The Architecture of Recurrent Neural Networks

Parth Agrawal

January 2026

1 Introduction to Recurrent Neural Networks

Recurrent Neural Networks are used for sequential data. When we need to predict the next term based on previous patterns, we use RNNs, which operate on sequences of vectors rather than single vectors. This makes them suitable for solving problems such as text generation and weather forecasting.

RNNs use the entire input history to process data, unlike conventional networks, which produce a single output per data point. This allows them to grasp complex patterns hidden in the data itself.

The gradients trained during these RNNs give it a sort of "Memory"- like understanding of the data. They perform this by maintaining a `hidden state` vector, which is updated with each input and passed along with every input.

Rather than Fixed Sequences of Feed-Forward Networks, RNNs use Variable Sequences, which are not limited by the size of the input shape.

The core update equation for a simple RNN:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$$

Where: * h_t: Current hidden state * h_{t-1}: Previous hidden state * x_t: Input at current step * W: Weight matrices

2 Vanishing Gradient Problem

The Vanishing Gradient Problem can be understood as repeated multiplications causing the Gradient Size to grow too large or too small.

Since every neuron participates in calculating the cost during backpropagation, the same weights are multiplied multiple times, increasing the gradient size and losing information from previous layers. We can see it as the network "forgetting" older data.

The problem lies at the core of backpropagation through neural networks, and, due to the unique issue in RNNs, it cannot be solved by other techniques like

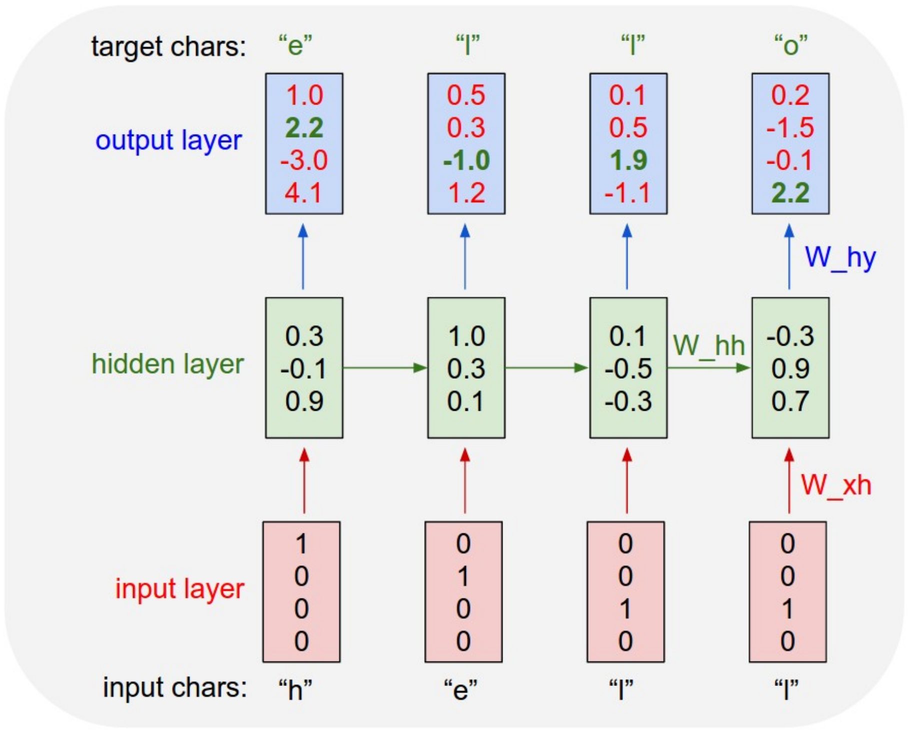

Figure 1: An Example RNN with 4-dimensional input and output layers

Residual Networks, He initialisation, or Gradient Clipping, since the problem in RNNs is a loss of information.

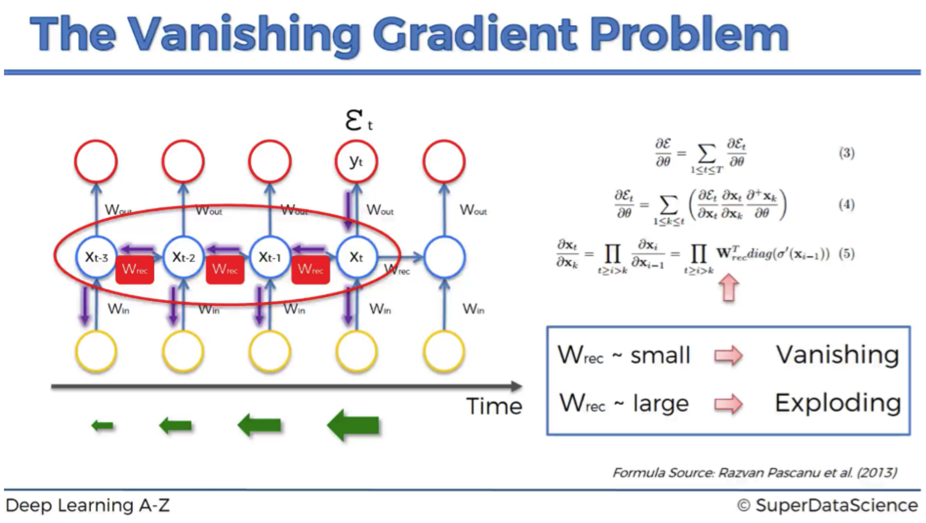
The Vanishing Gradient Problem
$\mathcal{E}_t$
y_t
W_{out}
X_{t-3}
X_{t-2}
X_{t-1}
X_t
W_{rec}
W_{in}
Time
$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta}$ (3)
$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$ (4)
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$ (5)
W_{rec} ~ small ⇨ Vanishing
W_{rec} ~ large ⇨ Exploding
Formula Source: Razvan Pascanu et al. (2013)
Deep Learning A-Z
© SuperDataScience

Figure 2: Vanishing Gradient Problem

3 Gated Architectures (GRU & LSTM)

LSTMs and GRUs solve the vanishing gradient problem. They have internal mechanisms, known as gates, to regulate the flow of information.

In essence, these gates can learn to distinguish between important and unimportant data in a sequence. This allows them to understand and perform much better at problems like text generation.

LSTM: An LSTM has a similar control flow to an RNN. The difference is in the operations within it. It uses a forget gate and an input gate to compute the cell state, which is then used to produce the next hidden state via the output gate.

GRU: A GRU is a simplified version of the LSTM network that combines the forget and input gates into a single Update gate.

4 Anatomy of an LSTM Cell

We will now cover in detail the gates and other important features of LSTM.

4.1 Forget Gate

This gate decides which information to keep and which to throw away. Information from the hidden state and the current input is passed through a sigmoid

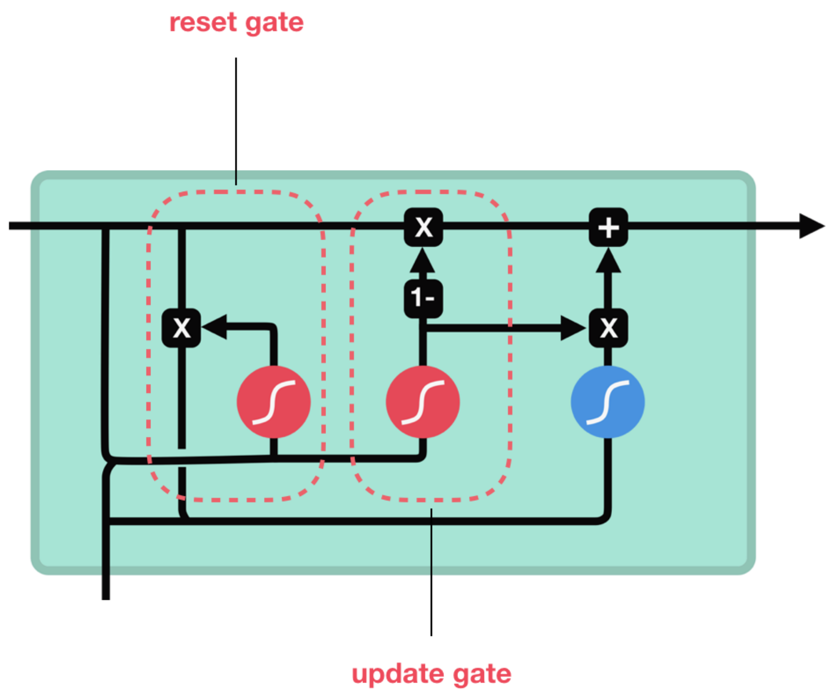

Figure 3: GRU Architecture

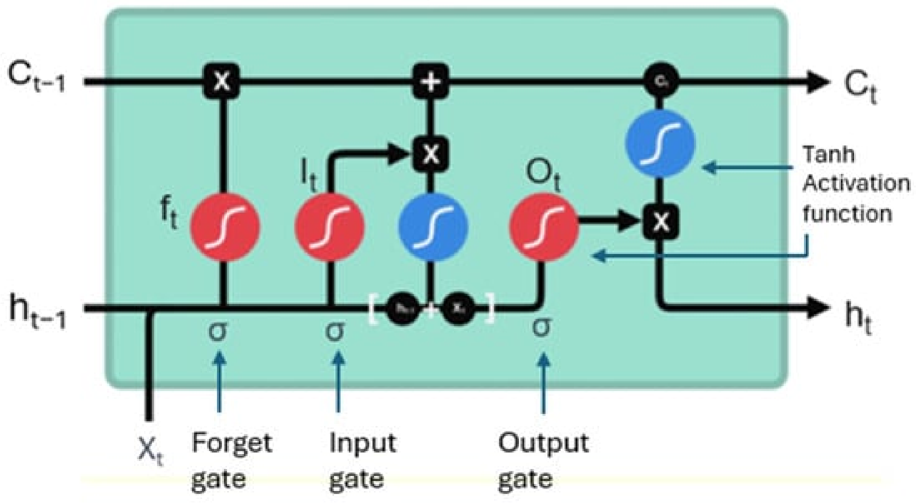

Figure 4: LSTM Architecture

function, which determines whether to forget (0) or retain (1). We then multiply this by the previous cell state.

4.2 Input gate

In this gate, to update the cell state, we pass the hidden state and the input through both sigmoid and tanh, then multiply the results and add them to the previous cell state. This gives the new cell state. Here, the sigmoid again decides which information to keep.

4.3 Cell State

As mentioned previously, the previous cell state is multiplied by the forget gate value and added to the input gate values. This gives the new cell state.

4.4 Output Gate

The output gate determines the next hidden state. First, we pass the previous hidden state and current input to a sigmoid function. Then we pass the new cell state through a tanh function. We multiply these outputs to form the new hidden state. We pass the new cell state and new hidden state forward to the next cell.

4.5 In GRU

In GRU, we only have -

- **Reset Gate:** A single sigmoid gate to decide what information to keep.
- **Update Gate:** A gate consisting of a sigmoid and a tanh function to update the cell hidden state.

5 References

[1] Karpathy, A. (2015, May 21). The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/ [2] SuperDataScience Team. (2018, August 23). Recurrent Neural Networks (RNN) – The vanishing gradient problem. SuperDataScience. https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem/ [3] Phi, M. (2018, September 24). Illustrated guide to LSTM's and GRU's: A step by step explanation. Medium. https://medium.com/data-science/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21/