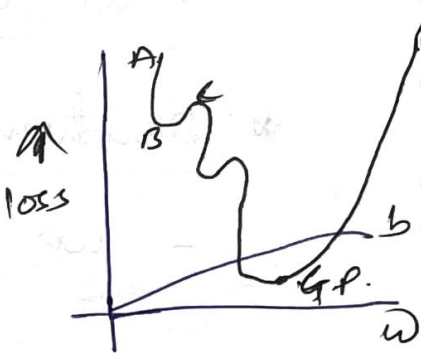
1st iteration → 10 → y_p → $loss_1$

w_1'', b_1'

2nd iteration → 4 → y_p → $loss_2$

w_2'', b_2''

3rd iteration → 8 → y_p → $loss_3$



* **Advantages of SGDA :-**

- take less time & resources consumption as compare to BGDA.

* **Disadvantage :-**

- very prone to local minima
- very noisy curve

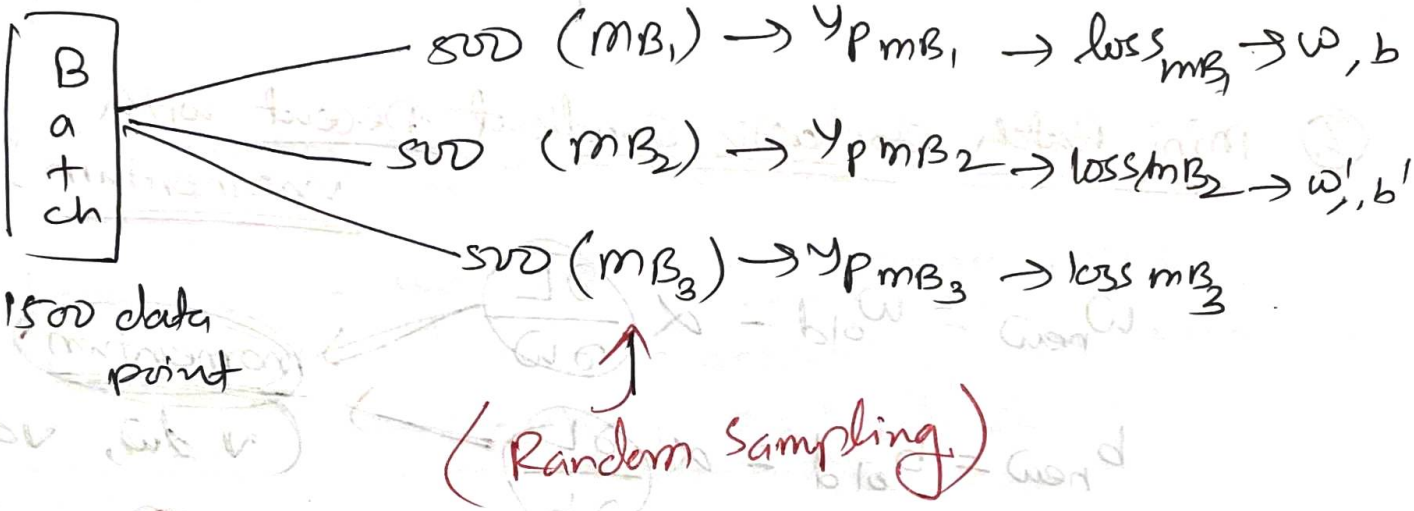
③ Mini Batch Stochastic Gradient Decent Algo :-

(MBSGDA)

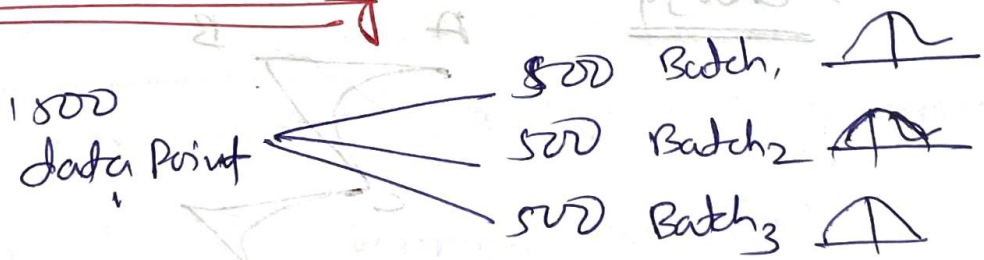
BGDA + SGDA

BGDA → Smooth learning curve

SGDA → Step by step learning

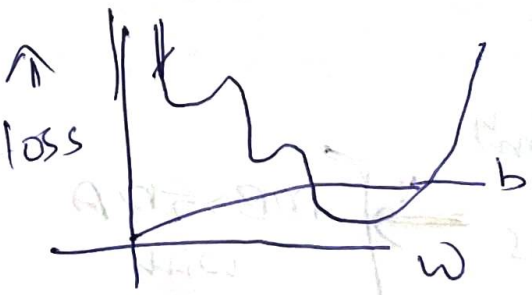


* Random sampling :-



1000 data points → 1000 Batches

↳ 100 data points / per batch



- In MBSGDA we learn from Mini Batch.
- Here, we have sufficient amount of learning to overcome local minima.

** Advantage :-
- much better than BGDA & SGDA

Disadvantage :-
- Noise is still present

⑤ Mini Batch Stochastic Gradient Decent with momentum :-

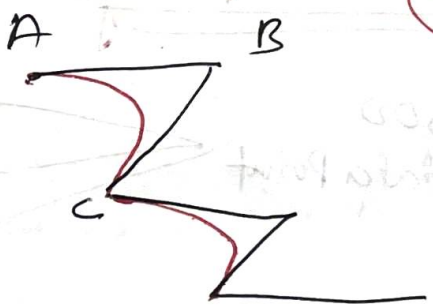
$$w_{new} = w_{old} - \alpha \left(\frac{\partial L}{\partial w}\right)$$

$$b_{new} = b_{old} - \alpha \left(\frac{\partial L}{\partial b}\right)$$

momentum ($v\,dw$, $v\,db$)

(Calculate from formula)

Gradient → Energy

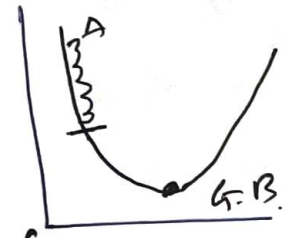


** Advantage of MBSGDA with momentum :-
- Better the MBSGDA.

Best Optimization fun^n for weight + Bias → MBSGDA with Momentum

* ~~Why~~ Why learning rate need to be dynamically updated ?

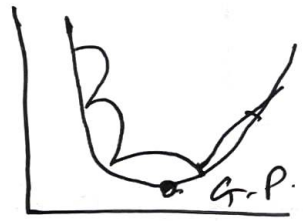
(I) If learning rate become too small :-



(vanishing Gradient)

- training time ↑ exponentially.
- we face vanishing gradient issue.
- we never achieve our Global minima.

(II) If learning rate become too big :-



(Exploding Gradient)

- we face over shooting issue
- due to overshooting we face exploding Gradient issue

Learning rate ⇒ Dynamically Update

⇒ initialize learning rate (α)
↳ 0.1
↳ too big
↳ ↓ - decrease

⇒ initialize learning rate (α)
↳ 0.0001
↳ too small
↳ ↑ - increase

① Ada grad (Adaptive Gradient) #learning Rate :-
② Ada delta. (RMS prop) (Root Mean Squared Propagation)

① Adagrad (Adaptive Gradient) :-

$$w_{new} = w_{old} - \alpha_{new} \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha_{new} \frac{\partial L}{\partial b}$$

$$\alpha_{new} = \frac{\alpha_{old}}{\sqrt{\eta + \epsilon}} \qquad \therefore \epsilon \cong e^{-6}$$

$$\eta = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial w_i}\right)^2$$

② Ada Delta (RMS prop) :-

$$w_{new} = w_{old} - \alpha_{new} \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha_{new} \frac{\partial L}{\partial b}$$

$$\alpha_{new} = \frac{\alpha_{old}}{\sqrt{Sdw\epsilon + \epsilon}} \qquad \therefore \epsilon \cong e^{-6}$$

$$Sdwt = \beta + Sdw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2$$

[β - Smoothing parameter
β = range of 0 to 1]

Best optimization fun^n for 'w' & 'b'

↓

- Mini Batch Gradient Decent Sto. with momentum (MBSGD with Momentum)

Best optimization fun^n for learning rate

↓

- Ada Delta (RMS prop)

Adam :- (Adaptive moment estimation)

⇒ MBSGDA with Momentum

$$w_{new} = w_{old} - \alpha . Vdw$$

$$b_{new} = b_{old} - \alpha . Vdb$$

⇒ Ada Delta (RMS Prop)

$$w_{new} = w_{old} - \alpha_{new} \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha_{new} \frac{\partial L}{\partial b}$$

$$\therefore \alpha_{new} = \frac{\alpha_{old}}{\sqrt{Sdw_t} * \epsilon}$$

⇒ Adam

$$w_{new} = w_{old} - \frac{\alpha_{old}}{\sqrt{Sdw_t} * \epsilon} Vdw$$

$$b_{new} = b_{old} - \frac{\alpha_{old}}{\sqrt{Sdb_t} * \epsilon} Vdb$$

Activation fun^n :- — [it control o/p layer]

1. Linear activation fun^n ×
2. Sigmoid activation fun^n ⇒ Binary classification.
3. tanh activation fun^n ⇒ Binary classification.
4. Softmax ⇒ Multiclass classification.
5. Relu ⇒ Base activation fun^n for Regression.
6. leaky relu
7. P-relu
8. Elu
9. Swish

→ varient of Relu.

1. Sigmoid activation fun^n :-

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \therefore z = \text{summation fun}^n$$

$$z = \sum_{i=1}^{n} w_i x_i + b$$

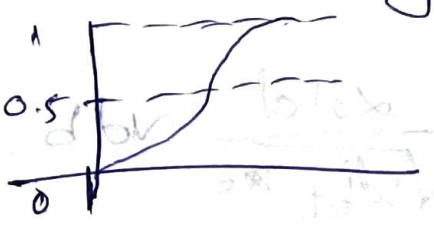
- Range ⇒ 0 to 1
- It gives probability value of any one class

eg:- Cat / Dog — two classes

$$P(\text{Dog}) = -$$

$$P(\text{Cat}) = 1 - P(\text{Dog})$$

- It's add non-linearity.



* Advantages :- (Sigmoid fun^n)

If we use sigmoid in our o/p layer.

① Sigmoid gives uniform result.

Range – 0 to 1

② on the basis of probabilisty values, we can take decision ~~consider~~ confidently.

* Why use not sigmoid activatin fun^n in our Hidden layer. :-

① zero centricity :-

mean $(f(x)) \cong 0$

↳ zero centric fun^n

mean $(f(x)) \neq 0$

↳ Non-zero centric fun^n

- zero centric fun^n converge faster than non-zero centric fun^n.

| Sigmoid – Range – 0 to 1 |

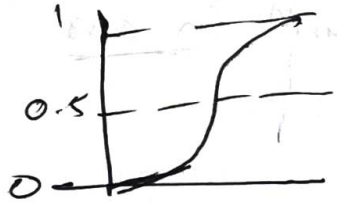
↳ non-zero centric fun^n

- Since, sigmoid is non-zero centric fun^n & non-zero centric fun^n takes longer time to converge, so we avoid it. ~~in~~ use in Hidden layer.

② tanh :-

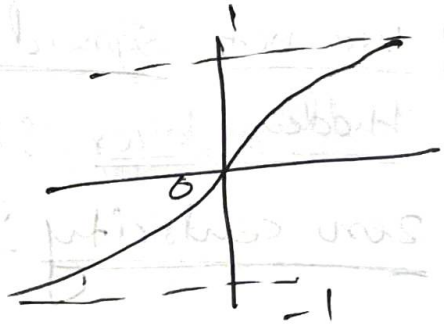
Sigmoid

- $\sigma(z) = \frac{1}{1+e^{-z}}$
- range - 0 to 1
- Non-zero centric fun^n



Tanh

- $\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$
- range = -1 to 1
- Zero centric fun^n



* Advantage

- Zero centric fun^n

* Disadvantage

- It is computationally heavy compare to sigmoid

③ Relu (Rectified Linear Unit) :-

- Non-zero centric fun^n

$$\text{relu}(z) = \max(0, z)$$

eg:- $z = 55$ or $z = -55$

where, $z = 55$

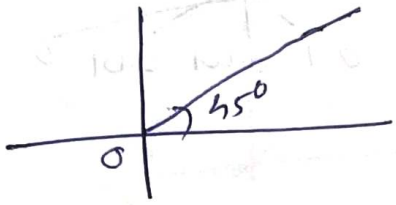
$\text{relu}(z) = \max(0, 55)$

$$\text{relu}(z) = 55$$

Where, $z = -55$

$$relu(z) = \max(0, -55)$$

$$relu(z) = 0$$



- 5% Chances - z(-ve)

* Advantages

- we can avoid Vanishing gradient issue by using relu.

* Disadvantages

- non-zero centric fun^n
- we might face Dead neuron / Dead relu issue

④ Leaky Relu :-

$$\text{relu}(z) = \max(0, z)$$

$$lr(z) = \max(0.001z, z)$$

0.1 0.001 0.01 0

Condition ① $z = +ve$

$$\frac{\partial}{\partial z} lr(z) = \frac{\partial}{\partial z} \max(0.001z, z)$$

$$= \frac{\partial}{\partial z}(z)$$

$$\boxed{\frac{\partial}{\partial z} lr(z) = 1}$$

Condition ② $z = -ve$

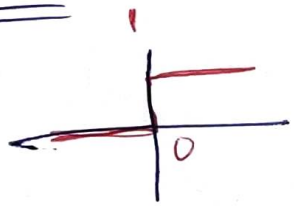
$$\frac{\partial}{\partial z} lr(z) = \frac{\partial}{\partial z} \max(0.001z, z)$$

$$\frac{\partial}{\partial z}(0.001z)$$

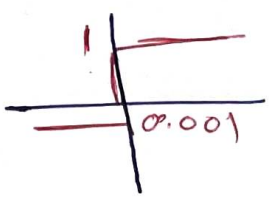
$$= 0.001 \frac{\partial}{\partial z}(z)$$

$$\boxed{\frac{\partial}{\partial z} lr(z) = 0.001}$$

Relu



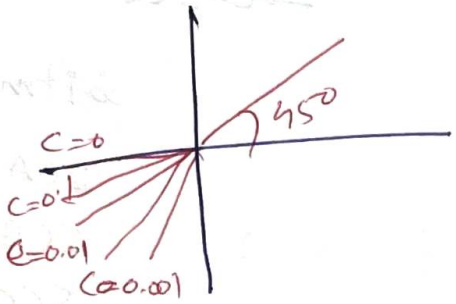
Leaky Relu



⑤ p-relu (Parametric Relu) :-

$$\text{Prelu}(z) = \max(0.001z, z) \quad \text{--- 1 x.}$$

leaky Relu.

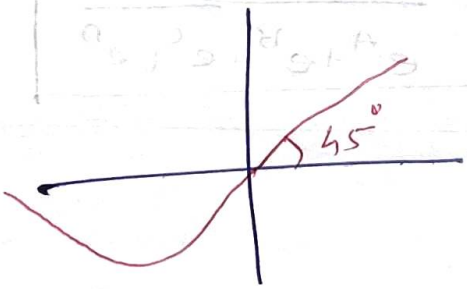


⑥ Elu (Exponential Linear Unit) :-

→ we get smooth learning curve in case of Elu & it is also a zero centric fun".

$$\text{Elu}(z) = \max(\alpha(e^z - 1), z)$$

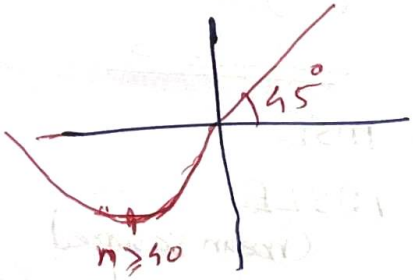
↳ learning parameter.



⑦ Swish :-

$$\text{Swish}(z) = z \times \sigma(z)$$

$$\text{Swish}(z) = z \times \frac{1}{1+e^{-z}}$$



- we can't use swish in case of shallow neural network.
- only use $N \geq 40$, (N = No. of hidden layers).

⑧ Softmax :-

- It is a modified version of sigmoid fun^n.
- only use in multiclass classification.

classes	Softmax Result
A	S(A) = P(A)
B	S(B) = P(B)
C	S(C) = P(C)
D	S(D) = P(D)

$$P(A) + P(B) + P(C) + P(D) = 1$$

$\Rightarrow$ max (P(A), P(B), P(C), P(D))

↳ classification

$$S(A) = P(A) = \frac{e^A}{e^A + e^B + e^C + e^D}$$

---

loss fun^n :-

Regression loss

① MSE
② MSLE (mean squared logarithmic Error)
③ MAE

⇒ relu, leaky relu, p-relu, Elu, Swish

Binary classificat. loss

① Binary cross entropy loss fun^n
② Hinge loss

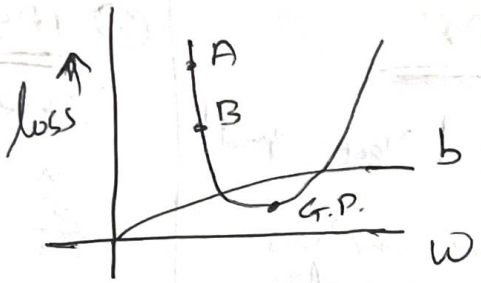
⇒ Sigmoid
· tan h

multiclass classifi. loss

① multi-class entropy
categorical cross entropy loss fun^n
② Spare multi class.
spare categorical.

⇒ Softmax.

Initializer :- (Distribution)



works.

Weight

Initializer (Distribution)

- He initializer
  - He Uniform distribution
  - He normal distribution
- Glorot initializer
  - Glorot Uniform
  - Glorot normal

* General thumb Rule :-

1. Regression problem ⇒ He initializer
2. Classification problem ⇒ Glorot initializer

① Regression loss fun^n :-

① MSE

$$= \sum_{i=1}^{n} (y_{act} - y_{pred})$$

- Use squared term
- cann't use, when 'outliers present'

② MAE

$$= \sum_{i=1}^{n} |y_{act} - y_{pred}|$$

- we use outliers are present.

③ MSLE

(mean squared logarithmic Errors)

$$\sum_{i=1}^{n} (y_{act} - y_{pred})^2$$

② Binary classification loss fun^n :-

① Binary cross entropy log fun^n / Log loss fun^n :-

$$= \frac{-1}{N} \sum_{i=1}^{N} [y_a \log y_p + (1 - y_a) + \log (1 - y_p)]$$

Condition,

1. classification ⇒ Binary.
2. output layer ⇒ Sigmoid ✓
3. target variable ⇒ 0 / 1

② Hinge loss :-

Condition

1. classification ⇒ Binary
2. o/p layer ⇒ tanh ✓
3. target variable ⇒ -1 / +1

③ multiclass classification loss fun^n :-

① multiclass cross entropy loss fun^n :-

conditions :

① classification ⇒ multiclass

② o/p layer ⇒ softmax ✓

③ activation fun^n

$$\text{loss} = \sum_{i=1}^{M} y_a \log(y_p)$$

② Sparse categorical cross entropy loss fun^n :-

① classification → multiclass

② o/p layer → softmax

activation fun^n

CNN :- (Convolutional Neural Network)

↳ image processing
or
image classification.

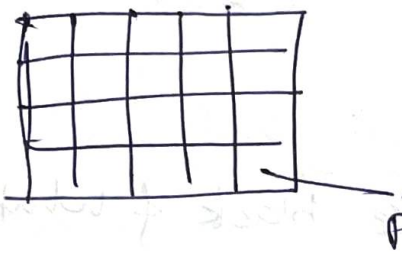
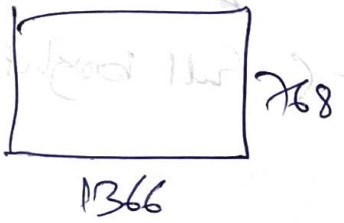


image
↳ pixel
↳ Resolution
↳ countpixel

Laptop Screen → 15 inches
↳ 1366 x 768
↳ 10 Lakh pixels.



pixel
↳ if we take any 3 primary color &
if we change the color intensity &
brightness intensity. we can form
any required color.

* RGB format/scale :-

- Color images form by using 'RGB' scale
- RGB scale we have 3 primary color.
(Red, Green, Blue)

- Range → 0 to 255

0 → Black (No Brightness)
255 → White (full Brightness)

Gray Scale :-

- Gray scale is use increase black & White image
- In Gray scale, we have only one primary color.
  ↳ Black
- Range ⇒ 0 to 255

0 → Black (No brightness)
↳ shades of gray color.
255 → White (full brightness)

Note :-

"in case of image processing & image classification each pixel act as individual features."

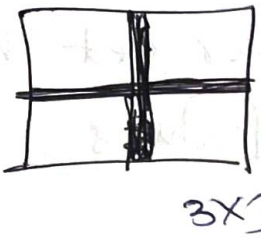
Convolutional Neural Network :-

Convolutional Layer

- it is used to extract features from the original image

pooling Layer

- it is used to reduce size of extracted feature



3×3

Black & white image

255	0	255
0	0	0
255	0	255

(Grey Scale)

↓

Black

↓

one

No. of features ⇒ 3×3×1 = 9

↑ Color.

Color image :-

RGB	RGB	RGB
+	−	−
+	−	−

→ primary color → 3

3×3

No. of features = 3×3×3 = 27

↳ color.

* Color depth / Depth of color :-

① Black & white image → Black → Color depth is one

↳ (primary color) (100×100, 100×100×1)

② Color image → 3 primary color → Color depth is 3

↳ (RGB) (100×100×3, 100×100×RGB)

* **Stride :-** By how many pixel filter get moved to form extracted features

* **pooling layer :-**

1. max pooling
2. Avg pooling
3. Sum pooling

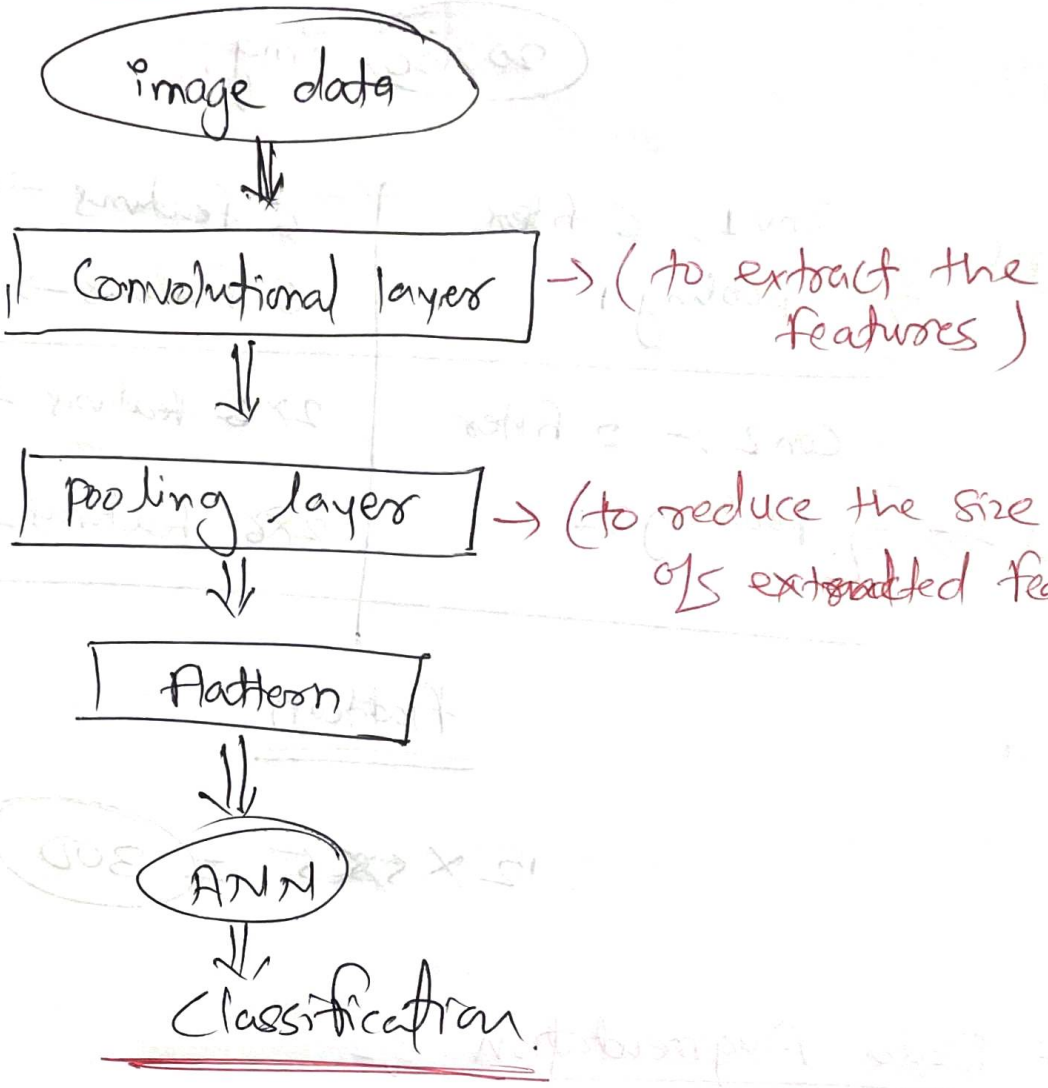
* If we have bad quality img. (that means pixels are not good)
  ↳ [bring good quality values pixel, we use max pooling]

* If we have good quality img. (that means pixels are good).
  ↳ [we use Avg pooling]

NOTE :-

1. Bad quality image ⇒ Max pooling
2. Good quality image ⇒ Avg pooling.

Flow of CNN :-



⇒ Suppose we have 2 image of size 3×3.

	p	q	r
a	0	121	25
b	123	255	256
c	255	145	125

cat

	p	q	r
a	12	125	25
b	34	40	52
c	125	0	35

Dog

feature Names ⇒ ap, aq, ar
bp, bq, br
cp, cq, cr
} (9) features.

⇒ Suppose we have 20 x 20 image

20 x 20 img.

set 1 {	Conv 1 :- 6 filter pooling 1	6 features → 20x20 → Reduce 50%. 6 features → 10 x 10
set 2 {	Con 2 :- 2 filter pooling 2	2x6 features → 10x10 → reduce 2x6 features → 5x5

Flatten

12 x 5 x 5 = 300 features.

Data Augmentation :-

We are trying to generate synthetic copies from the original image by considering different logic.