IN5550: Neural Methods in Natural Language Processing

Encoder-Decoder Models, Attention and Transformers

Sondre Wold

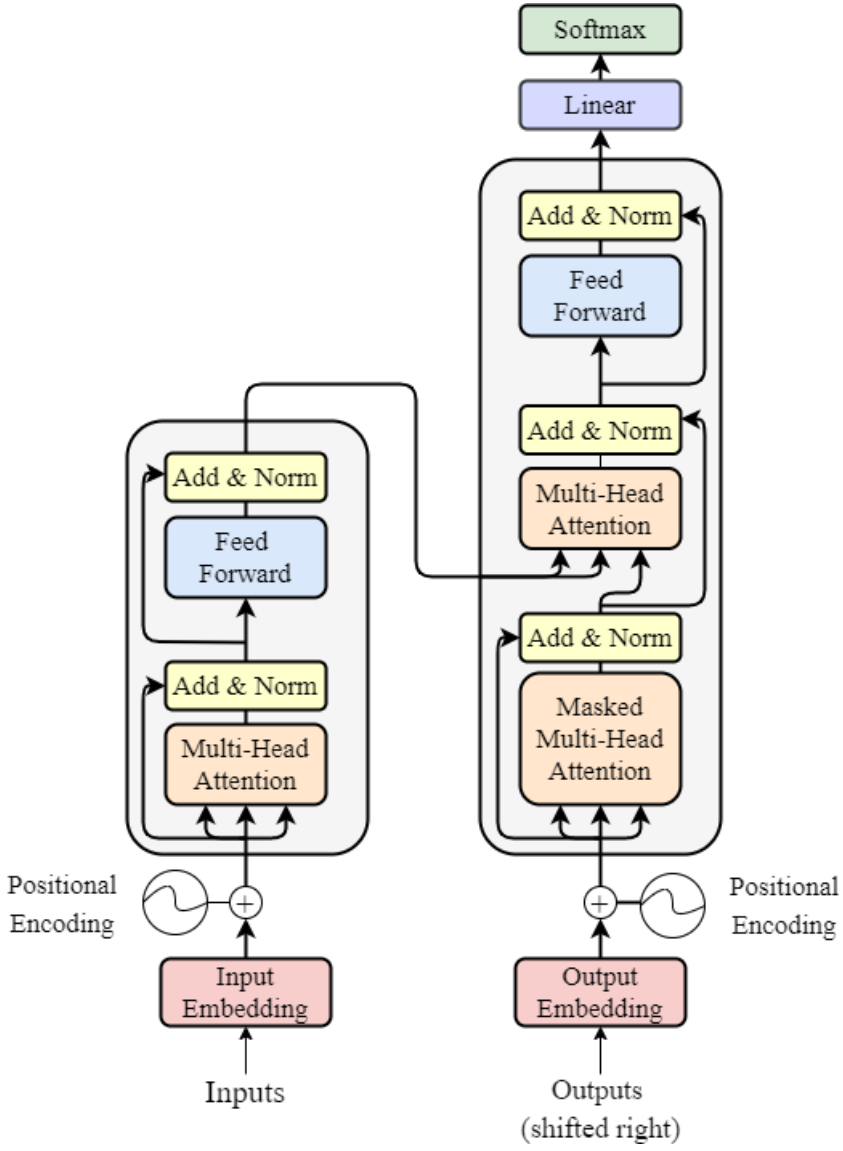
University of Oslo

March 2023



The One

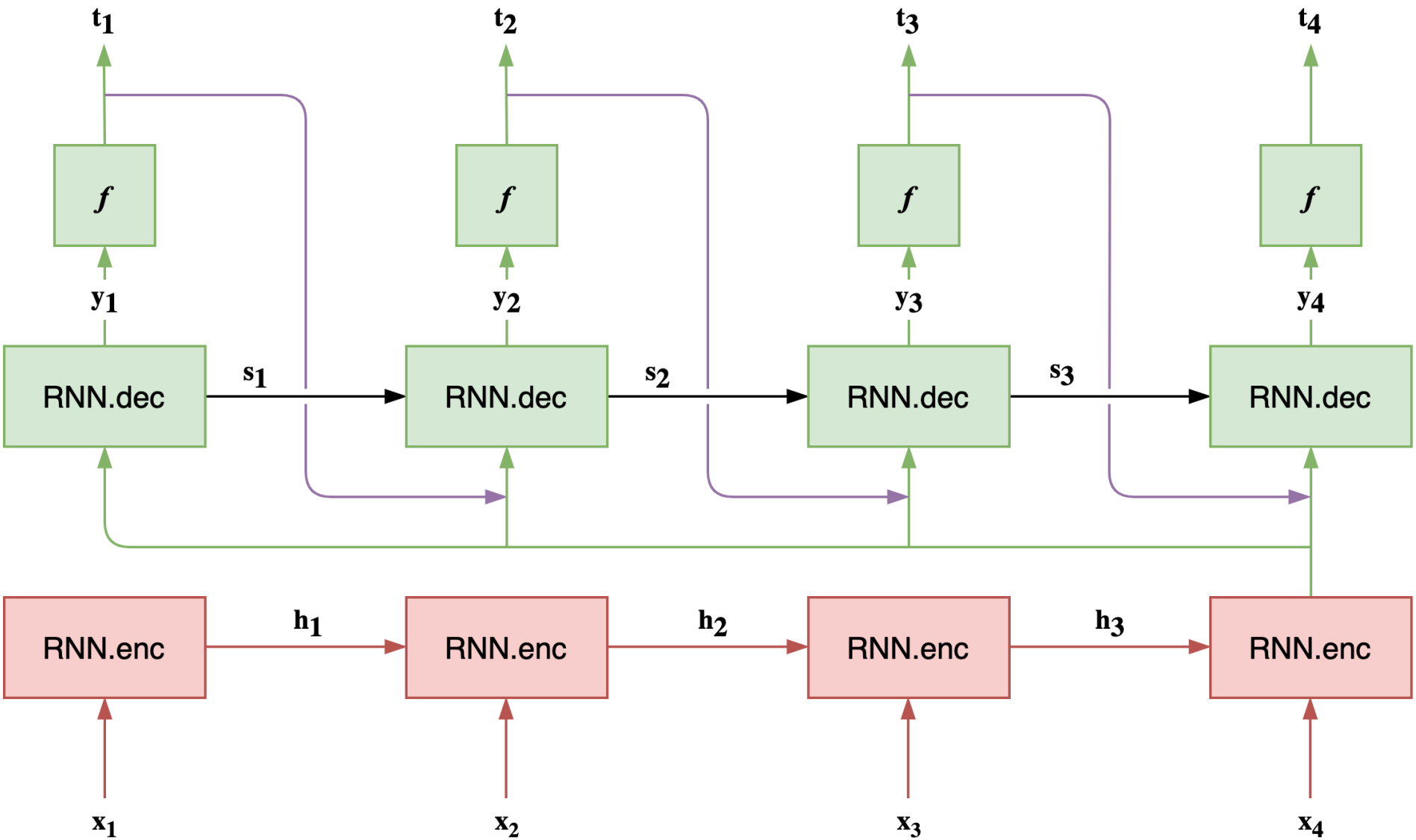
- BER**T**, RoBER**T**a, BAR**T**
- GP**T** variants
- Text-to-Text Transfer **T**ransformer (T5)
- PaLM, BARD, LaMDA, LLaMA bla bla bla
- Used for text-to-image in DALL-E, Stable Diffusion etc..



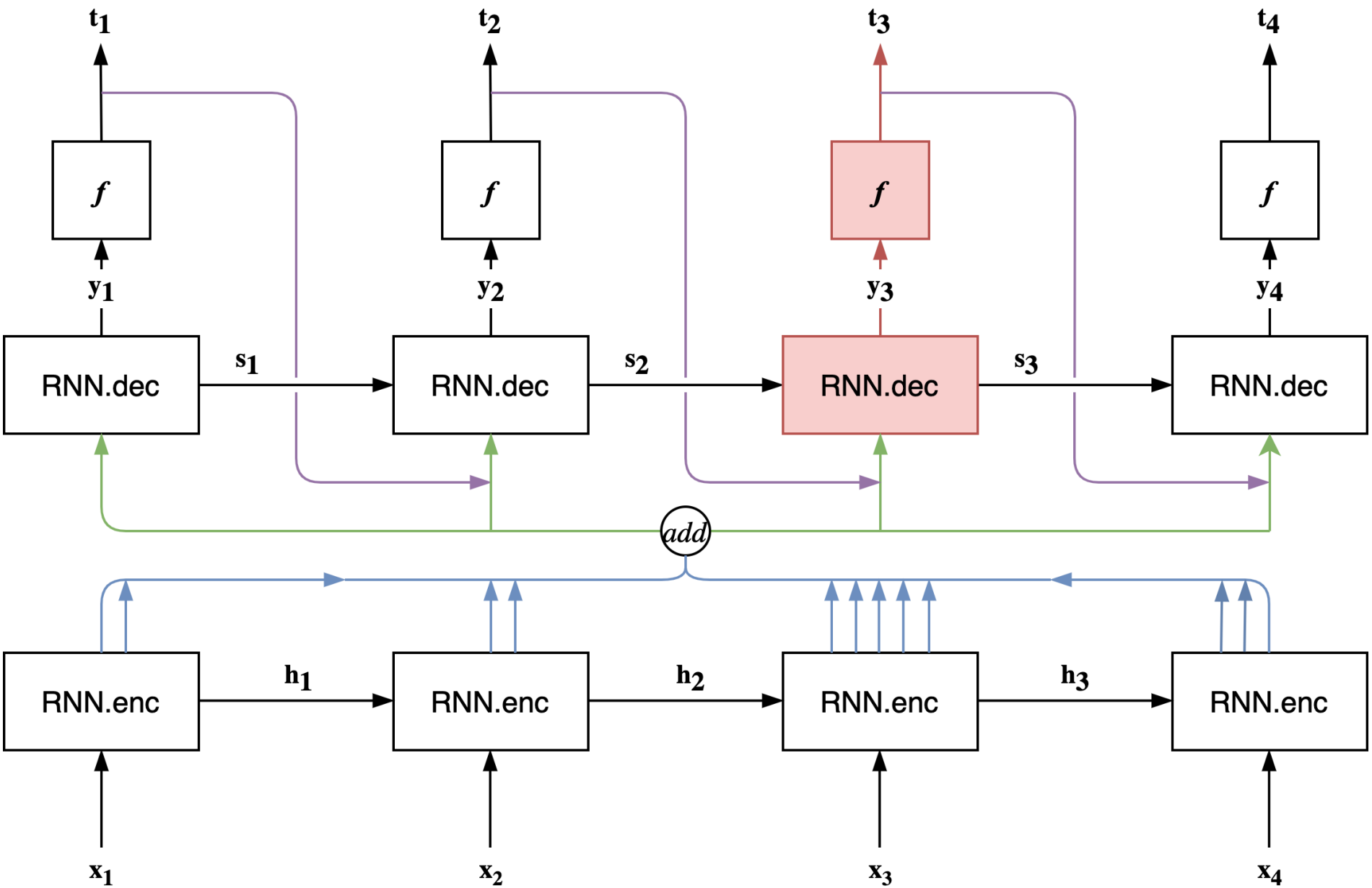
Outline

1. Motivation
2. High level explanation of the architecture
3. More detailed explanation of the components
   - Self-Attention
   - Adding more heads
   - Encoder
   - Decoder
   - Embeddings and positional encodings
4. Walking through the Annotated Transformer from the Harvard NLP group
5. Summary

Where we left off



Where we left off



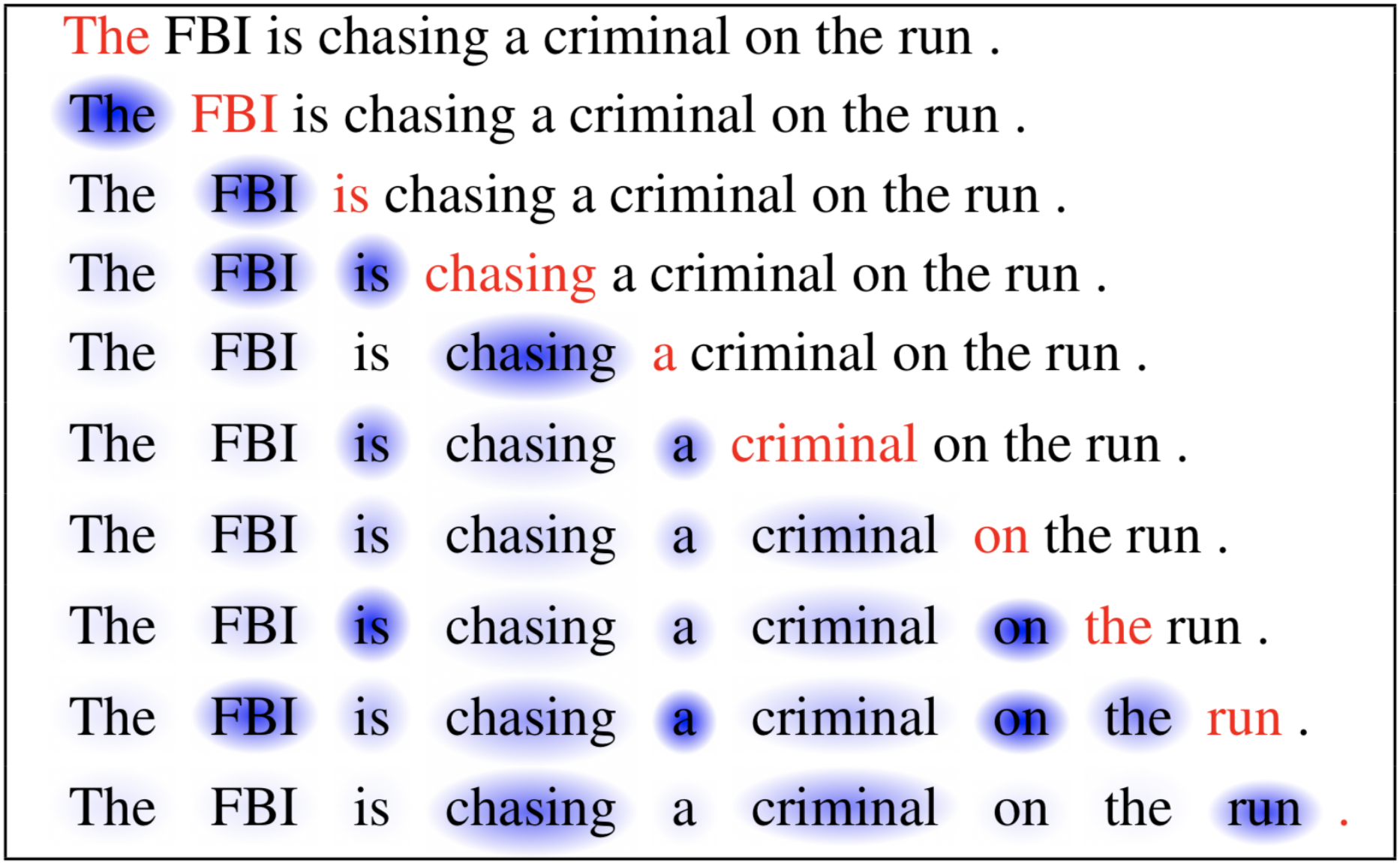
Attention

- We learn how to weight the encoder hidden states to the current time step in the decoder:
  - $c_j = \sum_{i=1}^{T_x} \alpha_{ij} h_i$
  - $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$
- We have a scoring function, an *alignment model*:
  - $e_{ij} = a(s_{i-1}, h_j)$
  - a is a function that tells how relevant an input state i is to an output token j
- In [Bahdanau et al., 2015]: "we parametrize the alignment model a as a feedforward neural network which is jointly trained with all the other components of the proposed system".

Self-Attention

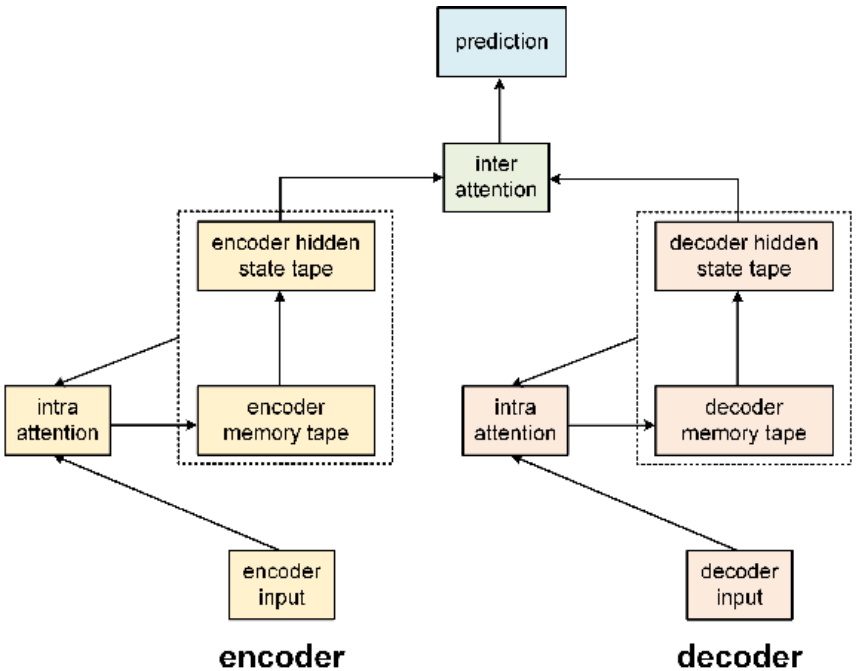
- In the classic Encoder-Decoder, attention is calculated between the decoder and all the hidden states from the encoder — inter level
- But what about the intra-level?
  - "Torvald drove down the street without paying much attention until he suddenly hit a mailbox."
  - We need a way to relate different positions in the same sequence.

Self-Attention

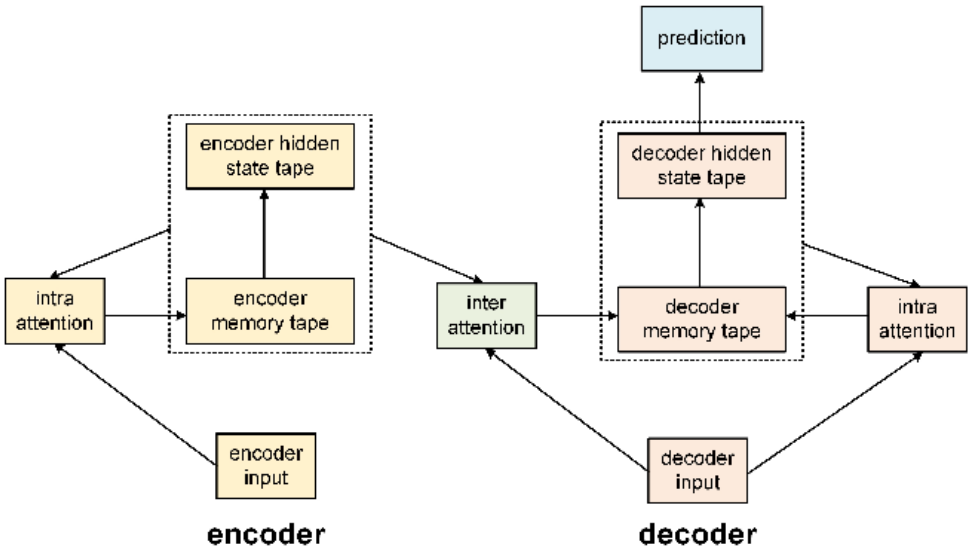


[Cheng et al., 2016]

Self-Attention



(a) Decoder with shallow attention fusion.



(b) Decoder with deep attention fusion.

[Cheng et al., 2016]

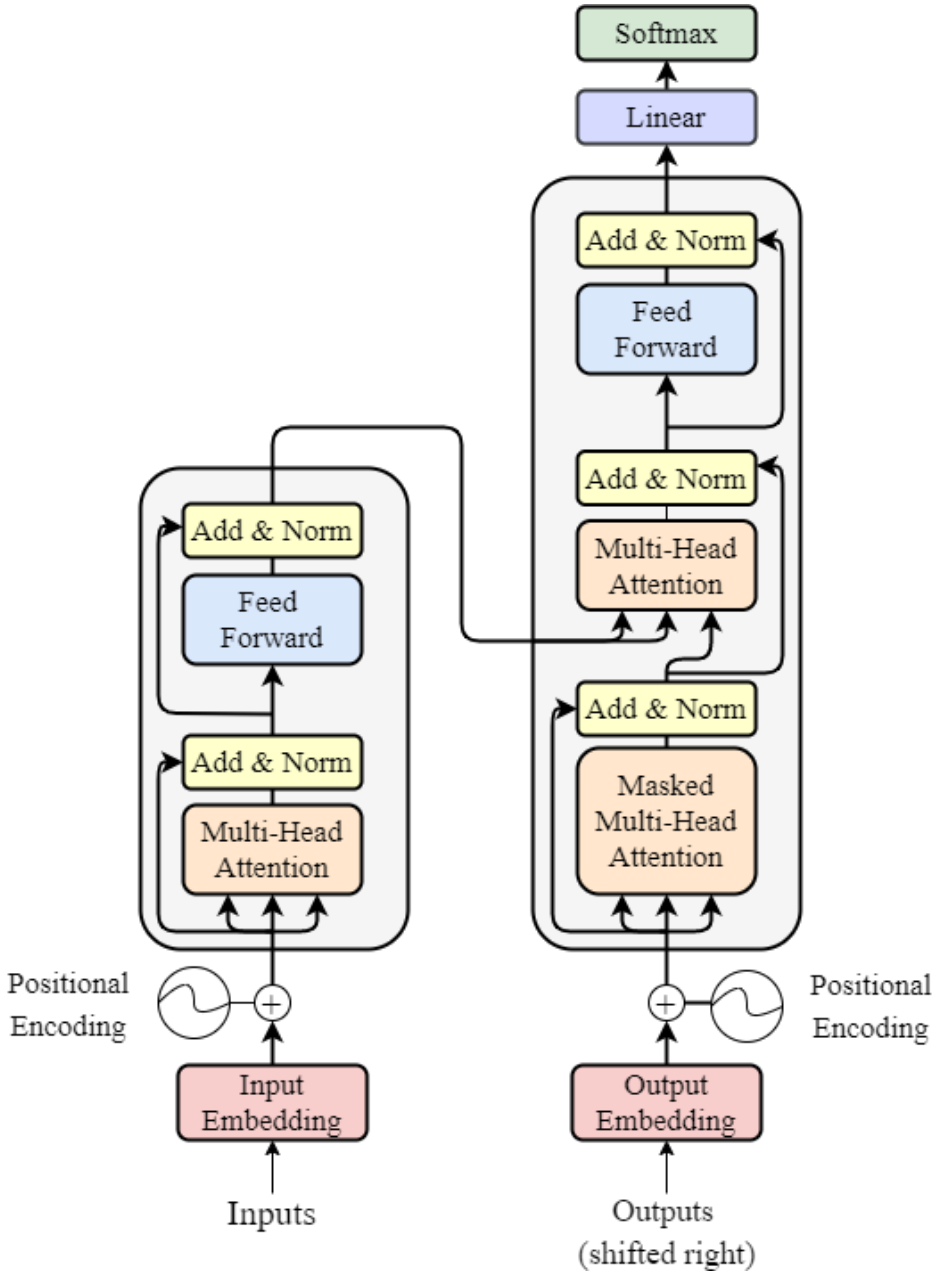
Attention is all you need

"The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder and decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a novel, simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely."

– [Vaswani et al., 2017]

– Per 28.02.2023: cited 66,7k times

The Transformer, at a glance



The Transformer, at a glance

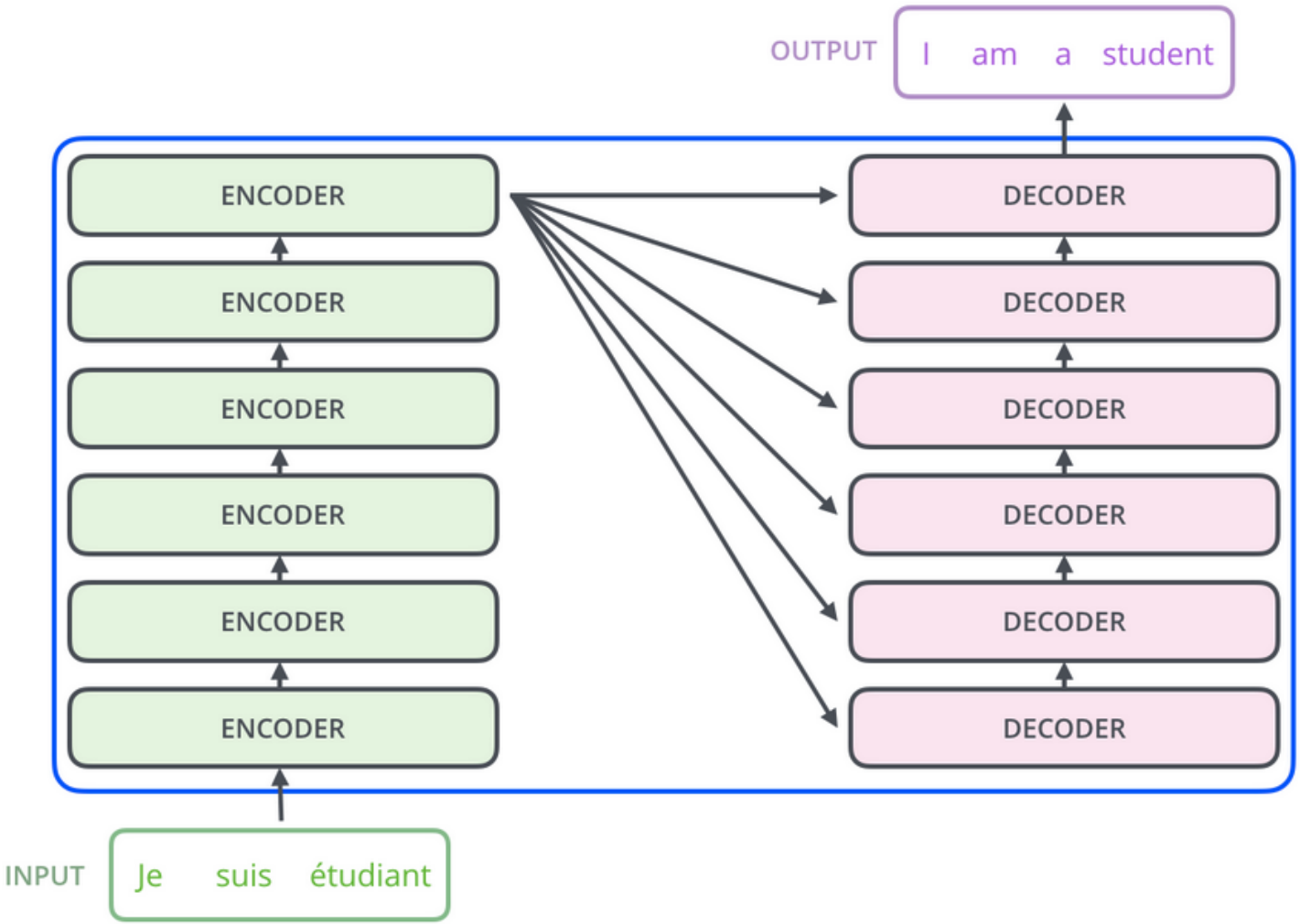
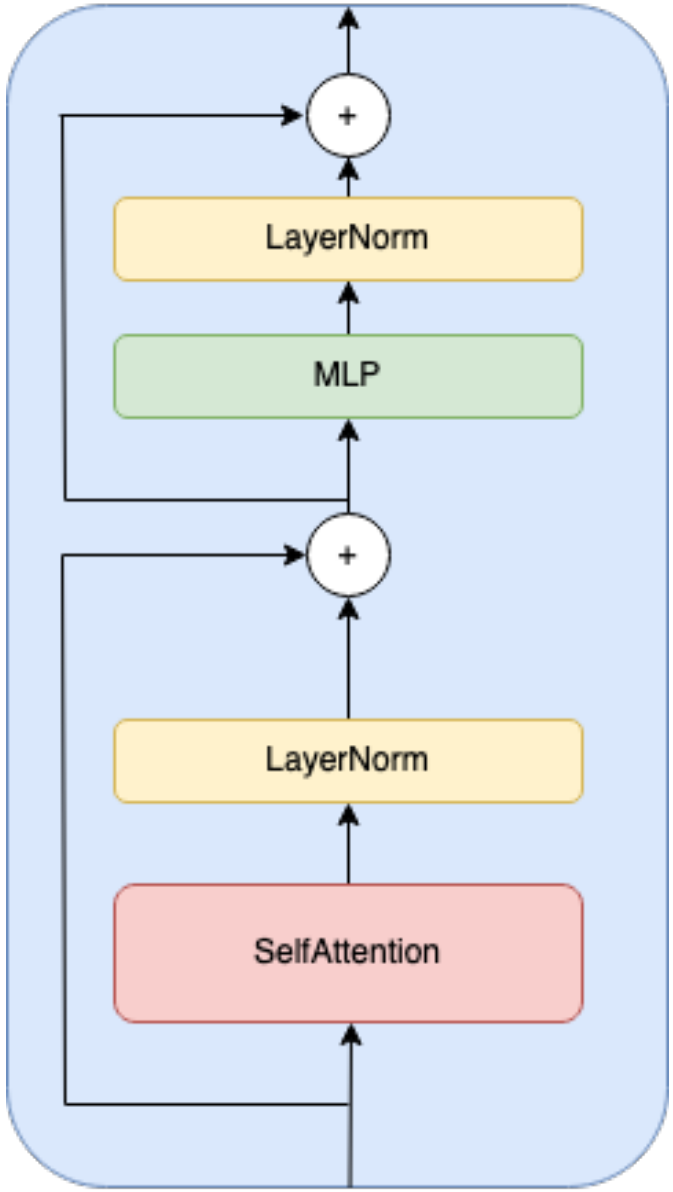


Figure from Jay Alammar's, *The Illustrated Transformer*[1]

[1] https://jalammar.github.io/illustrated-transformer/

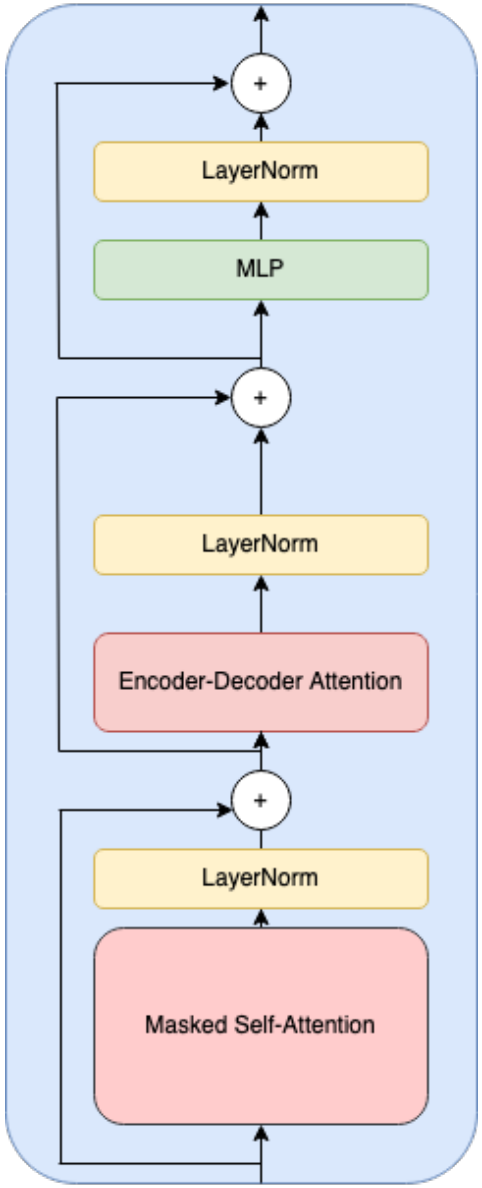
Encoder: High level explanation

- Self-Attention
- Layer normalization and residual
  - LayerNorm from [Ba et al., 2016]
- A FFNN
- Layer normalization and residual



Decoder: High level explanation

- Masked Self-Attention!
  - For the Self-Attention in the decoder, we want to prohibit the model to "look forward" in the sequence when generating an output.
  - Autoregressive property of transducers
- Encoder-Decoder Attention
  - Calculate the attention between decoder and encoder hidden states (inter-level)

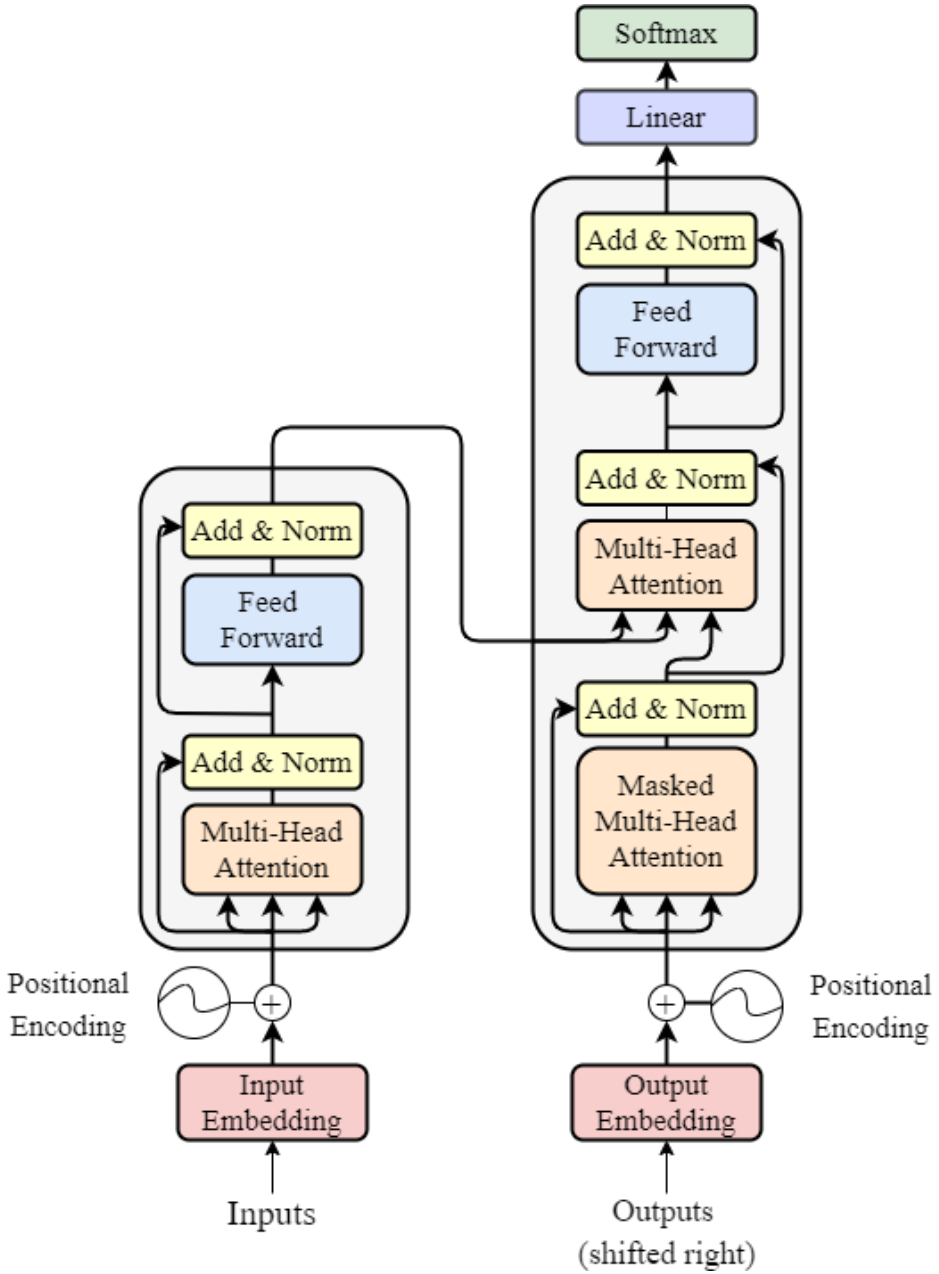


Applications of Attention: high level explanation

Three different applications of attention in the Transformer:

- In the Encoder-Decoder way: we use the "hidden state" from the encoders and do some operation with the state in the decoder
- Encoder Self-Attention: each position in the encoder can attend to previous positions
- Decoder Self-Attention: each position can attend to all previous positions up to that point (autoregressive via masking)

Transformer: piece by piece



Self-Attention: with vectors first

1. From each of the inputs, we create three vectors: the Query, the Key and the Value.

$$W_Q \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 4 & 5 & 6 \end{bmatrix} W_K \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 4 & 5 & 6 \end{bmatrix} W_V \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 4 & 5 & 6 \end{bmatrix} \tag{1}$$

$$\textit{chocolate} = X_1 \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{2}$$

$$Q_1 = X_1 \cdot W_Q = \begin{bmatrix} 12 & 27 & 33 \end{bmatrix} \tag{3}$$

$$K_1 = X_1 \cdot W_K = \begin{bmatrix} 12 & 27 & 33 \end{bmatrix} \tag{4}$$

$$V_1 = X_1 \cdot W_V = \begin{bmatrix} 12 & 27 & 33 \end{bmatrix} \tag{5}$$

Self-Attention with vectors, continued...

2. Calculate a score for each of the words in the input sequence against the other tokens in the sequence:

- $Q_1 \cdot K_1 = 31$
- $Q_1 \cdot K_2 = 21$

3. Then we divide each score by the root of the dimension of the key vector ($\sqrt{d_k}$) (More on this later):

- $x_1 = \frac{Q_1 \cdot K_1}{\sqrt{d_k}} = \frac{31}{\sqrt{3}}$

4. Then we softmax that score

- $\alpha_1 = Softmax(x)$

5. Then we multiply the softmaxed score with the Value vector

- $z_1 = V_1 \cdot \alpha_1$

6. We then sum all the weighted value vectors together:

- $attention_1 = \sum_{i=1}^{n} Z_i$

Self-Attention, the real way

In practice, we use matrices, not vectors:

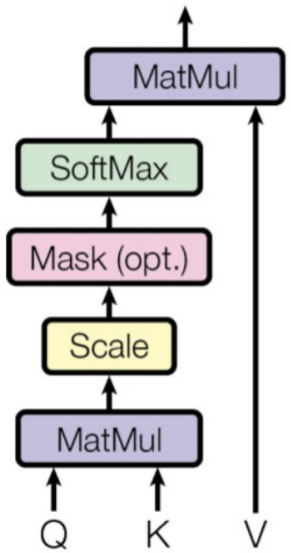
$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, W_Q \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \tag{6}$$

$$Q = X \cdot W_Q = \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \tag{7}$$

X has a row_count number of words, each represented with a col_count dimensional vector

Which means...

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



Why dot product attention?

- The two most used attention functions are additive attention (like we saw in [Bahdanau et al., 2015]), and dot-product attention (like we see here)

Why dot product attention?

- The two most used attention functions are additive attention (like we saw in [Bahdanau et al., 2015]), and dot-product attention (like we see here)
- *While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code*.

Why dot product attention?

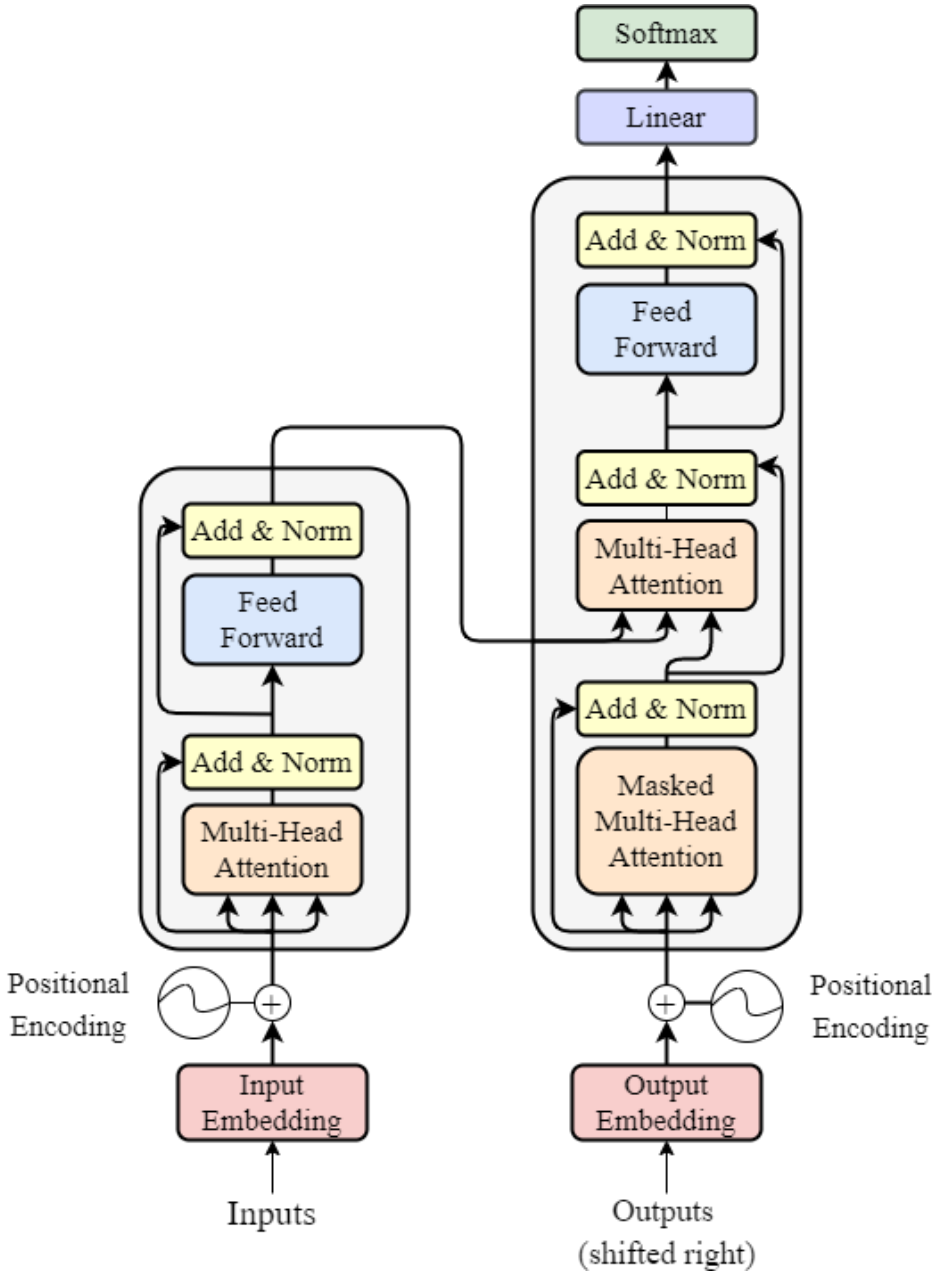
- The two most used attention functions are additive attention (like we saw in [Bahdanau et al., 2015]), and dot-product attention (like we see here)
- *While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.*
- However...what happens with the dot product if d_k gets large?

Why dot product attention?

- The two most used attention functions are additive attention (like we saw in [Bahdanau et al., 2015]), and dot-product attention (like we see here)
- *While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code*.
- However...what happens with the dot product if d_k gets large?
  - We will "push" the softmax, creating small gradients!
  - Solution: Scale down the dot products by $\frac{1}{\sqrt{d_k}}$

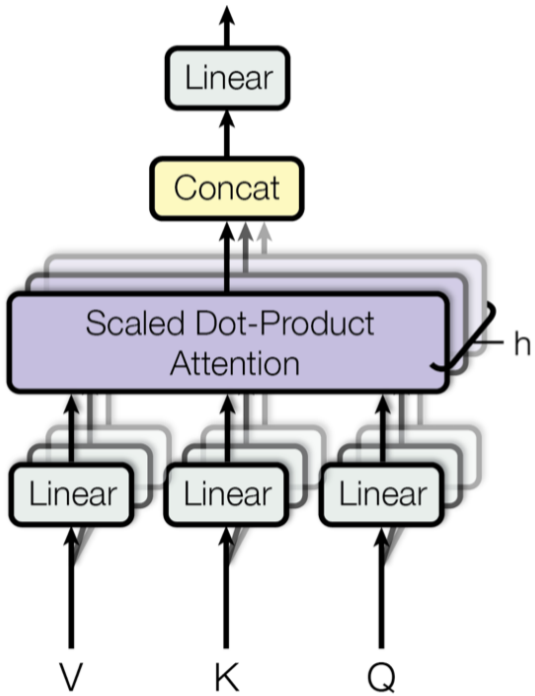
Not that complicated, right?

But wait.. it says MultiHead!?



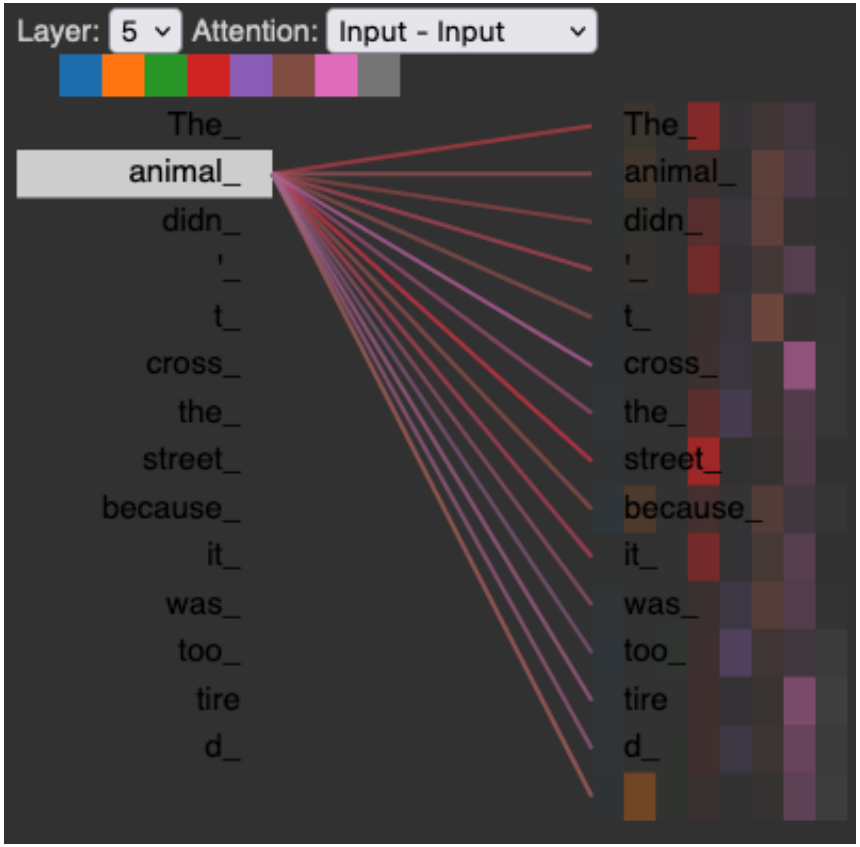
Multi-Head Attention

- Why use only a single set with Q, K, and V matricies — when we can use *many*
- We can use h differently learned linear projections to create Q, K and V
- In the original paper, $h = 8, d_k = d_{model}/h = 64$
- We calculate h different attention matrices, concatenate them and linearly project them
  - Just as computationally costly, since we concat 64-dim matrices 8 times, compared to projecting a 512-dim matrix once



Multi-Head Attention

- *.. allows the model to jointly attend to information from different representation subspaces at different positions*



Multi-Head Attention

...and that was attention!

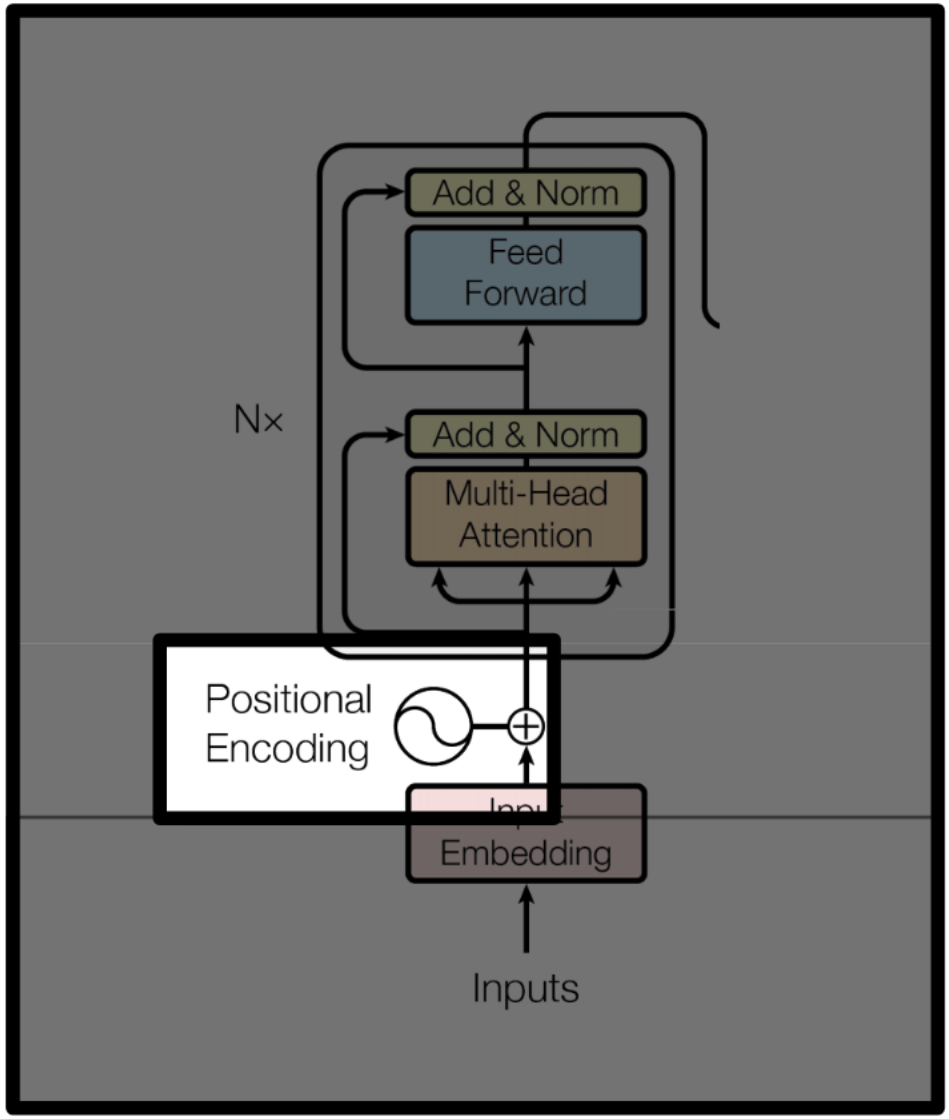
- In the Encoder-Decoder way: queries come from the previous decoder layer, and the keys and values come from the output of the encoder
- Encoder Self-Attention: All queries, keys and values come from the same place, the output of the previous layer (or the embedding if we are at the first layer)
- Decoder Self-Attention: Same as above, but with masking for autoregressive purposes.

But there is trouble in paradise...

There is a problem with our embeddings..

- RNNs process words sequentially
- We ditched the recurrence, opting for MultiHead-Attention
  - Which is done in parallel
- Consequence: our Transformer model has no sense of word ordering!

Solution: positional encoding



Positional encoding: intuition

- We want to add some information to each word about its position in a sequence

Positional encoding: intuition

- We want to add some information to each word about its position in a sequence
- First, simple idea:
  - Add a running number

Positional encoding: intuition

- We want to add some information to each word about its position in a sequence
- First, simple idea:
  - Add a running number
  - Word number one gets 0, word two gets 1, word n gets n

Positional encoding: intuition

- We want to add some information to each word about its position in a sequence
- First, simple idea:
  - Add a running number
  - Word number one gets 0, word two gets 1, word n gets n
  - Problem: Values would get large, and lengths not seen during training could show up during evaluation

Positional encoding: intuition

- We want to add some information to each word about its position in a sequence
- First, simple idea:
  - Add a running number
  - Word number one gets 0, word two gets 1, word n gets n
  - Problem: Values would get large, and lengths not seen during training could show up during evaluation
- Second idea:
  - Add a running number from [0,1]

Positional encoding: intuition

- We want to add some information to each word about its position in a sequence
- First, simple idea:
  - Add a running number
  - Word number one gets 0, word two gets 1, word n gets n
  - Problem: Values would get large, and lengths not seen during training could show up during evaluation
- Second idea:
  - Add a running number from [0,1]
  - Problem: How many words are there in a specific range? Not consistent across sequences...

Positional encoding: criteria

- We need a unique encoding for each timestep
- Timestep-deltas should be consistent across sequences
- We need to generalize to longer sentences
- It must be deterministic

Positional encoding: solution

From the original text sequence, we get an embedding:

$$X = \begin{bmatrix} \textit{Chocolate} \\ \textit{is} \\ \textit{good} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Positional encoding: solution

From the original text sequence, we get an embedding:

$$X = \begin{bmatrix} \textit{Chocolate} \\ \textit{is} \\ \textit{good} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$Embedding = X \cdot \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}, \mathbb{R}^{vocab \times dim_{model}} \tag{9}$$

Positional encoding: solution

From the original text sequence, we get an embedding:

$$X = \begin{bmatrix} \textit{Chocolate} \\ \textit{is} \\ \textit{good} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$Embedding = X \cdot \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}, \mathbb{R}^{vocab \times dim_{model}} \tag{9}$$

$$= \begin{bmatrix} 4.2 & 1.12 & 0.3 \\ 21.12 & 3.1 & 3.21 \\ 2.32 & 1.1 & 321 \end{bmatrix}, \mathbb{R}^{seq_{len} \times dim_{model}} \tag{10}$$

Positional encoding: solution

Instead of adding a number to each embedding, we can add a vector!

$$X = X + P = \begin{bmatrix} 4.2 & 1.12 & 0.3 \\ 21.12 & 3.1 & 3.21 \\ 2.32 & 1.1 & 321 \end{bmatrix} + \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \tag{11}$$

Positional encoding: solution

Instead of adding a number to each embedding, we can add a vector!

$$X = X + P = \begin{bmatrix} 4.2 & 1.12 & 0.3 \\ 21.12 & 3.1 & 3.21 \\ 2.32 & 1.1 & 321 \end{bmatrix} + \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \tag{11}$$

But what to put in P?

Positional encoding: solution

Enters: sine and cosine!

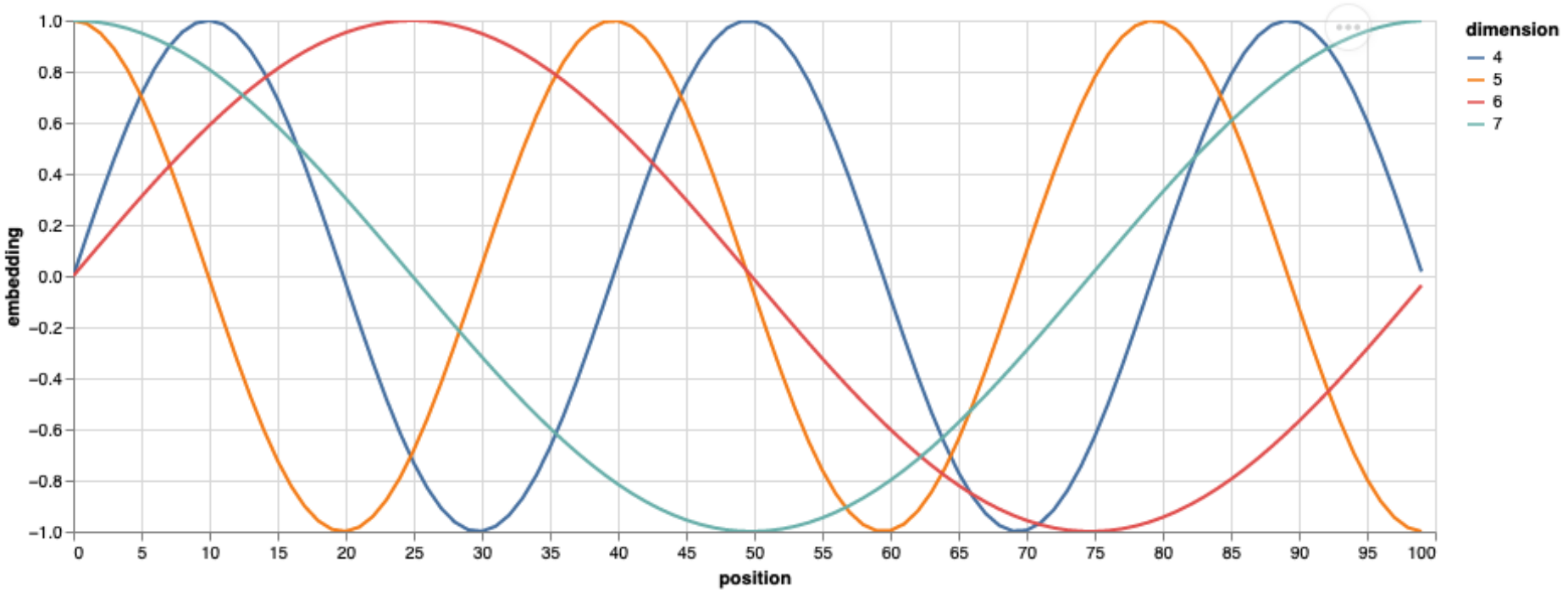
$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

i= dimension index, d_{model}=embedding size, pos=position of the word we are calculating

Positional encoding: solution

If the size of our embeddings is 4, so $d_{model} = 4$, then $d_P = 4$, which gives us four waves with different frequencies:



Then, the positional encoding for time step pos is the "column vector" at x=pos in the figure above.

Positional encoding: solution

$$X = X + P = \begin{bmatrix} 4.2 & 1.12 & 0.3 \\ 21.12 & 3.1 & 3.21 \\ 2.32 & 1.1 & 321 \end{bmatrix} + \begin{bmatrix} sin(pos/10000^{2*1/d_{model}}) \\ cos(pos/10000^{2*1/d_{model}}) \\ sin(pos/10000^{2*2/d_{model}}) \\ .. \end{bmatrix} \quad (12)$$

Need more?

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

So how good is it?

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

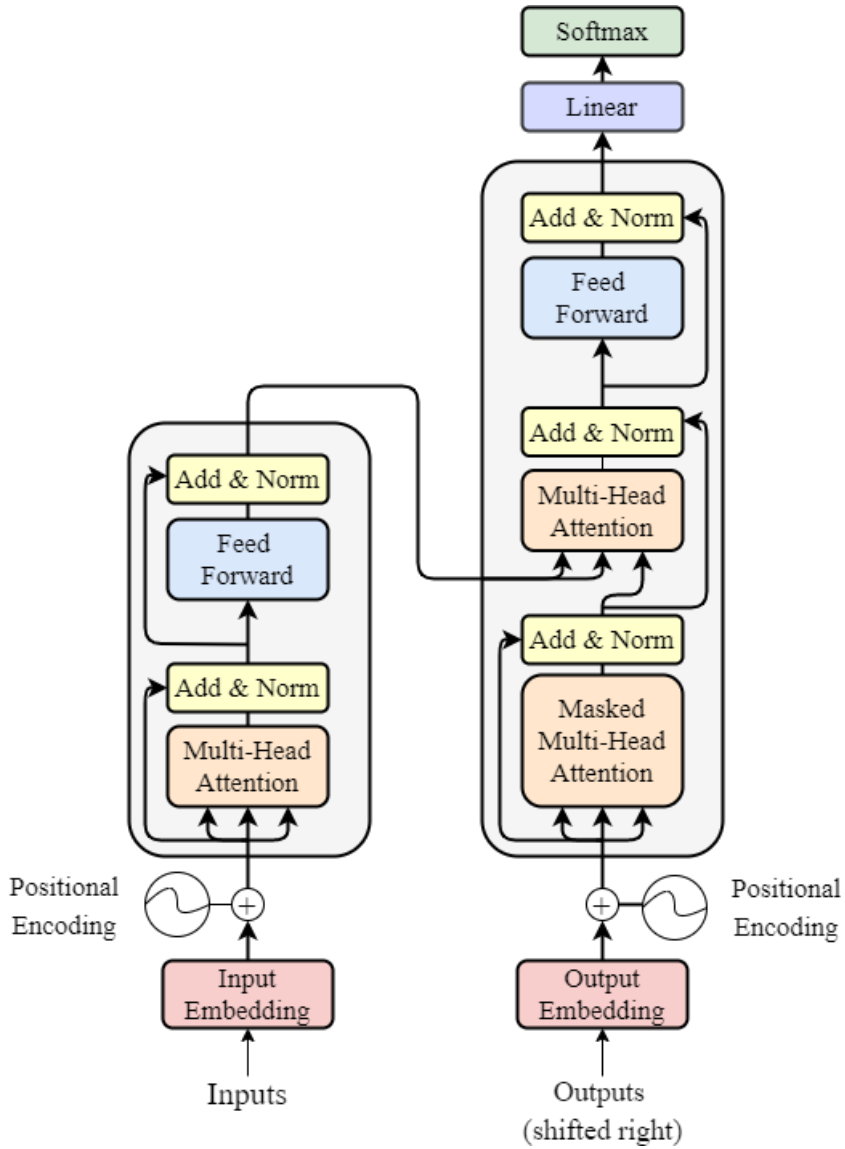
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

"A small fraction of the training cost"

[Vaswani et al., 2017]

The One

- BER**T** (Encoder-only)
- GP**T** (Decoder-only)
- Text-to-Text Transfer **T**ransformer (T5) (Encoder-Decoder)



References I


Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016).
Layer normalization.
arXiv preprint arXiv:1607.06450.


Bahdanau, D., Cho, K., and Bengio, Y. (2015).
Neural machine translation by jointly learning to align and translate.
In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.


Cheng, J., Dong, L., and Lapata, M. (2016).
Long short-term memory-networks for machine reading.
In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas. Association for Computational Linguistics.

References II


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
In *Advances in Neural Information Processing Systems*, pages 5998–6008.