# Large Vocabulary Handling in Recurrent Neural Networks Enhanced by Positional Encoding


## Abstract

This research presents a counterintuitive discovery: positional encoding, a high-dimensional representation of time indices on input data, improves the learning capabilities of recurrent neural networks (RNNs). Although positional encoding is widely recognized for complementing Transformer neural networks by enabling them to process data order, its application to RNNs seems unnecessary because RNNs inherently encode temporal information. However, our analysis using synthetic benchmarks shows that combining positional encoding with RNNs offers advantages, especially when dealing with extensive vocabularies that include low-frequency tokens. Further investigation reveals that these infrequent tokens cause instability in the gradients of standard RNNs, and positional encoding helps to mitigate this instability. These findings highlight a new function of positional encoding beyond its well-known role as a timekeeping mechanism for Transformers.


## 1 Introduction

Since their introduction, Transformer neural networks have become the preferred method for processing and generating time series data, surpassing traditional recurrent neural networks (RNNs). A significant difference between these models is their handling of temporal information, that is, the sequence of data points or tokens. RNNs process temporal information by adjusting their internal state based on new inputs and their existing state. Conversely, Transformers lack an intrinsic mechanism for understanding data sequence order and, therefore, depend on an external system known as positional encoding to keep track of time.

Positional encoding represents time indices in a high-dimensional format. A common method involves using sinusoidal waves of predetermined frequencies. This method marks input tokens by adding or appending these vectors to the input embeddings. Unlike RNNs, positional encoding's time representation remains constant regardless of input values until processed by a network.

Although positional encoding is often viewed as a way to represent time that can replace RNNs when used with Transformers, it is not incompatible with RNNs. Inputs to RNNs can be augmented with position-encoding vectors. Autonomous activities in biological neurons, such as oscillations, are believed to be important for time perception and other perceptual processes, as well as motor control.

This study, therefore, investigates the effects of adding positional encoding to the inputs of RNNs, using synthetic benchmarks. The results demonstrate that positional encoding helps RNNs manage a more extensive range of discrete inputs, or a larger vocabulary, compared to those without positional encoding.

The key contributions of this research are outlined below:

- It illustrates the challenges faced when training RNNs on large vocabularies using carefully designed benchmark tasks, a problem that has not been widely recognized or addressed in previous research, despite its potential impact on practical applications.

.

- It explains that the difficulties in training RNNs with extensive vocabularies are due to gradient instability caused by infrequent tokens, which inevitably occur as vocabulary size increases.

- It introduces a novel use of positional encoding, beyond its typical role in timing for Transformers, by integrating it with RNNs. It shows that positional encoding helps alleviate issues related to large vocabularies by stabilizing RNN gradients against the disruptions caused by infrequent tokens.

## 2 Related Studies

### 2.1 Theoretical and Empirical Computational Power of (Vanilla) RNNs

Mathematically, RNNs are recognized as being Turing-complete, capable of simulating Turing machines if their weights are infinitely precise and perfectly tuned. In practice, however, RNN weights are limited by finite precision and the need to optimize based on a finite set of observations. These constraints impose practical limitations on the capabilities of RNNs. For instance, empirical RNNs cannot store an infinite number of observations in their memory, and the memorized information tends to degrade over time.

More recently, research into extending memory retention has explored continuous-time models. Instead of modifying a latent state in discrete-time steps, these models use a linear combination of orthogonal polynomials in a continuous-time domain to approximate the input history. The coefficients of these polynomials provide a finite-dimensional representation of the input sequence, known as the High-Order Polynomial Projection Operator (HiPPO), and the dynamics of these coefficients can be described by an ordinary differential equation (ODE). This concept has been further developed into neural state-space models by replacing the fixed state matrix in the ODE with a learnable one, constrained to a diagonal structure plus a row-rank matrix. With additional enhancements, the latest state-space models have shown language modeling performance that rivals Transformer-based models.

### 2.2 Positional Encoding

Positional encoding serves as a high-dimensional representation of the temporal structures present in input data. This method is particularly crucial for Transformers, which, unlike RNNs, do not inherently capture the order of inputs. Therefore, input tokens to a Transformer are "time-stamped" by adding or concatenating a position-encoding vector.

In the initial implementation of the Transformer, token positions were represented using sinusoidal waves of various predefined frequencies. Although this method is effective for a wide range of tasks, researchers have explored other encoding schemes as well. For instance, the well-known BERT pretraining for natural language processing used learnable embeddings to indicate token positions. Some studies have suggested that combining sinusoidal and learnable encodings can enhance model performance. Another approach is to encode the distance between tokens instead of the time elapsed from the sequence's beginning.

Beyond Transformers, positional encoding is used to indicate elapsed time in diffusion processes. Its effectiveness is not limited to temporal information; studies on three-dimensional mesh and point-cloud modeling have shown that sinusoidal transformation of spatial data outperforms raw coordinate representation.

Despite its widespread use across various areas of machine learning, the application of positional encoding to pure RNNs has been largely unexplored. To the author's knowledge, only a few studies have investigated position-encoded RNNs. The time index in time series data has rarely been directly used by RNNs, likely due to perceived redundancy alongside RNN functionalities.

# 3 Methods

## 3.1 Task

The impact of positional encoding on RNNs was examined using a reverse-ordering task. In this task, RNNs were trained to reconstruct a sequence of random integers in reverse order (e.g., given 8, 29, 2, 11, the output should be 11, 2, 29, 8).

## 3.2 Model Architecture

This study's investigations were based on single-layer gated recurrent units (GRUs), long short-term memory (LSTM) networks, and a neural state-space model, S4D. Each integer in the input sequences was first embedded, concatenated with its positional encoding, and then fed into the RNN or S4D. After processing the entire input sequence, the network received a command to produce the output, represented by a time-invariant learnable vector. The outputs from the RNN or S4D module were linearly projected into classification logits, and the cross-entropy loss against the target sequence was used to optimize the entire network. Model predictions during testing were determined by the argmax of these logits for each time step.

The canonical sinusoidal positional encoding used for Transformers was adopted in this study. Specifically, each time step $t$ was encoded by a $D_{pos}$-dimensional vector, $(PE_{t,1}, ..., PE_{t,D_{pos}})^T$, defined as follows:

$$PE_{t,2i} := \sin\left(\frac{t-1}{10000^{\frac{2(i-1)}{D_{pos}}}}\right) \quad (1)$$

$$PE_{t,2i+1} := \cos\left(\frac{t-1}{10000^{\frac{2(i-1)}{D_{pos}}}}\right) \quad (2)$$

For learning stability, the positional encoding was normalized by dividing it by $\sqrt{D_{pos}/2}$, ensuring the encoding vectors had a unit $L_2$-norm. The time step $t$ incremented throughout both input and output phases (i.e., $t = 1, ..., L, L+1, ..., 2L$, where $L$ is the input length), without any hard-coded link between input and output positions.

## 3.3 Implementation Details

Across the experiments, the dimensionality of the hidden layer of the RNNs was set to 512. The embedding of the input integers and the memory cell of the LSTM also had the same dimensionality of 512. Similarly, the hidden dimensionality of S4D was set to 512, while its state size (or the order of the Legendre polynomials) was maintained at the default value of 64.

The models were trained for 300,000 iterations using the Adam optimizer with parameters $(\beta_1, \beta_2) :=$ (0.9, 0.999) and no weight decay. The learning rate was linearly warmed up from 0.0 to 0.001 for the first 1,000 iterations, and then annealed according to the cosine schedule. The batch size was 512.

All experiments were implemented in PyTorch (ver. 2.1.1).

# 4 Results

## 4.1 Key Findings

Positional encoding improved the ability of RNNs to handle a larger vocabulary in the reverse-ordering task. The position-encoded GRU and LSTM successfully reversed input sequences of 64 integers drawn uniformly at random from vocabularies of size 32-256 and 256-16,384, respectively, achieving token-wise accuracy above 95%. In contrast, the performance of the vanilla models without positional encoding degraded as the vocabulary size increased. Similarly, positional encoding enhanced the capacity of S4D to handle large vocabularies. These improvements are also evident in the reduced sequence-wise reconstruction errors, measured by the Damerau-Levenshtein distance. Neither extra training iterations nor greater batch sizes improved the performance of the vanilla models.

### 4.2 Frequency Matters

The most apparent consequence of the increased vocabulary size was the reduced chance of observing individual vocabulary items. Accordingly, additional experiments were conducted with non-uniformly distributed tokens to investigate the relation between their frequency and RNN performance. Specifically, the input vocabulary was evenly divided into Frequent and Rare groups, and the Frequent tokens had three times the probability of the Rare tokens.

The training data consisted of 64 independent samples from this dual-frequency vocabulary. By contrast, the test data were systematically constructed so that each sequence included a single "target" token (Frequent/Rare) whose retrieval was evaluated for accuracy assessment, along with 63 "disturbants" that were either all Frequent or all Rare. The experiment revealed that it was the disturbant tokens whose frequency significantly impacted the performance of the vanilla RNNs and S4D. On the one hand, the Rare targets were successfully retrieved as long as they were surrounded by the Frequent disturbants. On the other hand, the vanilla GRU struggled to recover the Frequent targets when the other input tokens were filled with the Rare disturbants. The LSTM performance was also degraded, especially when the targets were positioned in the first quarter of the input sequence (1 $\leq$ t $\leq$ 16). Similarly, the Rare disturbants were detrimental to the S4D; unlike the RNNs, however, the accuracy was worst when the targets were located in the middle of the input sequences (17 $\leq$ t $\leq$ 32).

In contrast, the position-encoded RNNs exhibited robustness to the frequency of the target and disturbant tokens. They achieved nearly perfect accuracies in most cases, except when the GRU processed the fully Rare data whose target was located in the first half of the sequence (1 $\leq$ t $\leq$ 32). Likewise, positional encoding enhanced the resilience of the S4D against the influence of Rare disturbants.

### 4.3 Analysis of Gradient Stability

To delve deeper into the influence of token frequency on RNN performance, the gradients of the RNN latent states were scrutinized. In the analysis, pairs of input sequences were processed by the RNNs trained on the dual-frequency vocabulary (comprising Frequent and Rare items). Each pair of sequences shared the same initial token (t = 1; "target") but varied in the subsequent tokens (2 $\leq$ t $\leq$ L; "disturbants"). Then, gradients were computed for the distant mapping between the first and last updated states (i.e., at time t = 1 and 2L) of the RNNs using backpropagation through time. The stability of RNN learning was assessed by measuring the dot-product similarity of the gradients between the paired input sequences (after normalization over output dimensions).

Formally, the paired input sequences, denoted as A and B, established two distinct, but ideally similar mappings, $f^{(A)}$ and $f^{(B)}$, from the first to the last latent state of the RNNs ($\tilde{h}_{2L}^{(s)} = f^{(s)}(\tilde{z}_1)$, where $s \in \{A, B\}$). The gradient stability of the RNNs was defined by the dot-product similarities between the normalized gradients of these paired mappings:

$$\text{Stability}(A,B) := \sum_{i=1}^{D} \langle \alpha_i^{(A)} \nabla f_i^{(A)}(\tilde{z}_1), \alpha_i^{(B)} \nabla f_i^{(B)}(\tilde{z}_1) \rangle = \sum_{i=1}^{D} \alpha_i^{(A)} \alpha_i^{(B)} \left( \frac{\partial h_{2L,i}^{(A)}}{\partial z_{1,j}} \cdot \frac{\partial h_{2L,i}^{(B)}}{\partial z_{1,j}} \right) \tag{1}$$

where the coefficients $\alpha_i^{(s)}$ normalized the raw gradients $\nabla f_i^{(s)}(\tilde{z}_1)$ over the output dimensions $i := 1, ..., D$:

$$\alpha_i^{(s)} := \sqrt{\sum_{j=1}^{2D} \left( \frac{\partial h_{2L,i}^{(s)}}{\partial z_{1,j}} \right)^2} \Bigg/ \sqrt{\sum_{k=1}^{D} \sum_{j=1}^{2D} \left( \frac{\partial h_{2L,k}^{(s)}}{\partial z_{1,j}} \right)^2} \tag{2}$$

Monitoring the gradients at training checkpoints revealed that Rare disturbants destabilize the learning of vanilla RNNs. The similarity of the paired gradients decreased gradually (GRU) or rapidly (LSTM) when the networks were exposed to the Rare disturbants. Positional encoding endowed the RNNs with robustness to these RARE disturbants. Both the GRU and LSTM maintained the high similarity of the paired gradients across the different target/disturbant conditions. By contrast, the impact of positional encoding on the gradient stability of the S4D was marginal; unlike the RNNs, the vanilla S4D was highly stable by itself against Rare disturbants throughout the training, even though there

was a visible relative destabilization due to Rare disturbants compared to Frequent disturbants in the early stages of training, as well as an observable improvement by positional encoding.

# 5 Discussion

## 5.1 Difficulties in Handling a Large Vocabulary

This study introduces a novel challenge in training (vanilla) RNNs: managing large vocabularies. While the manageable vocabulary size of RNNs is a pertinent research area, crucial for empirical applications like natural language processing, previous studies have primarily focused on evaluating and improving the memory duration of RNNs, typically with small vocabulary sizes.

This research examined RNN gradients and identified their destabilization when processing low-frequency tokens, which are necessarily included in a large vocabulary. Specifically, inputs that do not contribute to gradient-based optimization at a target time step were found to be detrimental.

In general time series processing, data points carrying crucial information for specific time steps become irrelevant otherwise. Consequently, each token exhibits a dual nature—both crucial and noisy—throughout the task. Processing rare tokens is particularly challenging, presumably because they are irrelevant most of the time while making a large impact on learning due to their greater loss, compensating for fewer learning opportunities. Dealing with such "unignorable noise" presents a pervasive challenge for RNNs.

## 5.2 Functionality of Positional Encoding beyond the Timekeeper for Transformers

Although low-frequency tokens destabilize the gradient-based learning of RNNs, this study also discovered that positional encoding can alleviate this issue. This enhancement of RNNs via positional encoding is noteworthy because RNNs were specifically designed to process time series data on their own. Unlike Transformers, they are presumed to function without relying on an "external clock". Consequently, position-encoded RNNs have remained largely unexplored. The findings of the present study—namely, the improvement in the manageable vocabulary size due to enhanced gradient stability—broaden the currently limited understanding of the impact of positional encoding on RNNs.

Additionally, the results of this study shed new light on the utility of positional encoding. While positional encoding has been viewed as nothing more than input timestamps for Transformers, the present study demonstrated its efficacy in stabilizing the gradients of RNNs against disruption by low-frequency tokens. This novel functionality of positional encoding would not have been visible in Transformer studies, as the model can dynamically adjust the relevance of input tokens through their attention mechanism, thus inherently mitigating the impact of disturbant tokens.

## 5.3 Limitations and Future Directions

A primary unresolved question in this study pertains to the mechanism behind the gradient stabilization by positional encoding. All the findings here are based on experimental investigations, lacking rigorous mathematical explanations for how and why the gradients of RNNs are destabilized by infrequent tokens and stabilized by positional encoding. Moreover, the present study primarily focused on the canonical implementation of sinusoidal positional encoding designed for Transformers, leaving open which parameters of the sinusoidal waves (i.e., frequencies and phases) are critical for gradient stabilization. Future research may broaden its scope to encompass more general forms of positional encoding, such as wavelets and non-periodic signals.

Moreover, the analysis of gradient stability did not fully address the enhanced performance of the position-encoded state-space model (S4D). In terms of accuracy, the positioned-encoded S4D exhibited greater robustness to infrequent tokens compared to the vanilla model, resembling the behavior observed in RNNs. However, the gradients of the vanilla S4D were too stable to account for this decline in performance. This leaves open the question of how positional encoding influences gradient-based learning of state-space models. Additionally, future studies may investigate a broader range of state-space models to achieve a comprehensive understanding of the interplay between positional encoding and these models.

In addition to these scientifically oriented questions, future studies could also address practical applications of position-encoded RNNs and neural state-space models. Although positional encoding enhanced model performance across different synthetic tasks, the extent of this enhancement is task-dependent. Thus, positional encoding is not a panacea for arbitrary tasks, and further investigations are necessary to determine when it is effective.

# 6 Appendix

## 6.1 Other Tasks

This section demonstrates the effectiveness of positional encoding on RNNs across different tasks, besides the reverse ordering task discussed in the main text.

### 6.1.1 Reverse-Ordering + Delayed-Addition

This section reports the performance of position-encoded RNNs on a more complicated, combinatorial task than the reverse ordering of input sequences. Extending the reverse-ordering task, the models received additional random input integers during the output phase, and added each of them to the corresponding token in the reverse-ordered input sequence (modulo the vocabulary size, so that the output range was bounded). This task was too challenging to GRUs—even after reducing the input length to L = 16—so only the results from LSTMs are reported below. Also, the network was trained for 600,000 iterations (i.e., twice longer than the other tasks) for ensuring the convergence. The other conditions/hyperparameters were the same as reported in the main text. Consequently, positional encoding improved the model performance as the vocabulary size grew from 896 to 1088.

### 6.1.2 Sorting

In the reverse ordering task, the order of input integers was important information for accomplishing the task. Thus, positional encoding may play its originally intended role in encoding the temporal information.

This section reports the effectiveness of positional encoding for a task in which the order of input observations was completely irrelevant; the learning objective was to simply sort the input integers in their inherent ascending order (e.g. 8, 29, 2, 11 -> 2, 8, 11, 29). The input integers were uniformly randomly sampled with replacement, allowing for ties in the sorting process.

As a result, positional encoding also proved effective for RNNs to handle a larger vocabulary in the sorting task, though the improvement remained marginal compared to the reverse-ordering task.

### 6.1.3 Predecessor Query

Finally, this section presents benchmark results for the predecessor-query task. The network first received a sequence of non-repeating random integers, $x_1, ..., x_L$. Subsequently, one of the non-initial input integers, $x_{t_{query}}$ ($2 \leq t_{query} \leq$ L), was randomly selected and reintroduced to the network at time t = L + 1. The learning objective is to return the predecessor of the reviewed integer (= $x_{t_{query}-1}$). The predecessor-query task evaluates the capacity of RNNs to integrate information regarding both the order and content of input sequences.

As in the reverse-ordering + delayed-addition task, the input sequence was reduced to L = 16 due to the complexity of the task, and the experiment focused on the LSTM. The number of training iterations was maintained at 300,000. Similar to the other benchmarks, positional encoding improved the LSTM's capacity to manage the larger vocabularies.

## 6.2 Robustness to Variations in Input Length

So far, all the tasks were experimented using fixed-length inputs (L = 64). One might wonder if positional encoding is exceptionally effective under this setting, informing RNNs with the exact timing when each input token should be returned as the output. Thus, it remains unclear whether or not position-encoded RNNs can also handle a larger vocabulary even when the input length is variable and, thus, the exact timing of the output emission is not identifiable from the positional encoding attached to the inputs.

To assess the robustness to variations in the input length, an additional experiment was conducted on the LSTM, with the input length varied between 32 and 64. In this setup, the maximum input length (= 64) covers the entirety of the shortest input sequence plus its reversed reconstruction (= 32 + 32). Consequently, the positional encoding per se cannot even distinguish the input vs. output phases at t = 33, ..., 64. The vocabulary size was set to 16,384.

As a result, the positional encoding still improved the LSTM's performance on the reverse-ordering task against the perturbations in the input length. This result suggests that the effectiveness of the positional encoding for RNNs is not limited to strictly scheduled tasks.

### 6.3 Effects of Additional Parameters in Position-Encoded RNNs

The concatenation of positional encoding with input embeddings inflates the number of learnable parameters in the input-to-hidden projection weights. This additional parameterization per se does not influence the learning of the input embeddings, and therefore does not elucidate the enhanced performance of position-encoded RNNs. This section substantiates this argument by equalizing the number of learnable parameters between the vanilla and position-encoded models.

Specifically, the equalization was achieved by concatenating two identical copies of the input embeddings and feeding them to the LSTM. This configuration—henceforth termed "double vanilla"—effectively doubled the size of the input- to-hidden weight for each gate in the LSTM, aligning it with that of the position-encoded LSTM, while maintaining all other parameters, including the dimensionality of the (non-repeated) input embeddings.

As illustrated, the double vanilla LSTM did not yield any improvements in the reverse-ordering or sort- ing tasks. These results affirm that the reported enhancement of RNNs is not merely attributable to the additional parameterization associated with the positional encoding.

### 6.4 Alternative Implementations of Positional Encoding

While this study implemented positional encoding by sinusoidal waves, there are alternative implementations proposed in the previous studies. For instance, the BERT-based models typically encode each token position by a learnable embedding. Moreover, the original study of Transformer pointed out that even random vectors can function as positional encoding.

Accordingly, these two alternative forms of positional encoding were tested on the LSTM performing the reverse- ordering task. The random position-encoding vectors were uniformly and independently sampled from the (512  1)- dimensional hypersphere. The learnable embeddings were implemented using the canonical embedding module of PyTorch (torch.nn.Embedding). The input length and vocabulary size were set to 64 and 16,384 respectively. Both the random vectors and learnable embeddings improved the performance of LSTM.

Among the different implementations of positional encoding, the sinusoidal encoding outperformed the two alterna- tives. The advantage of the sinusoidal encoding became more apparent when the input length was variable between 32 and 64; the sinusoidal encoding was more robust to the variations in the input length than the others.

### 6.5 Language Modeling

This section reports benchmark results for the language modeling task. Single-layer LSTMs with and without sinusoidal positional encoding were trained and tested on the WikiText-103 dataset. Due to constraints in computational resources, the vocabulary was reduced from the original size of 267,735 to 32,768 by retokenizing the raw data using SentencePiece. The headings were removed, and the main text was segmented by paragraphs (separated by the line break). Additionally, only the first 1024 tokens of each paragraph were utilized for training and testing, ensuring that the absolute positional encoding always aligned with the beginning of each paragraph. The hyperparameters were configured as specified in §3.3.

As illustrated, positional encoding proved effective only for marginally faster learning during the initial phase of training. The difference diminished around 10,000/30,000 iterations, and the test perplexities of the position-encoded model were inferior to those of the vanilla model.

Table 1: Test perplexities on the WikiText-103 dataset. The minimum, mean, and maximum are obtained from five trials with different random seeds.

| Model | Min | Mean | Max |
|---|---|---|---|
| Vanilla LSTM | 36.8257 | 37.7731 | 38.916589 |
| Position-Encoded LSTM | 38.0685 | 38.5384 | 38.893656 |