transformer, how many position embedding methods? please use detailed example and python code to explain

Great question! 🚀

In Transformers, **positional information** is essential because self-attention is **permutation-invariant** — it doesn't know the order of tokens unless we explicitly encode it.

There are several **positional embedding methods** used in practice. Let's go step by step.

🔑 Main Types of Position Embeddings

1. Absolute Position Embeddings

Attach position information **directly to each token embedding**.

(a) Learned Absolute Position Embedding

- Treat each position as an embedding (just like words).
- Example: BERT uses this.

Python code example:

python Copy Edit

```
import torch
import torch.nn as nn

class LearnedPositionalEncoding(nn.Module)
    def __init__(self, max_len: int, d_model: int):
```

Python code example:

python

```python
import torch
import torch.nn as nn

class LearnedPositionalEncoding(nn.Module):
    def __init__(self, max_len: int, d_model: int):
        super().__init__()
        self.pos_embedding = nn.Embedding(max_len, d_model)

    def forward(self, x):
        """
        x: shape [batch, seq_len, d_model]
        """
        batch_size, seq_len, d_model = x.shape
        positions = torch.arange(seq_len, device=x.device).unsqueeze(0)  # [1, seq_len]
        pos_embed = self.pos_embedding(positions)  # [1, seq_len, d_model]
        return x + pos_embed

# Example
x = torch.randn(2, 5, 16)  # [batch=2, seq_len=5, d_model=16]
pe = LearnedPositionalEncoding(max_len=50, d_model=16)
print(pe(x).shape)  # [2, 5, 16]
```

(b) Sinusoidal Absolute Position Embedding

- Fixed, not learned.
- Encodes position using sine and cosine functio ıt different frequencies.
- Original Transformer (Vaswani et al. 2017) used this.

(b) Sinusoidal Absolute Position Embedding

- Fixed, not learned.
- Encodes position using sine and cosine functions at different frequencies.
- Original Transformer (Vaswani et al. 2017) used this.

Formula:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Python code example:

python Copy Edit

```python
import math

def sinusoidal_position_encoding(seq_len, d_model):
    pos = torch.arange(seq_len, dtype=torch.float).unsqueeze(1)  # [seq_len, 1]
    i = torch.arange(d_model, dtype=torch.float).unsqueeze(0)    # [1, d_model]
    angle_rates = 1 / torch.pow(10000, (2 * (i//2)) / d_model)
    angle_rads = pos * angle_rates
    # apply sin to even indices; cos to odd indices
    pe = torch.zeros(seq_len, d_model)
    pe[:, 0::2] = torch.sin(angle_rads[:, 0::2])
    pe[:, 1::2] = torch.cos(angle_rads[:, 1::2])
    return pe

pe = sinusoidal_position_encoding(seq_len=10, d_model=16)
print(pe.shape)  # [10, 16]
```

2. Relative Position Embeddings

Instead of absolute positions, encode **the relative distance between tokens.**

- Used in Transformer-XL, T5, DeBERTa.
- Better for generalizing to longer sequences.

Example: relative attention bias (T5 style)

$$\text{Attention}(Q, K) = \frac{QK^\top}{\sqrt{d_k}} + b_{rel}(i - j)$$

where b_{rel} depends on the distance between positions i and j.

Python code example:

```python
class RelativePositionalBias(nn.Module):
    def __init__(self, num_heads, max_distance=128):
        super().__init__()
        self.num_heads = num_heads
        self.max_distance = max_distance
        self.relative_bias = nn.Embedding(2 * max_distance + 1, num_heads)

    def forward(self, seq_len):
        # distances matrix
        pos = torch.arange(seq_len)
        rel_dist = pos[None, :] - pos[:, None]  # [seq_len, seq_len]
        rel_dist = rel_dist.clamp(-self.max_distance, self.max_distance)
        rel_dist += self.max_distance  # shift to [0, 2*max_distance]
        # lookup bias
        bias = self.relative_bias(rel_dist)  # [seq_len, seq_len, num_heads]
        return bias.permute(2, 0, 1)  # [num_heads, seq_len, seq_len]
```

```python
# Example
bias_layer = RelativePositionalBias(num_heads=4, max_distance=10)
bias = bias_layer(seq_len=6)
print(bias.shape)  # [4, 6, 6]
```

---

3. Rotary Position Embeddings (RoPE)

- Popular in GPT-NeoX, LLaMA, ChatGLM.
- Instead of adding embeddings, it **rotates** query & key vectors in complex plane according to position.
- Supports extrapolation beyond training length.

Idea:
For each 2D pair (x_{2i}, x_{2i+1}), apply a rotation:

\text{RoPE}(x, pos) = \big[x_{2i}\cos(\theta_{pos}) - x_{2i+1}\sin(\theta_{pos}),\; x_{2i}\sin(\theta_{pos}) + x_{2i+1}\cos(\theta_{pos})\big]

Python code example (simplified):

```python
def rotary_embedding(x, seq_len, base=10000):
    """
    x: [batch, seq_len, d_model]
    """
    d_model = x.shape[-1]
    pos = torch.arange(seq_len, device=x.device).float()
    i = torch.arange(d_model//2, device=x.device).float()
    theta = pos[:, None] / (base ** (2 * i / d_model))  # [seq_len, d_model/2]

    cos = torch.cos(theta).repeat_interleave(2, dim=1)  # [seq_len, d_model]
    sin = torch.sin(theta).repeat_interleave(2, dim=1)
```

```
    cos = torch.cos(theta).repeat_interleave(2, dim=1)  # [seq_len, d_model]
    sin = torch.sin(theta).repeat_interleave(2, dim=1)

    x1 = x * cos + torch.cat([-x[..., 1::2], x[..., ::2]], dim=-1) * sin
    return x1

# Example
x = torch.randn(2, 5, 8)  # batch=2, seq_len=5, d_model=8
out = rotary_embedding(x, seq_len=5)
print(out.shape)  # [2, 5, 8]
```

✅ Summary Table

Method	Used in	Idea	Generalizes to longer seq?
Learned Absolute	BERT	Learn pos embedding table	❌ Limited to max length
Sinusoidal Absolute	Transformer (2017)	Fixed sin/cos function	✅ Yes
Relative Position Bias	T5, Transformer-XL, DeBERTa	Encode distance in attention	✅ Yes
Rotary Positional Embedding	GPT-NeoX, LLaMA	Rotate Q,K vectors by position	✅ Yes (good extrapolation)

👉 So, there are **at least 4 major categories**, with multiple variants inside.

\text{RoPE}(x, pos) = \big[x_{2i}\cos(\theta_{pos}) - x_{2i+1}\sin(\theta_{pos}),\; x_{2i}\sin(\theta_{pos}) + x_{2i+1}\cos(\theta_{pos})\big], 请用readable formulas to express it

Ask ChatGPT

好的 👍，你给的公式是 **Rotary Position Embedding (RoPE)** 的核心。它的本质是：
把每一对相邻的维度 (x_{2i}, x_{2i+1}) 当作平面坐标，然后根据位置 pos 旋转一个角度 $\theta_{pos,i}$。

---

1. 原始写法（你提供的）

$$\text{RoPE}(x, pos) = \Big[x_{2i}\cos(\theta_{pos,i}) - x_{2i+1}\sin(\theta_{pos,i}),\;\; x_{2i}\sin(\theta_{pos,i}) + x_{2i+1}\cos(\theta_{pos,i})\Big]$$

---

2. 更易读的矩阵形式

每个 2D 子向量 (x_{2i}, x_{2i+1}) 经过一个旋转矩阵：

$$\begin{bmatrix} x'_{2i} \\ x'_{2i+1} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{pos,i}) & -\sin(\theta_{pos,i}) \\ \sin(\theta_{pos,i}) & \cos(\theta_{pos,i}) \end{bmatrix} \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix}$$

---

3. 用复数形式表达（最简洁）

把 (x_{2i}, x_{2i+1}) 组合成一个复数：

$$z_i = x_{2i} + j \cdot x_{2i+1}, \quad j = \sqrt{-1}$$

那么旋转就是复数乘法：

3. 用复数形式表达（最简洁）

把 (x_{2i}, x_{2i+1}) 组合成一个复数：

$$z_i = x_{2i} + j \cdot x_{2i+1}, \quad j = \sqrt{-1}$$

那么旋转就是复数乘法：

$$z_i' = z_i \cdot e^{j\theta_{pos,i}}$$

最后再把实部、虚部分别拆开，得到 (x'_{2i}, x'_{2i+1})。

---

4. 角度 $\theta_{pos,i}$ 的定义

角度和位置、维度相关：

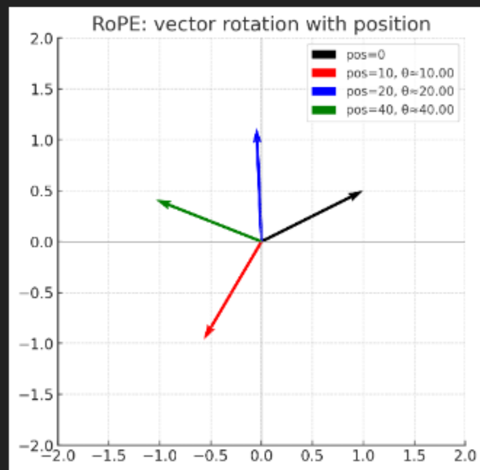
$$\theta_{pos,i} = \frac{pos}{10000^{2i/d}}$$

其中 d 是 embedding 维度。

✅ 总结

RoPE 其实就是：

- **把每对坐标 (x_{2i}, x_{2i+1}) 当作平面向量**
- **根据位置 pos 旋转一个角度 $\theta_{pos,i}$**
- 数学上等价于：
  - **旋转矩阵形式**（线性代数）
  - **复数乘法形式**（更简洁）



Here's the visualization 🎯:

Each arrow is the same base vector rotated by RoPE at different positions pos.

- **Black** = pos=0 (no rotation)
- **Red/Blue/Green** = larger positions → larger rotation angles θ.

This shows how RoPE encodes **position as a rotation in the embedding space**.

Want me to extend this to show **how multiple frequencies (different i) rotate at different speeds** — i.e., why RoPE can capture both short- and long-range dependencies? </>