# DEEP LEARNING

## Unit III Recurrent Neural Networks

May-Dec 2022

Q.1 a) Differentiate between feed forward neural networks and recurrent neural networks Explain the types of Recurrent Neural Network (RNN).

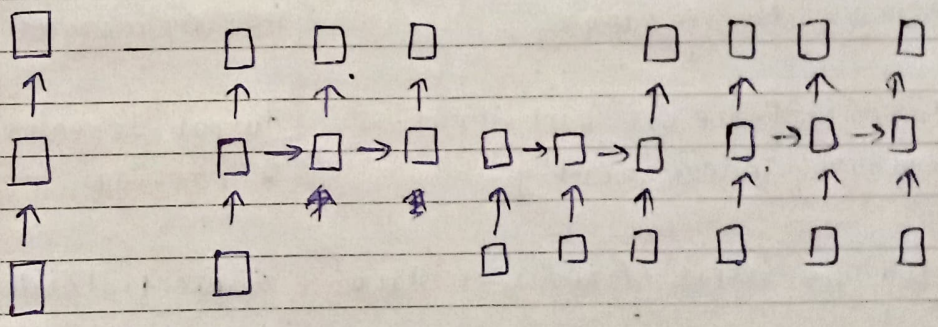
Feature | Feed forward Neural Network (FFNN) | Recurrent Neural Network (RNN)

| Feature | Feed forward Neural Network (FFNN) | Recurrent Neural Network (RNN) |
|---|---|---|
| Information Flow | One direction: Input → Hidden → Output | Cyclic: Output/Hidden fed back into network |
| Connections | No recurrent/feedback connections | Has recurrent loops |
| Memory | No memory | Has memory (hidden state) |
| Input Dependency | Processes each input independently | Output depends on current + previous inputs |
| Data Type | Static/fixed-size data | Sequential/time-dependent data |
| Sequence Handling | Cannot handle sequences | Designed for sequences |
| I/P-O/P Mapping | Mostly One-to-one | One-to-one, one-to-many, many-to-one, many-to-many |
| Context Understanding | No context | Learns context and temporal relationships |
| Training Complexity | Simple and fast | Slow, complex |

| | Feedforward Neural Network (FFNN) | Recurrent Neural Network (RNN) |
|---|---|---|
| Vanishing Gradient | less likely | Highly prone |
| Suitable Tasks | Imagle classification, tabular data | NLP, speech, time-series, video |
| Applications | Object detection, pattern recognition | Machine translation, sentiment analysis, speech recognition |

## Types of RNN

one-to-one one-to-many many-to-one many-to-many



1. One-to-One (Vanilla Neural Network)

In this type a single input is taken and it gives one single output. It is basically like a traditional feedforward neural network in which there is no sequence processing.

- No time dependency
- No memory required

* Example: Image Classification
  Input: one image
  output: one label
  e.g. Input: picture of a dog
  Output: "Dog"

~~$a^3 + 3a^2b + 3ab^2 + b^3$~~ ~~$(a+b)^3$~~

~~$1 + 3w + 3w^2 + w^3$~~

In image classification task network has to see current image. There is no role of previous images, therefore there is no need for a memory sequence. This network extracts features of image and predicts a category like cat, dog, cat, etc.

2 One-to-many

In this architecture a single input is taken but multiple outputs are generated over time. Network interprets one input and produces sequence output on its basis.

e.g 1 Image → Caption Generation

Input: one image

Output: sentence (multiple words)

e.g

Input image: dogplayinginpark

Output: "A dog is playing in the park"

Meaning of image is extracted by RNN and step-by-step words are generated. First "A", then "dog", then "is", etc. Here there is output sequence, so RNN r suitable.

e.g.2 Music Generation

RNN uses an inpute melody to generate future notes, which creates the complete tune.

- One-to-many RNNs are used when a singl concept needs to be expanded into a sequence

3 Many-to-One

In this type RNN processes multiple inputs in the form of sequence and gives a singl output at the end.

e.g. 1 sentiment Analysis.

Input: full sentence

Output: sentiment label

e.g.

sentence: "The movie was amazing"
Output: "positive"
RNN reads the words of the whole sentence sequentially, updates memory at every step and builds context. In the end network decides what is the sentences emotional tone.
e.g. 2 Spam Detection
Input: email text
Output: spam / not spam
Analyzes multiple words from email and identifies patterns.
- "free money"
- "click here"
- "win prize"
A final output is generated based on entire text.
Many-to-one models are used when the final decision depends on the entire sequence.

4. Many-to-many
This architecture has both multiple inputs and multiple outputs. Network processes input sequence and produces corresponding output sequence
e.g. 1: Machine Translation
Input: English sentence
Output: Hindi sentence
Input: "I love cats"
Output: "Mujhe billiyan pasand hain"
RNN first learns the meaning of the entire sentence, then outputs translated words step-by-step. Input and output both are sequences.
e.g. 2: Speech recognition
Input: speech audio frames
Output: text sequence

Speech signals continuously change, RNN processes them to generate words like "Hello how are you"

Thus, RNN architectures can be categorized based on the relationship between input and output sequences. They include one-to-one, one-to-many, many-to-one and many-to-many structures, which make RNNs suitable for tasks like image captioning, translation, speech recognition, and sentiment analysis.

Advanced Types

Bidirectional RNN (Bi-RNN)

- Reads sequence forward & backward
- Better context understanding

e.g. "bank" meaning depends on next words

Deep/Stacked RNN

- Multiple RNN layers stacked
- Better feature extraction

Q.2b) Explain how sequence to sequence model works

Ans. Sequence-to-Sequence model is a deep learning architecture that converts input sequence to output sequence, whether both sequences have the same or different lengths. Traditional models used to handle fixed lengths inputs or outputs, which made tasks like translation, summarization, captioning difficult. seq2seq model became important because input and output length change in the real world, and it is important to preserve the meaning. This model captures the overall meaning of the input and generates relevant output sequences. Seq2Seq model maps an input sequence to an output sequence using an Encoder-Decoder architecture, where the encoder produces a

context vector and the decoder generates the output step by step.

## Importance of Sequence to Sequence Models

- Real-world tasks like machine translation require variable-length outputs, which traditional RNNs cannot handle. The seq2seq model compresses the input ~~sequence~~ sentence and generates an output ~~sequen~~ sentence even if the lengths differ.
- Many tasks require understanding the entire context, not just word-by-word mapping. Seq2seq encodes the full sentence meaning and generates semantically accurate output.
- It is useful in many domains where input and output sequences are not aligned, such as speech-to-text, text-to-speech, summarization, captioning, and dialogue systems.

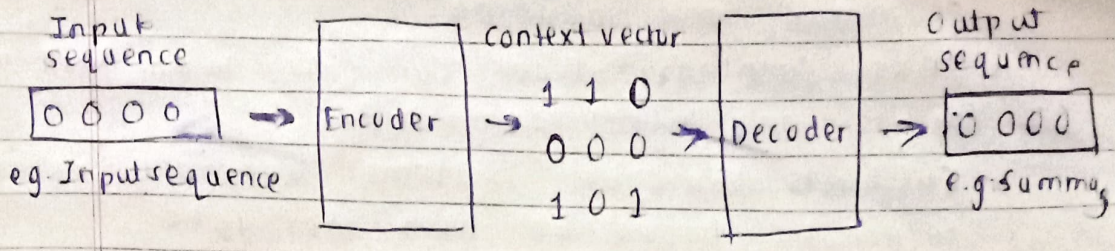
## Architecture (Encoder - Decoder)

1. Encoder: The encoder processes each token of the input sequence and builds a hidden representation. The final hidden state is stored as a context vector, which holds the complete meaning of the input.
2. Decoder: The decoder uses the context vector to generate the output sequence. It generates output step-by-step, where each next token depends on the previously generated token.

This Encoder-Decoder architecture is powerful and flexible because it efficiently handles sequence transformation tasks

Page No. 4
Date

## Diagram




## Working

1. Input sequence Encoding: The encoder processes input tokens one by one and forms hidden representations using RNN/LSTM/GRU cells to capture sequential dependencies.
2. Context Vector Formation: The final hidden state of the encoder becomes the context vector, representing the compressed meaning of the full input.
3. Decoder Initialization: The decoder begins with the context vector as its initial state, allowing the decoder to start generating output based on overall meaning.
4. Step-by-Step Output Generation: The decoder generates one token at a time, and each next token depends on the previously generated one. This auto-regressive process helps maintain grammatical structure.
5. Final Output Sequence Production: The decoder continues producing tokens until it predicts an end-of-sequence token, producing a meaningful final ~~repot~~ output.

During training, the decoder is given the correct previous output rather than the predicted output. This helps the model learn faster and improves decoding accuracy by learning correct sequence patterns

## Applications

- Machine Translation: Converts sentences from

English to Hindi/France etc while maintaining grammar and meaning.

- Text summarization: Converts long text into short meaningful summaries.
- Chatbots / Dialogue systems: Generates relevant responses based on user messages.
- Speech Recognition / Captioning: Converts audio signals or image features into text sequences.

The seq2seq model is a powerful architecture for sequence transformation tasks. The encoder captures the meaning of the input, and the decoder generates structured output. Due to its flexibility and performance, it is widely used in translation, summarization, and conversational AI applications.

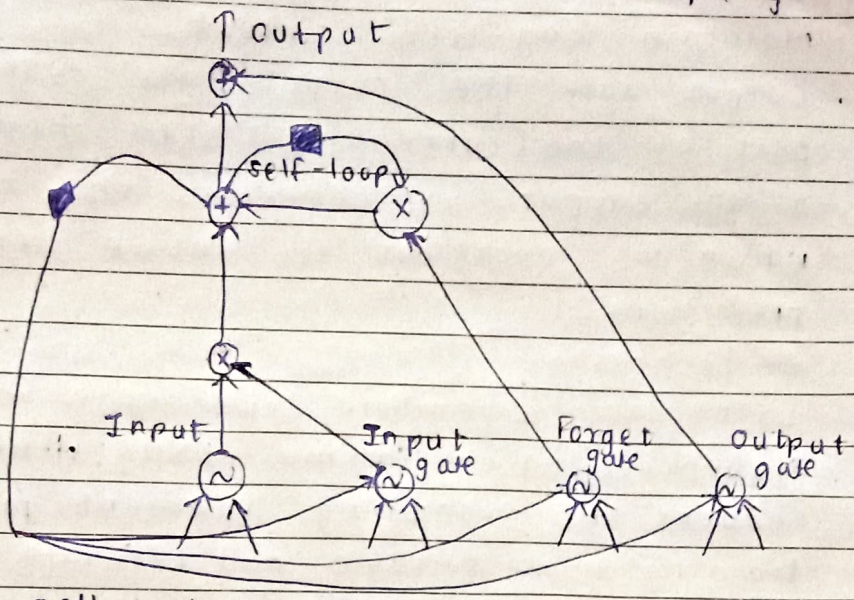
Q.2a) Describe the general layout of a Long-short Term memory Network (LSTM) with suitable diagram.

Ans. Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network architecture designed to handle long-term dependencies. Traditional RNNs face the vanishing gradient problem during training, due to which the network tends to forget important past information. LSTM solves this issue by using a memory cell and gate mechanisms that control the information flow. Because of this, LSTM performs very effectively in sequential data processing tasks where context plays an important role.

General Layout / Architecture

The architecture of an LSTM is more structured compared to a normal RNN because it

includes a cell state and multiple gates.




The cell state stores long-term information, while the gates decide which information should remain ain memory and which should be discarded. This controlled flow makes LSTM powerful since it can carry information from the beginning of a sequence to the very end.

Components of LSTM

- Forget Gate: The forget gate decides which parts of the previous memory should be removed. Through a sigmoid activation, it generates values between 0 and 1, allowing irrelevant information to be partially or completely deleted. This prevents the network from storing unnecessary patterns.
- Input Gate: The input gate determines how much of the new input should be added to the memory cell. It filters relevant information and helps the model continuously learn without overwriting old memory.
- Cell State: The cell state is the main memory storage of the LSTM that carries long-term information. It flows through the network like a highway, with small updates being applied

via the gates. This is what allows LSTMs to maintain long-term context.

- Output Gate: The output gate determines which part of the current cell state should be passed as the output. It generates the hidden state which is important for future steps and final predictions.

Advantages

LSTM can remember dependencies in long sequences, which normal RNNs struggle with. It reduces the vanishing gradient problem, making training more suitable and efficient. LSTMs also perform better with noisy and incomplete data because the forget gate removes irrelevant information, improving generalization.

Application Areas

- Natural Language Processing: LSTMs are widely used in machine translation, text generation, and language modeling. They understand sentence context and generate grammatically meaningful output, making them more accurate than basic RNNs.
- Speech Recognition: Speech is time-dependent and previous sounds influence future sounds. LSTM retains acoustic patterns and pronunciation details, providing better speech-to-text accuracy.
- Time-Series Prediction: In stock market prediction, weather forecasting, and sensor data analysis, LSTMs learn past trends to estimate future values, enabling real-time systems to provide reliable results.
- Handwriting Recognition: LSTM analyzes sequences of continuous pen strokes and identifies characters. It understands stroke order and shape dependencies, giving high accuracy, especially in cursive writing.

Thus, LSTM is a powerful RNN variant that handles long-term dependencies through. gating mechanisms It retains important information and generates accurate output, making it highly effective for sequential tasks such as NLP, speech recognition, and time-series prediction. Due to its architecture, LSTM has become a widely adopted model in real-world applications.

Q 2 b) What is Recurrent Neural Network (RNN)? State and explain types of RNN in brief

Ans. A Recurrent Neural Network (RNN) is a neural network architecture designed to process sequence data. A normal Feedforward Neural Network (FFNN) considers only the current input, but an RNN also stores memory (history) of past inputs

RNN is a type of neural network that has connections forming directed cycles, allowing it to maintain an internal memory of previous inputs, making it suitable for sequential data such as text, speech, and time series.

That means RNN learns

- Input at time t
- Plus previous output / hidden state

So it can understand

- sequences
- context
- order of data.

Why RNN?

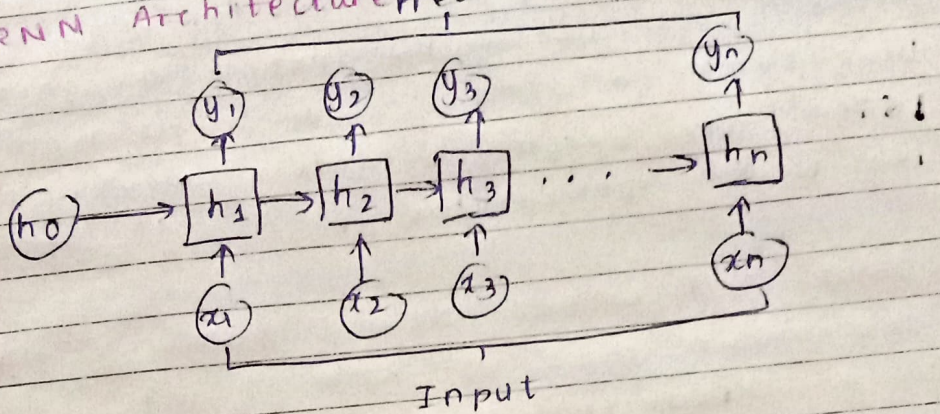
Traditional neural networks:

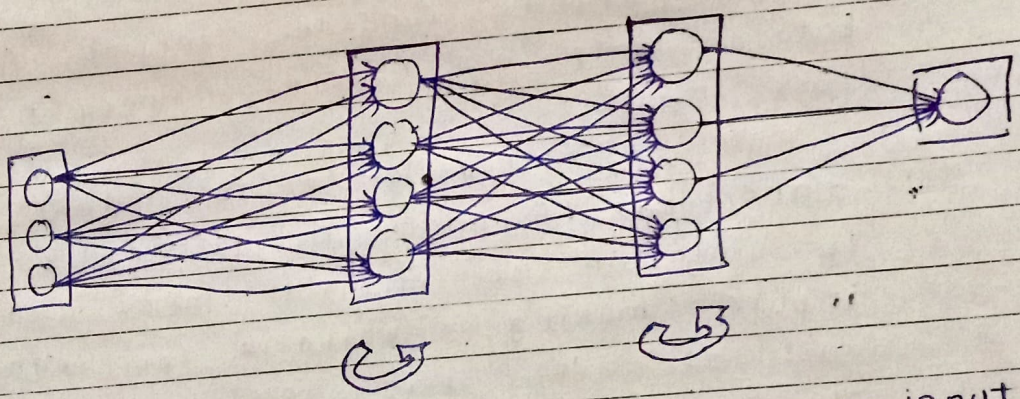
- cannot handle variable-length input
- forget previous information
- treat each input independently

RNN:

- remembers past information
- processes variable-length sequences
- maintains context

RNN Architecture




The architecture of an RNN contains an input layer, a hidden layer with recurrent connection and an output layer.

The hidden layer receives both the current and the previous hidden state, enabling information to persist over time

Working

The working of an RNN is sequential, meaning the network processes the input in the form of a sequence. Unlike normal neural networks, RNN uses the output of one time step to help in processing the next time step.

Therefore, RNN contains feedback loops / recurrent connections that maintain memory.

Step by - Step working

Step 1 - Input sequence is given

RNN receives data in the form of a sequence, for example:

| Type | Example |
|---|---|
| Sentence | "I love cats" |
| Time-series | [30°C, 31°C, 33°C, 35°C] |
| Speech | audio signal frames |

RNN receives input one time-step at a time, not all together.

This helps the network understand the order of the sequence, which FFNN cannot.

Step 2 - First Input Passes Through Hidden Layer

When the first input $X_1$ enters the RNN cell, the hidden layer processes it.

The output of the hidden layer is called: Hidden state ($H_1$)

It represents:

- meaning of first input
- extracted features
- initial memory

e.g. Sentence: "I love cats"

At the word "I", the hidden state stores: subject of sentence = I.

Step 3 - Hidden State stored as Memory

The hidden state is stored and passed to the next time step. This becomes a form of short term memory.

e.g. Word: "love"

Network remembers
"I" → subject
"love" → action
So memory becomes; "I love" → context is being built.

Step 4 - Next Input + Previous Memory Combined

When the next input $X_2$ arrives, the RNN combines:
new input + previous hidden state (memory)
Using the formula:

$$H_t = f(W_{hh} H_{t-1} + W_{xh} X_t + b)$$

Where
$W_{xh}$ → effect of current input
$W_{hh}$ → effect of previous memory
RNN updates its hidden state at every time step by combining the current input with the previous hidden state through recurrent connections

Step 5 - Output Generated

From the updated hidden state, output $Y_t$ is generated. Output depends on the task:

| Task | Output |
|---|---|
| Sentiment analysis | Positive / Negative |
| Translation | next word |
| Time-series | next value |
| Speech | Phoneme |

e.g. Sentence: "I love cats"
Final output: positive sentiment

Step 6 - Repeat for Entire Sequence

For every time step:
input → hidden update → output → memory update
This loop continues until the sequence ends

Example

Sentiment Analysis

Sentence: "The movie was amazing"

Time steps:

$x_1$ = "The"

$x_2$ = "movie"

$x_3$ = "was"

$x_4$ = "amazing"

Working

$x_1$ → subject

$x_2$ → topic ("movie")

$x_3$ → grammatical link

$x_4$ → emotional clue ("amazing")

Since RNN remmembers previous words, the

final output: Sentiment = Positive

If FFNN was used:

"amazing" alone might give wrong result because

it has no memory

Time-series Prediction

Input: 100, 102, 103, 104 → next will be 105

RNN maintains memory through hidden states,

learning patterns step by step and predicting the

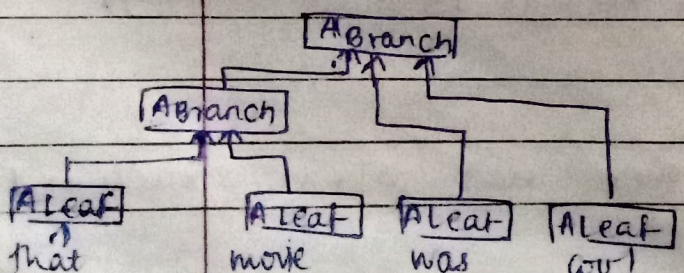
next value.

May - June 2023

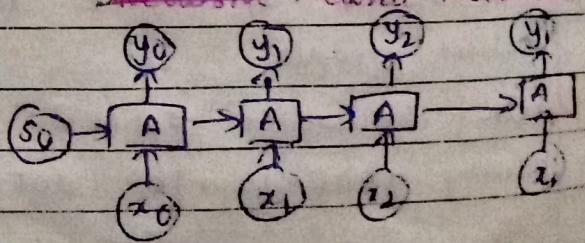
Q. 10 a) Differentiate between Recurrent Neural Network and Recursive Neural Network with appropriate diagram.

Ans

~~Recursive~~ ~~Recurrent~~ Neural Network




~~Recurrent~~ ~~Recursive~~ Neural Network.



| | | |
|---|---|---|
| Architecture | Network having Hierarchical structure, Tree-like structure | Chain-like structure known as sequential structure |
| Data Processing | It processes hierarchical data | It processes sequential and time-series data |
| Memory handling | Limited context handling | Captures context through sequential memory |
| Training Complexity | This network requires specific tree traversal algorithms for training | It involves training backpropagation Through time |
| Dependency Understanding | Explicitly models dependencies in a tree structure. | Implicitly captures dependencies in sequences. |
| Use cases | Image parsing, document structure analysis | Language modeling, speech recognition. |

(Q.2. b) Describe Recursive Neural Network and Types of Recursive Neural Network. Explain its advantages.

Ans. A Recursive Neural Network is a special type of neural network designed to process tree structured/hierarchical data. While an RNN handles sequential (linear) data, a Recursive NN handles hierarchical structure.

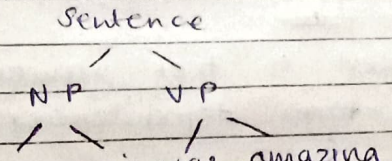
A Recursive Neural Network is a type of neural network that applies the same set of weights recursively over a structured input (such as a parse tree or hierarchical data) to produce a representation of the whole structure.

e.g. sentence structure tree, image part hierarchy, scene graph.

Why Recursive RNN?

Many real-world data are not linear, structured

and have parent-child relationships.
e.g. sentence "The movie was amazing"
Grammar structure tree:




Recursive NN processes this type of tree structure

Working

1. Takes input in the form of tree nodes: The input is represented as a tree structure where each element becomes a node (e.g., words in sentences or parts of an image).
2. Processes leaf nodes first: The smallest units (words/sub-parts) are converted to vector representation.
3. Combines representations: Two related leaf nodes are combined to form a richer semantic representation (e.g., "The" + "movie" → noun phrase meaning).
4. Creates parent node representation: The combined meaning is stored as the parent node and the process continues upward.
5. Produces final meaning for the whole structure: Top node = global semantic representation → used for classification / sentiment / similarity.

## Types of Recursive Neural Networks

1. Structure-Based Recursive NN
   - Follows parse tree structure to combine nodes.
   - Captures grammatical relationships
   - Used in NLP syntax trees.

2. Tensor-Based Recursive NN
   - Uses tensor operations for combining child nodes.

- Models complex interactions
- More expressive but computationally expensive

3. Tree LSTM
- LSTM extended to tree structure
- Handles long-term dependencies using gates
- Used in sentiment analysis and semantic tasks

4. Deep Recursive NN
- Multiple recursive layers for deeper feature extraction.
- Higher accuracy but high training cost.

Advantages

1. Handles hierarchical structured data.
2. Learns relationship between parts and whole.
3. Captures context more accurately.
4. Better semantic understanding
5. Weight sharing reduces parameters

Applications

1. Natural Language Processing: Processes sentences using parse trees to capture grammar, structure, long dependency relations. Applications: sentiment analysis, question answering, semantic similarity.
2. Sentiment Analysis: e.g Sentence: "The movie was not good" Sequential RNN may assume "good" → positive but Recursive NN understands "not good" → negative through tree structure
3. Image understanding: Breaks an image into parts (object → components → full scene) & extracts meaning
4. Scene Graph Processing: Understands relational meaning: person → holding → cup
5. Parsing / Syntax Trees used to extract grammatical structures

NOV-DEC-2024

(Q1 b) Explain in detail the following advantages of RNN over other types of Neural Network

i) Ability to Handle variable length Sequences

ii) Memory of Past Inputs

Ans i) Ability to Handle Variable length Sequences

RNNs can process inputs of different lengths, which is not easily possible in traditional feed-forward neural networks that require fixed-size input. Since RNNs take input step-by-step & update their hidden state at each time step, they can naturally work with sequences like sentences, speech, or time-series data which may not have a fixed size.

e.g. A sentence may be 5 words long or 20 words long

- An audio signal may be 1 second or 10 seconds long

RNNs can still process both smoothly because it reads them sequentially.

RNNs are capable of handling variable length sequential data because they process input one element at a time and maintain continuity through hidden states

ii) Memory of Past Inputs.

RNNs have a memory mechanism through their hidden state, which stores information about previous inputs in the sequence. This enables RNNs to understand context and dependencies over-time - something that feed-forward networks cannot do because they treat each input independently.

Example: In the sentence "The movie was amazing", understanding "amazing" requires knowing the earlier word "movie" RNN stores earlier context in its hidden state and uses it to interpret later words. This gives RNN advantage in tasks requiring context, history, and understanding, such as:

- Language modeling
- Sentiment Analysis
- Machine translation
- Stock price prediction.

RNNs maintain a hidden state that stores previous sequence information, giving them the ability to remember past inputs and use them for future predictions.

Q.2.a) Discuss how RNNs suffer from the problem of vanishing gradients And how it is overcome with Long Short-Term Memory Network (LSTM)?

Ans. Vanishing Gradient Problem in RNN

During the training of an RNN, the network uses Backpropagation Through Time (BPTT). Gradients are propagated backwards across many time steps. As the gradients flows backward through each layer:

- The gradients get multiplied repeatedly by small values (less than 1)
- They become very tiny and approach zero.
- This is called the vanishing gradient problem.

Because of vanishing gradients:

- RNN cannot learn long-term dependencies
- Only short-term relationships are captured
- Training becomes slow and unstable
- Model accuracy decreases for tasks requiring long context.

e.g. In a long sentence, an RNN may forget early words while processing the last words, so understanding long sentences becomes difficult.

Vanishing gradients make RNN unable to retain information for long sequences, as gradients shrink to extremely small values during backpropagation.

## How LSTM overcomes the Vanishing Gradient Problem

LSTM (Long Short-Term Memory) is a special type of RNN designed to handle long-term dependency problems. LSTM introduces a memory cell and gates:

- Forget Gate - decides what information to remove
- Input Gate - decides what new information to store
- Output Gate - decides what information to pass to the next step.

These gates allow controlled flow of information, preventing gradient values from becoming too small or too large.

LSTM maintains constant error flow through the cell state, ensuring gradients do not vanish. Thus, LSTMs can remember information for long periods of time and learn long-term dependencies effectively.

- LSTM overcomes vanishing gradients by using gated cell units that maintain stable gradients, enabling learning over long ~~term~~ sequences.