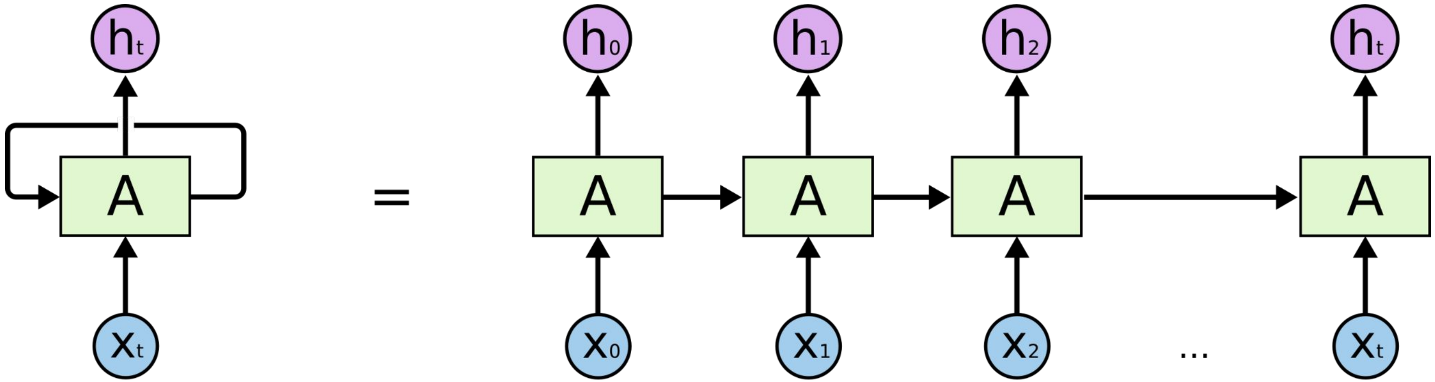
INFO-I590 Fundamentals and Applications of LLMs

Attention and Transformers

Jisun An

An unrolled Recurrent Neural Network (RNN)



Img: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Limitations of RNN Models

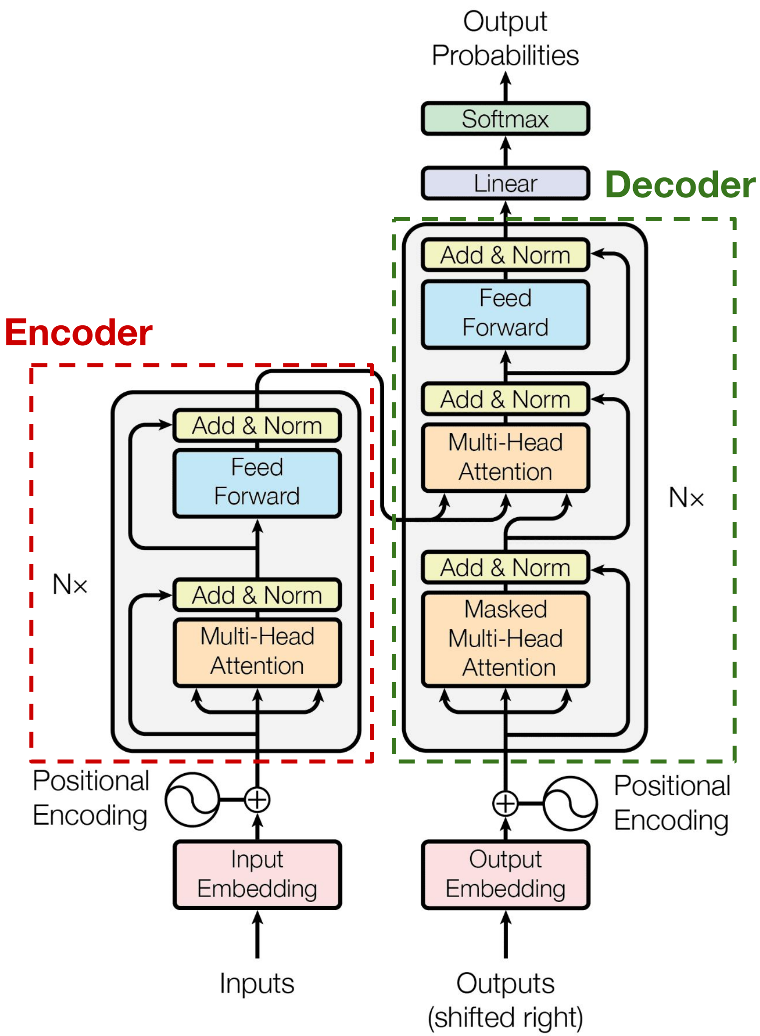
- RNN models reuse the output of the previous token as input, **making parallel processing impossible**.
  - Due to sequential processing, the training process is **inherently slower**.
- **Information from earlier tokens dissipates** as the input sequence grows longer, leading to performance degradation.
- Adding deeper layers to improve performance can result in:
  - **Gradient Vanishing**: Gradients shrink, stopping weight updates
  - **Gradient Exploding**: Gradients grow excessively, destabilizing training

Transformers

- Core Innovation: **Self-Attention**
  - Replaces sequential input processing
  - Calculates relationships between words to adjust their representations
- Advantages Over RNNs
  - Scalability: Deeper models train effectively using repeated blocks (layers)
  - Efficiency: Parallel computation shortens training time.
  - Handles Long Inputs: Maintains performance on long sequences
- Enabling "**Large**" Language Models (LLMs)
  - Transformers are the backbone of modern LLMs.

“Attention is All You Need”
(Vaswani et al. 2017)

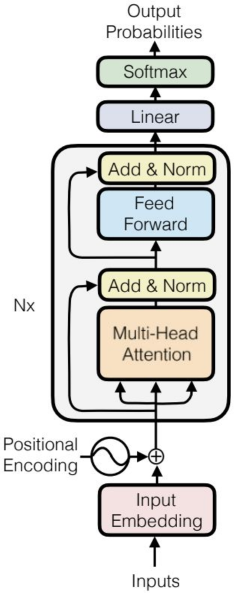
- A sequence-to-sequence model based entirely on attention
- Strong results on machine translation
- Fast: only matrix multiplications



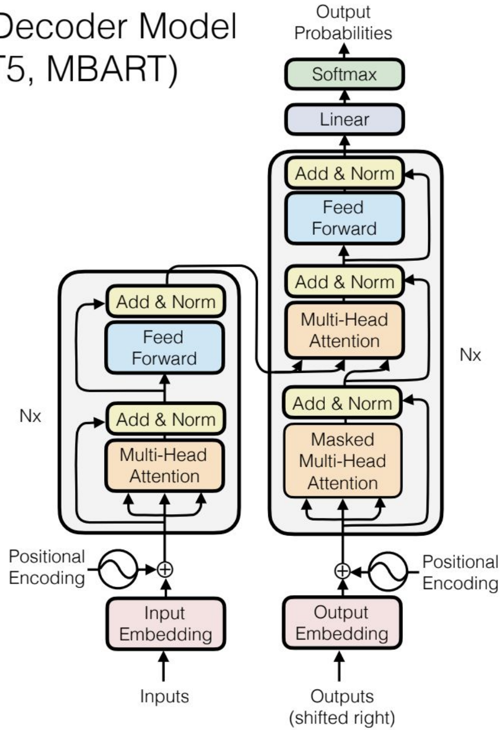
Vaswani et al. 2017. "Attention is all you need." Advances in Neural Information Processing Systems.
https://arxiv.org/abs/1706.03762

Three Types of Transformers

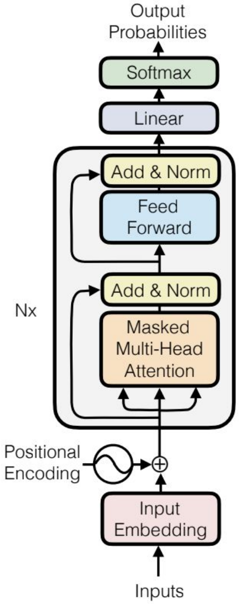
Encoder Only Model
(e.g. BERT)



Encoder-Decoder Model
(e.g. T5, MBART)



Decoder Only Model
(e.g. GPT, LLaMa)



Core Transformer Concepts

- Positional encoding
- Attention
- Multi-headed attention
- Masked attention
- Residual + layer normalization
- Feed-forward layer

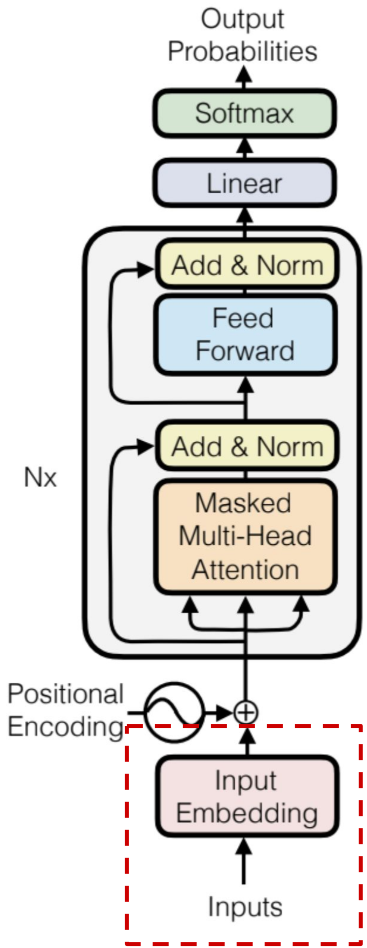
Positional Encoding

(Review) Inputs and Embeddings

- Inputs: Generally split using subwords

the books were improved
the book _s *were* ***improv _ed***

- Input Embedding: Looked up, like in previously discussed models



Positional Encoding

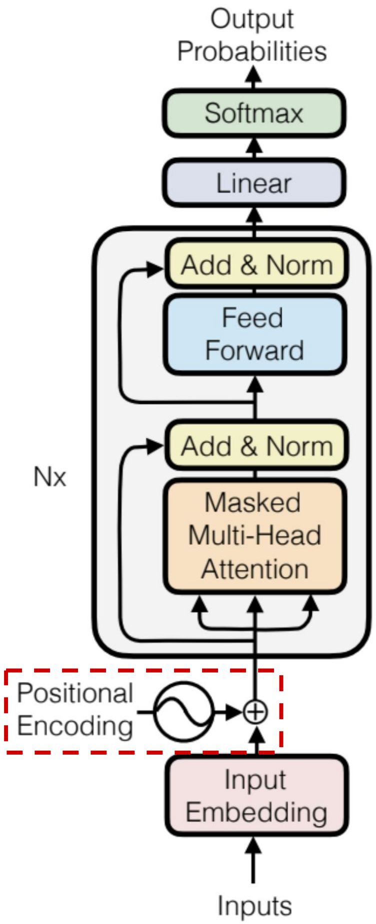
- Transformer: Processes all inputs simultaneously, losing order information. But, **order is critical in text.**
- If embeddings were used, there would be no way to distinguish between identical words

*A **big** dog and a **big** cat*

would be identical!

- Positional encodings add an embedding based on the word position

$W_{big} + W_{pos2}$ $\qquad$ $W_{big} + W_{pos6}$



Sinusoidal Encoding (Vaswani+ 2017, Kazemnejad 2019)

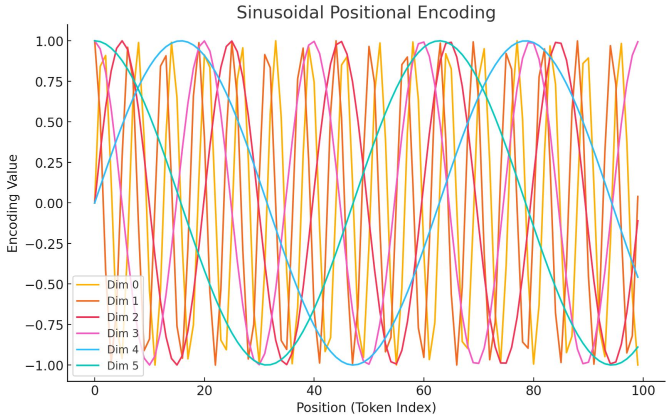
- Calculate each dimension with a sinusoidal function

Even dimensions ($2i$):

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

Odd dimensions ($2i+1$):

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



- Why? So the doc product between two embeddings becomes higher relatively.

Learned Encoding (Shaw+ 2018)

- More simply, just create a learnable embedding

- Advantages: flexibility
- Disadvantages: impossible to extrapolate to longer sequences

Absolute vs. Relative Encodings

- **Absolute** positional encodings add an encoding to the input in *hope* that relative position will be captured
- **Relative** positional encodings *explicitly* encode relative position

1 2 3 4 5 6 7 8
The black cat eats food and drinks water.

Yesterday morning, I saw a black cat.
1 2 3 4 5 6 7

Rotary Positional Encodings (RoPE) (Su+ 2021)

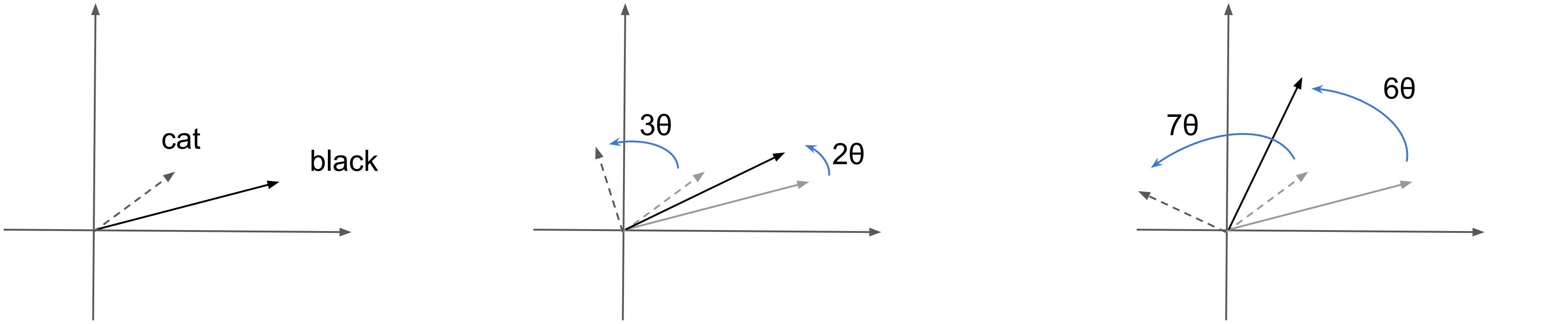
- Fundamental idea: we want the dot product of embeddings to result in a function of relative position

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m - n)$$

- In sum, RoPE uses trigonometry and imaginary numbers to come up with a function that satisfies this property.

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

RoPE Intuitions



The black cat eats food and drinks water.

Yesterday morning, I saw a black cat.

Attention

Attention: Mimicking Human Context Processing

- Consider this sentence

  "*Oo ooo oo ooo ooooo bank.*"

  - It's impossible to determine whether "bank" refers to financial institution or land near a river because the context is hidden.
- Now, consider this sentence:

  "*He sat by the river bank.*"

  - Here, "bank" clearly refers to land near a river due to the surrounding words.
- Humans don’t interpret words in isolation. We **derive meaning** from **surrounding words**.

Attention: Basic Idea

- The goal of attention is to
  1) determine which words to 'attend' to accurately interpret a word within its context
  , and 2) reinterpret the word based on its context.

"He **sat by** the **river** *bank*."

[0.10, 0.75, -0.22, ..., 0.80]

"She **deposited money** in the *bank*."

[-0.55, 0.20, 0.90, ..., -0.10]

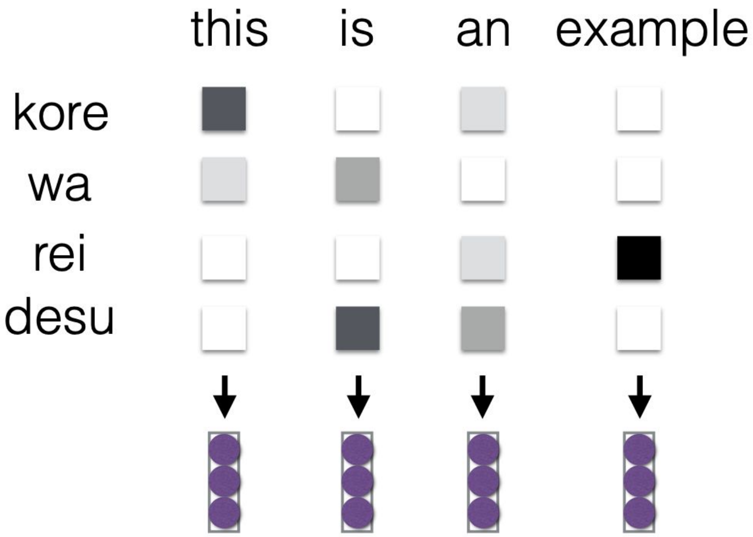
Attention: Basic Idea

- The goal of attention is to
  1) determine which words to ‘attend’ to accurately interpret a word within its context
  , and 2) reinterpret the word based on its context.

- To computes relationships between words
  - Uses Query (Q) & Key (K) to assign weights (higher for important words).
- To reinterprets words based on context
  - Multiplies weights with Value (V) to refine meaning.

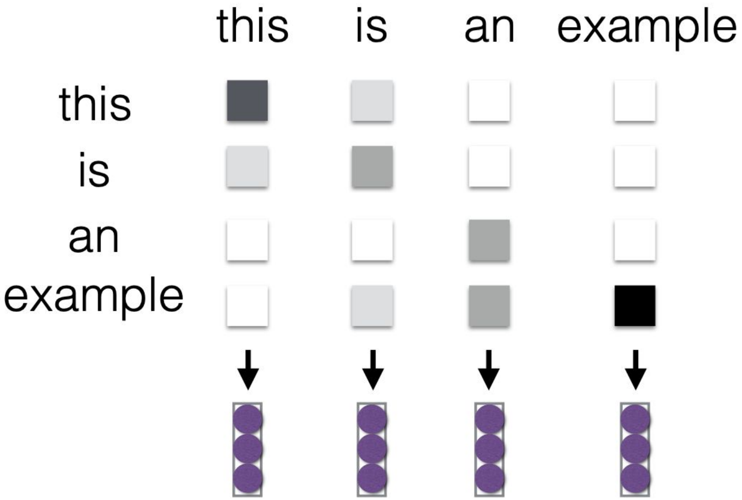
Cross Attention (Bahdanau et al. 2015)

- Each element in a sequence attends to elements of another sequence



Self Attention (Cheng et al. 2016, Vaswani et al. 2017)

- Each element in the sequence attends to elements of that sequence
  → context sensitive encodings!



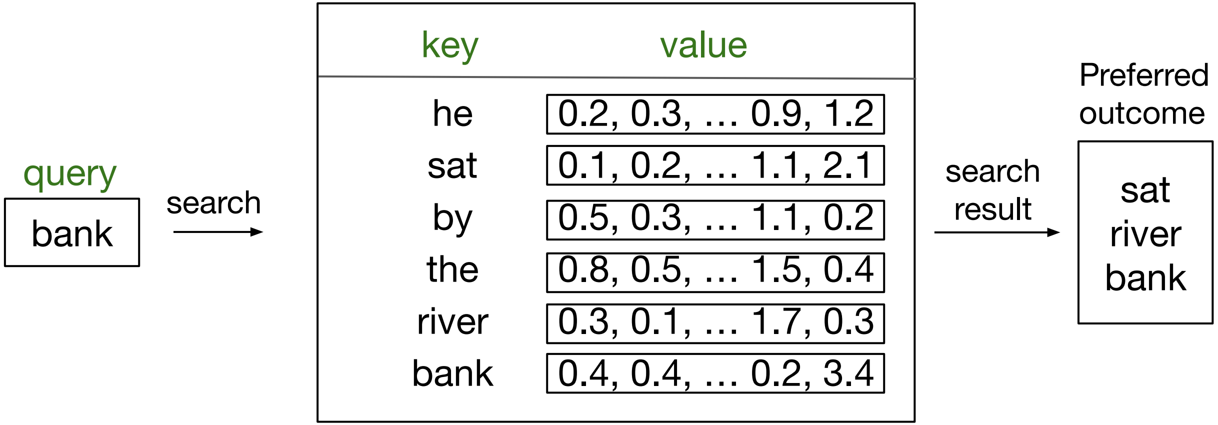
Query, Key, Value: The Core of Attention

The transformer architecture implements the way we process text using:

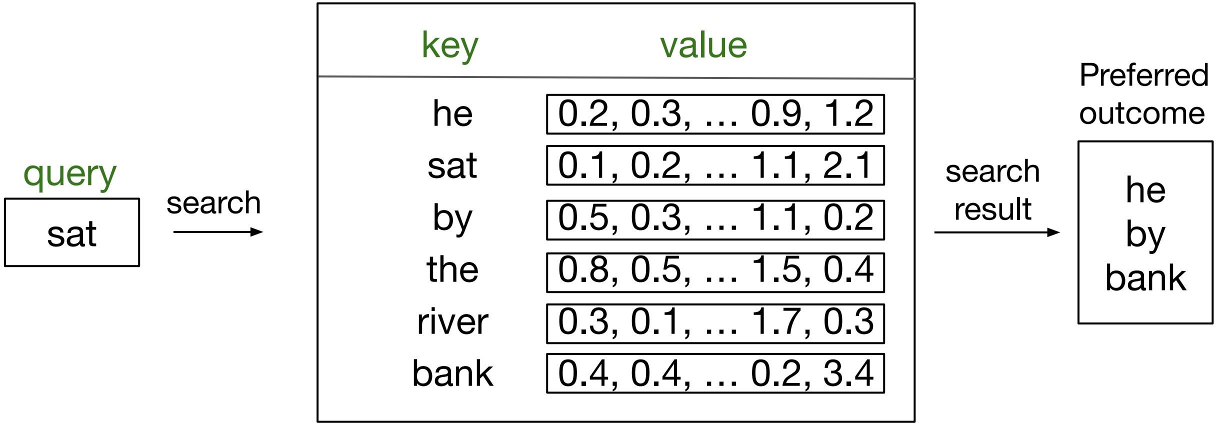
- Query (Q): The focus or "search term"
- Key (K): Features of the data used for relevance matching
- Value (V): The actual information retrieved

- Introduced to handle relationships between words efficiently.
- Terms from Information Retrieval

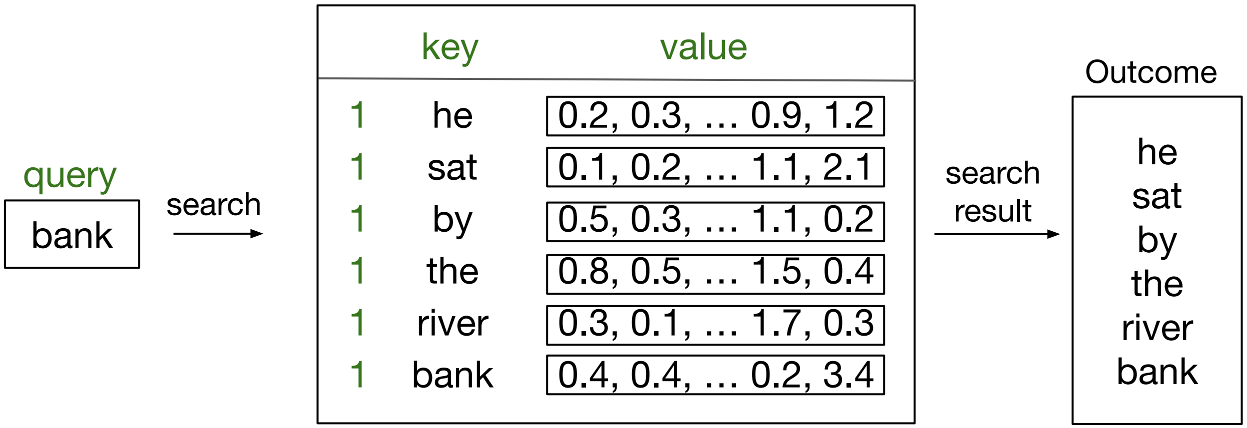
Query, Key, Value: Example



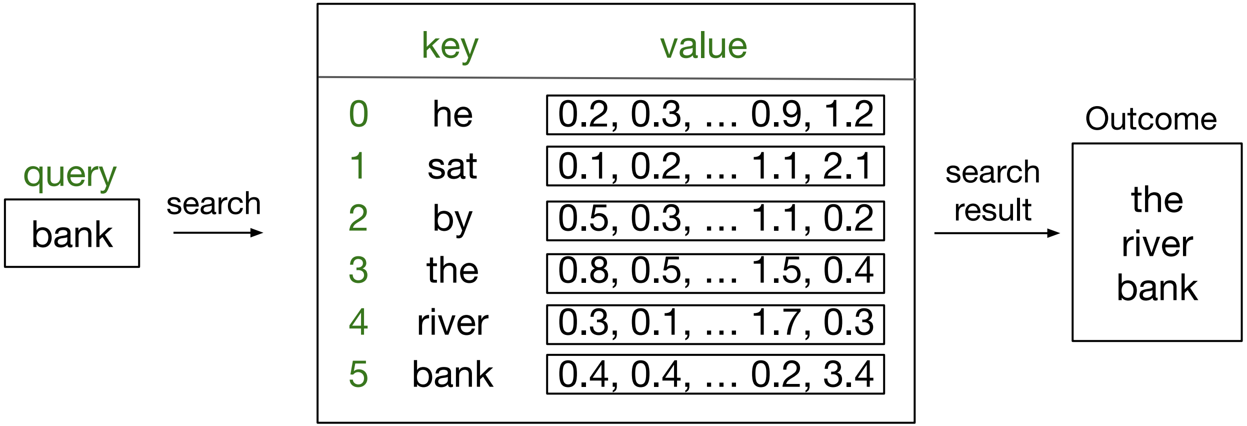
Query, Key, Value: Example



If we evenly reflect surrounding context

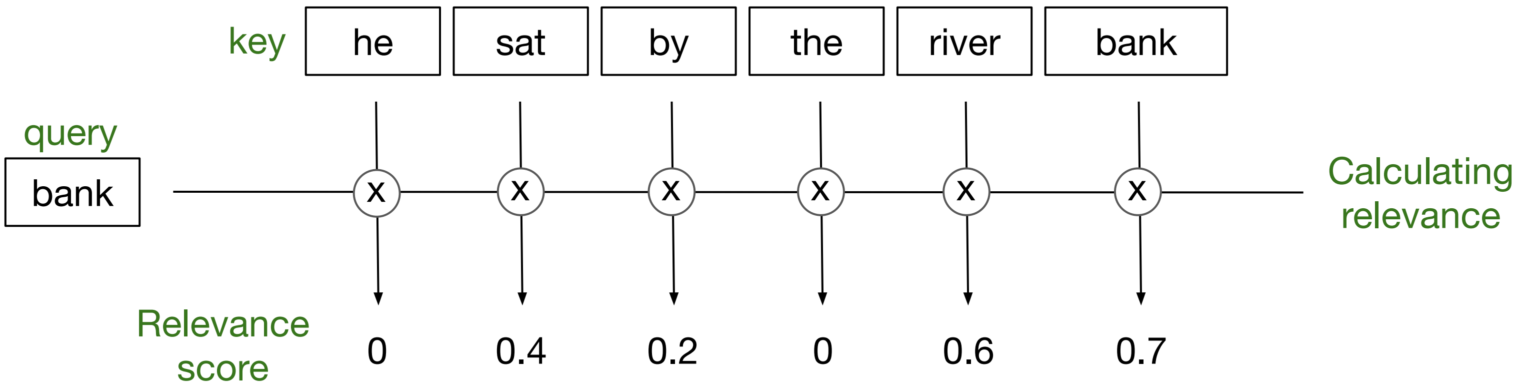


If we reflects surrounding context based on distance

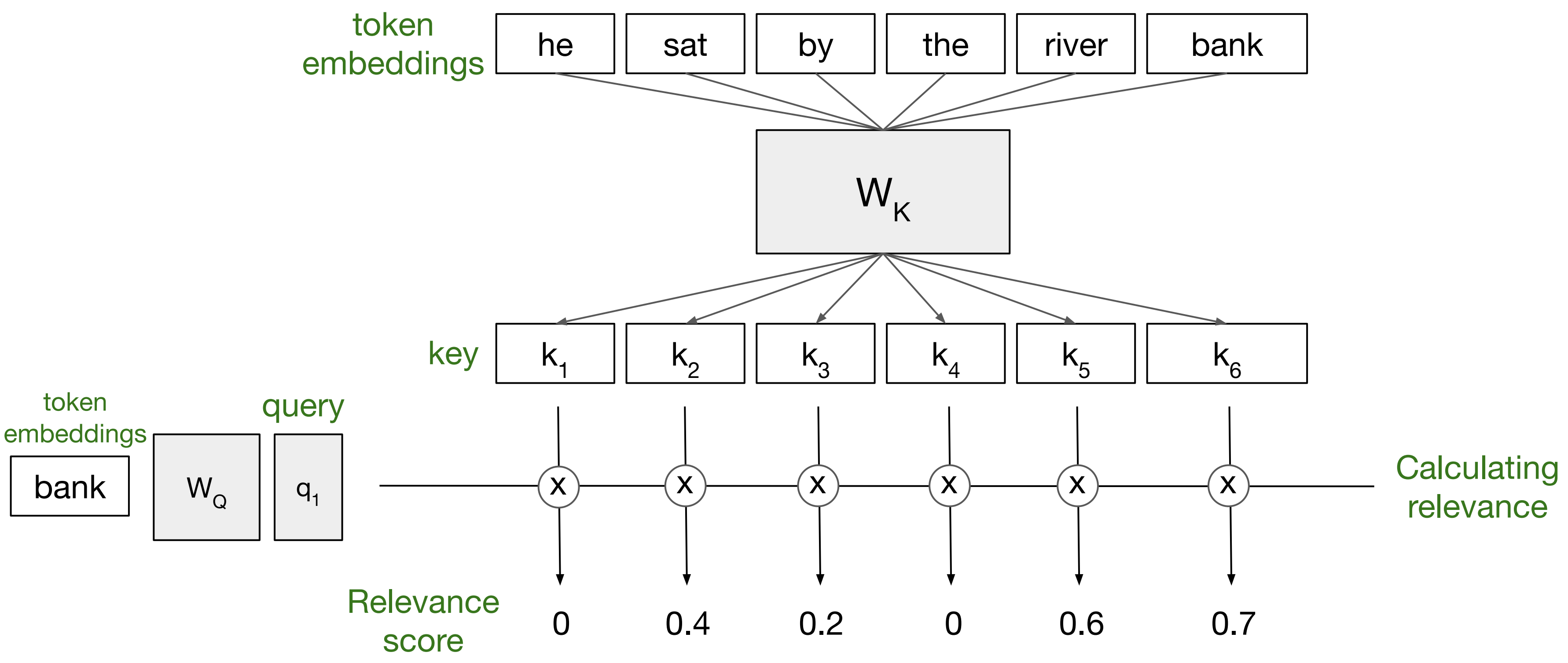


Calculating 'Attention'

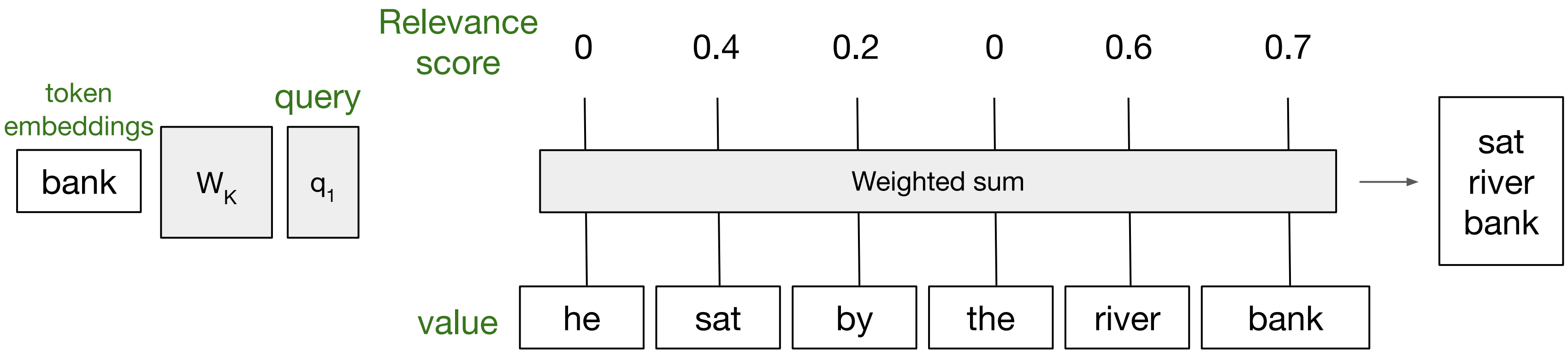
- Relevance must be calculated from the data itself, not determined by fixed rules



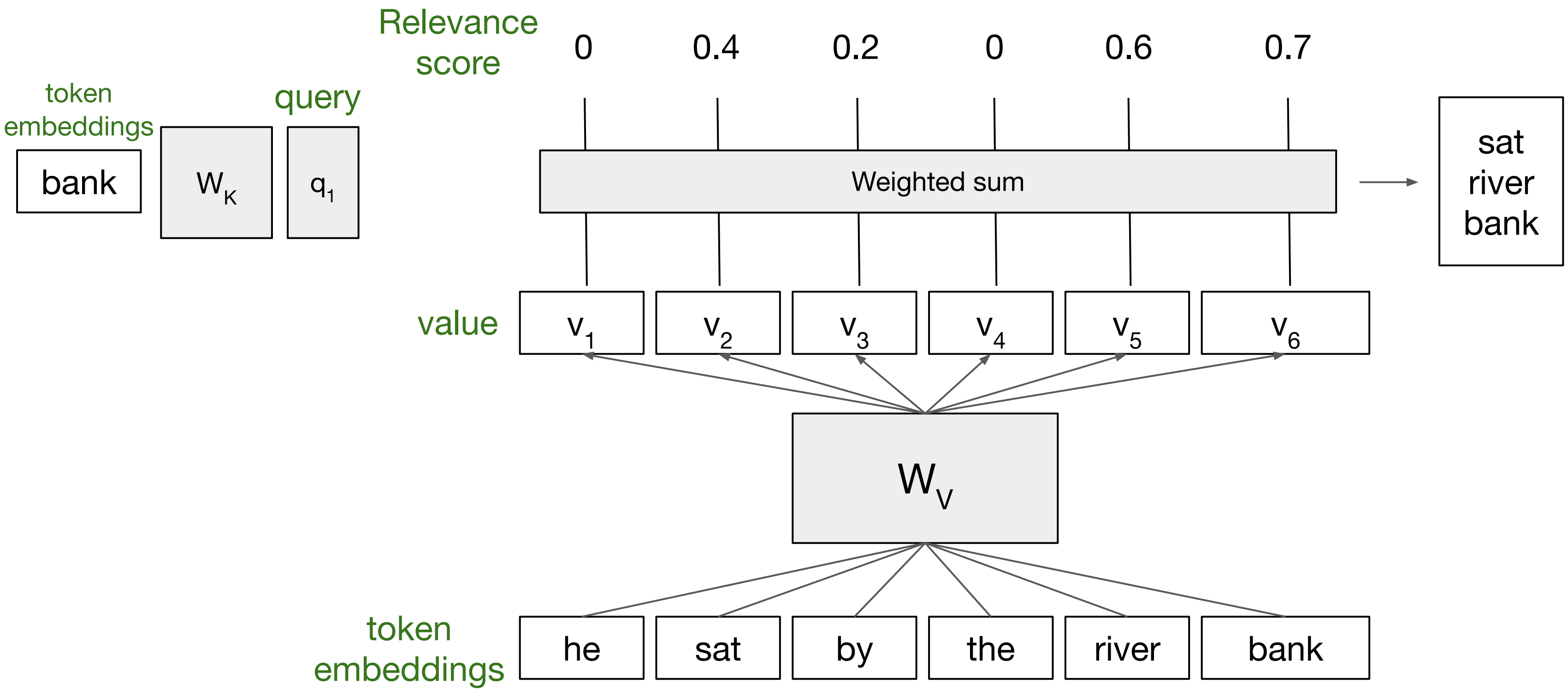
Calculating ‘Attention’ with weights



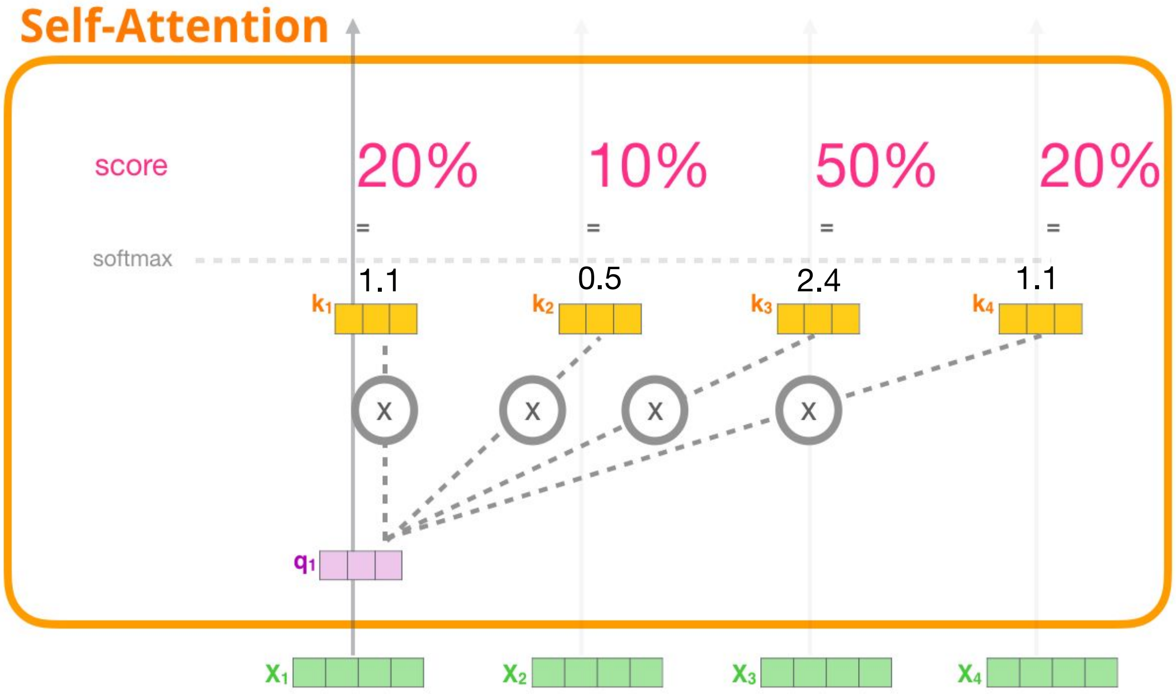
Combining Attention and Value



Calculating Value with weights

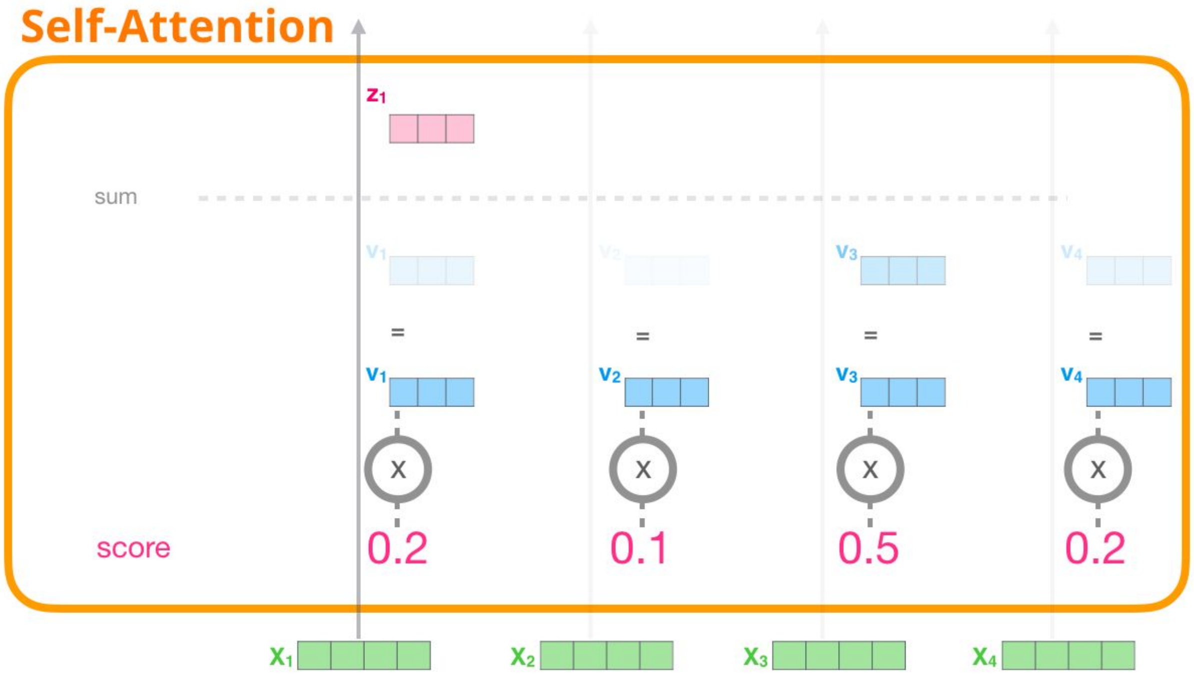


Calculating Attention (1)



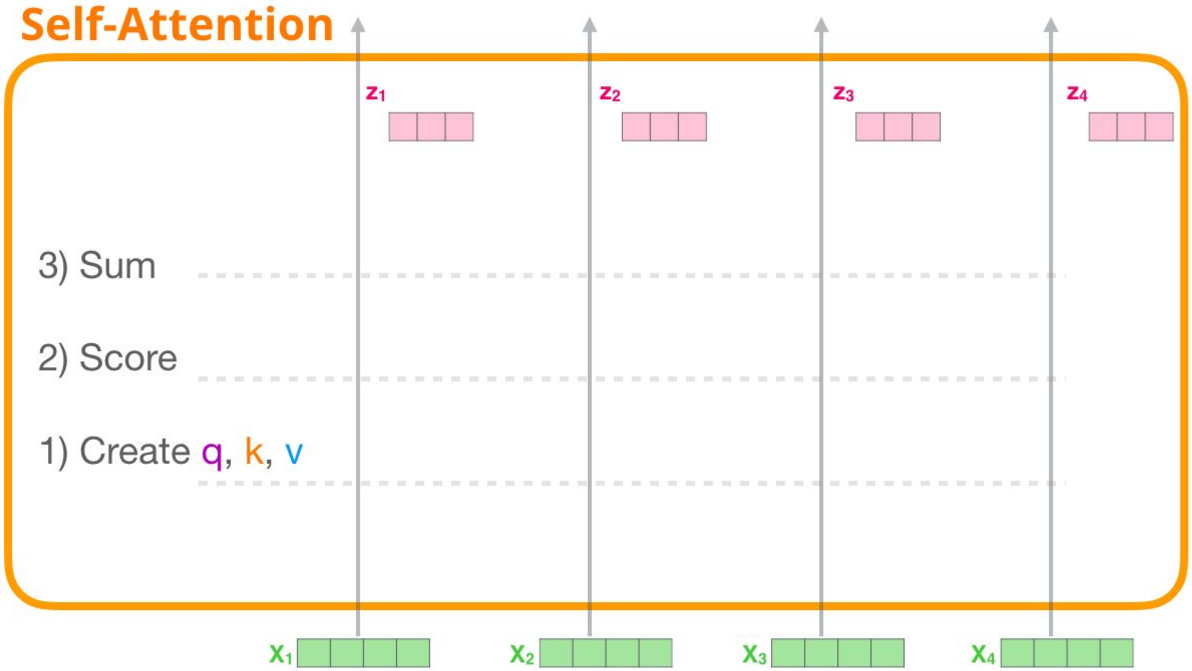
img: https://jalammar.github.io/illustrated-gpt2/

Calculating Attention (2)



img: https://jalammar.github.io/illustrated-gpt2/

Calculating Attention (3)



img: https://jalammar.github.io/illustrated-gpt2/

Attention Score Functions

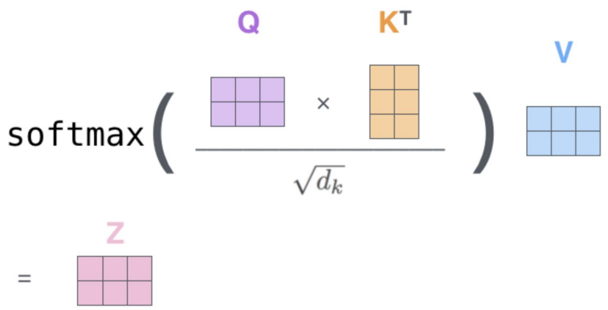
- Dot Product (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$

- Scaled Dot Product (Vaswani et al. 2017)
    - Problem: scale of dot product increases as dimensions get larger
    - Fix: scale by size of the vector

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

Self-attention Calculation in Matrix form




```
head_dim = 16


weight_q = nn.Linear(embedding_dim, head_dim)
weight_k = nn.Linear(embedding_dim, head_dim)
weight_v = nn.Linear(embedding_dim, head_dim)

querys = weight_q(input_embeddings) # (1, 5, 16)
keys = weight_k(input_embeddings) # (1, 5, 16)
values = weight_v(input_embeddings) # (1, 5, 16)
```

```
from math import sqrt
import torch.nn.functional as F

def compute_attention(querys, keys, values, is_causal=False):
        dim_k = querys.size(-1) # 16
        scores = querys @ keys.transpose(-2, -1) / sqrt(dim_k)
        weights = F.softmax(scores, dim=-1)
        return weights @ values
```

Img: https://jalammar.github.io/illustrated-transformer/

Multi-head Attention

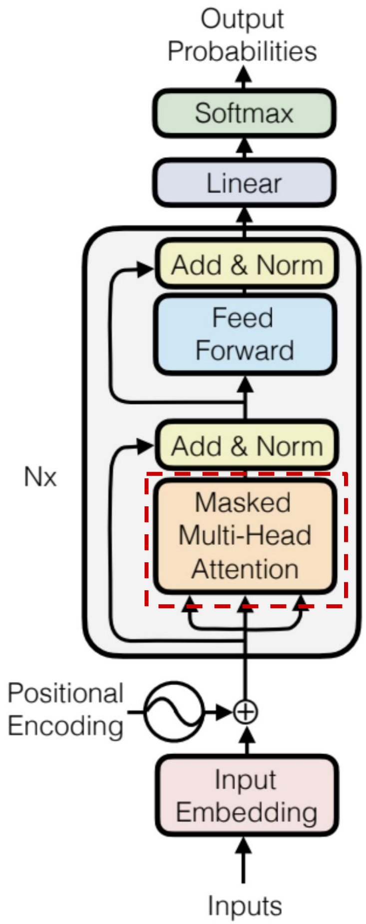
Intuition for Multi-heads

- Intuition: Information from different parts of the sentence can be useful to disambiguate in different ways

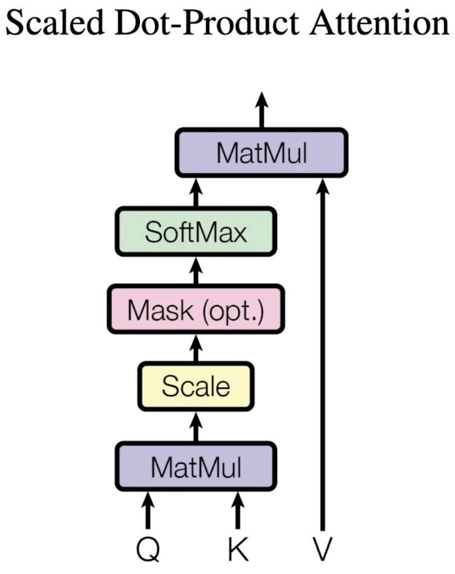
I **run** a small business
I **run** a mile in 10 minutes
The robber made a **run** for it
The stocking had a **run**

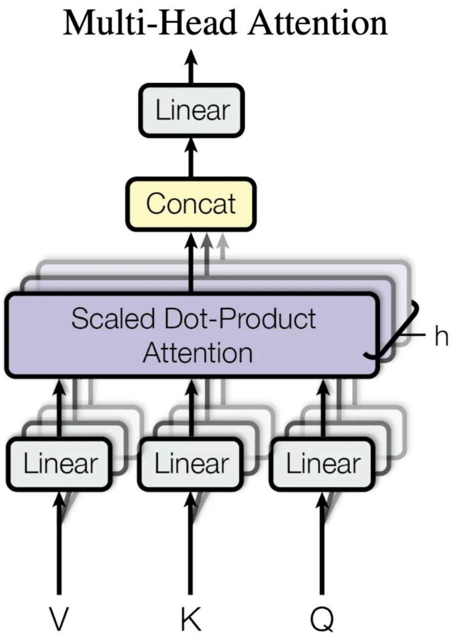
syntax
(nearby context)

semantics
(farther context)

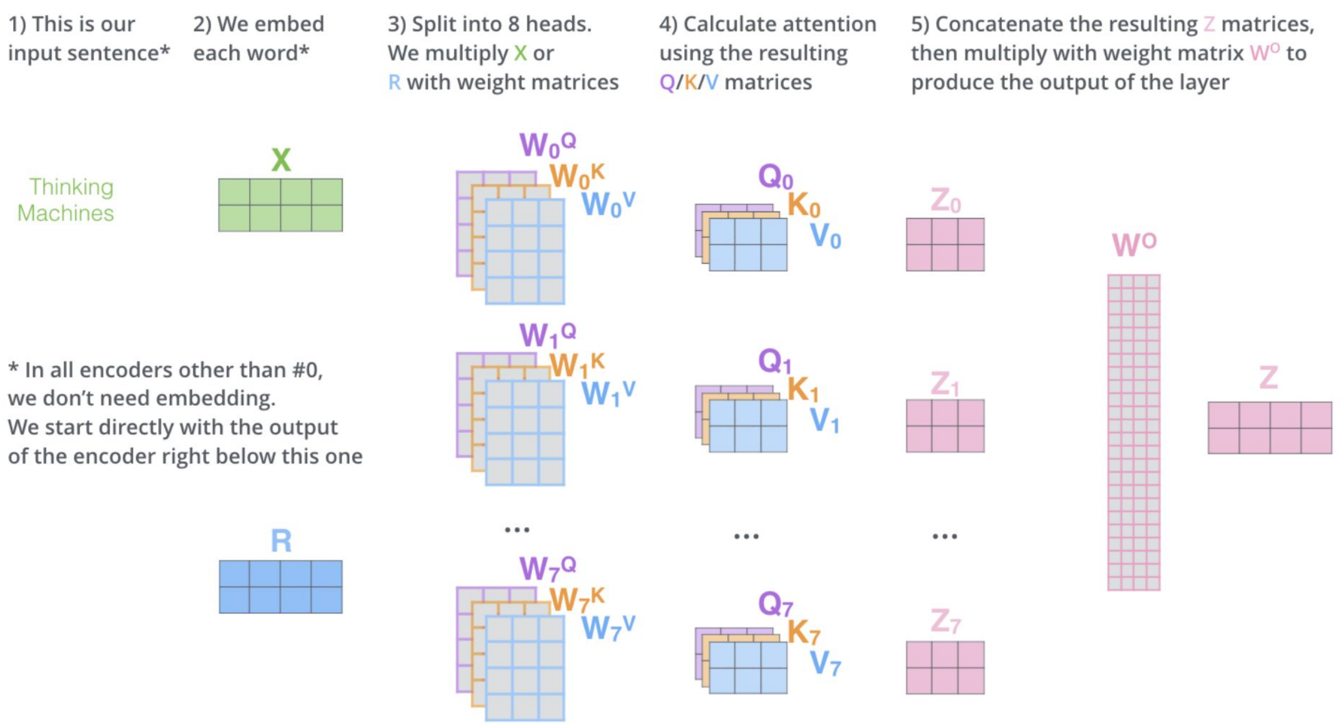


Multi-head Attention Concept





Multi-head Attention Concept



Img: https://jalammar.github.io/illustrated-transformer/

Code Example

```
class MultiheadAttention(nn.Module):
  def __init__(self, token_embed_dim, d_model, n_head, is_causal=False):
    super().__init__()
    self.n_head = n_head
    self.is_causal = is_causal
    self.weight_q = nn.Linear(token_embed_dim, d_model)
    self.weight_k = nn.Linear(token_embed_dim, d_model)
    self.weight_v = nn.Linear(token_embed_dim, d_model)
    self.concat_linear = nn.Linear(d_model, d_model)

  def forward(self, querys, keys, values):
    B, T, C = querys.size()
    querys = self.weight_q(querys).view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
    keys = self.weight_k(keys).view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
    values = self.weight_v(values).view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
    attention = compute_attention(querys, keys, values, self.is_causal)
    output = attention.transpose(1, 2).contiguous().view(B, T, C)
    output = self.concat_linear(output)
    return output

n_head = 4
mh_attention = MultiheadAttention(embedding_dim, embedding_dim, n_head)
after_attention_embeddings = mh_attention(input_embeddings, input_embeddings, input_embeddings)
after_attention_embeddings.shape
```

What Happens w/ Multi-heads?

- Example from Vaswani et al.

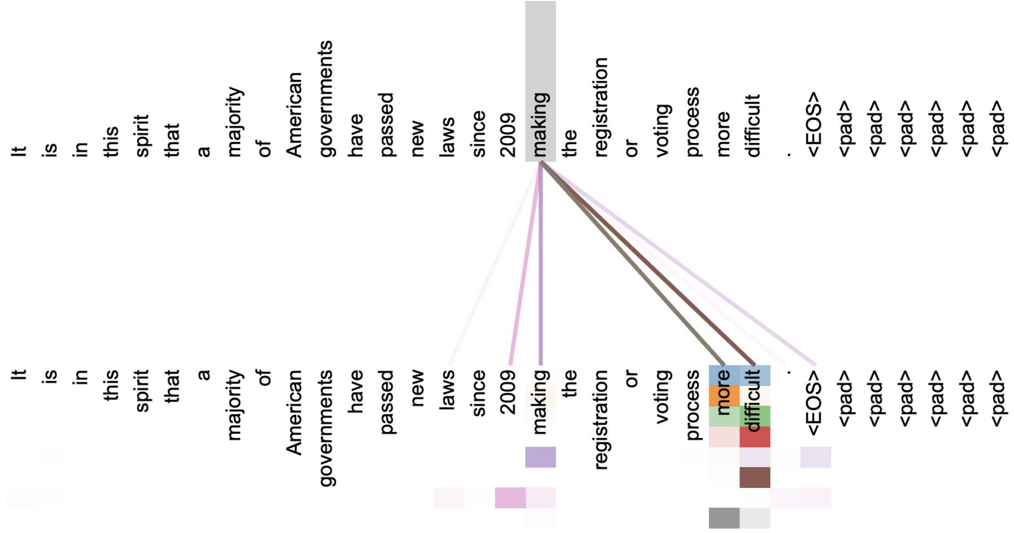
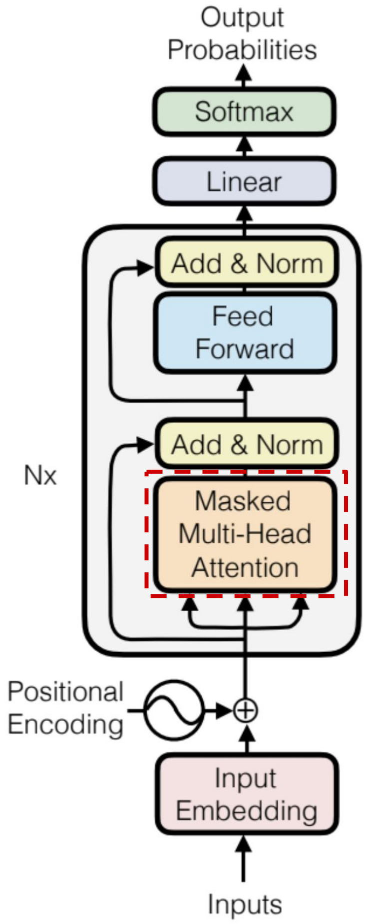


Figure 3. An example of the attention mechanism following long-distance dependencies in the encoder self-attention. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making … more difficult.' **Different colors represent different heads.**

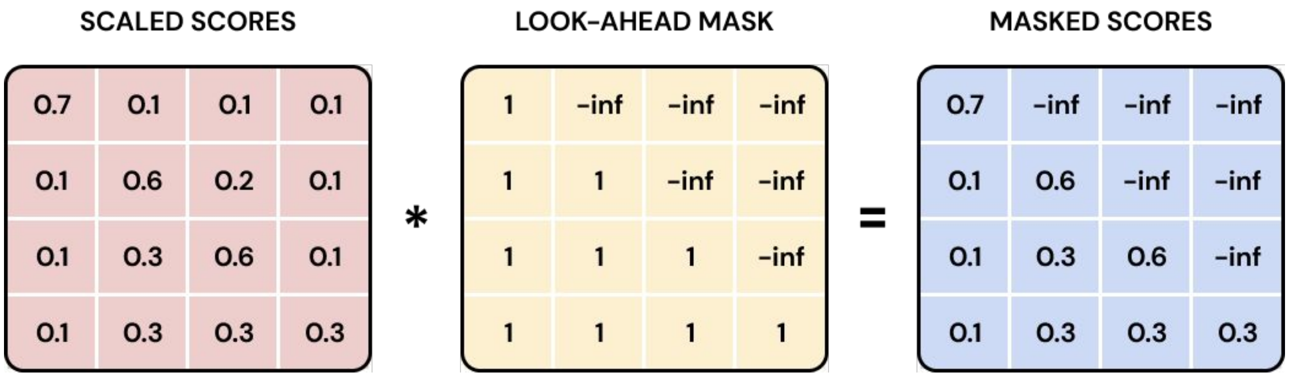
- See also BertVis: https://github.com/jessevig/bertviz

Masked Attention



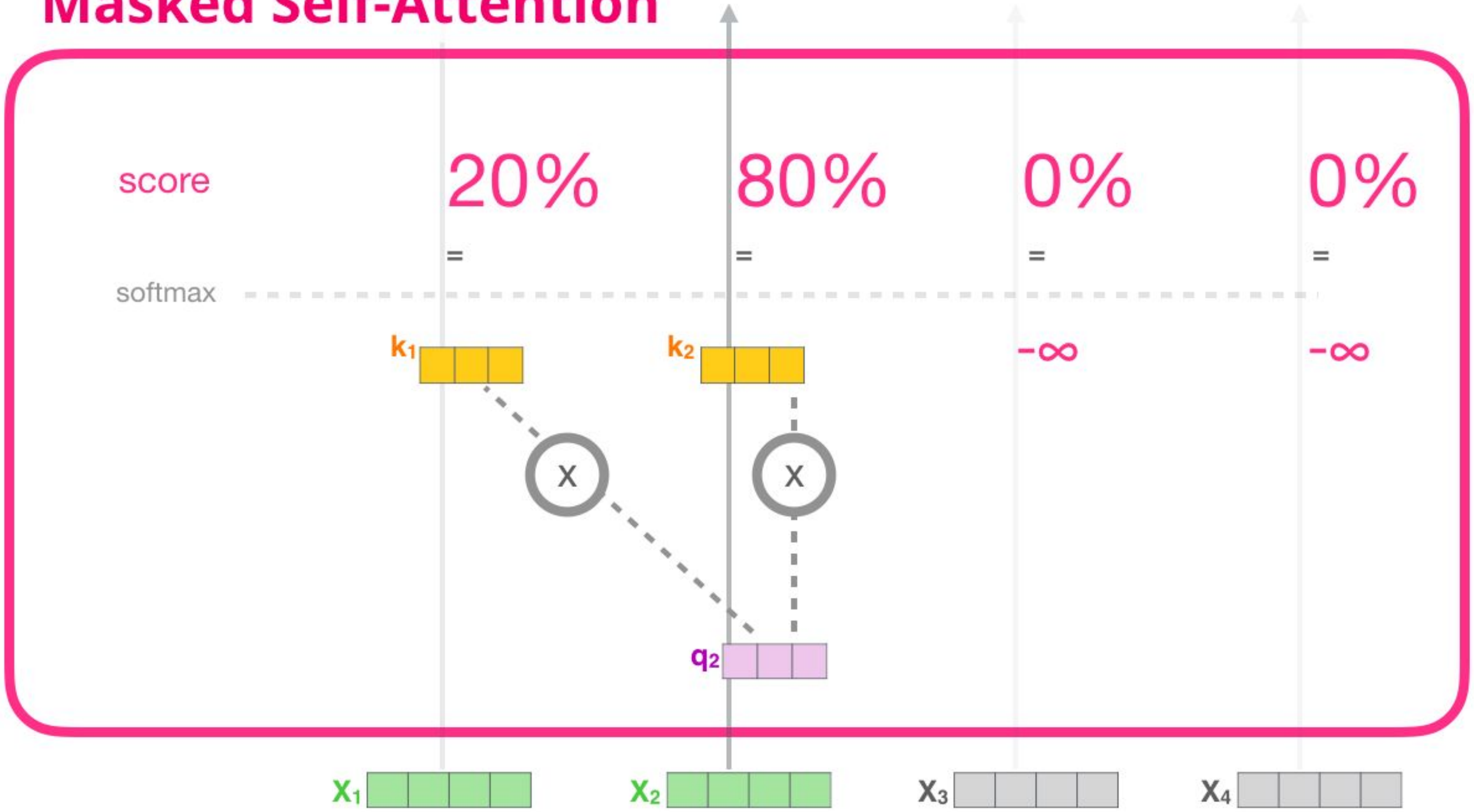
Masked Attention

- During generation, the decoder only sees previously generated text, but in training, it has access to the full text, causing data leakage.
- Masking ensures that the model only attends to past tokens.
  - Implementation: A triangular mask is applied, hiding future tokens.



Img: https://www.revistek.com/posts/transformer-architecture

Masked Self-Attention



img: https://jalammar.github.io/illustrated-gpt2/

Layer Normalization and Residual Connections

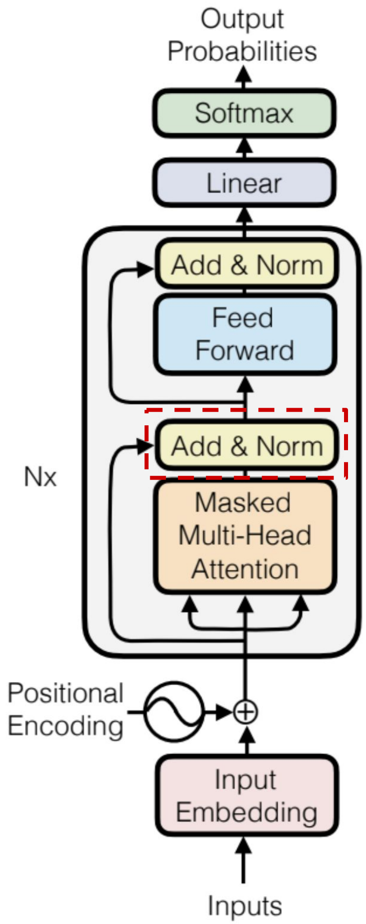
Layer Normalization (Ba et al. 2016)

- Normalizes the outputs to be within a consistent range, preventing too much variance in scale of outputs

$$\hat{x}_i = \frac{x_i - \mu}{\sigma + \epsilon}$$

$$y_i = \gamma \hat{x}_i + \beta$$

$$\mu = \frac{1}{N}\sum_{j=1}^{N} x_j \qquad \sigma = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x_j - \mu)^2}$$



Layer Normalization

					Avg	Std
The	[0.8	0.2	3.1	5.2]	2.4	2.33
I	[1.2	1.5	4.2	1.3]	2.2	1.48
It	[5.0	1.0	2.3	2.2]	2.7	1.62

The	black	cat	eats	food	and	drinks	water
I	saw	a	black	cat	PAD	PAD	PAD
It	is	ranging	from	0	to	1	PAD

RMSNorm (Zhang and Sennrich 2019)

- Root Mean Square (RMS) normalization simplifies LayerNorm by removing the mean and bias terms

$$\hat{x}_i = \frac{x_i}{\mathrm{RMS}(x) + \epsilon}$$

$$y_i = \gamma \hat{x}_i$$

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{N} \sum_{j=1}^{N} x_j^2}$$

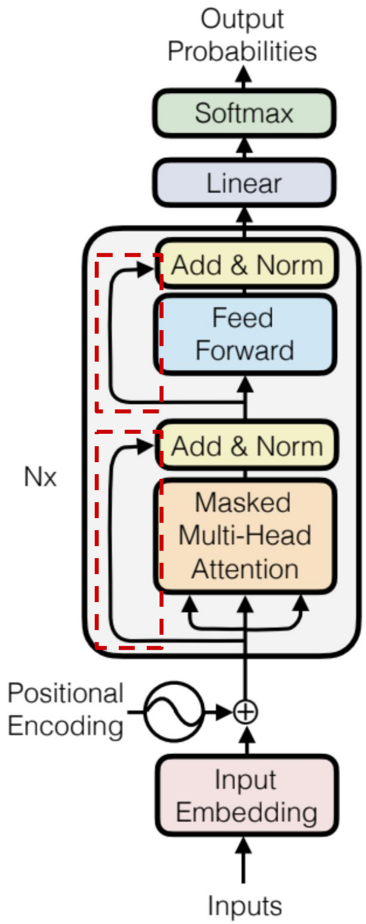
Residual Connections (Kaiming et al. 2015)

* Cited by 250467

- Add an additive connection between the input and output

$$y = \mathcal{F}(x) + x$$

- Prevents vanishing gradients and allows f to learn the difference from the input



Kaiming et al. 2015. Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385

Post- vs. Pre-Layer Norm (e.g., Xiong et al. 2020)

- Where should LayerNorm be applied? Before or after?
- Pre-layer-norm is better for gradient propagation

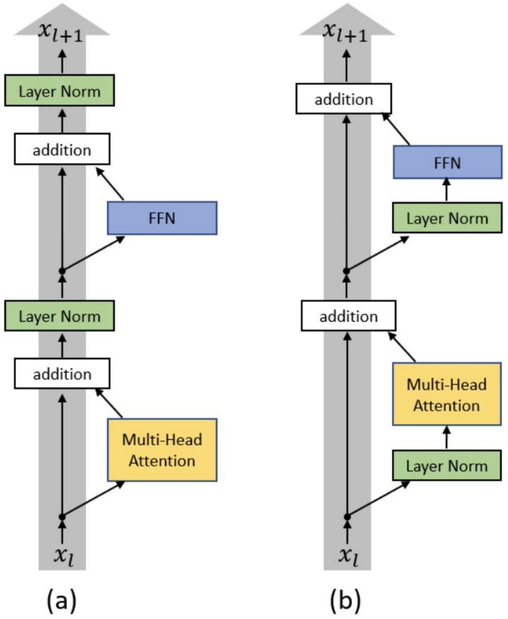


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

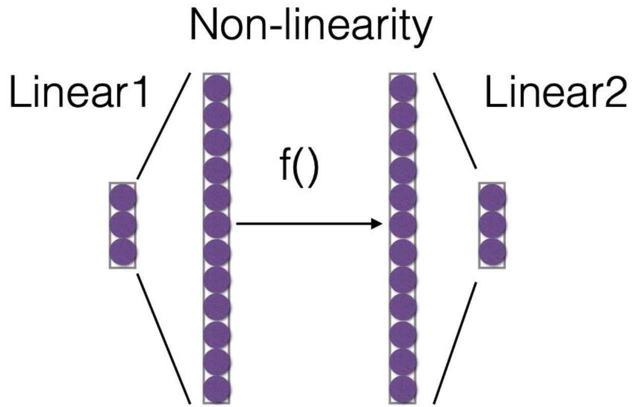
Xiong et al. (2020) On layer normalization in the transformer architecture. PMLR. https://arxiv.org/abs/2002.04745

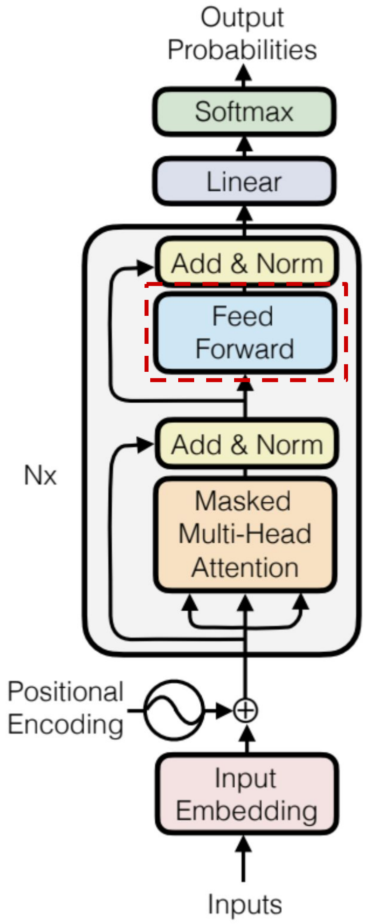
Feed Forward Layers

Feed Forward Layers

- Fully connected layer that extracts combination features from the attended outputs

$$\mathrm{FFN}(x; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2) = f(\mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

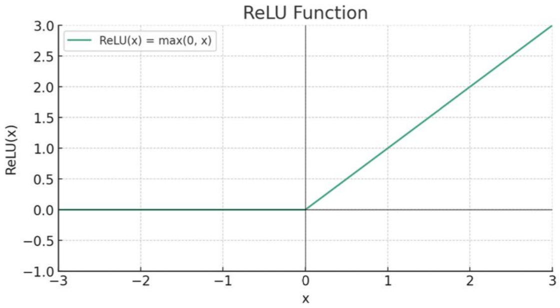




Some Activation Fundations in Transformers

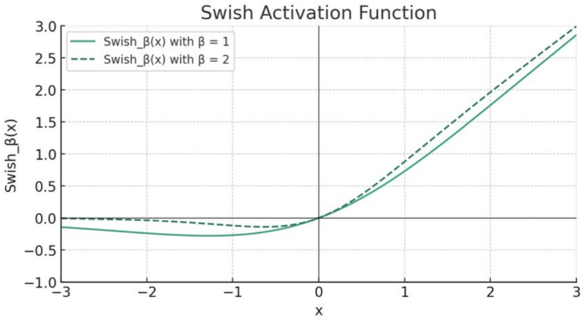
- Vaswani et al.: ReLU

$$\mathrm{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$$



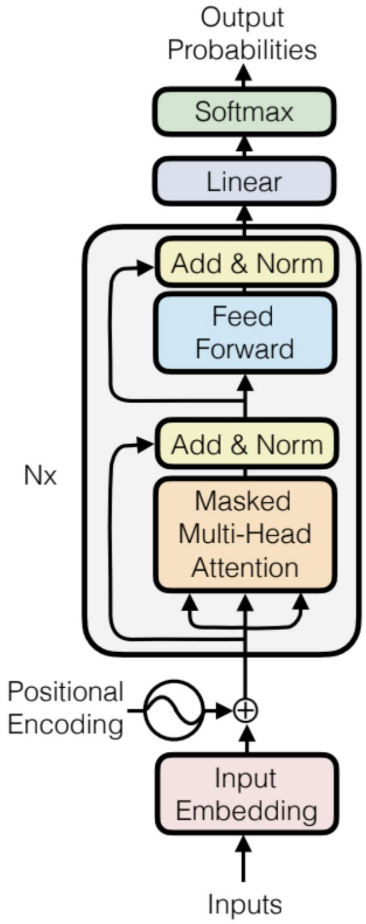
- LLaMa: Swish/SiLU (Hendricks and Gimpel 2016)

$$\mathrm{Swish}(\mathbf{x}; \beta) = \mathbf{x} \odot \sigma(\beta \mathbf{x})$$



Core Transformer Concepts

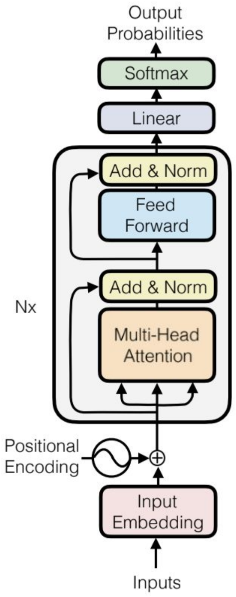
- ✓ Positional encoding
- ✓ Attention
- ✓ Multi-headed attention
- ✓ Masked attention
- ✓ Residual + layer normalization
- ✓ Feed-forward layer



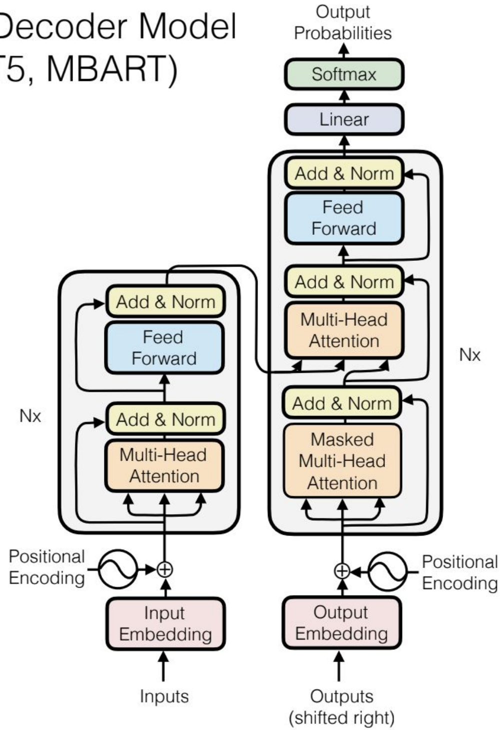
Transformer-based architectures (BERT, GPT, and T5)

Three Types of Transformers (1)

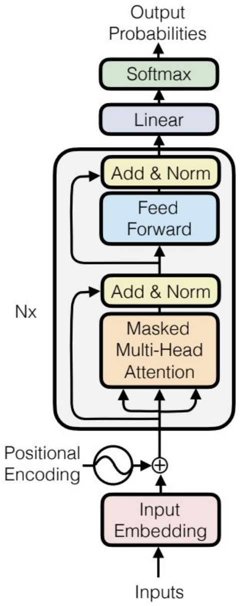
Encoder Only Model
(e.g. BERT)



Encoder-Decoder Model
(e.g. T5, MBART)



Decoder Only Model
(e.g. GPT, LLaMa)



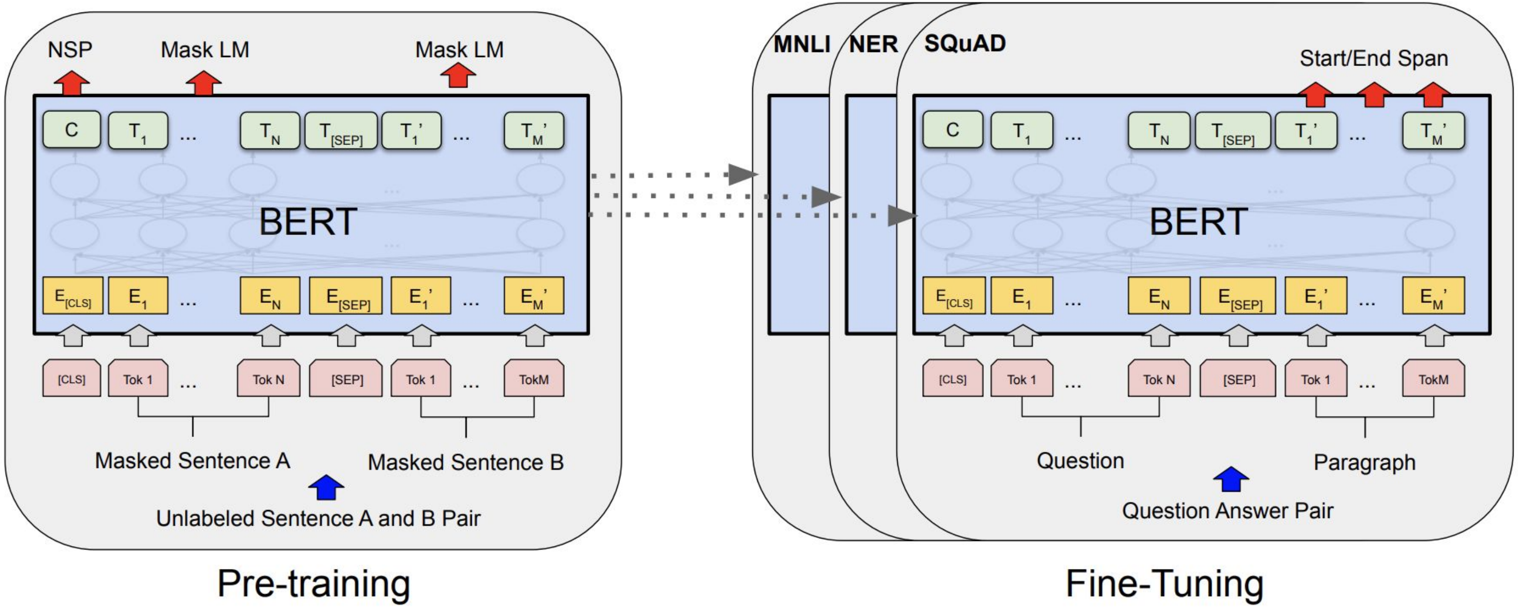
Three Types of Transformers (2)

Transformer-based models fall into three major groups:

- Encoder-only Models
  - Used for natural language understanding (NLU) tasks
  - Example: BERT
- Decoder-only Models
  - Used for natural language generation (NLG) tasks
  - Example: GPT, LLaMA
- Encoder-Decoder Models
  - Handles both understanding and generation tasks
  - Examples: T5, BART

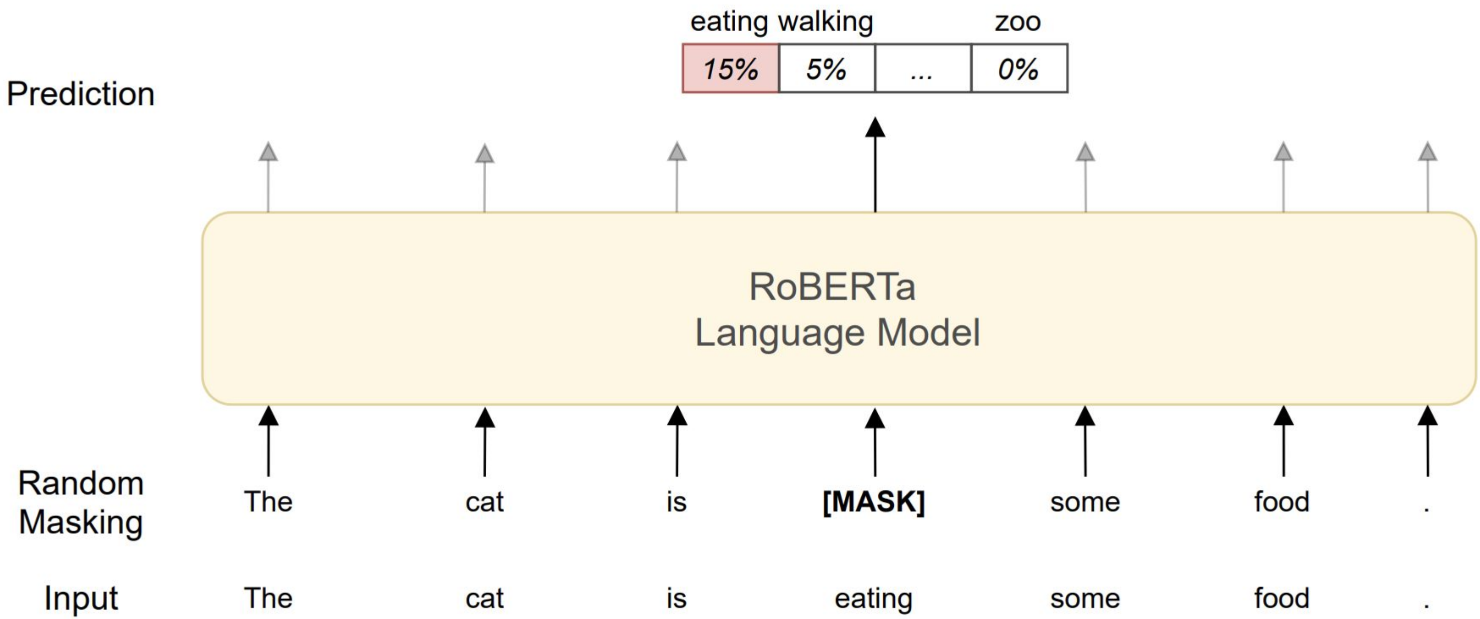
BERT (Devlin et al. 2018)

- Bidirectional Encoder Representations from Transformers



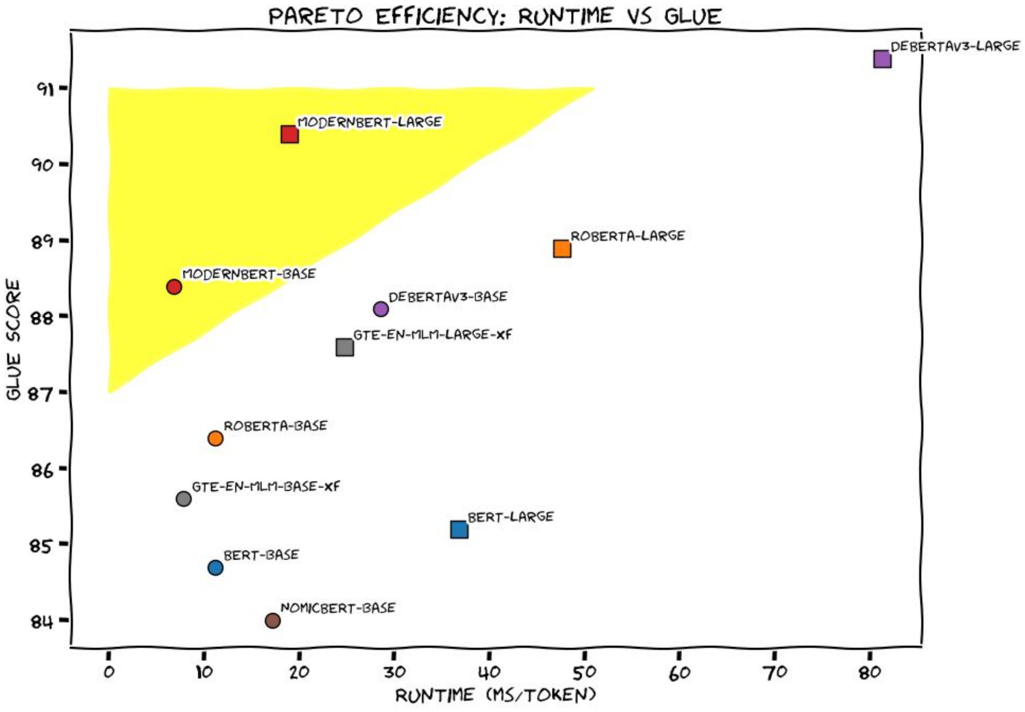
Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805

Masked Language Modeling (MLM)



Img: https://arxiv.org/abs/2104.01642

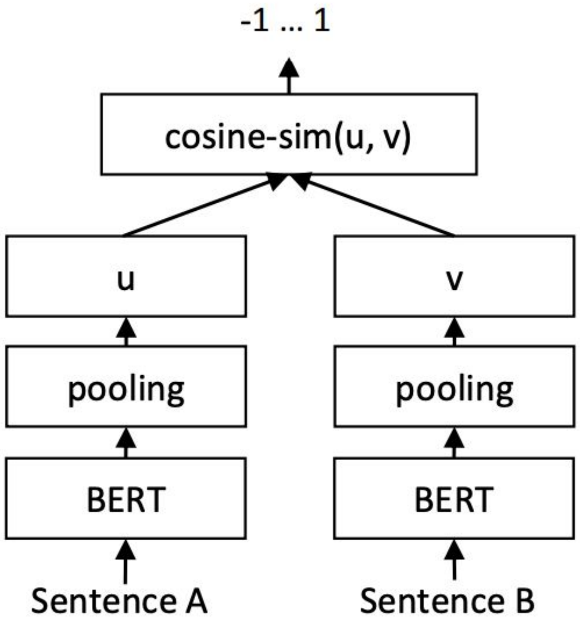
ModernBERT (Warner et al. 2024)



https://huggingface.co/blog/modernbert

Sentence-BERT (SBERT) (Reimers and Gurevych 2019)

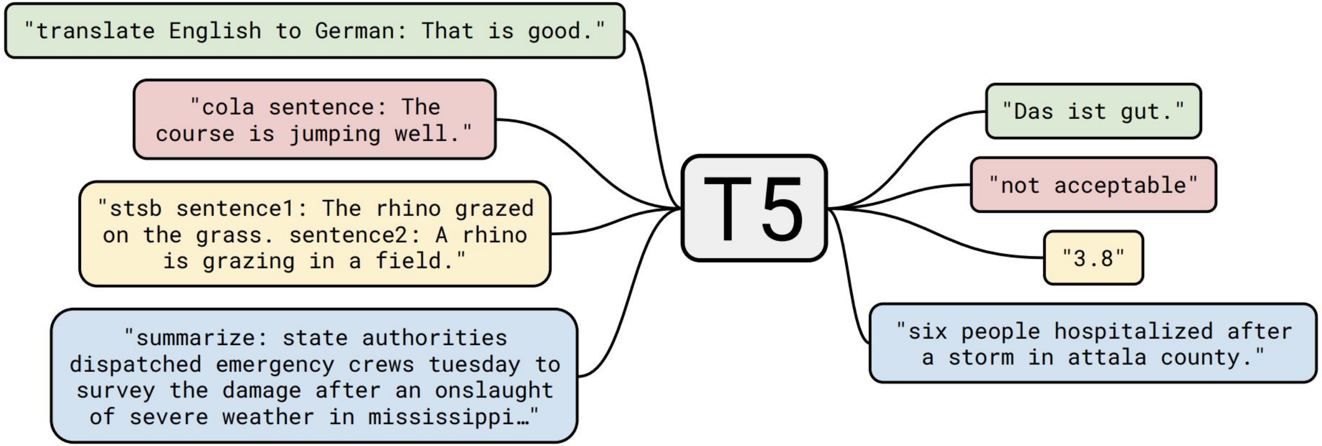
- A modification of BERT designed for sentence embeddings
- Generates context-aware vector representations of sentences
- Optimized for semantic similarity tasks via siamese or triplet networks, enabling fast & accurate sentence comparison
- Widely used in semantic search, clustering, and QA



Reimers and Gurevych 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://arxiv.org/abs/1908.10084

T5: Text-to-Text Transfer Transformer (Raffel et al. 2019)

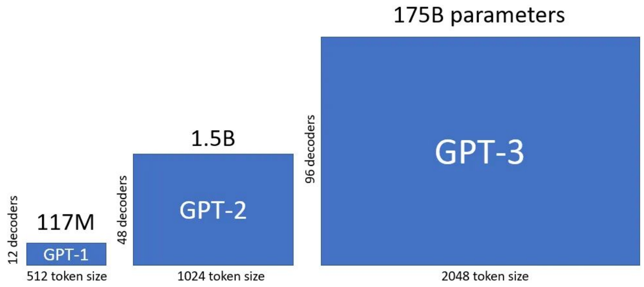
- Treats all NLP tasks as text-to-text translation
- Example: For summarization, input starts with "Summarize:"

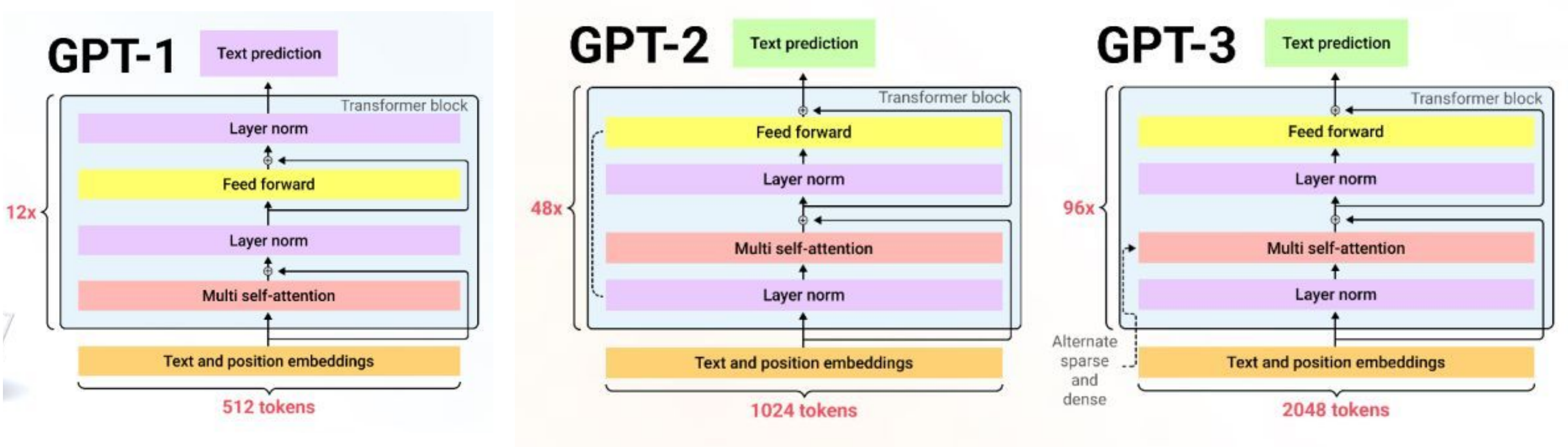


Raffel et al. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://arxiv.org/abs/1910.10683

GPT (Yenduri et al. 2023)

- Generative Pre-trained Transformer





Yenduri et al. 2023. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. https://arxiv.org/abs/2305.10435

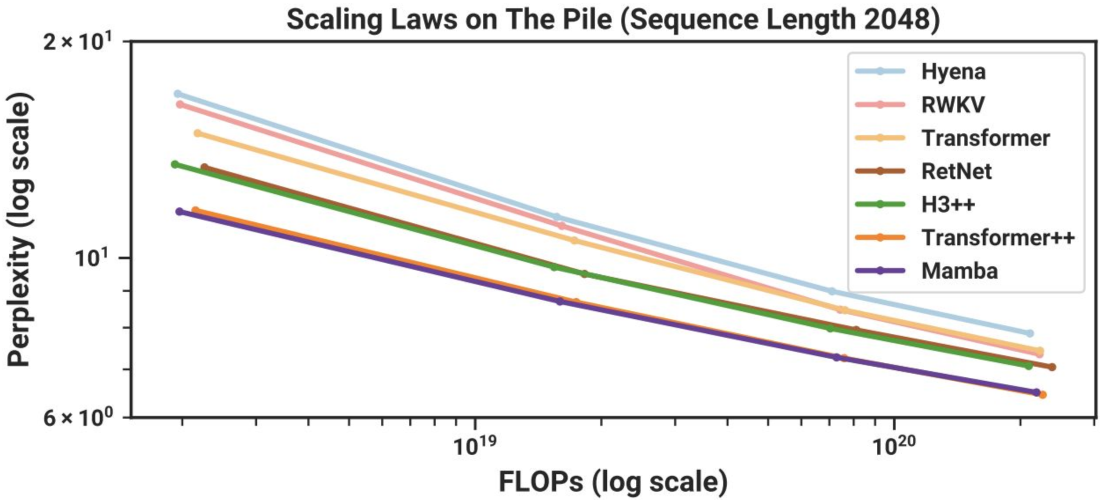
Img: https://www.linkedin.com/posts/ayushi-sharma-8a285a185_gpt-1-gpt-2-and-gpt-3-are-almost-similar-activity-7026040251622043648-NNUz, https://medium.com/@YanAIx/step-by-step-into-gpt-70bc4a5d8714

Original Transformer vs. LLaMA

	Vaswani et al.	LLaMA
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SiLU
Positional Encoding	Sinusoidal	RoPE

How Important is It?

- “Transformer” is Vaswani et al., “Transformer++” is (basically) LLaMA



- Stronger architecture is ≈10x more efficient!

Img: GU and Dao (2023).

Model Type	Rep. Model	Advantages	Disadvantage
Encoder	Google's BERT	● Achieves higher performance in NLU compared to decoder models due to bidirectional context comprehension ● Supports parallel computation, enabling faster training and inference ● Excels in downstream tasks across various applications	● Not suitable for NLG tasks ● Limited context length constraints performance
Decoder	OpenAI's GPT, Meta's LLaMA	● Excels in text generation tasks ● Performs well with relatively long context windows	● Unidirectional processing leads to lower performance in NLU tasks ● Can convert all tasks into a generation problem, but this may be inefficient
Encoder-Decoder	Meta's BART, Google's T5	● Strong performance in both generation and understanding tasks ● Uses bidirectional encoding for better comprehension and leverages encoder outputs in the decoder, enhancing context-aware generation	● More complex due to the combined use of both encoder and decoder ● Requires more data and computational resources for training

The Annotated Transformer

Attention is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

- *v2022: Austin Huang, Suraj Subramanian, Jonathan Sum, Khalid Almubarak, and Stella Biderman.*
- *Original: Sasha Rush.*

The Transformer has been on a lot of people's minds over the last ~~year~~ five years. This post presents an annotated version of the paper in the form of a line-by-line implementation. It reorders and deletes some sections from the original paper and adds comments throughout. This document itself is a working notebook, and should be a completely usable implementation. Code is available here.

https://nlp.seas.harvard.edu/annotated-transformer/

Any Questions?