

Ensemble Learning

[Module-2 ASML]

Bagging vs Boosting??

Bagging Algo $\Rightarrow$

Bootstrap Sampling
Base Model Training
Prediction Aggregation
Out of Bag Evaluation
Final Prediction

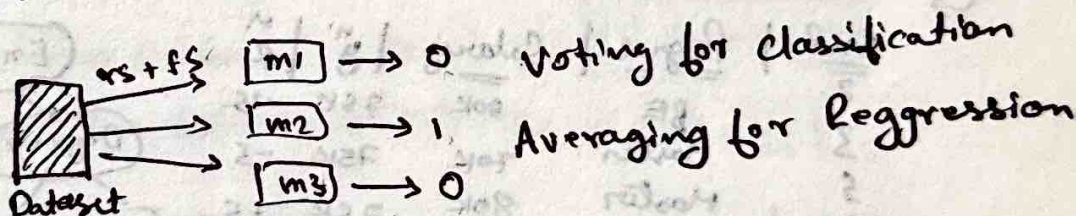
} Random Forest
Random Subspace Method

Boosting Algo $\Rightarrow$

Initialize Model weights
Train weak Learner
Sequential Learning
weight Adjustment

} Boosting
XG Boost
Ada Boost
Cat Boost

* Random Forest $\Rightarrow$



* AdaBoost $\Rightarrow$

- Calculate Gini Index of all features & get the sequence of stumps.

- Amount of say $= \frac{1}{2} \log_2 \left[\frac{1 - \text{Error}}{\text{Error}} \right] \Rightarrow$ for 4 features

- updates weights $\uparrow = \text{weight} \times e^{\text{Amount of say}}$
 $\downarrow = \text{weight} \times e^{-\text{Amount of say}}$

- Normalize weights $\Rightarrow \frac{\text{weight}}{\text{total weight sum}}$

- Create Bins

- Create New dataset (this will add most of wrong)

- Select features Again using Gini Index. Repeat Process

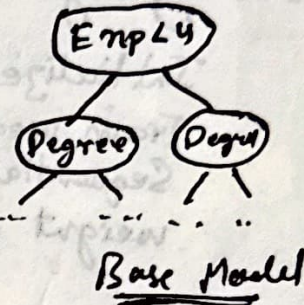
- We can create, n estimators.

* Gradient Boosting $\Rightarrow$

- Avg is calculated for Prediction
- Residuals are calculated.
- They are updated according to the feature Separation (R_1)
- Prediction is made $\Rightarrow$ Actual value + α (Avg + R_1) $\} R_2$
 $\hookrightarrow$ learning rate
- R_2 is used as residual for next tree.

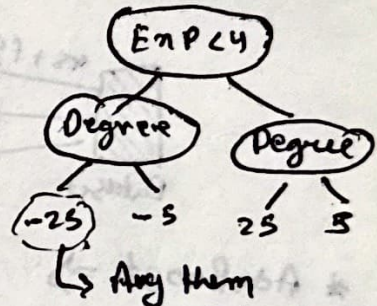
$\Rightarrow$

Emp	Degree	Salary	$\hat{y}$
2	BE	50K	75K
3	Master	70K	75K
5	Master	80K	75K
6	PHD BE	100K	75K



(Residual 1

Emp	Degree	Salary	$\hat{y}$	$\hat{r}$
2	BE	50K	75K	-25
3	Master	70K	75K	-5
5	Master	80K	75K	+5
6	BE	100K	75K	25



(Learning Rate Added

Emp	Degree	Salary	$\hat{r}$	$\hat{r}_2$	$\hat{r}_2 = \hat{y} + 0.1(\hat{y} + \hat{r})$ (α)
2	BE	50K	-25	55	
3	Master	70K	-5	84	
5	Master	80K	5	88	
6	BE	100K	25	112.5	

(Reevaluate Residual for Next DT

Emp	Degree	Salary	$\hat{r}_2$
2	BE	50K	5
3	Master	70K	14
5	Master	80K	8
6	BE	100K	12.5

Repeat Process $\rightarrow$ Avg
 $f(n) = \text{Base Model} + \alpha \sum \text{D.T}$

Hyper Parameter Tuning

Grid Search & Random Search

Hyperparameters are parameters that are not learned during training they need to set beforehand.

⇒ Grid Search Tries out all possible combination of Hyperparameter from their pre-defined space. → Optimal

Random Search samples the combination randomly from the pre-defined space. → Not guaranteed optimal

Unsupervised Learning

Clustering Association rule learning Dimensionality reduction

* Clustering ⇒ grouping data together that are similar in some criterion which you choose.

$$\text{Euclidean distance} = d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

$$\text{Manhattan distance} = d(x, y) = \sum |x_i - y_i|$$

types ⇒ Agglomerative • dendrogram way of visualizing clusters •
Divisive

(i) Agglomerative Clustering ⇒

(i) start with each data point as one cluster.

(ii) compute pairwise distance & create distance matrix

(iii) merge the closest clusters.

(iv) update Distance Matrix

(v) Repeat till one cluster remains.

Linkage Method

• min • max • Average • Ward's method

Single

complete

(ii) DBSCAN $\Rightarrow$ Density Based Spatial Clustering of Application with noise.

- Choose two hyper parameter.

ϵ = radius of cluster to be neighbor

minPts = minimum number of points required for core.

Classify Points $\left\{ \begin{array}{l} \bullet \text{ Core Points} = \text{minPts at } \leq \text{distance of } \epsilon \\ \bullet \text{ non Core Points} = \text{within } \epsilon \text{ but has minPts less in their neighbors} \\ \bullet \text{ outlier} = \text{none} \end{array} \right.$

- Assign Core & Non Core points to cluster, if remains core point start new cluster.

* $\Rightarrow$ Cluster Quality $\Rightarrow$

- Rand-Index =
$$\frac{TP + TN}{TP + TN + FP + FN} \cdot \frac{a + b}{n \cdot c_2}$$

$RI = 1 \rightarrow$ Perfect

$RI = 0 \rightarrow$ Random

RI higher is better

Cohesion $\Rightarrow$ avg distance bc centroid of cluster points,

Separation $\Rightarrow$ min distance between centroid of other cluster point.

• Silhouette Coefficient $\Rightarrow \frac{b - a}{\max(b, a)}$

• higher the score better it is.

$\int$ for one sample
for whole clusters mean is taken.

• Shows dense & separated clusters

b = avg distance between cluster

a = avg distance within cluster

Dimensionality Reduction

Reducing number of features in a dataset while preserving as much information we want (variance) in a possible standard.

* PCA - Principal Component Analysis

Reduction technique which reduces high dimension data into lower dimension.

- Standardize the data.

$$X_{\text{norm}} = \frac{X - M}{\sigma} \rightarrow \text{mean} \rightarrow \text{S.D}$$

- Compute Covariance Matrix.

$$C = \frac{1}{n} X^T X \text{ or } \text{Cov}(\text{features})$$

- Compute Eigen Vector & Values.

$$Cv = \lambda v \rightarrow \begin{matrix} \text{vector} \\ \downarrow \\ \text{values} \end{matrix}$$

- Sort Eigen vectors in descending eigen values.

- Select top K ($\sum \frac{\lambda_i}{n} = \text{Variance in one feature}$)

$$X' = X V_K$$

$\hookrightarrow$ Eigen vector with Top K Eigen Vectors.

Regularization

Regularization of any machine learning model reduces overfitting of the model on training dataset.

Linear Regularization $\rightarrow$ Ridge, Lasso & Elastic

$$\text{Ridge} = \text{RSS} + \lambda \sum_{i=1}^n w_i^2 \rightarrow \text{L2 Regularization}$$

$$\text{Lasso} = \text{RSS} + \lambda \sum_{i=1}^n |w_i| \rightarrow \text{L1 Regularization}$$

Elastic Net with L1 & L2

Performance Metrics - Regression

* MAE $\rightarrow$ Mean Absolute Error $= \frac{1}{n} \sum |y - \hat{y}|$

MSE $\rightarrow$ Mean Squared Error $= \frac{1}{n} \sum (y - \hat{y})^2$

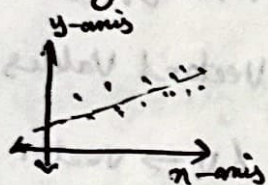
RMSE $\rightarrow$ Root Mean Squared Error $= \sqrt{\frac{\sum (y - \hat{y})^2}{n}}$

R^2 Score $\rightarrow$ R Squared $\downarrow$

$$R^2 = 1 - \frac{\sum (y - \hat{y})^2 \rightarrow \text{Error Lin}}{\sum (y - \bar{y})^2 \rightarrow \text{Error } \bar{y}}$$

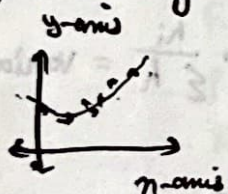
* Regressions $\Rightarrow$

Linear Regressions $\Rightarrow$



$$y_i = \overset{\text{intercept}}{b_0} + b_1 x_1 + b_2 x_2 + \dots + b_n x_n + \underset{\substack{\text{Bias} \\ \leftarrow E_0}}{E_0}$$

Polynomial Regression



$$y = \overset{\text{intercept}}{b_0} + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_n^n + \underset{\substack{\text{Bias} \\ \leftarrow E_0}}{E_0}$$

$\downarrow$ Capturing Polynomial features of Every feature.

* Performance Metric for classification $\Rightarrow$

Confusion Matrix =

	True	
	1	2
Predicted	1	2
	3	7

	True	
	1	2
Predicted	1	2
	3	7

- Convert Multiclass CF into single class for each feature.
- Calculate metrics of the Avg out the values.

- Precision = $\frac{TP}{TP + FP}$

- Recall = $\frac{TP}{TP + FN}$

- F1 = $2 \times \frac{Pr \times Rec}{Pr + Rec}$

	1	2	3
1	8	2	0
2	1	9	0
3	1	2	7

⇒

	1	NO
1	8	2
NO	2	18

	2	NO
2	9	1
NO	4	16

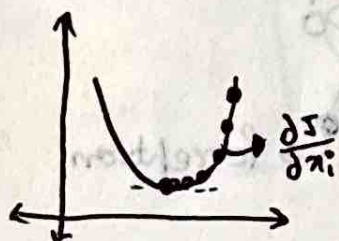
	3	NO
3	7	3
NO	0	20

Binary CF's for Each Feature

- ROC-AUC

- Accuracy = $\frac{TP + TN}{TP + FP + TN + FN}$

* Gradient Descent ⇒



Updating of weights in linear regression will fit the model to the training data. but updating it with best strategy makes difference.

Steps for updation ⇒

- Calculate slope of Cost function with respect to coefficient you want to update.

$$\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1}, \frac{\partial J}{\partial \beta_2}, \dots, \frac{\partial J}{\partial \beta_n}$$

- update Coefficient with desired learning rate.

$$\beta_0 = \beta_0 - \alpha \left(\frac{\partial J}{\partial \beta_0} \right)$$

↳ learning rate

- update J.

- Repeat Process.

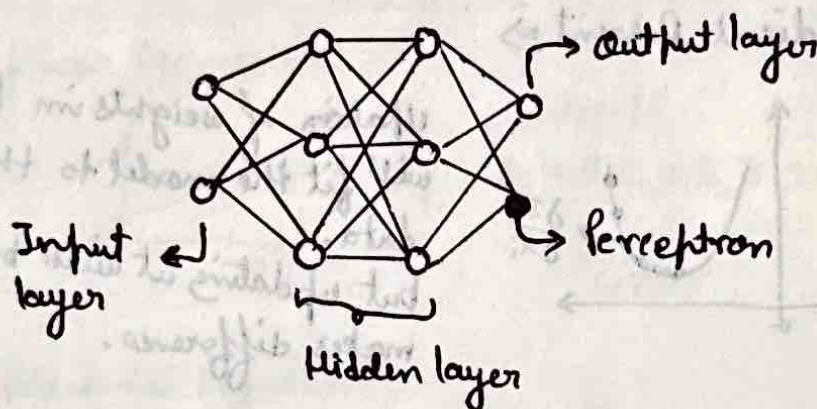
Ex:- $J = \frac{1}{2m} \sum (y_i - (w_1 x_{1i} + w_2 x_{2i} + b))^2$

$$\frac{\partial (J)}{\partial w_1} = -\frac{1}{m} \sum_{i=1}^m x_{1i} (y_i - \hat{y}_i), \quad \frac{\partial (J)}{\partial w_2} = -\frac{1}{m} \sum_{i=1}^m x_{2i} (y_i - \hat{y}_i)$$

Artificial Neural Network (ANN) [Module-3]

ANN are neural network composed of many nodes layers where each node is a perceptron which acts like human neuron. Each perceptron uses linear regression model & an activation function to predict output. Data is passed between perceptrons in feed forward.

- Weights & Bias of perceptron is tuned using training data.



There are many types of ANN $\Rightarrow$

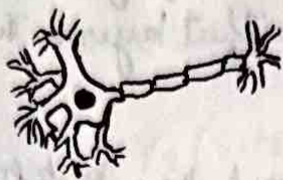
ANN, CNN, RNN, LSTM-NN

$\downarrow$ $\downarrow$ $\downarrow$
(Convolutional
neural network) Recurrent
Neural Network Long Short
Term Memory
Neural Network

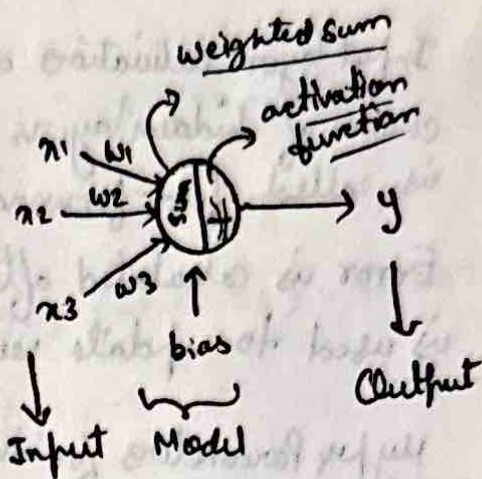
* Perceptron Model $\Rightarrow$

Perceptron is an artificial neuron in machine learning which imitates like a human which takes inputs from other neuron or parameters and activates by using weights of their value or activation.

Used for $\Rightarrow$ Binary Classification
Pattern recognition
Linear Separation Problem

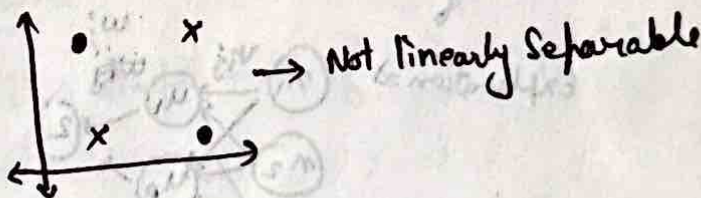
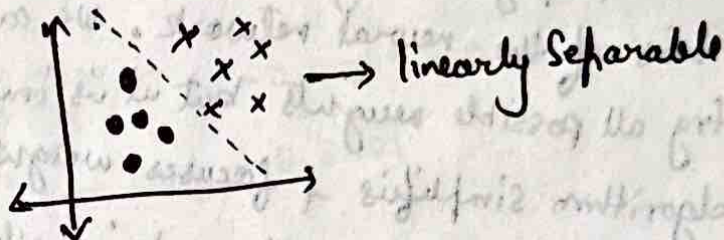


Human Neuron

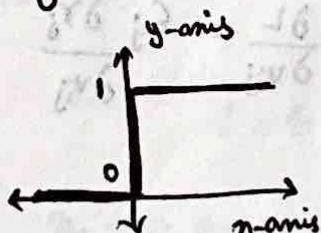


* Activation function decides whether the neuron is activated or not, converts value in range 0 to 1.

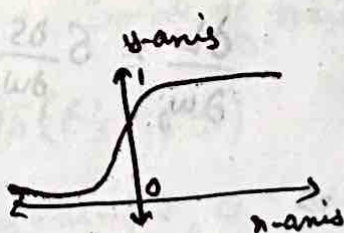
* Linearly Separable Problems $\Rightarrow$



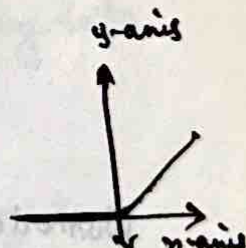
• Types of Activation function



Step function



Sigmoid function



ReLU function

Multi Layer Perceptron

Multi layer Perceptron is a model where multiple layers of neurons (Perceptron) are used to solve the problem. There are mainly three layers Input layer, Middle layer & Output layer.

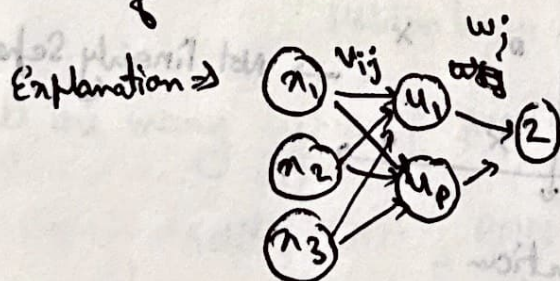
Input layer activations are passed on to fully connected chain of hidden layers then to output layer. This flow is called feed-forwards.

Error is calculated after feed forward then back propagation is used to update weights & biases.

Hyper Parameters for this model is number of hidden layer, Neuron count in each layer.

Backpropagation Algorithm

Backpropagation Algorithm is used to update weights and bias of the neural network. We can keep on and try all possible weights but it is impractical. This algorithm simplifies & focuses weight updation towards reduction of Error. Uses chain rule at its core.



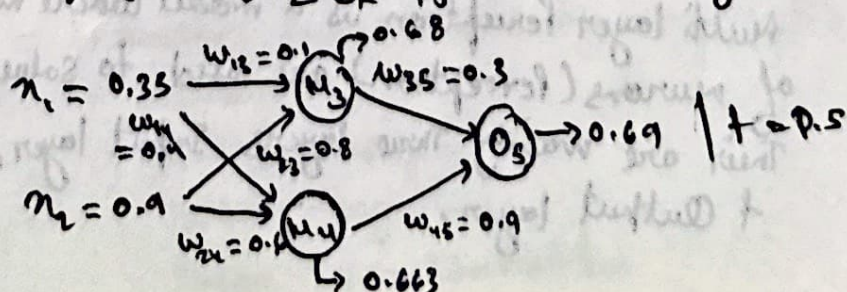
$$\frac{\partial L}{\partial w_{ij}} = \delta \frac{\partial S}{\partial w_{ij}}, \quad \frac{\partial L}{\partial v_{ij}} = \epsilon_j \frac{\partial r_j}{\partial v_{ij}}$$

Numerical $\Rightarrow$

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j (1 - o_j) (t_j - o_j) \rightarrow \text{Output layer}$$

$$\delta_j = o_j (1 - o_j) \sum \delta_k w_{kj} \rightarrow \text{hidden layer}$$



$$\delta_5 = y(1-y)(t-y)$$

$$= 0.69(1-0.69)(0.5 - 0.69) \Rightarrow -0.0406$$

$$\delta_3 = y_3(1-y_3)w_{35} \times \delta_5$$

$$= 0.68(1-0.68)(0.3 \times -0.0406) = -0.00265$$

$$\delta_4 = y_4(1-y_4)w_{45} \times \delta_5$$

$$= 0.6637(1-0.6637)(0.9 \times -0.0406) = -0.0082$$

$$\bullet \Delta w_{45} = \eta \delta_5 y_4 = 1 \times -0.0406 \times 0.6637 = -0.0269$$

$$w_{45} = w_{45} + \Delta w_{45} = 0.8731$$

$$\bullet \Delta w_{34} = \eta \delta_4 y_3 = 1 \times -0.0082 \times 0.35 = -0.00287$$

$$w_{34} = w_{34} + \Delta w_{34} = 0.3971$$

Gradient Descent

Gradient Descent is a simple rule of weight updation using slope of error function.

- Stochastic Gradient descent $\Rightarrow$ updating weights for each sample of training data

$$\theta = \theta - \eta \nabla J(\theta; x_i, y_i)$$

x_i, y_i are training example, θ = Weight

J is cost function.

- Batch Gradient descent $\Rightarrow$ errors with the same weights is cumulatively used for weight updation. multiple samples are first evaluated for error.

$$\theta = \theta - \eta \nabla \sum_{i=0}^n J(\theta; x_i, y_i)$$

- mini-batch Gradient descent $\Rightarrow$ uses a middle ground between batch & Stochastic where weight updation is done when a small batch is completed.

$$\theta = \theta - \frac{\eta}{m} \sum_{i=1}^m \nabla J(\theta; x_i; y_i)$$

* When & where are these different methods used $\Rightarrow$

Small dataset $\rightarrow$ Batch GD

large dataset $\rightarrow$ Stochastic GD

Difference between BGD, SGD, MBGD

BGD $\rightarrow$ • After computing the Gradient over the entire dataset.

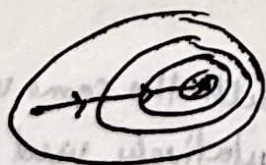
- More Stable Convergence
- Slow for large dataset.

SGD $\rightarrow$ • After computing the gradient for each individual

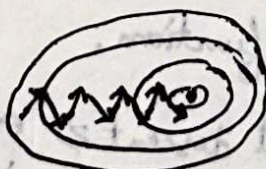
- Faster updates, better for large dataset
- High variance, noisy updates.

MBGD $\rightarrow$ • After computing small batch of dataset

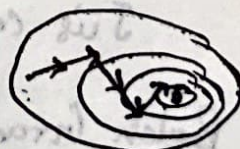
- Balance of speed and stability.
- Requires tuning of batch size.



BGD



SGD



MBGD

Vanishing Gradient

Vanishing Gradient is a common problem encountered when training of deep neural network with gradient based weight updation rule. The main problem is very small change in weights is seen when training because of very small value of Gradient. Which leads to no updation of weights.

Mainly encountered while using activation function

⇒ Sigmoid & Tanh

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial z} \times \frac{\partial z}{\partial a_1} \times \frac{\partial a_1}{\partial w_{11}} \quad \left| \quad w_n = w_0 - \eta \frac{\partial L}{\partial w_{11}} \right.$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
0.001 0.0001 0.00001 0.000001

near to zero

⇒ Overall no Gradient.

How to recognize ⇒ Loss vs Epochs
Weights vs Epochs

How to solve it ⇒ reduce model complexity
use ReLU activation function or other

Exploding Gradient

In exploding gradient the gradient grows exponentially large as they propagate backward through the network layers. Happens when multiple weights have weight greater than 1, gradients accumulate and multiply, leading to exponential growth. Causing divergence rather than converging.

Symptoms

⇒ weight in less time becomes large

fails to converge

loss increases to infinity

Overfitting Problem

Overfitting Problem in neural network occurs when a model learns the training pattern so comprehensively including noise and irrelevant data that it fails to generalize on new, unseen data. Leads to high training performance and low validation.

Cause $\Rightarrow$

- Insufficient data.
- Excessive Model Complexity.
- Long Training Time.

Solution $\Rightarrow$

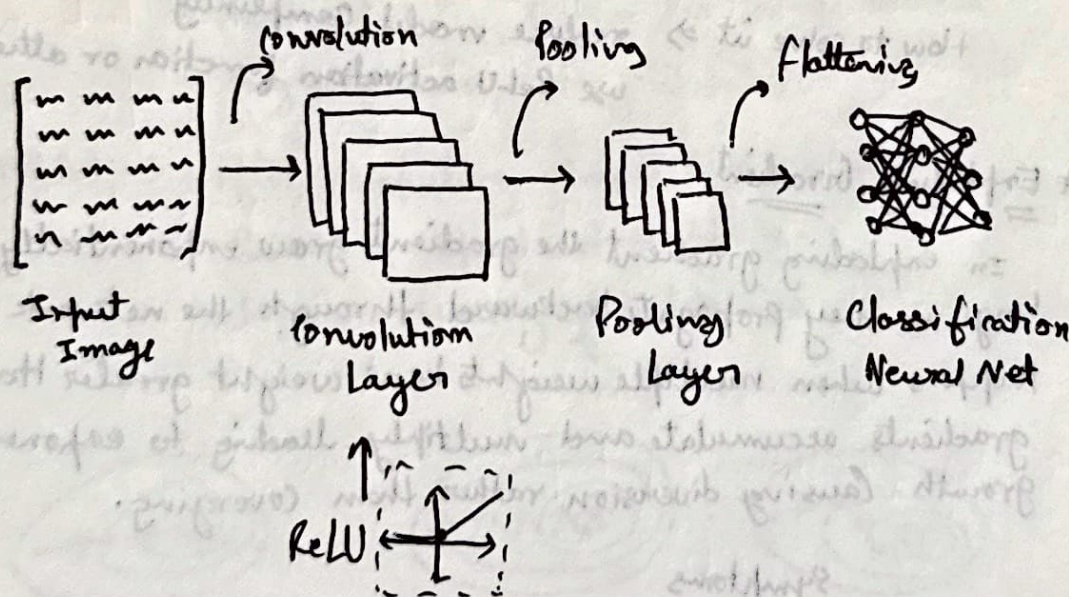
- Model simplification
- Data Augmentation
- Regularization

$\rightarrow L_1, L_2$
 $\rightarrow$ Dropout

Random removal of neuron during training

- Early stopping

Convolutional Neural Network



Input Image $\Rightarrow$ A 2D array or a Matrix representation of an Image. Where all pixel values are defined in matrix.

Convolutional layer $\Rightarrow$ Convolutional layer is used to extract features from Input Image. Kernel or filter is used to extract different features in an Image portion, the filter is slid over whole image to find patterns.

filters detects $\Rightarrow$ edges, blur, object parts etc.

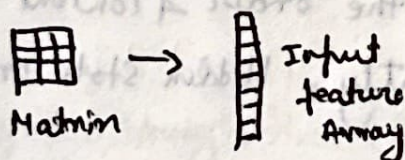
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

$\downarrow$ Input Image $\downarrow$ Kernel or Filter $\downarrow$ Feature Map

(general)
(same size)

Max-Pooling $\Rightarrow$ Max-Pooling is a downsampling technique used for size reduction of feature maps while preserving information.

Flattening $\Rightarrow$ Transforming 2D Matrix into 1D Array.



ANN $\Rightarrow$ Neural Network for classification is used on input feature array.

Formulas $\Rightarrow$ Output Size =
$$\frac{(\text{Input Size} - \text{Kernel Size} + 2 \times \text{padding})}{\text{Stride}} + 1$$
 (convolution)

Output Size =
$$\frac{(\text{Input Size} - \text{pool size})}{\text{Stride}} + 1$$

Underfitting $\Rightarrow$ Increase low sizes

Overfitting $\Rightarrow$ Increase Maxpooling

Sequence Learning

[Module - 4]

Sequence learning problem involves training model to process and predict data where the order and timing of elements matter, requires variable length inputs and outputs while capturing dependencies across time steps.

Variable length

Memory Requirement

Challenges $\Rightarrow$ Variable length Input
Vanishing Gradients

RNN's, LSTM, GRU, Transformers all are sequential learning models.

RNN

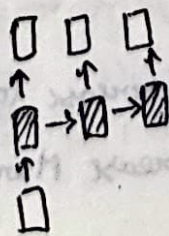
Recurrent Neural Network deep learning model for sequential data, where the order & content matters.

Each input, independently, Hidden state or memory is ^{is not treated} used to process them as sequence and retain info.

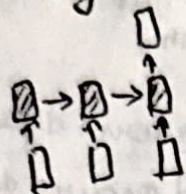
- Sequential processing $\rightarrow$ Input data one at time, updating H.S.
- Hidden state $\rightarrow$ vector acts like memory.
- feedback loop $\rightarrow$ output of previous step fed back like memory.

RNN's $\Rightarrow$ One to One

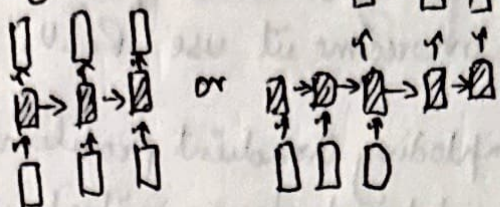
One to Many



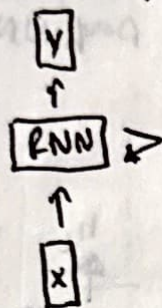
Many to One



Many to Many



Hidden state updation RNN $\Rightarrow$



$$h_t = f_{uw}(x_t, h_{t-1})$$

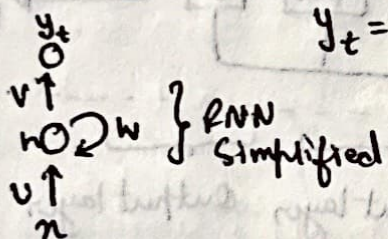
$$h_t = \tanh(Ux_t + Wh_{t-1}) \rightarrow \text{hidden state updation}$$

$$y_t = \text{SoftMax}(Vh_t)$$

$\rightarrow$ Prediction using hidden state

forward pass eqns $\Rightarrow h_t = \tanh(Ux_t + Wh_{t-1})$

$$y_t = \text{SoftMax}(Vh_t)$$

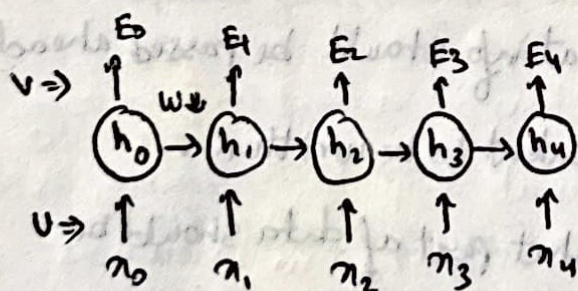


loss function $\Rightarrow$
(cross entropy)

$$E(y_t, \hat{y}_t) = - \sum_t y_t \log \hat{y}_t$$

y_t actual $\hat{y}_t$ Predicted

Back Propagation through time (BPTT) $\Rightarrow$



$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \prod_{j=k+1}^3 \left(\frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

$$\frac{\partial E_3}{\partial U} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \prod_{j=k+1}^3 \left(\frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U}$$

(Similarly like $\frac{\partial E_3}{\partial W}$)

final BPTT for

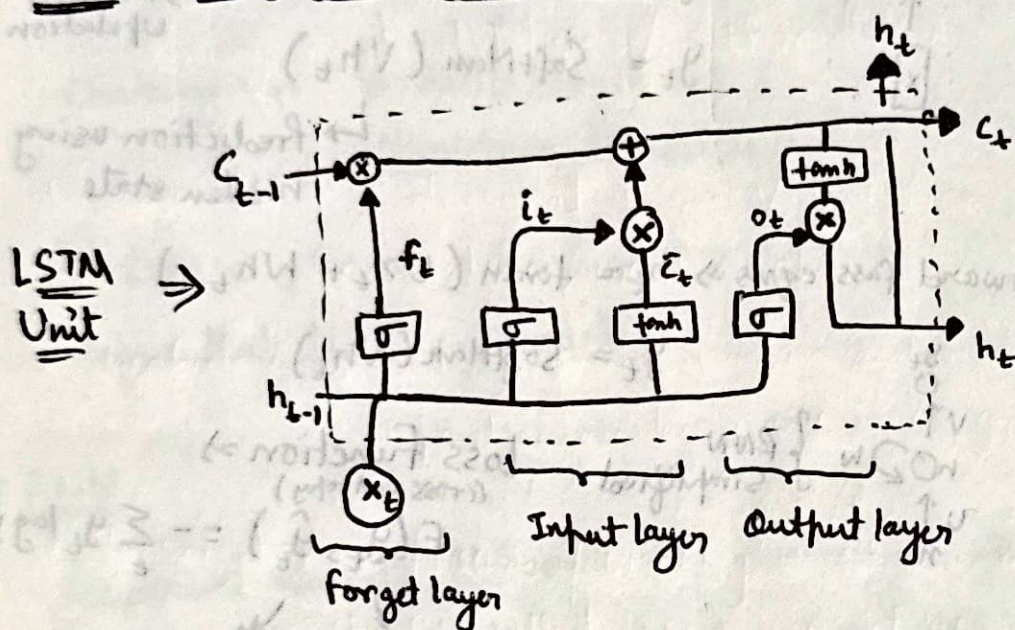
$\frac{\partial E_3}{\partial W} \rightarrow$ used in weight updation

Vanishing Gradient problem in longer sequences,
to overcome it use ReLU, Regularization.

Exploding Gradient problem when long term dependency
is captured, large initial weights and occurs during
BPTT.

Q $\Rightarrow$ Difference between RNN and Deep NN??

LSTM - Long Short Term Memory



$c_t \Rightarrow$ cell state, forget gate: control for what info should
be holded further.

Input gate: Controls what info should be passed ahead.

cell content: New cell content to be written.

Output gate: Controls what part of data should be
outputted.

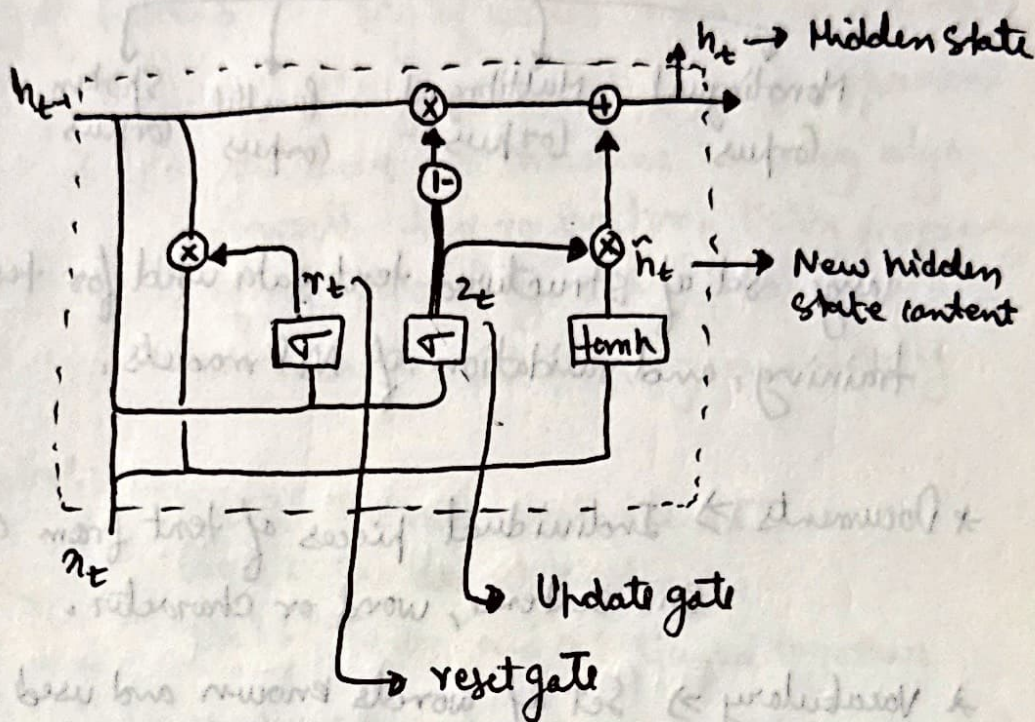
Sigmoid & tanh functions are used.

Vanishing Gradient $\Rightarrow$ Solved using cell state $\oplus$, Gates.

Exploding Gradient $\Rightarrow$ Solved using Bounded activation, Gates.

Q $\Rightarrow$ Difference between LSTM & RNN?

Gated Recurrent Unit - GRU



Hidden state: Info kept from previous input & what is new.

New hidden state content: selects useful info and creates new info

reset gate: Controls what info should be kept from prev.

update gate: Controls what part should be preserved, and used for new hidden state.

(i) No Internal memory like LSTM only h_t .

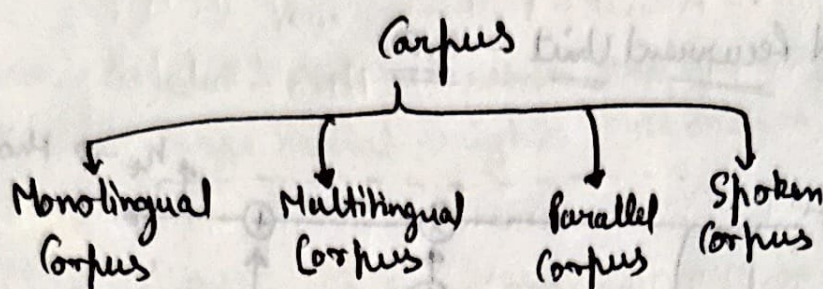
(ii) No output gate.

(iii) fewer parameters to learn.

NLP - Natural Language Processing

Natural Language Processing (NLP) enables computers to understand, interpret, and generate human language using machine learning and computational linguistics.

* Corpus $\Rightarrow$



large set of structured text data used for testing, training, and validation of NLP models.

* Documents $\Rightarrow$ Individual pieces of text from corpus
Ex: Sentence, word or character.

* Vocabulary $\Rightarrow$ Set of words known and used within a particular language or by group user.

* Tokenization $\Rightarrow$ Process of breaking ^{text} words into smaller chunks called tokens, might be word, subword and characters.

"I Love None." $\rightarrow$ ["I", "Love", "None", "."]

* Lemmatization $\Rightarrow$ Lemmatization is a text preprocessing technique in natural language processing that reduces word to their root form known as lemma, based on meaning context. Unlike stemming which chops off ending.

"better" $\rightarrow$ "good"

* Stemming $\Rightarrow$ Text preprocessing technique in NLP where word is converted to root form by chopping prefixes & suffixes.

"running" → "run", "runs" → "run"

"Universities" → "Univers" — (x)

* Bag of words → Bag of words model is a simple and widely used technique to represent text data for machine learning algo, converts text to vector. Using frequency of each word ignoring grammar and word order, but keeping multiplicity

Ex:-

Doc1: The cat sat on the mat

Doc2: The dog sat on the log

Doc3: The dog and cat played together

→ Vocab → [The, cat, sat, on, mat, log, dog, and, played, together]

Vector Representation →

	Doc1	Doc2	Doc3
the	2	2	2
cat	1	0	1
sat	1	1	0
on	1	1	0
mat	1	0	0
dog	0	1	1
log	0	1	0
and	0	0	1
played	0	0	1
together	0	0	1

BoW Representation →

Doc1 → [2, 1, 1, 1, 1, 0, 0, 0, 0, 0]

Doc2 → [2, 0, 1, 0, 1, 1, 0, 0, 0]

Doc3 → [2, 1, 0, 0, 0, 1, 0, 1, 1, 1]

* TF-IDF $\Rightarrow$ Term Frequency - Inverse Document Frequency.

Reflects how important a word is in a document relative to all documents in the corpus.

TF - Term frequency :- measures how frequently a term appears in a document.

$$TF(t, d) = \frac{\text{Number of times } t \text{ appears in } d}{\text{Total number of terms in } d}$$

IDF :- Measures how important a term is across all documents.

$$IDF(t) = \log \left(\frac{\text{Total number of documents}}{1 + \text{Number of docs containing } t} \right)$$

$$TF-IDF = TF(t, d) \times IDF(t)$$

Doc 1	Doc 2	Doc 3	
5	5	5	the
1	0	1	to
0	1	1	be
0	1	1	no
0	0	1	can
1	1	0	good
0	1	0	bad
1	0	0	low
1	0	0	high
1	0	0	together

Doc 1 $\Rightarrow$ [5, 1, 0, 0, 0, 1, 0, 0, 1, 1]
 Doc 2 $\Rightarrow$ [5, 0, 1, 1, 0, 1, 1, 0, 0, 0]
 Doc 3 $\Rightarrow$ [5, 1, 1, 1, 0, 0, 0, 0, 1, 1]