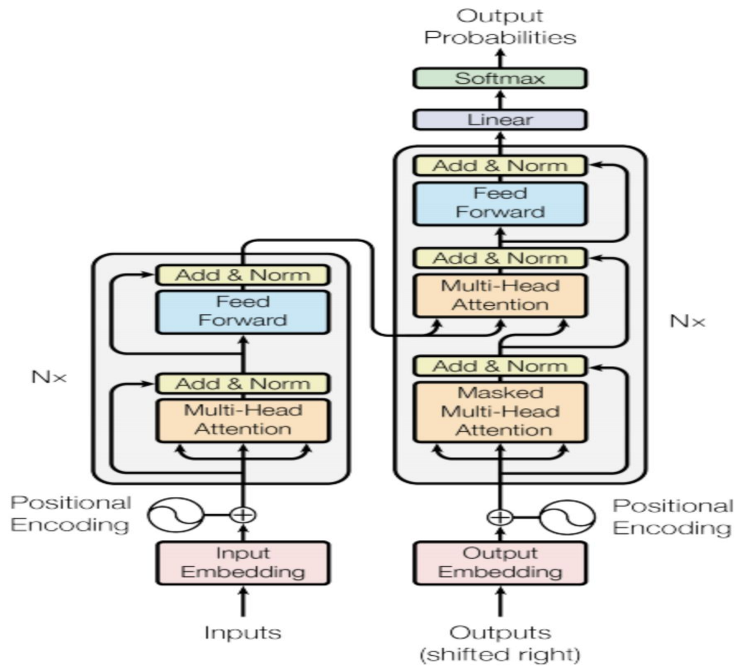
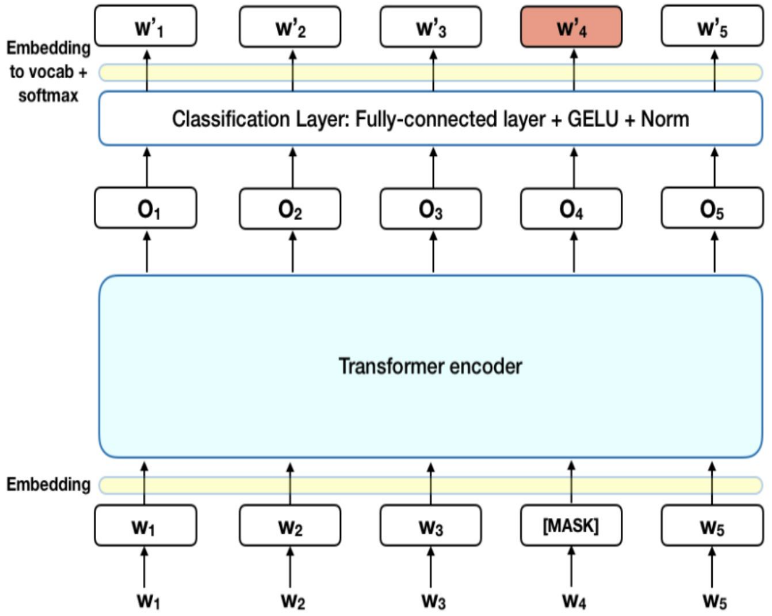



W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Intro to Transformers

Michael(Mike) Erlihson

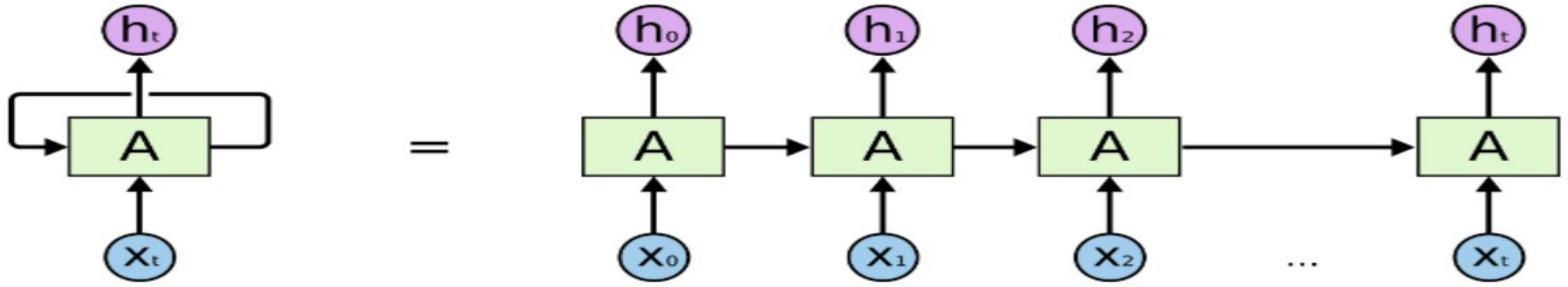
January, 2021

Agenda

- RNNs Recap
- Transformers: General Intuition
- Transformer: High-level Architecture
- Attention: Intuition and How it works
- Positional Embedding
- Encoder and Decoder Architecture
- Transformer Usage Modes

Pre-Transformer Era: RNNs

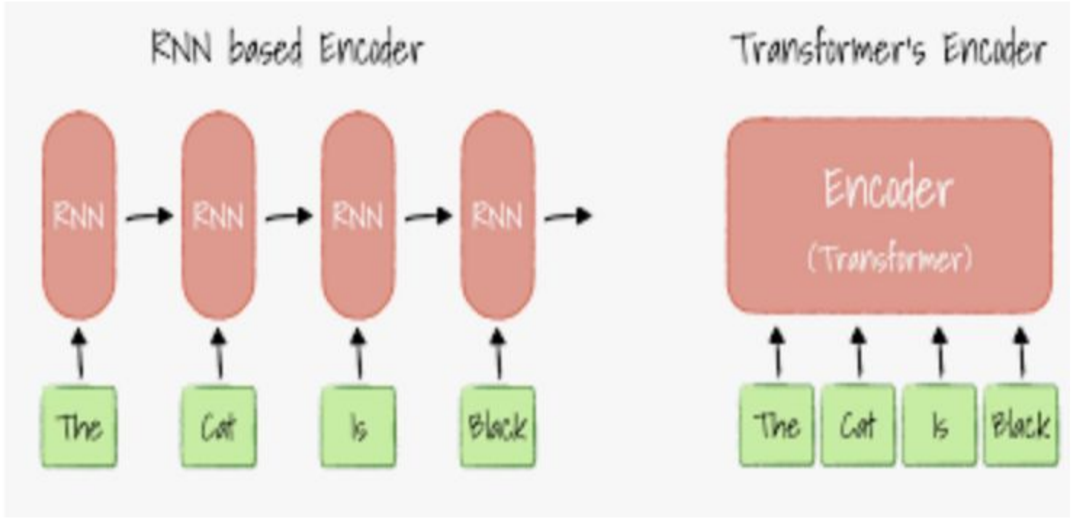
- Recurrent Neural Networks (RNNs) are designated to handle sequential data
- Predict next word in a sentence, next vowel in speech, next frame in video
- Perform the same task for every sequence element, with the output being depended on the previous (hidden) outputs
- Have a “memory” capturing information about what has been calculated so far

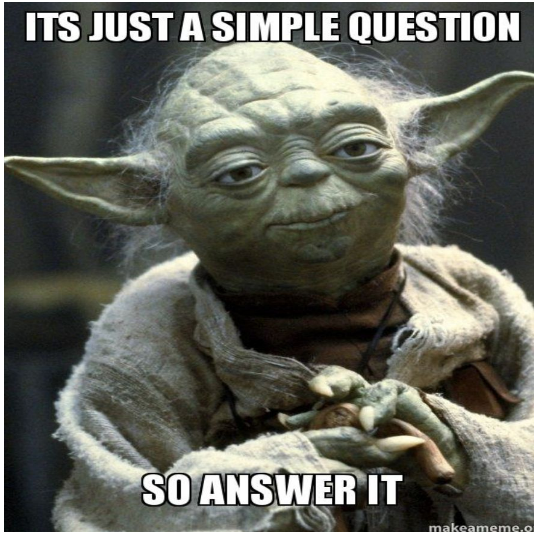


Transformers Revolution: One Simple Question

Why don’t we feed the entire input sequence?

No dependencies between hidden states! That might be cool!





But there is a problem here

Instead of a sequence of elements, we have a set now (**order is irrelevant - !!!**)

"Hello I love you" ==

"Hello", "I", "love", "you"

"love", "Hello", "I", "you"

"I", "love", "Hello", "you"

....

But the texts have an order!!
The Transformer needs to know it to "understand" the text meaning

Solution: Positional Embedding

- **Positional encoding**: set of small values, added to the word embedding vector before the first Transformer layer
- **Positional encoding**: The same word representation is dependent on its position in the input sentence

LET'S GET OUR HANDS DIRTY

Transformer: A New Emperor of the NLP World

- NLP models until 2018 used different flavours of RNN for most types of tasks

- But then… came “**Attention is all you need”** and “**BERT: …**” papers were published and took the NLP world by storm!

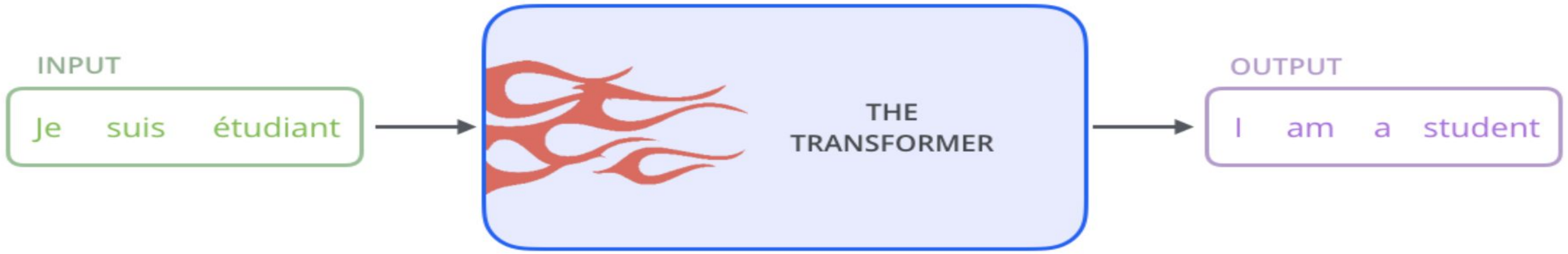
- They introduced new network architecture called **Transformer** relying on attention mechanisms

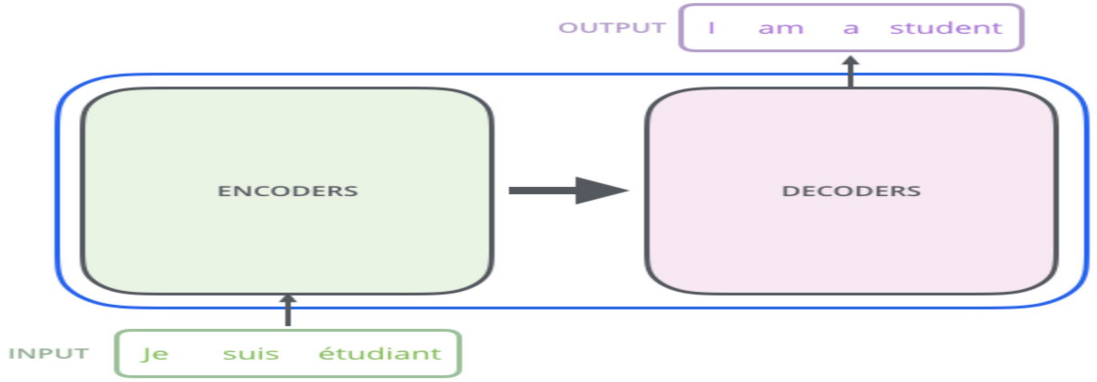
- Since then **ALMOST ALL** NLP models are based on Transformers/BERT (have BERT/Former inside the name RoBERT, AlBERT, LinFormer, Reformer, PerFormer etc)

Transformers: High-Level Look

Sentence in Language A

Sentence in Language B

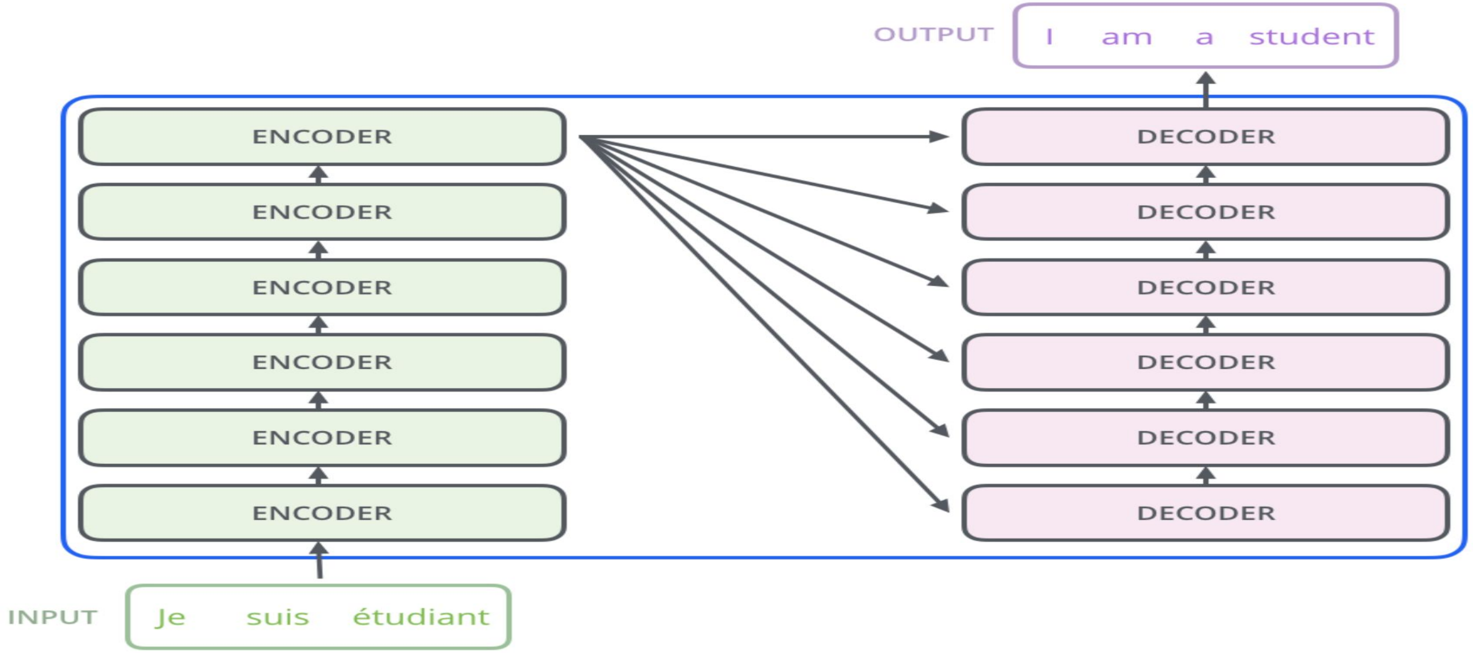




Encoders have the same architecture

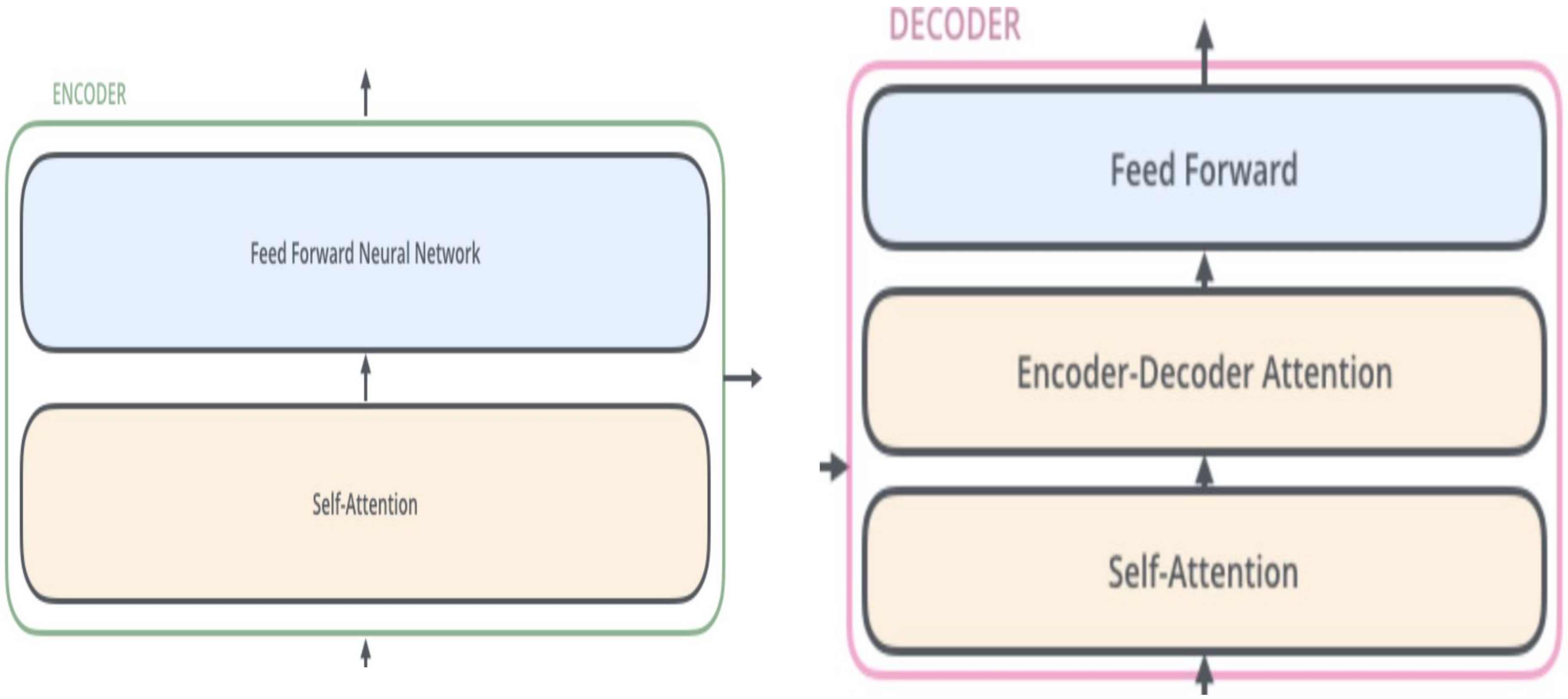
Decoders have the same architecture

Encoder-Decoder: High-Level Look

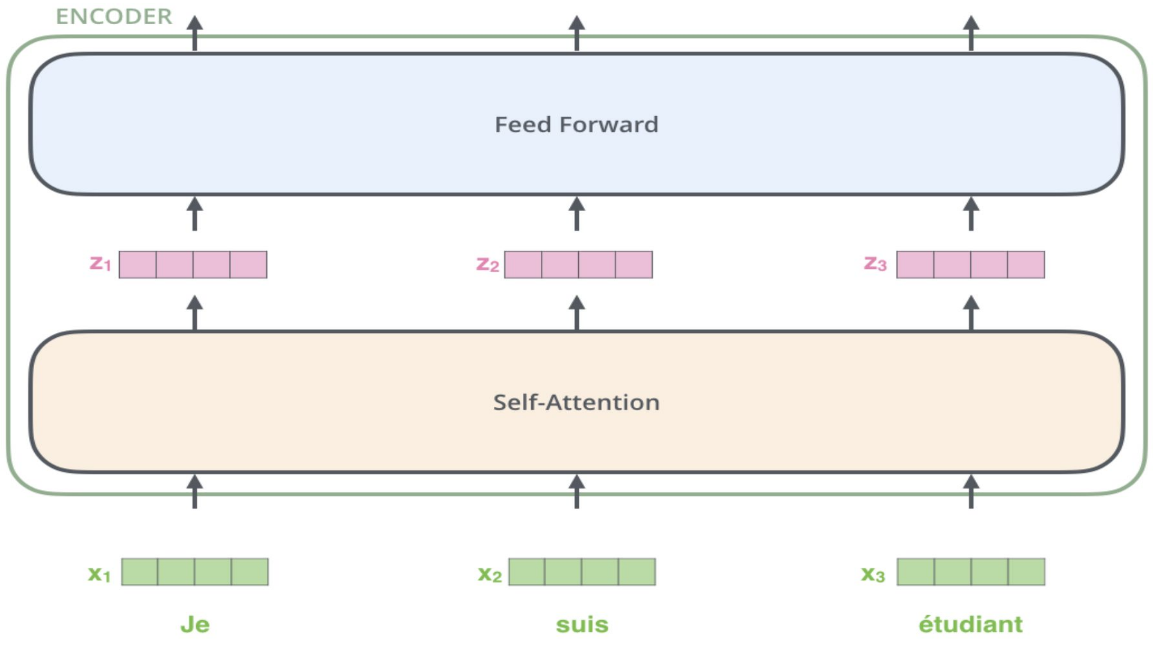


- **All encoders have the same architecture**
- **All decoders have the same architecture**

Encoder & Decoder: Main Blocks



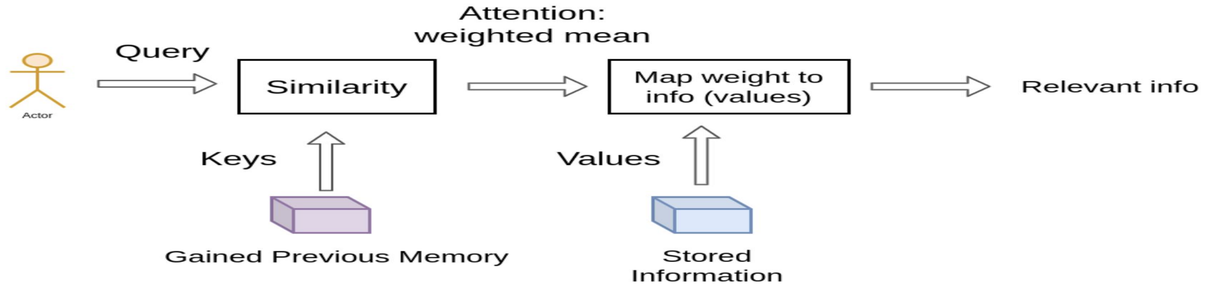
How Encoder Works: Self-Attention + Standard Layers



Wait, but what is this SELF-ATTENTION?

Attention Mechanism: An origin

- Key-value-query concepts originate from information retrieval systems
- **Video search (query)**: content/feature-based lookup
  - Search engine maps the query against a set of keys (video title, description, tags etc.) associated with different stored videos.
  - Algorithm present the best-matched videos (values)
  - **Attention** is the choice of the video with a maximum relevance score



Attention Mechanism: Transformer

But what are the “query”, “key”, and “value” vectors now?

$$\downarrow$$

Abstractions useful for computing/thinking about ATTENTION

But what does it really mean?

Transformer Attention: An Essence

Instead of choosing **where** to look according to the position in a sequence,

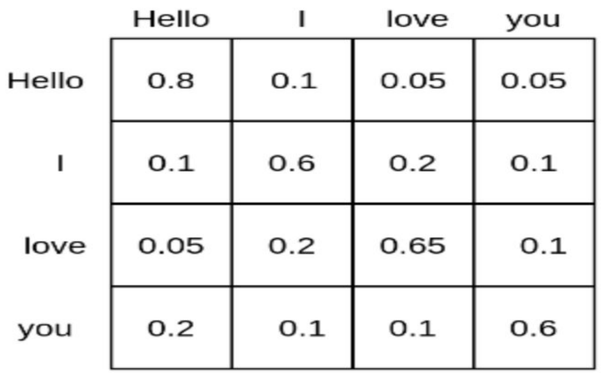
we now choose the **content that we wanna concentrate at**

- Split the data into key-value pairs (query preserves its original meaning)
- Keys define the attention weights to look at the data
- Values is the information that we will actually get
- Determine similarity metrics (cosine similarity = normalized inner product)

Self-Attention: Let's be More Specific

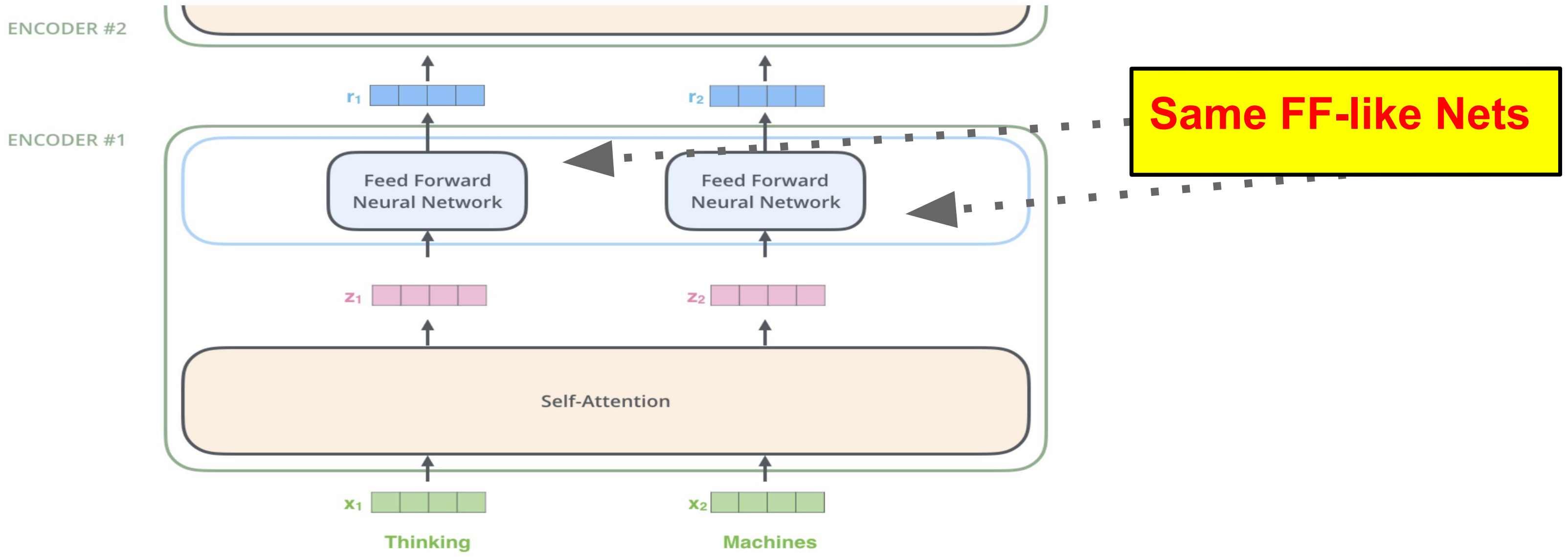
> “Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.” ~ Ashish Vaswani et al. [2] from Google Brain.

- Detects correlations between input words indicating the syntactic and contextual sentence structure
- These words share subject-verb-object relationship - that's what self-attention captures



	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

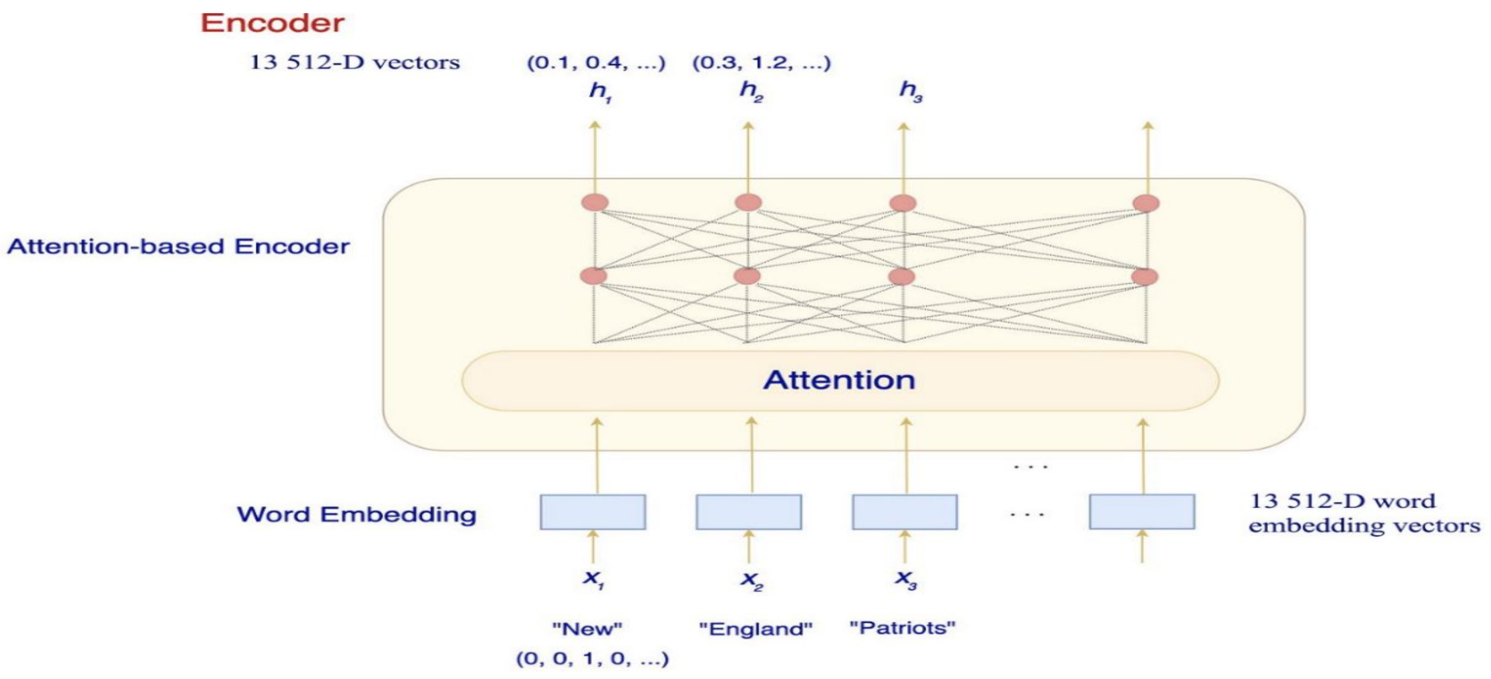
Encoder: Self-Attention + Standard Layers



Self-Attention: word flows through its **own path in the encoder**; dependencies between these paths

FF layer: various paths can be executed **in parallel**

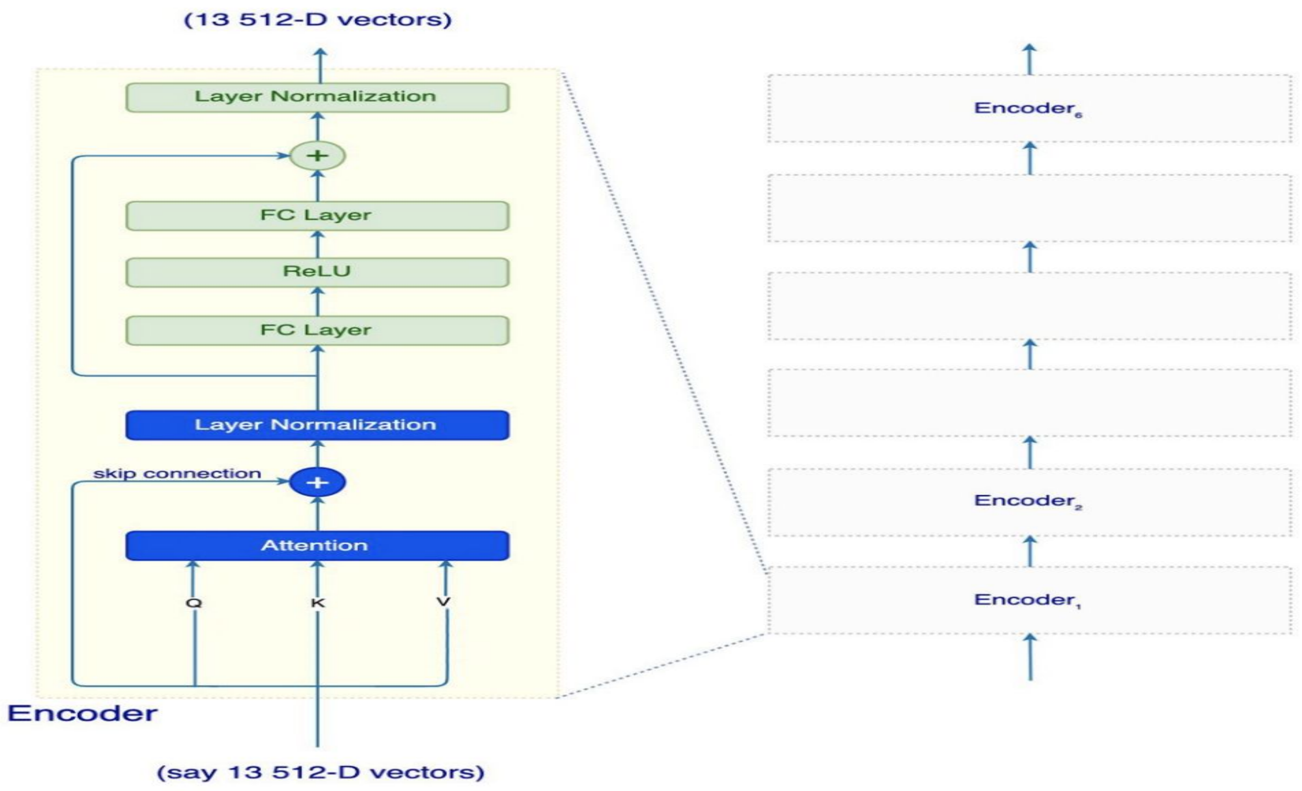
Encoder: Contextualized Token Embeddings



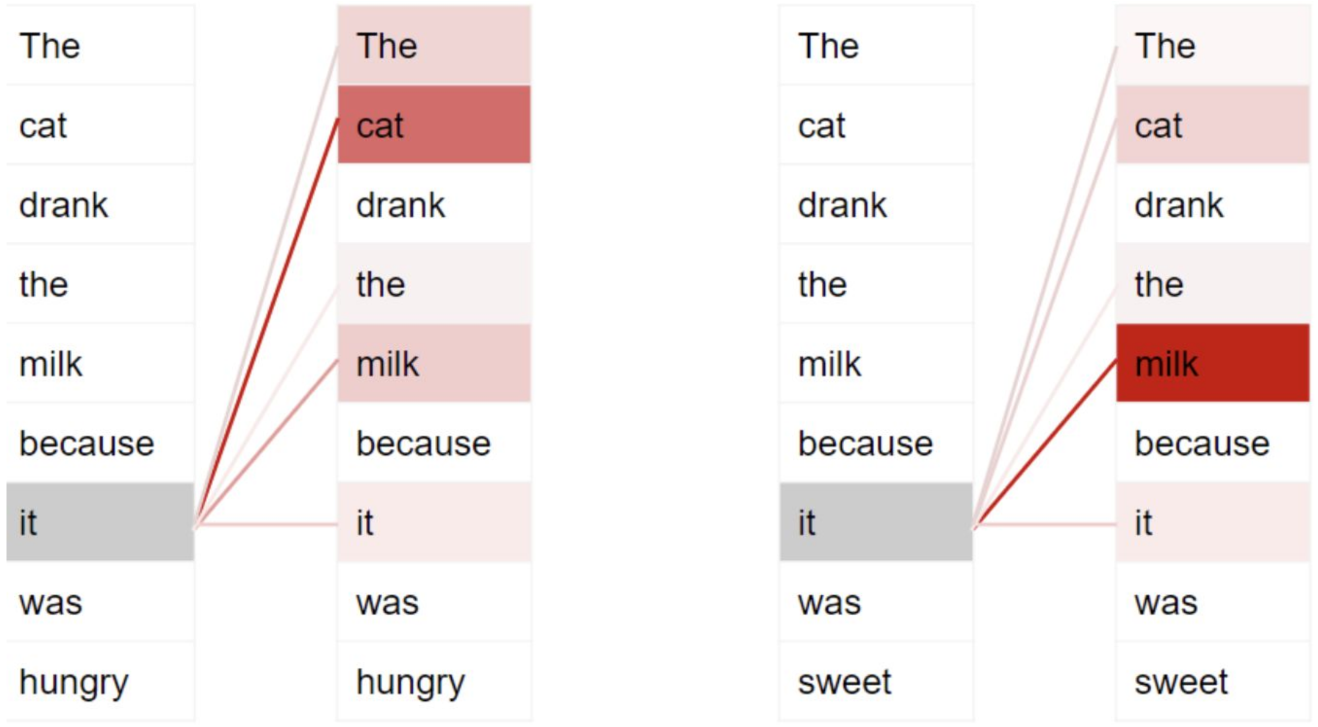
Input: Raw Text

Output: Contextualized tokens embeddings

Encoder: More Detailed Layer Scheme



Self-Attention: Reminder

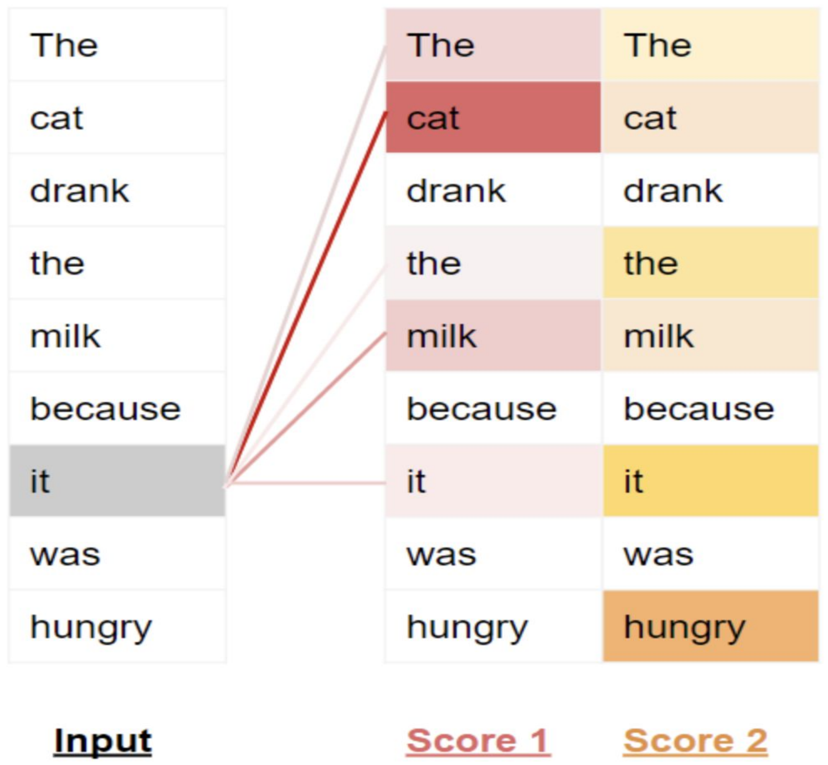


Dark colors represent higher attention (Image by Author)

Core Idea 1: Multiple Attention Scores

To enable handling more nuances about intent/semantics of the sentence, Transformers include multiple attention scores for each word

Multiple Attention Scores



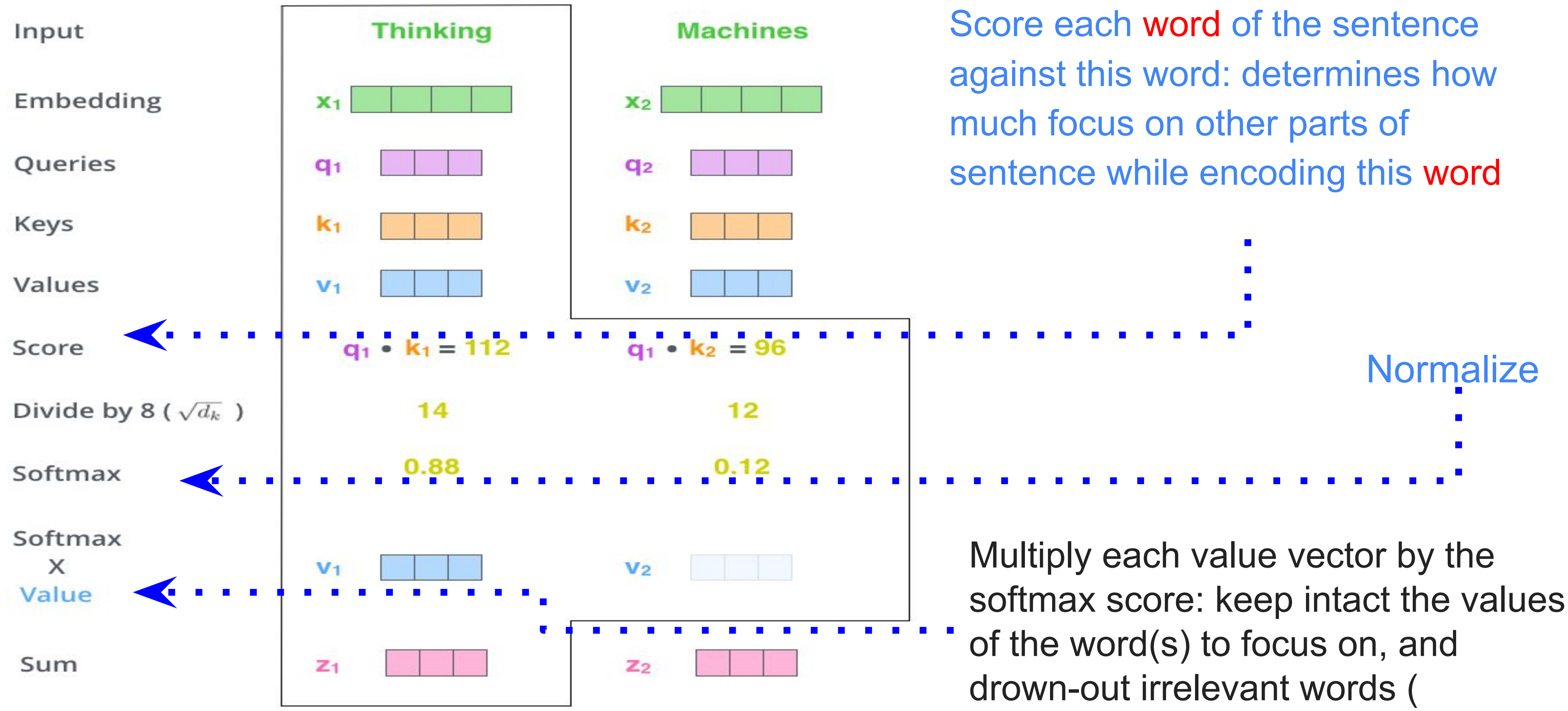
- ‘it’ word: the 1st score highlights ‘cat’, while the 2nd highlights ‘hungry’.
- Decoding ‘it’: (e.g. translation): incorporates some aspect of both ‘cat’ and ‘hungry’ into the translated word.

Contextualized Embeddings!!!

Self-Attention in Detail

- Construct 3 vectors from each of the encoder’s input vectors
- For each word, Query vector Q, a Key vector, K and a Value vector, V, are created
- K, Q and V are created by multiplying the embedding by 3 matrices learned during the training process.
- **Important:** these attention vectors can be of any dimension (depends on the dimension of the matrices multiplying embeddings)

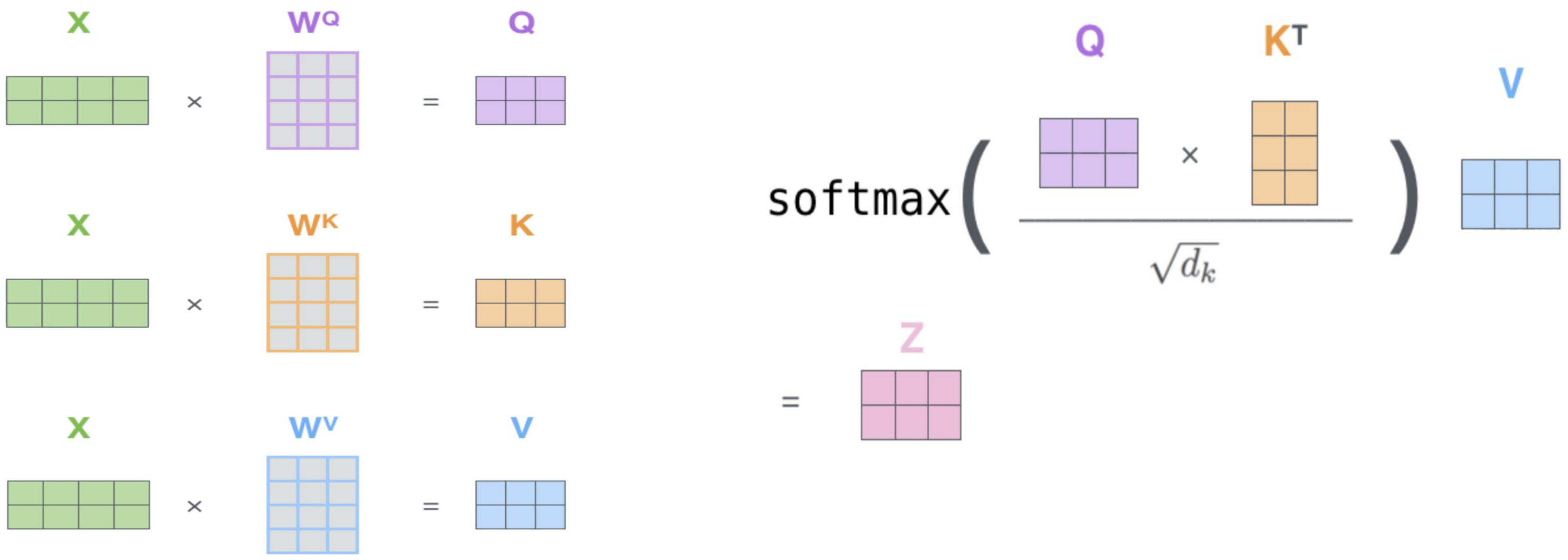
Compute self-attention for each word in sentence



Score each word of the sentence against this word: determines how much focus on other parts of sentence while encoding this word

Multiply each value vector by the softmax score: keep intact the values of the word(s) to focus on, and drown-out irrelevant words (multiplied by tiny numbers)

Self-Attention: General Scheme



$$Z = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V$$

Every row in the X matrix corresponds to a word in the input sentence

Multi-Head Attention

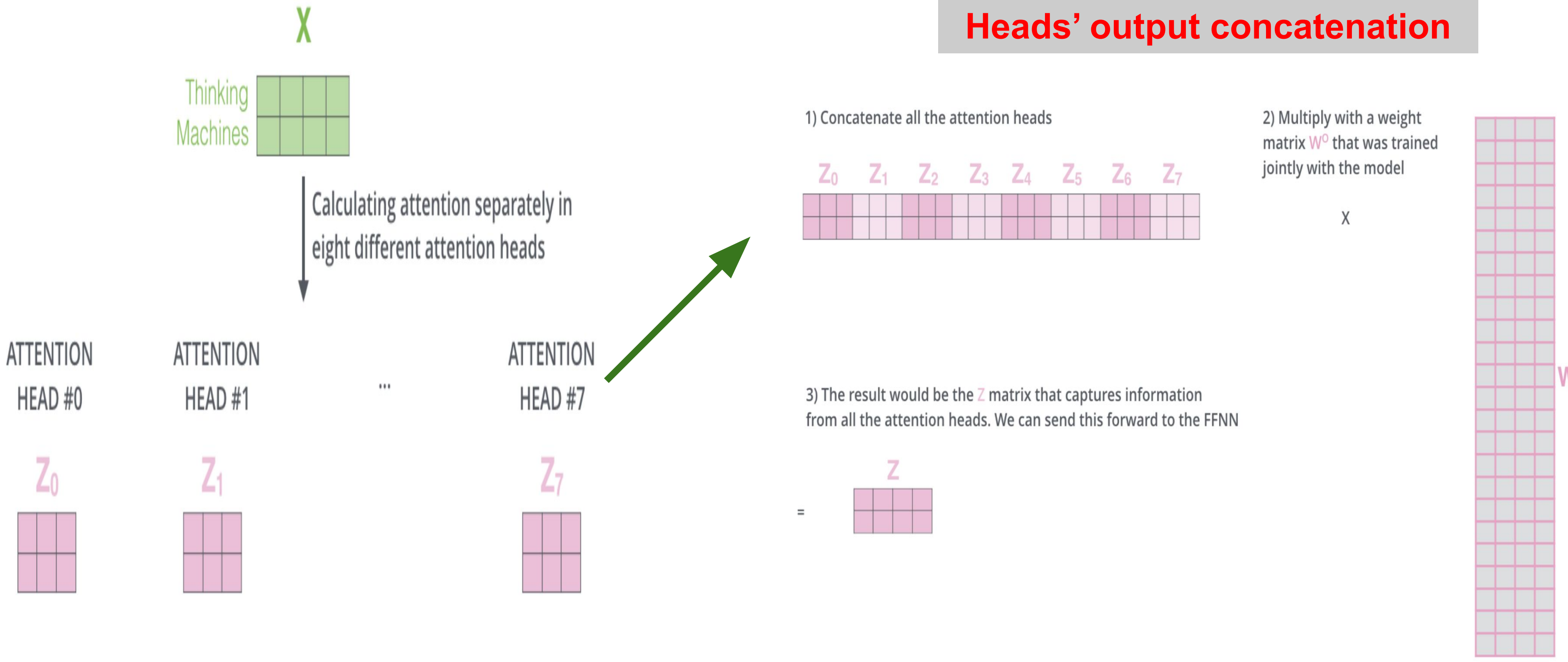
- Each head captures different contextual information by correlating words in a unique manner.

- Expands the model’s ability to focus on different positions
    - In translating a sentence like “The **animal** didn’t cross the street because it was too **tired**”, we would want to know which word “it” refers to.

- Gives the attention layer multiple “representation subspaces”.
    - Multiple sets of Query/Key/Value matrices (Transformer uses 8 heads).
    - After training, each set is used to project the input embeddings (or vectors from lower encoders) into a different representation subspaces

Multi-Head Architecture: All Pieces Together

Heads' output concatenation



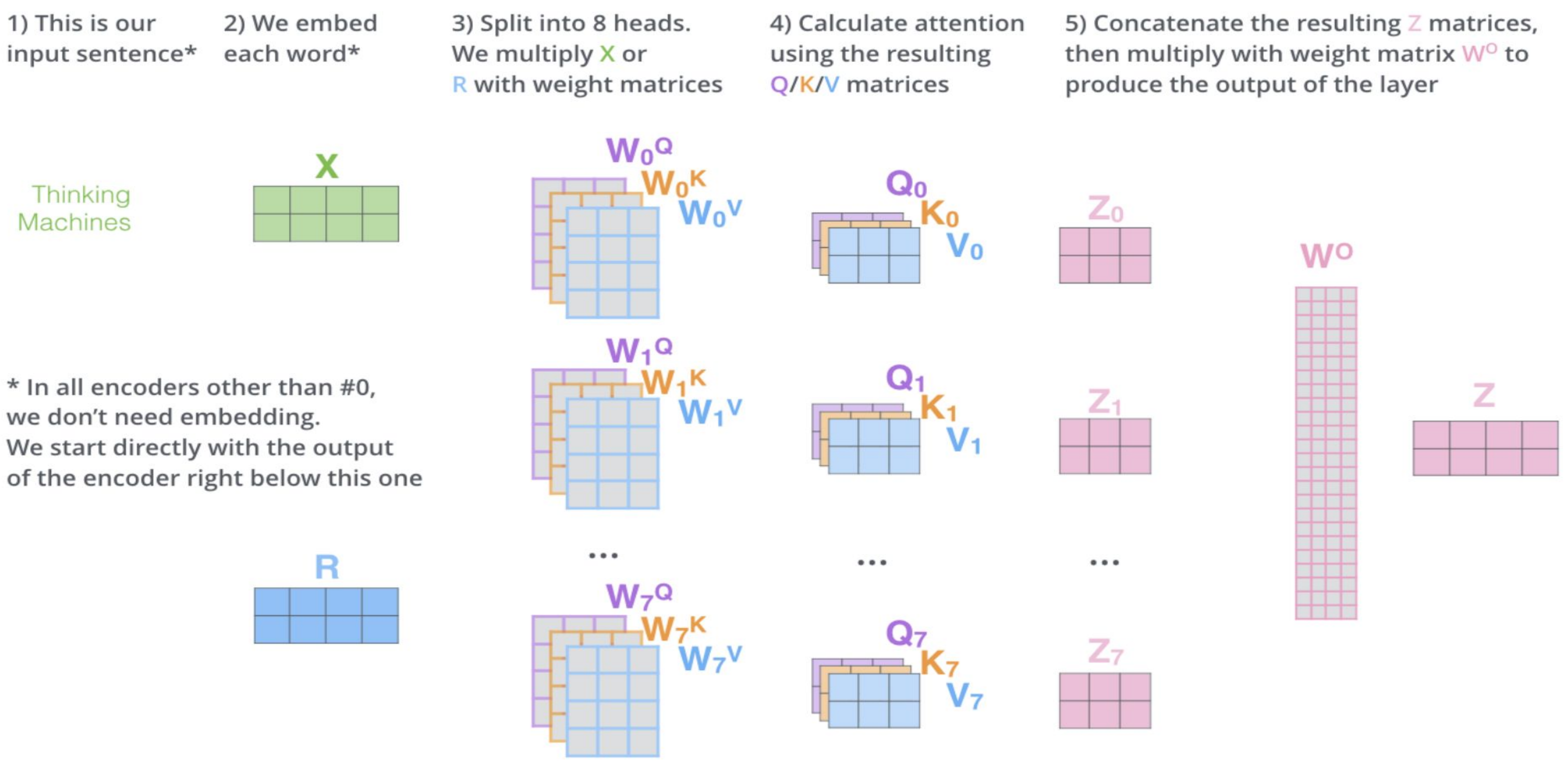
Calculating attention separately in eight different attention heads

1) Concatenate all the attention heads

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

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

Self-Attention: Detailed Scheme


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
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z
...

But what about the words' order?

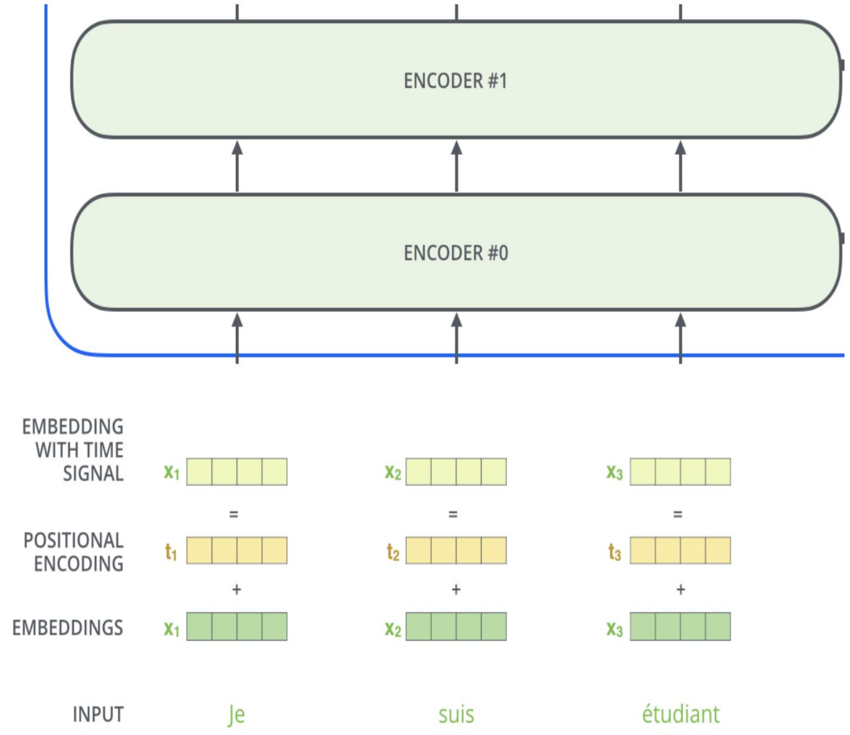
- Self-attention mechanism are agnostic to the words' order which is unacceptable for language model

- **Question:** How we take words order into account?

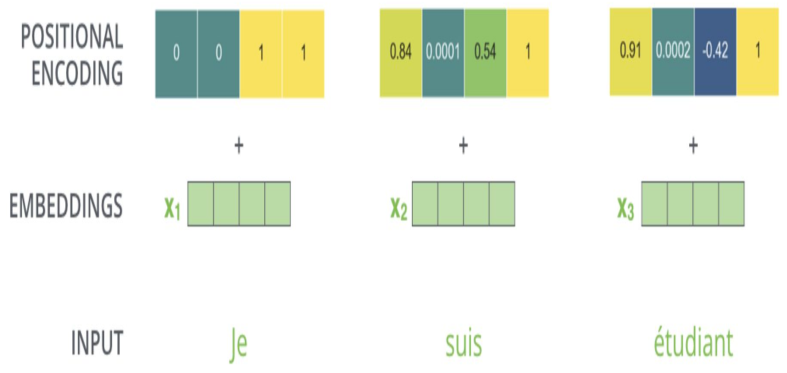
- **Answer:** Positional Embeddings -encodes the absolute or relative positional information into the word embedding.

- Adds a position embedding with the same dimension to the word embedding

Positional Embedding: How it works?



To give the model a sense of the order of the words, we add positional encoding vectors -- the values of wh



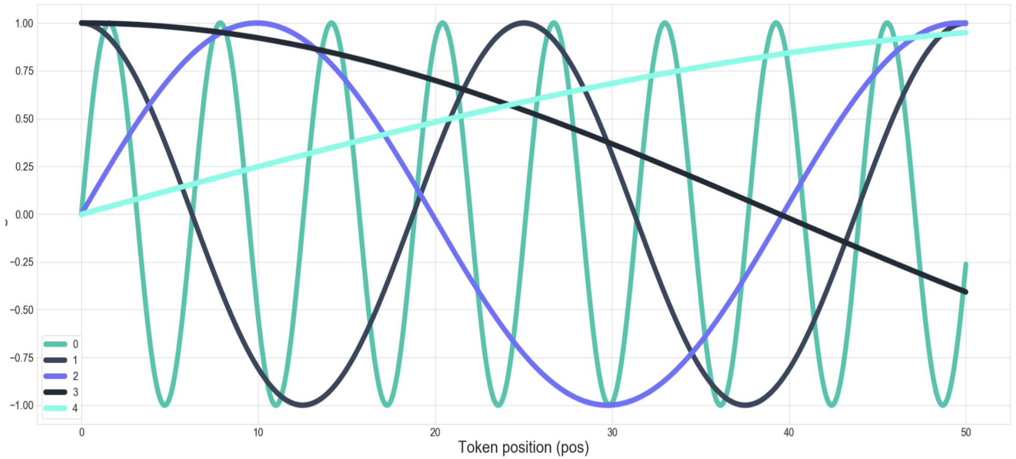
A real example of positional encoding with a toy embedding size of 4

Positional Embedding: How it works?

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

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

Alternating positional encoding values. Using word position **pos**, embedding dimension **i**, and the number of embedding dimensions **d_model**.



Sinusoidal functions of the first five embedding indices using a total embedding dimensionality of 20. The embedding index positions are shown in the legend.

Periodic function allows us to embed relative word position information into the word embedding.

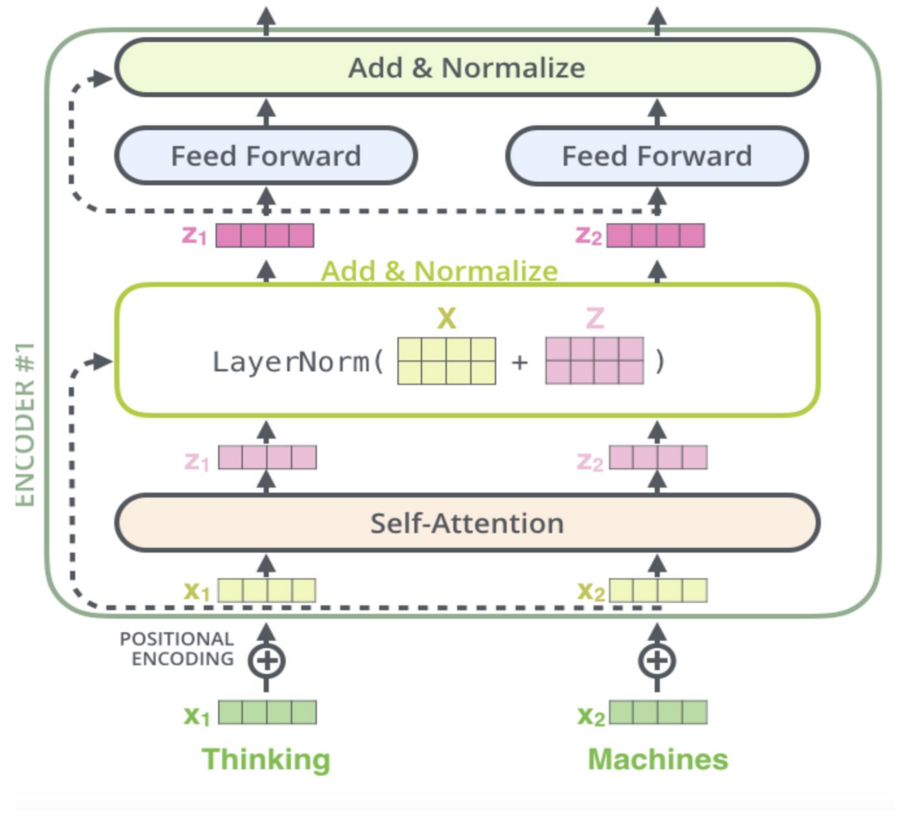
Encoder: Sum up

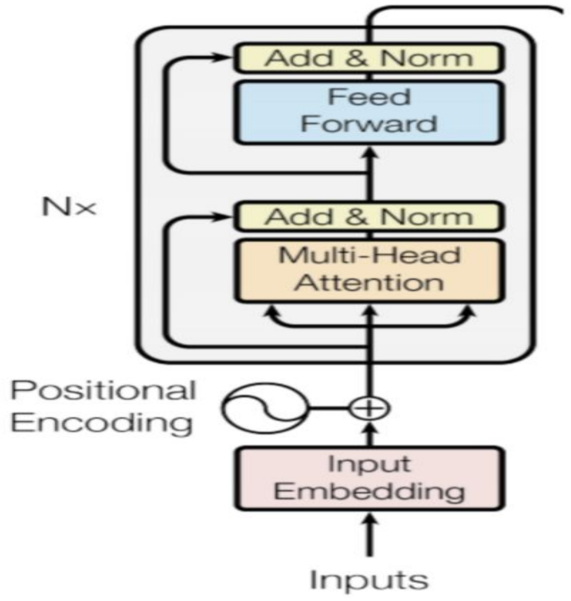
1. Word embeddings of the input sentence are computed simultaneously.

2. Positional encodings are then applied to each embedding resulting in word vectors that also include positional information.

3. The word vectors are passed to the first encoder block.

Full Encoder Architecture

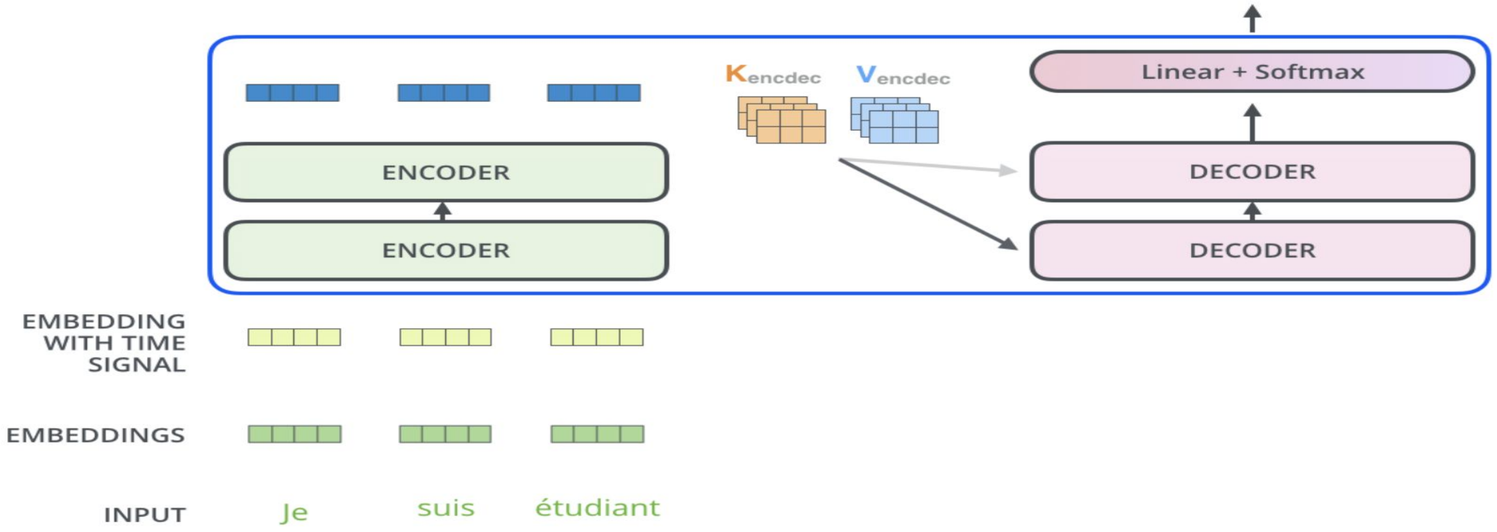




(a) Encoder

Original paper encoder architecture

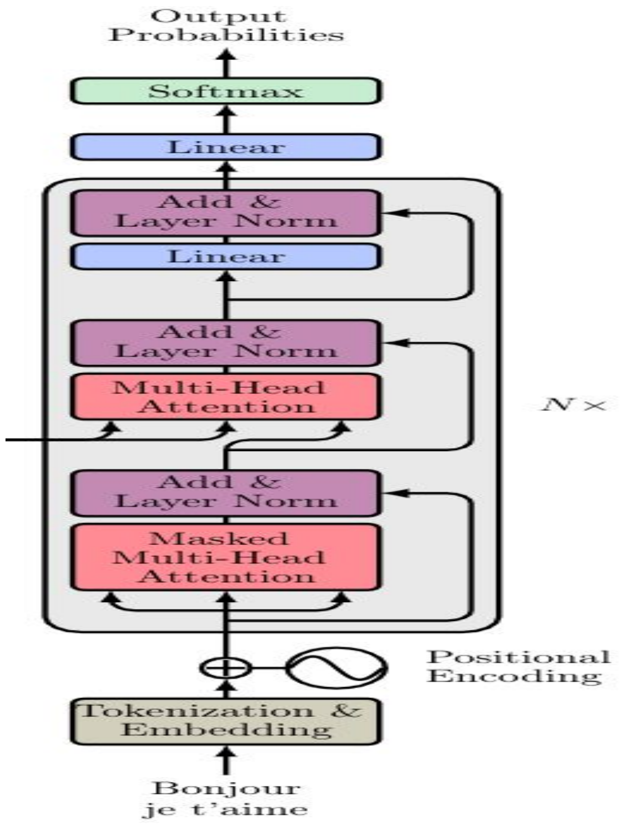
Decoder: Just a couple important notes



- The output of the top encoder is transformed into a set of attention vectors K and V

- They used by each decoder in its “encoder-decoder attention” layer helping the decoder focus on appropriate places in the input sequence

Decoder Architecture: Details



1. **Masked multi-head self-attention layer**
2. Normalization layer followed by a residual connection
3. **Multi-head attention layer (Encoder-Decoder attention)**
4. ….
5. The last layer predicts the next token in the output sentence(probabilities)

🟪 Most Important parts

Decoder: Masked Self-Attention

- Predict one word (token) after another - sequential processing unavoidable(e.g. translation)
- **We don't know the whole sentence because it hasn't been produced yet**
- Decoder self-attention layer **is only allowed** to use earlier positions in the output sequence.
- Mask future positions (set to -inf) in the self-attention calculation

$$\mathrm{MaskedAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{M}}{\sqrt{d_k}}\right)\mathbf{V}$$

M contains -infs and zeros ONLY

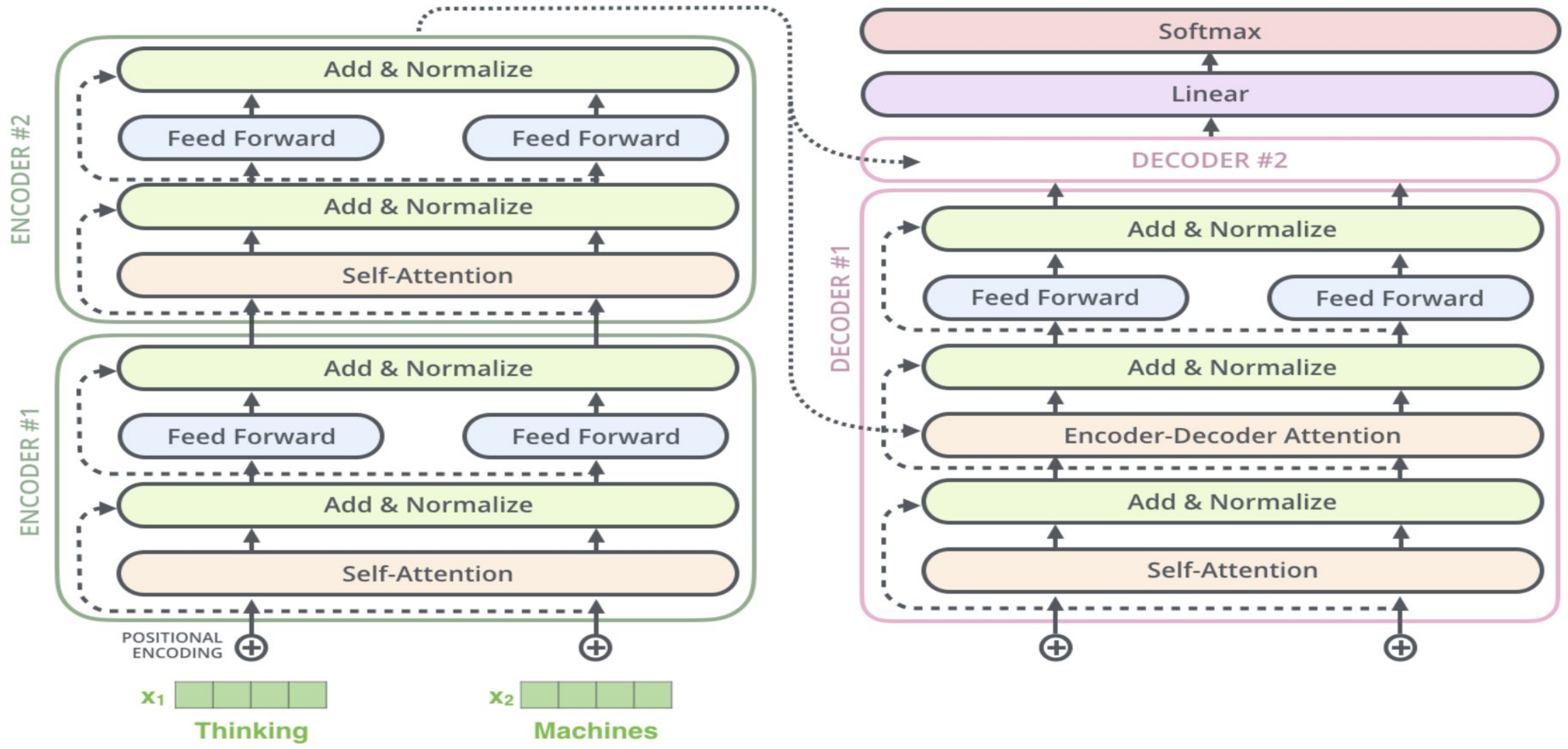
Decoder-Decoder Attention Layer

- Combines input and output sentence

- Encoder’s output constitutes input sentence embedding(database) - **used to produce Key and Value matrices**

- Masked Multi-head attention output contains so far generated sentence: represented as **Query matrix in attention layer** : “search” in the database.

Full Transformer Architecture



Transformer Success Reasons:

A. Distributed and independent representations at each block
   a. Each transformer block has 8 contextualized representations capturing different input features

B. Meaning heavily depends on the context: self-attention
   a. Relationships between word representation are expressed by attention weights
   b. No locality notion model makes global associations

C. Multiple encoder and decoder blocks
   a. Builds more abstract representations with more layers.
   b. Similar to stacking recurrent/convolution blocks we can stack multiple transformer blocks (~receptive field in terms of pairs of distributed representations)

Transformer Usage Modes

- Encoder-only (e.g. sentiment analysis, topic modeling, NER)
- Decoder-only (e.g. language modeling)
- Encoder-decoder (e.g. machine translation, summarisation, question answering)

Topics Not Covered :(

- How Transformer is trained (masking words)
- Transformer adaption to perform a wide range of NLP tasks
- Main issues with Transformers: Complexity!!
- Different Transformer architectures, improvements
- Transformers for images, time-series data and other domains

Thank You

For Your Attention