The architecture of a Recurrent Neural Network (RNN) is specifically designed for sequential data, such as time-series, text, or audio. RNNs are unique in that they use feedback loops to process sequences, allowing them to retain information from previous time steps and thus learn dependencies over time. Here's a breakdown of the key components and structure of an RNN:

## 1. Basic Components of an RNN Layer

- **Input Layer**: Receives sequential input data. For example, if processing a sentence, each word is processed in sequence, one at a time.

- **Recurrent (Hidden) Layer**: This layer has connections looping back onto itself, creating a "memory" that stores information from previous steps in the sequence.

    - The hidden layer at each time step has an associated hidden state, denoted by $ht$.
    - At each time step $t$, the hidden state $ht$ is updated based on the current input $xt$ and the previous hidden state $h$t−1 .

- **Output Layer**: Produces the output for each time step, based on the hidden state.

## 2. Mathematical Representation of RNN Computation

- At each time step $t$, the RNN computes the hidden state $h_t$ and output $o_t$ as follows:
    - **Hidden State Update:**

$$h_t = \tanh(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$$

Here:

- $W_h$: Weight matrix between the input and the hidden layer.
- $U_h$: Weight matrix for the recurrent connection in the hidden layer.
- $b_h$: Bias for the hidden layer.
- $h_{t-1}$: Previous hidden state (memory from the last time step).
- $x_t$: Current input.
- $\tanh$: Activation function that introduces non-linearity.

- **Output Calculation:**

$$o_t = \text{softmax}(W_o \cdot h_t + b_o)$$

Here:

- $W_o$: Weight matrix between the hidden layer and the output layer.
- $b_o$: Bias for the output layer.
- $o_t$: Output at time $t$, often a probability distribution if working with classification.

## 3. Feedback Loop and Information Flow

- Each hidden state $ht$ in an RNN depends not only on the current input $xt$ but also on the hidden state from the previous time step $ht−1$ , creating a chain-like dependency across time.
- This feedback loop enables the RNN to retain information about earlier steps in the sequence, allowing it to model temporal dependencies and sequential patterns.

## 4. Unrolling the RNN

- During backpropagation, RNNs are often "unrolled" across the entire sequence to calculate gradients for each time step.
- This unrolled representation makes it easier to visualize the time dependencies, with each "unrolled" copy representing the network's state at a specific time step.

## 5. Limitations of RNNs

- **Vanishing/Exploding Gradients**: RNNs struggle with learning long-term dependencies due to vanishing or exploding gradients during backpropagation through time (BPTT).
- **Short-Term Memory**: Standard RNNs are typically effective for short sequences but face limitations when modeling long-term dependencies, as older information may be overwritten by new input.