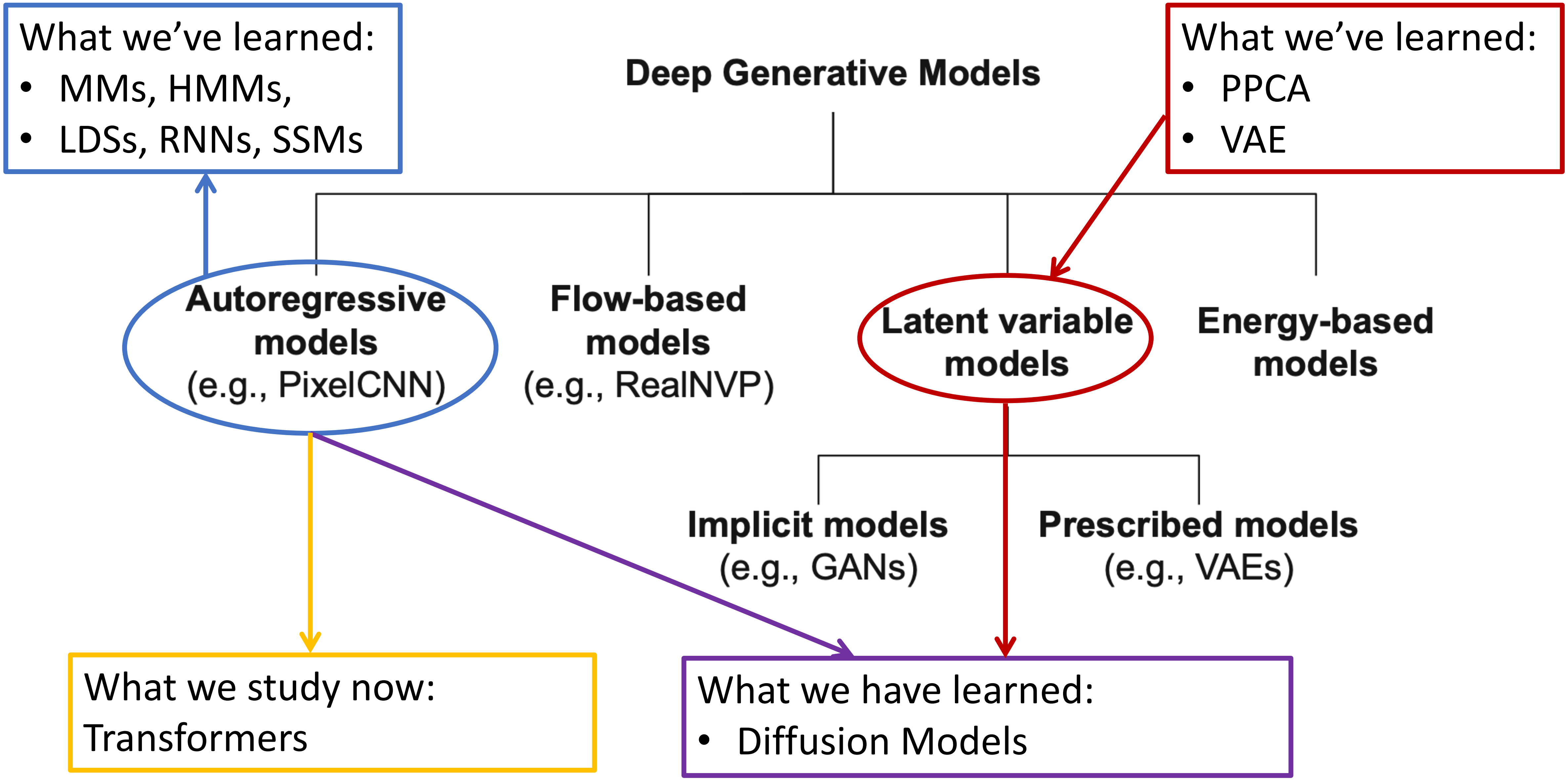
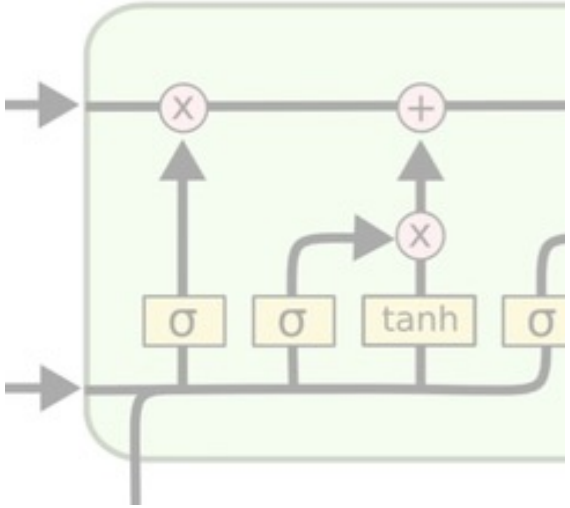
Deep Generative Models: Transformers

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





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

2025 — **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 (?!)

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.

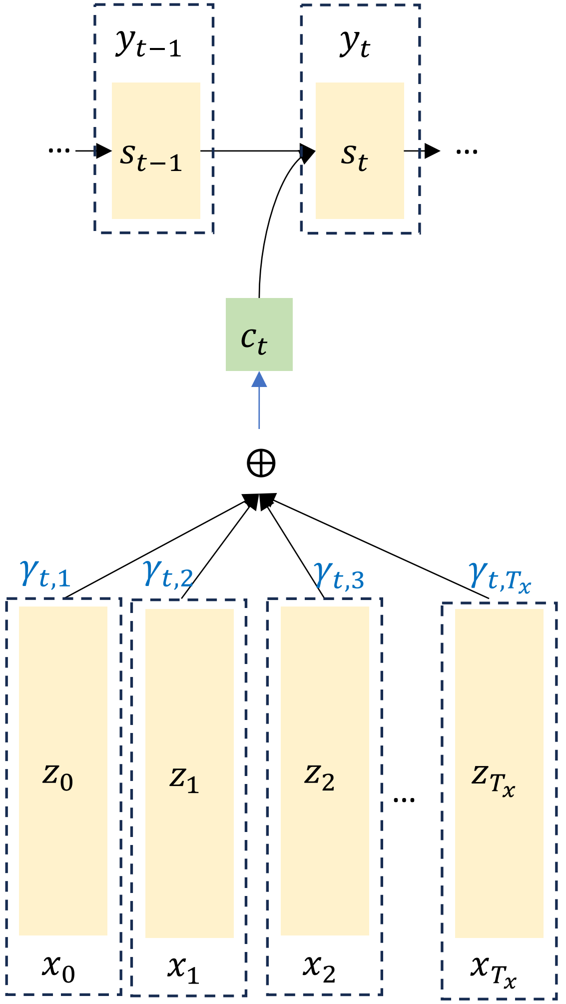
Recall the Translate and Align Model in RNNs

- **Problem:** Given source sentence $\mathbf{x} = (x_1, \ldots, x_T)$, build a model that generates target sentence $\mathbf{y} = (y_1, \ldots, y_T)$.
- **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:

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

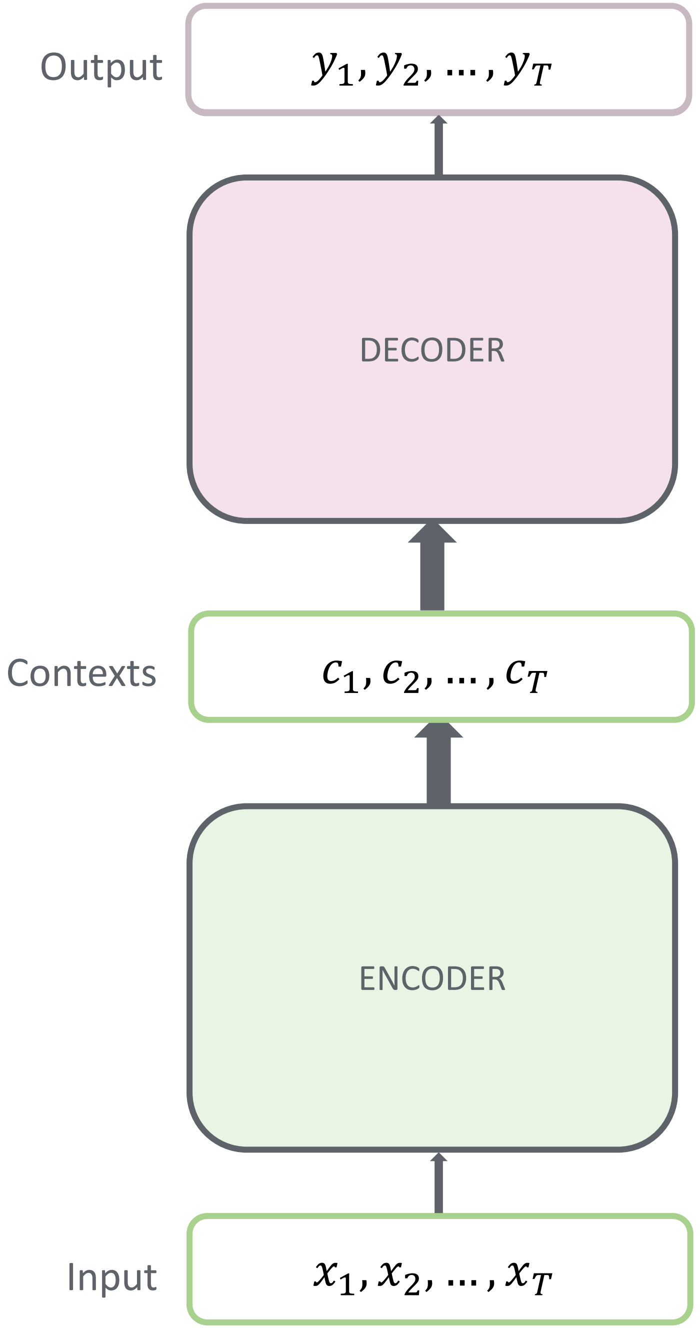
Context vector   Weights of hidden states   Alignment model

- **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

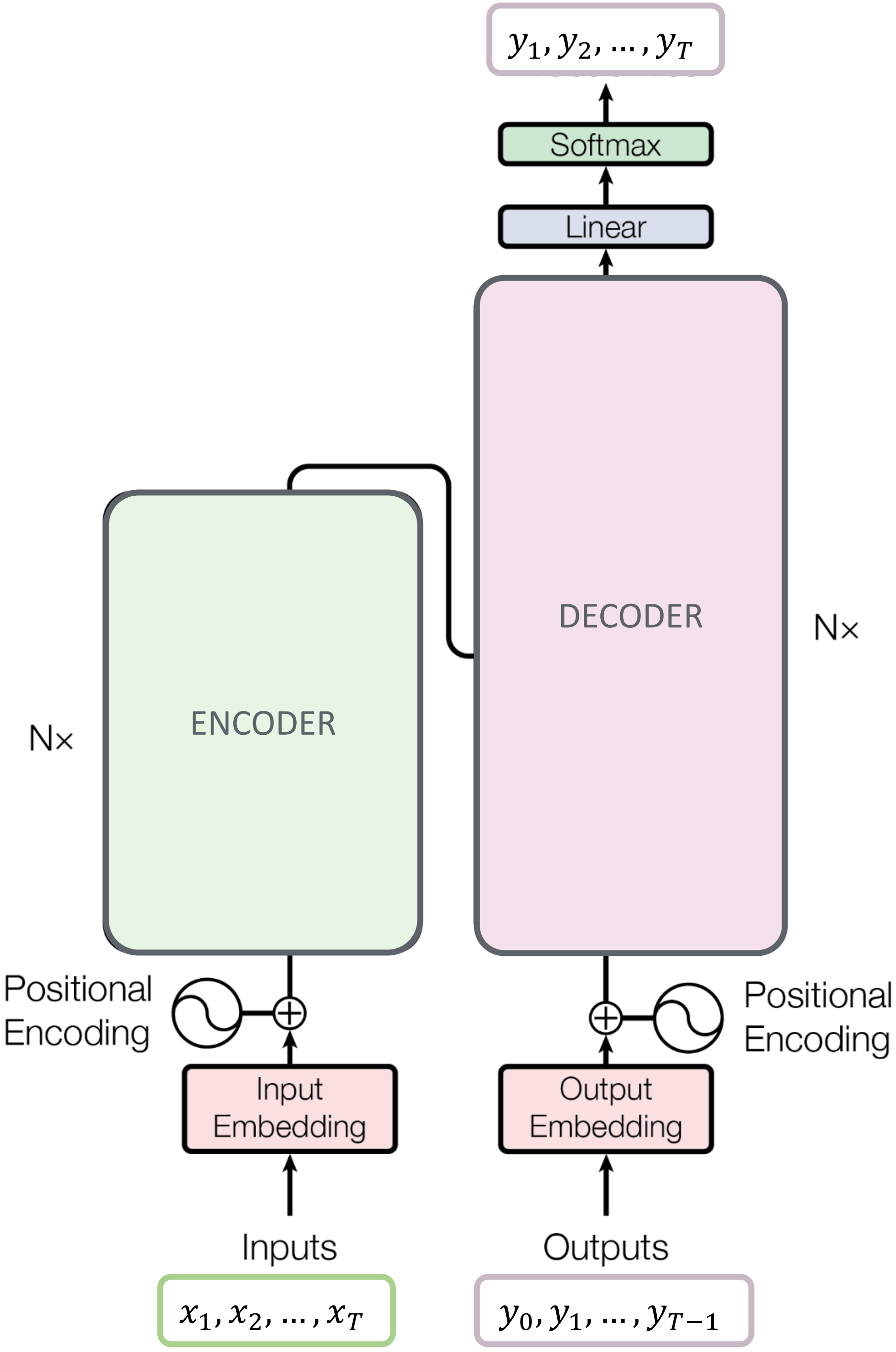


Figure 1: The Transformer - model architecture.

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

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

- Again $\mathbf{E} \in \mathbb{R}^{N \times d}$ 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*.

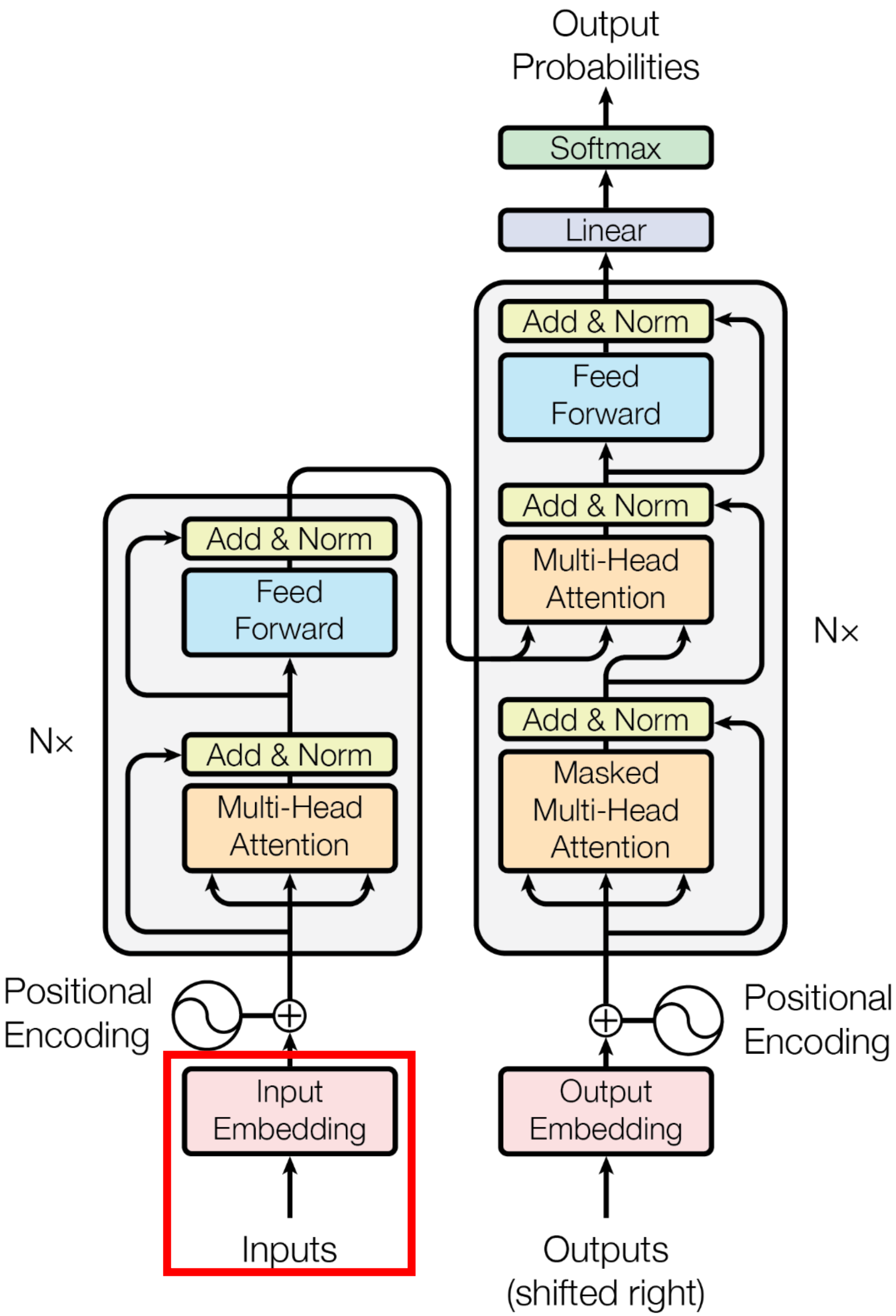


Figure 1: The Transformer - model architecture.

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

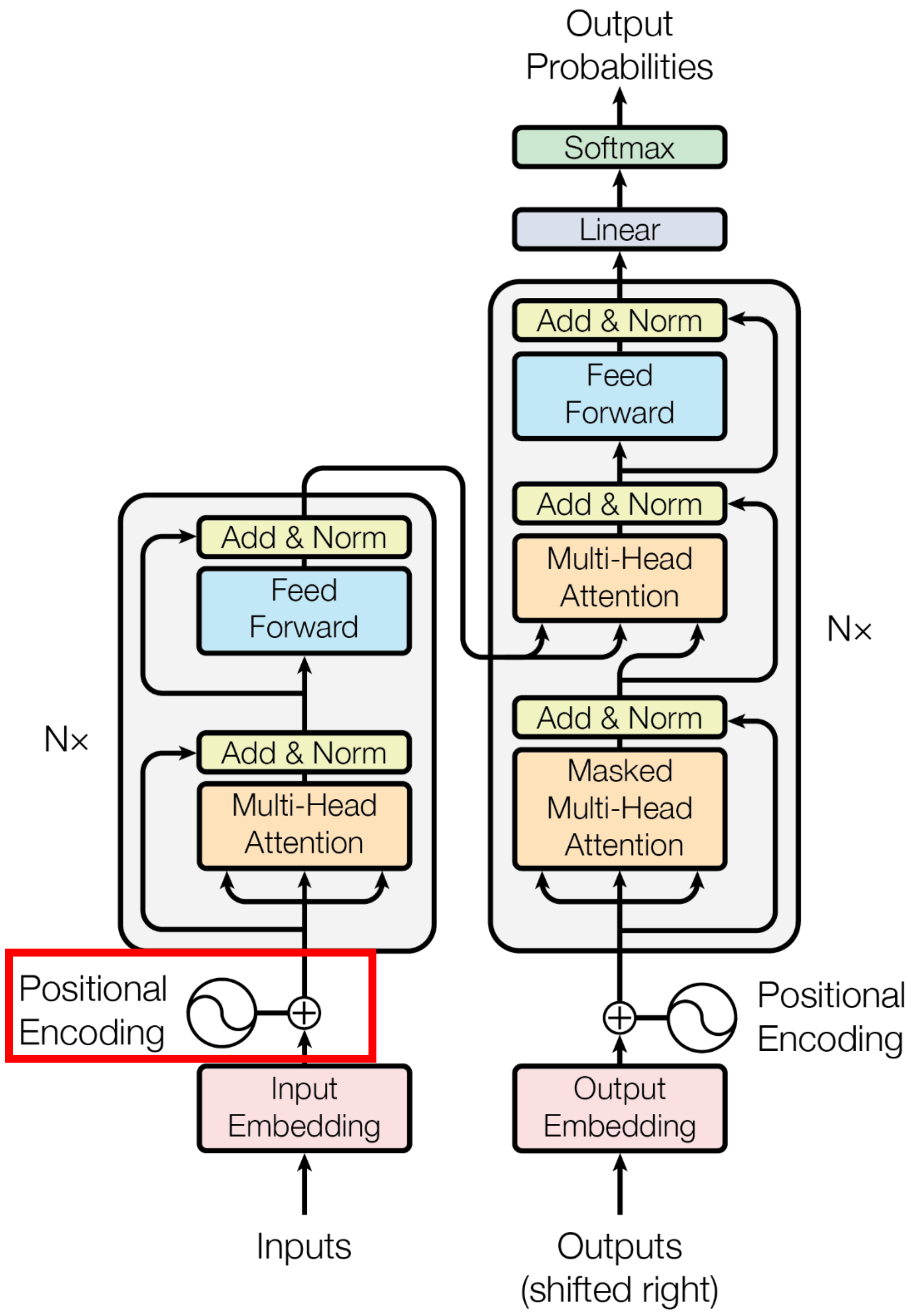
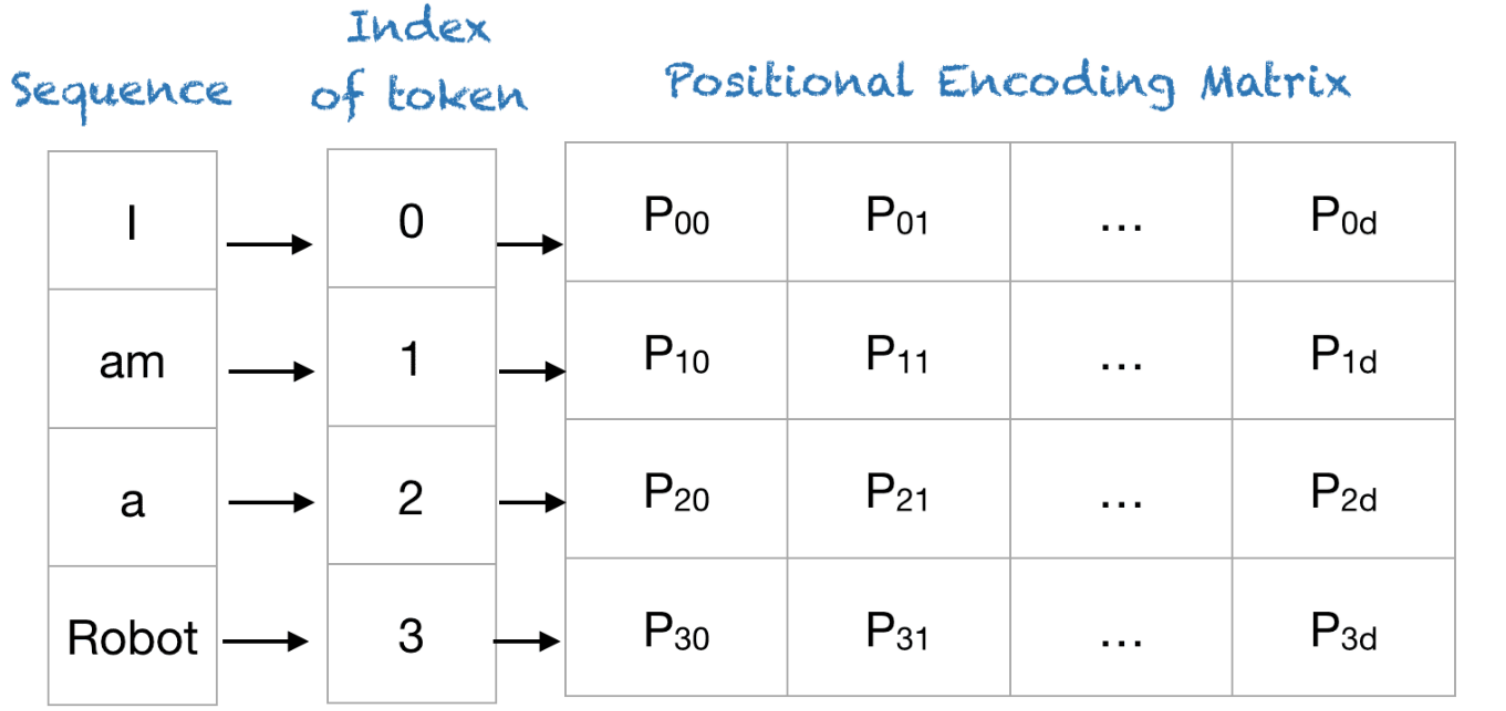


Figure 1: The Transformer - model architecture.

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



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 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

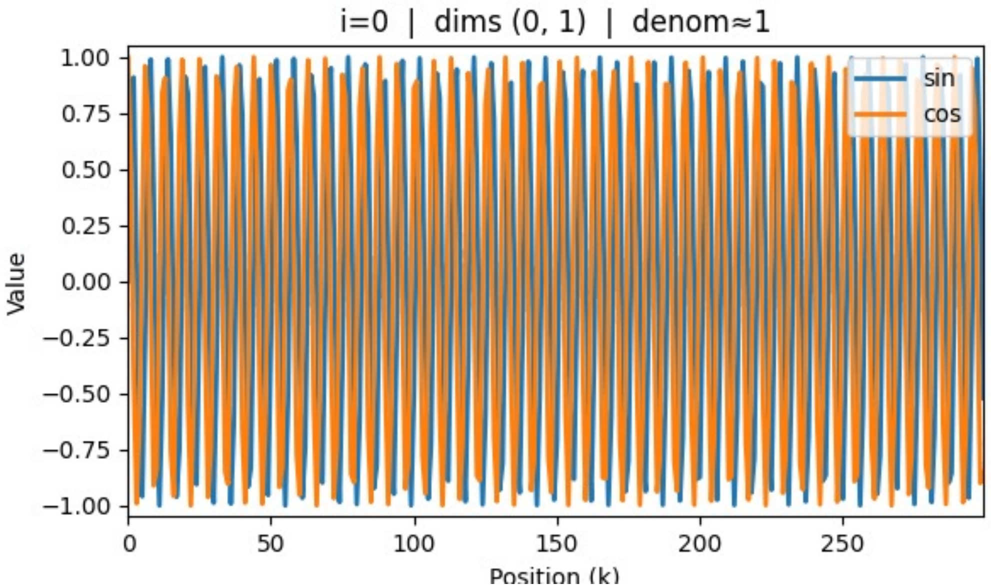
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \\ \vdots \\ \\ \sin\left(\omega_{\frac{d}{2}} t\right) \\ \cos\left(\omega_{\frac{d}{2}} t\right) \end{bmatrix}_{d\times 1}$$

Positional Encoding

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

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

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



Positional Encoding Matrix with d=4, n=100

Sequence	Index of token, k	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	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

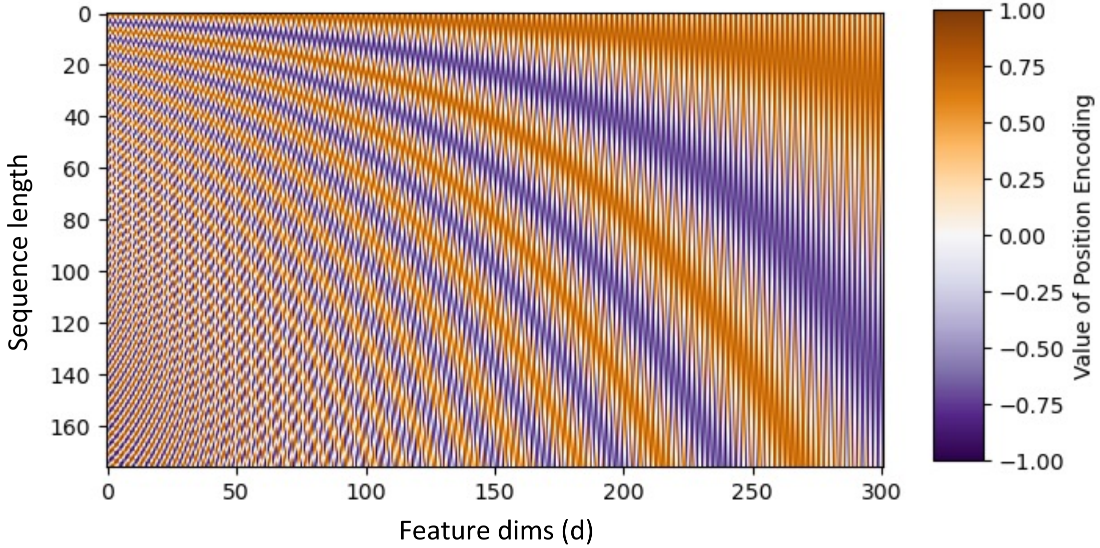
Positional Encoding Matrix for the sequence 'I am a robot'

Positional Encoding

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

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

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



- Let $\mathbf{x} = [x_1, \ldots, x_T] \in \mathbb{R}^{d \times T}$ 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. However, the encoder is not an RNN.
- The main components of an **Encoder Block** are:
  - Multi-Head Attention
  - Layer Normalization
  - Feedforward Neural Networks
  - Skip Connections
- Let’s break down the Multi-Head Attention Model!

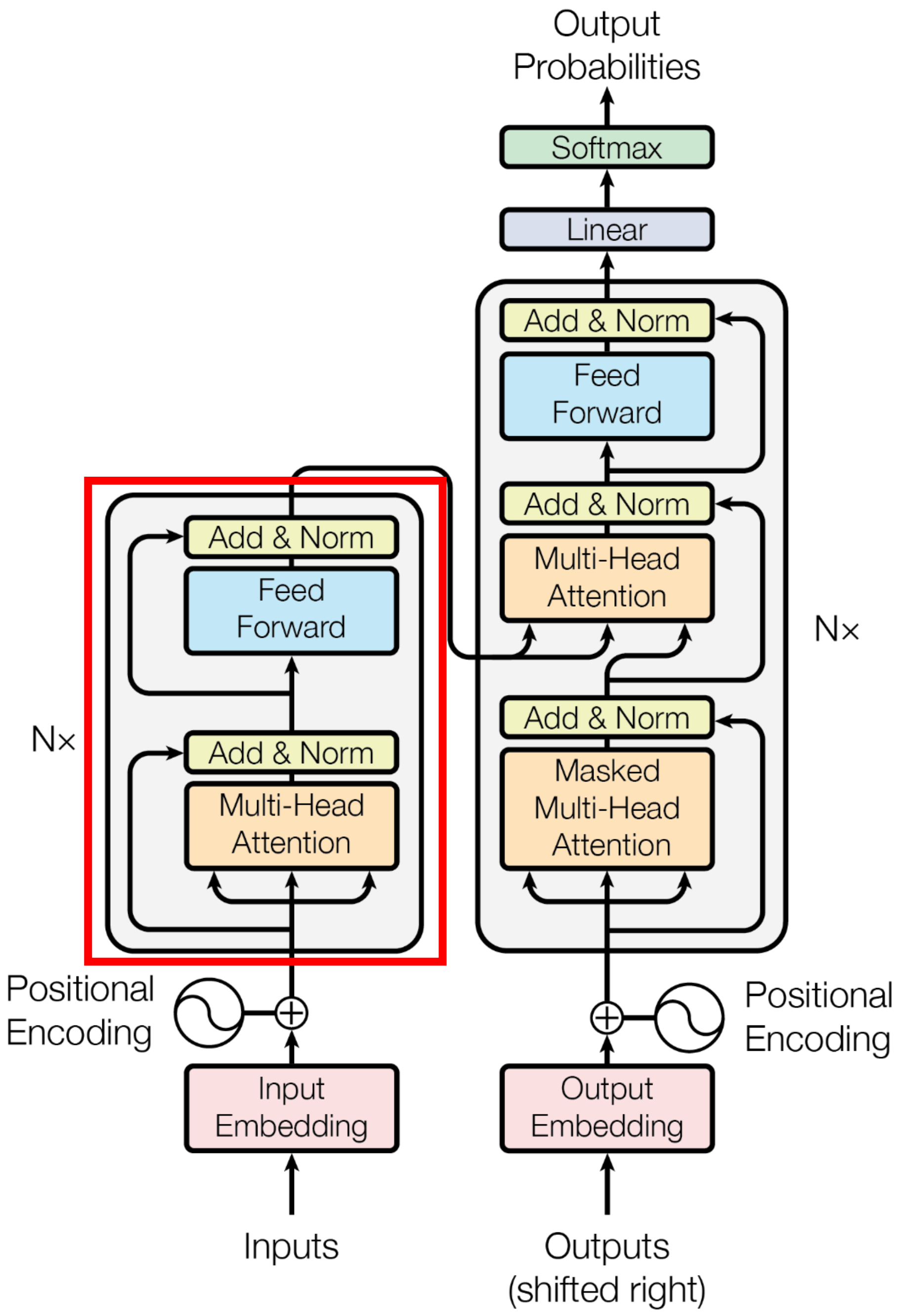
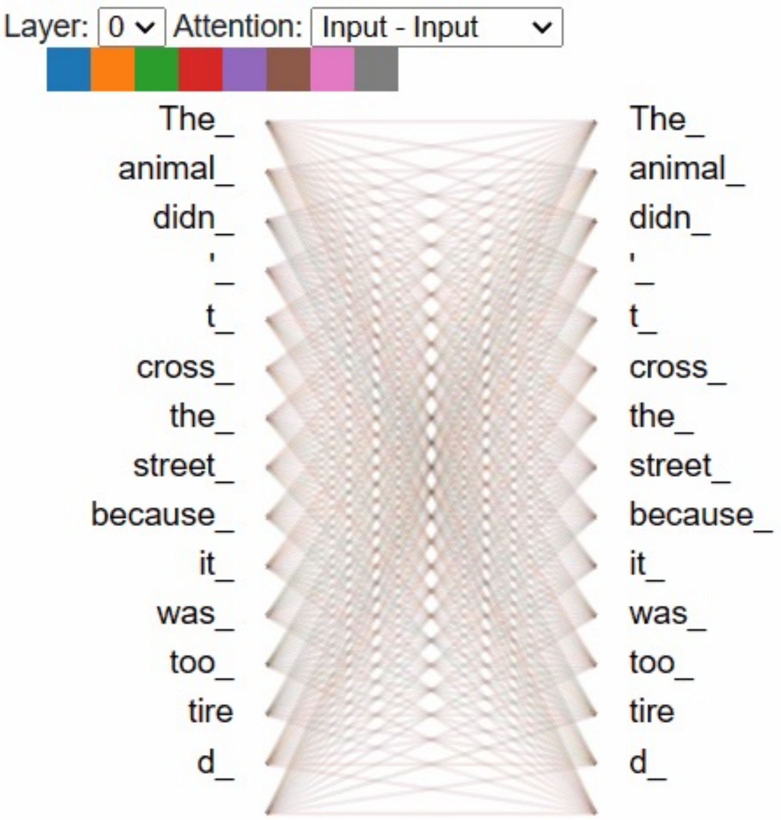


Figure 1: The Transformer - model architecture.

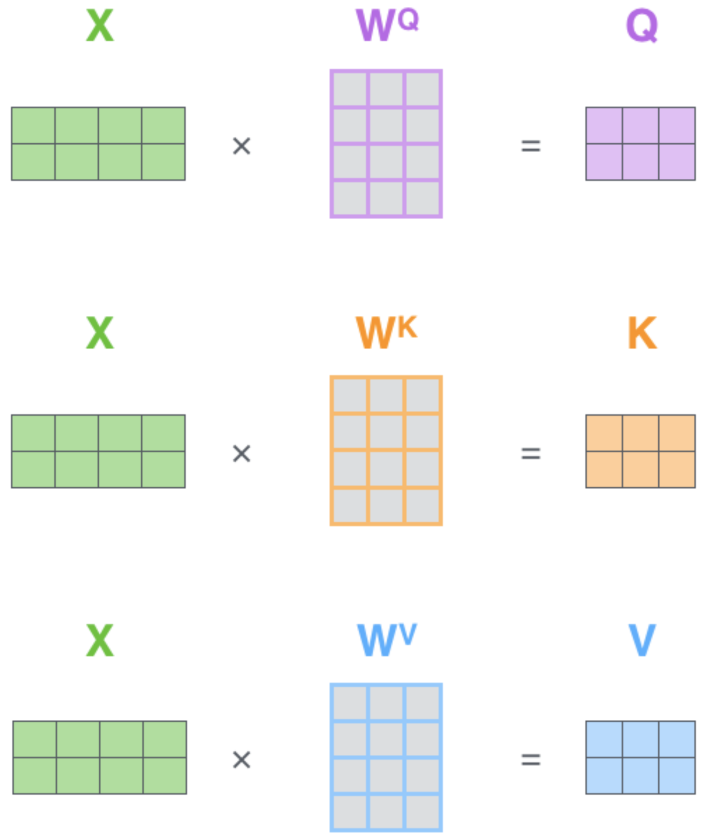
Self-Attention

- Self-attention focuses on important parts of the input by weighing the relevance of each token to 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.

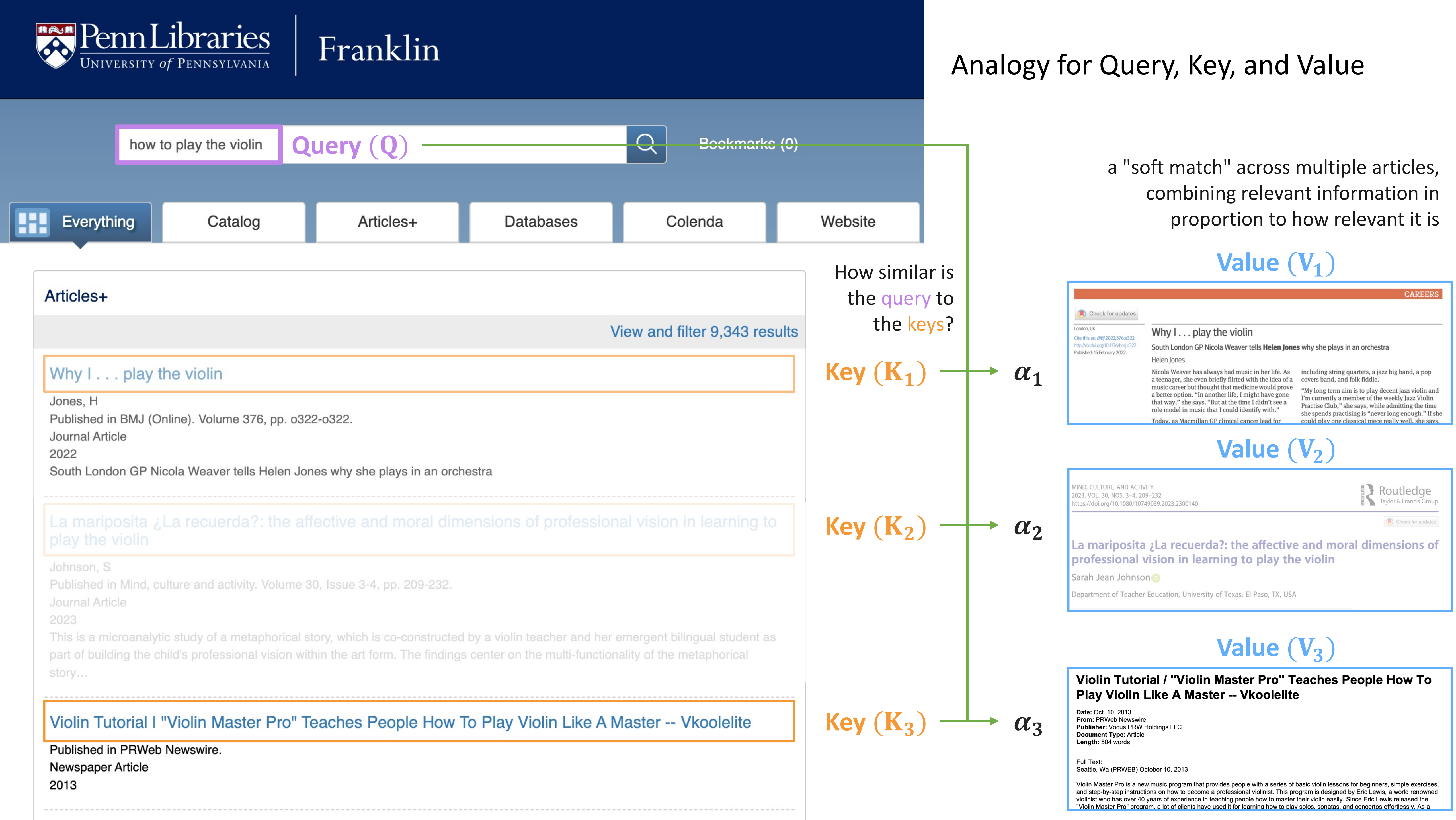


Self-Attention

- 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.)

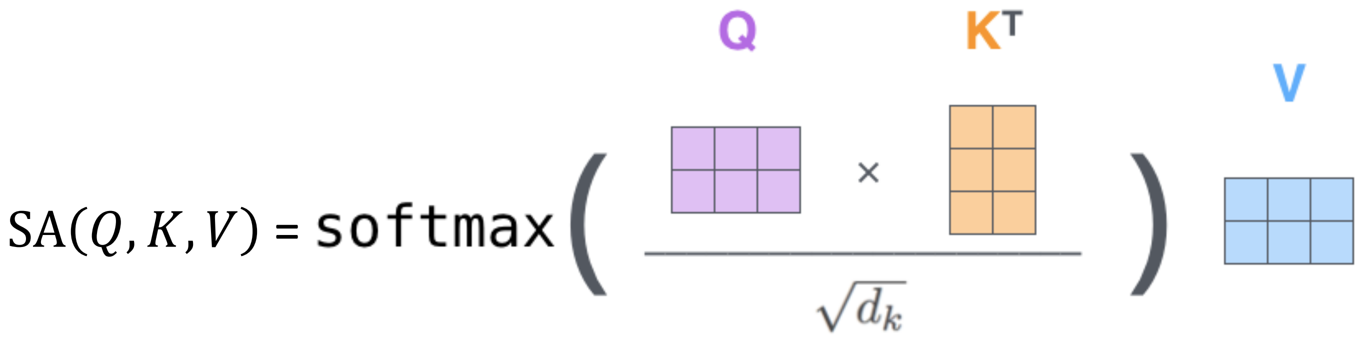


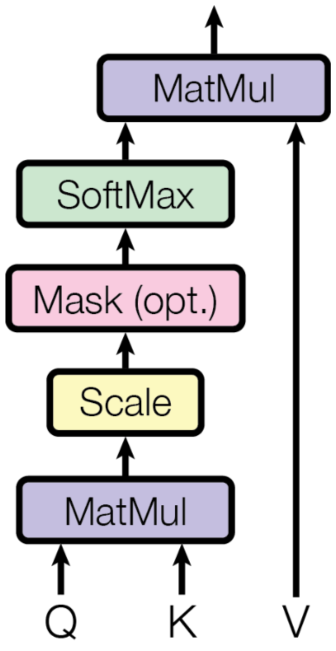
Analogy for Query, Key, and Value



Self-Attention

- Calculate the attention score by taking the dot product of $\mathbf{Q}$ and $\mathbf{K}^\top$.
- 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 $\mathbf{x} = [x_1, \dots, x_T]$

Multi-Head Self-Attention (MSA)

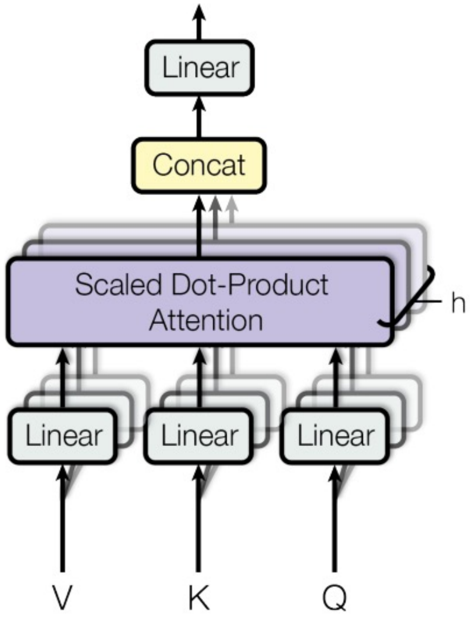
- **Multi-head Self Attention (MSA)** extends Self-Attention by introducing multiple independent attention heads, each focusing on different types of relationships.

- Each head has its own set of weight matrices:

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

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

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 $\mathbf{W}_O \in \mathbb{R}^{(h\cdot d_h)\times d}$ is the weighting matrix for fusing all attention heads.



Residual Connection & Layer Normalization

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

$$Output = LN(\mathbf{x} + SubLayer(\mathbf{x}))$$

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

- **LayerNorm:** normalizes x_i across feature dimension

$$LN(x_i) = \gamma \cdot \frac{x_i - \mu_i}{\sigma_i} + \beta$$

  - Parameters $\gamma, \beta \in \mathbb{R}^d$ are learned for each layer, and

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

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

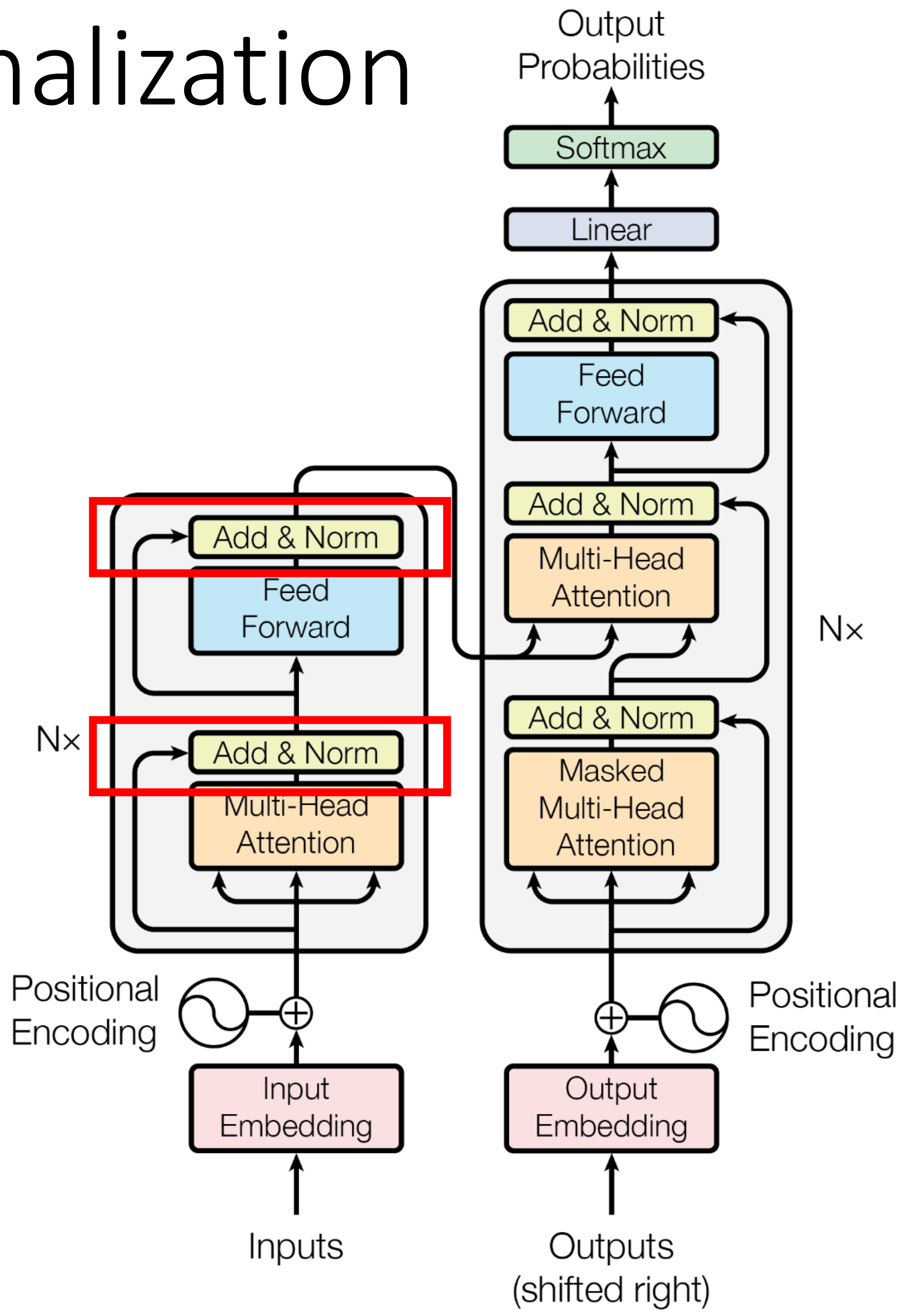


Figure 1: The Transformer - model architecture.

Encoder Block Summarized

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

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

  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**

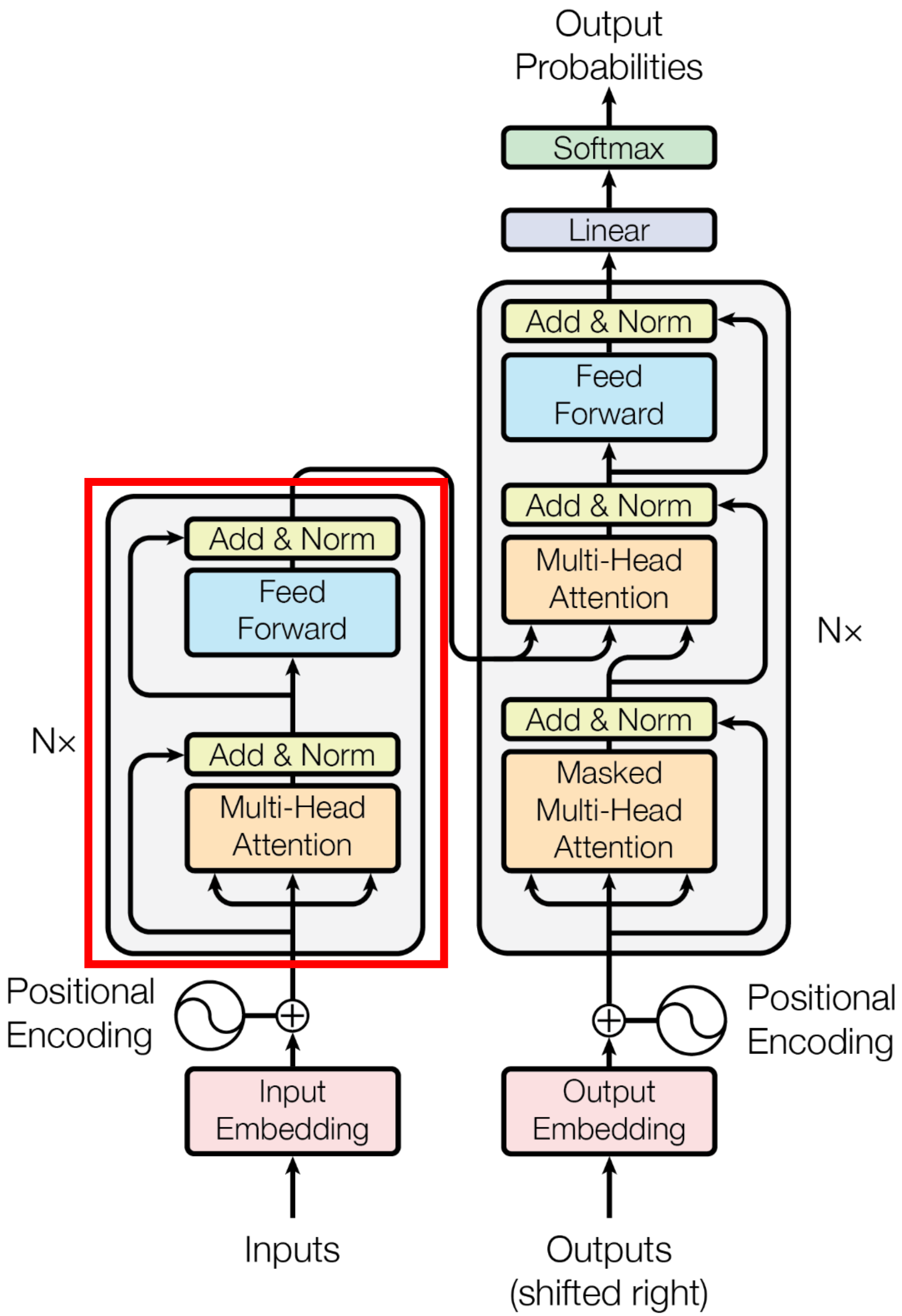
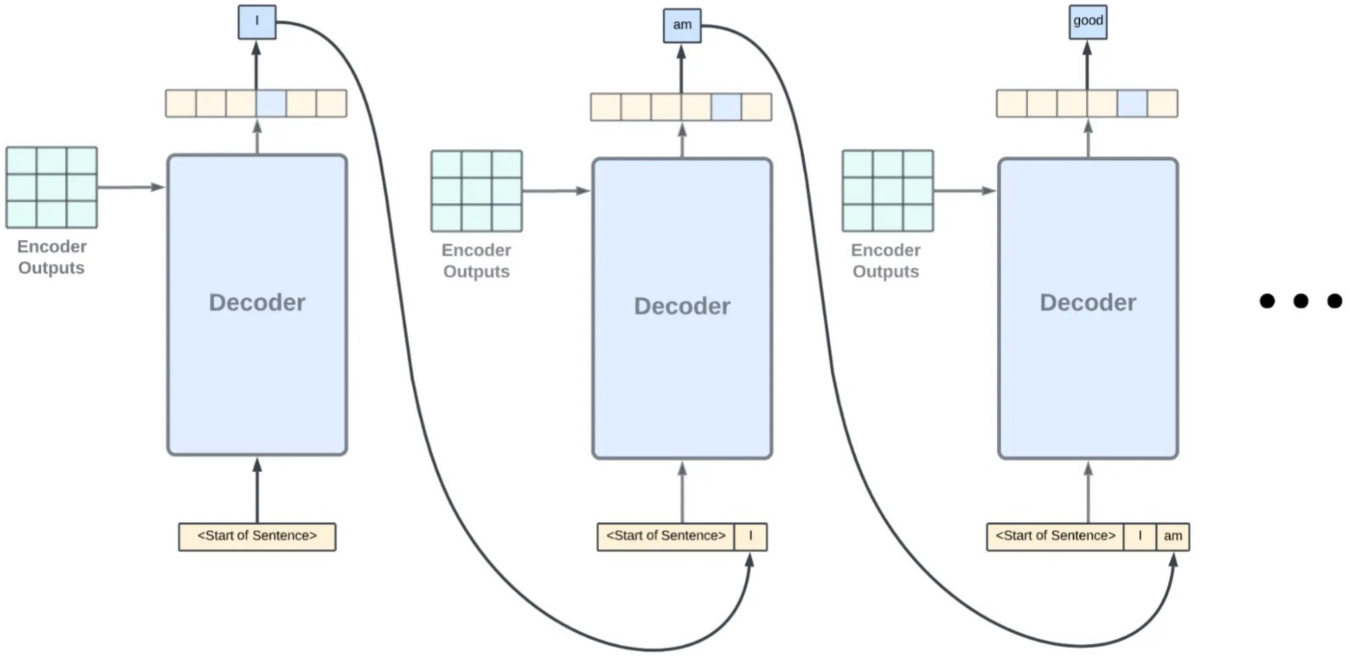


Figure 1: The Transformer - model architecture.

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

- 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.*

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

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

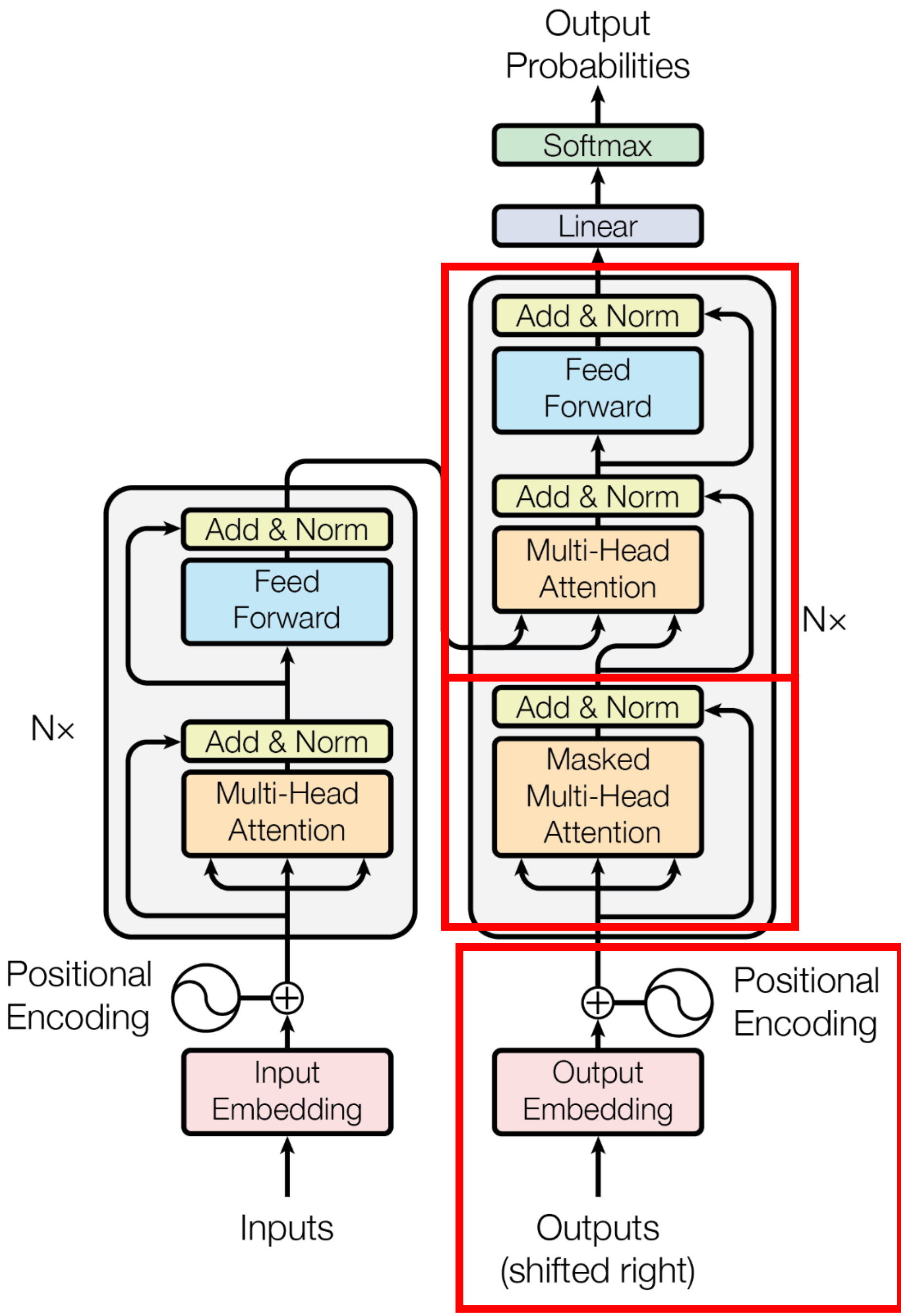
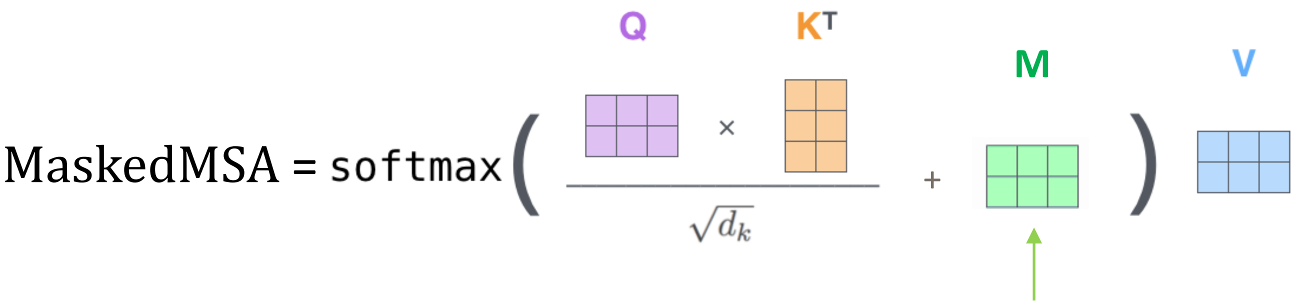


Figure 1: The Transformer - model architecture.

Decoder Block: Masked MSA

- Just liked 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.



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

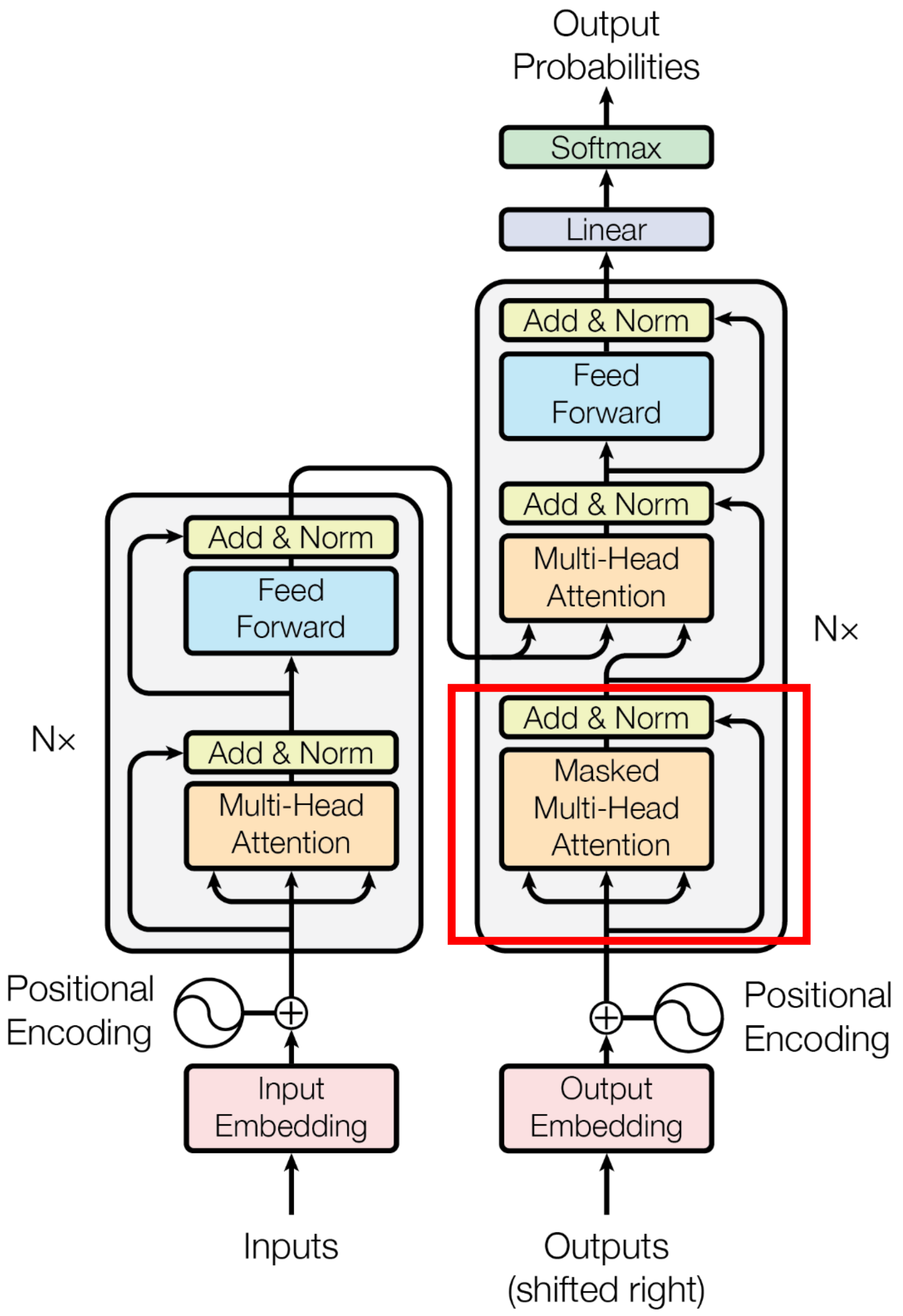


Figure 1: The Transformer - model architecture.

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, but the *queries*, *keys*, and *values* come from different sources.
  - *Key* and *value* matrices come from the output $\mathbf{x}^{(N)}$ of the encoder, and *query* matrix from the previous MSA.

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

  - This allows the decoder to focus on relevant part of encoded input

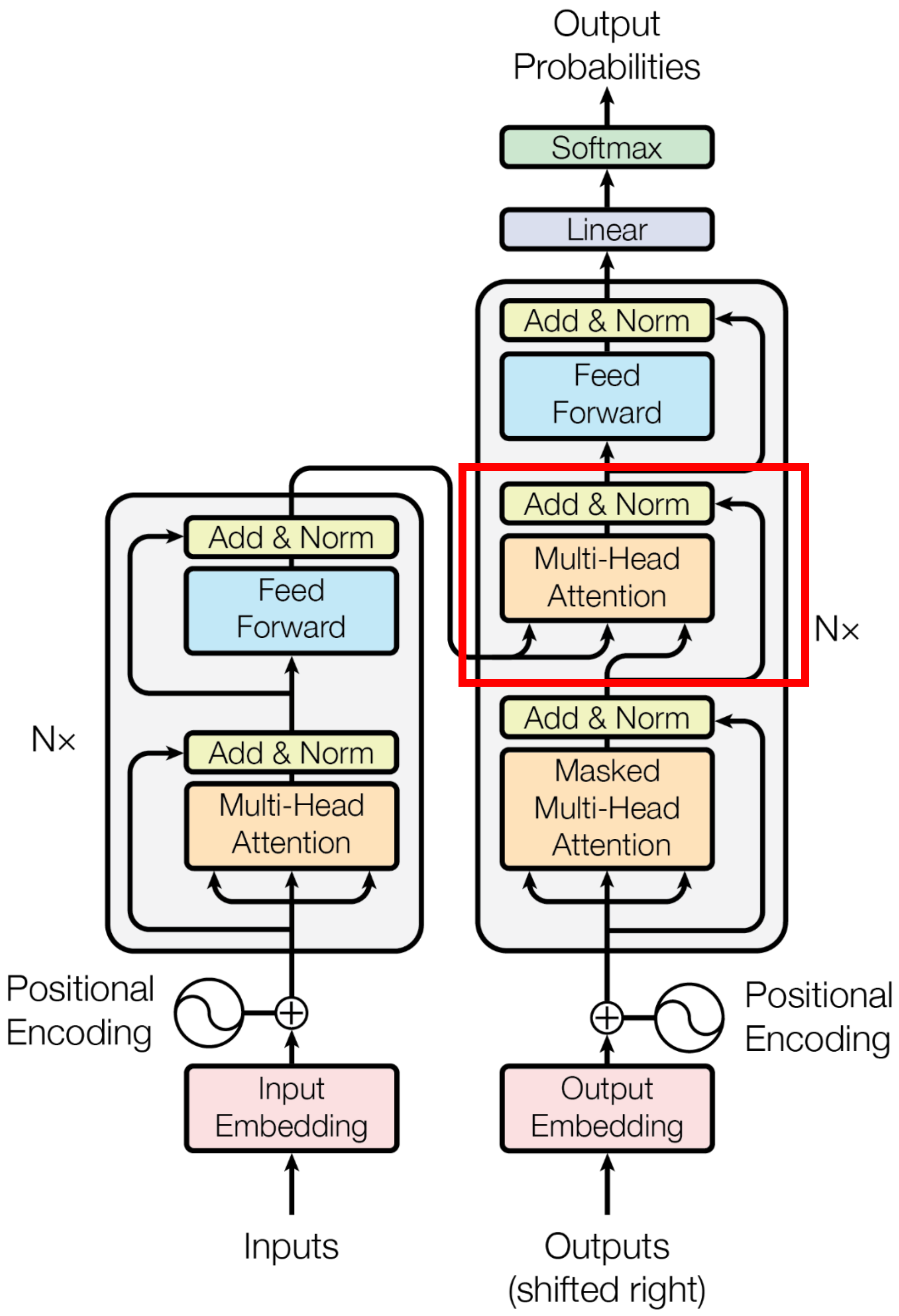


Figure 1: The Transformer - model architecture.

Decoder Block: Summarized

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

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

- After the decoder, the output is projected via a linear layer and softmax to obtain probabilities.
  - The final decoder output $\mathbf{y}^{(N)} \in \mathbb{R}^{T\times d}$ is mapped to vocabulary logits:

$$\mathbf{z} = \mathbf{y}^{(N)}\mathbf{W}_L + \mathbf{b}_L, \qquad \mathbf{W}_L \in \mathbb{R}^{d\times V}$$

  - Finally, the model predicts the probability distribution:

$$\mathbf{p} = \mathrm{softmax}(\mathbf{z})$$

    - Each row $\mathbf{p}_t \in \mathbb{R}^V$ represents model's predicted probability over the vocabulary for the next word: $\mathbf{p}_\mathrm{t} = P_\theta(y_t | y_{<t}, \mathbf{x})$.

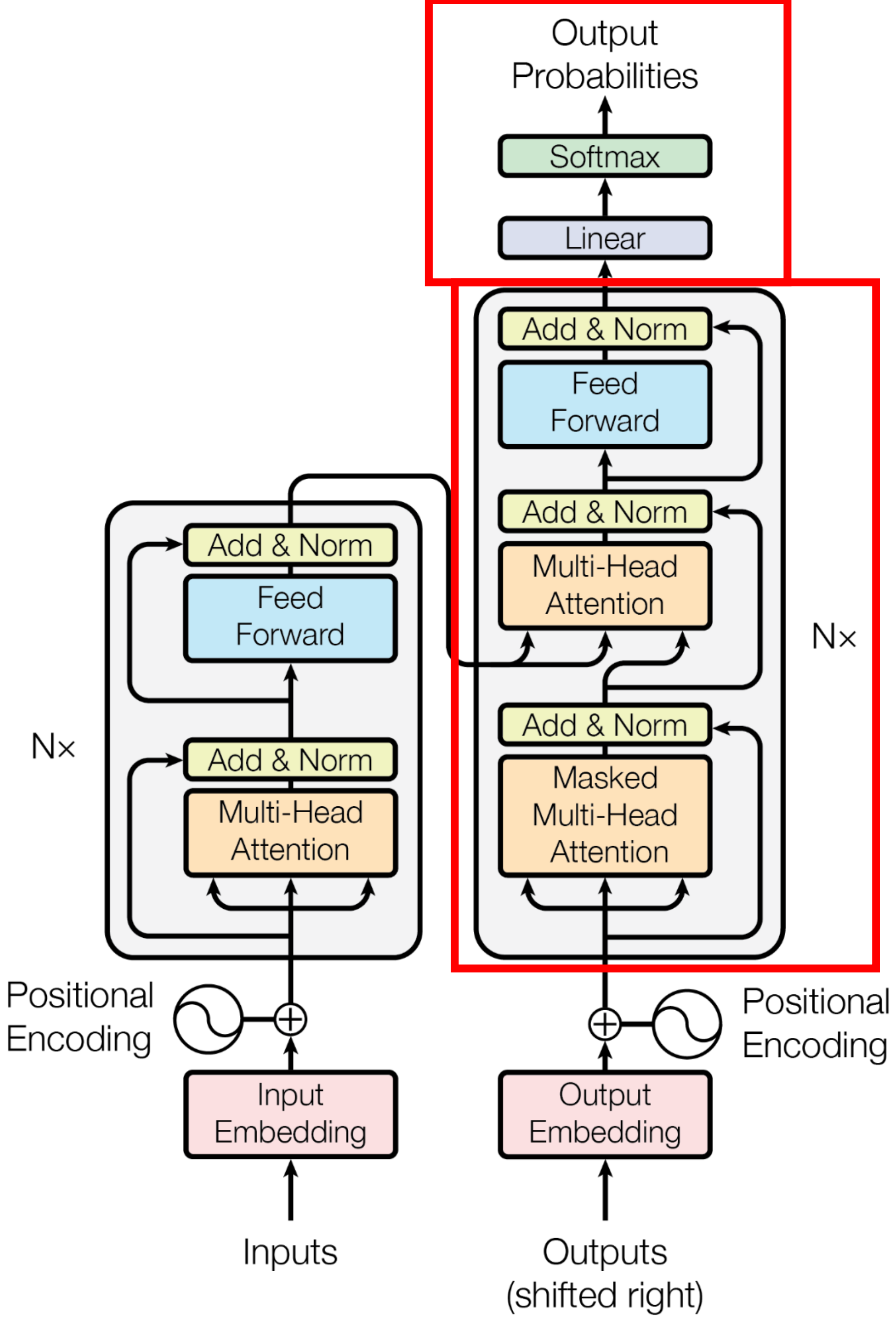


Figure 1: The Transformer - model architecture.

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:

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

$P_\theta(y_t|y_{<t}, x)$ 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

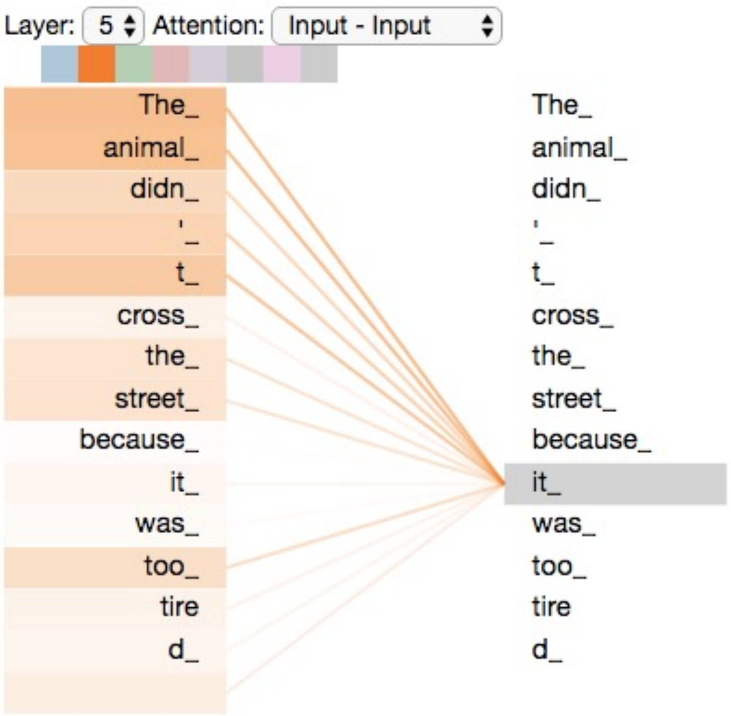
Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>

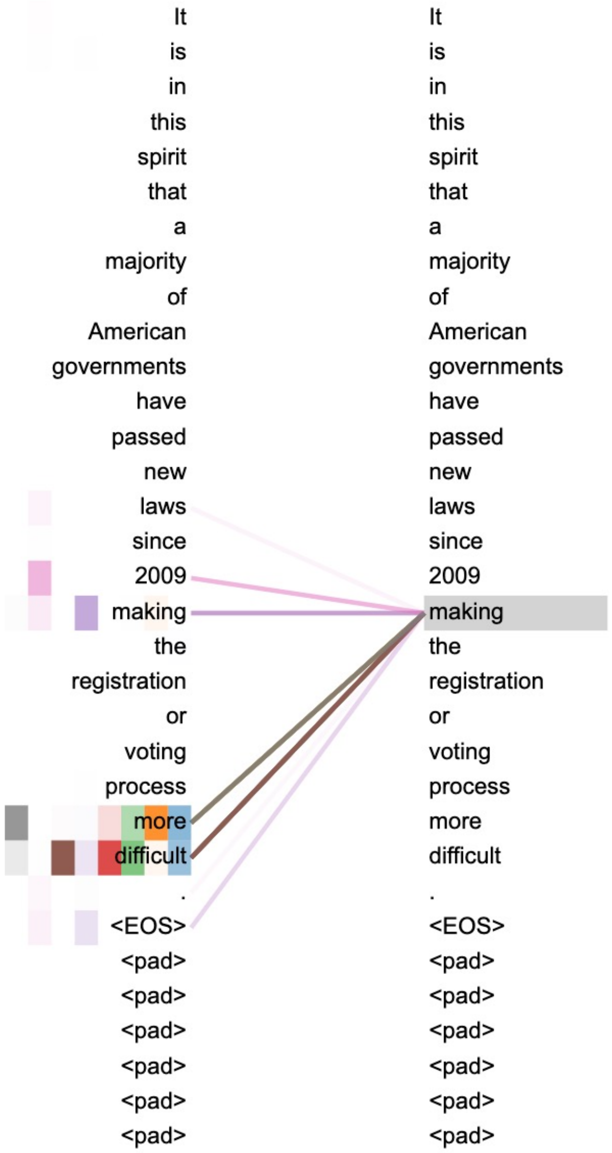
Trained Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98
	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”.

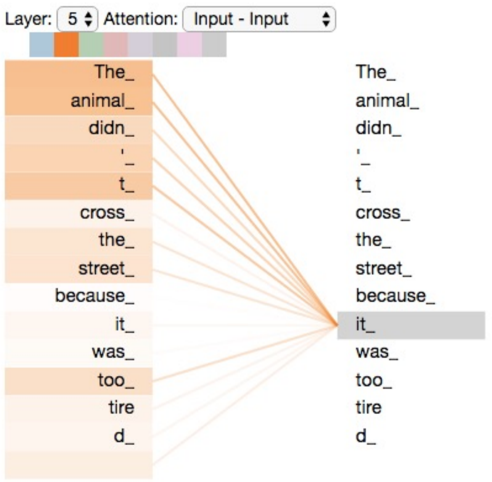


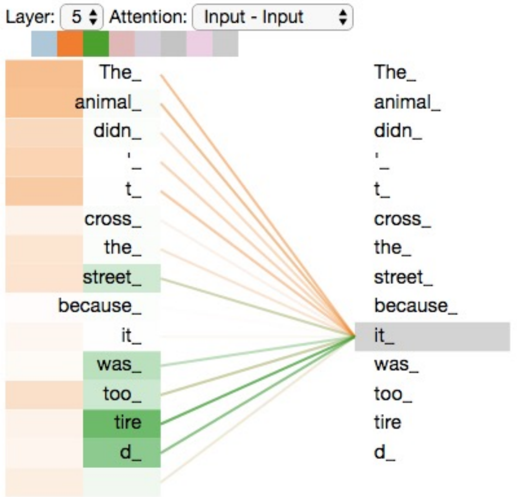


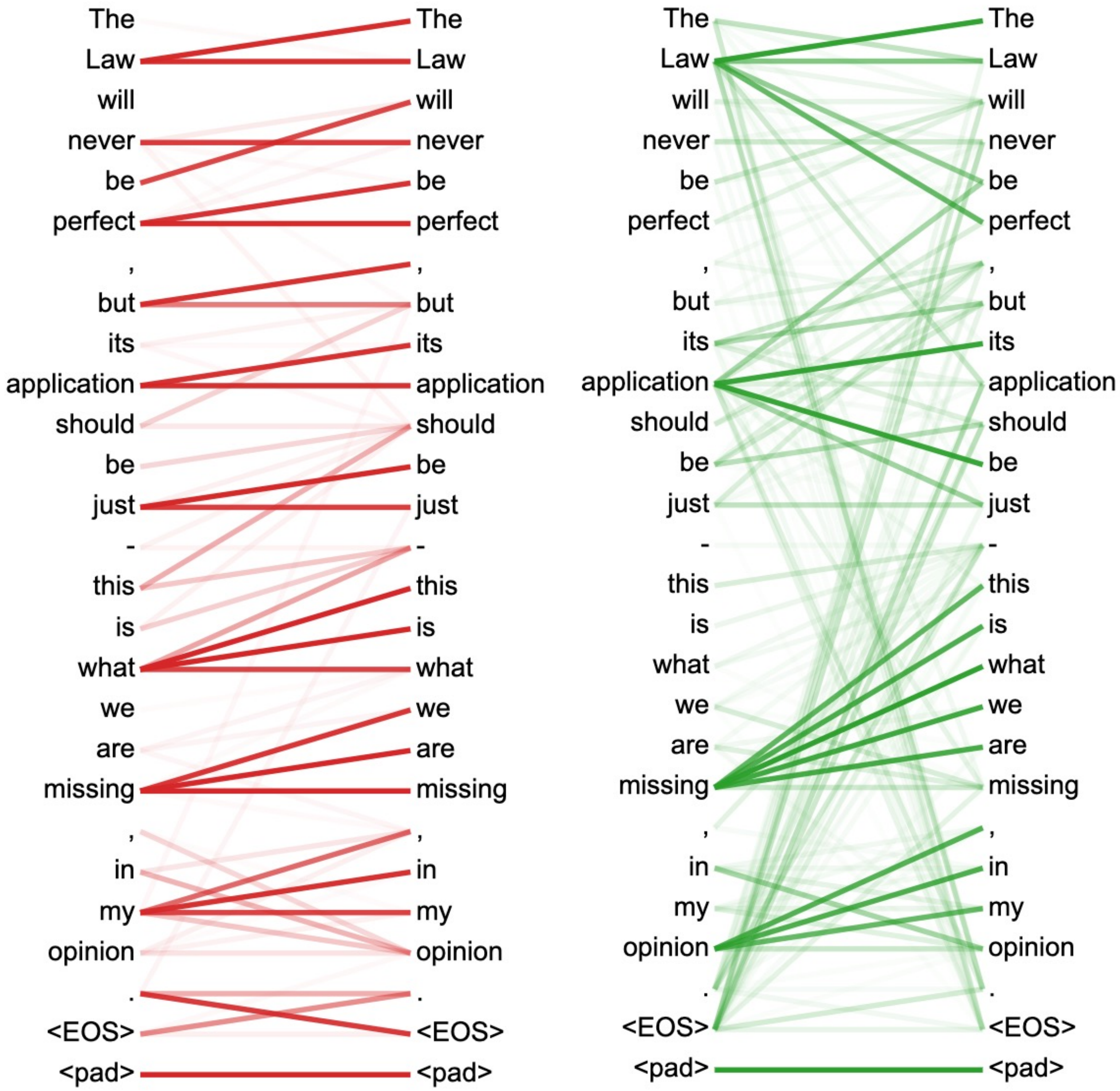
- On the right we see another visualization showing how different words in a longer sentence relate to each other.
- Check out this interactive visualization.

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.



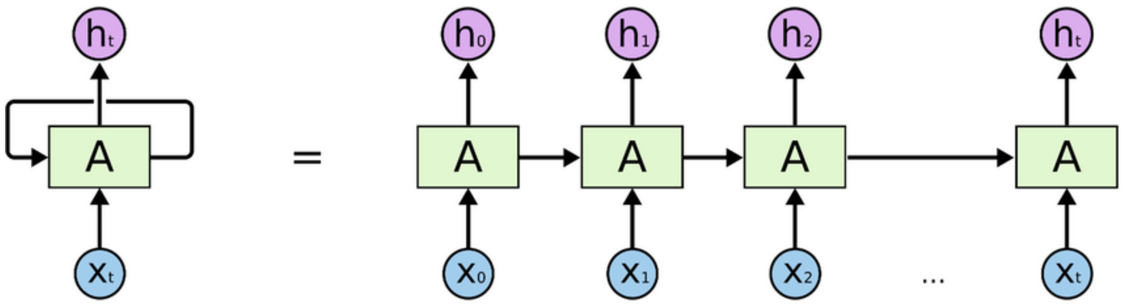




RNNs vs. Transformers

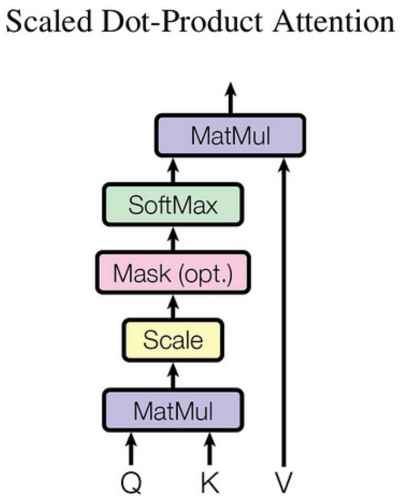
Recurrent Neural Network

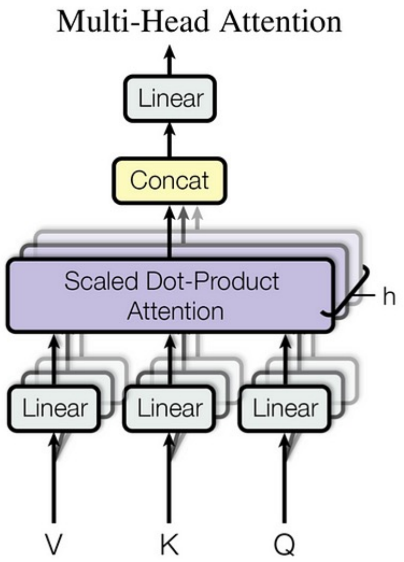
- Handle sequential data
- Learn sequential dependencies
- Each time step depends on the previous one



Transformers

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





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