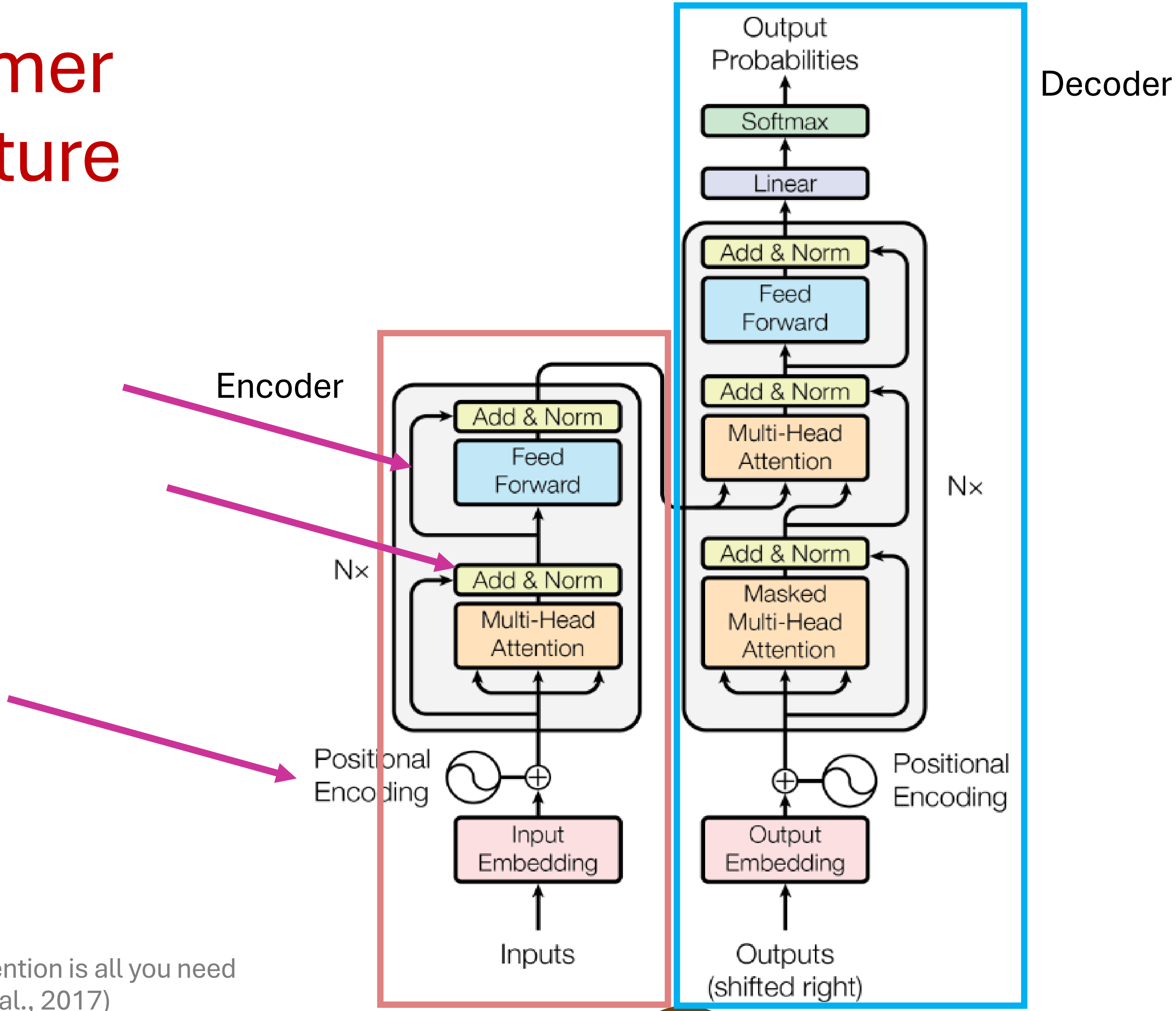
Introduction to Transformer

Part-2

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

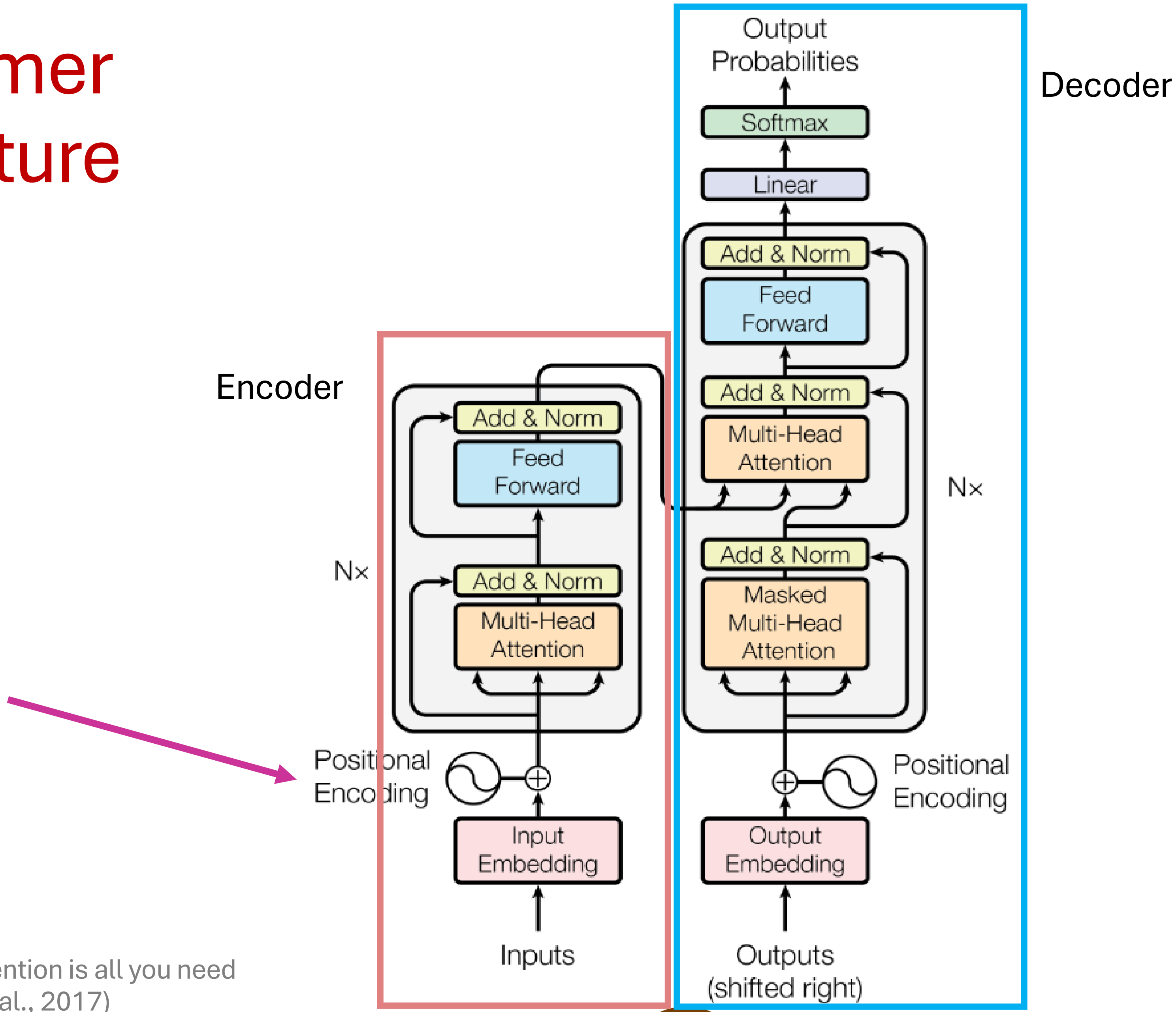
Introduction to Large Language Models

Transformer Architecture



Source of Image : Attention is all you need (Vaswani t al., 2017)

Transformer Architecture



Source of Image : Attention is all you need (Vaswani t al., 2017)

Positional Encoding

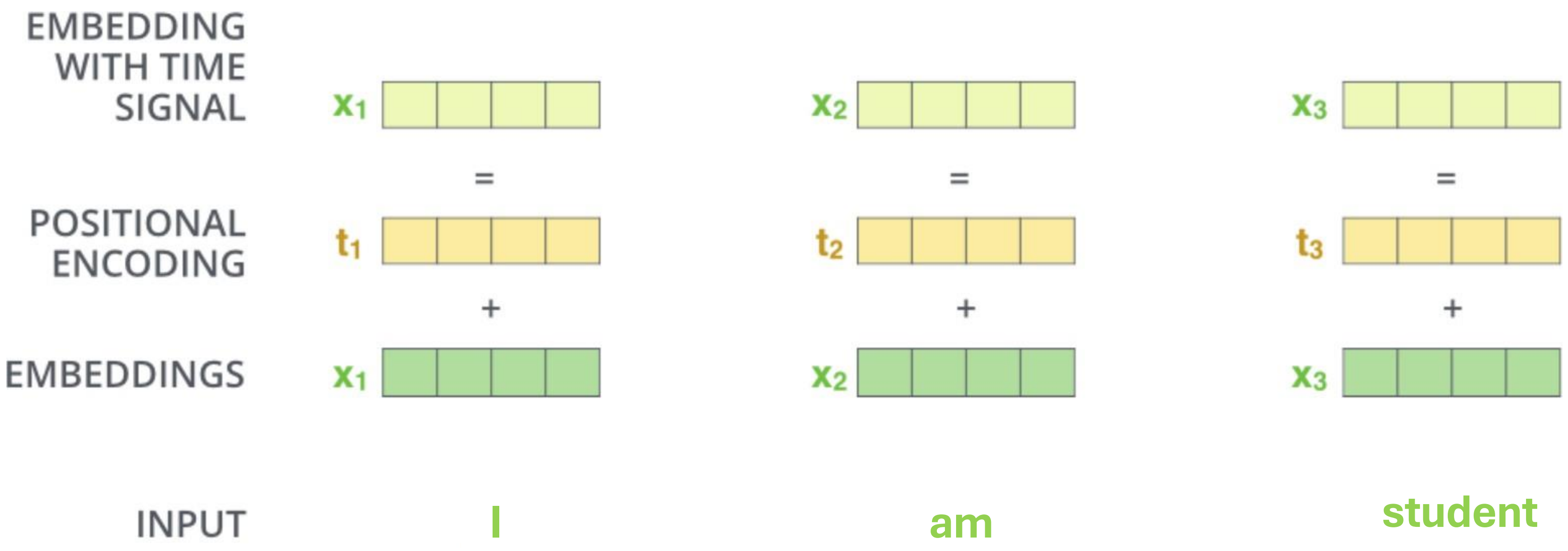
Position Information in Transformers: An Overview

Philipp Dufter, Martin Schmitt, Hinrich Schütze

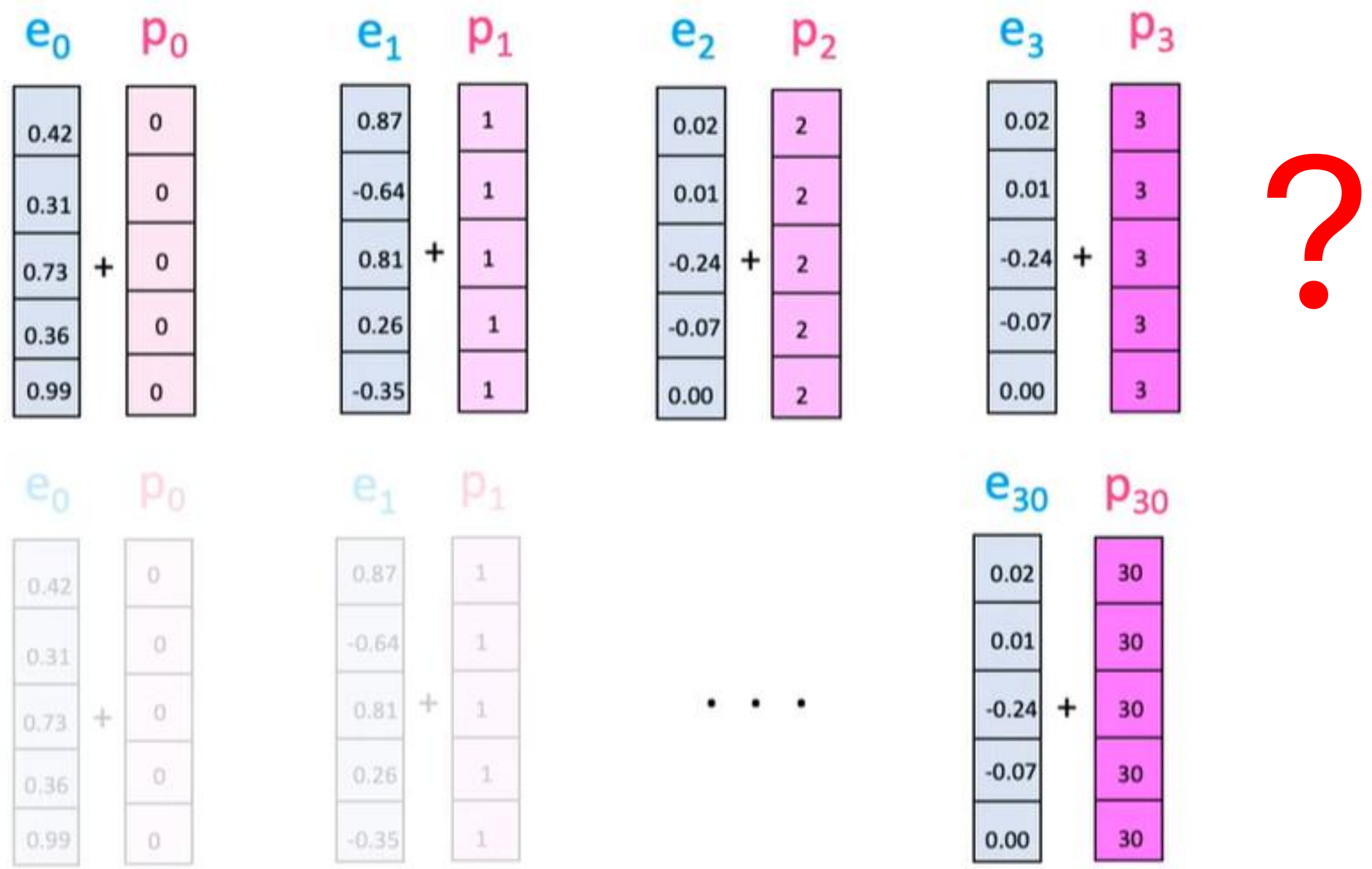
Abstract

Transformers are arguably the main workhorse in recent natural language processing research. By definition, a Transformer is invariant with respect to reordering of the input. However, language is inherently sequential and word order is essential to the semantics and syntax of an utterance. In this article, we provide an overview and theoretical comparison of existing methods to incorporate position information into Transformer models. The objectives of this survey are to (1) showcase that position information in Transformer is a vibrant and extensive research area; (2) enable the reader to compare existing methods by providing a unified notation and systematization of different approaches along important model dimensions; (3) indicate what characteristics of an application should be taken into account when selecting a position encoding; and (4) provide stimuli for future research.

Positional Encoding

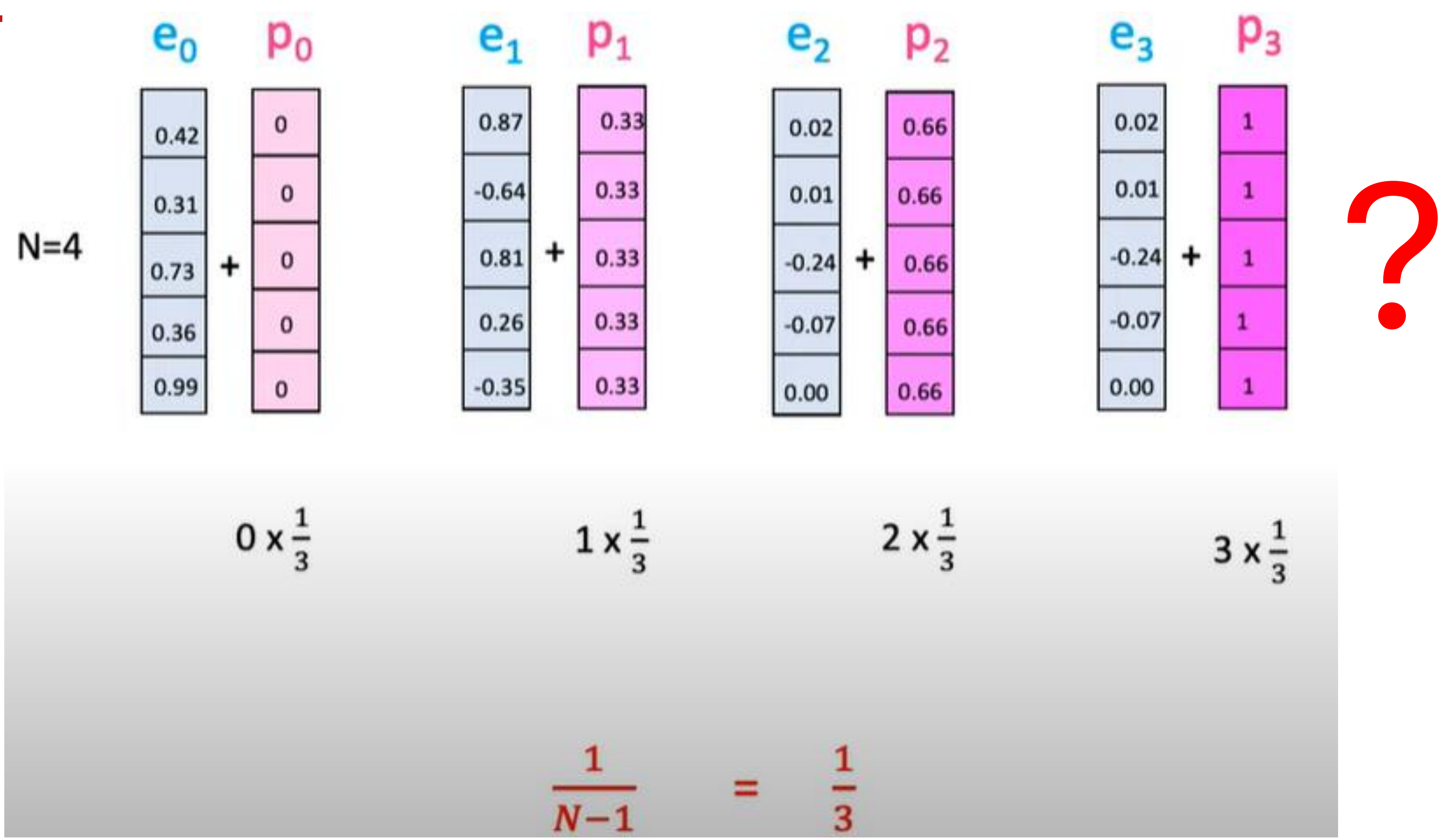


Option 1



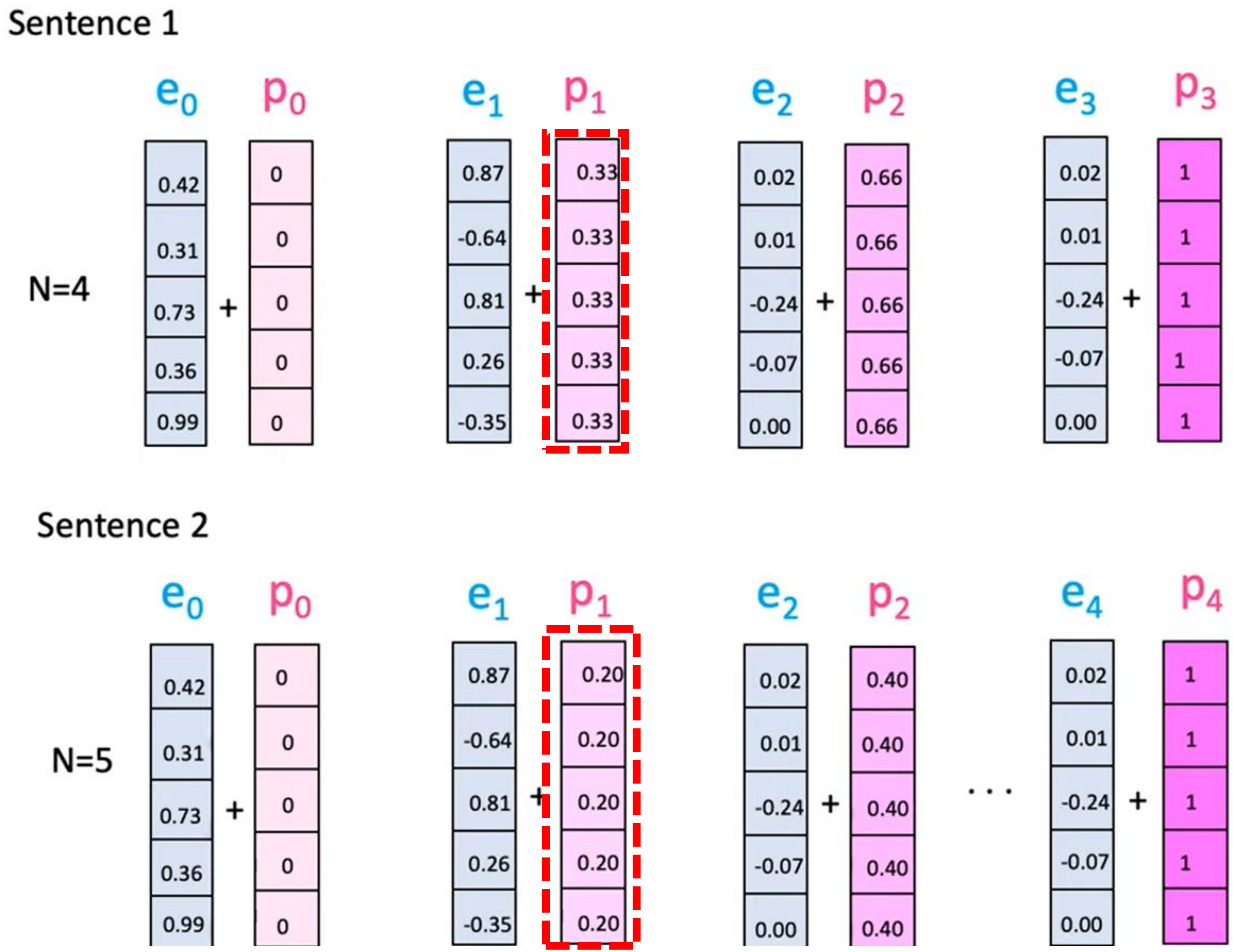
Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Option 2



Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Option 2



The positional embedding vector at a given position should remain the same irrespective of the length of the sequence

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Creating Positional Encodings

- We could just concatenate a fixed value to each time step (e.g., 1, 2, 3, … 1000) that corresponds to its position, but then what happens if we get a sequence with 5000 words at test time?
- We want something that can generalize to arbitrary sequence lengths. We also may want to make attending to *relative positions* (e.g., tokens in a local window to the current token) easier.
- Distance between two positions should be consistent with variable-length inputs

Intuitive Example

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

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

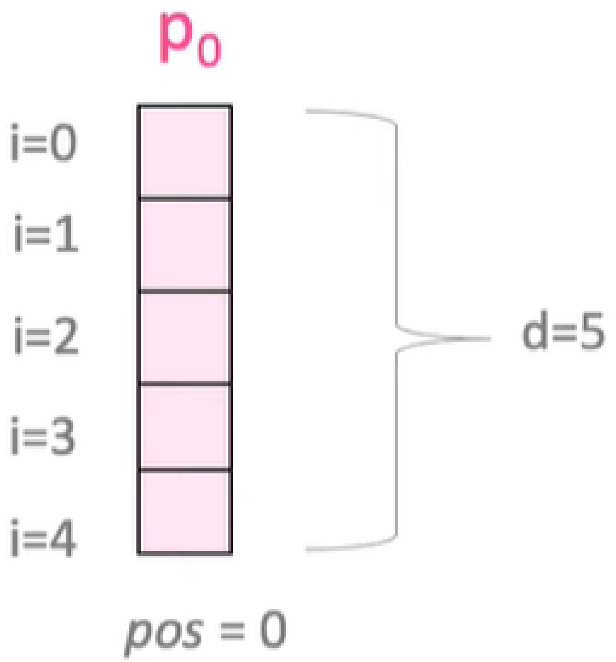
Transformer Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

For $\mathbf{d_{model}}$ **= 512,**
Positional encoding is a 512-dimensional vector
(Note: **Dimension of positional encoding is same as dimension of the word embeddings**)
i = a particular dimension of this vector
pos = position of the word in the sequence

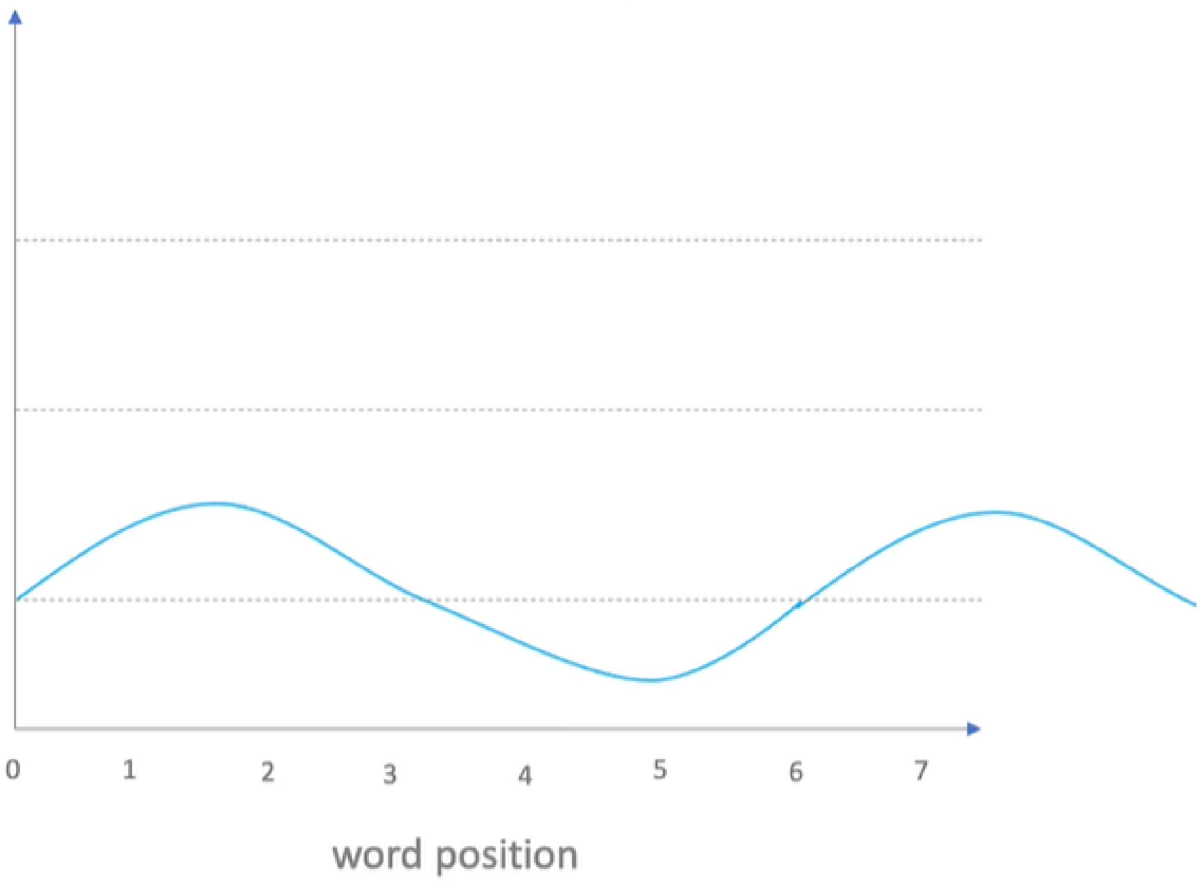
Transformer Positional Encoding



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

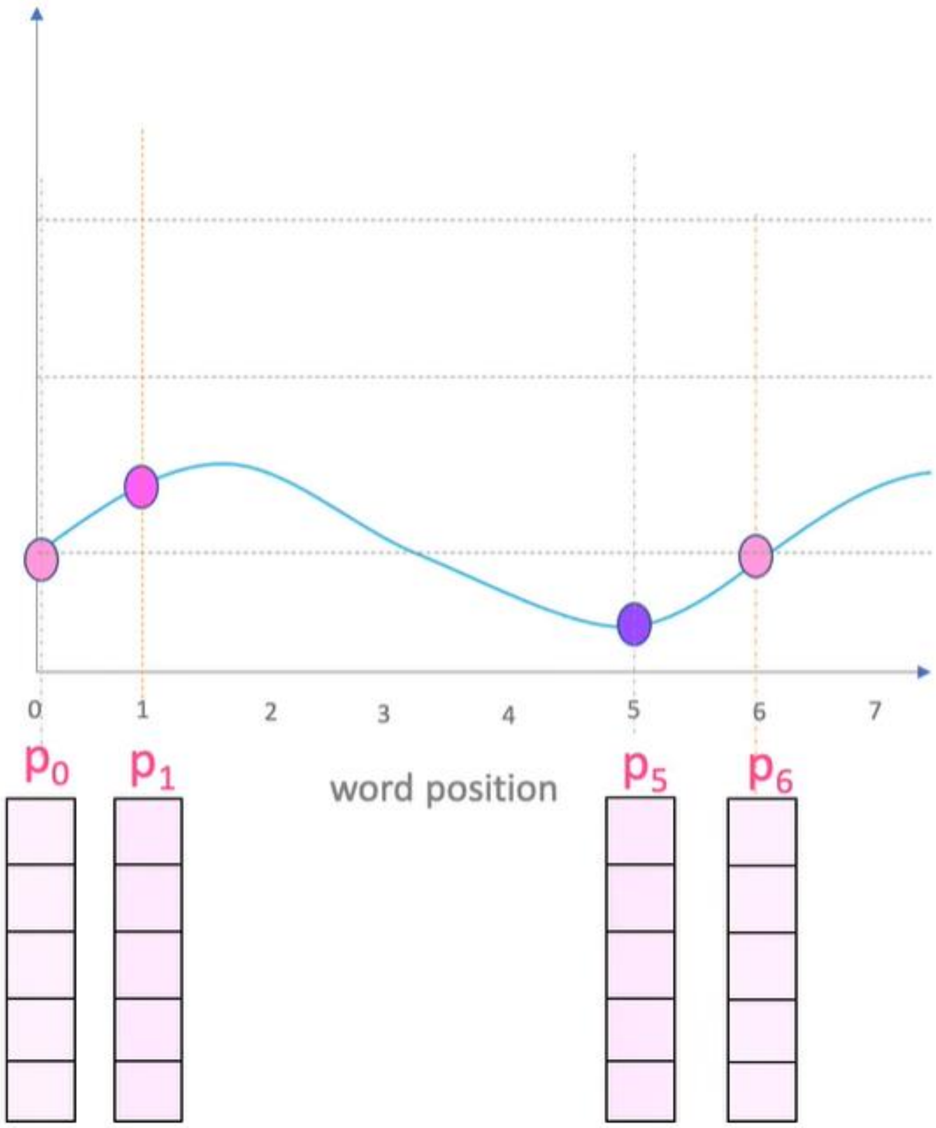
Only Varying The Position



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Only Varying The Position



Pros

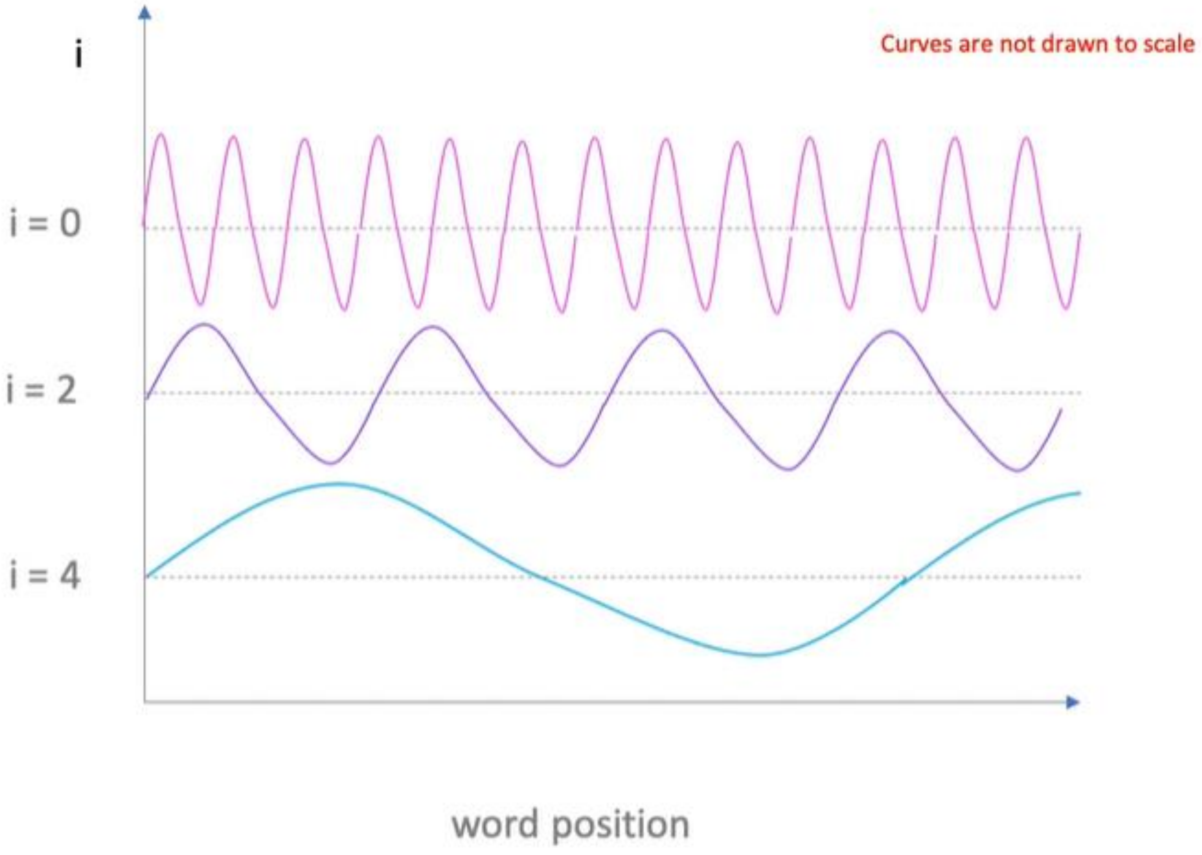
- It is independent of the length of the sequence

Cons

- p_0 and p_6 have same positional embeddings

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Varying Both Position and *i*

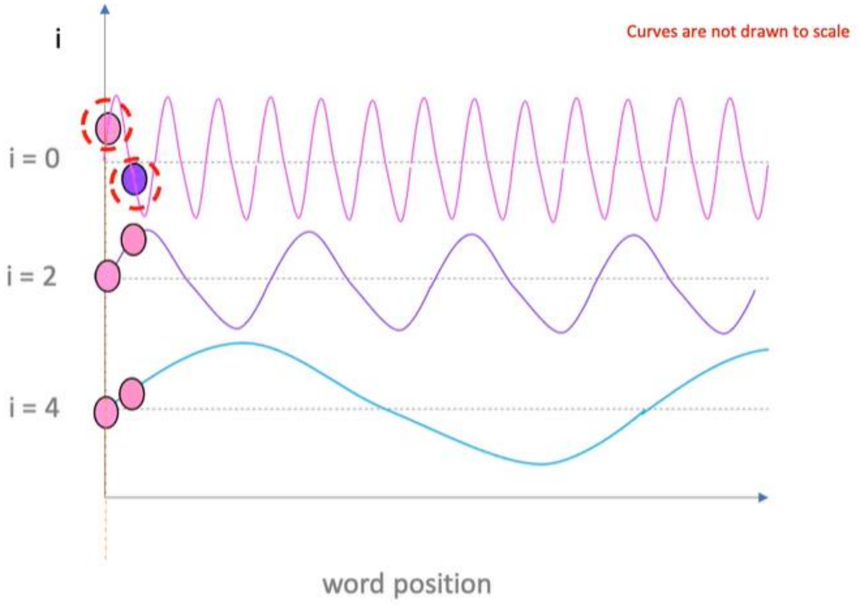


$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

i

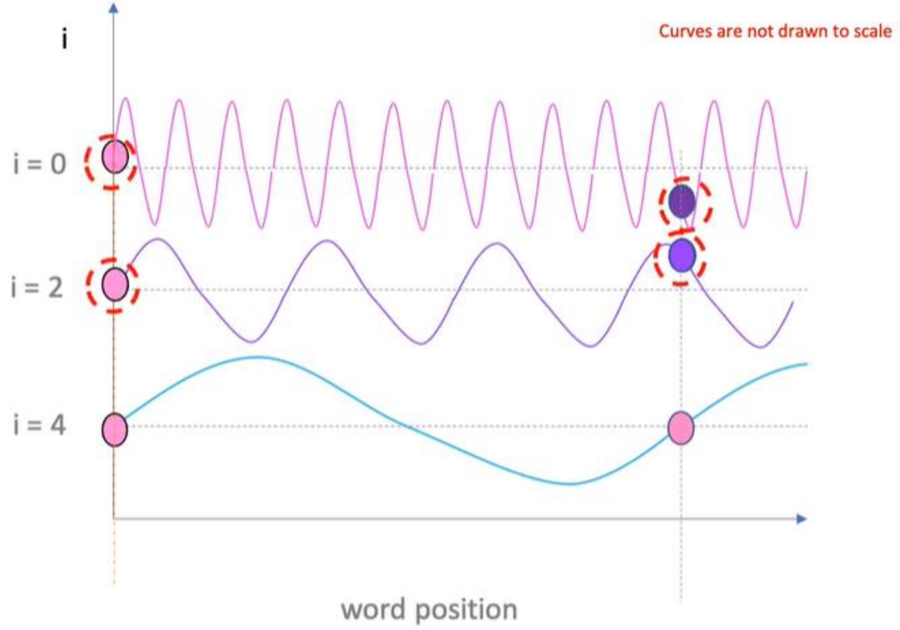
Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Varying Both Position and *i*



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$ i

If two points are close by on the curve, they will remain identical at low frequencies too. It is at the high frequency where their y-axis values differ and we may be able to take them apart.

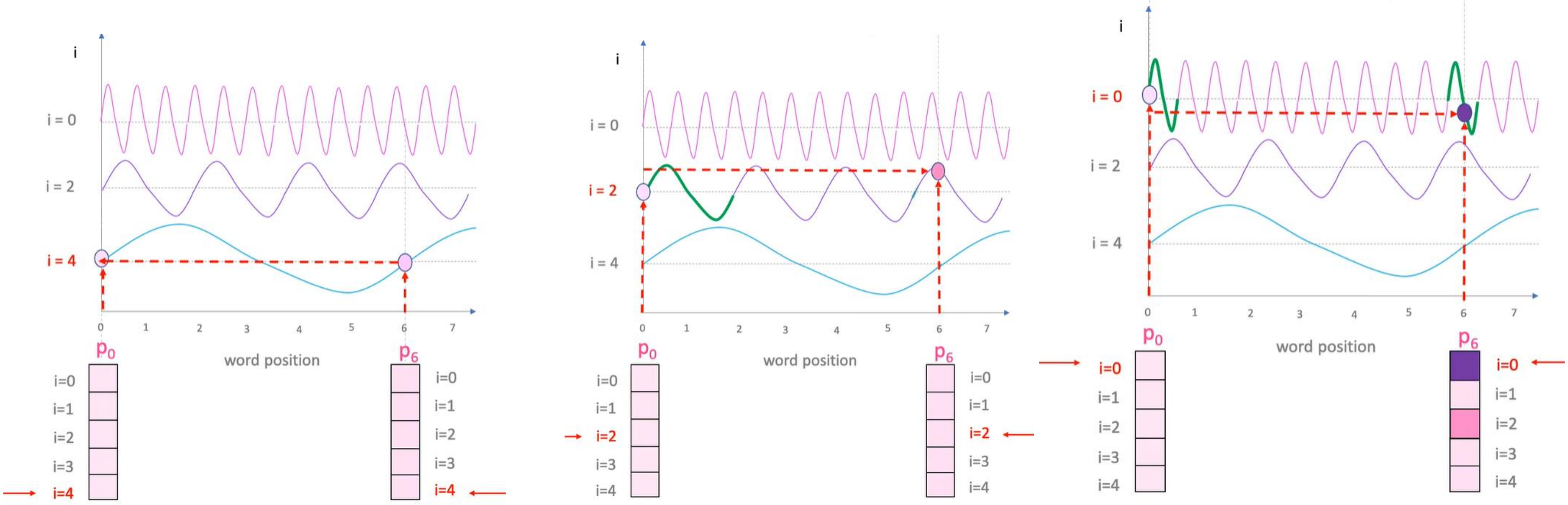


$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$ i

For points which are far apart, we will see them falling apart on the y-axis quite early on

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Varying Both Position and *i: Example*



At lower frequency, the value is exactly the same

This will start to differ significantly as we move to high frequencies

Credits: https://www.youtube.com/watch?v=dichIcUZfOw

Example

For example, for word w at position $pos \in [0, L-1]$ in the input sequence $\boldsymbol{w} = (w_0, \cdots, w_{L-1})$, with 4-dimensional embedding e_w, and $d_{model} = 4$, the operation would be

$$\begin{aligned} e'_w &= e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right) \right] \\ &= e_w + \left[sin\left(pos\right), cos\left(pos\right), sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right) \right] \end{aligned}$$

where the formula for positional encoding is as follows

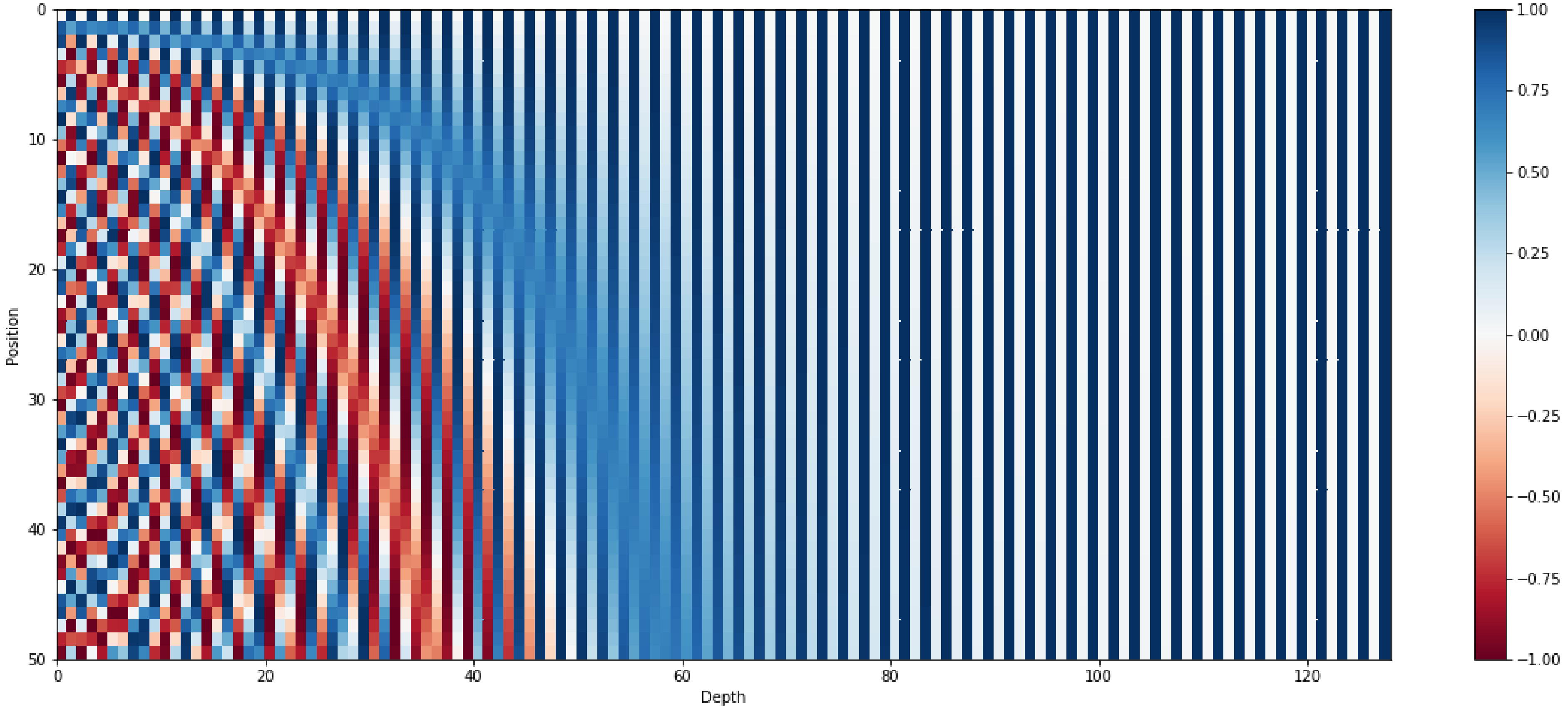
$$\mathrm{PE}(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right),$$

$$\mathrm{PE}(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).$$

https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model

What does this look like?

(each row is the positional encoding of a 50-word sentence)



https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Despite the intuitive flaws, many models these days use ***learned positional embeddings*** (i.e., they cannot generalize to longer sequences, but this isn't a big deal for their use cases)

General Properties of Positional Embeddings

We define the function ϕ(·, ·) to measure the proximity between positional embeddings.

- **Monotonicity:** The proximity of position embeddings positions decreases when positions are further apart.

$$\forall x, m, n \in \mathbb{N} : m > n \iff \phi(\vec{x}, \overrightarrow{x+m}) < \phi(\vec{x}, \overrightarrow{x+n})$$

- **Translation inv** … variant.

$$\forall x_1, \ldots, x_n, m \in \mathbb{N} : \phi(\vec{x}_1, \overrightarrow{x_1+m}) = \phi(\vec{x}_2, \overrightarrow{x_2+m}) = \cdots = \phi(\vec{x}_n, \overrightarrow{x_n+m})$$

- **S**

$$\forall x, y \in \mathbb{N} : \phi(\vec{x}, \vec{y}) = \phi(\vec{y}, \vec{x})$$

Paper: On Position Embeddings in BERT (Wang et al., ICLR 2021)

Rotary Positional Encoding (RoPE)

RoFormer: Enhanced Transformer with Rotary Position Embedding

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

November 9, 2023

Adopted by
- PaLM
- GPT–Neo and GPT-J
- LlaMa 1 and 2

Credit: Slides of RoPE is adopted from Manish Gupta

Absolute Positional Encoding

- Learned from data
- Sinusoidal function (like the original Transformer)

Absolute Position Embeddings

$$\mathbb{E}_N = \{\boldsymbol{x}_i\}_{i=1}^N$$

$$\begin{aligned} \boldsymbol{q}_m &= f_q(\boldsymbol{x}_m, m) \\ \boldsymbol{k}_n &= f_k(\boldsymbol{x}_n, n) \\ \boldsymbol{v}_n &= f_v(\boldsymbol{x}_n, n), \end{aligned}$$

$$a_{m,n} = \frac{\exp(\frac{\boldsymbol{q}_m^\intercal \boldsymbol{k}_n}{\sqrt{d}})}{\sum_{j=1}^N \exp(\frac{\boldsymbol{q}_m^\intercal \boldsymbol{k}_j}{\sqrt{d}})}$$

$$\mathbf{o}_m = \sum_{n=1}^N a_{m,n} \boldsymbol{v}_n$$

- w_i are tokens and x_i are embeddings

$$f_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i, i) := \boldsymbol{W}_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i + \boldsymbol{p}_i)$$

$$\begin{cases} \boldsymbol{p}_{i,2t} &= \sin(k/10000^{2t/d}) \\ \boldsymbol{p}_{i,2t+1} &= \cos(k/10000^{2t/d}) \end{cases}$$

- Generate p_i using the sinusoidal function
  - Sinusoidal functions provide continuity between close positions.
  - This also allows for the model to scale to virtually unlimited input sequence length

Relative Positional Encoding

The dog chased the pig

Once upon a time, the pig chased the dog

Relative Position Embeddings

$$\begin{aligned} f_q(\boldsymbol{x}_m) &:= \boldsymbol{W}_q \boldsymbol{x}_m \\ f_k(\boldsymbol{x}_n, n) &:= \boldsymbol{W}_k(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^k) \\ f_v(\boldsymbol{x}_n, n) &:= \boldsymbol{W}_v(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^v) \end{aligned}$$

- $\tilde{p}_r^k, \tilde{p}_r^v \in R^d$ are trainable relative position embeddings.
- r=clip(m−n, r_{min}, r_{max}) is relative distance between positions m and n.
  - Relative position info is not useful beyond a certain distance.

Relative Position Embeddings

Transformer-XL

$$f_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i, i) := \boldsymbol{W}_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i + \boldsymbol{p}_i)$$

- Decompose $q_m^T k_n$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n$$

- Replace absolute position embedding p_n with relative $\tilde{p}_{m-n}$
- Replace absolute position embedding p_m with two trainable vectors u and v independent of query positions.
- W_k is distinguished for content-based and location-based key vectors x_n and p_n, denoted as W_k and $\widetilde{W}_k$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \widetilde{\boldsymbol{W}}_k \tilde{\boldsymbol{p}}_{m-n} + \mathbf{u}^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \mathbf{v}^\intercal \boldsymbol{W}_q^\intercal \widetilde{\boldsymbol{W}}_k \tilde{\boldsymbol{p}}_{m-n}$$

Relative Position Embeddings

$$f_q(\boldsymbol{x}_m) := \boldsymbol{W}_q \boldsymbol{x}_m$$
$$f_k(\boldsymbol{x}_n, n) := \boldsymbol{W}_k(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^k)$$
$$f_v(\boldsymbol{x}_n, n) := \boldsymbol{W}_v(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^v)$$

- $\tilde{p}_r^k, \tilde{p}_r^v \in R^d$ are trainable relative position embeddings.
- r=clip(m−n, r_{min}, r_{max}) is relative distance between positions m and n.
    - Relative position info is not useful beyond a certain distance.
- **Transformer-XL**
    - Decompose $q_m^T k_n$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n$$

- Replace abs pos embedding p_n with relative $\tilde{p}_{m-n}$
- Replace abs pos embedding p_m with two trainable vectors u and v independent of query positions.
- W_k is distinguished for content-based and location-based key vectors x_n and p_n, denoted as W_k and $\widetilde{W}_k$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \widetilde{\boldsymbol{W}}_k \tilde{\boldsymbol{p}}_{m-n} + \mathbf{u}^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \mathbf{v}^\intercal \boldsymbol{W}_q^\intercal \widetilde{\boldsymbol{W}}_k \tilde{\boldsymbol{p}}_{m-n}$$

- T5 uses a very simplified relative position embedding

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + b_{i,j}$$

    - $b_{i,j}$ is a trainable bias.
- Another formulation

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \mathbf{U}_q^\intercal \mathbf{U}_k \boldsymbol{p}_n + b_{i,j}$$

Relative Position Embeddings

$$f_q(\boldsymbol{x}_m) := \boldsymbol{W}_q \boldsymbol{x}_m$$
$$f_k(\boldsymbol{x}_n, n) := \boldsymbol{W}_k(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^k)$$
$$f_v(\boldsymbol{x}_n, n) := \boldsymbol{W}_v(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^v)$$

- $\tilde{p}_r^k, \tilde{p}_r^v \in R^d$ are trainable relative position embeddings.
- r=clip(m−n, r_{min}, r_{max}) is relative distance between positions m and n.
  - Relative position info is not useful beyond a certain distance.
- **Transformer-XL**
  - Decompose $q_m^T k_n$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n$$

  - Replace abs pos embedding p_n with relative $\tilde{p}_{m-n}$
  - Replace abs pos embedding p_m with two trainable vectors u and v independent of query positions.
  - W_k is distinguished for content-based and location-based key vectors x_n and p_n, denoted as W_k and $\widetilde{W}_k$

- T5 uses a very simplified relative position embedding

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + b_{i,j}$$

  - $b_{i,j}$ is a trainable bias.
- Another formulation

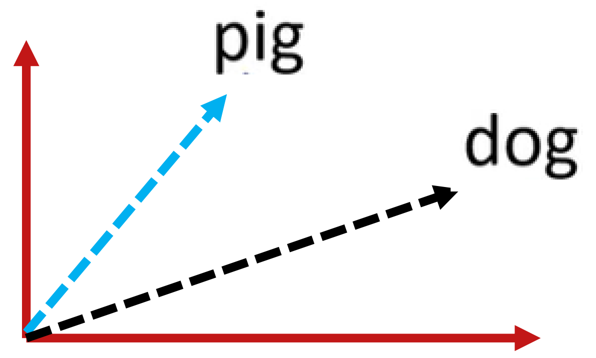
$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \mathbf{U}_q^\intercal \mathbf{U}_k \boldsymbol{p}_n + b_{i,j}$$

- DeBERTa
  - Absolute position embeddings p_m and p_n are replaced with the relative position embeddings $\tilde{p}_{m-n}$

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \tilde{\boldsymbol{p}}_{m-n} + \tilde{\boldsymbol{p}}_{m-n}^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n$$

Combining both Relative and Absolute Positions

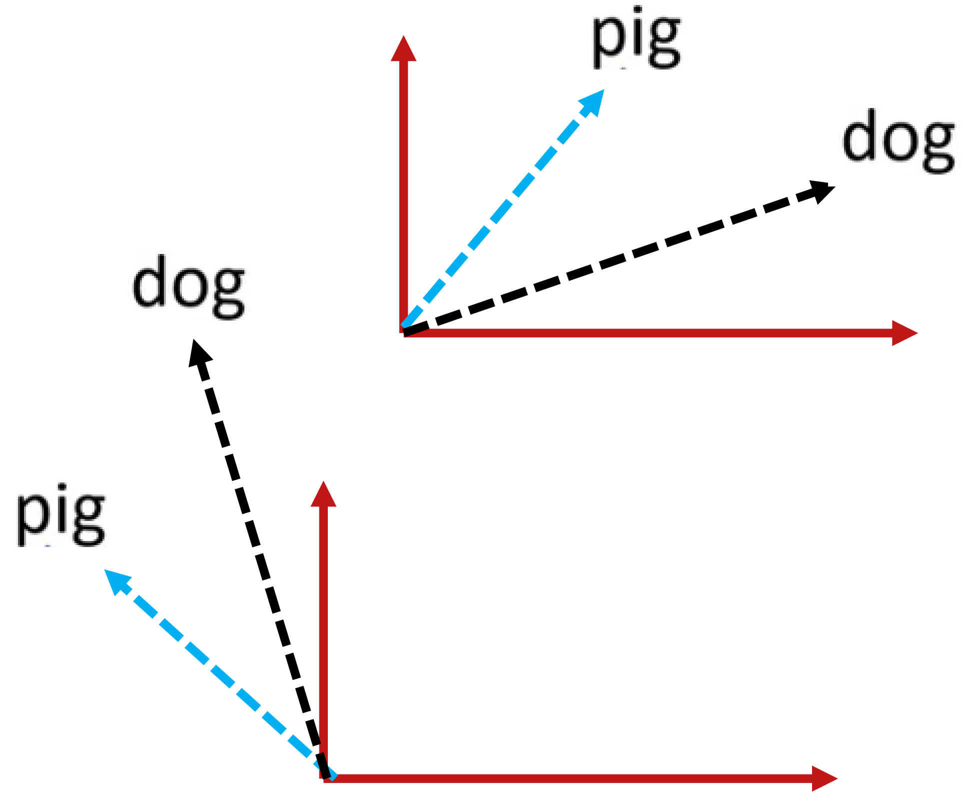
The dog chased the pig



Combining both Relative and Absolute Positions

The dog chased the pig

Once upon a time, the pig chased the dog



Rotary Position Embedding (RoPE)

- Encodes abs pos with a rotation matrix
- Incorporates explicit relative pos dependency in self-attention formulation.
- Require: $q_m^T k_n$ be a function (g) of only word embeddings x_m, x_n, and their relative position m−n

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

- What is a good choice for f_q and f_k?
- For dimension d = 2, a solution is

$$f_q(\boldsymbol{x}_m, m) = (\boldsymbol{W}_q \boldsymbol{x}_m)e^{im\theta}$$
$$f_k(\boldsymbol{x}_n, n) = (\boldsymbol{W}_k \boldsymbol{x}_n)e^{in\theta}$$
$$g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) = \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}]$$

  - Re[·] is real part of a complex number
  - $(W_k x_n)^*$ is conjugate complex number of $(W_k x_n)$.
  - θ ∈ R is a preset non-zero constant.

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

- Euler's formula: $e^{im\theta}$ = cos(mθ) + i sin(mθ)
- The matrix form of $e^{im\theta}$ is $\begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$
- Re[$e^{im\theta}$] = cos(mθ); Im[$e^{im\theta}$] = sin(mθ)
- This matrix captures the rotation by mθ.
- Rotary Position Embedding
  - Rotate the affine-transformed word embedding vector by amount of angle multiples of its position index

Rotary Position Embedding (RoPE): General Form

- Generalizing from 2D to d-dimensions
  - Divide the d-dimension space into d/2 sub-spaces

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

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

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

- Applying RoPE to self-attention

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n$$

$$\boldsymbol{R}^d_{\Theta,n-m} = (\boldsymbol{R}^d_{\Theta,m})^\intercal \boldsymbol{R}^d_{\Theta,n}$$

Rotary Position Embedding (RoPE): General Form

- Generalizing from 2D to d-dimensions
  - Divide the d-dimension space into d/2 sub-spaces

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

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

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

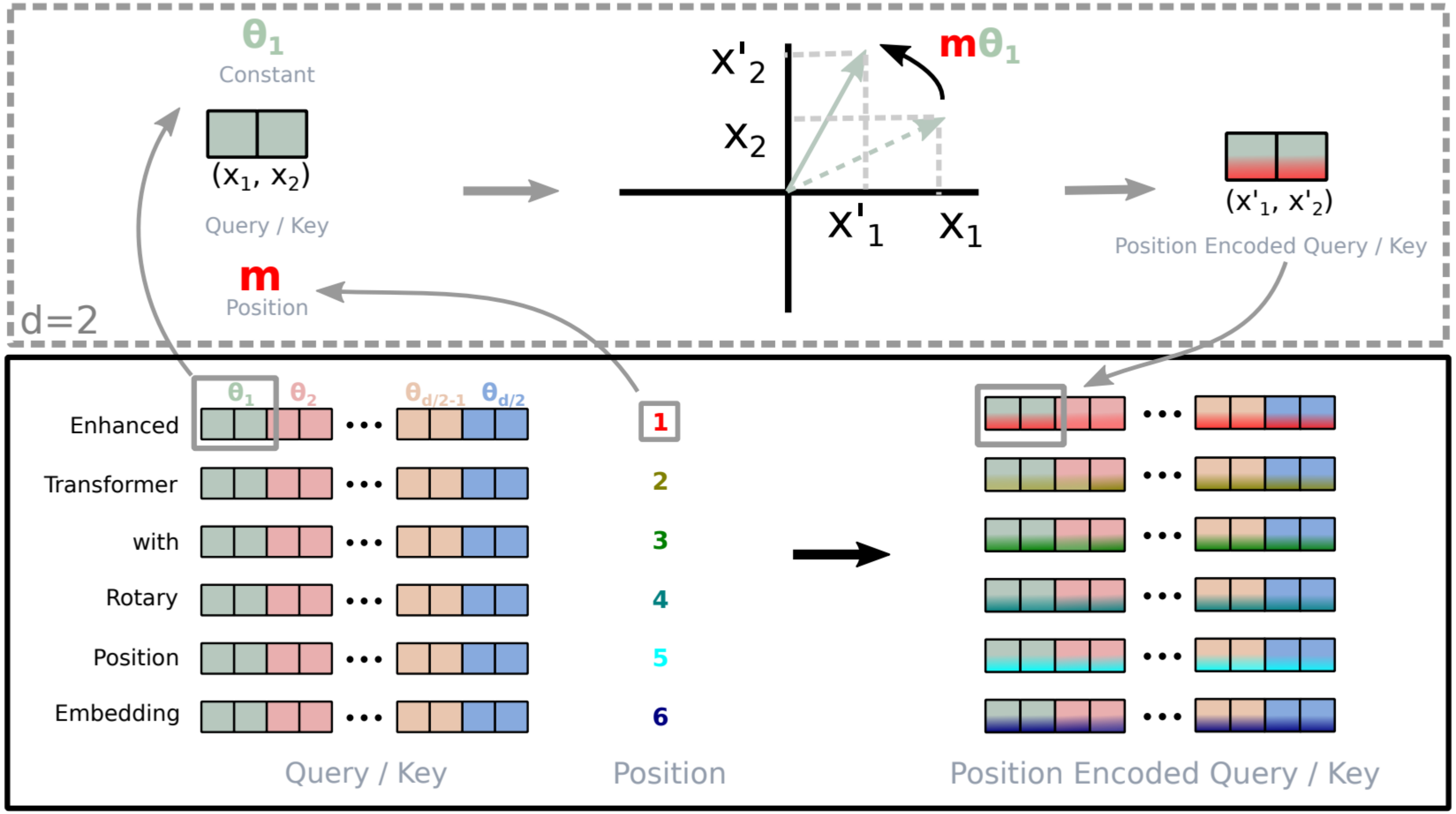
- Applying RoPE to self-attention

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n$$

$$\boldsymbol{R}^d_{\Theta,n-m} = (\boldsymbol{R}^d_{\Theta,m})^\intercal \boldsymbol{R}^d_{\Theta,n}$$

- In contrast to earlier position embedding methods, RoPE is multiplicative.
- RoPE naturally incorporates relative pos info through rotation matrix product instead of altering terms in the expanded formulation of additive position encoding when applied with self-attention.
- RoPE
  - Represents token embeddings as complex numbers
  - Represents their positions as pure rotations
  - 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, thus the angle between them will remain unchanged and thus the dot product will also remain unchanged.

RoPE Implementation

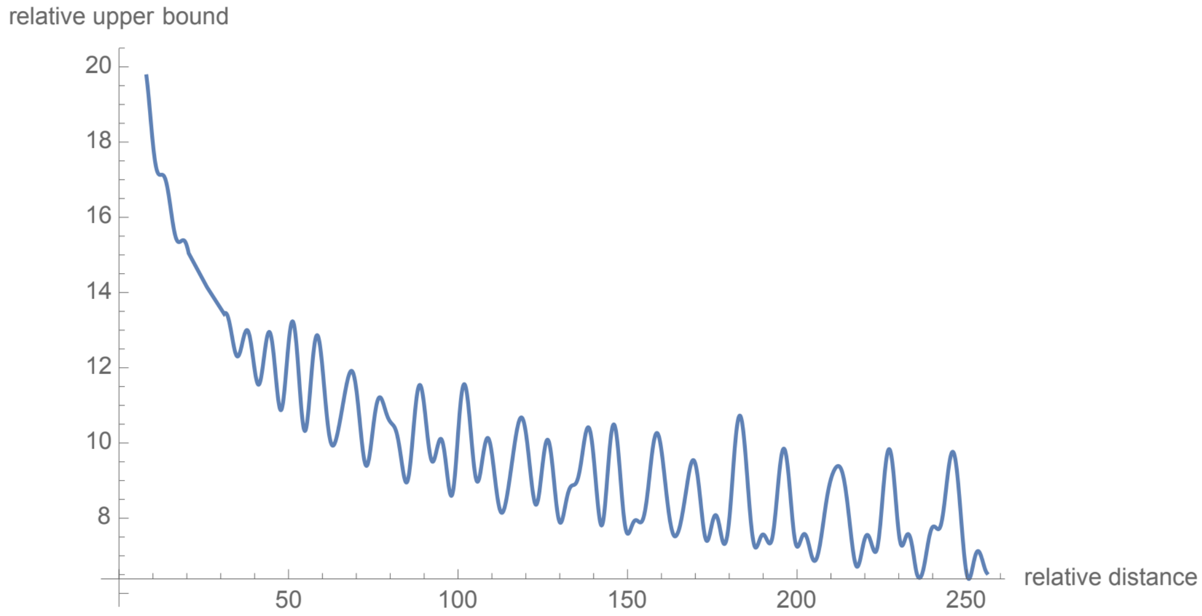


$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$$

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

Properties of RoPE

- Long-term decay
  - Inner-product decays when the relative position increase.
  - A pair of tokens with a long relative distance should have less connection.



- Computational efficient realization of rotary matrix multiplication

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

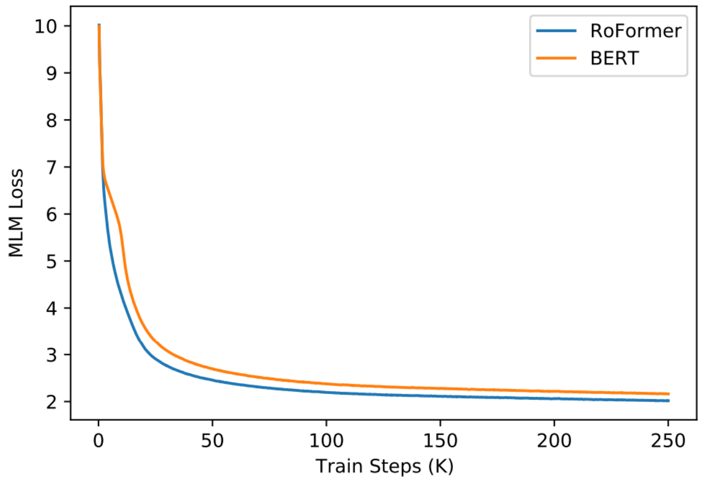
$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots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

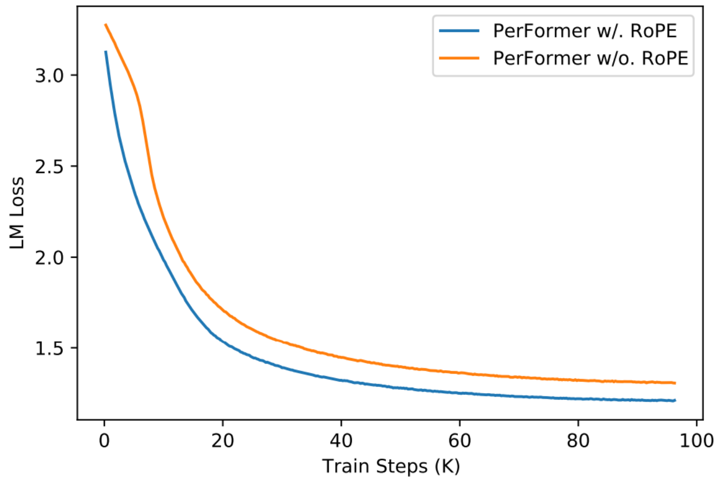
$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix}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

RoPE Performance

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

WMT 2014 English-to-German translation task



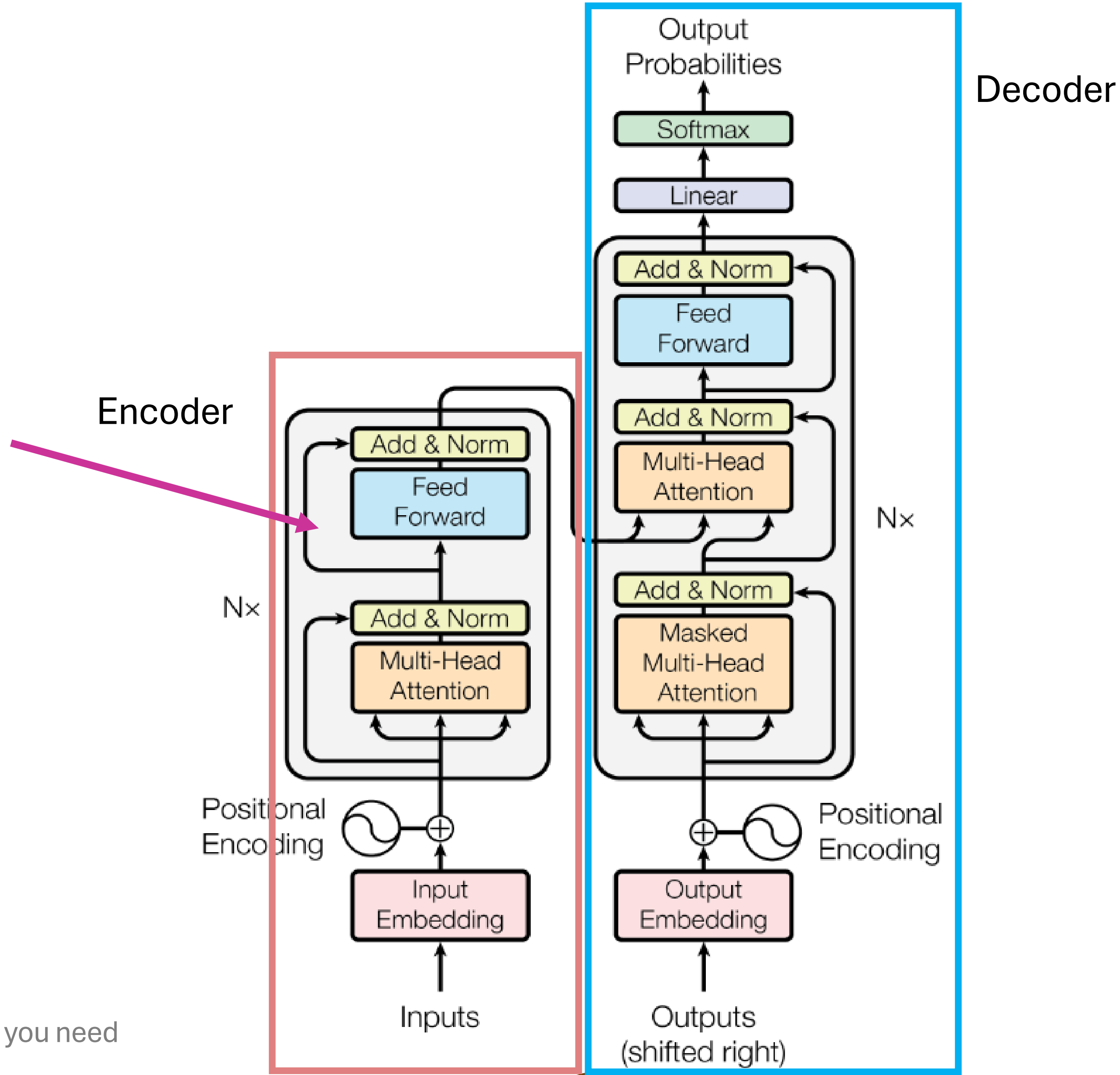


Language modeling pre-training. Left: training loss. Right: training loss for PerFormer with and without RoPE.

Table 2: Comparing RoFormer and BERT by fine tuning on downstream GLEU tasks.

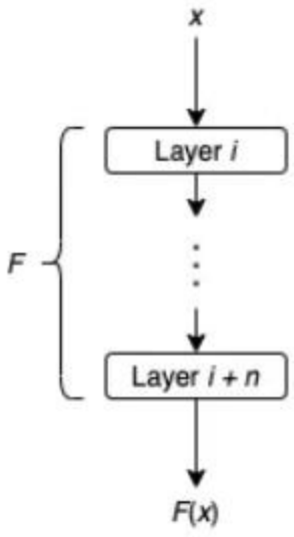
Model	MRPC	SST-2	QNLI	STS-B	QQP	MNLI(m/mm)
BERTDevlin et al. [2019]	88.9	93.5	90.5	85.8	71.2	84.6/83.4
RoFormer	**89.5**	90.7	88.0	**87.0**	**86.4**	80.2/79.8

Transformer Architecture

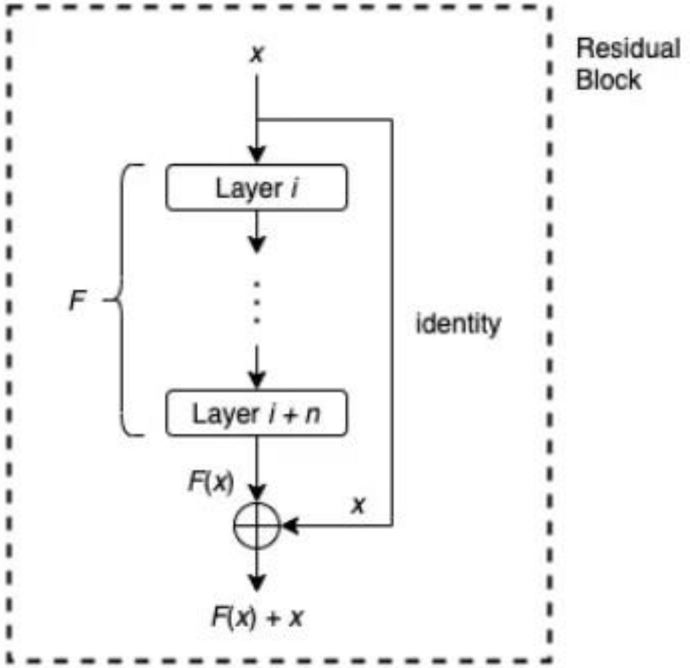


Source of Image : Attention is all you need (Vaswani t al., 2017)

Residual Connection

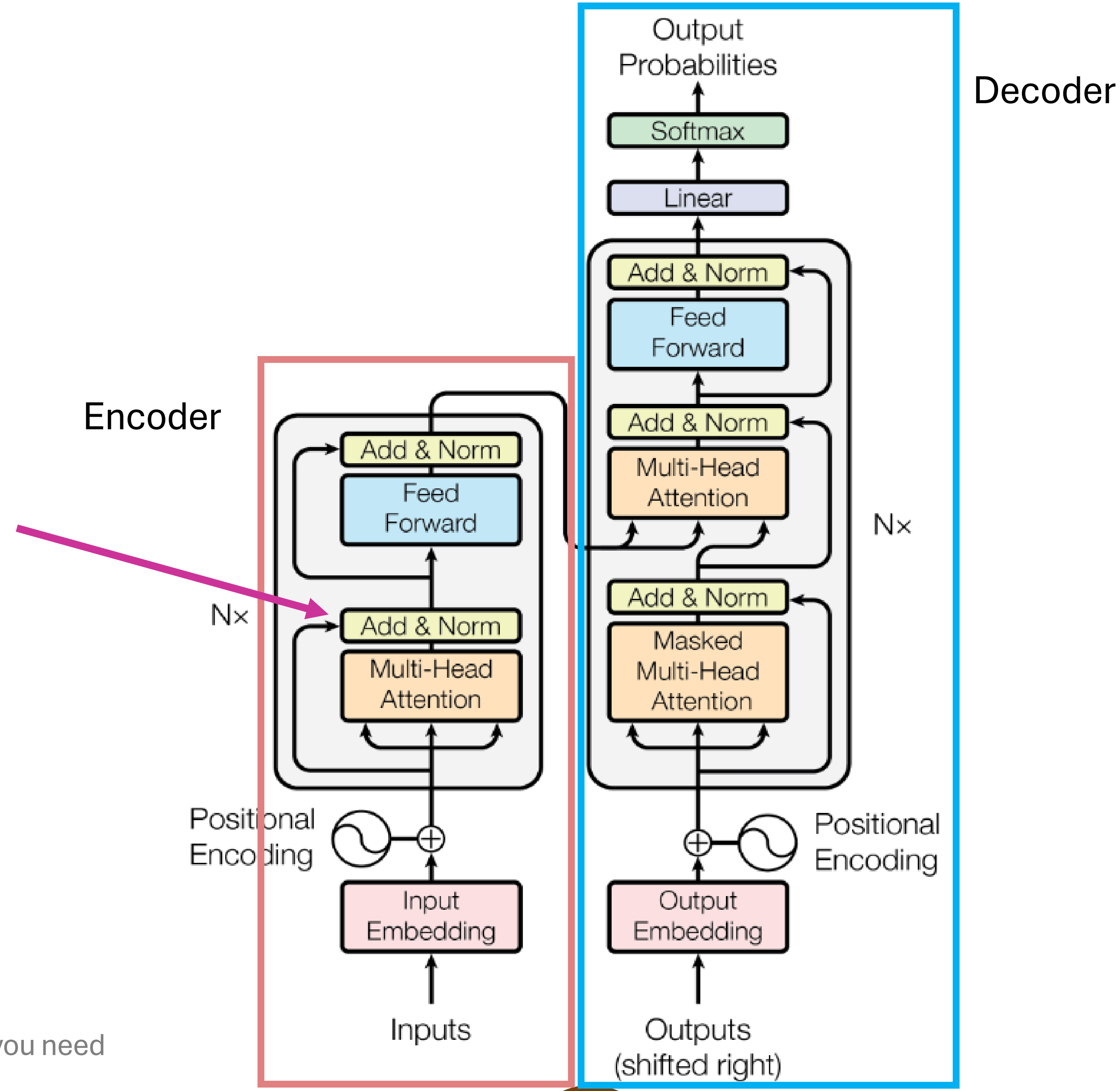


Traditional Feedforward
without Residual Connection



With Residual Connection

Transformer Architecture



Source of Image : Attention is all you need (Vaswani t al., 2017)

Batch and Layer Normalization

Normalization

- Example: student loans with the age of the student and the tuition as two input features
  - two values are on totally different scales.
    - the age of a student will have a median value in the range 18 to 25 years
    - the tuition could take on values in the range $20K - $50K for a given academic year.

Normalization works by mapping all values of a feature to be in the range [0,1] using the transformation

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Suppose a particular input feature `x` has values in the range `[x_min, x_max]`. When `x` is equal to `x_min`, `x_norm` is equal to 0 and when `x` is equal to `x_max`, `x_norm` is equal to 1. So for all values of `x` between `x_min` and `x_max`, `x_norm` maps to a value between 0 and 1.

Standardization

- Example: student loans with the age of the student and the tuition as two input features
  - two values are on totally different scales.
    - the age of a student will have a median value in the range 18 to 25 years
    - the tuition could take on values in the range $20K - $50K for a given academic year.

Standardization transforms the input values such that they follow a distribution with zero mean and unit variance.

$$x_{std} = \frac{x - \mu}{\sigma}$$

In practice, this process of *standardization* is also referred to as *normalization*

Batch Normalization

- For a network with hidden layers, the output of layer k-1 serves as the input to layer k
- Split the dataset into multiple batches and run the mini-batch gradient descent.
- The mini-batch gradient descent algorithm optimizes the parameters of the neural network by batch-wise processing of the dataset, one batch at a time.
- The input distribution at a particular layer keeps changing across batches.
- **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift refers to this change in distribution of the input to a particular layer across batches as internal covariate shift.**
- For instance, if the distribution of data at the input of layer K keeps changing across batches, the network will take longer to train.

Batch Normalization

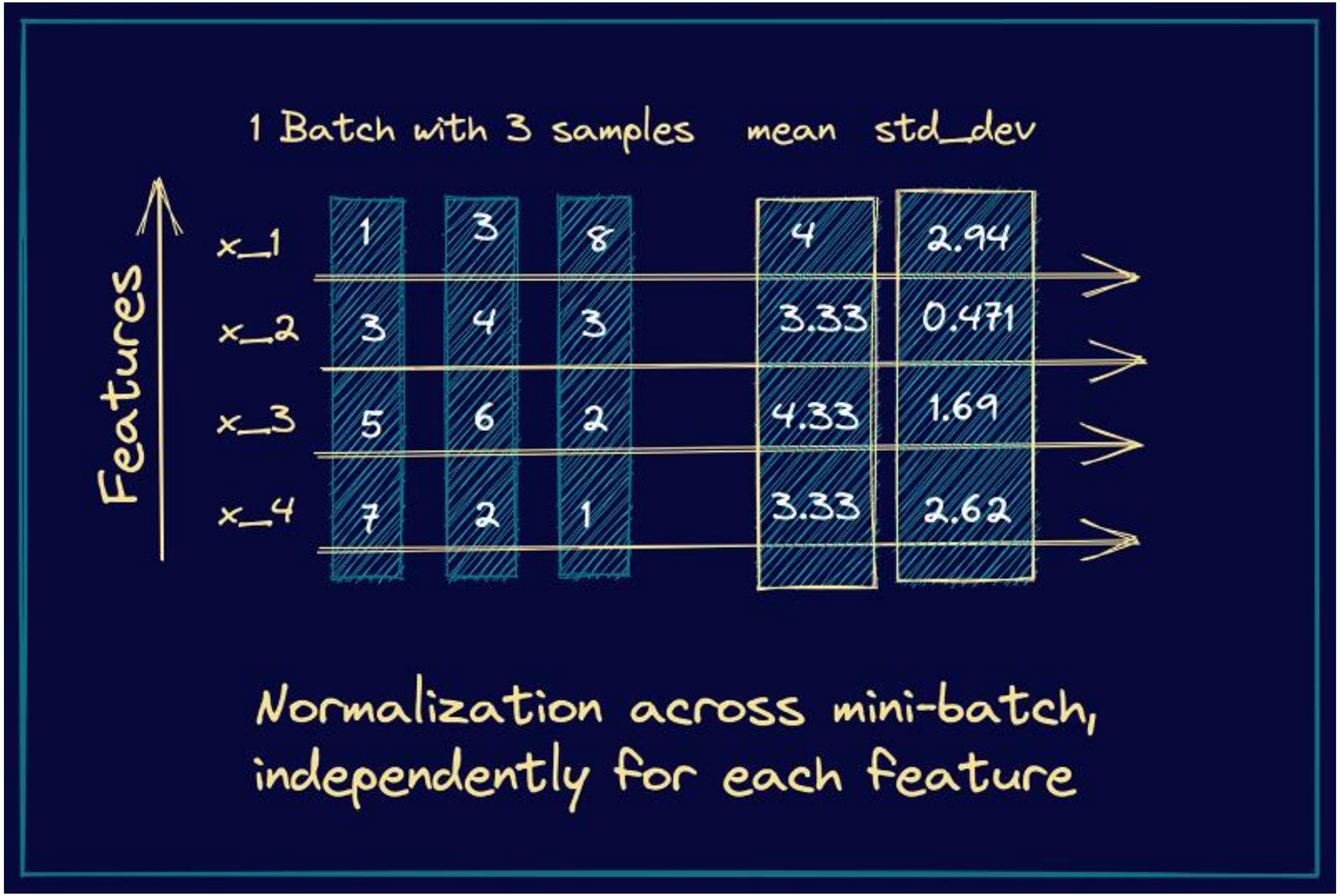


Image Source: https://www.pinecone.io/learn/batch-layer-normalization/

Batch Normalization

- Forcing all the pre-activations to be zero and unit standard deviation across all batches can be too restrictive.
- It may be the case that the fluctuant distributions are necessary for the network to learn certain classes better.

$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} x_i \ (1)$$

$$\sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_b)^2 \ (2)$$

$$\hat{x_i} = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2}} (3)$$

$$or\ \hat{x_i} = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \ (3)$$

$$Adding\ \epsilon\ helps\ when\ \sigma_b^2\ is\ small$$

$$y_i = \mathcal{BN}(x_i) = \gamma . x_i + \beta \ (4)$$

Trainable parameters

Batch Normalization: Limitations

- In batch normalization, we use **batch statistics**: mean and standard deviation for current mini-batch.
  - When **batch size is small, the sample mean and sample standard deviation are not representative enough** of the actual distribution and the network cannot learn anything meaningful.
- As batch normalization depends on batch statistics for normalization, it is **less suited for sequence models**.
  - This is because, in sequence models, we may have sequences of potentially different lengths and smaller batch sizes corresponding to longer sequences.

Layer Normalization

- All neurons in a particular layer effectively have the same distribution across all features for a given input

> If each input has d features, it's a d-dimensional vector. If there are B elements in a batch, the normalization is done along the length of the d-dimensional vector and not across the batch of size B.

- Normalizing *across all features* but for each of the inputs to a specific layer removes the dependence on batches.
- This makes layer normalization well suited for sequence models such as Transformers and RNNs.

Layer Normalization

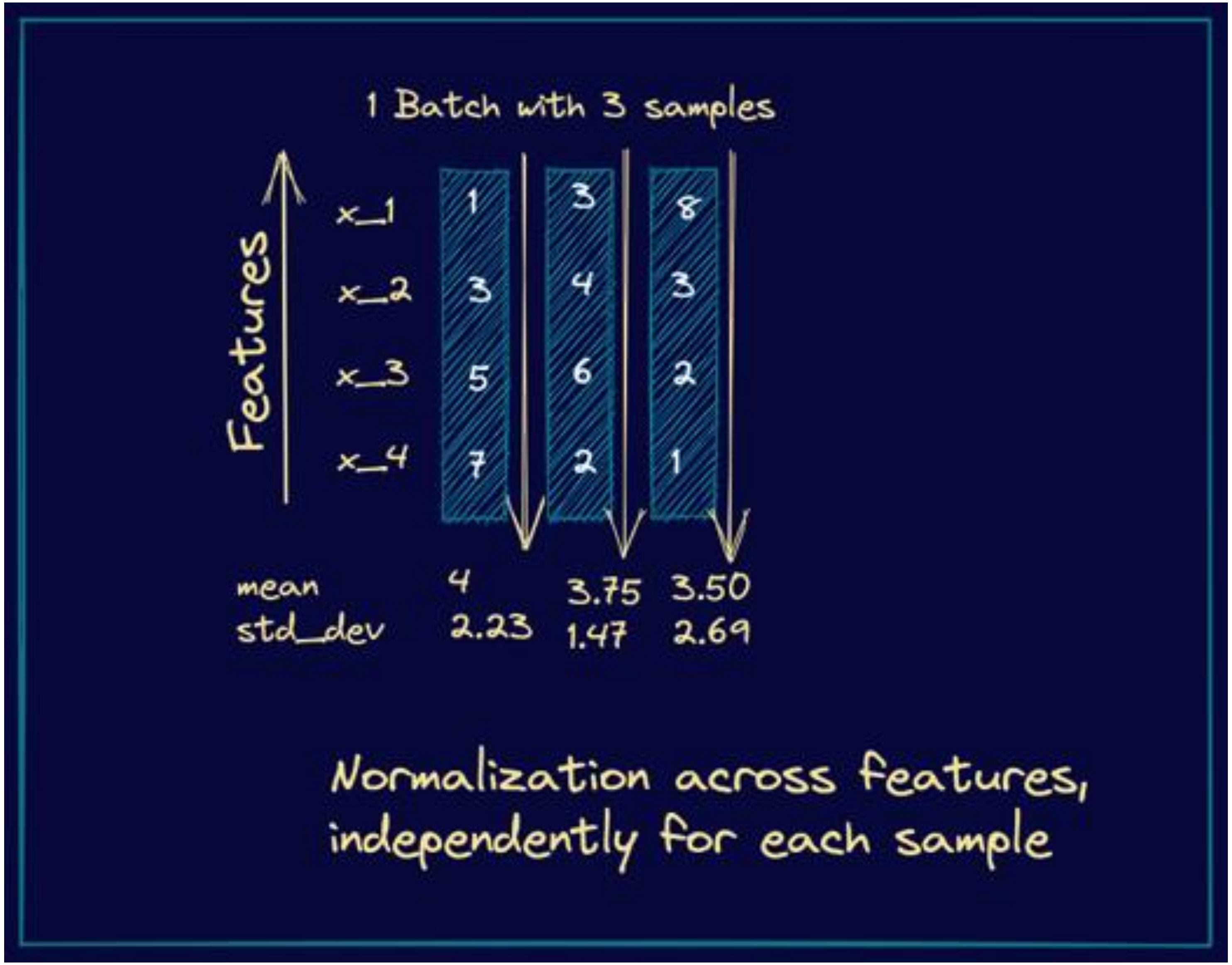


Image Source: https://www.pinecone.io/learn/batch-layer-normalization/

Layer Normalization

$$\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i \quad (1)$$

$$\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2 \quad (2)$$

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2}} \quad (3)$$

$$or\ \hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \quad (3)$$

$$Adding\ \epsilon\ helps\ when\ \sigma_l^2\ is\ small$$

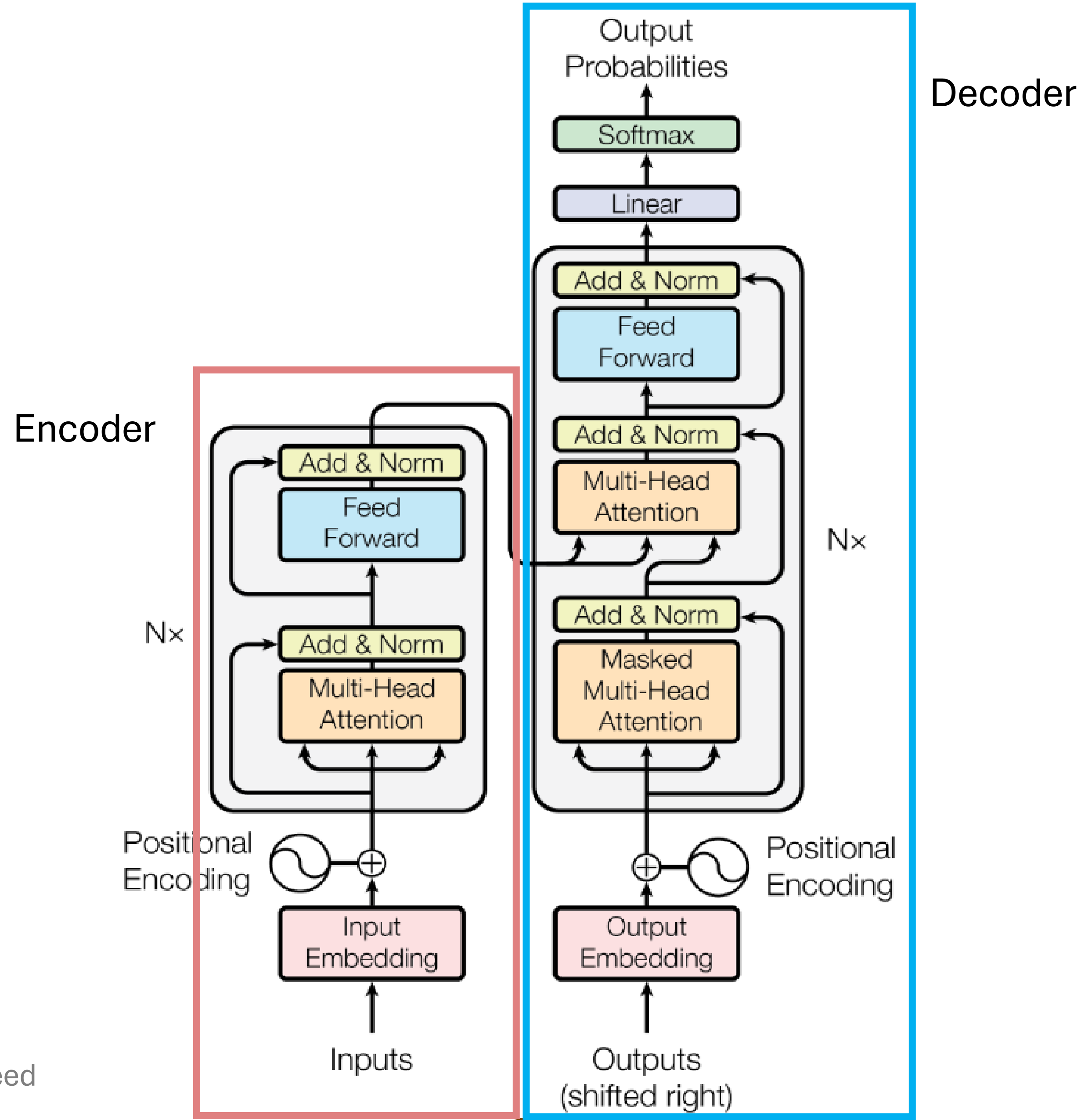
$$y_i = \mathcal{LN}(x_i) = \gamma . x_i + \beta \quad (4)$$

Trainable parameters

Batch Normalization vs Layer Normalization

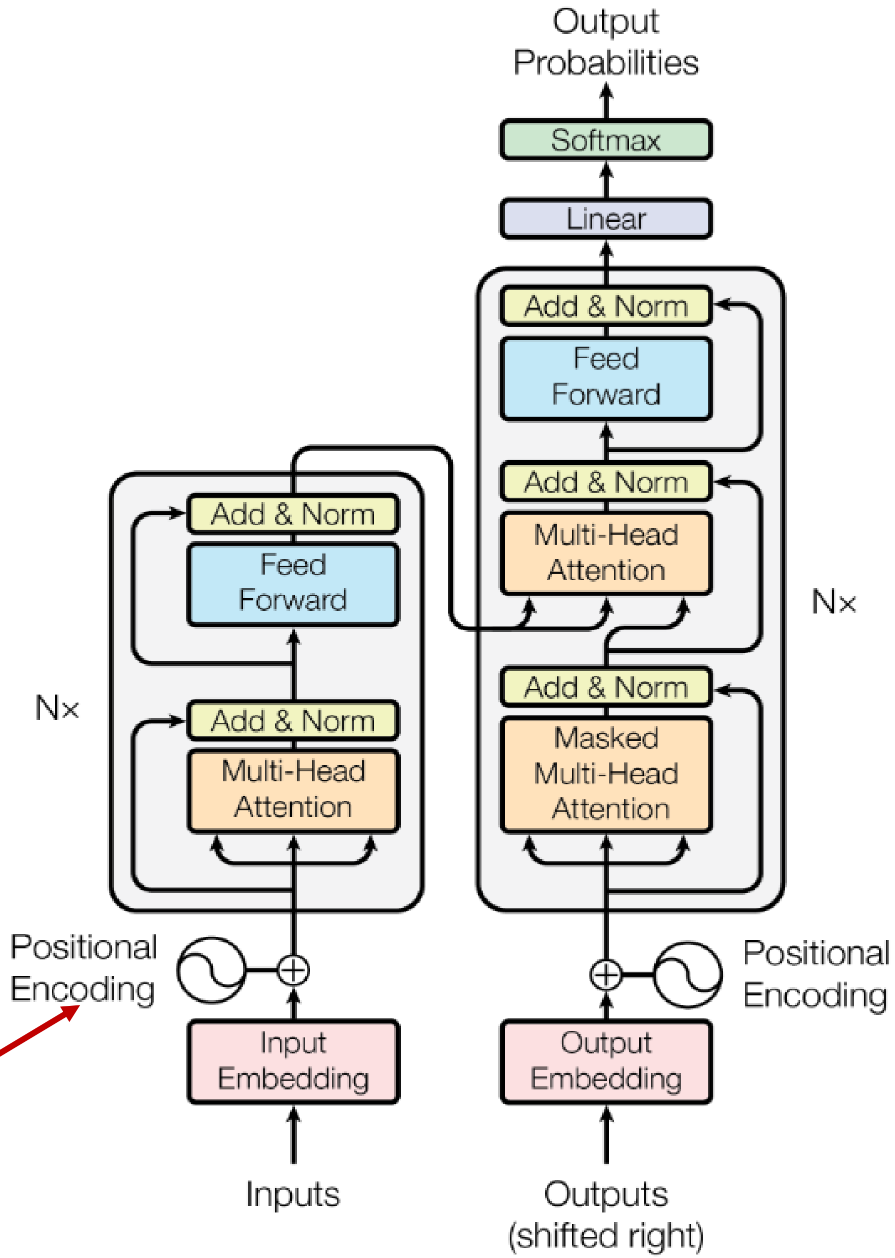
- Batch normalization normalizes each feature independently across the mini-batch. Layer normalization normalizes each of the inputs in the batch independently across all features.
- As batch normalization is dependent on batch size, it's not effective for small batch sizes. Layer normalization is independent of the batch size, so it can be applied to batches with smaller sizes as well.
- Batch normalization requires different processing at training and inference times. As layer normalization is done along the length of input to a specific layer, the same set of operations can be used at both training and inference times.

Transformer Architecture



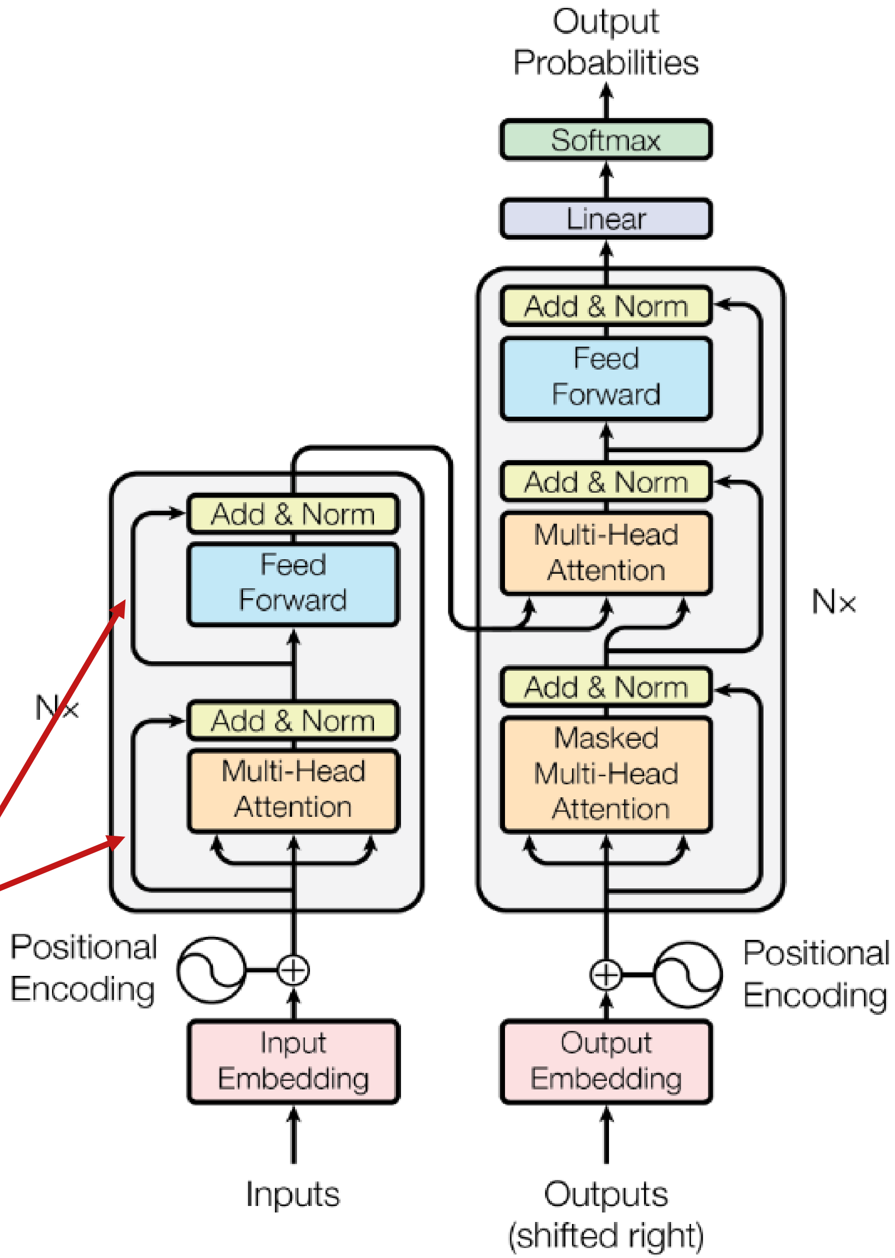
Source of Image : Attention is all you need (Vaswani t al., 2017)

Transformer Architecture



Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!

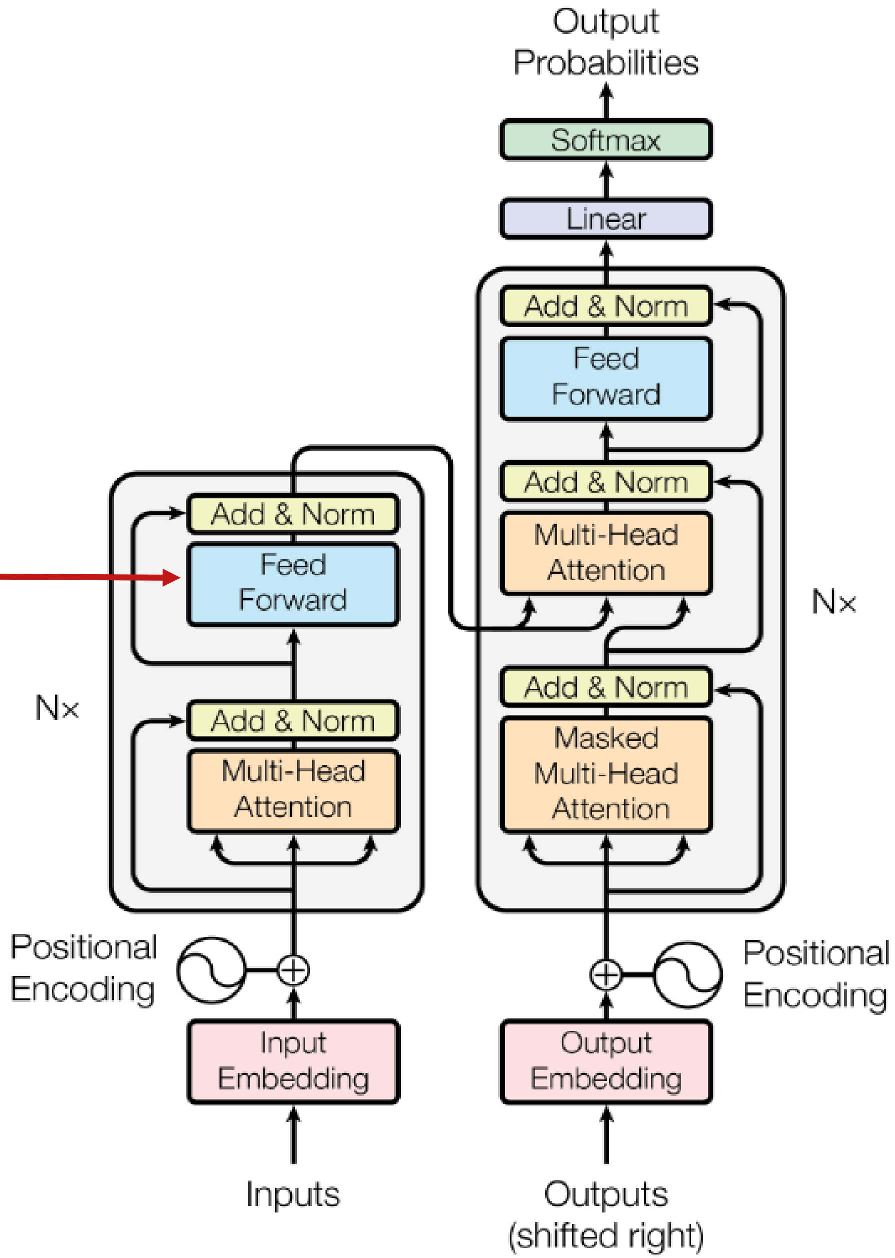
Transformer Architecture



Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow

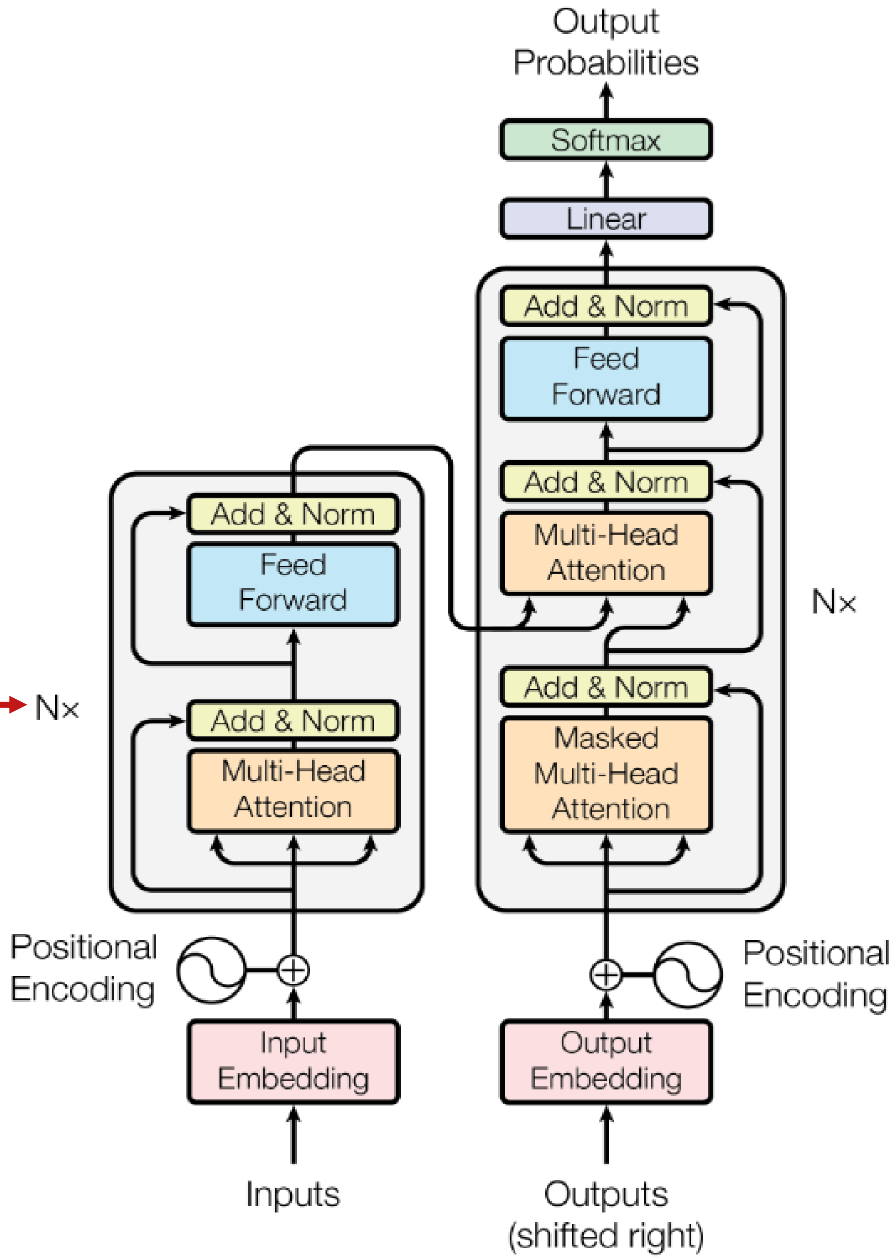
Transformer Architecture

A **feed-forward layer** on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity

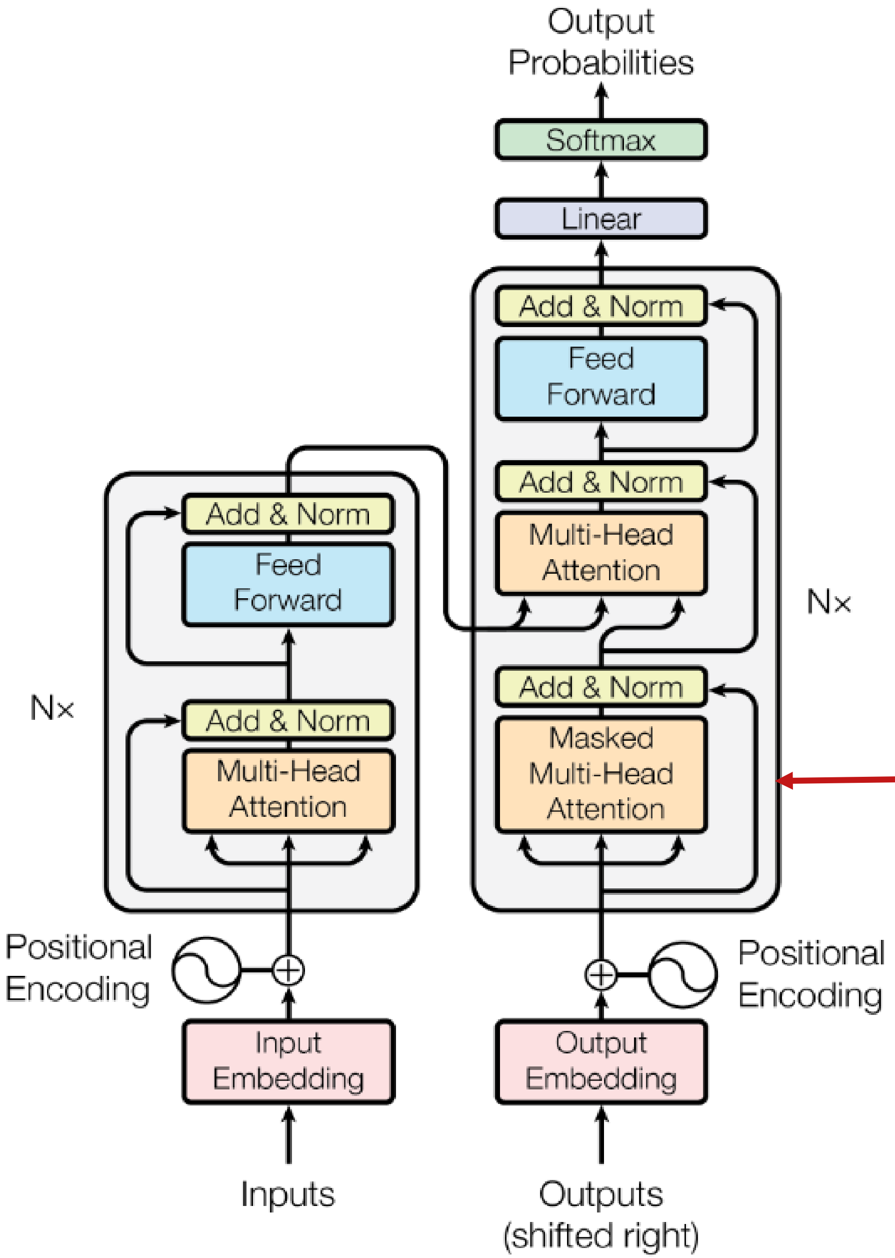


Transformer Architecture

We stack as many of these ***Transformer* blocks** on top of each other as we can (bigger models are generally better given enough data!)

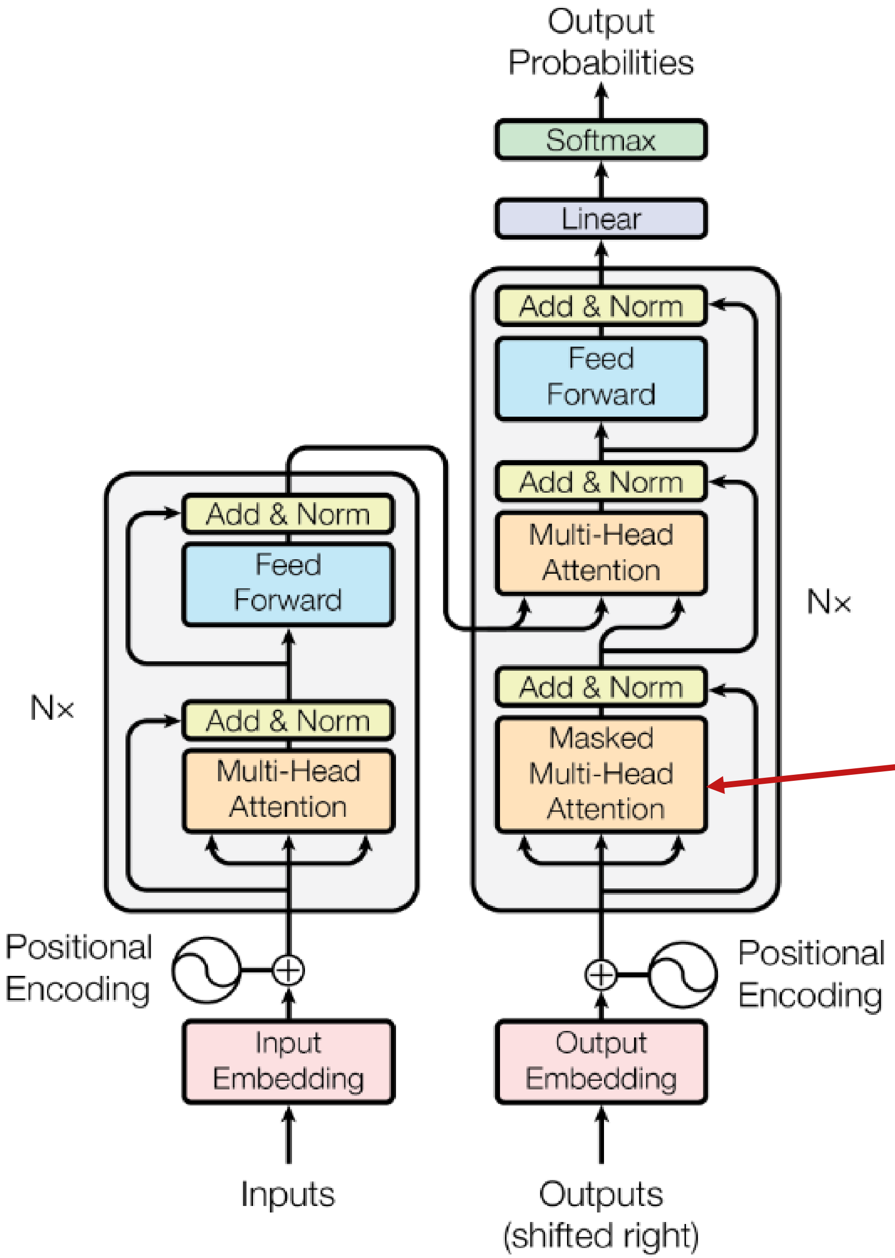


Transformer Architecture



Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)

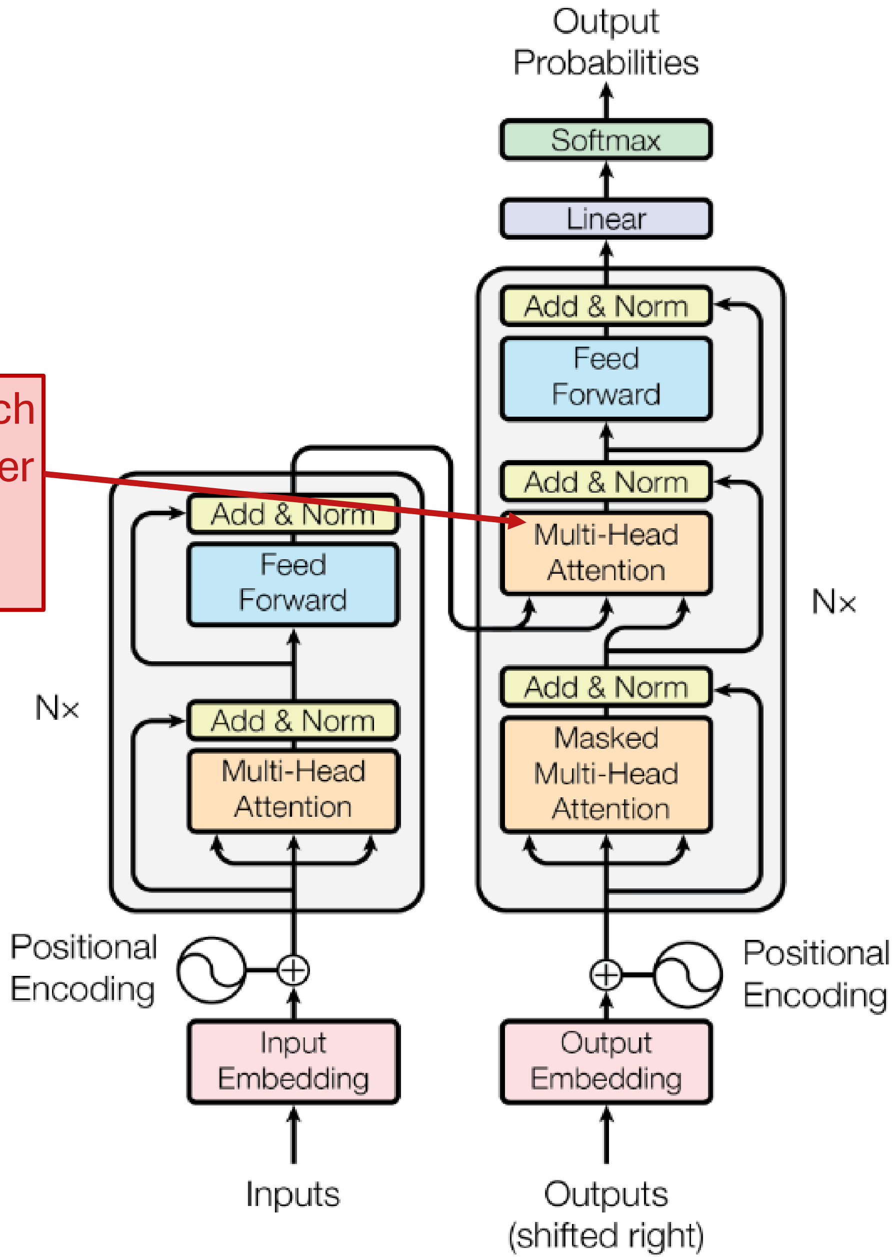
Transformer Architecture



We first have an instance of **masked self attention**. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

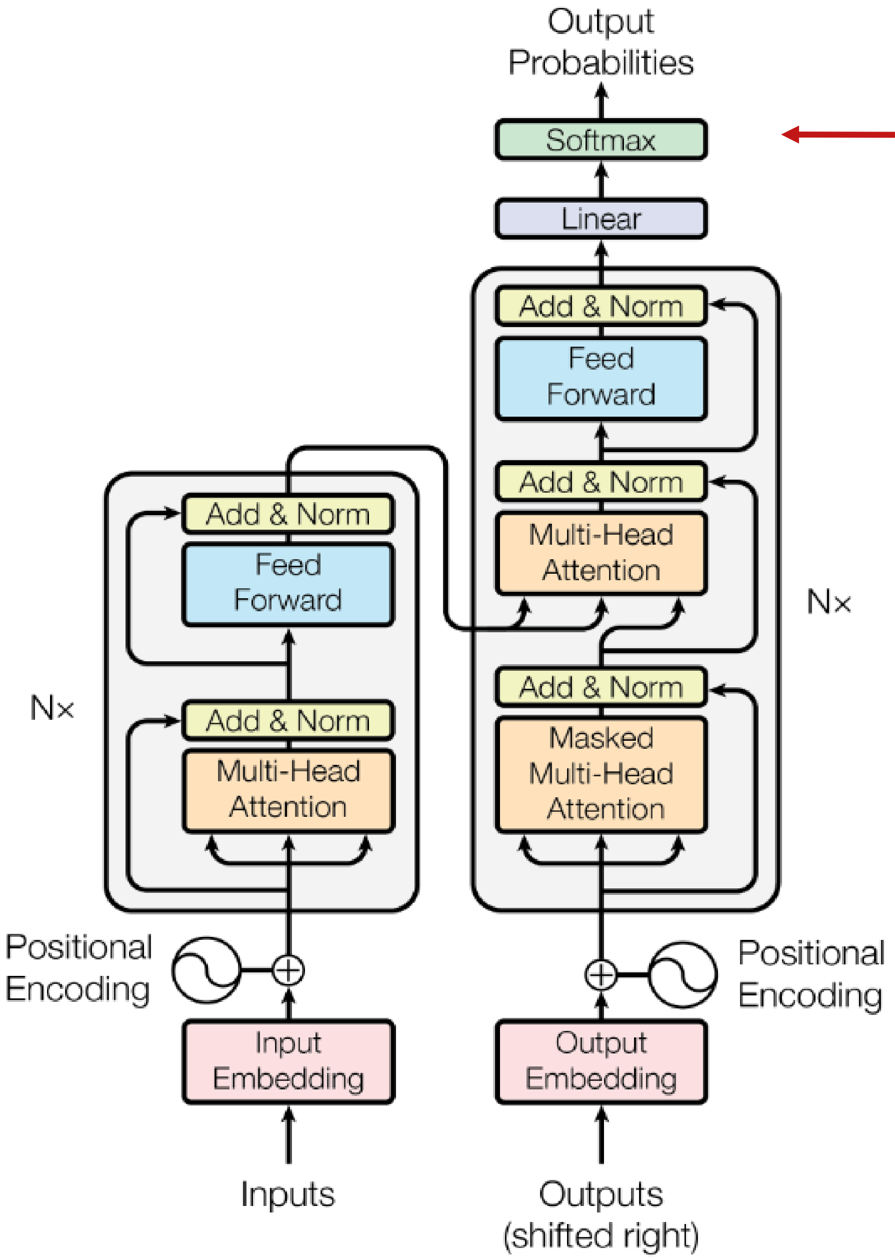
Transformer Architecture

Now, we have ***cross attention***, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



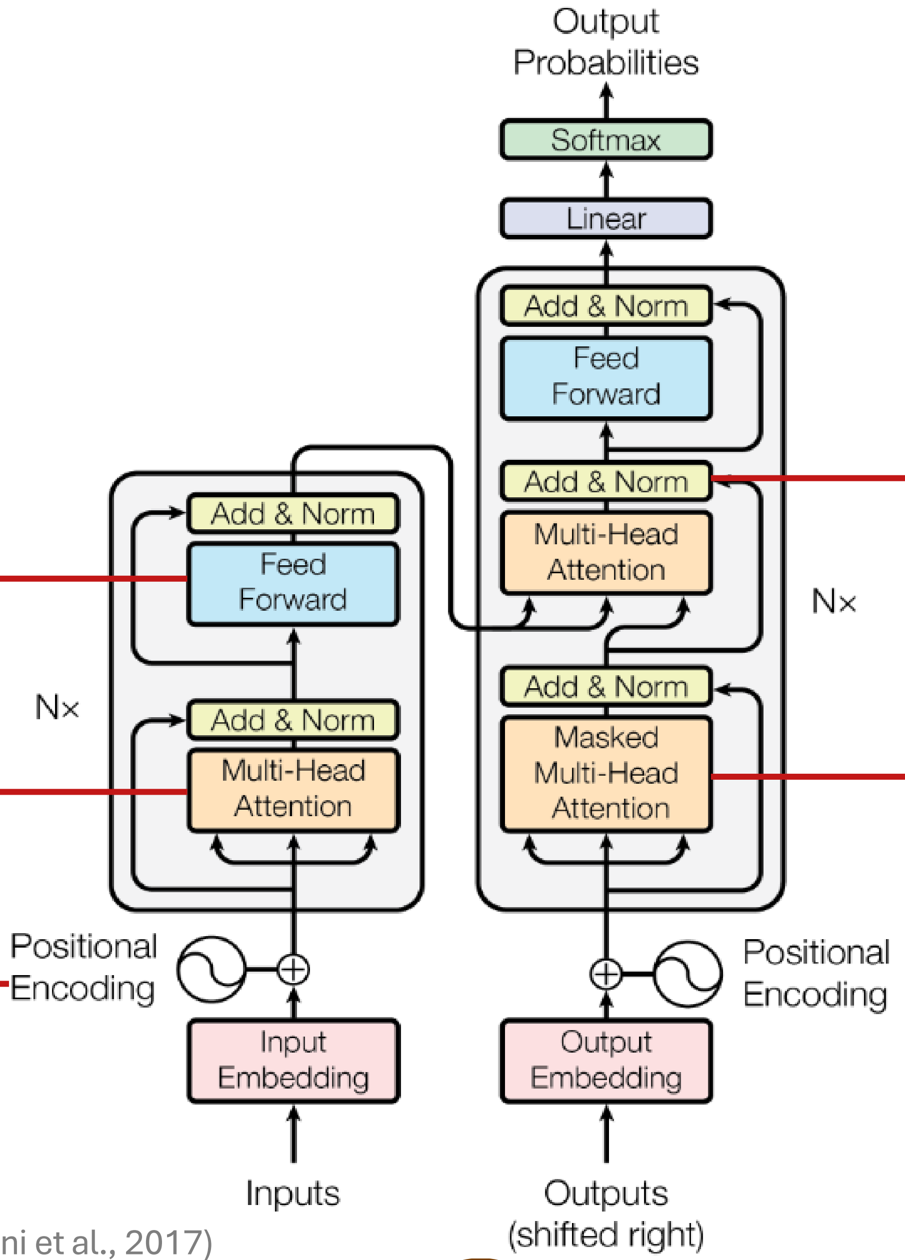
Source of Image : Attention is all you need (Vaswani et al., 2017)

Transformer Architecture



After stacking a bunch of these decoder blocks, we finally have our familiar **softmax** layer to predict the next English word.

Transformer Architecture



Adding non-linearities

Allows querying multiple positions at each layer

Adds positional information

Reduces covariance shift and makes the system stable

Prevents attention lookups into the future while decoding

Source of Image : Attention is all you need (Vaswani et al., 2017)