Transformers

from text to images to anything

DLIM 2025-01-28

J. Chazalon

Main sources

Lecture

- https://www.tensorflow.org/text/tutorials/transformer
- T. B. Brown et al., “Language Models are Few-Shot Learners,” Jul. 2020, http://arxiv.org/abs/2005.14165
- Y. Hao et al., “Language Models are General-Purpose Interfaces,” Jun. 13, 2022, 10.48550/arXiv.2206.06336
- A. Vaswani et al., “Attention is All you Need,” in Advances in Neural Information Processing Systems 30, 2017, pp. 6000–6010. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
- Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/
- Alammar, J (2018). The Illustrated BERT. https://jalammar.github.io/illustrated-bert/
- Alammar, J (2019). The Illustrated GPT-2. https://jalammar.github.io/illustrated-gpt2/
- Voigt Godoy, D. (2022), Deep Learning with PyTorch Step-by-Step — A Beginner's Guide, https://pytorchstepbystep.com/ https://github.com/dvgodoy/PyTorchStepByStep

Lab?

- https://huggingface.co/docs/transformers/v4.48.0/en/model_doc/vision-encoder-decoder#transformers.VisionEncoderDecoderModel
- https://huggingface.co/microsoft/trocr-base-handwritten

1. Introduction

10k feet view

Deep architecture

Suitable for **many tasks**
Both extractive and abstractive – a form of general seq2seq / translation / QA framework

Used in latest **LLMs**

Revolution in **2017**, Vaswani et al. (Google) pushed the boundaries of **attention** models
Key contributions include:

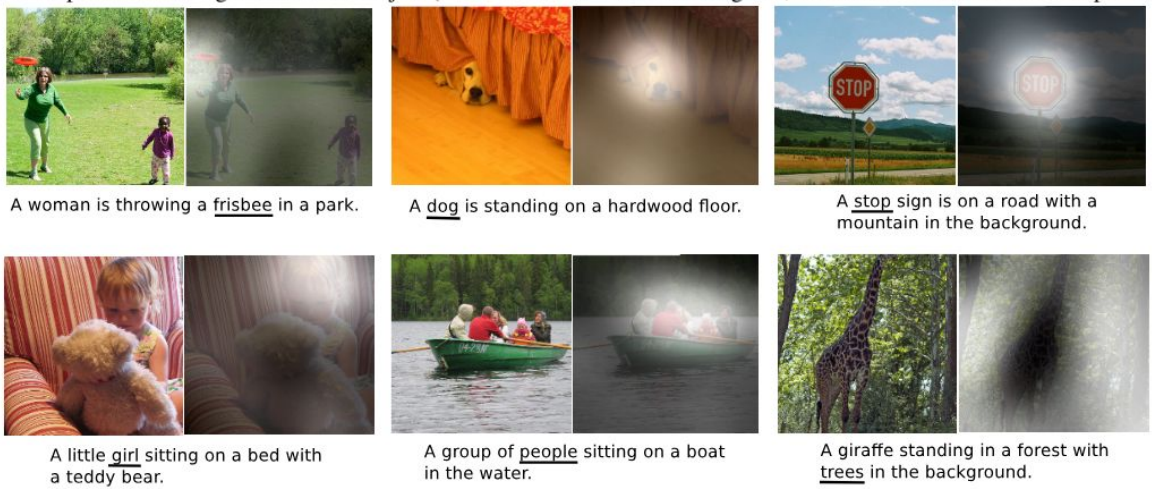
- *Encoder-decoder model*
- *Self-attention*
- *Positional embedding*

Require **large amounts of training data** and **computation power**
$O(n^2)$ wrt input length for base, full-precision attention

Vocabulary warning: “attention”

“Attention” usually refers to a process to **dynamically select** which **part of an input** should be considered in a later process, but many variants exist.

Figure 4. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



Source: K. Xu *et al.*, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. http://proceedings.mlr.press/v37/xuc15.pdf

Good read with variants illustrated: https://en.wikipedia.org/wiki/Attention_(machine_learning)

Vocabulary warning: “encoder-decoder”

Not the “encoder-decoder” we sometimes ear about in U-Net.
Note that in their original paper, **U-Net authors do not use these words**, and neither should you.

We can also sometimes ear these words in the context of autoencoders, which is not appropriate either.

We will see what “encoder-decoder” is about in a few slides.

Why Transformers are significant

Key points emphasized in Tensorflow's Transformer tutorial

- Transformers **excel at modeling sequential data**, such as natural language.
- Unlike recurrent neural networks (RNNs), Transformers are **parallelizable**. This makes them efficient on hardware like GPUs and TPUs. The main reasons is that Transformers replaced recurrence with attention, and computations can happen simultaneously. Layer outputs can be computed in parallel, instead of a series like an RNN.
- Unlike RNNs (such as seq2seq, 2014) or convolutional neural networks (CNNs) (for example, ByteNet), Transformers are able to **capture distant or long-range contexts and dependencies** in the data between distant positions in the input or output sequences. Thus, longer connections can be learned. Attention allows each location to have access to the entire input at each layer, while in RNNs and CNNs, the information needs to pass through many processing steps to move a long distance, which makes it harder to learn.
- *Transformers make no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a **set of objects** (for example, StarCraft units).*

Source: https://www.tensorflow.org/text/tutorials/transformer

What you can do with transformers

Regular ML tasks: classification, embedding, segmentation…

But also **sequence-oriented tasks**: summarization, translation…

With **advanced capabilities**: question answering, *planification*…

All of this for **text, image, sound, video…** at the same time (multimodal capabilities).

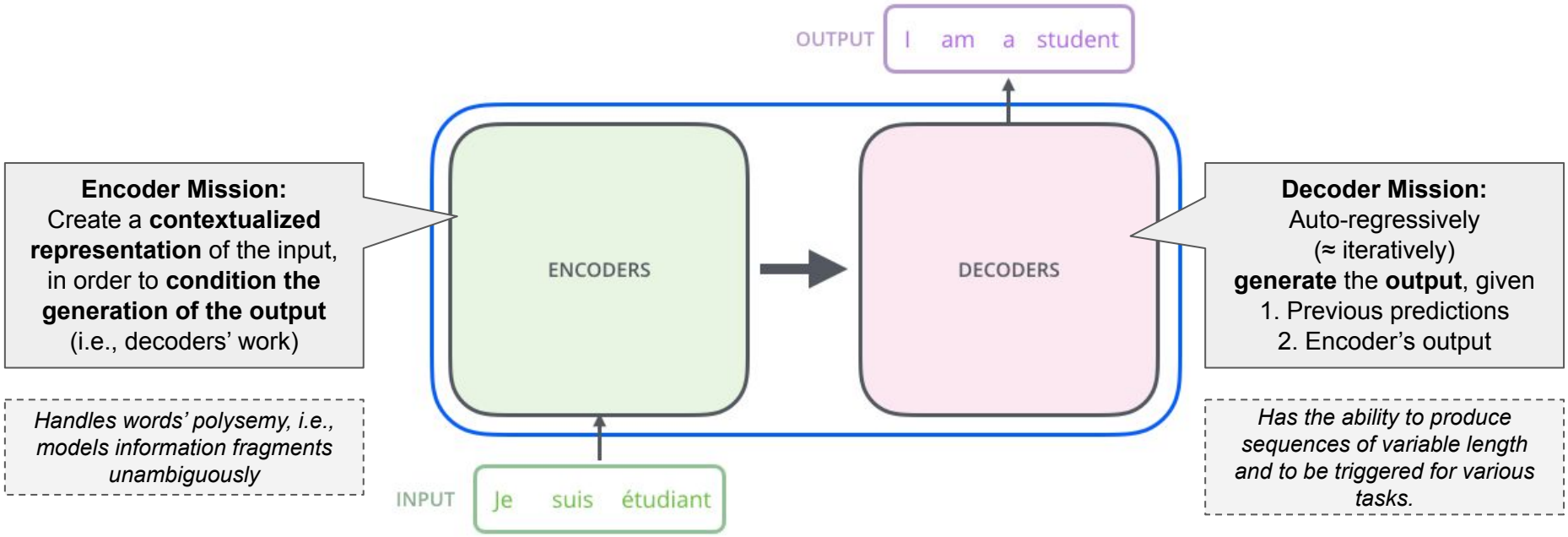
1. Transformer Architecture Overview

Looking at a translation example…

Ref. paper: A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 6000–6010.

Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

And opening the box…

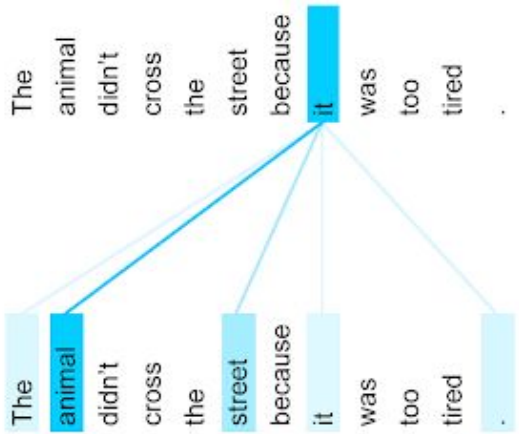


Ref. paper: A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 6000–6010.

Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Attention in action (input → input)

The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.



The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

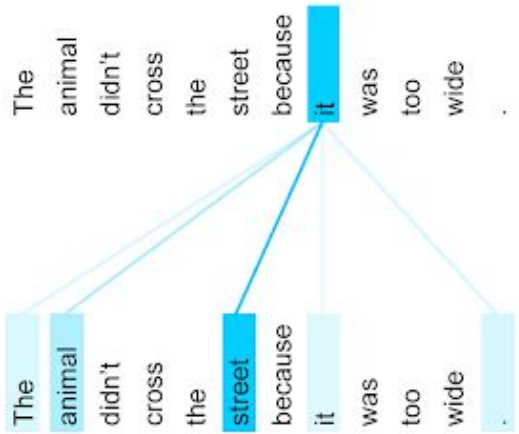
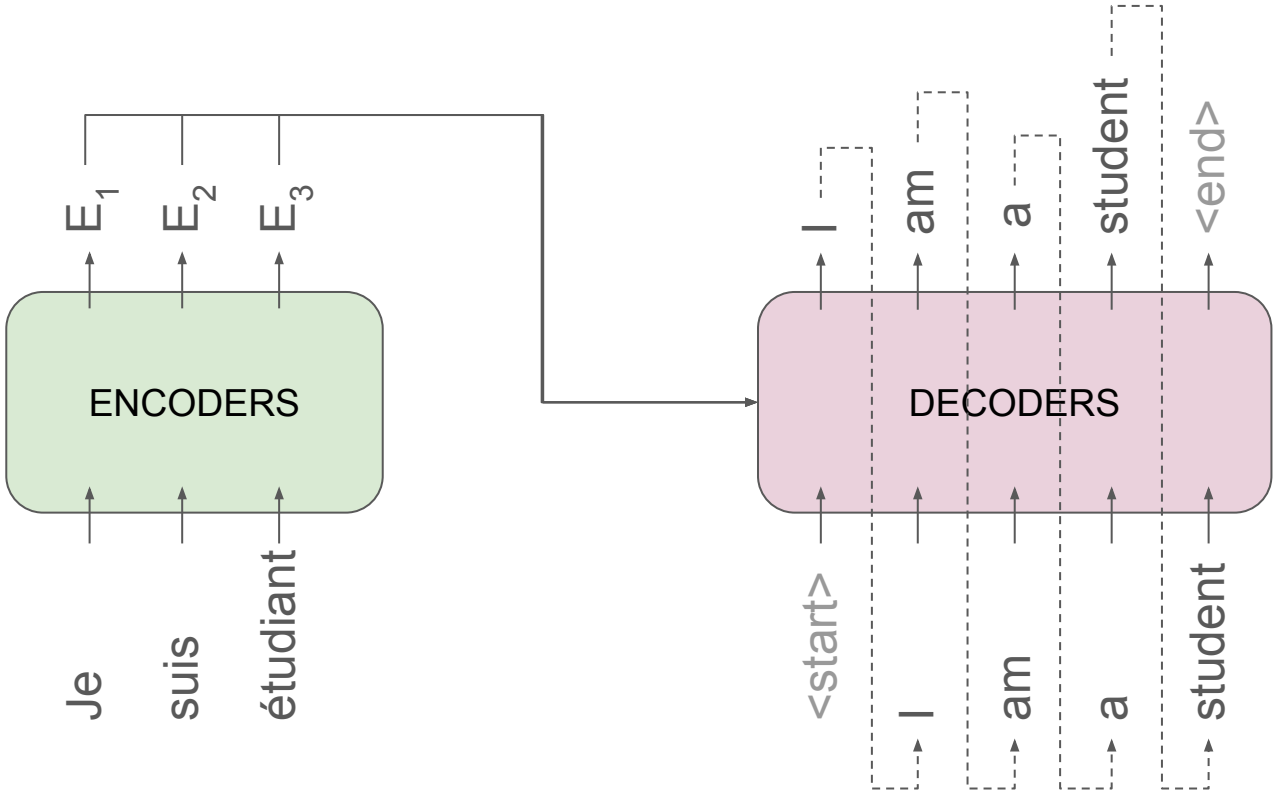


Image source: https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/

Inference (simplified illustration)

💡 Note that |in| ≠ |out|



📝 Memo:
Illustrate i→i, o→i, o→o attn on board

Digging one step further…

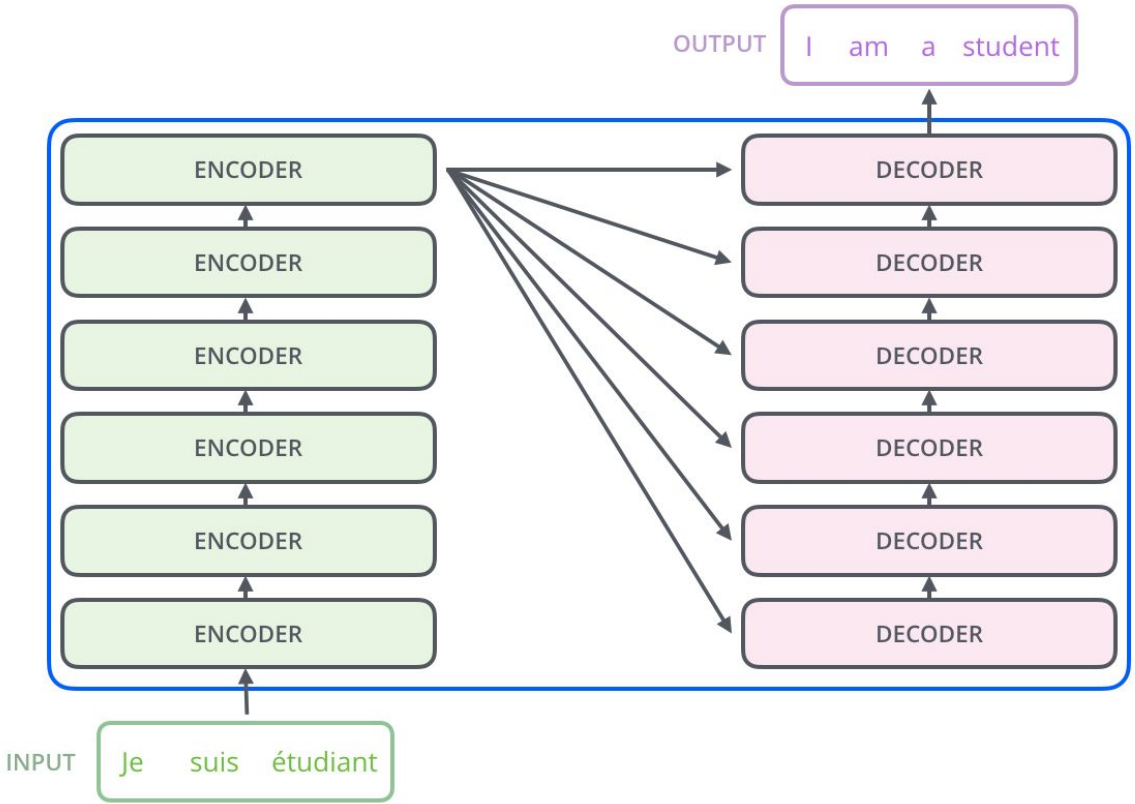


Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Here is what **cross-attention** is

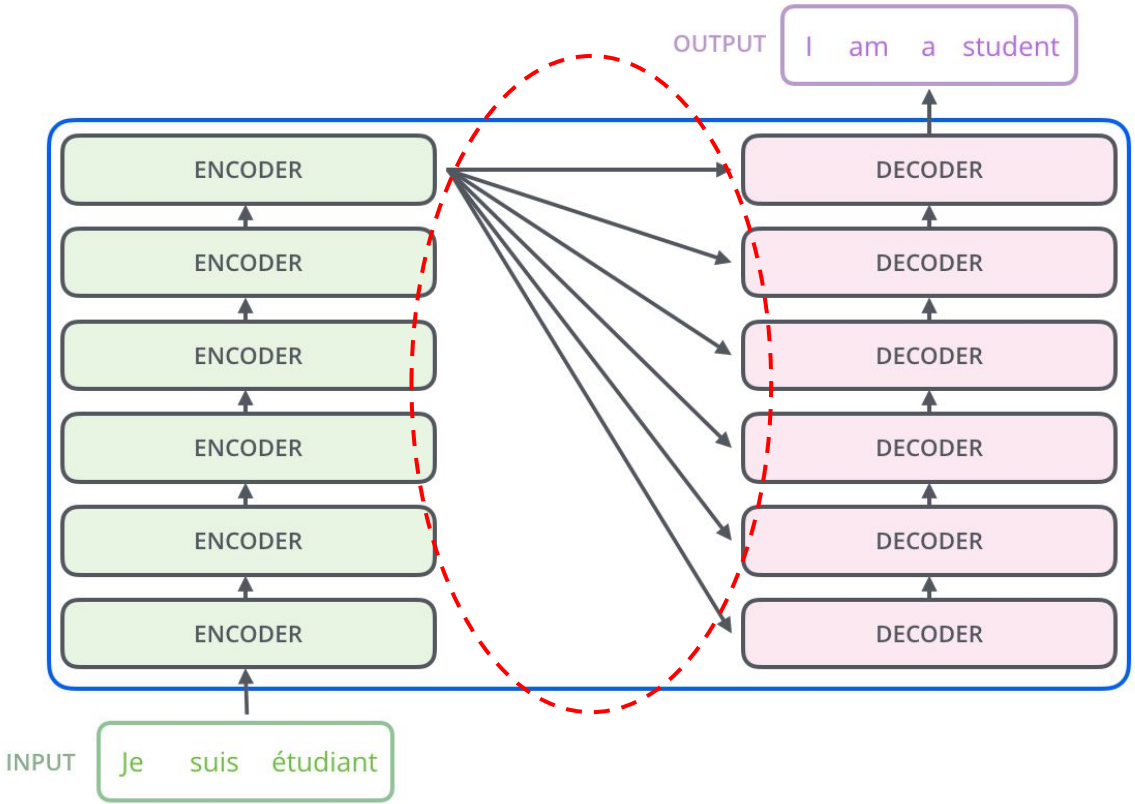
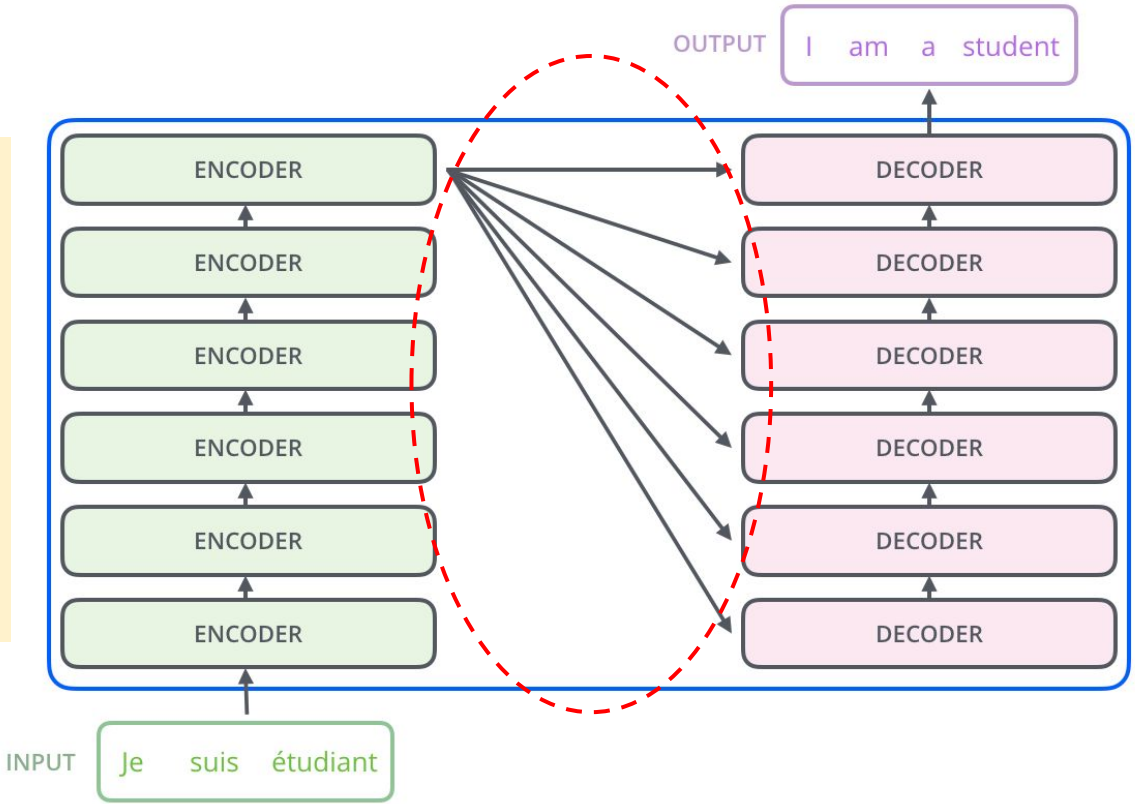


Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Here is what **cross-attention** is

Each decoder block can look at the **entire *embedded* version** of the **input***.

* input *tokens* (see later slides)



Funny fact: if you manage to take an image as input, you get the SotA architecture for 2024 OCR/HTR/HDR systems.

Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Cross attention is a form of attention

We'll see how it is computed once

- *We have studied **encoder** and **decoder** layers*
- *And understood **self-attention** principles*

So we have encoder and decoder blocks…

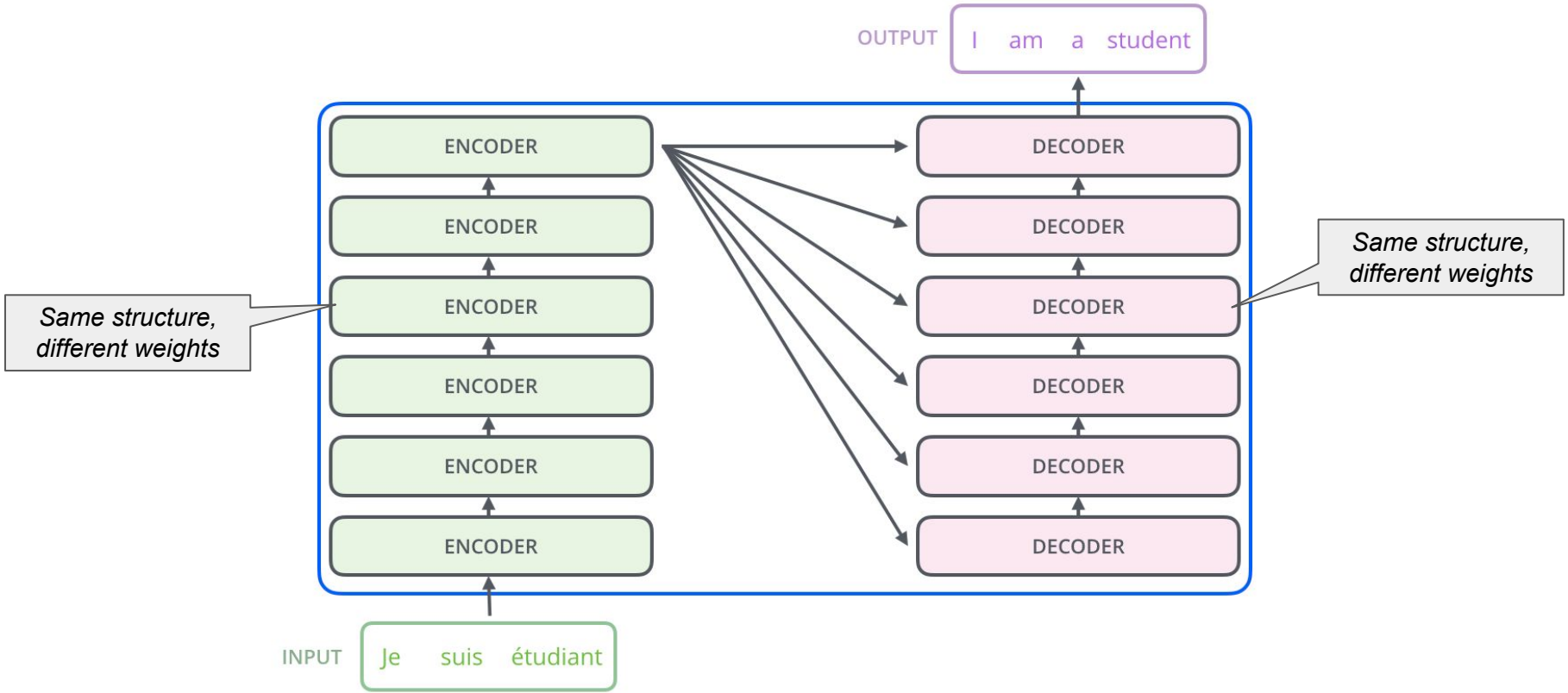


Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Encoder and decoder blocks are almost identical

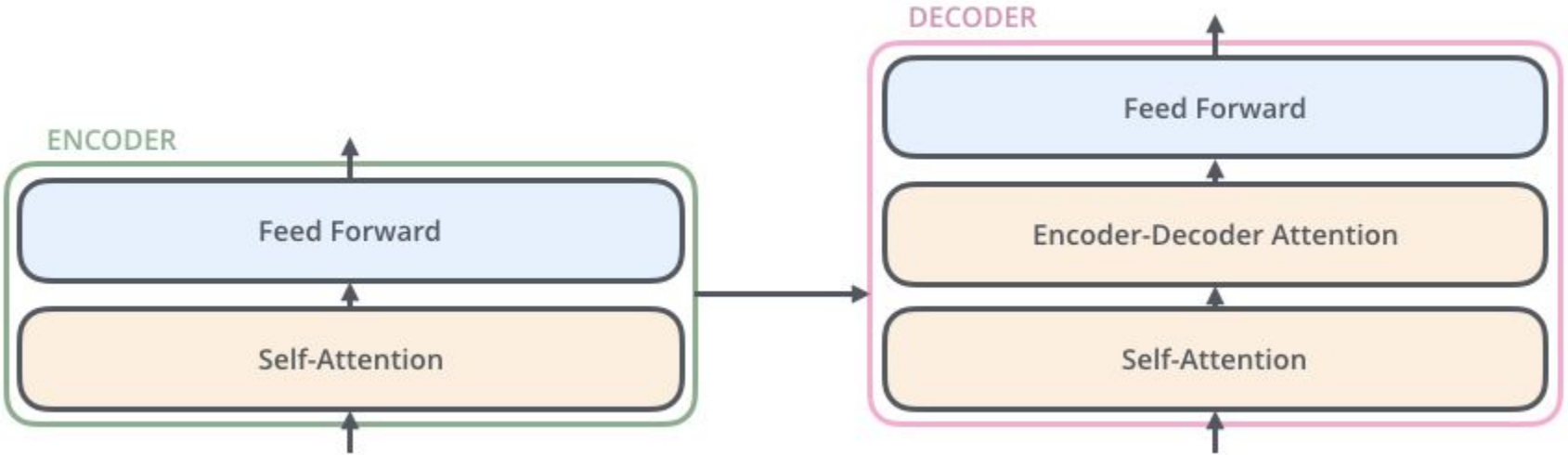
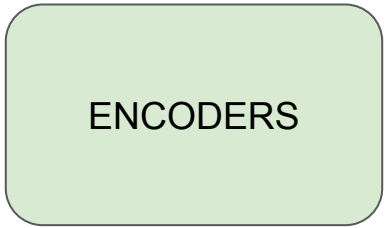
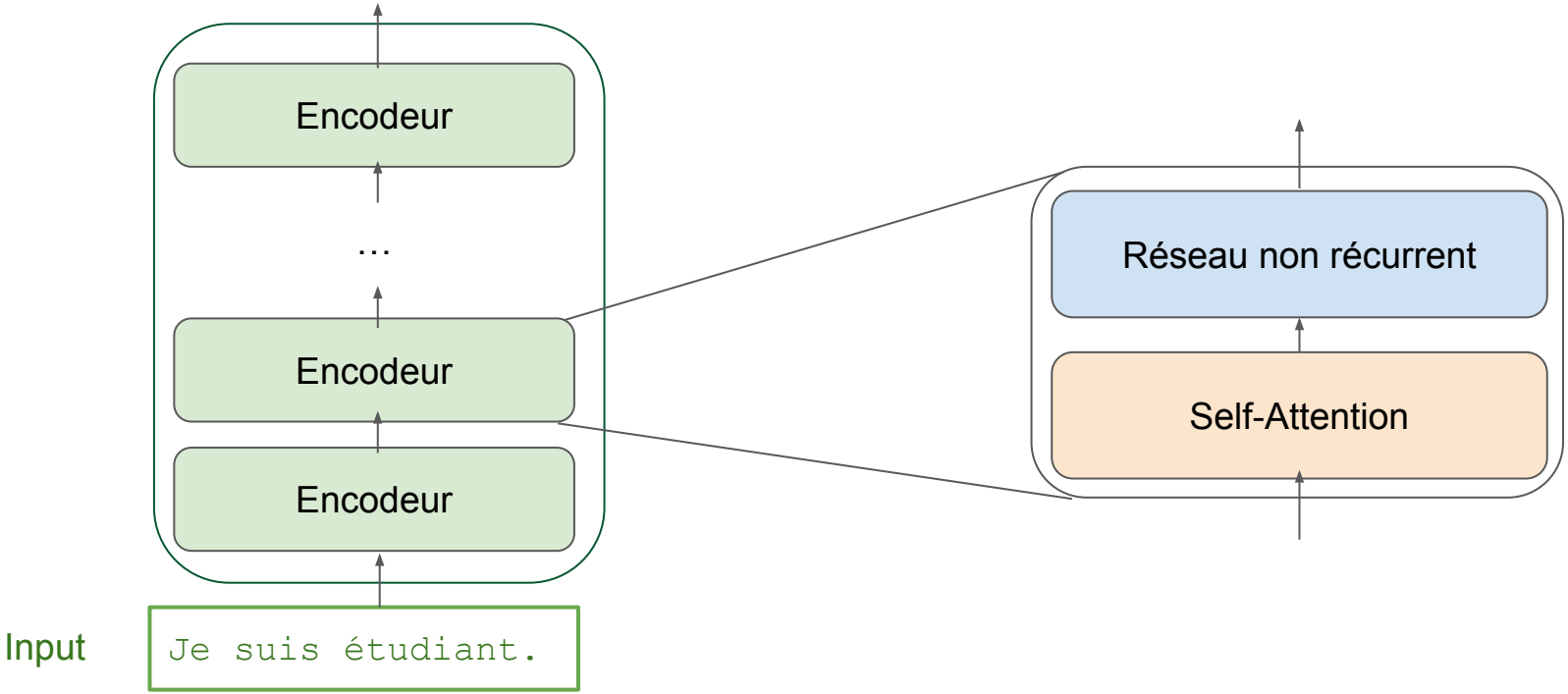


Image source: Alammar, J (2018). The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

2. Opening the boxes



Starting with the encoder side



2.1. Tokens

Neural nets can only process numerical data

So we need to encode textual information.

Bad strategy: 1 word → 1 number (problems with new words, with typos…)

Sometimes OK: 1 unicode char → 1 number (costly)

Modern tokenization: **1 text fragment → 1 number**

- **Deterministic** process
- Efficient mapping "**learned**" over training set + tricks (numbers, punctuation…)
- Standard techniques: SentencePiece, Byte Pair Encoding (BPE)
  - Techniques from the **compression** community!
- Must be **applied to inputs**, and **un-applied to outputs**
- Has to be **fast**
- Will fail to encode **unknown characters**

In practice

Good tutorial: https://huggingface.co/learn/nlp-course/chapter6/3

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("almanach/camembertv2-base")
example = "On fait des tests à l'EPITA !"
encoding = tokenizer(example)
print(" | ".join(encoding.tokens()))
# -> "[CLS] | On | fait | des | tests | à | l' | EP | ##IT | ##A | ! | [SEP]"
print(encoding.input_ids)
# -> [1, 5444, 4973, 4730, 10643, 177, 4736, 11856, 7194, 3244, 8, 2]
```

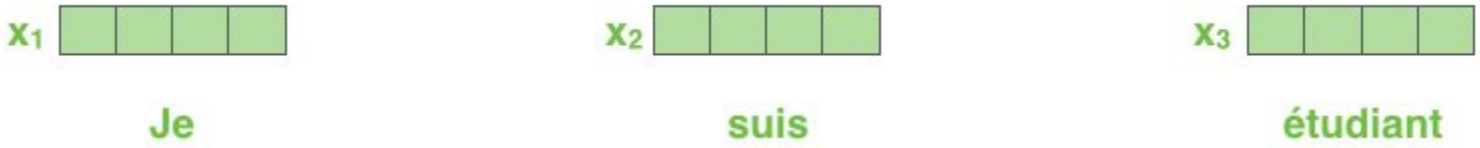
❓ So, can we now feed these numbers into our network? Not yet…

Embedding

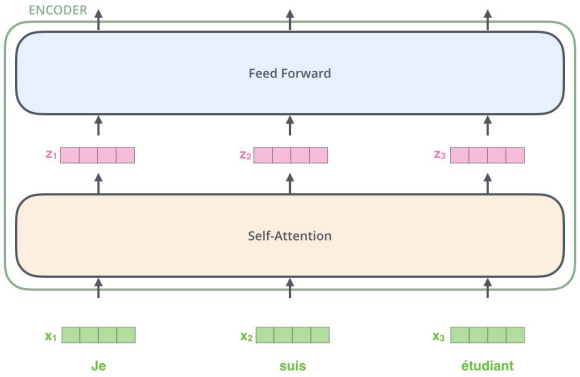
We have to **project the token ids** (integer, scalars) to a **higher dimension**.

We use a **learned embedding** as the first layer of the network.

Then, the **real input** of the network is made of a **sequence of float vectors**.

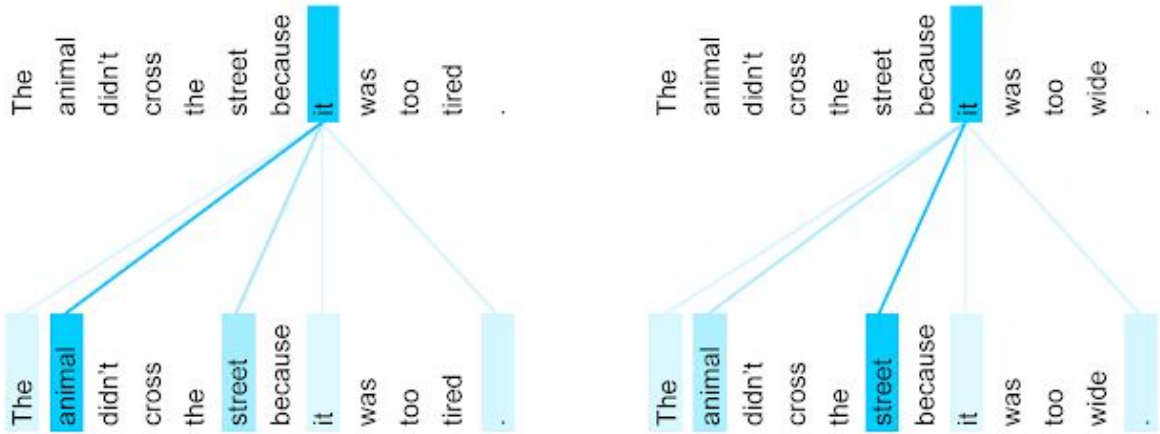


2.2. self-attention



What is the necessary context to disambiguate a token?

Many techniques, always the same underlying motivation:
→ create a new, more discriminative representation for each token



Contextual representation

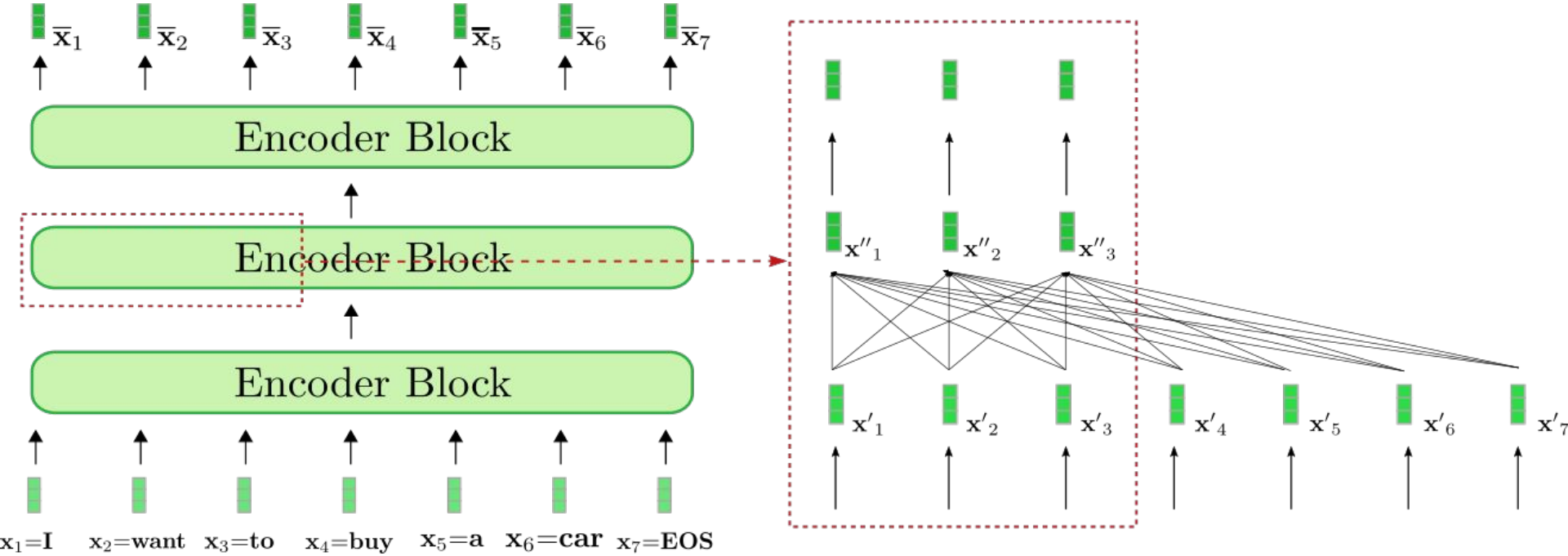
We will add “context” to isolated embeddings.

So, |in| = |out| for any encoder!!!

“Context” is **information from neighbors**, but not all of them.

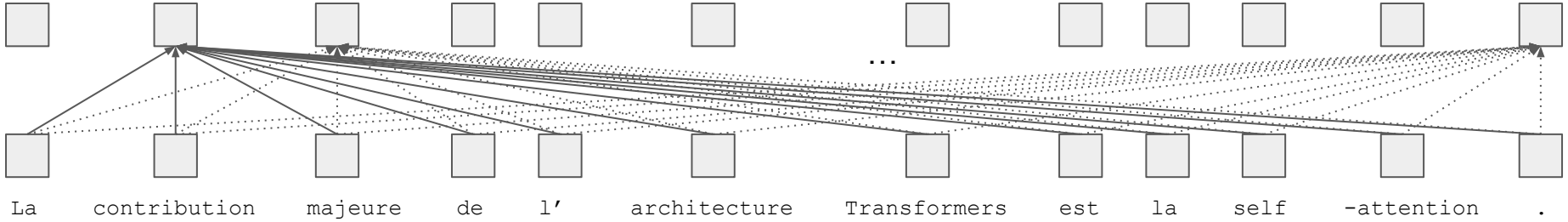
Check these two important historical techniques (not used in Transformers) if you have time:
Word2vec and GloVe: Global Vectors for Word Representation

It is all about blending x_i with $x_{j \neq i}$



https://colab.research.google.com/github/patrickvonplaten/notebooks/blob/master/Encoder_Decoder_Model.ipynb

Bad option: dense layer 👎



❌ Nombre de paramètres trop important

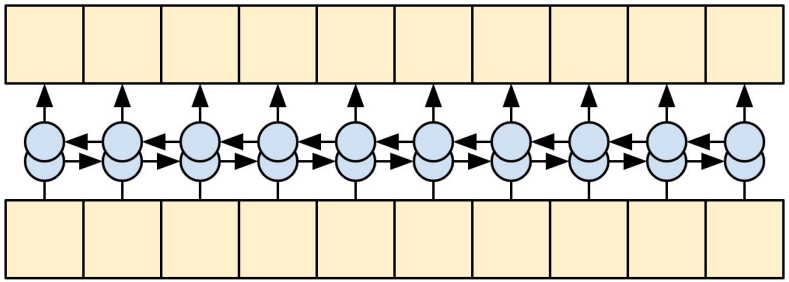
❌ impose taille fixe input

✔ Contexte complet

✔ calcul parallèle possible

⇒ Pas utilisable

Previous option 1: Recurrent Network



✔ Nombre de paramètres réduit

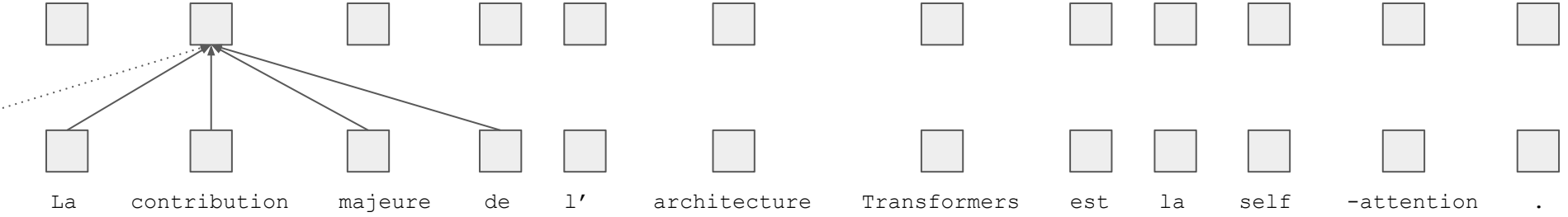
✔ Adapté aux chaînes longues

❌ Vanishing information

❌ Sequential processing

⇒ Limited information flow, and very slow (and directional, requiring forward/backward passes)

Previous option 2: CNNs



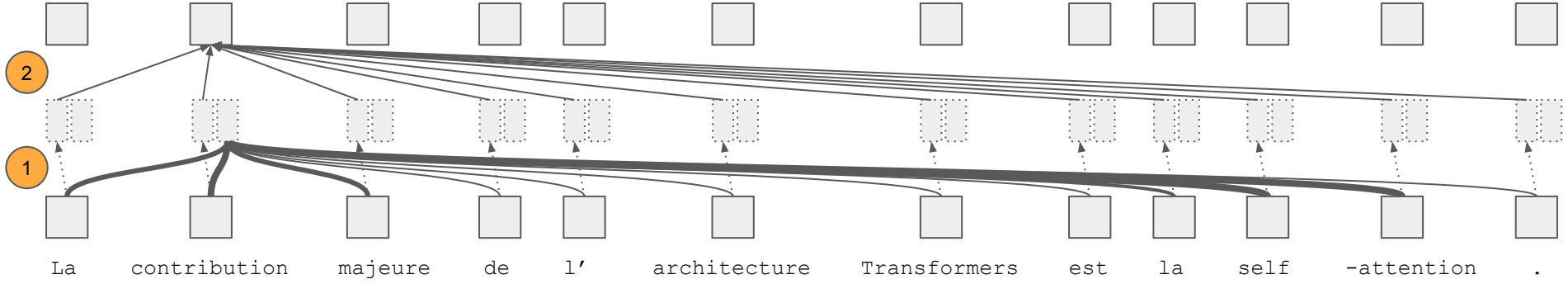
✔ Nombre de paramètres réduit

✔ Parallel evaluation

❌ Contexte limité

⇒ Impossible de relier deux termes éloignés (pas considérés simultanément — limited receptive field)

Entering *self-attention*



Processus en 2 étapes

1. *Pondération du voisinage*
2. *Fusion selon la pondération*

<u>À présent</u> : Couche de *self-attention*

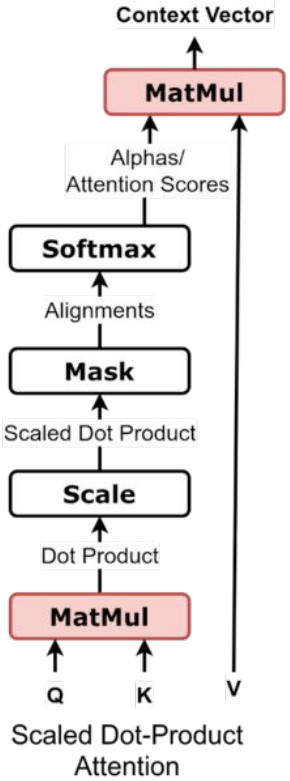
✔ Nombre de paramètres réduit

✔ Contexte très large

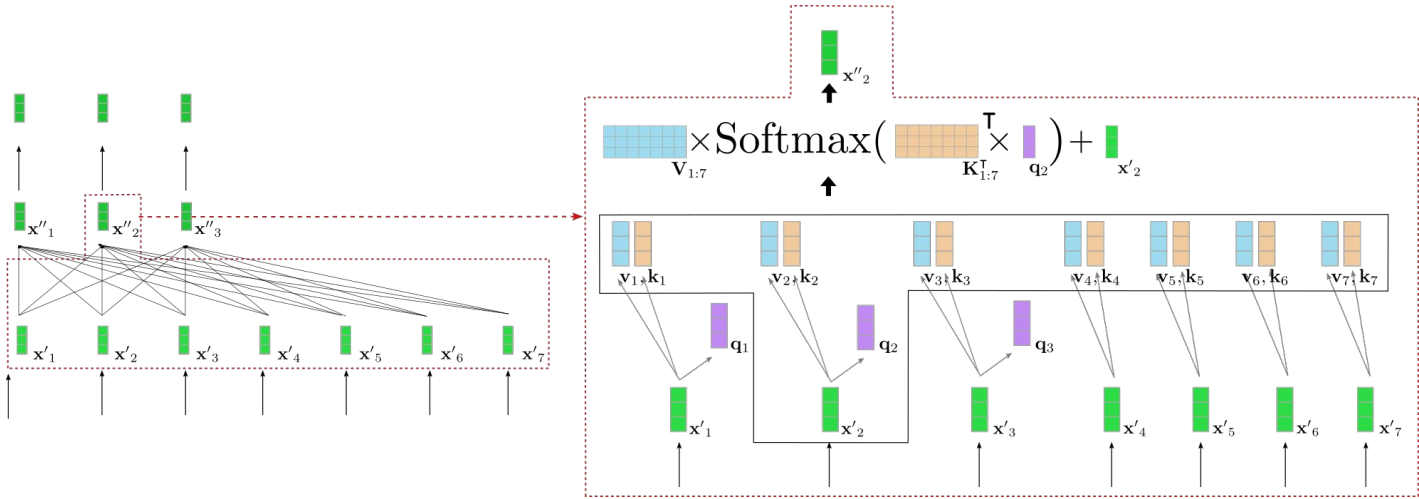
✔ Calcul parallèle

⇒ Possible de relier deux termes éloignés précisément

Implementation: use Queries, Keys and Values



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V$$



But keep in mind that we need to compare all x_i to $x_{j \neq i}$

So, complexity of attention computation is **$O(n^2)$ wrt the size of the sequence**.

This is one of the reasons why Transformers can be **slow to train**.

Many tricks to mitigate this:

- Limited attention span
- Architecture variants
- Better implementation (Flash Attention)

2.3. Multi-Head Attention

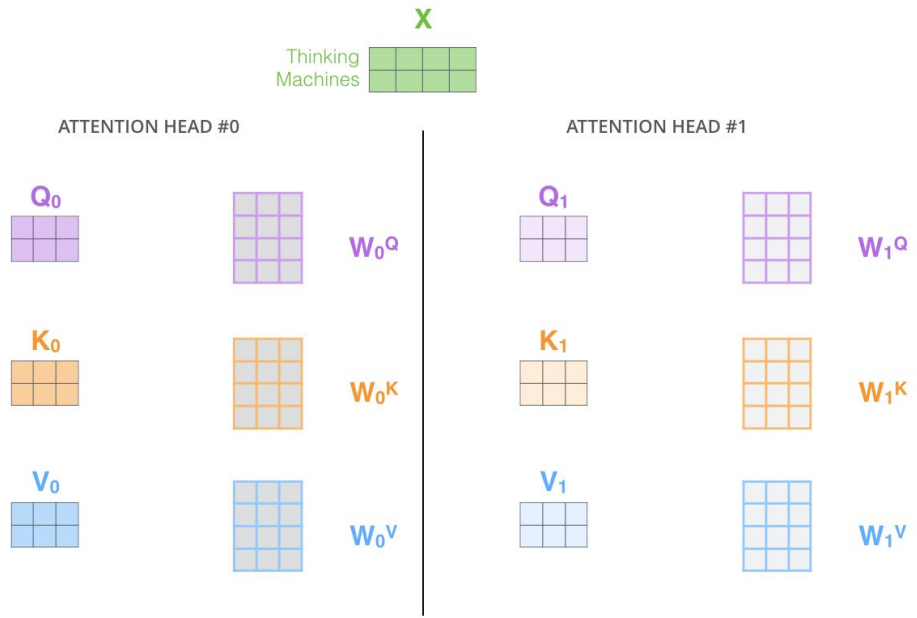
Different contexts for different purposes

Uses several **attention heads in parallel** to capture **different aspects** of the relation between tokens, automatically.

Can be linked to

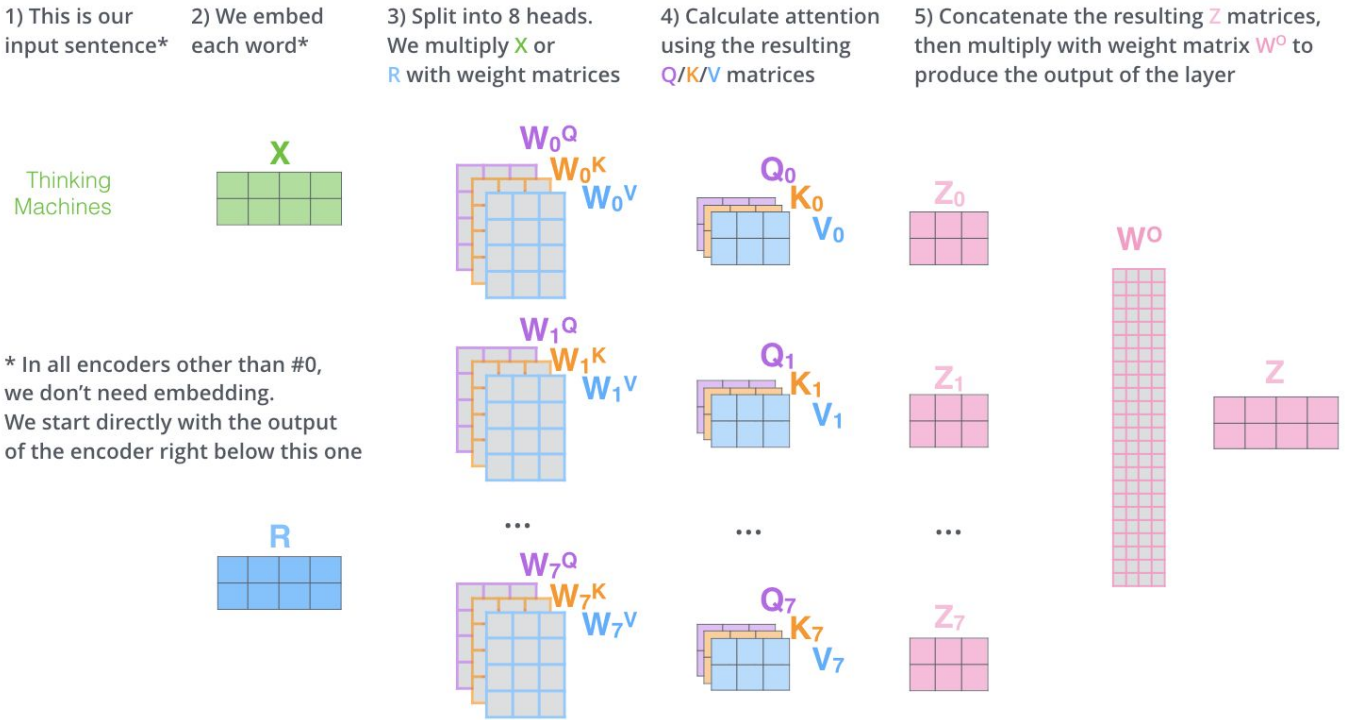
- relative positions in sentence*
- Coreferences
- Same polarity
- subject-object relationship
- and many others…
- and a blend of all of this.

*more on position in a moment



Aggregating values from multiple heads

Concatenate. Mix/select. Done.



Almost finished an encoder block…

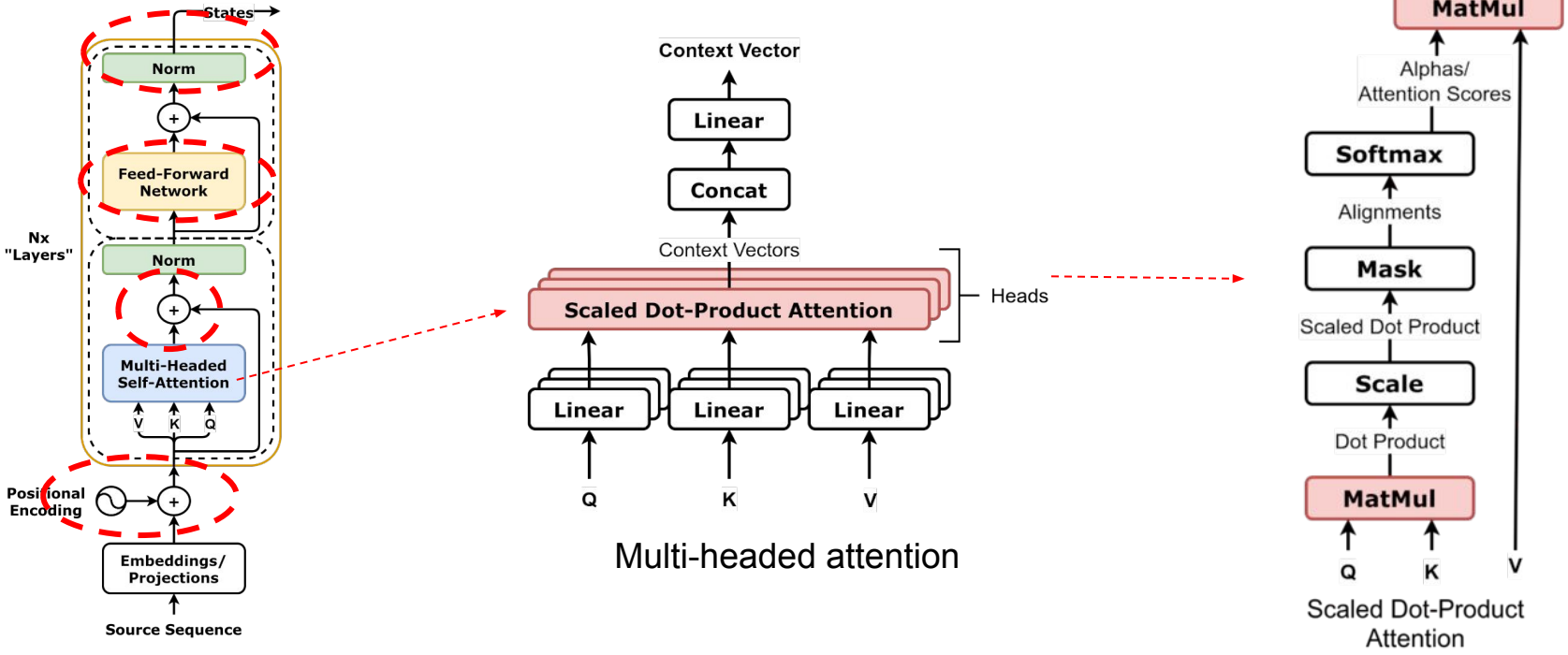


Image source: Voigt Godoy, D. dvgodoy / CC BY / https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

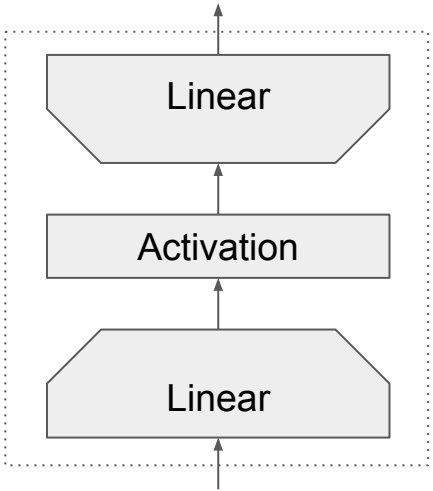
2.4. Feed forward network

A simple 2-layer network

$FFN(x) = \sigma(X\mathbf{W}_1+\mathbf{B}_1)\mathbf{W}_2+\mathbf{B}_2$

Applied on **each x_i separately: not a dense layer**

Mixes information feature-wise for each token.



2.5. Residuals

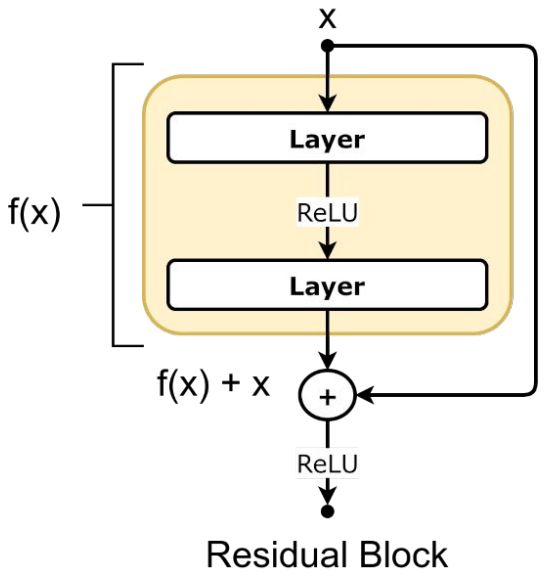
Residuals are standard since ResNet

Makes it easier to capture complex, uncommon, high-frequency patterns:

1. Model the general case
2. Add the output to the original signal

The model will learn to perform a difference if needed.

In any case, it helps gradient flow!



2.6. Normalization

Keep value domains sane using LayerNorm

Can be applied either before or after the main layers (attention, FFN).

Several techniques, BatchNorm was common before Transformers which uses LayerNorm.

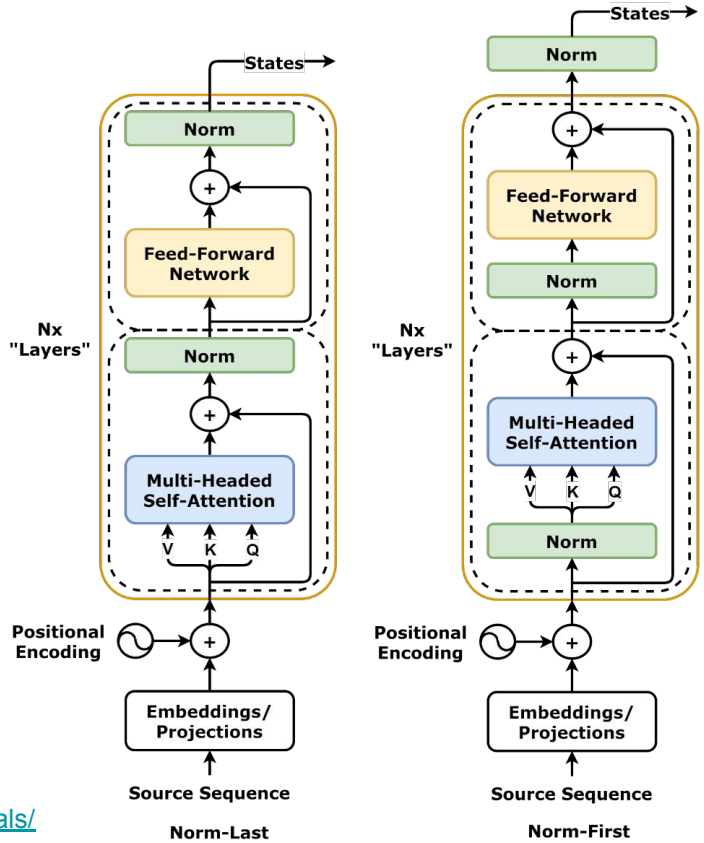
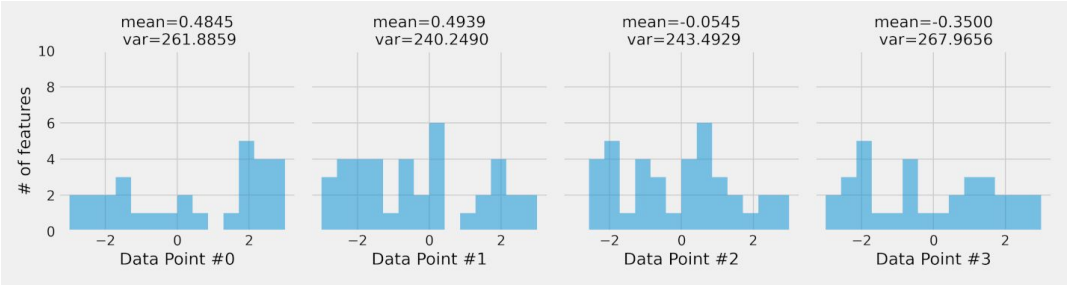


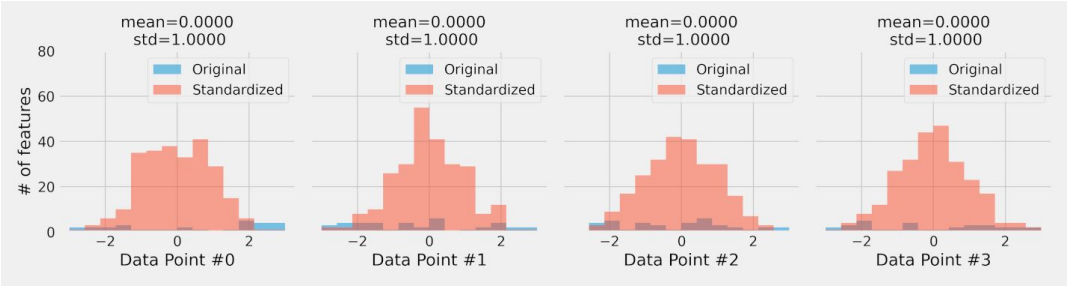
Image source: Voigt Godoy, D. dvgodoy / CC BY / https://github.com/dvgodoy/dl-visuals/

Effect of LayerNorm

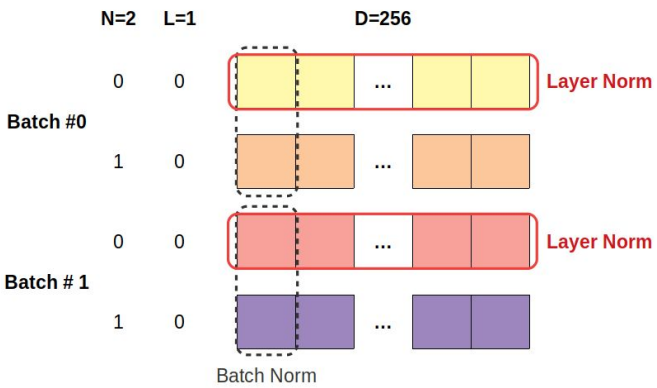
Before LayerNorm



After LayerNorm



BatchNorm vs LayerNorm



LayerNorm: Layer Normalization by Lei Ba, J. et al. (2016)

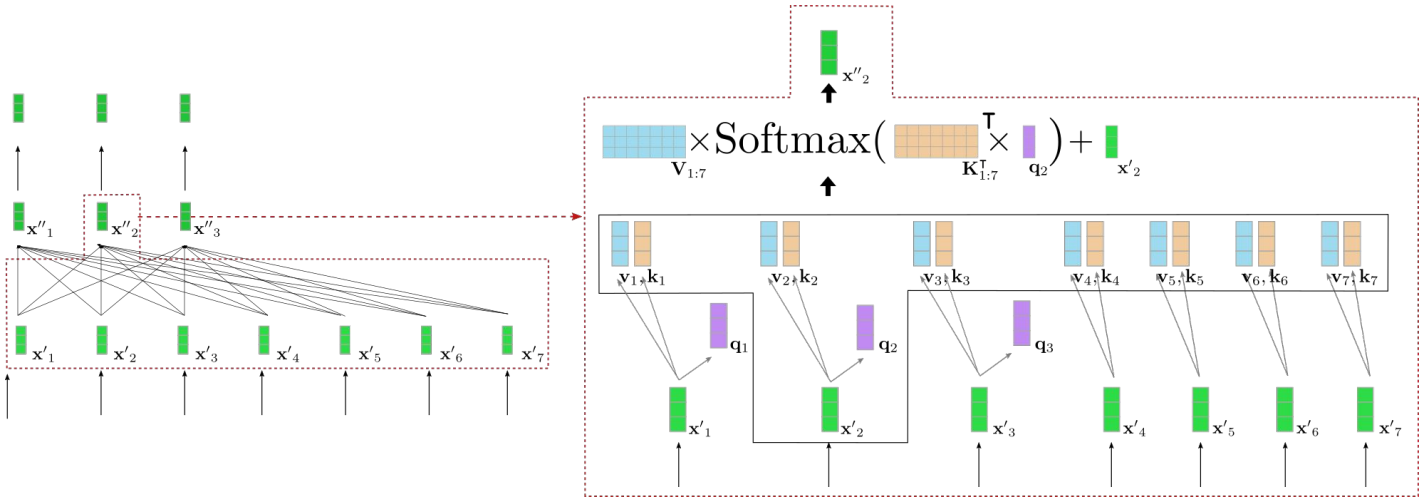
Image source: Voigt Godoy, D. dvgodoy / CC BY / https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

2.7. Positional Embeddings

Wait, where does this comes from?

Attention reminder: note that there is **no ordering** in the fusion operation!

How can we focus on left vs right, close vs distant elements?



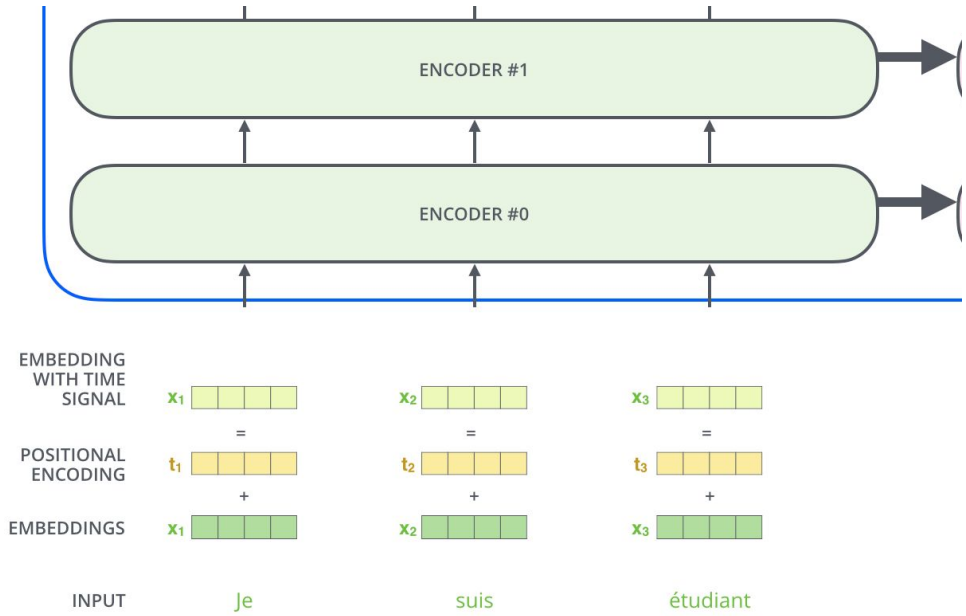
The solution: add Positional Embeddings

Because we have embedded our tokens into a **high-dimensional space**, we have **room for more information**!

We will **add** (literally) **extra information about the position** of the token in the sequence.

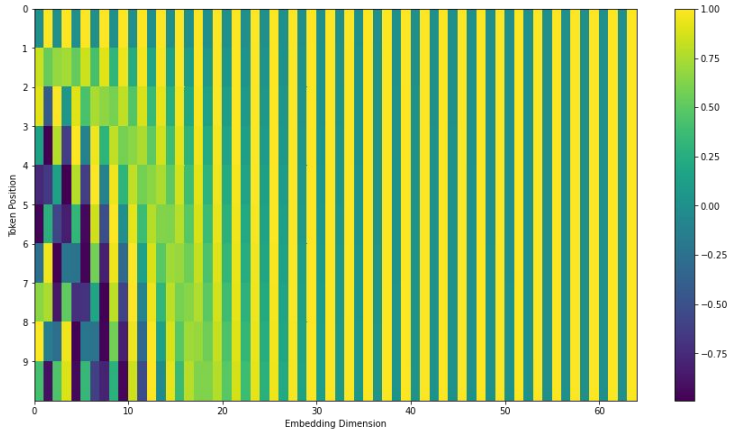
We must encode this information in a useful way so the network can easily assess the distance and direction between two tokens.

The original, complicated way: sin & cos signals



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

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



http://jalammar.github.io/illustrated-transformer/

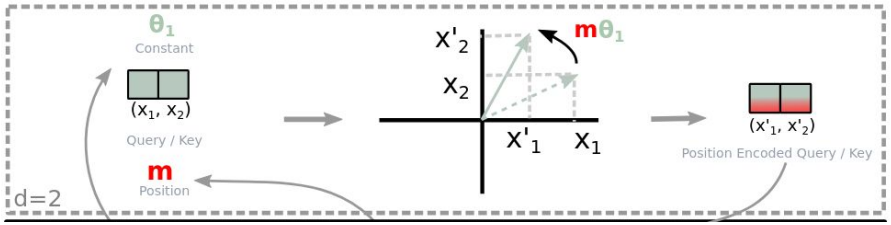
Variants

Learned: token_id → FFN → embedding

No positional embeddings: position-invariance!

The modern way: RoPE

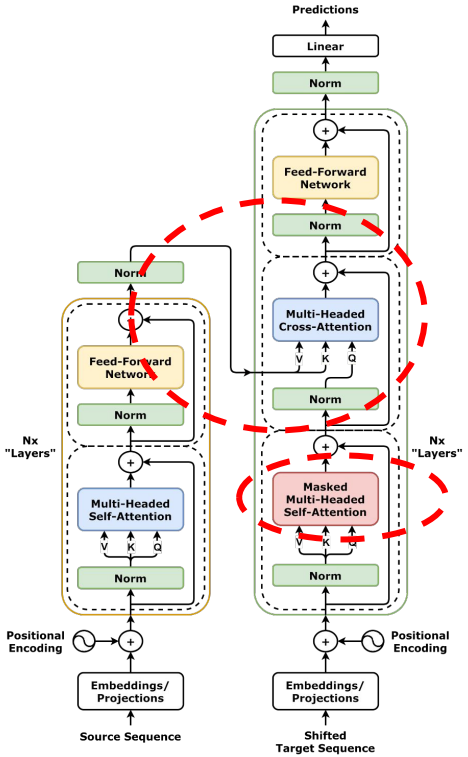
Su, Jianlin; Lu, Yu; Pan, Shengfeng; Murtadha, Ahmed; Wen, Bo; Liu, Yunfeng (2021-04-01). "RoFormer: Enhanced Transformer with Rotary Position Embedding". arXiv:2104.09864

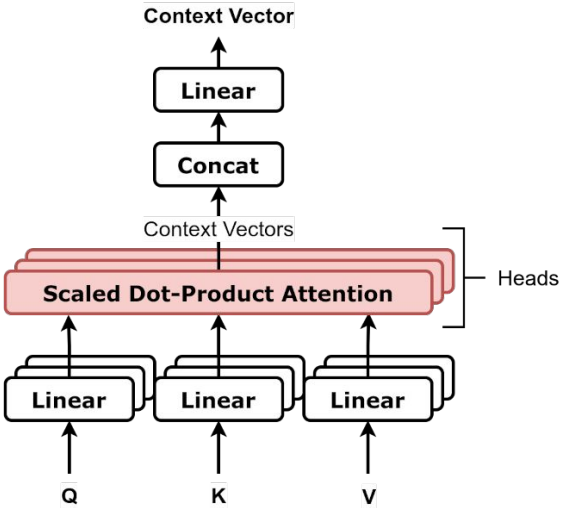


2.8. Full picture

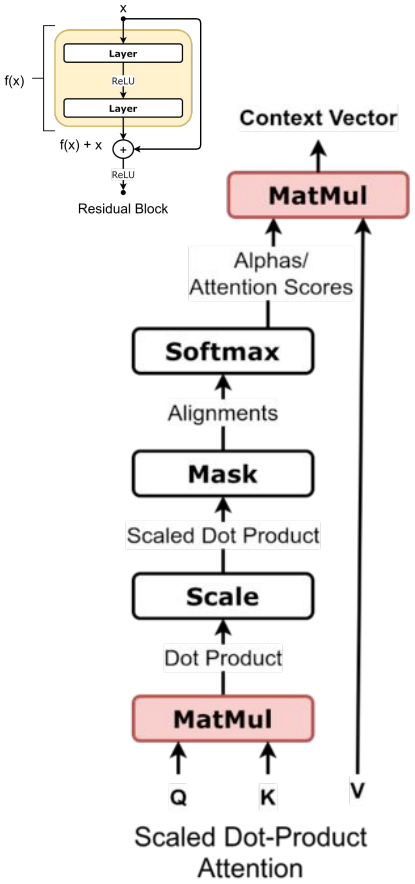
The full picture…

$$FFN(x) = \sigma(X\mathbf{W}_1+\mathbf{B}_1)\mathbf{W}_2+\mathbf{B}_2$$





Multiheaded attention



Scaled Dot-Product Attention

Image source: Voigt Godoy, D. dvgodoy / CC BY / https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

2.9. Masked attention in decoder

The decoder can only look at the past

Because it generates the output in a auto-regressive manner,
the decoder can only look at previous elements, contrary to the encoder.

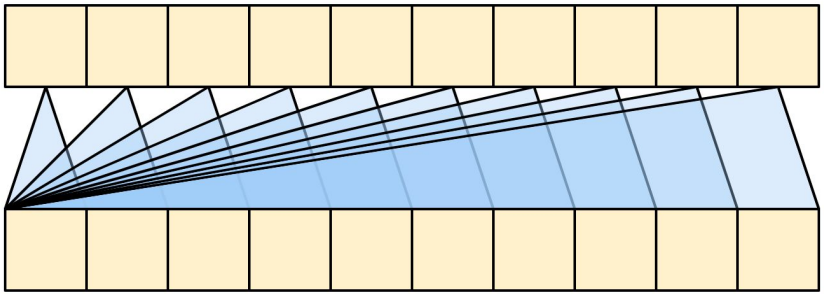
This is implemented by adding a **mask** to the attention matrix which zeroes attention to futur tokens.

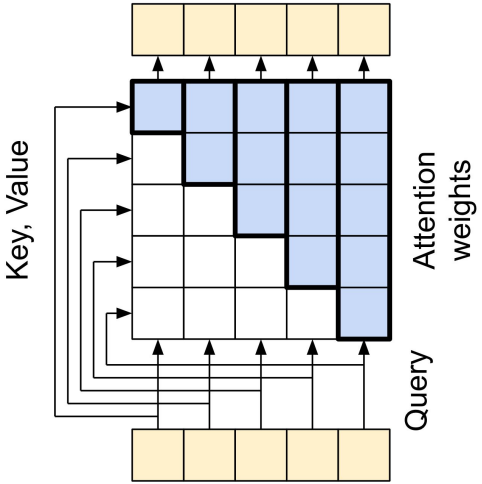
This implementation has two benefits:

- It make the implementation similar between causal and non causal self-attention
- It enables efficient parallel training of the decoder using “teacher forcing”*

**more about this later*

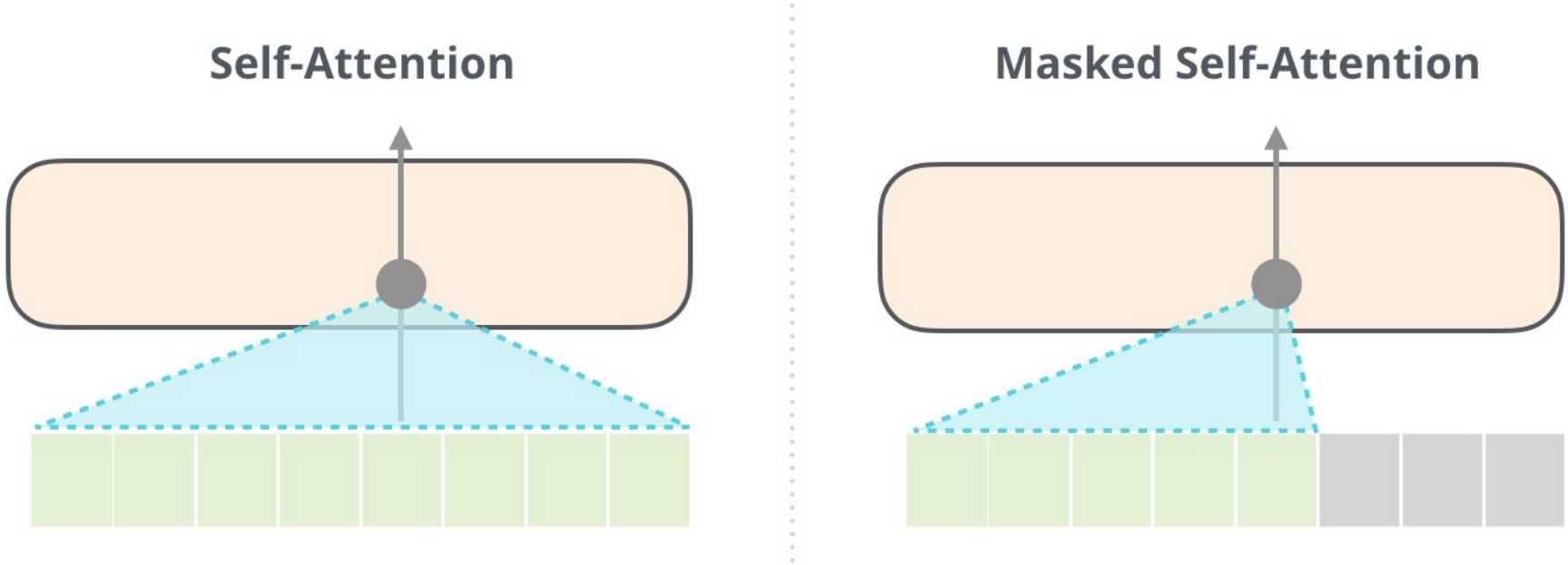
It is only takes a triangular matrix





https://www.tensorflow.org/text/tutorials/transformer

Self-attention vs masked self-attention for a token



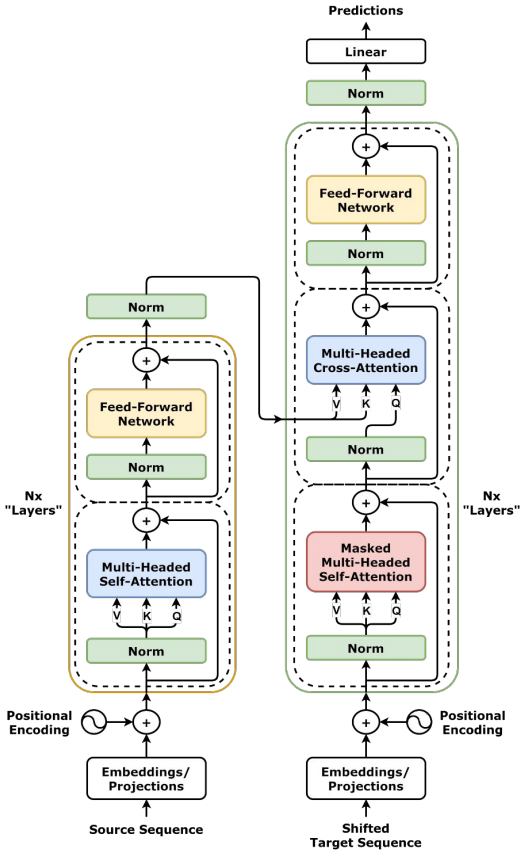
https://jalammar.github.io/illustrated-gpt2/

2.10. Cross attention in decoder

Cross attention is self-attention, but to encoder outputs

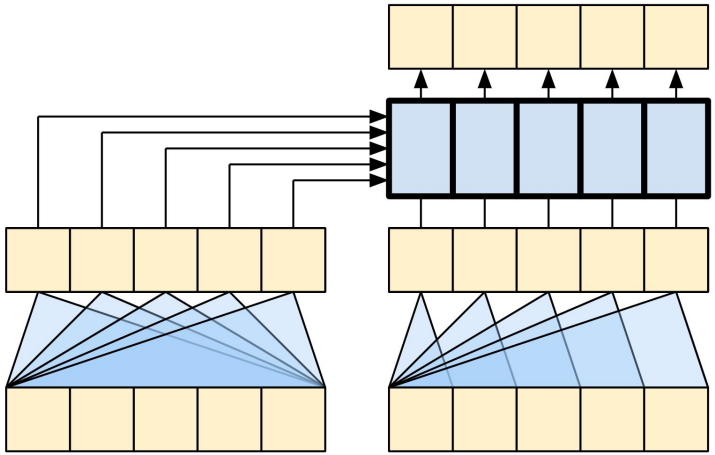
So at **each** decoder layer, the decoder:

1. Looks at previous tokens it predicted
   → **Masked** Multi-Head **Self**-Attention
2. Looks at all the contextual representations of the input produced by the encoder
   → Multi-Head **Cross**-Attention
3. Transforms information using a Feed-Forward Net.

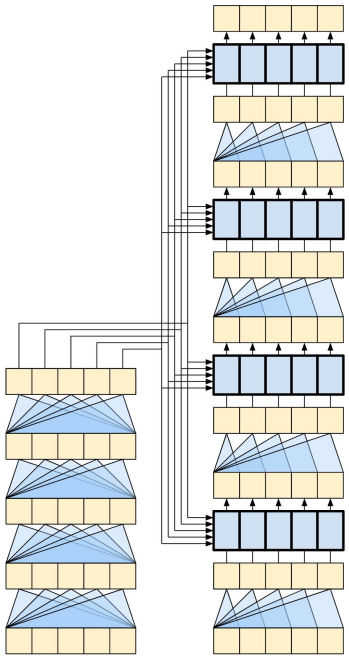


Cross-attention vs (masked) self-attention

A 1-layer transformer



A 4-layer transformer



https://www.tensorflow.org/text/tutorials/transformer

2.11. Architecture summary

So, in the end you get:

- Encoder and decoder
- Cross- and self-attention
- Multiple heads
- Positional embeddings

Embeddings and positional encodings are computed only at the beginning of the process.

Softmax is used to select the id of the token to predict at each stage.

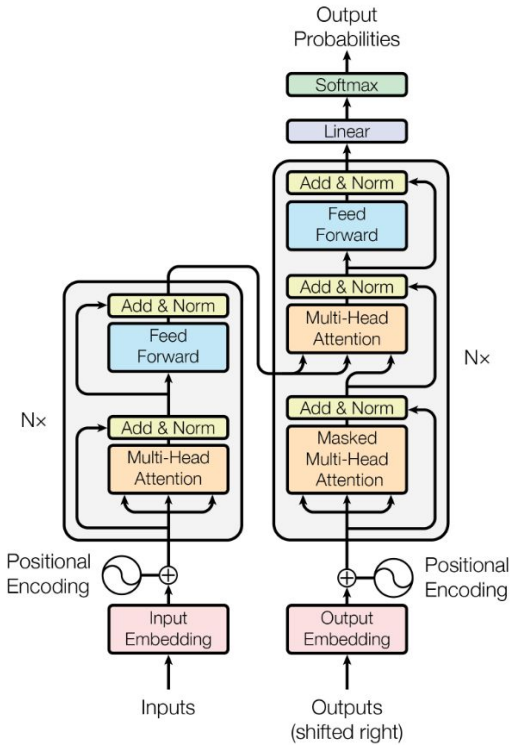
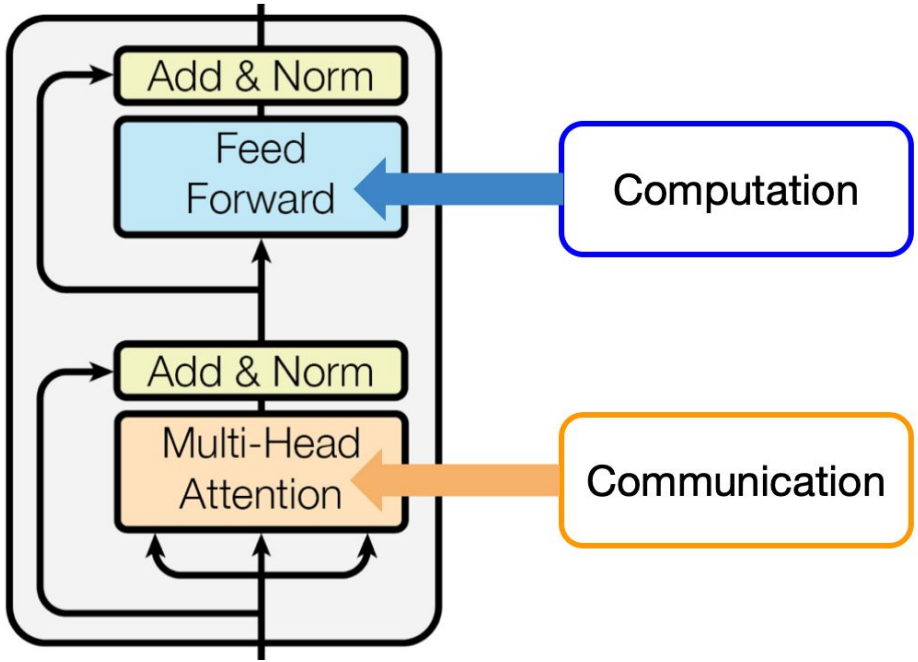


Figure 1: The Transformer - model architecture.

Image source: A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 6000–6010.

One last thing…



From: https://thegradient.pub/mamba-explained/