BERT

Pre-training of Deep Bidirectional Transformers for Language Understanding

Overview

- Define what are pre-trained language representation models
- Understand BERT architecture
- Which is Based on Transformers, which includes :
  - Which includes Encoder - Decoder
  - Attention and Self-Attention
-
- And finally, we'll see an application of BERT for document classification

Pre-trained language representation models

Language representation... We’ve got word2vec already!

- True ! But …
- Unable to process unknown or out-of-vocabulary (OOV) words.
- Not a multilingual models (requires new embedding matrices and not allow for parameters sharing)
- Represents every word as an independent vector
  - Only captures weak relations between words
  - No difference with “bank account” and “the bank of the river”
- We need to build more complex relationship than word encoding

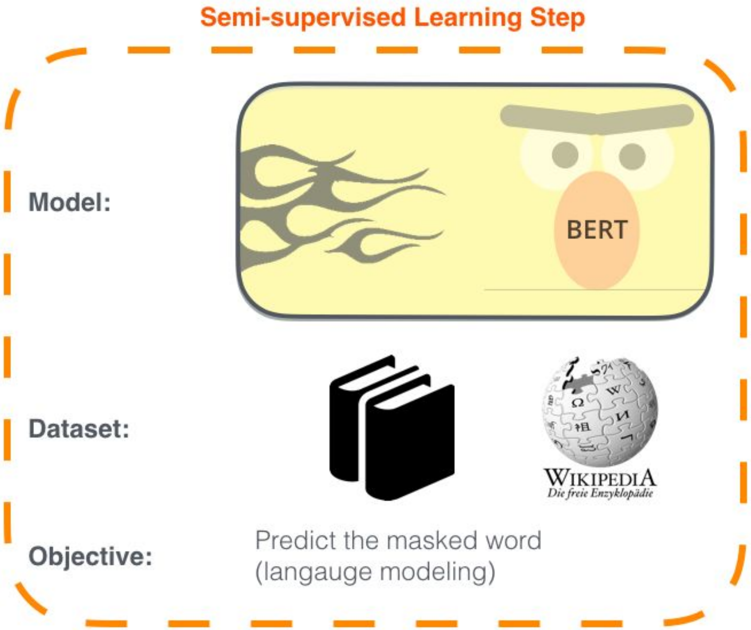
Why pre-trained ?

- Many NLP tasks lack labeled data specific to the task.
- It’s an issue as deep learning-based models benefit from training over millions or even billions of annotated examples.
- **Global idea over the past few years : train general purpose language representation models.**
- Training in two phase : pre-training over enormous amount of unannotated data from texts and web and fine-tune it on a smaller dataset corresponding to the downstream task.

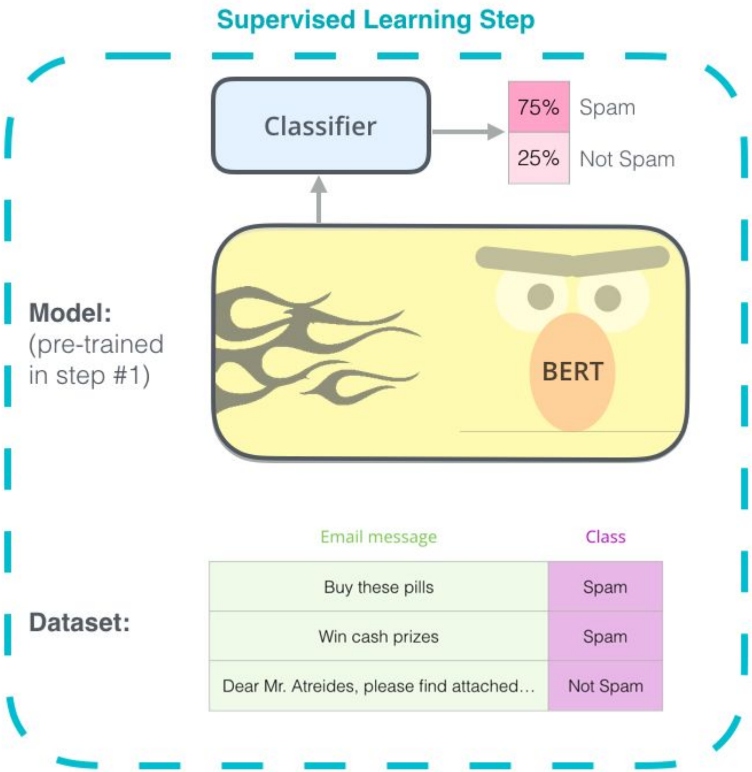
Example : sentence classification (Spam or not Spam)

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - Supervised training on a specific task with a labeled dataset.



BERT’s architecture

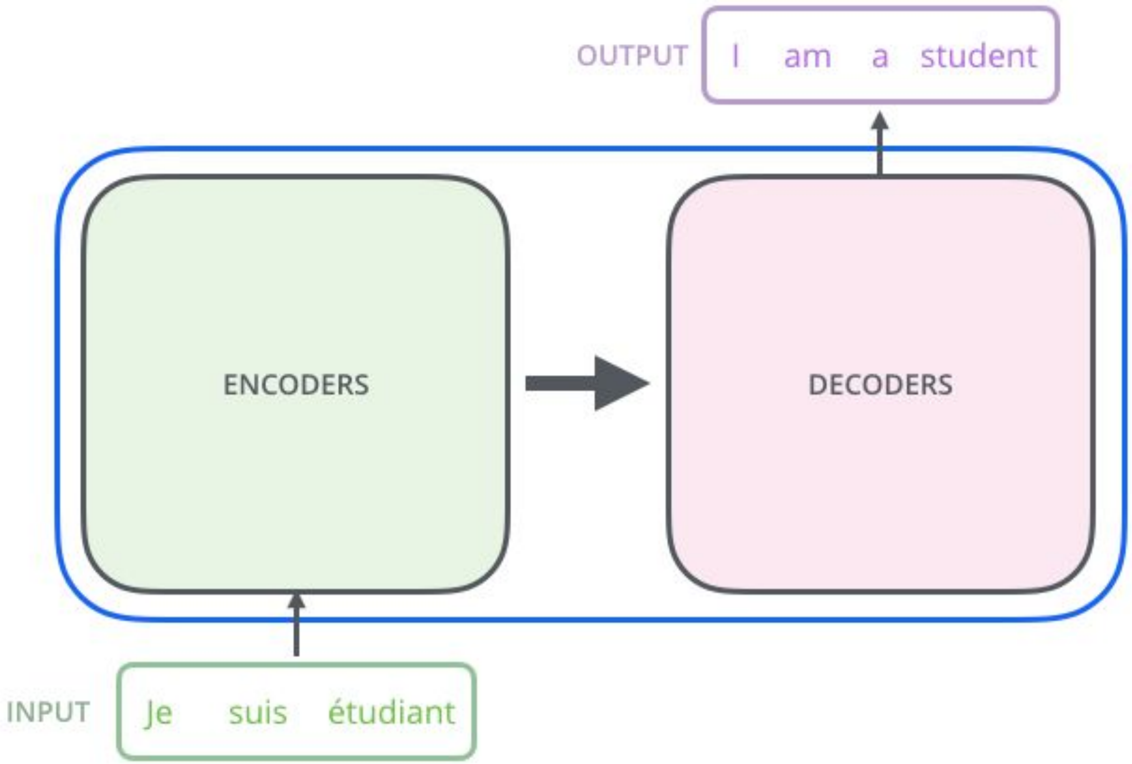
BERT : Bidirectional Encoder Representations from Transformers

- BERT is made of two big stacks : one of Encoders and one of Decoders
- Actually its architecture is based on the Transformer's architecture
- The
- So, let's start by understanding what a Transformer is...

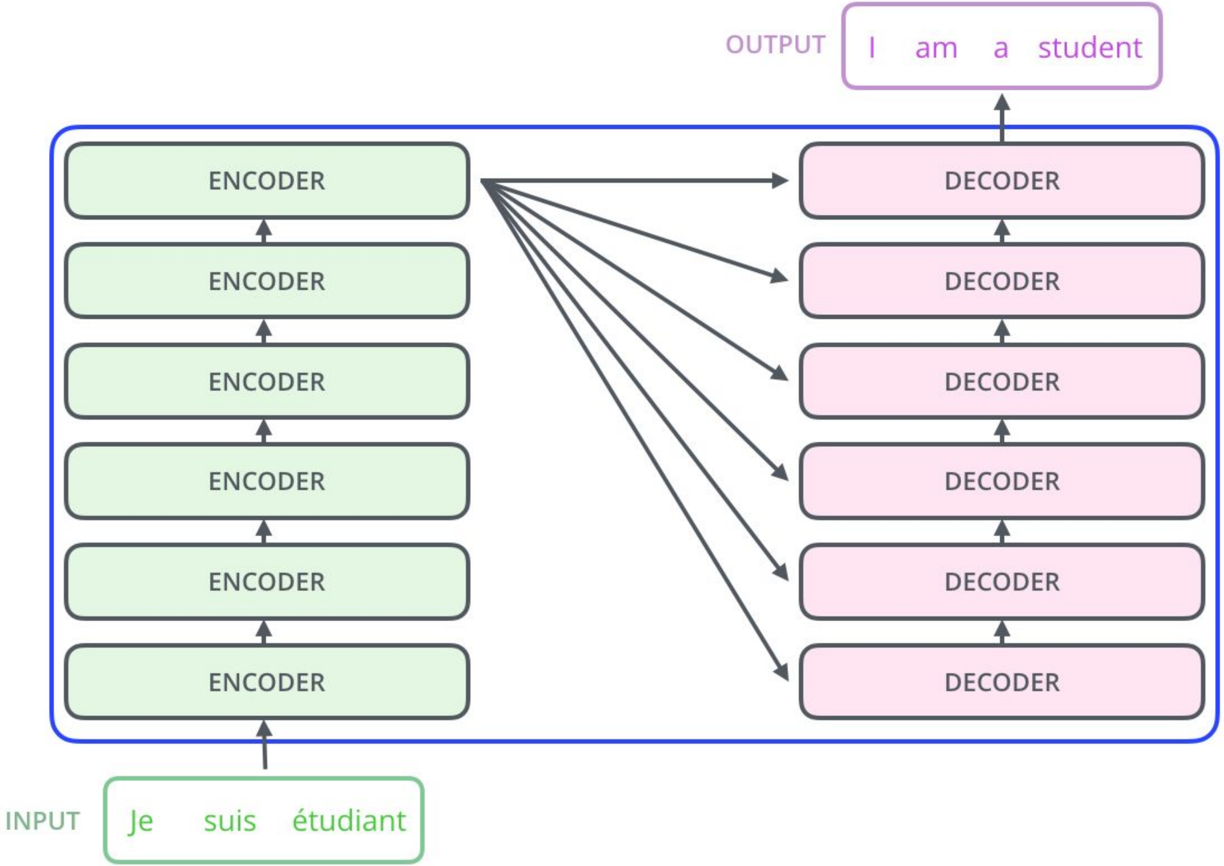
The Transformer

Transformer - Overview as a black box

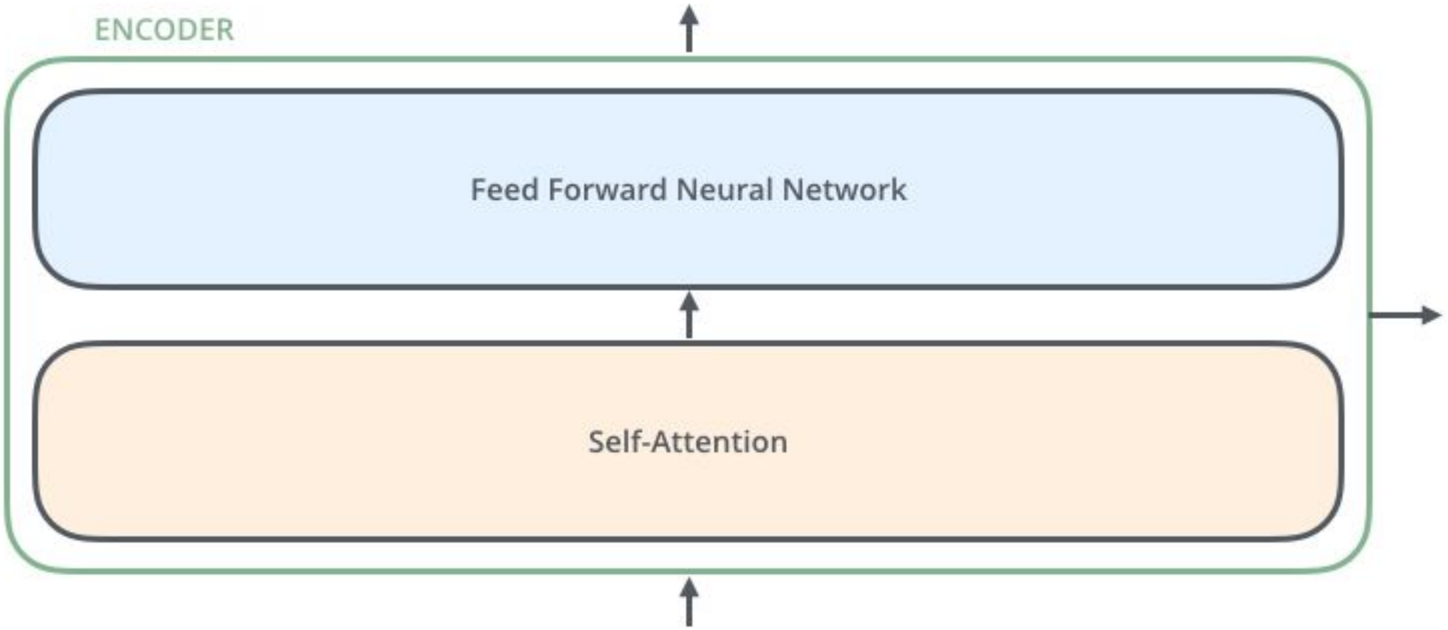
Transformer - Encoding component and Decoding component



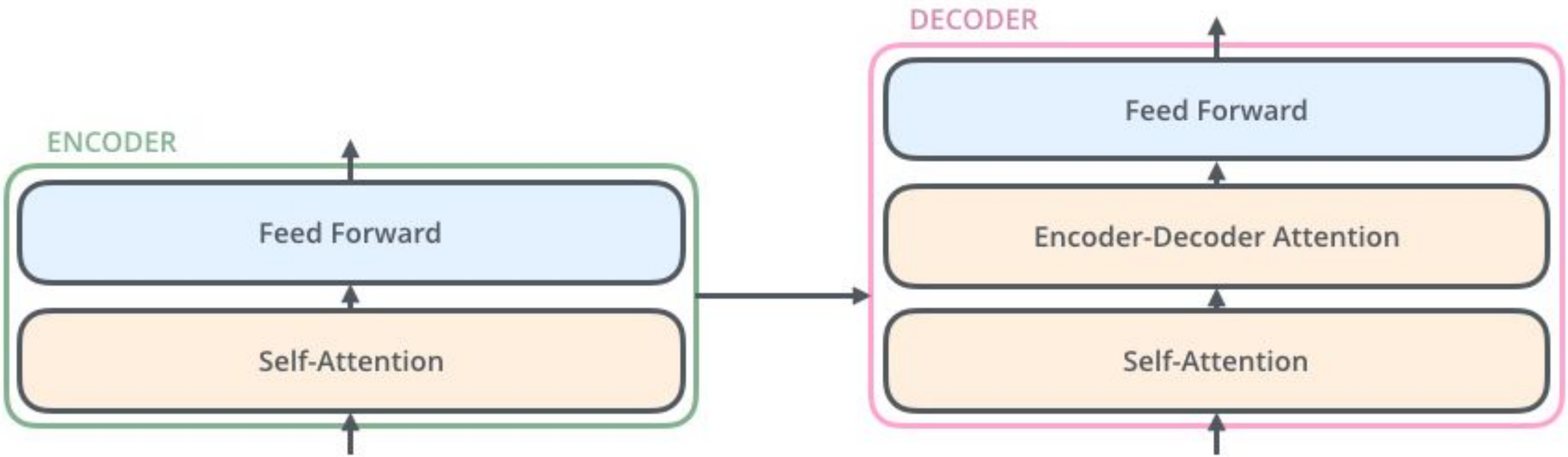
Transformer - Encoding component and Decoding component



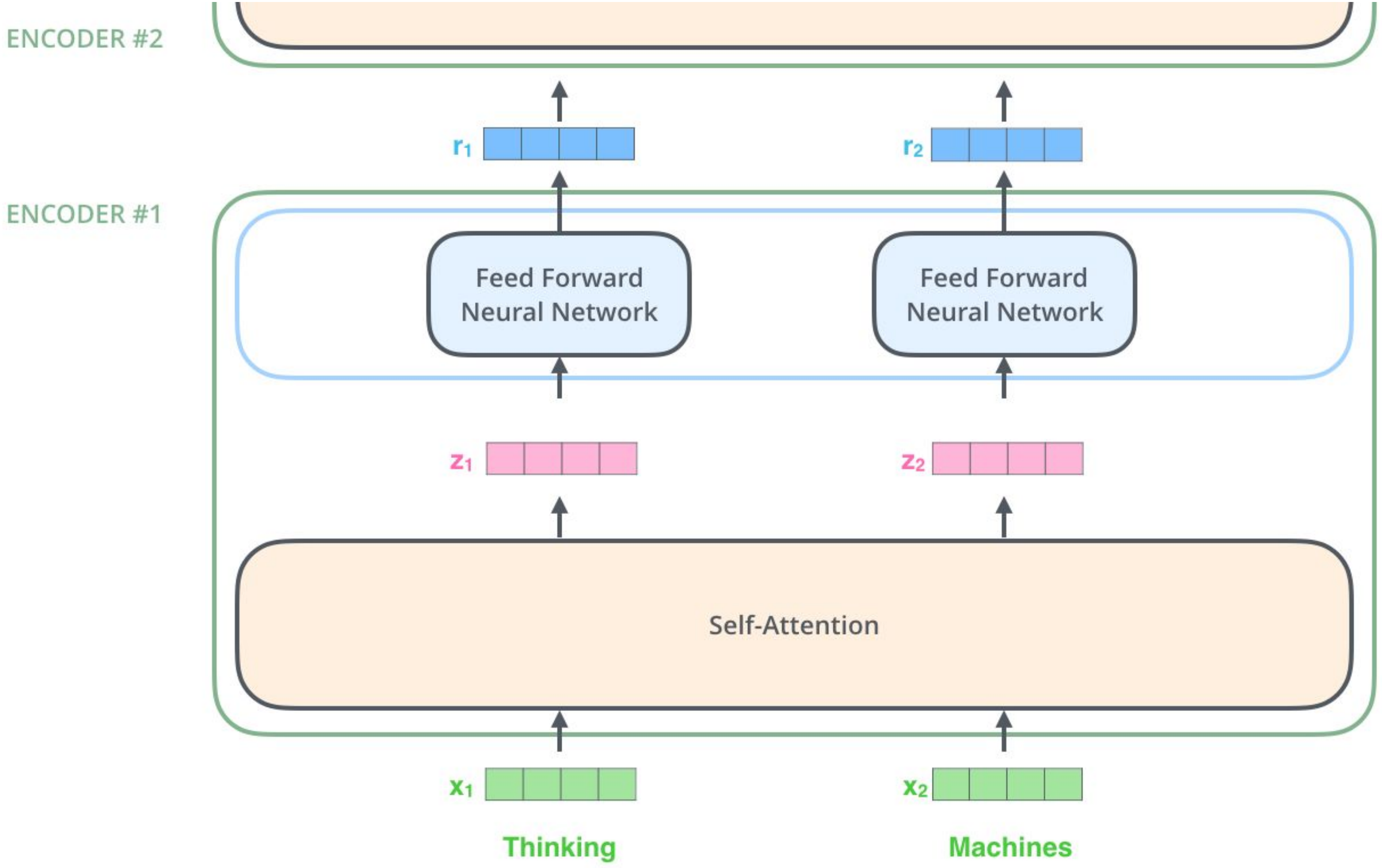
Transformer - Encoder



Transformer - Encoder and Decoder

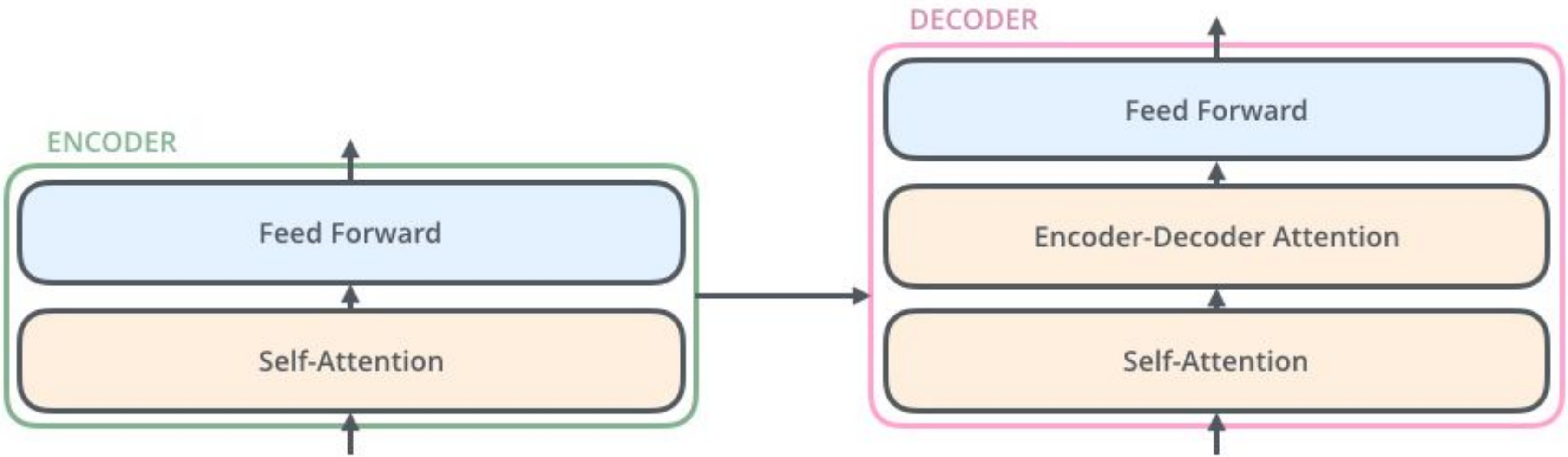


Transformer - Inputs and Outputs

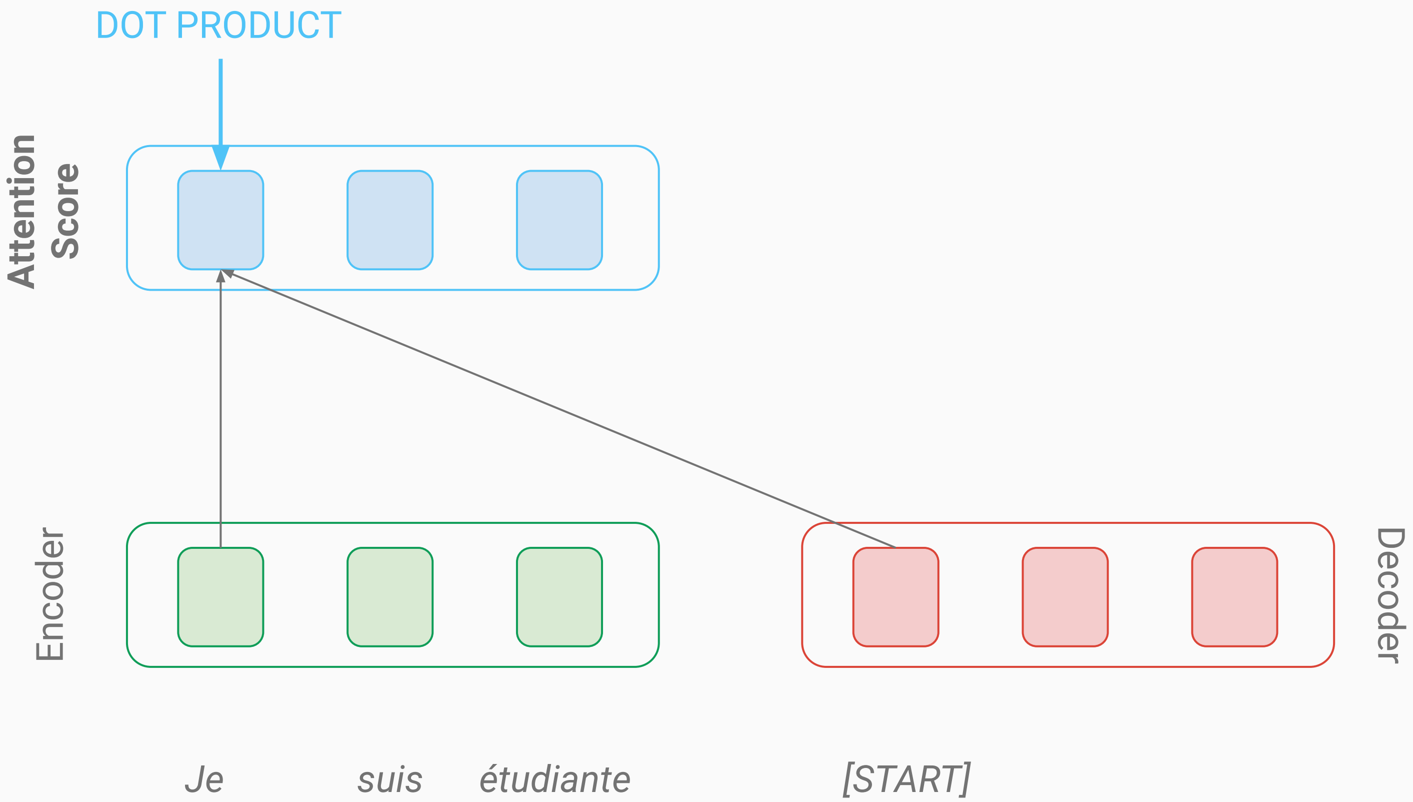


Attention and self-Attention

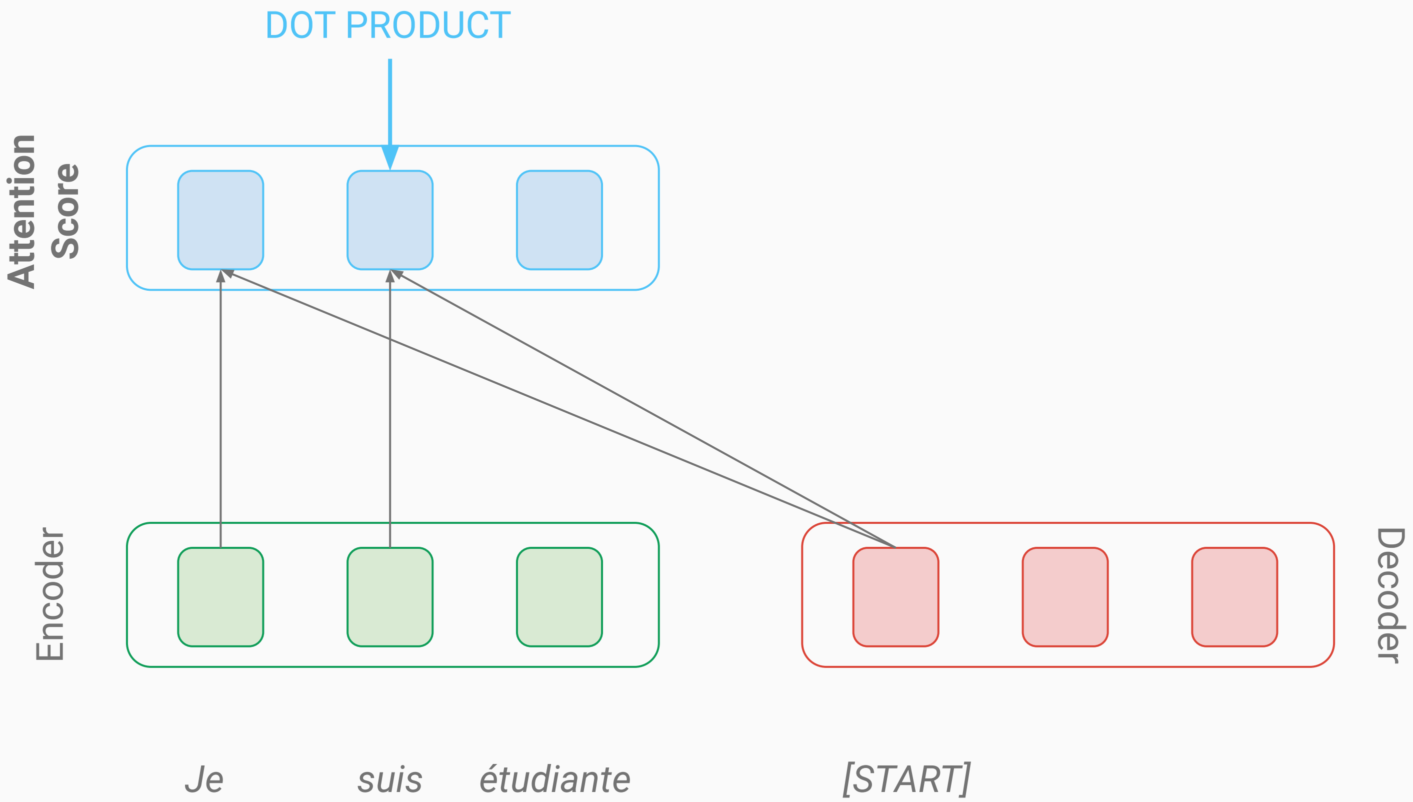
Transformer - Remember, Attention and self-Attention



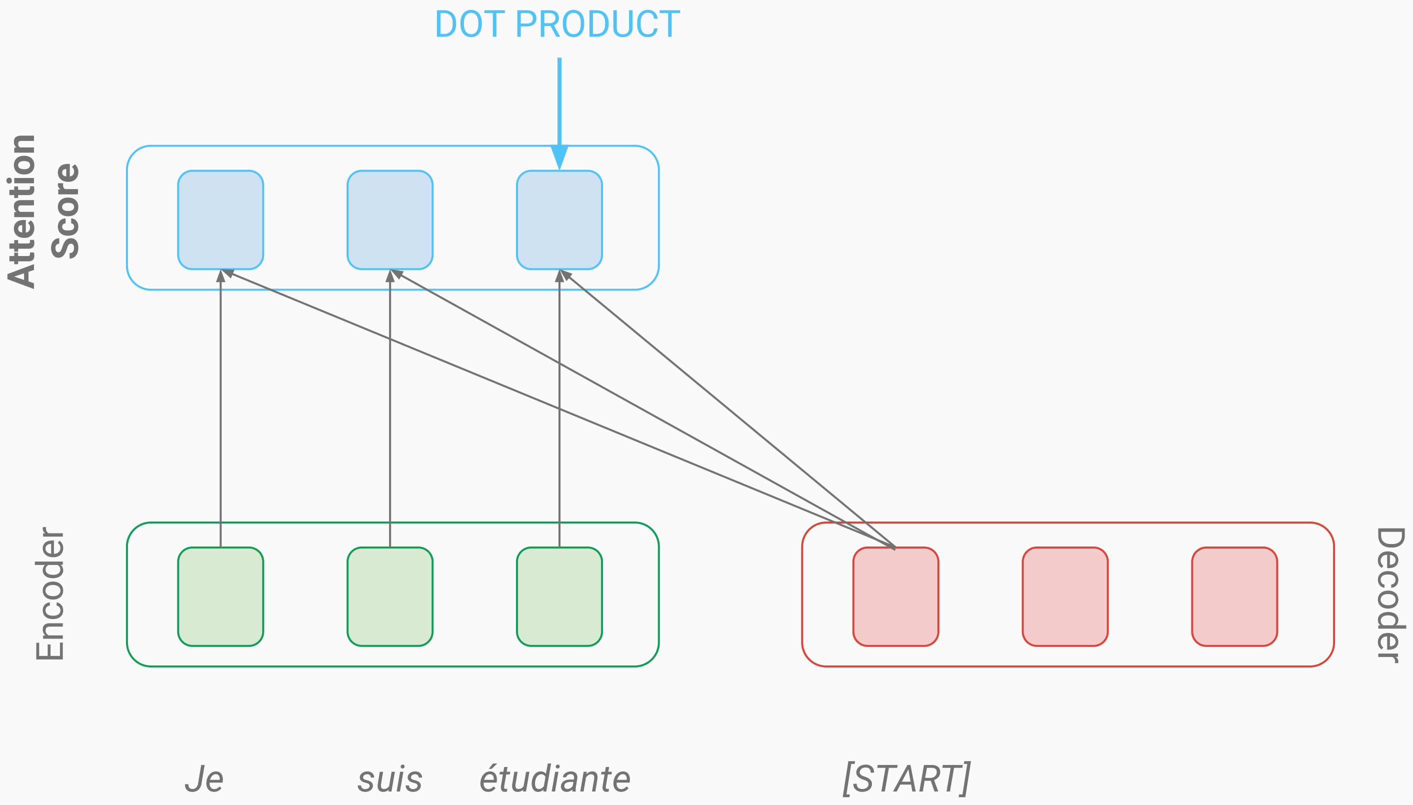
Transformer - Attention



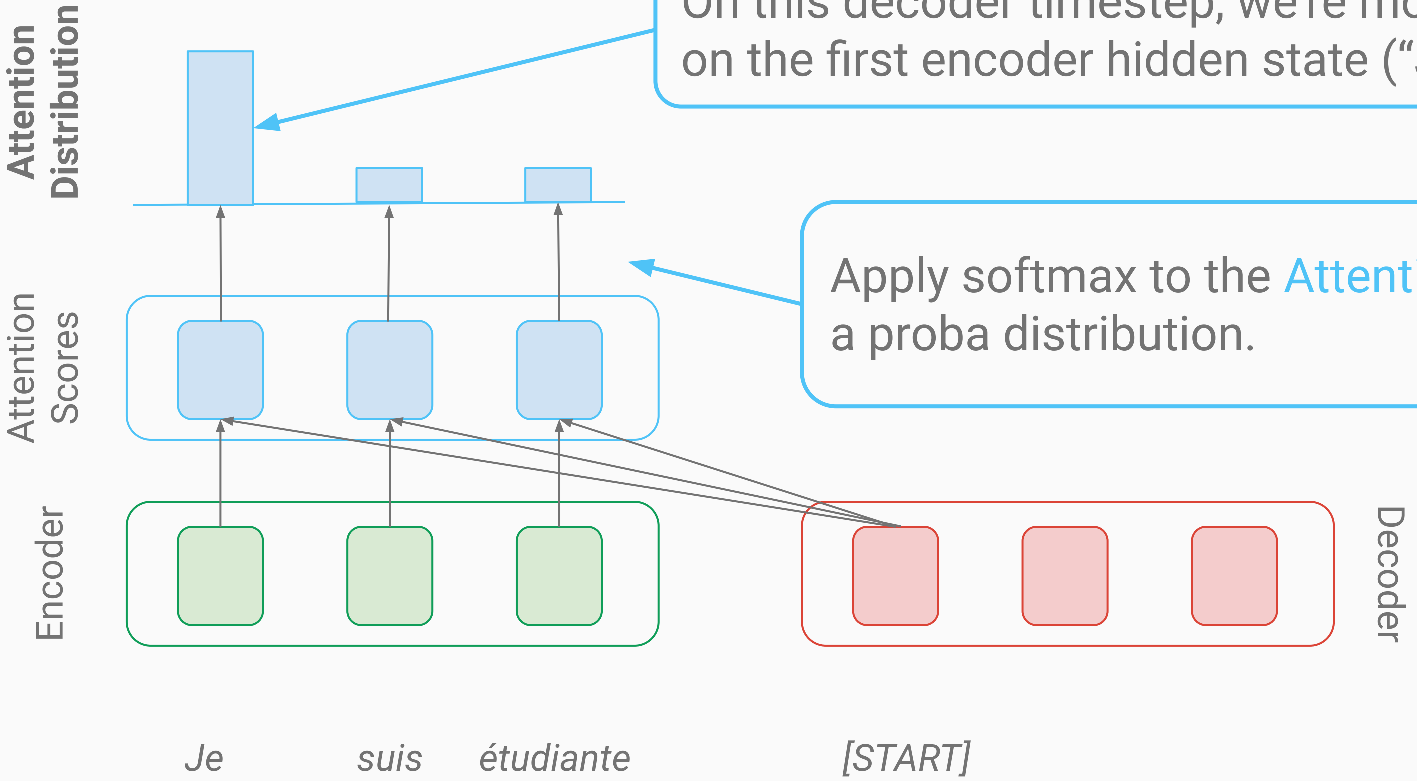
Transformer - Attention



Transformer - Attention



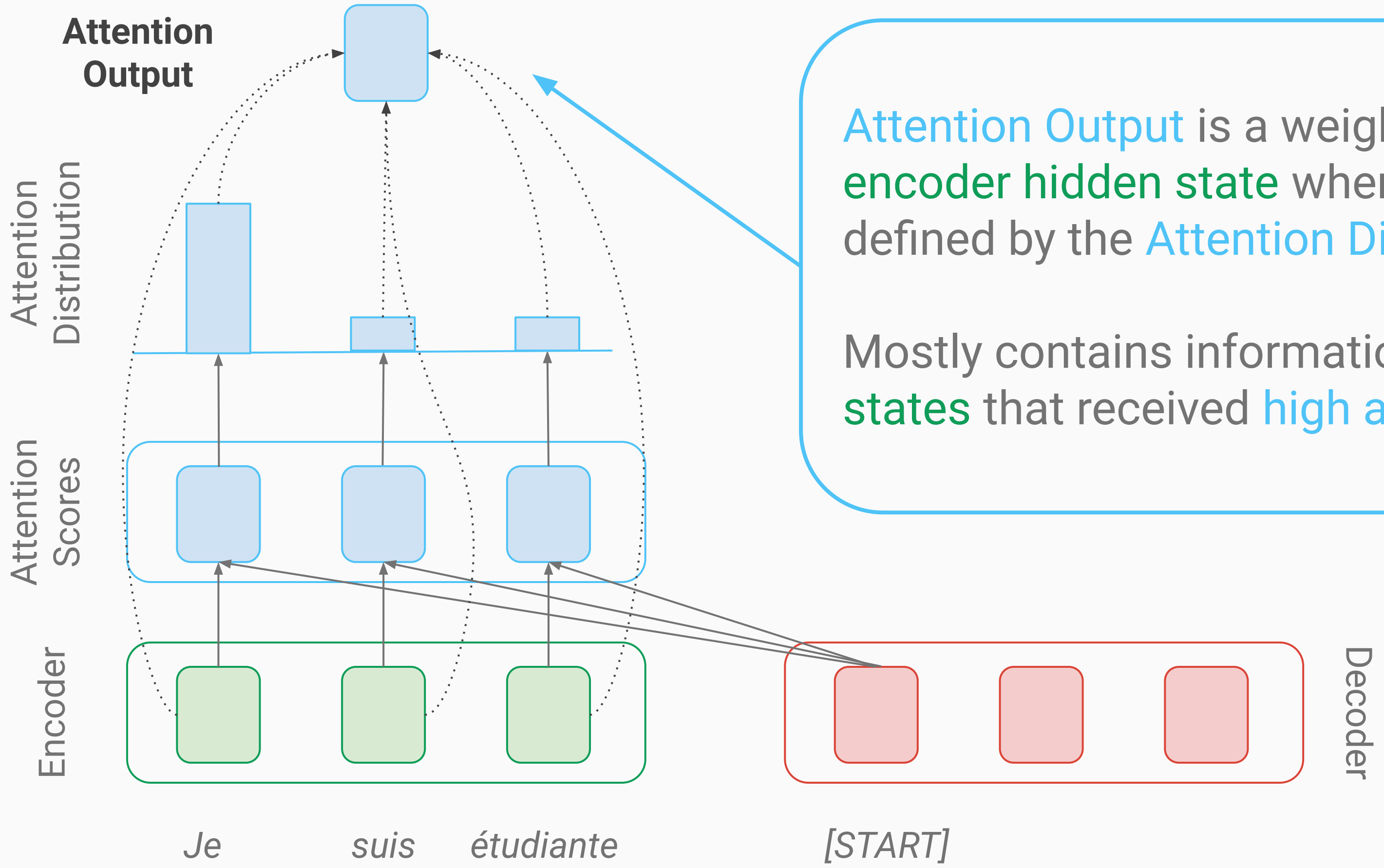
Transformer - Attention



On this decoder timestep, we're mostly focusing on the first encoder hidden state ("Je").

Apply softmax to the Attention Scores to get a proba distribution.

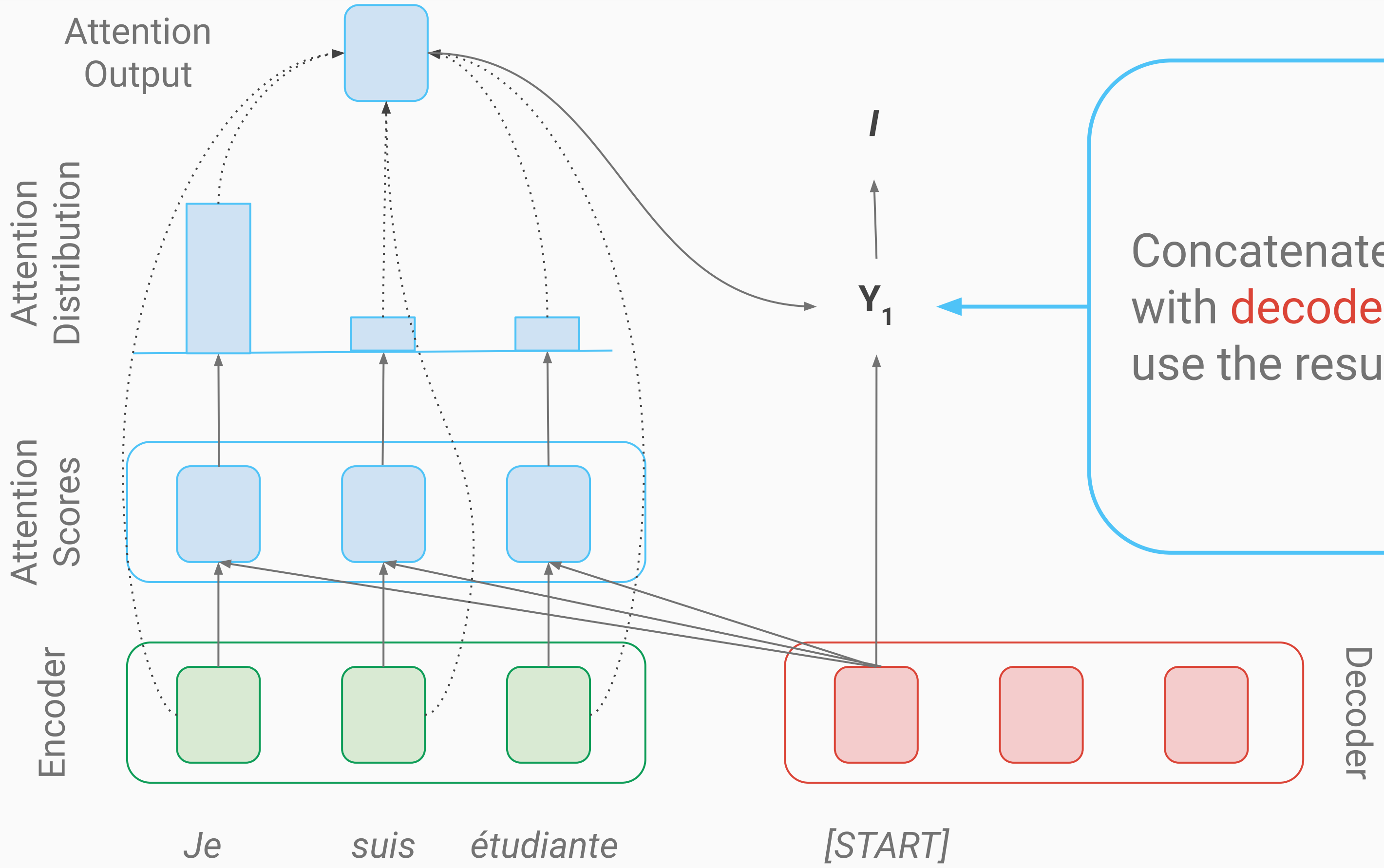
Transformer - Attention



Attention Output is a weighted sum of the encoder hidden state where the weights are defined by the Attention Distribution.

Mostly contains information from hidden states that received high attention.

Transformer - Attention



Concatenate Attention Output with decoder hidden state, use the result to predict Y_1.

Generalize Attention Definition

Given a set of vector values, and a vector query, **Attention** is a technique to compute a weighted sum of the values, dependent on the query.

In our case, we had decoder hidden state attending to encoder hidden state :

- Queries -> Decoder hidden state
- Values -> Encoder hidden state

Now, let’s see how we it works with self-Attention.

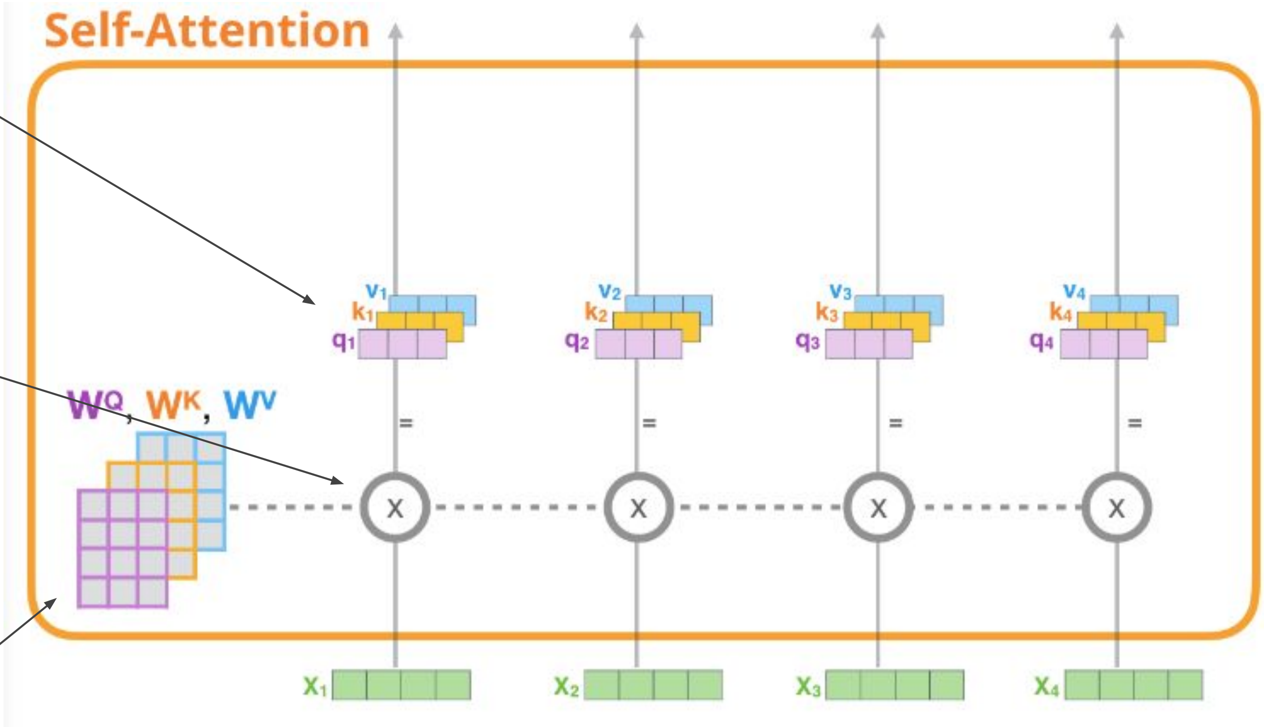
Transformer - Self-Attention : Queries, Keys and Values

Which builds our Query vectors, Key vectors and Value vectors

Multiply each input vectors (word embedded) by each matrices.

Three matrices (init random)

W^Q, W^K, W^V

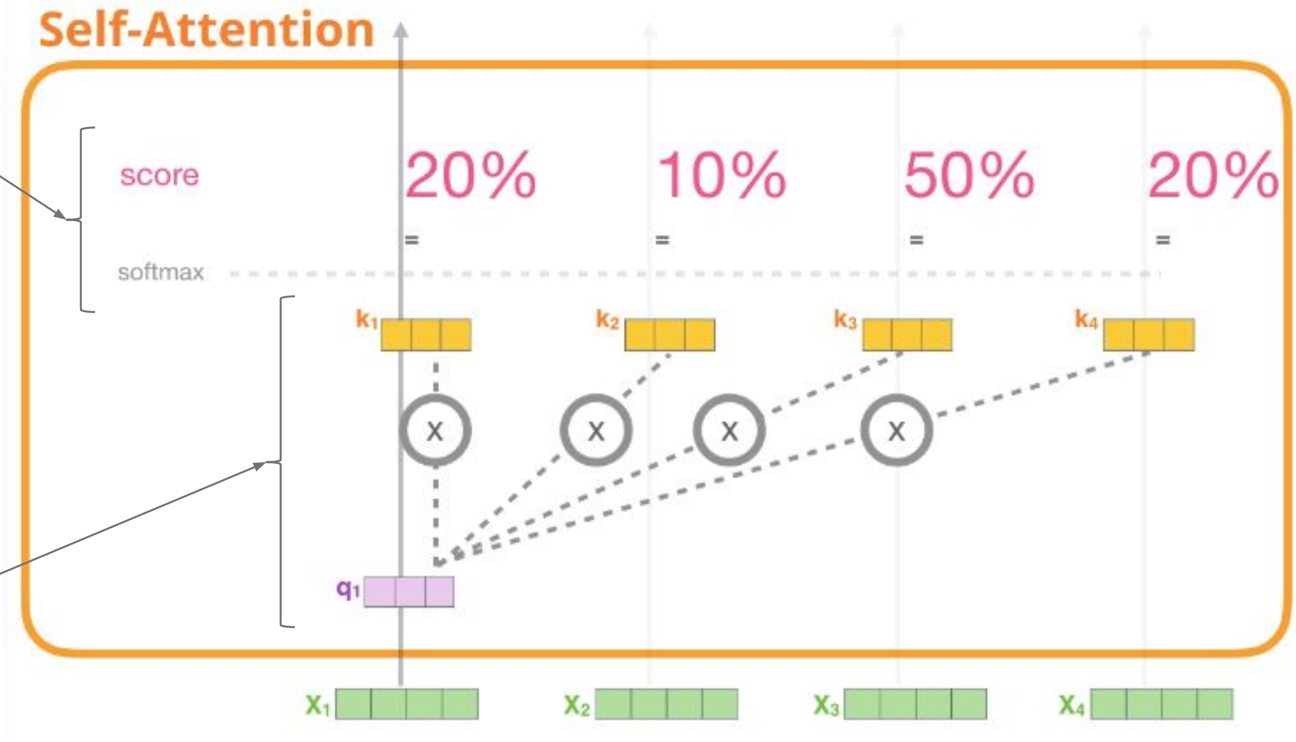


Transformer - Self-Attention : Compute Attention Score and Attention Distribution

Apply softmax on all attention scores to get the Attention Distribution.

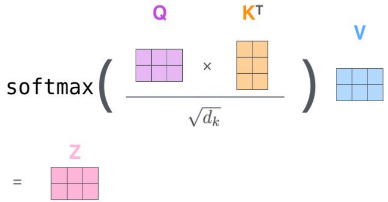
Attention score = $q_i \cdot k_i$

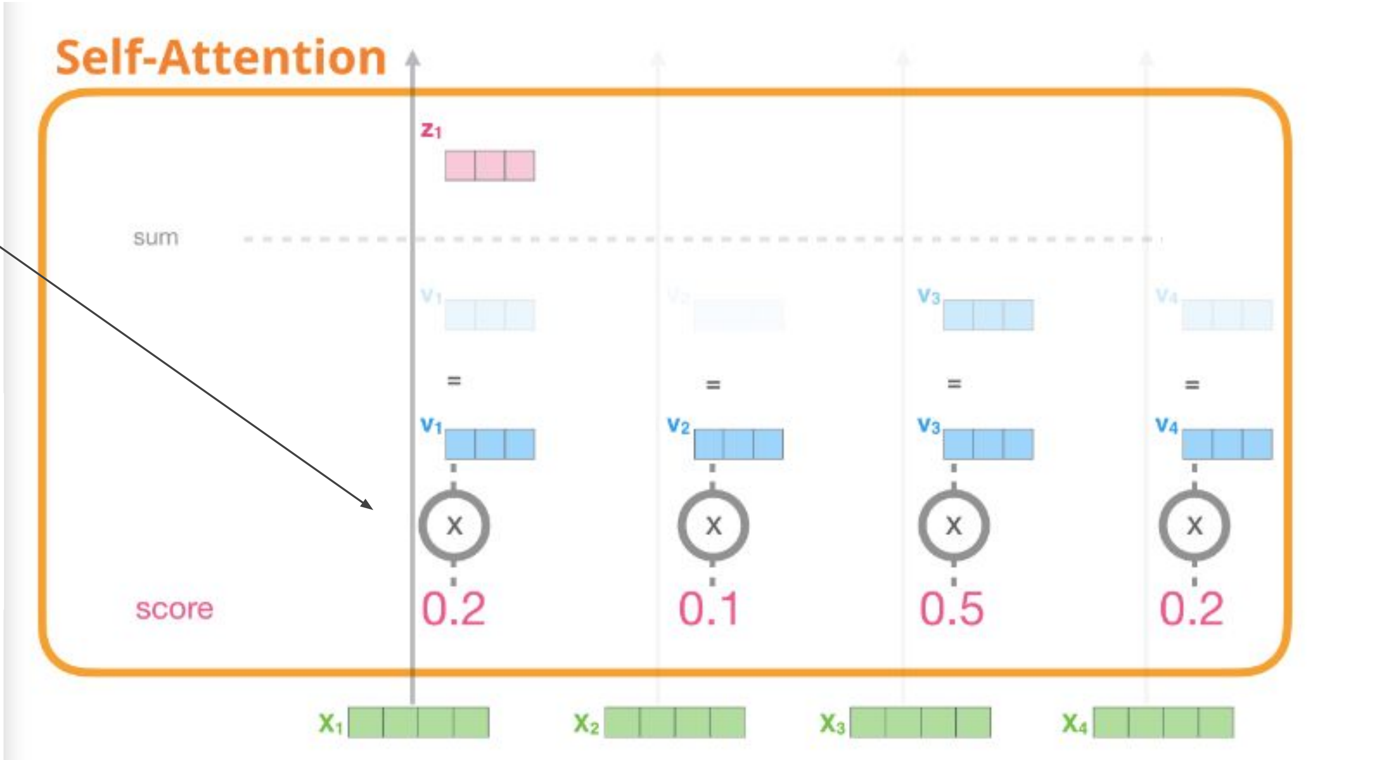
Scores are normalised afterward by the square root of the dimension of the key vectors



Transformer - Self-Attention : Compute Attention Output

Attention output = value vector x attention distribution





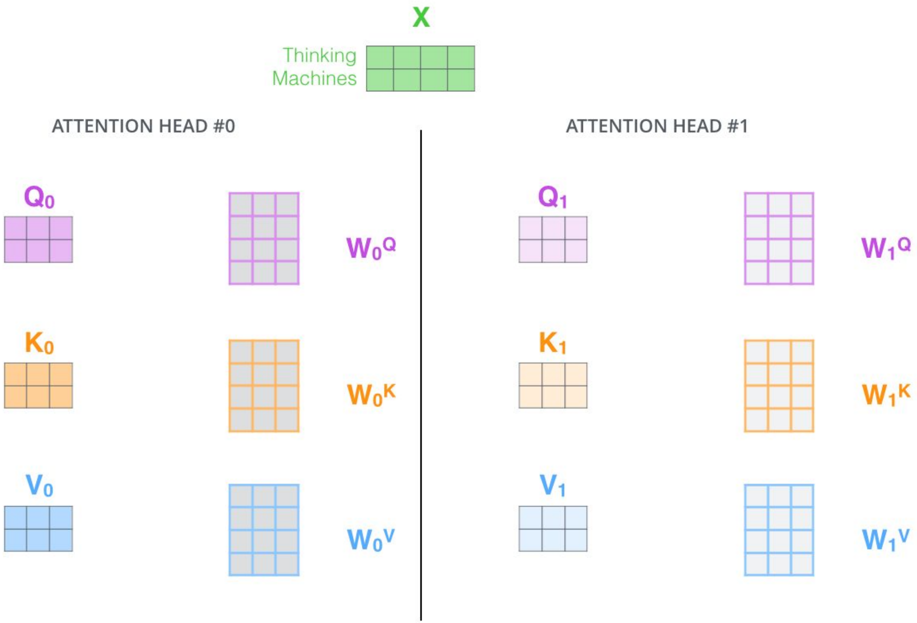
Multi-Head Self-Attention

Transformer - Multi-Head Self-Attention

Gives attention layer multiple "representation subspaces" by building multiple sets of Query, Key, Value weight matrices

- The transformer uses 8 attention head
- So each encoder and decoder as 8 set of matrices.

Each sets are randomly initialized and then trained.

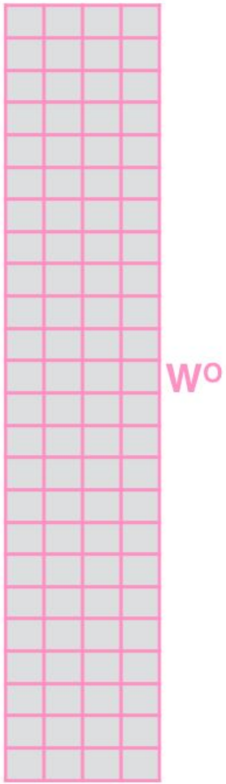


Transformer - Multi-Head Self-Attention

1) Concatenate all the attention heads

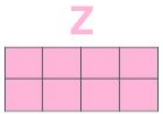
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

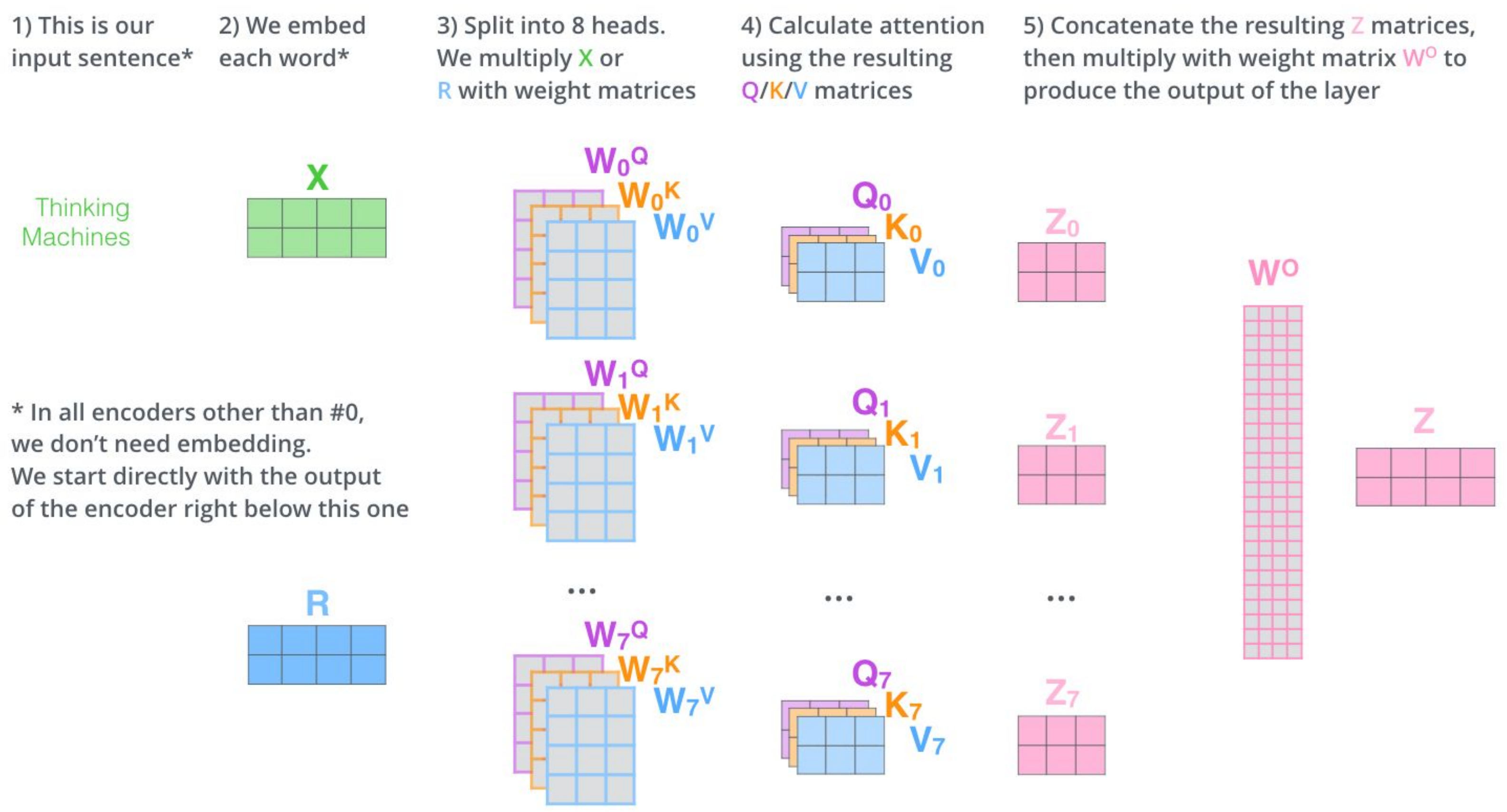


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

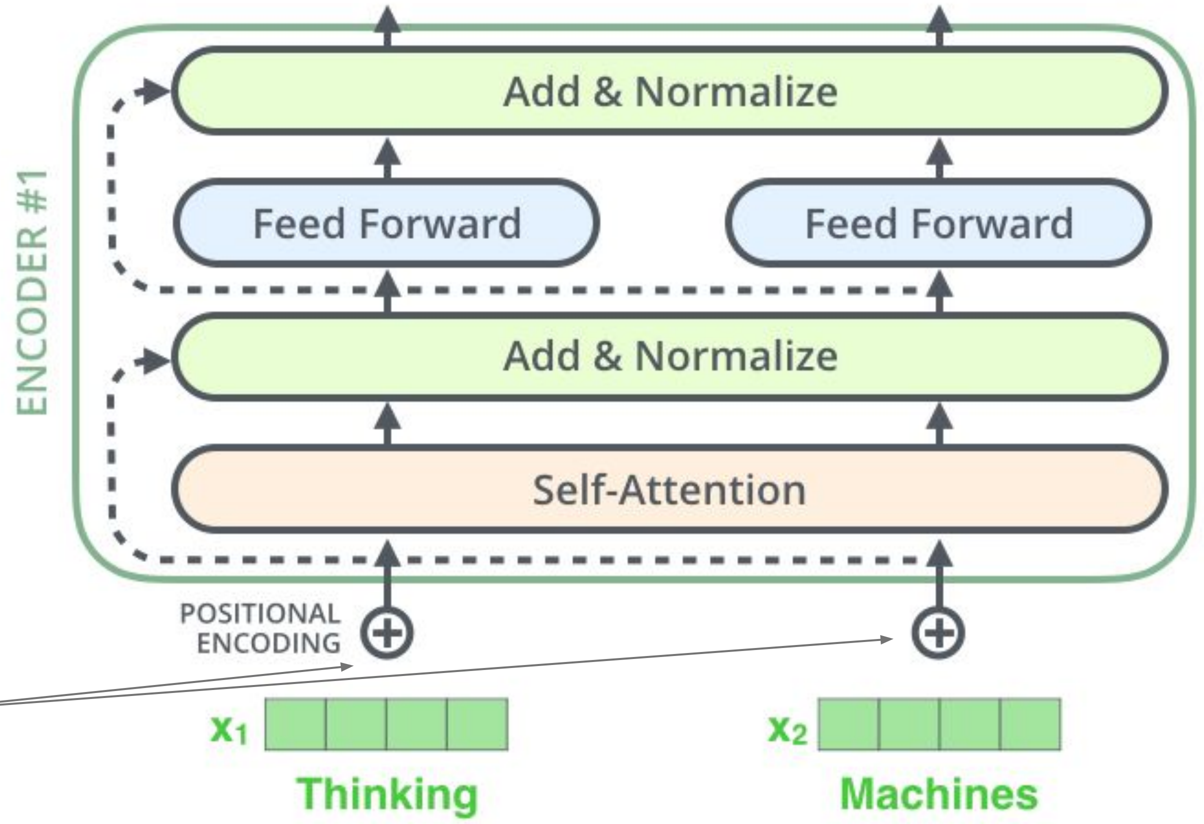


Transformer - Multi-Head Self-Attention : recap



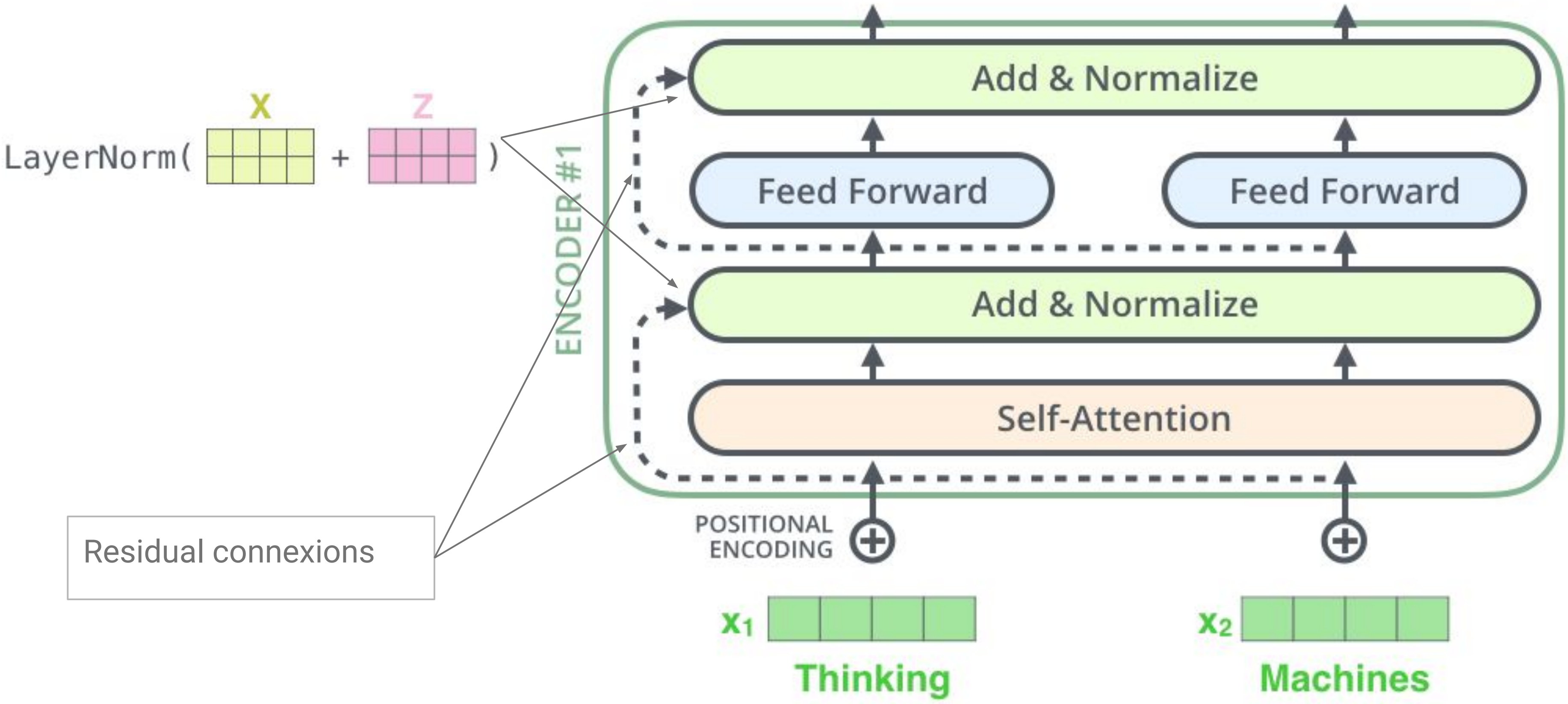
Transformer : almost done

Transformer - Positional encoding

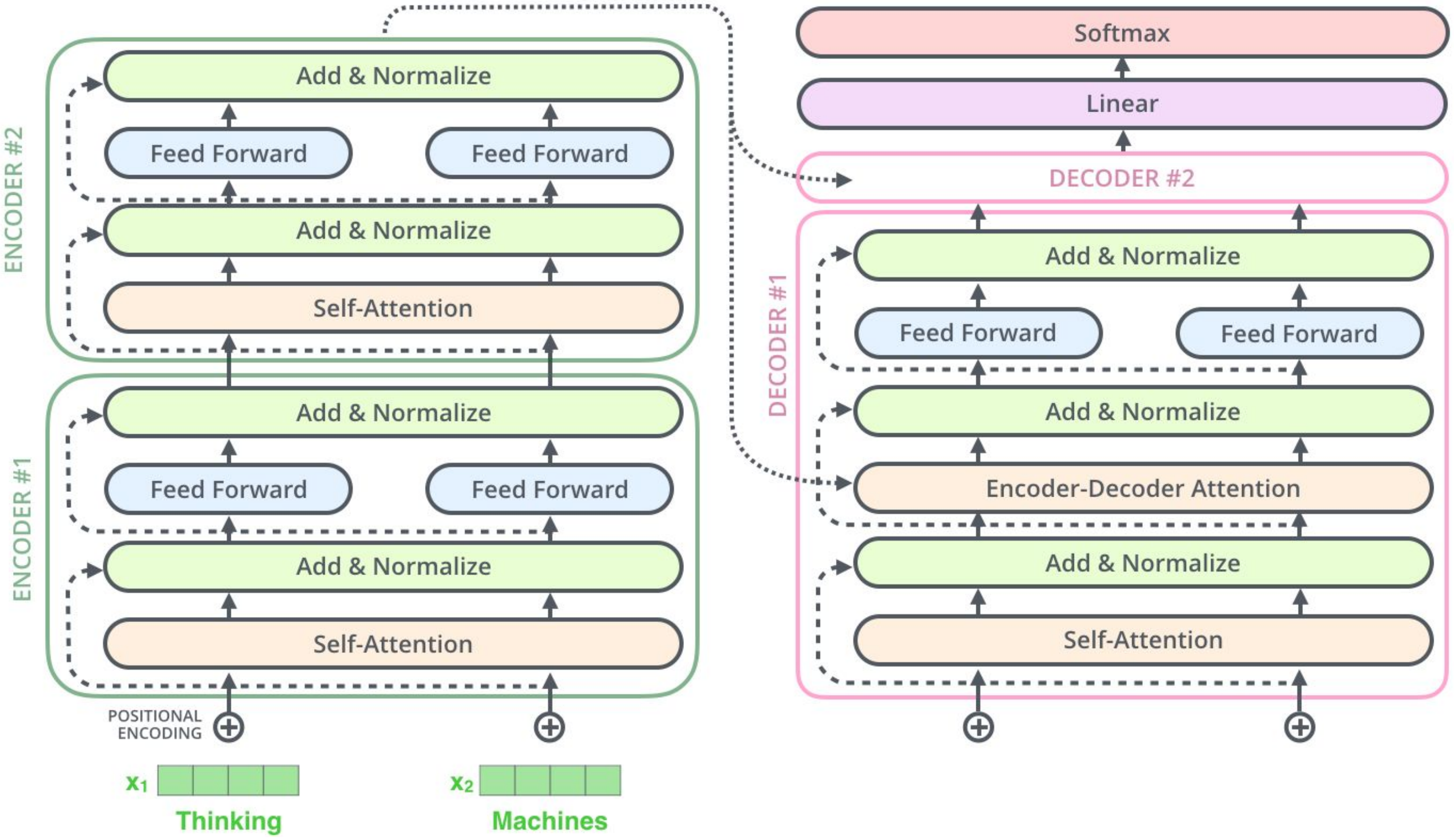


Add to each Input vectors (word embedded) a **positional encoding**

Transformer - Residual connexions for the “Add & Normalize” Layer



Transformer - Workflow



Transformer - Key ideas to keep in mind

- The Transformer is build with Encoders and Decoders, without any RNN nor CNN but solely with Attention.

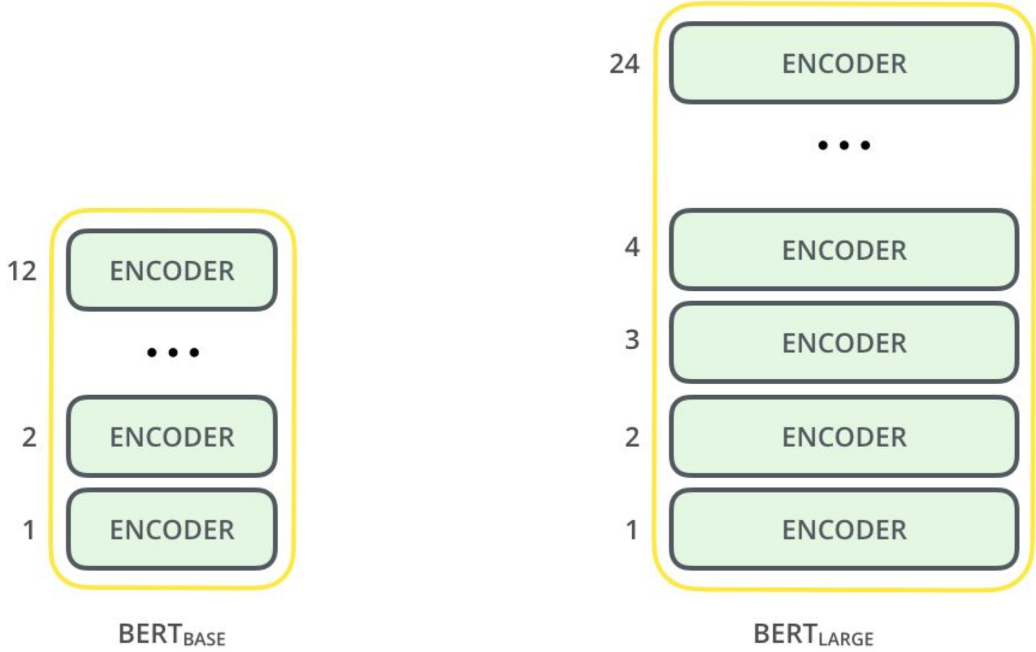
Multi-Head Self-Attention + Positional Encoding

= Context influence + Order in sequence = Sequence Representation

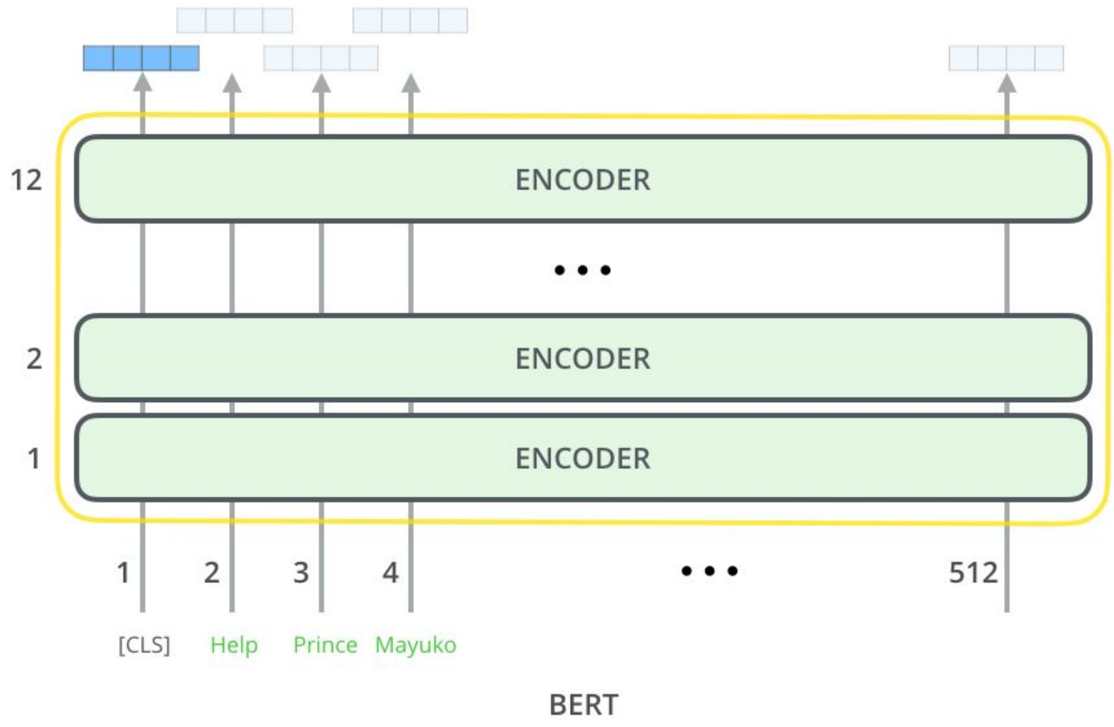
- Attention mechanism solve the bottleneck problem by allowing the decoder to look directly at the source.

Back to BERT

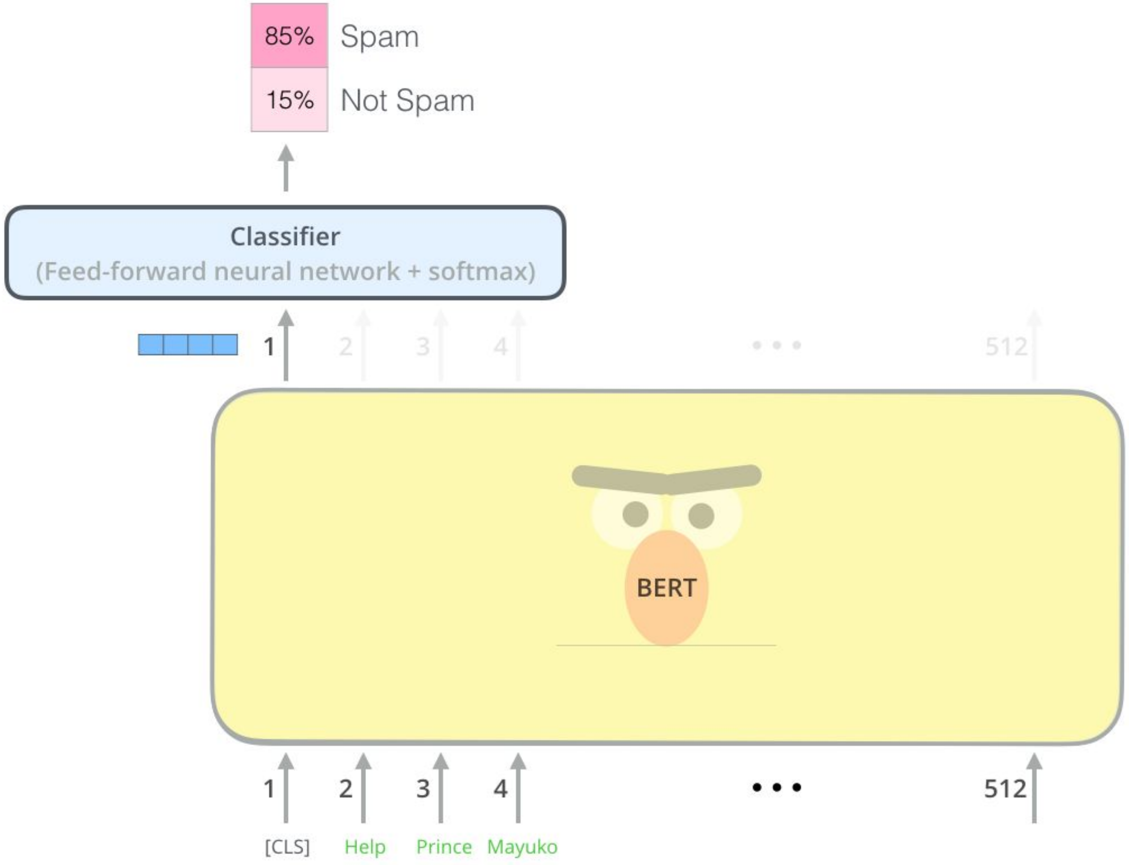
BERT architecture - Big Trained Transformer Encoder Stack



BERT - Input & Output



BERT - Fine-Tuning layer for classification (Spam or Not Spam)



Bert - Pre-training a deep bi-directional language representation

Masked LM:

Each sentences, 15% words replace by [MASK]

problem: In reality the [MASK] doesn't Exist:

Solution: the N% selected words

- 80% of time become the Token [MASK]
- 10% replaced by other Token
- 10% Token not change

PPS:

Sentence pair combination A&B.

In the Training set, 50% are good combinaison, other Random Combination of same corpus

Inject 2 Tokens

- <CLS> Start of A
- <SEP> End each sentence

Compute the probability that A&B is good combinaison.

Bert - Demo Time

Questions ?

Further readings

- Fine-Tuning
  - Semi-supervised Sequence Learning : Paper introducing improvement in sequence learning with RNN
- Transformers
  - Attention Is All You Need : Paper introducing the Transformer
  - The Annotated Transformer : Step by step annotated in python of “Attention in all you need”
- BERT
  - Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing : Google blog post announcing BERT release
  - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding : The paper
  - The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) : A good blogpost ;)