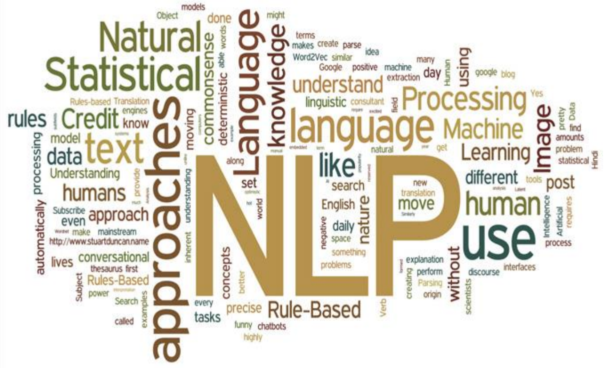


به نام خدا

گروه هوش مصنوعی، دانشکده مهندسی کامپیوتر

بخش چهاردهم

Transformers

حمیدرضا برادران کاشانی

Neural Machine Translation

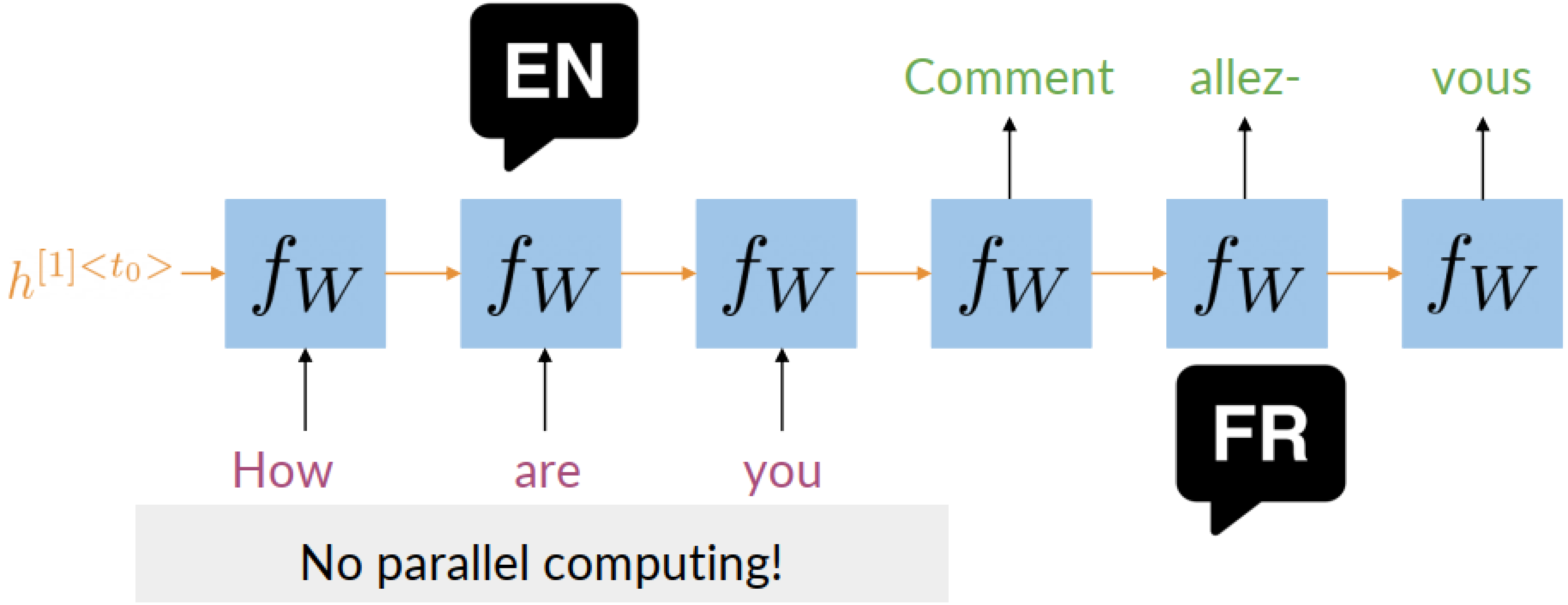
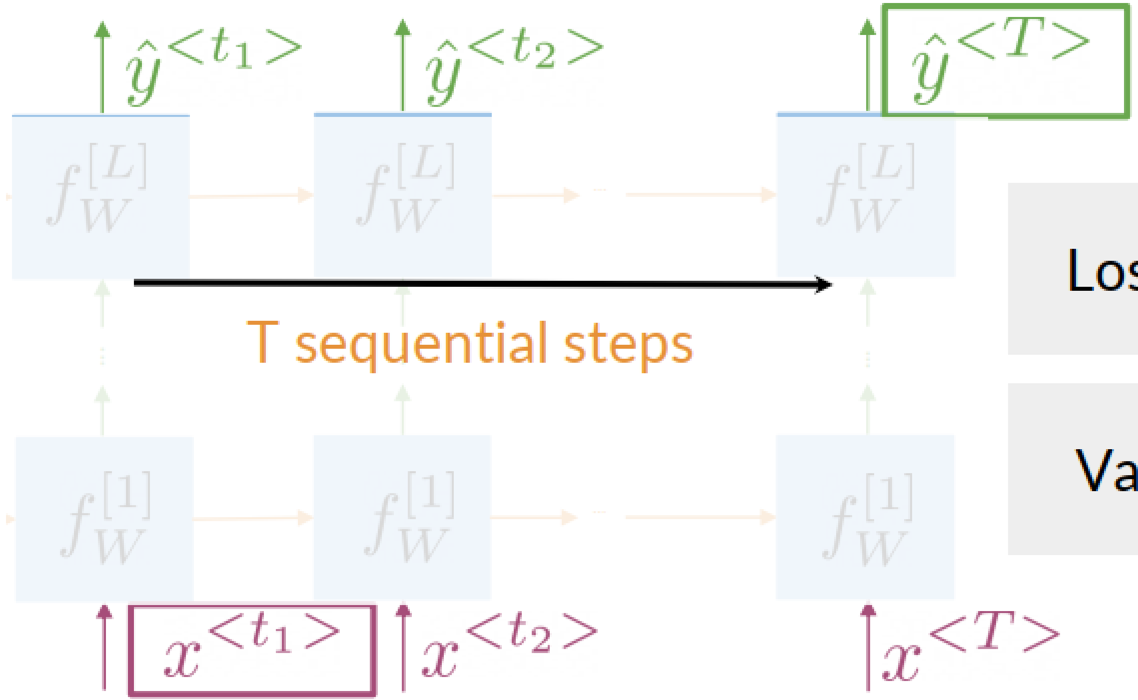


Figure from deeplearning.ai

Sequence-to-sequence architectures

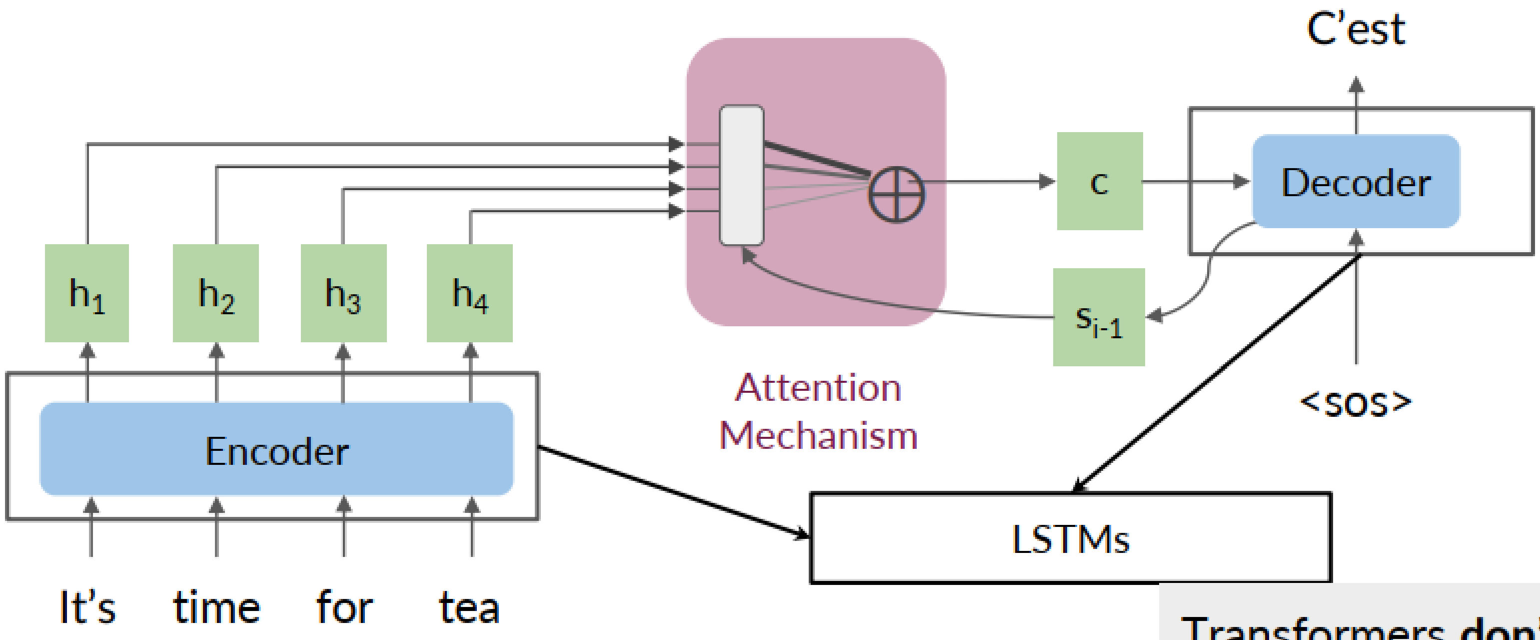


Loss of information

Vanishing gradient

Figure from deeplearning.ai

RNN vs. Transformers



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

Figure from deeplearning.ai

The Transformer Model

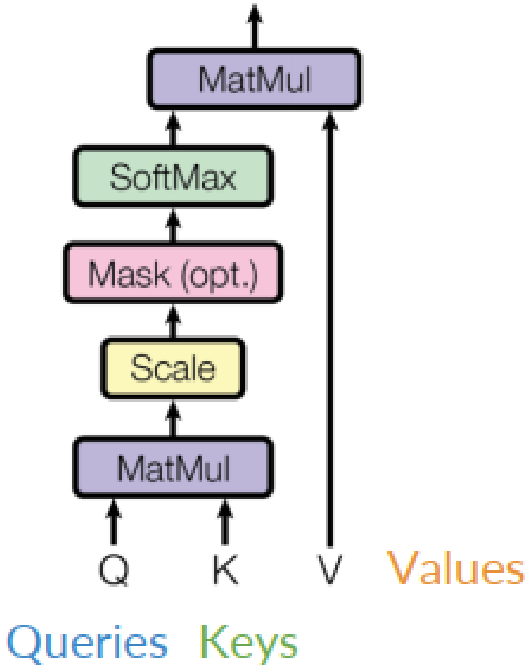
Attention Is All You Need

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

Figure from deeplearning.ai

Scaled Dot-Product Attention

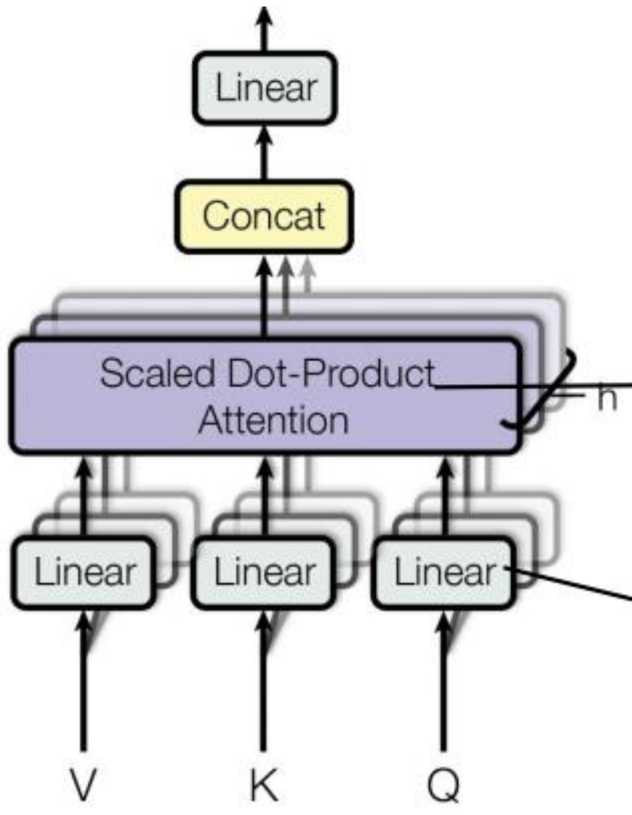


(Vaswani et al., 2017)

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

Figure from deeplearning.ai

Multi-Head Attention

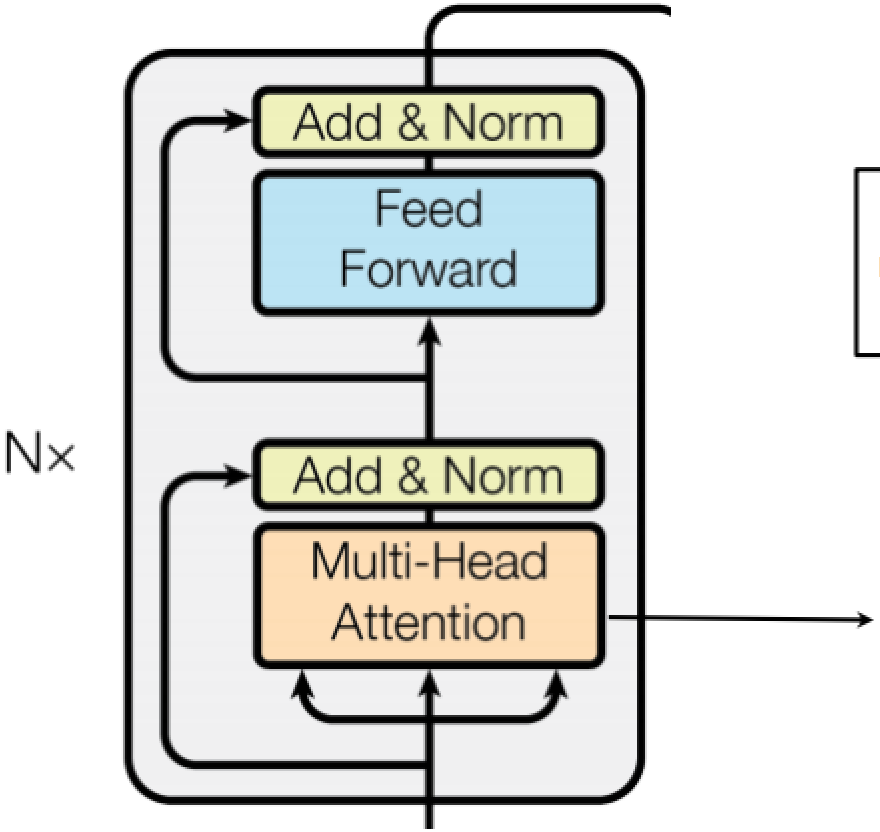


Scaled **dot-product attention multiple times** in parallel

Linear transformations of the input queries, keys and values

Figure from deeplearning.ai

The Encoder



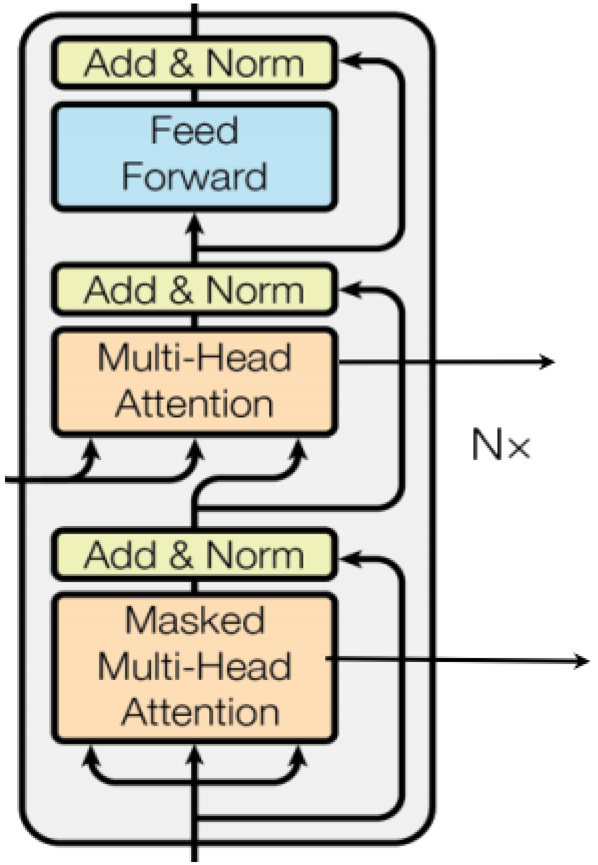
Provides contextual representation of each item in the input sequence

Self-Attention

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

Figure from deeplearning.ai

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

Figure from deeplearning.ai

RNNs vs Transformer: Positional Encoding

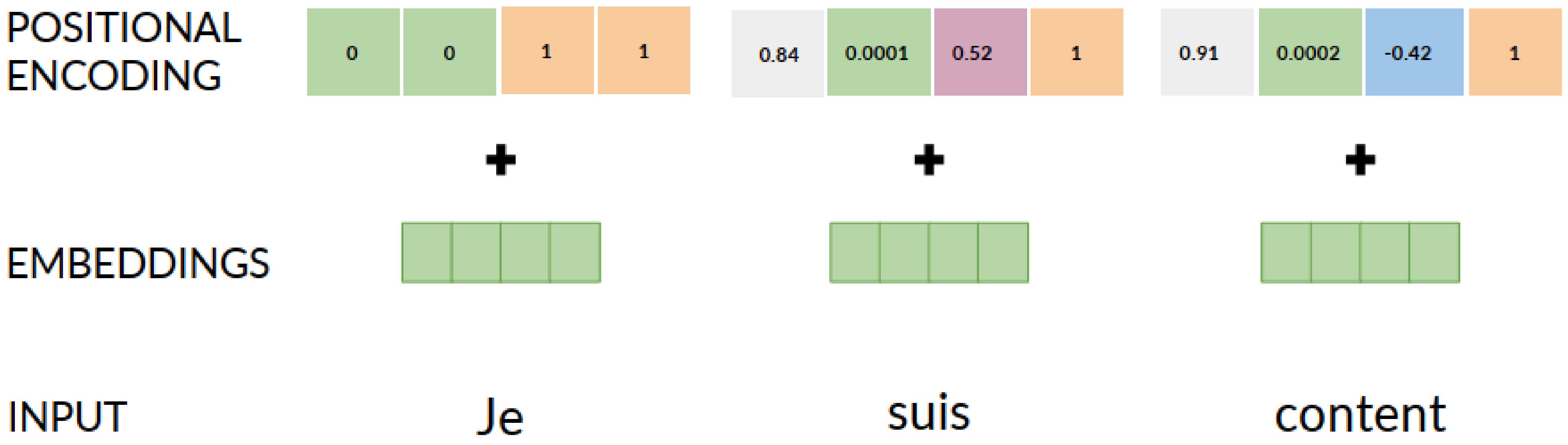
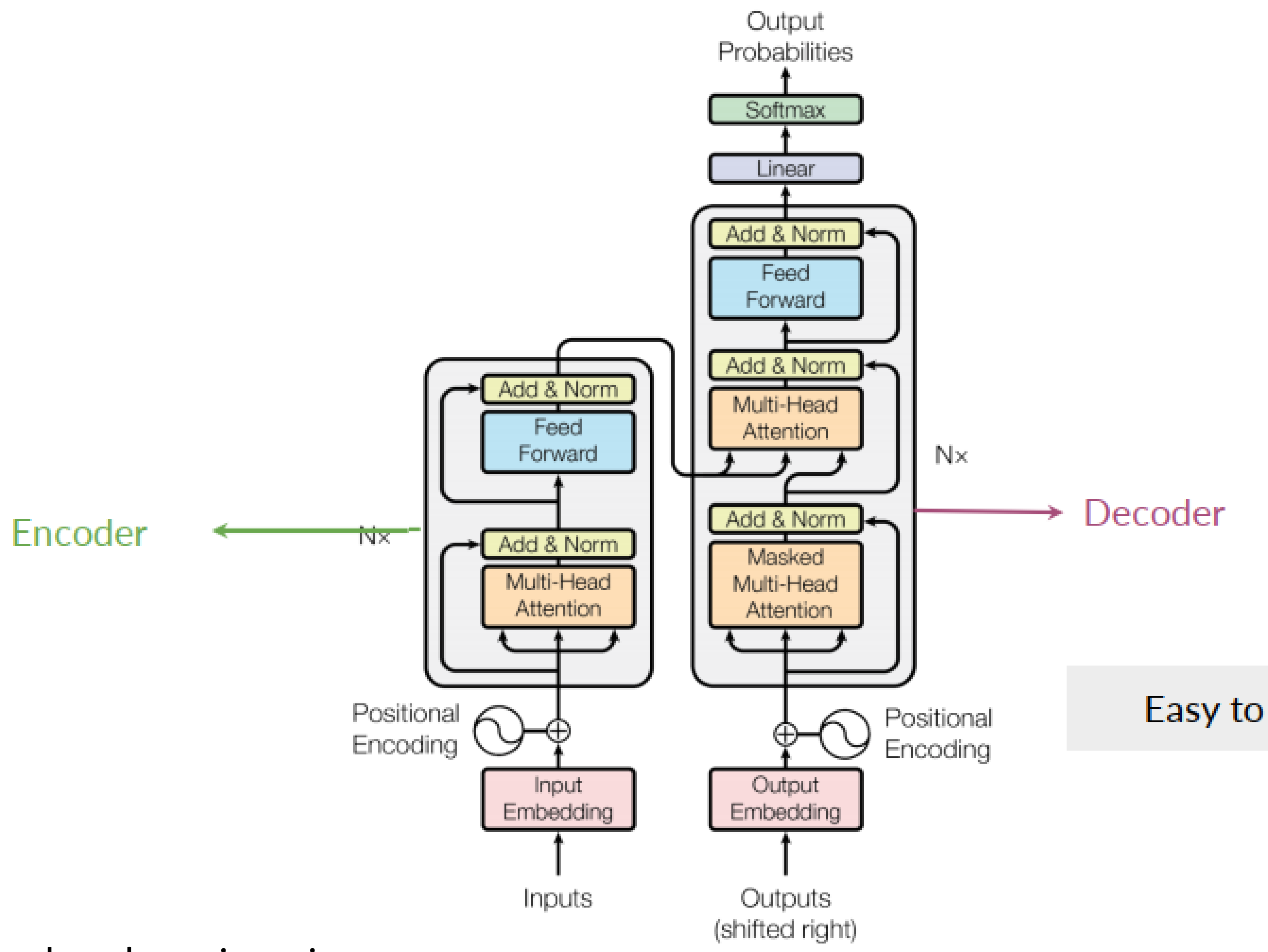


Figure from deeplearning.ai

The Transformer Model



Easy to parallelize!

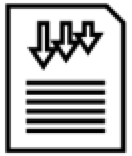
Figure from deeplearning.ai

Summary - 1

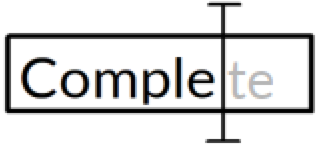
- In RNNs parallel computing is difficult to implement
- For long sequences in RNNs there is loss of information
- In RNNs there is the problem of vanishing gradient
- Transformers help with all of the above

Figure from deeplearning.ai

Transformer NLP applications



Text summarization

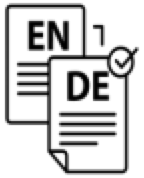


Auto-Complete

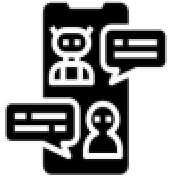
Named entity recognition (NER)

Question answering (Q&A)

Translation



Chat-bots



Other NLP tasks

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

Figure from deeplearning.ai

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

Figure from deeplearning.ai

T5: Text-To-Text Transfer Transformer

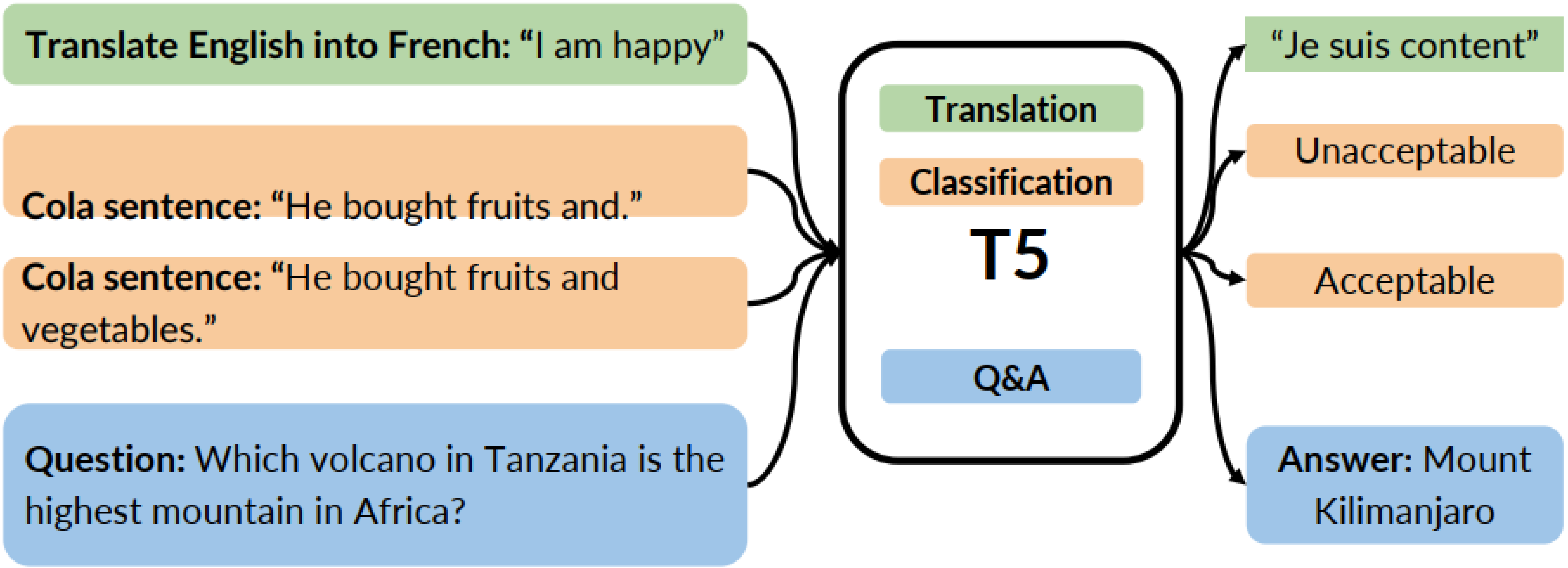


Figure from deeplearning.ai

T5: Text-To-Text Transfer Transformer

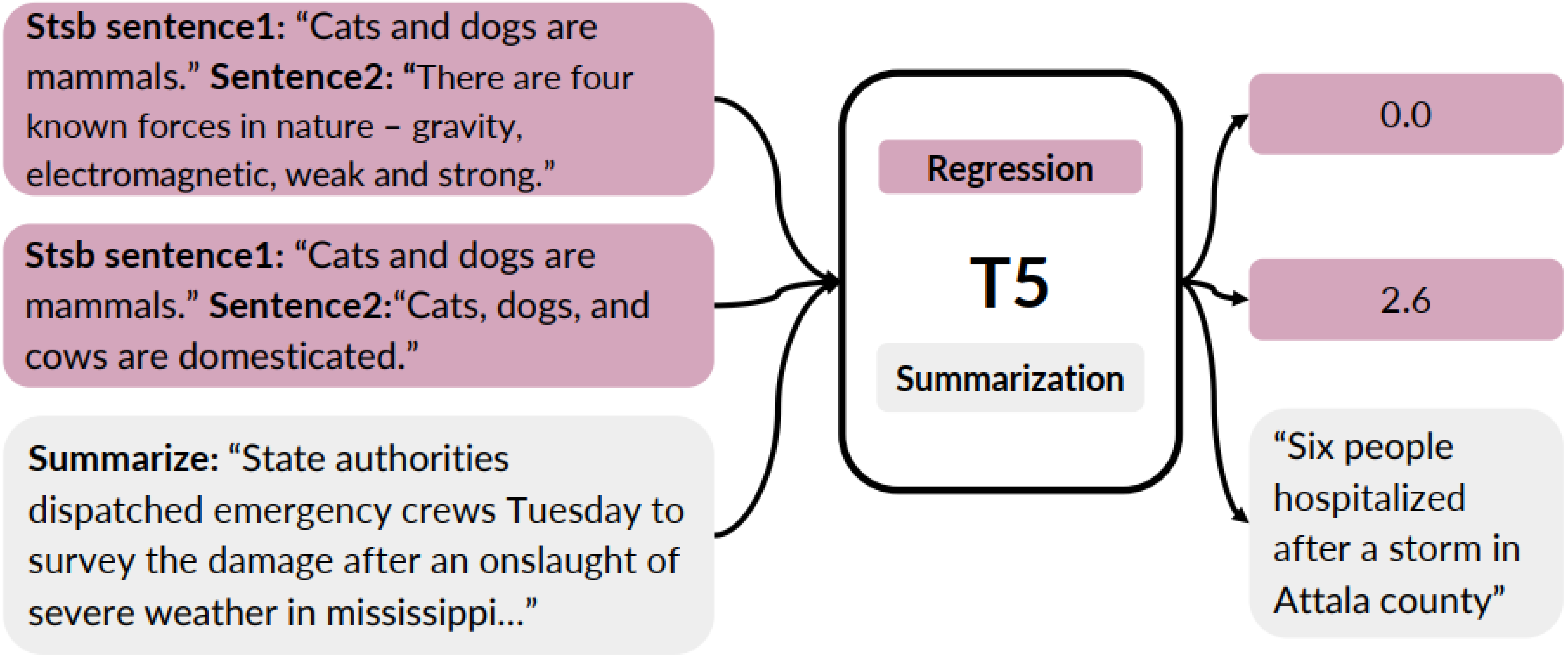
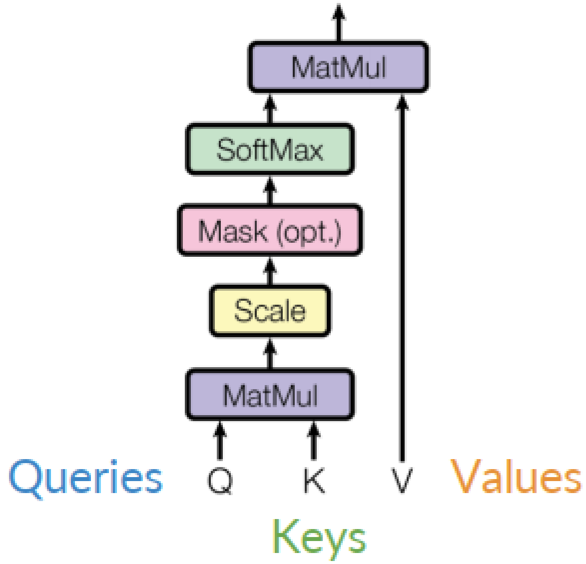


Figure from deeplearning.ai

Scaled dot-product 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!

Figure from deeplearning.ai

Queries, Keys and Values

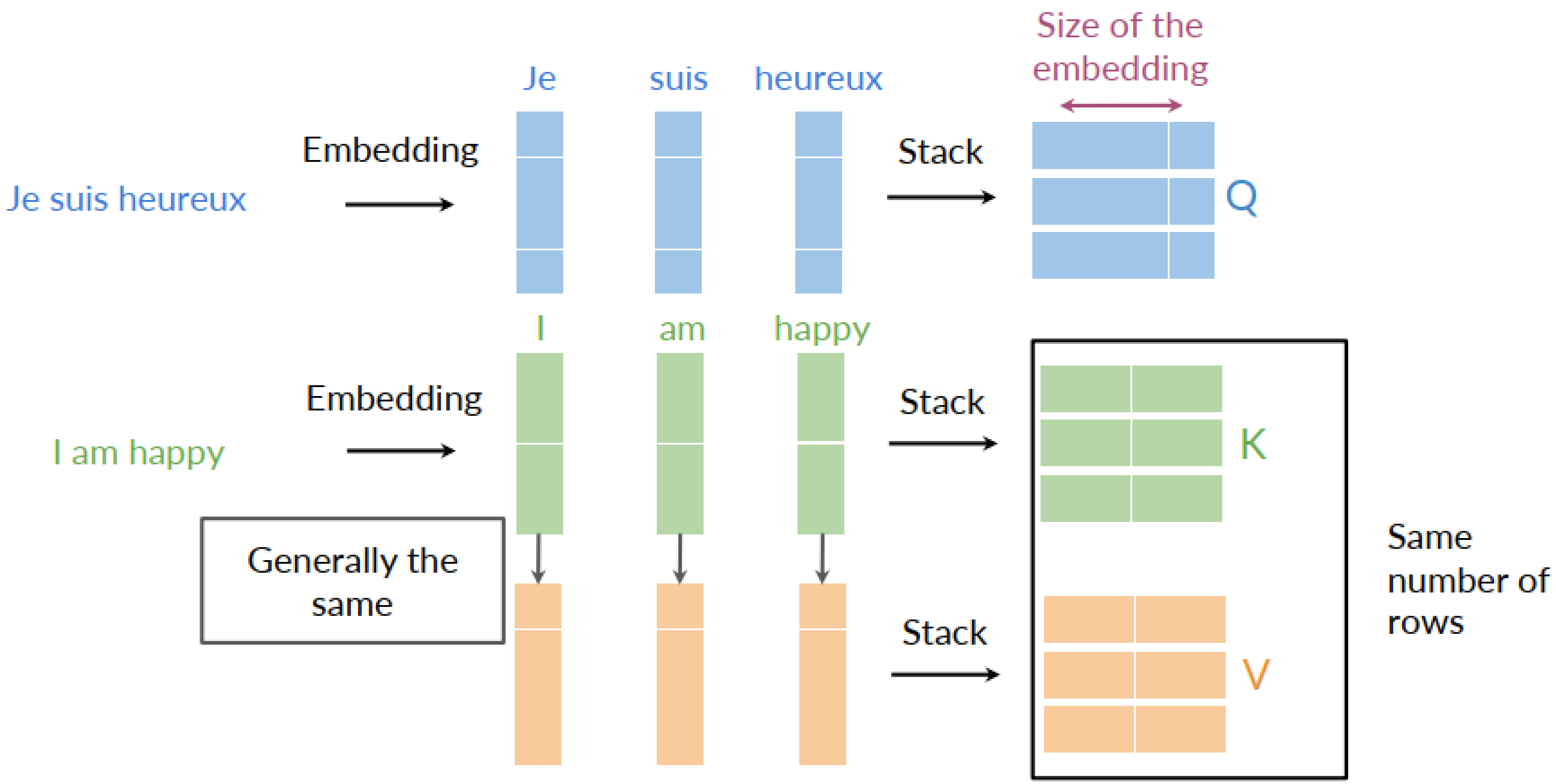
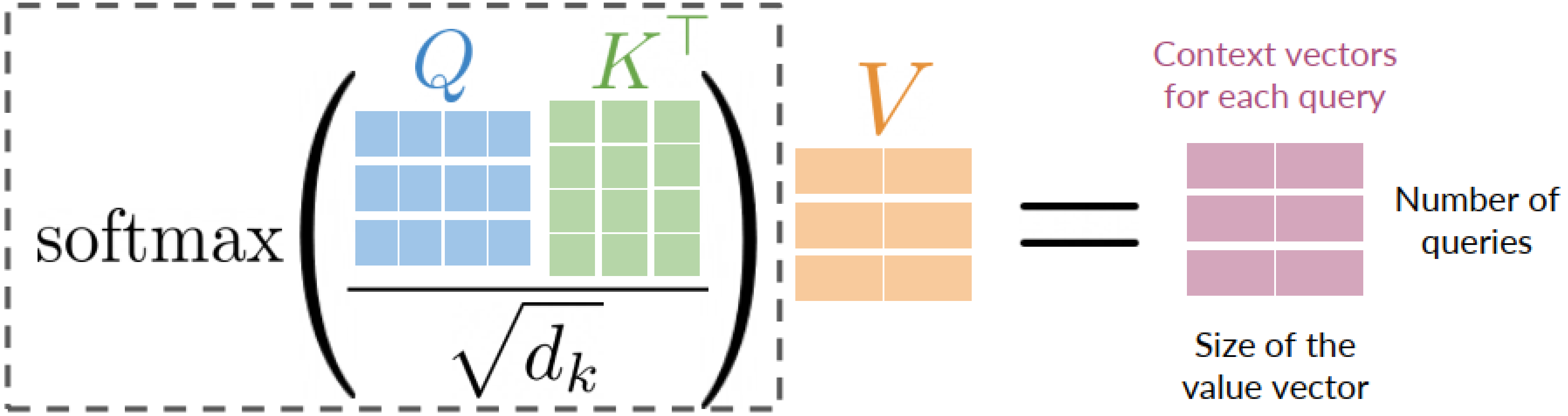
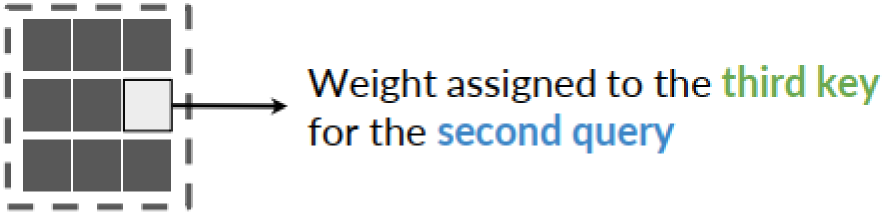


Figure from deeplearning.ai

Attention Math





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

Figure from deeplearning.ai

Three ways of attention

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

Queries, Keys, values and Attention

Figure from deeplearning.ai

Encoder-Decoder Attention

Queries from one sentence, keys and values from another

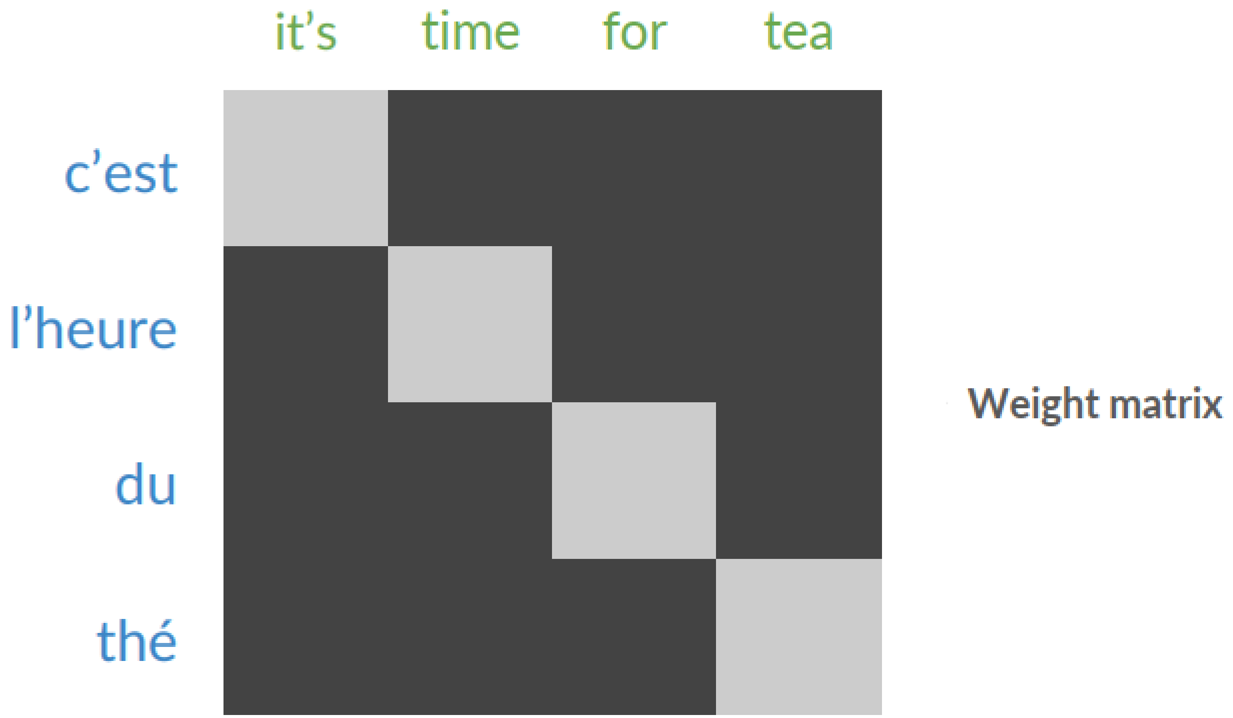
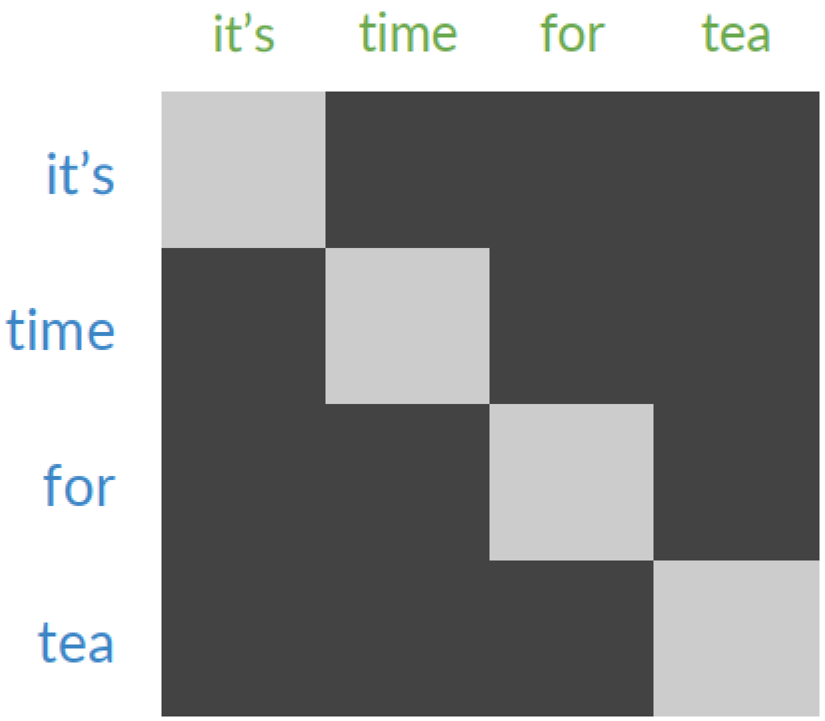


Figure from deeplearning.ai

Self-Attention

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



Weight matrix

Meaning of each word **within** the sentence

Figure from deeplearning.ai

Masked Self-Attention

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

Figure from deeplearning.ai

Masked self-attention math

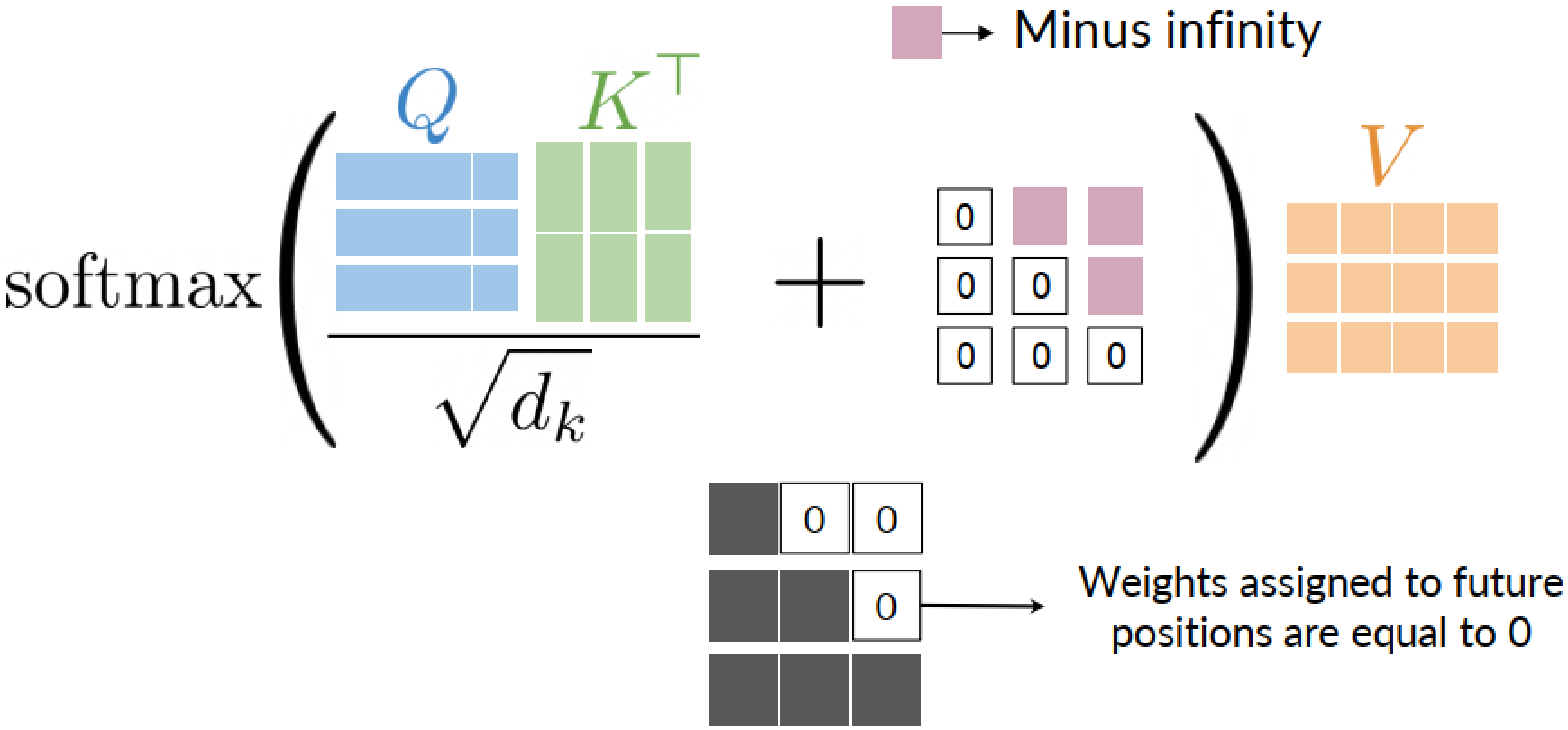


Figure from deeplearning.ai

Multi-Head Attention - Overview

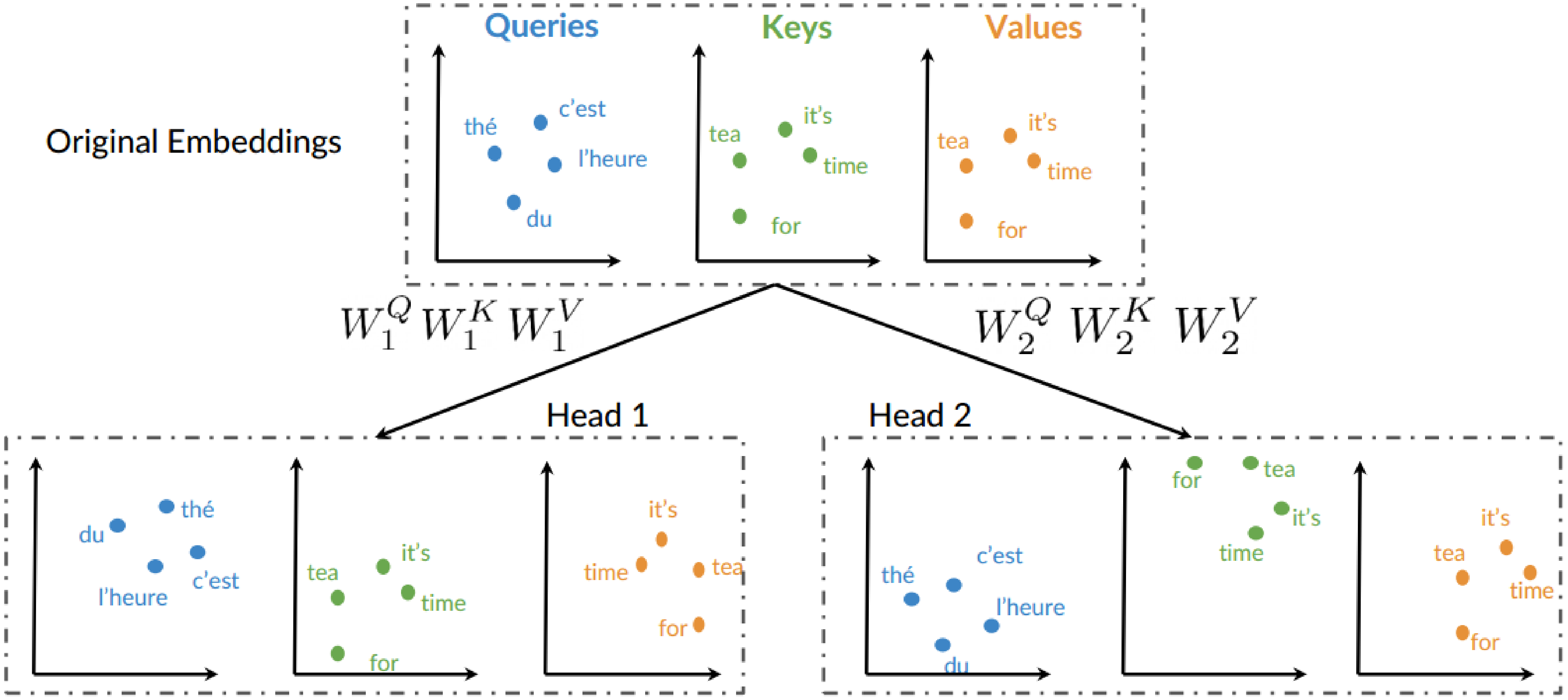


Figure from deeplearning.ai

Multi-Head Attention - Overview

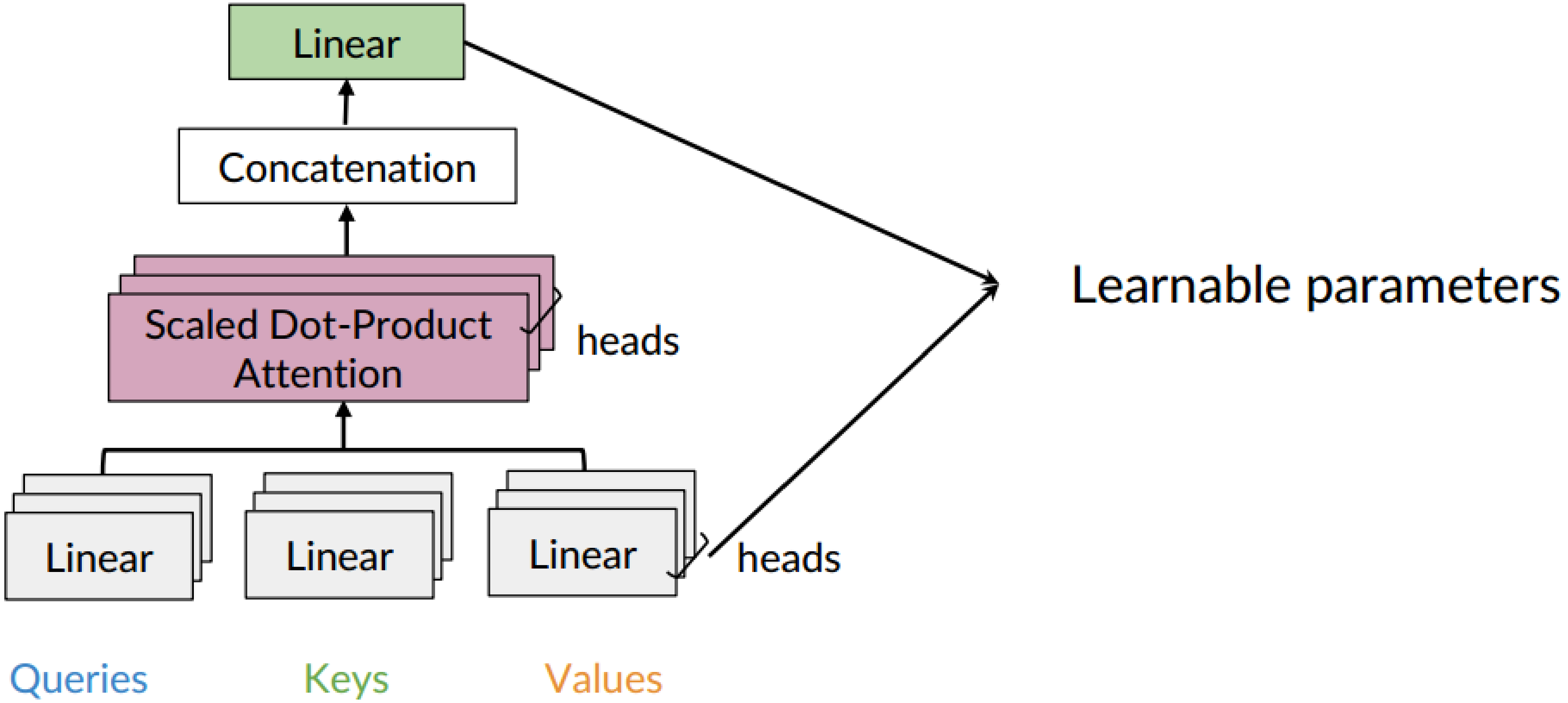
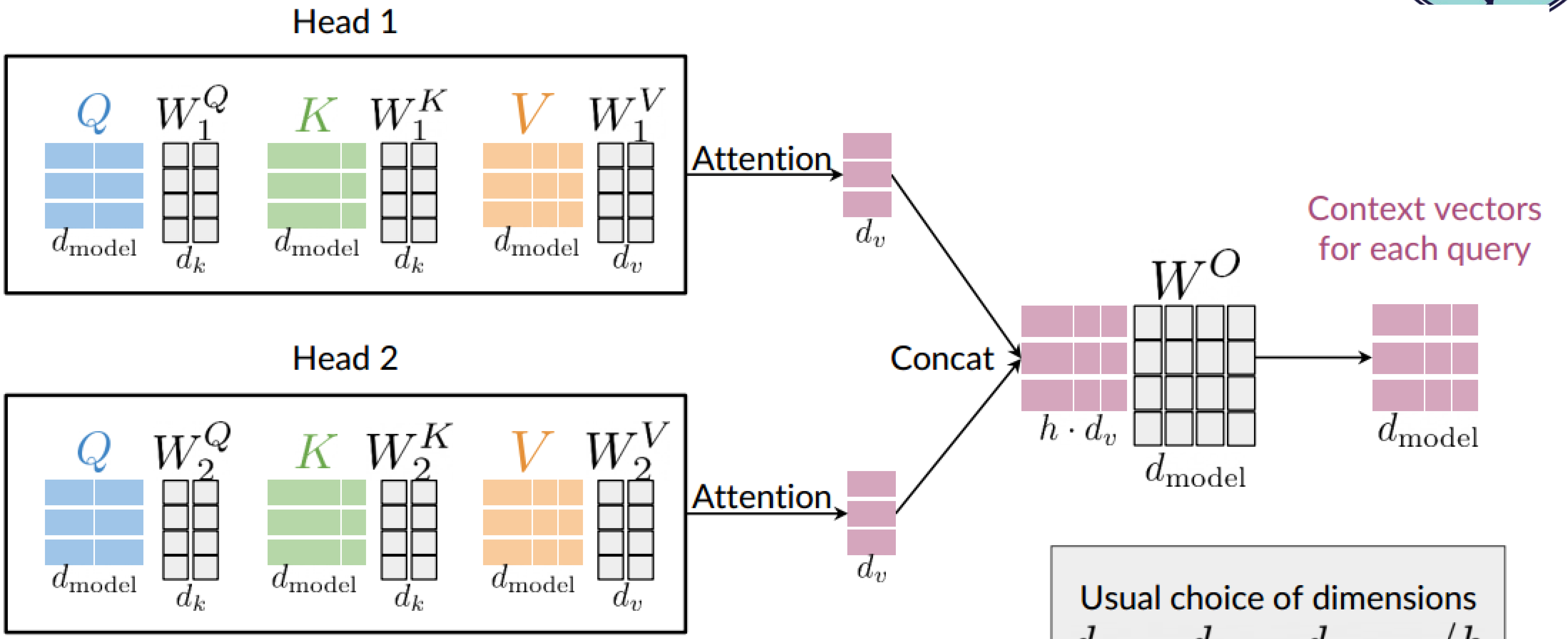


Figure from deeplearning.ai

Multi-Head Attention

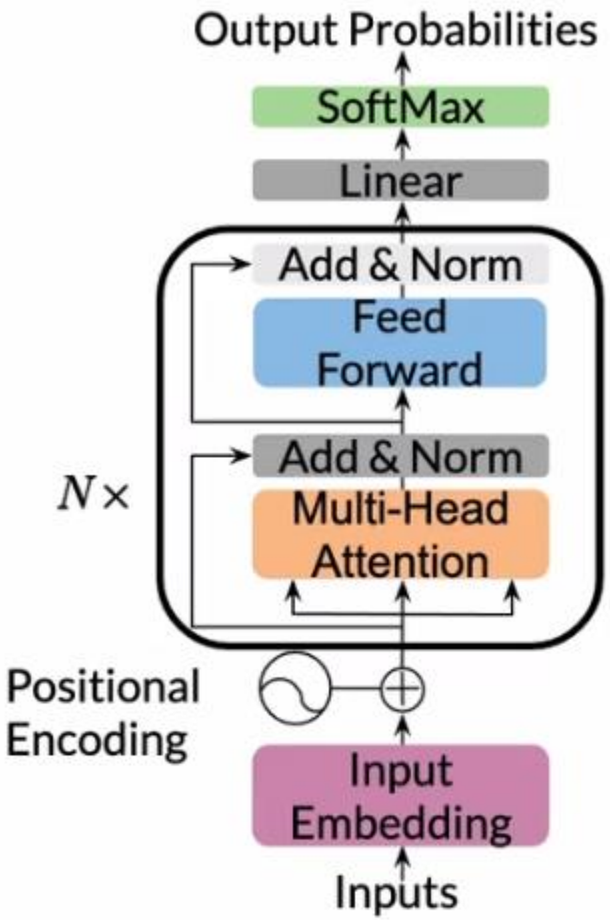


Usual choice of dimensions
$$d_k = d_v = d_{\text{model}}/h$$

d_{model} : Embedding size

Figure from deeplearning.ai

GPT-2: Transformer Decoder

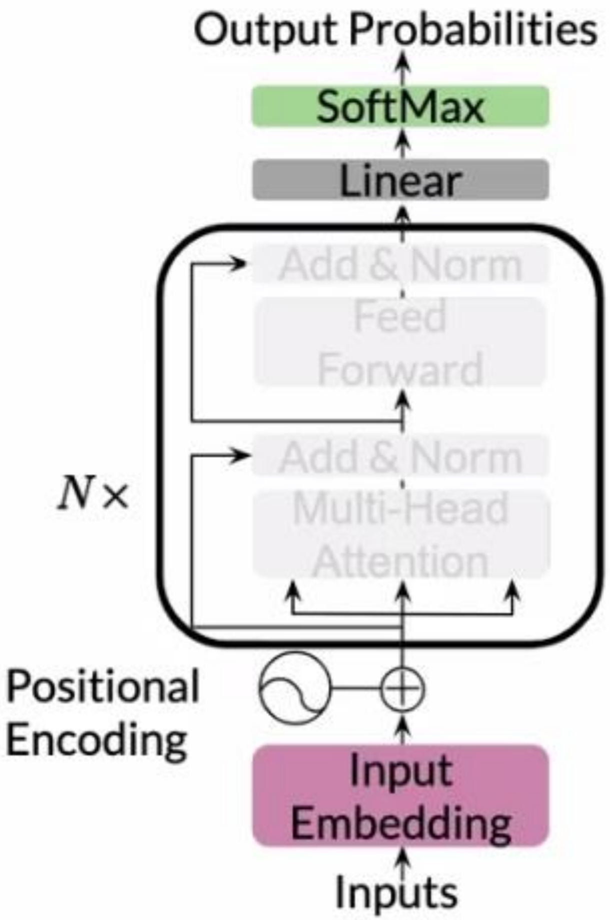


Overview

- 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

Figure from deeplearning.ai

GPT-2: Transformer Decoder



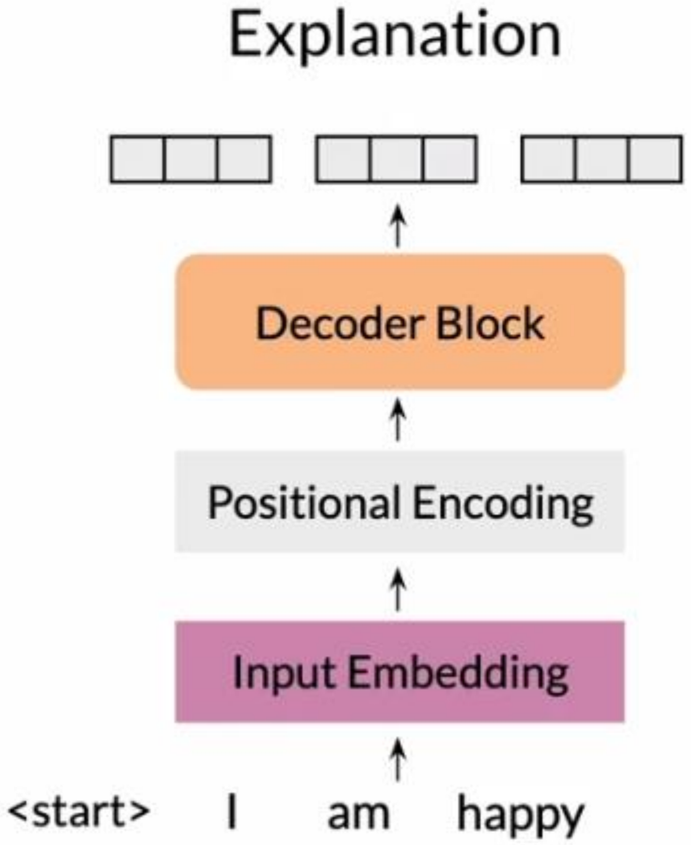


Figure from deeplearning.ai

GPT-2: Transformer Decoder

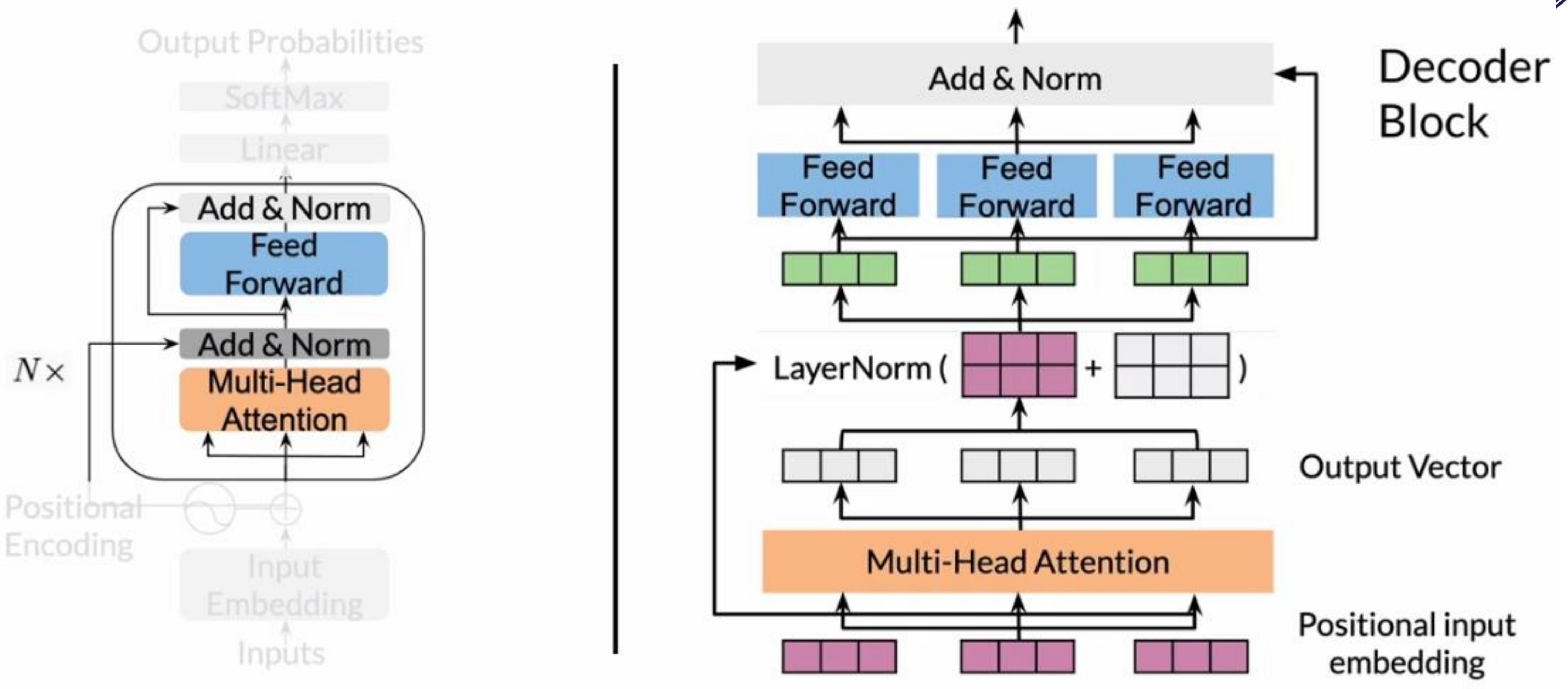
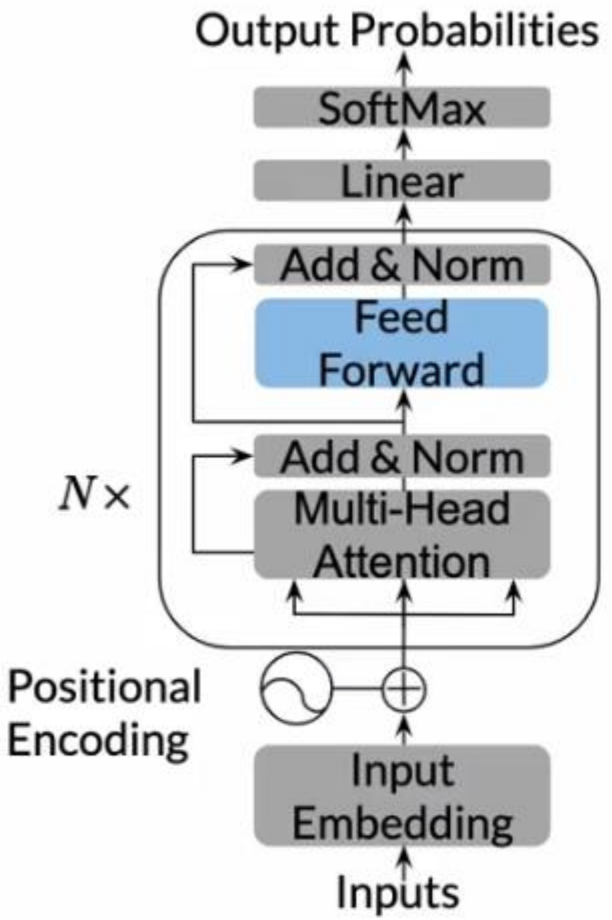


Figure from deeplearning.ai

GPT-2: Transformer Decoder



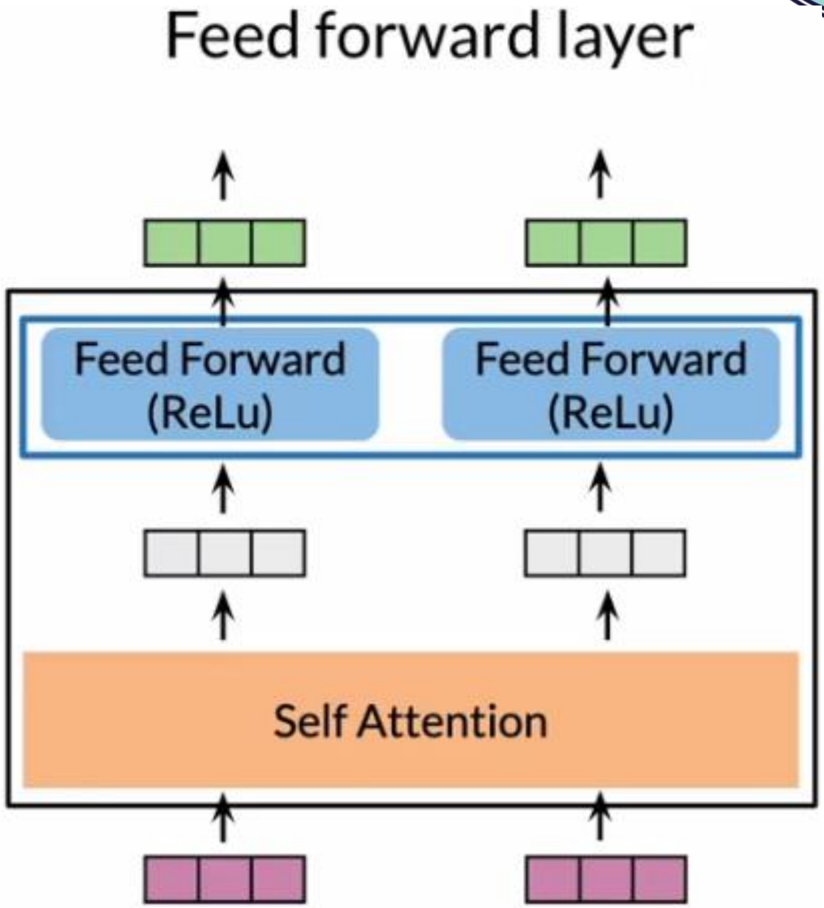
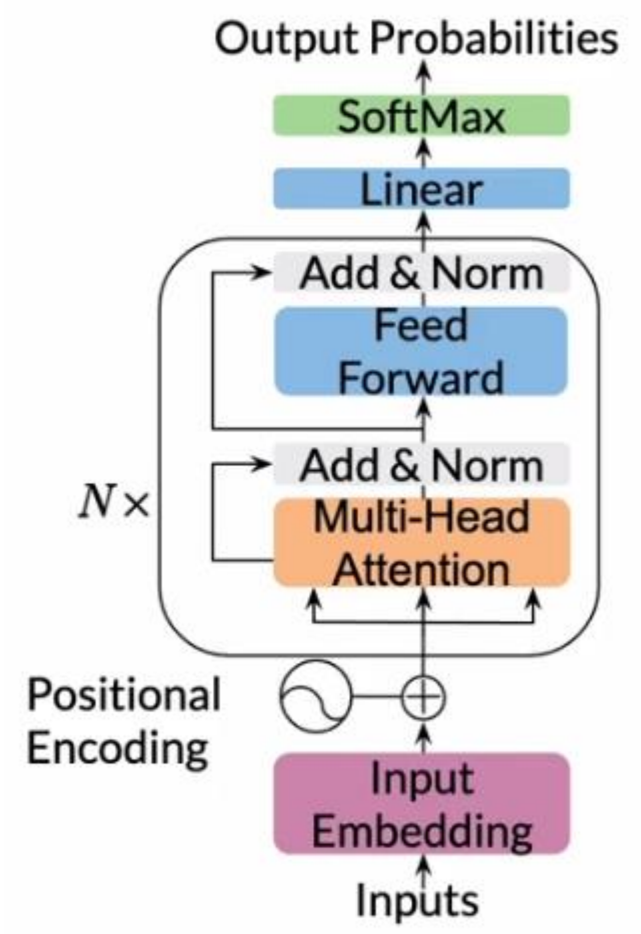


Figure from deeplearning.ai

Transformer for summarization



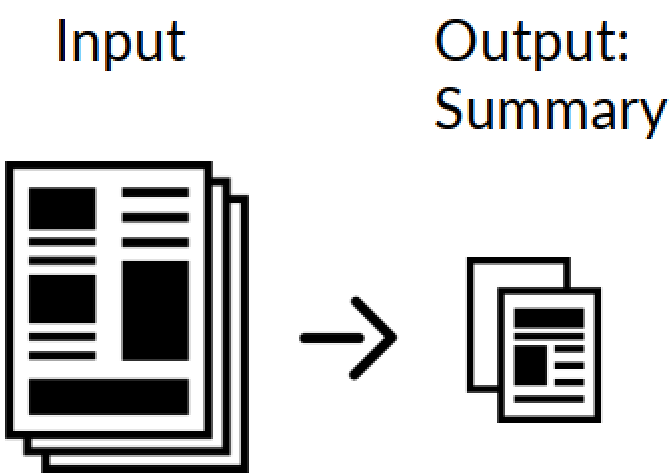


Figure from deeplearning.ai

Technical details for data processing

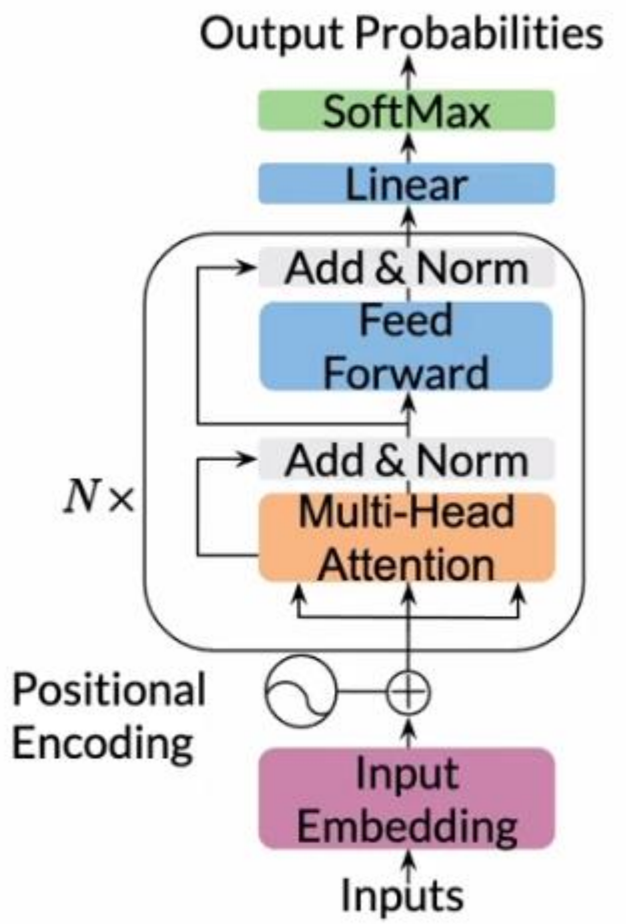


Figure from deeplearning.ai

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.

when there is little data for the summaries, it actually helps to weight the article loss with non-zero numbers, say 0.2 or 0.5 or even 1.