Attention Is All you need

Reading Group

E/15/021
E/15/330
E/15/366
E/15/373

Today’s Paper : ‘Attention Is All you need‘

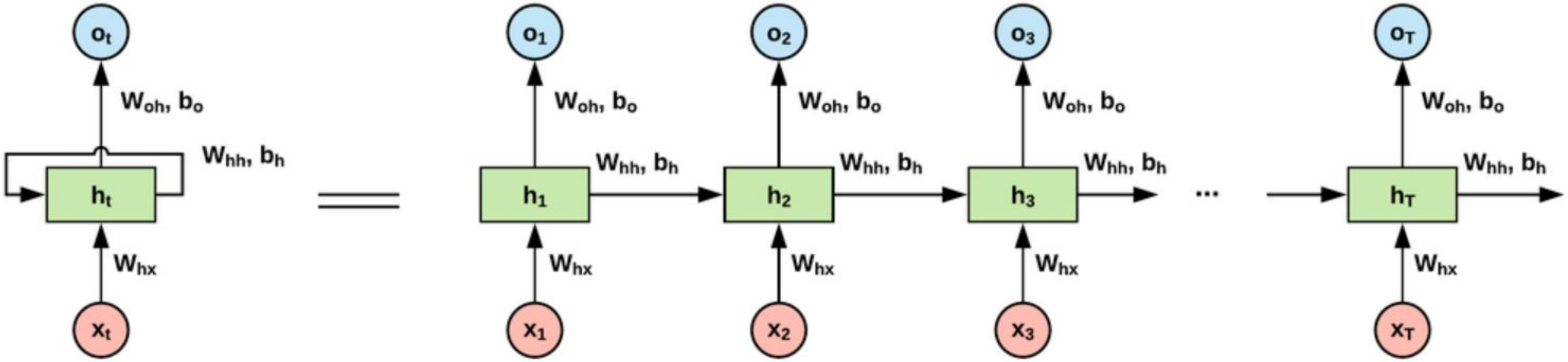
- Conference : NIPS 2017
- Cited 966 times.
- Authors :
  - Ashish Vaswani (Google Brain)
  - Noam Shazeer (Google Brain)
  - Niki Parmar (Google Research)
  - Jakob Uszkoreit (Google Research)
  - Llion Jones (Google Research)
  - Aidan N. Gomez (University of Toronto)
  - Łukasz Kaiser (Google Brain)
  - Illia Polosukhin

E/15/330

RECURRENT NEURAL NETWORKS: INTUITION

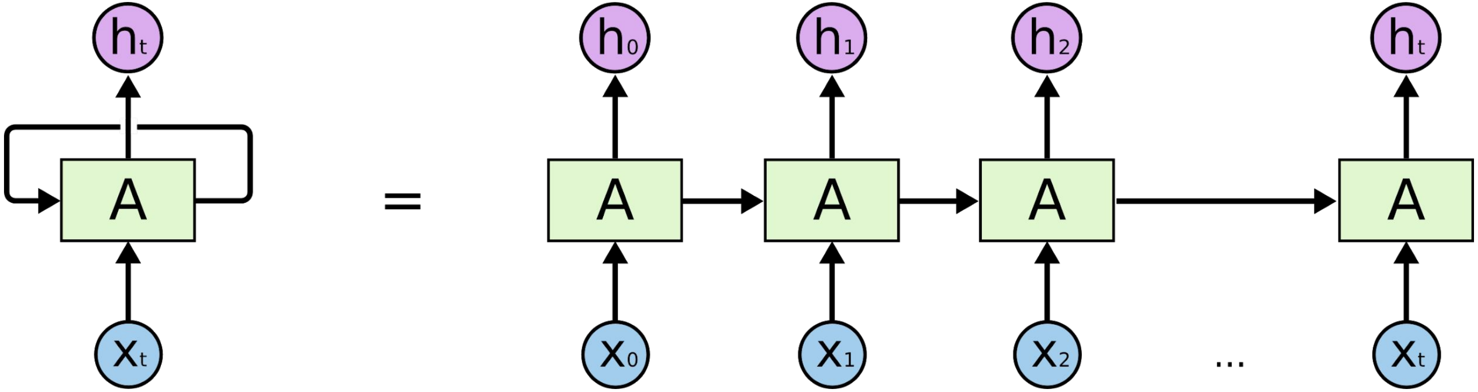
- Recurrent neural network (RNN) is a neural network model proposed in the 80's for modelling time series.

- The structure of the network is similar to feedforward neural network, with the distinction that it allows a recurrent hidden state whose activation at each time is dependent on that of the previous time (cycle)



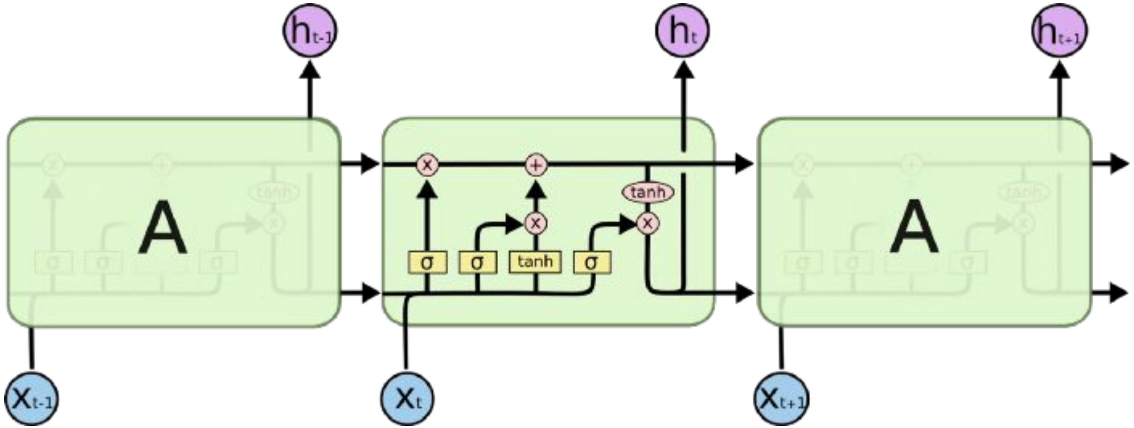
Introduction: former techniques are not good at parallelization

- Recurrent Neural Networks (RNNs) and their cell variants are firmly established as state of art in sequence modeling and transduction. *(Eg: machine translation)*
- RNN generates a hidden states(h_t) as a function of the previous hidden states(h_{t-1}) and the input.



The fall of RNN / LSTM

- Recurrent neural network(RNN) is a good way to process sequential data, but the capability of RNN to compute long sequence data is inefficient.
- RNN is that they are not hardware friendly.



- LSTM and GRU and derivatives are able to learn a lot of longer term information! but they can remember sequences of 100s, not 1000s or 10,000s or more.

Introduction

- Using attention mechanisms allow us to draw global dependencies between input and output by a constant number of operations.

- In this work, they propose the **Transformer** which doesn't use recurrent architecture or convolutional architecture, and reaches a state-of-the-art in translation quality.

E/15/330

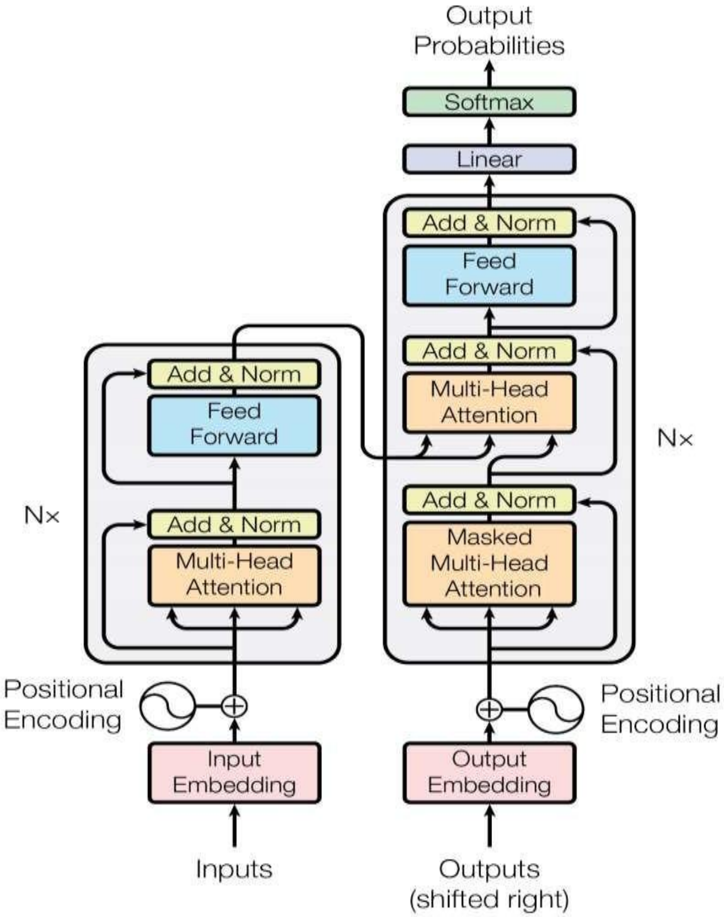
Background

- Attention mechanisms have become an integral part of recent models, but such attention mechanisms are used with a recurrent network.
- Reducing sequential computation is achieved by using CNN, or computing hidden states in parallel.
- But in these methods, the number of operations to relate two input and output positions grows in the distance between distances. It is difficult to learn dependencies between distant positions.

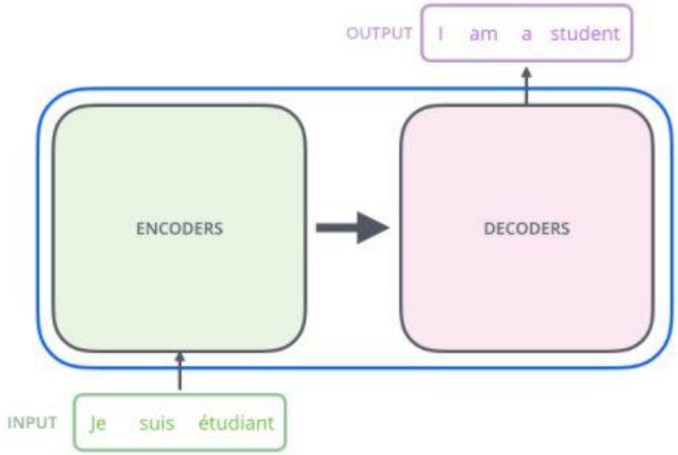
Entire Model Architecture

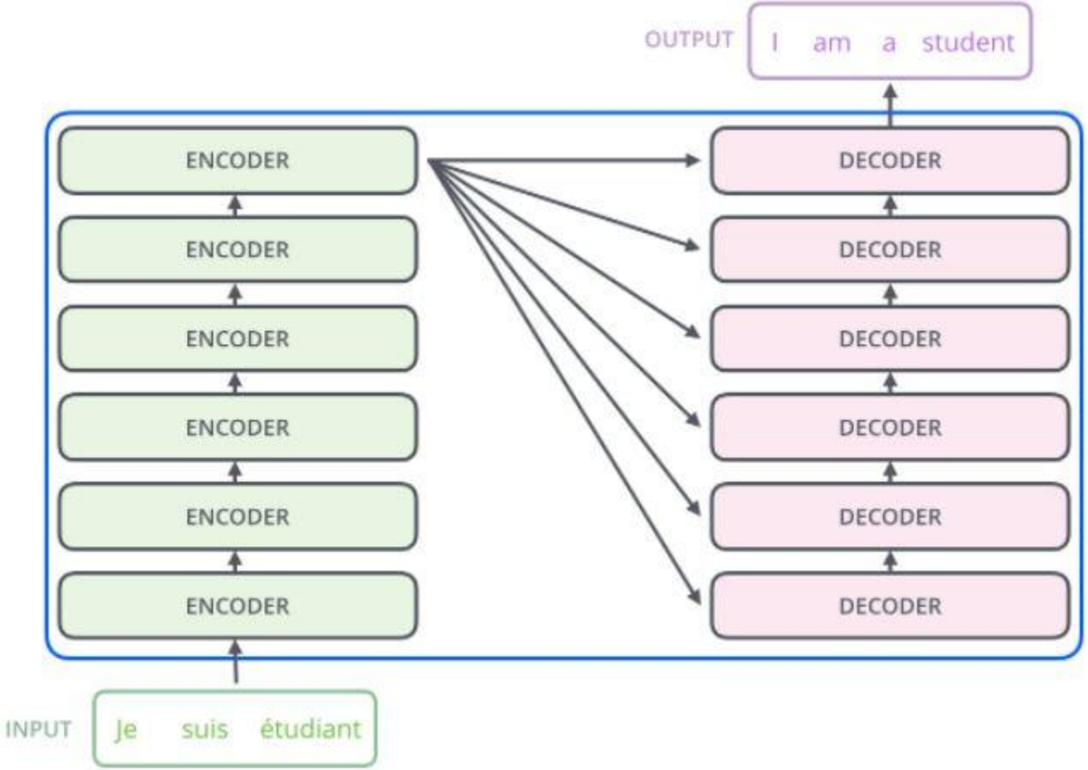
- Left side : Encoder
- Right side : Decoder

- Consisting layers:
  - **Multi-Head Attention layer**
  - **Position-wise Feed-Forward layer**
  - **Positional Encoding**
  - (Residual Adding and Normalization layer)

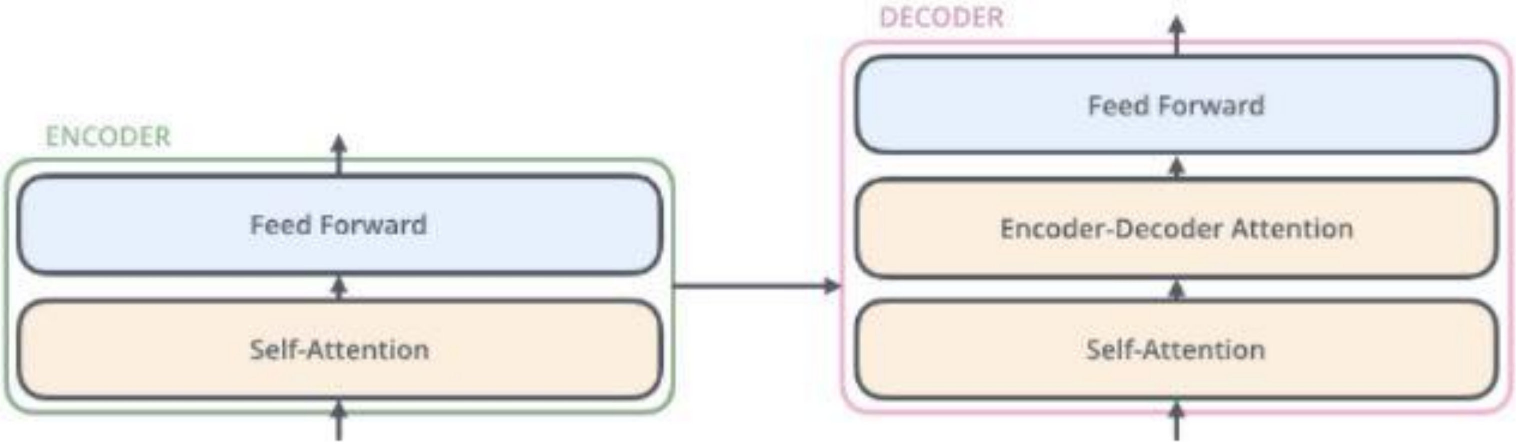


Encoders And Decoders- High Level Look



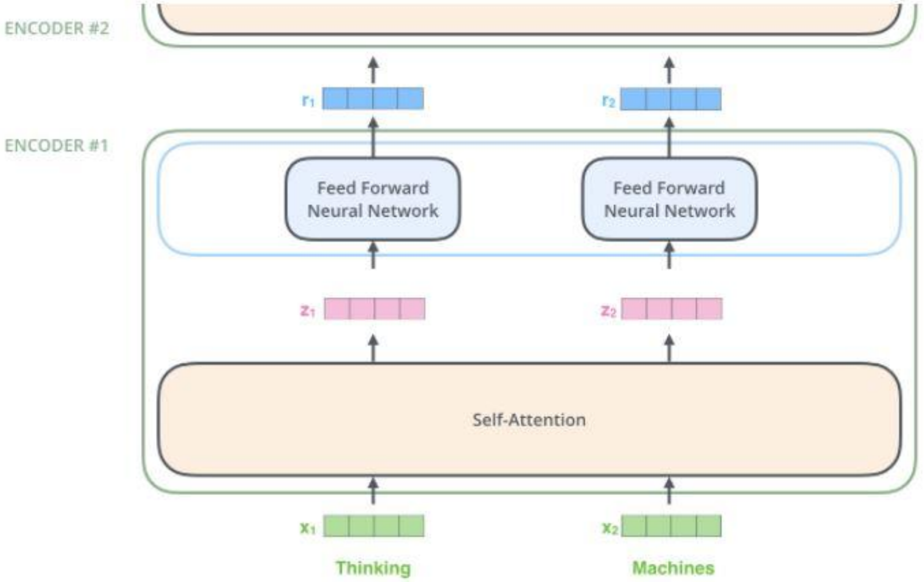


Inside the Encoder and Decoder



- The encoder's inputs first flow through a self-attention layer.
- The outputs of the self-attention layer are fed to a feed-forward neural network.
- The decoder has both those layers, but between them is an attention layer.

Bringing The Tensors Into The Picture



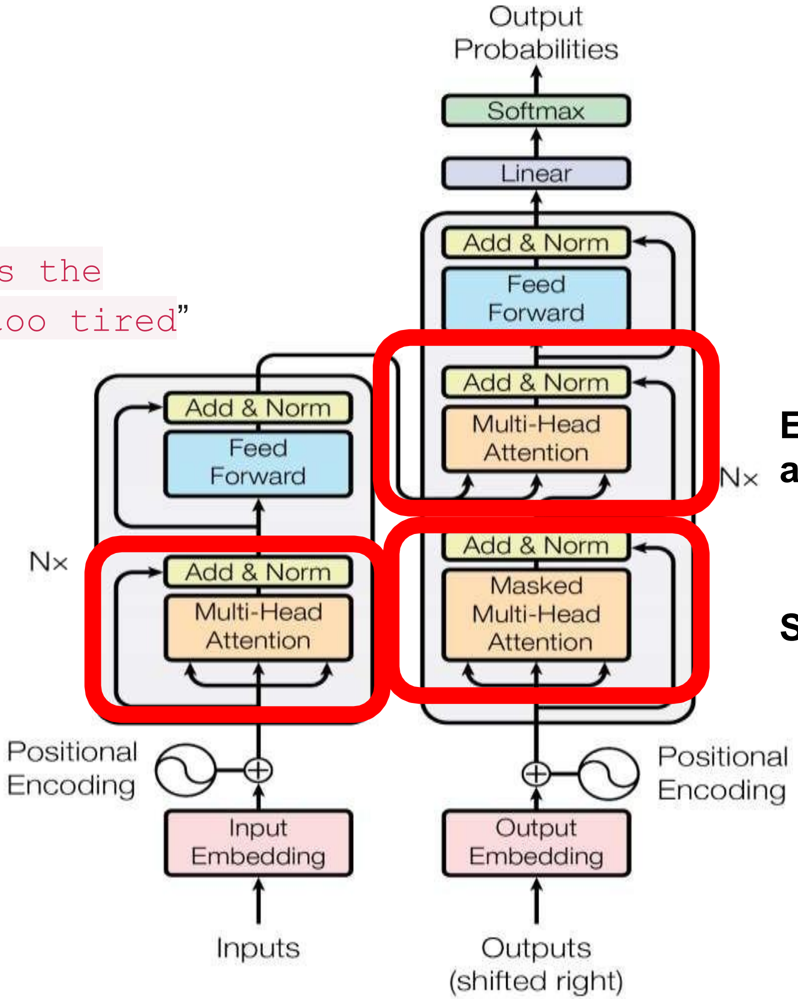
- NLP applications turning each input into a vector using an embedding algorithm.
- After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

1. Attentions

:

"The animal didn't cross the street because it was too tired"

Self attention in Encoder



Encoder-Decoder attention

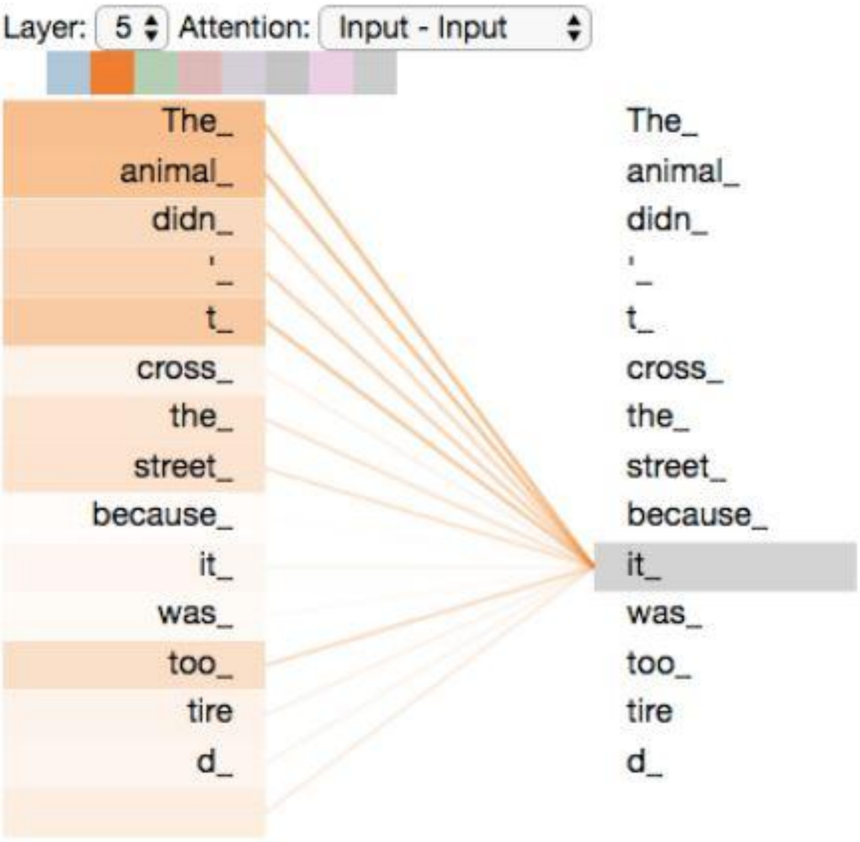
Self attention in Decoder

E/15/373

Self-Attention at a High Level

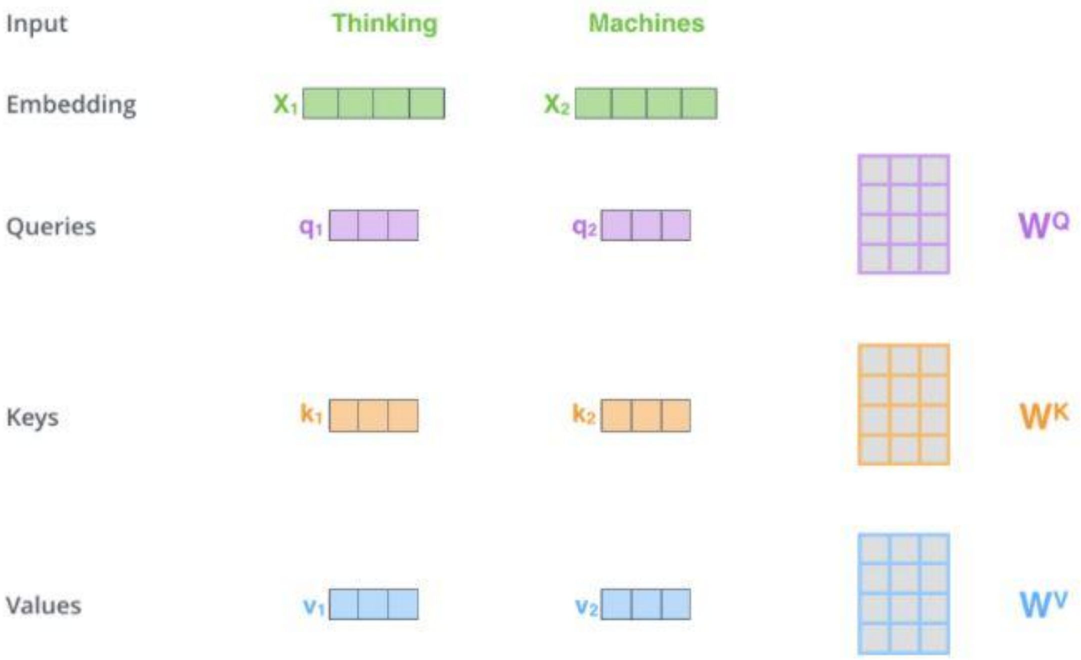
”`The animal didn't cross the street because it was too tired`”

- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.



Self-Attention in Detail

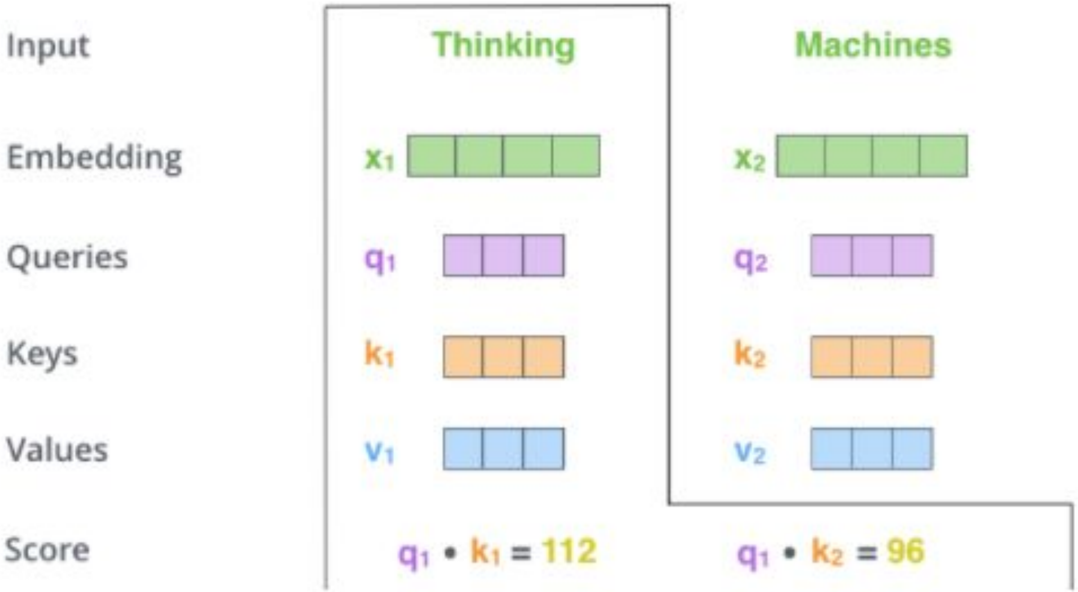
- Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.



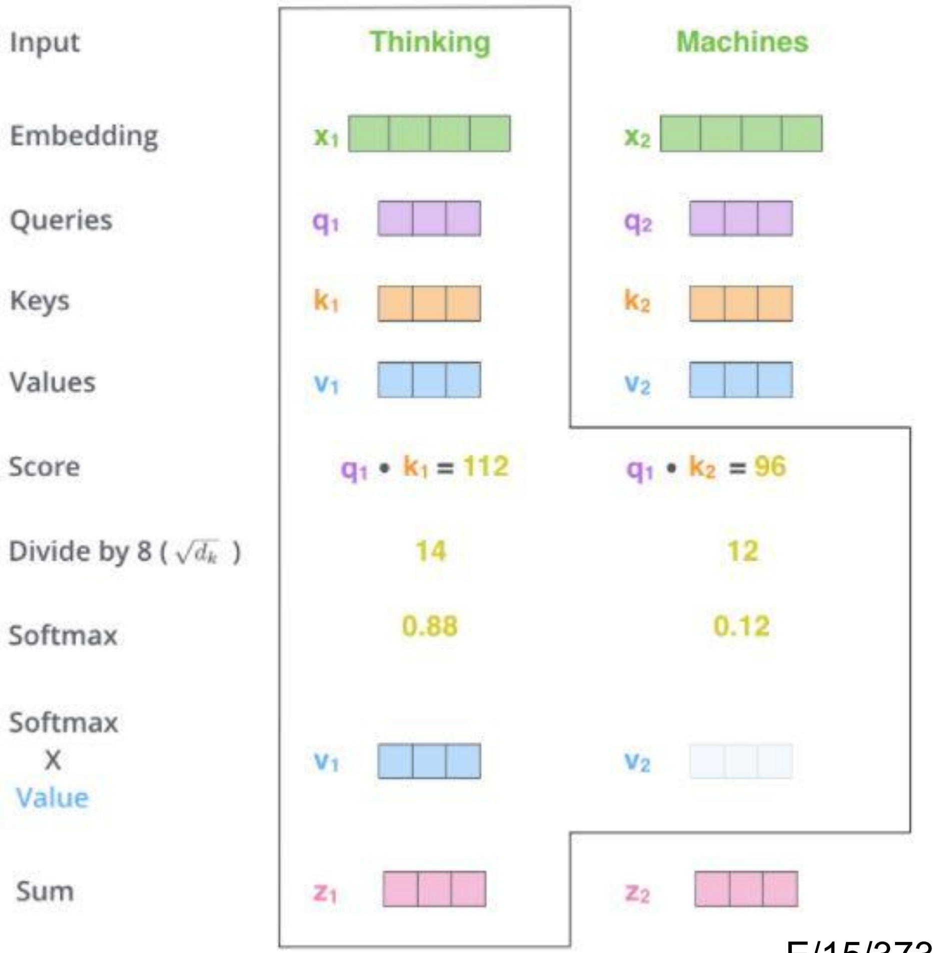
Self-Attention in Detail

- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.

- So if we're 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.

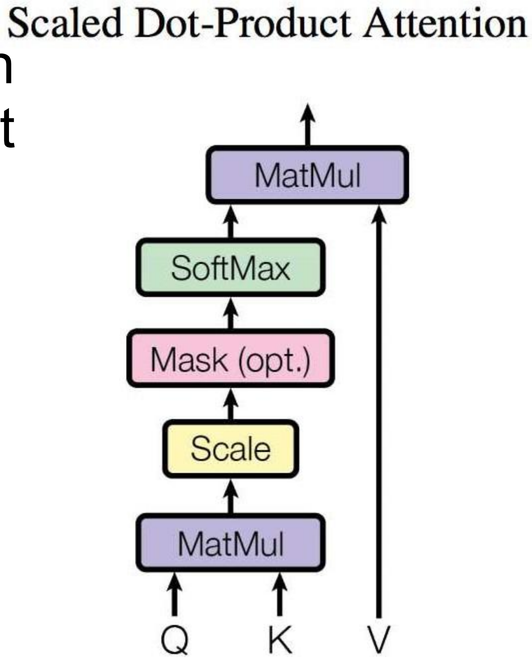


Self-Attention in Detail

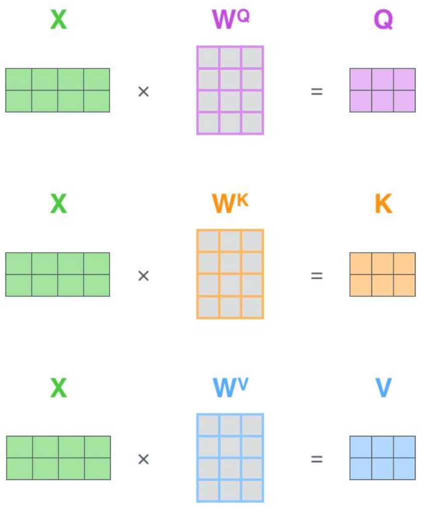


Scaled Dot-Product Attention

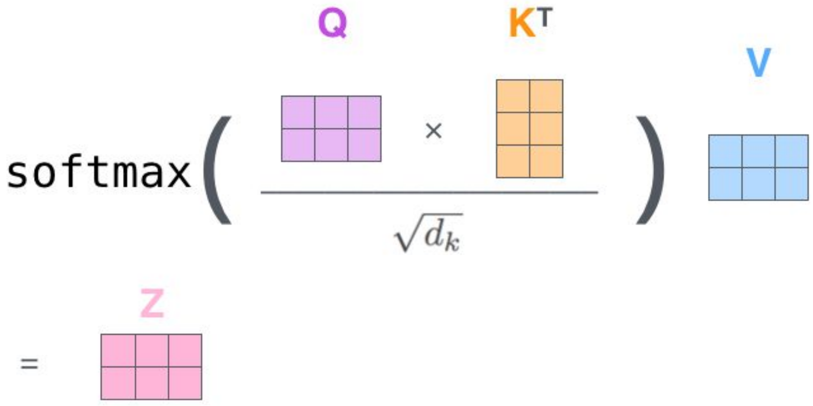
- There are two most commonly used attention functions : additive attention and dot-product attention.
  - Additive attention : $Softmax(\sigma(W \quad Q[, K \quad]+b))$
  - Dot-product attention : $Softmax\left(QK^T \quad \right)$
- Dot-product attention is much faster and space efficient.



Scaled Dot-Product Attention



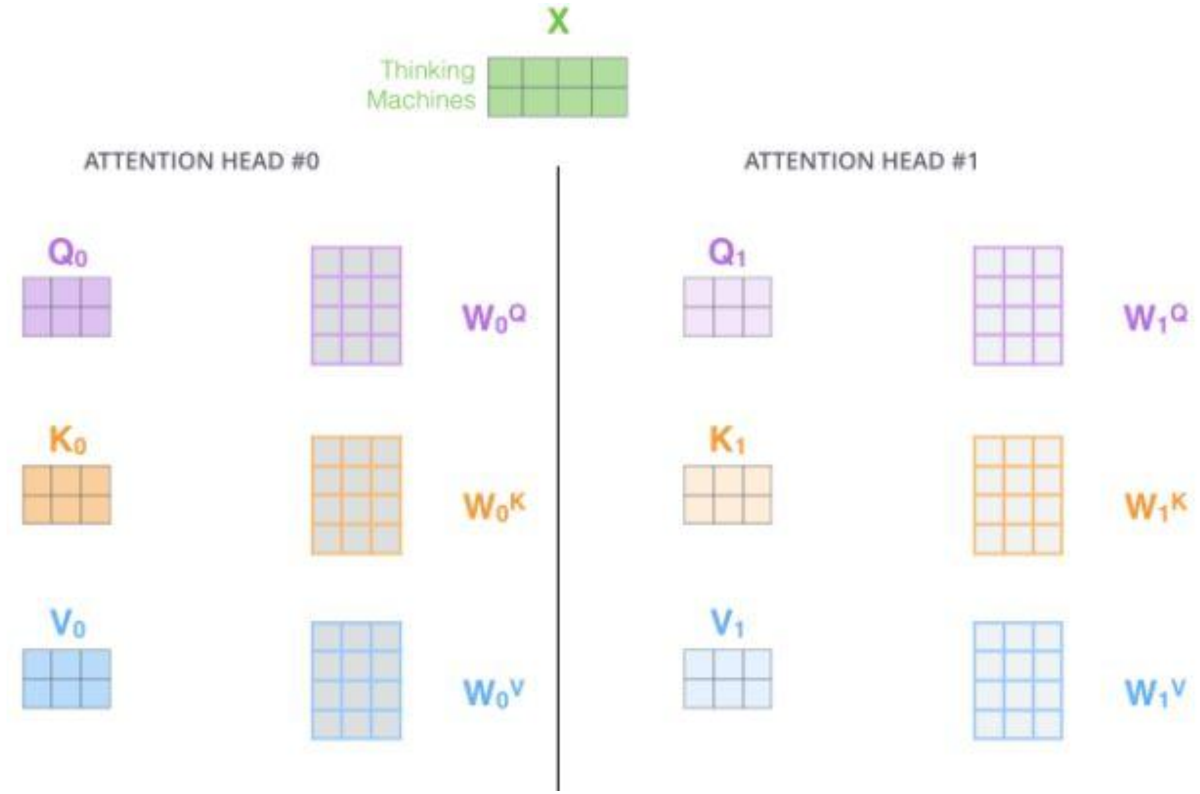
$Attention\ (Q, K, V) =$



where $\sqrt{d_k}$ is the scaling factor preventing softmax function pushed into regions where it has extremely small gradients.

Multi-Head Attention

- 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.



Multi-Head Attention

- Instead of calculating single dot-product attention, they calculate multiple attentions. (for example, h = 8)

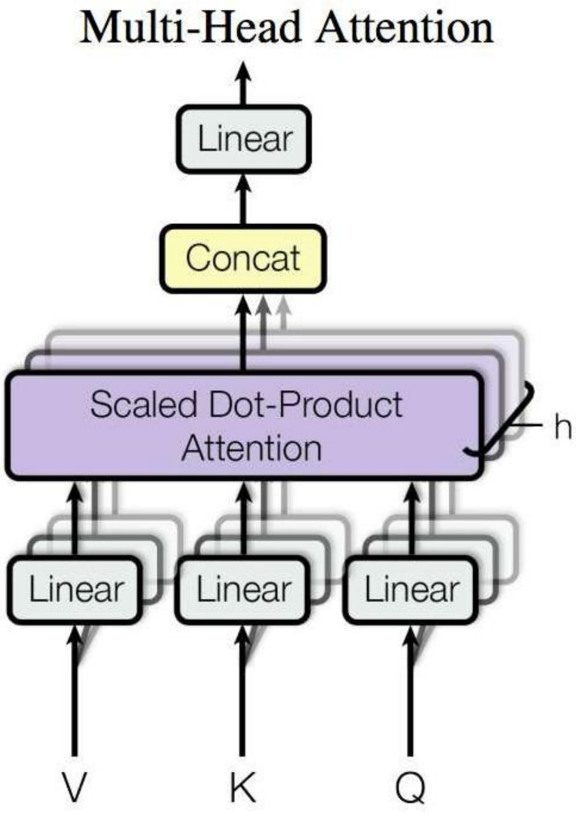
- They linearly project Q, K, and V h-times with different projections to d_k, d_k *and* d_v dimensions.

  (They use $d_k = d_v = \frac{d_{model}}{h}$)

$$MultiHead(Q, K, V) = Concat(head_1, \ldots,)$$

$$where \quad head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad W^O$$

Parameter weight matrices

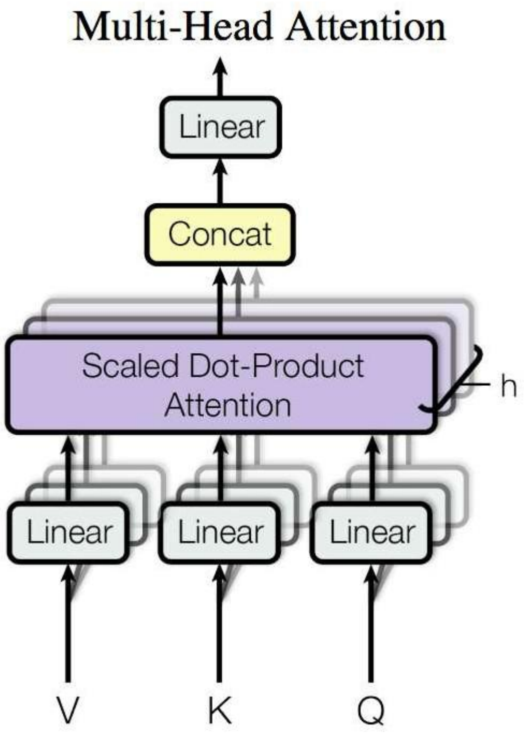


Applications of Multi-Head Attention in model

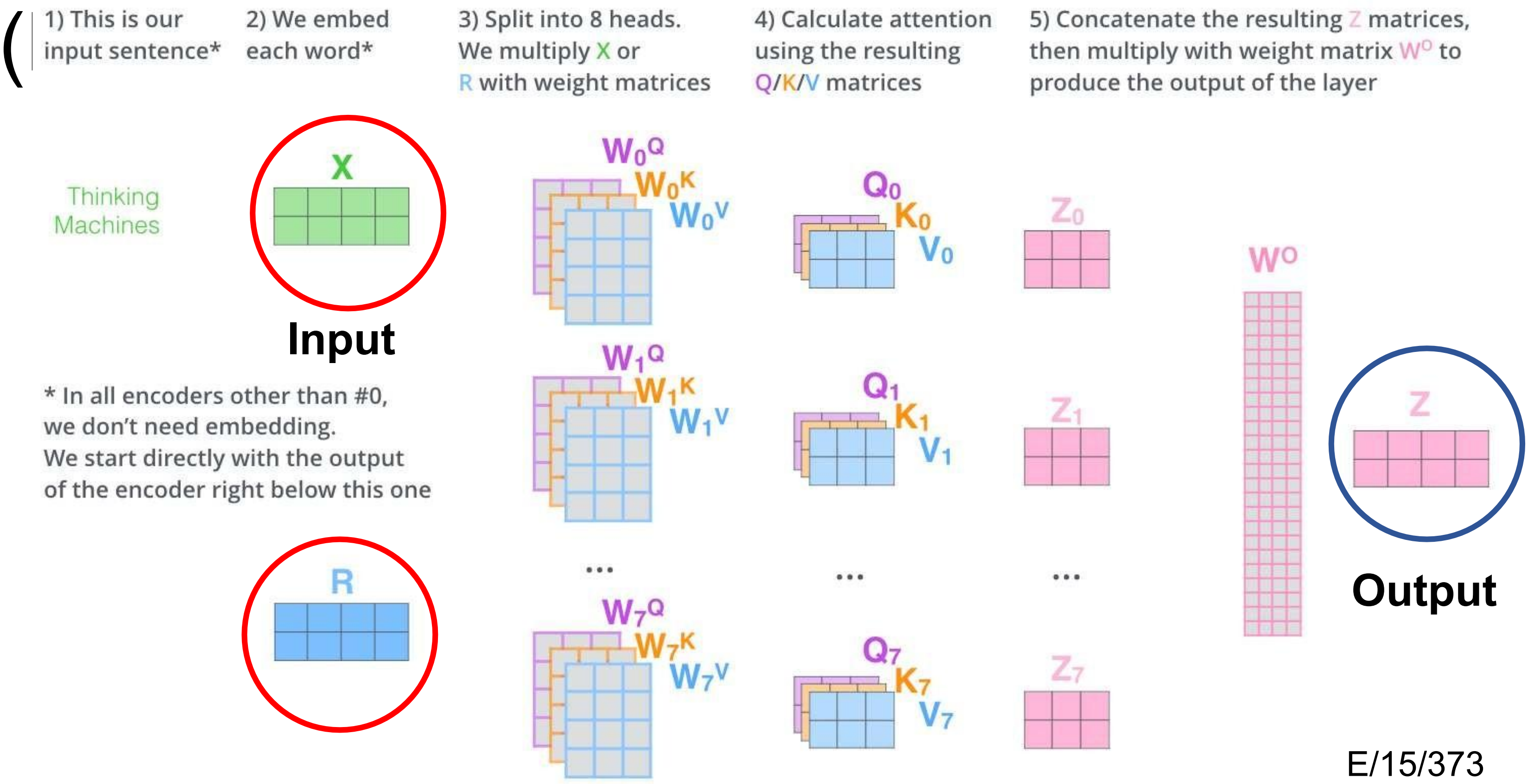
- Transformer uses multi-head attention in three ways.
  1. encoder-decoder attention : The queries(Q) come from the **previous decoder layer**, and keys(K) and values(V) come from the **output of the encoder**. (traditional attention mechanisms)

  2. Self-attention layers in the encoder : All of the keys(K), values(V) and queries(Q) come from the same place, in this case, the **output of the previous layer in the encoder**.

  3. Self-attention layers in the decoder : K,V,Q come from the **output of the previous layer in the decoder**. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property.

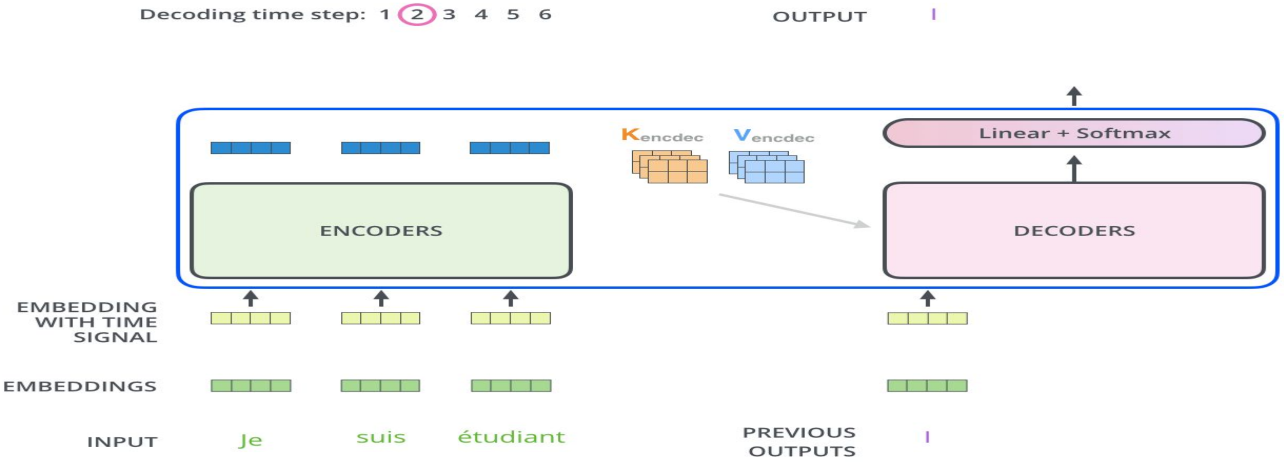


Data Flow in Attention



Self-Attention in Decoder

• In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is



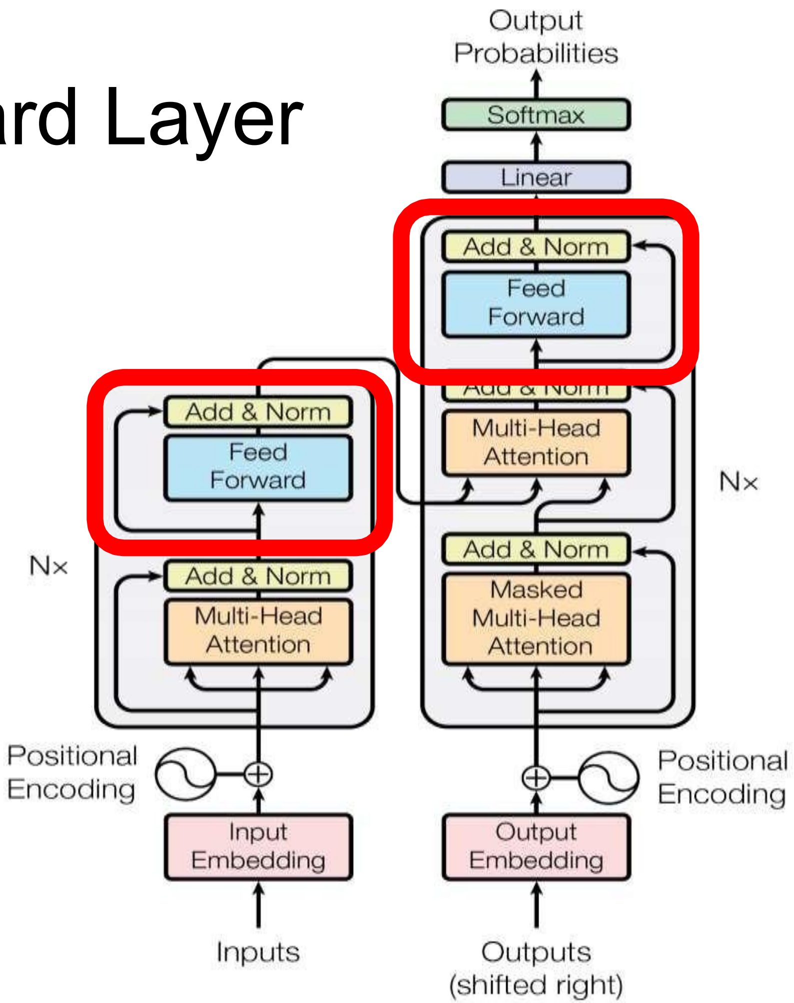
Why Self-Attention?

1. Total computational complexity per layer
2. The amount of computation that can be parallelized
3. The path length between long-range dependencies
4. (As side benefit, self-attention could yield more interpretable models.)

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

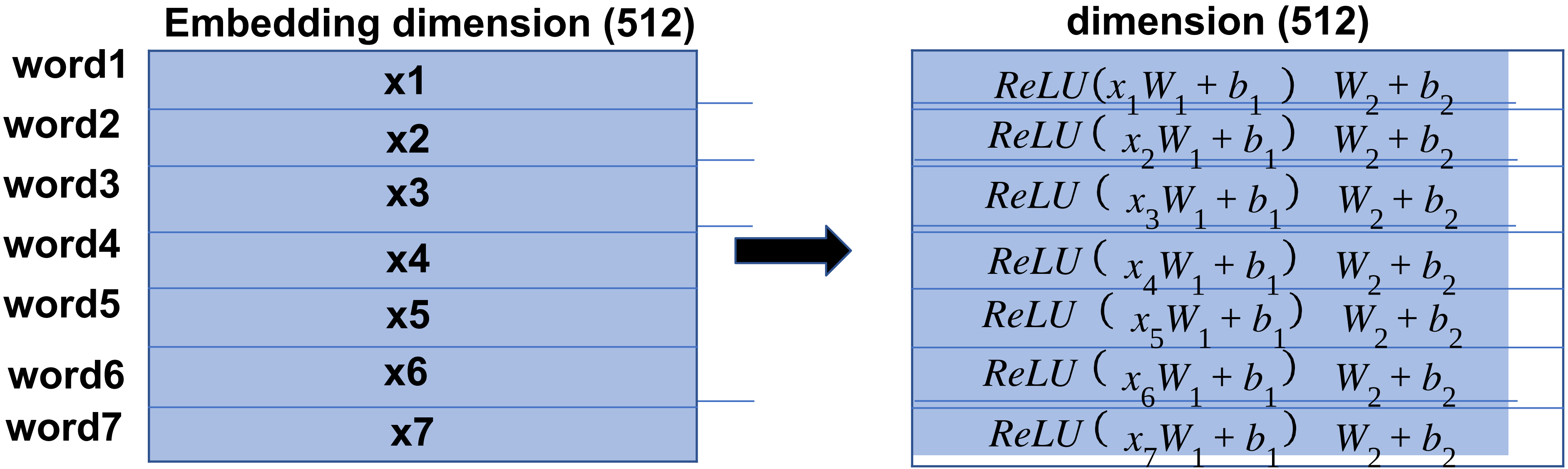
2. Feed Forward Layer



Position-wise Feed-Forward Networks

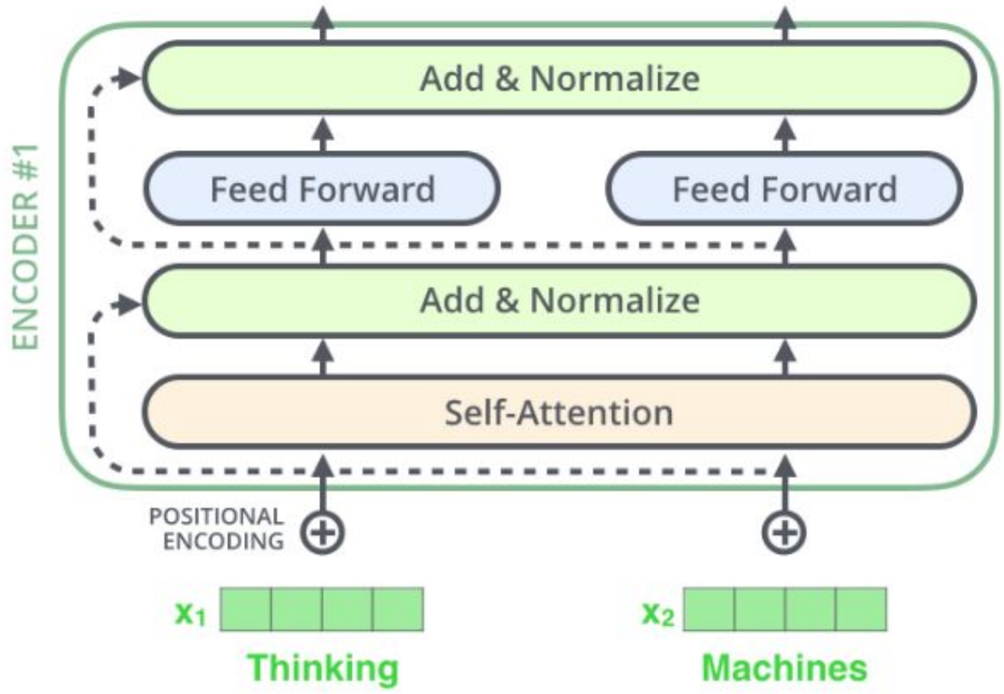
- Feed-forward networks are applied to each position separately.

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$



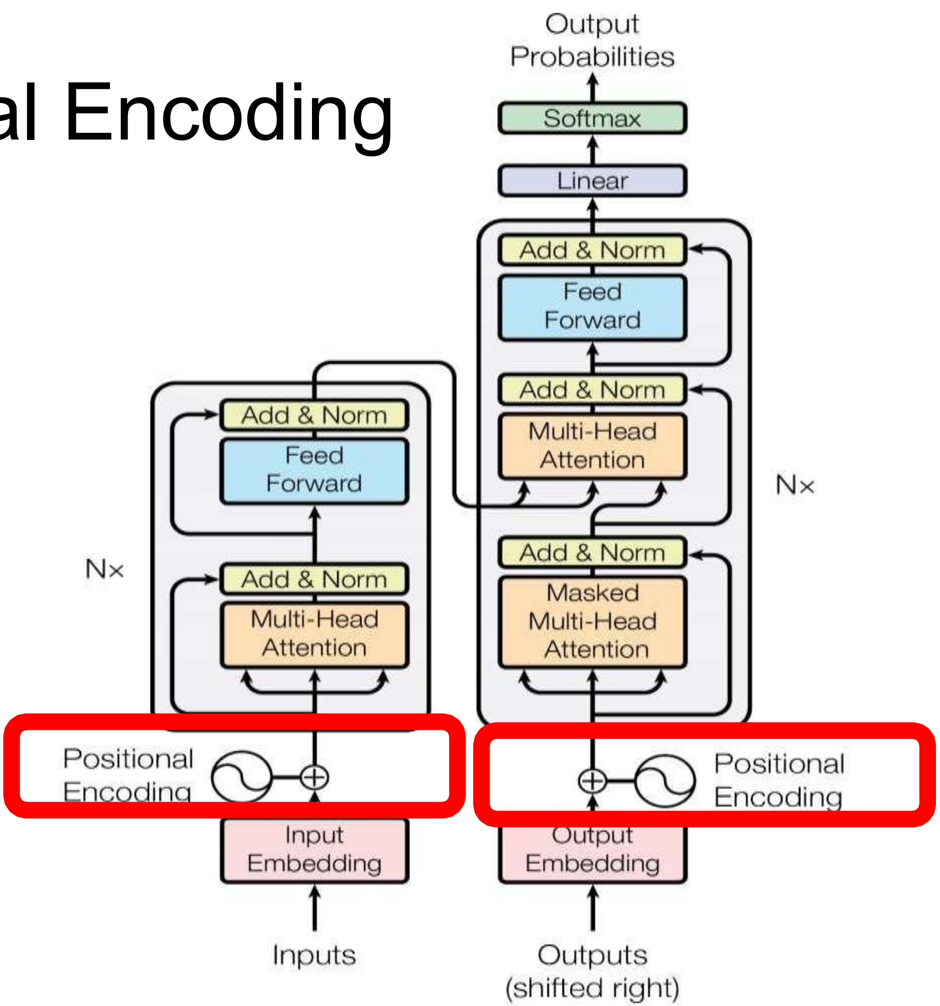
The Residuals

- Skipping some layers



E/15/021

3. Positional Encoding



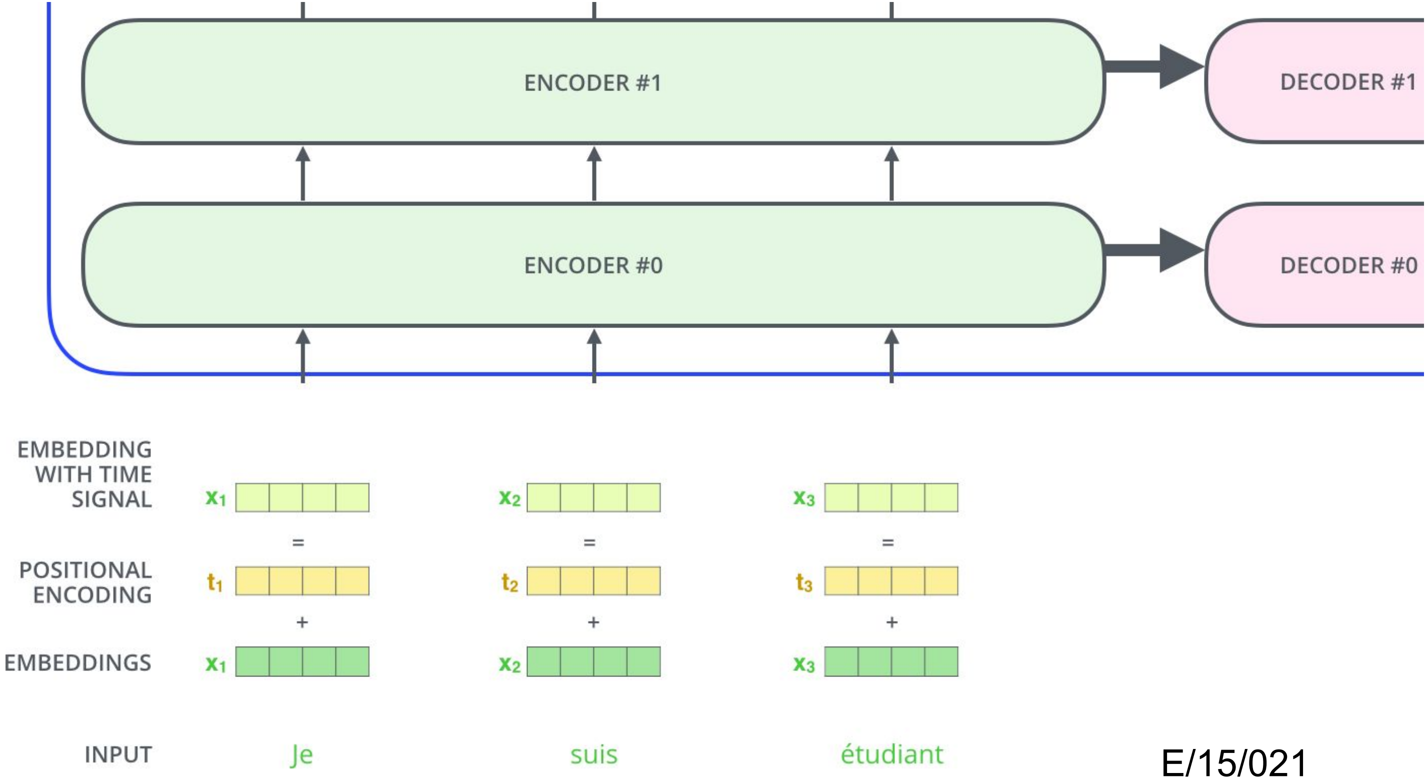
Positional Encoding

- Since their model doesn't contain recurrence and convolution, it is needed to inject some **information about the position of the tokens in the sequence**.
- So they add "positional encoding" to input embedding.
- Each element of PE is as following :

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

  - pos is the location of the word, i is the index of dimension in word embedding.

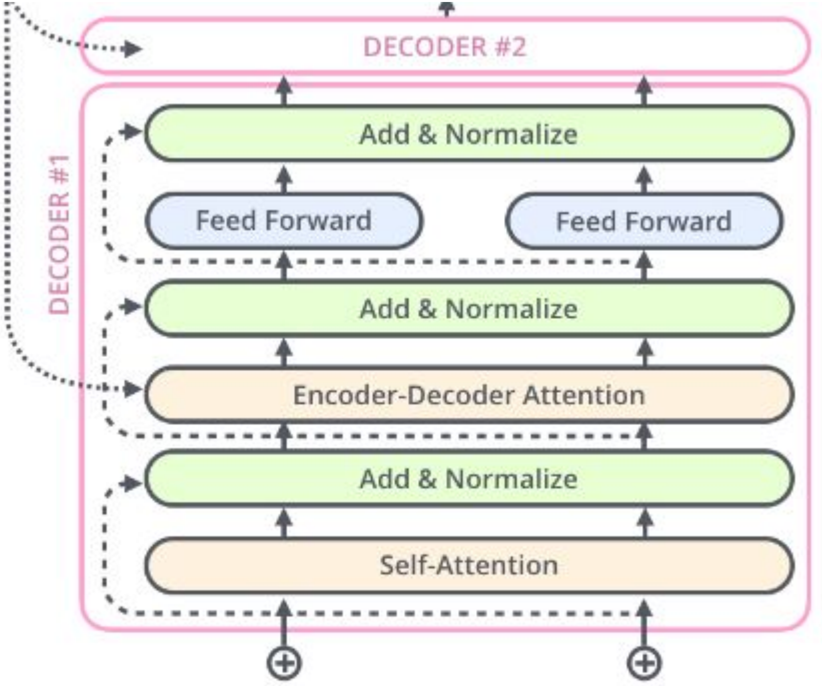
Positional Encoding



E/15/021

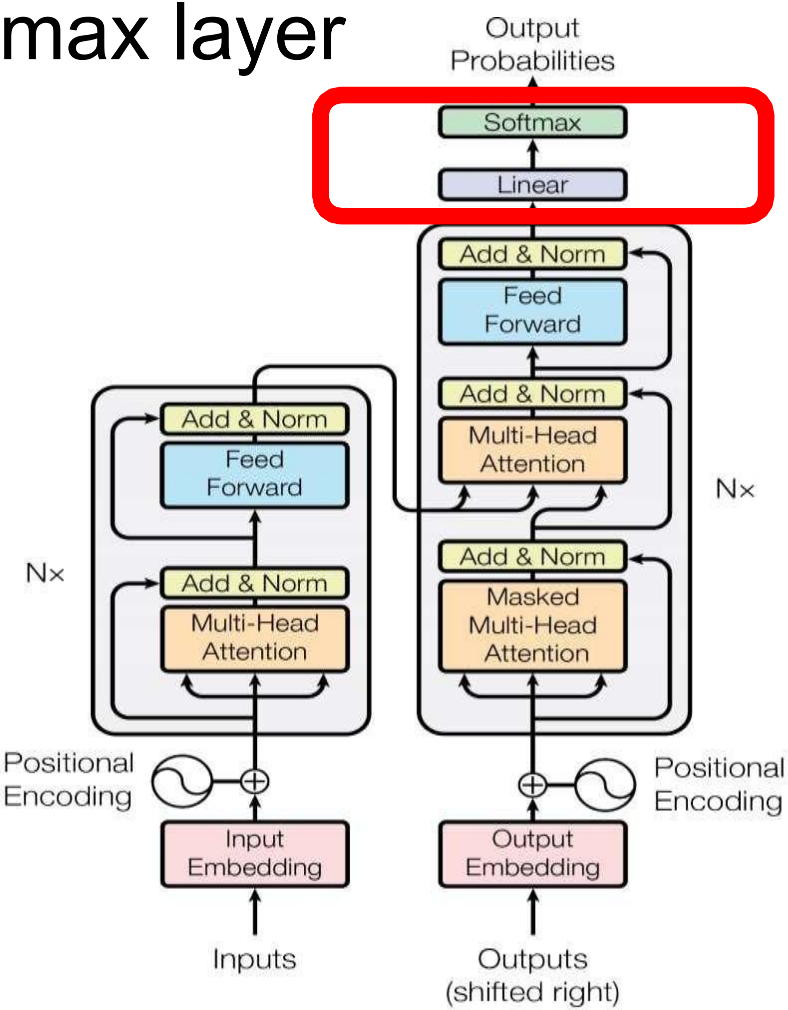
Decoder

- 6 layers
- 3 sub layers in each layer
  - Self-attention
  - Encoder-decoder attention
  - Feed-forward
- 2 inputs to each decoder



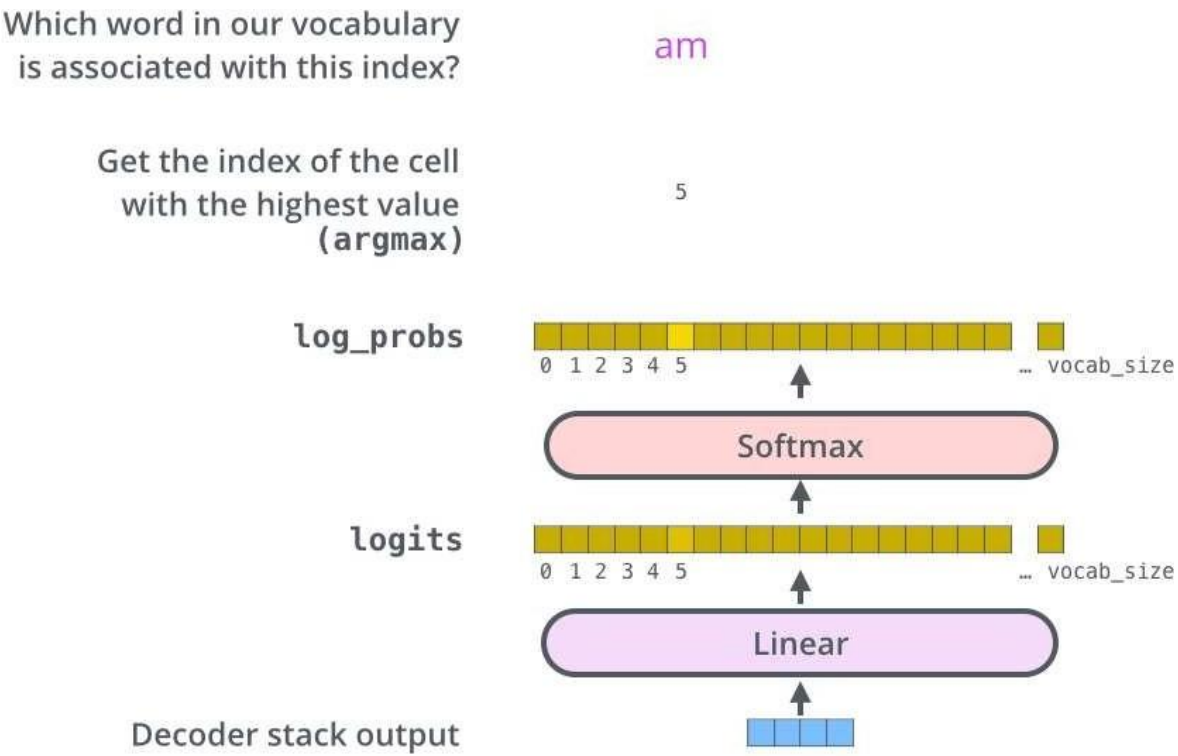
E/15/021

4. FC and Softmax layer



E/15/021

Final FC and softmax layer



E/15/021

Selecting model prediction

- When selecting model output, we can take the word with the highest probability and throw away the rest word candidates. : **greedy decoding**

- Another way to select model output is **beam-search**.

Beam-search

- **beam-search**
  - Instead of only predicting the token with the best score, we keep track of k hypotheses (for example k=5, we refer to k as the **beam size**).
  - At each new time step, for these k hypotheses, we have V new possible tokens. It makes a total of kV new hypotheses. Then, only keep top k hypotheses, … .
  - The length of words to hold is also a parameter.

Experiment- sequence to sequence task

- Data
  - WMT2014 English-German : 4.5 million sentence pairs
  - WMT2014 English-French : 36 million sentences
- Hardware and Schedule
  - 8 NVIDIA P100 GPUs
  - Base model : 100,000 steps or 12 hours
  - Big model : 300,000 steps (3.5 days)
- Optimizer : Adam
  - Warm up, and then decrease learning rate.

Regularization

Residual Dropout :

- apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks.

Label Smoothing :

- During training, employed label smoothing of value ls = 0.1. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

Experiment - Result

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

Conclusion

- They presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

- The Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.

Thank You