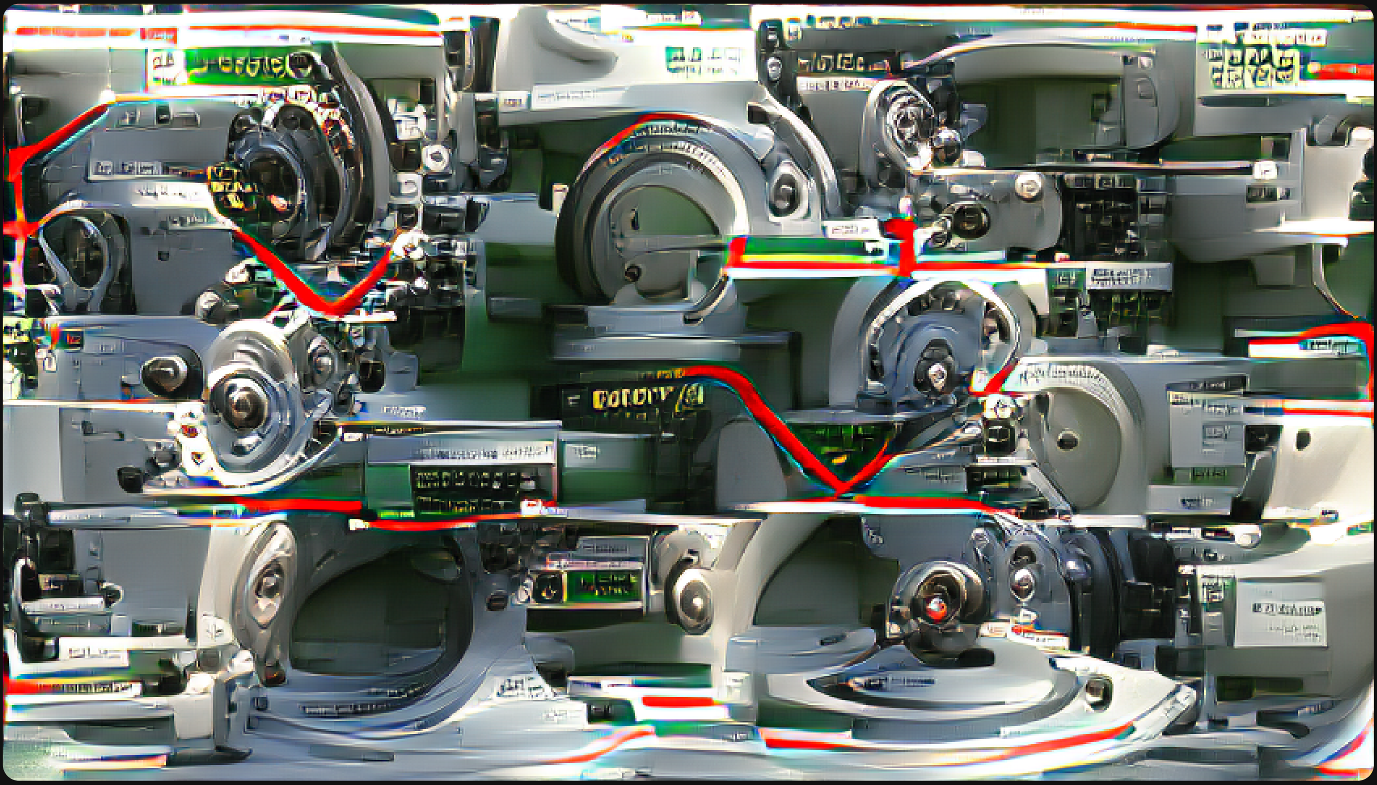
Rotary Embeddings: A Relative Revolution

Rotary Positional Embedding (RoPE) is a new type of position encoding that unifies absolute and relative approaches. We put it to the test.

April 20, 2021 · Stella Biderman, Sid Black, Charles Foster, Leo Gao, Eric Hallahan, Horace He, Ben Wang, Phil Wang



`Rotary position encoding` as imagined by Janus

TL;DR:

Rotary Positional Embedding (RoPE) is a new type of position encoding that unifies absolute and relative approaches. Developed by Jianlin Su in a series of blog posts earlier this year [12, 13] and in a new preprint [14], it has already garnered widespread interest in some Chinese NLP circles. This post walks through the method as we understand it, with the goal of bringing it to the attention of the wider academic community. In general we have found that across a large suite of setups including regular, linear, and local self-attention, it **either matches or surpasses all other methods currently available for injecting positional information into transformers.**

What's the Problem?

Since Vaswani et al., 2017 [16] there have been many schemes introduced for encoding positional information in transformers. When applying self-attention to a given domain, the choice of position encoding typically involves tradeoffs between simplicity, flexibility, and efficiency. For example, learned absolute positional encoding is very simple, but may not generalize and are not always particularly meaningful due to the common practices [1, 3, 9, 15] of packing short sentences and phrases together in a single context and breaking up sentences across contexts.

Another major limitation of existing methods is that they do not work with efficient transformers. Methods like T5's relative positional bias [10] require constructing the full $N \times N$ attention matrix between positions, which is not possible when using many of the efficient alternatives to softmax attention, including kernelized variants like FAVOR+ [2].

A principled, easy to implement, and generally-applicable method for relative position encoding---one that works for both vanilla and “efficient” attention---is of great interest. Rotary Positional Embedding (RoPE) is designed to address this need.

What's the Solution?

In this section we introduce and derive the rotary positional embedding. We begin with discussing the intuition, before presenting a full derivation.

Intuition

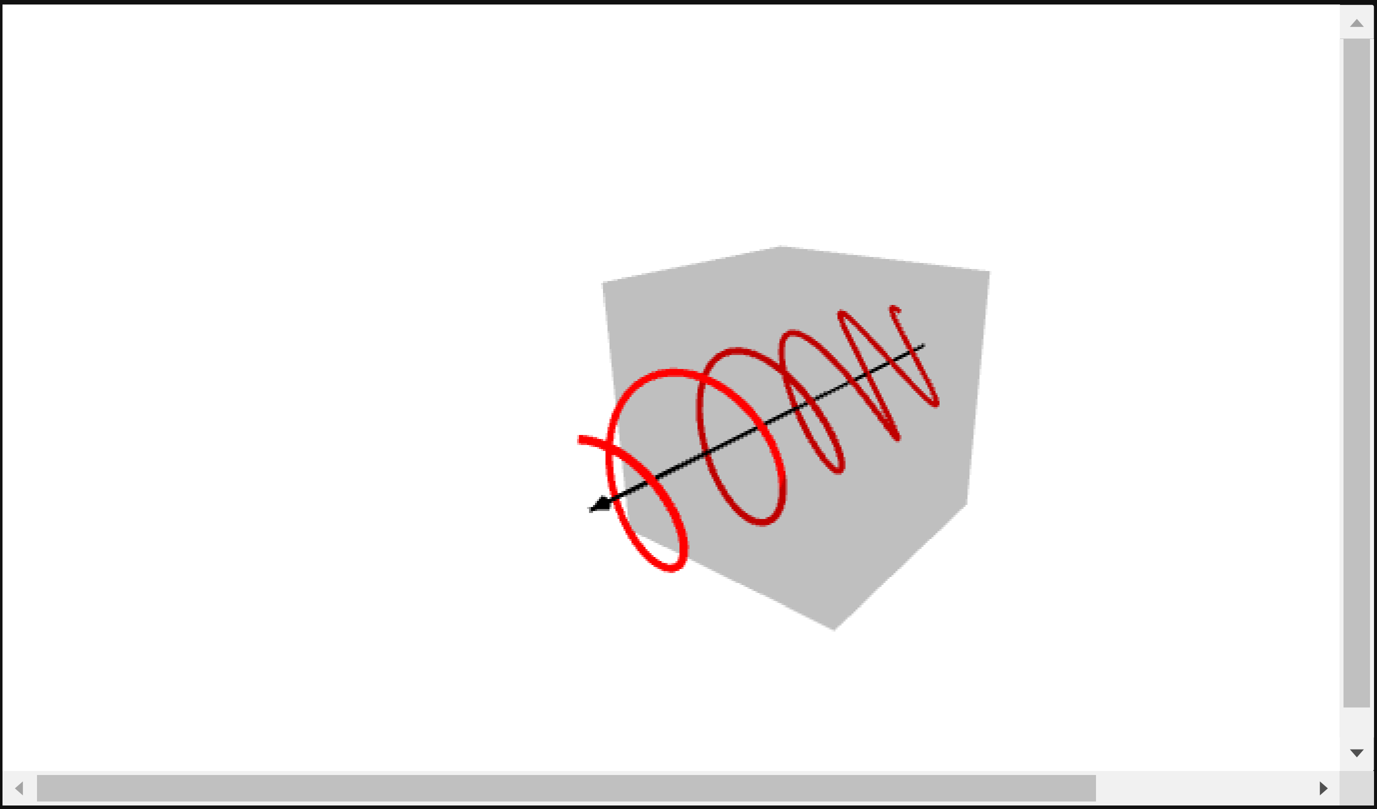
We would like to find a positional encoding function $f(\mathbf{x}, \ell)$ for an item $\mathbf{x}$ and its position ℓ such that, for two items $\mathbf{q}$ and $\mathbf{k}$ at positions m and n, the inner product between $f(\mathbf{q}, m)$ and $f(\mathbf{k}, n)$ is sensitive only to the values of $\mathbf{q}$, $\mathbf{k}$, and their relative position $m - n$. This is related in spirit to the kernel trick: we are searching for a feature map such that its kernel has certain properties. A key piece of information is the geometric definition of the dot product between Euclidean vectors: $\mathbf{q} \cdot \mathbf{k} = \|\mathbf{q}\| \|\mathbf{k}\| \cos(\theta_{qk})$

In plain English, the dot product between two vectors is a function of the magnitude of individual vectors and the angle between them. With this in mind, the intuition behind RoPE is that we can represent the token embeddings as complex numbers and their positions as pure rotations that we apply to them. If we shift both the query and key by the same amount, changing absolute position but not relative position, this will lead both representations to be additionally rotated in the same manner---as we will see in the derivation---thus the angle between them will remain unchanged and thus the dot product will also remain unchanged. By exploiting the nature of rotations, the dot product used in self-attention will have the property we are looking for, preserving relative positional information while discarding absolute position.

The following is an example illustrating the core idea of RoPE—a more rigorous derivation is presented in a subsequent section. Some arbitrary $0 < \varepsilon \leq \frac{\pi}{2N}$ is chosen, where N is the maximum sequence length. When viewed elementwise on $\mathbf{q}$ and $\mathbf{k}$, with j as the element index, RoPE can be viewed as follows:

$$\begin{aligned} \mathrm{RoPE}(x, m) &= xe^{mi\varepsilon} \\ \langle \mathrm{RoPE}(q_j, m), \mathrm{RoPE}(k_j, n) \rangle &= \langle q_j e^{mi\varepsilon}, k_j e^{ni\varepsilon} \rangle \\ &= q_j k_j e^{mi\varepsilon} \overline{e^{ni\varepsilon}} \\ &= q_j k_j e^{(m-n)i\varepsilon} \\ &= \mathrm{RoPE}(q_j k_j, m - n) \end{aligned}$$

Visual Intuition



A quarter-waveplate can change the polarization of an electromagnetic wave. (This figure is interactive, try dragging the cube!)

To see how relative position might be preserved in this transformation, we can look to an analogous situation in classical electrodynamics.

We imagine a linearly polarized electromagnetic wave that is sent through a quarter-wave plate at an angle of 45 degrees. This takes the incoming wave and shifts its phase on only one principal dimension as it travels. When the wave emerges from the waveplate, the polarization is no longer linear---it has become circular through a shift equal to quarter of a period.

As the wave travels through the waveplate, we can see how the magnitude of the wave is preserved. We can also better see how the relative position may be encoded as the angle between subsequent timesteps: the angle between timesteps, and therefore distance along the axis of travel, is constant. This means the positional information must be orthogonal to the amplitude in the modulated wave.

Derivation

We begin with absolute positional information: for each token, we know where it is in the sequence. However, dot products (and therefore attention) do not preserve absolute positional information, so if we encode that positional information in the absolute position of the embeddings, we will lose a significant amount of information. On the other hand, dot products do preserve relative position, so if we can encode the absolute positional information into the token embeddings in a way that only leverages relative positional information, that will be preserved by the attention function.

While it is common in machine learning to restrict our attention to the real numbers, for rotary embeddings it is mathematically more convenient to use the complex numbers as the base field for our space. Instead of working in the usual $\mathbb{R}^d$, we will work in $\mathbb{C}^{d/2}$ by considering consecutive pairs of elements of the query and key vectors to be a single complex number. Specifically, instead of viewing $\mathbf{q} = (q_1, q_2, q_3, q_4, \ldots, q_d)$ as a d-dimensional real vector we view it as $\mathbf{q} = (q_1 + iq_2, q_3 + iq_4, \ldots q_{d-1} + iq_d) \in \mathbb{C}^{d/2}$. As we will see, casting it in this fashion will make discussing the rotary embeddings easier. If d is odd, we can pad it with a dummy coordinate to ensure things line up correctly. Alternatively, we can simply increase d by one.

Let $\mathbf{q}$ and $\mathbf{k}$ be query and key vectors respectively and let m and n be the absolute positions of the corresponding tokens. Let $f(\mathbf{x}, \ell)$ be the function that takes the token embedding $\mathbf{x}$ in position ℓ and outputs a new embedding that contains (in some fashion) the relative positional information. Our goal is to find a "nice" function f that does this. Once the positional information is encoded, we need to compute the inner product like so:

$$\langle f(\mathbf{q}, m), f(\mathbf{k}, n) \rangle = g(\mathbf{q}, \mathbf{k}, m - n)$$

where $g(\mathbf{q}, \mathbf{k}, m - n)$ now represents the pre-softmax logit of the usual attention equation. Writing these three functions in exponential form gives

$$f(\mathbf{q}, m) = R_f(\mathbf{q}, m)e^{i\Theta_f(\mathbf{q},m)}$$
$$f(\mathbf{k}, n) = R_f(\mathbf{k}, n)e^{i\Theta_f(\mathbf{k},n)}$$
$$g(\mathbf{q}, \mathbf{k}, m-n) = R_g(\mathbf{q}, \mathbf{k}, m-n)e^{i\Theta_g(\mathbf{q},\mathbf{k},m-n)}$$

Computing the inner product and equating corresponding components yields

$$R_f(\mathbf{q}, m)R_f(\mathbf{k}, n) = R_g(\mathbf{q}, \mathbf{k}, m-n)$$
$$\Theta_f(\mathbf{q}, m) - \Theta_f(\mathbf{k}, n) = \Theta_g(\mathbf{q}, \mathbf{k}, m-n)$$

Substituting $m = n$ and applying the initial condition $f(\mathbf{x}, 0) = \mathbf{x}$ gives

$$R_f(\mathbf{q}, m)R_f(\mathbf{k}, m) = R_g(\mathbf{q}, \mathbf{k}, 0) = R_f(\mathbf{q}, 0)R_f(\mathbf{k}, 0) = \mathbf{qk}$$

As the prior equation is valid for all m, it means that R_f is independent of the value of m, so we can set $R_f(\mathbf{x}, y) = \mathbf{x}$. Similarly, if we denote $\Theta(\mathbf{x}) = \Theta_f(\mathbf{x}, 0)$ we obtain

$$\Theta_f(\mathbf{q}, m) - \Theta_f(\mathbf{k}, m) = \Theta_g(\mathbf{q}, \mathbf{k}, 0) = \Theta_f(\mathbf{q}, 0) - \Theta_f(\mathbf{k}, 0) = \Theta(\mathbf{q}) - \Theta(\mathbf{k})$$

which implies that $\Theta_f(\mathbf{q}, m) - \Theta(\mathbf{q}) = \Theta_f(\mathbf{k}, m) - \Theta(\mathbf{k})$ for all $\mathbf{q}, \mathbf{k}, m$. This allows us to decompose Θ_f as $\Theta_f(\mathbf{x}, y) = \Theta(\mathbf{x}) + \varphi(y)$. Examining the case of $m = n + 1$ reveals that

$$\varphi(m) - \varphi(m-1) = \Theta_g(\mathbf{q}, \mathbf{k}, 1) + \Theta(\mathbf{q}) - \Theta(\mathbf{k})$$

Since the right-hand side does not depend on m, the left hand side must not either and so φ is an arithmetic progression. Setting the initial values $\varphi(0) = 0$ and $\varphi(1) = \theta$, we have $\varphi(m) = m\theta$.

Putting all of these pieces together, we get the final formula for the rotary positional embedding:

$$f(\mathbf{q}, m) = R_f(\mathbf{q}, m)e^{i\Theta_f(\mathbf{q},m)} = \mathbf{q}e^{i(\Theta(\mathbf{q})+m\theta)} = \sum_{j=1}^{d/2} q_j e^{im\theta_j} \vec{e_j}$$

and likewise for $\mathbf{k}$. Since computers tend to like real numbers and matrices more than complex numbers, its convenient to convert this expression into the matrix equation

$$f(\mathbf{q}, m) = \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_{d/2} \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_d \end{pmatrix} = \mathbf{\Theta_m Q_m} = \mathbf{\Theta_m W_q X_m}$$

where $M_j = \begin{pmatrix} \cos m\theta_j & -\sin m\theta_j \\ sin m\theta_j & \cos m\theta_j \end{pmatrix}$, $\mathbf{\Theta_m}$ is the block diagonal rotation matrix, $\mathbf{W_q}$ is the learned query weights, and $\mathbf{X_m}$ is the embedding of the m token. Again, we also have the corresponding equation for $\mathbf{k}$.

Extension to multiple dimensions

With relative ease RoPE can be extended into the multidimensional case. To represent two dimensions, two independent 1-dimensional rotary embeddings can be used. To implement this, we can split each of $\mathbf{q}$ and $\mathbf{k}$ in half and apply rotary piece-wise as follows:

$$₩begin{align*} ₩langle f(₩mathbf{q}, m, i),f(₩mathbf{k}, n, j) ₩rangle &= ₩langle f_1(₩mathbf{q}*{:d/2}, m),f_1(₩mathbf{k}*{:d/2}, n) ₩rangle + ₩langle f_2(₩mathbf{q}*{d/2:}, i),f_2(₩mathbf{k}*{d/2:}, j) ₩rangle ₩ &= g_1(₩mathbf{q}*{:d/2}, ₩mathbf{k}*{:d/2}, m - n) + g_2(₩mathbf{q}*{d/2:}, ₩mathbf{k}*{d/2:}, i - j) ₩ &= g(₩mathbf{q}, ₩mathbf{k}, m - n, i - j) ₩end{align*}$$

This formulation can also be further extended to data of an arbitrary number of dimensions. This sort of multi-dimensional relative coding would let us, for example, implement relative timing and relative pitch embeddings similar to Music Transformer [4] in a drastically simpler manner. More generally, we believe there is potentially a large class of invariances that first-principles positional codes like RoPE may enable us to capture.

How is this different from the sinusoidal embeddings used in "Attention is All You Need"?

A response many of us at EleutherAI had when first coming across this was "how does this differ from sinusoidal embeddings," so we feel it is worth discussing this comparison. There are two ways that rotary embeddings are different from sinusoidal embeddings:

1. Sinusoidal embeddings apply to each coordinate individually, while rotary embeddings mix pairs of coordinates
2. Sinusoidal embeddings add a $\cos(m\theta)$ or $\sin(m\theta)$ term, while rotary embeddings use a multiplicative factor.

Okay, what About in Practice?

After reading Jianlin Su's original blog posts [12, 13], we were curious how well such a first-principles approach to positional encoding would stack up against existing methods. Despite a tremendous number of papers that have come out claiming to improve the transformer architecture, very few approaches generalize well across codebases and tasks. However, we have found that rotary positional embeddings perform as well or better than other positional techniques in every architecture we have tried.

Implementation

A naive implementation of rotary positional embeddings would use the block diagonal matrix form shown earlier. In practice, implementing rotary positional embeddings this way is highly inefficient, and more optimized forms are readily available. The original implementations of RoPE are available in roformer and bert4keras.

Additionally, we have implemented rotary positional embeddings in x-transformers, GPT-Neo, GPT-NeoX, and Mesh Transformer JAX. Below are implimentations for PyTorch and JAX pulled from these codebases.

▶ GPT-NeoX (PyTorch)

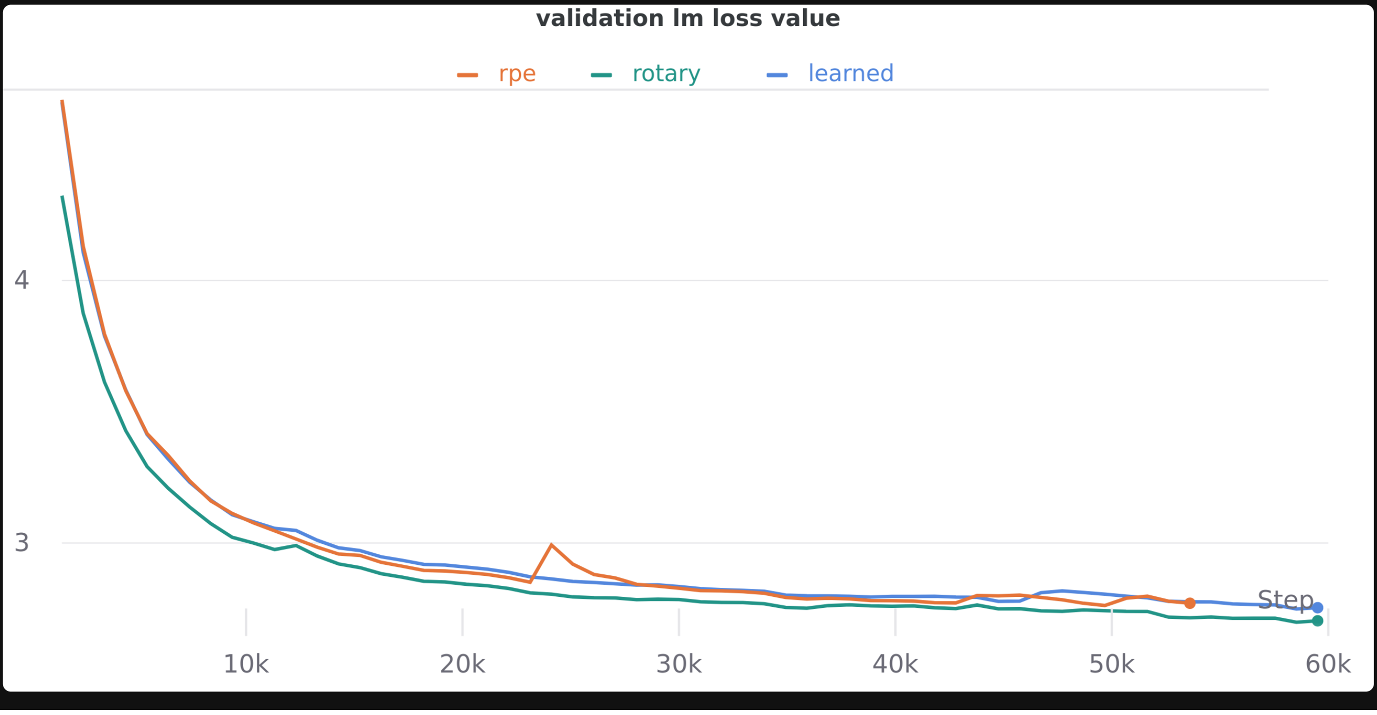
▶ Mesh Transformer JAX (JAX)

Experiments

We have found rotary embeddings to be effective for many varieties of attention.

Comparison against other PEs for Global attention

We conducted comparisons of rotary embeddings with learned absolute positional embeddings, used in GPT-3 [1], and the learned relative positional embeddings (henceforth RPE) used in T5 [10] using our GPT-Neox codebase. Comparisons were done using 125M parameter models with the same hyperparameters as the equally-sized model from [1]. Models were trained on OpenWebText2, a large and diverse dataset of online text. We see faster convergence of training and validation curves and a lower overall validation loss with a minimal decrease in throughput.



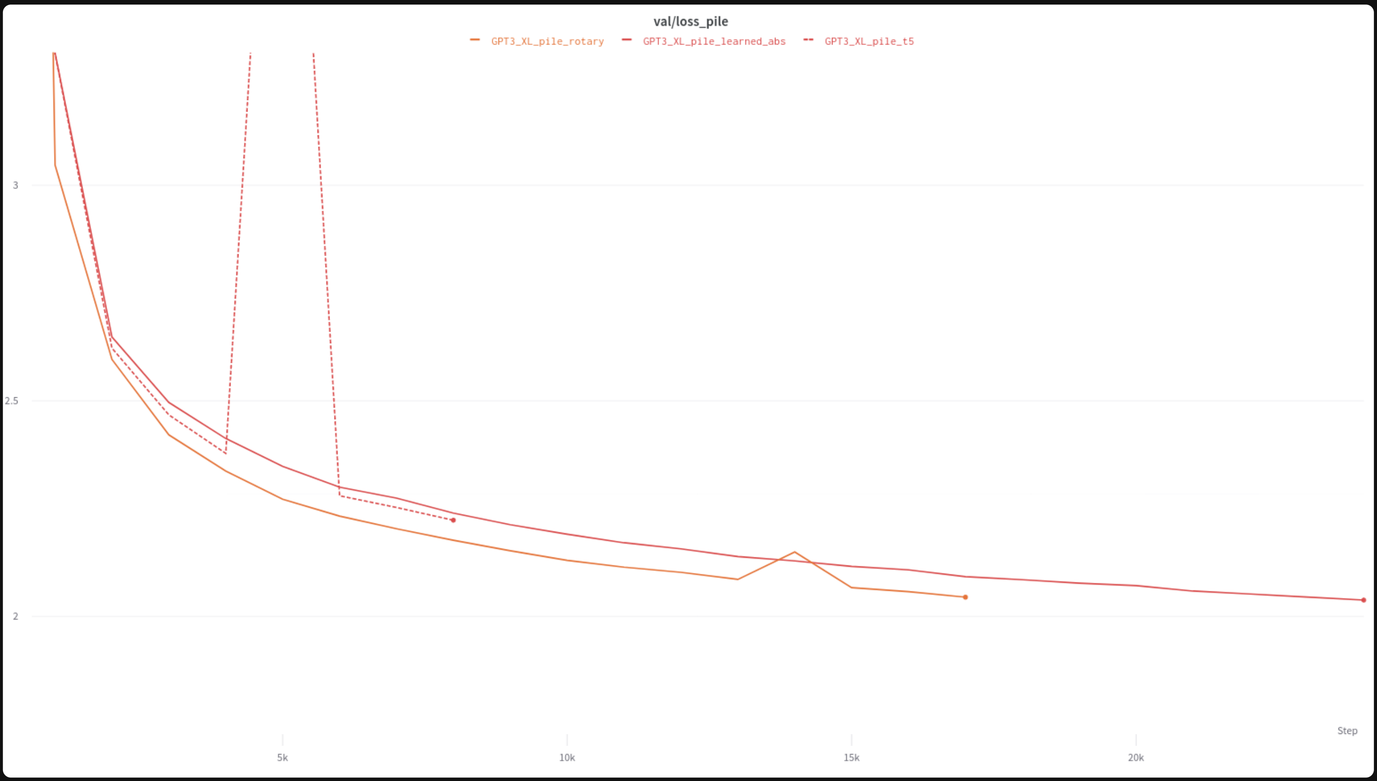
OWT2 validation loss with 150M parameter models in GPT-NeoX

Type	OWT2 Loss	OWT2 Ppl.
Learned Absolute	2.809	16.59
T5 RPE	2.801	16.46
Rotary	2.759	15.78

Final validation loss / ppl scores on OWT2 validation set at 55k steps (~30B tokens)

Billion+ parameter models

We additionally conducted additional larger scale experiments with the mesh-transformer-jax codebase and 1.4B parameter models, against baselines of learned absolute position embeddings and T5 RPE. Hyperparameters similar to GPT3's 1.3B model were used, with the dataset being the Pile [3]. A similar increase in convergence speed was observed as seen over learned absolute (~30%), and a smaller improvement (10-20%) was still seen over the T5 relative position encoding, demonstrating scalability into the billion parameter regimen. For full details, see here.



Pile validation loss with 1.5B parameter models

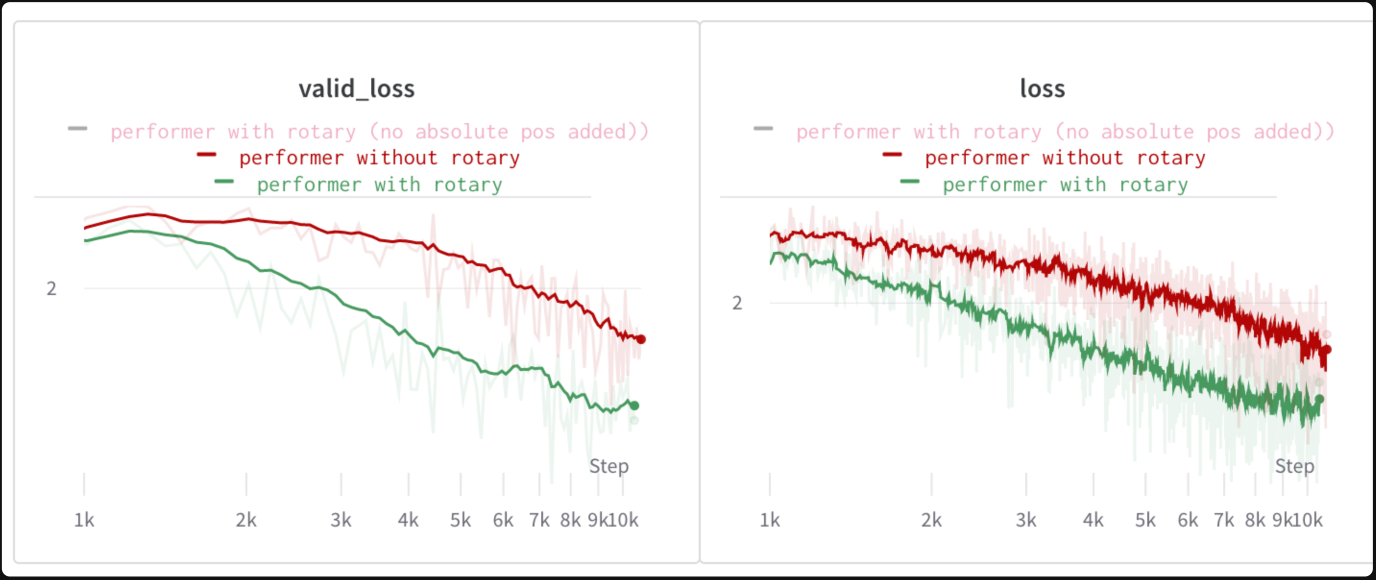
Type	Pile Loss	Pile Ppl.
Learned Absolute	2.240	9.393
T5 RPE	2.223	9.234
Rotary	2.173	8.784

Final validation loss / ppl scores on Pile validation set at 8k steps (~8B tokens)

Comparison against learned absolute for Performer

Performer [2] is an example of an alternative attention mechanism designed to avoid quadratic bottlenecks with respect to sequence lengths. We ran small scale tests of Performer on enwiki8, for 8 layer char-based transformers with 512 dimensions and 8 heads. These tests indicated that substituting rotary embeddings into the Performer leads to stark decreases in validation loss and to rapid convergence. Though these improvements do not close the gap between efficient and quadratic attention mechanisms, such a significant improvement makes mechanisms like Performer more attractive.

In smaller scale tests, we have also put RoPE head to head against other alternatives including the relative position method of Shaw et al. [11], TUPE [5], and position-infused attention [8], seeing positive results across the board.



Enwik8 validation/train loss with performer

Runtime

In general, we find that the runtime cost of rotary embeddings is fairly negligible. With the above implementation, we find that applying the rotary embeddings is naively about 4-5x the cost of applying additive positional embeddings. With the addition of a fusing optimizer like Torchscript, the runtime can be reduced to about 2-2.5x the runtime of additive positional embeddings. Concretely, for query and key tensors of shape $[2048, 16, 12, 64]$, applying rotary embeddings take 5.3 milliseconds, while applying additive positional embeddings takes 2.1 milliseconds.

Unlike standard positional embeddings, however, rotary embeddings must be applied at every layer. As large transformer models are typically dominated by matrix multiplies, we find that the overall overhead remains negligible. With fusion, we find that rotary embeddings impose a 1-3% overhead across a range of transformer sizes.

Conclusion

Rotary embeddings make it possible to implement relative attention in a straightforward and efficient manner, and we look forward to the work it inspires. Simple improvements to the transformer architecture that carry over robustly between different types of self-attention are few and far between [6].

Citation Information

To cite the RoPE methodology, please use:

```
@article{rope-paper,
  title={RoFormer: Enhanced Transformer with Rotary Position Embedding},
  author={Su, Jianlin and Lu, Yu and Pan, Shengfeng and Wen, Bo and Liu, Yunfeng},
  journal={arXiv preprint arXiv:2104.09864},
  year={2021}
}
```

To cite this blog post, please use:

```
@misc{rope-eleutherai,
  title = {Rotary Embeddings: A Relative Revolution},
  author = {Biderman, Stella and Black, Sid and Foster, Charles and Gao, Leo and Ha
  howpublished = \url{blog.eleuther.ai/},
  note = {[Online; accessed ]},
  year = {2021}
}
```

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking Attention with Performers. *arXiv preprint arXiv:2009.14794*, 2020.

[3] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*, 2021.

[4] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[5] Guolin Ke, Di He, and Tie-Yan Liu. Rethinking Positional Encoding in Language Pre-training. *arXiv preprint arXiv:2006.15595*, 2020.

[6] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, et al. Do Transformer Modifications Transfer Across Implementations and Applications? *arXiv preprint arXiv:2102.11972*, 2021.

[7] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient Large-Scale Language Model Training on GPU Clusters. *arXiv preprint arXiv:2104.04473*, 2021.

[8] Ofir Press, Noah A Smith, and Mike Lewis. Shortformer: Better Language Modeling using Shorter Inputs. *arXiv preprint arXiv:2012.15832*, 2020.

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[11] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. *arXiv preprint arXiv:1803.02155*, 2018.

[12] Jianlin Su. 让研究人员绞尽脑汁的 Transformer 位置编码. https://kexue.fm/archives/8130, 2021. [Online; accessed 18-April-2021].

[13] Jianlin Su. Transformer 升级之路：2、博采众长的旋转式位置编码. https://kexue.fm/archives/8265, 2021. [Online; accessed 18-April-2021].

[14] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[15] Hao Tan and Mohit Bansal. Vokenization: Improving Language Understanding with Contextualized, Visual-Grounded Supervision. *arXiv preprint arXiv:2010.06775*, 2020.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv preprint arXiv:1706.03762*, 2017.