Introduction to Recurrent Neural Networks

Deep Learning Architecture for Sequential Data

Kiran Bagale

Sept, 2025

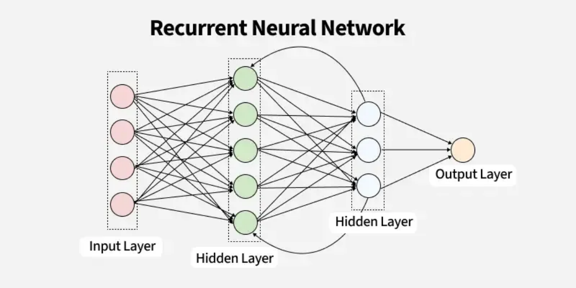
Outline

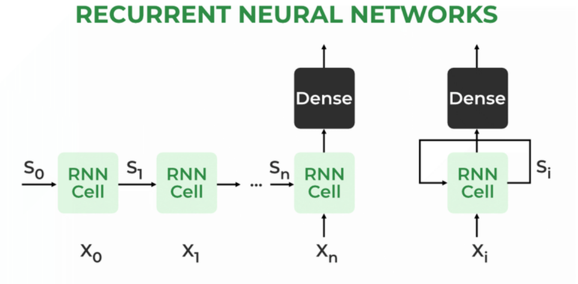
What are Recurrent Neural Networks?

- Recurrent Neural Networks (RNNs) differ from regular neural networks in how they process information
- While standard neural networks pass information in one direction (input to output), RNNs feed information back into the network at each step
- This feedback mechanism enables RNNs to remember prior inputs, making them ideal for tasks where context is important

Key Difference: RNNs have loops that allow information from previous steps to be fed back into the network

RNN


Recurrent Neural Network
Input Layer
Hidden Layer
Hidden Layer
Output Layer


RECURRENT NEURAL NETWORKS
Dense
Dense
S0
RNN Cell
S1
RNN Cell
...
Sn
RNN Cell
RNN Cell
Si
X0
X1
Xn
Xi

RNN vs Feed-Forward Networks

Recurrent Neural Network

- Has feedback loops
- Processes sequential data
- Maintains memory of previous inputs
- Suitable for time-series data

Feed-Forward Neural Network

- No feedback loops
- Processes independent inputs
- No memory mechanism
- Suitable for static classification

Feed-Forward Neural Networks process data in one direction from input to output without retaining information from previous inputs, making them struggle with sequential data since they lack memory.

RNN vs FNN

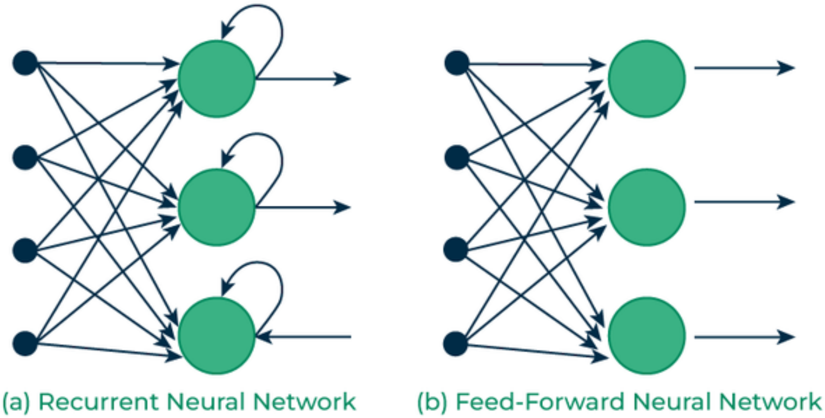


Figure: RNN vs FNN

How RNNs Work

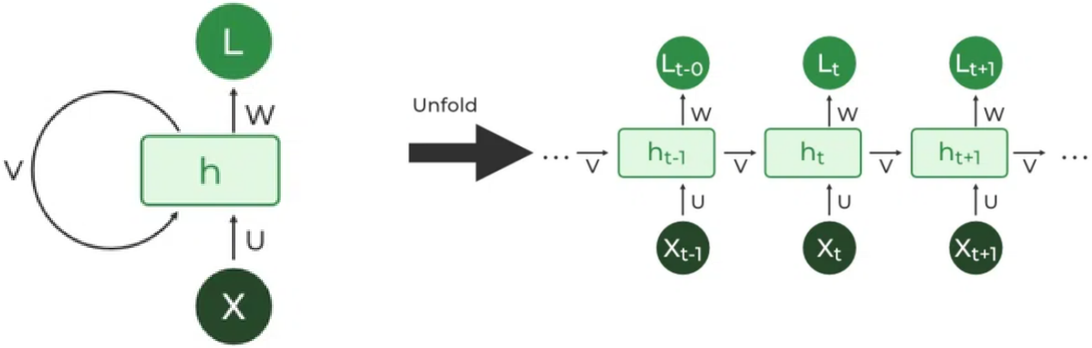
At each time step, RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory, retaining information from previous time steps.

Key Components:

- **Hidden State:** Acts as memory storing past knowledge
- **Input Processing:** Combines current input with previous state
- **Sequential Processing:** Processes data step by step through time

Memory Mechanism: This memory allows the network to store past knowledge and adapt based on new inputs.

RNN working



RNN Mathematical Formulation

1. State Update:

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

Where:

- h_t is the current state
- h_{t-1} is the previous state
- x_t is the input at the current time step

2. Activation Function Application:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

Here, W_{hh} is the weight matrix for the recurrent neuron and W_{xh} is the weight matrix for the input neuron.

3. Output Calculation:

$$y_t = W_{hy} \cdot h_t$$

Hidden State Calculation

1. Hidden State Calculation:

$$h = \sigma(U \cdot X + W \cdot h_{t-1} + B)$$

Where:

- h represents the current hidden state
- U and W are weight matrices
- B is the bias

2. Output Calculation:

$$Y = O(V \cdot h + C)$$

The output Y is calculated by applying O (an activation function) to the weighted hidden state where V and C represent weights and bias.

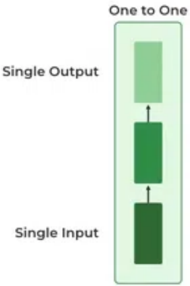
3. Overall Function:

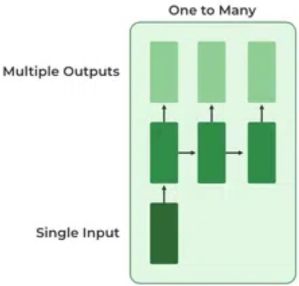
$$Y = f(X, h, W, U, V, B, C)$$

RNN Types Based on Input-Output Structure

There are four types of RNNs based on the number of inputs and outputs in the network:

1. **One-to-One RNN**
2. **One-to-Many RNN**
3. **Many-to-One RNN**
4. **Many-to-Many RNN**





1. One-to-One RNN

- This is the simplest type of neural network architecture
- Single input and single output
- Used for straightforward classification tasks such as binary classification
- No sequential data is involved

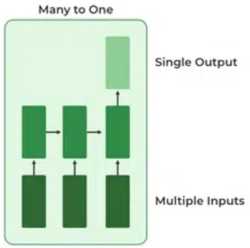
Use Case: Traditional classification problems where input-output relationship is direct and doesn't require sequence processing.

2. One-to-Many RNN

- The network processes a single input to produce multiple outputs over time
- Useful in tasks where one input triggers a sequence of predictions (outputs)
- Example: Image captioning - a single image can be used as input to generate a sequence of words as a caption

Key Feature: One input triggers multiple sequential outputs, making it ideal for generative tasks.

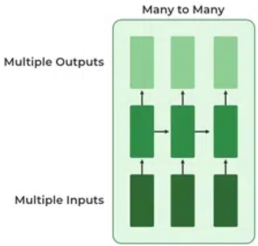
3. Many-to-One RNN



- Receives a sequence of inputs and generates a single output
- Useful when the overall context of the input sequence is needed to make one prediction
- Example: Sentiment analysis - the model receives a sequence of words (like a sentence) and produces a single output like positive, negative, or neutral

Application: Perfect for classification tasks that require understanding the entire sequence context.

4. Many-to-Many RNN



- Processes a sequence of inputs and generates a sequence of outputs
- Most complex RNN type
- Example: Language translation - a sequence of words in one language is given as input and a corresponding sequence in another language is generated as output

Key Advantage: Can handle complex sequence-to-sequence transformations, making it ideal for translation, transcription, and similar tasks.

Variants of Recurrent Neural Networks

There are several variations of RNNs, each designed to address specific challenges or optimize for certain tasks:

1. **Vanilla RNN**
2. **Bidirectional RNNs**
3. **Long Short-Term Memory Networks (LSTMs)**
4. **Gated Recurrent Units (GRUs)**

1. Vanilla RNN

- This simplest form of RNN consists of a single hidden layer where weights are shared across time steps
- Vanilla RNNs are suitable for learning short-term dependencies
- **Limitation:** Limited by the vanishing gradient problem, which hampers long-sequence learning

Challenge: The vanishing gradient problem makes it difficult for vanilla RNNs to capture long-term dependencies in sequences.

2. Bidirectional RNNs

- Process inputs in both forward and backward directions
- Capture both past and future context for each time step
- Ideal for tasks where the entire sequence is available, such as:
  - Named entity recognition
  - Question answering

Advantage: By processing information in both directions, bidirectional RNNs can make more informed predictions using complete sequence context.

3. Long Short-Term Memory Networks (LSTMs)

- Introduce a memory mechanism to overcome the vanishing gradient problem
- Each LSTM cell has three gates:
  - **Input Gate:** Controls how much new information should be added to the cell state
  - **Forget Gate:** Decides what past information should be discarded
  - **Output Gate:** Regulates what information should be output at the current step

Key Benefit: This selective memory enables LSTMs to handle long-term dependencies, making them ideal for tasks where earlier context is critical.

4. Gated Recurrent Units (GRUs)

- Simplify LSTMs by combining the input and forget gates into a single update gate
- Streamline the output mechanism
- Computationally efficient design
- Often perform similarly to LSTMs
- Useful in tasks where simplicity and faster training are beneficial

Trade-off: GRUs offer a good balance between performance and computational efficiency compared to LSTMs.

Backpropagation Through Time (BPTT)

Since RNNs process sequential data, **Backpropagation Through Time (BPTT)** is used to update the network's parameters.

The loss function $L(\theta)$ depends on the final hidden state h_3 and each hidden state relies on preceding ones forming a sequential dependency chain:

$$h_3 \text{ depends on } h_2, \quad h_2 \text{ depends on } h_1, \quad \ldots, \quad h_1 \text{ depends on } h_0$$

In BPTT, gradients are backpropagated through each time step. This is essential for updating network parameters based on temporal dependencies.

Backpropagation Through Time (BPTT)

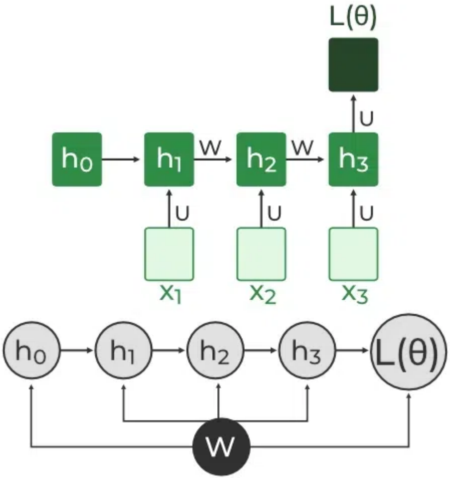


Figure: Backpropagation Through Time (BPTT) in RNN

1. Simplified Gradient Calculation:

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial W}$$

2. Handling Dependencies in Layers: Each hidden state is updated based on its dependencies:

$$h_3 = \sigma(W \cdot h_2 + b)$$

The gradient is then calculated for each state, considering dependencies from previous hidden states.

3. Gradient Calculation with Explicit and Implicit Parts: The gradient is broken down into explicit and implicit parts summing up the indirect paths from each hidden state to the weights:

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2^+}{\partial W}$$

Final Gradient Expression

4. Final Gradient Expression: The final derivative of the loss function with respect to the weight matrix W is computed:

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

This iterative process is the essence of backpropagation through time.

RNN Unfolding: The process of expanding the recurrent structure over time steps enables BPTT, where each step of the sequence is represented as a separate layer in a series, illustrating how information flows across each time step.

Advantages of Recurrent Neural Networks

- **Sequential Memory:** RNNs retain information from previous inputs making them ideal for time-series predictions where past data is crucial
- **Enhanced Pixel Neighborhoods:** RNNs can be combined with convolutional layers to capture extended pixel neighborhoods improving performance in image and video data processing

Key Strength: The ability to maintain context across sequences makes RNNs particularly powerful for temporal pattern recognition.

Limitations of RNNs

While RNNs excel at handling sequential data they face two main training challenges:

1. **Vanishing Gradient:**
   - During backpropagation gradients diminish as they pass through each time step
   - Leads to minimal weight updates
   - Limits the RNN's ability to learn long-term dependencies
   - Crucial for tasks like language translation

2. **Exploding Gradient:**
   - Sometimes gradients grow uncontrollably
   - Causes excessively large weight updates
   - Destabilizes training

These challenges can hinder the performance of standard RNNs on complex, long-sequence tasks.

Applications of Recurrent Neural Networks

RNNs are used in various applications where data is sequential or time-based:

- **Time-Series Prediction:** RNNs excel in forecasting tasks, such as stock market predictions and weather forecasting
- **Natural Language Processing (NLP):** RNNs are fundamental in NLP tasks like language modeling, sentiment analysis and machine translation
- **Speech Recognition:** RNNs capture temporal patterns in speech data, aiding in speech-to-text and other audio-related applications
- **Image and Video Processing:** When combined with convolutional layers, RNNs help analyze video sequences, facial expressions and gesture recognition

Implementing a Text Generator Using RNNs

Project Overview:

- Create a character-based text generator using Recurrent Neural Network (RNN) in TensorFlow and Keras
- Implement an RNN that learns patterns from a text sequence to generate new text character-by-character

Key Learning Objective: Understanding how RNNs can model sequential dependencies in text data and generate coherent text based on learned patterns.

Technical Approach: The RNN processes text sequences character by character, learning the probability distribution of the next character given the previous characters in the sequence.

Summary

- RNNs are powerful neural networks designed for sequential data processing
- They use feedback loops to maintain memory of previous inputs
- Various RNN types (One-to-One, One-to-Many, Many-to-One, Many-to-Many) serve different purposes
- Advanced variants like LSTMs and GRUs address the vanishing gradient problem
- RNNs have wide applications in NLP, time-series prediction, speech recognition, and more
- Understanding BPTT is crucial for training RNNs effectively

Key Takeaway: RNNs bridge the gap between static neural networks and dynamic, context-aware systems capable of understanding temporal patterns in data.

Thank You!

Questions?