

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>



deeplearning.ai

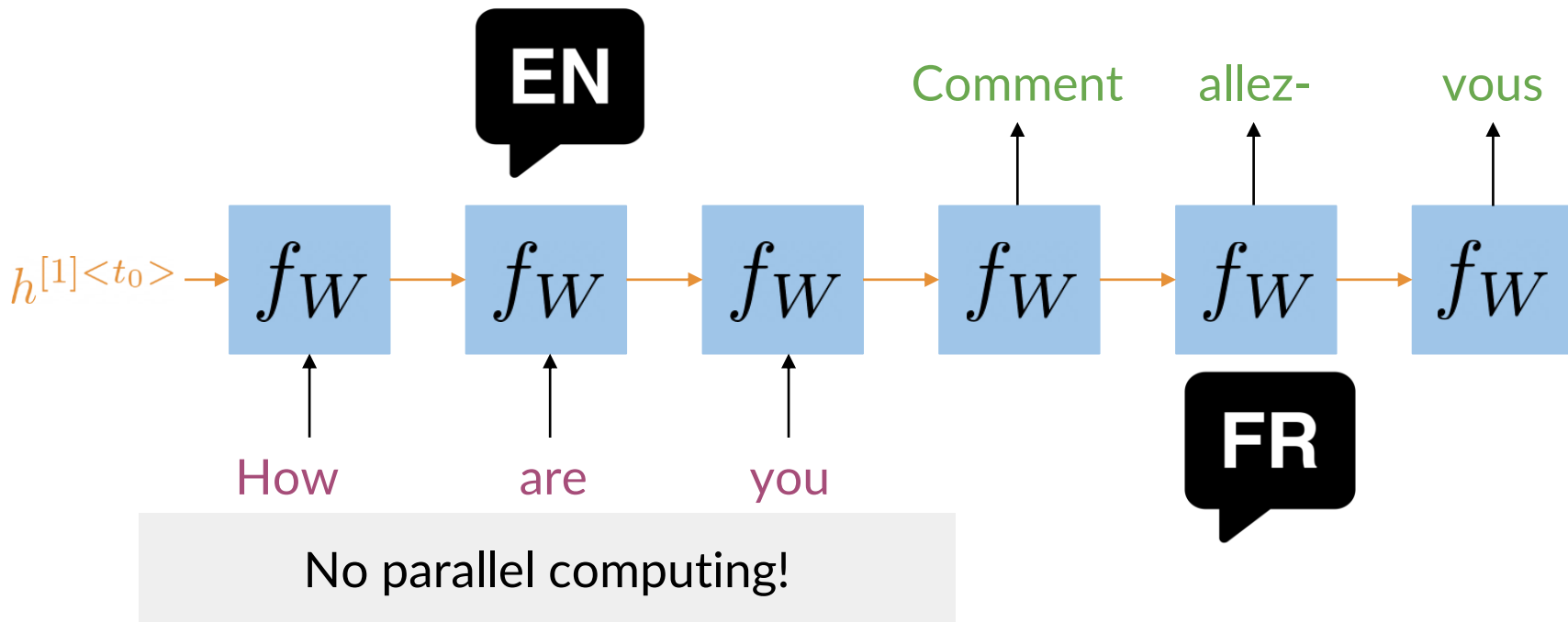
Transformers vs RNNs

Outline

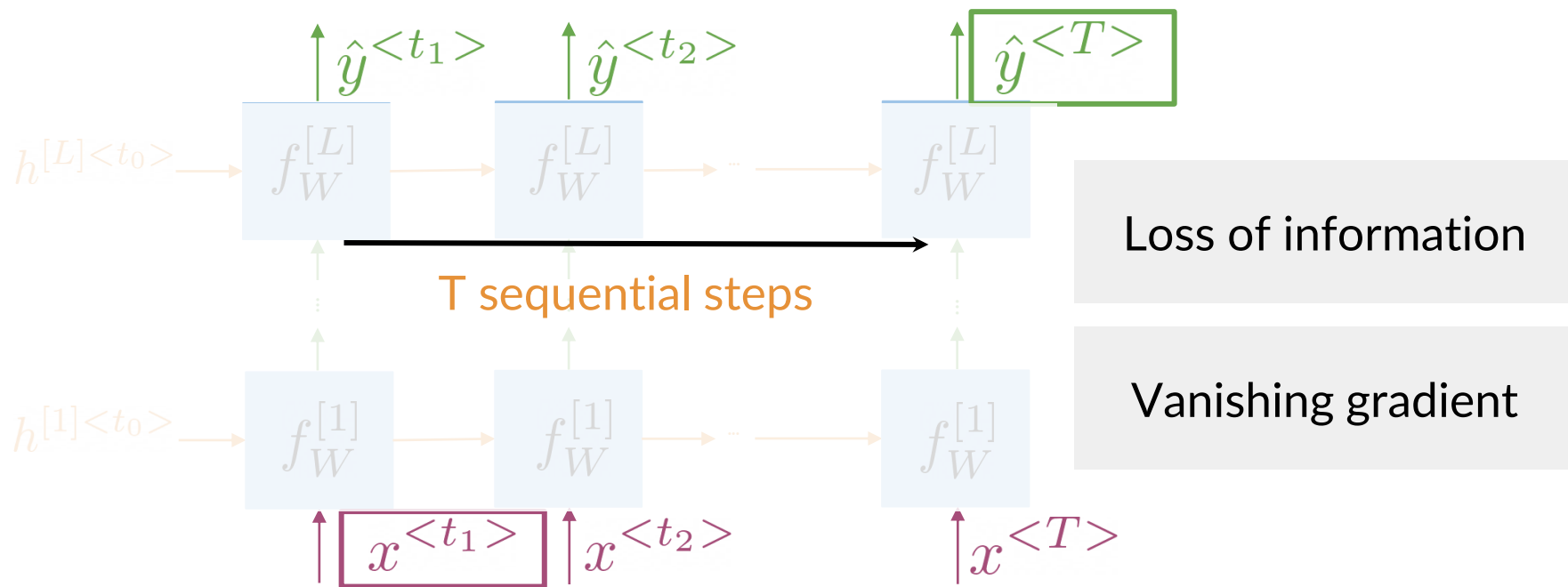
- Issues with RNNs
- Comparison with Transformers



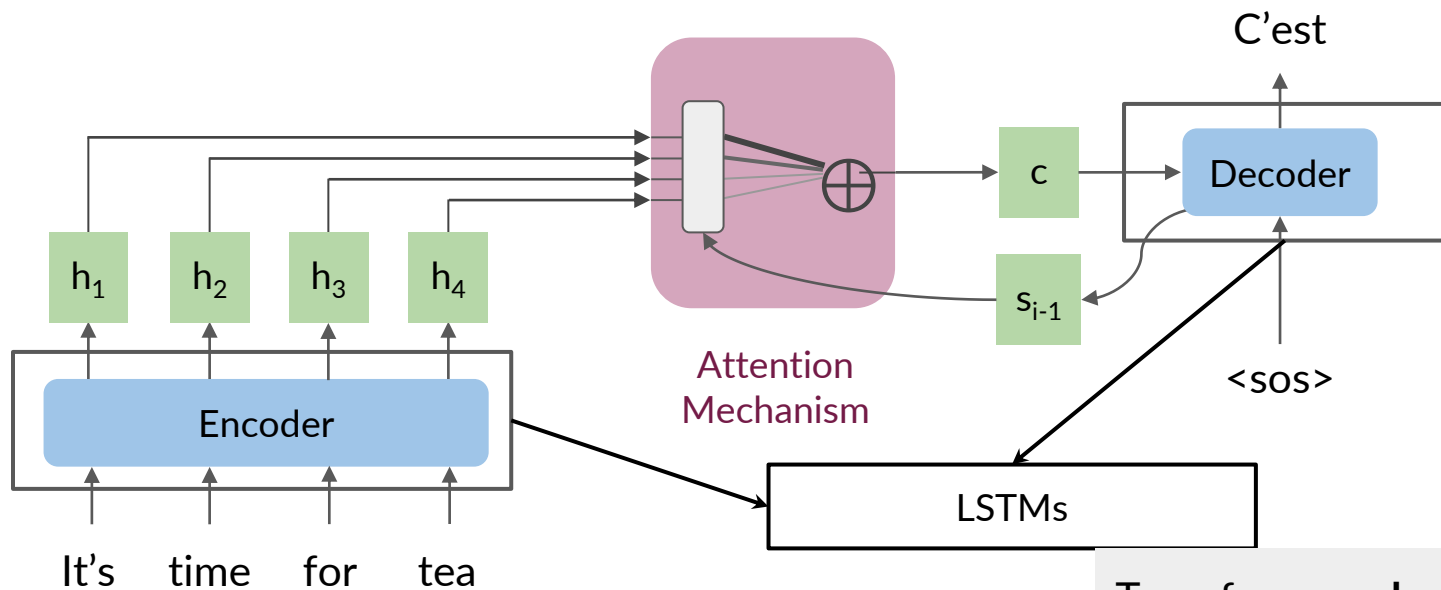
Neural Machine Translation



Seq2Seq Architectures



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

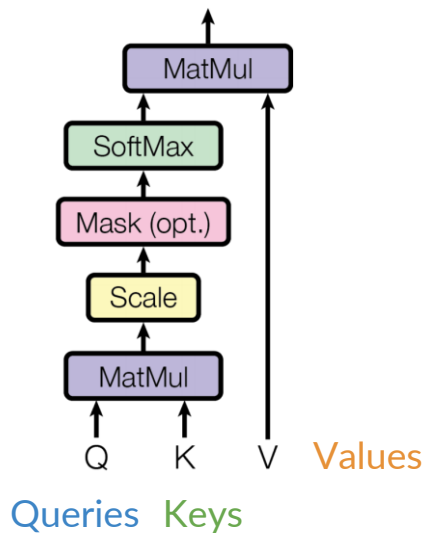
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

<https://arxiv.org/abs/1706.03762>

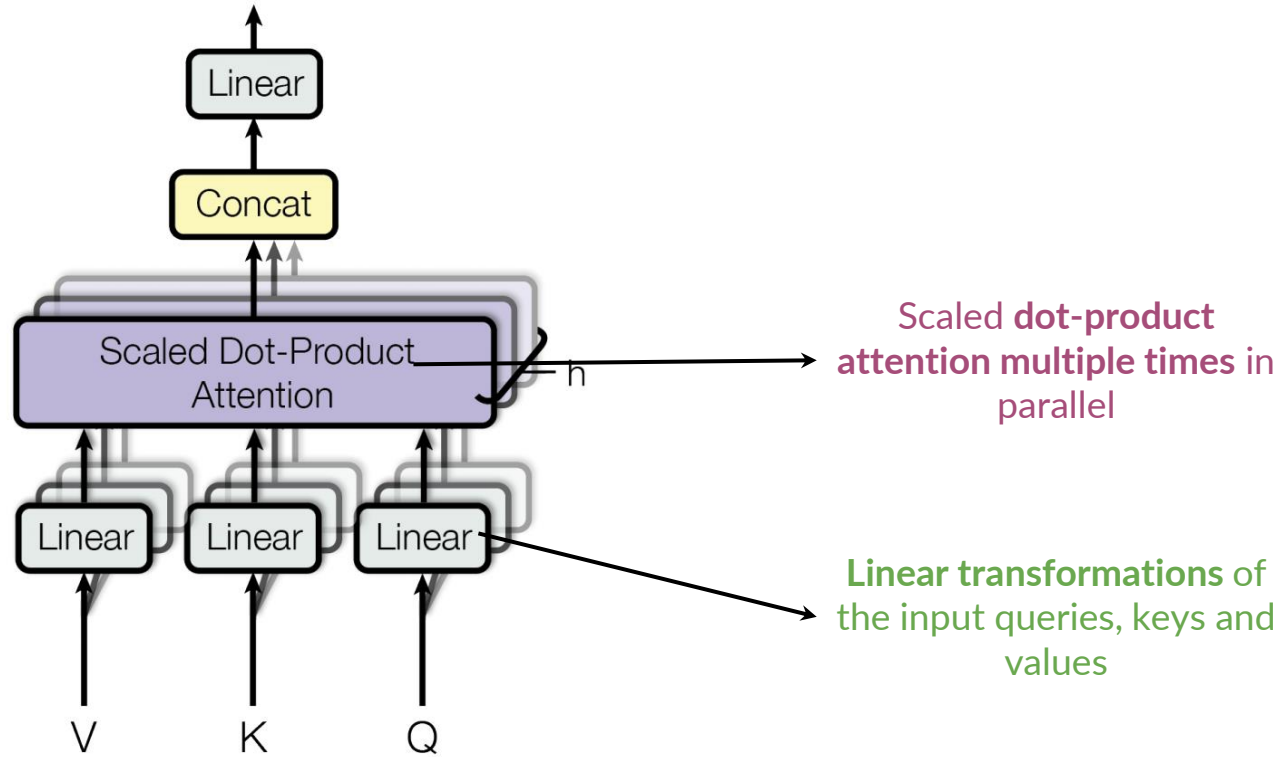
Scaled Dot-Product Attention



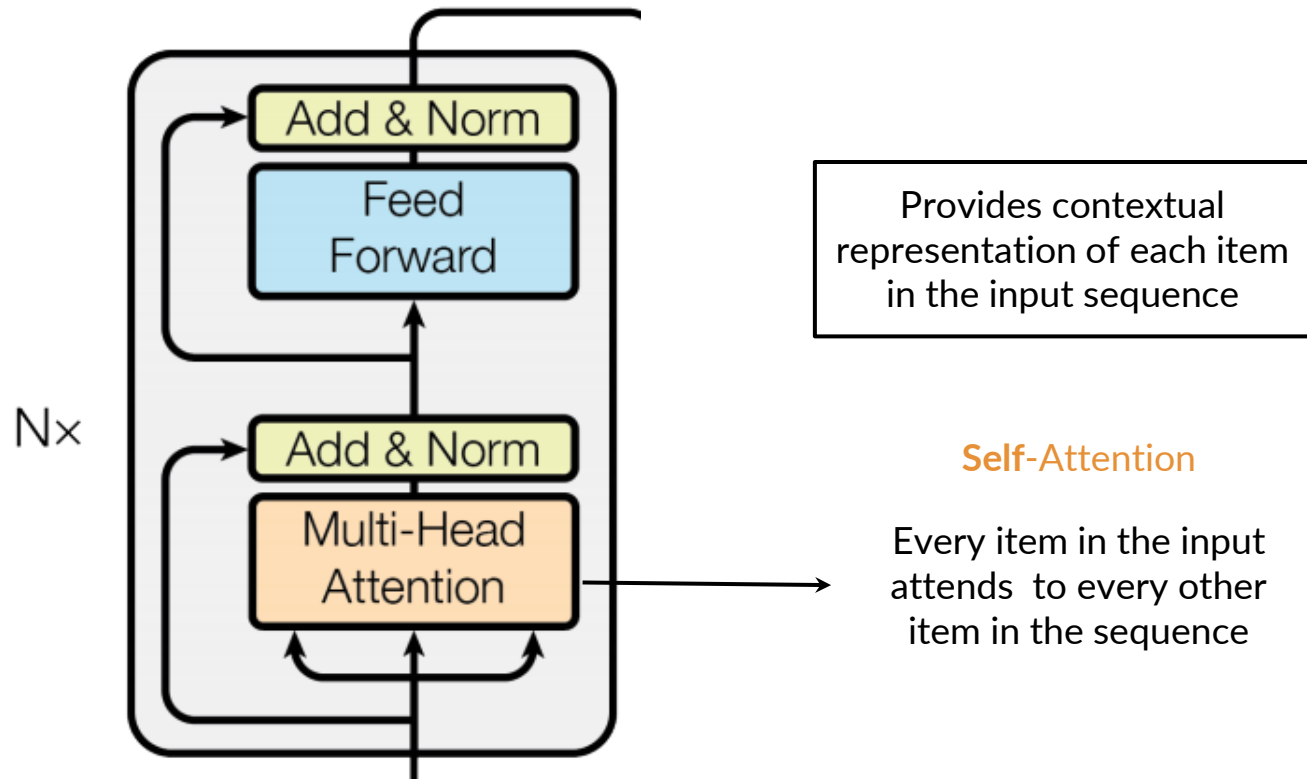
$$\text{softmax} \left(\frac{QK^{\top}}{\sqrt{d_k}} \right) V$$

(Vaswani et al., 2017)

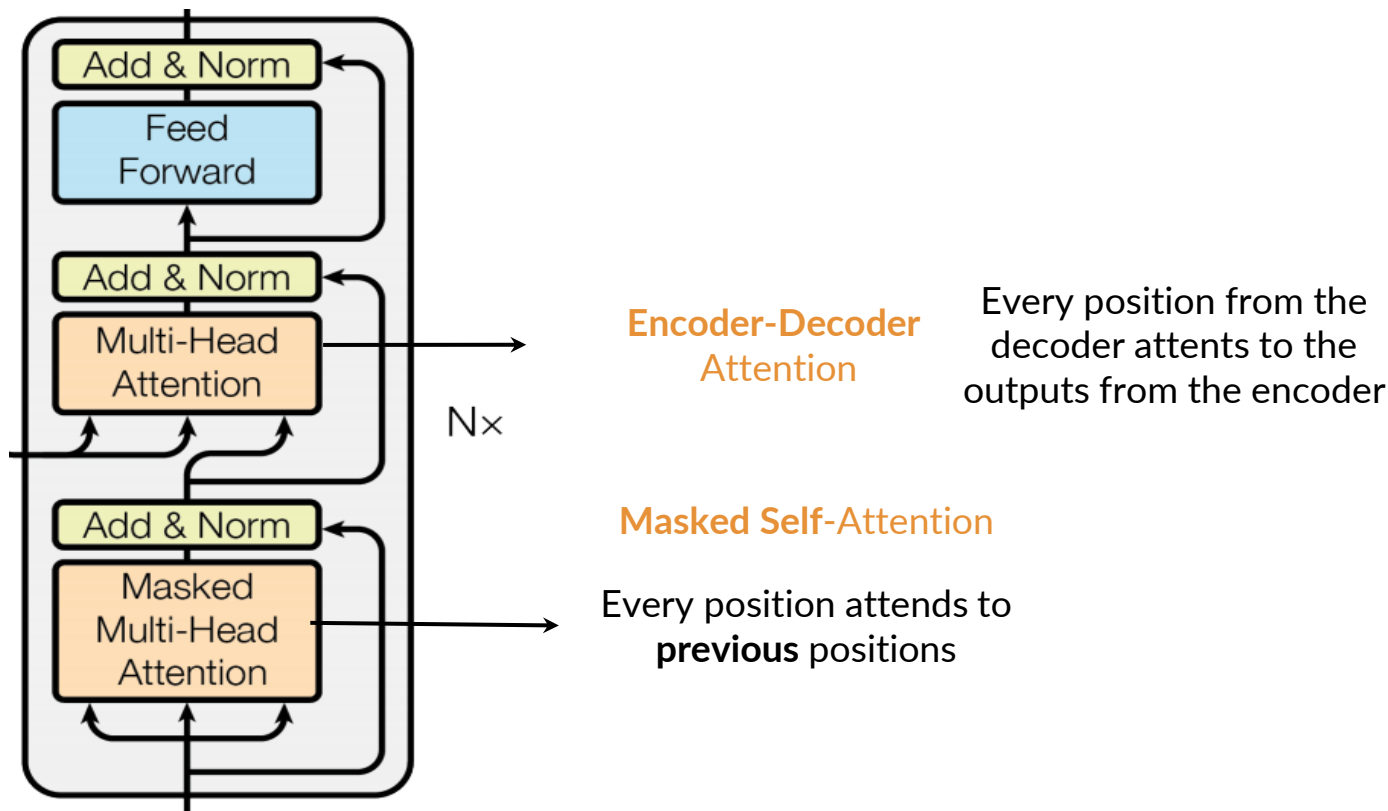
Multi-Head Attention



The Encoder



The Decoder



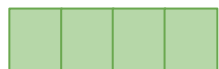
RNNs vs Transformer: Positional Encoding

POSITIONAL
ENCODING



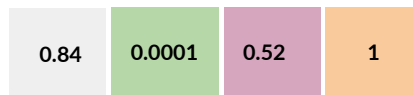
+

EMBEDDINGS

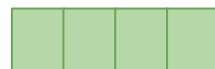


INPUT

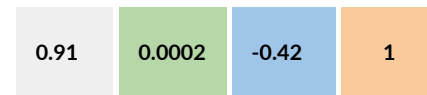
Je



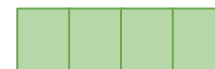
+



suis

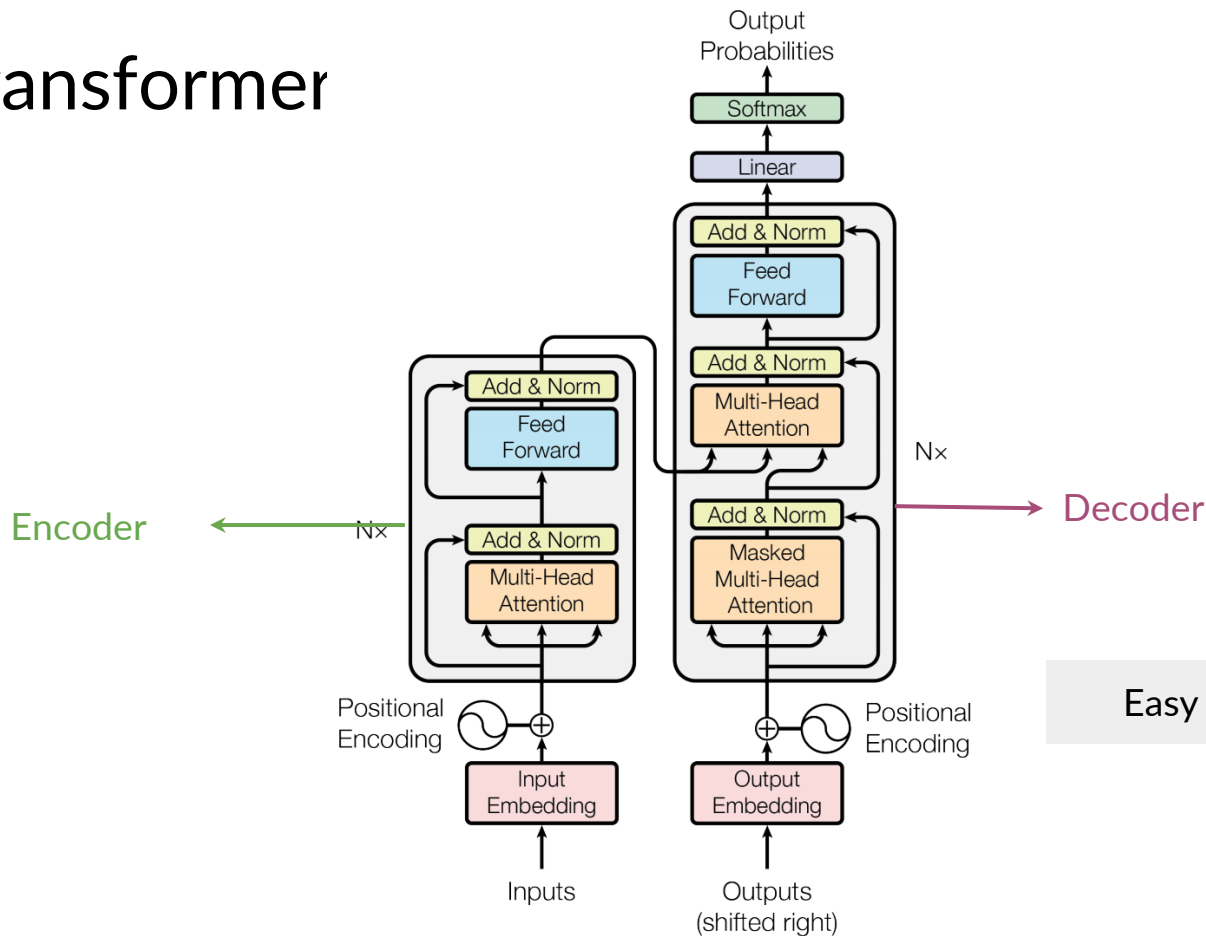


+



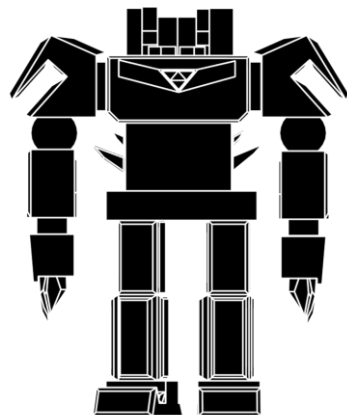
content

The Transformer



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



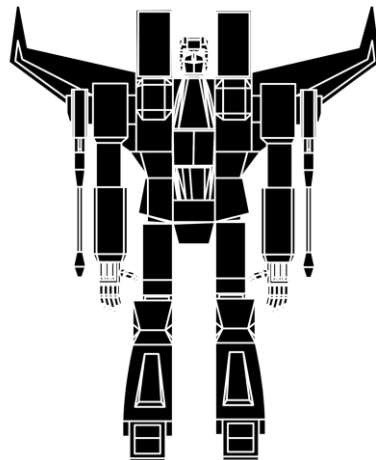


deeplearning.ai

Transformer Applications

Outline

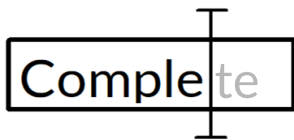
- Transformers applications in NLP
- Some Transformers
- Introduction to T5



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

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

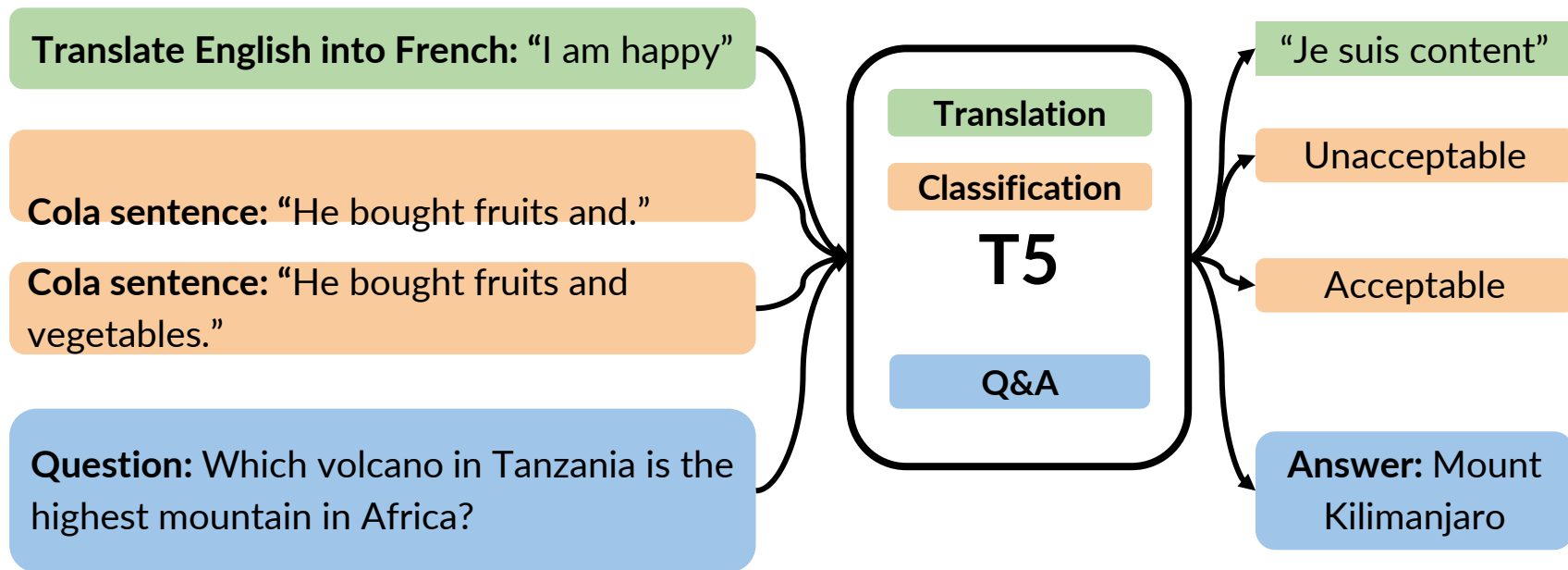
Colin, R., et al. (2019)
Google

GPT-2: Generative Pre-training for
Transformer

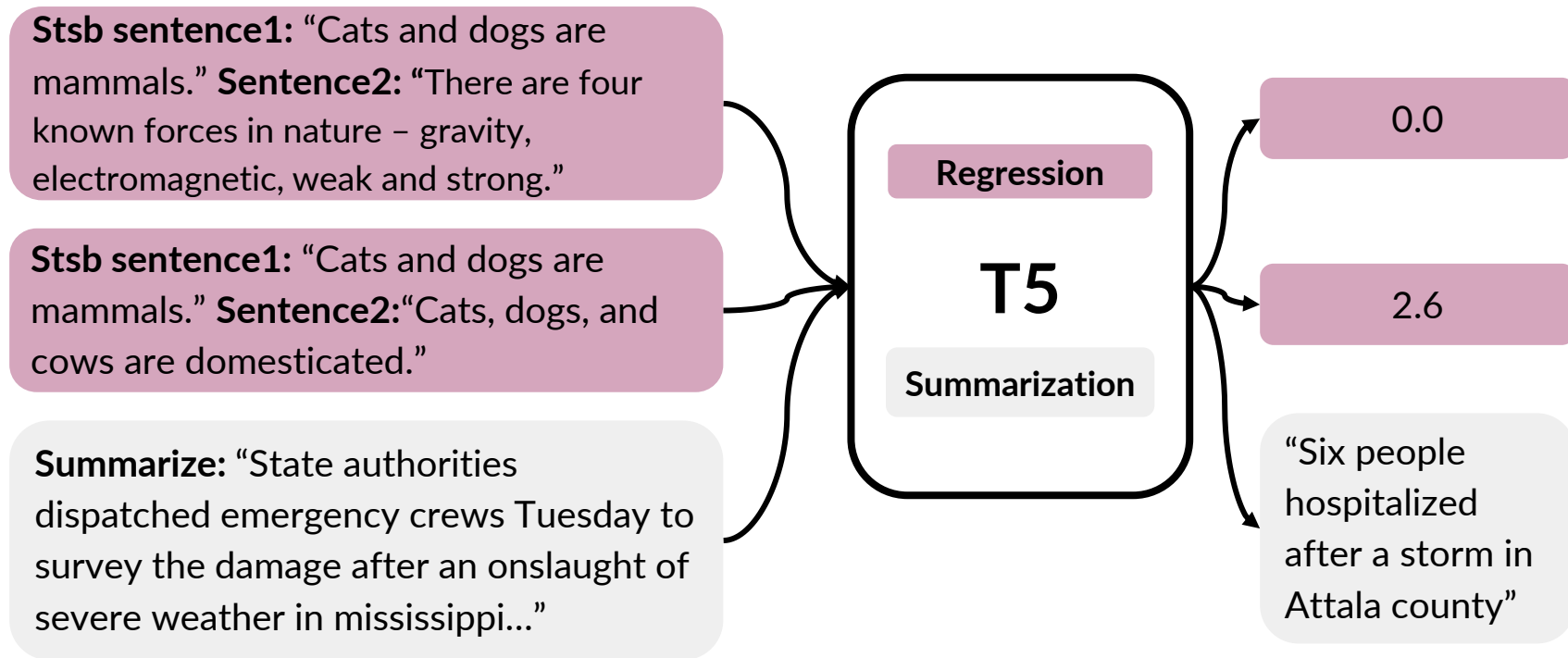
BERT: Bidirectional Encoder
Representations from Transformers

T5: Text-to-text transfer transformer

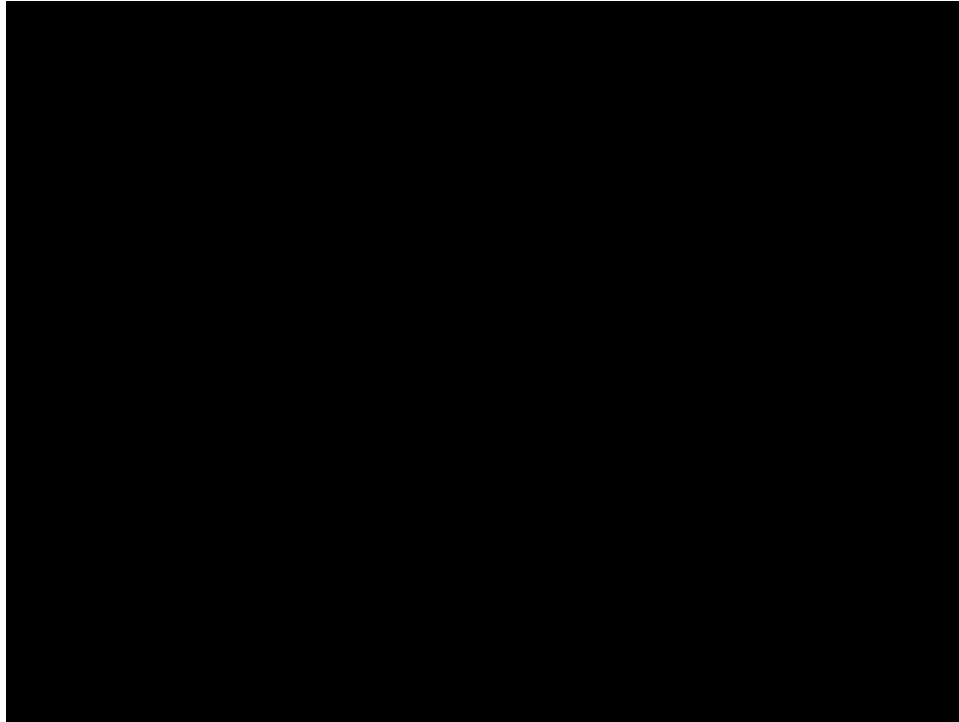
T5: Text-To-Text Transfer Transformer



T5: Text-To-Text Transfer Transformer

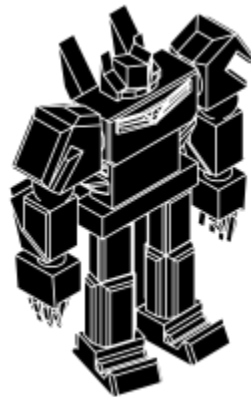


T5: Demo



Summary

- 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



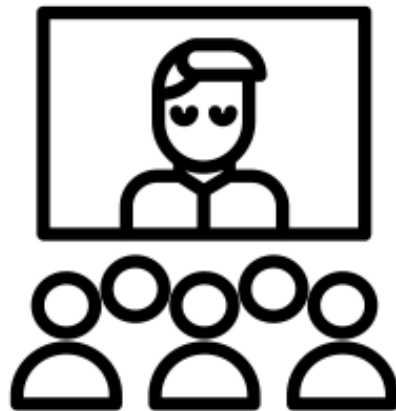


deeplearning.ai

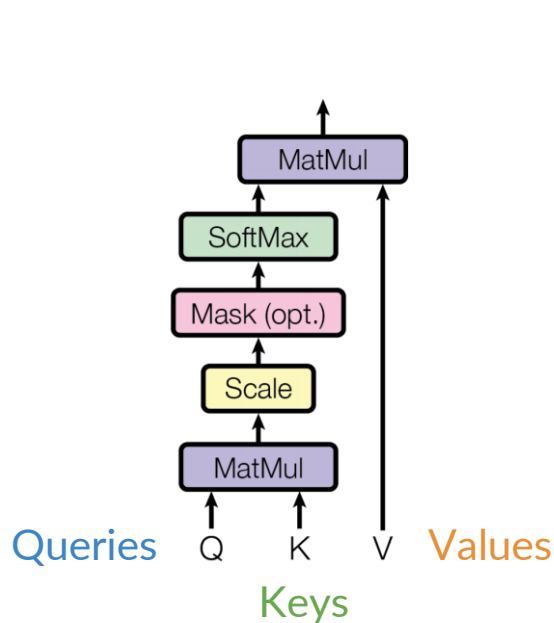
Scaled Dot-Product Attention

Outline

- Revisit scaled dot product attention
- Mathematics behind Attention



Scaled dot-product attention



(Vaswani et al., 2017)

Weights add up to 1

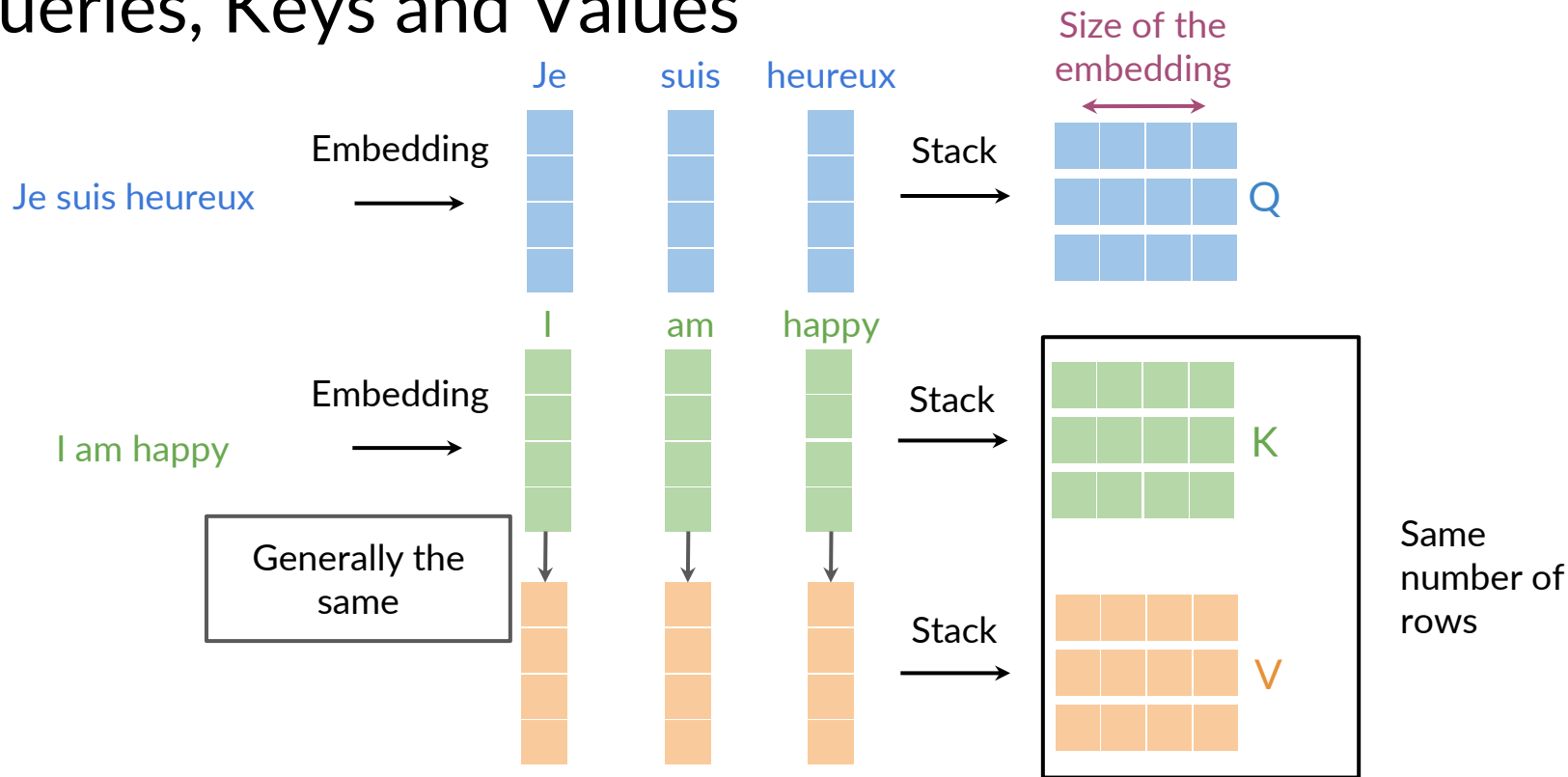
Improves performance

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

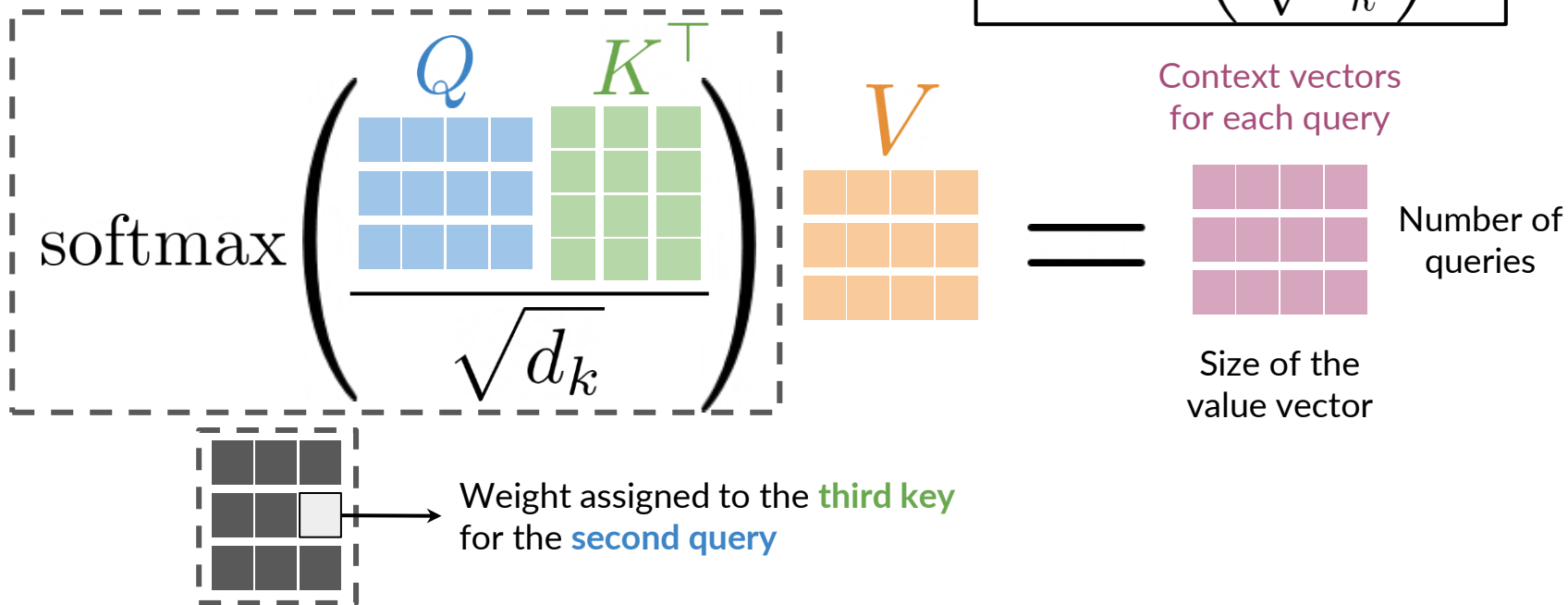
Weighted sum of values V

Just two matrix multiplications
and a Softmax!

Queries, Keys and Values



Attention Math



Summary

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



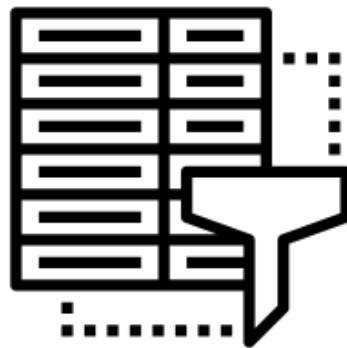


deeplearning.ai

Masked Self- Attention

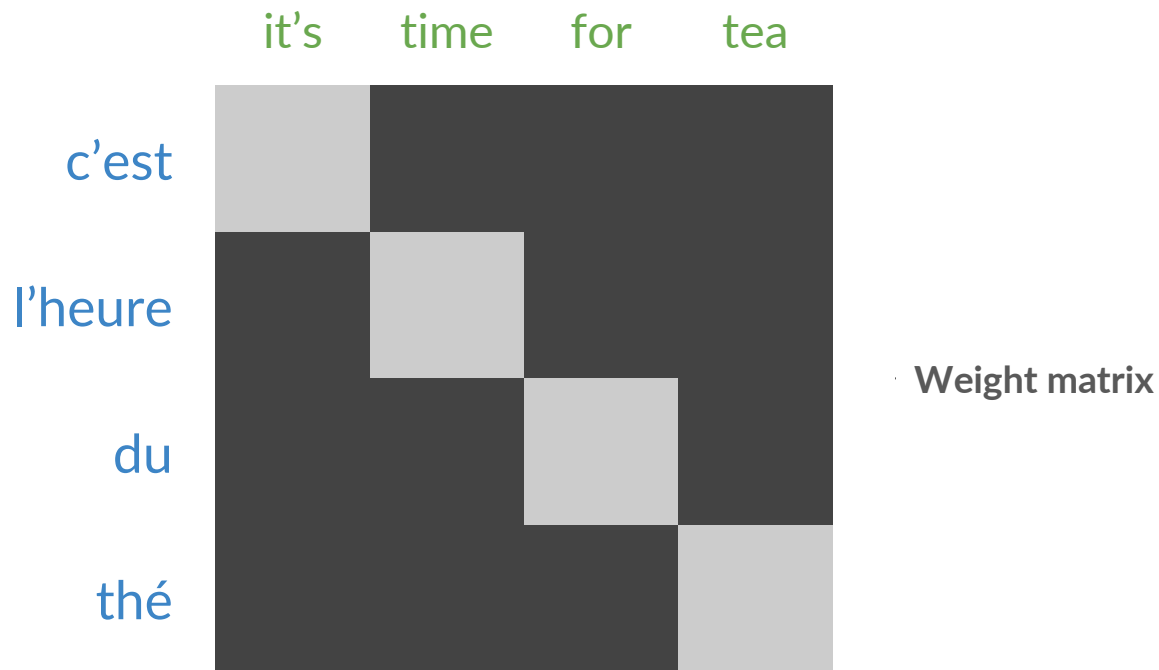
Outline

- Ways of Attention
- Overview of masked Self-Attention



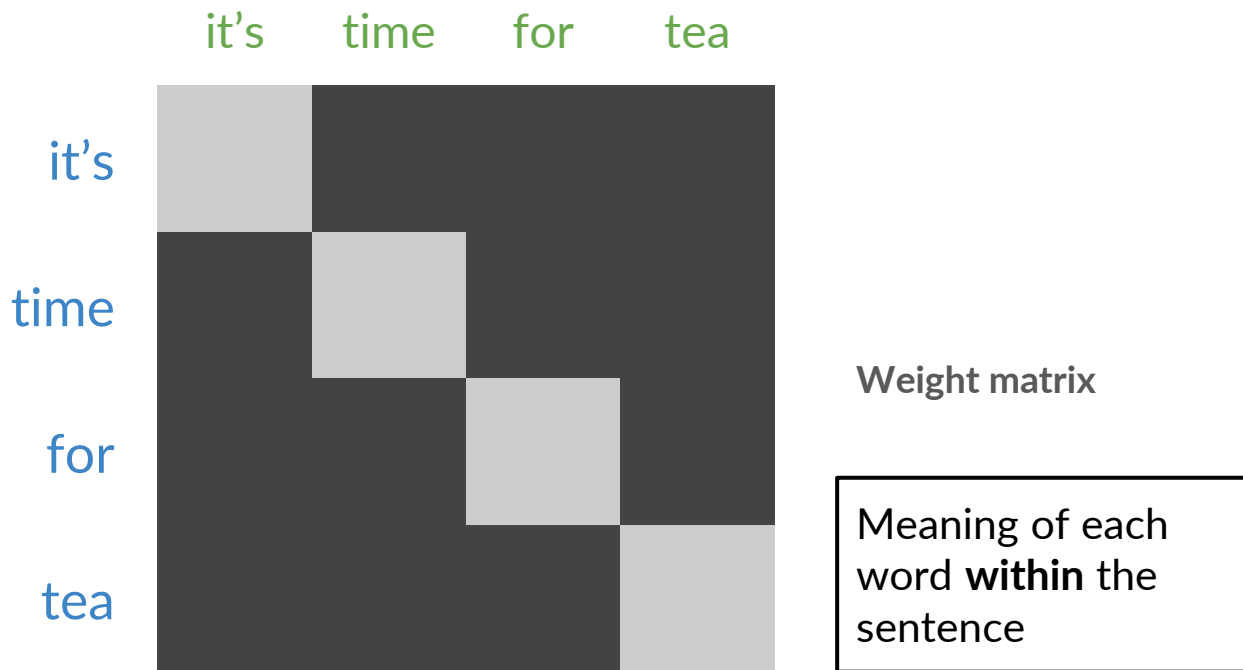
Encoder-Decoder Attention

Queries from one sentence, **keys** and **values** from another



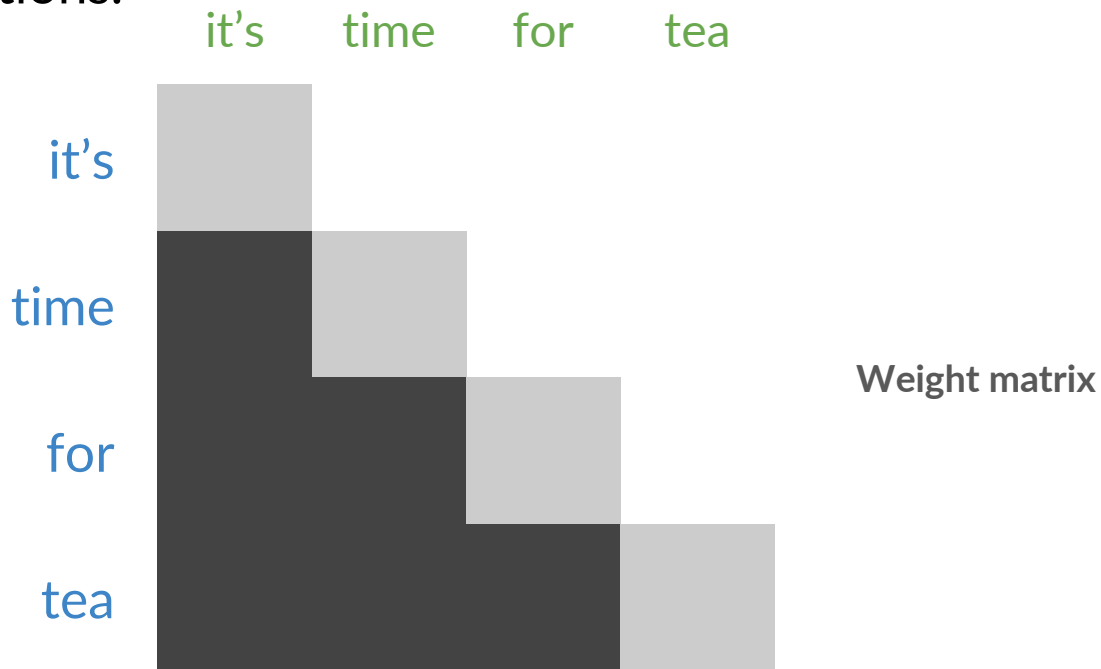
Self-Attention

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



Masked Self-Attention

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



Masked self-attention math

Diagram illustrating the masked self-attention mechanism:

The formula for masked self-attention is:

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

Legend:

- Minus infinity (represented by a pink square)
- Weights assigned to future positions are equal to 0 (represented by a dark gray square)

Matrix dimensions and values:

- Q (Query): 3x4 matrix (blue squares)
- K^T (Key Transpose): 4x3 matrix (green squares)
- Mask: 3x3 matrix (white squares with 0 or pink squares for minus infinity)
- V (Value): 3x4 matrix (orange squares)

Mask matrix (3x3):

0	Minus infinity	Minus infinity
0	0	Minus infinity
0	0	0

Weights assigned to future positions are equal to 0 (3x3):

Dark gray	0	0
Dark gray	Dark gray	0
Dark gray	Dark gray	Dark gray

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

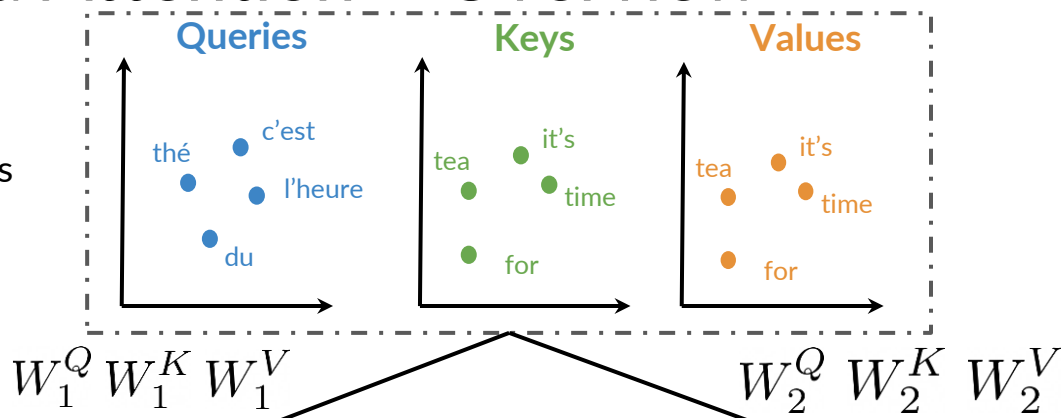
Outline

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

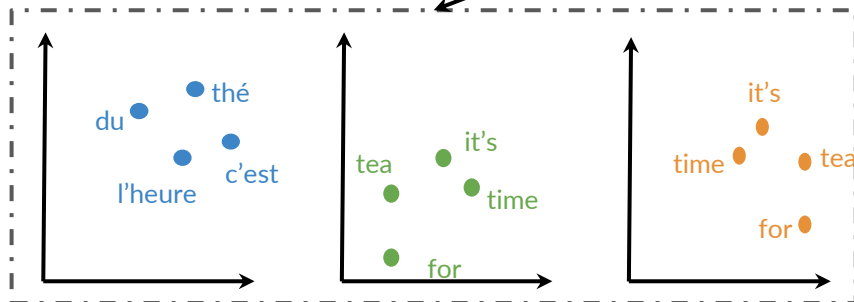


Multi-Head Attention - Overview

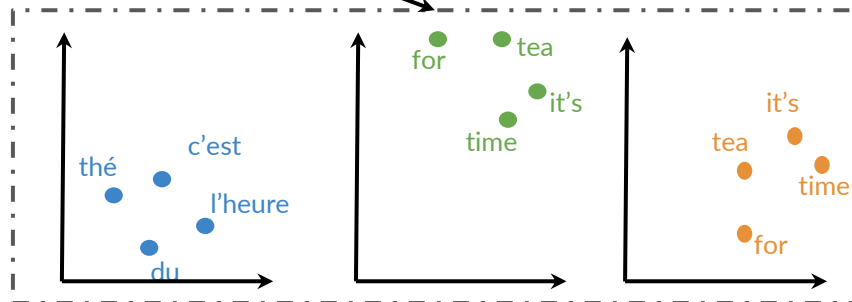
Original Embeddings



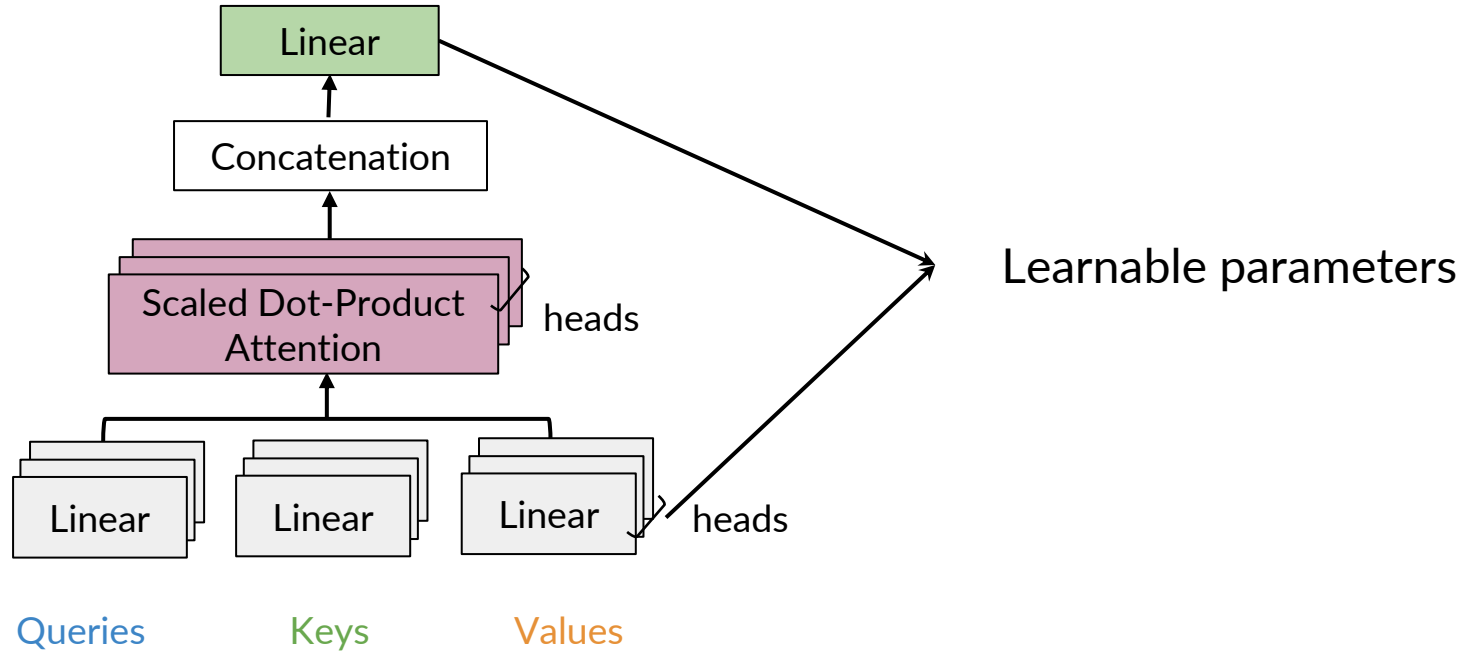
Head 1



Head 2

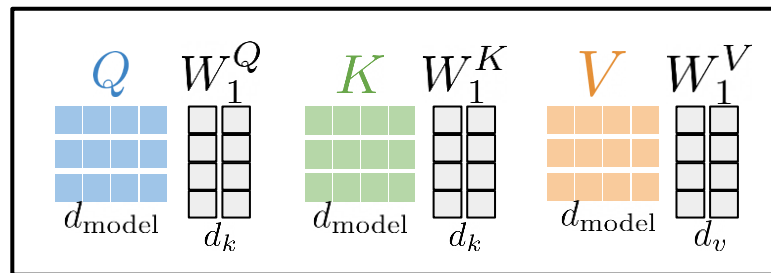


Multi-Head Attention - Overview

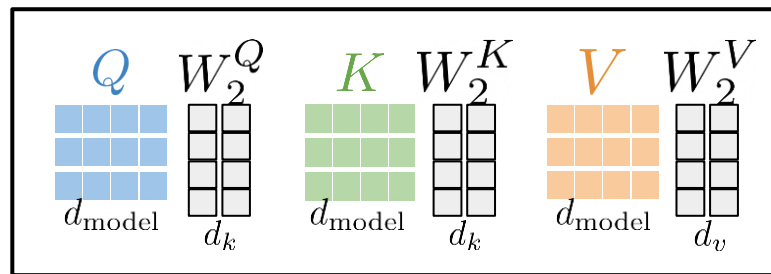


Multi-Head Attention

Head 1



Head 2

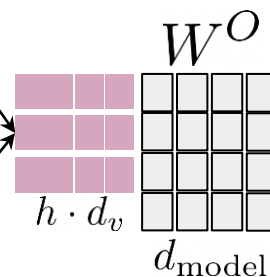


d_{model} : Embedding size

Attention



Concat



Context vectors
for each query

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





deeplearning.ai

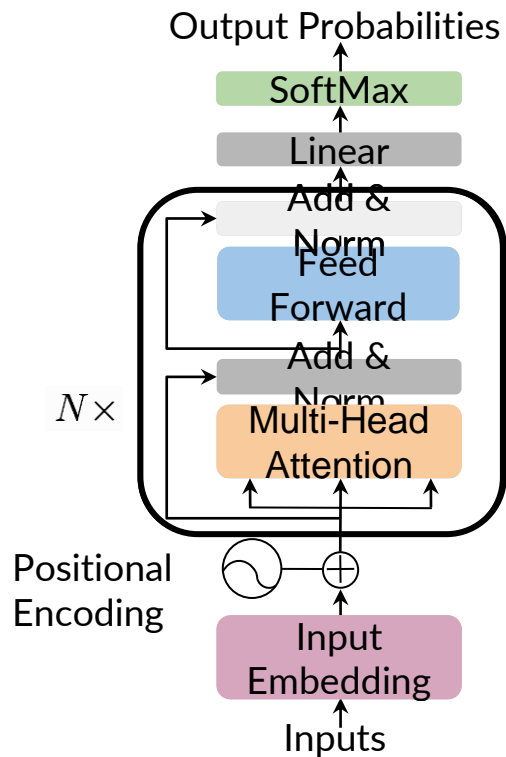
Transformer decoder

Outline

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



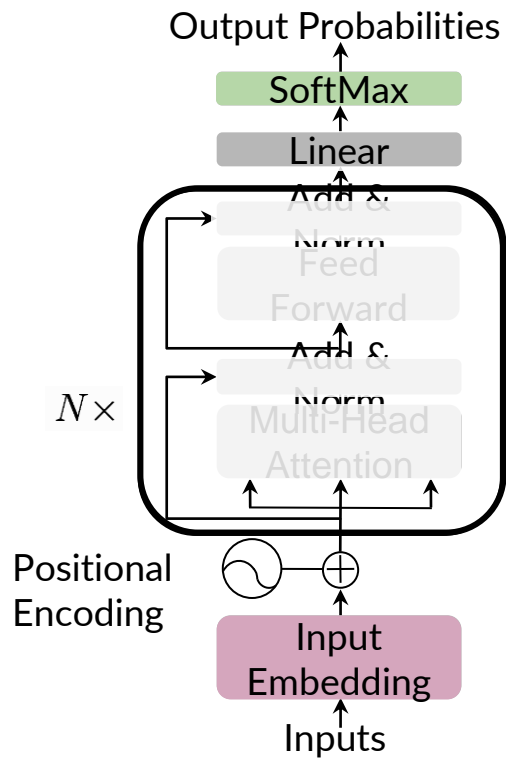
Transformer decoder



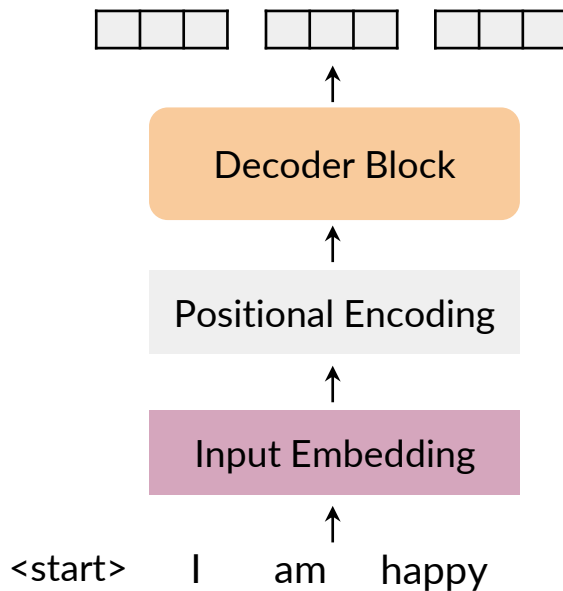
Overview

- input: sentence or paragraph
 - we predict the next word
- sentence gets embedded, add positional encoding $\{0, 1, 2, \dots, K\}$
 - (vectors representing)
- 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

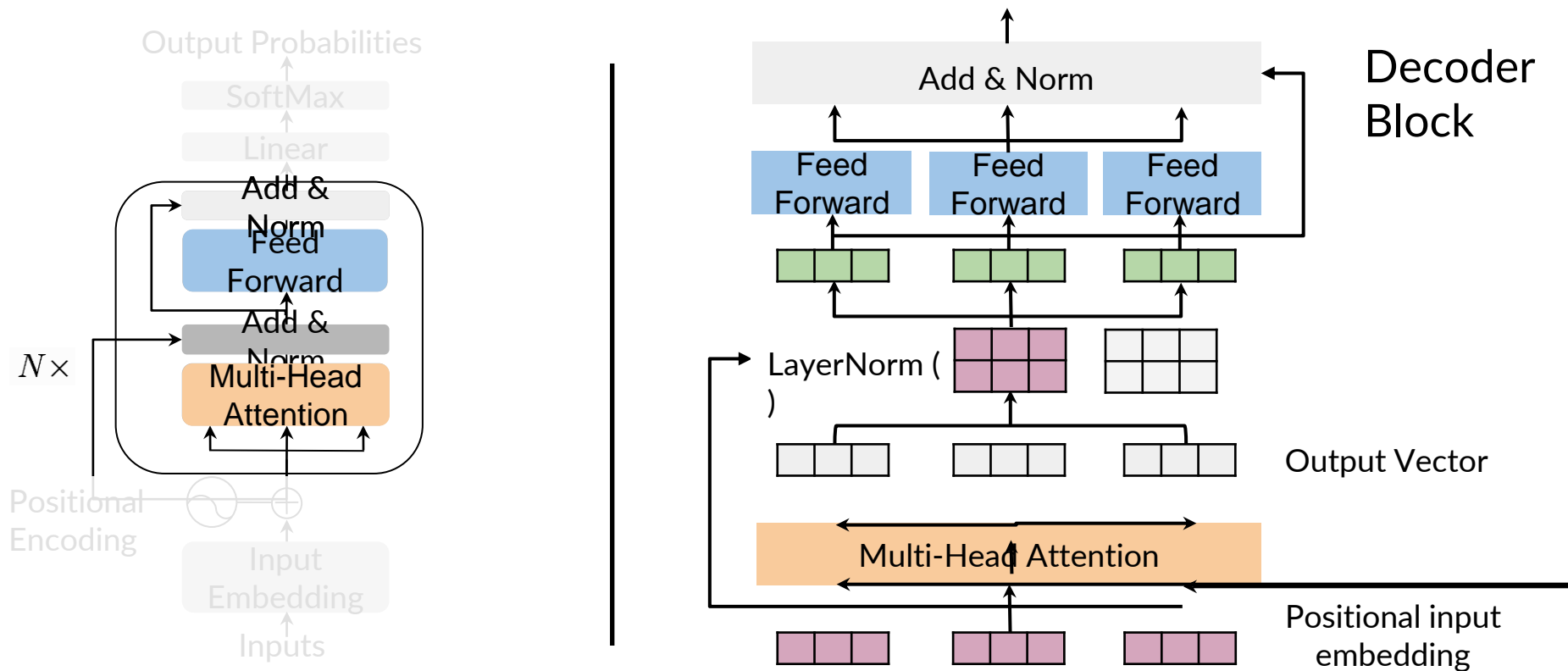
Transformer decoder

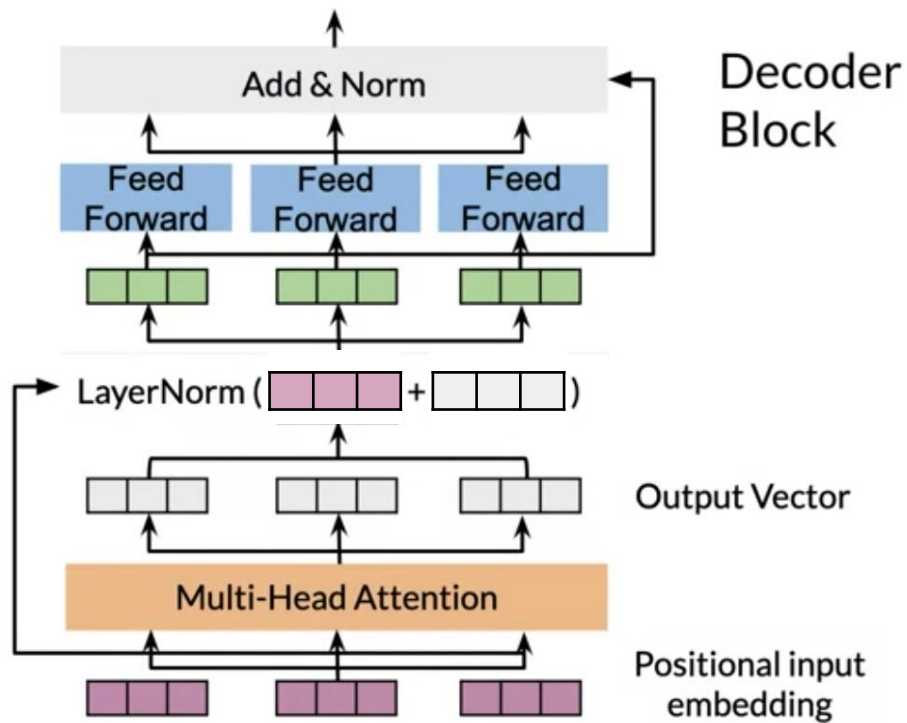


Explanation

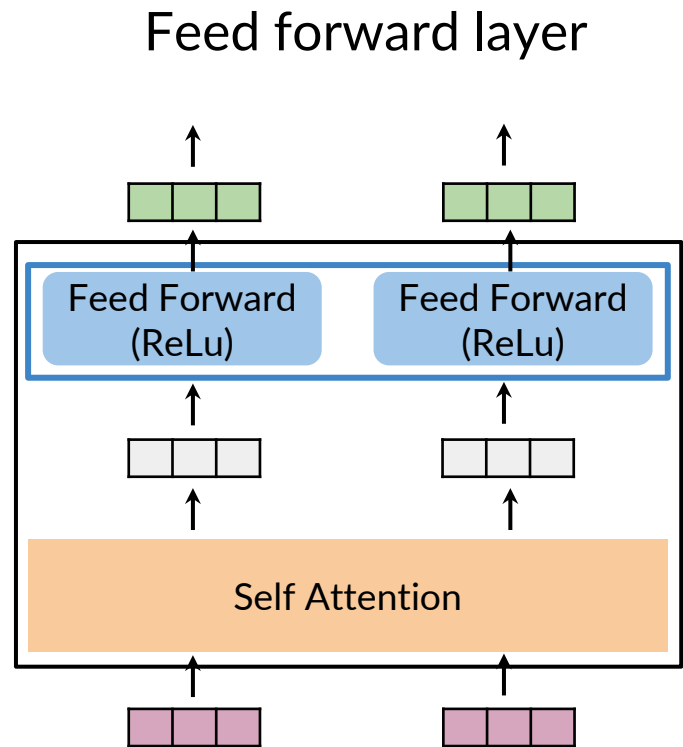
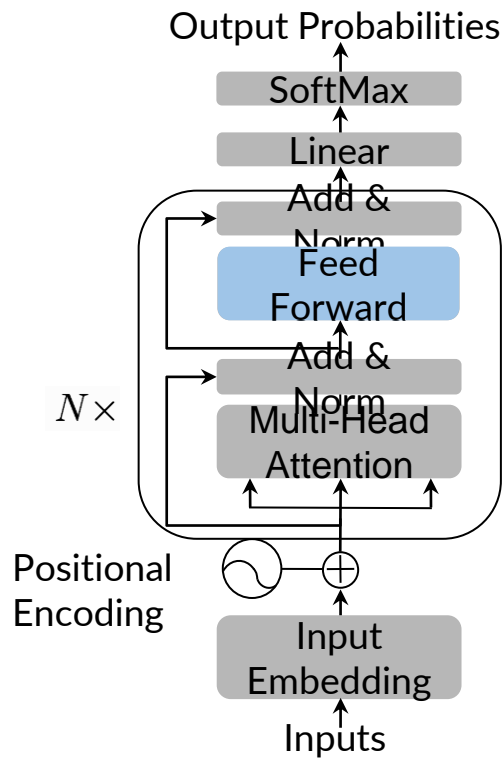


The Transformer decoder

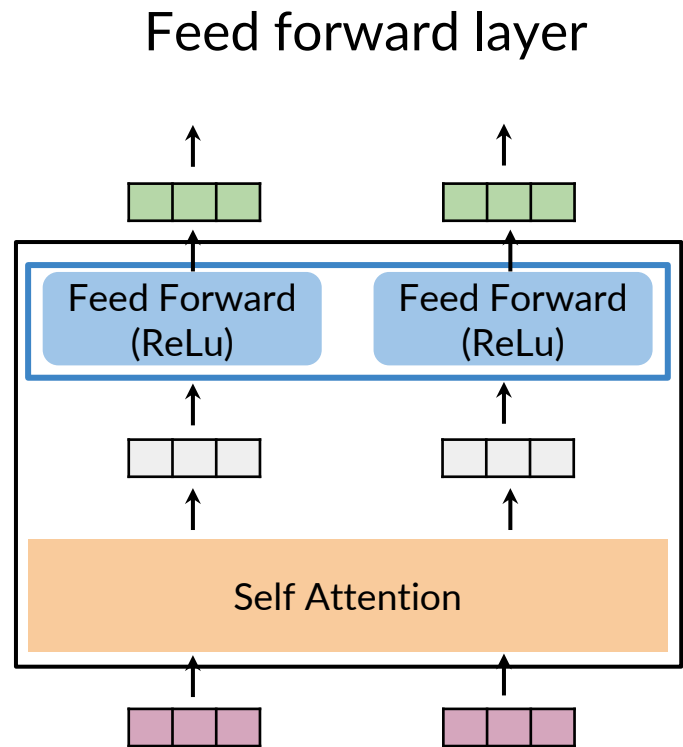
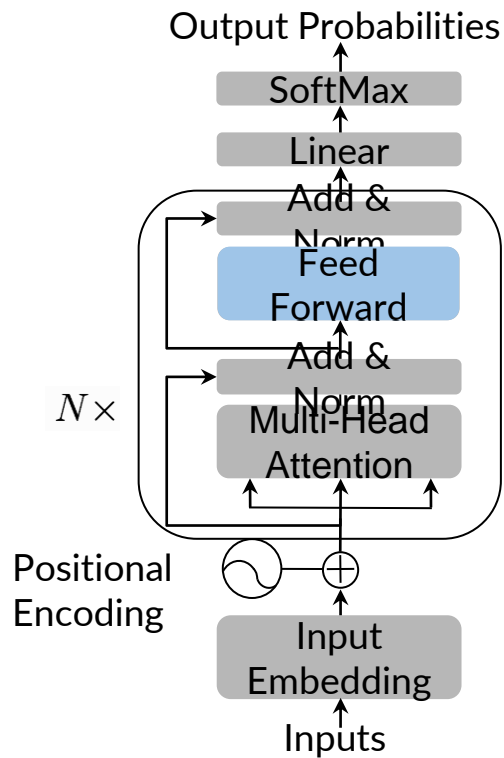




The Transformer decoder



The Transformer decoder



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



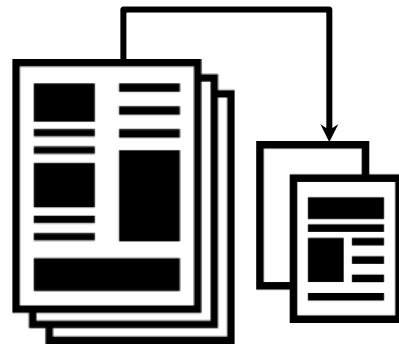


deeplearning.ai

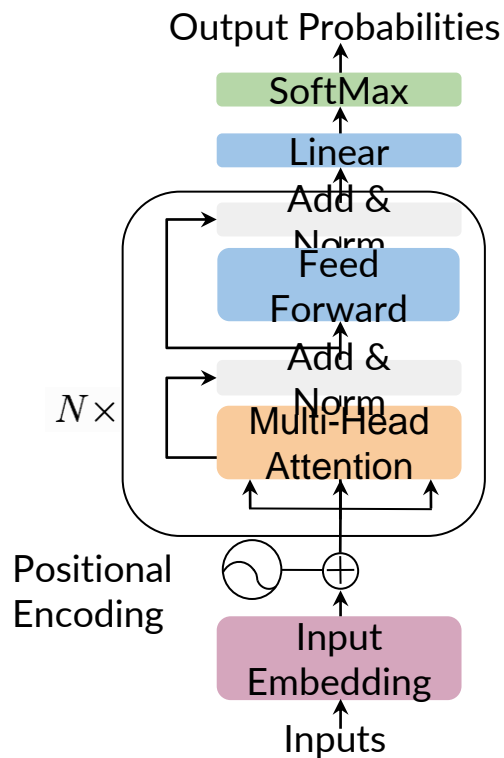
Transformer summarizer

Outline

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



Transformer for summarization



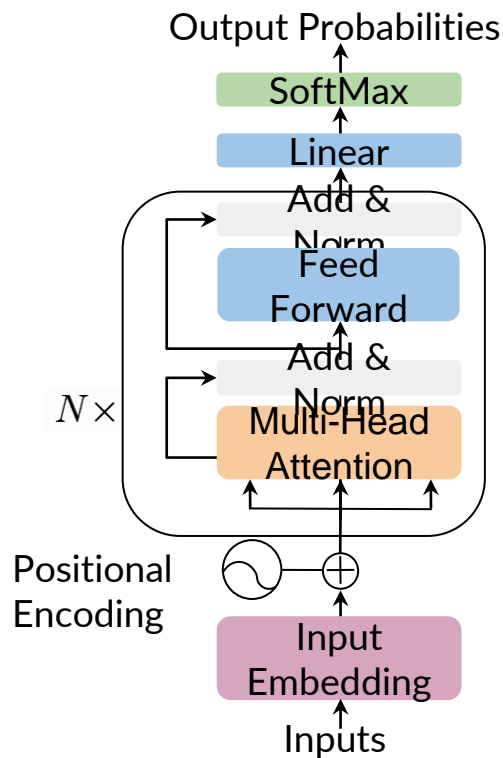
Input



Output:
Summary



Technical details for data processing



Model Input:

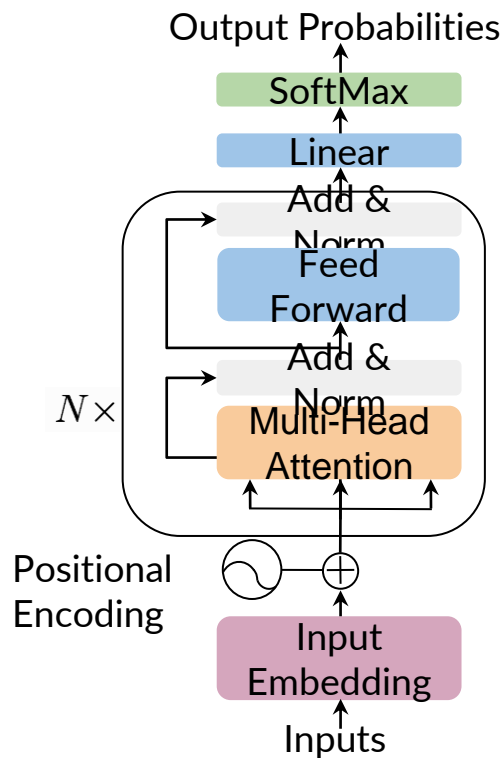
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...

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

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

j : over summary

i : batch 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

