

STOR566: Introduction to Deep Learning

Lecture 17: Transformer

Yao Li
UNC Chapel Hill

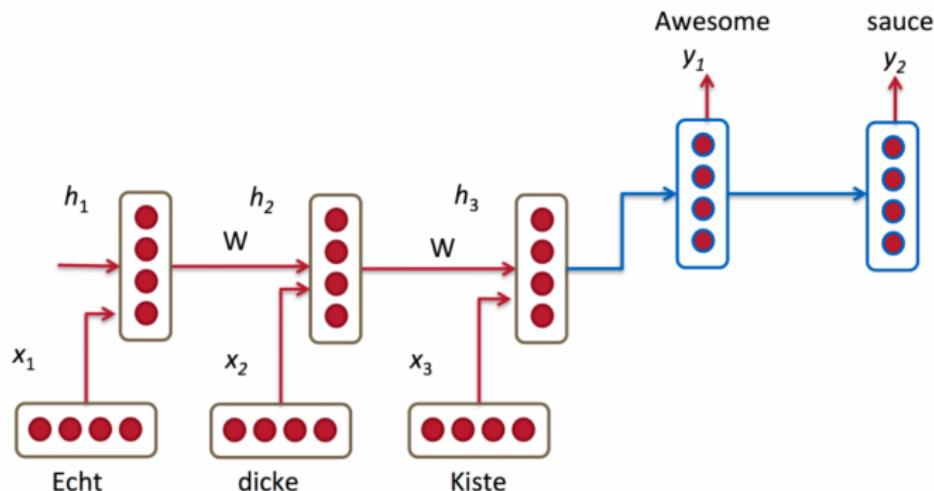
Oct 27, 2022

Materials are from *Deep Learning (UCLA)*

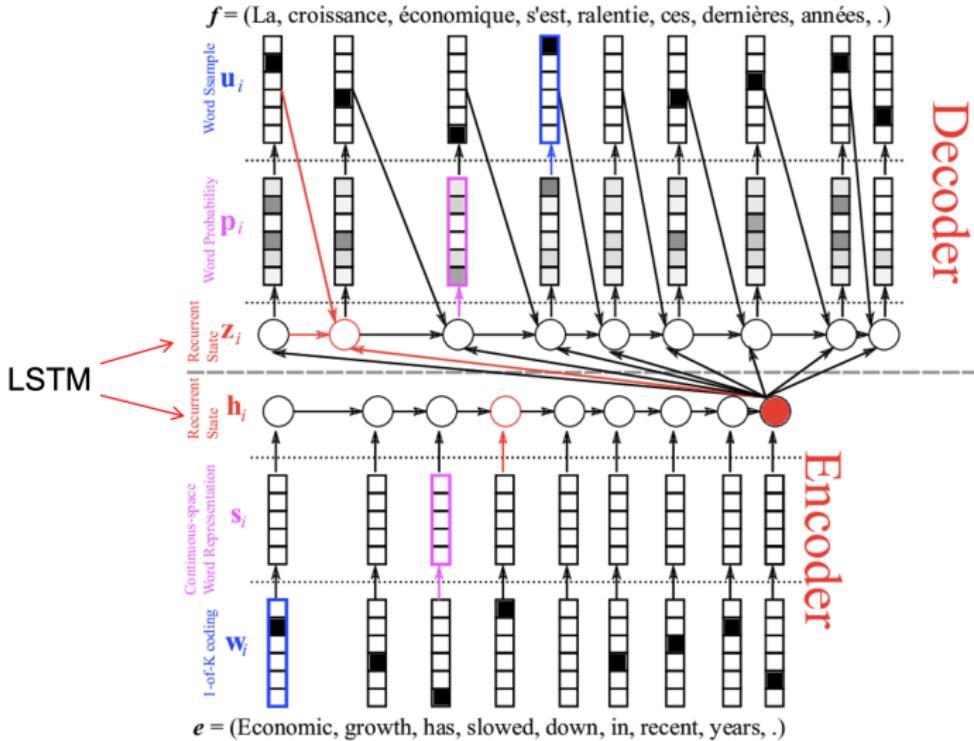
Neural Machine Translation

Neural Machine Translation (NMT)

- Get the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
 - Encoder: Use (RNN/LSTM) to encode the input sentence into a latent vector
 - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector

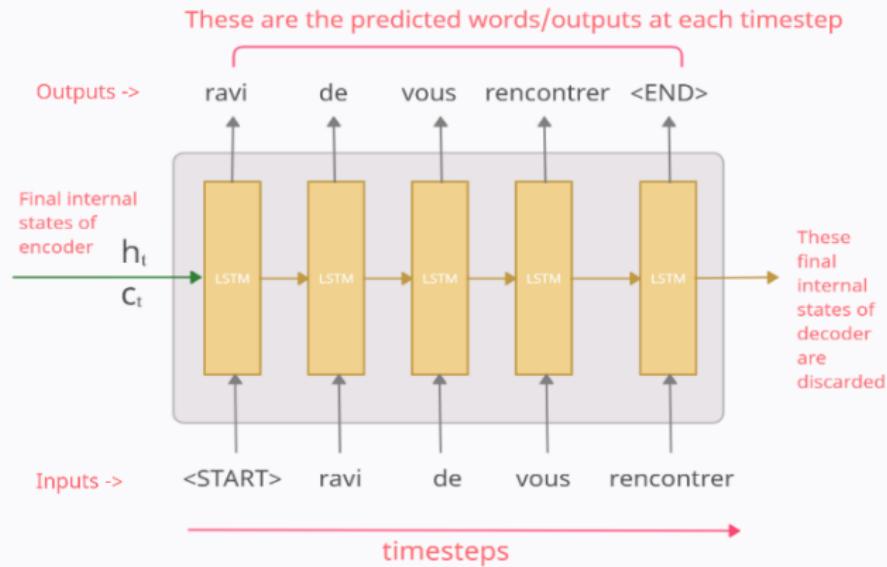


Neural Machine Translation



RNN: Neural Machine Translation

- Start input of the decoder?
- When to stop?



picture from <https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b>

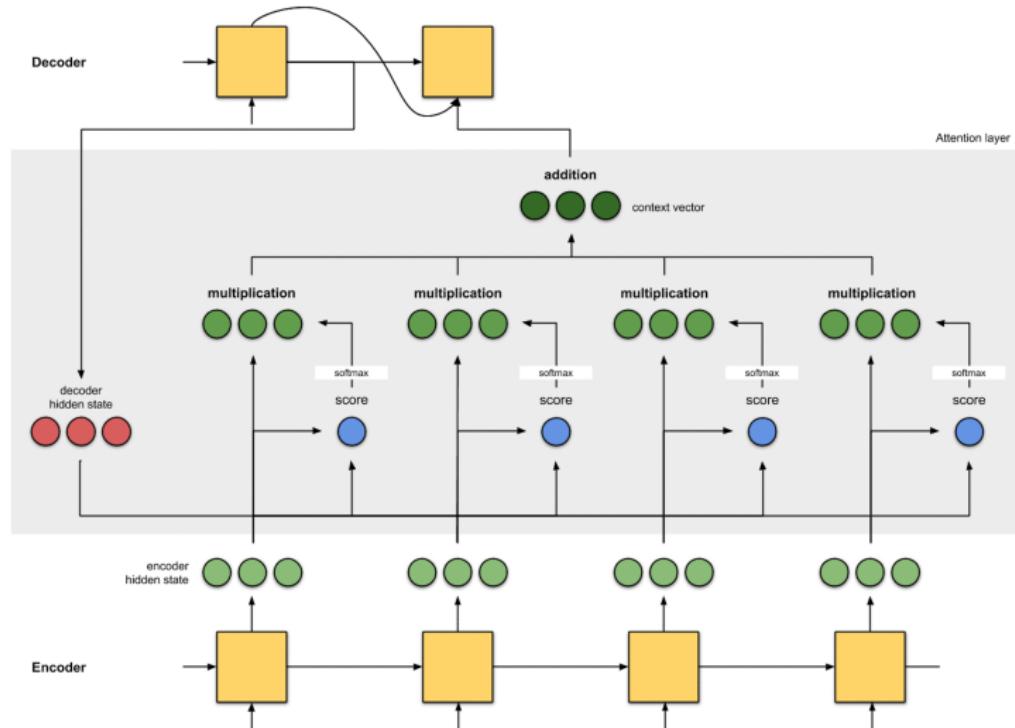
Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let \mathbf{u} be the current decoder latent state $\mathbf{v}_1, \dots, \mathbf{v}_n$ be the latent state for each input word
- Compute the weight of each state by

$$\mathbf{p} = \text{Softmax}(\mathbf{u}^T \mathbf{v}_1, \dots, \mathbf{u}^T \mathbf{v}_n)$$

- Compute the context vector by $V\mathbf{p} = p_1 \mathbf{v}_1 + \dots + p_n \mathbf{v}_n$

Attention in NMT

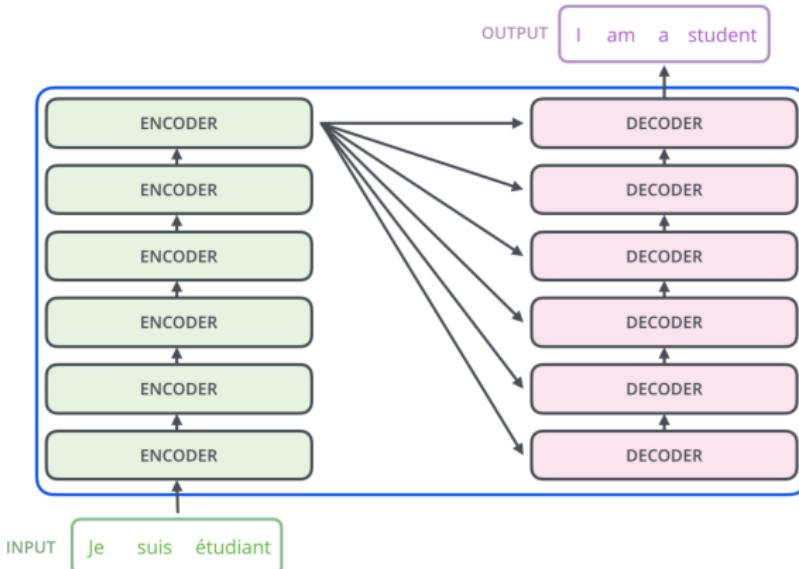


(Figure from <https://towardsdatascience.com/>)

Transformer

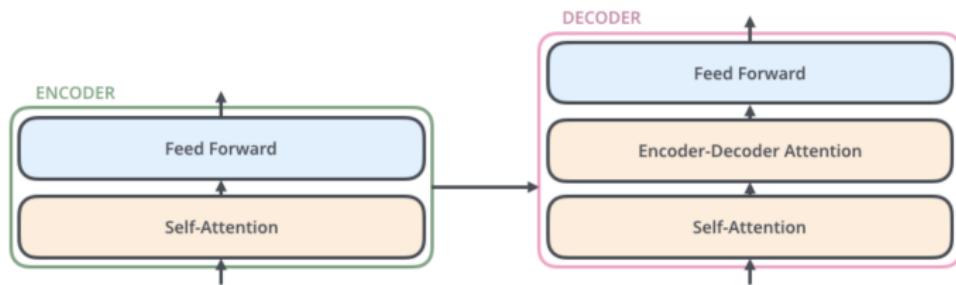
Transformer

- An architecture that relies entirely on attention without using CNN/RNN
- Proposed in “Attention Is All You Need” (Vaswani et al., 2017)
- Initially used for neural machine translation



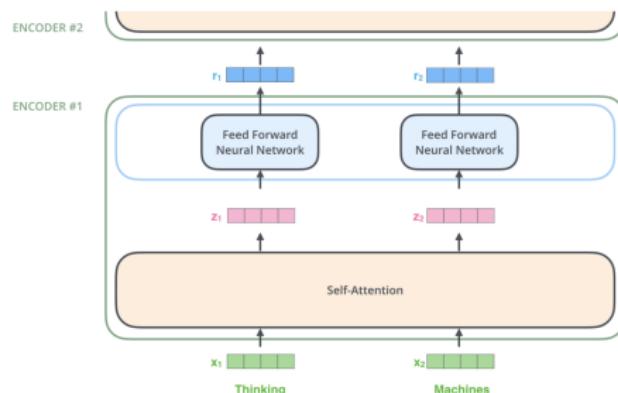
Encoder and Decoder

- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focuses on relevant parts of input sentences.



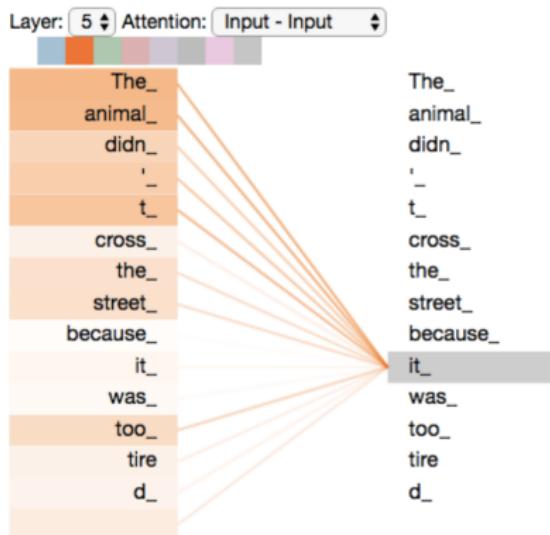
Encoder

- Each word has a corresponding “latent vector” (initially the word embedding for each word)
- Each layer of encoder:
 - Receive a list of vectors as input
 - Passing these vectors to a **self-attention** layer
 - Then passing them into a feed-forward layer
 - Output a list of vectors



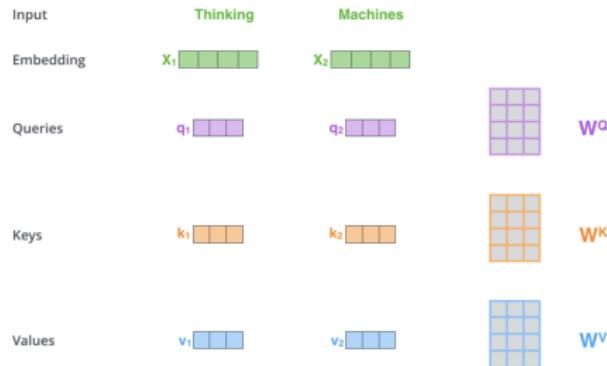
Self-attention layer

- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentences



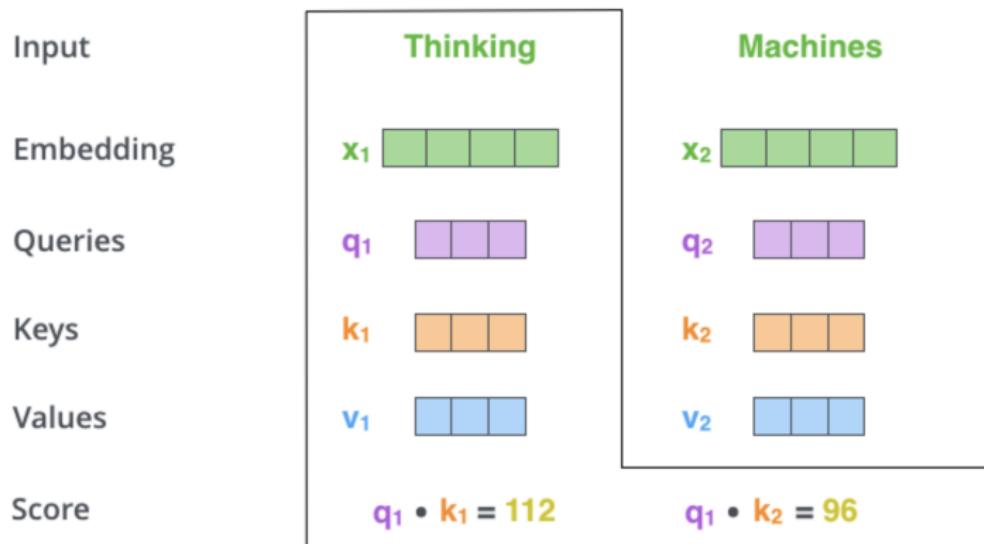
Self-attention layer

- Input latent vectors: x_1, \dots, x_n
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i , compute
 - Query vector: $q_i = x_i W^Q$
 - Key vector: $k_i = x_i W^K$
 - Value vector: $v_i = x_i W^V$



Self-attention layer

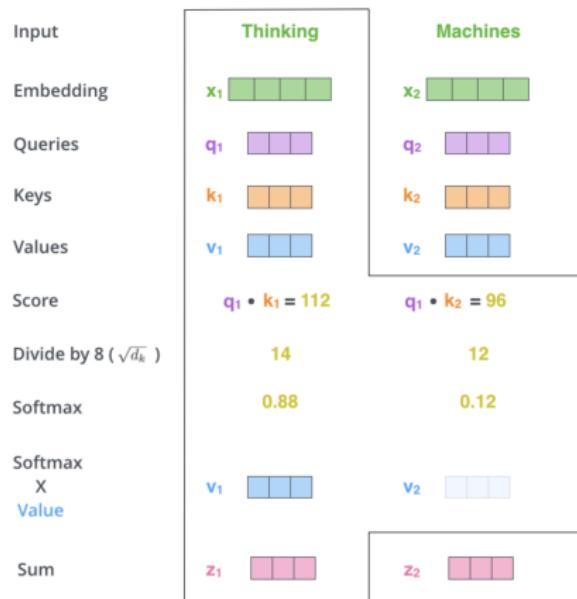
- For each word i , compute the scores to determine how much focus to place on other input words
 - The **attention** score for word j to word i : $q_i^T k_j$



Self-attention layer

- For each word i , the output vector

$$\sum_j s_{ij} \mathbf{v}_j, \quad \mathbf{s}_i = \text{softmax}(\mathbf{q}_i^T \mathbf{k}_1, \dots, \mathbf{q}_i^T \mathbf{k}_n)$$



Matrix form

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V, \quad Z = \text{softmax}(QK^T)V$$

$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^Q \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} Q \\ \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \end{matrix}$$

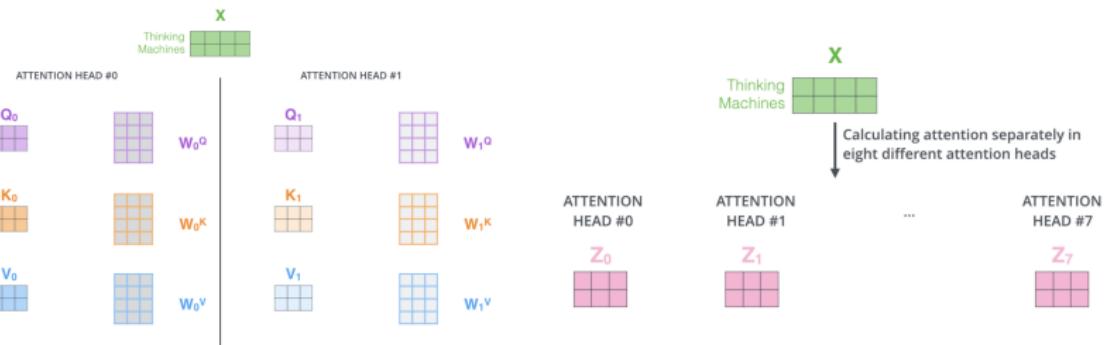
$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^K \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} K \\ \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^V \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} V \\ \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \end{matrix}$$

$$\text{softmax}\left(\frac{\begin{matrix} Q \\ \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \end{matrix} \times \begin{matrix} K^T \\ \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \end{matrix}}{\sqrt{d_k}}\right) \begin{matrix} V \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} = \begin{matrix} Z \\ \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

Multiple heads

- Multi-headed attention: use multiple set of (*key*, *value*, *query*) weights
- Each head will output a vector Z_i



Multiply with weight matrix to reshape

- Gather all the outputs Z_1, \dots, Z_k
- Multiply with a weight matrix to reshape
- Then pass to the next fully connected layer

1) Concatenate all the attention heads



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

X

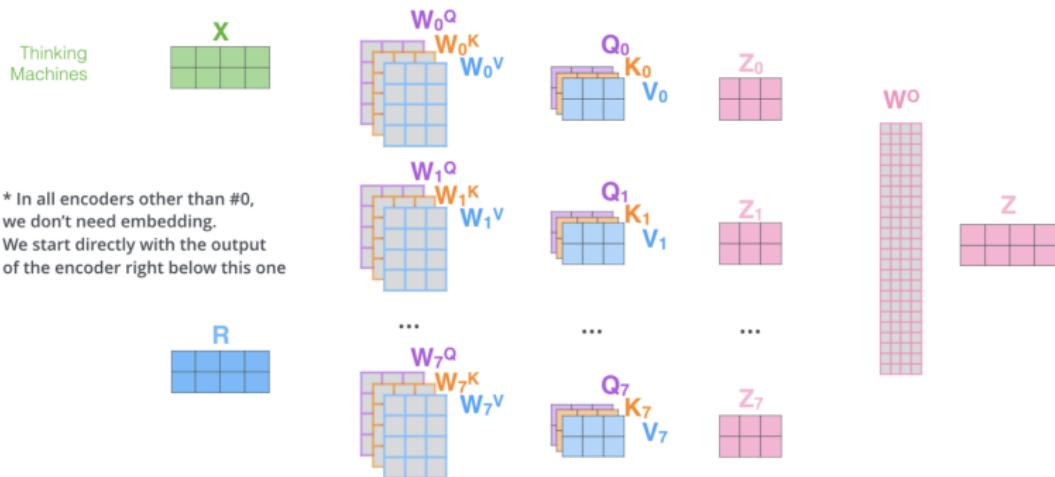


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} Z \\ \begin{matrix} \text{---} & \text{---} & \text{---} & \text{---} \end{matrix} \end{matrix}$$

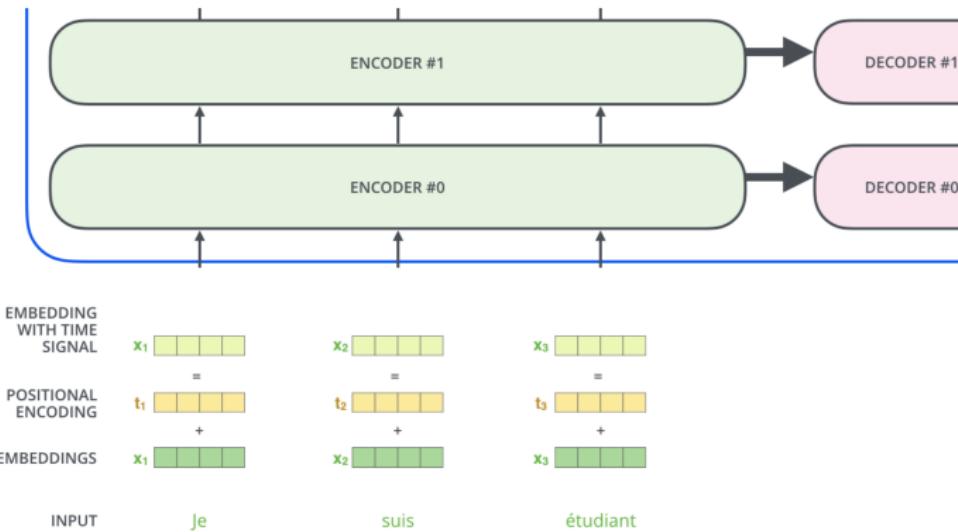
Overall architecture

- 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



Position encoding

- The above architecture ignores the sequential information
- Add a position encoding vector to each x_i (according to i)



Position encoding

- Sin/cosine functions with different wavelengths (used in the original Transformer)

$$P(k, i) = \sin\left(\frac{k}{n^{2i/d}}\right), P(k, i) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

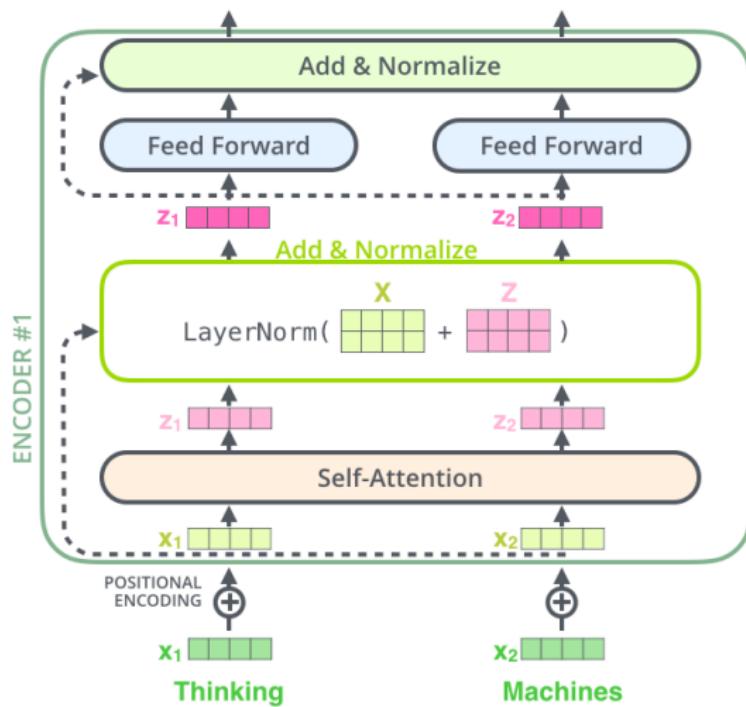
- smooth, parameter-free, inductive
- k : position, i : $0 < i < d/2$, n : user-defined scalar, 10,000 in the original paper

Sequence	Index of token, k	Positional Encoding Matrix with $d=4$, $n=100$			
		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	1	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	2	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	3	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Positional Encoding Matrix for the sequence 'I am a robot'

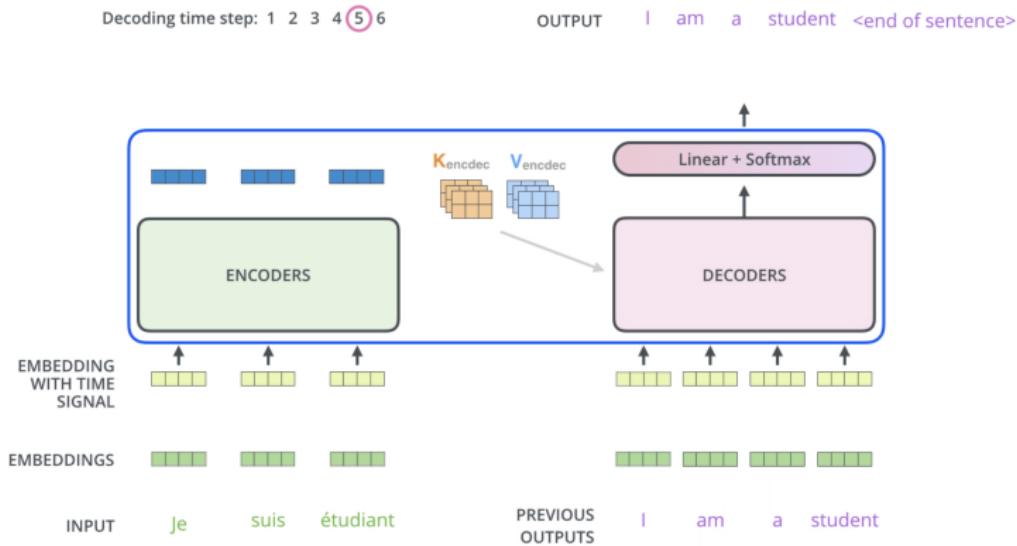
The Residuals

- Residual connection and Normalization



Decoder

- K and V from the final encoder layer used by all the encoder-decoder attention layers in the decoding part.
- Decoding sequentially not parallelly
- Attention layer only uses information from previous positions



Conclusions

- A review of RNN and NMT
- A brief introduction of Transformer.

Questions?