Encoding Position by Decaying and Updating Different Exponentiated States Differently

Franz A. Heinsen
franz@glassroom.com

Abstract

We propose encoding position information in token sequences by decaying and updating the exponentiated states of different position-encoding features differently. At each step, we exponentiate the previous state of each position-encoding feature, decay it by a hidden probability, add to it an exponentiated hidden logit, and take the logarithm, obtaining the feature's updated state. We compute each hidden probability and logit dynamically from token state. We implement our proposed method and verify that it works well in practice.[1] The benefits of our proposed method include large representational capacity, unbounded sequence length, flexibility of implementation, low compute cost, and small memory footprint.

1 Summary

The attention mechanism used in Transformers, widely used for sequence modeling, is invariant with respect to the reordering of tokens in an input sequence, requiring the incorporation of position information in token states. Numerous methods have been proposed for encoding position information, including learnable position embeddings, manipulation of attention matrices, and preprocessing of input sequences with recurrent neural networks. For an overview of many methods proposed to date, see Dufter et al. (2022)'s survey.

Here we propose what we believe is a new method for encoding position information that can be applied recurrently or in parallel. Our proposed method is to decay and update the exponentiated states of different position-encoding features differently: At each step, we exponentiate the previous state of each position-encoding feature, multiply it by a hidden probability, add to it an exponentiated hidden logit, and take the logarithm, obtaining the position encoding feature's updated state (Figure 1). We obtain hidden probabilities and logits by applying a feed-forward function $\mathcal{H}$ to token state x_i at each step $i = (1, 2, \dots)$, and update the token with a residual obtained by applying a feed-forward function $\mathcal{R}$ to the position-encoding features' updated state s_i:

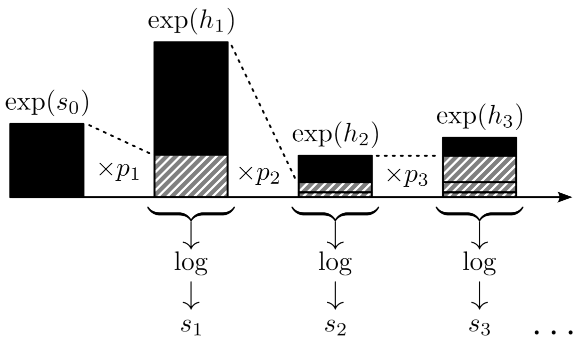

Figure 1: Updating a single position-encoding feature's state $s_i \in \mathbb{R}$, with initial state s_0, given a hidden probability $p_i \in (0, 1)$ and logit $h_i \in \mathbb{R}$ at each step $i = (1, 2, \dots)$. For d position-encoding features, $s_i \in \mathbb{R}^d$, $p_i \in (0,1)^d$, and $h_i \in \mathbb{R}^d$.

$$\begin{aligned}(p_i, h_i) &\longleftarrow \mathcal{H}(x_i) \\ s_i &\longleftarrow \log(p_i \exp(s_{i-1}) + \exp(h_i)) \\ x_i &\longleftarrow x_i + \mathcal{R}(s_i),\end{aligned} \tag{1}$$

where $p_i \in (0,1)^d$ and $h_i \in \mathbb{R}^d$ are the hidden probabilities and logits, respectively, and d is the number of position-encoding features. At each step $i = (1, 2, \dots)$, the state of position-encoding

[1]Code and instructions for replicating our results are at: https://github.com/glassroom/heinsen_position_embeddings.

features $s_i \in \mathbb{R}^d$ is a log-linear recurrence of their previous exponentiated state, $\exp(s_{i-1}) \in \mathbb{R}^d_+$.

2 Benefits of Our Proposed Method

The benefits of our proposed method include large representational capacity, unbounded sequence length, flexibility of implementation, low compute cost, and small memory footprint. We briefly describe each of these benefits below.

2.1 Large Representational Capacity

At each step, our method can update each position-encoding feature's state to a negative value, by setting its exponentiated state < 1, or to a positive value (including zero), by setting its exponentiated state ≥ 1. A state that can be negative or positive can trivially represent a bit, giving us a lower bound of at least 2^d distinct positions that can be represented by d position-encoding features.

2.2 Unbounded Sequence Length

We formulate our proposed method as a recurrent transformation, enabling application over an unbounded stream of tokens—*i.e.*, there is no preset limit on the number of tokens in the stream.

2.3 Flexibility of Implementation

Feed-forward functions $\mathcal{H}$ and $\mathcal{R}$ are customizable. Our proposed method can be applied recurrently, or in parallel via a parallel scan of the log-linear recurrence of exponentiated states.

2.4 Low Compute Cost

The compute cost per step is fixed, and largely determined by the compute cost incurred by the two customizable feed-forward functions, $\mathcal{H}$ and $\mathcal{R}$.

2.5 Small Memory Footprint

There are no learnable position embeddings. The only learnable parameters are those in the two customizable feed-forward functions, $\mathcal{H}$ and $\mathcal{R}$.

3 Reference Implementation

We implement our proposed method, defining feed-forward functions $\mathcal{H}$ and $\mathcal{R}$ as follows:

$$\begin{aligned} \mathcal{H}(x_i) &:= \big(\sigma(W_p x_i + b_p),\, W_h x_i + b_h\big) \\ \mathcal{R}(s_i) &:= W_\mathcal{R} s_i + b_\mathcal{R}, \end{aligned} \tag{2}$$

where σ denotes the sigmoid or logistic function; W_p, W_h, and $W_\mathcal{R}$ are matrices of learnable weights; and b_p, b_h, and $b_\mathcal{R}$ are learnable biases.

For numerical stability, we execute all computations in the domain of logarithms, updating the state of position-encoding features with

$$s_i \longleftarrow \textsc{lse}(\log p_i + s_{i-1}, h_i), \tag{3}$$

where LSE denotes an elementwise log-sum of exponentials that preserves the dimensions of its operands. We compute $\log p_i$ directly by applying a log-sigmoid instead of a sigmoid function to $W_p x_i + b_p$ in our implementation of $\mathcal{H}$. We parallelize execution of the log-linear recurrence with a recently proposed method that is efficient, numerically stable, and simple to implement with current software tools (Heinsen, 2023). We set the initial state of all position-encoding features to zero.

We test our proposed method on small autoregressive and non-autoregressive Transformers, and are unable to find any discernible difference in generative performance, as measured by cross-entropy loss, compared to a matrix of learnable position embeddings, one of the most commonly used methods for encoding position information.

4 Conclusion

Our preliminary testing suggests that in practice, our proposed method, as we have implemented it, will perform neither better nor worse than other conventional methods, many of which are less space-efficient (*e.g.*, methods that learn an embedding for each possible position in the sequence), more complicated (*e.g.*, methods that modify the attention matrix on-the-fly), and/or more limited (*e.g.*, methods that cannot handle an unbounded stream of tokens). Our proposed method offers multiple benefits, making it a worthwhile candidate for further study and practical application.

References

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. 2022. Position Information in Transformers: An Overview. *Computational Linguistics* 48(3):733–763.

Franz A. Heinsen. 2023. Efficient parallelization of a ubiquitous sequential computation.