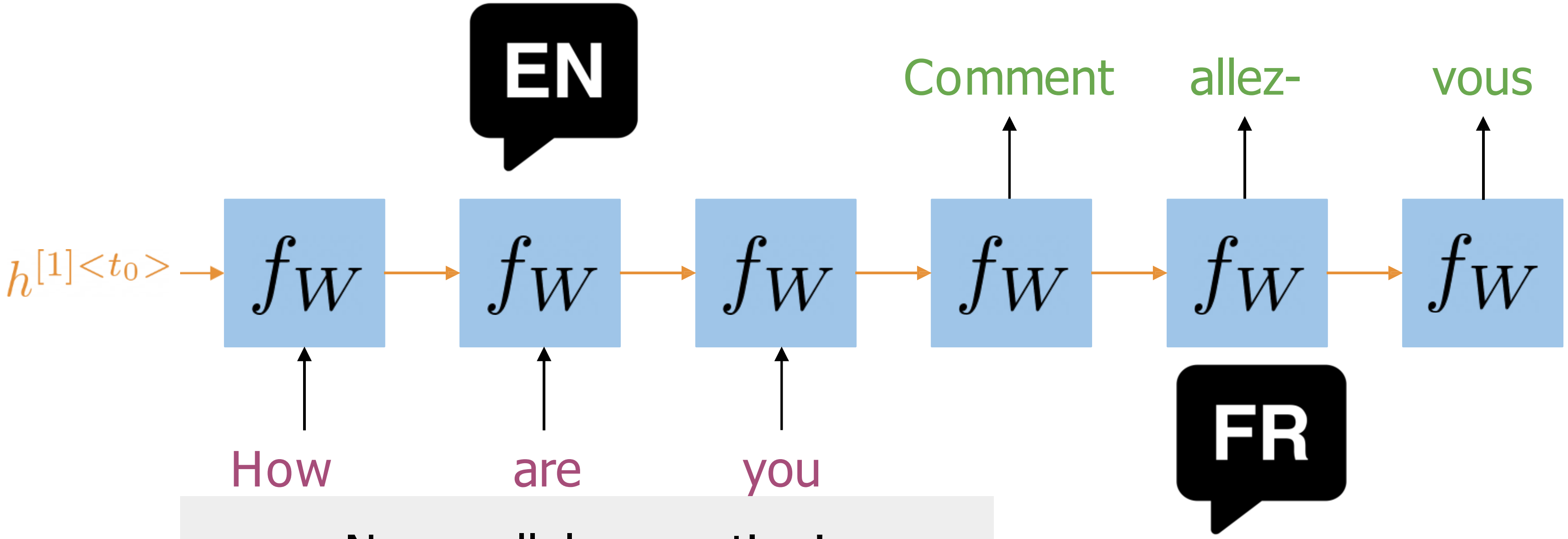
Text sumarization

- Transformers vs RNNs
- Transformers Overview
- Transformer NLP applications
- Transformer summarizer

- Transformers vs RNNs
  - Outline
  - Issues with RNNs
  - Comparison with Transformers

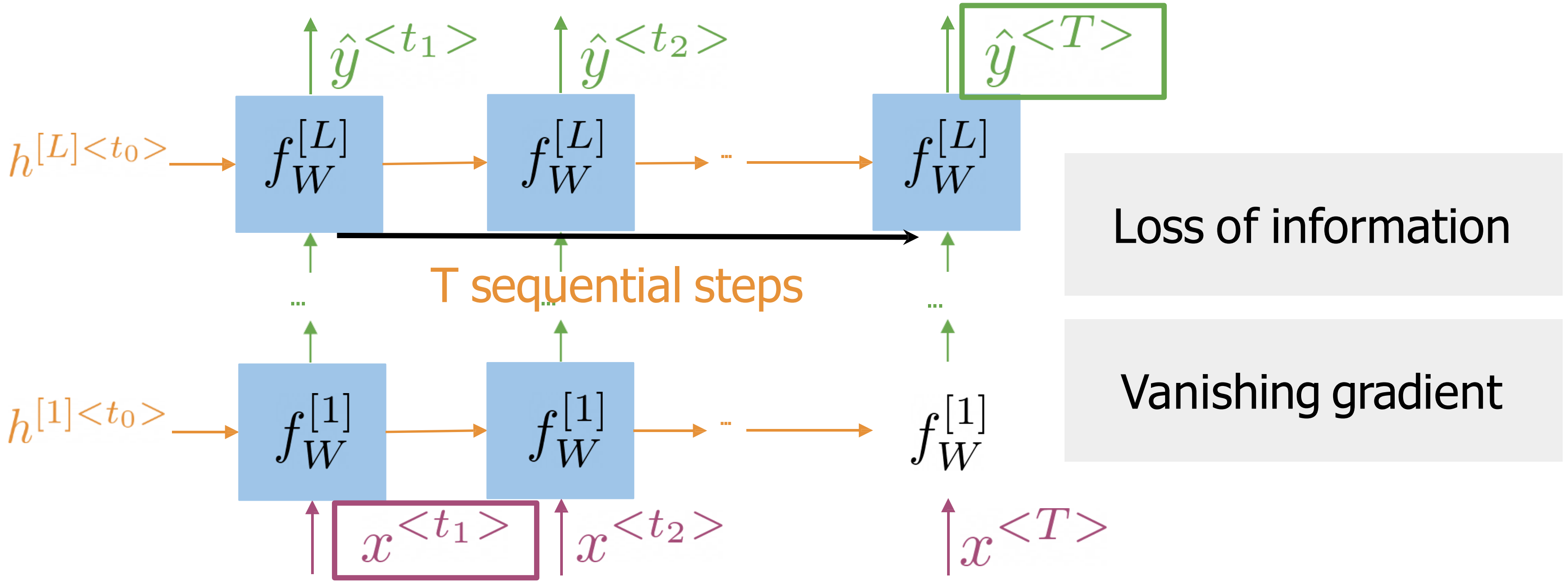
Neural Machine Translation



No parallel computing!

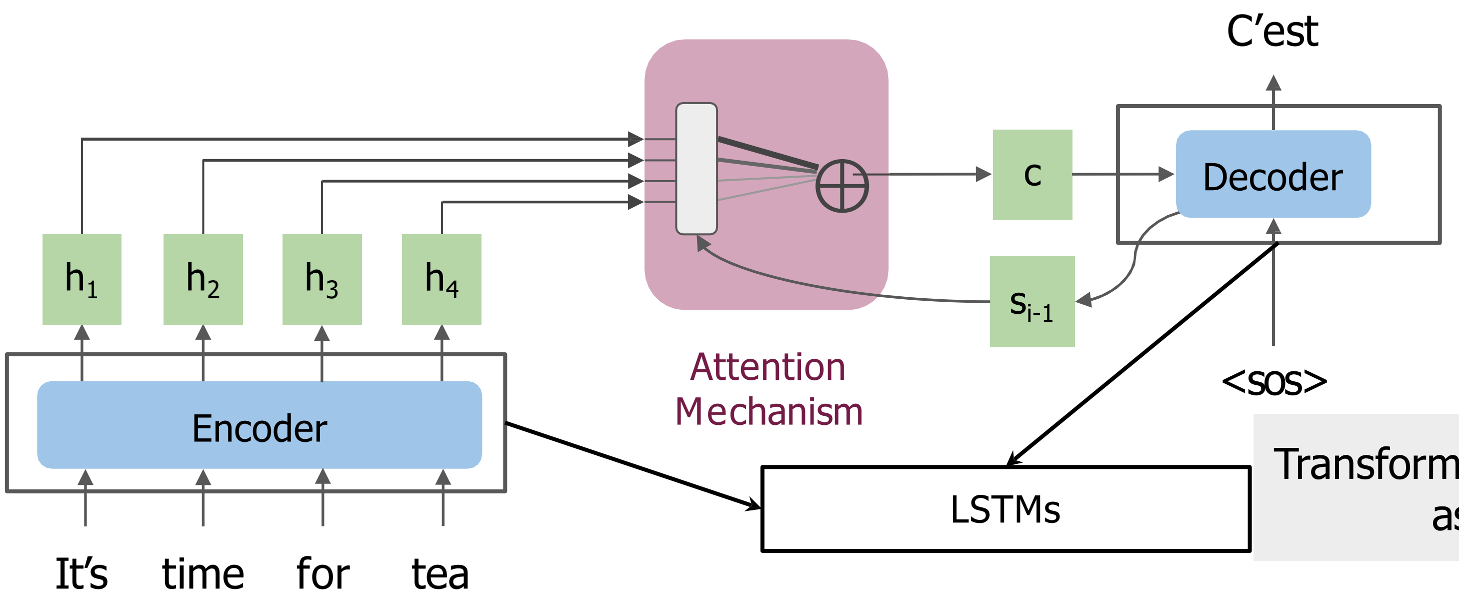
- inputs making computations at every step until the end
- decode the information following a similar sequential procedure
- the more words have in the input sentence, the more time take to process that sentence

Seq2Seq Architectures



- the model will take T times steps to encode their sentence
- the information tends to get lost within the network and vanish ingredients problems arise related to the length of your import sequences.

RNNs vs Transformer: Encoder-Decoder



Transformers **don't** use RNNs, such as LSTMs or GRUs

- LSTMs for your encoder and decoder but you could also have used GRUs or just vanilla RNNs.
- In contrast, transformers rely only on attention mechanisms and don't require the use of recurrent networks

The Transformer Model

FPT Education
FPT UNIVERSITY

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

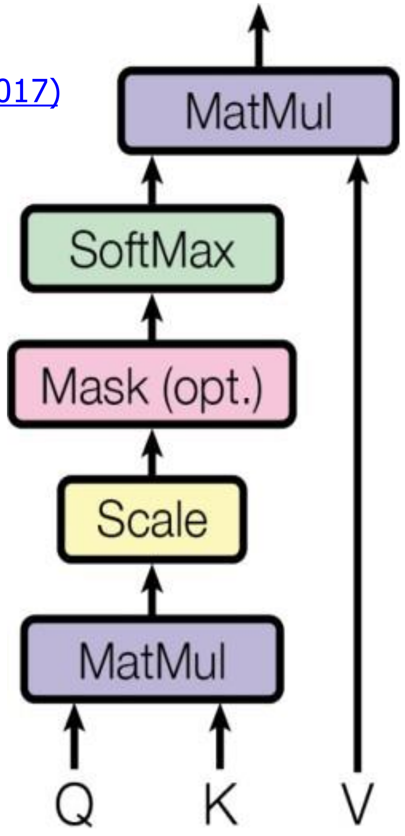
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

https://arxiv.org/abs/1706.03762

Scaled Dot-Product Attention

(Vaswani et al., 2017)

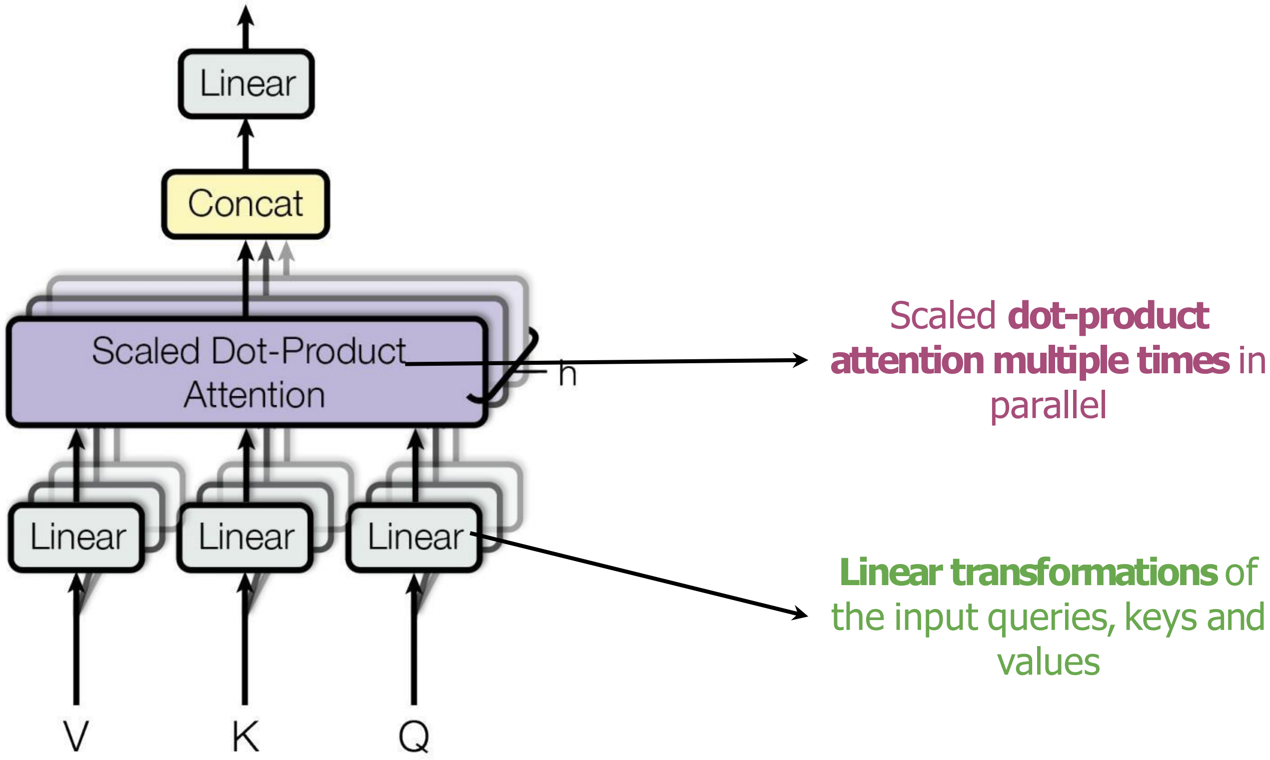


Queries Keys Values

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

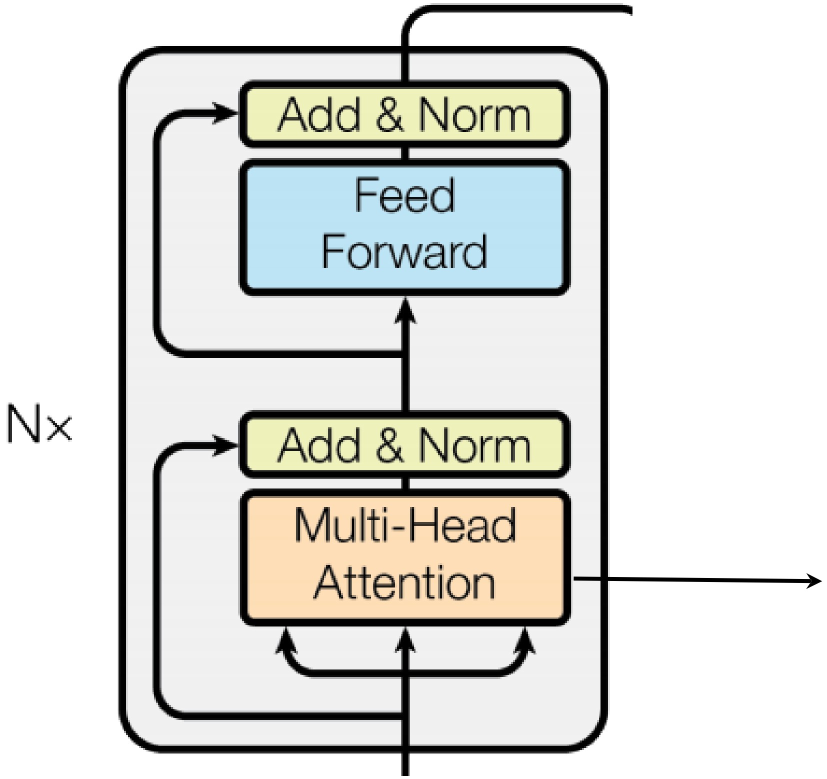
- uses scaled dot-product attention
- very efficient in terms of computation and memory
- consisting of just matrix multiplication operations.
- it allows the transformer to grow larger, and more complex while being faster, and using less memory

Multi-Head Attention



- a number of scaled dot-product attention mechanisms at multiple linear transformations of the inputs, queries, keys, and values.
- In this layer, the linear transformations are learnable parameters

The Encoder

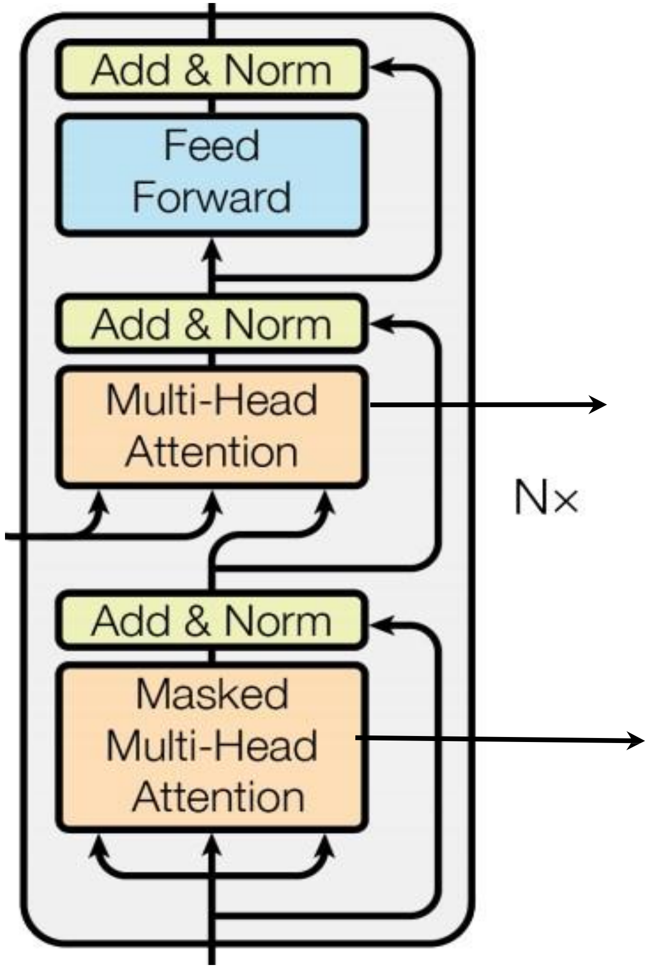


Provides contextual representation of each item in the input sequence

Self-Attention

Every item in the input attends to every other item in the sequence

The Decoder



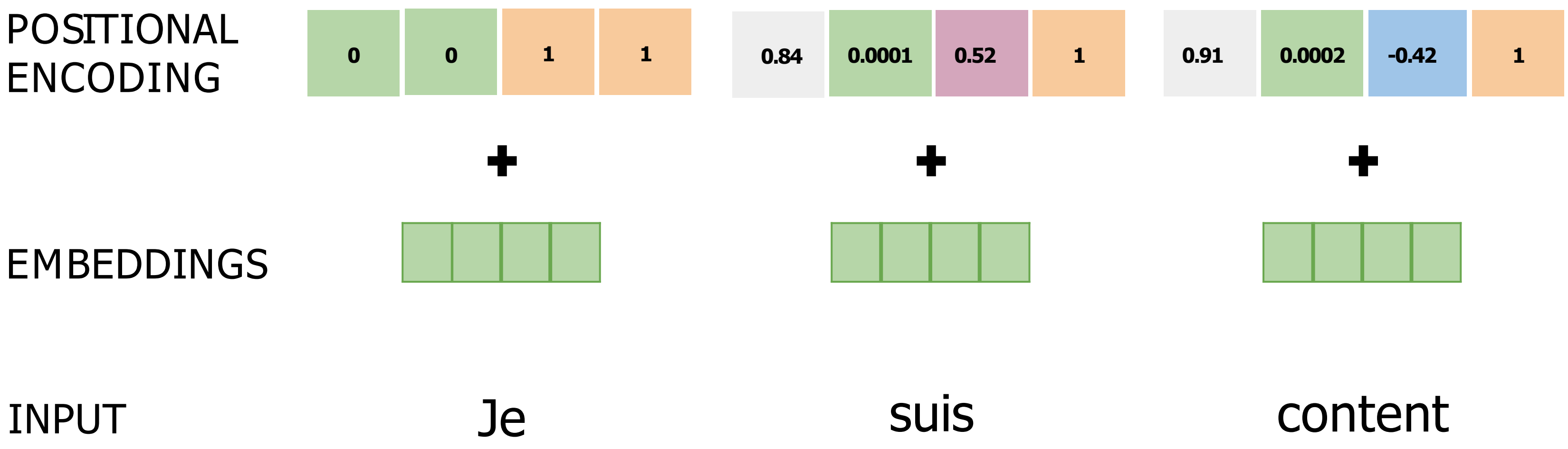
Encoder-Decoder Attention

Every position from the decoder attents to the outputs from the encoder

Masked Self-Attention

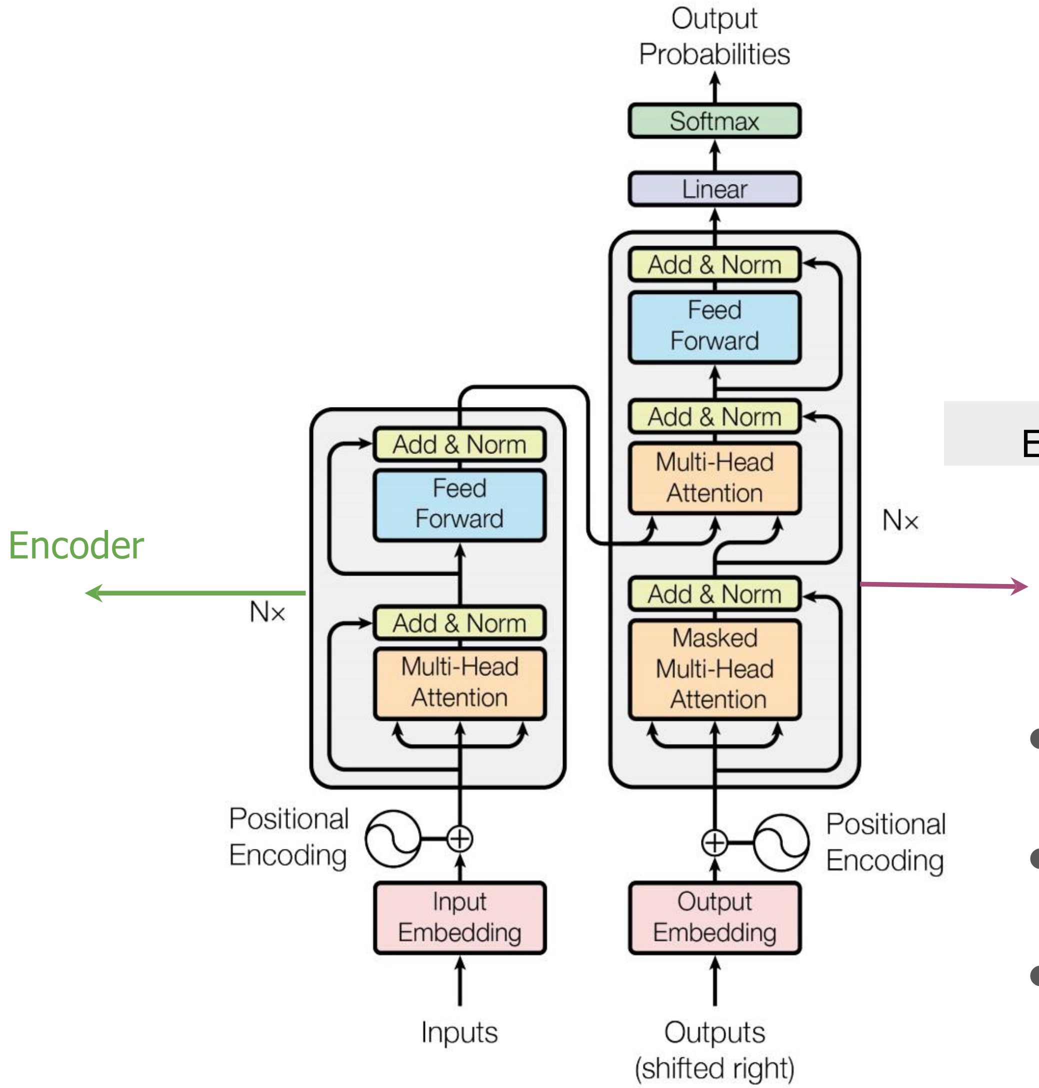
Every position attends to **previous** positions

RNNs vs Transformer: Positional Encoding



- transformers incorporate a positional encoding stage, which encodes each inputs position in the sequence.
- transformers don't use recurrent neural networks.
- positional encoding can be learned or fixed.

The Transformer



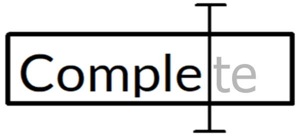
Easy to parallelize!

- In RNNs parallel computing is difficult to implement
- For long sequences in RNNs there is loss of information
- In RNNs there is the problem of vanishing gradient
- Transformers help with all of the above

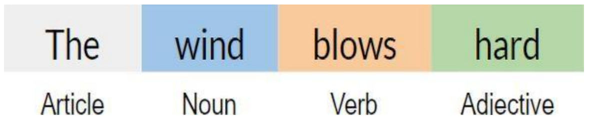
Transformer NLP applications

- Transformers applications in NLP
- Some Transformers
- Introduction to T5

Text summarization



Auto-Complete



Named entity recognition (NER)

Question answering (Q&A)

Translation

Chat-bots

Other NLP tasks

Sentiment Analysis
Market Intelligence
Text Classification
Character Recognition
Spell Checking

State of the Art Transformers

Radford, A., et al. (2018)
Open AI

Devlin, J., et al. (2018)
Google AI Language

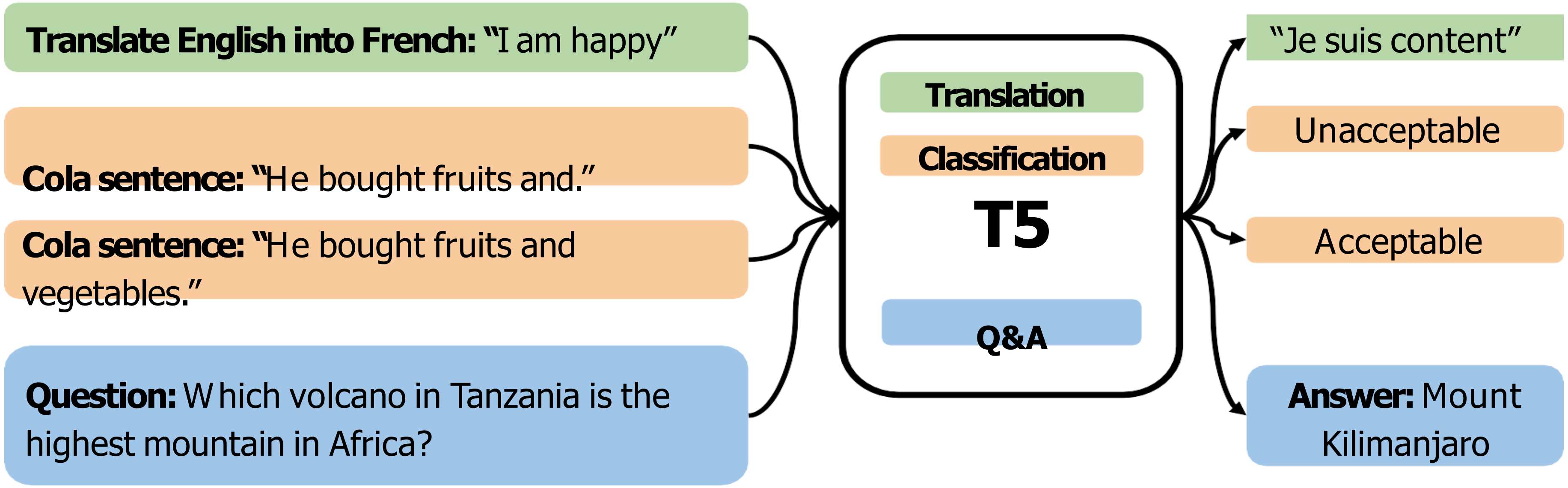
Colin, R., et al. (2019) Google

GPT-2: Generative Pre-training for Transformer

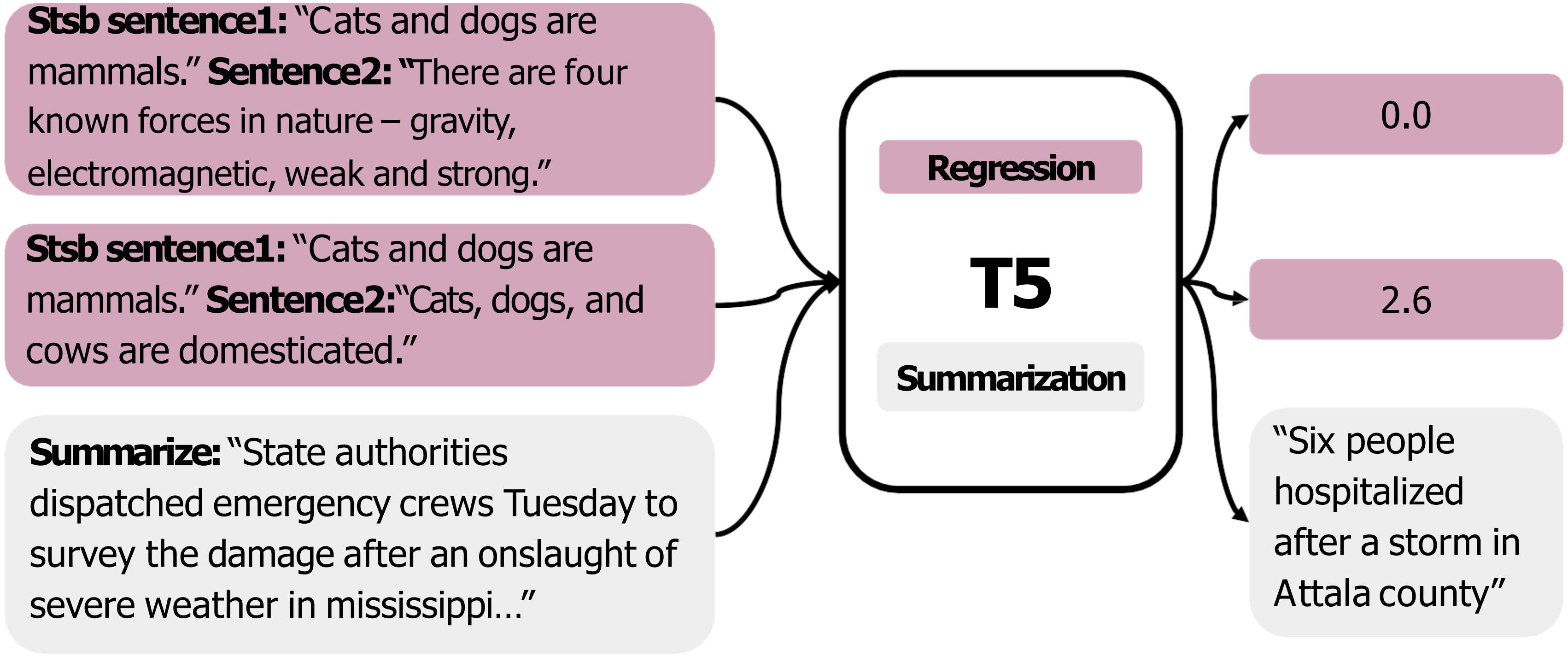
BERT: Bidirectional Encoder Representations from Transformers

T5: Text-to-text transfer transformer

- T5: Text-To-Text Transfer Transformer



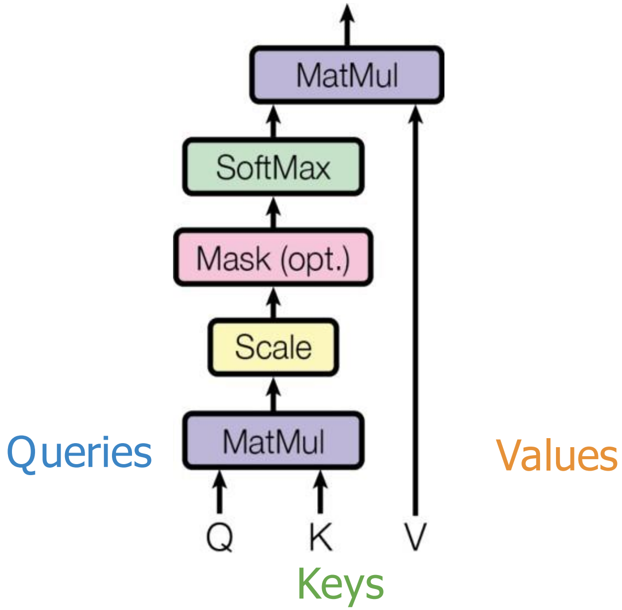
T5: Text-To-Text Transfer Transformer



- Transformers are suitable for a wide range of NLP applications
- GPT-2, BERT and T5 are the cutting-edge Transformers
- T5 is a powerful multi-task transformer

Scaled dot-product attention

- Revisit scaled dot product attention
- Mathematics behind Attention



(Vaswani et al., 2017)

Weights add up to 1

Improves performance

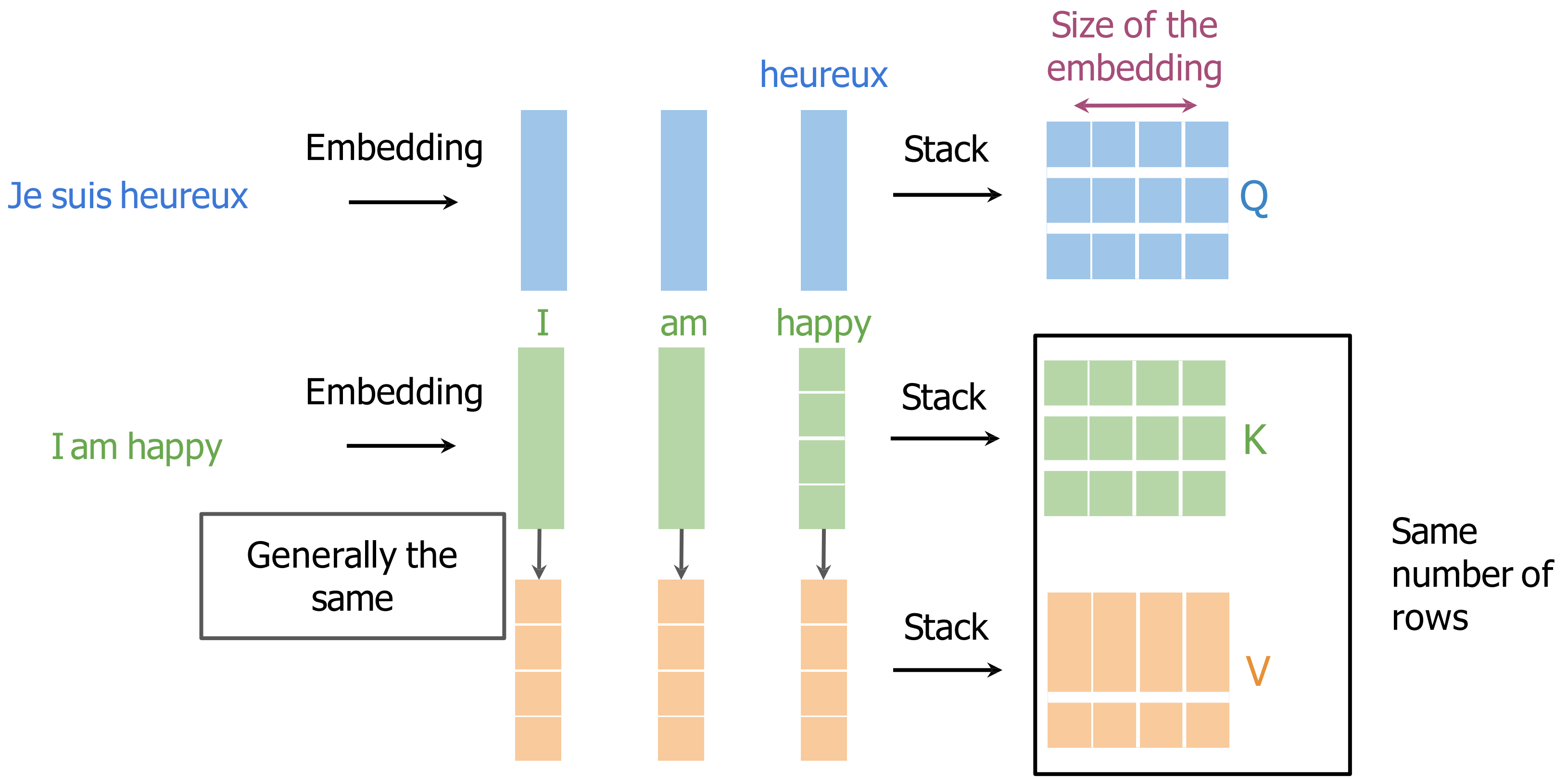
$$\mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Weighted sum of values V

Just two matrix multiplications and a Softmax!

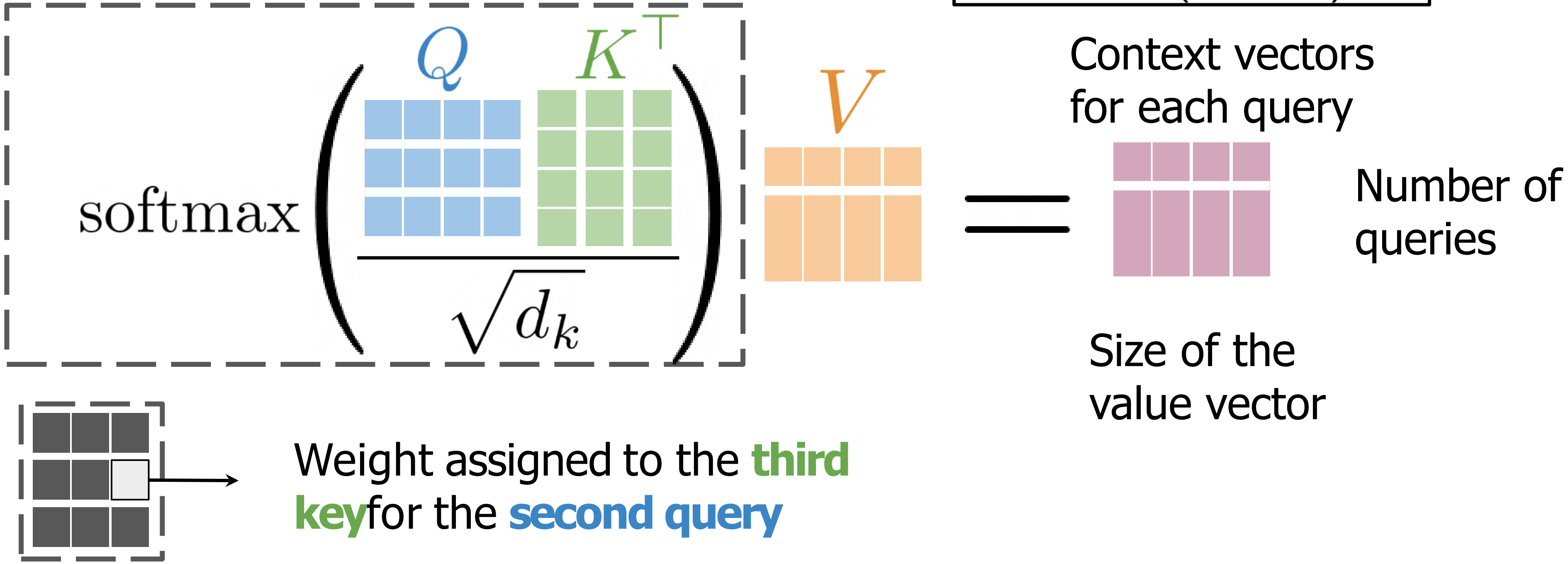
Queries, Keys and Values

Je suis



Attention Math

$$\boxed{\mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V}$$

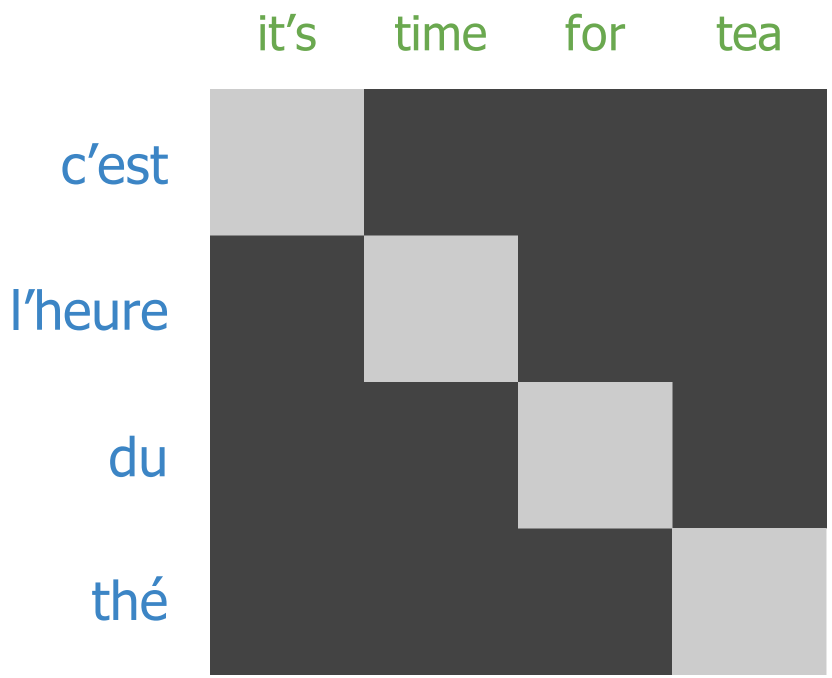


- Scaled Dot-product Attention is essential for Transformer
- The input to Attention are queries, keys, and values
- Using GPUs and TPUs to speed up the training of models

Masked Self-Attention

- Encoder-Decoder Attention
  - Ways of Attention
  - Overview of masked Self-Attention

Queries from one sentence, keys and values from another



Weight matrix

Self-Attention

Queries, keys and values come from the **same sentence**

Weight matrix

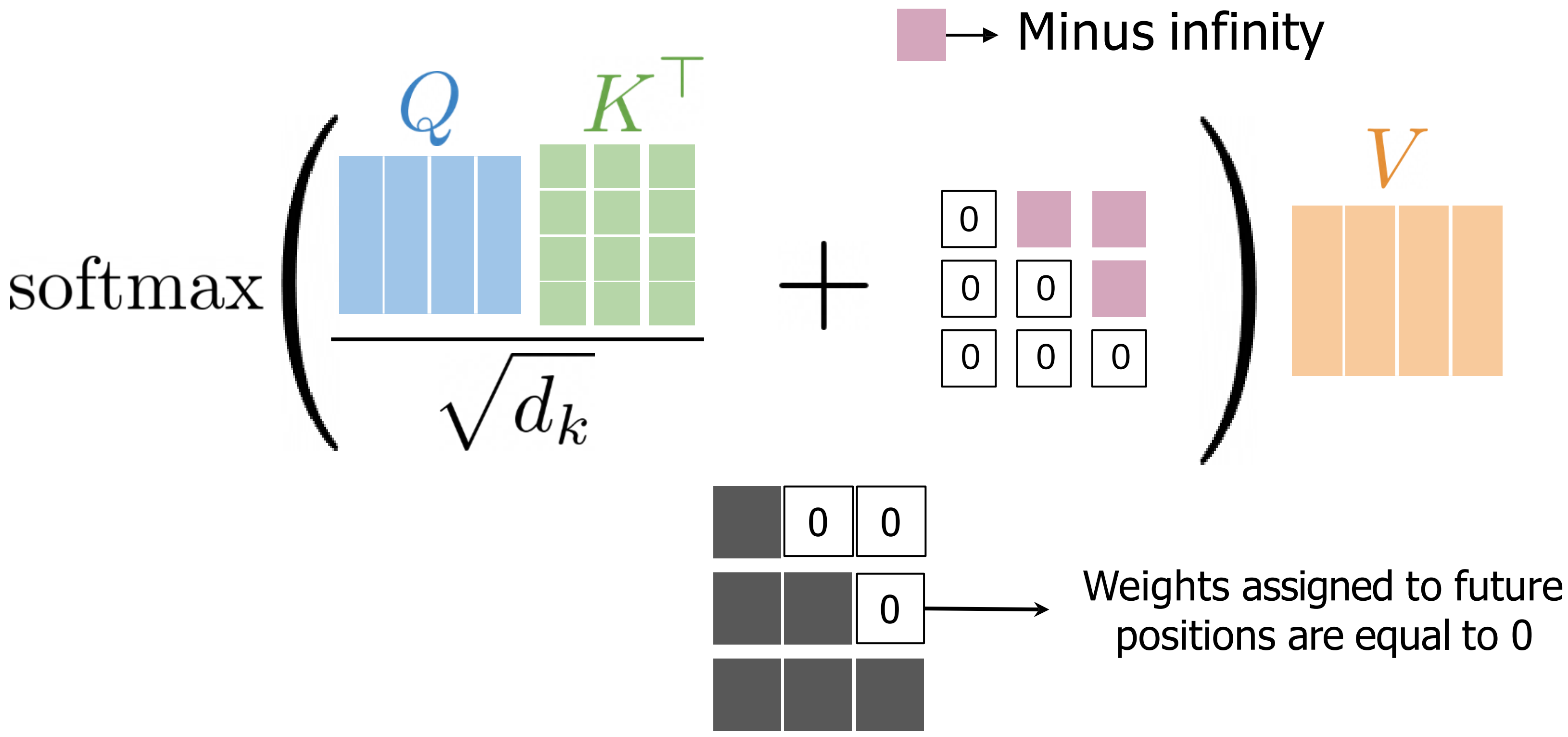
Meaning of each word **within** the sentence

Masked Self-Attention

Queries, keys and values come from the **same sentence**. Queries don't attend to future positions.

Weight matrix

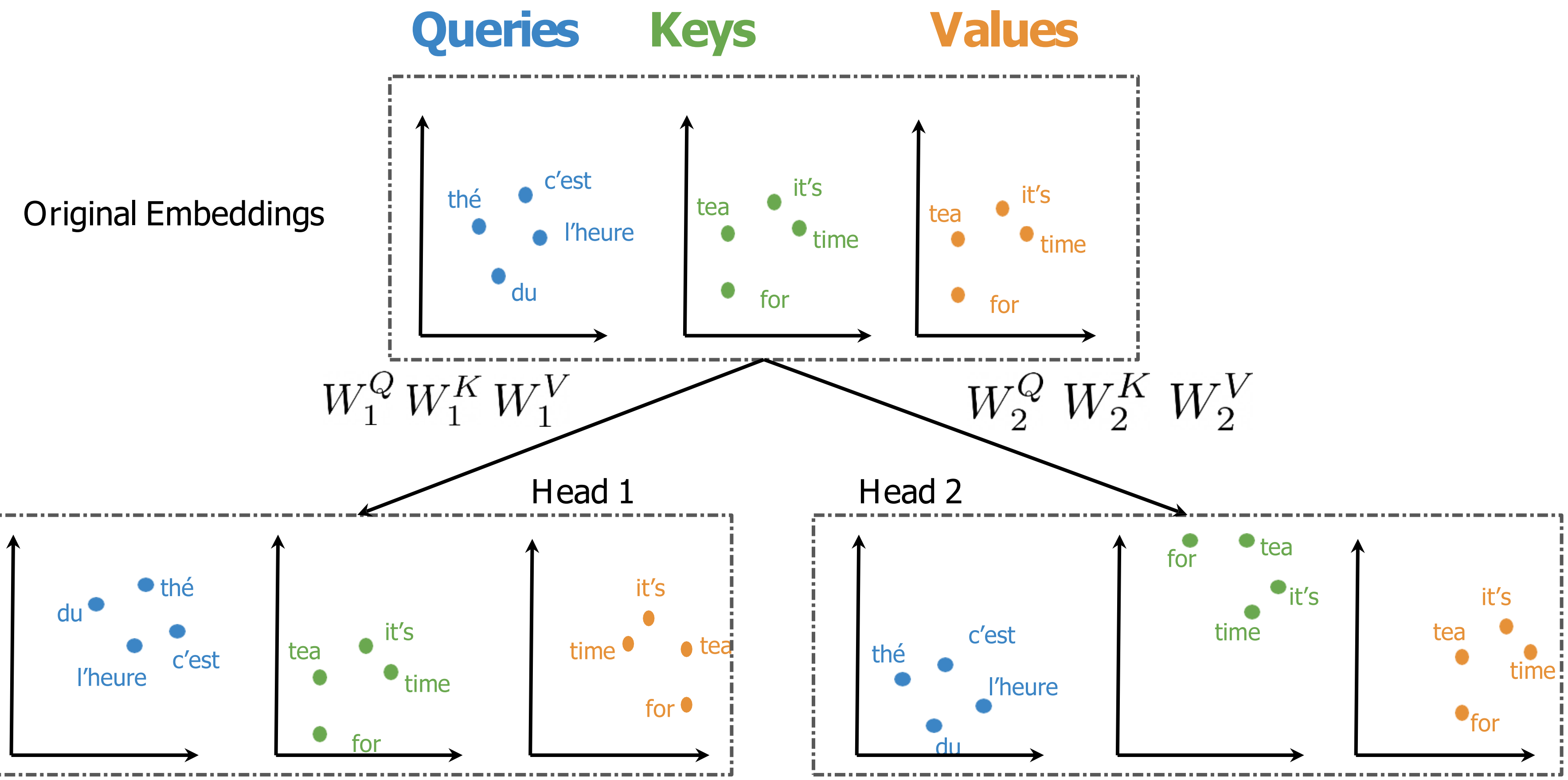
Masked self-attention math



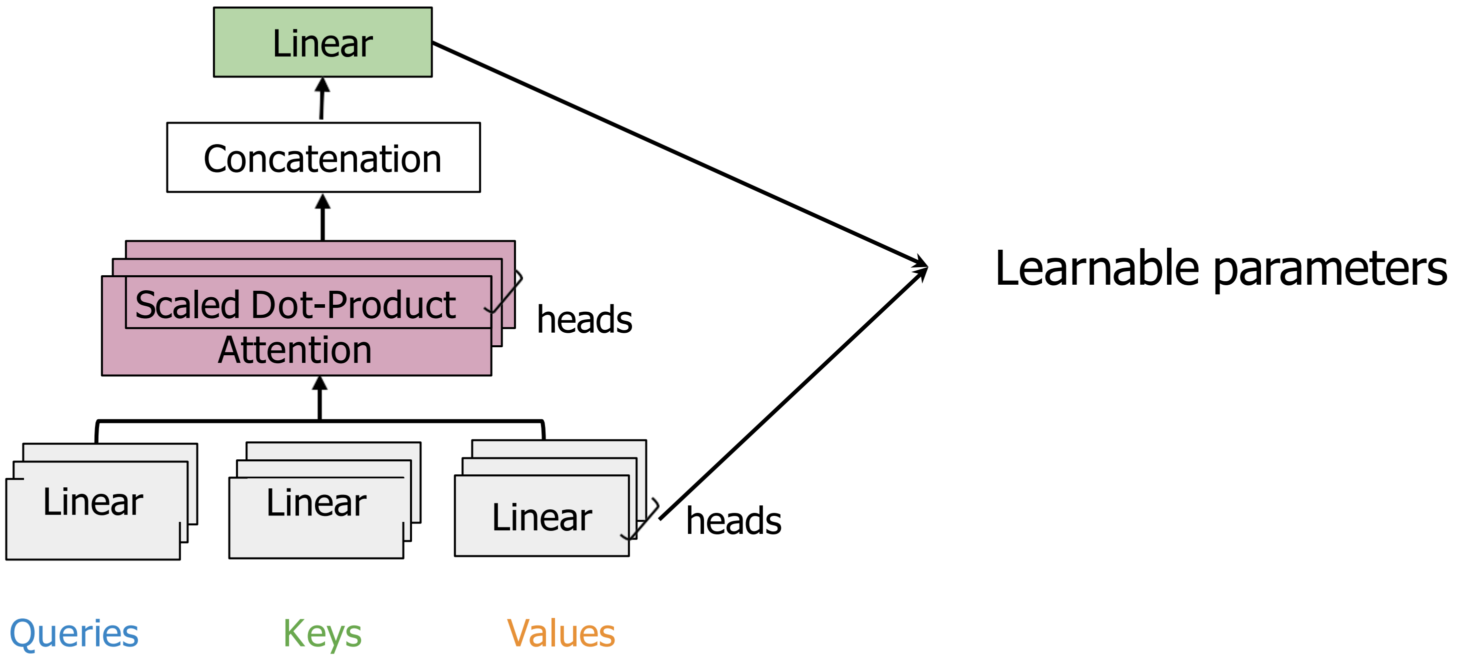
- There are three main ways of Attention: Encoder/Decoder, self- attention and masked self-attention.
- In self-attention, queries and keys come from the same sentence
- In masked self-attention queries cannot attend to the future

Multi-Head Attention

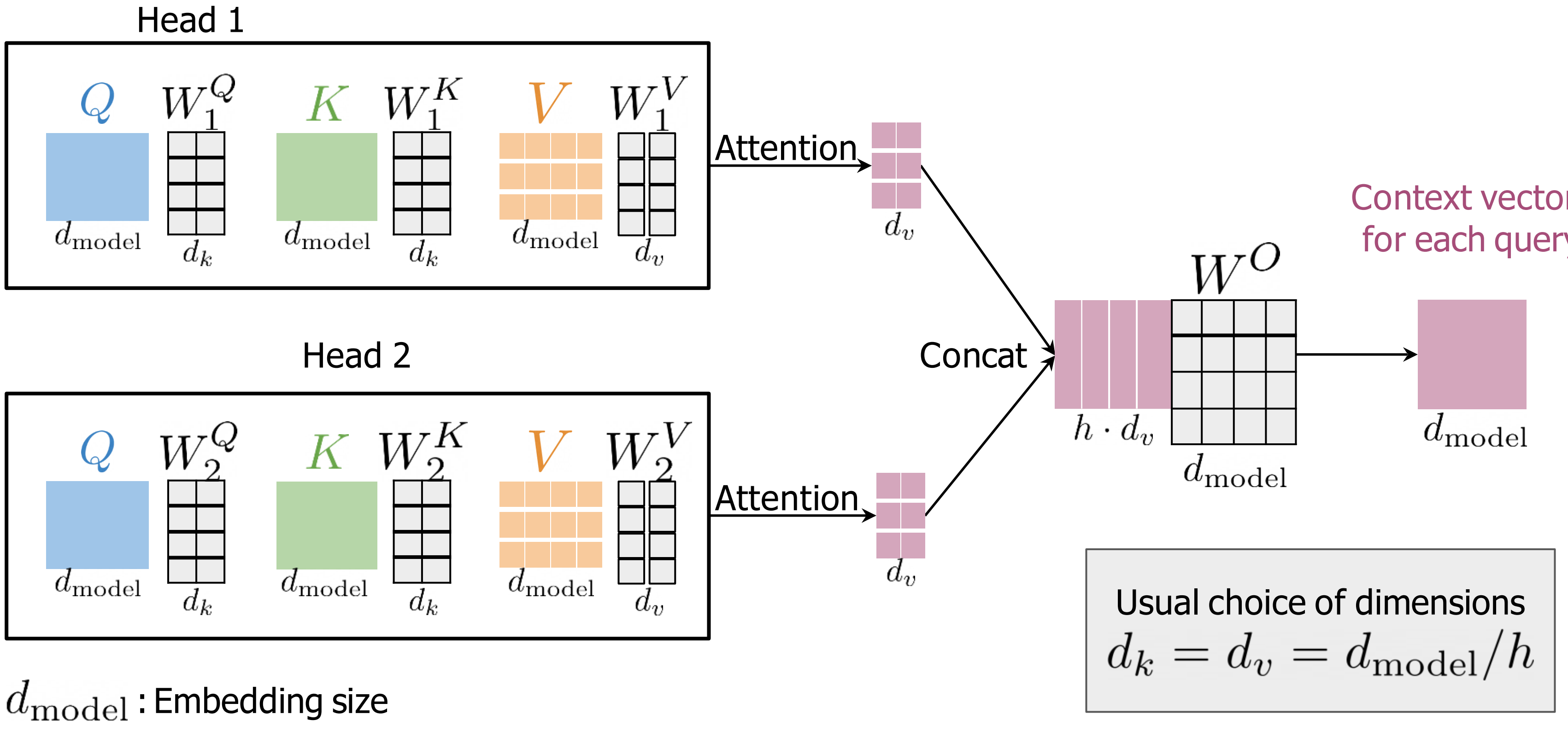
- Intuition Multi-Head Attention
- Math of Multi-Head Attention



Multi-Head Attention - Overview

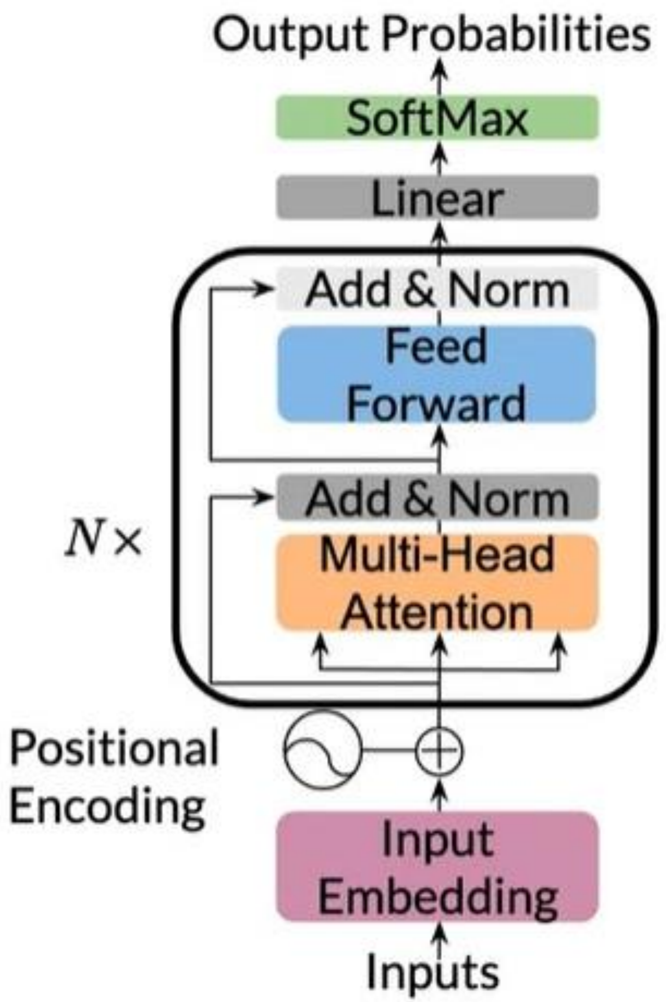


Multi-Head Attention



- Multi-Headed models attend to information from different representations
- Parallel computations
- Similar computational cost to single-head attention

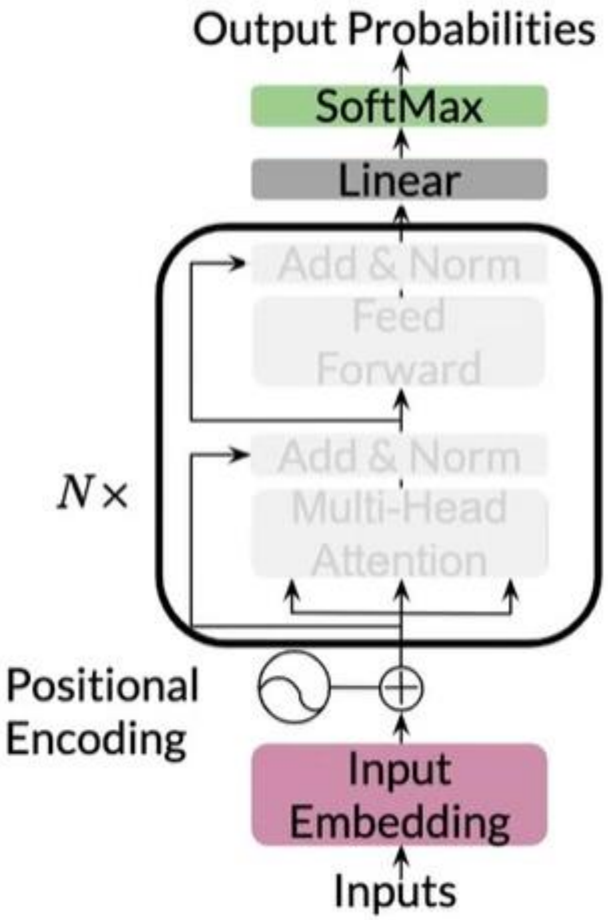
Transformer decoder

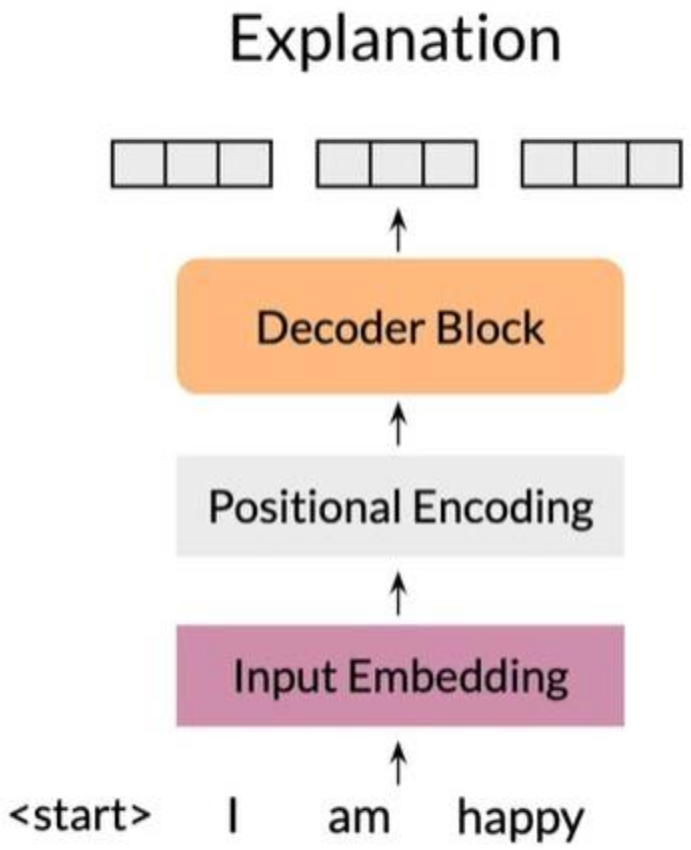


Overview

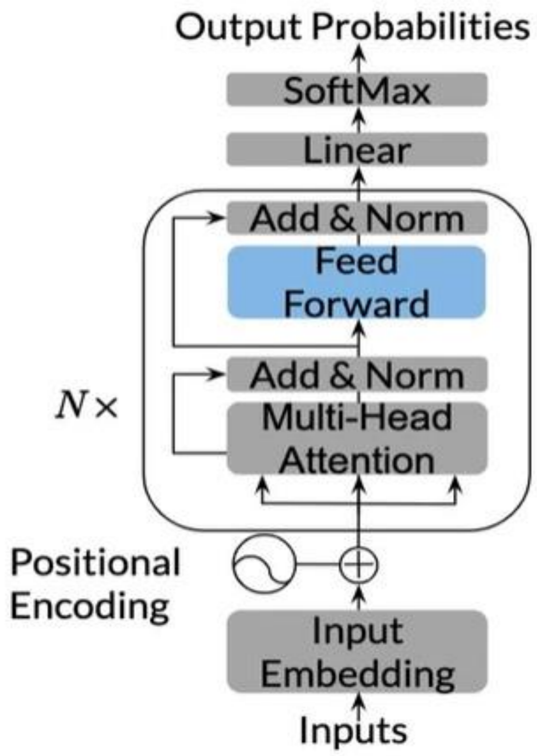
- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization
- repeat N times
- dense layer and softmax for output

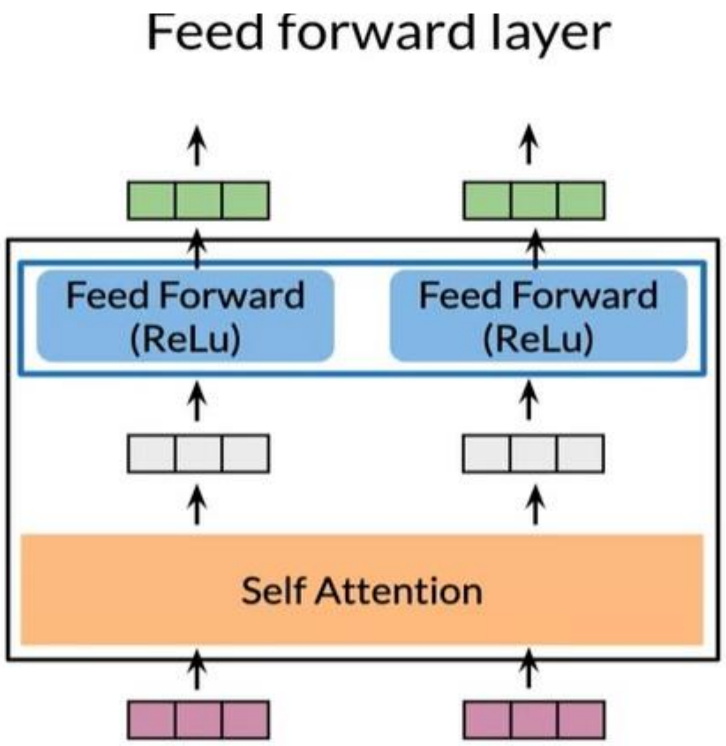
Transformer decoder





The Transformer decoder

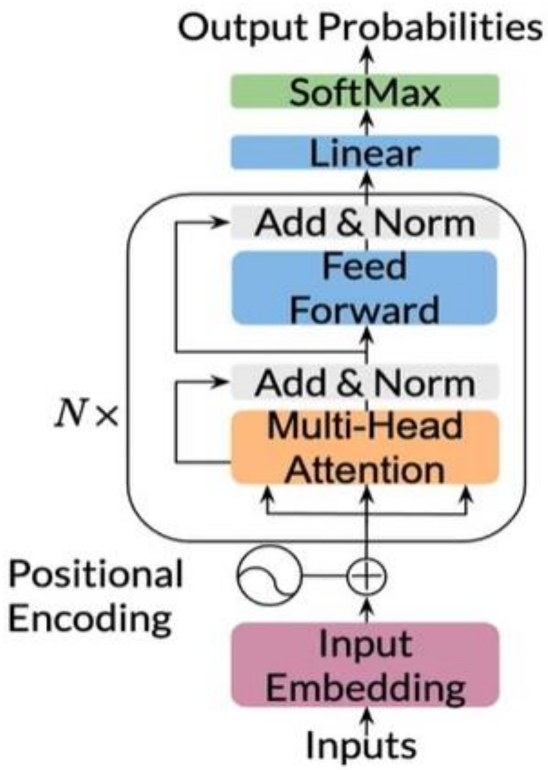


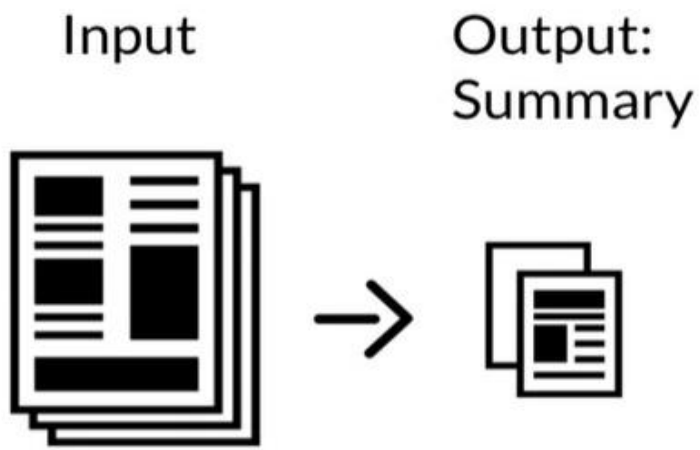


- Transformer decoder mainly consists of three layers
- Decoder and feed-forward blocks are the core of this model code
- It also includes a module to calculate the cross-entropy loss

Transformer for summarization

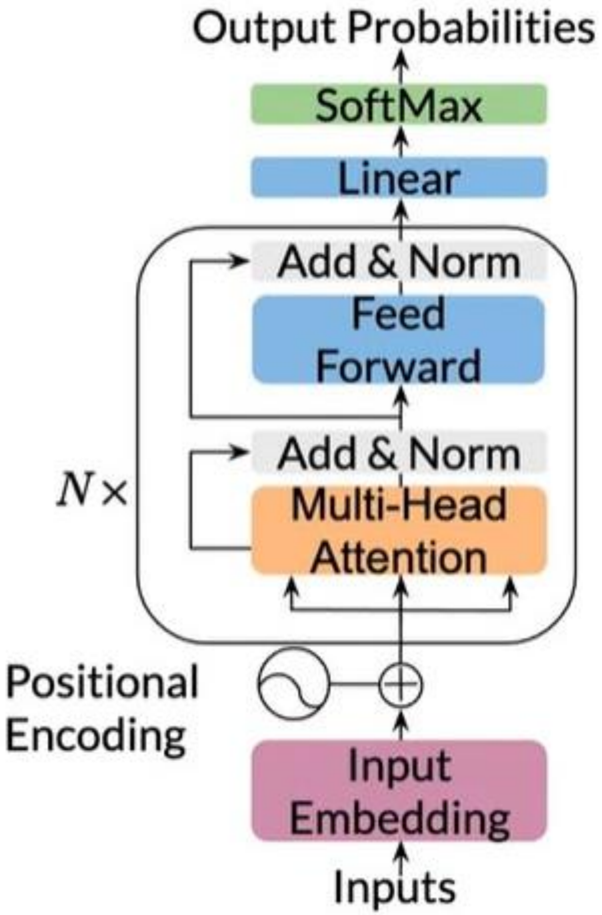
- Overview of Transformer summarizer
- Technical details for data processing
- Inference with a Language Model





Technical details for data processing

FPT Education
FPT UNIVERSITY



Model Input:

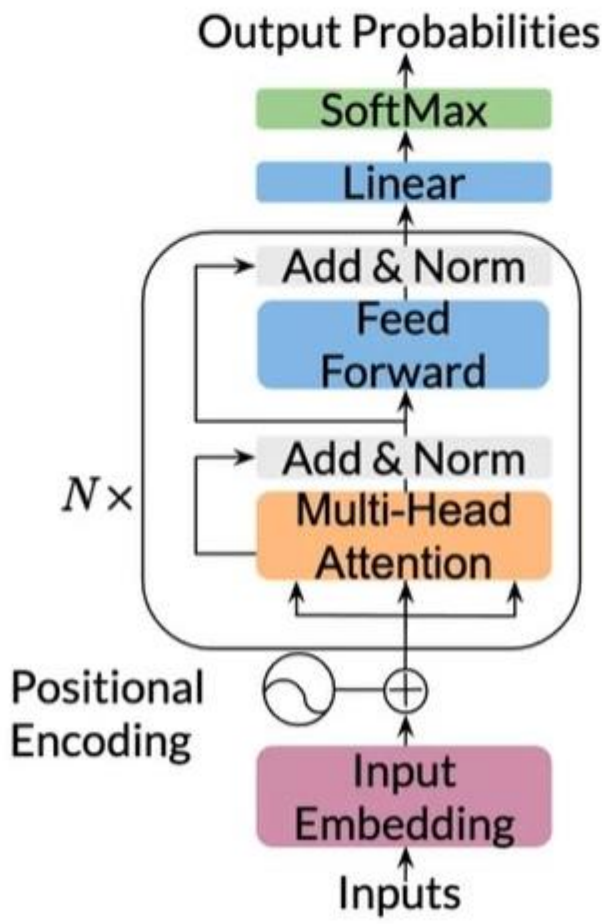
```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

Tokenized version:

```
[2,3,5,2,1,3,4,7,8,2,5,1,2,3,6,2,1,0,0]
```

Loss weights: 0s until the first `<EOS>` and then 1 on the start of the summary.

Cost function



Cross entropy loss

$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

j : over summary

i : bach elements

Inference with a Language Model

Model input:

```
[Article] <EOS>  [Summary] <EOS>
```

Inference:

- Provide: [Article] <EOS>

- Generate summary word-by-word
  - until the final <EOS>

- Pick the next word by random sampling
  - each time you get a different summary!

- For summarization, a weighted loss function is optimized
- Transformer Decoder summarizes predicting the next word
- The transformer uses tokenized versions of the input