Department of Electrical Engineering and Computer Science
University of Liege, Belgium

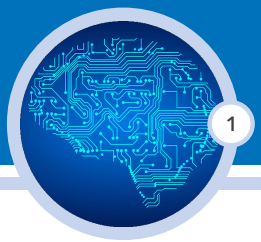
Master Thesis

An Illustrated Explanation of BERT

Antoine Louis

Feb, 2020

Outline



Part I : Introduction

Part II : Self-Attention

Part III : Transformer

Part IV : BERT

Part I : Introduction

Deep learning with text

Figure: Example of a Movie Review Sentiment classifier, taking as input a user review, and outputting a predicted sentiment on that review (either positive, negative, or neutral).

Word Embedding

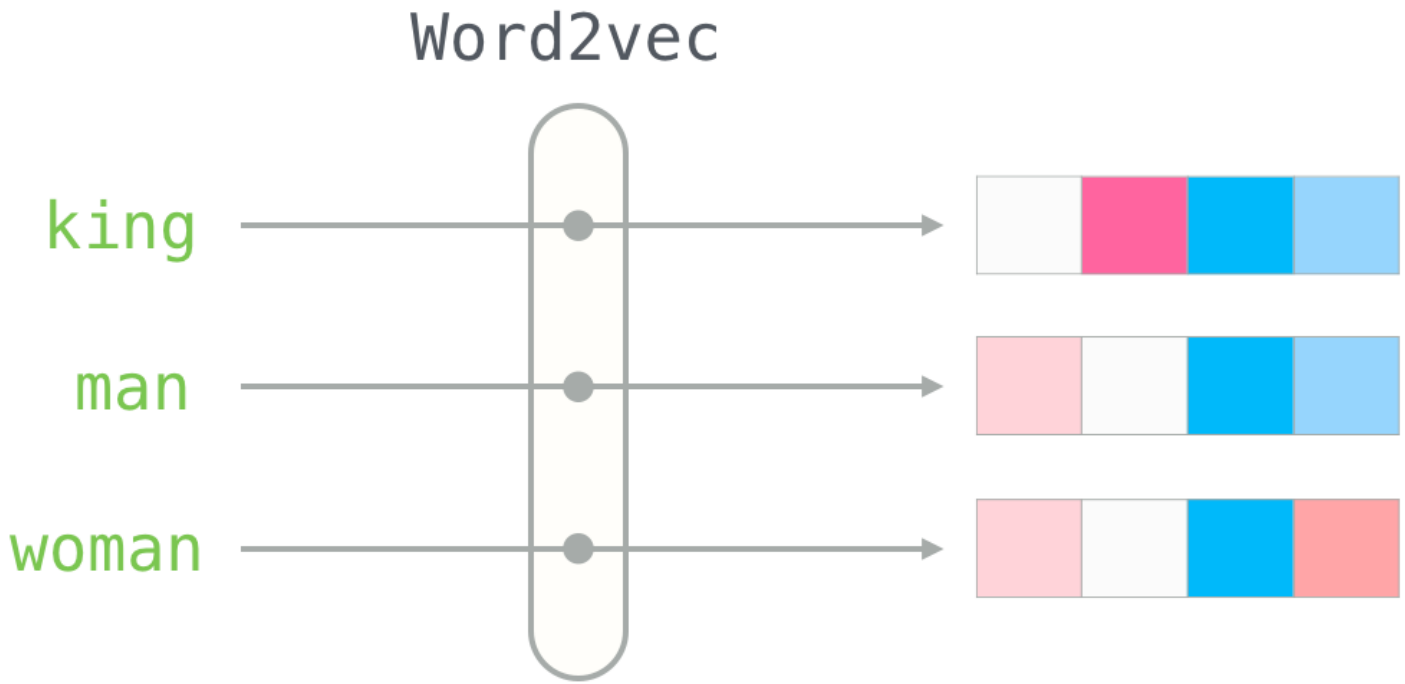


Figure: Word embeddings are representations of text data as lower dimensional vectors. This transformation is necessary because many machine learning algorithms (including deep neural networks) require their input to be vectors of continuous values (they just won't work on strings of plain text).

Classical representation of words

Figure: Well-known and largely used Word Embedding methods such as **Word2Vec** (2013), **GloVe** (2014) and **FastText** (2016), all have a major weakness: they create a fixed embedding for each word. Hence, whatever the meaning of the word, it will always be mapped to the same vector.

https://www.insider.com/words-with-the-most-definitions-2019-1

Part II : Self-Attention

Contextual representation of words

> ***"The meaning of a word is its use in the language."***
>
> (Wittgenstein, 1953)

Self-attention at high level

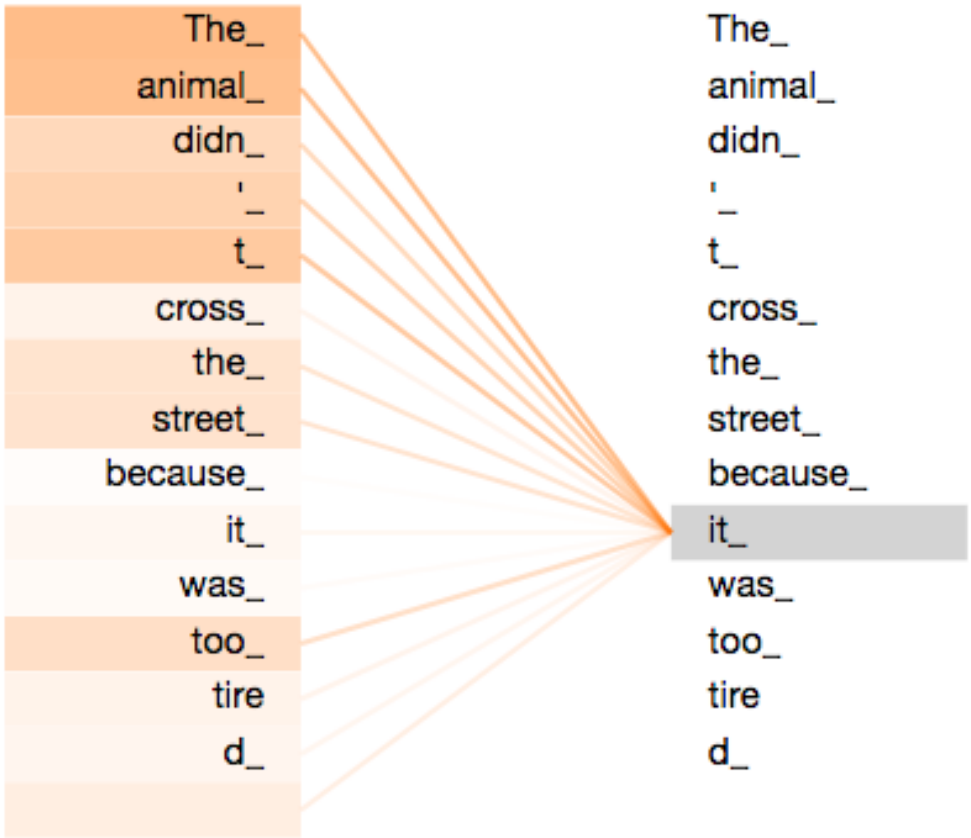


Figure: As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.

Self-attention in details

- **Step 1**: Create a **query vector**, a **key vector** and a **value vector** from each of the input vectors.

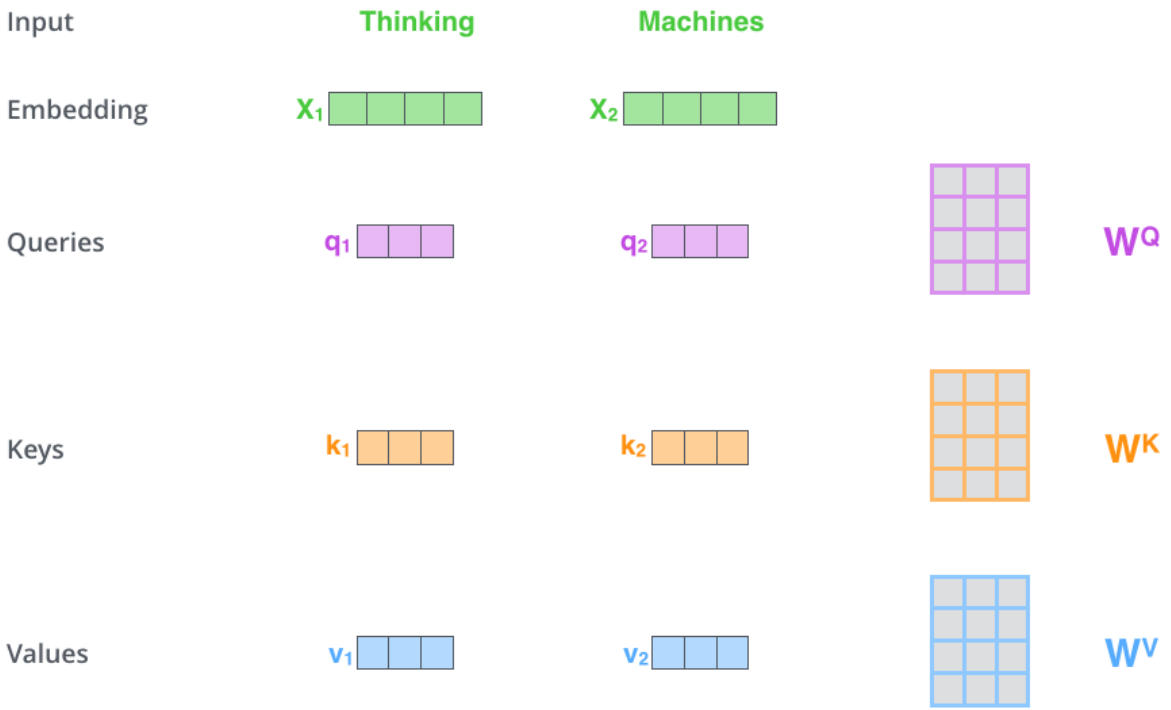


Figure: The **query**, **key** and **value** vectors are created by multiplying the embedding by three matrices that are trained during the training process.

Self-attention in details

- **Step 2**: For each word of the input sequence, **score** it against all the words in the sequence.

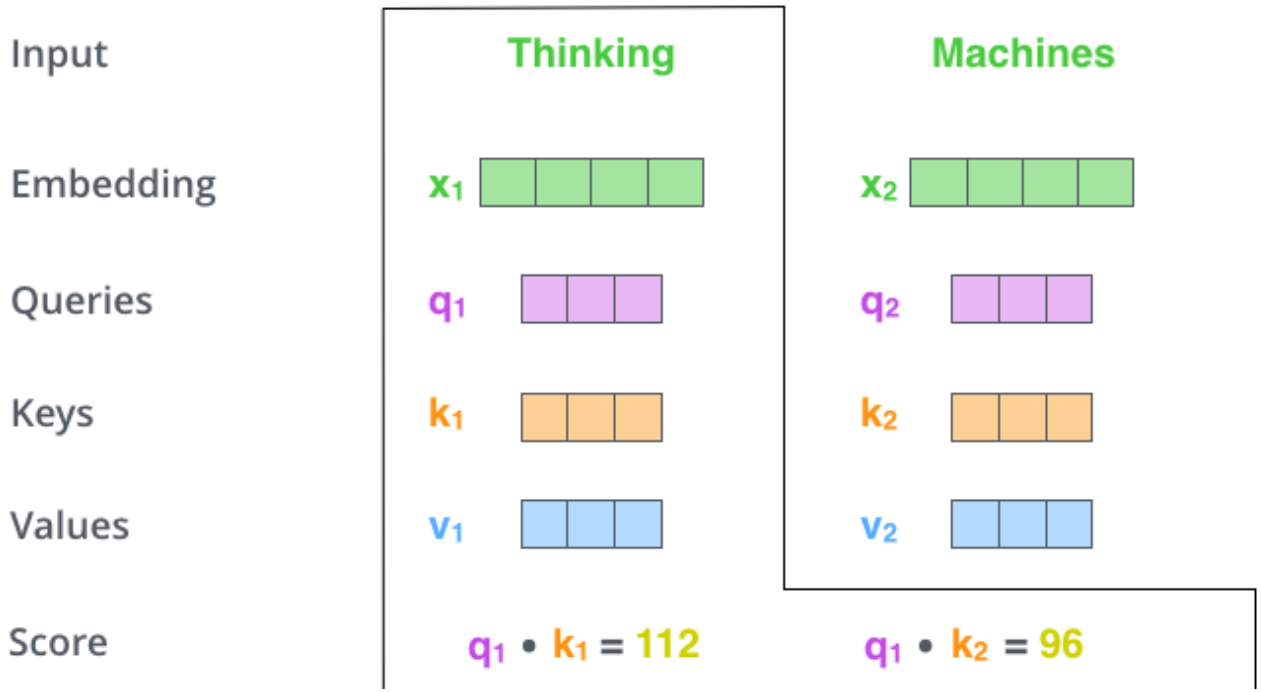


Figure: The **score** is calculated by taking the dot product of the query vector with the key vector of the respective word that is being scored.

Self-attention in details

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

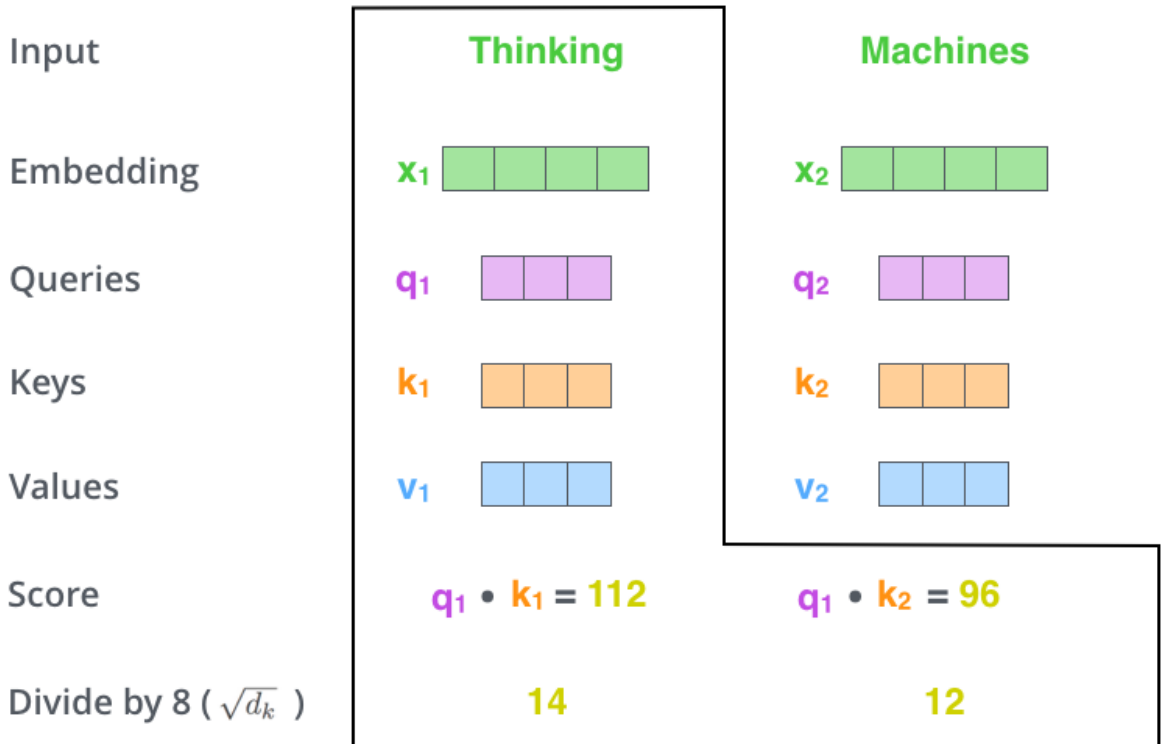


Figure: The division of the scores leads to having more stable gradients.

Self-attention in details

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

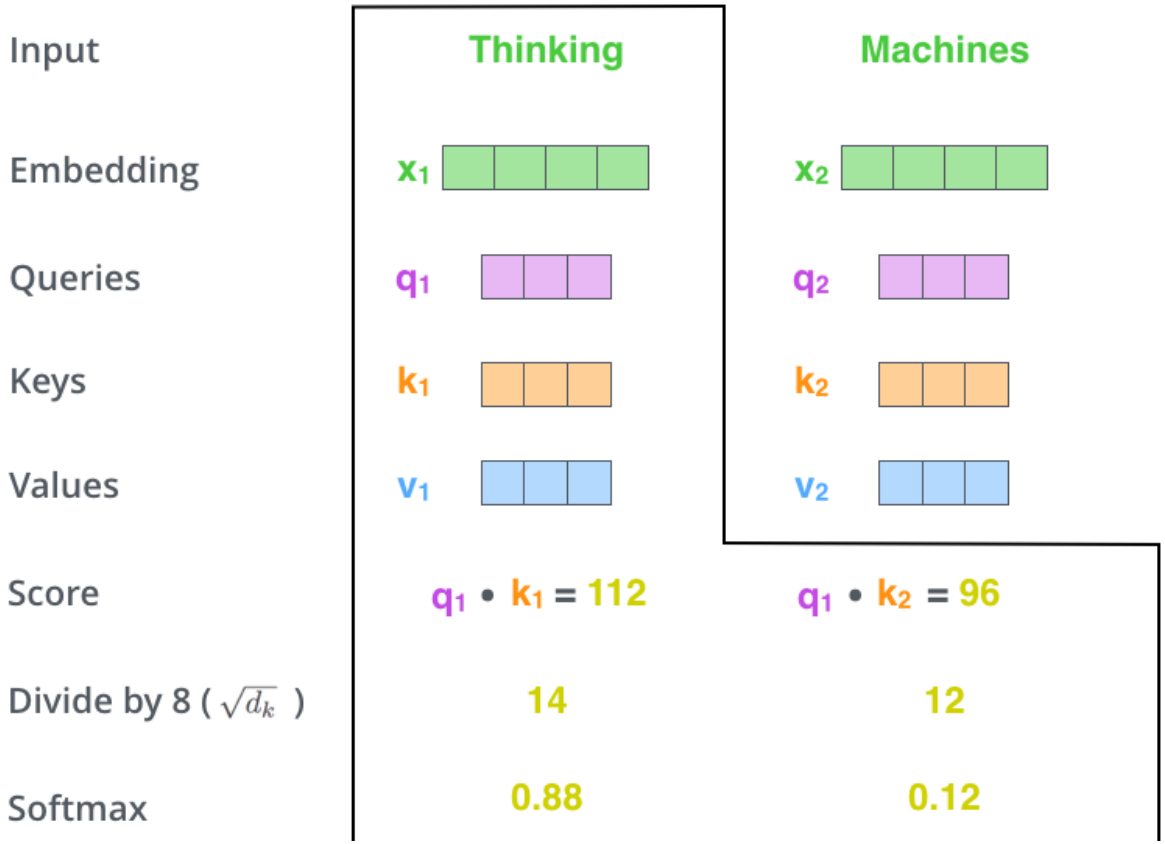


Figure: Softmax normalizes the scores so they are all positive and add up to 1.

Self-attention in details

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

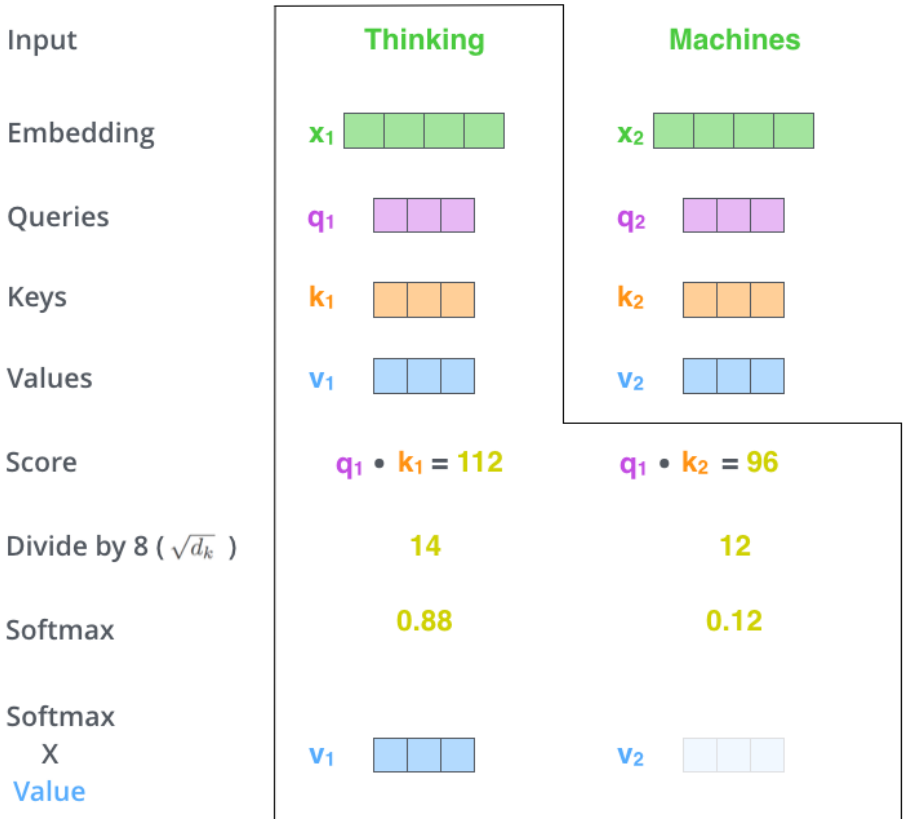


Figure: The intuition of this operation is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words.

Self-attention in details

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

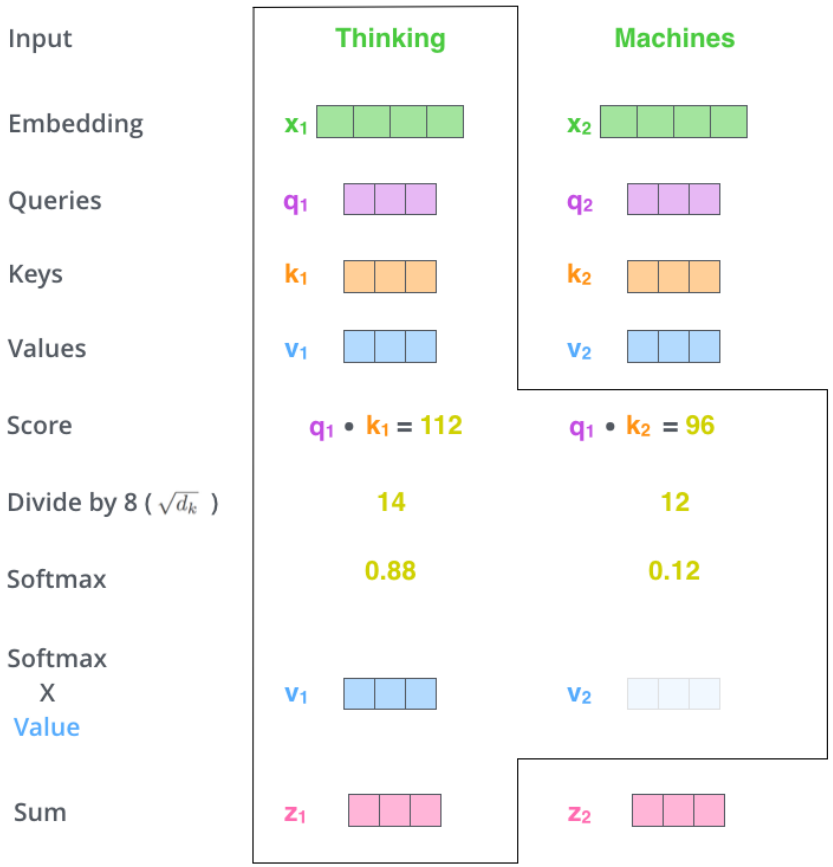


Figure: This produces the output of the self-attention calculation for a given word in the input sequence.

Self-attention in matrix form

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

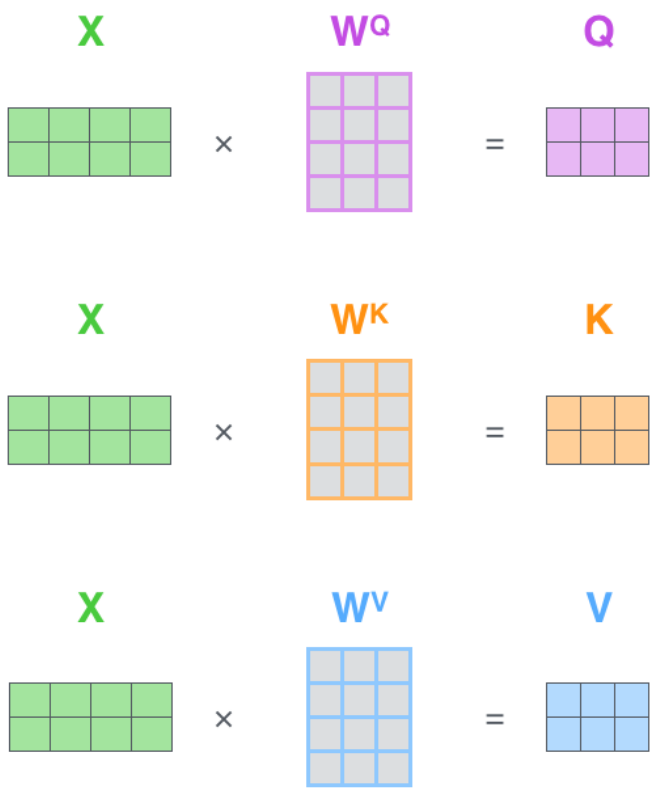


Figure: The **Query**, **Key**, and **Value** matrices are computed by packing the input embeddings into a matrix X, and multiplying it by the weight matrices that are being trained (W^Q, W^K, W^V).

Self-attention in matrix form

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

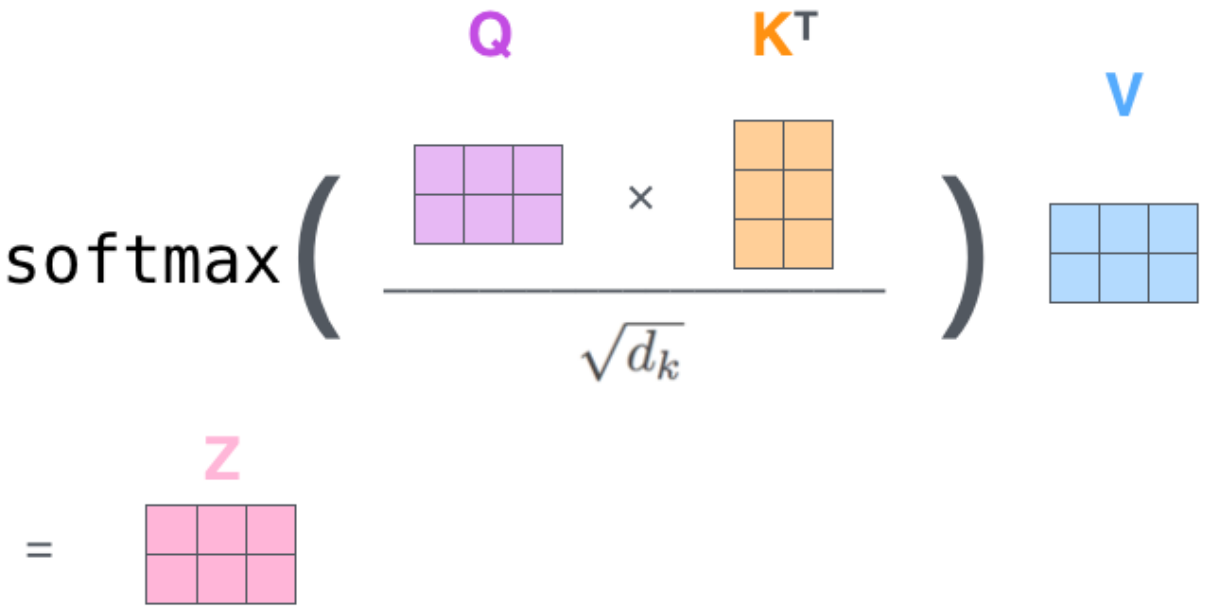


Figure: Self-attention calculation in matrix form.

Multi-headed attention

- In the actual **Transformer**, self-attention is further refined by adding a mechanism called **"multi-headed" attention**.
- Multi-headed attention involves multiple sets of Query/Key/Value weight matrices (called **attention heads**) in the attention calculation.

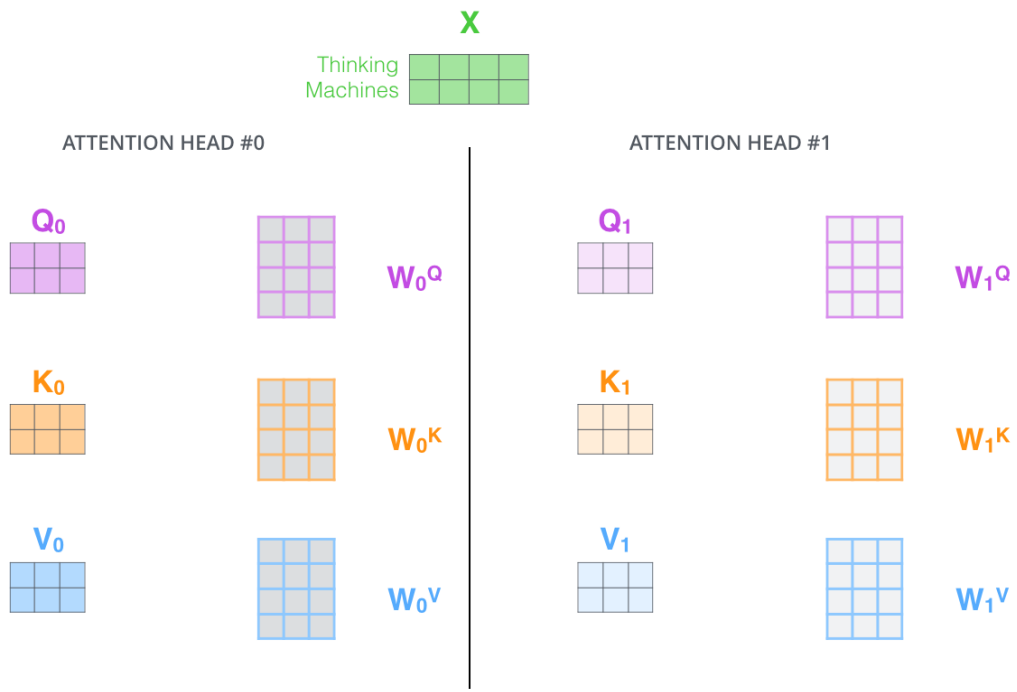


Figure: With multi-headed attention, separate $Q/K/V$ weight matrices are maintained for each head.

Multi-headed attention

- The actual **Transformer** uses 8 attention heads.

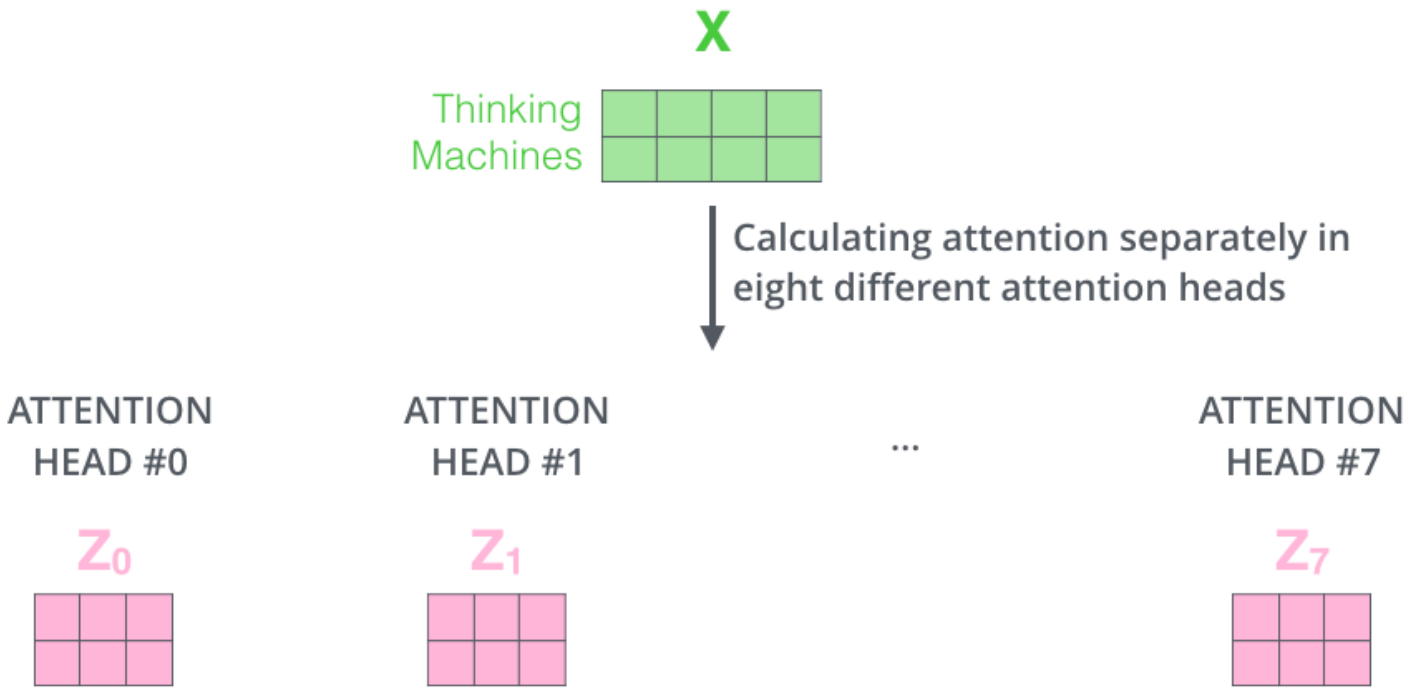
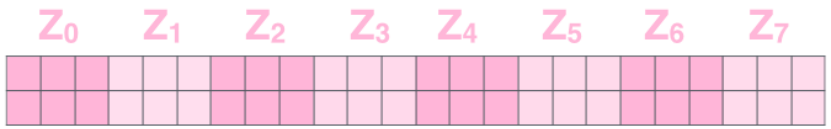


Figure: After the self-attention calculation, it will result 8 different *Z* matrices.

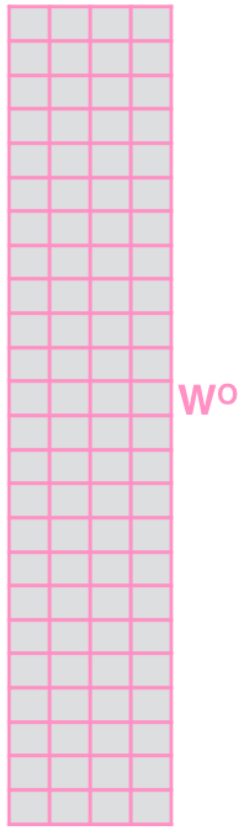
Multi-headed attention

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

=

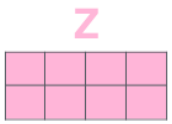


Figure: The final Z matrix is obtained by concatenating the different Z matrices, then multiplying them by an additional weights matrix W^O.

Multi-headed attention

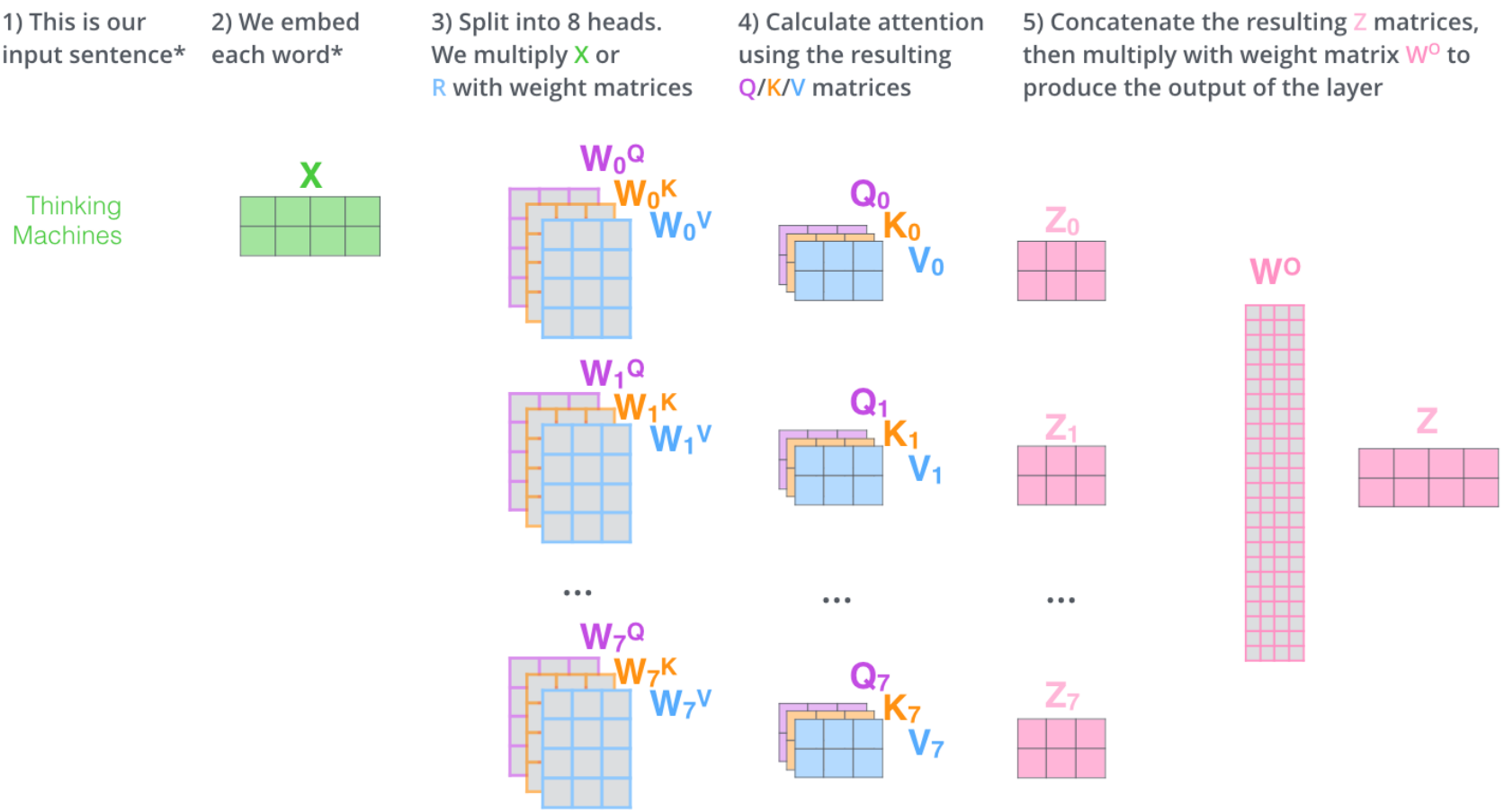


Figure: Computing multi-headed attention.

Part III : Transformer

Architecture

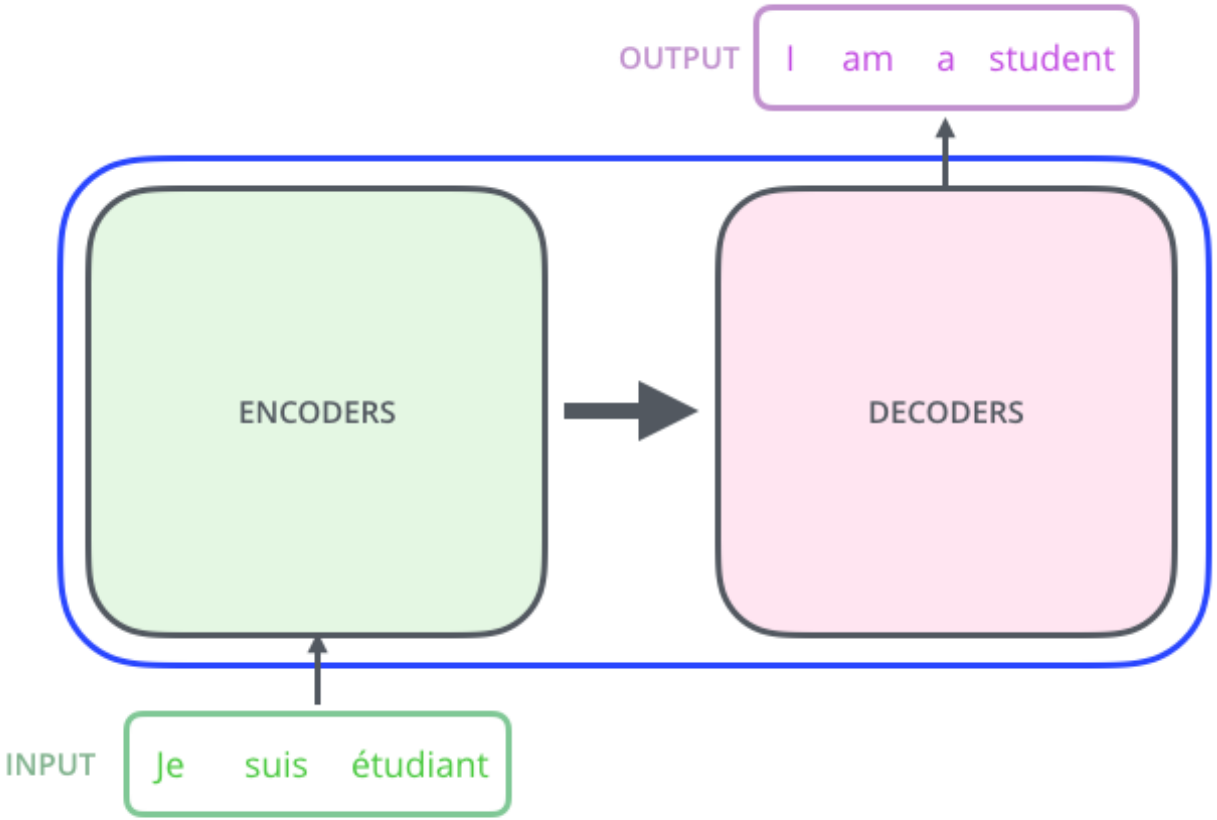


Figure: A typical **Transformer** architecture is composed of an **Encoder** and a **Decoder**.

Architecture

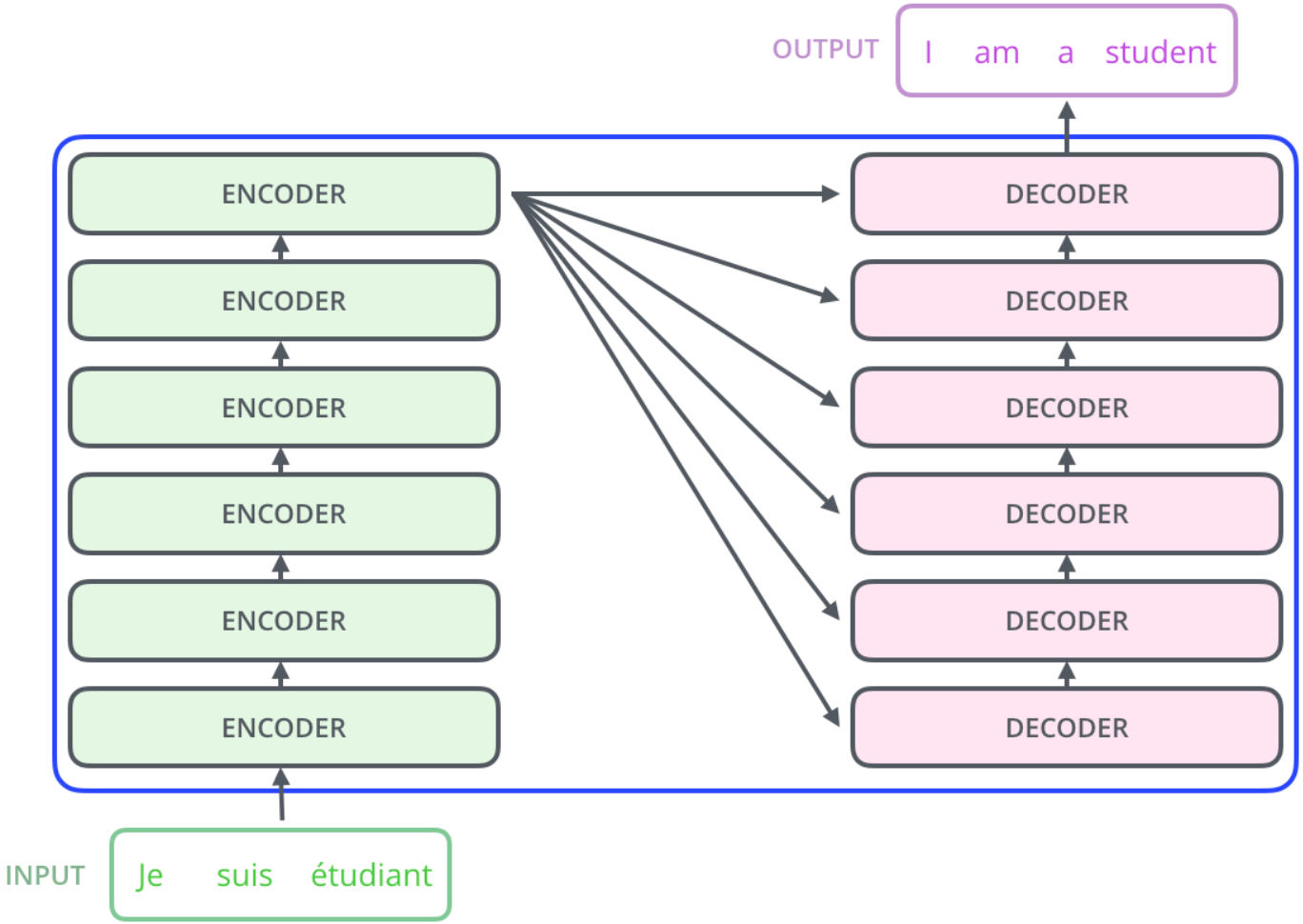


Figure: The Encoding and Decoding components are actually stacks of six encoders and decoders respectively, all identical in structure but not sharing the same weights.

Architecture

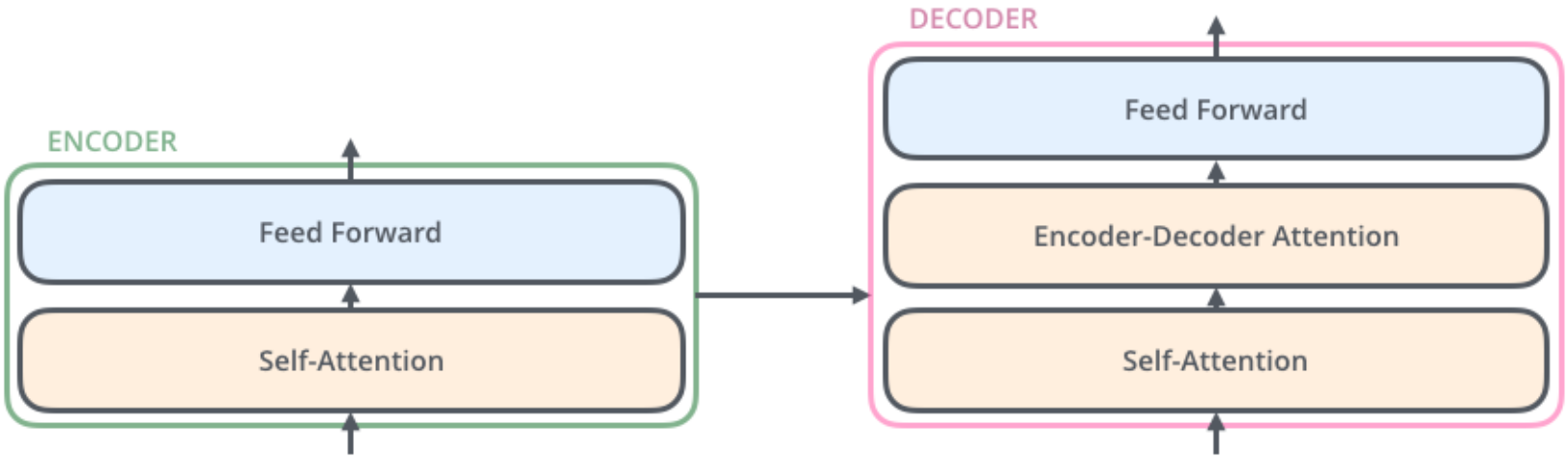


Figure: Each encoder/decoder contains two types of sub-layer: a **self-attention layer**, which helps look at other words in the sequence while encoding a specific word, and a **feed-forward layer**. In addition, the decoder contains a **encoder-decoder attention layer** which helps focus on relevant parts of the original input sequence.

Encoder

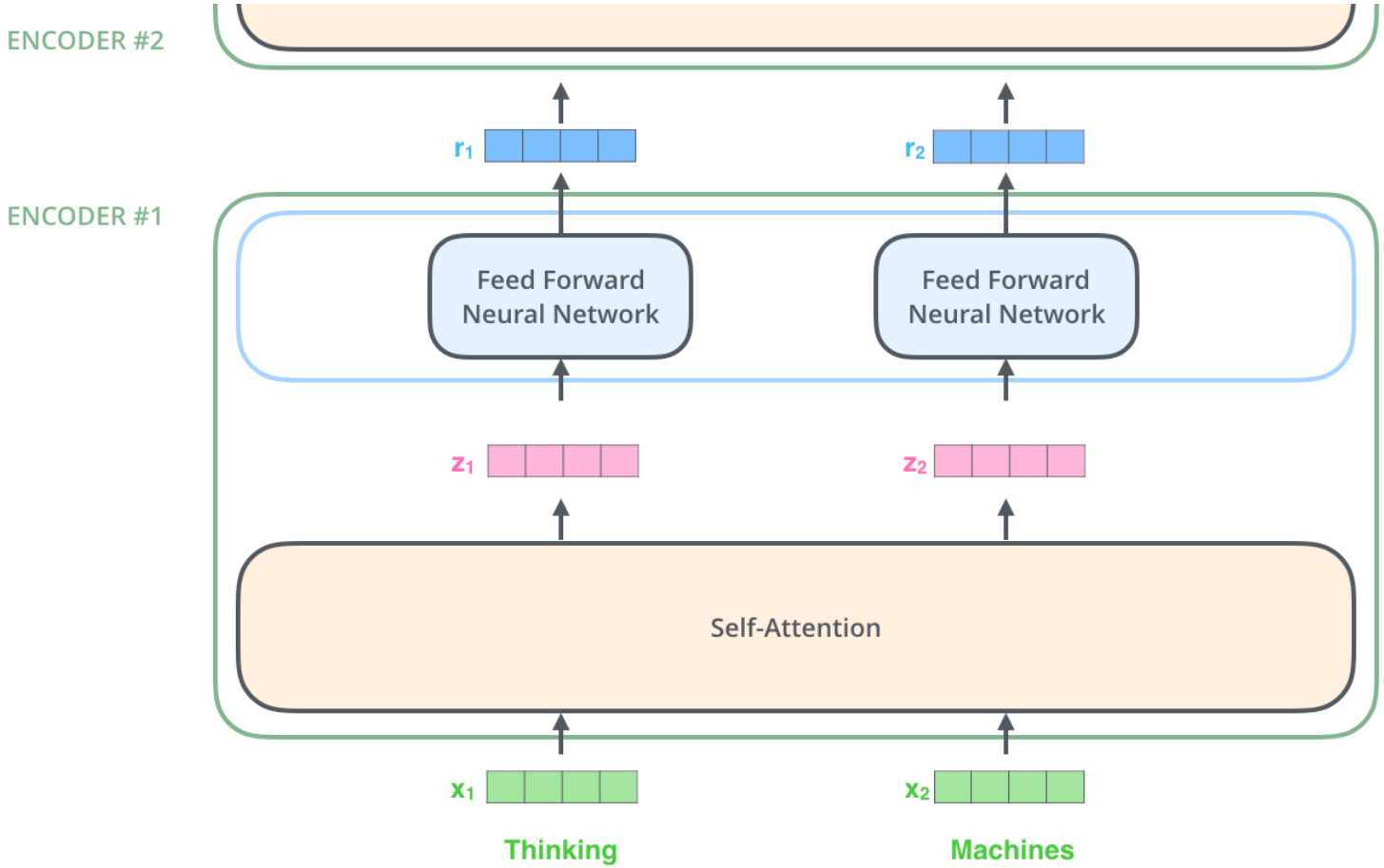


Figure: The exact same feed-forward network is independently applied to each position. It results in one key property of the Transformer, which is that each word flows through its own path in the encoder, allowing the various paths to be executed in **parallel**.

Positional Encoding

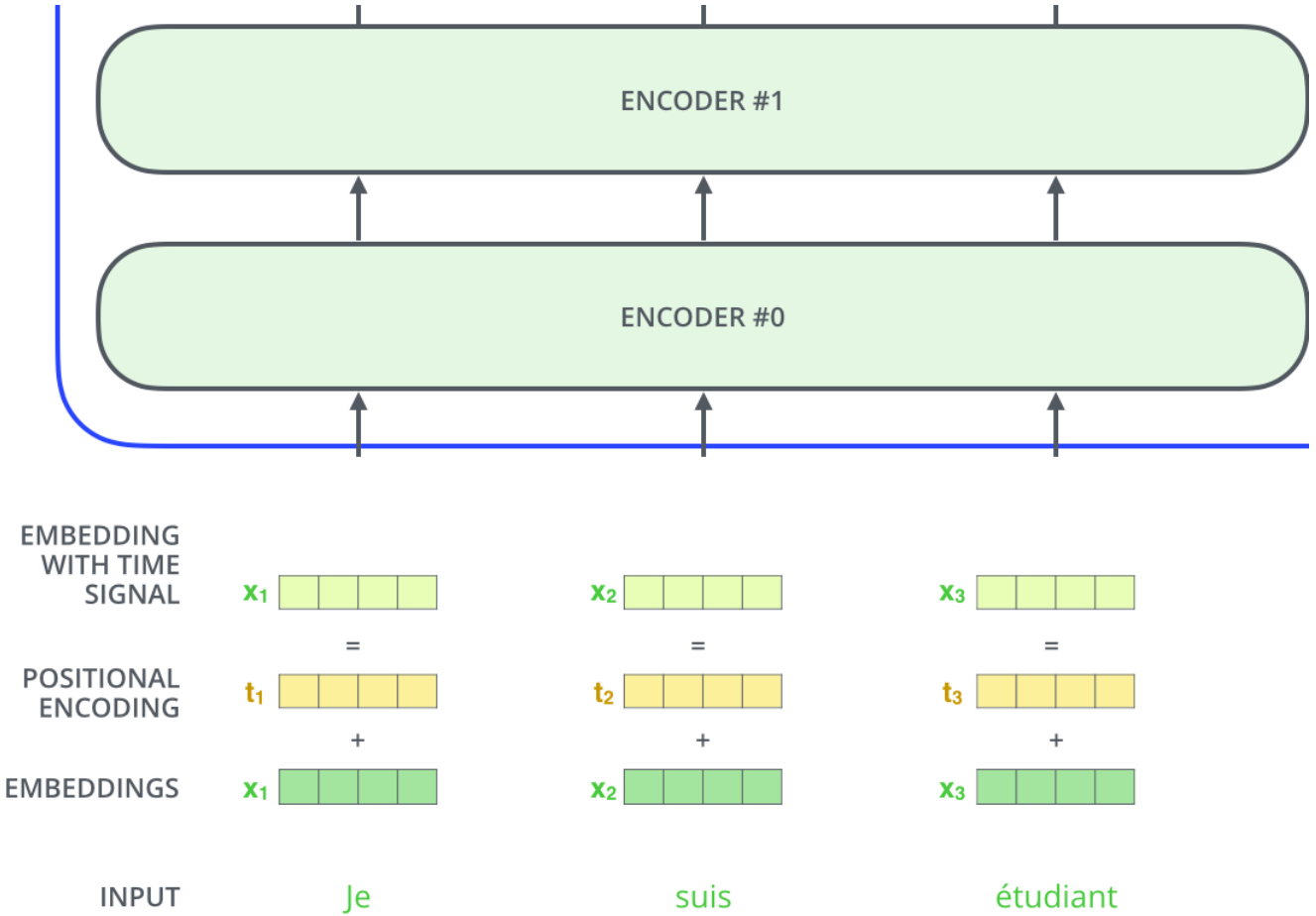


Figure: To give the model a sense of the order of the words, **positional encoding vectors** are added to the original input vectors. These vectors follow a specific pattern that the model learns.

Residual connections

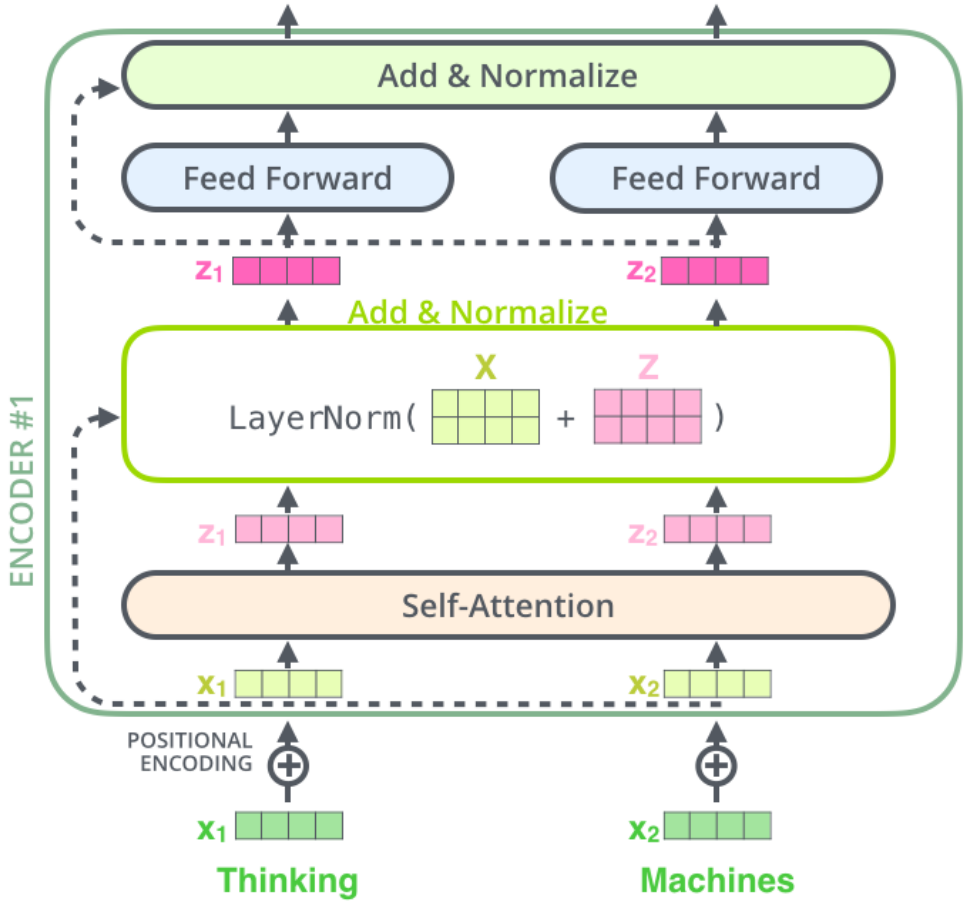


Figure: Each sub-layer in an encoder has a **residual connection** around it, and is followed by a **layer-normalization** step.

Residual connections

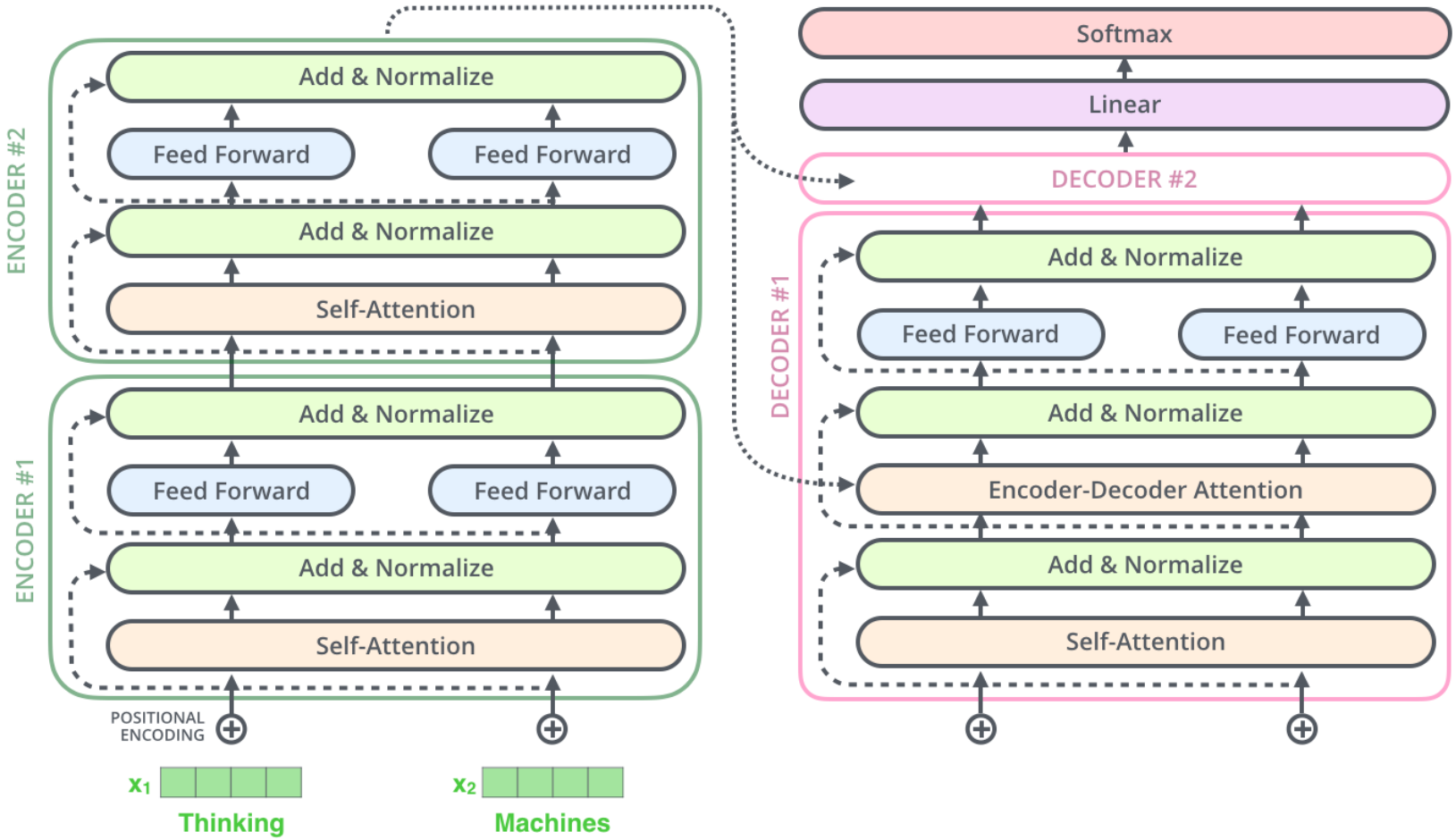


Figure: Each sub-layer in a decoder also has a **residual connection** around it, followed by a **layer-normalization** step.

Decoder

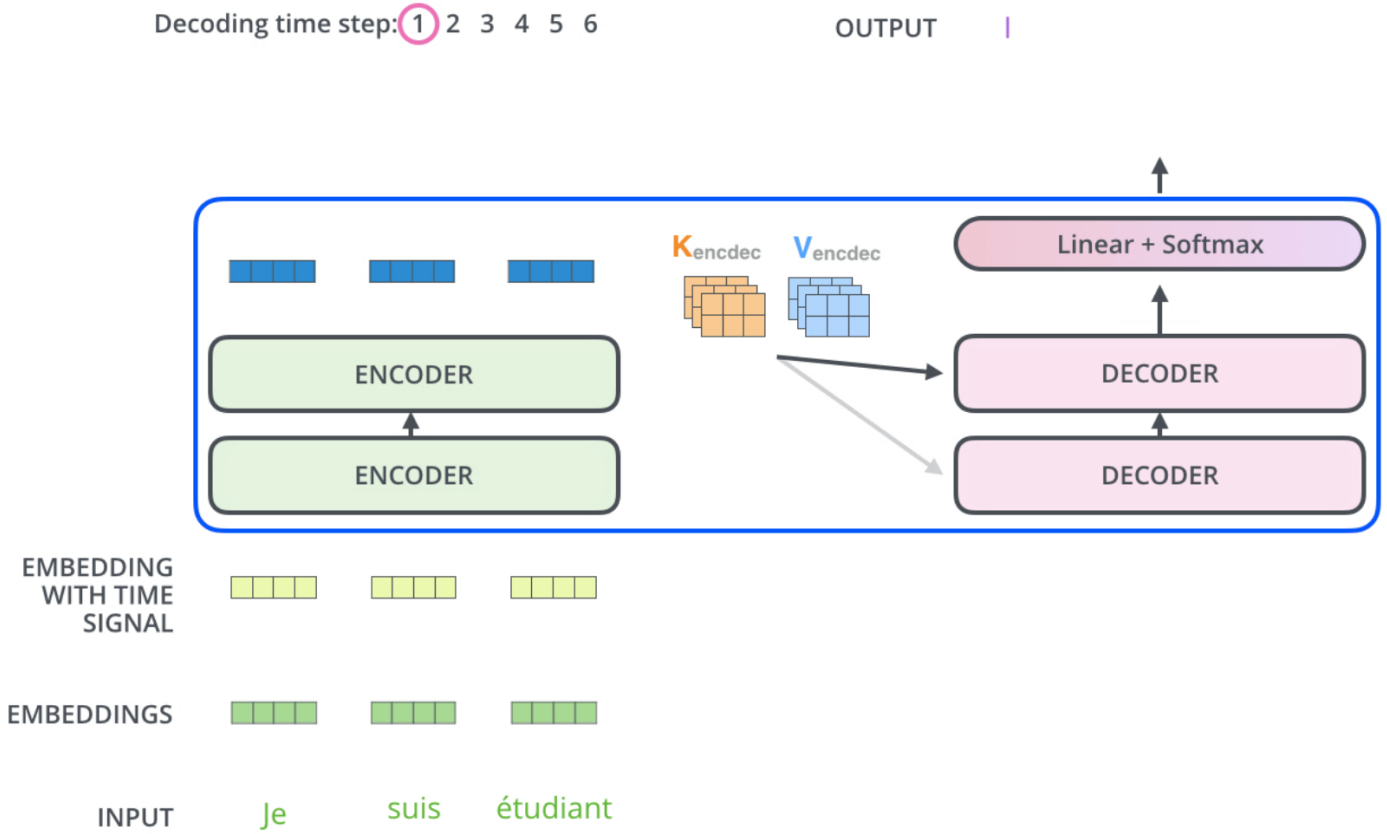


Figure: The encoders start by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V (result of the multiplication of the outputs r_i with the weight matrices Q^K and Q^V). Matrices K and V are to be used by each decoder in its **encoder-decoder attention layer**, which works just like multi-headed self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack. This layer helps the decoder focus on appropriate places in the input sequence.

Decoder

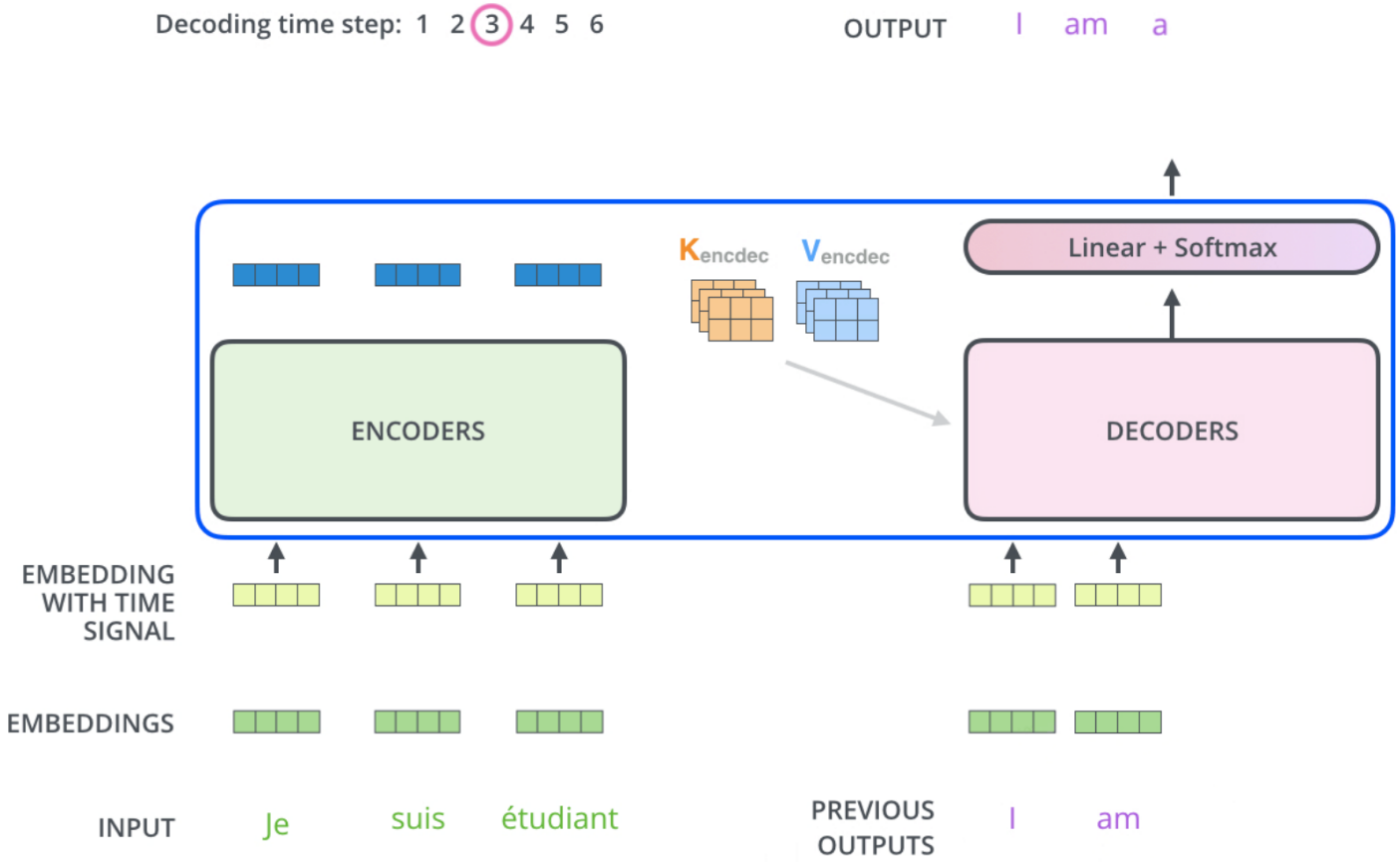


Figure: The output of each step is fed to the bottom decoder in the next time step. Also, the **self-attention layer** in the decoder is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

Decoder

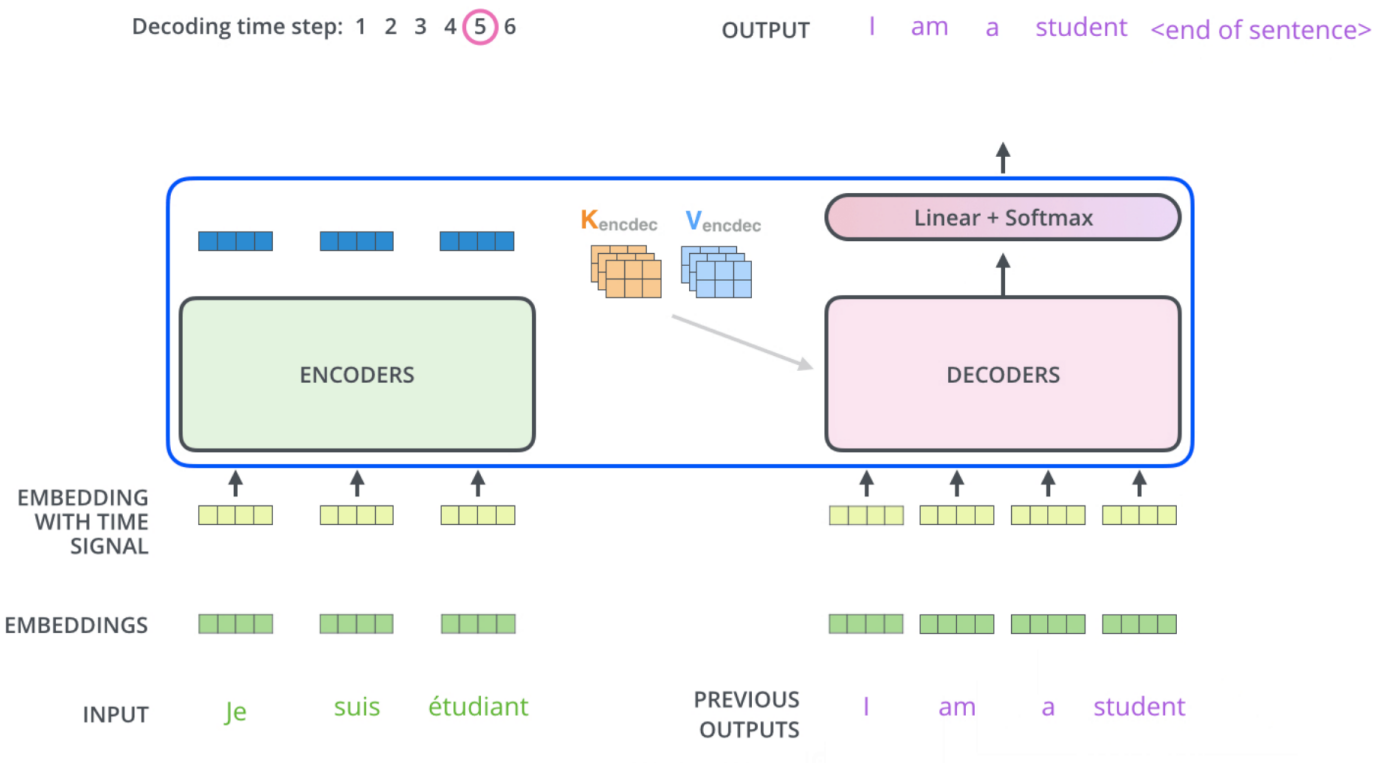


Figure: The decoding process continues until a special symbol is reached indicating that the transformer decoder has completed its output.

Final Linear and Softmax Layer

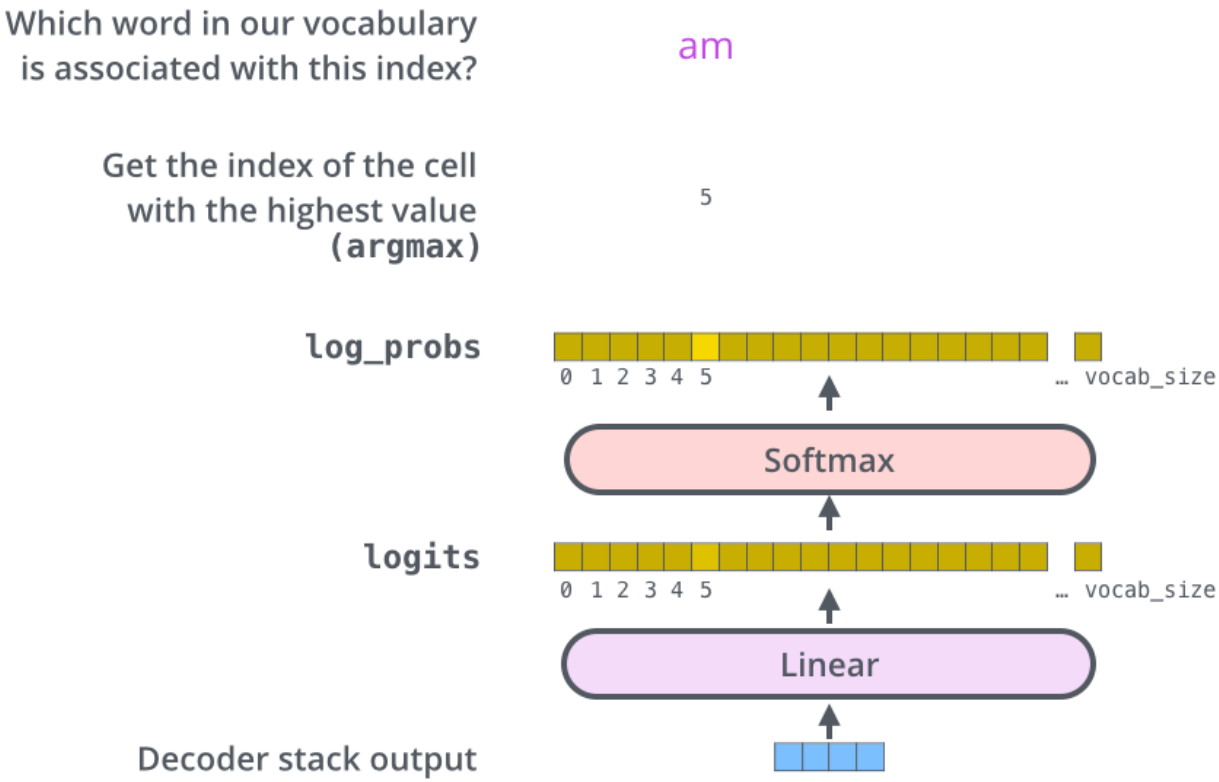


Figure: The **Linear layer** is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much larger vector called a logits vector. Intuitively, each cell of the logits vector corresponds to the score of a unique word. The **Softmax layer** then turns those scores into probabilities. The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Part IV : BERT

BERT: Bidirectional Encoder Representations from Transformers

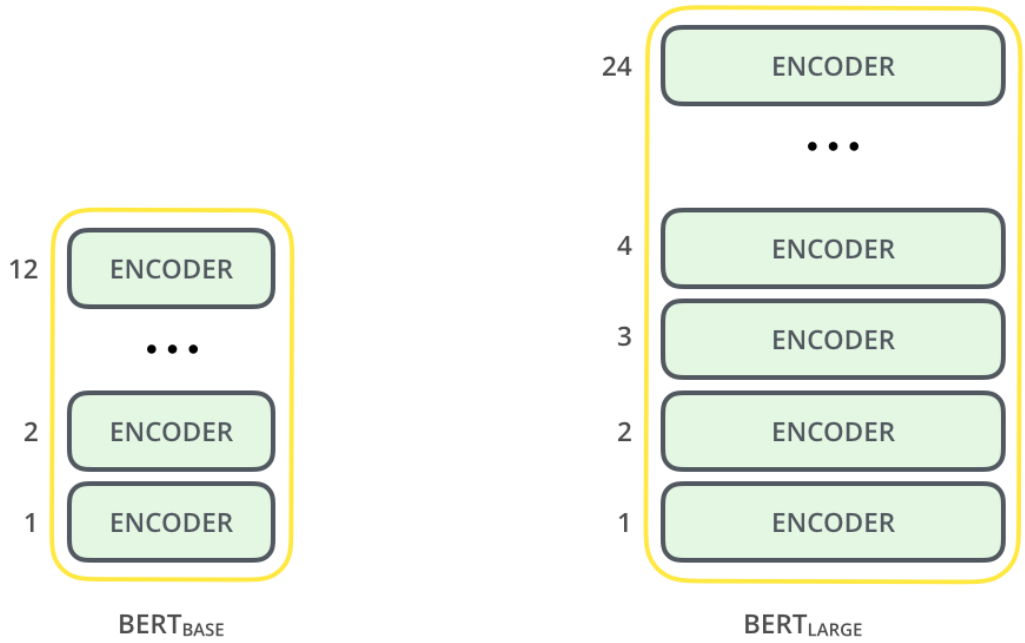


Figure: BERT is basically a trained **Transformer Encoder stack**, with 12 Encoder layers for the Base version, and 24 for the Large version. These also have larger feedforward-networks (768 and 1024 hidden units respectively), and more attention heads (12 and 16 respectively) than the default configuration in the reference implementation of the Transformer in the initial paper (6 encoder layers, 512 hidden units, and 8 attention heads).

Model inputs and outputs

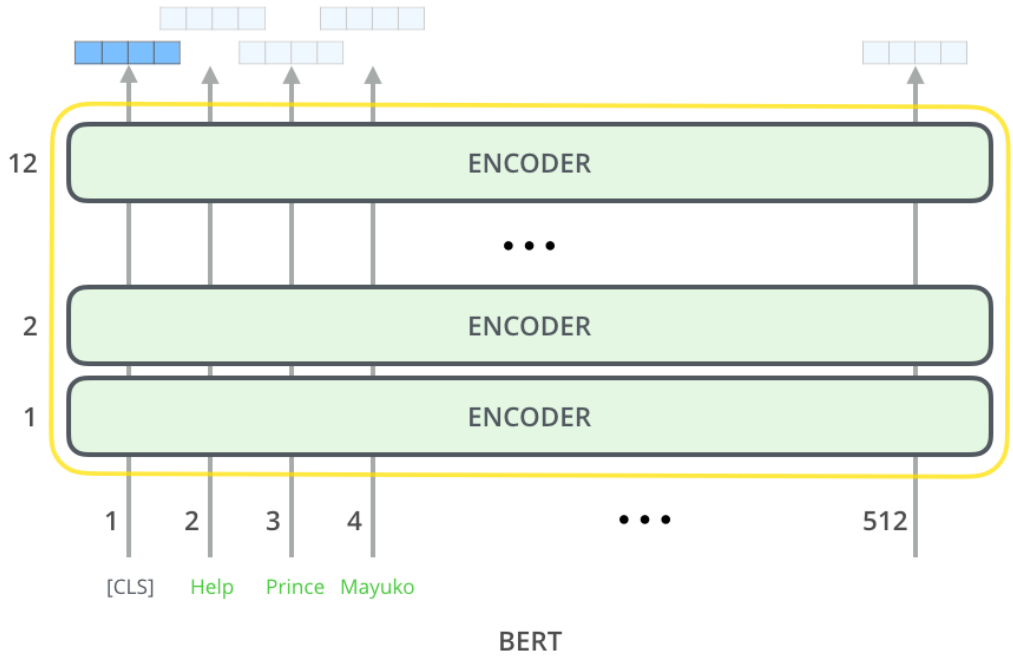


Figure: Just like the vanilla encoder of the Transformer, BERT takes a sequence of words as input which keep flowing up the stack. Each position outputs a vector of size *hidden_size* (768 in BERT Base).

Tokenization

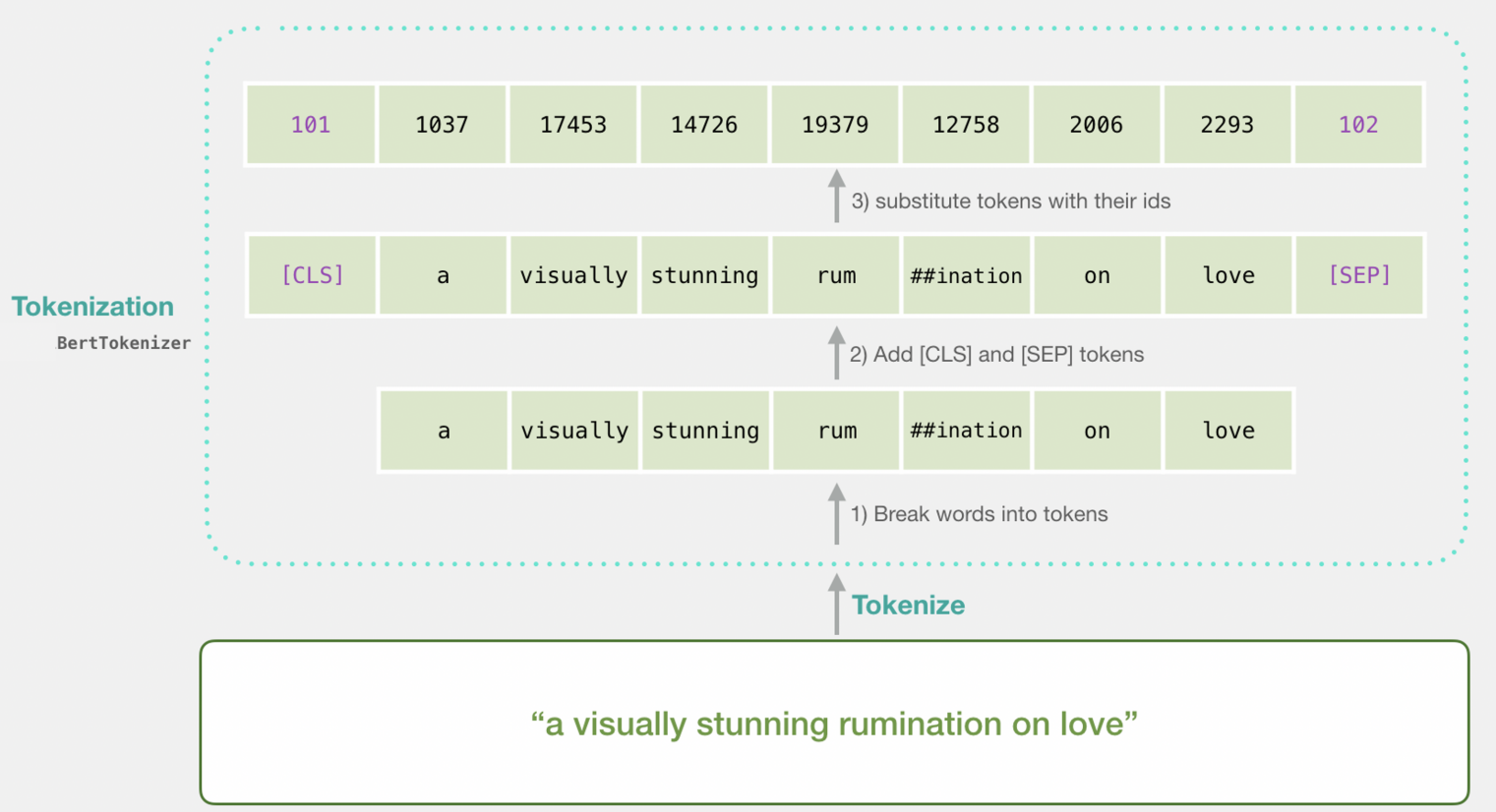


Figure: The BERT tokenizer is a **WordPiece model**, a data-driven tokenization method that creates a fixed-size vocabulary of individual characters, subwords and words that best fits a given language corpus. To tokenize a word under this model, the tokenizer first checks if the whole word is in the vocabulary. If not, it tries to break it into the largest possible subwords contained in the vocabulary, and as a last resort will decompose the word into individual characters. The vocabulary used by BERT contains the $\sim$ 30.000 most common words and subwords found in the English language plus all English characters.

Input embeddings

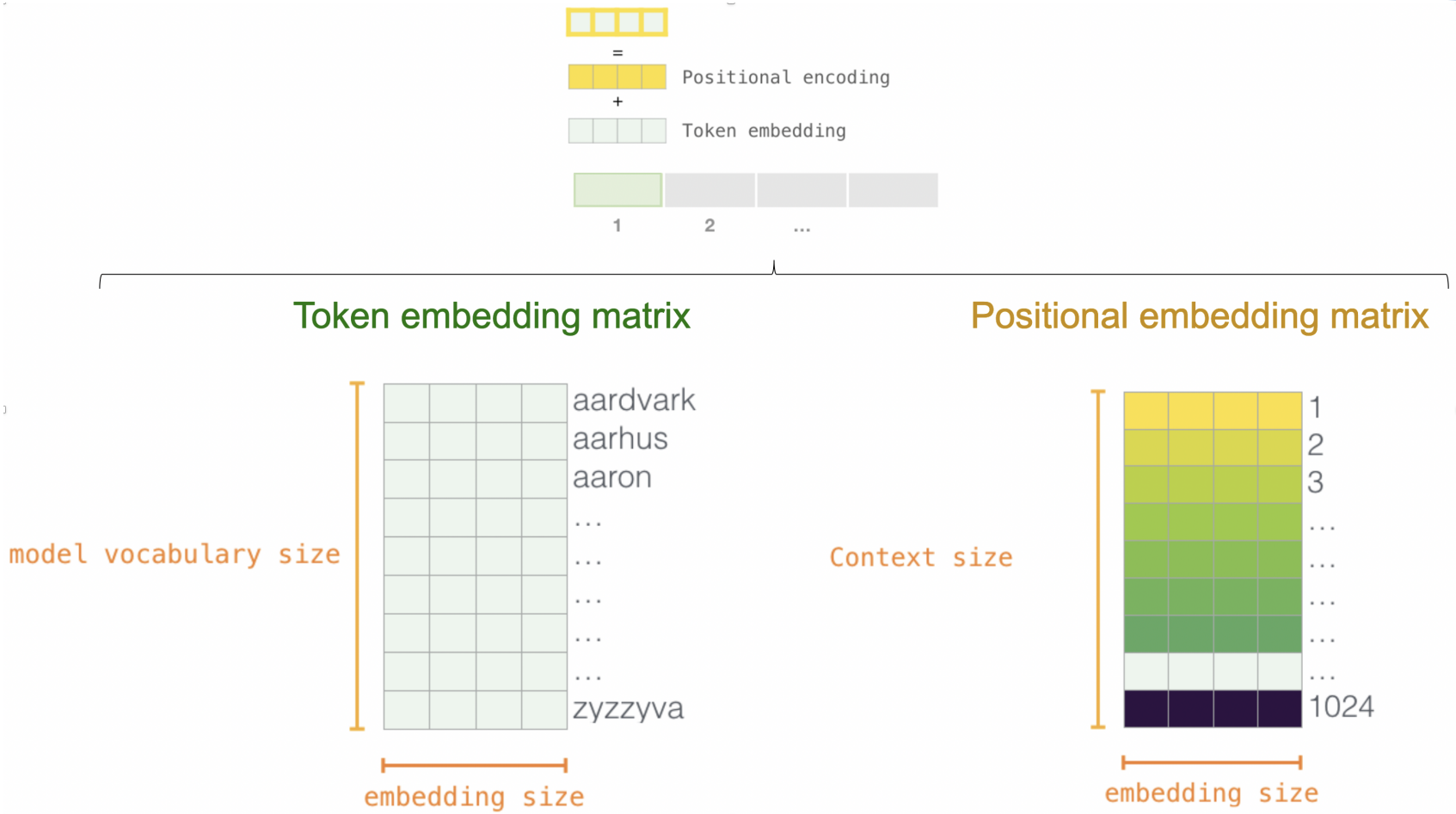


Figure: **Token embedding** and **Positional embedding** matrices (weight matrices part of the trained model).

Weight matrices

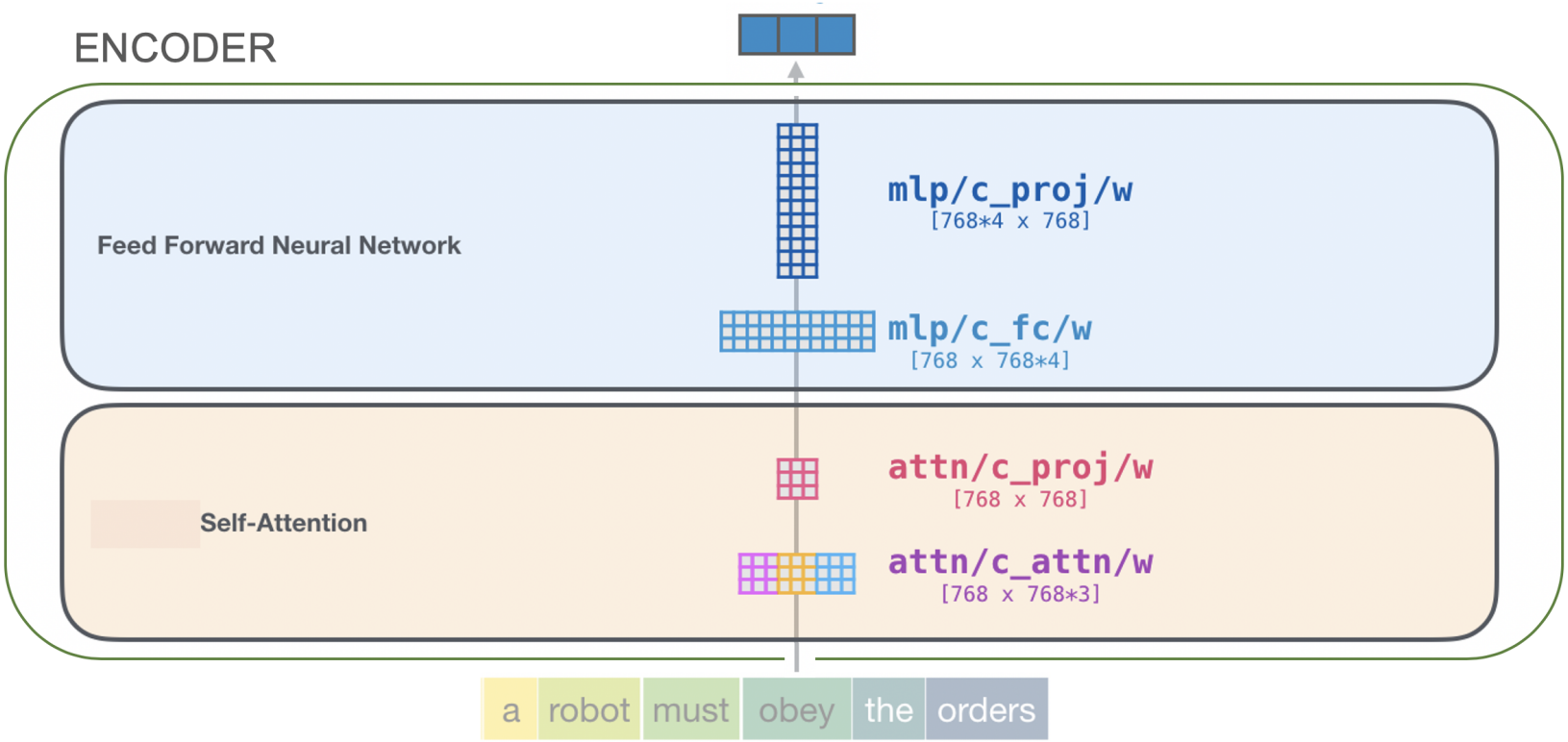


Figure: Every block in a transformer has its own weights. (i) the weight matrix used to create the queries, keys, and values (**attn/c_attn/w**); (ii) the weight matrix that projects the results of the attention heads into the output vector of the self-attention sub-layer (**attn/c_proj/w**); (iii) the 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**).

Masked Language Model (MLM)

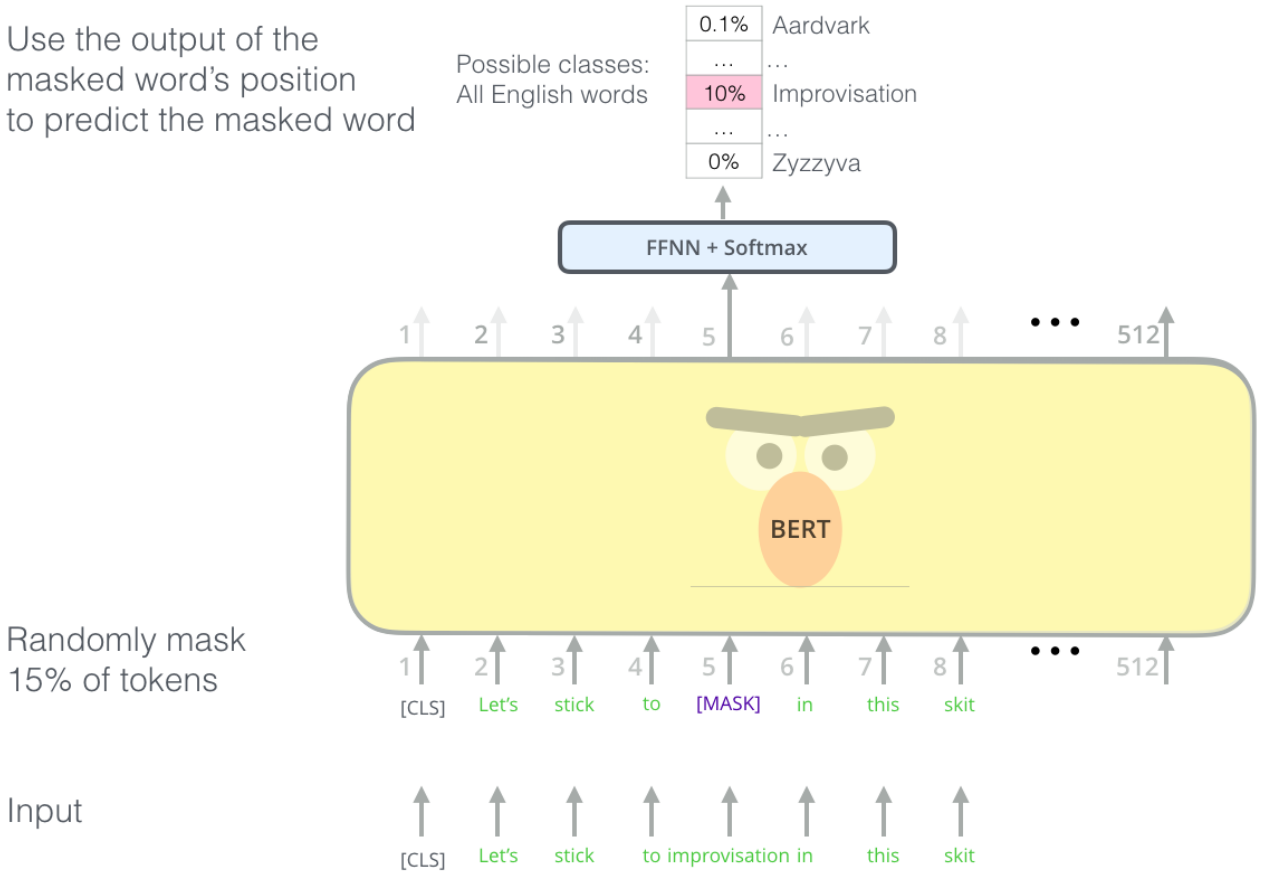


Figure: The first task BERT is pre-trained on is a masked language modeling task. Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the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

Next Sentence Prediction (NSP)

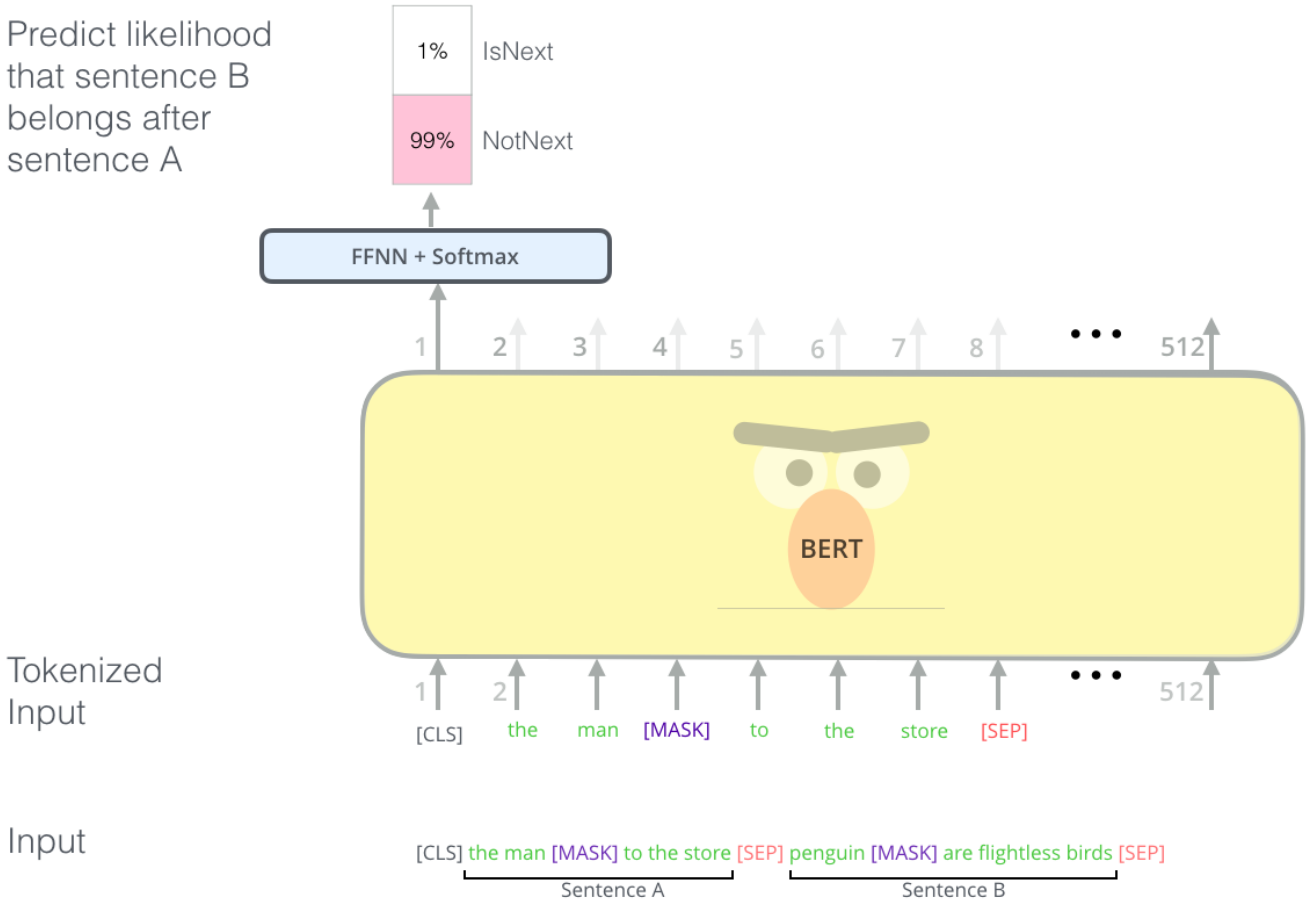


Figure: The second task BERT is pre-trained on is a two-sentence classification task. In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document.

Contextualized embeddings with BERT

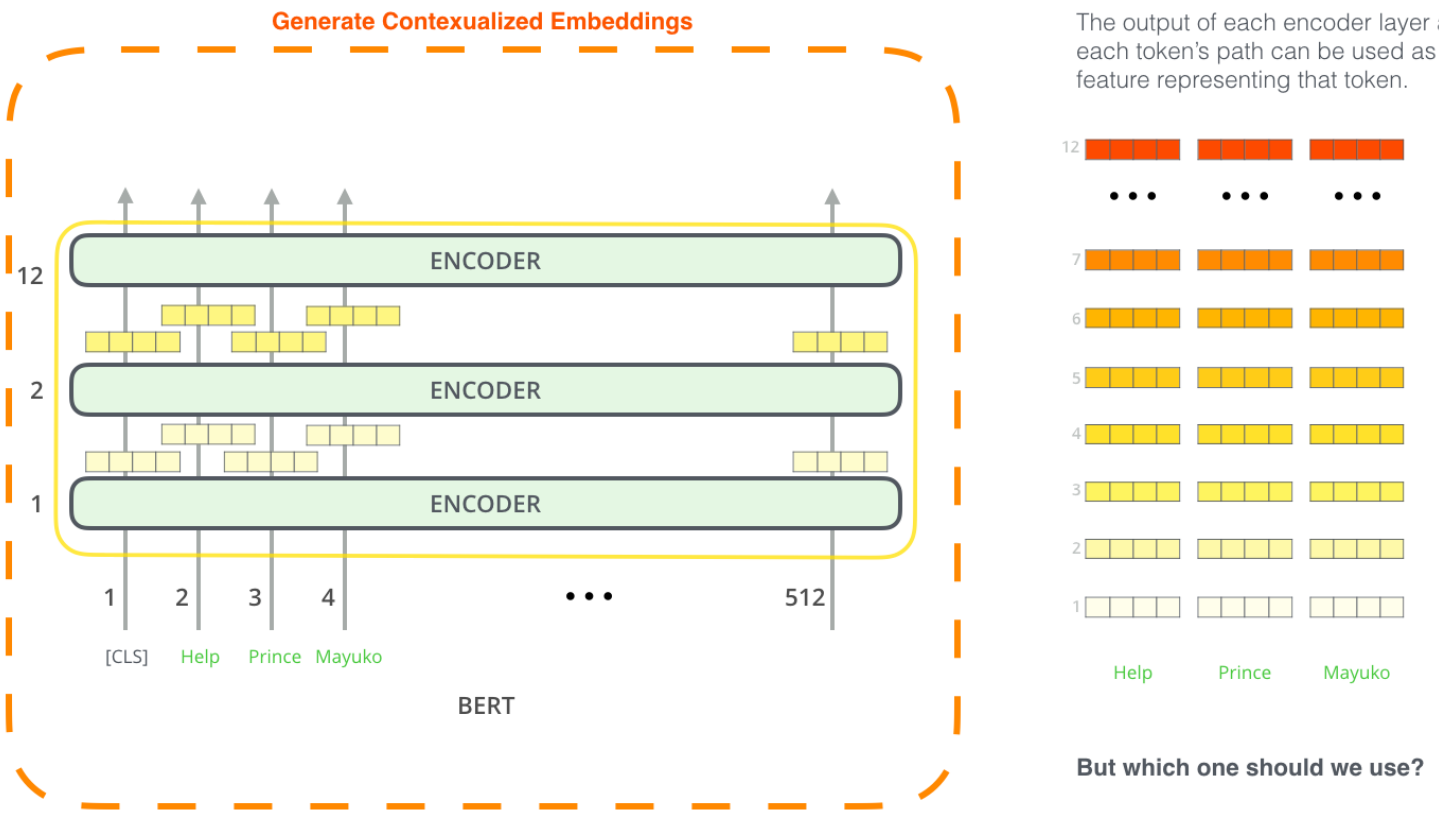


Figure: Once pre-trained, BERT's output can be used as contextualized word embeddings. One can choose among multiple options to get the embedding of a word (summing up, averaging or concatenating all or some layer's output).

Contextualized embeddings with BERT

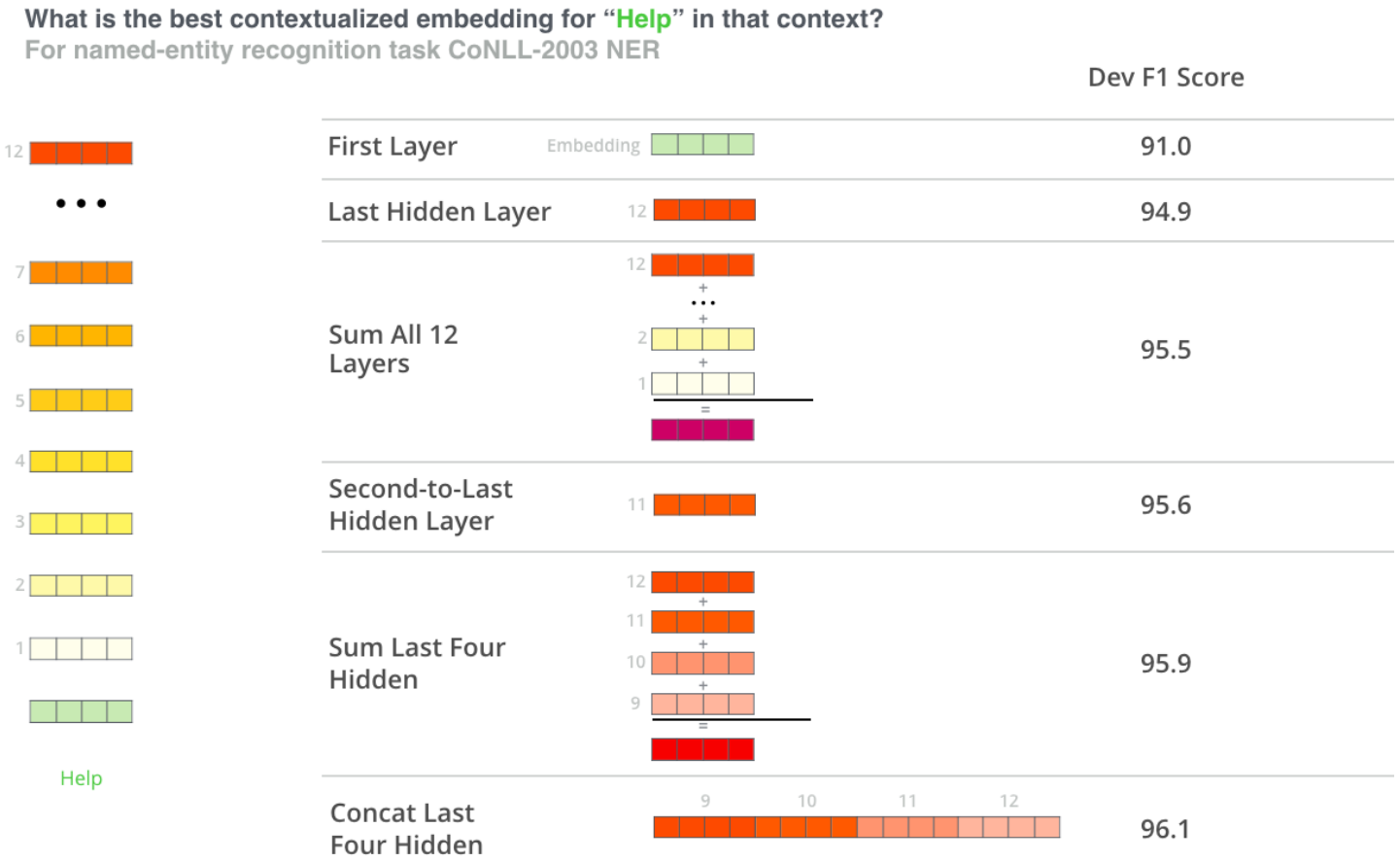
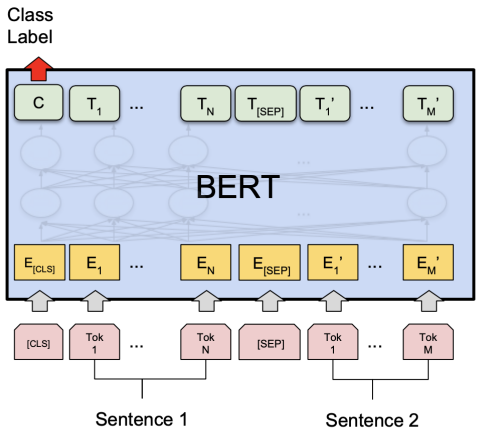
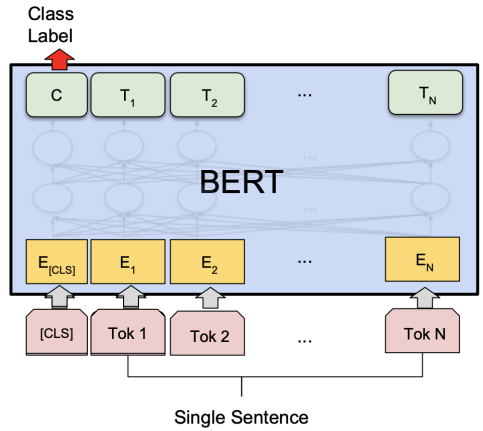


Figure: Which vector work best as contextualized embedding depends on the task. Here is an example for the task of Named Entity Recognition (NER), where six choices and their associated scores are presented in the original BERT paper.

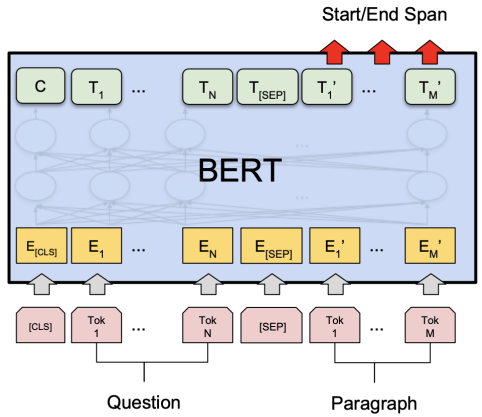
Fine-tuning BERT on downstream tasks



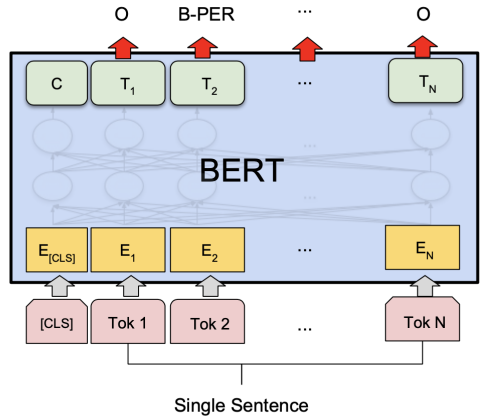
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



(c) Question Answering Tasks: SQuAD v1.1



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

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

Example: sentence classification

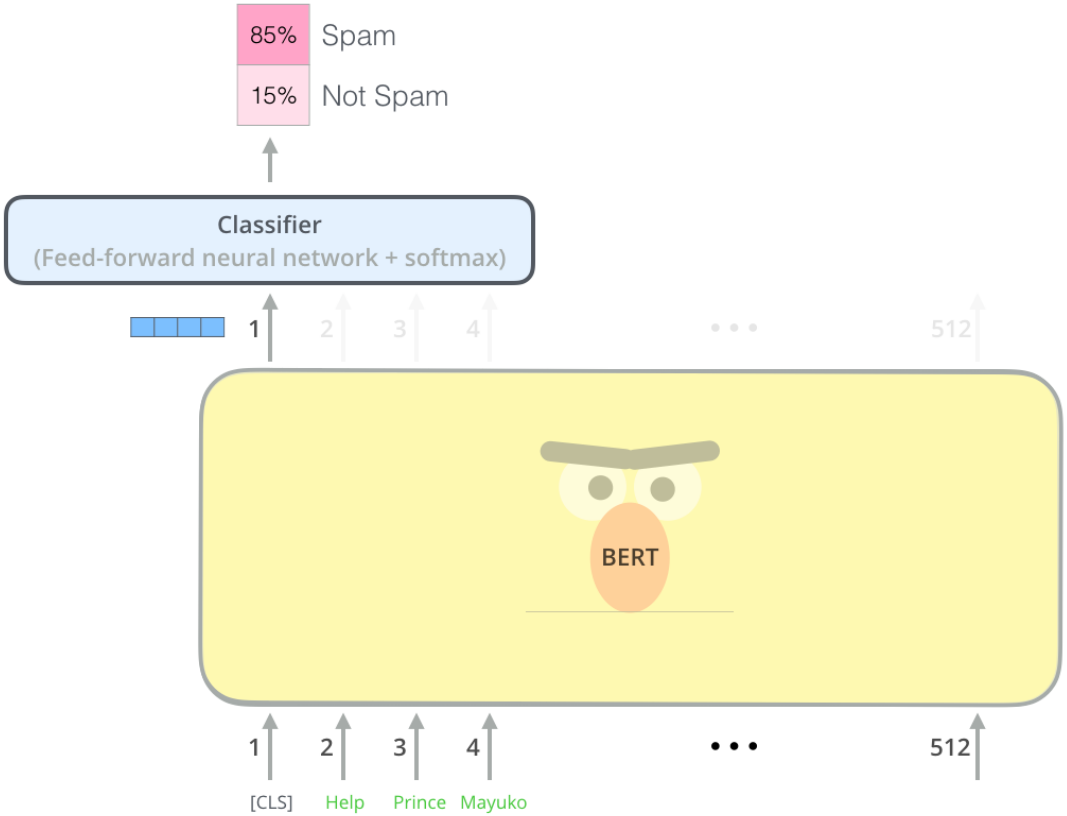


Figure: The first input token is supplied with a special [*CLS*], standing for "Classification", and used for sentence classification tasks, where only the output vector of this special token is then used as the input for a classifier. The paper achieves great results by just using a single-layer neural network as the classifier.

Conclusion

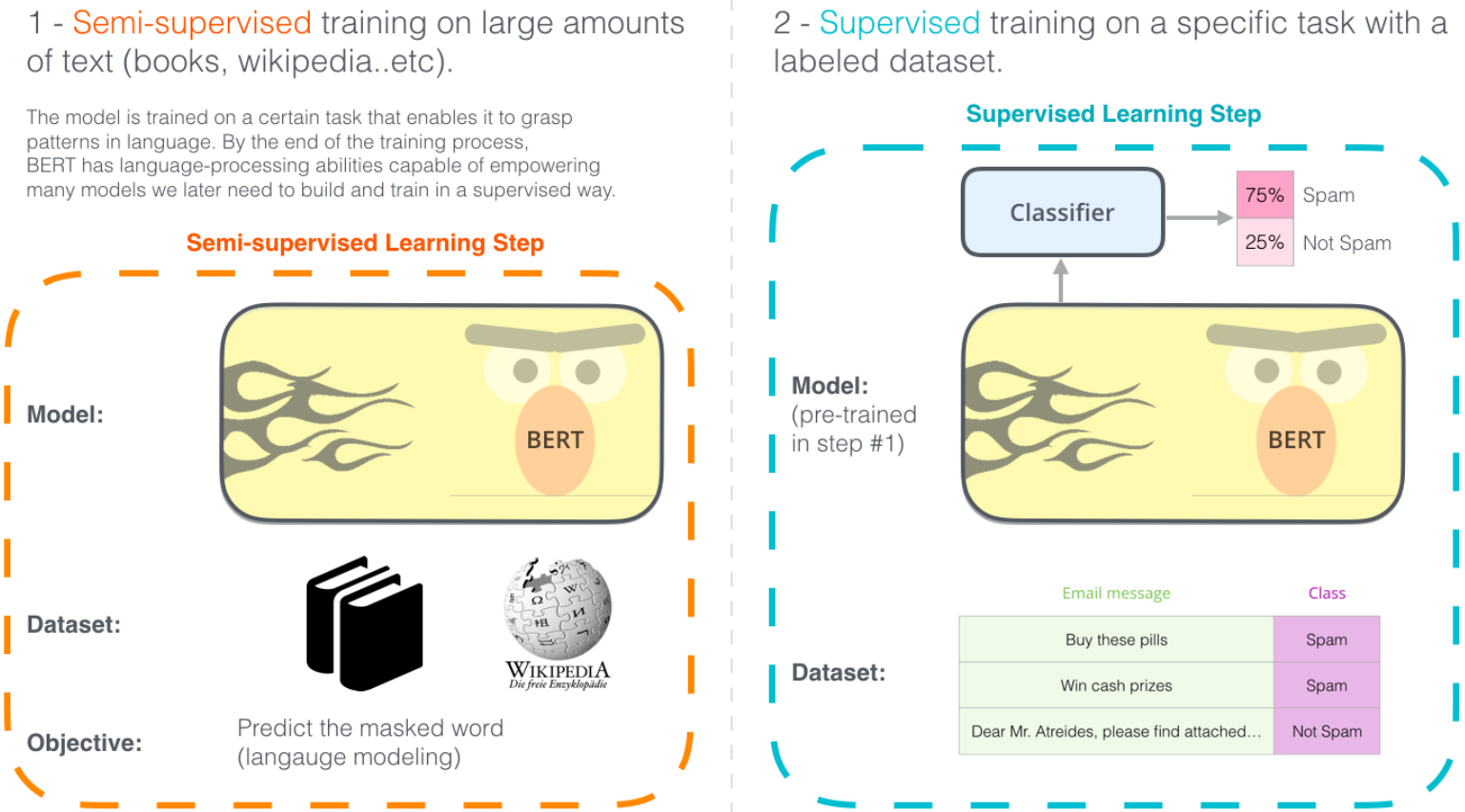


Figure: BERT is a model that has been released in late 2018 by Google, and broke several records for how well models can handle language-based tasks. The figure illustrates the two steps of how BERT is developed.

Questions ?

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]
- BERT Research Series, Chris McCormick. [6]

---

[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/*

[6] *https://www.youtube.com/playlist?list=PLam9sigHPGwOBuH4_4fr-XvDbe5uneaf6*