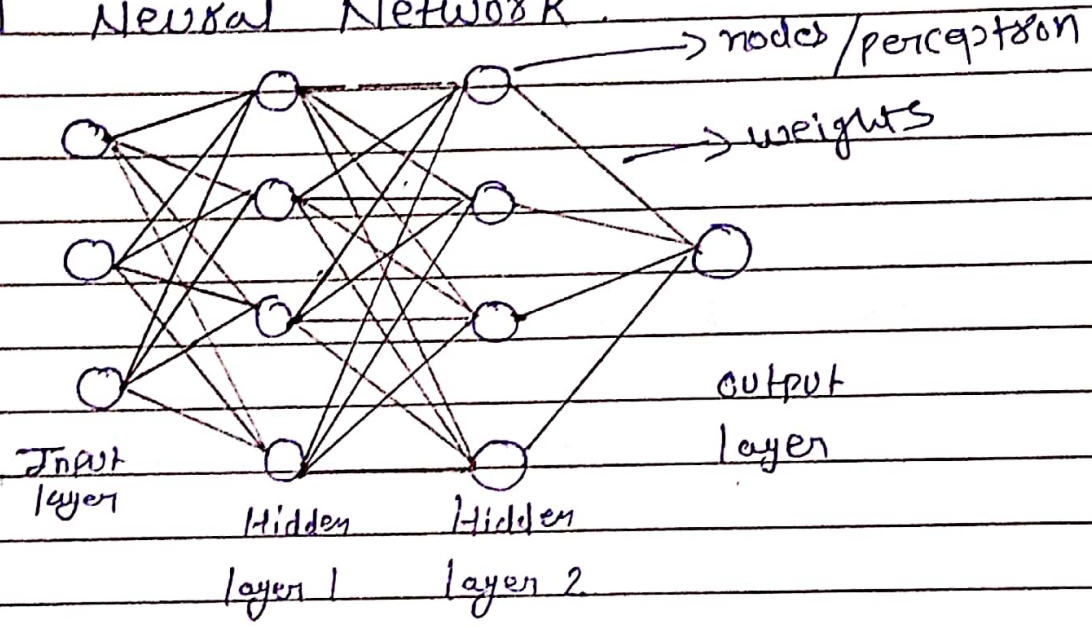
"Deep Learning"

What is deep learning ?

Deep Learning is a Subfield of Artificial Intelligence and Machine Learning that is inspired by the structure of human brain.

Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given with a given logical structure called Neural Network.



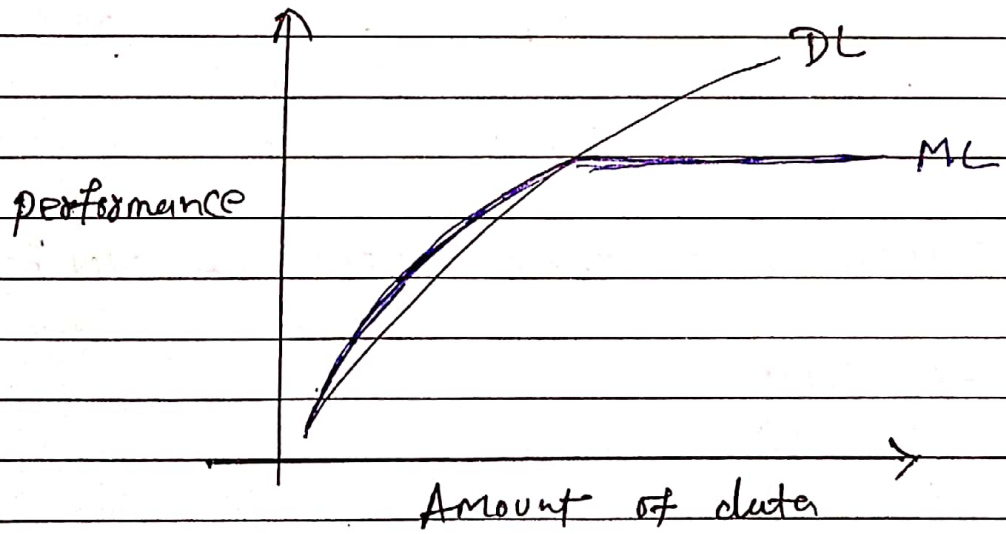
There are some types of NN :-

* ANN
* CNN
* RNN
* GAN
* etc.

Deep learning is a part of broader family of ML methods depend on ANN with representation / feature learning.

Deep learning algorithms uses multiple layers to progressively extract higher-level features from the raw input. for Eg, in image processing, lower layer may identify edges, while higher layers may identify the concept relevent to a human such as digits or letters or faces.

Deep learning Vs ML :-



	ML	DL
①	ML required less data. After a limit of data ML performance are fixed.	DL is data hungry. ~~After a~~ its perform improves linearly to data.
②	Does not require powerful hardware's i.e. it can work on CPU, so it is budget friendly.	Requires powerful hardware like GPU's So it is expensive.
③	Training time is low.	Training time is high (Cause big data)
④	in ML we provide some features so that model generate result by using extract the given features	but in DL uses representation / feature learning which allows System to extract feature's according to requirement.
⑤	ML is a Superset of DL	DL is a Subset of ML
⑥	The data represented in ML is quite differ Compared to DL as it uses structured data	The data represented in DL is quite diff as it uses ANN.

Why DL so famous :- & why now

- Datasets
- Hardware
- Economy
- Architecture
- People / community

Types of Neural Networks :-

There are various types of NN and let us discuss some main types of NN :

Neural networks are computational model that mimic the way biological neural networks in the human brain operate.

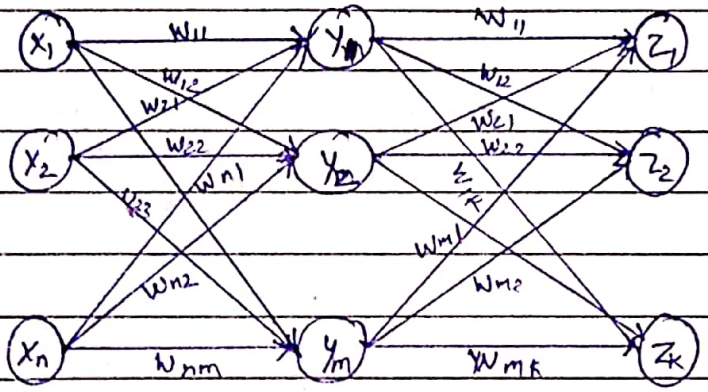
They consist of layers of neural that transform the i/p data into meaningful o/p through a series of mathematical operations.

(1) Feedforward Neural Networks :-

Feedforward neural networks are a form of ANN where without forming any cycles bet" layers or nodes means i/p can pass data through those nodes within the hidden level to the o/p nodes.

Applications :

in visual and voice recognition, NLP, financial forecasting and recommending system.

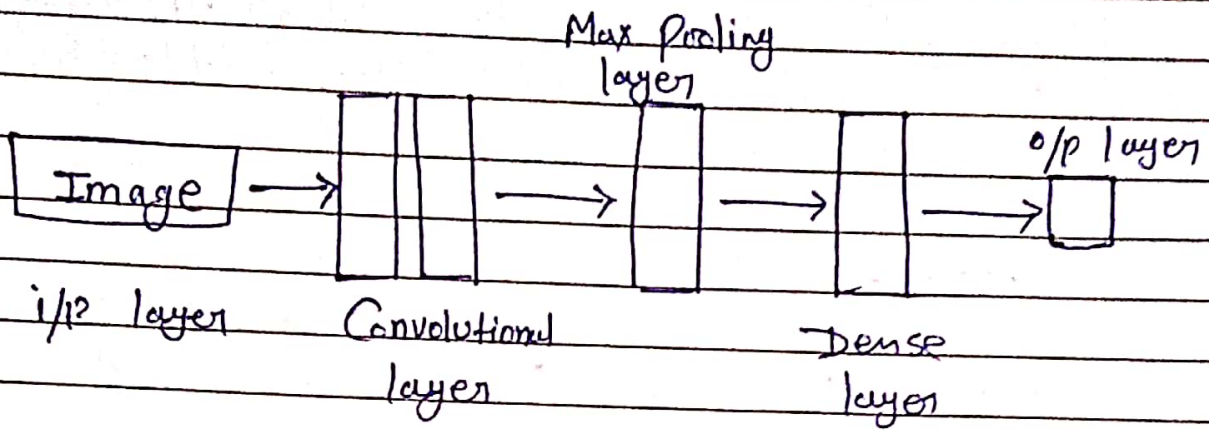


(2) Convolutional neural network :-

CNN structure is focused on processing the grid type data like images & videos by using convolutional layers filtering driving the patterns & spatial hierarchies.

Application :

Used for classification of images, object detection, autonomous driving and visualization in augmented reality.

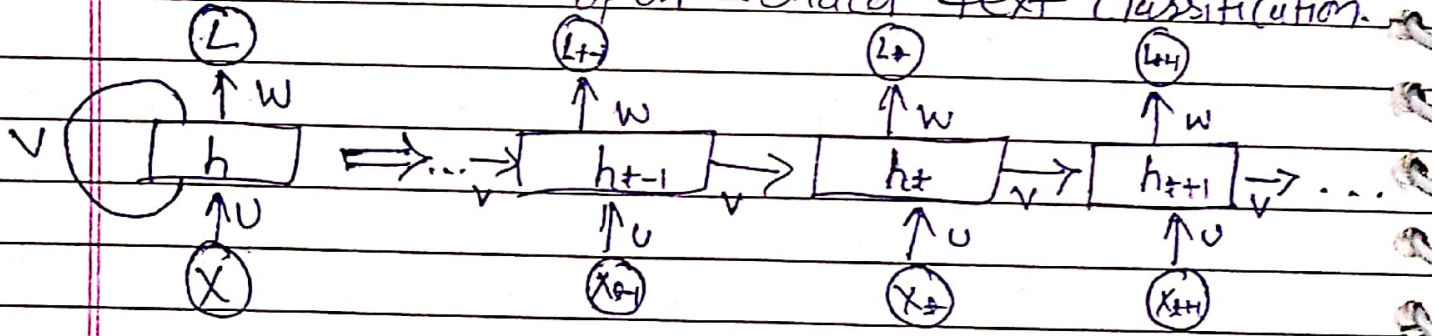


(B) Recurrent Neural Network (RNN) :-

RNN handles sequential data in which the current o/p is a result of previous i/p by looping over themselves to hold internal state.

Application :-

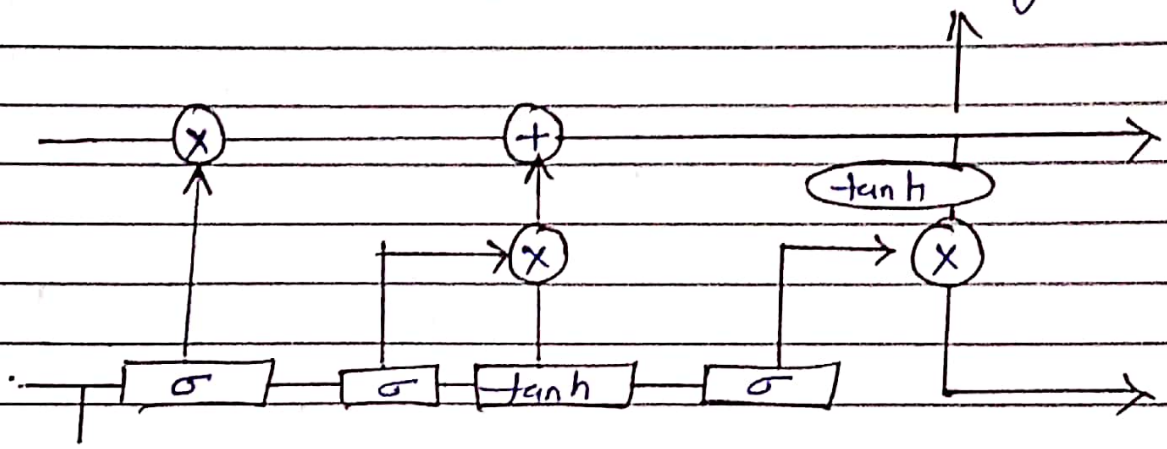
language translation, open-ended text classification.



④ LSTM :-

Long - Short-term memory Networks are a variant of RNNs They exhibit memory cells to solve the disappearing gradient issue and keep large ranges of info in their memory.

Applications : language translation and time series forecasting.

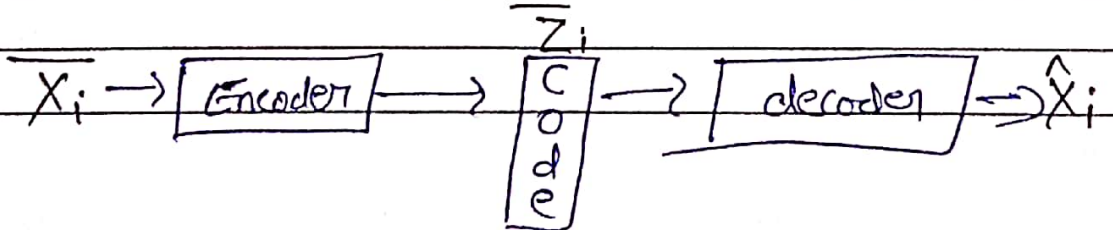


⑤ Autoencoders (AE) :-

Autoencoders are feedforward networks (ANN) that are trained to acquire the most helpful presentations of the info through the process of re-coding the i/p data.

Types :

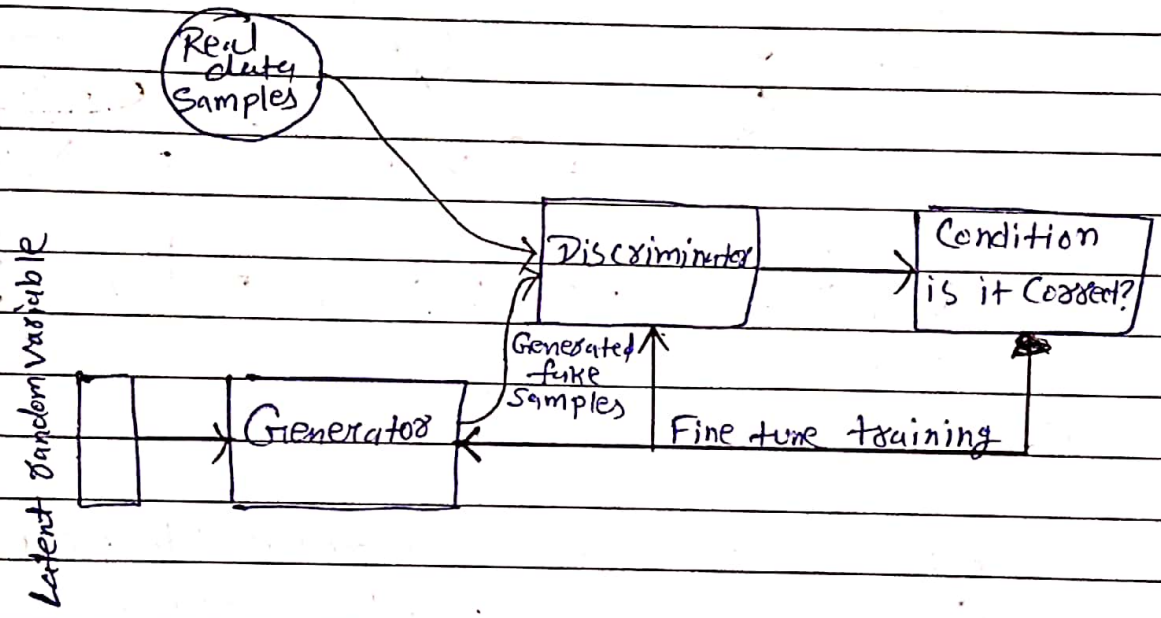
Variants include Undercomplete, Overcomplete, and Variational autoencoders.



(6) GANs :-

Generative Adversarial Networks has made up of two neural networks, the generator and discriminator, which Compete against each other.

The generator creates a fake generated data and the discriminator learns to differentiate the real from and fake data.



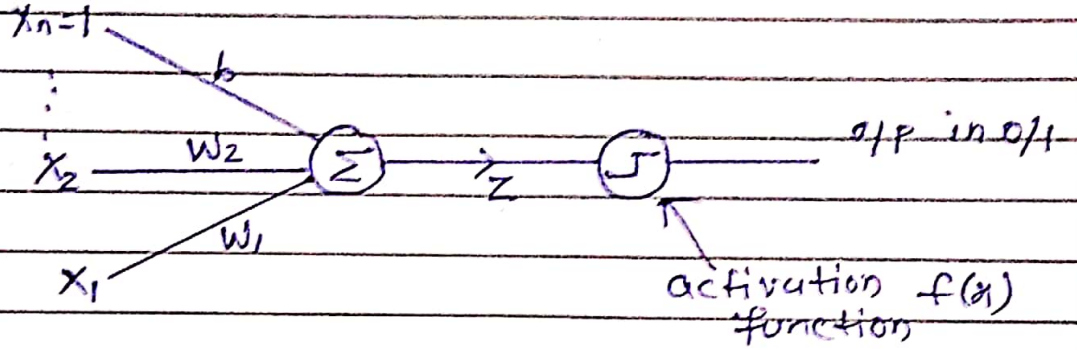
Perceptron :-

* Perceptron is a fundamental unit (building block) of NN.

* Multilayer Perceptron is also known as ANN.

* Basically perceptron is a algo that is work on supervised learning

* Design / mathematical model / function :-



where,

x_1 & x_2 are inputs
w_1 & w_2 are weights
b is bias
$\Sigma = w_1x_1 + w_2x_2 + b \times 1$

$$f(z) = \begin{cases} 1 & \Sigma \geq 0 \\ 0 & \Sigma \leq 0 \end{cases}$$

* weights showing how strongly connected input. (feature importance)

Geometric Intuition :-

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

if

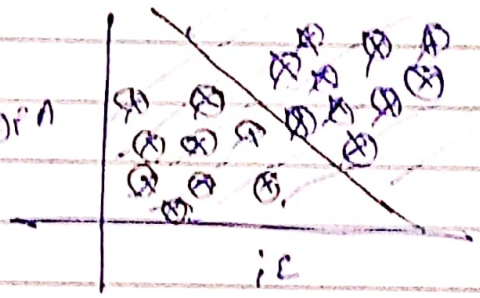
$w_1 = A$, $w_2 = B$, $b = C$

$x_1 = x$, $x_2 = y$

$$z = Ax + By + C$$ & this is the eqn of line $Ax + By + C = 0$

$$f(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

i.e $Ax + By + C > 0$ show ie it's generate a region.



So perceptron is nothing but line & its work to create a region & divide class.

because of these perceptron is called binary classifier.

in 2D perceptron work like a line
in 3D ——————— plane
in 4D ——————— hyperplane

but always Perceptron divide data into 2 regions so it is called Binary Classifier.

Perceptron can be use only on linear or sort of linear data.

Loss function :—

it is a way to define how over model is work (good or bad).

formula of Loss function :

$$L(w_1, w_2, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w_1, w_2)$$

(Regularization)

$$L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$$

$f(x_i) = w_1 x_1 + w_2 x_2 + b$

So

$$L(w_1, w_2, b) = \frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i f(x_i))$$

where

n is no of rows in data

Explanation of loss function :—

$$L = \frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i f(x_i))$$

With the help of eg

	X_1	X_2	Y
1	X_{11}	X_{12}	y_1
2	X_{21}	X_{22}	y_2
⋮			
n			

where $f(x_i) = w_1 x_{i1} + w_2 x_{i2} + b$

So if we have only 2 data than

$$L = \frac{1}{2}\left[\max(0, -y_1 f(x_1)) + \max(0, -y_2 f(x_2))\right]$$

McCulloch – pitts Neural Model :-

The McCulloch and pitts (MCP) model holds a place of historic Significance.

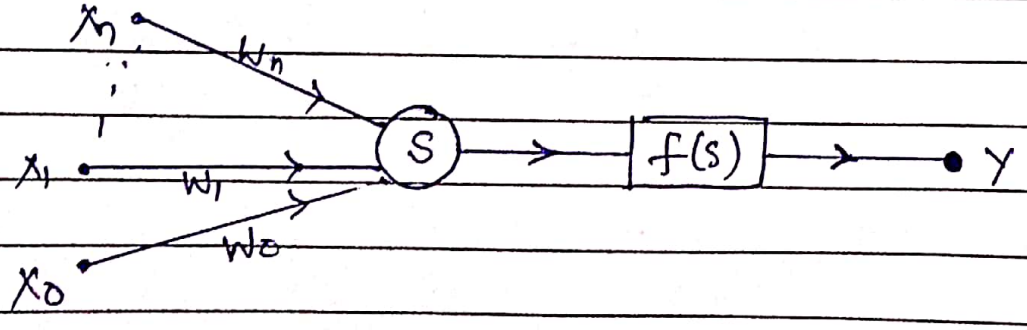
Proposed in 1943 by warren McCulloch and walter pitts, this model laid the foundation framework for understanding how neural network can mimic the brain's processing abilities.

Definition =>

The MCP model is a mathematical representation of a biological neuron.

It Simplifies the Complex operations of a neuron into a basic Computational unit.

Each neuron receives multiple binary inputs, processes them through weighted Summation, and produces a binary o/p based on a threshold function.



Mathematical foundation :—

The MCP model operates on a straight forward yet powerful set of principles :

1. Inputs and Weights :

Each neuron receives binary inputs $x_1, x_2 \ldots$ and assigns weights $w_1, w_2 \ldots$ to these i/p

2. Summation :

The neuron computes the weighted sum of its i/p

$$S = \sum_{i=1}^{n} w_i x_i$$

3. Activation Function :

The sum S is compared to a threshold θ. using a step function :

$$y = \begin{cases} 1 & \text{if } S \geq 0 \\ 0 & \text{if } S < 0 \end{cases}$$

Eg Calculation :

Let's, Consider a neuron with three binary i/p (x_1, x_2, x_3) and weights $w_1 = 0.5$, $w_2 = 0.3$, & $w_3 = 0.2$ & a threshold $\theta = 0.7$

for inputs $x1 = 1$, $x2 = 1$, $x_3 = 0$

$$S = w_1x_1 + w_2x_2 + w_3x_3$$

$$S = 0.5 \times 1 + 0.3 \times 1 + 0.2 \times 0$$

$$S = 0.5 + 0.3 = 0.8$$

Since $0.8 \geq 0.7$ the o/p $y = 1$

* The MCP model is one of the first attempts to create an artificial neuron. Think of it as a very simple brain cell in a Computer.

It helped lay the groundwork for today's AI, but it has its limits.

Limits of the MCP model :-

① Basic o/p :

This model only gives two possible results : yes or no

0 or 1

(2) Simple Problems only :

It can only solve problems that are straightforward and can be separated by a single line. More complex issues, like the XOR problem are beyond its reach.

(3) Fixed weights :

In the MCP model the weight of each i/p is fixed.

This means it doesn't learn or get better over time by adjusting these weights.

(4) No learning ability :

This model doesn't have a way to learn from its experience, it can't adjust itself to improve its performance.

Truth table Example :

Let's implement an AND logic gate using this model

Conditions :

→ Inputs : X_1 and X_2 (binary)
→ Weights : $W_1 = 1$, $W_2 = 1$
→ Threshold : $\theta = 2$

Now apply the model logic :

x_1	x_2	weighted sum $(x_1w_1 + x_2w_2)$	Sum $\geq \theta$	o/p
0	0	0	No	0
0	1	1	No	0
1	0	1	No	0
1	1	2	yes	1

Q.1 : Explain the structure and functioning of a biological neuron. How is it modeled in Artificial networks?

Ans : A biological neuron is the basic unit of the human brain responsible for receiving and transmitting info using electrical and chemical signals.

Biological Neural Network (BNN) is a structure that consists of Synapse, dendrites, cell body and axon.

in this neural network the processing is carried out by neurons. Dendrites receive signals from other neurons, Soma sums all the incoming signals and axon transmits the signals to other cells.

Dendrites → Cell body (Soma) → Axon → Synapse → next neuron.

Dendrites : Branch-like structures that receive signals from other neurons

Soma : Also called the cell body processes i/p signals.

Axon : A long fiber that sends signals to other neurons.

Synapse : Gap bet" two neurons where signal transmission occurs

How is it modeled in ANNs ?

Artificial neurons in neural networks are mathematical models that mimic biological neurons.

Diagram from previous notes

Qus: What are Computational units in DL?

Ans: in deep learning, Computational units are the basic building blocks of a neural network.

These are mathematical functions or models that take inputs, process them through a function and produce an o/p.

Think of a computational unit as an artificial version of a biological neuron — it computes an o/p based on the weighted sum of i/ps and an activation function.

Structure of a Computational Unit:

Each Computational unit typically performs the following steps:

(1) Takes inputs: $x_1, x_2, x_3, \ldots, x_n$

(2) Applies weights and Bias:

$$z = (w_1 x_1) + (w_2 x_2) + (w_3 x_3) + \ldots + (w_n x_n) + b$$

→ weights define the importance of each i/p
→ Bias allows shifting the activation function left or right.

(3) Activation function is applied.

This adds non-linearity so the model can learn complex patterns.

$$y = f(z)$$

Mathematical Summary ⇒

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Eg of Computational units

1. Perceptron :

The simplest Computational unit, proposed by Rosenblatt

$$y = \text{Sign}(w \cdot x + b)$$

it o/p +1 or −1 depending on the weighted sum.

2. Sigmoid unit :

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

Used in older networks or binary classification problems.

3. ReLU Unit :

$$y = \max(0, z)$$

Fast, efficient, and widely used in CNNs, RNNs etc.

4. LSTM Unit :

Advanced Computational unit for Sequential data.

it Contains Gates, Memory Cell Helps remember long-term dependencies.

5 Convolutional Unit :

Used in CNNs for images
- it performs Convolutional operations
- Helps extract features like edges, textures.

Qus What is a linear perceptron? Derive its learning ~~rule~~ rule.

The linear perceptron is the simplest form of a neural network — a single-layer binary classifier

It is designed to separate i/p data into two classes using a linear decision boundary (a line, plane or hyperplane depending on the no of i/p features).

The perceptron learn this boundry by adjusting its internal weights during training so that it can make correct classification.

It is suitable only for linearly separable datasets.

perceptron diagram

Derivation of the learning rule :

To adjust the weights in such a way that the perceptron reduces the classification error & makes better

predictions over time.

let's

inputs $x = [x_1, x_2, \cdots x_n]$

weight vector $W = [w_1, w_2, \cdots w_n]$

Bias : b

Target o/p : $y \in \{-1, 1\}$

Predicted o/p : $\hat{y} = \text{Sign}(w \cdot x + b)$

learning rate : η

Step 1 : Calculate error

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

Step 2 : Update Rule for weights

if the prediction $\hat{y}$ is wrong, update the weights as :

$$w_{new} = w_{old} + \eta (y - \hat{y}) \cdot x$$

Step 3 : Update Rule for Bias

$$b_{new} = b_{old} + \eta (y - \hat{y})$$

Convergence Theorem :-

If the training data is linearly Separable the Perceptron learning algo is guaranteed to converge to a Sol^n in a finite no of steps.

if the data is not linearly Seperable (like XOR problem) it will never Converge.

Qus Explain the Perceptron learning algo with steps and example ?

The Perceptron learning algo is used to train a Single-layer perceptron to classify input data into two classes (+1 or -1)

It works by updating weights based on misclassifications.

The aim is to find the best-fit hyperplane that separates the data.

~~Algorithm Steps :-~~

~~1) Initialize the~~

Perceptron is a linear Supervised ML algo it is used for binary classification

Perceptron learning algo is also understood as an artificial neuron or neural network unit that helps to detect certain i/p data computations in business intelligence.

The perceptron learning algorithm is treated as the most straight forward ANN. Hence, it is a single layer neural network with four main parameters, i.e., input values, weights and Bias, net Sum and an activation function.

There are 4 significant steps in a PLA :

(1) First, multiply all i/p values with corresponding weight values and then add them to determine the weighted Sum. Mathematically.

$$\sum w_i \cdot x_i = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

& then add another essential term called bias 'b' to the weighted sum to improve the model performance.

$$\sum w_i \cdot x_i + b$$

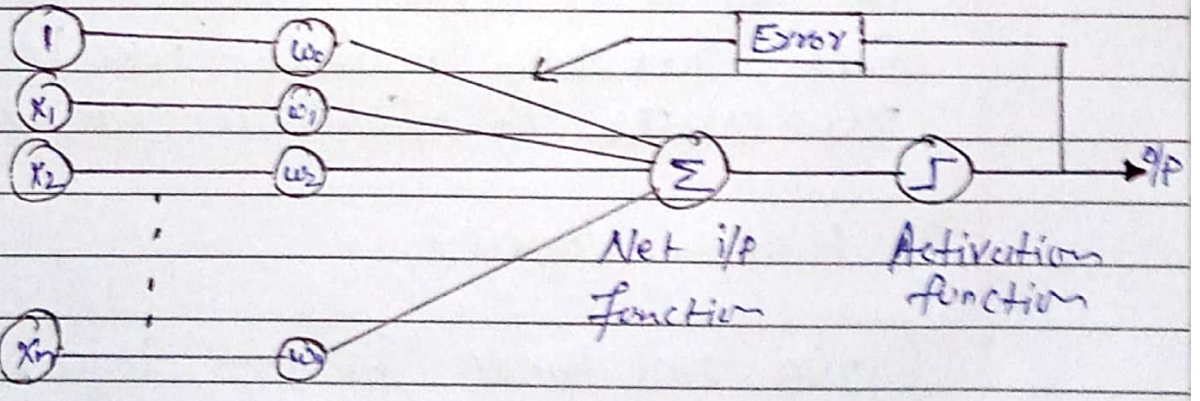
(2) An activation function is applied to this weighted sum, producing a binary or a Continuous valued o/p

$$y = f(\Sigma w_i \cdot x_i + b)$$

(3) Now the diff. bet^n this o/p and the actual target value is computed to get the error term, E. generally in terms of mean squared error.

$$E = (y - y_{actual})^2$$

(4) we optimize this error (loss function) using an optimization algo. Generally some form of GDA (Gradient Descent Algo) is used to find the optimal values of the hyperparameter like learning rate, weight, Bias etc.



Feed Forward Network :-

Feed forward Neural Network (FFNN) is a type of ANN in which info flows in a single direction from the i/p layer through hidden layers to the o/p layer. — without loops or feedback.

It is mainly used for pattern recognition tasks like image & speech classification

Structure of FFNN :

FFNN have a structured layered design where data flow sequentially through each layer.

① Input layer :

The i/p layer consists of neurons that receive the i/p data. Each neuron in the i/p layer represents a feature of the i/p data.

② Hidden layer :

One or more hidden layers are placed bet^n the i/p & o/p layers.

These layers are responsible for

learning the complex patterns in the data.
Each neuron in a hidden layer applies a weighted sum of i/p followed by a non-linear activation function.

(3) o/p layer:

The o/p layer provides the final o/p of the network.

Diagram

Activation function :-

Common activation functions:

→ Sigmoid:

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

→ Tanh:

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

→ ReLU:

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

Foswood Propagation :

in Foswood propogation i/p data moves through each layer of neural network where each neuron applies weighted sum, adds bias, passes the result through an activation functin & making predictions.

~~These~~ This process is crucial before backpropogation updates the weights.

it determines the o/p of neural network with a given set of input and current state of model parameters (weights and bias).

Mathematical explanation of Foswood pu:
let's taryet o/p is 0.5 & the learning rate is 1.

① Initial Calculation

The weighted sum at each node is Calculated using:

$$a_j = \Sigma (w_{i,j} * x_i)$$

Where,

a_j = is the weighted sum of all the i/p & weights at each node.

- $w_{i,j}$ represents the weights bet^n the i^{th} i/p and j^{th} neuron.
- x_i represents the value of the i^{th} i/p

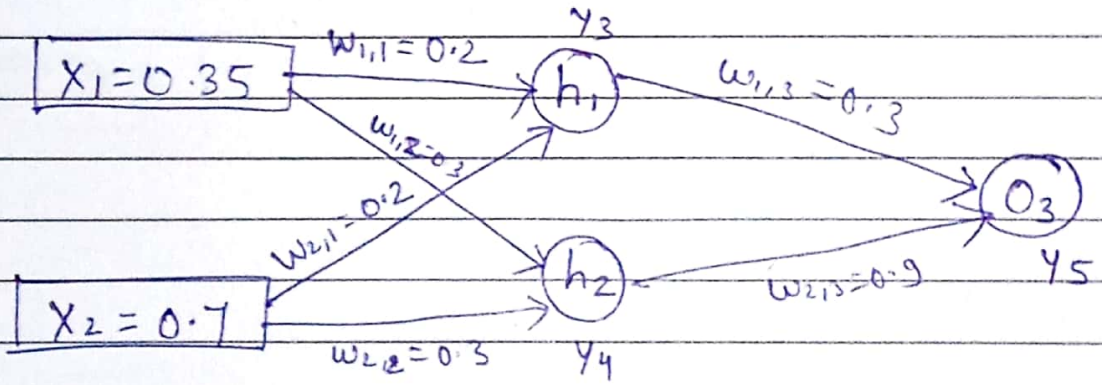
→ After applying the activation function to a_1, we get the o/p of the neuron

$$o_j = \text{activation function}(a_j)$$

② Sigmoid function :

The Sigmoid function returns a value bet^n 0 and 1, introducing non-linearity into the model.

$$y_j = \frac{1}{1+e^{-a_j}}$$



③ Computing Outputs :

At h1 node

$$a_1 = (w_{1,1} \cdot x_1) + (w_{2,1} \cdot x_2)$$
$$a_1 = (0.2 \times 0.35) + (0.2 \times 0.7)$$
$$a_1 = 0.21$$

Once we Calculated the a_1 Value, we can now proceed to find y_3

$$y_j = F(a_{y_j}) = \frac{1}{1+e^{-a_1}}$$

$$y_3 = F(0.21)$$

$$y_3 = \frac{1}{1+e^{-0.21}} = 0.56$$

Similarly find the values of y_4 at h_2 and y_5 at O_3

$$a_2 = (w_{1,2} * x_1 + w_{2,21} * x_2) = (0.3 \times 0.35 + 0.3 \times 0.7)$$

$$a_2 = 0.315$$

So

$$y_4 = F(0.315) = \frac{1}{1+e^{-0.315}} = 0.59$$

&

$$\boxed{y_5 = 0.67}$$

(4) Error Calculatim

let our actual o/p is 0.5 but we obtained 0.67.

$$\text{Error} = y_{\text{target}} - y_5$$
$$= 0.5 - 0.67$$
$$\text{Error} = -0.17$$

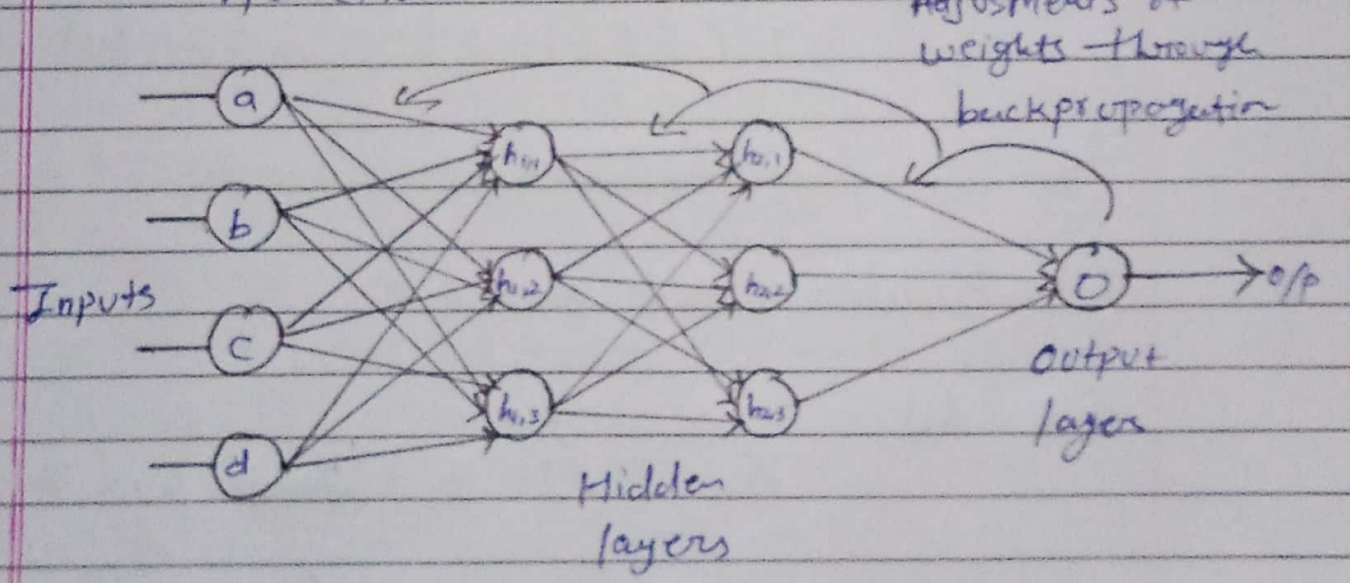
Backpropagation :

Backpropagation is a technique used in DL to train ANNs particularly Feed forward networks.

It works iteratively to adjust weights and bias to minimize the cost function

Backpropagation often uses optimization algo's like Gradient Descent or Stochastic gradient Descent.

The algo computes the gradient using the chain rule from Calculus allowing it to effectively navigate Complex layers in the neural network to minimize the cost function.



Backpropagation :

① Calculating Gradients =>

The change in each weight is calculated as:

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

where,

- δ_j is the error term for each i/p
- η is the learning rate.

② o/p unit Error =>

For O3:

$$\delta_5 = y_5(1-y_5)(y_{target} - y_5)$$

$$= 0.67(1-0.67)(0.5-0.67)$$

$$= -0.0376$$

③ Hidden unit Error =>

For h1 :

$$\delta_3 = y_3(1-y_3)(w_{1,3} * \delta_5)$$

$$= 0.56(1-0.56)(0.3 * (-0.0376))$$

$$= -0.0027$$

For h_2:

$$\delta_4 = y_4(1-y_4)(w_{2,3} * \delta_5)$$
$$= 0.59\,(1-0.59)\,(0.9 \times (-0.0376))$$
$$= -0.0819$$

④ Weight Updates

For the weights from hidden to o/p layer

$$\Delta w_{2,3} = 1 * (-0.0376) \times 0.59 = -0.022184$$

New weight:

$$w_{2,3}(new) = -0.022184 + 0.9 = 0.877816$$

Similarly other weights are updated:

- $w_{1,1}$ (new) = 0.200945
- $w_{1,2}$ (new) = 0.273225
- $w_{1,3}$ (new) = 0.086615
- $w_{2,1}$ (new) = 0.269445
- $w_{2,2}$ (new) = 0.18534

After updating the weights the forward pass is repeated:

$y_3 = 0.57$
$y_4 = 0.56$
$y_5 = 0.61$

So now

$$\text{Error} = y_{target} - y_5$$
$$= 0.5 - 0.61$$
$$= -0.11$$

Unit : 2

Multilayer Perceptron :-

Multilayer perceptron (MLP) is an ANN widely used for solving classification & regression tasks.

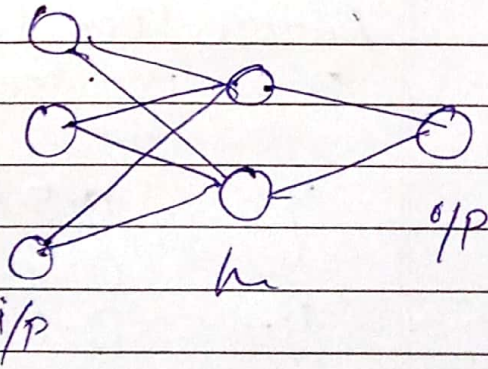
MLP Consists of fully connected dense layers that transform i/p data from one dimension to another.

it is called "multi layer" because it contains an input layer, one or more hidden layers and an o/p layers.

The purpose of an MLP is to model complex relationships bet" i/p & o/p making it a powerful tool for various ML tasks.

Key Components (Architecture) of MLP :-

→ i/p layer
→ hidden layers
→ o/p layer



Working of MLP :

Step 1 : Forward propagation
→ weighted sum
→ Activation function

Step 2 : loss function Calculate

Step 3 : Backpropagation

→ Gradient Calculation
→ Error propagation
→ Gradient Descent

Step 4 : Optimization

Applications of MLP :-

→ Tabular Data Classification & Regression

→ Handwritten Digit and Character Recognition

→ Function Approximation & Control

→ Time Series forecasting

→ Feature Extraction & pre-training

→ Anomaly Detection

Advantages :

→ Versatility
→ Non - linearity
→ Parallel Computation

Disadvantages :

→ Computationally expensive
→ prone to overfitting
→ Sensitivity to Data scaling

Gradient Descent Algo :-

Gradient Descent is an optimization algo used in ML to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy bet^n the predicted o/p of the model and the actual o/p.

Gradient descent aims to find the parameters that minimize the discrepancy and improve the model performance.

The algo operates by calculating the gradient of the cost function, which indicates the direction and magnitude of the ~~cost function~~. steepest ascent.

By iteratively updating the model's parameters in the negative gradient direction, gradient descent gradually converges towards the optimal set of parameters that yields the lowest cost.

Gradient descent can be applied to various ML algo including linear regression, logistic regression, neural networks & SVM.

Cost function :

It is a function that measures the performance of a model for any given data.

Cost function quantifies the error bet" predicted values and expected values and presents in the form of a real no.

Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$

parameters : θ_0, θ_1

Cost function : $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Goal : minimize $(J(\theta_0, \theta_1))$

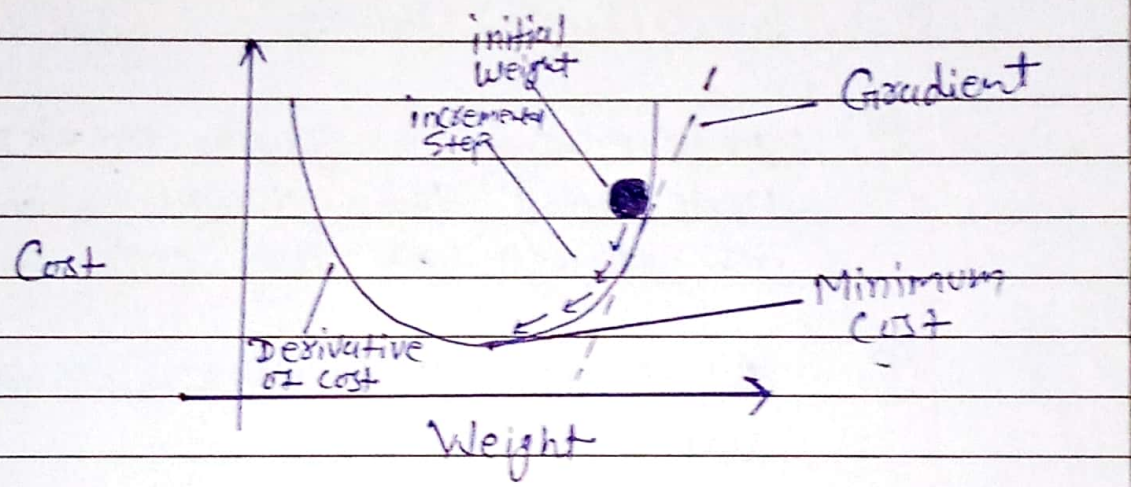
Given a cost function $J(\theta)$, where θ represents the model parameters, the update rule for gradient descent is:

$$\theta := \theta - \alpha \nabla J(\theta)$$

Where.

θ are the parameters
α is learning rate
$\nabla J(\theta)$ is the gradient of the Cost function with respect to θ.

Gradient descent was originally proposed by CAUCHY in 1847. it is also known as steepest descent.



The goal of the GDA is to minimize the Cost function. To achive this goal, it performs two steps iteratively.

1. Compute the gradient
2. Make a move in the opposite direction of the gradient.

GDA Steps :

(1) Start with initial guesses for the parameter θ.

(2) Use the current parameter to make predictions.

(3) Evaluate the cost function $J(\theta)$ to measure the error bet^n prediction & actual values.

(4) Determine the gradient $\nabla J(\theta)$ to understand the direction and rate of change of the cost function.

(5) Adjust the parameters using the update rule :

$$\theta := \theta - \alpha \nabla J(\theta)$$

(6) Repeat step 2–5 Until Convergence.

Variants of GD :

→ Batch Gradient Descent

→ Stochastic Gradient Descent (SGD)

→ Mini – Batch

Batch, Stochastic & Mini-batch GD :-

(1) BGD :-

Batch Gr. D. Computes the gradient using the entire dataset before updating model parameters.

While ~~these~~ this approach provides accurate gradient estimates, it can be Computa-tionally expensive for large datasets.

- pros => Accurate updates, Stable Convergence
- Cons => High memory Consumption, Slow for large datasets.

(2) Stochastic Gr.D. :-

SGD updates model parameters after Computing the gradient from a single data point. This approach is Computationally Cheaper but introduces noise, leading to an unstable Convergence path.

- pros => Fast updates, works well for online learning
- Cons => High Variance

③ MBGD :-

MBGD Combines the benefits of both batch and stochastic gradient descent. Instead of using the entire dataset or a single sample, MBGD processes small batches of data at a time. This balances computational efficiency & model stability.

Pros : Faster training than batch gradient descent, more stable than SGD.

Cons : Choosing the right batch size can be challenging.

Aspect	BGD	SGD	MBGD
Data processed per update	Entire training dataset	Single training ex.	Small subset of training data
Update frequency	Once per epoch	After each training ex	After each mini batch.
Convergence Speed	Can be slow due to processing the entire dataset	Typically faster but may fluctuate	Faster & more stable than SGD

Convergence Stability	Stable Convergence to the min—	May oscillate around the min due to noise	More stable than SGD, less than Batch.
Memory requirement	High	low	Moderate
Computational Efficiency	less efficient for large dataset	Highly eff—	balance effic— & performance
Typical use cases	Small to medium size dataset where precision is crucial	large scale or online learning scenarios	Most DL applications.

Empirical risk minimization :—

In SL, the goal is to find ~~the~~ a function f that maps inputs x to o/p y such that the predictions $f(x)$ are as close ase possible to the actual o/p s.

The 'risk' or 'loss' quantifies the discrepancy bet" predicted & actual o/p typically define as :

$$R(f) = E_{(x,y)\sim P}[\, l(f(x), y)\,]$$

Here, l is a loss function &
P is the true but unknown
data distribution.

Since P is unknown, we approximate the risk using the empirical risk over a training dataset $\{(x_i, y_i)\}_{i=1}^{n}$:

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

ERM involves selecting the function f from a hypothesis space F that minimizes this empirical risk:

$$\hat{f}_n = \arg\min_{f \in F} \hat{R}_n(f)$$

This approach assumes that minimizing the loss on training data will lead to good performance on unseen data.

Applications :—

→ SVM , → logistic regression
→ Linear Regression

Advantages :

→ Theoretically Sound
→ Wide Applicability
→ Flexibility

Challenges :

→ Overfitting
→ No Guarantee of Generalization
→ Sensitive to Noise

Regularization :-

Regularization in DL is a technique used to prevent overfitting by adding a penalty term to the loss function, discouraging the model from becoming overly Complex.

This helps the model generalize better to unseen data.

Overfitting happens when a model learns the training data too well, including the noise and outliers which causes it to perform poorly on new data.

Types of Regularization :

① Lasso Regression :

A regression model which uses the L1 Regularization technique is called LASSO (Least Absolute Shrinkage and Selection Operator) regression.

Lass regression adds the "absolute value of magnitude" of the Coefficient as a penalty term to the loss function (L).

$$\text{Cost} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |W_i|$$

where,

m = no of features
n = no of Eg
y_i = Actual Target Value
$\hat{y}_i$ = predicted Target Value

② Ridge Regression :-

A regression model that uses the L2 regularization technique is called Ridge regr.

R.R. adds the "Squared magnitude" of the Coefficient as a penalty term to the loss function (L).

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

Where,

m = ——— , n ——— , y_i ——— , $\hat{y}_i$ ———

w_i = Coefficients of the features

λ = Regularization parameter.

(3) Elastic Net Regression :

Elastic Net Regression is a Combination of both L1 as well as L2 regularization

That implies that we add the absolute norm of the weights as well as the squared measure of the weights. With the help of an extra hyperparameter that controls the ratio of the L1 & L2 regu——

$$\text{Cost} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \left((1-\alpha) \sum_{i=1}^{m} |w_i| + \alpha \sum_{i=1}^{m} w_i^2 \right)$$

where,

α = Mixing Parameter (where $0 < \alpha < 1$)

→ $\alpha = 1$ Corresponds to L1 Regu——

→ $\alpha = 0$ ——————— L2 ———

→ Values bet^n 0 & 1 provide a balance of both L1 & L2 regu——

Bias :

Bias measures the error introduced by approximating a real world problem by a much simpler model.

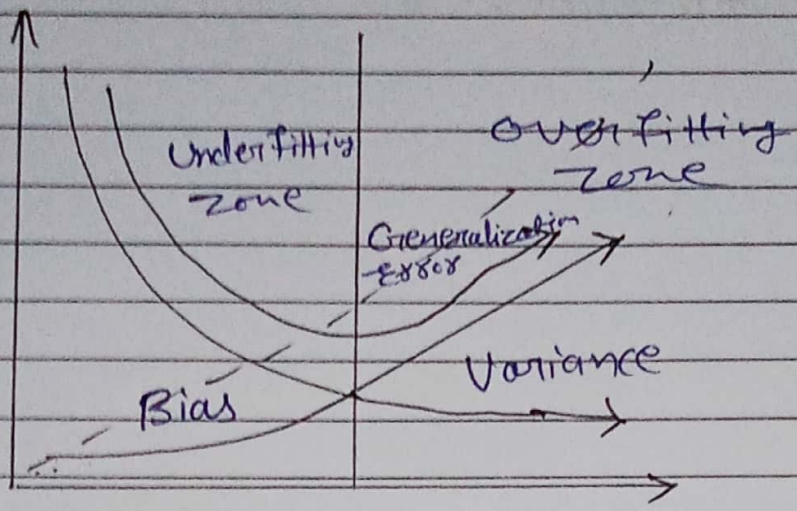
Variance :

Variance measures how much function would change if we estimated it using a different training dataset drawn from the same distribution.

Bias Variance tradeoff :

The Bias-Variance Tradeoff is a fundamental concept in ML. It refers to the balance bet^{n} bias & Variance which affect predictive model performance.

The bias-Variance tradeoff demonstrates the inverse relationship bet^{n} bias & Variance. When one decreases the other tends to increase & vice versa.



Benefits of Regularization :

→ Prevents Overfitting

→ improves interpretability

→ Enhances Performance

→ Stabilizes models

→ Prevents Complexity

→ Allows Fine - Tuning

Autoencoders :—

An autoencoder is a type of artificial neural network that learns to represent data in a compressed form and then reconstructs it as closely as possible to the original i/p.

Autoencoder consists of two components:

- Encoder ⇒

This compresses the i/p into a compact representation & capture the most relevant features.

- Decoder ⇒

It reconstructs the i/p data from this compressed form to make it as similar as possible to the original i/p.

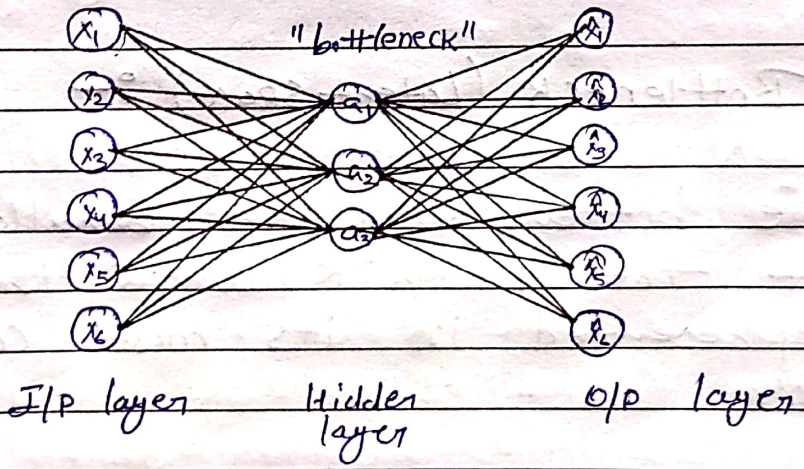
→ Autoencoder aim to minimize reconstruction error which is the difference bet^n the i/p & the reconstructed o/p. They use loss functions such as Mean Squared Error (MSE) or Binary Cross Entropy (BCE) and optimize through backpropagation and gradient descent

They are used in applications like image processing, anomaly detection, noise removal and feature extraction.

Architecture of autoencoder =>

The architecture of an autoencoder consists of 3 main components:

The encoder, the bottleneck (Latent space) and the Decoder.



1. Encoder =>

The encoder is the part of the network that takes the i/p data and compresses it into a smaller lower dimensional representation.

- i/p layer =>

This is where the original data enters the network.

- Hidden layer ⇒

These layers apply transformations to the i/p data. The encoder's goal is to extract essential features and reduce the data's dimensionality.

- 'O/p of Encoder (Latent Space)=>

The encoder o/p a compressed version of the data often called the latent representation or encoding.

(2) Bottleneck (Latent Space) :-

The bottleneck is the smallest layer of the network where the data is represented in its most compressed form

(3) Decoder :-

The decoder is responsible for taking the compressed representation from the latent space & reconstructing it back into orginal data form.

Loss function in autoencoder Training :-

During training an autoencoder aims to minimize the reconstruction loss which measures the difference bet" the original i/p & the reconstructed o/p. The choice of loss function depends on the type of data:

* Mean Squared Error (MSE) :-

This is commonly used for continuous data. It measures the average squared differences bet" the i/p and the reconstructed data.

* Binary Cross-Entropy :-

Used for binary data (0 or 1 value). it calculate the diff. in probability bet" the o/p & ~~o/p~~.

Types of autoencoders :-

→ Denoising Autoencoder

→ Sparse ——

→ Variational ——

→ Convolutional ——

Unit : 3
Convolutional Networks

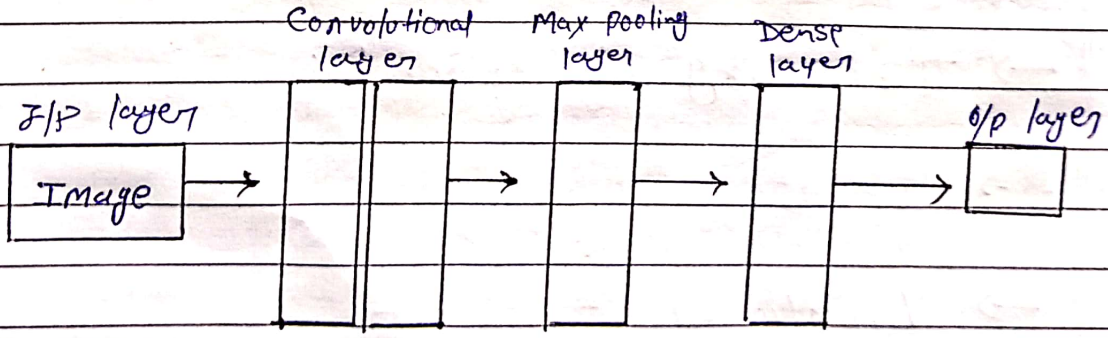
CNN :-

Convolutional Neural Network (CNN) is an advanced version of ANN, primarily designed to extract features from grid-like matrix datasets.

This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role.

CNN are widely used in Computer Vision applications due to their effectiveness in processing visual data.

CNNs Consist of multiple layers like the i/p layer, Convolutional layer, Pooling layer & fully Connected layers.



Advantages :-

-> Good at detecting patterns & features in images, Videos and audio Signals.

-> Robust to translation, rotation and scaling invariance.

-> End-to-end training, no need for manual feature extraction.

-> Can handle large amount of data & achieve high accuracy.

Disadvantages :-

-> Computationally expensive to train & require a lot of memory.

-> Can be prone to overfitting if not enough data or proper regularization is used

-> Requires large amount of labeled data.

-> Interpretability is limited, it's hard to understand what the network has learned.

Convolutional operation :-

What is Kernel / filter =>

The Kernel is a rectengular small matrix, which slides over the image from left to right and top to bottom.

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

what is stride ?

The no of pixels we slide over the i/p image by the Kernel is called a stride

what is Conv Opn =>

The Convolutional operation is the process of implying a Combination of two functions that produce the third function as a result, employing filters across the entire i/p image allows the filter to discover that feature of the image. which is also called a feature map.

A jigsaw puzzle is a perfect example of a Convolutional operation.

In a jigsaw puzzle, each piece has a portion of an img that reveals something about the complete picture when assembled.

like a jigsaw puzzle in Convolutional networks, multiple filters are taken to slice through the img and map them one by one and learn different portions of an input image.

Padding :

Adding extra borders around the i/p to control o/p size

Variants of basic Convolution function :

There are several variants of the basic Convolutional functin that are commonly used in DL, especially in CNNs.

Each variants is designed to improve performance, reduce computation or capture different features.

→ Dilated Convolution (Atrous Convolution)

→ Transposed Convolution

→ Depthwise Separable Convolution

→ Grouped Convolution

→ 1X1 Convolution (Pointwise Convolution)

→ Circular Convolution (Periodic padding)

→ Dynamic / Deformable Convolution

Structured O/P in CNN :-

Convolutional Neural Networks are not just for classifying images (like saying "this is a cat").

They can also be trained to produce detailed, structured o/p - where the prediction isn't a single label but something more complex.

A common eg is image segmentation, it's goal to label every singel pixel in the img with a class

That means the o/p of the model looks like the i/p image in size but each pixel contains a probability.

for each possible class.

How to keep o/p same size as i/p :

To make the o/p map the same size as the i/p, CNNs use same padding which means adding zeros around the i/p so that each convolution doesn't shrink it.

Smoothing the labels :

This is where the graphical models come in — they model the relationships bet^n pixels.

A popular CNN architecture for segmentation is the U-Net, especially in medical imaging. It's shaped like a "U" because it first shrinks the img to understand the big picture and then grows it back to get precise, pixel-level o/p

Efficient Convolution Algorithm :

① Depthwise Separable Convolution :

Depthwise Separable Convolution is a technique used in CNN to reduce the Computational cost and the no of parameters compared to standard Convolution operations. This technique is particularly useful in mobile vision applications where Computational resources are limited.

Depthwise separable Convolutions break down the Convolution operation into two Separate steps :

⊕ Depthwise Convolution :

~~in depthwise Convolution, each i/p channel is Convolved with a Separate filter of size~~
Convolution is a very imp mathematical opr. in ANN. CNNs Can be used to learn features as well as classify data with the help of img frames.

There are many types of CNN's.

One class of CNN's are depth wise separable convolutional neural network. These type of CNN's are widely used because of the following two reasons :-

→ They have lesser no of parameters to adjust as compared to the standard CNN's, which reduces overfitting.

→ They are computationally cheaper because of fewer computations which makes them suitable for mobile vision app.

LeNet :-

LeNet is a pioneering Convolutional neural network architecture that laid the foundation for modern image recognition, introducing key concepts like Convolution, Pooling & hierarchical feature learning.

Overview of LeNet =>

LeNet, Particularly LeNet-5, was developed by Yann Lecun and his Collaborators at AT&T Labs in the late 1980s & early 1990s, during a time when traditional methods heavily relied on hand-Crafted features for image processing.

LeNet was Specifically designed for handwritten digit recognition, making it one of the first Significant applications of CNN in practical Scenarios Such as banking & mail processing.

Architecture of LeNet-5 :-

The Archit- of LeNet-5 Consists of multiple layers that enable effective

feature extraction from images.
The primary layers include:

① Input layer =>

Accepts grayscale images typically sized at 32×32 pixels.

② Convolutional layer (C1 & C3) =>

The first Convolution layer (C1) uses 6 filters of size 5×5, producing 28×28 feature maps.

The second Con layer (C3) applies 16 filters of the same size, outputting 10×10 feature maps.

These layers automatically learn spatial hierarchies in the i/p data.

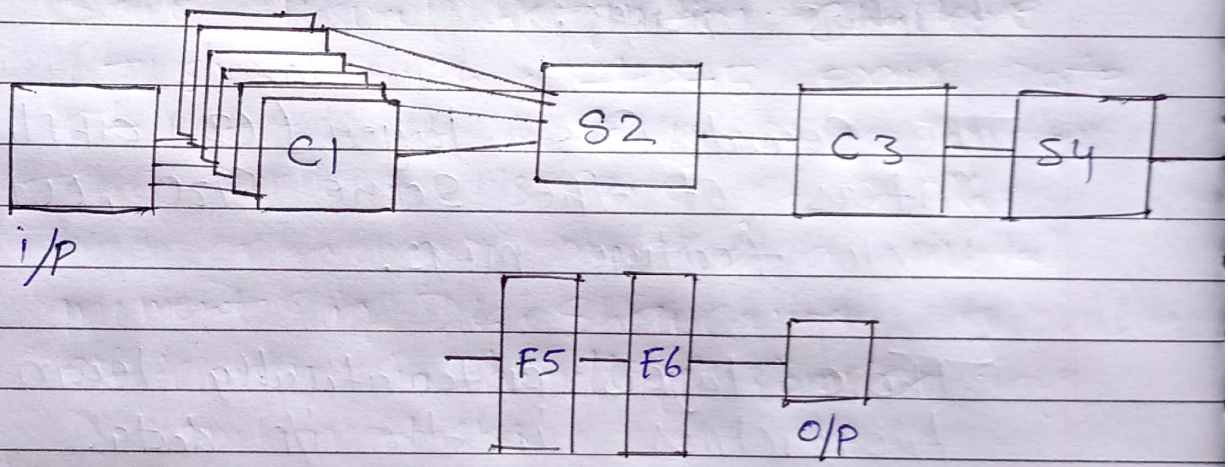
③ Pooling layers (S2 & S4) =>

After each Convolutional layer, average pooling layers reduce the dimensionality of the feature maps, increasing translation invariance and decreasing computational load. S2 reduces the size to 14×14 & S4 further reduces it to 5×5.

④ Fully Connected layers (F5 & F6) :-

The o/p from the pooling layers is flattened and passed through two fully Connected layers, with 120 & 84 nodes, respectively.

The final layer o/p probabilities for each of the 10 classes using a Softmax activation function.



Significance in DL :-

LeNet's intro popularized several key innovations within CNNs, including:

→ Convolution Opr :-

Allowing the network to automatically learn & extract relevant features from images.

→ Pooling Mechanism =>

Reducing the dimensionality of feature maps while retaining essential info, aiding in building translational invariance.

→ Hierarchical feature Learning =>

Enabling layers to learn from raw pixel values to a ~~reach~~ rich & abstract representation of visual data, which mimics human visual perception.

Applications :-

→ Handwriting recognition in postal services & banking

→ Object & face recognition in img & videos.

→ Autonomous driving systems for interpreting road signs & obj on the road.

AlexNet :-

AlexNet is a groundbreaking CNN archi- that revolutionized image classification by significantly outperforming previous models, highlighting the potential of DL.

Introduced in 2012 by Alex Krizhevsky, Ilya Suteskever & Geoffrey Hinton, AlexNet won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a top-5 error rate of 15.3% dramatically outperforming the runner-up's rate of 26.2%. This achievement marked a pivotal moment in DL, showcasing its efficiancy in handling Complex img recognition tasks.

Key innovations :-

- ReLU Activation =>

Accelerated training by 6 times Compared to tanh or Sigmoid function.

- Dropout =>

prevented overfitting by randomly

dropping neurons during training.

- Data Augmentation =>

Enhanced model generalization by artificially increasing the no of images through techniques like img translation & reflections.

Architecture =>

(1) i/p layer =>

Accepts RGB images sized 227 x 227 pixels.

(2) Convolutional layers =>

The initial layer uses 96 filters of size 11x11 followed by layers with increasing filter counts but smaller sizes, going down to 3x3 in deeper layers. This design helps extract essential features from images effectively.

(3) Pooling layers =>

Max-pooling is employed to downsample feature maps, reducing dimensionality while retaining critical spatial info.

④ Fully Connected Layers :-

- The last 3 layers are fully connected culminating with a Softmax layer that o/p probabilities for 1000 diff. classes.

Impact & Legacy :-

AlexNet was a big breakthrough in DL & inspired many other models like VGGNet, GoogLeNet, and ResNet.

It made Computer vision easier by letting models learn imp features from data on their own, instead of relying on people to manually pick out those features.

Applications :-

→ Medical imaging

→ Autonomous vehicles

→ Robotics

Unit 8 4
RNN

Recurrent Neural Network (RNNs) are specialized neural networks that excel in handling sequential data such as text, audio, or time-series.

Unlike traditional feedforward networks, RNNs have connections that loop back, allowing them to use o/p from previous time steps as i/p for future steps, thus integrating context from past data into current predictions.

Key Components & Architecture :-

- Memory :-
  The core feature of RNNs is their internal memory, which allows them to remember previous i/ps.

- Recurrent Connections :-
  RNNs are structured with loops in their architecture. At each time step, they process current i/p along with info. retained from previous i/p stored in hidden states.

- Hidden states :-
  This is a memory of

the network, continuously updated as the network processes each sequence element.

Types of RNNs :-

→ One - to - One (A single i/p produce single o/p)

→ One - to - Many

→ Many - to - One

→ Many - to - Many

Variants of RNNs :-

① LSTM =>

A special type of RNN that solves problems like forgetting info. too quickly. It uses gates to better remember things for a longer time.

② GRU => (Gated Recurrent Unit)

like LSTM but Simpler & faster. It still helps the network remember imp info, but with fewer parts.

③ Bidirectional RNN =>

This type of

RNN looks at the data both forward and backward, so it understands the context better.

Applications :-

→ NLP
→ Speech Recognition
→ Time-Series prediction.

Challenges :-

→ Vanishing & Exploding Gradient problems

→ Computational Complexity

Bidirectional RNNs :-

A Bidirectional Recurrent Neural Network extends traditional RNN by allowing sequential data to be processed in two directions: forward & Backward.

This dual processing capability enables the BRNN to capture info from both past & future contexts, providing a richer understanding of the i/p sequence

How BRNNs work :—

① Architecture =>

A BRNN Consists of two Separate RNN layers :

→ forward layer :
processes i/p sequence from the start to the end.

→ Backward layer :
Processes the same i/p sequence from the end to the start.

② Combining o/p =>
The o/ps from these two layers are typically combined to form the final o/p, which leverages the contextual info from both directions.

③ Applications =>

→ NLP
→ Machine Translation
→ Speech Recognition

Advantages :-

→ Enhanced Context understanding

→ Better performance

→ Flexibility.

Limitations :-

→ Increased Computational Cost

→ Real - Time Processing challenges

→ Greater Complexity in Training

Deep Recurrent Network :-

Deep Recurrent Networks use memory and step-by-step processing to understand ordered data like time series, text or Audio. They stack multiple RNN layers, which helps the model learn more detailed patterns & understand info at different levels & time steps.

Key Components of Deep RNN Architecture =>

- Recurrent Neurons =>

These are the core of RNNs and can remember past i/ps using a hidden state.

- Hidden State =>

Updated at each step to keep track of previous info & maintain context.

- Layer Stacking =>

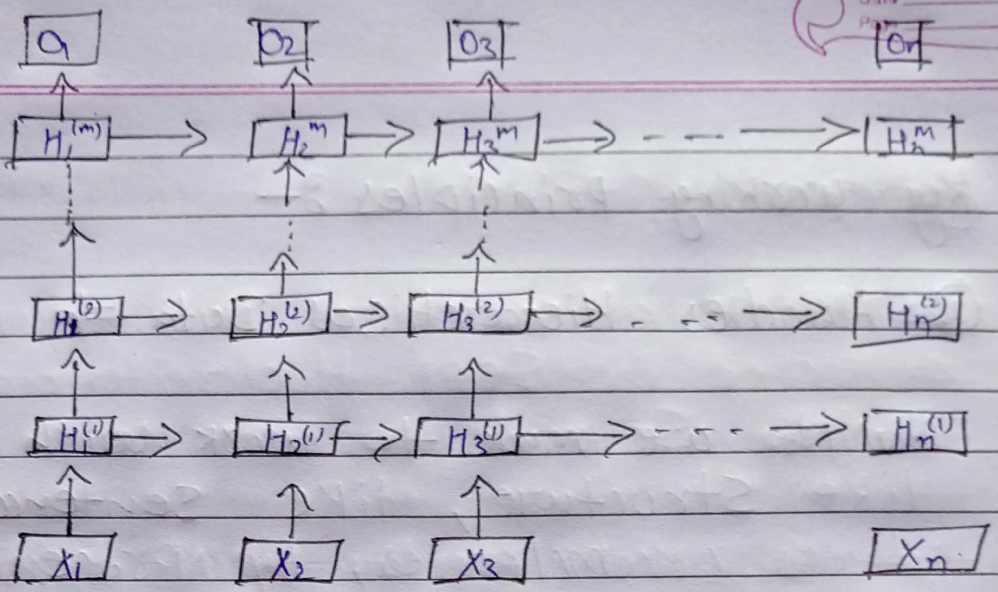
Multiple RNN layers can be stacked to learn more complex patterns

- Connection =>

O/Ps from one step influence future steps, helping the model understand sequence flow.

- O/p layer =>

Uses the final hidden state to make predictions, often with LSTM or GRU for better memory handling



Advantages :→

→ Hierarchical learning

→ Contextual Understanding

→ Versatility

Recursive Neural Network :-

Recursive Neural Networks (RvNNs) are models designed to work with tree structured data. Instead of processing data in a straight line, they combine smaller parts to build bigger parts. This helps them understand complex, nested relationships in the data.

Key working principles :-

① Handles Hierarchical Data =>

RvNNs are made to work with tree like structure, like sentence parse trees or molecules, by processing smaller parts and building up to larger ones.

② Shared parameters =>

They use the same set of rules at all levels, which helps the model work well across different parts of the structure.

③ Tree Traversal =>

RvNNs can move through the tree from the bottom up or top down, updating each part based on its connected parts.

④ Composition function =>

This ~~Comp fn~~ Combines info. from smaller parts to form a bigger part helping the model understand complex relationships.

Applications :-

→ NLP
→ Image Analysis

Difference bet" Recursive & Recurrent Neural Network =>

① Type of Data =>

Traditional RNN work with sequences like Time Series or text, using memory to track past steps.

Recursive N.N., on the other hand, handle tree-like or hierarchical data, such as sentence structures or molecules.

② How They Remember =>

Traditional RNNs use loops to remember earlier info. RNN (Recursive) don't rely on time steps - instead, they build relationships by combining parts of a structure like putting pieces of a puzzle together.

LSTM :-

LSTM is an advanced architecture developed by Sepp Hochreiter and Jurgen Schmidhuber in 1997.

It improves regular RNNs by helping them remember info over long sequences, which is useful for tasks like language modeling, speech recognition and time series prediction. Unlike standard RNNs, LSTMs use memory cells and gates to decide what info to keep or forget.

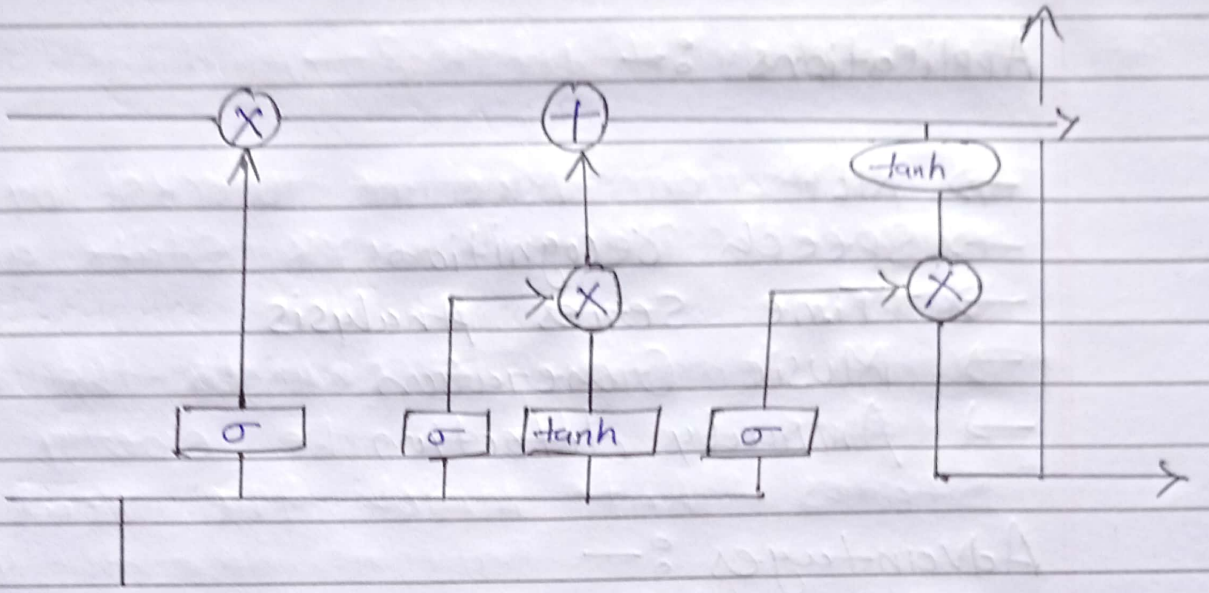
Architecture :-

The typical LSTM unit includes a cell state and three types of gates:

(i) Input Gate :-
Controls what new info is added to the cell state.

(ii) Forget Gate :-
Decides which info to discard from the cell state

(iii) o/p Gate :-
Determines what info to o/p from the cell state.



Structure of an LSTM Network

Working Mechanism :-

- Cell State =>
  Carries imp info through the sequence.

- Gates =>
  Use Special functions to decide what to keep or forget, with values bet" 0 & 1.

Together, these parts help LSTMs remember long term info, making them useful for many tasks.

Applications :-

→ NLP
→ Speech Recognition
→ Time Series Analysis
→ Music Generation
→ Anomaly detection

Advantages :-

→ Handle long term dependencies

→ Flexible

Disadvantages :-

→ Complexity

→ Resource Intensive.

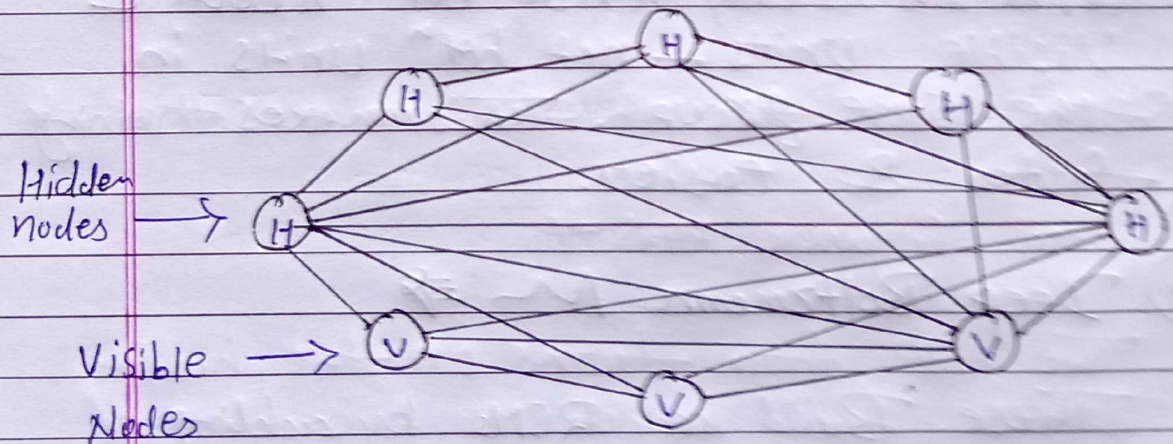
Unit : 5

Boltzmann Machine :-

A Boltzmann Machine is a network of Symmetrically Connected units that makes stochastic decisions about their states (on/off). Named after the physicist Ludwig Boltzmann, it operates based on principles from Statistical mechanics. The network includes two type of nodes :

(i) Visible Units :-
These units are used to i/p & o/p data.

(ii) Hidden units :-
These units learn patterns from i/p data, capturing the underlying structure in the data.



How does it work ?—

A Boltzmann Machine works by giving each possible network state an energy value.
During learning, it changes the connection weights to lower the energy for states that match the real data.

It updates the units based on probabilities linked to these energy levels, often using a method called Gibbs Sampling to estimate these probabilities.

Types :—

① Restricted Boltzmann Machine (RBMs) :

These are simpler versions where connections only exist bet" hidden & visible units — not bet" units in the same group. This makes training faster & easier.

② Deep Boltzmann M/c

These build on RBMs by adding multiple hidden layers, helping the model learn more complex patterns.

(2) Full Boltzmann M/c :-

These allow all units to connect with each other, but they are much harder & slower to train.

Applications :-

→ feature Extraction
→ Dimensionality Reduction
→ Recommendation Systems
→ Image Recognition.

Challenges :-

→ Computationally intensive
→ Training Complexity.

Restricted Boltzmann Machine :-

An RBM is a simple type of neural network with two layers :

- Visible layer => Takes in the i/p data.
- Hidden layer => Learns features or patterns from the data.

The word "restricted" means that units in the same layer don't talk to each other - only connections bet^n visible and hidden units are allowed. This setup makes learning faster and easier. RBMs don't need labeled data, so they're great for tasks like reducing data size or finding useful patterns.

How RBMs work :-

RBMs learn through two main steps =>

① Feed Forward pass =>

The i/p data turns on the hidden units. Each hidden unit is turned on or off based on a probability calculated from the i/p.

(ii) Feed Backward Pass :-

The hiddlen layer tries to rebuild the original i/p. The RBM adjusts its connections to make this reconstruction better.

Training RBMs :-

Training means improving the network by adjusting the weights betn units. RBMs use Contrastive Divergence (CD) to do this efficiently. A more advanced version called persistent ~~Cont~~ Contrastive Divergence (PCD) can improve learning by remembering previous states

Applications :-

→ Collaborative filtering

→ Image & Video processing

→ NLP

→ Bioinformatics

→ Financial modeling.

Advantages =>

→ Can find useful features in data without labels.

→ Can create new data that looks like the i/p

→ Easier to train than more complex models like full BM.

Disadvantages =>

→ Training can be tricky & slow

→ Needs careful tuning to work well

→ Hard to understand what ~~model~~ exactly the model is learning.