Law & Tech Lab, Maastricht University

The Transformer:

An illustrated explanation of the model architecture

Antoine Louis

November, 2020

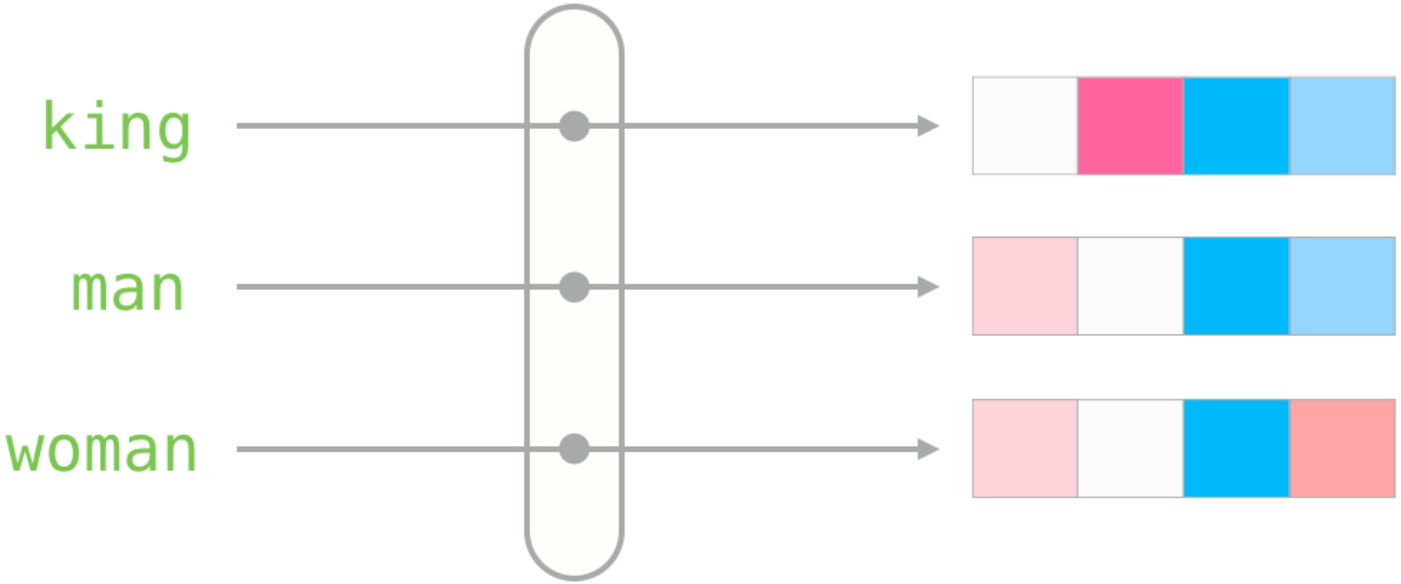
Outline

Introduction

A well-known NLP task

NB: Example of a text classifier, which takes a movie review as input and outputs a predefined sentiment class (positive, negative, or neutral).

Word embedding

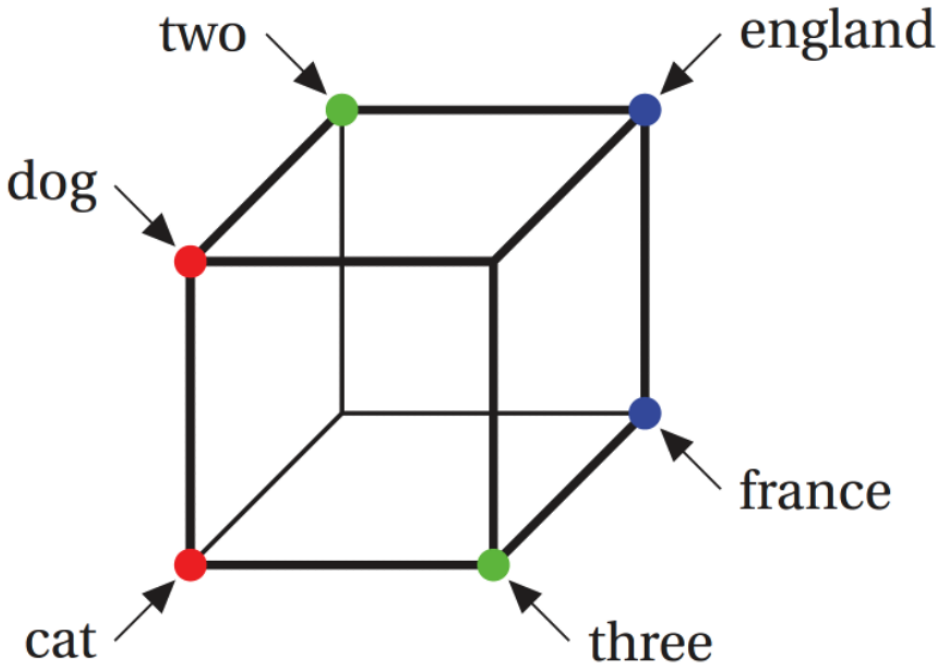


NB: Word embeddings are representations of textual data as low dimensional vectors. This transformation is necessary as deep neural networks require their input to be vectors of continuous values (they cannot work on plain text).

One-hot encoding

dog	1	0	0	0	0	0
cat	0	1	0	0	0	0
two	0	0	1	0	0	0

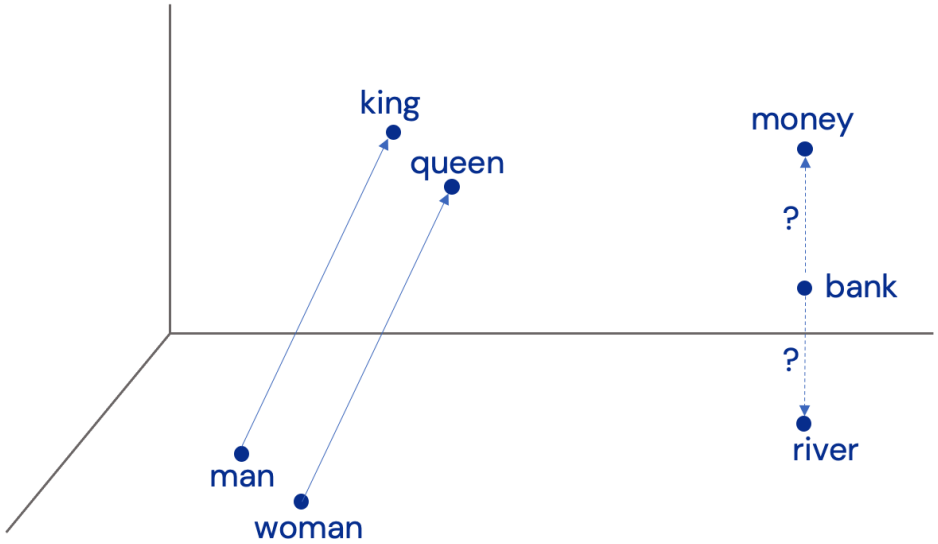
...



NB: Given a vocabulary of N words, one-hot encoding consists in assigning an integer index $i \in \{1, \cdots, N\}$ to each word such that the word is represented as a N-dimensional sparse vector mostly composed of zeros, except for a single entry at the position corresponding to the word's index in the vocabulary that takes the value 1. In addition to suffering from an obvious feature size downside, this approach does not take similarities between word into account.

Learning good word embeddings

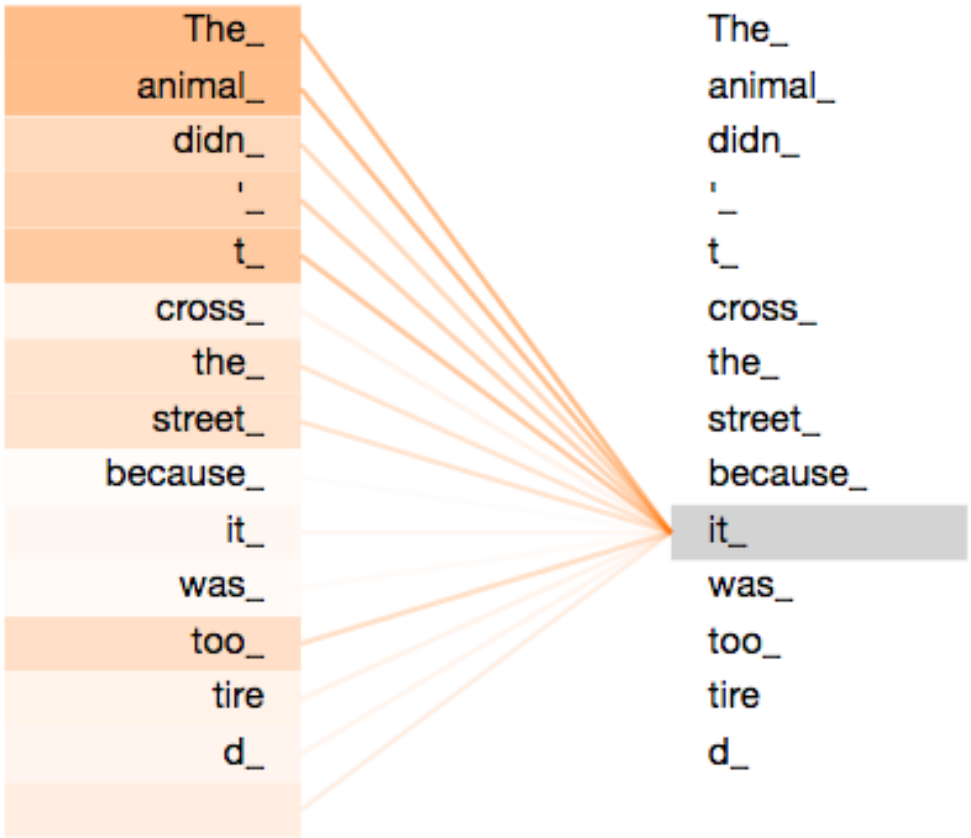
> *"You shall know a word by the company it keeps."*
>
> - J.R. Firth (1957)



NB: Well-known and largely used word embedding methods such as **word2vec** (Mikolov et al., 2013), **GloVe** (Pennington et al., 2014) and **fastText** (Bojanowski et al., 2016), all have a major weakness: they create a fixed embedding for each word. Therefore, whatever the meaning of the word, it will always be mapped to the same vector.

Self-attention

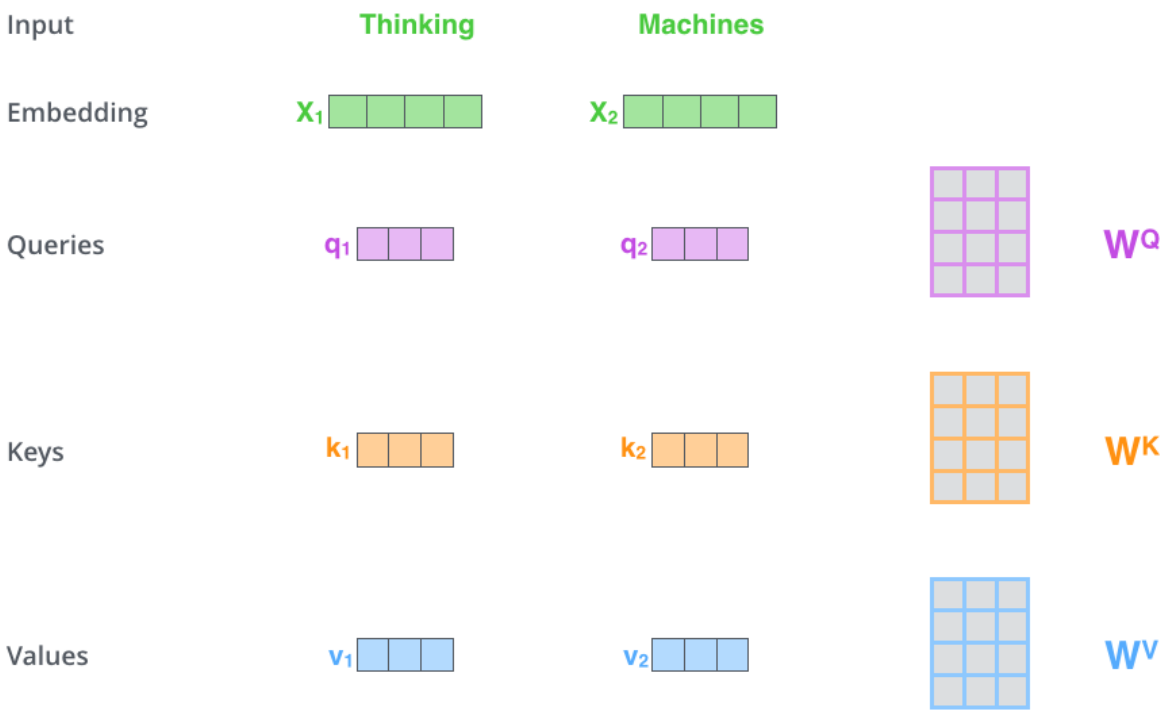
Self-attention: what it does



NB: As the model processes a certain word from the input sequence, self-attention allows the model to attend to all the other words from the sequence for clues to a good contextual representation of that word.

Self-attention: how it works

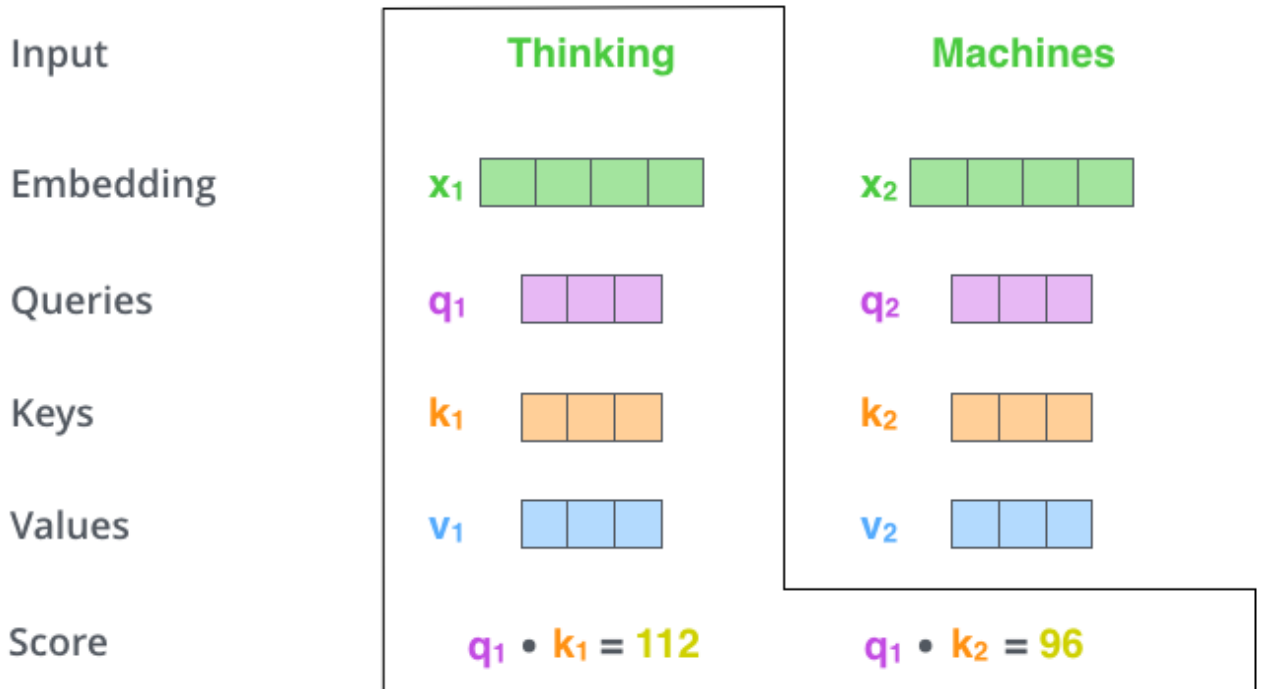
- **Step 1**: Create a **query**, **key**, and **value** vectors for each input word.



NB: These vectors are obtained by multiplying an initial word embedding (learned during training) by three distinct weight matrices W^Q, W^K, W^V (also learned during training).

Self-attention: how it works

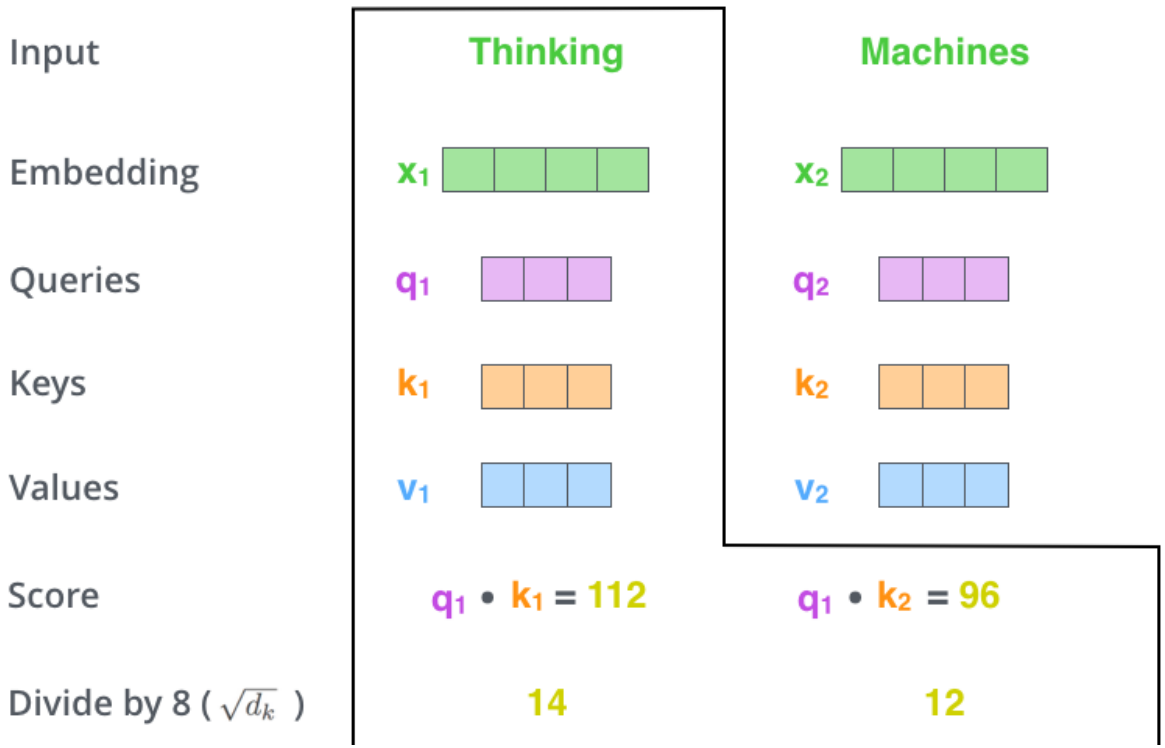
- **Step 2**: Given a word of the input sequence, **score** its query vector with the key vectors of all other words.



NB: Scores are calculated by computing the **dot product** of the query-key vectors.

Self-attention: how it works

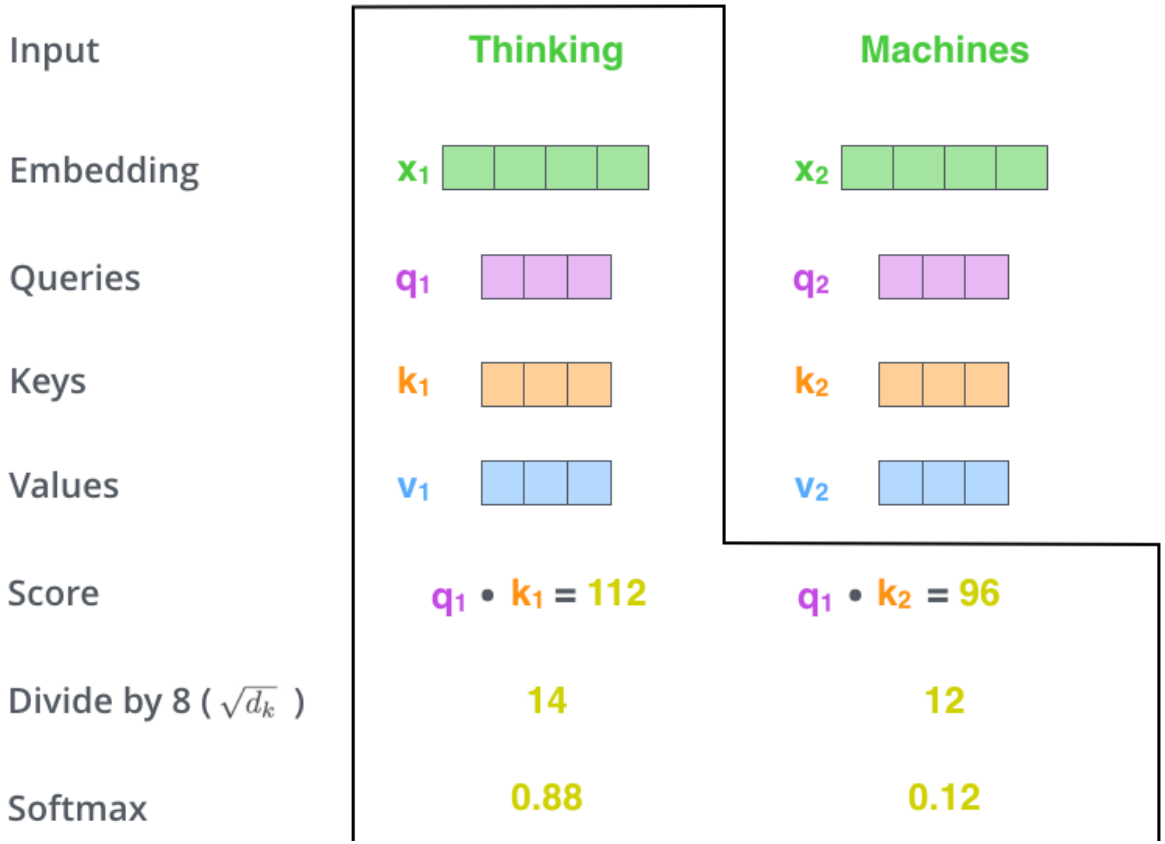
- **Step 3**: **Divide the scores** by the square root of the key vector dimension.



NB: This division leads to more stable gradients.

Self-attention: how it works

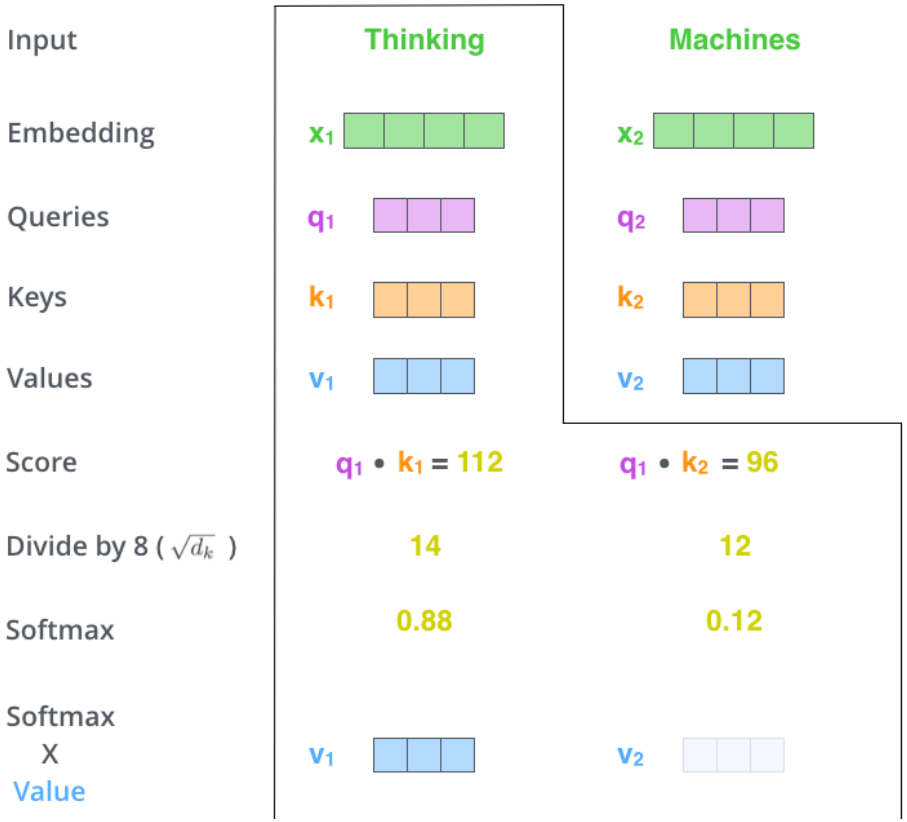
- **Step 4**: Pass all the scores through a **softmax** operation.



NB: Softmax normalizes the scores so they are all positive and add up to 1.

Self-attention: how it works

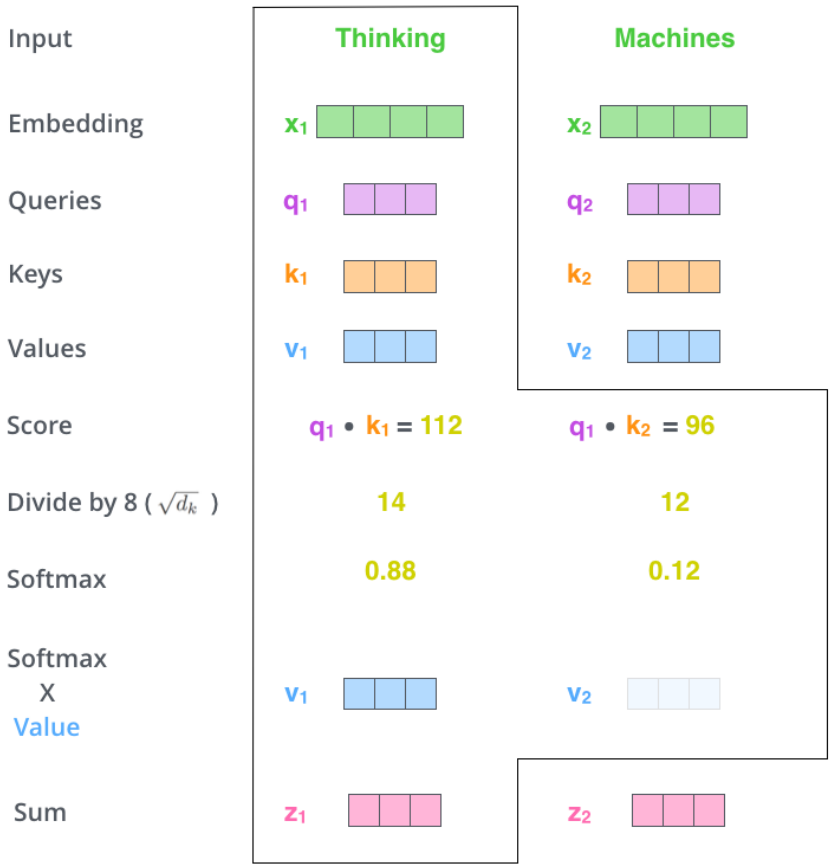
- **Step 5**: Multiply each **value vector** by its softmax score.



NB: The intuition here is to keep intact the value vectors of the word(s) the model decides to focus on, and drown out those of irrelevant words.

Self-attention: how it works

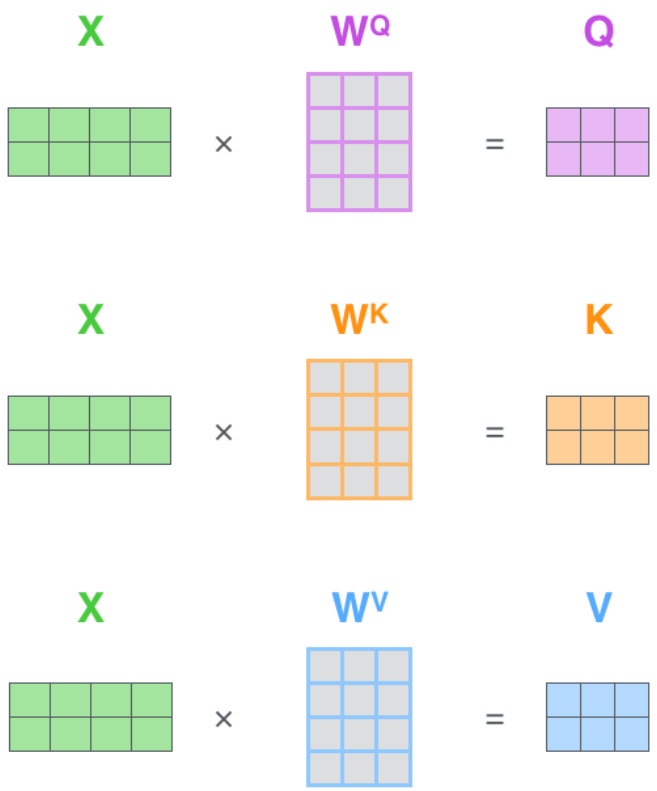
- **Step 6**: **Sum up** all weighted value vectors.



NB: This produces the output vector of the self-attention calculation for a given word in the input sequence.

Self-attention: matrix form

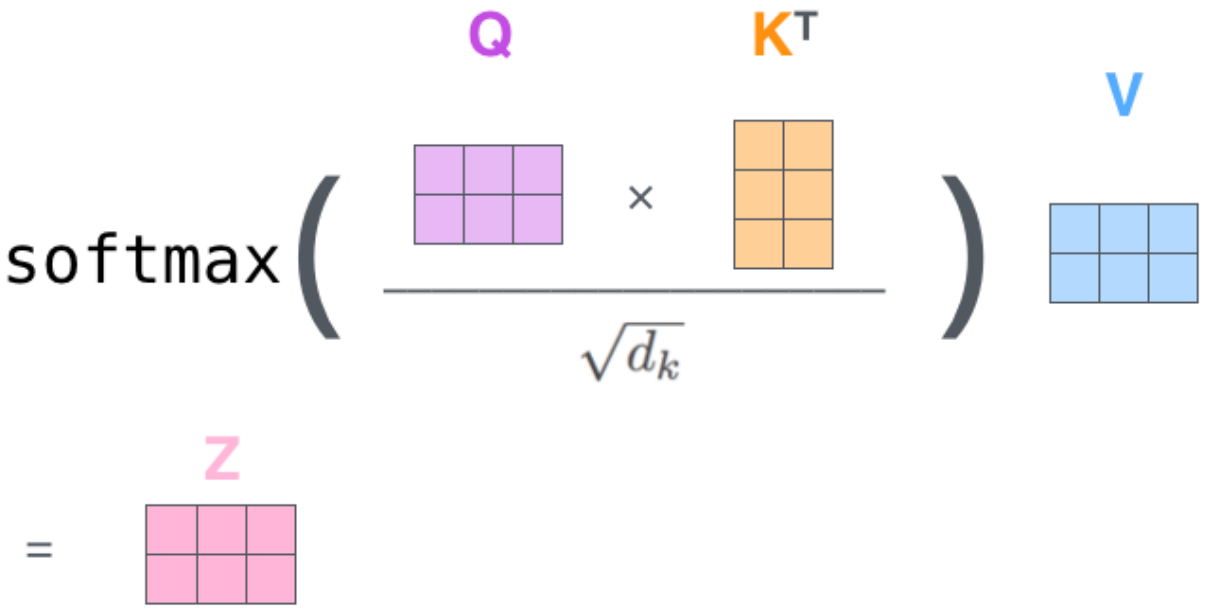
- The first step is to calculate the **Query**, **Key**, and **Value** matrices.



NB: The **Query**, **Key**, and **Value** matrices are computed by packing the input embeddings into a matrix $\boldsymbol{X}$ and multiplying it by the respective weight matrices $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, $\boldsymbol{W}^V$.

Self-attention in matrix form

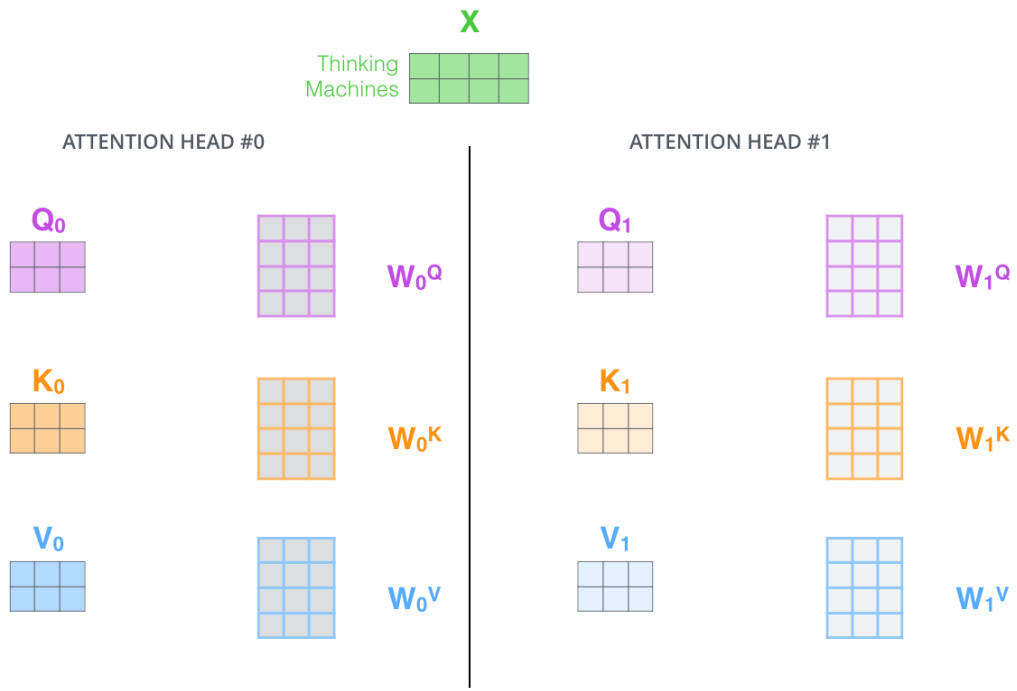
- Finally, steps 2 to 6 can be condensed in one formula.



NB: Self-attention calculation in matrix form.

Multi-head attention

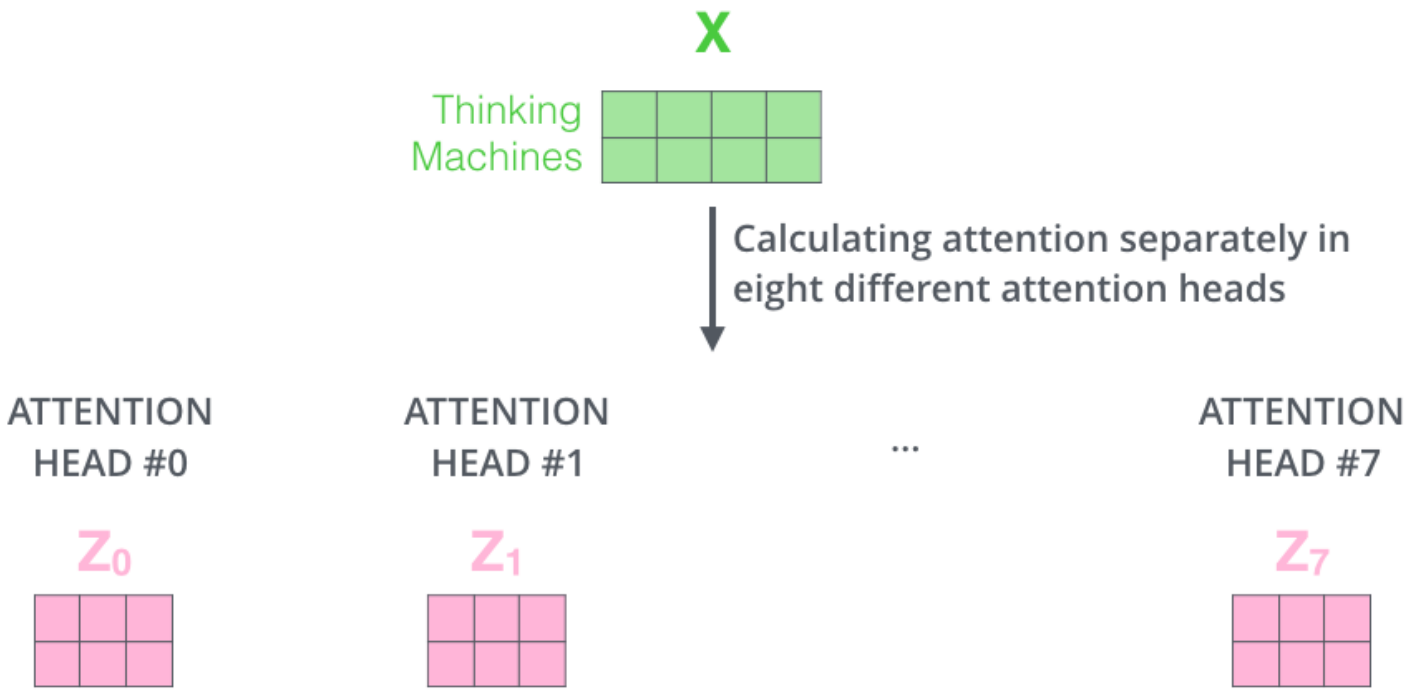
- In the Transformer, the self-attention computation is performed **several times** on each input word using different sets of Query/Key/Value weight matrices.
- Each of these sets is called an **attention head** and the whole mechanism is called **multi-head attention**.



NB: Separate W^Q, W^K, W^V matrices are maintained for each attention head.

Multi-head attention

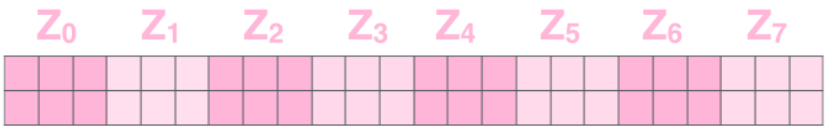
- Hence, the Transformer uses **8** attention heads.



NB: After the different self-attention calculations, it results 8 different output matrices Z_i.

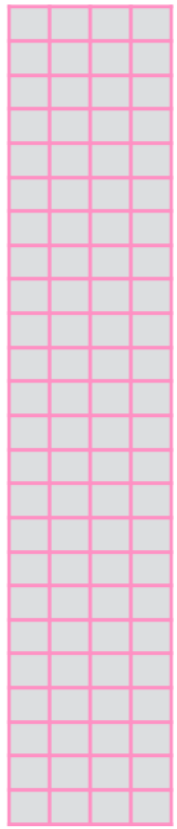
Multi-headed attention

1) Concatenate all the attention heads



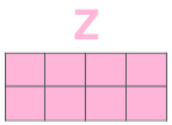
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



NB: The final output from the self-attention layer is obtained by concatenating the different Z_i matrices and multiplying them by a weight matrix W^O.

Multi-headed attention

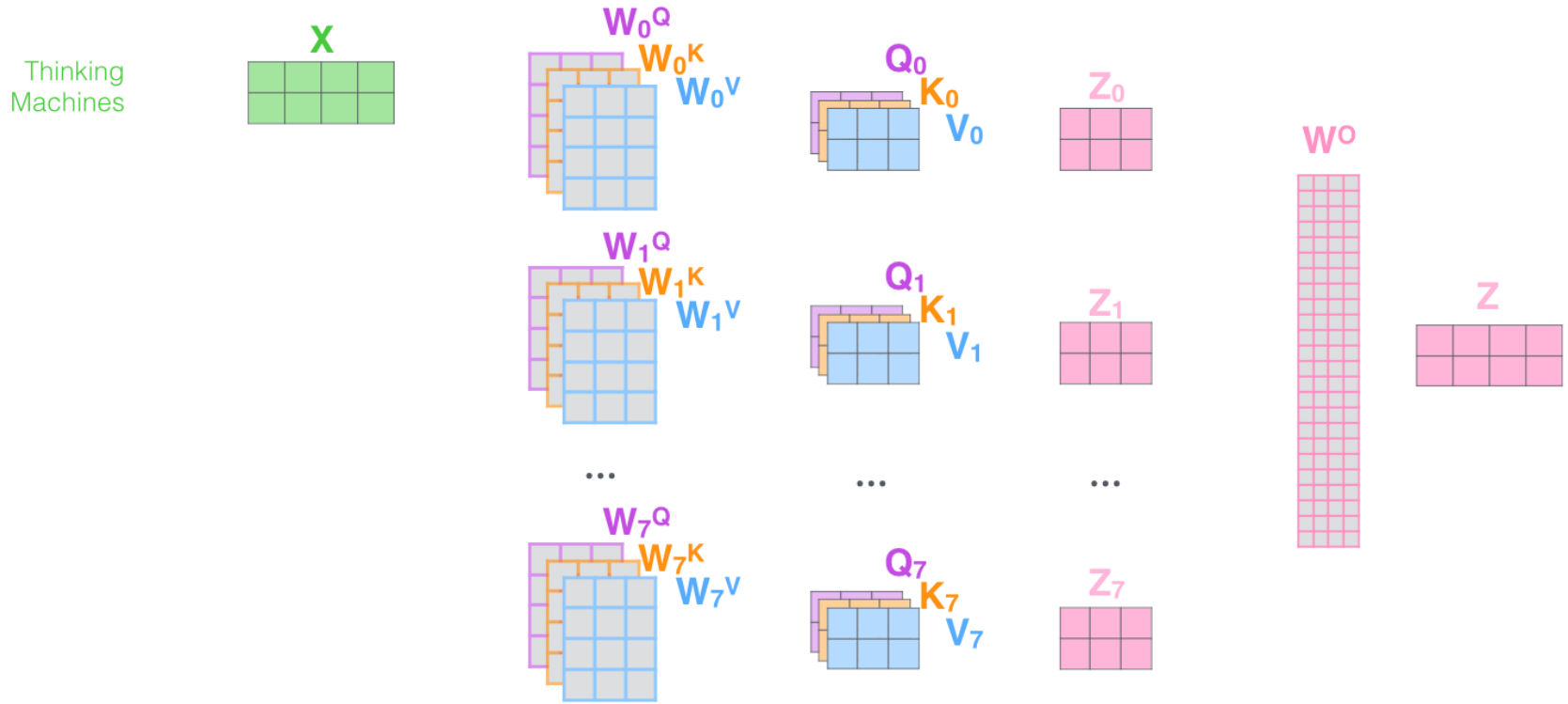
1) This is our input sentence*

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

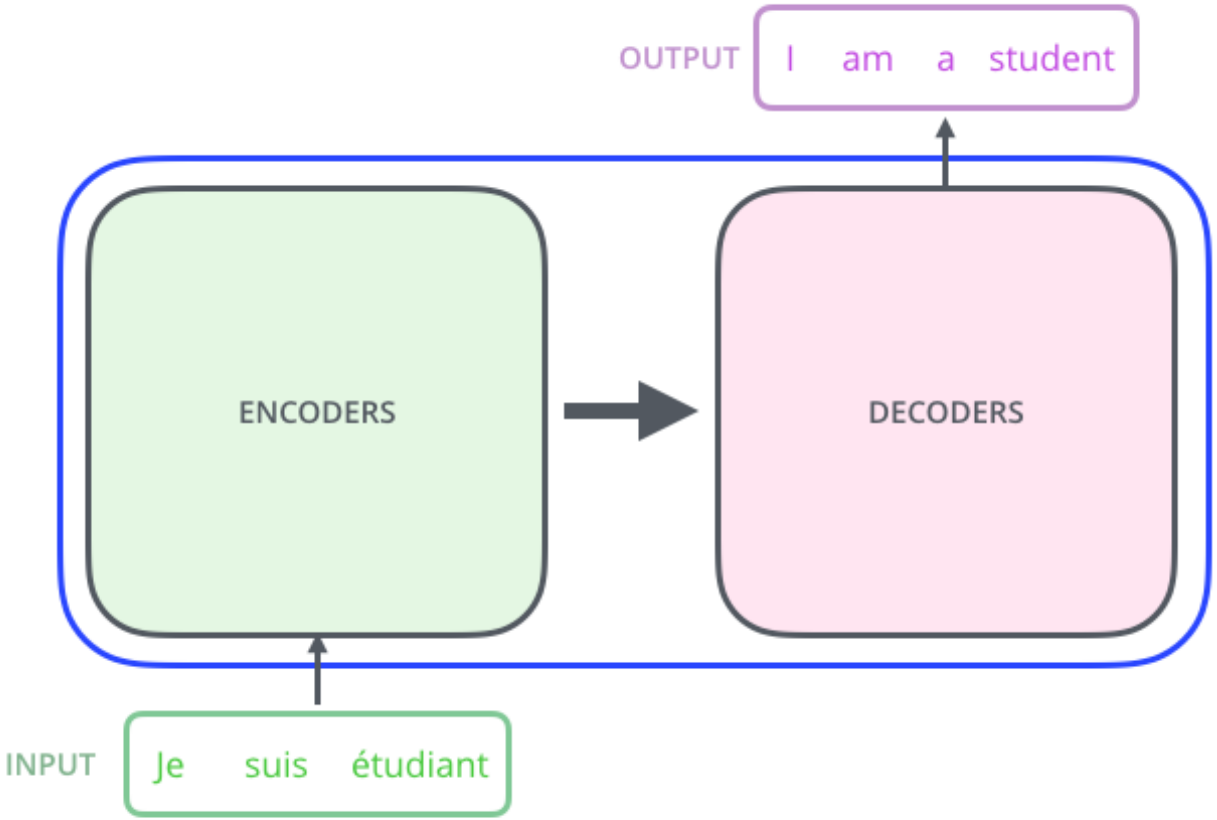
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



NB: Multi-head attention computation.

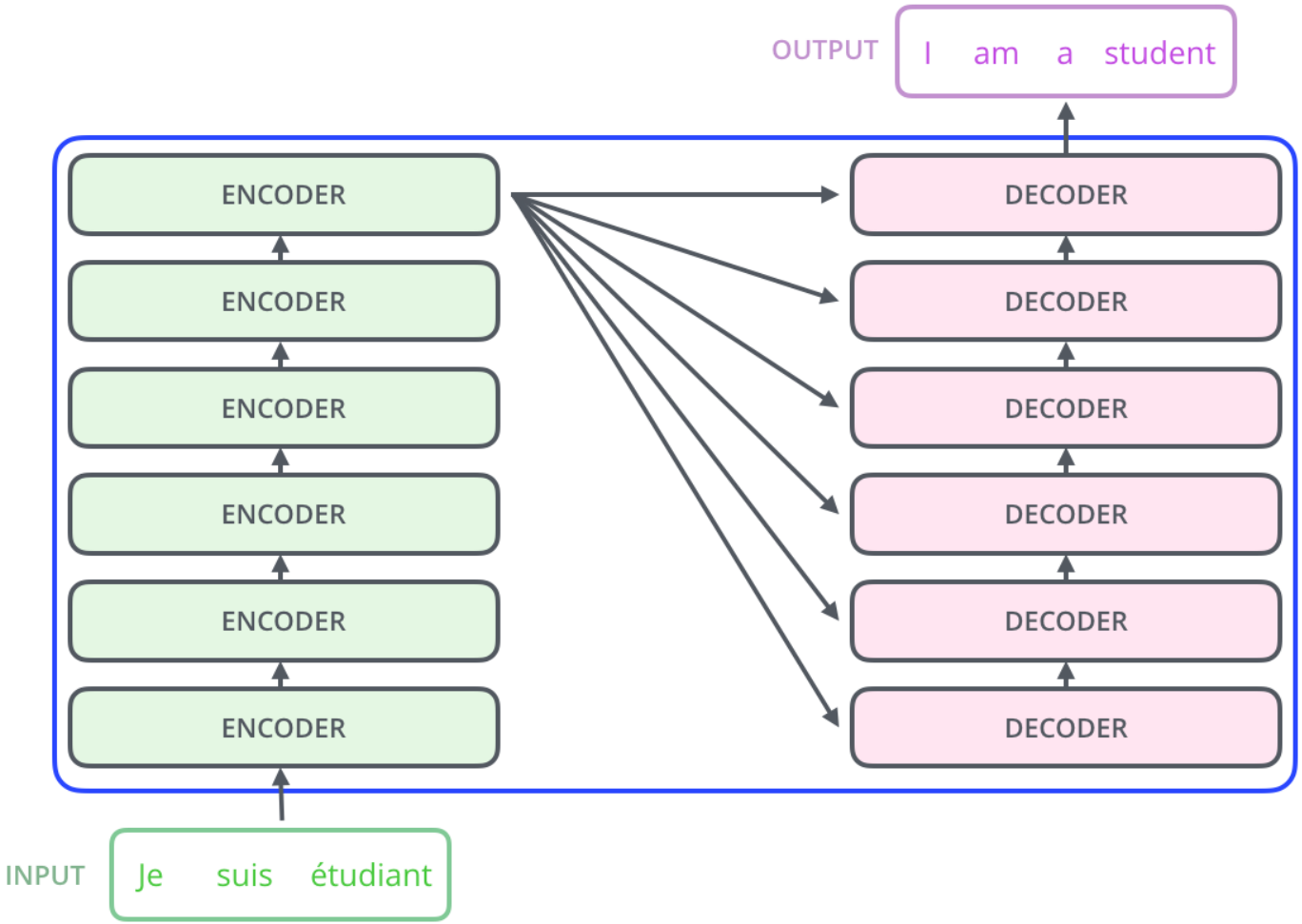
Transformer (Vaswani et al., 2017)

Architecture



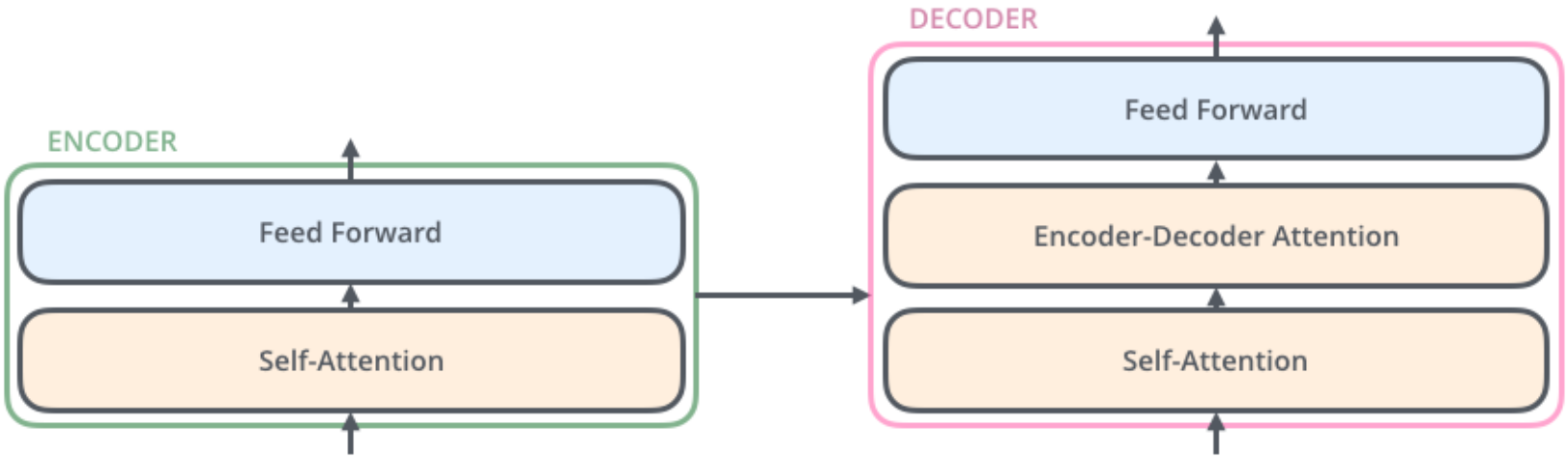
NB: The **Transformer** architecture is composed of an **encoder** and a **decoder**.

Architecture



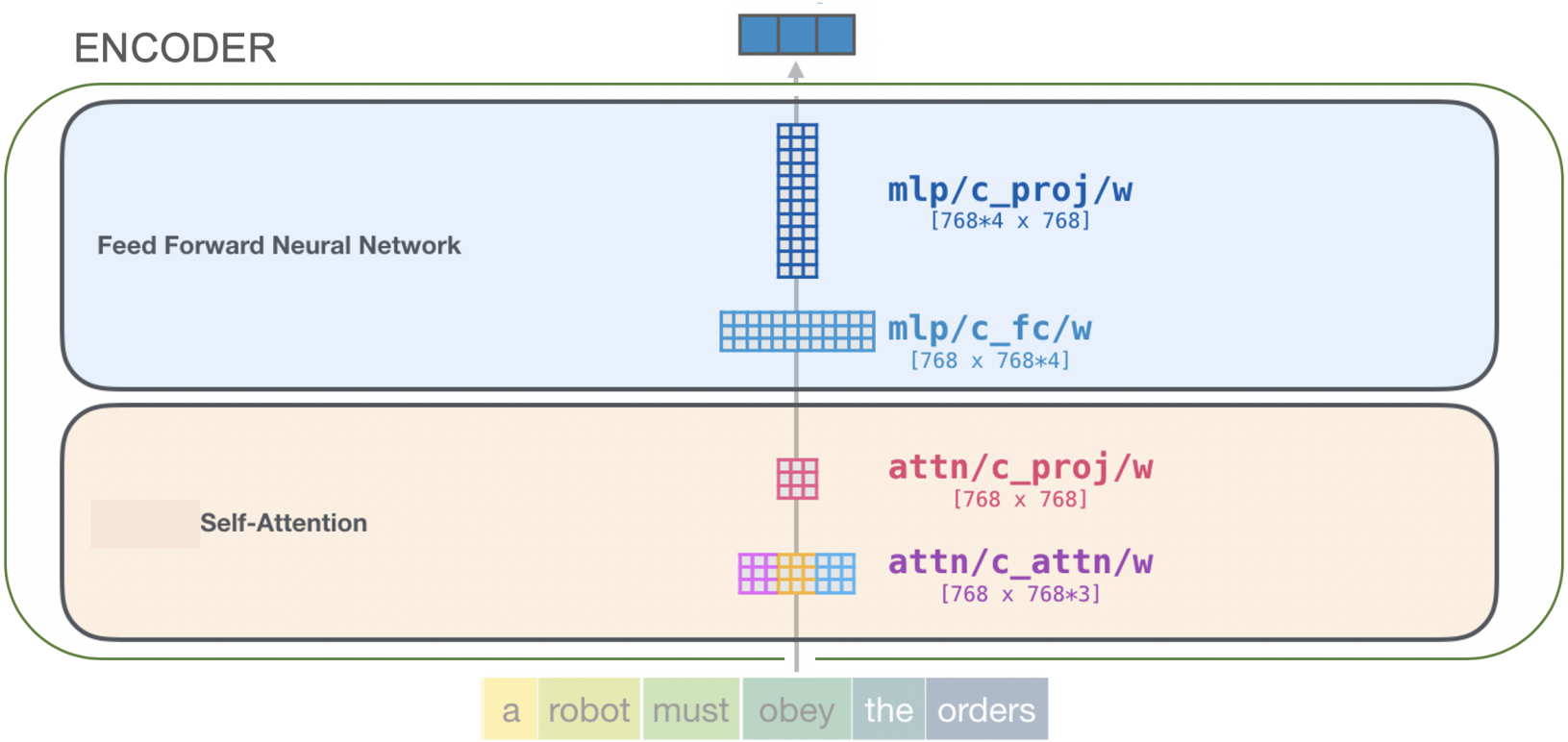
NB: Both the encoder and decoder are stacks of several encoding and decoding layers respectively, all identical in structure but not sharing the same weights.

Architecture



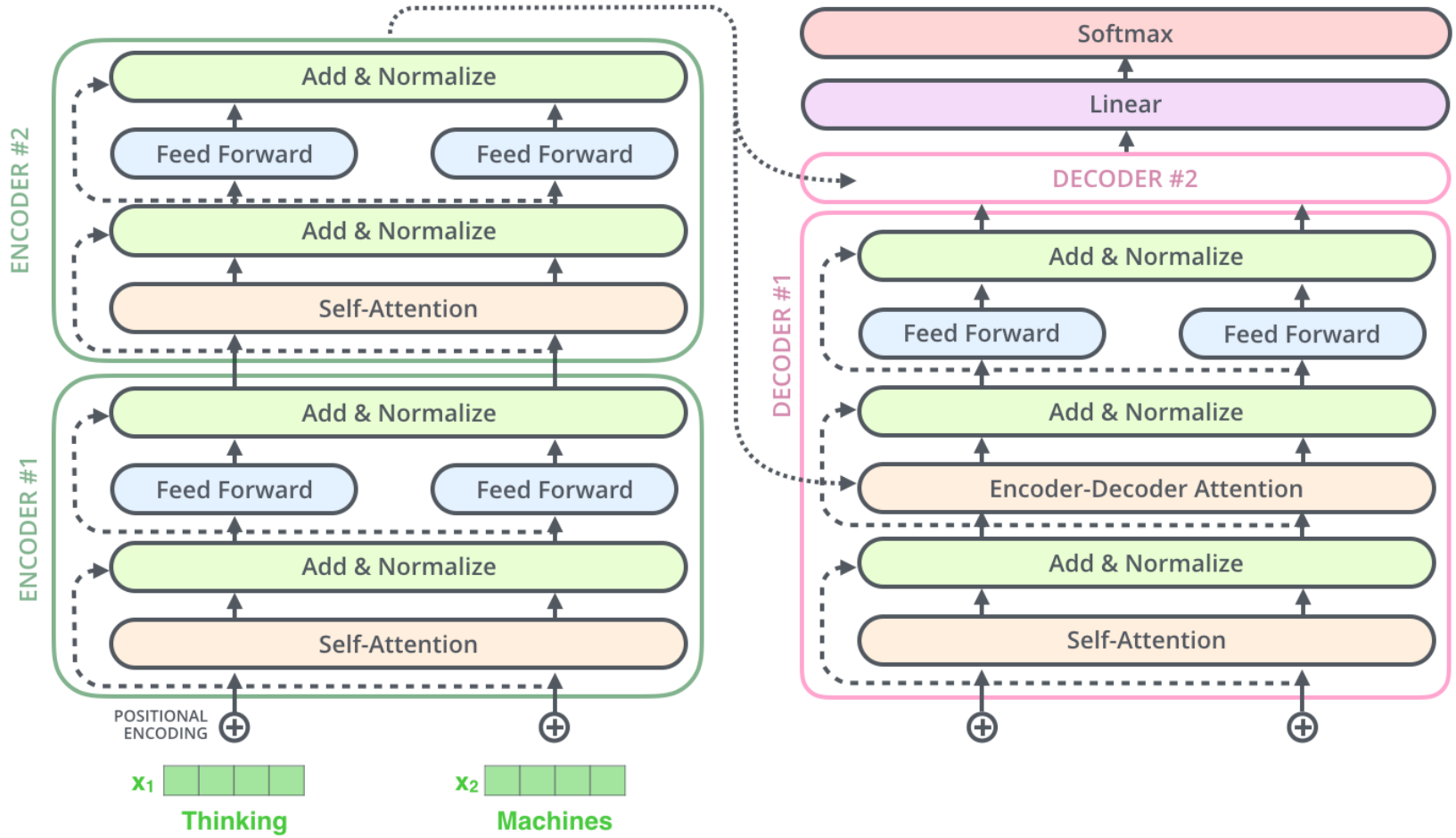
NB: Each encoder layer contains two types of sub-layers: (1) a **self-attention layer** that helps look at other words in the sequence when encoding a given word, and (2) a shallow **feed-forward layer** that is independently applied to each word vector, allowing the various paths to be executed in *parallel*.
Each decoder layer contains an extra sub-layer in-between: (3) an **encoder-decoder attention layer**, which helps focus on relevant parts of the initial input sequence.

Model parameters



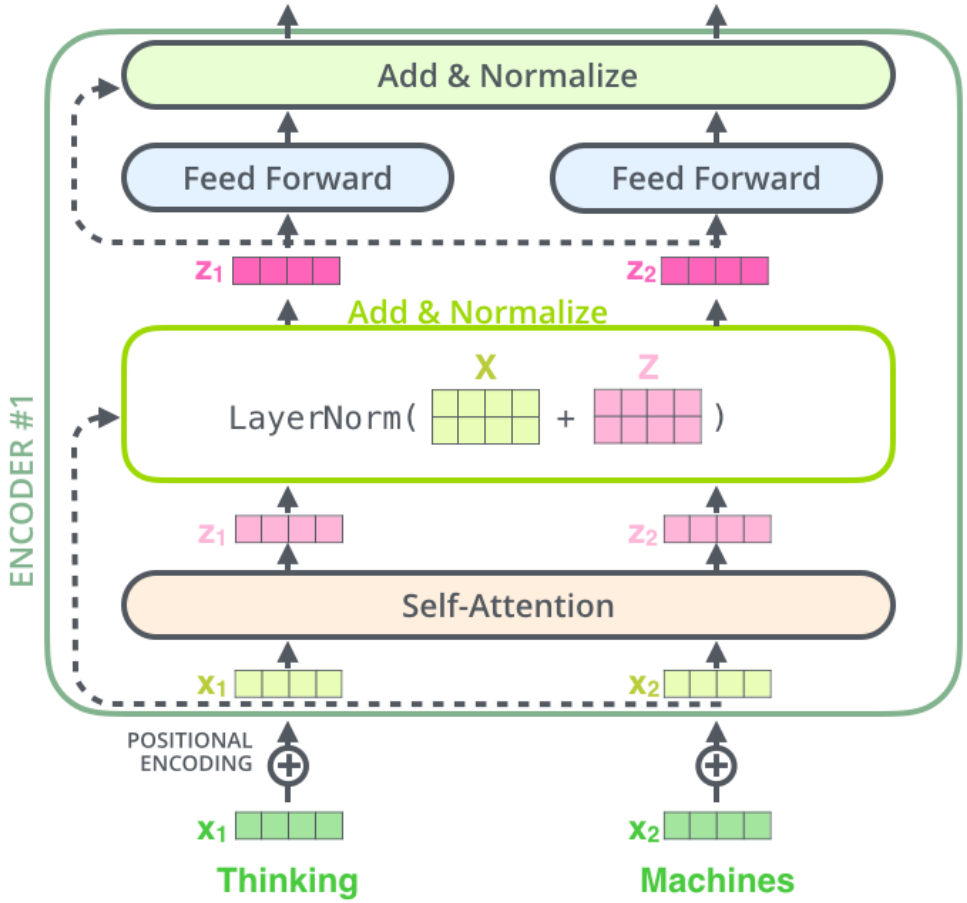
NB: Every block in a transformer has its own weights. (i) the weight matrix used to create the queries, keys, and values (**attn/c_attn/w**); (ii) the weight matrix that projects the results of the attention heads into the output vector of the self-attention sub-layer (**attn/c_proj/w**); (iii) the weight matrix corresponding to the first layer of the Feed Forward Neural Network (**mlp/c_fc/w**); (iv) the weight matrix corresponding to the second layer of the Feed Forward Neural Network (**mlp/c_proj/w**).

Residual connections



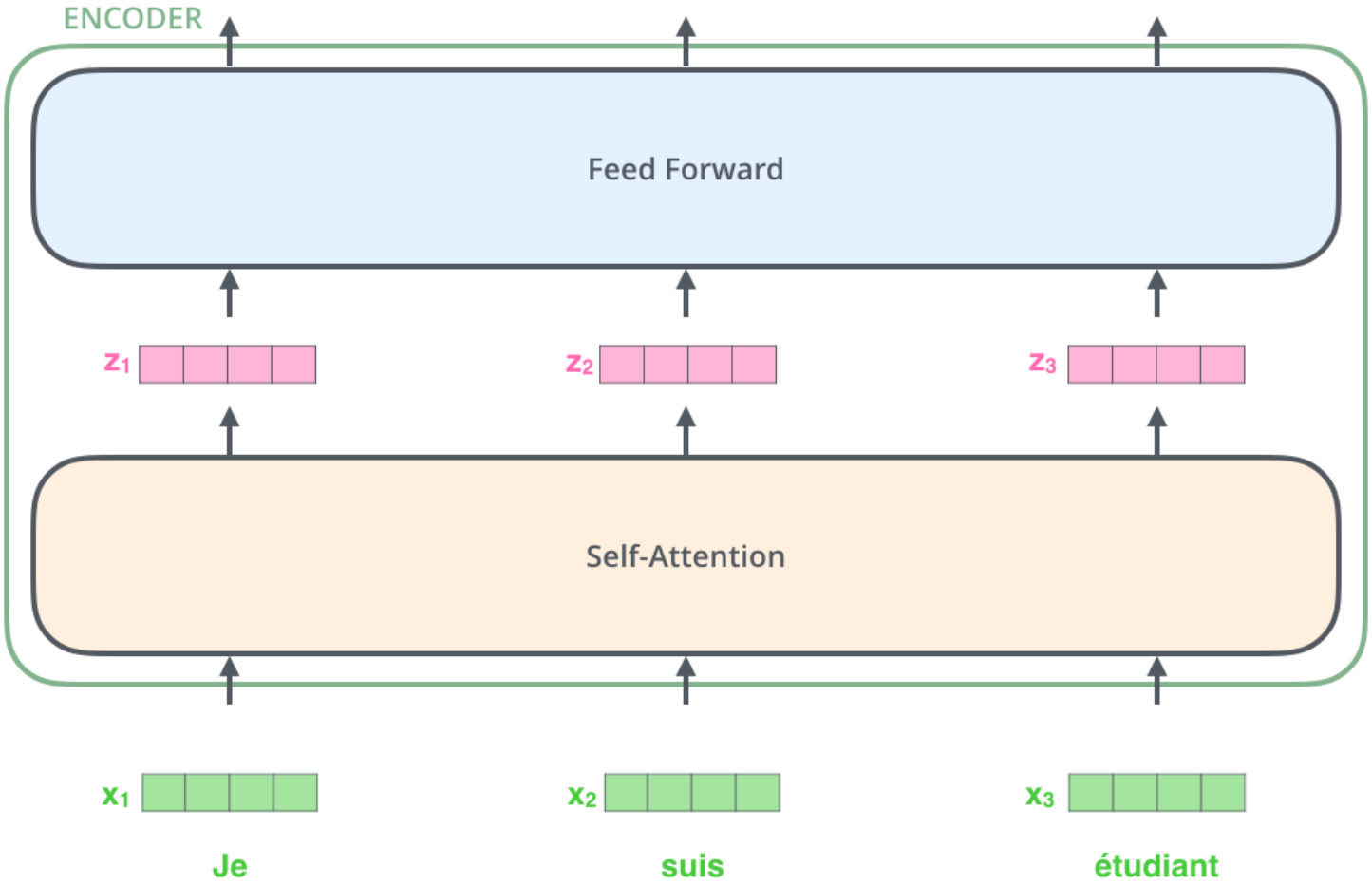
NB: Each sub-layer in a encoder/decoder layer has a **residual connection** around it, followed by a **layer-normalization** step.

Residual connections



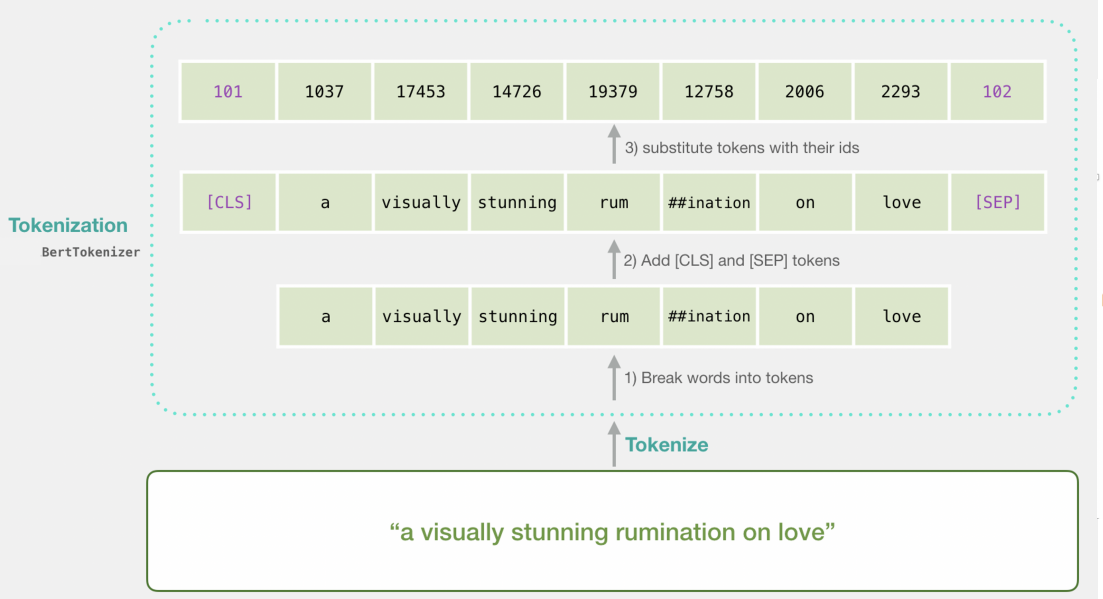
NB: After each sub-layer, the output matrix ***Z*** is added to the input matrix ***X*** and the result is normalized.

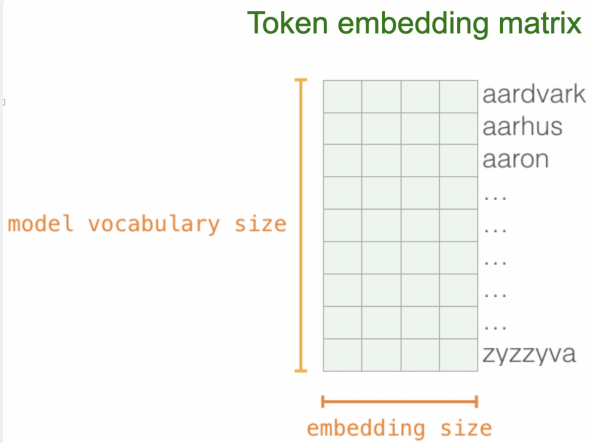
Encoder inputs



NB: Before being given to the encoder, each word must be converted to an input vector.

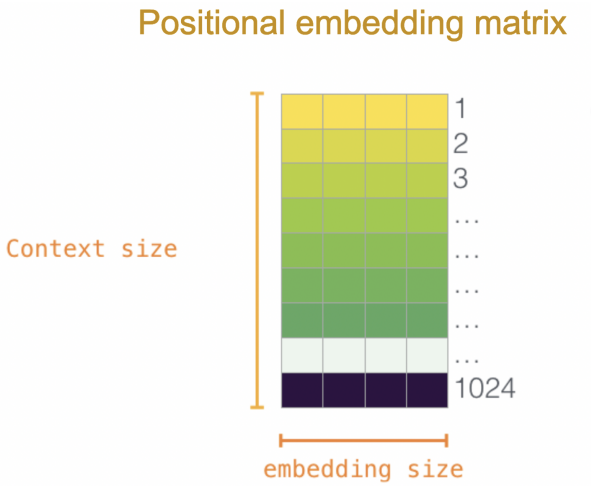
Tokenization

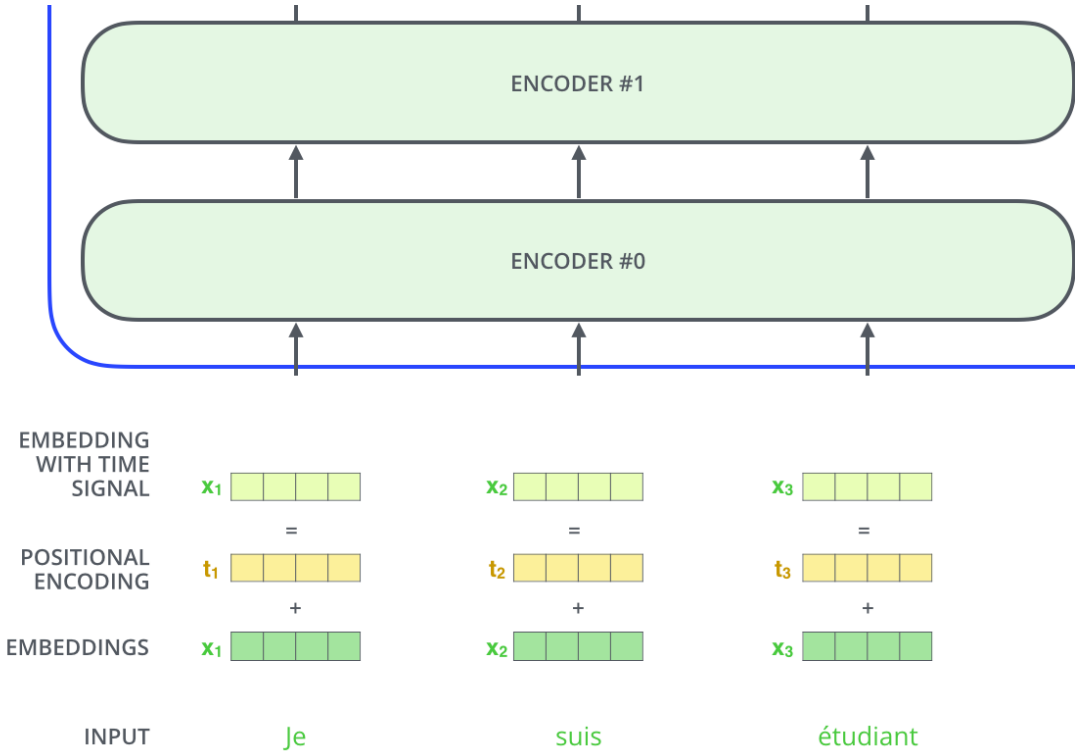




NB: The Transformer uses a data-driven **tokenization** method that creates a **fixed-size** vocabulary of individual characters, sub-words and words that best fits a given language corpus.
For example, the *WordPiece* tokenization model first checks if the full word is in the vocabulary. If not, it tries to break it down into the largest possible sub-words from the vocabulary. As a last resort, it decomposes the word into individual characters.

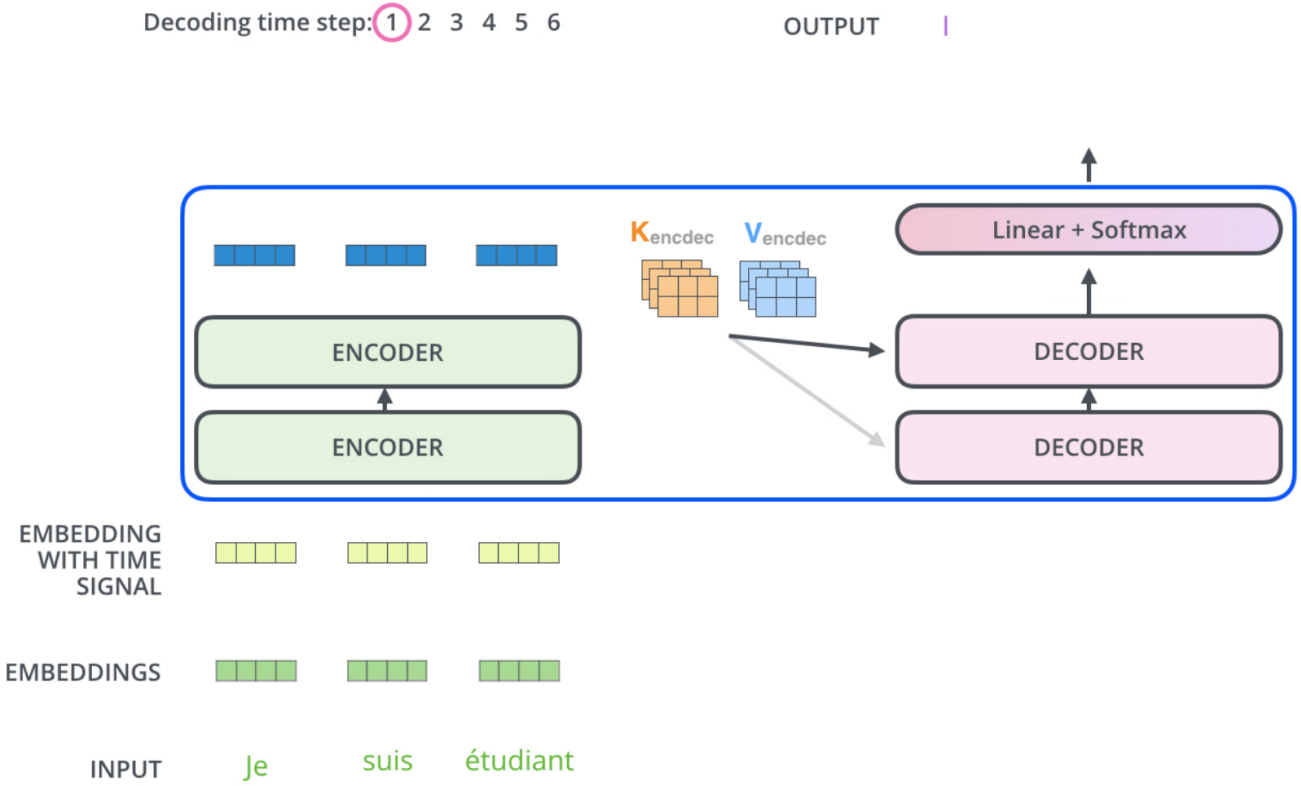
Positional encoding





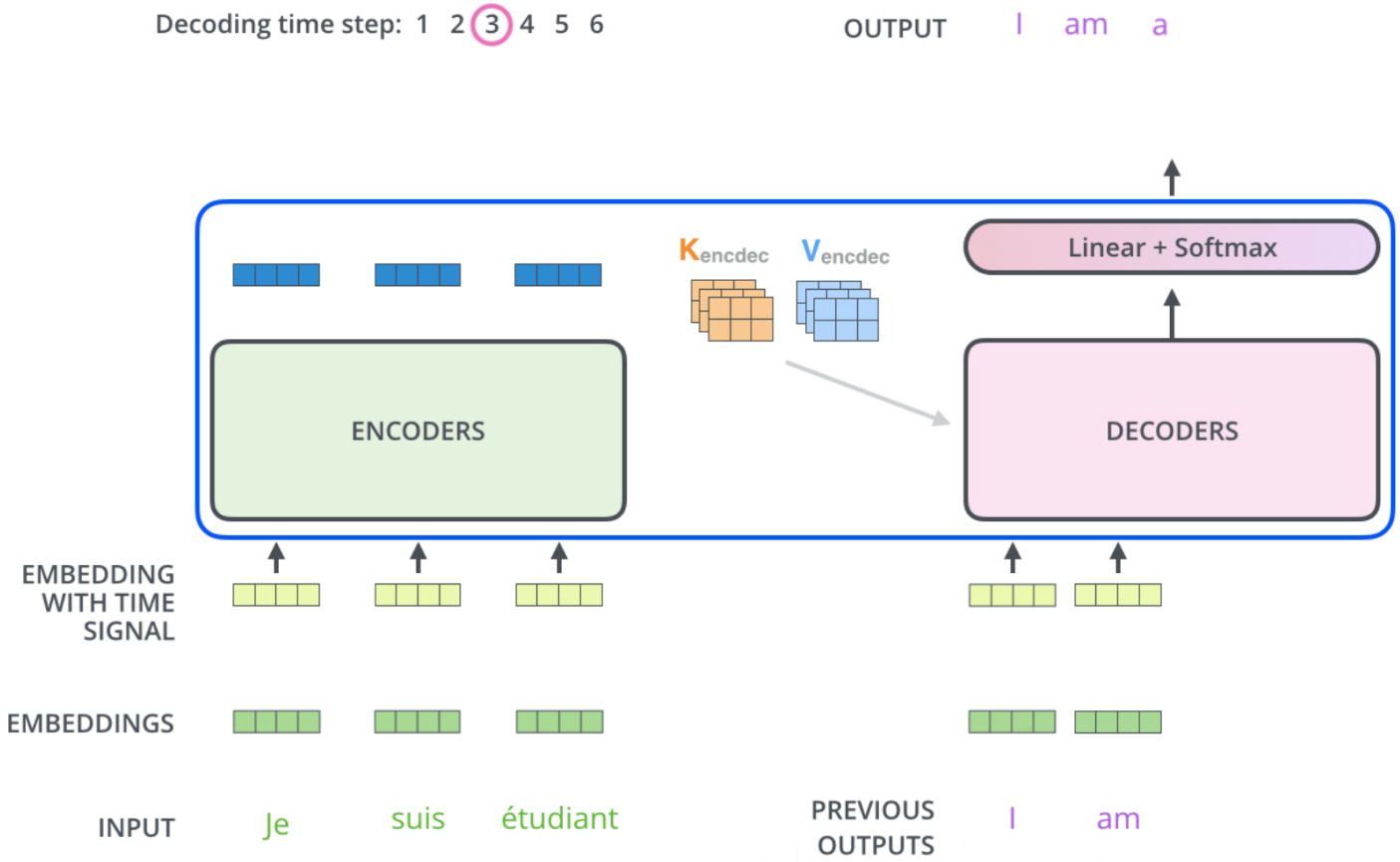
NB: To give the model a sense of order of the input words, **positional encoding** vectors are added to the token vectors. This results in the final input embedding that goes to the encoder.

Decoder



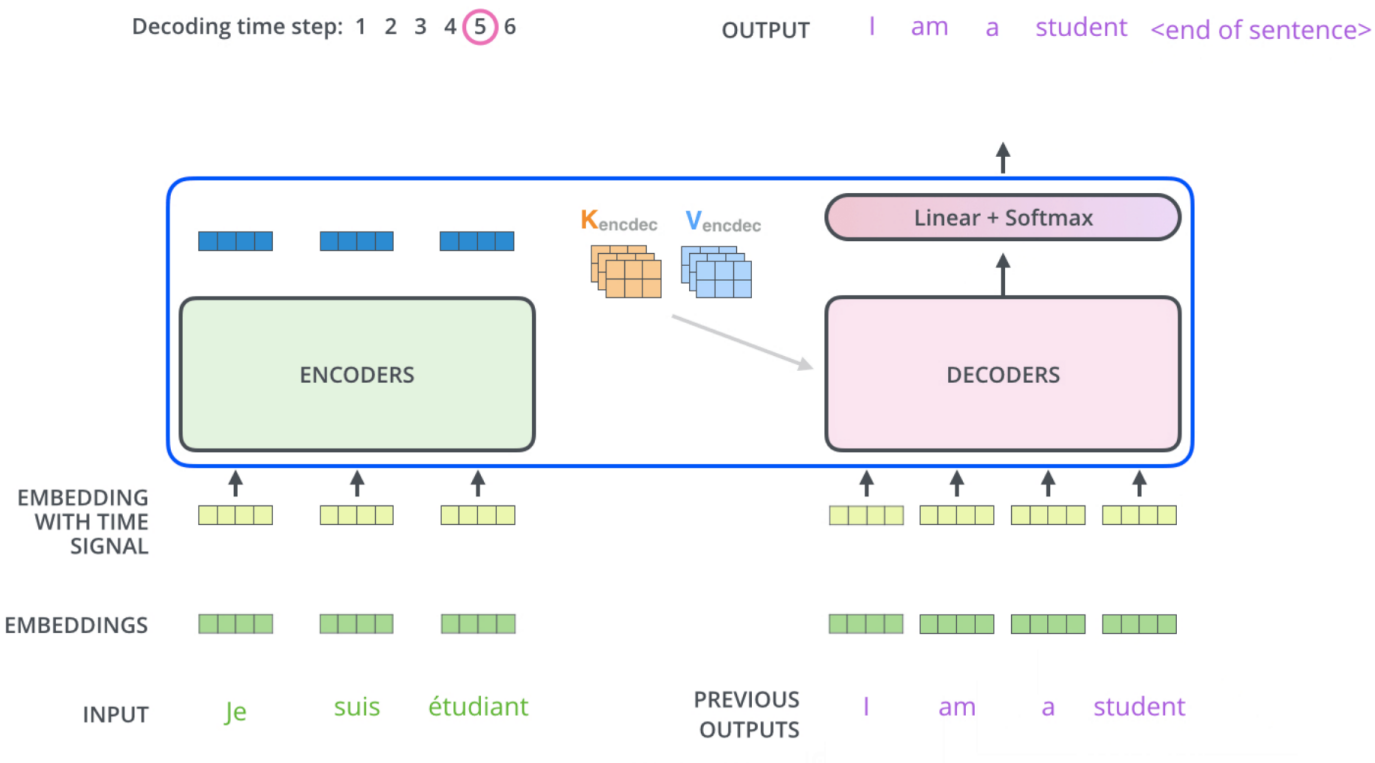
NB: The output of the last encoder layer is transformed into a set of attention vectors ***K*** and ***V*** (result of the multiplication of the FFN output matrix ***R*** with the weight matrices Q^K and Q^V). Matrices ***K*** and ***V*** are used by each decoder layer in its **encoder-decoder attention** sub-layer, which works just like the self-attention sub-layer except it creates the query matrix $\boldsymbol{Q}^V$ from the sub-layer below it and takes the key and value matrices from the output of the encoder stack. This sub-layer helps the decoder focus on appropriate positions from the initial input sequence.

Decoder



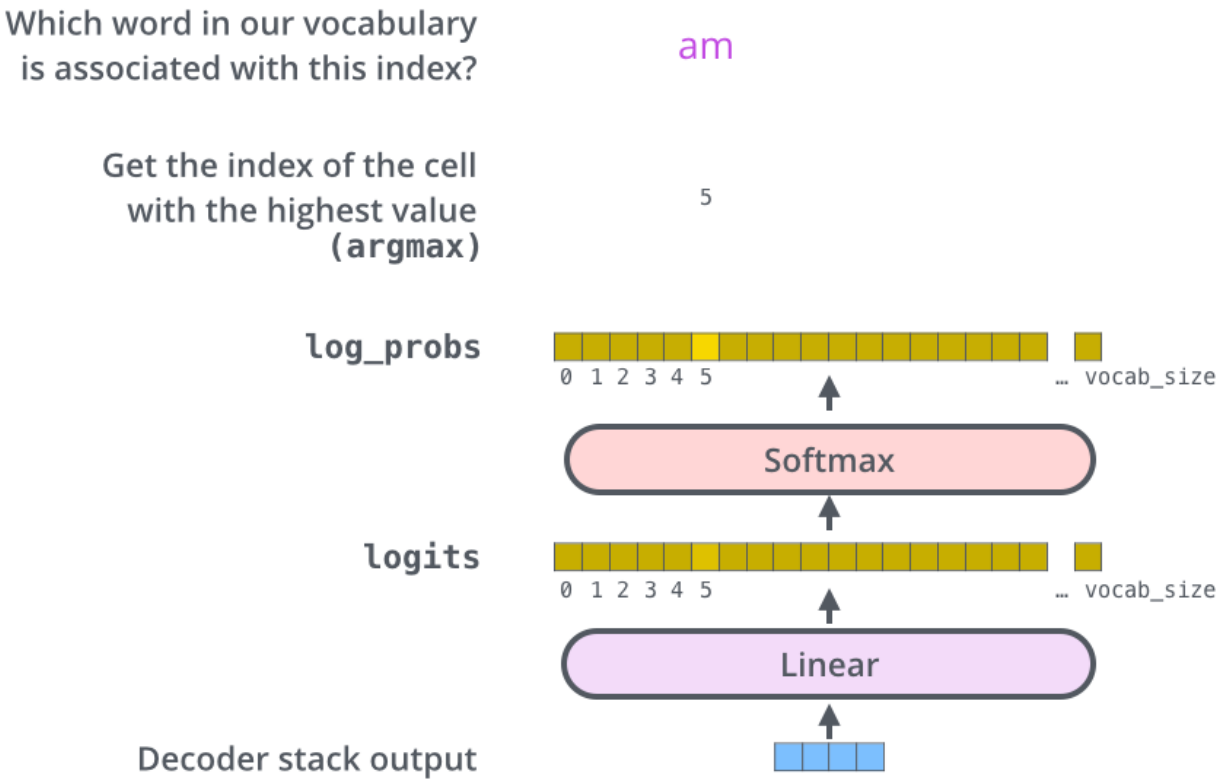
NB: The decoder output of each step is fed back to the decoder as input for the next time step. Note that the **self-attention** sub-layer in the decoder is only allowed to attend to earlier positions from the input sequence. This is done by masking future positions (setting them to $-inf$) before the softmax step in the self-attention calculation.

Decoder



NB: The decoding process continues until a special symbol is reached indicating that the decoder has completed its output.

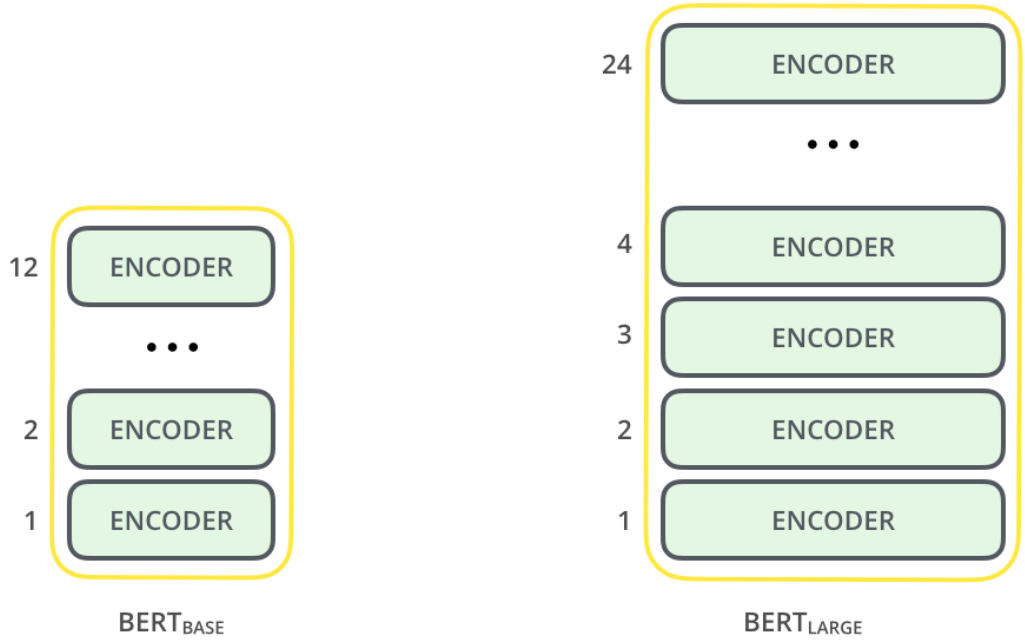
Final sub-layer



NB: The **linear** sub-layer is a simple fully connected neural network that projects the vector produced by the stack of decoders into a much larger vector of the size of the vocabulary (logits vector). Intuitively, each cell of the logits vector corresponds to the score given to the corresponding word. The **softmax** then turns those scores into probabilities. The cell with the highest probability is chosen and the word associated to it is the output for that time step.

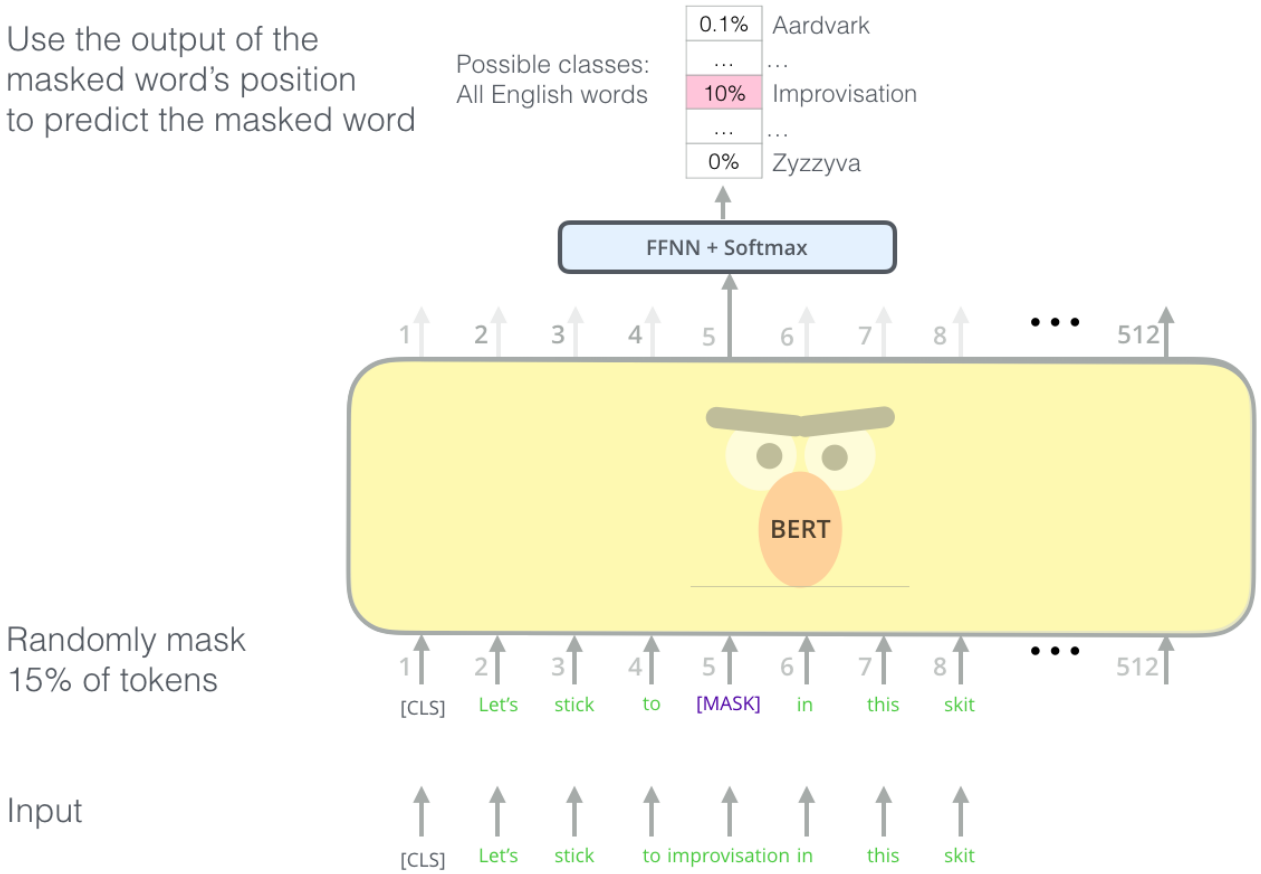
BERT (Devlin et al., 2018)

BERT: Bidirectional Encoder Representations from Transformers



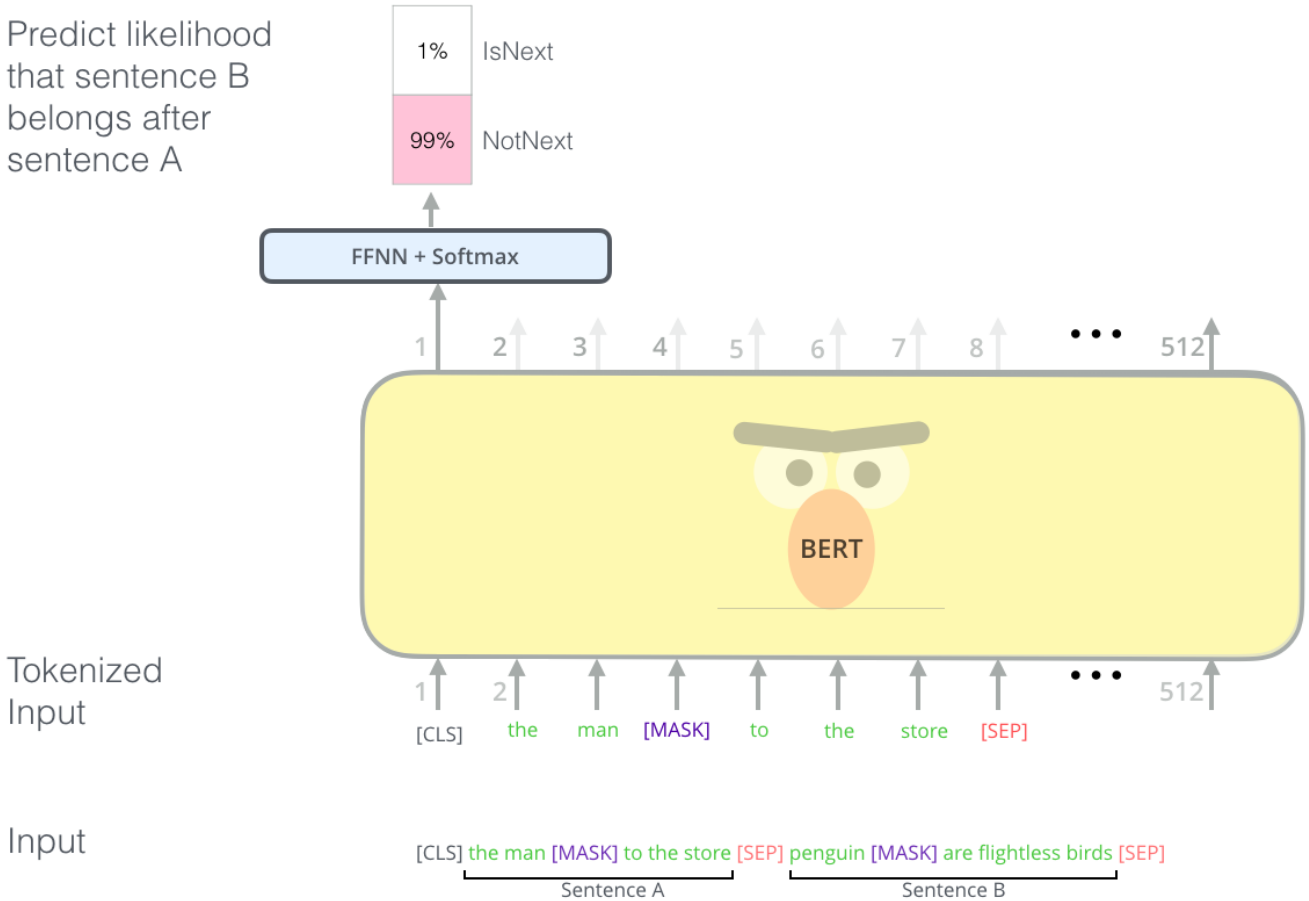
NB: BERT is a pre-trained Transformer **encoder** with 12 layers for its BASE model and 24 for its LARGE version (compared to 6 for the Transformer). In terms of other notable differences with the original architecture, BERT has a larger FFN (768 and 1024 hidden units for $BERT_{BASE}$ and $BERT_{LARGE}$ respectively, compared to 512 for the original Transformer) and more attention heads (12 and 16 for $BERT_{BASE}$ and $BERT_{LARGE}$ respectively, compared to 8 for the Transformer).

Masked Language Modeling (MLM)



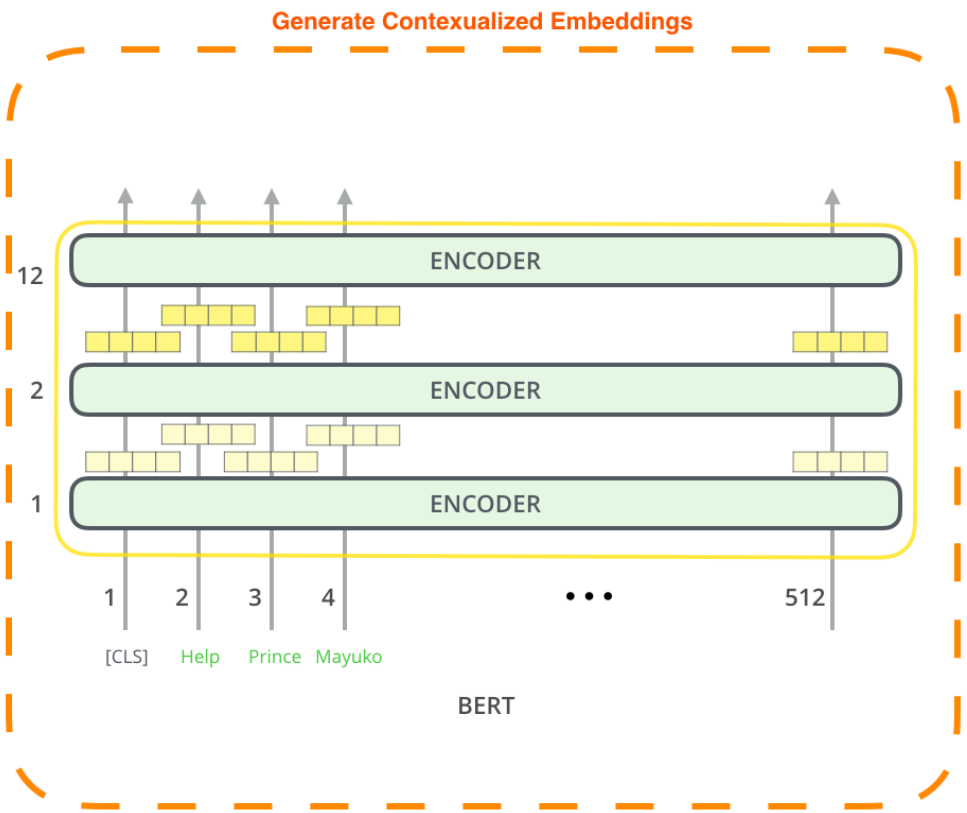
NB: The first task BERT is pre-trained on is a masked language modeling. Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words based on the context provided by the other, non-masked, words in the sequence.

Next Sentence Prediction (NSP)

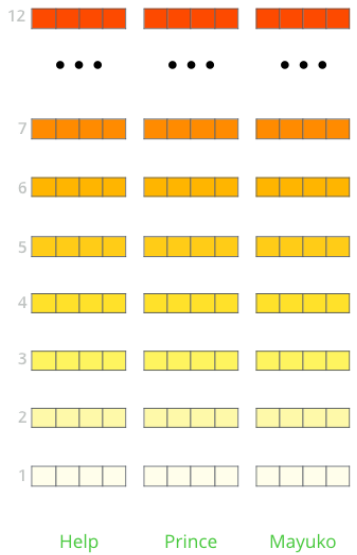


NB: The second task BERT is pre-trained on is a two-sentence classification task. In BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair entails the first one.

Contextualized embeddings with BERT



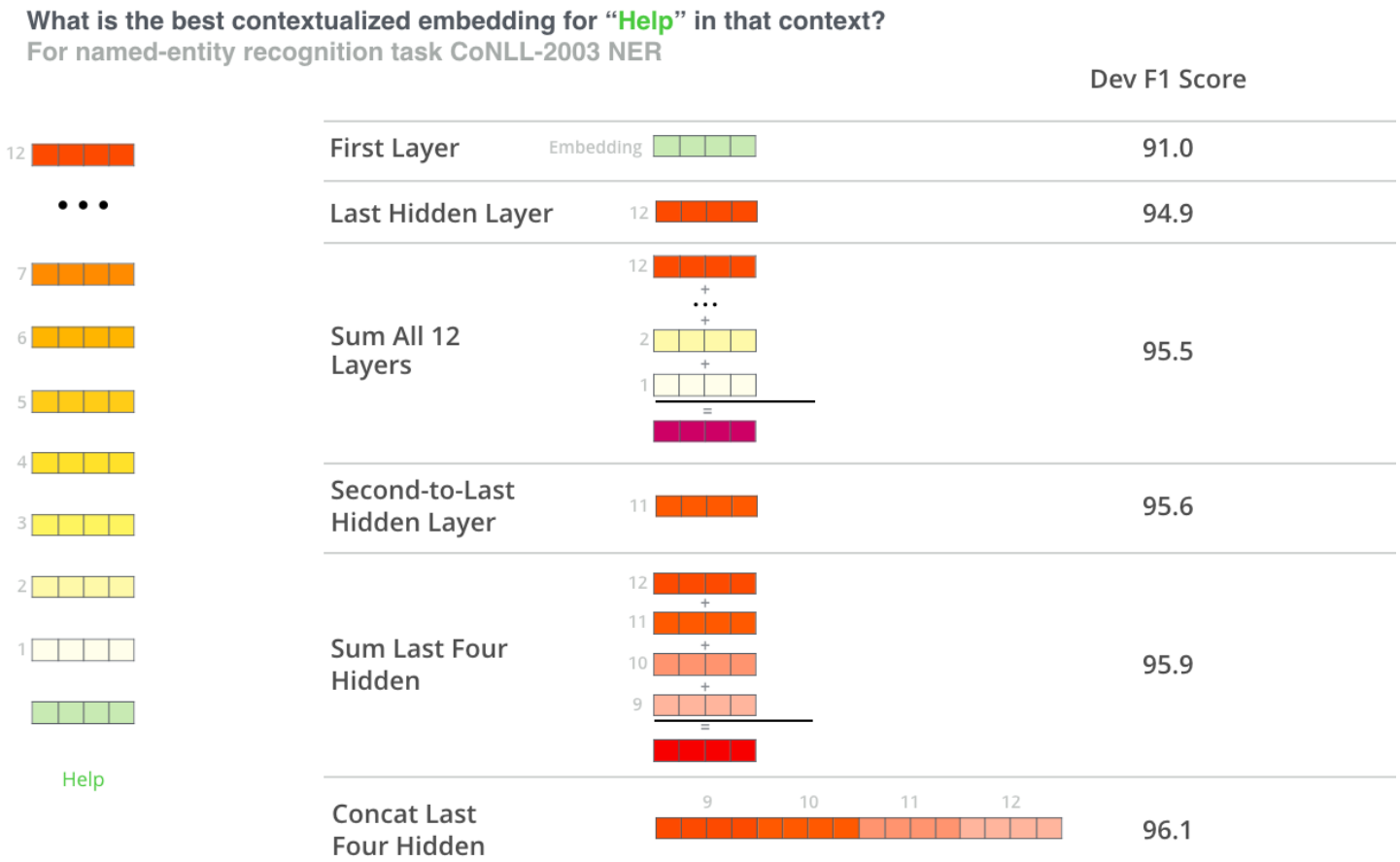
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

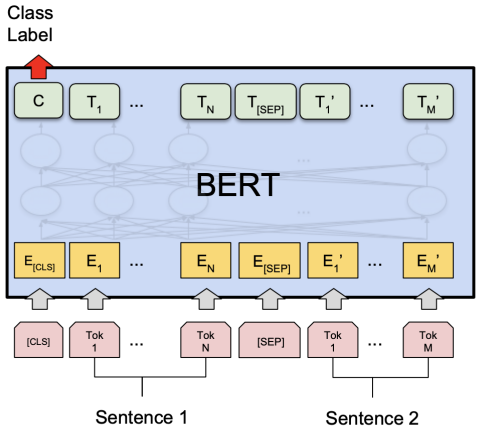
NB: Once pre-trained, BERT's output can be used as contextualized word embeddings. One can choose among multiple options to get the embedding of a word (summing up, averaging or concatenating all or some layer's output).

Contextualized embeddings with BERT

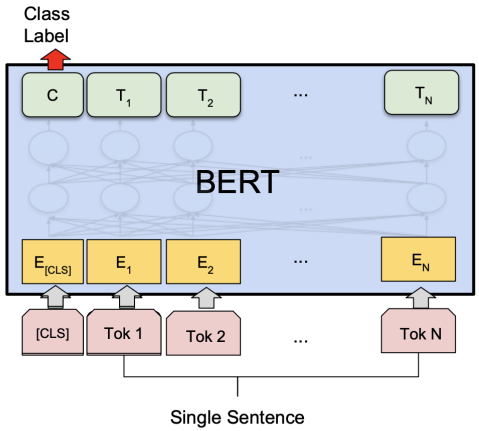


NB: Which vector work best as contextualized embedding depends on the task. Here is an example for Named Entity Recognition (NER) on the CoNLL-2003 dataset.

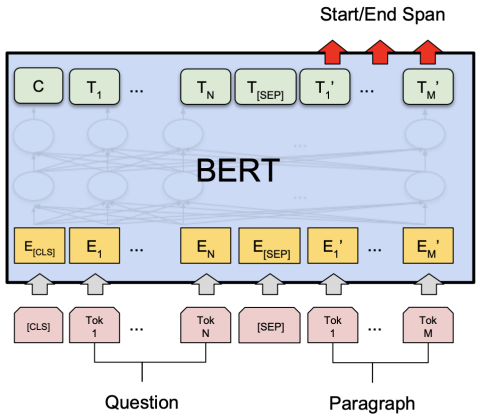
Fine-tuning BERT on downstream tasks



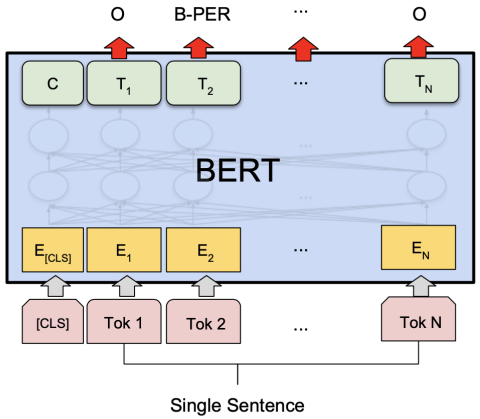
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



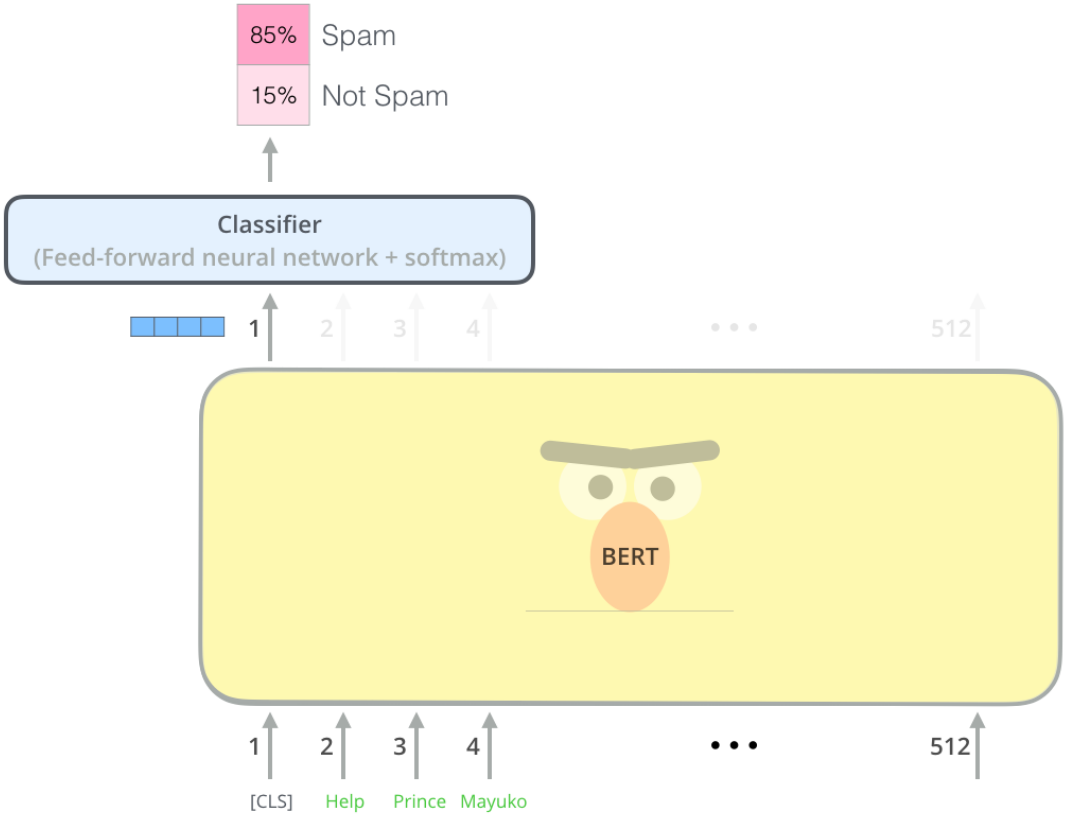
(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

NB: BERT obtained new state-of-the-art results on 11 natural language processing tasks.

Example: sentence classification

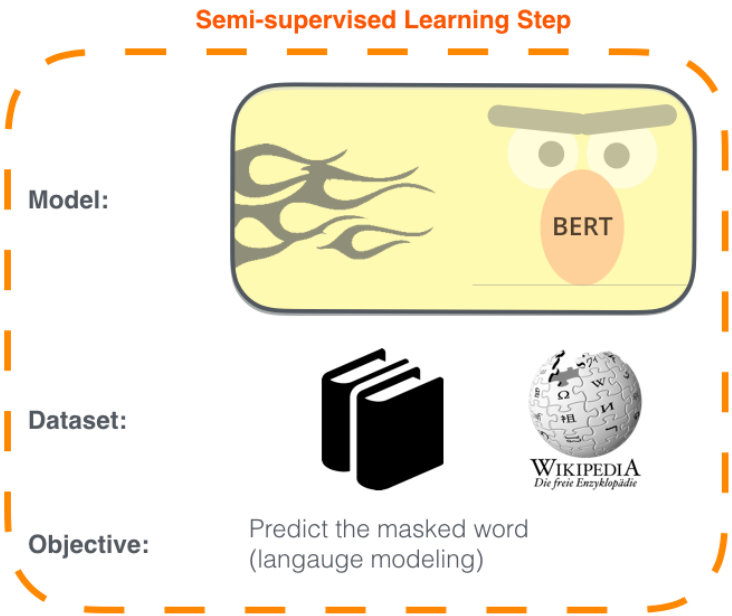


NB: The first input token is supplied with a special [*CLS*] token (standing for "classification") and is used for classification tasks where only the output vector of this special token is sent to the classifier.

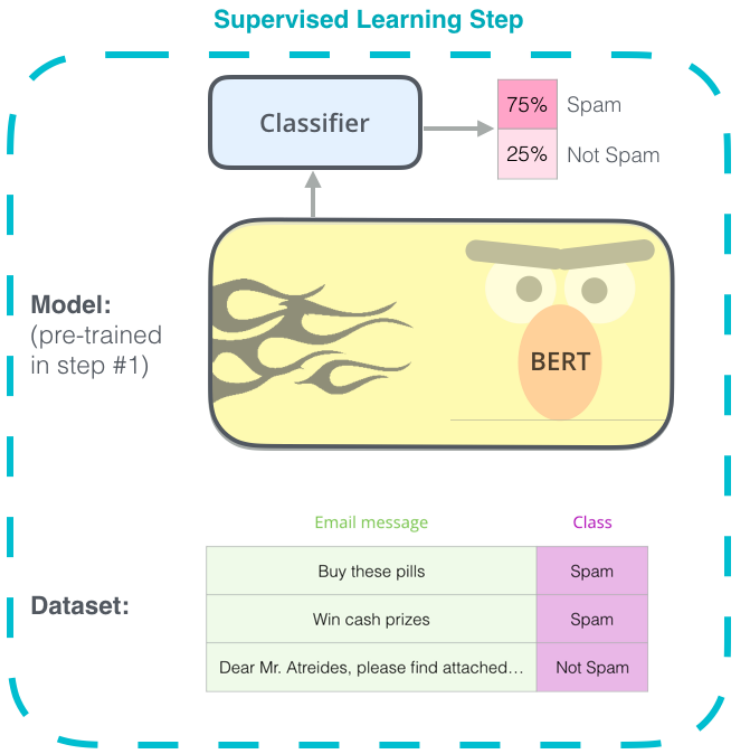
A two-step training approach

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.



Credits

- The Illustrated Word2vec, Jay Alammar. [1]
- The Illustrated Transformer, Jay Alammar. [2]
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), Jay Alammar. [3]
- The Illustrated GPT-2 (Visualizing Transformer Language Models), Jay Alammar. [4]
- A Visual Guide to Using BERT for the First Time, Jay Alammar. [5]

---

[1] *http://jalammar.github.io/illustrated-word2vec/*

[2] *http://jalammar.github.io/illustrated-transformer/*

[3] *http://jalammar.github.io/illustrated-bert/*

[4] *http://jalammar.github.io/illustrated-gpt2/*

[5] *http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/*