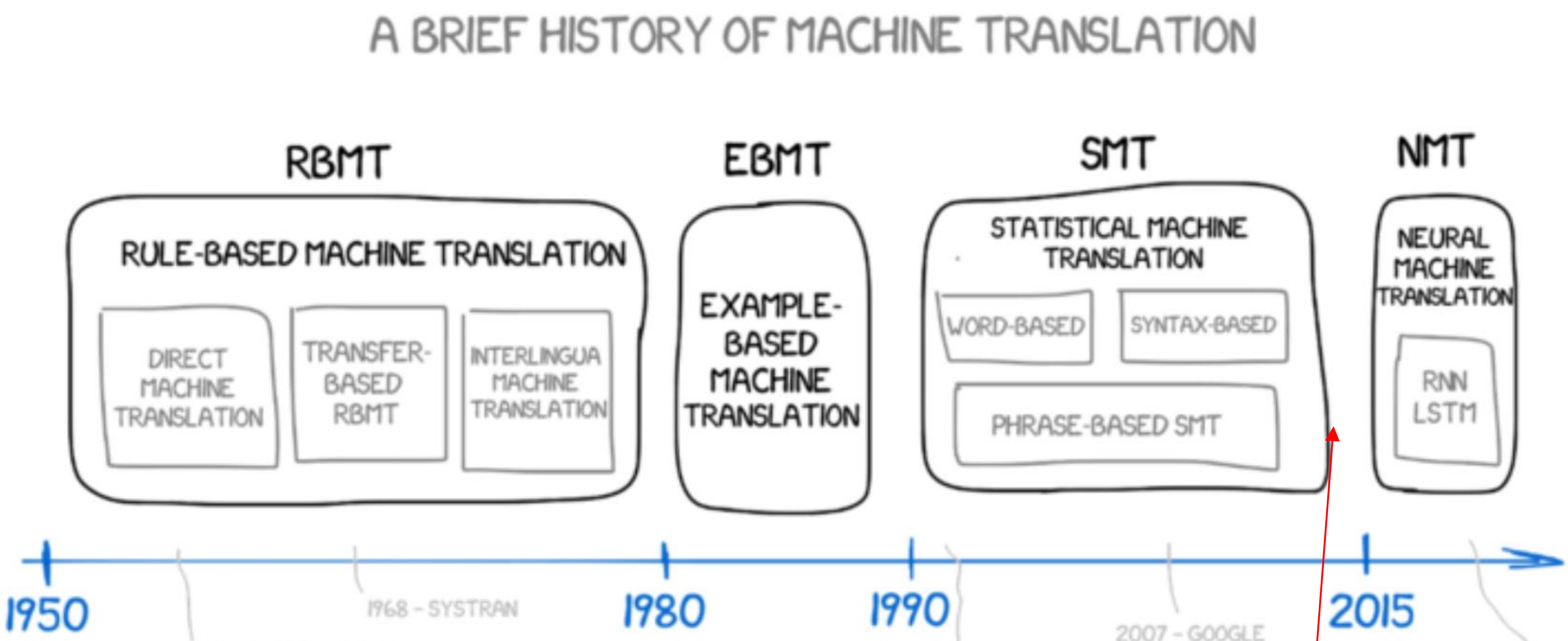
Transformer

노기섭 교수
(kafa46@hongik.ac.kr)

Goal of Machine Translation

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} P_{x \rightarrow y}(y|x)$$

History of Machine Translation



Almost all modern technologies are using NMT!!!

2014, Evolution of NMT:
Attention Mechanism → Seq2Seq

https://www.freecodecamp.org/news/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5/

GNMT from Google in 2016

2016, Google announced GNMT (Google Neural Machine Translation)

- Seq2Seq + Atte

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

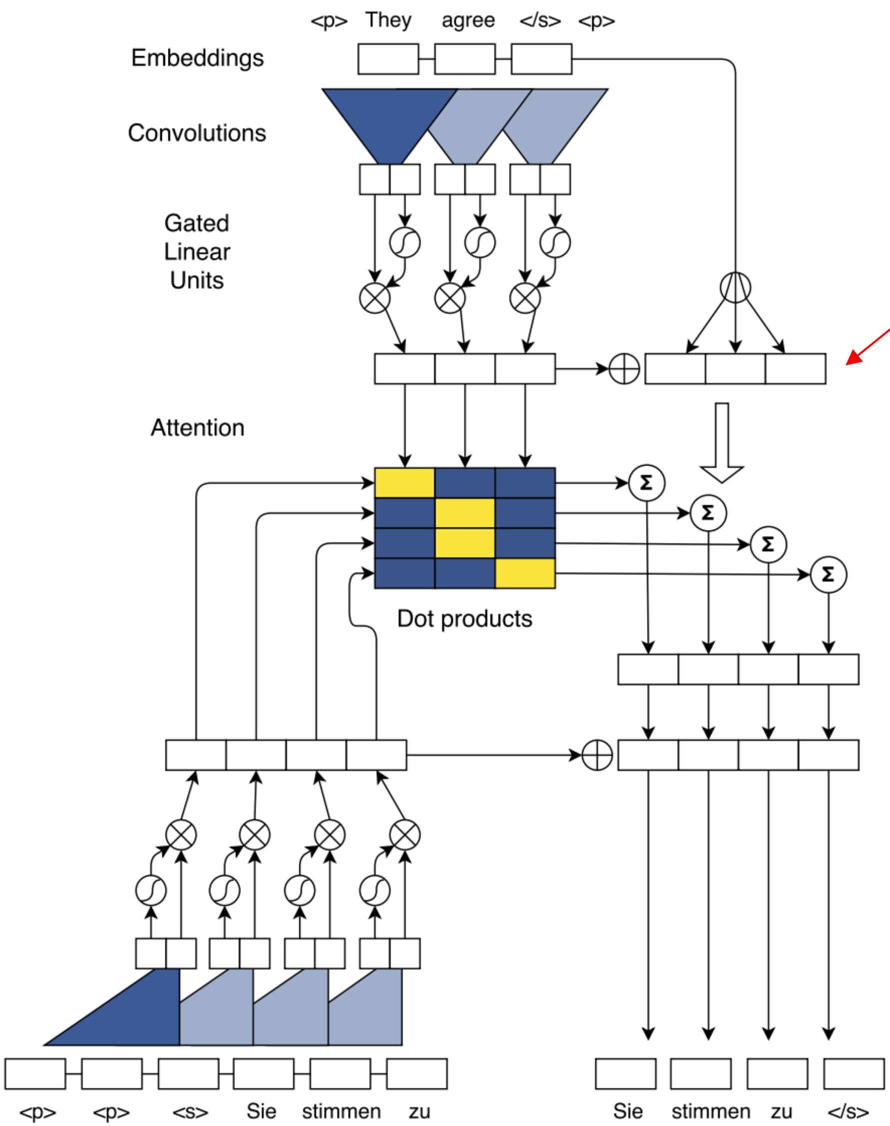
Paper Link: https://arxiv.org/pdf/1609.08144.pdf

Side-by-side (SxS) score
- Human evaluation
- Range = [0, 6]
  - 6: perfect translation
  - 0: nonsense

Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

Fully Convolutional Seq2Seq from Facebook in 2017



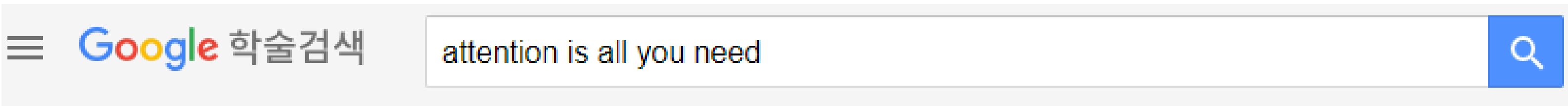
Paper Link:
https://arxiv.org/pdf/1705.03122.pdf

WMT'14 English-German	BLEU
Luong et al. (2015) LSTM (Word 50K)	20.9
Kalchbrenner et al. (2016) ByteNet (Char)	23.75
Wu et al. (2016) GNMT (Word 80K)	23.12
Wu et al. (2016) GNMT (Word pieces)	24.61
ConvS2S (BPE 40K)	25.16

WMT'14 English-French	BLEU
Wu et al. (2016) GNMT (Word 80K)	37.90
Wu et al. (2016) GNMT (Word pieces)	38.95
Wu et al. (2016) GNMT (Word pieces) + RL	39.92
ConvS2S (BPE 40K)	40.51

After one month of ConvS2S, Transformer comes!

- **Same structure of Seq2Seq, but only Attention Mechanism**



Please, click & check it out!

https://scholar.google.com/scholar?hl=ko&as_sdt=0%2C5&q=attention+is+all+you+need

Paper Link: https://arxiv.org/pdf/1706.03762.pdf

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**	$2.3 \cdot 10^{19}$	

Now, everything is Transformers!

References

- Beautiful Sources
  - Blog: [Jay Alammar]

- **Paper Link**
  - https://arxiv.org/abs/1706.03762

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Overview

Transformer: Attention only Mechanisim

January 16, 2013

WORD2VEC

Word2Vec Paper by Tomas Mikolov et al

January 2, 2014

GLOVE

GloVe: Global Vectors for Word Representation by Jeffrey Pennington et al.

July 15, 2016

FASTTEXT

Enriching Word Vectors with Subword Information by Piotr Bojanowski et al

June 12, 2017

TRANSFORMER

Attention Is All You Need by Ashish Vaswani et al

February 15, 2018

ELMO

Deep contextualized word representations by Matthew E. Peters et al

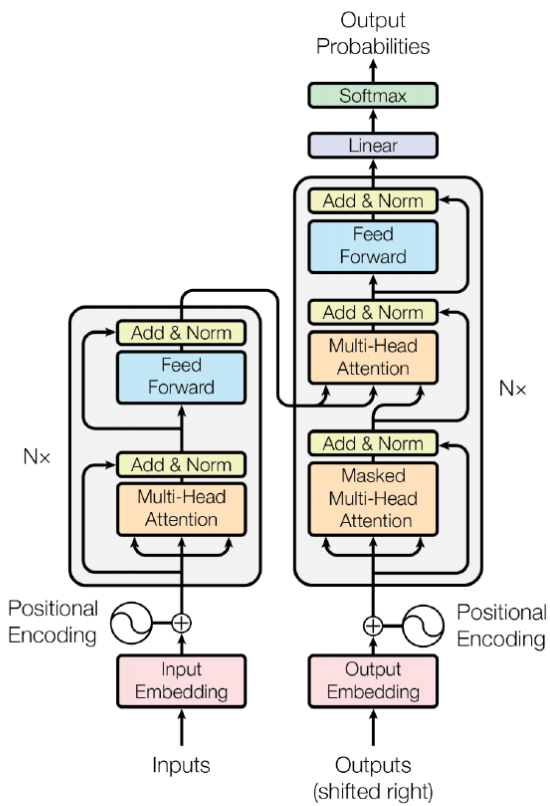
October 11, 2018

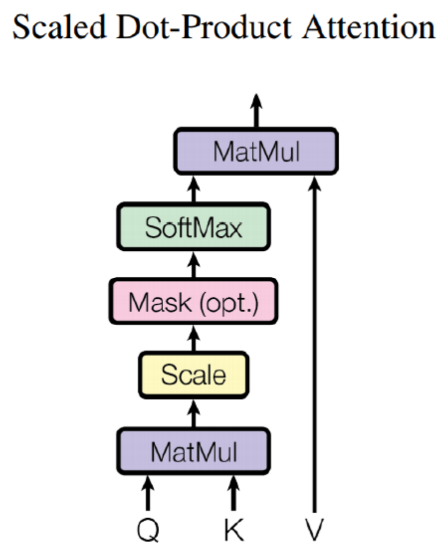
BERT

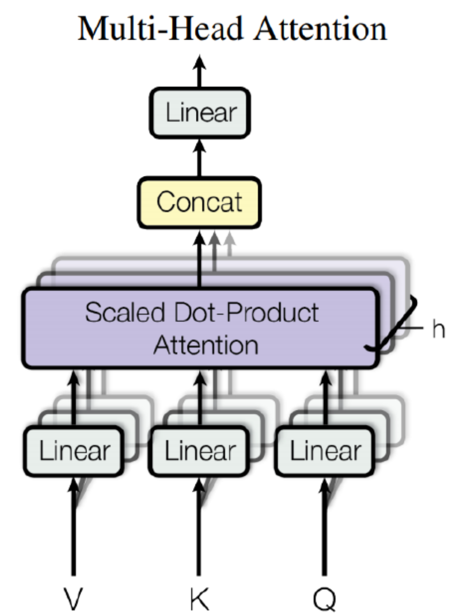
BERT: Pre-training of Deep Bidirectional Transformers for...

Overview

- **Transformer (Vaswani et al., 2017, Google)**
  - A model that uses attention to boost the speed with which these models can be trained and easy to parallelize





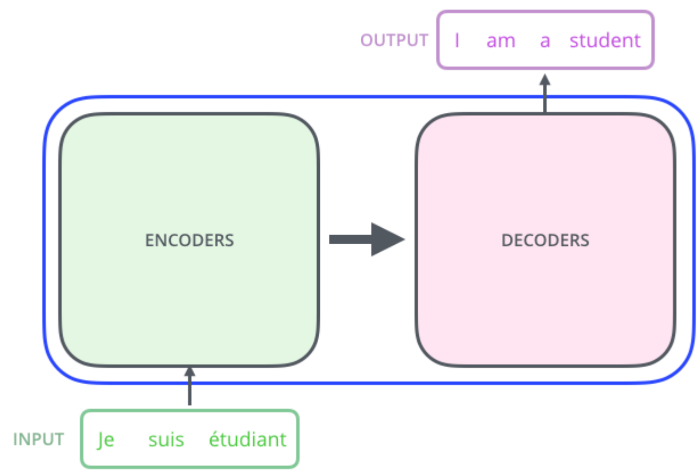


High Level View

- A model that uses attention to boost the speed with which these models can be trained and easy to parallelize
- A high level look

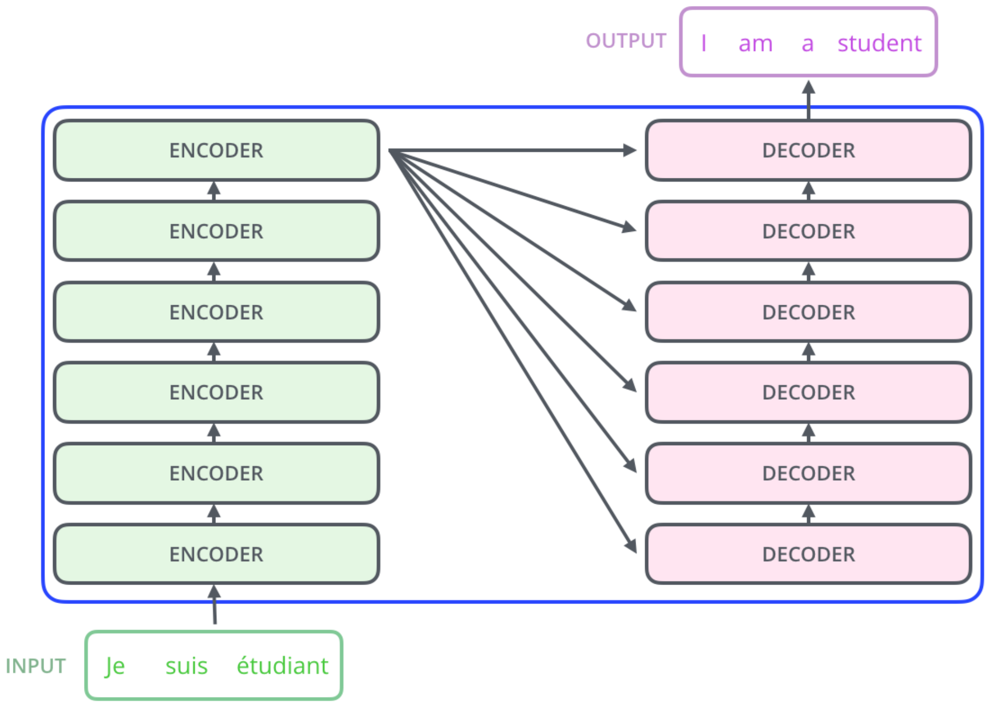
- Inside the transforme

components and connections between them



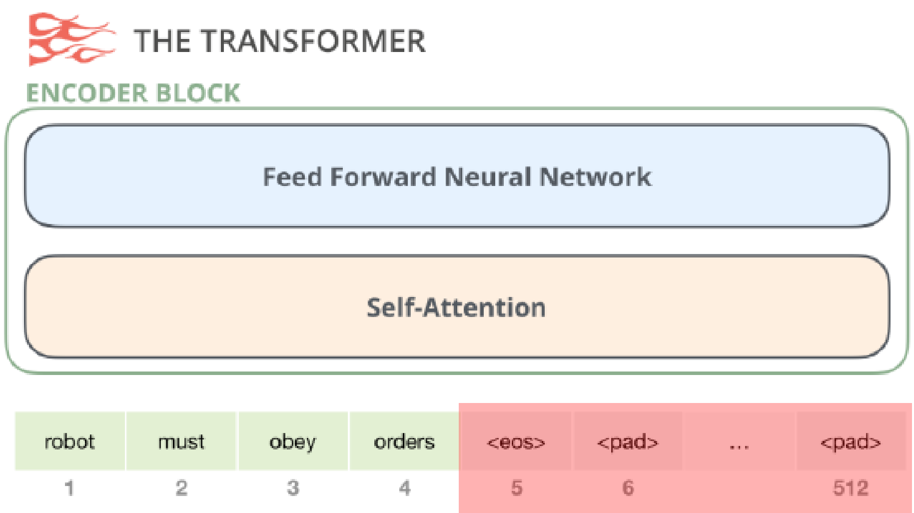
Encoder-decoder Stacking Structure

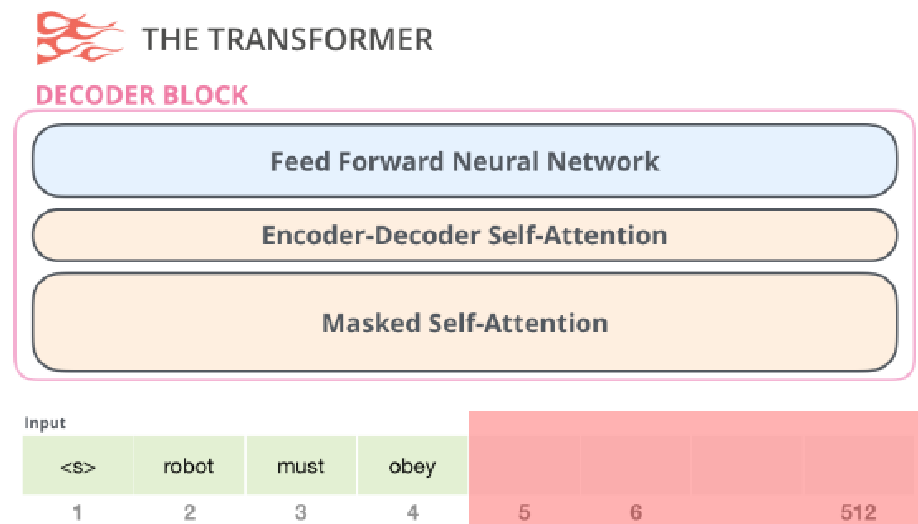
- Stack of encoders & decoders
  - The original paper stacks six of them on top of each other, but there is nothing magical about the number six
  - The decoding component is a stack of decoders of the same number



Difference between Encoder & Decoder

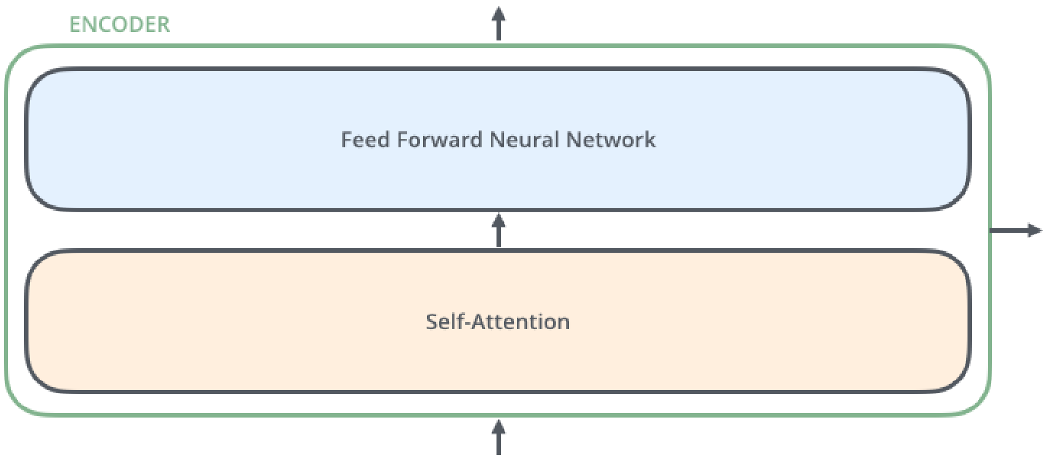
- Encoding block vs. Decoding block == Unmasked vs. Masked





Encoder Structure

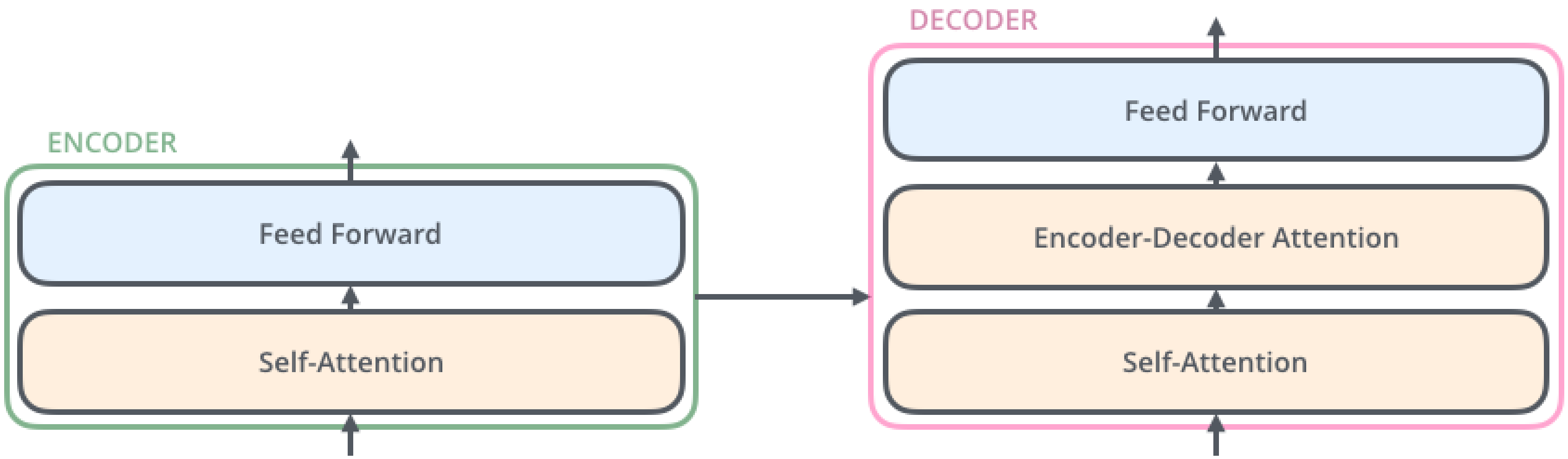
- The encoder are all identical in structure (does not mean that they share the weights), each of which is broken down into two sub-layers



  - The … t helps the encoder look … d)
  - The output of the self-attention layer are fed to a feed-forward neural network
    - The exact same feed-forward network is independently applied to each position

Decoder Structure

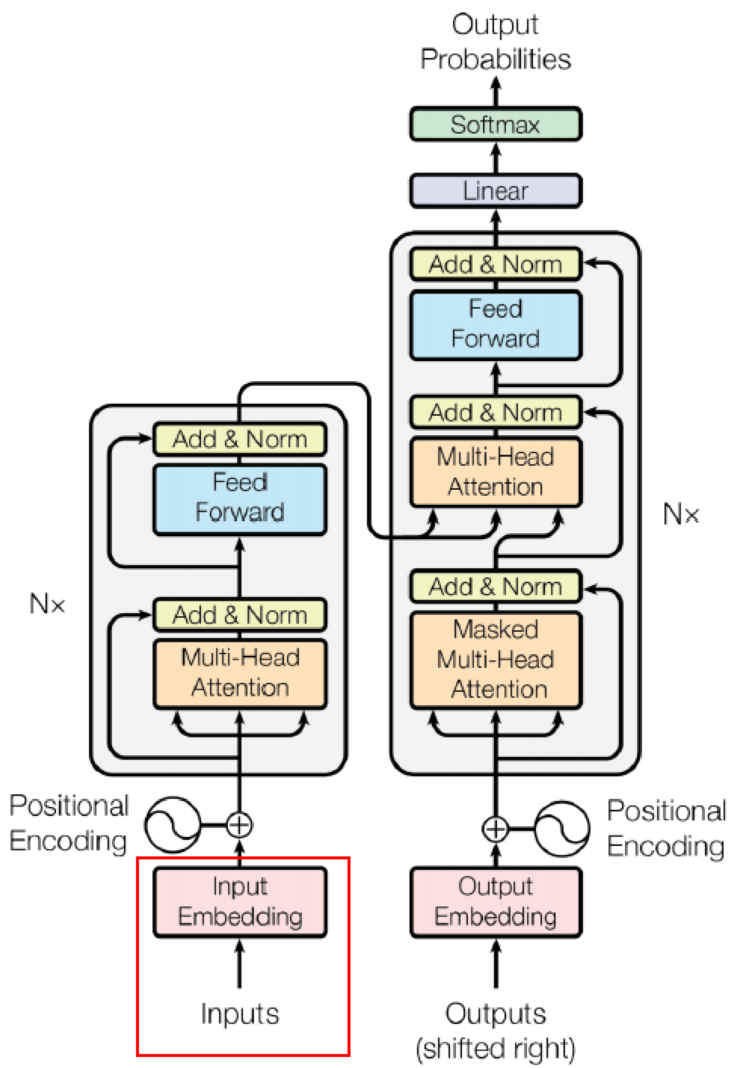
- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



Input Embedding

Input Embedding

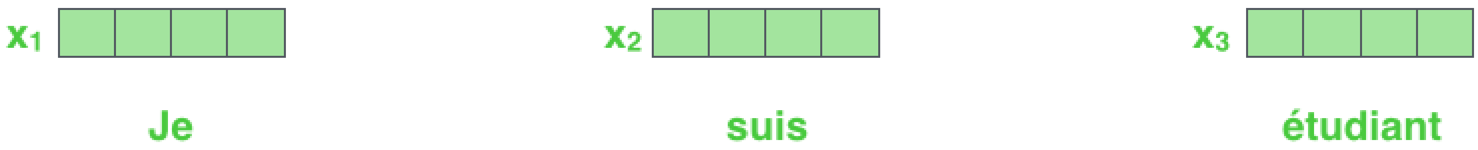
Embedding



Input Embedding

More specific explanation

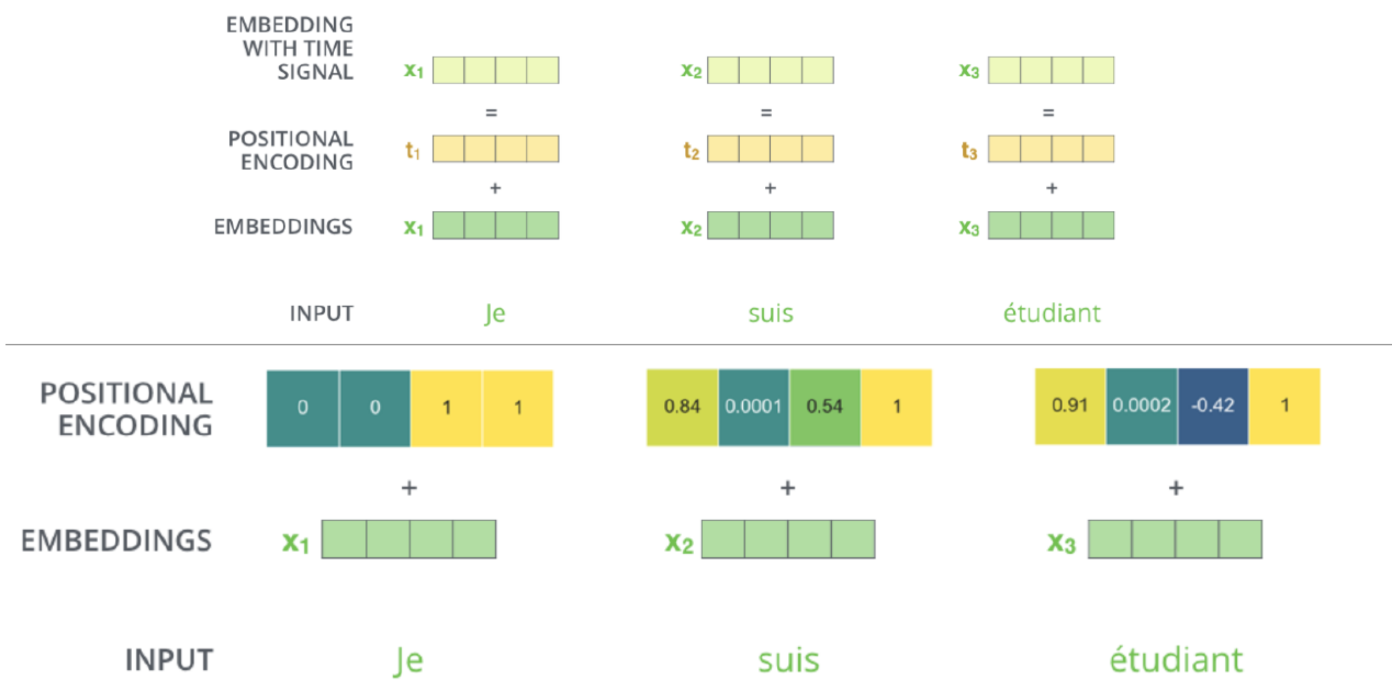
- Let's begin by turning each input word into a vector using an embedding algorithm



- The embedding only happens in the bottom-most encoder
- The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512
- In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that is directly below
- The size of this list is a hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset

Positional Encoding

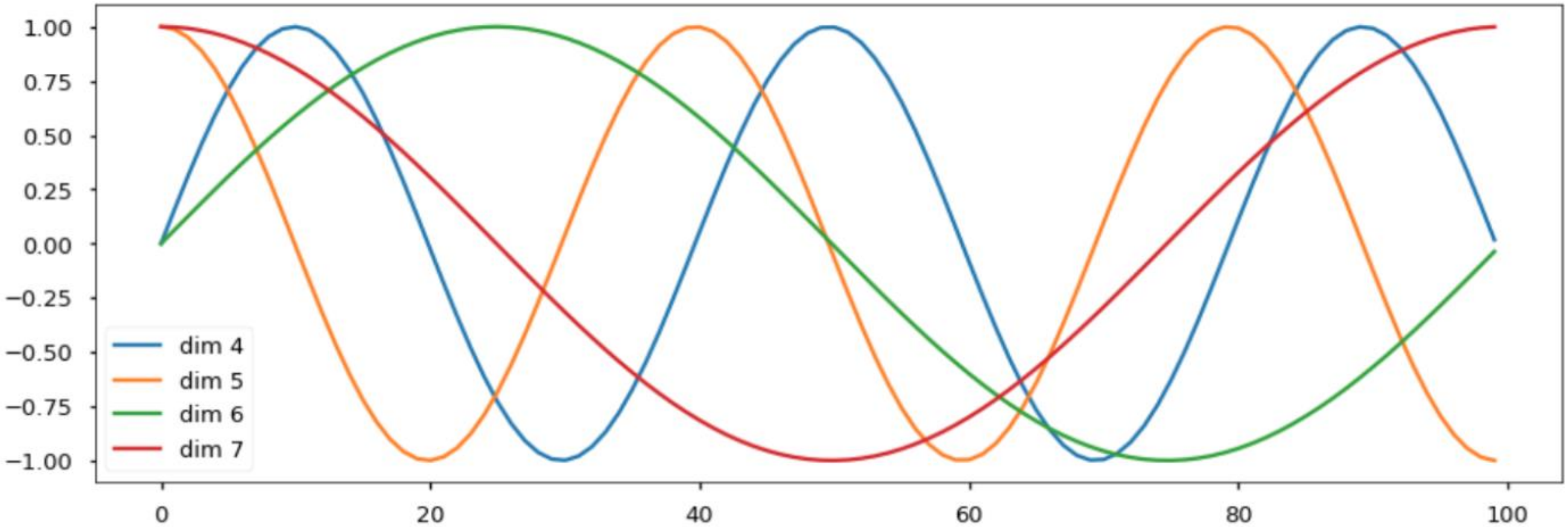
- A way to account for the order of the words in the input sequence
- A vector added to each input embedding
  - Provides meaningful distances between the embedding vectors once they are projected into Q/K/V vectors and during dot-product attention



Positional Encoding

Below the positional encoding will add in a sine wave based on position. The frequency and offset of the wave is different for each dimension.

```
plt.figure(figsize=(15, 5))
pe = PositionalEncoding(20, 0)
y = pe.forward(Variable(torch.zeros(1, 100, 20)))
plt.plot(np.arange(100), y[0, :, 4:8].data.numpy())
plt.legend(["dim %d"%p for p in [4,5,6,7]])
None
```



Data Source: https://nlp.seas.harvard.edu/2018/04/03/attention.html

Positional Encoding - Required Properties

- Two properties that a good positional encoding scheme should have
  - The norm of encoding vector is the same for all positions
  - The further the two positions, the larger the distance
    - A Simple Example (n = 10, dim = 10)

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

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

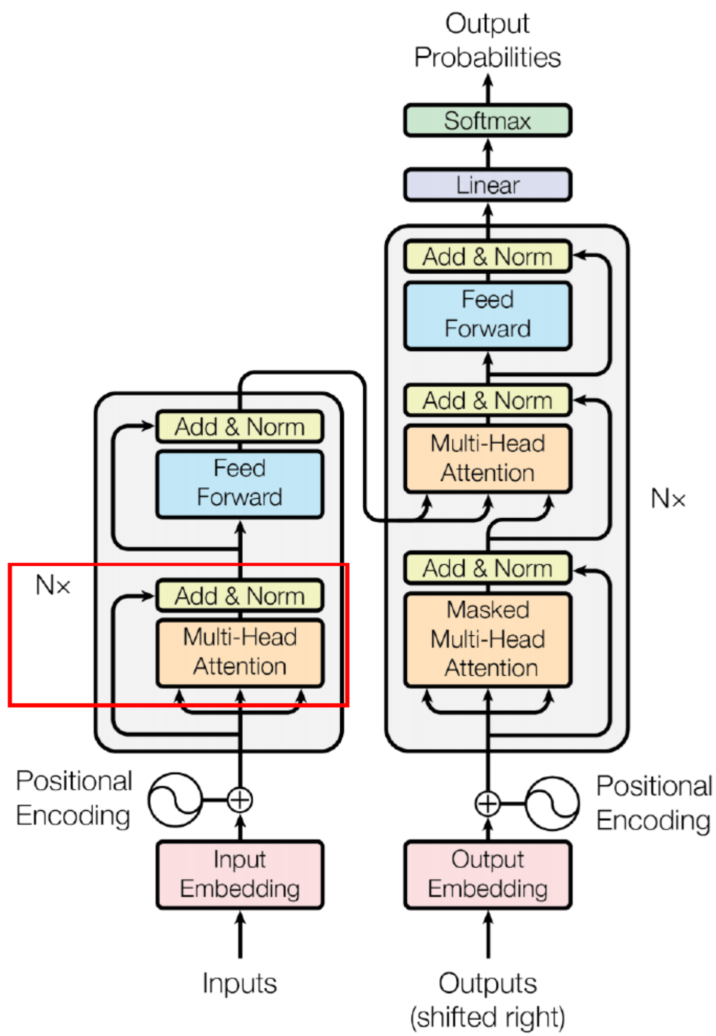
Distances between two positional encoding vectors

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

Self Attentions

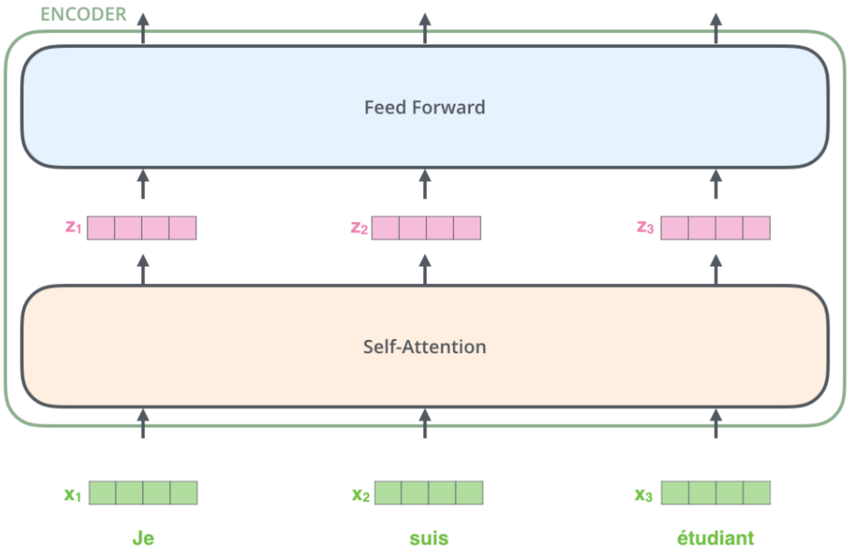
Self-attention

Area of Self-attention



Layers

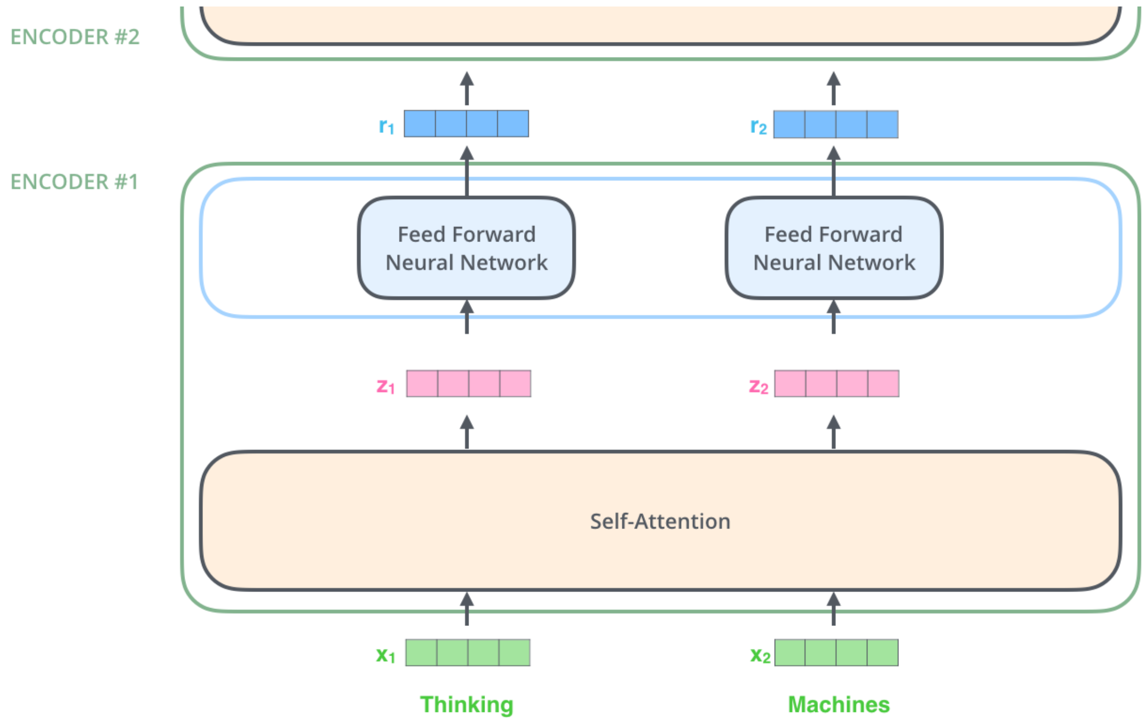
- After embedding the words, each of them flows through each of the two layers of the encoder



- Word in each position flows through its own path in the encoder
    - There are dependencies between these paths in the self-attention layer
    - The feed-forward layer does not have those dependencies (parallelization becomes possible)

Encoding Procedure

- An encoder receives a list of vectors as input
- It processes this list by passing these vectors into a 'self-attention' layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder



Self-Attention at a High Level

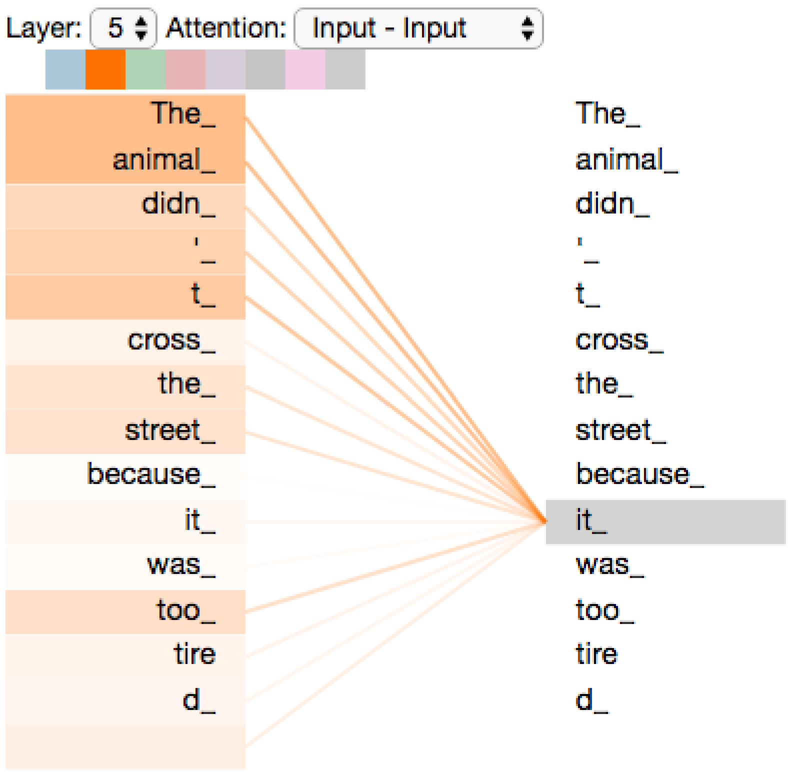
- Input sentence to translate:

  `The animal did’t cross the street because it was too tired`

  - What does “`it`” refer to? `street` or `animal`?
  - Simple question to a human but not as simple to an algorithm

- Self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word
- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing

Self-Attention example

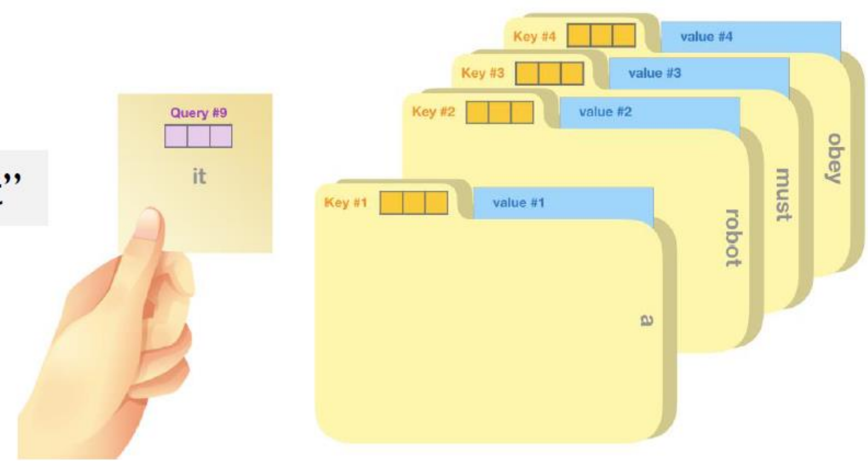


Source codes:
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Self-Attention - Step 1

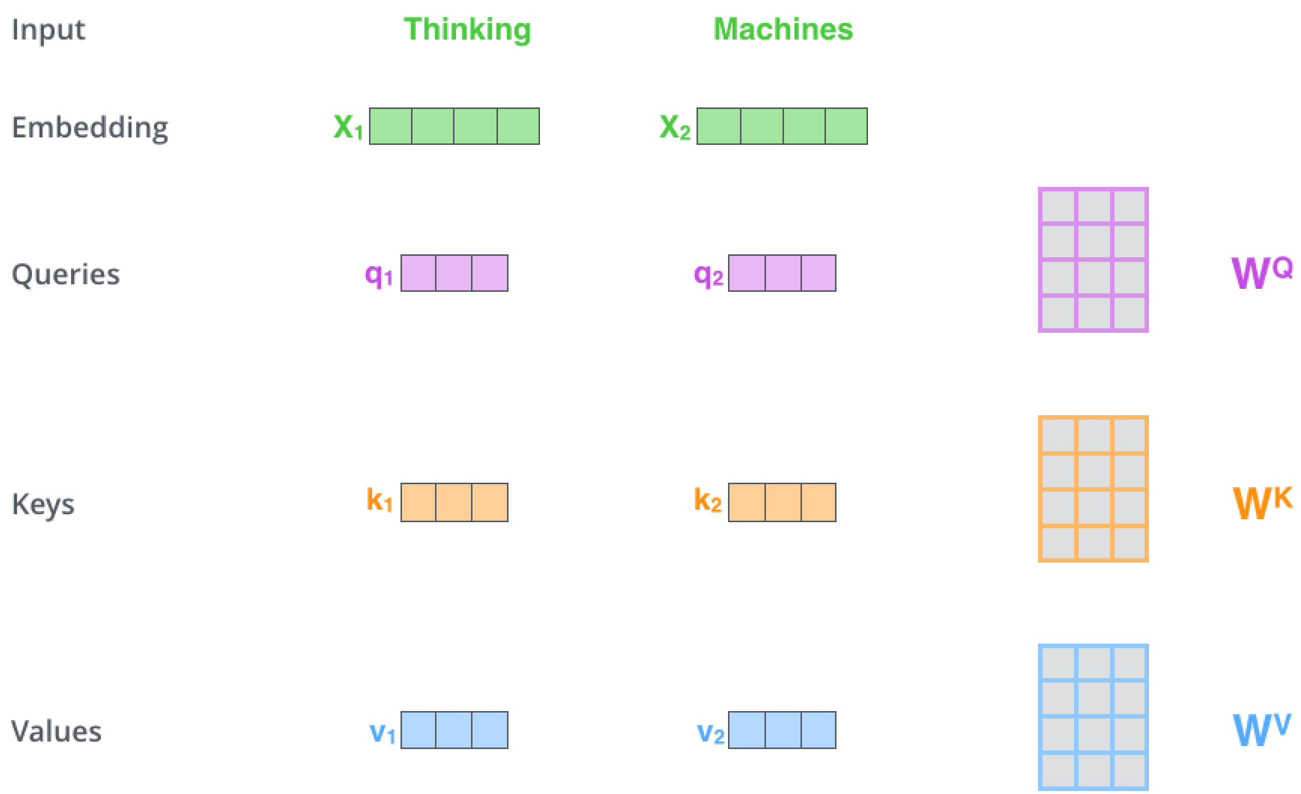
- Step 1: Create three vectors from each of the encoder's input vectors
  - Query: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
  - Key: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
  - Value: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

"A robot must obey the orders given it"



Self-Attention - Step 1 (cont.)

- Step 1: Create three vectors from each of the encoder's input vectors
  - These vectors are created by multiplying the embedding by three matrices that we trained during the training process

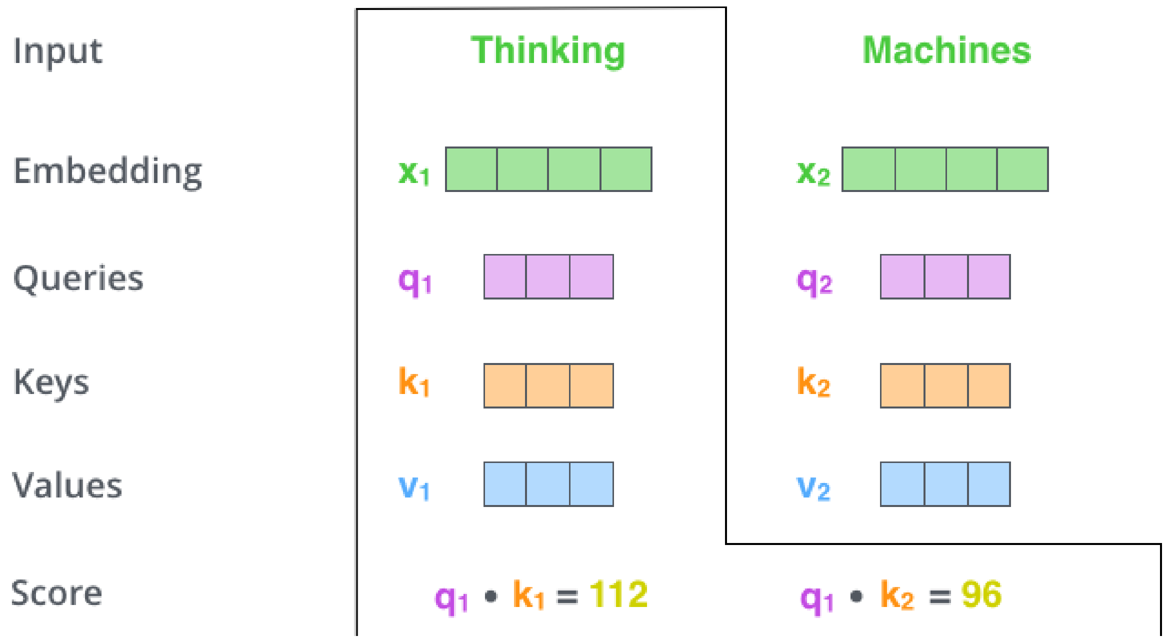


Self-Attention - Step 1 (cont.)

- Step 1: Create three vectors from each of the encoder's input vectors
  - Note) These new vectors are smaller in dimension that the embedding vector
    - Q, K, and V are 64-dim. while embedding and encoder input/output vectors are 512-dim.
    - They do not have to be smaller, but it is an architecture choice to make the computation of multi-headed attention (mostly) constant

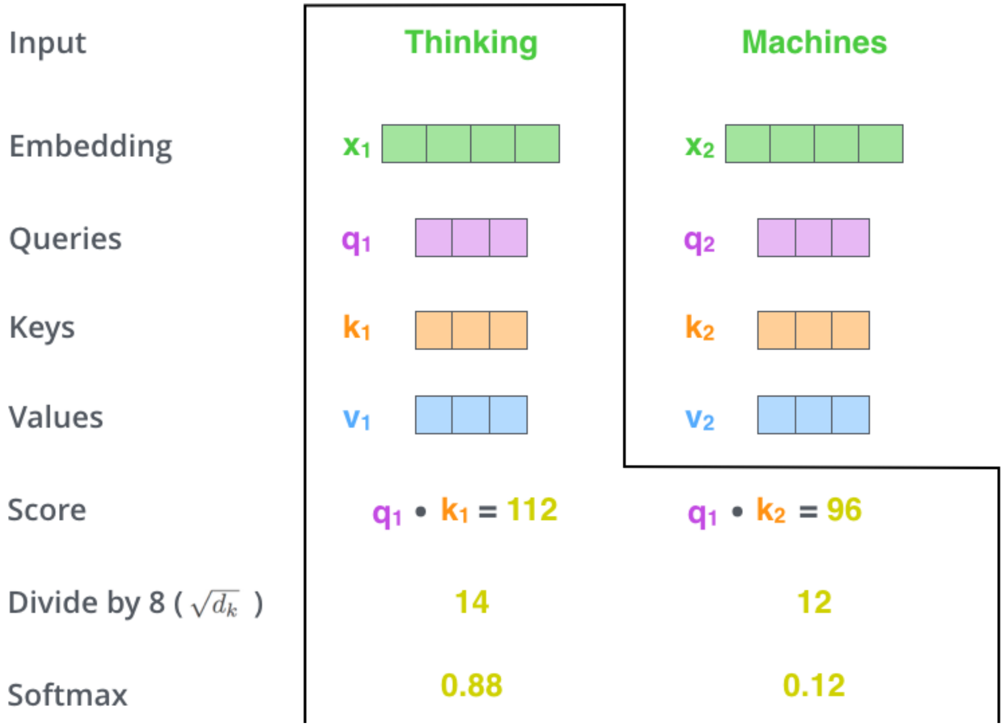
Self-Attention - Step 2

- Step 2: Calculate a score, i.e., how much focus to place on other parts of the input sentence as we encode a word at a certain position
  - The score is calculated by taking the dot product of the query vector with the key vector of the respective word we are scoring



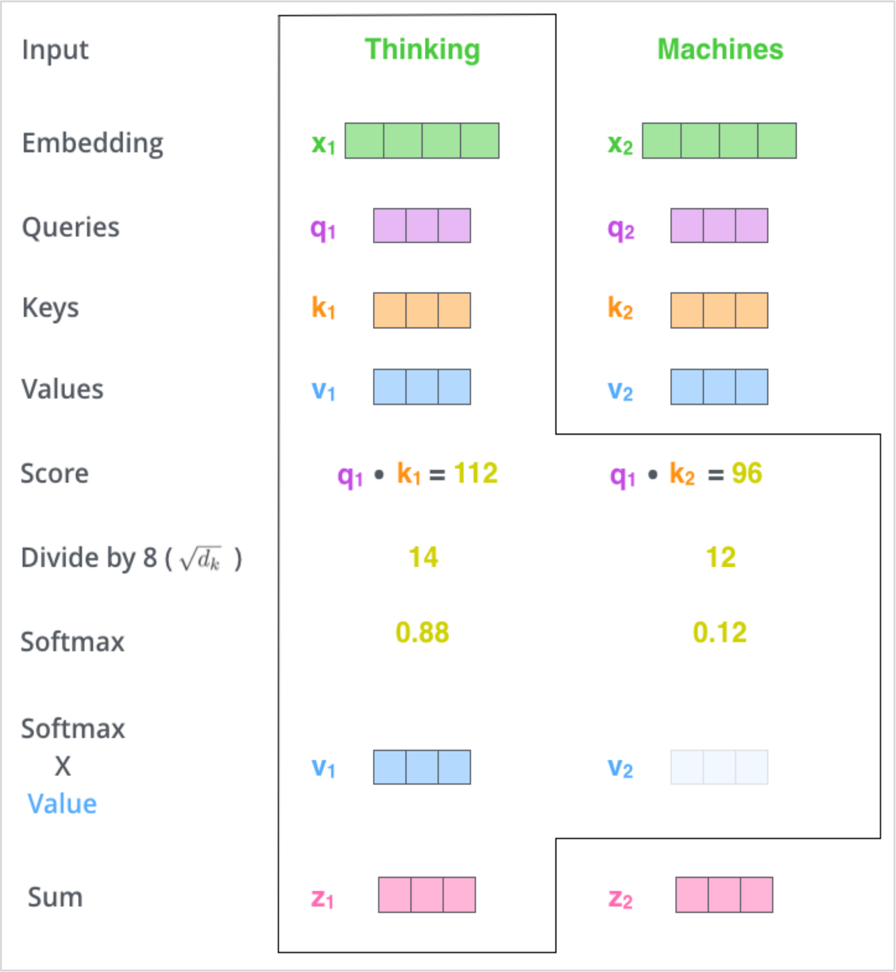
Self-Attention - Step 3, 4

- Step 3: Divide the score by $\sqrt{d_k}$ (= 8 in the original paper since d_k= 64)
  - This leads to having more stable gradients
- Step 4: Pass the result through a softmax operation
  - The softmax score determines how much each word will be expressed at this position



Self-Attention - Step 5

- Step 5: Multiply each value vector by the softmax score
  - to keep intact the values of the words we want to focus on
  - drown-out irrelevant words

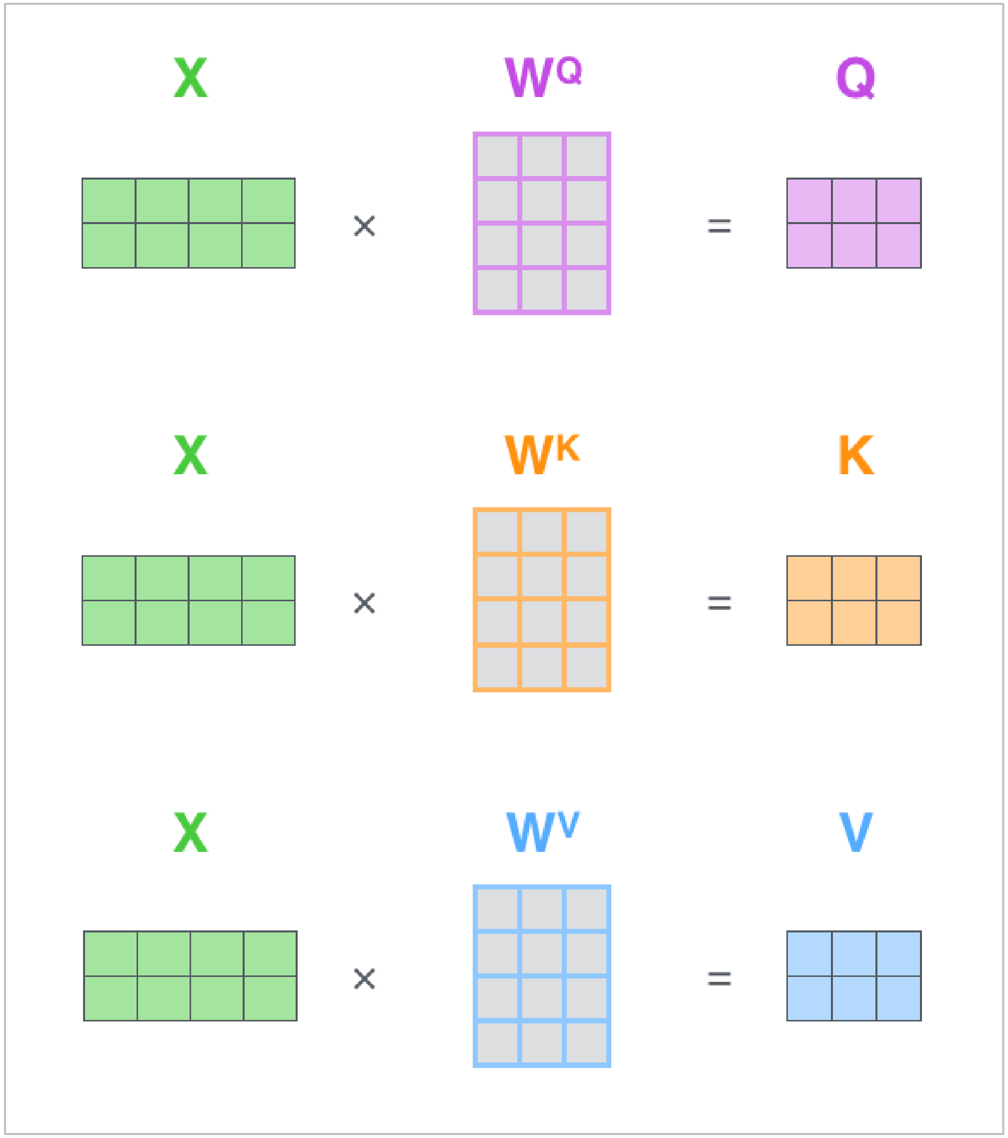


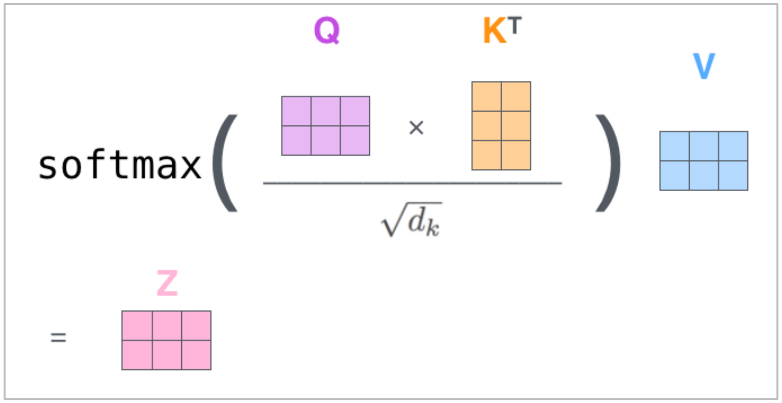
Self-Attention - Step 6

- Step 6: Sum up the weighted value vector which produces the output of the self-attention layer at this position

Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

Matrix Calculations of self-attention

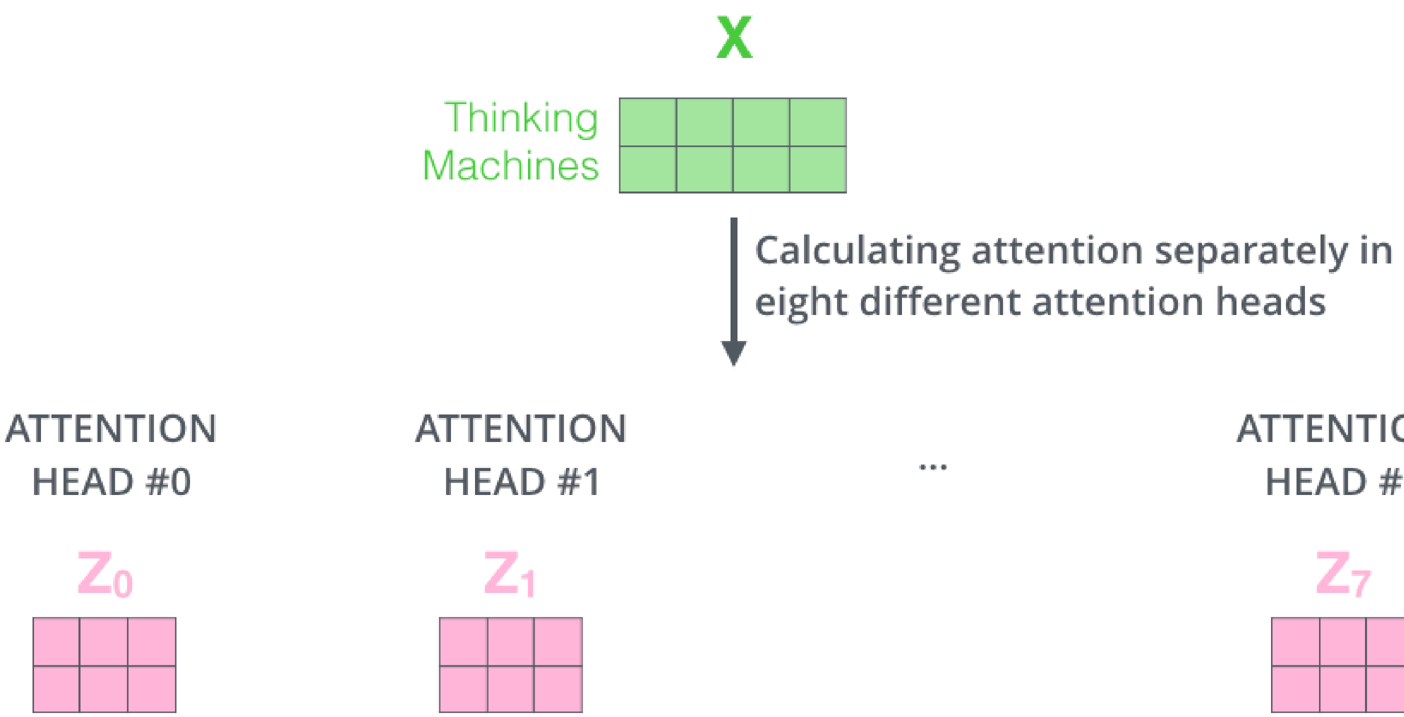




Multi-head Attentions

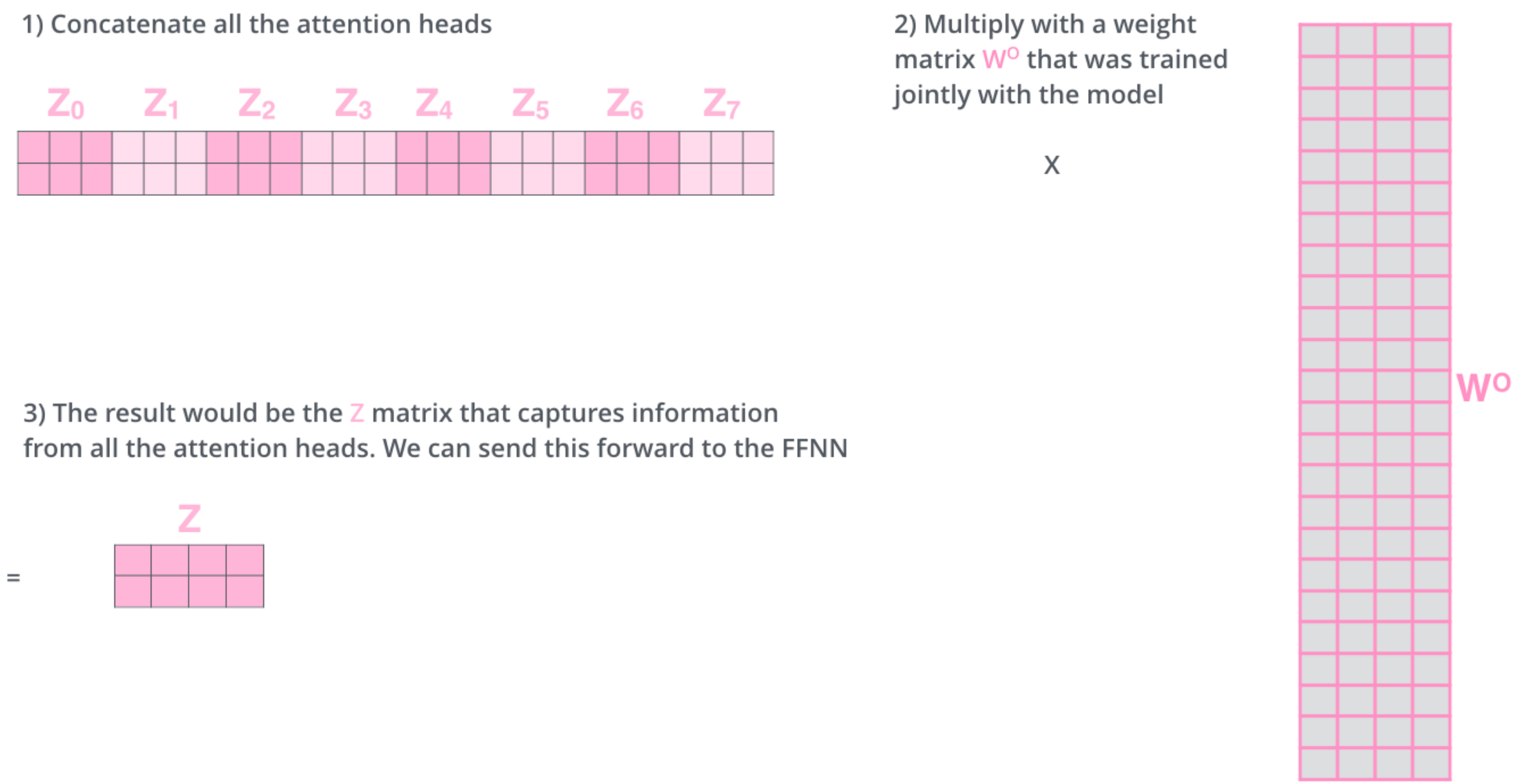
Multi-head Attention

- Expand the model's ability to focus on different positions



Multi-head Attention

- Attention heads are concatenated and multiplied by an additional weight matrix to be used as an input of feed-forward neural network



Multi-head Attention

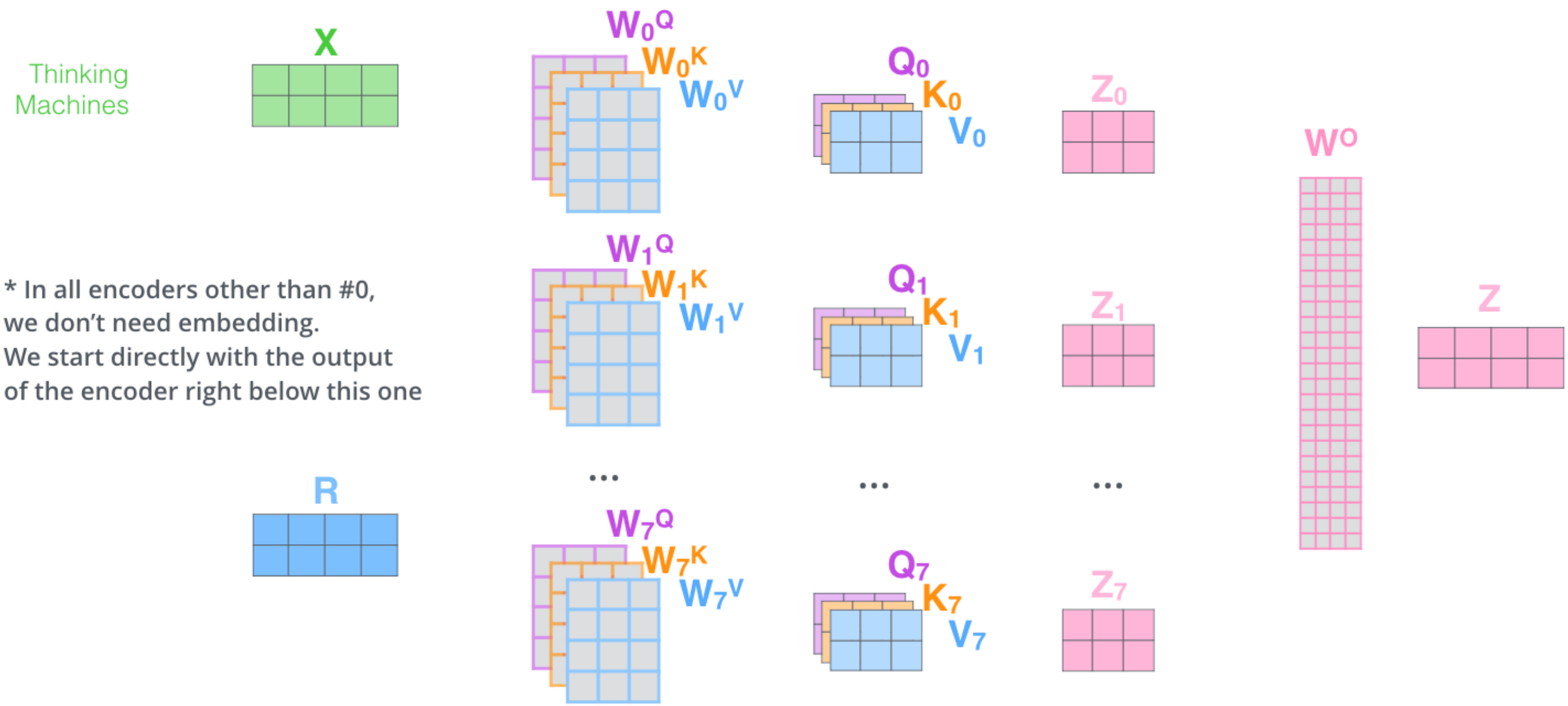
1) This is our input sentence*

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

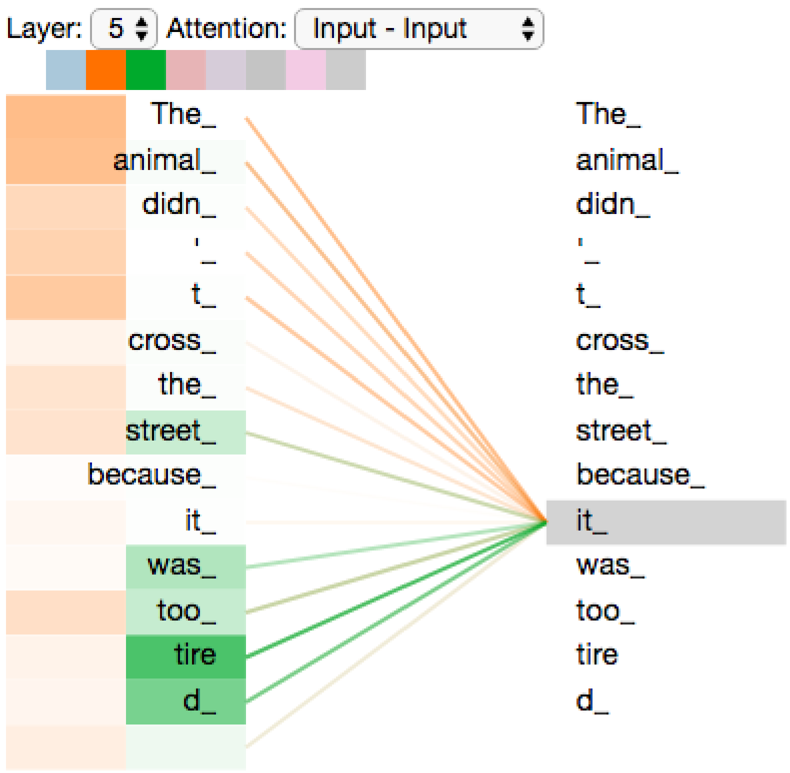
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



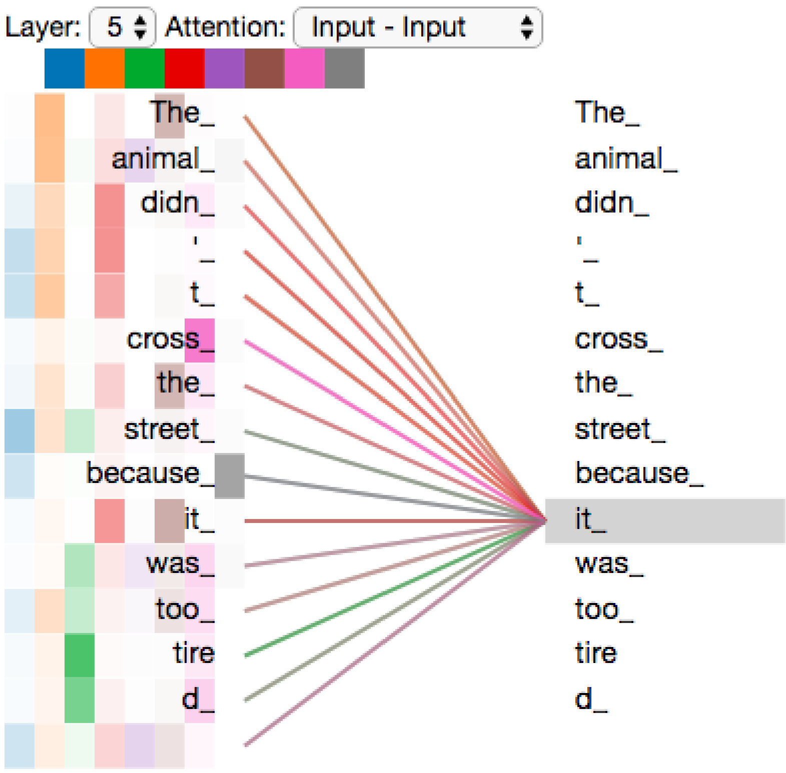
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Multi-head Attention

Attention with two heads



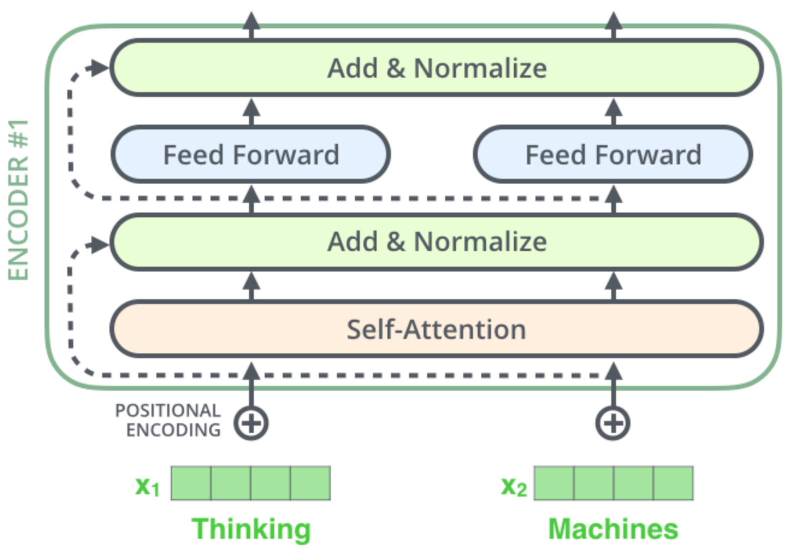
Attention with eight heads

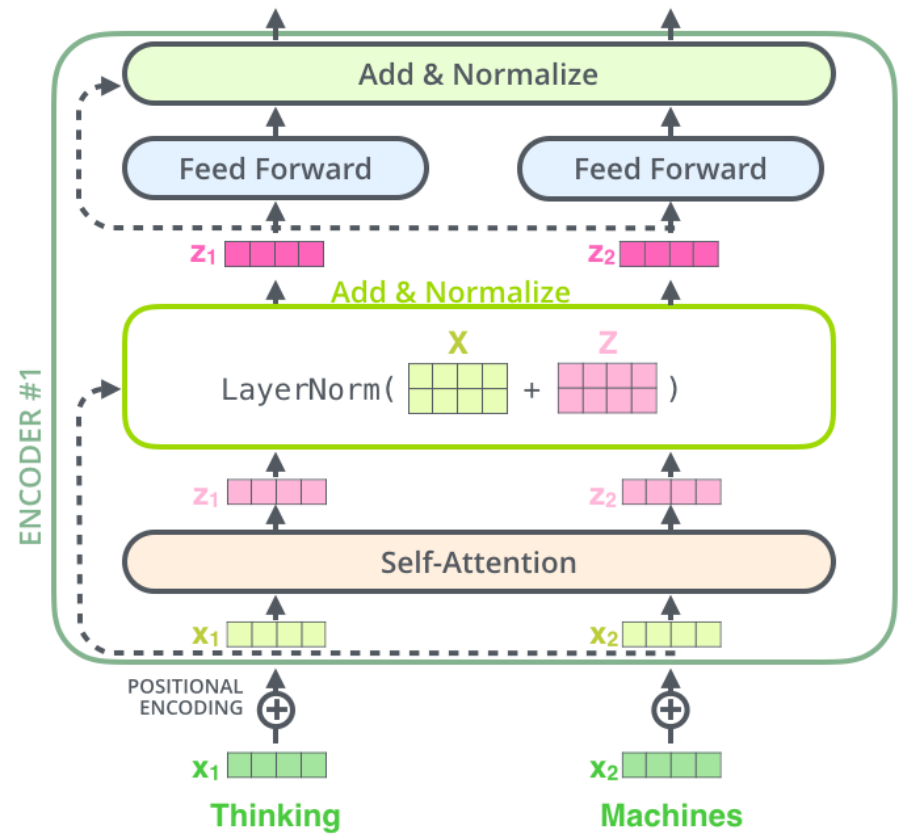


Residual Connections

Residual Connections

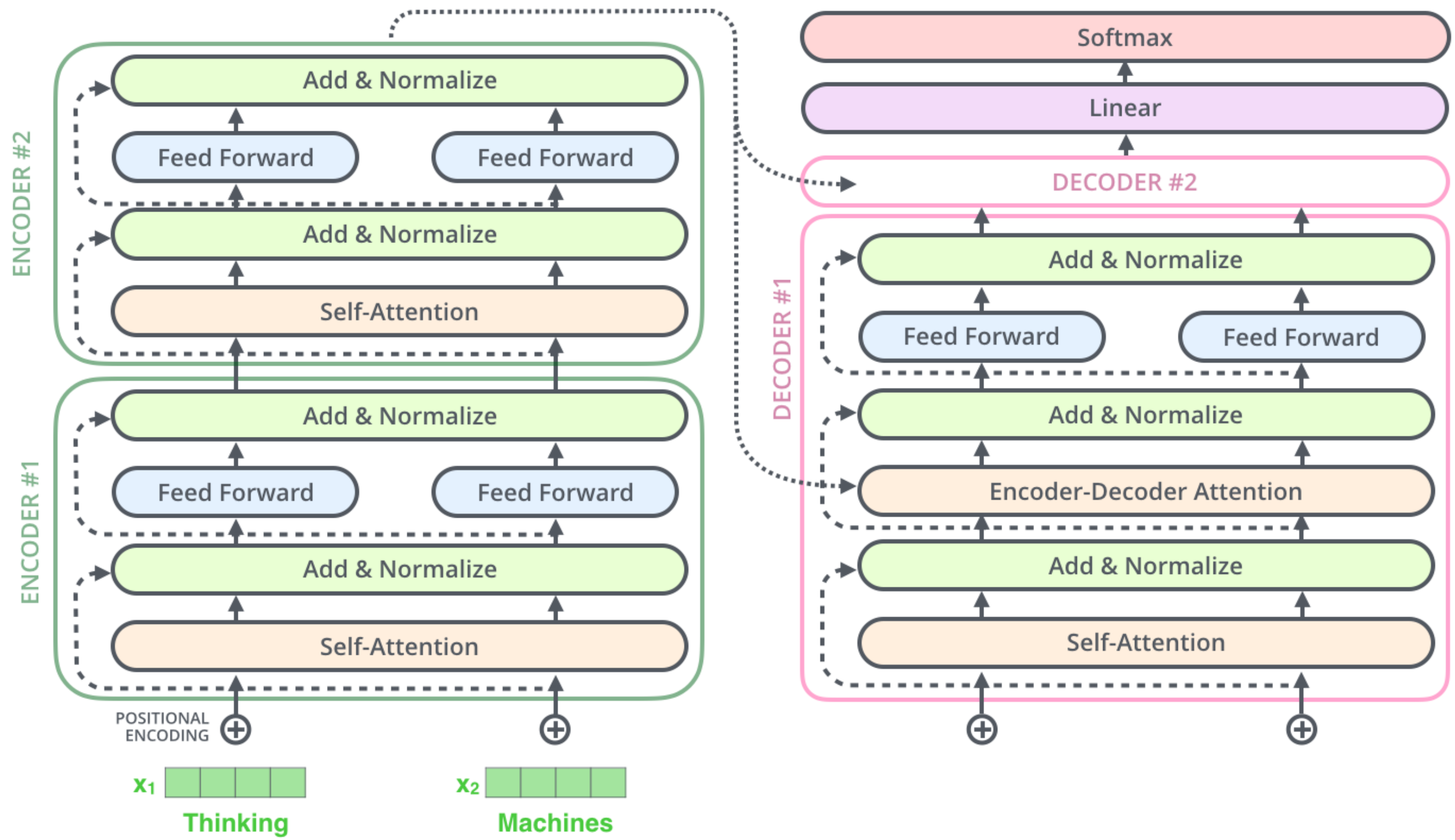
- Each sub-layer (self-attention, FFNN) in each encoder has a residual connection around it followed by a layer-normalization step





Residual Connections

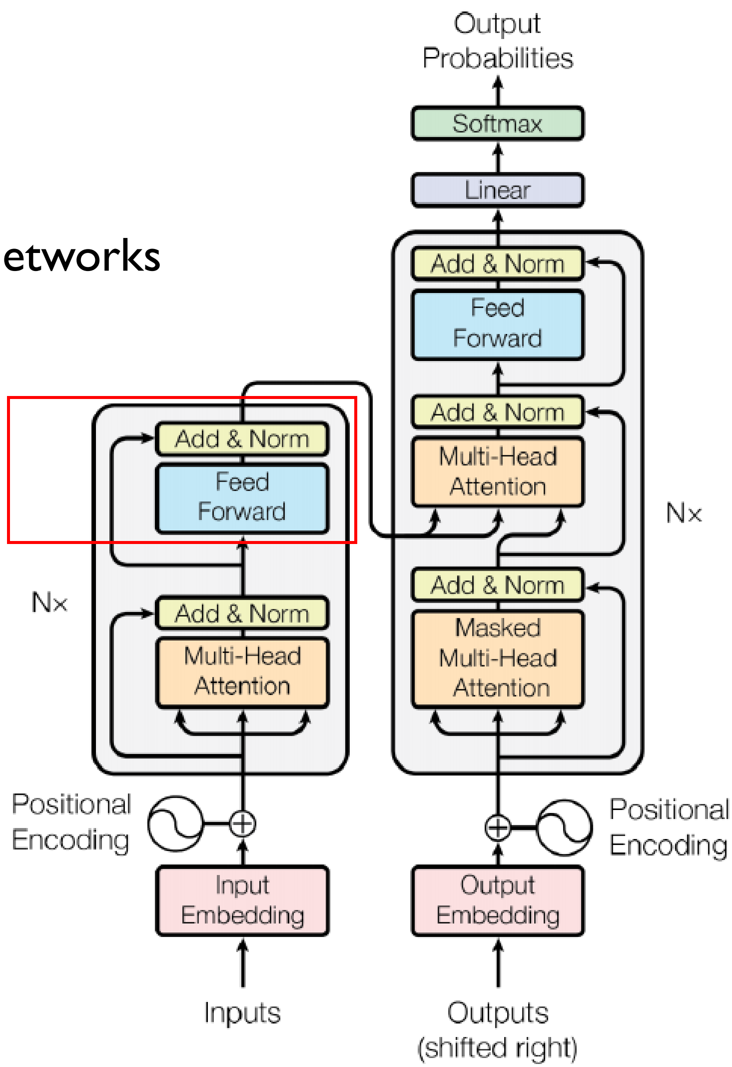
- This goes for the sub-layers of the decoder as well
  - Ex: 2 stacked encoders and decoders



Position-wise FF Networks

Position-wise FF Networks

Position-wise Feed-Forward Networks

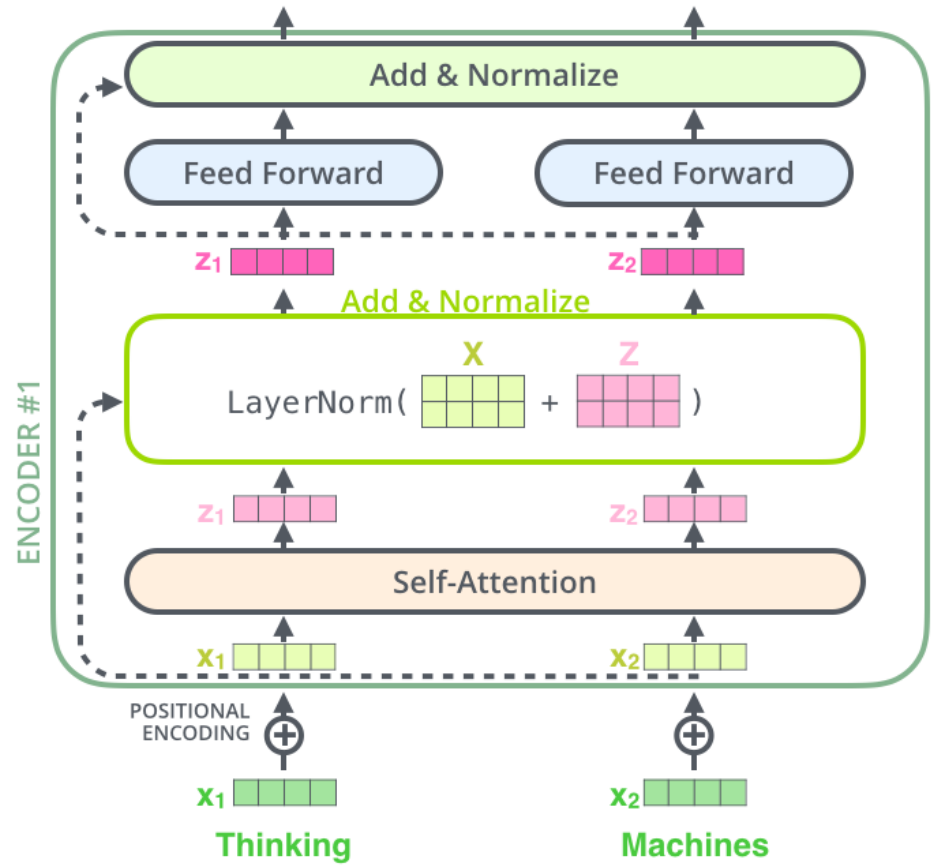


Position-wise FF Networks

- Position-wise Feed-Forward Networks
  - Fully connected feed-forward network
  - Applied to each position separately and identically

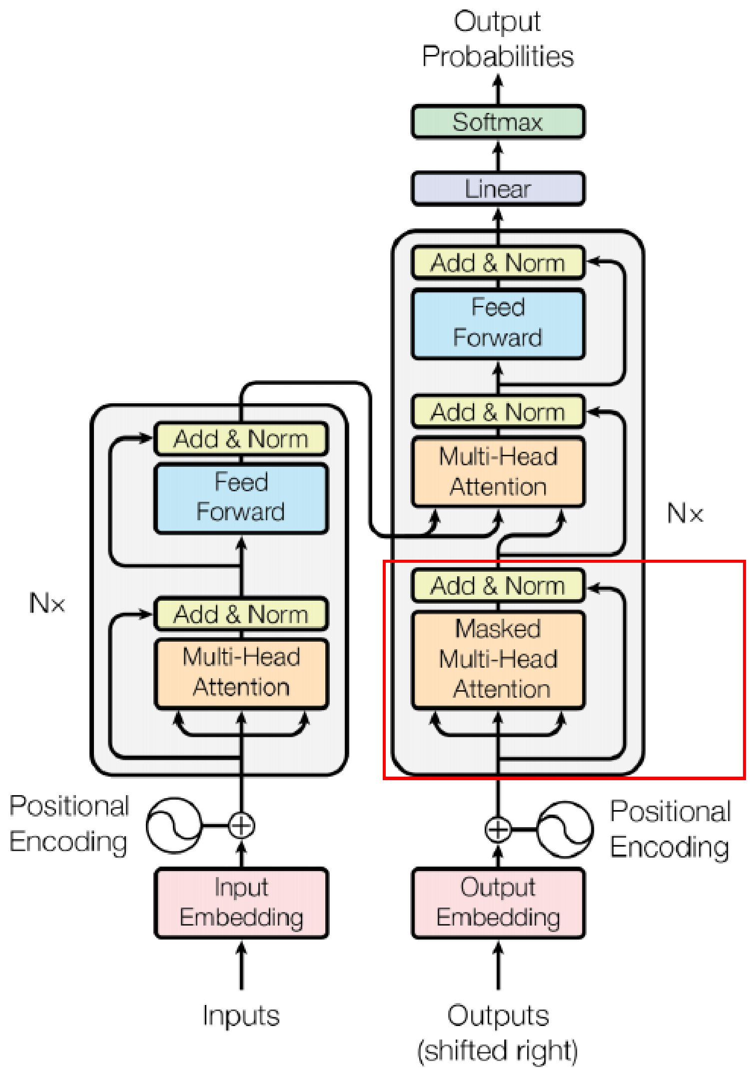
    $$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

  - The linear transformations are the same across different positions
  - They use different parameters from layer to layer



Masked Multi-head Attention

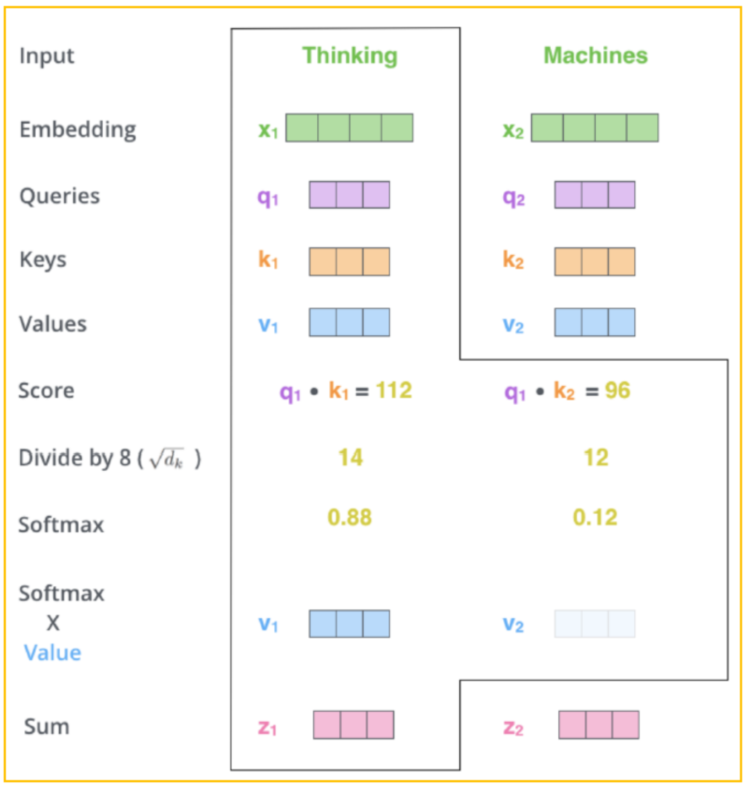
Masked Multi-head Attention

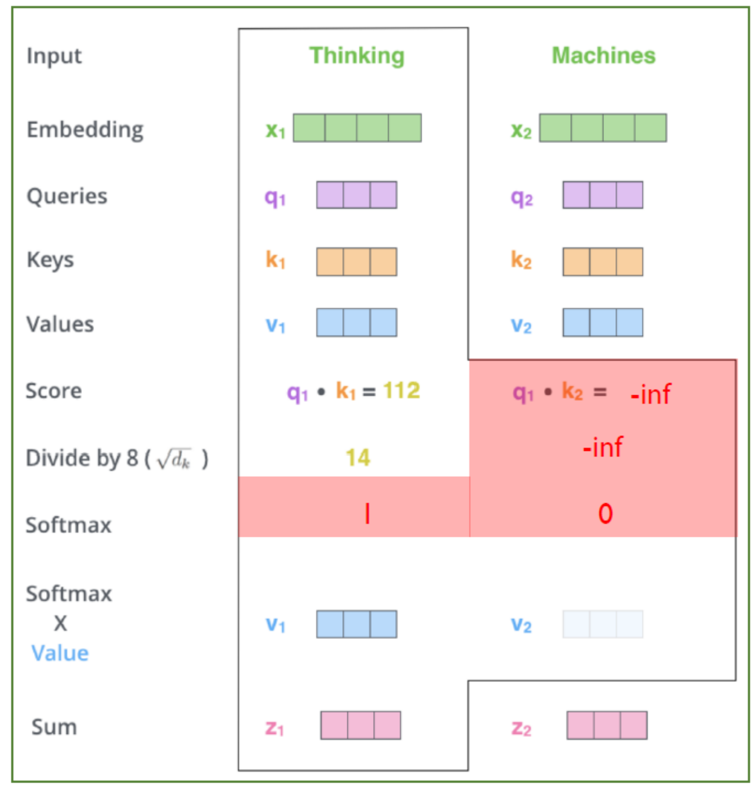


Masked Multi-Head Attention

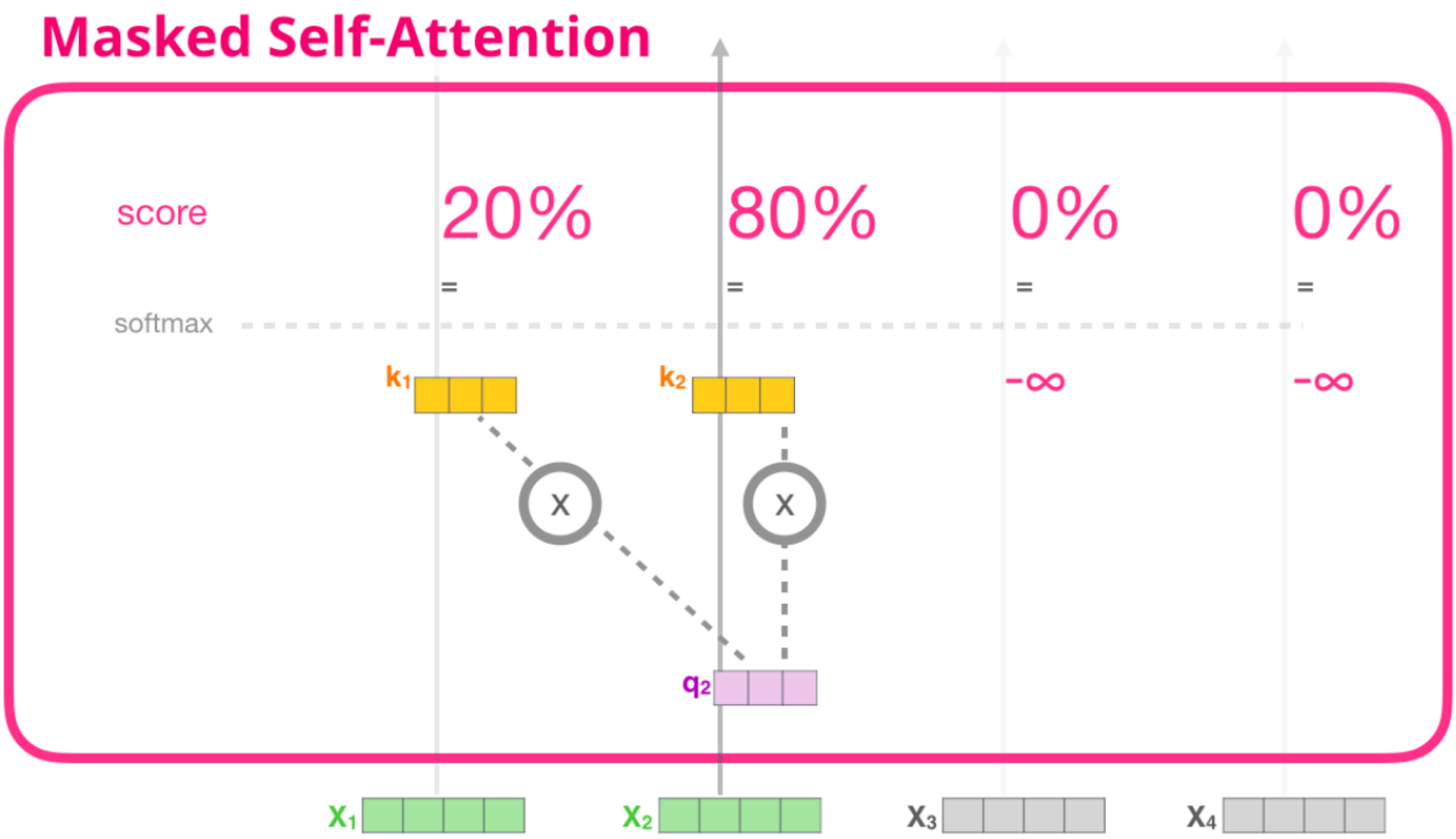
Masked Multi-head Attention

- Self attention layers in the decoder is only allowed to attend to earlier positions in the output sequence, which is done by masking future positions (setting them to –inf) before the softmax step in the self attention calculation.





Masked Multi-head Attention



Masked Multi-head Attention

- Do not need to be done sequentially, but can be done at one batch

	Features				Labels
Example: / position:	1	2	3	4	
1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

Masked Multi-head Attention

Masked Multi-head Attention

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax (along rows) →

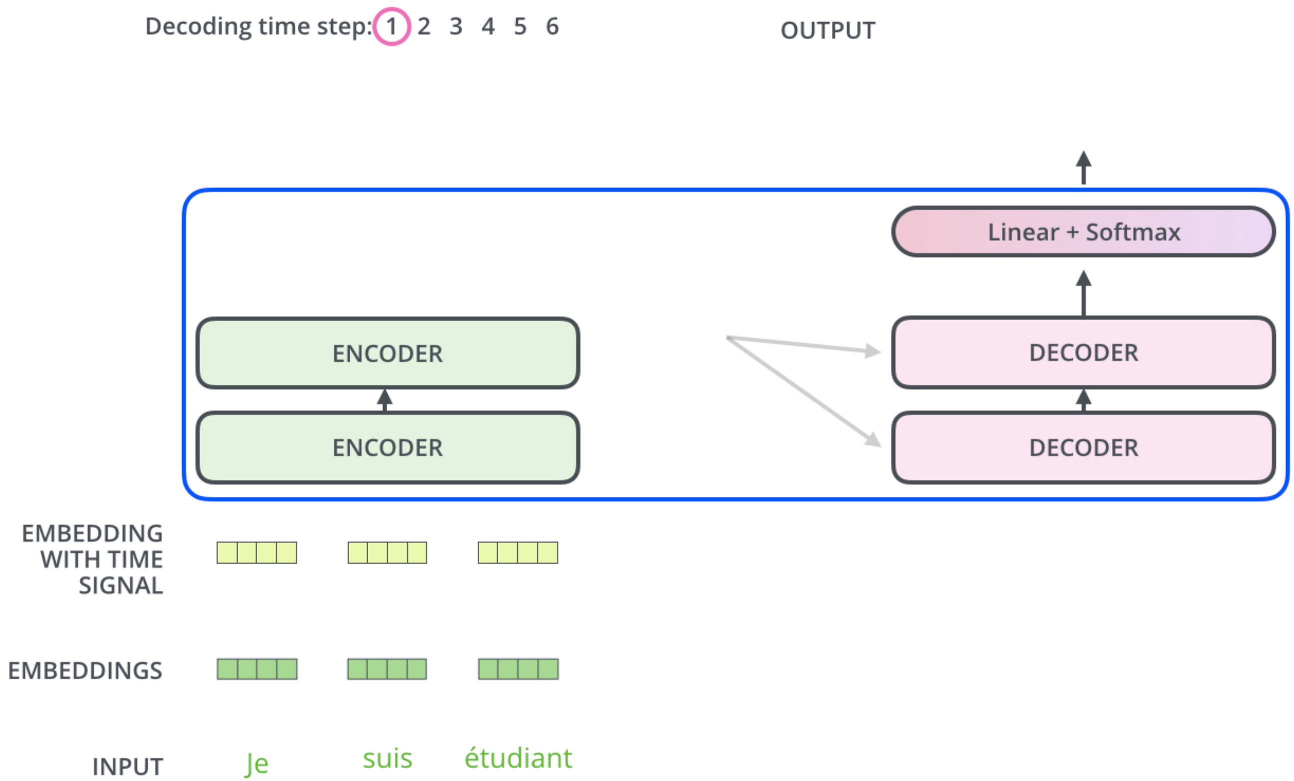
Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

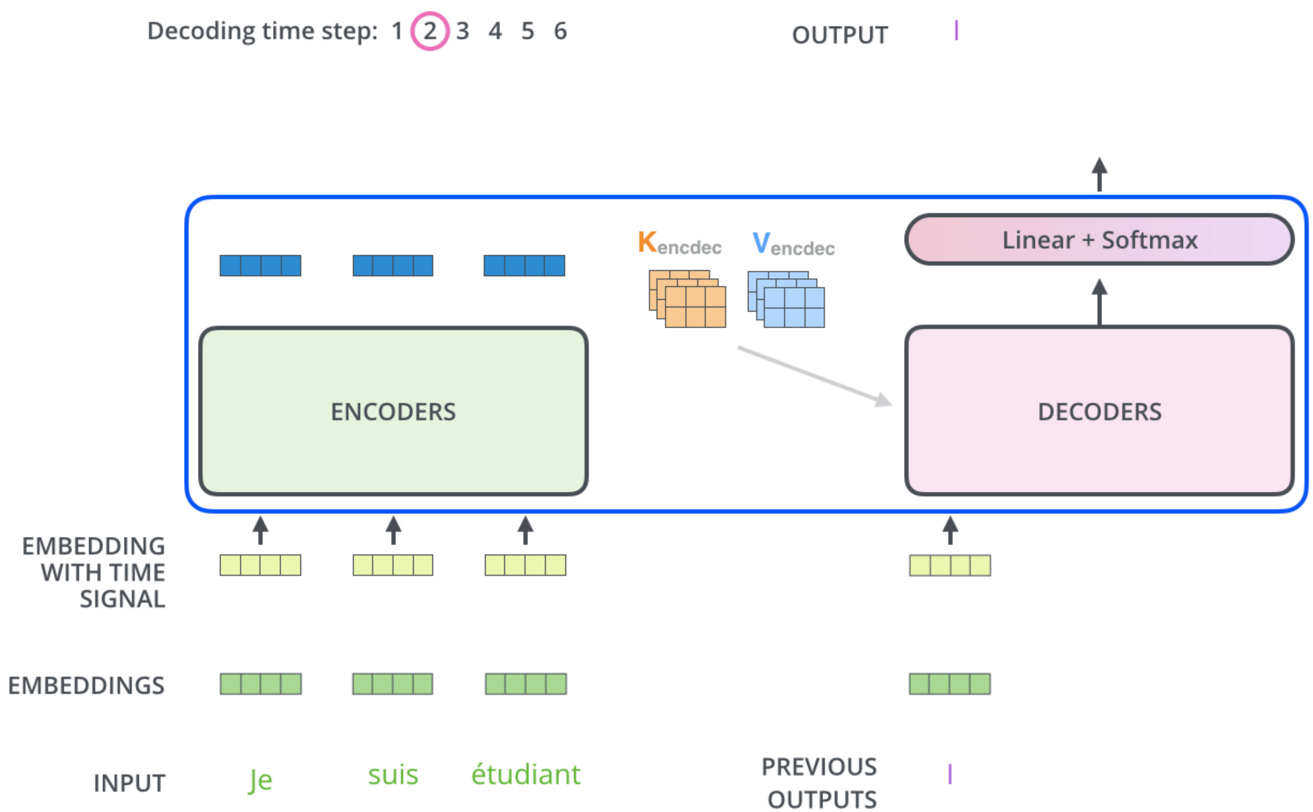
Combining Encoder & Decoder

Combining Encoder & Decoder

- The encoder start by processing the input sequence.
- The output of the top encoder is then transformed into a set of attention vectors K and V.
- These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence:

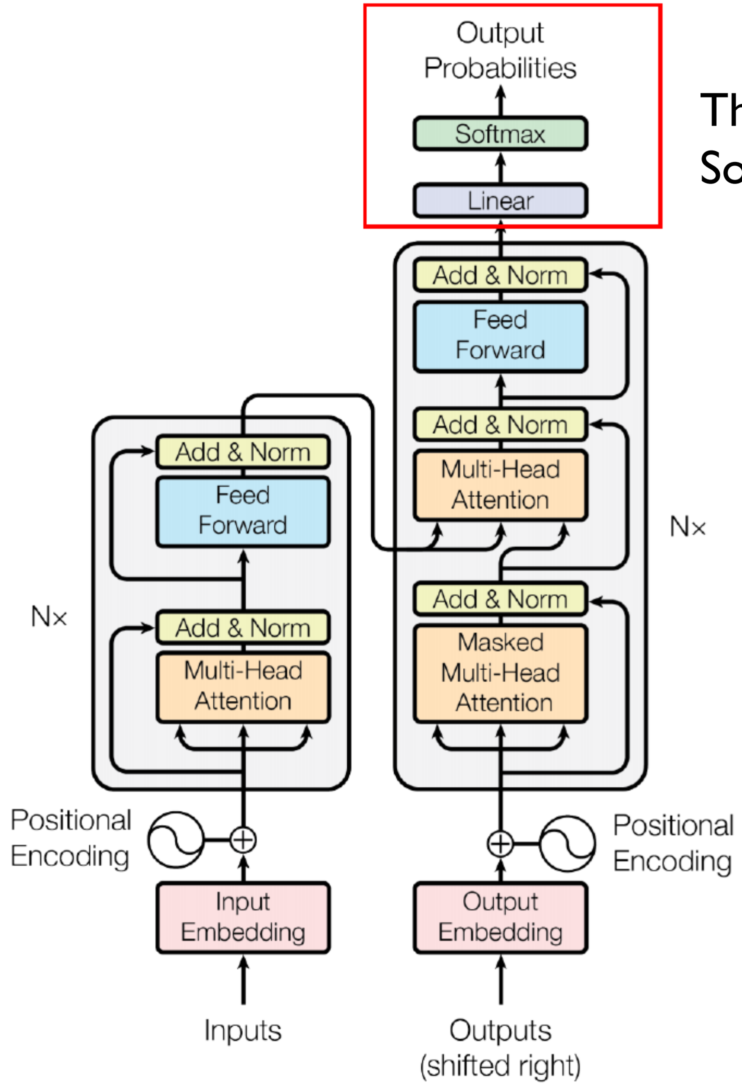


- Repeat the process until a special symbol is reached indicating decoder has completed its output.
- The output of each step is fed to the bottom decoder in the next time step



Final Linear & Softmax Layer

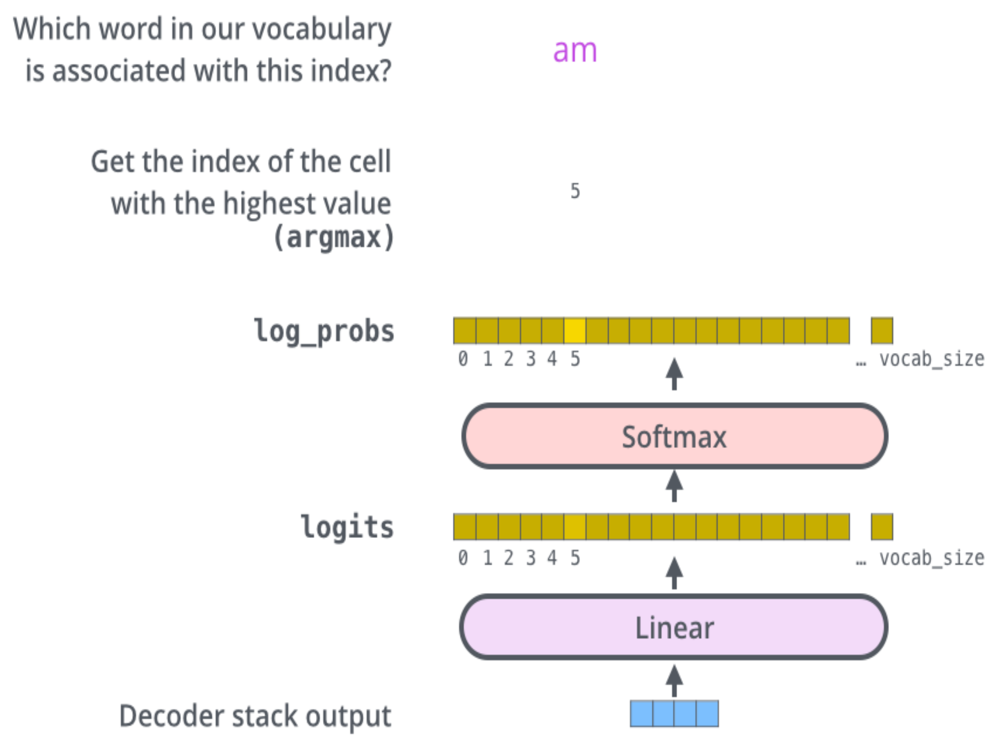
Final Linear & Softmax Layer



The Final Linear and Softmax Layer

Final Linear & Softmax Layer

- Linear layer
  - a simple fully connected neural network that projects the vector produced by the stack of decoders into a much larger vector called a logits vector
- Softmax layer
  - turns those scores into probability
  - The cell with the highest probability is chosen, the word associated with it is produced as the output of this time step



수고하셨습니다 ..^ ^..