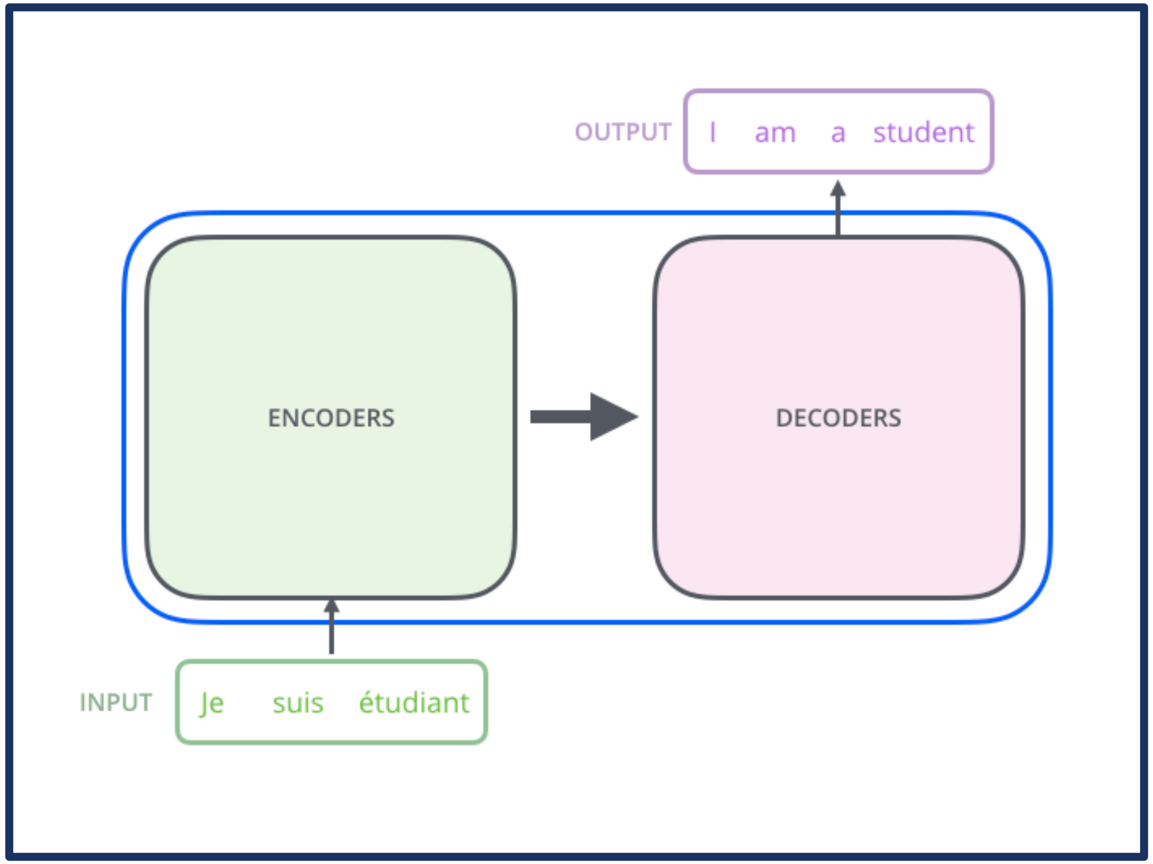
Lecture 09: Transformers

OVERVIEW

1. Transformer architecture
2. Examples of transformers
3. Transfer Learning

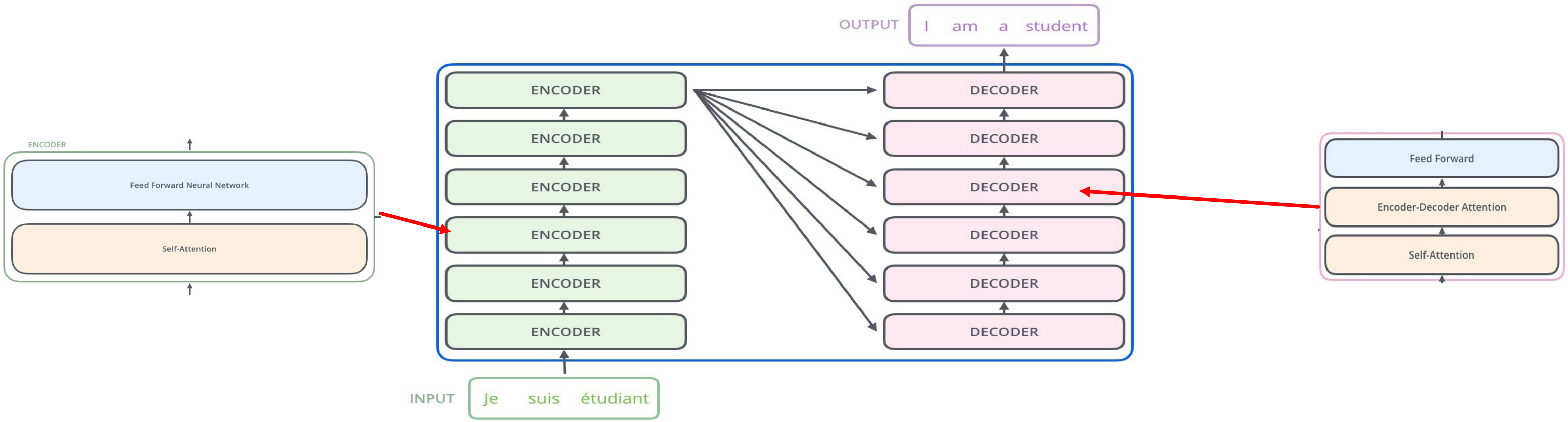
TRANSFORMERS



- **The Transformer** – proposed by Vaswani A., 2017 (Attention is all you need)
  - uses attention to boost the speed with which these models can be trained.
  - The transformer is based on a feedforward neural network rather than RNN.
  - The biggest benefit of the Transformer is the speed of computation due to parallelization.
  - All input sequences are processed simulataneously.

TRANSFORMER – ARCHITECTURE

- Architecture: stacked encoder – decoder architecture
  - The encoding component is a stack of encoders (original paper had six encoders)
    - encoders are all identical in structure
  - The decoding component is a stack of decoders of the same number as the encoder



TRANSFORMER – ARCHITECTURE

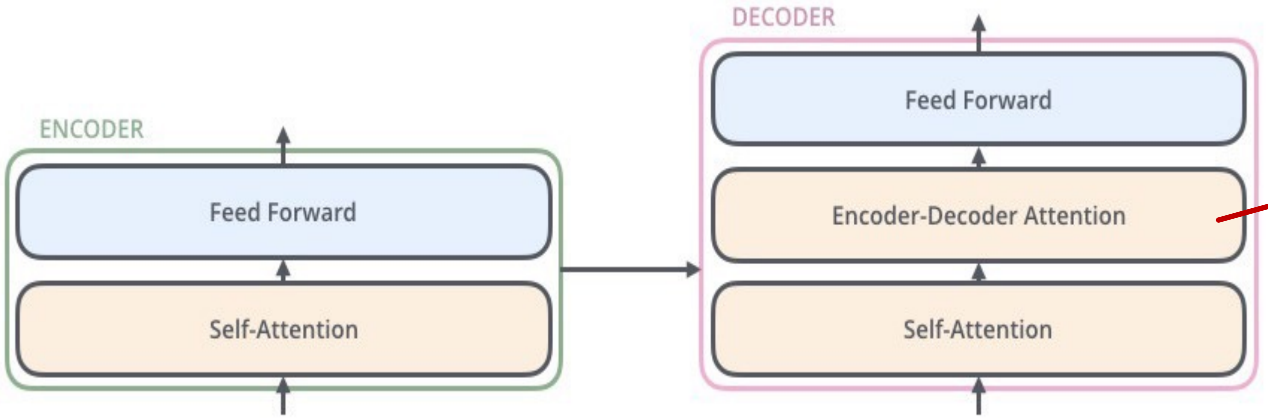
Encoder

Inputs first flow through a self-attention layer

Outputs of the self-attention layer are fed to a feed-forward neural network.

Decoder

Input from encoder flows through self-attention layer then Encoder-Decoder Attention layer and finally through Feedforward layer

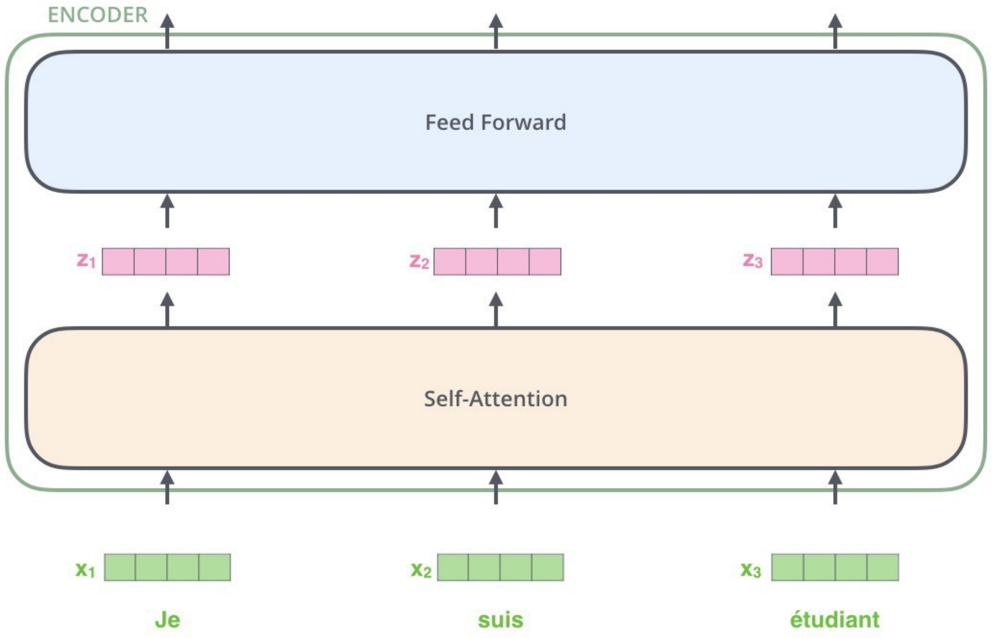


Attention layer:

- helps decoder focus on relevant parts of the input

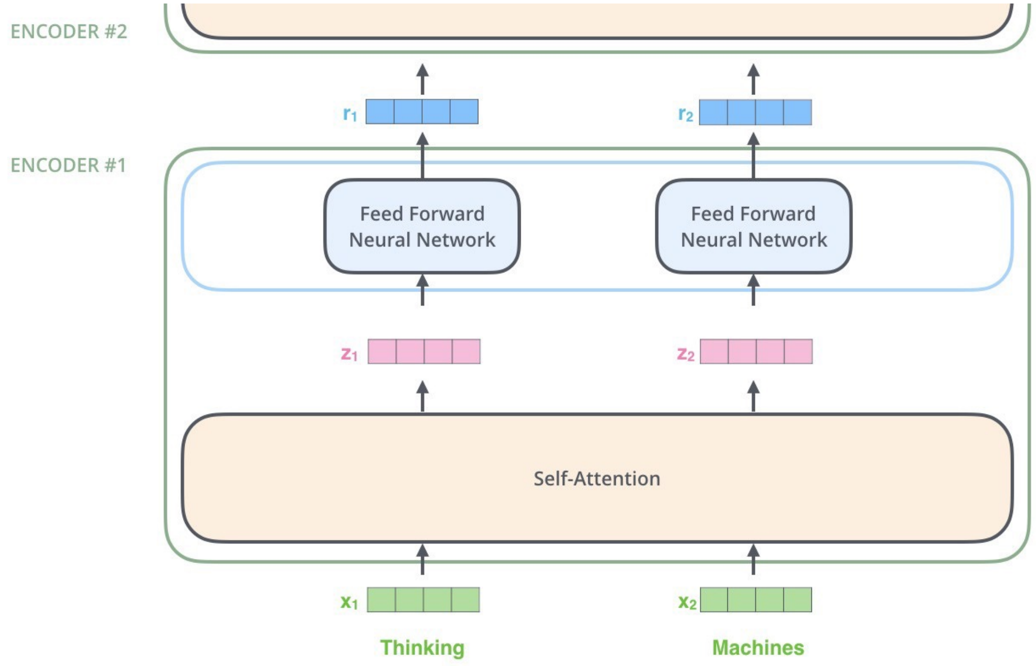
HOW THE TRANSFORMER WORKS

- Convert input word into a word embedding vectors
- The words in the sentence flows through its own path in the encoder



HOW THE TRANSFORMER WORKS – ENCODING

- Encoder receive input words as vectors
- Self attention processes the input and sent to the feedforward network
- Feedforward sends output to next encoder layer.



TRANSFORMER – WHAT IS SELF ATTENTION?

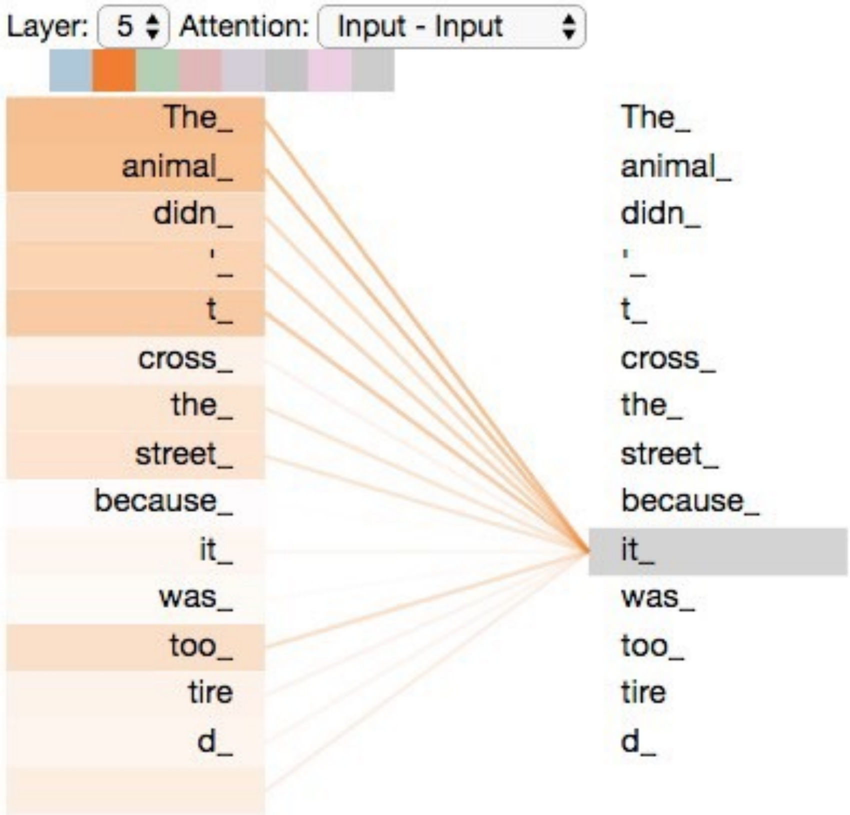
Example:

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

What does "***it***" in this sentence refer to?

- *Is it the street or*
- *the animal*

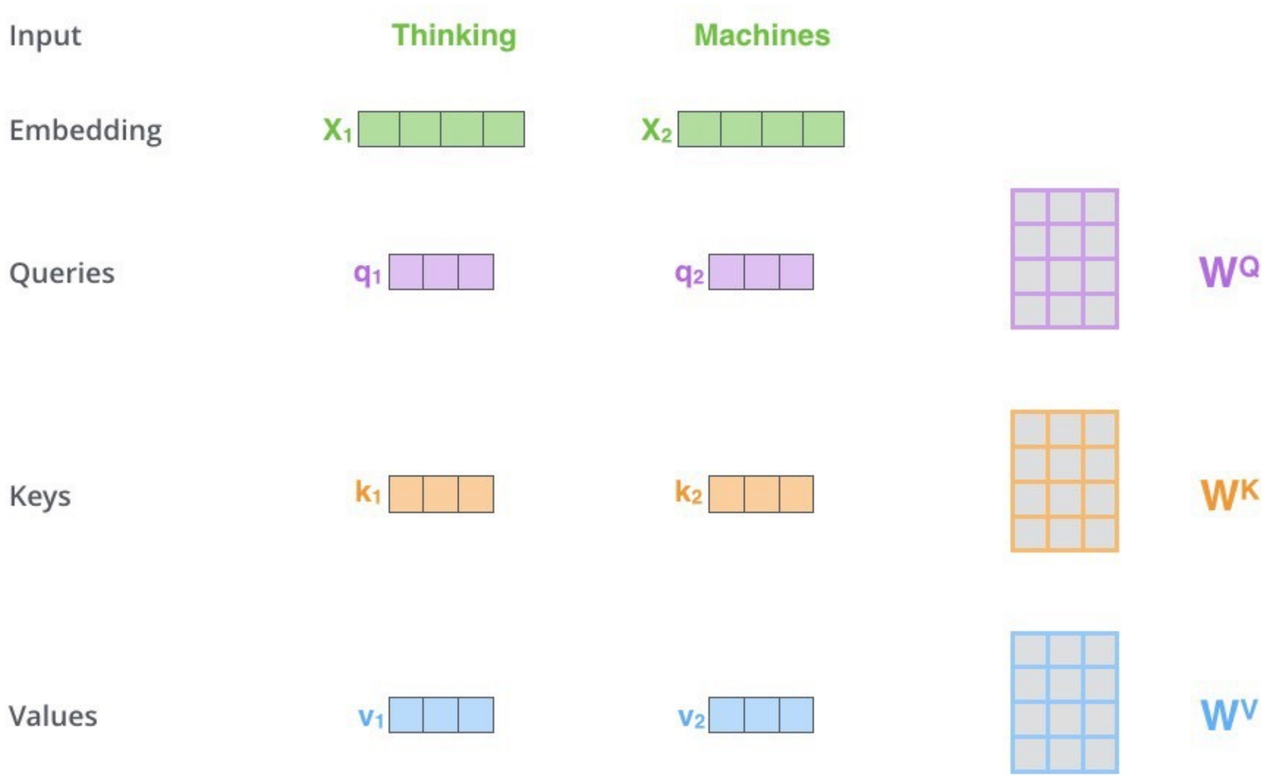
- self-attention allows the transformer model to associate "***it***" with "***animal***".
- self attention allows it to look at other positions for clues to help the model better encode the word ***it***



https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

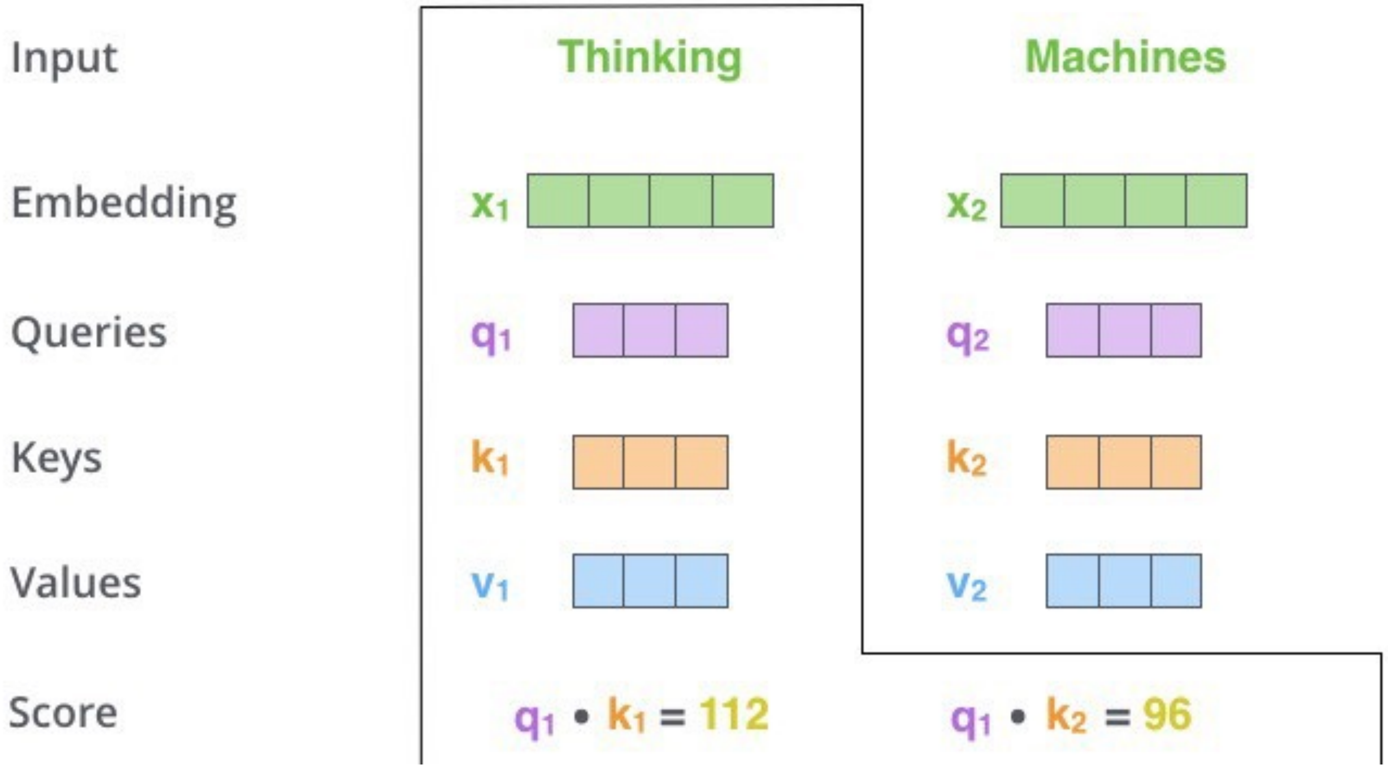
HOW TO CALCULATE SELF ATTENTION – STEP 1

Create 3 vectors (***Q***uery, ***k***ey, ***V***alue) from each encoder's input vector



HOW TO CALCULATE SELF ATTENTION – STEP 2

calculate self-attention score



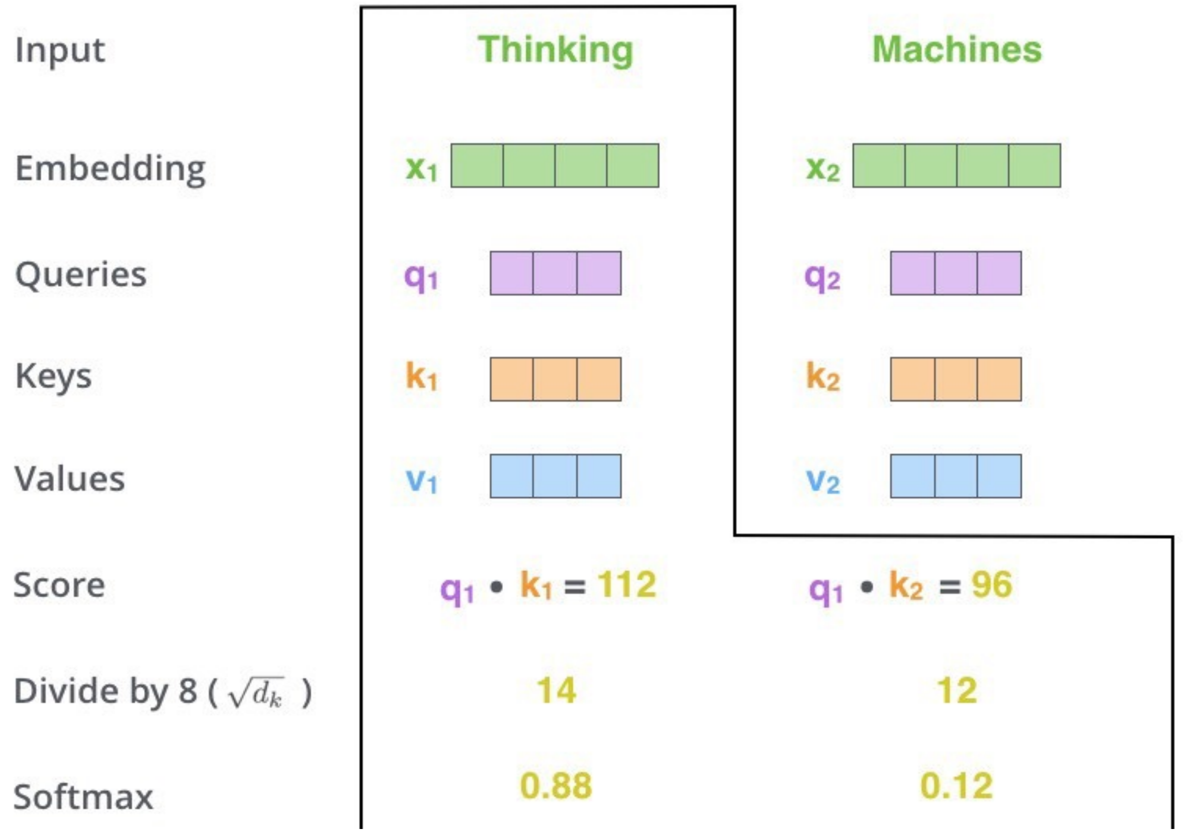
HOW TO CALCULATE SELF ATTENTION – STEP 3 & 4

Step 3:

- divide the core by 8
- square root of the dimension of the key vector

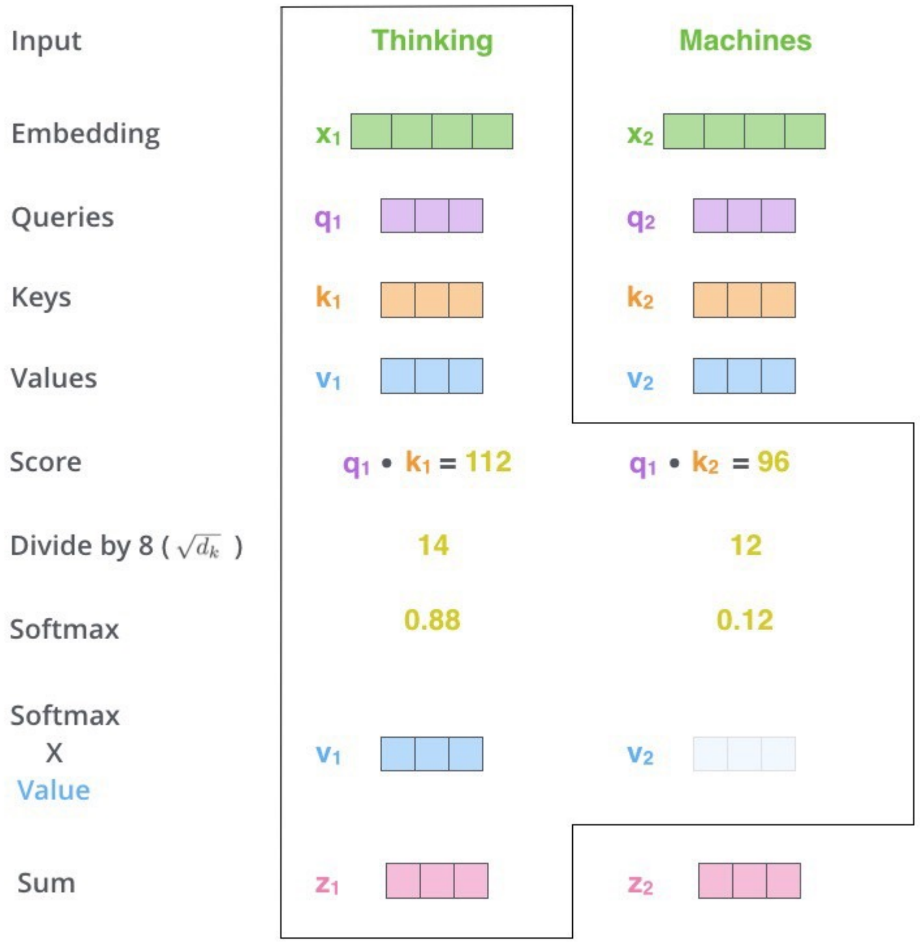
Step 4:

- apply SoftMax operation to normalizes scores

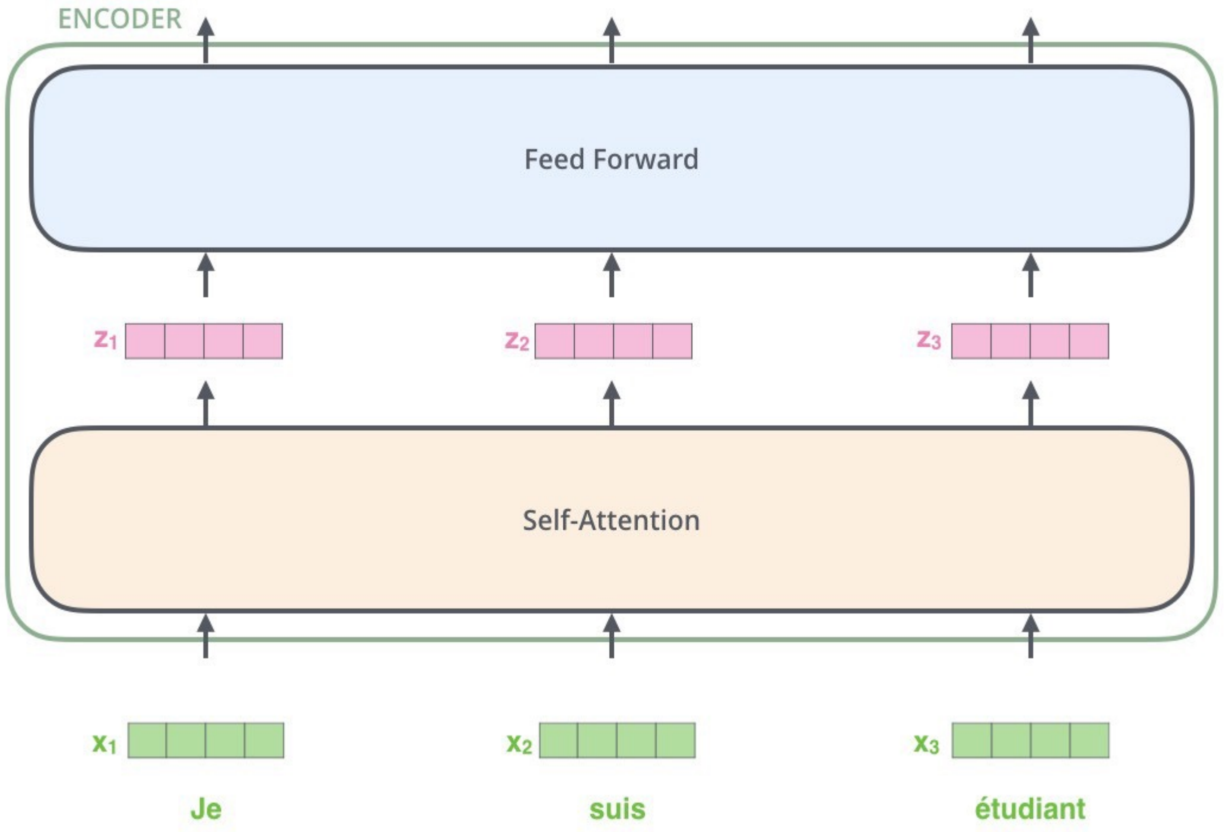


HOW TO CALCULATE SELF ATTENTION – STEP 5 & 6

- **Step 5:**
  - multiply each value vector by the SoftMax score
  - **Intuition:** to amplify relevant words and down grade irrelevant words
- **Step 6:**
  - sum up the weighted value vectors.
  - produces the output of the self- attention layer



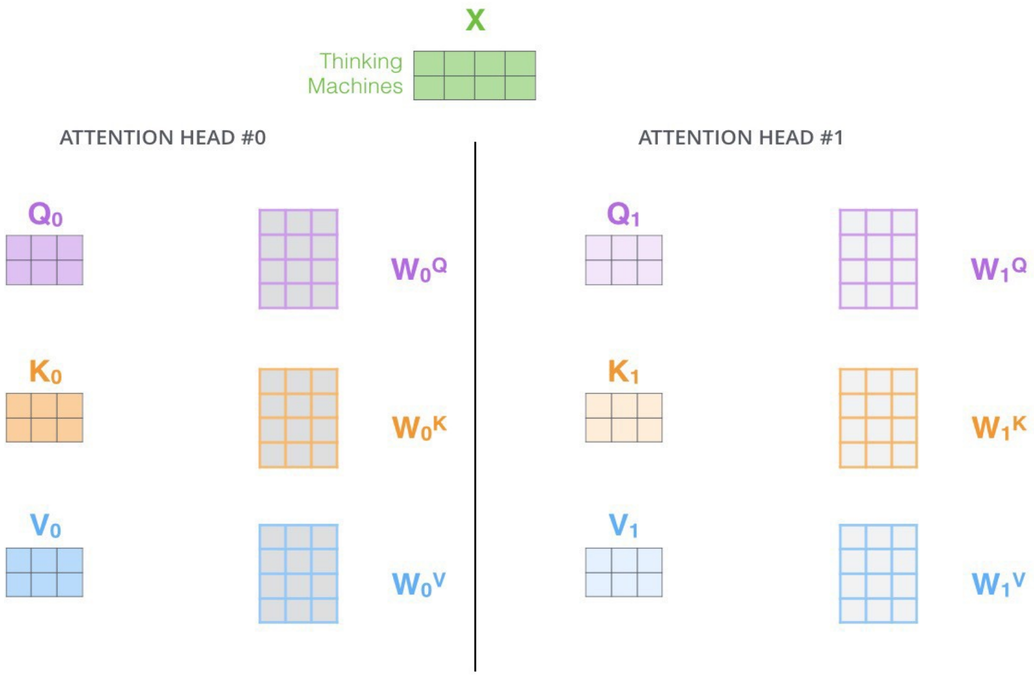
HOW TO CALCULATE SELF ATTENTION

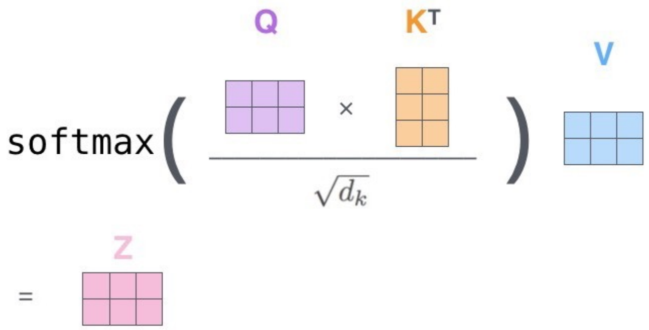


MULTI-HEADED ATTENTION

Multi-headed attention improves the performance of the attention layer by allowing for:

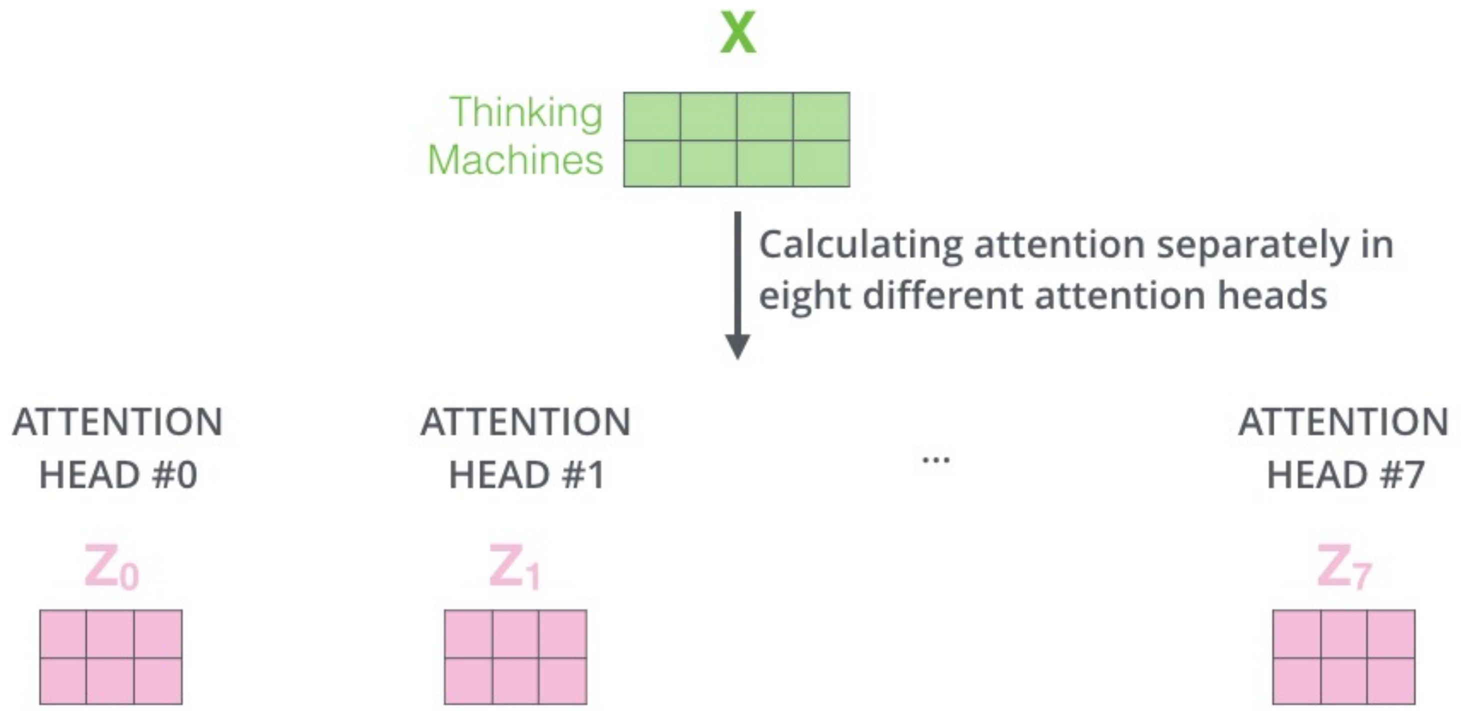
1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces".





MULTI-HEADED ATTENTION

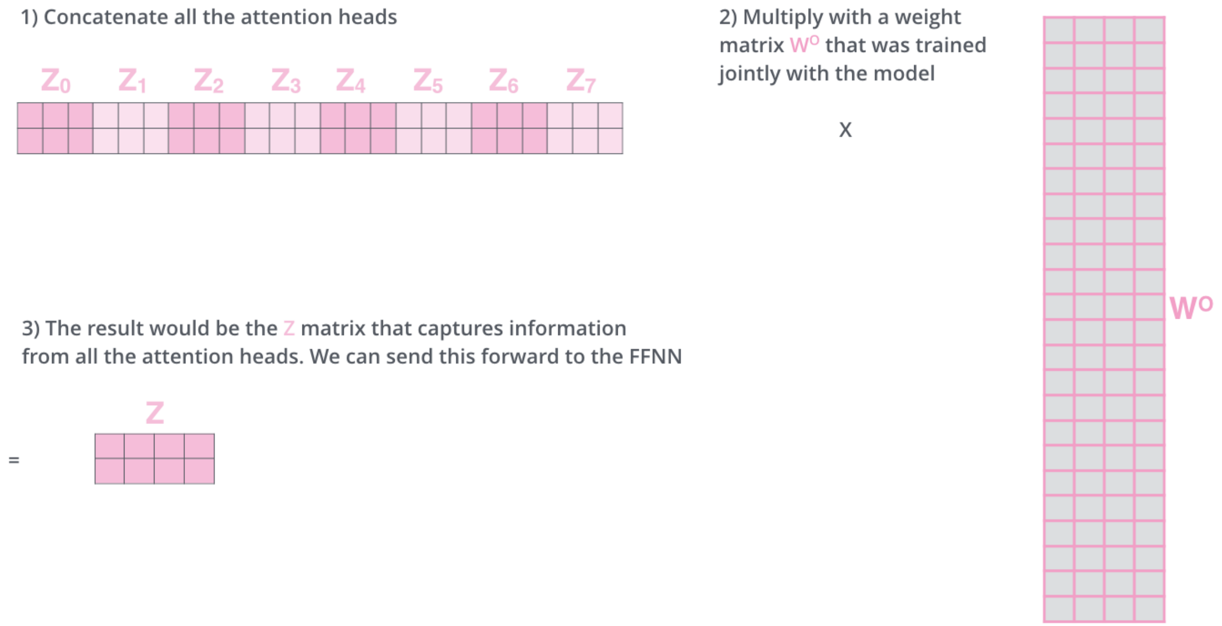
Assume we have multi-headed attention with 8 self attention (eight *Z* matrices)



MULTI-HEADED ATTENTION

Assume we have multi-headed attention with 8 self attention (eight ***Z*** matrices)

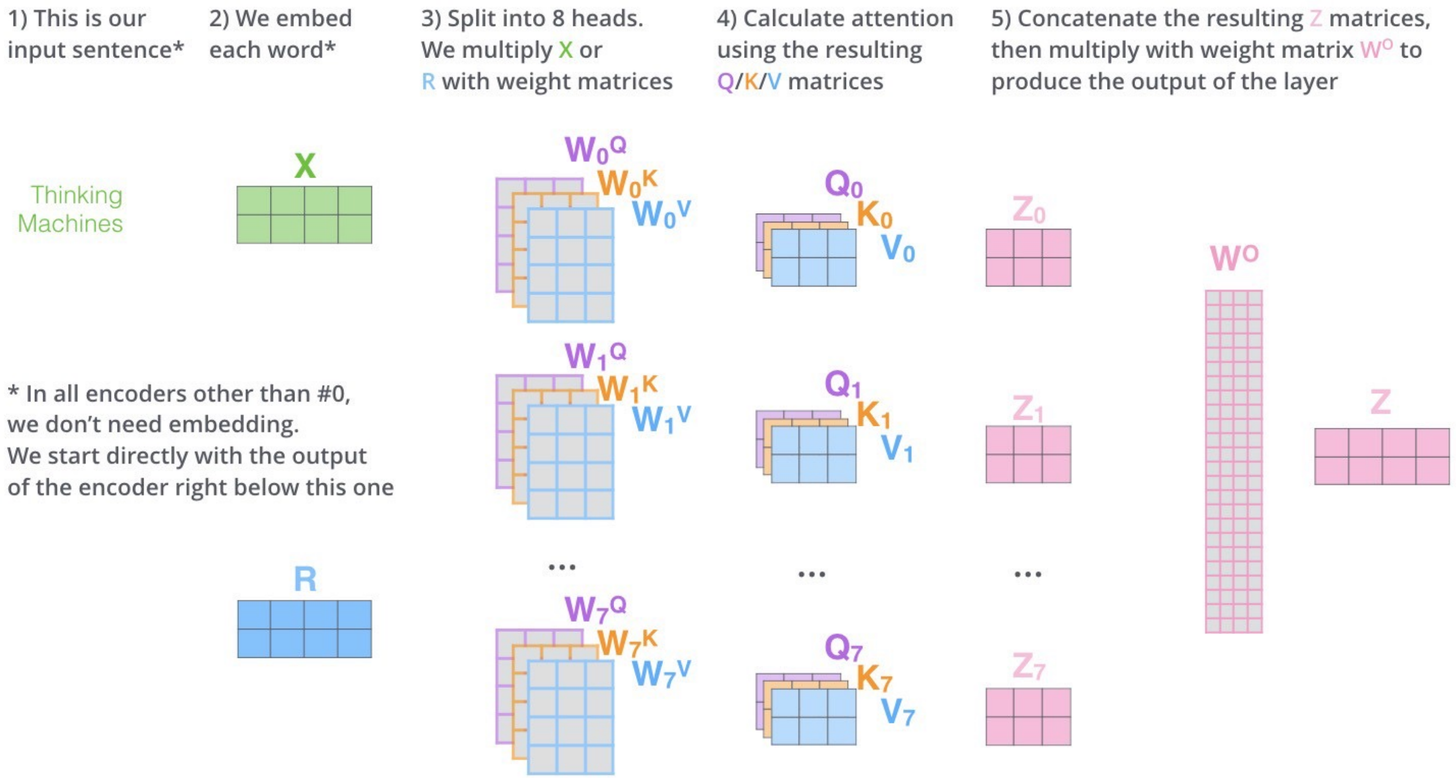
- the feedforward layer expects a single matrix
- We concatenate the matrices then multiply them by an additional weight matrix W^O



MULTI-HEADED ATTENTION

Assume we have multi-headed attention with 8 self attention (eight *Z* matrices)

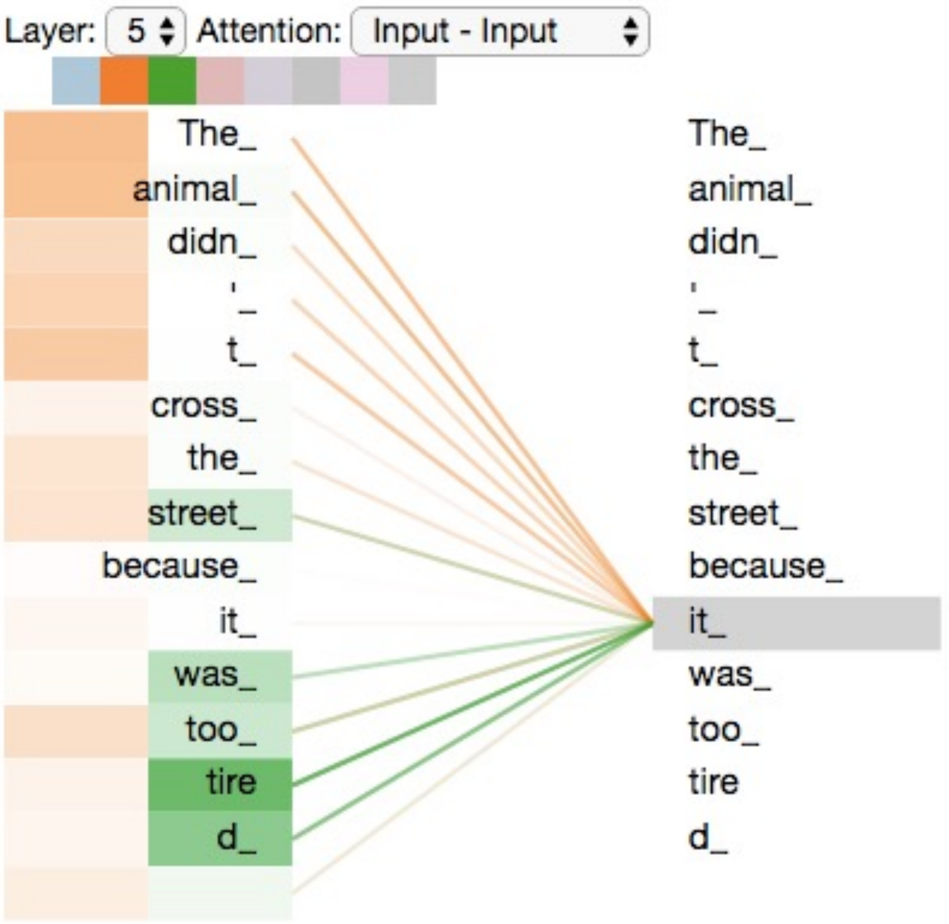
- the feedforward layer expects a single matrix



MULTI-HEADED ATTENTION

Multi head attention "*it*"

- one attention head is focusing on most on ***the animal***
- another is focusing on ***tired***

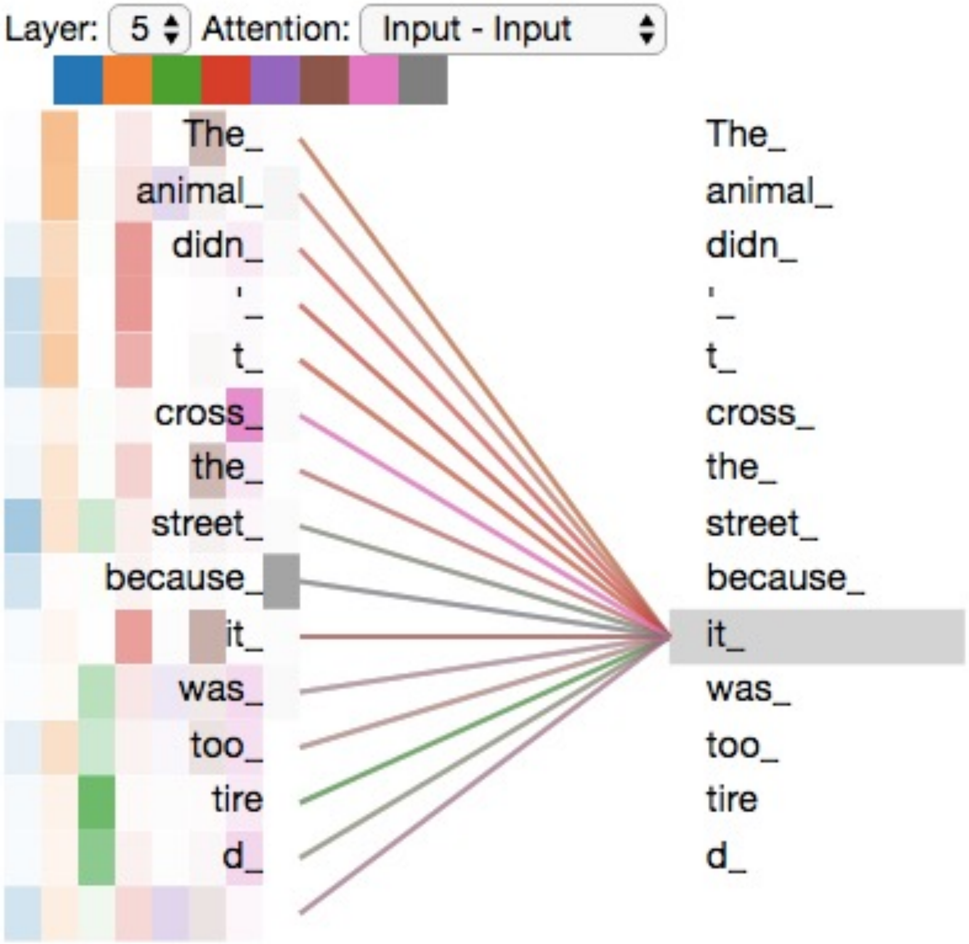


MULTI-HEADED ATTENTION

Multi head attention "*it*"

- one attention head is focusing on most on ***the animal***
- another is focusing on ***tired***

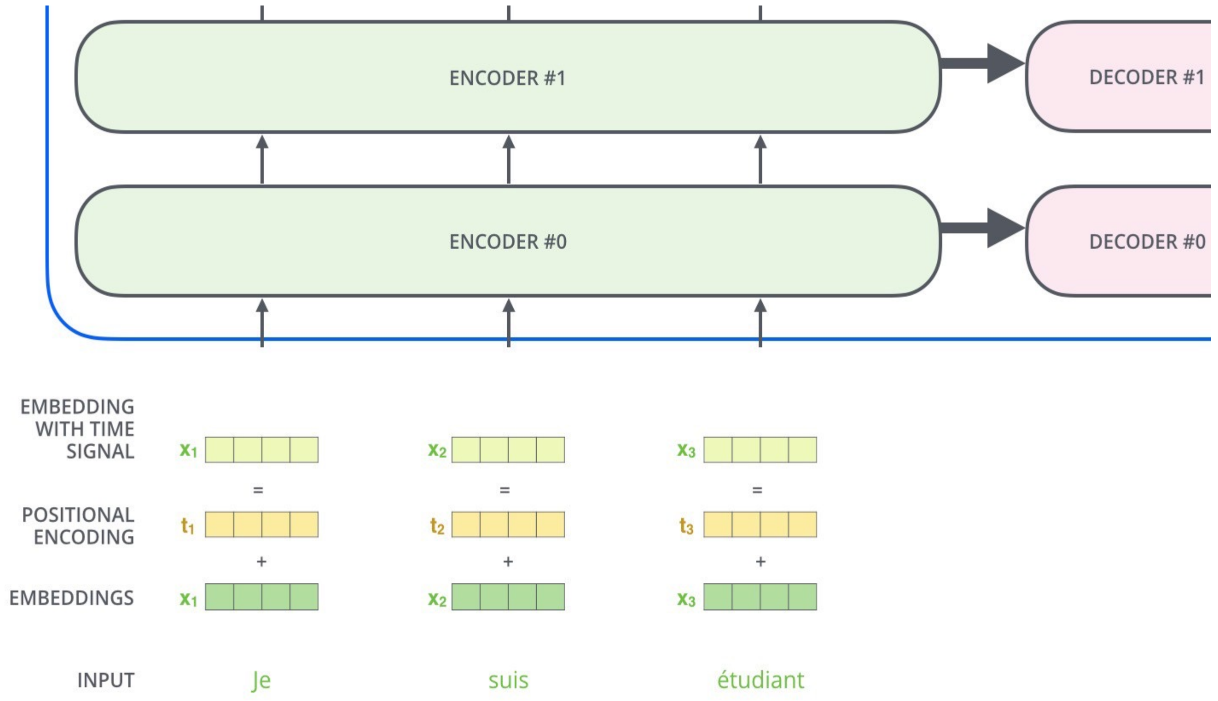
- Visualizing all 8 attention heads



POSITIONAL ENCODING IN TRANSFORMERS

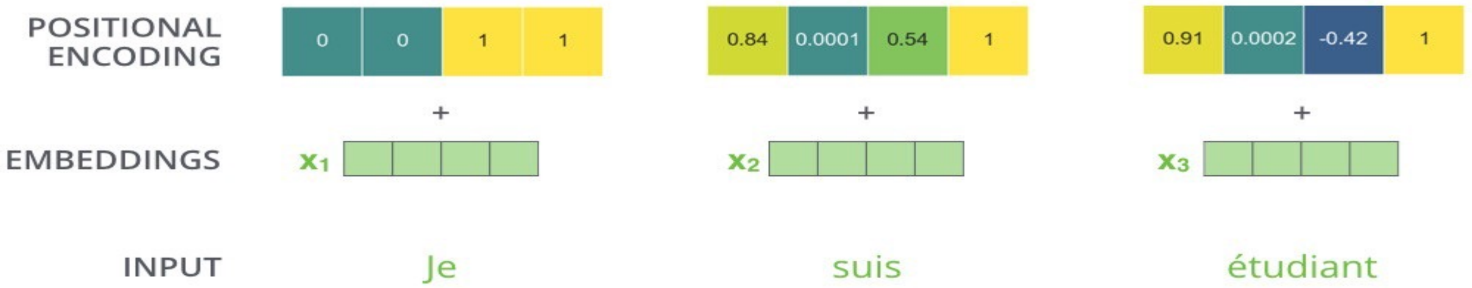
How do we account for the order of the words?

- Transformer adds a vector to each input embedding
- These vectors allow it to determine the position of each word



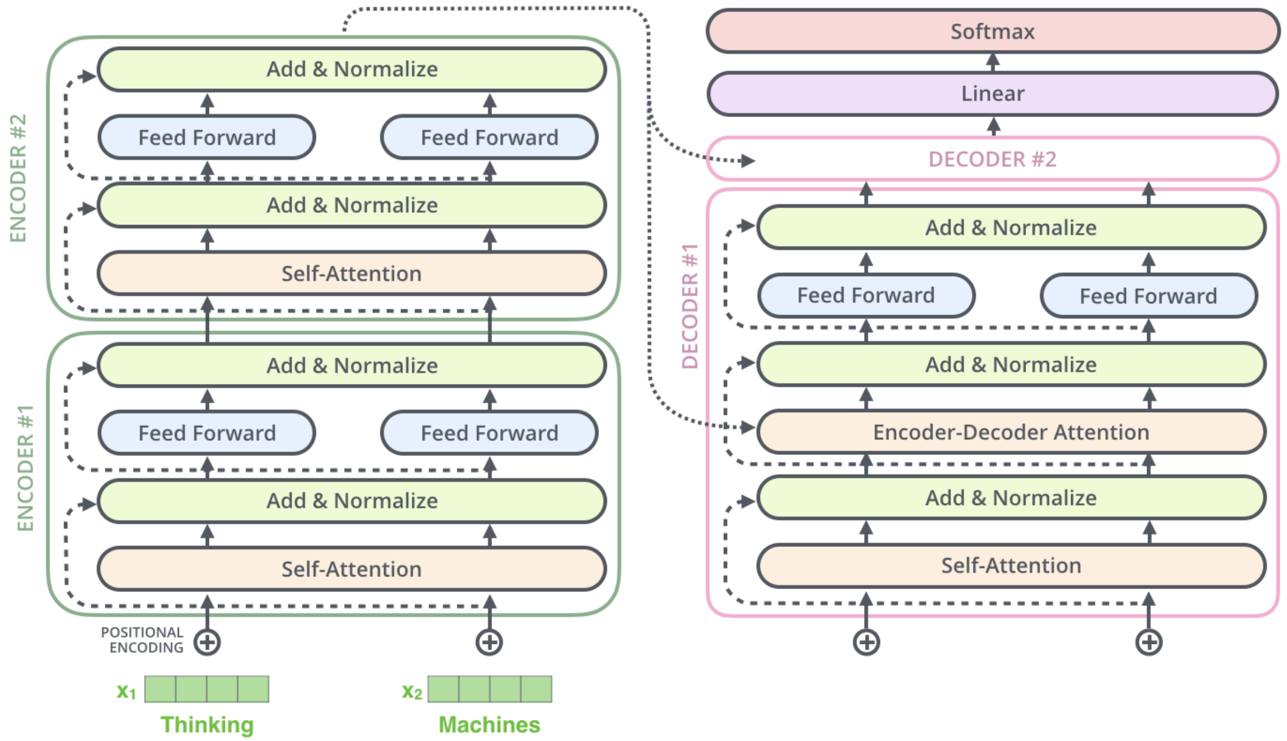
POSITIONAL ENCODING IN TRANSFORMERS

- **Intuition:**
  - adding these values to the embeddings provides meaningful distances between the embedding vectors once they are projected into Q/K/V vectors and during dot-product attention.
  - These positional encoding are generated during training (detail in the paper)
- Assume encoding of dimension 4, the positioning will look like



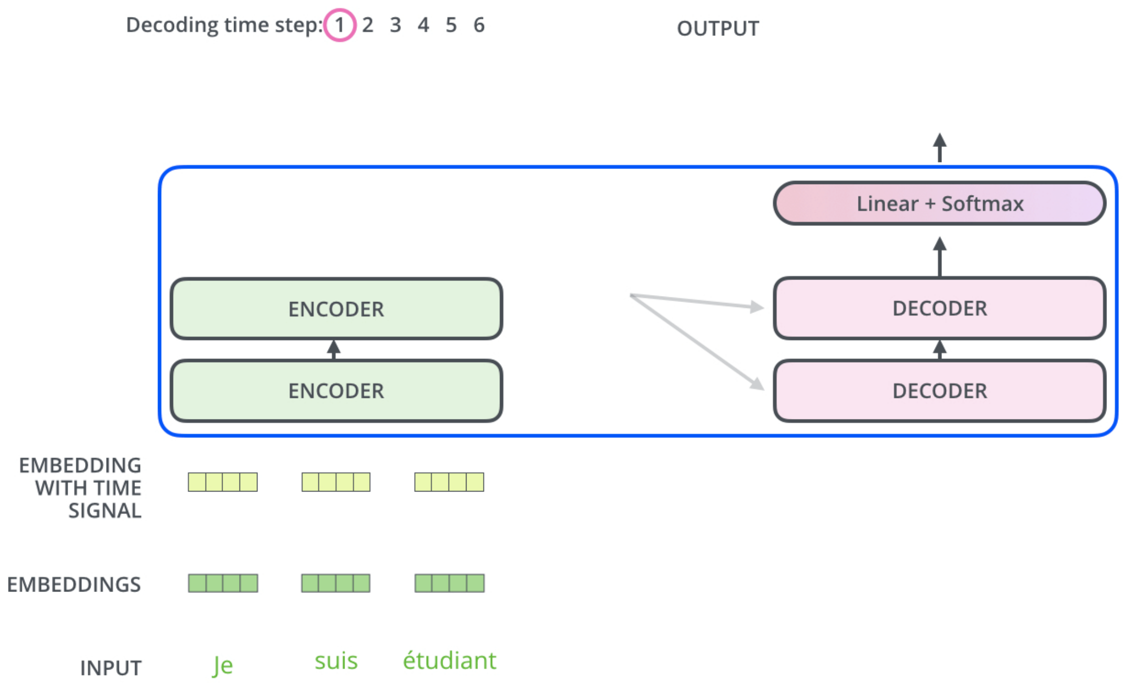
RESIDUAL CONNECTION

Residual connection: Each sub-layer (self-attention, feed forward neural network) in each encoder and decoder have a residual connection around it followed by a layer normalization



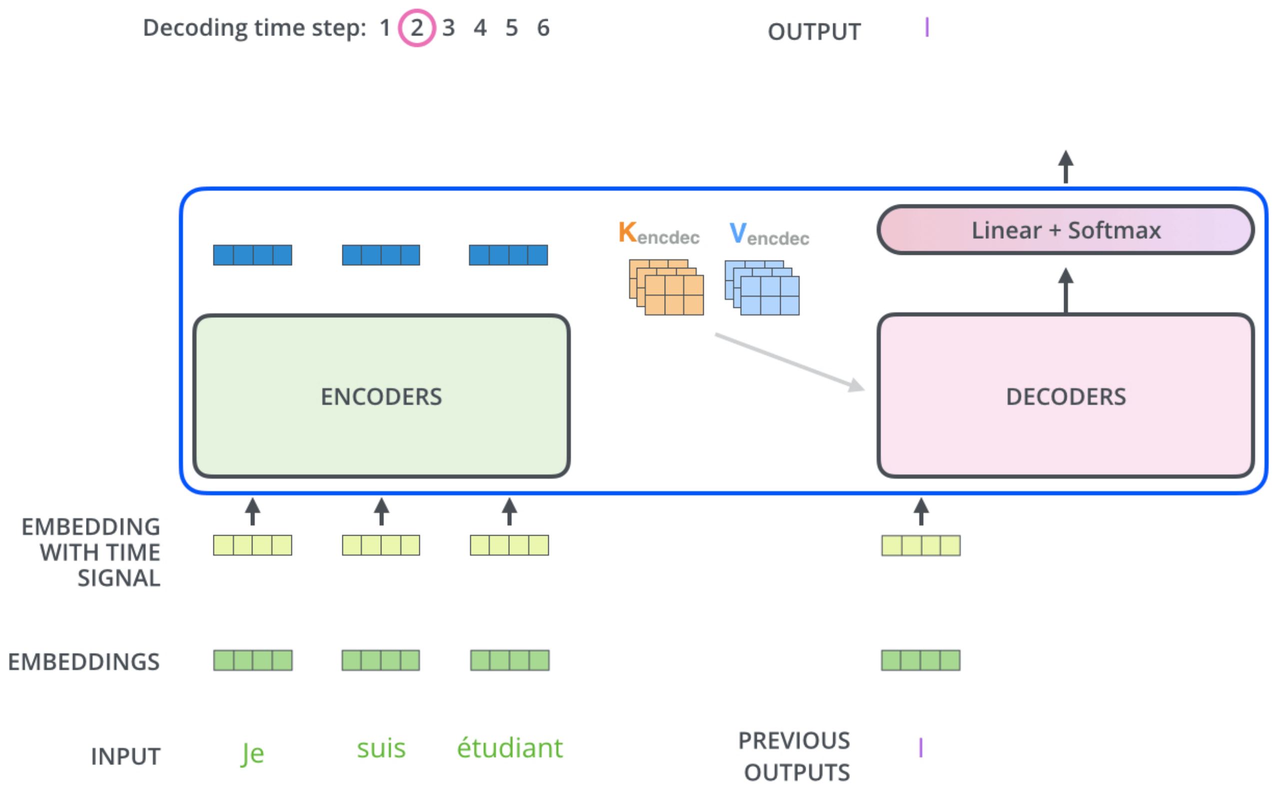
THE DECODER SIDE

- The encoder process the input sentence
- Output of top encoder layer is converted to attention vectors K and V.
- K & V vectors are used by each decoder in its “encoder-decoder attention” layer
- K & V helps decoder focus on appropriate words.



THE DECODER SIDE

The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output



THE DECODER SIDE

The self attention layers is only allowed to focus on earlier positions in the output sequence.

- This is done by masking future positions (setting them to -inf) before the SoftMax step in the self-attention calculation.

The "Encoder-Decoder Attention" layer is like multiheaded self-attention

- except it creates its Queries matrix from the layer below it,
- takes the Keys and Values matrix from the output of the encoder stack.

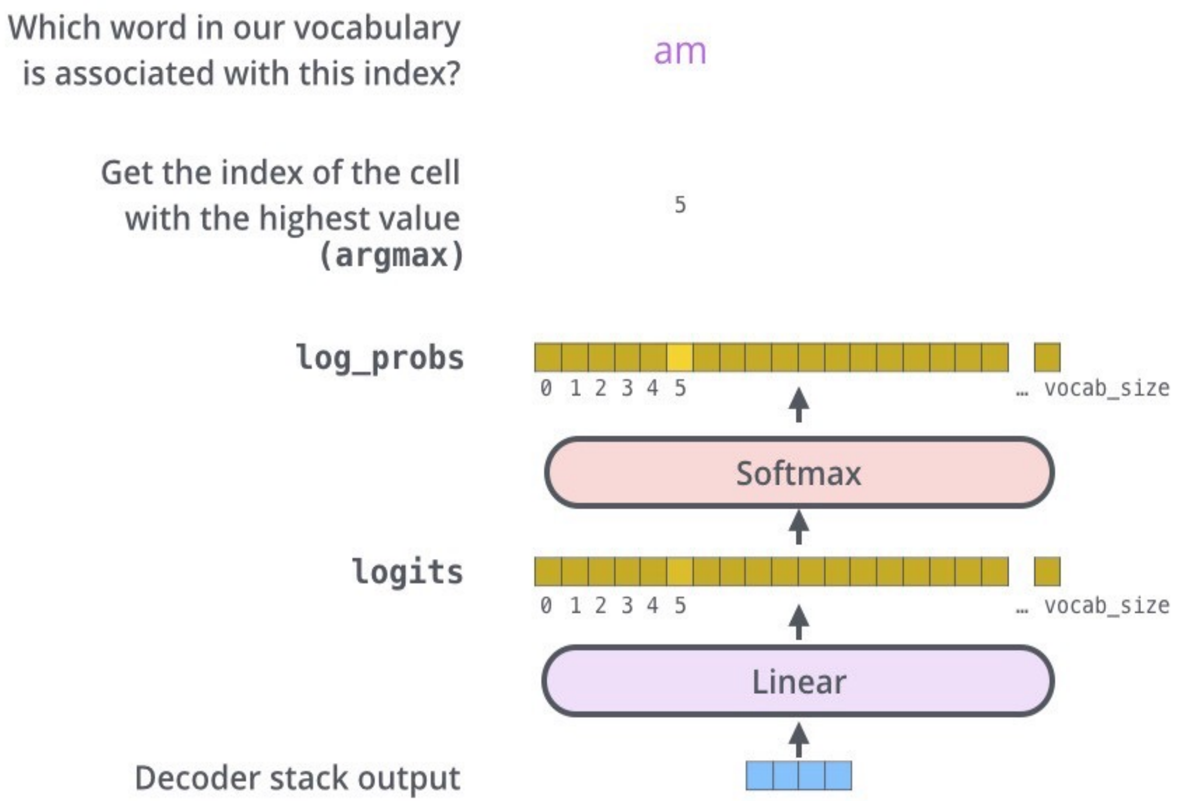
FINAL LINEAR AND SOFTMAX LAYER

Final Linear Layer:

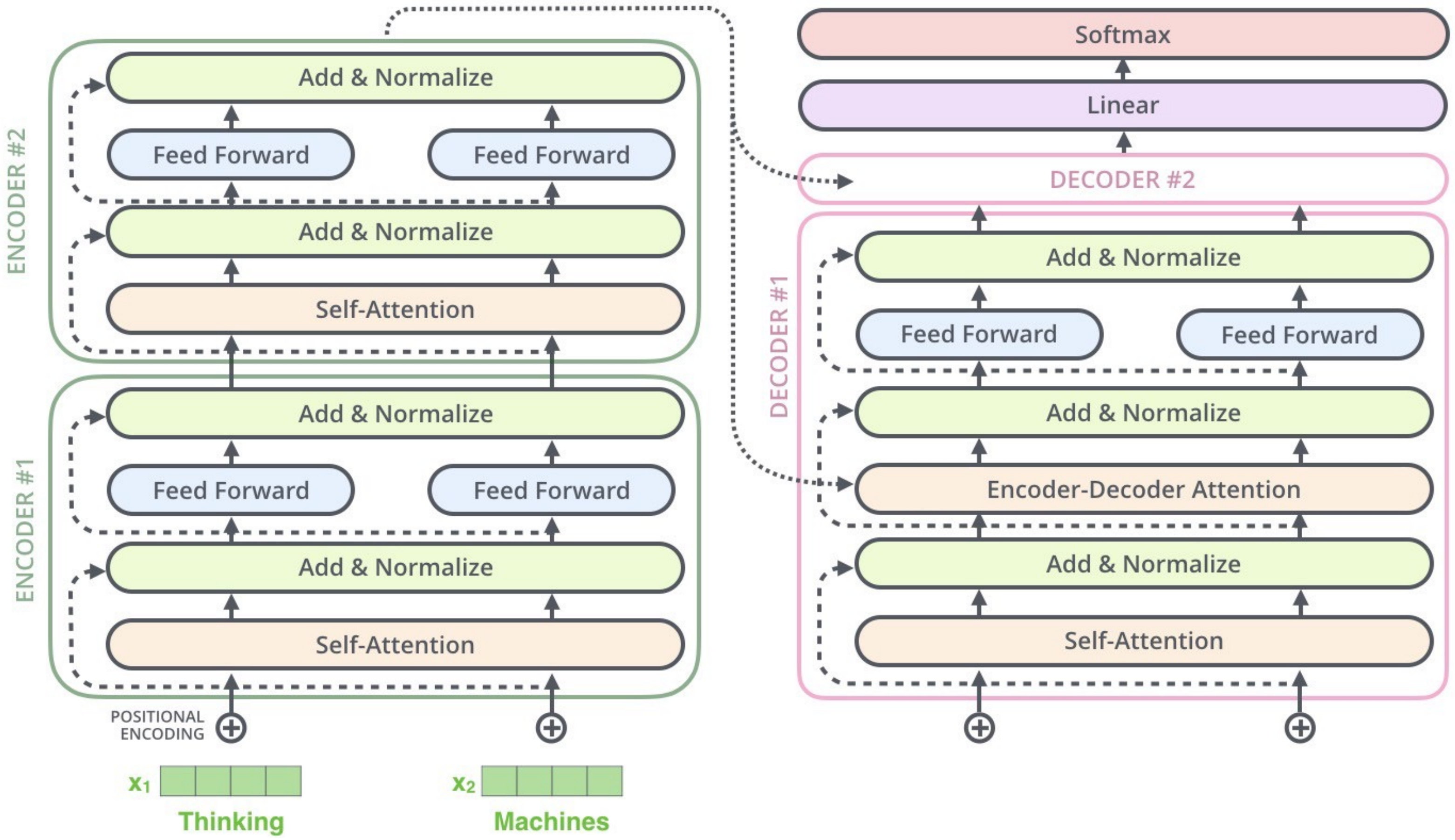
- fully connected connected neural network that projects the vector produced by the decoders into a logits vector.

Assume our model was trained on a 1000-word vocabulary

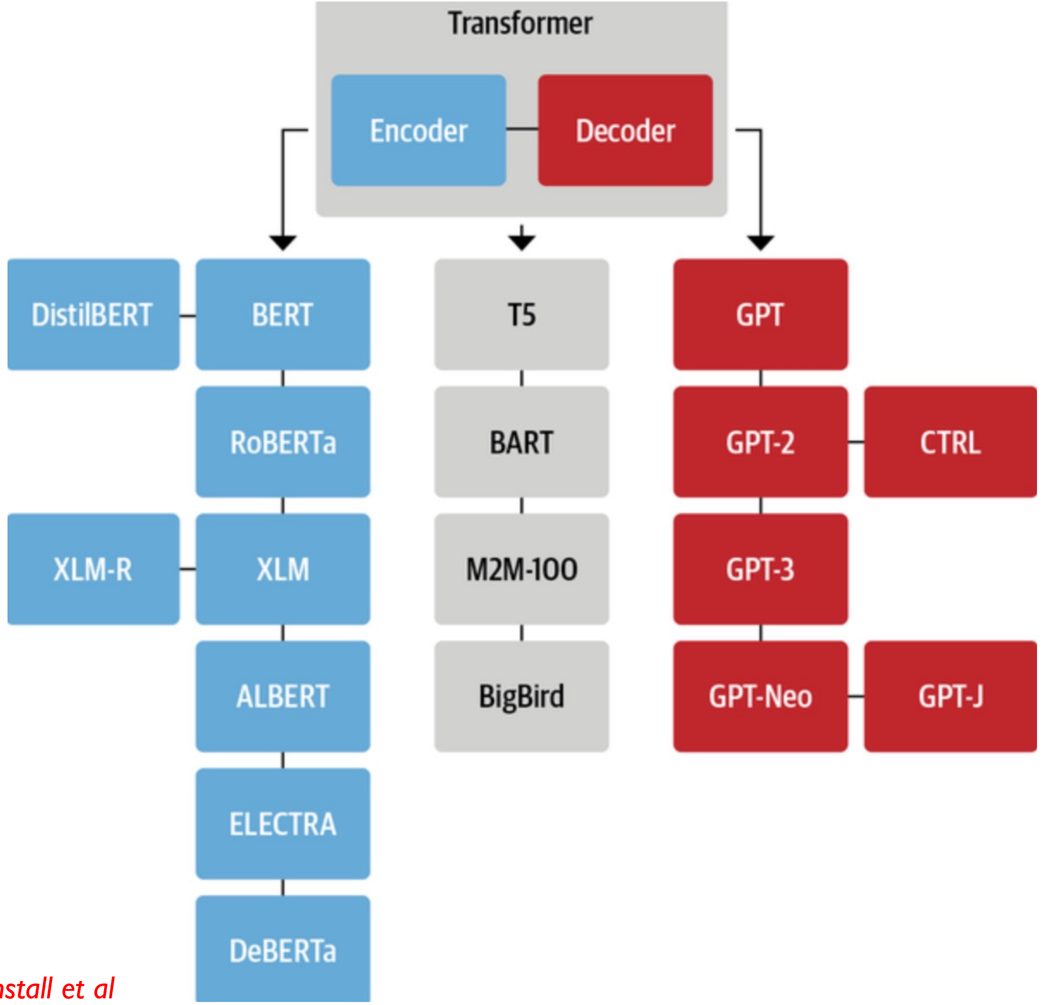
- Then logit vector has dimension 1000 cells
- Each cell the score of a unique word
- The SoftMax layers turns these scores into probabilities and the word with the highest probability is chosen.



TRANSFORMER ARCHITECTURE – TWO STACKED ENCODERS



OVERVIEW OF MOST PROMINENT TRANSFORMERS



Natural language processing with transformers by Tunstall et al

USEFUL LINKS

- Attention is all you need (https://arxiv.org/abs/1706.03762)
- Jupyter Notebook (https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb)
- Tensor2Tensor repo (https://github.com/tensorflow/tensor2tensor)
- Huggingface
  - https://huggingface.co/transformers/index.html
  - https://github.com/huggingface/transformers