Attention Is All You Need

Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

From: Google brain Google research

NIPS 2017

Slides prepared by: Chandra Thapa (July 2020)

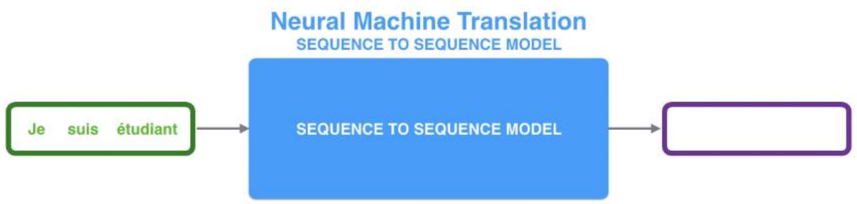
Applications of Attention mechanism

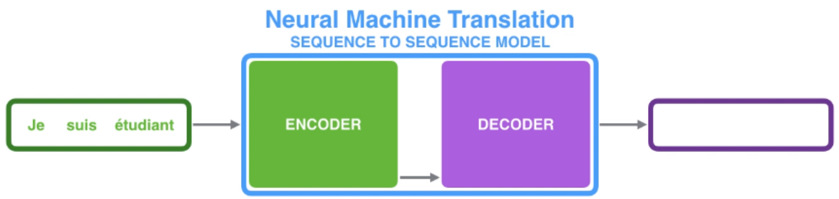
- Cyber-security:
  - Attention-Based Automated Feature Extraction for Malware Analysis [1]
  - Neural Malware Analysis with Attention Mechanism [2]
  - How to Make Attention Mechanisms More Practical in Malware Classification [3]
  - I-MAD: A Novel Interpretable Malware Detector Using Hierarchical Transformer [4]
- Phishing email detection
  - Phishing Email Detection Using Improved RCNN Model With Multilevel Vectors and Attention Mechanism [5]
- Fake news detection
  - Self Multi-Head Attention-based Convolutional Neural Networks for fake news detection [6]
  - HGAT: Hierarchical Graph Attention Network for Fake News Detection [7]

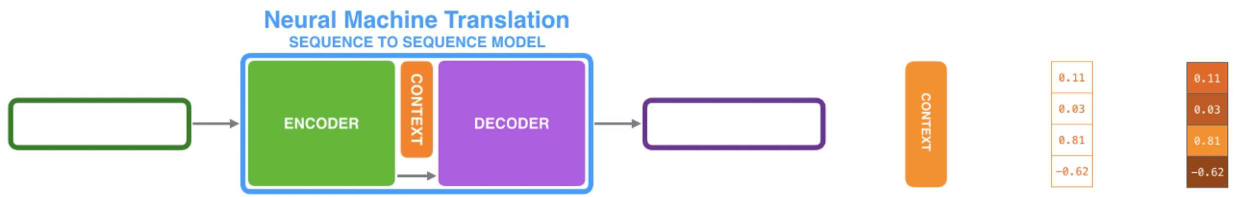
[1] https://www.mdpi.com/1424-8220/20/10/2893
[2] https://www.sciencedirect.com/science/article/pii/S0167404819300264
[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8876839
[4] https://arxiv.org/abs/1909.06865
[5] https://ieeexplore.ieee.org/abstract/document/8701426
[6] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0222713
[7] https://arxiv.org/abs/2002.04397

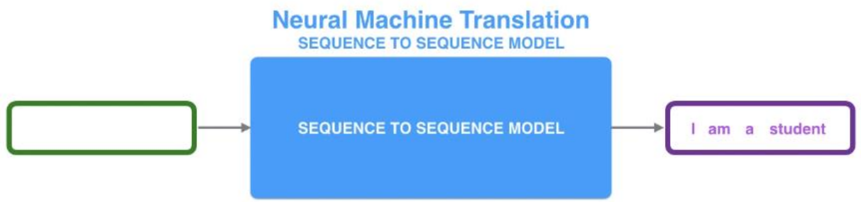
Background

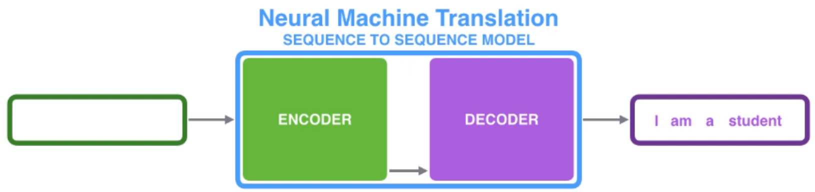
- In machine translation: Sequence-to-sequence model





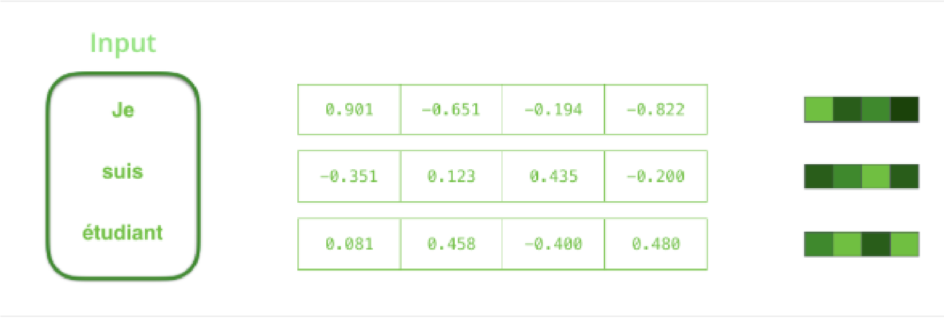




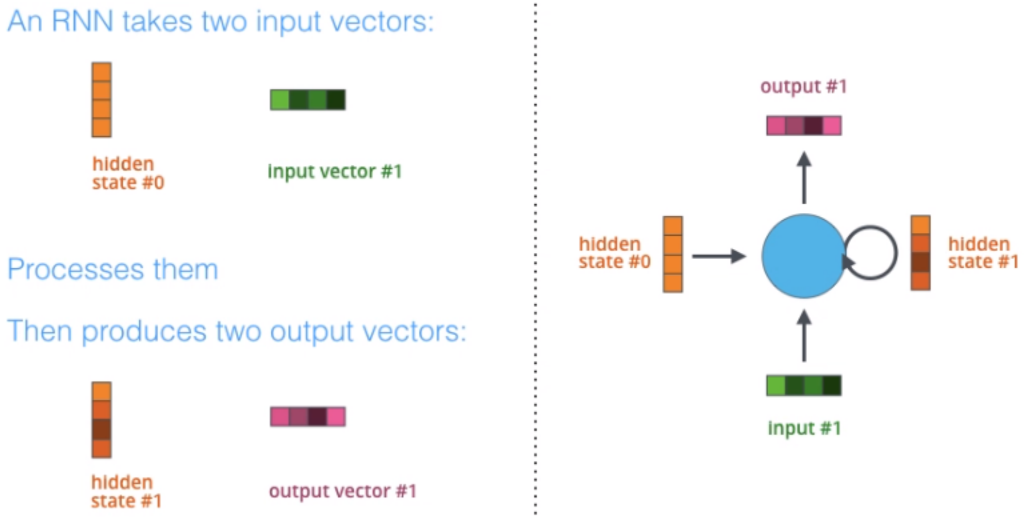


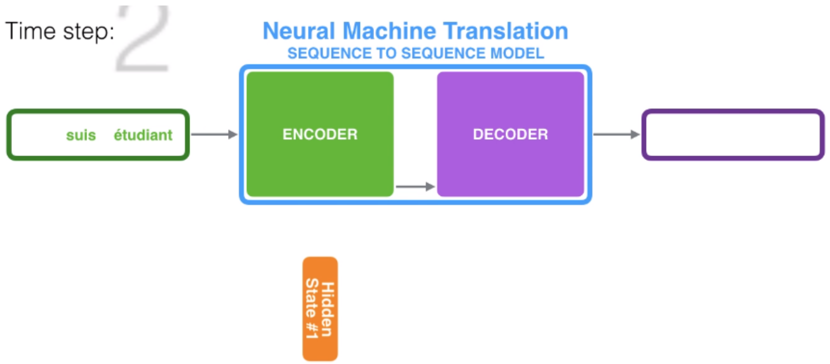
Reference : https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

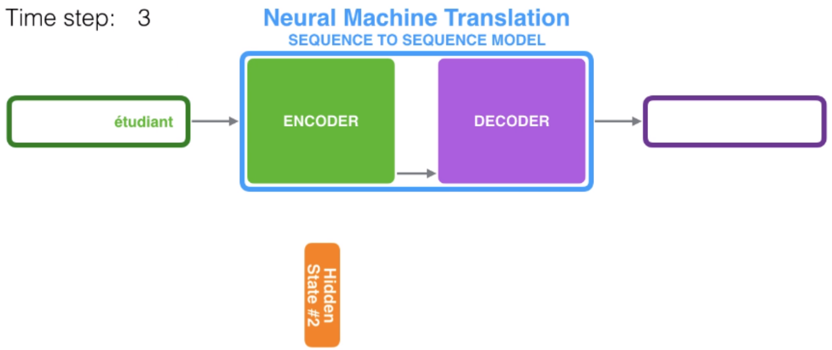
- Word embeddings

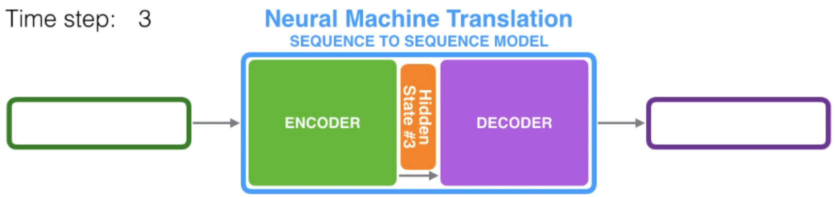


- A RNN takes two inputs at each time step:
  - Input (e.g., one word of the input sentence, if encoder)
  - Hidden state



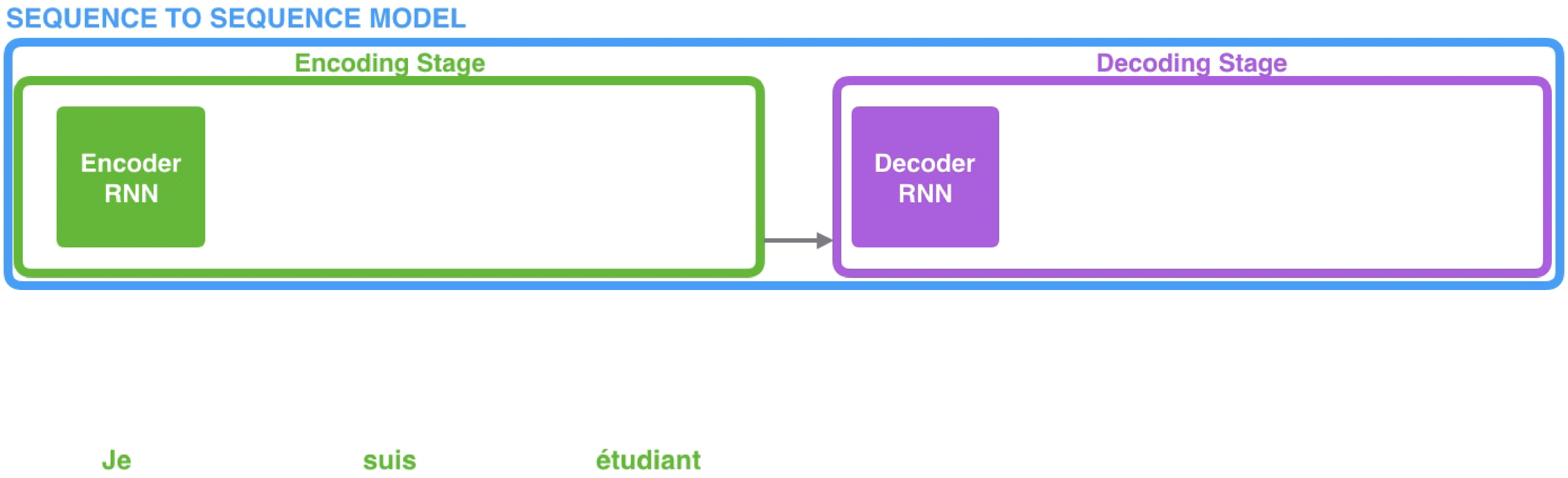






Reference : https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

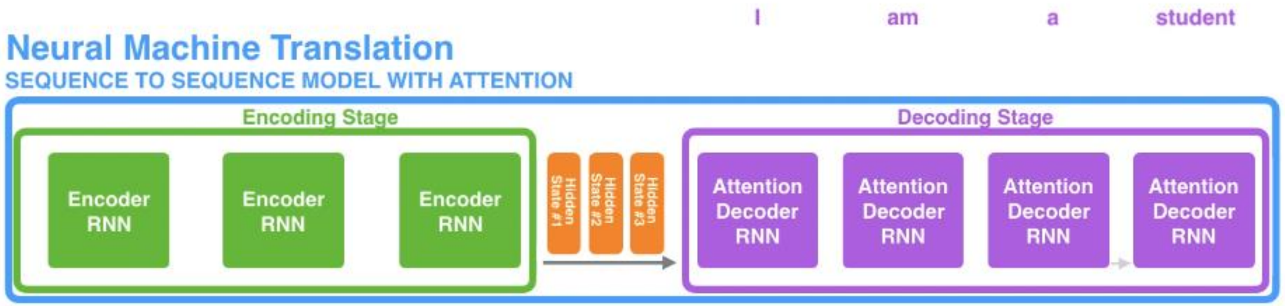
Neural Machine Translation



Reference : https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

- Challenging if long sequence – context vector is a bottleneck
- (Classic) Attention is introduced [1,2] – it allows the model to focus on the relevant parts of the input sequences as needed

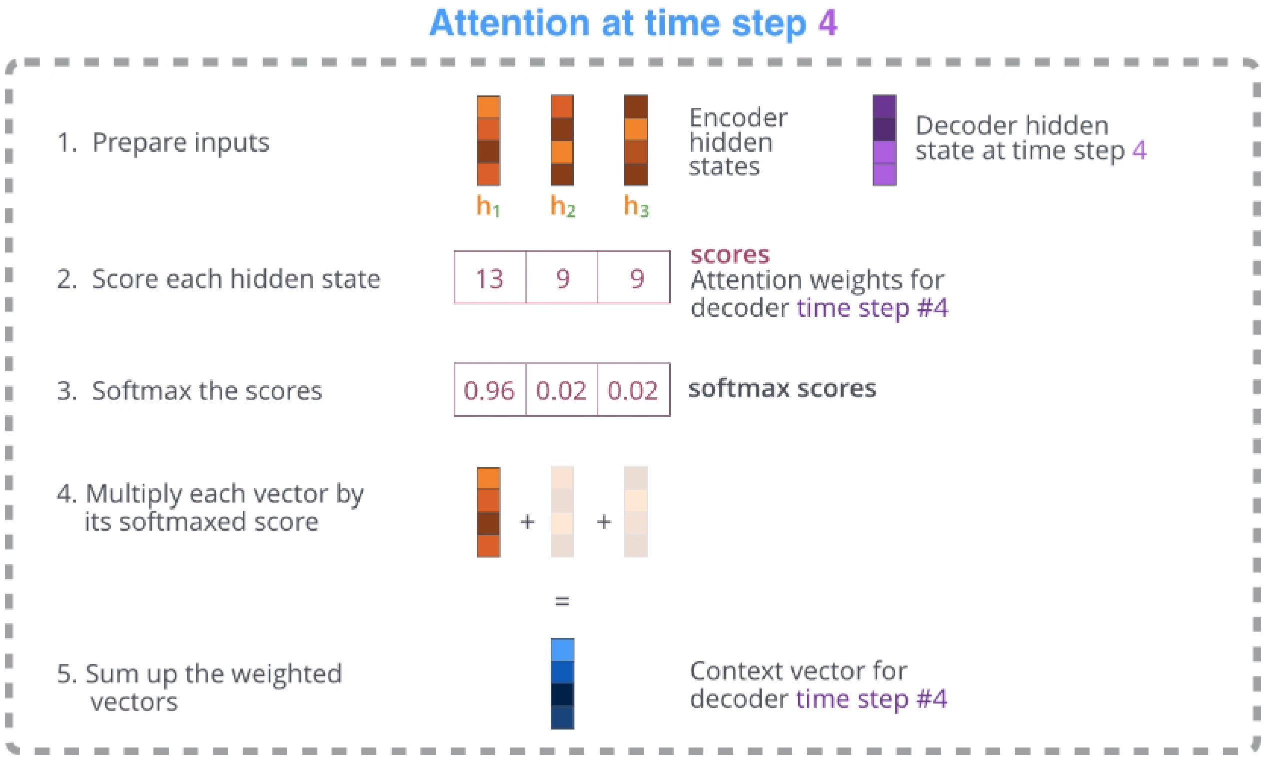
- An attention model differs from a classic sequence-to-sequence model in the following ways:
  - the encoder passes *all* the hidden states to the decoder



[1] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, **Neural Machine Translation by Jointly Learning to Align and Translate,** https://arxiv.org/abs/1409.0473

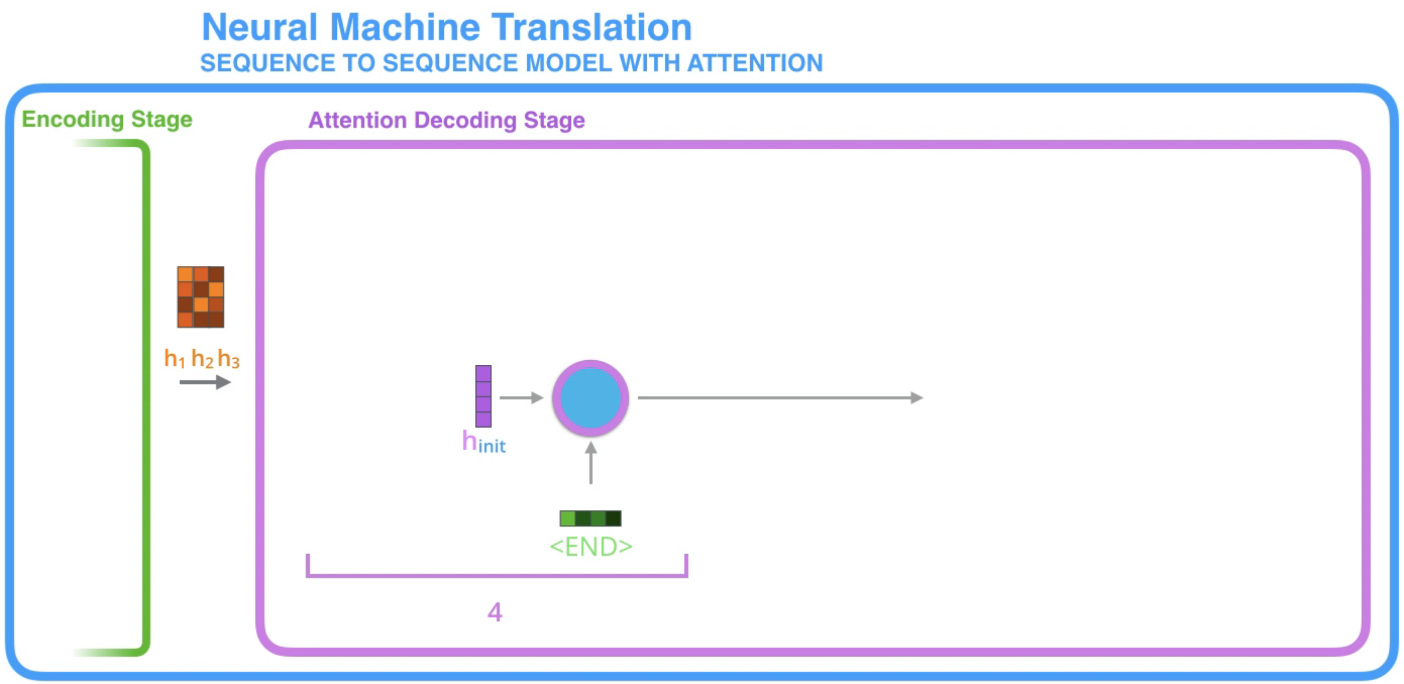
[2] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, **Effective Approaches to Attention-based Neural Machine Translation,** https://arxiv.org/abs/1508.04025

- Attention decoder does the following:
  - Look at the set of encoder hidden states it received – each encoder hidden states is most associated with a certain word in the input sentence
  - Give each hidden states a score
  - Multiply each hidden states by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores



Reference : https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

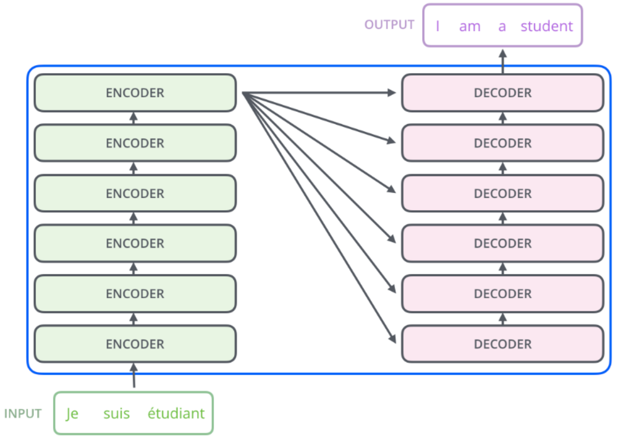
- How the attention process works?
  - The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
  - The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.
  - Attention Step: Use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
  - Concatenate h4 and C4 into one vector.
  - Pass this vector through a feedforward neural network (one trained jointly with the model).
  - The output of the feedforward neural networks indicates the output word of this time step.
  - Repeat for the next time steps

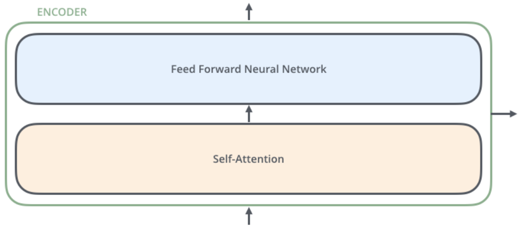


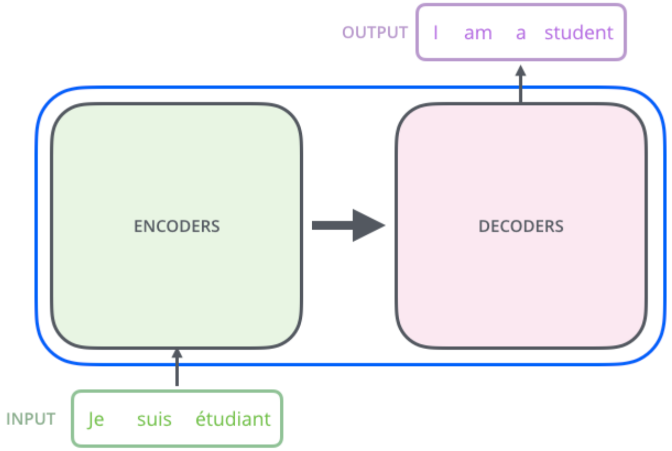
Reference : https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

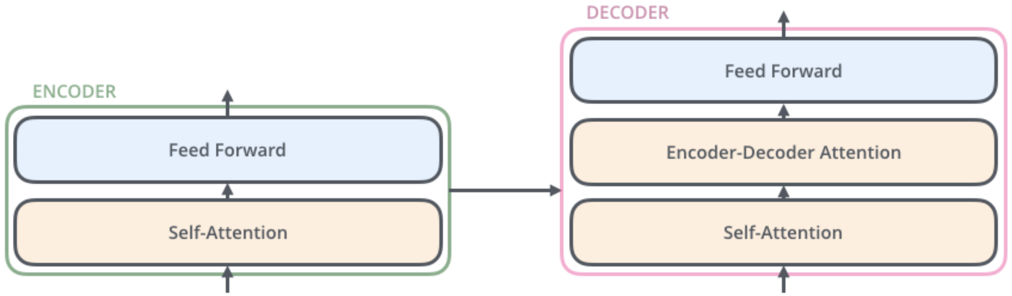
Transformer

High-level look





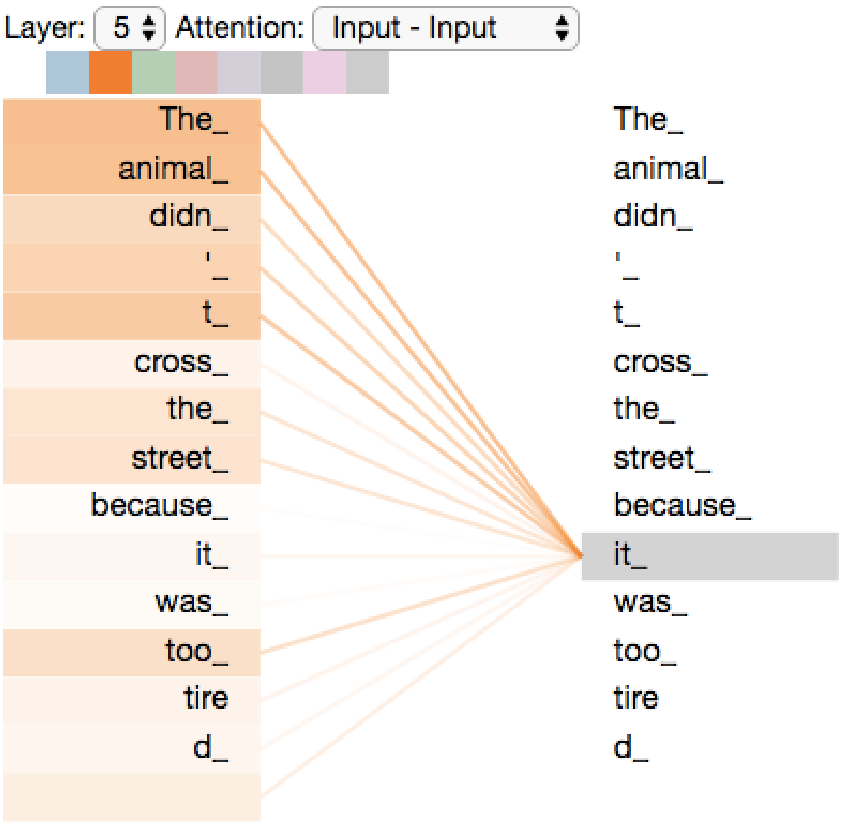




Reference : https://jalammar.github.io/illustrated-transformer/

Self-Attention at a high level

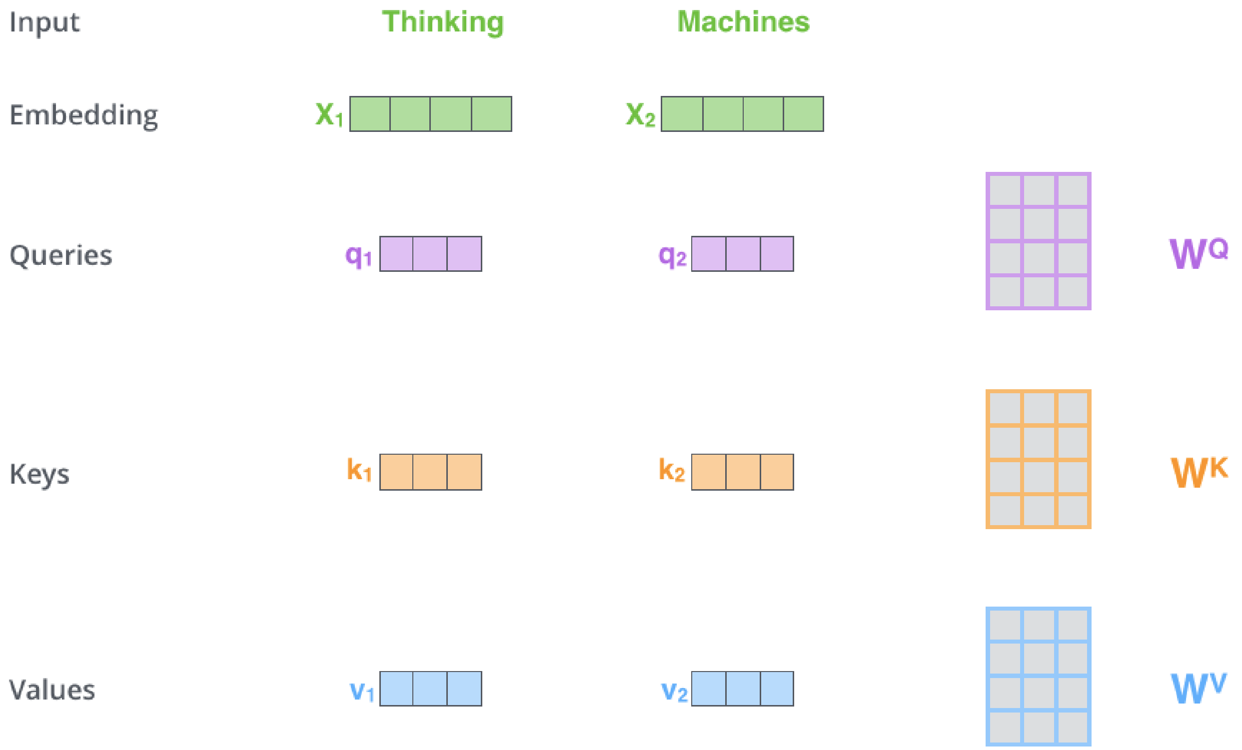
- Translating: “The animal didn't cross the street because it was too tired”
- What does “it” in this sentence refer to?
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.



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

Self-attention

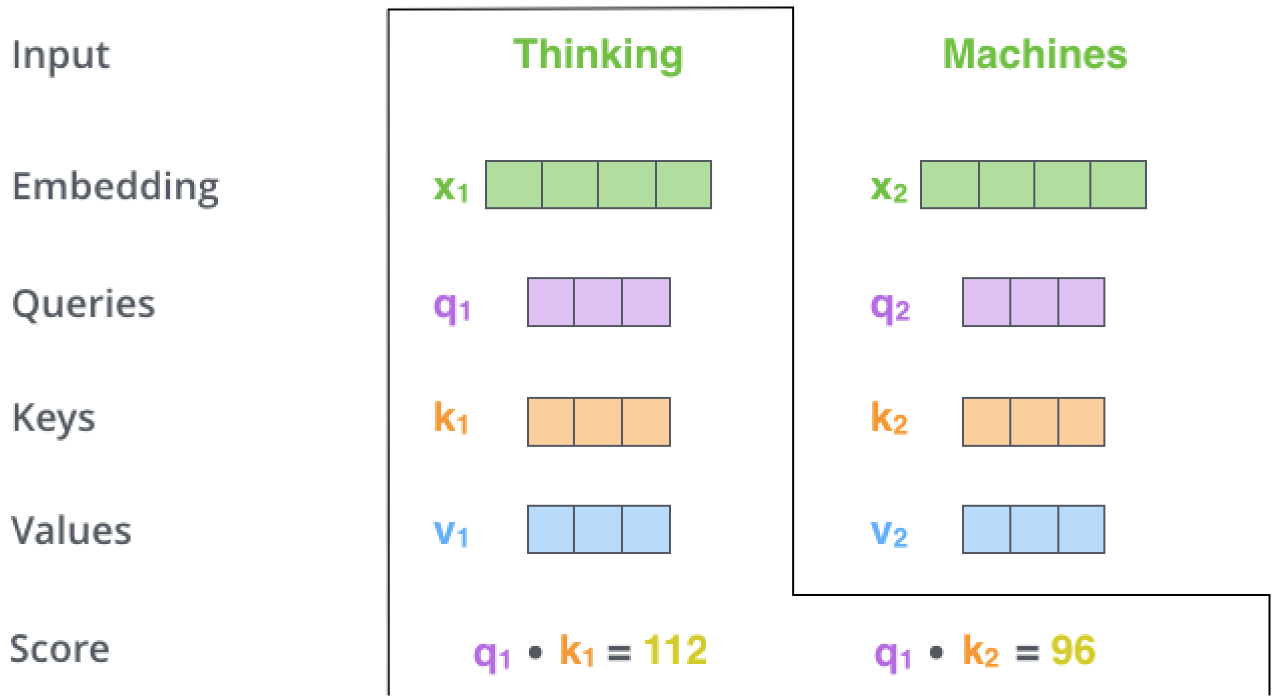
- Q, K , V are created for each of the encoder's input vector (i.e., embedding of each word)
- These vectors are created by multiplying the embedding by three matrices W^Q, W^K and W^V– trained during the training process



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

Self-attention

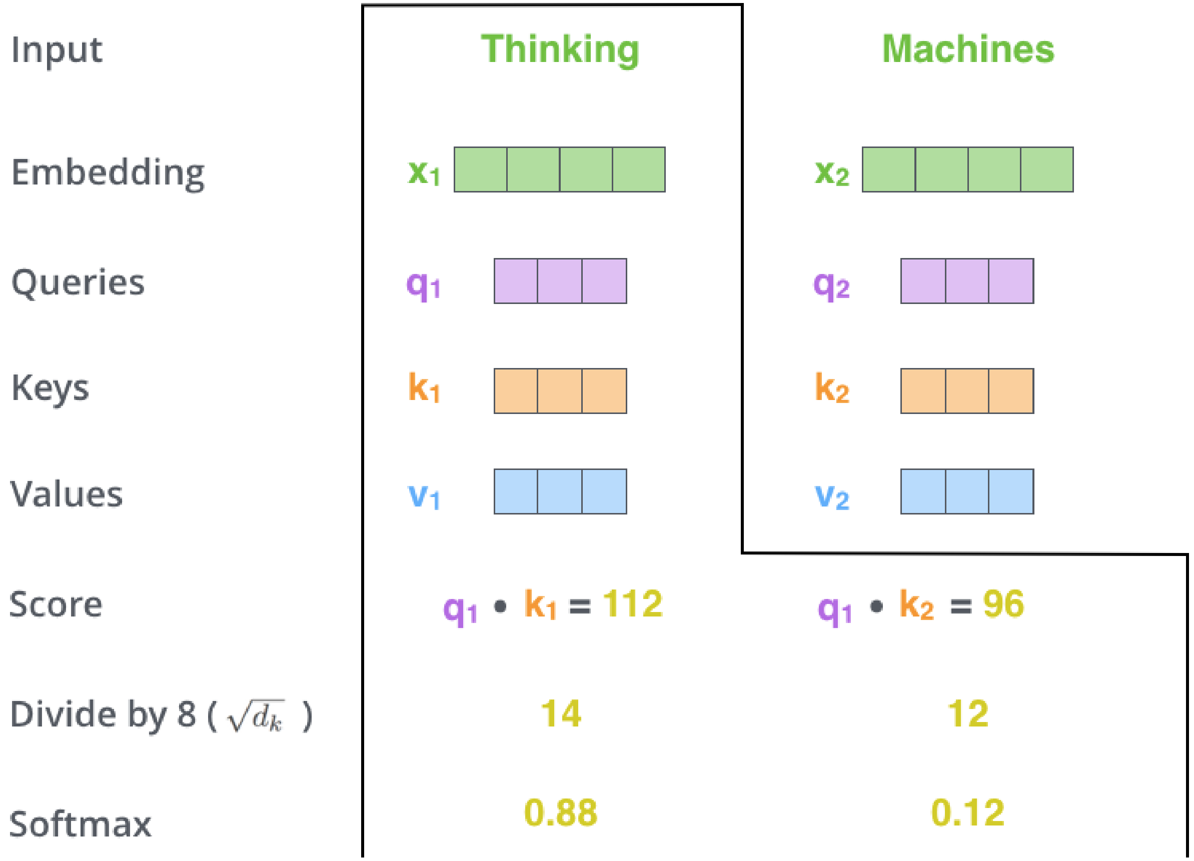
- Calculate a score
    - calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.
    - So if processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.



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

Self-attention

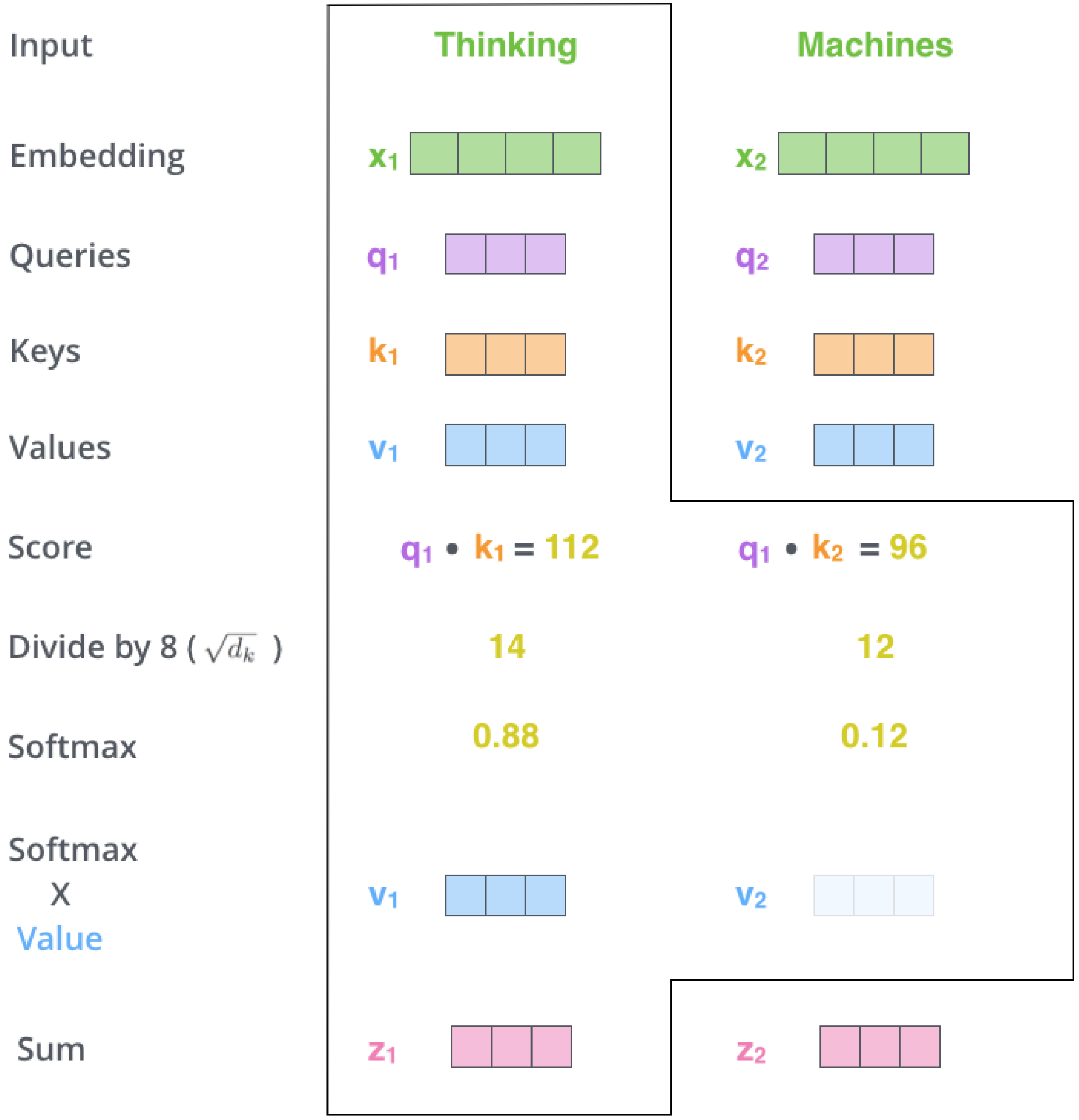
- Divide the score by the square root of the dimension of the key vectors
- Take softmax operation – determines how much each word will be expressed at this position



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

Self-attention

- Multiply each value by the sofmax score.
- Sum up the weighted value of vectors – this is attention value.

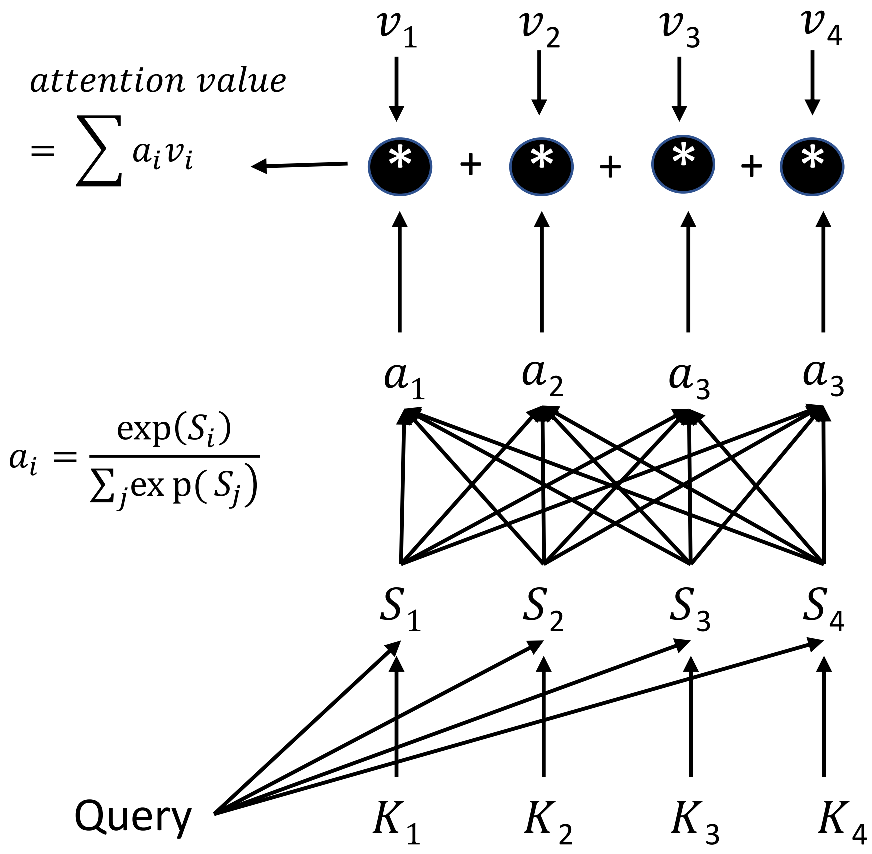


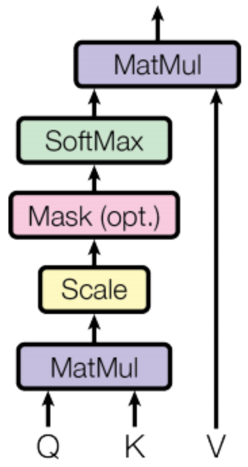
Reference : https://jalammar.github.io/illustrated-transformer/

Attention mechanism

- Mimics the retrieval of a **value** v_i for a **query** q_i based on a **key** k_i in database.

$attention(q, \boldsymbol{k}, \boldsymbol{v}) = \sum_i similarity(q, k_i) \times v_i$





$$S_i = F(q, k_i) = \begin{cases} q^T k_i, & dot\ product\ similarity \\ \frac{q^T k_i}{\sqrt{d}}, & scaled\ dot\ product \\ q^T W k_i, & general\ dot\ product \end{cases}$$

Ref: https://cs.uwaterloo.ca/~ppoupart/teaching/cs480-spring19/slides/cs480-lecture19.pdf

Multihead attention

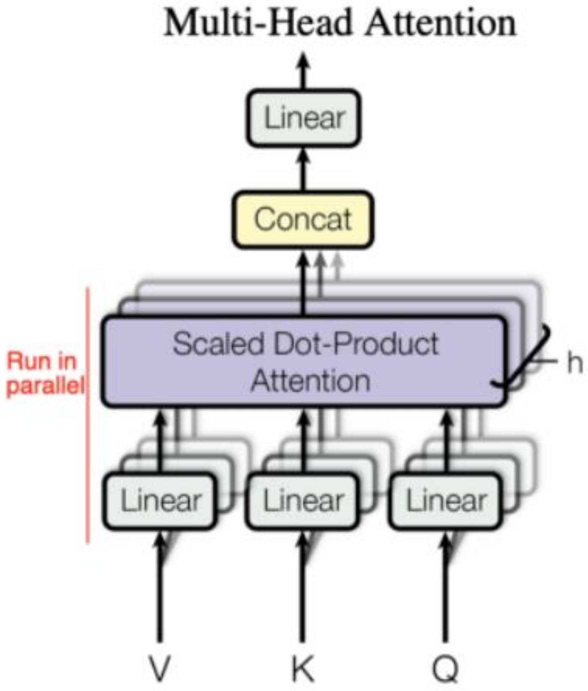
- Compute multiple attentions per query with different weights

$$multihead(Q,K,V) = W^O concat(head_1, head_2, \dots, head_h)$$
$$head_i = attention(W_i^Q Q, W_i^K K, W_i^V V)$$
$$attention(Q,K,V) = softmax\left(\frac{Q^T K}{\sqrt{d_k}}\right)V$$

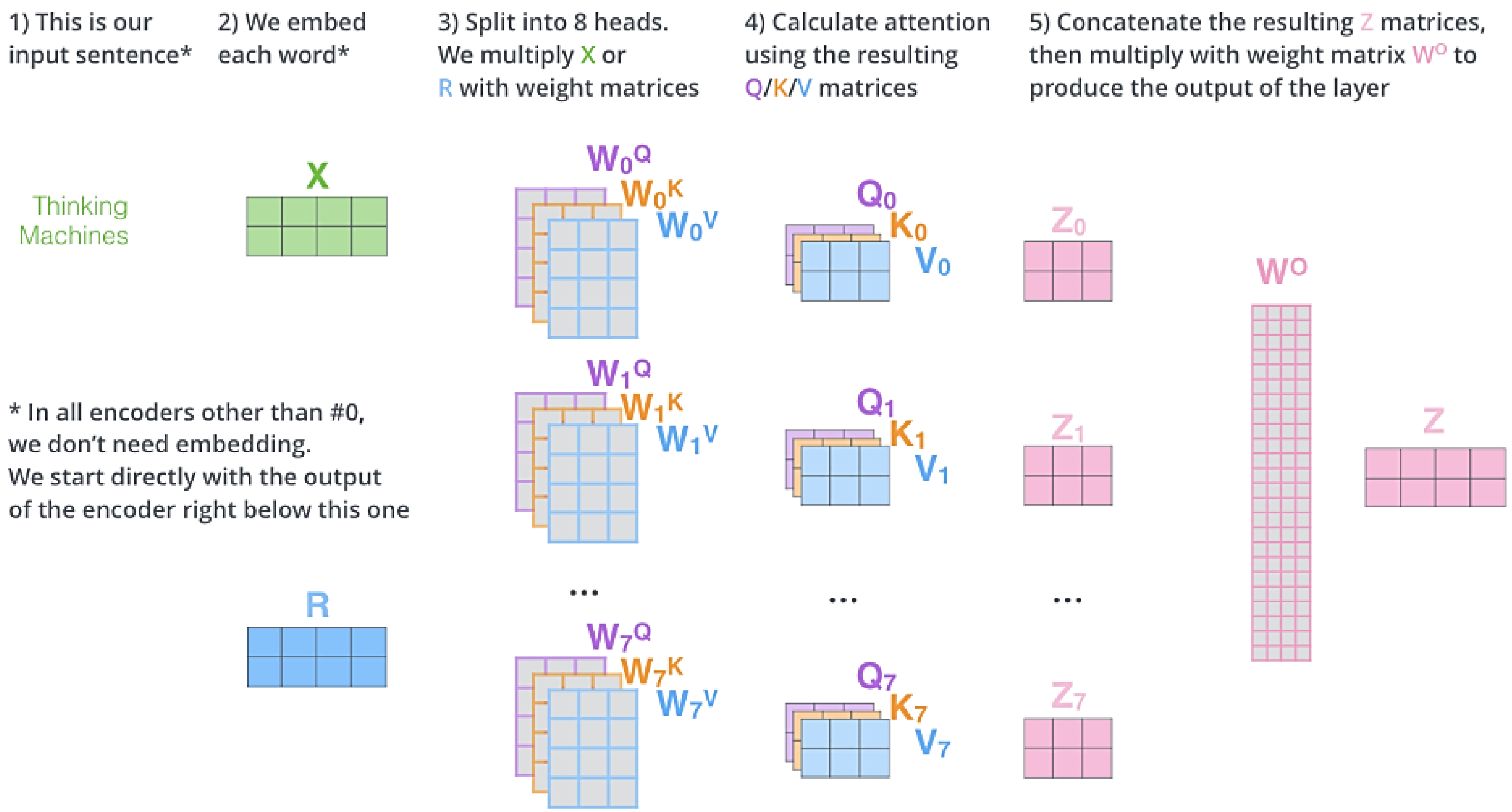
- Masked multi-head attention: multi-head where some values are masked (i.e., probabilities of masked values are nullified to prevent them from being selected).
- When decoding, an output value should only depend on previous outputs (not future outputs). Hence we mask future outputs.

$$attention(Q,K,V) = softmax\left(\frac{Q^T K}{\sqrt{d_k}}\right)V$$
$$maskedAttention(Q,K,V) = softmax\left(\frac{Q^T K + M}{\sqrt{d_k}}\right)V$$

where M is a mask matrix of 0's and $-\infty$'s

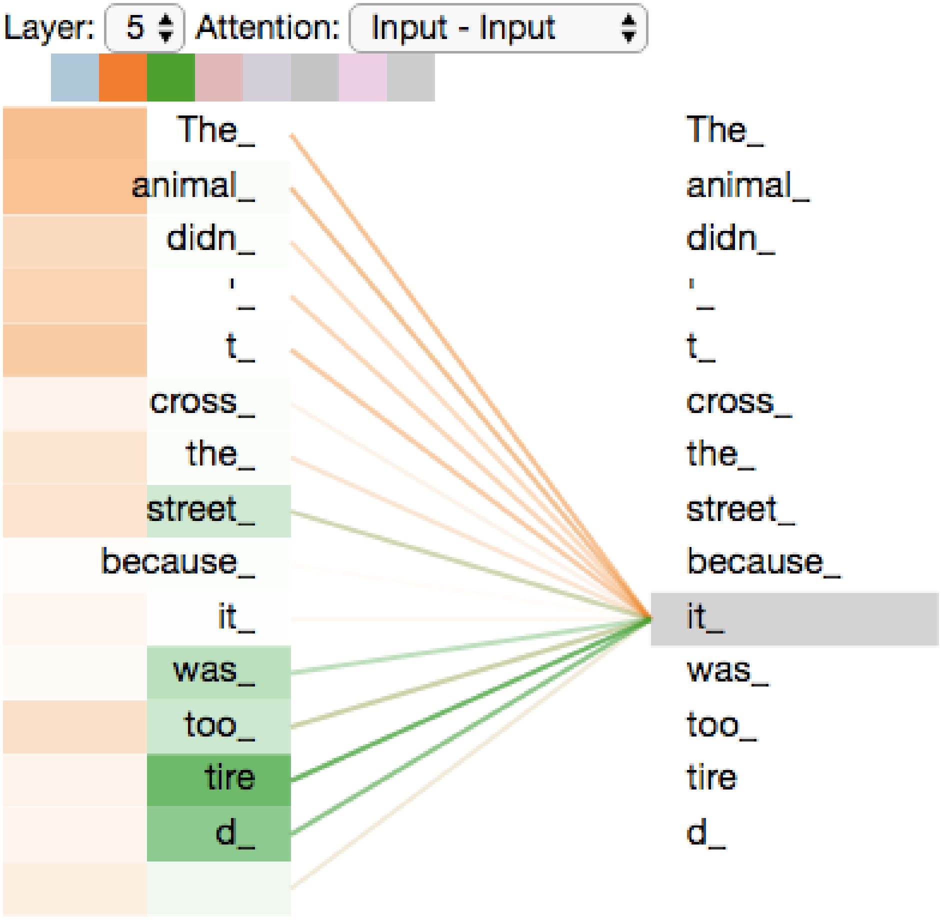


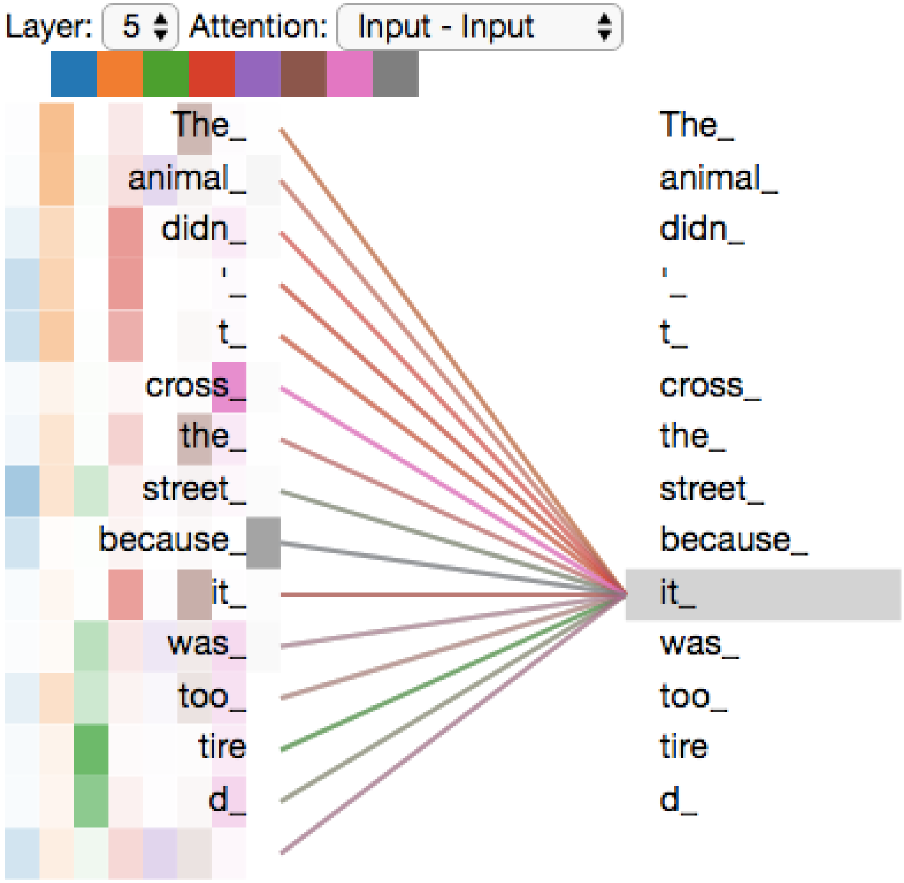
Multihead attention



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

Multihead attention





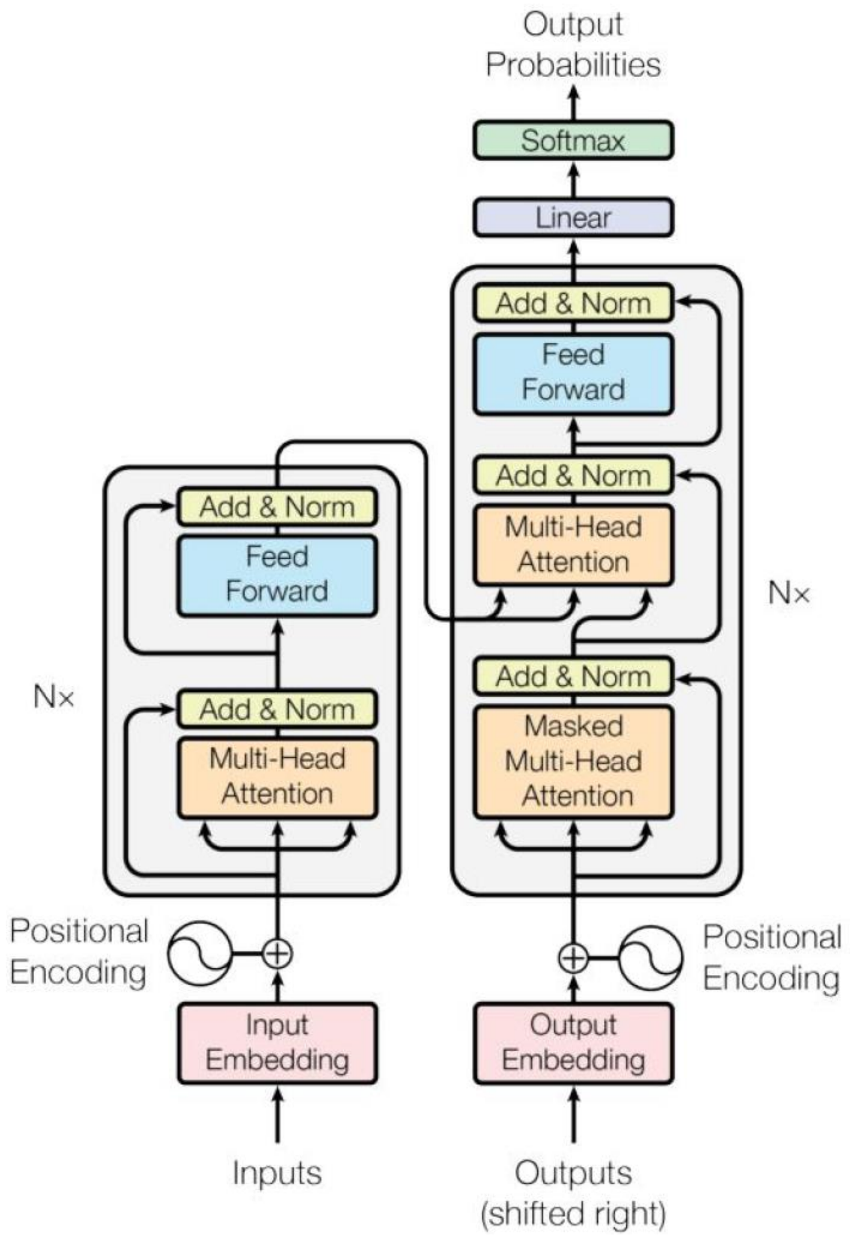
Reference : https://jalammar.github.io/illustrated-transformer/

Transformer architecture

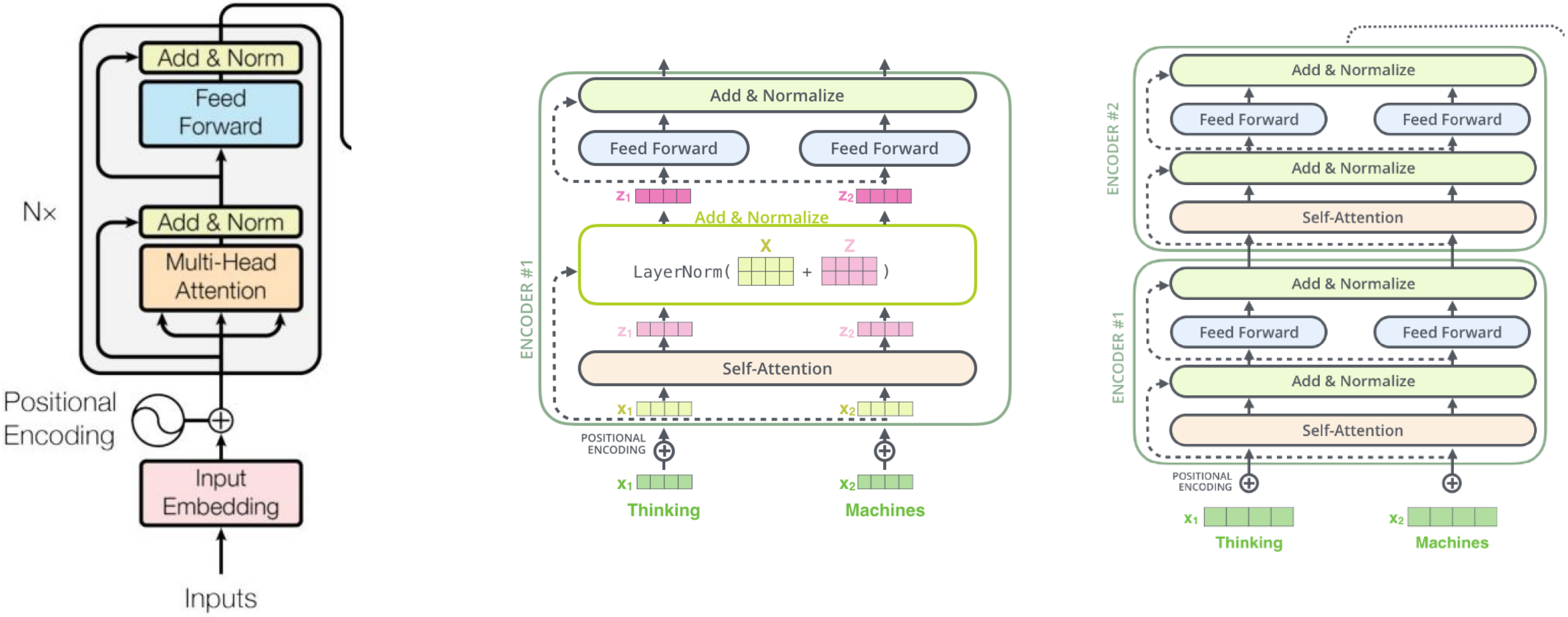
- Encoder-decoder based on attention (no recurrence)

- Positional encoding:
  - use sine and cosine functions of different frequencies to encode the position information

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

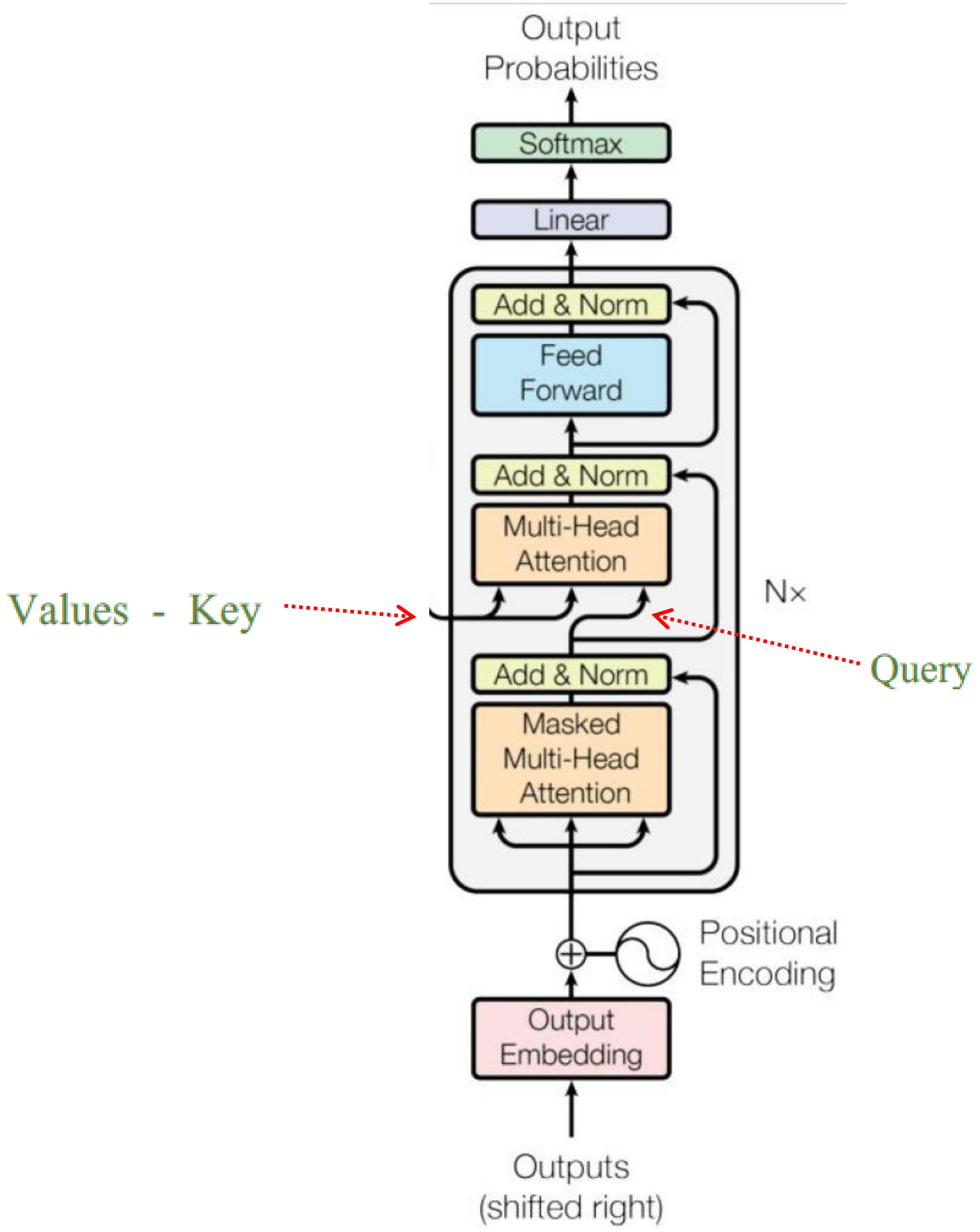


Encoder

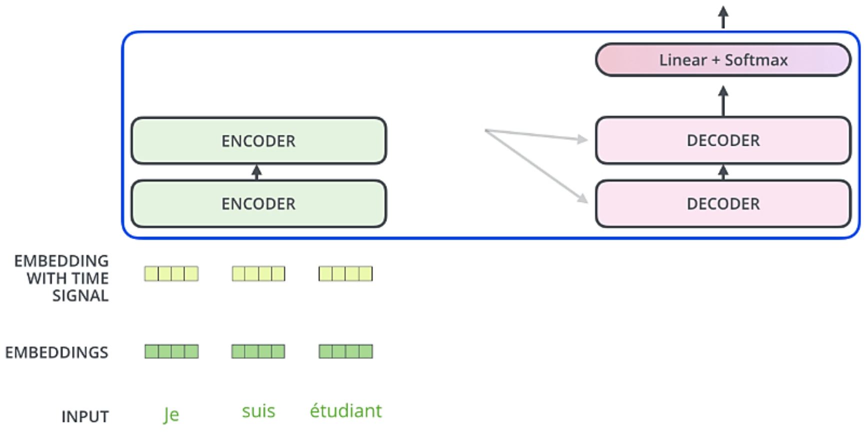


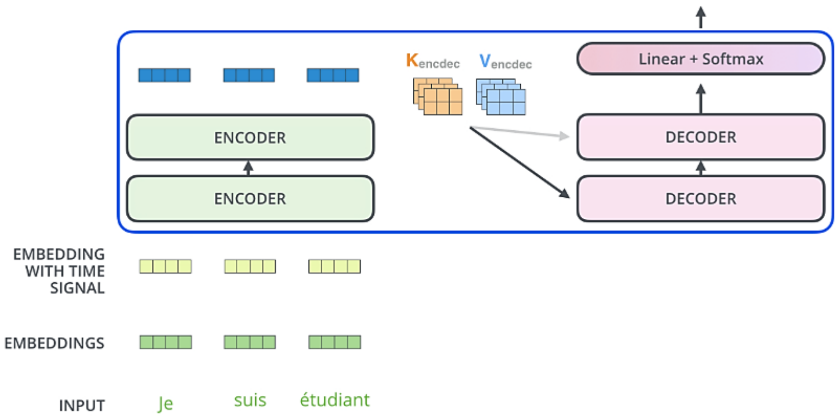
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
LayerNorm(X + Z)
Self-Attention
x1
x2
POSITIONAL ENCODING
Thinking
Machines
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
x2
Thinking
Machines

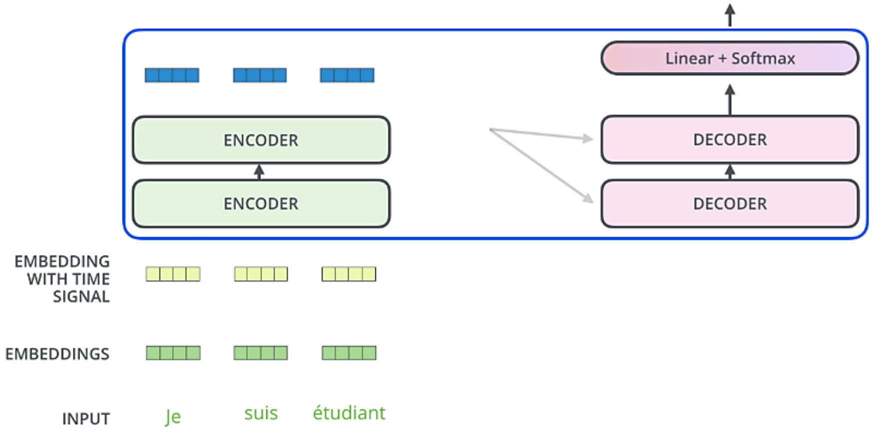
Decoder

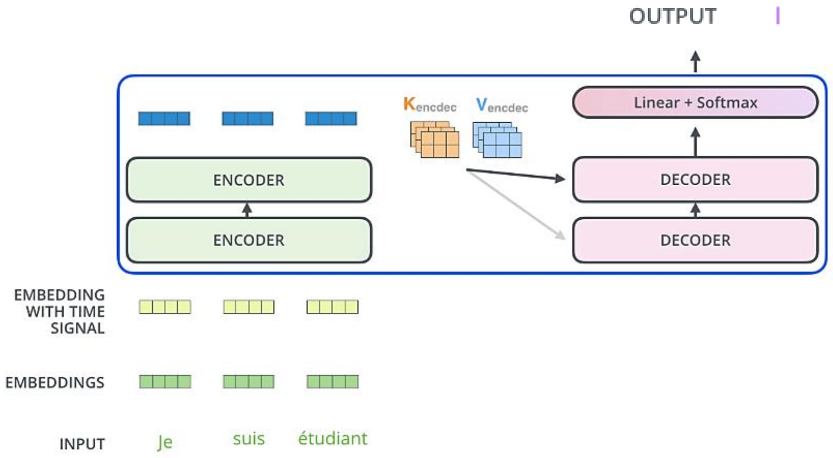

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Values - Key
Nx
Query
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

Decoder



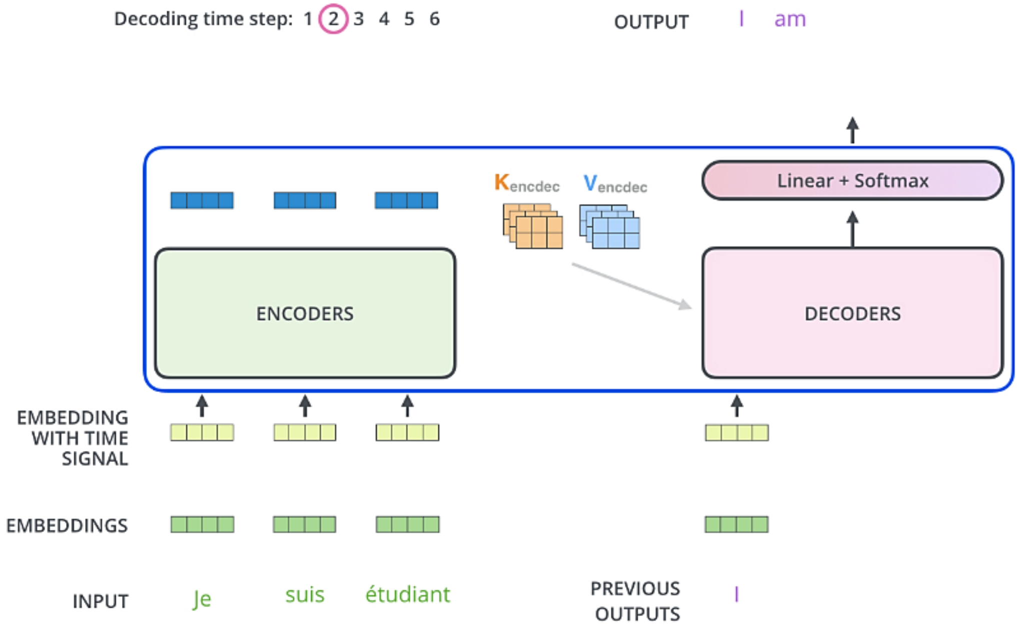


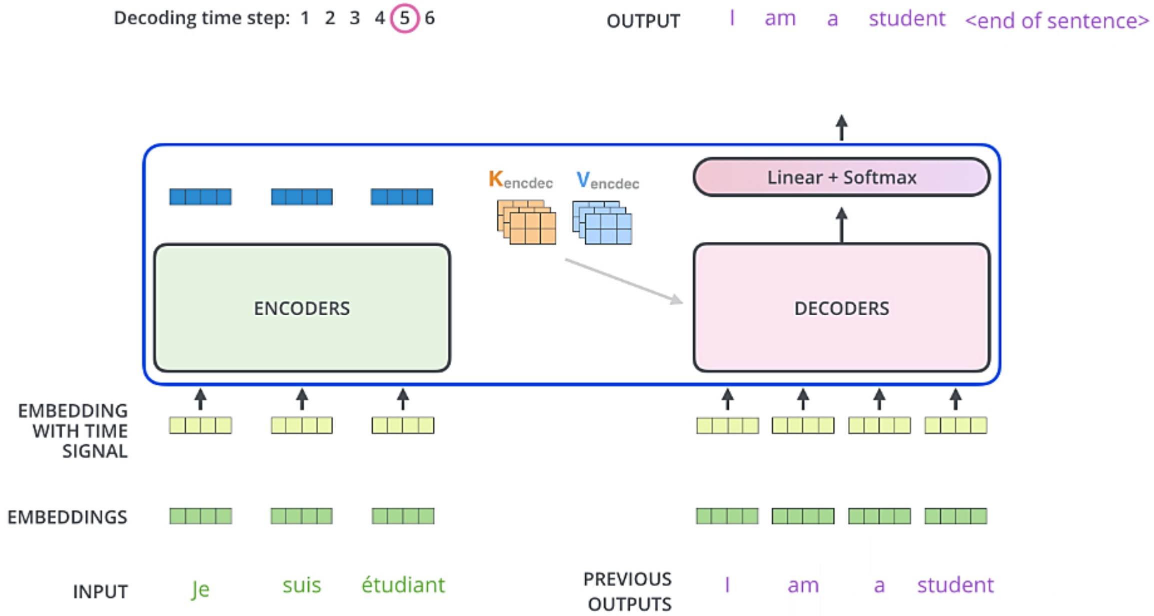




Reference : https://jalammar.github.io/illustrated-transformer/

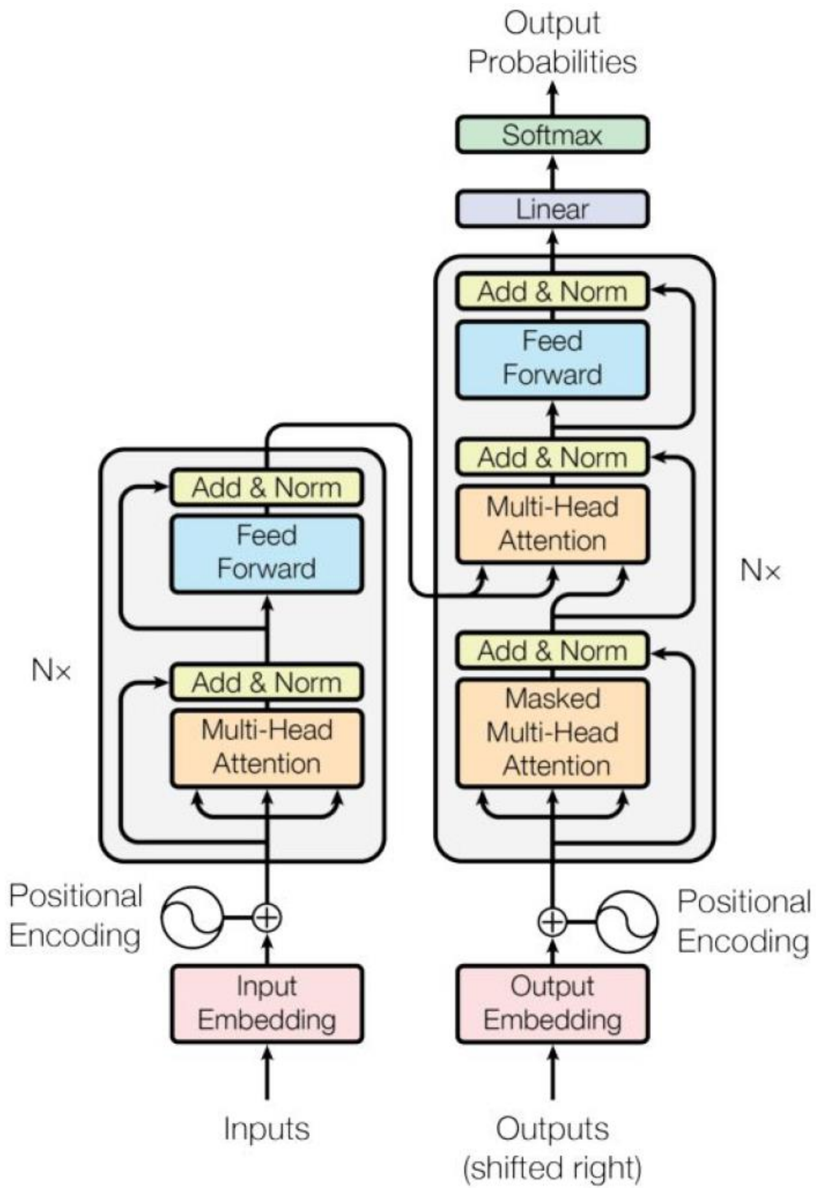
Decoder





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

Transformer



Results

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Results

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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Variants of Transformers

- BERT
- XLNet
- RoBERTa
- GPT-2
- GPT-3
- ALBERT

Thank you for your attention!!