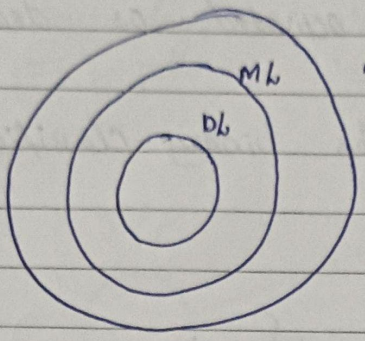
__/__/____

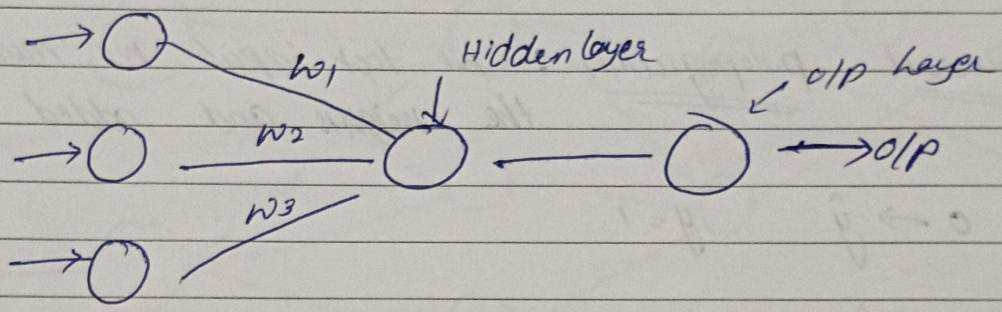
Deep learning



AI => Application which can do its own task without any human intervention

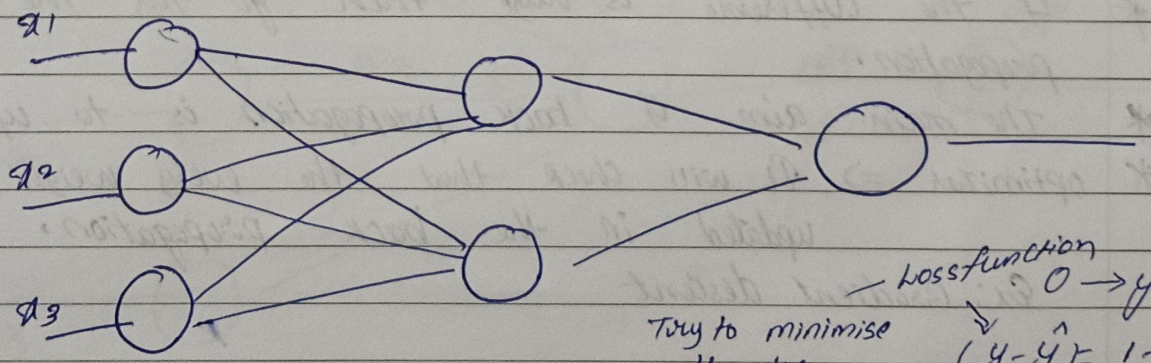
ML => Stats tool to analyze the data, visualize the data, predictions, forecastings

Perceptron { Single layered Neural Network }



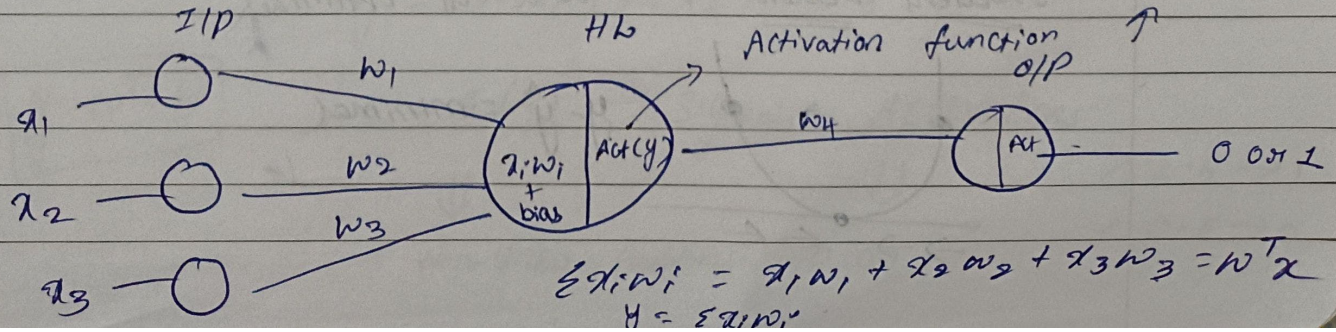
Eg:-

Study (hr)	Play (hr)	Sleep (hr)	Pass/Fail
7	3	7	1
2	5	8	0
4	3	7	1



Loss function = $0 \rightarrow \hat{y}$; $y=1$

Try to minimise the difference $(y-\hat{y}) = 1-0 = 1$



$\Sigma x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 = w^T x$

$y = \Sigma x_i w_i$

__/__/____

Weight → how much the neuron gets activated or deactivated

Activation function →

Sigmoid ⇒ $\frac{1}{1+e^{-y}}$ ⇒ used for binary classification

$= \frac{1}{1+e^{-(\sum_i w_i + b)}}$

⇓

0 to 1 $\begin{cases} \geq 0.5 \rightarrow 1 \\ < 0.5 \rightarrow 0 \end{cases}$

Forward propogation : we took input we multiplied with the weights and added a bias

$0 \rightarrow \hat{y} \quad y = 1$

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

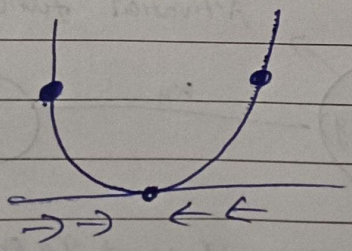
↓ ⟶ this difference should be near to 0

loss function ⇒ difference b/w predicted and real value we should try to minimize the difference and it always nearly to 0

* If the difference is huge then go for the back propogation.
* The main aim of back propogation is to update weights
* optimizer ⇒ It will check that the every weight should be updated in the back propogation.

Ex: Gradient descent

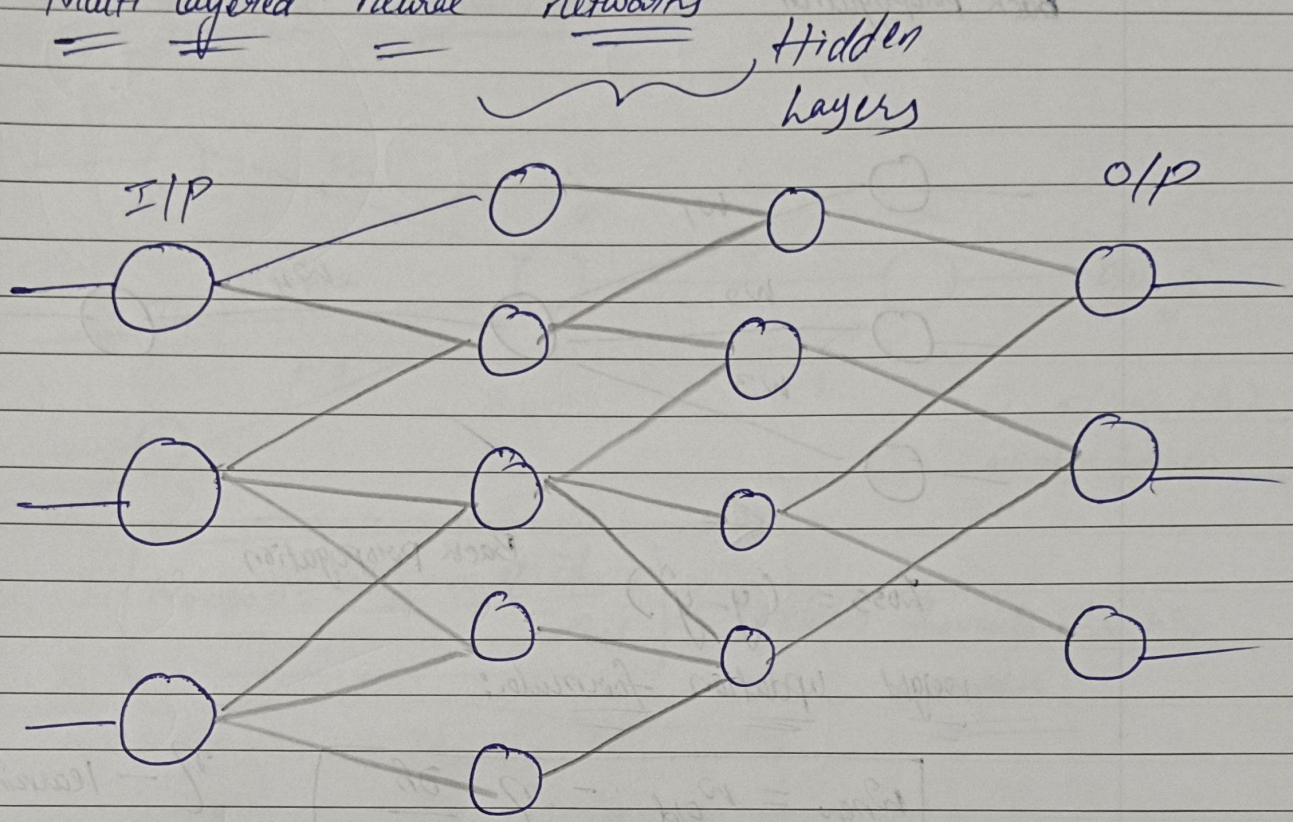
Gradient Descent → kind of optimizer



$y - \hat{y}$ = minimal

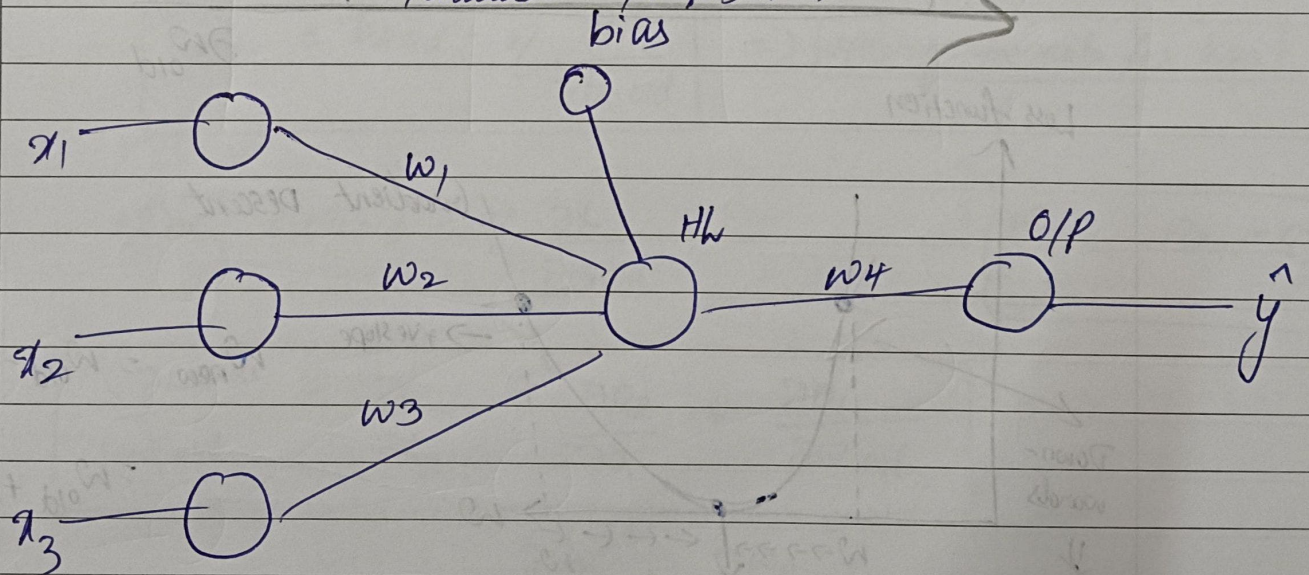
__/__/__

Multi layered neural networks



DAY-2

Forward propogation →



$$y = (w_1 x_1 + w_2 x_2 + w_3 x_3) + bias$$

$$y = w^T + b$$

$z = \sigma(y)$ → Sigmoid Activations

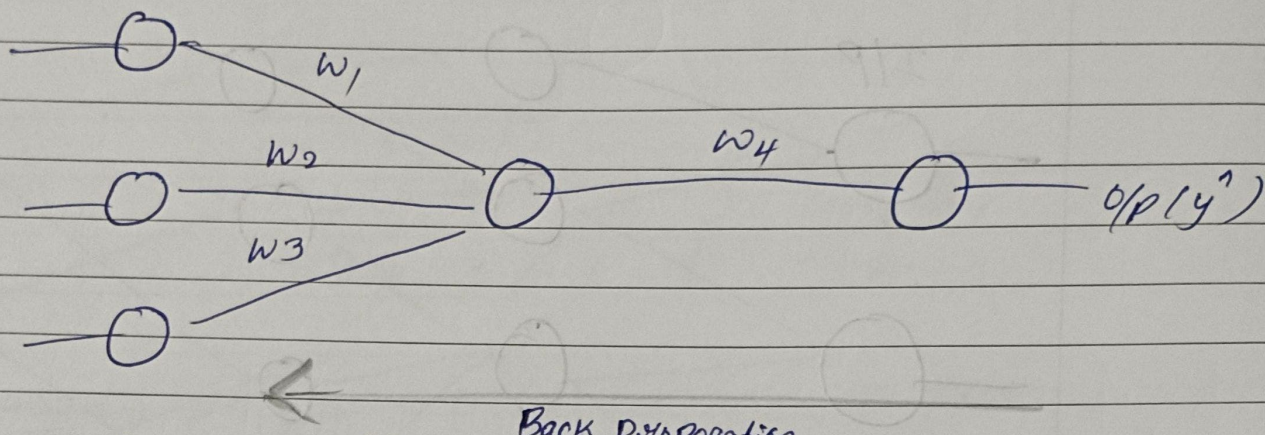
⇓ Non-linear problems

⇓ 0 or 1

$\geq 0.5 \rightarrow 1$

$< 0.5 \rightarrow 0$

Back propogation



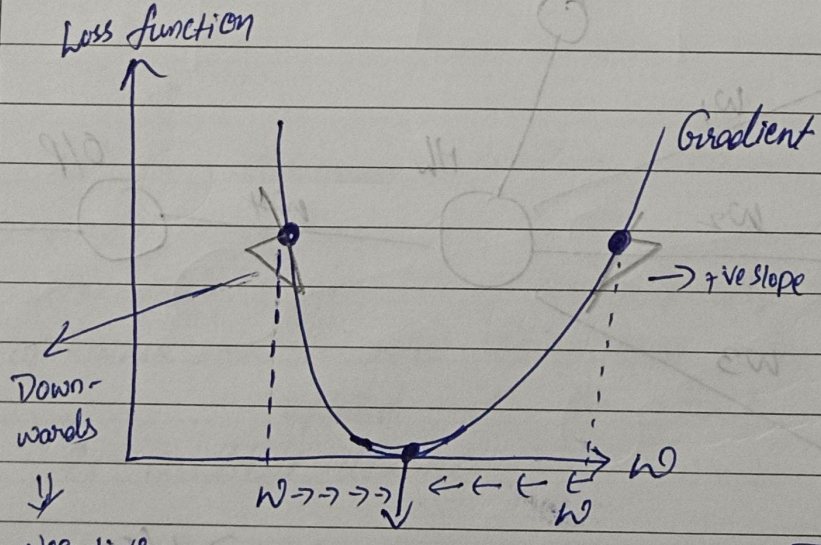
$Loss = (y - \hat{y})$

Weight updation formula:

$$w_{new} = w_{old} - \eta \frac{\partial h}{\partial w_{old}}$$

η — learning rate

$\frac{\partial h}{\partial w_{old}}$ — slope



$w_{new} = w_{old} - \eta(-ve)$

$= w_{old} + \eta(ve)$

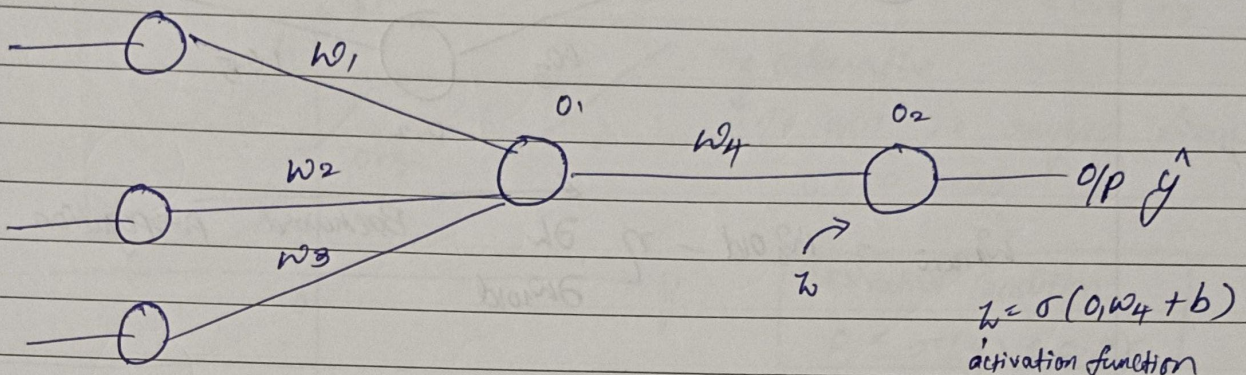
$w_{new} >> w_{old}$

$w_{new} = w_{old} - \eta(+ve)$

$w_{new} << w_{old}$

__/__/____

Chain Rule of Differentiation



$$w_{4new} = w_{4old} - \eta \frac{\partial h}{\partial w_{4old}}$$ → updating formula for w_4 weights

$$\frac{\partial h}{\partial w_{4old}} = \frac{\partial h}{\partial o_2} * \frac{\partial o_2}{\partial w_{4old}}$$

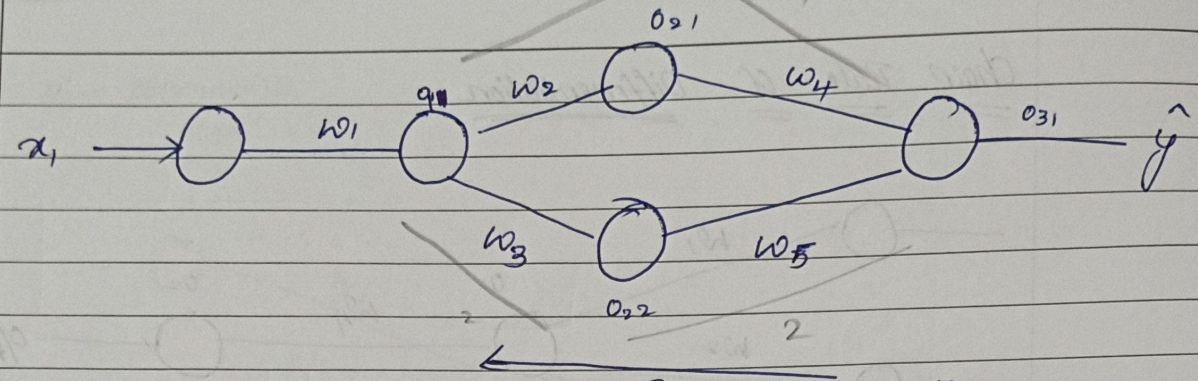
$$b_{2new} = b_{2old} - \eta \frac{\partial h}{\partial b_{2old}}$$ ⇒ updating formula for bias

$$w_{1new} = w_{1old} - \eta \frac{\partial h}{\partial w_{1old}}$$

$$w_{2new} = w_{2old} - \eta \frac{\partial h}{\partial w_{2old}}$$

$$\frac{\partial h \to loss}{\partial w_{1old}} = \frac{\partial h}{\partial o_2} * \frac{\partial o_2}{\partial o_1} * \frac{\partial o_1}{\partial w_{1old}}$$

$$\frac{\partial o_2}{\partial w_4} * \frac{\partial w_4}{\partial o_1}$$

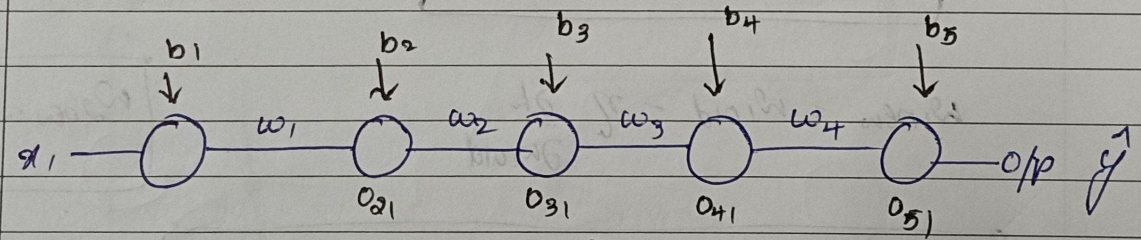


Backward propagation

$$w_{2new} = w_{2old} - \eta \frac{\partial L}{\partial w_{2old}}$$

$$\frac{\partial L}{\partial w_{1old}} = \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1old}}\right] + \left[\frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{22}} * \frac{\partial o_{22}}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{1old}}\right]$$

(3) Vanishing Gradient problem



$$\text{loss} = \frac{1}{2}(y - \hat{y})^2 \rightarrow \text{MSE}$$

$$w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_{1new}}$$

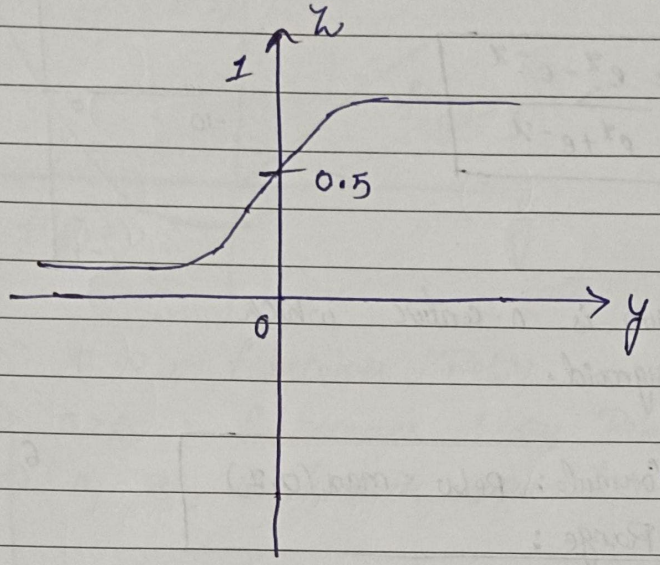
$$\frac{\partial L}{\partial w_{1new}} = \frac{\partial L}{\partial o_{51}} * \frac{\partial o_{51}}{\partial o_{41}} * \frac{\partial o_{41}}{\partial o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial w_1}$$

__/__/____

Sigmoid Activation formula $\boxed{y = \frac{1}{1+e^{-x}}}$ → 0 or 1

$> 0.5 \rightarrow 1$

$< 0.5 \rightarrow 0$



The derivative

↓

It will be ranging from 0 to 0.25

$\boxed{\text{Derivative condition: } 0 \leq \sigma(y) \leq 0.25}$

$O_{51} = \sigma[(O_{4i} * w_4) + b]$

↓

Sigmoid Activation

$W_{new} = W_{old} - \eta$ (small number) (η: small)

$\boxed{W_{new} \approx W_{old}}$

⇓

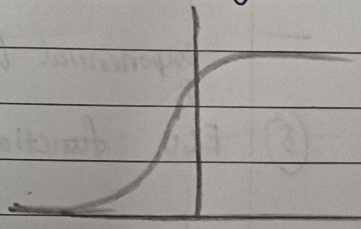
Vanishing gradient problem

Activation functions:

(1) Sigmoid function $\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$ Range: 0 to 1

=) The function output is not centered on 0, which will reduce the efficiency of weight update.

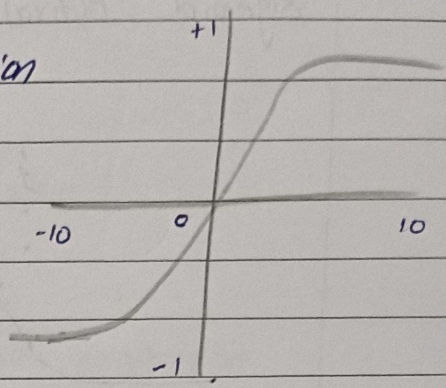
=) when the input is slightly away from the origin then the gradient is small or almost zero.



__/__/____

(2) Tanh function :

=> Also called hyperbolic tangent function



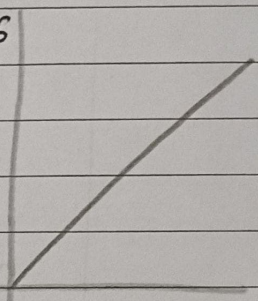
Formula : $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Range : -1 to 1

=> The whole function is 0-centric which is better than sigmoid.

(3) Relu function : formula : Relu = max(0, x)

Range :



=> Not 0-centric

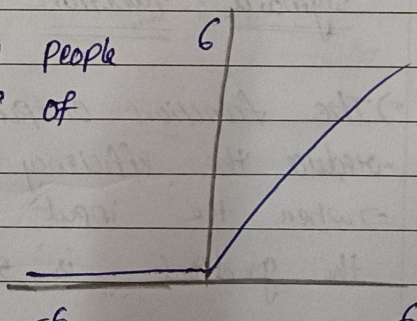
=> Better than sigmoid & tanh

=> calculation speed is much faster

=> when input is -ve, Relu completely inactive which means that once a -ve number is entered Relu will die

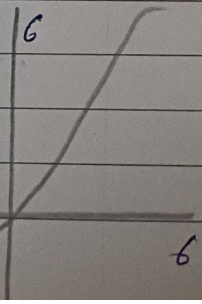
(4) Leaky Relu function : $f(x) = \max(0.01x, x)$

=> To Solve the Dead Relu problem people proposed to set the first half of Relu 0.01x instead of 0.



exponential linear unit

(5) Elu function : $f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases}$



=> Slightly more computationally intensive

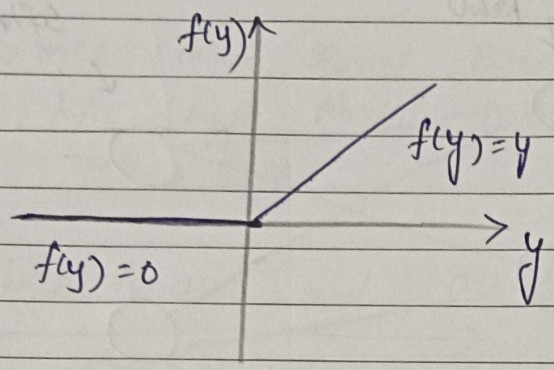
=> No Dead Relu issues

=> The mean of the output is close to 0, zero-centered

__/__/____

⑥ PReLU function:

$$f(y_i) = \begin{cases} y_i & \text{if } y_i > 0 \\ a_i y_i & \text{if } y_i \leq 0 \end{cases}$$



⇒ a = 0, f becomes ReLU

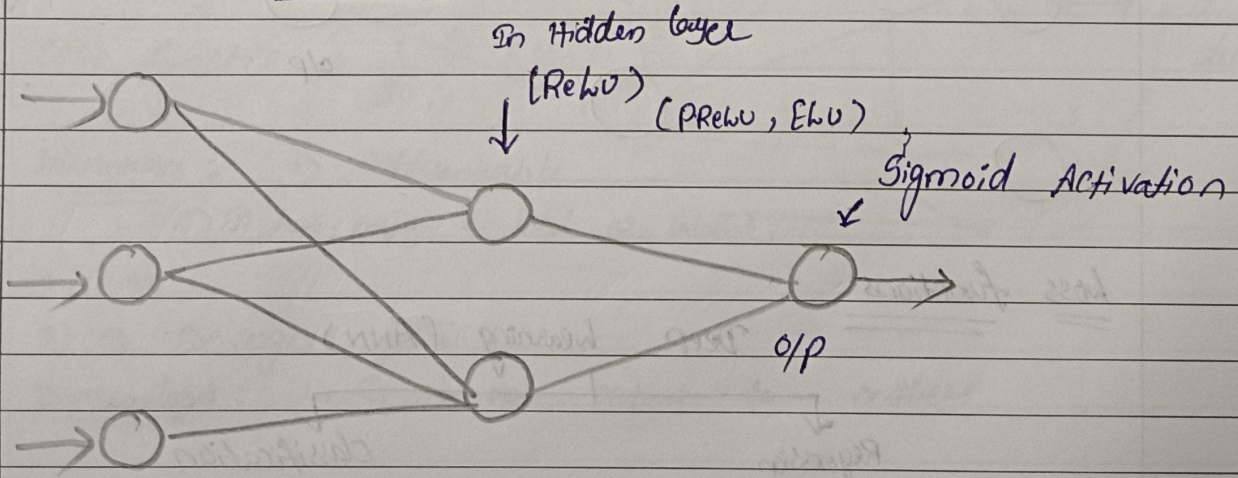
⇒ a > 0, f becomes leaky ReLU

⇒ a is a learnable parameter, f becomes PReLU

⑦ Swish (A Self-Gated) function;

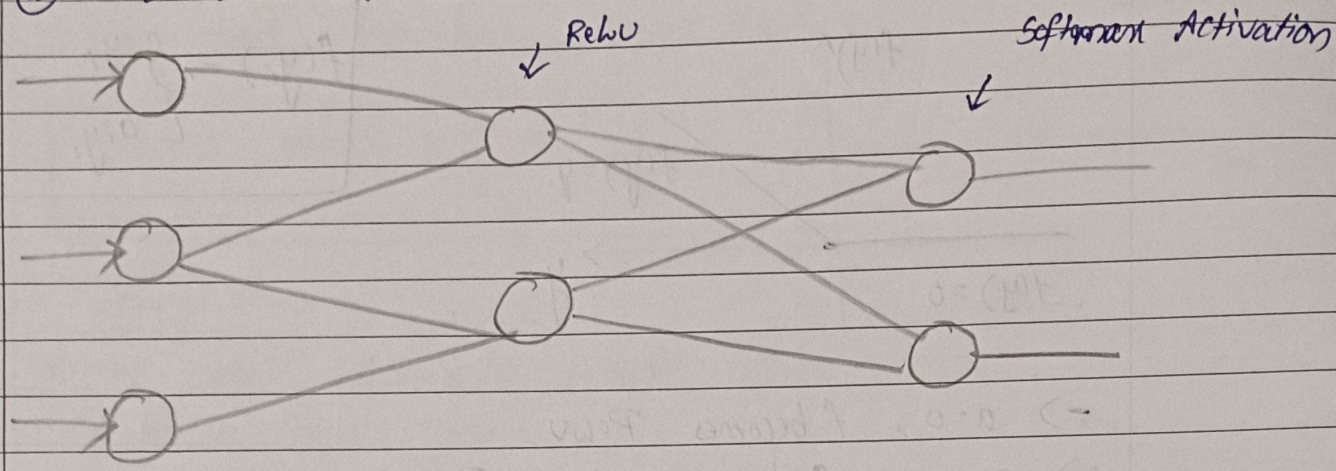
Technique which activation function we should use

① Binary classification



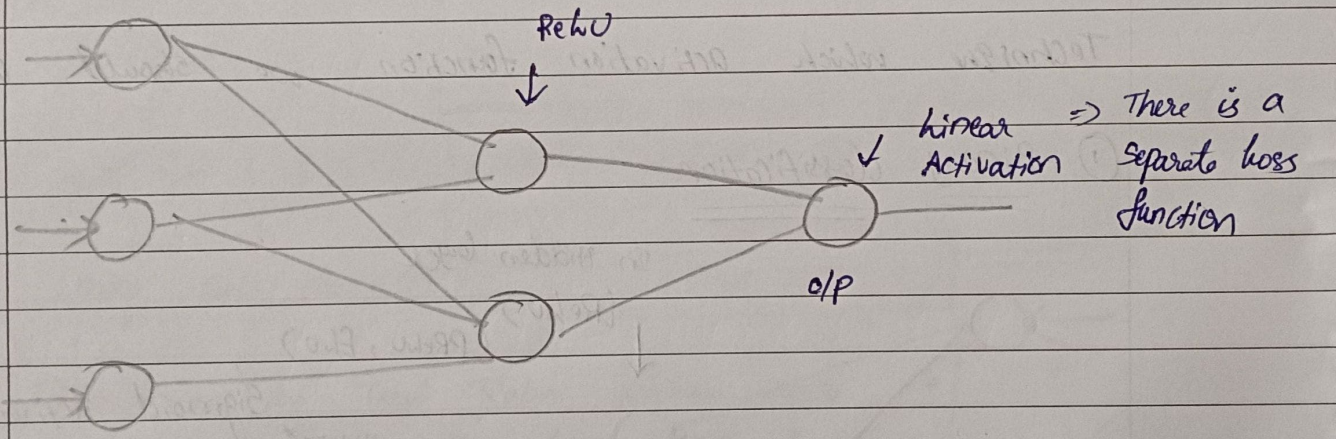
__/__/____

② Multi class classification



* In Hidden layers use ReLU maximum it solves

③ Regression



loss functions

Deep learning (ANN)

Regression

Experience	Degree	salary (o/p)
10	Phd	-
-	-	-
-	-	-
-	-	-

Regression Problem

↓ continuous values

Classification

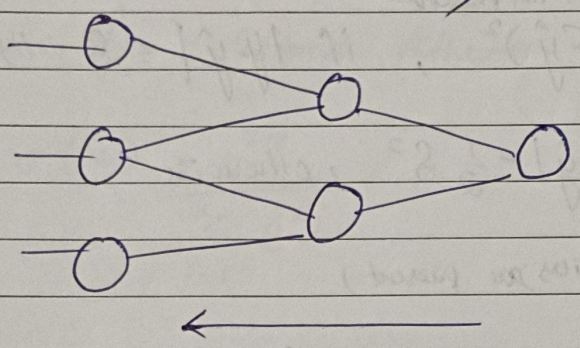
Play	Study	pass / Fail
10	2	F
4	3	F
5	5	P
2	7	P

__/__/____

① Regression

(i) MSE (Mean Squared Error)
(ii) MAE (Mean Absolute Error)
(iii) Huber Loss

Loss function and cost function



Dataset = 100 records

$\hat{y}$ Loss $= \frac{1}{2}(y-\hat{y})^2$

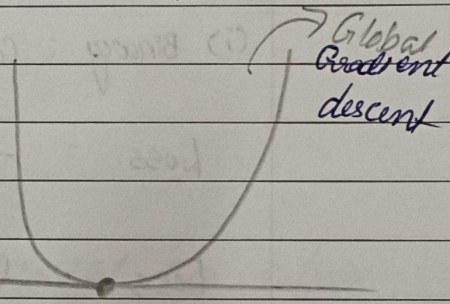
Cost $= \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})^2$

↓

(Provide batch of data points)

(i) MSE

Loss function $= \frac{1}{2}(y-\hat{y})^2$ → Quadratic eqn



Cost function : $\frac{1}{2n}\sum_{i=1}^{n}(y-\hat{y})^2$

Advantages : 1) Differentiable
2) It has only 1 local or Global minima
3) It converges faster

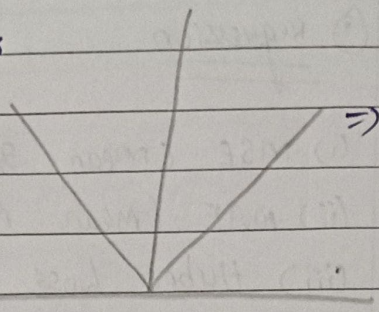
Disadvantages : 1) It is not robust to outliers

(ii) MAE

Loss function $= \frac{1}{2}|y-\hat{y}|$

Cost Function $= \frac{1}{2n}\sum_{i=1}^{n}|y-\hat{y}|$

Advantages: 1) Robust to outliers



=> Time Consuming

(3) Huber loss : Combination of MSE and MAE

(outliers not present)

$$loss = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & \text{if } |y-\hat{y}| \le \delta \rightarrow \text{Hyperparameter} \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

(outliers are present)

Classification :-

1) Cross Entropy → Binary Cross Entropy (Binary classification)
→ categorical Cross Entropy (multi class classification)

(i) Binary Cross Entropy :

$$loss = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$ (Logistic Regression)

$$loss = \begin{cases} -\log(1-\hat{y}) & \text{if } y=0 \\ -\log(\hat{y}) & \text{if } y=1 \end{cases}$$ (Binary classification) $\hat{y} = \frac{1}{1+e^{-x}}$

(ii) Categorical Cross Entropy : (Multiclass classification)

f_1	f_2	f_3	O/P	Good	Bad	Neutral
2	3	4	Good	1	0	0
5	6	7	Bad	0	1	0
8	9	10	Neutral	0	0	1

__/__/____

$$\text{Loss } (L(x_i, y_i)) = \sum_{j=1}^{c} y_{ij} * \ln(\hat{y}_{ij})$$

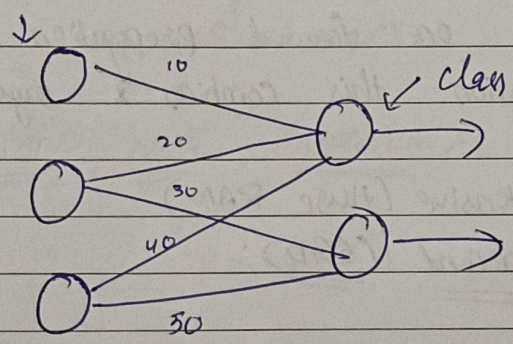
i = Row
j = column

$y_i = [y_{i1}, y_{i2}, y_{i3}, \ldots\ldots y_{ic}]$ → category

$$y_j = \begin{cases} 1 & \text{if the element is in class} \\ 0 & \text{Otherwise} \end{cases}$$

$\hat{y}_{ij}$ = Softmax Activation → o/p layer

$$\sigma(z) = \frac{e^{z_e}}{\sum_{j=1}^{K} e^{z_j}}$$



$$= \frac{e^{10}}{e^{10+20+30+40+50}}$$

Conclusion:-

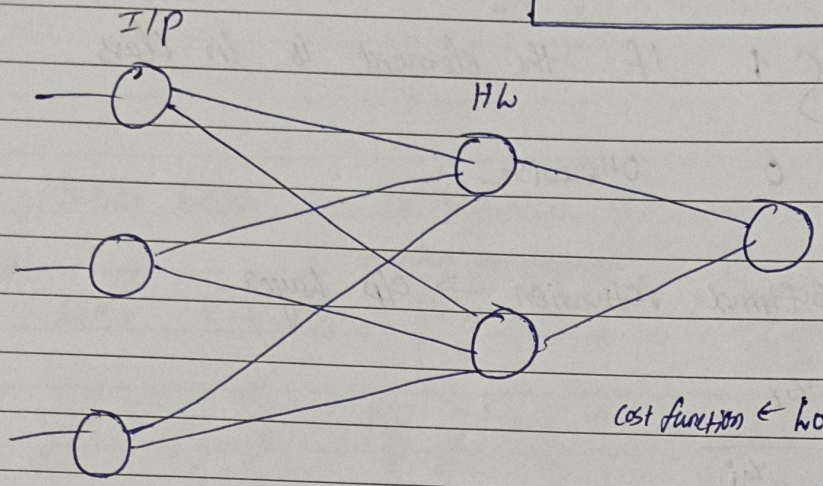
① Relu, Softmax => Multiclass } → Categorical Cross Entropy
② Relu, Sigmoid => Binary } → Binary Cross Entropy

Relu, linear Activation → MSE, MAE, Huber loss

__/__/____

Day-3 ① Gradient Descent :-

weight updation formula : $\boxed{w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}}$



cost function ← Loss $= \frac{1}{2n} \sum_{i=1}^{n} (y - \hat{y})^2$

Epoch : when ever we do one forward propogation and one backward propogation this combines & says one epoch.

Disadvantage : 1) Resource extensive (Huge RAM)

② Stochastic Gradient Descent (SGD) :-

Epoch 1

1 record → $\hat{y}$ → Loss → } → Iteration 1

update weights ←

2 record → } → Iteration 2

⋮

3 Million Iteration

Disadvantage :-

1) Convergence will be very slow

2) Time complexity will also be high

__/__/____

(3) Mini batch SGD :-

1000000 batch size = 1000

epoch - 1

1000 → ← } Iteration 1

→ ← } Iteration 2

⋮

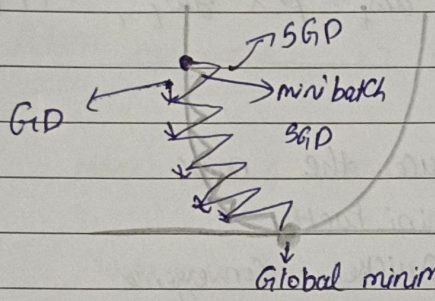
} Iteration 1000

(i) Advantages :-

(i) Resource Intensive reduced

(ii) Convergence will be better

(iii) Time complexity will be improved



SGD having highest Noise

(4) SGD with Momentum :- Reduce the noise and comes to the global minima smoother the journey.

Exponential weighted Average

⇓

Time series

⇓

ARIMA, ARMA

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

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

Exponential weighted Average

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

t_1 t_2 t_3 t_4 ..... t_n

a_1 a_2 a_3 a_4 ..... a_n } forecasting

$v_{t1} = a_1$

$v_{t2} = \beta * v_{t_1} + (1-\beta) * a_2$

$\beta \Rightarrow$ Hyper parameter

↓

In which value we should focus more

__/__/____

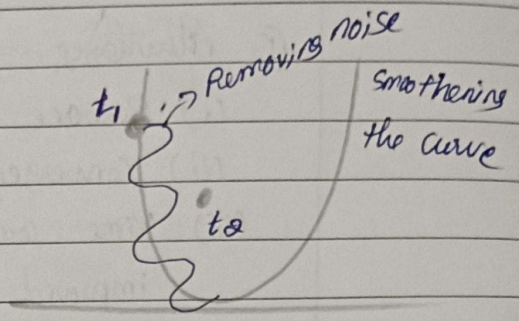
$\beta = 0$ to 1

$\Downarrow$

0.95

$V_{t_2} = (0.95) * V_{t_1} + (0.05) * a_2$

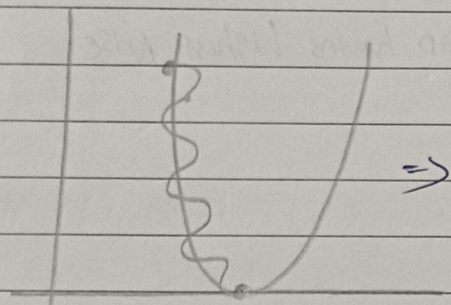
$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$



Exponential weighted Average

$$w_t = w_{t-1} - \eta V_{dw}$$

$$V_{dw_t} = \beta \times V_{dw_{t-1}} + (1-\beta) * \frac{\partial h}{\partial w_{t-1}}$$



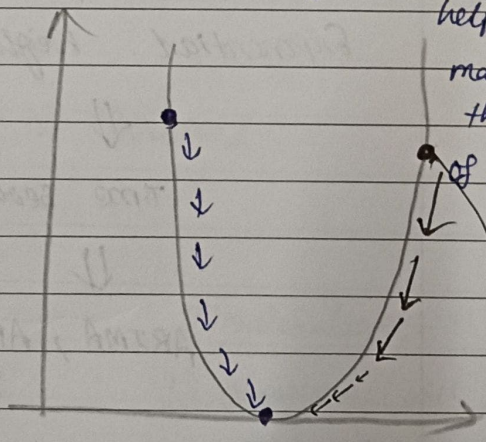
$\Rightarrow$ Reduce the noise
mini bacth
Quicker convergence

③ Adagrad → Adaptive Gradient Descent

$$w_t = w_{t-1} - \eta \frac{\partial h}{\partial w_{t-1}}$$

η = fixed $\xrightarrow{\text{making}}$ adaptive

η = learning rate helps to maintain the speed of convergence



$$w_t = w_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

ϵ - Small number (denominator never become zero)

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial h}{\partial w_t}\right)^2$$

t - Current timestamp

η decreases as we are reaching the global minima

__/__/____

Adadelta and RMSprop

$$\eta' = \frac{\eta}{\sqrt{sd_w + \epsilon}}$$

$$sd_w = 0$$

$$sdw_t = \beta\, sdw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

$$\beta = 0.95$$

$$sdw_t = (0.95)\, sdw_{t-1} + (0.05)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

Adam optimizer => Best optimizer

Momentum + RMS prop (Adaptive learning Rate)

$V_{dw}=0 \quad V_{db}=0 \quad S_{dw}=0 \quad S_{db}=0$

For weight =) $w_t = w_{t-1} - \eta' V_{dw}$

For bias =) $b_t = w_{bt-1} - \eta' V_{db}$

$$\eta' = \frac{\eta}{\sqrt{sd_w + \epsilon}}$$

$$V_{dw_t} = \beta \times V_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

(1) Smoothening
(2) learning rate adaptive

$$Vdb_t = \beta \times Vd_{bt-1} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

ANN (Artifical Neural Networks)

(1) For which all algorithms feature scaling is required?

ANN =) Any algorithm that related to some distance based problems
LR
Log R =) Gradient Descent
KNN
KMeans

__/__/____

Tensorflow => popular library => Google
Tensorflow —wrapper→ Keras > 2.0 => They both got integrated

Dense => we can create I/p, O/p, Hlw
Sequential =)
Dropout =)

Black Box Model vs White Box Model

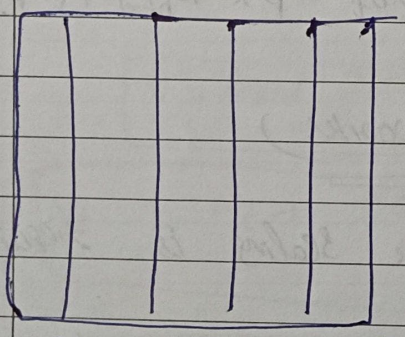
Random Forest → Black Box Model ANN, xgboost
Decision Tree → White Box Model linear regression

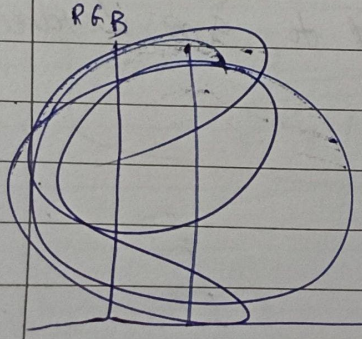
CNN (Convolutional Neural Networks) => Image, Video frame

CNN

=> convolution

Images =>
Black & white



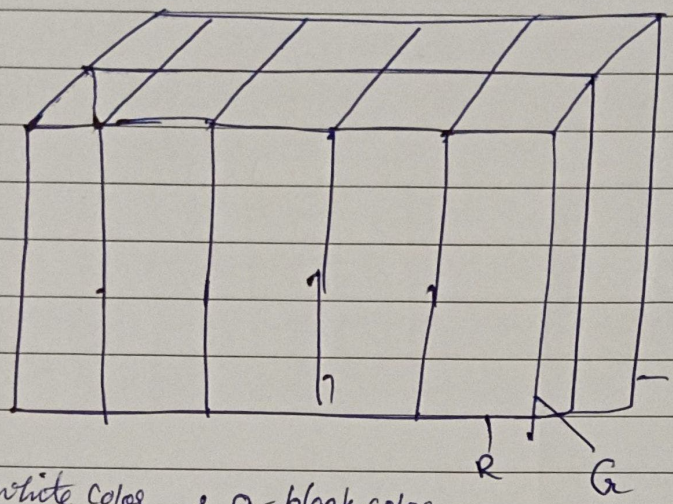


Visual Cortex

V_1 → Moving things
↓
V_2 — Animals (Cat)
↓
V_3 — Map the Environment
↙ ↘ →
V_4 V_5 V_6
↘ ↙
V_7
O/p

__/__/____

RGB



0 - 255

255 - white color , 0 - black color

Convolution

Image

Stride = 1

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

6X6

min max scaling

*

Filter / kernel

1	2	1
0	0	0
-1	-2	-1

3X3

⇒

O/P

0	0	0	0
-4	-4	-4	-4
-4	-4	-4	-4
0	0	0	0

4X4

⇓

Horizontal edge filter

Changing the filter =>

1	0	-1
2	0	-2
1	0	-1

⇒

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

⇓

vertical edge filter/kernel

padding => Build the Compound around the image

$n - f + 1 = 6 - 3 + 1 = 3 + 1 = 4$

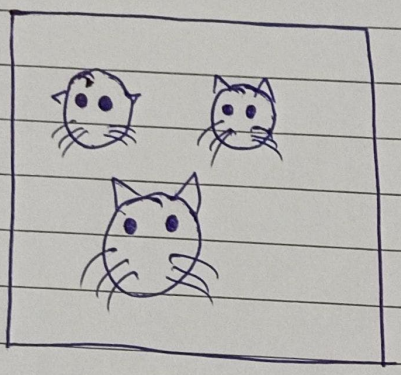
After padding => $8 - 3 + 1 = 6$

padding filter

n=6, p=1, f=3 → n+2p-f+1 = 6+2-3+1 = 6

Max pooling

Avg pooling
min pooling
Max pooling



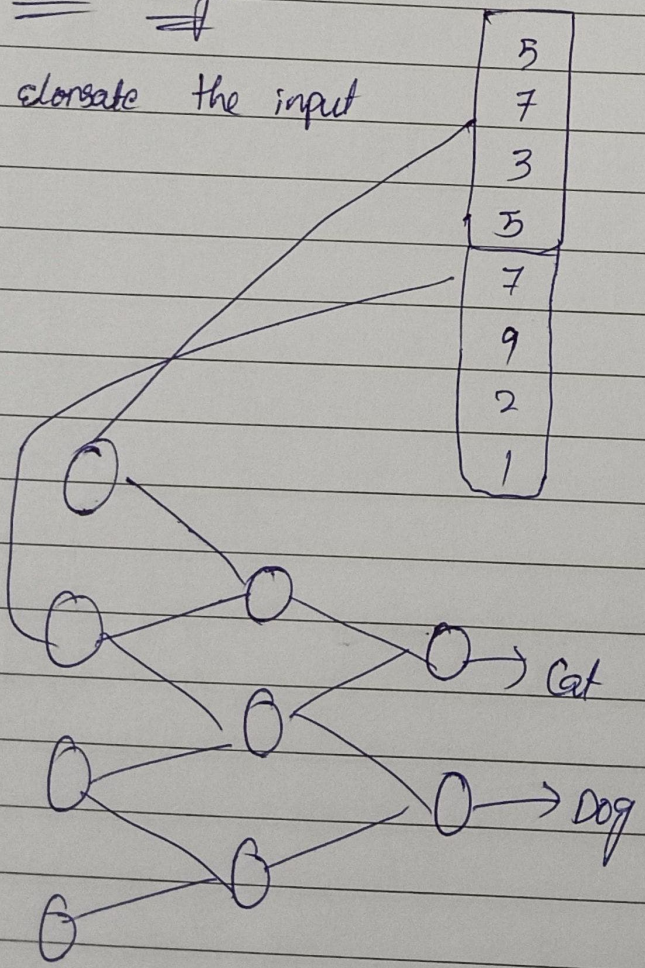
1	0	3
4	5	7
0	3	5

5	7
3	5

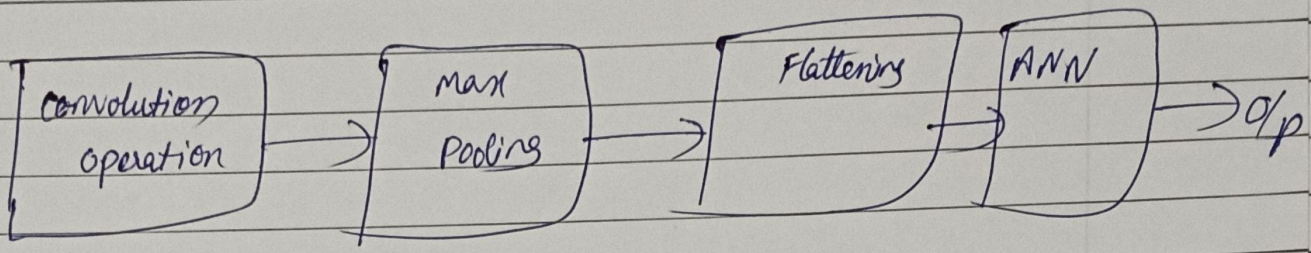
7	9
2	1

Flattening layer

elongate the input



__/__/____


convolution operation
max Pooling
Flattening
ANN
o/p