1 * AI vs ML vs DL vs DS :-

AI : It is an application, able to do it own task without human interaction.

e.g :- Netflix has its own AI Tool

① ↳ Let'say, ~~today~~ frequently you have watched action movies, & then next time it will give reccommendation of action movie automatically.

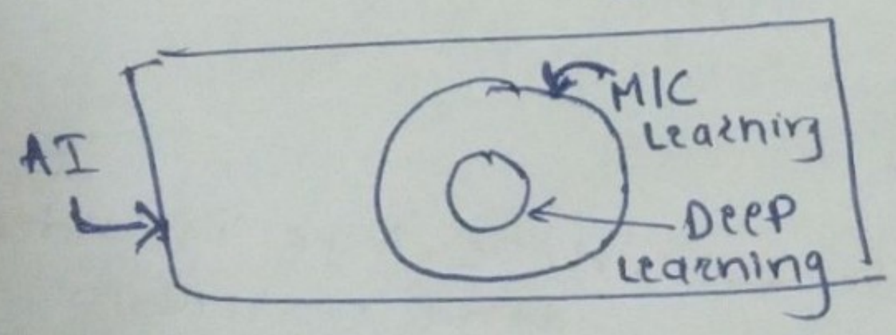
② ↳ Amazon.in → Iphone → → → Headphone.
recommend you

③ ↳ self driving car → Tesla.

* simply AI tool understand users behaviour and it is being able to do its task by own without human iteraction.

2 Machine Learning :- M/c It is a subsets of AI

M/c Learning provides - ① stats Tools to analyze data
↳ visualization
↳ prediction
↳ forecasting



3 Deep Learning :- It is a subsets of M/c Learning

In 1950-60, scientist thought that can we make machine Learning like how we human being learn

↳ Here plan is to mimic a haman brain, to implent something, to learn something.

↳ So for this, we use Multi-Layer Neural network [that's why Deep] It is a subset of M/c Learning. used to Mimic human brain.

↳ This Mult-layered Neural Network is used help train Machine or application.

↳ with help of deep learning we are able to solve complex problems

* Classification :-

No. of hours study	No. of playing hours	No. of sleeping hours	— ? some

[4] **Data science** :- It contain star blue, power BI, tableu to Visualize & Analyze data.

- Machine Learning & Deep Learning
  - Supervised M/L
    - Regression
    - classification
  - unsupervised M/L
    - Clustering
    - Dimensionality Reduction

* **supervised M/L** :- Here we will consider following Dataset

Age	weight
24	62
25	63
21	72
27	62

Age → [Hypothesis] → o/p weight (Machine)

Independent variable → Here Age is independent variable
&
Dependent variable → weight is a dependent variable.

↳ It means that weights is depends upon age.

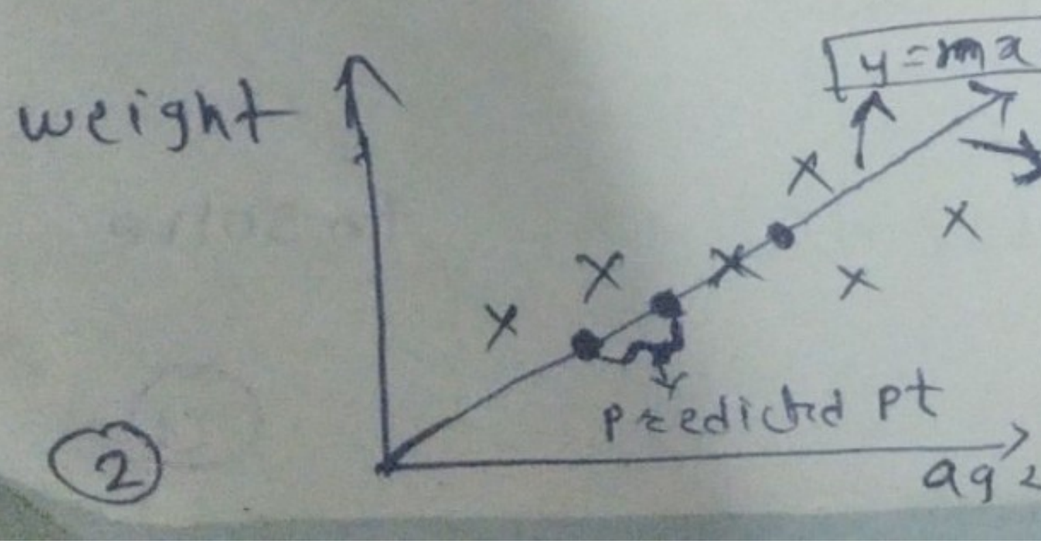
* **Regression problem** :-

Age	weights
24	72
23	71
25	71.5

weights → o/p var (dependent var)

It contain continous variable

Regression problem



Straight Line indicate Linear regression & it has Formula $y = mx + c$

↳ With help of formula or equation we will try to find predicted point (•)

* Classification :-

No. of hours study	No. of playing hours	No. of sleeping hour	O/P (P/F)
—	—	—	P pass
—	—	—	F fail
—	—	—	P

Independent variable — Dependent variable

↳ Whenever you have a output, with fix number of categories, then that become a classification problem

↳ suppose you have, just 2 types of output then it becomes Binary classifications.

↳ suppose, if you have more than 2 different categories then it is called as Multiclass classification.

* Unsupervised ML :-

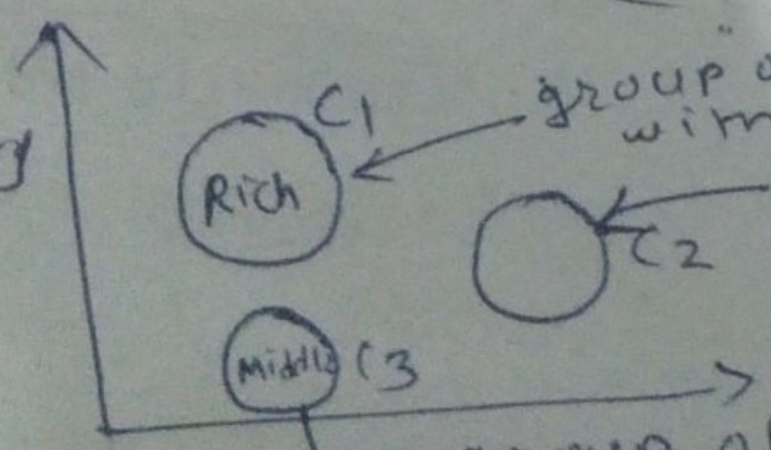
↳ Lets say my Dataset it contain salary & Age, but it does not contain output variable or Dependent var

Salary	Age	
—	—	No Dependent variable
—	—	

↳ In this case, i would like to use clustering → In clustering i am going to do customer segmentation.

↳ In clustering based on data present in table, i am going to do different clusters Findout similar groups.

* Grouping



group of young people with amazing salary

group of more age people & amazing salary.

group of middle class background with respect to age not getting amazing salary.

* **Dimensionality Reduction** :-

↳ suppose, we will have 1000 Features

1000 $\xrightarrow{\text{can we Reduce}}$ To lower Dimension (100) Dimension.

* **Hyper parameter** :

↳ Parameter ⟹ Model ↓ Training Data

* In a neural network weights (w), bias (b) all are parameters.
* While Training neural network we need to specify no. of neurons in hidden layers, How many layer will require, How many epoch you want it to train on all are called Hyperparameter.

Defination :- Most of M/C learning Algorithm have hyper-parameters, settings that can used to control algorithm's behaviour.

* parameter's are something learnt during machine learning process.

, whereas Hyperparameter is something you need to mannually specify.

e.g :- λ - This value is used to specify the strength of weight decay it is also example of Hyper parameter.

* The values of Hyperparamter are not adapted by learning algorithm itself (need to specify mannually).

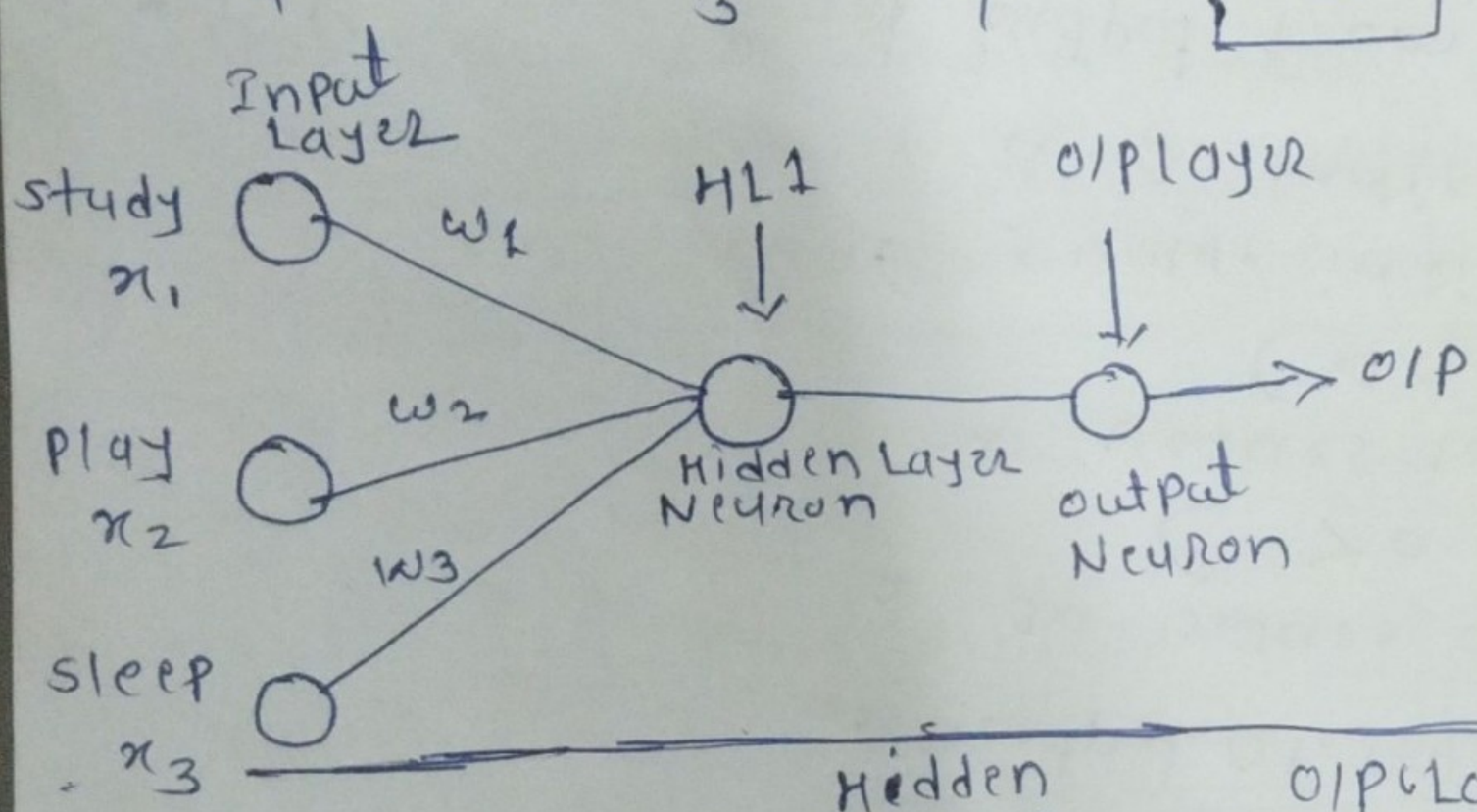
* **Perceptron :-** [single Layer Neural N/W] & [Multi Layer Neural N/W]

* **Dataset** of student :-

study hours	play hours	sleep h/r	Pass/Fail
7	3	7	1
2	5	8	0
4	3	7	1

→ Truth value = y

Problem Given is :- Binary classification problem



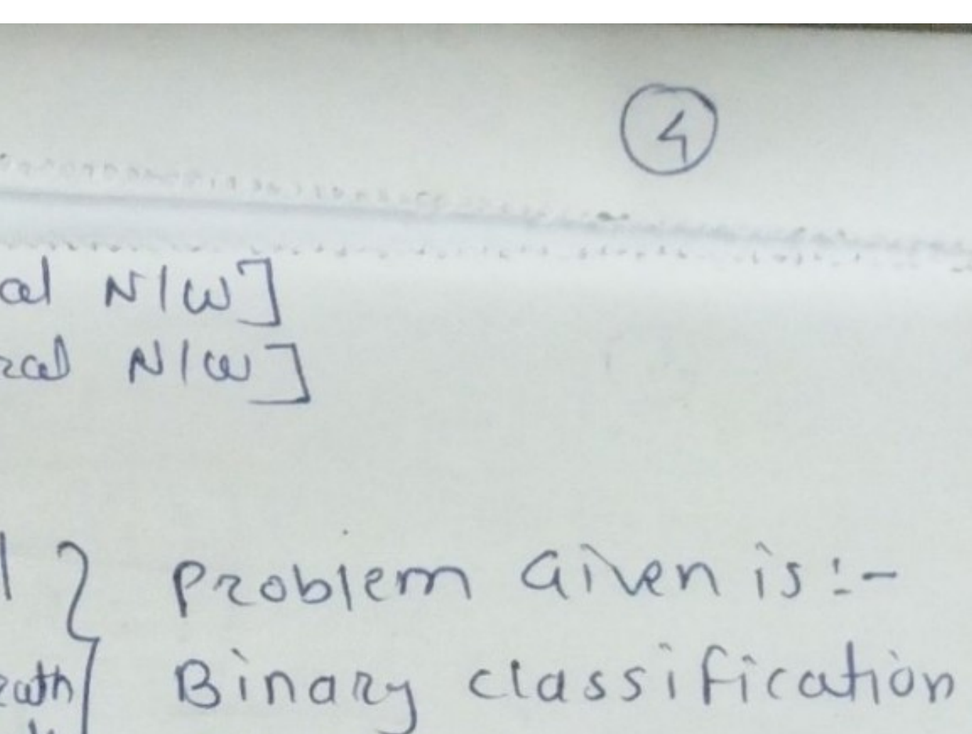


Diagram Multi Layer

Entire process is called Forward propagation.

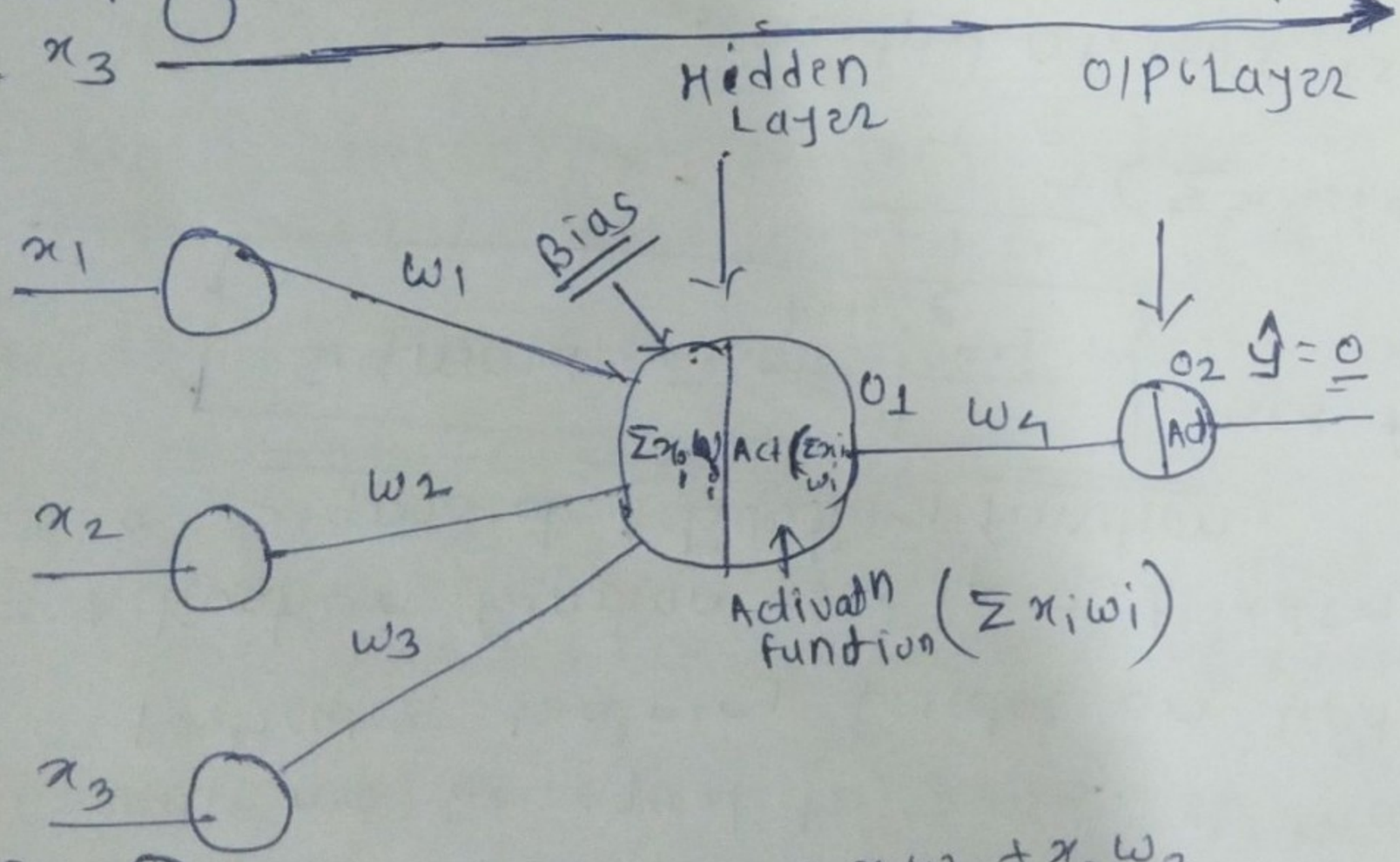


Fig. (A)

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

→ (1) Took input
→ (2) Multiply with weights
→ (3) Added a Bias
→ (4) Activate called As a

* **Forward Propagation**

* **weight** are used to (help) **activate** the **Neuron**
* During the Training of Neural Network, we will be Taking care ~~of~~ that, which weight has what value, so that Neuron should be activated at that level.
* so weight indicates that **how much Neurons should be** activated or deactivated.

* Let's say, weights are initialized with '0' (zero) value.

↳ $\Sigma x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$

$= x_1 \times 0 + x_2 \times 0 + x_3 \times 0$

$= 0$

↳ So we are passing '0' (zero) input, & that should not happen to avoid this, we will be adding <u>Bias</u>.

↳ So Bias will have some value, that some value it will not be equal to zero, so that we can continue particular training & later on updates the weight values.

↳ Activation function used in a Hidden Layer Neuron's is a <u>sigmoid Activation function</u>.

$$\boxed{\text{Sigmoid} = \frac{1}{1+e^{-y}}}$$ used for Binary classificatn

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

= output of this is equal to <u>0</u> or <u>1</u>.

= so values greater 0.5 taken as 1 ($> 0.5 \rightarrow 1$)

& values less than 0.5 taken as 0 ($< 0.5 \rightarrow 0$)

= 1 value indicates Neuron is activated & 0 indicates not activated.

* Let's say i got output from forward propagation is <u>0</u>. so <u>0</u> will be a predicted value i.e $\boxed{\hat{y} = 0}$

but for input <u>7 (study)</u>, <u>3 (play)</u> & <u>7 sleep</u> i got truth value $= \boxed{1}$ in Table. so $\boxed{y = 1}$

* so $(y - \hat{y}) = 1 - 0 = \boxed{1}$

* since truth value is 1, so it mean's that i am getting completely wrong output from model. so i have to make sure that $\boxed{(y - \hat{y})}$ this <u>difference</u>

$= \boxed{1}$

* Since truth value is 1, so it mean's that i am getting completely wrong output from model. so i have to make sure that $\boxed{(y - \hat{y})}$ this difference is very close to 0 (zero). It means we should get predicted value & truth values are same.
* So we, should consider over hear *loss function. Loss value are used to findout difference between predicted value & Truth value.

$$\boxed{\text{Loss function} = \text{predicted value} - \text{Truth value}}$$

$$= \hat{y} - y$$

* Now $\hat{y} - y$ this difference is huge, next thing i will do is *Back propagation.
* Main aim of Back propagation is updats the weights. to reduce the difference. $(\hat{y} - y)$
* so in Back propagation, weights are updated with the help of *optimizers.
* These optimizers make sure that each & every weights will updated while back propagating.

1. I/P Layers
2. weights
3. Bias is added
4. Activation function
5. Loss function $\{\hat{y} - y\}$ ↓↓↓
6. optimizers
7. update the weight

} Forward propagation

} Backward propagation

↳ ~~So~~ η - Learning rate should be a small number.
~~So~~ If it is a small number we will be slowly converging in a global Minima.

↳ e.g: $\eta = 0.001$ or $\eta = 0.01$

2 * Chain Rule of Differentiation

↳ consider how w_4 is updated.

$$\boxed{w_{4new} = w_{4old} - \eta \frac{\partial L}{\partial w_{4old}}}$$ .. see the fig. A

$$\boxed{\frac{\partial L}{\partial w_{4old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial w_{4old}}}$$ weight updatation Formula } * chain Rule To update w_4 weight

Loss basically depend upon $\boxed{O_2}$ that's why

$O_2 = Z = \overset{\text{Activation fn Applied}}{\sigma}(O_1 w_4 + b)$

$$\boxed{b_{2new} = b_{2old} - \eta \frac{dL}{\partial b_{2old}}}$$ Bias updation formula

* Let's consider w_1 & it's updations using chain rule.

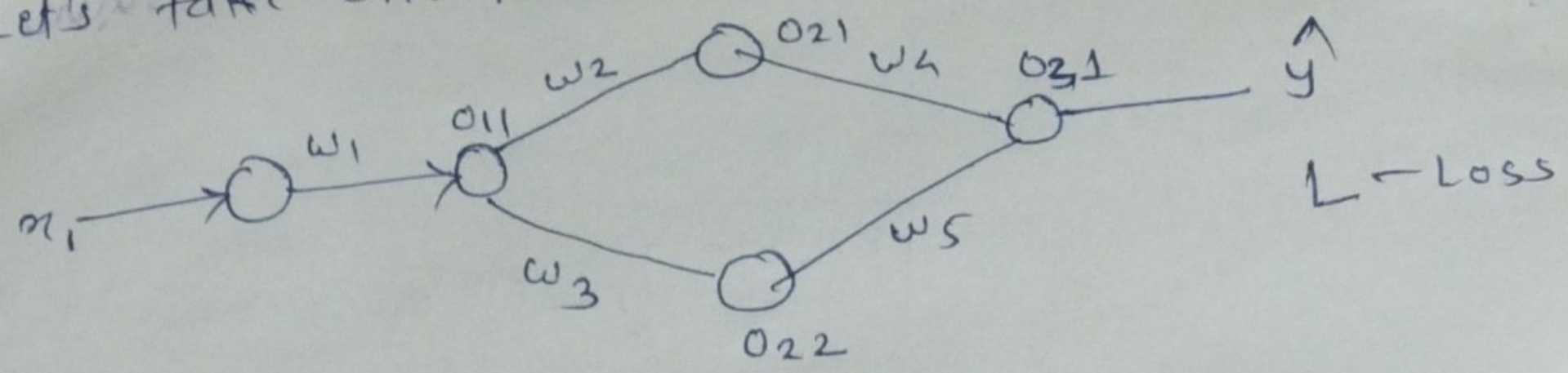
$$\boxed{w_{1new} = w_{1old} - \eta \boxed{\frac{dL}{\partial w_{1old}}}}$$

$$\boxed{\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_{1old}}}$$ chain rule to update w_1 weight

Loss $\xrightarrow{\text{depend}}$ upon $O_2 \xrightarrow[\text{upon}]{\text{depend}} O_1 \xrightarrow[\text{upon}]{\text{depend}} w_1$

"g. $\boxed{w_{2new} = w_{2old} - \eta \frac{\partial L}{\partial w_{2old}}}$

Let's take one more example



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

$$\frac{\partial L}{\partial w_{1old}} = \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}\right] \longrightarrow \text{1-path}$$

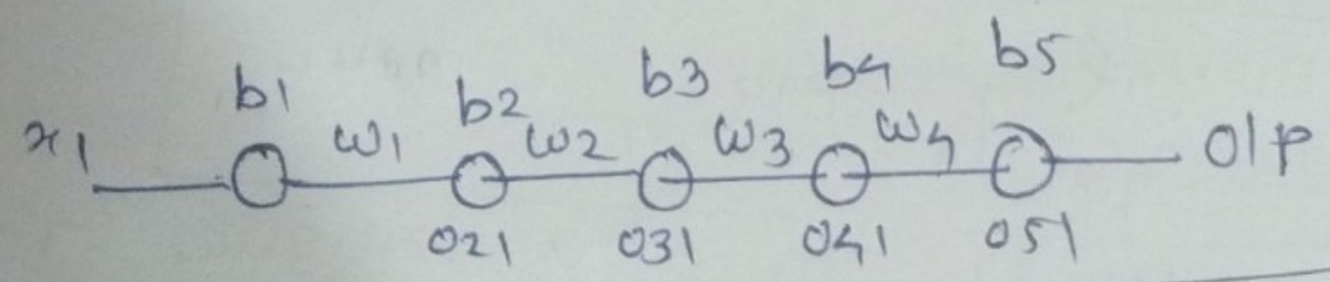
$$+$$

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

* so This happen in a Backward propagation called as chain rule of diffentiation.

3 Vanishing Gradient Problem :-

Let's Take one more example



sigmoid A

$$o_{51} = \sigma\left[(o_{41} + w_4) + b\right]$$

$$\boxed{\text{Mean square Error} = \text{Loss} = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \hat{y})}$$

* Now try to update weight w_1

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

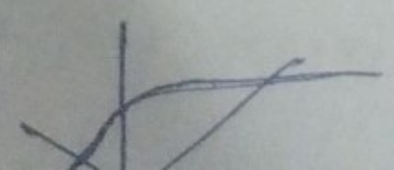
$$\frac{\partial L}{\partial w_{1new}} = \frac{\partial L}{\partial o_{51}} \times \frac{\partial o_{51}}{\partial o_{41}} \times \frac{\partial o_{41}}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{1old}}$$

Now consider sigmoid activation function

$$\boxed{y = \frac{1}{1+e^{-x}}} \longrightarrow \text{output Fall's in either } 0 \text{ or } 1$$

$\geq 0.5 \rightarrow 1$

$\leq 0.5 \rightarrow 0$



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

* Now try to update weight w_1

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

$$\boxed{\frac{\partial L}{\partial w_{1new}} = \frac{\partial L}{\partial o_{51}} \times \frac{\partial o_{51}}{\partial o_{41}} \times \frac{\partial o_{41}}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{1old}}}$$

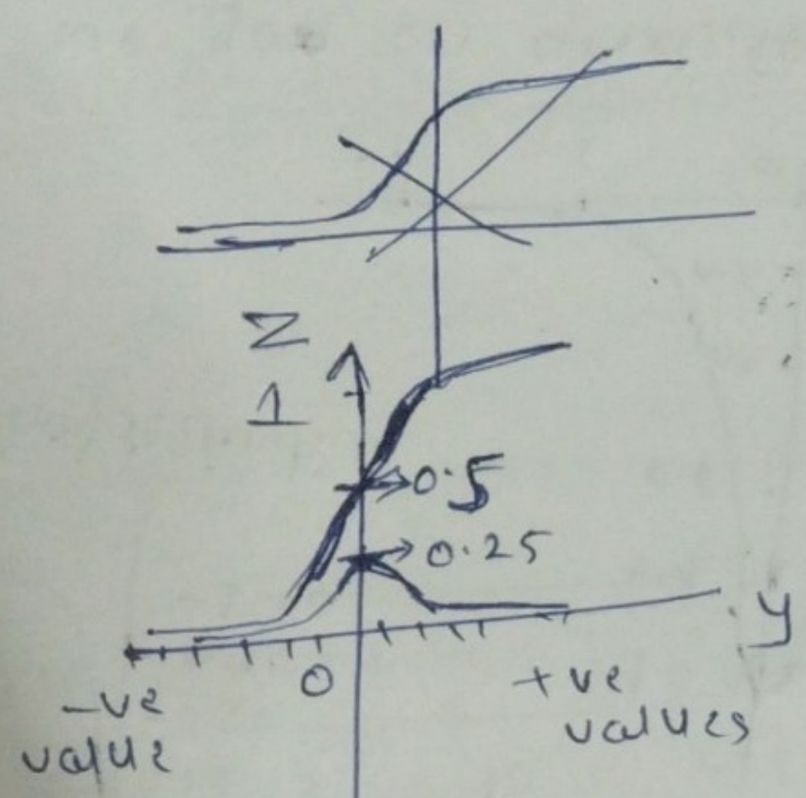
(9)

Now consider sigmoid activation function

$$\boxed{y = \frac{1}{1+e^{-x}}} \rightarrow \text{output fall's in either } 0 \text{ or } 1$$

$\geq 0.5 \rightarrow 1$

$\leq 0.5 \rightarrow 0$



* Always derivative of sigmoid (fall's in) or ranging in between $\boxed{0 \text{ to } 0.25}$

$$\boxed{0 \leq \sigma(y) \leq 0.25}$$

zero — sigmoid only activation fn

* Derivative of sigmod

$o_{51} = \sigma[(o_{41} + w_4) + b]$

$o_{41} = \sigma[(o_{31} + w_3] + b]$

It means that at every stage, we should use sigmoid activation function that has values in between 0 to 0.25

↳ so at each stage value keep on decreasing

e.g $0.025 \times 0.015 \times 0.010 \times 0.05 \times 0.012$

↳ Now, since we are multiplying smaller values at each stage, don't you think, we will be getting very small value.

$$w_{new} = w_{old} - \eta \text{ (small value)} \rightarrow$$

small

$\boxed{w_{new} \approx w_{old}}$ at one point of time $w_{new} = w_{old}$ (10)

where weight are not getting changed so this no change in weight called vanishing gradient. so use another activation

* **Advantage of sigmoid Activation function**

1. smooth gradient, & prevent jumping o/p or
2. output value between 0 & 1, normalizing each Neuron
3. clear prediction & very close to 1 or 0.

* **Disadvantages of sigmoid:**

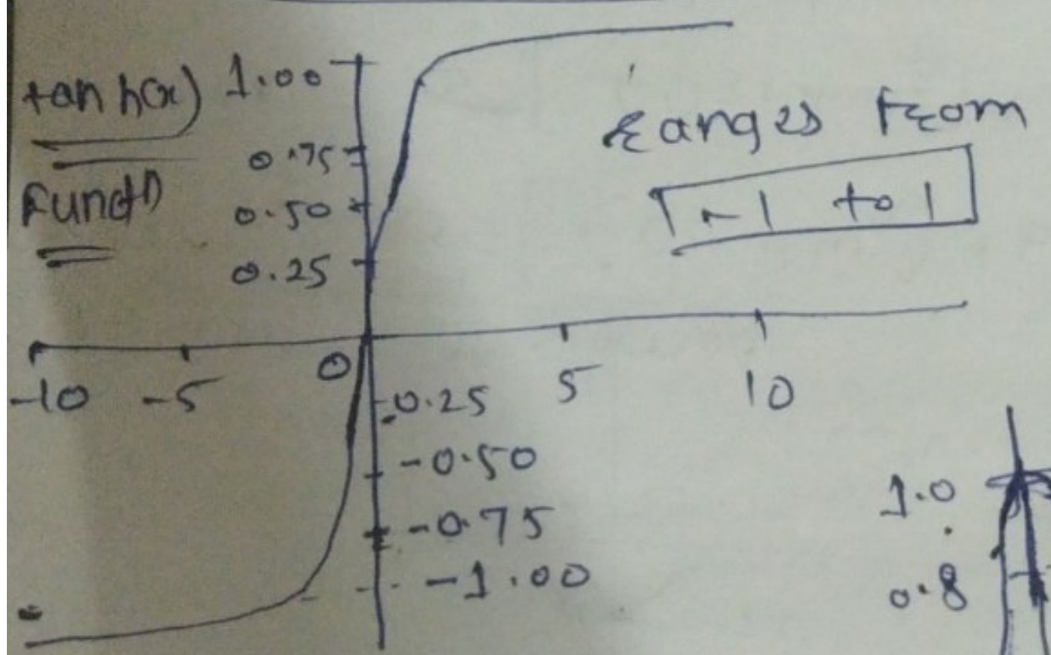
1. It is prone to gradient vanishing
2. function output is not zero centered.
3. power operation (e^{-x}) relatively Time consuming.

2 Tanh function (Activation function)

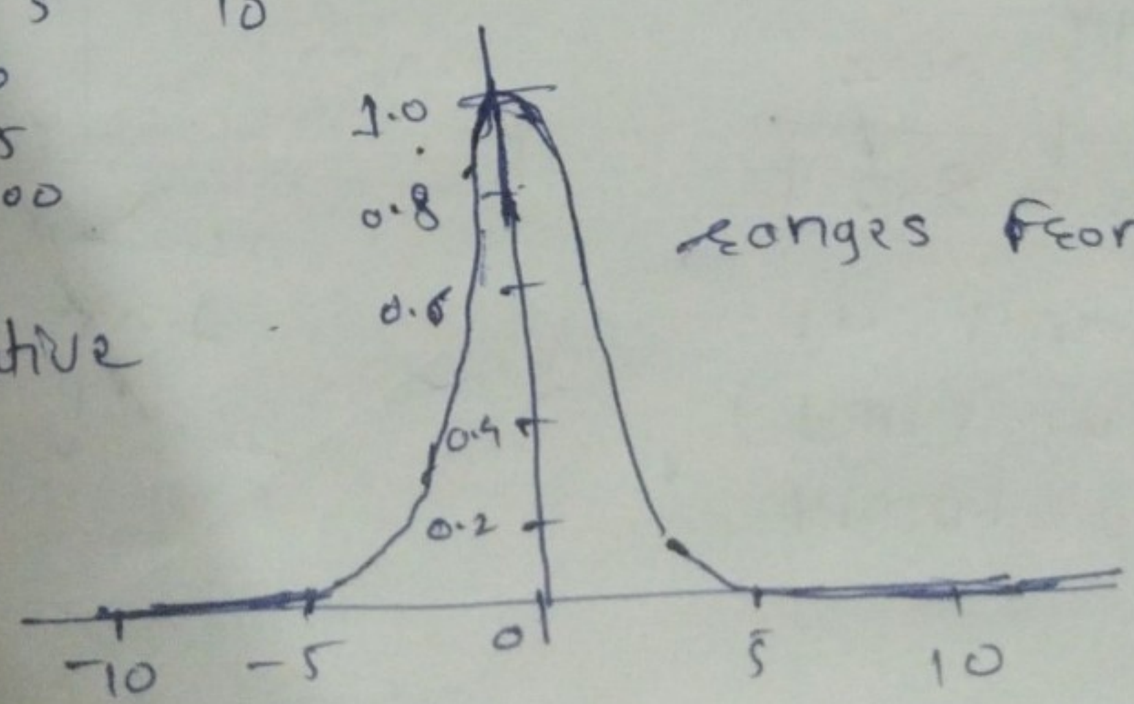
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- It is also called as a hyperbolic tangent function
- It alway's ranges betwe $\boxed{-1 \text{ to } 1}$ in tanh fn.

& In case of derivative it ranges from $\boxed{0 \text{ to } 1}$



* tanh(x) derivative

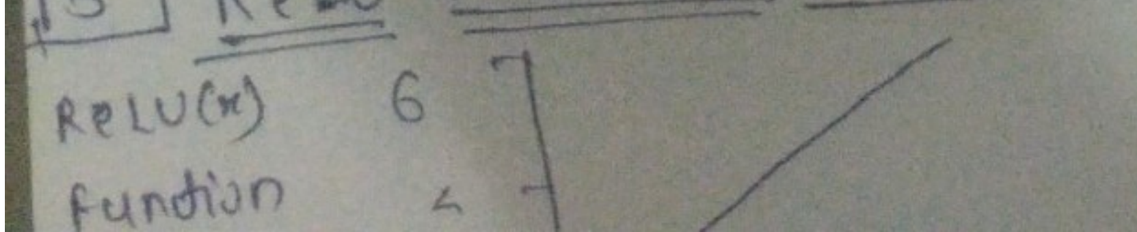


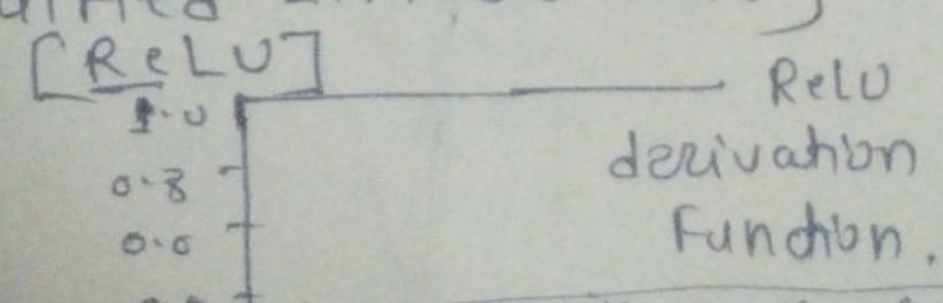
ranges from $\boxed{0 \text{ to } 1}$

* If we go on and constructing deep neural N/W at one point of time there may be a chance of vanishing gradient problem occur
* Input is larger with output is small & gradient is also smooth.

* Tanh(x) function is a zero centric, which is better than sigmoid.
* In binary classification problem we basically use Tanh(x) function for hidden layer & sigmoid funct^n for o/p layer.

3 ReLU Activation function [Rectified Linear unit]





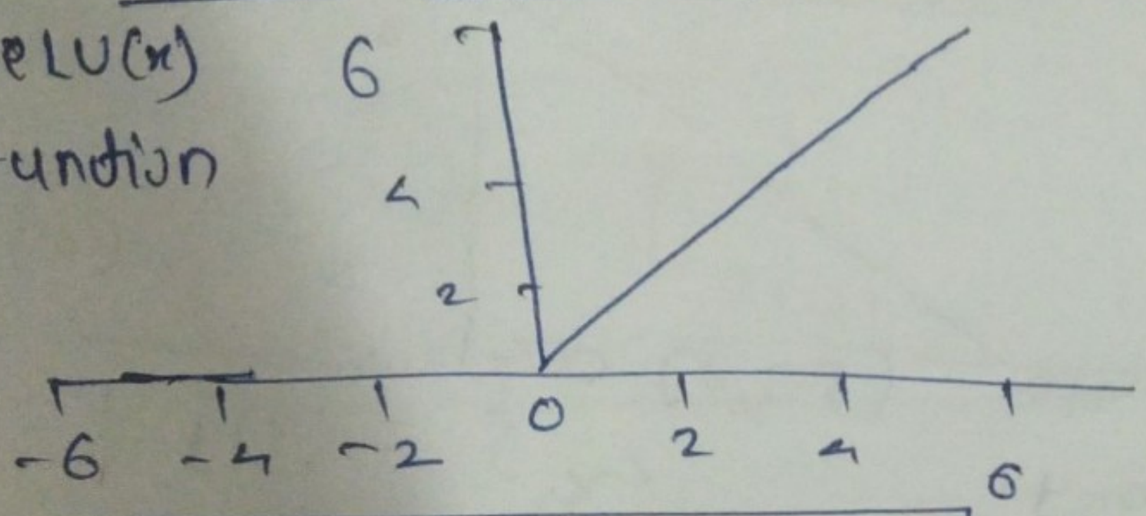
ReLU derivation function.

input is larger with output is small & gradient is also smooth.

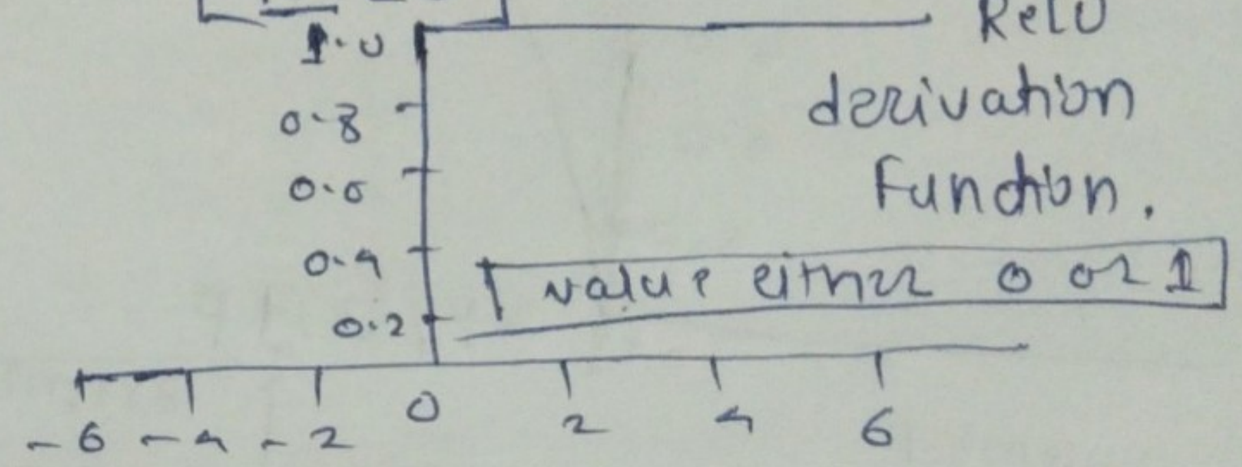
* Tanh(x) function is a zero centric, which is better than sigmoid.
* In binary classification problem we basically use Tanh(x) function for hidden layer & sigmoid funcn for o/p layer.

3 ReLU Activation function [Rectified Linear unit]

ReLU(x) function



[ReLU]



ReLU derivation function.

value either 0 or 1

$\boxed{ReLU = Max(0, x)}$

zero

↳ if x value is -ve then it will be 0.

↳ if x value is +ve then it will be that provided +ve value.

e.g :- if $x = 1 \rightarrow$ we will get 1

↳ so value is either 0 or 1.

↳ In derivative, if $\boxed{x = 0}$, then RLU back propogation neuron will dead. (bcz in derivative value either 0/1 or 2)

↳ then $\boxed{\frac{\partial L}{\partial w_{old}} = 0}$ then $\boxed{w_{new} \approx w_{old}}$ This is disadvantage of ReLU.

Derivative chain rule

↳ It is obviously ~~greater~~ better than sigmoid

↳ It is solving vanishing gradient problem (if +ve input)

↳ It is not zero centric (one more disadvantages)

Advantages :-

1. When input is positive, there is not gradient saturation prob. Calculation speed is faster. ReLU function has only a linear relationship.

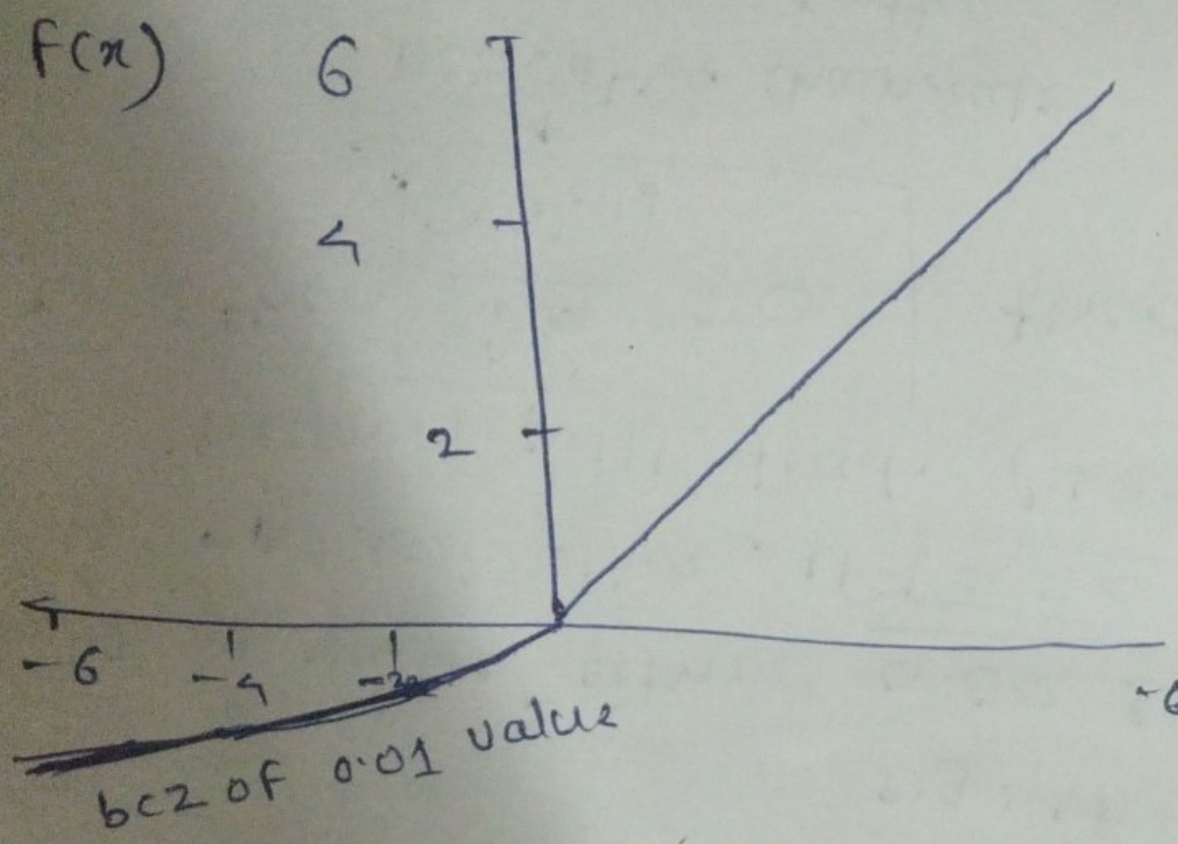
2. Disadvantages :-

1. when -ve input ReLU completely inactive. (2) ReLU O/P is either 0/1 (3) Not 0-centric.

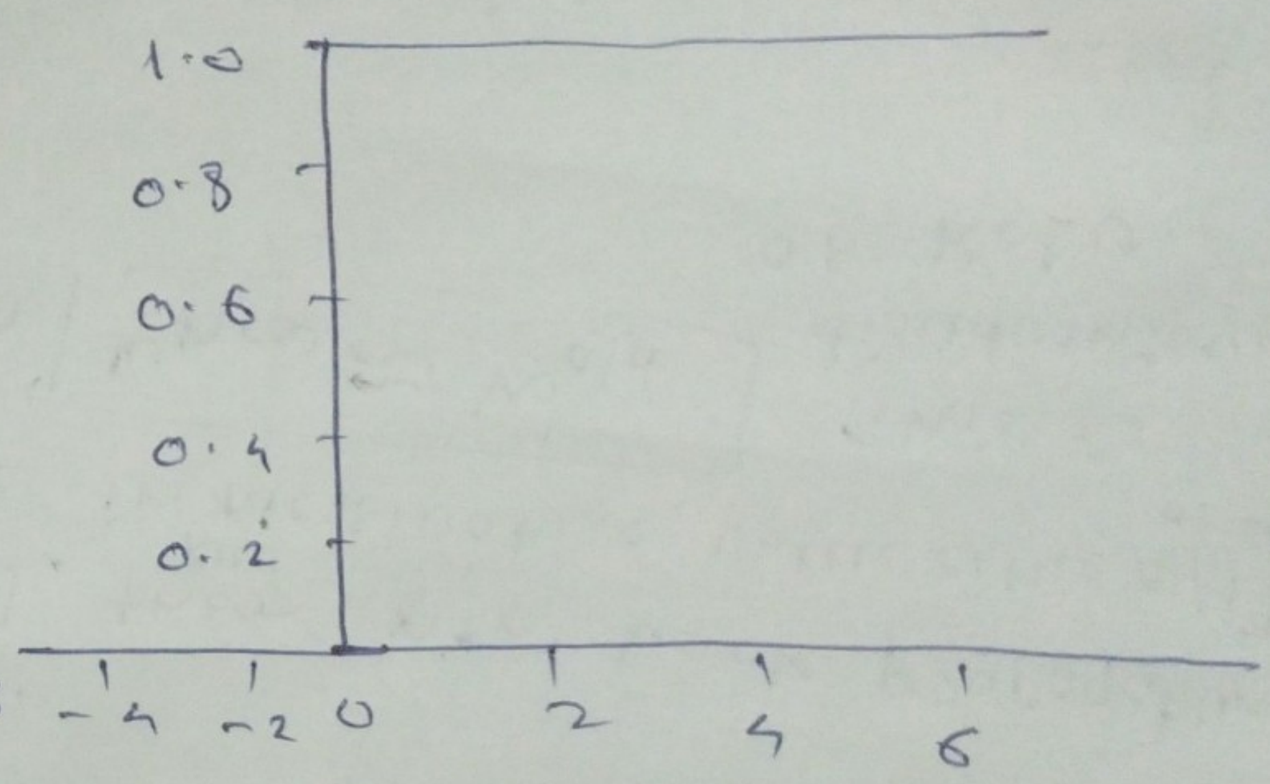
4 Leaky ReLU function

↳ In order to solve, Dead neuron problem in ReLU we will have Leaky ReLU function. In this instead of 0 they have used $\boxed{0.01}$

$$\boxed{f(x) = \max(0.01x, x)}$$



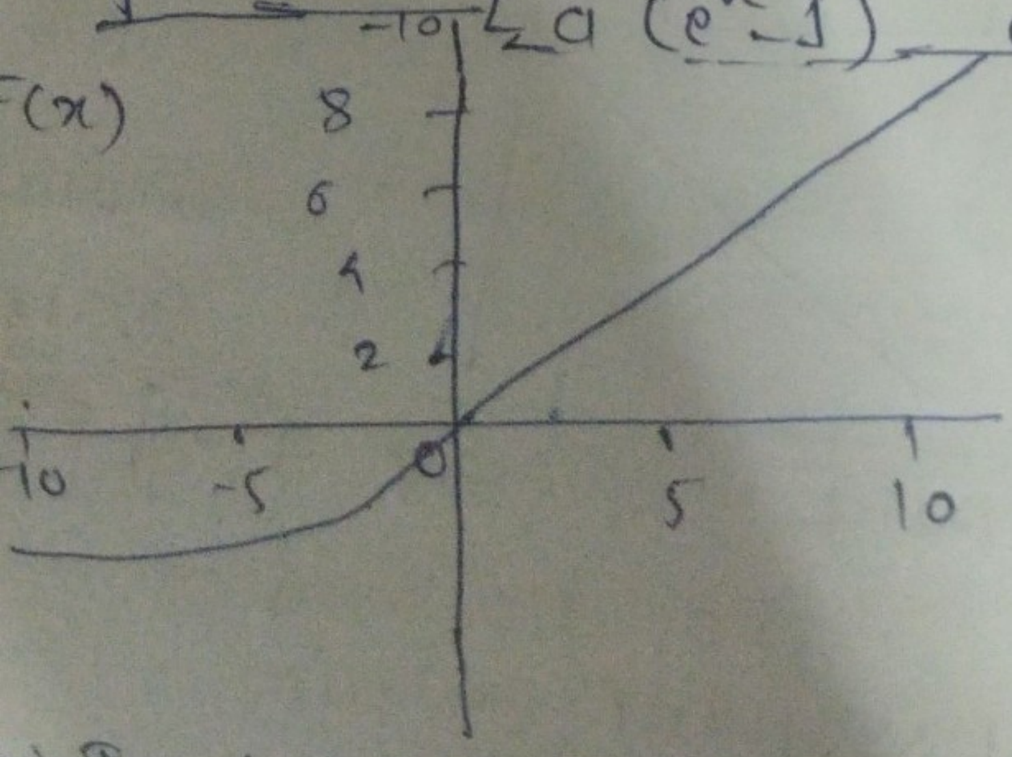
bcz of 0.01 value



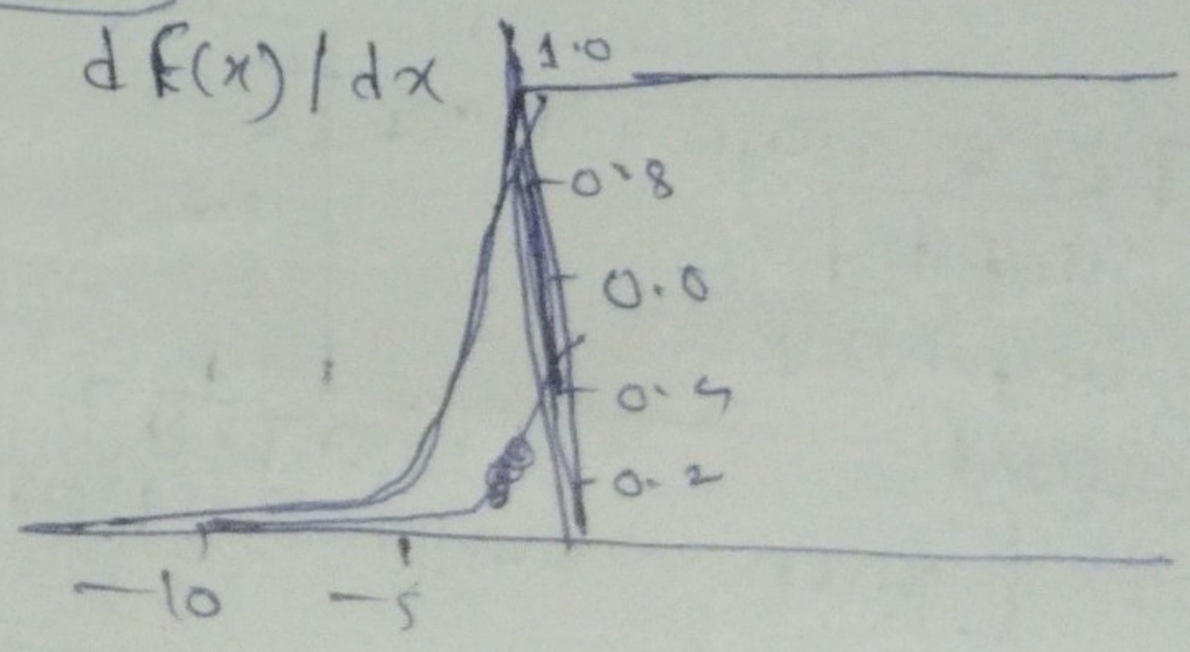
so Leaky ReLU solve dead neuron problem

5 ELU [Exponential Linear units]

$$\boxed{f(x) = \begin{cases} x, & \text{if } x > 0 \\ a(e^x - 1) & \text{otherwise} \end{cases}}$$



derivative

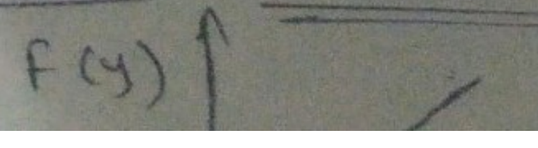


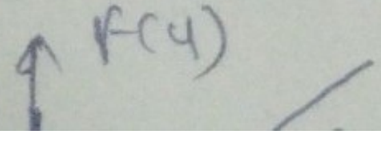
→ In this no dead ReLU Or dead Nuron issue.

↳ o/p is close to 0, zero centric.

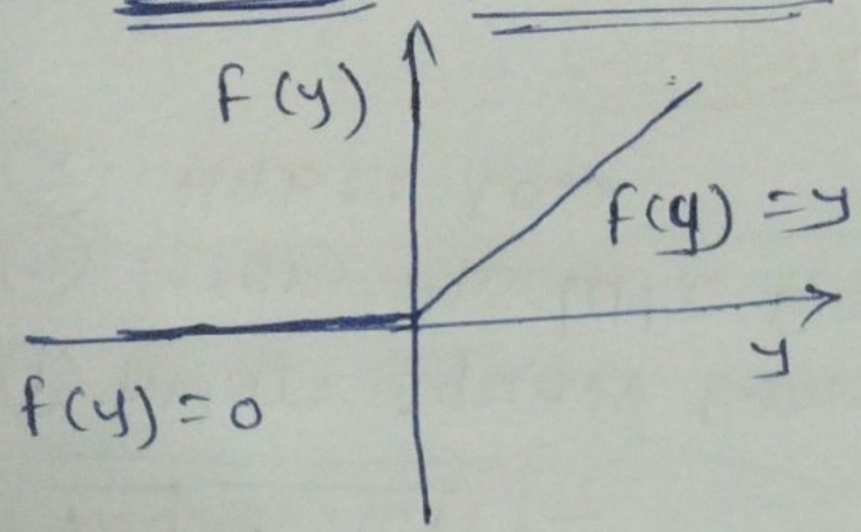
expensive as it uses exponential function

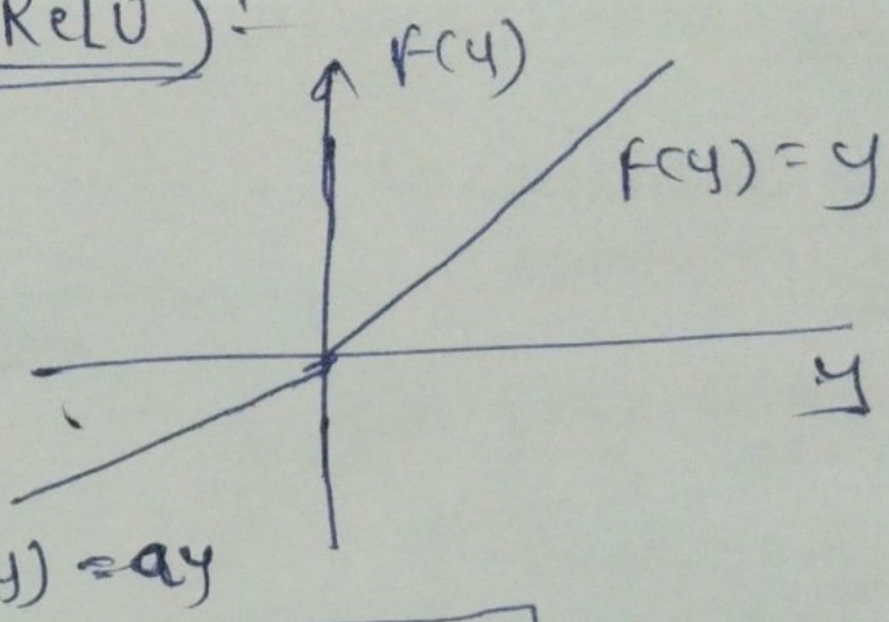
6 PReLU (Parametric ReLU)





6 PReLU (Parametric ReLU):

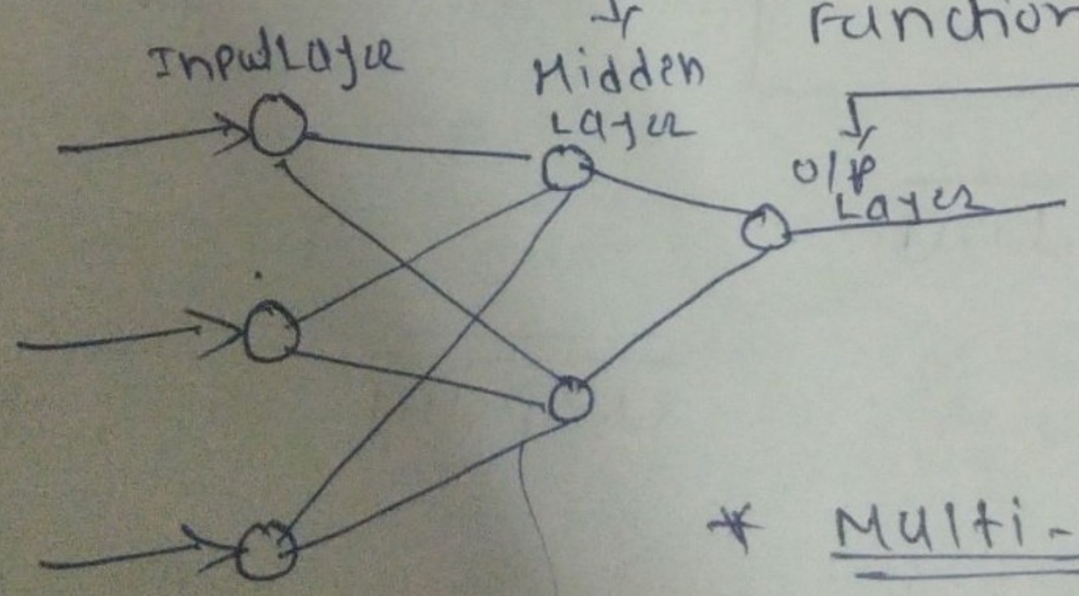




$$\boxed{f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i & \text{if } y_i \leq 0 \end{cases}}$$

1. if $a = 0$ function f becomes ReLU
2. if $a > 0$ function f becomes Leaky ReLU
3. If ~~a<0~~ a_i is a Learnable parameter it becomes PReLU so this learnable parameter keep on changing always.

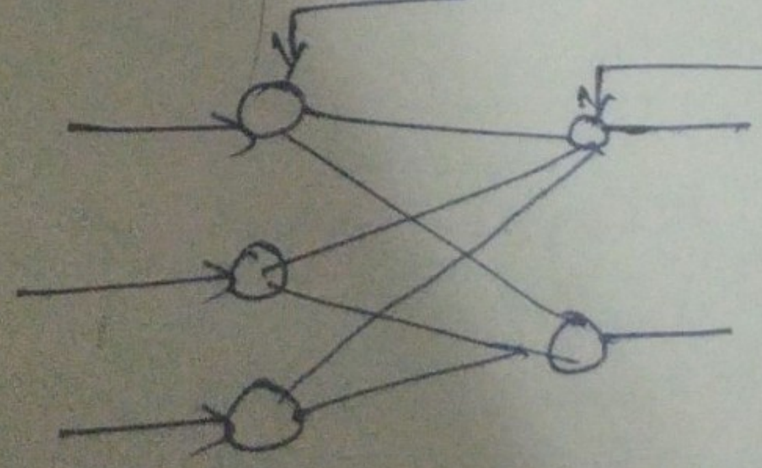
* Technique to which Activation function when it is used



→ use ReLU Activation function in Hidden Layer } if using ReLU conversion are not upto mark then use PReLU or ELU.

→ whenever Binary classificatn problem then use sigmoid function in O/P Layer.

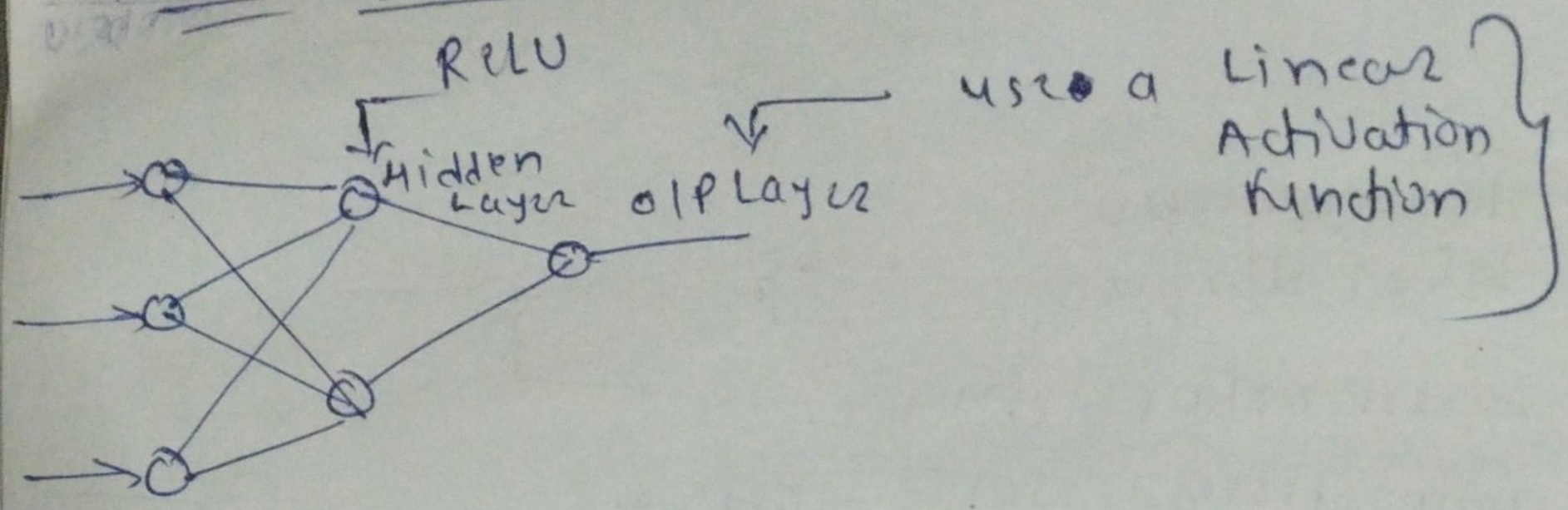
* Multi-class classification function



In hidden layer use a ReLU activatn fn

In O/P Layer use a softmax activation function.

* In Regression



* Loss function

Deep learning

- Regression
- Classification

Dataset:

Exp	Degree	Salary
10	Ph.d	$, 10,000
—	—	—
—	—	—

Dataset:-

Playing Hours	study Hours	Pass/Fail
10	2	Fail
4	3	Fail
5	5	Maybe
2	7	Pass

So this is a Multiclass classification problem

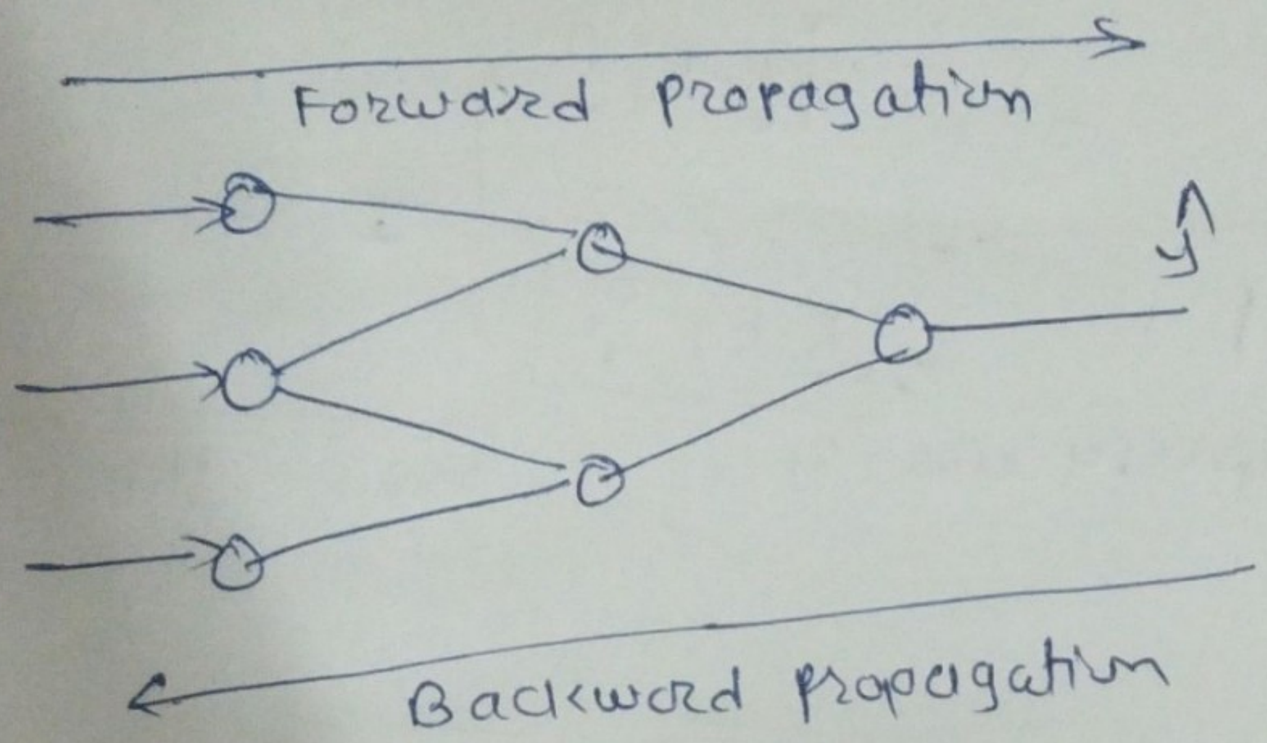
⇒ so with respect to Regression & classification problem we will have different loss-functions.

* Regression :-

1. Mean square Error
2. Mean Absolute Error
3. Huber Loss

↳ In Regression Letus understand **Loss function** & **cost function**.

↳ consider one **Dataset** which contain **100 records**



↳ **Loss function** =

$$\boxed{\text{Loss} = \frac{1}{m}(y_i - \hat{y}_i)^2}$$

if we pass single record from Dataset to neural network (out of 100 record) one by one sequentially.

↳ If we pass, **10-record** at a time to a neural N/w or **batch of record**,

↳ Then in this case formula will change & we treat Loss function as cost function.

$$\boxed{\text{cost function} = \frac{1}{m}\sum_{i=1}^{n}(y-\hat{y})^2}$$

$$MSE_{test} = \frac{1}{m}\sum_i (\hat{y}^{(test)} - y^{(test)})^2$$

1 **Mean Squared Error [MSE]** :-

Loss Function (single record)

$$\boxed{\text{Loss function} = \frac{1}{m}(y-\hat{y})^2}$$

⇓ This eqn is a quadratic eqn

cost function [Batch of Record]

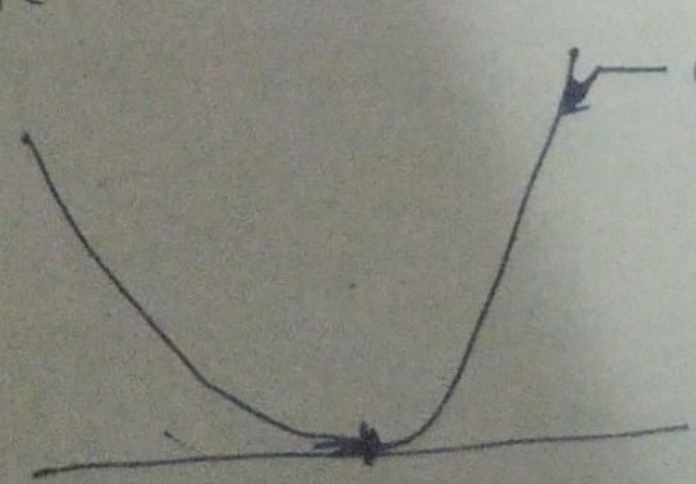
$$\boxed{\text{cost function} = \frac{1}{m}\sum_{i=1}^{n}(y-\hat{y})^2}$$

example of Quadratic equation is a $(a-b)^2 = a^2 - 2ab + b^2$

⇓

$$\boxed{ax^2 + bx + c}$$

This is also.



* **Advantages of Quadratic eqn**

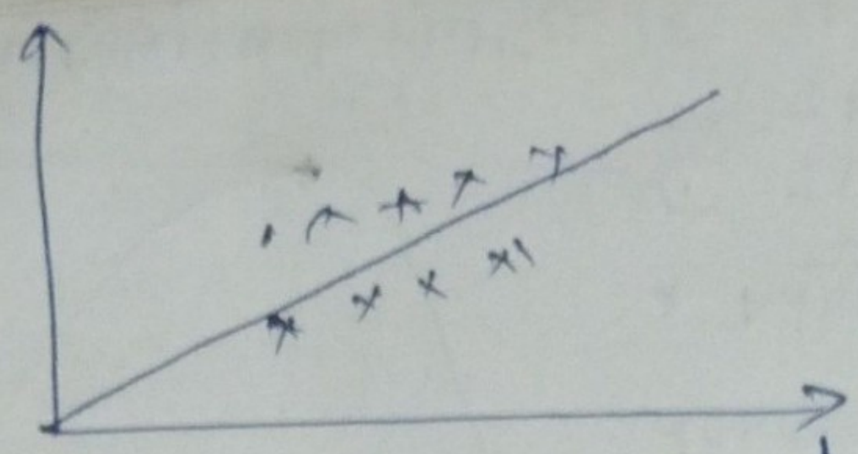
① It can be differtiable

② It has only 1-local or Global Minima

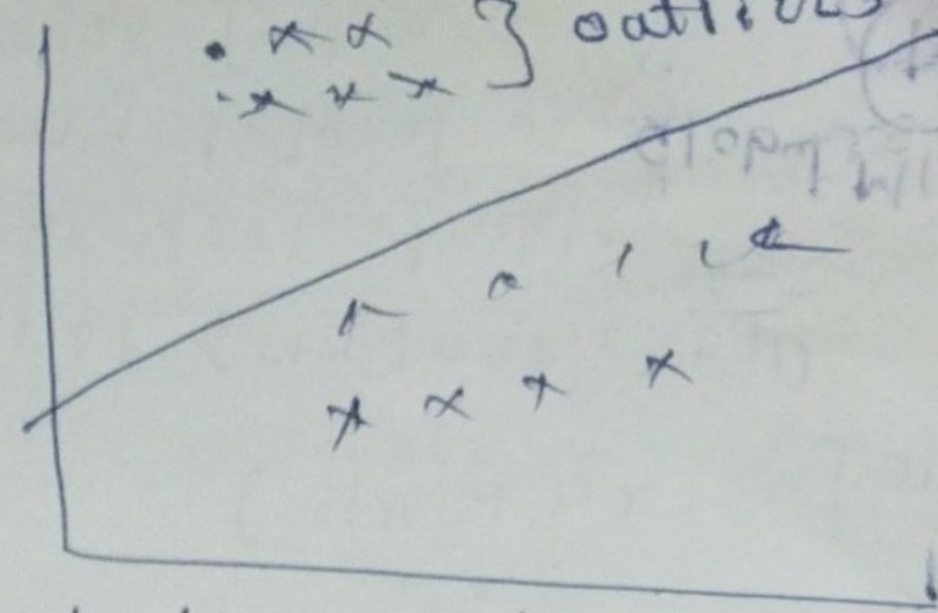
③ It converges faster

Disadvantage

① It is not Robust to outlier



1. Quadratic equation without outliers



2. Quadratic equation with outliers. There is big change in line → line is (shifted) This is bcz of square of function (Cost Function)

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

This is called Penalizing errors.

↳ MSE is not robust to outliers.

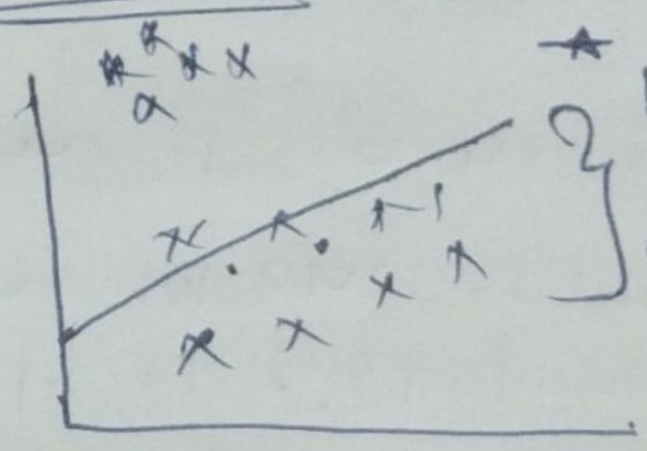
2 Mean Absolute Error: [MAE]

$$\boxed{\text{Loss function} = \frac{1}{n}(y-\hat{y})} \qquad \boxed{\text{cost funct}^n = \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})}$$

without square — without square.

As it is without square | Not penalizing errors

↳ so Mean Absolute Error is Robust to outliers.



3 Huber Loss: It is combination of MSE & MAE

$$\text{Loss} = \begin{cases} \boxed{\frac{1}{2}(y-\hat{y})^2} \ldots \text{if } |y-\hat{y}| < \delta \rightarrow \boxed{\text{without outlier}} \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 \quad \text{otherwise } \boxed{\text{(with outlier)}} \end{cases}$$

← This delta represent some hyperparam

B Classification Loss function

1. Cross Entropy –
   - → Binary cross Entropy → Binary classification
   - → categorical cross Entropy → Multi-class classification

(i) Binary cross Entropy (Binary classification)

$$\boxed{Loss = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})}$$

⇒ Logistic Regression using Loss called as Log Loss.

↑ In above formula

↳ if we put $y=0$ & $y=1$, then we will get following formula:-

$$\boxed{Loss = \begin{cases} -\log(1-\hat{y}) & \dots \text{ if } y=0 \\ -\log(\hat{y}) & \dots \text{ if } y=1 \end{cases}}$$

for Binary classification 0 or 1

* How do we calculate $\hat{y}$, To calculate $\hat{y}$ in last output layer we use sigmoid activation function.

$$\boxed{\hat{y} = \frac{1}{1+e^{-x}}}$$

(ii) categorical cross Entropy (Multi class classification)

↳ Let's say we have given one dataset.

f_1	f_2	f_3	o/p
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

⇒ Now we will convert this table into one hot encoded format.

Good	Bad	Neutral	
$1_{(y_{i1})}$	$0_{y_{i2}}$	$0_{y_{i3}}$	– Good value
0	1	0	– Bad value
0	0	1	– neutral

i = row, & j = column

$$\boxed{L(x_i, y_i) = -\sum_{j=1}^{c} \boxed{y_{ij}} * \log(\hat{y}_i)}$$

$y_i = [y_{i1}, y_{i2}, y_{i3} \dots y_{ic}]$

↳ It mean No. of categories

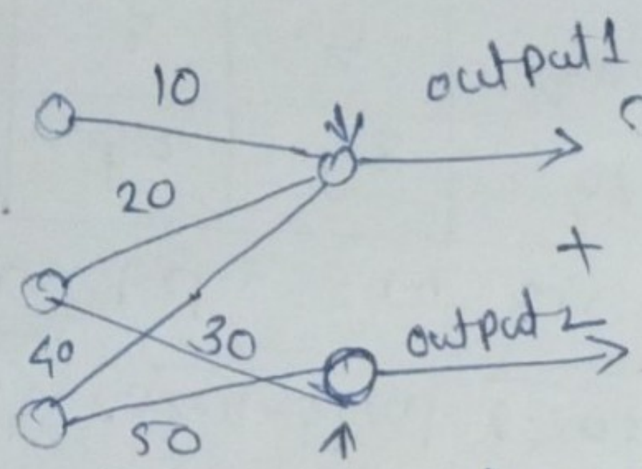
$$\boxed{y_{ij} = \begin{cases} 1 & \dots \text{ if element is in class} \\ 0 & \dots \text{ otherwise} \end{cases}}$$

eg ↳ (For first row → 1 Good value & other values are 0)

↳ Now we calculate $\hat{y}_{ij}$ = It is a softmax Activation function it is applied in the O/P Layer of Multiclass classification.

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

⇒ softmax activation function.



2-output Function In output layer

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

⇒ This i will be getting in the form of probability like 0.4, 0.5, 0.6

→ Both probability summation will be 1 & whichever will be higher probability that class output will be displayed.

★★

	Hidden layer	output Layer	
1.	ReLU ,	Softmax	⇒ Multiclass
2.	ReLU ,	sigmoid	⇒ Binary

categorial cross Entrophy loss function

→ Binary cross Entropy

$\boxed{\text{Linear Regression}}$

ReLU → Linear Activation → MSE | MAE | Huber loss function

★ Day.4:- optimizers → 80

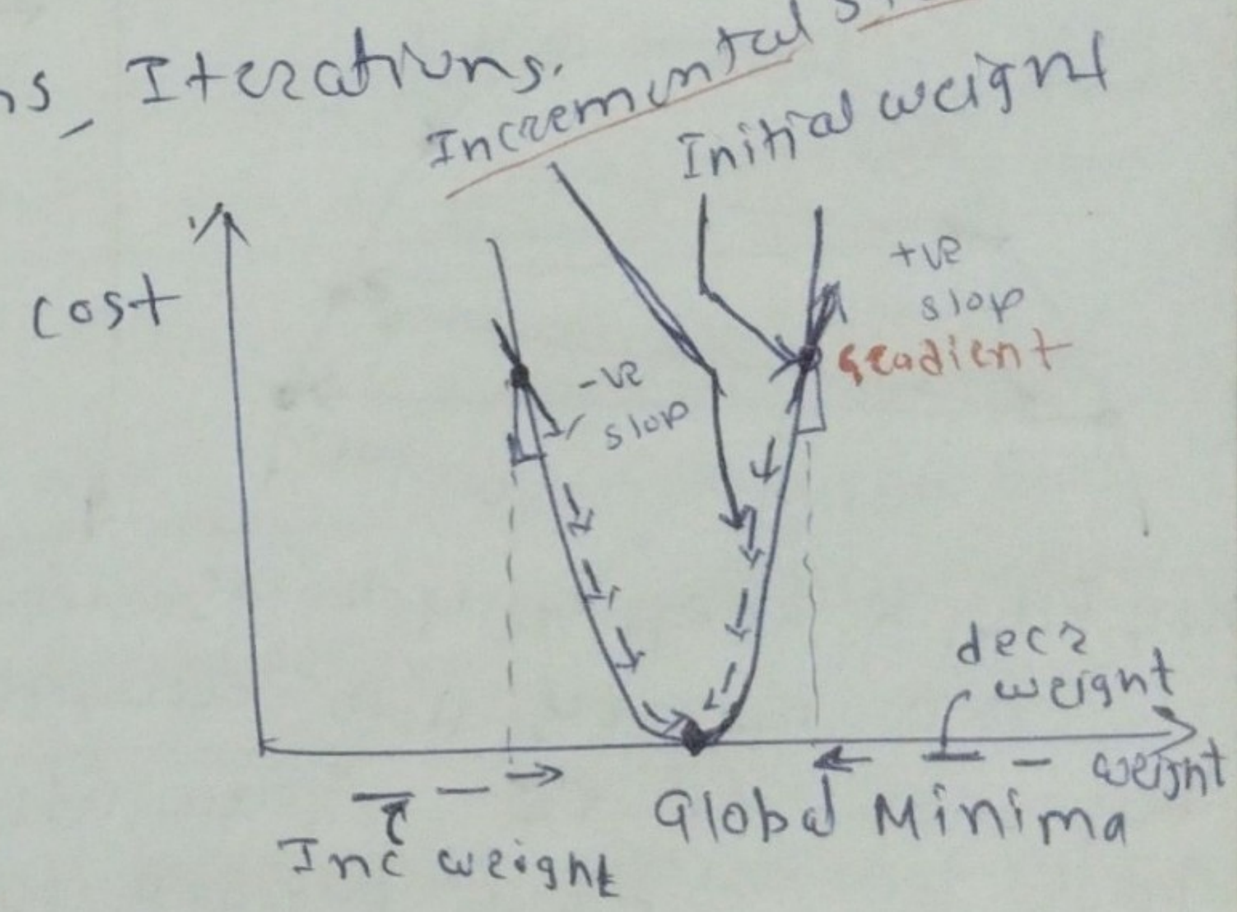
* Day.4:- optimizers → 80

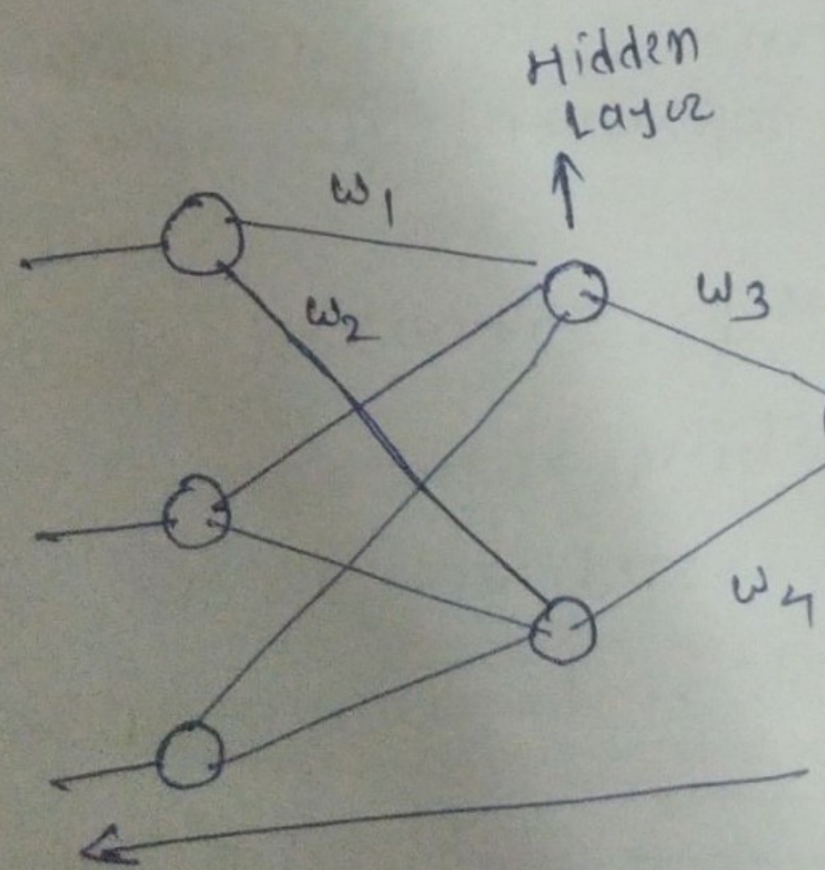
i) gradient Descent
ii) SGD [stochastic gradient Descent]
iii) Mini Batch [SGD]
iv) SGD with momentum
v) Adagrad
vi) RMSPROP
vii) Adam optimizer
viii) what is Batch, Epochs, Iterations.

1 Gradient Descent

↳ weigh updation formula

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





This is nothing but MSE
Mean square error

$$\boxed{\text{cost fn} \quad \text{Loss} = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2}$$

→ This is a Loss function & our aim is to minimize loss function with help of optimizers

* In Back propagation weight's are updated based on cost function

* Let's understand Epoch :-

Epoch → 1 Forward propagation
← 1 Backward — " — } → we Treat this as 1 - Epoch.

if 1000000 → 1 Million record's in my dataset
↳ In forward propagation we will do millions different calculation, input's weights.

& at a Batch we will take million data point & we are doing forward propogation, & we determine $\hat{y}$.

↳ and in Backward propagation, weight updation is getting updated based on loss or cost function.

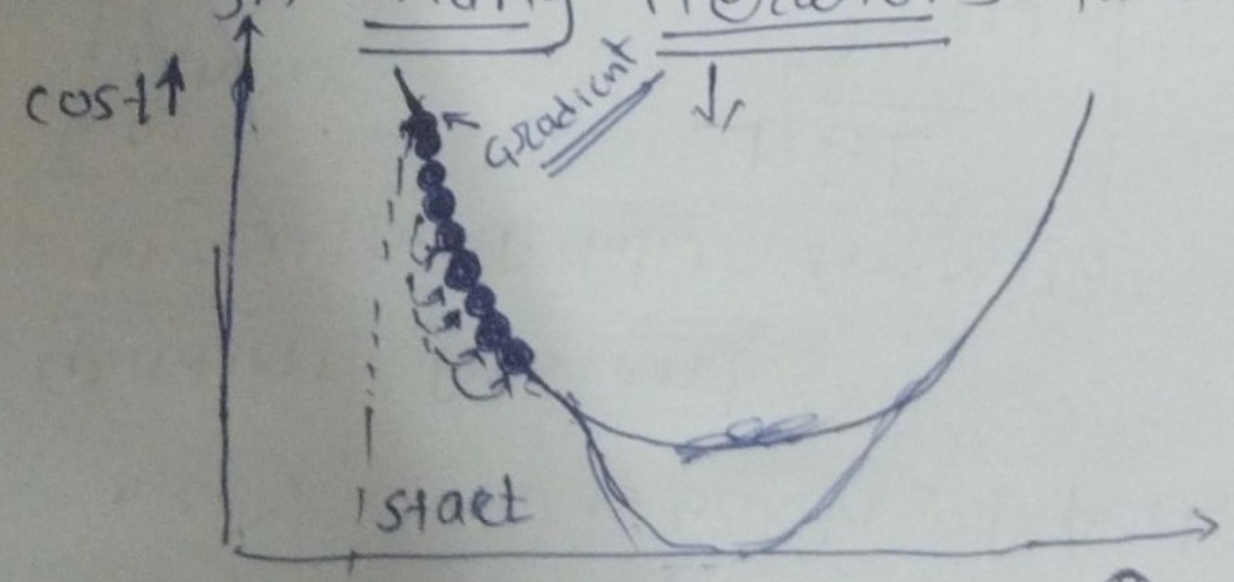
↳ Epoch $\xrightarrow[\longleftarrow]{1000\,000}$ } 1 Epoch (Batch of 1000,000) records / data points

* Disadvantages: of Gradient Descent

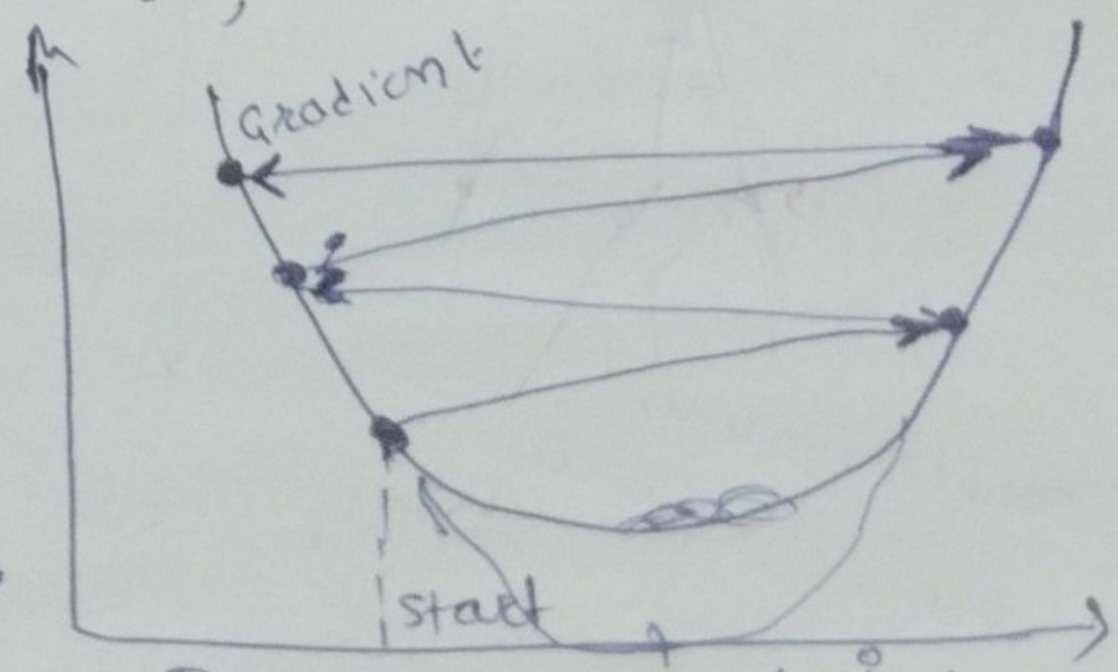
↳ If we pass millions of records definetly we will require huge RAM to load that records so

① Resource Extensive { require huge RAM SIZE for every Epoch }

* Important parameter in Gradient descent is a size of steps, determined by learing rate Hyperparameter. If the learning rate is too small, then algorithm will have to go through many iterations to converge, which take a long time



* The learning rate is small



* The Learning rate is too large

* on the other Hand, if the learning rate is too large you might jump across the valley and end up on the other side,

- This might make algorithm divarge, with large values, falling to find good solution.

* Types of gradient Descent

① Batch gradient ② stochastic ③ mini Batch

* Bias & variance:

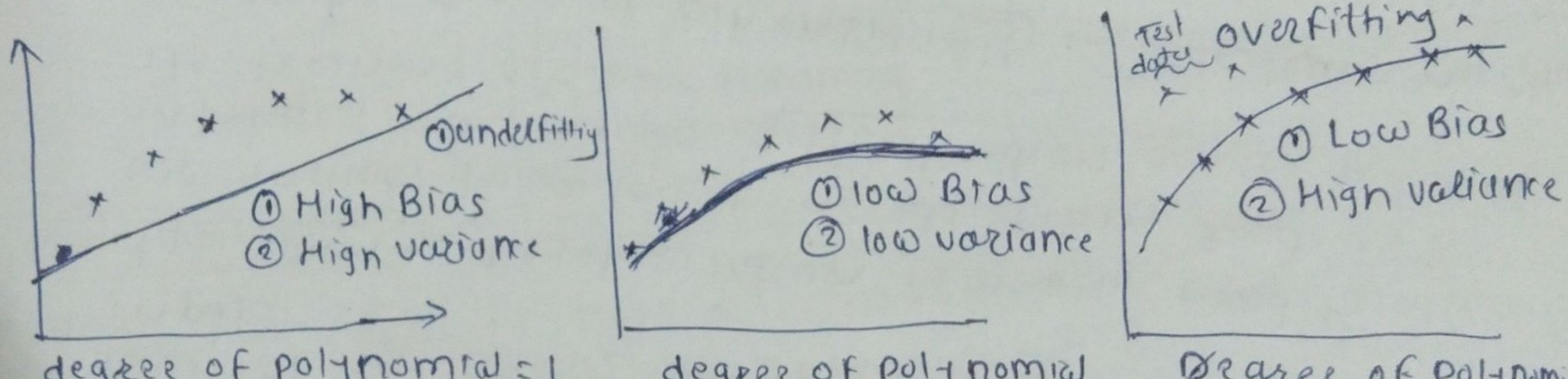
① underfitting: when model is not able

① **underfitting**:- underfitting occurs when model is not able to obtain ~~too~~ sufficiently low error value on the training data set.

↳ whatever data set train my model, for that error is quit high.

② **overfitting**:- It occurs when the gap between the training error & test error is too large.

↳ over a training dataset accuracy of model is very high but over test data set accuracy is very low.



degree of polynomial = 1 | degree of polynomial = 2 | Degree of polynomial = 4

* **Bias**:- It means error of a training data.
* **variance**:- It means errors of a test data.

$$\boxed{\text{bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta}$$

where E - indicate estimate fund
m - indicate Model

θ - ~~True~~ underlying true value.

$\hat{\theta}_m$ - Estimated value of a Model

↳ An estimator $\hat{\theta}_m$ is said to be unbiased, if $\boxed{\text{bias}(\hat{\theta}_m) = 0}$

which implies that $\boxed{E(\hat{\theta}_m) = \theta}$.

↳ **Model 1**:-
Training Error = 1%
Test Error = 20%
⇓ (low Bias, high variance)
so This is a overfiting prob.

Model 2:
Training Error = 25%
Test Error = 25%
⇓ High Bia & High variance
so This is a underfitting

Model 3:
Training Error = 10%
Test Error = 10%
⇓ low Bias
low variance
Most generalized Model.

↳ consider set of samples $\{x^1 \dots x^m\}$ distributed according to Bernoulli's distribution with mean θ:

$$\boxed{P(x^{(i)}; \theta) = \theta^{x^{(i)}} (1-\theta)^{(1-x^{(i)})}}$$

where $\boxed{\hat{\theta}_m = \frac{1}{m} \sum_{i=1} x^{(i)}}$

* **Generalization Error**:- our Model must perform well on new, previously unseen inputs - not just on those on which model is trained. The ability of Model to perform well on previously unobserved inputs is called Generalization. [Test error]

↳ The Generalization error, is defined as expected value of the error on new or previously unobserved inputs.

* **Regularization**:- Regularization is a any modification we make to a learning algorithm that is intended to reduce generalization error, but not its training error.

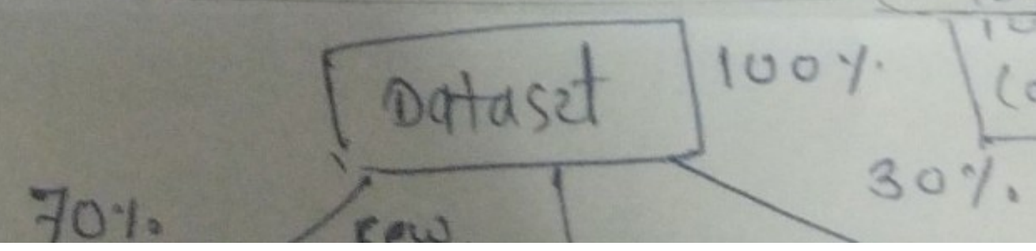
* ~~Hyperparameter~~. **Curse of Dimensionality**:- Mlc learning Algo becomes exceeding difficult when the number of dimensions in the data is high. This phenomenon is know as the curse of dimensionality. statistical challenges arrises ~~when~~ because number of possible configurations of x is much larger than number of training examples. To understand this, let us consider that input space is organized into a grid.

↳ we can describe low-dimension space with a small number of grid cell.

↳ when generalizing to a new data point, (is nothing but) training is checked by a example that lie in the same cell as a new input

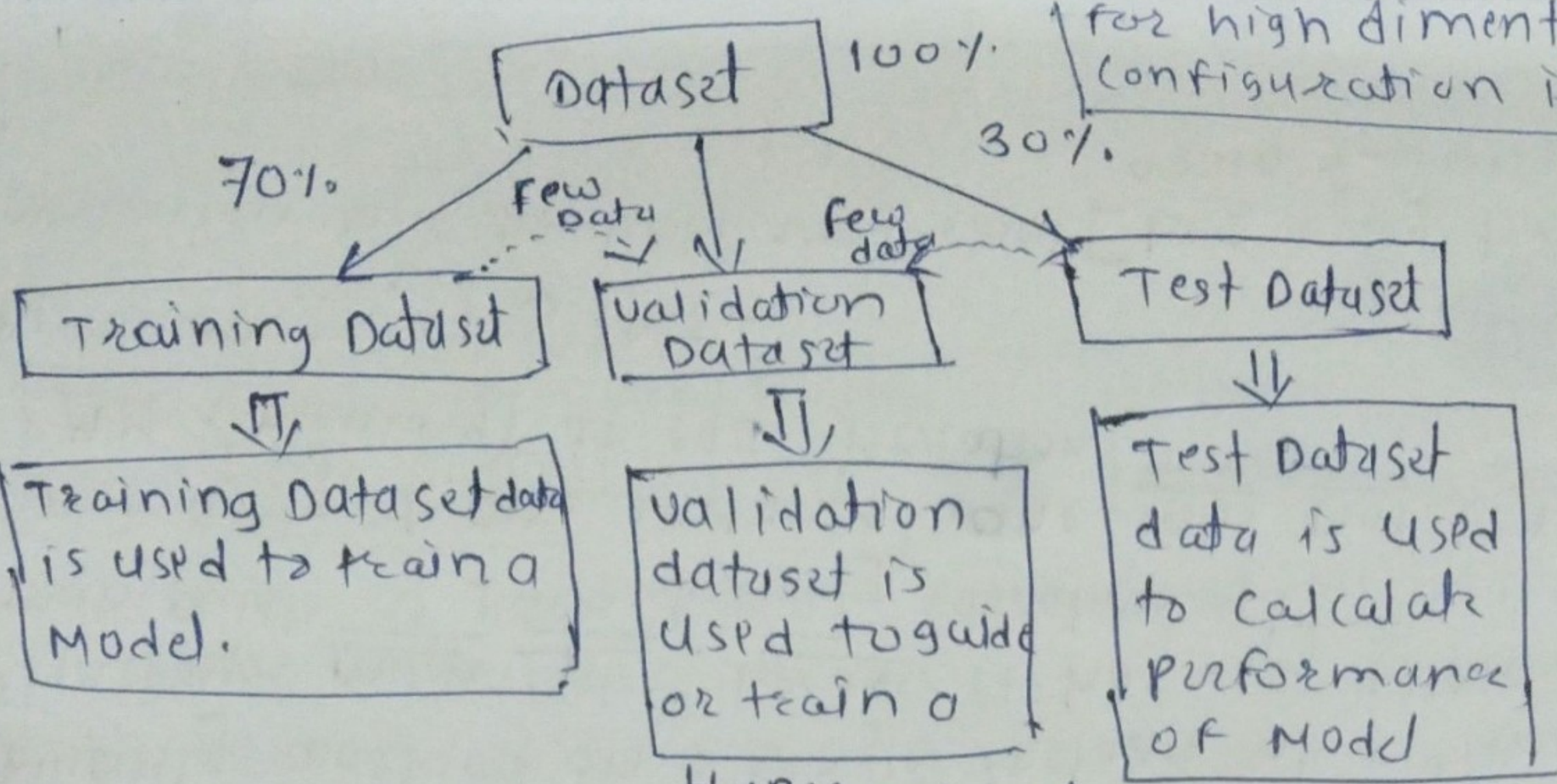
Probability density at some point x = $\frac{\text{No. of Training example in same}}{\text{Total no. of training Example}}$

configuration is also huge.



Example

For high dimention, no. of configuration is also huge.



Hyperparameter are the parameter used to control algorithm/Model behaviour.

* **cross validation** :-

* Now, 70% data is passed into Training data set & 30% data is passed into Test dataset then from where data comes in validation dataset.
* So, Data comes in validation dataset with the help of term cross validation.
* So, using cross validation some data of Training & Test data are added into validation set.
* These procedure are based on idea of repeating the training & testing compatation on different randomly chosen subsets' or splits of the original data set.
* The Most common of these is these is the K-fold cross validation, in which partition of dataset is formed by splitting it into K-nonoverlapping subsets.
* Test error may be estimated by ~~(splitting it into K)~~ by taking average test error.

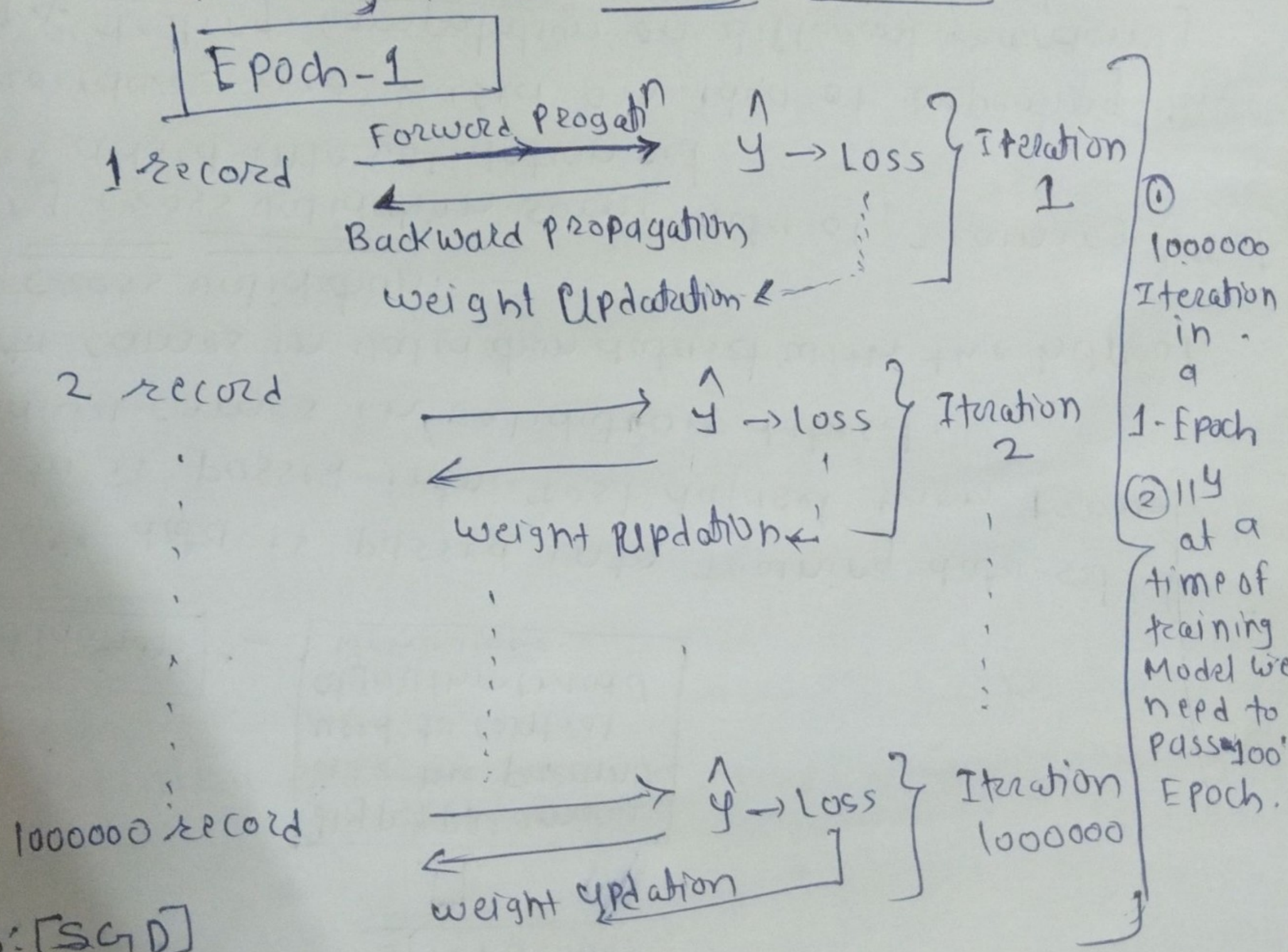
On i^{th} trial, i^{th} subset of data is used, as test set & rest data as training set

* Continue from page (21): stochastic gradient descent is a valiant of gradient descent algo is used for optimizing ML model.

[2] SGD (Stochastic Gradient Descent)

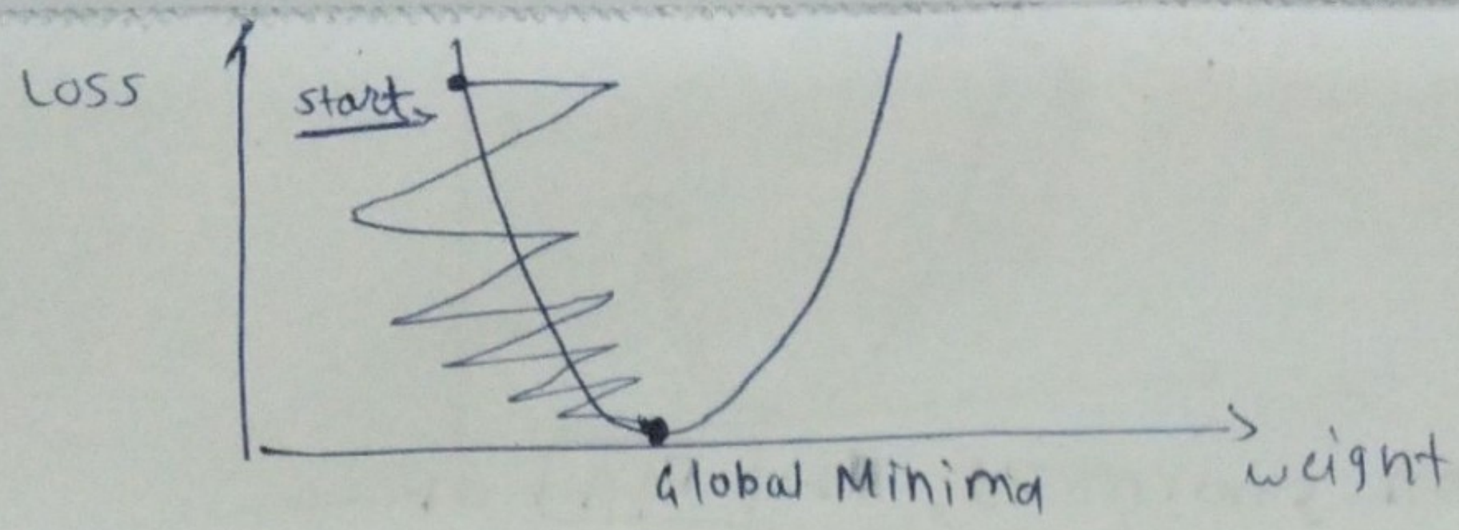
* Let's say we have Millions of records, in Dataset.
* Every Epoch is combination forward & backward propagation & then we calculate $\hat{y}$ → from that we calculate Loss.

→ with respect to loss, we will do weight updatation.



* Advantages: [SGD]
* Obviously, working on a single instance at a time makes algorithm much faster, because it has very little data to manipulate at each & every iteration.
* since only 1 one instance - one may not require that much RAM (memory) at each iteration.

* Disadvantage of SGM:-
↳ conversion will be very-very slow [bcz every time 1-record & again $\hat{y}$ → weight updation & all]



2

3 **Mini-Batch SGD** :- Let's say, 1000000 Million record, now instead of sending 1-Million record, we will setup a Batch size.

* Let's say My Batch size = 1000 records.
* It means that inside a every epoch, we will pass 1000 record.

Epoch-1

1000 records → $\hat{y}$ → Loss } Iteration 1
← weight updatedn

* Then How many itration i have consider = $\frac{1000000}{1000}$ = 1000 Iteration

1000 records → $\hat{y}$ → Loss } Iteration 2
← weight updation

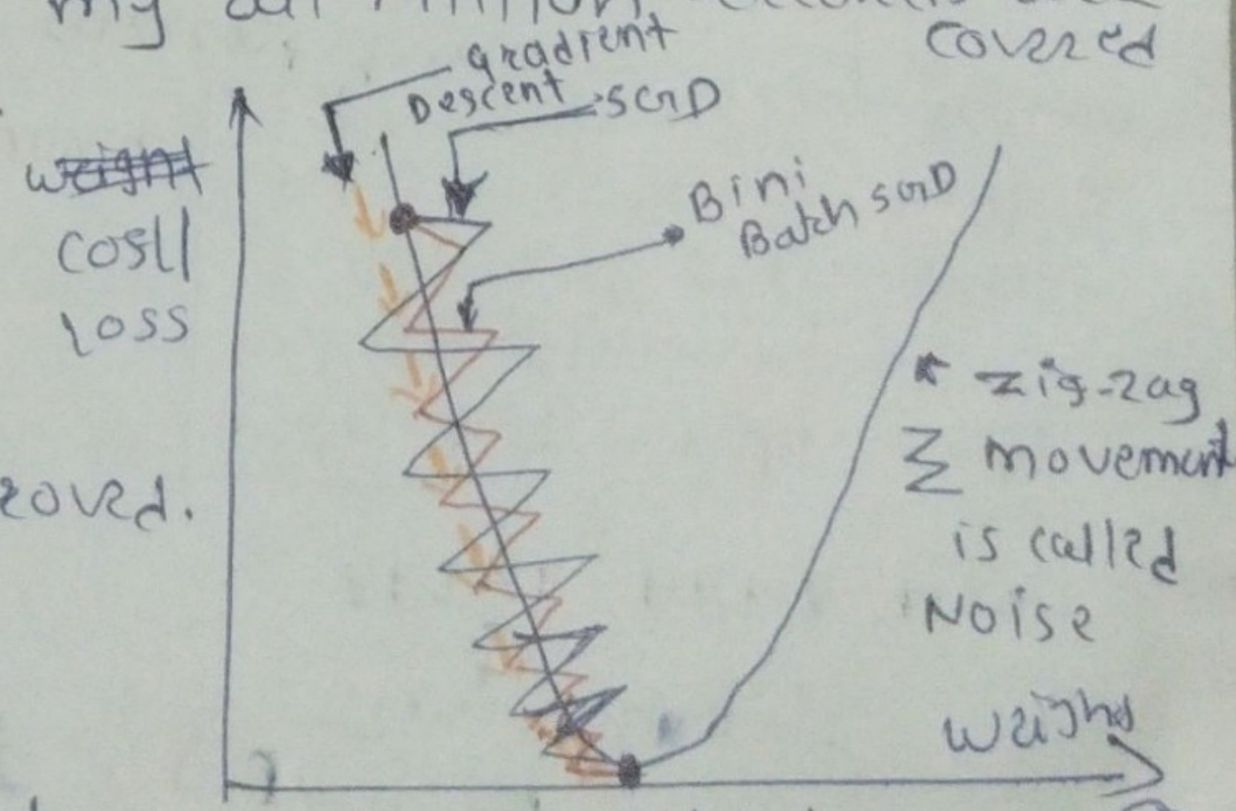
⋮

1000 records → $\hat{y}$ → Loss } Iteration 1000
←

L) Now with respect 1000 iteration my all Million records are covered this looks like better aproch.

* Advantages of Mini-Batch SGD

① Resource Intensive
② conversion will be better.
③ Time complexity will be improved.



— Mini-Batch

* from the diagram we conclude that Mini-Batch SGD has less Noise & SGD has large noise, but with Mini-Batch small noise

④ SGD with momentum :- In this Exponential weighted Avg is used

$$\boxed{w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}} \approx \boxed{\theta = \theta - \eta \nabla_\theta J(\theta)}$$

↓ used in Time series.

↓ ARIMA, ARMA Model

This is weight updation formula

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

This is a Bias updation formula

* In Mini-Batch SGD bit smoothening of curve is got but still some noise is there, that is reduced using momentum

* Let me write weight updation formula in new form.

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

t - current time
$t-1$ - previous time
η - learning rate.

* Exponential weighted Average

t_1	t_2	t_3	t_4	... t_n	→ time
a_1	a_2	a_3	a_4	... a_n	→ value associated

$V_{t_1} \rightarrow a_1$ ... value of time $t_1 = a_1$

$$\boxed{V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2}$$

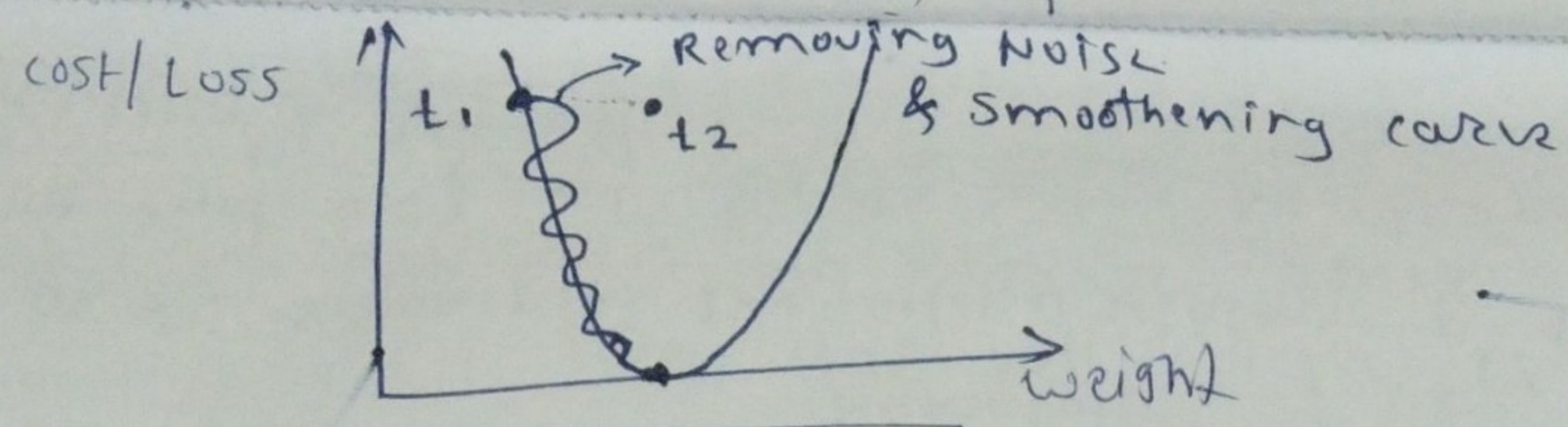
value t_2 depends upon previous time t_1.

β - it is a Hyperparameter
β - it tell us that on which value we should focus more. (previous value)
β - usually range bet^n 0 to 1.

* Lets say $\beta = 0.95$ & put values

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

↳ so it indicate that we should give more importance to $\boxed{V_{t_1}}$ i.e. previous value as 0.95 > 0.05



$$\boxed{V_{t_3} = \beta \times V_{t_2} + (1-\beta)\, a_3}$$

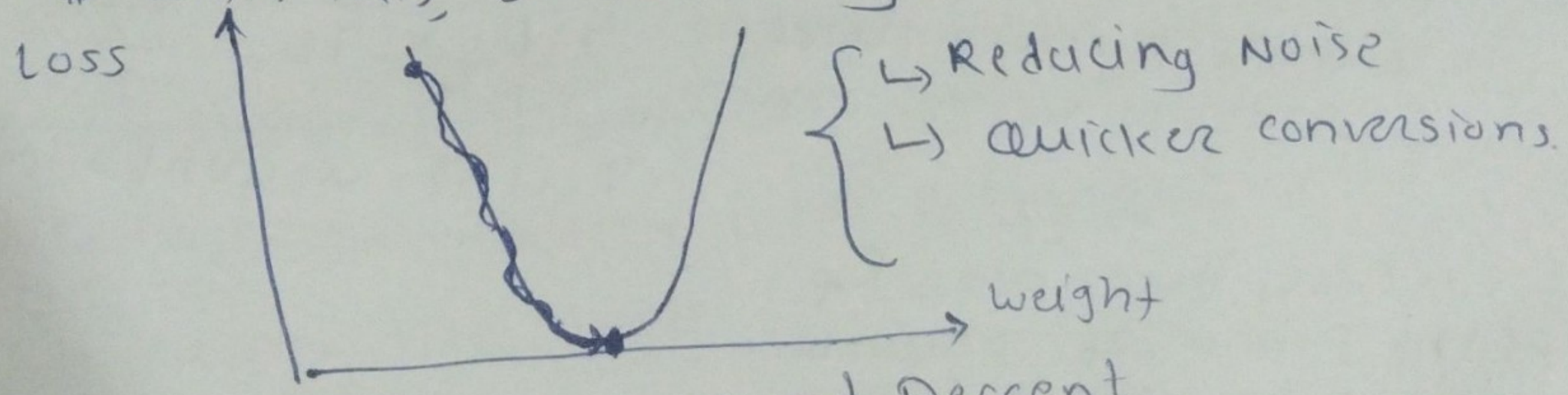
↳ **Exponential weighted Avg**

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

where V_{dw} → value of derivative (weight updatation)
↳ It will be weighted Avg.

$$\boxed{V'_{dw_t} = \beta \times V_{dw_{t-1}} + (1-\beta) * \frac{\partial L}{\partial w_{t-1}}}$$

* with this, Smoothening of curve is done.



5 Adagrad :- Adaptive Gradient Descent

↳ In gradient descent η - indicate learning rate, which has fix rate, or value.

↳ In Adagrad - Adaptive → This term is used, it means that learning rate (η) need to be change or Adaptive.

↳ So weight updatation formula :-

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

Here $\boxed{\eta = \text{Fixed} \Rightarrow \text{Adaptive}}$

* Now In Adaptive gradient descent we replace this η by Etha or Eta η'. so formula will become :-

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

where

Eta

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

alpha, epsilon

t - current timestamp
t-1 - previous timestamp

where epsilon is a small number to avoid divide by zero condition, if α is zero.

so new equation will be:

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

current time, summat^n of all previous weights

where with respect to current time stamp t, summation of all previous ~~time~~ weights & squaring.

* our main objective to decrease η ↓ for this we need to increase α_t.

e.g:

t=1	t=2	t=3	→ Time increasing
η=0.01	η 0.05	η=0.002	→ Decreasing

0.1 - - - - - - - - -

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

→ "if α_t increases then η decreases bcz α_t at denometer & n at Numerator,

So as we reach near the Global Minima η' decreases
So in that way we try to bring Adaptiveness in Learning rate (eg)

5) Adadelta & RMSprop

numerator,

* So as we reach near the Global Minima η' decreases
* So in that way, we try to bring Adaptiveness in Learning rate (29)

5 Adadelta & RMSProp

↳ we also makesure that $\boxed{\alpha_t}$ does not reach a very ~~reach~~ high value → How do we make sure that :- [How we control that]

↳ α_t does not be a big number, other wise there will be a huge gap between $\boxed{w_t \& w_{t-1}}$, so

It should happen, always there will be a $\boxed{w_t \approx w_{t-1}}$

↳ so we use $\boxed{\delta dw}$ instead $\boxed{\alpha_t}$

$$\boxed{\eta' = \frac{\eta}{\sqrt{\delta dw + \epsilon}}} \Rightarrow \delta dw = 0$$

↑ data

$$\boxed{\delta dw_t = \beta \delta dw_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2}$$

↳ Let's assume that $\boxed{\beta = 0.95}$

$$\boxed{\delta dw_t = (0.95)\, dw_{t-1} + (0.05)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2}$$

* Now if $\left(\frac{\partial L}{\partial w_t}\right)^2$ increase in a huge term then by multiplying it ~~with~~ by 0.05 it automatically reduced. In this way we will try reduce & control to become huge value.

$$\cancel{\eta' = \frac{\eta}{dw + \epsilon}} \qquad \boxed{\eta' = \frac{\eta}{\sqrt{\delta dw + \epsilon}}}$$

* Here value of η → learning or η' Eta is controled or adaptive (changeable) but smoothening of curve is not considered. So we will use Adam optimizer which contain both Smoothening & Adaptive learning rate.

6 **Adam optimizer**: It is called a Best optimizer which is a combination of both:

$$\boxed{\text{Adam optimizer} = \text{Momentum} + \text{RMSprop}}$$

= Smoothening + Adaptive Learning Rule

↳ Instead of writing V_{dw}; I am write V_{db} - bias updation
↳ weight updation

↳ $V_{dw} = 0$, $V_{db} = 0$ $S_{dw} = 0$ $S_{db} = 0$ } initially all values 0

↳ weight updation

$$\boxed{w_t = w_{t-1} - \eta' V_{dw}}$$

|| y Bias updatation

$$\boxed{b_t = b_{t-1} - \eta' V_{db}}$$

$$\boxed{\eta' = \frac{\eta}{\sqrt{S_{dw} + \epsilon}}}$$

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

$$\boxed{V_{db_t} = \beta \times V_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial b_{t-1}}}$$

Advantages

1. Solve problem of smoothening
2. Learning rate is Adaptive.

* Convolutional Neural Network:- CNN emerged from the study of the cerebral cortex of human brain's & they have been used in image recognition since the 1980.

↳ In cerebral cortex, our main focus on visual cortex also called as V_1, V_1 is a first part of a brain that begins to perform significantly advanced processing of visual input.

↳ In Cartoon view, images are formed by light arriving in the eye and stimulating the retina the light sensitive tissue in the back of the eye.

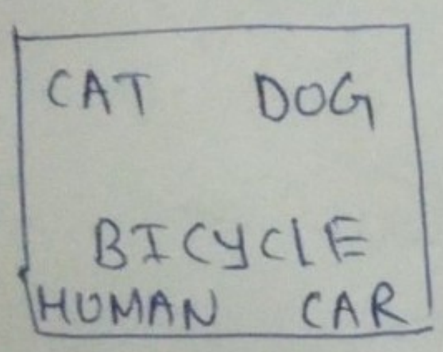
↳ Neuron present in the retina perform simple preprocessing of the image, but does not alter the way it is represented.

↳ Image then passes through the optic nerve & brain region called Lateral geniculate nucleus.

↳ Anatomical region just carry signal from eye to V_1 which is located at the back of head.

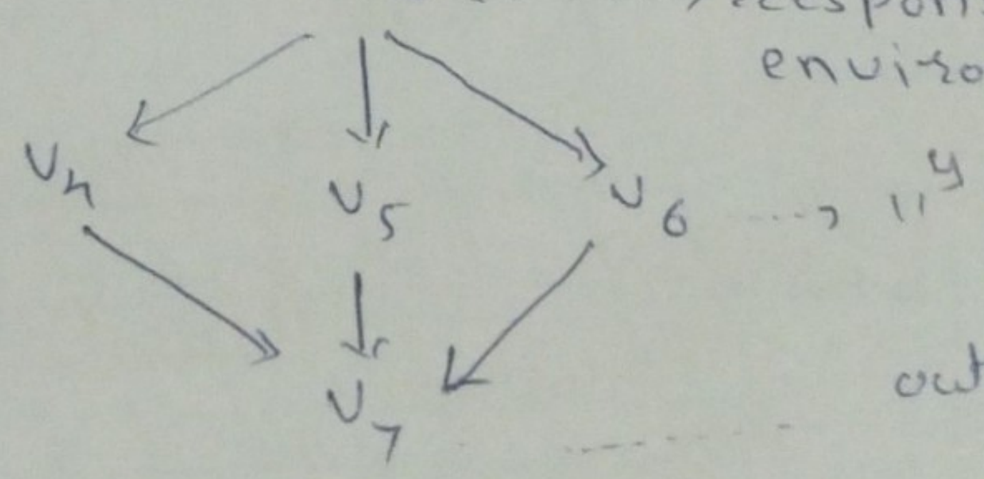
↳ suppose i am seeing specific image: ~~of~~ I may have cat
I may have Dog
I may have Bicycle
I may have Bird
etc.

↳ Anatomical region carry object image & signal from eye to cerebral cortex.



Image

cerebral cortex
↓ .... which contain visual cortex
visual cortex ....
↓ .... It may have many layers represented as V_1, V_2 ---etc
V_1 — .... This layer May responsible to recognize object
↓
V_2 .....> responsible for to identify nature of object (e.g. Animal)
↓
V_3 .....> responsible for map the environment & its features.



↓ O/P:- CAT

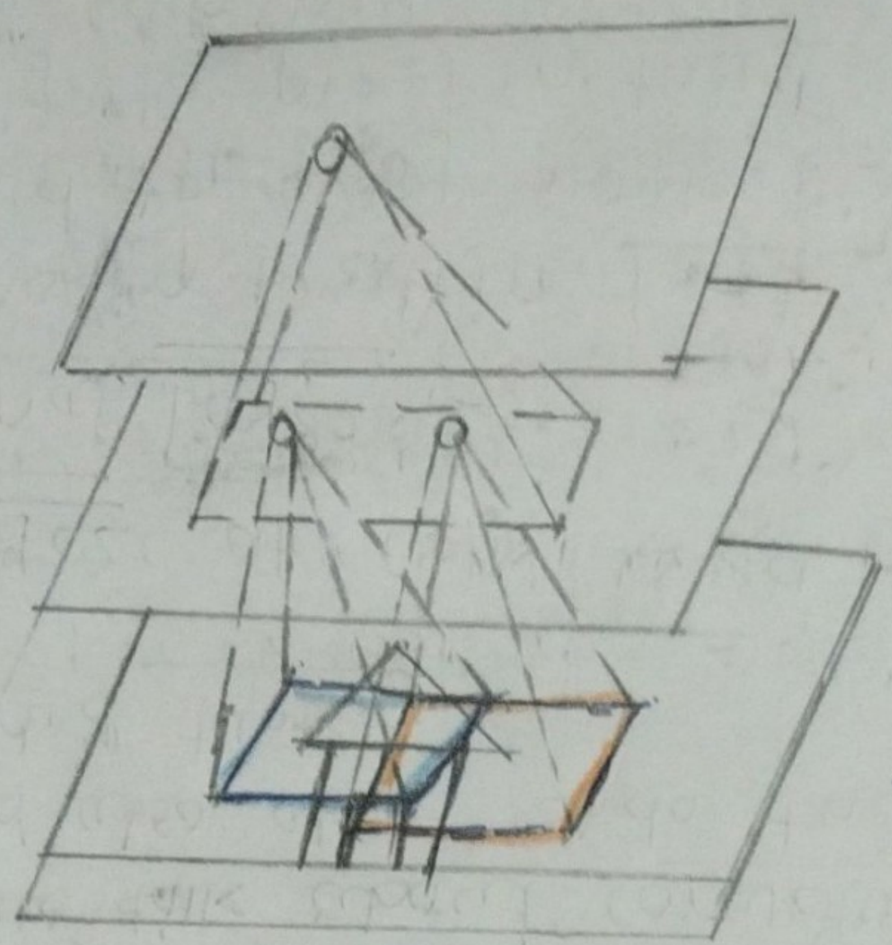
↳ ||y If we talk about convolution Neural Network, It should also able to do the same kind of above mentioned tasks.

↳ First layer of CNN is a convutional Layer.

① convutional Layers:-

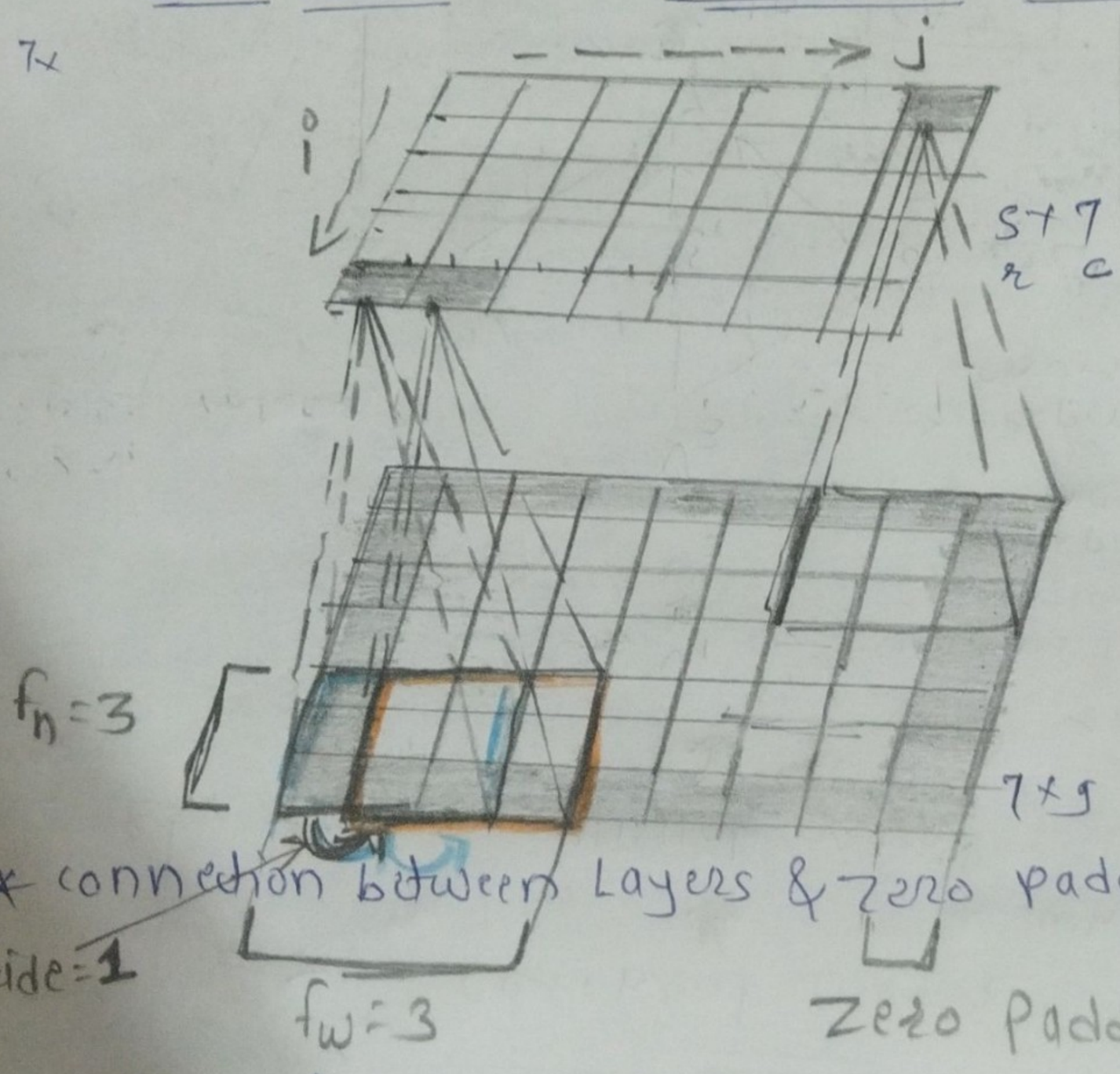
↳ Neurons (~~in first~~) in first convulional Layer are not connected to every single pixel in the input image but only to pixel in their receptive fields.

↳ In turn, each neuron in second Layer (convolutional) is connected only to neuron located within small rectangle in layer first



* CNN layers with rectangular local receptive fields

5×7 7×9



↳ A Neuron located in row i, column j of a given layer is connected to the outputs of the neuron in the previous layer located in row i to $i + f_h - 1$, columns j to $j + f_w - 1$, where f_h & f_w are height & width of the receptive field.

↳ In order for a layer to have the same height & width as the previous layer, it is common to add zeros around the inputs. This is called <u>zero padding</u>.

↳ row i to $i + f_n - 1$

stride = 1

$= 1$ to $1 + (3-1)$

$= 1$ to $1 + 2$

$= \boxed{1 \text{ to } 3}$ <u>row</u>

$= j$ to $j + (f_w - 1)$

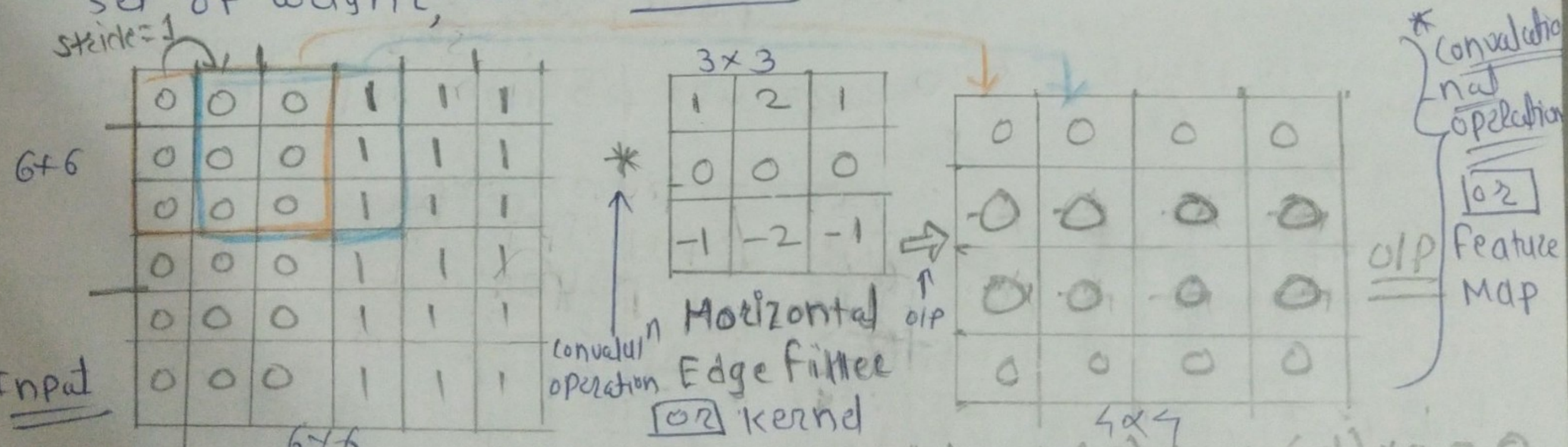
$= 1$ to $1 + (3-1)$

$= 1$ to $1 + 2$

$= \boxed{1 \text{ to } 3}$ <u>columns</u>

↳ The shift from one receptive field to the next is called <u>stride</u>.

↳ A <u>Neuron's</u> <u>weights</u> can be represented as a <u>small image</u> the size of the receptive field, there are two possible set of weight, called <u>Filters</u>. (Horizontal & vertical filter)



① $1\times0 + 2\times0 + 1\times0 + 0\times0 + 0\times0 + 0\times0 + (-1)\times0 + (-2)\times0 + (-1)\times0 = 0$

② $1\times0 + 2\times0 + 1\times1 + 0\times0 + 0\times0 + 0\times1 + (-1)\times0 + (-2)\times0 + (-1)\times1 =$
$0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + (-1) = 0$

③ $1\times0 + 2\times1 + 1\times1 + 0\times0 + 0\times1 + 0\times1 + (-1)\times0 + (-2)\times1 + (-1)\times1 =$
$0 + 2 + 1 + 0 + 0 + 0 + 0 + (-2) + (-1) = 0$

④ $1\times1 + 2\times1 + 1\times1 + 0\times1 + 0\times1 + 0\times1 + (-1)\times1 + (-2)\times1 + (-1)\times1 =$
$1 + 2 + 1 + 0 + 0 + 0 + (-1) + (-2) + (-1) = 0$

⑤ $1\times0 + 2\times0 + 1\times0 + 0\times0 + 0\times0 + 0\times0 + (-1)\times0 + (-2)\times0 + (-1)\times0 = 0$

⑥ $1\times0 + 2\times0 + 1\times1 + 0\times0 + 0\times0 + 0\times1 + 0\times(-1) + 0\times-2 + 0\times1 =$
$0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + (-1) = 0$

⑦ $1\times0 + 1\times2 + 1\times1 + 0\times0 + 0\times1 + 0\times1 + (-1)\times0 + (-2)\times1 + (-1)\times1$
$0 + 2 + 1 + 0 + 0 + 0 + 0 + (-2) + (-1) = 0$

* Pooling :-

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

1	1	1
0	0	0
-1	-1	-1

$\Rightarrow$

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Horizontal Edge Filter kernel

Let's consider i will be getting above value as o/p

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

6x6

$10 + (-10) + (-10) = -10$

$*$

1	0	-1
1	0	-1
1	0	-1

3x3

vertical Edge Filter kernel

$\Rightarrow$

0	30	30	0
0	10	10	0
0	-10	-10	0
0	-30	-30	0

4x4

$\Rightarrow$ if we Apply Feature scaling then -4 become 0 & 0 become 255

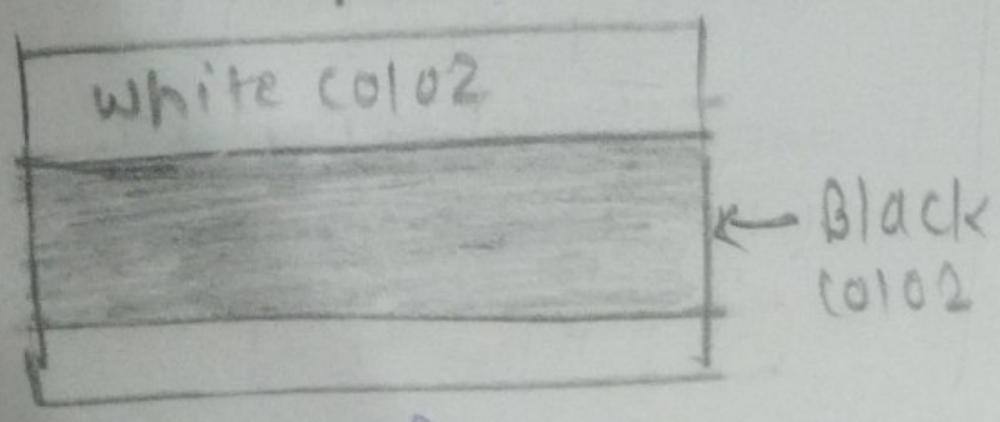
↳ Lowest value will get converted 0 & highest value get converted 255.

① In Horizontal Edge filter

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

0 – Black
255 – White

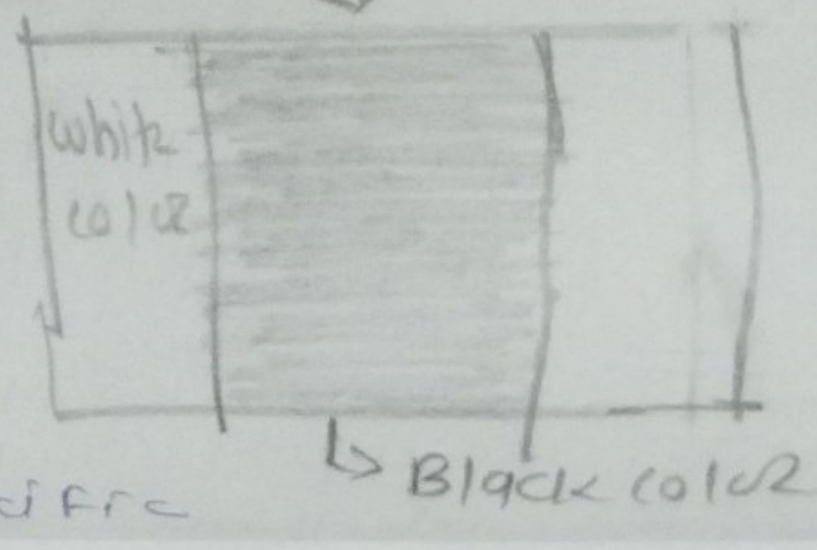
$\Downarrow$



② In vertical Edge filter

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

$\Downarrow$



This process is Basically called convolution operation.

* so with help of filter extract specific

↳ Input given here is 6x6 matric, so n = 6, & filter of 3x3 then we will be getting output of 4x4.

n = 6 - input
f = 3 - filter
} o/p = 4x4

↳ So $\boxed{\text{Formula} = ((n-f)+1)}$ = ~~6 - (3+1)~~

information from image.

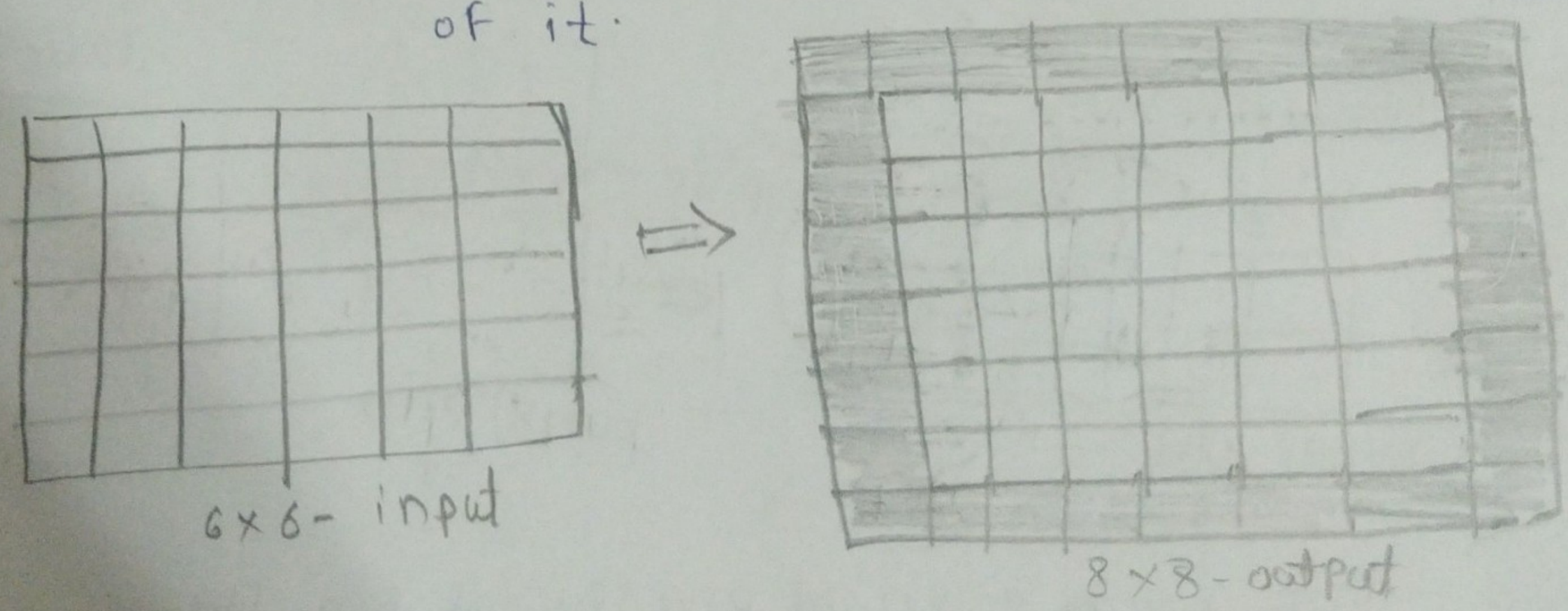
↳ Input given here is 6×6 matrix, so n = 6, & filter of 3×3 then we will be getting output of 4×4.

n = 6 - input
f = 3 - filter } o/p = 4×4

↳ So $\boxed{\text{formula} = ((n-f)+1)}$ = ~~6 - (3+1)~~
(6 - 3) + 1 = 3+1 = 4

** if i give 6×6 input then i will be getting 4×4 output here image size is decreases, & this should not happen. if image size decrease it mean loosing some kind of information. Now in order to prevent we will have one term called padding.

* padding :- Applying layer (Extra) on top of image is called padding. It means protecting image (6×6) by adding extra layer on top of it.



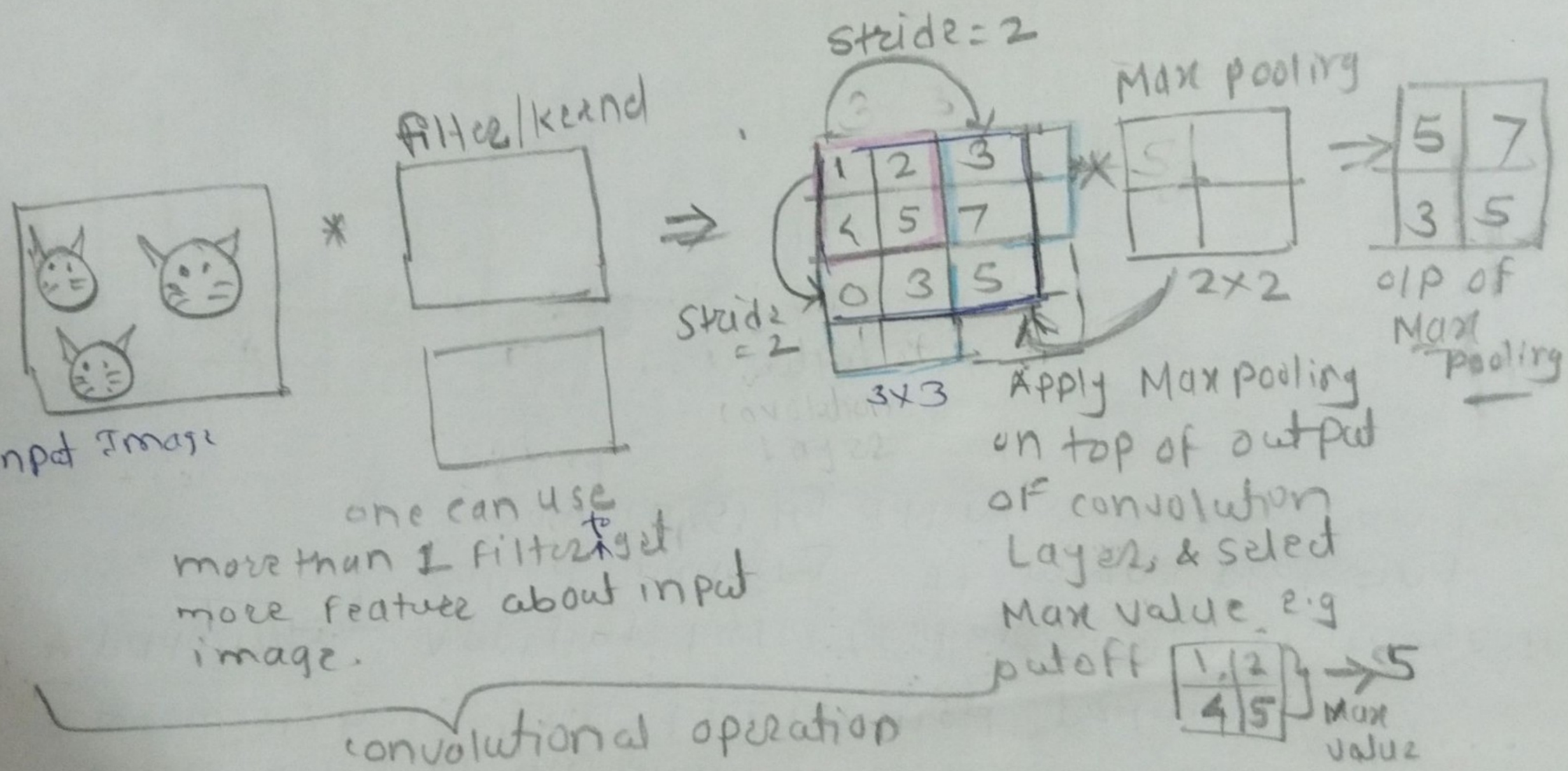
Formula = (n-f) + 1 = (8-3) + 1 = 5+1 = 6 ⇒ so o/p = 6×6 image.

* Pooling Layer : Main Goal of pooling is to subsample (shrink) the input image in order to reduce computational load, Memory usage, & number of parameters.

→ Just like convolution layer, each neuron in a pooling layer is connected to the output of a limited number of neurons in

→ the previous layer, located within a small rectangular receptive field.

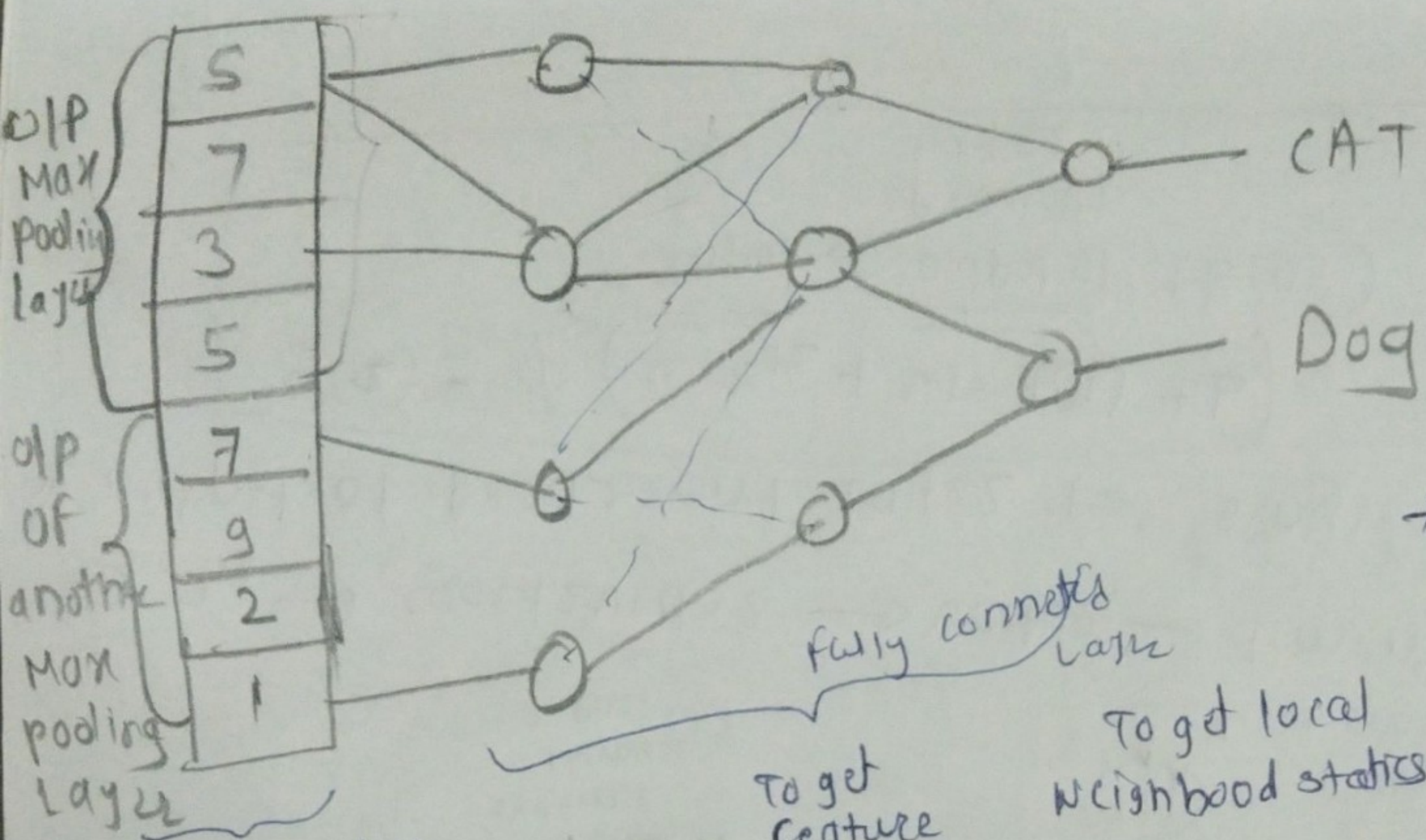
* My convolutional N/W, till it goes to next convution ~~N/W~~ layer, it should extract more, better & in a clear way.
so extracting more, better & clear information from input image is called Max pooling. different types of pooling avg pooling, Min pooling & Max pooling



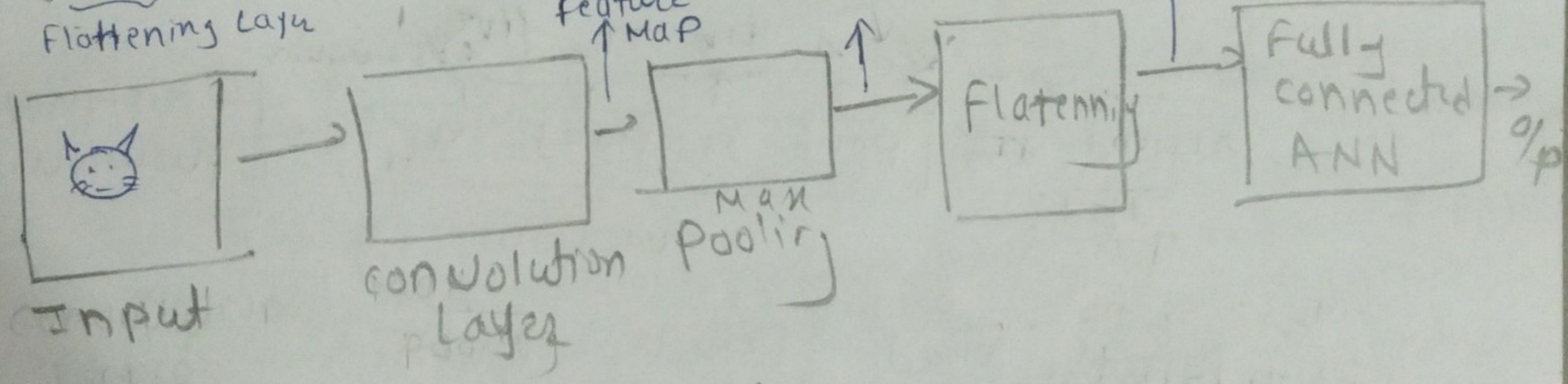
*

* if i have multiple object in an input image, & one of my filter able to determine that object, it will able to clearly pickup that using Max pooling.

* Flattening Layer:— It looks like ANN Dense Layer
↳ whatever output we will get from max pooling, just we will going to straight forward or flatten or elongate



* Finally feature map is converted to vector called <u>Flattening</u>

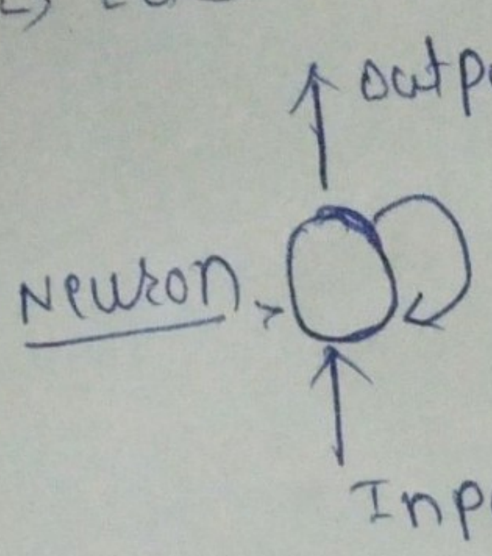


* <u>CNN Architecture</u>

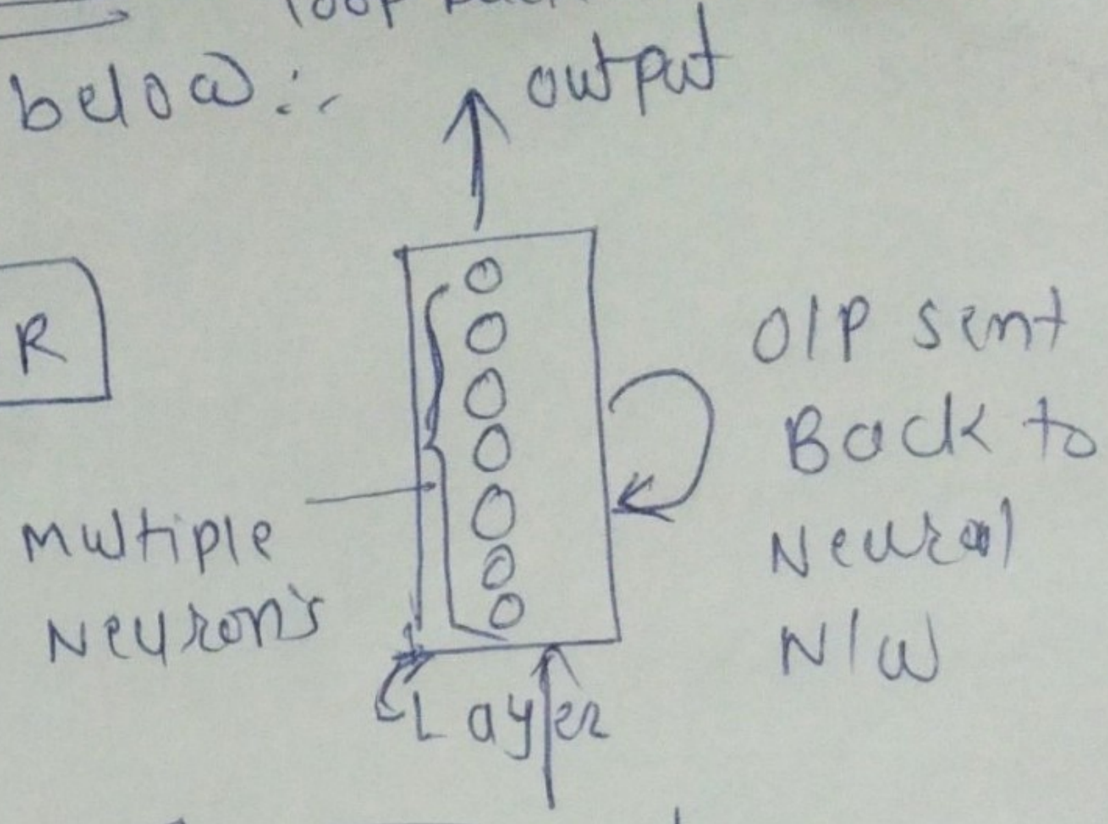
* Recurrent Neural Network [RNN] :

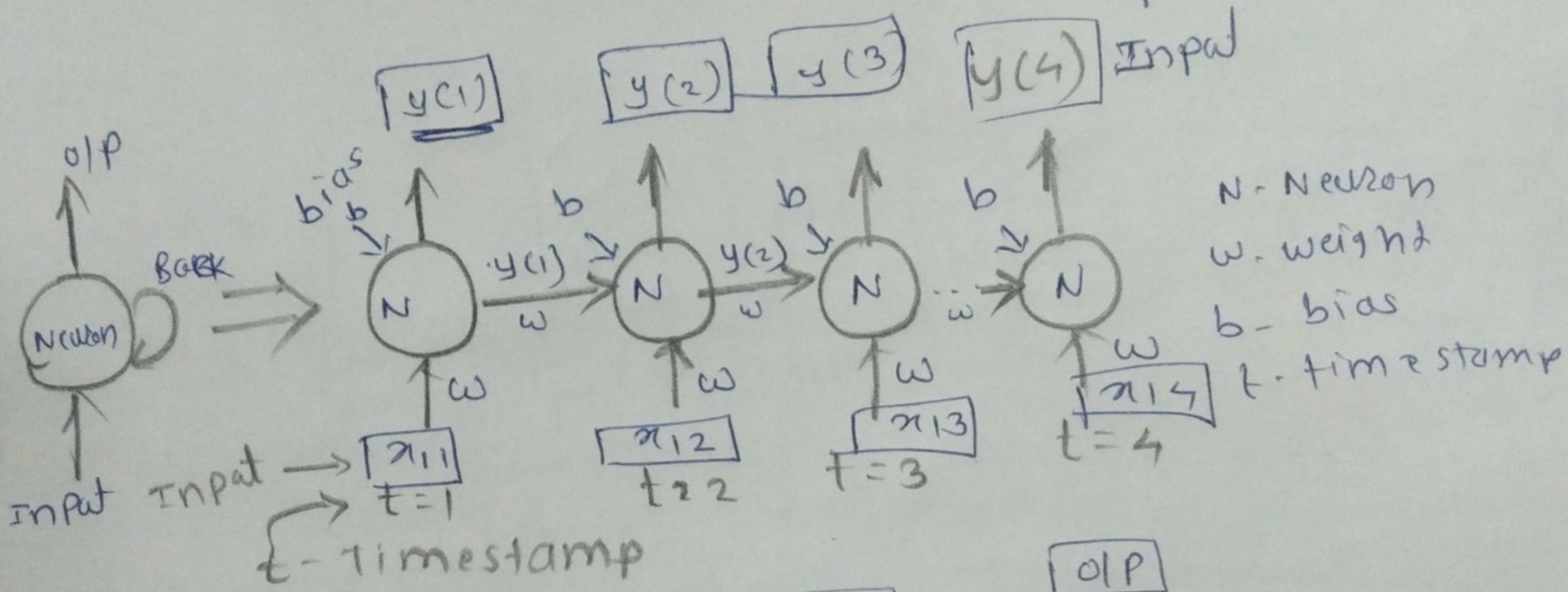
→ N/w has connectn that loop back on themselve.

↳ Let's on Neuron is given below :-



OR





sentiment Analysis } ⇒ $\boxed{\text{The food is good}}$ ⇒ $\boxed{\text{O/P}}$ positive

sentence1 x_{11} x_{12} x_{13} x_{14}

sentence Number & Part

x_{11} — first word in first sentence

first sentence

↳ x_{11} → word2vector → vector → dimension (d) → $\boxed{300}$ feat

[Let's sa 300

* output of Recurrent layer for single instance

$$y(2) = \phi\left(\underbrace{w\,x_{12}}_{\text{Input}} + \underbrace{w \times y(1)}_{\text{output of previous layer}} + b\right)$$

(Input + output of previous layer + bias)

$$\boxed{y(t) = \phi\left(\omega_x x_t + \omega_y y(t-1) + b\right)}$$

t - maximum time stamp.

* Types of RNN :-

1. one to one RNN
2. one to Many RNN
3. Many to one RNN
4. Many to Many RNN

* Training RNNs :-

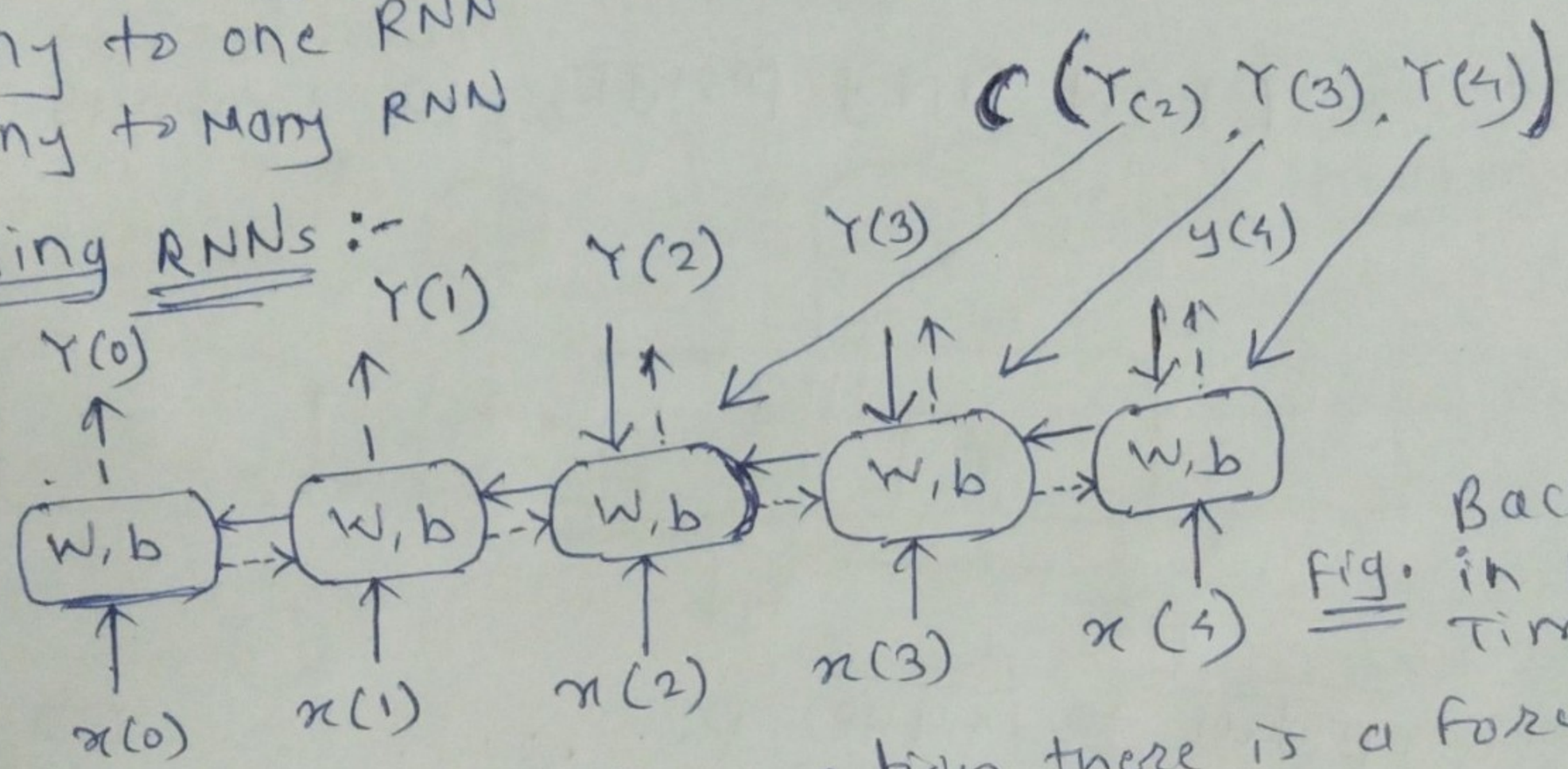


Fig. Backpropagation in RNN through Time

Just like regular Backpropagation, there is a forward pass through the unrolled network. Then output sequence is evaluated using cost function $C(Y_{(0)}, Y_{(1)}, Y_{(2)} \cdots Y_{(T)})$ where (T - is the max time step).

↳ gradient of cost function are then propagated backward through unrolled N/W.

↳ Finally Model's parameters updated using gradients computed during Backpropagation through time [BPTT].

↳ cost function is calculated using last three outputs of the networks $Y_{(2)}, Y_{(3)}, Y_{(4)}$, so gradient flow through ~~has~~ three outputs.

* Drawbacks of RNN :-

1. In RNN, there is a problem vanishing gradient problem & during backpropagation weights are getting updated using chain rule, at that time weights become very small value, so weight updation doesnot happen.

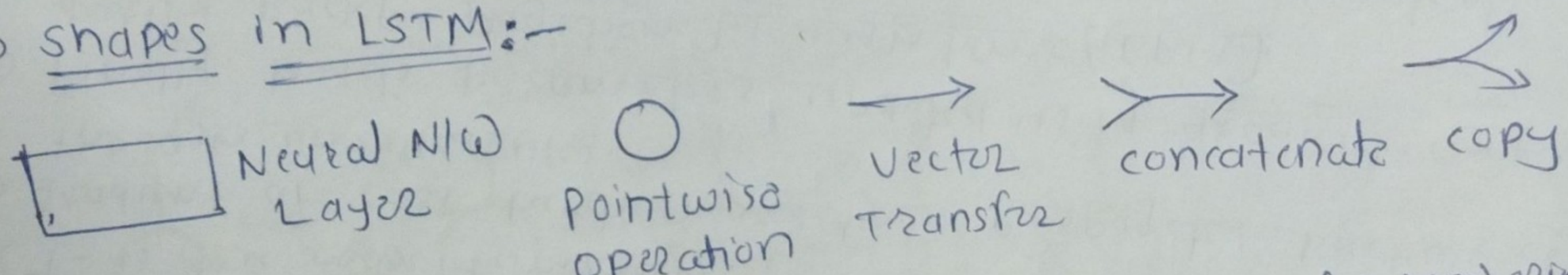
LSTM & GRU

↳ To overcome problem of RNN, i.e vanishing gradient problem in RNN.

↳ so LSTM explicitly designed to avoid long term dependancy problem, so focus is to resolve vanishing gradient problem.

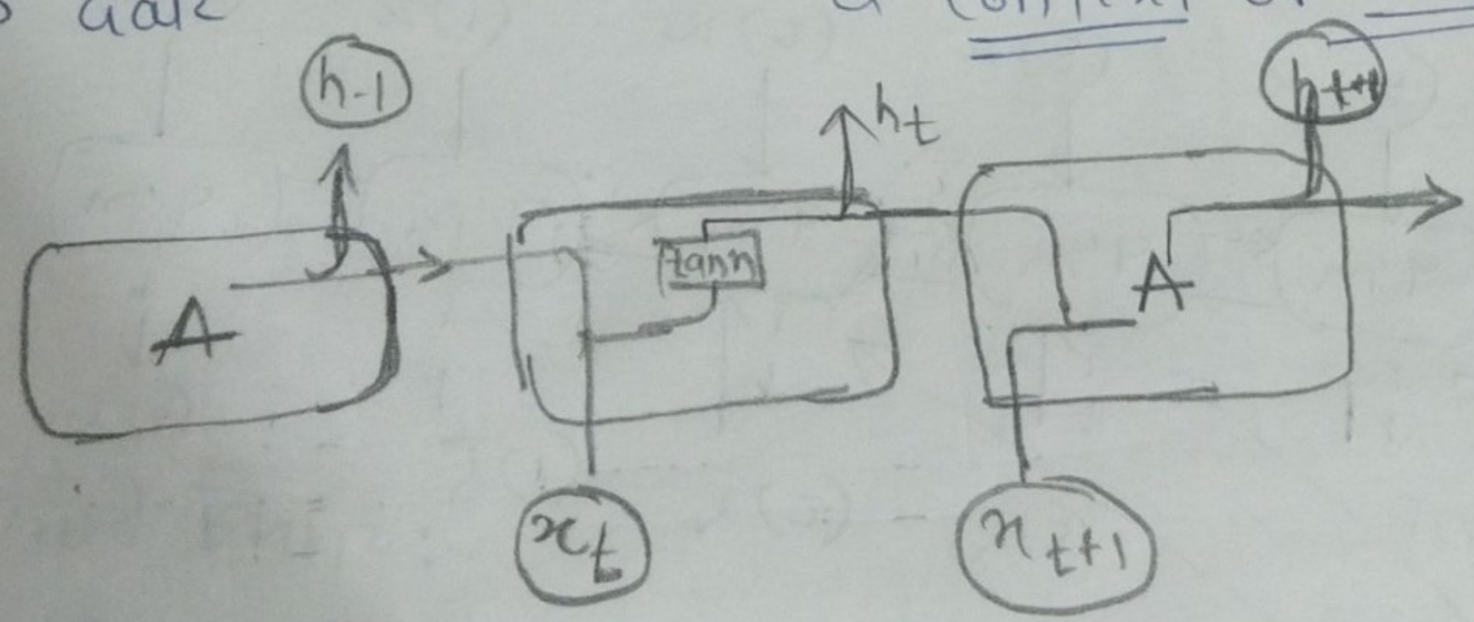
↳ It is designed by Hochreiter & schmidhuber in 1997.

↳ shapes in LSTM:-

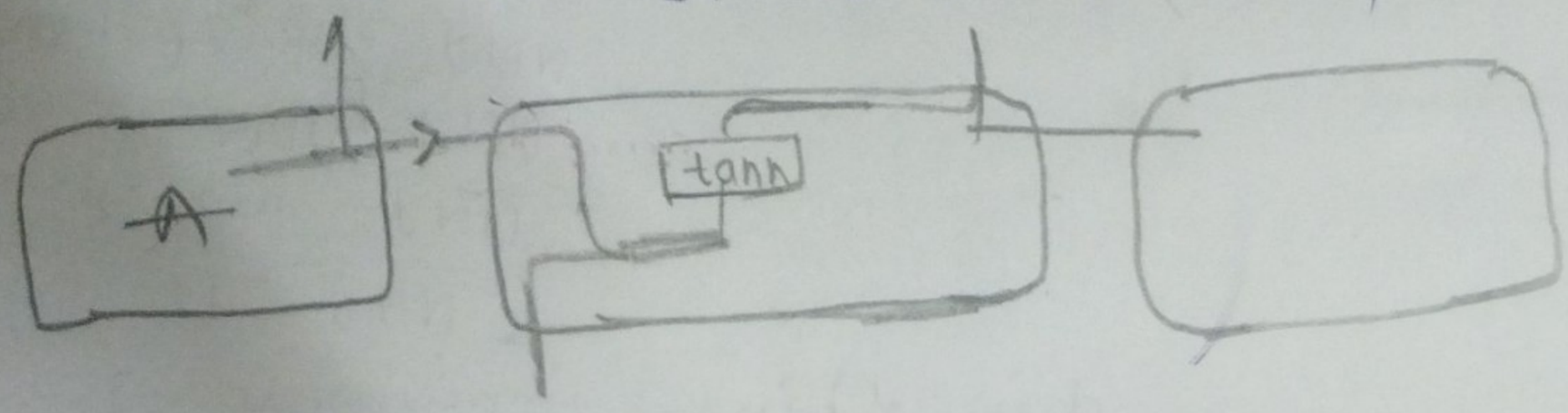


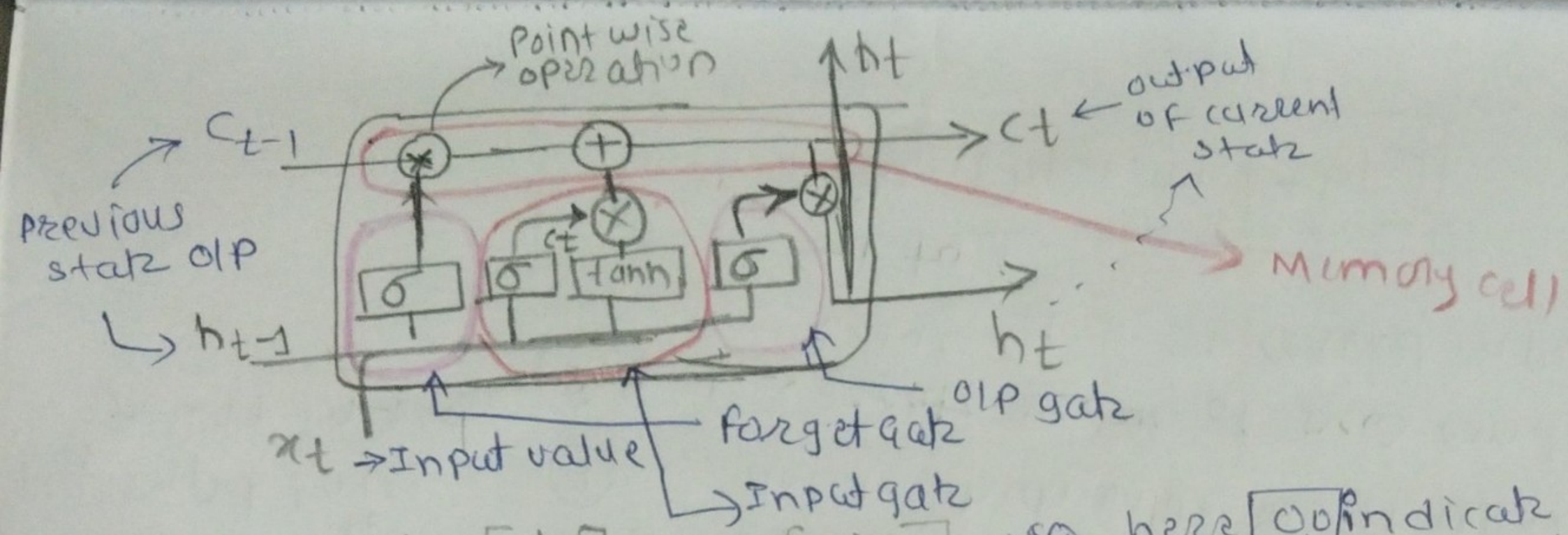
① Memory cell
② forget gate
③ Input gate
④ o/p gate

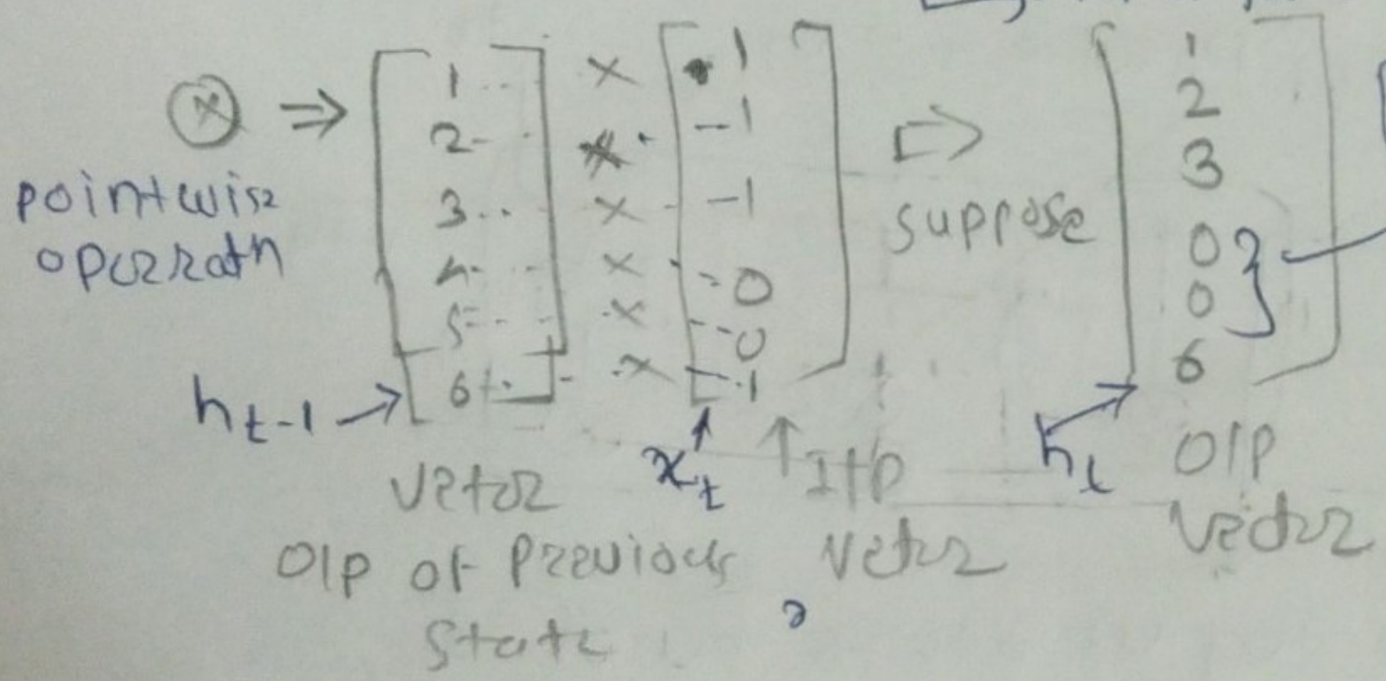
① Memory cell :- Memory cell is used for remembering & forgetting things. How do we remember & forget it is based on a context of input.



* structure of Typical RNN Model





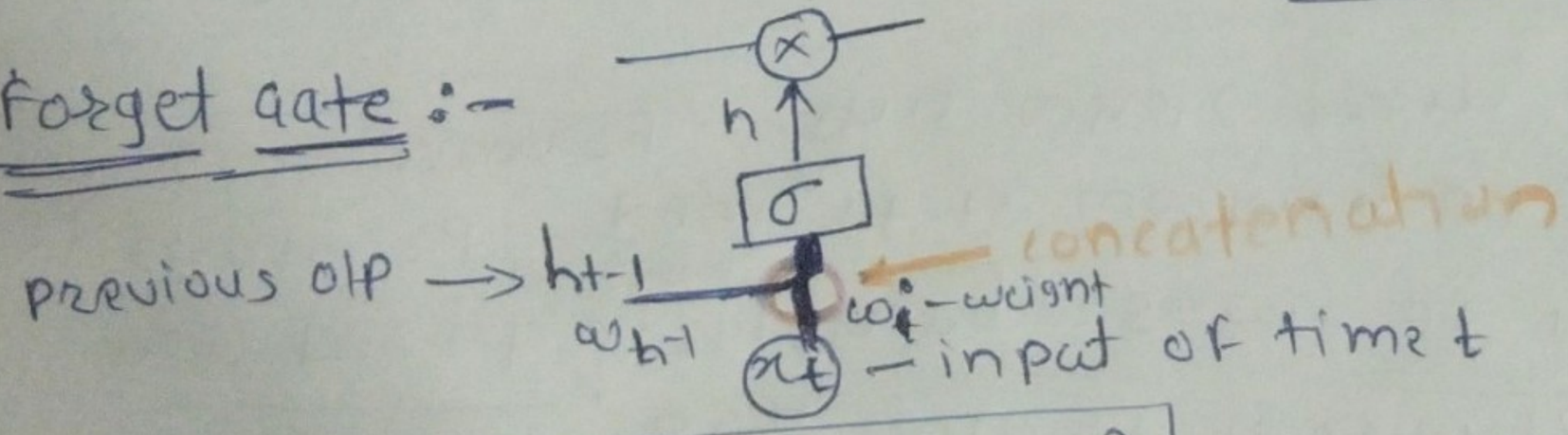


so here [0,0] indicate forgetting specific information.

↳ bcz content is changed

↳ so whenever content is changing LSTM RNN forgetting some information. but we should add some new information → ⊕ operation

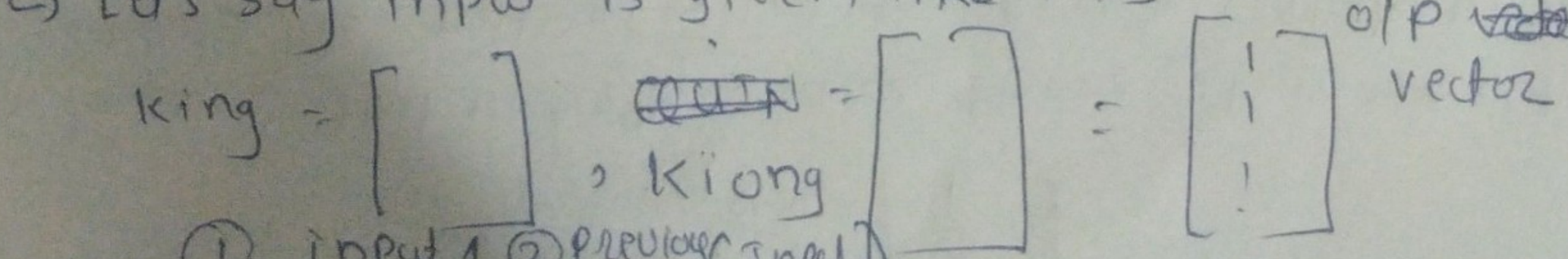
* Forget gate :-



$$\boxed{f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)}$$ forget gate

↳ W_f is a combination of 2 weights (w_i & w_{h-1}) $[h_{t-1}, x_t]$ (e.g $y = w^T x + b$)

↳ Let's say input is given like this



↳ when vector is similar, or close to similar, we will be getting more number of 1 in o/p vector. It means that content has not changed.

↳ Input1 : king [ ] Input2 : poor [ ] = [1, 0, 0, 0, 1] o/p vector

previous state output

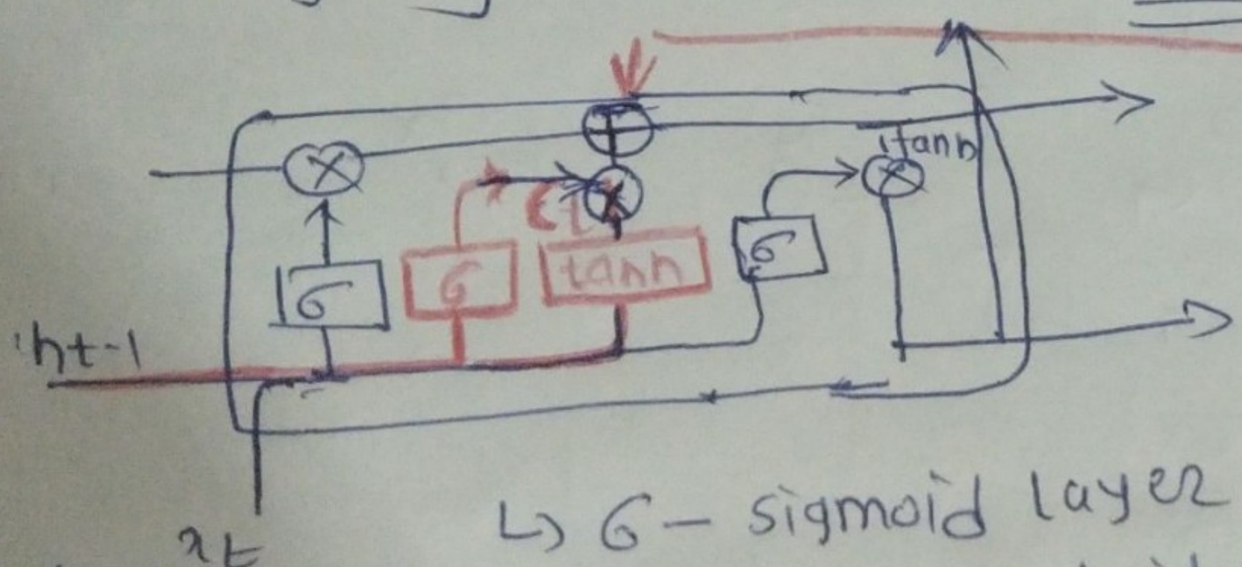
↳ Input Output vector (h) 0's are there, it means that input1 is not similar to previous state vector (Input2) so output vector contain lot's of zero's (when we apply or pass h to point wise operation in diagram)

↳ σ — It is a sigmoid activation function.

↳ lot's of zero's in output vector it mean it is forgetting information, bcz context is change but it will not forget everything, some of information still preserved captured, maintained. Now it is working as Long short term ~~work~~ Memory. This whole term is called cell state

↳ Now Next step is to decide, what new information we're going store in the cell state.

↳



Adding Information to the Memory cell

Fig Input Layer shown in red.

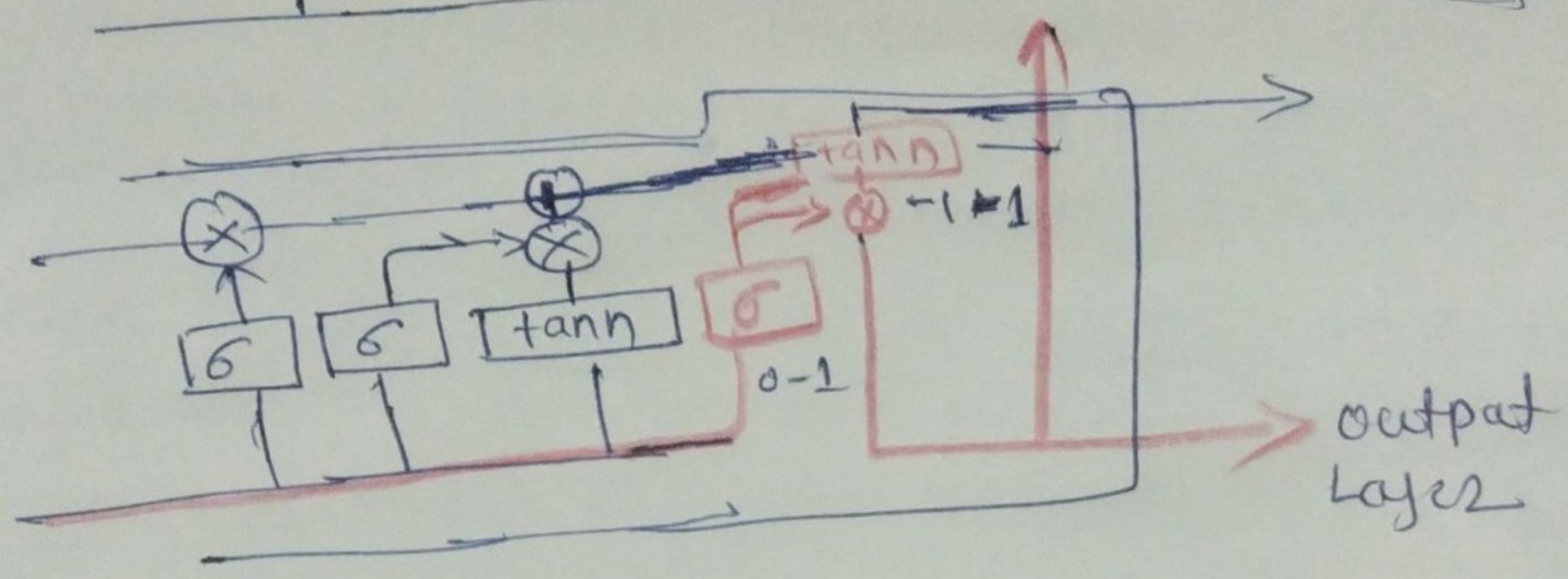
↳ σ — sigmoid layer called $\boxed{\text{input gate layer}}$

① decide which value we'll update

② tanh → create vector of new canditate value C_t. It convert input into −1 to 1.

$$\boxed{i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)}$$

$$\bar{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$



* output layer (shown in red) between

① σ- sigmoid function pass information in ~~the form of~~ 0 & 1

② Than function passes data in between -1 to 1

o_t