MO434 - Deep Learning
Transformers for Text Analysis

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

Motivation

- Processing text requires understanding the relationships between all words in a sentence, not just nearby ones.

Motivation

- Processing text requires understanding the relationships between all words in a sentence, not just nearby ones.

- Sequential models process words one at a time, creating a bottleneck: early words may be "forgotten" by the time later words are processed (long-range dependency problem).

Motivation

- Processing text requires understanding the relationships between all words in a sentence, not just nearby ones.
- Sequential models process words one at a time, creating a bottleneck: early words may be "forgotten" by the time later words are processed (long-range dependency problem).
- Transformers overcome this limitation by processing all words in parallel and using self-attention to capture relationships between any pair of words, regardless of distance [1].

Motivation

- Processing text requires understanding the relationships between all words in a sentence, not just nearby ones.
- Sequential models process words one at a time, creating a bottleneck: early words may be "forgotten" by the time later words are processed (long-range dependency problem).
- Transformers overcome this limitation by processing all words in parallel and using self-attention to capture relationships between any pair of words, regardless of distance [1].
- A transformer is a deep architecture to solve sequence-to-sequence tasks.

Motivation

- Processing text requires understanding the relationships between all words in a sentence, not just nearby ones.
- Sequential models process words one at a time, creating a bottleneck: early words may be "forgotten" by the time later words are processed (long-range dependency problem).
- Transformers overcome this limitation by processing all words in parallel and using self-attention to capture relationships between any pair of words, regardless of distance [1].
- A transformer is a deep architecture to solve sequence-to-sequence tasks.
- Let's understand how a transformer works with a language translation task.

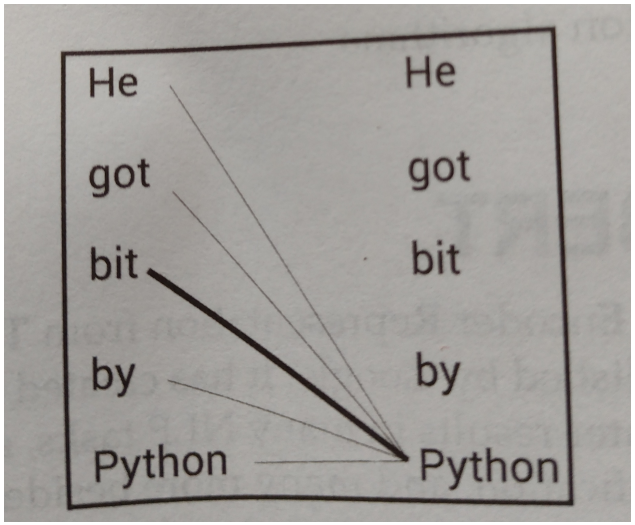
Agenda

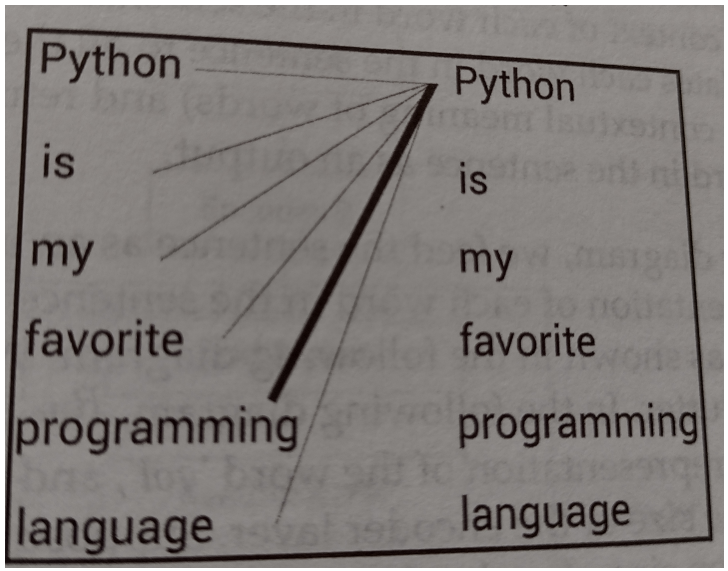
- The encoder of a transformer.
  - Positional encoding.
  - Self-attention mechanism.
  - Other operations.
- The decoder of a transformer.
  - Masked multi-head attention.
  - Cross-attention.
- Tokenization and special tokens.
- Encoder-only models (BERT) for representation learning.

The encoder of a transformer

- The encoder of a transformer is a context-based embedding model, differently from word2vec which is a context-free embedding model.

- The difference is that the former correlates each word of a sentence with the others (self-attention), generating a different representation when they have distinct meanings.

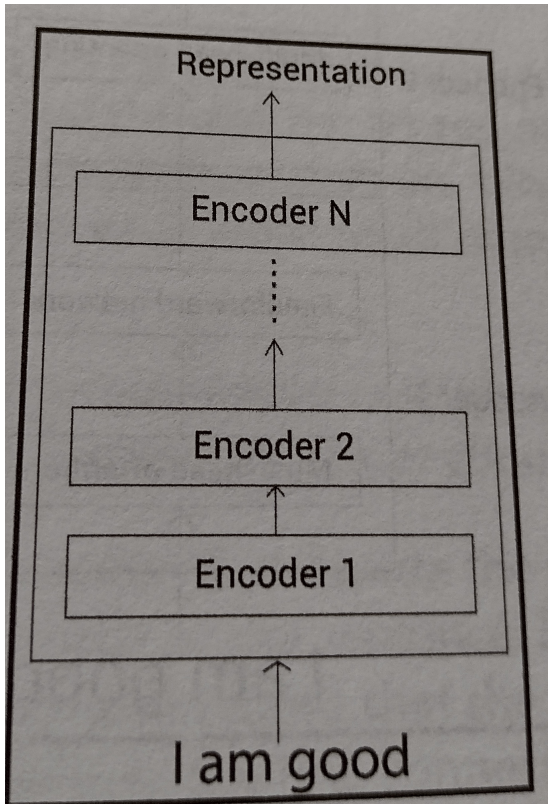


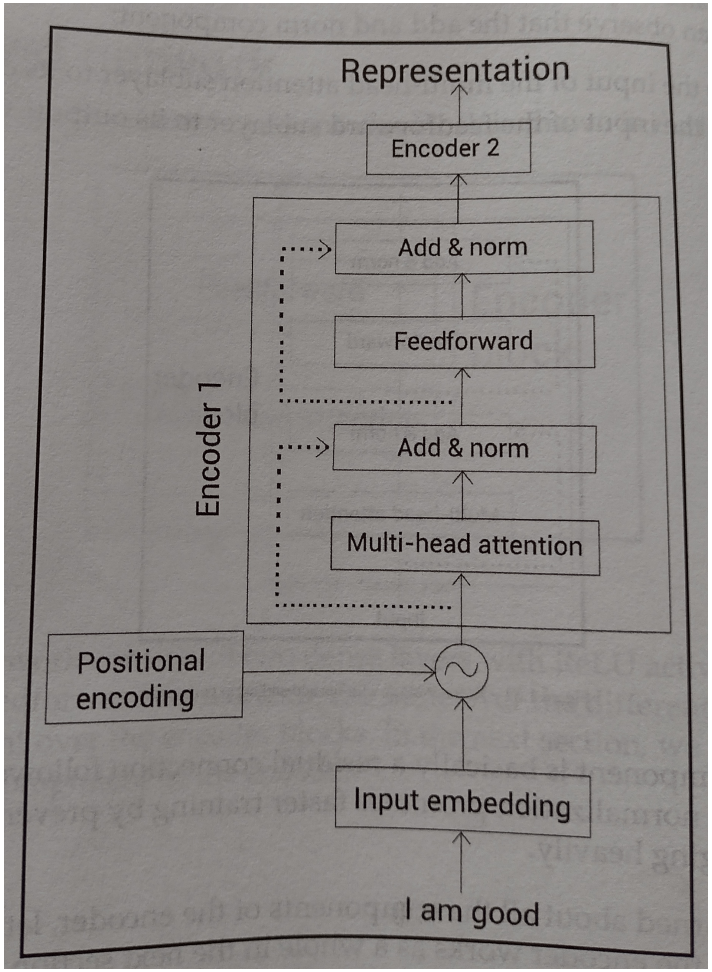


Figures from Getting Started with Google BERT from now on.

The encoder of a transformer

A transformer may have multiple encoders (left), being the configuration of each encoder as shown on the right.



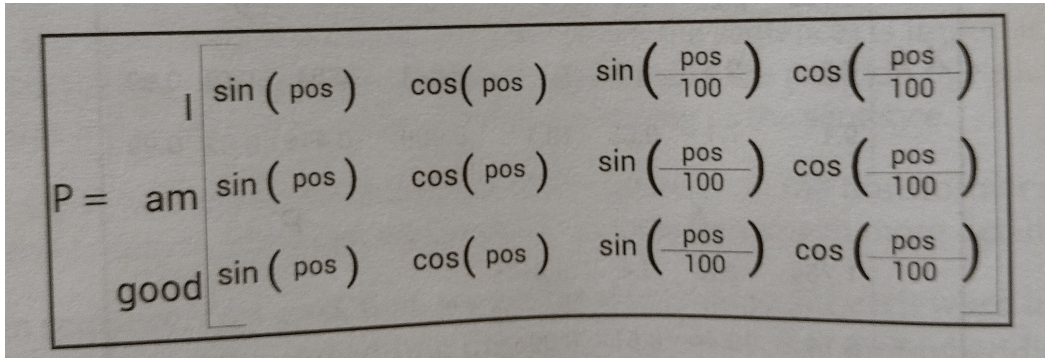


The encoder of a transformer

- An input sentence is always converted into a sequence of token ids (by a tokenizer) when inserted into the model.

- By training, a transformer learns an input embedding for each word in a sentence, forming an input matrix $\mathbf{X}$.

- However, before the self-attention mechanism, it is important to encode the position of each word in the sentence. This requires a positional encoding matrix $\mathbf{P}$ such that $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{P}$.



where pos is the position of the word in the sentence: 0 for 'I', 1 for 'am' and 2 for 'good'.

The encoder of a transformer

The self-attention mechanism requires three matrices, $\mathbf{Q}$ (query), $\mathbf{K}$ (key) and $\mathbf{V}$ (value), such that $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}^K$ and $\mathbf{V} = \mathbf{X}\mathbf{W}^V$ with the weight matrices $\mathbf{W}^*$ learned by training.

The encoder of a transformer

The self-attention mechanism requires three matrices, $\mathbf{Q}$ (query), $\mathbf{K}$ (key) and $\mathbf{V}$ (value), such that $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}^K$ and $\mathbf{V} = \mathbf{X}\mathbf{W}^V$ with the weight matrices $\mathbf{W}^*$ learned by training.

A self-attention matrix $\mathbf{Z}$ with a new embedding for each word is then defined by

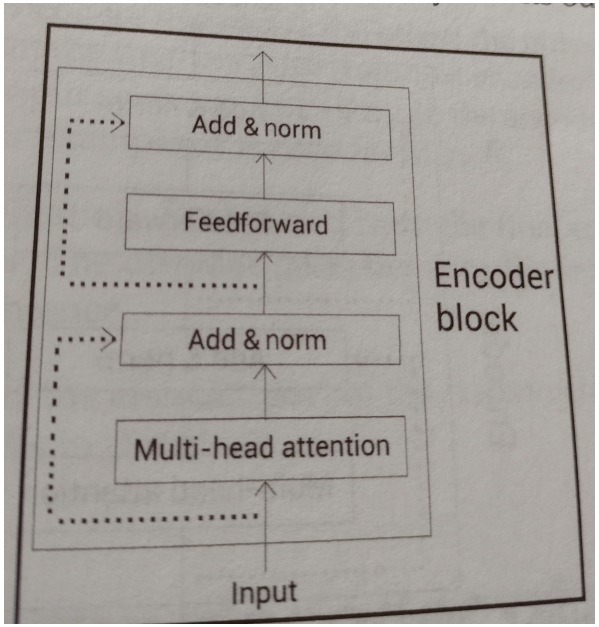
$$\mathbf{Z} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^t}{\sqrt{d}}\right)\mathbf{V},$$

where d is the embedding dimension of each word. Matrix $\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^t}{\sqrt{d}}\right)$ contains the attention weights between each pair of words in the sentence. Matrix $\mathbf{V}$ provides the content representations that get aggregated according to these attention weights, producing a context-aware embedding for each word.

The encoder of a transformer

- To treat possible ambiguities, we usually use multiple attention heads and the resulting self-attention matrices are concatenated and multiplied by another weight matrix to create the final **Z**.

- The encoder block also contains a feedforward layer with two dense layers with ReLU activation, additive skip connections, and normalization to speed up convergence.



Encoder-only models for representation learning

The encoder alone is powerful for learning contextual representations. Given a sentence, each word's embedding captures its meaning *in context*.

Encoder-only models for representation learning

The encoder alone is powerful for learning contextual representations. Given a sentence, each word's embedding captures its meaning *in context*.

- Models like BERT (Bidirectional Encoder Representations from Transformers) use only the encoder stack, trained on large corpora with masked language modeling.

Encoder-only models for representation learning

The encoder alone is powerful for learning contextual representations. Given a sentence, each word's embedding captures its meaning *in context*.

- Models like BERT (Bidirectional Encoder Representations from Transformers) use only the encoder stack, trained on large corpora with masked language modeling.
- The resulting embeddings can be used directly for downstream tasks: text classification, sentiment analysis, named entity recognition, and more.

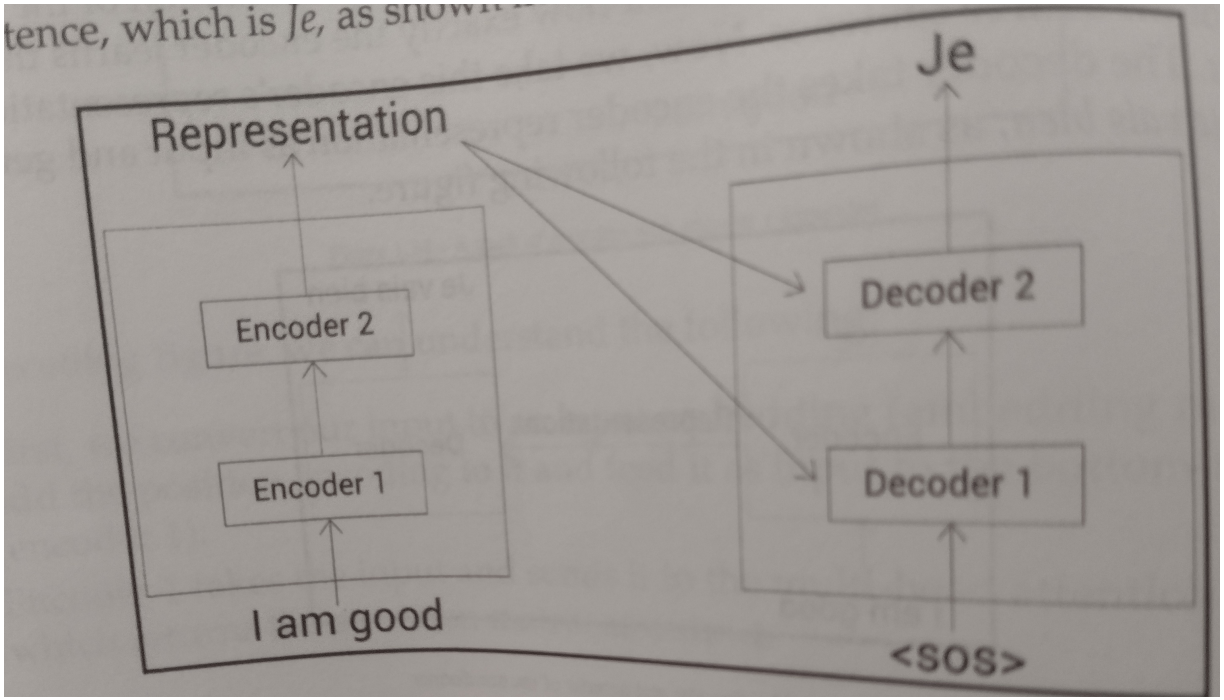
Encoder-only models for representation learning

The encoder alone is powerful for learning contextual representations. Given a sentence, each word's embedding captures its meaning *in context*.

- Models like BERT (Bidirectional Encoder Representations from Transformers) use only the encoder stack, trained on large corpora with masked language modeling.
- The resulting embeddings can be used directly for downstream tasks: text classification, sentiment analysis, named entity recognition, and more.
- This is the foundation of transfer learning for NLP: pretrain on a large corpus, then fine-tune on a smaller task-specific dataset.

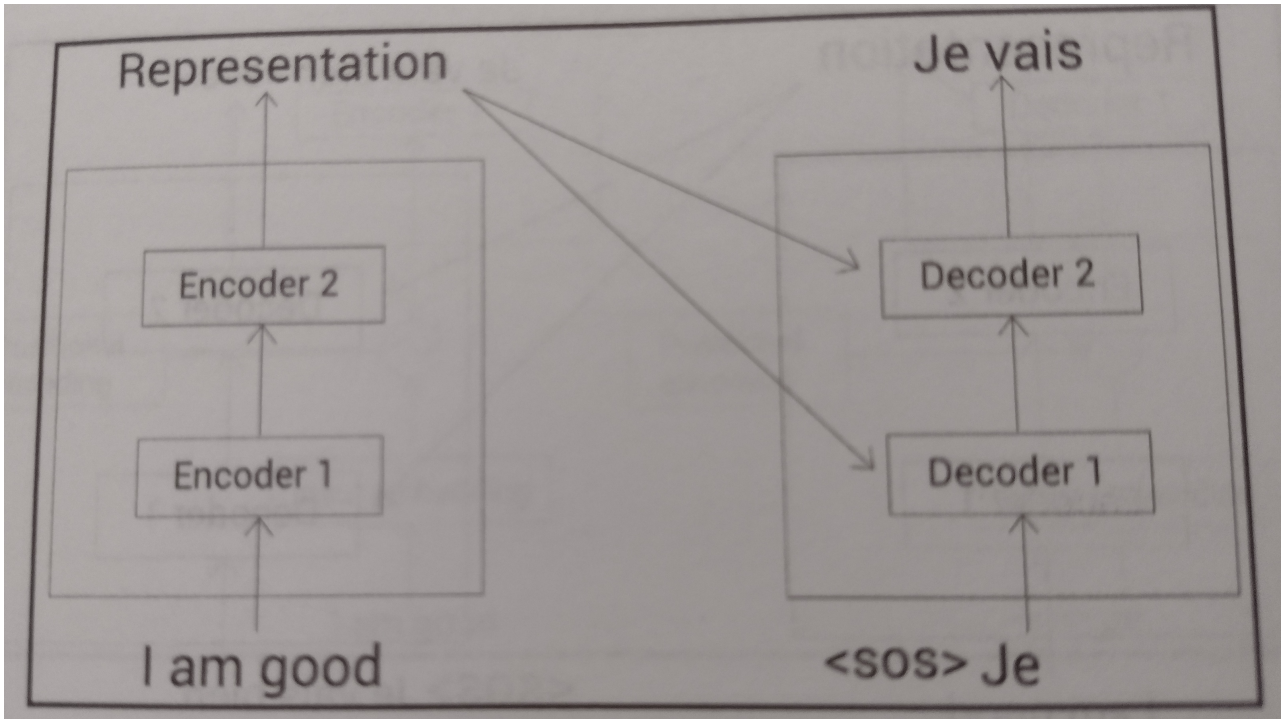
The decoder of a transformer

Similarly to the encoder, a transformer usually has a stack of decoders. At each step t, the output of step $t-1$ is used as input, being $< sos >$ and $< eos >$ the start-of-sentence and end-of-sentence tags.



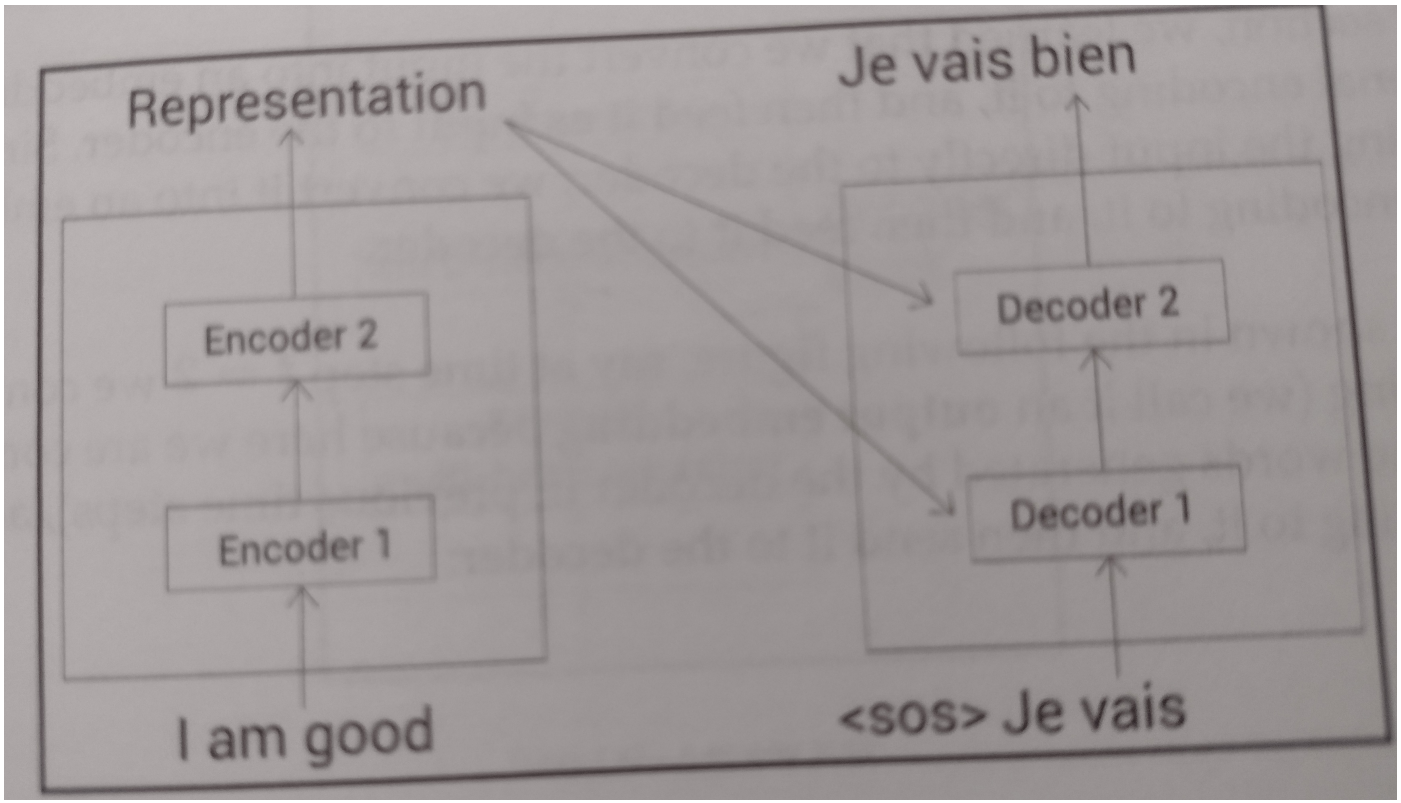
The decoder of a transformer

Similarly to the encoder, a transformer usually has a stack of decoders. At each step t, the output of step $t-1$ is used as input, being $< sos >$ and $< eos >$ the start-of-sentence and end-of-sentence tags.



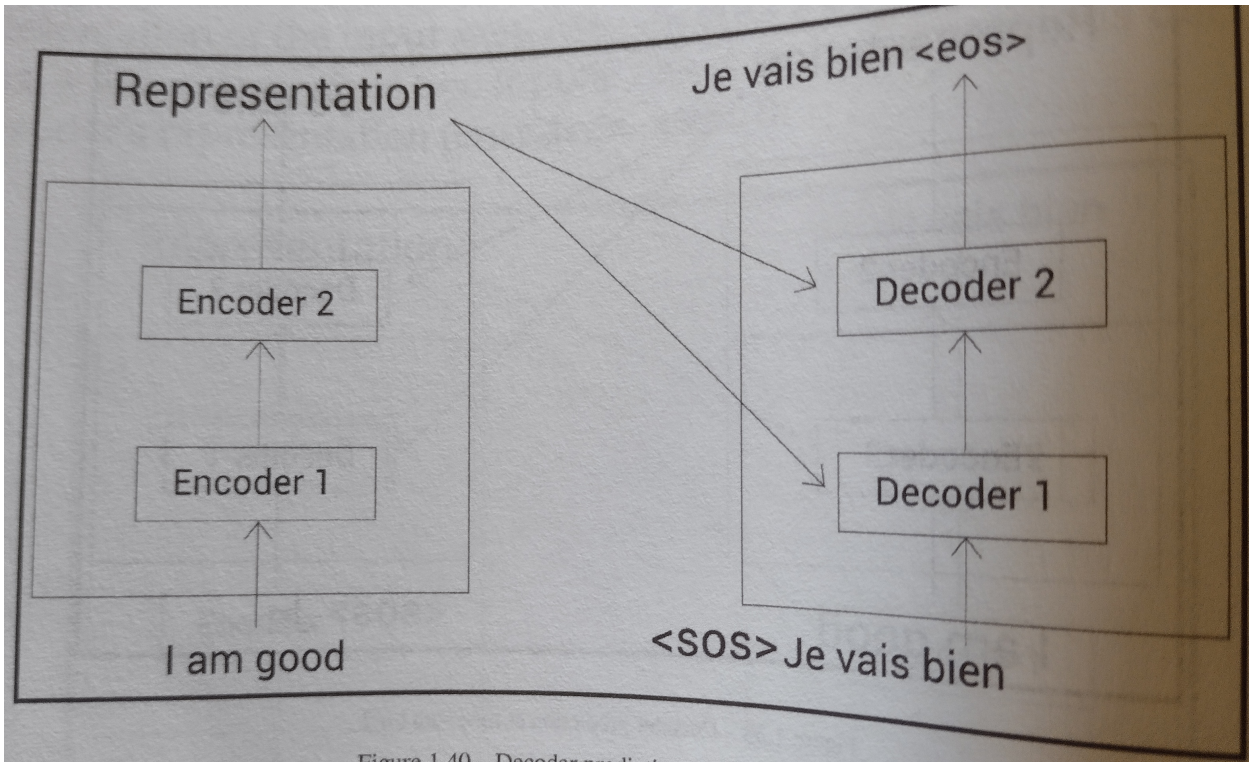
The decoder of a transformer

Similarly to the encoder, a transformer usually has a stack of decoders. At each step t, the output of step $t-1$ is used as input, being $< sos >$ and $< eos >$ the start-of-sentence and end-of-sentence tags.



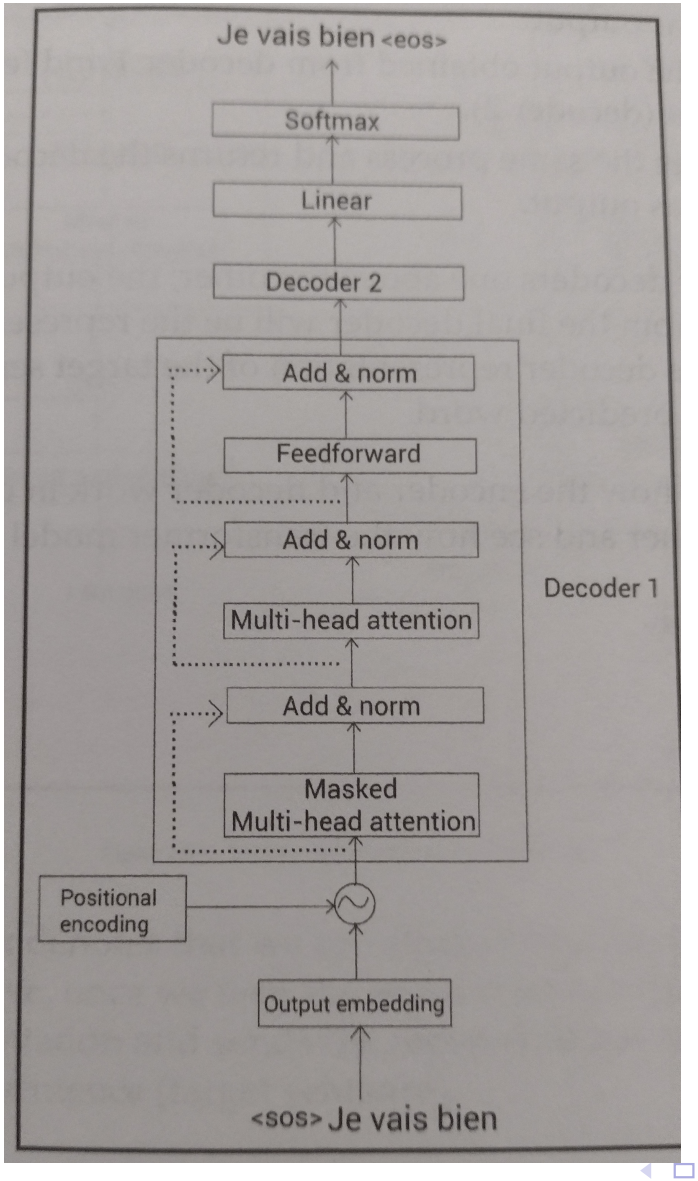
The decoder of a transformer

Similarly to the encoder, a transformer usually has a stack of decoders. At each step t, the output of step $t-1$ is used as input, being $< sos >$ and $< eos >$ the start-of-sentence and end-of-sentence tags.



The decoder of a transformer

Differences lie on both multi-head attention sublayers.



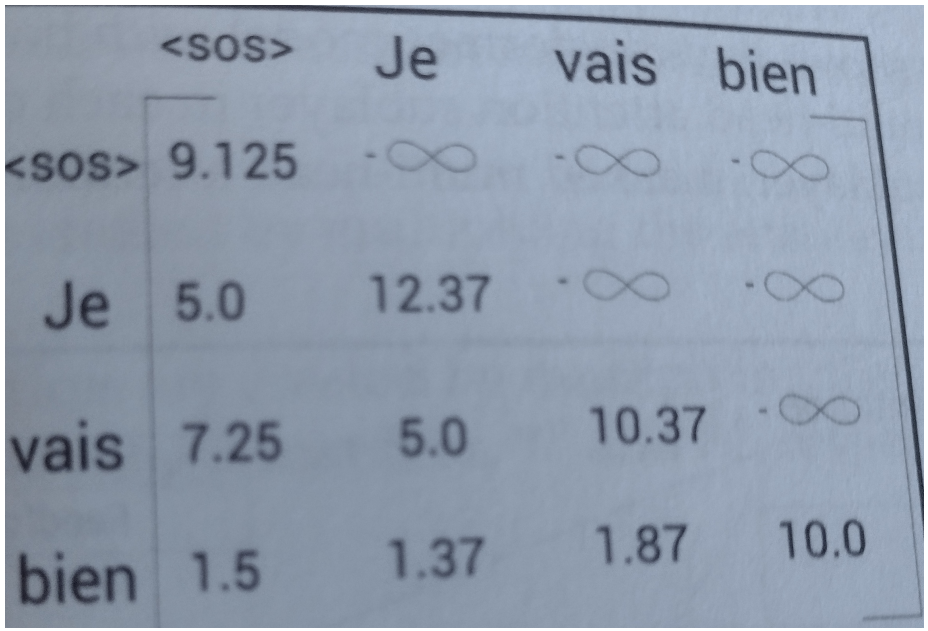
The decoder of a transformer

- A self-attention matrix of the entire input sentence $< sos >$ **Je vais bien** can be computed at each head, but it has to simulate all four steps: $< sos >$, $< sos >$ **Je**, $< sos >$ **Je vais**, and $< sos >$ **Je vais bien**.

The decoder of a transformer

- A self-attention matrix of the entire input sentence $<$ *sos* $>$ **Je vais bien** can be computed at each head, but it has to simulate all four steps: $<$ *sos* $>$, $<$ *sos* $>$ **Je**, $<$ *sos* $>$ **Je vais**, and $<$ *sos* $>$ **Je vais bien**.

- Therefore, the elements to the right of each word can be masked by $-\infty$ in each given self-attention matrix **Z**.



	<sos>	Je	vais	bien
<sos>	9.125	-∞	-∞	-∞
Je	5.0	12.37	-∞	-∞
vais	7.25	5.0	10.37	-∞
bien	1.5	1.37	1.87	10.0

The decoder of a transformer

- Again, the matrices of each head are concatenated and multiplied by a weight matrix to obtain a final matrix.

The decoder of a transformer

- Again, the matrices of each head are concatenated and multiplied by a weight matrix to obtain a final matrix.
- Now, let **M** be the output of the add&norm sublayer after masked multi-head attention.

The decoder of a transformer

- Again, the matrices of each head are concatenated and multiplied by a weight matrix to obtain a final matrix.
- Now, let **M** be the output of the add&norm sublayer after masked multi-head attention.
- The subsequent multi-head attention must be able to use **M** and the output matrix **R** from the encoder.

The decoder of a transformer

- Again, the matrices of each head are concatenated and multiplied by a weight matrix to obtain a final matrix.
- Now, let $\mathbf{M}$ be the output of the add&norm sublayer after masked multi-head attention.
- The subsequent multi-head attention must be able to use $\mathbf{M}$ and the output matrix $\mathbf{R}$ from the encoder.
- At each given head, the query, key and value matrices are $\mathbf{Q} = \mathbf{M}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{R}\mathbf{W}^K$, and $\mathbf{V} = \mathbf{R}\mathbf{W}^V$.

The decoder of a transformer

- Again, the matrices of each head are concatenated and multiplied by a weight matrix to obtain a final matrix.
- Now, let $\mathbf{M}$ be the output of the add&norm sublayer after masked multi-head attention.
- The subsequent multi-head attention must be able to use $\mathbf{M}$ and the output matrix $\mathbf{R}$ from the encoder.
- At each given head, the query, key and value matrices are $\mathbf{Q} = \mathbf{M}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{R}\mathbf{W}^K$, and $\mathbf{V} = \mathbf{R}\mathbf{W}^V$.
- As described earlier, they are used to obtain one cross-attention matrix per head, which captures the relationships between target (decoder) and source (encoder) words. Note that this is different from self-attention, where queries, keys, and values all come from the same sequence.

Tokenization strategies

Before feeding text into a transformer, it must be split into tokens. Common strategies include:

- Word-level: each word is a token. Simple, but cannot handle unknown words (out-of-vocabulary problem).

Tokenization strategies

Before feeding text into a transformer, it must be split into tokens. Common strategies include:

- Word-level: each word is a token. Simple, but cannot handle unknown words (out-of-vocabulary problem).
- Character-level: each character is a token. No unknown words, but sequences become very long and lose word-level meaning.

Tokenization strategies

Before feeding text into a transformer, it must be split into tokens. Common strategies include:

- Word-level: each word is a token. Simple, but cannot handle unknown words (out-of-vocabulary problem).
- Character-level: each character is a token. No unknown words, but sequences become very long and lose word-level meaning.
- Subword-level (BPE, WordPiece, SentencePiece): splits words into frequent subword units. Balances vocabulary size with coverage – e.g., "playing" $\rightarrow$ "play" + "##ing".

Tokenization strategies

Before feeding text into a transformer, it must be split into tokens. Common strategies include:

- Word-level: each word is a token. Simple, but cannot handle unknown words (out-of-vocabulary problem).
- Character-level: each character is a token. No unknown words, but sequences become very long and lose word-level meaning.
- Subword-level (BPE, WordPiece, SentencePiece): splits words into frequent subword units. Balances vocabulary size with coverage – e.g., "playing" $\rightarrow$ "play" + "##ing".

Most pretrained transformers (BERT, GPT) use subword tokenization. Each token is mapped to an integer id and then to a learned embedding vector.

Special tokens and the [CLS] representation

Pretrained models add special tokens to the input:

- [CLS]: a classification token prepended to every input. After passing through the encoder, its embedding aggregates information from the entire sentence – used as the sentence-level representation for classification tasks.

Special tokens and the [CLS] representation

Pretrained models add special tokens to the input:

- [CLS]: a classification token prepended to every input. After passing through the encoder, its embedding aggregates information from the entire sentence – used as the sentence-level representation for classification tasks.
- [SEP]: a separator token placed between sentence pairs (e.g., for question answering or natural language inference).

Special tokens and the [CLS] representation

Pretrained models add special tokens to the input:

- [CLS]: a classification token prepended to every input. After passing through the encoder, its embedding aggregates information from the entire sentence – used as the sentence-level representation for classification tasks.
- [SEP]: a separator token placed between sentence pairs (e.g., for question answering or natural language inference).
- [PAD]: padding token to make all sequences in a batch the same length.

Special tokens and the [CLS] representation

Pretrained models add special tokens to the input:

- [CLS]: a classification token prepended to every input. After passing through the encoder, its embedding aggregates information from the entire sentence – used as the sentence-level representation for classification tasks.
- [SEP]: a separator token placed between sentence pairs (e.g., for question answering or natural language inference).
- [PAD]: padding token to make all sequences in a batch the same length.

For text classification and sentiment analysis, we feed the **[CLS]** embedding into a dense layer – this is the basis of fine-tuning, which we will explore in the hands-on notebook.

Hands-on with transformers

Training can use cross entropy since the decoder generates a probability distribution of the words in a vocabulary.

Let's see how to use BERT from HuggingFace for text classification and sentiment analysis ▸ TEXT CLASSIFICATION AND SENTIMENT ANALYSIS.

Although this course does not address self-supervised learning, let's see ▸ tTHE DESIGN OF THE ChatGPT MODEL FROM SCRATCH.

In the next lecture, we will apply Visual Transformers to image analysis.

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin.

Attention is all you need, 2017.