

In this lesson, we are going to learn about **deep learning models** that work well with **sequential data**, which means data that comes in a particular order or sequence. For example, a paragraph of text, a series of weather readings, or even a recording of someone's voice.

**What are Sequence Models?**

**Sequence models** are a type of deep learning model designed to handle **ordered lists of data points**, like sentences, stock prices over time, or musical notes in a song. These models try to find patterns or relationships in the sequence and use this understanding to make **predictions**, classify the data, or even generate new sequences.

Now basically Deep-learning may **2 types** of model atay hain ek wo model jo **sequential data** ko process krtay hain or dusray wo models jo **non-sequential data** ko process krtay hain . Now sequential data means text, audio, music etc these all are counted as sequential data bcuz jo yeh sequential model hotay hain yeh sequence ki base pa data learn krtay hain such as **RNN** (this model learn data by predicting the next word in data on the basis of previous word. So it using a order for making predictions,). Now non-sequential data is like : Image . AB jo text hota hai usko toh words/token may break/divide krkay ek sequence may predict krletay hain but jo Image hai that is combination of **Pixels** so agar Image ko pixel may break krkay then one by one in sequence may classify krnay ki koshish kreinga toh wo Image ka context hi nhi smjh payega kay yeh Image hai kis ki isi lia Image is non-sequential data and isko process krnay kay liya jo popular Algo hai that is **CNN.**

**Where are Sequence Models Used?**

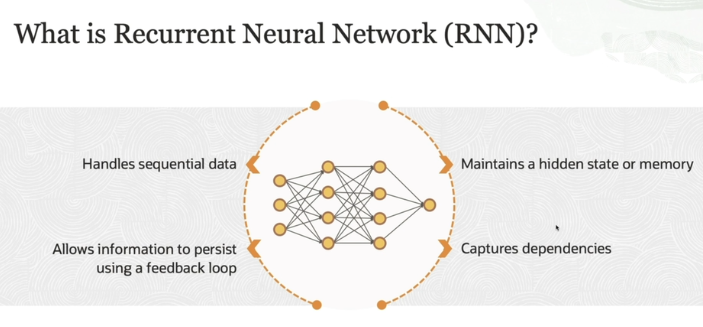
* **Natural Language Processing (NLP)**: In tasks like **machine translation** (translating a sentence from one language to another), **sentiment analysis** (figuring out whether a sentence expresses a positive or negative feeling), or even **text generation** (writing new sentences based on previous input).
* **Speech Recognition**: These models listen to recorded **audio** and convert it into **text**. Think of how Siri or Google Assistant can understand what you're saying.
* **Music Generation**: Sequence models can learn patterns in music and generate entirely new musical pieces based on those patterns.
* **Sign Language Recognition**: These models can analyze a series of **hand gestures** and interpret them into meaningful text or commands.
* **Finance and Weather Prediction**: In fields like **finance** (predicting stock prices) or **weather forecasting** (predicting future weather based on historical data), sequence models are used to analyze **time-series data**, where past values help predict future outcomes.

**How do Deep Learning Models Work with Sequence Data?**

There are different types of deep learning models that are well-suited to **sequential data**, such as **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks. These models can understand the **context** within a sequence, which is crucial for making accurate predictions or generating realistic outputs.

In summary, sequence models are essential for working with data that has an inherent order, and they are used across various fields like language processing, music, finance, and more.

 Let us see some of the deep learning models which can be used to work with sequence data.



**For more info:** [**https://www.ibm.com/topics/recurrent-neural-networks**](https://www.ibm.com/topics/recurrent-neural-networks)

**Recurrent Neural Networks (RNNs):**

* **Neural Networks in general:** These are models that try to mimic how the brain processes information. They are made up of layers of interconnected "neurons" that can learn patterns from data.
* **Sequential Data:** Unlike images or static data, sequential data includes time-based or ordered data. Examples are things like sentences (where the order of words matters), stock prices (which change over time), or music notes (where each note depends on the previous one).
* **Feedforward Neural Networks vs. RNNs:**
  + **Feedforward Neural Networks:** Information moves in one direction—from input to output—without looping back.
* **RNNs and Feedback Loops:**
  + Imagine you're reading a story, and each word you read helps you understand the next word better because you remember what you've already read.
  + **RNNs** work the same way! They don't just look at the current piece of information (like the current word); they also **remember what came before** (like the previous words in the story). This is thanks to their **feedback loop**.
  + In a regular neural network, once the network looks at some data, it forgets it immediately. But in an RNN, it can **remember** past data and **use it to make better predictions** for what comes next.
* **Internal State (Hidden State or Memory):**
  + In RNNs, there’s something called a **hidden state** or "memory."
  + This hidden state keeps track of the past information in a sequence. Every time the model processes a next part of the sequence (like a next word in a sentence), it updates this hidden state.
  + Then, when the next part of the sequence comes in, the RNN doesn't just rely on the current input but also takes into account the hidden state (the memory of what it has seen before).
* **Capturing Patterns Across Time:**
  + The feedback loop allows RNNs to spot patterns and connections that aren't just in the current input but also come from earlier in the sequence.
  + For example, in a sentence, the meaning of a word might depend on the words that came before it. RNNs help capture these kinds of dependencies.

In short: RNNs are like memory-boosted neural networks that are specially designed for time-based or ordered data. They remember past information and use it to influence future steps, helping them understand patterns that unfold over time.

let's talk about a limitation or **"fallback"** of Recurrent Neural Networks (RNNs):

**The Problem: Vanishing Gradient**

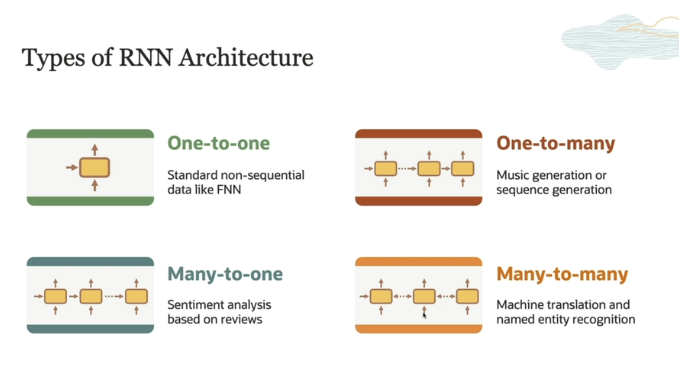
* When RNNs try to capture long-term dependencies (patterns or information far back in the sequence), they struggle because the information fades as the sequence gets longer. This is known as the **vanishing gradient problem**.
* During training, RNNs use a method called **backpropagation through time (BPTT)** to update their weights. However, when dealing with long sequences, the gradients (which tell the model how to adjust its weights) can become extremely small, causing earlier information in the sequence to "vanish" or be lost.

**Impact:**

* This means RNNs are great at handling short-term dependencies but struggle to remember information from much earlier in long sequences. For example, in a long paragraph, they may "forget" important context from the beginning while processing the end.

**Solution: LSTMs and GRUs**

* To overcome this fallback, two advanced versions of RNNs were developed: **LSTMs (Long Short-Term Memory networks)** and **GRUs (Gated Recurrent Units)**.
  + These models include special "gates" that help them maintain important information for longer periods, reducing the vanishing gradient issue.
  + They can decide what to remember and what to forget, making them more efficient for long sequences.



**1. One-to-One:**

 This is just like a **basic neural network**, not an RNN. It only takes **one input** and gives **one output**.

 **Example:** Imagine a machine that takes a single image and says, "This is a cat" or "This is a dog." It gives **one result for one input**.

 It’s not good for handling sequences like sentences or time-based data because there’s no memory of past inputs.

**2. One-to-Many:**

* In this setup, you have **one input** but generate **multiple outputs** over time.
* **Example:** Music generation or sequence generation. Here, the model takes in one piece of information, like a musical note or a start prompt, and then generates a whole sequence of outputs (like a melody or a text).

**3. Many-to-One:**

* This is when the model takes in **multiple inputs** and gives out **one output**.
* **Example:** Sentiment analysis. The input could be a sentence (multiple words), and the model produces a single output (like predicting whether the sentiment is positive or negative).

**4. Many-to-Many:**

* In this architecture, you give the model **multiple inputs** and it produces **multiple outputs**.
* **Example:** Machine translation or named entity recognition (NER). For machine translation, the input is a sentence in one language, and the output is a sentence in another language, where both inputs and outputs are sequences.

**Key Limitation:**

* **Vanishing Gradient Problem:** RNNs struggle when it comes to **long-term memory** or patterns that happen far apart in a sequence. This is because of the **vanishing gradients problem** during training, where information from earlier parts of the sequence fades away.
  + **Solution:** This problem is usually handled by using **LSTM (Long Short-Term Memory) networks**, which are designed to remember long-term dependencies much better.

**What is vanishing gradient problem ?**

The **vanishing gradient problem** happens when training deep neural networks, especially when they have many layers. Let me explain it in simple terms:

**What is Gradient?**

When a neural network learns, it adjusts the **weights** (connections between neurons) using a method called **backpropagation**. In this process, the network calculates something called a **gradient**, which tells it how much to change the weights to reduce the error (make better predictions). This is like giving directions on how to move toward a better solution.

**What is the Vanishing Gradient Problem?**

In deep networks, gradients are calculated layer by layer, starting from the output and going backward to the input. But as we move back through the layers, the gradients (the directions for changing the weights) can become **very small** or even **close to zero**.

This is the **vanishing gradient problem**. When the gradients are very small, the weights in the earlier layers (or we can say jo shuru ki layers hoti hain) barely change, and the network doesn’t learn well. This slows down or even stops the training process because the earlier layers can't adjust properly.

**Why Does it Happen?**

It mainly happens because of the **activation functions** (like Sigmoid or Tanh), which can "squash" large input values into a small range (e.g., between 0 and 1). This squashing makes the gradients smaller and smaller as they pass through layers, leading to the vanishing effect.

**What’s the Impact?**

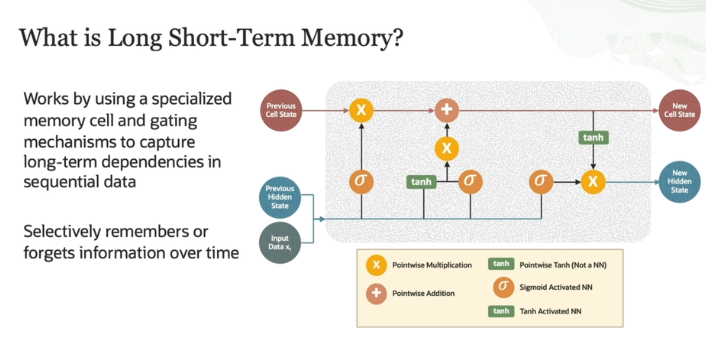
* **Slow learning**: Since the weights don't change much in the early layers, the network takes longer to learn.
* **Poor performance**: The model may not learn complex patterns well, affecting its ability to make accurate predictions.

**How Do We Solve It?**

Some techniques to fix the vanishing gradient problem include:

* Using **ReLU (Rectified Linear Unit)** activation functions, which don’t squash values as much as Sigmoid or Tanh.
* Using architectures like **LSTMs** or **GRUs**, especially in sequence models like RNNs, which help with better gradient flow.

In short, the vanishing gradient problem makes it hard for deep networks to learn, but modern techniques have been developed to overcome this issue!



**What is LSTM?**

LSTM is a special type of neural network designed to handle **sequential data** (data that comes in a sequence, like text or time series). The main challenge with sequences is that it's hard for traditional neural networks to remember information from earlier in the sequence when processing later parts. That's where LSTM comes in.

**How does LSTM work?**

LSTM has **memory cells** that allow the network to store information for long periods and **gates** that control what information is remembered or forgotten. This helps overcome the **vanishing gradient problem** (when gradients get too small during backpropagation, making it hard to train).

1. **Memory Cell**: This is where LSTM stores long-term information.
2. **Gates**: Think of gates like doors that decide whether information should enter, stay in memory, or leave. There are three main gates:
   * **Forget Gate** (σ in the image): Decides what information from the previous state to forget.
   * **Input Gate** (also σ): Decides what new information to add to the memory.
   * **Output Gate**: Decides what part of the memory to output.

**What happens step by step (from the image)?**

* **Previous Cell State**: Information from the past comes in, and LSTM decides if it should keep or forget this information (Forget Gate).
* **Input Data**: The current input (like the next word in a sentence) also comes in. The Input Gate decides whether this new data should be stored.
* **New Cell State**: After processing, LSTM updates its internal memory with new information (addition or removal of information based on the gates).
* **New Hidden State**: Finally, the updated memory is passed on as the output for this step, which will be used in future steps.

**Key Idea:**

LSTM **remembers** important things for a long time and **forgets** things that aren’t useful, allowing it to work well with sequences like sentences, speech, or time-series data.

This ability to **selectively remember or forget** over time is what makes LSTM so powerful for tasks like **speech recognition**, **text generation**, and even predicting the **stock market**.

**1. Speech Recognition:**

In speech recognition, the input is a sequence of sounds, which are processed one after another.

* **How LSTM helps**: LSTM can remember patterns in the sound over time. For instance, the sound of a word is not just based on the current sound but also on the sounds before it. LSTM uses its memory to store these patterns and recognizes words more effectively because it can remember the context from earlier parts of the speech.
* **Example**: If you're saying, "Hello, how are you?"—the LSTM network will remember the beginning sound ("Hel") while it's processing the end ("lo") to help recognize that the word is "Hello." It doesn't forget earlier sounds too quickly.

**2. Text Generation:**

In text generation, the input is a sequence of words or characters, and the model’s task is to predict what comes next.

* **How LSTM helps**: LSTM remembers the context from earlier words in a sentence. For example, if you type "I love to eat", LSTM will use the memory from earlier words to predict the next word (e.g., "pizza" or "cake").
* **Example**: Imagine you are generating a story. When the LSTM sees "Once upon a time, there was a..." it remembers the past words and predicts "princess" or "dragon" because it learned from similar stories in the past. It keeps track of the sequence and context to generate meaningful sentences.

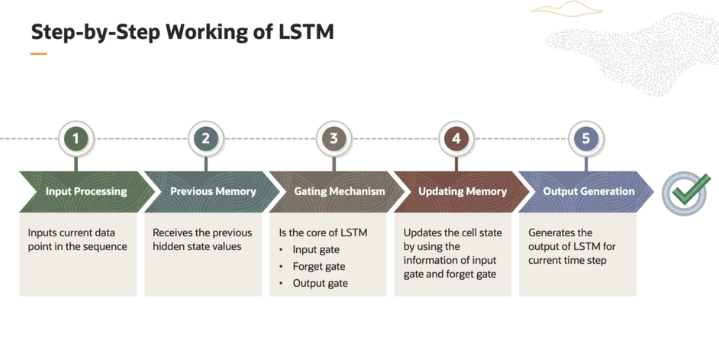
**3. Predicting the Stock Market:**

In stock market prediction, the input is time-series data (e.g., stock prices at different times), and the goal is to predict future stock prices.

* **How LSTM helps**: LSTM can capture long-term dependencies in the data. For example, the stock price today depends on not just yesterday's price but also on trends from several days or even weeks ago. LSTM can "remember" these patterns and use them to predict future prices.
* **Example**: If a company's stock rises after each quarterly earnings report, LSTM can remember this pattern over several quarters. When the next earnings report is approaching, LSTM will predict an increase in the stock price because it learned from past behavior.

**Why LSTM is Important for These Tasks:**

* In **speech recognition**, you need to remember earlier sounds to understand the full word or sentence.
* In **text generation**, you need to keep track of the context to generate meaningful and coherent sentences.
* In **stock market prediction**, you need to recognize trends and patterns over time to make accurate predictions.



**Step 1: Input Processing**

* **What happens?**  
  At each time step, LSTM takes the current input data point from the sequence. For example, if you're working with a sentence, this could be a word.
* **Example:**  
  Let's say you have the word "Hello." LSTM takes this word as the input for this time step.

**Step 2: Previous Memory**

* **What happens?**  
  LSTM remembers past information using two things: the **previous hidden state** and the **previous cell state**. These two are like a memory bank that keeps track of what the LSTM has learned so far.
* **Example:**  
  If LSTM is reading a sentence, the previous memory might contain information about earlier words like "I" and "am."

**Step 3: Gating Mechanism**

* **What happens?**  
  This is the core part of LSTM where three gates—**Input gate**, **Forget gate**, and **Output gate**—decide what to do with the new input and previous memory. The gates work like filters that control what information to keep, what to forget, and what to send forward.
  + **Input gate:** Decides what part of the new input should be added to the memory cell.
  + **Forget gate:** Decides which information from the previous memory cell should be forgotten.
  + **Output gate:** Controls what information is passed to the next time step as the output.
* **Example:**  
  If LSTM is processing the sentence "I am eating pizza," the **Forget gate** might decide to forget the less important information (like "I am") while remembering the key part ("eating pizza").

**Step 4: Updating Memory**

* **What happens?**  
  Based on the decisions made by the input and forget gates, the LSTM updates its memory (the cell state). This is where the LSTM either adds new information or discards old information.
* **Example:**  
  If the forget gate decided to forget the earlier words "I am," the memory is updated to focus on "eating pizza."

**Step 5: Output Generation**

* **What happens?**  
  The LSTM now generates the output for this time step, using the updated memory. This output becomes the hidden state for the next time step.
* **Example:**  
  After processing the word "eating," the LSTM produces an output that influences how it will understand the next word "pizza."

So its mean jo input,forget,output gate hota hain yeh decide krtay hain kay memory cell may kya store hoga. Like input gate input receive krta hai then it decide that what should be added to memory cell from this input, then **forget gate** check that what should be removed from the previous memory cell and on the basis of **input gate and forget gate** memory cell is updated . Or then **output gate** check that which information should be outputted from the memory cell.