Encoding Time: A Comparison of Temporal Representation in RNNs, LSTMs and Transformers

Youssef Soula
Engineering Student at Sup'Com

August 2025

1 Abstract

This paper explores how Recurrent Neural Networks (RNNs), Long-Short-Term Memory Networks (LSTMs), and Transformers encode temporal information in sequential data. While RNNs and LSTMs have been the standard architectures for time-dependent tasks for a long time, Transformers, originally designed for NLP, have recently shown high effectiveness for sequence modeling. In this paper, I presented a theoretical overview of each model's internal representation of time, followed by comparative experiments using synthetic tasks that require memory and sensitivity to time. The results highlight differences in how these architectures preserve, retrieve, and exploit temporal signals, offering insight into their respective strengths and limitations.

2 Introduction

Temporal information is important for many machine learning tasks, from language modeling to time series forecasting. Capturing the order, delay, and duration of events within sequences remains a challenge. Over the past decades, several architectures have been proposed to address this problem, especially Recurrent Neural Networks (RNNs), Long-Short-Term Memory networks (LSTMs), and, more recently, Transformers.

RNNs process sequences one element at a time, maintaining a hidden state that theoretically encodes the full history. However, issues such as vanishing/exploding gradients limit their capacity to store long-range dependencies. LSTMs handle this by introducing gating mechanisms that regulate information flow across time steps, enabling longer memory spans. Transformers, however, cut with recurrence entirely, using attention mechanisms to model relationships between sequence elements, but they rely on positional encodings to provide any notion of order.

This paper investigates the different strategies these models use to represent and reason about time. It aims to provide a clear theoretical comparison and evaluate each model on specifically designed tasks that isolate their temporal capabilities.

3 Mathematical Modelization of the Three Models

3.1 Recurrent Neural Networks (RNNs)

RNNs update their hidden state at each time step using the equation:

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

Where:

- x_t is the input at time step t
- h_t is the hidden state at time t
- W_{xh}, W_{hh} are weight matrices
- b_h is a bias term
- ϕ is a non-linear activation function (typically tanh or ReLU)

The hidden state h_t carries information from previous time steps, but recursive gradient computations can lead to vanishing or exploding gradients for long sequences.

3.2 Long Short-Term Memory Networks (LSTMs)

LSTMs introduce a memory cell c_t and gating mechanisms to control information flow:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) && \text{(forget gate)} \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) && \text{(input gate)} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) && \text{(output gate)} \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) && \text{(candidate cell state)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(cell state update)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(hidden state)}
\end{aligned}$$

Where:

- σ is the sigmoid function
- $\odot$ denotes element-wise multiplication
- W_*, U_*, and b_* are learnable parameters

3.3 Transformers

Transformers use self-attention rather than recurrence. The core attention mechanism is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- $Q, K, V \in \mathbb{R}^{T \times d_k}$ are the query, key, and value matrices
- d_k is the dimension of the keys

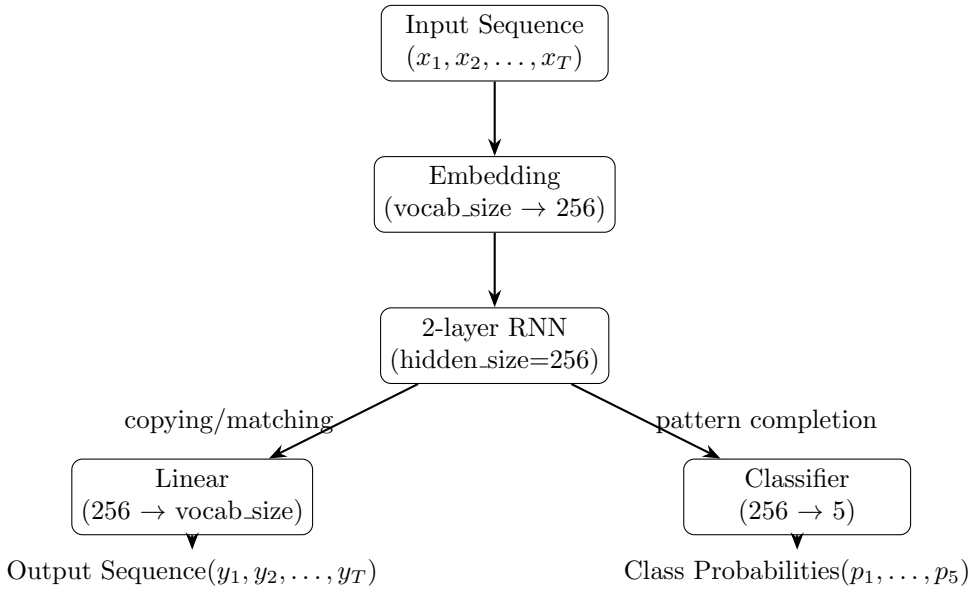
To encode positional information, a positional encoding is added to the input embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

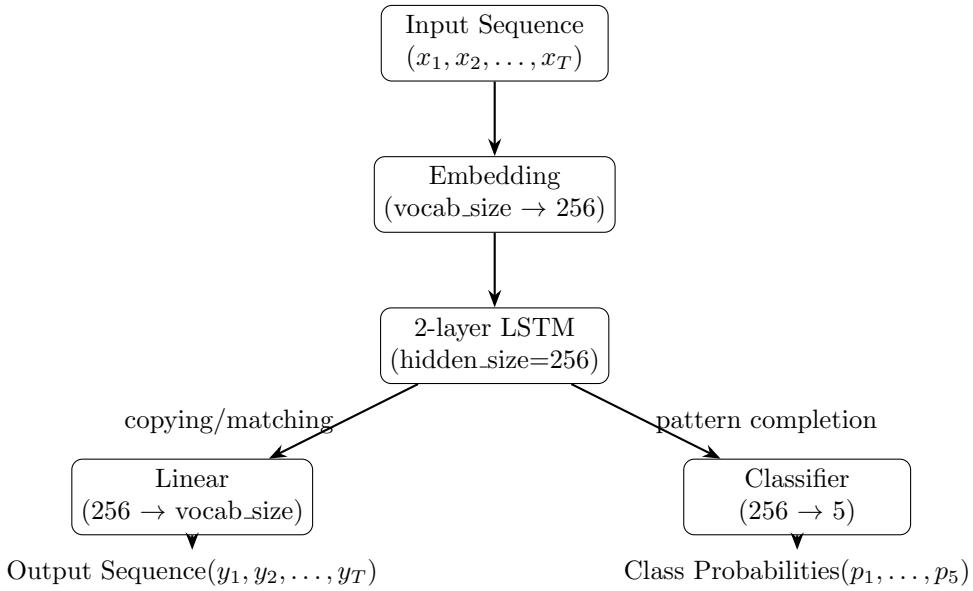
4 Model Architectures

In this section, I described the architecture used for each of the three models: RNN, LSTM, and Transformer. All models were implemented in PyTorch and designed to handle both sequence prediction (copying, matching) and classification (pattern completion) tasks with minimal changes.

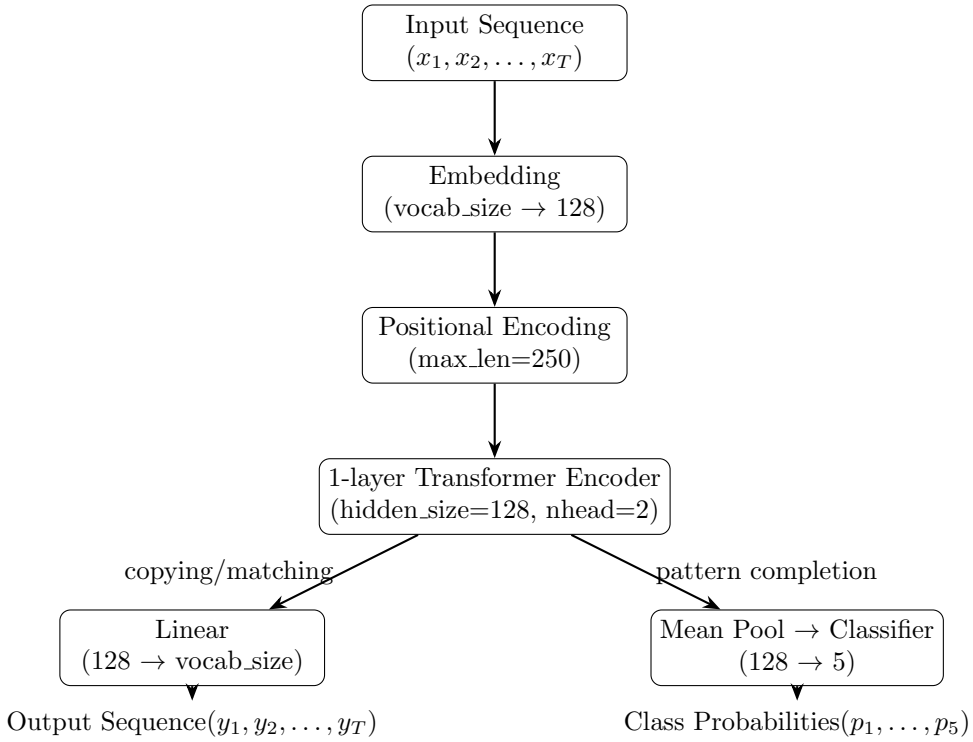
4.1 Recurrent Neural Network (RNN)



4.2 Long Short-Term Memory (LSTM)



4.3 Transformer Encoder



Each model was trained using cross-entropy loss on the appropriate output (sequence or class), with slight architectural differences to support both generation and classification tasks.

5 Experiment Overview

To compare the temporal representation capabilities of RNNs, LSTMs, and Transformers, I have designed synthetic tasks that isolate time-sensitive reasoning. My goal is to evaluate each model's ability to hold, retrieve, and compare temporal information without interference from real-world noise. I focused on three diagnostic experiments:

- The Copying Test, which evaluates long-term memory retention.
- The Matching Test, which assesses the ability to compare temporally separated symbols.
- The Pattern Completion Test, which targets pattern learning.

Each task uses controlled sequences with specific structures, allowing us to analyze the temporal representation mechanisms without interference from natural language semantics or noise in real-world data.

5.1 Copying Test

Objective: Assess a model's capacity to memorize and reproduce a sequence after a delay.
Setup:

- The input consists of:
  - A prefix of L random tokens from a vocabulary of size V
  - A delay of D blank (zero) tokens
  - A trigger token at the end
- The model must output the original prefix that starts right after the trigger.

Example (with $L = 5$, $D = 10$):

$$\text{Input: } [x_1, x_2, x_3, x_4, x_5, 0, 0, \ldots, 0, \texttt{trigger}]$$

$$\text{Target: } [0, 0, \ldots, 0, x_1, x_2, x_3, x_4, x_5]$$

5.2 Matching Test

Objective: Evaluate the model's ability to remember a token presented at the beginning of the sequence and to recall it after a temporal delay when encountering a query signal.

Setup:

- Each input sequence begins with a single token, followed by a sequence of random noise tokens.
- At the end of the sequence, a special query token (e.g. 0) indicates the time to recall the initial token.
- The model must output the initial token at the position of the query marker; all other positions in the target sequence are set to zero.

Example (with delay = 5):

$$\text{Input: } [x, n_1, n_2, n_3, n_4, n_5, \texttt{0}]$$

$$\text{Target: } [0, 0, 0, 0, 0, 0, x]$$

Where x is the token to remember and n_i are random noise tokens.

5.3 Pattern Completion Test

Objective: Assess the model's ability to complete short, locally structured patterns.

Setup:

- Each input is a short sequence (typically 5–10 tokens) that follows a simple, deterministic local pattern.
- Common structures include repeating bigrams, trigrams, or alternating values.
- The model must predict the next token in the sequence based solely on recent context (the last 1 to 3 tokens).

Examples:

- Input: $[A, B, A, B, A,\ ?]$ Output: B
- Input: $[1, 2, 1, 2, 1,\ ?]$ Output: 2
- Input: $[A, A, B, B, A, A,\ ?]$ Output: B

6 Experiment Results

I trained RNN, LSTM, and Transformer models on each of the three tasks described above for 50 epochs. Below is a summary of the training dynamics observed through the training loss curves.

6.1 Copying Task

As expected, the vanilla RNN struggled during early epochs due to vanishing gradients but eventually converged. The LSTM and Transformer, both equipped to handle long-range dependencies, achieved low loss values rapidly within the first 10 epochs. The Transformer had a slight edge in convergence speed compared to LSTM.

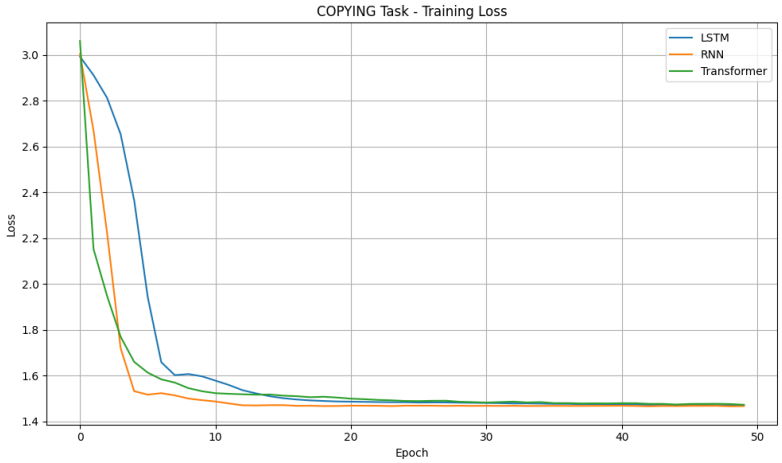


Figure 1: Training loss on the Copying Task for RNN, LSTM, and Transformer.

6.2 Matching Task

All three models performed well, with rapid convergence observed across the board. The Transformer achieved near-zero loss marginally faster than RNN and LSTM, thanks to its direct attention mechanism. However, the overall difference in performance was minimal, given the task's low memory complexity.

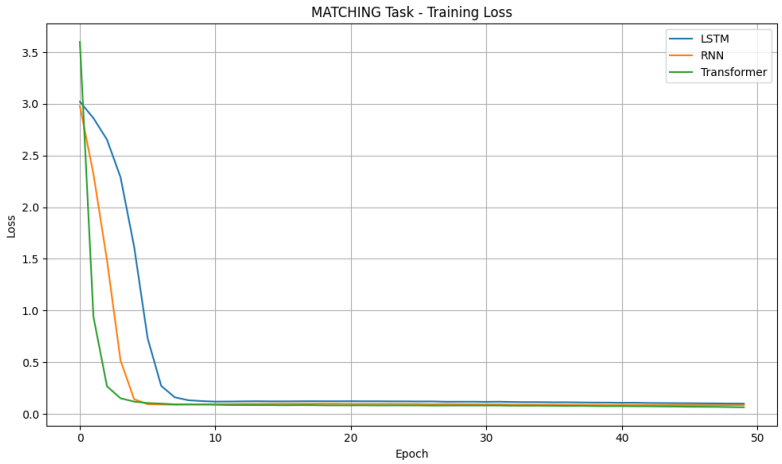


Figure 2: Training loss on the Matching Task for RNN, LSTM, and Transformer.

6.3 Pattern Completion Task

The RNN and LSTM performed best, quickly learning the underlying bigram/trigram patterns. The Transformer, however, showed higher loss and noisy convergence. This is consistent with the inductive bias of Transformers, which are not inherently optimized for local sequential patterns without significant training data or architectural tuning. These results highlight the importance of model inductive biases relative to task structure.

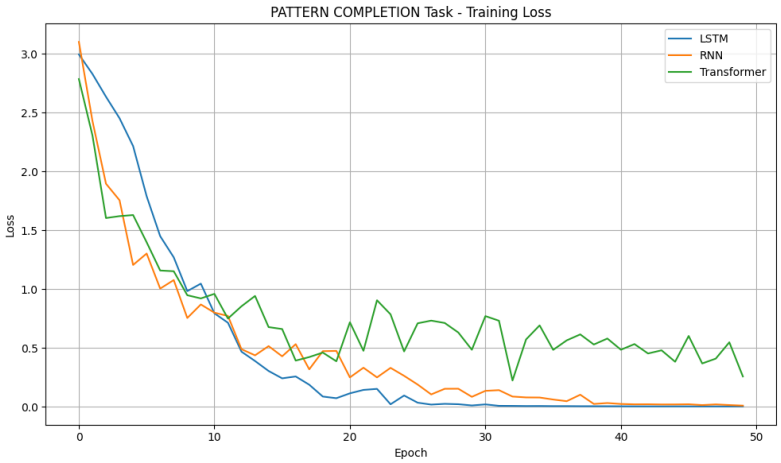


Figure 3: Training loss on the Pattern Completion Task for RNN, LSTM, and Transformer.

Overall: LSTM was the most robust across all tasks, while the Transformer excelled in long-term dependencies but lagged in local pattern completion. RNN was competitive in short-memory tasks but slower to converge in more complex settings.

7 Conclusion:

The theoretical analysis demonstrates strong correspondence with the practical experimental results, validating how RNNs, LSTMs, and Transformers encode temporal information differently and how these differences impact performance on sequence tasks. This alignment between theory and practice reinforces the understanding of each model's unique capabilities in handling time-dependent data.

Each architecture plays an important role in tackling real-world challenges: RNNs excel in modeling simpler, shorter sequences; LSTMs effectively manage longer dependencies through gating mechanisms; and Transformers offer powerful solutions for complex, long-range dependencies with their self-attention approach.

Looking ahead, I plan to explore novel LSTM variants aimed at optimizing their efficiency and memory capabilities, as well as to dive deeper into the self-attention mechanism to better understand how it processes temporal data from a totally different perspective. These future directions may lead to improved models that combine the strengths of existing architectures to advance temporal sequence learning.