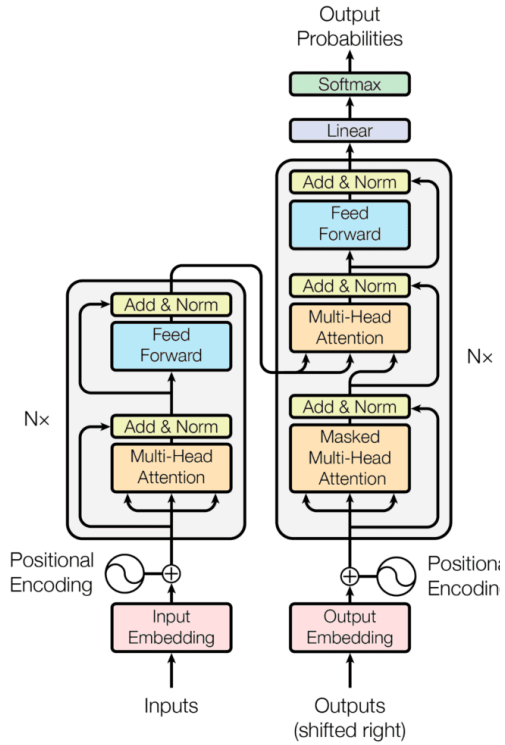
Transformer for AI
Week 1: Transformers for signal processing, the rationale

Stanley Liang, PhD

Research Fellow, NLM

What is the transformer architecture

- Transformer is a deep neural network (NN) architecture
- Transformer started to attract the public attention from the paper “Attention Is All You Need” on sequence-to-sequence (Seq2Seq) translation task (English-German & English-French) merely relied on the attention mechanism
- It is the SOTA (state of the art) NN architecture for NLP tasks
- Its application has been extended to non-NLP AI tasks, such as image processing, video processing, time series data, etc.



Journal Club agenda

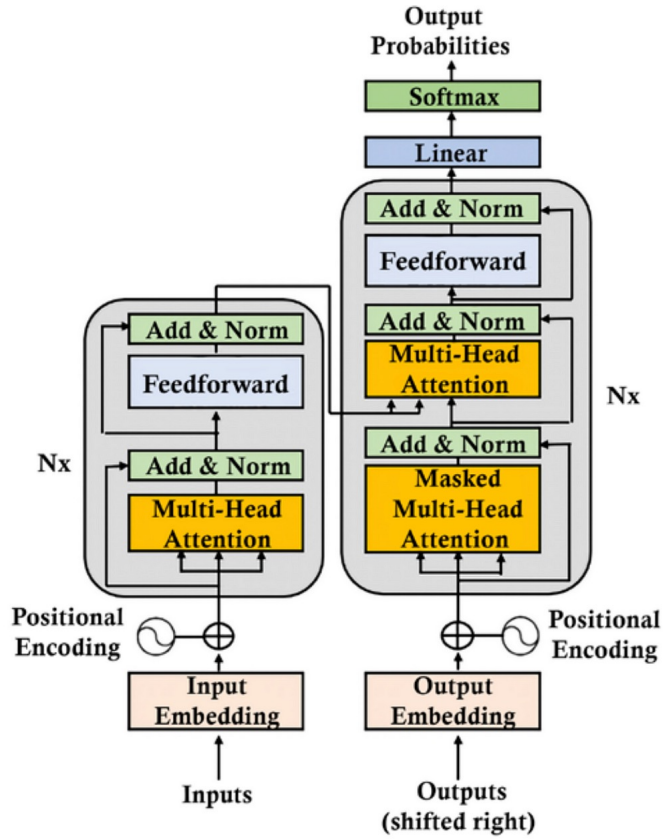
- This journal club will be held for six weeks, Thursday afternoon at 1:00 PM – 2:00 PM
  - Week 1: Transformers for signal processing, the rationale (June 13)
  - Week 2: Attention is all you need, the attention (June 20) mechanism for transformers.
  - Week 3: Transformers and RNN: BERT vs Seq2Seq (NLP) (June 27)
  - Week 4: Vision transfer, breaking images into tokens (July 11)
  - Week 5: Large langue models, GPT and ChatGPT (July 18)
  - Week 6: Generative models, GANs, and denoising diffusion models (July 25)
- The demo notebook and papers are available on GitHub: https://github.com/StanleyLiangYork/2024_journal_club_Transformer_AI
- The discussion will combine both the rationale and the model implementation
- To get the certificate, you should attend five discussion sections

The resource on GitHub

- The journal club is on CS and AI – both reading and coding
- GitHub resource: https://github.com/StanleyLiangYork/2024_journal_club_Transformer_AI/tree/main
- The “Papers” folder contains the recommended readings – mostly short, easy math, no pressure
- The Python notebooks are ready to run on Google Colab
- Recommended run order
    1. Python coding tutorial – if you are familiar with numpy, pandas, OOP in python, skip it
    2. Recurrent neural network
    3. RNN with attention
    4. Implementation self-attention
    5. Positional encoding
    6. Embedding with positional encode – pipeline from tokenize to embedding with positional encoding

The components of transformer

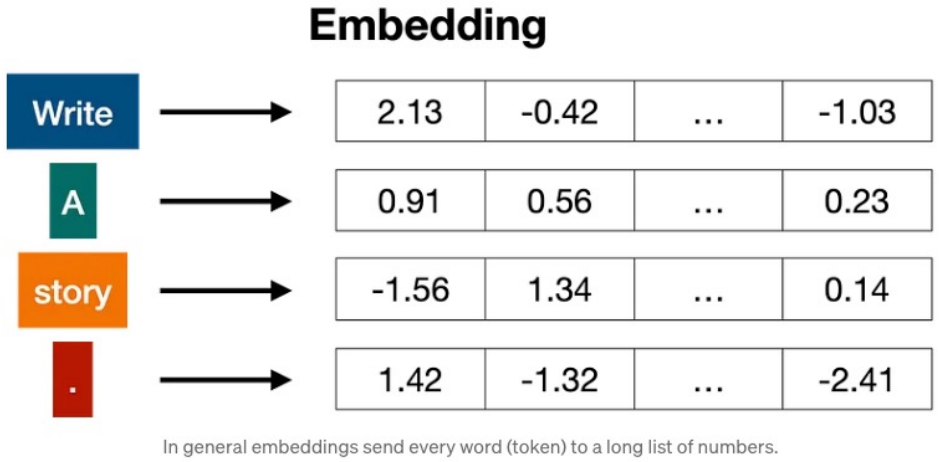
- For NLP, the data processing of a transformer can be simplified as the following parts
  - Tokenization
  - Embedding
  - Positional encoding
  - Transformer block (several identical subunits)
  - Softmax - prediction
- The key components of a transformer are the multi-head self-attention mechanism (week 2) and the encoder-decoder architecture (week3)



Tokenization and embedding

- Tokens are small units of characters that carries semantic meaning, in English, words are usually considered as tokens
- Tokenization is the processing to split text data into tokens, including words, punctuation signs, etc.

- Embedding is the process to convert text tokens to numbers.
- Each word will be represented by a vector with identical dimension.
- The word embedding model is trained to understand the context of natural language, similar or closely related words have higher similarity scores than irrelevant words.



Positional encoding

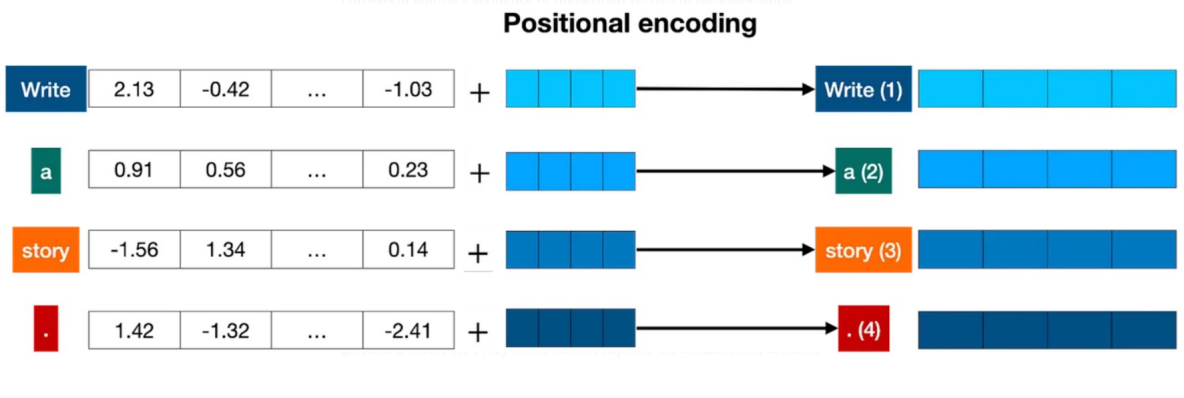
- The word embedding represents the meaning of the tokens in the context
- We also need to know the position of the tokens in the text, especially their relative position to each other
- Unlike recurrent Neural Networks (RNNs), transformer gave up the recurrence mechanism to speed up training and capture longer dependencies of the sequence.
- Transformer adds extra information to the tokens about their position in the text –positional encoding -- each word with information about its position in a sentence

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

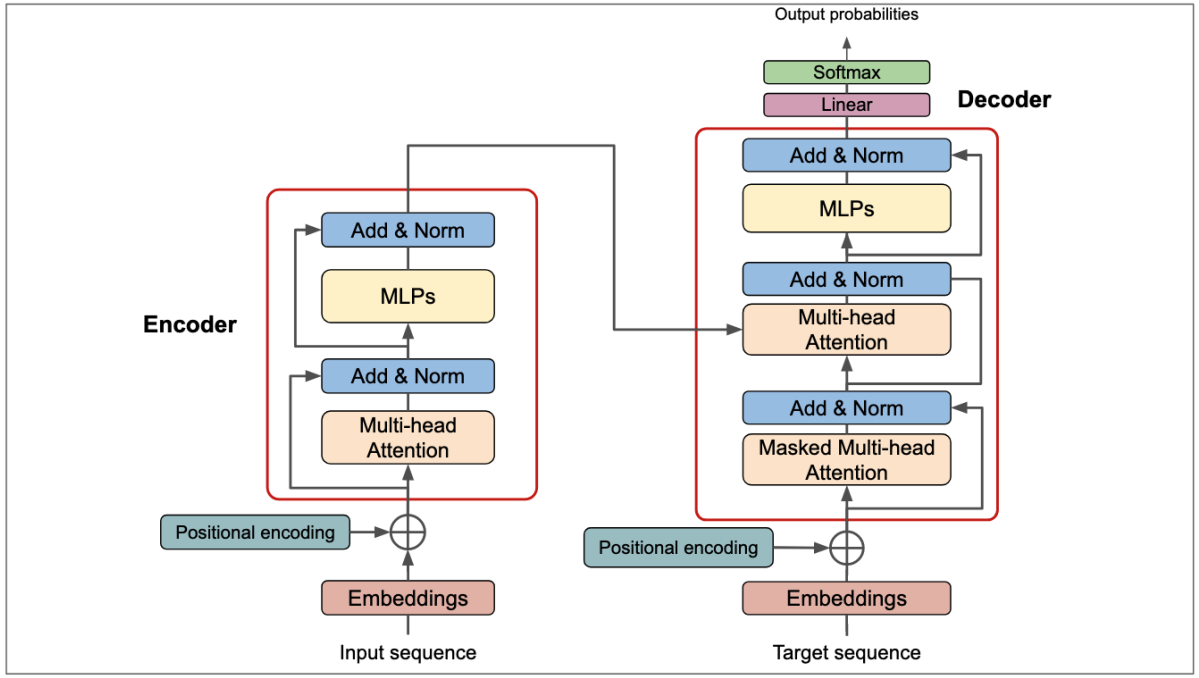
$$\omega_k = \frac{1}{10000^{2k/d}}$$

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



The encoder-decoder architecture

- After the text is encoded and the position is attached to each token, we will send the numeric information to the backbone of the transformer: encoder-decoder architecture
- The basic transformer architecture consist of an encoder and a decoder, with similar structure
  - multi-head self-attention
  - feed forward MLP

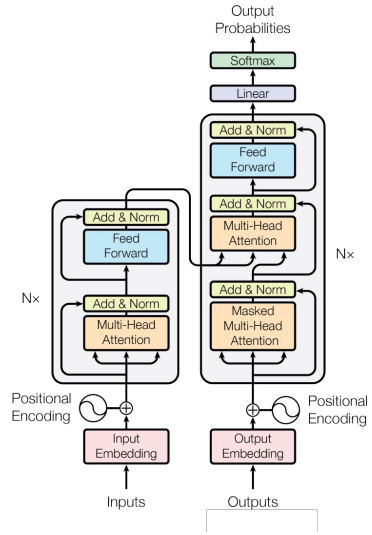


Encoder and Decoder

- Encoder – Decoder: originally designed for language translation.
  - Encoder: takes the input text from the source language, encodes to the latent vector space
  - Decoder: receive feature vector from the latent space, use both self-attention and cross-attention mechanism to learn the semantic features to decode the vector features to target text (language)
- Encoder and decoder can be detached and use separately
  - Encoder – focus on the input sequence for semantic understanding – BERT (Bidirectional Encoder Representations from Transformers)
  - Decoder – take the encoded text vector representation to generate next text conditioned on the previous text – GPT (Generative Pre-trained Transformer)
  - Masking for attention: prevent the decoder cheating by looking at the un-generated text

BERT

Encoder

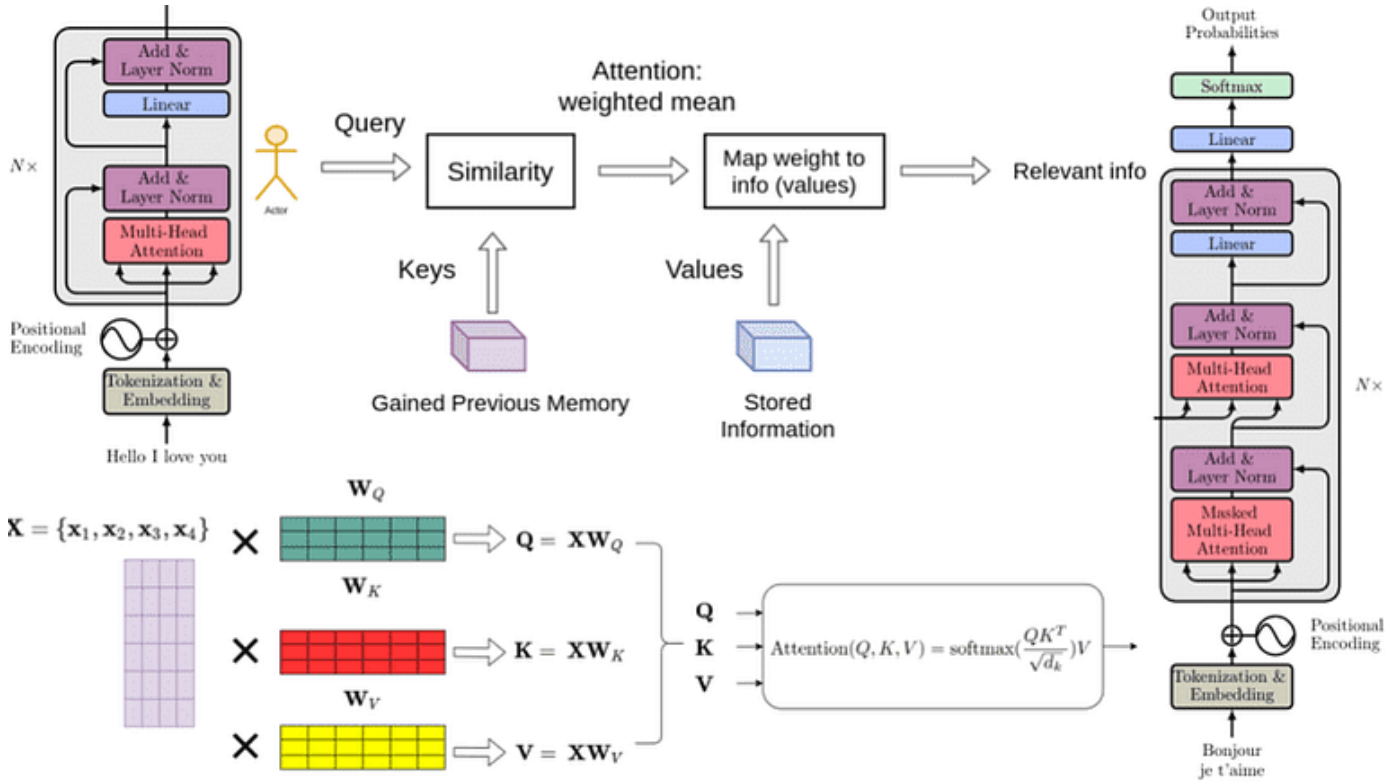


GPT

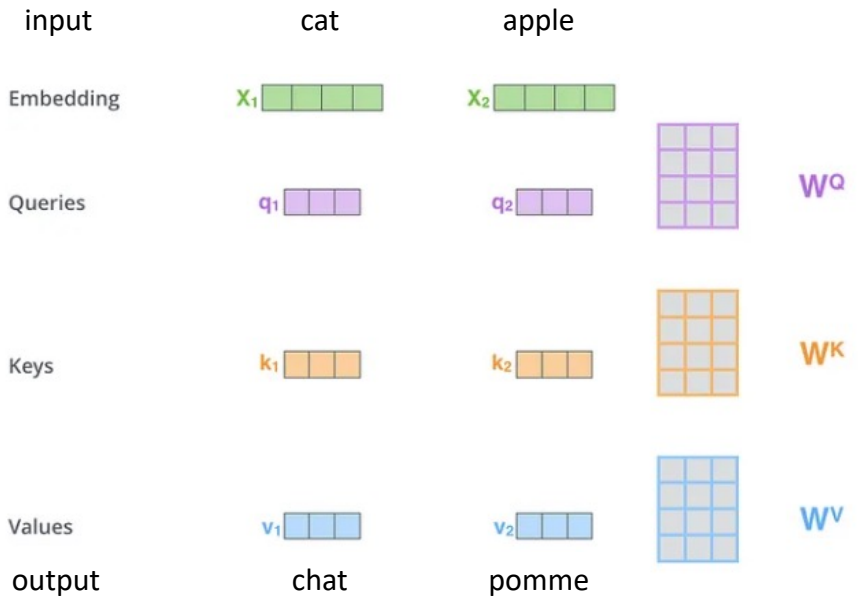
Decoder

Transformer Attention: self-attention

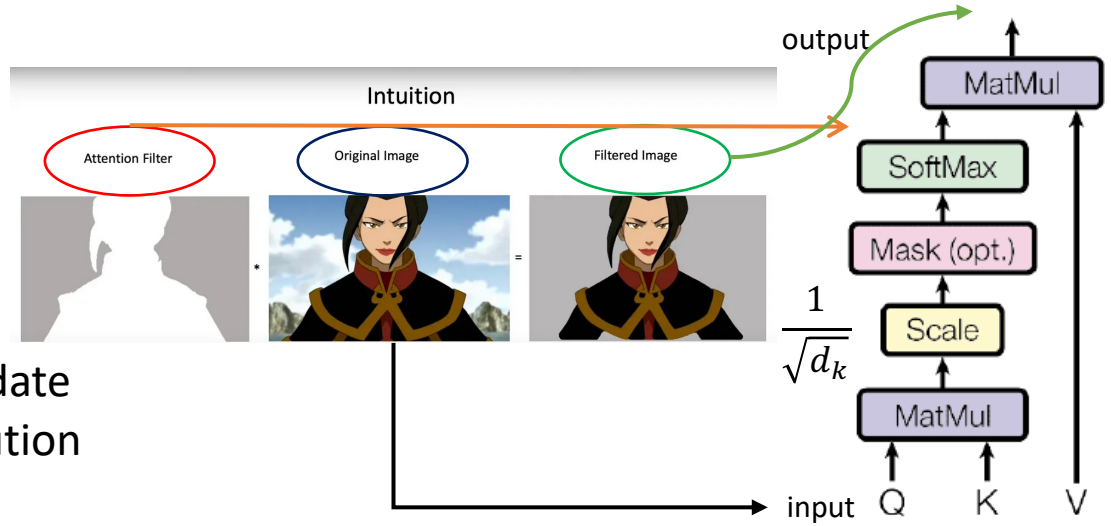
- The transformer uses attention by dispensing with recurrence (time pattern) and convolutions (position pattern)
- The core component and the building block of the transformer is the attention + Feedforward MLP
- Attention is the technique to bind the means of words to the context, using three components
  - Key -- K
  - Query -- Q
  - Value – V
- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$



The attention mechanism



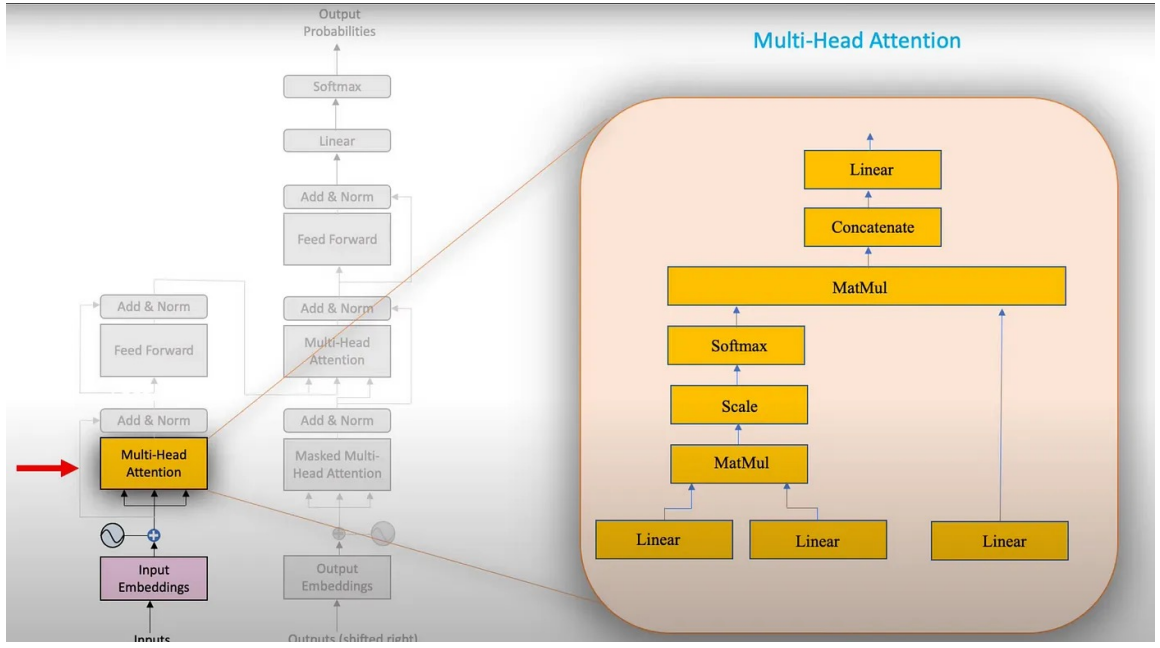
- Query vector:
  - represents the element of interest or the context with desired information, derived from the current position of the input sequence or the output of the previous layer
  - define the similarity / relevance of the current context to other tokens in the sequence
  - the representations of the source text
- Key vector:
  - the projection of the token encodings in the query vector
  - Keys are used to compute the degree of relevance to the query
  - Dot product or other similarity measures
- Value vector:
  - the projection of the input information like the keys
  - store the information (weighted by attention score) to update the representation of the query to determine the contribution of each token to the final output
  - Higher attention score means the corresponding token more important

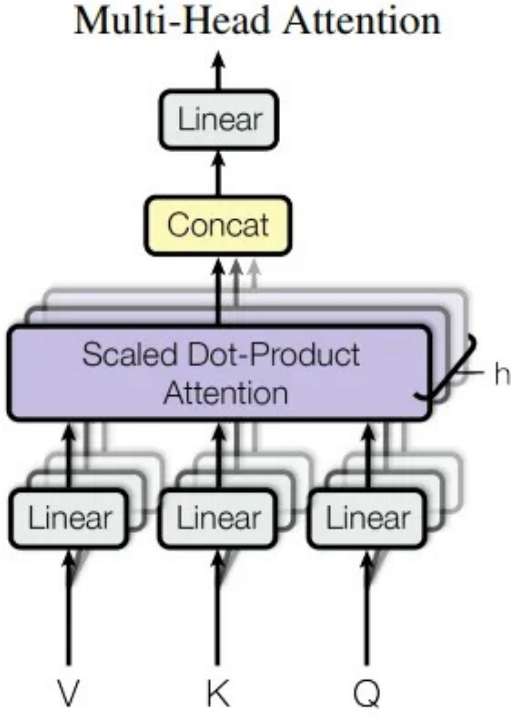


$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-head attention mechanism

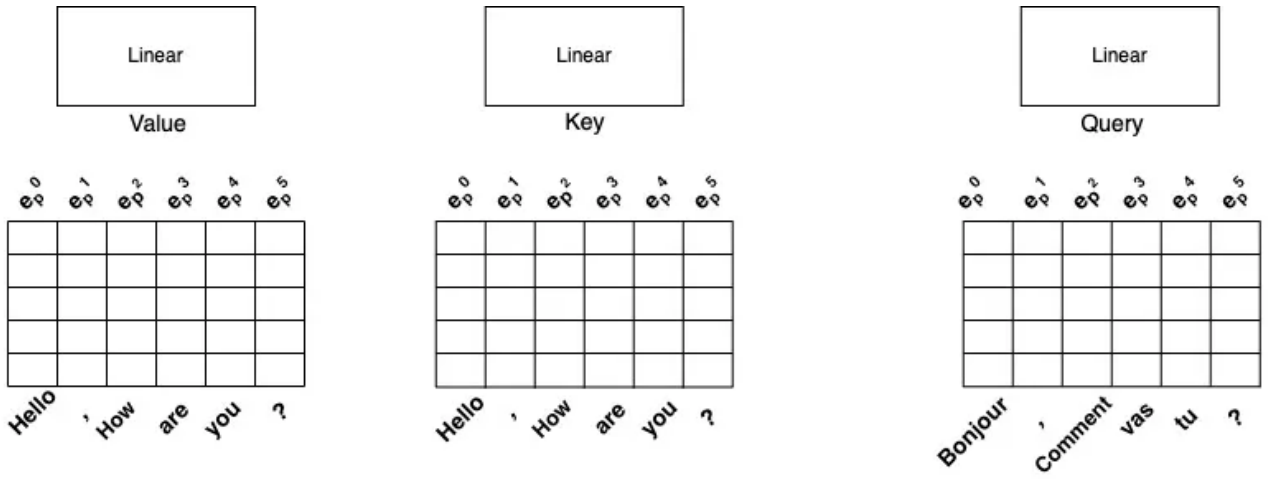
- The multi-head attention mechanism project the query, key, and value multiple times, using different learned projection each time
- Each head performs the single attention mechanism in parallel for an independent output, which are concatenated to produce the final result
- It allows the transformer to extract information from different representation subspaces
- $multihead(Q, K, V) = concat(head_1, \dots, head_h)W^O$
  - where $head_i = attention(QW_i^Q, KW_i^k, V_i^V)$

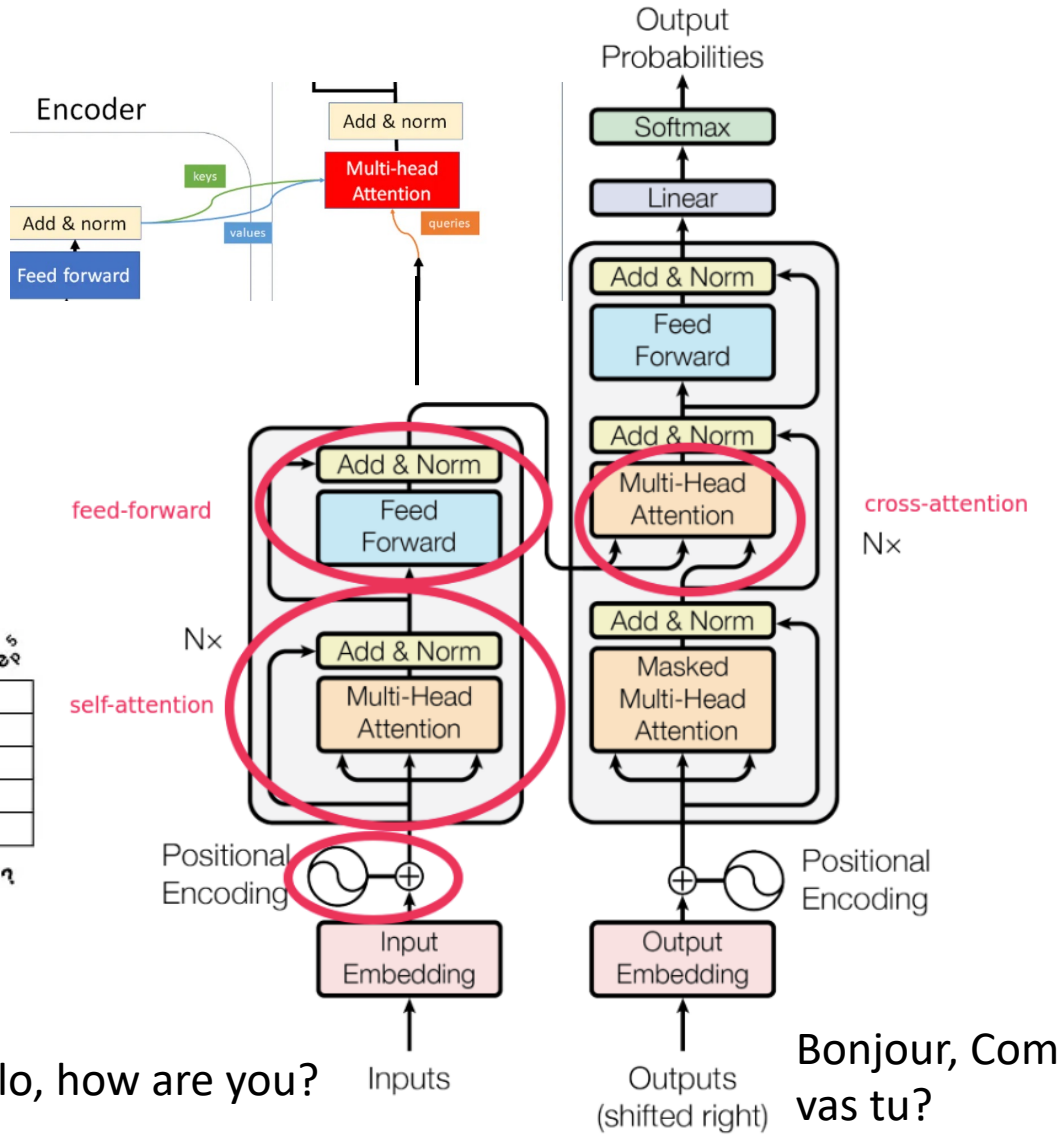




Cross-attention

- Cross attention is the part to pass information from encoder to decoder
- During the training phase, the decoder accesses both the attention vector from the encoder and the expected outcome



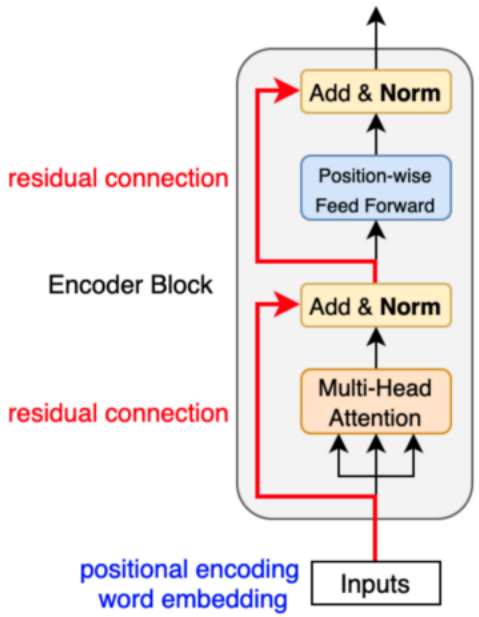


Hello, how are you?

Bonjour, Comment vas tu?

The encoder

- A unit block of the encoder is composed of a multi-head attention sublayer and a feed forward sublayer with the Rectified Linear Unit (ReLU) activation
- To increase the gradient flow, a normalization layers and a residual connection are added between sublayers
- The lack of recurrence structure lets the transformer relies on the positional encodings to the inputs to get the relative positions of tokens in the sequence



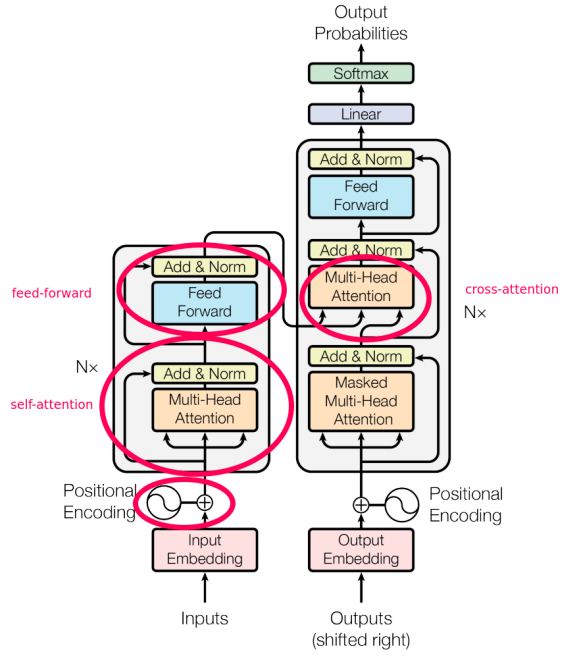


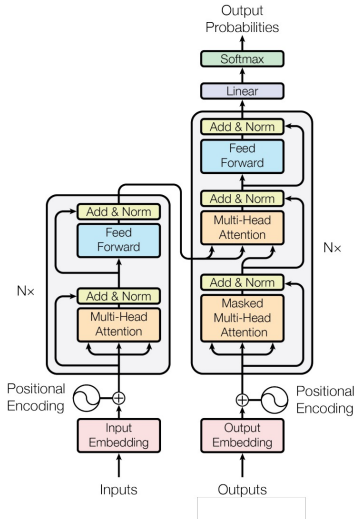
Figure 1: The Transformer - model architecture.

The Decoder

- The decoder has similar unit blocks as the encoder
- The first sublayer of decoder receives the previous output of the decoder stack, adds the position encoding, and uses the multi-head self-attention to process it
- Unlike the encoder taking the whole sequence as input, the decoder only takes the preceding token
- The predicted output of the decoder only depends on the known outputs in front of it
- Self-attention with masking: to suppressing the matrix values from illegal connections
- Queries from the previous sublayers, keys and values from the output of the decoder
- The feature goes through the fully connected layer, the softmax layer to predict the next token

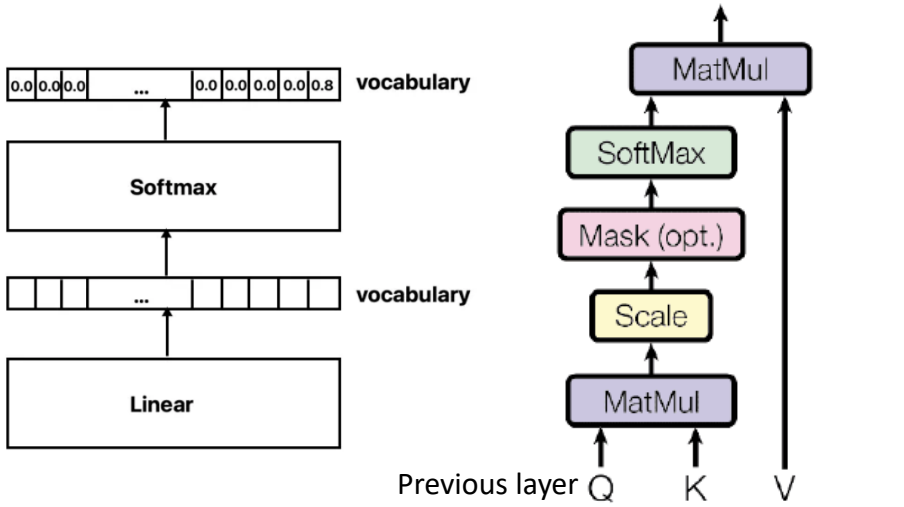
BERT

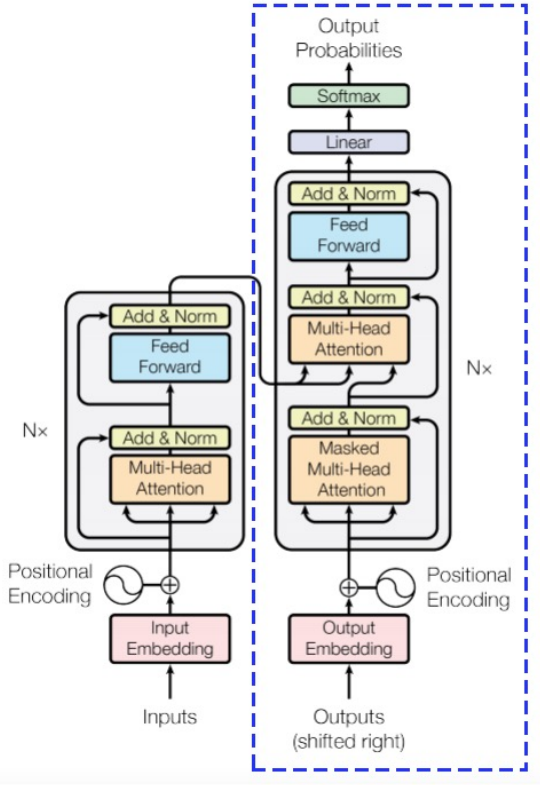
Encoder



GPT

Decoder





Summary

- Each word forming an input sequence is transformed into a d-dimensional embedding vector
- Each embedding vector representing an input word is augmented by a positional encoding vector of the same length to get the positional information into the input
- The augmented embedding vectors are fed into the encoder block consisting of multi-head attention sublayer + feed forward sublayer. The encoder attends to all words in the input sequence, including the preceding and succeeding words, the transformer encoder is bidirectional.
- The decoder receives as input its own predicted output word at timestep $t - 1$
- The input to the decoder is also augmented by positional encoding as the encoder
- The augmented decoder input is fed into the three sublayers comprising the decoder block. Masking is applied in the first sublayer in order to prevent the decoder from attending to succeeding words. At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to attend to all the words in the input sequence.
- The output of the decoder finally passes through a fully connected layer, followed by a softmax layer, to generate a prediction for the next word of the output sequence

Comparison to RNN models

- Self-attention layers were found to be faster than recurrent layers for shorter sequence lengths
- CNN can be restricted to consider only a neighborhood in the input sequence for very long sequence lengths.
- The number of sequential operations required by a RNN is based on the sequence length
- The number of sequential operations remains constant given the self-attention mechanism
- In CNN, the kernel width directly affects the long-term dependencies between the input and output position. Tracking long-term dependencies needs larger kernels or high stacks of CNN with expensive runtime cost