Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.
For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode.

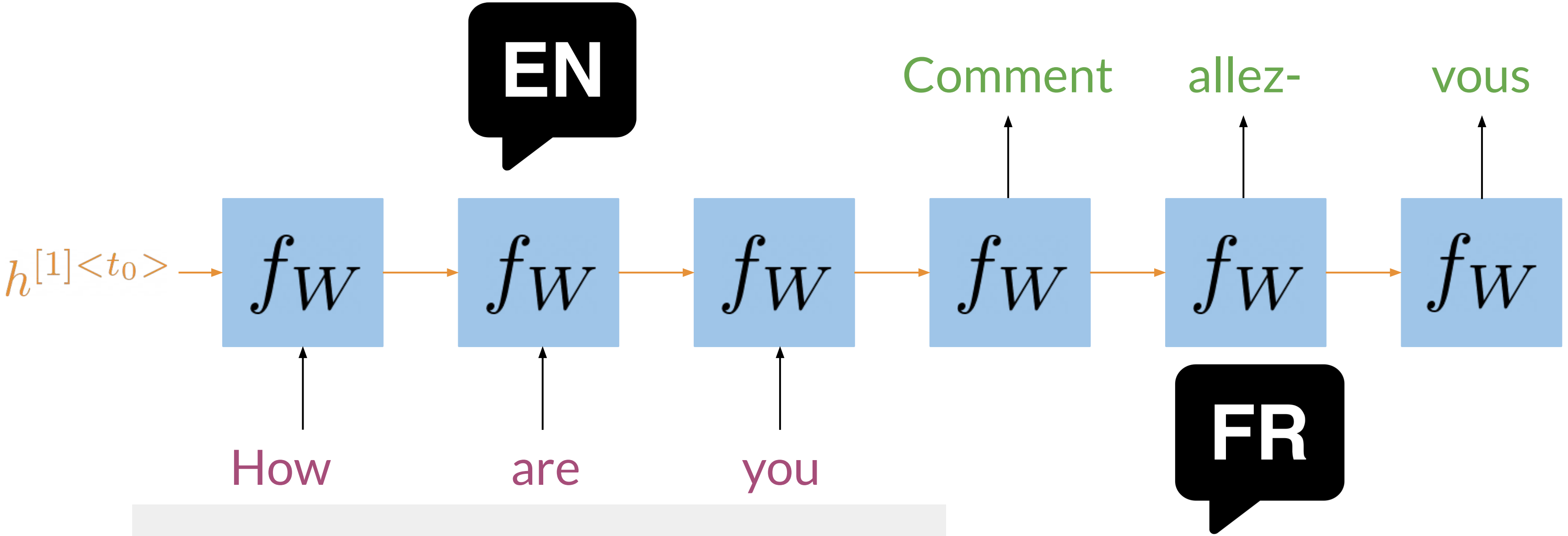
Transformers vs RNNs

deeplearning.ai

Outline

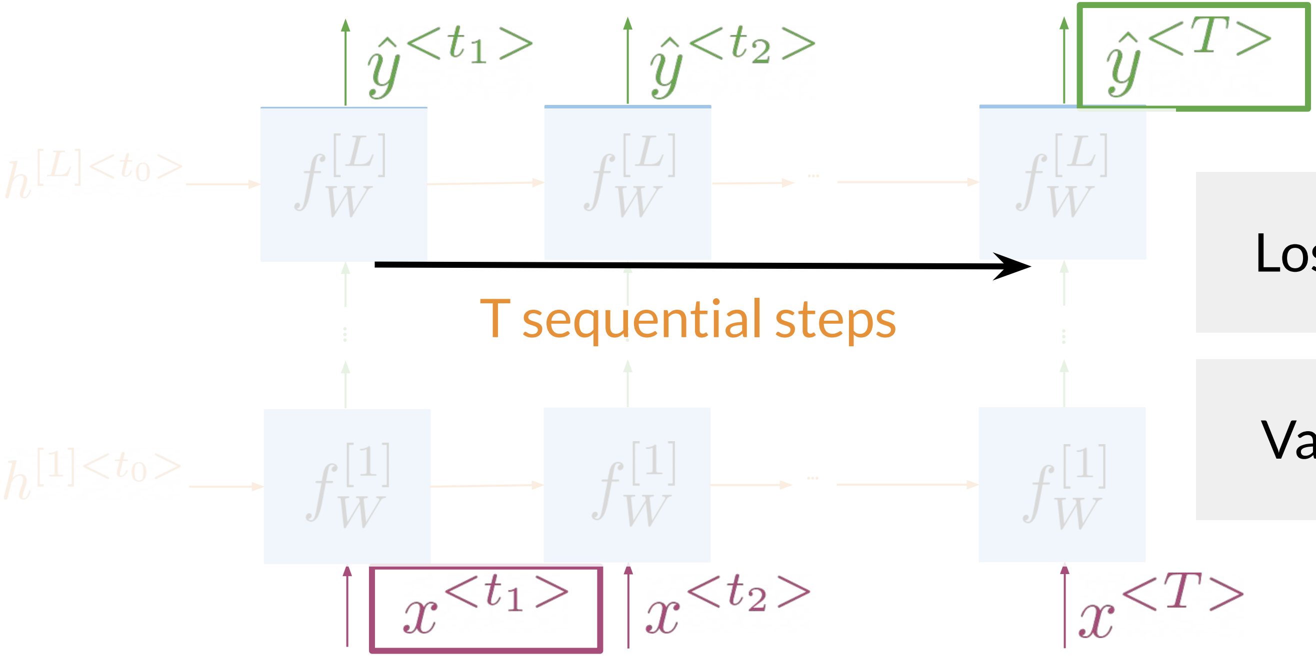
- Issues with RNNs
- Comparison with Transformers

Neural Machine Translation



No parallel computing!

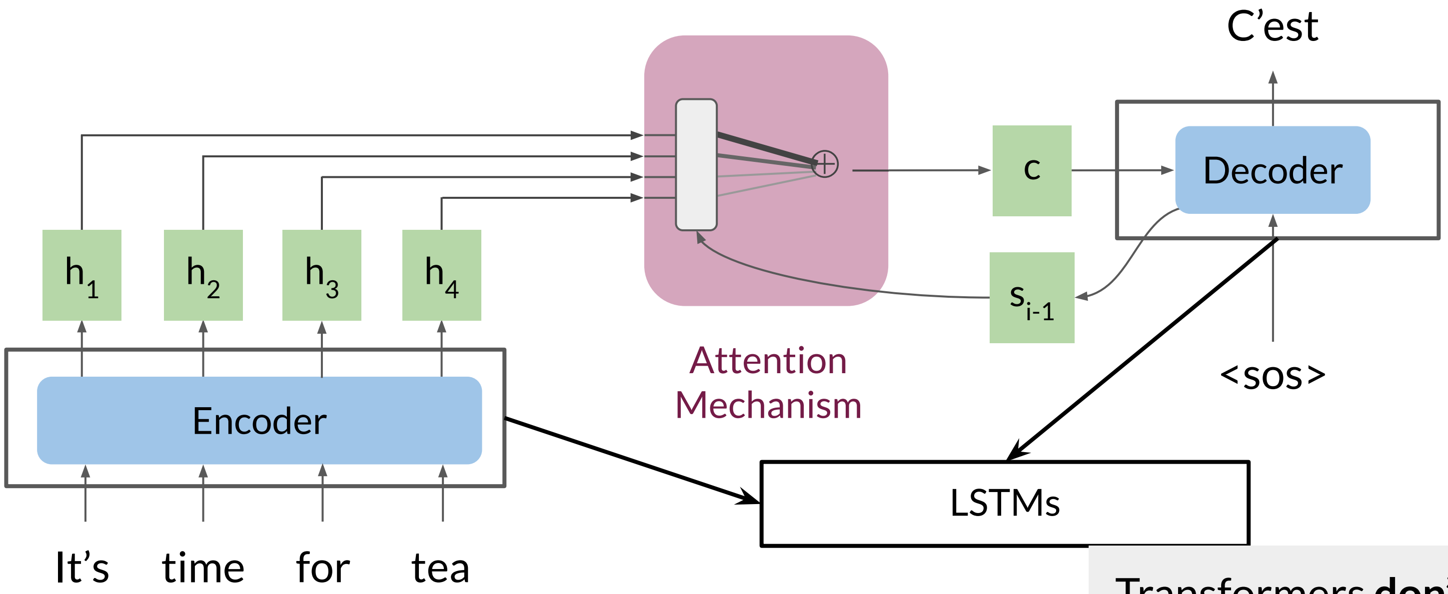
Seq2Seq Architectures



Loss of information

Vanishing gradient

RNNs vs Transformer: Encoder-Decoder



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

deeplearning.ai

Transformers Overview

The Transformer Model

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

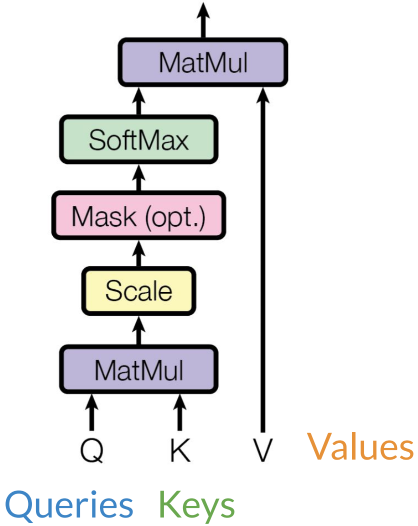
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

https://arxiv.org/abs/1706.03762

https://youtu.be/zxQyTK8quyY?feature=shared

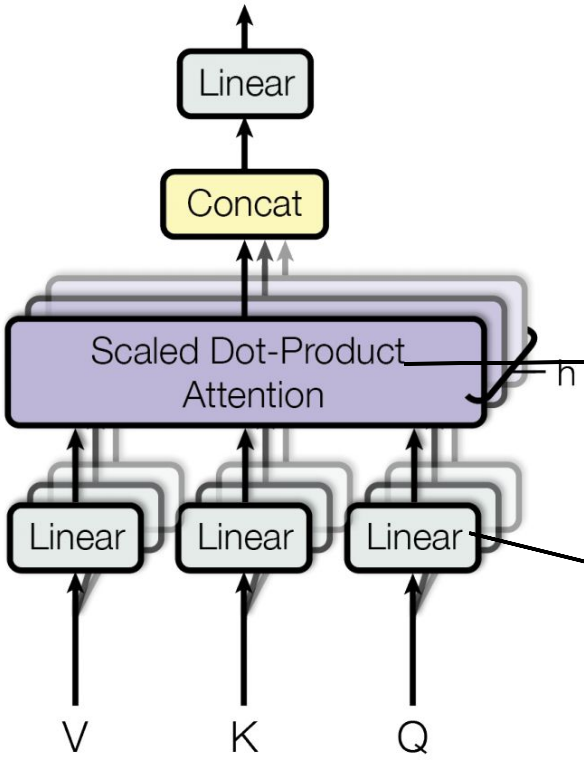
Scaled Dot-Product Attention



(Vaswani et al., 2017)

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

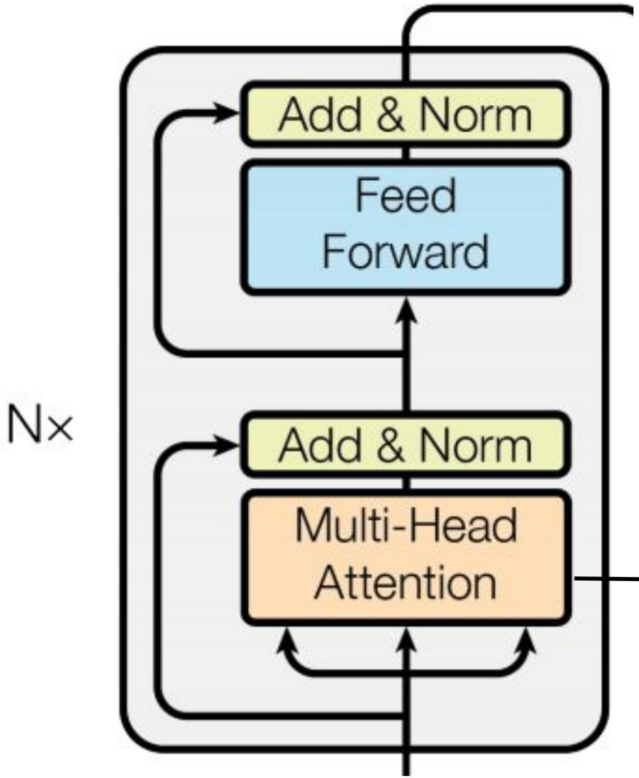
Multi-Head Attention



Scaled **dot-product attention multiple times** in parallel

Linear transformations of the input queries, keys and values

The Encoder

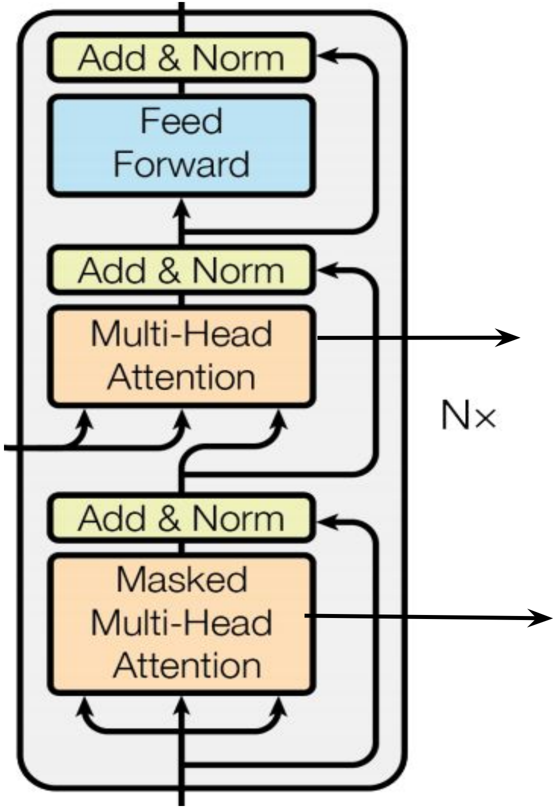


Provides contextual representation of each item in the input sequence

Self-Attention

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

The Decoder



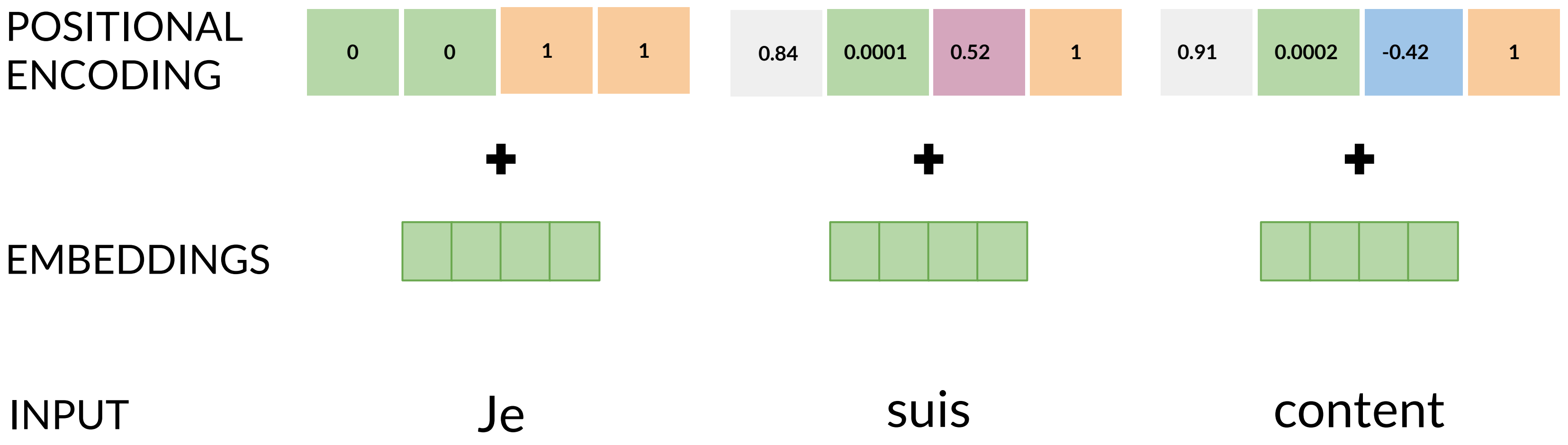
Encoder-Decoder Attention

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

Masked Self-Attention

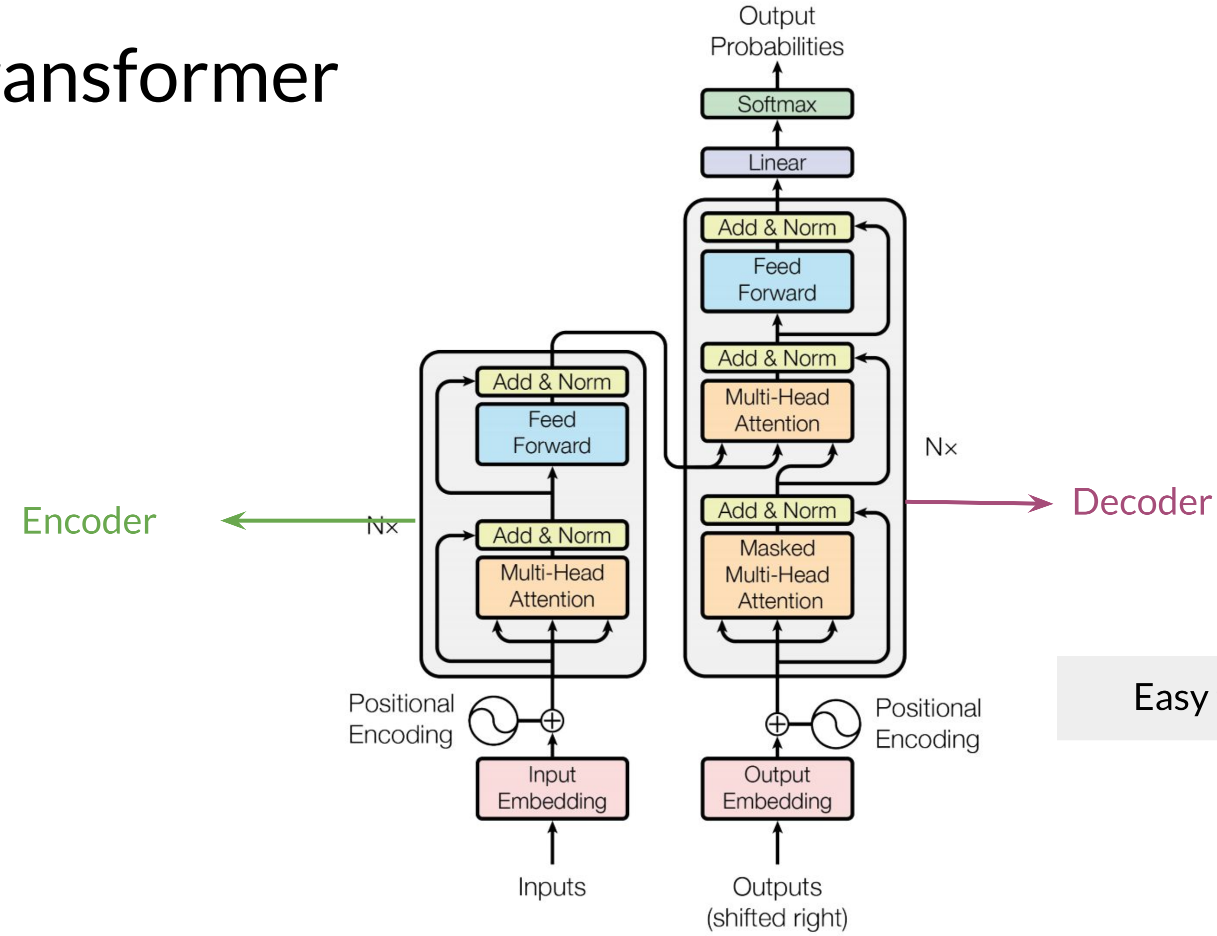
Every position attends to **previous** positions

RNNs vs Transformer: Positional Encoding



deeplearning.ai

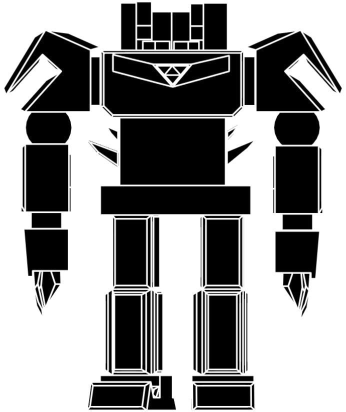
The Transformer



Easy to parallelize!

Summary

- 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

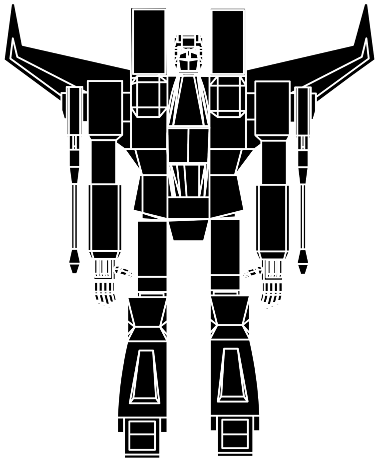


deeplearning.ai

Transformer Applications

Outline

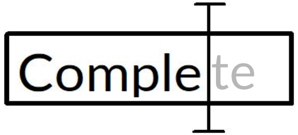
- Transformers applications in NLP
- Some Transformers
- Introduction to T5



deeplearning.ai

Transformer NLP applications

Text summarization



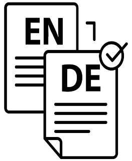
Auto-Complete

The	wind	blows	hard
Article	Noun	Verb	Adjective

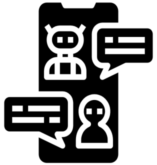
Named entity recognition (NER)

Question answering (Q&A)

Translation



Chat-bots



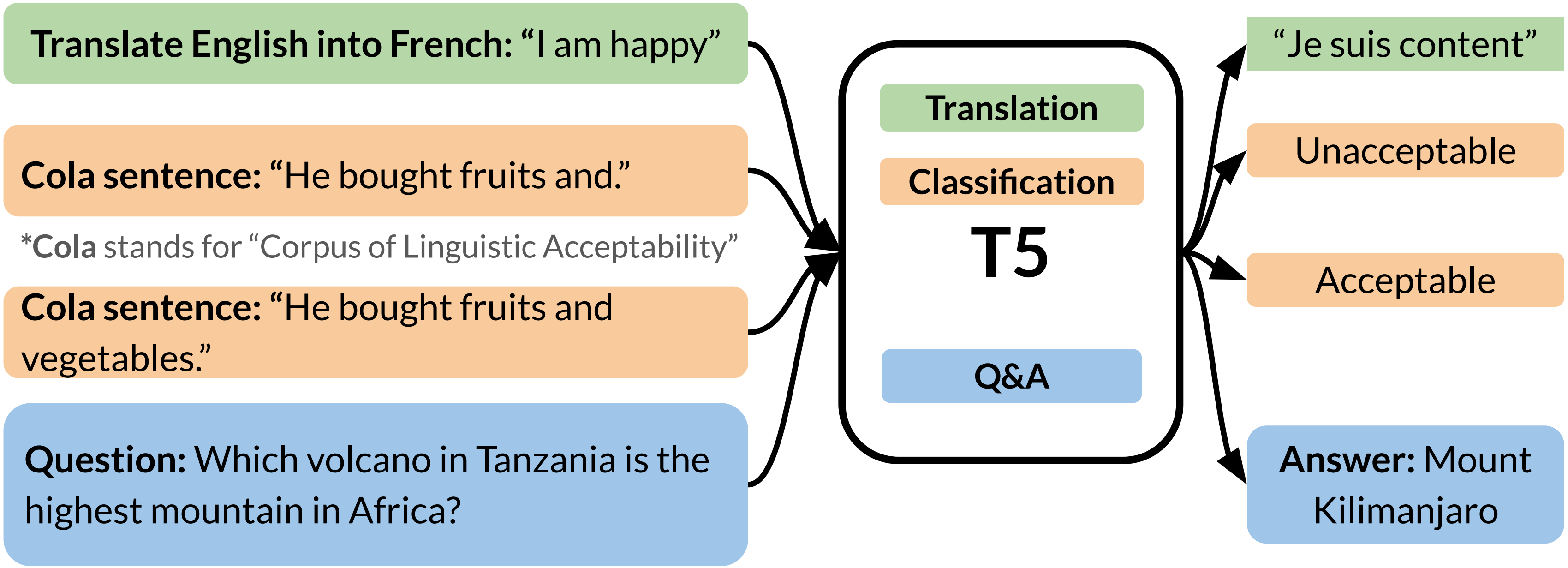
Other NLP tasks

- Sentiment Analysis
- Market Intelligence
- Text Classification
- Character Recognition
- Spell Checking

State of the Art Transformers

Radford, A., et al. (2018) Open AI	GPT-2: Generative Pre-training for Transformer
Devlin, J., et al. (2018) Google AI Language	BERT:Bidirectional Encoder Representations from Transformers
Colin, R., et al. (2019) Google	T5: Text-to-text transfer transformer

T5: Text-To-Text Transfer Transformer



T5: Text-To-Text Transfer Transformer

Stsb sentence1: "Cats and dogs are mammals." **Sentence2:** **"**There are four known forces in nature – gravity, electromagnetic, weak and strong."

Stsb sentence1: "Cats and dogs are mammals." **Sentence2:**"Cats, dogs, and cows are domesticated."

Summarize: "State authorities dispatched emergency crews Tuesday to survey the damage after an onslaught of severe weather in mississippi…"

Regression

T5

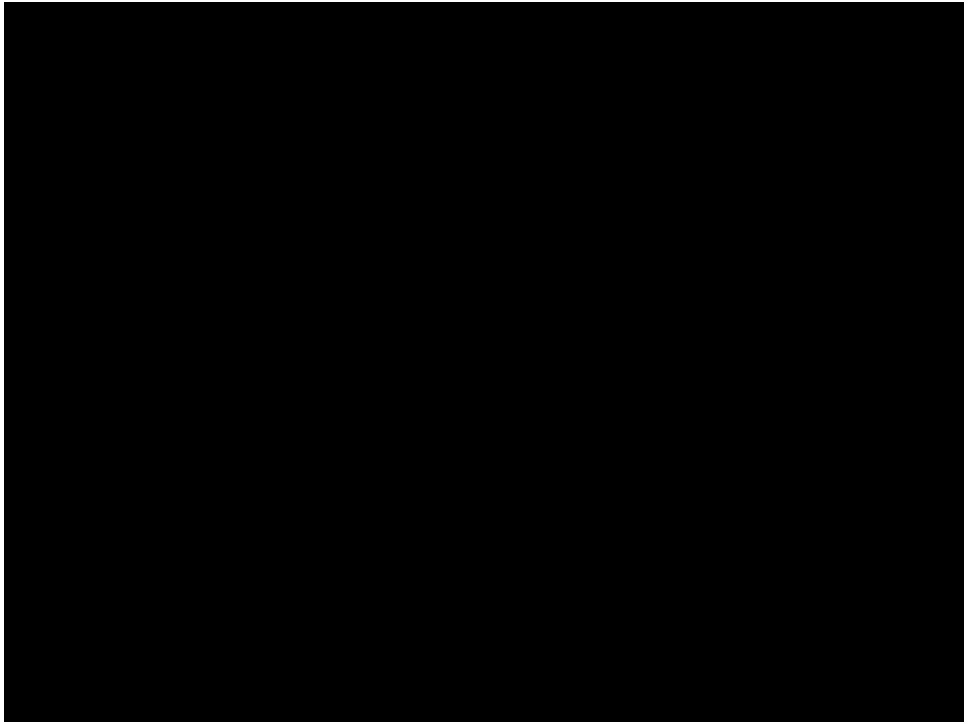
Summarization

0.0

2.6

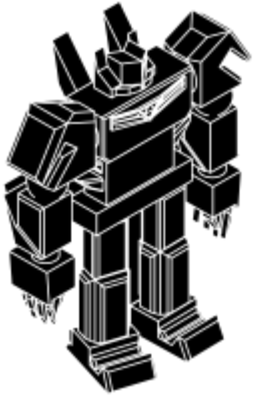
"Six people hospitalized after a storm in Attala county"

T5: Demo



Summary

- Transformers are suitable for a wide range of NLP applications
- Some transformers include GPT, BERT and T5
- T5 is a powerful multi-task transformer

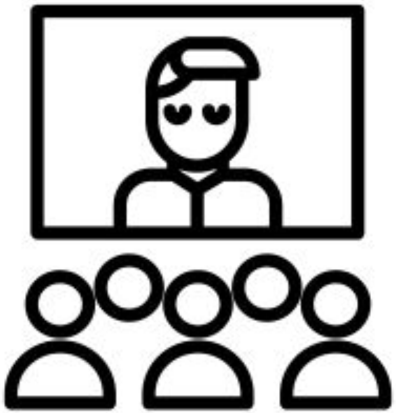


Scaled Dot-Product Attention

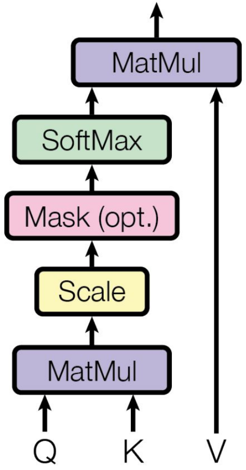
deeplearning.ai

Outline

- Revisit scaled dot product attention
- Mathematics behind Attention



Scaled dot-product attention



Queries

Keys

Values

(Vaswani et al., 2017)

Weights add up to 1

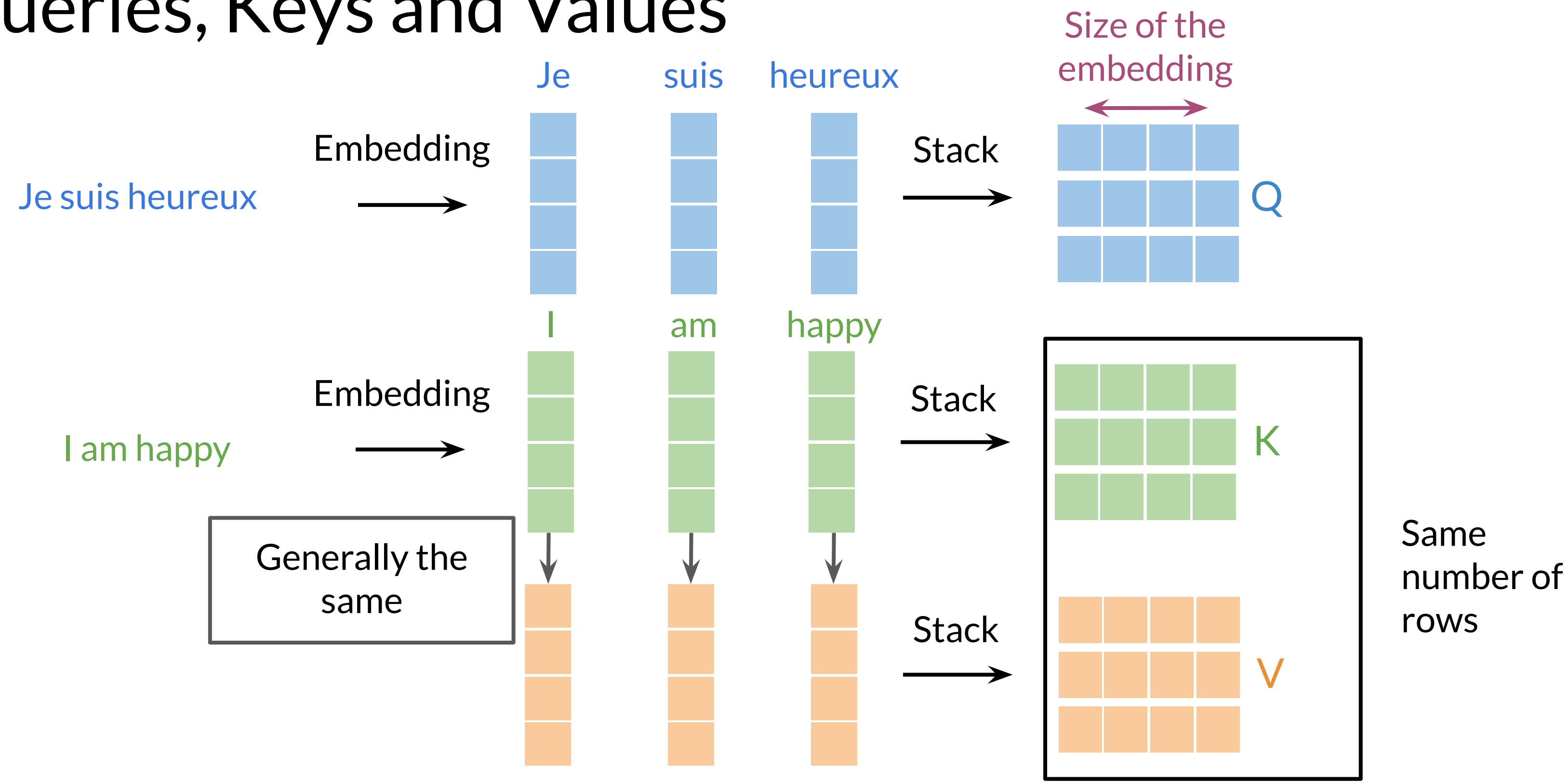
Improves performance

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

Weighted sum of values V

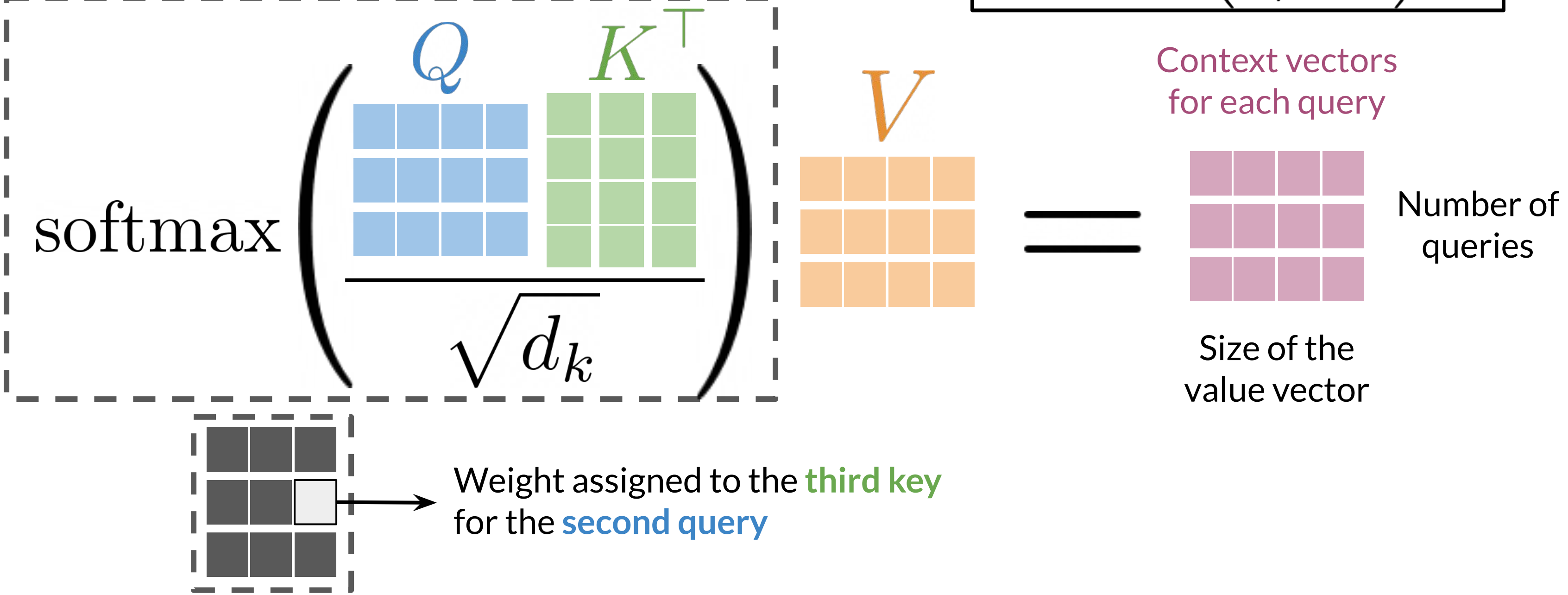
Just two matrix multiplications and a Softmax!

Queries, Keys and Values



Attention Math

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



Summary

- Scaled Dot-product Attention is essential for Transformer
- The input to Attention are queries, keys, and values
- GPUs and TPUs

deeplearning.ai

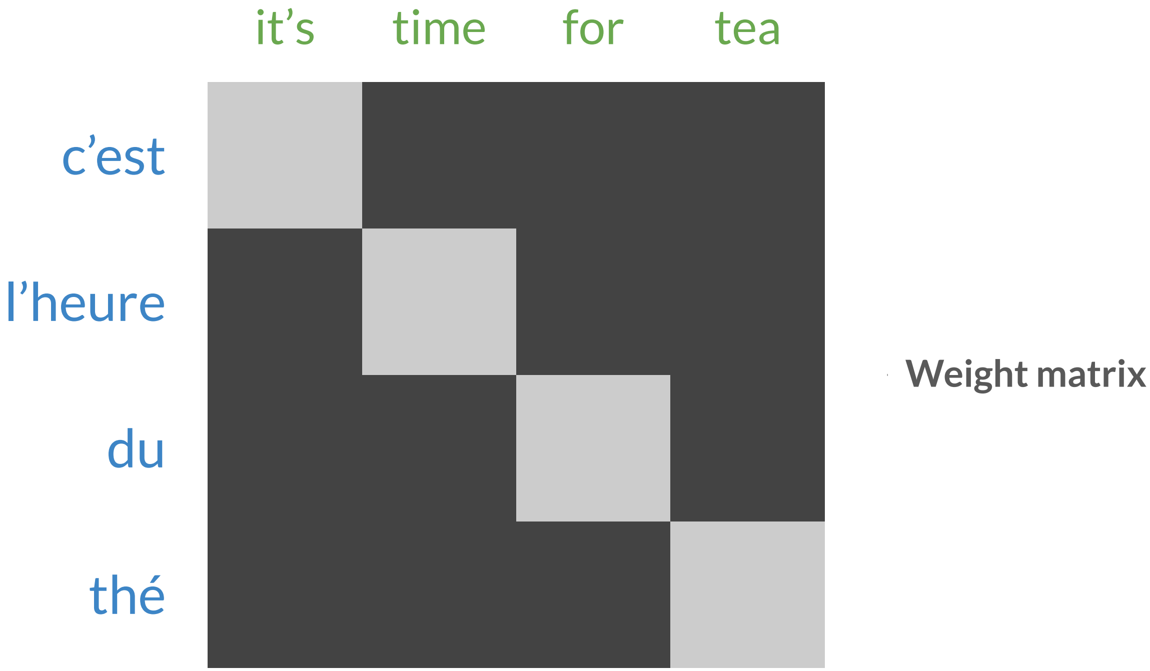
Masked Self-Attention

Outline

- Ways of Attention
- Overview of masked Self-Attention

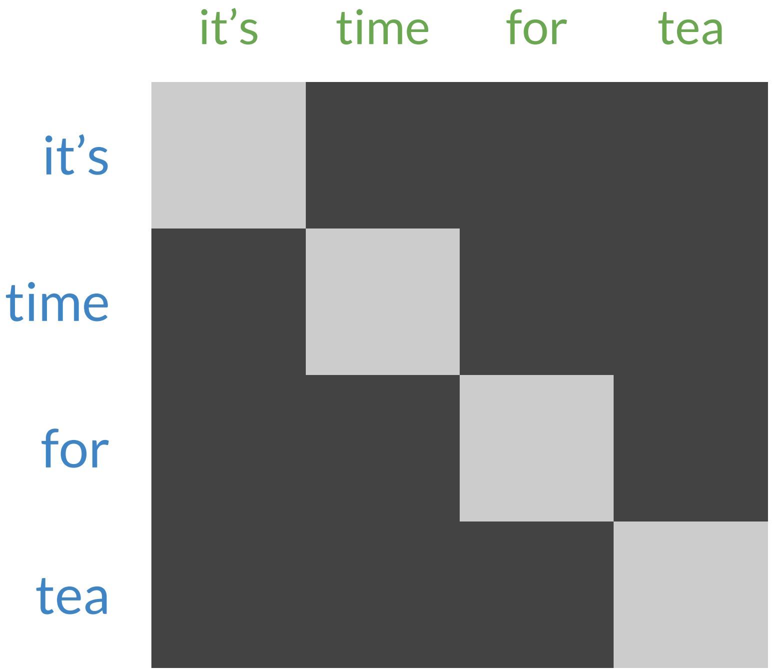
Encoder-Decoder Attention

Queries from one sentence, keys and values from another



Self-Attention

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

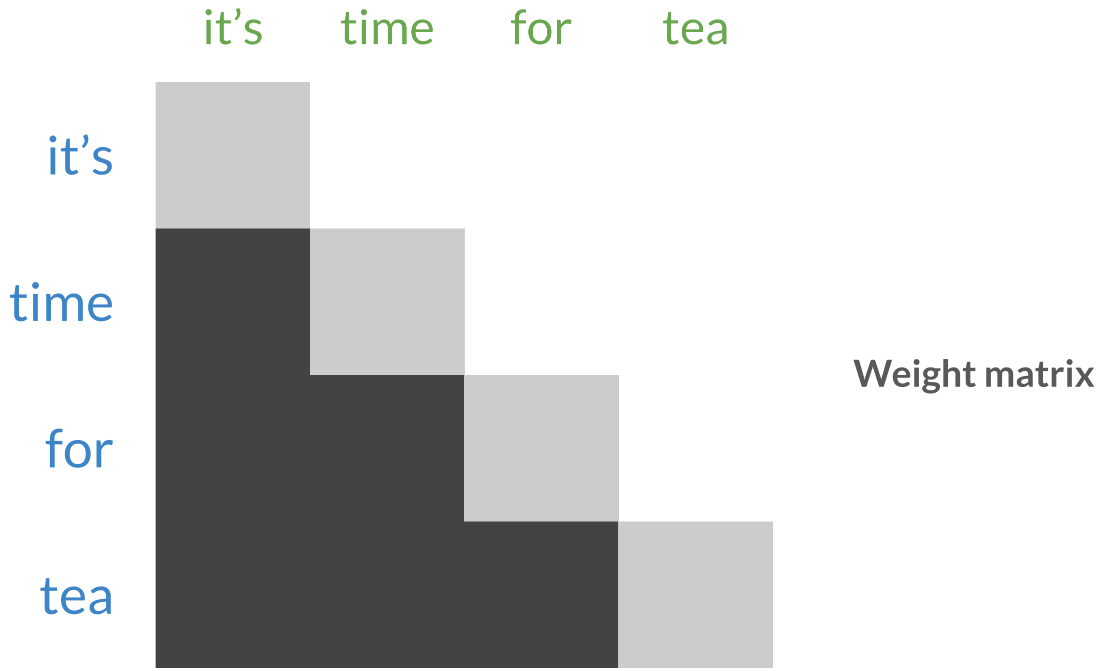


Weight matrix

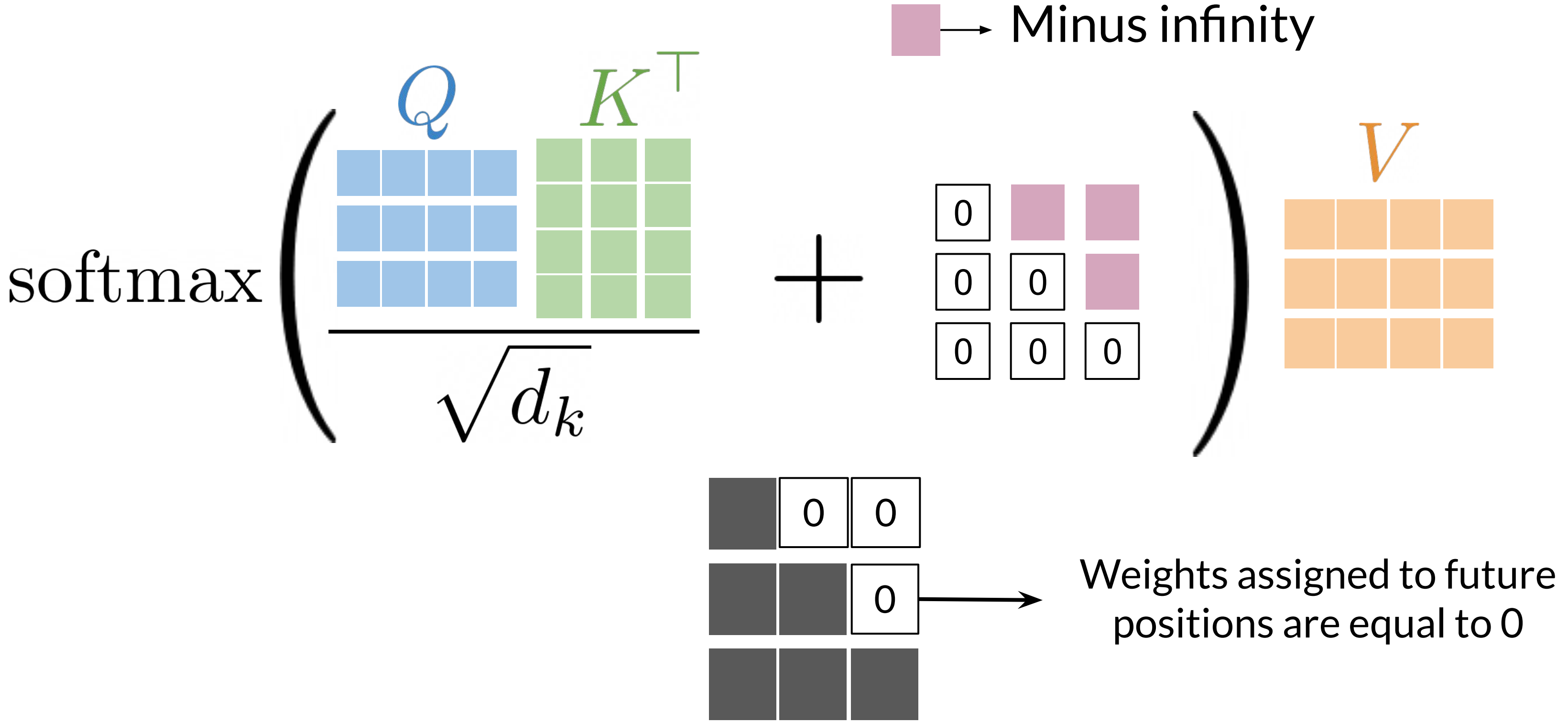
Meaning of each word **within** the sentence

Masked Self-Attention

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



Masked self-attention math



Summary

- 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

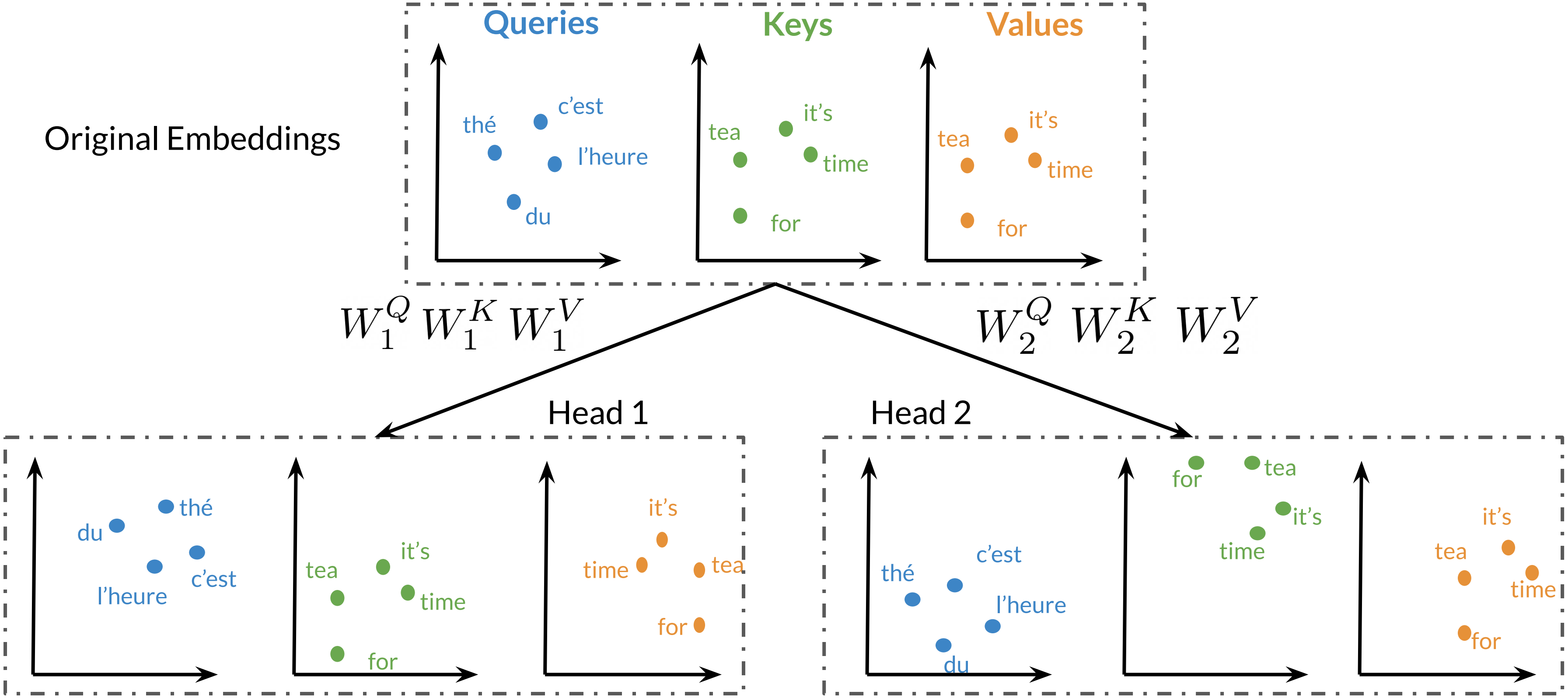
deeplearning.ai

Multi-head Attention

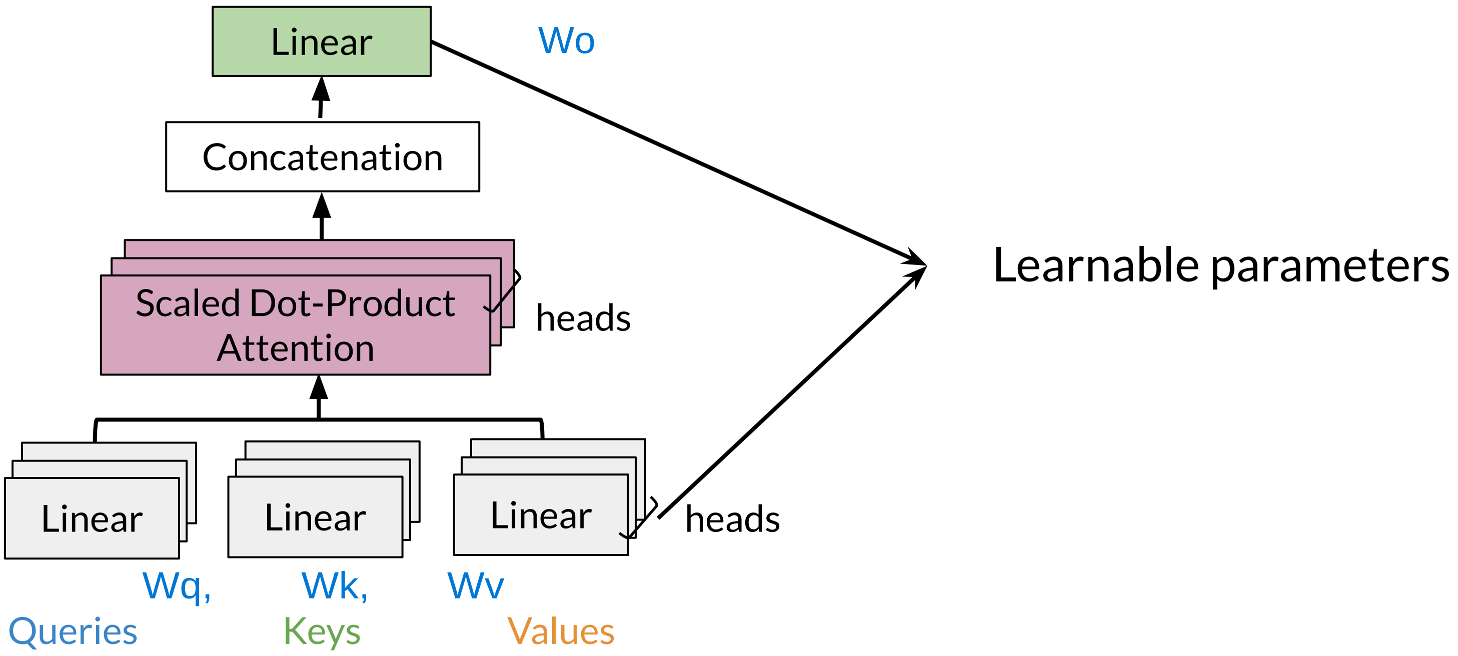
Outline

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

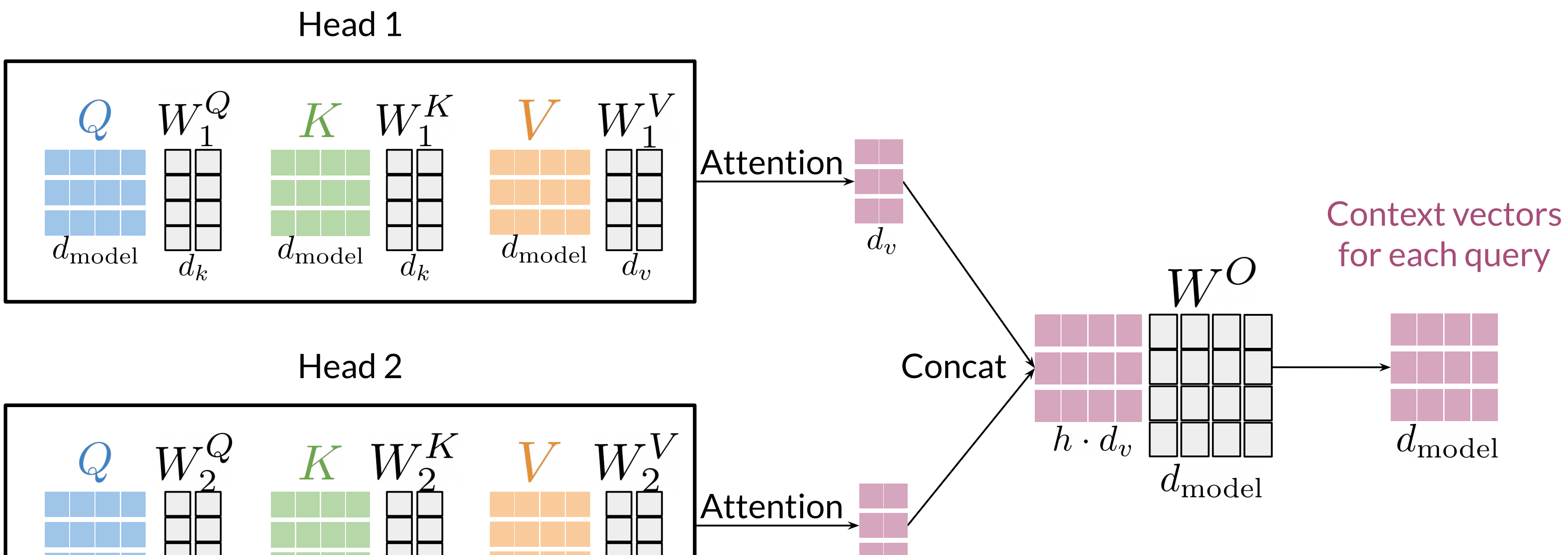
Multi-Head Attention - Overview



Multi-Head Attention - Overview



Multi-Head Attention



d_{model} : Embedding size

Usual choice of dimensions

$$d_k = d_v = d_{\text{model}}/h$$

Summary

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

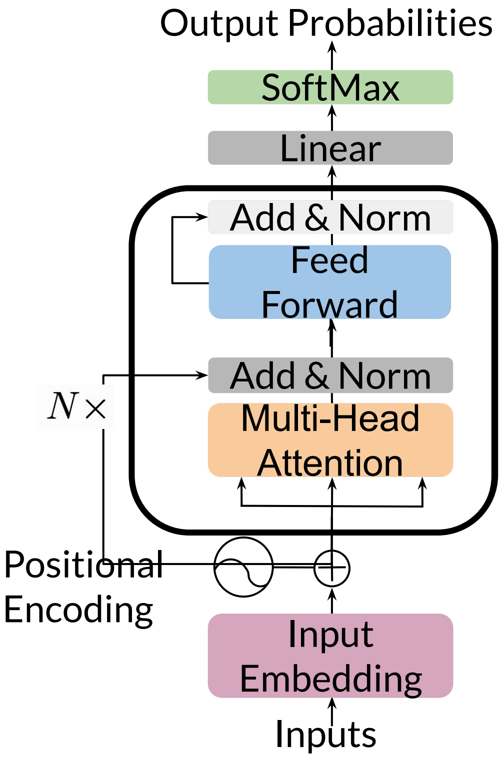
Transformer decoder

deeplearning.ai

Outline

- Overview of Transformer decoder
- Implementation (decoder and feed-forward block)

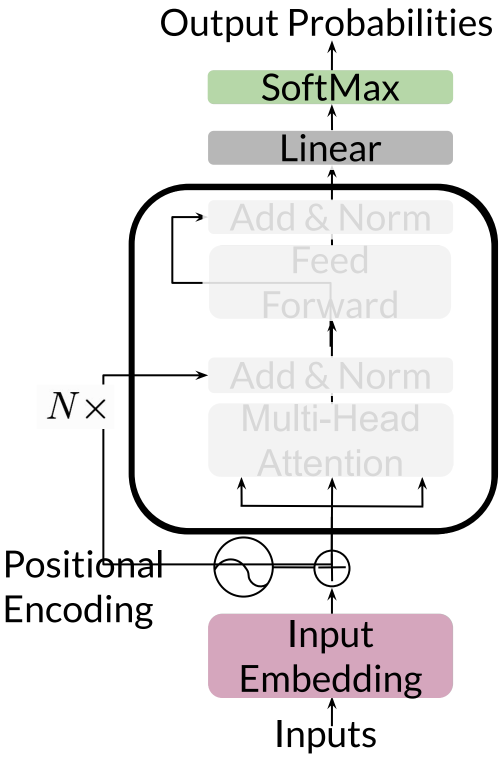
Transformer decoder



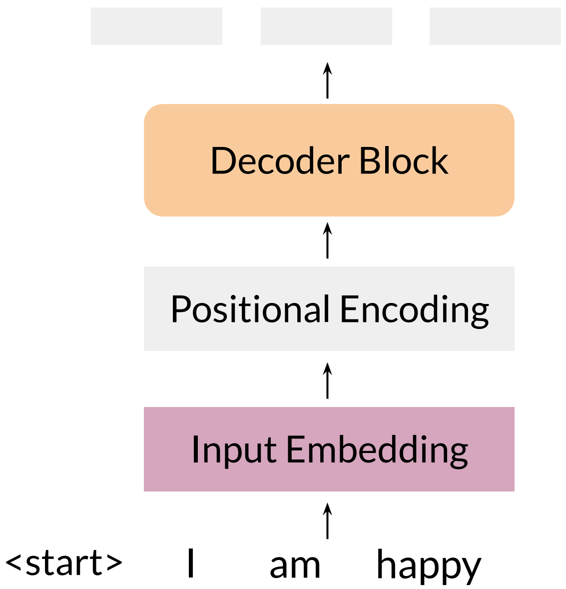
Overview

- 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

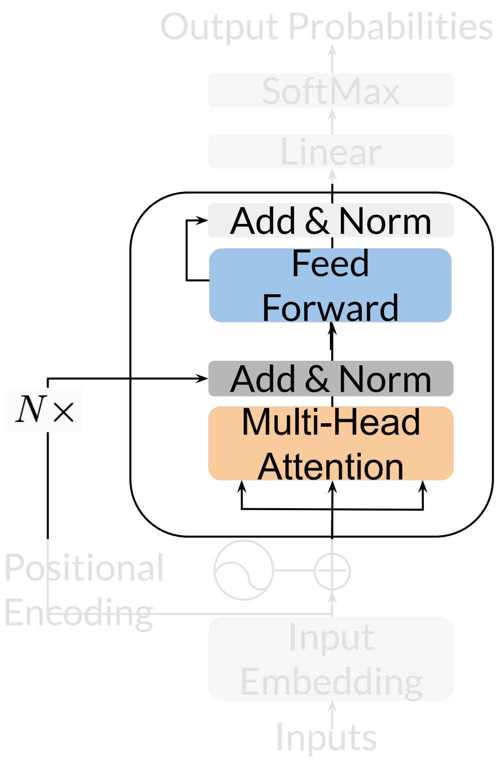
Transformer decoder

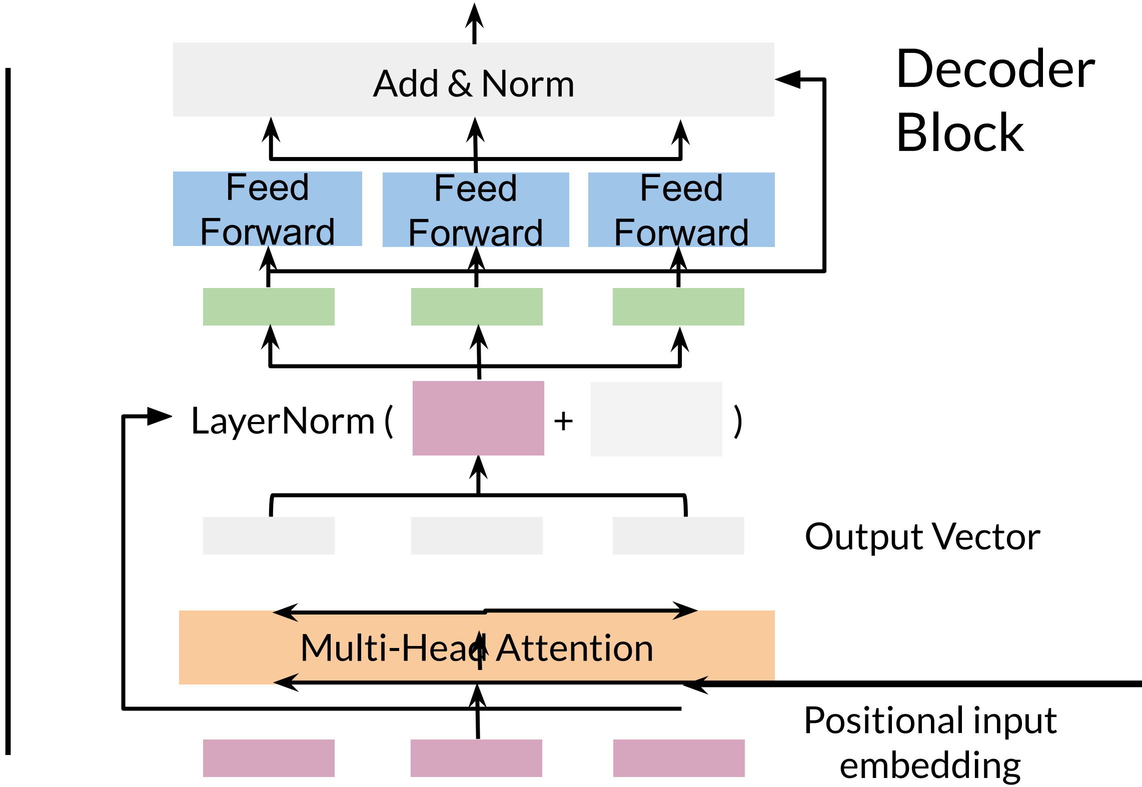


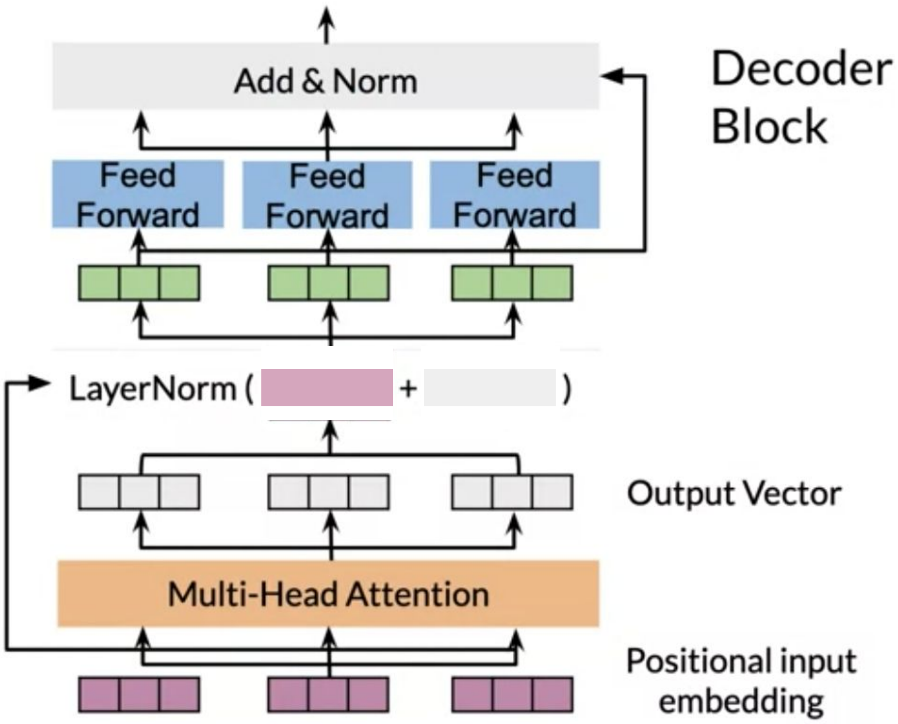
Explanation



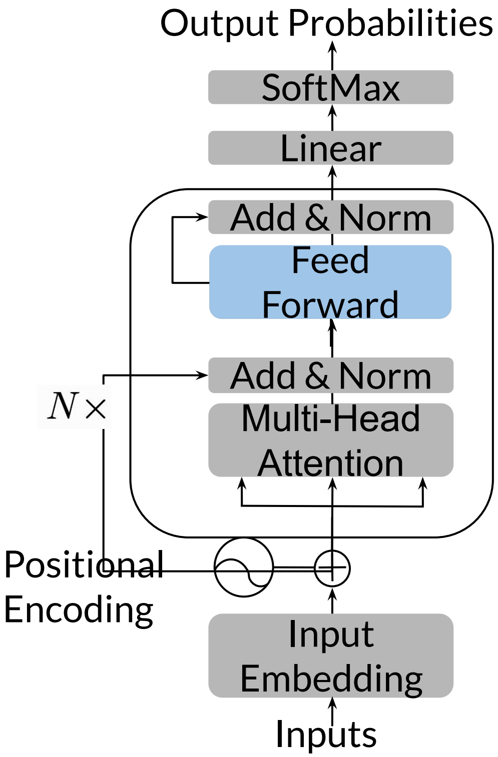
The Transformer decoder



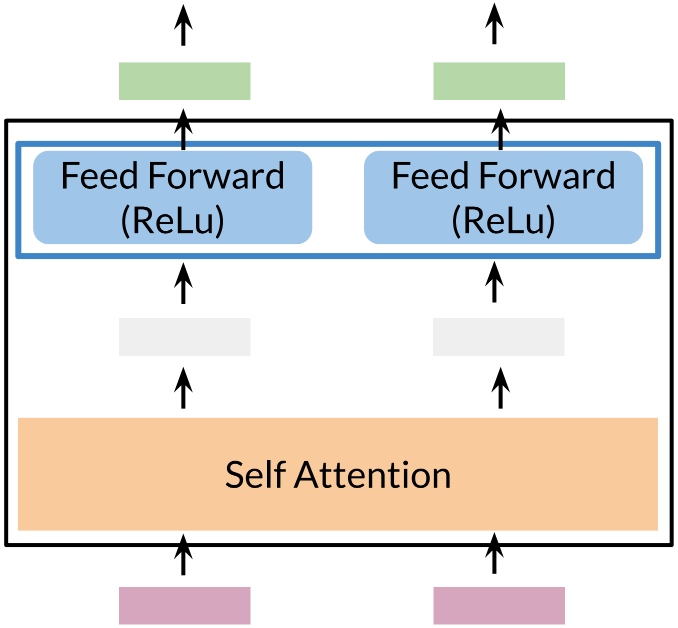



Decoder
Block
Add & Norm
Feed Forward
Feed Forward
Feed Forward
LayerNorm (+)
Output Vector
Multi-Head Attention
Positional input
embedding

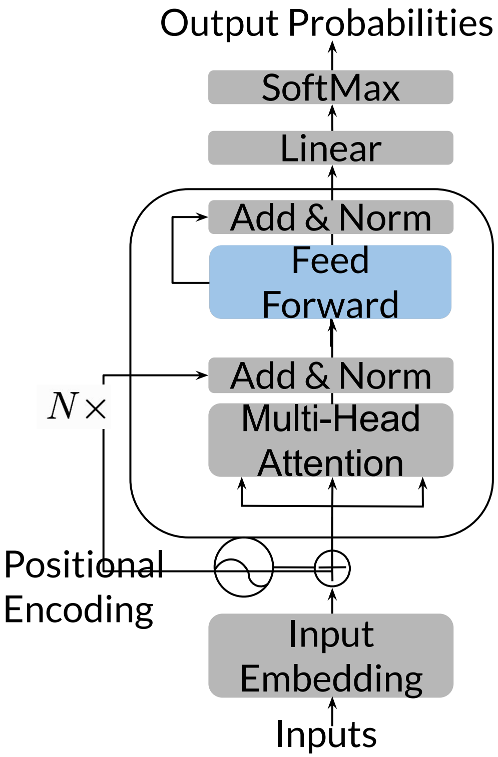
The Transformer decoder



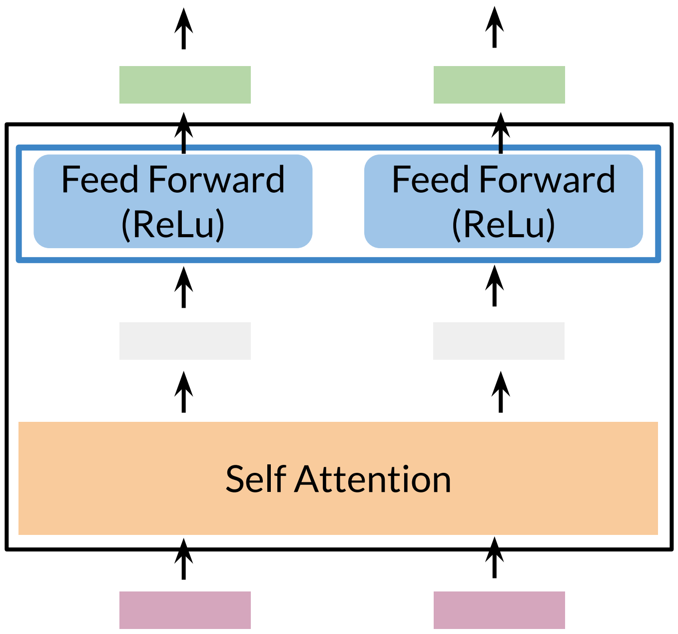
Feed forward layer



The Transformer decoder



Feed forward layer



Summary

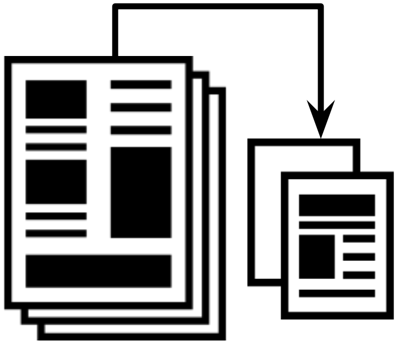
- Transformer decoder mainly consists of three layers
- Decoder and feed-forward blocks are the core of this model code
- It also includes a module to calculate the cross-entropy loss

deeplearning.ai

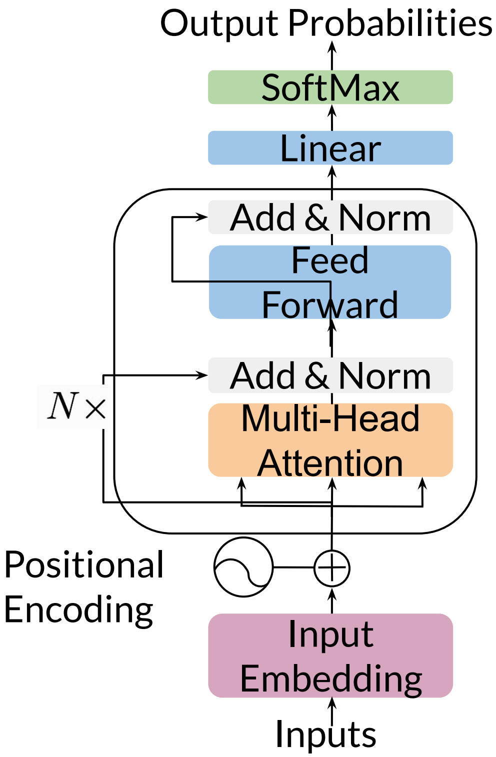
Transformer summarizer

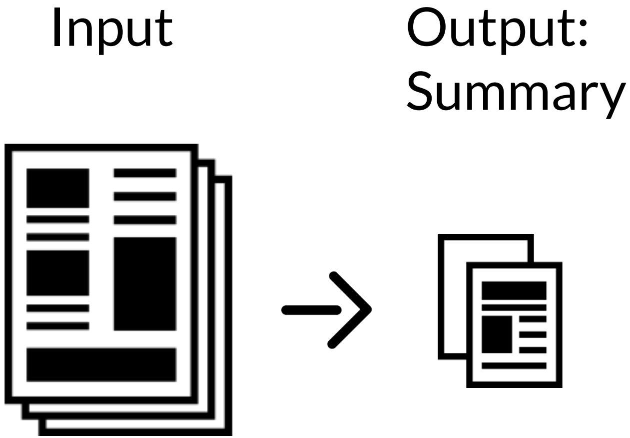
Outline

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

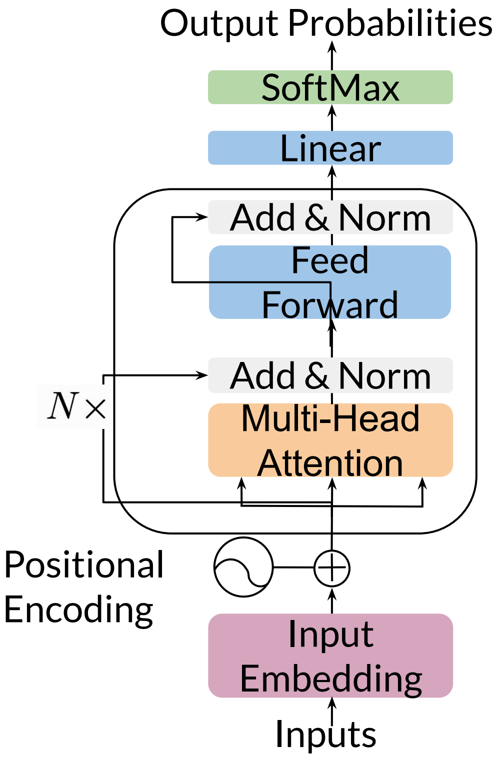


Transformer for summarization





Technical details for data processing



Model Input:

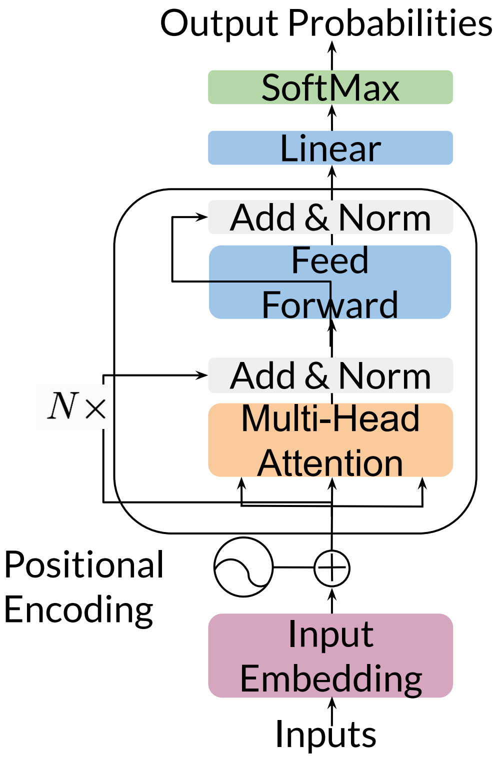
```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

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

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

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!

Summary

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