Transformer

Assoc. Prof. Peerapon Vateekul, Ph.D.
Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
peerapon.v@chula.ac.th
Credits to: TA.Pluem, TA.Knight, and all TA alumni

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

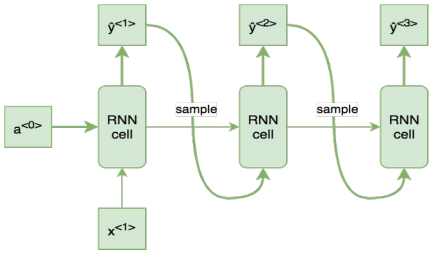
Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

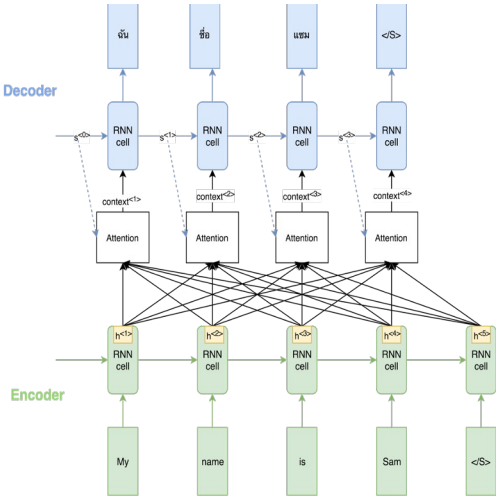
Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

The Transformer

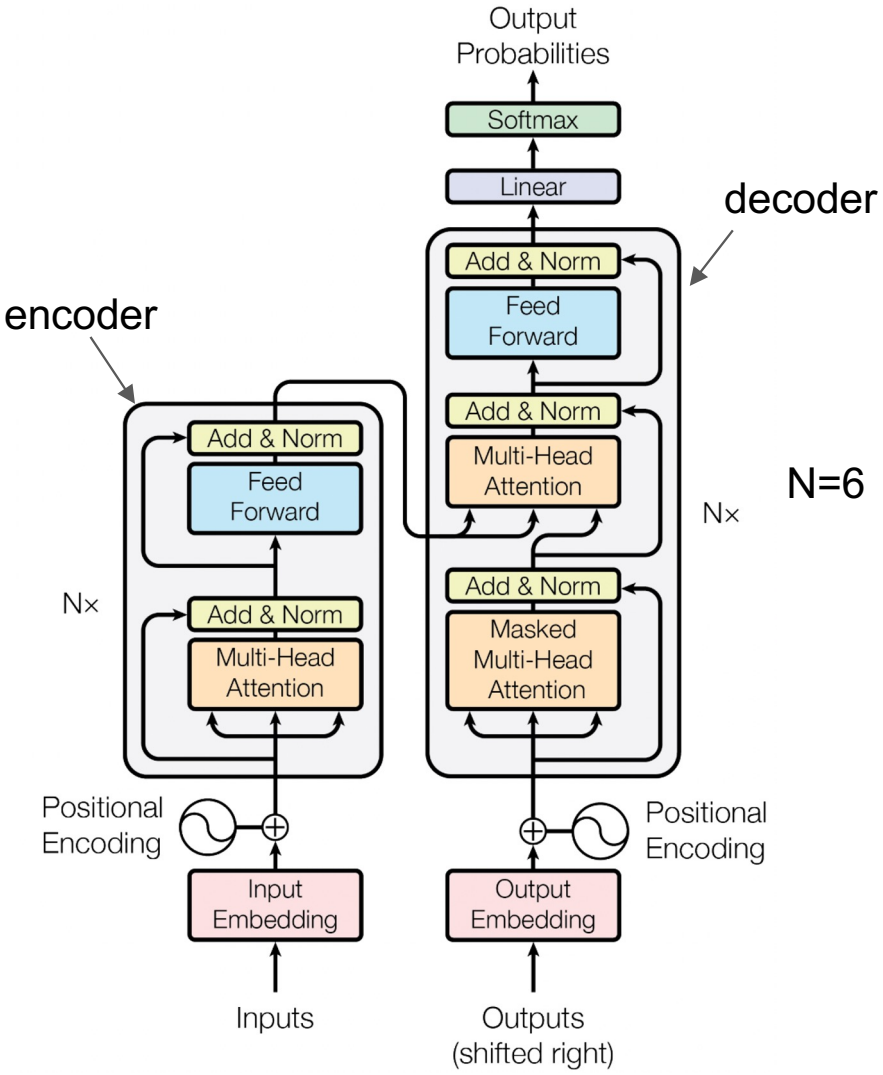




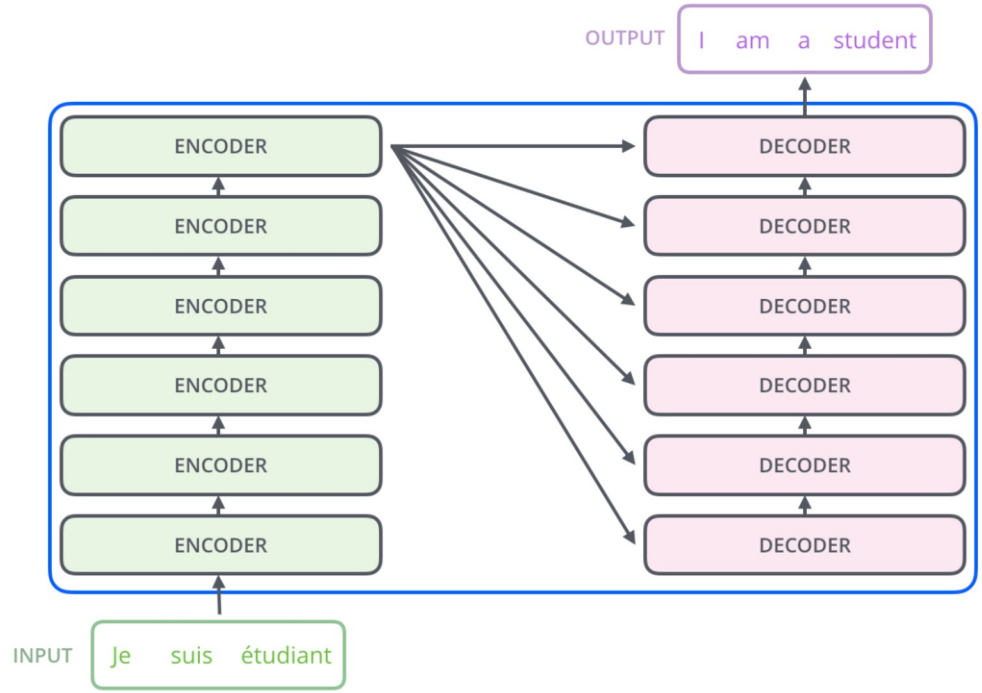
Only rely on the attention mechanism - **No RNN**!

- More parallelizable -> Faster training time
- Better at longer sequence

Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

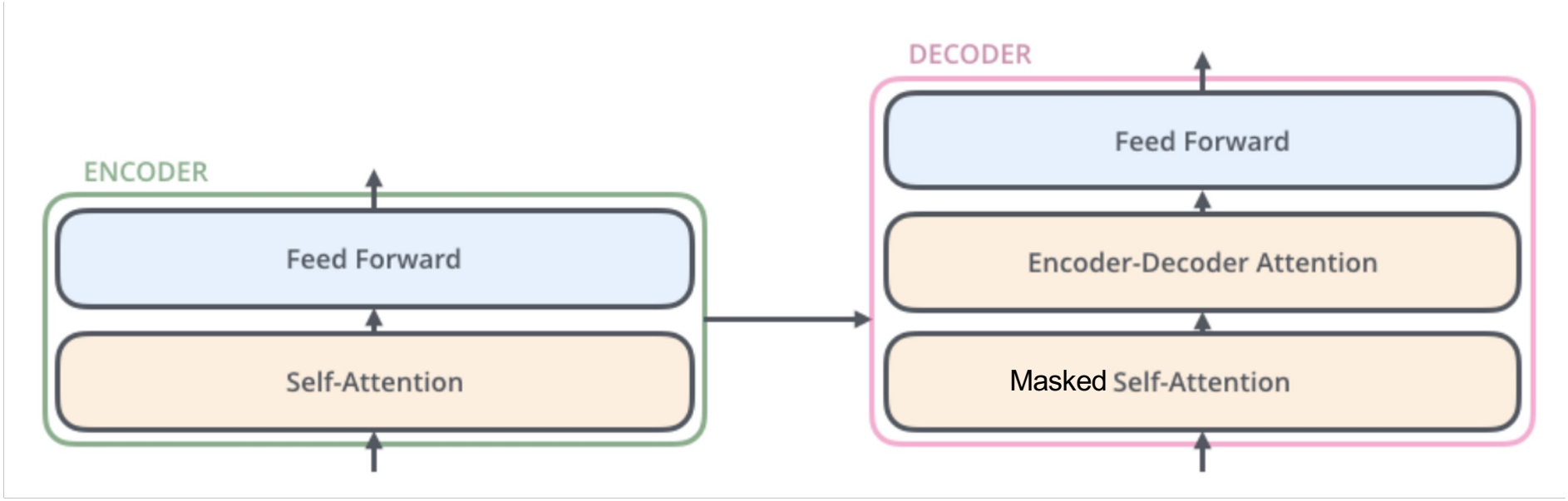


=

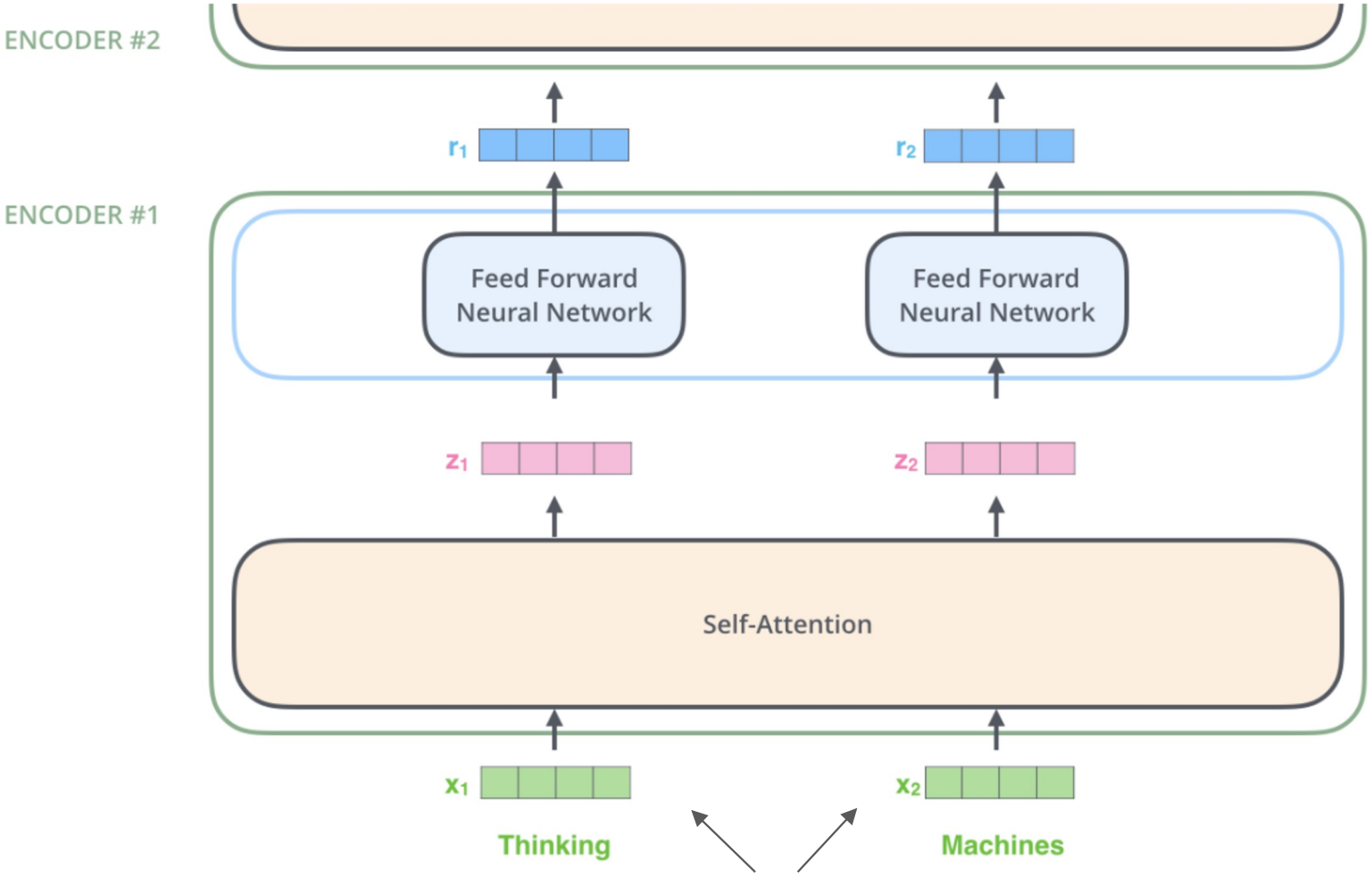


http://jalammar.github.io/illustrated-transformer/

In each layer..



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



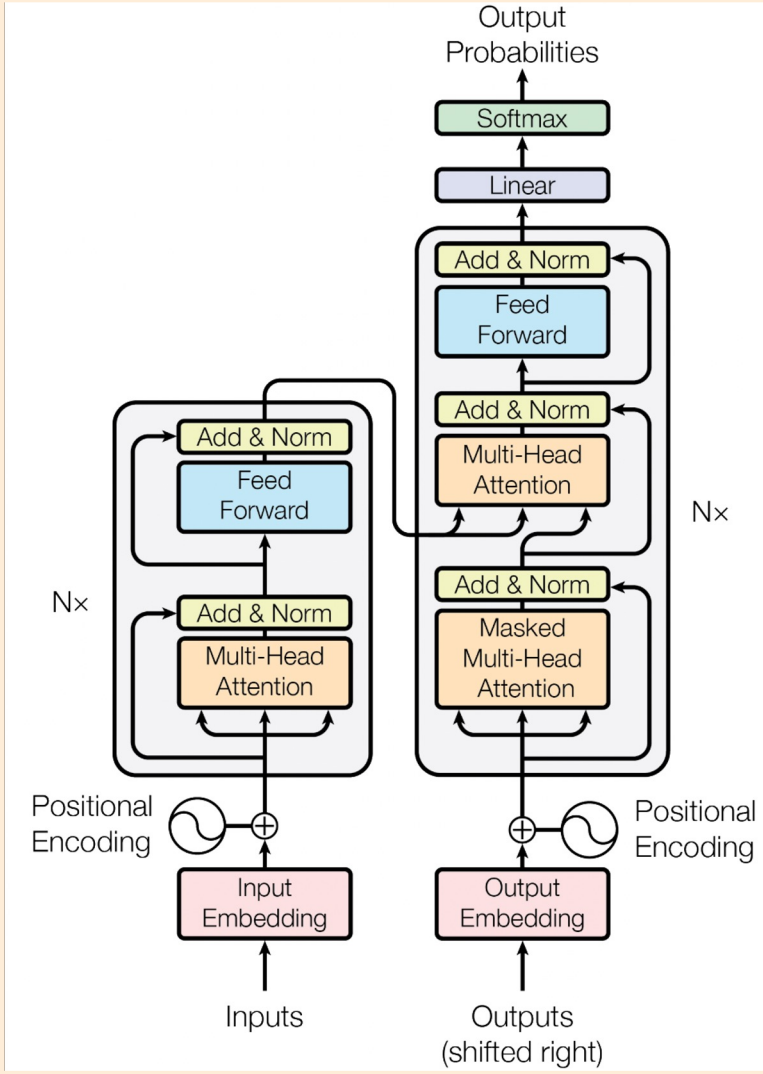
Word Embeddings

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

Scaled Dot-Product Attention

There are many variations used in the Transformer:

1. Self Attention
2. Encoder-decoder Attention
3. Masked Self Attention
4. Multi-headed Self Attention

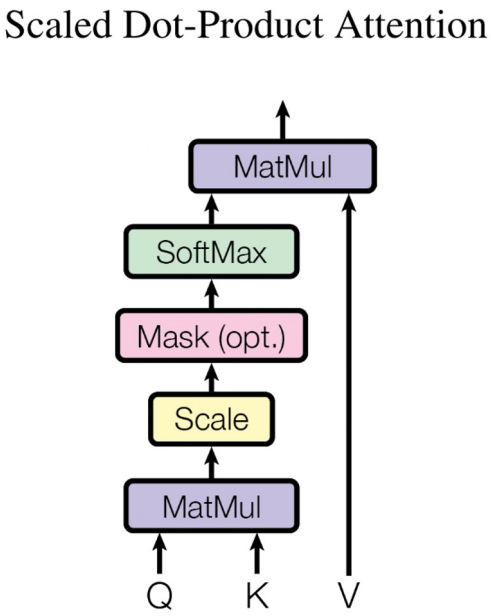


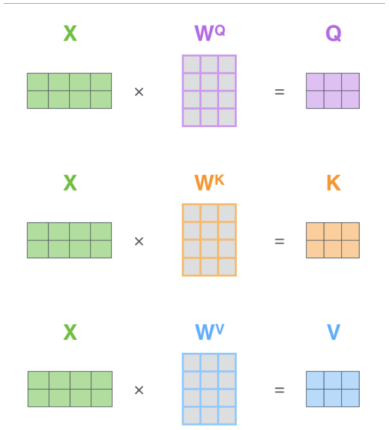
1) Self Attention

Q, K, and V are all from the same input.

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

where the query, keys, values, and output are all vectors and d_k is a number.

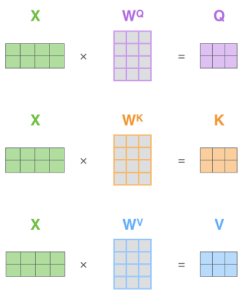


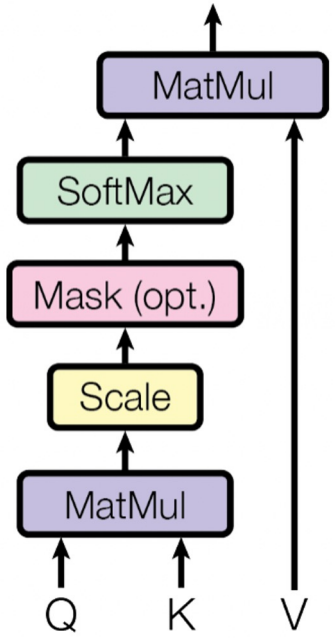


Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2) Encoder-Decoder Attention

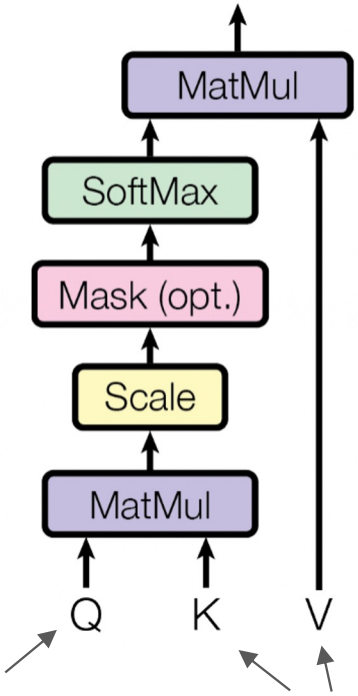
Self Attention

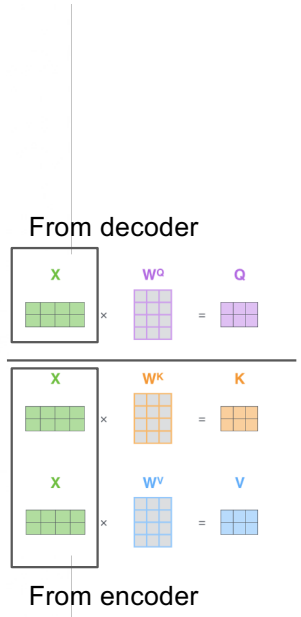




ALL from same input

Encoder-Decoder Attention
(or Cross Attention)



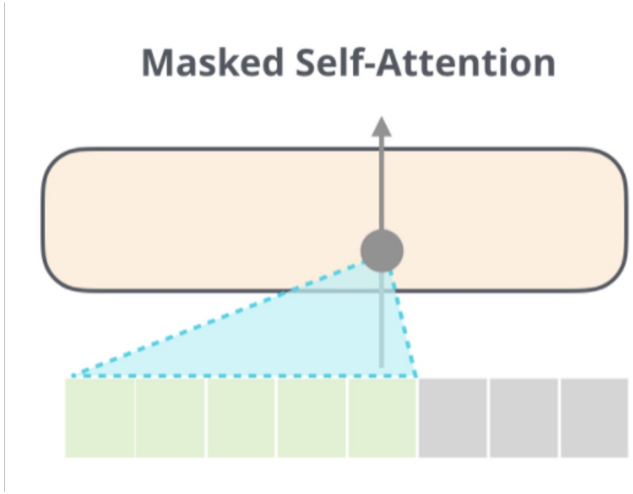


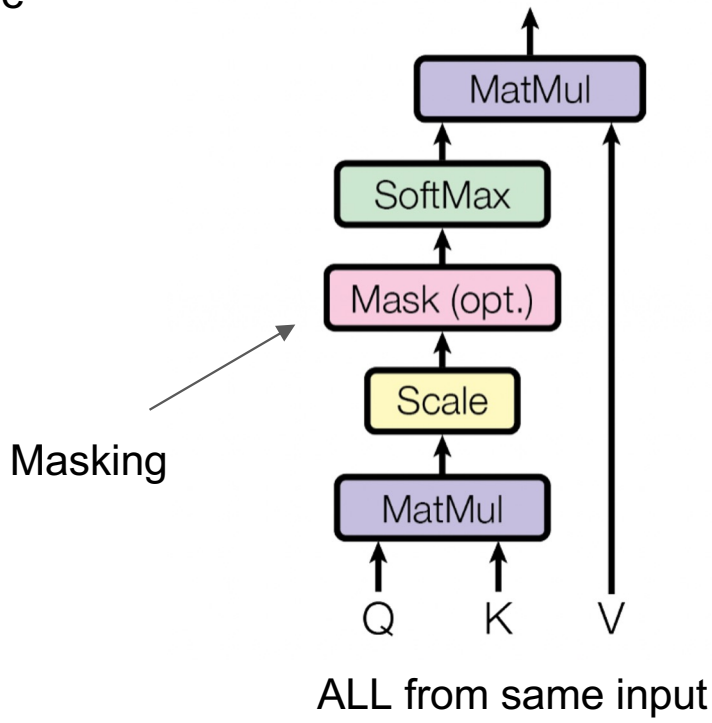
From decoder From encoder

3) Masked Self Attention

Prevent the model from **seeing into the future** to preserve the autoregressive property.

Used in *decoder* only (so the model can't see the answer).

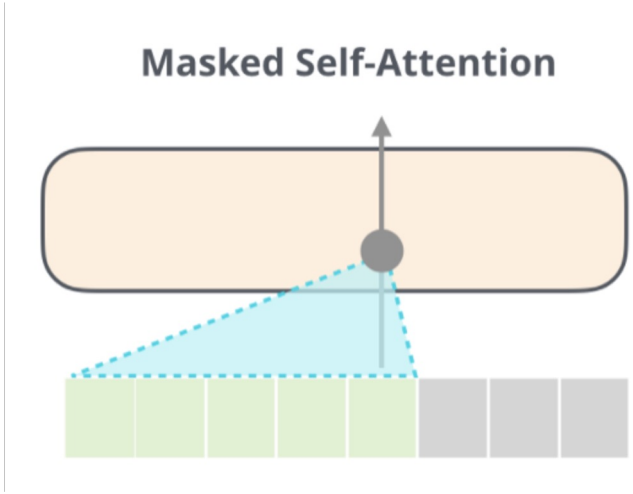


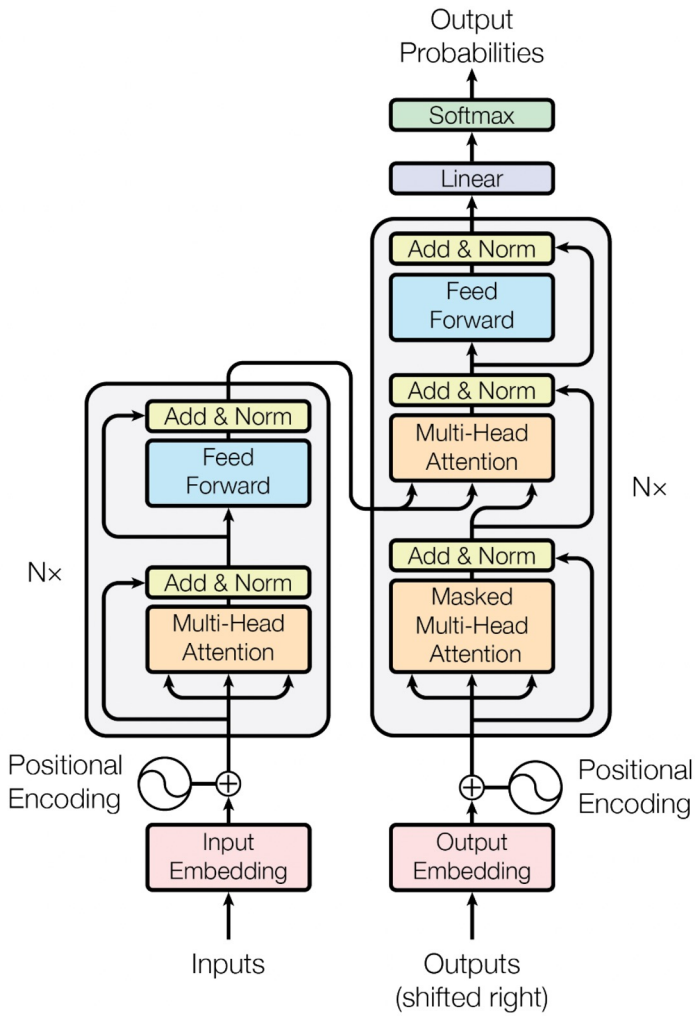


Masked Self Attention

Prevent the model from **seeing into the future** to preserve the autoregressive property.

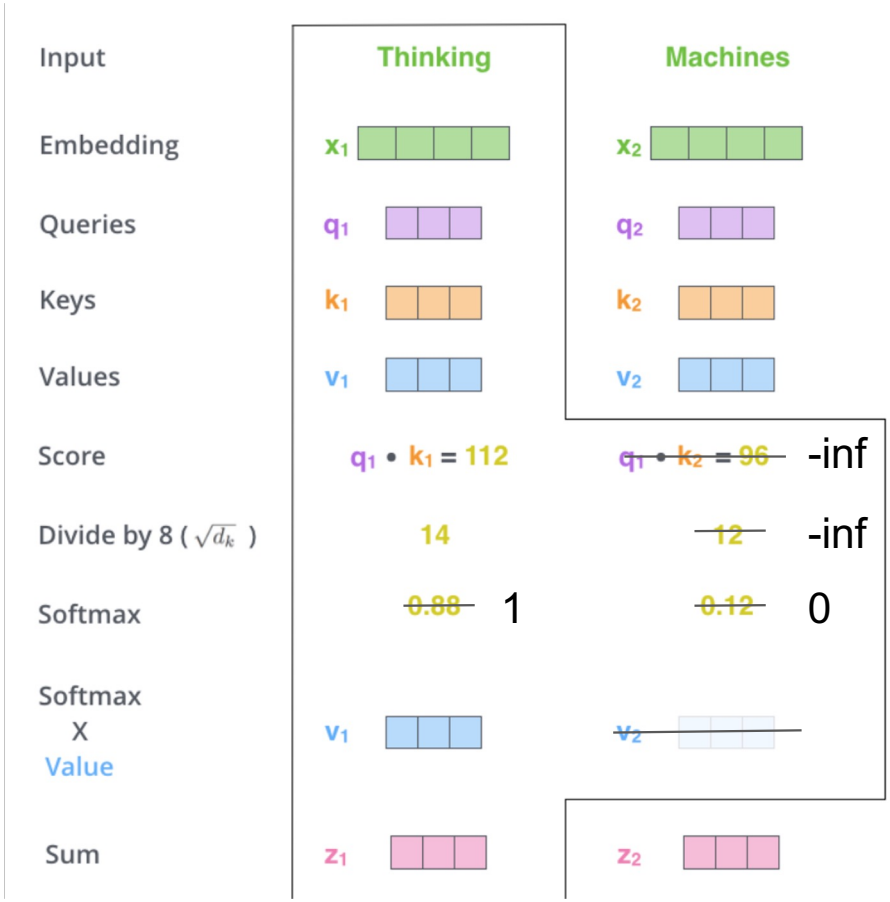
Used in *decoder* only (so the model can't see the answer).





I like dogs <eos> <bos> ฉัน ชอบ ~~สุนัข~~

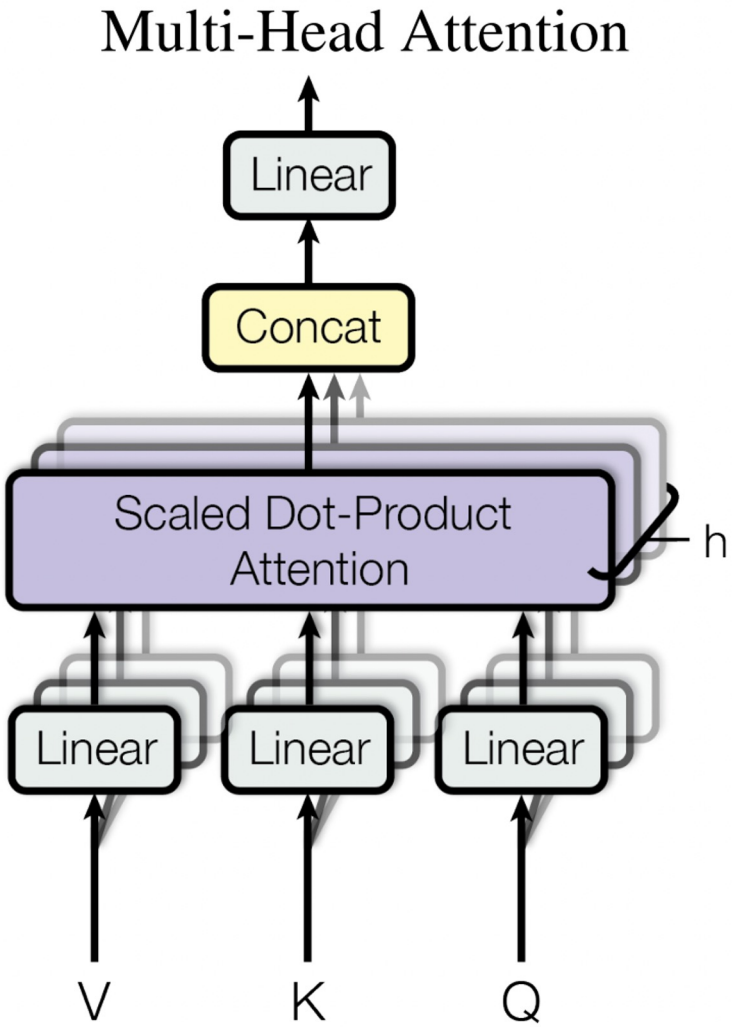
Masked Self Attention



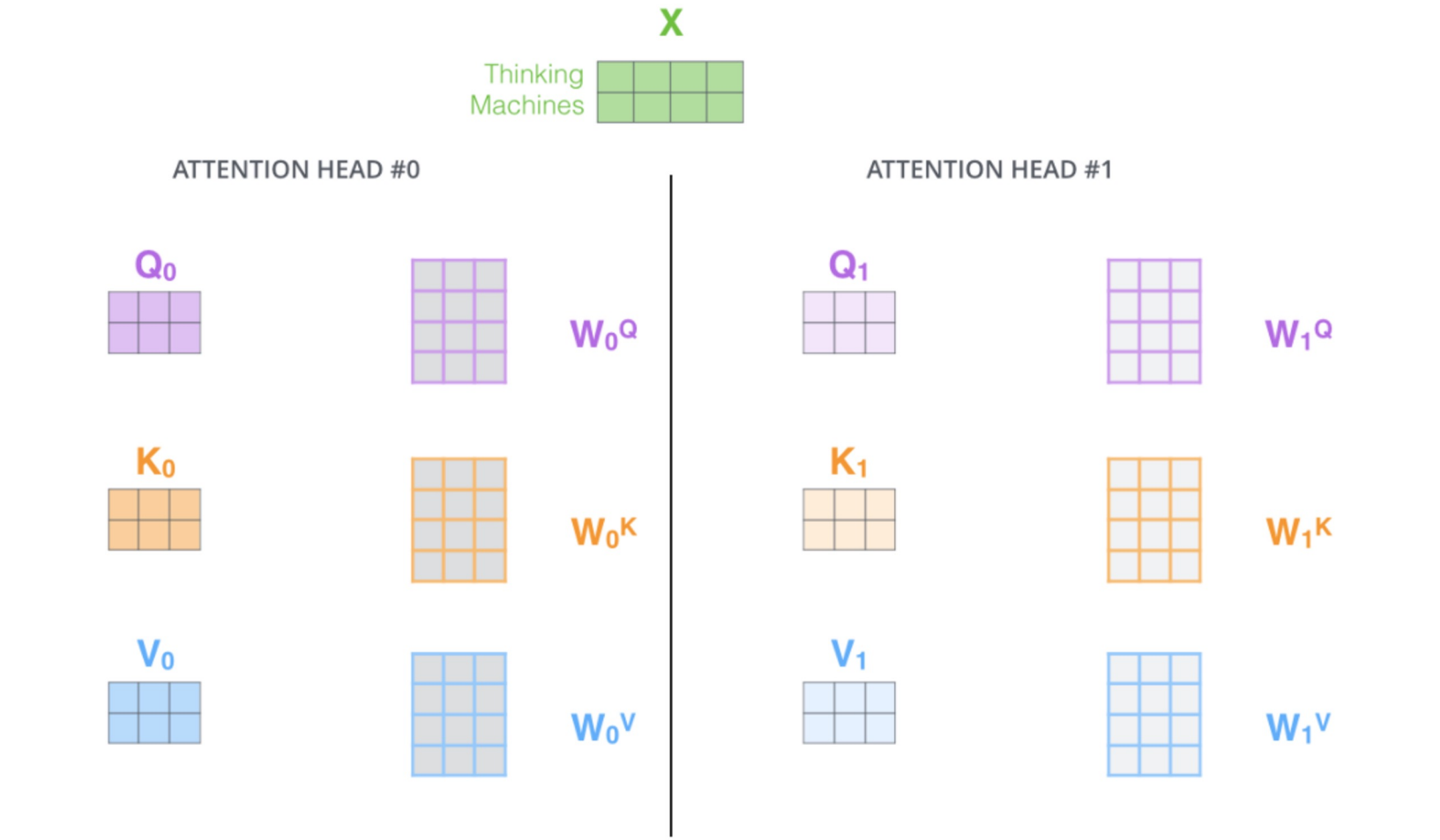
4) Multi-head Self Attention

Multi-head attention allows the model to jointly attend to information from different representations.

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

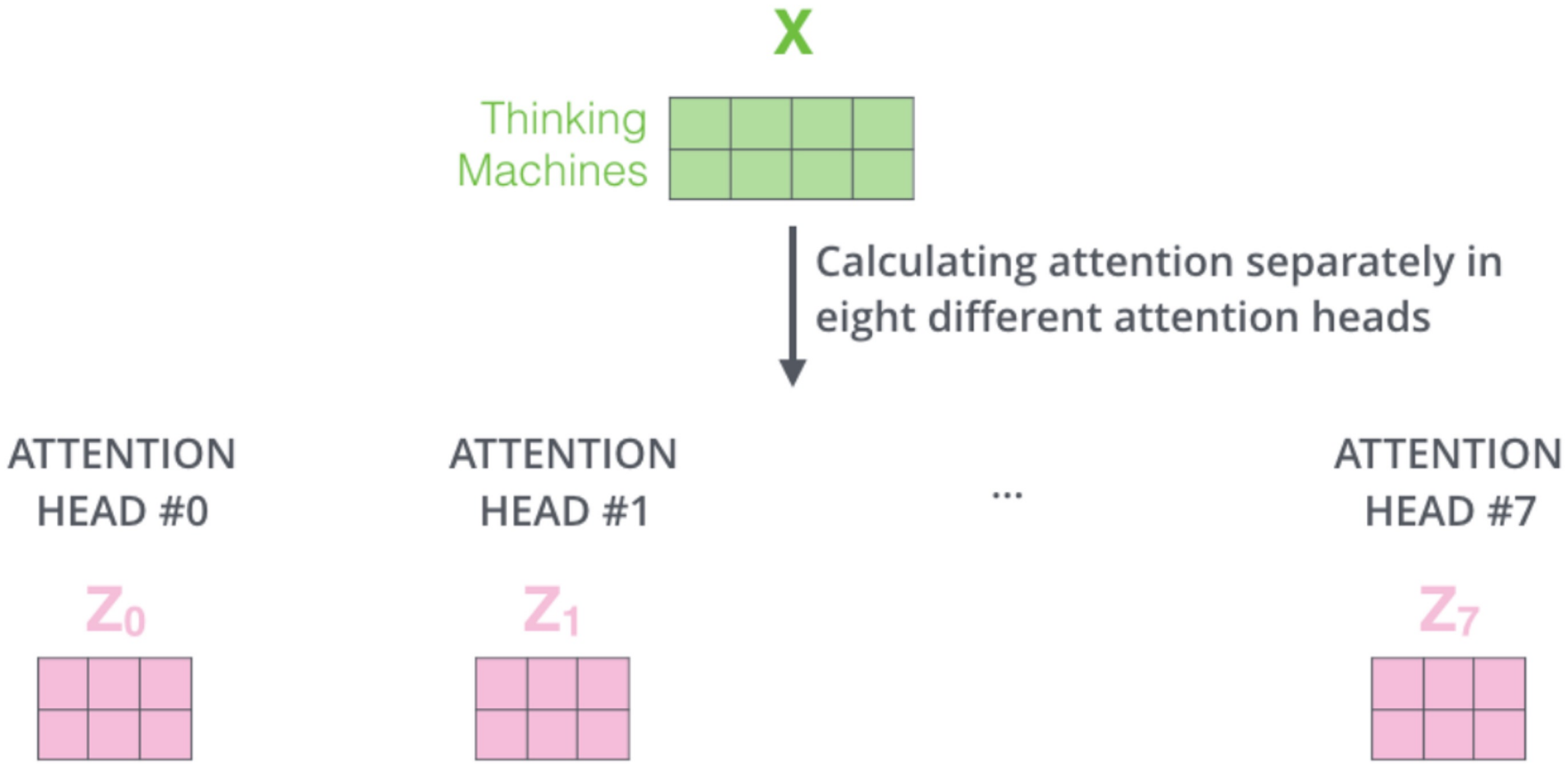


Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.



With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

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

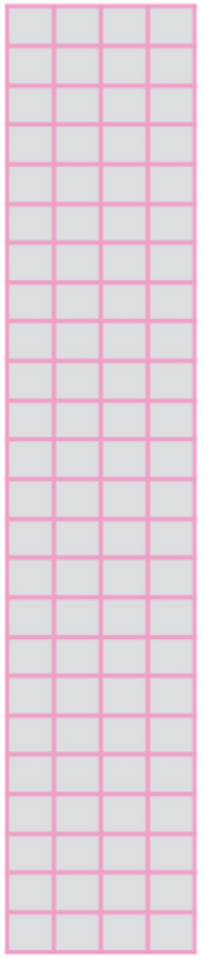


http://jalammar.github.io/illustrated-transformer/

1) Concatenate all the attention heads

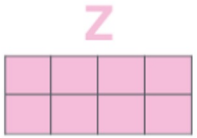
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



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

1) This is our input sentence*

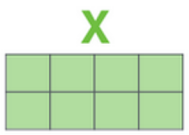
2) We embed each word*

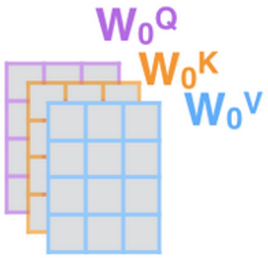
3) Split into 8 heads. We multiply X or R with weight matrices

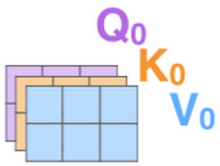
4) Calculate attention using the resulting Q/K/V matrices

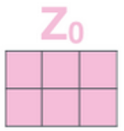
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

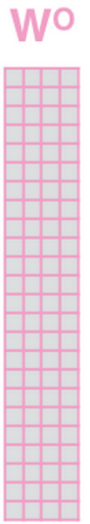
Thinking Machines

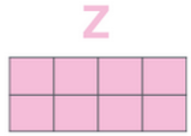




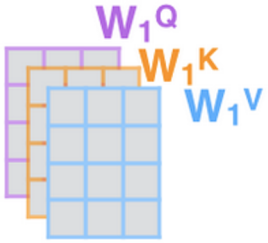


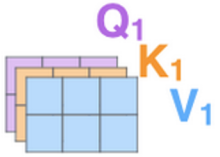


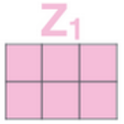




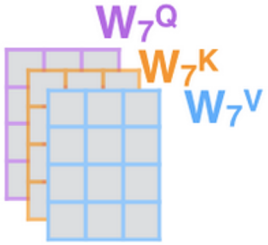
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

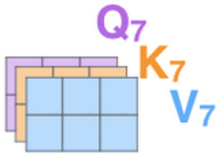


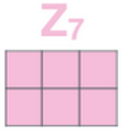




...



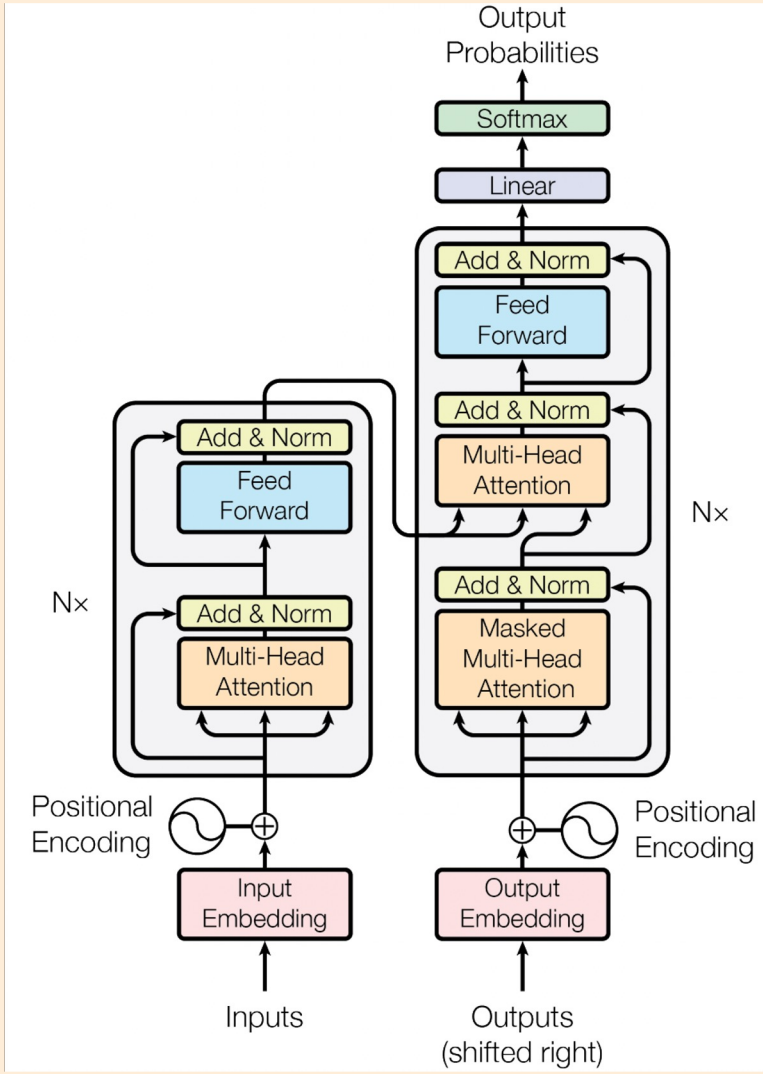




http://jalammar.github.io/illustrated-transformer/

Positional Encodings

The black cat fights the white cat.



Positional Encodings

Without RNN, the model **cannot** make use of the *order* of the input sequence, e.g. the first, second, or third token.

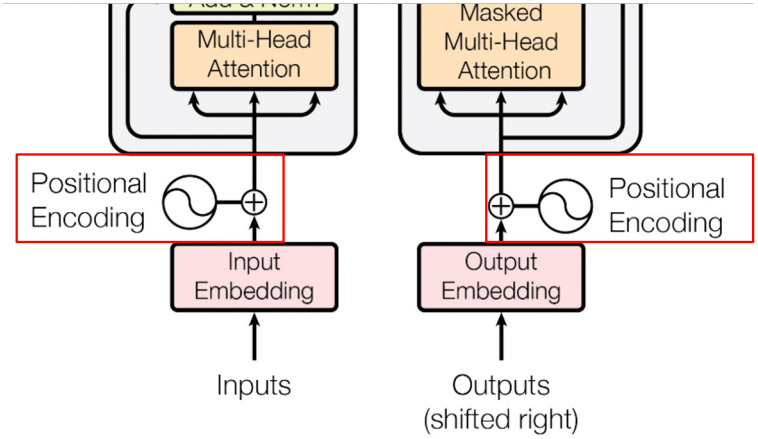


Figure 1: The Transformer - model architecture.

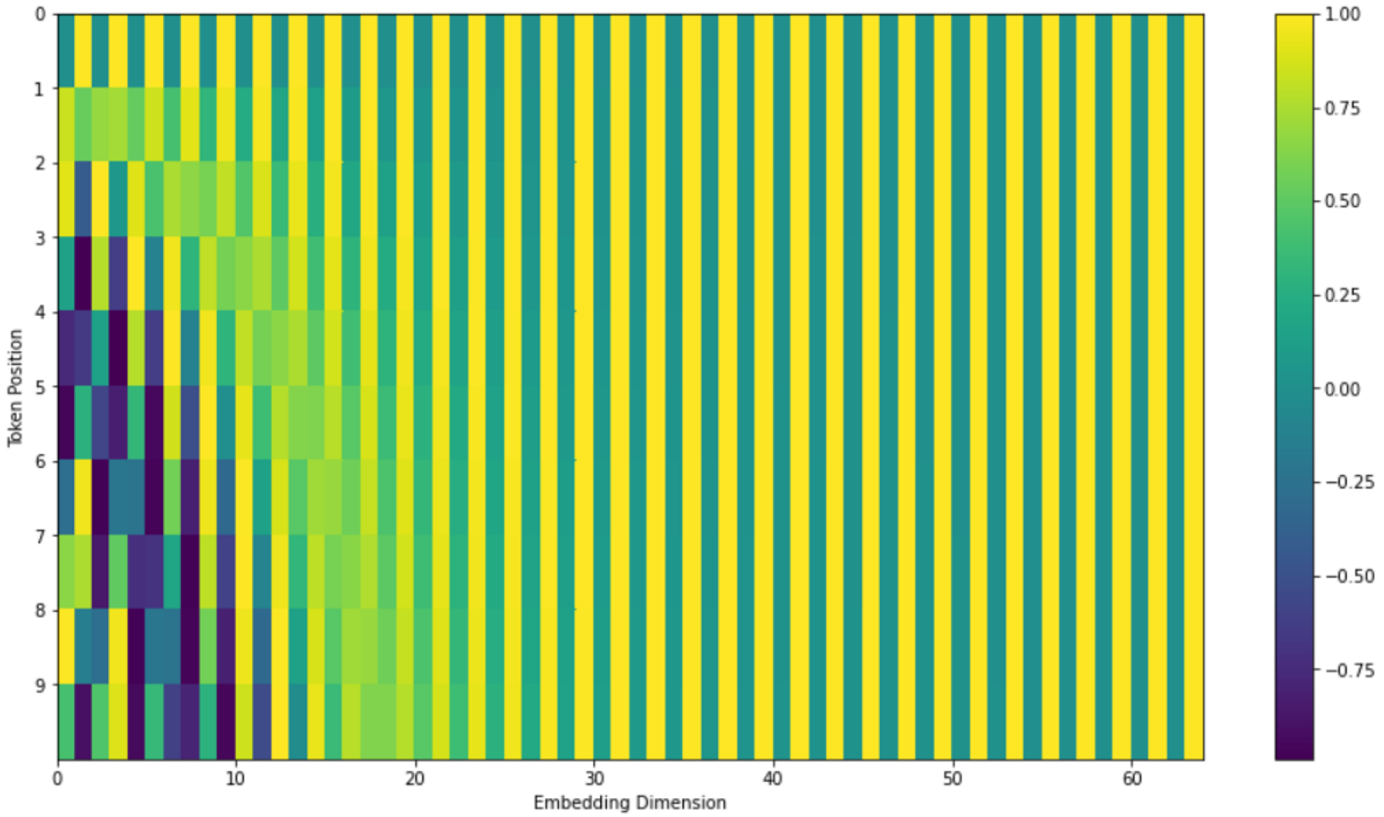
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos is the token position, *i* is the dimension

The black cat fights the white cat.

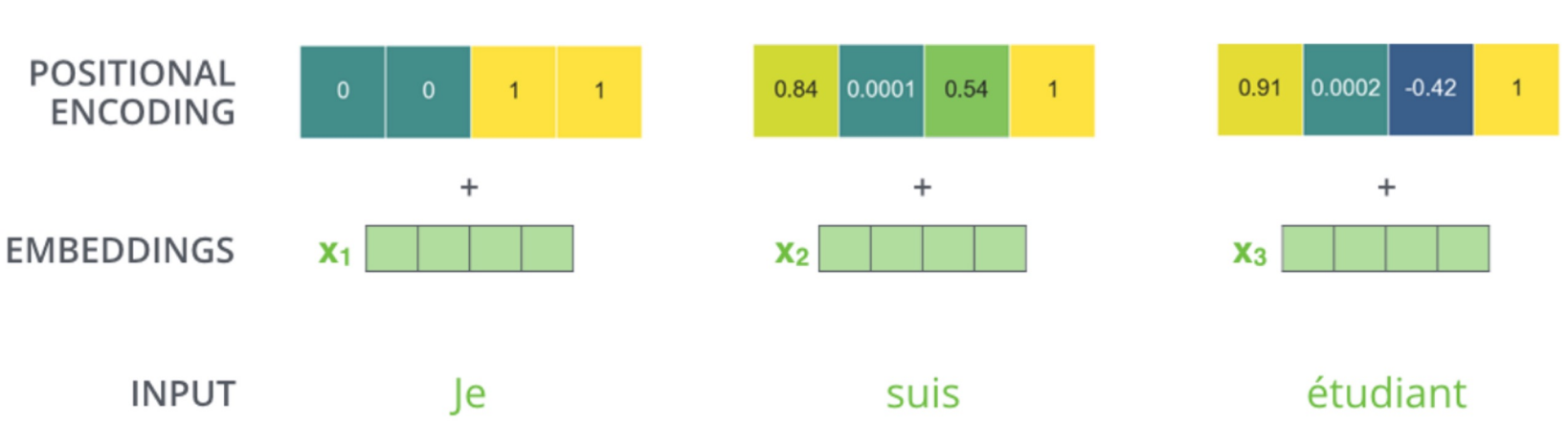
Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

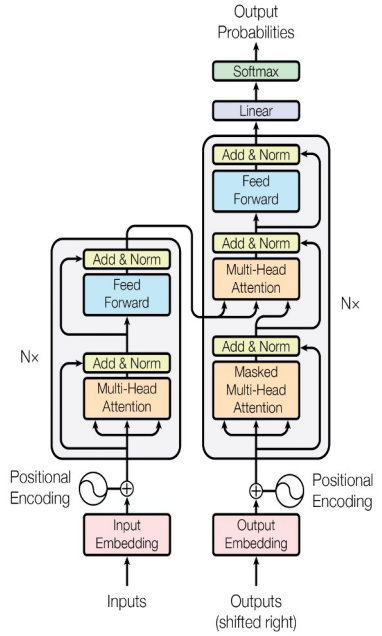
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Attention Is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

i = 4





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

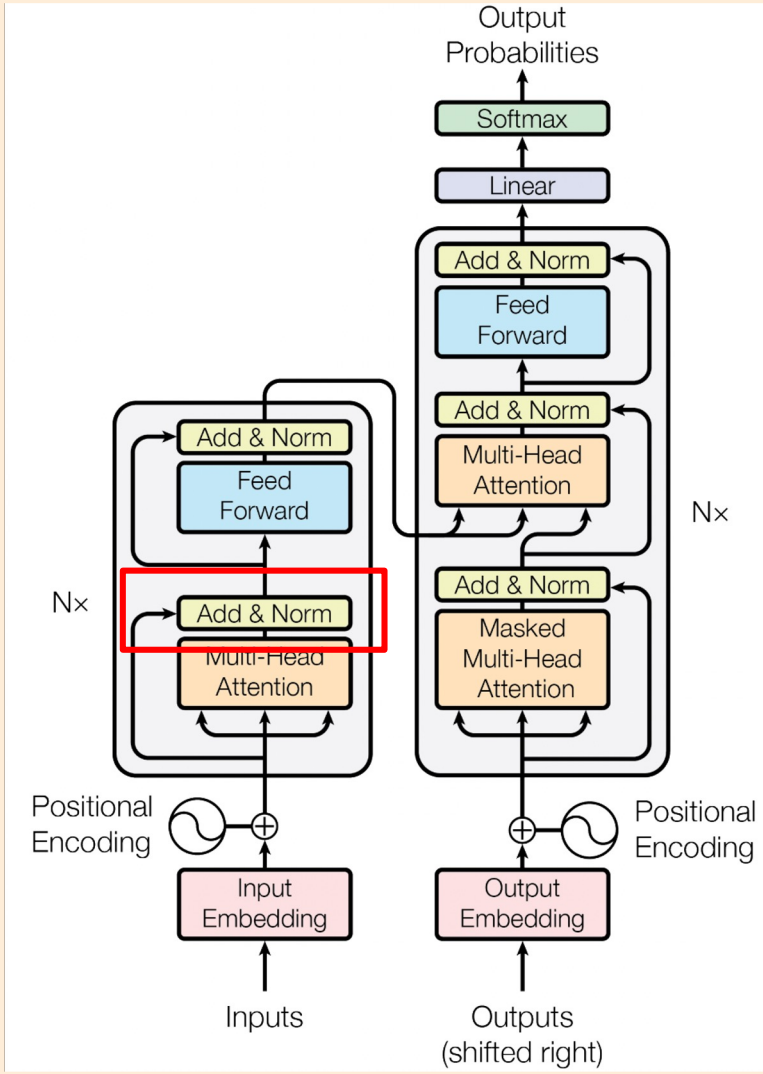
Types of positional encoding

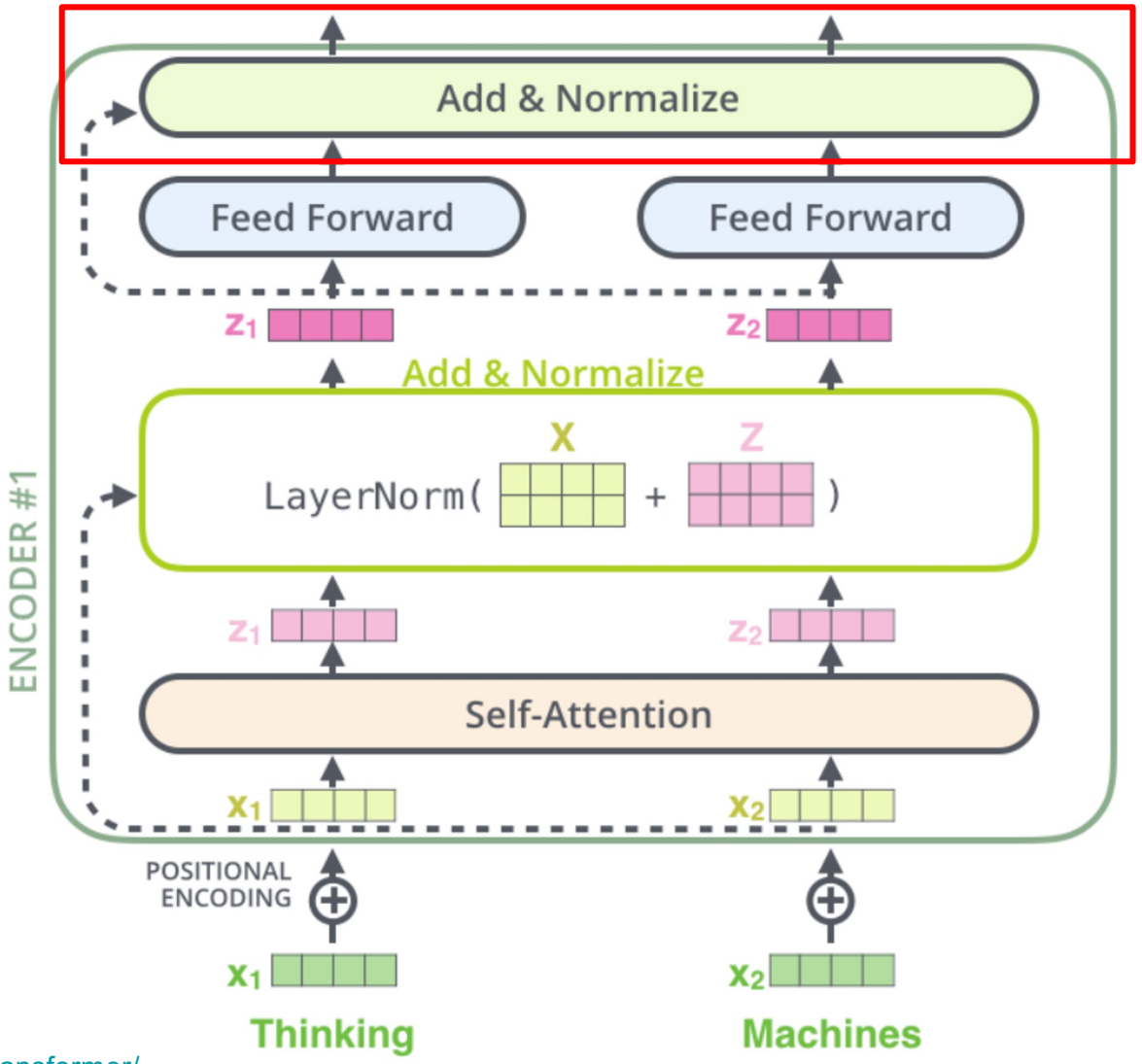
1. Absolute Positional encoding
    - 1.1) Fixed encoding (Transformer)
        - E.g. sinusoidal forms
    - 1.2) Learned encoding (GPT)

2. Relative Positional encoding (GPT NeoX)

	Token 1	Token 2	Token 3	Token 4	Token 5	Token 6
Absolute	1	2	3	4	5	6
Relative	-2	-1	0	1	2	3

Layer Normalization





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

Batch norm. (each feature (channel)) vs Layer norm. (each word)

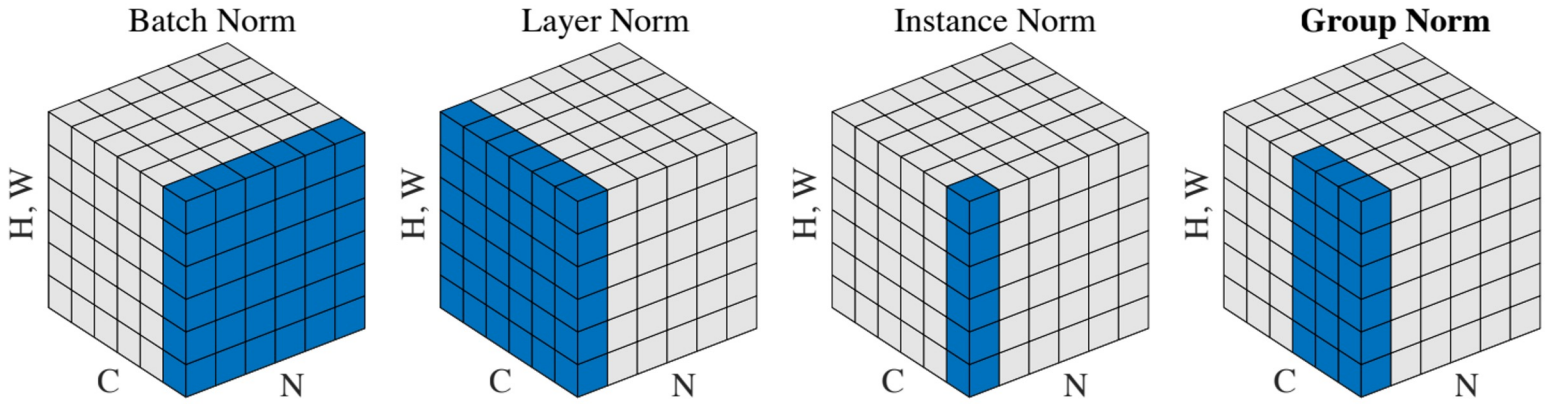
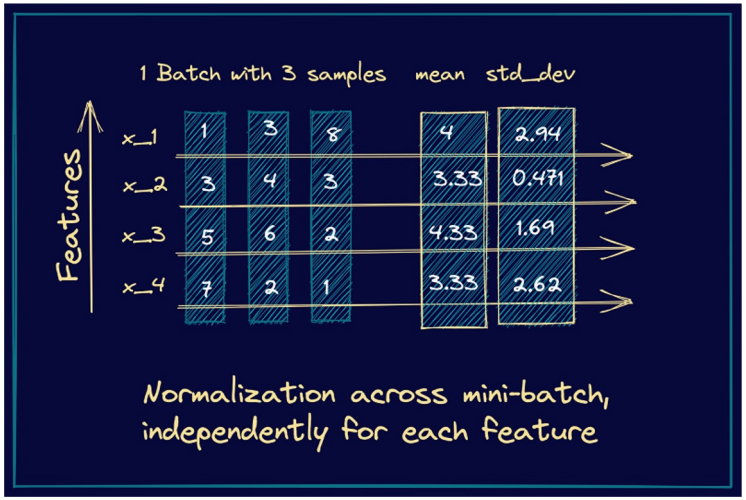
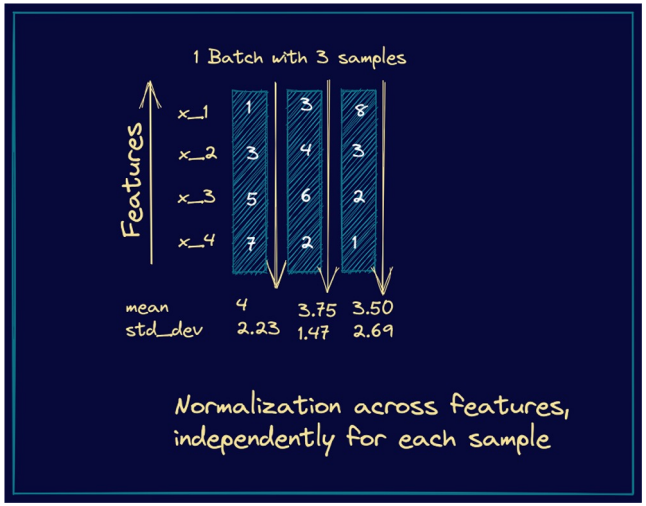


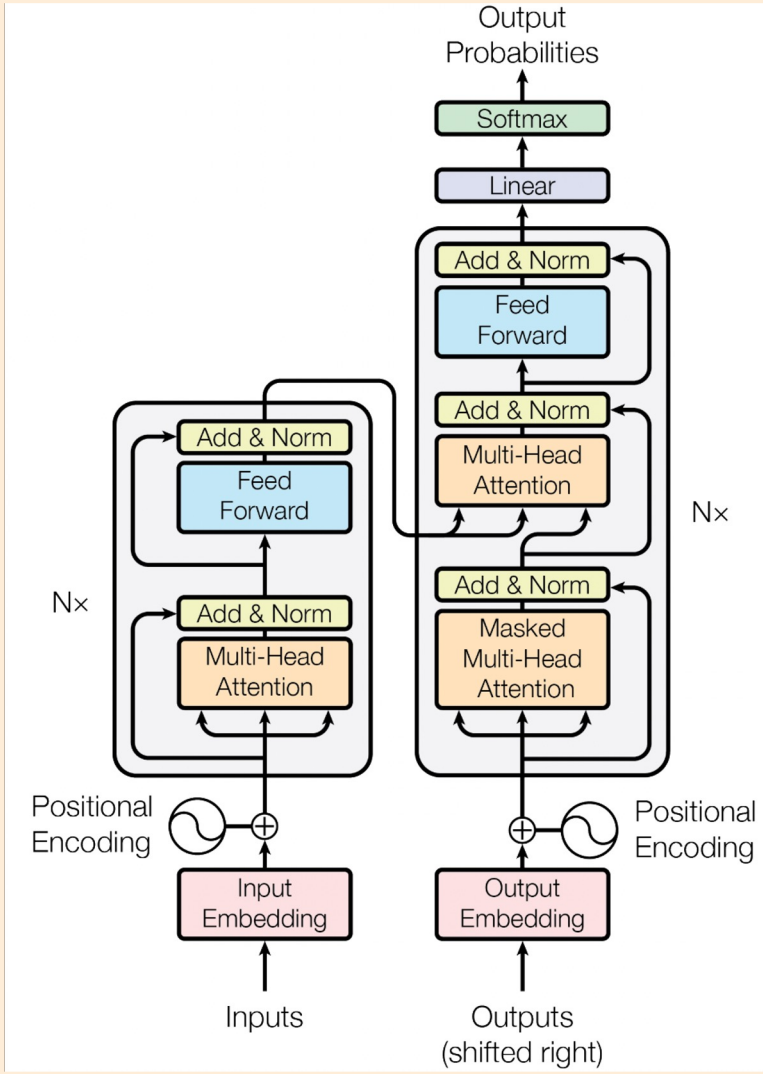
Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.



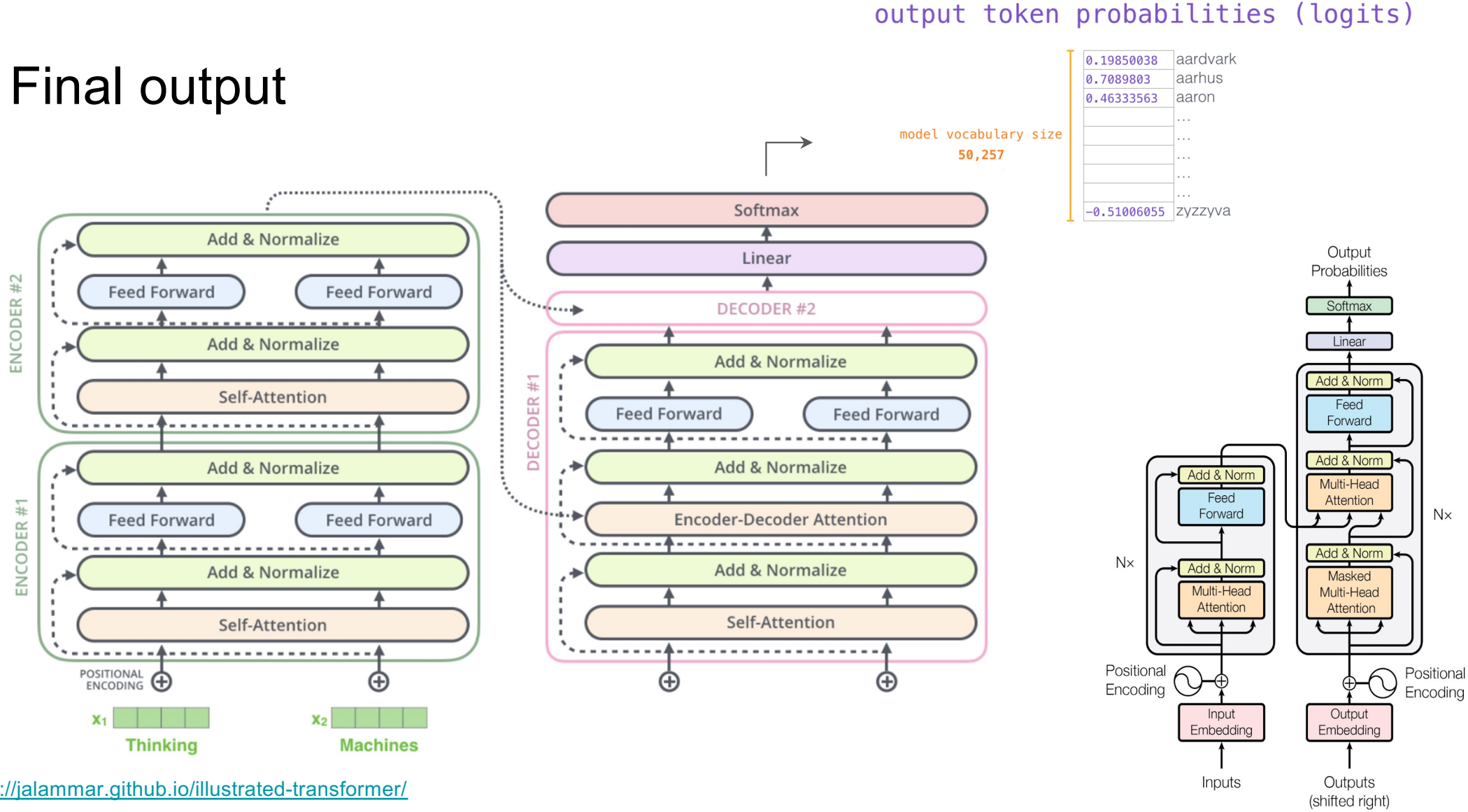


Done:

- Let’s link encoder to decoder
- Then, generate final output

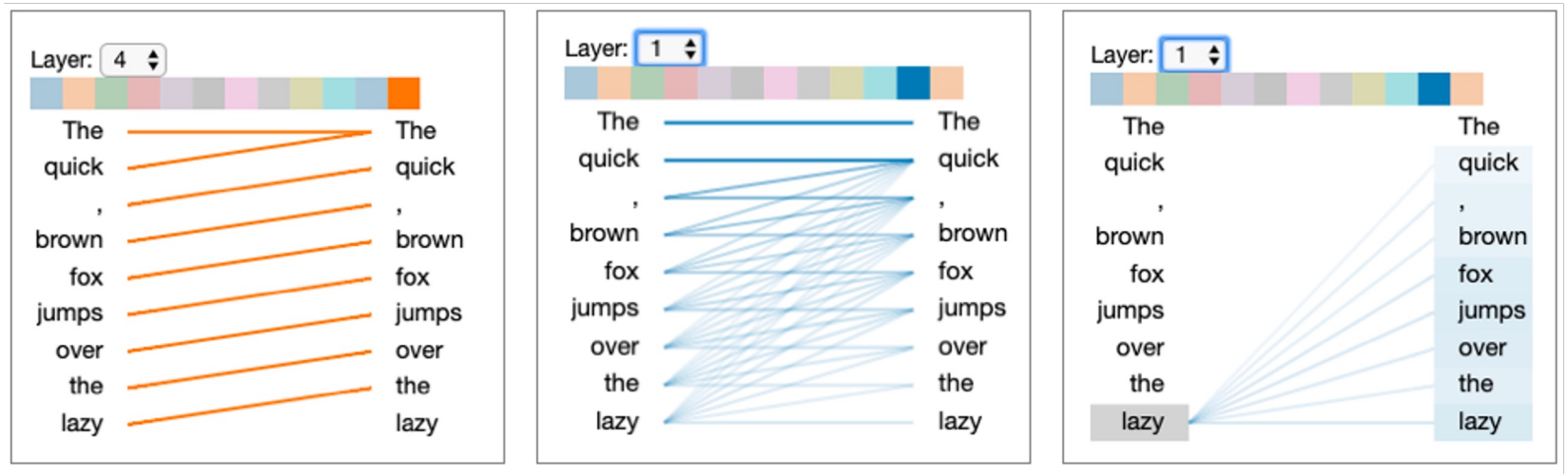


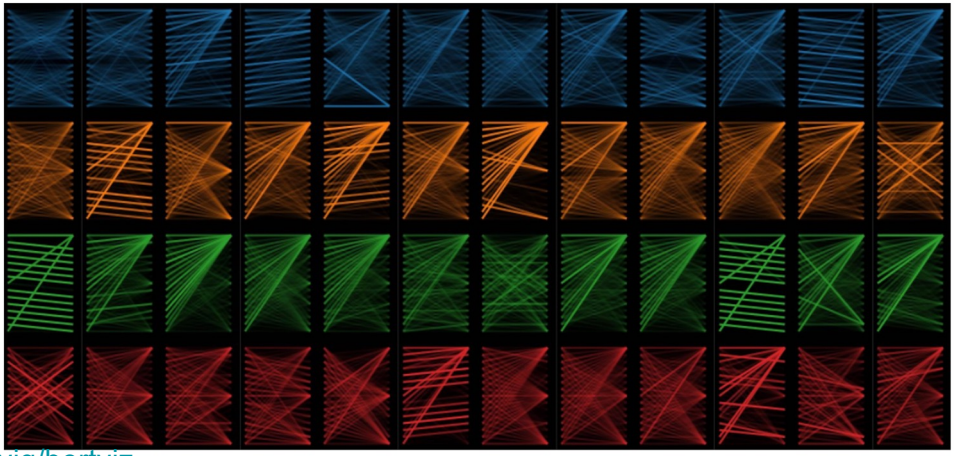
Final output



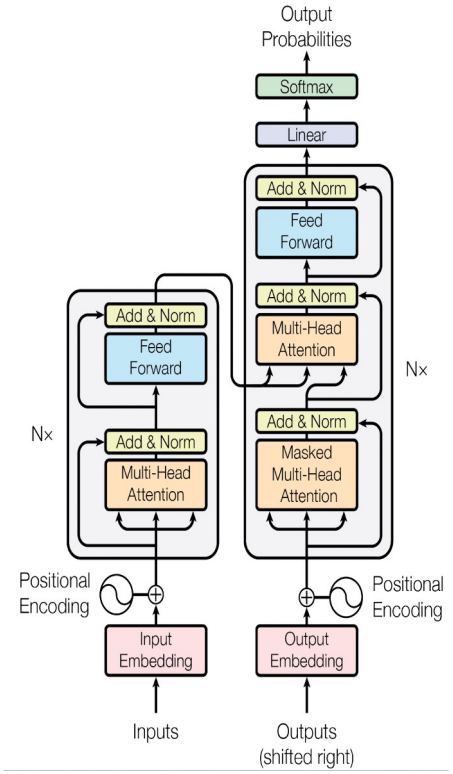
http://jalammar.github.io/illustrated-transformer/

Visualizing Attention (N encoders & N decoders)





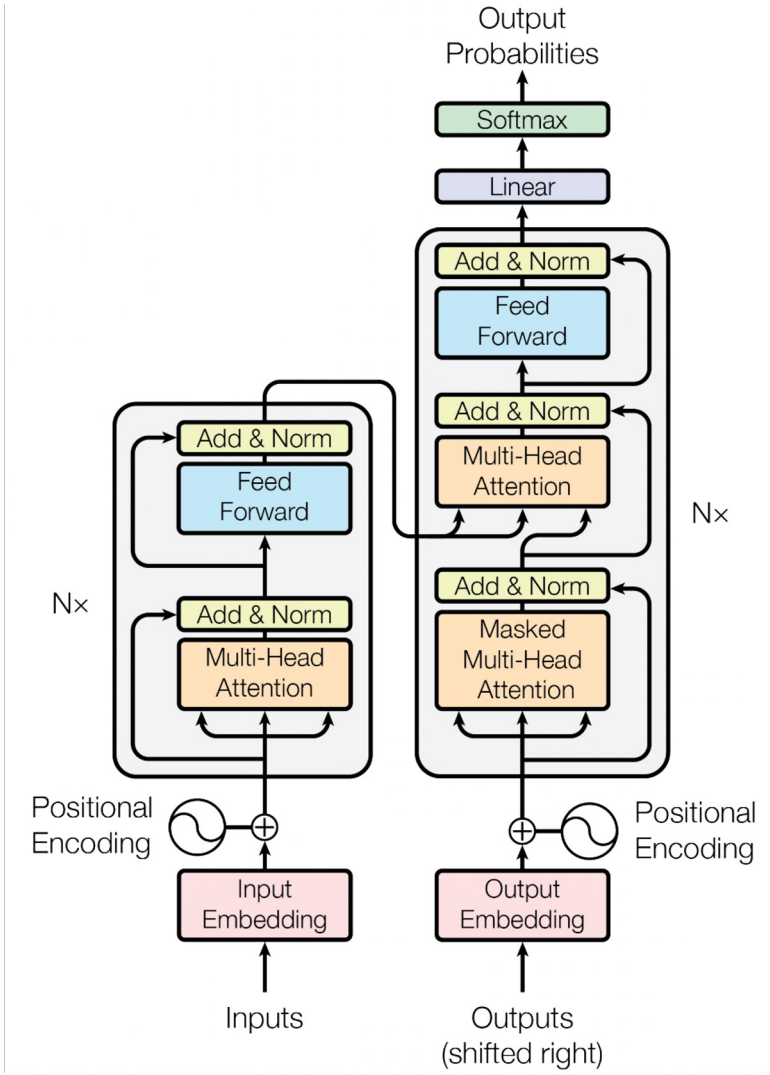
https://github.com/jessevig/bertviz



Transformer-based models

All Transformer based

1. Decoder-based model: GPT
2. Encoder-based model: BERT
3. Encoder and Decoder: BART



Any questions ☺