DEEP LEARNING FUNDAMENTALS

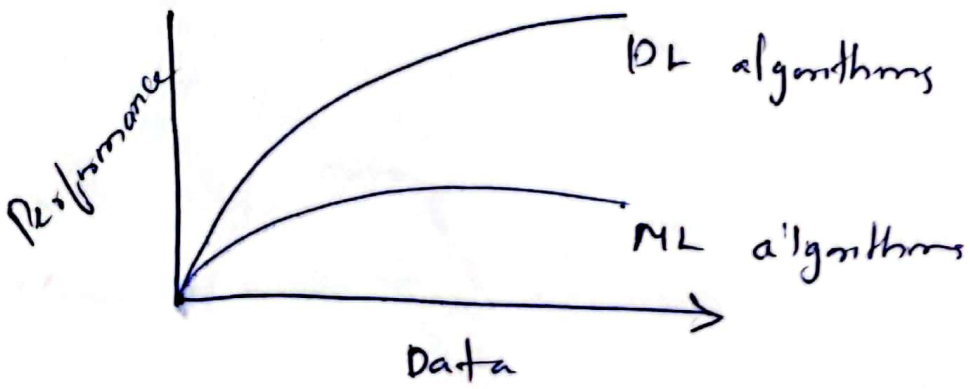
Deep Learning

1) ANN → Artificial Neural N/w → classification
                                → Regression

2) CNN → Convolutional Neural N/w → I/p → Images/videos frames
                                    ↓
RCNN, Masked RCNN
Detection, Yolo V5, V6, V7

3) RNN → Recurrent Neural N/w → NLP
I/p: Text, Time Series
↓
{Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder, Decoder, Transformers, BERT}

Framework: Tensorflow

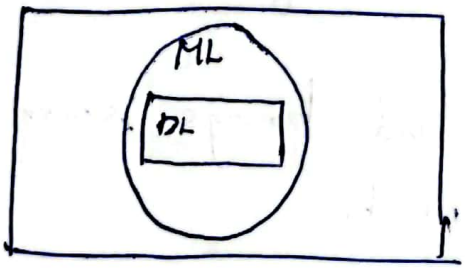


Deep learning

Day 1

Agenda:

1) Deep learning: Perceptron. {AI v/s ML v/s DL v/s DS}
2) Forward Propagation
3) Backward propagation
4) Loss function
5) Activation function
6) Optimizers



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

eg: 1) Netflix App
2) Self driving Cars
3) Amazon Application
4) Sofia

ML: provides stats tools to analyze the data, visualize the data, predictions, forecasting - clustering.

DL: Mimic Human brain

[The final goal of Datascientist is to create an AI application]

Q) Why DL becoming so popular?

1. In 2005, → ORCUT

After that, → Facebook, Instagram, whatsapp, Linkedin, Twitter.

Because of this data started to get generated exponentially.

2. Concept of Bigdata was very high in demand.

3. Company had huge amount of data.

⇓

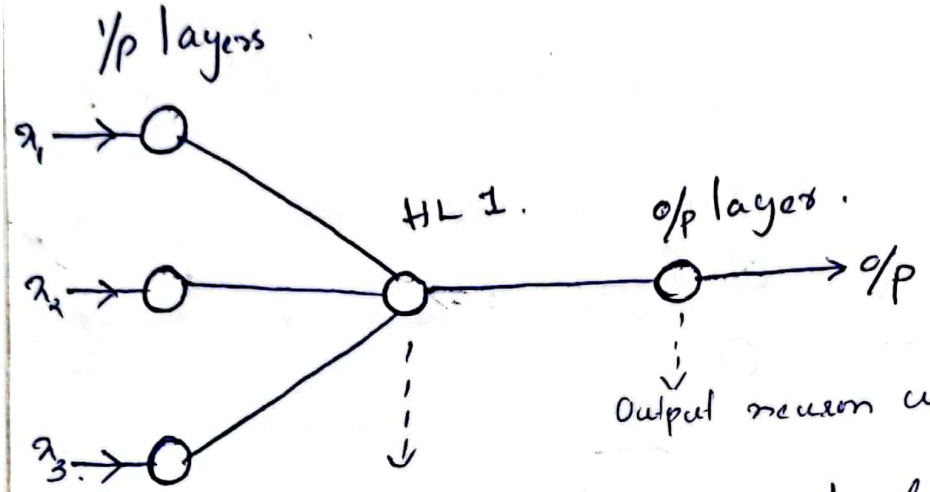
use this data and brings seamless effect on their product.

⇓

AI became so popular.

Q) Hardware Advancement. (NVIDIA) → GPU's → (Training the model)

Perceptrons : Single layer and Multilayer Neural N/w

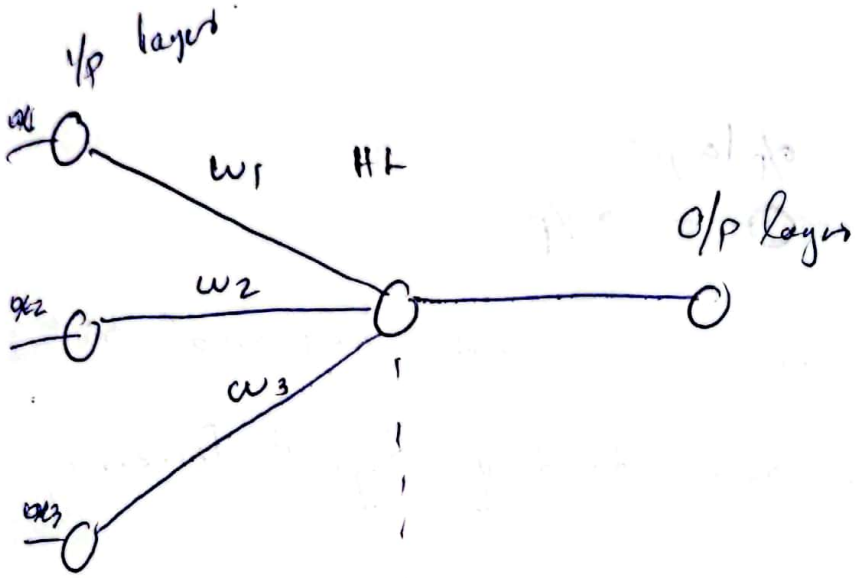


Dataset

(x_1) Study	(x_2) Play	(x_3) Sleep	Pass/Fail
7	3	7	1
2	5	8	0
9	3	7	1

Whenever we create a neural n/w from scratch we need to train them based on the o/p.

When the signal is getting passed from the i/p layers, what is happening?



As soon as the signal goes to the HL, assigning different weights.

(1) $\sum x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$

$= w^T x$

(2) Pass the above function to an activation function

Imp

Weights will actually helps the neurons to act whether it should be activated or what level it should be activated

For weights, we initialize it as zero then $\sum x_i w_i$ will be zero.

So Inorder to avoid that we introduce a Parameter called bias

If by mistake we initializes the ~~wait~~ weight as zero, bias will have some value so that some value it will no equal to zero so we can continue the training and later can update the weights.

(Bias initialization will automatically happen by neural n/w) bias will required in every HL.

Then in step 2,

we pass an activation function on top of y where $y = \sum x_i w_i$

Activation function

It is a function that calculates the o/p of a node based on the ip. input it receives.

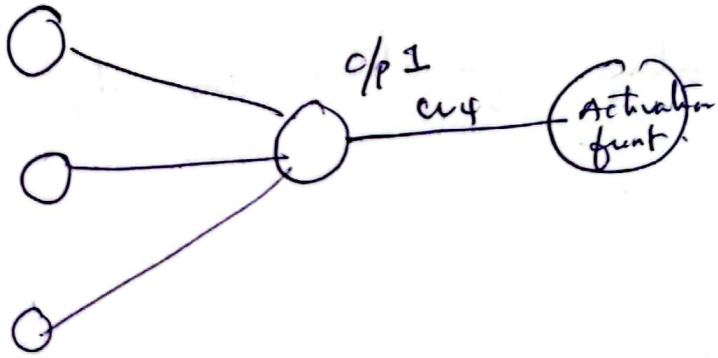
Sigmoid Activation function (used for binary classification)

$$\text{Sigmoid} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-(\Sigma x_i w_i + \text{bias})}}$$

Output of the equation will be zero or one.
Or it gives values b/w 0 and 1.

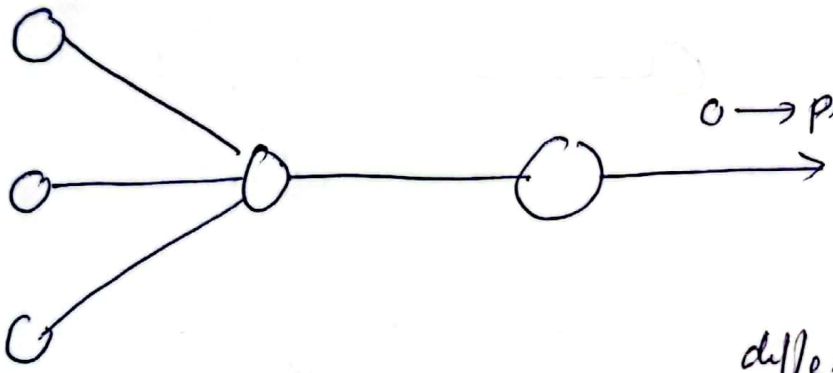
if ≥ 0.5 considered as 1
< 0.5 considered as 0.

Then we go the next layer, another weight will be assigned and next activation function will be applied and finally will be able to get the o/p.



we basically took the i/p multiplied by weights. added a bias and activated.

So the entire process from the beginning till the end is called Forward Propagation



o → predicted Value $\hat{y}$.

y : actual Value

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

The difference should be at near to zero

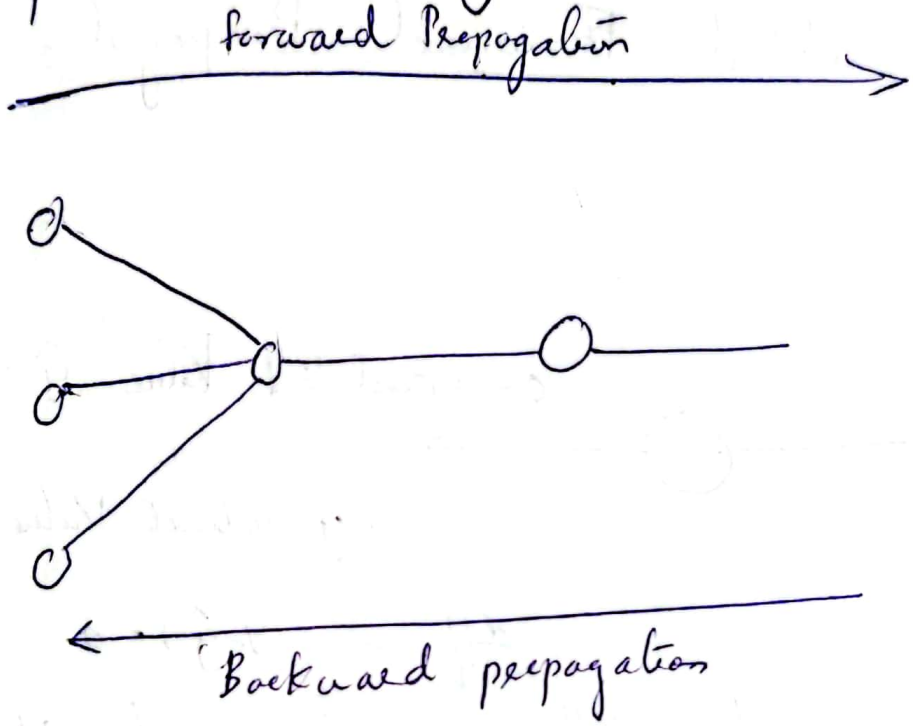
Loss function

It finds out/measures the discrepancy b/w the predicted outcomes and the actual targets. The primary goal of a loss fund is to guide the optimization process towards reducing the error.

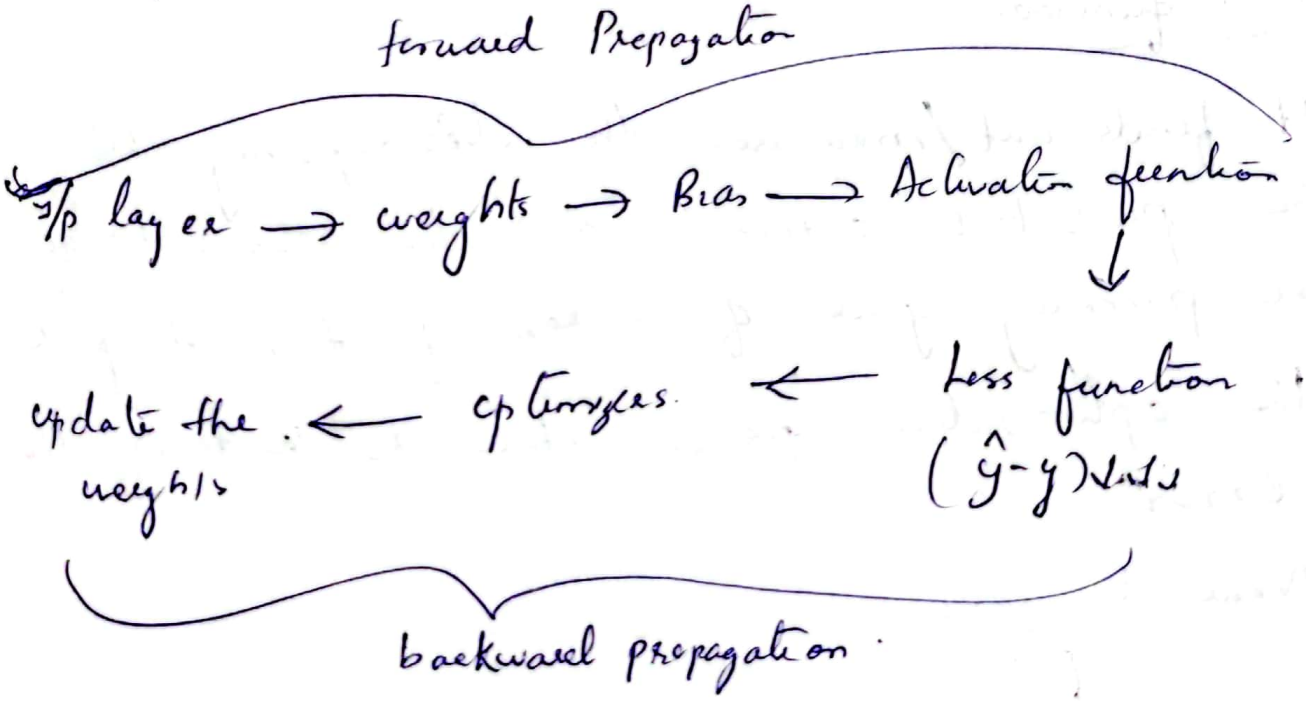
Near to zero.

then we do an important process known as backward propagation.

the main aim of backward propagation is to update the weights.

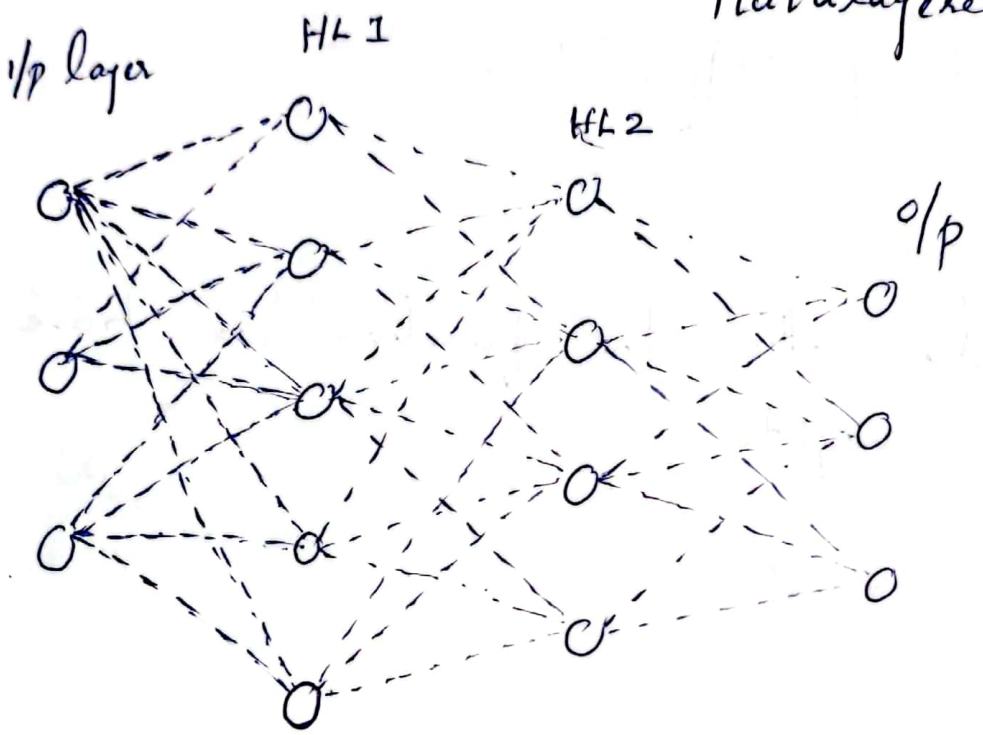


An Optimizer.



@ ANN, CNN, RNN, object detection will works on forward and backward propagation.

Multilayered Neural N/w



Neural N/w are for mimicing the human brain

Day 2

Agenda:

1. Forward Propagation
2. Chain rule of derivatives
3. Vanishing Gradient Problem
4. Loss functions

Backward propagation

1. Weight updation formula
2. chain rule of Differentiation

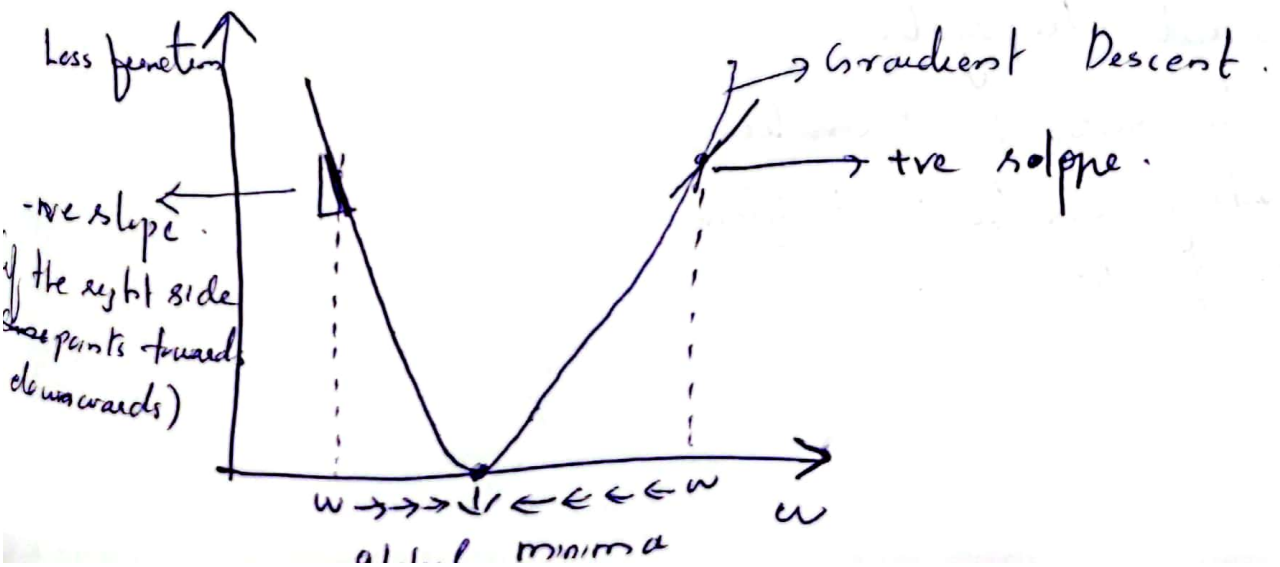
inorder to reduce the loss function we have to update the weights.

Weight Updation formula

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

$\eta \rightarrow$ learning rate.

$\frac{\partial L}{\partial w_{old}} \longrightarrow$ slope.



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

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

$\therefore w_{new} > w_{old}.$

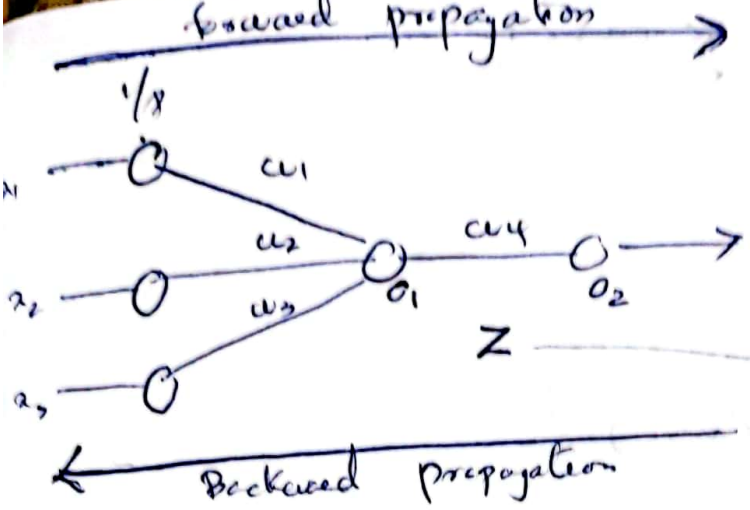
(+ve case)

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

$\therefore w_{new} < w_{old}.$

- Here the main aim is to come to the global minima.
- Learning rate (η) should be a small number, so that are slowly converging to the global minima
- If η = larger number, there may be chance that it will never reach global minima.
- Usually $\boxed{\eta = 0.001}$ or $\boxed{\eta = 0.01}$

Chain Rule of differentiation



$$\left[w_{4\,new} = w_{4\,old} - \eta \frac{\partial L}{\partial w_{4\,old}} \right]$$

Loss is dependant on o_2 — Chain rule of derivative.

So, $$\frac{\partial L}{\partial w_{4\,old}} = \frac{\partial L}{\partial o_2} \times \frac{\partial o_2}{\partial w_4}$$

$$Z = \sigma (o_1 w_4 + bias)$$

it is nothing but activation function applied on $\underbrace{o_1 w_4 + bias}$

$$\boxed{b_{2\,new} = b_{2\,old} - \eta \frac{\partial L}{\partial b_{2\,old}}}$$ bias updation formula.

$$w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_{1old}}$$ → for updating w_1

[Loss] dependant on [O_2] dependant on [O_1] dependant on [w_1] dependant ~~on~~ ~~w_4~~

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

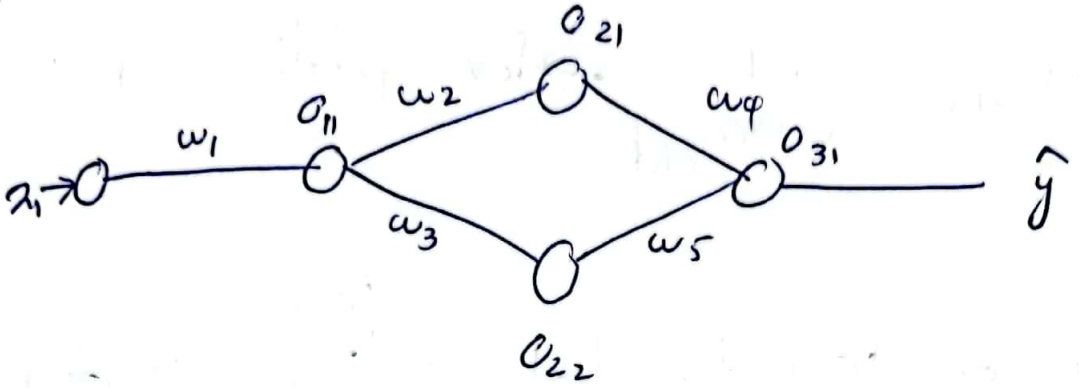
$$\frac{\partial L}{\partial w_{2old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_{2old}}$$

both can also be written as

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial w_4} \times \frac{\partial w_4}{\partial O_1} \times \frac{\partial O_1}{\partial w_{1old}}$$

and

$$\frac{\partial L}{\partial w_{2old}} = \frac{\partial L}{\partial O_2} \times \frac{\partial O_2}{\partial w_4} \times \frac{\partial w_4}{\partial O_1} \times \frac{\partial O_1}{\partial w_{2old}}$$



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

$$\frac{\partial L}{\partial w_{1old}} = \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_4} \times \frac{\partial w_4}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_2} \times \frac{\partial w_2}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}\right].$$

$$+$$

$$\left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial w_5} \times \frac{\partial w_5}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_3} \times \frac{\partial w_3}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{1old}}\right]$$

can written as

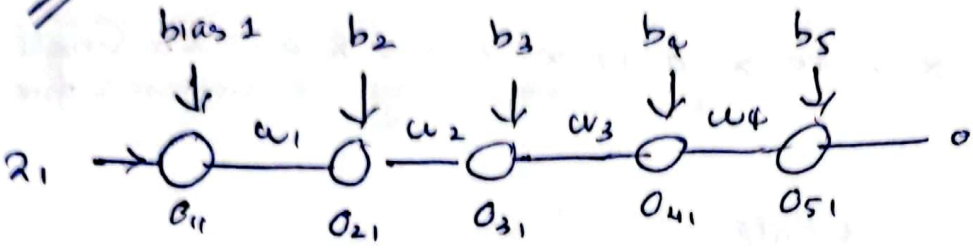
$$\left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial a_{1old}}\right]$$

$$+$$

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

V.Imp

Vanishing Gradient Problem



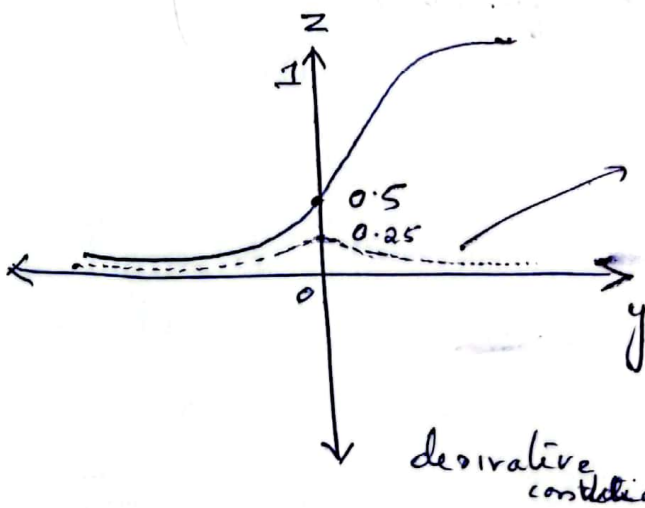
$$\text{Loss} = \frac{1}{2}(y - \hat{y})^2 : \text{MSE}$$

Sigmoid activation functions

a) Properties of Sigmoid activation function ?

a) Loss function M.S.E. ?

Sigmoid Activation function



$$z = \frac{1}{1+e^{-x}}$$

$$0 \le \sigma(y) \le 0.25$$

The derivative curve of sigmoid function will be ranging between **0 and 0.25**

Because of this,

Since derivative is b/w 0 and 0.25.

for eg: $0.25 \times 0.15 \times 0.10 \times 0.05 \times 0.02 =$ Small value.

then,

$$W_{new} = W_{old} - \eta \times (\text{small number})$$

(η: small)

$\Rightarrow W_{new} \approx W_{old} \Longrightarrow$ Vanishing Gradient Problem.

No change in weights.

How do we basically solve this?

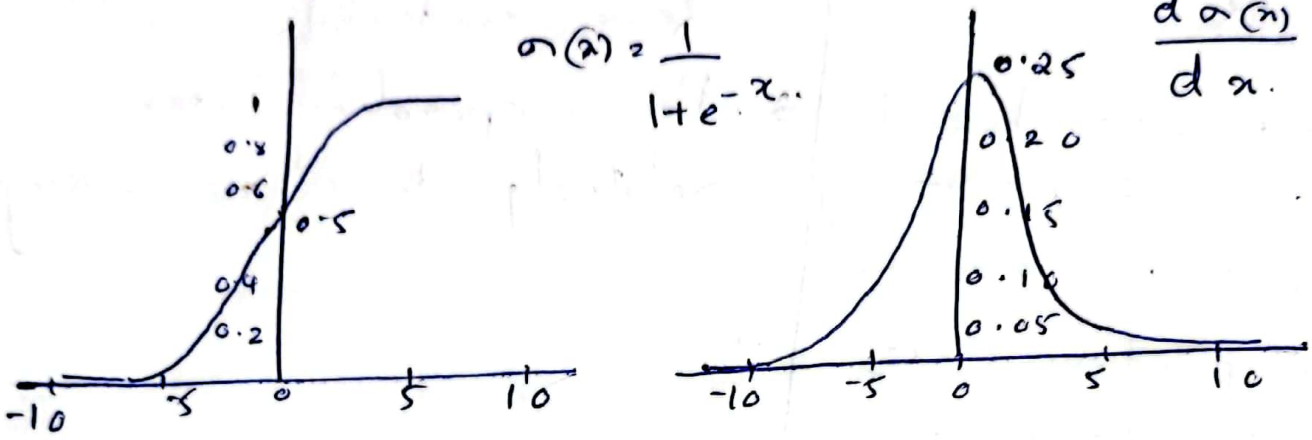
Use another activation functions

Activation functions

1) Sigmoid
2) Tanh
3) ReLu
4) Leaky ReLu
5) Pre ReLu.

Commonly Used Activation functions

1. Sigmoid functions / Logistic function



- It is the most frequently used activation function in the beginning of the Deep learning. It is a smoothing function that easy to derive.

- In the Sigmoid function, we can see that its o/p is in the open interval (0, 1).
It can be thought of as the firing rate of a neuron. In the middle where the slope is relatively large, it is the sensitive area of the neuron. On the sides where the slope is very gentle, it is the neuron's inhibitory area.

Advantages

1. Smooth gradient, preventing "jumps" in o/p values.
2. Output values bound b/w 0 and 1, normalizing the o/p of each neuron.
3. Clear predictions ie, very close to 1 or 0.

Disadvantages

1. Prone to gradient Vanishing
2. Function o/p is not zero centered
3. Power operation are relatively time consuming.

Defects

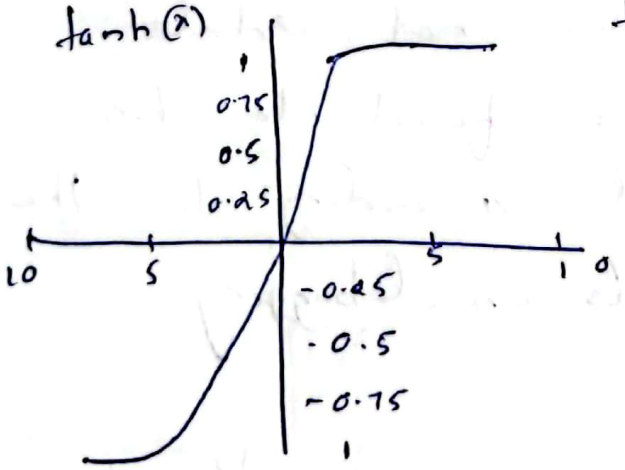
1) When the i/p is slightly away from the coordinate origin, the gradient of the function becomes very small, almost zero. In the process of neural n/w backpropagation, we all use the chain rule of differential to calculate the differential of each weight w. When the back propagation passes through the sigmoid funct the differential on this chain is very small. Moreover, it may pass through many sigmoid functions, which will eventually

cause the weight to do have little effect on the loss function, which is not conducive to the optimization of the weight. This problem is called gradient saturation or gradient dispersion

2) The function o/p is not centered on 0 (origin (0,0)), which will reduce the efficiency of weight update.

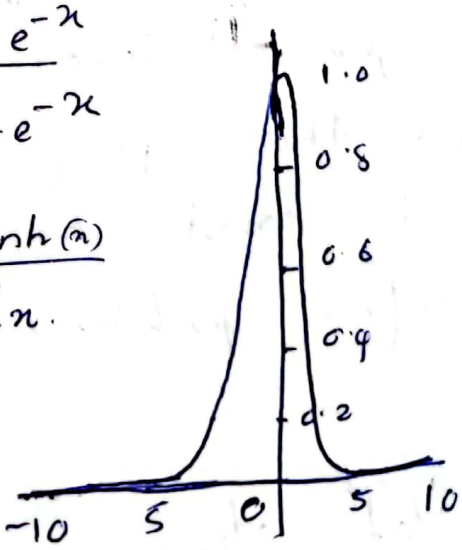
3) The Sigmoid function performs exponential operations, which is slower for Computers.

2. tanh function (Hyperbolic Tangent function)



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

$$\frac{d\tanh(x)}{dx}$$



Tanh is a hyperbolic tangent function. The curves of tanh function and sigmoid are relatively similar.

- Range -1 to 1.
- Derivative range 0 - 1.

First of all, when the i/p is large or small, the o/p is almost smooth and the gradient is small, which is not conducive to weight update. The difference is the o/p interval

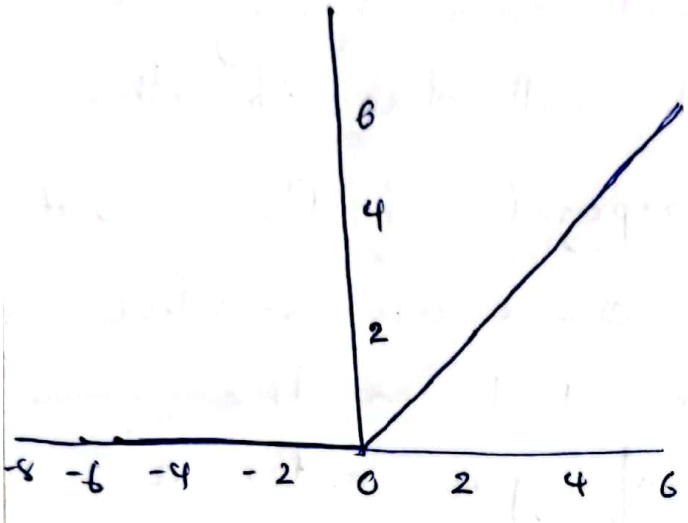
The o/p of interval of tanh is -1 to 1 and the whole function is 0-centered which is better than sigmoid

In general binary classification Problems, the tanh function is used for the hidden layers and sigmoid function is used for the o/p layers. However, these are not stable, and the specific activation funct to be used must be analysed according to the specific problem, or depends on debugging.

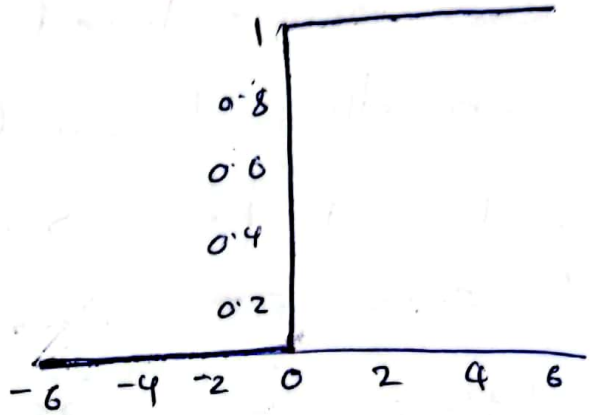
Yet there will be chances of Vanishing Gradient Descent.

3. ReLU function/Rectified Linear unit

ReLu = Max(0, x)



$\frac{d\,ReLu(x)}{dx}$



derivate o/p → 0 or 1.

⇒ The ReLu funct is actually a funct that takes the Max value. Note that this is not fully interval-derivable, but can take sub-gradient as shown in the above fig,

⇒ Comparing to Sigmoid & tanh it has the following advantages.

1. When the i/p is positive, there is no gradient saturation problem
2. The calculation speed is much faster. The ReLu funct has only a linear Relationship. Whether it is forward or backward, it is much faster than Sigmoid and tanh

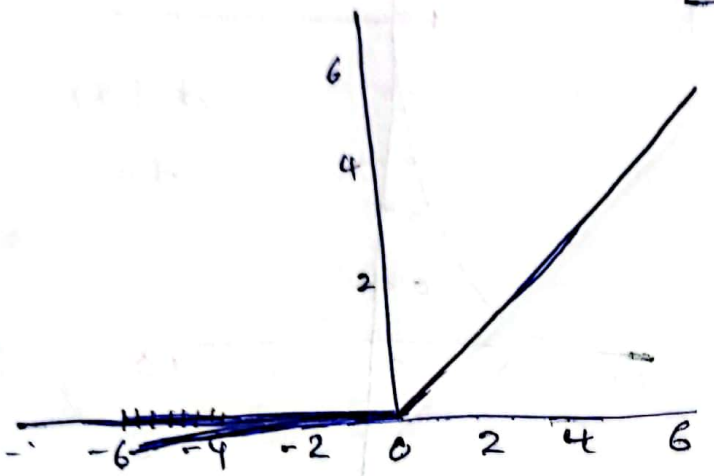
Disadvantages

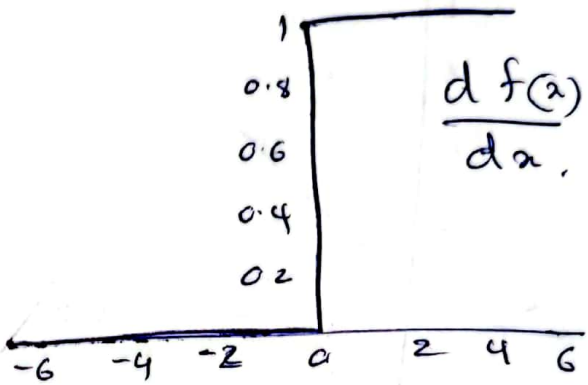
① When the i/p is -ve, ReLu is completely inactive, which means that once a negative number is entered, ReLu will die. In this way, in the forward propagation process, it is not a problem. Some areas are sensitive and some are insensitive. But in the backpropagation process, if you enter a negative number, the gradient will be completely zero, which has the same problems as the sigmoid functions and tanh funct.

② We find that the o/p of the ReLu funct is either 0 or a positive number, which means that the ReLu function is not a 0-centric funct.

4. Leaky ReLu function

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

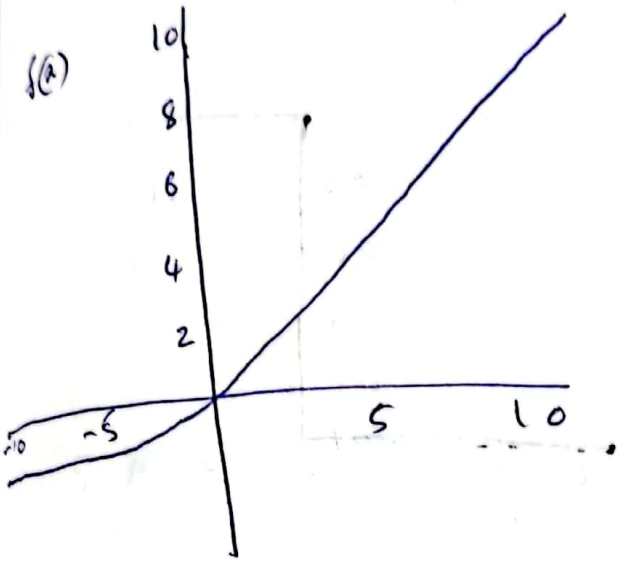


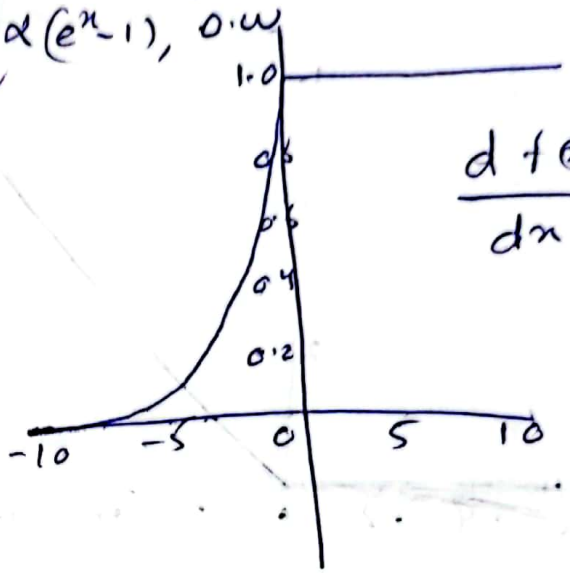


- Inorder to Solve the Dead ReLu Problem, Peple proposed to set the first half of ReLu $0.01x$ instead of 0. Another intuitive idea is a Parameter-based method, Parametric ReLu: $f(x) = \max(\text{alpha}\_x, x)$, which alpha can be learned from back propagation.
- In theory Leaky ReLu has all the advantages of ReLu, plus there will be no problems with Dead ReLu, but in actual operation it has not been fully proved that Leaky ReLu is alway better than ReLu

5. ELU (Exponential Linear Units) function

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





ELU is proposed to solve the problem of ReLU. Obviously ELU has all the advantages of ReLU.

- No Dead ReLU Issues
- The mean of the o/p is close to 0, zero centered.

The small problem is that it is slightly more computationally sensitive (because of exponential value in the f(x)). Similar to Leaky ReLU, although theoretically better than the ReLU; there is currently no good evidence in practice that ELU is always better than ReLU.

6. Softmax

$$S(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, 2, \ldots K.$$

- for arbitary real vector of length K, Softmax can compress it into a real vector of length K with a value in the range (0, 1) and the sum of the elements in the vector is 1.
- It also has many application in Multiclass classification and neural n/w. Softmax is different from the normal max function; the max function only outputs the largest value and Softmax ensures that smaller values have a smaller probability and will be be discarded directly. It is a 'max' that is 'soft'
- The denominator of the Softmax function combines all factors of the original o/p value, which means that the different probabilities obtained by the Softmax function are related

to each other in the case of binary classification. for Sigmoid.

these are,

$$P(y=1/x) = \frac{1}{1+e^{-\theta^T x}}.$$

$$P(y=0/x) = 1 - P(y=1/x) = \frac{e^{-\theta^T x}}{1+e^{-\theta^T x}}.$$

For Softmax with $K=2$, these are,

$$P(y=1|x) = \frac{e^{\theta_1^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{1}{1+e^{(\theta_0^T - \theta_1^T)x}} = \frac{1}{1+e^{-\beta x}}.$$

$$P(y=0|x) = \frac{e^{\theta_0^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{e^{(\theta_0^T - \theta_1^T)x}}{1+e^{(\theta_0^T - \theta_1^T)x}}$$

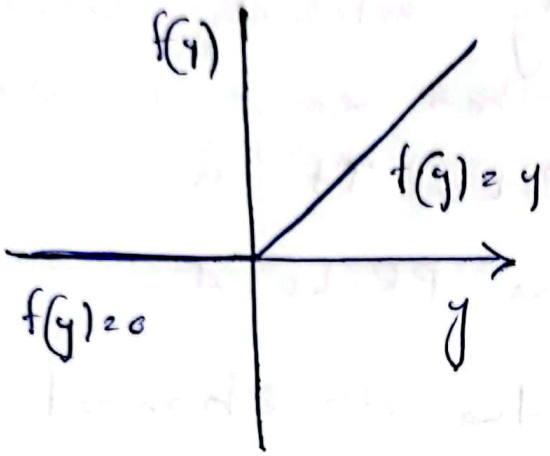
$$= \frac{e^{-\beta x}}{1+e^{-\beta x}}.$$

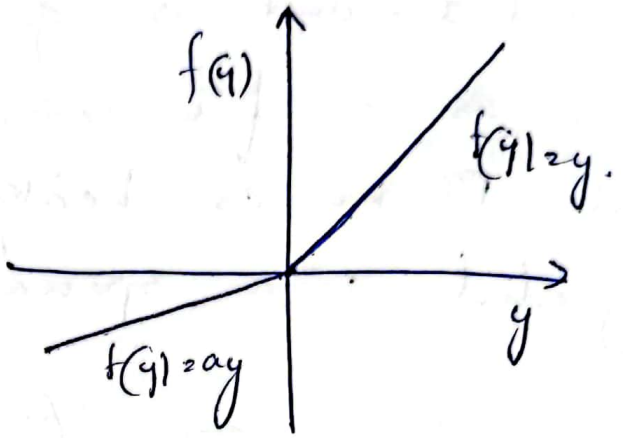
Among them: it can be seen that

$$\beta = -(\theta_0^T - \theta_1^T)$$

In the case of binary classification, Softmax is degraded to Sigmoid.

7. PReLU (Parametric ReLU)





ReLU V/S PReLU

For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

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 advantage.

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

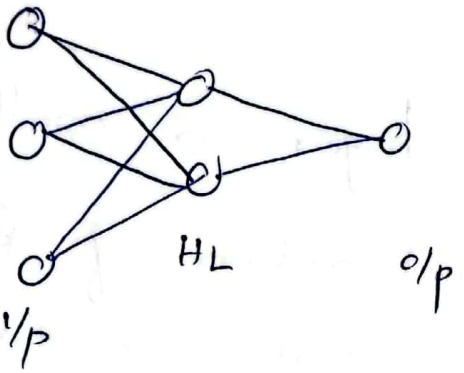
We look at the formula of PReLU. The parameter α is generally a number b/w 0 and 1, and is generally relatively small such as a few zeroes. When $\alpha = 0.01$ we call PReLU as Leaky ReLU. It is regarded as a special case PReLU.

Above y_i is any i/p on the ith channel 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 }

Imp Techniques on which activation functions are should use ?

If Suppose you have binary classification, You have a neural n/w

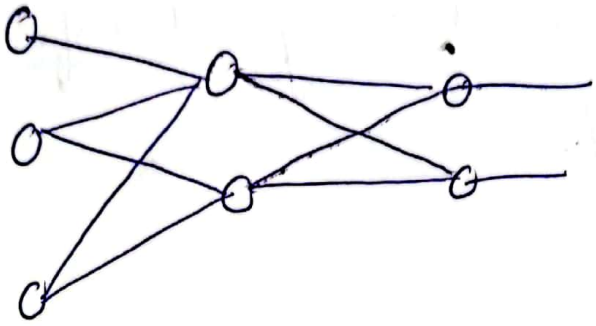


- In the hidden layer always try to use ReLu Activation funt.
- In o/p layer, Sigmoid activation function.

Nb: If using ReLu, convergence is not happening then change ReLu to PReLu or ELU

Imp But the o/p must have Sigmoid for binary classifications.

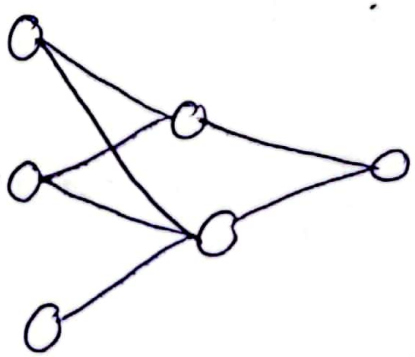
Suppose You have multiclass classification Problem.



- Use **Softmax** for o/p layer in multiclass.
- Can Use ~~ReLU~~ the combinations of ReLU, PReLU, ELU if the convergence is not happening.

Suggested : ReLU

In Case of Regression Problems,



- In the hidden layer use any variant of ReLU.
- In the o/p layer use Linear Activation funct

- There will be separate loss function for this.

Loss functions

For Regression

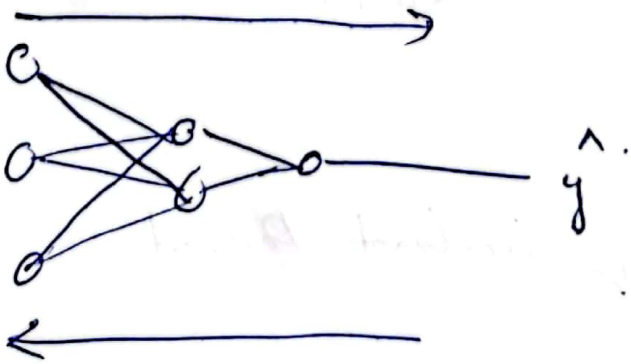
In Deep learning specifically ANN, we typ solve two types of Problems classification and regression

In Regression Problems,

1. MSE (Mean Squared Error)
2. MAE (Mean Absolute Error)
3. Huber loss.
4. RMSE

Epochs, Batch size?

Loss function and Cost function



Suppose I have 100 records, If 1 Pass

each records separately and do the forward and backward propagation at that time. So instead I pass a batch of records for eg: 10.

So difference b/w Loss funct & Cost funct is

Loss function	→ Specifically for Single record
Cost function	→ Specifically for batch of records.

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

Cost function : $\frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y_i})^2$; n is the batch size

1. Mean Squared Error (MSE)

Loss funct

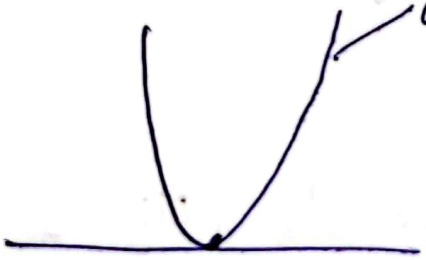
$\frac{1}{2n}(y-\hat{y})^2$

↓

Quadratic equation.

Cost funct

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



Advantages

1) Differentiable.
2) It has only 1 local or Global Minima
3) It converges faster

Disadvantages

1) It is not Robust to outliers.

If I have some outlier.

The reason it changes because we are penalizing the error. Since we are squaring the error and it will have major shift

2. Mean Absolute Error (MAE)

Loss funt

$$\frac{1}{2}|y-\hat{y}|$$

Cost function

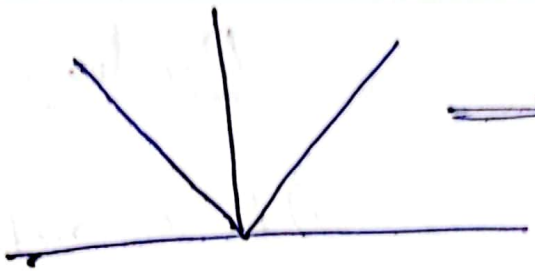
$$\frac{1}{2}\sum_{i=1}^{n}|y_i-\hat{y}_i|$$

Advantages

1. Robust to outliers.

Disadvantages

1. Time consuming >MSE.

looks like  ⟹ Subgradient

3. Huber loss.

It is a Combination of MSE and MSE.

~~Cost function~~

Loss function (outliers are not Present)

$\frac{1}{2}(y-\hat{y})^2$ if, $|y-\hat{y}| \leq \delta$

$\delta|y-\hat{y}| - \frac{1}{2}\delta^2$, o.w.

↓

Hyperparameter.

Cost function

$\frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$ if $|y_i-\hat{y}_i| \leq \delta$

$\delta|y-\hat{y}| - \frac{1}{2}\delta^2$, o.w.

4. Root Mean Squared Error (RMSE)

Loss function

$$\sqrt{\frac{(y_i - \hat{y}_i)^2}{n}}$$

Cost function

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

Advantages

- Penalize the large errors
- Optimizes According to MSE
- Differentiable
- Respects zero values.

Disadvantages

- Sensitive to outliers
- Doesn't consider error relative to actual value
- Not Robust to outliers

for Classification

CROSS ENTROPY

- Binary CROSS Entropy → binary class
- Categorical CROSS Entropy → multi class
- Sparse Categorical Cross Entropy → multi class

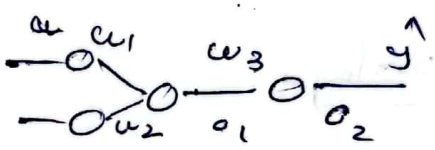
1. Binary Cross Entropy.

y: Actual Value
$\hat{y}$: Predicted Value

$\Downarrow$ log loss.

$$\text{Loss} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y}) \Rightarrow \text{Logistic Regression}$$

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



How do we calculate $\hat{y}$?

$\hat{y} \Leftarrow o_2 = \sigma(o_1 * w_3 + \text{bias})$

$$\hat{y} = \frac{1}{1+e^{-x}}$$, sigmoid activation funct in last layer.

2. Categorical CROSS ENTORY (Multiclass classification Problem)

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

one hot encoding →

Good	Bad	Neutral
1	0	0
0	1	0
0	0	1

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

i = row
j = column

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

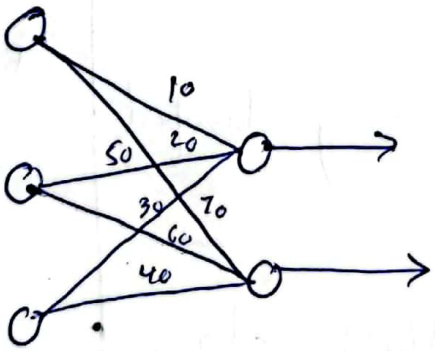
c: no of categories.

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

$\hat{y}_{ij}$ = Softmax Activation function

(It is applied on o/p layer of multiclass classification Problem)

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



For $z_1 = 10$ $\quad \frac{e^{10}}{e^{10+20+30+40+60+70}}$ will be probability.

Total will be 1.

3. Sparse Categorical Cross Entropy

0 1st 2nd → categories

[0.2, 0.3, 0.5

⇓

2nd Index ⟹ o/p.

0 1st 2nd 3rd 4th

[0.2, 0.3, 0.1, 0.2, 0.2] ⟹ 1st Index.

Category → o/p.

Disadvantage

1) Loose info about the probability of other category

* Right Combinations.

Hidden layers.	o/p layer	Problem Statement	Loss function
1) ReLu or it's variant	Sigmoid	Binary Classification	Binary Cross Entropy
2) ReLu or it's variant	Softmax	Multi class	Categorical CE or Sparse CE
3) ReLu or it's variant	Linear	Regression	MSE, MAE, Huber Loss, RMSE.

The Sparse categorical Cross Entropy loss function computes the cross-entropy loss b/w

the integer labels and the predicted Probabilities. It is suitable for scenarios where there are two or more label classes, and the labels are provided as integers rather than one-hot representations.

Usage

This loss function is particularly useful when working with classification tasks that have multiple classes and where labels are represented as integers. It expects integer labels for each sample and computes the CE loss based on these integer labels.

Difference b/w Categorical CE and Sparse Categorical CE

Categorical CE	Sparse Categorical CE
• Requires label to be one-hot encoded, where each label is represented as a binary vector with a single "1" indicating the class	• Utilizes integer-encoded labels where each label is represented by an integer value corresponding to the class.

Ideal for Scenarios where each sample belongs to only one class, making it suitable for mutually exclusive classes.

(2) Suited for problems where samples can belong to multiple classes or when dealing with soft probabilities.

3 Mathematical formulation

The loss function calculates the cross entropy b/w the true one-hot encoded labels and the predicted probabilities

(3) Mathematical formulation:

The loss function computes cross entropy with integer targets, optimizing based on the difference b/w Predicted Probabilities and integer labels

TensorFlow → google Keras, PyTorch.
PyTorch → facebook.

Day 3

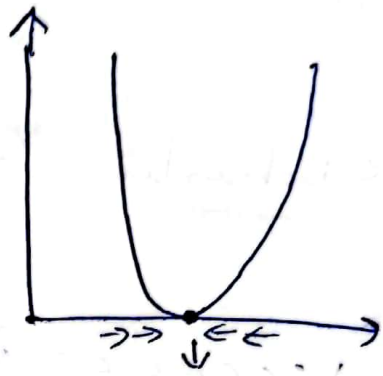
1) Optimizers
   1. Gradient Descent
   2. SAD (Stochastic Gradient Descent)
   3. Mini Batch SAD
   4. SGD with momentum
   5. Adagrad - Adaptive Gradient
   6. RMSPROP
   7. Adam optimizers

2) Batch
3) Epochs
4) Iterations.
5) ANN practical.

~~GRT~~ Optimizers

1. Gradient Descent

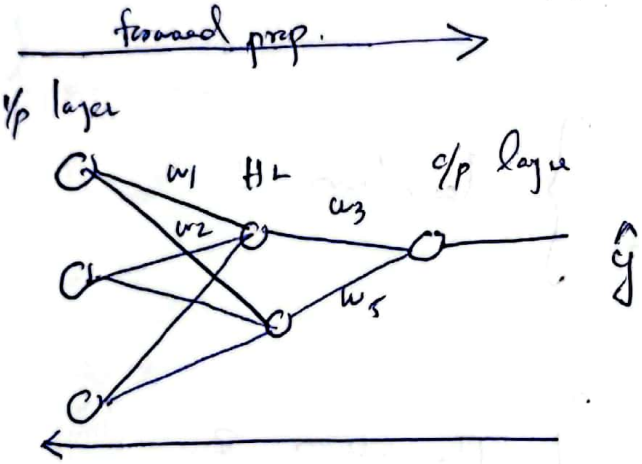
Loss/cost



Weight Updation Formula.

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

η → Learning rate.



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

Epochs

A combination of forward and backward propagation

Epochs, Iteration

Dataset = 1000

1 Epoch { → ← weights will get updated.

1 Epoch = 1 iteration

$\hat{y}_i$ ⇒ cost function

100 iteration, batch size = 10

2nd Epoch { → ←

100 Epoch { → ←

millions of records in every epochs,

Advantages

Convergence will happen.

Disadvantages

① Resource Extensive {required huge RAM to process million records.

2. Stochastic Gradient Descent

1000000s of records.

1 record ⟶ $\hat{y}$ ⟵ Loss fun ⟵ update weights } Iteration 1

2 record ⟶ $\hat{y}$ ⟵ Loss fun ⟵ update weights } Iteration 2.

⋮

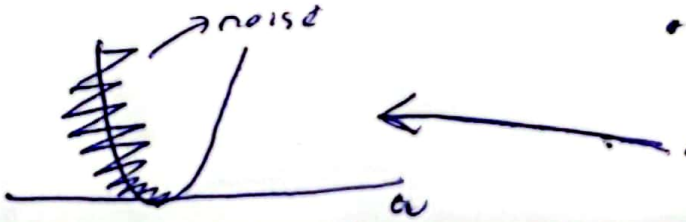
millionth record ⟶ $\hat{y}$ ⟵ loss fun ⟵ update weights } iteration 1000000

Advantages

- Solve Resource Issue.

Disadvantage.

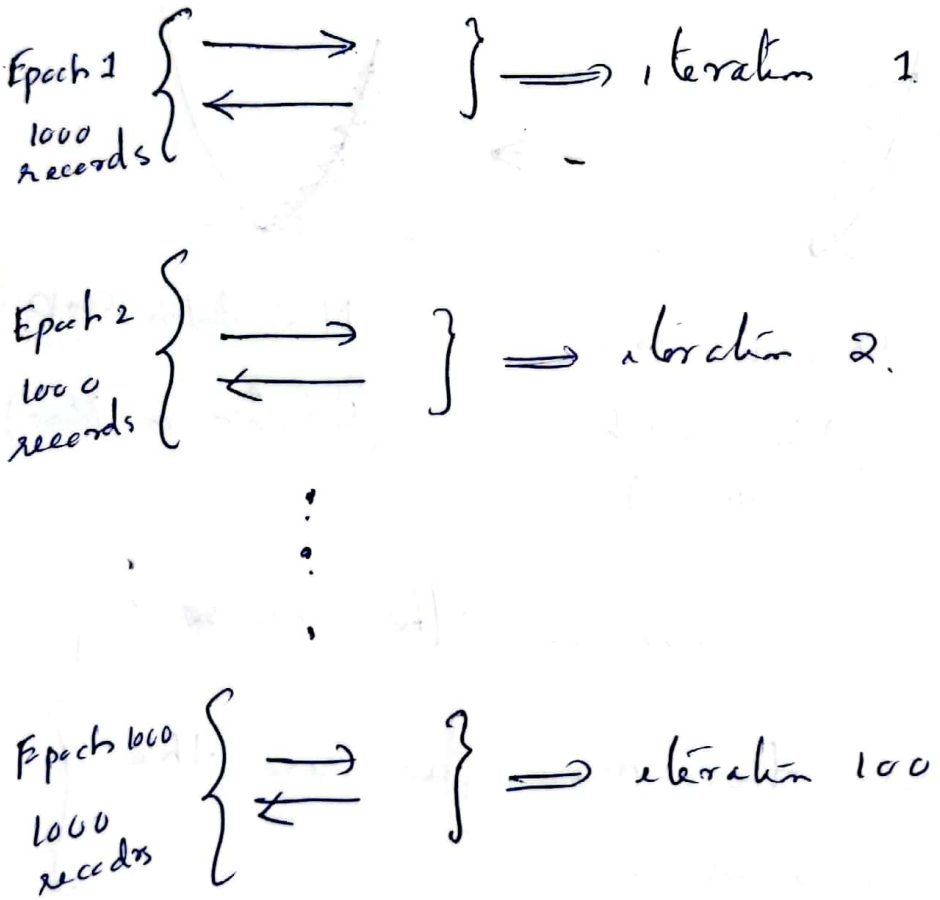
- Time Complexity ↑↑
- Convergence will also take more time
- Noise gets introduced.



3. Mini Batch SGD

Instead of passing single records we pass batch of records.

If we have 1000000 of records and batch size = 1000. Then for every iteration,



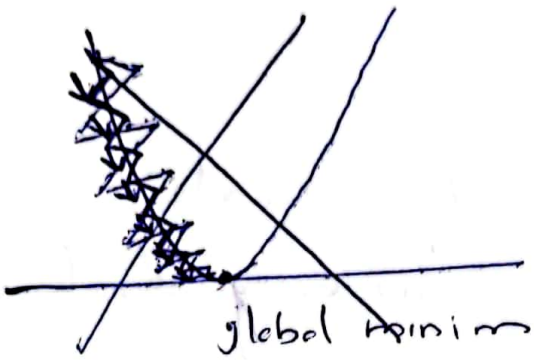
Advantages.

1. Convergence Speed will increase
2. Noise will be less when compared to SGD
3. Efficient Resource Usage (RAM)

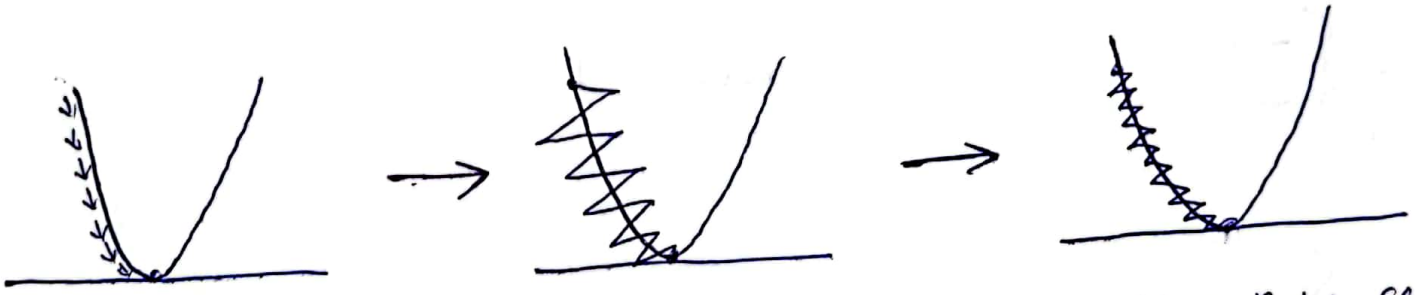
Disadvantages

1. Noise Still exists

4. Time Complexity will ~~increase~~ improve.



Main aim is to reach the global minima.



Gradient Descent → Stochastic GD (Noise will be there) → Mini Batch SGD. (Reduction in Noise)

So how do we remove the noise?

In order to remove the noise we use the concept called Momentum

4. SGD with Momentum.

Min Batch

Momentum will smoothen the gradient and reduce the noise.

Exponential Weighted Average.

⇓

Time Series

⇓

ARIMA, ARMA, ...

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

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

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

Exponential Weighted Average:

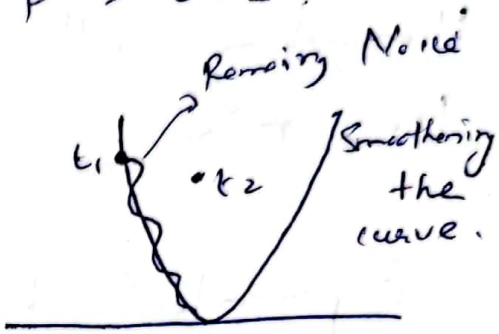
time	t_1	t_2	t_3	t_4	...	t_n
Value	a_1,	a_2	a_3	a_4	...	a_n

Value at time $t_1 = a_1$

$\beta \rightarrow$ hyperparameter

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

$\beta \rightarrow 0 - 1$.



$\beta = 0.95$

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

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

$$= 0.95 (0.95 \times a_1 + 0.05 \times a_2) + 0.05 \times a_3.$$

Where do we apply this? (Exponential Weighted Average)

We apply this on the derivate of loss wrt derivate of weights.

Exp. Weighted Avg

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

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

$$\therefore V_{dw} = \beta \times V_{da_{t-1}} + (1-\beta) * \frac{\partial L}{\partial w_{t-1}}$$

Advantages.

- Reduce the noise
- Working on Mini Batch
- Quicker convergence

Interview Qn

If there is so much noise in the Gradient Descent. what should ge do?

ans: Tryyy to bring the smoothening factor in the optimizer.

Imp

5. Adagrad → Adaptive Gradient Descent

η → Helps us to maintain the Speed of the convergence: learning rate

? How can we change the value of the learning rate while we move towards the global minima.

In previous optimizers the learning rate is fixed. So here instead of fixing the learning rate, we need to make it adaptive as we move towards the global minima.

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

$$\Downarrow$$

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

$\eta' = \dfrac{\eta}{\sqrt{\alpha_t + \epsilon}}$ → learning rate

→ small number to avoid the divide by zero condition. ie, the denominator never becomes zero.

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

t → current time stamp, (the weights at that particular time stamp)

Main aim to decrease the learning rate as we move to Global Minima.

As time goes,

$$\alpha t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w t}\right)^2 \uparrow\uparrow\uparrow.$$

because it is the Summation from i=1 to i=t.

As α_t value ↑ , η' ↓

Then there is possibility that αt value will be huge.

Because of this in $\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$ there will be negligible change.

~~5. Ada Delta and~~ Disadvantage

$\eta' \rightarrow$ Possibility to become a very small value $\simeq 0$

So we need to ensure that α_t should reach a huge value.

6. AdaDelta and RMSProp

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

(Dynamic LR + Smoothening EWA)

{We are going to apply exponential} weighted avg

$S_{dw} = 0$ ↓

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

if $\beta = 0.95$

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

Here we can control the increase of S_{dw}.

Hence there will be slow decrease in η' and will be able to reach the global minima.

Whenever we combine the exponential weighted avg and the adaptive learning rate. We get,

7. Adam Optimizer

Best Optimizers

Momentum + RMS Prop (Adaptive Learning Rate)

We have V_{dw}, S_{dw}.

V_{db} S_{db}

initially $V_{db} = 0$ $S_{db} = 0$.

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

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

$$\eta' = \frac{\eta}{\sqrt{S_{dw}} + \epsilon} \qquad V_{dw} = \beta \times V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$$

EWA

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

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

⇒ Momentum Smoothening

$$\eta' = \frac{\eta}{\sqrt{S_{d_{wt}} + \epsilon}}$$

$$S_{d_{wt}} = 0.$$

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

- It solves problem of smoothening
- Make sure that learning rate becomes adaptive

Adam Optimizer

1) It is an optimizer which combines the best properties of AdaGrad and RMSProp.
2) It is based on adaptive estimation of first order and second order moments, making it computationally efficient and memory efficient.

3) Adam Adjusts learning rates individually for each parameter leading to more efficient optimization, especially in complex models with many parameters.

Day 4-

1. ANN Practical Implementation
2. Early Stopping
3. Black box Models V/s White Box Models
4. CNN Introduction.

In Case of ANN

Interview Question

For which all algorithm feature scaling is required ?

ans: ANN ? ✓ Linear ? ✓ Log ? ✓ Decision Tree ? Not necessary RF ? Not Necessary

XGboost ~~Not~~ Necessary K Means ✓

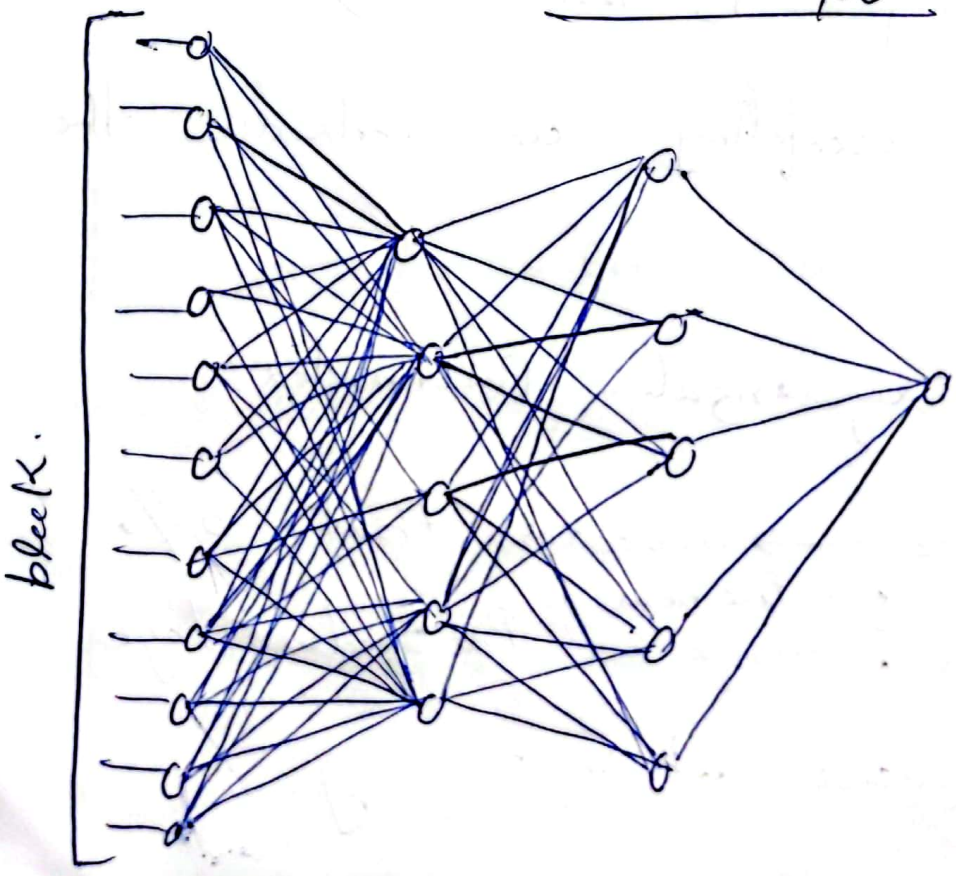
KNN ✓

⇒ Anything related to Distance based, we need to do feature scaling.

⇒ Wherever gradient Descent / optimizers are used, scaling will be required.

1. Sequential
2. Dense
3. Activation

Neural N/w



→ If I take this entire neural n/w together, as a block → Sequential.
And can done forward and backward propagation.

→ Whenever we want to create neurons we use dense

With the help of dense → we will be able to create HLs, I/p layers, O/p layers.

⇒ Dropout

Sometime the entire neural n/w leads to overfitting (Training acc ↑, Test acc ↓)

To reduce the overfitting we introduce the dropout layers.

It is like a regularisation parameter.

If dropout = 0.3 → indicates that 30% of data present, of entire neurons inside a particular layer, will get deactivated while training.

Q) How many number of epochs should we basically stop?

While I was faced with overfitting, used the dropout = 0.2 to reduce the overfitting in each HL.

Black Box Model V/s White box Model

Random Forest → Black Box Model

Decision Tree → ~~Recession~~ for White Box Model

ANN → Black Box Model

Xgboost → Black Box Model

Linear Regression → White box Model.

In Some Algorithms the internal working and all are difficult to monitor.

People ~~with~~ came up with solutions to monitor each model, how the model is performing and details about each i/p feature → that is where explainable AI comes up.

White Box Model	Black Box Model
• They are transparent and interpretable, allowing users to understand how predictions are made.	• Black box models prioritize innovation and accuracy but lack transparency in how predictions are generated.
• Suitable for scenarios requiring high accountability and trust, like risk assessment, robotics, and scientific research.	• Commonly used for complex scenarios with deep, non-linear interactions b/w data.
• Utilizes models with higher transparency and interpretability such as linear models or decision tree.	• Employ techniques like deep learning, boosting models, and random forests.
Pros	Pros
• Rising in Popularity due to increasing mistrust of	• High Predictive accuracy, making them indispensable for tasks like Speech

Opaque AI Systems.

- Provides reliable Prediction and is suitable for scenarios where understanding the model's decision making process is crucial.

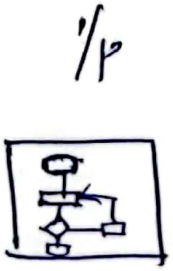
recognition and fraud detection.

- Effective at handling large datasets with complex relationship

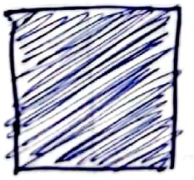
Cons

- May not Produce ground-breaking results or innovative ideas.
- Less suitable for modelling complex relationships, potentially leading to lower accuracy

- Low algorithms interpretability making it challenging to explain decisions to stakeholders
- Results may not be Perfectly reproducible due to the complexity of the model



White box Testing



Black Box testing

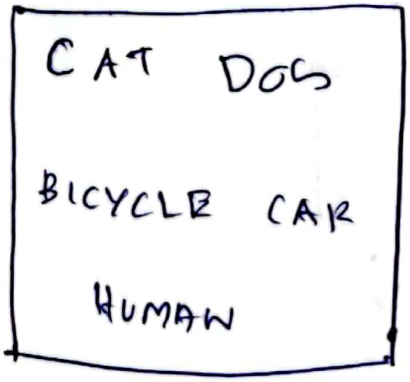
Day 5

Convolutions Nueral N/w.

- Images / videos or Video frames
- Object detection

Agenda

1) CNN v/s Human Brain
2) Convolution Operation.
   - → Convolution
   - → Padding
   - → Stride
   - → Filters (Kernels)
3) Max-Pooling
4) Flattening
5) Practical.



Major Part of Brain → Cerebral Cortex.

↓

visual Cortex (Part)

It has many layers

V_1 → Moving object

V_2 → Animal (CAT)

V_3 → Map the environment

V_4

V_5

V_6

V_7

↓

O/P

Similarly in case of CNN.

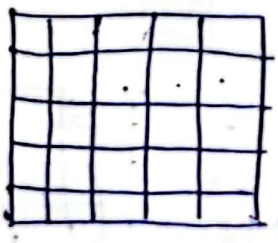
CNN.

1. Convolution:

Images ? B & W, RGB

↓

1 channel



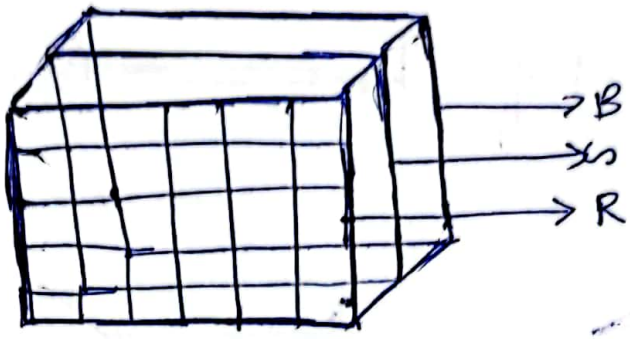
0-255

0 - black

255 - White

Every Pixel may have different values based on composition.

WRT RGB,

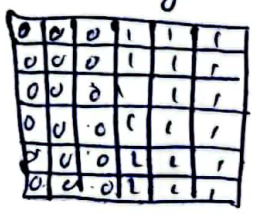


0-255

5 x 5 x 3

Convolution

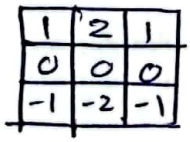
Image.



6x6

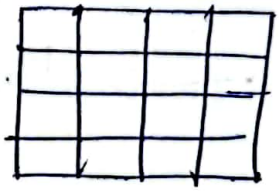
minmax scaling,
Converting all the values of the Pixels to 0, 1

filter/Kernel.



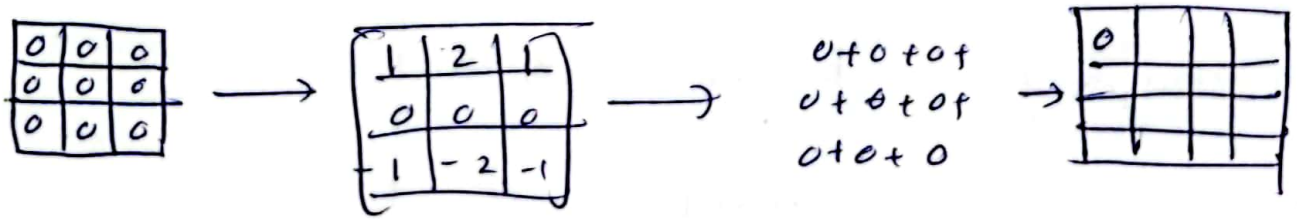
3 x 3

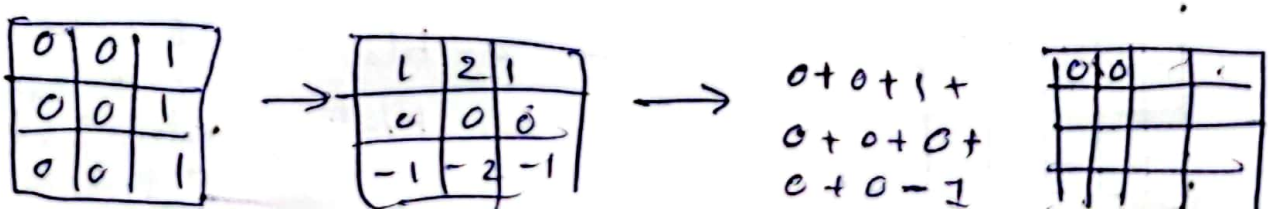
horizontal edge filter



4x4

We have 6x6 image.
Passed it through a 3x3 filter specifically called horizontal edge filter.

1) 

2) 

So we

3)

0	1	1
0	1	1
0	1	1

4)

1	1	1
1	1	1
1	1	1

5) ...

Let as considered

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

as the output.

if the filter is

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

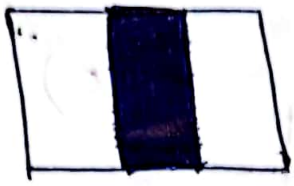
Vertical edge filter/kernel.

If we are reverting back the feature scaling we get the values ranging b/w 0-255

it becomes

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

The highest value will be converted to 255 (white) and lowest value will be " " 0 (black)



output.

Similarly in the case of human brain, there a numerous layers are inserted a layer. Here horizontal edge filter / vertical edge filter / other filters.

⇒ Filters: With the help of these filters we are trying to extract the information from the image.

Imp Whenever we pass a 6x6 image into a 3x3 filter we get a 4x4 o/p.

How this happens?

Let say size of the image = n = 6 and filter size = 3.

o/p size = $n - f + 1 = 6 - 3 + 1 = 4$

? How do you define the filter?

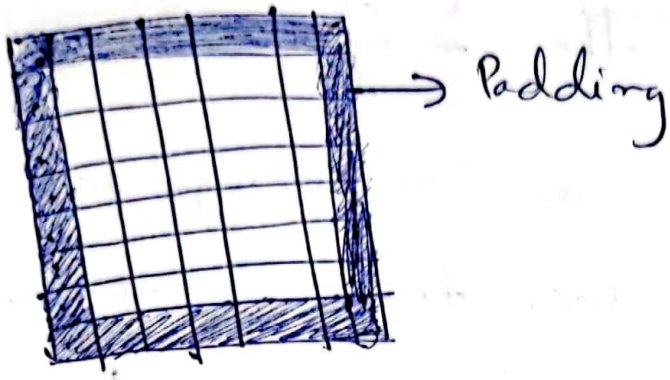
? While we are passing 6×6 image into a 3×3 filter we get 4×4. That means we are loosing some kind of information. So inorder to prevent what can we actually do?

ans: Solution is Padding.

Padding

Padding is building a compound around the image.

It is a technique used to preserve the spatial dimensions of the i/p data after convolution operations. It involves adding extra elements, such as zeros, to the i/p data before applying any convolutional filter. This helps prevent information loss, especially from the edge of the image, and maintains the spatial relationship and characteristic of the i/p data.



Zero Padding → fill the shaded cells with zero.
→ fill the cells with nearest value.

1. Same Padding
2. Valid Padding
3. Causual Padding

The updated formula = $\boxed{n + 2p - f + 1}$

$n \rightarrow$ size of the image
$p \rightarrow$ No of layers of Padding
$f \rightarrow$ filter size

$n = 6 \qquad p = 1 \qquad f = 3$

$$6 + 2 - 3 + 1 = \underline{\underline{6}}$$

In CNN we have to make sure that, we update the filters based on the i/p image.
So with the help of back propagation we need to update the filters.

Where do we specifically use the activation func?

After the convolution operation, wherever we get the output, on top of each value. we apply a ReLu activation function(max(0, x)).

? But why do we apply ReLu

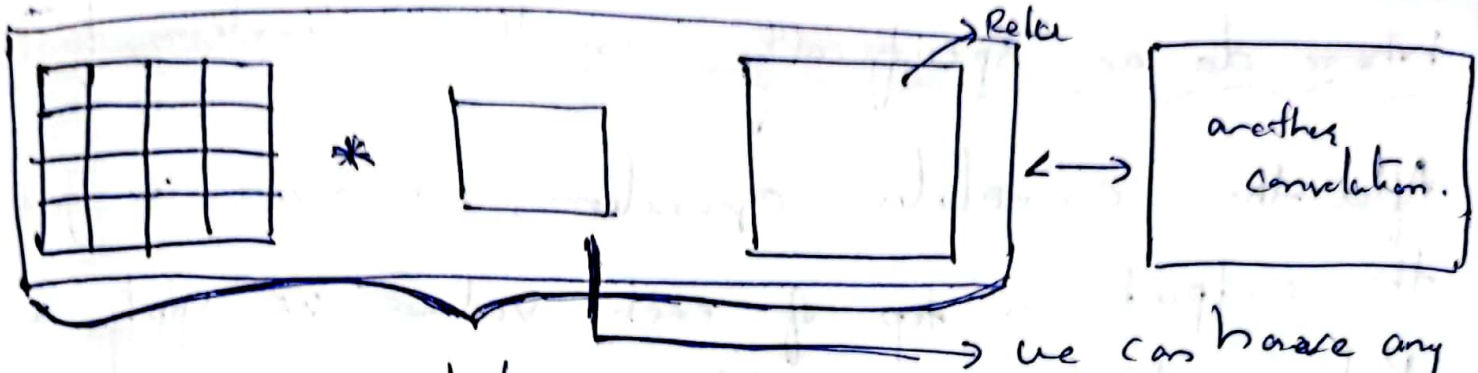
During the backpropagation the derivative can be found out. Since we can find out the derivative we can do the backpropagation and hence update the filter.

Stride

It is the number of Pixels by which a filter moves accross the input image during the convolution operation. It is a crucial para that determines how many units the filter shifts at each step, either horizontally, vertically, or both.

If, if Stride ≠ 1., then the updated formula

$$\frac{n+2P-F+1}{S}.$$

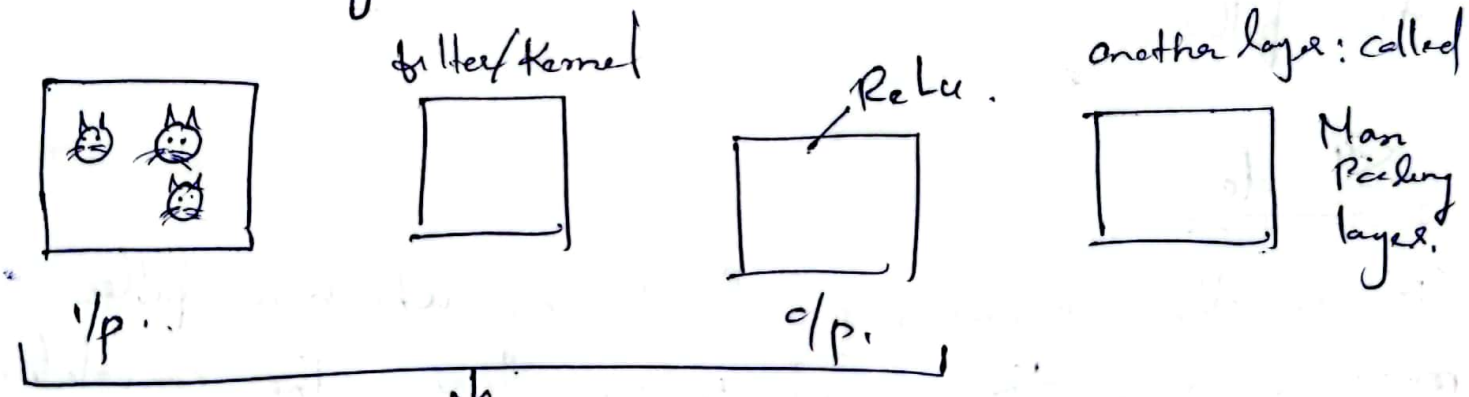


convolution operation.

we can have any no. of filters

Main aims of convolution operation is we need to learn ~~[illegible]~~ and update based on the o/p images

Max-pooling



convolution operations

Location invariant

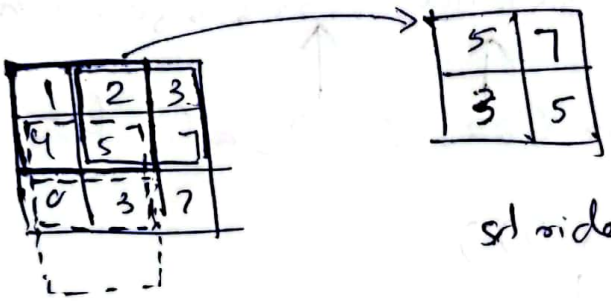
Let's say there are multiple objects ~~images~~ in the images, the cnn till it goes ahead towards the next neural n/w it should be able to extract the more and more clear info.

So we use max pooling.

Types of ~~Max~~ Pooling

1. Avg Pooling 2. Min Pooling 3. Max Pooling

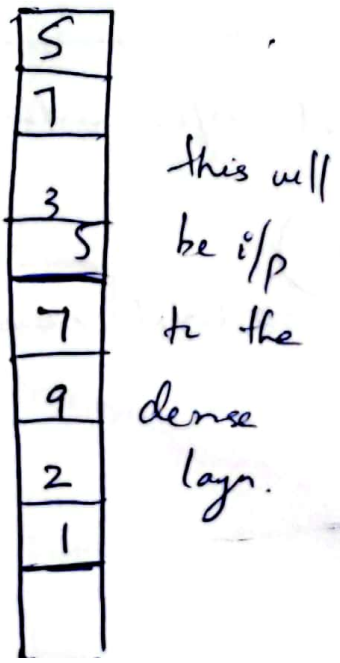
We are going to place the max Pooling on top of the o/p, and the o/p will only be focussing on the biggest number



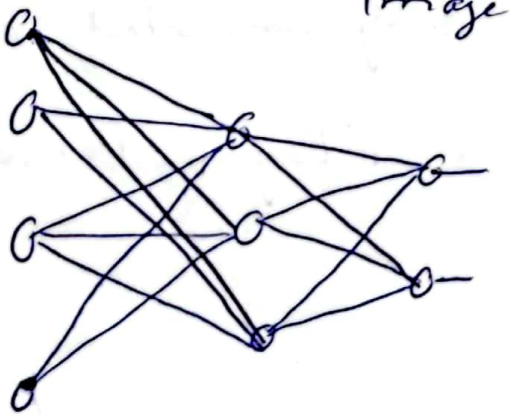
Based on different Problem statement, we can use min pooling, max pooling, avg pooling.

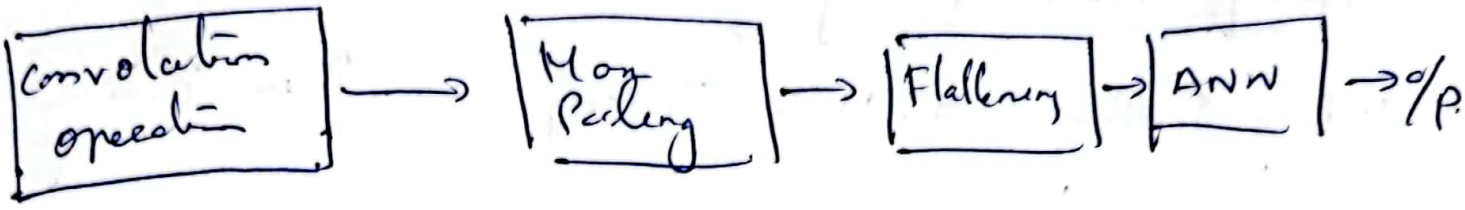
Flattening layer (looks like ANN Dense layer)

The pooling o/p will be elongated on this stage



this will be i/p to the dense layer.





Max pooling → refers to extracting more information from the images.

32 filters 3×3 size each.

model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)))

32 × 32 (I/p image size) 3 channels

How do you decide the number of filters?

Flatten layer:
model.add(layers.Flatten())

HL: ANN
model.add(layers.Dense(64, activation='relu'))

fully connected neural n/w with 64 dense neurons

o/p layer
model.add(layers.Dense(10))

→ Total number of o/p