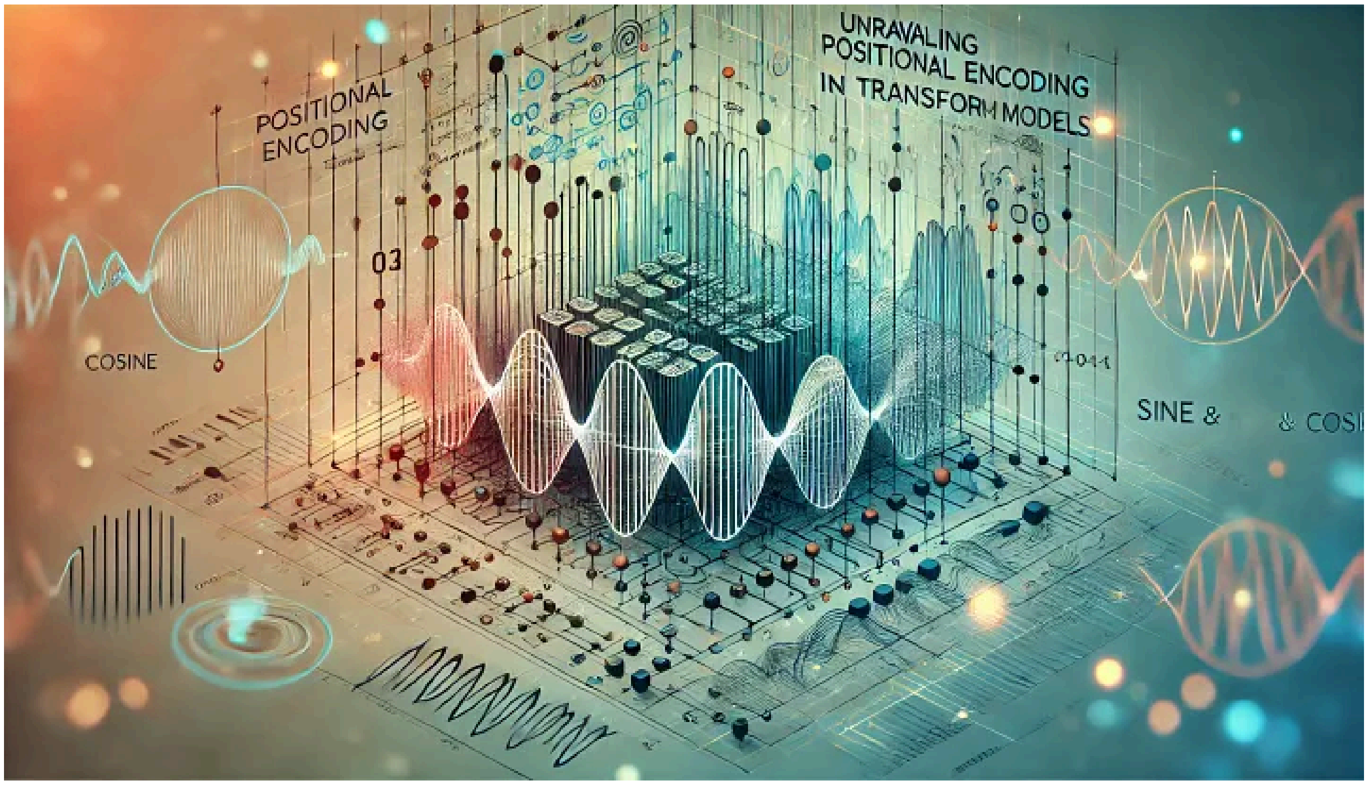
Positional Encoding Explained: A Deep Dive into Transformer PE

Nikhil Chowdary Paleti · Follow

Published in The Deep Hub · 16 min read · Jul 5, 2024



Dalle's creative interpretation of my article

Positional encoding is a crucial component of transformer models, yet it's often overlooked and not given the *attention* it deserves. Many articles present colorful diagrams of positional encodings but fail to provide a proper interpretation or deep understanding of why they work. This article aims to tackle that gap, offering a comprehensive exploration of positional encoding techniques

Prerequsites: Familiarity with transformer architecture and the self-attention mechanism will greatly enhance your understanding of the topics we'll explore in this article.

The Problem: Sequence Matters

In natural language processing and sequential data analysis, the order of elements is crucial. Consider a simple sentence: "The cat sat on the mat." Rearrange these words, and you might end up with nonsense or a completely different meaning.

Parallel processing is part of what makes Transformers so efficient, but it also means they don't inherently understand the concept of order. Unlike recurrent neural networks, which have an inbuilt mechanism to deal with sequence order, transformers do not use recurrence or convolution. Instead, they treat each data point as independent of the others. This poses a significant challenge when dealing with sequential data.

Positional Encoding: The Solution

To address this limitation, transformers employ a technique called positional encoding. It's the secret sauce that allows transformers to make sense of sequences. The key to solving this problem lies in finding a way to encode position information directly into the input embeddings.

There are various types of positional encoding, but in this article, we'll focus on two important and popular ones:

1. Absolute Positional Encoding
2. Rotary Position Embedding (RoPE)

Absolute Positional Encoding

While various methods exist for absolute positional encoding, the sinusoidal approach has gained widespread adoption due to its elegance and effectiveness. This method gained significant popularity through its use in the paper "Attention Is All You Need" by Vaswani et al., which introduced the Transformer architecture. The paper's success and the subsequent rise of Transformer models have made sinusoidal positional encoding a cornerstone technique in the field.

The sinusoidal positional encoding method works as follows:

- It generates a unique vector for each position in the sequence, using a combination of sine and cosine functions.
- The encoding vector has the same dimensionality as the token embeddings, allowing them to be simply added together.
- Different dimensions in the encoding vector correspond to sinusoids of different frequencies, creating a spectrum from high to low frequencies.
- This approach allows the model to easily attend to relative positions, as the encoding for any fixed offset can be represented as a linear function of the encoding at a given position.

The Intuition: From Bits to Waves

To develop our intuition about absolute positional encoding, let's start with something fundamental: how we represent numbers in binary.

Binary Representation

Consider how we count in binary:

```
0:  0 0 0 0    8:  1 0 0 0
1:  0 0 0 1    9:  1 0 0 1
2:  0 0 1 0    10: 1 0 1 0
3:  0 0 1 1    11: 1 0 1 1
4:  0 1 0 0    12: 1 1 0 0
5:  0 1 0 1    13: 1 1 0 1
6:  0 1 1 0    14: 1 1 1 0
7:  0 1 1 1    15: 1 1 1 1
```

Notice the pattern in how each bit (column) changes:

- The rightmost bit (Least Significant Bit) alternates with every number (frequency: 1/2)
- The second bit from the right alternates every two numbers (frequency: 1/4)
- The third bit alternates every four numbers (frequency: 1/8) And so on…

This pattern of different frequencies is the key to positional encoding. But instead of using discrete bits, we can use something smoother: sine and cosine waves.

Sinusoidal Positional Encoding

In the original transformer paper, the authors proposed using a combination of sine and cosine functions at different frequencies to encode position.

Here's how it works:

1. For each position in the sequence, we generate a vector of numbers.
2. Each number in this vector is calculated using either a sine or cosine function.
3. We use different frequencies for different dimensions of the vector.

$$PE_{(pos,i)} = \begin{cases} \sin\left(\dfrac{pos}{10000^{i/d_{model}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\dfrac{pos}{10000^{(i-1)/d_{model}}}\right) & \text{if } i \text{ is odd} \end{cases}$$

The positional encodings have the same dimension ***d_model*** as the embeddings, so that the two can be summed, ***pos*** is the position and ***i*** is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid.

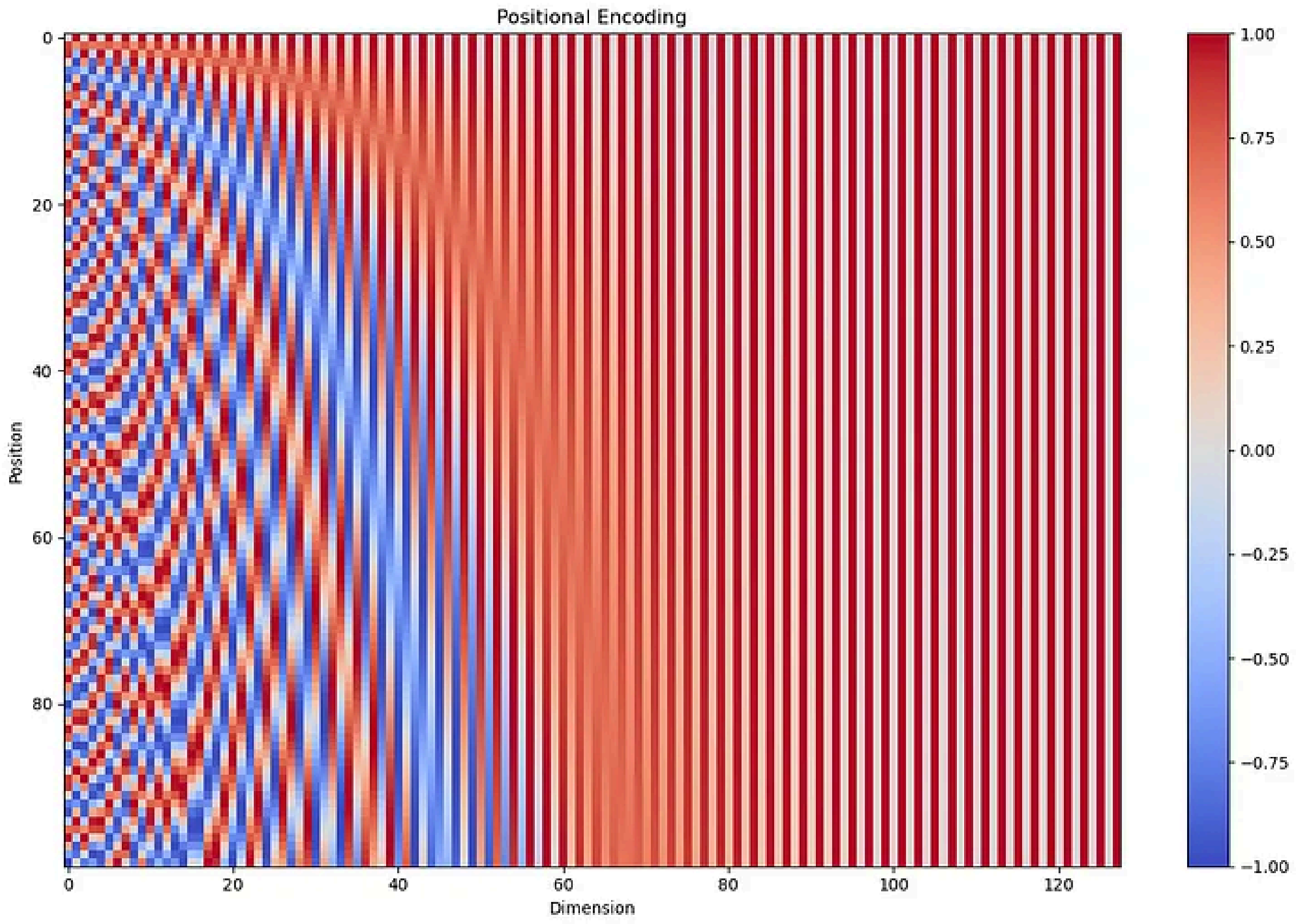
Here's the Python code that implements this:

```
import numpy as np
import matplotlib.pyplot as plt

def sinusoidal_positional_encoding(max_position, d_model):
    position = np.arange(max_position)[:, np.newaxis]
    # The original formula pos / 10000^(2i/d_model) is equivalent to pos * (1 /
    # I use the below version for numerical stability
    div_term = np.exp(np.arange(0, d_model, 2) * -(np.log(10000.0) / d_model))

    pe = np.zeros((max_position, d_model))
    pe[:, 0::2] = np.sin(position * div_term)
    pe[:, 1::2] = np.cos(position * div_term)

    return pe
```

```
max_position = 100  # Maximum sequence length
d_model = 128        # Embedding dimension

pe = sinusoidal_positional_encoding(max_position, d_model)

plt.figure(figsize=(12, 8))
plt.imshow(pe, cmap='coolwarm', aspect='auto', vmin=-1, vmax=1)
plt.colorbar()
plt.title('Sinusoidal Positional Encoding')
plt.xlabel('Dimension')
plt.ylabel('Position')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
dimensions = [0, 21]
for d in dimensions:
    plt.plot(pe[:, d], label=f'Dim {d}')
plt.legend()
plt.title('Sinusoidal Positional Encoding for Specific Dimensions')
plt.xlabel('Position')
plt.ylabel('Value')
plt.tight_layout()
plt.show()
```

When you run this code, you'll see two plots:

1. Heat map



This plot is analogous to our binary representation table, but with a continuous spectrum instead of discrete 0s and 1s:

1. Each row represents a token in the sequence, similar to how each row in our binary table represented a number.

2. Each column corresponds to a dimension in our tokens encoding, analogous to the bit positions in binary.

3. The colors represent values oscillating between -1 (blue) and 1 (red), which is a continuous version of the 0s and 1s in binary.

Key Observations:

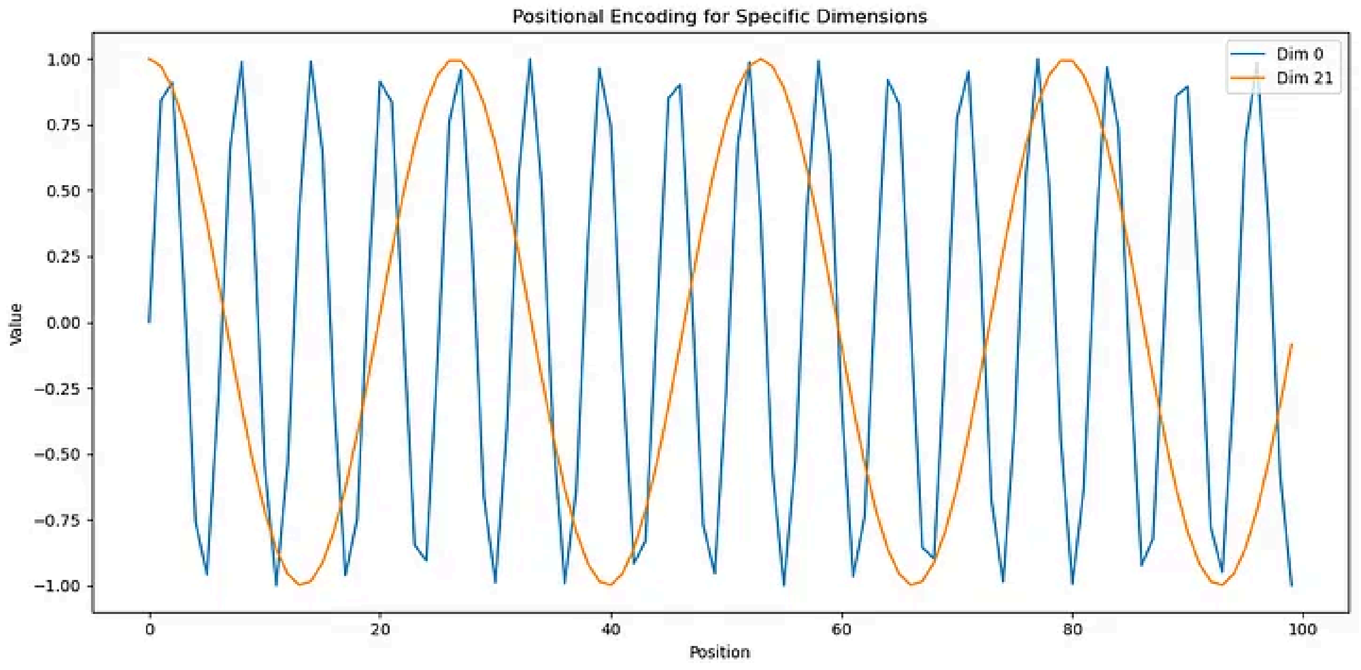
— The first row (position 0) is like our binary "0000", serving as the starting point.

— As we move down the rows (increasing positions), we see patterns of color changes, similar to how bits flip in binary counting.

— The leftmost columns (lower dimensions) change rapidly, like the least significant bits in binary.

— The rightmost columns (higher dimensions) change more slowly, analogous to the most significant bits in binary.

2. Line Plot



The line plot focuses on specific dimensions and how they change with position, directly mirroring our binary analogy:

1. Each line represents a specific dimension, similar to tracking a single column (bit position) in our binary table.

2. The x-axis shows the position in the sequence, equivalent to counting up in our binary example.

Key Observations:

— Dimension 0 (blue line) oscillates rapidly, changing with every position. This is exactly like the rightmost bit in binary, which flips with every count.

— Dimension 21 (orange line) changes much more slowly, similar to how the bits further left in our binary representation flip less frequently.

Relative Positioning

The authors of the original Transformer paper stated:

> *We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE(pos+k) can be represented as a linear function of PE(pos).*

But what does this mean in practice? Let's break it down.

At the heart of this property lies a beautiful mathematical relationship. For each sine-cosine pair corresponding to a frequency ω_k, there exists a linear transformation M that can shift the position by any fixed offset ϕ. In mathematical terms:

$$M \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (t + \phi)) \\ \cos(\omega_k \cdot (t + \phi)) \end{bmatrix}$$

This equation tells us that we can represent the positional encoding of any position (t + ϕ) as a linear transformation of the encoding at position t. The key here is that this transformation M is independent of t, meaning it works the same way regardless of the absolute position we're starting from.

The Proof (For the Curious Minds)

1. We start by expanding the right side of the equation using the trigonometric addition formulas.

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot t)\cos(\omega_k \cdot \phi) + \cos(\omega_k \cdot t)\sin(\omega_k \cdot \phi) \\ \cos(\omega_k \cdot t)\cos(\omega_k \cdot \phi) - \sin(\omega_k \cdot t)\sin(\omega_k \cdot \phi) \end{bmatrix}$$

2. This expansion gives us two equations, one for sin(ω_k · (t + ϕ)) and one for cos(ω_k · (t + ϕ)).

$$\begin{aligned} u_1 \sin(\omega_k \cdot t) + v_1 \cos(\omega_k \cdot t) &= \cos(\omega_k \cdot \phi)\sin(\omega_k \cdot t) + \sin(\omega_k \cdot \phi)\cos(\omega_k \cdot t) \\ u_2 \sin(\omega_k \cdot t) + v_2 \cos(\omega_k \cdot t) &= -\sin(\omega_k \cdot \phi)\sin(\omega_k \cdot t) + \cos(\omega_k \cdot \phi)\cos(\omega_k \cdot t) \end{aligned}$$

3. By solving these equations, we find that the transformation matrix M is:

$$M_{\phi,k} = \begin{bmatrix} \cos(\omega_k \cdot \phi) & \sin(\omega_k \cdot \phi) \\ -\sin(\omega_k \cdot \phi) & \cos(\omega_k \cdot \phi) \end{bmatrix}$$

4. Interestingly, this matrix looks very similar to a rotation matrix in linear algebra.

Why this matters?

- **Efficient Computation of Relative Positions:** The model can compute the positional encoding for any relative position by applying a simple linear transformation to the encoding of the current position. This allows the attention mechanism to easily focus on relative distances between tokens.
- **Ease of Learning Position-Relative Patterns:** While not providing true translation invariance, this property makes it easier for the model to learn patterns that depend on relative positions rather than absolute positions. For example, it can more easily learn that "not" typically modifies the meaning of nearby words, regardless of where this pattern appears in the sentence.

Why Sinusoidal Encoding Works:

- Uniqueness: Each position in the sequence gets a unique pattern of high and low frequency components. This ensures that the model can distinguish between different positions.
- Relative Position Information: The sinusoidal functions have a useful property where the positional encoding for a position can be expressed as a linear function of the encoding at another position. This allows the model to more easily learn to attend to relative positions.
- Bounded Range: The values of sine and cosine functions are always between -1 and 1. This ensures that the positional encodings don't grow or shrink excessively for very long sequences, which helps maintain numerical stability.

Encoded Embedding Visualization

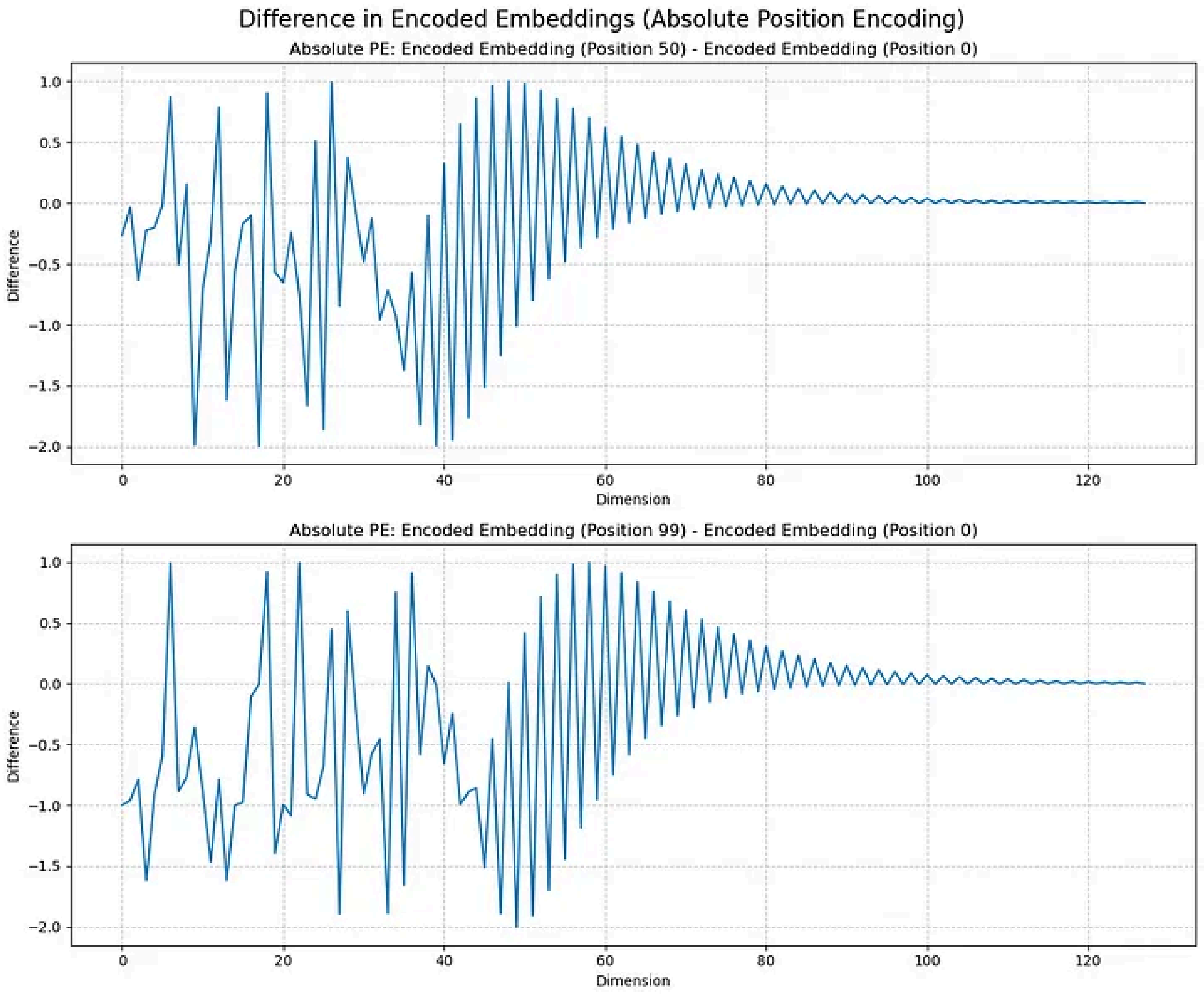
To gain deeper insights into how absolute positional encoding affects input embeddings, we'll visualize the differences between encoded embeddings at various positions. This analysis will help us understand where and how position information is incorporated into the embedding.

Methodology

1. We create a dummy embedding vector of size 128, representing a single token.
2. We then encoded this embedding with positional information for positions 0, 50, and 99.
3. These positions were chosen to represent the same word appearing at the beginning, middle, and end of a sequence, respectively.
4. We calculated the differences between these encoded embeddings to isolate the positional information.

```
np.random.seed(42)

dummy_embedding = np.random.randn(d_model)

positions = [0, 50, 99]

encoded_embeddings = [dummy_embedding + pe[pos] for pos in positions]

fig, axs = plt.subplots(2, 1, figsize=(12, 10))
fig.suptitle('Difference in Encoded Embeddings (Absolute Position Encoding)', fo

diff_50 = encoded_embeddings[1] - encoded_embeddings[0]
axs[0].plot(diff_50)
axs[0].set_title('Absolute PE: Encoded Embedding (Position 50) - Encoded Embeddi
axs[0].set_xlabel('Dimension')
axs[0].set_ylabel('Difference')
axs[0].grid(True, linestyle='--', alpha=0.7)
```

```
diff_99 = encoded_embeddings[2] - encoded_embeddings[0]
axs[1].plot(diff_99)
axs[1].set_title('Absolute PE: Encoded Embedding (Position 99) - Encoded Embeddi
axs[1].set_xlabel('Dimension')
axs[1].set_ylabel('Difference')
axs[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



key observations

- **Localization of Positional Information**: The most significant differences appear in the initial dimensions of the embedding. This suggests that positional information is primarily encoded in these early dimensions.
- **Decreasing Amplitude**: As we move to higher dimensions, the amplitude of the differences decreases. This indicates that later dimensions are less affected by positional encoding.
- **Token Information Preservation**: The minimal differences in later dimensions suggest that these dimensions primarily retain information about the token itself, with little influence from position.
- **Periodic Pattern**: The differences exhibit a periodic pattern, which is a characteristic of the sinusoidal functions used in the encoding

Implications

1. **Dimensional Roles:** The embedding dimensions seem to have specialized roles, with earlier dimensions capturing more positional information and later dimensions focusing on token semantics.
2. **Model Attention:** This structure might influence how the model attends to different aspects of the input. The model can focus on early dimensions for positional relationships and later dimensions for token-specific information.
3. **Information Balancing:** The encoding scheme balances the preservation of token information with the addition of positional context, allowing the model to leverage both aspects in its processing.

Rotary Position Embedding (RoPE)

While sinusoidal encoding has been widely successful, researchers have continued to innovate. One development is Rotary Position Embedding (RoPE), introduced by Su et al. in their paper "RoFormer: Enhanced Transformer with Rotary Position Embedding".

The Basics of RoPE

Rotary Position Embedding takes a fundamentally different approach to encoding positional information compared to traditional methods:

1. Instead of adding a separate positional encoding vector, RoPE applies a rotation to the existing token embeddings.
2. The rotation angle is a function of both the token's position in the sequence and the dimension of the embedding.
3. This rotation preserves the norm of the embeddings while encoding positional information.

The Intuition Behind RoPE

To understand the elegance of Rotary Position Embeddings (RoPE), we need to dive into the self-attention layer.

1. Sequence Representation

1. We have a sequence of tokens (words or subwords) w_1, w_2, …, w_l, where the length of this sequence is L.
2. Each token gets transformed into a vector (called an embedding) in a high-dimensional space. Let's call the dimension of this space |D|.
3. The result is a set of vectors x_1, x_2, …, x_l in this |D|-dimensional space.

$$x_1, x_2, \cdots, x_L \in \mathbb{R}^{|D|}$$

2. Attention Computation Basics

In transformer models, each position (*m*) in a sequence has both a query (q_m) and a key vector (k_m) .

These vectors are created using two functions, f_q and f_k:

$$\begin{aligned} q_m &= f_q(x_m, m) \in \mathbb{R}^{|L|} \\ k_m &= f_k(x_m, m) \in \mathbb{R}^{|L|} \end{aligned}$$

When computing the attention matrix, we aim to encode two essential characteristics:

1. **Token Similarity**: Tokens with similar embeddings should have a higher attention score. For example, “cat” and “dog” might have a higher score as they appear in similar contexts.
2. **Positional Proximity:** Words that are closer together in the sequence should generally have a higher score, as they’re more likely to be related.

To figure out how much one word should “pay attention” to another, we use something called an attention score. For any pair of positions m and n, it looks like this: The dot product of query q_m and key k_n. This dot product encapsulates both token similarity and positional information.

$$\text{softmax}\left(\frac{q_m^T k_n}{\sqrt{|D|}}\right)$$

In this dot product : (q · k = ||q|| ||k|| cos(θ))

- **Magnitude :** This contributes to the token similarity. The similarity between the magnitudes of q and k (denoted as ||q|| · ||k||) corresponds to the token embedding similarity.
- **Angle:** The angles of Q and K (θ_m and θ_n) contribute to the positional similarity.

This design has an important property:

- The angle component depends only on the position of the vector in the sequence and is independent of the actual token embeddings.
- The token similarity is independent of the angle.

RoPE's Approach

RoPE leverages this view in a clever way:

1. Instead of adding separate positional encodings, RoPE rotates the query and key vectors based on their position in the sequence.
2. The rotation preserves the magnitude (maintaining token similarity) while encoding positional information in the angle.

When computing the dot product between rotated vectors:

- The magnitudes still capture token similarity.
- The relative angle between the vectors captures positional proximity.

This approach allows RoPE to seamlessly integrate both token and positional information into a single operation, making it more efficient and potentially more effective than traditional positional encoding methods.

Now, about rotation, RoPE says: "Hey, let's make these attention scores depend only on how far apart the words are, not their absolute positions."

In math terms, we want:

$$f_q(x_m, m)^T f_k(x_n, n) = g(x_m, x_n, m - n)$$

This equation is saying: the attention between positions m and n should only depend on their content (x_m and x_n) and how far apart they are (m — n).

It's like caring more about whether "not" comes right before "happy" rather than if they're the 5th and 6th words in a sentence.

RoPE strikes a balance between encoding absolute positional information and facilitating the computation of relative positional relationships:

1. Each token's embedding retains information about its absolute position through the applied rotation.

2. The interaction between rotated embeddings in the attention mechanism makes it easier for the model to compute and utilize relative positional information.
3. This approach allows the model to be aware of both absolute and relative positions, potentially leading to more nuanced understanding of sequence structure.

Rope Formulation

(Derivation is skipped to not make this article more lengthy than it already is, the ROFORMER paper does a good job of explaining it or you can check this explanation: https://pub.towardsai.net/an-in-depth-exploration-of-rotary-position-embedding-rope-ac351a45c794)

Mathematical Formulation in 2D

For a query vector

$$q_m = \begin{pmatrix} q_m^{(1)} \\ q_m^{(2)} \end{pmatrix} \text{ at position } m$$

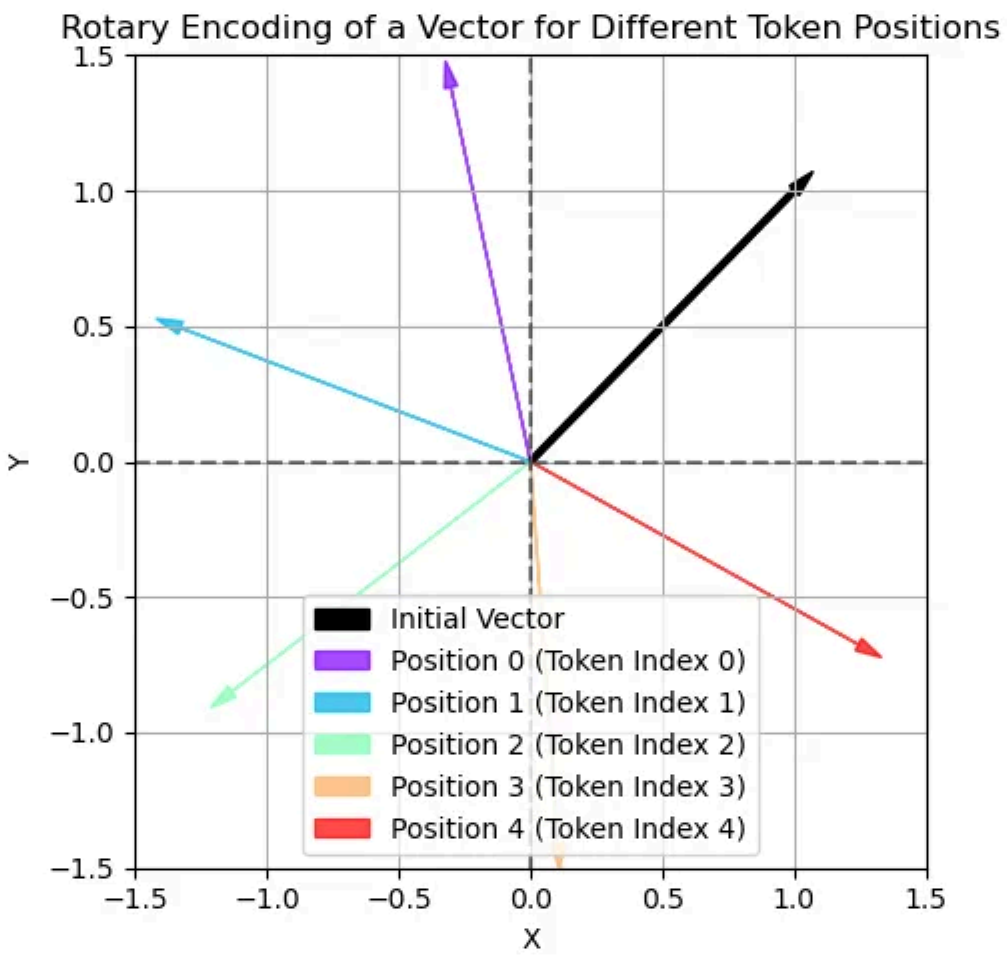
the rotation matrix R_{θ,m} is a 2x2 matrix formulated as:

$$R_{\Theta,m} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$

where $\theta \in R$ is a preset non-zero constant.

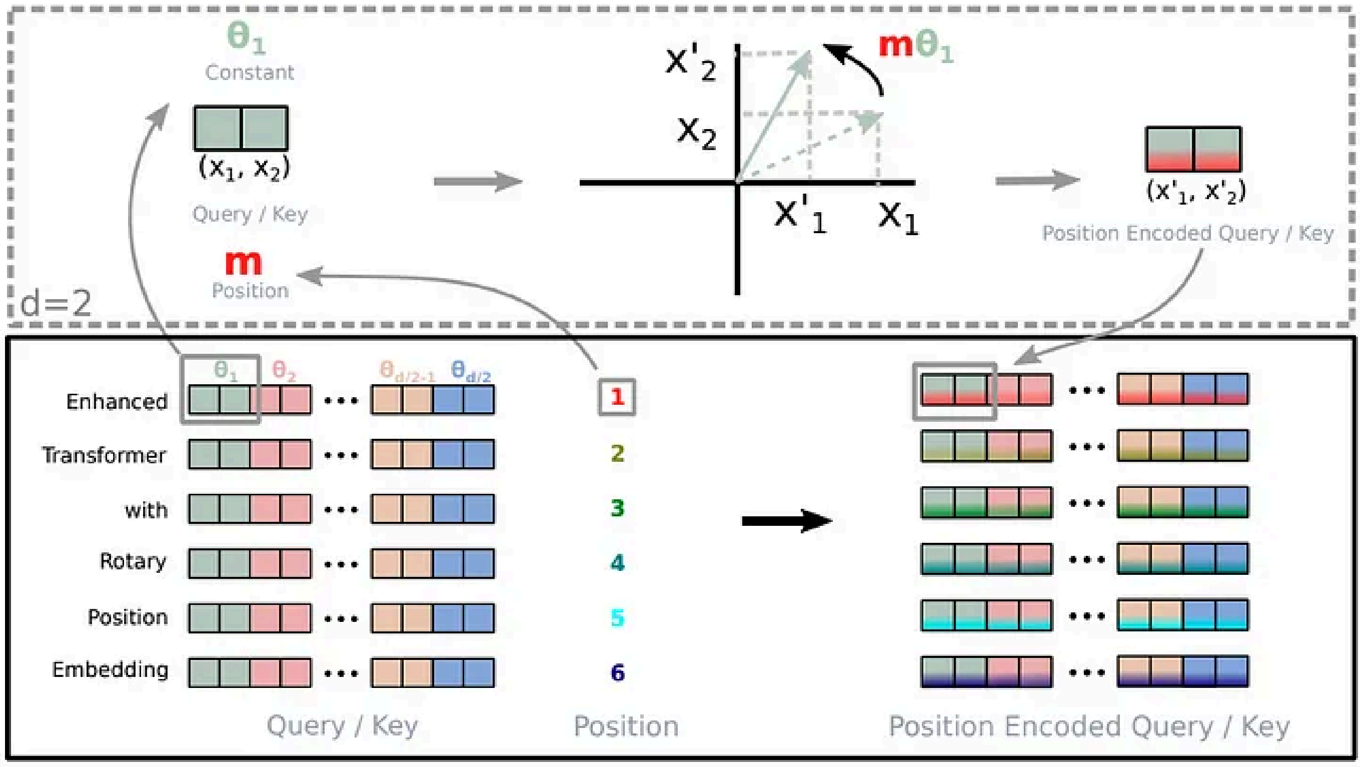
We can use this matrix to rotate a vector q, to obtain the new rotated vector q^.

Here is a visualization of the rotation applied to the 2D vector q. The initial vector is displayed in black and the vectors for various positions (various m) is displayed:

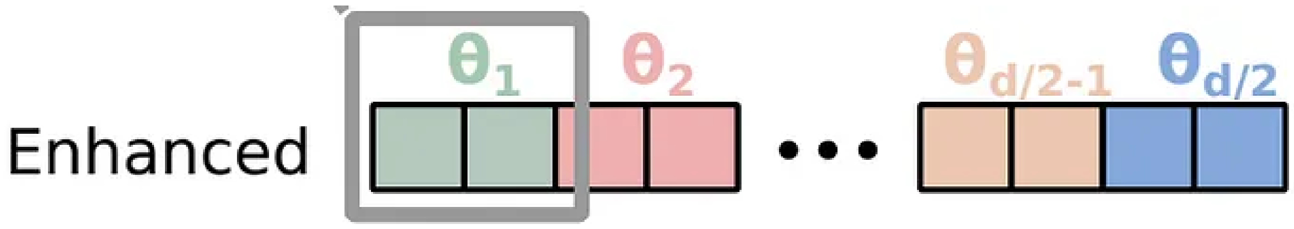


2D embedding represented at different positions

General Mathematical Formulation



— we divide the d-dimension (embedding dimension) space into d/2 sub-spaces and apply rotations individually.



The embedding vector for “Enhanced” is divided into d/2 subspaces

Each subspace is rotated by the rotation matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

where,

$$\Theta = \theta_i = 10000^{-2i/d}, i \in [1, 2, 3, 4..\frac{d}{2}]$$

Key Observation:

- Notice how theta is decaying as it increases
- Consider the query/key vector for enhanced that's shown above, the sub space that is represented in light green (theta 1) will rotate a lot compared to the next sub space and towards the end, the subspaces will have negligible rotation.

Here's the python code for RoPE:

In the below code, we create 3D matrix and each slice contains the rotaion matrix for poisition "m". i.e., the first query/key vector gets multiplied with the first slice, the second vector with the second slice and so on.. for the rotations.

```
import numpy as np
import matplotlib.pyplot as plt

def get_rotary_matrix(context_len: int, d_model: int) -> np.ndarray:
    """
    Generate the Rotary Matrix for ROPE
    Args:
        context_len (int): context len
        d_model (int): embedding dim
    Returns:
        np.ndarray: the rotary matrix of dimension context_len x d_model x d_mod
    """
    R = np.zeros((context_len, d_model, d_model))
    positions = np.arange(1, context_len + 1)[:, np.newaxis]
    # Create matrix theta (shape: context_len x d_model // 2)
    slice_i = np.arange(0, d_model // 2)
    theta = 10000. ** (-2.0 * slice_i.astype(float) / d_model)
    m_theta = positions * theta
    # Create sin and cos values
    cos_values = np.cos(m_theta)
    sin_values = np.sin(m_theta)
    # Populate the rotary matrix R using advanced indexing
    R[:, 2*slice_i, 2*slice_i] = cos_values
    R[:, 2*slice_i, 2*slice_i+1] = -sin_values
    R[:, 2*slice_i+1, 2*slice_i] = sin_values
    R[:, 2*slice_i+1, 2*slice_i+1] = cos_values
    return R
```

Efficient Implementation

Considering how sparse the rotation matrix is, the authors have come up with this efficient way of computing RoPE:

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

You can try implementing this in python as an exercise.

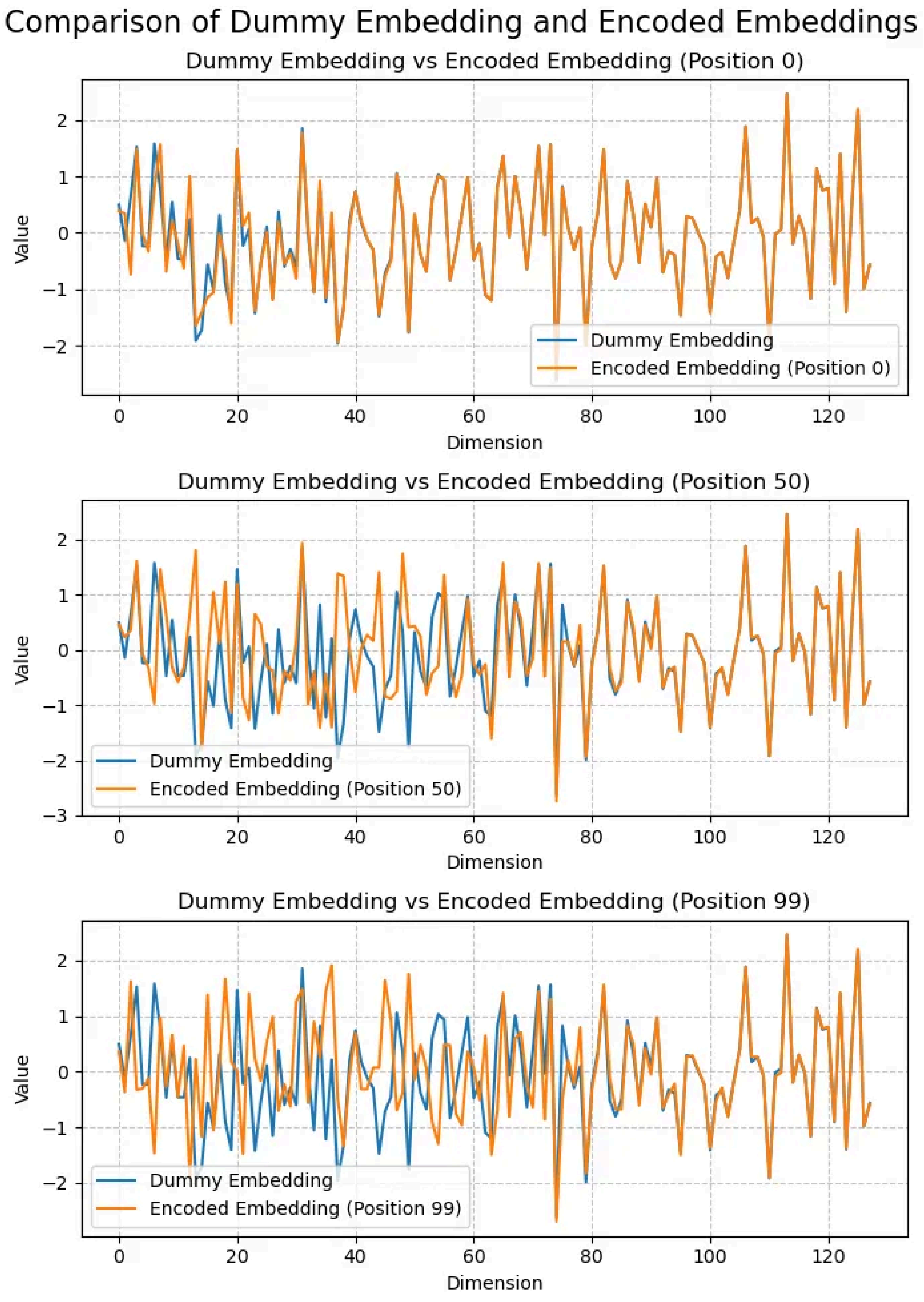
Visualizing RoPE

remember how different sub spaces were rotating at different rates, here let's visualize it and explore what it means.

```
context_len = 100
d_model = 128
R = get_rotary_matrix(context_len, d_model)

np.random.seed(42)

# Generate dummy embedding
dummy_embedding = np.random.randn(d_model)

# Positions to encode
positions = [0, 50, 99]

# Create encoded embeddings
encoded_embeddings = [R[pos] @ dummy_embedding for pos in positions]

# Create subplots
fig, axs = plt.subplots(3, 1, figsize=(7, 10))
fig.suptitle('Comparison of Dummy Embedding and Encoded Embeddings', fontsize=16

# Plot dummy embedding and encoded embedding at position 0
axs[0].plot(dummy_embedding, label='Dummy Embedding')
axs[0].plot(encoded_embeddings[0], label='Encoded Embedding (Position 0)')
axs[0].set_title('Dummy Embedding vs Encoded Embedding (Position 0)')
axs[0].set_xlabel('Dimension')
axs[0].set_ylabel('Value')
axs[0].legend()
axs[0].grid(True, linestyle='--', alpha=0.7)

# Plot dummy embedding and encoded embedding at position 50
axs[1].plot(dummy_embedding, label='Dummy Embedding')
axs[1].plot(encoded_embeddings[1], label='Encoded Embedding (Position 50)')
axs[1].set_title('Dummy Embedding vs Encoded Embedding (Position 50)')
axs[1].set_xlabel('Dimension')
axs[1].set_ylabel('Value')
axs[1].legend()
axs[1].grid(True, linestyle='--', alpha=0.7)

# Plot dummy embedding and encoded embedding at position 99
axs[2].plot(dummy_embedding, label='Dummy Embedding')
axs[2].plot(encoded_embeddings[2], label='Encoded Embedding (Position 99)')
axs[2].set_title('Dummy Embedding vs Encoded Embedding (Position 99)')
axs[2].set_xlabel('Dimension')
```

```
    axs[2].set_ylabel('Value')
    axs[2].legend()
    axs[2].grid(True, linestyle='--', alpha=0.7)

    plt.tight_layout()
    plt.show()
```

Observe how the orange line (rotated vector) changes at position 0, 50 and 99:


Comparison of Dummy Embedding and Encoded Embeddings
Dummy Embedding vs Encoded Embedding (Position 0)
Dummy Embedding
Encoded Embedding (Position 0)
Dummy Embedding vs Encoded Embedding (Position 50)
Dummy Embedding
Encoded Embedding (Position 50)
Dummy Embedding vs Encoded Embedding (Position 99)
Dummy Embedding
Encoded Embedding (Position 99)
Value
Dimension
0
20
40
60
80
100
120
−3
−2
−1
0
1
2

As we discussed above,

— The initial subspaces rotate a lot compared to the ending sub spaces.

Let us look at the observations of Absolute positional encoding and see if they hold true here also:

key observations

- **Localization of Positional Information**: The most significant differences appear in the initial dimensions of the embedding. This suggests that positional information is primarily encoded in these early dimensions. (TRUE HERE ALSO)
- **Decreasing Amplitude**: As we move to higher dimensions, the amplitude of the differences decreases. This indicates that later dimensions are less affected by positional encoding. (TRUE HERE ALSO)
- **Token Information Preservation**: The minimal differences in later dimensions suggest that these dimensions primarily retain information about the token itself, with little influence from position. (TRUE HERE ALSO)

Implications

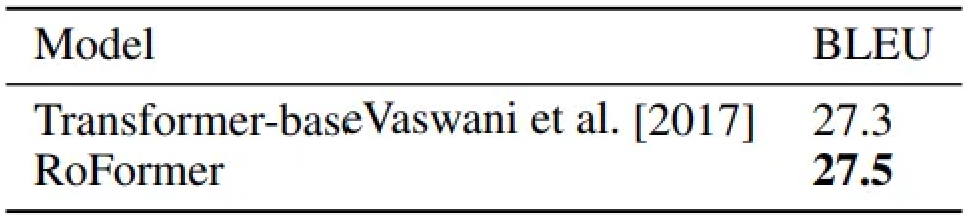
1. **Dimensional Roles:** The embedding dimensions seem to have specialized roles, with earlier dimensions capturing more positional information and later dimensions focusing on token semantics.
2. **Model Attention:** This structure might influence how the model attends to different aspects of the input. The model can focus on early

dimensions for positional relationships and later dimensions for token-specific information.

3. **Information Balancing:** The encoding scheme balances the preservation of token information with the addition of positional context, allowing the model to leverage both aspects in its processing.

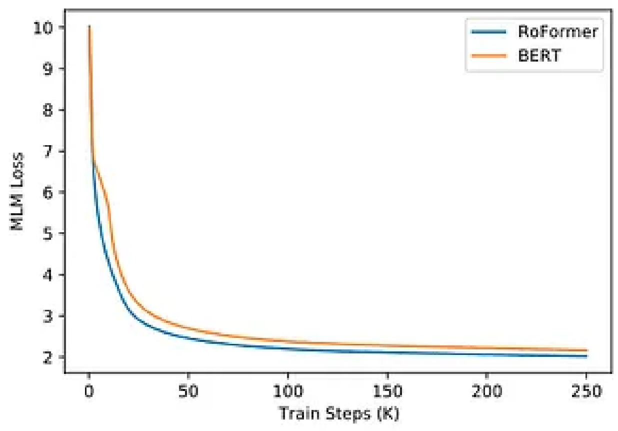
Sinusoidal and RoPE Encodings in Practice

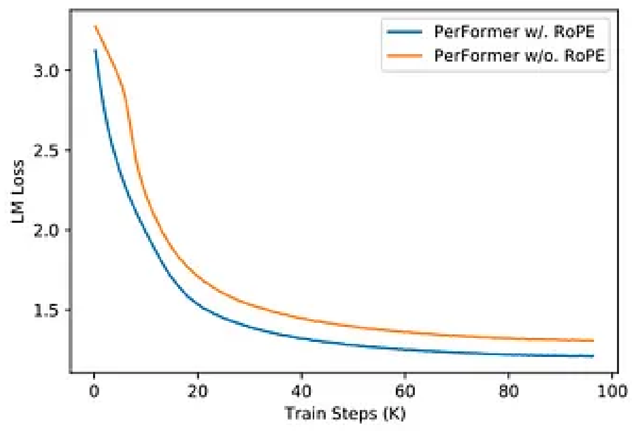
looking at the results presented in RoFormer paper,



Model	BLEU
Transformer-baseVaswani et al. [2017]	27.3
RoFormer	**27.5**

The RoFormer gives better BLEU scores compared to its baseline alternative Vaswani et al. [2017] on the WMT 2014 English-to-German translation task





Rotaty positional encoding seems to have a better convergence rate.

- Lot of the recent models, Gemma 2, LLama 3 etc., use the RoPE encodings.

Article by Nikhil Chowdary Paleti

Nikhil Chowdary Paleti | AI Engineer and Researcher

Portfolio of Nikhil Chowdary Paleti, AI Engineer and Researcher specializing in Machine Learning, Data Science, and…

nikhil-paleti.github.io

https://www.linkedin.com/in/nikhil-paleti/

https://x.com/nikhil2362

References

- https://arxiv.org/abs/1706.03762 — Attention is All you need
- https://arxiv.org/abs/2104.09864 — RoFormer: Enhanced Transformer with Rotary Position Embedding
- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- https://blog.eleuther.ai/rotary-embeddings/
- https://blog.eleuther.ai/yarn/
- https://afterhoursresearch.hashnode.dev/rope-rotary-positional-embedding
- https://www.youtube.com/watch?v=GQPOtyITy54

```
@article{paleti2024:positionalencoding,
  title   = "Positional Encoding Explained: A Deep Dive into Transformer PE",
```