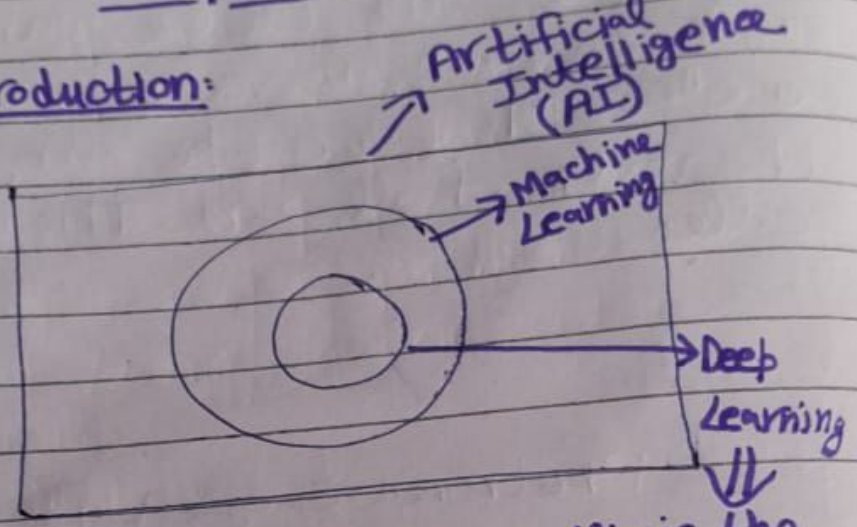
Deep Learning

· Introduction:



Deep Learning
⇓
Mimic ~~the~~ human brain
⇓
Multi-layered Neural Networks

Deep Learning:

① ANN → Artificial Neural Network → Regression / Classification

② CNN → Convolutional Neural Network
↳ Input data is in form of images and video frames
↳ RCNN, YOLO V5 etc used for object Detection in CV.

Computer Vision

③ RNN → Recurrent Neural Network

↳ NLP → Input: Text, Time Series

↳ { ~~Word Embedding, LSTM RNN, GRU RNN,~~ ~~Bidirectional LSTM RNN, Encoder/decoder, Transformers, BERT~~ }

Framework to be used: TensorFlow

↓

Developed by Google

* Why Deep Learning is becoming popular?

2005 → Facebook, Youtube, Whatsapp, LinkedIn, Twitter

⇓

Data was generated exponentially (Images, Text, Documents, Videos)

↳ **Big Data** (Efficient storage and retrieval of data)

(HADOOP)

2011 – 2012 →

① Hardware Requirement

↓

GPU's price decreasing

(Easy to train DL models)

② Huge amount of data is getting generated → Deep learning performs well

③ As the data getting increase the traditional Machine Learning Algorithms improve performance upto a limit but Deep Learning models keep on increasing performance.

④ Deep Learning is being used in many domains:
(i) Medical
(ii) Ecommerce
(iii) Retails
(iv) Marketing and many others

⑤ There are famous opensource frameworks for Deep Learning:
(i) **Tensorflow** (by Google)
(ii) **Pytorch** (by Facebook)

⇒ Perceptron Intuition: (Artificial Neuron or Neural Network unit)

- Perceptron is a single layered Neural Network.

Some Important things to be discussed here are:

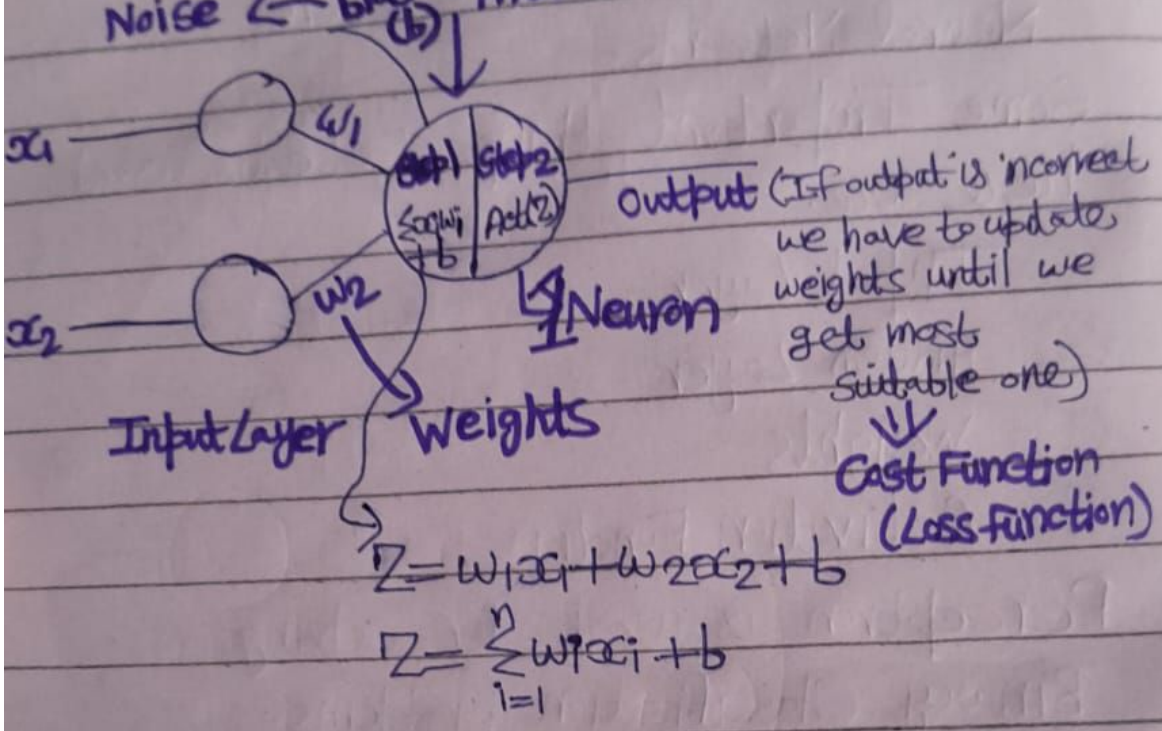
1. Input Layer
2. Hidden Layer
3. Weights
4. Activation Function

- Perceptron is used for solving Binary Classification problems.

Let we have Dataset:

x_1	x_2	
IQ	No. of study hours	Output (Pass/Fail)
95	3	0
110	4	1
100	5	1

As we have 2 input feature so in our Input Layer we will have 2 inputs



(If output is incorrect we have to update weights until we get most suitable one)

⇓

Cost Function (Loss Function)

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

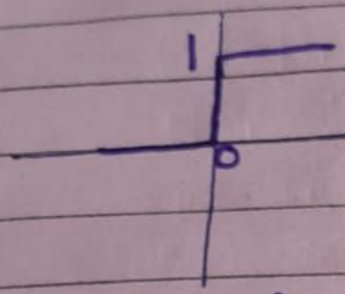
$$Z = \sum_{i=1}^{n} w_i x_i + b$$

Activation Function:

In step2, we will apply activation function on Z. It is used to transform the output i.e from 0 to 1 or between 1 to −1

Some activation functions are:

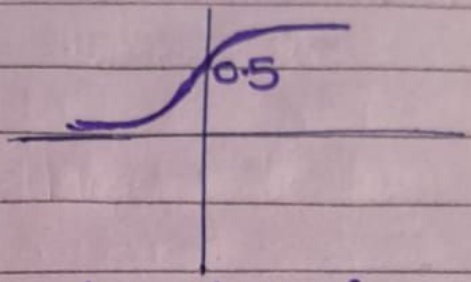
Step Function



(threshold value=0)

$$\begin{cases} 0 \text{ if } z \leq 0 \\ 1 \text{ if } z > 0 \end{cases}$$

Sigmoid Function



(threshold value = 0.5)

$$\begin{cases} 1 \text{ if } z > 0.5 \\ 0 \text{ if } z \leq 0.5 \end{cases}$$

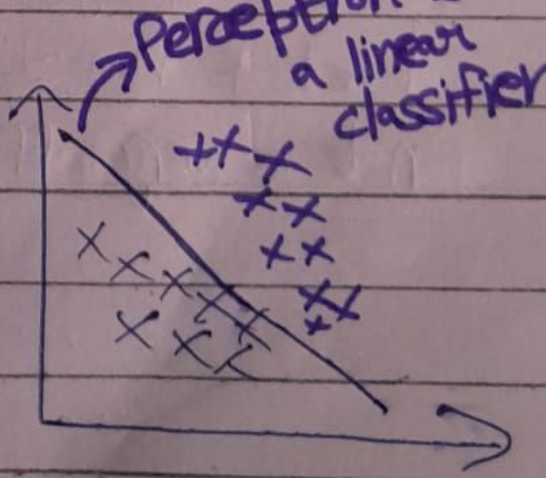
- **Step1:**

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

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

⇓ It is similar to linear classification

$$y = mx + c$$

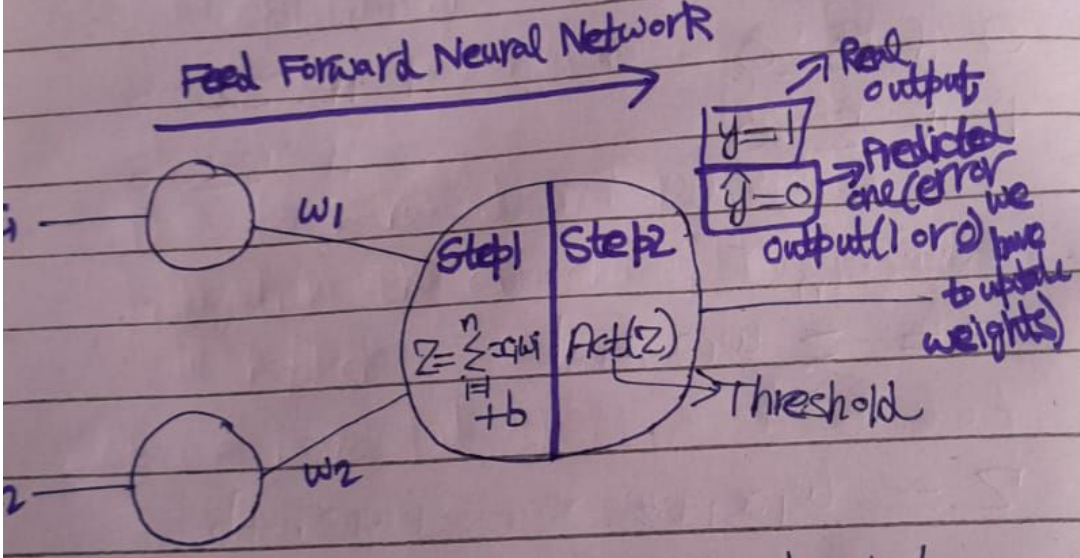


Perceptron Models

↓ Single Layer perceptron Model

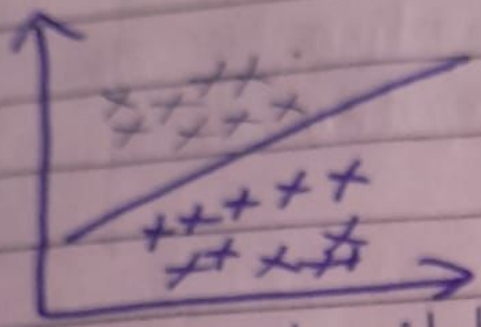
↓ Multi-Layered Perceptron Model

⇓ ANN



* Every time we get error we have to go back change weights and reperform Feed Forward Neural Network. It's not a very efficient technique

But in Perceptron (Single Layered) it is only good for Linearly Seperable data



* It is not suitable for Non-Linear.

Multi-layered Neural Network

offers:

1. Forward Propogation
2. Backward Propogation
3. Loss Function
4. Activation Functions
5. optimizers

So Multi-layered Neural Network can be used for more complex problems and perform better.

Multi-layered Neural Network

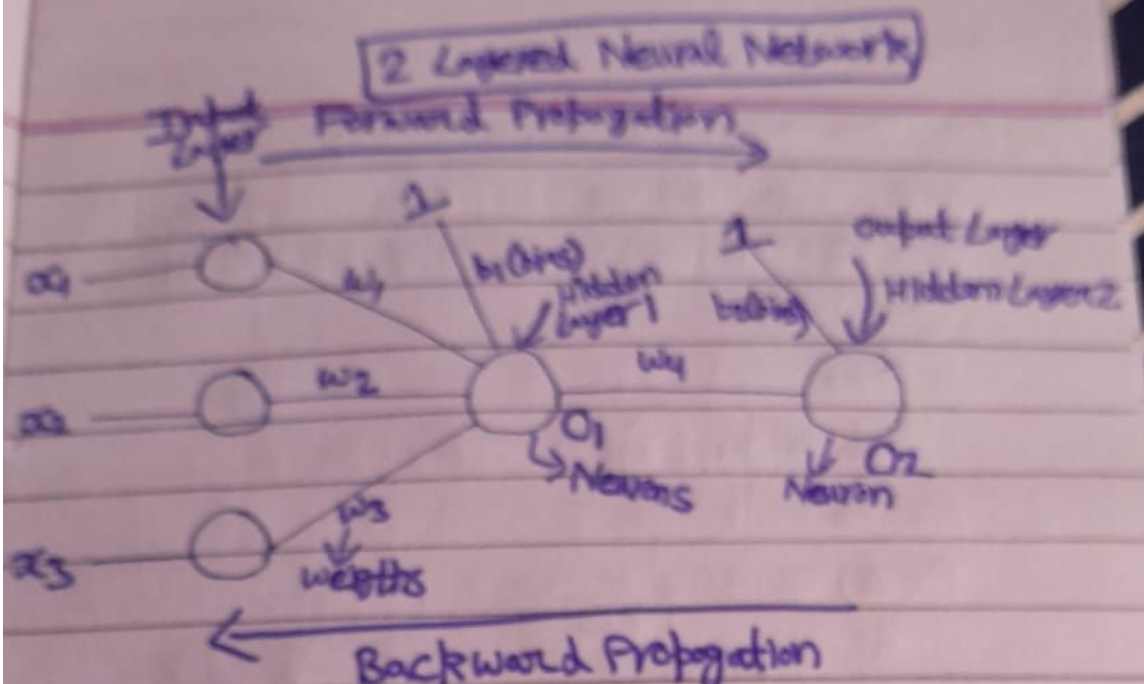
⇒ Artificial Neural Network (ANN) (Multilayered Perceptron Model)

① Forward Propogation
② Backward Propogation
③ Loss Function
④ Optimizers
⑤ Activation Function

* Due to Forward and Backward propogation the efficient updation of weights will happen

Let we have datased

x_1	x_2	x_3	
IQ	Study hours	Play hours	O/P (Pass/Fail)
95	4	4	1
80	5	2	1
95	2	7	0



① Forword Propogation:

Hidden Layer01:

Step1:

$$Z = x_1 w_1 + x_2 w_2 + x_3 w_3 + b_1$$

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

Let's initialize our weights randomly

$w_1 = 0.01, w_2 = 0.02, w_3 = 0.03$

and we are picking 1st record of our dataset

$x_1 = 95 \quad x_2 = 4 \quad x_3 = 4$

Let b_1 (bias) $= 0.01$

As $Z = x_1w_1 + x_2w_2 + x_3w_3 + b_1$

So,

$Z = 95 \times 0.01 + 4 \times 0.02 + 4 \times 0.03 + 0.01$

$\boxed{Z = 1.51}$

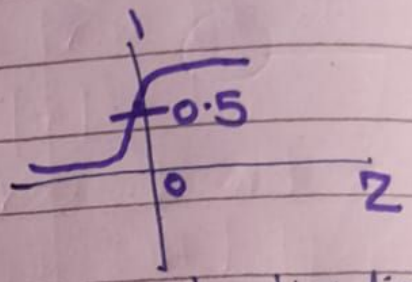
Step 02:

Activation (Z)

Sigmoid activation

$$f(Z) = \frac{1}{1+e^{-Z}}$$

$$f(Z) = \frac{1}{1+e^{-1.51}}$$



(Sigmoid activation keeps the value b/w 0 and 1)

$\boxed{O_1 = 0.759}$

⇓

output of Hidden Layer 1

Hidden Layer 2:

Step 1:

Now let $w_4 = 0.02$

and b_2 (bias) $= 0.03$

$Z = O_1w_4 + b_2$

$Z = (0.759)(0.02) + 0.03$

$\boxed{Z = 0.04518}$

Step 2:

Activation (Z)

$f(z) = \frac{1}{1+e^{-z}}$ (Sigmoid activation)

$f(z) = \frac{1}{1+e^{-0.04518}}$

$f(z) = 0.51129$

$O_2 = 0.51129$

⇓

Ouput of output Layer/Hidden Layer2

As we get output Now we will calculate

Loss Function:

$\hat{y} = O_2 = 0.51129$ (Predicted output)

$y = 1$ (Actual output of record we taken)

Loss Function = $y - \hat{y} = 1 - 0.51129$

Loss Function = 0.49

⇓

Error

- Our main aim is to reduce error.

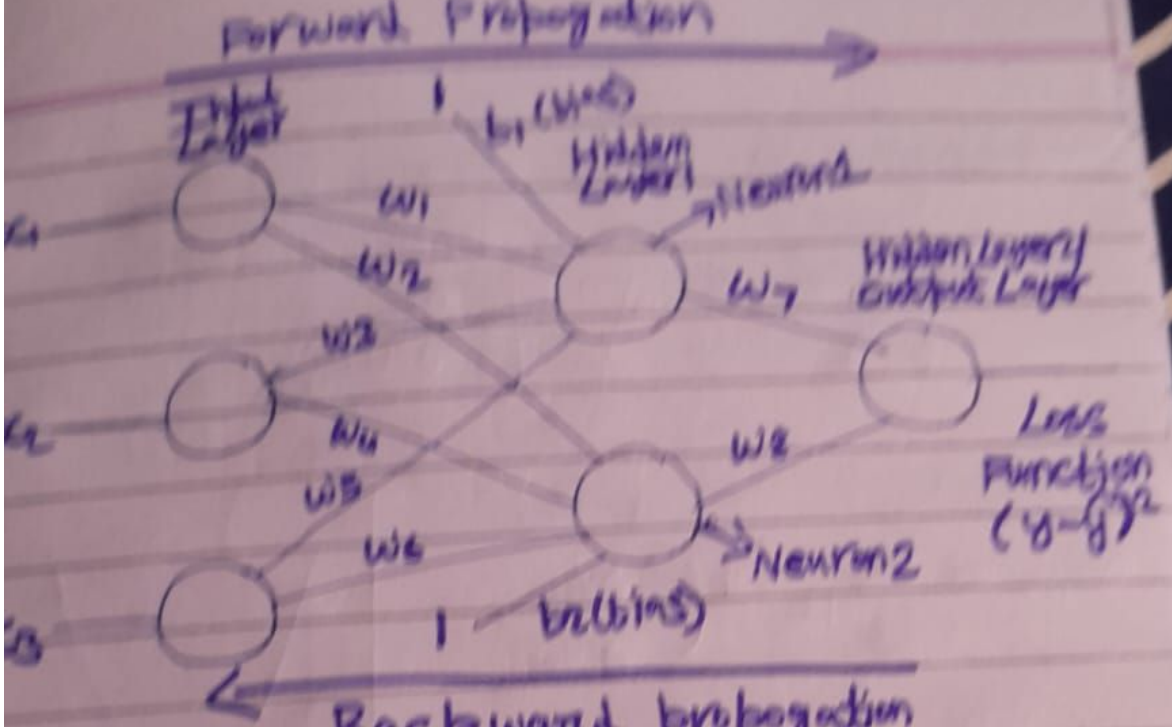
- We will reduce error by updating weights in the manner of **backward propogation.**
- After updation of weights by backward propogation, the next record will be passed from dataset and again forward propogation will happen. This forward and backward propogation will happen until we get minimized loss function

Backward Propogation and Weight Updation Formula:

Let we have dataset

Input Features

x_1 IQ	x_2 Study hours	x_3 Play hours	Pass/Fail Output
95	4	4	1
100	5	2	1
95	2	7	0



Loss Functions for Regression:

1. MSE
2. MAE
3. Huber Loss

Loss Functions for Classification:

1. Binary Cross Entity
2. Categorical Cross Entropy

- When loss function is high we reduce it by updating weights in backward propagation.

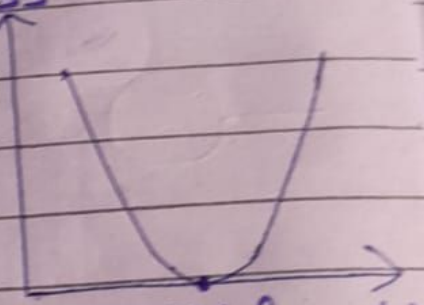
Weight Updation Formula:

$$w_{7_{new}} = w_{7_{old}} - \eta \left[\frac{\partial L}{\partial w_{7_{old}}} \right]$$

Slope (derivative of loss with w_7)

Learning Rate

$$w_{8_{new}} = w_{8_{old}} - \eta \frac{\partial L}{\partial w_{8_{old}}}$$



Generically:

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

(our aim is to reach global minima as slope is 0 there)

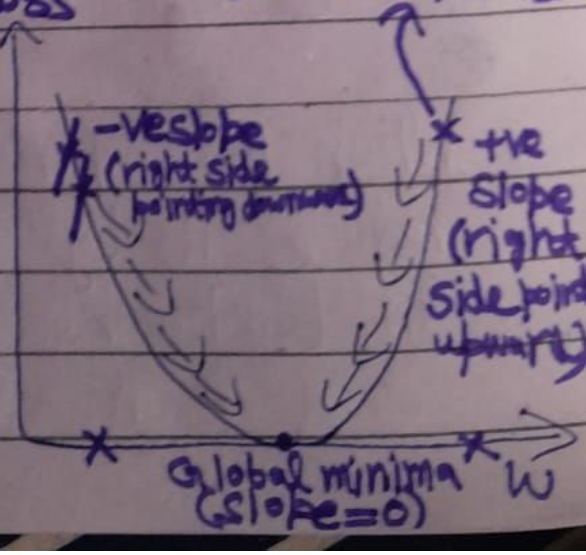
⇓

Weight Updation Formula

Gradient Descent optimizer

Optimizers:

To reduce the loss value



For -ve slope

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

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

$\boxed{w_{new} >> w_{old}}$ (Increase weight)

For +ve slope

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

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

$\boxed{w_{new} << w_{old}}$ (Decrease weight)

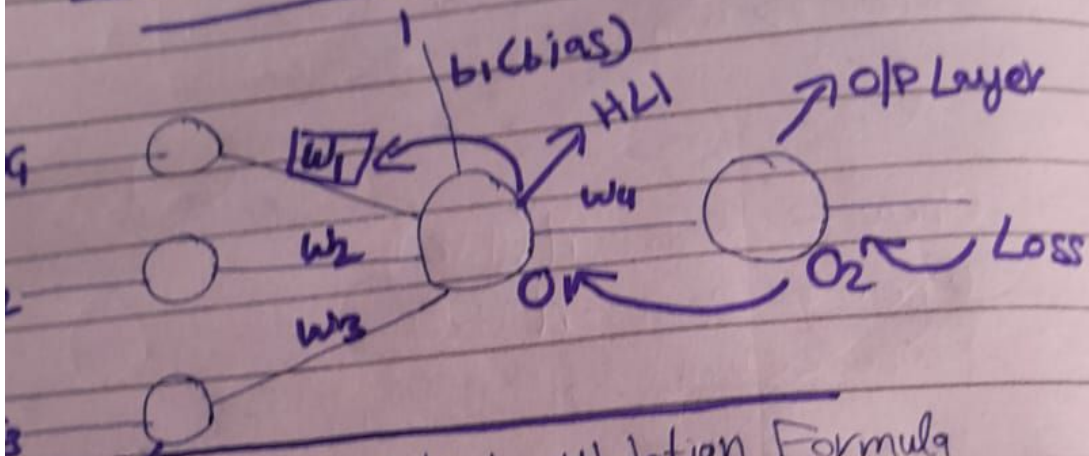
⇒ Learning rate (η) decides the step size how soon we have to converge so better to have smaller learning rate i.e 0.001

When w reaches global minima

$$\text{slope} \Rightarrow \frac{\partial L}{\partial w_{old}} = 0$$

$\boxed{w_{new} = w_{old}}$

Chain Rule of Derivative:



We know Weigh Updation Formula

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

We start backward so first update W_4

$$W_{4new} = W_{4old} - \eta \boxed{\frac{\partial L}{\partial W_{4old}}}$$

Expand

$$\frac{\partial L}{\partial W_{4old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial W_4} \Rightarrow \text{Chain Rule of Derivative}$$

Now update W_1

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

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_{1old}}$$

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

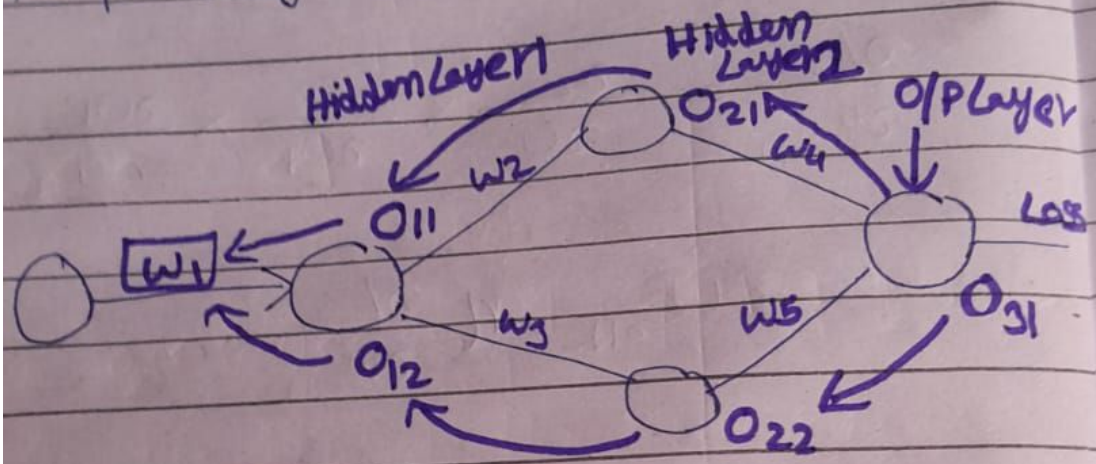
$$\frac{\partial L}{\partial w_{2old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_{2old}}$$

(Loss is dependent on O_2, O_2 is dependent on O_1 and O_1 is dependent on w_{2old})

$$w_{3new} = w_{3old} - \eta \boxed{\frac{\partial L}{\partial w_{3old}}}$$

$$\frac{\partial L}{\partial w_{3old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_{3old}}$$

Now let's take another example for practicing Chain Rule of Derivation

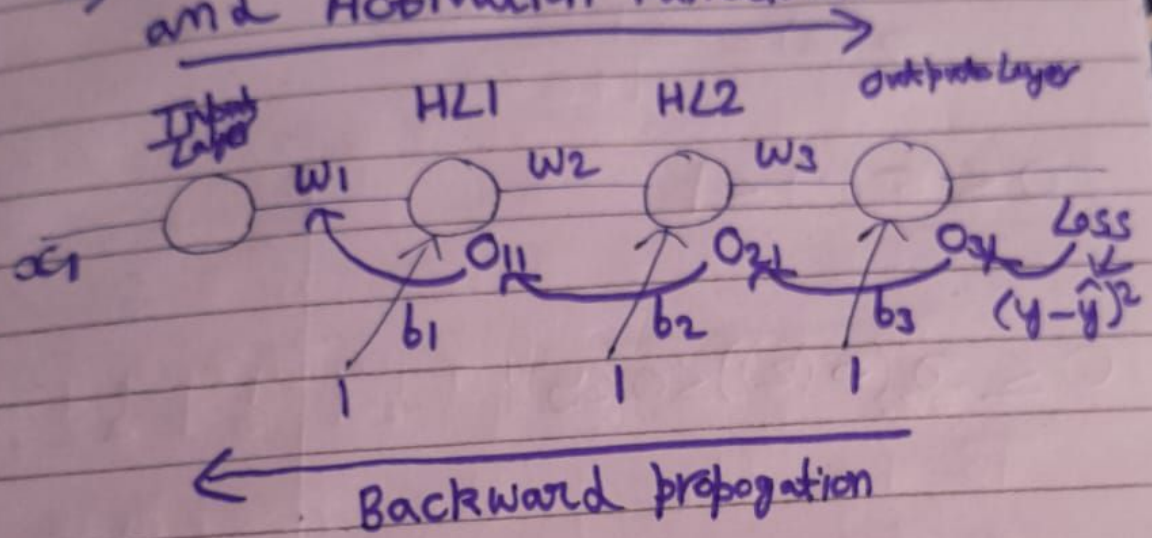


Now let we have to update W_1

$$W_{1new} = W_{1old} - \eta \left[\frac{\partial L}{\partial W_{1old}} \right]$$

$$\frac{\partial L}{\partial W_{1old}} = \left(\frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial W_{1old}} \right) + \left(\frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial W_{1old}} \right)$$

⇒ Vanishing Gradient Problem and Activation Functions:



We know

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

Then activation Function is applied Let's use Sigmoid Activation

$\sigma(Z) \Rightarrow 0$ to 1 (transform)

↳ Sigmoid

⇒ Activation Functions:

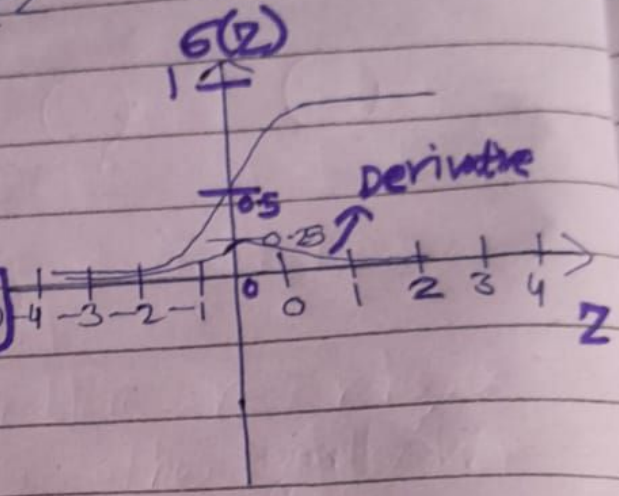
1: Sigmoid Activation Function:

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

$$0 < \sigma(z) < 1$$

Derivative ($\sigma(z)$)

$$0 < \frac{\partial}{\partial z}(\sigma(z)) < 0.25$$



$$w_{1_{new}} = w_{1_{old}} - \eta \left[\frac{\partial L}{\partial w_{1_{old}}}\right]$$

Expand (Chain Rule of Derivation)

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

Now let's try to solve this one $\frac{\partial o_{31}}{\partial o_{21}}$

$$O_{31} = \sigma(w_3 \times O_{21} + b_3)$$

Let $z = w_3 \times O_{21} + b_3$

So, $O_{31} = \sigma(z)$

Now

$$\frac{\partial O_{31}}{\partial O_{21}} = \frac{\partial(\sigma(z))}{\partial z} \times \frac{\partial z}{\partial O_{21}}$$ {Chain Rule}

We know

$$\boxed{0 \leq \frac{\partial}{\partial z}(\sigma(z)) \leq 0.25}$$

⇓ Derivative of Sigmoid

So,

$$\frac{\partial O_{31}}{\partial O_{21}} = 0 \leq \sigma(z) \leq 0.25 \times \frac{\partial(w_3 \times O_{21} + b_3)}{\partial(O_{21})}$$

$$\boxed{\frac{\partial O_{31}}{\partial O_{21}} = 0 \leq \sigma(z) \leq 0.25 \times w_{3_{old}}}$$

- So we are getting smaller values b/w 0 to 0.25 so multiplying it we also get small value in $\frac{\partial L}{\partial w_{1_{old}}}$

the change or update in w_1
ill be very small. So weights
re actually never get updated
convergence not happening
his is called vanishing gradient
roblem ($w_{1new} \approx w_{1old}$) and this
s happening due to Sigmoid activation.

To fix this problem, researchers
started exploring other activation
unctions like
tanh ② relu ③ trelu ④ swiss

f neural networks is very deep,
hen Sigmoid Activation cause
anishing Gradient Problem

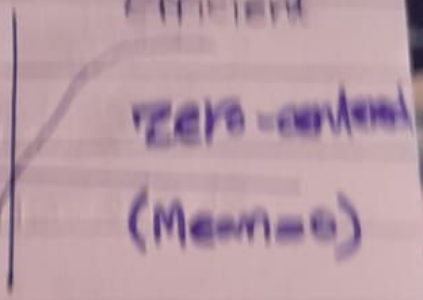
antages:

Binary Classification Suitable
Clear prediction i.e 0 or 1

Disadvantages:

① Cause Vanishing Gradient Problem
② Function output is not zero-centered ⇒ Weight Updation is not efficient

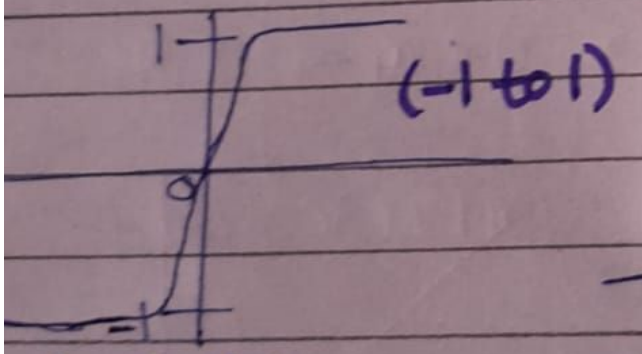
③ Mathematical operations are relatively time consuming (as e^x) power operations.



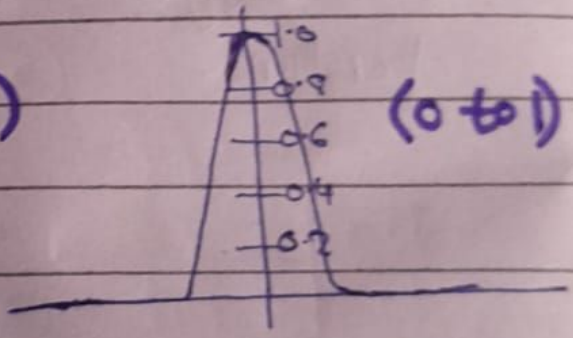
② tanh Activation Function:

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

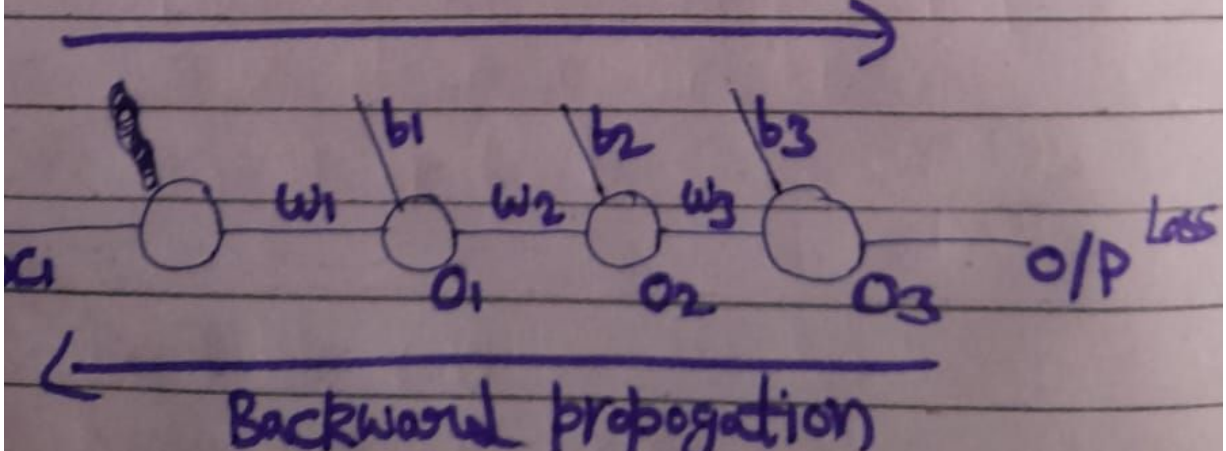
- It's output is b/w -1 to 1
- The derivative of tanh ranges between 0 to 1



tanh (x)



derivative(tanh(x))



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

$$\frac{\partial L}{\partial W_{2old}} = \frac{\partial L}{\partial O_3} \times \boxed{\frac{\partial O_3}{\partial O_2}} \times \frac{\partial O_2}{\partial W_{2old}}$$

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial (\tanh(Z))}{\partial Z} \times \frac{\partial Z}{\partial O_2}$$

$$Z = (O_2 \times W_3) + b_3$$

$$\frac{\partial O_3}{\partial O_2} = (0 \text{ to } 1)\left(\frac{\partial ((O_2 \times W_3) + b_3)}{\partial O_2}\right)$$

$$\frac{\partial O_3}{\partial O_2} = (0 \text{ to } 1)(W_{3old}) \Rightarrow$$ Still small value in case of Deep neural network and cause vanishing gradient problem

Advantages:

The output is zero-centric so efficient weight updation.

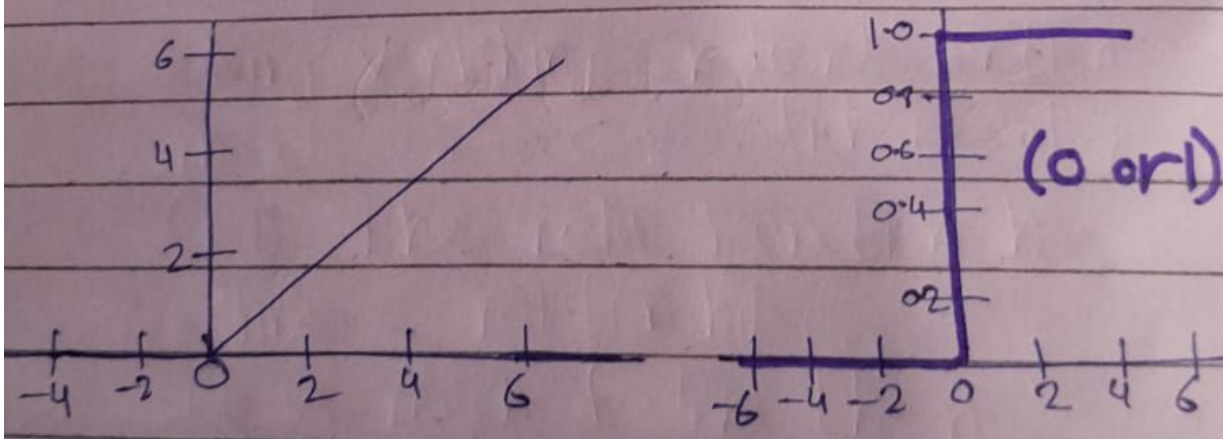
Disadvantages:

1. Prone to vanishing gradient problem
2. Still exponential mathematical operations are there which are time consuming.

③ Relu Activation Function:

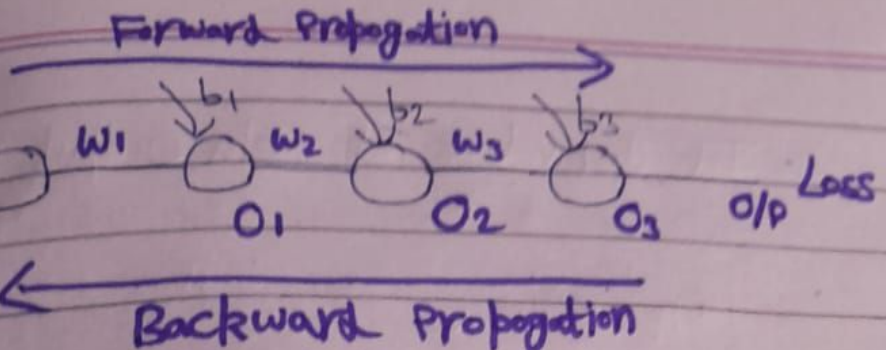
Rectified linear unit

$$ReLU(x) = max(0, x)$$



Relu(x) = max(0, x)

derivative relu(x)



$$W_{2_{new}} = W_{2_{old}} - \eta\left(\frac{\partial L}{\partial W_{2_{old}}}\right)$$

$$\frac{\partial L}{\partial W_{2_{old}}} = \frac{\partial L}{\partial O_3} \times \frac{\partial O_3}{\partial O_2} \times \frac{\partial O_2}{\partial W_{2_{old}}}$$

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial(\text{relu}(z))}{\partial z} \times \frac{\partial z}{\partial O_2}$$

$\therefore$ derivative $(\text{relu}(z)) \Rightarrow 0$ or 1

$\therefore z = (W_3 \times O_2) + b_3$

$$\frac{\partial O_3}{\partial O_2} = [0 \text{ or } 1] \times \frac{\partial (W_3 \times O_2 + b_3)}{\partial O_2}$$

$$\frac{\partial O_3}{\partial O_2} = [\underset{-ve}{0} \text{ or } \underset{+ve}{1}] \times W_3$$

$\Rightarrow$ If Derivative of Relu output is **1**

$\Rightarrow$ Weight Updation will happen

⇒ If derivative of Relu output is 0

⇒ Dead Neuron (no weight updation)

$w_{new} \approx w_{old}$ ⇒ Dead Neuron

If $z = +ve$ then $\frac{\partial(relu(z))}{\partial z} = 1$

if $z = -ve$ the $\frac{\partial(relu(z))}{\partial z} = 0$

Advantages:

1. Solving Vanishing Gradient Problem
2. $max(0, \alpha)$ → Calculation is super fast. The relu Function has a linear relationship.
3. It is much faster than sigmoid or tanh.

Disadvantages:

1. Dead Neuron
2. Not zero-centric

④ Leaky Relu and Parameteric Relu:

$f(x) = \max(\alpha x, x)$ → Hyperparameter $\alpha = 0.01, 0.02, 0.03$ etc.

$f(x) = \max(0.01x, x)$

- So the derivative of relu leaky will never be zero something greater than 0 even in case of -ve 2 so no **Dead Neurons.**

Advantages:

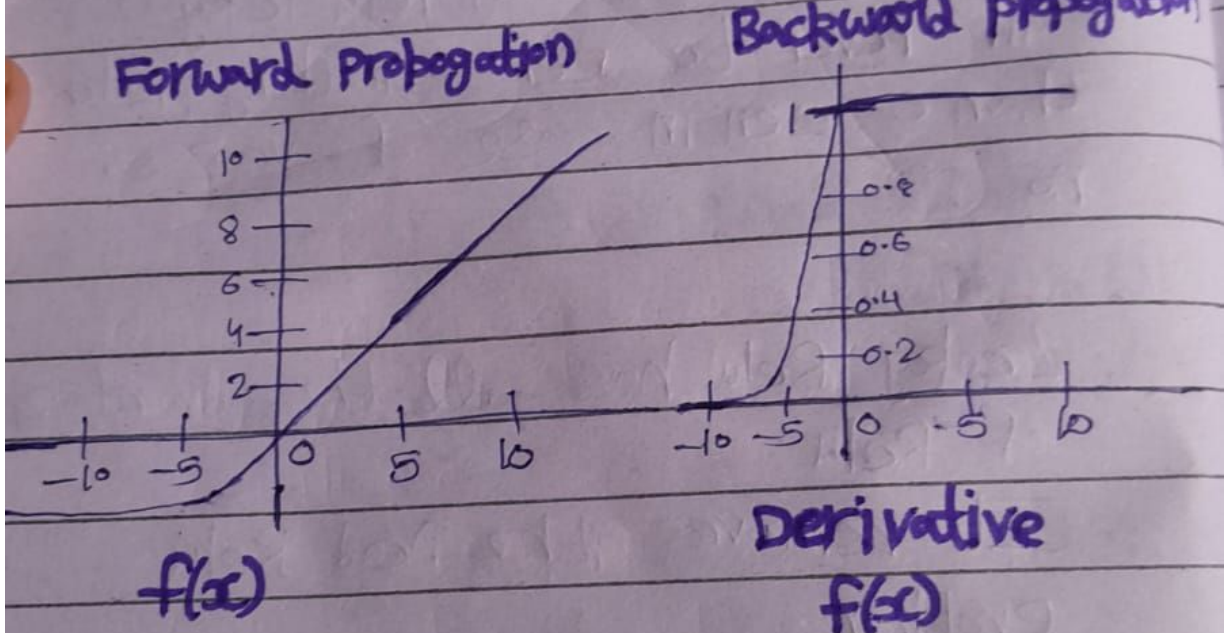
1. Leaky Relu has all the advantage of Relu
2. It removes the Dead Relu problem

Disadvantage:

1. It is not zero-centric

⑤ ELU (Exponential Linear Unit):

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



f(x)

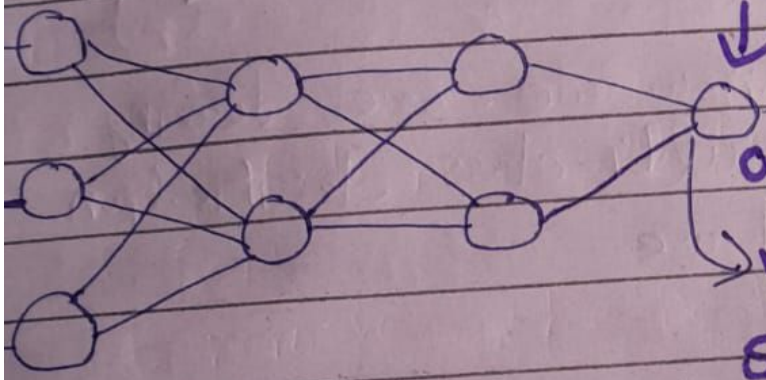
Derivative f(x)

- **Advantages:**
  1. No Dead Relu issues
  2. It is zero centered

- **Disadvantages:**
  1. Slightly more computationally intensive because of exponential operation

) Softmax Activation Function:

- It is specifically used for Multi-class classification problem



Binary classification (one neuron)

Loss

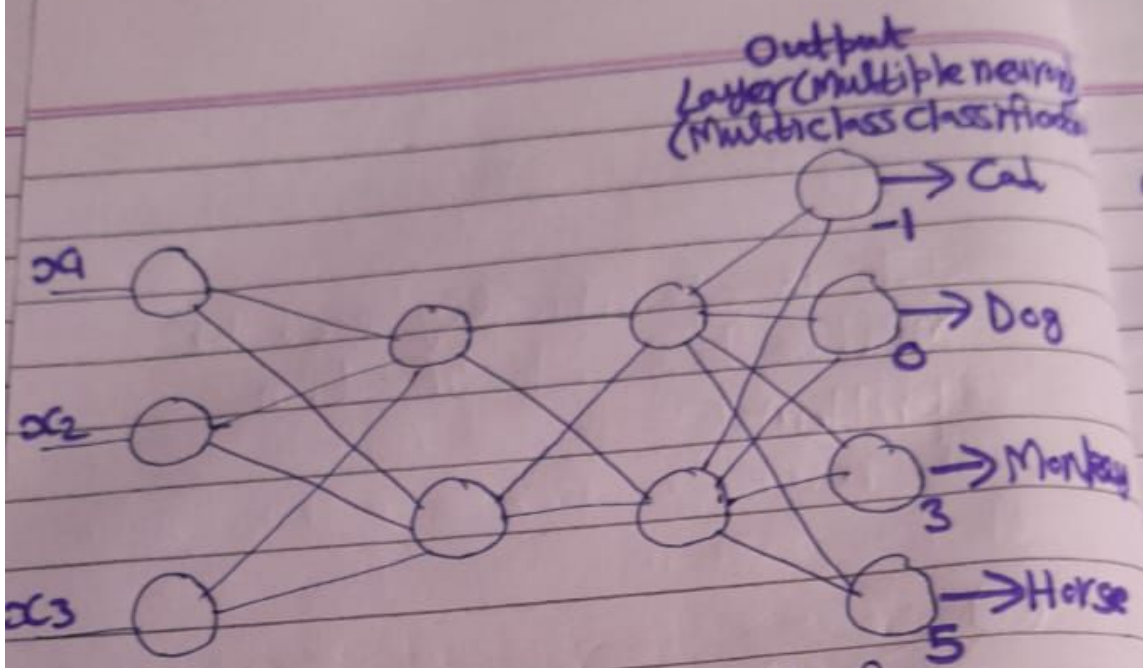
O/P layer

We apply Sigmoid activation ranges from 0 to 1

(if ≥0.5 then 1

<0.5 then 0)

This is how output decide in binary classification



- So as now we are dealing with multi-class classification we will use **Softmax activation function.**
- Let we have got outputs without applying activation as -1, 0, 3 and 5 respectively

$$\text{Softmax} = \frac{e^{y_i}}{\sum_{K=0}^{n} e^{y_K}}$$

ftmax ⇒ act = $\frac{e^{-1}}{(e^{-1}+e^{0}+e^{3}+e^{5})}$ = 0.00053

∴ y is output without apply activation

Dog ⇒ $\frac{e^{0}}{e^{-1+0+3+5}}$ = 0.0024

Monkey ⇒ $\frac{e^{3}}{e^{-1+0+3+5}}$ = 0.0183

Horse ⇒ $\frac{e^{5}}{e^{-1+0+3+5}}$ = 0.1353

$Pr(Horse) = \frac{0.1353}{0.00033+0.0024+0.0183}$

Pr(Horse) = 86%

⑦ Which Activation Function to use when?

In Output Layer:

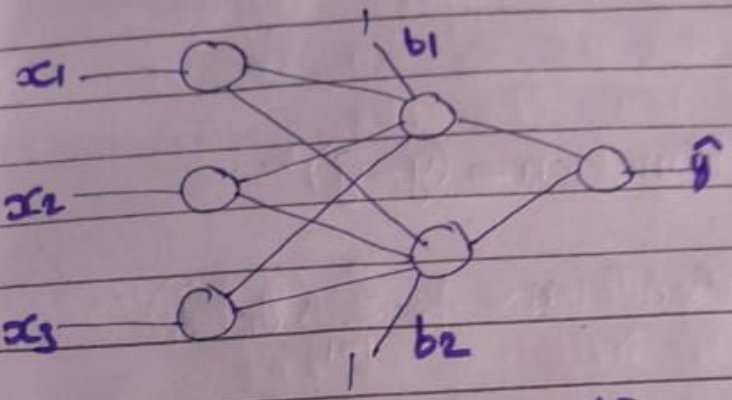
Binary Classification → Sigmoid

Multi-class Classification → Softmax

In Hidden Layers:

Use Relu or it's variants like Pre-Relu, Leaky-Relu or Elu

Loss Function and Cost Function:



x_1	x_2	x_3	O/P
—	—	—	0
—	—	—	1
—	—	—	0
—	—	—	1

Loss Function

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

- In loss function we pass one datapoint from dataset at a time.

Cost Function

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

- In cost function we pass all datapoints of dataset all at once.

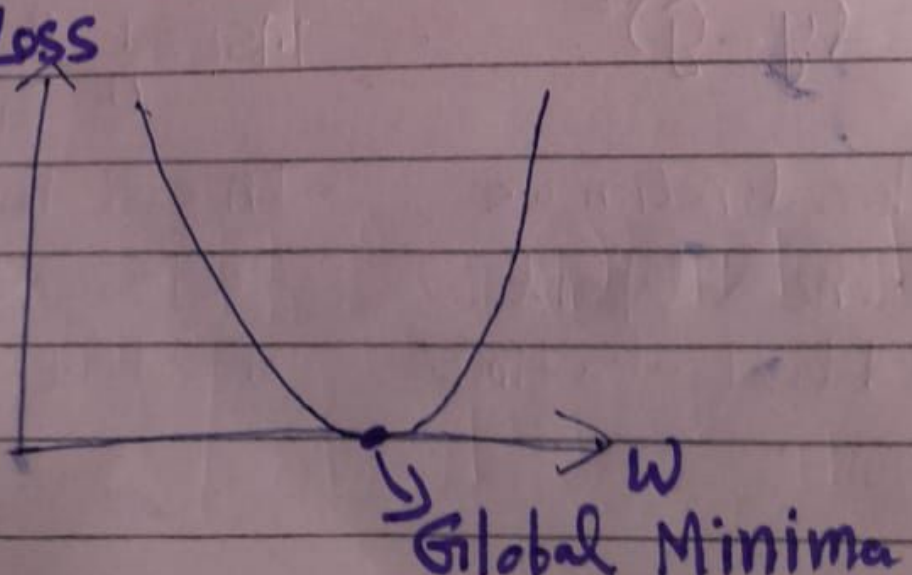
⇒ Regression Loss and Cost Functions:

① Mean Squared Error (MSE):

$$\text{Loss Function} = (y - \hat{y})^2$$

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

As the equation of Loss and Cost is quadratic so it's graph with weight gives us gradient descent (parabola) which has one global minima and we have to reach that.



⇒ **Advantages:**

① MSE is differentiable.
② It has one local or global minima so we don't get stuck.
③ It converges faster.

⇒ **Disadvantages:**

① MSE is not robust to outlier as we square up the error and we are penalizing the cost function so line is changing much just due to one outlier.

Mean Absolute Error (MAE):

$$\text{Loss Function} = |y - \hat{y}|$$

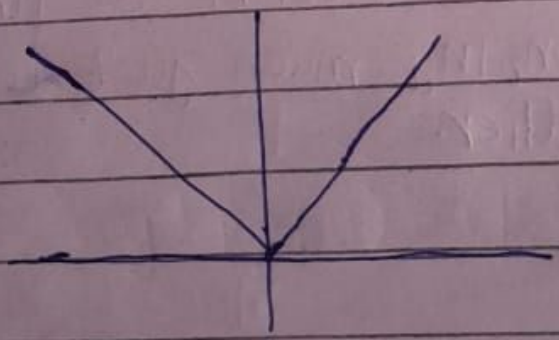
$$\text{Cost Function} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

⇒ **Advantages:**

① Robust to outliers because we are not squaring or penalizing the error.

⇒ **Disadvantages:**

① Convergence usually takes time in MAE, as we don't get parabola curve and we have to use sub gradient concept.



③ Huber loss:

$$\text{Cost Function} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, & \text{if } |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

δ is a specific threshold treated as Hyperparameter.

- Huber Loss is actually combination of both MSE and MAE if there is no outlier than it use MSE equation otherwise adjusted MAE.

④ Root Mean Squared Error (RMSE):

$$\text{Cost Function} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Advantages:

- Same unit
- Differentiable

Disadvantages:

- Not robust to outliers

⇒ Loss and Cost Functions for Classification Problems.

① Binary Cross Entropy:

// Log Loss

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

where

$y \Rightarrow$ Actual value, $\hat{y} \Rightarrow$ Predicted value

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

- It is used for Binary classification

② Categorical CRoss Entropy.

- It is used for Multi-class classification problem

f_1	f_2	f_3	Output
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

⇓ Step1: ~~Perform One Hot Encoding of output variable~~

$j=1$	$j=2$	$j=3$
Good	Bad	Neutral
1	0	0
0	1	0
0	0	1

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

where $i = 1$ to n (For multiple records

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

And $\hat{y}_{ij}$ is calculated by getting output from neuron and apply softmax function.

⇓

gives us probabilities

- So Categorical CROSS Entropy also gives the probability of other categories

③ Sparse Categorical CROSS Entropy:

- Very similar to Categorical CROSS Entropy

Let we put some index with respect to categories and mention their probabilities

0 1st 2nd → categories

$[0.2, 0.3, 0.5]$

⇓

2nd index ⇒ output

Disadvantage:

- It don't give probability of other categories

⇒ Which Loss Function to use when?

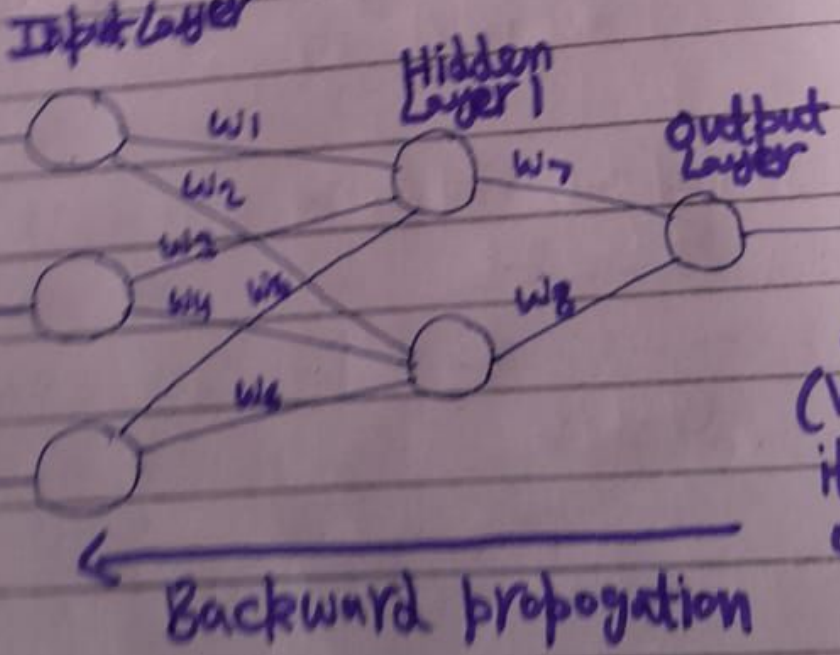
* Right Combination

	Hidden Layers	O/P Layer	Problem	Loss Function
①	Relu or variants	Sigmoid	Binary classification	Binary cross Entropy
②	Relu or variants	Softmax	Multi-class	Categorical or Sparse
③	Relu or variants	Linear	Regression	MSE, MAE, Huber loss, RMS

Optimizers:

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini Batch SGD
4. SGD with Momentum
5. Adagrad and RMSPRoP
6. Adam optimizers

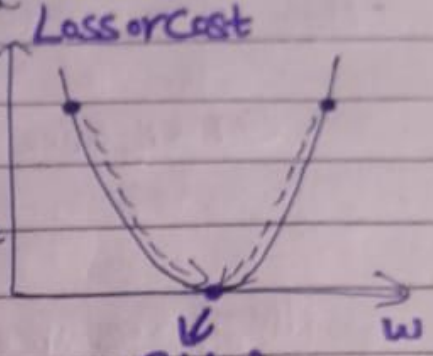
1. Gradient Descent optimizer:



We know weight updation formula generally

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

(η → Learning rate)



MSE

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

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

Epochs, Iteration

1 epoch { → 1000 datapoint → $\hat{y}_i$ ⇒ Cost Function

← Weights will get updated by weight updation formula

Our aim is to update weights in backward propogation to reach global minima (At global minima slope = 0 so $W_{new} \approx W_{old}$ no weight updation. This is the point where loss is minimum)

- Mean in gradient descent optimizer we pass all datapoints at once not seperately and use cost function
- We can perform any number of epochs until we reach global minima

- In Gradient Descent optimizers
  1 epoch = 1 iteration
  (as we pass all datapoints at once)

- **Advantages:**

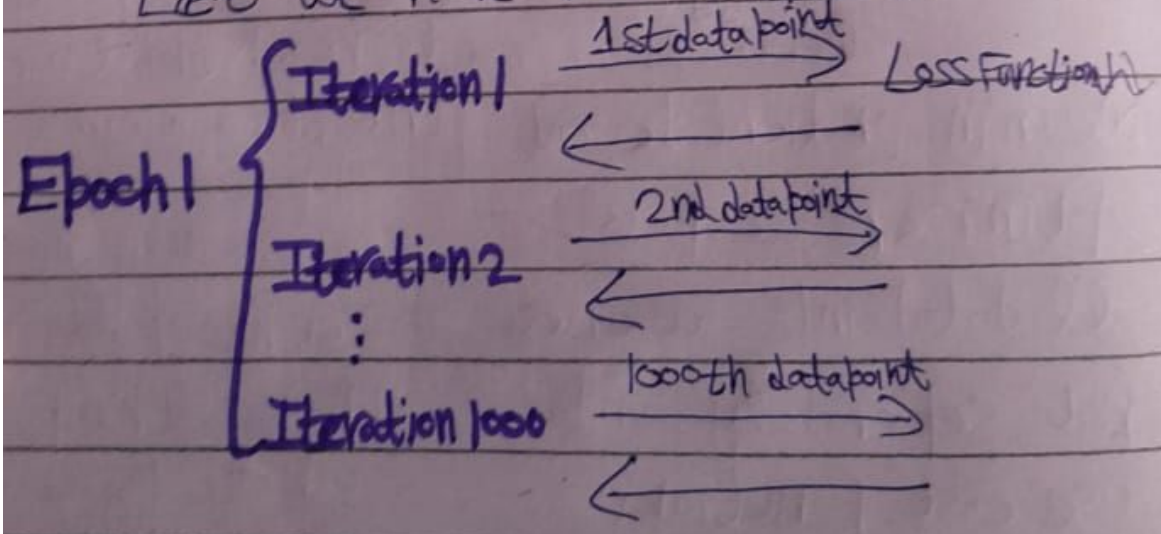
  ① Convergence will happen

- **Disadvantage:**

  ① Huge Resource Required i.e RAM & GPU
  as we are passing all datapoints
  So, Resource Intensive

② **Stochastic Gradient Descent (SGD):**

Let we have 1000 datapoints



In SGD we pass one datapoint at a time and that is called Iteration and the epoch has as many number of iteration as many datapoints we have

Advantage:

Less Space Time

Disadvantage:

Time Complexity is huge

Convergence will also take more time

Noise gets introduced (as we are taking single datapoint)

③ Mini Batch SGD:

⇒ In Mini-Batch SGD Batch size is used means like SGD instead of passing one datapoint in each iteration we pass datapoints equal to batch size

Let

Datapoints = 100000

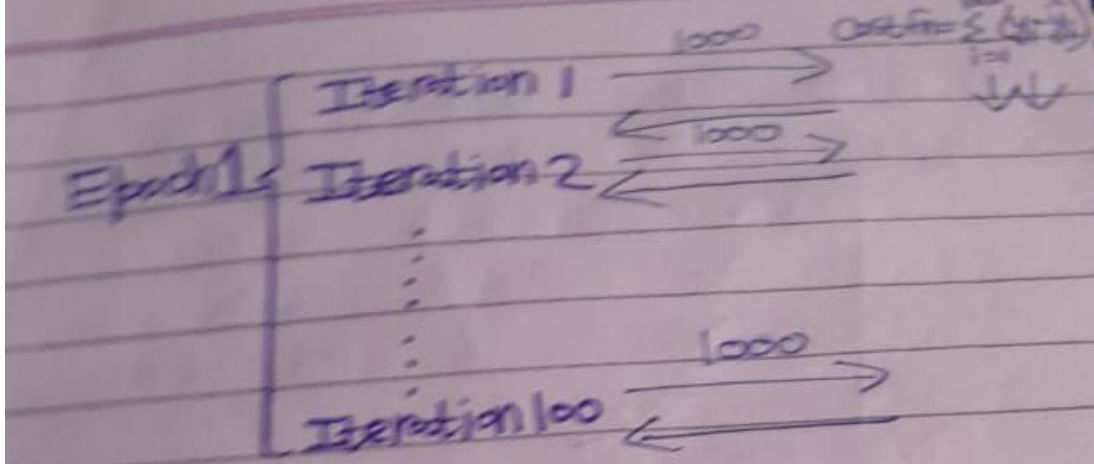
batchsize = 1000

No. of iterations = $\frac{\text{Datapoints}}{\text{batchsize}} = \frac{100000}{1000}$

No. of iterations = 100

⇓

Means there will be 100 iteration per epoch and we pass 1000 datapoints in one iteration.



⇒ In Mini-Batch SGD the noise gets reduce but still exist

Advantages:

① Convergence speed will increase. ↑ compared to SGD.

② Noise will be less when compared to SGD.

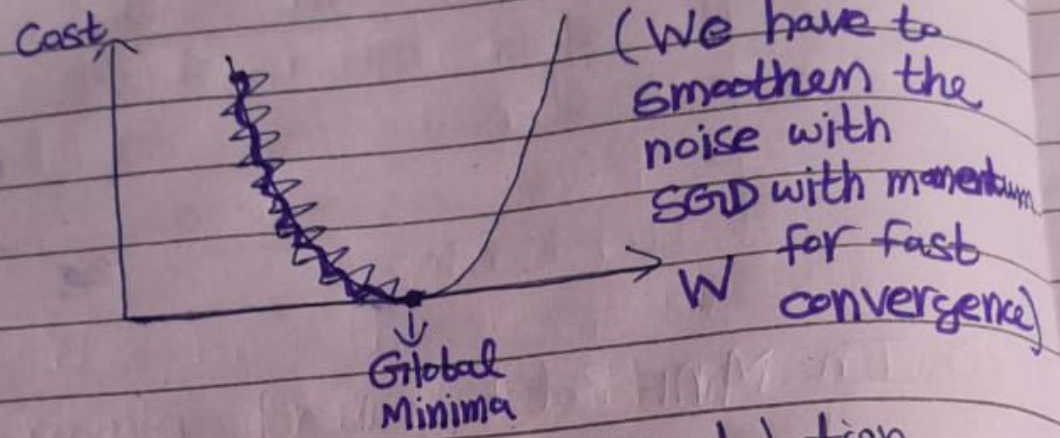
③ Efficient Resource Usage

Disadvantage:

① Noise still exist

,,,

④ SGD with Momentum:



We know the weight updation Formula and bias updation formula

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

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

Now the updated weight updation Formula for SGD with momentum is:

$$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W_{t-1}}$$

⇓

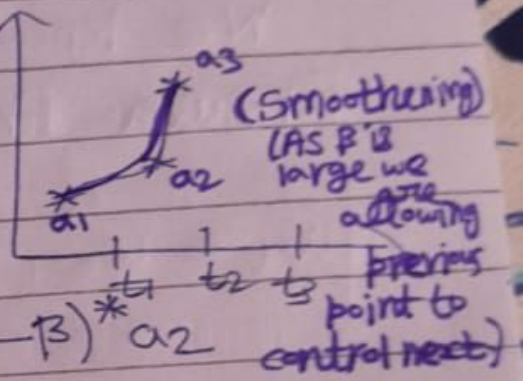
Weight Updation Formula

Exponential Weighted Average {Smoothening} ⇒ ARIMA, SARIMA

Time = $t_1\ t_2\ t_3\ t_4 \cdots t_n$

Values = $a_1\ a_2\ a_3\ a_4 \cdots a_n$

Time series



(Smoothening) (As β is large we are allowing previous point to control more)

Now

$V_{t_1} = a_1$ (Value at time t_1)

$\beta = 0.95$

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

β = 0 to 1 (parameter)

Let $\beta = 0.95$

$V_{t_2} = (0.95) * a_1 + (0.05) a_2$

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

$V_{t_3} = 0.95[0.95 * a_1 + 0.05 * a_2] + (0.05) * a_3$

Advantage:

① Reduces the noise

② Quick convergence

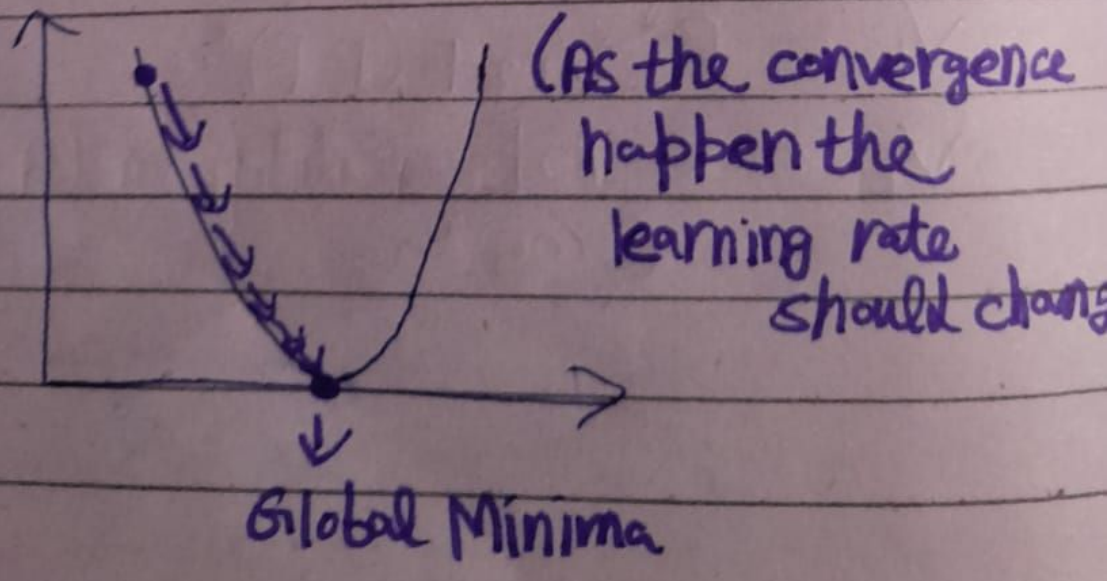
⑤ Adagrad (Adaptive Gradient Descent):

We know the new weight updation formula from SGD with momentum

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

(η → Learning rate)

- In previous optimizer η (learning rate) was fixed. Now we will make this η (learning rate) dynamic



Now our new weight updation formula for Adagrad is:

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

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$ → Epsilon (small value) (so that denominator don't become 0 even when α_t is 0)

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

As α_t keeps on increasing the learning rate keeps on decreasing means learning rate(η) will be large initially then reduce as we get closer to convergence

Disadvantage:

① $\eta' \rightarrow$ possibility to become a very small value ≈ 0 (in very deep neural network having large α_t)

⑥ Adadelta and RMSPROP:

We know the weight updation formula from Adagrad

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

But now,

$$\eta' = \frac{\eta}{\sqrt{S_{dw_t} + \varepsilon}}$$

Exponential weighted average (Smoothening)

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

We initialize $S_{dw_t} = 0$

⑦ Adam optimizer:

⇒ Adam optimizer is the best optimizer

⇒ It combines:

SGD with momentum + RMSPROP

$w_t = w_{t-1} - \eta' V_{dw_t}$ ⇒ Weight Updation

$b_t = t_{t-1} - \eta' V_{d_{b_t}}$ ⇒ Bias Updation

We know

$$\eta' = \frac{\eta}{(S_{dw_t} + \varepsilon)}$$

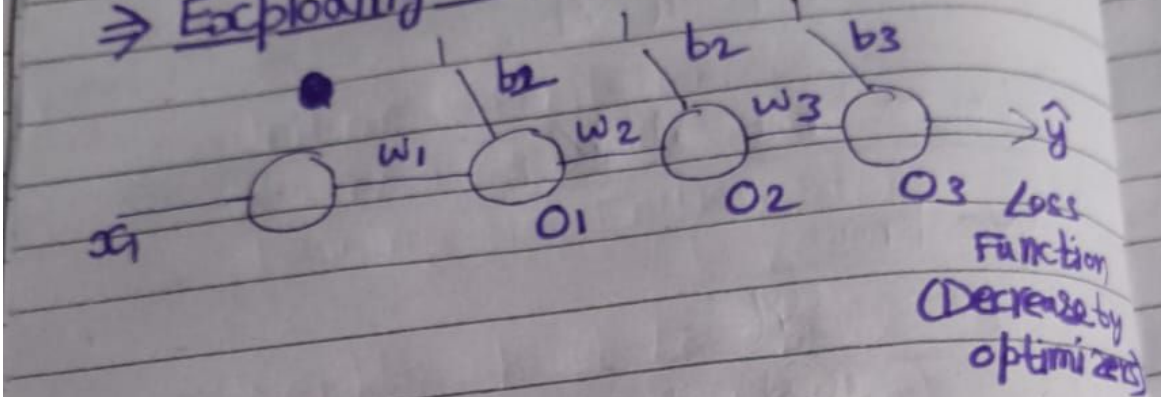
$$S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

Now

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

$$V_{d_{b_t}} = \beta * V_{d_{b_{t-1}}} + (1-\beta)\frac{\partial L}{\partial b_{t-1}}$$

⇒ Exploding Gradient Problem:



We know

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

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_3} \times \frac{\partial O_3}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_1}$$

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial (\sigma(z))}{\partial z} \times \frac{\partial z}{\partial O_2}$$

We know derivative of sigmoid is from 0 to 0.25 and $z = O_2 \times w_3 + b_3$

$$\frac{\partial O_3}{\partial O_2} = [0 \text{ to } 0.25] * \frac{\partial (O_2 \times w_3 + b_3)}{\partial O_2}$$

$$\boxed{\frac{\partial O_3}{\partial O_2} = [0 \text{ to } 0.25] * w_3}$$

Now if the weight initialization is with very large number like

$w_{3012} = 500$ to 1000

So when we multiply big value we also get very big value and after multiplying with learning rate we get very big value now

$w_{i\,new} >>> w_{i\,old}$ or $w_{i\,new} <<< w_{i\,old}$

⇓

($w_{i\,new}$ is very bigger than $w_{i\,old}$)

($w_{i\,new}$ is very smaller than $w_{i\,old}$)

⇒ As weight updation is with very large value so convergence become difficult and this is called exploding gradient problem which happen due to wrong weight initiali

(ii) Xavier Uniform Initialization:

$W_{ij} \approx$ Uniform Distribution $\left(\frac{-\sqrt{6}}{\sqrt{\text{input}}}, \frac{\sqrt{6}}{\sqrt{\text{input}}}\right)$

$W_{ij} \approx$ Uniform Distribution $\left(\frac{-\sqrt{6}}{\sqrt{6}}, \frac{\sqrt{6}}{\sqrt{6}}\right)$

(3) Kaiming He Initialization:

(i) He Normal

$W_{ij} \approx$ Normal Distribution $(0, \sigma)$

$\sigma = \sqrt{\frac{2}{\text{input}}}$

In our case

$\sigma = \sqrt{\frac{2}{3}}$

$W_{ij} \approx$ Normal Distribution $\left(0, \sqrt{\frac{2}{3}}\right)$

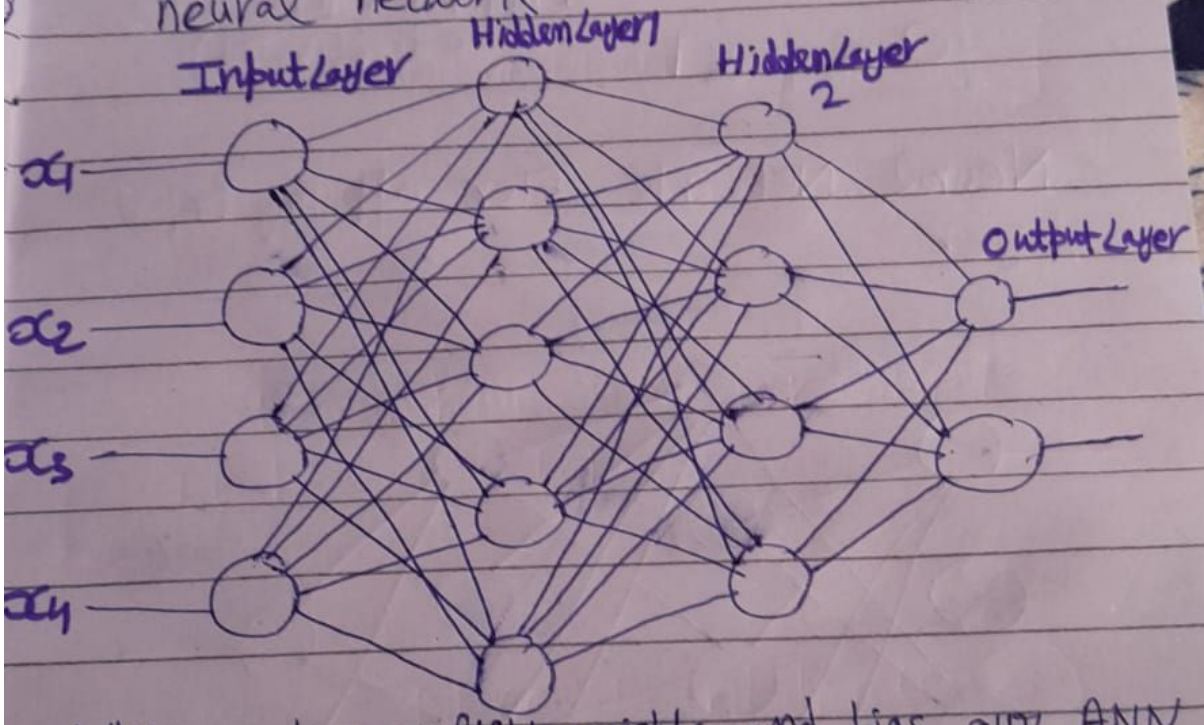
(ii) He Uniform:

$W_{ij} \approx$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{\text{input}}}, \frac{\sqrt{6}}{\sqrt{\text{input}}}\right.$

$W_{ij} \approx$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{3}}, \frac{\sqrt{6}}{\sqrt{3}}\right.$

⇒ Dropout Layer:

Let we have a multi-layered neural network



When we have multiple weights and bias our ANN can get overfitted

Overfitting:

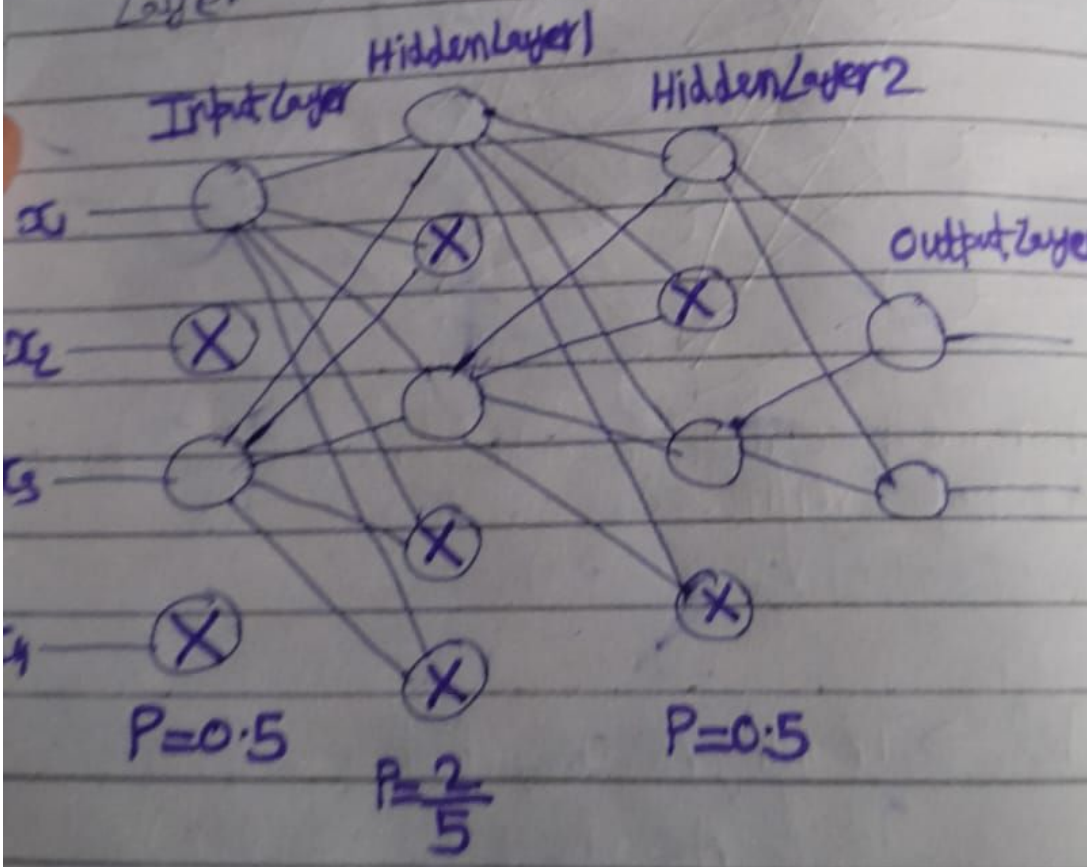
Model → Training Accuracy (high e.g 90%)
Test Accuracy (low e.g 60%)

⇒ But we have to get generalized model.

- To reduce overfitting in Neural network we use **Dropout Layer**.
  Let $p = 0.5$ (Means 50% of nodes from input will pass other will be deactivated)

$$0 \leq p \leq 1$$

Neural Network after applying Dropout Layer

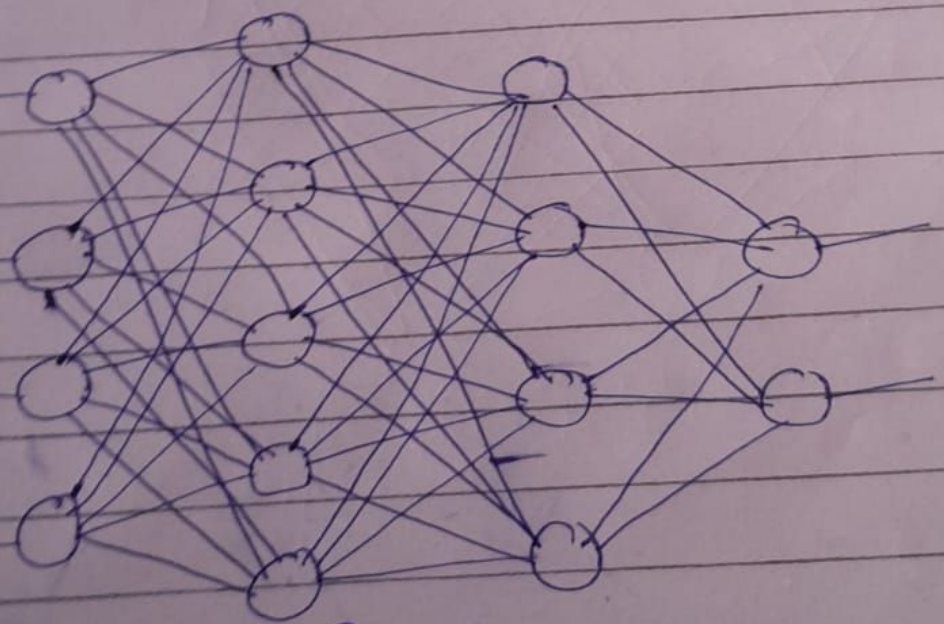


- In 1st epoch depending on P some neurons gets activated and others will be deactivated and they are

selected randomly and will not be same.

- The purpose is that we are performing dropout (deactivating some neurons) to avoid overfitting.
- P is the Hyperparameter $0 \leq P \leq 1$

- Now for test data



$P=0.5$ $P=\frac{2}{5}$

- For test data we will not apply hidden layer all neurons will be activated. Here we multiply the weights with the probability of hidden layer.