Positional Encoding, Tokenization, and Architecture Hyperparameters for Transformers

Sergio Charles
Department of Statistics
Stanford University

Thomas Lau
Department of Statistics
Stanford University

1 Abstract

In our study, we investigated the effect of different positional encoding schemes (learned, sinusoidal, and rotary), tokenization schemes (character-level, word-level, and byte pair encoding), and model size (hyperparameter search over number of heads, layers, and context size) on validation model perplexity. You can find a link to our complete code here: `https://drive.google.com/file/d/1-MVtOoWXc1eyioYy7NBujryHwtm4wnkJ/view?usp=sharing`. Please note that for the rotary positional encoding, we used a function from the original implementation: `https://github.com/princeton-nlp/CEPE/blob/main/modeling_llama_flash.py#L122`.

2 Proposed Improvements

Our first proposed improvement is to implement different positional encoding schemes, namely comparing methods of absolute and relative positional encoding. Without positional encodings, there is no inherent ordering of a sequence of input tokens $X \in \mathbb{R}^{T\times d}$. That is, the self-attention mechanism in the Transformer is equivariant to permutations on the set of input tokens. However, this symmetry violates the common sense assumption that the ordering of words in a sentence matter. Hence, as proposed in the original *Attention is All You Need* [1] paper, one can inject information about the relative or absolute positions of tokens. More precisely, consider token embeddings $x_m, x_n \in \mathbb{R}^d$ at positions m and n, respectively. In the self-attention mechanism, position information is fed to token embeddings which are transformed into key, query, and value vectors:

$$\begin{aligned} k_n &= f_k(x_n, n) \\ q_m &= f_q(x_m, m) \\ v_n &= f_v(x_n, n) \end{aligned} \tag{1}$$

where k_n, q_m, v_n incorporate the m and n positions through f_k, f_q, f_v, respectively. For **relative positional encoding**, when we apply f to both vectors and take an inner product, the result should depend on the difference $m - n$:

$$\langle f_q(x_m, m), f_k(x_n, n)\rangle = g(x_m, x_n, m - n). \tag{2}$$

We hypothesize that relative position encodings allow for more effective distinction between positions, as they exhibit **long-term decay** [2] wherein $g(x_m, x_n, m - n) \to 0$ as $m - n \to 0$. This is a fundamental property, as a pair of tokens with long relative distance should have less dependence.

2.1 Absolute Encoding: Sinusoidal

In **absolute position encoding**, the query, key and value representations are:

$$f_{\{q,k,v\}}(x_t, t) = W_{\{q,k,v\}}(x_t + p_t) \tag{3}$$

for $p_t \in \mathbb{R}^d$ a d-dimensional vector depending on the position of the token. Computing the inner product:

$$\langle f_q(x_m, m), f_k(x_n, n)\rangle = \langle W_q(x_m + p_m), W_k(x_n + p_n)\rangle = (x_m + p_m)^\top W_q^\top W_k(x_n + p_n) \tag{4}$$

we find that absolute encodings *do not* satisfy Equation 2 and, as such, do not exhibit the desirable long-term decay as $m - n \to \infty$. We ran experiments with the sinusoidal encoding scheme, originally proposed by Vaswani et al. [1], such that for the t-th token, the embedding at even and odd dimensions is defined as:

$$P_{t,2i} = \sin(t/\eta^{2i/d})), \quad P_{t,2i+1} = \cos(t/\eta^{2i/d}), \tag{5}$$

Preprint. Under review.

where we typically set $\eta = 10,000$. One can think of each dimension of the positional encoding as a sinusoidal with frequency $\omega = \frac{1}{\eta^{2i/d}}$. Thus, as the dimension i increases, the oscillations become more frequent. Although this is an absolute position encoding scheme, since $P_{t+k,\cdot}$ can be represented as a linear function of $P_{t,\cdot}$ for all offsets k, it was *theorized that self-attention should be able to learn based on relative positions* [1].

2.2 Relative Encoding: Rotary

Rotary Positional Embeddings (RoPE) [2] implicitly encode absolute position using a rotation matrix and relative positional dependency in the key and query vectors. For $d = 2$, we can represent a vector with a complex number $x_m = |x_m|e^{i\theta_m}$ and find that the following functions *satisfy the relative position condition* in Equation 2:

$$\begin{aligned} f_k(x_n, n) = (W_k x_n)e^{in\theta}, \quad f_q(x_m, m) = (W_q x_m)e^{im\theta} \\ g(x_m, x_n, m-n) = \text{Re}[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}]. \end{aligned} \tag{6}$$

Since the key and query vectors are linear transformations of the inputs, we choose f to be a linear transformation so it will be easy for the network to learn. In fact, we can write f as a matrix multiplication:

$$f_{\{q,k\}}(x_m, m) = \begin{bmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{bmatrix} \begin{bmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{bmatrix} \begin{bmatrix} x^{(1)}_m \\ x^{(2)}_m \end{bmatrix} \tag{7}$$

To generalize this to any even dimension d, we partition the embedding space into $d/2$ sub-spaces and combine them as follows:

$$\begin{aligned} f_{\{q,k\}}(x_m, m) &= R_{\Theta,m} W_{\{q,k\}} x_m \\ R_{\Theta,m} &= \begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{bmatrix} \end{aligned} \tag{8}$$

with parameters $\Theta = \{\theta_i = 10,000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]\}$. In the 2D case, since $f(x_m, m) = x_m e^{im\theta}$ is a multiplication by a periodic function, it will produce the same representation at different distances. Making θ large could allow the network to distinguish between positions that are far away. However, it might also mean the amplitude of the positional encodings are too close, making it hard to distiguish nearby positions. To get around this, rotary encodings use different values of θ for each pair of dimensions, which can be thought of as encoding information at varying lengths. Unlike sinusoidal encoding, rotary encoding has the nice property that it encodes relative positional information as per Equation 2 and *exhibits long-term decay*. Another key difference between rotary and sinusoidal encodings is that the rotary method mix pairs of coordinates, rather than individually changing each coordinate.

2.3 Learned Pre-Layer Normalized Encoding

We evaluated applying a layer normalization to the token embeddings, before adding a learned positional encoding $P \in \mathbb{R}^{T \times d}$. That is, for sequence embedding $X \in \mathbb{R}^{T \times d}$:

$$\tilde{X} := \text{LayerNorm}(X) + P. \tag{9}$$

We postulate *this could lead to faster convergence due to smoother gradients and greater generalization capabilities*. Such behavior is attributed to the re-centering and re-scaling of backward gradients induced by LayerNorm [3].

2.4 Word-level and Byte Pair Encoding Tokenizers

We also evaluated the performance of a learned word-level tokenizer and a word-level Byte Pair Encoding (BPE) tokenizer relative to the original character-level tokenizer used by the assignment. *Neural Machine Translation of Rare Words with Subword Units* [4] was an early successful use case of the BPE tokenizer, which, put simply, is a "data compression technique that replaces the most frequent pair of bytes in a sequence with a single unused byte" [4]. In NLP, one merges characters or sequences of characters in place of frequent bytes. We hypothesize that due to the small size of the dataset, *character-level tokenizers will perform better than word-level tokenizers*.

3 Results

3.1 Positional Encoding Schemes

We first compared the model performance for learned, sinusoidal, rotary, and learned pre-layer normalized positional encodings. During this experiment, we fixed the default model hyperparameters suggested in the assignment ($L = 6, T = 256, H = 6$). We report the train/validation PPL and training time for the Transformer with different positional encoding schemes in Table 1. Evidently, the rotary embedding performs the best of all methods, while the sinusoidal embeddings, as implemented in the original *Attention is All You Need* [3] paper is considerably worse. The training time for all encoding schemes are commensurate, with the sinusoidal encoder being ~ 10 seconds faster than the rotary encoder.

Method	**Train PPL**	**Val PPL**	**Time (s)**
Baseline (learned)	5.24	6.13	102.39
Sinusoidal	8.37	8.62	**101.22**
Rotary	**3.86**	**4.80**	114.64
Learned Encoding with Layer Norm	6.77	7.57	101.35

Table 1: Comparison of different embedding methods on train perplexity (PPL), validation perplexity (PPL), and training time.

We also plot the training loss curves in Figure 1.

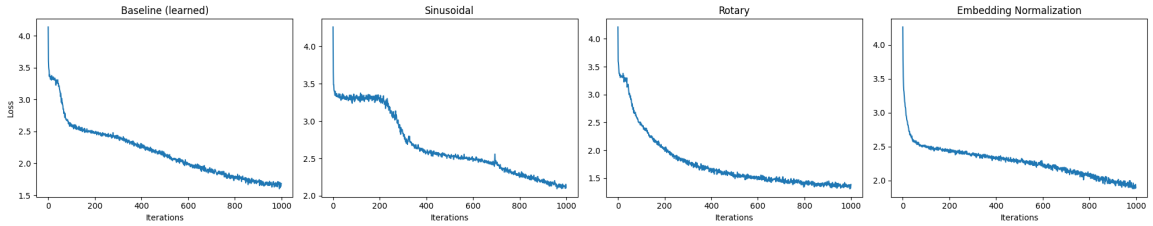


Figure 1: Training curves of Transformer using different positional encoding schemes.

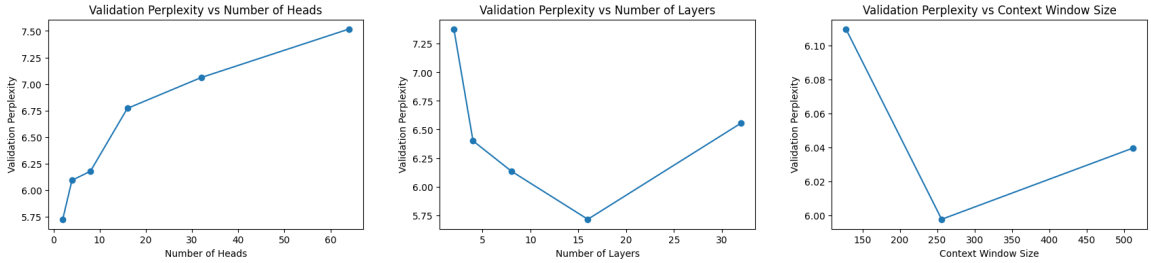
3.2 Hyperparameter Search: Heads, Layers, and Context Size

Next, we performed a hyperparameter search over the number of heads ($H = 2, 4, 6, 8, 16, 32, 64$), the number of Transformer block layers ($L = 2, 4, 6, 16, 32$), and context window size ($T = 128, 256, 512$). We performed this hyperparameter search in a "coordinate-ascent" fashion, whereby we used the optimal hyperparameter found in the first sweep for H to initialize the subsequent hyperparameter sweep over L. The two optimal hyperparamters were then used to initialize the hyperparamter sweep over T. We found $H = 2$, $L = 16$, and $T = 256$ were optimal for the rotary encoding scheme.

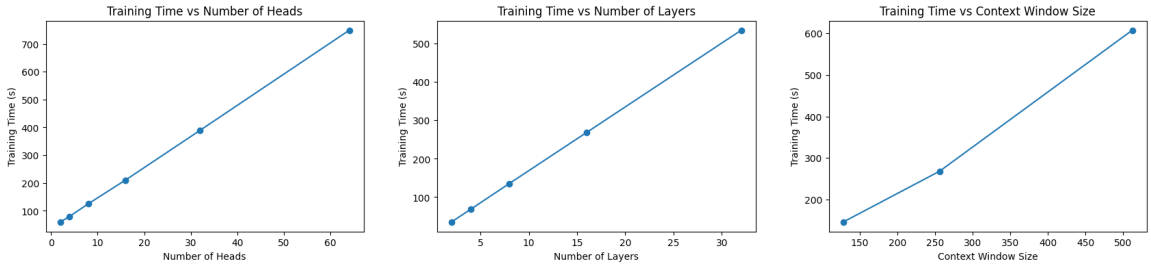
As seen in Figure 2 below, adding more heads hurts performance, which we posit is due to the small size of the Shakespeare dataset. That is, we require only a few latent embedding spaces because the datasets are small, with a train size of 1M tokens and validation size of 111K tokens. As expected, as we increase the number of layers, the validation PPL reaches a minimum for $L = 16$ and begins to increase due to overfitting. The context window exhibits similar behavior, which indicates a saturated context window size given the number of layers.

3.3 Word-level Tokenization and BPE

Fixing the architecture hyperparameters to the ones with lowest validation perplexity from Section 3.1, ($H = 2$, $L = 16$, and $T = 256$ with a rotary encoding), we compared the performance of word-level tokenization to BPE. As presented in Table 2, learned word-level tokenization is considerably worse than our baseline character-level tokenization. This is expected, as the word-level vocabulary size is $13,331$ with $236,634$ training tokens. Hence, the ratio of training tokens to the number of categories in the softmax prediction is very low at 17.7. One can think of this ratio as the amount of *information we have per category*. The BPE has



(a) PPL vs. Heads. (b) PPL vs. Layers. (c) PPL vs. Context Size.



(d) Time vs. Heads. (e) Time vs. Layers. (f) Time vs. Context Size.

Figure 2: Validation PPL and Training Time for H, L, and T.

a higher ratio of ~ 3.6K. However, character-level tokenization is best-in-class with a ratio of ~ 15.4K. We suspect this *ratio is highly correlated with the tractability of the next-token prediction task and, consequently, low PPL.*

Tokenization Method	**Train PPL**	**Val PPL**	**Time**	**Vocab Size**	**Train Tokens**	**Ratio**
Baseline Tokenization (Character-level)	**3.86**	**4.80**	**114.64**	65	1,003,854	**15,443**
Word-level Tokenization	31.80	310.27	321.22	13,331	236,634	17.7
BPE Tokenization	6.84	14.64	303.09	159	578,662	3,639

Table 2: Comparison of tokenization methods based on training perplexity (PPL), validation perplexity (PPL), and training time.

Character-level tokenization furnishes 2.8x and 2.6x faster training times than word-level and BPE tokenization, respectively. The fast convergence of BPE relative to word-level tokenization is shown in Figure 3, as well as the discrepancy in PPL.

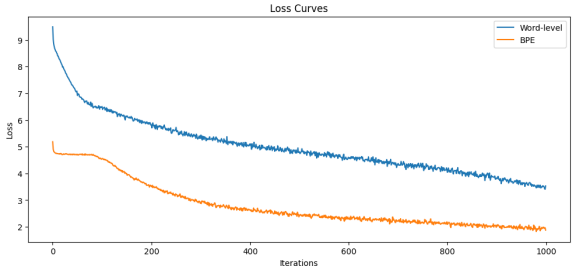


Figure 3: Loss Curves for Word-level vs BPE Tokenization.

4 Conclusion

We find that rotary encoding, character-level tokenization, combined with a model architecture of 2 heads, 16 layers, and 256 context length gives the lowest validation perplexity on the Shakespeare dataset. Furthermore,

character-level tokenization has the highest level of training information per token and yields considerably lower PPL than word-level and BPE tokenizers.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[3] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization, 2019.

[4] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.