STOR566: Introduction to Deep Learning

Lecture 17: Transformer

Yao Li
UNC Chapel Hill

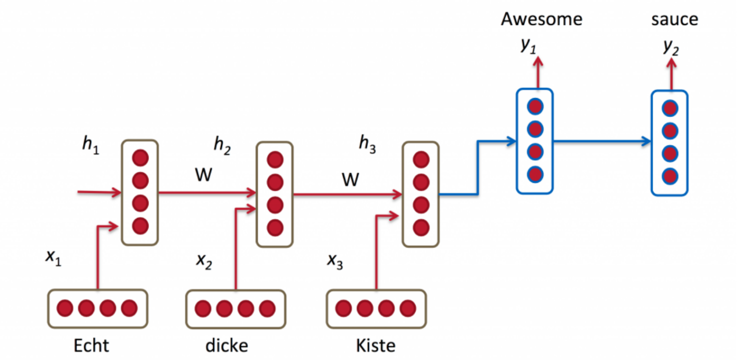
Oct 29, 2024

Materials are from *Deep Learning (UCLA) and Jay Alammar's Blog*

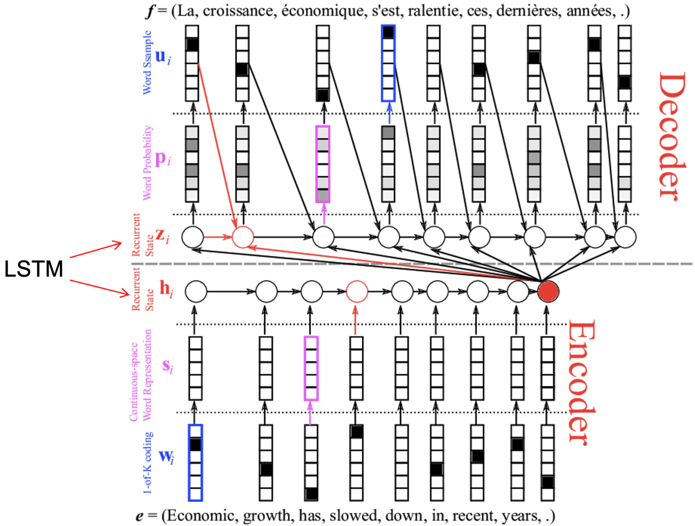
Neural Machine Translation

Neural Machine Translation (NMT)

- Out 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 input a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector

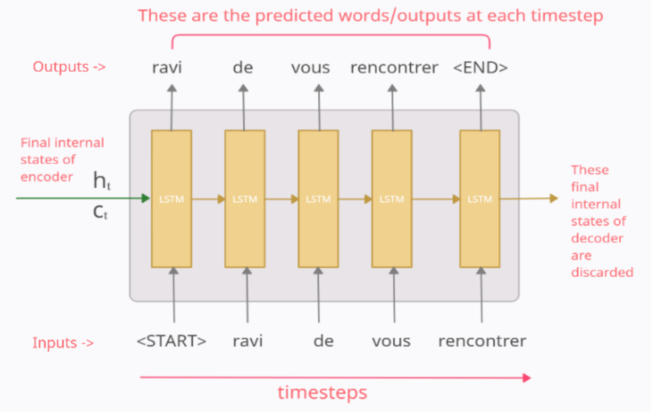


Neural Machine Translation



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

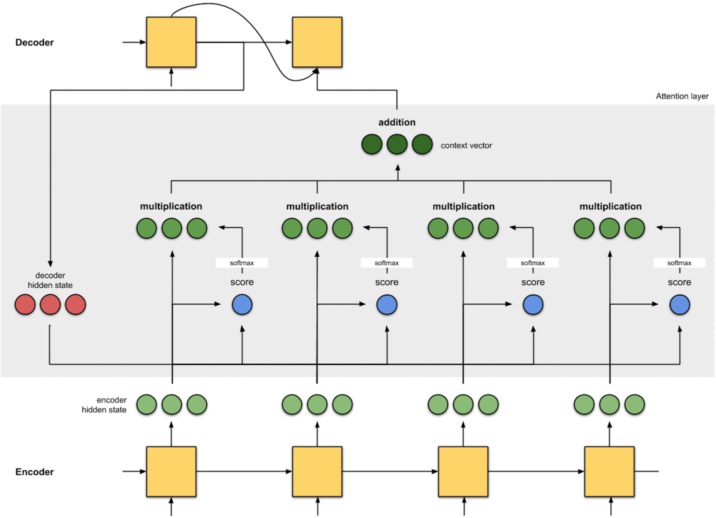
Attention in NMT

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

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

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

Attention in NMT



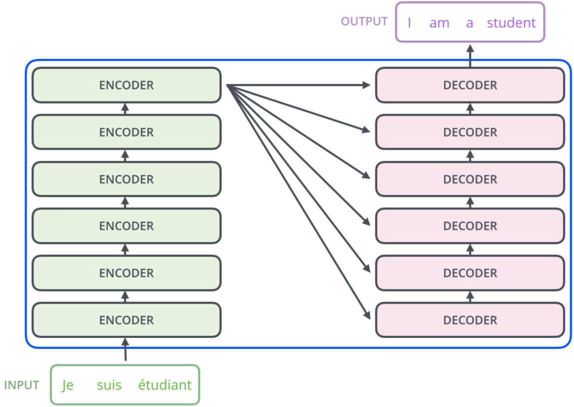
(Figure from `https://towardsdatascience.com/`

Transformer

Materials are from *Jay Alammar's Blog*

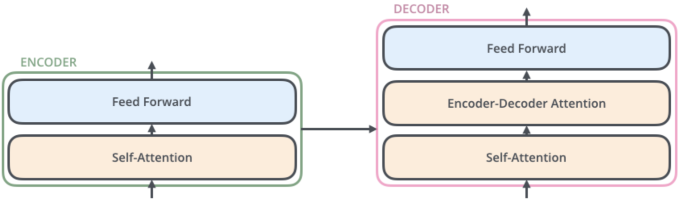
Transformer

- 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



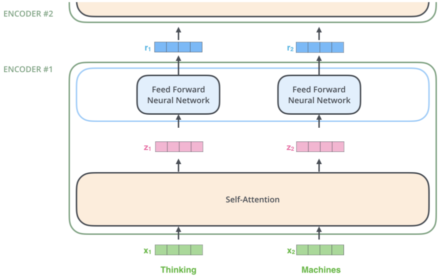
Encoder and Decoder

- 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.



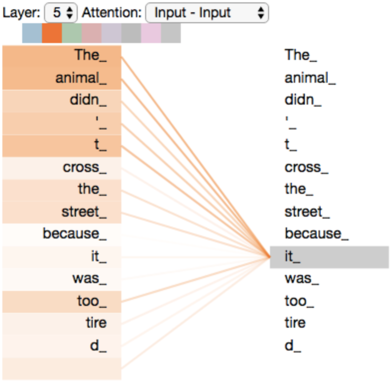
Encoder

- 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-foward layer
  - Output a list of vectors



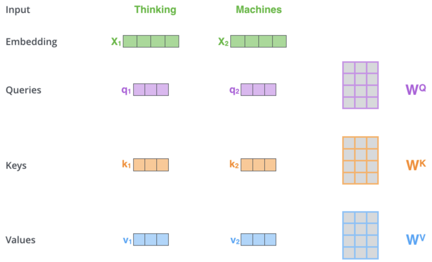
Self-attention layer

- 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



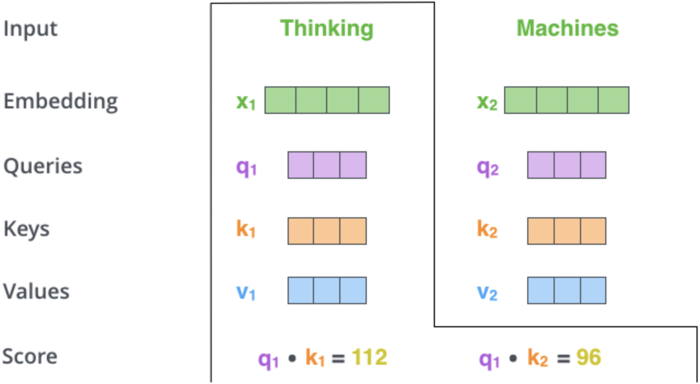
Self-attention layer

- Input latent vectors: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $\boldsymbol{q}_i = \boldsymbol{x}_i W^Q$
  - Key vector: $\boldsymbol{k}_i = \boldsymbol{x}_i W^K$
  - Value vector: $\boldsymbol{v}_i = \boldsymbol{x}_i W^V$



Self-attention layer

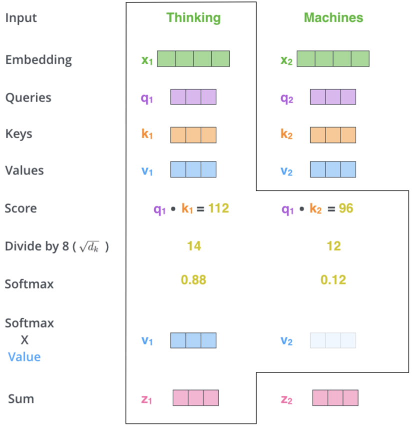
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $\boldsymbol{q}_i^T \boldsymbol{k}_j$



Self-attention layer

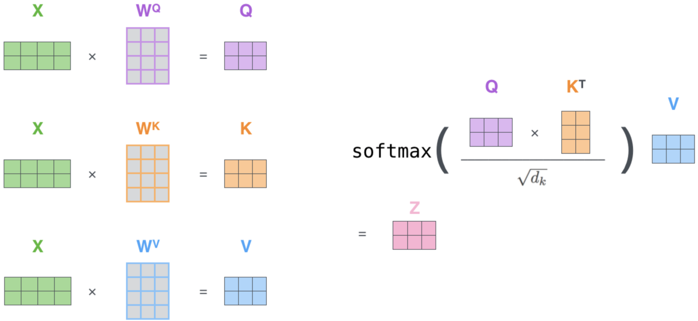
- For each word i, the output vector

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



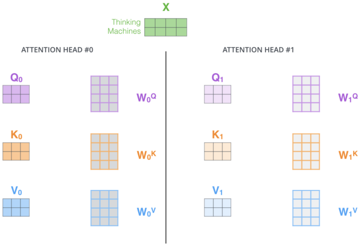
Matrix form

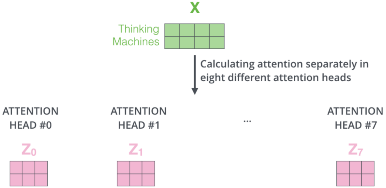
$$Q = XW^Q, \; K = XW^K, \; V = XW^V, \quad Z = \text{softmax}(QK^T)V$$



Multiple heads

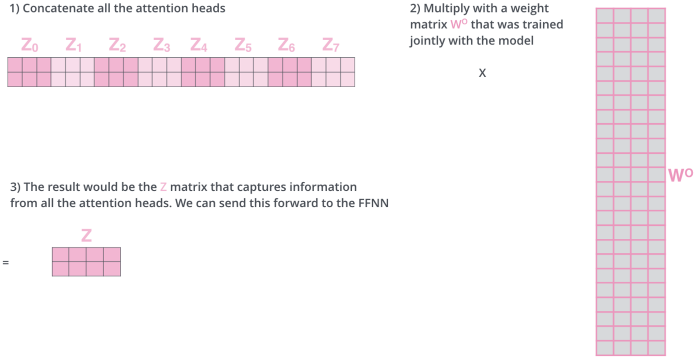
- Multi-headed attention: use multiple set of $(key, value, query)$ weights
- Each head will output a vector Z_i



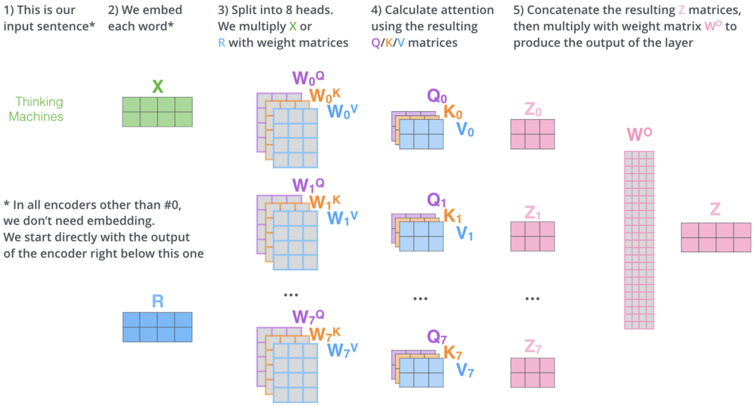


Multiply with weight matrix to reshape

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

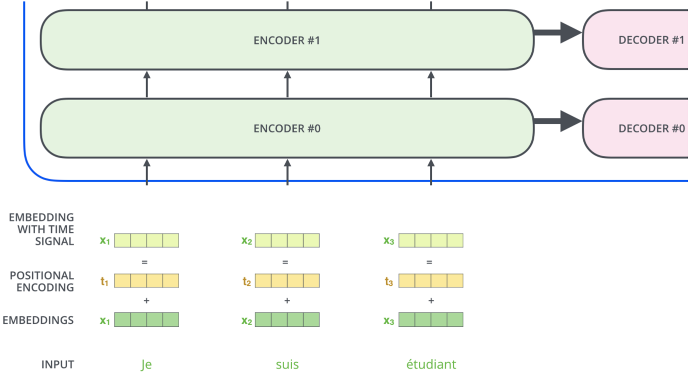


Overall architecture



Position encoding

- The above architecture ignores the sequential information
- Add a position encoding vector to each $\boldsymbol{x}_i$ (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: $0 \le i < d/2$, n: user-defined scalar, 10,000 in the original paper

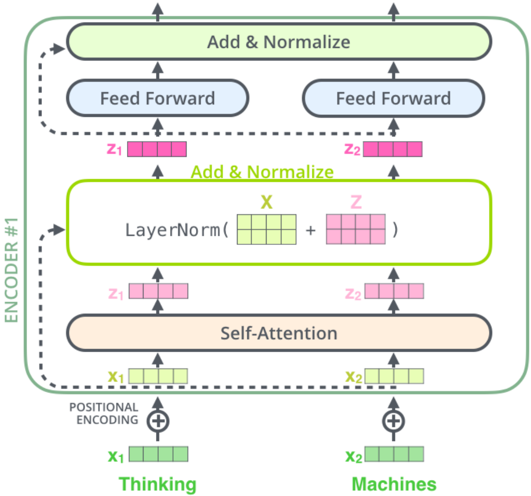
Positional Encoding Matrix with d=4, n=100

Sequence	Index of token, k	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

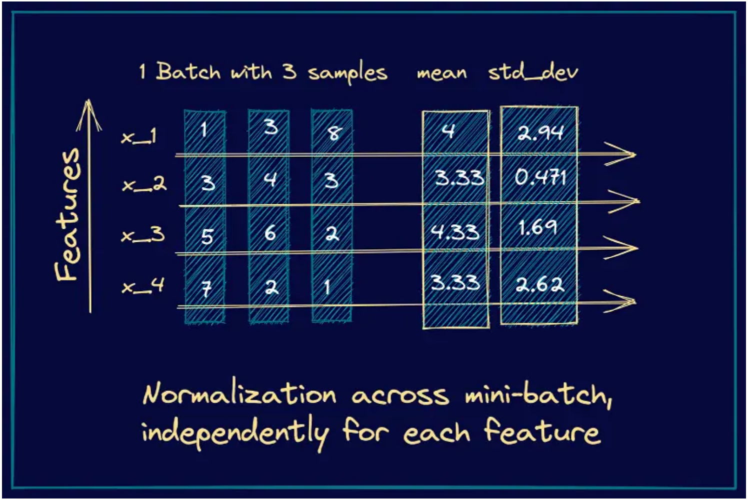
Positional Encoding Matrix for the sequence 'I am a robot'

The Residuals

- Residual connection and Normalization

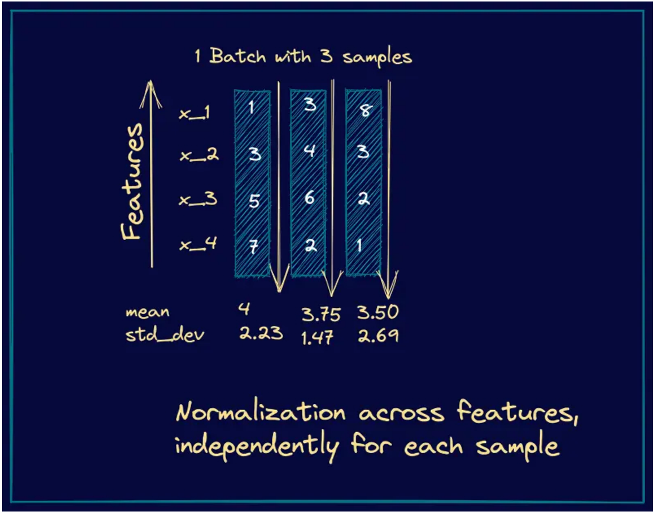


Batch Normalization



picture from https://www.pinecone.io/learn/batch-layer-normalization/

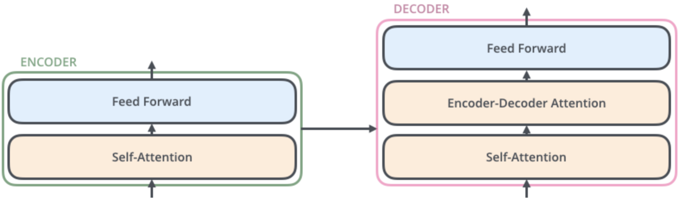
Layer Normalization



picture from https://www.pinecone.io/learn/batch-layer-normalization/

Decoder

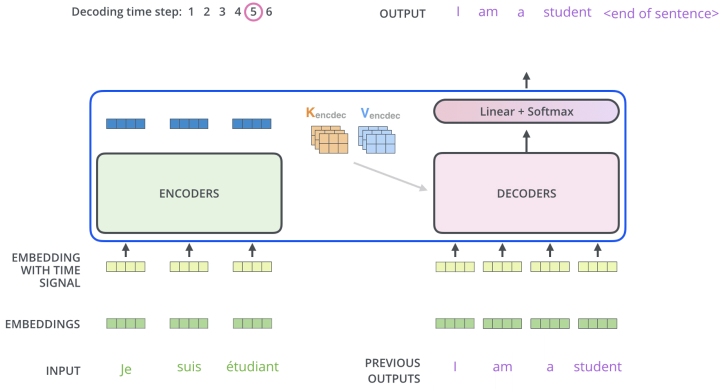
- Decoder: Self attention layer + Encoder-Decoder Attention Layer + Feed Forward



- New: Encoder-Decoder Attention Layer

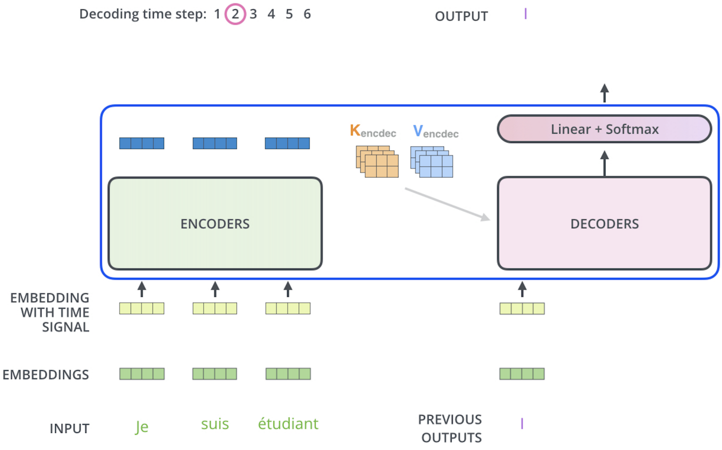
Encoder-decoder attention layer

- K and V from the final encoder layer used by all the encoder-decoder attention layers in the decoding part.
- Q query vectors produced by the decoder inputs will be used with K and V to produce the output of encoder-decoder attention layer.



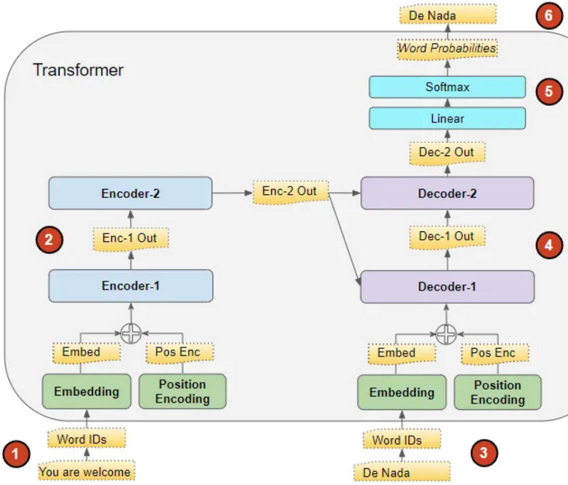
Decoder Self-attention

- Self-attention layer only uses information from previous positions in the output sequence
- Mask future positions with $-\infty$



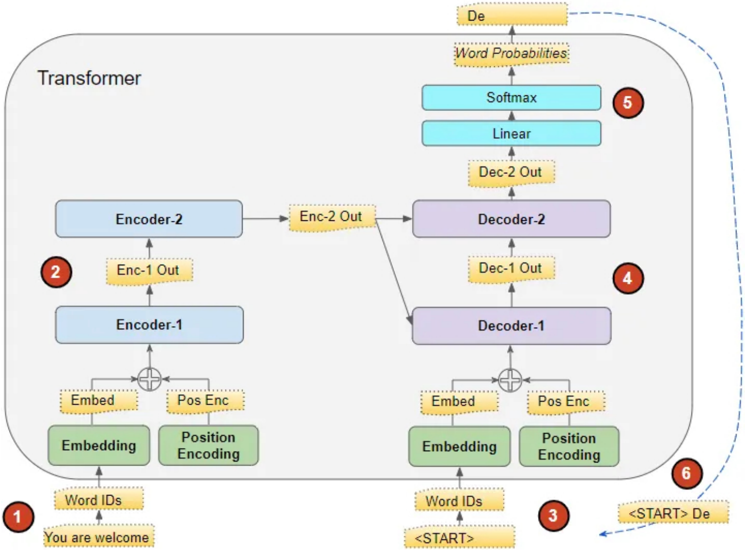
Training

- Decoding parallelly
- Mask future positions with $-\infty$



Inference

- Decoding sequentially not parallelly



Conclusions

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

Questions?