Deep Learning with Deep NLP

Artificial Intelligence

Machine Learning

Deep Learning

Today we are die into the basics of deep learning

Q- what is deep learning? why is so much hype about deep learning. what is the history of deep learning. How did it ivolue and why company spend so much money on ~~deep~~ deep learning.

Deep learning → which is a subset of machine learning which itself is a part of much greater Artificial intelligence.

Deep learnings has algorithms that are inspired by the structure and the function of the brain.

~~ia~~ ,

Deep Learning with Deep NLP

Artificial Intelligence

Machine Learning

Deep Learning

Today we are dic into the basics of deep learning

Q- what is deep learning? why is so much hype about deep learning. what is the history of deep learning. How did it evolve and why company spend so much money on ~~dep~~ deep learning.

Deep learning -> which is a subset of machine learning which itself is a part of much greater Artificial intelligence.

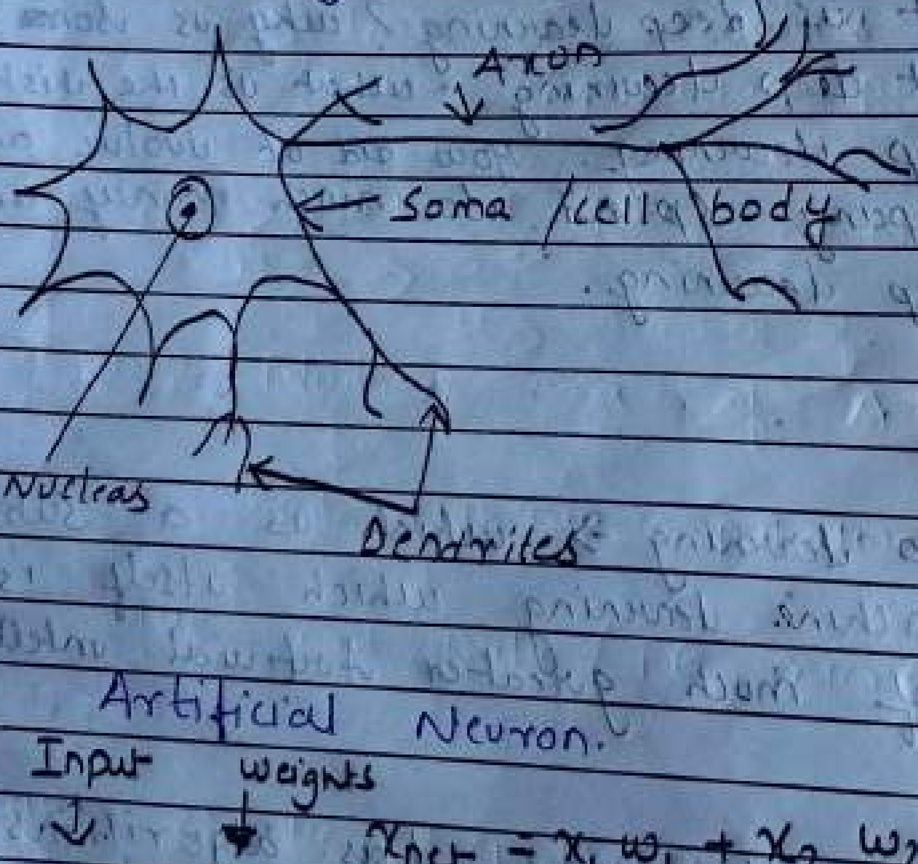
Deep learnings has algorithms that are inspired by the structure and the function of the brain.

History of Deep learnings

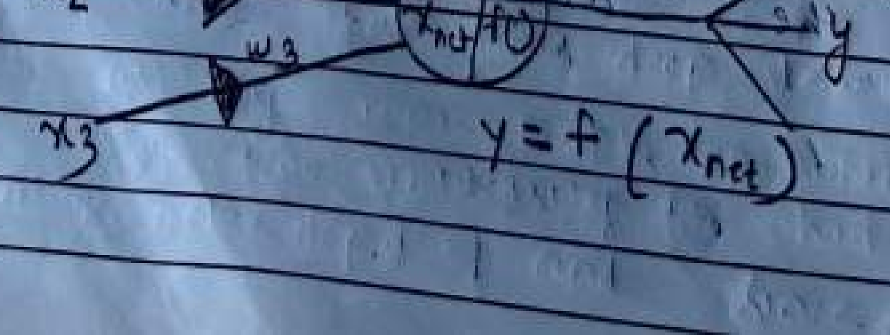
→ its started in 1958 where we get first idea of perceptron, so perceptron is inspired by the biological neuron.

neuron is a basic building block of the brain.

Biological Neuron



Artificial Neuron.



$$x_{net} = x_1 w_1 + x_2 w_2 + x_3 w_3 = \sum_{i=1}^{n} x_i w_i$$

$$y = f(x_{net})$$

$$x_{net} = \cancel{w_0}\, x_1 w_1 + x_2 w_2 + x_3 w_3 = \sum_{i=1}^{n} x_i w_i$$

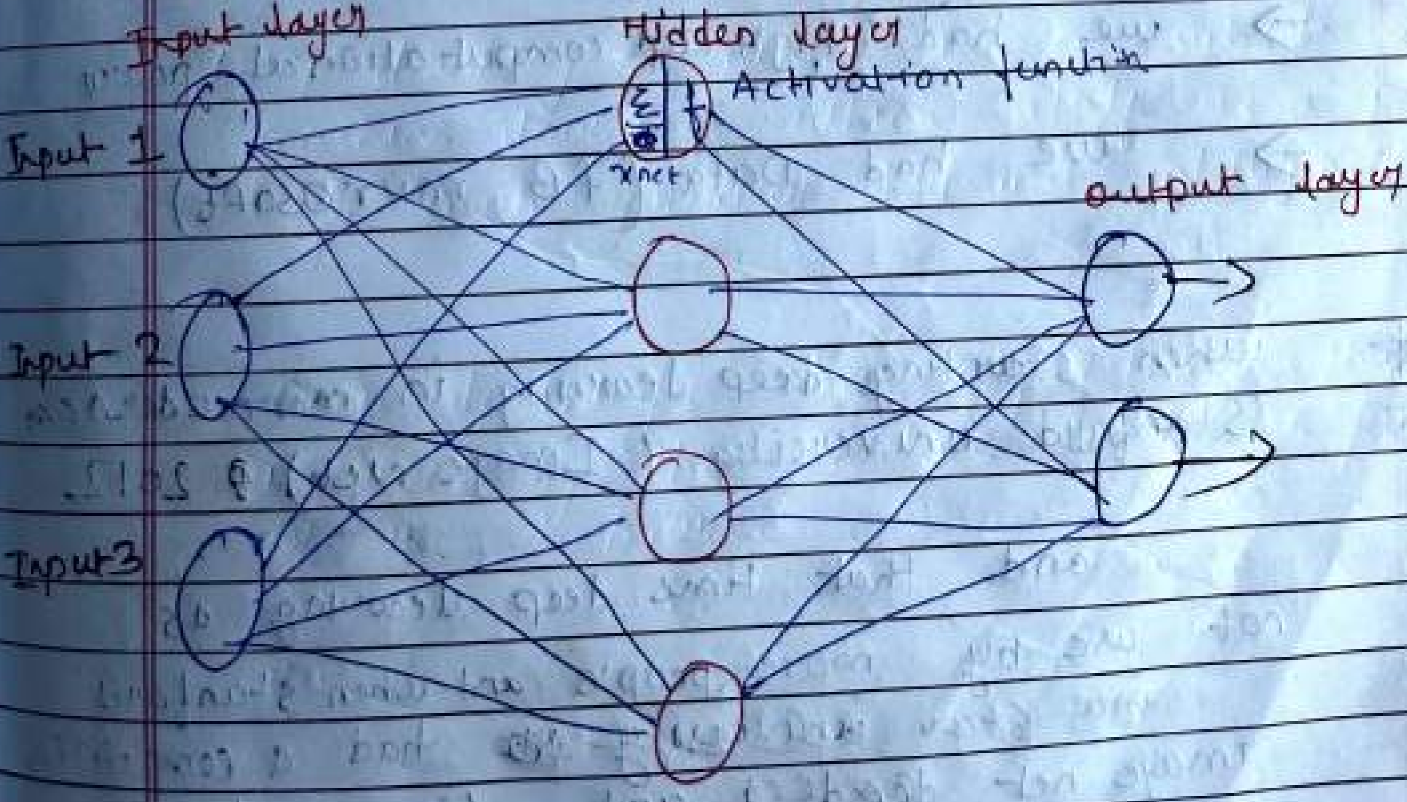
$f(x)$ -> passing the function. and this function is known as Activation function

* problems

-> A single perceptrons/node

-> No learning

2-> MLP (Multi layer perceptron) 1980



4

Hidden layer -> That means all the layer that is between the input layer and the output layer are known as hidden layer.

these layers are neither directly ~~connecte~~ connected to the input or are they directly connected to the output.

problem -> this only provided theory.
2 -> lack of resources.
3 -> less computational power
4 -> lack of data

-> 2012 Modern era of AI.

-> we had high computational power

-> we had Data (FB, microsoft)

* when starting deep learning in modern era
* Stanford university (Image Net) 2012

and that time deep learning is not use by more people and when stanford ~~stan~~ ~~university~~ had a competition Image net ~~dee~~ deep and since ~~dee~~ deep learning is not popular people used to stick with ~~classic~~ machine learning algo like Random forest, SVM but a ~~team~~ team use basic deep learning algo to ~~the~~ solve this image net problem.

image net -> To identify objects from images (very difficult).

Deep learning beats all the ML algo.

* Now Days

-> Now we have high resources

-> lots of data

-> Because of globalization we keep getting new algorithms to solve new problems.

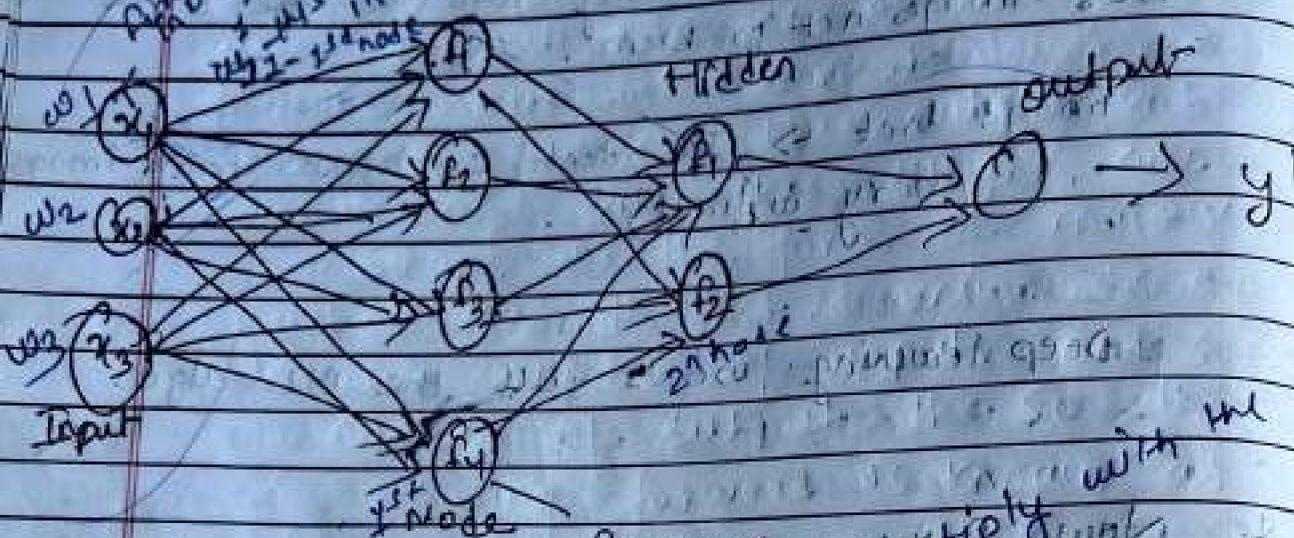
CV -> CNNs
80s / 90s

-> history of deep learning and we read 2 different concepts perceptrons and multilayer perceptron or (MLP). So let us quickly recap. we ~~had~~ we had inputs going in we cell body here, we performed first the summision ~~fo~~ this is followed by an activation

Optimisers & Gradient Descent

Basic structure of neuron and MLP.

First about sum of 1 it is going into node.



→ So, first inputs you will be multiply with the weights. Perceptron

$w^1_{34} \to$ 3 → stands of Number of inputs

4 → Number of nodes

$$z^1_1 = w^1_{11} x_1 + w^1_{21} x_2 + w^1_{31} x_3 + \text{bias}$$

Activation function

$$h^1_1 = f(z^1_1)$$

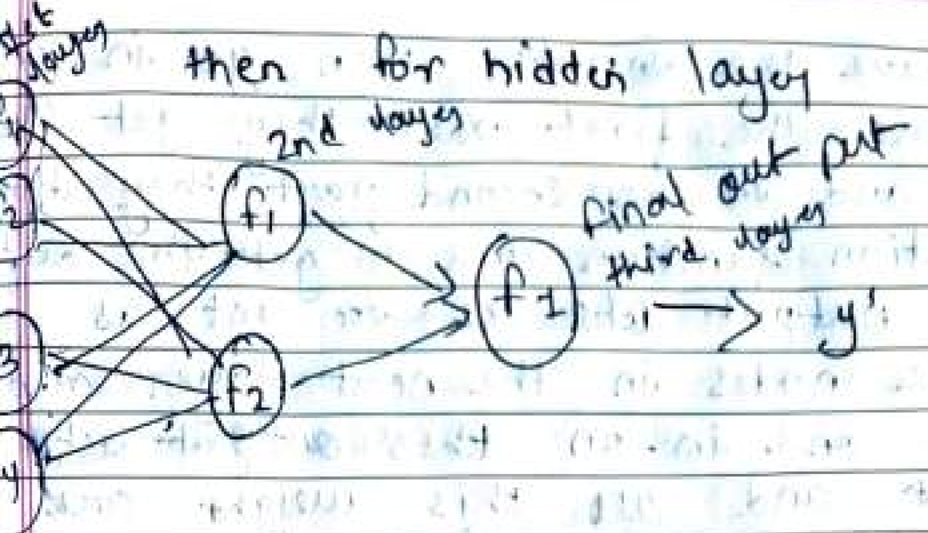
$$h^1_2 = f(z^1_2)$$

then we get $h^1_1, h^1_2, h^1_3, h^1_4$

7

w = weight
h = hidden

then for hidden layer



$$z_1^2 = w_{11}^2 h_1^1 + w_{21}^2 h_2^1 + w_{31}^2 h_3^1 + w_{41}^2 h_4^1 + \text{bias}$$

$$\rightarrow h_1^2 = f(z_1^2)$$

This whole process going foreword, you have the inputs go and make a go foreword first to the input layer then hidden layer then on and on they will reach the output layer that is predicted by the model (y')

That moving forword feed (very importan

~~this whole process is going foreword you have~~

→ so, since in one way it is moving foreword is called (forward feed) we we have the inputs, then go in one by one. the first part they get summed up and in the second part they have activation function and to get the outputs this output ~~act~~ act as input for the nodes in the next layer and so on in on ~~this output act as~~ ~~input~~ and all this weight are shape in the form of matrix. where, the ~~t~~ first dimension is equal to the number of inputs and second number of output. ~~the~~ So that moving forword is known as forword feed.

So, ~~we~~ the next part we learning is move backword

* optimisers - Gradient Descent

So, in the last class we learned about foreword feed

foreward feed → you have the input and each input get multiply with weights to add weightage to the input and get passed through a multilayer perceptron to given a output. in a multilayer perceptron we have nodes so the input get multiply with the weights and each node they get all summed up together and pass through an function that is an activation function. This gives us ~~ad~~ an output which acts as a input for the nodes in some other layer and this goes on an on untill you get the predicted value

Since all the weight and biases are randomly initialize there is no way that the predicted value is going to be close to the actual value

-> we can ~~only~~ ~~chang~~ have inputs and we cant change the input we can only change the weight and bias.

-> weight, bias need to be change so that predicted value very close the actual value

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

-> weight and biases are known as trainable or learnable parameters.

* LOSS -> Loss basically means how ~~difference~~ differ much ~~value~~ does predicted value differe from the actual value.

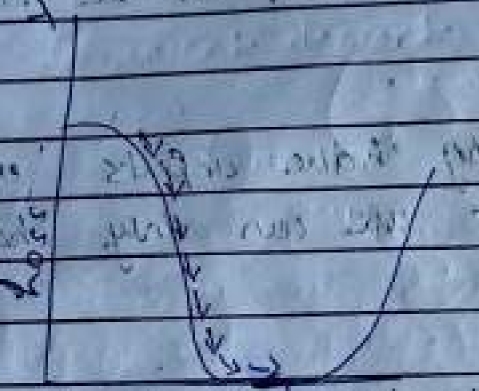
we heard about ① ~~L1~~. L1 loss, L2 loss.

Q-> we Need to change weights and biases so that we have very low loss.

~~So~~ for that we use optimiser gradient.

we have to optimise the weights and biases so that we have a very low loss minimal loss.

Q-> Lets talk about the basic optimiser Gradient Descent.



so, to reach the minimum loss you have to slowly descent downwards.

$$w_{new} = w_{old} - \frac{dL}{dw_{old}}$$

$$\frac{dy}{dx}$$

$y = loss$

$x =$ old weight

you have a chance of going 1 step at a time you have ~~to~~ chance to go .5 at a time so, this magnitude is known as the learning rate. this is denoted - η - eta.

η - eta

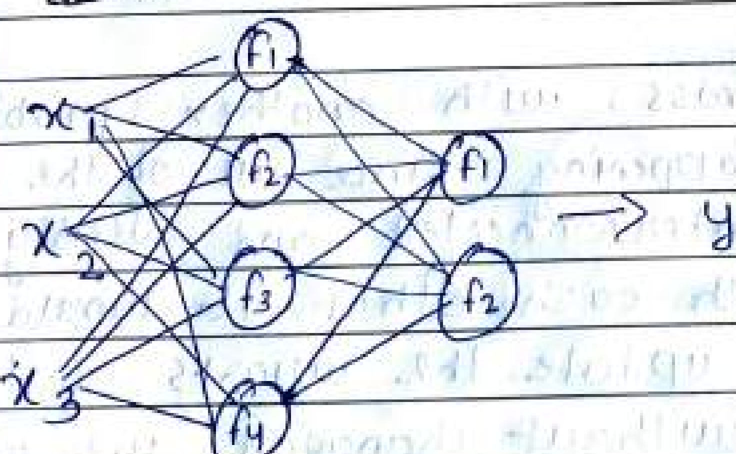
$$w_{new} = w_{old} - \underset{\text{Eta}}{\eta} \frac{dL}{dw_{old}}$$

function

we can update the loss and differentiate the loss with respect to the weights in 1st node. but we need some how ~~we~~ have find the way to do that. then we came across and that is known as

* Back propogation. this is a lot like chain rule.

* ~~Process~~



in the last few ~~videos~~ claues we have learned about forward ~~feed~~ propogation so you have the inputs they get multiply with the weights the ~~ad~~ get all summed together which is followed by in Activation function and this on an on until you get a predicted value. No this value can not be equal to the actual value initially. So that it's why you have the idea of loss which shows the how much different the predicted value is the actual value. Now you have the loss calculated so you to some how chang the weight and biases to minimize the loss. to do this we use optimisers. to optimise the weight and function we talked about one optimiser Gradient descent.

formula

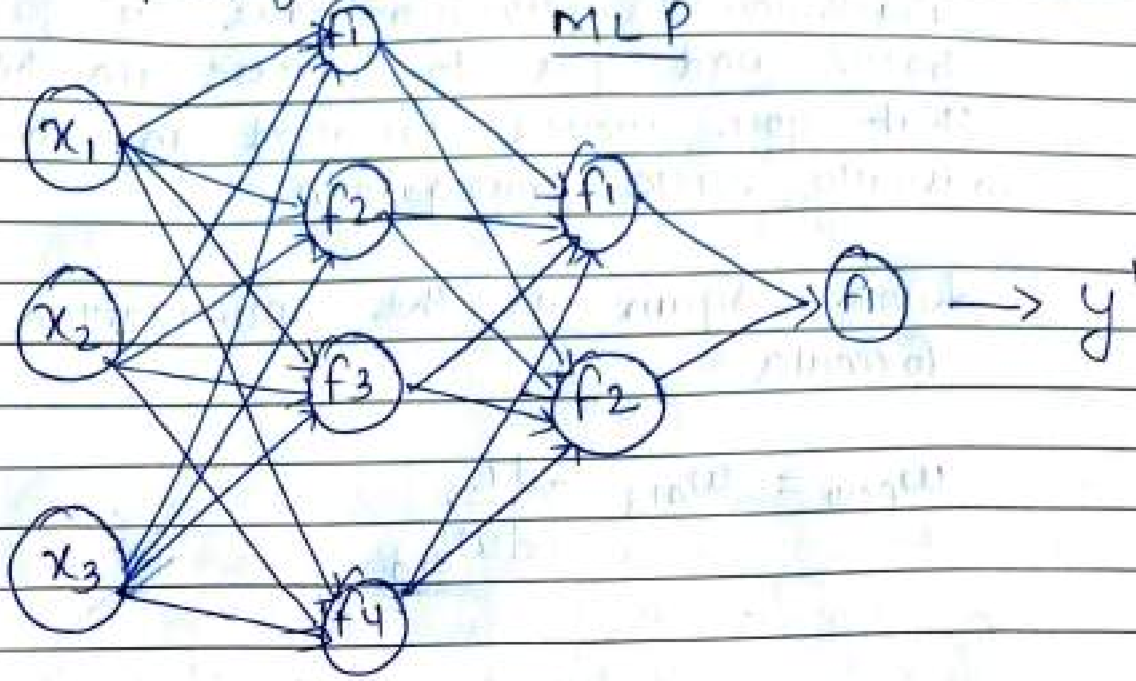
$$w_{new} = w_{old} - \frac{dL}{w_{old}}$$

we came across with another problem that was happening (since all of the nodes are interconnected and densly connected with each other it would be impossible to update the weights of the inner layer without changing) the weights of the outer layer. So this was a problem hence we came across what was known as the Back propogation

Here we follow chain rule of differentiation

Back propogation

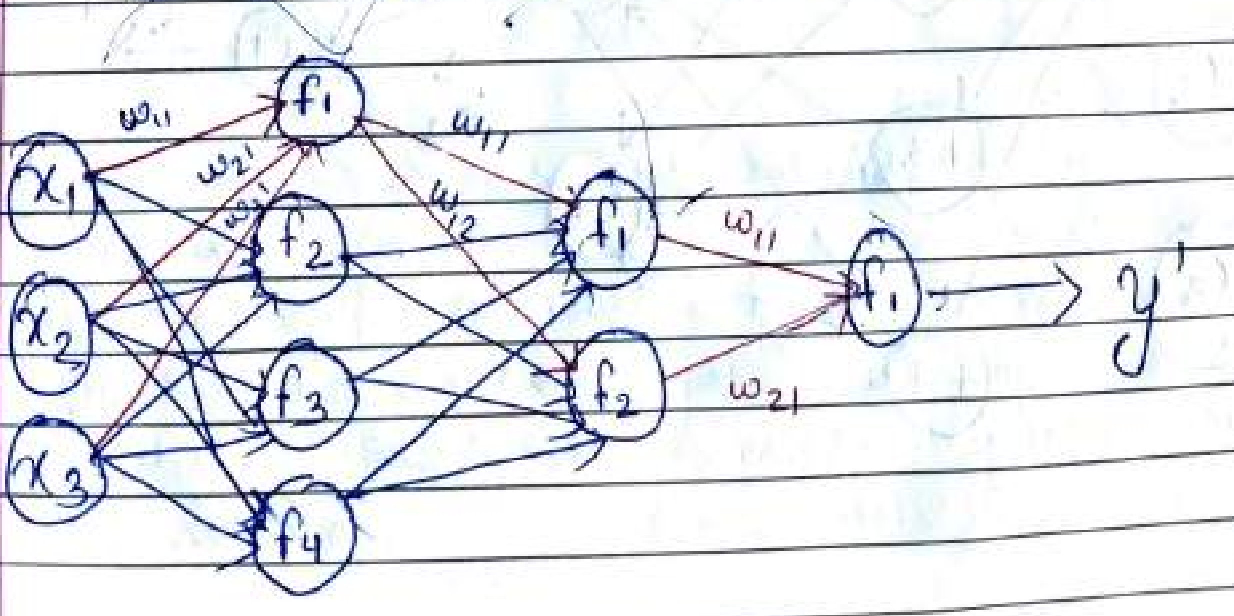
MLP



So, we had this output

$$w_{iJ}^{K}$$

K -> represent which layer
i -> from which node
J -> into which node



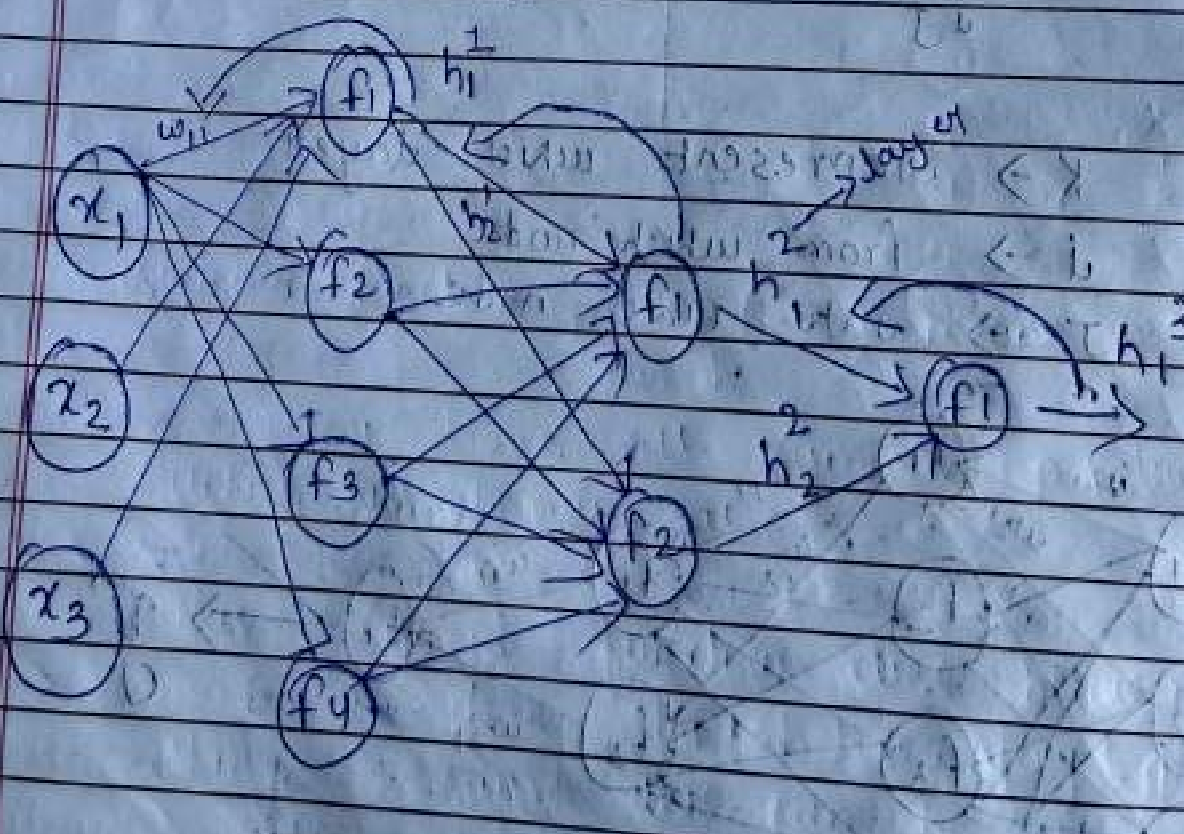
So as the name suggest, we have to go backwords you ~~have~~ have to go back back and you to traverse in the path that you moved foreword in that is basically back propogation.

Look Again in the optimization formula

$$w_{new} = w_{old} - \frac{dL}{dw_{old}}$$

$$w_{new} = w_{old} - \frac{dL}{dh_1^3} \times \frac{dh_1^3}{dw_{old}}$$

$$w_{new} = w_{old} \rightarrow w_{11}^3$$



$$\frac{dL}{dw_{11}^1} = \frac{dL}{dh_1^3} \times \frac{dh_1^3}{dh_1^2} \times \frac{dh_1^2}{dh_1^1} \times \frac{dh_1^1}{dw_{11\,old}^1} +$$

$$\frac{dL}{dh_1^3} \times \frac{dh_1^3}{dh_2^2} \quad \frac{dh_2^2}{dh_2^1} \times \frac{dh_2^1}{dw_{11}^1}$$

So, in short we have foreward feed where you go foreward in the layers one by one and you have backward propagation where you go back find the differentiation one by one and update the weights and bias. So this whole (forward feed + Backward propagation) together for 1 epoch.

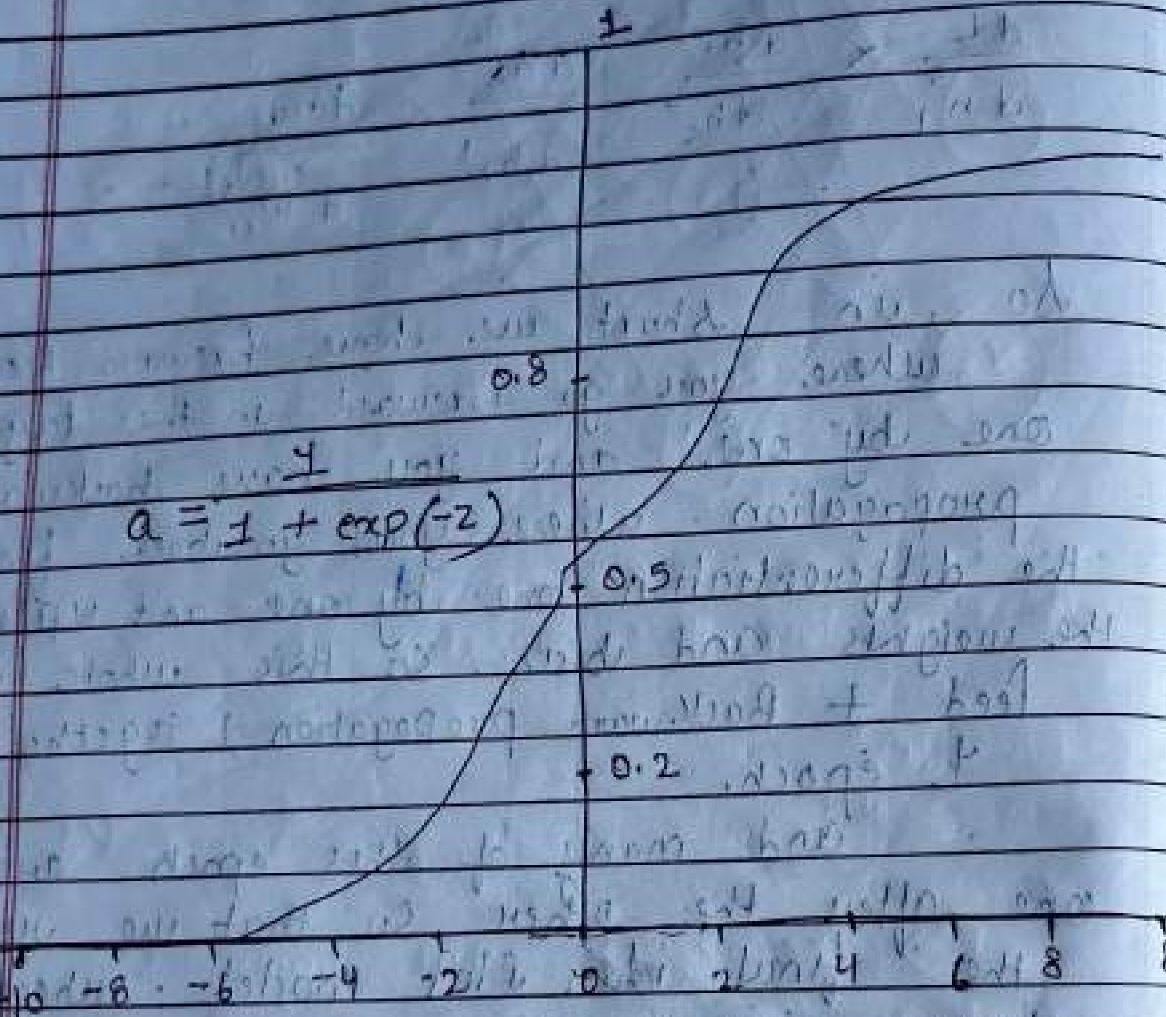
and many of this epoch are done one after the other. So that we reach the final idol model. where you have minimum loss.

* Activation function in depth

Activations function → you had this neuron here, you had all inputs going in you first had a summission where the input multiplied with different weights and after that this output here which was known as Z was passed through a function that is an activation function to give the final output

(1) First Activation function

(1) Sigmoid function

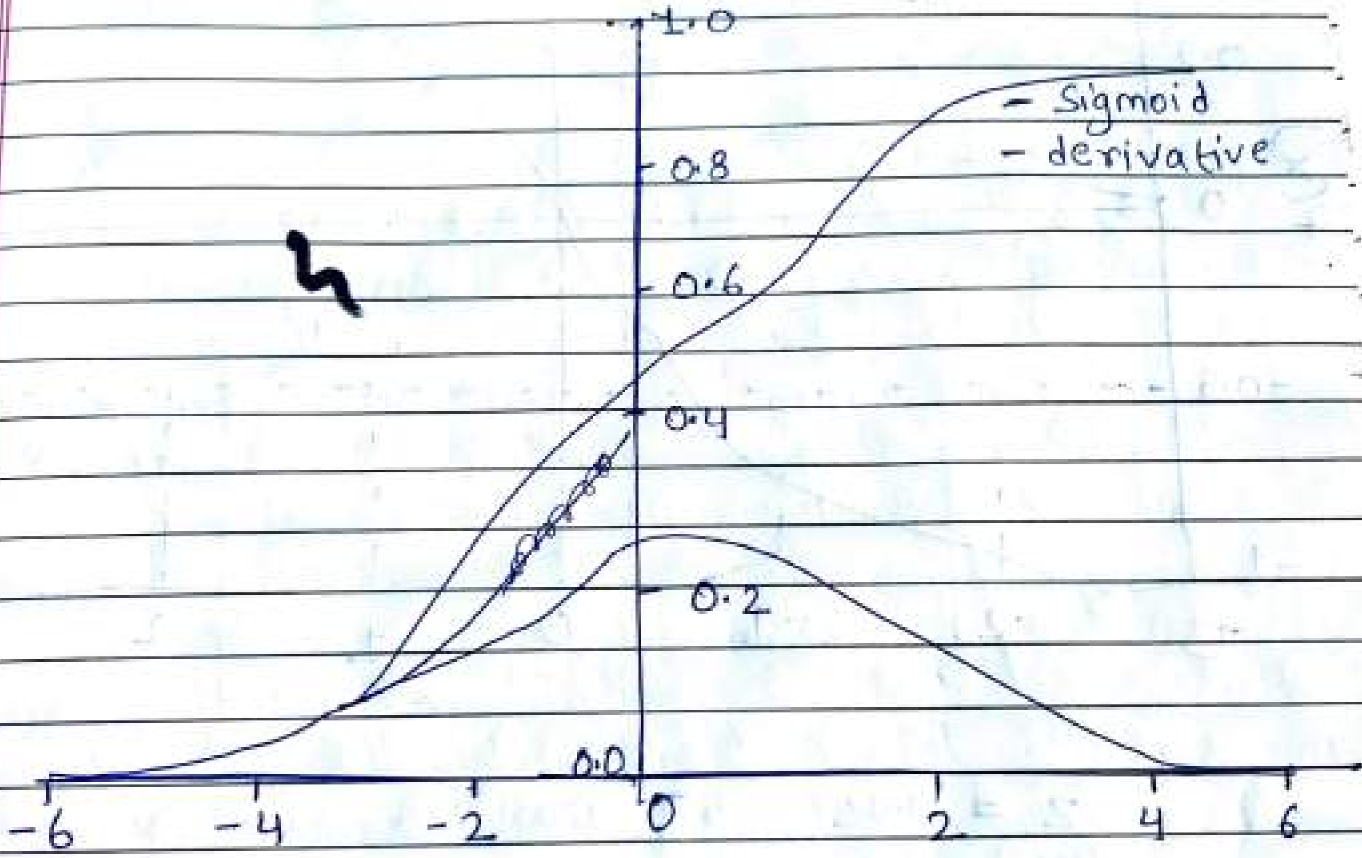


Sigmoid function, $exp(-z)$ means e^{-z}

z = you have all summed up value which is known as z.

this sigmoid function lies between 0 to 1, It is a very good thing because this can be ~~gs~~ we us classifiers then to give a ~~Semened values~~. Some values and this values we can take as prediction of or the probalities of how this happening or not happening

* Derivative of sigmoid Function

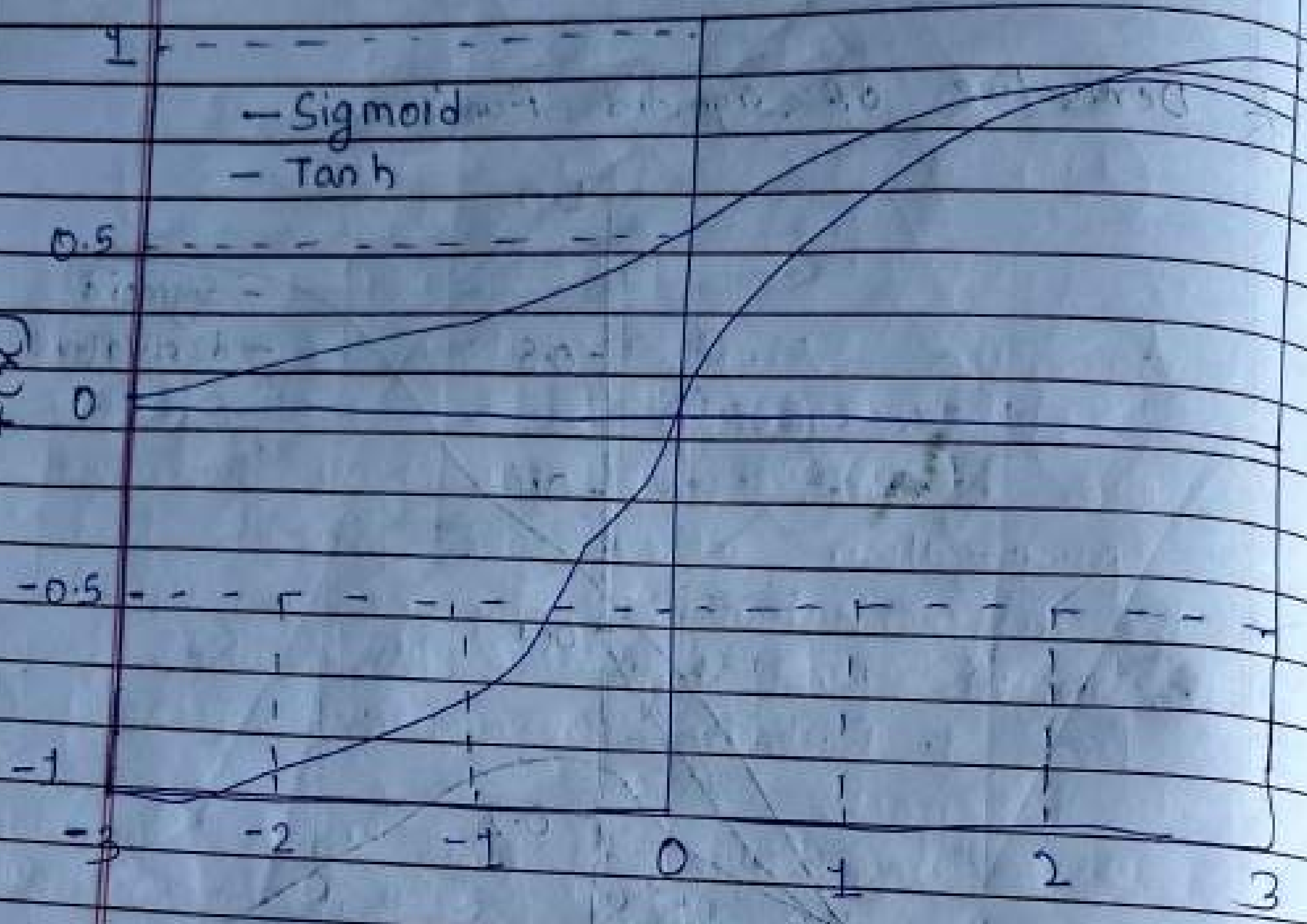


* why we do need Derivative
of we had otimiser and in optimiser and in optimiser we use to calculate the loss with respect to the weights

$$\frac{dL}{dw} \rightarrow \frac{dL}{dh} \times \frac{dh}{dw}$$

→ Next function

Tan h



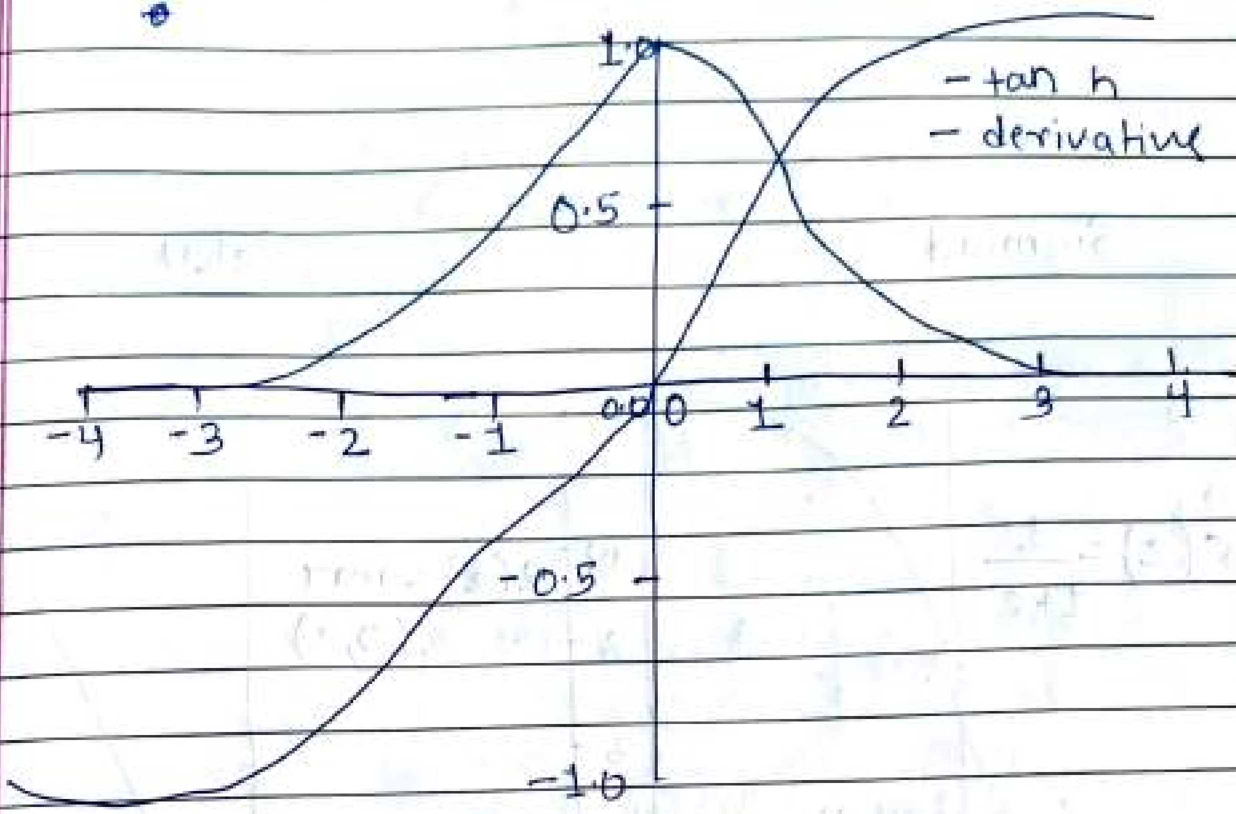
z = -ve

~~Tanh~~ [-Derivative Tan h]

equation of Tanh → $\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

→ Tan h function lies between [-1, 1]

- Tan h Derivative



derivative - $1-a^2$

where $a = \frac{e^2 - e^{-2}}{e^2 + e^{-2}}$

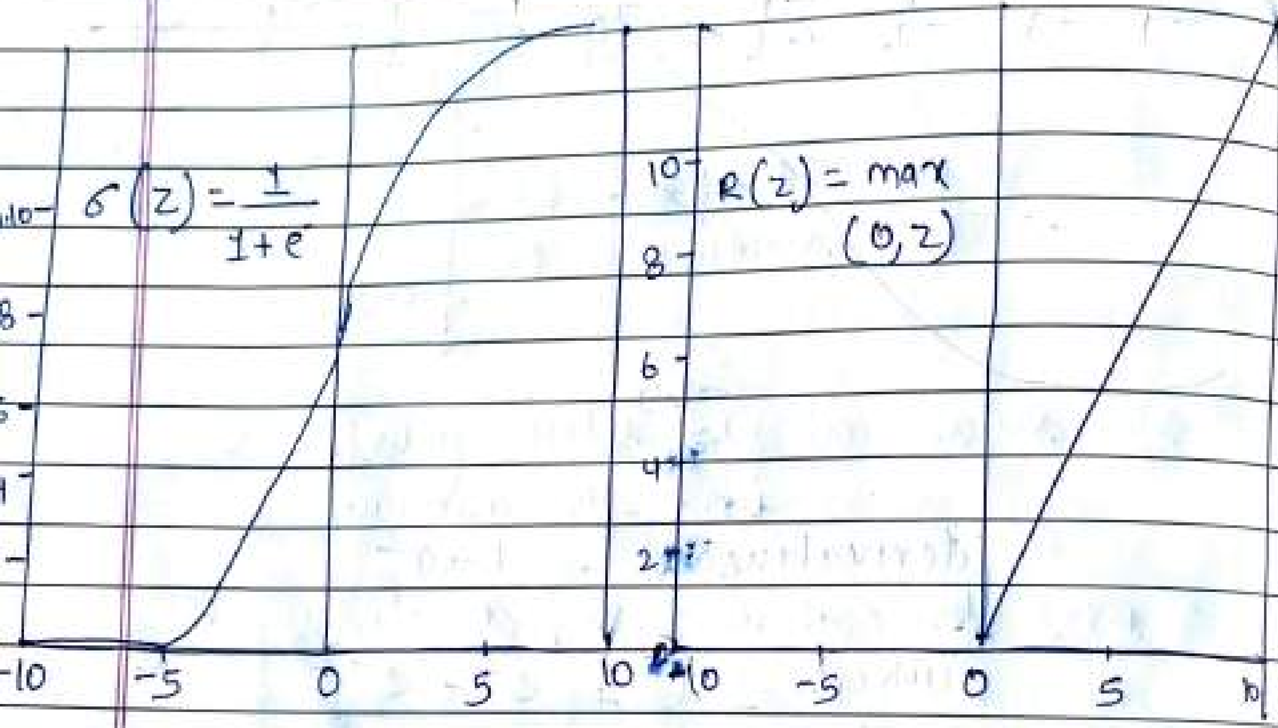
ReLU

Date
Page

* Rectified Linear Units (ReLU)

Sigmoid

ReLU



- ~~dera~~ for any positive number derivative will be one

- for any negative numbers derivative will be zero

* problems with Relu
-> Dying Rely
-> Dying neurons

$\frac{dL}{dw} = 0$

→ Suppose for any case you have Z is less then zero, so in that case equally forms a zero it comes on equate the value of zero.

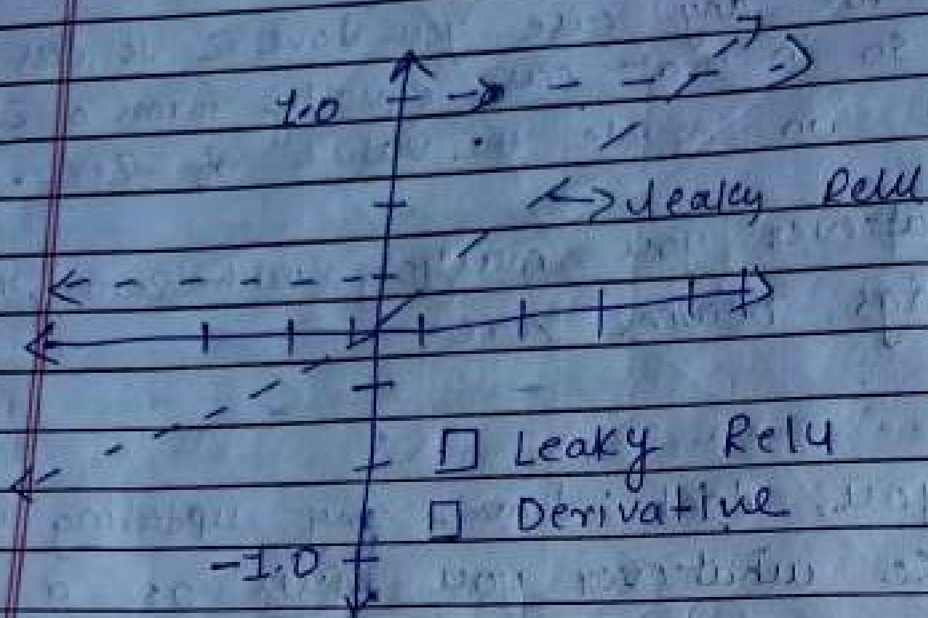
now, whatever you multiply with zero, then whole things become zero

→ So you dont have any updating weight here. so whatever you gives as a random weight remains same weight even after alot of calculations. so this problem is known as dying Relu or Dying Neuron problems

Q→ what was the problem with sigmoid and tan h

Ans → ~~wt~~ well it was lots of calculation and sigmoid have values lies between 0 and 1 while tanh had it -1 to

* Next function
Leaky ReLU



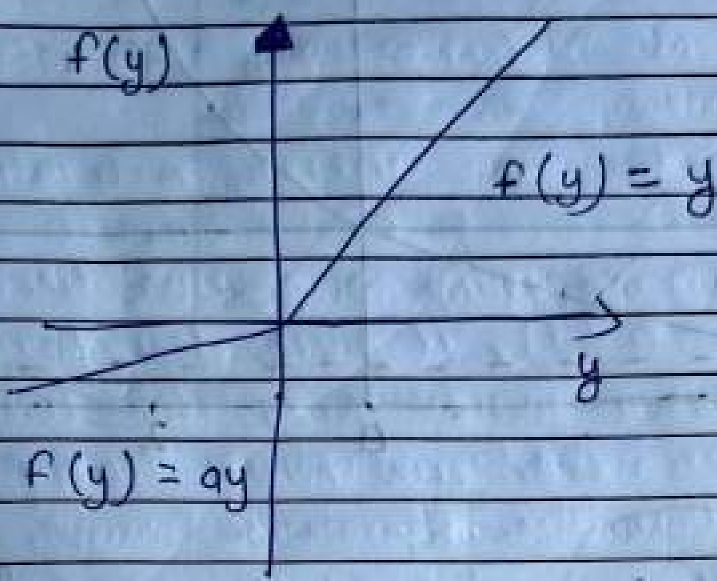
-> leaky relu try to solve the function of Relu

-> what is the function of Leku Relu

max(0.01z, z)

-> it trying to elimate the problem of dying Relu and dying Neuron using max(0.01z, z) function

* Next parametrized Relu.

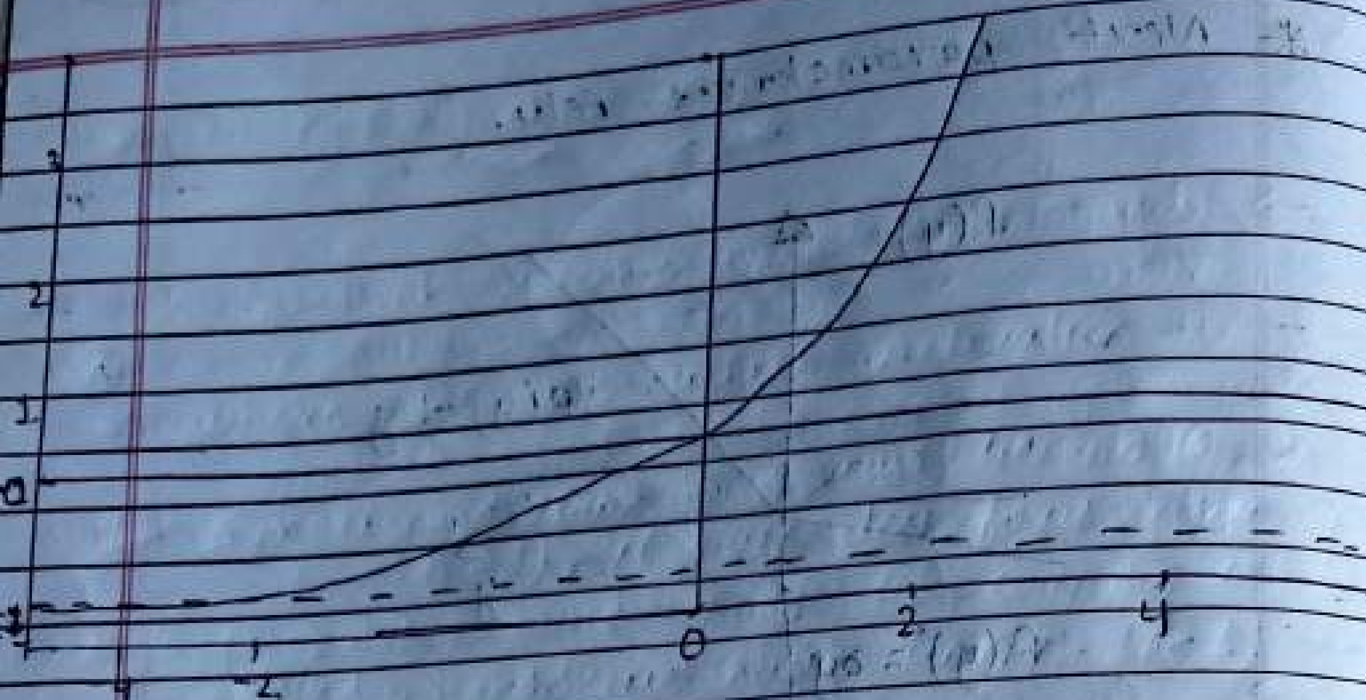


Here the function is
max $(\alpha z, z)$ α - Learning ra
↓
alpha can any number that has been train by the model
model has to decide

So we have alpha if $z < 0$ and 1 if $z >= 0$.

* ELU -> exponential Relu

ELU activation function $(\alpha = 1)$

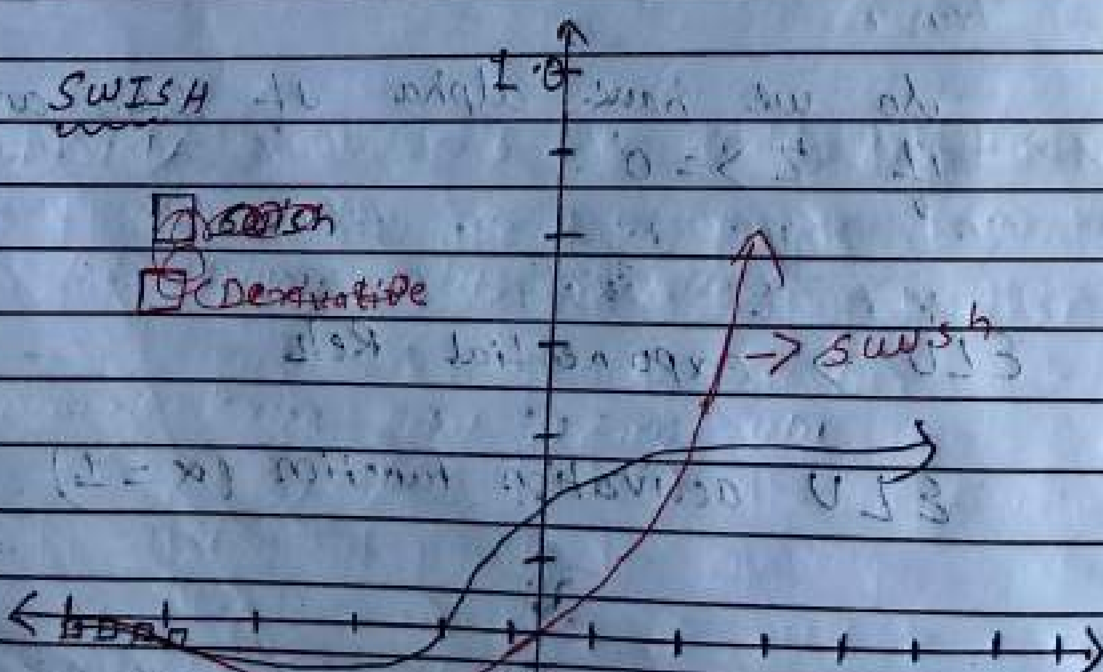


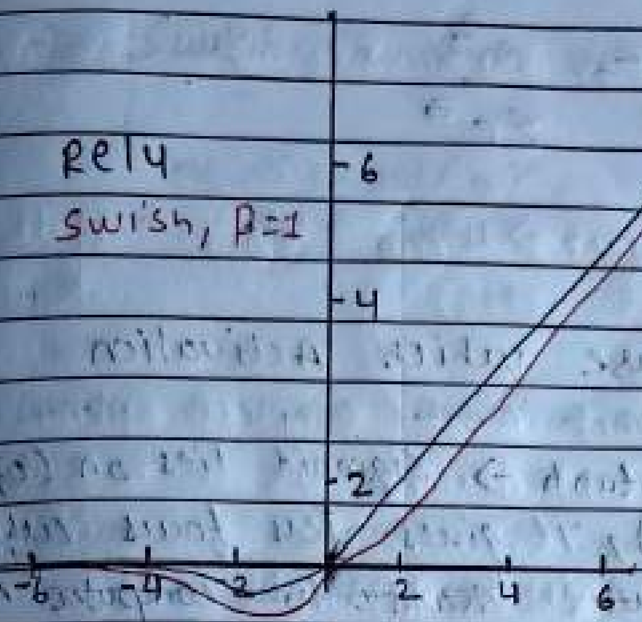
$$f = \begin{cases} \alpha\left(e^{z}-1\right) & z<0 \\ z & z \geq 0 \end{cases}$$

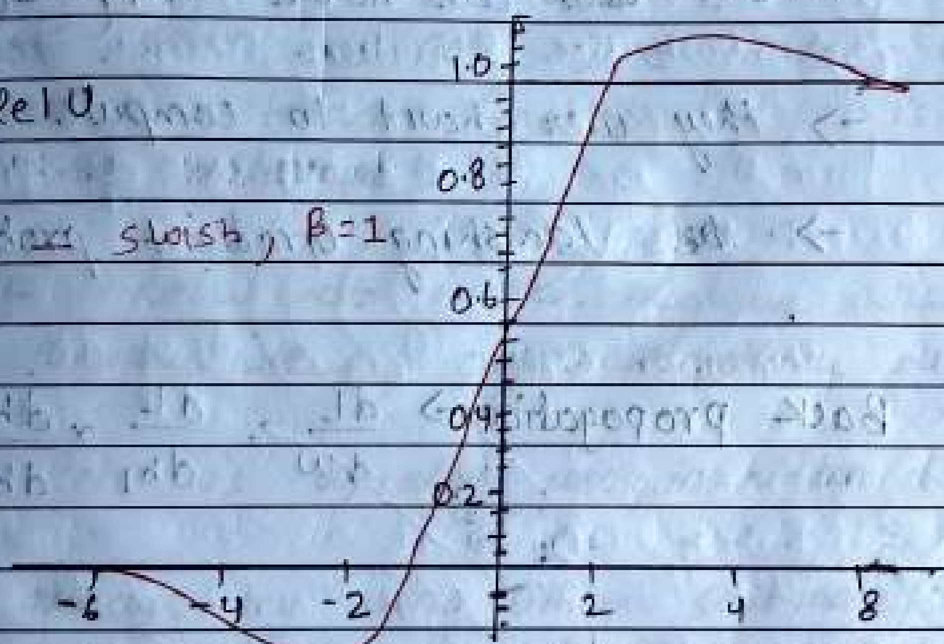
$\alpha \cdot e^{z} \rightarrow z<0$

$1 \rightarrow z \geq 0$

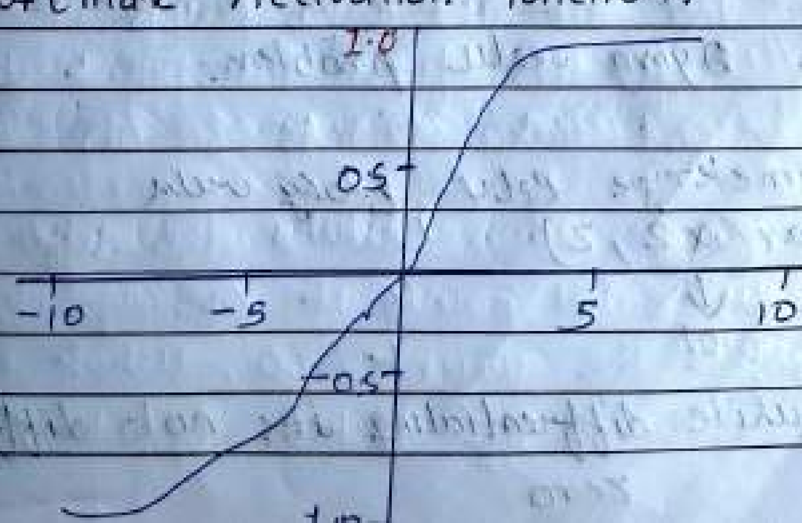
* SWISH







* Softmax Activation function.



$$a = \frac{e^z}{\Sigma e^z}$$

* Where to use which Activation

1- Sigmoid and tanh → sigmoid lies on (0 - 1), it gives less focus when it comes to ~~→ they were hard to computer~~ negative values. so we have tanh (-1 to 1)

→ they were hard to compute

→ Vanishing Gradient problem.

* Back propogation → $\frac{dL}{dw} = \frac{dL}{dh_1} \times \frac{dh_1}{dh_2} \times \frac{dh_2}{dw}$

*2 ReLU

the Dying relu problem.

3- parametrize ReLU, Leaky relu
- $\max(\alpha z, z)$
  ↓
  0.01

→ while diffrentiating its not diffrenciable at zero

4 -> Swish function

$x \cdot \text{sigmoid}(n)$

* where to use which activation

- Sigmoid and tan h avoided (vanishing gradient)

-> ~~Relu~~ Relu used it is a very versatile activation function and it is use most problems.

-> if Relu not work then we shift to leaky relu, or parameterize relu

-> Relu is usually use in hidden layer

-> always try using either softmax/sigmoid when it comes to classification problem

* optimisers in depth 1

Optimisers

Gradient Descent $\rightarrow w_{new} - w_{old} - \eta \frac{dL}{dw_{old}}$

Gradient descent most simplest optimiser and even gradient descent is a 3 types

(1) Batch Gradient Descent
(2) Mini Batch Gradient Descent
(3) Stochastic Gradient Decent

formula for all 3 types of gradient descent is same.

$$w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$$

$\rightarrow$ Batch GD $\rightarrow \frac{dL}{dw}$

It calculates the ~~dot~~ differentiation $\frac{dL}{dw}$ for all the data points it calculates for the first datapoint then second data points so on. ~~dot~~

Once it calculated the differentiation of all data points it average out and updates the weight. So, what is the problem of Batch Gradient descent

-> it becomes kind of slow

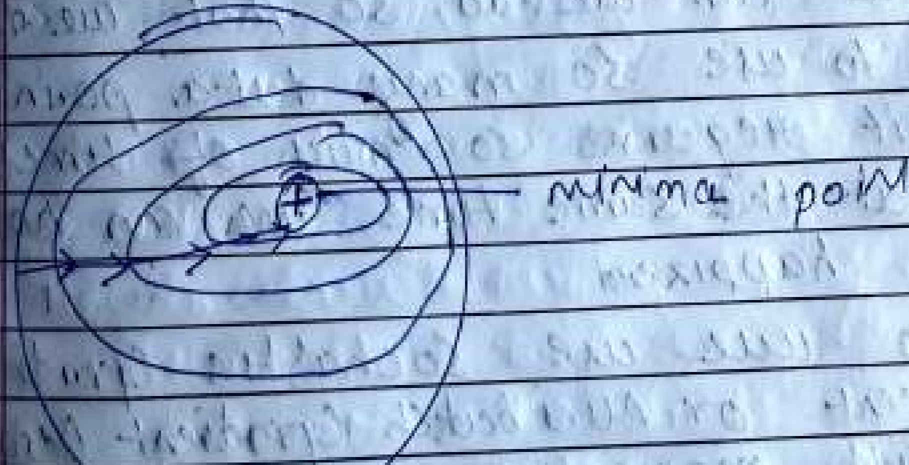
2- Mini Batch Gradient Descent

As the name suggest it will not use the all data points. it will randomly select data points it will calculate the ~~dL~~ $\frac{dL}{dw_{old}}$ of these 100 data points and it will update the weight.

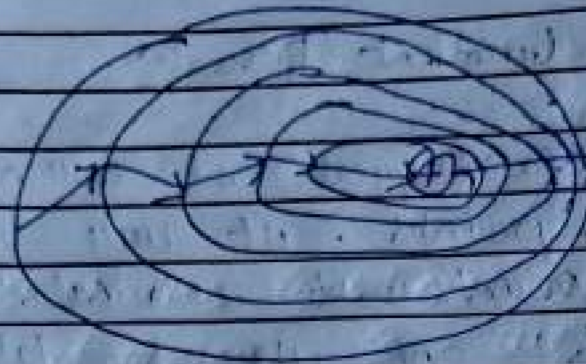
3-> Stochasti Gradient Descent

Here, it ~~use~~ randomly select one data points and update the weight $\frac{dL}{dw}$. and weights are change base on the differentia of that one data points
it take so much time

1- Batch Gradient Descent



* Mini Batch Gradient Descent

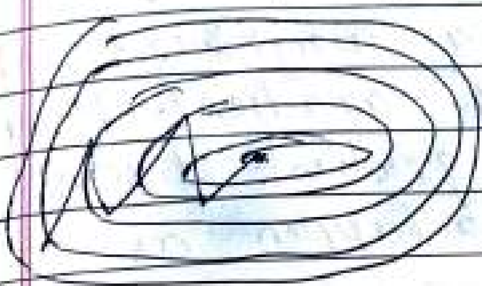


* Stochastic Gradient Descent

* So what can we do, so that we dont have to use so many data points and it require so much of times but at the same time we can not be this happizerd

So we use Sochastic Gradient Descent or Mini batch Gradient Descent with momentum.

S.G.D
Sochastic Gradient Descent



Sochastic Gradient Descent with momentum

so $w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$

~~wt = wt-1 -~~

$w_t = w_{t-1} - \eta \frac{dL}{dw_{t-1}}$

$= V_t = \gamma V_{t-1} + \eta g_t$

$= w_t =$

$$\Rightarrow w_t = w_{t-1} - \eta g_t - \gamma \eta g_{t-1} - \gamma^2 \eta g_{t-2} - \dots - \gamma^t \eta g_1$$

* Stochastic Gradient Descent momentum →
So, we learn about the term momentum, so in short basically means giving weightage to the previous calculated gradient descent so that the next gradient descent ~~is~~ not as happen ~~to~~ happen as previous one.

* optimisers in depth 2

1) Adagrad →

$$w_t = w_{t-1} - \lambda g_t$$

idea → So ~~main~~ main idea behind Adagrad was, we change instead of changing of gradient descent at all time what if some how come with a method that can change the learning rate so, we need to change the learning rate with every epock or iteration

* To change learning rate with every epoch so that if we get closer to the minima in decreases it self, so we dont have to change the learning rate.

$$w_t = w_{t-1} - \eta' g_t$$

$$\frac{\eta}{\sqrt{\alpha_{t-1}}}$$

problem → since α_{t-1} is sum of all the gradient ~~seq~~ squares

(2) Ada delta → optimising the weight so that we can keep it in control our learning rate does not decrease very fast. So what is the formula behind ada delta?

$$w_t = w_{t-1} - \eta' g_t$$

$$\eta' = \frac{\eta}{\sqrt{eda_{t-1} + \epsilon}}$$ eda → exponential dying

$$eda_{t-1} = \gamma\, eda_{t-2} + (1-\gamma)\, g_{t-1}^2$$

Gama Gama

Suppose → $\gamma = 0.99$

$$eda_{t-1} = 0.01\, g_{t-1}^2 + 0.99\, eda_{t-2}$$

$$= 0.01\, g_{t-1}^2 + 0.99 \left[0.01 g_{t-2}^2 + 0.99\, eda_{t-3} \right]$$

$$= 0.01\, g_{t-1}^2 + 0.99 \times 0.01\, g_{t-2}^2 + 0.99^2\, eda_{t-3}$$

→ So in one sense this is kind of like momentum term!

EDA → Exponential dying average

* Adam → Adaptive Moment ~~e~~ estimation

So, what adam came across was along with using gradient square we were also use gradient that means instead of ~~using EDA~~ doing EDA of just gradient square we will also do EDA for gradience.

$$m_t = \beta_1 m_{t-1} + (1-\beta) g_t$$

$$v_t = \beta_2 m_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$w_t = w_{t-1} - \alpha' \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

→ Adam has been proven to be a very good method it is used ~~to~~ in most of the cases. Now let us look into which optimiser is to use well.

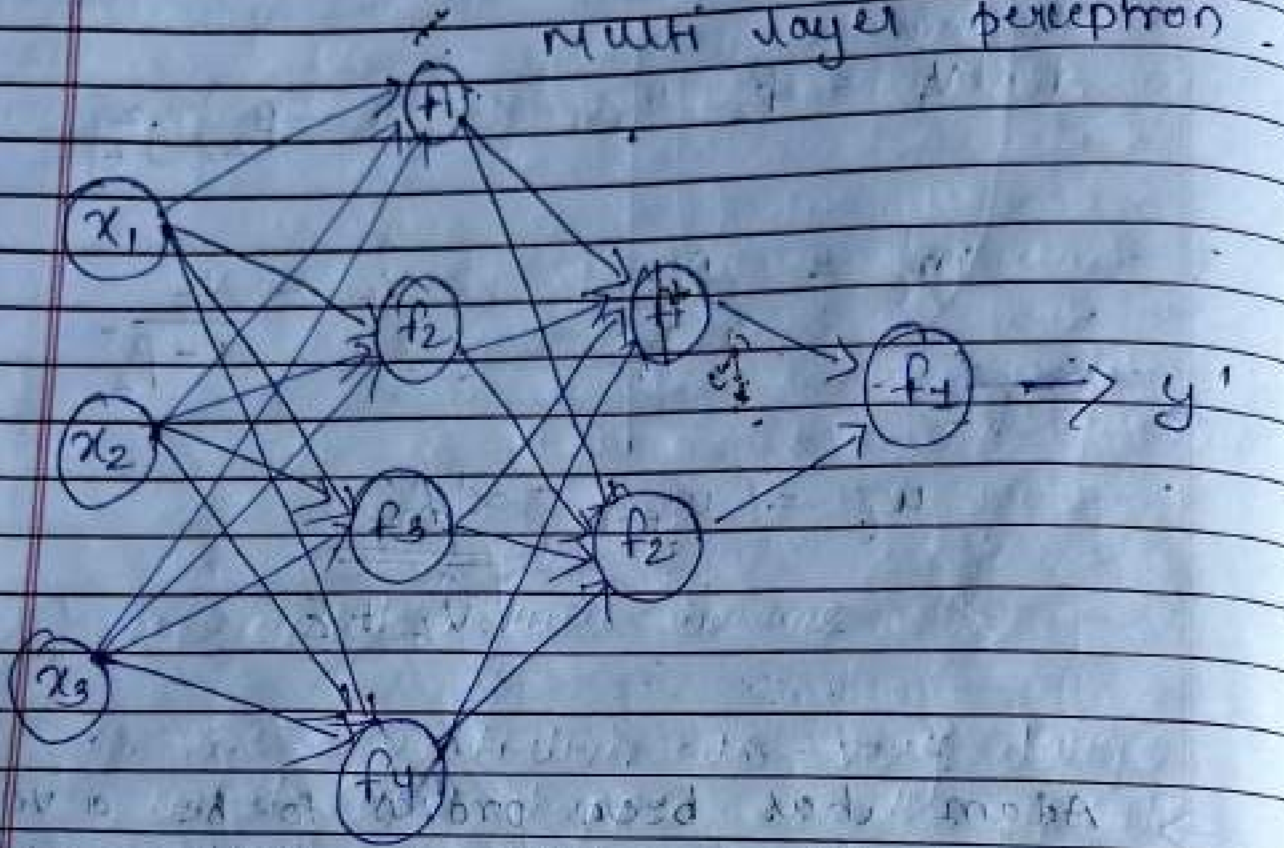
* Some general rules

* ~~Sparse~~ Sparse data ⇒ like data from bag of words. in that case go for adam function (means any ada can do working fine) most likely use adam.

* 1. Adam → we should use Adam most of the time. So, adam is very versatile. ~~So~~ it convert its faster and we should use this is most of time in most of the cases.

→

* weight initialiser =>

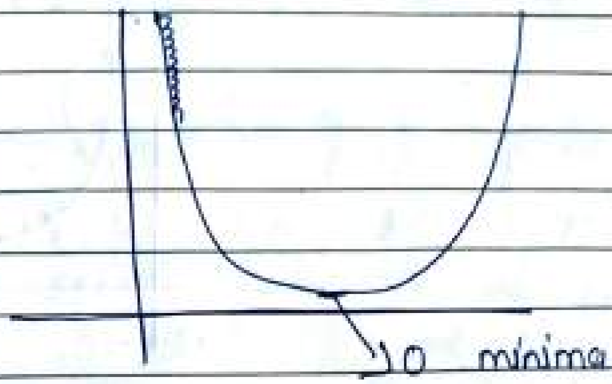


if you ~~cannot~~ remember structure of a MLP and whenever we assigned weights what i mention was this weights randomly assign. now this statement partially true. so what that means is it's is randomly assign yes, but its is randomly assign from a distribution

so all the weights will be in some sort of the distribution (Normal distribution, uniform distribution or something)

classmate
Date
Page

-> we know that weight should ~~no~~ neither be too small and not too big so. why is that.



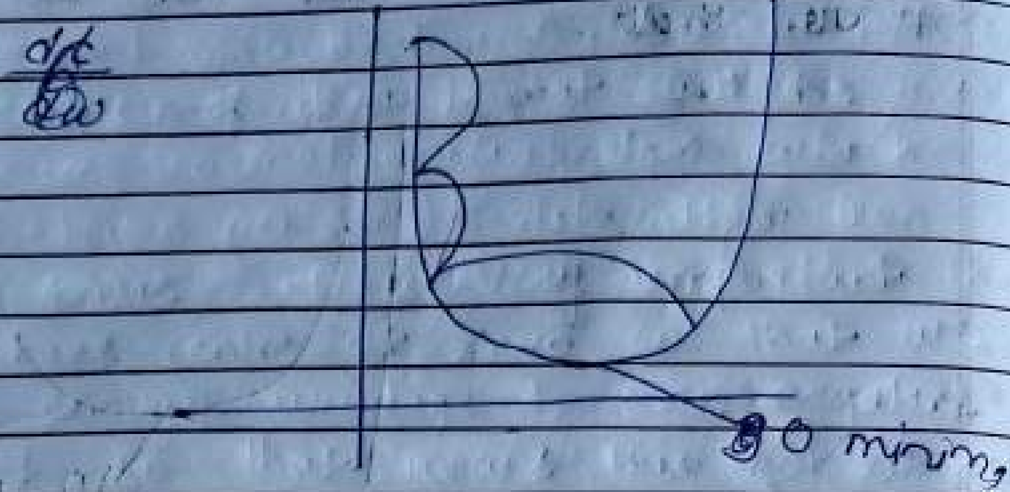
so, we have the loss here which is against the weight so what i mention is if we scope very slowly our weights are not getting updated a lot and it will take very long time to converg a may be not even reach here

so, if differentiation is very sma our weight will not get updated and this problem statement was knowns as vanishing problem.

this happen when weights ar too small

Q- what happen if the weight are too big

Ans-> we come across the problem of Gradient explosion or exploding gradient Just the opposite of vanishing gradient.



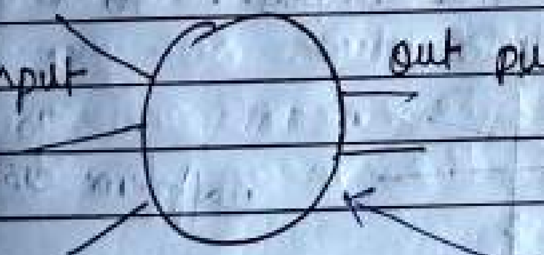
~~the update will become~~

if the differentiation is very large what will happen is the update will become haphazard. So let say we are here. So here we go down very big step the next turn again we will very big step and suppose we realise foreward a bit so we want to go back but since the steps are very big again go back this whole big time so this problem is known as Exploding Gradient problem. and this happen when weights are too big.

* So we need to some how clip this weights so we need to keep them in desk ~~do~~ so that do not cross the particular range. So for that ~~we~~ we need a distribution

* weights should not be equal ?
If all this weights are equal there updates will be equal. the new weights are also equal. hence there is no some how defining which input should be given more weightage compare to the other input hence we should make sure the weights are ~~to~~ 1 not too big not too small
2 the weights are not to be equal

So they should be randomise. but randomi through a distribution

* input

out put

So let say this is a neuron it has three inputs and two output
So we have a term called fan in.
(Fan in) = number of inputs

fan out -> number of output

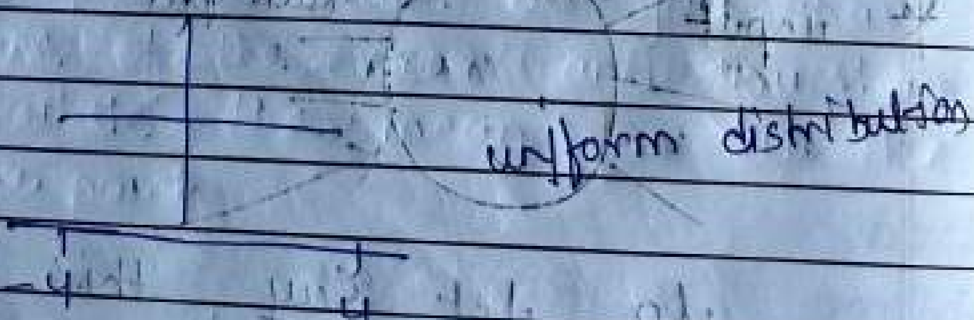
40 σ -> sigma

* Initialization

1 -> Normal initialization -> So all the weights of this format will be initialize from a normal initialization which lies between $(0, \sigma)$, like this is very basic it is very important that we have centre at zero, so that we give enough waidage to even negative weight. So that means this not at all related in any way to the output that we are getting.

uniform distribution -> So the weights will be initialize from a uniform distribution which lies between $\frac{-1}{\sqrt{fan\,in}}$ to

$\left.\frac{1}{\sqrt{fan\,in}}\right]$ here it totally depends on upon the input. So



3 -> Xavier/ Glorot initialization -> So, these both of the name of scientist and they

- It's usually use for sigmoid activation function
- They again have two types of different initialization

1- Normal initialization -> where weight.

$$w_{ij}^{k} \sim N(0, \sigma) \quad \sigma = \frac{2}{\text{fan in} + \text{fan out}}$$

2 -> uniform distribution -> the weights randomly initialize from uniform distribution which lies between $\left[\frac{-\sqrt{6}}{\sqrt{\text{fan in} + \text{fan out}}}, \frac{\sqrt{6}}{\sqrt{\text{fan in} + \text{fan out}}} \right]$

3 -> He initialization

1- Normal initialization -> the weight's random from a normal distribution which lies between $(0, \sigma)$ $\sigma = \sqrt{\frac{2}{\text{fan in}}}$

2 -> uniform distribution -> weights come from a uniform which lies between

$$\sqrt{\frac{F.\sqrt{6}}{\text{fan in}}} \text{ to } \sqrt{\frac{6}{\text{fan out}}}.$$

Now, normal and uniform totally depends on what type of data you are using. So he initialization totally depends on what type of data you are using. So he initialization is usually used with activation function like ReLU, Leaky ReLU,

* Losses in Neural Networks

Regression loss →

(1) MSE →

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

So you calculate the mean of this and you get mean squared error so this is the squared error. So this squared error, error is basically the difference between the actual value and the predicted value. So you have the square of the error and you have to calculate the mean for all the data point

(2) MAE → it is same as MSE but instead of doing the square part we will be doing

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

43

Q- why do we need this two function and How are they different

Ans- first of all MSE -> is very bad when it comes to finding the outliers. so its not at all robust to outliers.

in a way like suppose you have a value that is way outside the dataset or way different from the actual distribution of the dataset. So in that case if you will subtract the value is going to be very large value and then again if you squared this is going to be a huge loss. So it actually skewss the actual loss. So that is why mean squared error is not robust two outliers.

MAE -> it Robus to outliers.
when you have very small error so in that case the problem is the value become very small and the model will not actually learn anything because the loss is going to very small.

$\delta \to$ delta

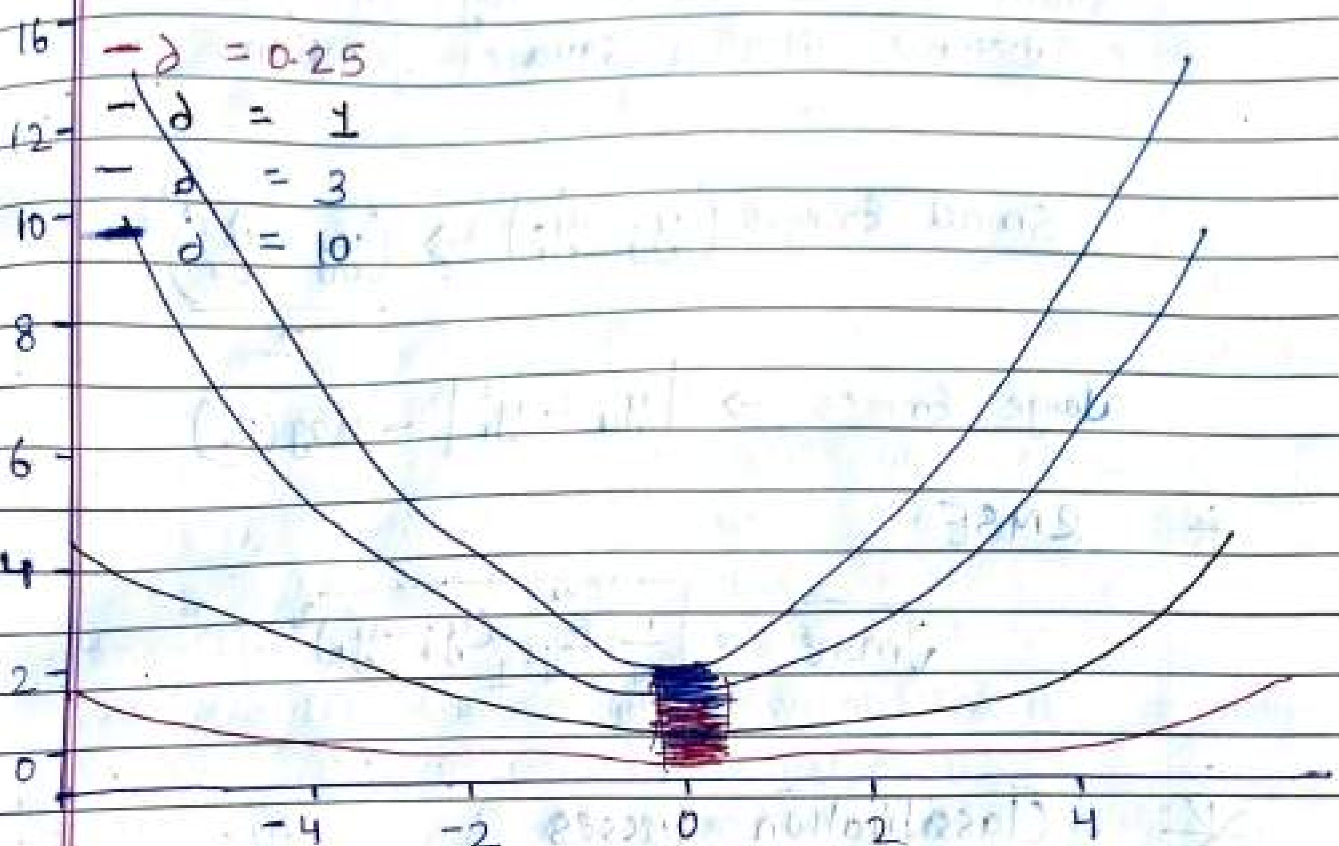
** Huber loss -> it's basically a combination of MSE and MAE taking a best of both.

Huber loss = $\begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{when} \\ \text{mod} \leftarrow |y_i - \hat{y}_i| \le \delta \end{cases}$

otherwise $\delta |y_i - \hat{y}_i| - \frac{\delta^2}{2}$

if your error is very small so at that case square the losses or the errors. So it will square the error. So you loss is not the very small and it will penalized the loss as well as the loss very big it will not do the square part of it will just use the absolute value

Huber loss for target = 0



predictions for target = 0

~~parade~~

problem → deciding the value of delta

δ

* log cosh → it is a good loss function but it is not exactly used a lot.

$$\log(\cosh(y_i - \hat{y}_i))$$

for very small values ~~to~~ ~~w~~. for very small errors. log cosh function behave like a squared function.

$$\text{Small Errors } (y_i - \hat{y}_i) \rightarrow \frac{(y_i - \hat{y}_i)^2}{2}$$

$$\text{large Errors} \rightarrow |y_i - \hat{y}_i| - \log(2)$$

* RMSE

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

* Classification losses

(I) Binary classification problem

(1) Binary cross entropy $\rightarrow -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$

labels $\rightarrow (0, 1)$

(i) Entropy - $y_i \log y_i$

(ii) Cross Entropy - $y_i \log \hat{y}_i$

(2) Hinge loss $\rightarrow \text{Max}(0, 1 - \hat{y}_i y_i)$

labels are $\rightarrow (-1, 1)$

Suppose your $\hat{y}_i$ is very close to y_i let say both are 1, so in that case

what will happen. you will have maximum off
max(0, 1 - (1x1)

max(0, 1 - 1)

max (0)

if your actual predicted value is very close to the actual value then your loss will be 0. ~~in the same way~~

(II) classification losses for multiclass ~~p~~ problem

It is a same thing you dont have a lots of different as such so instead of binary cross entropy we have categoral cross entropy.

(i) categorical cross entropy ->

$$= \frac{1}{N} \sum_{J=1}^{N} \boxed{\sum_{i=1}^{c} y_i \log \hat{y}_i}$$

(ii) KL Divergence -> this is we when you have very different distribution and you need to find the difference between the distribution.

KL:

$$= \sum p(x) \left[\log(p(x)) - \log(q(x)) \right]$$

- KL divergence is used in generative models

we ~~???~~ discuss about the different loss functions of neural network and we discuss how similar this was to machine learning losses. So finally we completed one basic neural network structure. which is multilayer perceptron. we went over all the concepts of optimisers layers, activation function and everything so we completed multi layer perceptron.