

Module-5

Deep Learning (BCA-701)

Module-5

Sequence Modeling:

1. Recurrent and Recursive Nets

Sequence modeling deals with data where **order matters**. Examples include text, speech, time-series, and signals. Unlike feedforward networks, **Recurrent and Recursive Neural Networks** are designed to model **dependencies across sequences and structures**.

Why Sequence Modeling is Needed

- Output depends on **previous inputs**
- Data has **temporal or structural relationships**
- Past information influences present decisions

Example:

- Sentence meaning depends on word order
- Stock price depends on previous prices

1. Recurrent Neural Networks (RNNs)

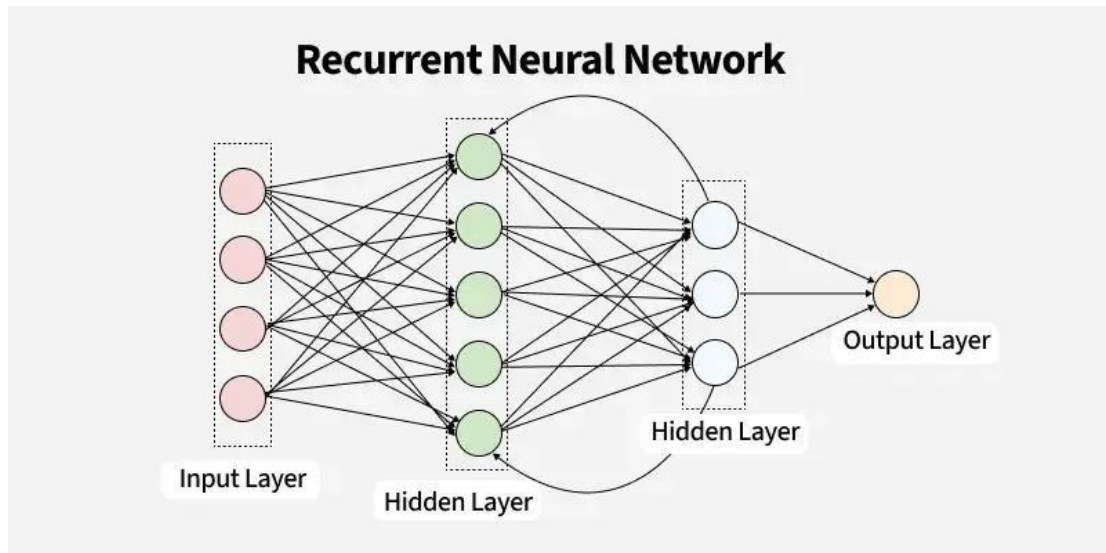
What is an RNN?

A **Recurrent Neural Network** is a neural network that has **feedback connections**, allowing information to **persist over time**.

RNNs have a **memory** of past inputs.

Basic Idea

- Same network is applied at each time step
- Output at time t depends on:
 - Current input x_t
 - Previous hidden state h_{t-1}



Mathematical Representation

$$h_t = f(Wx_t + Uh_{t-1} + b)$$
$$y_t = g(Vh_t)$$

Where:

- h_t = hidden state (memory)
- x_t = input at time t

Characteristics of RNN

- Handles sequential data
- Uses shared weights across time
- Can model temporal dependencies

Limitations of RNN

- Vanishing gradient problem
- Difficulty learning long-term dependencies

Applications of RNN

- Speech recognition
- Language modeling
- Time-series prediction

2. Recursive Neural Networks (Recursive Nets)

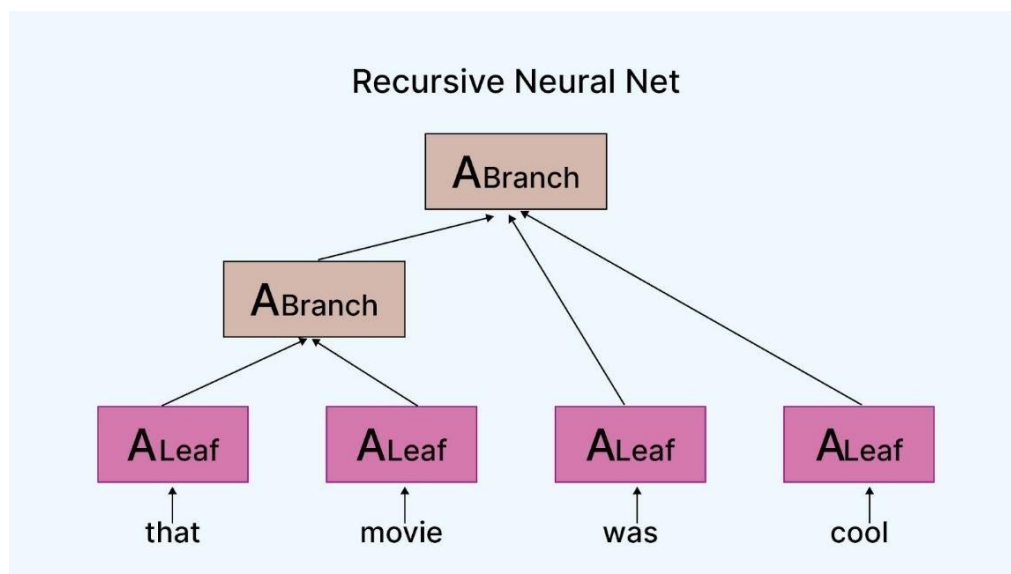
What is a Recursive Neural Network?

A **Recursive Neural Network** processes data with a hierarchical or tree structure, instead of time sequences.

Structure matters more than time.

Key Idea

- Same network is applied **recursively** over structured inputs
- Used where data forms a **tree**, not a sequence



Characteristics of Recursive Nets

- Operate on **tree-like structures**
- Combine child nodes to form parent representations
- Used in structured data processing

Applications of Recursive Nets

- Natural language parsing
- Syntax tree analysis
- Scene understanding

Recurrent vs Recursive Neural Networks

Feature	Recurrent NN	Recursive NN
Data type	Sequential	Structured (tree)
Dependency	Time-based	Hierarchical
Memory	Yes	Implicit via structure
Example	Speech, text	Parse trees

Key Points

- RNNs model time-dependent sequences
- Recursive nets model hierarchical structures
- Both reuse the same weights
- Useful for NLP and sequence tasks

2. Unfolding Computational Graphs

Unfolding computational graphs is a technique used mainly with **Recurrent Neural Networks (RNNs)** to represent how the network operates **over time**. It converts a cyclic (looped) network into a **sequence of feedforward networks**, making analysis and training easier.

Why Unfolding is Needed

RNNs contain **feedback loops**, which makes:

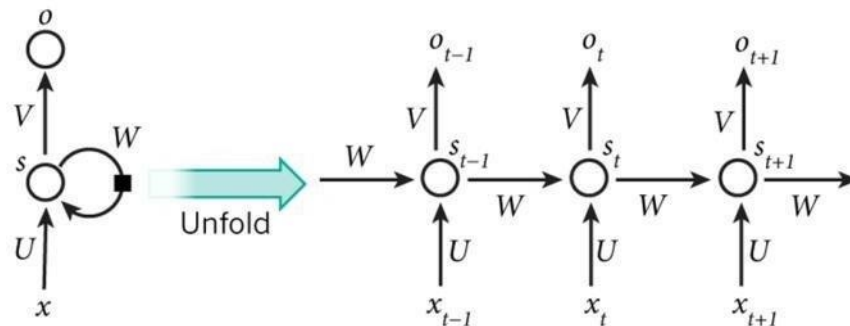
- Understanding signal flow difficult
- Gradient computation complex

By **unfolding**, we:

- Remove cycles
- Visualize computation at each time step
- Apply **Back-Propagation Through Time (BPTT)**

Basic Idea of Unfolding

- An RNN is copied **once for each time step**
- Each copy shares the **same weights**
- Output at time t depends on:
 - Input at time t
 - Hidden state from time $t - 1$



Mathematical View

For time step t :

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

$$y_t = g(Vh_t)$$

Unfolding shows this computation **repeated across time steps** $t = 1, 2, 3, \dots, T$.

Unfolded Graph Representation

- Each time step is a **layer**
- Hidden states are connected sequentially
- Weight matrices W , U , V remain **same across all steps**

This transforms the RNN into a **deep feedforward network** across time.

Training Using Unfolded Graph

- Error is computed at each time step
- Gradients flow **backward through time**
- This process is called **Back-Propagation Through Time (BPTT)**

Advantages of Unfolding

- Simplifies gradient computation
- Makes RNN training possible
- Helps visualize temporal dependencies

Limitations

- Long sequences → very deep graphs
- Leads to **vanishing or exploding gradients**
- Computationally expensive for long time steps

Key Points

- Unfolding converts cyclic RNN into acyclic graph
- Same weights reused at every time step
- Enables Back-Propagation Through Time
- Depth depends on sequence length

3. Recurrent Neural Networks (RNN's)

A **Recurrent Neural Network (RNN)** is a special type of neural network designed to process **sequential data**, where the **current output depends on previous inputs**. Unlike feedforward networks, RNNs have **feedback connections**, giving them a form of **memory**.

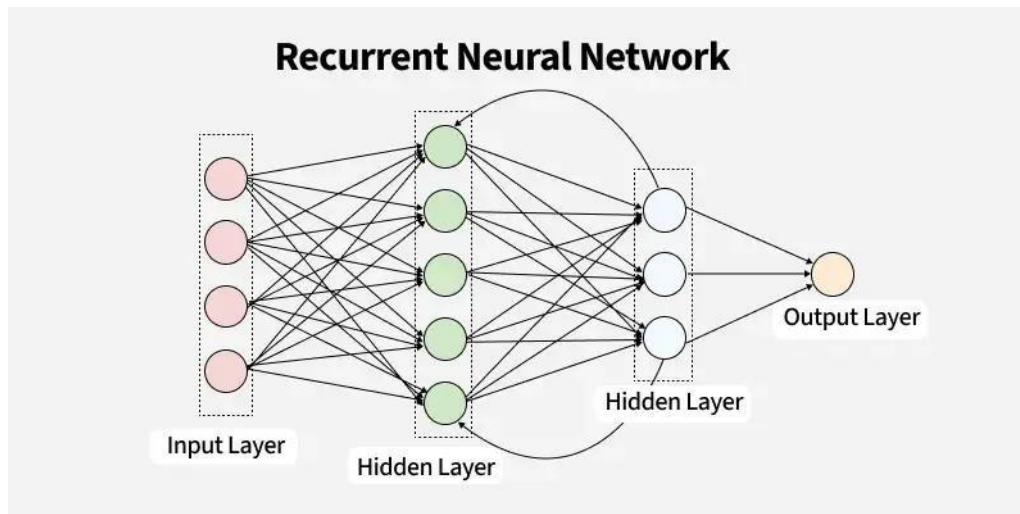
What is a Recurrent Neural Network?

A **Recurrent Neural Network** is a neural network in which the output of a neuron is fed back into the network as input for the next step, allowing the network to **remember past information**.

Key Idea of RNN

- Same network is reused at every time step

- Information flows **forward in time**
- Past information influences present output



Basic Architecture of RNN

An RNN consists of:

- **Input layer**
- **Hidden layer (with memory)**
- **Output layer**

The hidden layer stores the **state** of the network.

Mathematical Representation

At time step t :

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

$$y_t = g(Vh_t)$$

Where:

- x_t = input at time t
- h_t = hidden state (memory)
- h_{t-1} = previous hidden state
- W, U, V = weight matrices

Working of RNN (Step-by-Step)

1. Input sequence is given one element at a time
2. Hidden state updates using current input and past state
3. Output is generated at each time step (or at the end)
4. Same weights are shared across all time steps

Types of RNN Output Structures

- **One-to-One** (standard NN)
- **One-to-Many** (image → caption)
- **Many-to-One** (sentiment analysis)
- **Many-to-Many** (machine translation)

Training of RNN

- Trained using **Back-Propagation Through Time (BPTT)**
- Error is propagated backward across time steps

Problems with RNN

- **Vanishing gradient problem**
 - **Exploding gradient problem**
 - Difficulty learning long-term dependencies
- These issues led to advanced models like **LSTM** and **GRU**.

Applications of RNN

- Speech recognition
- Language modeling
- Text generation
- Time-series prediction

4. Bidirectional RNNs

A **Bidirectional Recurrent Neural Network (BiRNN)** is an extension of the standard RNN that processes a sequence in **both forward and backward directions**. This allows the network to use **past and future context** for making predictions.

Why Bidirectional RNNs are Needed

In many tasks, understanding a data point requires:

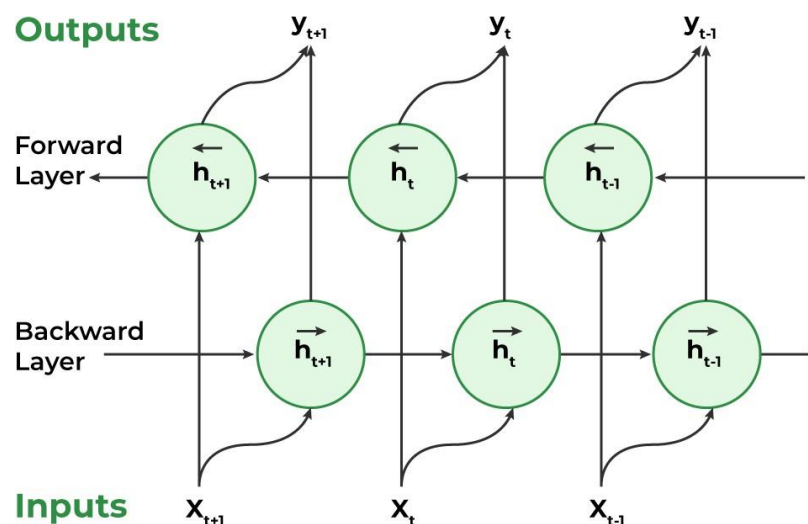
- Information from **previous inputs**
- Information from **future inputs**

Example:

In a sentence, the meaning of a word often depends on **both earlier and later words**.

Basic Idea

- Two separate RNNs are used:
 - **Forward RNN** → processes input from $t = 1$ to T
 - **Backward RNN** → processes input from $t = T$ to 1
- Outputs from both RNNs are **combined**



Architecture of BiRNN

- Input layer feeds into two hidden layers
- One hidden layer processes forward

- Another hidden layer processes backward
- Output layer combines both hidden states

Mathematical Representation

$$\begin{aligned}
 \vec{h}_t &= f(W_f x_t + U_f \vec{h}_{t-1}) \\
 \overleftarrow{h}_t &= f(W_b x_t + U_b \overleftarrow{h}_{t+1}) \\
 y_t &= g(\vec{h}_t, \overleftarrow{h}_t)
 \end{aligned}$$

Advantages of Bidirectional RNNs

- Uses full context of sequence
- Improved accuracy for sequence labeling
- Better understanding of dependencies

Limitations

- Cannot be used in real-time applications
- Requires complete sequence before processing
- Higher computational cost

Applications

- Speech recognition
- Handwriting recognition
- Named Entity Recognition (NER)
- Part-of-speech tagging

Key Points

- BiRNNs process data in two directions
- Combines past and future context
- Not suitable for online learning
- Often combined with LSTM/GRU

5. Encoder-Decoder Sequence-to- Sequence Architectures

The **Encoder-Decoder sequence-to-sequence (Seq2Seq) architecture** is a neural network framework designed to map an input sequence to an output sequence, where the **lengths of the sequences may be different**. It is widely used in tasks like **machine translation, summarization, and speech recognition**.

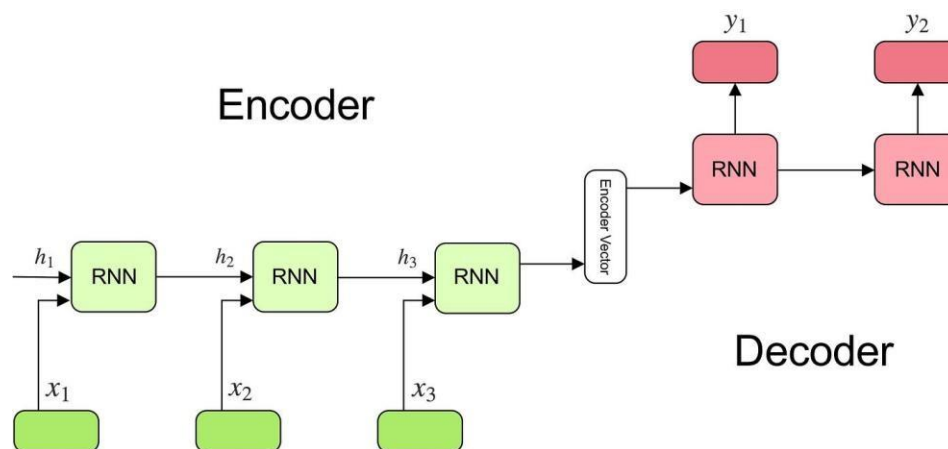
Why Encoder-Decoder Architecture is Needed

- Input and output sequences may have **different lengths**
- Output depends on the **entire input sequence**
- Traditional RNNs struggle with long sequences

Basic Structure

The architecture consists of **two main components**:

1. **Encoder**
2. **Decoder**



1. Encoder

- Reads the **input sequence** one element at a time
- Converts input into a **fixed-length context vector**
- Final hidden state summarizes the entire input sequence

$$h_t = f(Wx_t + Uh_{t-1})$$

2. Decoder

- Takes the **context vector** from the encoder
- Generates the **output sequence step by step**
- Each output depends on previous outputs and context

$$y_t = g(Vh_t)$$

Working of Encoder-Decoder Model

1. Encoder processes input sequence
2. Final encoder state becomes **context vector**
3. Decoder starts generating output
4. Process continues until end-of-sequence token

Variants Used

- RNN-based Encoder–Decoder
 - LSTM-based Encoder–Decoder
 - GRU-based Encoder–Decoder
- LSTM/GRU are preferred to handle long-term dependencies.

Advantages

- Handles variable-length sequences
- Flexible and powerful
- Works well with sequence tasks

Limitations

- Fixed-length context vector bottleneck
 - Performance drops for very long sequences
- This led to the development of **Attention Mechanisms**.

Applications

- Machine translation
- Text summarization
- Chatbots
- Speech-to-text systems

Key Points

- Consists of encoder and decoder
- Encoder compresses input into context vector
- Decoder generates output sequence
- Used in Seq2Seq problems

6. Deep Recurrent Networks

Deep Recurrent Networks (Deep RNNs) are an extension of standard RNNs in which multiple recurrent layers are stacked on top of each other. This depth allows the network to learn hierarchical temporal representations—simple patterns in lower layers and more abstract patterns in higher layers.

What is a Deep Recurrent Network?

A **Deep RNN** is a recurrent neural network with **more than one hidden recurrent layer**, where the output (hidden state) of one RNN layer becomes the input to the next RNN layer **at the same time step**.

Why Use Deep RNNs?

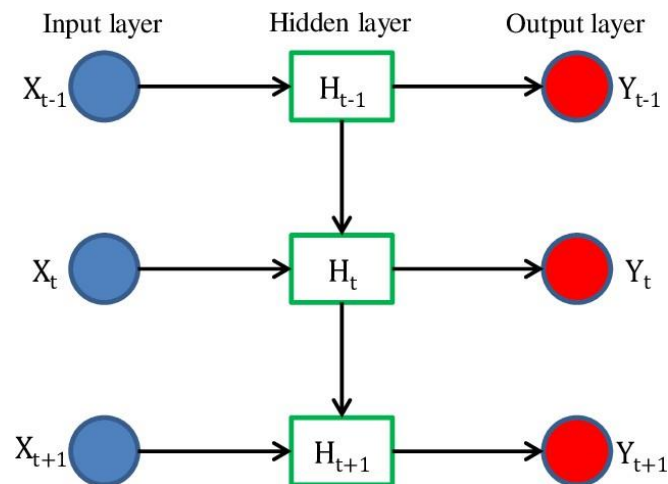
- Capture **complex temporal dependencies**
- Learn **multi-level features** from sequences
- Improve performance on long and structured sequences

Examples:

- Speech recognition (phonemes → words → sentences)
- Language modeling (characters → words → meaning)

Architecture of Deep RNN

- Input sequence flows **vertically across layers** and **horizontally across time**
- Each layer has its own recurrent connections
- Weights are shared **across time**, not across layers



Mathematical View

For layer l at time t :

$$h_t^{(l)} = f(W^{(l)}h_t^{(l-1)} + U^{(l)}h_{t-1}^{(l)} + b^{(l)})$$

Where:

- $h_t^{(l)}$ = hidden state of layer l
- $h_t^{(l-1)}$ = output of previous layer
- $h_{t-1}^{(l)}$ = previous time-step state of same layer

Training Deep RNNs

- Trained using **Back-Propagation Through Time (BPTT)**
- Gradients propagate:
 - **Backward in time**
 - **Backward across layers**

This makes training **computationally expensive**.

Problems in Deep RNNs

- **Vanishing gradients** (worse with depth and time)
 - **Exploding gradients**
 - Long training time
- These are addressed using **LSTM, GRU, residual connections, and normalization.**

Deep RNN vs Shallow RNN

Feature	Shallow RNN	Deep RNN
Hidden layers	One	Multiple
Representation	Simple	Hierarchical
Accuracy	Moderate	Higher
Complexity	Low	High

Applications

- Speech recognition
- Machine translation
- Video sequence analysis
- Handwriting recognition

7. Recursive Neural Networks

A Recursive Neural Network (RvNN) is a type of neural network designed to process structured data, especially tree-like or hierarchical structures, rather than time-based sequences.

Unlike Recurrent Neural Networks (RNNs), which work over time steps, Recursive Neural Networks work over structures.

What is a Recursive Neural Network?

A **Recursive Neural Network** applies the **same set of weights recursively** to combine smaller parts of a structure into larger representations.

- The network is applied repeatedly following the **structure of the data (tree)**.

Why Recursive Neural Networks are Needed

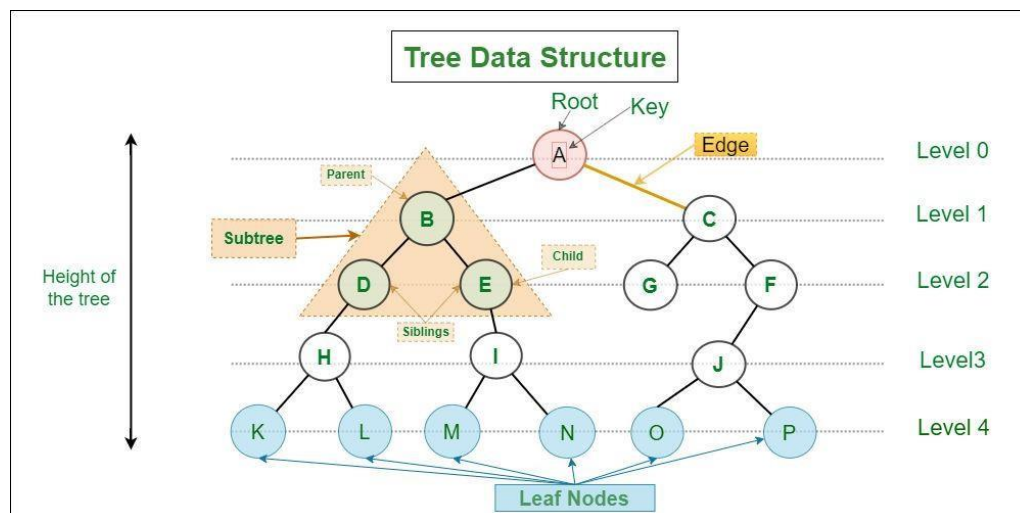
Some data is **not sequential**, but **hierarchical**, such as:

- Parse trees in sentences
- Abstract syntax trees in programs
- Scene hierarchies in images

RNNs are not suitable for such data → **Recursive Nets** are used.

Basic Idea

- Inputs are arranged in a **tree structure**
- Child nodes are combined to form a **parent node**
- Same neural network is reused at every node of the tree



Working of Recursive Neural Network

1. Leaf nodes receive input vectors
2. Parent node computes its representation from child nodes

3. Process continues **bottom-up**
4. Root node represents the entire structure

Mathematical Representation

For a parent node with two children:

$$h_p = f(W[h_{c1}, h_{c2}] + b)$$

Where:

- h_{c1}, h_{c2} = child node representations
- h_p = parent representation
- $f(\cdot)$ = activation function

Training of Recursive Neural Networks

- Trained using **Back-Propagation Through Structure (BPTS)**
- Similar to BPTT, but applied over **tree structures**

Characteristics

- Operates on **trees, not sequences**
- Same weights shared across all nodes
- Captures hierarchical relationships

Recursive NN vs Recurrent NN

Feature	Recursive NN	Recurrent NN
Data type	Tree / Hierarchical	Sequential
Dependency	Structural	Temporal
Flow	Bottom-up	Time-based
Example	Parse trees	Speech, text

Applications

- Natural language parsing
- Sentiment analysis using syntax trees
- Program structure analysis
- Scene understanding

Key Points

- Recursive nets process structured data
- Use shared weights across tree nodes
- Trained using back-propagation through structure
- Suitable for hierarchical representations

8. The Long Short-Term Memory and Other Gated RNNs

Standard Recurrent Neural Networks (RNNs) suffer from the vanishing and exploding gradient problems, making it difficult to learn long-term dependencies. To overcome this, gated RNN architectures such as LSTM and GRU were introduced.

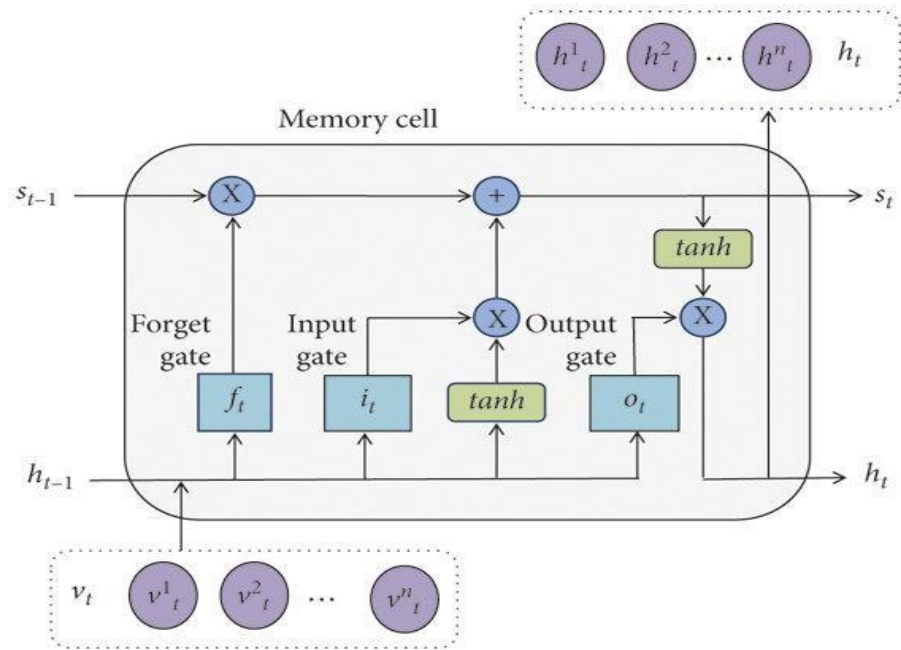
1. Long Short-Term Memory (LSTM)

What is LSTM?

Long Short-Term Memory (LSTM) is a special type of RNN designed to remember information for long periods of time using a memory cell and gates that control information flow.

Key Idea

LSTM replaces the simple hidden state of an RNN with a **cell state**, which acts like a **memory highway**, allowing information to flow with minimal modification.



Components of an LSTM Cell

An LSTM has **three gates** and **one cell state**:

1. Forget Gate

Decides what information to **remove** from the cell state.

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

2. Input Gate

Decides what **new information** to store.

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

3. Cell State Update

Updates the memory:

$$C_t = f_t C_{t-1} + i_t C_t$$

4. Output Gate

Decides what to **output**.

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

Why LSTM Works Better

- Controls gradient flow
- Preserves long-term information
- Avoids vanishing gradient problem

Applications of LSTM

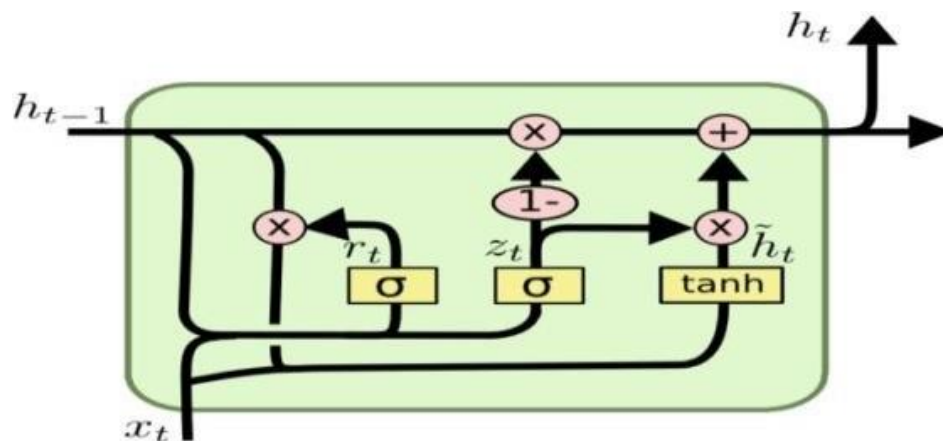
- Machine translation
- Speech recognition
- Text generation
- Time-series forecasting

2. Other Gated RNNs

Gated Recurrent Unit (GRU)

What is GRU?

A **GRU** is a simplified gated RNN that combines memory and hidden state into one and uses **fewer gates** than LSTM.



Gates in GRU

1. **Update Gate** – controls how much past information to keep
2. **Reset Gate** – controls how much past information to forget

Why GRU is Popular

- Simpler than LSTM
- Faster training
- Comparable performance

LSTM vs GRU

Feature	LSTM	GRU
Gates	3	2
Cell state	Separate	Combined
Complexity	High	Lower
Training speed	Slower	Faster
Performance	Very strong	Comparable

Advantages of Gated RNNs

- Handle long-term dependencies
- Reduce vanishing gradients
- More stable training

Limitations

- Computationally expensive
- Slower than simple RNNs
- Require large datasets

Key Points

- LSTM uses memory cells and gates
- GRU is a simplified gated RNN
- Gated RNNs solve long-term dependency issues
- Widely used in sequence modelling