① ANN
② CNN
③ RNN

PAGE No.
DATE / /

Deep Learning

① ANN

ANN ⇒ {Unit of Deep Learning

Input layer — Hidden layer (Hidden layer, it can have any no. of Neuron) — output layer

f1, f2, f3 are my features

The info is getting passed to Hidden layer

Some kind of pre-processing happen over here (in HL)

then we get o/p

↑ all are node

↑ all are Neuron

(Also it can be any no. of o/p)

Eg: Dog, Cat

★ How Neural Network Works

polynomial Linear Equation

$y = w_1x_1 + w_2x_2 + w_3x_3 + b$

$z = Act(y)$

(If we are going to do summation (Σ) then passing through sigmoid function (step1, step2))

I/P layer — Hidden neuron — O/P Layer

w_1, w_2, w_3, w_4

$z = z * w_4$

$= Act(z)$

Forward Propagation

Neuron

Step 1: Σ (weight and I/P) $\sum_{i=1} w_i x_i$

Step 2: $Act\left(\sum_{i=1} w_i x_i\right)$

Act Σ (weight and I/P)

★ Activation Function

① Sigmoid Af

$\frac{1}{1+e^{-y}}$

1 — 0.5 — 0

0.5 to 1 = 1

0 to 0.5 = 0

Activation function Transform your (y) between 0 and 1

$y = \sum_{i=1} w_i x_i + b_i$

$Act(y)$

② Relu Af

$max(y, 0)$

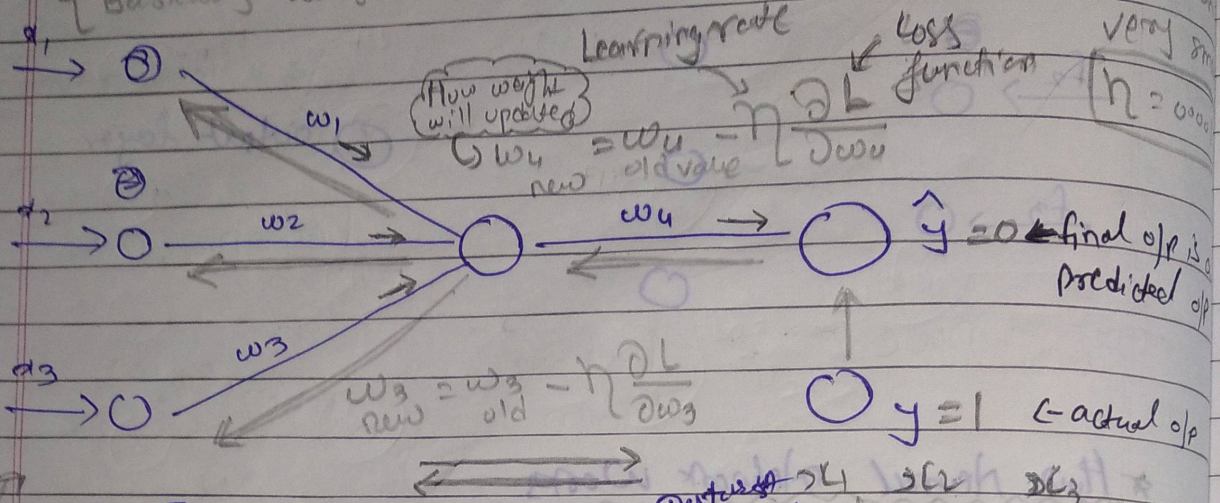
$y = \sum_{i=1} w_i x_i + b_i$

(-ve) → max (-ve, 0) = 0

(+ve) → max (+ve, 0) = +ve

* Neural Network Training (with Back Propagation)

- To reduce the loss value, we have to back propagate
- Basically weights will be updated due to back propagation



$y = [x_1 w_1 + x_2 w_2 + x_3 w_3] + b_1$

$x = Act(y)$

Dataset

x_1 Play	x_2 Study	x_3 Sleep	o/p
2h	4h	8h	1

↑ Pass these value

∴ Minimize loss (or) we have to decrease the loss using optimizer

↳ Optimizer

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

(actual value, predicted value)

$Loss = (1-0)^2 = 1$

(y and $\hat{y}$ should be same)

Note :- Sigmoid function transform the value between 0 to 1.

$$Loss = \sum_{i=1}^{n} (y - \hat{y})^2 \leftarrow \text{multiple record}$$

- Forward And Backward propagation

* MNN (Multi layer Neural Network) Traing

* Above MNN Explanation

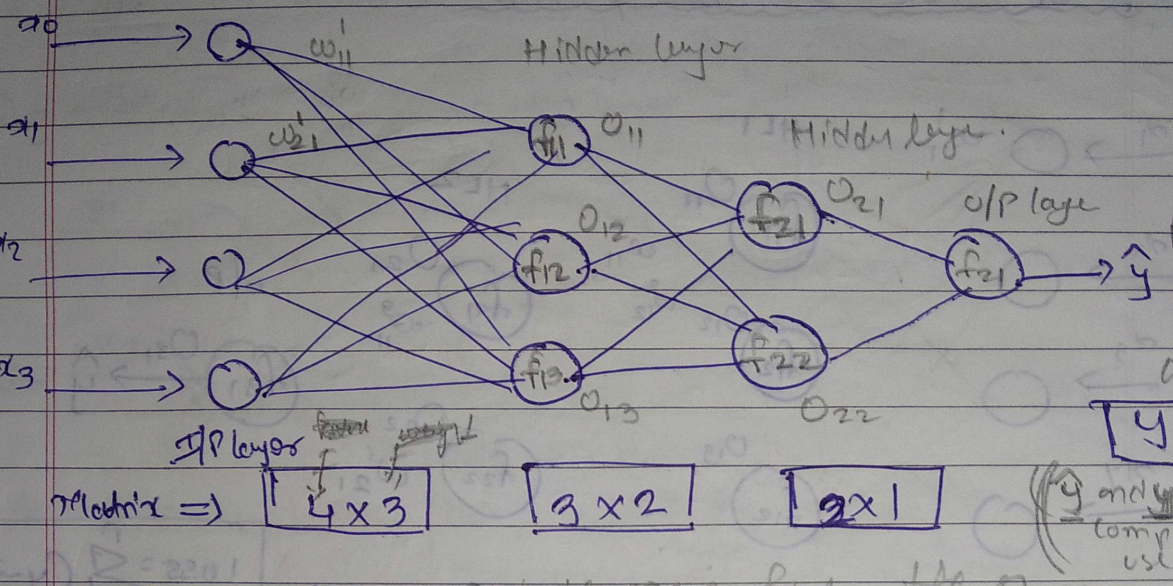
$\hat{y}$ → predicted o/p

I already know that my traing Dataset the o/p for this kind of Data is → 1.

Suppose my $\hat{y}$ is predicting as 0. we need to compare these whether $\hat{y}$ and y are almost same. This Difference can be found out using loss

$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$ (Gradient Descent)

{Gradient is Slope
{Decent → Downward



(y and $\hat{y}$ now we compare, we use Loss fun)

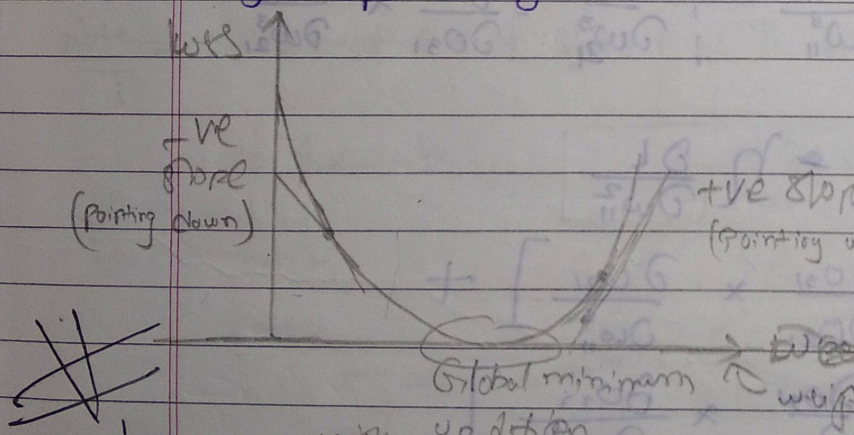
$Loss = (y - \hat{y})^2$

* Derivate means finding the slope.

Main aim is to reduce the loss value, for getting similar value y and $\hat{y}$.

↳ we use Optimizer (GD) ⇒ $w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$ ← learning rate

weight updation formula



derivative means we need to find out slope

Imp

The main aim of GD, is, we should be reaching to the Global minimum

Note { Unless and Until we don't reach atleast Global minima the updation of weights will happen with the help of Back propagation

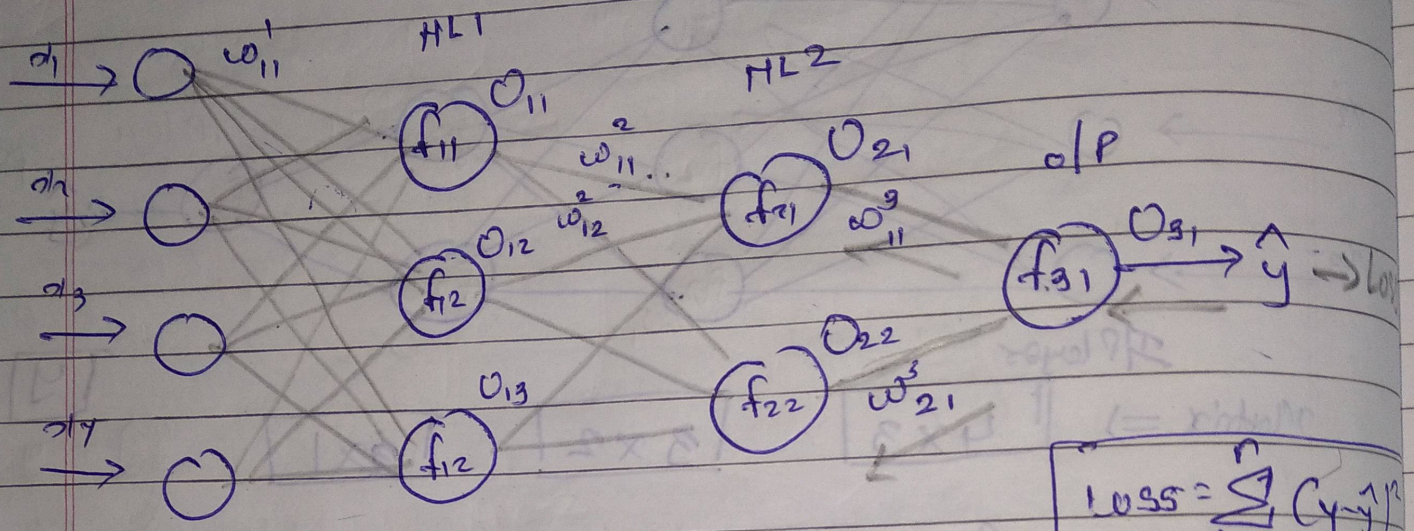
⇒ { Once we get the o/p then what we compute the Loss $L(y, \hat{y})$, the loss need to optimize using optimize

Note $\frac{dy}{dx}$ what is this dy/dx ?

⇒ we are going to find out Tangent or Slope over the single point

* Chain Rule In Back Propagation

w_{11}^3 → Impact the o/p of O_{31}



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

(i) $w_{11}^3 \text{ new} = w_{11 \text{ old}}^3 - \eta \frac{\partial L}{\partial w_{11}^3}$ slope

Update

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^3} \quad ; \quad \frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial w_{21}^3}$$

(ii) $w_{11}^2 \text{ new} = w_{11 \text{ old}}^2 - \eta \frac{\partial L}{\partial w_{11}^2}$

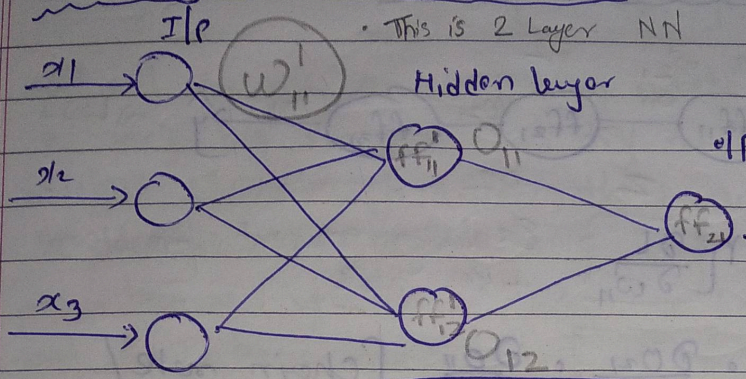
$$\frac{\partial L}{\partial w_{11}^2} = \left[\frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial w_{11}^2} \right] + \left[\frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial w_{12}^2} \right]$$

* Sigmoid transform the value between 0 to 1

Note ⇒ The derivative of Sigmoid will be specifically between 0 to 0.25

PAGE No. DATE

* Vanishing Gradient Problem



- This is 2 Layer NN

optimizer ↓

- **Bias updation** $b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$
- **weight updation**

$$w'_{11\,new} = w'_{11\,old} - \eta \frac{\partial L}{\partial w'_{11\,old}}$$

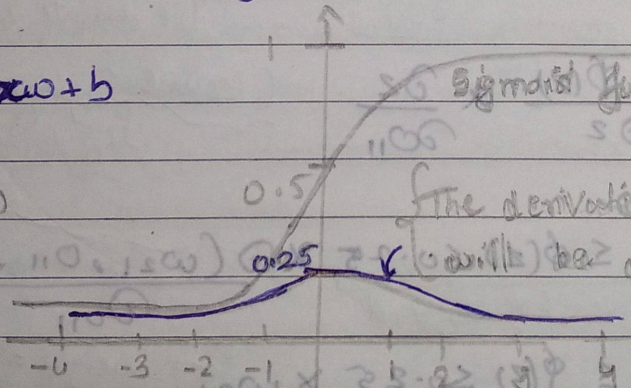
$$\frac{\partial L}{\partial w'_{11}} = \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial '_{11}} \quad \text{(chain rule)}$$

The major problem with Sigmoid function when we are doing Back Propagation, Vanishing GD

Note ⇒ The derivative of sigmoid will be value between 0 to 0.25

$z = \sum_i x_i w_i + b$

Some problem with tanh



$z = \sum_i w_i x_i + b$

$$\frac{\partial \sigma(z)}{\partial z} = 0 \text{ to } 0.25$$

range = $0 \le \sigma(z) \le 0.25$

Why problem

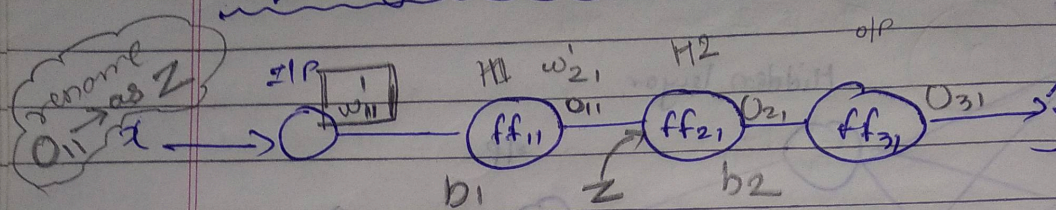
Before like 2006, Researchers where not able to create a Deep Neural Network in ANN, the reason is that we, have to use Sigmoid in Each and Every function in Each and Every neuron. Because of this sigmoid we are facing problem is called Vanishing Gradient Problem.

(Note:- Activation function called Relu was not invented)

{ Vanishing ⇒ problem due to Activation function
{ Exploding ⇒ Problem due to weight Initialization

PAGE No.
DATE / /

* Exploding Gradient Problem



weight updation ⇒ $w_{11}' = w_{11old} - \eta \left[\frac{\partial L}{\partial w_{11}}\right]$ ← Exploding gradient problem takes here

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial O_{31}}{\partial O_{21}} \cdot \left[\frac{\partial O_{21}}{\partial O_{11}}\right] \cdot \frac{\partial O_{11}}{\partial w_{11}} \quad \text{[chain rule]}$$

Act

$O_{21} = \phi(z) \quad \frac{1}{1+e^{-z}}$

$z = w_{21} \cdot O_{11} + b_2$

$$0 \le \phi(z) \le 0.25$$

Note:- Exploding Gradient Problem, it does not just happen because of ~~sigmoid~~ Sigmoid function. The main reason this Exploding gradient problem happens is because of weights. Let's take $\frac{\partial O_{21}}{\partial O_{11}}$ and let's compute this value (i.e derivate value over here)

$$\frac{\partial O_{21}}{\partial O_{11}} = \frac{\partial \phi(z)}{\partial z} \times \frac{\partial z}{\partial O_{11}}$$

(Assume Sigmoid Act function) ↳ $\frac{1}{1+e^{-z}}$

- Consider my weight initialize here is 500. Then multiply ⇒ 0.25 × 500 ↳ 125
- Whenever weights is higher then only it will perform "Exploding"

$$= \underbrace{0 \le \phi(z) \le 0.25}_{\text{derivative of sigmoid}} \times \frac{\partial (w_{21} \cdot O_{11} + b_2)}{\partial O_{11}}$$ ← derivative of constant

$$= 0 \le \phi(z) \le 0.25 \times w_{21}$$ ← initialize weight = 500

$0.25 \times 500 = 125$

Imp ⇒ • If the weights are higher, what may happen you may not converge to the global minimum point.

- If my weights are higher i usually get a higher value of derivate, but my derivative of sigmoid is between 0 to 0.25, but because of weights this derivate value is becoming higher larger and when it becomes larger because of the chain rule, as i create a deep ANN

this particular derivate [i.e $\frac{\partial L_{oss}}{\partial w_{11}}$] with respect to w_{11}

will become a very big number and when it becomes a very big number if i try to apply that in the weight updation formula, then wold and wnew will be completely different will have huge gap between them and because of that, what will happen after each and every back propagation it will never reach the global minimum point. So that is why weight updation is important.

* Drop Out & Regularization [Refer Random forest]

* whenever we have multilayered NN underfitting will never happen, because we will be having multiple layers in a multiple layer NN.
* If we have one layer NN at that underfitting will usually happen.
* For multilayered NN we will never face underfitting but, overfitting will be a problem

Note * There are two basic ways to solve and overfitting problem

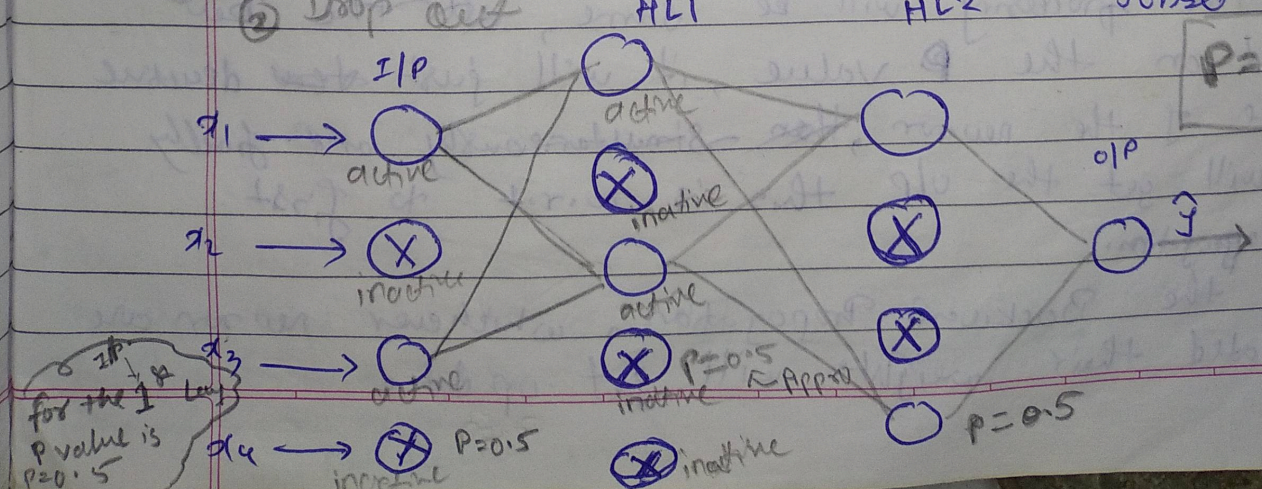
(1) Regularization
(i) L1 (ii) L2

In RF we create multiple DT

Random forest
DT → overfitting
Subset of features.

P = Drop out ratio

(2) Drop out



→ w.r.t Probability (P) → features will be selected Randomly

PAGE No.
DATE / /

→ To implement drop out layer, we basically do is that we select our dropout ratio

→ Drop out ratio between
$0 \leq P \leq 1$

The drop out ratio indicate that how we did for the random forest like a selected sample or subset of features similarly over here we'll be selecting subset of features from the i/p layer, Similarly we'll be selecting subsets of Activation function or the Hidden neurons in the hidden layer, we will not be selecting everything we will ~~&~~ just be selecting subset of features in Each and Every hidden layer along with the ~~is~~ i/p features. so

• Refer Diagram After Reading ~~the~~ this ↑

→ when my forword and Backward propagation will be going on when i select the P=0.5, it will ~~select some features~~ randomly ~~select~~ select some features it will deactive them, when it deactive them suppose in this case my ~~first~~ second and third node are deactive, so my input will get put all the activation function now in my second layer also in my first hidden layer you should see that again I've selected P=0.5

→ Randomly it will select some of the Activation function some of the neuron over here, it will deactive them all the processing will be same, ~~with~~ the best based on the P value, it will just ~~dea~~ deactive some of the neuron, ~~then~~ simultaneously and finally we will get the o/p this is w.r.t to first Propogation.

→ In the Backward Propogation, whichever neuron are activated their weights will get updated

Note → for Test Data all my neurons get connected for test data, there will be no deactivated or activated neurons everything will be connected.

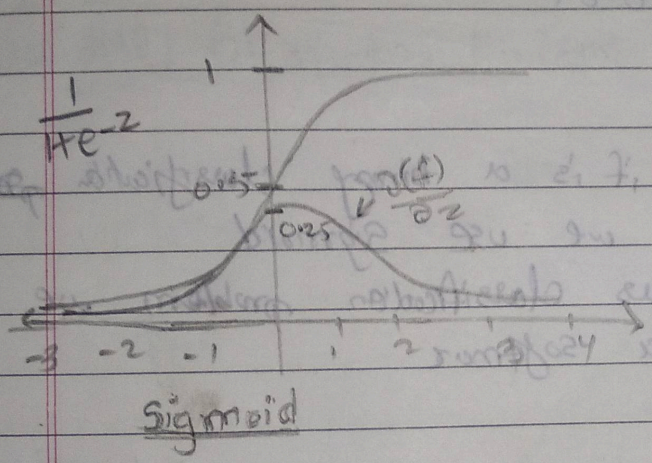
- W*P (W*P) will happen for each and every weights in each and every layer.
- for test Data just multiply W*P

- How we can Select P-value?

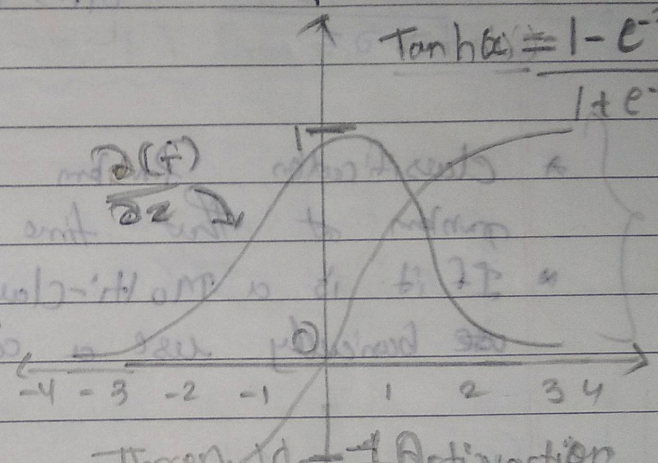
1) - Whereas a Deep NN is doing an overfitting, p-value should be little bit higher

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

* Rectified Linear Unit (RELU)



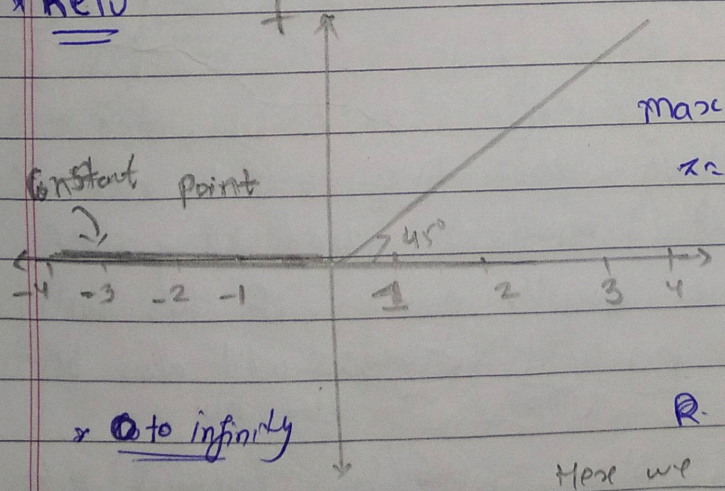
Sigmoid



Threshold → Activation (-1 to 1)

tan 45° = 1

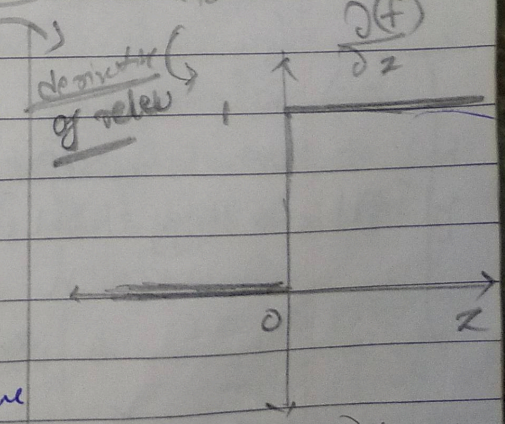
* Relu



max(0, z)

z = +ve

derivative of relu



* 0 to infinity

R. Derivative

Here we have only 2 value (1, 0)

$\begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$

$\frac{\partial z}{\partial z} = 1$

• Relu used in Hidden layer, Because, it solve the Problem of Vanishing Gradient Problem.

(Derivative → 1 or 0 not between 1 or 0)

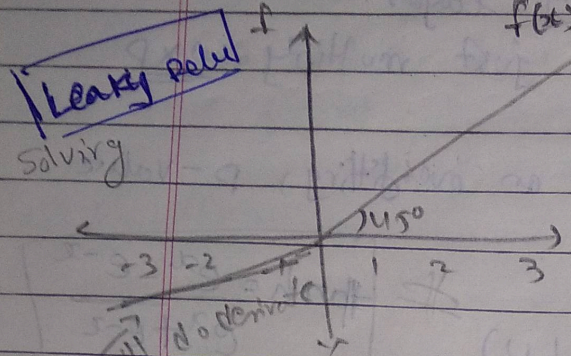
Leaky Relu

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

$= 0 \times 1 \times 1$ ← derivation value

↳ Dead Neuron (Dead Relu) (or) Dead Activation function

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



$$\begin{cases} z & z \geq 0 \\ 0.01(z) & z < 0 \end{cases}$$

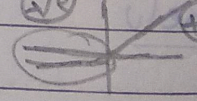
↳ small constant value

$\frac{\partial (0.01) \times 2}{\partial z} = 0.01 \Rightarrow 0.01$

↳ Here o/p will not be 0, like Relu

- Classification Problem, if it is a binary classification problem at that time we use Sigmoid
- If it is a Multi-class classification problem we use basically use a a softmax

Note:- Sometimes Leaky Relu brings up sometime Vanishing Gradient problem, i.e $w_{new} \approx w_{old}$



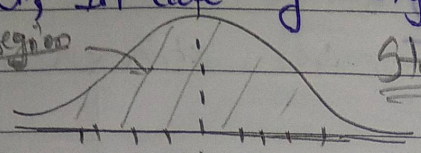
⟹ Reason:- Suppose most of weights are (-ve), maximum no. weights are (-ve), that time w.r.t derivate ($\frac{\partial L}{\partial w}$) it'll be multiplying → (0.01, 0.01, 0.01) everywhere, because of these it will become a smaller number, if this will become a smaller no. ($\frac{\partial L}{\partial w}$) ← this will also become smaller no.

⟹ When we updating Along with (η) learning rate our $w_{new} \approx w_{old}$ ← ($\frac{\partial L}{\partial w}$)

DATE / /

* Activation Functions - Elu, PRelu, Softmax, Swish and Softplus

- The most specific thing about Normal Distribution Data or Gaussian Distribution Data, the Data is zero centered
- But, In case of sigmoid it is not centered



Standard ND

$\mu = 0$

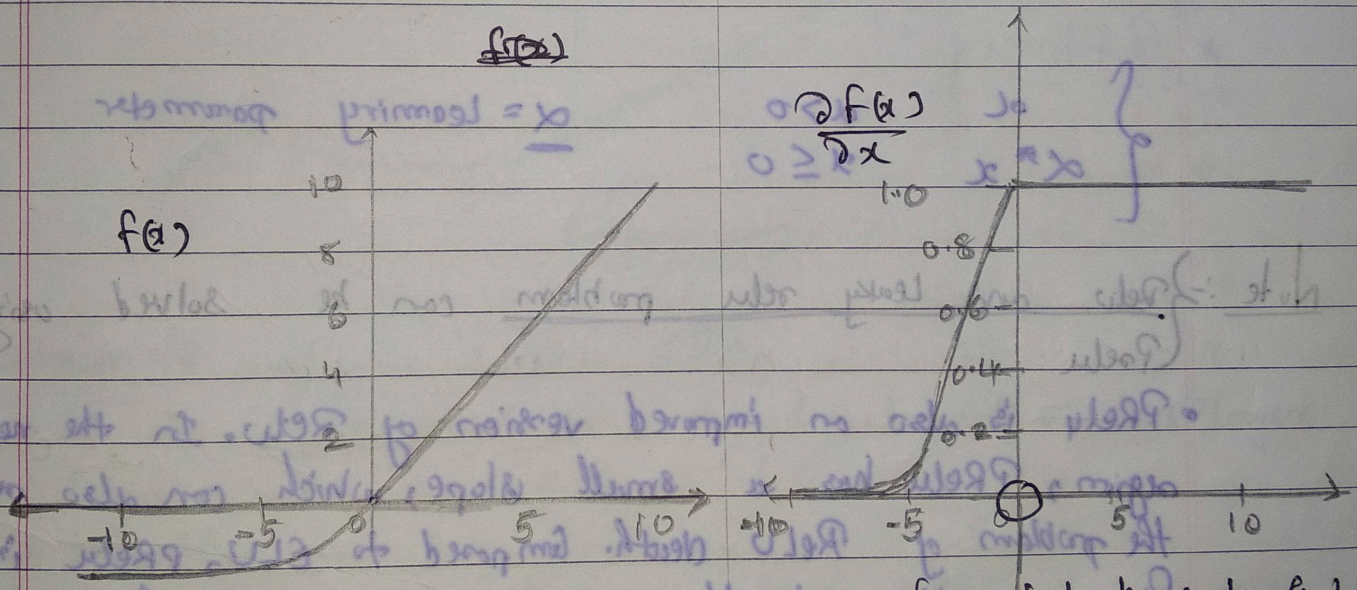
$\sigma = 1$

↳ Data is zero centred

most of the data will be either (+ve) or (-ve), but it will be always be in this specific region which is centred towards 0

- tan h zero centred

* ELU (Exponential Linear Unit) function.



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

o/p

$x \leq 0$

we don't have to find out the derivative y zero

major problem we are solving over here is we don't have to find derivative of 0

$\alpha(e^x - 1)$

α ← hyperparameter

learning parameter

it can be 0.01, 0.1, 0.3

Problem Name

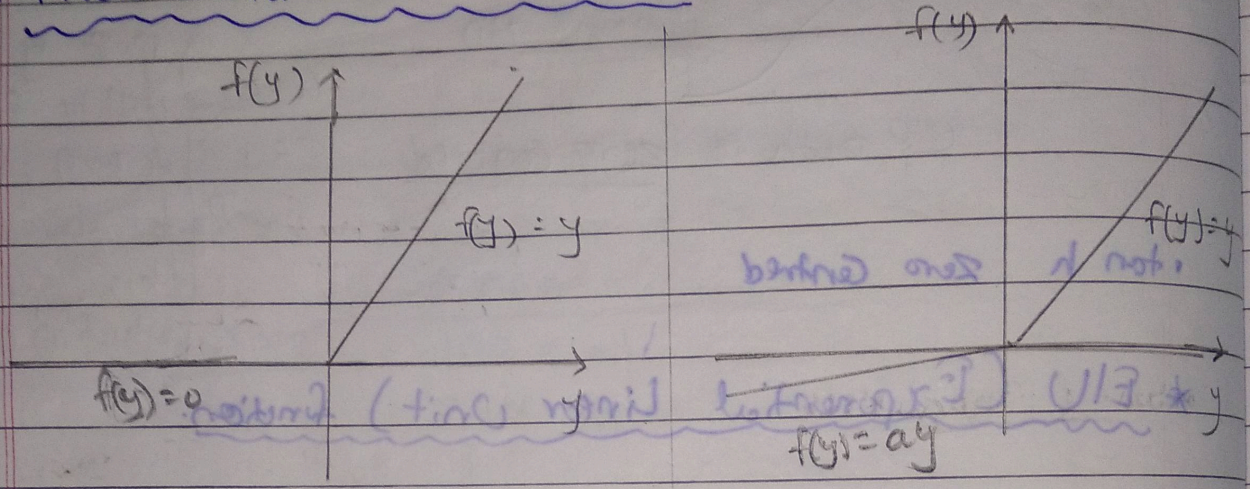
$\frac{\partial \alpha(e^x - 1)}{\partial x}$

Computationally Expensive

same with tan h

- ELU is also proposed to solve the problem of RELU, ELU has all the advantages of RELU
  - No Dead RELU issues
  - The mean of the o/p is close to 0, zero-centered

PRelu (Parametric ReLU)



$$\begin{cases} \alpha & x > 0 \\ \alpha * x & x \leq 0 \end{cases}$$

α = Learning parameter

Note :- Relu and Leaky relu problem can be solved using Prelu

- PRelu is also an improved version of Relu. In the negative region, PRelu has a small slope, which can also avoid the problem of ReLU death. Compared to ELU, PRelu is a linear operation in the negative region. Although the slope is small, it does not tend to 0, which is a certain advantages.

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

The parameter α is generally a number between 0 and 1, and is generally relatively small.

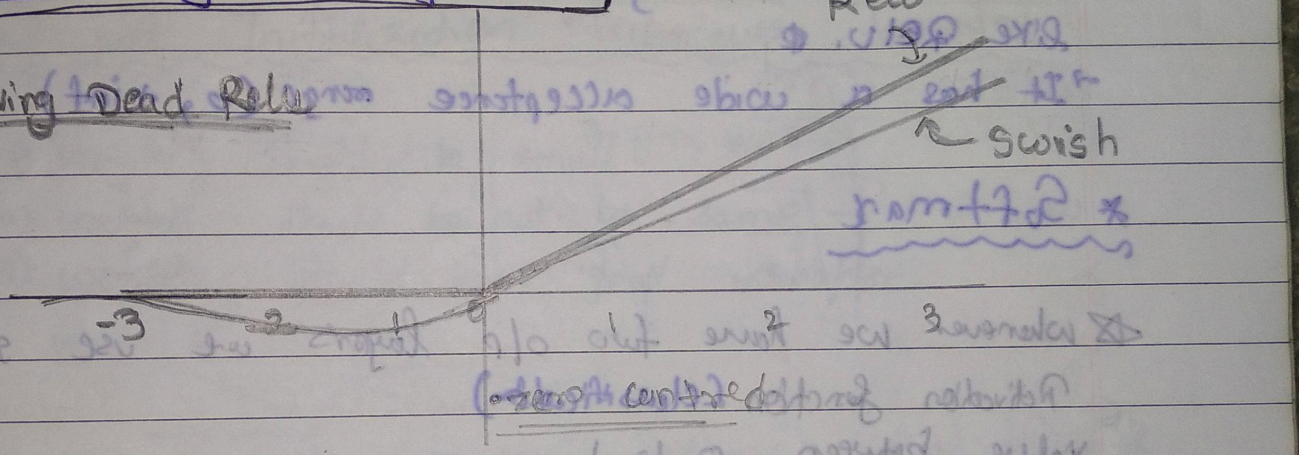
Such as a few times. when $\alpha = 0.01$ we call PRelu as Leaky Relu it is regarded as a special case PRelu it.

- Above, y_i is any input on the i^{th} channel and a_i is the negative slope which is a learnable parameter.

{
- → if $a_i = 0$, f becomes ReLU
- → if $a_i > 0$, f becomes leaky ReLU
- → if a_i is a learnable parameter, f becomes PReLU

* Swish (A self-Gated) function

formula = $y = X * \text{Sigmoid}(x)$



- Solving Dead Relu

Note → This is used in LSTM mostly
→ we use swish only when, Neural Network > 40 layer (min greater 40 layer)
→ less than 40 layer not work

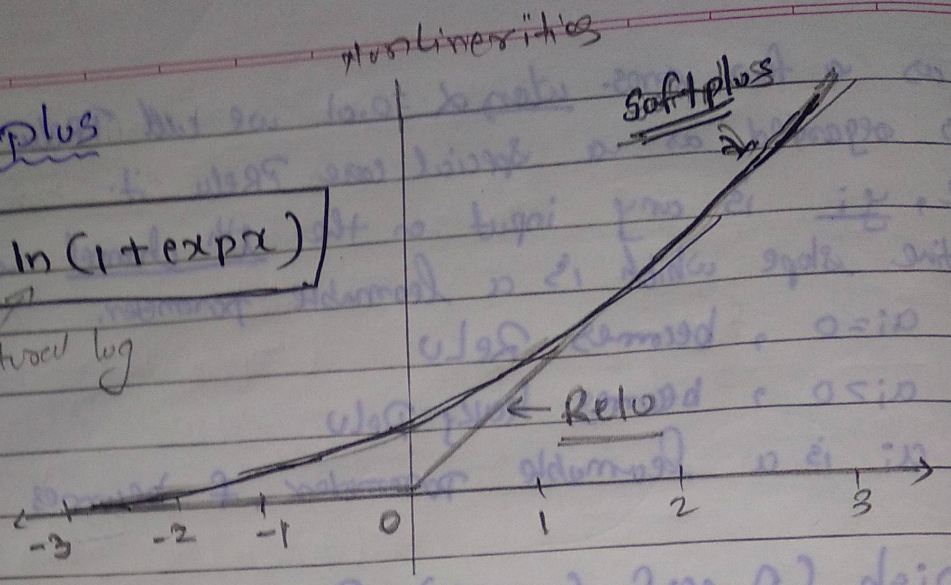
- The advantage of self-Gating is that it only requires a simple input, while normal gating requires multiple scalar inputs.

* → whenever we have Deep Neural Network, means min More than 40 layer we should use swish function.

Softplus

$$f(x) = \ln(1 + \exp x)$$

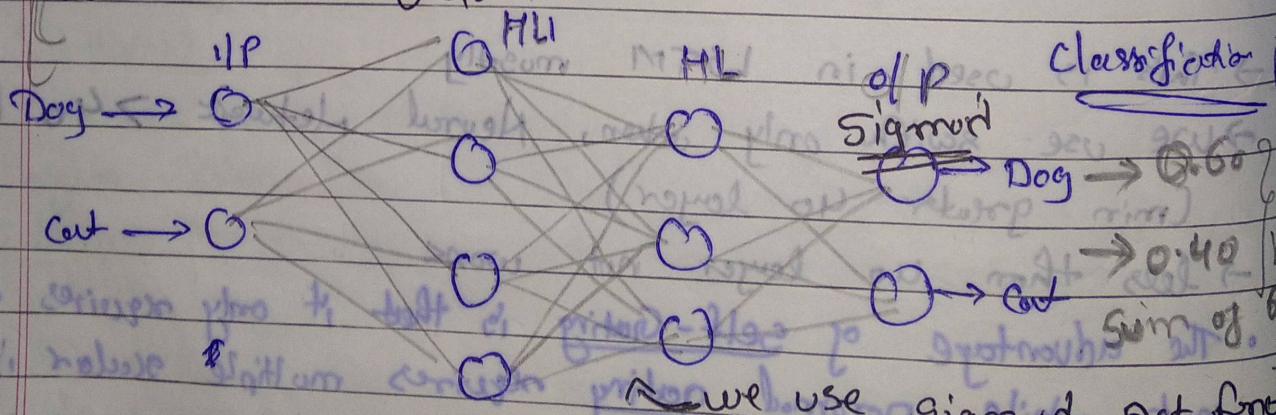
Natural log



→ The softplus function is similar to the Relu function, But it is relatively smooth. It is unilateral suppression like Relu.
→ It has a wide acceptance range (0, +inf)
→ Variant of Relu only

Softmax

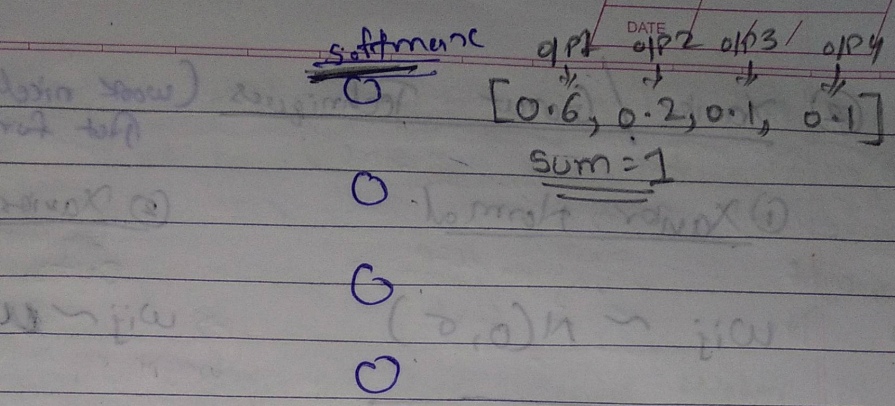
Whenever we have two o/p layers we use sigmoid Activation function (classification) value between 0 to 1



Note:- Whenever we have more than 2 o/p layer we use Softmax Activation function.

Note :- Sigmoid and Softmax Activation function are kept for Last phase

Softmax

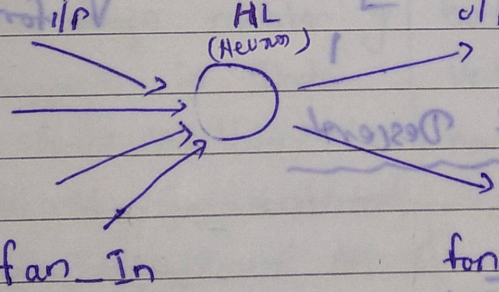


$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \quad , j = 1, 2, \ldots, K$$

$$\Rightarrow x_j = w^* i/p + b$$

* Weight Initialization Techniques in NN

Key points => 1) weights should be small
2) weights should not be same
3) weights should have good variance.



(1) Uniform Distribution Technique

→ (work nicely with Sigmoid fnct)

$W_{ij} \sim$ Uniform $\left[\frac{-1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}} \right]$ (a, b: lower, maximum)

How to Read → weight are basically selected from uniform

- weights are selected from uniform Distribution with some a and b value.

② Xavier / Glorot Techniques (work nicely with Sigmoid Act funct)

Weights are taken from N(0) with some value σ

① Xavier Normal

Normal Distribution $\mu = 0$ mean = 0

$w_{ij} \sim N(0, \sigma)$

$$\sigma = \sqrt{\frac{2}{(fan\_in + fan\_out)}}$$

② Xavier Uniform

Uniform Dist

$w_{ij} \sim U$ [Weights are taken from uniform Distribution]

$$\left[\frac{-\sqrt{6}}{\sqrt{fan\_in + fan\_out}}, \frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}} \right]$$

③ He init (work nicely with ReLu)

① He uniform

weights are taken from uniform Dist with some a and b value

$$w_{ij} \approx U\left[\frac{-\sqrt{6}}{\sqrt{fan\_in}}, \frac{\sqrt{6}}{\sqrt{fan\_in}} \right]$$

② He Normal

weights are Selected from N(0) with $\mu = 0$ and

$$w_{ij} \approx N(0, \sigma) \qquad \sigma = \sqrt{\frac{2}{fan\_in}}$$

* Stochastic Gradient Descent

weight updation

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

-ve slope

+ve slope

Global min

* Suppose we have 1000 data point

$\frac{\partial L}{\partial w_{old}}$ → n data point → Gradient Descent

at every Epoch taking only 1 data point at a time

→ 1 data point → SGD

where k is always less than n (total datapoint)

→ 1< data point → Mini Batch SGD

- **Mini Batch SGD** { In many NN like RNN, CNN we basically used this technique

$$\text{Loss} = \sum_{i=1}^{K} (y - \hat{y})^2$$

- **Gradient Descent** { In GD we require Higher Computational memory

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

- **SGD** In linear Reg we used in ML

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

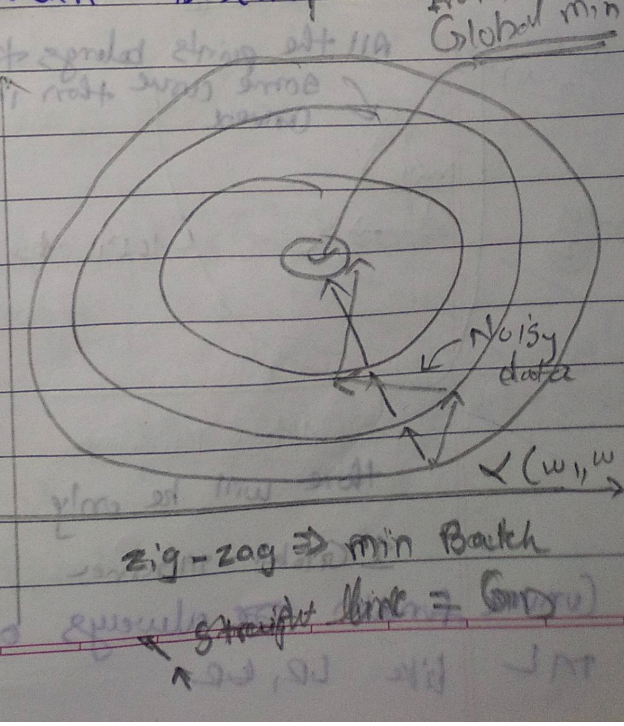
GD

→ In Gradient Descent, if there will be a huge number of records suppose in Data set you have 1 Million records so it becomes Computationally powerful and it will require a lot of resources

- GD will take less time to reach Global Min Point
- Mini Batch SGD will take some time to reach Global Min point

Mini Batch SGD

→ In case of Mini Batch SGD Based on the particular K value your updation will be happening and it will also require less number of resources when compared to the Gradient Descent



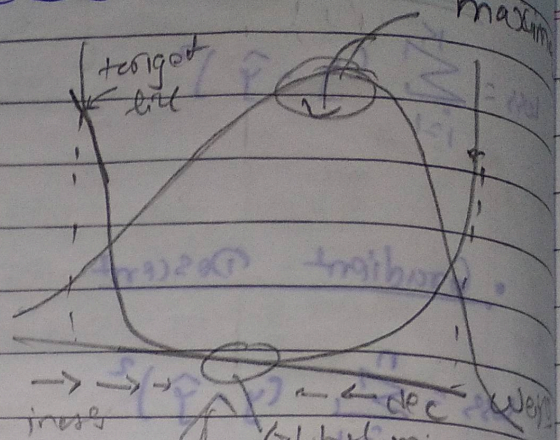
$$\left[\frac{\partial L}{\partial w_{old}}\right]_{\text{Min Batch SGD}} \approx \left[\frac{\partial L}{\partial w_{old}}\right]_{GD}$$

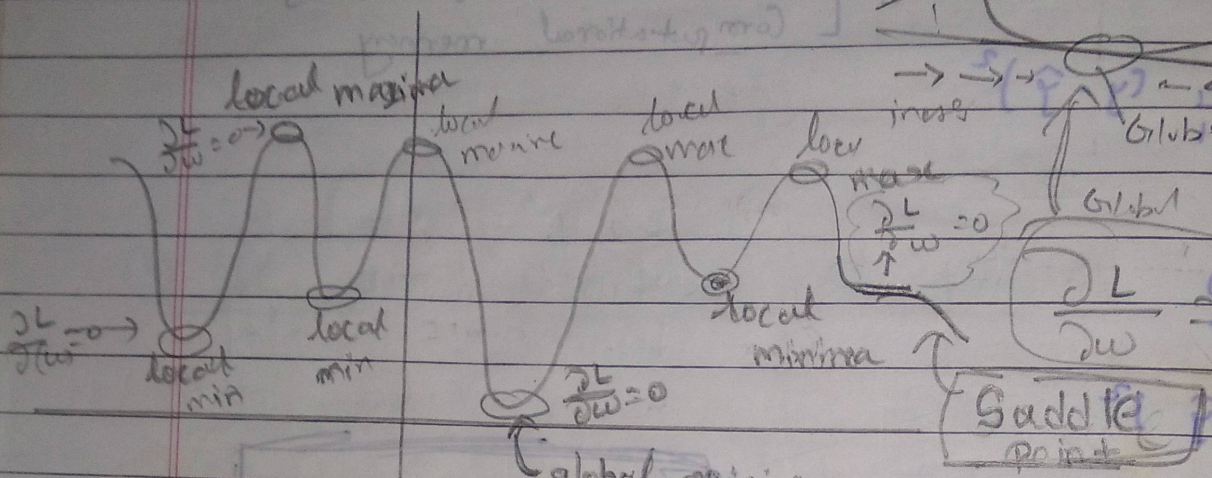
Sample — Population

* Global Minimal Vs Local Minima

optimizer → GD, SGD, Mini Batch GD

$$L(w) = \sum_{i=1}^{k} (y - \hat{y})^2$$





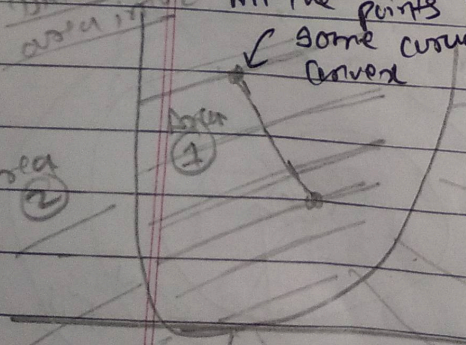
$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

(1) Convex function

Linear Reg, Logistic Reg this type of curve usually happens in ML

Consider these 2 point

All the points belongs to some curve then it is Convex

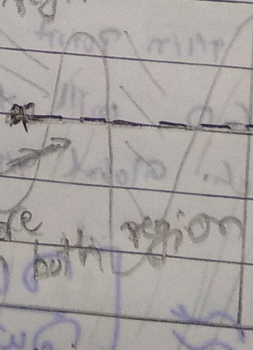


→ Global minima there will be only 1 Global minima

Convex function always occur in ML like LR, LR

(2) Non Convex function

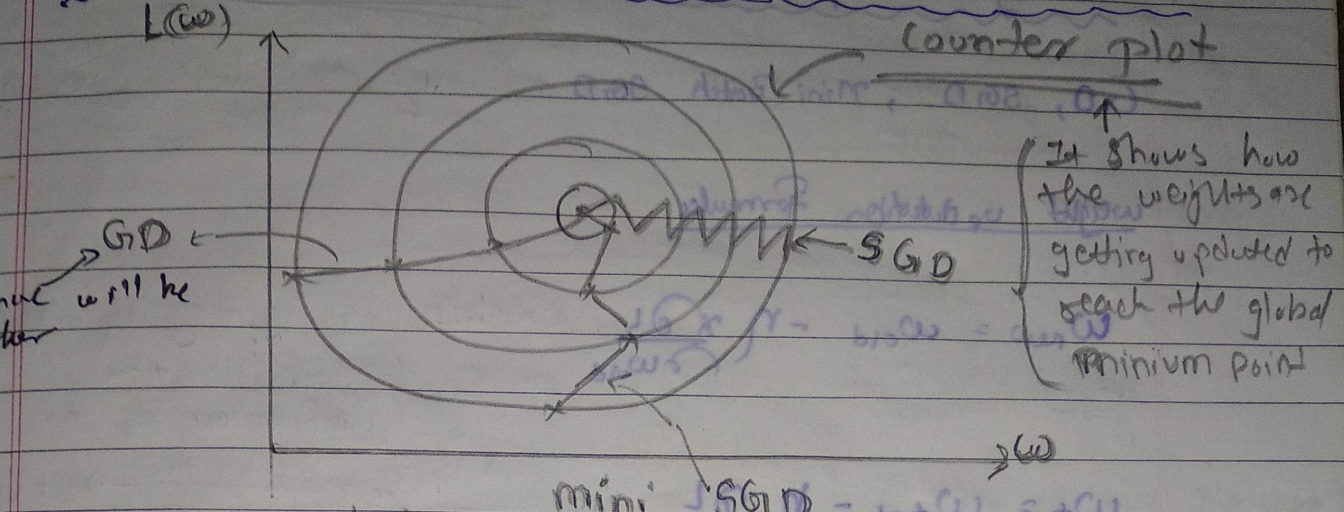
→ usually occurs in most of the DL techniques

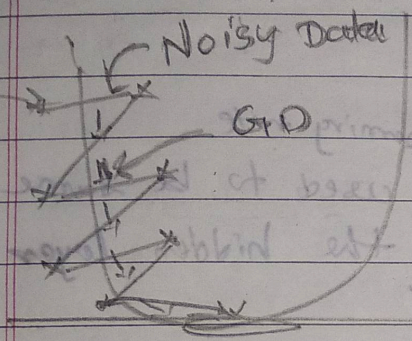


→ due to so many weights parameters so many neuron we are using + each and every neuron needs to converge

PAGE No.
DATE / /

Stochastic Gradient Descent with Momentum





GD ← smooth curve towards Global minima

- Noisy Data will take more time for convergence i.e to it take more time to reach Global minima

→ How noisy Data can be removed with the help of

Exponential weighted average

Exp Moving Avg

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

Note: Values reduce the noise as we are going towards global minima point

Momentum

Gama

$\gamma = 0.9$ appropriate value

$$w_{(t)} = w_{old} - \left[\gamma V_{t-1} + \eta \frac{\partial L}{\partial w_{old}}_{(t-1)} \right] \qquad 0 \le \gamma \le 1$$

Calculating (V_{t-1})

$$V_{t-1} = 1 \times \left[\frac{\partial L}{\partial w_{old}}\right]_t + \gamma \left[\frac{\partial L}{\partial w_{old}}\right]_{t-1} + \gamma^2 \left[\frac{\partial L}{\partial w_{old}}\right]_{t-2} + \dots$$

Exponentially increase

forward direction this Equation

Adagrad Optimizers (Adaptive Gradient Optimizers)

GD, SGD, Mini Batch SGD ← (using these) Reduce the loss function (w.r.t weight)

weight updation formula

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

weights are getting updated per epoch, or iteration

then we can say that

w_{new} we can treat as $\rightarrow w_t$ (Current)

$w_{old} \rightarrow w_{t-1}$ (Previous)

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

$t \rightarrow$ Current Time, $t-1$ → previous time

Note → In GD, SGD, Mini Batch SGD this η (learning rate) used to be same for all the Neuron and for all the hidden layer

Note → The idea behind Adagrad optimizers is that, It should be different Learning Rate

Idea behind Adagrad → (1) we can use Different Learning Rate for Each and Every Neuron and Each and Every hidden layer based on different-different iteration, weights

→ Two types of features

(1) → Dense (Most of the features will be Non-Zero)

(2) → Sparse (Most of the value are basically 0)
eg :- BoW features are in 0 and 1 or 0.1 (Decimal)

* Adagrad Equation

$$w_t = w_{t-1} - \left[\frac{\eta'}{\eta_t}\right] \times \frac{\partial L}{\partial w_{t-1}}$$

Learning rate w.r.t time

dash

- t-specifies w.r.t iteration (like previous iteration, current iteration previous to previous iteration)

$\alpha_t \uparrow \quad \eta'_t \downarrow$

η'_t ← dash $= \dfrac{\eta}{\sqrt{\alpha_t + \varepsilon}}$ ← constant learning rate initialize $\eta = 0.01$

α_t ← alpha

we update this η according to the Adagrad algo ← (As the iteration goes on)

Q) why do we required 'ε' (Episolen) value

→ If the α_t value is "0" the whole value will be 0 then η'_t will 0 then, ← will be 0

$$w_t = w_{t-1} - \left[\eta'_t \times \frac{\partial L}{\partial w_{t-1}}\right]$$

$w_t = w_{t-1}$ (Equal)

Note→ to prevent this problem we use ε (small +ve number)

ε → small +ve number

$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_i}\right)^2$ ← Here we are doing Summation for all the iterations

α_t can we write as

here α_t → will become high no because are squaring it

→ Learning rate η decrease, As the iteration goes on $\alpha_t \uparrow \quad \eta'_t \downarrow$

→ Weights Decreasing slowerly

Use :- we can see that η values changing w.r.t the features, w.r.t different layers, w.r.t iterations.

Initial → w_t

Initial it jump big step then goes on decreasing

Based on the no. of iteration w goes on decreasing

finally converge global minima point

Disadvantage ① As the no. of iteration increase ↑
At some time α_t will be very high ↑↑↑

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

To fix this disadvantage we use

* AdaDelta and RMSprop Optimizer

Adagrad $w_{(t)}^{new} = w_{(t-1)}^{old} - \eta'_t \frac{\partial l}{\partial w_{t-1}}$

learning rate should not become very very small

$\eta'_t = \frac{\eta \rightarrow (0.01)}{\sqrt{\alpha_t + \varepsilon}}$ ← small +ve

initial rate

w.r.t time
↳ because it will changing with each and every iteration

where $\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial l}{\partial w_i}\right)^2$

→ we taking ε because, when $\alpha_t = 0$, then we try to add ε value, so that anything suppose if we don't have ε the α_t will become 0,
→ So anything Divid by 0 will give your a very big error to handle this we will use ε (small +ve no.)

- $\alpha_t \uparrow \eta'_t \downarrow$

* As our NN becomes much deeper in Adagrad we use see is that the learning rate became smaller, hence the convergence become

- To handle the Disadvantage we use
  ↳ Adadelta & RMSprop

Optimizers ⇒ Update the w in the Backpropagation

→ ① GD ⇒ we take whole Dataset → ③ Mini Batch → Batch Size

→ ② SGD ⇒ 1 Data at a time

DATE / /

To prevent this $\gamma = 0.95$ Gamma

$$\eta'_t = \frac{\eta}{\sqrt{w_{avg_t} + \varepsilon}}$$

$$w_{avg_t} = \gamma w_{avg_{t-1}} + (1-\gamma)\left(\frac{\partial L}{\partial w_t}\right)^2$$

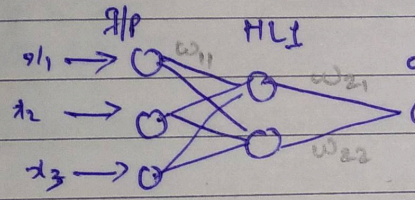
weighted average at time t

η will decrease very very slowly.

Note:- This prevent to α_t ↑↑ to become very very higher

* All Optimizers in One video - SGD with Momentum, Adagrad, Adadelta, RMSprop, Adam optimizers.

① Gradient Descent Optimizer



we calculate the error

IGD

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

→ we reduce the loss function with the help of optimizer

→ weight updation formula

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$ (Gradient Descent)

① Epoch → Complete Dataset is taken in GD

② Iteration

② SGD (convergence will be very slow) # SGD $\text{Loss} = \frac{1}{2}(y-\hat{y})^2$

Default → Taking 1 record at a time.

Note: Suppose i have 10k records,

for every Epoch, i have to do 10k Iteration of forward and Backward

for, 1 Epoch → 10k Iteration

for, 20 Epoch → 20 x 10k Iteration

How many no. of time you are training the Model

Epoch

Convergence will be in Zig-Zag

Noise

⑤ Mini Batch SGD

→ suppose we have 10k records

→ for Epoch we will take Batch size

• Taking Batch size = 1000

Noise we getting Because we are creating a Batch Data

Dataset 10k / 1000 = 10

20 Epoch

1 Epoch → 10 Iteration

Batch size

10 forward and Backward propagation

Epoch And Iteration

Suppose 10K Records

① GD Epoch 1 → forward / ← backward Iteration 1

② SGD

Epoch 1 → / ← Iteration ⇒ based on ⇒ No. of records

③ Mini batch SGD ⇒ (Batch size)

Epoch ⇒ Iteration ⇒ $\frac{10k}{\text{batch size}} = \frac{10,000}{1000}$

• for every Epoch, for 10k records there will be 10 iteration

PAGE No.
DATE / /

GD with momentum

Implementation details

Initially $V_{dw} = 0$ $V_{db} = 0$

On iteration t inside Epoch

compute dw, db on the current Mini-batch

moving Exponential average

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

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

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

$$b_t = b_{t-1} - \eta * V_{db}$$

→ (smoothing of convergence)

* Adam Optimizer (Adaptive Moment Estimation)

⇒ Here we change η

Adam combine :- (i) Momentum (for smoothing)

(ii) RMS prop (Learning Rate) → Changing Learning Rate in efficient manner so that α_t value does not go High

Implementation detailed

Initially → $V_{dw} = 0$ $V_{db} = 0$ $S_{dw} = 0$ $S_{db} = 0$

On iteration t

(i) Compute $\frac{\partial L}{\partial w}$ (weight), $\frac{\partial L}{\partial b}$ (bias) using current Mini-batch

(ii)
$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) \frac{\partial L}{\partial w}$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1) \frac{\partial L}{\partial b}$$
} w.r.t momentum

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) \left(\frac{\partial L}{\partial w}\right)^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) \left(\frac{\partial L}{\partial b}\right)^2$$
} w.r.t RMS prop

PAGE No.
DATE / /

$$w_t = w_{t-1} - \frac{\eta * V_{dw}^{correction}}{\sqrt{S_{dw}^{correct} + \varepsilon}}$$

initial h

correction

Exp moving avg

$\propto$ Value

$$b_t = b_{t-1} - \frac{\eta * V_{db}^{correction}}{\sqrt{S_{db}^{correction} + \varepsilon}}$$

These 2 form is the Updation with Adam Optimizer

Bias Correction

$$V_{dw}^{correction} = \frac{V_{dw}}{(1-\beta_1^t)}$$

$$V_{db}^{correct} = \frac{V_{db}}{(1-\beta_1^t)}$$

Bias correction w.r.t Momentum

$$S_{dw}^{correct} = \frac{S_{dw}}{(1-\beta_2^t)}$$

$$S_{db}^{correct} = \frac{S_{db}}{(1-\beta_2^t)}$$

PAGE No.
DATE / /

* Loss function (Differ types)

- **Loss function** = Is basically only for 1 record for forward and backward propagation

$$\text{Loss fun} \Rightarrow \frac{1}{2}(y-\hat{y})^2$$

- **Cost function** = we are passing **Batch size** of records

$$J = \frac{1}{t}\sum_{i=1}^{t} \frac{1}{2}(y-\hat{y})^2 \quad \ldots t = \text{batch size}$$

(1) Squared Error loss	(MSE) Mean Squared error loss
$L = (y-\hat{y})^2$	$\sum_{i=1}^{t} (y-\hat{y})^2$

- **Advantages of Squared Error loss**

(1) It is in the form of Quadratic Equation
ax^2+bx+c

(i) Whenever, Plot the Quadratic Equation, we will get a Gradient Descent with only Global Minima
(ii) we dont get any local Minimas

(2) The MSE loss penalizes the model for making large error by squaring them

- **Disadvantage Mse loss function**
→ It is not robust to outliers

(1) **Loss function in Regression** (Adv & Dis)

(1) Squared Error loss

$$L = (y-\hat{y})^2 \xrightarrow{\text{convert to Cost function}} J = \frac{1}{t}\sum_{i=1}^{t}(y-\hat{y})^2 \quad \ldots t = \text{Batch size}$$

② Absolute Error Loss

local minima

→ $L = |y - \hat{y}|$ ← absolute value / modulus → $J = \frac{1}{t}\sum_{i=1}^{t} |y - \hat{y}|$

MAE

Adv (i) It is more robust to outlier as compared to MSE

MSE

Dis (i) Computation is very difficult

③ Huber Loss

↳ Combine both (i) MSE, (ii) MAE

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

Quadratic Equation — Linear Equation — Hyper parameter

⇒

② Classification

④ Cross entropy → used in → Logistic Regression specifically

log loss ⇒ $\text{Loss} = -y^A \log(\hat{y}) - (1-y) * \log(1-\hat{y})$ ← Binary cross entropy

for computing $\hat{y}$ we use sigmoid

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

$$\hat{y} = \text{sigmoid} = \frac{1}{1+e^{-z}}$$

PAGE No.
DATE / /

⑤ Multiclass Cross Entropy Loss

$$L(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \log(\hat{y}_{ij})$$ (categorical cross entropy)

i = row number

↳ Here y_i is one hot encoded target vector

o/p → multiclass

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

Good, Bad, Neutral ⟶ One hot Encoding.

Suppose i have features

f1	f2	f3	o/p	← Bits
2	3	4	Good →	[1 0 0]
4	5	6	Bad →	[0 1 0]
7	8	9	Neutral →	[0 0 1]

{ internal it will be in Bits

f1	f2	f3	y_{i1}	y_{i2}	y_{i3}
			y_1	y_2	y_3
2	3	4	1	0	0 → 3 bits
4	5	6	y_{21}	y_{22}	y_{23}
7	8	9	y_{31}	y_{32}	y_{33}

vector

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

$\hat{y}_{ij}$ ⇒ Softmax activation function [multiclass classification]

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

$$z = w^T x + b$$

DATE

• Keras ⊙⊙ Loss Function

→ Probabilistic losses ← (Classification most common)

(i) Binary Crossentropy class [internally they will be doing One hot Encoding]

↳ Specifically used for Binary classes

o/p $\hat{y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(ii) Categorical Crossentropy class ← [we will get probability of each and every class]

we have o/p as 3 categories

o/p $\hat{y} = [0.3 \quad 0.2 \quad 0.5]$ [which will be high we will be taking that particular class]

(iii) Sparse Categorical Crossentropy class

Here will be get Index as a o/p

↳ 0 1 2

$[0.3 \quad 0.2 \quad 0.5]$

$\hat{y}$ ∈ will be maximum probability index [2]

↳ internally

* Environments