MODI

I) What is ML?

→ It is Programming computer to optimize a performance Criterion

→ It is the science of programming computers so they can learn from data

→ It is a subfield of AI

→ Widely used in image classification, clustering, etc

→ Appropriate

- When rules cannot be written manually
- When patterns keep changing over time
- When large amount of data are available

→ Not to use

- When the task is simple and rule-based
- When you don't have enough data
- When accuracy must be perfect

Steps

1) Data collection
2) Data preparation
3) Choosing a model
4) Training
5) Evaluation
6) Parameter Tuning
7) Prediction.

II) Type of Learning

Topic	Supervise Learning	Unsupervise Learning	Semi-supervise Learning	Reinforcement Learning
Data used	Labeled data	Unlabeled data	Few labeled + many unlabeled samples	No label, Uses reward signals
Goal	Predict output of unseen data	~~Learns mapping~~ Find hidden structure / patterns	Improves accuracy when labels are Limited	Learn best actions (Policy)
How it Learns	Learns mapping from input → output	Groups data based on similarity	Combines supervised + unsupervised	Agents interacts → reward or Punishment
Example	Classification, Regression	Clustering, PCA	Web page classification, Speech labelling	Robotics, Games

III) Bias & Variance

→ Bias

- Refers to error that come from overly simple assumptions made by the model, a high bias model underfits the data
- Characteristics
  - Model is too simple
  - Ignores important pattern
  - High training error
  - Poor performance

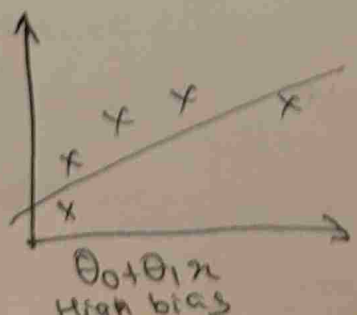
→ Variance

- Refers to the error caused by the model being too sensitive to the training data, a high variance model overfits the data
- Characteristics:
  - Model is too complex
  - Learns noise along with patterns
  - Low training error but high test error
  - Poor generalization

IV) Overfitting & Underfitting

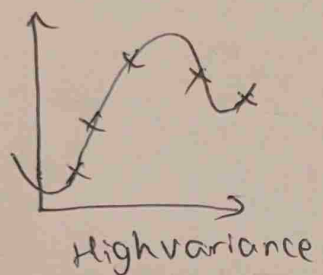
→ Underfitting

- Happens when a model is too simple & fails to learn the pattern in the training data
- Characteristics
  - High bias
  - High error on both traing & test data
  - Model does not capture complexity
- How to reduce it
  - Use a more complex model
  - Reduce regularization
  - Train for more epoch
  - Add more relevant features



→ Overfitting

- Happens when a model is too complex & learns noise in the training data instead of general patterns
- Characteristics
  - High variance
  - very low training error but test error
  - Model memorizes the dataset
- How to reduce
  - Use regularization (L1/L2)
  - Use Dropout in neural networks
  - Use Early stopping
  - Use Cross-validation
  - Collect more training data
  - Reduce model complexity



VI) Cross Validation

→ It is a statistical technique used in ML to evaluate how well a model generalizes to unseen data.

→ Instead of using a single train-test split, cross validation uses multiple splits of data set to ensure that every data point gets a chance to be in both training & testing sets

→ It provides a more reliable estimate of the model's performance

→ Need

- Single train-test split is unreliable
- Helps detect overfitting
- Works well on small datasets
- Provides a stable performance estimate

→ Working

1) Shuffle the dataset
2) Split the data into 'K' equal parts called "folds"
3) For each fold i:
   - Use i as test ~~test~~ set
   - Use K-1 as training set
4) Train the model on training fold
5) Test the model on test fold
6) Repeat for all K folds
7) Avg. all scores → Final accuracy / performance

→ Types

1) K-Fold
2) Stratified K-Fold
3) Leave-one out
4) Hold Out Validation
5) Time Series

→ Advantage

- Reduces bias
- Reduces variance by averaging K results
- Work well with small dataset
- Detects overfitting
- Uses the entire dataset for both training & testing

→ Disadvatages

- Computationally Expensive
- Not ideal for time series
- very slow for large dataset.

VI) Feature Engineering

→ It is the process of transforming raw data into meaningful input features that help machine learning models learn better patterns
→ It involves creating, modifying, selecting & preprocessing features to improve the performance & accuracy.

→ Steps

A) Data cleaning
- Handling missing values
- Handling outliers

B) Feature Construction:
- Polynomial features: x^2, $\sqrt{x}$
- Ratios: price / quantity
- Date features
- text features

→ Importance

- Improves model accuracy
- Reduces overfitting by removing irrelevant features
- Reduces training time
- Handles missing or noisy data
- Helps convert non-linear relationships into linear form

→ Challenges

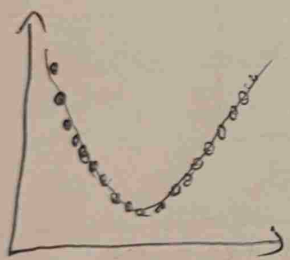
- Requires Domain Knowledge
- Time consuming
- Not always automated

VII Gradient Descent

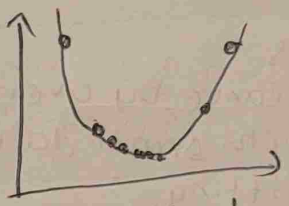
→ It is used to find the optimal values of model parameters that minimizes cost/loss function.

→ The goal is to take steps downhill until the bottom is reached

For a given ex



No Gradient Descent



Gradient Descent

→ Gradient descent works by taking big steps till it doesn't get closer to the optimal level at that point it will take small steps till it ~~doe~~ either doesn't reach the optimal point or the max no° of Gradient descent has reached

→ Steps.

1) Initialize parameter: $w = 0$ or small random value
2) Compute the cost function: cost = $J(w)$
3) Compute the gradient (That is the derivation) $\frac{\partial J(w)}{\partial w}$
4) Update the parameters: $w = w - \alpha \frac{\partial J(w)}{\partial w}$
5) Repeat until convergence

→ Importance of learning rate as it controls the step size, too small → slow, too large → may overshoot it

~~→ Advantage~~

→ Advantage
- Simple and easy to implement
- Works for large scale problems
- Can optimize non linear, multiparameter model

→ Limitation
- Sensitive to learning rate
- Can get stuck in local minima
- Slow for large dataset
- Requires computing derivation

VIII) Loss functions

→ It is a mathematical function that measures how wrong the model's predictions are compared to the actual target values

→ It is needed because the model must reduce this error during training by updating weights.

→ Type

1) Mean Square loss : $L = (y - f(x))^2$
- Penalizes large errors heavily
- Not robust to outliers
- Has a single global minimum

2) Mean Absolute loss : $L = |y - f(x)|$
- More robust to outliers
- Measures distance between prediction & actual value

3) Huber Loss : $L_\delta = \begin{cases} \frac{1}{2}(y - f(x))^2 & |y - f(x)| \leq \delta \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{o.w} \end{cases}$
- More robust for outliers than MSE & smoother than MAE
- Quadratic for small errors
- Linear for large errors.

4) Binary Cross Entropy : $L = -[y \log(p) + (1-y) \log(1-p)]$
- Measures uncertainty using the idea of entropy
- Loss decreases as predicted probability becomes more accurate

5) Hinge loss : $L = \max(0, 1 - y f(x))$
- Encourages not only correct prediction, but also margin based confidence
- Penalizes even correct prediction, if they are too close to the decision boundary

6) Multi Class Cross Entropy : $L(x_i, y_i) = \sum_{j=1}^{K} y_{ij} \log(p_{ij})$

MOD 2

~~Purpose~~

Points	Machine learning (ML)	Deep learning (DL)
Feature Engineering	Features are hand crated by humans	Learns features automatically from data
Learning Structure	Mostly shallow models	Deep neutral networks with many hidden layers
Data Requirement	Works well with small to medium dataset	Needs large datasets to perform well
End to End Learning	Feature Extraction + model building	Provides end to end learning from raw input to output
Interpretability	Easier to interpret	Acts as a black box, harder to interpret

I) Neural Network

→ It is a computational model inspired by the structure and functioning of the human brain

→ It consists of interconnected units called neurons, arranged in layers that process information

→ It use nonlinear mapping to compute complex decision boundaries.

→ Advantage

Ability to Learn complex nonlinear patterns
Automatic Feature learning
High accuracy in many AI tasks
Adaptability & Flexibility
Robust to Noisy Data

→ Disadvantage

- Requires large amounts of data
- High computational cost
- Difficult to interpret
- Risk of overfitting
- Complex to train

→ Elements

a) Neurons

- It is the basic computation unit
- Each neuron receive inputs, multiplies them with weights, adds a bias, and applies an activation function.

$$Z = Wx + b$$
$$a = \sigma(z)$$

b) Layers

1) Input Layer: Receives raw data, does not perform computation & only passes information forward
2) Hidden Layer: Perform most of the computation, apply weights, bias & activation functions.
3) Output Layer: Produces the final Prediction, often uses softmax for multi class classification or linear activation for regression.

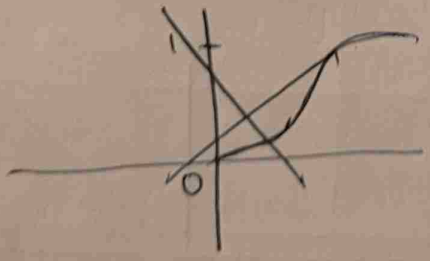
c) Weights: Represents the strength of connection between neurons.

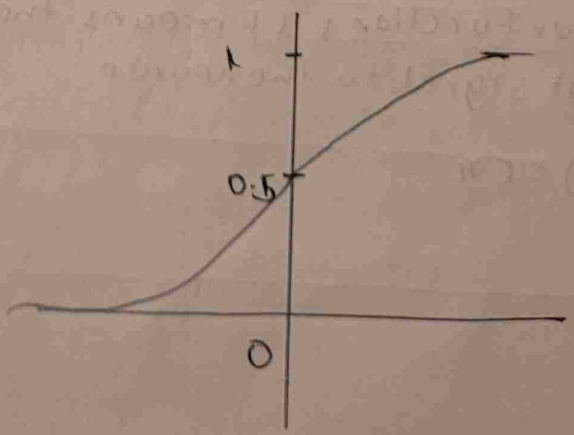
d) Biases: It shifts the activation function & allows the network to learn even when all inputs are zero.

e) Activation functions

1) Sigmoid: takes a real valued number & puts it into the range between 0 & 1

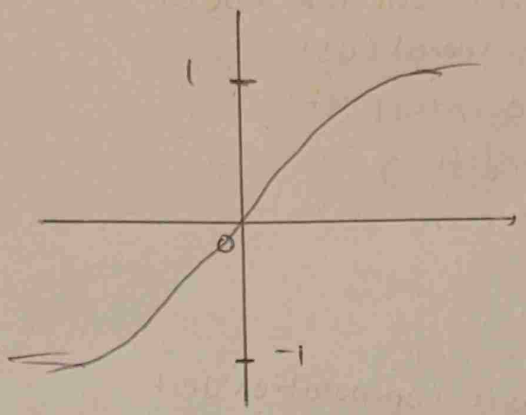
$$f(x) = \frac{1}{1+e^{-x}}$$





2) Tanh : takes a real-valued number & puts it into range between -1 & 1

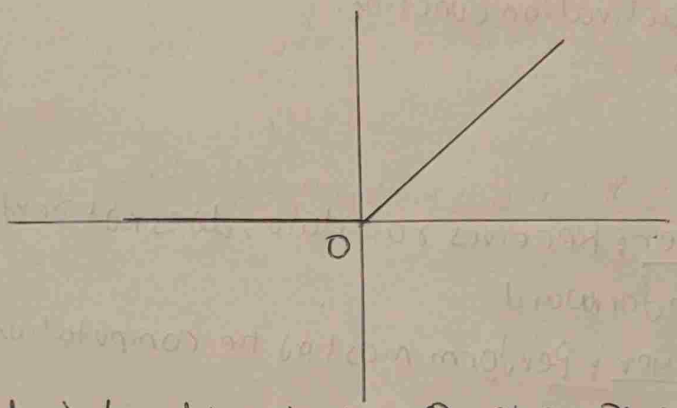
$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



3) ReLU : takes a real valued number & ~~puts~~ thresholds it at zero

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

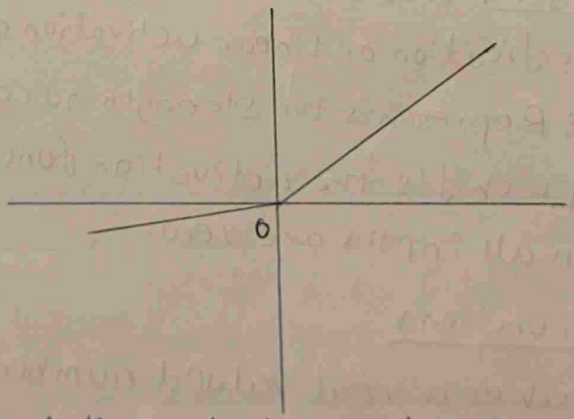
$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$



4) Leaky ReLU : Instead of function being 0 when $x<0$, it has a small negative slope

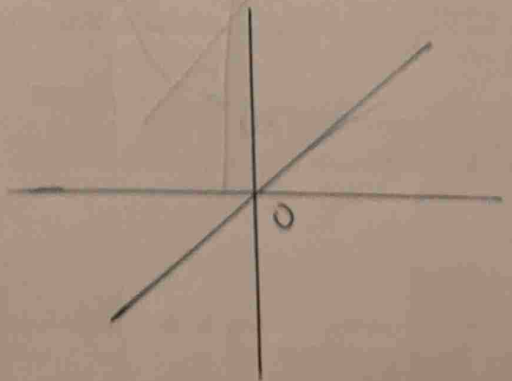
$$f(x) = \begin{cases} x & x \geq 0 \\ ax & x < 0 \end{cases}$$

$$f(x) = \max(0.1 \times x, x)$$



5) Linear Function : It means that the output signal is proportional to the input signal to the neuron

$$f(x) = cx$$



→ Training

1) Parameters of NN
- A neural network contains multiple layers, and each layer has matrixes ($W_1, W_2 \dots W_4$) & bias vectors
- All these together form the parameter set : $\theta = \{W_1, b_1, W_2, b_2 \dots$

2) Data Preprocessing

3) Forward Pass

4) Loss Function

5) Back propagation

6) Optimization using Gradient Descent

II) Learning Rate
→ It is one of the most important hyper parameters in training a nn.
→ It controls how big a step the optimizer takes in the direction opposite to the gradient to minimize to loss

→ Effects
- If learnigng rate is too small, then training becomes very ~~small~~ slow
- If learning rate is too large, then it might overshoot the next step.

→ LR scheduling
- To improve training, LR is often changed during training
- Common

1) Annealing LR decay
2) Exponential or cosine decay
3) Reduce on Plateau
4) Warm-up

III) Regularization
→ It is a technique used in training nn to prevent overfitting
→ It works by adding constraints or penalities to the model so that it does not memorize the training data.

→ Methods

1) L2 Weight Decay
- Adds penalty term $\lambda \|w\|^2$ to the loss function : $L_{reg}() = L() + \sum_k w_k^2$
- Penalizes large weights, forcing the network to keep weights small
- During gradient descent, weights are pulled towards zero.

2) L1 weight Decay

- Adds penalty based on absolute values of weights → $L_{reg}() = L() + \sum_{K} |k|$
- Encourages sparsity

3) Dropout

- Randomly 'drops' neurons & their connections during training with probability p
- Prevents co-dependency between neurons.

4) Early stopping

- Training stops when validation loss stops improving for several epochs
- Prevents the model from overfitting after a certain point.

II) Hyperparameter Learning

→ It is the process of selecting the best values for the hyperparameters of a ML or DL model

→ Method

- Grid search
- Random search
- Bayesian optimization
- Manual tuning

III)

Shallow network	Deep network
Have only one or very few hidden layer	Have many hidden layers
Learn simple or low level features Limited representational power	Learn complex hierarchical features
Easier to train & requires less computation	Harder to train requires more computation, memory & GPUs
Work well for simple tasks & smaller dataset	Performs better for complex tasks
Higher risk of underfitting	Lower risk of underfitting but higher risk of overfitting

Topics	Gradient Descent	Stochastic GD	Mini batch GD	Momentum	Nesterov Accelerated Momentum	AdaGrad	RMSProp	AdaDelta	Adam (Adaptive Moment Estimation)
Definition	Basic optimizer that update weights using full dataset Gradient	Updates weights using one sample at a time	Uses a batch of samples for each update	Adds a velocity term to smooth updates & accelerate training	Looks ahead by computing gradient at future approximated position	Adaptive learning rate method that scales LR per parameter	Fixes AdaGrad diminishing LR using exponential moving average	RMSProp variant that removes need for LR α	Most pop. optimizer combining Momentum + RMSProp
Formula	$W_{new} = W - \alpha \nabla$	$W_{new} = W - \alpha \nabla J(W)$	$W_{new} = W - \alpha \nabla J_{mini}(W)$	$V_t = \beta V_{t-1} + \alpha \nabla J(W)$ $W_t = W_{t-1} - V_t$	$V_t = \beta V_{t-1} + \alpha + \nabla J(W - \beta V_{t-1})$ $W_t = W_{t-1} - V_t$	$G_t = \Sigma(\nabla J(W))^2$ $W_{t+1} = W_t - \frac{\alpha \nabla J(W)}{\sqrt{G_t + \varepsilon}}$	$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g^2_t$ $W_{t+1} = W_t - \frac{\alpha g_t}{\sqrt{E[g^2]_t + \varepsilon}}$	same $\Delta W_t = \sqrt{\quad} \, g_t$ $\Delta W_t = (\sqrt{\quad}) \, g_t$	See notes
How it works	Computes using entire training set then update parameters	Same as GD but randomly Chose a training	Done over a mini batch	Accumulates past gradients	Peeks into the future	Accumulates squared gradients, parameters with small gradients get bigger update	Uses moving average of squared gradient	Uses ratio of RMS of updates & gradients	Tracks mean & variance of gradients to adapt LR
Advantage	Stable, accurate, good for convex functions	Very fast, escapes local minima	Faster training supports parallelization	Faster Convergence, avoids shallow minima	More accurate than Momentum, faster convergence	Great for sparse data, no manual LR tuning	Works well for RNNs, solves AdaGrad's decay problem	No LR needed Fixes AdaGrad completely	Fastest Convergence, works well on most DL
Disadvantage	Very slow for large data set	Noisy Update unstable	Need to choose Proper batch size	Requires tuning β	Slightly more computation due to look ahead gradient	LR shrinks too much → Learning stops.	Need to tune γ	More complex, can converge slower.	Slightly more memory use, may generalize worse than SGD.

MOD 3

I) CNN

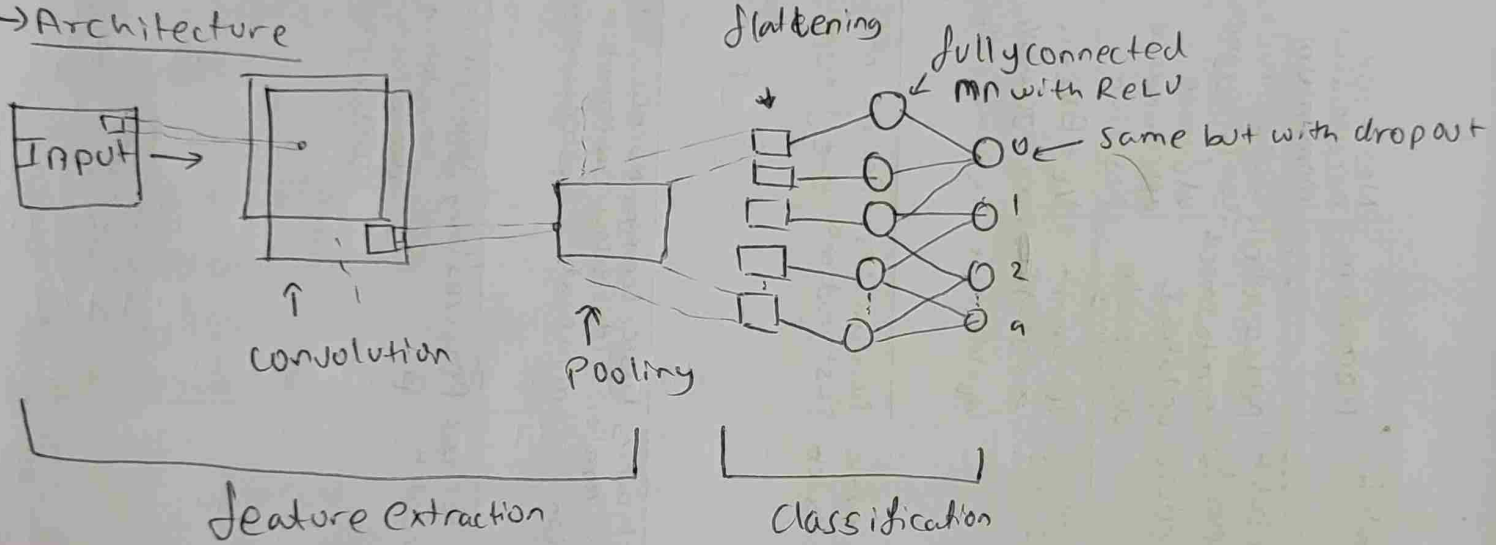
→ It is a specialized type of feed forward neural network designed to process data with grid-like topology, mainly images.

→ It takes advantage of 2D structure of image & avoid full connectivity by using local receptive fields, drastically reducing the no of parameters compared to MLPs

→ Working

- It uses learnable filters (kernel) that slide across the image & perform convolution operations to detect local features
- Each filter extracts a specific feature such as vertical edges, horizontal edges, corners, etc.
- These filter produces feature maps, which represent the presence of features in different image region.

→ Architecture



1) Convolution Layer

- Applies multiple small filters
- perform dot product between filter & local image regions
- Produces feature maps
- Detects structural info instead of raw pixel

Formula : $O = 1 + \frac{(W - F + 2P)}{S}$

2) ReLU Layer

Activation function : $ReLU(x) = \max(0, x)$

Removes negative values & keeps computation efficient

3) Pooling Layer

- Performs downsampling to reduce spatial dimensions
- Types: Max & ~~Avgpro~~ Average Pooling
- Reduces computation & controls overfitting

4) Fully Connected Layer

- Flattens output from pooling layer
- Acts like a standard neural network
- Often ends with Softmax for classification.

→ Type

Points	LeNet-5 (1998)	AlexNet (2012)	VGG-16/19 (2014)	ResNet (2015)
Intro	Fist successfull CNN designed for handwritten digit recognition. A simple, shallow architecture	Landmark CNN that won ImageNet. Triggered the Deep Learning revolution.	A very deep & uniform architecture based purely on 3×3 conv. Known for simplicity + depth	Break through model Solving vanishing gradient Problem using residual connections.
Input size	32×32 grayscale image	224×224 RGB image	→ — →	→ —
Architecture Overview	(6×5×5) C1 → AvgPool → (16×5×5) C3 ↓ FC ← C5 / FC ← AvgPool ↓ Output ~~Depth: 7~~	(11×11, stride 4) C1 → MaxPool → (5×5) C2 ↓ MaxPool ← (3×3) C3, 4, 5 ← MaxPool ↓ FC6, FC7 → Softmax	5 Conv Block. Each block = (2 - 3 Conv Layer of 3×3) + ReLU + MaxPool → Flatten ↓ FC3 Softmax ← FC2 ← FC1	Initial 7×7 Conv + MaxPool → Multiple Residual Block (Conv Layer + identity skip) → Global Avg Pooling ↓ FC softmax
Depth	7 trainable layers → shallow	8 trainable layers	16 or 19 layers → deep	50, 101, 152+ layers → Ultra deep
Activation Function	Tanh / sigmoid	ReLU (~~major innovation~~)	ReLU	ReLU + Batch normalization
Special Features	Simple structure, low computation, early CNN milestone	ReLU to avoid vanishing gradients, dropout for regularization, trained using GPU's	Uniform design with repeated 3×3 filters, strong feature extractor used in many tasks	Residual blocks solve degradation problem, enables deeper network to train effectively
Strength	Efficient, easy to train, good for small datasets	High accuracy, faster training, can learn large features	Very deep, consistent, powerful features, good for transfer learning	Extremely deep, Stable training, Superior performance
Weakness	Too simple for complex images	Very heavy computation (GPU needed)	Very large model size, slow inference & training	Complex architecture, difficult to implement manually

→ Challenges

1) Vanishing & exploding Gradient Problem

- In a very deep CNNs, during back propagation the gradients can become extremely small or extremely large
- Vanishing → makes learning slow
- Exploding → makes it unstable & overflow

2) Degradation Problem in Deep network

- The deeper the network the performance initially increases & then becomes worse
- This happens not because of overfitting, but because deeper network become harder to optimize.

3) Too many training Parameters

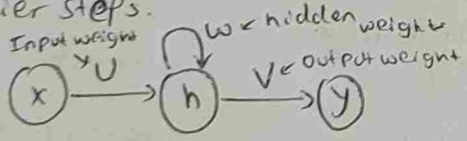
- CNNs contain million of parameters due to many conv filters, deep layers & FC Layer
- Which leads to heavy computational requirements, slow training time, Need for powerful GPU's & large memory usage

4) Long training time

II) RNN

→ It is designed for sequential data.

→ It uses the previous hidden state memory, which carries information from earlier steps.



→ Forward Propagation

- The hidden state h at time stamp t can be computed by $h_t = \tanh(Ux_t + Wh_{t-1})$
- The output at time step t can be computed by $\hat{y}_t = \text{Softmax}(Vh_t)$
- Thus, in forward propagation to predict output, RNN uses the current input & the previous hidden state

→ Backpropagation

- After the initial prediction we calc the loss $L_t = -\overset{\text{actual}}{y_t} \log(\overset{\text{predicted}}{\hat{y}_t})$
- Total loss over the entire sequence $L = \sum_{j=0}^{T-1} L_j$

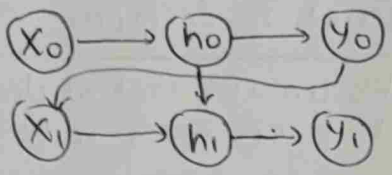
- Goal is to minimize loss by updating weights

$$V = V - \alpha \frac{\partial L}{\partial V}, \quad W = W - \alpha \frac{\partial L}{\partial W}, \quad U = U - \alpha \frac{\partial L}{\partial U}$$

→ Types

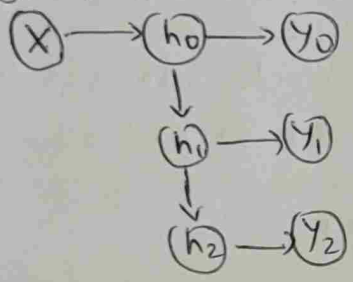
1) One to one

- A single input is mapped to single output, and output from the time step t is fed as an input to the next time step. Ex stock market prediction



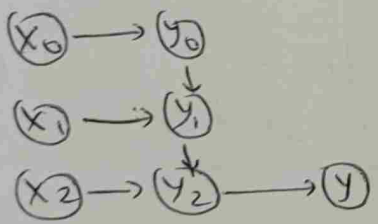
2) One to many

- A single input is mapped to multiple hidden states & multiple output values
  Ex Image caption generator



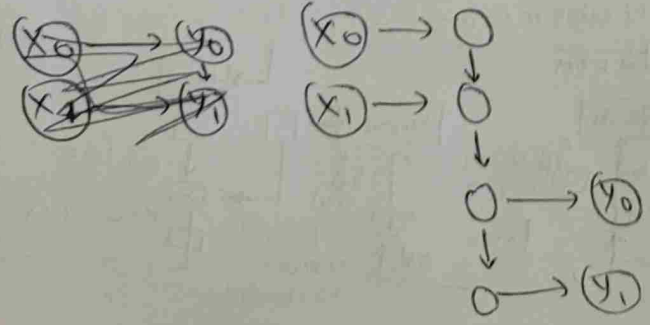
3) Many to one

- It takes sequence of input & maps it to a single output value. Ex sentiment classification



4) Many to many

- We map a sequence of input of arbitrary length to sequence of output of arbitrary length. Ex language translation.



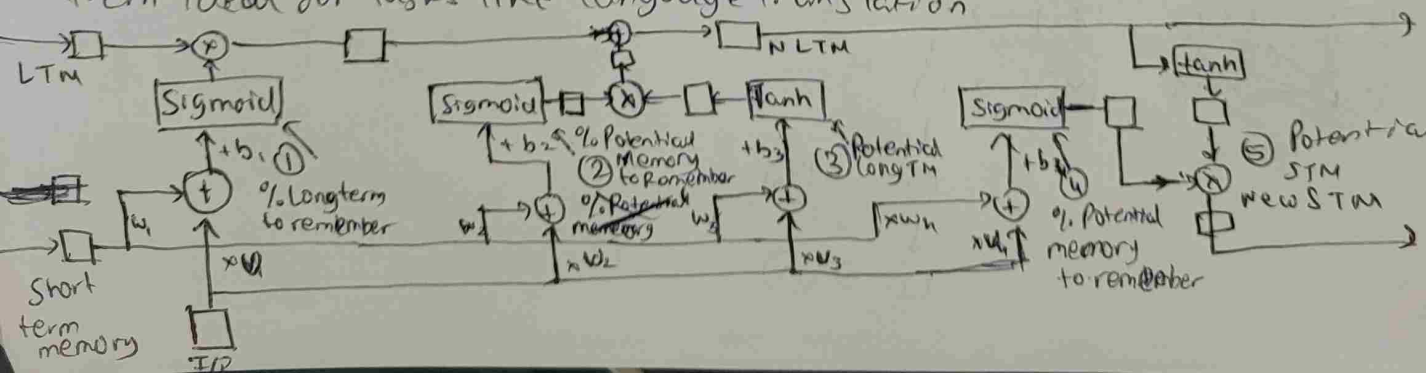
Improvements

Point	Feed Forward NN	Recurrent NN
Data flow direction	Information flows only in one direction. No loops	Contains a loop / feed back connection in the hidden layer
Memory	No memory of previous inputs. Each input is processed independently	Has memory using the previous hidden state, which stores contextual information from past inputs.
Suitability	Works on non sequential data like image classification, simple tabular data	Works on sequential data such as text, time series, speech, music, etc.
Input dependency	Output depends only on the current input	Output depends on current input + previous hidden state
Context Understanding	Cannot understand context	Understands context across word / time steps.
Architature	Simple layered structure Input → hidden → Output	Unrolled
Handling Long Sequences	Cannot handle sequential dependencies	Designed to handle short term dependencies.
Limitation	Cannot process sequences Lacks memory	Suffers from vanishing / exploding gradients

III) LSTM

→ It is an enchanced version of the RNN.

→ It can capture long term dependencies in sequencial data making them ideal for tasks like language translation



→ Gates

1) Forget Gate

- Decides what info to remove from previous memory

$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$

If $f_t \approx 0$ delete else if $f_t \approx 1$ keep

2) Input Gate

- Decides what new info. to store in memory

$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$

3) Candidate state

- Potential new info to add into memory

$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$

→ Update Cell State

- $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$
- Forget gate removes unwanted memory
- Input + candidate add new memory

→ Output Gate & Hidden State

1) Output Gate : $O_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$

2) Hidden State : $h_t = O_t \cdot \tanh(C_t)$

→ Backpropagation

- It is similar to RNN but with more parameters.
- Loss $\Rightarrow L_t = -\sum y_t \log(\hat{y}_t)$, total $\Rightarrow L = \sum_{t=1}^{T} L_t$
- Compute Gradient for:

1) Forget Gate weights
2) Input Gate weights
3) Output Gate weights
4) Candidate state weights
5) Hidden to output

- Weight update : $W = W - \eta \frac{\partial L}{\partial W}$

- Why does it work better in LSTM

- Cell state allows constant error flow
- Gates prevent vanishing gradients
- LTM is preserved

MODU 4

I) NLP

→ It is a branch of AI that enables computers to understand, interpret & generate human language

→ Deals with text & speech in natural form

→ Combines Linguistic + ML + DL

→ Applications

- Machine Translation
- Speech to text
- Text classification
- Sentiment Analysis
- Chat bot

→ Approaches

1) Rule based

- Relies on predefined linguistic rules & patterns to process text
- Linguistic experts & programmers manually create rules that encode knowledge about language
- These rules are used to perform task such as tokenization.
- Good for small, controlled domains. But difficult to scale

2) Statistical

- Learns patterns from labeled datasets
- These use probabilistic techniques to make predictions.
- Useful for text classification
- Requires large corpus to learn accurate patterns. But less manual than rule based

3) Neural Network

- Uses RNN, LSTM, etc
- Learns deep semantic & contextual meaning
- Handles long term dependencies effectively
- Achieves state of the art performance

→ Components

1) NLU

- Focuses on enabling the machine to understand human language
- Key functions
  - Text interpretation
  - Syntax Analysis
  - Semantic Analysis
  - Entity Recognition
  - Context understanding

2) NLG

- Focuses on enabling the machine to produce human like language
- Key functions
  - Content planning
  - Sentence Planning
  - Lexical choice
  - Surface realization
  - Response Generation

II) Phases of NLP

1) Lexical/Morphological Analysis

- This phase scans the source code as a stream of characters & converts it into meaningful lexemes
- It divides the whole text into para, sentences & words.
- It also assigns the possible POS to the word.
- Relation Type
  - Inflectional
  - Derivational

2) Syntactic Analysis

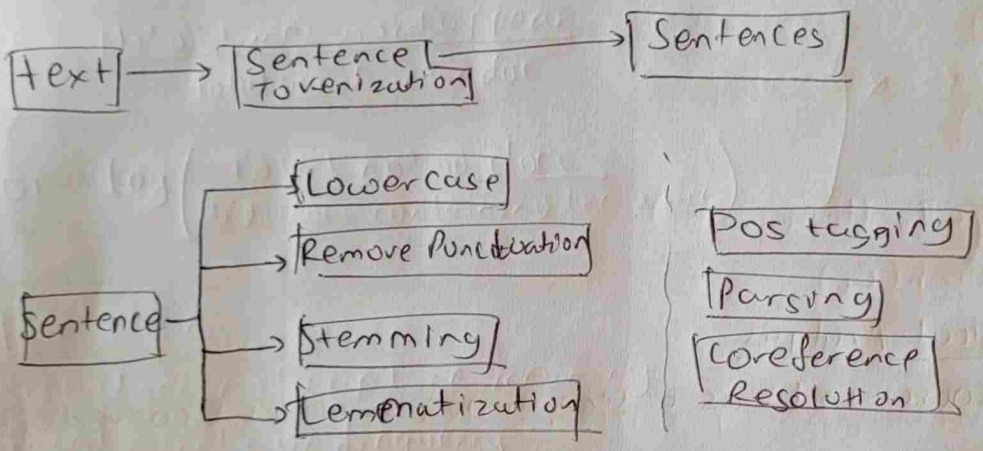
- Syntax is a set of rules to construct grammatically correct sentences out of words & phrases in a language
- It ensures that a given piece of text is correct structure
- It tries to parse the sentence to check correct grammer at the sentence level
- ~~This is do~~

3) Semantic Analysis
- It is concerned with the meaning representation
- It focuses on the literal meaning

4) Discourse Integration
- It deals with the effect of a previous sentence on the sentence in consideration
- Resolves references like, he, she, it.

5) Pragmatic Analysis
- Understands intended meaning beyond literal words
- Uses world knowledge & context.

III) NLP Preprocessing



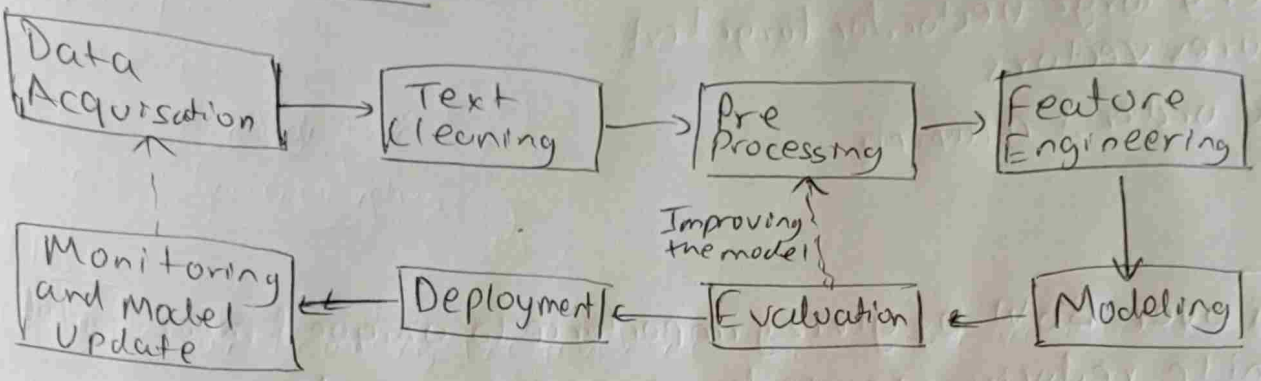
1) Tokenization
→ Splitting text into words, sentence or sub words
2) Stopword Removal
3) Stemming
→ Reducing the words to root form using Crude rule
4) Lemmatization
→ Convert words to dictionary base form
5) Part of Speech tag
→ Identifies noun, verb, etc
6) Lowercasing
7) Normalization
→ Convert to standard format

8) Removing of Punctuation & Special characters

9) Regular Expressions

10) Sentence Segmentation

Generic Pipline



IV) Text Representation

→ It refers to converting text into numerical form so that ML or DL model can process it

→ Since machines cannot understand raw text, we use vectorization techniques.

→ Techniques

1) One-Hot encoding

- It represents each unique word in the vocabulary as a binary vector with '1' at the index of the word & 0 at all other position
- Key points
  - Simple & easy to implement
  - All vectors have equal length
  - No information about word meaning
  - Vectors become high dimensional & sparse
  - Cannot capture similarity between words

2) Bag of words

- It converts text into fixed length vector by counting how many times each word appears in the document. It ignores grammar, order & context
- Steps
  - Create vocab of unique words
  - Count frequency of each word in every document
  - Represent each doc. as vector of word counts

- Advantage
  - Simple & effective
  - Works well for small dataset
  - Good baseline model
- Limitation
  - Produces very large vector for large text
  - Creates spares vectors
  - Loses word order & meaning

3) TF-IDF

- Improves over BoW by giving importance to unique & meaningful words, while reducing weight for very common words.
- Formulas

$$TF = \frac{\text{No° of times word appear}}{\text{Total words in the doc}}$$

$$IDF = \text{Log}\left(\frac{\text{Total no° of docs}}{\text{No° of docs containing the word}}\right)$$

$\Big\}$ X $\Rightarrow$ TFIDF

- Limitation
  - Still does not capture meaning
  - Based purely on frequency
  - Produces sparse vectors

What is Word Embedding in NLP?

Word Embedding is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space.

- Method of extracting features out of text so that we can input those features into a machine learning model to work with text data.
- It allows words with similar meanings to have a similar representation. Thus, Similarity can be assessed based on Similar vector representations.
- High Computation Cost: Large input vectors will mean a huge number of weights. Embeddings help to reduce dimensionality.
- Preserve syntactical and semantic information.
- Some methods based on Word Frequency are Bag of Words (BOW), Count Vectorizer and TF-IDF.

Need for Word Embedding?

- To reduce dimensionality.
- To use a word to predict the words around it.
- Helps in enhancing model interpretability due to numerical representation.
- Inter-word semantics and similarity can be captured.

⭐ TYPES OF WORD EMBEDDINGS (From PPT – Full 10-Mark Notes)

Your PPT covers two major categories:

1. Frequency-Based Embeddings

2. Prediction-Based Embeddings

Below I explain BOTH completely so you can write 5 or 10 marks.

⭐ 1. FREQUENCY-BASED EMBEDDINGS (10 Marks Answer)

(Based directly on PPT slides: VSM, TF-IDF, Co-occurrence Matrix)

word embedding.pptx

Frequency-based embeddings convert text into numerical values using **word frequency**, **co-occurrence**, or **statistical counts**. These methods create sparse, high-dimensional vectors that do not capture deeper meaning but are simple and effective.

⭐ A. Vector Space Model (VSM)

- Represents each document or word as a **vector of term frequencies**.
- Uses techniques like **Bag-of-Words** and **TF-IDF**.
- Produces **very high-dimensional sparse vectors**.
- Does NOT capture word order or meaning.
- Distance measures (cosine similarity) used for comparison.

⭐ B. TF-IDF Vectorization

- **Term Frequency (TF)** captures how often a word appears.
- **Inverse Document Frequency (IDF)** down-weights common words.
- TF × IDF gives weight to **important words**.
- Still sparse, still no context/semantic meaning.
- Good for search engines + keyword extraction.

⭐ C. Co-occurrence Matrix

- Counts how often words appear **together in a fixed context window**.
- Example from PPT:
  "India won the match. I like the match."
- Represents semantic relationships using co-occurrence frequencies.
- Very interpretable but **huge matrix size** (|V| × |V|).
- High sparsity → wastes memory.

⭐ Advantages (from PPT)

- Simple
- Easy to compute
- Works decently for small datasets

⭐ Limitations (from PPT)

- Sparse, high-dimensional vectors
- Cannot capture deep semantic meaning
- Cannot handle polysemy ("bank" money vs river)
- Large vocabulary → huge memory

⭐ 2. PREDICTION-BASED EMBEDDINGS (10 Marks Answer)

(Based directly on PPT slides: Word2Vec, CBOW, Skip-Gram, GloVe, FastText)

word embedding.pptx

Prediction-based embeddings use neural networks to **predict surrounding words** and learn **dense, low-dimensional semantic vectors.**

⭐ A. Word2Vec (Mikolov, 2013)

Word2Vec uses a shallow 2-layer neural network to learn embeddings.
Two models:

⭐ i. CBOW (Continuous Bag of Words)

- Predicts **center word** using **context words**.
- Ignores order (bag-of-words style).
- Smooths noise and works well for large datasets.
- Architecture in PPT:
  - One-hot → Embedding → Hidden → Softmax
- Loss: Cross-entropy.
- Efficient for frequent words.

⭐ ii. Skip-Gram

- Opposite of CBOW.
- Predicts **context words** from **center word.**
- Works well for **rare words**.
- Uses:
  - Hierarchical softmax
  - Negative sampling

⭐ Advantages of Word2Vec

- Captures **semantic** and **syntactic** relationships
- Supports vector arithmetic:
  "king - man + woman ≈ queen"
- Dense, low-dimensional vectors
- Fast training

⭐ B. GloVe (Global Vectors – Stanford)

- Combines **global co-occurrence + local context windows**.
- Learns word vectors by minimizing weighted least squares.
- Uses ratio of co-occurrence probabilities.
- Captures global statistics better than Word2Vec.
- Performs well on analogy tasks.

⭐ C. FastText (Facebook AI)

- Represents a word as a **bag of character n-grams.**
- Example: "where" → <wh>, <whe>, <her>, <ere>, <re>
- Helps capture **morphology** (prefix/suffix).
- Produces embeddings for **OOV (out-of-vocabulary) words.**
- Better for languages with rich word formation.

⭐ D. Limitations of Prediction-based Embeddings (from PPT)

- Learn only **one embedding per word**
- Cannot capture **contextual meaning** (bank = money vs river)
- Modern models like **ELMo, BERT, GPT** fix this by creating contextual embeddings

---

⭐ 1. What is Text Generation? (Definition)

- Text generation is the process of using **AI / NLP models** to produce new, human-like written content.
- Models are trained on **large text datasets** to learn patterns, sentence structure, grammar, and context.
- Once trained, they can generate **coherent and meaningful text**, similar to how predictive text works but far more advanced.

⭐ 2. How Text Generation Works

1. **Training on Large Text Corpora**
   - Model reads millions of sentences and learns:
     ✔ grammar
     ✔ sentence flow
     ✔ word relationships
     ✔ style
2. **Next-Word Prediction**
   - For each input token, the model predicts the **most likely next word**:

$$P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

3. **Sampling Techniques**
   - Greedy decoding
   - Beam search
   - Temperature sampling
4. **Generates full sentences or paragraphs**
   - Maintains **coherence**, **fluency**, and **contextual relevance**.

⭐ 3. Approaches to Text Generation (from screenshot)

1. Autoregressive Models

- Predict next word using previous words.
- Examples: GPT-4, GPT-3.
- Ensures logical flow & coherence.

2. Seq2Seq Models

- Encoder–decoder architecture.
- Used for machine translation & summarization.

3. Fine-Tuned Models

- Pretrained models adapted to specific tasks:
  – Legal documents
  – Medical reports
  – Financial summaries

⭐ 4. Applications of Text Generation

(from screenshots)

- **Content Creation** → blog posts, product descriptions
- **Chatbots & Virtual Assistants** → Siri, Alexa
- **Language Translation** → Google Translate
- **Summarization** → condense articles, textbooks
- **Code Generation** → GitHub Copilot, auto-code suggestions

---

⭐ 5. Limitations of Text Generation

- **Lack of deep contextual understanding**
- **Bias** in training data
- **Plagiarism / originality concerns**
- **Ethical issues & misinformation risk**

⭐ 1. What is Seq2Seq?

- Seq2Seq = **Sequence-to-Sequence** model.
- Converts one sequence → another (e.g., English → French).
- Uses **Encoder + Decoder** neural networks (RNN/LSTM/GRU).

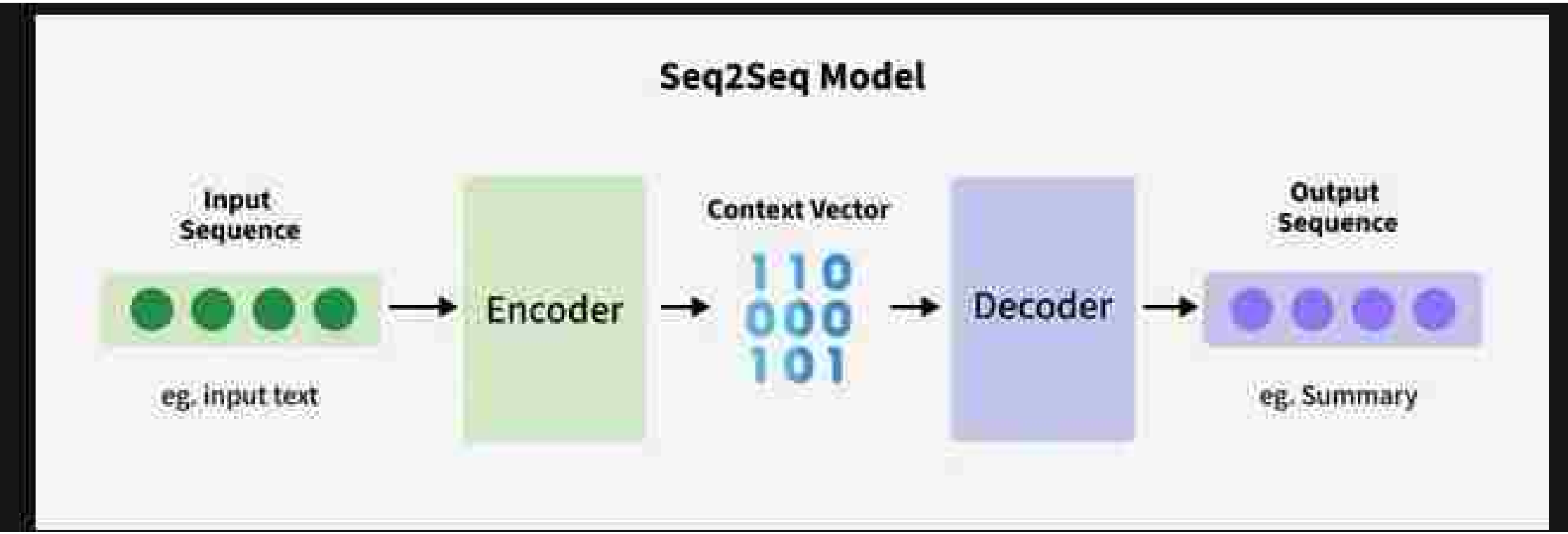
⭐ 2. Architecture Components

⭐ A. Encoder

- Reads input sequence **token by token**.
- Produces a **context vector** (fixed-length summary).
- Updates hidden state at each step.

⭐ B. Decoder

- Takes context vector from encoder.
- Generates output **one token at a time**.
- Uses previous output + context to predict next token.
- Uses softmax to choose next word.


Seq2Seq Model
Input Sequence
eg. input text
Encoder
Context Vector
1 1 0
0 0 0
1 0 1
Decoder
Output Sequence
eg. Summary

⭐ 3. How Seq2Seq Works (from screenshots)

Step 1: Encoding

- Encoder processes entire input sequence.
- Final hidden state = "summary" of input.

Step 2: Decoding

- Decoder generates target sequence.
- Example:
  Input → "I am learning"
  Output → "Je suis apprenant"

Step 3: Teacher Forcing

- During training, the true previous word is fed to the decoder instead of predicted one.
  Benefits:
  ✔ Faster training
  ✔ Less error propagation

⭐ 4. Applications of Seq2Seq

- Machine translation
- Text summarization
- Speech recognition
- Image captioning
- Time-series prediction

⭐ 5. Advantages

- Handles variable-length inputs & outputs
- Good for sequential data
- Captures context of input sequence
- Works even better with **Attention**

⭐ 6. Disadvantages

- Computationally expensive
- Hard to interpret internal decisions
- Prone to overfitting
- Struggles with rare words