Enhanced Vocabulary Handling in Recurrent Neural Networks Through Positional Encoding

Abstract

This research presents a counterintuitive discovery: positional encoding, a high-dimensional representation of temporal indices, improves the learning capabilities of recurrent neural networks (RNNs). While positional encoding is well-known for its crucial role in enabling Transformer networks to process sequential data, its application to RNNs, which inherently manage temporal information, seems unnecessary. However, our experiments with synthetic benchmarks demonstrate that incorporating positional encoding into RNNs enhances their performance, particularly when dealing with extensive vocabularies that result in numerous low-frequency tokens. A detailed analysis reveals that these infrequent tokens introduce instability to the gradients of standard RNNs, and positional encoding effectively counteracts this instability. These findings highlight a previously unrecognized benefit of positional encoding, extending its utility beyond its conventional function as a temporal marker for Transformers.

1 Introduction

Since their introduction, Transformer neural networks have become the preferred method for processing and generating time series data, surpassing traditional models like recurrent neural networks (RNNs). A significant distinction between these two types of models lies in their approach to encoding temporal information, which refers to the sequence of individual data points, or tokens, within the time series. RNNs encode this information by sequentially updating their internal state based on both the current input and the preceding state. Conversely, Transformers do not inherently possess a mechanism to represent the order of data points; thus, they depend on an external system known as positional encoding to provide this temporal context.

Positional encoding offers a high-dimensional representation of the temporal indices associated with input data. Its most common implementation involves the use of sinusoidal waves with predetermined frequencies. This method "timestamps" input tokens by adding or concatenating these encoding vectors to the corresponding input embeddings. In contrast to RNNs, the temporal representation provided by positional encoding remains unchanged by input values until processed collectively by a network.

Although positional encoding has often been viewed as a substitute for the temporal processing capabilities of RNNs when used with Transformers, the two are not inherently incompatible. Inputs to RNNs can be augmented with position-encoding vectors, despite this appearing redundant. The presence of autonomous activities in biological neurons, like neural oscillations, is believed to be significant in time perception and other perceptual processes, as well as in motor control.

This study, therefore, investigates the application of positional encoding to the inputs of RNNs using synthetic benchmarks. The results demonstrate that positional encoding helps RNNs manage a more diverse set of discrete inputs, effectively handling a larger vocabulary, compared to those without positional encoding.

The contributions of this research are outlined as follows:

- Challenges in training RNNs with extensive vocabularies are shown through carefully designed benchmark tasks. This issue, despite its potential implications for practical applications, has not been previously identified or has received minimal attention.
- The identified training challenges for RNNs with large vocabularies are explained by gradient instability caused by infrequent tokens, which are inevitable when expanding vocabulary size.
- A new effectiveness of positional encoding is revealed by combining it with RNNs, showing it mitigates the large-vocabulary issue 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 Turing-complete, meaning they can simulate any Turing machine if their weights have unlimited precision and are perfectly tuned. Even RNNs with random recurrent and input-to-hidden weights, known as reservoir computers, can achieve universal approximation if their hidden-to-output weights are idealized. These theoretical insights have driven the use of RNNs in processing complex time series like human languages and weather patterns.

However, in practical scenarios, RNN weights are limited by finite precision and must be optimized based on a finite set of data observations. These constraints place limitations on the actual capabilities of RNNs. For instance, empirical RNNs cannot store an infinite number of observations in their memory, and the stored information degrades over time. This issue of memory duration has been a focal point for researchers, leading to extensive exploration of RNN architectures that can retain memory for longer periods.

More recently, the focus of research on extending memory retention has moved towards continuous-time models. Instead of representing the memory of an input sequence through discrete-time changes in a latent state, these models approximate the input history using a linear combination of orthogonal polynomials in continuous-time space. 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 of continuous-time memory representation has been further developed into neural state-space models by replacing the fixed state matrix in the ODE with a learnable one, while restricting its structure to a diagonal matrix plus a row-rank matrix. Notably, with further refinements, the latest state-space model has achieved language modeling performance that rivals that of Transformer-based models.

2.2 Positional Encoding

Positional encoding serves as a high-dimensional representation of the temporal structures present in input data. The primary need for this type of representation arises from Transformers, which, unlike RNNs, do not have an inherent mechanism for representing the order of inputs. Consequently, 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 encoded using sinusoidal waves of various predefined frequencies. While this original encoding method is effective for a wide range of tasks, researchers have also explored other possibilities. For instance, the well-known BERT pretraining for natural language processing used learnable embeddings to encode token positions. Some research has also indicated that combining sinusoidal and learnable encoding can enhance model performance. Another approach involves encoding the distance between tokens rather than the time elapsed since the beginning of the sequence.

Beyond Transformers, positional encoding is utilized to represent elapsed time in diffusion processes. Furthermore, the effectiveness of positional encoding is not restricted to temporal information; previous studies in three-dimensional mesh/point-cloud modeling have shown that sinusoidal transformation of spatial data improves model performance compared to using raw coordinate representations.

Despite the extensive use of positional encoding across various areas of machine learning, its application to pure RNNs remains largely unexplored. To the author's knowledge, only two studies have previously investigated position-encoded RNNs. Karanikolos and Refanidis (2019) found that a position-encoded LSTM outperformed a standard LSTM as well as a shallow Transformer in text summarization tasks. In another study, which predates the introduction of sinusoidal positional encoding in the deep learning community, Vincent-Lamarre et al. (2016) demonstrated that oscillatory signals at random frequencies enhanced the performance of a random RNN (i.e., reservoir computer) in a timing task, evaluating the model's memory duration by its ability to generate a smoothed output pulse after a specific time interval from an onset signal.

Similarly, the time index in time series data has rarely been directly used by RNNs, likely due to its perceived redundancy alongside the functionality of RNNs. As an exception, Neil et al. (2016) introduced a periodic gating mechanism for updating the state and memory cell of LSTM. This periodic gating was scheduled based on a triangular wave interspersed with a plateau at the floor value (= 0.0; the frequency, phase, and duration of the wave phase were learnable parameters).

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.

3.2 Model Architecture

The research in this study was based on single-layer gated recurrent units (GRUs), long short-term memory (LSTMs), and a neural state-space model, S4D (S4 with a diagonal state matrix). Each integer in the input sequences was first embedded, then concatenated

with the positional encoding, and subsequently fed into the RNN/S4D. After processing the entire input sequence, the network received a command to produce the output. This command was represented by a time-invariant learnable vector and was fed to the RNN in place of the input embedding. The outputs from the RNN/S4D module were linearly projected into classification logits. The cross-entropy loss between these logits and the target sequence was used to optimize the entire network. Model predictions during the testing phase were determined by the argmax of these logits for each time step.

This study used the standard sinusoidal positional encoding designed for Transformers. Specifically, each time step t was encoded by the Dpos-dimensional vector, defined as follows:

$$PEt, 2i := \sin\left(\frac{t-1}{10000^{\frac{2(i-1)}{Dpos}}}\right) \tag{1}$$

$$PEt, 2i+1 := \cos\left(\frac{t-1}{10000^{\frac{2(i-1)}{Dpos}}}\right) \tag{2}$$

For learning stability, the positional encoding was divided by the square root of Dpos/2, ensuring that the encoding vectors had a unit L2-norm. The time step t incremented throughout both the input and output phases (i.e., t = 1, ..., L, L+1, ..., 2L, where L is the input length), without any hard-coded association between the 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 (ŏ3b21, ŏ3b22) := (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) and each training-test trial was executed on a single NVIDIA A100 GPU (with 80GB VRAM).

4 Results

4.1 Key Findings

Positional encoding was found to enhance the ability of RNNs to manage 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 sizes 32-256 and 256-16,384, respectively, achieving token-wise accuracy above 95%. In contrast, the performance of the standard models without positional encoding deteriorated as the vocabulary size increased. Similarly, positional encoding improved the capacity of S4D to handle large vocabularies. These improvements are also evident in the reduced sequence-wise reconstruction errors, as measured by the Damerau-Levenshtein distance. Neither additional training iterations nor larger batch sizes improved the performance of the standard models.

4.2 Frequency Matters

The most noticeable effect of increasing the vocabulary size was the decreased probability of observing individual vocabulary items. Therefore, additional experiments were conducted with non-uniformly distributed tokens to examine the relationship between token frequency and RNN performance. Specifically, the input vocabulary was evenly divided into Frequent and Rare groups, with Frequent tokens having three times the probability of Rare tokens. The probability of each Frequent token was 7/8 ŏ0d7 2/K (where K is the total vocabulary size, set to 64, 1024, and 2048 for GRU, LSTM, and S4D, respectively), while the probability of each Rare token was 1/8 ŏ0d7 2/K.

The training data consisted of 64 independent samples from this dual-frequency vocabulary. The test data were systematically constructed so that each sequence included a single "target" token (Frequent/Rare) whose retrieval accuracy was assessed, along with 63 "disturbants" that were either all Frequent or all Rare. The experiment revealed that the frequency of the disturbant tokens significantly affected the performance of the standard RNNs and S4D. Rare targets were successfully retrieved as long as they were surrounded by Frequent disturbants. However, the standard GRU struggled to recover Frequent targets when the other input tokens were filled with Rare disturbants. LSTM performance also degraded, especially when targets were positioned in the first quarter of the input sequence (1 ŏ2264 t ŏ2264 16). Similarly, Rare disturbants were detrimental to S4D; unlike the RNNs, the accuracy was lowest when targets were located in the middle of the input sequences (17 ŏ2264 t ŏ2264 32).

In contrast, the position-encoded RNNs showed robustness to the frequency of both target and disturbant tokens. They achieved nearly perfect accuracies in most cases, except when the GRU processed fully Rare data with the target in the first half of the sequence (1 Ž264 t Ž264 32). Likewise, positional encoding enhanced the resilience of S4D against the influence of Rare disturbants.

4.3 Analysis of Gradient Stability

To further investigate the influence of token frequency on RNN performance, the gradients of the RNN latent states were analyzed. Pairs of input sequences were processed by RNNs trained on the dual-frequency vocabulary. Each pair shared the same initial token (t = 1; "target") but varied in subsequent tokens (2 Ž264 t Ž264 L; "disturbants"). Gradients were then computed for the distant mapping between the first and last updated states (at 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. The gradient stability of the RNNs was defined by the dot-product similarities between the normalized gradients of these paired mappings:

$$\text{Stability}(A,B) := \sum_{i=1}^{D} \langle \alpha_i^{(A)} \nabla f_i^{(A)}(\vec{z}_1), \alpha_i^{(B)} \nabla f_i^{(B)}(\vec{z}_1) \rangle = \sum_{i=1}^{D} \alpha_i^{(A)} \alpha_i^{(B)} \left(\sum_{j=1}^{2D} \frac{\partial h_{2L,i}^{(A)}}{\partial z_{1,j}} \cdot \frac{\partial h_{2L,i}^{(B)}}{\partial z_{1,j}} \right) \tag{3}$$

where the coefficients Ŏ3b1(s) i normalized the raw gradients Ž207f (s) i (z1) over the output dimensions i := 1, . . . , D:

$$\alpha_i^{(s)} := \sqrt{\left(\sum_{j=1}^{2D} \left(\frac{\partial h_{2L,i}^{(s)}}{\partial z_{1,j}} \right)^2 \right)} \Bigg/ \sqrt{\left(\sum_{k=1}^{D} \left(\sum_{j=1}^{2D} \left(\frac{\partial h_{2L,k}^{(s)}}{\partial z_{1,j}} \right)^2 \right) \right)} \tag{4}$$

Consequently, the stability metric emphasizes the consistency of the paired gradients that both have a greater L2-norm across the output dimensions.

It is important to note that the mapping from the first to the last RNN state was conditioned on the disturbant tokens occurring at 2 Ž264 t Ž264 L. Nevertheless, the reverse-ordering task trained the networks to retrieve the initial token as their final output regardless of the intervening tokens. Thus, a well-trained RNN would maintain invariance in its final state over the disturbants. Conversely, consistent gradient directions across varied disturbants would lead to successful learning, which is the premise of the proposed analysis.

Unlike the RNN models, both the standard and position-encoded S4Ds achieved high accuracy over 96% for the initial target token (t = 1), regardless of the frequency of the target and disturbants. Therefore, for the analysis of S4D, the target token was positioned in the middle at t = 23, where the standard model exhibited its poorest accuracy with Rare disturbants. The disturbants were prefixed and suffixed to this target to construct input sequences. The prefix disturbants were shared between the paired sequences, ensuring that the latent dynamics of the model remained identical up to the target token.

It should also be noted that the latent states of S4D are complex-valued (while its outputs are real-valued), and consequently, the gradients and their dot-product similarities are also complex-valued. For this analysis, the complex-valued gradients were treated as double-sized real arrays, and a real-valued similarity was defined by Eq. 3. This is equivalent to taking the real component of the complex-valued similarity and is intuitively natural given that a perfect alignment between complex gradient directions yields a real-valued score of 1.0. Additionally, the extra dimension in the latent states representing the order of the Legendre polynomials was merged with the channel dimension, and the entire state was treated as a flattened vector.

Monitoring the gradients at training checkpoints revealed that Rare disturbants destabilize the learning of standard RNNs. The similarity of the paired gradients decreased gradually (GRU) or rapidly (LSTM) when the networks were exposed to Rare disturbants.

Most notably, positional encoding endowed the RNNs with robustness to these Rare disturbants. Both the GRU and LSTM maintained high similarity of the paired gradients across different target/disturbant conditions. In contrast, the impact of positional encoding on the gradient stability of S4D was marginal; unlike the RNNs, the standard S4D was highly stable by itself against Rare disturbants throughout training, although 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. It is also noteworthy that the difference between Frequent and Rare disturbants diminished after 10,000 training iterations. Consequently, gradient stability does not fully account for the decline in S4D accuracy in the presence of Rare disturbants, nor does it explain the enhancement brought about by positional encoding.

5 Discussion

5.1 Difficulties in Handling a Large Vocabulary

This study introduced a novel challenge in training standard RNNs: large vocabularies. While investigating the manageable vocabulary size of RNNs appears to be a relevant research area, crucial for practical applications like natural language processing, previous studies have primarily focused on evaluating and improving the memory duration of RNNs, typically setting the vocabulary size to a small value (= 8).

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 (e.g., tokens at 2 ̆2264 t ̆2264 L upon the retrieval of the initial token at t = 2L in the reverse-ordering task) 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 through the greater loss to compensate for their 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 this issue can be alleviated by positional encoding. This enhancement of RNNs via positional encoding is noteworthy because RNNs were specifically designed to process time series data on their own; hence, unlike Transformers, they are presumed to function without relying on an "external clock". Consequently, position-encoded RNNs have remained largely unexplored, with only two exceptions to the best of the author's knowledge. The findings of this 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, this study demonstrated its effectiveness 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 and thus inherently mitigate 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 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, this study primarily focused on the canonical implementation of sinusoidal positional encoding designed for Transformers (Eqs. 1, 2), leaving it 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 standard model, resembling the behavior observed in RNNs. However, the gradients of the standard 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—including the state-of-the-art architecture of Mamba—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. Indeed, while a previous study reported the effectiveness of positional encoding for an LSTM text summarizer, the present study found no empirical advantage for the language modeling task, aside from a slightly more rapid decline in training loss. 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 A 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 A.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 A.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 ĭ192 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 A.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, x1, . . . , xL. Subsequently, one of the non-initial input integers, xtquery (2 ĭ264 tquery ĭ264 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 (= xtqueryĭ22121). 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 B 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 C 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 standard 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 standard"—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.

The double standard 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 D 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, it has been 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 E 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 Section 3.3.

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 standard 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