Why RNN cam into being, what was the main purpose?

Recurrent Neural Networks (RNNs) came into being primarily to **handle sequential data**—that is, data where the **order of elements matters**, like:

- Time series (e.g., stock prices, weather data)
- Natural language (e.g., sentences, speech)
- Video frames
- Music/audio streams

Main Purpose of RNNs:

Traditional neural networks (like feedforward or multilayer perceptrons) **cannot retain context or memory** across inputs. This makes them ineffective for tasks where the previous input influences the current one.

RNNs were introduced to address this limitation. Their key purpose was to:

Model temporal or sequential dependencies by maintaining a "memory" of previous inputs via recurrent connections.

This enables RNNs to:

- Predict the next word in a sentence
- Translate sentences (machine translation)
- Recognize speech
- Generate text or music
- Analyze time-dependent patterns in data

Core Idea:

RNNs process sequences **one element at a time**, passing hidden state information from one step to the next. This hidden state acts like a **memory**, allowing the network to use prior context when making predictions.

In Summary:

RNNs were created to **solve problems where context and sequence matter**, by enabling neural networks to maintain memory of previous inputs. This was a major step forward for tasks in NLP, speech, and time-series analysis.

Explain to me the architecture of the RNN in easy words. Explain from input to the output layer?

🎯 Goal of the RNN in the Image

To process **sequential data** (like a sentence) for a task such as **sentiment analysis**, and produce a final output prediction y^4\hat{y}_4y^4, like "positive" or "negative".

🧱 Step-by-Step Architecture Explanation

---

1. Input Sequence (bottom of the image)

plaintext

CopyEdit

$x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$

- This is a sequence of vectors (e.g., words in a sentence turned into vectors).
- Each $x_{1t} \in \mathbb{R}^d$ — a d-dimensional input at time step t.

2. Unrolling Over Time

plaintext

CopyEdit

t=1 → t=2 → t=3 → t=4

- RNNs process data one time step at a time.
- The same neural network (function **f**) is **repeated** (or "unrolled") for each step in time.

3. Hidden States (o_1, o_2, o_3, o_4)

- At each time step, a **hidden state vector** o_t is calculated:

$$
\begin{aligned}
o_1 &= f(x_{11}, w) \\
o_2 &= f(x_{12}, w + o_1, w) \\
o_3 &= f(x_{13}, w + o_2, w) \\
o_4 &= f(x_{14}, w + o_3, w)
\end{aligned}
$$

- These states **carry memory** — they depend on:
  - the current input x_t
  - and the **previous hidden state** o_{t-1}

4. Final Output ($\hat{y}_4$)

- After processing the whole sequence, the **last hidden state** o_4 is passed to a function (another layer) to get the output:

$$\hat{y}_4 = \text{Softmax}(f(o_4, w''))$$

- Softmax turns the output into **probabilities** over categories (e.g., 80% positive, 20% negative sentiment).

5. Loss Function

plaintext

CopyEdit

Loss = |ŷ - y|

- The predicted value y^4\hat{y}_4y^4 is compared to the true label y4y_4y4.
- The difference is used to **train** the model using backpropagation through time (BPTT).

🔁 **Summary:**

Step	What Happens
Input	A sequence of vectors (e.g. word embeddings) is fed in over time.
Hidden States	Memory is carried forward across time using hidden states.

Step	What Happens
Output	The last hidden state is used to make a final prediction.
Training	A loss function compares prediction to truth and updates weights.

Take an example and apply it on the architecture to understand the working of RNN.

Great! Let's walk through a **simple example** of how an **RNN** processes input **step-by-step** using the architecture from your image — especially for a task like **sentiment analysis**.

🎯 **Task:**

Predict the **sentiment** (positive or negative) of the sentence:

"**I love this**"

🧩 **Step 1: Break the sentence into words**

The input sequence has 3 words:

plaintext

CopyEdit

x_1 = ["I", "love", "this"]

🔤 **Step 2: Convert each word into a vector**

Word	Vector (x_t)
"I"	[0.1, 0.2, 0.3, 0.4]
"love"	[0.9, 0.1, 0.4, 0.7]

Word Vector (x_t)

"this" [0.3, 0.5, 0.2, 0.1]

🔁 Step 3: Process each word one-by-one through the RNN

At each time step t, the RNN computes:

$$h_t = f(x_t, h_{t-1})$$

Where:

- x_t: input at time t
- h_{t-1}: previous hidden state
- f: some function (usually tanh or ReLU + matrix operations)

Let's assume the hidden state size is 4, and initial state $h_0 = [0, 0, 0, 0]$

🕐 t=1: Input "I"

$$h_1 = f(x_1, h_0) = f([0.1, 0.2, 0.3, 0.4], [0, 0, 0, 0]) = \text{some output} = o_1$$

🕑 t=2: Input "love"

$$h_2 = f(x_2, h_1) = f([0.9, 0.1, 0.4, 0.7], o_1)$$

🕒 t=3: Input "this"

$$h_3 = f(x_3, h_2) = f([0.3, 0.5, 0.2, 0.1], o_2)$$

🎯 Final output

Now we take h3h_3h3 (the last hidden state) and pass it through a **Softmax layer** to predict sentiment.

Let's assume:

plaintext Copy Edit

```
f(h₃) → Softmax → [0.91, 0.09]
```

This means:

- 91% probability it's **positive**
- 9% probability it's **negative**

So the model predicts **positive sentiment**.

📉 Loss calculation:

Compare predicted output $\hat{y} = [0.91, 0.09]$ with true label
Let's say true label = [1, 0] (positive)

Loss is calculated (e.g., cross-entropy loss), and gradients are **backpropagated** through all time steps (**Backpropagation Through Time - BPTT**).

🧠 Summary of the Example

Time Step	Word	Input Vector	Hidden State	Output
t=1	"I"	x_1	$o_1 = h_1$	—
t=2	"love"	x_2	$o_2 = h_2$	—
t=3	"this"	x_3	$o_3 = h_3$	ŷ

- The final hidden state o3o_3o3 contains context from all previous words.
- It's used to **predict the sentiment**.