UNIT 4

ADVANCED ANALYTICS ON BIG DATA

SYLLABUS:

1. Deep Learning for Big Data Analytics

- Neural Networks (NNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

2. Text Analytics and Natural Language Processing (NLP) on Big Data

- Text mining and feature extraction
- Sentiment analysis
- Topic modelling
- Language modelling and embeddings

3. Graph Analytics

- Social network analysis
- Recommendation systems
- Graph-based algorithms

4. Ensemble Learning Methods

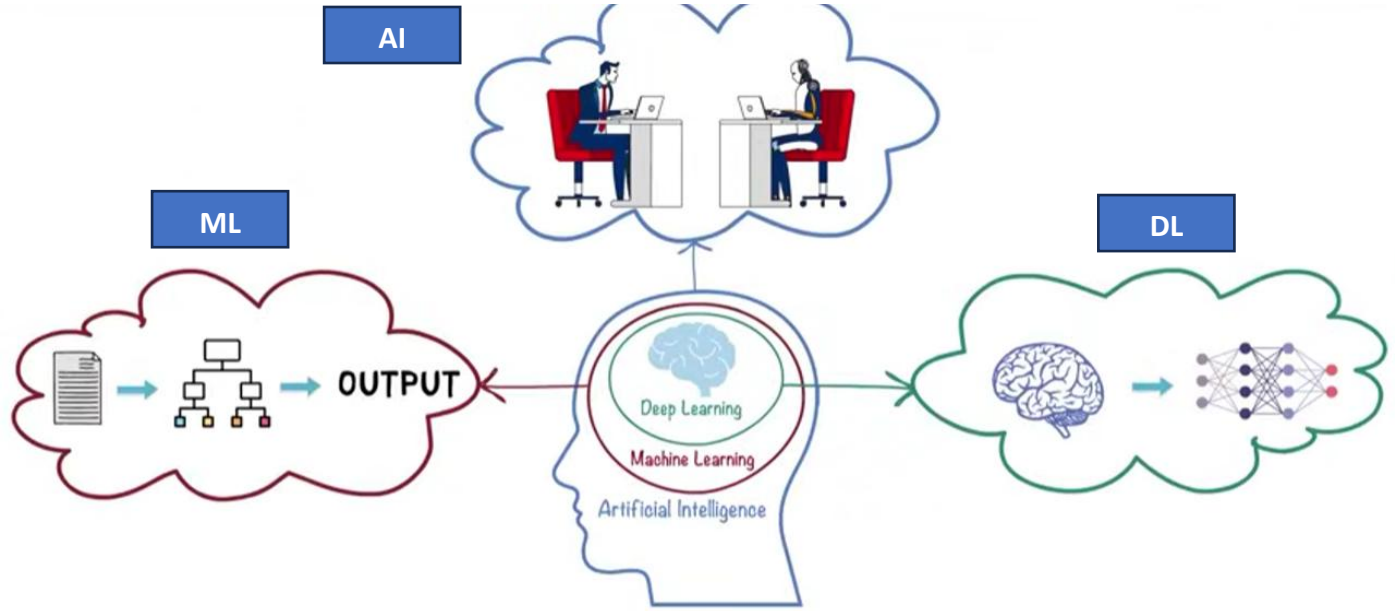
- Bagging (e.g., Random Forest)
- Boosting (e.g., AdaBoost, XGBoost)
- Stacking and voting techniques

5. Model Evaluation Techniques

- Cross-validation
- Confusion matrix, accuracy, precision, recall, F1-score
- ROC curve and AUC

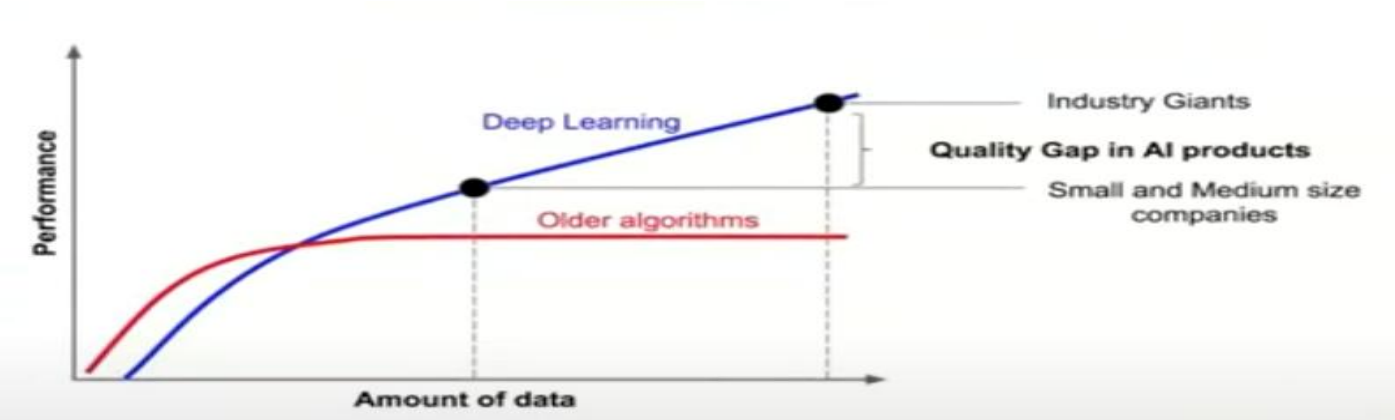
Deep Learning

Deep learning is a subset of machine learning that uses artificial neural networks with multiple layers to learn from and make decisions about large amounts of data. Inspired by the human brain, it processes information through these layers to recognize complex patterns without needing explicit human programming for every step, enabling tasks like image recognition, speech recognition, and language translation.



Why do we need Deep Learning?

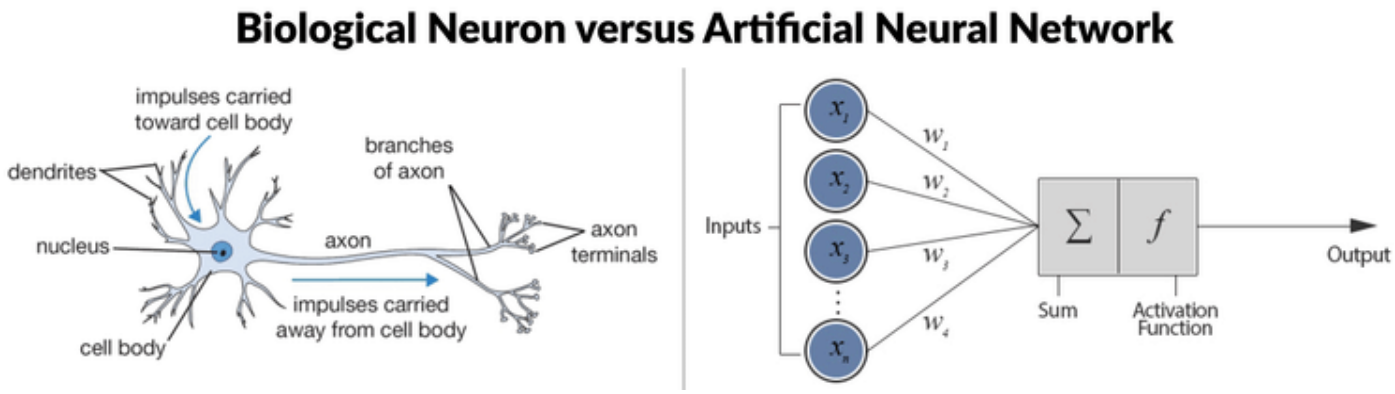
- It works well if you provide more and more data.
- It prevents overfitting, unlike ML which after a threshold value becomes stagnate.



What is Deep learning?

- Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions
- Deep learning technology drives many AI applications used in everyday products, Digital assistants, Voice-activated television remotes, Fraud detection, Automatic facial recognition.
- Deep learning models are computer files that data scientists have trained to perform tasks using an algorithm or a predefined set of steps.
- Businesses use deep learning models to analyze data and make predictions in various applications.

Artificial Neural Network



Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain.

1. Introduction

- **Definition:** An Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the human brain, designed to recognize complex patterns and relationships in data.
- It consists of interconnected processing elements called **neurons** arranged in **layers**.

Neurons:

The basic units of neural networks that process inputs to produce outputs.

Layers:

ANNs consist of an input layer, hidden layers, and an output layer. "Deep" refers to networks with many hidden layers.

Activation Functions:

Functions like ReLU (Rectified Linear Unit), sigmoid, and tanh introduce non-linearity to help the network learn complex patterns.

2. Architecture of ANN

a. Basic Structure

1. **Input Layer**
    - Receives input features from the dataset.
    - Each neuron represents one feature.
    - No computation; it only passes data to the next layer.
2. **Hidden Layers**
    - Perform computations using **weighted connections**.
    - Each neuron applies a **weighted sum** followed by a **non-linear activation function**.
    - The number of hidden layers and neurons controls model complexity.

3. **Output Layer**
    - Produces the final output (classification, regression, etc.).
    - Activation function depends on the task (e.g., softmax for classification, linear for regression).

b. Neuron Model (Perceptron)

Each neuron performs:

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

Where:

- xi: input features
- wi: weights
- b: bias term
- f: activation function (e.g., sigmoid, tanh, ReLU)
- y: output of the neuron

3. Working Principle of ANN

1. **Forward Propagation**
    - Input data passes through layers.
    - Each neuron computes weighted sums and applies activation functions.
    - Outputs move layer by layer until the final prediction.
2. **Loss Calculation**
    - The predicted output is compared with the actual label using a **loss function** (e.g., MSE, cross-entropy).
    - The loss measures the model's prediction error.
3. **Backward Propagation (Backpropagation)**
    - Error is propagated backward to update weights.
    - Gradients of the loss function w.r.t. weights are computed using **chain rule of calculus**.
    - **Gradient Descent** (or its variants) updates the weights to minimize loss.
4. **Iteration (Training)**
    - Steps 1–3 are repeated for many **epochs** until the network learns optimal weights.

4. Activation Functions

Common activation functions introduce non-linearity:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- ReLU: $f(x) = \max(0, x)$

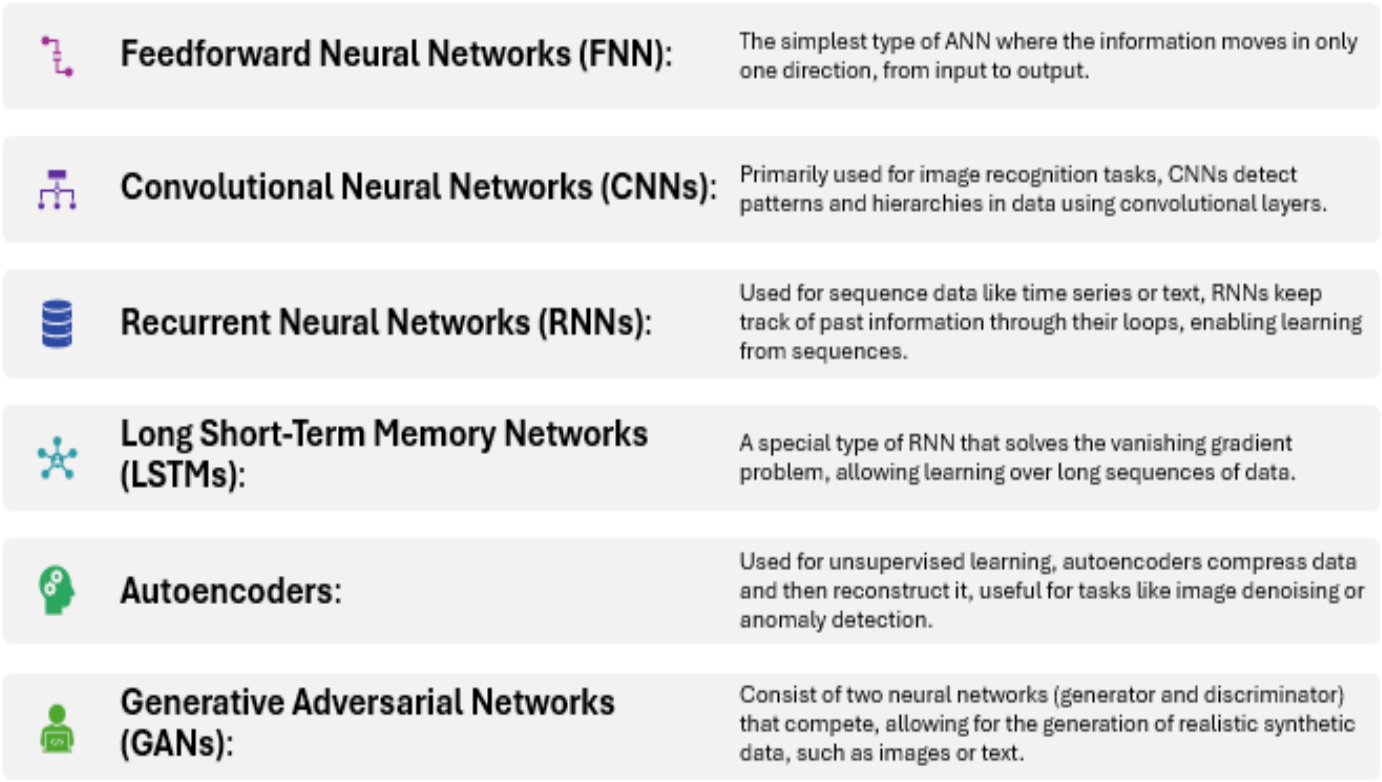
5. Key Characteristics

- Learns complex, non-linear mappings.
- Requires large data and computational power.
- Forms the foundation of **deep learning** when extended to multiple layers.

6. Applications

- Image and speech recognition
- Natural language processing
- Medical diagnosis
- Financial forecasting

Key Architectures



1. Feedforward Neural Network (FNN)

Definition:

- The simplest type of Artificial Neural Network where information moves **only in one direction** — from input → hidden layers → output.
- No feedback or looping connections.

Architecture:

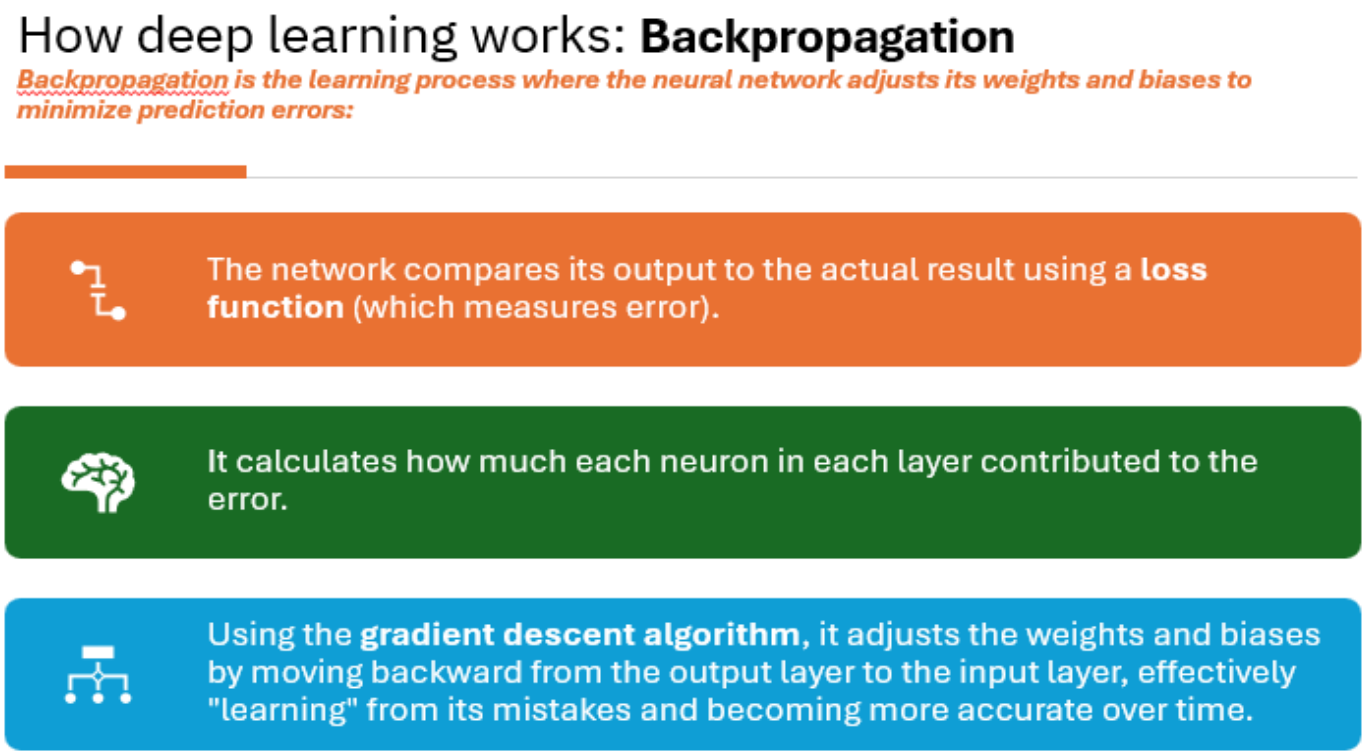
- Layers: **Input layer**, one or more **hidden layers**, and an **output layer**.
- Each neuron in one layer connects to all neurons in the next layer with associated **weights**.

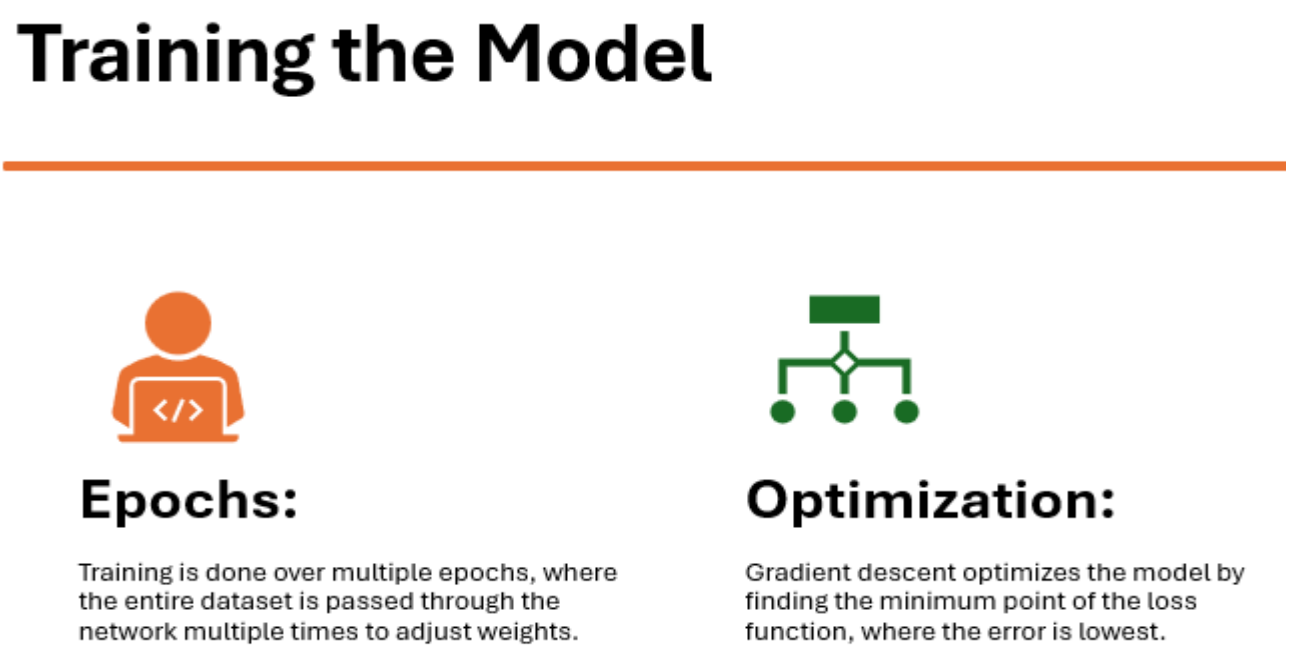
Working:

1. Input data passes forward through the layers.
2. Each neuron computes a **weighted sum** followed by an **activation function**.
3. The final output is compared with the target using a **loss function**.
4. **Backpropagation** adjusts weights to minimize loss.

Applications:

- Simple classification/regression tasks, e.g., credit scoring, spam detection.





Deep Learning Frameworks

JAX:

A relatively new framework from Google, designed for high-performance machine learning with easy-to-use automatic differentiation and GPU/TPU support.

PyTorch:

A flexible and user-friendly framework developed by Facebook.

It's popular in research due to its dynamic computational graph and straightforward debugging.

TensorFlow:

Developed by Google, TensorFlow is widely used in both academia and industry due to its scalability and production-ready features.

Improving Accuracy Over Time

The combination of forward propagation (to make predictions) and backpropagation (to adjust weights and biases) allows deep learning models to continuously improve their accuracy

Over time, as the model processes more data and fine-tunes its parameters, it becomes more adept at making precise predictions or classifications

This cyclical process of learning, predicting, and improving underpins the effectiveness of deep learning models

Computational Requirements

Deep learning models require massive computational resources

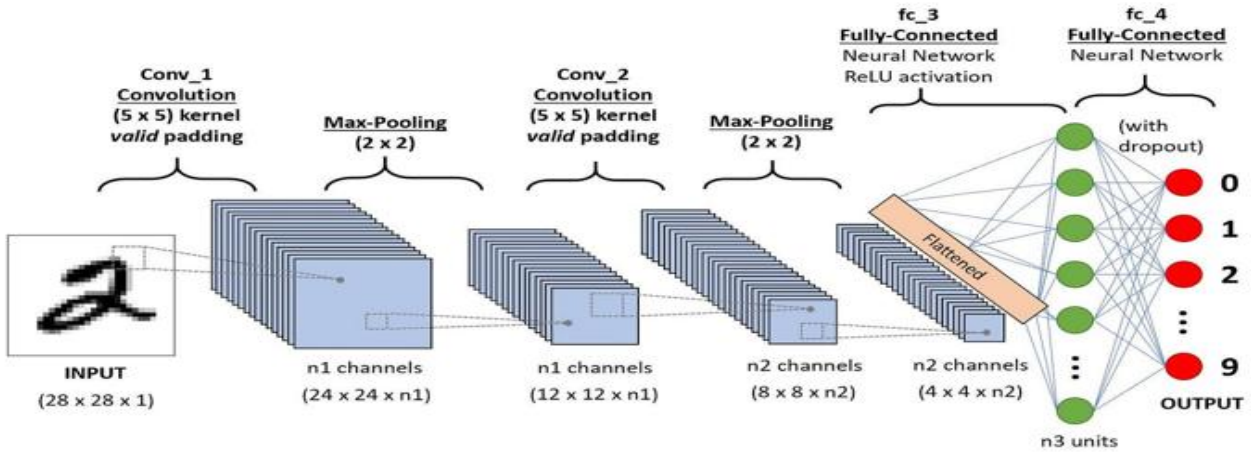
GPUs (Graphical Processing Units):

GPUs are often used to train deep learning models as they can handle parallel computations efficiently, speeding up the training process.

Cloud Computing:

Cloud platforms, like Google Cloud, AWS, and Azure, offer distributed computing power that can scale to meet the computational needs of deep learning models, especially when managing multiple GPUs on-premise becomes costly or resource-intensive.

2. Convolutional Neural Network (CNN)

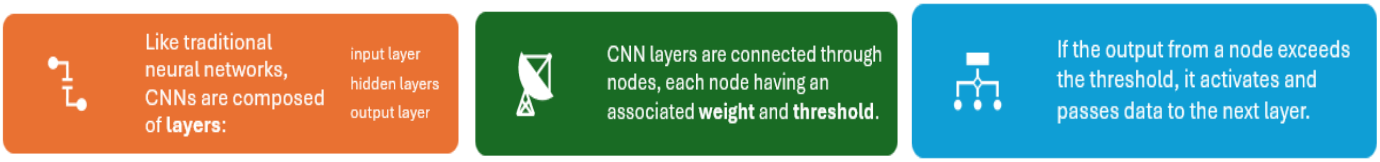


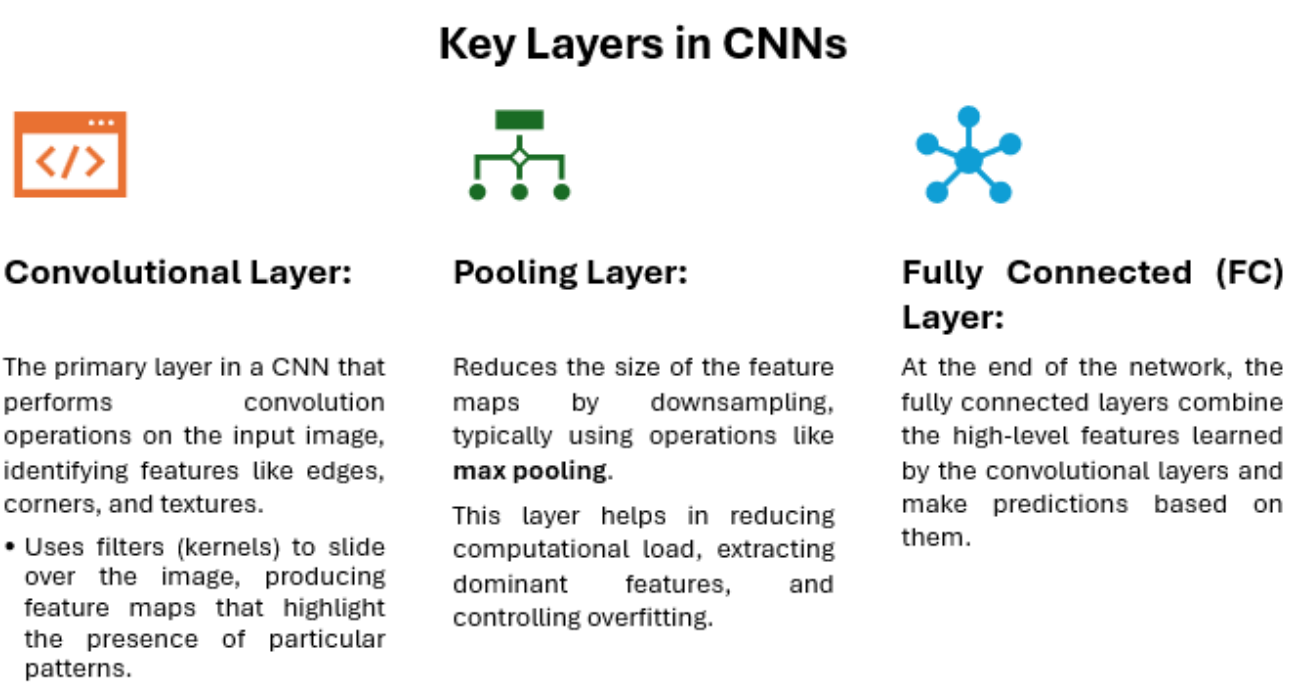
Definition:

- Specialized for **image and spatial data**.
- Uses **convolution operations** to extract spatial hierarchies (edges, textures, patterns).

Architecture:

- **Convolutional Layers:** Apply filters/kernels to extract features.
- **Pooling Layers:** Downsample feature maps (e.g., max pooling).
- **Fully Connected Layers:** Combine extracted features for classification.





Pooling layers

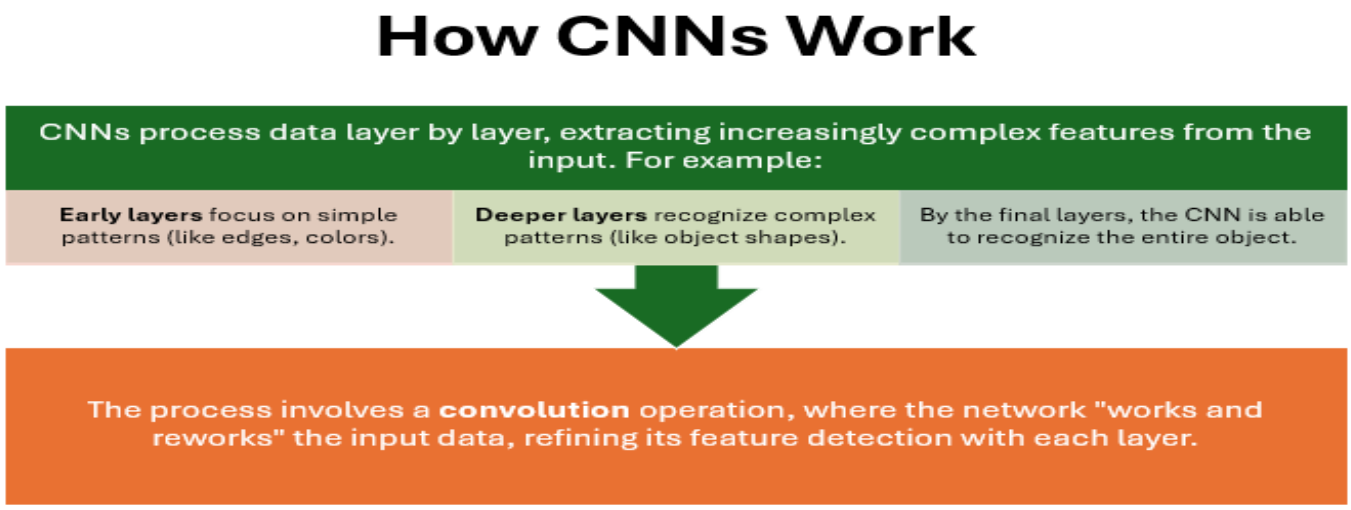
reduce complexity by down sampling the feature maps, which may lead to a **loss of information**.

CNN offset

However, this loss is often offset by CNN's ability to reduce model complexity, improve efficiency, and minimize overfitting.

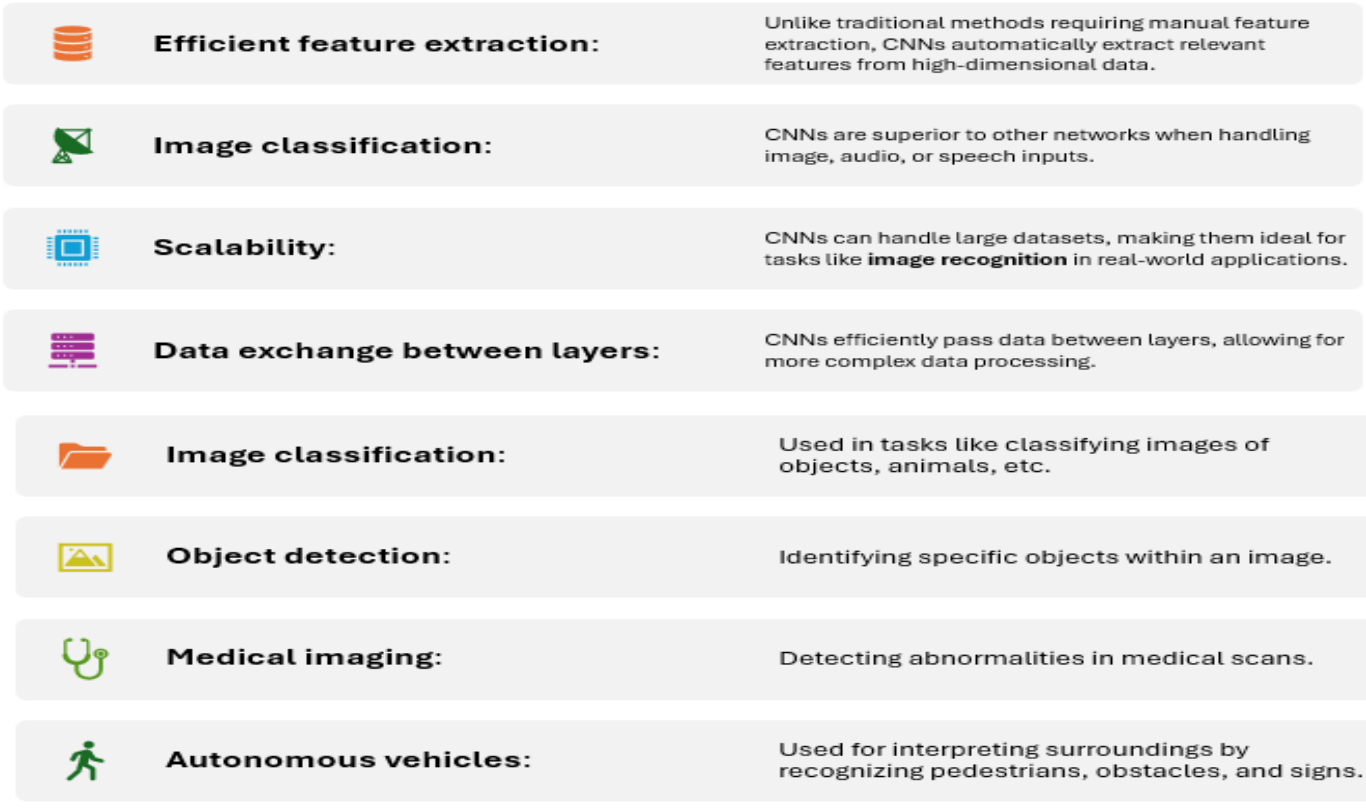
Working:

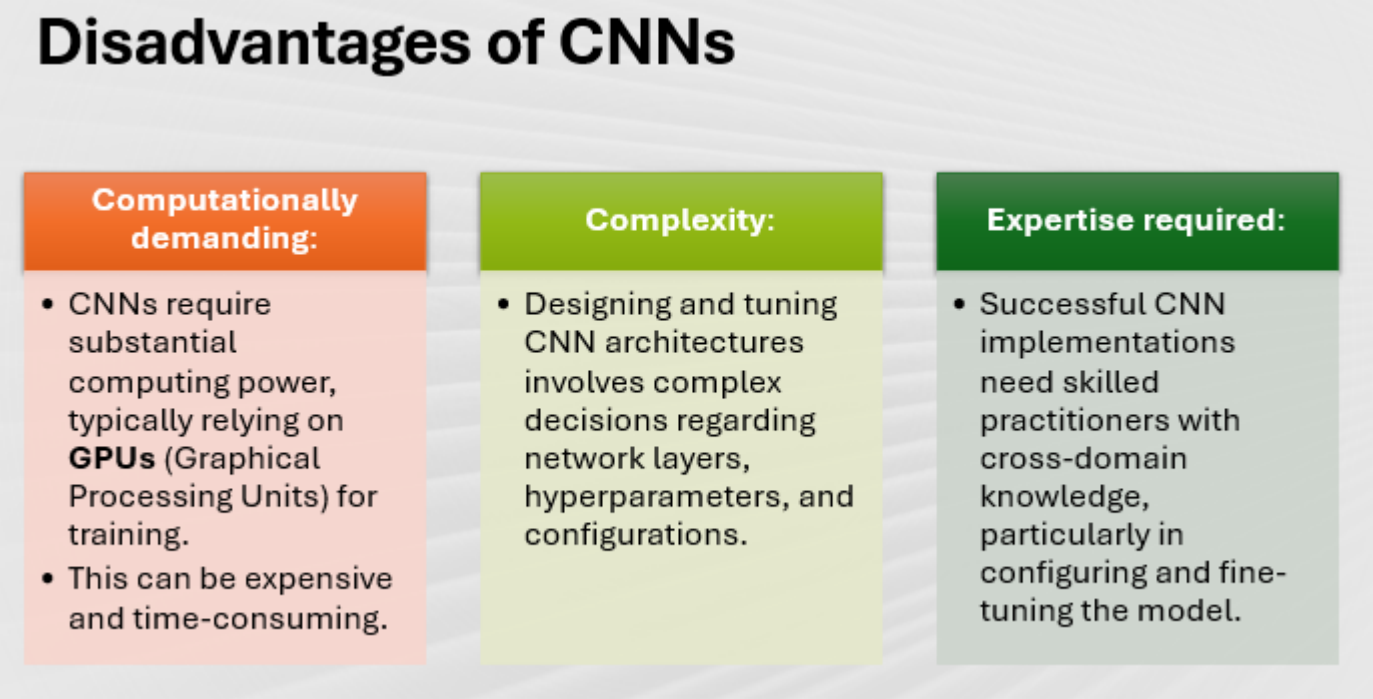
1. Convolution detects local patterns in the image.
2. Pooling reduces dimensions, making the network efficient.
3. The final layer outputs class probabilities or feature embeddings.



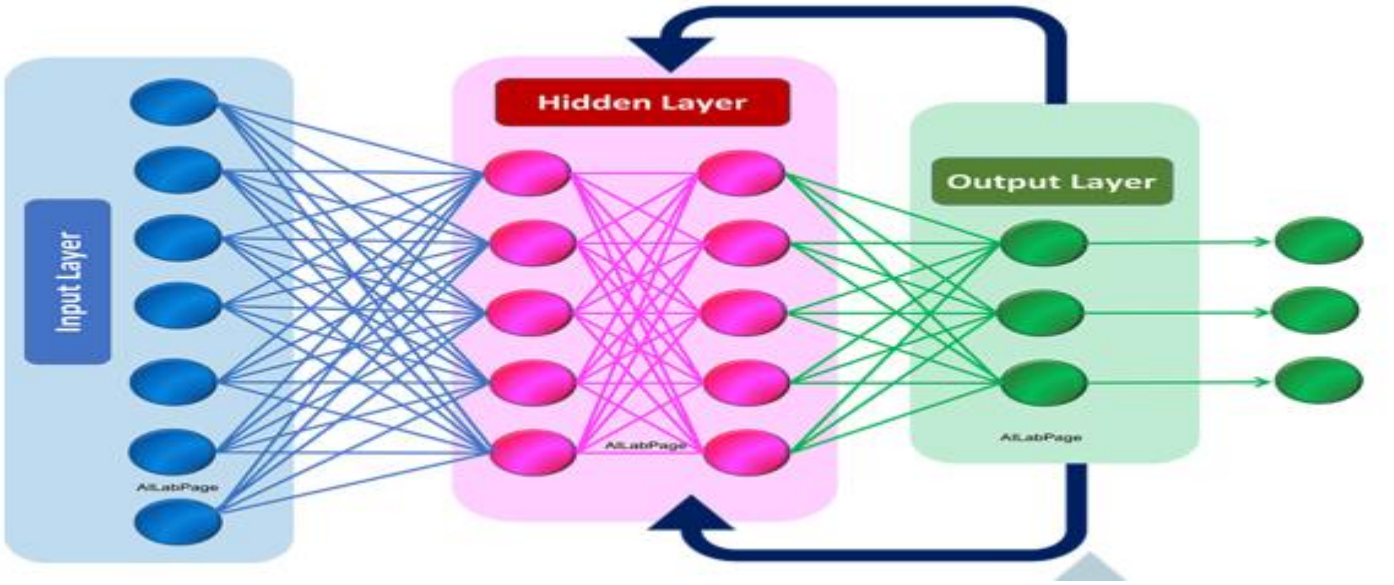
Applications:

- Image recognition, object detection, medical imaging.





3. Recurrent Neural Network (RNN)

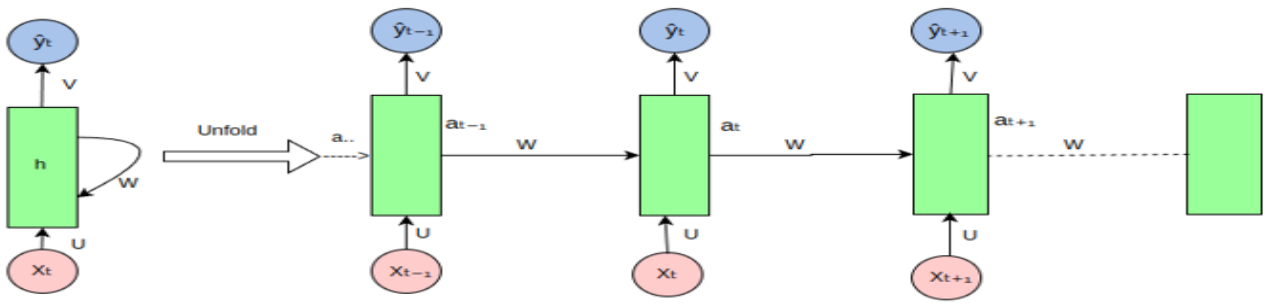


Definition:

- Designed for **sequential data** where previous outputs influence future ones.
- Unlike FNNs, RNNs have **feedback loops** that maintain a **hidden state**.

Architecture:

- Each neuron receives input from the current timestep **and** the previous hidden state.



Key Features of RNNs

Memory:

RNNs retain information from previous inputs in a sequence, which influences future predictions.

This gives them an advantage over traditional neural networks, where inputs and outputs are treated as independent.

Sequential Data:

Ideal for time-series data (e.g., stock prices) and sequential data (e.g., sentences in language translation).

Unidirectional:

Standard RNNs process information in a single direction (from past to future), making them unable to account for future events in predictions.

Working of Recurrent Neural Network (RNN)

- An RNN consists of multiple units with fixed activation functions, one for each time step.
- Each unit maintains a **hidden state**, representing the network's memory of past information.
- The hidden state is updated at every time step to capture changes in the sequence context.

State Update Equation:

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

With Activation Function (tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

- h_t: current hidden state
- h_{t-1}: previous hidden state
- x_t: current input
- W_{hh}: recurrent weight
- W_{xh}: input weight

Output Calculation:

$$y_t = W_{hy}h_t$$

where W_{hy} is the output layer weight.

All parameters are optimized using **Backpropagation Through Time (BPTT)**, a variant of backpropagation designed for sequential data.

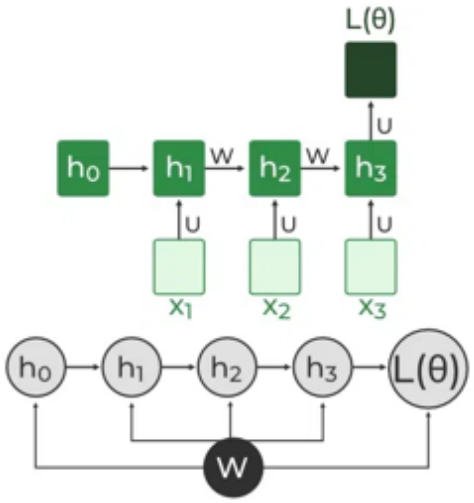
Learning Process:

Backpropagation Through Time (BPTT):

- RNNs use BPTT, an extension of the traditional backpropagation algorithm, to calculate errors and update weights.
- The key difference is that BPTT sums errors across time steps, as RNNs share parameters across layers.

Gradient Descent:

- The model adjusts weights using gradient descent to minimize errors over time.



•L(θ)(loss function) depends on h3
•h3 in turn depends on h2 and W
•h2 in turn depends on h1 and W
•h1 in turn depends on h0 and W
•where h0 is a constant starting state.
$\partial L(\theta)\partial W = \sum t=1T \partial L(\theta)\partial W$

1. Input sequence is processed one step at a time.
2. Hidden state stores context from previous steps.
3. Output depends on both current input and past information.

Structure of RNNs

Shared Parameters:

RNNs share the same parameters (weights) across each time step, unlike traditional feedforward networks.

This parameter sharing reduces complexity.

Feedback Loops:

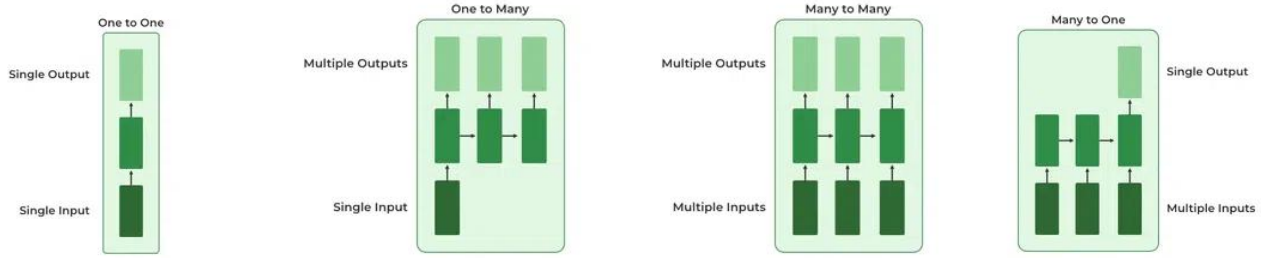
RNNs have loops that allow the network to pass information from one time step to the next.

Input-output configurations:

One-to-Many: Single input produces multiple outputs (e.g., image captioning).

Many-to-One: Multiple inputs produce a single output (e.g., sentiment analysis).

Many-to-Many: Multiple inputs produce multiple outputs (e.g., machine translation).



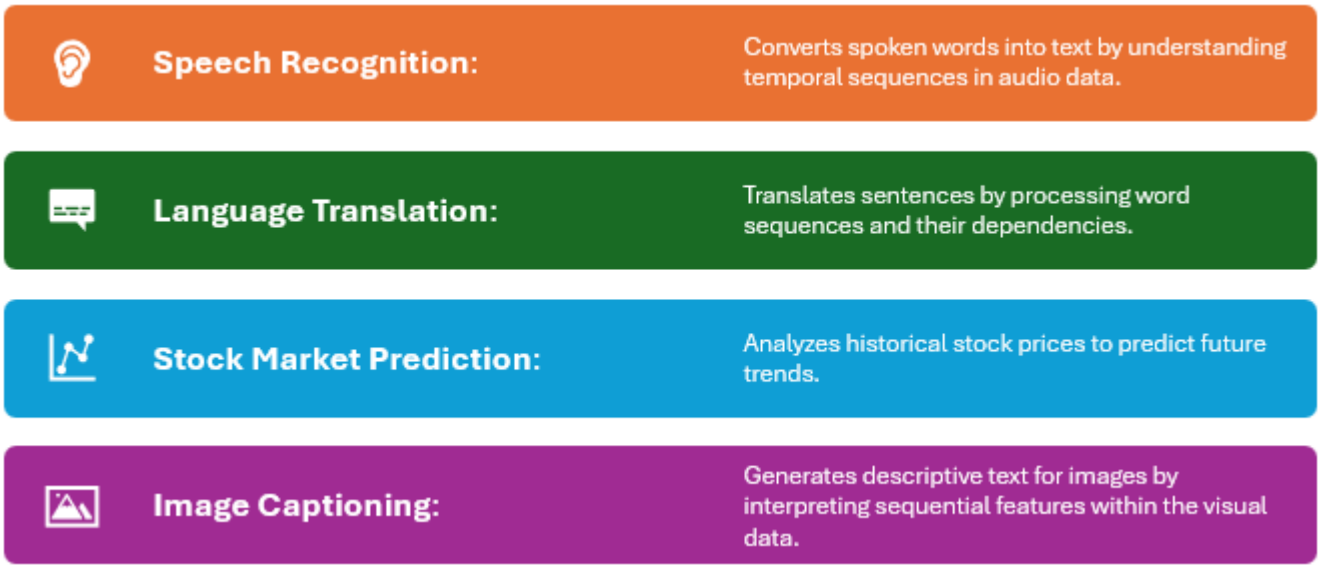
One to One: This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.

One To Many: In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.

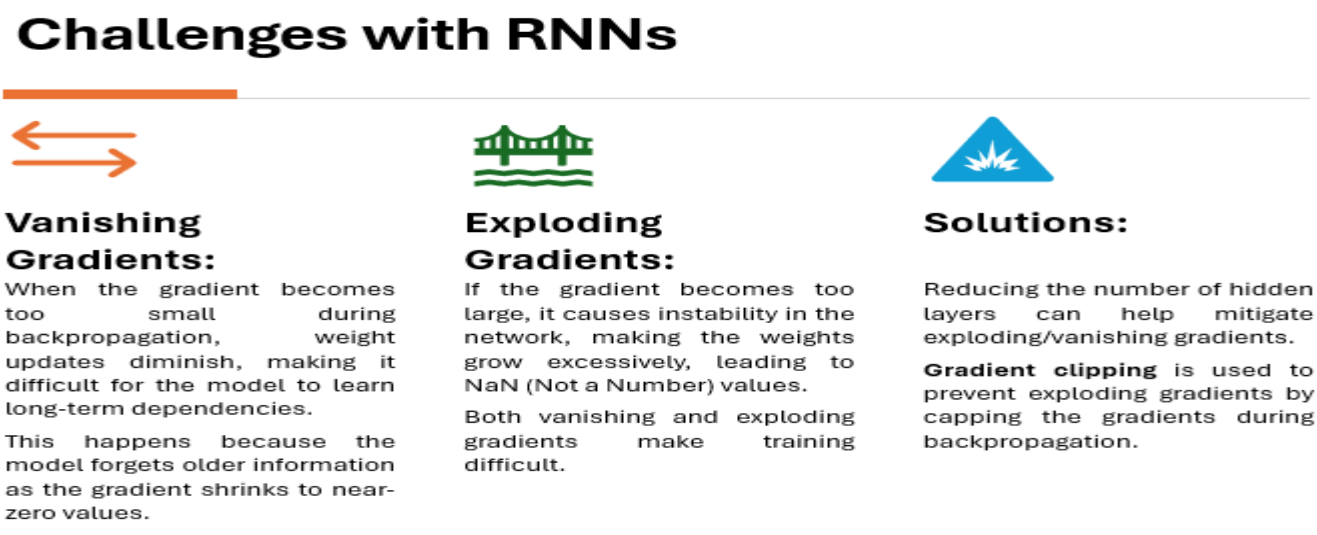
Many to One: In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.

Many to Many: In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.

Applications:



- Time-series forecasting, speech recognition, language modelling.



Variants of RNNs:

Long Short-Term Memory (LSTM):

LSTM is an improved version of RNNs designed to overcome the vanishing gradient problem.

It uses memory cells and gates (input, forget, and output gates) to learn and retain longer-term dependencies, making it suitable for more complex tasks.

Gated Recurrent Units (GRU):

A simplified variant of LSTM, GRU also solves the vanishing gradient problem while using fewer gates than LSTM, making it more efficient.

Advantages of RNNs:

• **Sequential Data Handling**:
  • RNNs are excellent for tasks involving sequences, where previous inputs influence future predictions.
• **Parameter Sharing**:
  • RNNs share weights across time steps, making them more efficient than other networks in learning sequential data.

Disadvantages of RNNs:

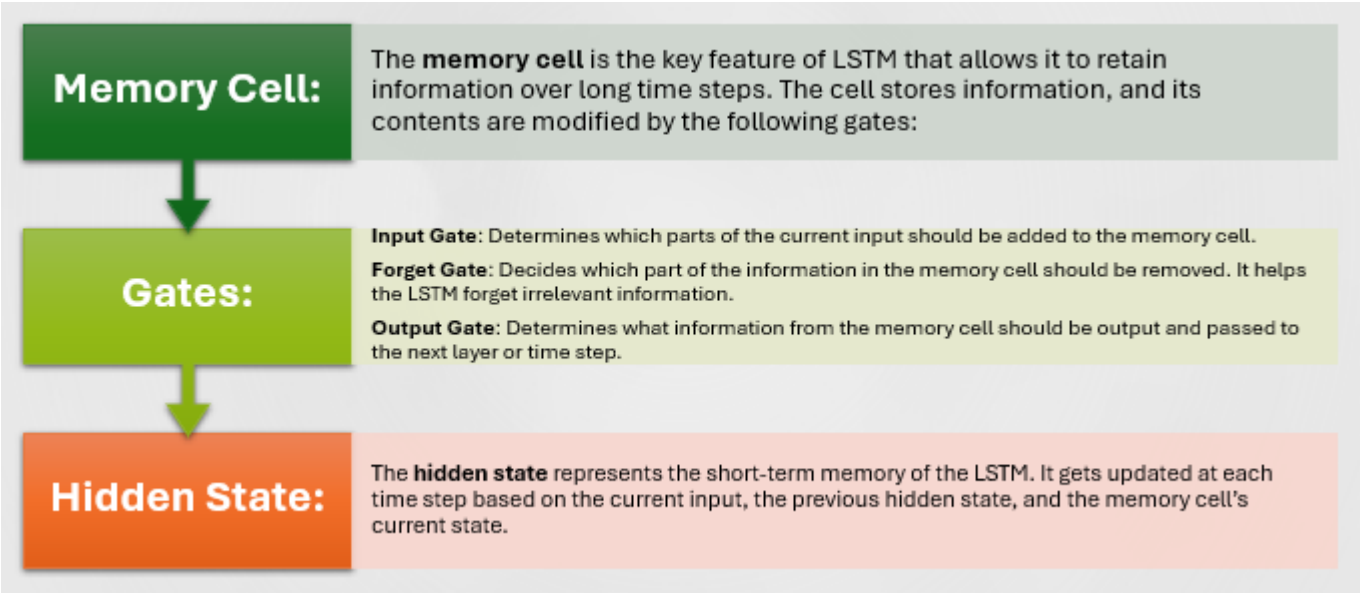
• **Training Complexity**:
  • Training RNNs can be difficult and time-consuming, especially with long sequences, due to exploding and vanishing gradients.
• **Computational Cost**:
  • Large datasets and deep RNNs require significant computational power, making the model slow and resource-intensive.
• **Long Training Times**:
  • RNNs may take longer to train compared to other neural network types due to the complexity of learning time dependencies.

4. Long Short-Term Memory Networks (LSTM)

Definition:

- A special type of RNN that solves the **vanishing gradient problem**.
- Can remember long-term dependencies using a **memory cell**.

Architecture:



- Contains **cell state** and three gates:
    1. **Input Gate:** Decides what new information to store.
    2. **Forget Gate:** Decides what information to discard.
    3. **Output Gate:** Controls what part of memory to output.

Bidirectional LSTM (BiLSTM):

Bidirectional LSTM extends the standard LSTM by processing the input sequence in both forward and backward directions.

This is particularly useful in tasks where the entire sequence context is important for prediction.

One LSTM processes the sequence in a forward direction (from start to end).

Another LSTM processes the sequence in a backward direction (from end to start).

The outputs from both LSTMs are combined (concatenated or averaged), which enables the network to capture information from both past and future contexts.

Stacked LSTM:

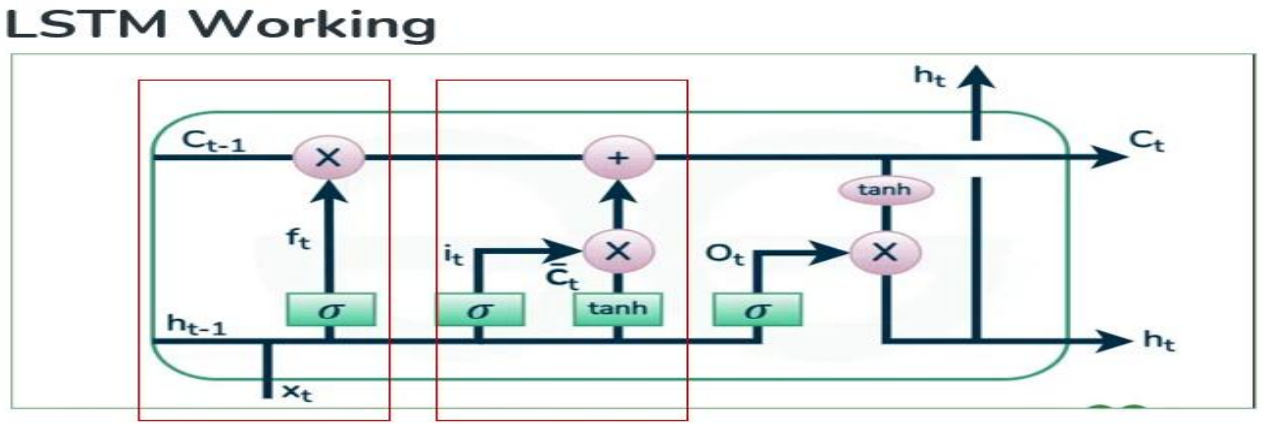
- **Stacked LSTM networks** consist of multiple LSTM layers stacked on top of each other. Each layer captures different levels of abstraction and temporal dependencies:
    - **Lower layers** capture simple patterns and short-term dependencies.
    - **Higher layers** capture more complex patterns and long-term dependencies.

Deep LSTMs

- (with stacked layers) are particularly effective for handling highly complex sequential data and are commonly used in applications like natural language processing (NLP) and speech recognition.

Working:

- Information flows through these gates to selectively retain or forget data.
- Maintains long-term context across long sequences.



1. Forget Gate

Function:

The forget gate decides which information from the previous cell state (C_{t-1}) should be discarded. It allows the LSTM to "forget" irrelevant data and retain only useful information.

Inputs:

- x_t: Current input at time step t
- h_{t-1}: Hidden state from the previous time step

Computation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Where:

- W_f: Weight matrix for the forget gate
- b_f: Bias term
- σ: Sigmoid activation function

Interpretation:

The output f_t ranges between 0 and 1 for each value in the cell state:

- **0** → Completely forget the information
- **1** → Fully retain the information

2. Input Gate

Function:

The input gate determines which new information from the current input should be stored in the cell state.

Inputs:

- x_t: Current input at time step t
- h_{t-1}: Hidden state from the previous time step

Computation Steps:

1. **Regulate the input with a sigmoid function:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

   This decides which values to update.

2. **Create candidate cell state values:**

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

   These are potential new values to be added to the cell state.

3. **Update the cell state:**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

   Where ⊙ denotes element-wise multiplication.

Interpretation:

This step determines how much of the previous cell state to keep and how much new information to add.

3. Output Gate

Function:

The output gate controls which part of the cell state will be output as the new hidden state h_t. It determines what information should be exposed to the next time step.

Inputs:

- x_t: Current input
- h_{t-1}: Previous hidden state

Computation:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Hidden State Update:

$$h_t = o_t \odot \tanh(C_t)$$

Interpretation:

This equation decides which parts of the updated cell state should be sent forward as the output (hidden state).

Applications:

- Text generation, machine translation, video analysis.

Difference between RNN and LSTM

Feature	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)
Architecture	Simple network with a single hidden state	Complex network with memory cells and three gates (input, forget, output)
Handling Long-term Dependencies	Struggles with long-term dependencies due to vanishing gradient problem	Designed to handle long-term dependencies using memory cells
Vanishing Gradient Problem	Highly susceptible, leading to poor performance for long sequences	Mitigates the vanishing gradient problem via gating mechanisms
Memory	Uses the hidden state to store short-term information	Uses memory cells and gates to store both short-term and long-term information
Gates	No gates, just relies on the hidden state	Input, forget, and output gates control information flow
Learning Ability	Learns short-term patterns effectively but struggles with long-term ones	Learns both short-term and long-term patterns effectively

Feature	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)
Training Time	Faster due to simpler architecture	Slower due to the complex structure and additional computations
Applications	Simple tasks like time-series prediction, speech recognition (limited sequence lengths)	Complex tasks like machine translation, long-sequence time-series prediction, text generation
Gradient Flow	Gradient tends to diminish or explode over long sequences	Better gradient flow due to the forget gate and memory cell mechanism
Suitability	Suitable for problems with short sequential dependencies	Suitable for problems with long-term dependencies and large time gaps between important events
Complexity	Simpler architecture with fewer parameters	More complex architecture with more parameters to train

5. Autoencoders

Definition:

- **Unsupervised neural networks** used for **representation learning** or **dimensionality reduction**.
- Aim to **reconstruct input data** after compressing it.

Architecture:

1. **Encoder:** Compresses input into a **latent representation (bottleneck)**.
2. **Decoder:** Reconstructs input from the latent code.

Working:

1. Network is trained to minimize **reconstruction error** between input and output.
2. The encoder learns meaningful compressed features.

Applications:

- Noise reduction, anomaly detection, feature extraction, image compression.

6. Generative Adversarial Networks (GANs)

Definition:

- Consist of **two networks** — a **Generator** and a **Discriminator** — competing in a **minimax game**.
- Used for **data generation** (e.g., realistic images, videos).

Architecture:

- **Generator (G):** Creates fake data samples from random noise.
- **Discriminator (D):** Classifies whether a sample is real or fake.

Working:

1. Generator tries to fool the Discriminator by producing realistic data.
2. Discriminator learns to differentiate between real and fake.
3. Both networks improve simultaneously through adversarial training.

Applications:

- Image synthesis, style transfer, super-resolution, deepfakes.

Network Type	Data Type	Key Idea	Main Application
FNN	Tabular	One-way feedforward flow	Basic classification/regression
CNN	Spatial/Image	Convolutional feature extraction	Image recognition
RNN	Sequential	Feedback through time	Time series, speech
LSTM	Sequential	Long-term memory via gates	NLP, long sequences
Autoencoder	Unsupervised	Encode-decode for feature learning	Dimensionality reduction
GAN	Generative	Adversarial training (G vs D)	Image generation

Graph analytics for social network analysis

Graph analysis is a data analysis technique that studies the relationships between objects in a graph, or network, to understand how they are connected. It can be used to discover hidden information, optimize processes, and deliver key information to decision makers.

1. Introduction

- **Graph analytics** is the study of relationships and connections among entities represented as a **graph**.
- A **graph** consists of:
    - **Nodes (vertices):** Represent entities (e.g., people, users, products).
    - **Edges:** Represent relationships (e.g., friendships, interactions, purchases).

2. Graph Analytics Concepts

- **Degree:** Number of connections a node has.
- **Path:** Sequence of edges connecting two nodes.
- **Centrality:** Measures importance of a node in the network.
- **Community Detection:** Identifies clusters or groups of closely connected nodes.
- **Similarity Measures:** Quantify how similar two nodes are based on shared connections or attributes.

GRAPH ANALYSIS

- **Social networks**
  - Graph analysis can help identify influencers and communities in social media networks. It can also help determine how many people a person is connected to, or how many degrees of separation exist between two people.
- **Cyber threat detection**
  - Graph analysis can help detect cyber threats.
- **Financial fraud detection**
  - Graph analysis can help identify suspicious patterns and alerts in financial data, such as circular transactions or shell companies.
- **Life sciences**
  - Graph analysis can help researchers navigate and understand complex data, such as genomic, clinical, or drug data.
- **IT operations management**
  - Graph analysis can help understand dependencies within an IT infrastructure, and perform impact or root cause analysis.
- Some graph analytics tools include:
  - Neo4j, CosmosDB, Spark, RedisGraph, Memgraph, Amazon Neptune, TigerGraph, and JanusGraph

3. Social Network Analysis (SNA)

Definition:

- The process of analyzing social relationships using graph-based methods.
- Helps uncover hidden patterns, influential users, and community structures.

Techniques:

1. **Centrality Measures:**
   - **Degree Centrality:** Influence based on number of direct connections.
   - **Betweenness Centrality:** Measures a node's control over information flow.
   - **Closeness Centrality:** Indicates how quickly a node can reach others.
2. **Community Detection:**
   - Identifies groups of users with strong internal connections.
   - Algorithms: **Louvain**, **Girvan–Newman**, **Label Propagation**.
3. **Link Prediction:**
   - Predicts potential new connections between users.
   - Based on shared friends, common interests, or graph similarity.

Centrality Measures

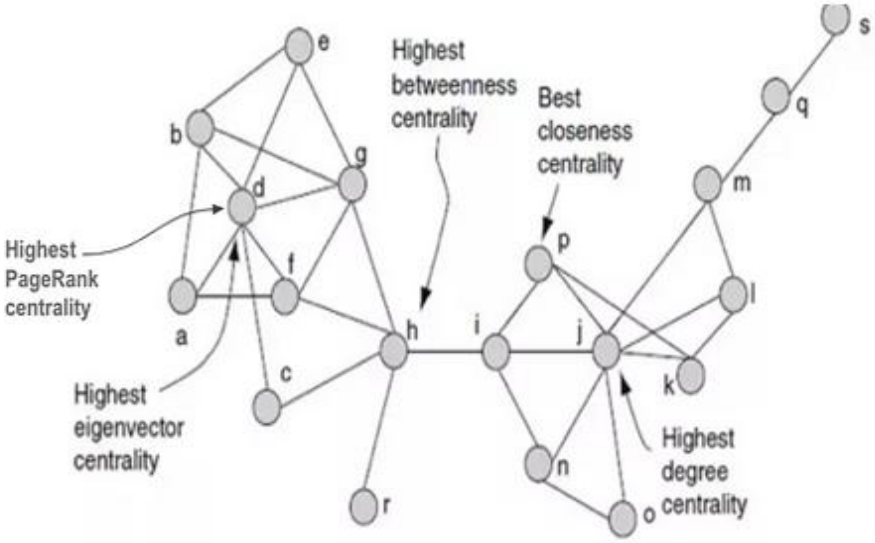
Highly central nodes play a key role of a network, serving as hubs for different network dynamics.

However, the definition and importance of centrality might differ from case to case, and may refer to different centrality measures:

Degree — the amount of neighbors of the node	**EigenVector / PageRank** — iterative circles of neighbors	**Closeness** — the level of closeness to all of the nodes	**Betweenness** — the amount of short path going through the node

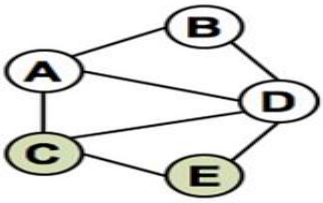
Centrality Measures

- Different measures can be useful in different scenarios such web-ranking (page-rank), critical points detection (betweenness), transportation hubs (closeness) and other application



Network Theory

Nodes



Edges

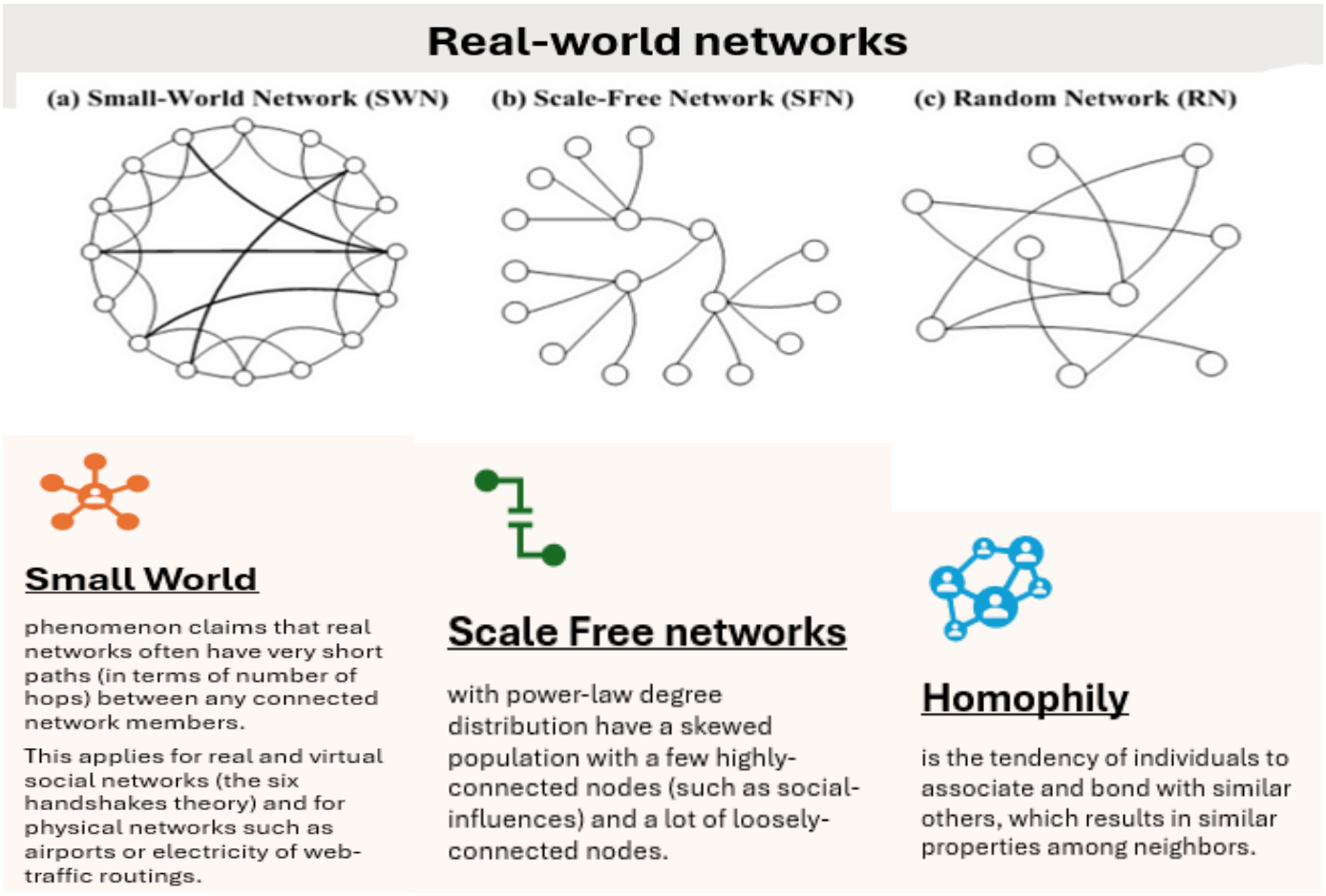
Nodes (A,B,C,D,E in the example)

usually representing entities in the network, and can hold self-properties (such as weight, size, position and any other attribute) and network-based properties (such as *Degree*- number of neighbors or *Cluster*- a connected component the node belongs to etc.).

Edges

represent the connections between the nodes, and might hold properties as well (such as weight representing the strength of the connection, direction in case of asymmetric relation or time if applicable).

These two basic elements can describe multiple phenomena, such as *social connections, virtual routing network, physical electricity networks, roads network, biology relations network* and many other relationships.



Applications:

- Detecting **influencers** in social media.
- Studying **information spread** and **viral trends**.
- Detecting **fraud** or **fake accounts**.

4. Graph Analytics in Recommendation Systems

Definition:

- Uses graph structures to model relationships between **users**, **items**, and **interactions** (e.g., ratings, clicks, purchases).

Approaches:

1. **User-Item Bipartite Graphs:**
    - Users and items are represented as two sets of nodes.
    - Edges represent user–item interactions.
2. **Graph-Based Similarity:**
    - Recommends items connected to similar users (user-based CF).
    - Or users who liked similar items (item-based CF).
3. **Random Walks and Personalized PageRank:**
    - Estimate node relevance by simulating traversal in the graph.
    - Example: Google's **PageRank** algorithm.

4. **Graph Neural Networks (GNNs):**
    - Learn embeddings for users and items by aggregating neighbourhood information.
    - Improve accuracy of personalized recommendations.

Applications:

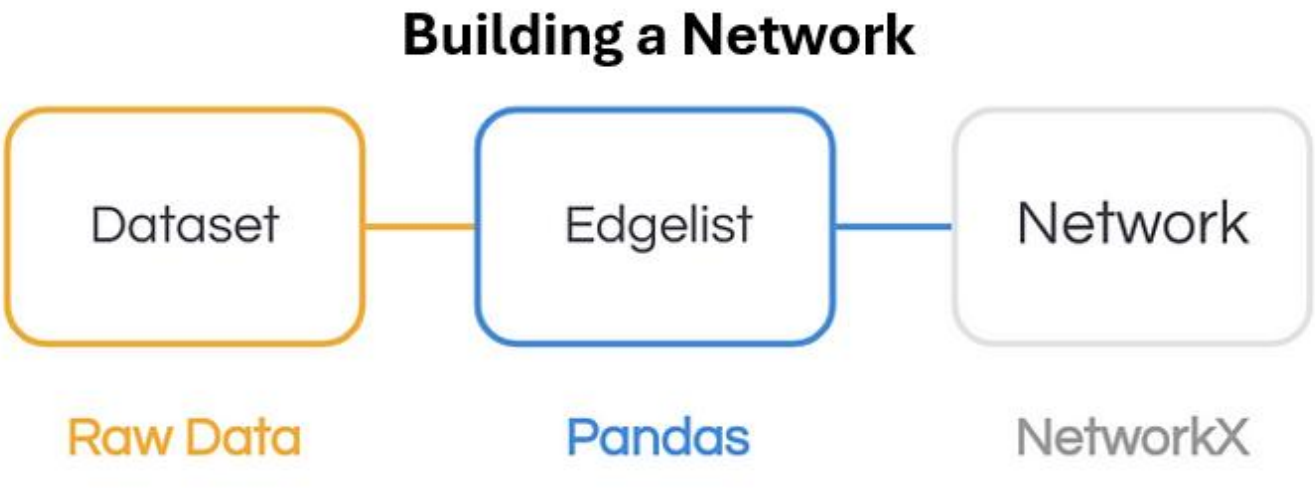
- **E-commerce:** Product and movie recommendations.
- **Social media:** Friend suggestions and content feeds.
- **Music streaming:** Song or playlist recommendations.

5. Advantages of Graph Analytics

- Captures **complex relationships** between entities.
- Enables **community detection**, **influence analysis**, and **link prediction**.
- Improves personalization and relevance in recommendation systems.

6. Example

- In a social graph:
    - **Nodes:** Users (A, B, C, D)
    - **Edges:** Friendships or interactions
    - Graph analytics can find:
        - Most influential users (centrality)
        - New friend suggestions (link prediction)
        - Similar interest groups (community detection)



- Networks can be constructed from various datasets, as long as we're able to describe the relations between nodes.
- In the following example we'll build and visualize the Eurovision 2018 votes network (based on official data) with **Python *networkx*** package.
    - We'll **read the data** from excel file to a ***pandas*** dataframe to get a tabular representation of the votes.
    - Since each row represents all of the votes of each country, we'll **melt** the dataset to make sure that each row represents a single vote (*edge*) between two countries (*nodes*).
    - Then, we will **build a directed graph** using *networkx* from the edgelist we have as a pandas dataframe.

Text Analytics and Natural Language Processing (NLP) on Big Data

1. Introduction

- **Text Analytics** is the process of extracting meaningful information, trends, and patterns from large volumes of textual data.
- **Natural Language Processing (NLP)** combines **linguistics**, **computer science**, and **machine learning** to help computers understand and interpret human language.
- With **Big Data**, NLP techniques are applied at scale to process millions of documents, social media posts, reviews, and more.

2. Text Mining and Feature Extraction

Definition:

- **Text Mining** involves transforming unstructured text into structured data for analysis.
- **Feature Extraction** converts text into numerical features usable by machine learning models.

Key Steps:

1. **Text Preprocessing:**
    - Tokenization (splitting sentences into words)
    - Stop-word removal (e.g., "the", "and")
    - Stemming/Lemmatization (reducing words to root form)
2. **Feature Representation Methods:**
    - **Bag of Words (BoW):** Counts word occurrences.
    - **TF-IDF (Term Frequency–Inverse Document Frequency):** Weights words by importance.
    - **Word Embeddings:** Represent words as dense vectors (Word2Vec, GloVe).

Applications:

- Document classification
- Spam detection
- Information retrieval

3. Sentiment Analysis

Definition:

- The process of identifying and classifying opinions or emotions expressed in text (positive, negative, neutral).

Approaches:

1. **Lexicon-based:** Uses predefined dictionaries of positive/negative words.

2. **Machine Learning-based:** Uses labelled datasets to train classifiers (e.g., logistic regression, LSTM).
3. **Deep Learning-based:** Uses embeddings with CNNs or RNNs for contextual understanding.

Applications:

- Social media monitoring
- Product review analysis
- Political opinion tracking

Example:

"The movie was fantastic!" → Positive sentiment
"The service was terrible!" → Negative sentiment

4. Topic Modelling

Definition:

- An **unsupervised learning technique** that discovers hidden themes or topics within large text collections.

Key Techniques:

1. **Latent Dirichlet Allocation (LDA):** Represents each document as a mixture of topics and each topic as a mixture of words.
2. **Non-negative Matrix Factorization (NMF):** Decomposes term-document matrix to reveal topic patterns.
3. **BERTopic (Modern):** Uses transformer-based embeddings and clustering.

Applications:

- News and article categorization
- Research paper analysis
- Discovering customer concerns or trends

Example:

In tweets about airlines:

- Topic 1 → "Flight delays, cancellations"
- Topic 2 → "Staff behavior, service quality"

5. Language Modelling and Embeddings

a. Language Modelling

- Predicts the probability of a sequence of words.
- Used in **text generation**, **speech recognition**, **autocorrect**, etc.

Types:

1. **Statistical Models:**
    - *n-gram models* estimate word probabilities based on the previous n words.

- Example: Bigram model —

$$P(w_n|w_{n-1})$$

2. **Neural Language Models:**
    - Use neural networks (e.g., RNN, LSTM, Transformer).
    - Learn long-term dependencies and context.

b. Word Embeddings

- Dense vector representations capturing **semantic meaning** of words.
- Words with similar meaning have similar vector representations.

Popular Methods:

- **Word2Vec (Skip-gram, CBOW)**
- **GloVe (Global Vectors)**
- **BERT and Transformer embeddings**

Applications:

- Machine translation
- Chatbots and virtual assistants
- Semantic search

Concept	Purpose	Technique/Model	Key Application
Text Mining & Feature Extraction	Convert raw text to features	BoW, TF-IDF, embeddings	Text classification
Sentiment Analysis	Detect emotions/opinions	Lexicon, ML, DL models	Reviews, social media
Topic Modelling	Discover hidden topics	LDA, NMF, BERTopic	News, feedback analysis
Language Modelling & Embeddings	Understand word context	n-grams, RNNs, Transformers	Translation, text generation

Ensemble Learning Methods

1. Introduction

- **Definition:**
  Ensemble Learning combines **multiple individual models (called base learners)** to produce a stronger overall model with better accuracy and generalization.
- The idea is:

"Many weak models together form a strong model."

- Improves performance by **reducing variance, bias, or both**.

2. Types of Ensemble Methods

(a) Bagging (Bootstrap Aggregating)

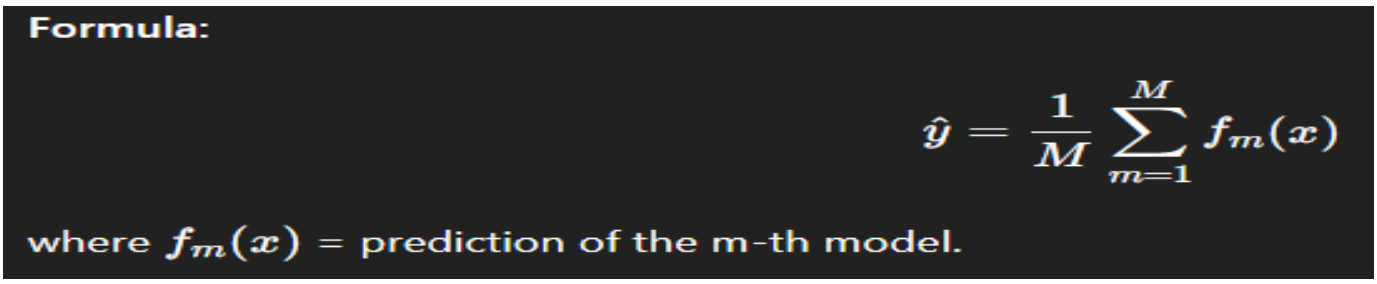
Concept:

- Reduces **variance** by training multiple models independently on **different random subsets** of data.
- The final prediction is made by **averaging (for regression)** or **majority voting (for classification)**.

Steps:

1. Generate multiple **bootstrap samples** (random samples with replacement) from the dataset.
2. Train a **separate model** (e.g., decision tree) on each sample.
3. Combine outputs of all models to get the final result.

Formula:



Example:

- **Random Forest** = Bagging + Decision Trees.

Advantages:

- Reduces overfitting.
- Works well with high variance models (e.g., decision trees).

(b) Boosting

Concept:

- Reduces **bias and variance** by training models **sequentially**, where each new model focuses on the **errors** made by previous ones.

Steps:

1. Train the first model on the dataset.
2. Increase the weights of misclassified samples.
3. Train the next model to correct those errors.
4. Combine all weak models into one strong model (weighted sum).

Formula:

Formula:

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

where α_m is the weight of each weak learner.

Popular Algorithms:

- **AdaBoost (Adaptive Boosting)**
- **Gradient Boosting**
- **XGBoost**, **LightGBM**, **CatBoost**

Advantages:

- High accuracy on structured/tabular data.
- Focuses learning where it's needed most.

Disadvantages:

- Sensitive to noisy data and outliers.
- Sequential nature → slower training.

(c) Stacking and Voting Techniques

1. Stacking (Stacked Generalization):

- Combines **predictions from multiple base models** using a **meta-model (blender)** that learns the best way to combine them.

Architecture:

1. **Level 0 (Base Models):** Train multiple models (e.g., SVM, Decision Tree, Logistic Regression).
2. **Level 1 (Meta Model):** Takes outputs of base models as inputs and learns the optimal combination.

Example:

- Base models: Decision Tree, KNN, SVM
- Meta model: Logistic Regression

Advantages:

- Captures diverse learning patterns from different models.
- Usually gives better results than single models.

2. Voting:

- Simple ensemble of models where the final output is decided by **majority vote (classification)** or **average (regression)**.

Types:

- **Hard Voting:** Each model votes for a class; the majority wins.
- **Soft Voting:** Averages predicted probabilities across models.

Example:

If three classifiers predict [A, A, B], → Final = **A (majority vote)**

Advantages:

- Simple to implement.
- Works well when individual models perform comparably.

3. Comparison Summary

Method	Key Idea	Training Type	Reduces	Example	Final Combination
Bagging	Train on random subsets	Parallel	Variance	Random Forest	Averaging/Voting
Boosting	Sequential correction of errors	Sequential	Bias + Variance	AdaBoost, XGBoost	Weighted Sum
Stacking	Combine diverse model outputs	Meta-learning	Both	Stacked Ensembles	Meta Model
Voting	Aggregate outputs directly	Parallel	Variance	Ensemble of classifiers	Majority/Average

4. Applications

- **Credit scoring**
- **Fraud detection**
- **Recommendation systems**
- **Predictive analytics** in healthcare and finance

Model Evaluation Techniques

Introduction

- After training a model, **evaluation** is essential to assess its **performance, accuracy, and generalization ability** on unseen data.
- Common techniques include **cross-validation**, **confusion matrix–based metrics**, and **ROC–AUC analysis**.

1. Cross-Validation

Definition:

- A **resampling technique** used to estimate how well a model generalizes to an independent dataset.
- Helps avoid **overfitting** and ensures **robust evaluation**.

Types:

1. **K-Fold Cross-Validation:**
    - Dataset is split into **K equal parts (folds)**.
    - Model is trained on (K–1) folds and tested on the remaining fold.
    - Process repeats K times, each fold used once as test data.
    - Final score = **average of all K test scores**.

$$Accuracy = \frac{1}{K}\sum_{i=1}^{K} Accuracy_i$$

Example: For 5-Fold CV → Train on 4 folds, test on 1, repeat 5 times.

2. **Leave-One-Out CV (LOOCV):**
    - Special case where K = number of samples (each observation is used once as test data).
3. **Stratified K-Fold:**
    - Used in classification — maintains class proportion in each fold.

Advantages:

- Provides reliable performance estimation.
- Utilizes entire dataset efficiently.

2. Confusion Matrix and Performance Metrics

Confusion Matrix:

A table that summarizes classification results by comparing predicted and actual labels.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics:

1. **Accuracy:**

   Proportion of total correct predictions.

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

2. **Precision:**

   How many predicted positives are truly positive.

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

3. **Recall (Sensitivity / True Positive Rate):**

   How many actual positives are correctly identified.

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

4. **F1-Score:**

   Harmonic mean of Precision and Recall (balances both).

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Interpretation:

- **High Precision:** Few false positives.
- **High Recall:** Few false negatives.
- **F1-Score:** Best when both precision and recall matter.

Example:

Suppose a model classifies spam emails:

- TP = 80, TN = 900, FP = 20, FN = 10

  Then:

$$Accuracy = \frac{980}{1010} = 97.03\%, \quad Precision = \frac{80}{100} = 0.8, \quad Recall = \frac{80}{90} = 0.89, \quad F1 = 0.84$$

3. ROC Curve and AUC

ROC (Receiver Operating Characteristic) Curve:

- Graphical representation of the **trade-off between True Positive Rate (TPR)** and **False Positive Rate (FPR)** across different classification thresholds.
- $$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}$$
- The curve plots **TPR (y-axis)** vs **FPR (x-axis)**.

Interpretation:

- **Ideal ROC Curve:** Passes close to the top-left corner (high TPR, low FPR).
- **Diagonal Line:** Represents random guessing (AUC = 0.5).

AUC (Area Under Curve):

- Represents overall model performance — **probability that the model ranks a positive example higher than a negative one.**
- $$0.9 \leq AUC \leq 1.0 \Rightarrow Excellent$$
$$0.8 \leq AUC < 0.9 \Rightarrow Good$$
$$0.5 \Rightarrow Random$$

Advantages:

- Works well for **imbalanced datasets**.
- Threshold-independent metric.

4. Summary Table

Technique	Purpose	Formula / Key Idea	Suitable For
Cross-Validation	Estimate model performance	Average accuracy across folds	Any ML model
Confusion Matrix	Show prediction breakdown	TP, FP, TN, FN	Classification
Accuracy	Overall correctness	(TP + TN)/(Total)	Balanced data
Precision	Correctness of positive predictions	TP/(TP + FP)	When FP costly
Recall	Detection ability	TP/(TP + FN)	When FN costly
F1-Score	Balance of precision & recall	2PR/(P + R)	Imbalanced data
ROC–AUC	Measure ranking ability	Area under ROC	Binary classifiers