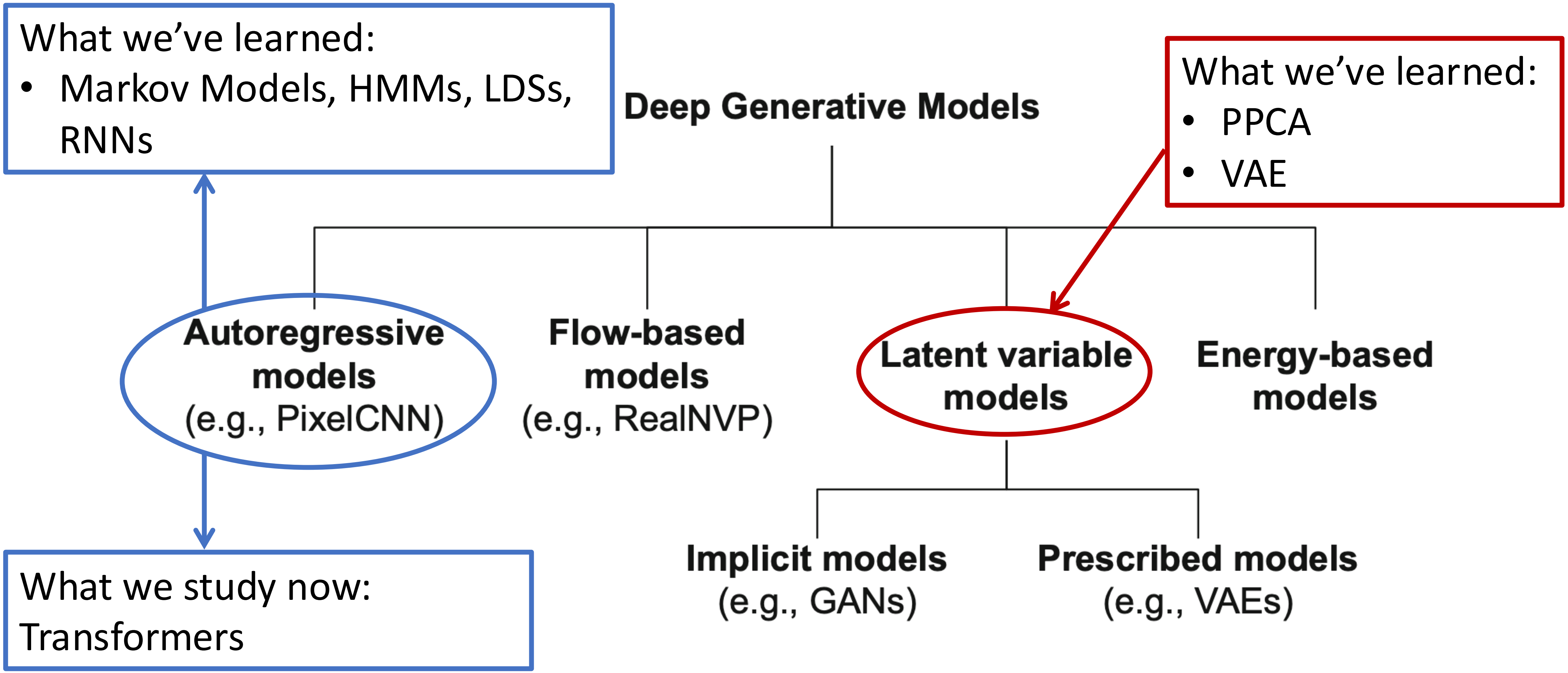
Deep Generative Models: Transformers

Fall Semester 2024

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



Autoregressive Models

- Many kinds of models
  - Markov Chains
  - Hidden Markov Models
  - Markov Random Fields
  - Linear Dynamical Systems
  - Recurrent Neural Networks
  - **Transformers**

- Last lecture
  - **Model**: Introduced the vanilla RNN architecture
  - **Inference**: Unfolding
  - **Training**: Backpropagation Through Time
  - **Variants of RNNs**: LSTMs, GRUs
  - **Seq2Seq**: Machine Translation, Image Captioning
  - **Attention Mechanism**: Soft and Hard Attention

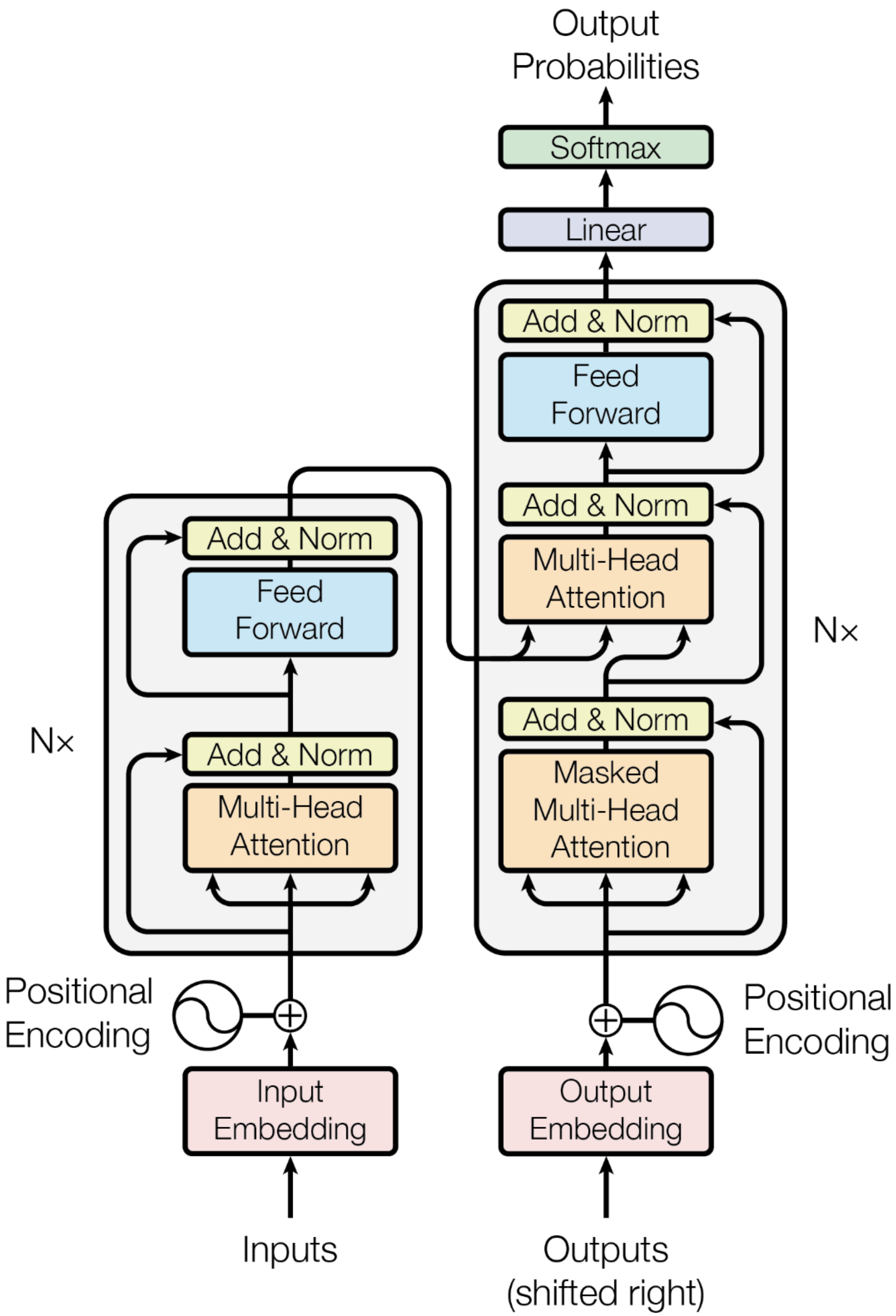


Figure 1: The Transformer - model architecture.

Last Lecture: Why RNNs fall short?

- **Hard to capture long-term dependencies**
  - Require modification to architectures
- **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.

Lc s!

1990s — **Recurrent Neural Networks**

RNNs, GRUs, LSTMs

2014 — **Simple attention mechanisms**

2017 — **Beginning of transformers**

Attention is all you need

2018 — **Explosion of transformers in NLP**

BERT, GPT-3

2018-2020 — **Explosion into other fields**

Explosion into other fields: ViTs, Alphafold-2.

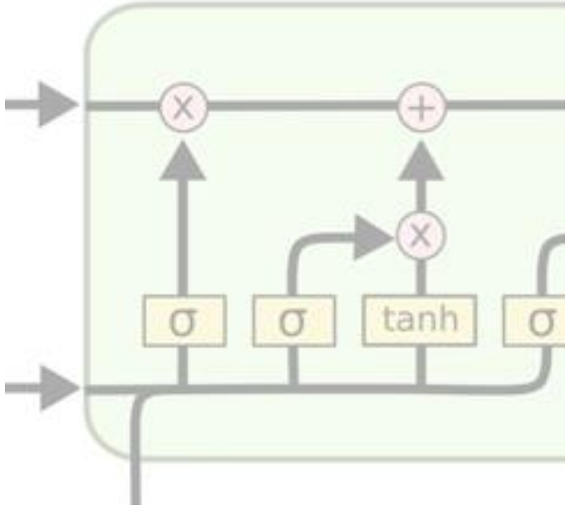
2021-2022 — **Start of Generative Era**

Codex, Decision Transformers, GPT-X, DALL·E

2024 — **Present Day**

Huge models, more applications: Chat-GPT, GPT-4, Gemini, Llama and open-source LLMs, Whisper, Robotics Transformer, Stable Diffusion, Sora, and so much more...!

Future (?!)

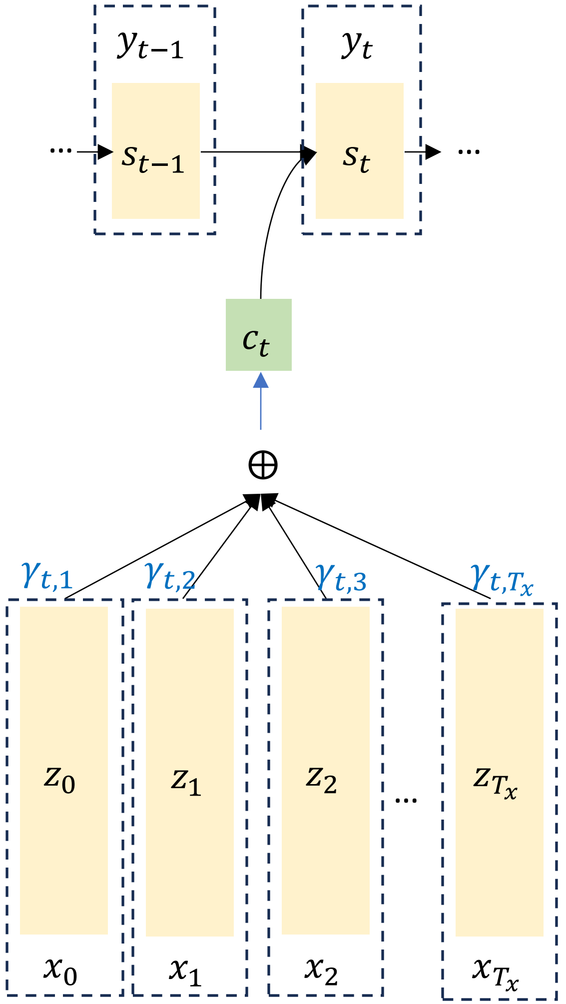


Recall the Translate and Align Model in RNNs

- Decoder: **context vector** c_t is computed as a weighted sum of the hidden states z_j:

$$c_t = \sum_{j=1}^{T_x} \gamma_{tj} z_j \qquad \gamma_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})} \qquad e_{tj} = a(s_{t-1}, z_j)$$

Context vector Weights of hidden states Alignment model

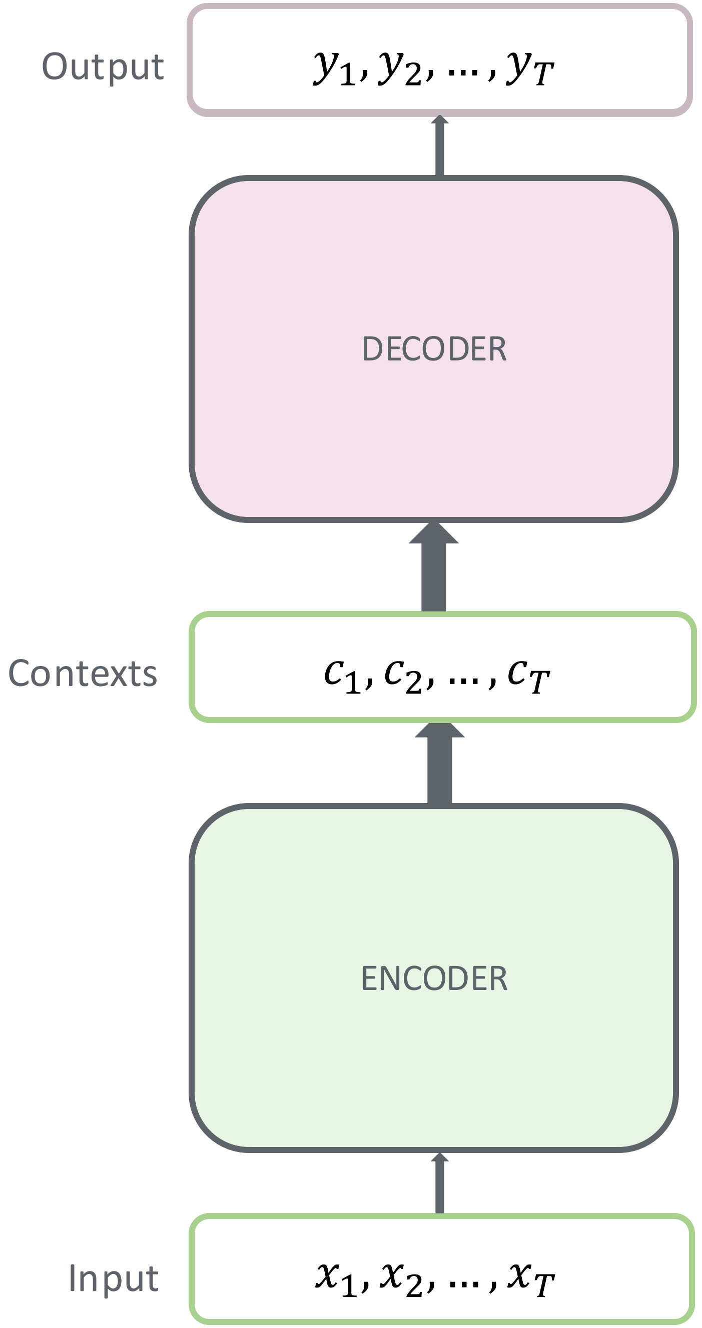


- Here:
  - a is called the **Alignment model**
    - Computes how well the inputs around position j and the output at position t match
    - Typically chosen to be a feedforward neural network
  - γ_{tj} is the probability that the target word y_t is aligned to, or translated from, a source word x_j.
  - c_t is the expectation of the hidden state w.r.t. the distribution γ_{tj}.

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

- Let's recap our setting: Machine Translation
  - We are given a sentence, a sequence of *tokens* (words) as input, represented by $x = (x_1, \dots, x_T)$. We want to build an architecture that takes a sentence as input and produces a translated target sentence $y = (y_1, \dots, y_T)$ as output.



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

- Let's recap our setting: Machine Translation
  - We are given a sentence, a sequence of *tokens* (words) as input, represented by $x = (x_1, \ldots, x_T)$. We want to build an architecture that takes a sentence as input and produces a translated target sentence $y = (y_1, \ldots, y_T)$ as output.

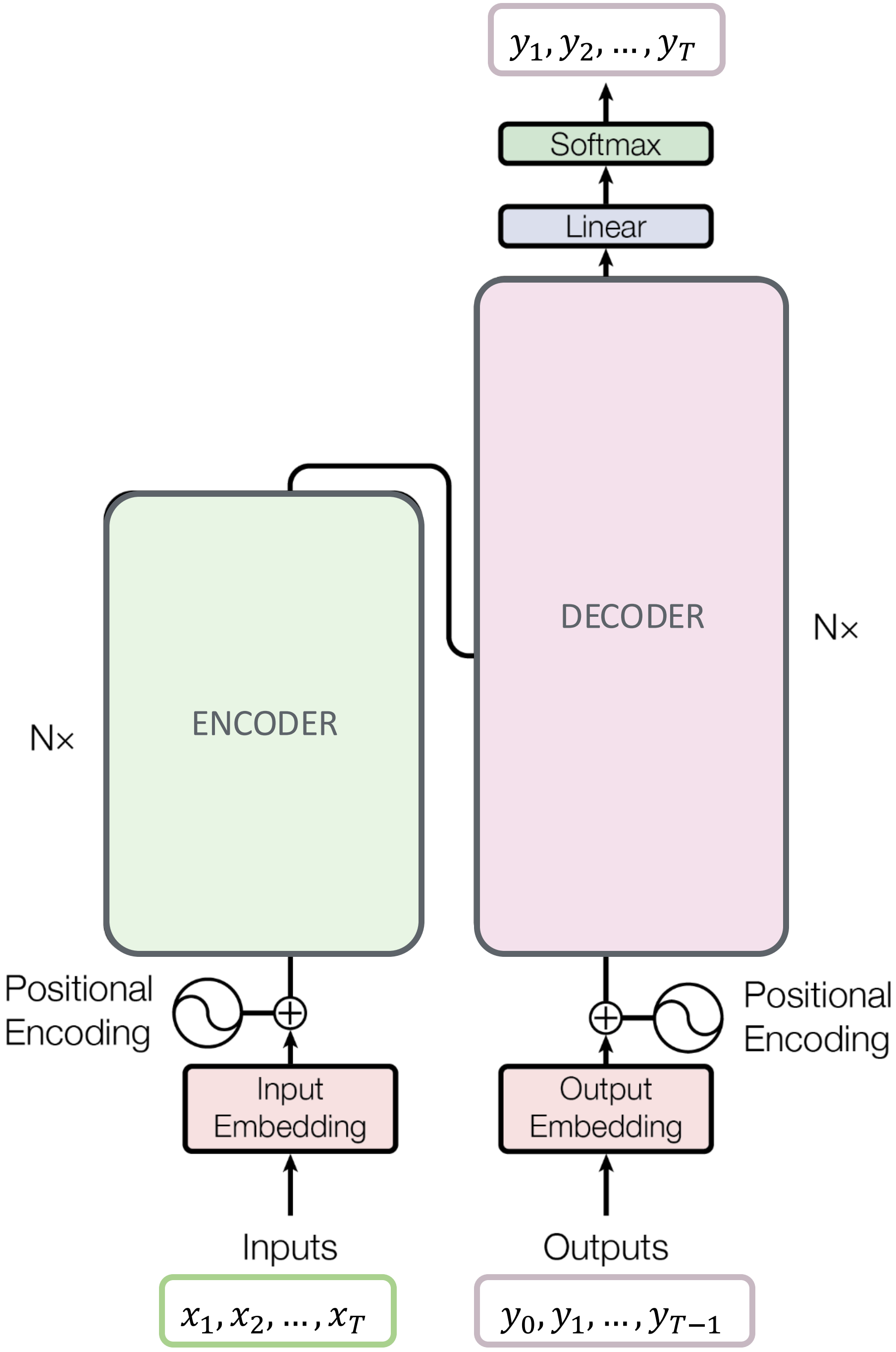


Figure 1: The Transformer - model architecture.

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
$$x_t = E\tilde{x}_t$$
- Again $E \in \mathbb{R}^{d \times N}$ 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.*

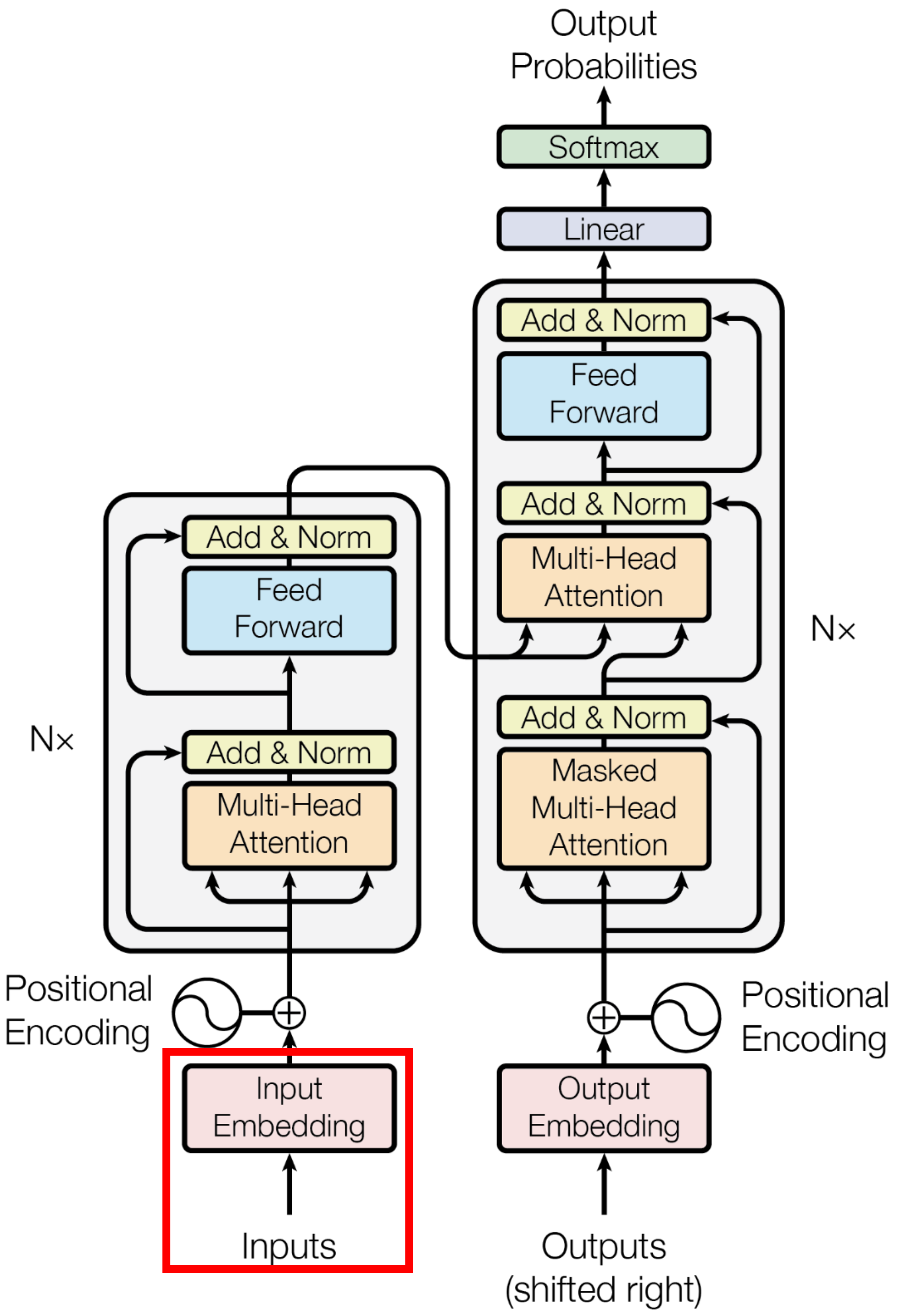


Figure 1: The Transformer - model architecture.

What about the order?

- In RNNs, the recurrence plays a role in telling us the order of the words in a sentence. But now, we won't have that, since we lose the recurrence
- Simple example:
  - {*I, do, not, like, apples, and, you, like, oranges*} and {*you, like, apples, and, I, do, not, like, oranges*}
  - Since they contain the same words, they are actually the same set!
- 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

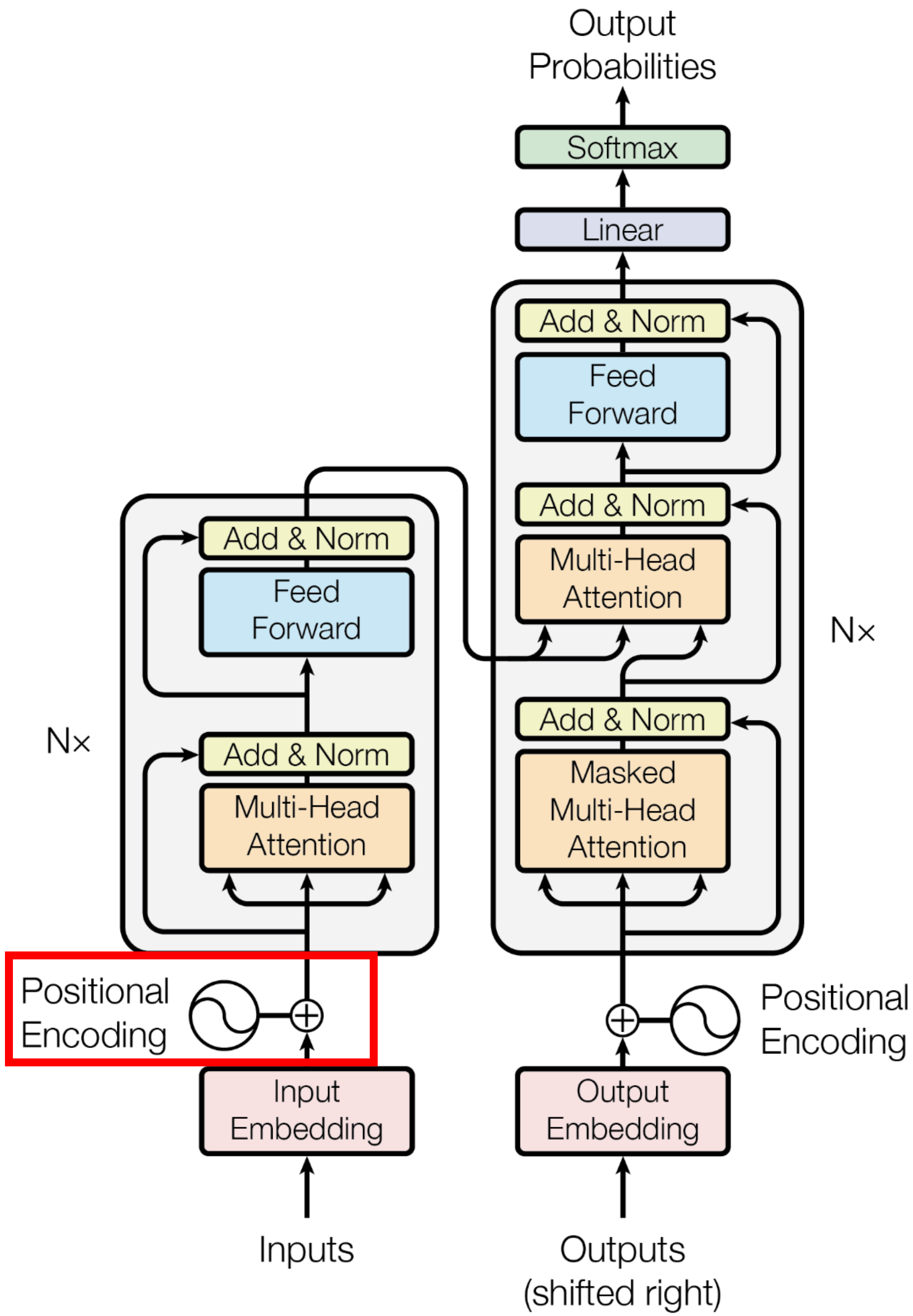
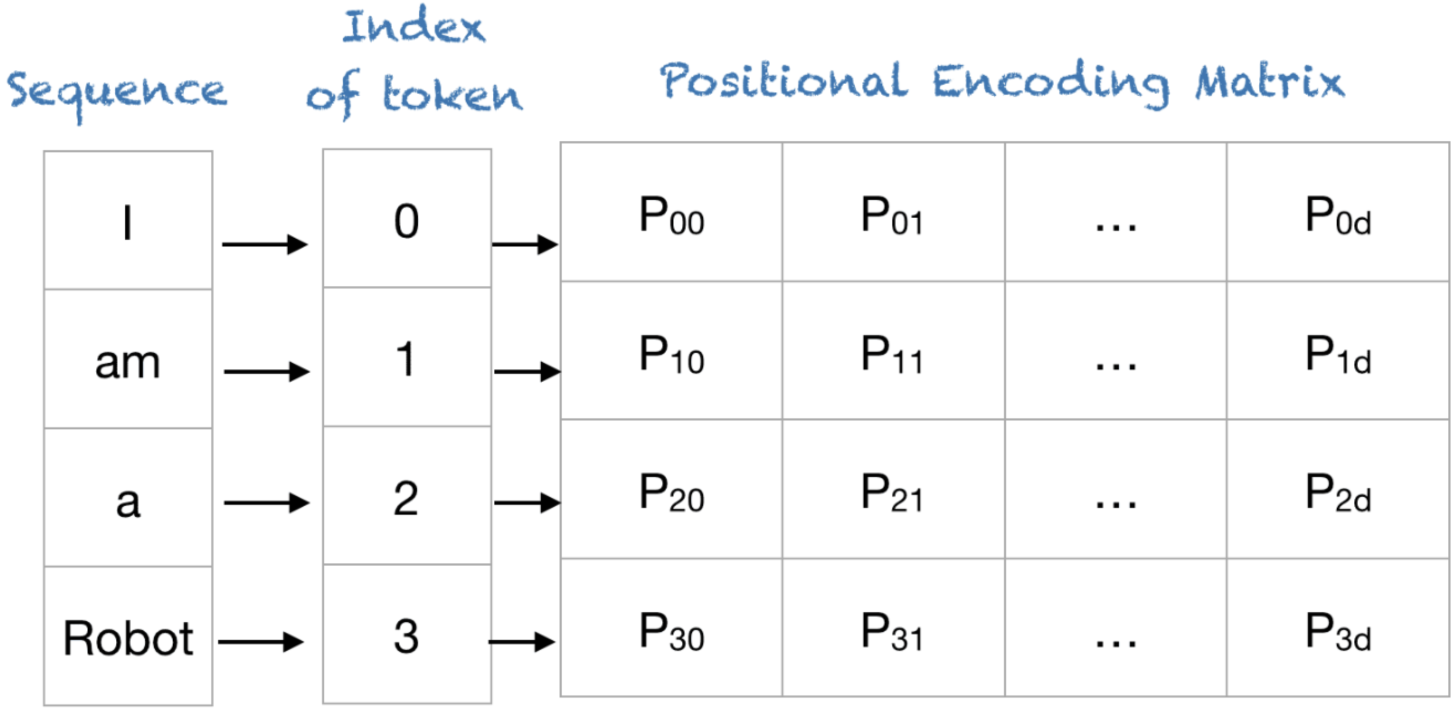


Figure 1: The Transformer - model architecture.

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



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

- 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 a number in binary
  - The lowest bit alternates with every number
  - The second-lowest bit alternates every two numbers, and and and higher bits continue this pattern.
- But using binary values would be a waste of space
- 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

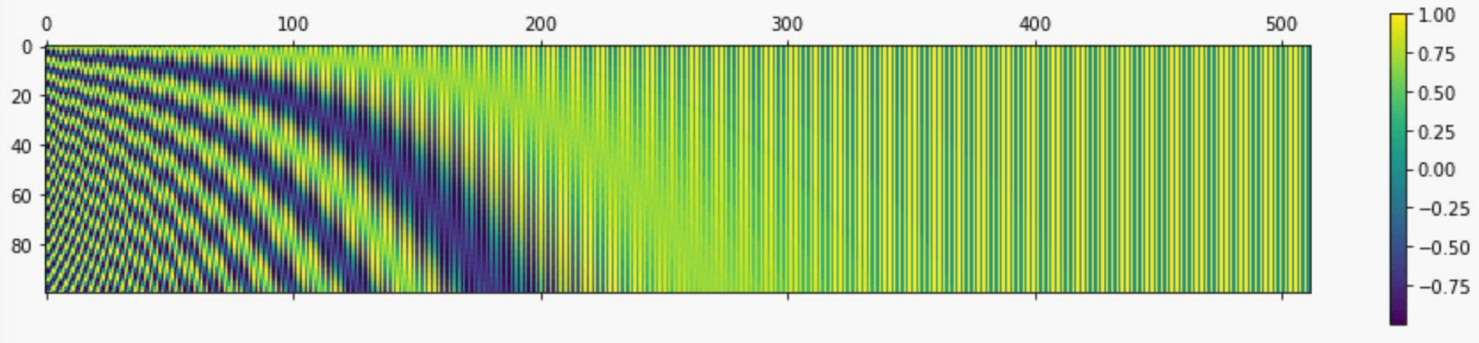
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

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 suggested

$$P_{k,i} = \sin\left(\frac{k}{10000^{\frac{2i}{d}}}\right) \quad \text{if } i \text{ is even}$$

$$P_{k,i} = \cos\left(\frac{k}{10000^{\frac{2i}{d}}}\right) \quad \text{if } i \text{ is odd}$$



The positional encoding matrix for n=10,000, d=512, sequence length=100

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

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

- 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!

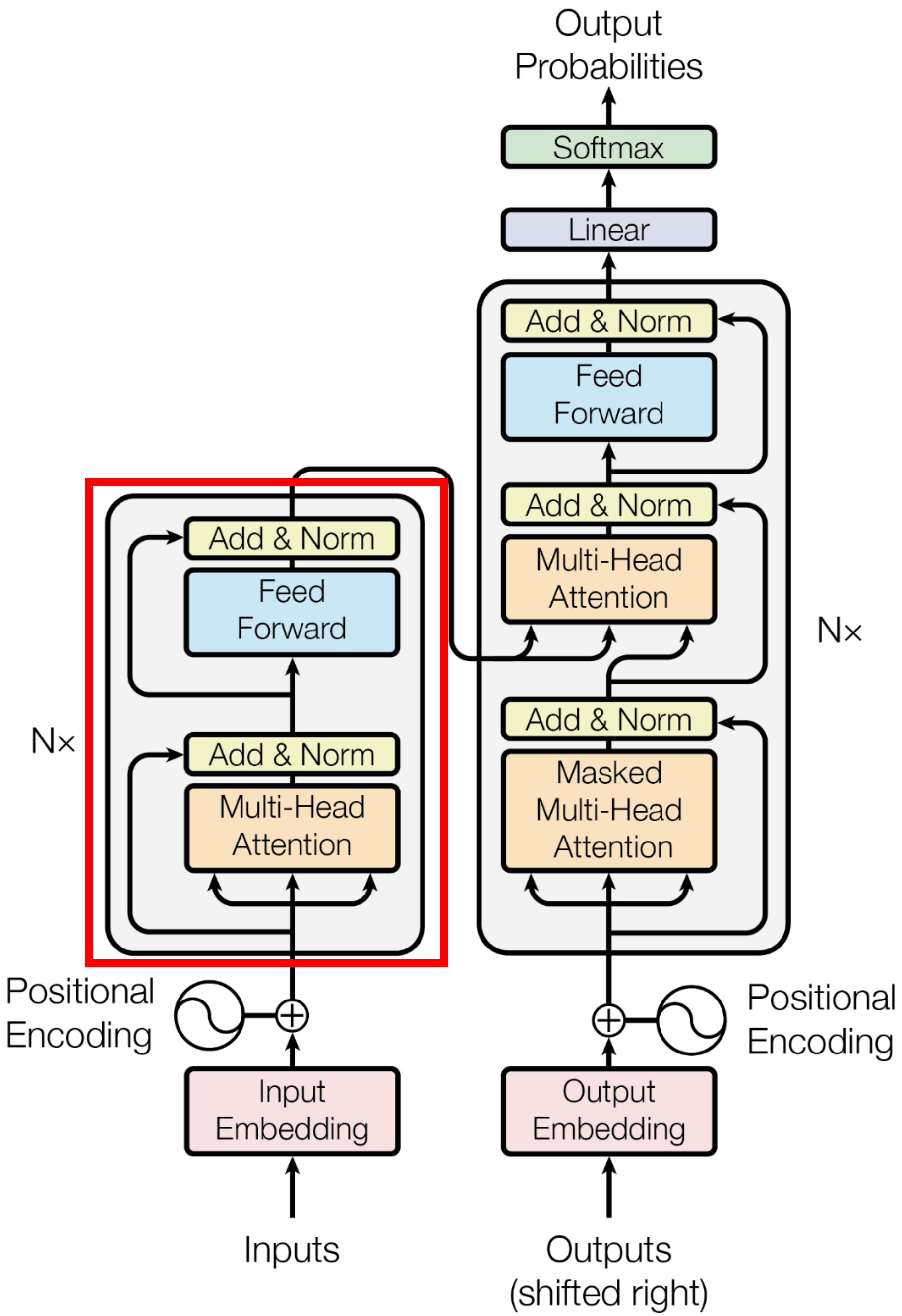


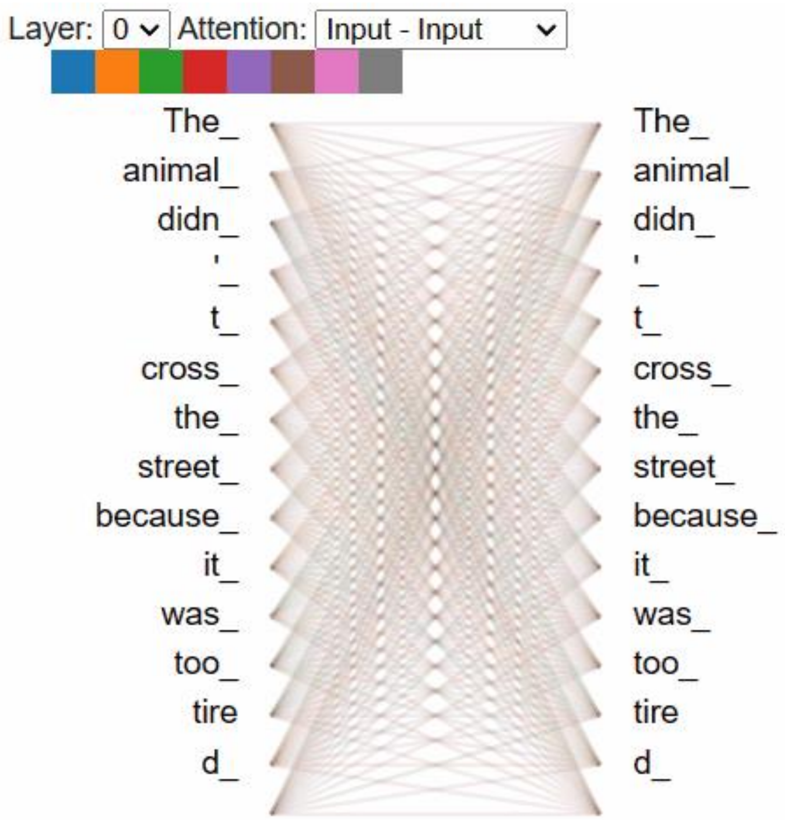
Figure 1: The Transformer - model architecture.

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?

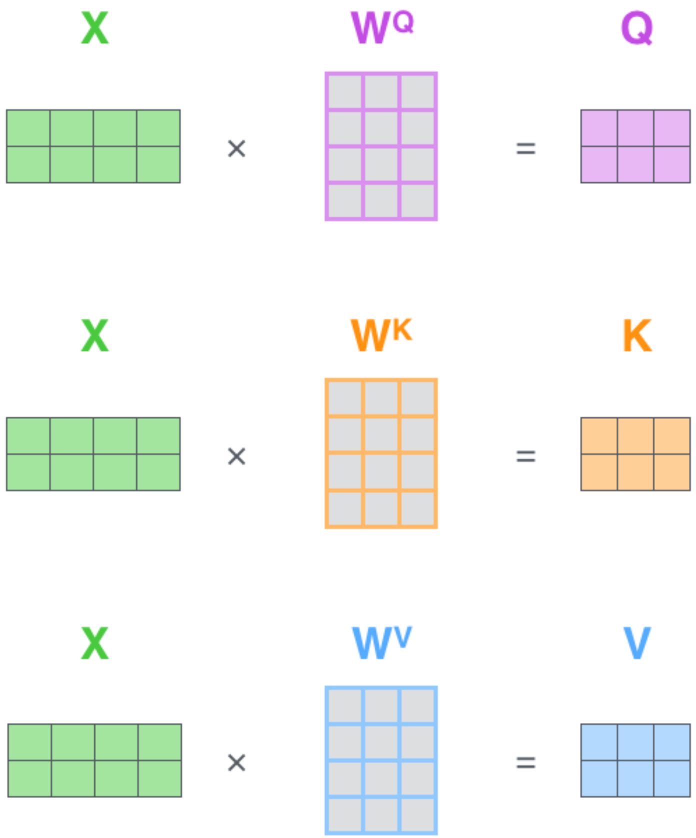
- Self-attention allows each token to attend to every other token in the sequence, helping the model capture context and relationships between words.
  - When processing "it", the model uses attention to understand that "it" refers to "animal.”

- 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.

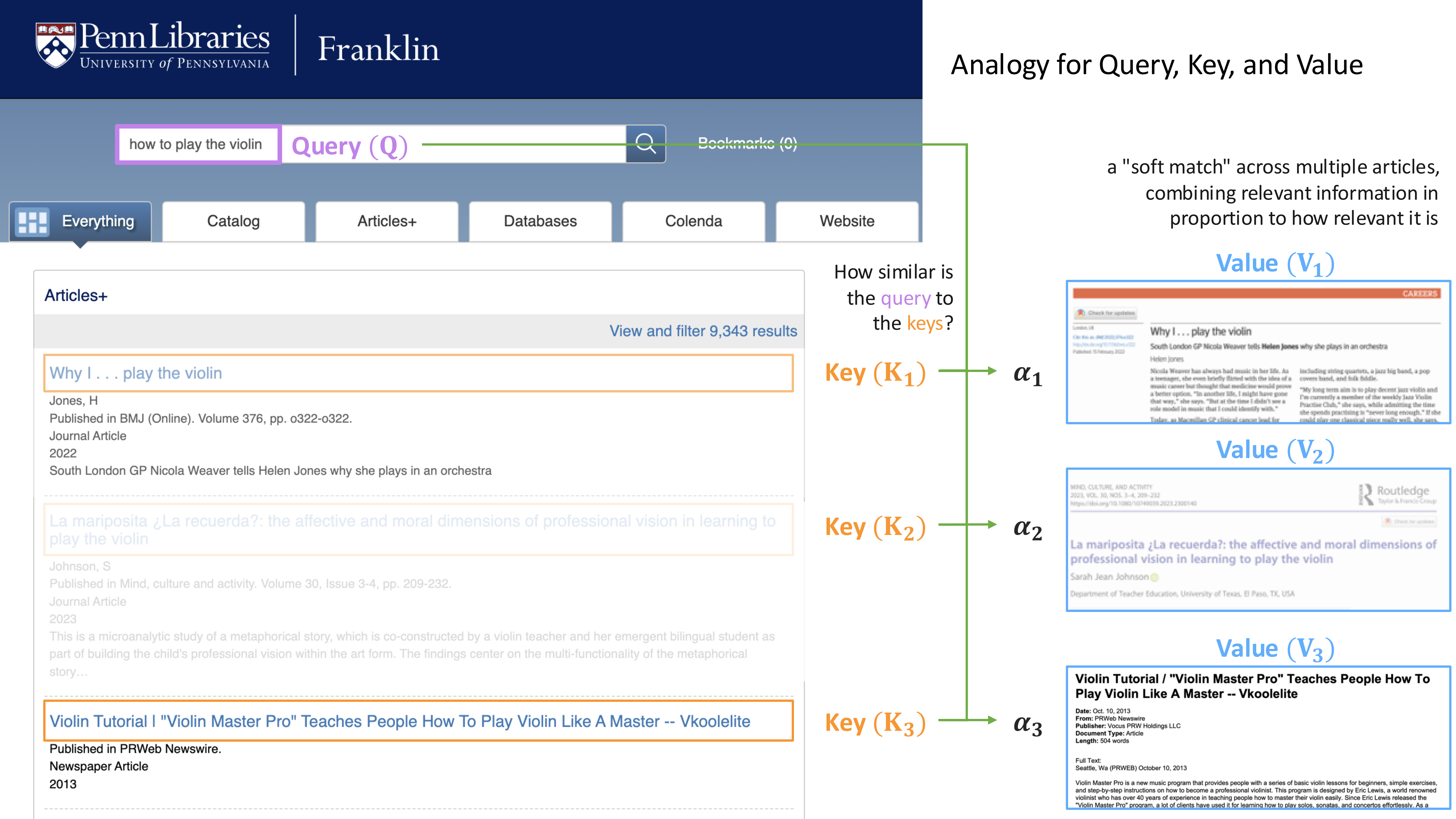


Self-Attention

- Given the input embeddings $x = [x_1, \ldots, x_T]$, we generate three matrices:
  - Query matrix Q
  - Key matrix K
  - Value matrix V
- Input embeddings are transformed into these matrices by multiplying the embeddings by three weight matrices W^Q, W^K, W^V that we learn during the training process.



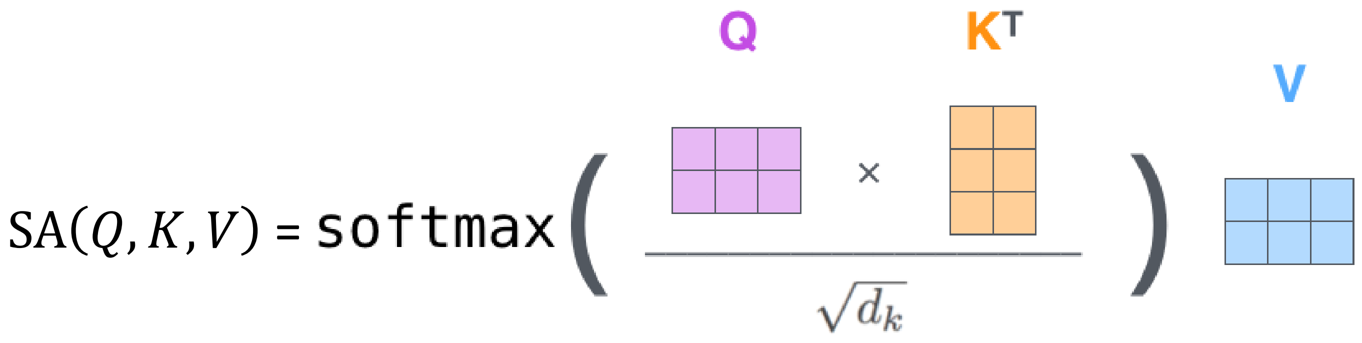
- Analogy for Query, Key, and Value: Library System
  - Imagine you're looking for information on a topic (**query**)
  - Each book has a summary (**key**) to help you identify if it contains relevant information.
  - Once you find a match, you access the book to get the detailed information (**value**) you need.
  - In Attention, we do a "soft match" across multiple books, combining relevant information from each book in proportion to how relevant it is (e.g., book 1 is most relevant, then book 2, etc).

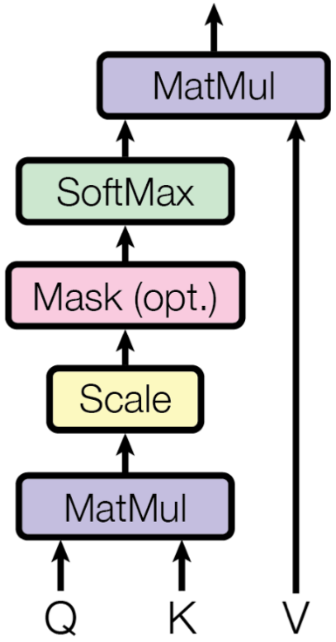


Penn Libraries
University of Pennsylvania
Franklin
Analogy for Query, Key, and Value
how to play the violin
Query (Q)
Bookmarks (0)
Everything
Catalog
Articles+
Databases
Colenda
Website
Articles+
View and filter 9,343 results
Why I . . . play the violin
Jones, H
Published in BMJ (Online). Volume 376, pp. o322-o322.
Journal Article
2022
South London GP Nicola Weaver tells Helen Jones why she plays in an orchestra
La mariposita ¿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 | "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)
Key (K_2)
Key (K_3)
α_1
α_2
α_3
a "soft match" across multiple articles, combining relevant information in proportion to how relevant it is
Value (V_1)
CAREERS
Why I . . . play the violin
South London GP Nicola Weaver tells Helen Jones why she plays in an orchestra
Helen Jones
Value (V_2)
Routledge
La mariposita ¿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
Date: Oct. 10, 2013
From: PRWeb Newswire
Publisher: Vocus PRW Holdings LLC
Document Type: Article
Length: 504 words
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 Q and 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 V to obtain the final attention output.
- The self-attention, thus, is defined as:





- The term "self" comes from the fact that Q, K, V are all derived from the same input sequence $x = [x_1, \ldots, x_T]$

Multi-Head Self-Attention (MSA)

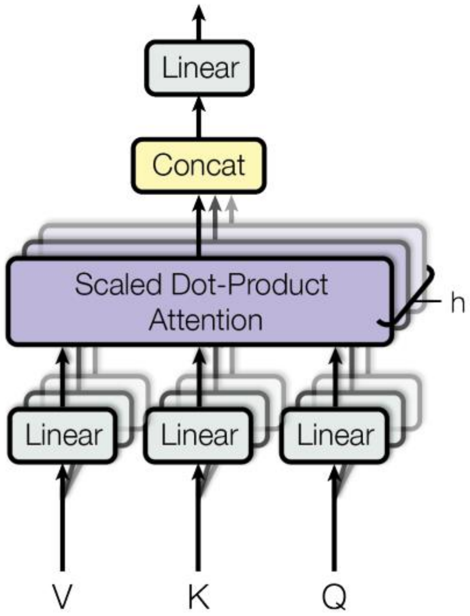
- **Multi-head Self Attention (MSA)** extends Self-Attention by introducing multiple independent attention heads, each focusing on different types of relationships.

- Each head is considered as one copy of a single Self Attention, with additional weight matrices W_i^Q, W_i^K, W_i^V for each head, indexed by i:

$$\mathrm{MSA}(Q, K, V) = [\mathrm{SA}(Q_1, K_1, V_1), \dots, \mathrm{SA}(Q_h, K_h, V_h)]W_O$$

$$Q_i = W_i^Q Q \qquad K_i = W_i^K K \qquad V_i = W_i^V V$$

- Where $W_O \in \mathbb{R}^{(h \cdot d_v) \times d}$ is the weighting matrix between all attention heads, and W_i^Q, W_i^K, W_i^V are weight matrices of query, key value for each head $i = 1, \dots, h$



- **Multi-Head Cross Attention (MCA)** applies the same mechanism in the context where the *queries*, *keys*, and *values* might come from different sources.

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.

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

- **LayerNorm** normalizes the inputs across the features instead of the batch dimension.
  - This ensures consistent scaling across layers, leading to more stable training.

$$LN(x) = \gamma \cdot \frac{x - \mu}{\sigma} + \beta$$

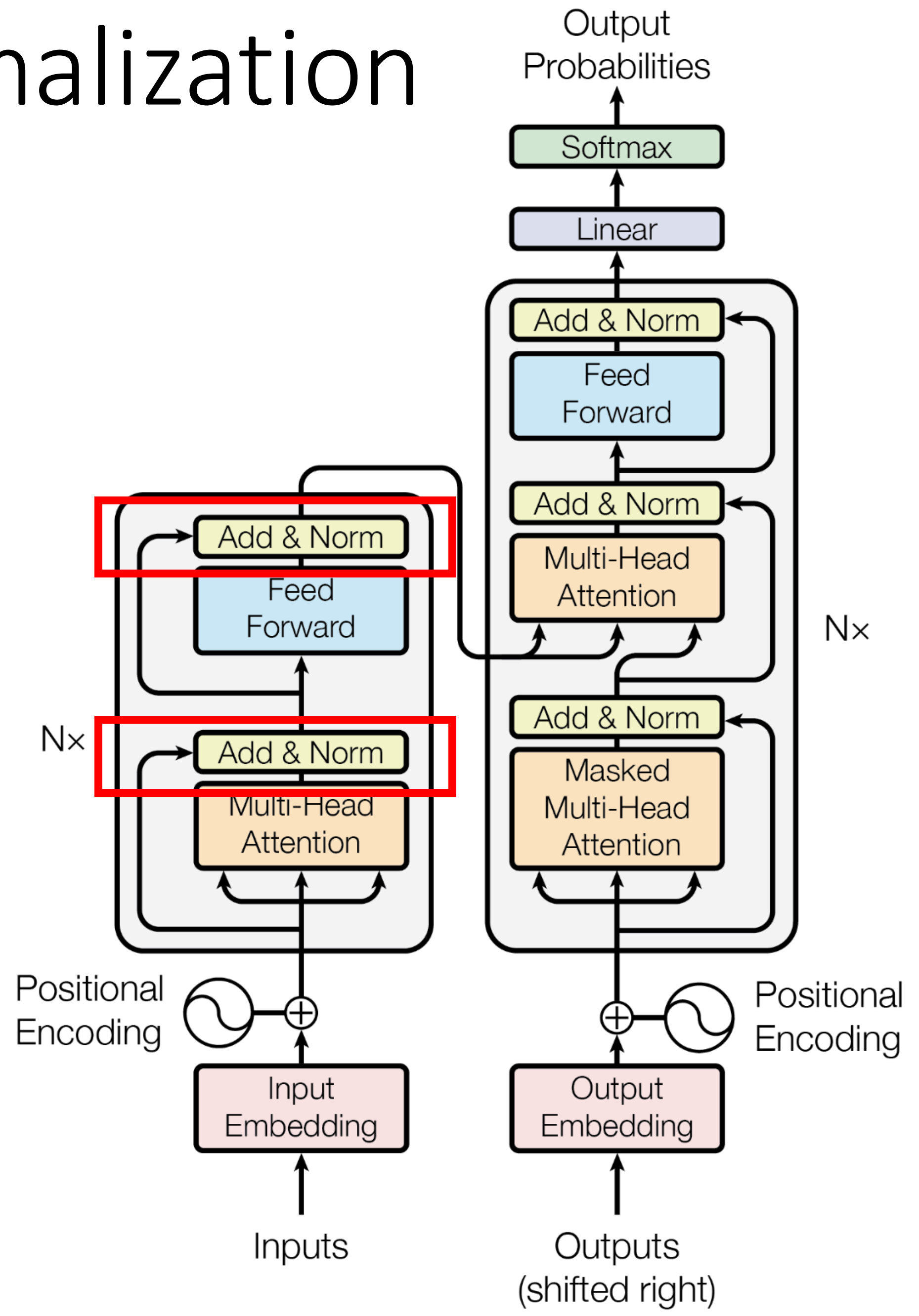


Figure 1: The Transformer - model architecture.

Encoder Block Summarized

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

$$\hat{x}^{(l)} = \mathrm{LN}\big(\mathrm{MSA}\big(x^{(l-1)}, x^{(l-1)}, x^{(l-1)}\big) + x^{(l-1)}\big)$$
$$x^{(l)} = \mathrm{LN}\big(\mathrm{FFN}\big(\hat{x}^{(l)}\big) + \hat{x}^{(l)}\big)$$

  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**

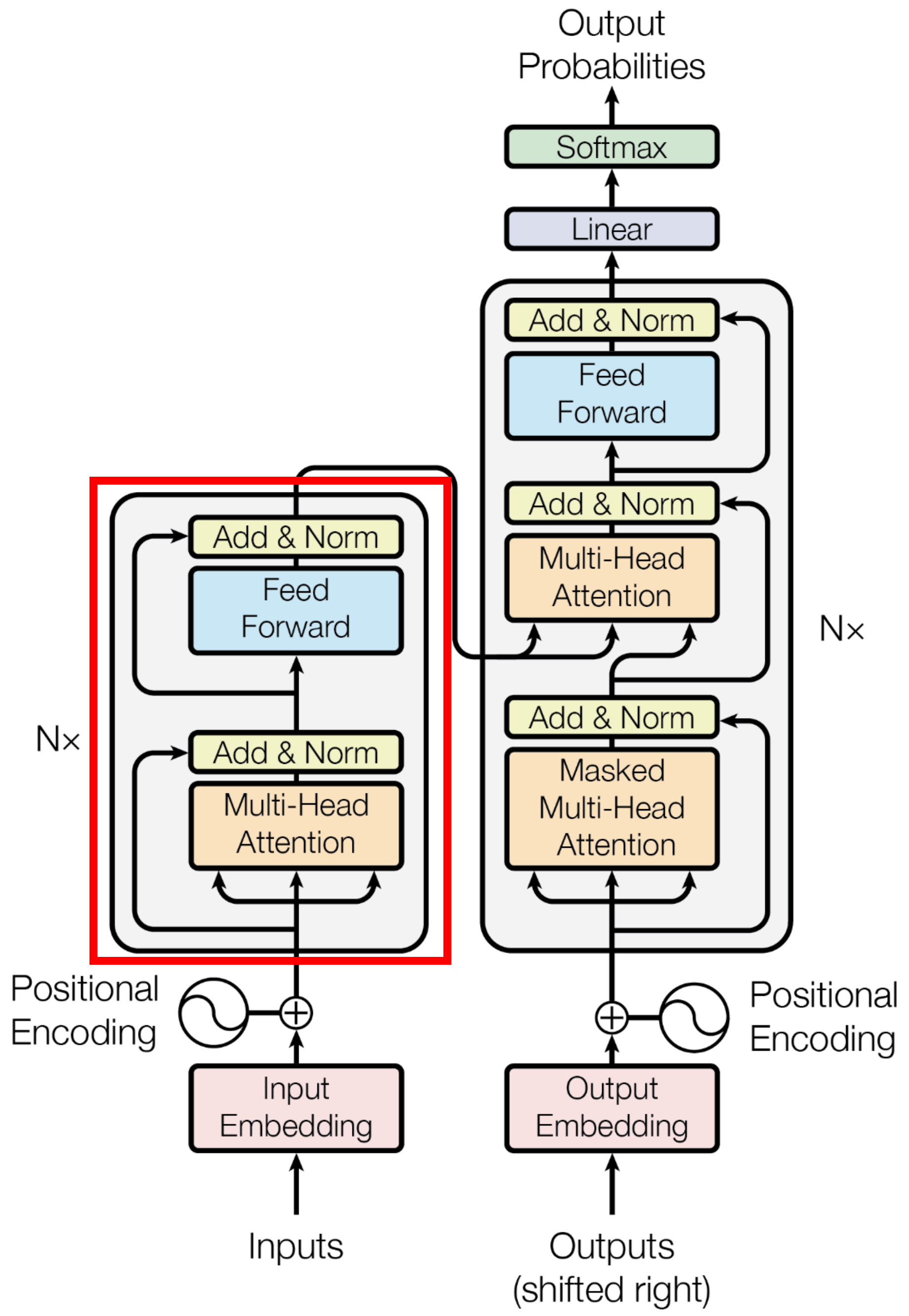
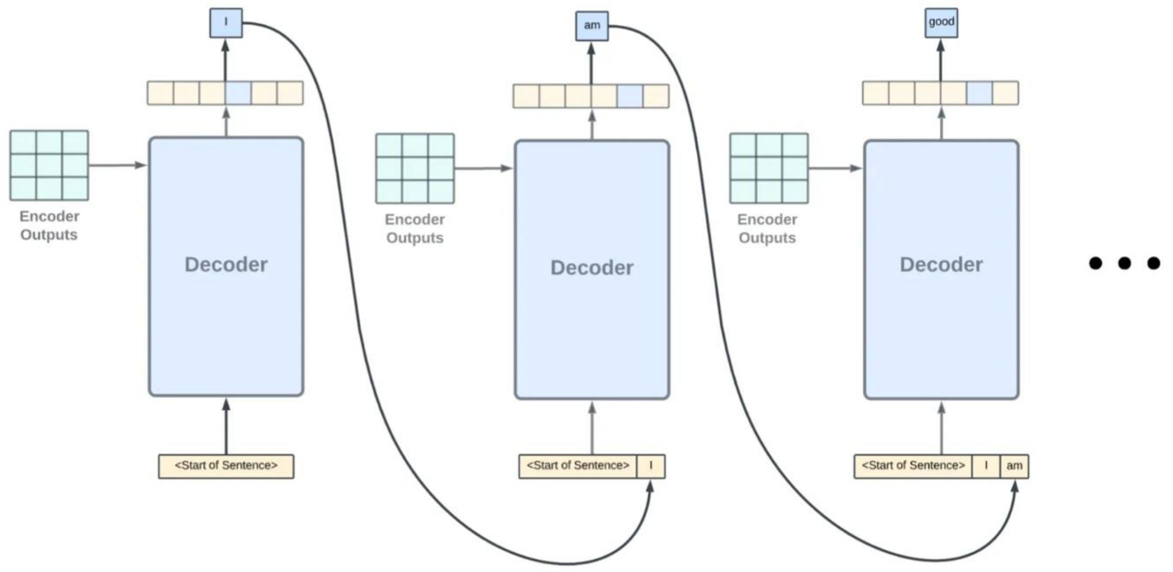


Figure 1: The Transformer - model architecture.

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

- 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 (ignore "masked" part for now)
    - *Allows each token to attend to previous ones in the sequence.*

$$\hat{y}^{(l)} = \mathrm{LN}\left(\mathrm{MaskedMSA}\left(y^{(l-1)}, y^{(l-1)}, y^{(l-1)}\right) + y^{(l-1)}\right)$$

  - The Second one is a Multi-Head *Cross* Attention with key and values matrices from the output of the encoder, and query matrix from the previous Multi-Head Self-Attention
    - *Allows the decoder to focus on relevant part of encoded input*

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

  - $x^{(N)}$ is the output of the encoder (composed of N encoder layers)

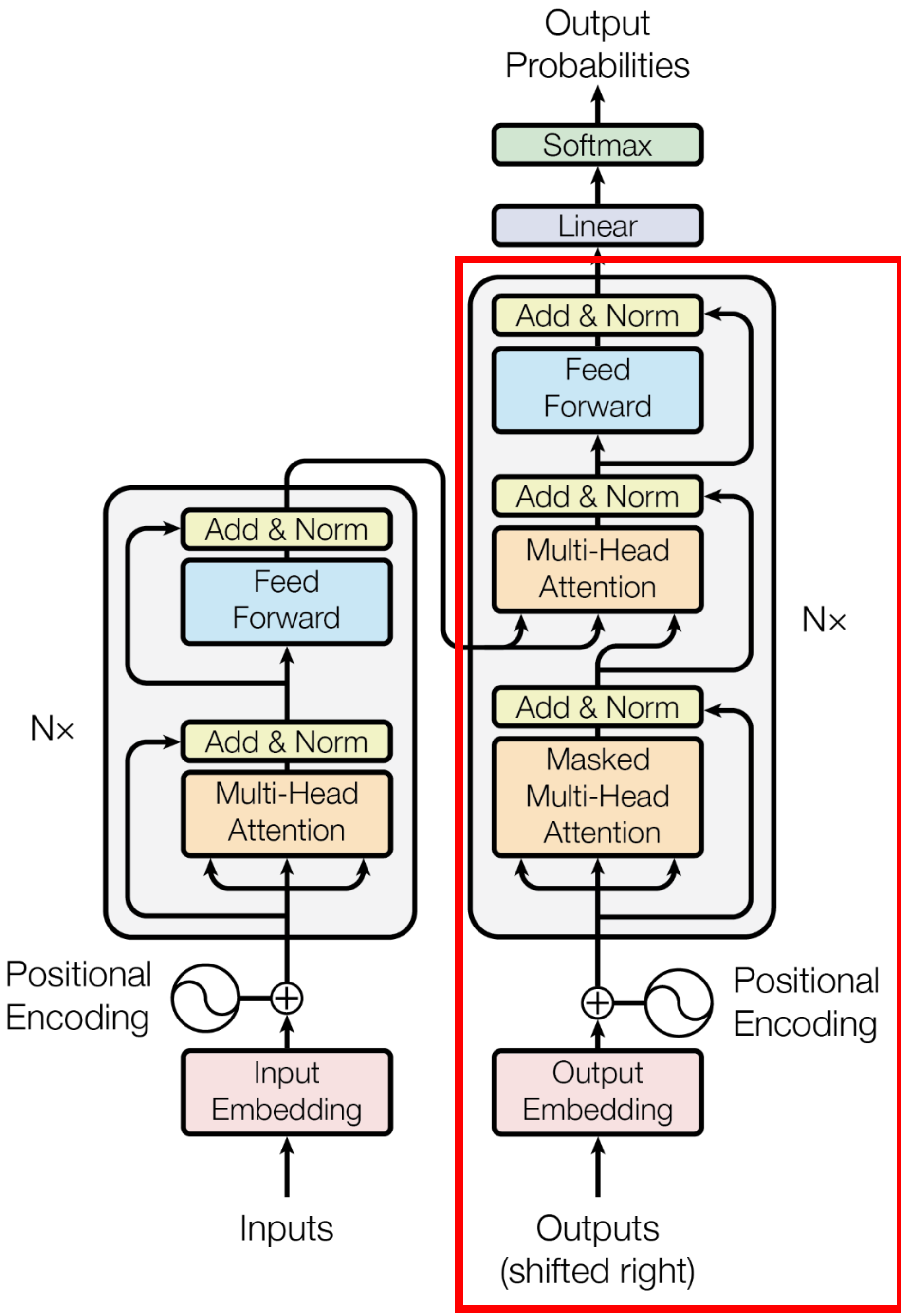


Figure 1: The Transformer - model architecture.

Decoder Block: Summarized

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

$$\hat{y}^{(l)} = \mathrm{LN}\left(\mathrm{MaskedMSA}\left(y^{(l-1)}, y^{(l-1)}, y^{(l-1)}\right) + y^{(l-1)}\right)$$
$$\tilde{y}^{(l)} = \mathrm{LN}\left(\mathrm{MCA}\left(\hat{y}^{(l-1)}, x^{(N)}, x^{(N)}\right) + \hat{y}^{(l-1)}\right)$$
$$y^{(l)} = \mathrm{LN}\left(\mathrm{FFN}\left(\tilde{y}^{(l)}\right) + \tilde{y}^{(l)}\right)$$

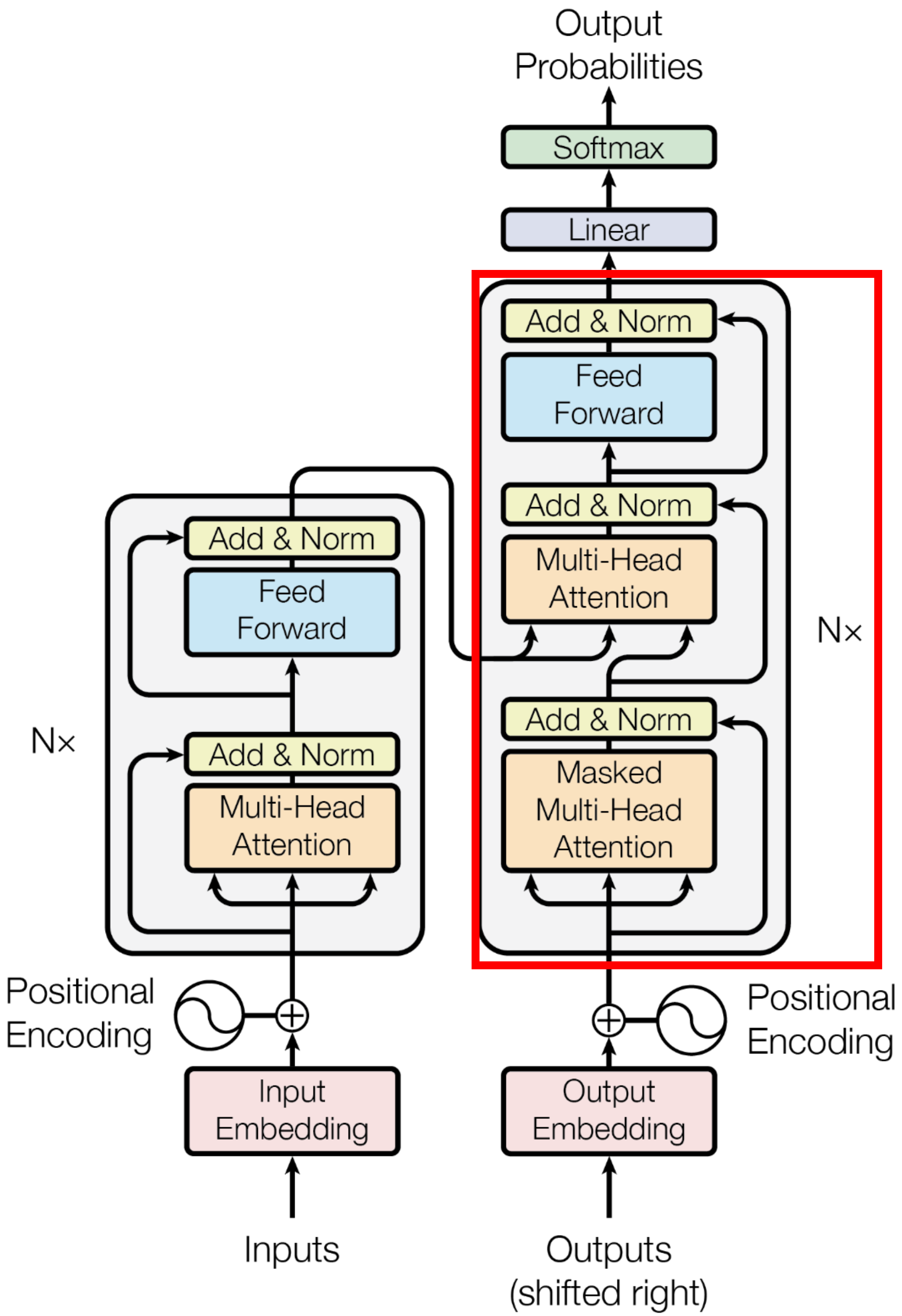


Figure 1: The Transformer - model architecture.

Decoder Block: Masked?

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

$$\hat{y}^{(l)} = \mathrm{LN}\Big(\mathrm{MaskedMSA}\big(y^{(l-1)}, y^{(l-1)}, y^{(l-1)}\big) + y^{(l-1)}\Big)$$
$$\tilde{y}^{(l)} = \mathrm{LN}\Big(\mathrm{MCA}\big(\hat{y}^{(l-1)}, x^{(N)}, x^{(N)}\big) + \hat{y}^{(l-1)}\Big)$$
$$y^{(l)} = \mathrm{LN}\big(\mathrm{FFN}\big(\tilde{y}^{(l)}\big) + \tilde{y}^{(l)}\big)$$

- But, what does the “Masked” in Masked Multi-Head Attention mean?

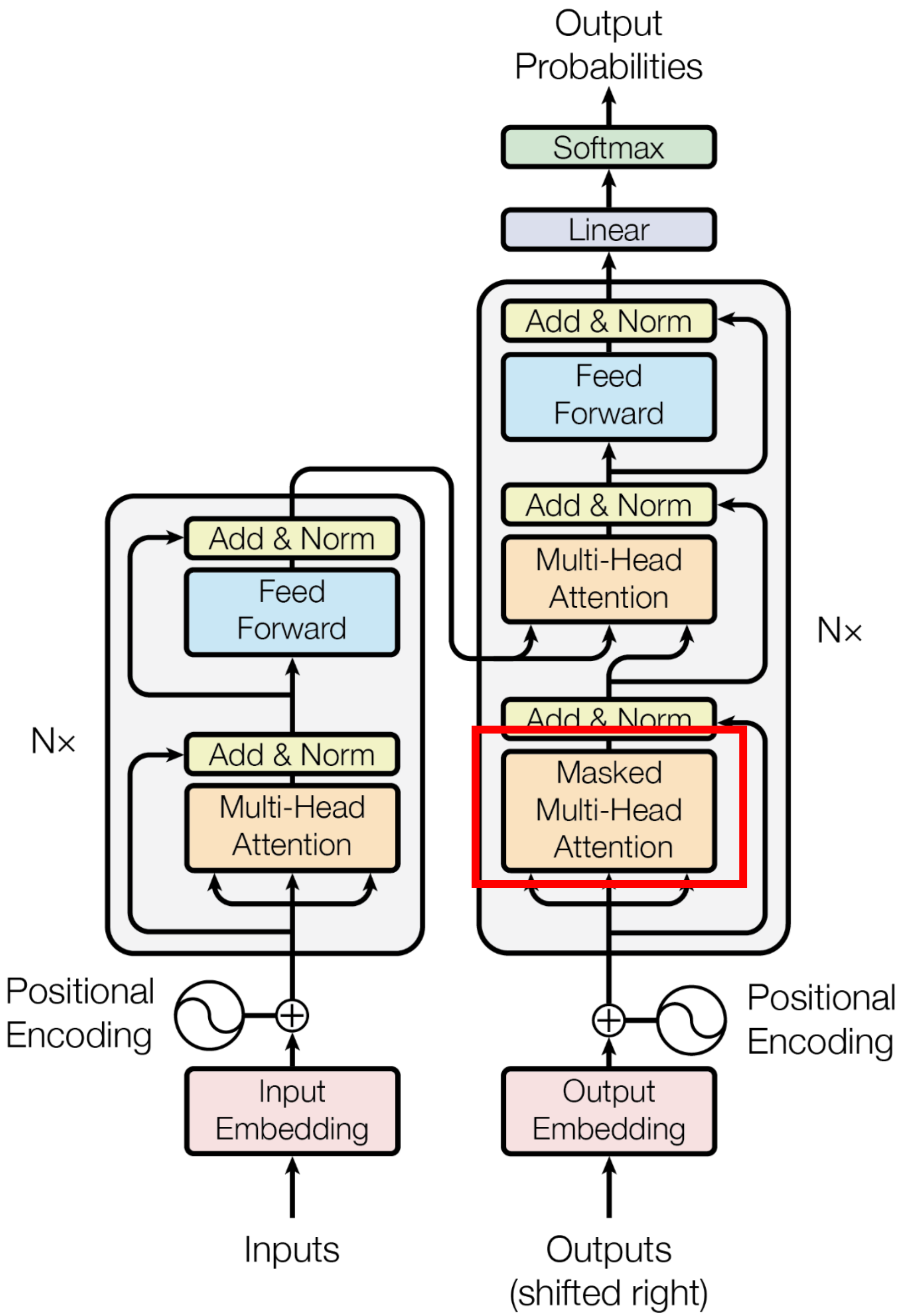
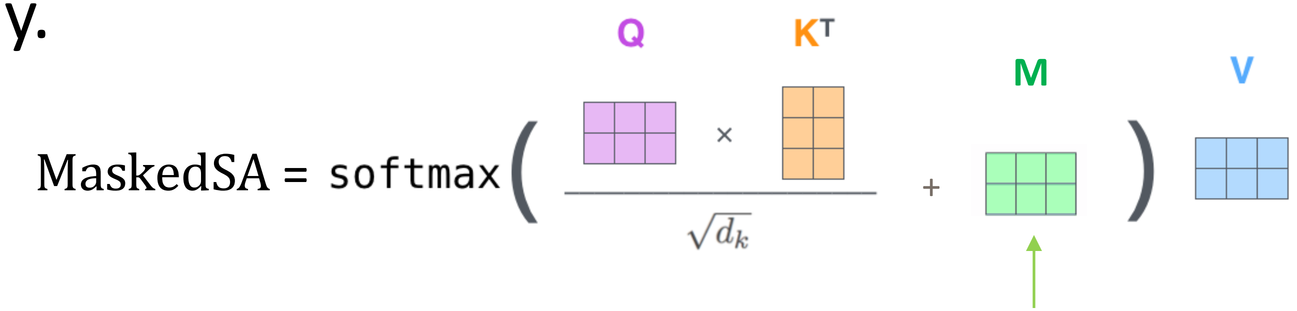


Figure 1: The Transformer - model architecture.

Decoder Block: Masked MHA

- Just liked Multi-Head Attention, MaskedMHA 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, MMHA 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.



M is a mask that applies −∞ to all **future positions** so they don't contribute to the softmax output.

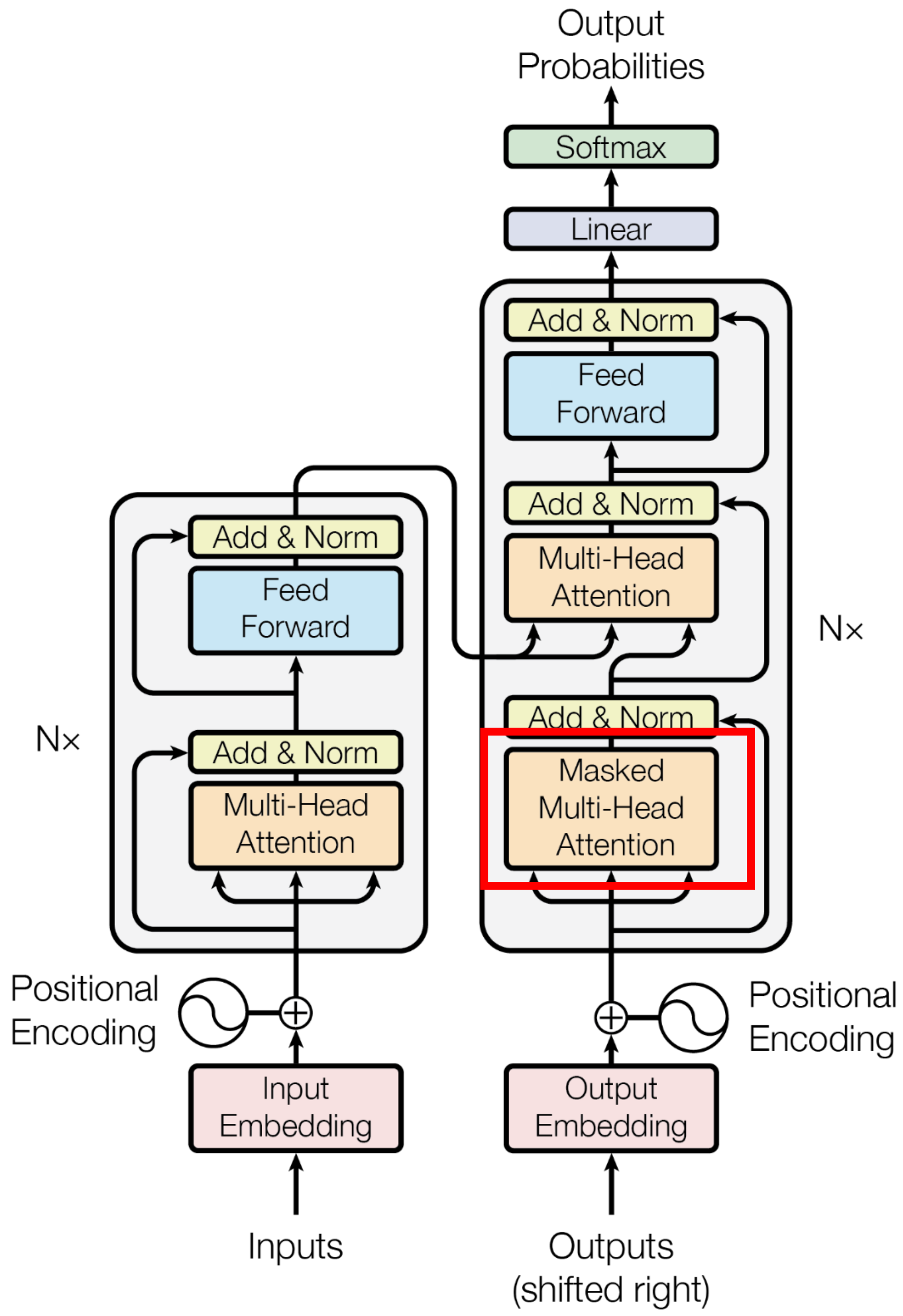


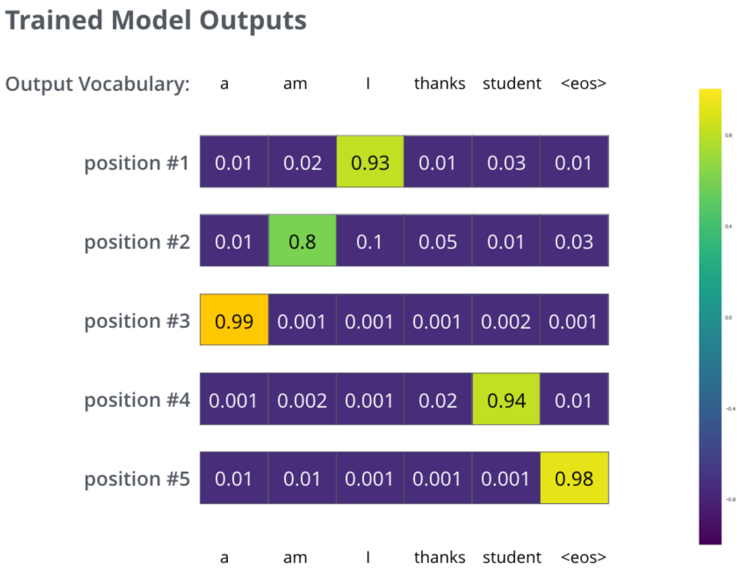
Figure 1: The Transformer - model architecture.

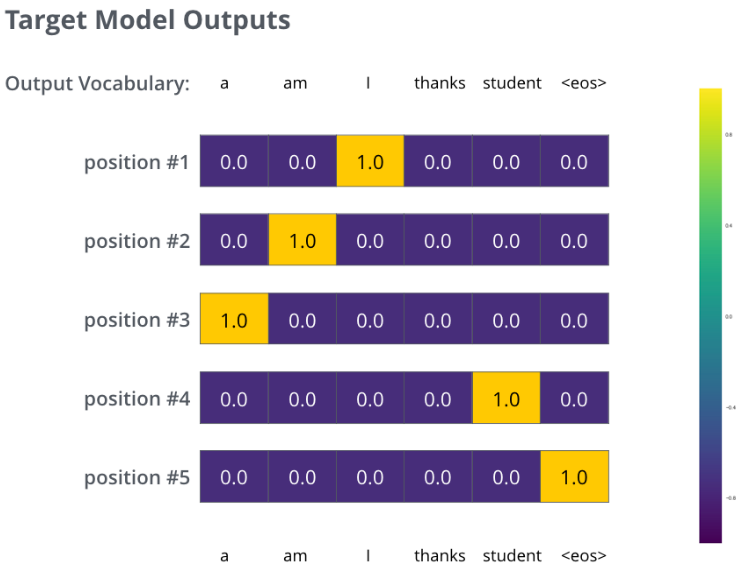
Training

- Training the Transformer shares similar intuition with other Seq2Seq models. The transformer uses masked self-attention in the decoder, which doesn't depend on future words in the sequence.

- The objective is to minimize the prediction error for the next word of the target sequence. For example, when translating "Soy un estudiante" to "I am a student", the training of transformer (θ) is to minimize the KL divergence of the target sequence prediction (y) and the ground truth (x) across the dataset (D).

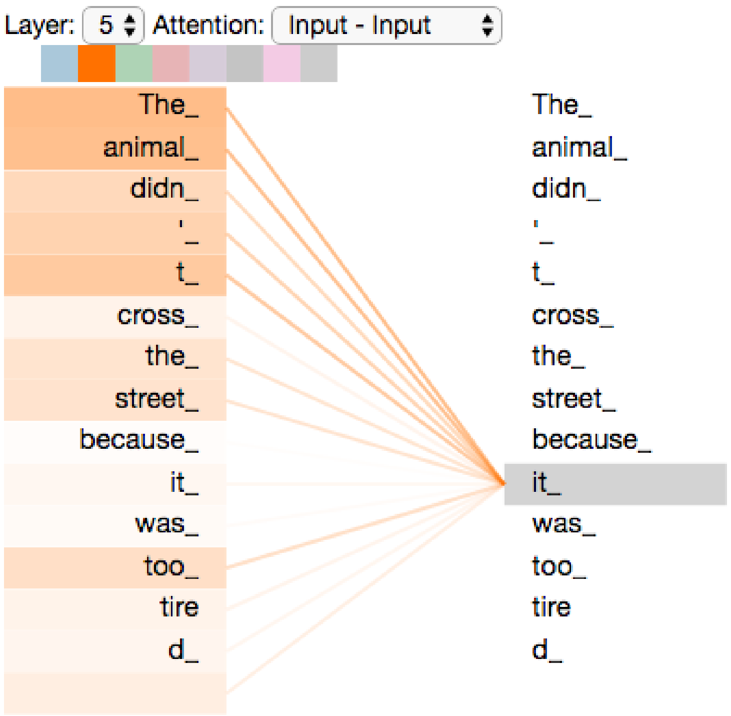
$$\mathcal{L} = - \sum_{(x,y)\in\mathcal{D}} \log P(y|x;\theta)$$

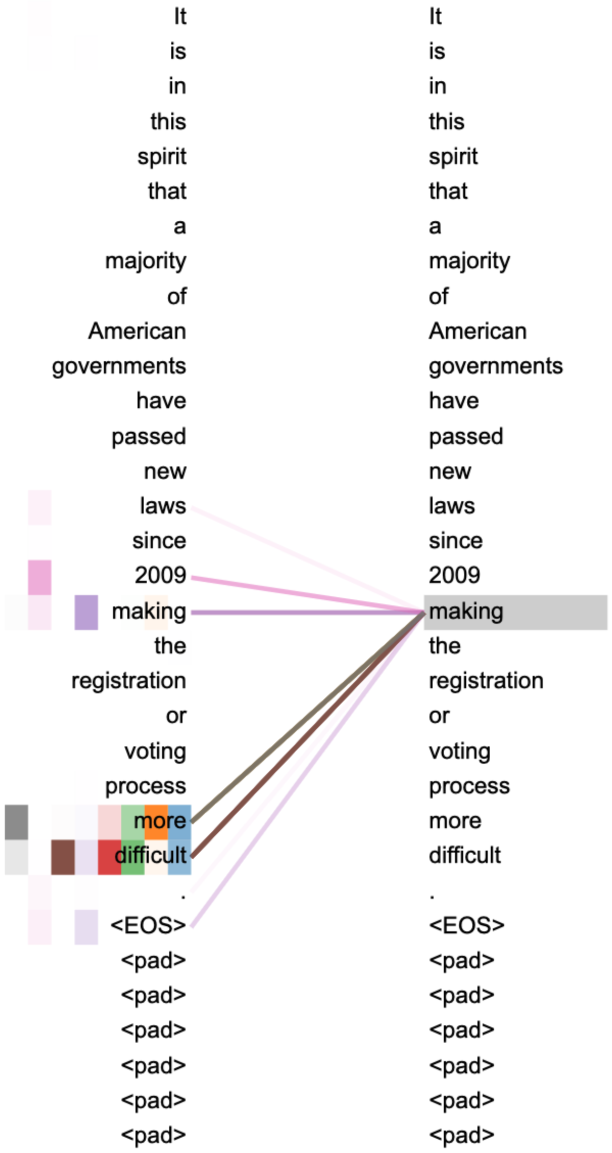




Attention Visualization: Long distance dependency

- 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".

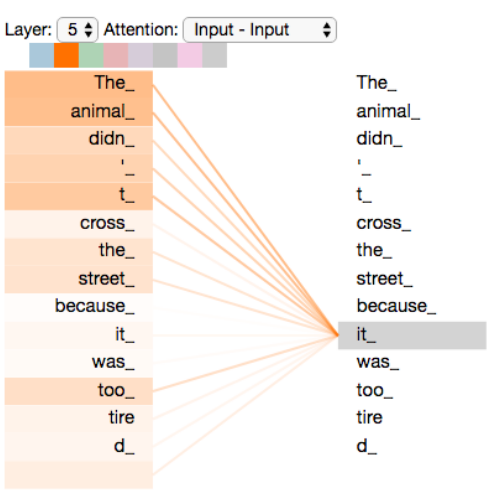


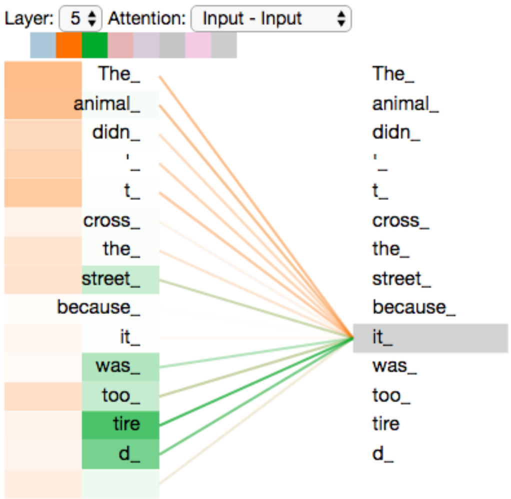


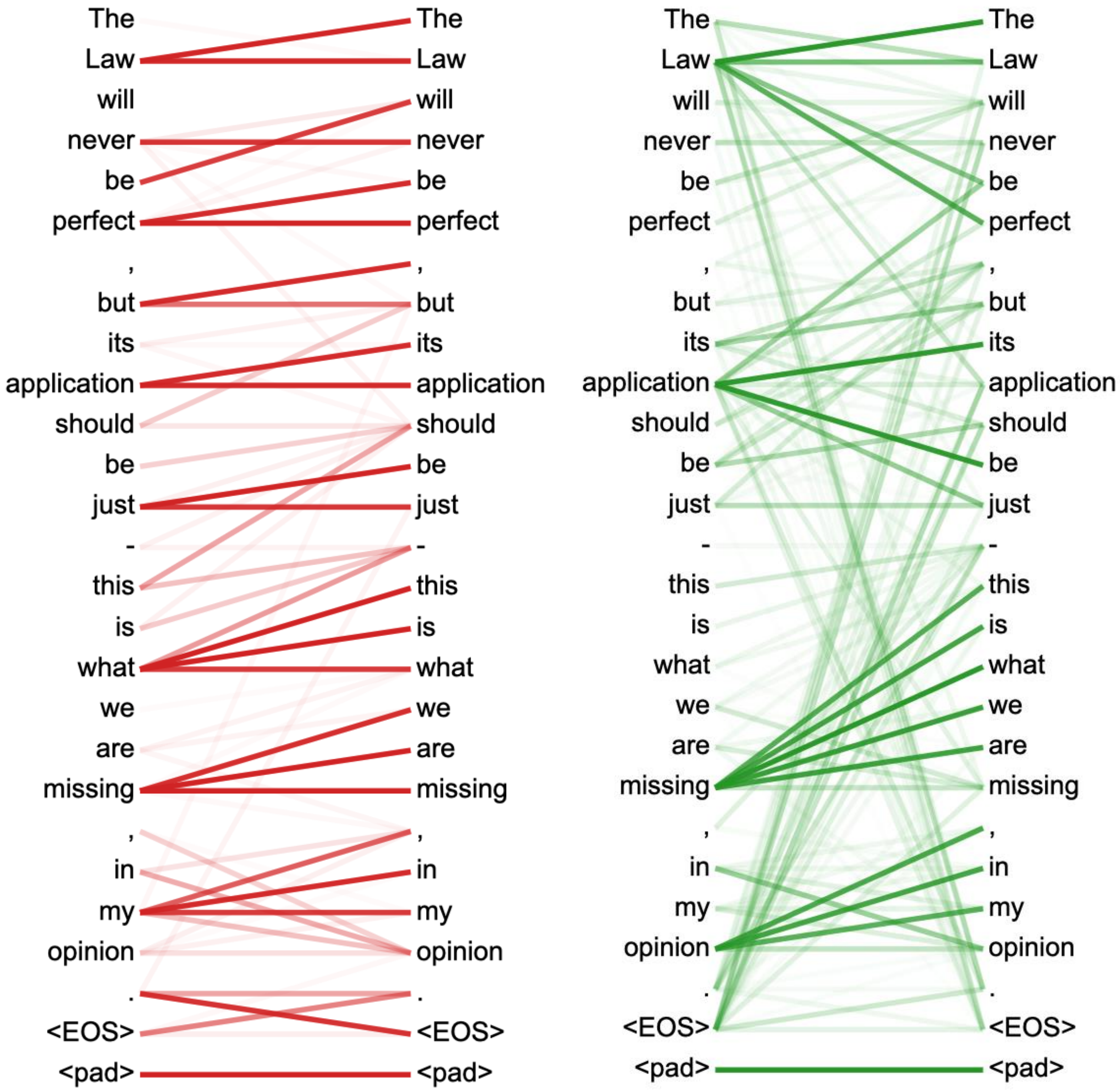
- On the right we see another visualization showing how different words in a longer sentence relate to each other.
- Check out this interactive visualization.

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

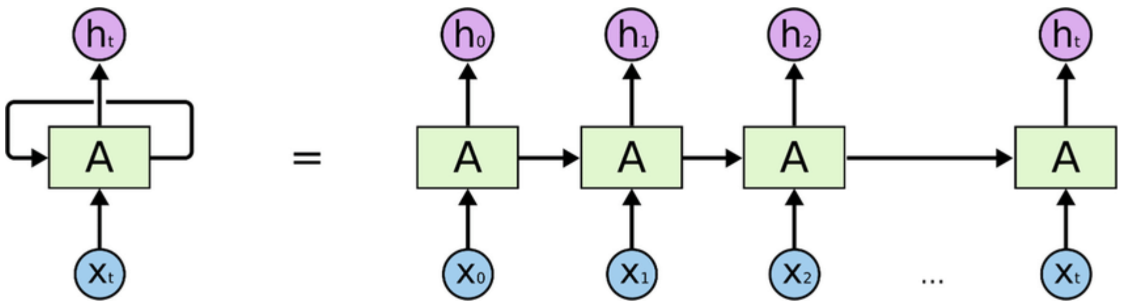
- Handle Sequential Data
- Learn Sequential Dependencies

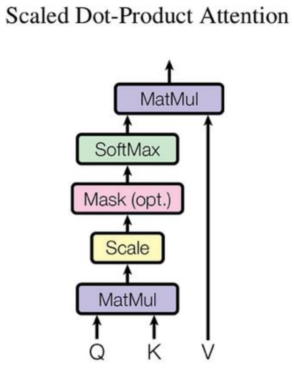
- Each time step depends on the previous one

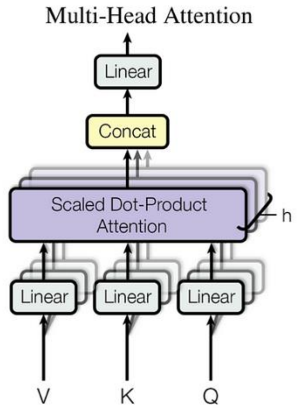
Transformers

- Handle Sequential Data
- Learn Sequential Dependencies

- Use self-attention to capture global context







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

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)

Vision

- Vision Transformer
- Swin Transformer, Pyramid Vision Transformer