RNN Intro

What are RNNs?

- Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to work with **sequential data or time series data**.
- Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing information to persist from one step of the sequence to the next.

The key is:

RNNs **remember** past inputs — they have a "memory" of previous steps in a sequence.

RNNs are a type of neural network that work **really well with sequence data**, like:

- 🔤 Text (e.g., predicting next word)
- 🗣️ Speech/audio
- 📈 Time series (stock prices, weather)

🧠 How Does It Work?

Imagine you're reading a sentence word-by-word. RNNs process:

```
Input 1 → Hidden state → Output 1
Input 2 + Hidden state from step 1 → Output 2
Input 3 + Hidden state from step 2 → Output 3
...
```

Each step shares info forward 🔁

So the network "remembers" what it saw before.

- The word **"Recurrent"** means **"repeating" or "looping"**.

- Unlike normal neural networks, an RNN has a **memory** — it remembers past inputs when making decisions.

🧠 Simple Analogy:

Imagine you're reading a sentence, **"I went to the market and bought... 🍎"**

To guess the next word, you need to remember what came before. A regular neural network looks at each word separately, but an RNN **remembers the past words** and uses that to guess what comes next.

🔄 How RNN Works Internally?

Let's say you feed in a sentence:

"I love pizza"

RNN will process it like this:

Step	Input	Memory (from previous step)	Output
1	"I"	None (first word)	Something
2	"love"	Memory from "I"	Something else
3	"pizza"	Memory from "I love"	Final output

So it **passes memory forward** at every step.

Internally:

At each time step ttt, the RNN does this:

$$h_t = \text{activation}(W \cdot x_t + U \cdot h_{t-1} + b)$$

Where:

- x_t = input at time t
- h_{t-1} = memory from previous time
- W, U = weight matrices (learned values)
- b = bias
- h_t = current hidden state (memory + current input)

🧱 Structure of an RNN Cell

Think of each RNN cell like a little engine that:

1. Takes the current input
2. Takes what it remembers from before
3. Updates what it knows (memory)
4. Passes new memory forward

A sequence of these cells can process **a whole sentence or time series** step-by-step.

Common Applications

- Time series prediction
- Natural language processing (text generation, machine translation)
- Speech recognition
- Video analysis
- Music composition

Limitations of Basic RNNs

1. **Vanishing/Exploding Gradients**: Difficulty learning long-range dependencies due to gradient issues during backpropagation through time

2. **Short-term Memory**: Basic RNNs struggle with remembering information from far back in the sequence

Improved Variants

To address these limitations, more advanced architectures were developed:

- **LSTM (Long Short-Term Memory)**: Uses gating mechanisms to control information flow
- **GRU (Gated Recurrent Unit)**: Simplified version of LSTM with fewer parameters

🧪 Sample RNN in TensorFlow/Keras:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),  # For wo
rd embeddings
    SimpleRNN(64),  # The RNN layer
    Dense(1, activation='sigmoid')  # Output for binary classification
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
acy'])
```