Why do we need to encode the position of an input token ?

A transformer operates on sequences (ordered lists) $\mathbf{x}_{(1...\bar{T})}$.

But, rather than processing the elements of the sequence in order

- the transformer implements a direct function of the entire sequence

Since the relative ordering of elements of the sequence can be important

- Machine learning is easy not hard
- Machine learning is hard not easy

How does the transformer discover the relative ordering of input tokens ?

To represent the relative position of each element in the sequence,

- we can pair the input element with an Positional Encoding (PE) of its position in the sequence.

$$\langle \mathbf{x}_{(t)}, \mathrm{PE}(t) \rangle$$

The obvious way to implement the Positional Encoding is as an integer

$$\text{PE}(t) = t$$

This encoding is problematic

- The magnitude of t may exceed the magnitude of $\mathbf{x}_{(t)}$ particularly for long sequences
- We are counting on the "circuits" in the Transformer to perform arithmetic comparison ($t' < t$?)

As an alternative, we could encode the position as a fraction relative to length of the sequence

$$\text{PE}(t) = t/\bar{T}$$

The encoding

- is small magnitude: range of $[0, 1]$
- but the fraction doesn't convey the *number* of preceding tokens
    - 50% of a sequence of length 100 much different than 50% of a sequence of length 1000

Relative Positional Encoding

The authors of the original transformer paper implement a [clever encoding (https://arxiv.org/pdf/1706.03762.pdf#page=6)](https://arxiv.org/pdf/1706.03762.pdf#page=6)

The implementation is clever and a bit subtle. There is a great [blog (https://kazemnejad.com/blog/transformer_architecture_positional_encoding/)](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/) that explains the details.

- the [blog's FAQ (https://kazemnejad.com/blog/transformer_architecture_positional_encoding#faq)](https://kazemnejad.com/blog/transformer_architecture_positional_encoding#faq) addresses many interesting questions

We will summarize.

By way of analogy: consider how integers are represented in decimal notation:

- an array of cyclic digits
- least significant position (position 0) has a cycle length of 10
- digit at position i has cycle length of 10^i.

But digits are categorical rather than continuous values.

The authors use a cyclic representation of "digits" that is continuous

- `sine`, `cosine`

The encoding will be an array of length d_{model} of these values.

- all layers in the Transformer use length d_{model}

$\text{PE}(t)_i$ will denote position i within the array $\text{PE}(t)$ of the encoding of position t.

$$\text{PE}(t)_i = \begin{cases} \sin(\frac{t}{\omega}) & \text{if } i \mod 2 = 0 \\ \cos(\frac{t}{\omega}) & \text{if } i \mod 2 = 1 \end{cases}$$

where

$$\omega = 10000^{\frac{i'}{d_{\text{model}}}} \quad i' = \text{int}(i/2) * 2$$

Why alternate between `sine` and `cosine` in "digits" of PE ?

The authors state one objective for the encoding

- can express *relative* position
- there is a *fixed linear transformation (as a function of k)* for the encoding of any two positions at distance k

$$
\begin{aligned}
\text{PE}(t+k) &= T_k * \text{PE}(t) \quad \text{for all } t \\
& \quad\quad\quad\quad\quad\quad\quad T_k \text{ a matrix specific to distance } k
\end{aligned}
$$

The proof is summarized in the [blog (https://kazemnejad.com/blog/transformer_architecture_positional_encoding/#relative-positioning)](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/#relative-positioning) and detailed [here (https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/)](https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/)

A further objective motivating sinusoidal wave forms

- it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

Representing the combined token and positional encoding

If we represented the positionally-encoded token as a pair

$$\langle \mathbf{x}_{(t)}, \text{PE}(t) \rangle$$

it would be *twice the length* of other data items flowing in the transformer.

Instead, the authors represent the positionally-encoded token as a sum of two vectors of length d_{model}

$$\mathbf{x}_{(t)} + \text{PE}(t)$$

This seems strange: mixing the token embedding and positional embedding.

- the concepts seem to be orthogonal

Here is one [theory (https://www.reddit.com/r/MachineLearning/comments/cttefo/comment/exs7d08/)](https://www.reddit.com/r/MachineLearning/comments/cttefo/comment/exs7d08/) of why addition does not corrupt the meaning of each element of the pair

- Ultimately the encodings affect the query, keys, and values of attention lookup
- Each of these values is subject to a linear transformation before attention lookup
    - via *learned* matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$,
- *Perhaps* the learned transformation is such that
    - within the d_model array, there are disjoint sub-arrays
        - for the token embedding
        - for the positional embedding
        - for large d_model this could be possible

Does the positional encoding ”disappear” as data flows through multiple layers ?

The result of Attention Lookup is an attention-weighted sum of values (positionally encoded tokens, for self-attention).

Why doesn't the position embedding lose its meaning as the value flows through multiple layers of Transformer blocks ?

The FAQ (https://kazemnejad.com/blog/transformer_architecture_positional_encoding#faq) reminds us of the *skip connection*

- the positionally-encoded token *also flows around* the attention layer !
- preserving the information up-stream