

Data Science 2

Large Language Models

Ondřej Týbl

Charles University, Prague

For problems related to natural language processing, such as *Sentiment Analysis, Machine Translation, Text Summarization* or *Chatbots* we use so called *Large Language Models (LLMs)*.

Here, *large* refers to the number of parameters but the key feature is so called *self-attention* present in most of the current LLMs¹.

¹Historically, Recurrent Neural Networks (RNNs) played an important role, see Chapter 10 of Deep Learning. In most use-cases RNNs have been replaced by LLMs now with a rare exception of time-series. However, for time-series RNNs have not convincingly beaten classical statistical approaches yet in majority of use-cases

1. Architecture

Transformer

Self-Attention

2. Training

3. Evaluation

Architecture

The key idea of *self-attention* can be traced back to a famous paper *Attention is All You Need* from 2017 (see [7]) which gained over 180,000 citations as of 2025.

Following a great tutorial in [1] we describe the network architecture for machine translation.

Transformer

Our network assumes a fixed sentence length, i.e. a maximum number of encoded (sub)words is given

- for shorter sentences, we use *padding* with a special (sub)word to get the desired length,
- for longer sentences, we simply discard the end of the text

For GPT-4 we have up to 8,192 (sub)words, for GPT-4o its up to 128,000.



Figure 1: In our example we suppose architecture with 3 (sub)words. Before fed into the network we begin by encoding each (sub)word into a vector using some embedding algorithm. If d is the embedding dimension, the input is a $3 \times d$ matrix.

Transformer

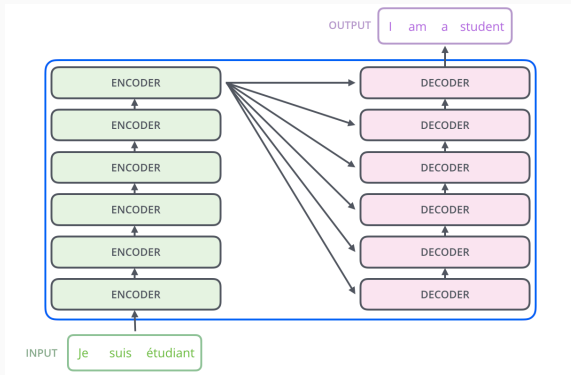


Figure 2: High-level transformer architecture. The sentence enters a stack of *encoders* which outputs a sentence representation which is then inputted into a stack of *decoders*. Encoders (and decoders) share inner structure, but weights are different.

Self-Attention

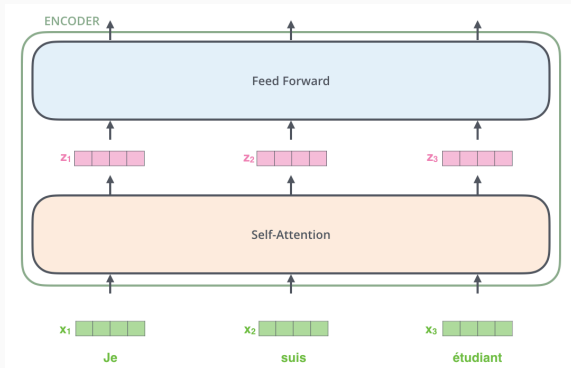


Figure 3: Detailed overview of encoder. It is inputted with a sentence representation from previous step and *self-attention* and *feed forward* layers are applied. The representation dimension is kept constant.

Dependencies among (sub)words are only in the self-attention layer.

Self-attention is a mechanism that allows dynamic connections among (sub)words.

The animal didn't cross the street because it was too tired

Here, *it* refers to *the animal*. Self-attention allows to look at other positions in the sentence for clues.

Self-Attention

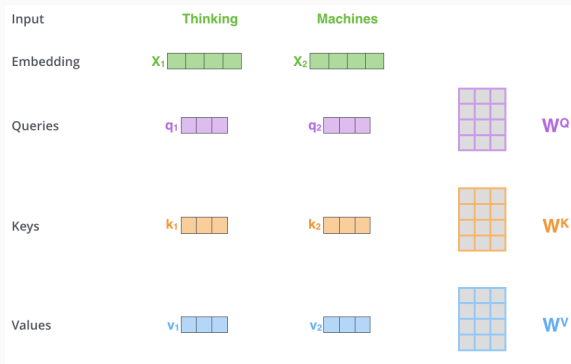


Figure 4: Each (sub)word represented by a vector in \mathbb{R}^d is associated with a triplet of vectors in \mathbb{R}^{d_k} : query, key, value. This is done through matrix multiplication and typically $d_k < d$ for computation feasibility.

- The triplet *query*, *key*, *value* are abstractions that are useful for calculating and thinking about attention.
- Mathematically, those are linear transformations of the embedding vectors, where the transformation matrix is a trainable parameter to be estimated from the data, that is

$$q_i = x_i W^Q, \quad k_i = x_i W^K, \quad v_i = x_i W^V$$

for each $i = 1, \dots, N$, where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$

Self-Attention

We shall combine queries, keys and values for all (sub)words into

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

where $X = (x_1^T, \dots, x_N^T)^T \in \mathbb{R}^{N \times d}$. Then self-attention is computed according to the schema in Figure 5, where the softmax is applied row-wise and the normalization factor $\sqrt{d_k}$ is described in [7].

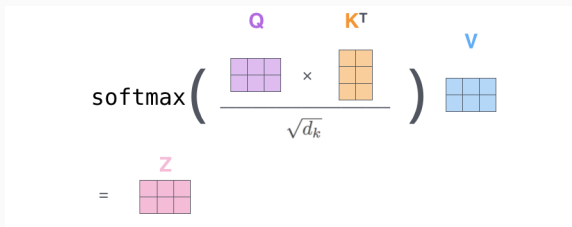


Figure 5: Example of self-attention output with $N = 2$ (sub)words encoded in \mathbb{R}^3 , $d = 3$ and $d_k = 2$.

Self-Attention

Input

Embedding

Queries

Keys

Values

Score


Divide by $8 (\sqrt{d_k})$

Softmax

Softmax
X
Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$


14

0.88

v_1 

z_1 

Machines

x_2 

q_2 

k_2 

v_2 

$q_2 \cdot k_2 = 96$

12

0.12

v_2 

z_2 

For each (sub)word, the self-attention formula produces a convex combination of values of all the (sub)words with coefficients given by similarity of the query/key pairs, where the similarity is computed using scalar product.

Self-Attention

	When	you	play	the	game	of	thrones
When	1.00	0.00	0.00	0.00	0.00	0.00	0.00
you	0.00	0.97	0.03	0.00	0.00	0.00	0.00
play	0.00	0.00	0.68	0.00	0.10	0.00	0.22
the	0.00	0.00	0.00	0.65	0.14	0.00	0.21
game	0.00	0.00	0.03	0.02	0.72	0.00	0.23
of	0.00	0.00	0.00	0.00	0.00	1.00	0.00
thrones	0.00	0.00	0.05	0.17	0.00	0.00	0.75

Figure 6: Example of the distributions $\text{softmax}(QK^T)$ for a sentence with $N = 6$ (sub)words. The rows represent probability distributions.

Self-Attention

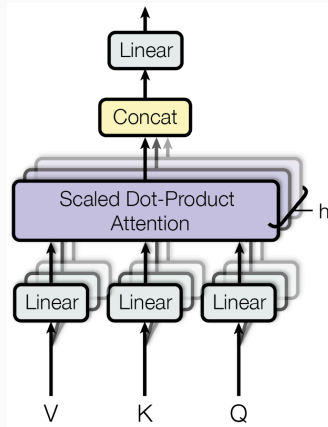


Figure 7: Our self-attention layer in fact involves h independent self-attention *heads* to allow more variability. Here, *concat* denotes concatenation of vectors and *linear* is a fully connected layer. Taken from [7].

Self-Attention

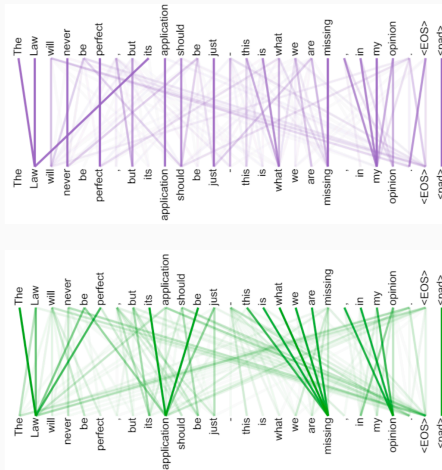


Figure 8: Example of the distributions $\text{softmax}(QK^T)$ for a sentence with $N = 26$ (sub)words. Two separate self-attention heads for each color with stronger color indicating stronger relation.

Self-Attention

Now back to the encoder as in Figure 3 now in more detail.

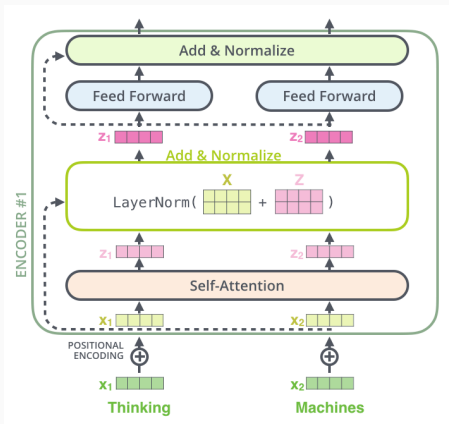


Figure 9: Encored architecture in detail.

In Figure 9 we have

- (sub)word encodings are enhanced with information on the position within sentence by summation with *positional encoding* (details below),
- self-attention with residual connection,
- *layer normalization*, similar to batch normalization but we normalize in the feature dimension d (separately for each example),
- then *feed forward network* which is simple multilayer perceptron with one hidden layer for each (sub)word separately,
- finally again residual connection and layer normalization

Self-Attention

Finally, we look at the entire architecture of Figure 2 in detail.

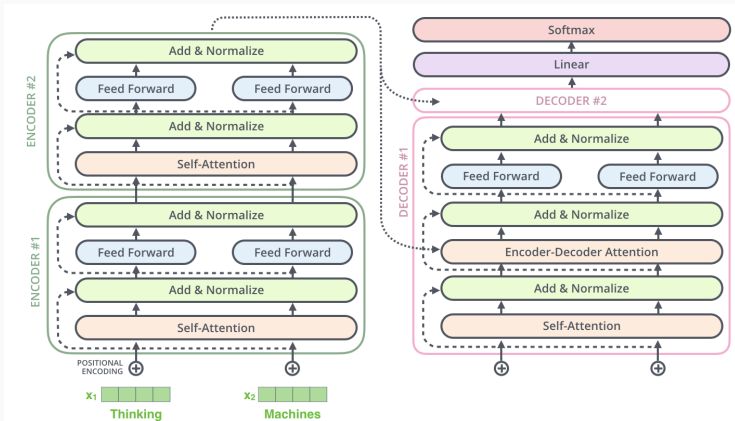


Figure 10: Transformer architecture in detail.

In Figure 10 we have

- L encoders and decoders in general, encoders are as in Figure 9,
- the final encoder outputs a sentence representation which is then fed into all the decoders
- decoder is inputted with
 - training: the same input as encoder, but the self-attention mask is added so that the (sub)word can reach only already seen (sub)words in the sentence²,
 - inference: output of the whole network one by one as inference progresses
- *encoder-decoder attention* refers to the same procedure as self-attention, now with queries coming from the decoder and keys and values from the encoder

²Implemented by overwriting the all values of the upper triangle of QK^T with $-\infty$

The formula for self-attention is permutation invariant, that is, it does not depend on the (sub)word position within the sentence which contradicts our understanding of language.

We would like to improve our (sub)word encodings $x_1, \dots, x_N \in \mathbb{R}^d$ with position encodings.

One-hot encoding again has several drawbacks³.

³Compare to word embeddings

Self-Attention

We want

- unique encoding for each position in sentence,
- natural generalization irrespective of sentence length (that is, we want a relative position),
- distance between two positions depends only on their difference

Definition (Positional Encoding)

Suppose a (sub)word with i th position in a sentence and its encoding $x \in \mathbb{R}^d$. We define the *position encoding*, $\text{pos} \in \mathbb{R}^d$ by

$$\text{pos}_{i,2j} = \sin\left(\frac{i}{c^{\frac{2j}{d}}}\right), \quad \text{pos}_{i,2j+1} = \cos\left(\frac{i}{c^{\frac{2j}{d}}}\right), \quad j = 1, \dots, d/2$$

for some $c > 0$.

The final i th (sub)word encoding is defined as

$$x + \text{pos}_i.$$

Self-Attention

We have (see link) that for any k there exists $M_k \in \mathbb{R}^{2 \times 2}$ such that

$$\text{pos}_i = M_k \text{pos}_{i+k},$$

which allows the network to easily learn to attend relative positions.

Moreover, the scalar product and distance depend only on the offset.

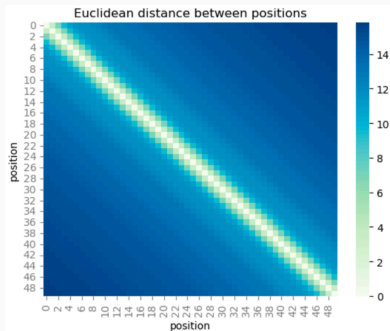
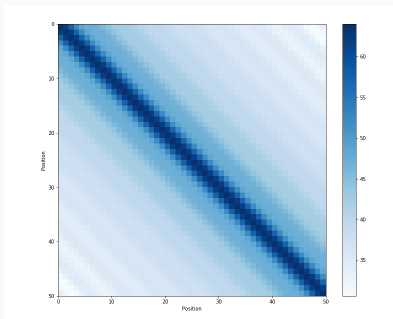


Figure 11: Scalar products (left) and Euclidean distances (right) of $\text{pos}_i, \text{pos}_j$.

Training

The architecture in Figure 10 contains both encoder and decoder and was originally proposed for the machine translation task.

The original setting:

- number of encoders and decoders $L = 6$,
- (sub)word encoding dimensionality $d = 1024$ with 37K (sub)words in total,
- number of hidden cells in the feed forward sublayer $d_{ff} = 4096$,
- number of heads in self-attention $h = 16$,
- query, key and value dimensionality $d_k = 64$,
- in total $> 200\text{M}$ parameters; trained with 300K iterations on each language pair with loss being cross-entropy per (sub)word excluding padded positions

But there are situations where encoder-only or decoder-only architectures are desirable.

If we take encoder-only architecture, then full self-attention allows to look both to the right and to the left while decoder-only contains mask to prevent from looking ahead in the sentence.

see printscreens: compare use cases for encoder-decoder, encoder and decoder, add training part from <https://jalammar.github.io/illustrated-transformer/>

GPT (decoder-only), see [6]

Trained on a large text to predict next (sub)word in a sentence.

Good for text generation.

Applied as a chat-bot: our question is considered as a start of a sentence and the model tries to finish it.

BERT (encoder-only), see [3]

Trained on a large text to predict masked (sub)words within the sentence (described on the next slide).

Good for semantic understanding (the model sees the whole sentence).

Applied as a sentiment classifier, named entity recognizer, (sub)word encoder,... These applications need *fine-tuning* – after initial training we train *few* iterations more on a specific task with labeled data⁴.

⁴The key point here is that we started with training with a large dataset cheap to obtain but when needed on a specific task where the data are expensive, we need just a fraction of the data size.

When training BERT on masked (sub)words predictions (i.e. filling up a sentence after erasing some (sub)words, we need to choose carefully the loss function.

First, we mask 15% of (sub)words in each sentence and we predict them.

- 80% out of these are replaced by a special token *MASK* to truly represent missing information,
- 10% out of these are replaced by a random (sub)word⁵
- 10% out of these are left there without any change

⁵In this way we make the model not being able to distinguish which (sub)words are correct and which not.

Warning

Architectures with self-attention usually suffer from over-fitting as the quadratic term allows too much variability. Therefore, such models typically work well only when the training dataset is large enough to cover all scenarios and good generalization is not needed.

The common understanding is that such models lack *inductive bias* (in contrast to convolution networks in computer vision which are based on the ideas like shift invariance or local interactions).

Training

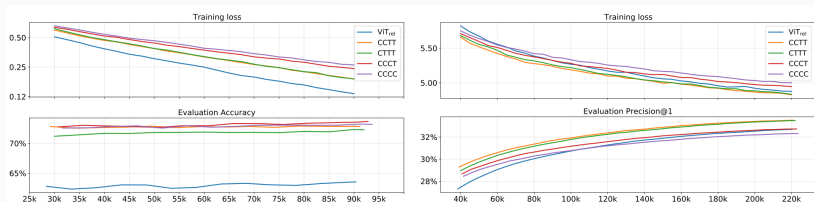


Figure 12: Self-attention-only model (blue) compared to models mixed with convolutions on image classification. On the left we train on smaller dataset ImageNet and self-attention-only model performs the best on train but worst on the test dataset. When the dataset is large enough (JFT, the dataset on the right) the difference disappears. Taken from [2]

Evaluation

The problem of performance evaluation for text generation is complex.

Evaluation metric⁶ should express the ability of the model to fulfill given task.

⁶Do not confuse with training loss which is directly optimized when optimizing the parameters.

Therefore, simply counting the number of correctly generated (sub)words, i.e. *accuracy*, does not suit well here as

- sometimes a one word difference in sentence completely changes the meaning,
- we appreciate non-deterministic behavior, i.e. we want our model to give unique answers

Examples:

- *I like it not vs. I like it.*
- *The match was spectacular. vs The game was amazing.*

One word does not capture the meaning, but a couple of words might to better.

Definition (n-gram)

An n -gram is any sequence of n (sub)words⁷ in a text. We say unigram instead of 1-gram and bi-gram instead of 2-gram.

⁷The use either of words or subwords depends on chosen embedding.

Instead of comparing (sub)words, we shall compare n -grams. Now one more decision has to be made – do we care more about *precision* or *recall*?

We prioritize recall when it's critical to capture all relevant information (e.g., summarization or medical diagnosis), and precision when it's more important that retrieved or generated content is highly relevant (e.g., document retrieval or spam detection).

Denote the n -gram precision, recall and F score for a given pair of *candidate* and *reference* texts

$$R_n = \frac{\#n\text{-grams both in candidate and reference}}{\#n\text{-grams in reference}}$$

$$P_n = \frac{\#n\text{-grams both in candidate and reference}}{\#n\text{-grams in candidate}}$$

$$F_n = \frac{2R_nP_n}{R_n + P_n}$$

Definition (ROUGE), see [4]

ROUGE (Recall-Oriented Understudy for Gisting⁸ Evaluation) of order n (ROUGE- n) includes the set of values: recall (R_n), precision (P_n), and F-score (F_n).

There is a common confusion in the community regarding the definition of ROUGE. The original paper—and the name itself—primarily refer to the recall component (R_n). However, in practice, implementations such as PyTorch's return the full set: R_n , P_n , and F_n .

⁸*Gist* can be interpreted as *essence*

Another variant of ROUGE uses the *longest common subsequence* instead of n -grams.

Definition (ROUGE-L), see [4]

(ROUGE-L) is equal to the set of values

$$R_L = \frac{\text{length of the longest common subsequence}}{\text{\#unigrams in reference}}$$

$$P_L = \frac{\text{length of the longest common subsequence}}{\text{\#unigrams in candidate}}$$

$$F_L = \frac{2R_n P_n}{R_n + P_n}$$

The idea behind ROUGE-L is that the sentences

My father, who was born in Texas, enjoys ice hockey.

and

My father enjoys ice hockey.

have similar meaning. If the latter is the reference sentence, then ROUGE-L is equal to 1 here as subsequences can avoid inserted subsentences.

Example I

Suppose reference text

It is cold outside.

and a candidate text

It is very cold outside.

Then we have

$$R_1 = \frac{4}{4}, \quad P_1 = \frac{4}{5}, \quad F_1 = 0.89$$

$$R_2 = \frac{2}{3}, \quad P_2 = \frac{2}{4}, \quad F_2 = 0.57$$

$$R_L = \frac{2}{4}, \quad P_L = \frac{2}{5}, \quad F_L = 0.44$$

Some metrics are vulnerable to word shuffle.

Example II

Suppose reference text

It is cold outside.

and a candidate text

Outside it is cold.

Then we have

$$\begin{aligned} R_1 &= \frac{4}{4}, & P_1 &= \frac{4}{5}, & F_1 &= 0.89 \\ R_2 &= \frac{0}{3}, & P_2 &= \frac{0}{4}, & F_2 &= 0.00 \\ R_L &= \frac{1}{4}, & P_L &= \frac{1}{5}, & F_L &= 0.22 \end{aligned}$$

Evaluation

Another common metric type is based solely on precision thus focusing on text relevance. It combines different (sub)word sequences as follows.

Definition (BLEU), see [5]

For a candidate and reference pair texts with precisions P_1, \dots, P_n we define their Bilingual Evaluation Understudy of order n (BLEU _{n}) as

$$\text{BLEU}_n = c \left(\prod_{i=1}^n P_i \right)^{\frac{1}{n}},$$

where the constant c , called a *brevity penalization* is given by

$$c = \begin{cases} 1 & \text{if } |\text{candidate}| \geq |\text{reference}| \\ e^{1 - \frac{|\text{reference}|}{|\text{candidate}|}} & \text{otherwise} \end{cases}$$

Few notes on BLEU:

- usually, we take $n \sim 4$,
- we may also consider weights in the geometric average (see [5]),
- BLEU as well as ROUGE have no semantic understanding and are based on the words only (hoping that the high orders explain the semantic well),
- comparability is very low across different datasets and languages
- several metrics which are more robust against synonyms or word order have been introduced, see e.g. METEOR, generally, these differ by: preference over precision/recall, length penalty strategies, weights for different (sub)words combinations



J. Alammam.

The illustrated transformer.

The Illustrated Transformer–Jay Alammam–Visualizing Machine Learning One Concept at a Time, 27:1–2, 2018.



Z. Dai, H. Liu, Q. V. Le, and M. Tan.

Coatnet: Marrying convolution and attention for all data sizes.

Advances in neural information processing systems,
34:3965–3977, 2021.



J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova.

Bert: Pre-training of deep bidirectional transformers for language understanding.

In Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), pages 4171–4186, 2019.



C.-Y. Lin.

Rouge: A package for automatic evaluation of summaries.

In Text summarization branches out, pages 74–81, 2004.



K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu.

Bleu: a method for automatic evaluation of machine translation.

In Proceedings of the 40th annual meeting of the Association for Computational Linguistics, pages 311–318, 2002.



A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al.

Improving language understanding by generative pre-training.
2018.



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin.

Attention is all you need.

Advances in neural information processing systems, 30, 2017.