📌 Tokenization, Encoding, and Embedding

✒️ **Tokenization**: The process of breaking down text into smaller units called tokens, such as words or subwords.
Example: "Machine learning is fun" → ['Machine', 'learning', 'is', 'fun']

✒️ **Encoding**: Converting tokens into numerical representations (integers) that models can process.
Example: ['Machine', 'learning', 'is', 'fun'] → [1, 2, 3, 4]

Primary Types of Encoding

Encoding Type	Description	Example
Label Encoding	Assigns a unique integer to each category.	Categories: ['Low', 'Medium', 'High'] Low → 0 Medium → 1 High → 2
One-Hot Encoding	Converts categories into binary vectors.	Categories: ['Red', 'Green', 'Blue'] Red → [1, 0, 0] Green → [0, 1, 0] Blue → [0, 0, 1]

✒️ **Embeddings**: Embeddings are continuous vector representations of words or categories that capture their meanings and relationships in a lower-dimensional space.

How it works:
- Words or categories are represented as vectors in a multi-dimensional space
- Similar words have vectors that are closer together

Example: Vector('king') - Vector('man') + Vector('woman') ≈ Vector('queen')

Commonly used Embedding Models: Word2Vec, GloVe, BERT

🔹 Why ANN and CNN Are Not Enough?

1. **Fixed Input and Output Size**
    - Traditional ANNs and CNNs work with fixed-size input and output structures.
    - However, many real-world tasks require processing variable-length sequences (e.g., speech, text, stock prices).
2. **Lack of Memory**
    - ANNs and CNNs treat each input independently and do not retain past information.
3. **Not Designed for Time-Series and Sequential Data**
    - Time-dependent relationships in sequences (e.g., word order in a sentence) cannot be captured well.

📌 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of artificial neural network specifically designed to handle sequential and time-dependent data by maintaining a memory of previous inputs.

Unlike feedforward networks (such as ANN and CNN), which process inputs independently, RNNs retain information from past inputs using a hidden state. This makes them suitable for tasks where order and context matter, such as speech recognition, machine translation, and time-series forecasting.

Example:
Used in **Text Generation, Machine Translation, Speech-to-Text (ASR), Sentiment Analysis, Speech Recognition, Voice Cloning & Text-to-Speech (TTS), Stock Market Prediction, Weather Forecasting, Sales & Demand Forecasting**, etc.

🔹 Why Do We Need RNN?

1. **Handles Sequential Data**
   - RNNs process inputs sequentially, maintaining a hidden state that carries information from previous steps.
   - This makes them ideal for tasks like speech recognition, language modelling, and time-series forecasting.

2. **Captures Temporal Dependencies**
   - Unlike CNNs, RNNs have a memory mechanism that retains past information and uses it in future predictions.
   - This is crucial for context-based understanding in NLP and time-series predictions.

3. **Flexible Input and Output Lengths**
   - RNNs can handle sequences of variable lengths, making them useful for applications like:
     - Machine translation (e.g., input: English, output: French)
     - Speech-to-text conversion
     - Stock price prediction

✒️ Important Terms:

- **Hidden State:** In a **Recurrent Neural Network (RNN)**, the **hidden state** is a vector that stores **contextual information** from previous time steps, allowing the network to retain memory across sequences. Unlike traditional feedforward networks, **RNNs process sequential data one step at a time**, and the hidden state acts as a **memory** that is updated at each step.

- **Why is the Hidden State Important?**
  - **Captures Sequential Dependencies:** Stores memory of previous inputs. Helps model long-term relationships in text, speech, and time-series data.
  - **Passes Information Across Timesteps**: Unlike feedforward networks, RNNs use hidden states to **preserve context**.

- **Time Steps:** In a **Recurrent Neural Network (RNN)**, a **time step** refers to a single point in the sequence where the network processes an input and updates its **hidden state**. RNNs handle **sequential data**, meaning they process one element at a time while maintaining memory through hidden states.

✒️ Visual Representation of a state flow in RNN:

Diagram

```markdown
    x1          x2          x3          x4
     ↓           ↓           ↓           ↓
┌────►────┬────►────┬────►────┬────►────┐
│   h0    │   h1    │   h2    │   h3    │
└────▲────┴────▲────┴────▲────┴────▲────┘
     |           |           |           |
    y1          y2          y3          y4
```

Explanation

- **Inputs** (x_1, x_2, x_3, x_4) → Words, audio frames, or time-series data.
- **Hidden States** (h_0, h_1, h_2, h_3) → Carries context forward.
- **Outputs** (y_1, y_2, y_3, y_4) → Predictions at each step (e.g., next word in a sentence).

✒️ Example: Weather Prediction Using RNN:

Diagram

```sql
  Day 1      Day 2      Day 3      Day 4      Day 5
 Temp1 →   Temp2 →   Temp3 →   Temp4 →   Temp5
   ↓          ↓          ↓          ↓          ↓
┌───►───┬───►───┬───►───┬───►───┬───►───┐
│  h0   │  h1   │  h2   │  h3   │  h4   │
└───▲───┴───▲───┴───▲───┴───▲───┴───▲───┘
    |          |          |          |          |
  Pred1      Pred2      Pred3      Pred4      Pred5
```

How It Works

- **Input** $(Temp_t)$ → Past weather data (temperature, humidity, wind speed).
- **Hidden State** (h_t) → Stores memory of past weather conditions.
- **Prediction** $(Pred_t)$ → Forecasted weather for the next day.

✒️ What is an Out-of-Vocabulary (OOV) Token in RNN?

In **Recurrent Neural Networks (RNNs)** used for **Natural Language Processing (NLP)**, an **Out-of-Vocabulary (OOV) token** represents any word that is **not present in the model's vocabulary** during training.

✒️ Types of RNN:

- **Basic RNN:**
  - Simple form of RNN with recurrent connections, but suffers from vanishing gradient issues.
  - **Example:** Predicting stock prices based on past trends.

- **Long Short-Term Memory (LSTM):**
  - Overcomes vanishing gradients using gates (forget, input, output) to retain long-term dependencies.
  - **Example:** Sentiment analysis in NLP.

- **Gated Recurrent Unit (GRU):**
  - A simplified LSTM with fewer parameters, making it computationally efficient.
  - **Example:** Real-time translation apps.