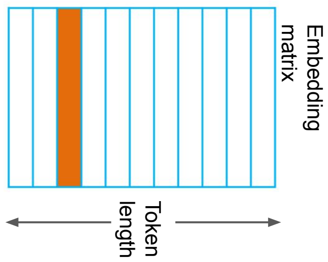
fun ai course

Evolution of Positional Encoding in the Transformer Module



Learned absolute

$$sin(pos/10000^{2i/d_{\text{model}}})$$
$$cos(pos/10000^{2i/d_{\text{model}}})$$

Sinusoidal

$$softmax(\frac{QK^T \textbf{+ rel\_K}}{\sqrt{d_k}})(V \textbf{ + rel\_V})$$

Learned relative

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

Rotary **(using complex number exponential format)**

hululu.zhu@gmail.com

06/2022

Agenda

- Quick self intro
- Intro to [vanilla] Transformer and Position Encoding

- Intuition and some math (not code deep-dive)
    - Learned position encoding (e.g. BERT)
    - Sinusoidal position encoding (e.g. Transformer)
    - Relative positional encoding (e.g. T5)
    - Rotary positional encoding (e.g. RoPE used in PaLM)

How are you today?

About me, 10+ years at Google…

My ML work experience starts at 2016 at Google Fiber

intern

SWE

Quick Intro to [vanilla] Transformer and Position Encoding

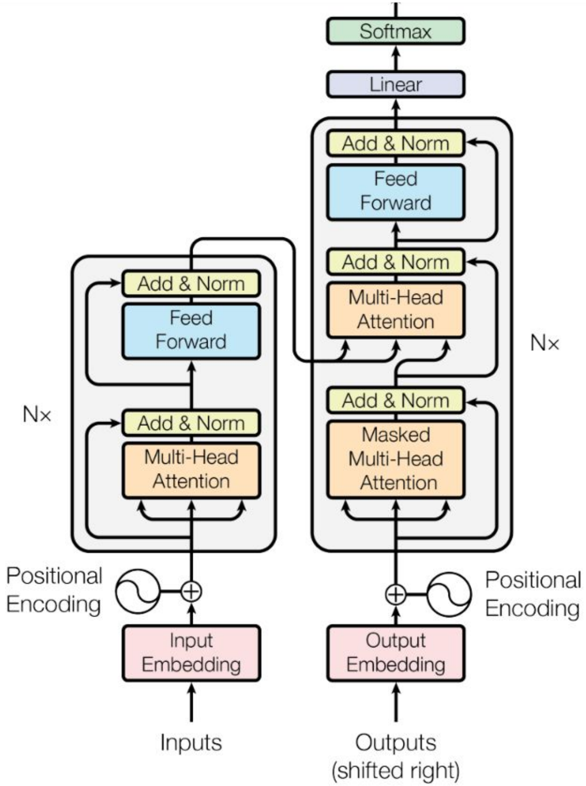


Figure 1: The Transformer - model architecture.

Transformer is the foundational building blocker

- NMT (neural machine translations)
- BERT
- GPT family
- T5
- LaMDA
- AlphaFold
- AlphaCode
- PaLM
- ViT (vision transformer)
- …

Quick Intro to [vanilla] Transformer and Position Encoding

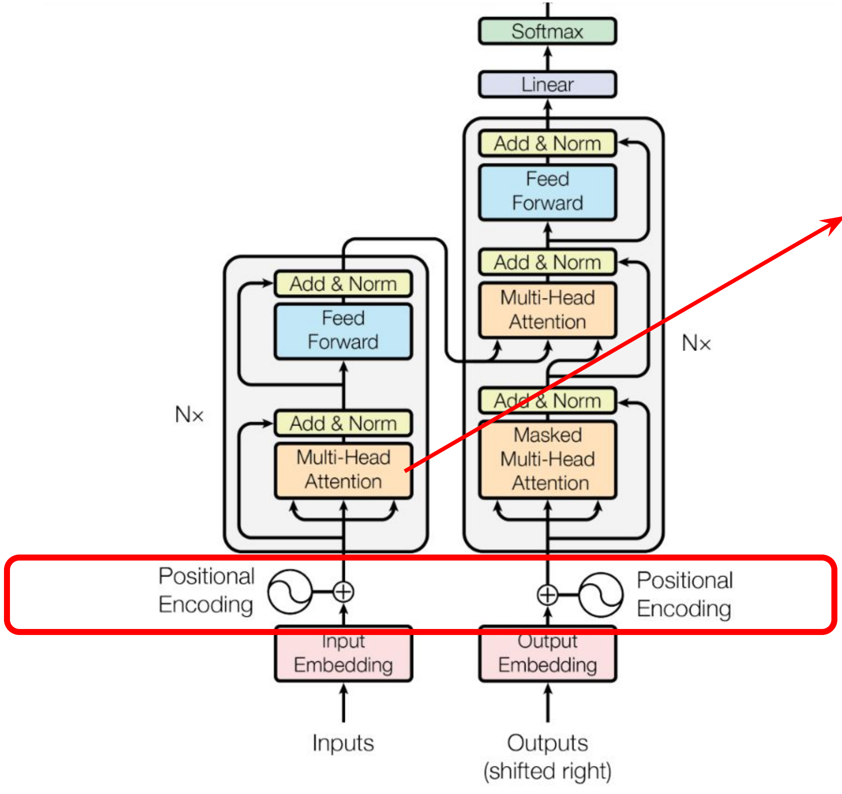
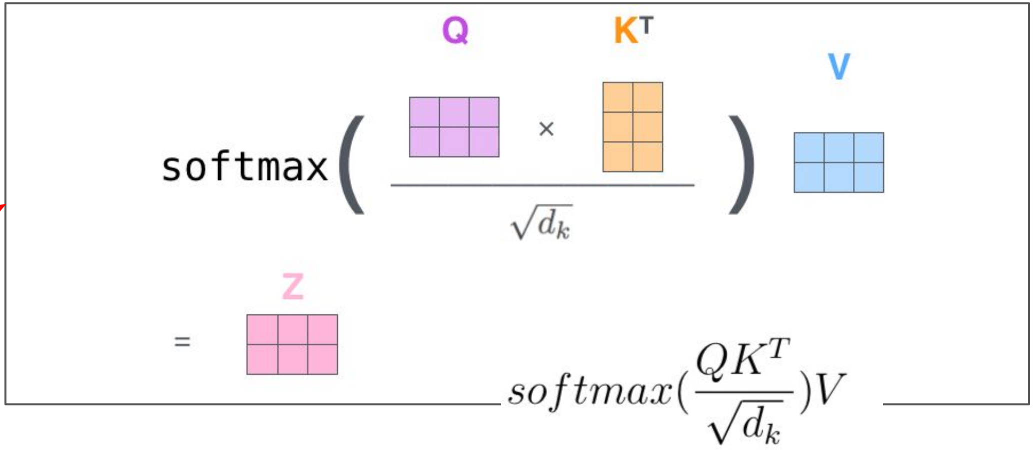


Figure 1: The Transformer - model architecture.



- Self attention implementation through multi-head attention (the famous so called “softmax(qkv)” implementation)
- **Positional encoding is a critical piece!**

Quick Intro to [vanilla] Transformer and Position Encoding

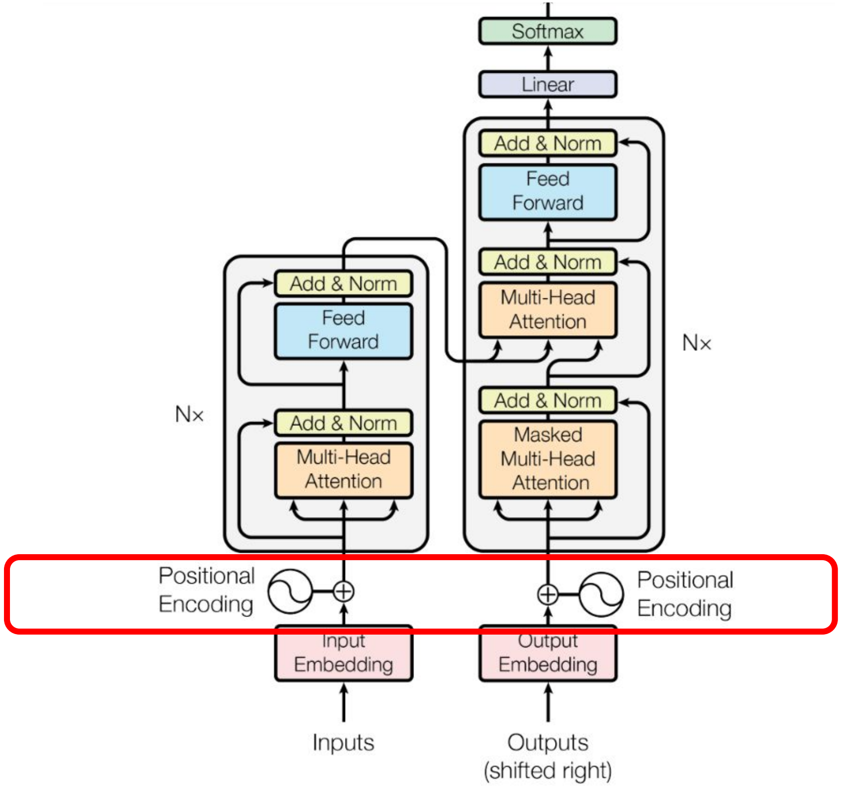
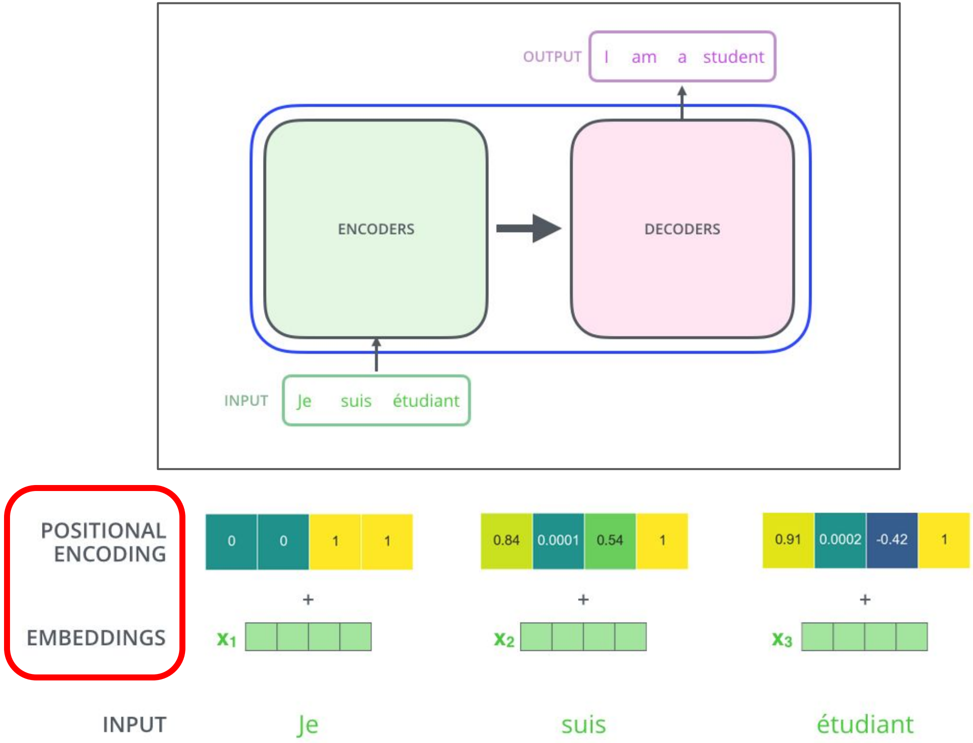


Figure 1: The Transformer - model architecture.



Ref: The Illustrated Transformer – Jay Alammar

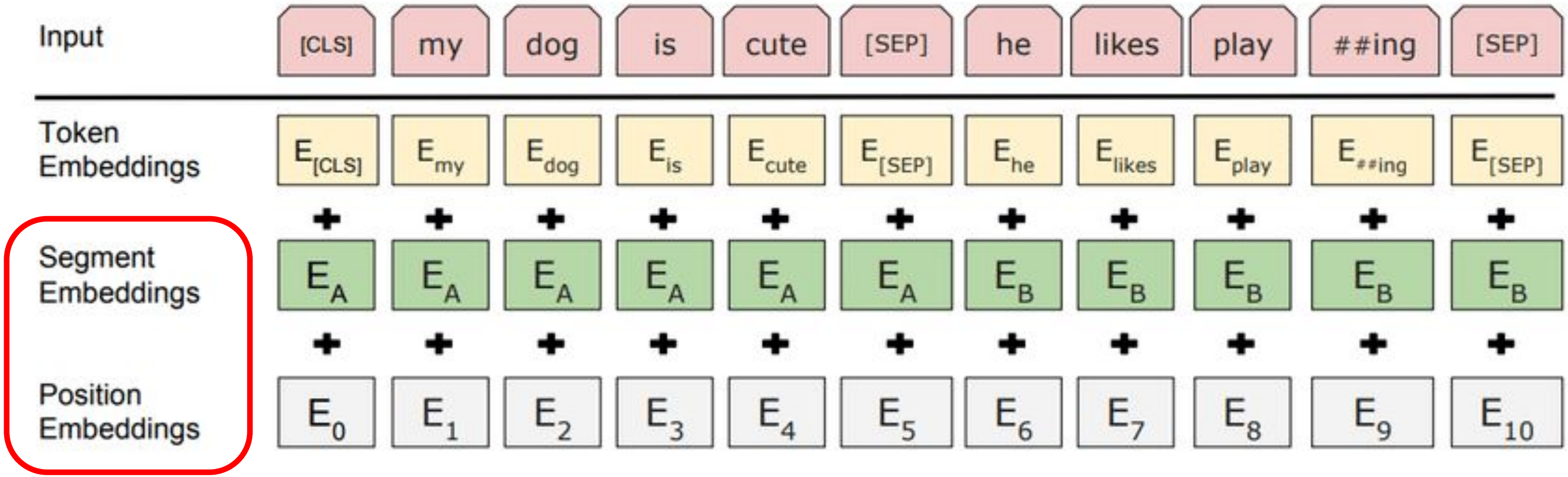
An ideal positional encoding will

1. Make each token's position encoding to respect
    a. The order
        i. e.g. "***Warriors beat Celtics***" vs "***Celtics beat Warriors***" , the same "**beat**" should ideally pay different amount of attention to the same "**Warriors**" and "**Celtics**" tokens (through multi-head attention matrix calculation)
    b. The relative distance
        i. E.g. "Awesome! Warriors beat Celtics" vs "I really **don't like** Warriors beat Celtics", "**beat**" has different absolution positions this time, but it should ideally pay same amount of attention to "**Warriors**" and "**Celtics**" because relative distances are the same!
2. Be easy to be fused with token embedding
    a. Thus it is transformation from scalar_pos_id -> position_embedding_vector (same size as token embedding)
    b. Ideally to have zero mean and reasonable std than token (e.g. word piece) embedding
    c. Please note the signal fusion of token embedding and positional embedding will discussed further

Simplest, learned positional embedding, e.g. BERT

Learned positional (and segment) embeddings **added** to token_embedding

- Train embedding_lookup from abs_position/segment id to size_bert embedding
- Summation(token_embed, seg_embed, abs_pos_embed) before attention module



Sinusoidal Positional Encoding

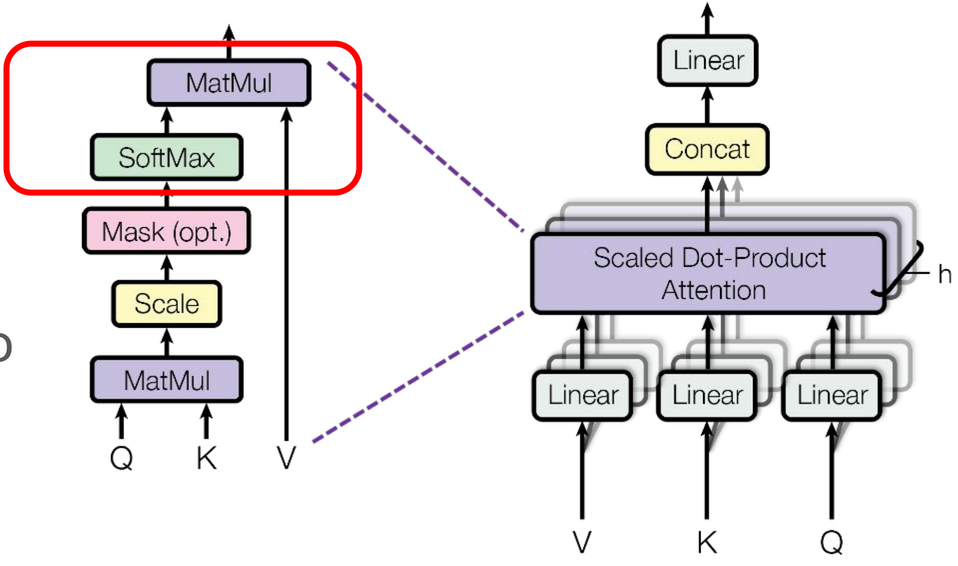
Some characteristics

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

1. Between -1 to 1
2. Centered at 0
3. Why combing Sin/Cos?
   a. sin(a+b)=sina*cosb+cosa*sinb
   b. cos(a+b)=cosa*cosb−sina*sinb
   c. IMO, implicitly considers the relative position distance (because of nature of sin/cos addition)
4. Why 10000^pos?
   a. 10000 belongs to the so called “hyper parameter”
5. As compared to learned embedding?
   a. Pre-calculated instead of learning, thus more efficient
   b. Empirically as good as learning embedding in translation (BERT has other positional embeddings e.g. segment, thus learning embedding makes sense)

Think about the Relative Position Representations

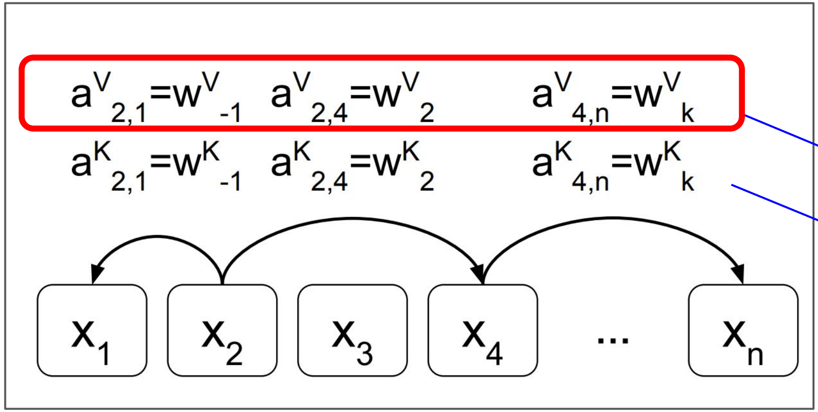
- We want to keep QKV as “pure” token embeddings
- Before Softmax (attention scores to be multiplied with V), we like to learn relative embeddings for relative positions!
    - QK^T and V are both promising! Try both!

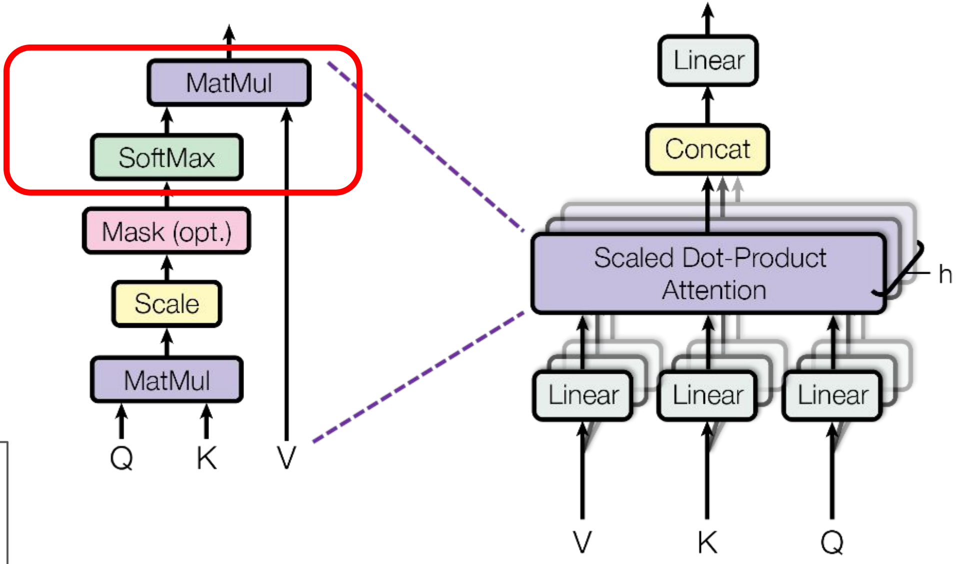


$$softmax(\frac{QK^T}{\sqrt{d_k}})V \longrightarrow softmax(\frac{\boxed{QK^T}}{\sqrt{d_k}})\boxed{V}$$

Learn Relative Position Representations

- Relative distance is important!
  - Assume same relative distance embedding are same regardless of absolute position!
- Clipping relevance window!
  - Do not pay attention to tokens that are too far apart!
- Learn 2 embedding sets (V & K)



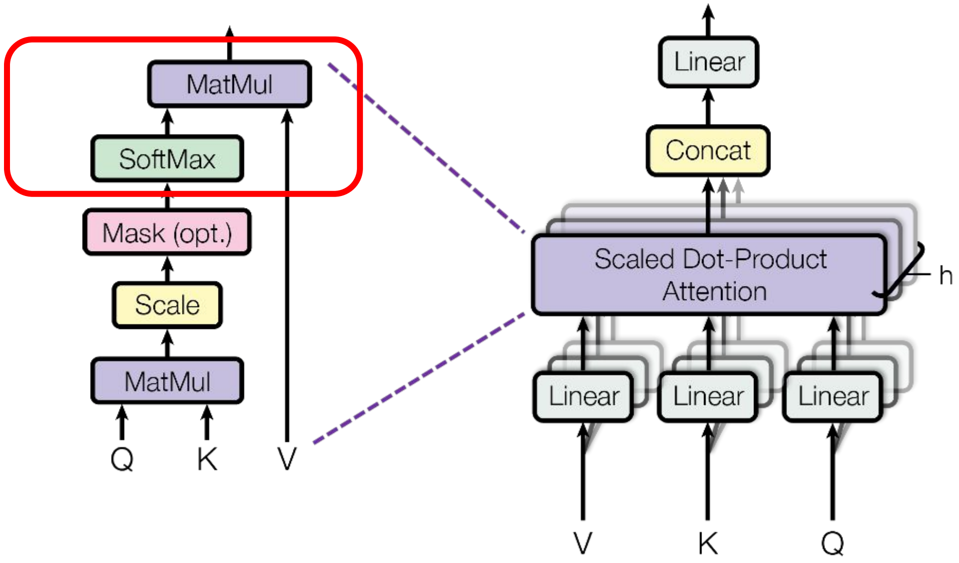


Learn 2 groups of embedding!

- **relative_pos_embed_V**
- **relative_pos_embed_K**

Apply the learned Relative Position Representations

- Keep QKV as "pure" token embeddings
- Learned K/V relative positional embeddings applied to pre-softmax and post-softmax



$$softmax\left(\frac{QK^T + \textbf{relative\_pos\_embed\_K}}{\sqrt{d_k}}\right)\left(V + \textbf{relative\_pos\_embed\_V}\right)$$

Rotary Position Embedding (RoPE)

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n)$$

Rethink the core problem

- In other words, we hope the inner product of embeddings (f function with x embedding and position m/n) is **only related** to relative difference **m-n**
- And we only need one possible solution
    - Answer: Use complex number multiplication
    - In complex number exponential format
        - Multiplication is basically angle addition
        - Complex number inner product uses its **conjugate format (x + iy -> x - iy)**! Thus it the **negation(add_angle) = substract_angle**!!

Walk through Rotary Position Embedding (RoPE) formulas

Try to solve using complex number

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n)$$

R is the magnitude, Θ is the complex space angle, so we hope

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{q}, m) &= R_f(\boldsymbol{q}, m)e^{\mathrm{i}\Theta_f(\boldsymbol{q},m)} \\ \boldsymbol{f}(\boldsymbol{k}, n) &= R_f(\boldsymbol{k}, n)e^{\mathrm{i}\Theta_f(\boldsymbol{k},n)} \\ \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{k}, m-n) &= R_g(\boldsymbol{q}, \boldsymbol{k}, m-n)e^{\mathrm{i}\Theta_g(\boldsymbol{q},\boldsymbol{k},m-n)} \end{aligned}$$

Walk through Rotary Position Embedding (RoPE) formulas

Try to solve using complex number

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n)$$

R is the magnitude, Θ is the complex space angle, so we hope

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{q}, m) &= R_f(\boldsymbol{q}, m) e^{\mathrm{i}\Theta_f(\boldsymbol{q}, m)} \\ \boldsymbol{f}(\boldsymbol{k}, n) &= R_f(\boldsymbol{k}, n) e^{\mathrm{i}\Theta_f(\boldsymbol{k}, n)} \\ \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{k}, m-n) &= R_g(\boldsymbol{q}, \boldsymbol{k}, m-n) e^{\mathrm{i}\Theta_g(\boldsymbol{q}, \boldsymbol{k}, m-n)} \end{aligned}$$

$$\begin{aligned} R_f(\boldsymbol{q}, m) R_f(\boldsymbol{k}, n) &= R_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \\ \Theta_f(\boldsymbol{q}, m) - \Theta_f(\boldsymbol{k}, n) &= \Theta_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \end{aligned}$$

Walk through Rotary Position Embedding (RoPE) formulas

Try to solve using complex number

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n)$$

R is the magnitude, Θ is the complex space angle, so we hope

$$\boxed{\begin{aligned} R_f(\boldsymbol{q}, m)R_f(\boldsymbol{k}, n) &= R_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \\ \Theta_f(\boldsymbol{q}, m) - \Theta_f(\boldsymbol{k}, n) &= \Theta_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \end{aligned}}$$

If we have m=n, then

$$\boxed{R_f(\boldsymbol{q}, m)R_f(\boldsymbol{k}, m) = R_g(\boldsymbol{q}, \boldsymbol{k}, 0) = R_f(\boldsymbol{q}, 0)R_f(\boldsymbol{k}, 0) = \|\boldsymbol{q}\|\,\|\boldsymbol{k}\|}$$

Walk through Rotary Position Embedding (RoPE) formulas

Try to solve using complex number

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n)$$

R is the magnitude, Θ is the complex space angle, so we hope

$$\boxed{\begin{aligned} R_f(\boldsymbol{q}, m)R_f(\boldsymbol{k}, n) &= R_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \\ \Theta_f(\boldsymbol{q}, m) - \Theta_f(\boldsymbol{k}, n) &= \Theta_g(\boldsymbol{q}, \boldsymbol{k}, m-n) \end{aligned}}$$

If we have m=n, then

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

A simplified version to summarize Rotary Position Embedding (RoPE)

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

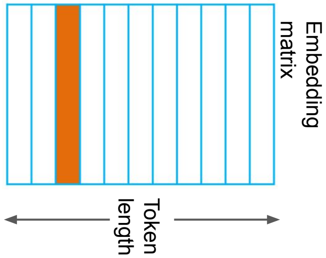
Fusion of token embedding and RoPE is through complex number **multiplication**!

- As compared to **addition** in learning of sinusoida embeddings

References

- [1706.03762] Attention Is All You Need
- [1803.02155] Self-Attention with Relative Position Representations
- [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- [2104.09864] RoFormer: Enhanced Transformer with Rotary Position Embedding
- Transformer升级之路:2、博采众长的旋转式位置编码- 科学空间
- Rotary Embeddings: A Relative Revolution | EleutherAI Blog
- Master Positional Encoding: Part I | by Jonathan Kernes | Towards Data Science
- The Illustrated Transformer – Jay Alammar

Thank you!
Evolution of Positional Encoding in the Transformer Module



Learned absolute

$$sin(pos/10000^{2i/d_{\text{model}}})$$
$$cos(pos/10000^{2i/d_{\text{model}}})$$

Sinusoidal

$$softmax(\frac{QK^T + rel\_K}{\sqrt{d_k}})(V + rel\_V)$$

Learned relative

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

Rotary **(using complex number exponential format)**

hululu.zhu@gmail.com

06/2022

The core problem

1. The position of tokens in a sequence matters!
    a. “Warriors beat Celtics” vs “Celtics beat Warriors” are different!
    b. “Warriors beat Celtics” is anagram to “Celtics beat Warriors”, so from word-embedding perspective, they are the same!
    c. So we need extra positional info for “Warriors” vs “Celtics”
2. The absolution position is less important than relative position
    a. “You know **Warriors** beat **Celtics**?” vs “**Warriors** beat **Celtics**, that is great!”
    b. Absolute positions for “Warriors” vs “Celtics” are different, but their relative position is the same!
    c. Another example: “I kind of like the team of Warrior” vs “I like warrior”
3. Position to embedding is a scalar (single value) to vector (size matches transformer embedding size) mapping

Vanilla Transformer Embedding

1. Number?
2. Binary encoding?
3. Periodical pattern to [hopefully] center at zero?

Relative pos encoding

Rotary positional encoding

Let’s take a look at the code together

Summary, tldr, use RoPE for large models