
TRANSFORMERS



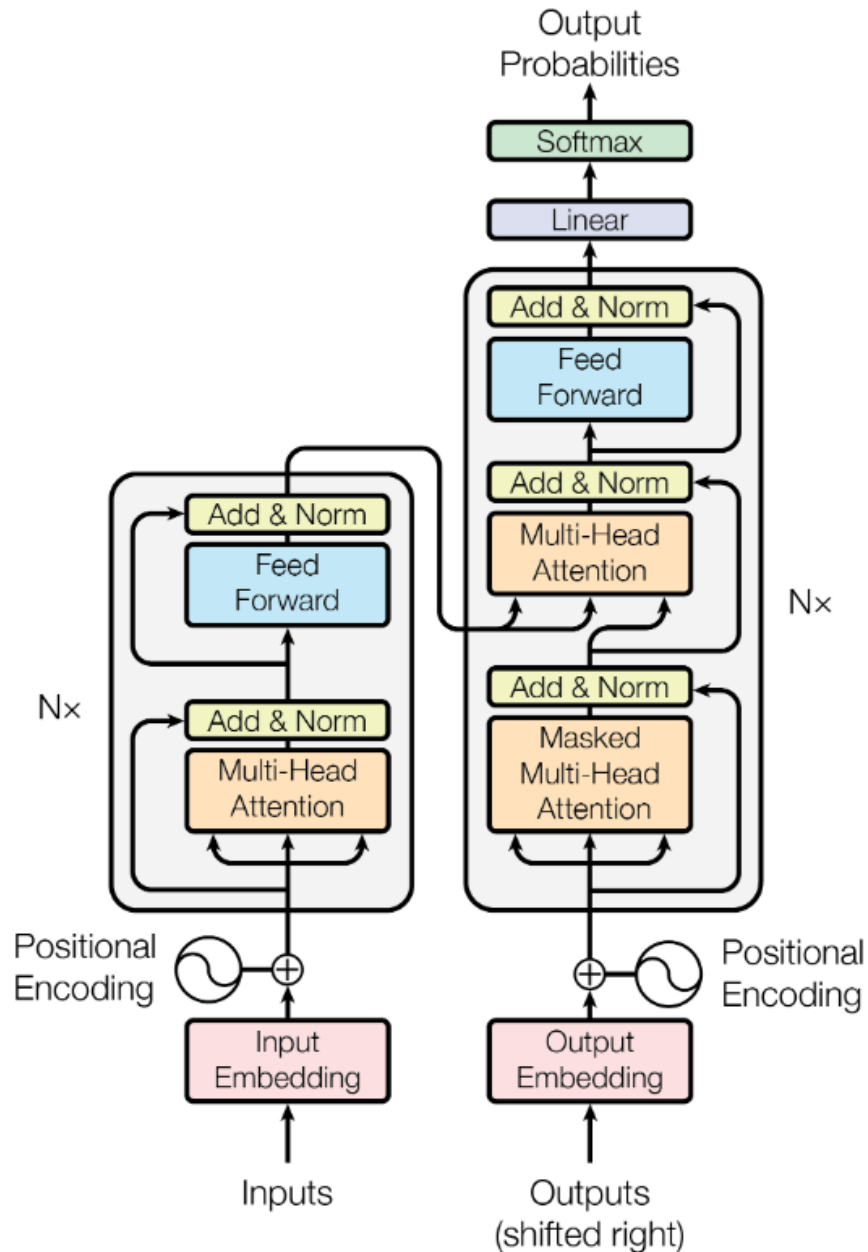


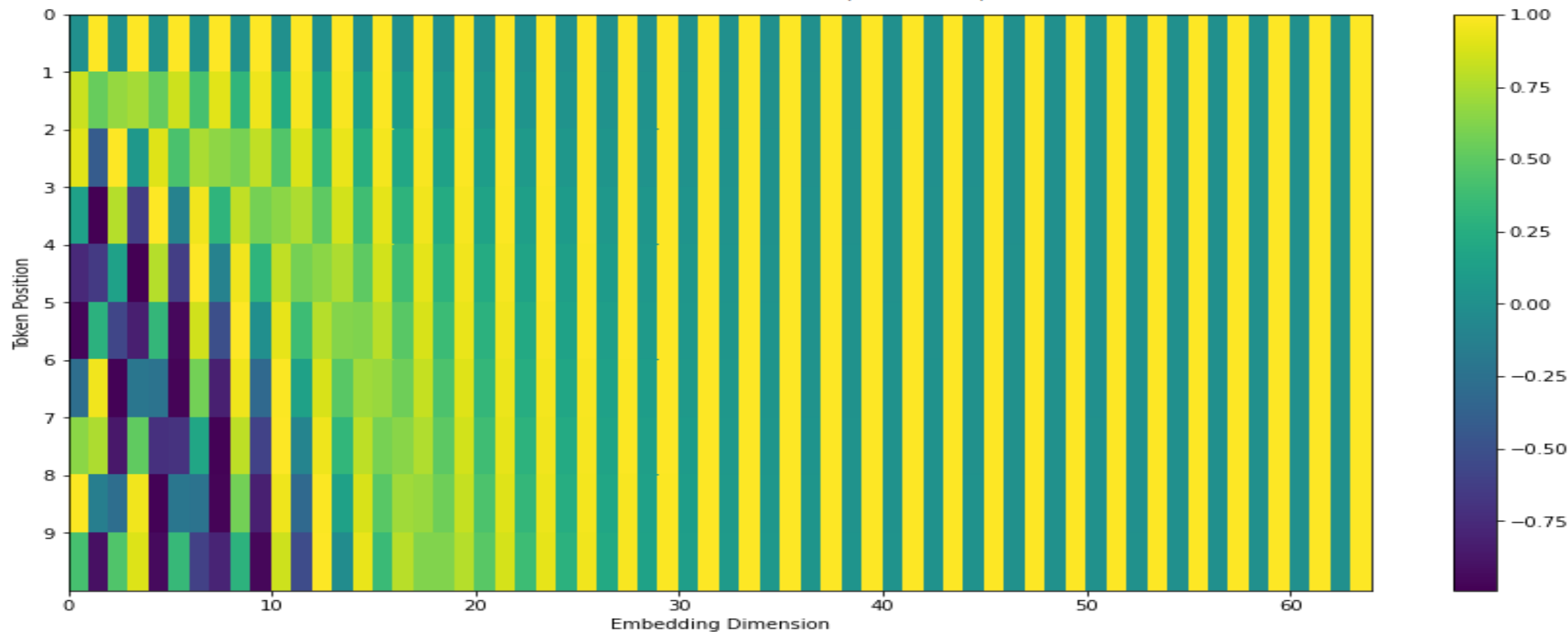
Figure 1: The Transformer - model architecture.

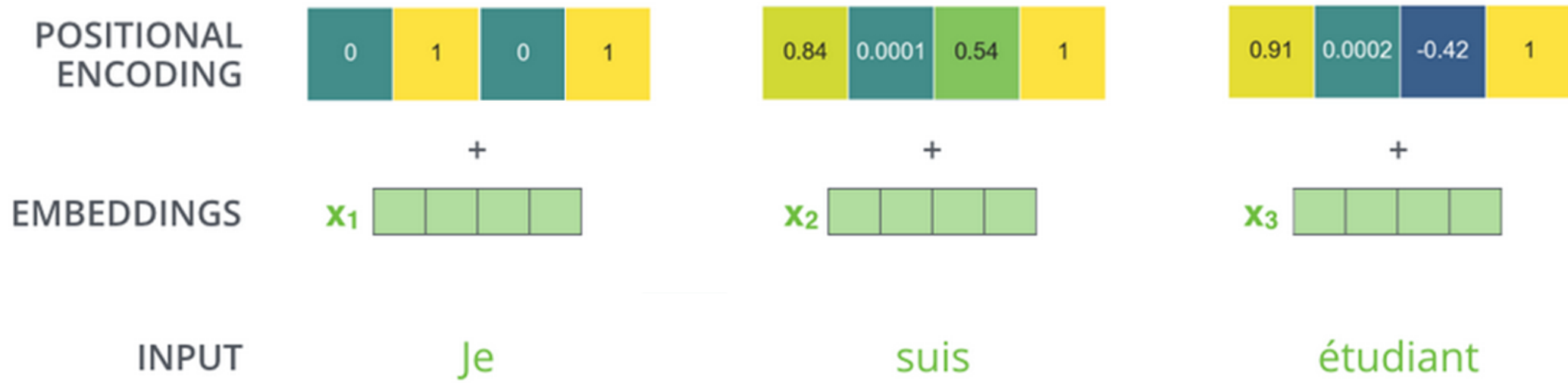
-
- Attention is All You Need
 - <https://arxiv.org/pdf/1706.03762.pdf>
 - First transduction model relying entirely on self-attention, without using sequence-aligned RNNs or convolution
 - Input Embedding – representation of the input (sequence)
 - Positional Encoding – element-wise addition of position representation
 - Multi-Head Attention – core building block
 - Add & Norm – element-wise addition + Layer Norm (<https://arxiv.org/abs/1607.06450>)
 - Feed Forward – fully connected neural network (two layers)
 - Linear – one layer of fully connected NN
-

POSITIONAL ENCODING

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

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





A real example of positional encoding with a toy embedding size of 4

$$y = W_e u + PE$$

- $W_e \in \mathbb{R}^{E \times N}$ is the linear embedding – implemented as a Linear Layer
 - $u \in \mathbb{R}^{N \times 1}$ is the input vector (feature)
 - $PE \in \mathbb{R}^{E \times 1}$ is the positional encoding
-

LEARNABLE POSITIONAL EMBEDDING

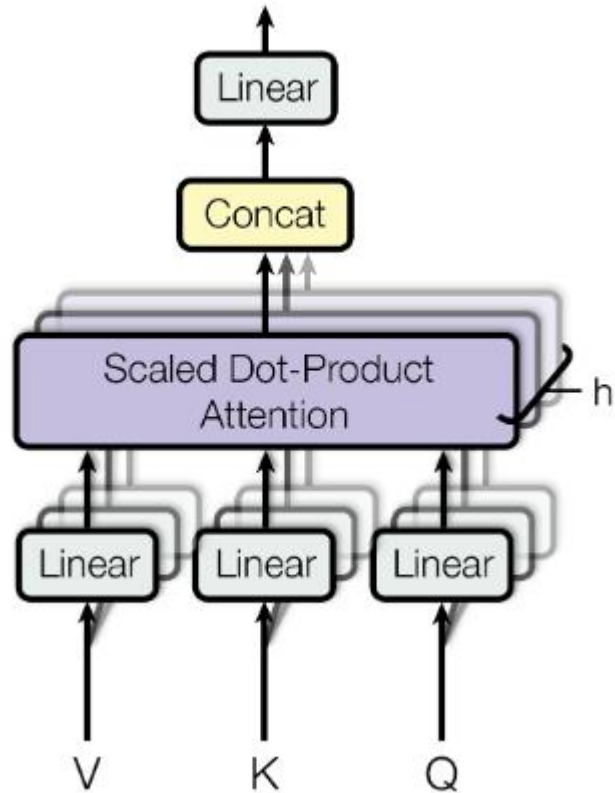
- In many cases it has been demonstrated that instead of sinusoidal PE, we can use a learnable positional embedding.
 - A learnable positional embedding is implemented as a learnable parameter (e.g. `torch.nn.Parameter(max_pos, model_dim)` or `torch.nn.Embedding(max_pos, model_dim)`).
 - $y = W_e u + PE(p, :)$ or $y = W_e u + W_p x_p$
 - Where $PE \in \mathbb{R}^{M \times E}$ is the learnable pos. embedding, and $PE(p, :) \in \mathbb{R}^{M \times 1}$ is the embedding at position p , alternatively $W_p \in \mathbb{R}^{M \times E}$ and $x_p \in \mathbb{R}^{M \times 1}$ is a one-hot-vector representing the position p (i.e., one is p th dimension).
 - M is the maximal allowed position in the input (i.e., the max. length)
-

LEARNABLE POSITIONAL EMBEDDING

- For a 2D learnable embedding (e.g., for images) each dimension is modeled by a different Embedding.
 - It might be implemented again as `torch.nn.Parameter(max_pos, model_dim / 2)` or `torch.nn.Embedding(max_pos, model_dim / 2)`.
 - Then, we form the positional embedding as concatenation of the two.
 - Note that the `model_dim` must be divisible by 2.
-

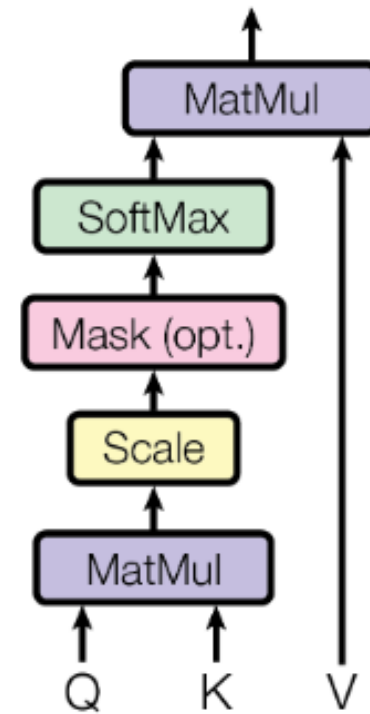
MULTI-HEAD ATTENTION

Multi-Head Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

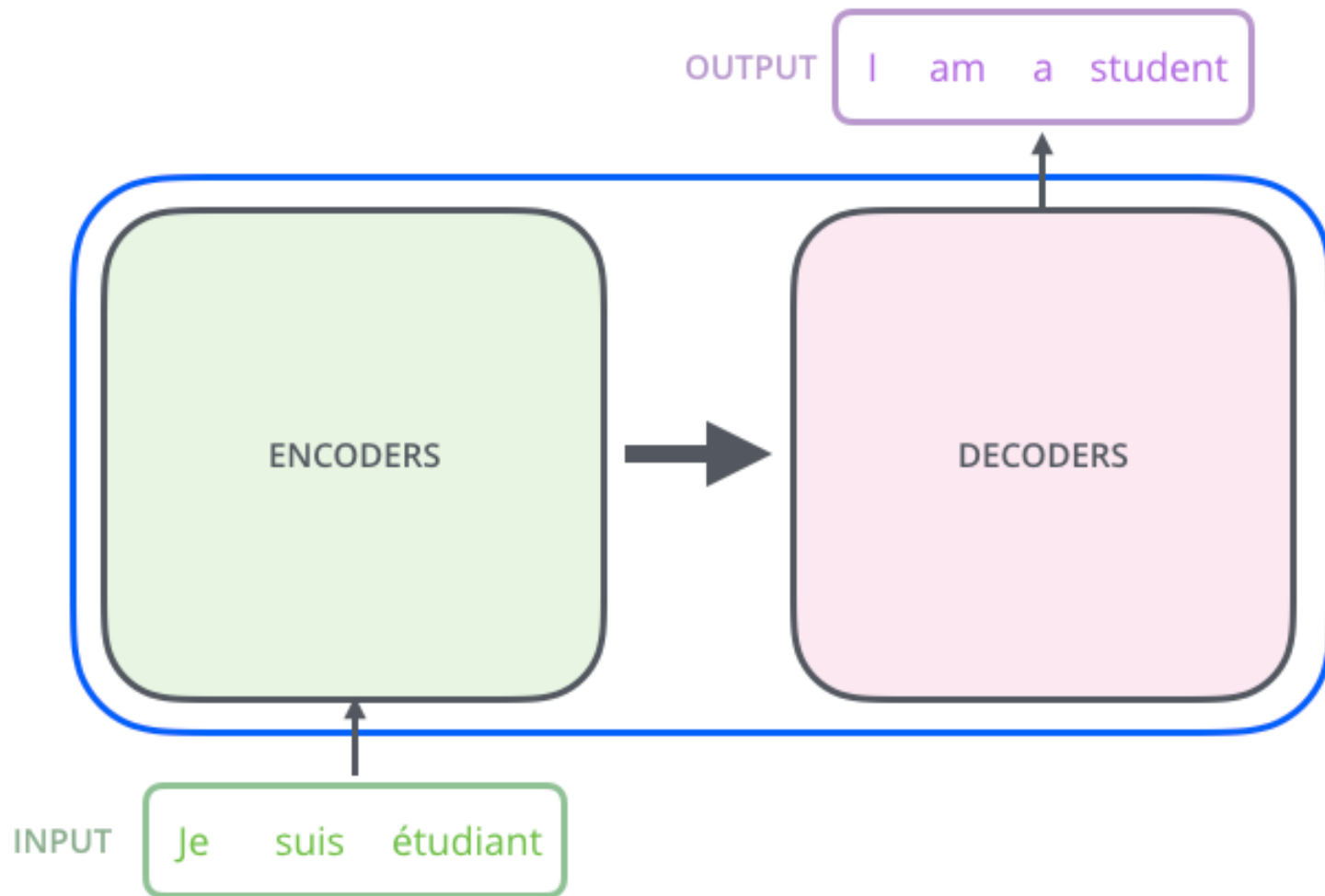
Scaled Dot-Product Attention



INFORMATION FLOW

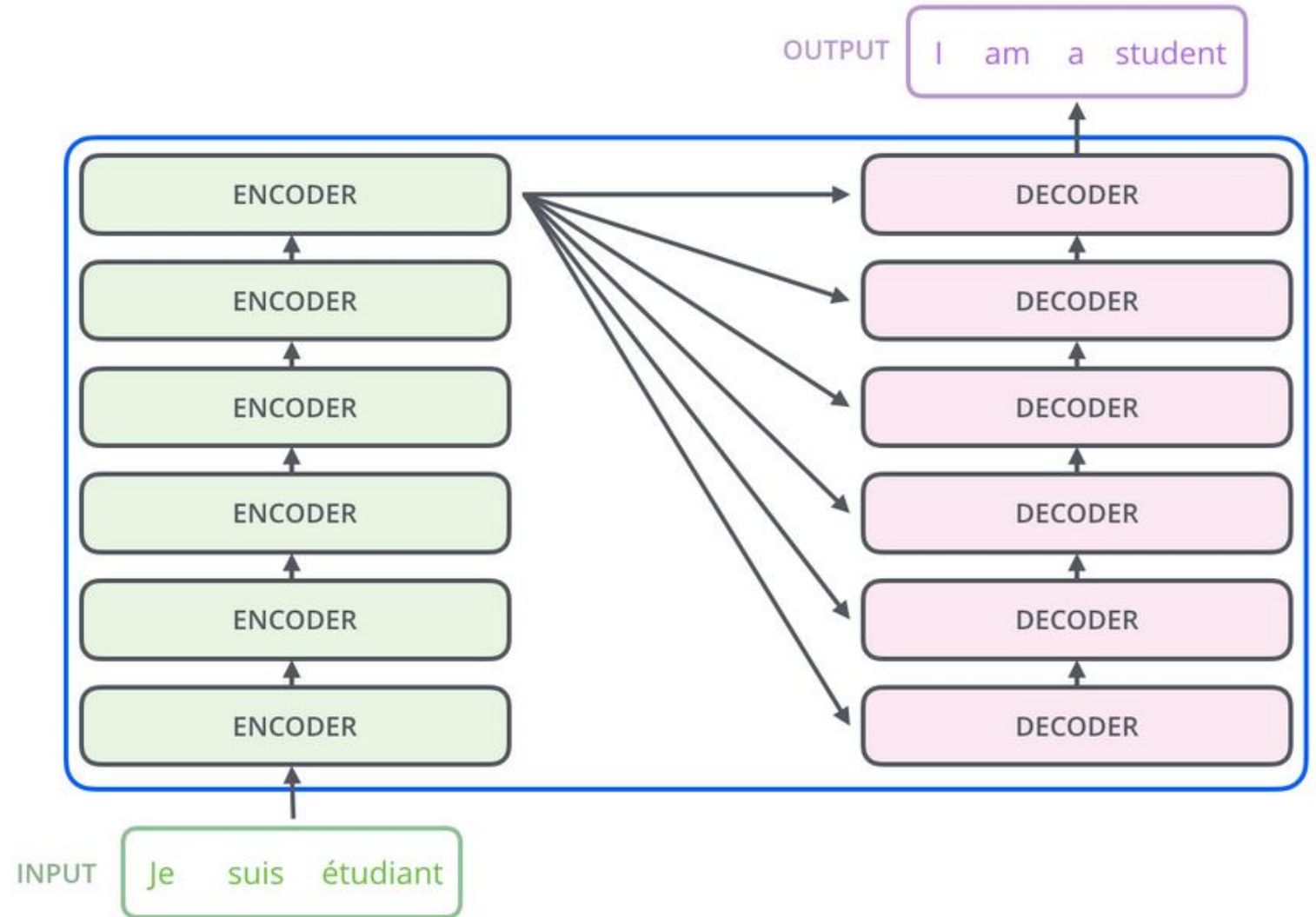
- Source:
- <http://jalammar.github.io/illustrated-transformer/>

-
- High abstraction
 - The Transformer consists of an **Encoder** and a **Decoder**.
 - Encoder "encodes" input sequence into a latent/semantic/abstract representation.
 - Decoder "decodes" this representation into the target sequence.



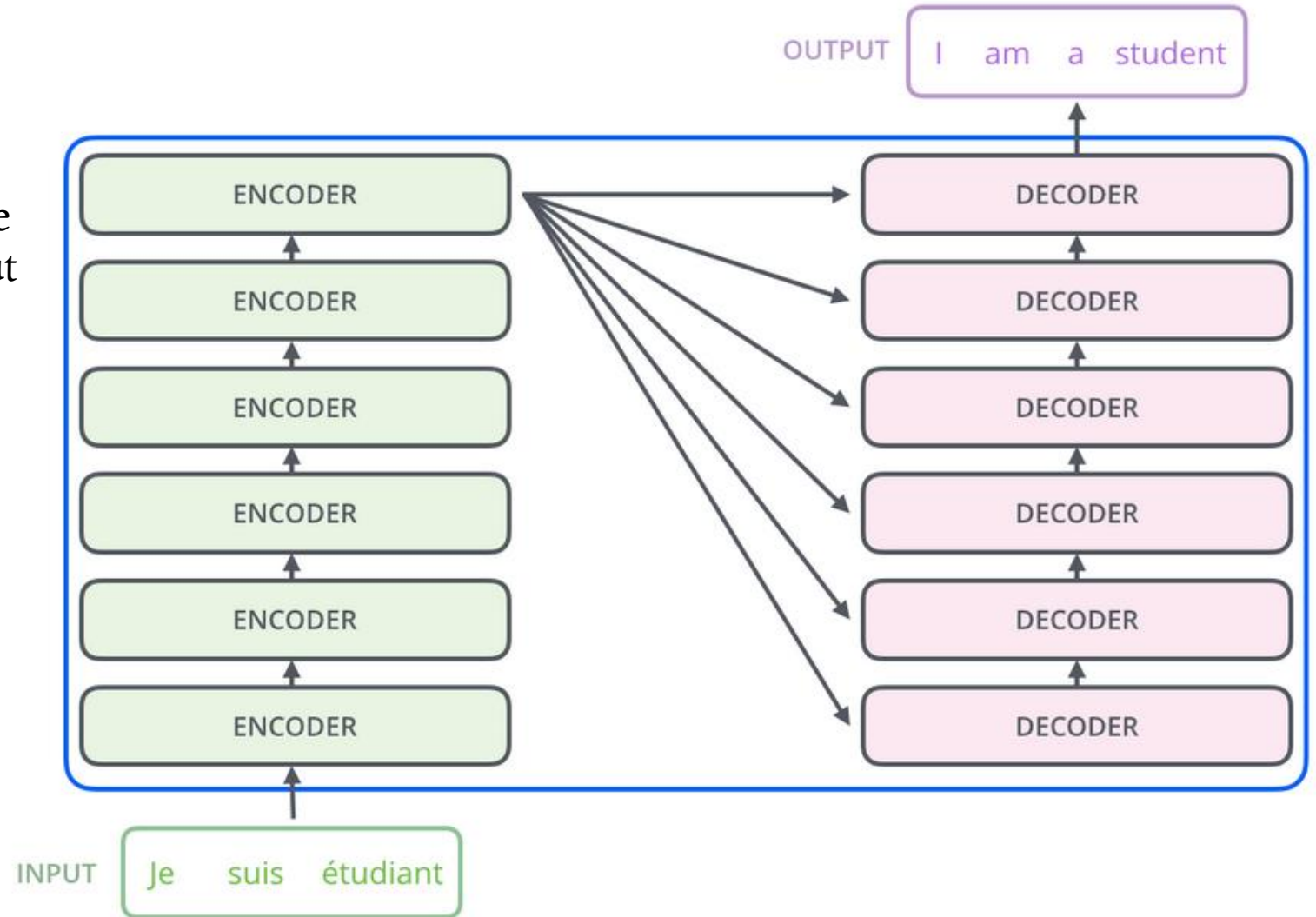
- High abstraction

- Both the encoder and decoder are "stacked".
- The abstract representation of the input sequence is produced at the last "layer" of the encoder.
- It is inputted into each layer of the decoder.
- Decoder outputs the sequence token by token (auto-regressive), or all tokens are outputted at once (non-auto-regressive).

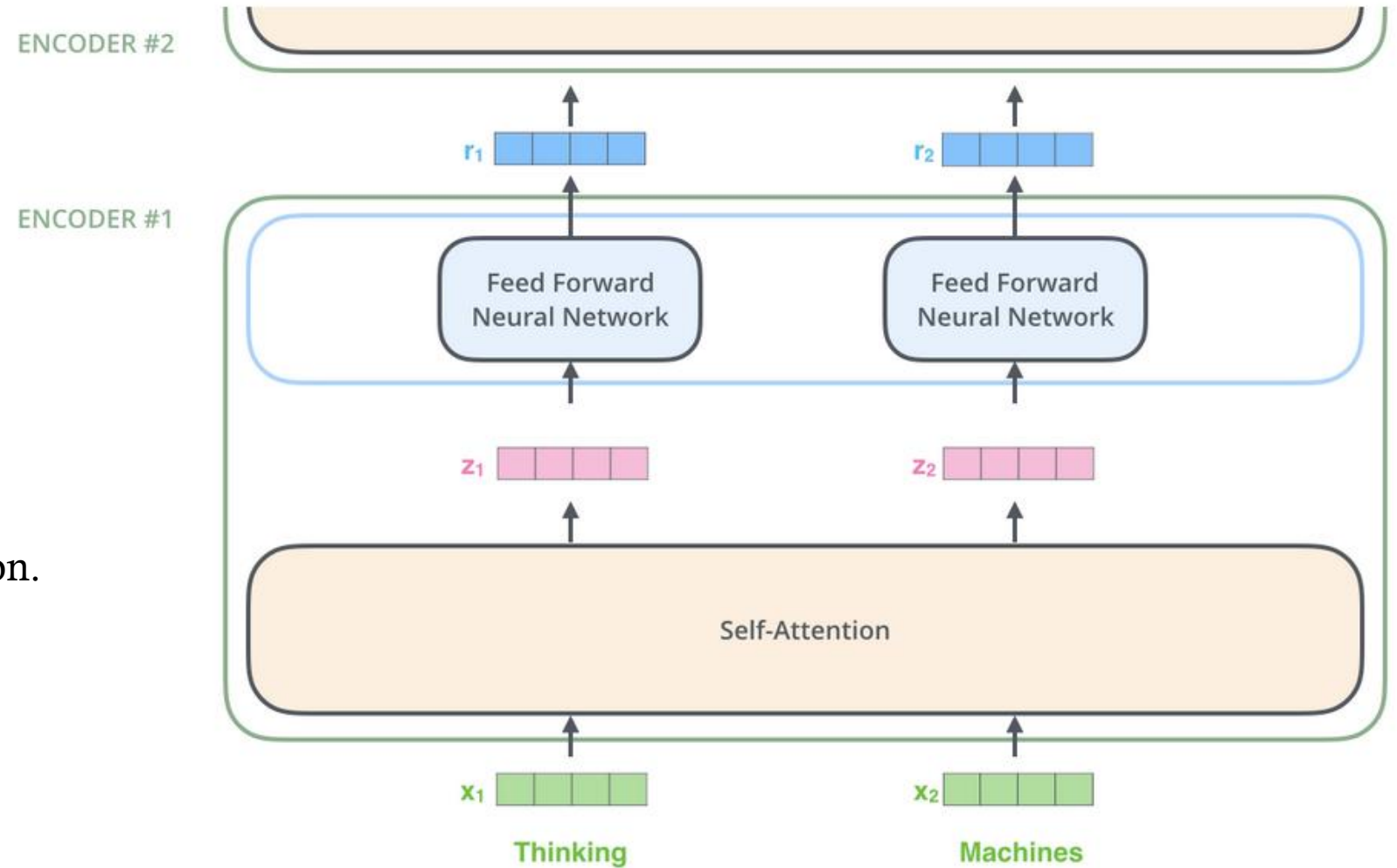


- Types of decoder

- **Auto-regressive** decoding is suitable for tasks where the size of the output is not known.
- Example is translation.
- **Non-auto-regressive** decoding is suitable for tasks where the number of outputs is known beforehand.
- Example is Pose Estimation.

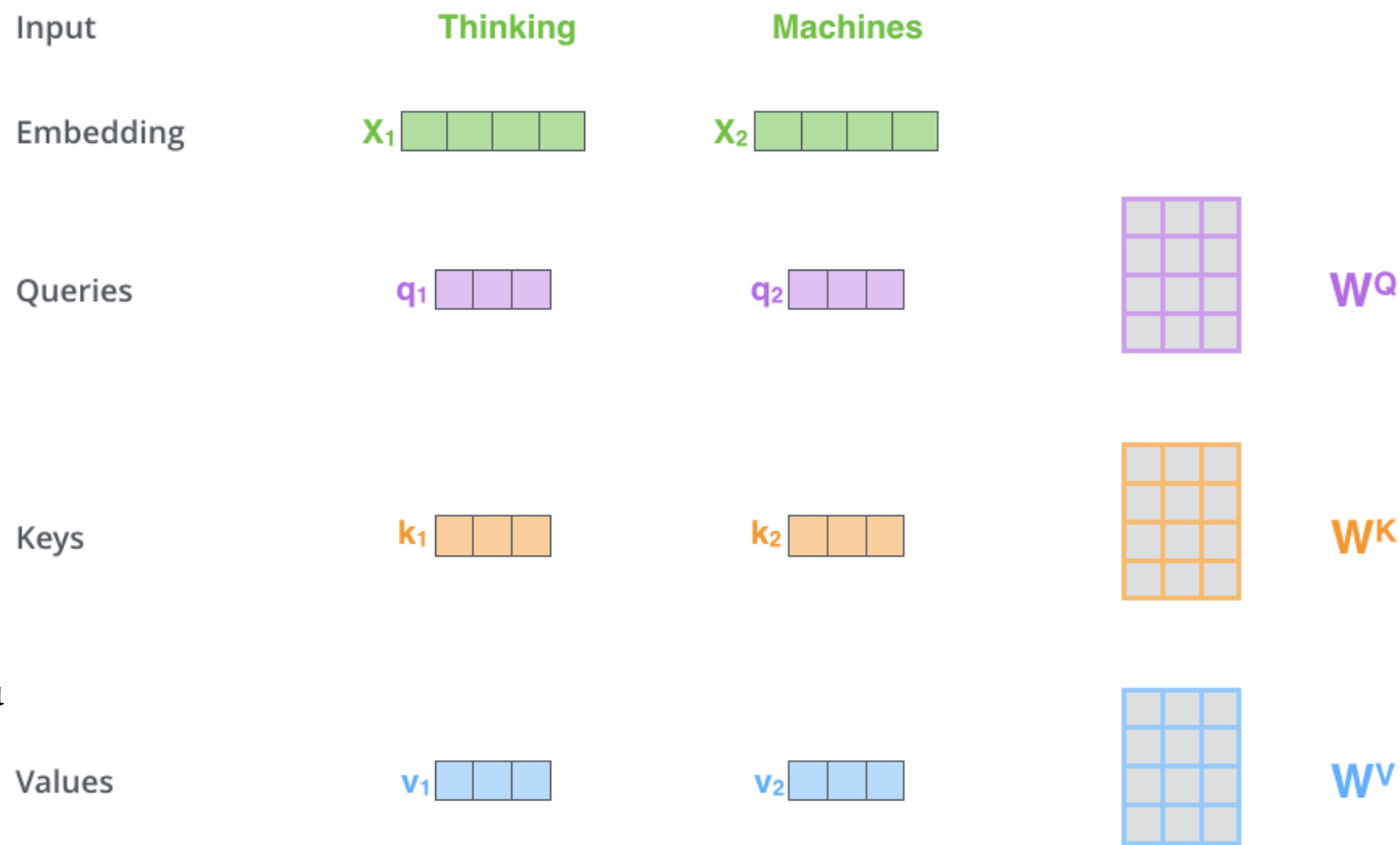


- The encoder must process the whole input sequence before the decoder begins to decode.
- The **Self-Attention** module produces an attentioned representation of the inputs.
- A 2-layered FFN transforms the representation further.
- The FFN is upscaling the representation and then downscaling to original dimension.
- i.e., $512 \rightarrow 2048 \rightarrow 512$



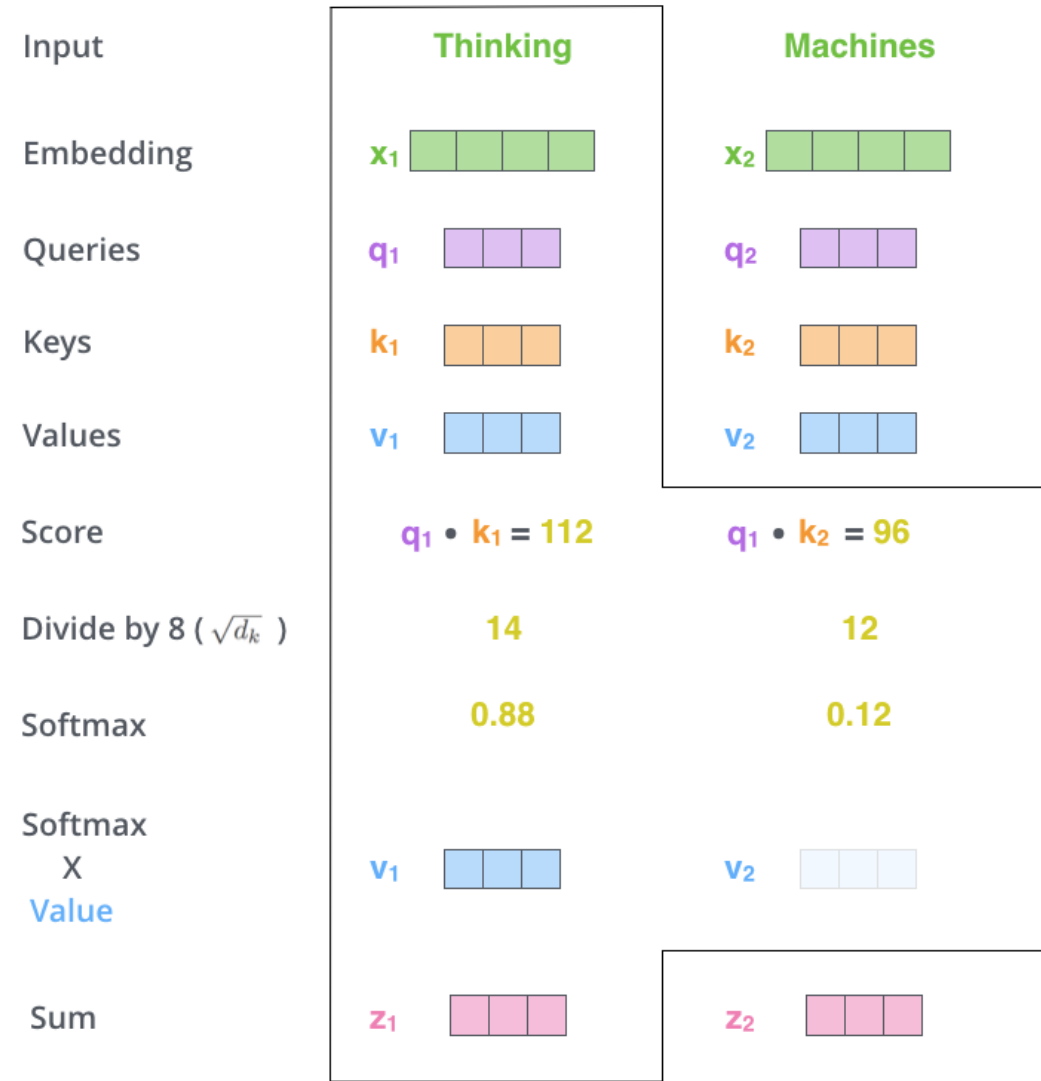
• Self-Attention in detail

- The input embedding is transformed into **Key**, **Value**, and **Query** vectors.
- W^* are the *Linear blocks* in the self-attention module.
- They define the dimensionality of the latent representation of the Transformer.
- **Value** represents the input. It is changed based on the attention through the layers.
- **Key** represents an offer of the current value.
- **Query** represents a question.
- **Value:** “Hey guys, I want to ask: **query?**”
- Other Values: “I have this answer: **key**. Do you like it?”
- **Value:** “Your answer is really answering my query. I will pay great attention to you.” or “Your answer really sucks! I will not pay attention to you!”



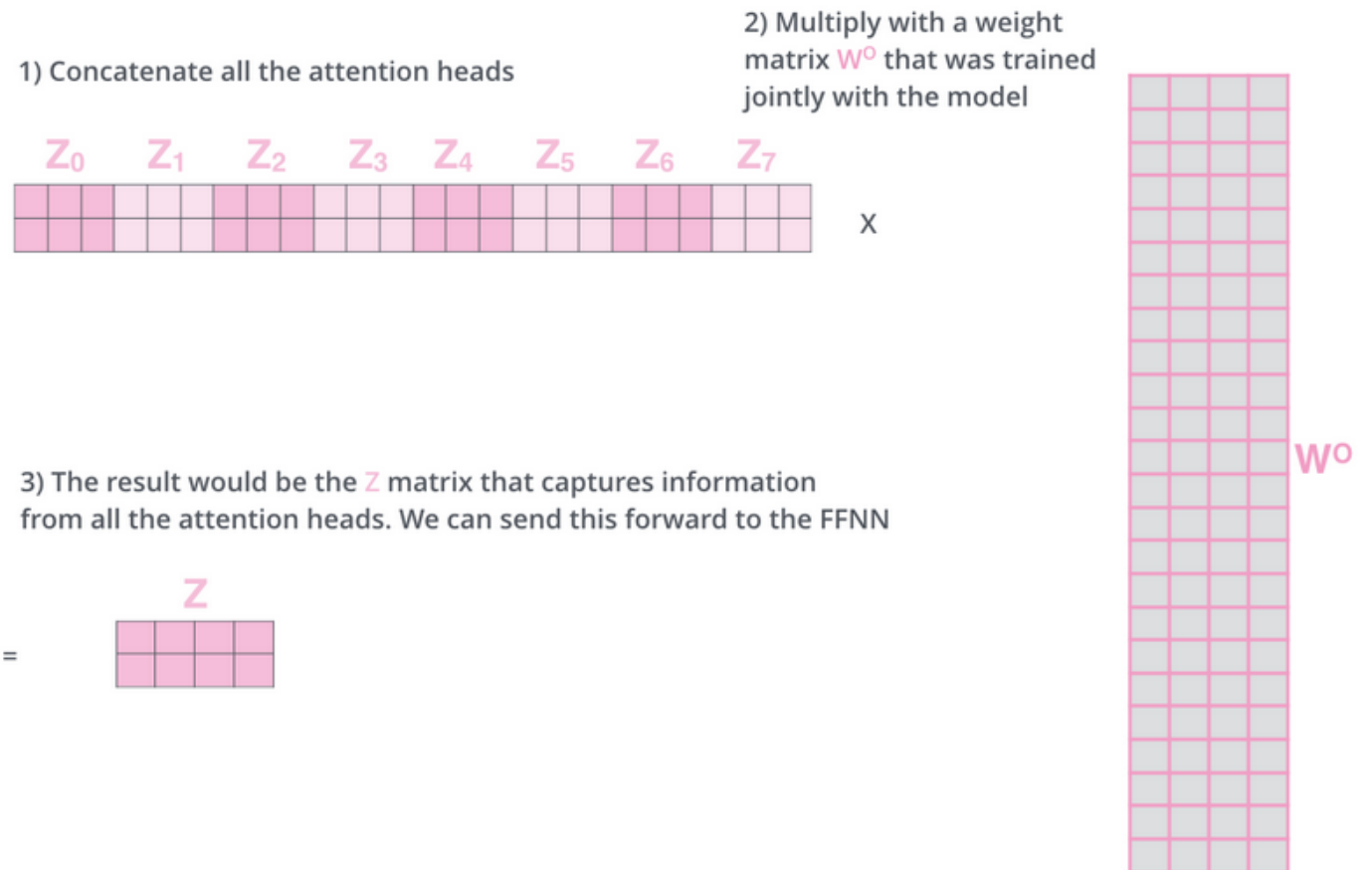
- **Self-Attention in detail**

- **Query** and **Key** produce the **attention**.
- The self-attention for the first word is computed as scalar products of the first query and all the keys (*Softmax*).
- Thus, the self-attention for the 1st word is a vector with dim equal to the number of words in the input sequence.
- The self-attended representation (z) of the word is the sum of all **value** vectors weighted by the attention vector (element-wise) (*softmax X value*).



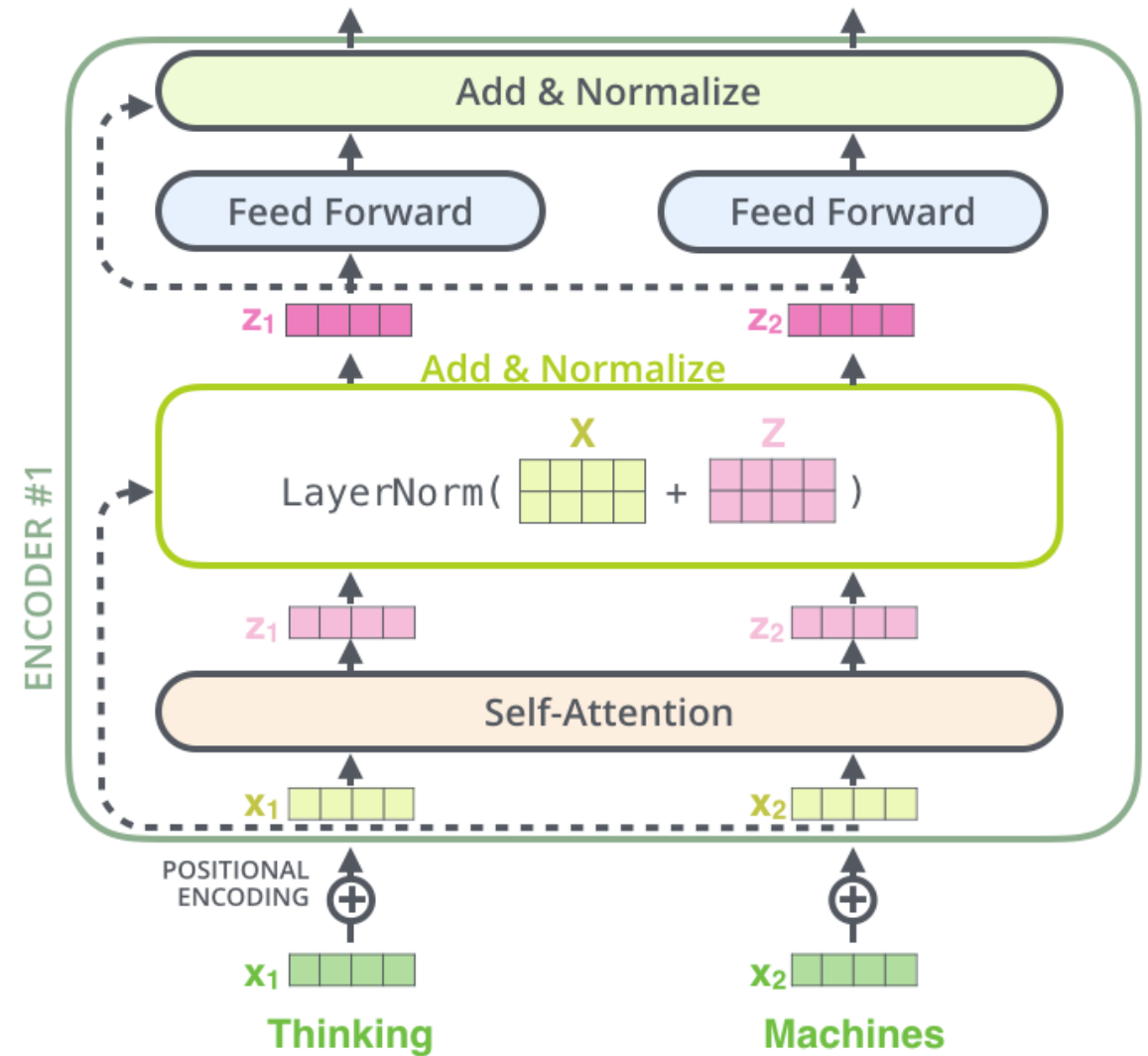
• Multi-Head attention

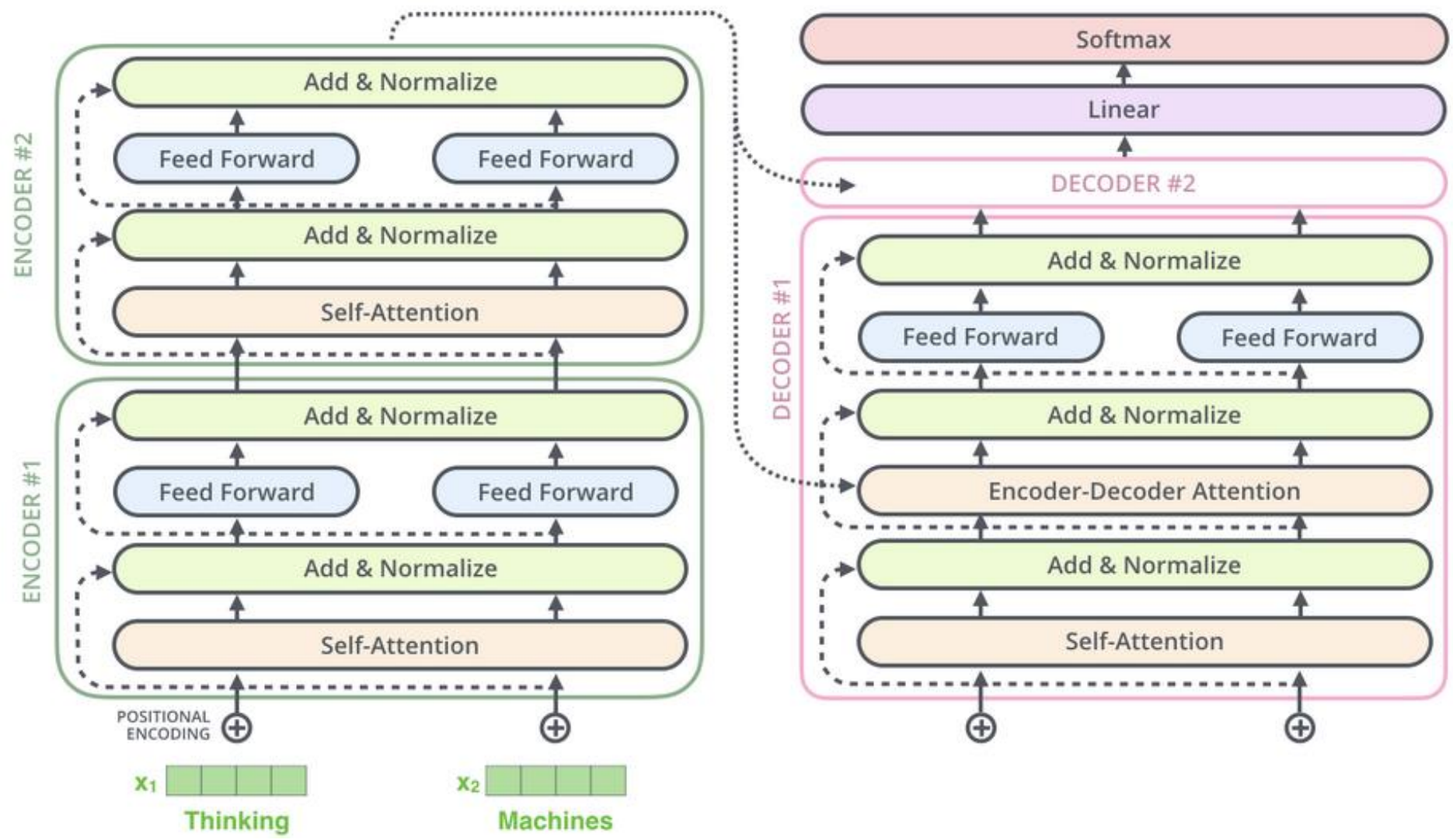
- Simply, there are more parallel *Scaled Dot-Product Attentions*.
- The outputs of them are concatenated
- The concatenated matrix is transformed by a Linear Layer to produce the final attentioned representation.
- Each parallel attention module can be viewed as an expert in different field.
- The transformation must be into a space of the embedding dimension.



- **Residual connection**

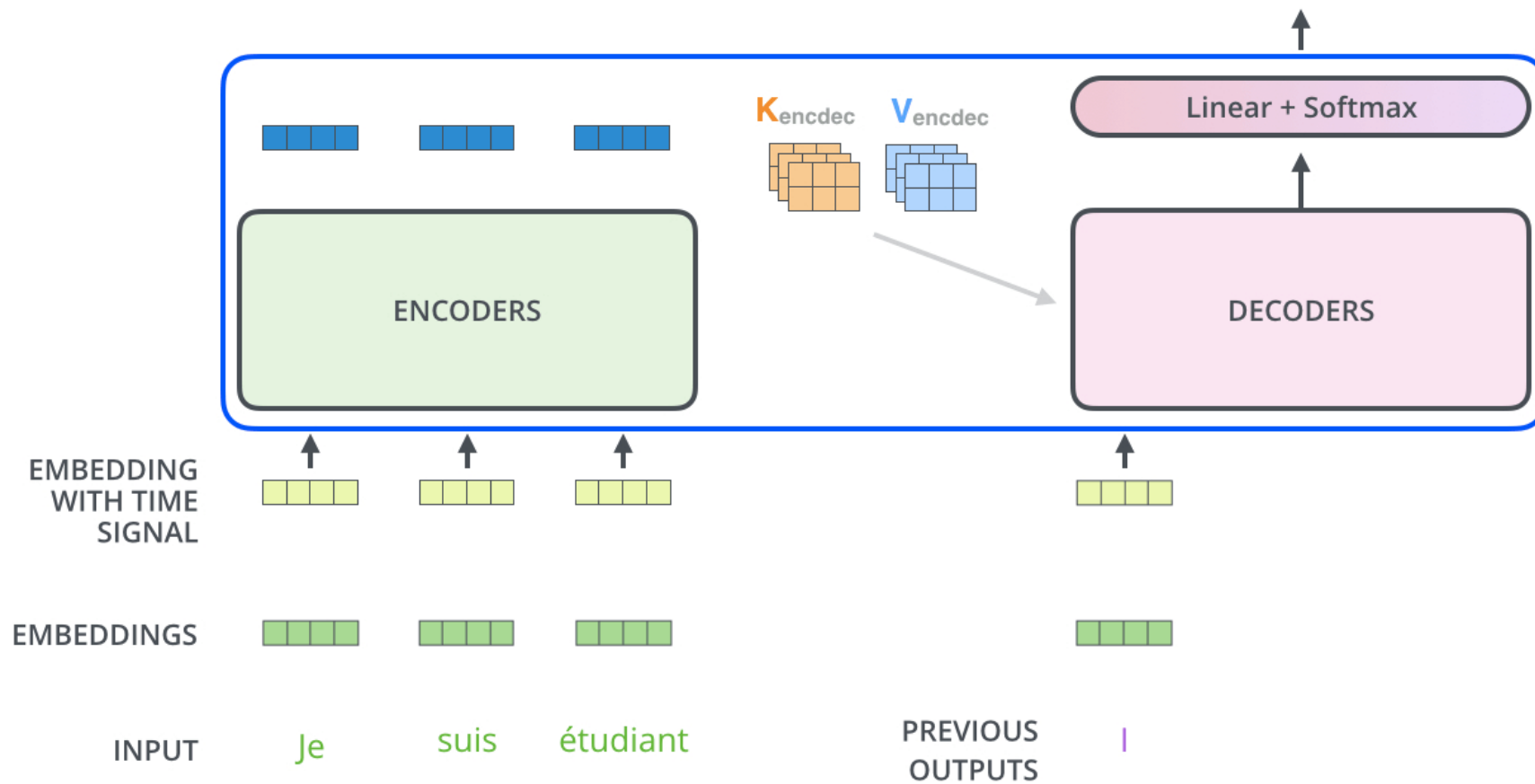
- In each encoder block, there are two residual connections.
- One is at the Self-Attention module.
- Second is at the FFN module.
- The residual connection helps to “transfer” the positional encoding through the encoder blocks.





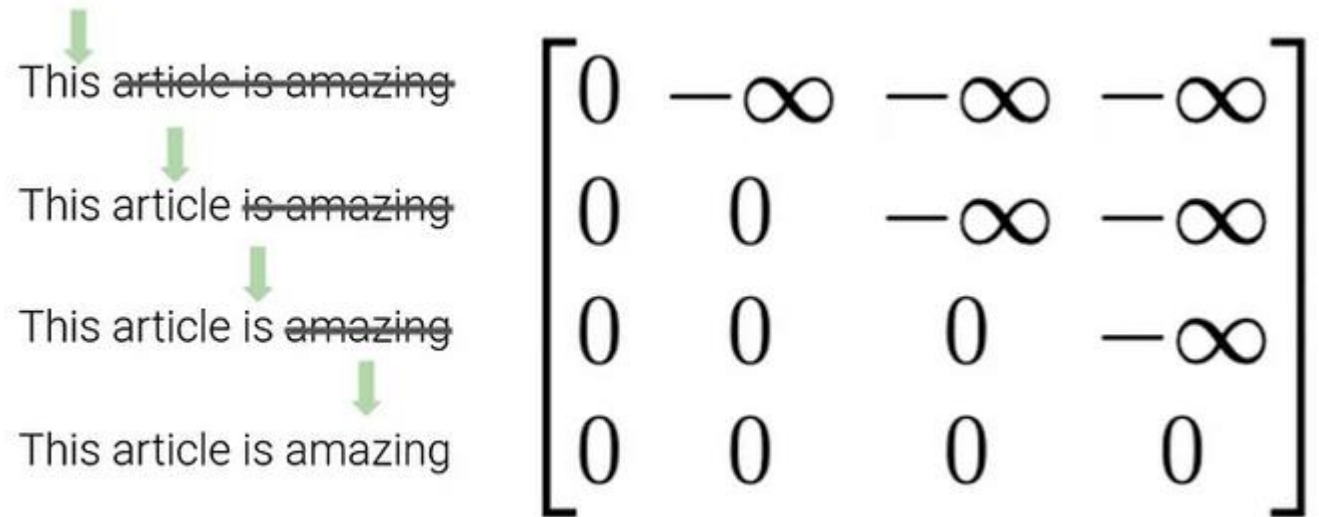
Decoding time step: 1 2 3 4 5 6

OUTPUT |



MASKING

- To be able to train auto-regressive models in batches, we need to introduce **masking**.
- Masking in the decoder is applied via a matrix multiplication.
- We define an upper triangular matrix with $-\infty$ as values (generally, we can use different masking).
- After the pre-softmax attention is computed, we multiply (element-wise) the resulting matrix with the mask matrix.
- Then, after softmax is applied, the attention becomes 0 at the masked positions





PADDING

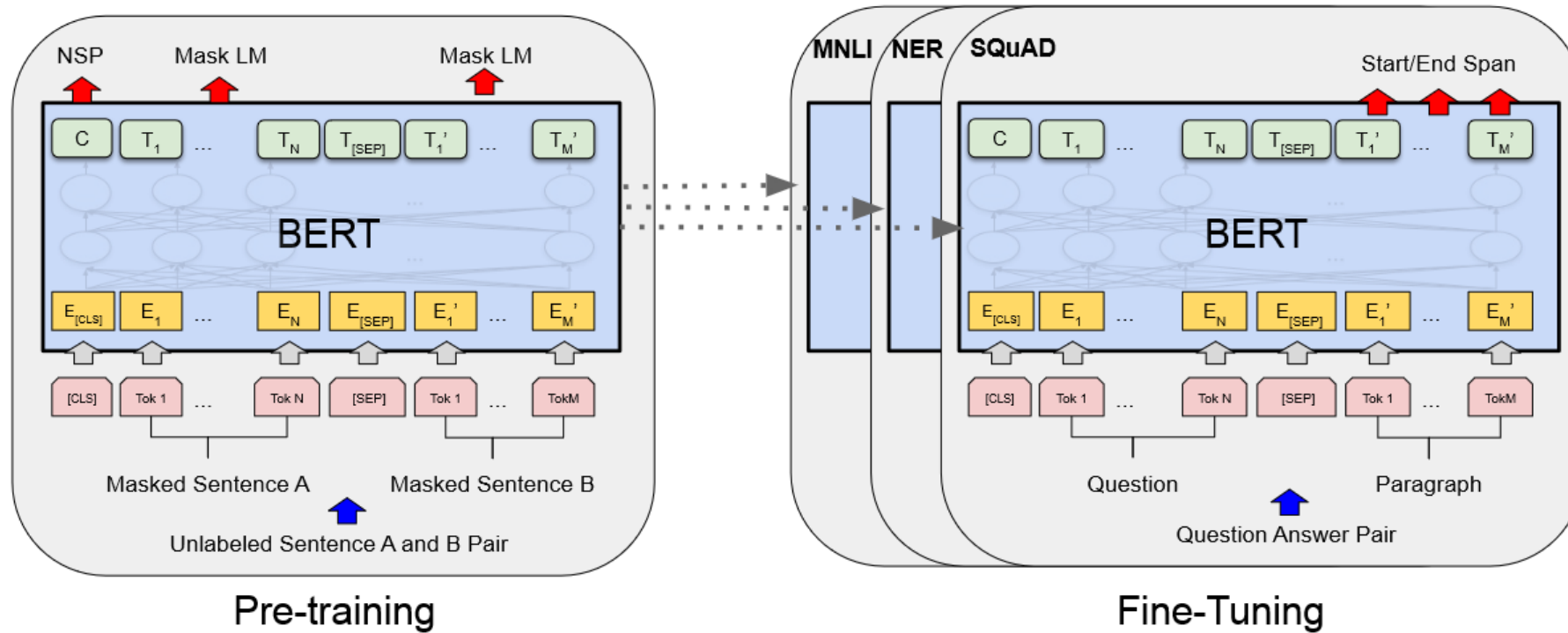
- To be able to train sequences of variable length (such as sentences), we need to introduce padding.
- The padding is a “filler” that makes all sentences “max length” long.
- PyTorch provides a parameter, where we can use bool values.
- WARNING! *False* means there is **no padding** and *True* means that there is **padding**.

1	0	1	0	4	4
1	1	1	1	1	0
0	0	0	4	4	4
1	0	1	0	1	4

<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>True</i>
<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>
<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>

 Sequence
 Padding

BERT: PRE-TRAINING OF DEEP BIDIRECTIONAL TRANSFORMERS FOR LANGUAGE UNDERSTANDING

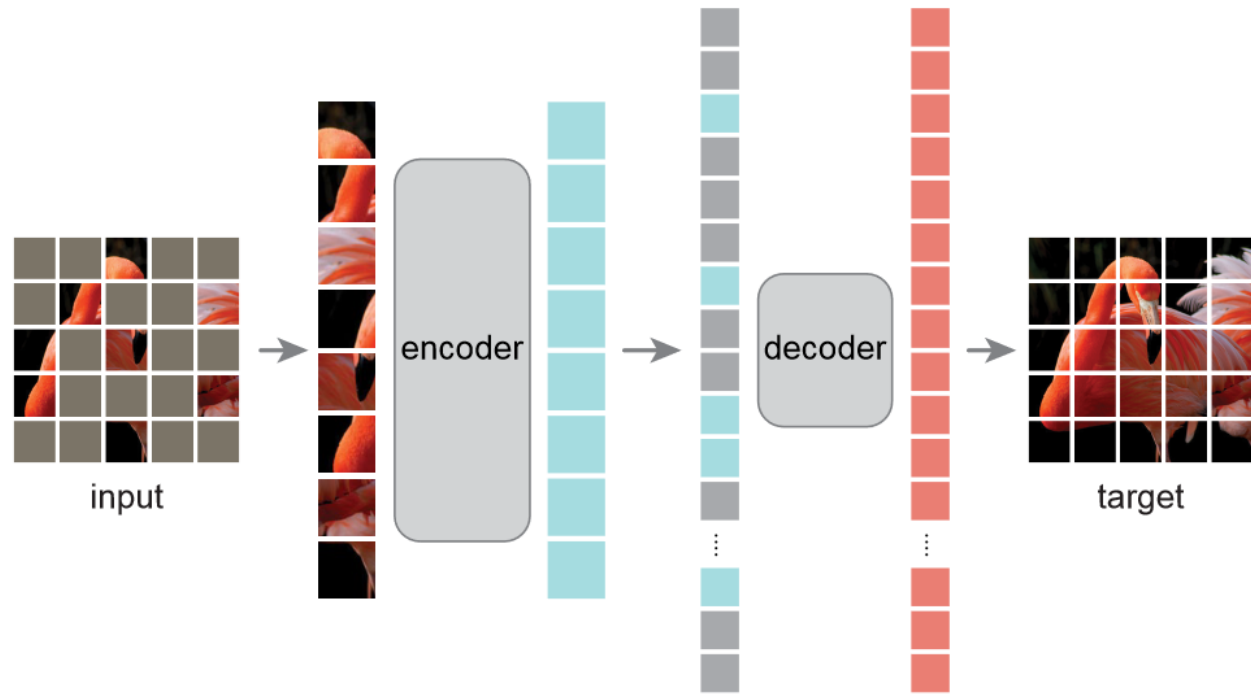


Pre-training

Fine-Tuning

SELF- SUPERVISION. MASKED AUTOENCODER

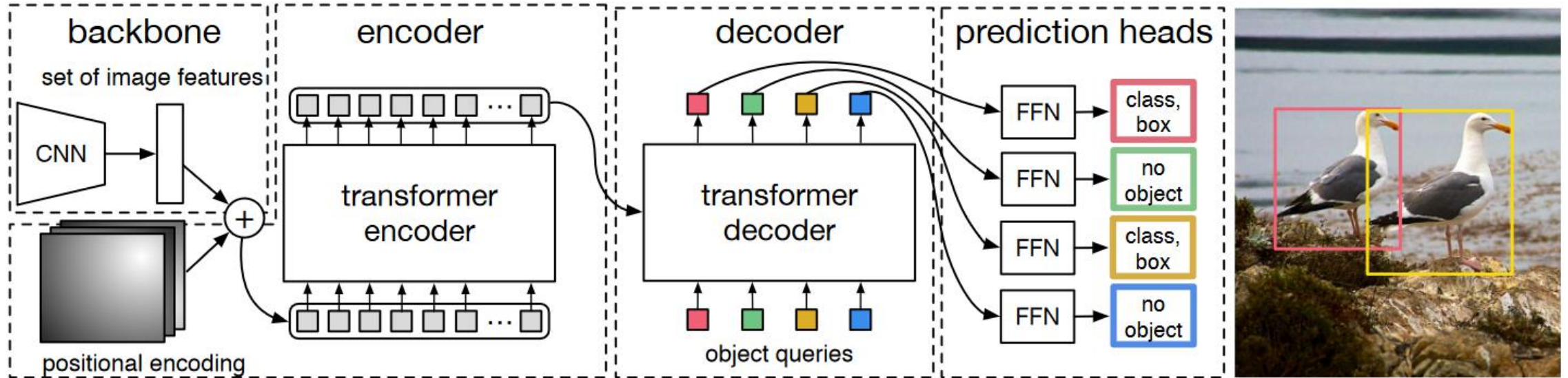
<https://arxiv.org/pdf/2111.06377.pdf>



- Visual self-supervision is inspired by its language counterpart.
- It involves masking parts of an image and learning how to reconstruct them.
- This leads to a strong pre-trained foundation model that “understands” the input modality.

DETR – DETECTION TRANSFORMER

<https://arxiv.org/pdf/2005.12872.pdf>



RESULTS

Model	GFLOPS/FPS	#params	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	47.8	27.2	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	44.9	64.7	47.7	23.7	49.5	62.3

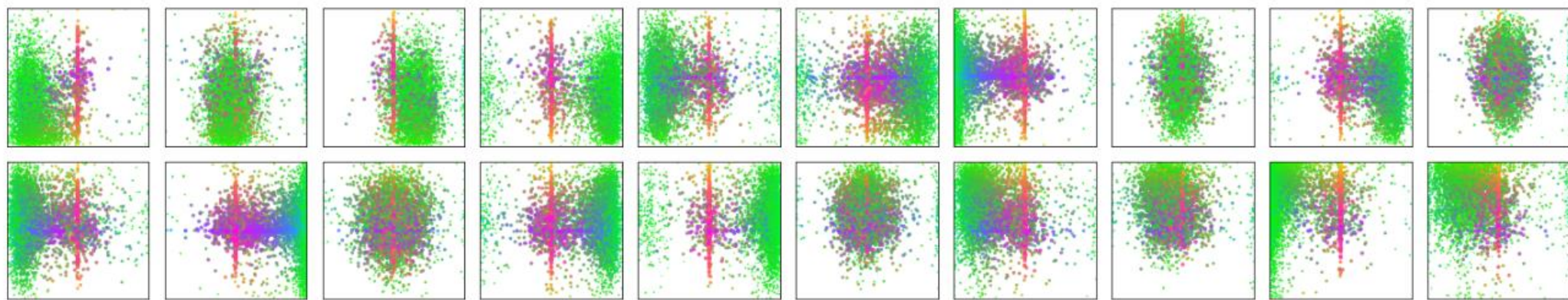
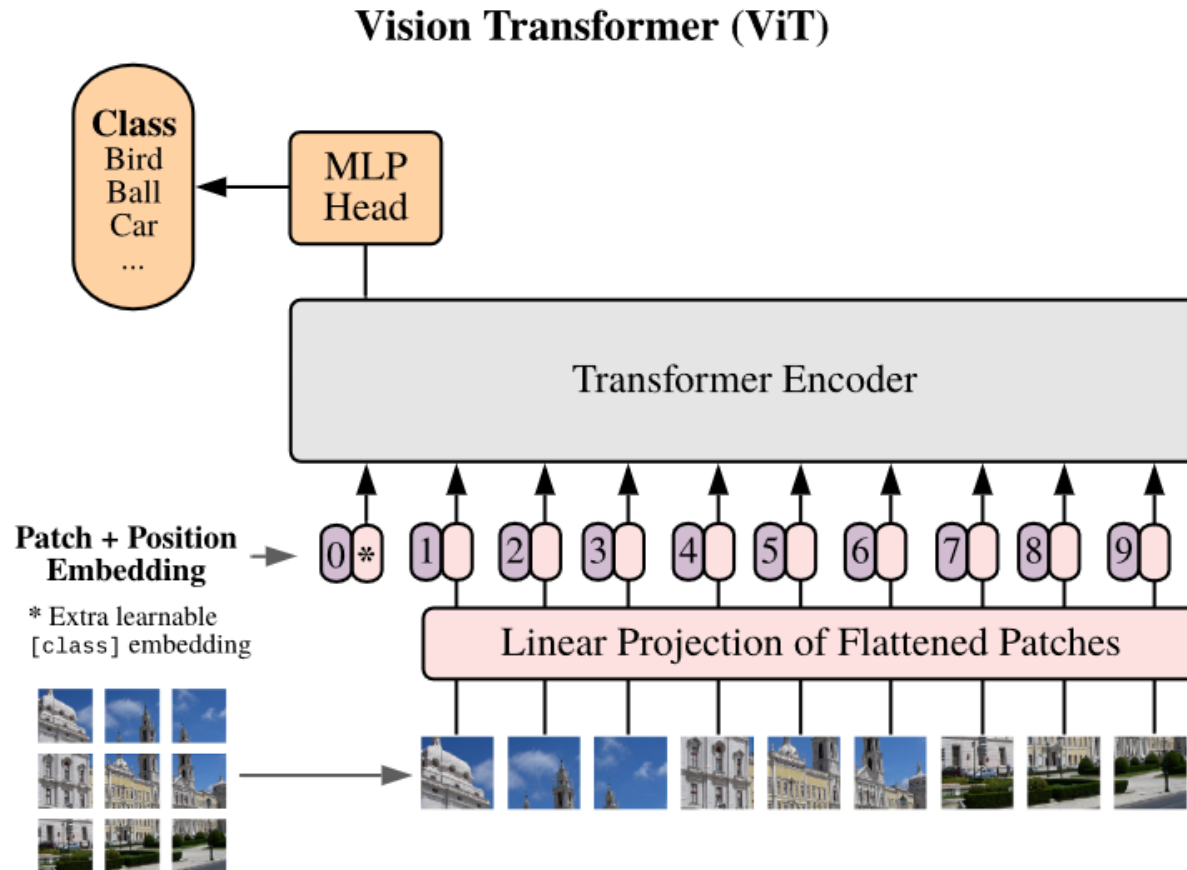


Fig. 7: Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total $N = 100$ prediction slots in DETR decoder. Each box prediction is represented as a point with the coordinates of its center in the 1-by-1 square normalized by each image size. The points are color-coded so that green color corresponds to small boxes, red to large horizontal boxes and blue to large vertical boxes. We observe that each slot learns to specialize on certain areas and box sizes with several operating modes. We note that almost all slots have a mode of predicting large image-wide boxes that are common in COCO dataset.

VISION TRANSFORMER

An Image is worth 16 x 16 words: <https://arxiv.org/pdf/2010.11929.pdf>



Transformer Encoder

