ADVANCED ANALYTICS ON BIG DATA

Unit 4

Content

Deep learning for big data analytics

neural networks

convolutional neural networks(CNNs)

recurrent neural networks (RNNs).

Text analytics and natural language processing (NLP) on big data

Graph analytics for social network analysis and recommendation systems

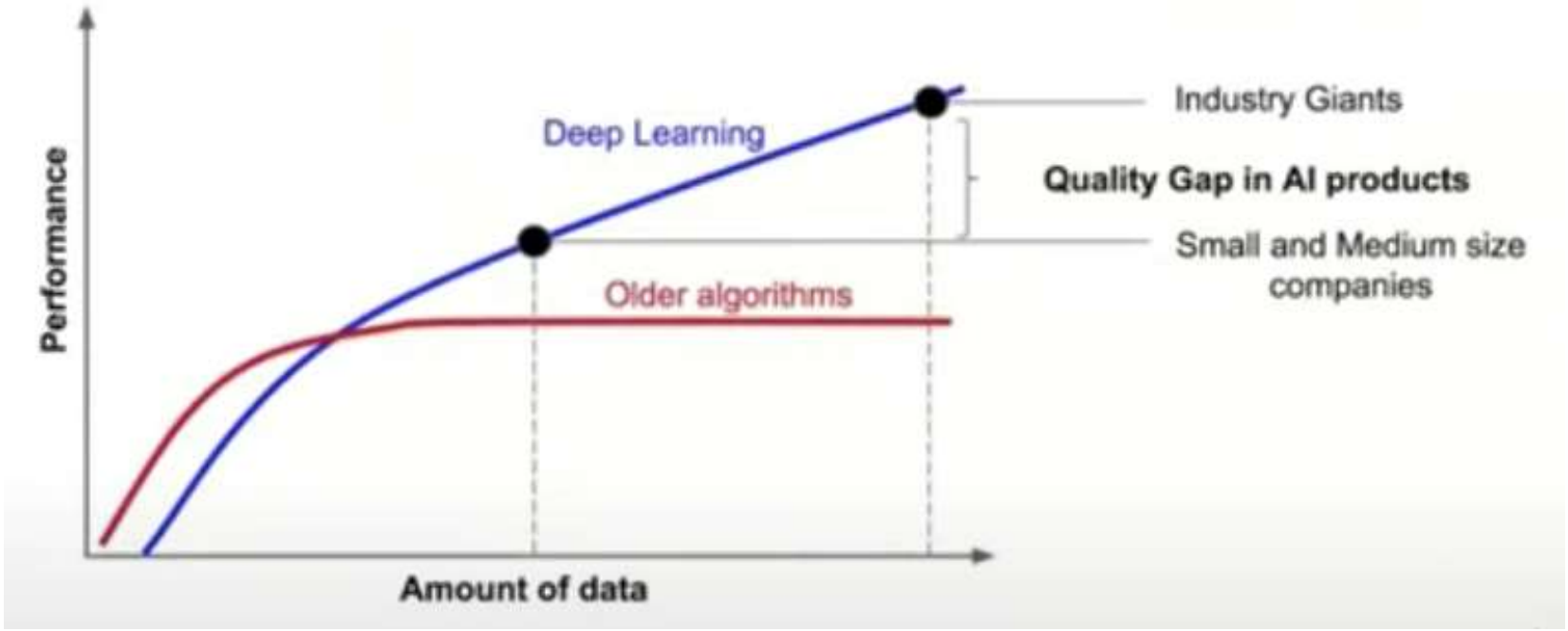
Ensemble learning methods and model evaluation techniques

Deep Learning

- https://youtu.be/6M5VXKLf4D4?si=LkBPIbB2mAyms_3u

Why do we need Deep Learning?

- Deep learning is subset of ML algorithm
- It work well if you provide more and more data.
- It prevents overfitting, unlike ML which after a threshold value becomes stagnate.

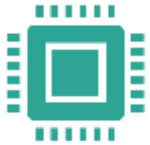


What is deep learning?

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

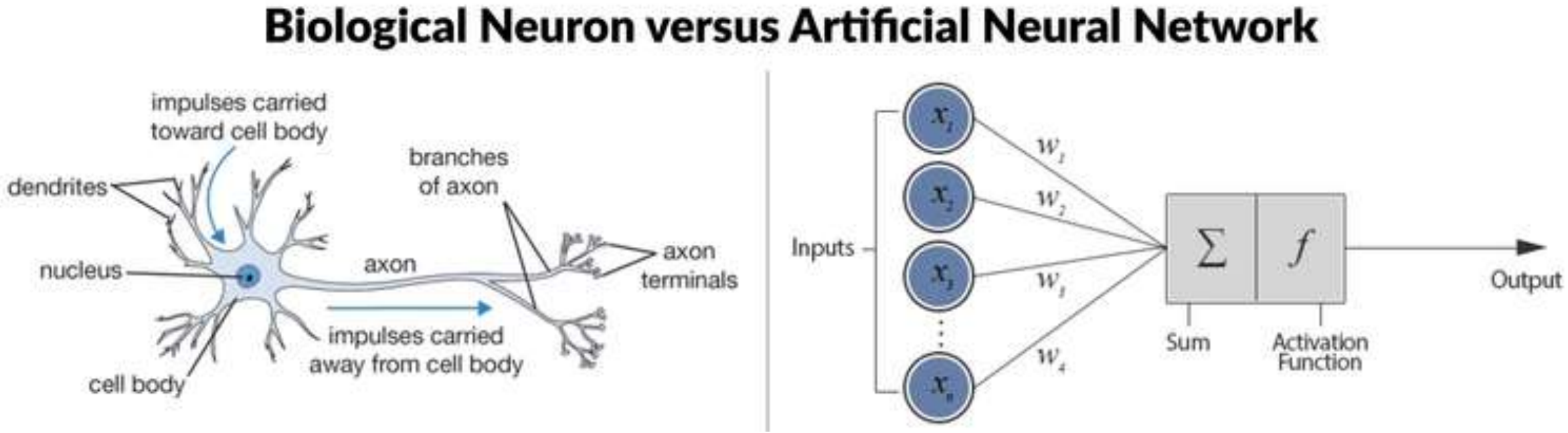


Deep learning models are computer files that data scientists have trained to perform tasks using an algorithm or a predefined set of steps.

Businesses use deep learning models to analyze data and make predictions in various applications.

What is deep learning?

- Deep learning is a subset of machine learning
  - that uses multilayered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain.
- 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.



ANN

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.

Key Architectures

Feedforward Neural Networks (FNN):	The simplest type of ANN where the information moves in only one direction, from input to output.
Convolutional Neural Networks (CNNs):	Primarily used for image recognition tasks, CNNs detect patterns and hierarchies in data using convolutional layers.
Recurrent Neural Networks (RNNs):	Used for sequence data like time series or text, RNNs keep track of past information through their loops, enabling learning from sequences.
Long Short-Term Memory Networks (LSTMs):	A special type of RNN that solves the vanishing gradient problem, allowing learning over long sequences of data.
Autoencoders:	Used for unsupervised learning, autoencoders compress data and then reconstruct it, useful for tasks like image denoising or anomaly detection.
Generative Adversarial Networks (GANs):	Consist of two neural networks (generator and discriminator) that compete, allowing for the generation of realistic synthetic data, such as images or text.

How deep learning works

Neural Networks:

- Neural networks consist of layers of neurons (also called nodes),
- simulate the way the human brain works
- Each neuron takes in data, processes it, and sends it to the next layer.
- The goal is to learn patterns from input data, make predictions, and adjust to minimize errors over time.

Core Elements of Neural Networks:

- **Inputs:** The data (e.g., images, text) fed into the model for learning.
- **Weights and Biases:** Weights determine the strength of the connection between neurons. Bias is an additional parameter added to adjust the output along with weights.
- **Activation Function:** Each neuron uses an activation function (like ReLU, Sigmoid) to introduce non-linearity into the network, which helps in learning complex patterns.

How deep learning works-eep Neural Networks (DNNs)

Deep neural networks consist of multiple **hidden layers** between the input and output layers.

These hidden layers enable the model to learn hierarchical features from the data:

Input Layer

- Takes in raw data (e.g., an image's pixel values).

Hidden Layers

- These layers perform computations on the input data to extract more abstract features (e.g., edges in an image in early layers, and complex shapes in later

Output Layer:

- Provides the final result, such as a classification label (e.g., "cat" or "dog").

How deep learning works: Forward Propagation

Forward propagation refers to the movement of data through the layers of the network from input to output.

The input data is multiplied by these weights and passed through an activation function to produce an output, which becomes the input for the next layer.

Each neuron in one layer is connected to neurons in the next layer, where each connection has a weight.

The process continues until the output layer provides the final prediction.

How deep learning works: **Backpropagation**

Backpropagation is the learning process where the neural network adjusts its weights and biases to minimize prediction errors:

The network compares its output to the actual result using a **loss function** (which measures error).

It calculates how much each neuron in each layer contributed to the error.

Using the **gradient descent algorithm**, it adjusts the weights and biases by moving backward from the output layer to the input layer, effectively "learning" from its mistakes and becoming more accurate over time.

Training the Model

Epochs:

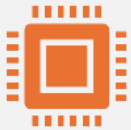
Training is done over multiple epochs, where the entire dataset is passed through the network multiple times to adjust weights.

Optimization:

Gradient descent optimizes the model by finding the minimum point of the loss function, where the error is lowest.

Computational Requirements

Deep learning models require massive computational resources



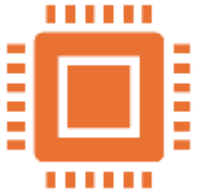
GPUs (Graphical Processing Units):

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.

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.

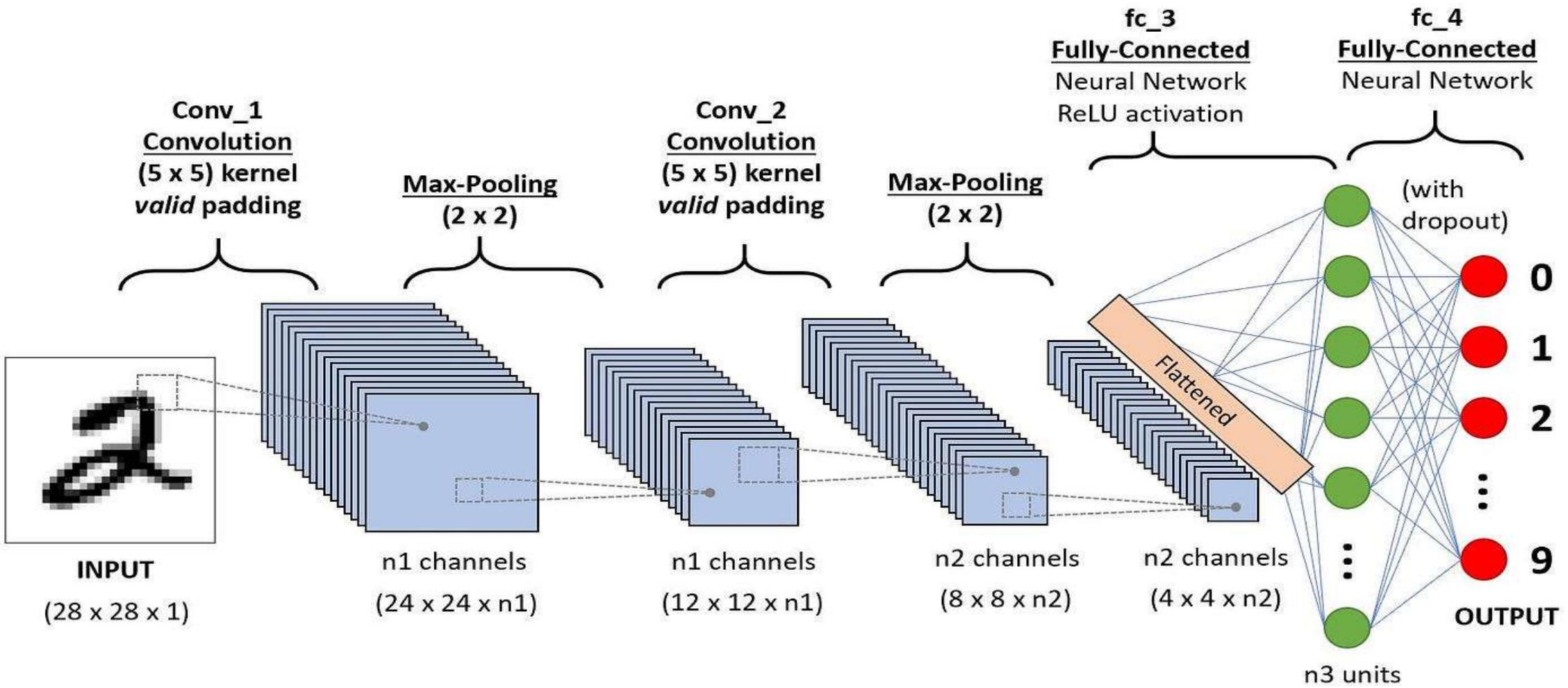
Improving Accuracy Over Time

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

Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)

CNNs (Convolutional Neural Networks) are specialized neural networks primarily used for **computer vision** tasks like image and video classification, object detection, pattern recognition, and face recognition.

CNNs excel at processing and understanding **visual data** by identifying patterns such as edges, shapes, and textures in images.

Structure of CNNs

Like traditional neural networks, CNNs are composed of **layers:**

- input layer
- hidden layers
- output layer

CNN layers are connected through nodes, each node having an associated **weight** and **threshold.**

If the output from a node exceeds the threshold, it activates and passes data to the next layer.

Key Layers in CNNs

Convolutional Layer:

The primary layer in a CNN that performs convolution operations on the input image, identifying features like edges, corners, and textures.

- Uses filters (kernels) to slide over the image, producing feature maps that highlight the presence of particular patterns.

Pooling Layer:

Reduces the size of the feature maps by downsampling, typically using operations like **max pooling**.

This layer helps in reducing computational load, extracting dominant features, and controlling overfitting.

Fully Connected (FC) Layer:

At the end of the network, the fully connected layers combine the high-level features learned by the convolutional layers and make predictions based on them.

How CNNs Work

CNNs process data layer by layer, extracting increasingly complex features from the input. For example:

Early layers focus on simple patterns (like edges, colors).	**Deeper layers** recognize complex patterns (like object shapes).	By the final layers, the CNN is able to recognize the entire object.

The process involves a **convolution** operation, where the network "works and reworks" the input data, refining its feature detection with each layer.

CNN Advantages

Efficient feature extraction:	Unlike traditional methods requiring manual feature extraction, CNNs automatically extract relevant features from high-dimensional data.
Image classification:	CNNs are superior to other networks when handling image, audio, or speech inputs.
Scalability:	CNNs can handle large datasets, making them ideal for tasks like **image recognition** in real-world applications.
Data exchange between layers:	CNNs efficiently pass data between layers, allowing for more complex data processing.

Pooling and Loss of Information

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.

Disadvantages of CNNs

Computationally demanding:

- CNNs require substantial computing power, typically relying on **GPUs** (Graphical Processing Units) for training.
- This can be expensive and time-consuming.

Complexity:

- Designing and tuning CNN architectures involves complex decisions regarding network layers, hyperparameters, and configurations.

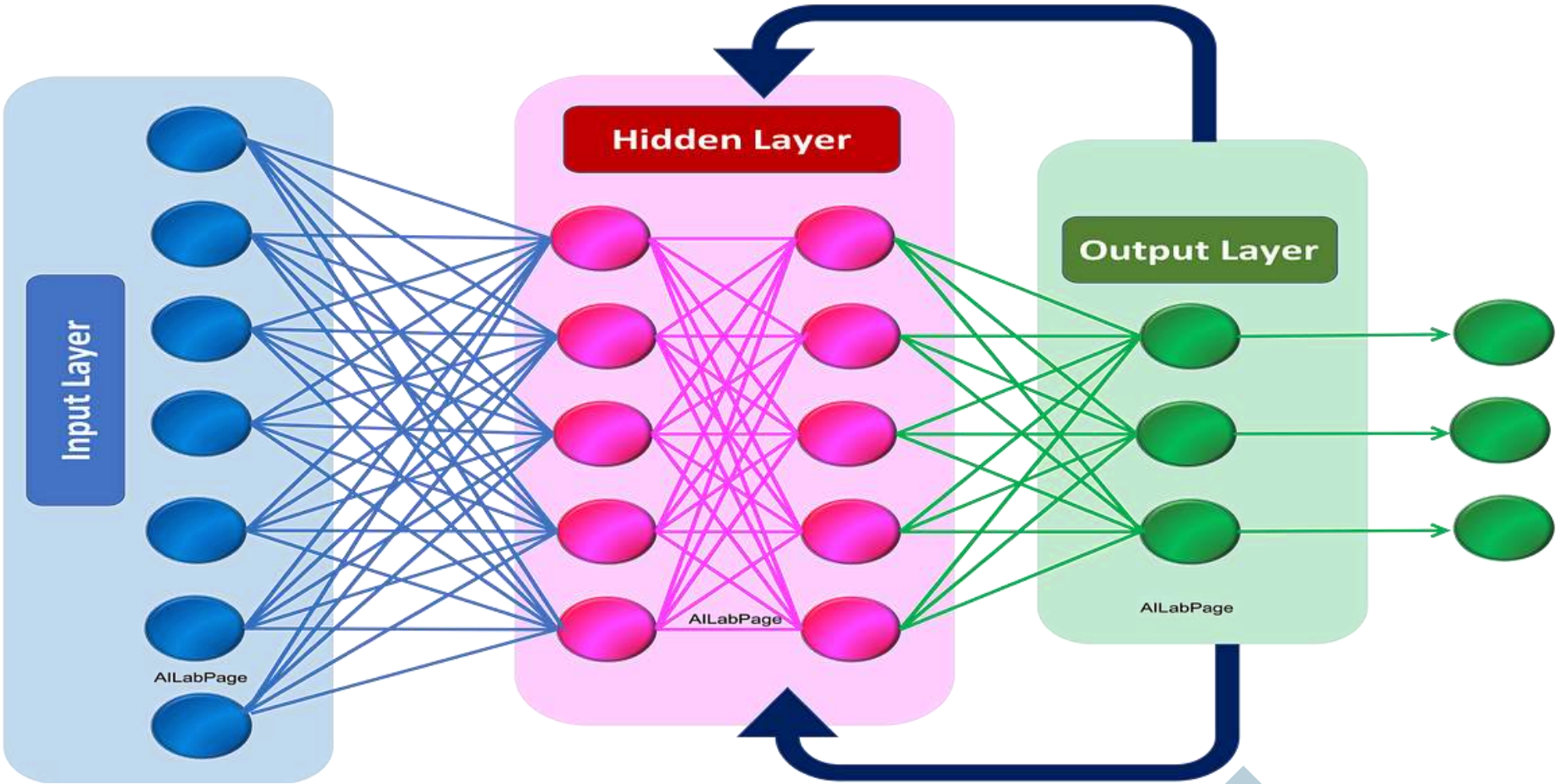
Expertise required:

- Successful CNN implementations need skilled practitioners with cross-domain knowledge, particularly in configuring and fine-tuning the model.

Modern Applications of CNNs

Image classification:	Used in tasks like classifying images of objects, animals, etc.
Object detection:	Identifying specific objects within an image.
Medical imaging:	Detecting abnormalities in medical scans.
Autonomous vehicles:	Used for interpreting surroundings by recognizing pedestrians, obstacles, and signs.

Recurrent Neural Networks



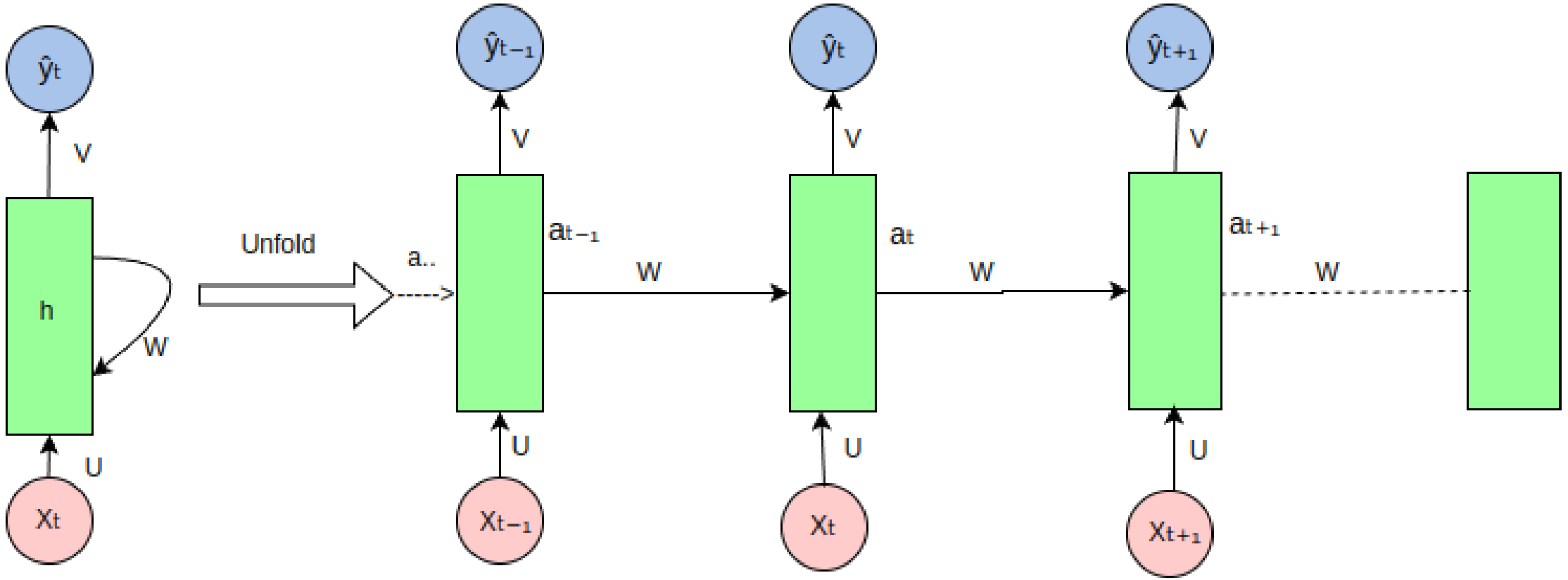
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are designed to handle **sequential or time-series data.**

This makes them useful for tasks like:

Natural language processing (NLP)
Speech recognition
Language translation
Stock market prediction,
Sales forecasting.

RNNs differ from other neural networks by using **feedback loops** to retain memory from previous inputs, allowing them to handle data with temporal dependencies.


ŷt
V
h
W
U
Xt
Unfold
a..
ŷt−1
V
at−1
W
U
Xt−1
ŷt
V
at
W
U
Xt
ŷt+1
V
at+1
W
U
Xt+1

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.

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:

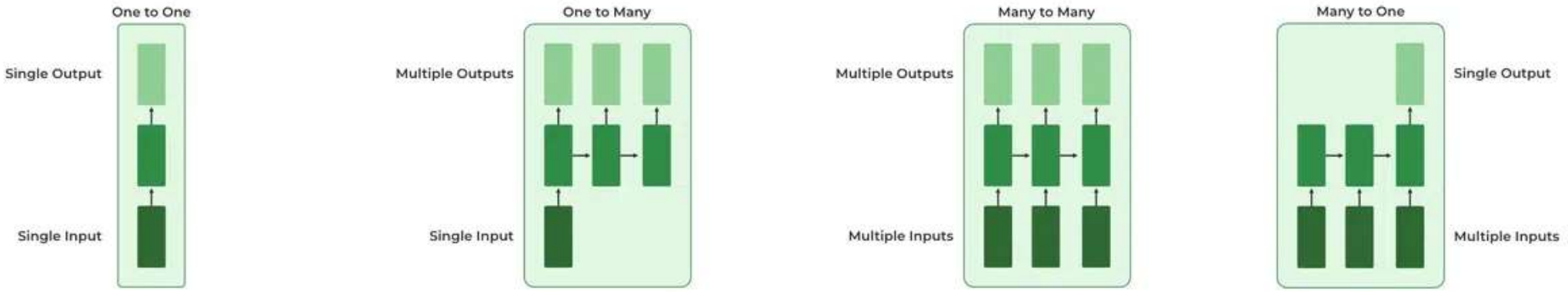
RNNs have loops that allow the network to pass information from one time step to the next.

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).



Types Of RNN

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.

How does RNN work?

- The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step.
- Each unit has an internal state which is called the hidden state of the unit.
- This hidden state signifies the past knowledge that the network currently holds at a given time step.
- This hidden state is updated at every time step to signify the change in the knowledge of the network about the past.
- The hidden state is updated using the following recurrence relation:-
- **The formula for calculating the current state:**

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

  - where,
    - h_t -> current state
    - h_{t-1} -> previous state
    - x_t -> input state

How does RNN work?

- **Formula for applying Activation function(tanh)**
  - $h_t=\tanh(W_{hh}h_{t-1}+W_{xh}x_t)$
  - where,
    - w_{hh} -> weight at recurrent neuron
    - w_{xh} -> weight at input neuron
- **The formula for calculating output:**
  - $y_t=W_{hy}h_t$
  - Y_t -> output
  - W_{hy} -> weight at output layer
- These parameters are updated using Backpropagation.
- However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

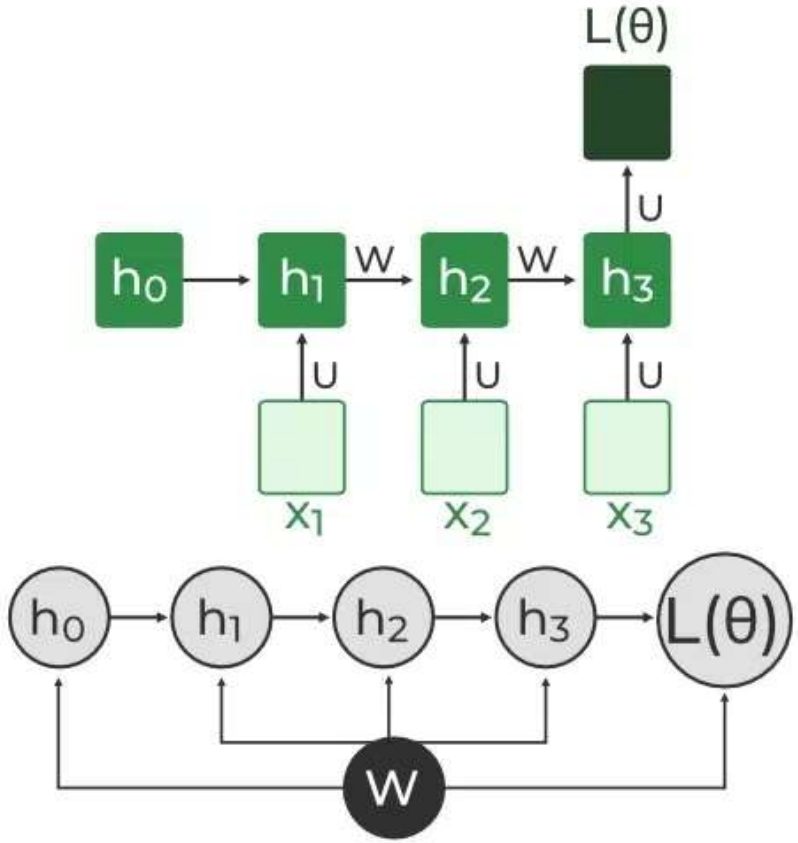
Learning Process:

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.

$$\partial L(\theta)\partial W=\sum t=1T\partial L(\theta)\partial W$$

Challenges with RNNs

Vanishing Gradients:

When the gradient becomes too small during backpropagation, weight updates diminish, making it difficult for the model to learn long-term dependencies.

This happens because the model forgets older information as

Exploding Gradients:

If the gradient becomes too large, it causes instability in the network, making the weights grow excessively, leading to NaN (Not a Number) values.

Both vanishing and exploding gradients make training difficult.

Solutions:

Reducing the number of hidden layers can help mitigate exploding/vanishing gradients.

Gradient clipping is used to prevent exploding gradients by capping the gradients during backpropagation.

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.

Applications of RNNs:

Speech Recognition: Converts spoken words into text by understanding temporal sequences in audio data.

Language Translation: Translates sentences by processing word sequences and their dependencies.

Stock Market Prediction: Analyzes historical stock prices to predict future trends.

Image Captioning: Generates descriptive text for images by interpreting sequential features within the visual data.

Advantages & Disadvantages

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

Long Short-Term Memory

LSTM excels in sequence prediction tasks, capturing long-term dependencies.

Long Short-Term Memory is an improved version of recurrent neural network

A traditional_RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies.

LSTMs model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.

LSTM architectures are capable of learning long-term dependencies in sequential data: well-suited for tasks such as language translation, speech recognition, and time series forecasting.

LSTM Architecture

Memory Cell:

The **memory cell** is the key feature of LSTM that allows it to retain information over long time steps. The cell stores information, and its contents are modified by the following gates:

Gates:

Input Gate: Determines which parts of the current input should be added to the memory cell.

Forget Gate: Decides which part of the information in the memory cell should be removed. It helps the LSTM forget irrelevant information.

Output Gate: Determines what information from the memory cell should be output and passed to the next layer or time step.

Hidden State:

The **hidden state** represents the short-term memory of the LSTM. It gets updated at each time step based on the current input, the previous hidden state, and the memory cell's current state.

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.

LSTM Architecture

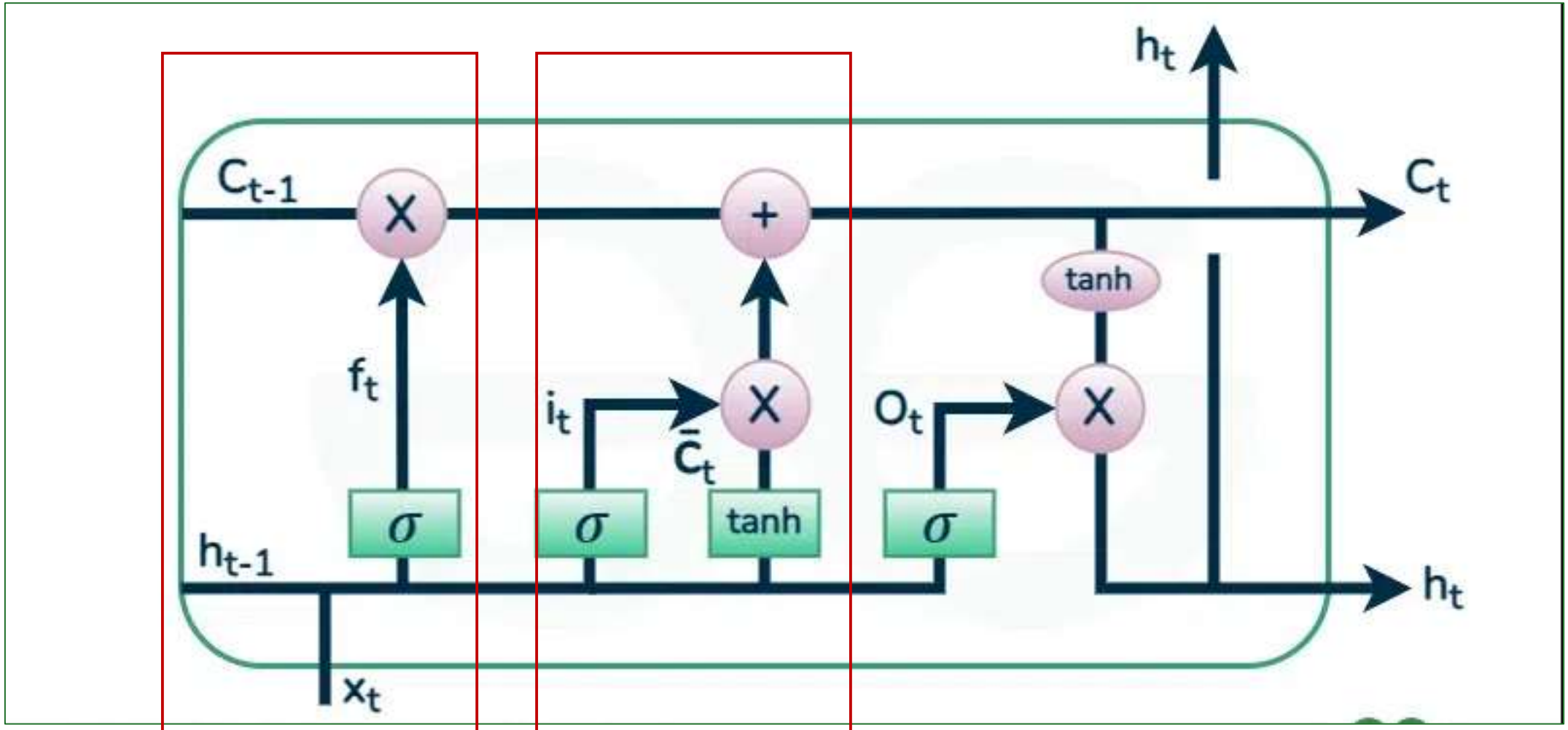
Stacked LSTM:

- **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.

LSTM Working



LSTM Working:Forget Gate:

- **Function**: The forget gate controls which information in the cell state should be discarded. It helps the LSTM "forget" irrelevant data.
- **Input**:
  - x_t: Current input at time step t.
  - h_{t-1}: Hidden state from the previous time step.
- **Computation**:

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

- W_f: Weight matrix for the forget gate.
- b_f: Bias term for the forget gate.
- σ : Sigmoid activation function.
- The forget gate output, f_t, is a value between 0 and 1, where:
  - 0: Forget the information.
  - 1: Retain the information.

LSTM Working: Input Gate:

- **Function:** The input gate decides which information from the input should be added to the cell state.
- **Input:** Similar to the forget gate, it uses the current input x_t and the previous hidden state h_{t-1}.
- **Computation:**
  - Regulate the input using the sigmoid function:

    $i_t=\sigma(W_i\cdot[h_t-1,x_t]+b_i)$
  - Create candidate values for updating the cell state:

    $\widehat{C}_t=\tanh(W_c\cdot[h_{t-1},x_t]+b_c)$
  - Update the cell state: $C_t=f_t\odot C_{t-1}+i_t\odot \widehat{C}_t$
  - $\odot$: Element-wise multiplication.
  - This equation decides what part of the previous cell state to keep and how much of the new candidate values to incorporate.

LSTM Working: Output Gate:

- **Function:** The output gate determines what part of the cell state should be output as the hidden state for the current time step.
- **Input:** Similar to the other gates, it uses h_{t-1} and x_t.
- **Computation:**
  - $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
    - The current hidden state (or output) is computed as: $h_t = o_t \odot \tanh(C_t)$
    - This equation extracts information from the updated cell state and decides what information will be carried forward as the hidden state.

Difference between RNN and LSTM

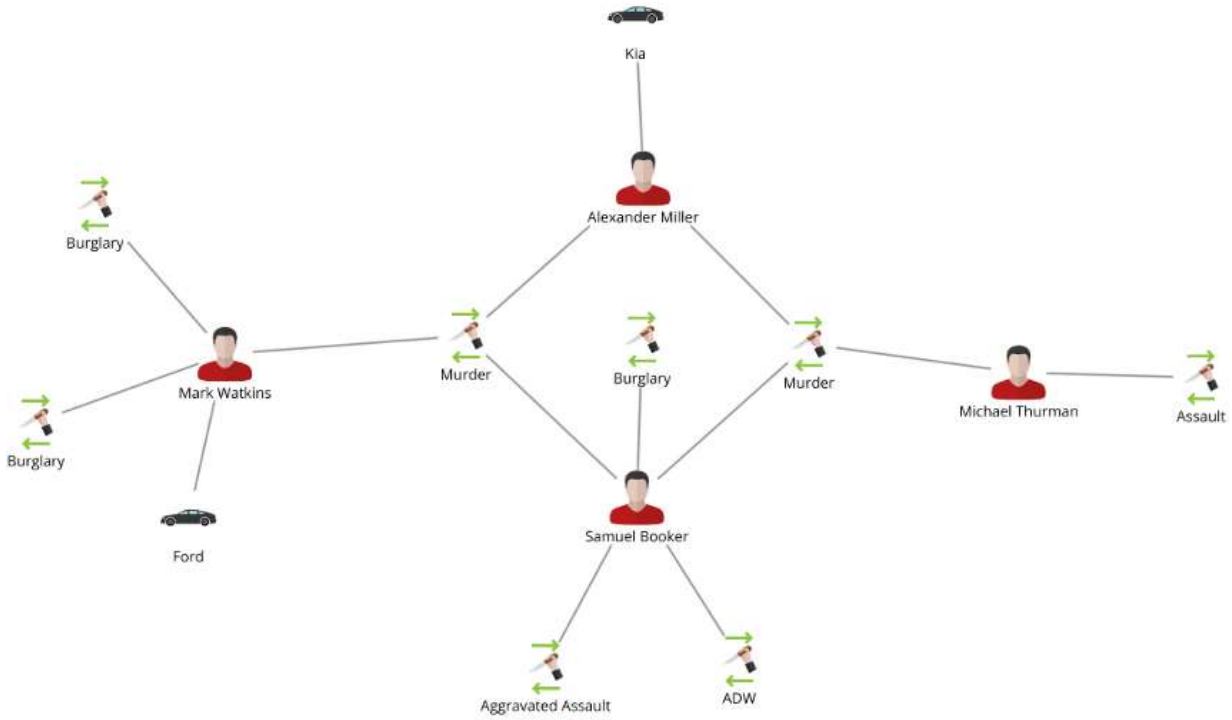
Feature	RNN (Recurrent Neural Network)	LSTM (Long Short-Term Memory)
Architecture	Simple network with a single hidden state	Complex network with memory cells and three gates (input, forget, output)
Handling Long-term Dependencies	Struggles with long-term dependencies due to vanishing gradient problem	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

Difference between RNN and LSTM

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

Graph analytics for social network analysis

GRAPH ANALYSIS



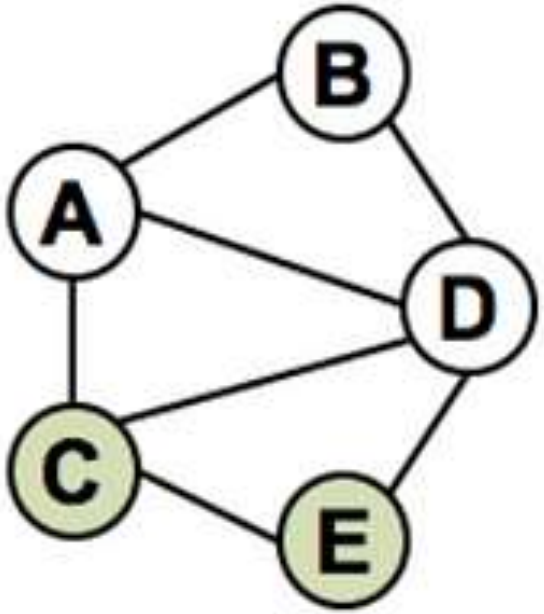
- Graph analysis is a data analysis technique that studies the relationships between objects in a graph, or network, to understand how they are connected.
- It can be used to discover hidden information, optimize processes, and deliver key information to decision makers.

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

Network Theory

Nodes



Edges

Network Theory

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.)

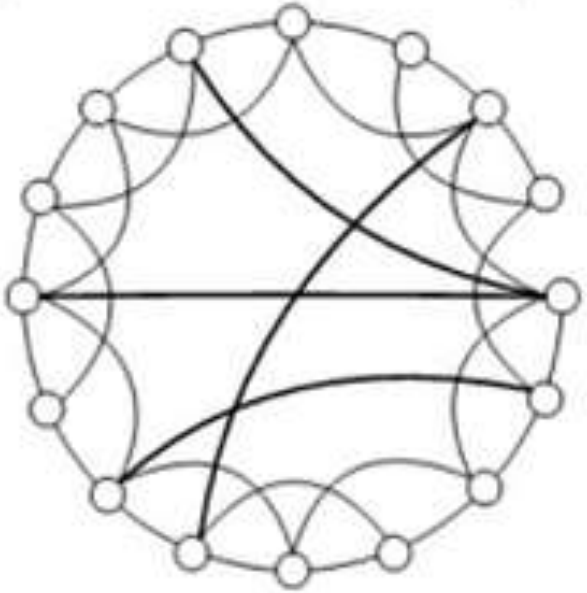
Edges

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

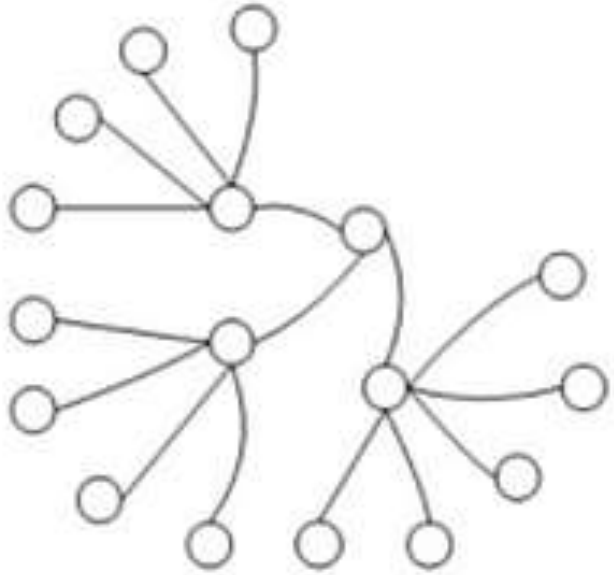
These two basic elements can describe multiple phenomena, such as *social connections, virtual routing network, physical electricity networks, roads network, biology relations network* and many other relationships.

Real-world networks

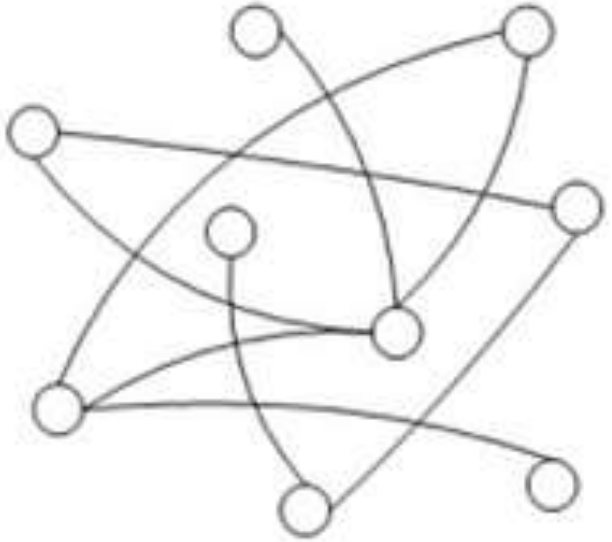
(a) Small-World Network (SWN)



(b) Scale-Free Network (SFN)



(c) Random Network (RN)



Real-world networks

Small World

phenomenon claims that real networks often have very short paths (in terms of number of hops) between any connected network members.

This applies for real and virtual social networks (the six handshakes theory) and for physical networks such as airports or electricity of web-traffic routings.

Scale Free networks

with power-law degree distribution have a skewed population with a few highly-connected nodes (such as social-influences) and a lot of loosely-connected nodes.

Homophily

is the tendency of individuals to associate and bond with similar others, which results in similar properties among neighbors.

Centrality Measures

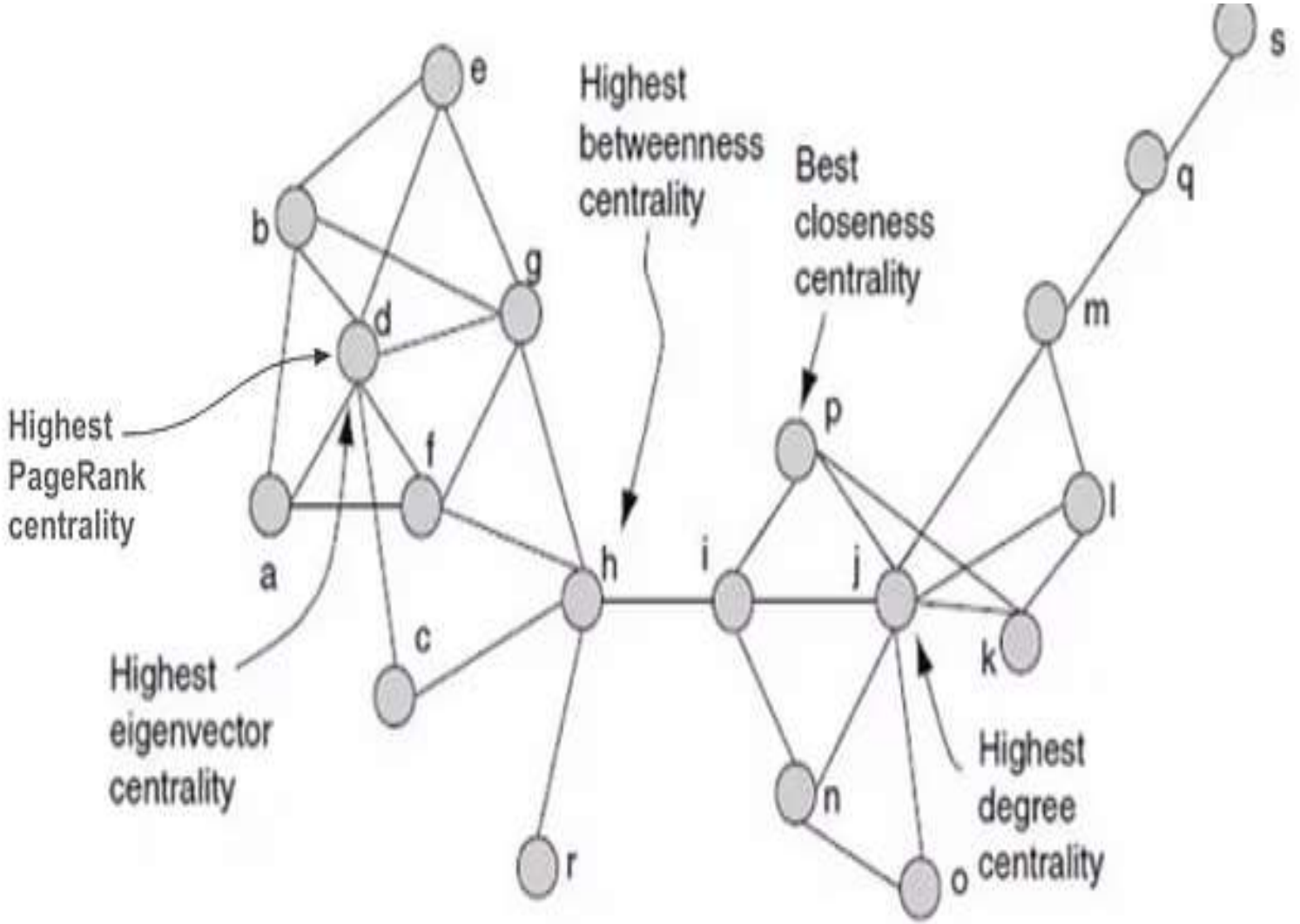
Highly central nodes play a key role of a network, serving as hubs for different network dynamics.

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

ity Measures

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



Building a Network

Building a Network

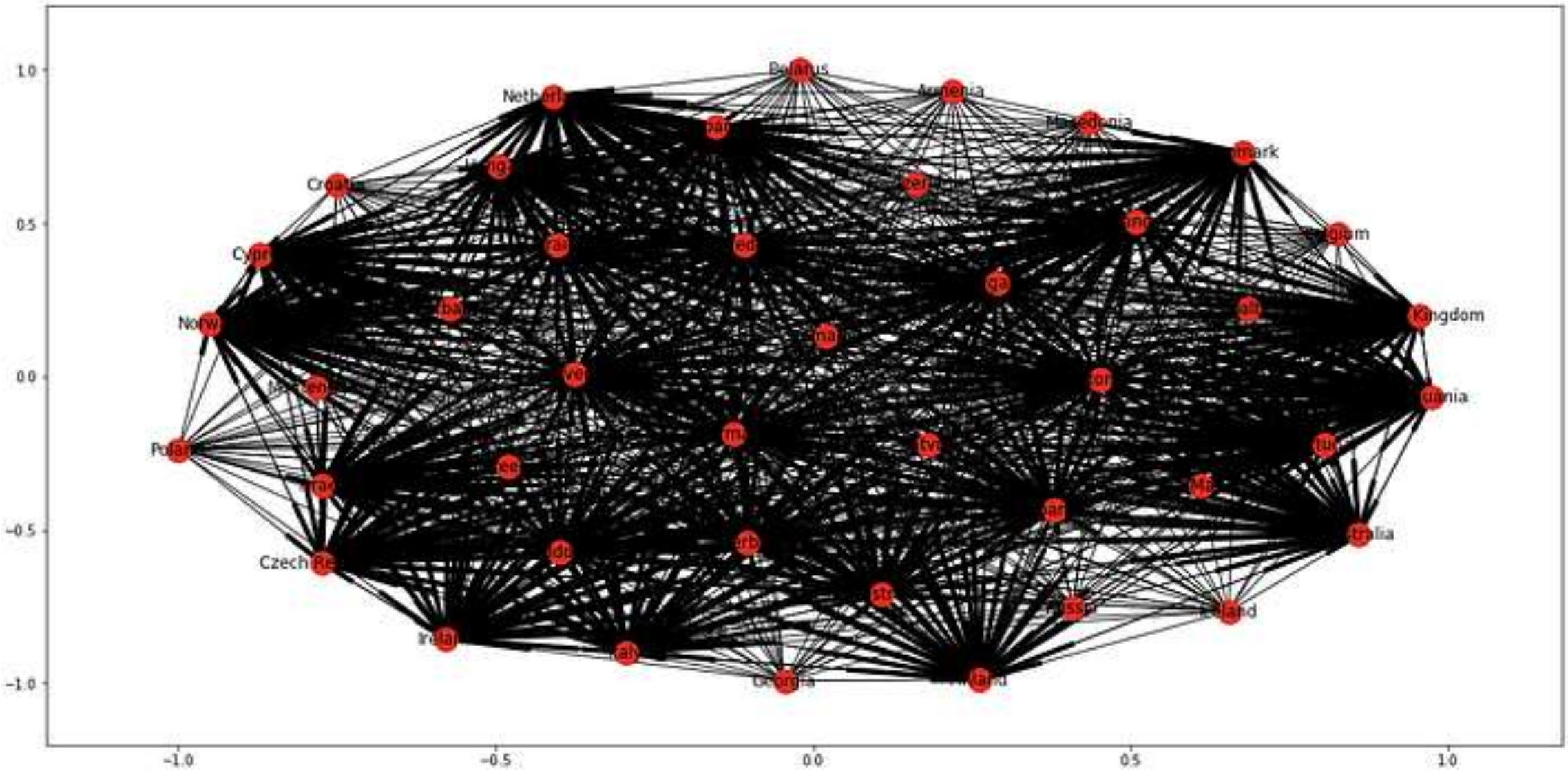
- Networks can be constructed from various datasets, as long as we're able to describe the relations between nodes.
- In the following example we'll build and visualize the Eurovision 2018 votes network (based on official data) with **Python *networkx*** package.
  - We'll **read the data** from excel 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.
  - Finally, we'll try the generic method to **visualize**, as shown in the code below:

```
votes_data = pd.read_excel('ESC2018_GF.xlsx',sheetname='Combined result')


votes_melted = votes_data.melt(
['Rank','Running order','Country','Total'],
var_name = 'Source Country',value_name='points')


G = nx.from_pandas_edgelist(votes_melted,
source='Source Country',
target='Country',
edge_attr='points',
create_using=nx.DiGraph())


nx.draw_networkx(G)
```


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
Kingdom

Visualization

- Unfortunately the built-in ***draw*** method results in a very incomprehensible figure.
- The method tries to plot a highly connected graph, but with no useful “hints” it’s unable to make a lot of sense from the data.
- We will enhance the figure by **dividing and conquering** different visual aspects of the plot with a prior knowledge that we have about the entities:
  - **Position —** each country is assigned according to its geo-position
  - **Style** — each country is recognized by its flag and flag colors
  - **Size** — the size of nodes and edges represents the amount of points

Eurovision 2018 Final Votes