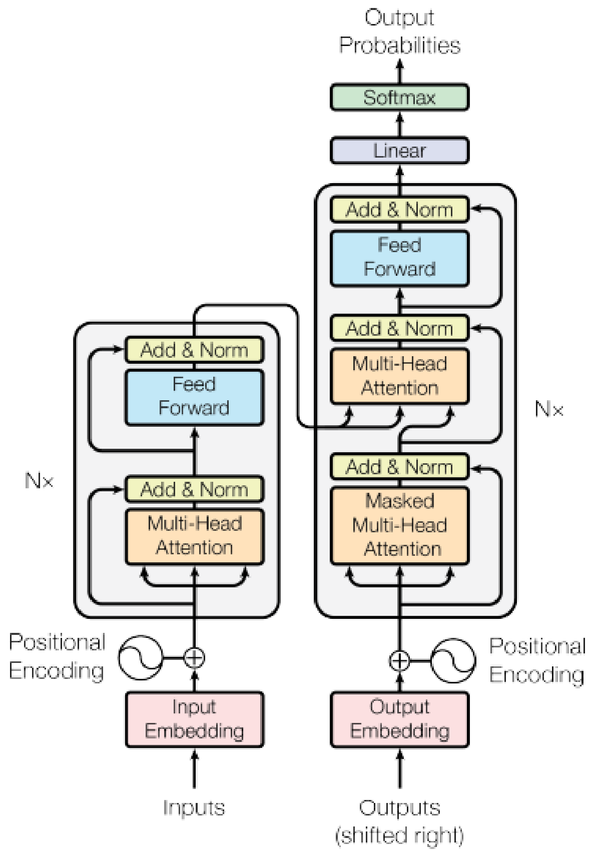
CSE Workshop
NLP - Transformers

Satwik Ram Kodandaram

Ph.D. Student - Accessible Computing Lab / WS-DL Research Group

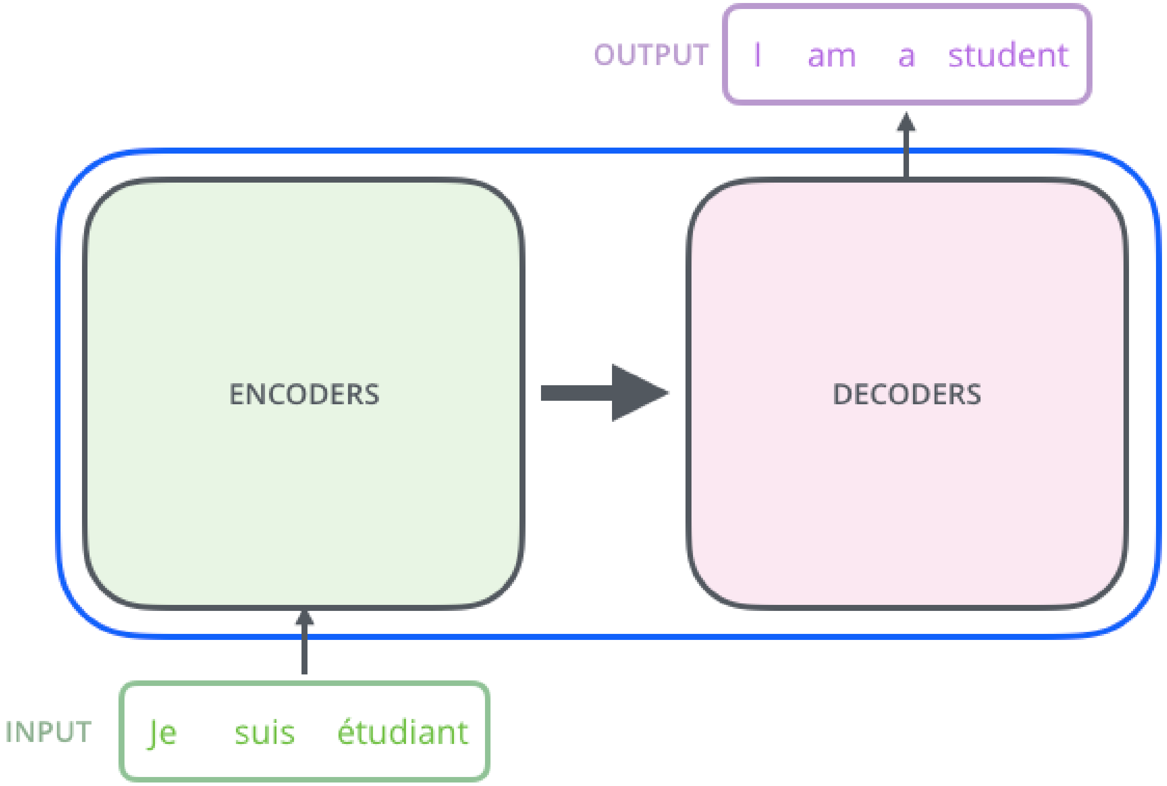
Transformers Architecture



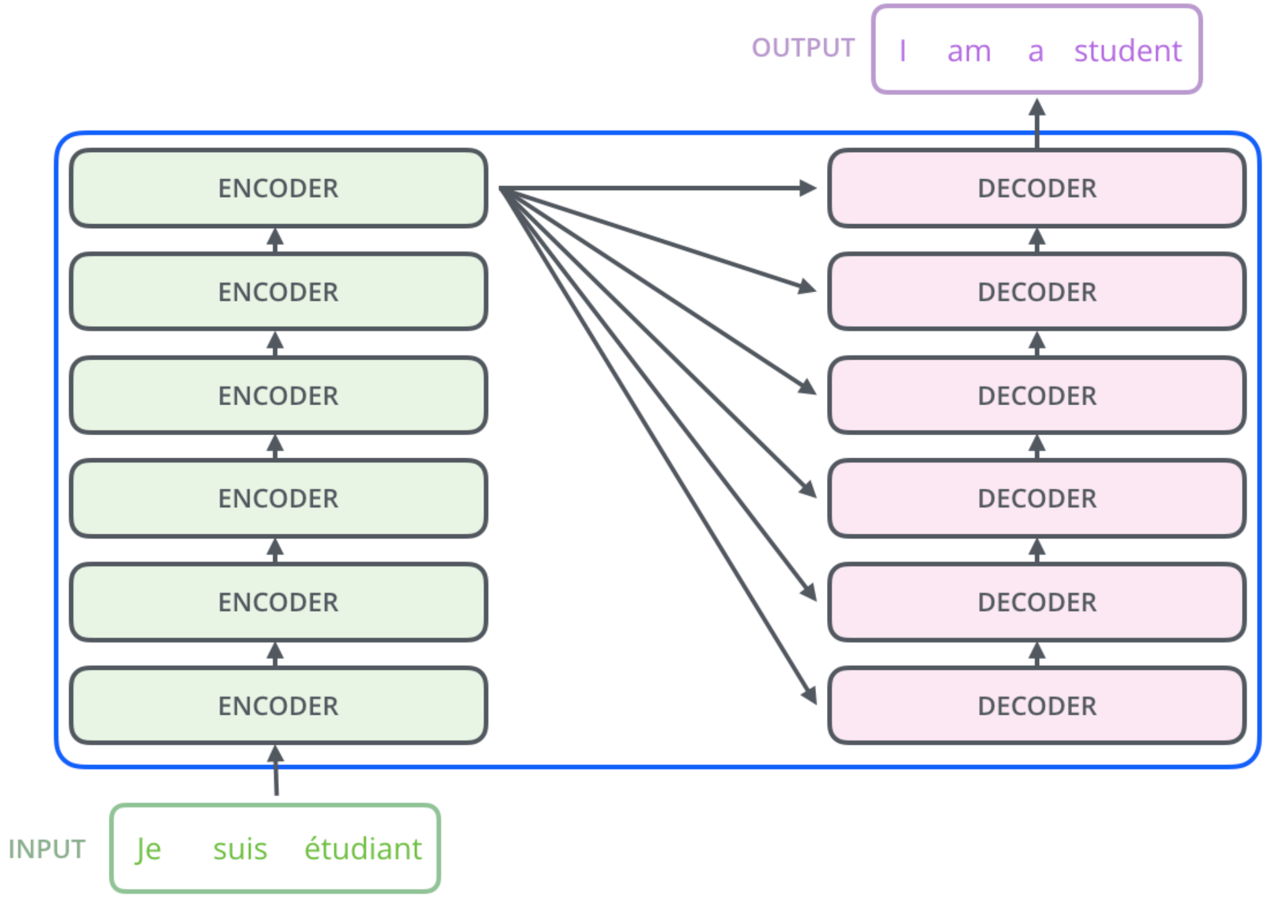
A High-Level Look

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

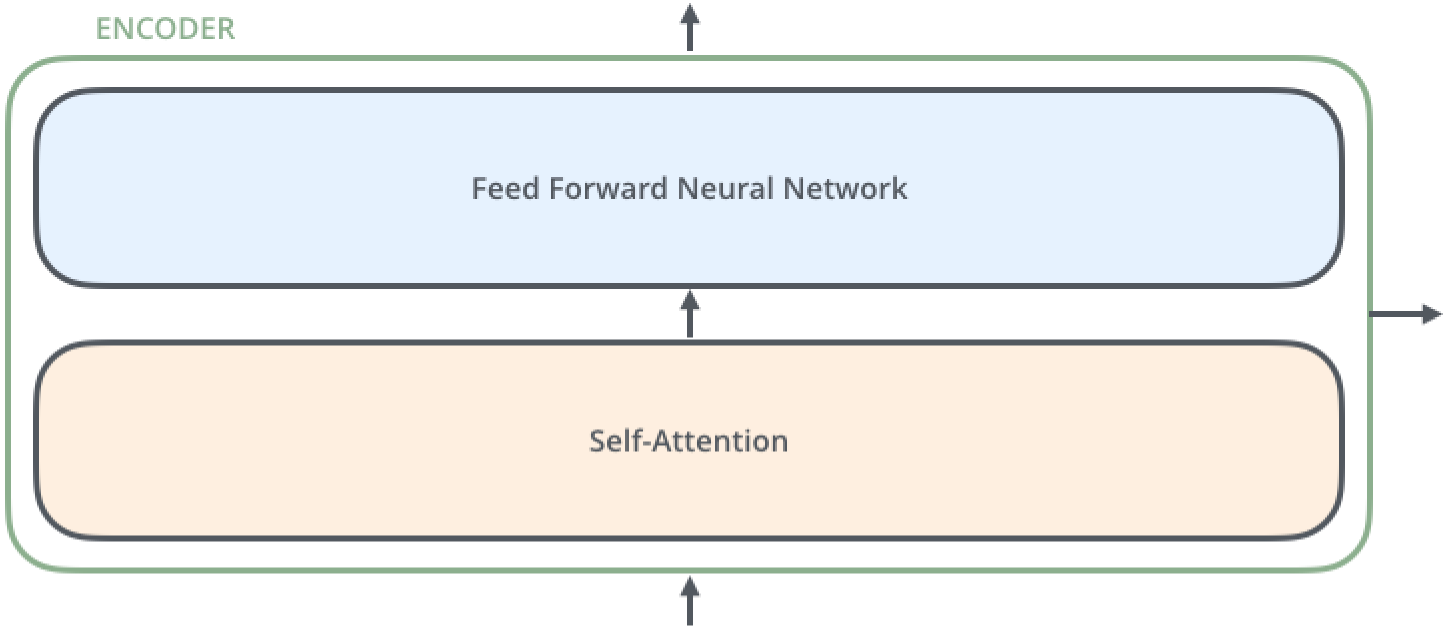
A High-Level Look



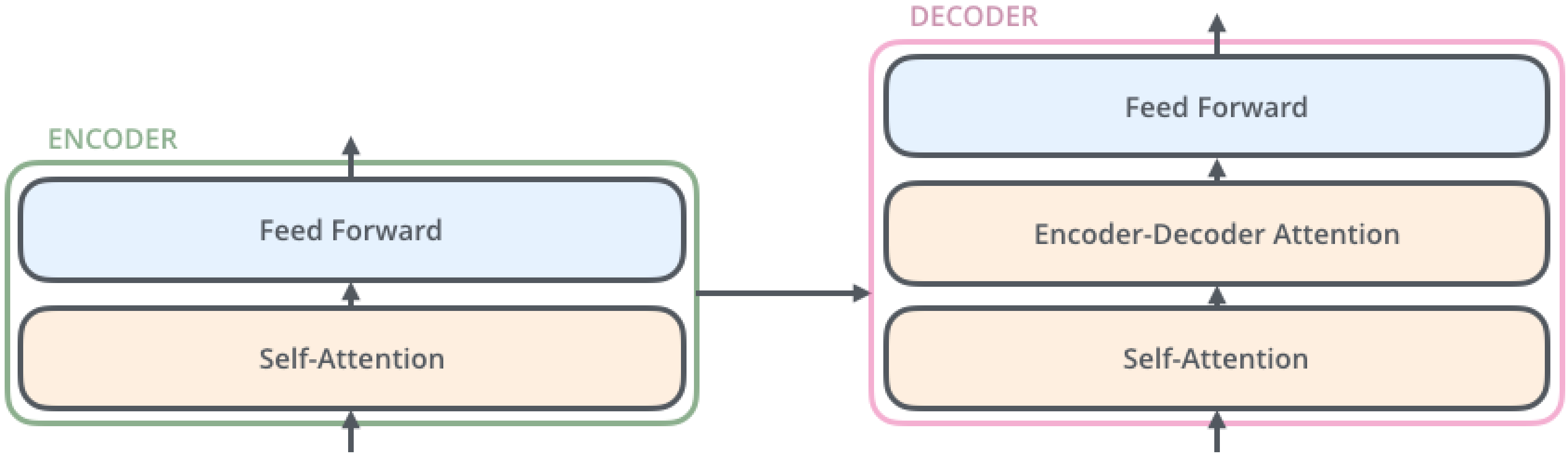
Encoders Decoders



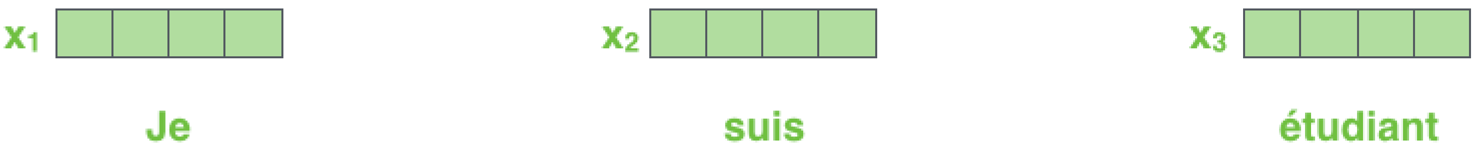
Encoders Decoders



Encoders Decoders

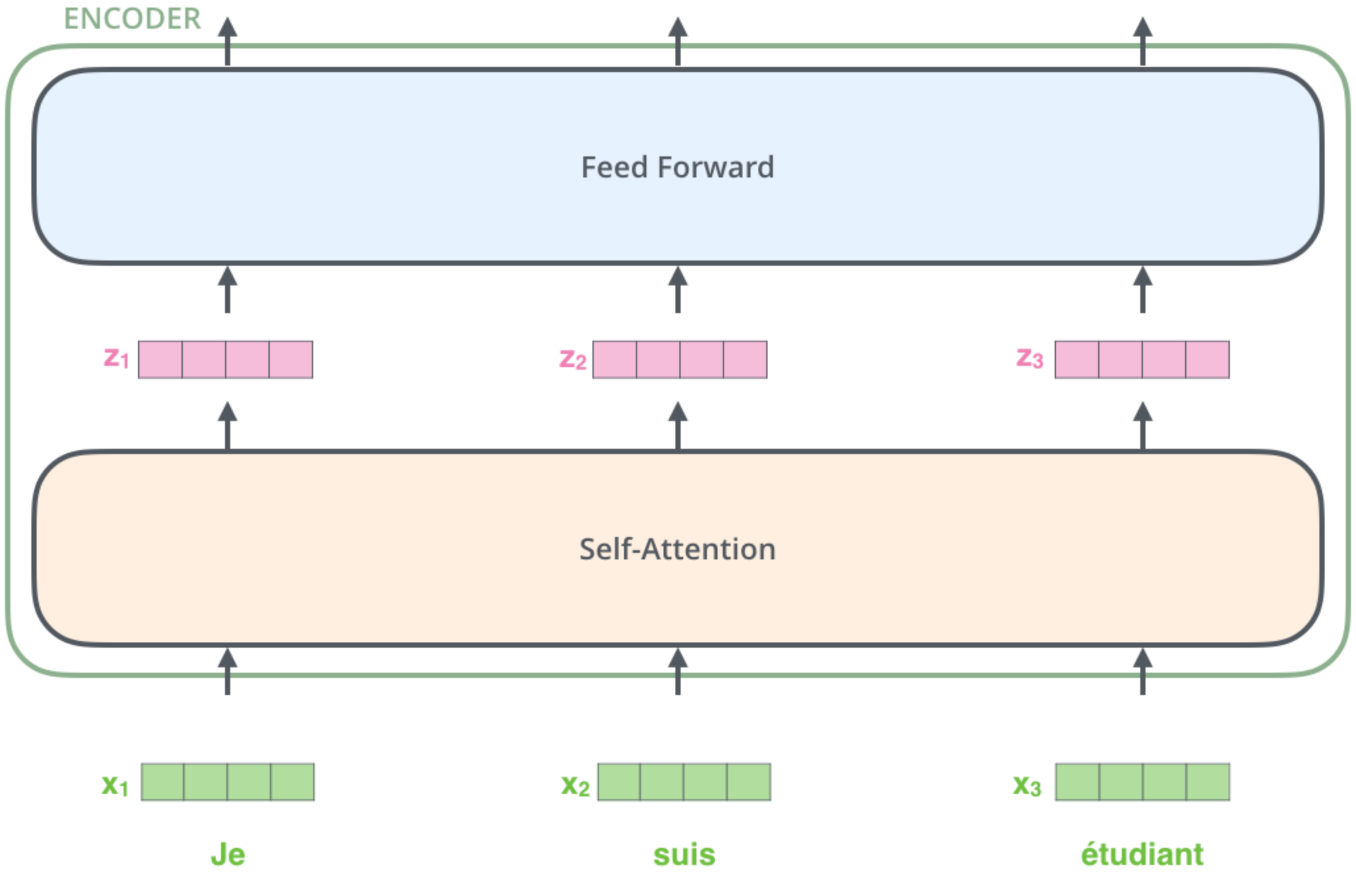


Bringing The Tensors Into The Picture

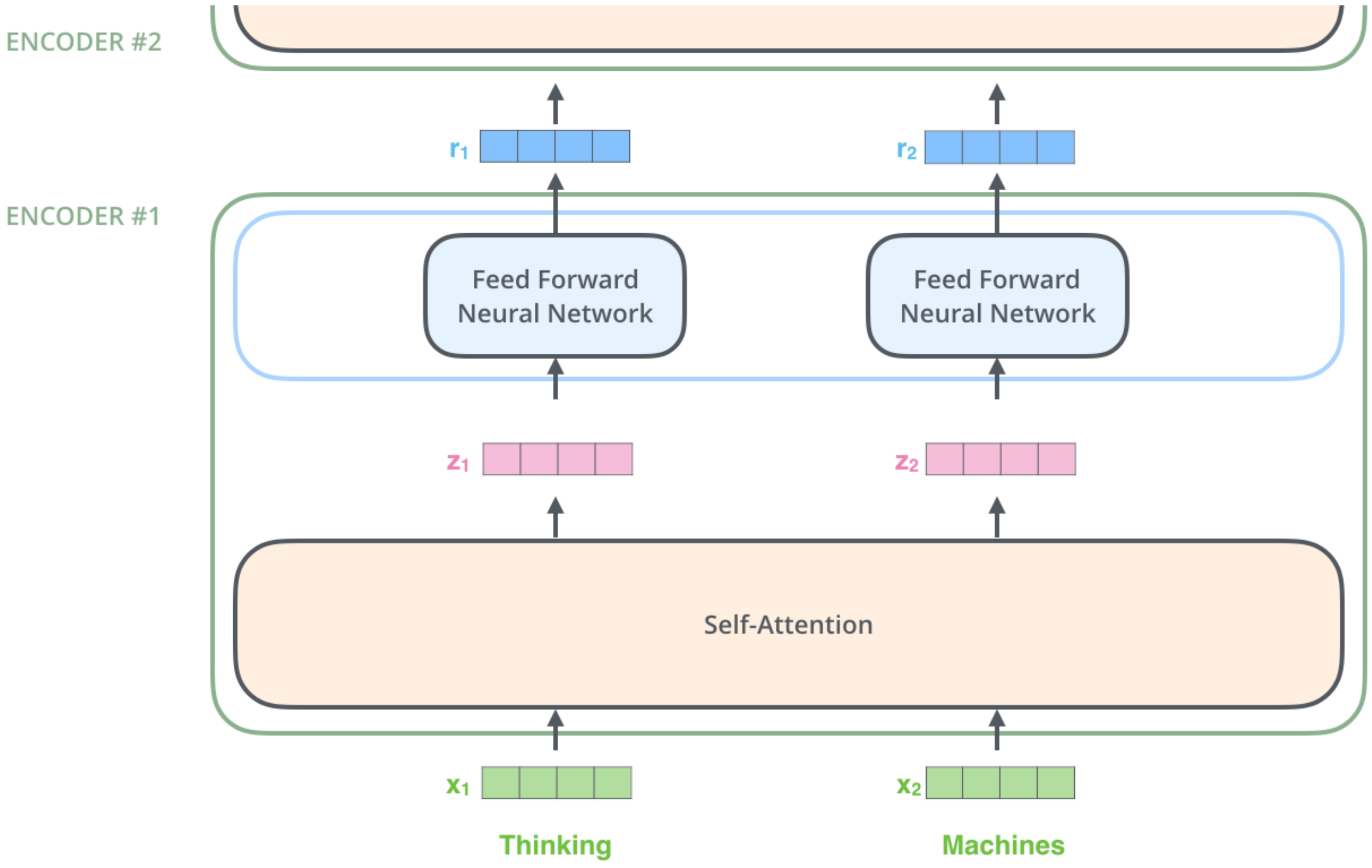


Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

Bringing The Tensors Into The Picture



Now We're Encoding!

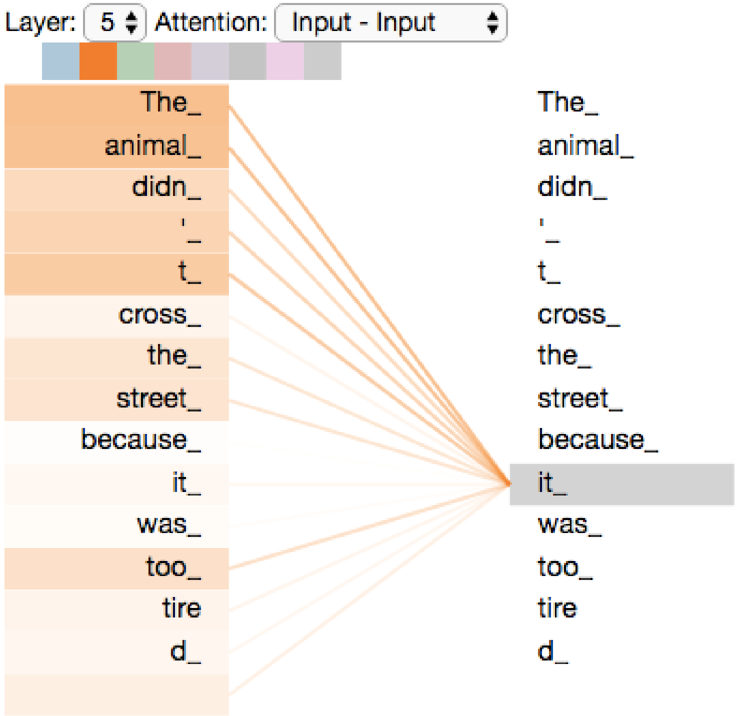


Self-Attention at a High Level

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

- What does "it" in this sentence refer to? Is it referring to the street or to the animal? It's a simple question to a human, but not as simple to an algorithm.
- When the model is processing the word "it", self-attention allows it to associate "it" with "animal".

Self-Attention at a High Level

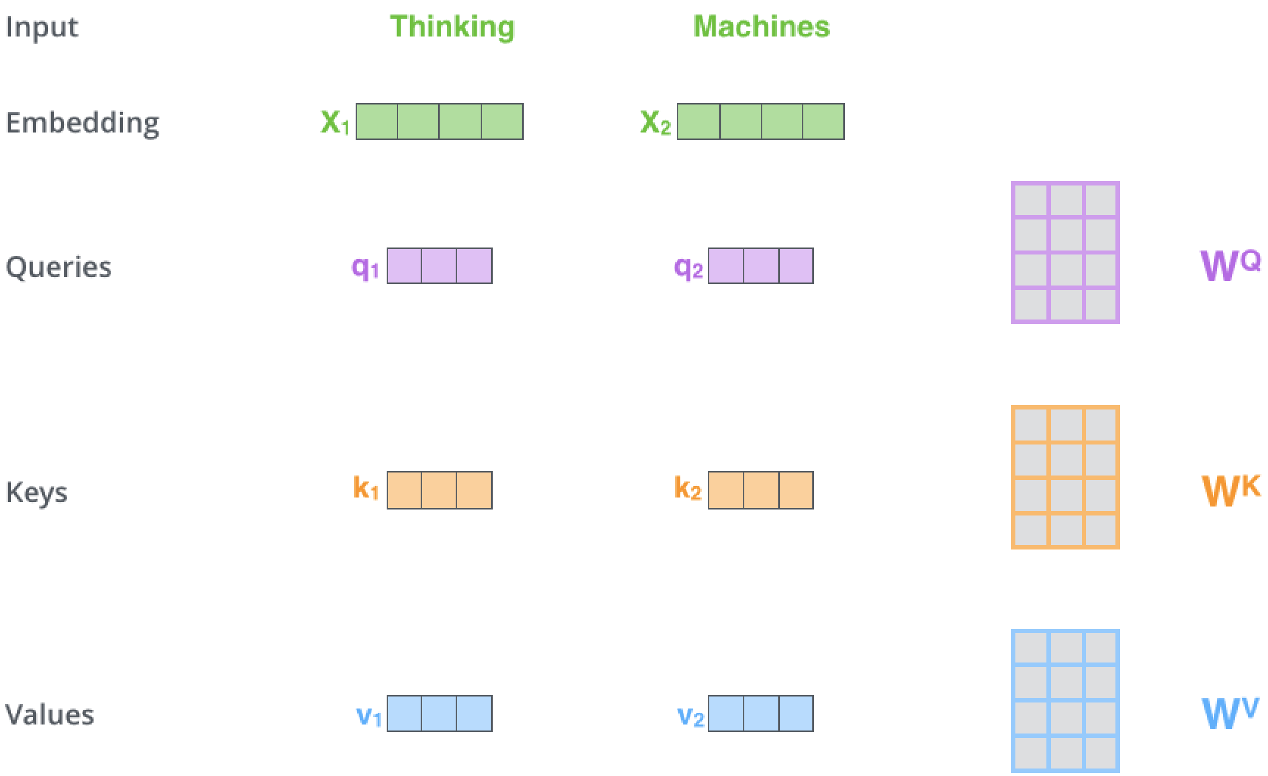


As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Self-Attention in Detail

- The **first step** in calculating self-attention is to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word)
- So, for each word, we create a Query vector, a Key vector, and a Value vector
- Their dimensionality is 64
- the embedding and encoder input/output vectors have dimensionality of 512

Self-Attention in Detail

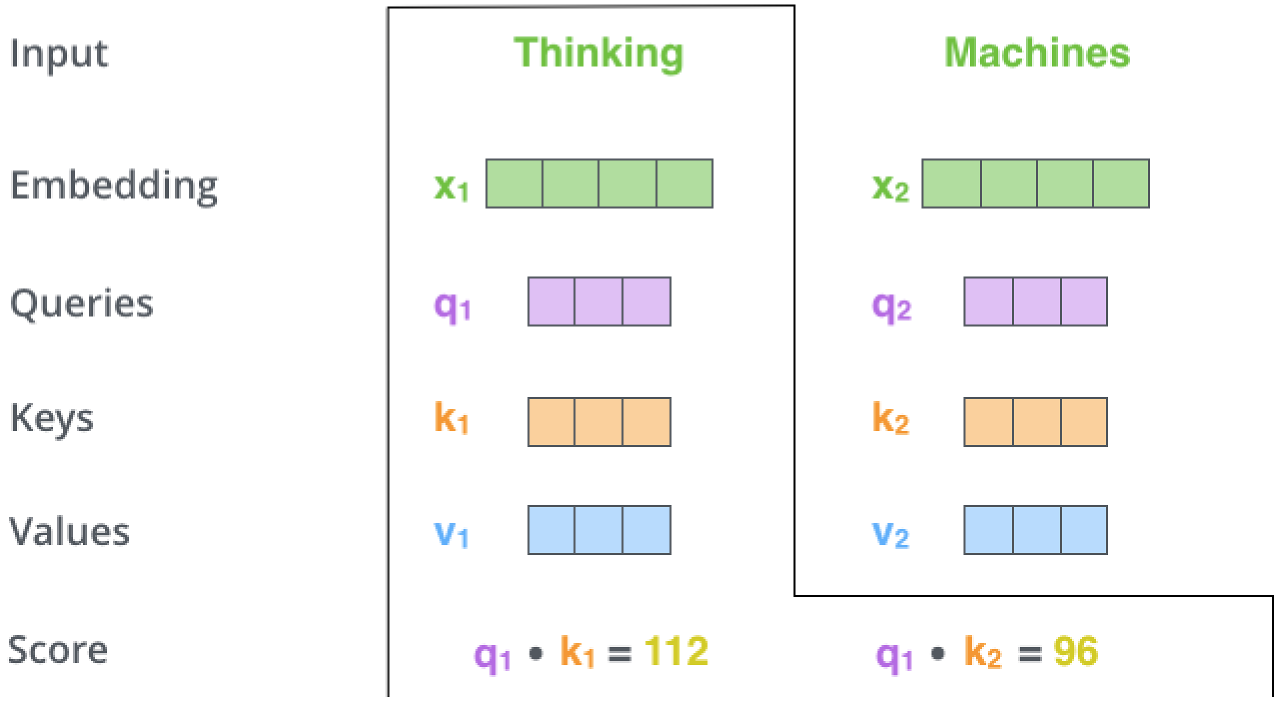


Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

What are the “query”, “key”, and “value” vectors?

- The **second step** in calculating self-attention is to calculate a score.
- Say we’re calculating the self-attention for the first word in this example, “Thinking”
- The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position
- Score = dot product of the query vector with the key vector
- Self-attention for the word in position #1, the first score would be the dot product of q1 and k1.

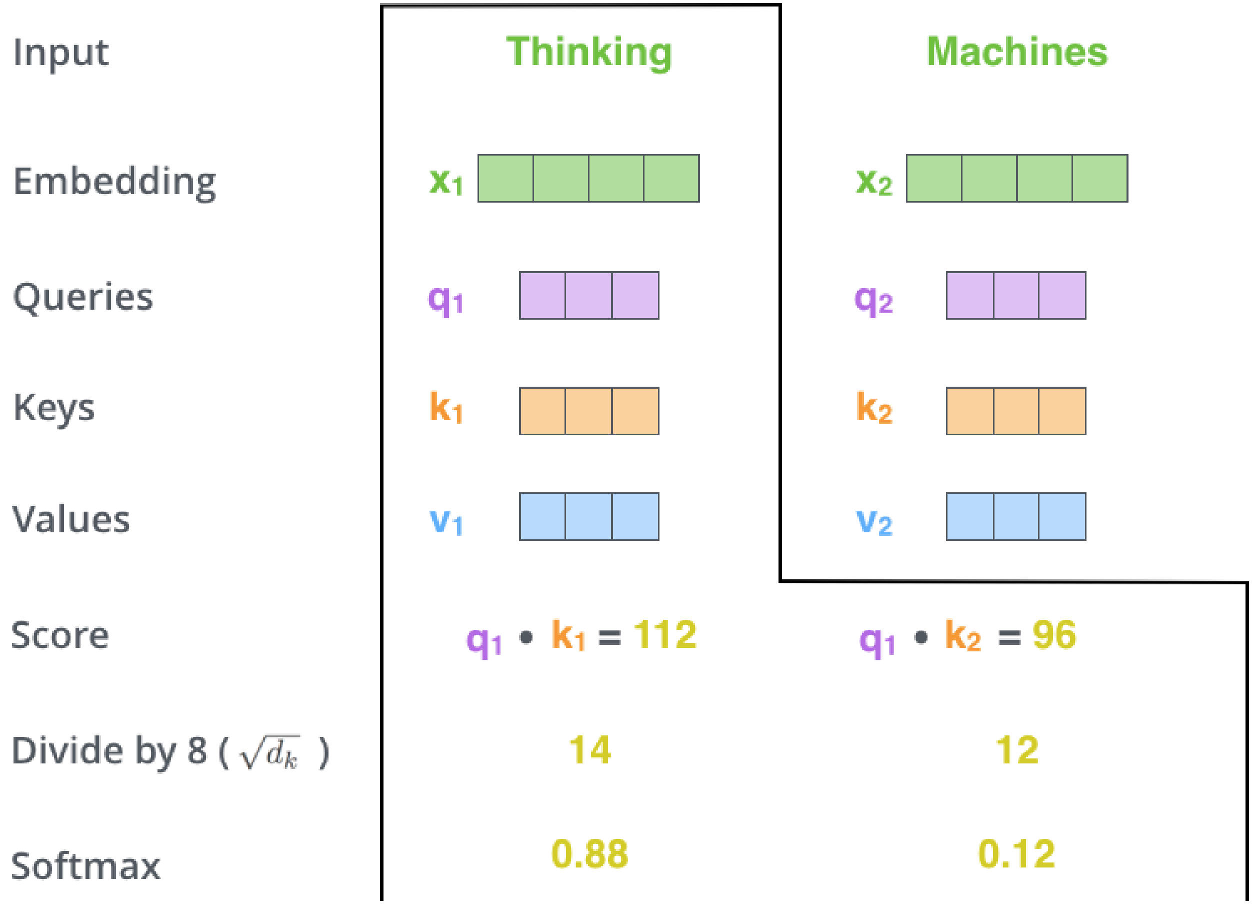
Self-Attention Score



Self-Attention Score

- The **third and fourth steps** are to divide the scores by 8
- Square root of the dimension of the key vectors used in the paper – 64
- There could be other possible values here, but this is the default), then pass the result through a softmax operation
- Softmax normalizes the scores so they're all positive and add up to 1

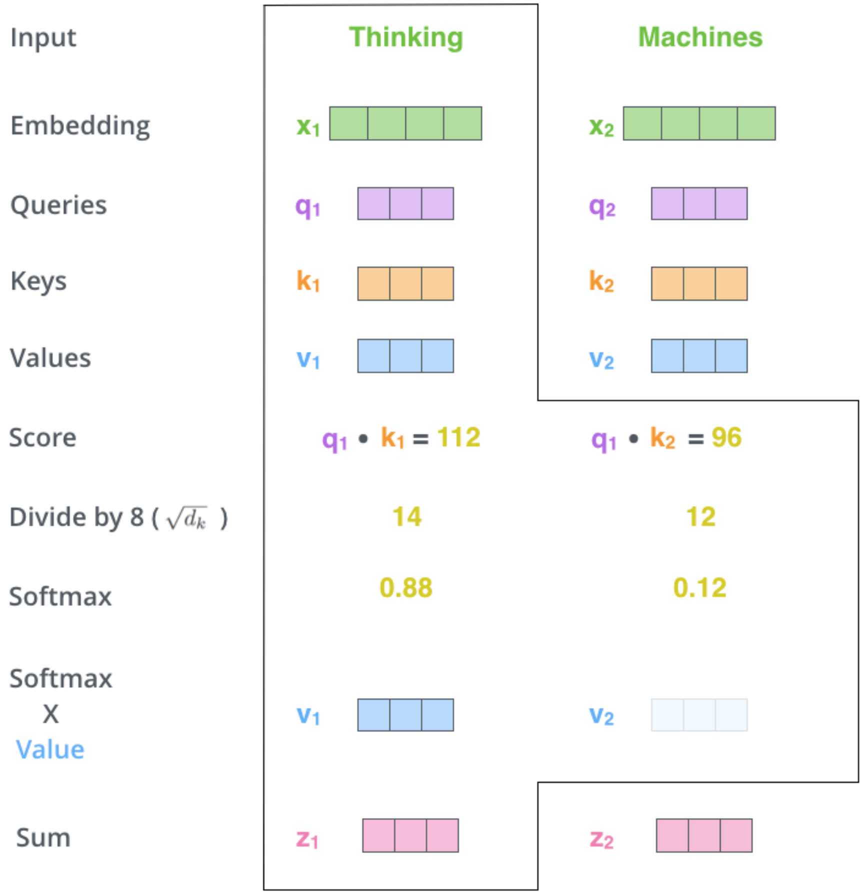
Self Attention Score



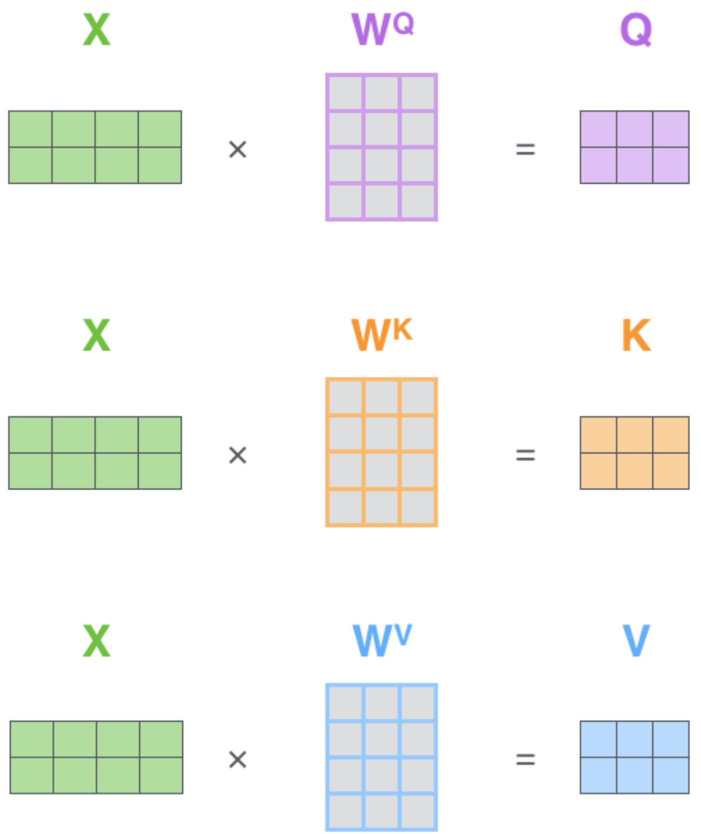
Self-Attention Score

- This softmax score determines how much each word will be expressed at this position
- The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up)
- The **sixth step** is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

Self-Attention Score

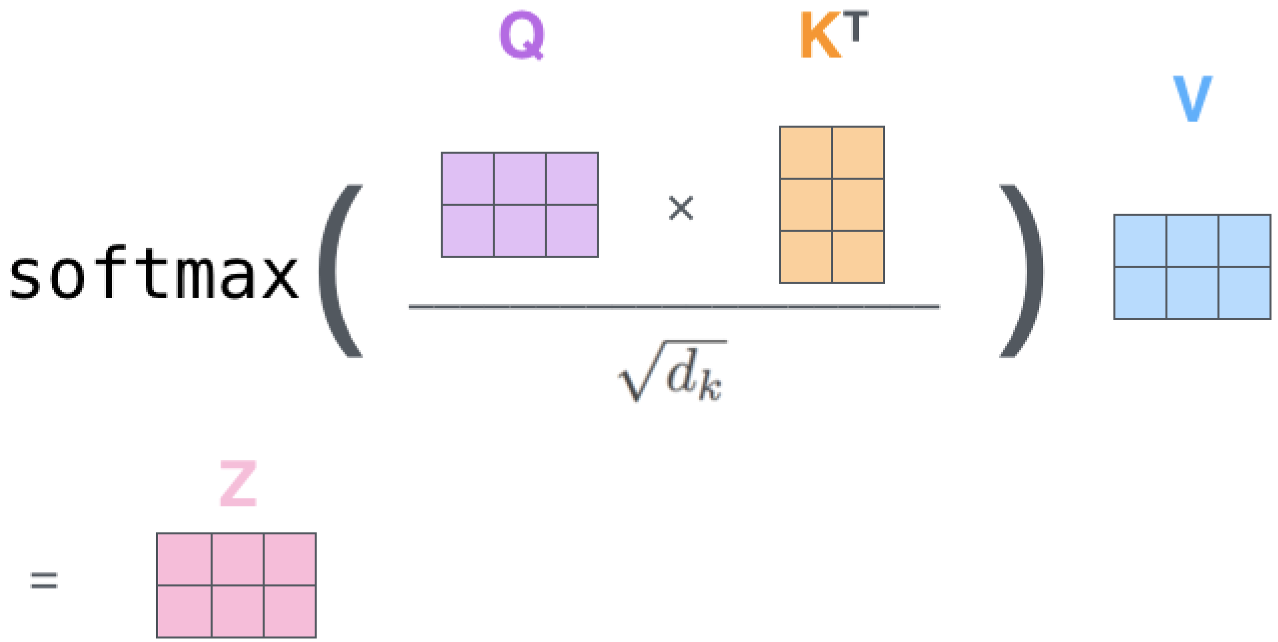


Matrix Calculation of Self-Attention



Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

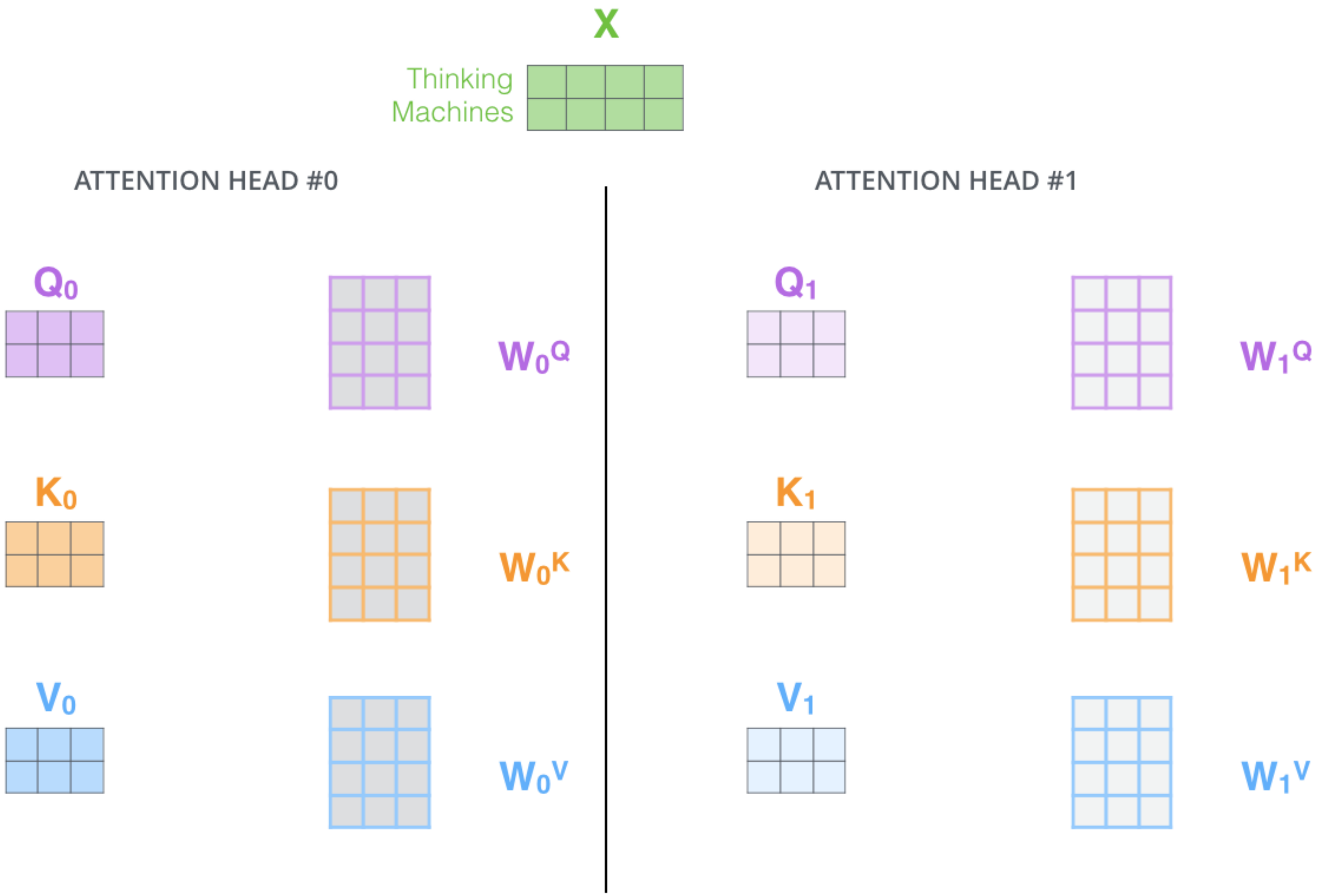
Matrix Calculation of Self-Attention



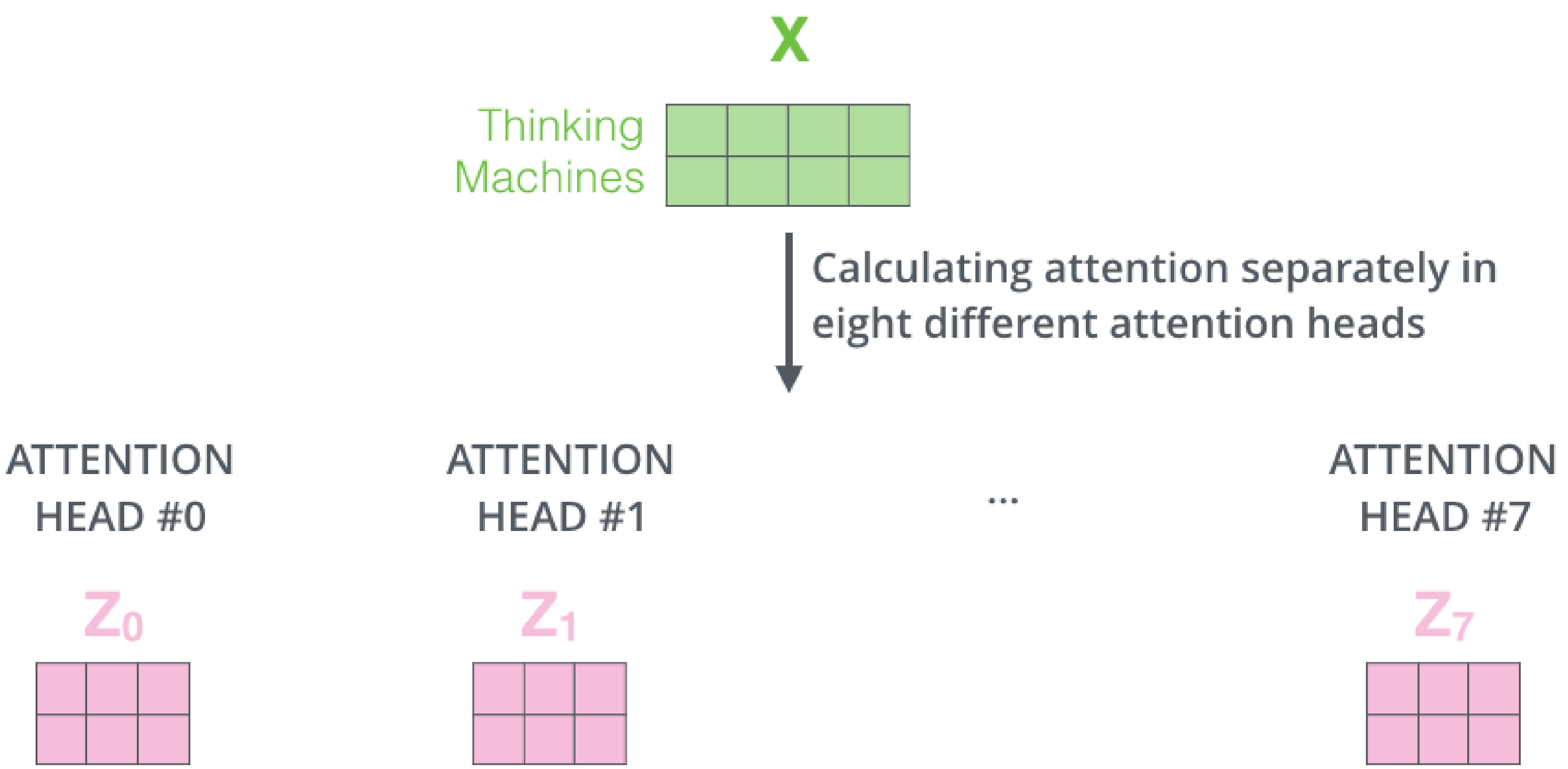
The Beast With Many Heads

- The paper further refined the self-attention layer by adding a mechanism called “multi-headed” attention
- It expands the model’s ability to focus on different positions
- It gives the attention layer multiple “representation subspaces”
- with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices
- Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder

Multi-headed Attention

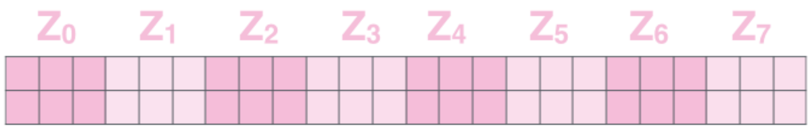


Multi-headed Attention



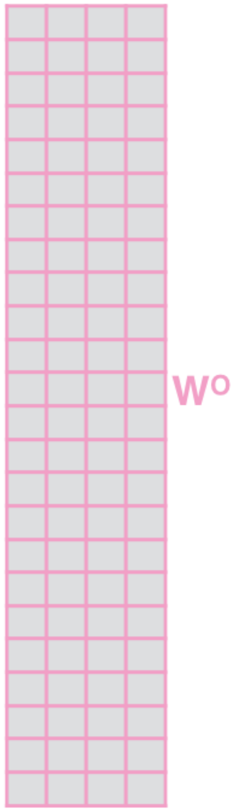
Multi-headed Attention

1) Concatenate all the attention heads



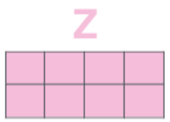
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

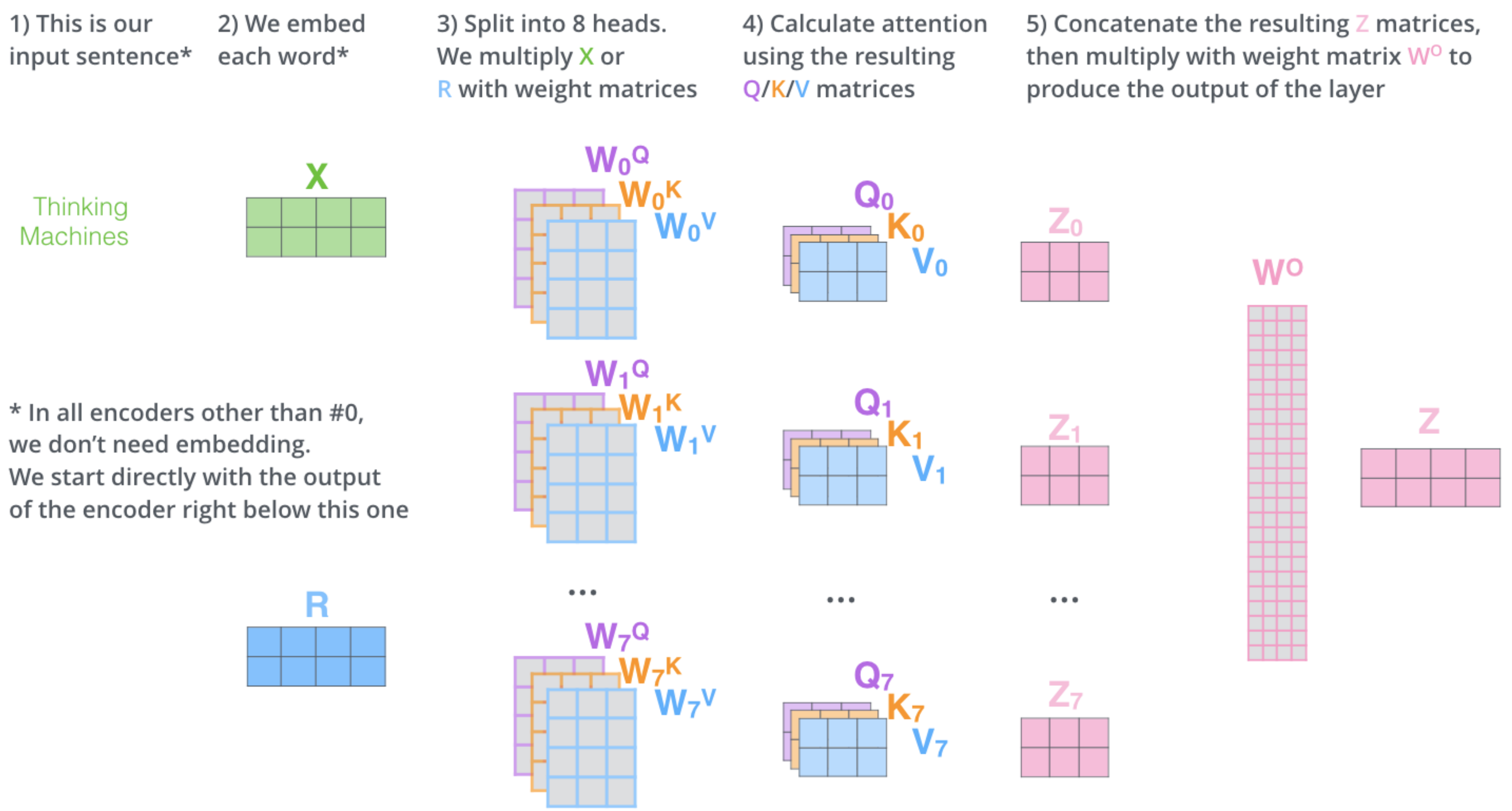


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

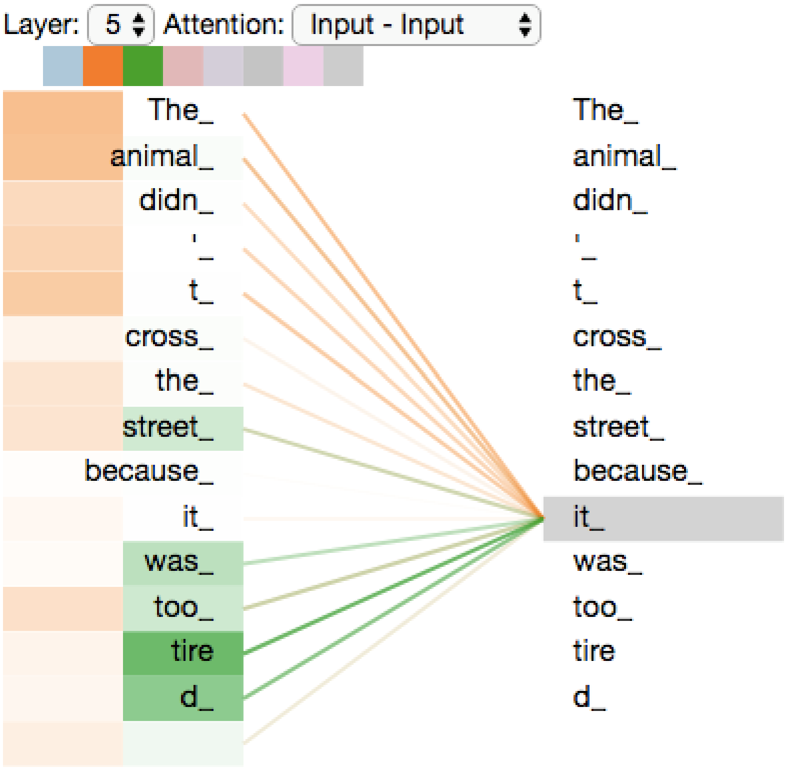
=



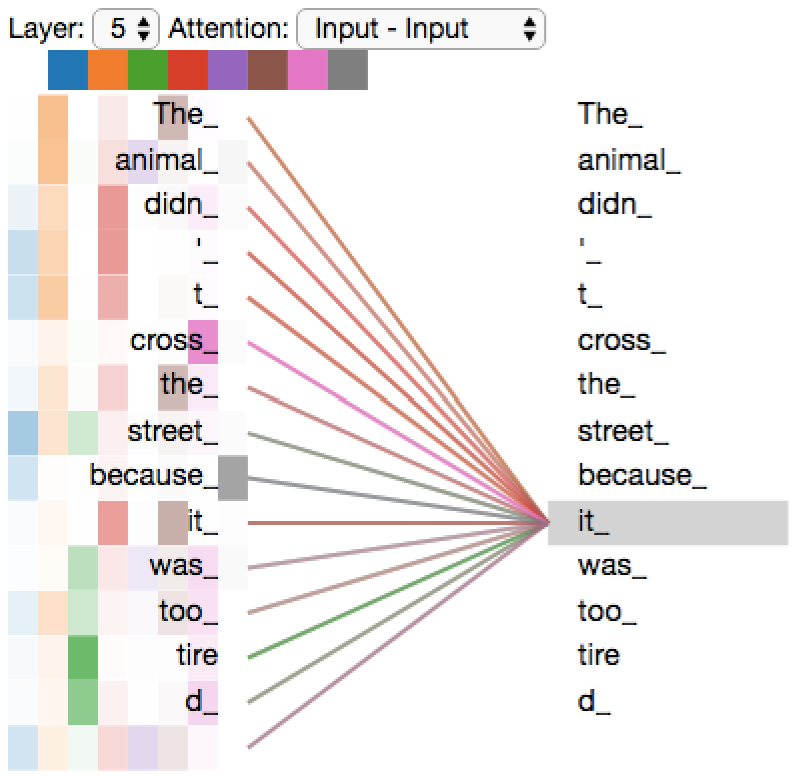
Multi-headed Attention



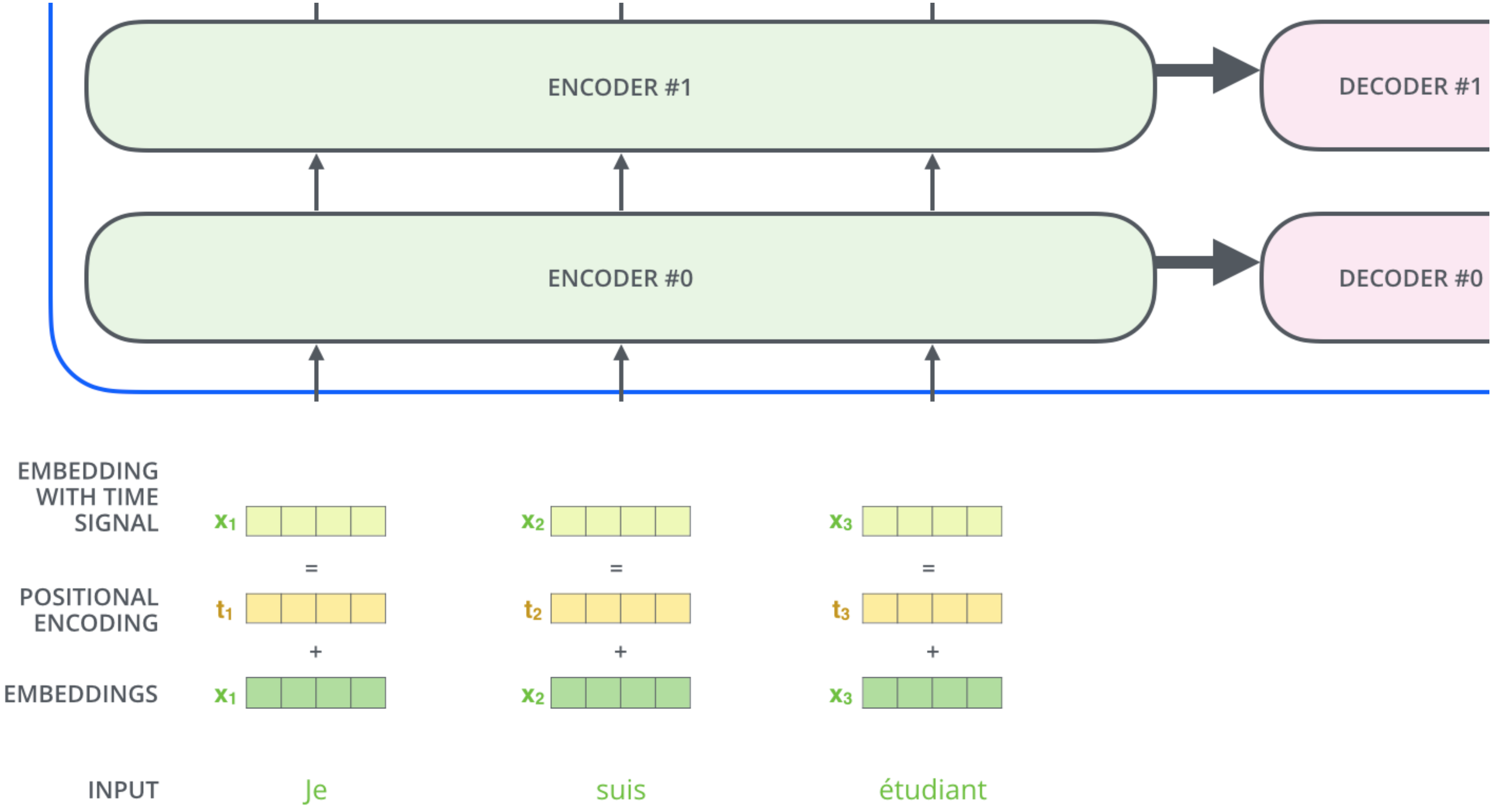
Multi-headed Attention



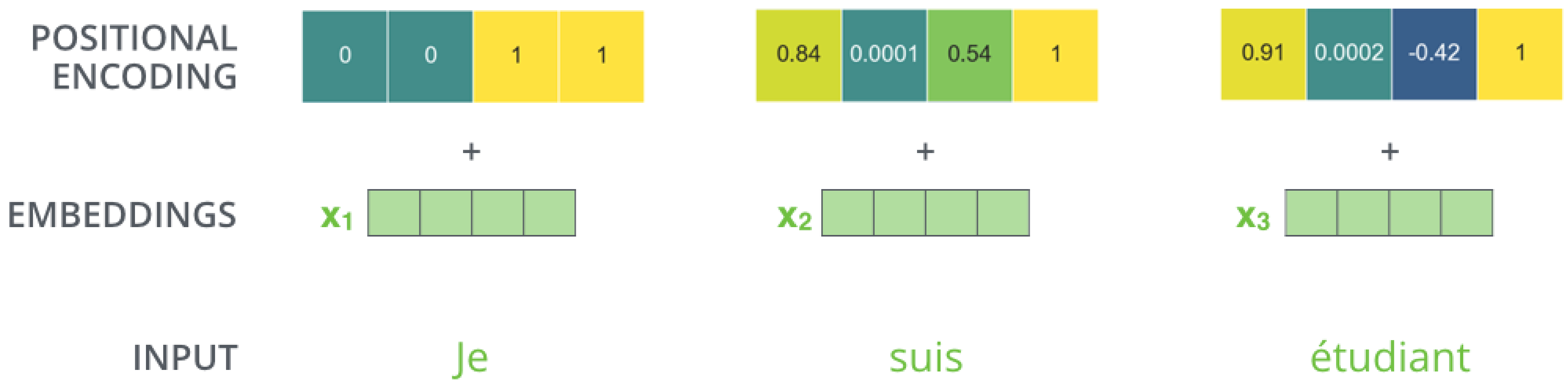
Multi-headed Attention



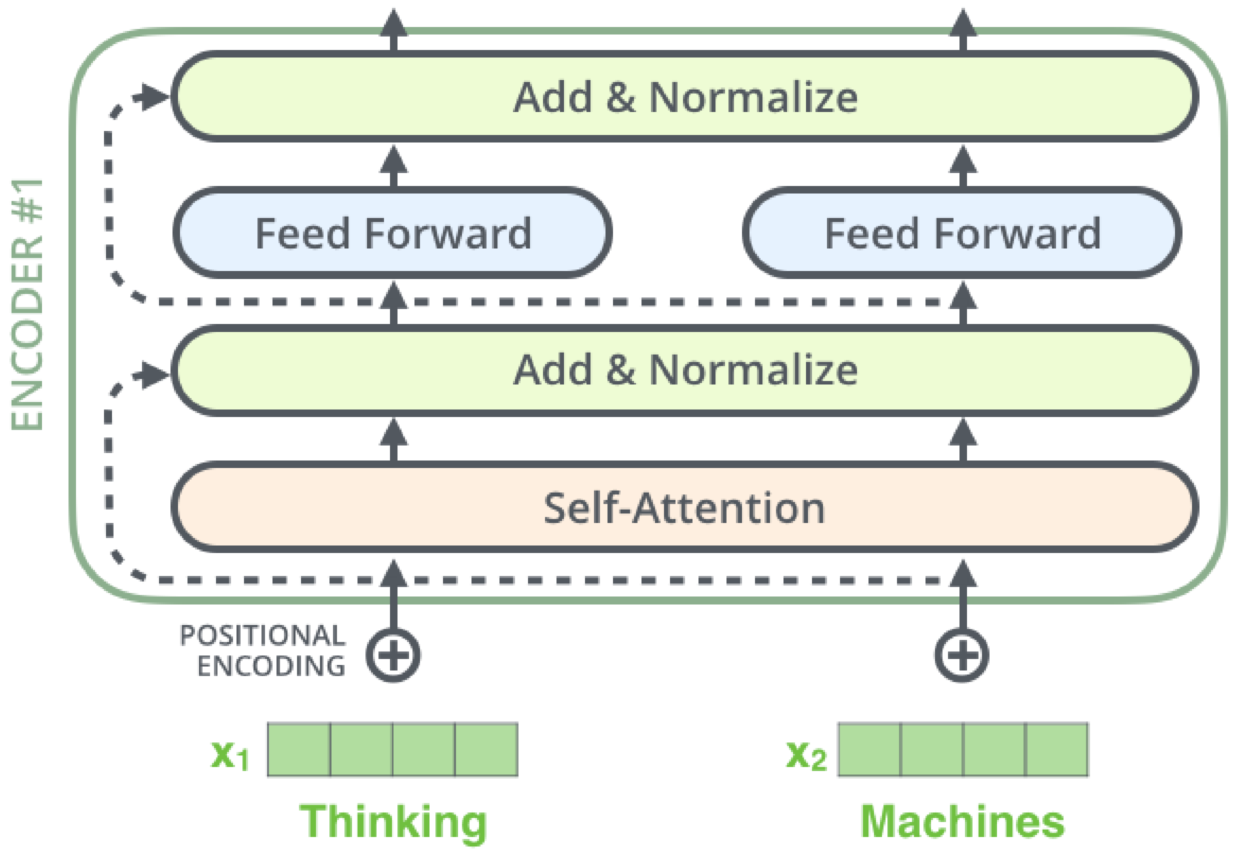
Representing The Order of The Sequence Using Positional Encoding



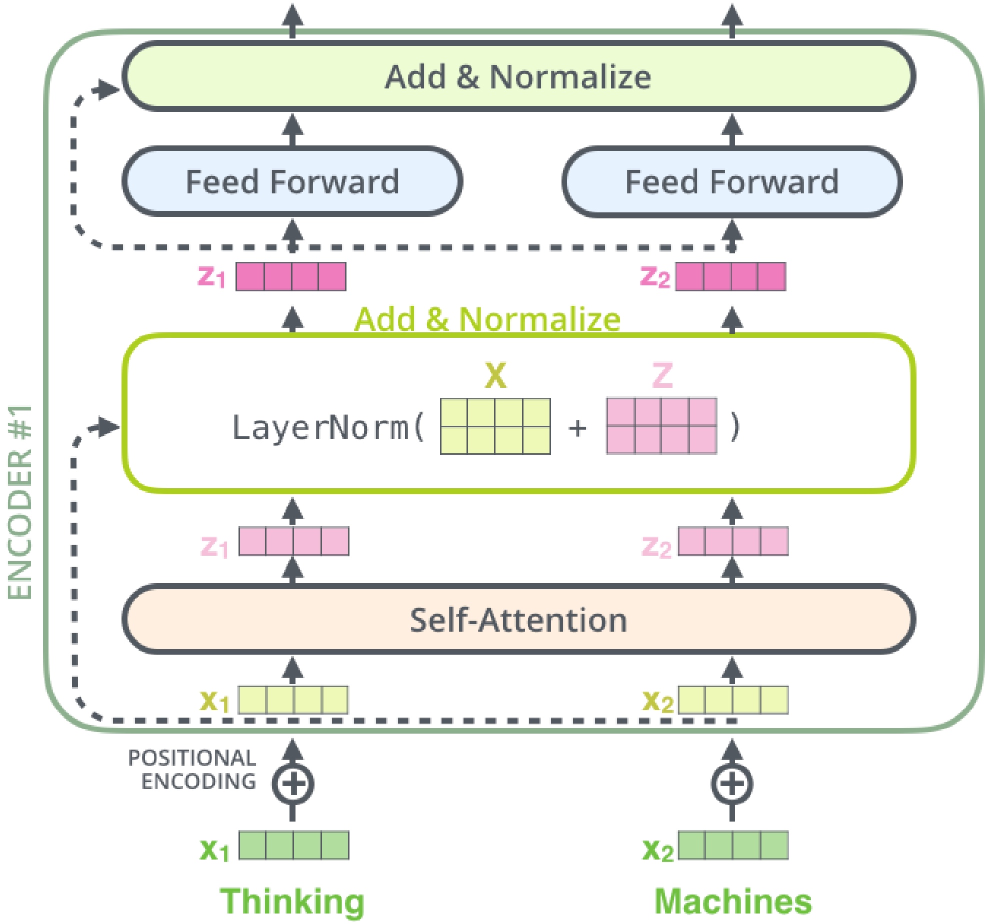
Representing The Order of The Sequence Using Positional Encoding



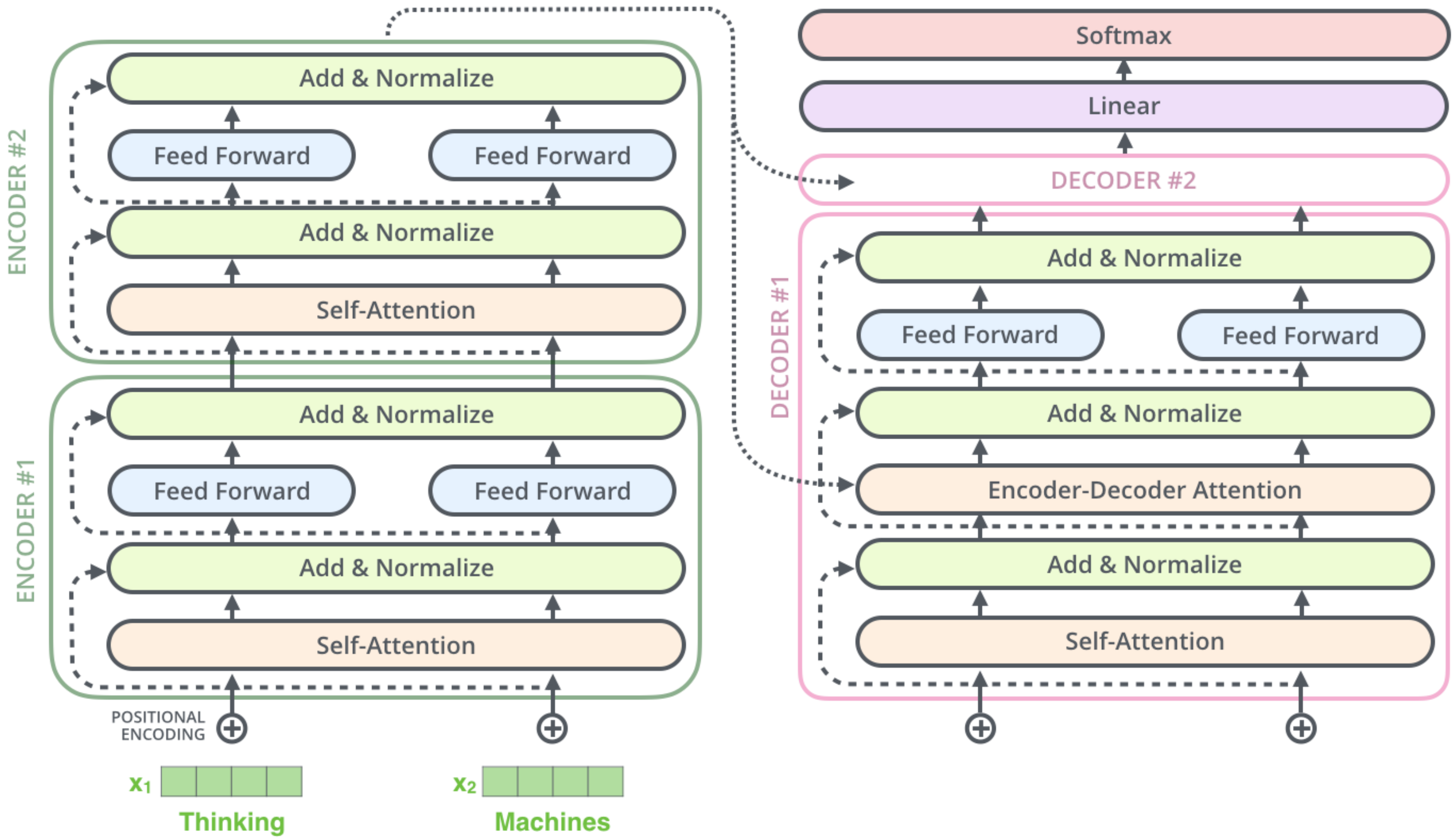
The Residuals



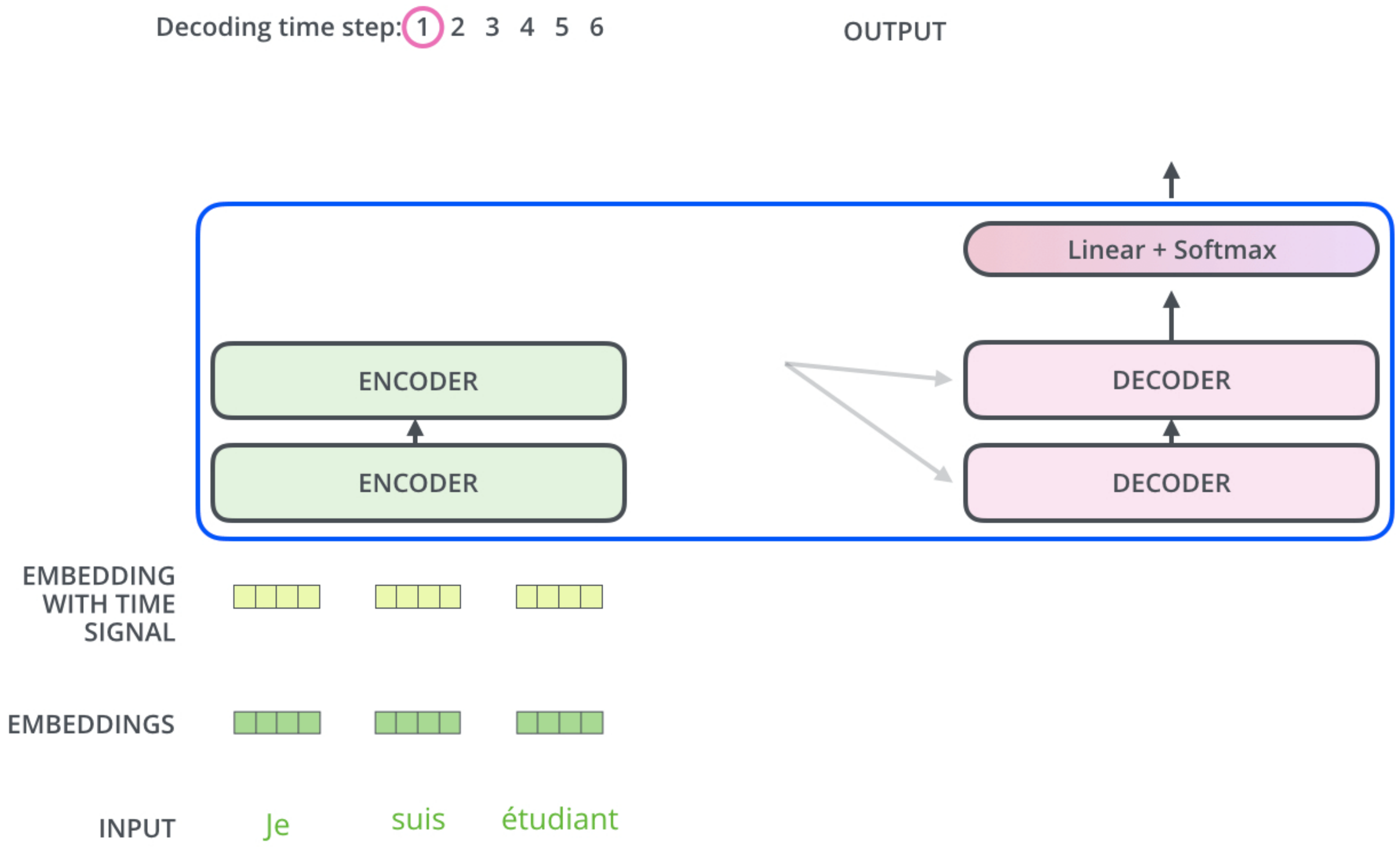
The Residuals



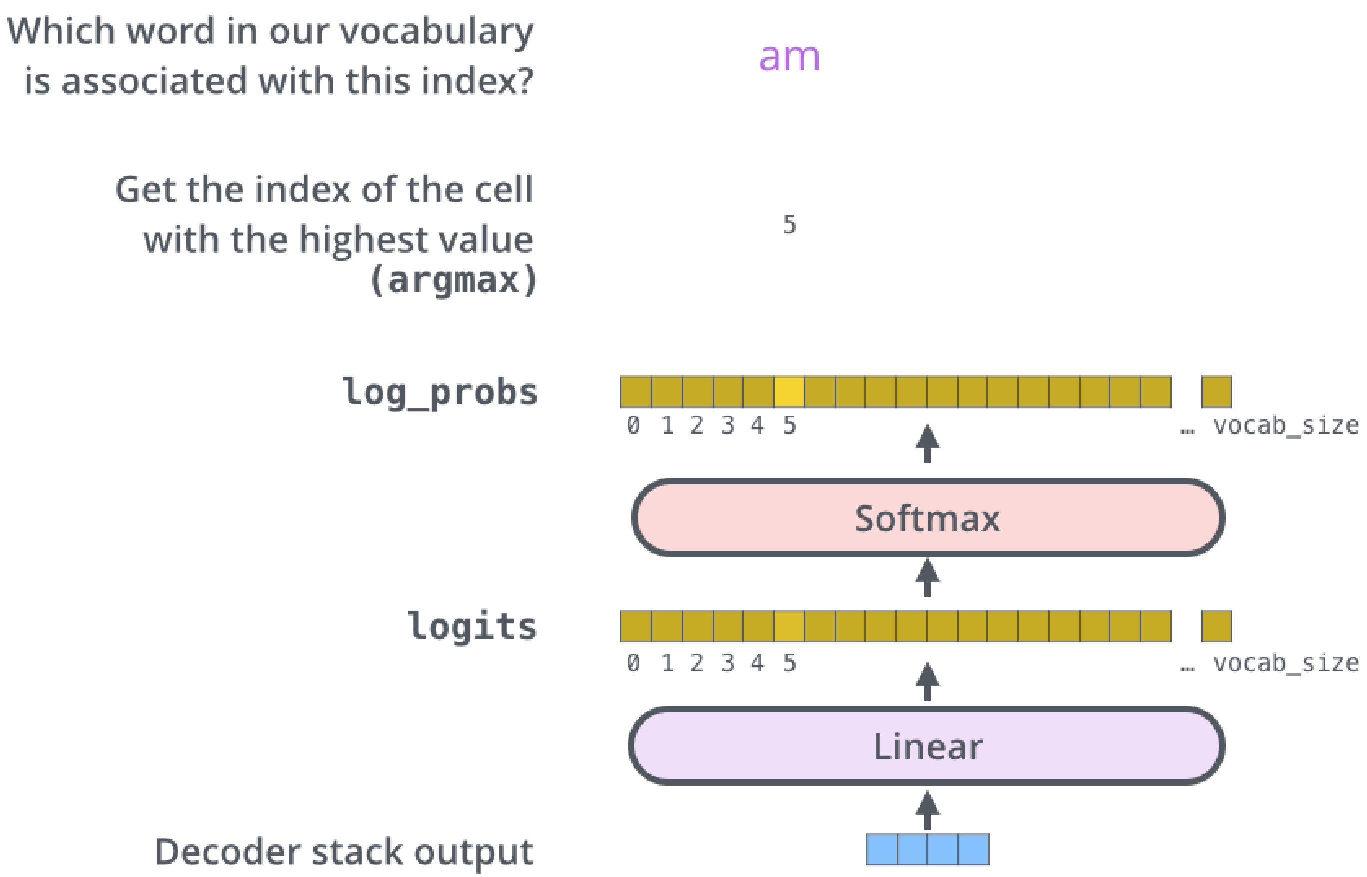
The Residuals



The Decoder Side



The Decoder Side



The Decoder Side

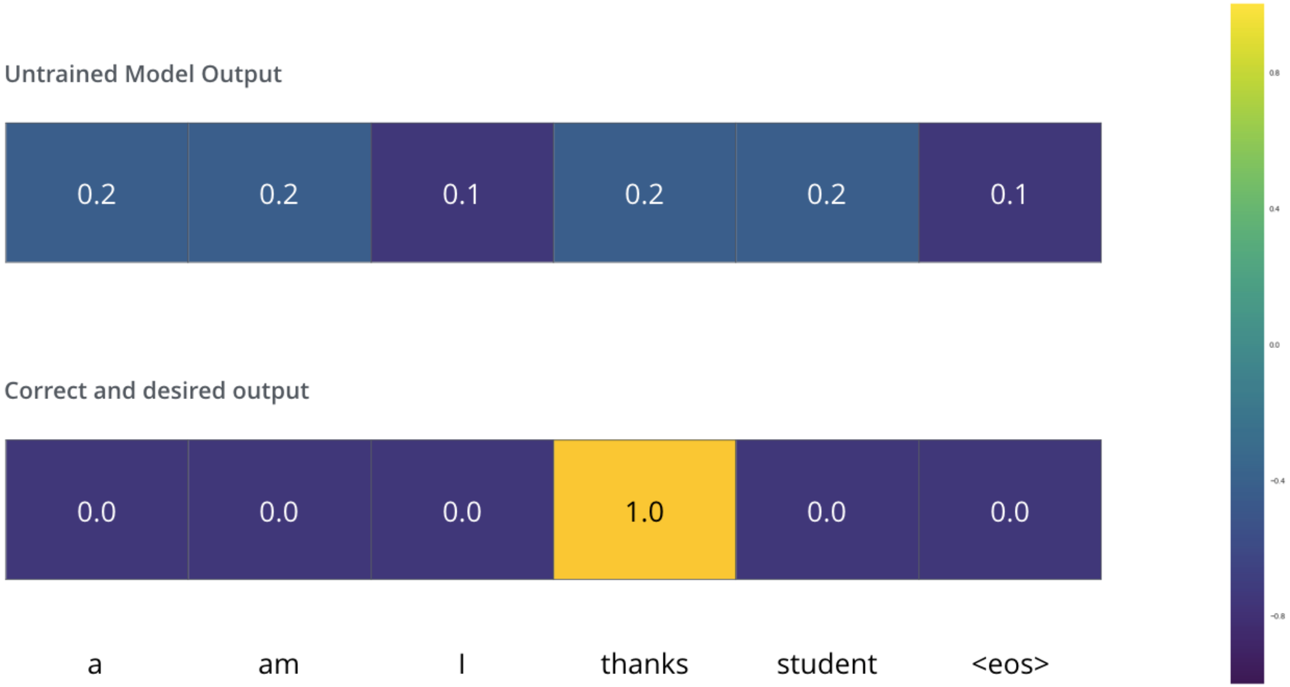
Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word “am”

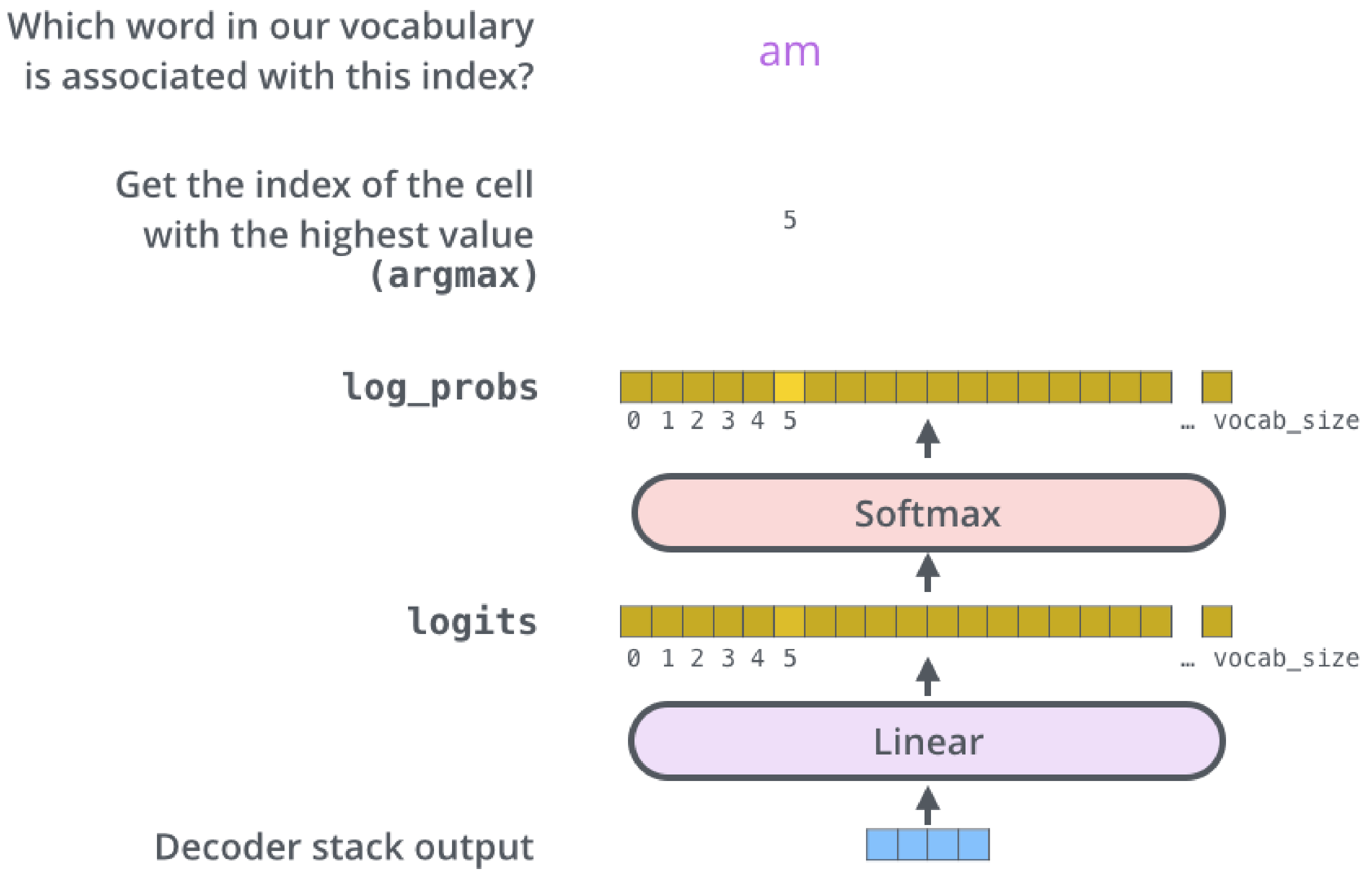
0.0	1.0	0.0	0.0	0.0	0.0

The Decoder Side

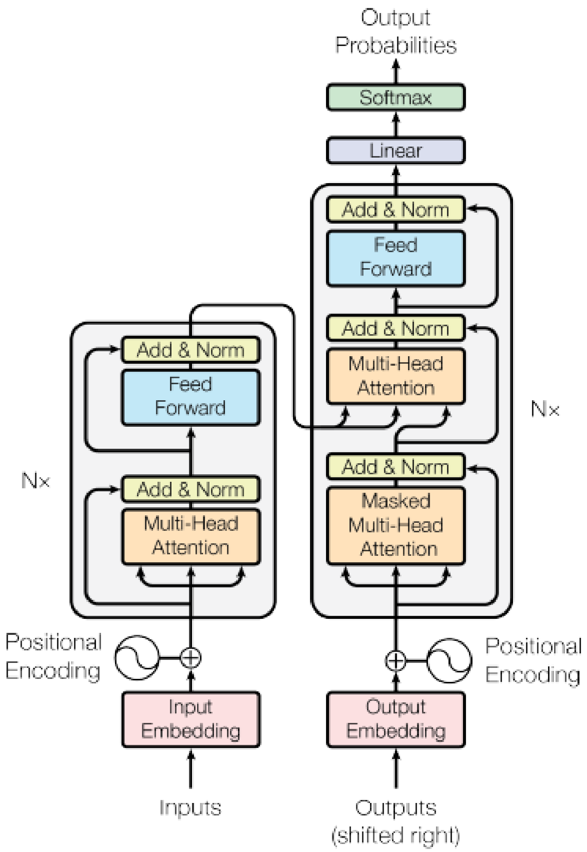


Since the model's parameters (weights) are all initialized randomly, the (untrained) model produces a probability distribution with arbitrary values for each cell/word. We can compare it with the actual output, then tweak all the model's weights using backpropagation to make the output closer to the desired output.

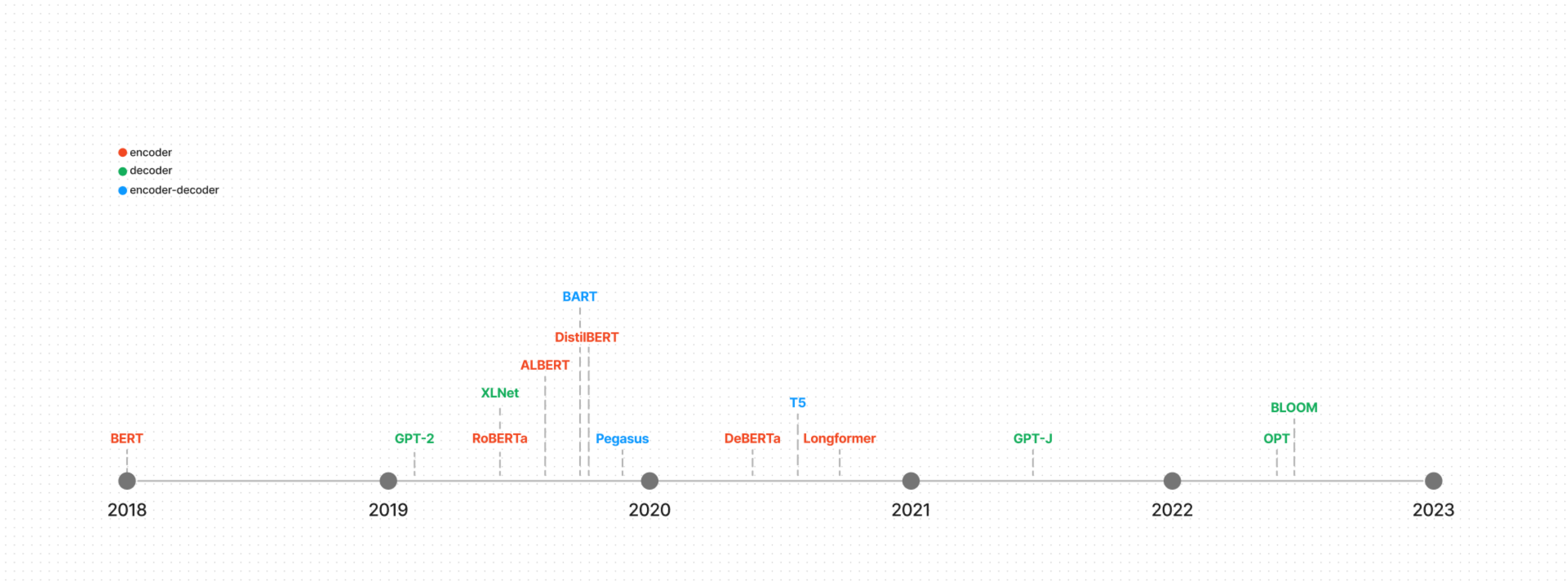
The Final Linear and Softmax Layer



Transformers Architecture - Recap

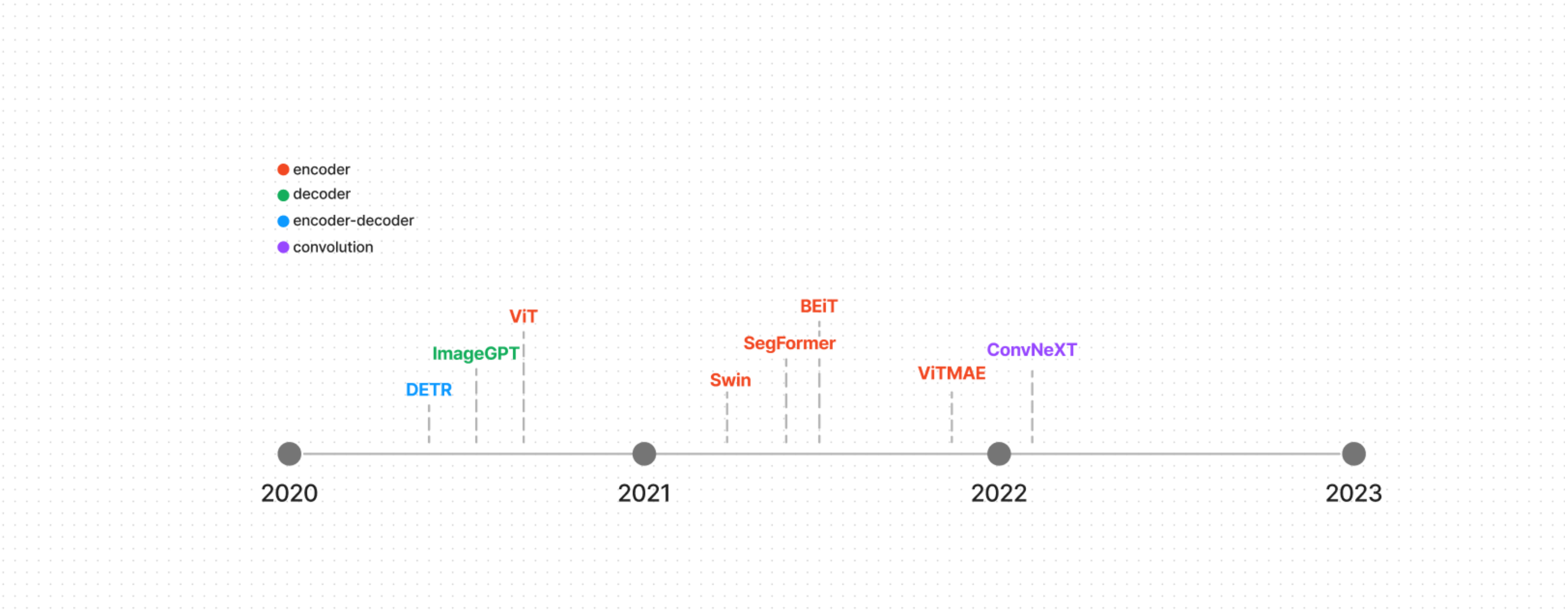


Transformers model timeline - NLP



Source: https://huggingface.co/docs/transformers/model_summary

Transformers model timeline – CV



Source: https://huggingface.co/docs/transformers/model_summary