Architecture of RNNs: From Vanilla RNNs to GRUs and LSTMs

Atharv Bhatt

January 29, 2026

Contents

1 Introduction

Recurrent Neural Networks (RNNs) are a class of neural networks designed for **sequential data**. Unlike feedforward networks, RNNs have loops, allowing information to persist across time steps. They are widely used in:

- Natural Language Processing (e.g., text generation, translation)
- Time Series Prediction (e.g., stock prices, weather forecasting)
- Speech Recognition

However, vanilla RNNs face limitations such as the **vanishing gradient problem**, which gave rise to more advanced architectures like **GRUs** and **LSTMs**.

2 Vanilla RNN Architecture

2.1 Structure

At each time step t, an RNN takes:

- Input vector x_t
- Previous hidden state h_{t-1}

The hidden state is updated as:

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

The output y_t is computed as:

$$y_t = W_{hy}h_t + b_y \tag{2}$$

2.2 Limitations: Vanishing/Exploding Gradients

When training with **Backpropagation Through Time (BPTT)**, gradients can vanish or explode:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial h_t} \prod_{k=1}^{t} W_{hh}^T \cdot \text{diag}(\tanh'(h_k)) \tag{3}$$

Repeated multiplication of small derivatives < 1 leads to vanishing gradients, making it difficult to learn long-term dependencies.

3 Gated Architectures: GRU and LSTM

To solve RNN limitations, **gated mechanisms** were introduced.

3.1 GRU (Gated Recurrent Unit)

GRU simplifies LSTM by combining forget and input gates into an **update gate**.

3.1.1 Equations

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \quad \text{(update gate)} \tag{4}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \quad \text{(reset gate)} \tag{5}$$
$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1})) \quad \text{(candidate state)} \tag{6}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad \text{(final hidden state)} \tag{7}$$

3.2 LSTM (Long Short-Term Memory)

LSTM introduces **three gates** and a **cell state** C_t to store long-term memory explicitly.

3.2.1 Anatomy of LSTM Cell

- **Cell state** C_t: conveyor belt running through time
- **Hidden state** h_t: output at current time
- **Gates**: control information flow (forget, input, output)

3.2.2 LSTM Gates

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad \text{(forget gate)} \tag{8}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad \text{(input gate)} \tag{9}$$
$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad \text{(candidate cell state)} \tag{10}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(updated cell state)} \tag{11}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad \text{(output gate)} \tag{12}$$
$$h_t = o_t \odot \tanh(C_t) \quad \text{(hidden state update)} \tag{13}$$

3.3 Comparison: RNN vs GRU vs LSTM

Feature	**Vanilla RNN**	**GRU**	**LSTM**
Memory mechanism	Hidden state only	Hidden state + gates	Hidden state + cell state + gate
Gates	None	2 (update	reset)
3 (input, forget, output)			
Vanishing gradient	Severe	Better	Best
Parameters	Fewest	Moderate	Most
Training speed	Fast	Faster than LSTM	Slower than GRU
Use case	Short sequences	Medium sequences	Long sequences

Table 1: Comparison of RNN architectures

4 Vanishing Gradient Problem

Vanilla RNNs struggle with long-term dependencies. Gradient over time steps:

$$\frac{\partial h_T}{\partial h_0} = \prod_{t=1}^{T} W_{hh}^T \cdot \text{diag}(\tanh'(h_t)) \tag{14}$$

If $|W_{hh}| < 1$, gradients vanish. LSTM/GRU gates and cell states mitigate this by preserving gradients over long sequences.

5 Conclusion

- RNNs are fundamental for sequence modeling but limited by vanishing gradients.
- GRUs offer a simpler gated mechanism suitable for medium-length sequences.
- LSTMs provide robust long-term memory handling via gates and cell states.
- Understanding forget, input, and output gates is crucial for designing sequential models.