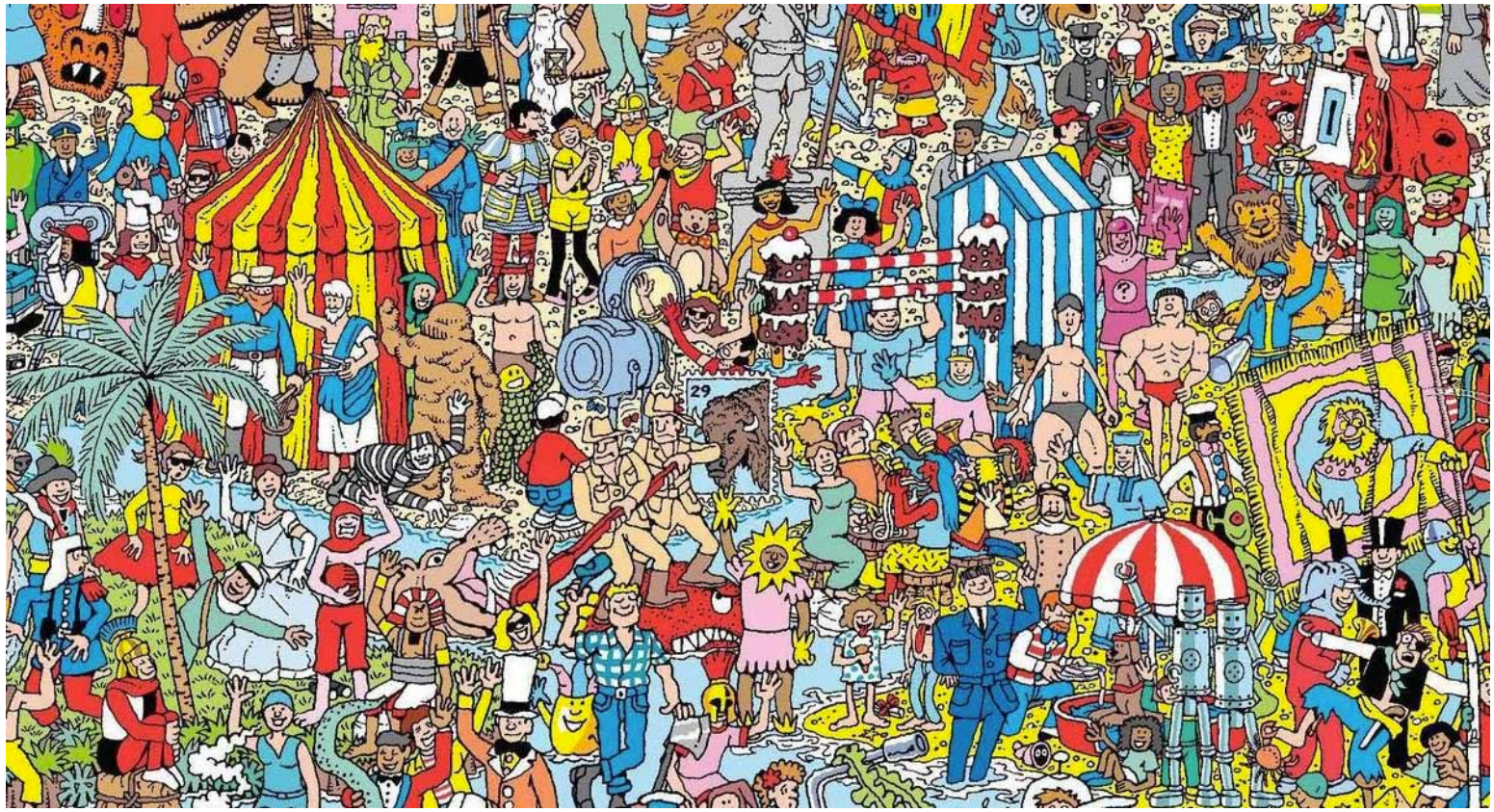
Transformers

John Tan Chong Min

30 May 2022

Introduction

- Find Waldo



What issues did you face?

- Too much information?
- What are the winning strategies?

Introduction

- Activity:
  - Try to memorize a sequence (grab a paper and pen)
- You are only allowed to write 5 digits on your paper
- These 5 digits can be changed anytime you want
- Ready?

Start!

- 1 6 2 3 8

2nd line

- <Hidden>
- 8 7 4 3 4

3rd line

- <Hidden>
- <Hidden>
- 2 4 9 3 0

Code

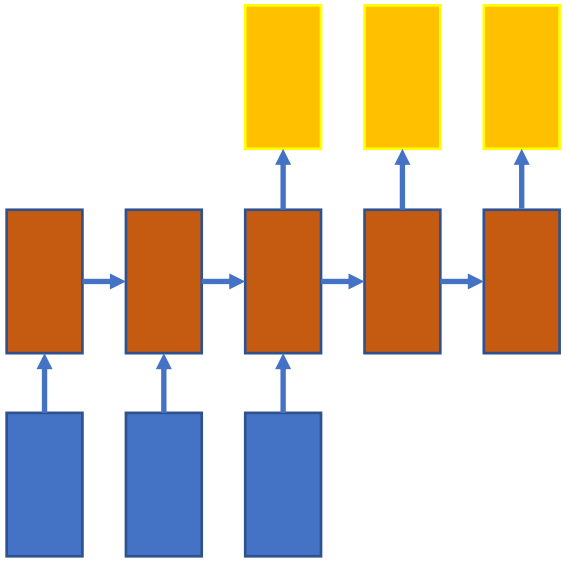
- <Hidden>
- <Hidden>
- <Hidden>

Answer

- 1 6 2 3 8
- 8 7 4 3 4
- 2 4 9 3 0

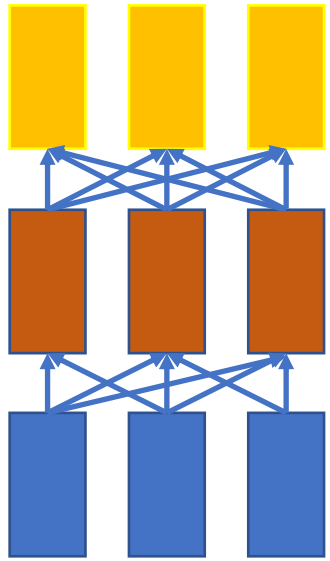
Problem with existing networks

- Bottleneck



RNNs

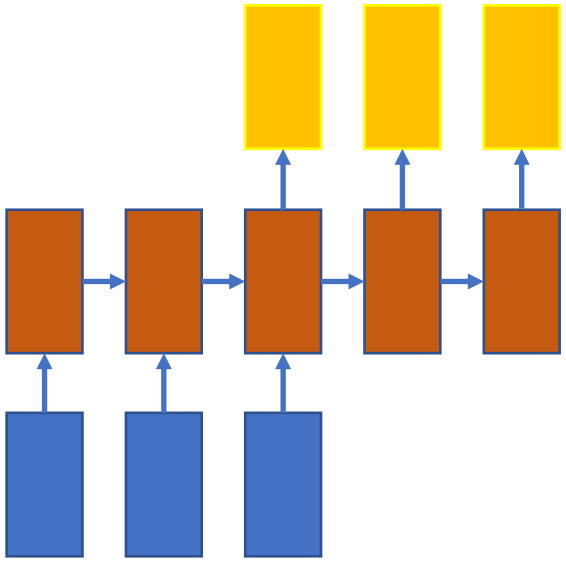
- Lack of sequential info
- No sparsity = high chance of overfitting



Fully-connected model

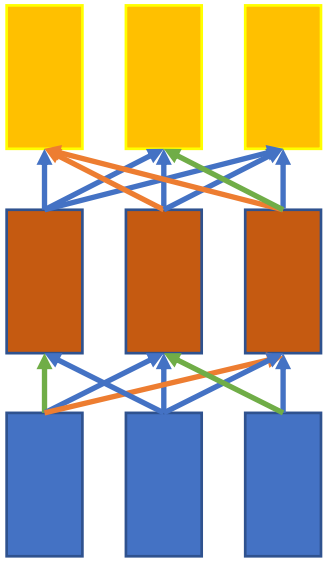
Potential Solution

- Bottleneck



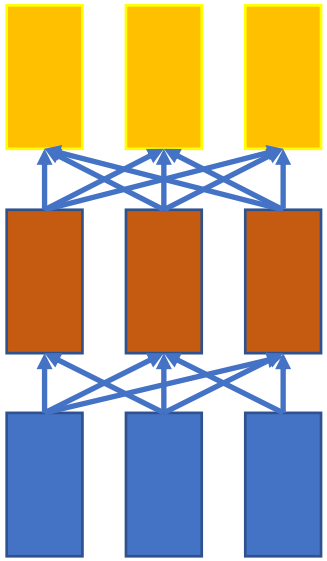
RNNs

- All information in input can potentially pass through the network
- Connection is learnable via similarity in embedding space (Attention)
- Sequence info: Positional Embeddings



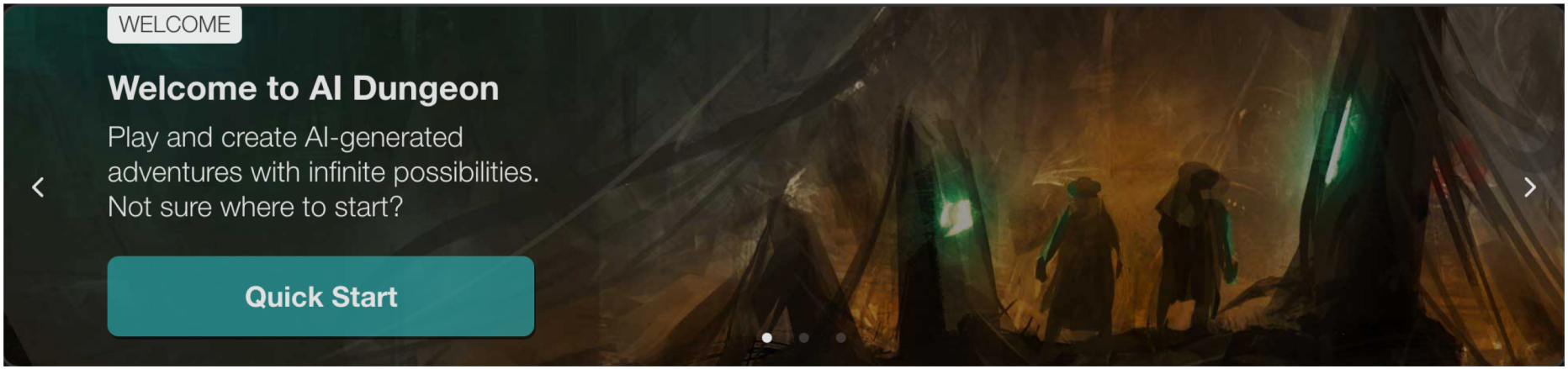
Transformers

- Lack of sequential info
- No sparsity = high chance of overfitting



Fully-connected model

Try-it-out Transformers!

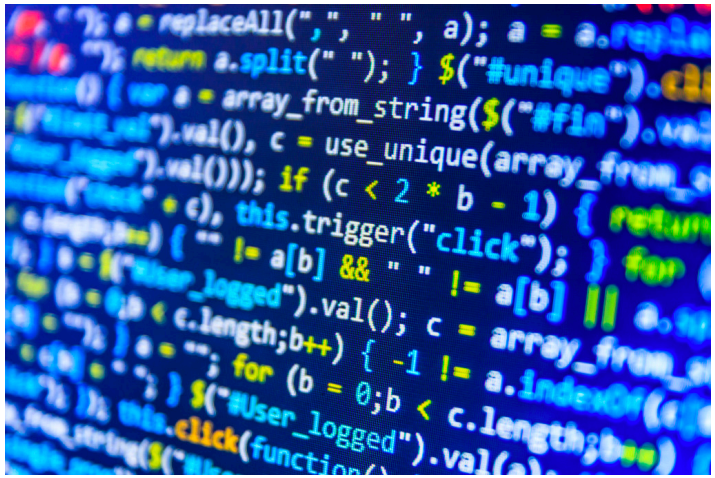


- AI Dungeon:
  - https://play.aidungeon.io/

Other things to try out

- Text generator:
  - https://deepai.org/machine-learning-model/text-generator

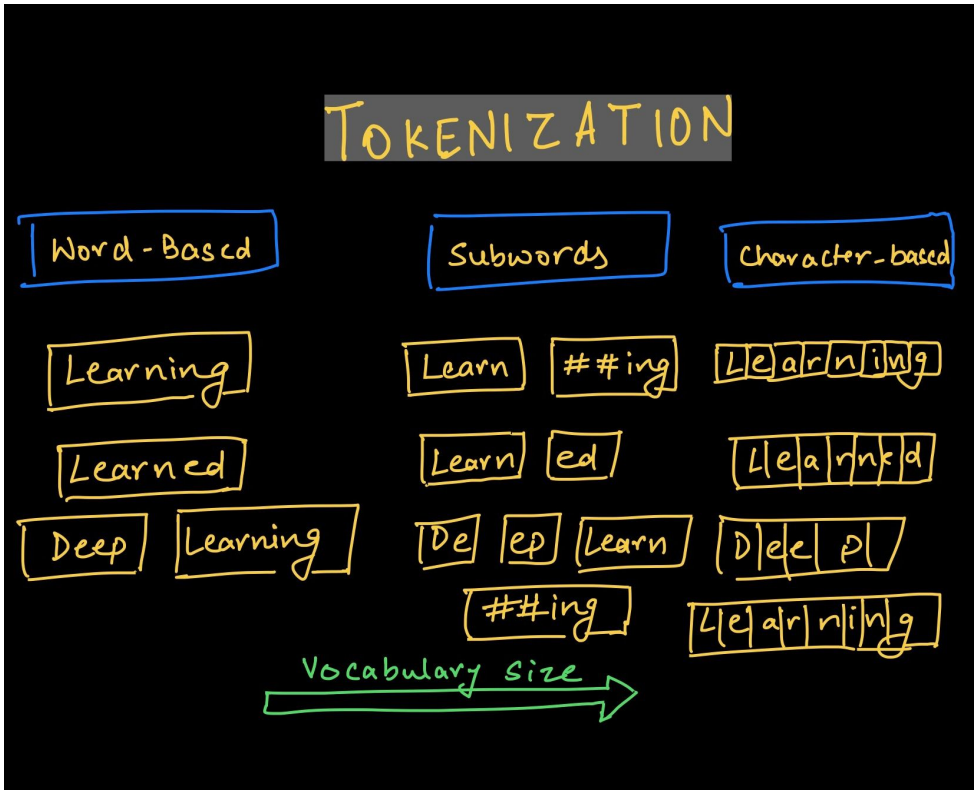
- Competitive Programming Solver (not perfect):
  - https://alphacode.deepmind.com/



Pre-processing

Word tokenization

Varying levels of tokenization



- Word-based: More semantic meaning, more tokens
- Character-based: Less semantic meaning, fewer tokens

https://www.freecodecamp.org/news/evolution-of-tokenization/

One-hot Encoding

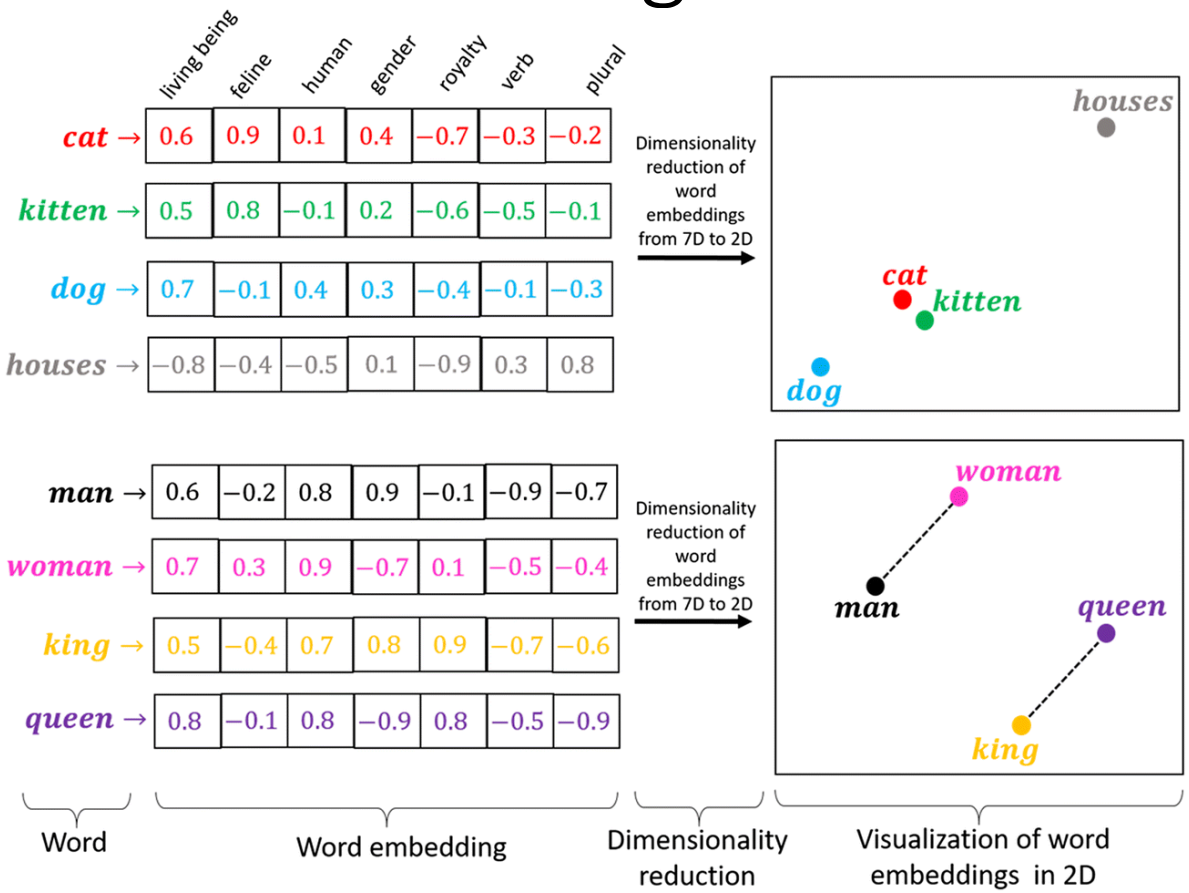
- Vocabulary = {I, he, she, like, Machine, Learning, <eos>}
- Sentence = “I like Machine Learning <eos>”
- Embedding =

	I	he	she	like	Machine	Learning	<eos>
I	1	0	0	0	0	0	0
like	0	0	0	1	0	0	0
Machine	0	0	0	0	1	0	0
Learning	0	0	0	0	0	1	0
<eos>	0	0	0	0	0	0	1

Embedding Space

Semantic Meaning

Word Embeddings



- Extracting semantic meaning in higher-dimensional space
- Number of dimensions depends on use case

Taken from: https://medium.com/@hari4om/word-embedding-d816f643140

Transformer Architecture

Overall view

Transformers: Overall

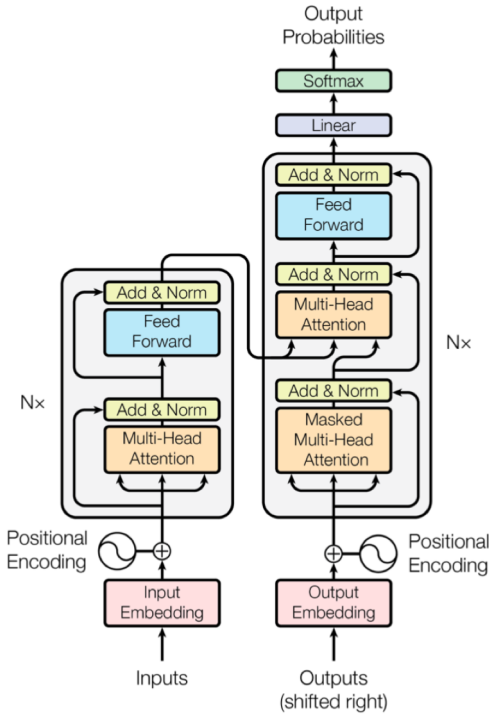


Figure 1: The Transformer - model architecture.

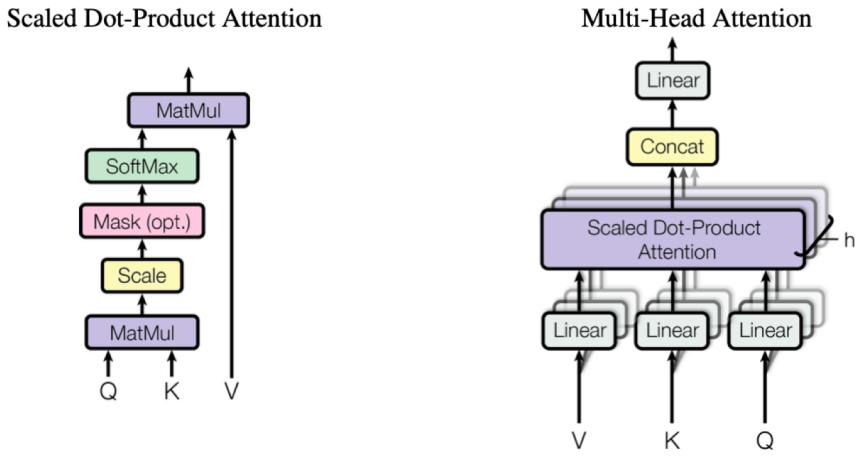


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

Taken from: Attention is all you need. Vaswani et al. (2017)

Encoder-Decoder Structure

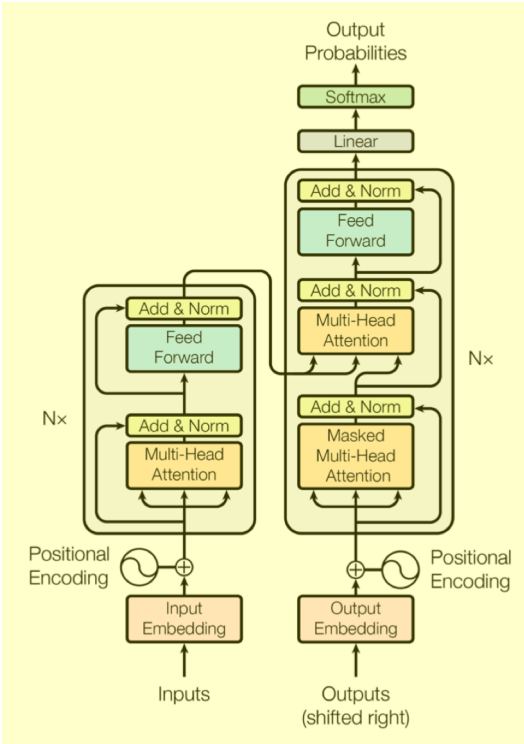
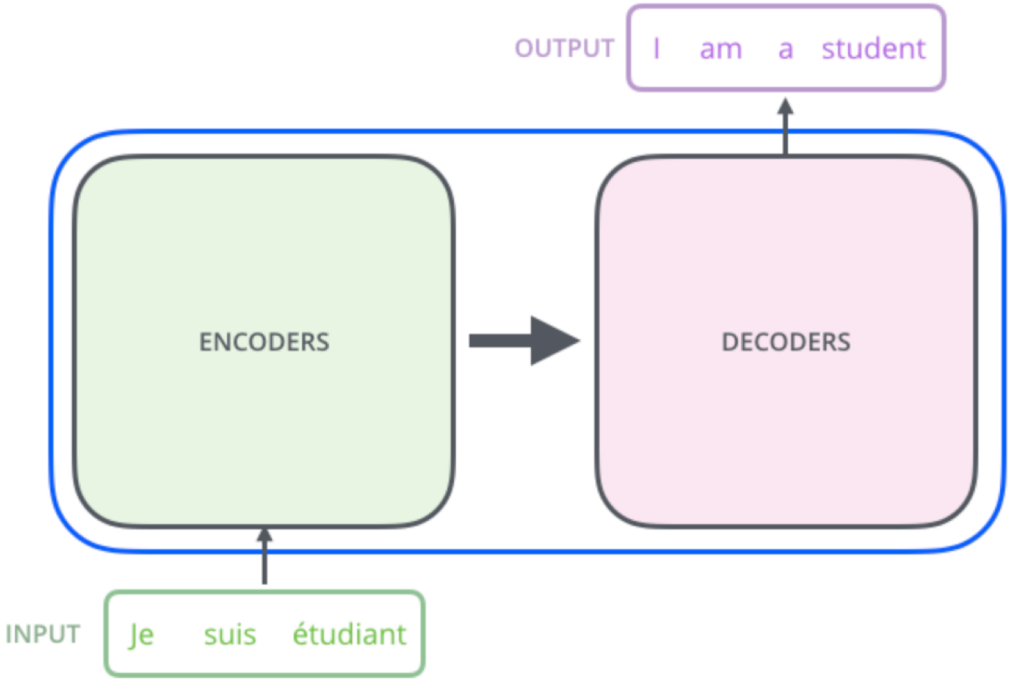


Figure 1: The Transformer - model architecture.



https://jalammar.github.io/illustrated-transformer/

Stacked Encoder & Decoder

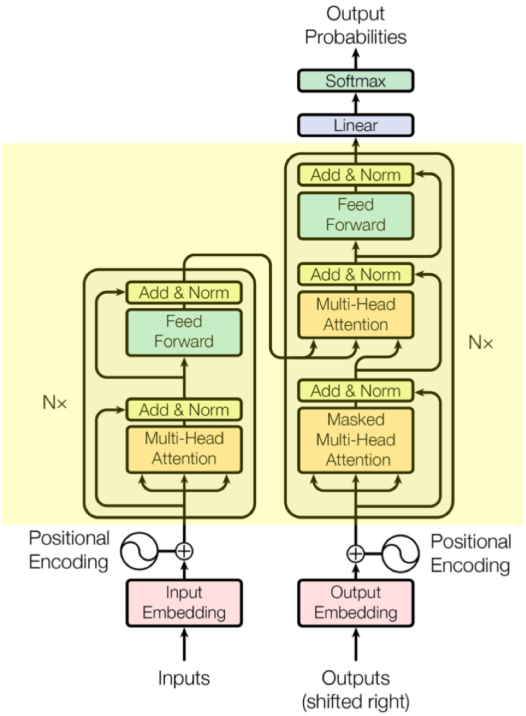
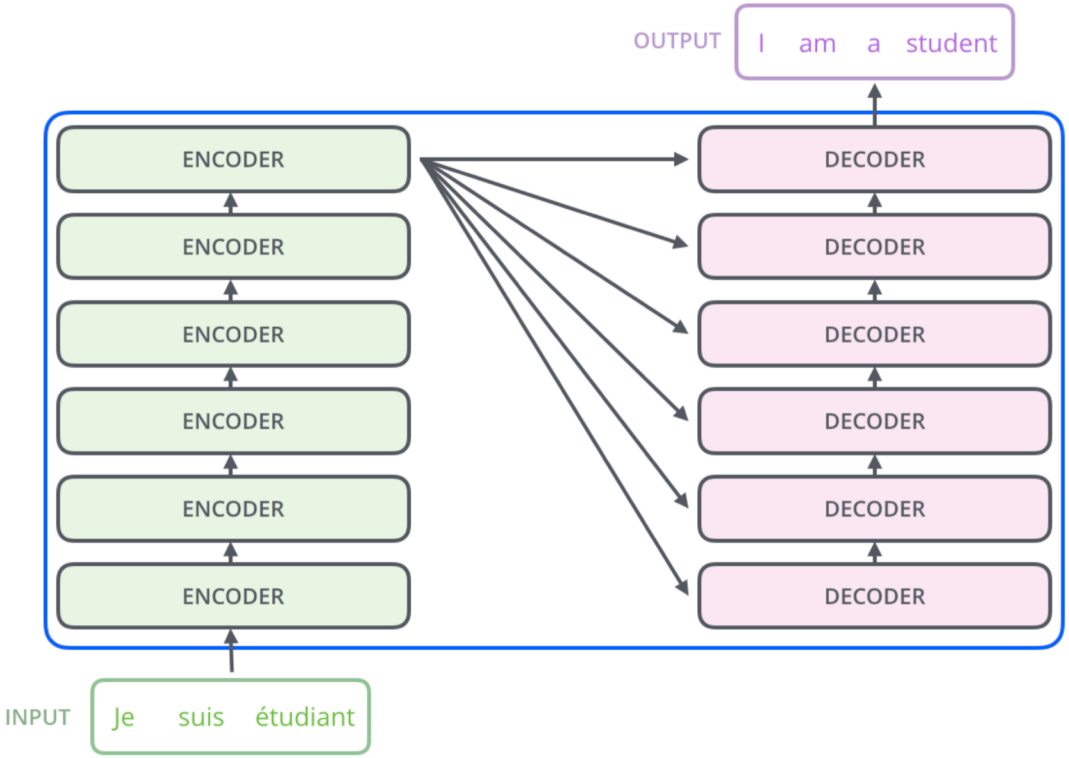


Figure 1: The Transformer - model architecture.



https://jalammar.github.io/illustrated-transformer/

Structure of a single Encoder unit

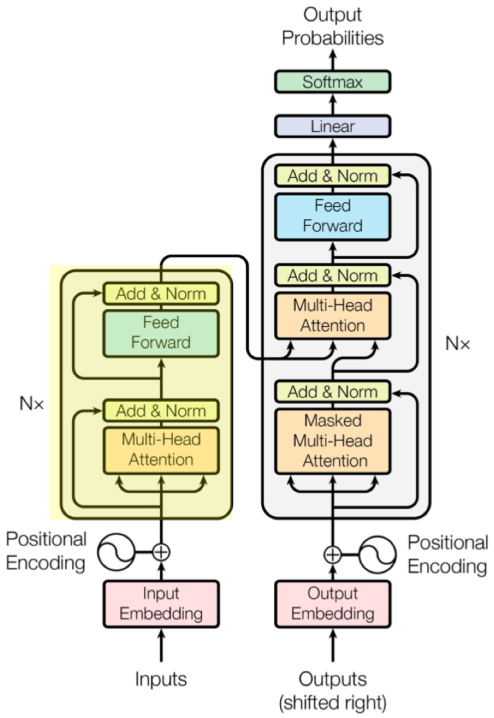
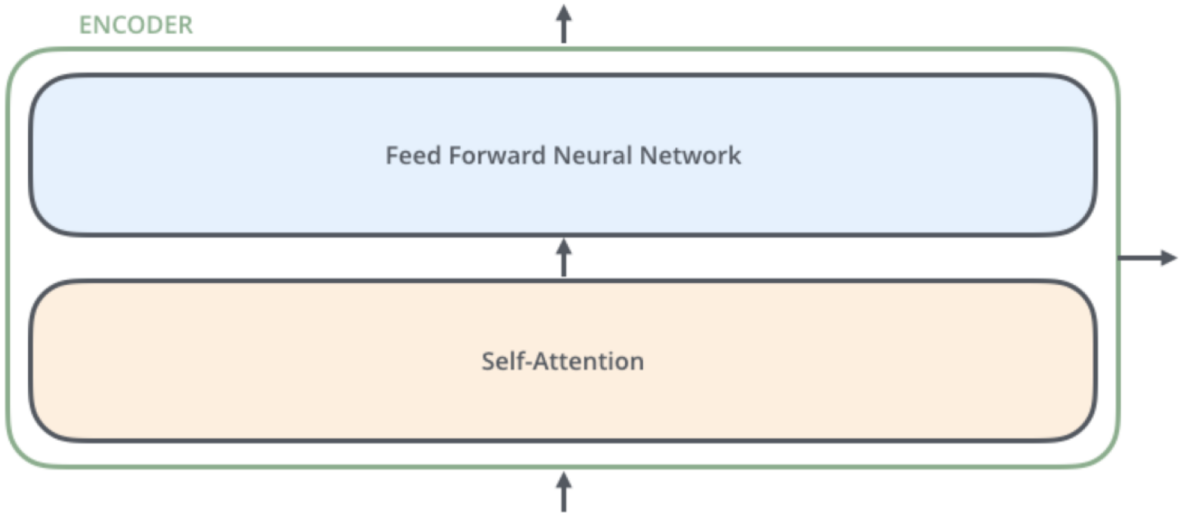


Figure 1: The Transformer - model architecture.



https://jalammar.github.io/illustrated-transformer/

Encoder-Decoder link

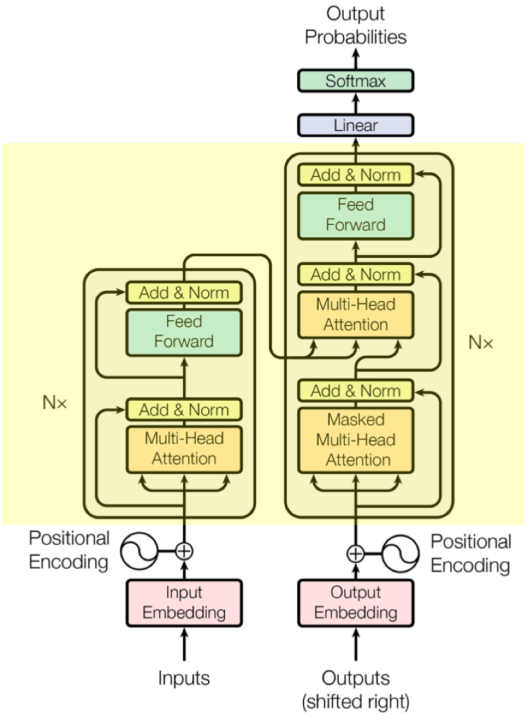
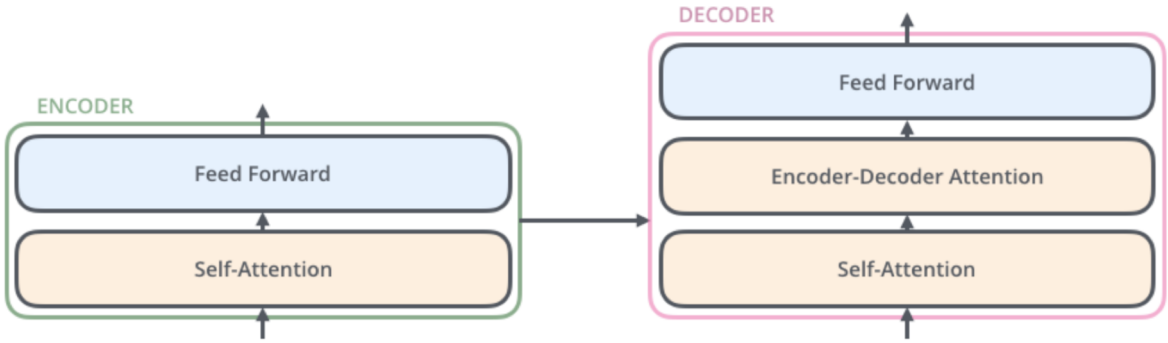


Figure 1: The Transformer - model architecture.

Self-attention in the Encoder & Decoder block,
Then Encoder-Decoder Attention



https://jalammar.github.io/illustrated-transformer/

Self-Attention block

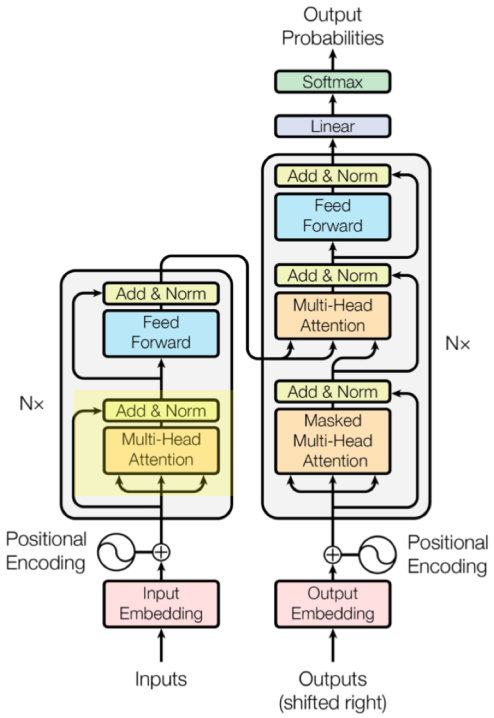
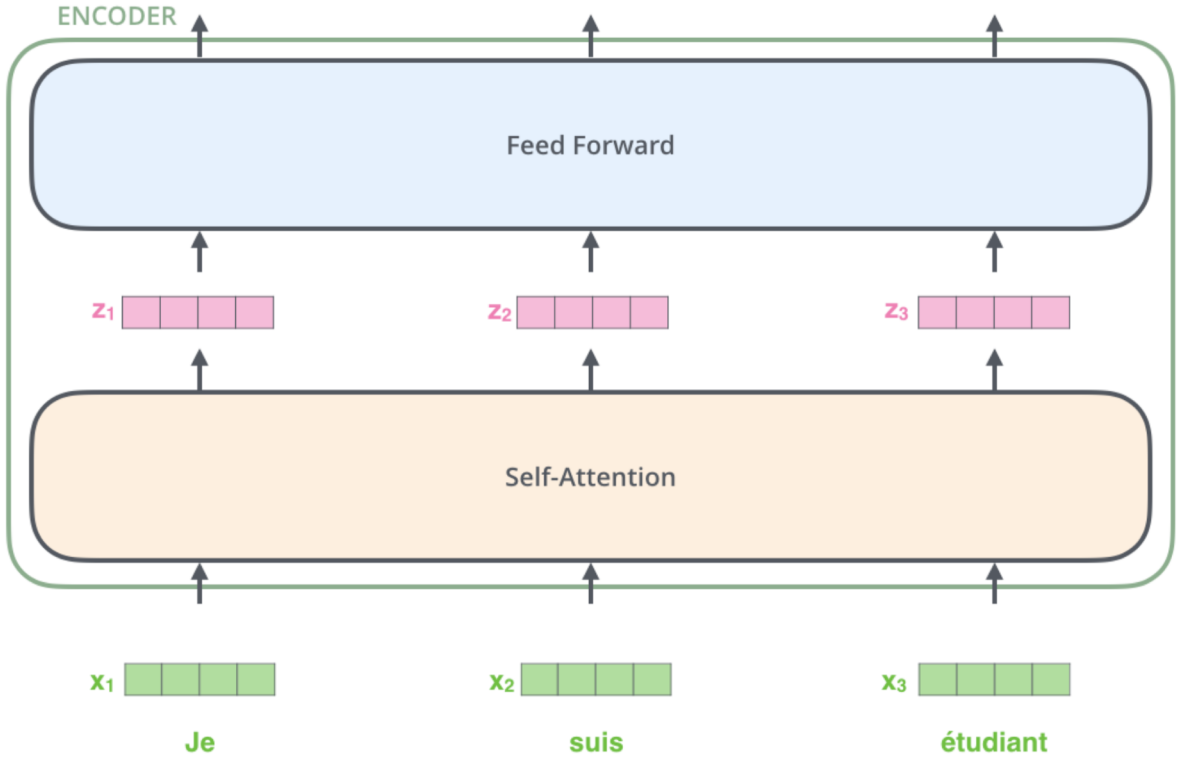


Figure 1: The Transformer - model architecture.

Converting input dimension to embedding dimensions



https://jalammar.github.io/illustrated-transformer/

Self-Attention then Feed-Forward block

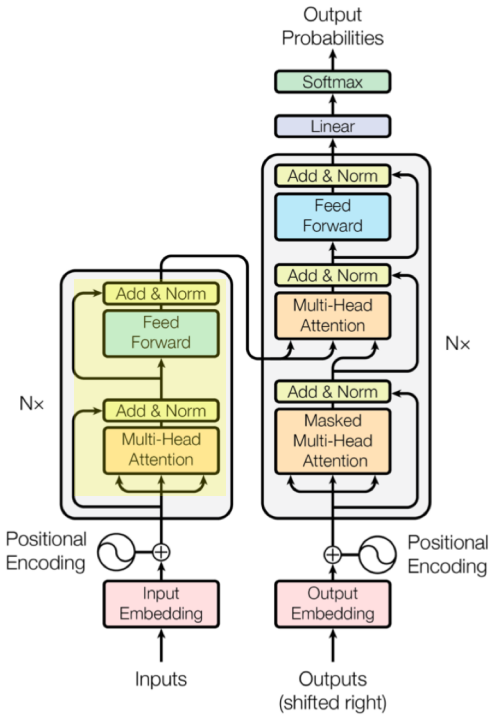
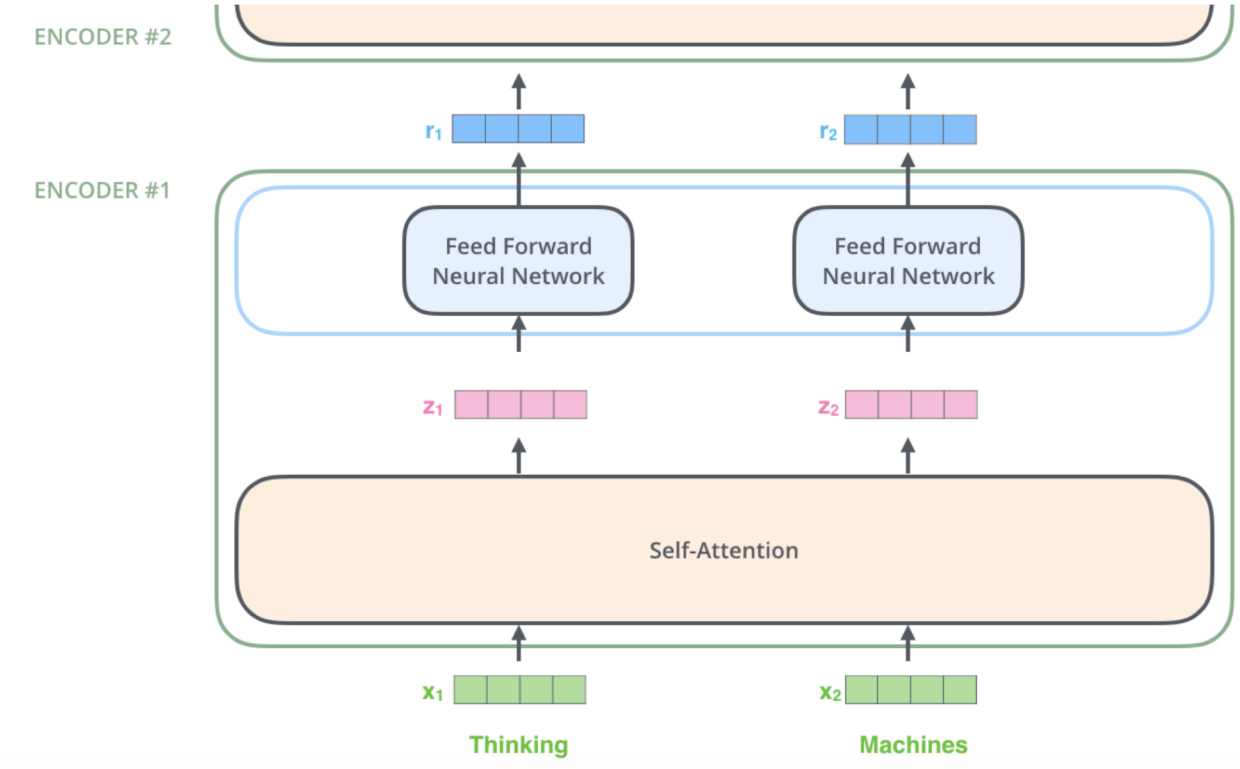


Figure 1: The Transformer - model architecture.

Fully Connected NN to represent vector in a new dimension

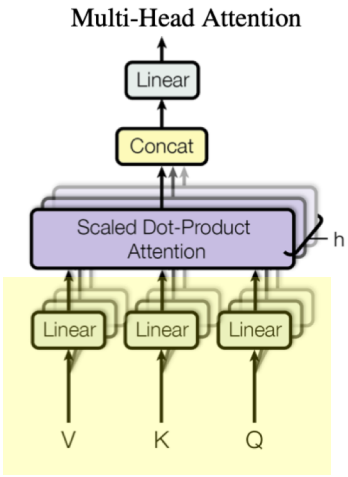


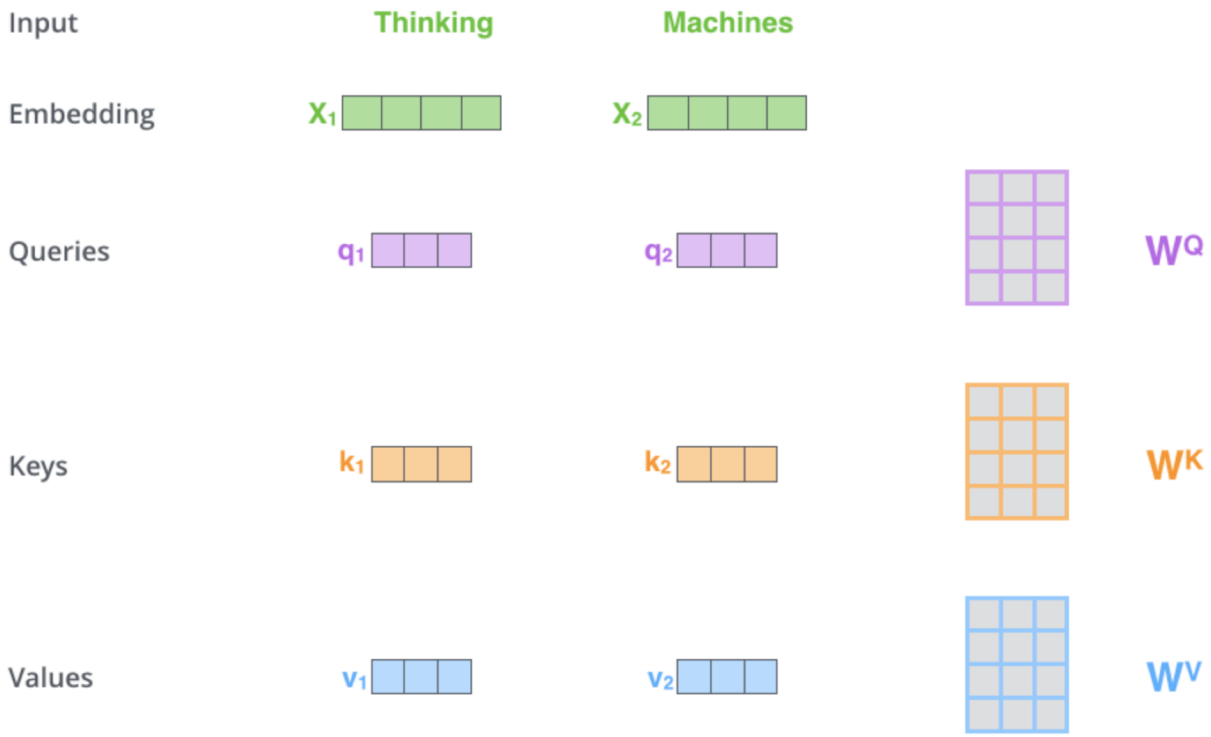
https://jalammar.github.io/illustrated-transformer/

Let’s zoom inside a Self-attention block

Generate Embedding Space

Convert Q, K, V to embedding space dimensions





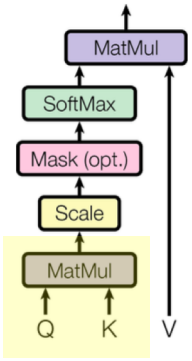
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

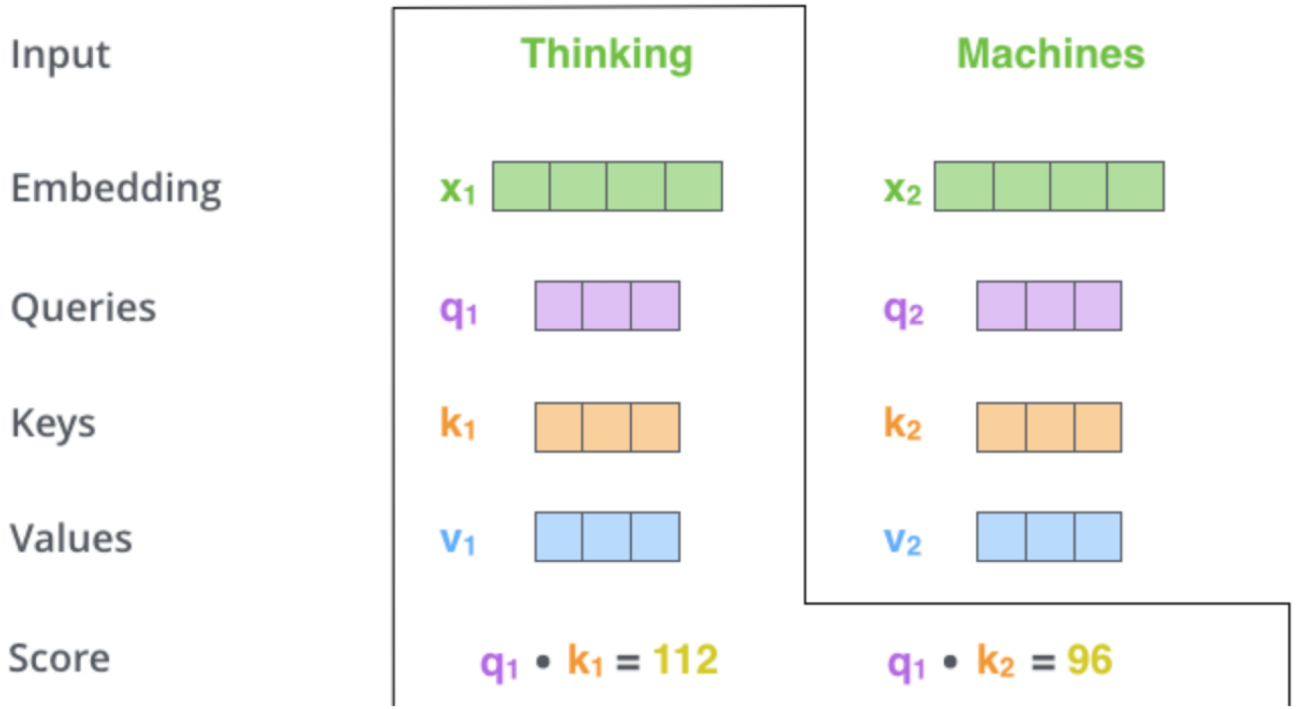
https://jalammar.github.io/illustrated-transformer/

Dot Product between Q and K

Dot product to measure similarity between embedding vectors of Q and K

Scaled Dot-Product Attention





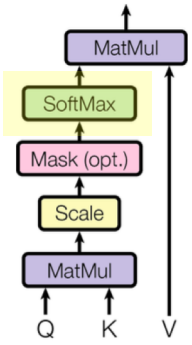
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

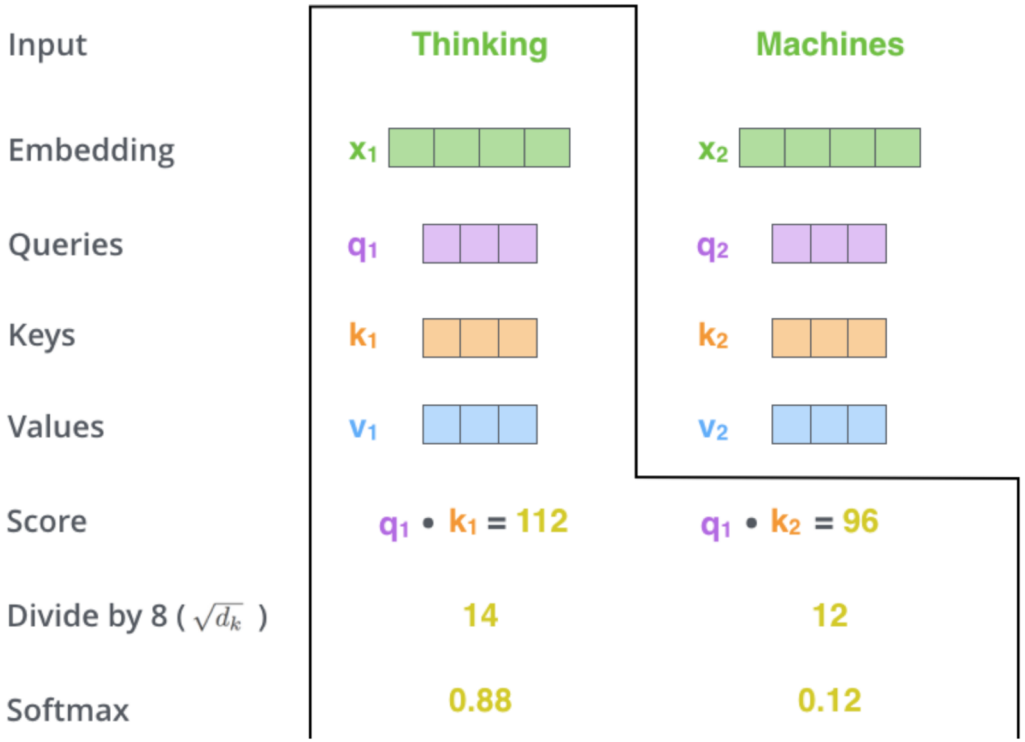
https://jalammar.github.io/illustrated-transformer/

Softmax the dot product

Softmax to maintain overall magnitude scale of value vectors (softmax sums to 1)

Scaled Dot-Product Attention





Division to smooth out softmax

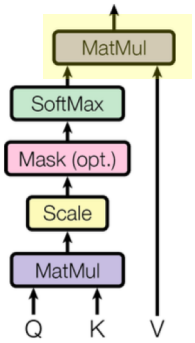
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

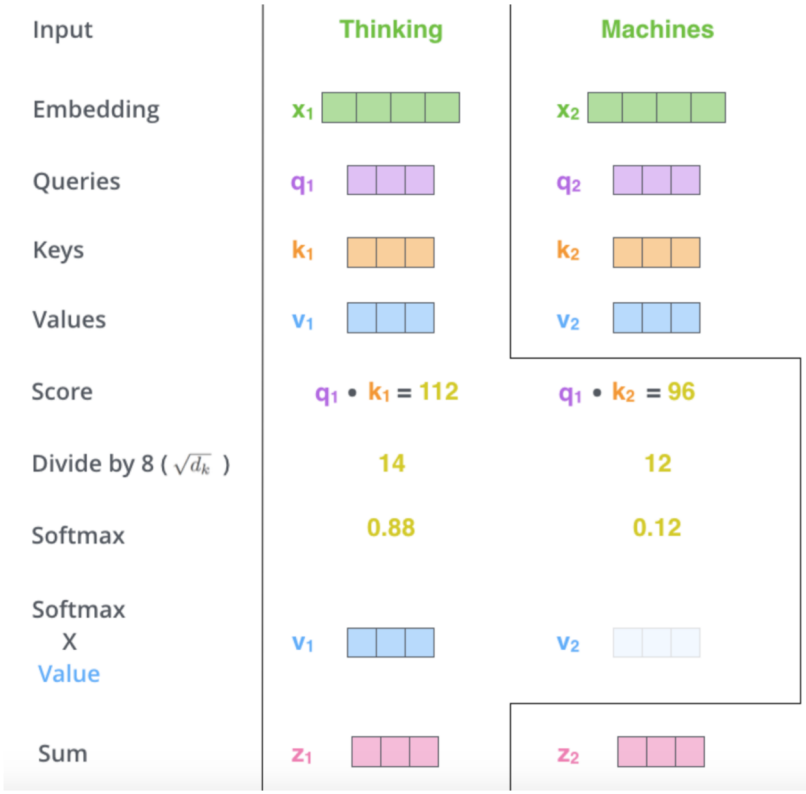
https://jalammar.github.io/illustrated-transformer/

Sum up the values weighted by softmax

Softmax to maintain overall magnitude scale of value vectors (softmax sums to 1)

Scaled Dot-Product Attention

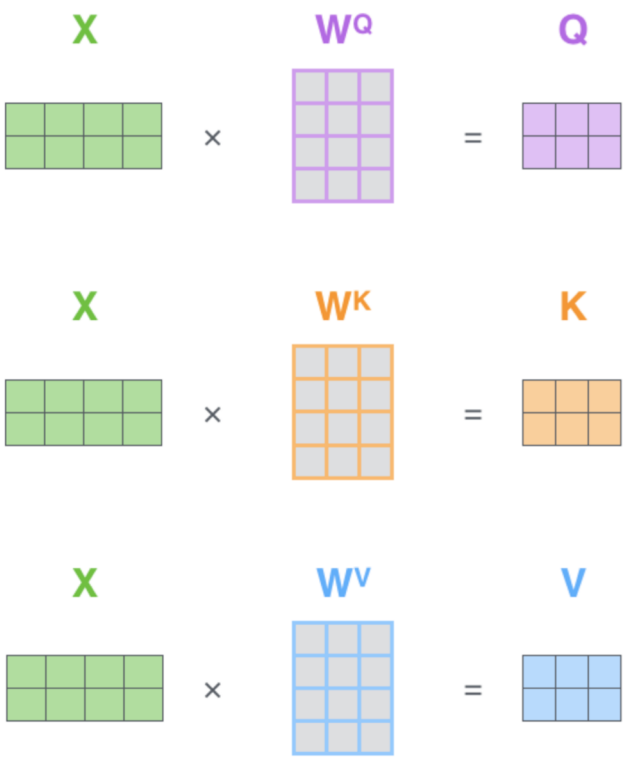


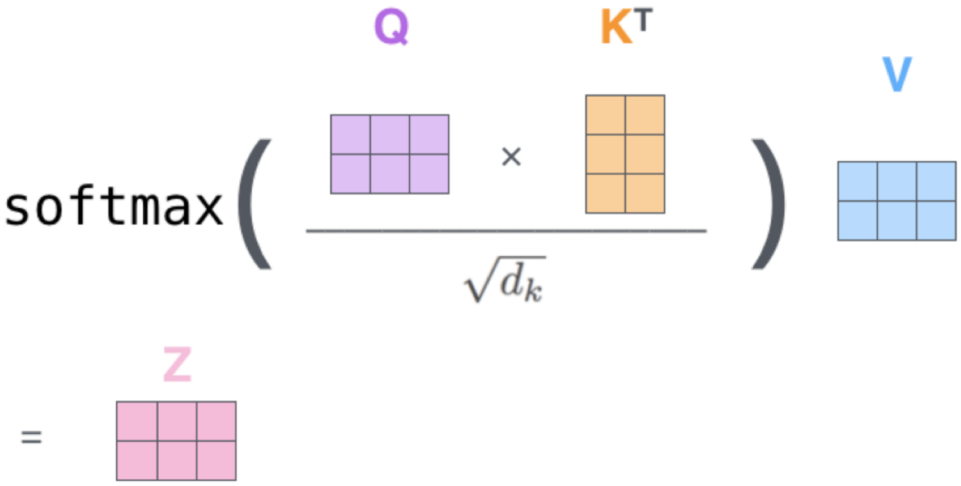


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

https://jalammar.github.io/illustrated-transformer/

Overall: Self-attention in matrix form





https://jalammar.github.io/illustrated-transformer/

Multi-headed Attention

Have a group of people.

Each person pays attention to something.

Combine the people’s wisdom together at the end.

Multi-headed attention

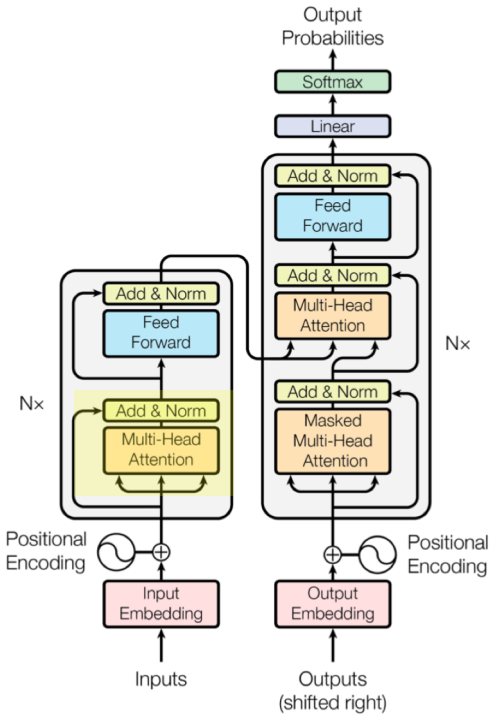
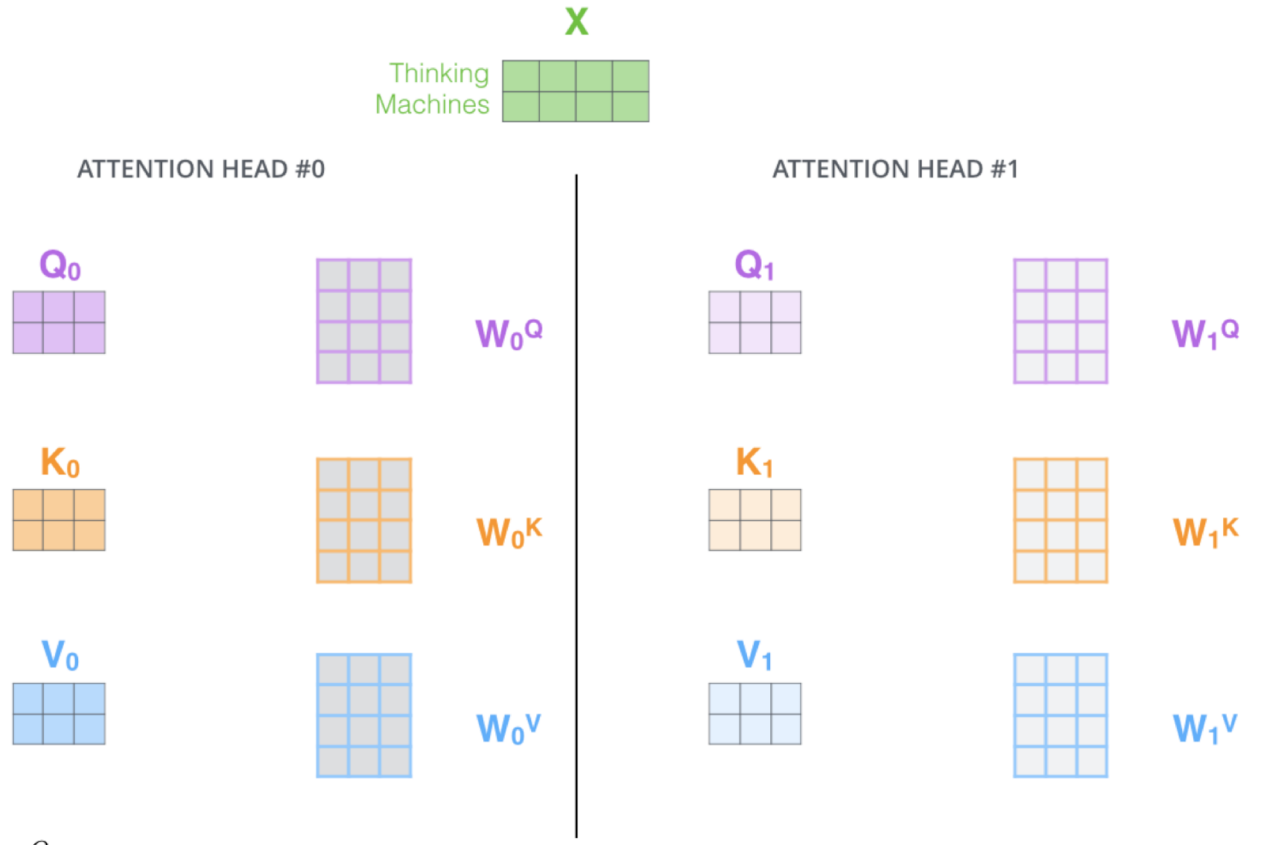


Figure 1: The Transformer - model architecture.



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

https://jalammar.github.io/illustrated-transformer/

Generate multiple heads

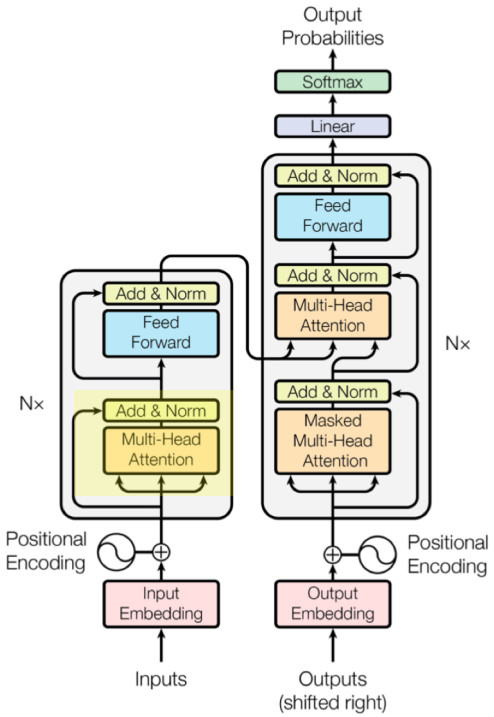
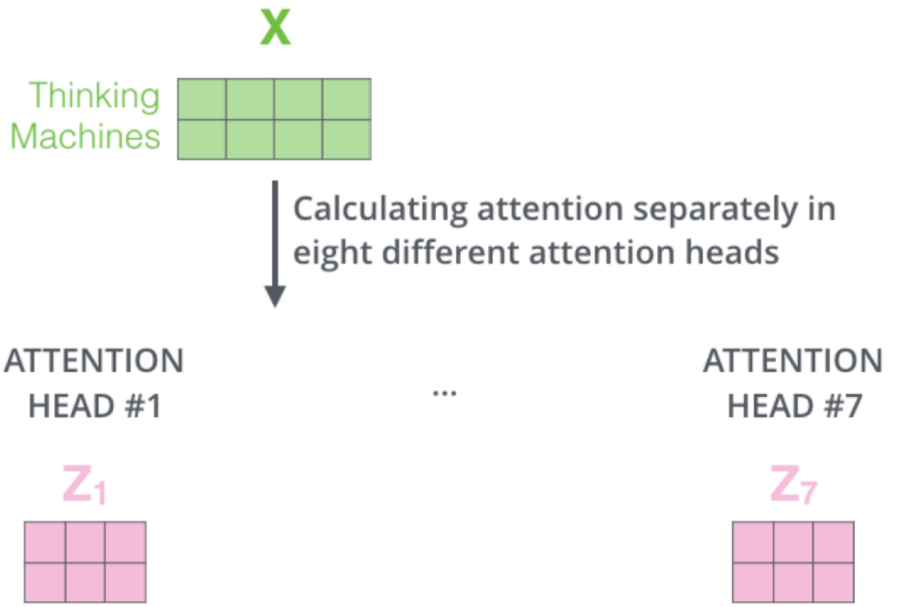


Figure 1: The Transformer - model architecture.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



https://jalammar.github.io/illustrated-transformer/

Concatenate multiple attention heads

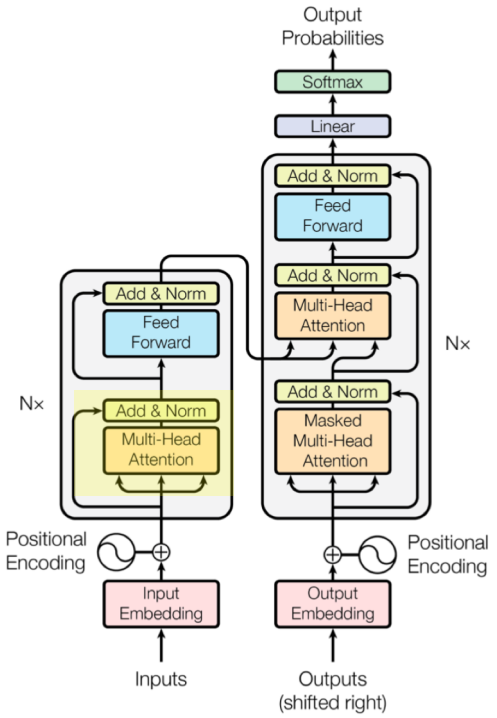
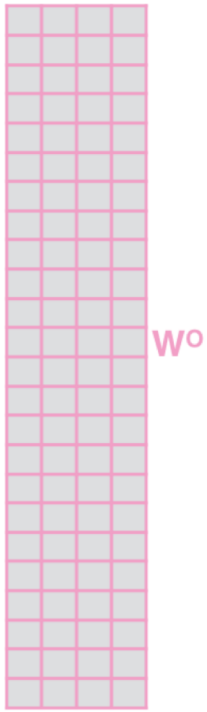


Figure 1: The Transformer - model architecture.

1) Concatenate all the attention heads

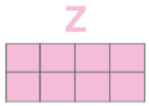
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

https://jalammar.github.io/illustrated-transformer/

Sum up the various attention heads

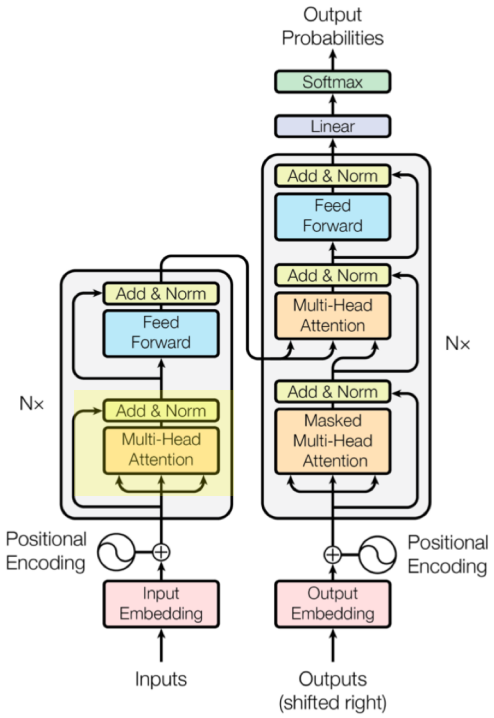
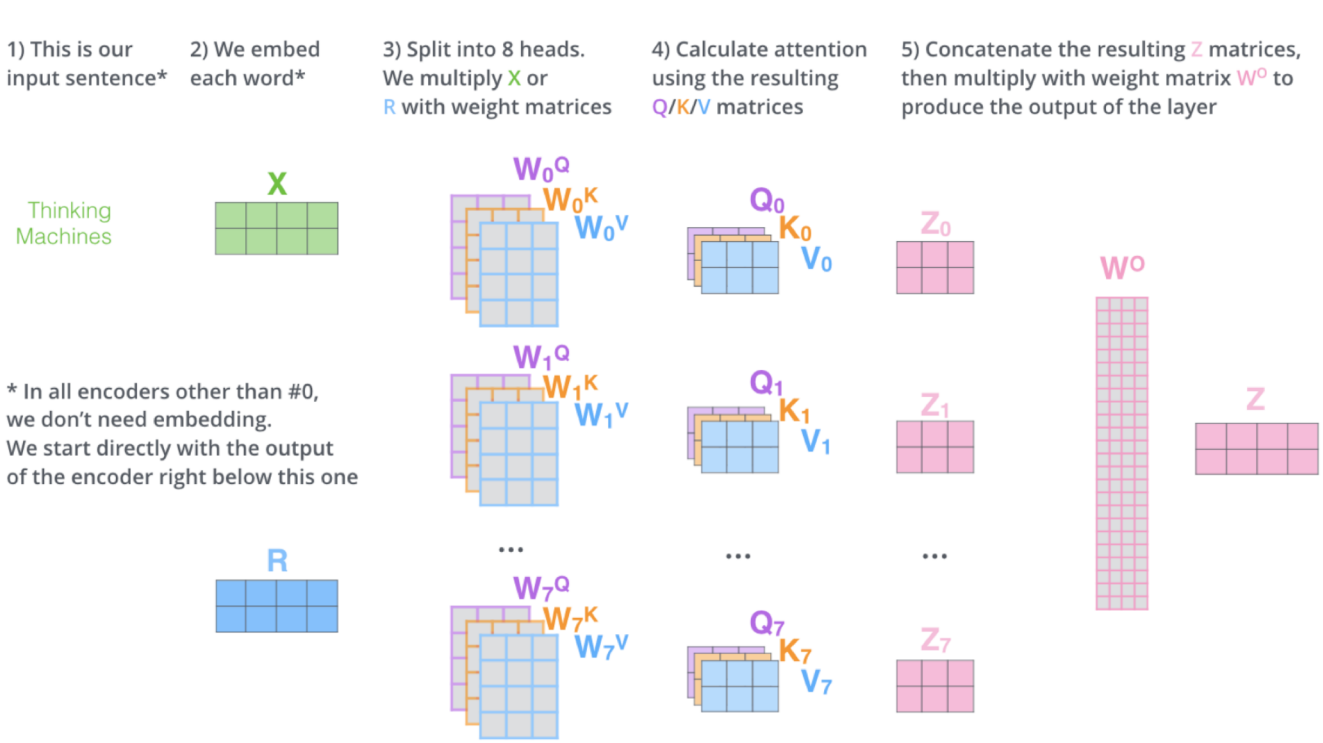


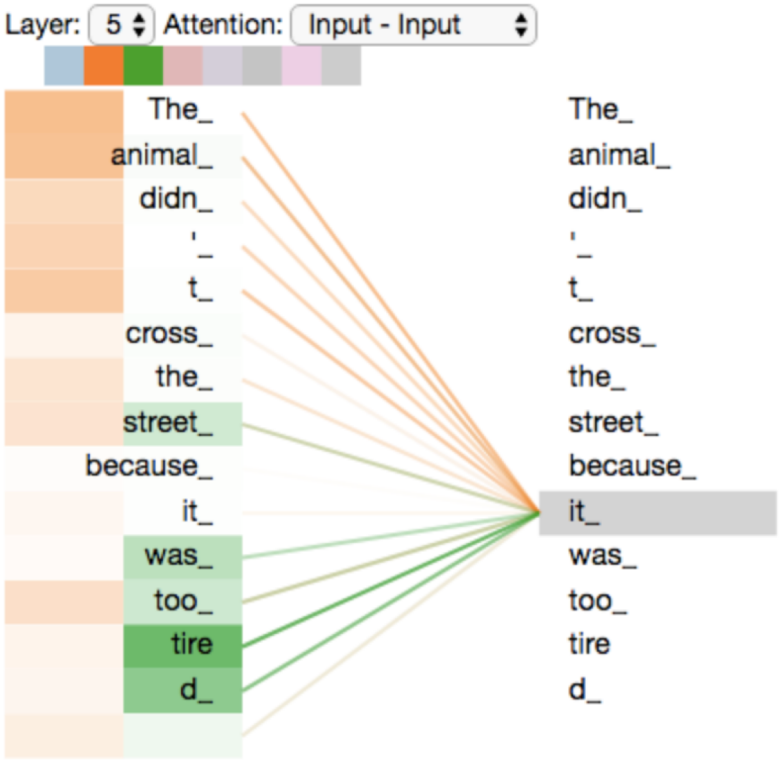
Figure 1: The Transformer - model architecture.



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

https://jalammar.github.io/illustrated-transformer/

Visualizing 2 attention heads



https://jalammar.github.io/illustrated-transformer/

Skip-connections / LayerNorm

Skip-connection

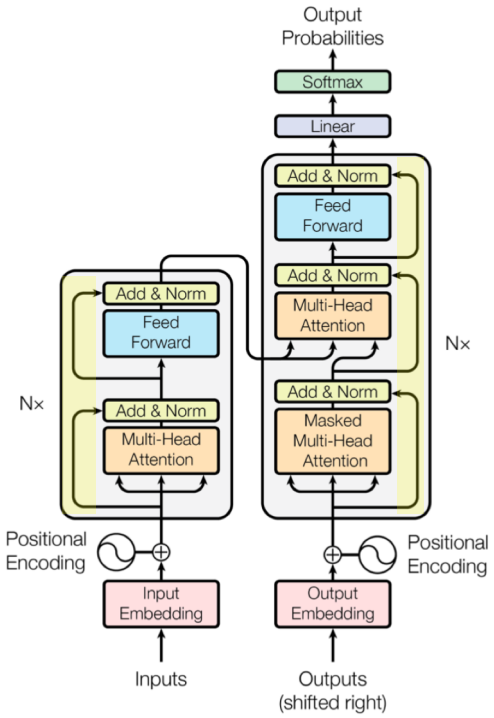
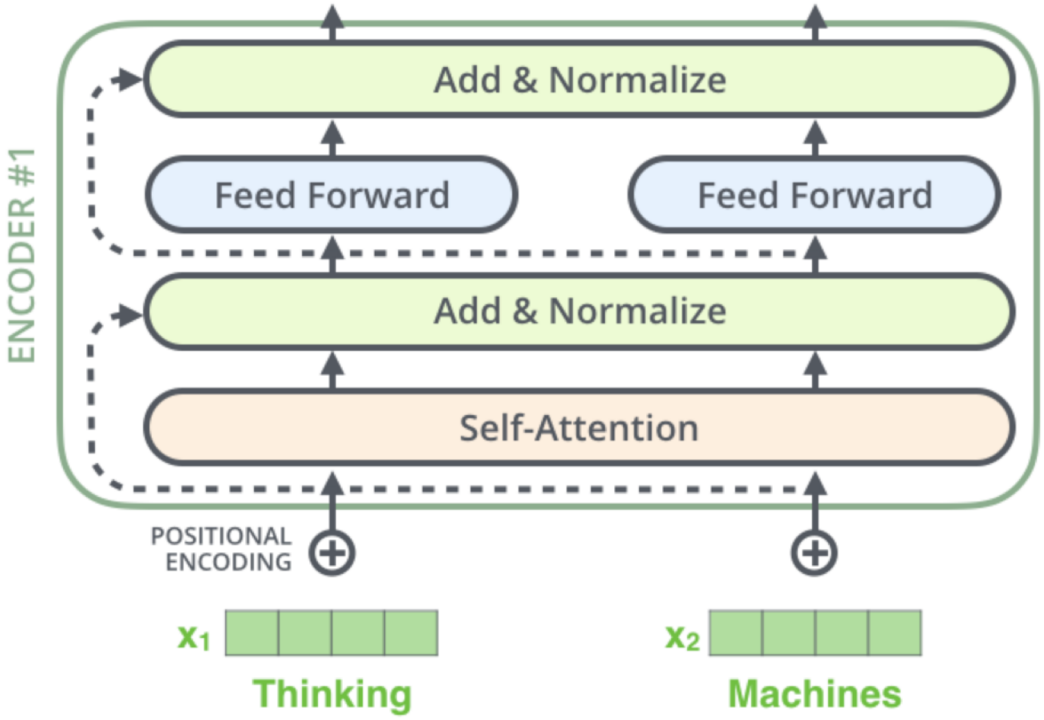


Figure 1: The Transformer - model architecture.

Skip connection helps the network choose which sections are important



https://jalammar.github.io/illustrated-transformer/

Layer Normalization

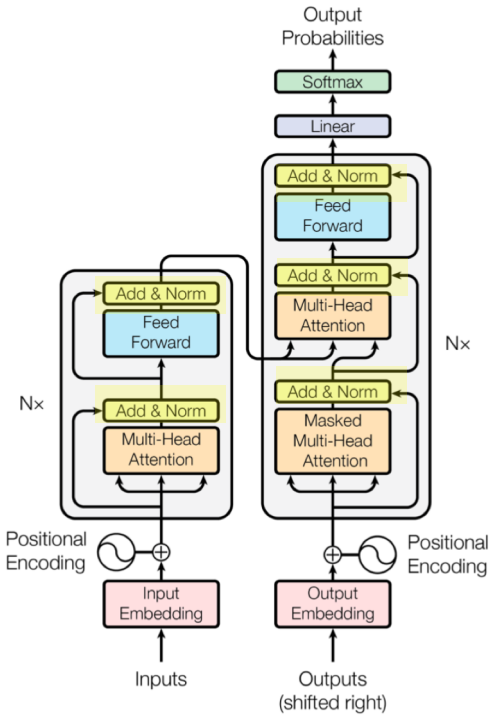
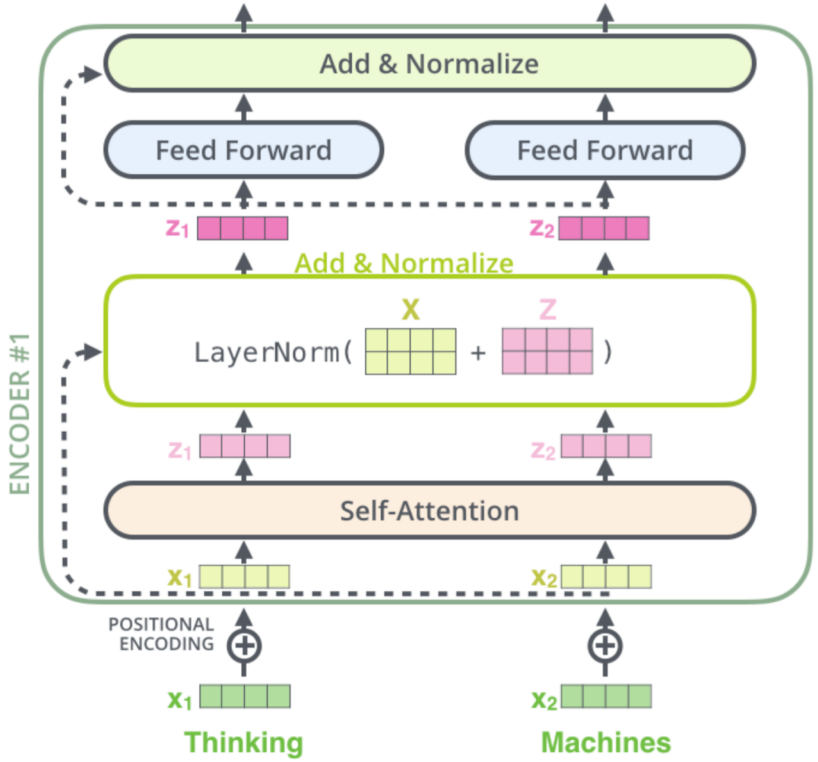


Figure 1: The Transformer - model architecture.

Layer normalization helps prevent gradient vanishing/exploding

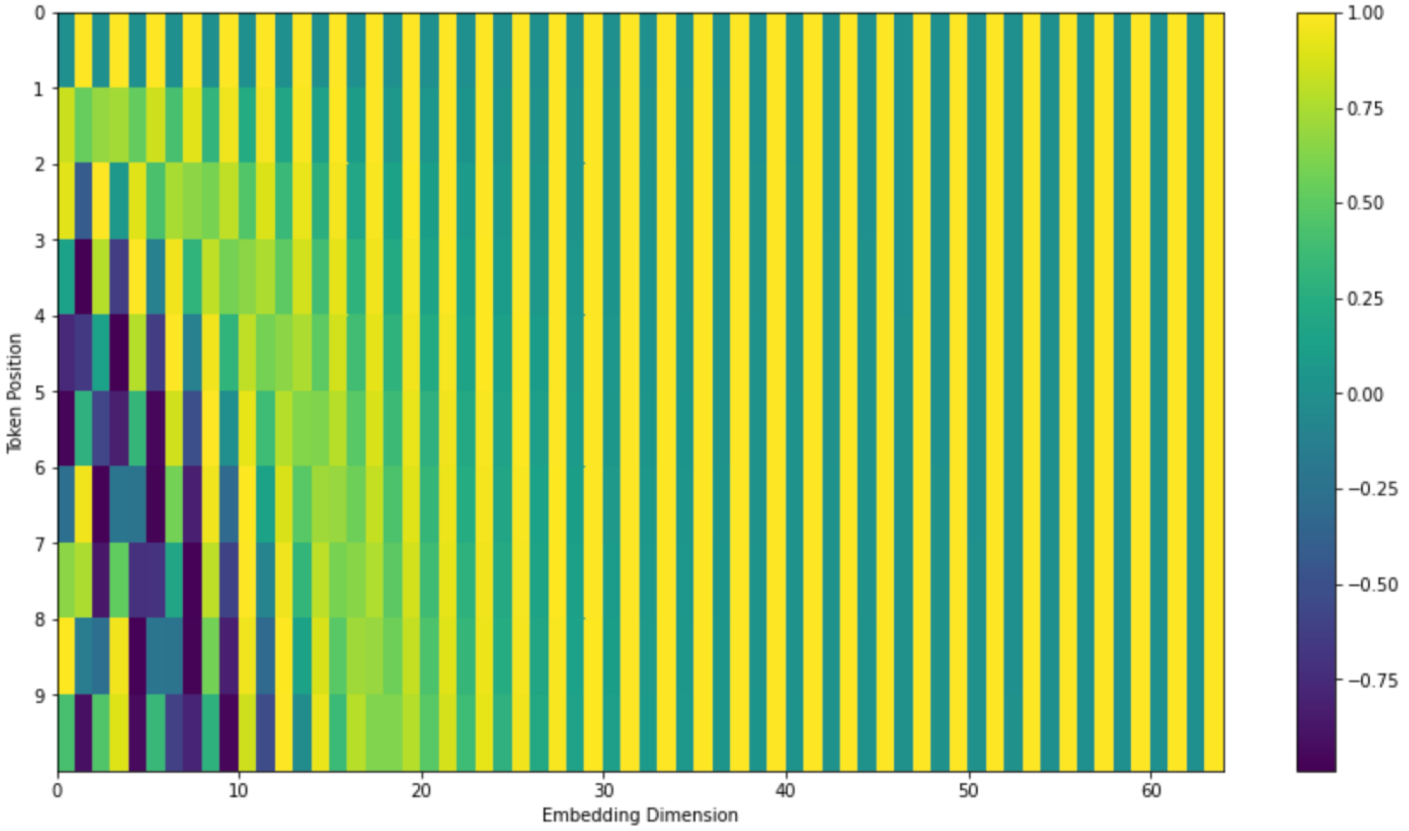


https://jalammar.github.io/illustrated-transformer/

Positional Encoding

Adding sequence information to a Transformer

Positional Encoding



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos: position of word/subword

i: dimension *i* in embedding space

d_{model}: number of dimensions in embedding space

https://jalammar.github.io/illustrated-transformer/

Can we do differently?

- Have a separate vector for word/subword embeddings concatenated to the original input embedding

- Use a learned positional embedding vector
  - Around the same performance
  - Limitation is cannot extrapolate to word/subword lengths beyond the learned positional embeddings

- Relative positional embeddings

BERT

Bidirectional Encoder Representations from Transformers

Masked Language Modelling (BERT)

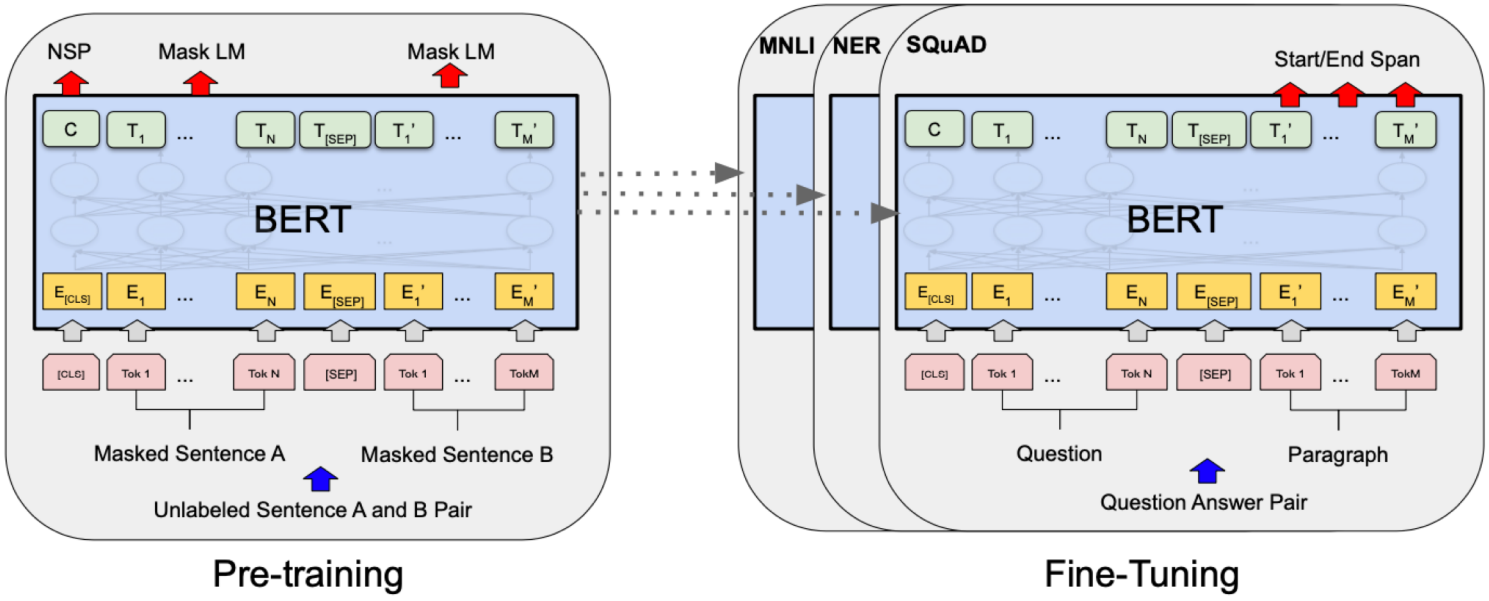
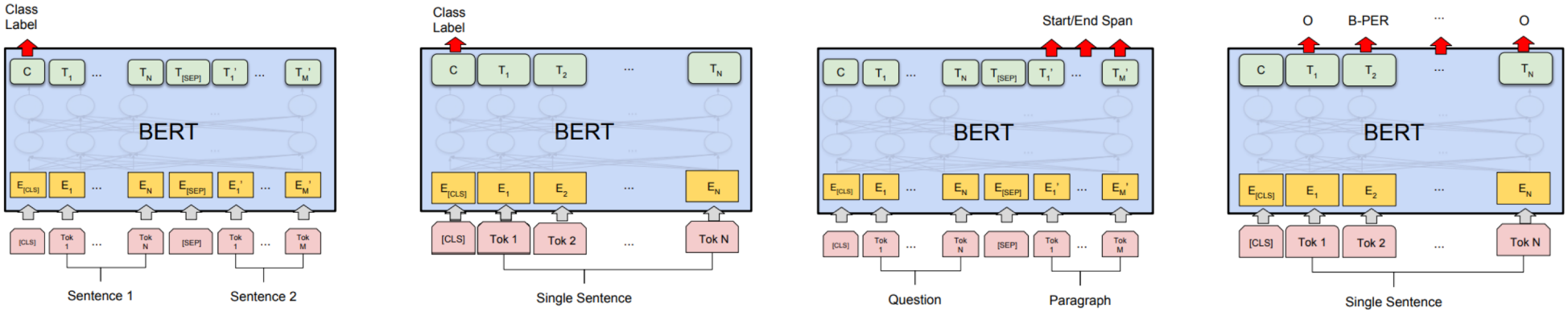


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Taken from: BERT. Delvin et al. (2019)

Fine-tuning BERT



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

- Fine-tune by adding in new task-specific layer to output
- [CLS] for start of sentence
- [SEP] for separation between two sentences

Taken from: BERT. Delvin et al. (2019)

BERT Advantage

- Can pre-train on an extensive unlabeled corpus
  - 15% random masking of words, model predicts those words
    - The <mask> ate fish => predict: cat
    - Categorical Cross-entropy Loss for masked word prediction
  - Given two sentences, model predicts whether next sentence follows first
    - Capture semantic relationship between sentences
    - Binary Cross-entropy Loss for binary value prediction of 0/1 for next sentence
- As long as task can be framed as a single sentence / as an input sentence + output sentence, can fine-tune BERT to do the task
- **Overall philosophy: Train once, fine-tune on many**

Discussion