Transformers - Introduction

SICSS Paris '25

Julien Boelaert

CERAPS, Université de Lille

24/06/2025

Outline

Transformers: one architecture to rule them all

1. Motivation
   - Beyond static embeddings, beyond RNNs
   - Power laws
2. Transformers building blocks
   - Embeddings, Attention, and MLPs
   - Positional embeddings
   - Tokenizers

1/ Transformers: motivation

Transformers are a neural network architecture that have been state of the art in all NLP tasks for the last 5+ years.

- Original article (Google): Vaswani *et al*, 2017, "Attention is all you need", *Proceedings of the 31st Conference on Neural Information Processing Systems*
- Originally an encoder-decoder network (*eg* for machine translation), today usually either encoder-only (BERT) or decoder-only (GPT)
- Also speech-to-text (Whisper), in diffusion models for SOTA image generation (Dall-E, Midjourney, Stable diffusion, ...)
- NLP benchmarks: GLUE and superGLUE `https://super.gluebenchmark.com/leaderboard`

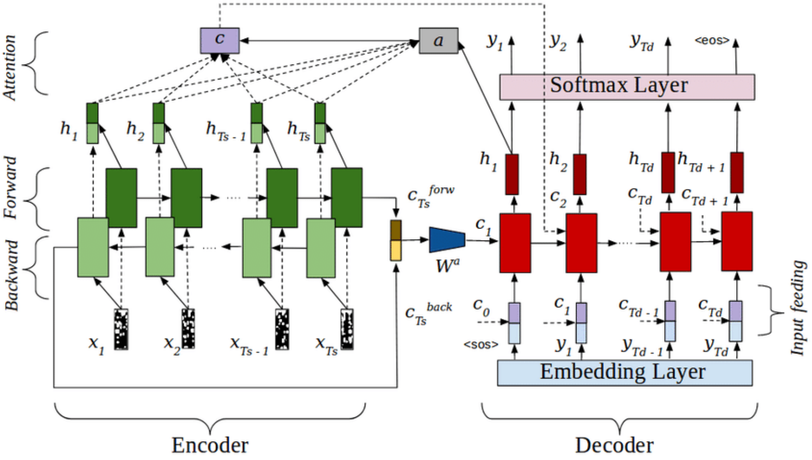
1/ Transformers: motivation

Static word embeddings will only get you so far:

- No context-awareness
  ("Michael" $+$ "Jackson" $\stackrel{?}{=}$ "Michael Jackson")
- Recurrent nets can mitigate this (treat words one sequentially)
  But: tricky and expensive to train
- ELMo (2018): contextual embeddings (character-level bi-LSTM), subsequently fine-tuned on downstream tasks

1/ Transformers: motivation

Previous SOTA: encoder-decoder **RNN with attention**
$\rightarrow$ For each output token, learn to "attend" to relevant input tokens



1/ Transformers: motivation

Transformers: how, why?

- Key concept: **self-attention** (*eg* attention without recurrence) $\rightarrow$ treat text as a block, not word by word
- Very parallelizable $\rightarrow$ very **scalable** (billions of parameters, trillions of tokens)
- **New paradigm** in machine learning research:
  - transformers replaced nearly all other neural models
  - scaling laws $\rightarrow$ bigger models, more data
  - pre-training a good model costs billions

1/ Transformers: motivation

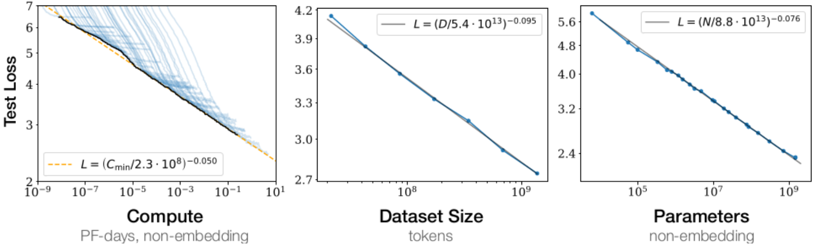
Scaling laws: predictable LLM quality improvement based on compute time, model size and amount of data

- Traditional NN wisdom: "bigger is better"

1/ Transformers: motivation

Scaling laws: predictable LLM quality improvement based on compute time, model size and amount of data

- Traditional NN wisdom: "bigger is better"
- Kaplan et al 2020 (OpenAI):
  - GPT quality improvements are predictible power laws
  - Optimal allocation: bigger models rather than bigger data
  - Justification for massive investments in LLMs
  - Validated by GPT-3 (Brown et al, 2020)



Kaplan et al (2020), "Scaling Laws for Neural Language Models", `https://arxiv.org/abs/2001.08361`

1/ Transformers: motivation

Scaling laws:

- Traditional NN wisdom: "bigger is better"
- Kaplan et al 2020 (OpenAI): "use bigger models"
- Hoffmann et al 2022 (DeepMind): "also use more data"
  - "Current LLMs are significantly under-trained"
  - Optimal allocation: increase data as much as parameters
  - Proof: *Chinchilla* (70B) outperforms *Gopher* (280B), with same computational budget but 4x more training data

1/ Transformers: motivation

Scaling laws: and now, the wall?

What to do when all the text web is already in the models?

1/ Transformers: motivation

Scaling laws: and now, the wall?

What to do when all the text web is already in the models?

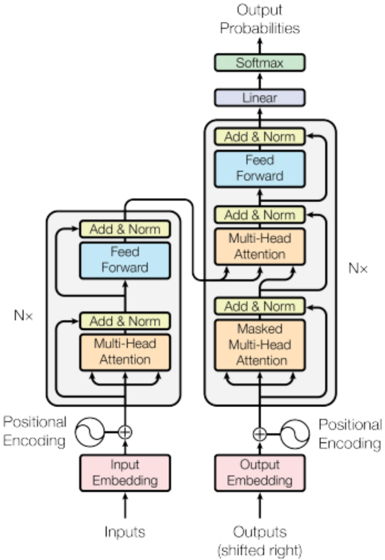
- New data sources: synthetic, images, videos, videogames...
- Scaling laws for AI assistants (beyond foundational LLMs)
  - more instruction fine-tuning
  - test-time computation ("thinking models")
  - active research field

2/ Transformers: building blocks

2/ Transformers: building blocks

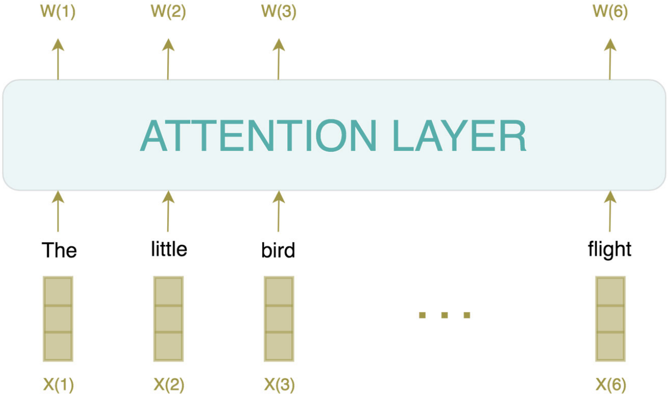
Transformer building blocks:
(GPT-3 % of parameters)

- Token embeddings ($< 1\%$)
  $+$ positional encoding
- Self-attention block (33%)
- MLP (66%)
- Layer normalization ($< 1\%$)
- Skip-layer connections



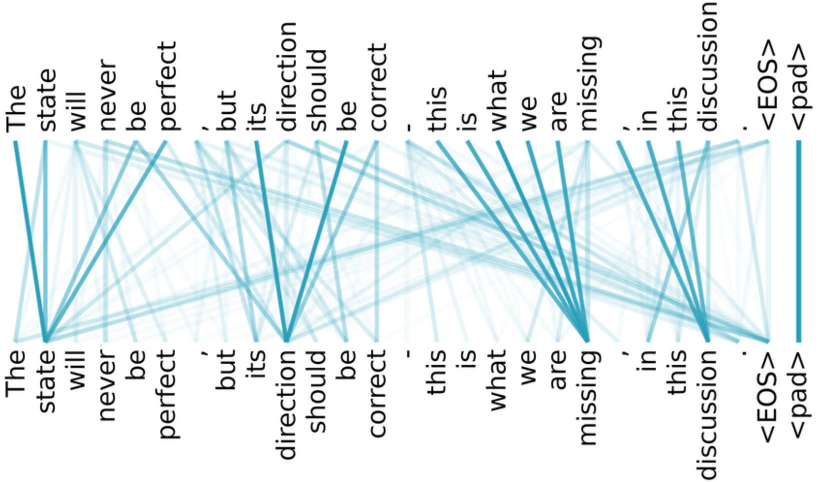
2/ Transformers: building blocks

Each layer transforms a (fixed-size) sequence of embeddings to a (same-size) sequence of embeddings.



2/ Transformers: building blocks

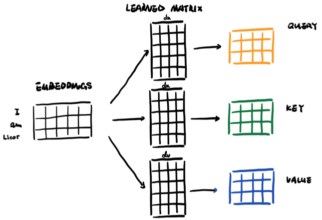
At each layer, each token embedding receives additional information from its context



2/ Transformers: building blocks

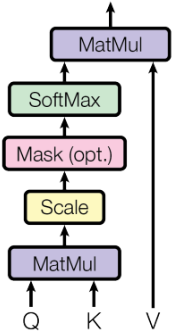
Key innovation: **self-attention** block

- Each token has an embedding (learned only in first layer)
- Multiplied by M_q, M_k, M_v (learned for each layer)
- Result:
  - Queries (what to look for)
  - Keys (context information)
  - Values (passed to next layer)



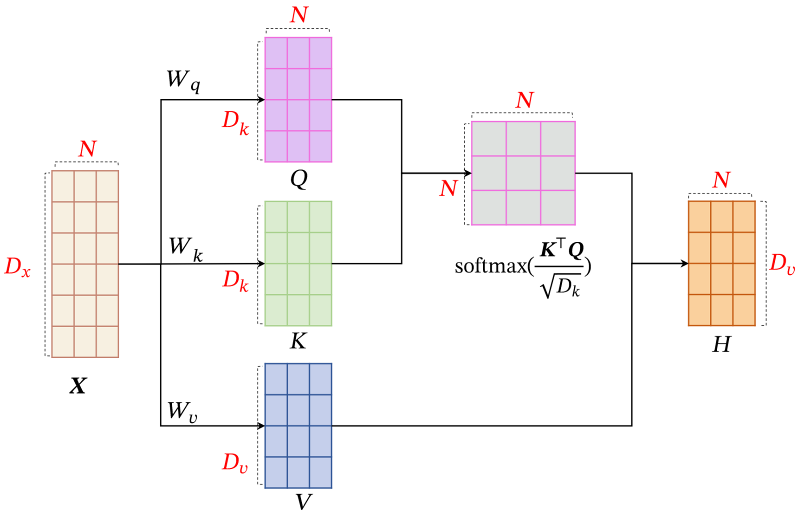
Self-attention, formally:

- $Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$
- Similarity between Q and K: dot-product
- Multi-head: many parallel attention blocks, each specializes in some semantic aspect
- (Relatively) cheap way of computing features to pass to the MLP



2/ Transformers: building blocks

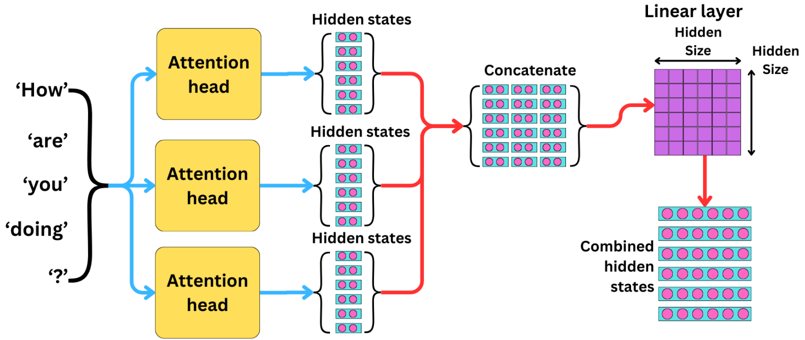
Self-attention: the QKV mechanism



2/ Transformers: building blocks

Multi-head attention: several parallel self-attention blocks

→ each parallel "head" learns to focus on a different aspect
eg for nouns: "gender-ness" (male, female, neuter, neither) or "cardinality" (singular vs plural)



2/ Transformers: building blocks

Self-attention: prosaically

- Each token is directly connected to all other tokens
- QKV learns, for each token, which tokens to **attend** to
- Each output token is a weighted sum of all the Values
- Efficient way to combine the information of all tokens

NB: Memory requirement $\sim$ (sequence length)^2 $\rightarrow$ **bottleneck**

2/ Transformers: building blocks

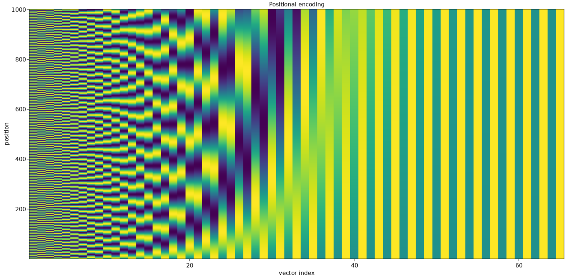
Self-attention treats sentence as a block, but has no positional information: order of tokens is not important.

2/ Transformers: building blocks

Self-attention treats sentence as a block, but has no positional information: order of tokens is not important.

Solution: add **positional encoding** to input embeddings: position signature. Usually deterministic:

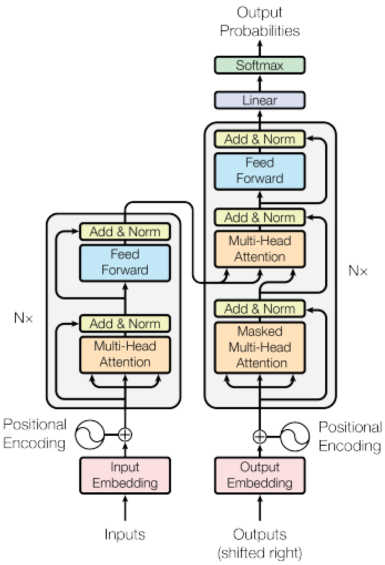
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



2/ Transformers: building blocks

Building blocks of the transformer:

- Token embeddings + positional encoding
- Repeat:
  - Self-attention block
  - **MLP**
  - Layer normalization, skip-layer connections
- **Masked attention**: each token can only attend to previous tokens (decoder only)



2/ Transformers: building blocks

But what are **tokens**? parts of words

- Each transformer model comes with its specific **tokenizer** (typically $\sim$ 40k vocabulary)
- Tokens strike a balance between two extremes:
  - characters (min vocabulary size, $\sim$ 100)
  - whole words (max vocab size, could be millions)
- Ex bert-base-cased: "This is bedazzling!" -> ('[CLS]', 'This', 'is', 'bed', '##az', '##z', '##ling', '!', '[SEP]')

2/ Transformers: building blocks

When you download a pre-trained Transformer model, you also download its **pre-trained tokenizer**:

- The pre-trained tokenizer applies the same pre-treatment of raw text as was used during training
  - cased / uncased
  - adds special tokens: *eg* "[CLS]" at start, "[SEP]" at end, "[UNK]" when unknown, padding
  - turns tokens into indices of model's vocabulary

- No more elaborate pre-processing (no lemmatization, phrase consolidation, filtering on POS or frequency...): just feed tokenizer with natural text in the same form as used for training (usually raw, but GPT-1 used spacy pre-tokenization into words).

2/ Transformers: building blocks

Tokenizers are pre-trained on the training corpus:

- byte-pair encoding (BPE, originally used for compression):
  - start: vocabulary = all characters
  - iteratively add to vocabulary the most common pair of vocabulary items
  - repeat until pre-defined vocabulary size is attained

2/ Transformers: building blocks

Tokenizers are pre-trained on the training corpus:

- byte-pair encoding (BPE, originally used for compression):
  - start: vocabulary = all characters
  - iteratively add to vocabulary the most common pair of vocabulary items
  - repeat until pre-defined vocabulary size is attained
- modern variants, sentencepiece or wordpiece (with / without whitespace):
  - score for merging = $freq\_pair/(freq\_1st\_elt \times freq\_2d\_elt)$
  - tokenization: find longest subword in vocabulary
- Common words will be a single token
  Most OOV words will be several tokens:
  "hypatia" $\rightarrow$ "hyp", "##at", "##i", "##a"
  Extremely OOV words will be [UNK] (emojis, alphabets...)

2/ Transformers: building blocks

Transformers in practice:

- Central hub for open source models:
  `https://huggingface.co`
- Python library: transformers
  Useful bindings, pipelines (*eg* sequence classification, token classification, text generation, translation, ...)
- Need GPU to run/train in reasonable time
- Most models come in different sizes: small, base, large... The bigger, the better (and slower).

Further reading

- Dan Jurafsky and James H. Martin, 2023, *Speech and Language Processing* (3rd ed. draft), chapters I.7, I.9-12, `https://web.stanford.edu/~jurafsky/slp3`

Tutorials:

- Illustrated transformer: `https://jalammar.github.io/illustrated-transformer/`
- Learn prompting: `https://learnprompting.org/`

YouTube:

- 3Blue1Brown Deep Learning series "Transformers explained visually" `https://www.youtube.com/watch?v=wjZofJX0v4M` (and following videos)
- Welch Labs "How DeepSeek rewrote the transformer" `https://www.youtube.com/watch?v=0VLAoVGf_74`

Questions and comments: julien.boelaert@univ-lille.fr