Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a "memory" of past inputs. This is achieved by having connections that loop back to previous time steps. In simple terms, RNNs are like a network with feedback loops that enable them to process sequences of data, such as time series, text, or speech.

The key feature of an RNN is its ability to store information from previous steps (hidden state), which is then passed on and updated as the network processes new inputs. This memory of past inputs is what allows RNNs to work well for tasks like predicting the next word in a sentence or forecasting the next value in a time series.

The RNN architecture typically consists of:

1. **Input Layer**: Takes in the current input at each time step.
2. **Hidden Layer**: Updates the hidden state based on the current input and the previous hidden state.
3. **Output Layer**: Generates the output at each time step, based on the hidden state.

The equation for the RNN update at time step t is:

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

Pros:

- Handles sequential data effectively, preserving context from previous steps.
- Flexible input length, useful for varying-length sequences.
- Efficient parameter sharing, reducing the number of parameters compared to traditional networks.
- Good for NLP tasks such as text generation and translation.

Cons:

- Vanishing and exploding gradients can make learning long-range dependencies difficult.
- Struggles with long-term dependencies, even though they theoretically can remember past inputs.
- Training can be complex and slow due to sequential processing and difficulties with gradient stability.

- Limited parallelization during training due to the sequential nature of the network.

RNNs are powerful but have limitations that advanced variants like **LSTMs** and **GRUs** address more effectively.