

# Deep Generative Models: Transformers

Fall Semester 2025

René Vidal

Director of the Center for Innovation in Data Engineering and Science (IDEAS)  
Rachleff University Professor, University of Pennsylvania  
Amazon Scholar & Chief Scientist at NORCE



# Taxonomy of Generative Models

What we've learned:

- MMs, HMMs,
- LDSs, RNNs, SSMs

## Deep Generative Models

**Autoregressive  
models**  
(e.g., PixelCNN)

**Flow-based  
models**  
(e.g., RealNVP)

**Latent variable  
models**

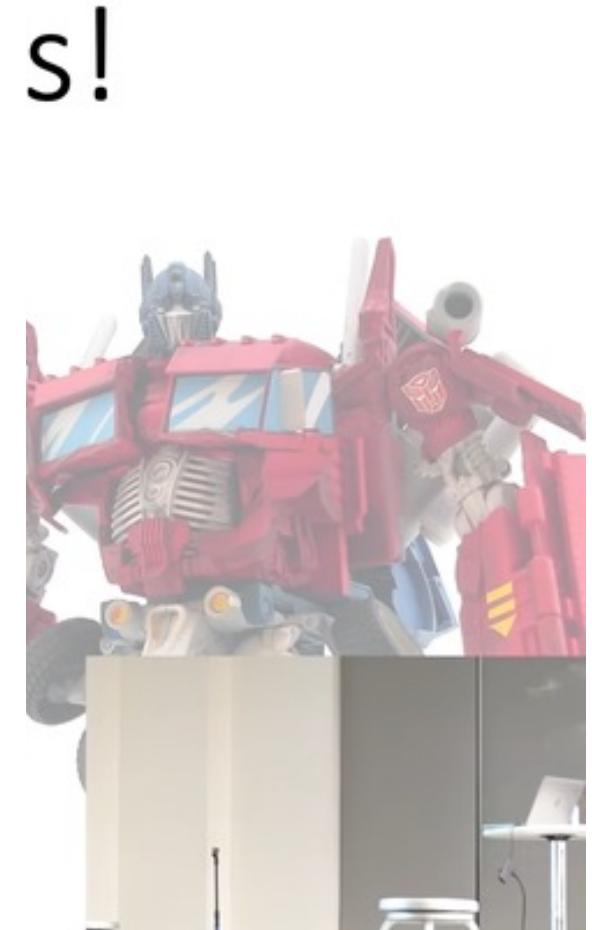
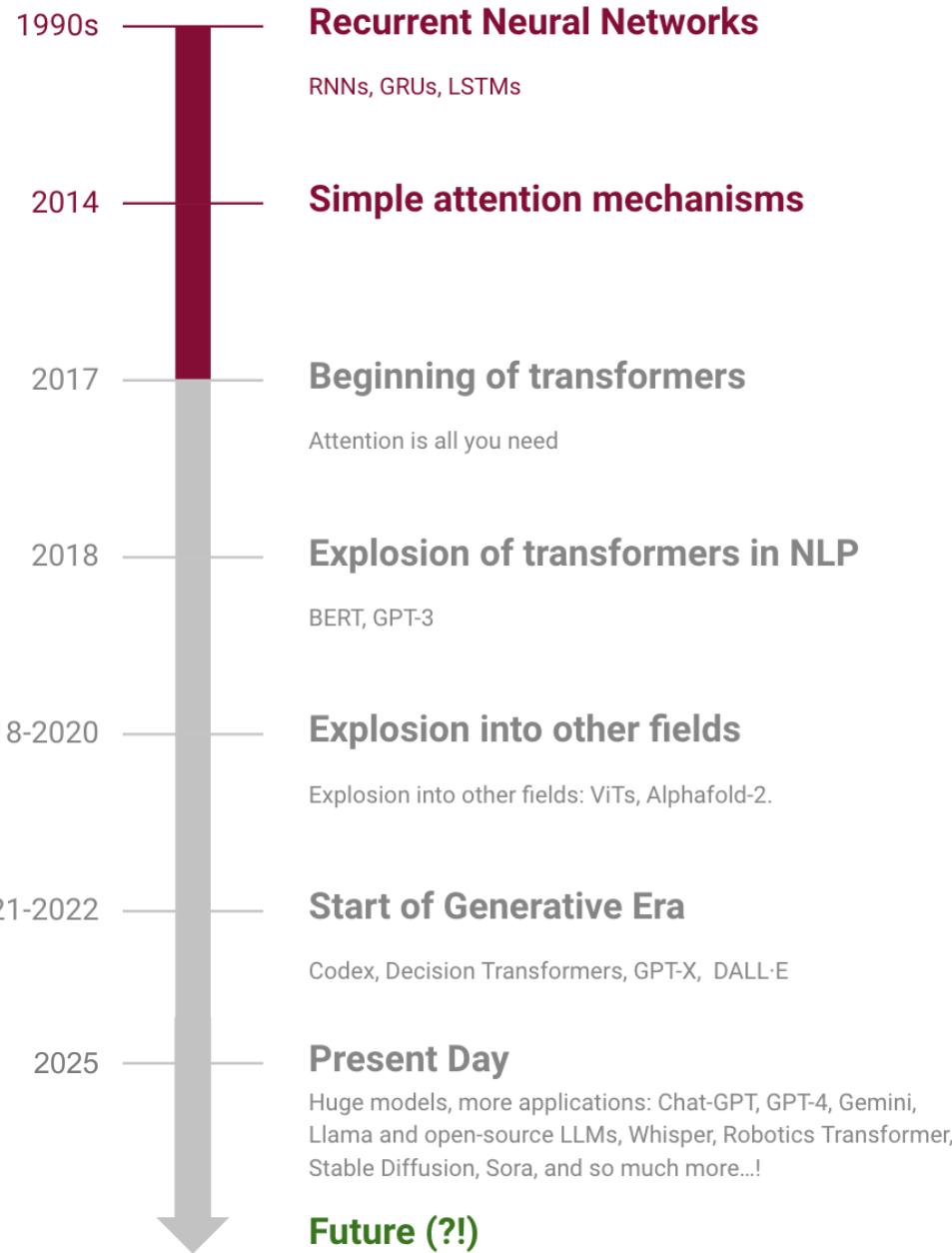
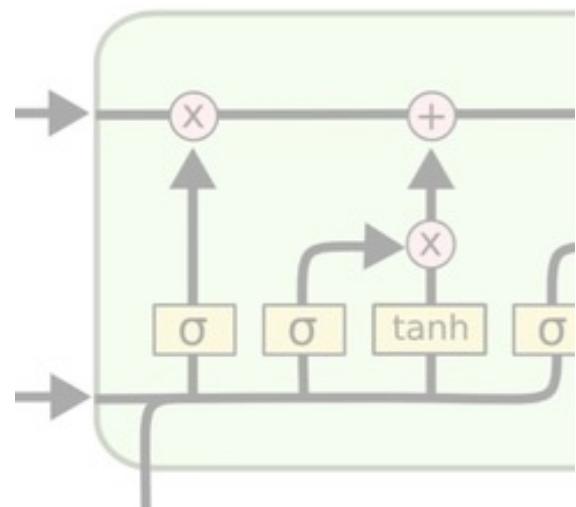
What we've learned:

- PPCA
- VAE

**Energy-based  
models**

What we study now:  
Transformers

What we have learned:  
• Diffusion Models



# RNN vs Transformers: Why RNNs fall short?

- **RNNs**
  - **Hard to capture long-term dependencies** without modifying the architecture.
  - **Hard to train** due to vanishing and exploding gradients.
  - **Hard to process in parallel** due to sequential nature.
- **Transformers: A non-recurrent solution that solely relies on “attention”:**
  - **No reliance on recurrence:** Transformers capture dependencies across all input *tokens* (*words*) simultaneously, processing the entire sequence at once. This allows for parallel computation, unlike RNNs that rely on sequential processing.
  - **Captures global dependencies:** The attention mechanism enables modeling of long-range dependencies without the vanishing gradient problem.

# Recall the Translate and Align Model in RNNs

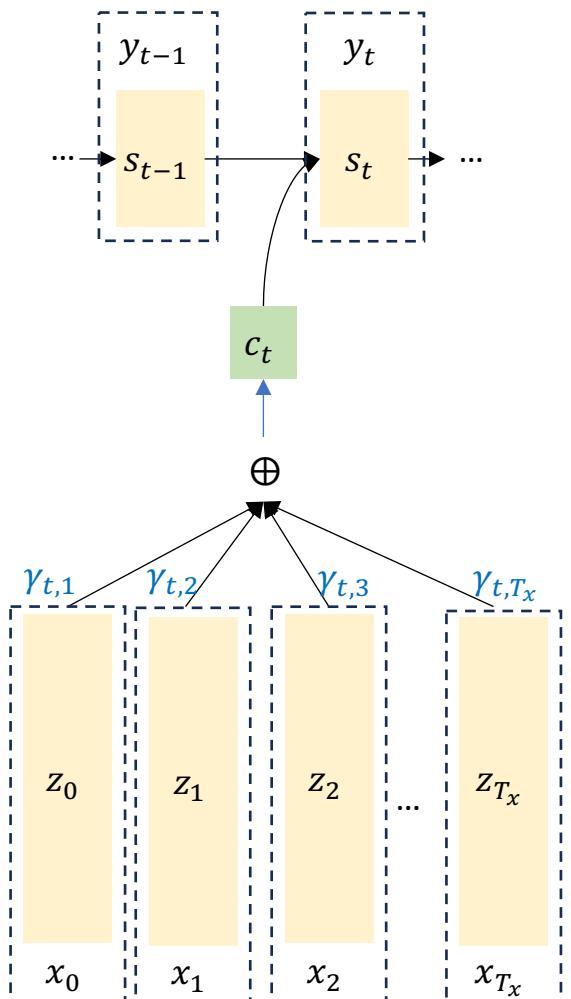
- **Problem:** Given source sentence  $\mathbf{x} = (x_1, \dots, x_T)$ , build a model that generates target sentence  $\mathbf{y} = (y_1, \dots, y_T)$ .
- **Model:** RNN encoder & decoder map source and target sentences to hidden states  $\mathbf{z}$  &  $\mathbf{s}$ , and **context vector**  $c_t$  is computed as a weighted sum of the hidden states  $z_i$ :

$$c_t = \sum_{i=1}^{T_x} \gamma_{t,i} z_i \quad \gamma_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})} \quad e_{ti} = a(z_i, s_{t-1})$$

Context vector    Weights of hidden states    Alignment model

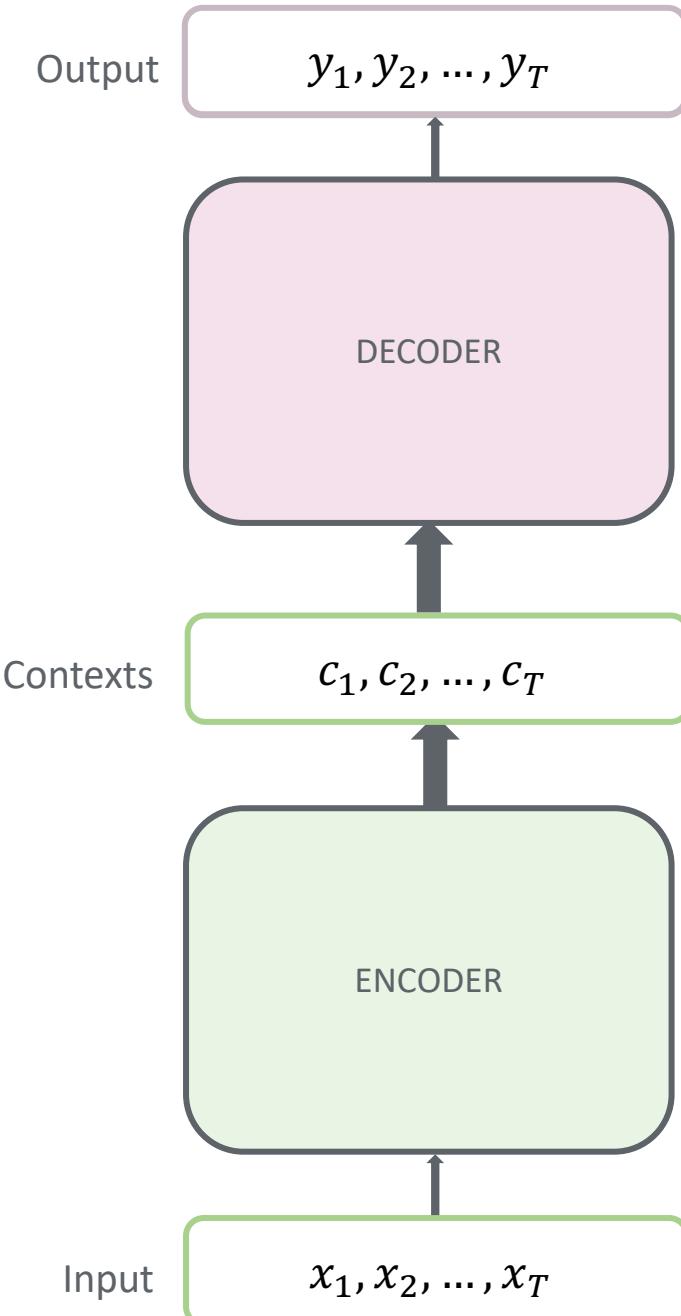
- **Attention mechanism:**

- $a$  is the **alignment model**, typically a feedforward network, and measures how well source word  $x_i$  and target word  $y_t$  match
- $\gamma_{t,i}$  is the probability that the target word  $y_t$  is aligned to, or translated from, a source word  $x_i$ .
- $c_t$  is the expectation of the hidden state w.r.t. distribution  $\gamma_{t,i}$ .



# From RNNs to Transformers

- Let's keep what is good from Align & Translate:
  - Use encoder to learn **latent representation of source sentence**
  - Use decoder to learn **latent representation of target sentence**
  - **Align the latent representations** of the source/target sentences and form **global contexts**
  - Use decoder to map contexts to target sentences



# Transformer

- Let's keep what is good from Align & Translate:
  - Use encoder to learn **latent representation of source sentence**
  - Use decoder to learn **latent representation of target sentence**
  - Align the latent representations of the source/target sentences and form **global contexts**
  - Use decoder to map contexts to target sentences

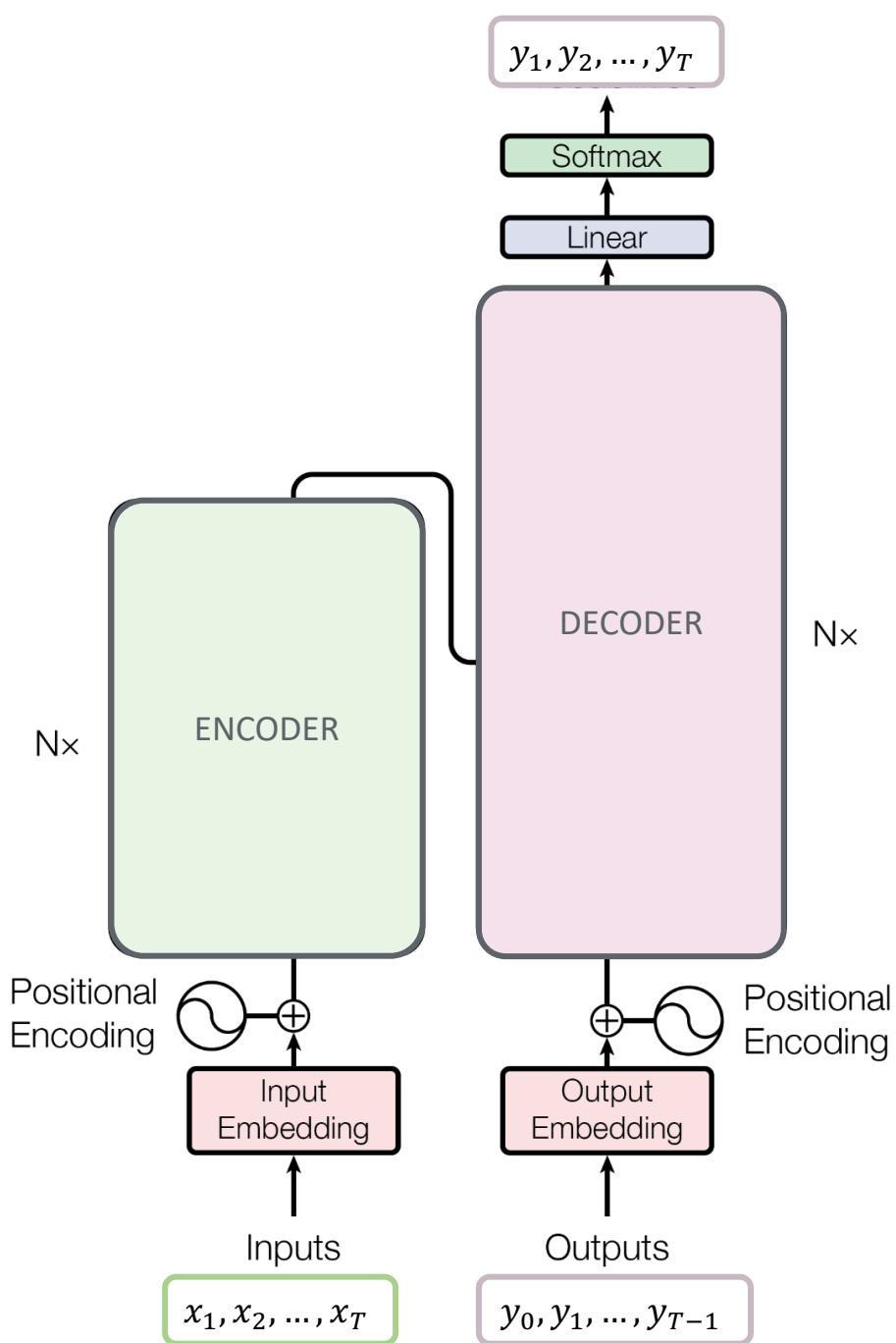


Figure 1: The Transformer - model architecture.

# Word to Word Embedding

- First, just like any RNN language tasks, we convert our one-hot vector into embeddings through a word embedding
- Given a sentence, a sequence of one-hot vectors,  $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_T), \tilde{x}_t \in \{0, 1\}^N$
- We obtain the embedding for each word by

$$x_t = \mathbf{E}\tilde{x}_t$$

- Again  $\mathbf{E} \in \mathbb{R}^{d \times N}$  is the embedding matrix, and can be pre-trained or learned end-to-end
- In the context of transformers,  $x_t$  is also known as a *token*.

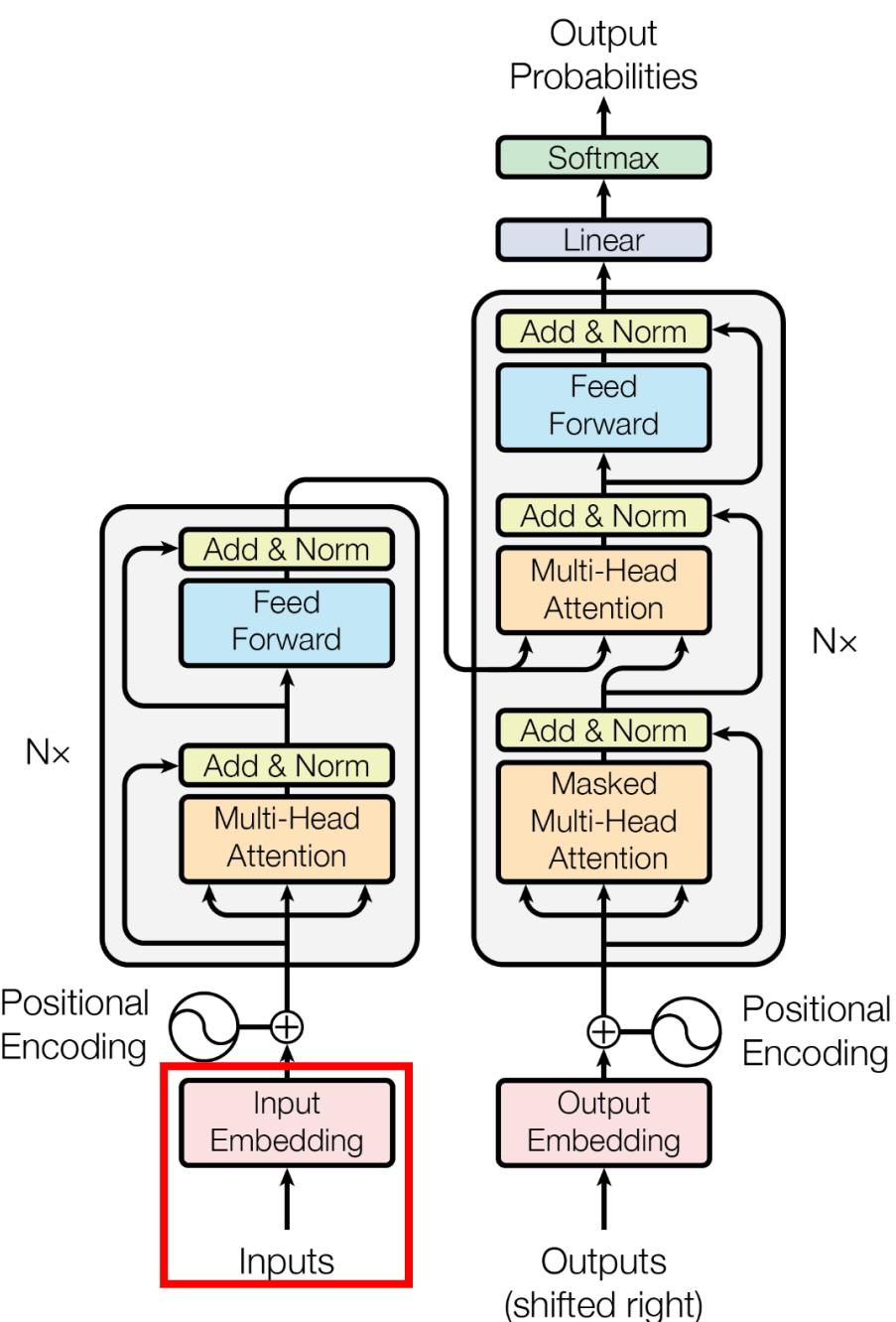


Figure 1: The Transformer - model architecture.

# What about the order?

- In RNNs, the recurrence plays a role in telling us the order of the words in a sentence. In transformers, we lose such recurrence.
- Without order, same set of words = same meaning:
  - {*I, do, not, like, apples, and, you, like, oranges*}
  - {*you, like, apples, and, I, do, not, like, oranges*}
- Need method to encode position of an entity that
  - Outputs a unique encoding for each position
  - Distance between any two positions should be consistent across sentences with different lengths
  - Generalize to longer sentences without any efforts
  - Its values should be bounded

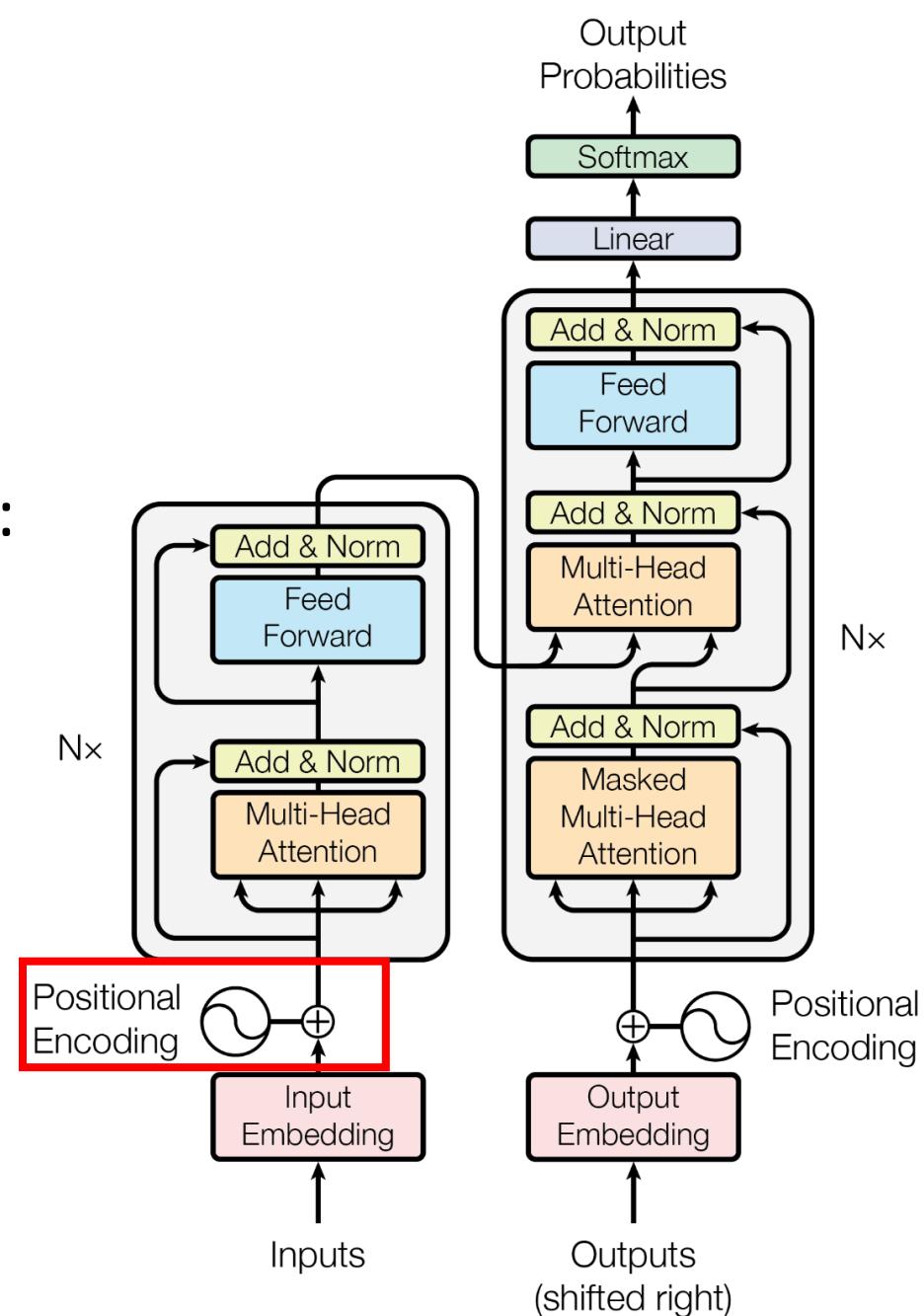
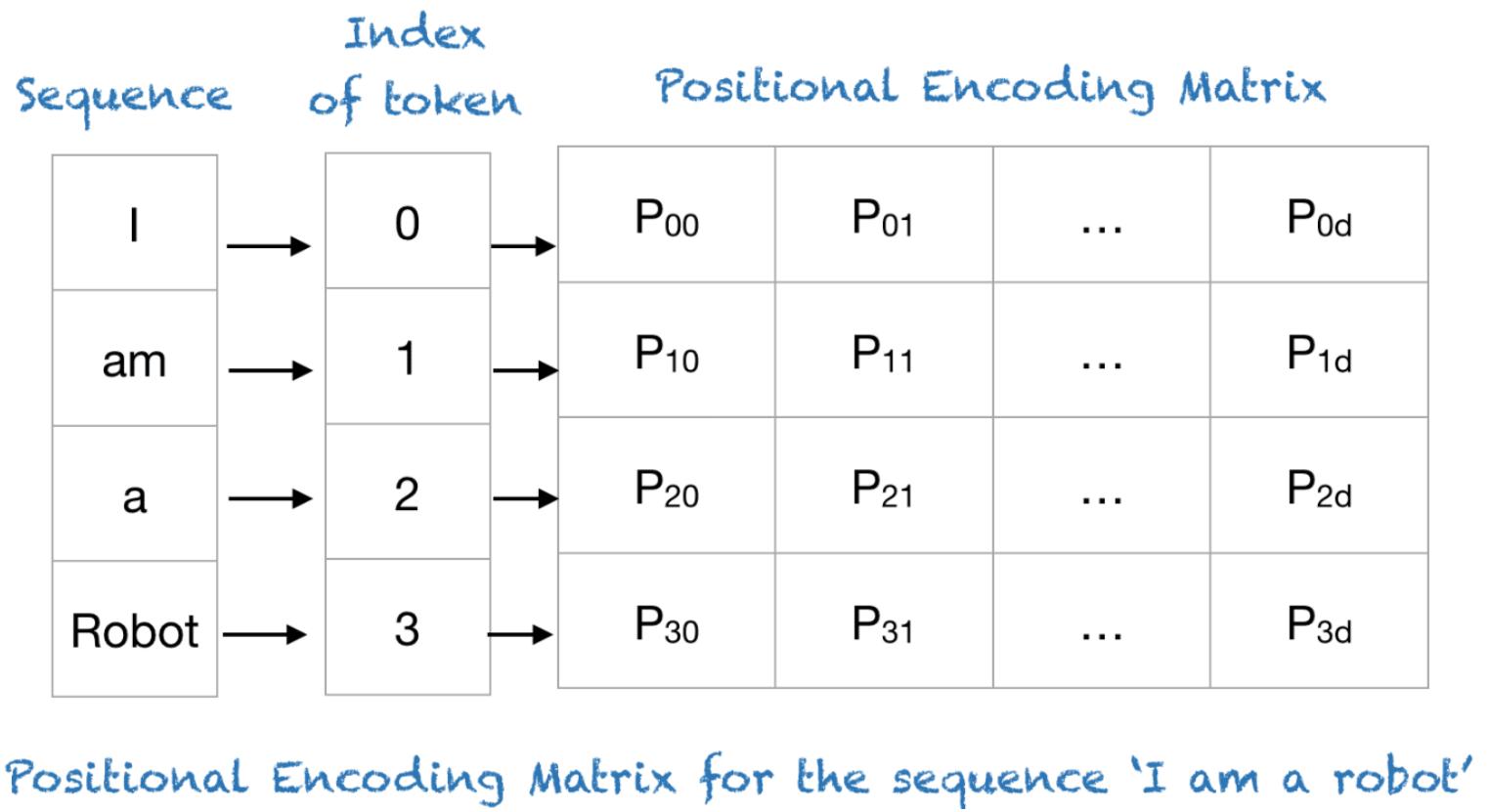


Figure 1: The Transformer - model architecture.

# Positional Encoding: Why vectors instead of indexes?

- Positional encoding describes the location or position of an entity in a sequence
- Each position is assigned a unique representation



- Why not just use the index?
  - For long sequences, the indices can grow large in magnitude.
  - If you normalize the index value to lie between 0 and 1, it can create problems for variable length sequences as they would be normalized differently

# Positional Encoding: Intuition

- Suppose you want to represent it in binary format:
  - The lowest bit alternates with every number
  - The second-lowest bit alternates every two numbers, and higher bits continue this pattern.
- But using binary values would be a waste of space and doesn't satisfy the consistent distance between any two positions requirement.
- Instead, we can use their continuous counterparts: sinusoidal functions.
- By decreasing their frequencies, we replicate the behavior of binary bits:
  - Higher frequencies alternate more rapidly, similar to the lower bits in binary (e.g., red bits).
  - Lower frequencies alternate more slowly, similar to the higher bits in binary (e.g., orange bits).

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

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1, t) \\ \cos(\omega_1, t) \\ \vdots \\ \sin(\omega_d, t) \\ \cos(\omega_d, t) \end{bmatrix}_{d \times 1}$$

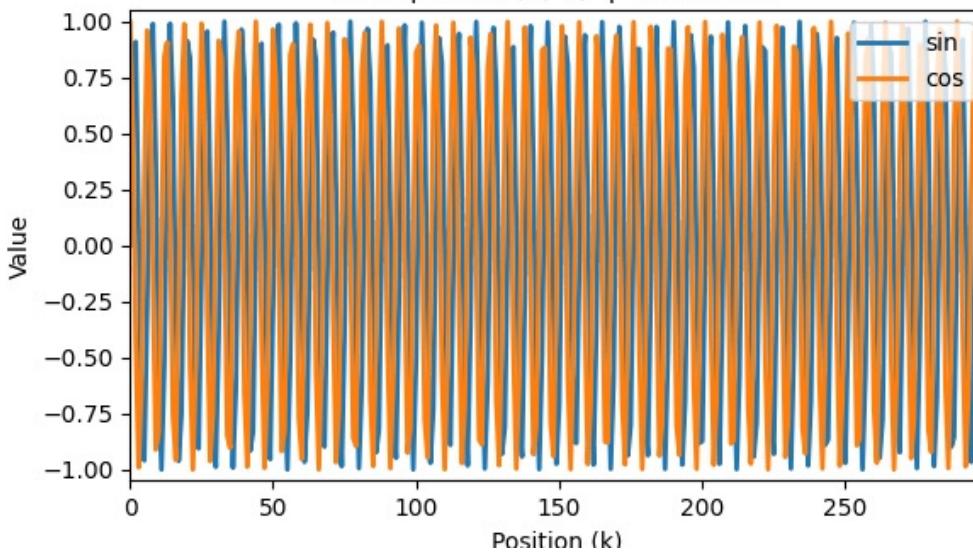
# Positional Encoding

- To convey the ordering information , we use **Positional Embeddings**  $P \in \mathbb{R}^{d \times T}$
- In “Attention is All you Need”, authors used

$$P_{k,2i} = \sin\left(\frac{k}{10000\frac{d}{2i}}\right)$$

$$P_{k,2i+1} = \cos\left(\frac{k}{10000\frac{d}{2i}}\right)$$

i=0 | dims (0, 1) | denom≈1



Sequence	Index of token, $k$	Positional Encoding Matrix with $d=4$ , $n=100$			
		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	1	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	2	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	3	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Positional Encoding Matrix for the sequence 'I am a robot'

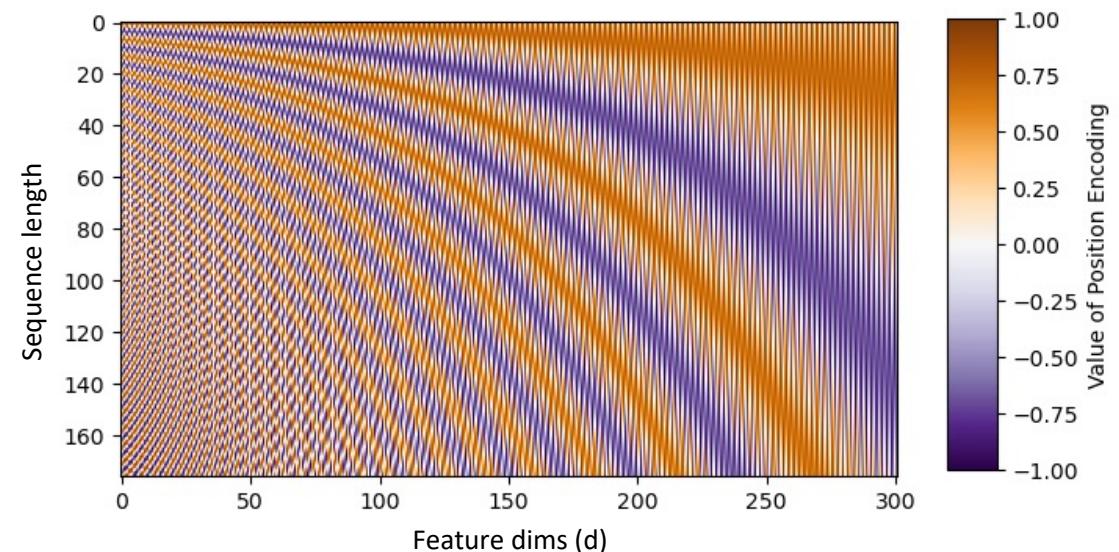
# Positional Encoding

- To convey the ordering information , we use **Positional Embeddings**  $P \in \mathbb{R}^{d \times T}$

- In “Attention is All you Need”, authors used

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

$$P_{k,2i+1} = \cos\left(\frac{k}{10000^{\frac{2i}{d}}}\right)$$



- Let  $\mathbf{x} = [x_1, \dots, x_T] \in \mathbb{R}^{d \times T}$  be the (row) matrix of tokens concatenated together
- The positional embedding gets added to the input directly to the set of tokens:

$$\mathbf{x}^{(0)} = \mathbf{x} + \mathbf{P} \in \mathbb{R}^{d \times T}$$

- We use superscript  $(0)$  to denote the input, zero-th layer

# Encoder Block

- Just like in the Attend & Align model, we have an encoder that turns input embeddings into hidden embeddings
- The main components of an **Encoder Block** is
  - Multi-Head Attention
  - LayerNorms
  - Feedforward Neural Networks
  - Skip Connections
- Let's break down the Multi-Head Attention!

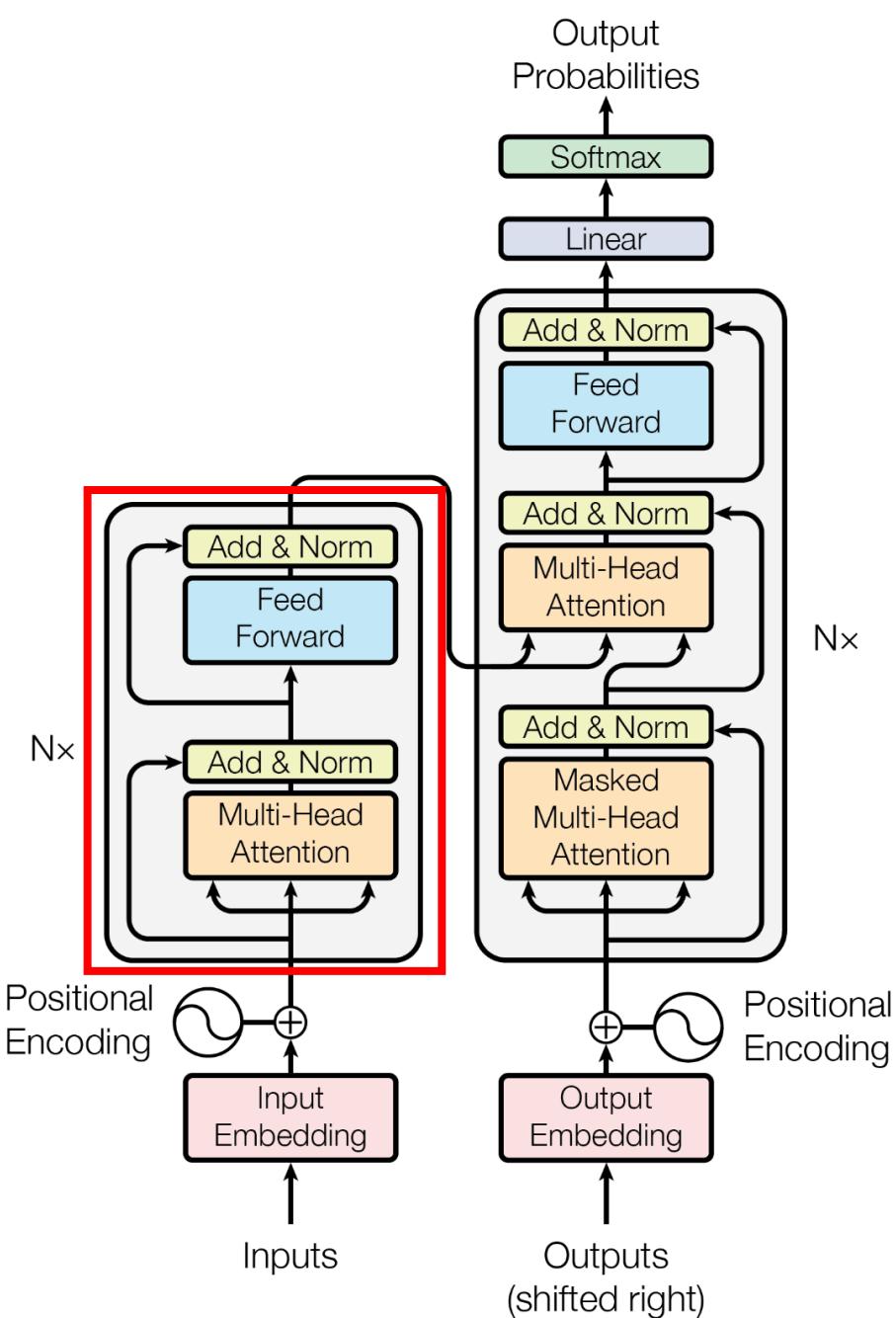
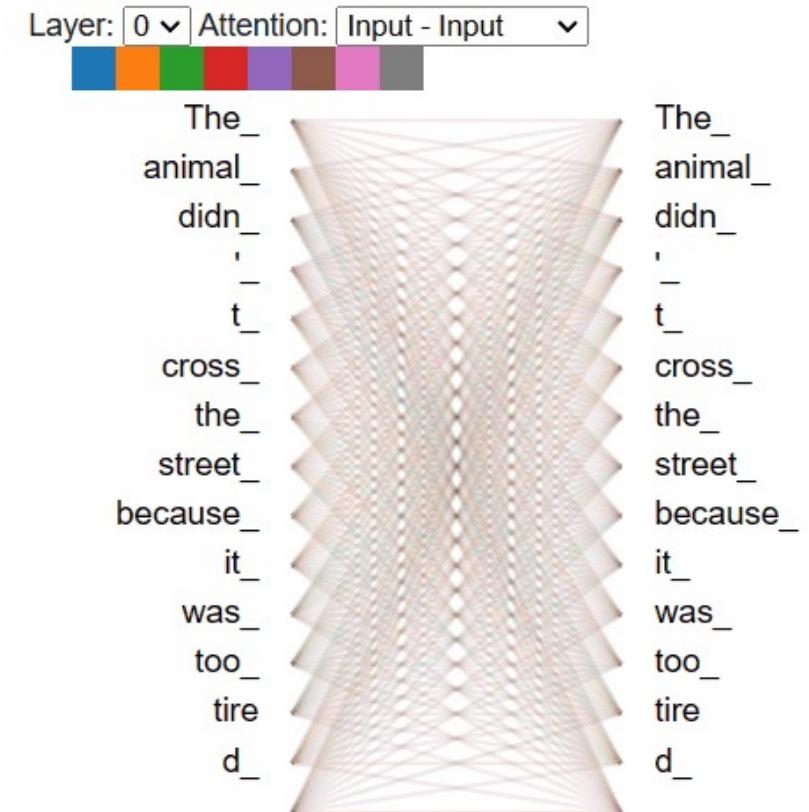


Figure 1: The Transformer - model architecture.

# Self-Attention

- Focuses on important parts of the input by weighing the relevance of each token to the others.
  - What does “**it**” in the sentence “The animal didn't cross the street because **it** was too tired.” refer to?
  - Is it referring to the **street** or to the **animal**?
- Self-attention allows each token to attend to every other token in the sequence, helping the model capture context and relationships between words.
  - When processing "it", the model uses attention to understand that "it" refers to "animal."
- In RNNs, a hidden state carries context from previous tokens, but attention mechanisms allow direct access to all tokens, without relying on a sequential flow.



# Self-Attention

- Given the input embeddings  $\mathbf{x} = [x_1, \dots, x_T]$ , we generate three matrices:
  - Query matrix  $\mathbf{Q}$
  - Key matrix  $\mathbf{K}$
  - Value matrix  $\mathbf{V}$
- Input embeddings are transformed into these matrices by multiplying them by three weight matrices  $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$  that we learn during the training process.
- Analogy for **Query**, **Key**, and **Value**: Library System
  - Imagine you're looking for information on a topic (**query**)
  - Each book has a summary (**key**) to help you identify if it contains relevant information.
  - Once you find a match, you access the book to get the detailed information (**value**) you need.
  - In Attention, we do a "soft match" across multiple books, combine information from each book in proportion to their relevance (e.g., book 1 is most relevant, then book 2, etc.)

$$\begin{array}{c} \mathbf{X} \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} \end{array} \times \begin{array}{c} \mathbf{W}^Q \\ \begin{matrix} \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \end{matrix} \end{array} = \begin{array}{c} \mathbf{Q} \\ \begin{matrix} \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \\ \textcolor{purple}{\square} & \textcolor{purple}{\square} & \textcolor{purple}{\square} \end{matrix} \end{array}$$
$$\begin{array}{c} \mathbf{X} \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} \end{array} \times \begin{array}{c} \mathbf{W}^K \\ \begin{matrix} \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \end{matrix} \end{array} = \begin{array}{c} \mathbf{K} \\ \begin{matrix} \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \\ \textcolor{orange}{\square} & \textcolor{orange}{\square} & \textcolor{orange}{\square} \end{matrix} \end{array}$$
$$\begin{array}{c} \mathbf{X} \\ \begin{matrix} \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \end{matrix} \end{array} \times \begin{array}{c} \mathbf{W}^V \\ \begin{matrix} \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \end{matrix} \end{array} = \begin{array}{c} \mathbf{V} \\ \begin{matrix} \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \end{matrix} \end{array}$$

## Analogy for Query, Key, and Value

how to play the violin **Query (Q)**  Bookmarks (0)

**Everything** Catalog Articles+ Databases Colenda Website

**Articles+**

**View and filter 9,343 results**

**Why I . . . play the violin**

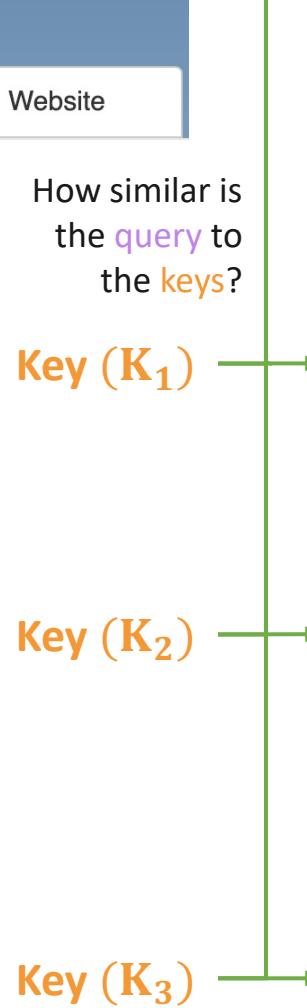
Jones, H  
Published in BMJ (Online). Volume 376, pp. o322-o322.  
Journal Article  
2022  
South London GP Nicola Weaver tells Helen Jones why she plays in an orchestra

**La mariposa ¿La recuerda?: the affective and moral dimensions of professional vision in learning to play the violin**

Johnson, S  
Published in Mind, culture and activity. Volume 30, Issue 3-4, pp. 209-232.  
Journal Article  
2023  
This is a microanalytic study of a metaphorical story, which is co-constructed by a violin teacher and her emergent bilingual student as part of building the child's professional vision within the art form. The findings center on the multi-functionality of the metaphorical story...

**Violin Tutorial I "Violin Master Pro" Teaches People How To Play Violin Like A Master -- Vkoolelite**

Published in PRWeb Newswire.  
Newspaper Article  
2013

How similar is the **query** to the **keys**? 

**Key ( $K_1$ )**  $\alpha_1$

**Key ( $K_2$ )**  $\alpha_2$

**Key ( $K_3$ )**  $\alpha_3$

a "soft match" across multiple articles, combining relevant information in proportion to how relevant it is

**Value ( $V_1$ )**

Check for updates  
London, UK  
Cite this as: BMJ 2022;376:e322  
<https://doi.org/10.1136/bmj.e322>  
Published: 15 February 2022

Why I . . . play the violin  
South London GP Nicola Weaver tells Helen Jones why she plays in an orchestra  
Helen Jones  
Nicola Weaver has always had music in her life. As a teenager, she even briefly flirted with the idea of a rock career, but the sight of the doctor could prove a better option. "In another life, I might have gone that way," she says. "But at the time I didn't see a role model in music that I could identify with."  
Today, as Macmillan GP clinical cancer lead for including string quartets, a jazz big band, a pop covers band, and folk fiddle.

**Value ( $V_2$ )**

MIND, CULTURE, AND ACTIVITY  
2023, VOL. 30, NOS. 3-4, 209-232  
<https://doi.org/10.1080/10749039.2023.2300140>  
Routledge Taylor & Francis Group  
Check for updates

La mariposa ¿La recuerda?: the affective and moral dimensions of professional vision in learning to play the violin  
Sarah Jean Johnson  
Department of Teacher Education, University of Texas, El Paso, TX, USA

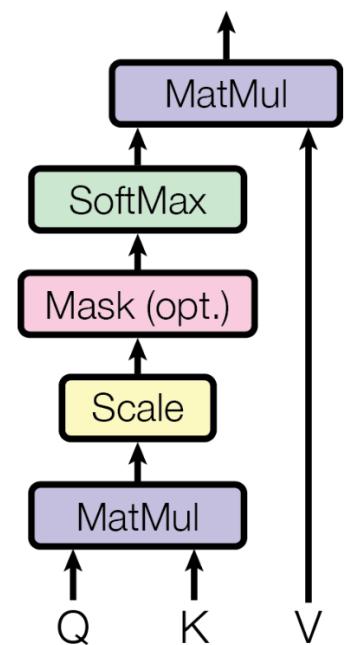
**Value ( $V_3$ )**

Violin Tutorial / "Violin Master Pro" Teaches People How To Play Violin Like A Master -- Vkoolelite  
Full Text:  
Seattle, Wa (PRWEB) October 10, 2013  
Violin Master Pro is a new music program that provides people with a series of basic violin lessons for beginners, simple exercises, and step-by-step instructions on how to become a professional violinist. This program is designed by Eric Lewis, a world renowned violinist who has over 40 years of experience in teaching people how to master their violin easily. Since Eric Lewis released the "Violin Master Pro" program, a lot of clients have used it for learning how to play solos, sonatas, and concertos effortlessly. As a

# Self-Attention

- Calculate the attention score by taking the dot product of  $\mathbf{Q}$  and  $\mathbf{K}^T$ .
- Divide the scores by  $\sqrt{d_k}$ , where  $d_k$  is the dimension of the hidden embedding, to ensure the variance of the dot product does not grow with  $d_k$ , leading to unstable attention mechanism.
- Apply the softmax function to the scaled scores, turning them into probabilities.
- Multiply softmax scores by  $\mathbf{V}$  to obtain the final attention output.
- The self-attention, thus, is defined as:

$$\text{SA}(Q, K, V) = \text{softmax}\left( \frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$



- The term “self” comes from the fact that  $Q, K, V$  are all derived from the same input sequence  $\mathbf{x} = [x_1, \dots, x_T]$

# Multi-Head Self-Attention (MSA)

- **Multi-head Self Attention (MSA)** extends Self-Attention by introducing multiple independent attention heads, each focusing on different types of relationships.

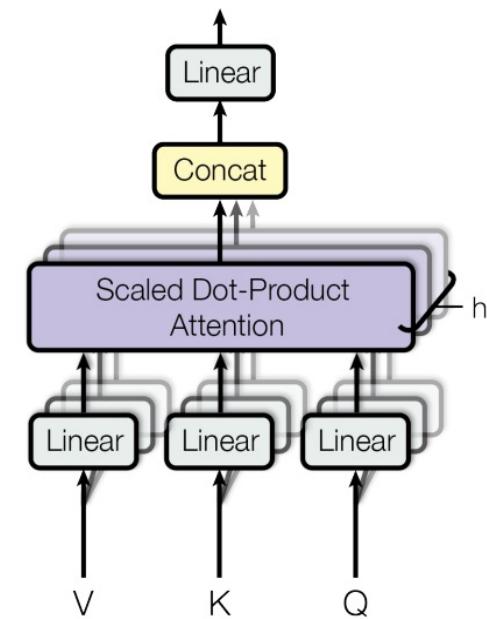


- Each head has its own set of weight matrices:

$$\text{MSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{SA}(\mathbf{Q}_1, \mathbf{K}_1, \mathbf{V}_1), \dots, \text{SA}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)]\mathbf{W}_O$$

$$\mathbf{Q}_i = \mathbf{W}_i^Q \mathbf{x} \quad \mathbf{K}_i = \mathbf{W}_i^K \mathbf{x} \quad \mathbf{V}_i = \mathbf{W}_i^V \mathbf{x}$$

where  $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d \times d_h}$  are weight matrices for the query, key, and value of each head  $i = 1, \dots, h$ , and  $\mathbf{W}_O \in \mathbb{R}^{(h \cdot d_v) \times d}$  is the weighting matrix for fusing all attention heads.



# Residual Connection & Layer Normalization

- **Residual Connection:** combines the input with the output of a sub-layer (either self-attention or feed forward).

- It allows the gradients to flow through the network directly, bypassing non-linear transformations.

$$Output = LN(x + SubLayer(x))$$

- **LayerNorm** normalizes the input tokens across the  $d$  feature dimensions.

- For each token  $x_i$ :

$$\mu_i = \frac{1}{d} \sum_{k=1}^d x_{i,k}, \quad \sigma_i^2 = \frac{1}{d} \sum_{k=1}^d (x_{i,k} - \mu)^2$$

$$LN(x_i) = \gamma \cdot \frac{x_i - \mu_i}{\sqrt{\sigma_i}} + \beta, \quad \gamma, \beta \in \mathbb{R}^d \text{ (learned for each layer)}$$

- This ensures consistent scaling across layers, leading to more stable training.

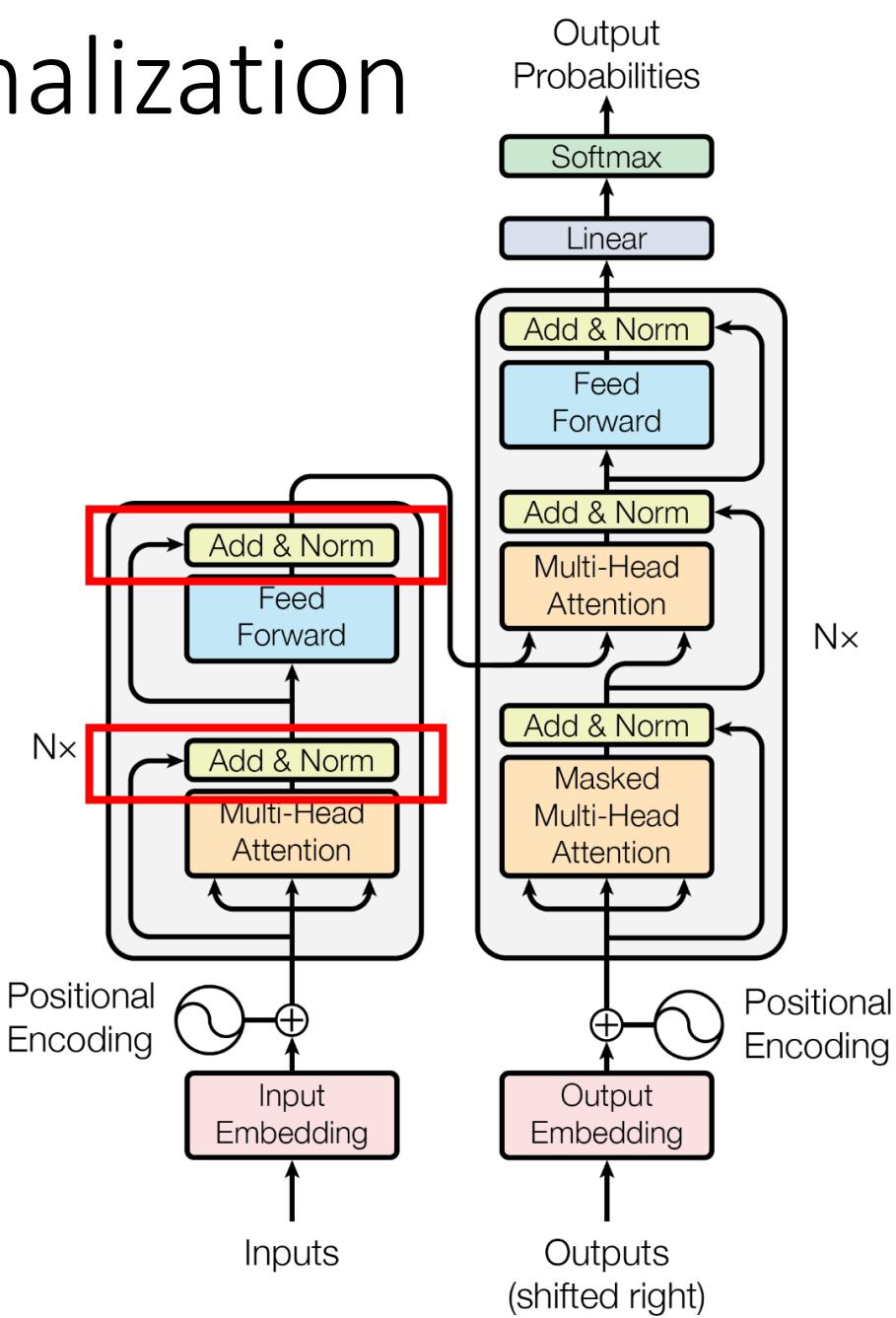


Figure 1: The Transformer - model architecture.

# Encoder Block Summarized

- Putting everything together mathematically, the encoder block can be described by

$$\hat{\mathbf{x}}^{(l)} = \text{LN}(\text{MSA}(\mathbf{x}^{(l-1)}, \mathbf{x}^{(l-1)}, \mathbf{x}^{(l-1)}) + \mathbf{x}^{(l-1)})$$

$$\mathbf{x}^{(l)} = \text{LN}(\text{FFN}(\hat{\mathbf{x}}^{(l)}) + \hat{\mathbf{x}}^{(l)})$$

where FFN is a feed forward neural network and LN denotes Layer Norm

- Note that the input and output dimension of the encoder block is the same:  $\mathbb{R}^{T \times d}$
- We can stack encoder blocks together to make it *deeper*
- The output is like the input: a collection of tokens, but **in context with other tokens**

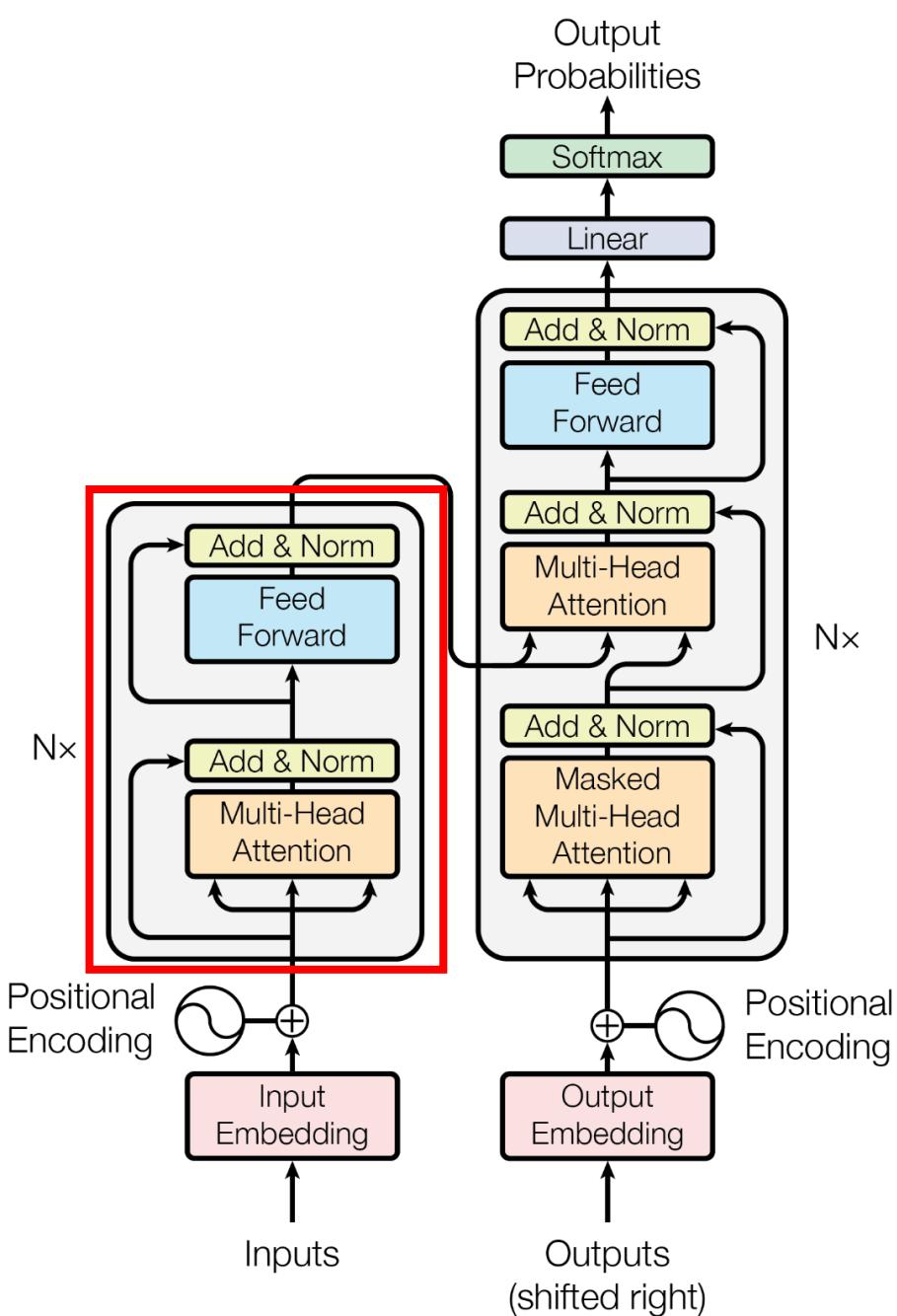
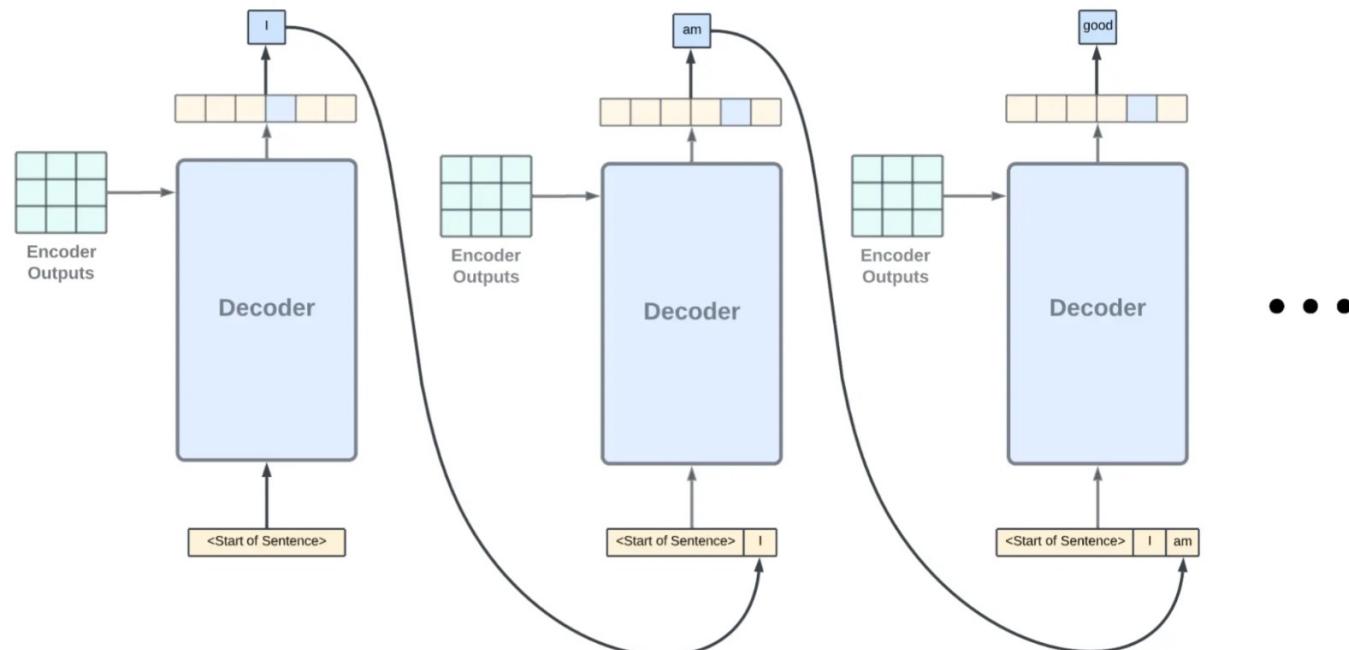


Figure 1: The Transformer - model architecture.

# Decoder Block

- Now, we are going to switch gears into the decoder blocks
- At a high level,
  - During inference, the decoder will take in a <BOS> (beginning of sentence) token as input, and recursively predict the next word until the <EOS> (end of sentence) token is predicted
  - Just like our previous methods for machine translation, the decoder should take in *context* from the encoder to predict what the next token should be



# Decoder Block: Attention Layers

- What is the input?
  - Token embeddings (shifted right): the decoder sees previous target only.
  - Positional encoding: same as the encoder.
- In the Encoder, each block consists of only *one* Multi-Head Self-Attention layer.
- In the Decoder, each block consists *two layers*:
  - The first one is a Masked Multi-Head Self-Attention with tokens from input.
    - *Allows each token to attend to previous ones in the sequence.*
  - But, what does the “Masked” in Masked Multi-Head Attention mean?

$$\hat{\mathbf{y}}^{(l)} = \text{LN}(\text{MaskedMSA}(\mathbf{y}^{(l-1)}, \mathbf{y}^{(l-1)}, \mathbf{y}^{(l-1)}) + \mathbf{y}^{(l-1)})$$

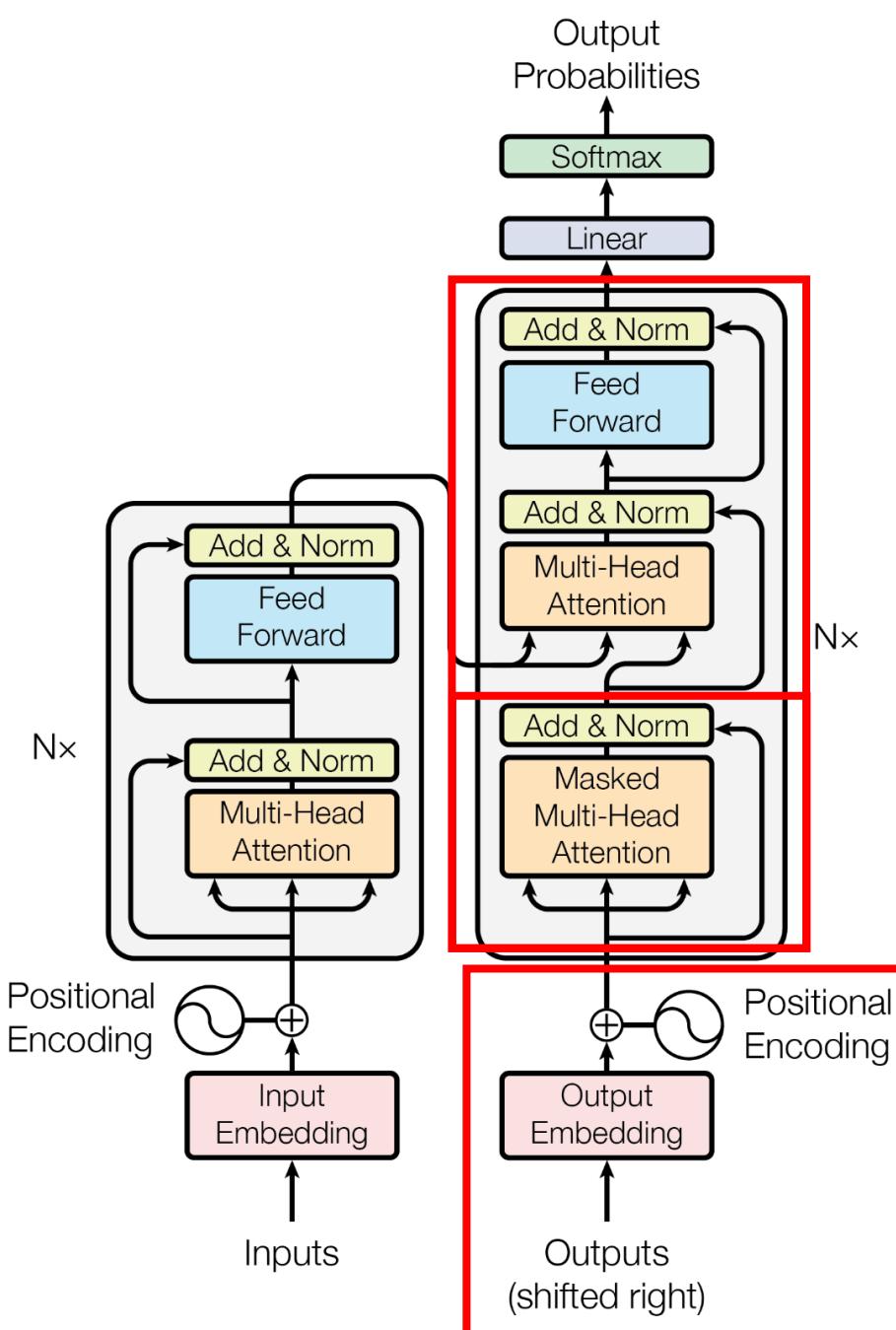


Figure 1: The Transformer - model architecture.

# Decoder Block: Masked MSA

- Just like MSA, **MaskedMSA** calculates attention scores using a scaled dot-product of query and key vectors and normalizes these scores with a *softmax* function to obtain attention weights.
- During training, **MaskedMSA** applies masks on the attention matrices. This is important to preserve the autoregressive property, where each token is predicted based on the preceding tokens only.

$$\text{MaskedMSA} = \text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} + M \right) V$$

M is a mask that applies  $-\infty$  to all **future positions** so they don't contribute to the *softmax* output.

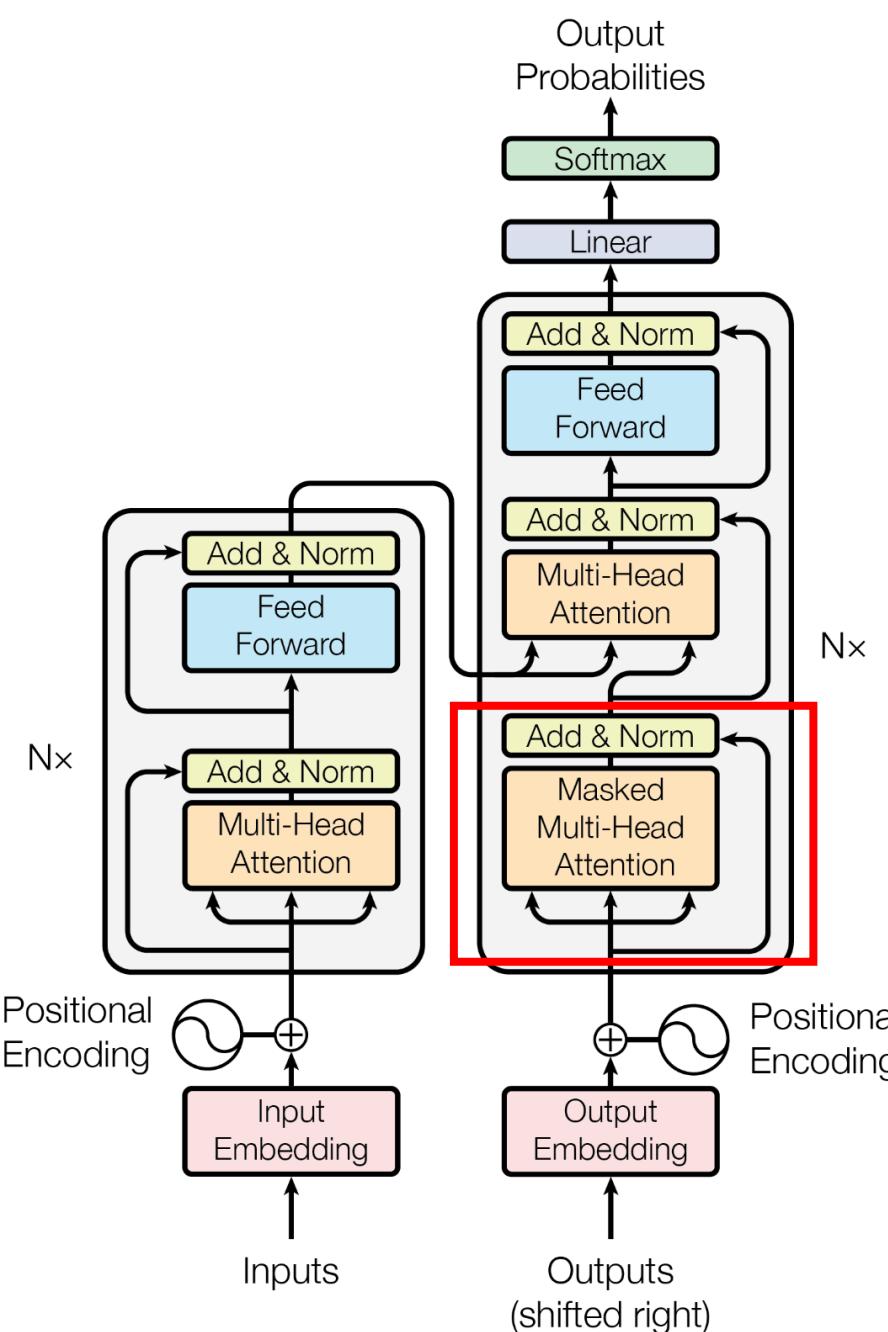


Figure 1: The Transformer - model architecture.

# Decoder Block: Attention Layers

- In the Encoder, each block consists of only *one* Multi-Head Self-Attention (MSA) layer.
  - In the Decoder, each block consists *two layers*:
    - The Second one is a Multi-Head Cross Attention (MCA).
    - MCA applies the same mechanism as MSA in the context where the *queries*, *keys*, and *values* might come from different sources.
    - In this case, *key* and *value* matrices come from the output of the encoder, and *query* matrix from the previous MSA.
      - Allows the decoder to focus on relevant part of encoded input
- $$\tilde{\mathbf{y}}^{(l)} = \text{LN}(\text{MCA}(\hat{\mathbf{y}}^{(l-1)}, \mathbf{x}^{(N)}, \mathbf{x}^{(N)}) + \hat{\mathbf{y}}^{(l-1)})$$
- $\mathbf{x}^{(N)}$  is the output of the encoder (composed of  $N$  encoder layers)

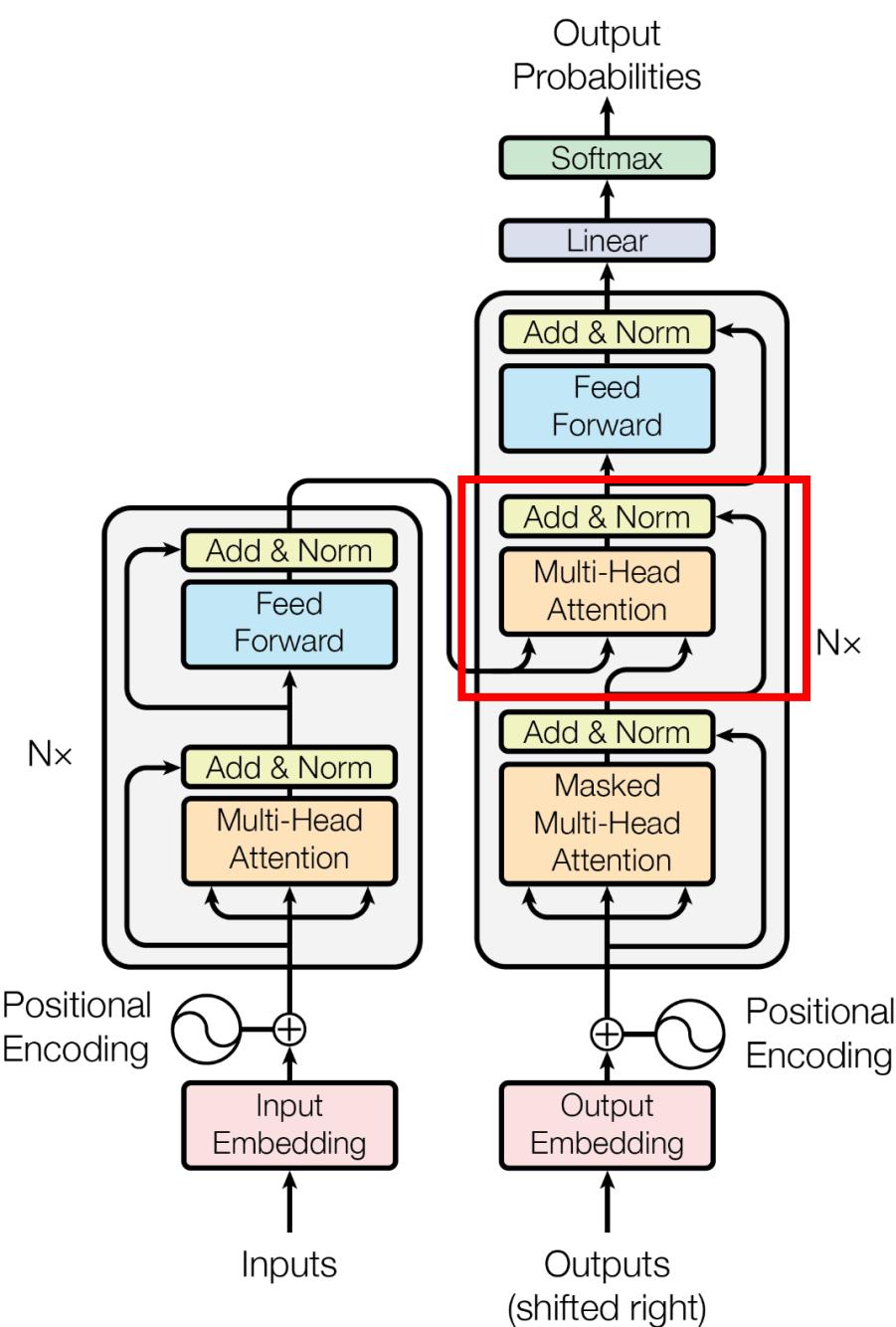


Figure 1: The Transformer - model architecture.

# Decoder Block: Summarized

- Summarizing a forward pass of the Decoder Block, along with Layer Norms and Feedforward Networks like the Encoder:

$$\hat{\mathbf{y}}^{(l)} = \text{LN}(\text{MaskedMSA}(\mathbf{y}^{(l-1)}, \mathbf{y}^{(l-1)}, \mathbf{y}^{(l-1)}) + \mathbf{y}^{(l-1)})$$

$$\tilde{\mathbf{y}}^{(l)} = \text{LN}(\text{MCA}(\hat{\mathbf{y}}^{(l-1)}, \mathbf{x}^{(N)}, \mathbf{x}^{(N)}) + \hat{\mathbf{y}}^{(l-1)})$$

$$\mathbf{y}^{(l)} = \text{LN}(\text{FFN}(\tilde{\mathbf{y}}^{(l)}) + \tilde{\mathbf{y}}^{(l)})$$

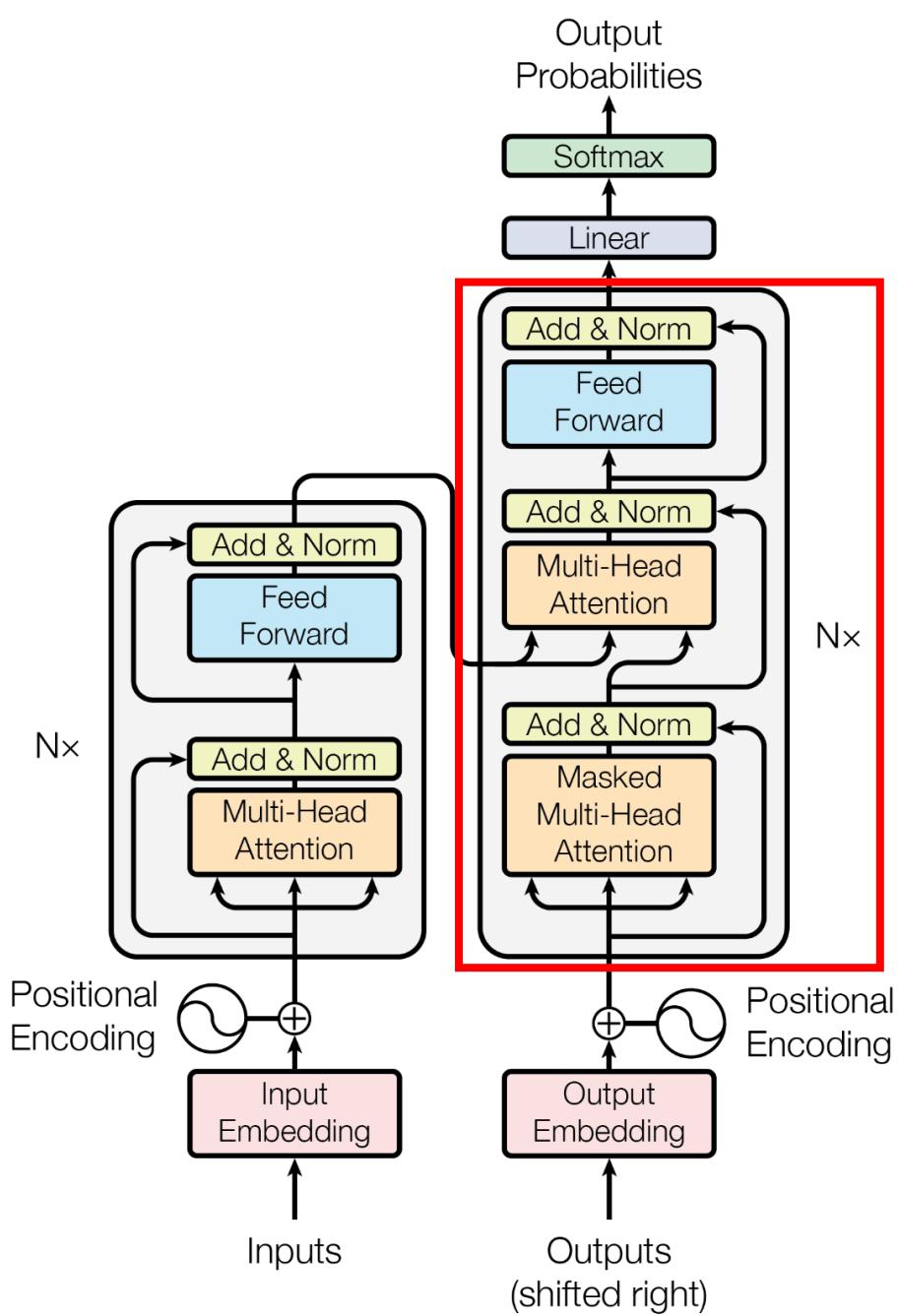


Figure 1: The Transformer - model architecture.

# Training the Transformer

- Training the Transformer follows the same intuition with other Seq2Seq models.
- Autoregressive Sequence Modeling:
  - The decoder uses masked self-attention so that each token predicts the next word without looking at future tokens.
  - The model defines the conditional probability of the target sequence  $\mathbf{y}$  given the source  $\mathbf{x}$  as follows:

$$P_{\theta}(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^T P_{\theta}(y_t|y_{<t}, \mathbf{x})$$

$P_{\theta}(y_t|y_{<t}, \mathbf{x})$  corresponds to the decoder's *softmax* output for the next token.

- The model is trained to minimize the cross-entropy between the predicted distribution and the ground truth.

$$\mathcal{L} = - \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log P_{\theta}(\mathbf{y}|\mathbf{x})$$

This is  $P_{\theta}(y_4|y_{<4}, \mathbf{x})$

For example, when translating "Soy un estudiante" to "I am a student"

## Target Model Outputs

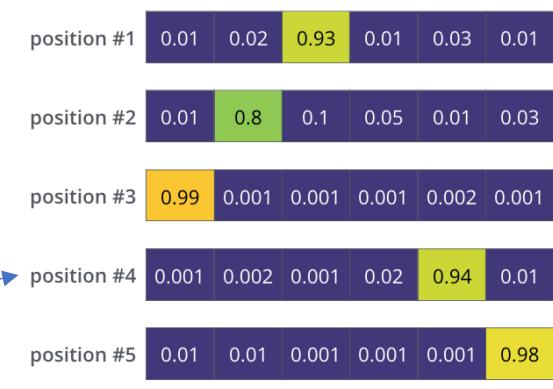
Output Vocabulary: a am I thanks student <eos>



a am I thanks student <eos>

## Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>

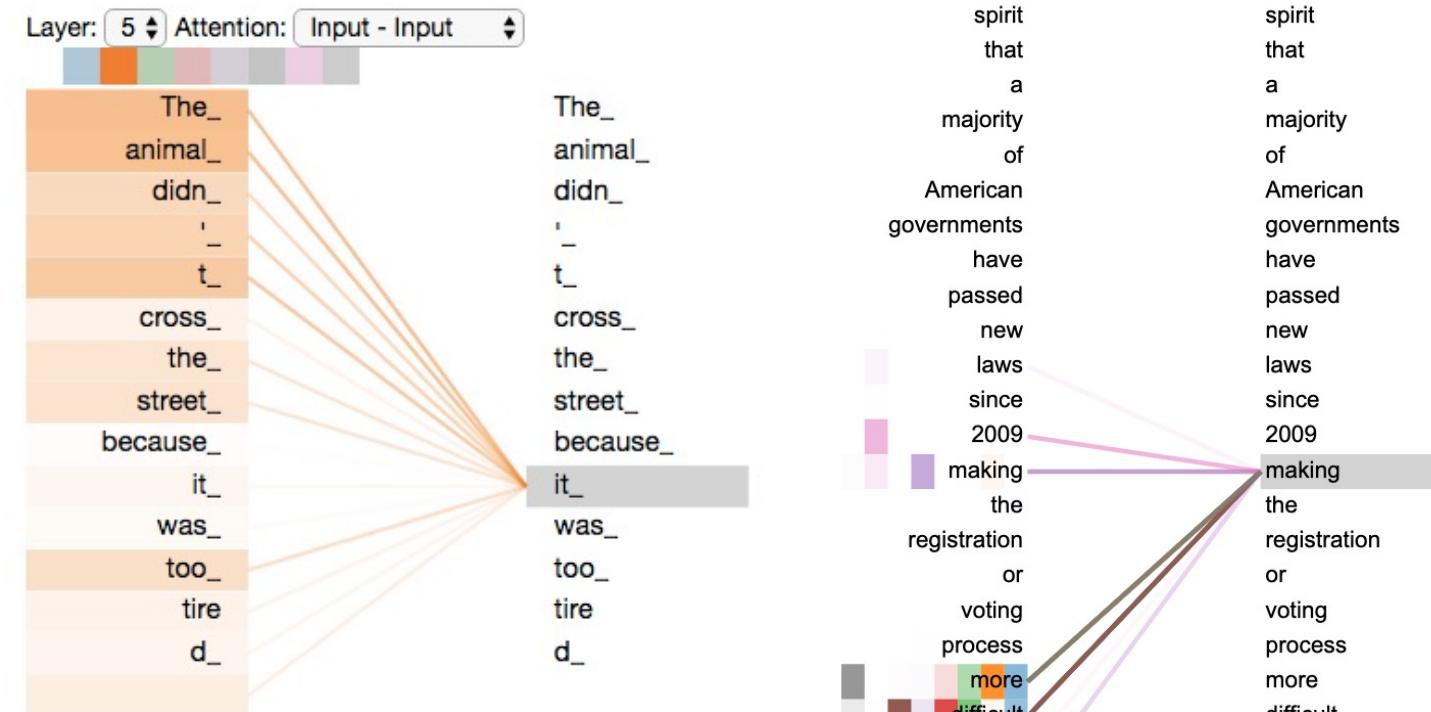


a am I thanks student <eos>

# Attention Visualization: Long distance dependency

- Earlier we saw the sentence: “The animal didn't cross the street because **it** was too tired.”

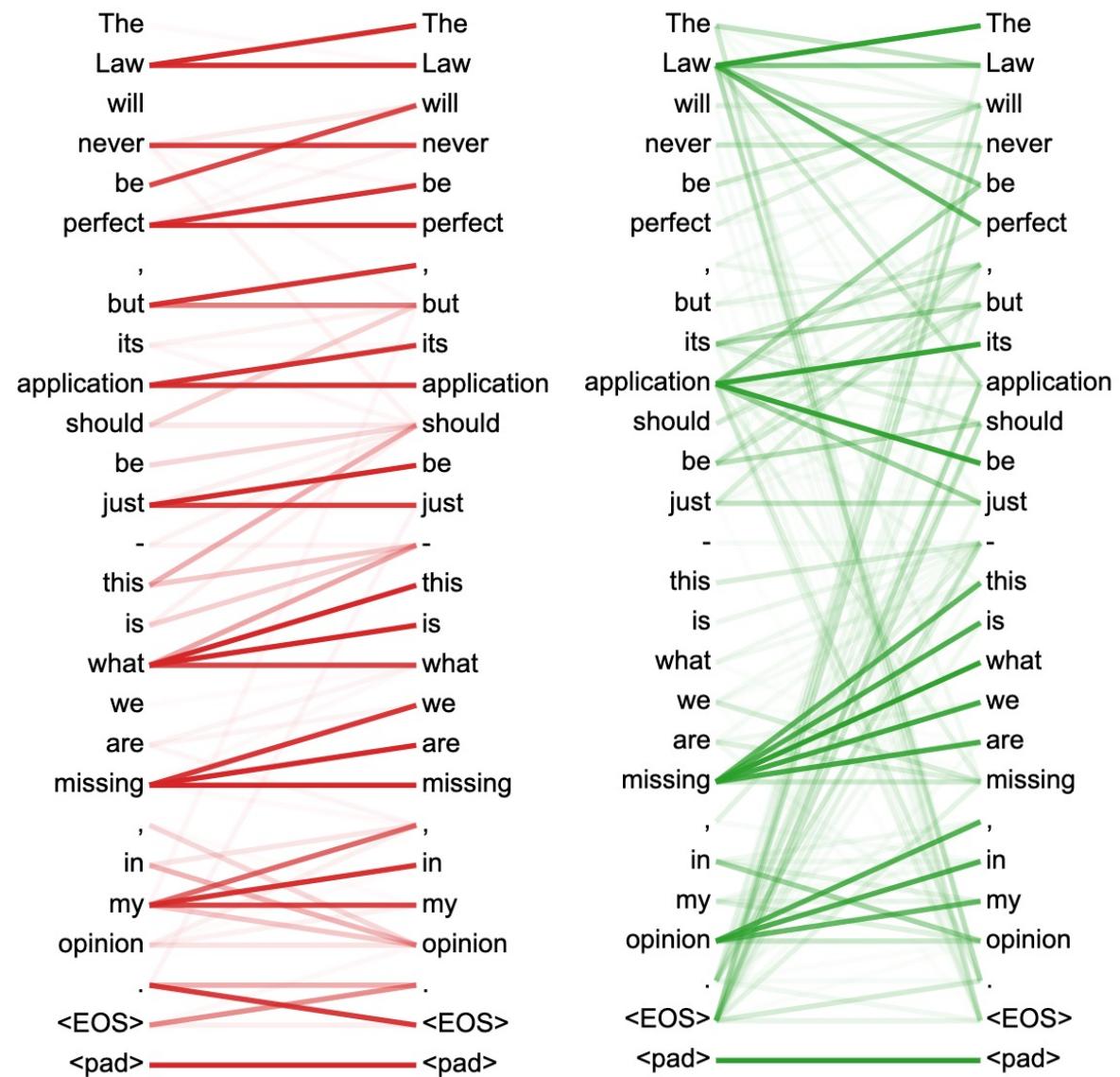
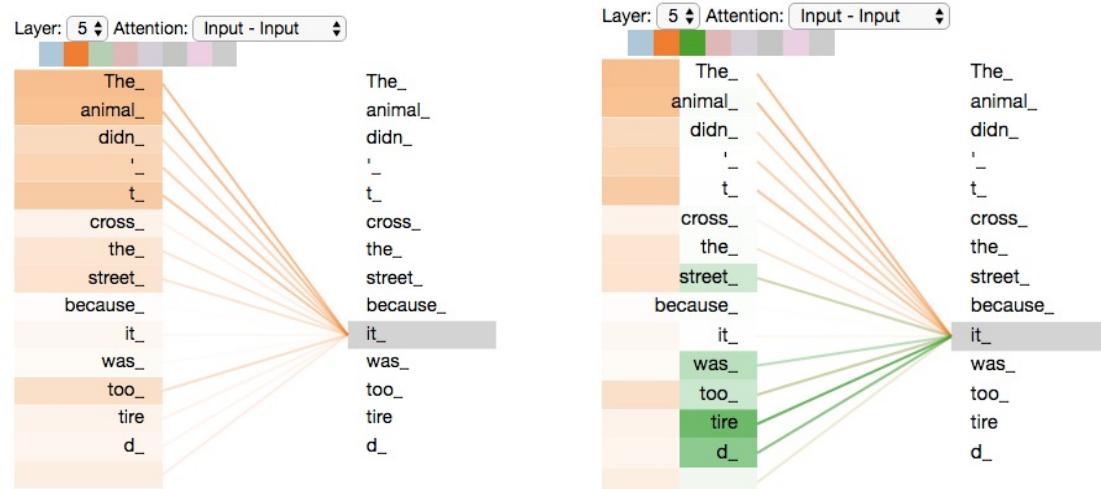
- What does “**it**” in this sentence refer to? The visualization of self-attention shows the association of “**it**” with beginning parts like “The animal”.



- On the right we see another visualization showing how different words in a longer sentence relate to each other.
- Check out this interactive [visualization](#).

# Attention: Attention from Different Heads

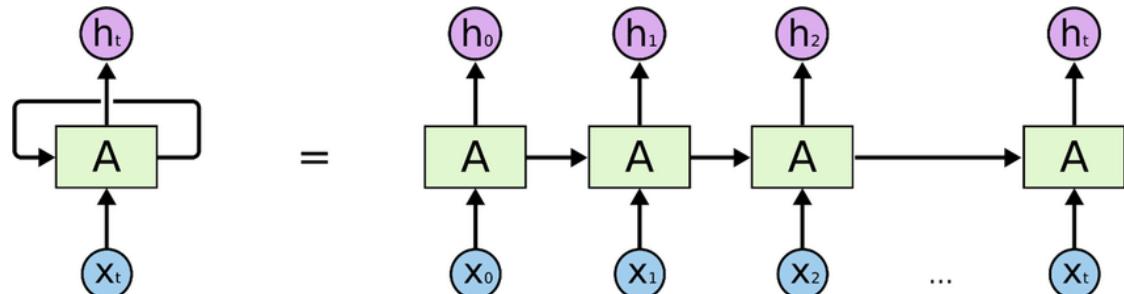
- Attention heads can specialize to capture various dependencies, such as syntactic and semantic relationships.
- This allows the model to attend to different types of causalities between words in a sentence.



# RNNs vs. Transformers

## Recurrent Neural Network

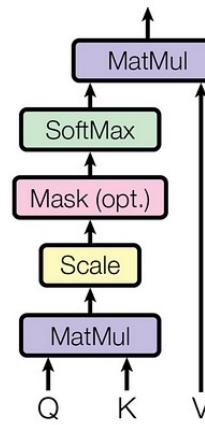
- Handle sequential data
- Learn sequential dependencies
- Each time step depends on the previous one



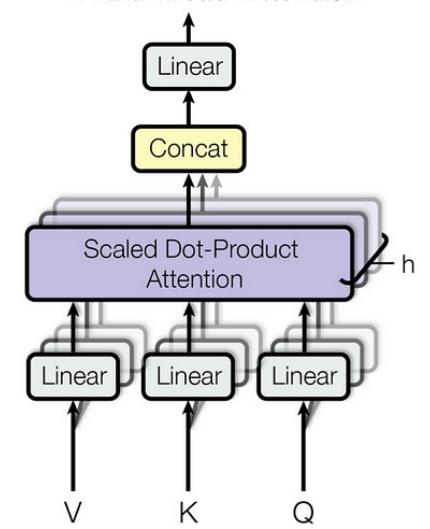
## Transformers

- Handle sequential data
- Learn sequential dependencies
- Use self-attention to capture global context

Scaled Dot-Product Attention



Multi-Head Attention



# RNNs vs. Transformers

## Recurrent Neural Network

- (-) Learning long-range dependences is challenging due to recurrent structure
  - Can be aided by specialized architectures like LSTM and GRU
  - Suffer from training issues such as vanishing gradient
- (-) Hard to scale up because each time step depends on the previous one
- (+) Usually smaller number of parameters, does not require lots of data to train

## Transformers

- (+) Attention mechanism better captures long-range dependences
  - Able to handle both global context and local context
  - No vanishing gradient issues
- (+) Processes tokens in parallel, makes it efficient for training on GPUs
- (-) Usually large number of parameters, requires lots of data to train

# Next Two Lectures: Iterations of Transformers

## Natural Language Processing

- **BERT (Bidirectional Encoder Representations from Transformers)**
- GPT (Generative Pre-trained Transformer)
- RoBERTa (Robustly Optimized Bert Pre-training)
- T5 (Text-to-Text Transfer Transformer)

## Computer Vision

- **ImageGPT**
- **Vision Transformer**
- Swin Transformer, Pyramid Vision Transformer