RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

ENS DE LYON

Large Language Models and Transformers

Mathurin Massias

https://mathurinm.github.io

Inria Lyon, OCKHAM team

ICJ, 09/11/2026

Outline and disclaimers

- I am not an expert in LLMs
- audience: mathematicians (and computer scientists), a priori not doing ML
- goal: get a broad understanding of how a LLM works
- ⚠ some content is blatantly wrong but simplified to grasp the big picture

Outline and disclaimers

- I am not an expert in LLMs
- audience: mathematicians (and computer scientists), a priori not doing ML
- goal: get a broad understanding of how a LLM works
- ⚠ some content is blatantly wrong but simplified to grasp the big picture
- I am not an expert in LLMs

Outline

Machine and Deep learning in a nutshell

Large language models

Architecture: Transformers and Attention

Machine learning: learn to map inputs to outputs

In Machine learning we have:

- *inputs* $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ e.g. flattened images of dogs and cats, $d = \#$pixels
- *labels* $y_1, \ldots, y_n \in \{0, 1\}$ e.g. $y =$ 0 if image of cat, 1 if image of dog

We want:

- to learn to predict if $y = 0$ or 1 for a new image $\mathbf{x}$ (= if it's a cat or a dog)

learning = finding a good function $f : \mathbb{R}^d \to \{0, 1\}$

this is only "binary classification", there is much more to ML

What should the model f output?

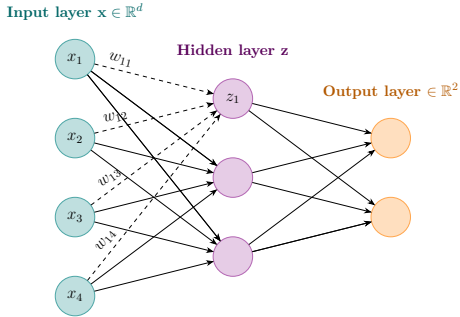
- predicting discrete values $f(\mathbf{x}) \in \{0, 1\}$ is a pain
- remedy: predict score vector $\in \mathbb{R}^2$:

$$f(\mathbf{x}) \approx \begin{pmatrix} \mathbb{P}(y = \text{cat}|\mathbf{x}) \\ \mathbb{P}(y = \text{dog}|\mathbf{x}) \end{pmatrix}$$

- this idea adapts to more labels e.g. cat/dog/horse/cow, $f(\mathbf{x}) \in \mathbb{R}^4$

Let's do this with Deep learning

- deep neural networks are parametric functions (here $f : \mathbb{R}^4 \to \mathbb{R}^2$)

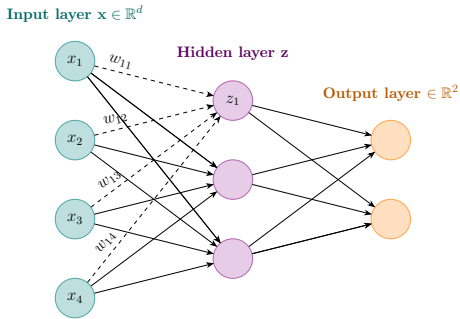


- connections between neurons have **weights** $w_{ij} \in \mathbb{R}$ (parameters: can be tuned)
- the value of a neuron is determined by the value of its antecedents:

$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 \in \mathbb{R}$$

Let's do this with Deep learning

- deep neural networks are parametric functions (here $f : \mathbb{R}^4 \to \mathbb{R}^2$)

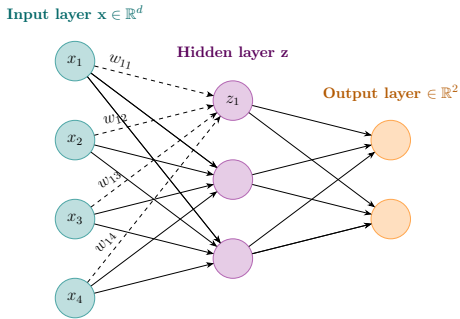


- connections between neurons have **weights** $w_{ij} \in \mathbb{R}$ (parameters: can be tuned)
- the value of a neuron is determined by the value of its antecedents:

$$z_1 = \phi(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4) \in \mathbb{R} \qquad \phi : \mathbb{R} \to \mathbb{R} : \textit{ non linearity}$$

Let's do this with Deep learning

- deep neural networks are parametric functions (here $f : \mathbb{R}^4 \to \mathbb{R}^2$)

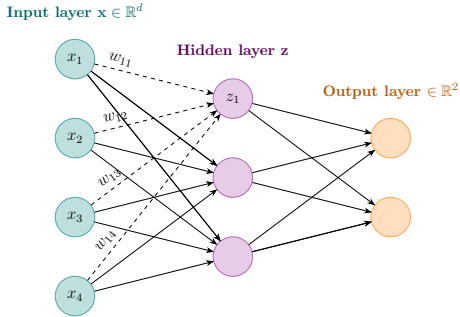


- connections between neurons have **weights** $w_{ij} \in \mathbb{R}$ (parameters: can be tuned)
- the value of a neuron is determined by the value of its antecedents:

$$z_1 = \phi(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4) \in \mathbb{R} \qquad \phi : \mathbb{R} \to \mathbb{R} : \textit{ non linearity}$$

Let's do this with Deep learning

- deep neural networks are parametric functions (here $f : \mathbb{R}^4 \to \mathbb{R}^2$)

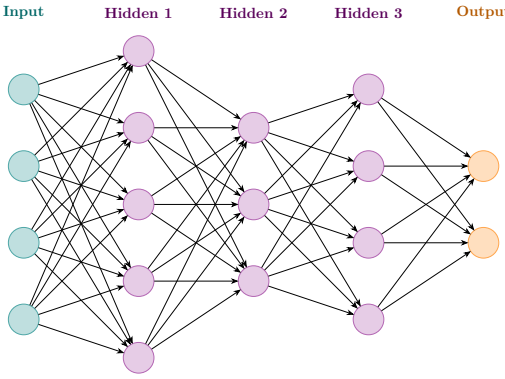


- connections between neurons have **weights** $w_{ij} \in \mathbb{R}$ (parameters: can be tuned)
- the value of a neuron is determined by the value of its antecedents:

$$z_1 = \phi(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4) \in \mathbb{R} \qquad \phi : \mathbb{R} \to \mathbb{R} : \textit{ non linearity}$$

- compact notation: $\mathbf{z} = \phi(\mathbf{W}\mathbf{x}) \in \mathbb{R}^3, \quad \mathbf{W} \in \mathbb{R}^{3\times 4}$ ϕ applied pointwise

Moving on to the next layer



each layer ℓ has its weights $\mathbf{W}^\ell$

- $\mathbf{x}$: input
- $\mathbf{z}^1 = \phi(\mathbf{W}^1\mathbf{x})$
- $\mathbf{z}^2 = \phi(\mathbf{W}^2\mathbf{z}^1)$
- $\mathbf{z}^3 = \phi(\mathbf{W}^3\mathbf{z}^2)$
- $f(\mathbf{x}) = \phi(\mathbf{W}^4\mathbf{z}^3)$

$$f(\mathbf{x}) = \phi(\mathbf{W}^4\phi(\mathbf{W}^3\phi(\mathbf{W}^2\phi(\mathbf{W}^1\mathbf{x})))) \in \mathbb{R}^2$$

How to get probability vectors as output?

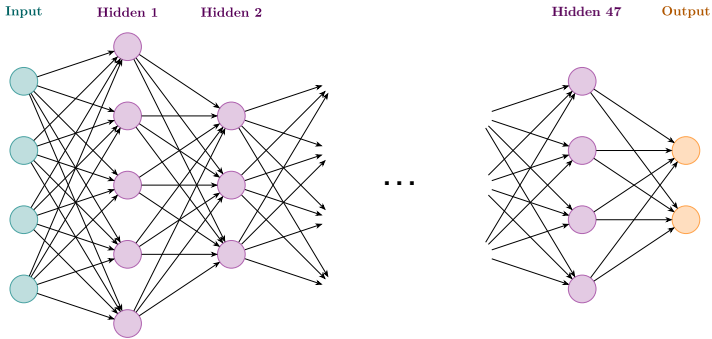
- remember we want $f(\mathbf{x}) \approx \begin{pmatrix} \mathbb{P}(y = \text{cat}|\mathbf{x}) \\ \mathbb{P}(y = \text{dog}|\mathbf{x}) \end{pmatrix}$: must be ≥ 0 and sum to 1
- the last layer nonlinearity is different: we use this nonlinearity is not pointwise

$$\text{softmax}(\mathbf{v}) : \mathbb{R}^p \to \Delta_p$$
$$\mathbf{v} \mapsto \left(\frac{\exp(v_i)}{\sum_{j=1}^{p} \exp(v_j)} \right)_i$$

- gives a positive vector & entries sum to 1 by design: yay
- ex: $\text{softmax}(\begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}) \approx \begin{pmatrix} 0.67 \\ 0.09 \\ 0.24 \end{pmatrix}$

DEEP neural networks

- simply means many hidden layers:



- large models: easily millions of parameters (quadratic in number of neurons in layer)
- GPT3: $\approx$ 175 billion parameters, current models: around 1 trillion (rumored?)

Finding good weights (= optimization)

- fix *architecture* (number of layers, size of each layer)

Finding good weights (= optimization)

- fix *architecture* (number of layers, size of each layer)
- pick random initial values for weights $\mathbf{W}^{(\ell)}$ in each layer ℓ

Finding good weights (= optimization)

- fix *architecture* (number of layers, size of each layer)
- pick random initial values for weights $\mathbf{W}^{(\ell)}$ in each layer ℓ
- for each image $\mathbf{x}_i$ in training set, compare true label y_i with network prediction $f(\mathbf{x}_i)$

Finding good weights (= optimization)

- fix *architecture* (number of layers, size of each layer)
- pick random initial values for weights $\mathbf{W}^{(\ell)}$ in each layer ℓ
- for each image $\mathbf{x}_i$ in training set, compare true label y_i with network prediction $f(\mathbf{x}_i)$
- modify each weight a little bit to make prediction closer to true label y_i, and iterate over samples

  keywords: stochastic gradient descent, ADAM, Muon, cross-entropy loss

Finding good weights (= optimization)

- fix *architecture* (number of layers, size of each layer)
- pick random initial values for weights $\mathbf{W}^{(\ell)}$ in each layer ℓ
- for each image $\mathbf{x}_i$ in training set, compare true label y_i with network prediction $f(\mathbf{x}_i)$
- modify each weight a little bit to make prediction closer to true label y_i, and iterate over samples

  keywords: stochastic gradient descent, ADAM, Muon, cross-entropy loss
- when reaching the end of the dataset, start a new loop over samples

Outline

Machine and Deep learning in a nutshell

Large language models

Architecture: Transformers and Attention

LLM : the output

Large Language Models simply do **next word prediction**:

- let's say we work in a language in which there are $n_{\text{words}} = 100000$ words (vocabulary)
- given a prompt $\mathbf{x}$, the LLM (the model f) predicts the probability distribution of the next word.
  It simply outputs a probability vector over each possible word in the vocabulary:

$$f(\mathbf{x}) = \begin{pmatrix} \mathbb{P}(\text{"a"}|\mathbf{x}) \\ \mathbb{P}(\text{"the"}|\mathbf{x}) \\ \mathbb{P}(\text{"of"}|\mathbf{x}) \\ \mathbb{P}(\text{"mathurin"}|\mathbf{x}) \\ \vdots \end{pmatrix} \quad \updownarrow \; n_{\text{words}}$$

- this is exactly the same setting as with cats and dogs! but much bigger output space

LLM : the output

Large Language Models simply do **next word prediction**:

- let's say we work in a language in which there are $n_{\text{words}} = 100000$ words (vocabulary)
- given a prompt $\mathbf{x}$, the LLM (the model f) predicts the probability distribution of the next word.
  It simply outputs a probability vector over each possible word in the vocabulary:

$$f(\mathbf{x}) = \begin{pmatrix} \mathbb{P}(\text{"a"}|\mathbf{x}) \\ \mathbb{P}(\text{"the"}|\mathbf{x}) \\ \mathbb{P}(\text{"of"}|\mathbf{x}) \\ \mathbb{P}(\text{"mathurin"}|\mathbf{x}) \\ \vdots \end{pmatrix} \quad \updownarrow \; n_{\text{words}}$$

- this is exactly the same setting as with cats and dogs! but much bigger output space

but what is the input $\mathbf{x}$?

LLM: the input

- input $\mathbf{x}$ = last $T \in \mathbb{N}$ words of discussion (**context window**)

LLM: the input

- input $\mathbf{x}$ = last $T \in \mathbb{N}$ words of discussion (**context window**)
- $T = 5$: IRL $T \approx 100k$ (?)

$$f\Big(\text{"can", "you", "help", "me", "?"}\Big) = \begin{pmatrix} \mathbb{P}(\text{"a"}|\mathbf{x}) = 0.001 \\ \mathbb{P}(\text{"the"}|\mathbf{x}) = 0.002 \\ \mathbb{P}(\text{"of"}|\mathbf{x}) = 0.4 \\ \mathbb{P}(\text{"mathurin"}|\mathbf{x}) = 0 \\ \vdots \end{pmatrix}$$

LLM: the input

- input $\mathbf{x}$ = last $T \in \mathbb{N}$ words of discussion (**context window**)
- $T = 5$: IRL $T \approx 100k$ (?)

$$f\Big(\text{"can", "you", "help", "me", "?"}\Big) = \begin{pmatrix} \mathbb{P}(\text{"a"}|\mathbf{x}) = 0.001 \\ \mathbb{P}(\text{"the"}|\mathbf{x}) = 0.002 \\ \mathbb{P}(\text{"of"}|\mathbf{x}) = 0.4 \\ \mathbb{P}(\text{"mathurin"}|\mathbf{x}) = 0 \\ \vdots \end{pmatrix}$$

- super easy to find training data with true answers (our $\mathbf{x_i}/y_i$'s)

Autoregressive prediction

To get the full answer of the model:

- given the prompt, pick the most likely word according to model as prediction

  you can also sample from the proba vector, to get non deterministic prediction

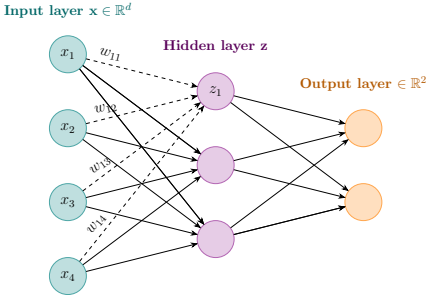
- shift all words in the context by one to the left, add predicted word to the right

$$f\Big(\text{"can", "you", "help", "me", "?"}\Big) = \begin{pmatrix} \mathbb{P}(\text{"a"}|\mathbf{x}) = 0.001 \\ \mathbb{P}(\text{"the"}|\mathbf{x}) = 0.002 \\ \mathbb{P}(\text{"of"}|\mathbf{x}) = 0.4 \\ \mathbb{P}(\text{"mathurin"}|\mathbf{x}) = 0 \\ \vdots \end{pmatrix}$$

$$f\Big(\text{"you", "help", "me", "?", "Of"}\Big) = \cdots$$

- that's why models answer like they're typing progressively

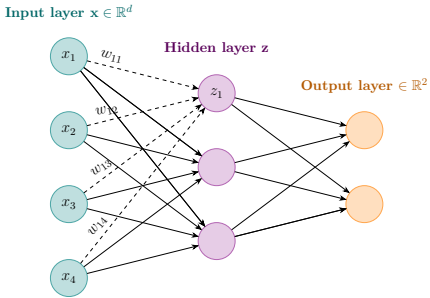
About the input: embedding



$$\mathbf{x} = (\text{"can"}, \text{"you"}, \text{"help"}, \text{"me"})$$

$$z_1 = \phi(w_{11} \times \text{"can"} + w_{12} \times \text{"you"} + ...) \text{ ???????}$$

About the input: embedding



$$\mathbf{x} = (\text{"can"}, \text{"you"}, \text{"help"}, \text{"me"})$$

$$z_1 = \phi(w_{11} \times \text{"can"} + w_{12} \times \text{"you"} + ...) \text{ ???????}$$

- first, models don't work with words, but with **tokens**
- second, they work with vector representations of tokens, called **embeddings**

About the input: tokenization

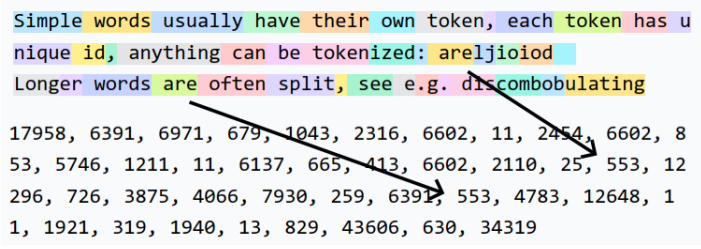
- working with words is inefficient: instead, use *tokens* (≈ pieces of words)
- set of tokens is learned independently, a priori, with the Byte Pair Encoding (BPE) algorithm 5 min explanation: `https://www.youtube.com/watch?v=HEikzVL-lZU`
- a tokenizer takes any text and returns the list of indices of tokens that compose it

```
Simple words usually have their own token, each token has u
nique id, anything can be tokenized: areljioiod
Longer words are often split, see e.g. discombobulating

17958, 6391, 6971, 679, 1043, 2316, 6602, 11, 2454, 6602, 8
53, 5746, 1211, 11, 6137, 665, 413, 6602, 2110, 25, 553, 12
296, 726, 3875, 4066, 7930, 259, 6391, 553, 4783, 12648, 1
1, 1921, 319, 1940, 13, 829, 43606, 630, 34319
```

About the input: tokenization

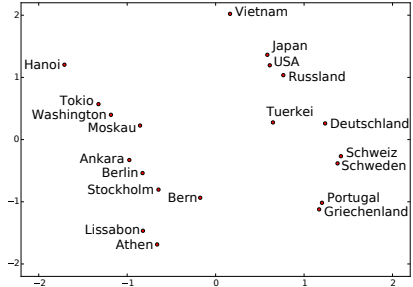
- working with words is inefficient: instead, use *tokens* (≈ pieces of words)
- set of tokens is learned independently, a priori, with the Byte Pair Encoding (BPE) algorithm 5 min explanation: `https://www.youtube.com/watch?v=HEikzVL-lZU`
- a tokenizer takes any text and returns the list of indices of tokens that compose it



- play with various tokenizers: `https://tiktokenizer.vercel.app/`
- typically $n_{\text{tokens}} \approx$ 150 000 distinct tokens

About the input: embedding the tokens

- each of the 150 000 tokens in the vocabulary is assigned a location in $\mathbb{R}^e$: its **embedding**. Example in dim 2 ($e = 2$) :

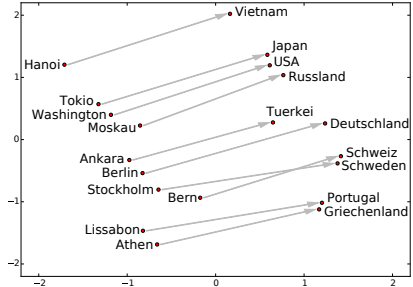


here I use words instead of tokens for simplicity (and they are german words...)

- the position of each embedding is a parameter of the model, and is learned during training, like the network weights!
- the model learns some kind of useful geometric representation of each token/word

About the input: embedding the tokens

- each of the 150 000 tokens in the vocabulary is assigned a location in $\mathbb{R}^e$: its **embedding**. Example in dim 2 ($e = 2$) :



here I use words instead of tokens for simplicity (and they are german words...)

- the position of each embedding is a parameter of the model, and is learned during training, like the network weights!
- the model learns some kind of useful geometric representation of each token/word

Tokenization and embedding: recap

- the input prompt is tokenized, goes into context window of T tokens (T fixed, pad/truncate if necessary): we have a list of T token indices
  typically context length $T \approx 100\,000$
- each token is mapped to its *embedding* $\in \mathbb{R}^e$: we have a list of T vectors in $\mathbb{R}^e$
  typically embedding size $e \approx 10\,000$
- the input of model f consists of T embeddings, and it outputs a vector of probabilities over all possible tokens; in the end a basic LLM is:

$$\boxed{f : \mathbb{R}^{e \times T} \to \mathbb{R}^{n_{\text{tokens}}}}$$

Tokenization and embedding: recap

- the input prompt is tokenized, goes into context window of T tokens (T fixed, pad/truncate if necessary): we have a list of T token indices
  typically context length $T \approx 100\,000$
- each token is mapped to its *embedding* $\in \mathbb{R}^e$: we have a list of T vectors in $\mathbb{R}^e$
  typically embedding size $e \approx 10\,000$
- the input of model f consists of T embeddings, and it outputs a vector of probabilities over all possible tokens; in the end a basic LLM is:

$$\boxed{f : \mathbb{R}^{e \times T} \to \mathbb{R}^{n_{\text{tokens}}}}$$

- in practice your GPT prompt is prefixed with a fixed set of instructions (see answer to "what's your name?")
- how do we do if input prompt is shorter than T?

Tokenization and embedding: recap

- the input prompt is tokenized, goes into context window of T tokens (T fixed, pad/truncate if necessary): we have a list of T token indices
  typically context length $T \approx 100\,000$
- each token is mapped to its *embedding* $\in \mathbb{R}^e$: we have a list of T vectors in $\mathbb{R}^e$
  typically embedding size $e \approx 10\,000$
- the input of model f consists of T embeddings, and it outputs a vector of probabilities over all possible tokens; in the end a basic LLM is:

$$\boxed{f : \mathbb{R}^{e \times T} \to \mathbb{R}^{n_{\text{tokens}}}}$$

- in practice your GPT prompt is prefixed with a fixed set of instructions (see answer to "what's your name?")
- how do we do if input prompt is shorter than T? special `PAD` token
- when does the model decide to stop its answer?

Tokenization and embedding: recap

- the input prompt is tokenized, goes into context window of T tokens (T fixed, pad/truncate if necessary): we have a list of T token indices
  typically context length $T \approx 100\,000$
- each token is mapped to its *embedding* $\in \mathbb{R}^e$: we have a list of T vectors in $\mathbb{R}^e$
  typically embedding size $e \approx 10\,000$
- the input of model f consists of T embeddings, and it outputs a vector of probabilities over all possible tokens; in the end a basic LLM is:

$$\boxed{f : \mathbb{R}^{e \times T} \to \mathbb{R}^{n_{\text{tokens}}}}$$

- in practice your GPT prompt is prefixed with a fixed set of instructions (see answer to "what's your name?")
- how do we do if input prompt is shorter than T? special `PAD` token
- when does the model decide to stop its answer? special `END` token

Refinement: reasoning

- reasoning / thinking / chain of thought: before answering you, the model writes intermediate answers (for itself)

  like a rough copy for during an exam: breaks the problem into steps, tries an approach, checks it, backtracks…

- first observed through prompting: adding “Let’s think step by step” improves math accuracy (*Large Language Models are Zero-Shot Reasoners*, Kojima et al., 2022)

Refinement: pretraining, finetuning and alignment

- LLMs are first pretrained on massive corpora ($\approx$ the whole web); pretraining learns general linguistic and world knowledge
- finetuning = small retraining of pretrained model, on curated input/output pairs, aimed at adapting the model to a specific task (coding, maths)
  keyword: LoRA (Low Rank Adaptation)
- alignment = making sure the model performs as desired in some cases
  keyword: RLHF (Reinforcement Learning with Human Feedback)

"Foundation model" (Bommasani et al., Stanford, 2021): a big model that does various tasks (translation, summarization, coding, Q&A…) for which it was never explicitly trained for (simply next token prediction + finetuning or RLH or nothing)

Refinement: Mixture of Experts (MoE)

- replace one large network by several (≈ 10?) experts networks
- a router (= a neural network) selects which experts process each token
- only a few experts are active at a time
- allows model size to grow while keeping computation nearly constant

Vocabulary: Agents

- so far we've seen the basic version of a LLM, that only produces text
- an *agent* is a LLM that can also take actions by calling *tools*
  web search, running code, reading files, sending an email, clicking in a browser…
- the model f emits a tool call as special tokens with structured arguments
  `<tool> search("weather Lyon") </tool>`
- a program outside the model executes the call and appends the result to the context, then f continues generating next tokens

Vocabulary: World models

- an LLM predicts the next token; a world model predicts the next *state of an environment*, given an action:

$$\boxed{s_{t+1} = g(s_t, a_t)}$$

$\approx$ like playing a video game. keyword: reinforcement learning (RL)

Vocabulary: World models

- an LLM predicts the next token; a world model predicts the next *state of an environment*, given an action:

$$\boxed{s_{t+1} = g(s_t, a_t)}$$

  $\approx$ like playing a video game. keyword: reinforcement learning (RL)

- an agent can then *imagine* the consequences of its actions before taking them, and plan

Outline

Machine and Deep learning in a nutshell

Large language models

Architecture: Transformers and Attention

Transformers

- the T in GPT
- 2017 paper, 281 090 citations as of yesterday



Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

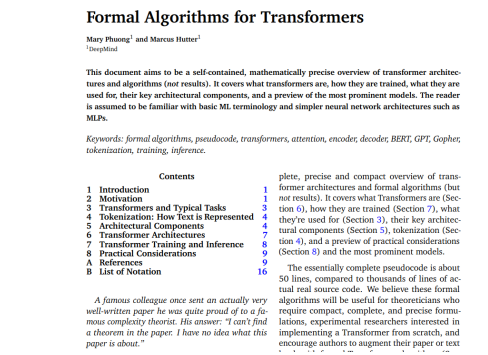
Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

- a special neural network architecture, designed for sequence modelling

Two great ressources

- Phuong & Hutter, *Formal Algorithms for Transformers*
  `https://arxiv.org/abs/2207.09238`
- Karpathy, *Let's build GPT: from scratch, in code, spelled out.*
  `https://www.youtube.com/watch?v=kCc8FmEb1nY`



Formal Algorithms for Transformers

Mary Phuong[1] **and Marcus Hutter**[1]
[1]DeepMind

This document aims to be a self-contained, mathematically precise overview of transformer architectures and algorithms (*not* results). It covers what transformers are, how they are trained, what they are used for, their key architectural components, and a preview of the most prominent models. The reader is assumed to be familiar with basic ML terminology and simpler neural network architectures such as MLPs.

Keywords: formal algorithms, pseudocode, transformers, attention, encoder, decoder, BERT, GPT, Gopher, tokenization, training, inference.

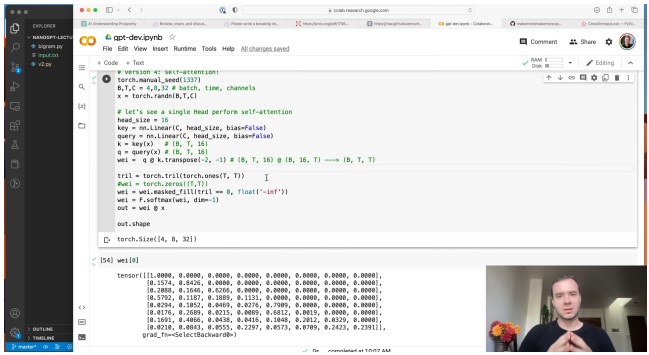
Contents

1 **Introduction** 1
2 **Motivation** 1
3 **Transformers and Typical Tasks** 3
4 **Tokenization: How Text is Represented** 4
5 **Architectural Components** 4
6 **Transformer Architectures** 7
7 **Transformer Training and Inference** 8
8 **Practical Considerations** 9
A **References** 9
B **List of Notation** 16

A famous colleague once sent an actually very well-written paper he was quite proud of to a famous complexity theorist. His answer: "I can't find a theorem in the paper. I have no idea what this paper is about."

plete, precise and compact overview of transformer architectures and formal algorithms (but *not* results). It covers what Transformers are (Section 6), how they are trained (Section 7), what they're used for (Section 3), their key architectural components (Section 5), tokenization (Section 4), and a preview of practical considerations (Section 8) and the most prominent models.

The essentially complete pseudocode is about 50 lines, compared to thousands of lines of actual real source code. We believe these formal algorithms will be useful for theoreticians who require compact, complete, and precise formulations, experimental researchers interested in implementing a Transformer from scratch, and encourage authors to augment their paper or text



The Attention mechanism

- remember our inputs consist of embedded tokens:

$$\mathbf{x} = (\mathbf{t}_1, \ldots, \mathbf{t}_T) \in (\mathbb{R}^e)^T$$

  with e embedding size, T context length

- neural networks are stacked layers
- the *Attention layer* is a layer that maps a sequence of T token embeddings to a sequence of T transformed token embedding:

$$(\mathbf{t}_1, \ldots, \mathbf{t}_T) \mapsto (\mathbf{t}'_1, \ldots, \mathbf{t}'_T) \in (\mathbb{R}^e)^T$$

  I will say “token” when I should say “token embedding”

- Transformers are basically stacked Attention layers (intertwined with fully connected layers)

$$(\mathbf{t}_1, \ldots, \mathbf{t}_T) \mapsto (\mathbf{t}'_1, \ldots, \mathbf{t}'_T) \mapsto (\mathbf{t}''_1, \ldots, \mathbf{t}''_T) \mapsto \ldots$$

The idea of Attention

- the transformed token number i, $\mathbf{t}'_i$, is a combination of all input token values $\mathbf{t}_j$:

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{t}_j$$

- weight λ_{ij} should be large if $\mathbf{t}_i$ similar to $\mathbf{t}_j$, small otherwise
- weights (λ_{ij}) should be positive and sum to 1 over j

The idea of Attention

- the transformed token number i, $\mathbf{t}'_i$, is a combination of all input token values $\mathbf{t}_j$:

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{t}_j$$

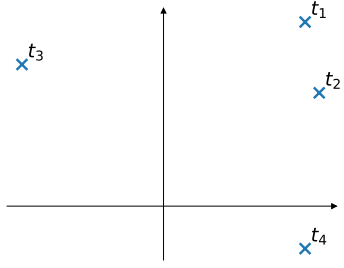
- weight λ_{ij} should be large if $\mathbf{t}_i$ similar to $\mathbf{t}_j$, small otherwise
- weights (λ_{ij}) should be positive and sum to 1 over j
- $\rightsquigarrow$ use $\lambda_{i\cdot} = \mathrm{softmax}\left((\langle \mathbf{t}_i, \mathbf{t}_j \rangle)_{j=1,\dots,T}\right) \in \Delta_T$ Δ is the simplex

The idea of Attention

- the transformed token number i, $\mathbf{t}'_i$, is a combination of all input token values $\mathbf{t}_j$:

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{t}_j$$

- weight λ_{ij} should be large if $\mathbf{t}_i$ similar to $\mathbf{t}_j$, small otherwise
- weights (λ_{ij}) should be positive and sum to 1 over j
- $\rightsquigarrow$ use $\lambda_{i\cdot} = \mathrm{softmax}\left((\langle \mathbf{t}_i, \mathbf{t}_j \rangle)_{j=1,\dots,T}\right) \in \Delta_T$ Δ is the simplex



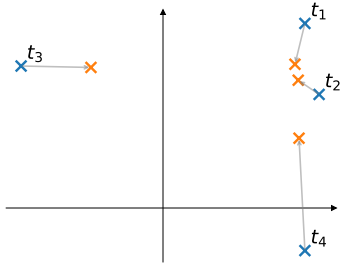
$$(\lambda_{ij}) = \begin{pmatrix} 0.56 & 0.32 & 0.05 & 0.07 \\ 0.47 & 0.35 & 0.04 & 0.13 \\ 0.14 & 0.08 & 0.76 & 0.03 \\ 0.25 & 0.32 & 0.04 & 0.40 \end{pmatrix}$$

The idea of Attention

- the transformed token number i, $\mathbf{t}'_i$, is a combination of all input token values $\mathbf{t}_j$:

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{t}_j$$

- weight λ_{ij} should be large if $\mathbf{t}_i$ similar to $\mathbf{t}_j$, small otherwise
- weights (λ_{ij}) should be positive and sum to 1 over j
- $\rightsquigarrow$ use $\lambda_{i\cdot} = \mathrm{softmax}\left(\left(\langle \mathbf{t}_i, \mathbf{t}_j \rangle\right)_{j=1,\ldots,T}\right) \in \Delta_T$ Δ is the simplex



$$(\lambda_{ij}) = \begin{pmatrix} 0.56 & 0.32 & 0.05 & 0.07 \\ 0.47 & 0.35 & 0.04 & 0.13 \\ 0.14 & 0.08 & 0.76 & 0.03 \\ 0.25 & 0.32 & 0.04 & 0.40 \end{pmatrix}$$

The real Attention

- previous approach is too crude and doesn't work well
- improvement: gain flexibility by using three linear transformations of tokens:

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{V} \mathbf{t}_j$$

with $\lambda_{i\cdot} = \mathrm{softmax}\left((\langle \mathbf{Q}\mathbf{t}_i, \mathbf{K}\mathbf{t}_j \rangle)_{j=1,\dots,T} \right) \in \Delta_T$

key, query, value matrices of adequate size
(different matrices for each attention layer, learnable)

in the real formula there is a scaling factor inside the softmax to avoid saturation

Intuition behind Q, K, V

$$\mathbf{t}'_i = \sum_{j=1}^{T} \lambda_{ij} \mathbf{V}\mathbf{t}_j$$

with $\lambda_{i\cdot} = \mathrm{softmax}\left(\left(\langle \mathbf{Q}\mathbf{t}_i, \mathbf{K}\mathbf{t}_j \rangle\right)_{j=1,\dots,T}\right) \in \Delta_T$

- attention is a “content-based retrieval mechanism”
- query $\mathbf{Q}\mathbf{t}_i$ asks: “what is token i looking for?”
- keys $\mathbf{K}\mathbf{t}_j$ describe: “what does token j contain?”
- values $\mathbf{V}\mathbf{t}_j$ are: “what information does token j pass on?”

Intuition behind Q, K, V

$$\mathbf{t}_i' = \sum_{j=1}^{T} \lambda_{ij} \mathbf{V}\mathbf{t}_j$$

with $\lambda_{i\cdot} = \mathrm{softmax}\left(\left(\langle \mathbf{Q}\mathbf{t}_i, \mathbf{K}\mathbf{t}_j \rangle\right)_{j=1,\dots,T}\right) \in \Delta_T$

- attention is a “content-based retrieval mechanism”
- query $\mathbf{Q}\mathbf{t}_i$ asks: “what is token i looking for?”
- keys $\mathbf{K}\mathbf{t}_j$ describe: “what does token j contain?”
- values $\mathbf{V}\mathbf{t}_j$ are: “what information does token j pass on?”
- matching score:

  $$\langle \mathbf{Q}\mathbf{t}_i, \mathbf{K}\mathbf{t}_j \rangle$$

  measures compatibility between request and memory

The whole Transformer architecture

- chain several Attention layers (≈ 50)
- end up with sequence of T embeddings
- at last layer, use fully connected layer to map to $\mathbb{R}^{n_{\text{tokens}}}$ + softmax to get probability vector of next token

The whole Transformer architecture

- chain several Attention layers ($\approx$ 50)
- end up with sequence of T embeddings
- at last layer, use fully connected layer to map to $\mathbb{R}^{n_{\text{tokens}}}$ + softmax to get probability vector of next token
- not covered: multi-head attention, normalization layers, residual connections, causal masking, positional embedding

What make models different?

GPT, DeepSeek, Gemini, Claude, LeChat, Emmy...

- all based on Transformer architecture
- performance differences arise from:
  - model size (parameters, context length)
  - training data and filtering (quality data matter)
  - training and finetuning
  - broader choices (e.g. mixture of experts)
  - inference-time techniques (chain of thought)

AI and maths

- Navier-Stokes, Open AI 722 papers: some brutality, maths used for flag planting (IPO context) & progress much faster than anticipated
- reasons? flagship problems where well formalized + Lean + exploiting “attack plans” present in the literature + connections between so-far distant fields
- undeniable: complex proofs / computation can be done by LLMs
- (for me) it is hard to envision a future in which significant results can be obtained without AI
- “end of romantic era” (H. Duminil-Copin), “get over a 1000 years-old way of doing maths” (C. Villani)

It is difficult to make predictions, especially about the future

- optimistic view: LLMs empower mathematicians, now we'll build theories and explore more? can learn new topics easily, get feedback; brainstorm
- current system cannot last (?): surge in submissions / desk rejections (ICLR: 20k 2025, 60k 2026; TMLR: 50% desk rejection rate); some journals have paused submissions
  https://medium.com/@TmlrOrg/asking-authors-about-their-own-papers-3d2e04e5dee0
- will our work be about understanding AI-generated proofs and disseminating the results? is this fun?
- less importance to individual results, big collaborative projects, physics/biology -like?
- all phenomena were already present (submission increase, bad reviews, inequalities between labs), just moved one order of magnitude up?
- develop an ethics of mathematics?

Ressources

- Maths et IA à midi (MIAM) 45 min seminar every other Thursday, with replays: https://smf.emath.fr/actualites-smf/seminaire-miam
- Various opinion of 5 researchers: https://www.liberation.fr/sciences/cinq-chercheurs-en-maths-face-aux-dernieres-avancees-dopenai-avec-lia-il-y-a-6ZCC5SMFT5BJ3OLSU6POHGTGYQ/
- Yves Meyer & Stéphane Jaffard column: https://www.lemonde.fr/sciences/article/2026/11/05/le-grand-danger-serait-une-vision-de-la-recherche-mathematique-cantonnee-a-sa-6806850_1650684.html
- 3blue1brown: “There’s never a better time to get into mathematics” https://www.youtube.com/watch?v=LKrwhjOI8Nc

Some hope from Meyer & Jaffard column?

Mais les projecteurs braqués sur quelques personnalités cachent l'essentiel de l'activité de recherche, qui s'effectue, en particulier, en interaction avec les autres sciences et la technologie. L'art du mathématicien consiste alors à déceler et à comprendre les blocages mathématiques qui s'y trouvent, puis à les formaliser grâce à des allers-retours avec les spécialistes de ces domaines, ces activités nécessitant de surmonter les multiples incompréhensions liées aux différences de cultures entre disciplines afin d'interpréter correctement les résultats fournis par les dispositifs expérimentaux. Tout ce travail n'a pas pour point de départ des questions bien formalisées et susceptibles d'être soumises à une IA.