2 – Sequence processing architectures

IASD App – LLMs course

Florian Le Bronnec

January 29, 2025

Table of Contents

Table of Contents

Document vectors

In order to process text in most of the machine learning algorithms, we need to convert it into a numerical representation.

What we can do:

- Word-based features: count the numbers of times a word / n-gram appears in the document.
- Process the documents based on the words they contain, using learned numerical rules.

Example

- **Sentence**: "I love this movie."
- **Word count**: {I: 1, love: 1, this: 1, movie: 1}.
- **Document vector**: $(0, \dots, 0, 1, 0, \dots, 0, 1, 0 \dots, 0, 1, 0 \dots, 0, 1, 0, \dots, 0)$ of size V.

Existing document vectors methods

- **Bag of words**: Naive counting (like the previous example).
- **Frequency-based**: Normalize by the total number of words in each document.

Despite being simple, they can provide extremely good results.

TF-IDF

Intuition: words that appear in many documents are less informative than words that appear in few documents.
TF-IDF is a way to normalize the word frequencies by the number of documents in which the word appears over the whole corpus.

TF-IDF formula

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

- $f_{t,d}$: Frequency of term t in document d.
- $\sum_{t' \in d} f_{t',d}$: Total number of terms in document d.

$$\mathrm{IDF}(t) = \log\left(\frac{N+1}{\mathrm{DF}(t)+1}\right) + 1$$

- N: Total number of documents.
- $\mathrm{DF}(t)$: Number of documents containing the term t.

TF-IDF:

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

Word vectors

Document vectors provide a way to represent a document with a single vector, aggregating the information of all the words in the document.

But for some applications, we need finer representations:

- Classification tasks over words themselves: Named Entity Recognition, Part-of-Speech tagging, etc.
- Grammatical tasks: parsing, etc.
- Generating texts: machine translation, summarization, etc.

Word vectors

Document vectors provide a way to represent a document with a single vector, aggregating the information of all the words in the document.

But for some applications, we need finer representations:

- Classification tasks over words themselves: Named Entity Recognition, Part-of-Speech tagging, etc.
- Grammatical tasks: parsing, etc.
- Generating texts: machine translation, summarization, etc.

Word embeddings

Assign to each word a vector in a continuous space, of dimension d.

- **Intuition:** words with similar meanings should be close in the vector space.
- We learn the embeddings from a large corpus.

Word embeddings learning methods

- Word2Vec, learn embeddings by predicting the context of a word.
- GloVe, learn embeddings by predicting the co-occurrence of words.

We can then perform classification using either each word vectors or averaging it into a document vector.

Word embeddings learning methods

- Word2Vec, learn embeddings by predicting the context of a word.
- GloVe, learn embeddings by predicting the co-occurrence of words.

We can then perform classification using either each word vectors or averaging it into a document vector.

Question: Do you see any limitation with this approach?

Word embeddings learning methods

- Word2Vec, learn embeddings by predicting the context of a word.
- GloVe, learn embeddings by predicting the co-occurrence of words.

We can then perform classification using either each word vectors or averaging it into a document vector.

Question: Do you see any limitation with this approach?

- The order of the words is not taken into account.
- The meaning of a word can change depending on the context.

Word embeddings learning methods

- Word2Vec, learn embeddings by predicting the context of a word.
- GloVe, learn embeddings by predicting the co-occurrence of words.

We can then perform classification using either each word vectors or averaging it into a document vector.

Question: Do you see any limitation with this approach?

- The order of the words is not taken into account.
- The meaning of a word can change depending on the context.

We would like to capture interactions between words.

Solution: find some ways to make the vectors depend on the context.

Table of Contents

Text convolution models

1D convolutional networks

For a kernel size of $2k+1$:

$$y_i = F_w(x_{i-k}, \ldots, x_i, \ldots, x_{i+k}), \tag{1}$$

$$= \sum_{j=-k}^{k} w_j x_{i+j}. \tag{2}$$

- **Advantage:** fast, known to work well with images.
- **Limitation:** fixed-size receptive field, texts are not exactly like images, contextualization does not depend on the sequence.

Recurrent neural networks

Recurrent neural networks

RNNs are based on a markovian assumption: the next word embedding depends on the previous one.

$$\boldsymbol{h}_1 = \boldsymbol{0}, \quad \boldsymbol{y}_i, \boldsymbol{h}_i = F_\theta(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i), \tag{3}$$

$$\begin{cases} \boldsymbol{h}_i & = \tanh(\boldsymbol{W}_h \boldsymbol{h}_{i-1} + \boldsymbol{W}_x \boldsymbol{x}_i), \\ \boldsymbol{y}_i & = \boldsymbol{W}_y \boldsymbol{h}_i. \end{cases} \tag{4}$$

- Capture interactions over the whole sequence.
- **Advantage:** receptive field is theoretically infinite.
- **Limitation:** slow to train, hard to capture long-range dependencies.

A new sequence to sequence model

Requirements:

1. Ability to explicitly capture all the interactions between words.
2. Fast and efficient computation.

A new sequence to sequence model

Requirements:

1. Ability to explicitly capture all the interactions between words.
2. Fast and efficient computation.

(1) $\implies$ Each word contextualization should explicitly depend on all the other words.

(2) $\implies$ Use mainly "vectorized" operations (no for-loops), for each parallel computation on GPUs.

Building attention

With convolution we had:

$$\boldsymbol{y}_i = \sum_{j=-k}^{k} w_j \boldsymbol{x}_{i+j}.$$

This is a **weighted sum** of the input.

Requirements

- ✓ **Weighted sum** of the input (linear operation).
- ✓ **Explicit dependenc**y on the other words.
- ✗ **Fixed-size** receptive field.

Extended convolution

Extended convolution

We would like to be able to do something like this:

$$\boldsymbol{y}_i = \sum_{j=1}^{L} w_j \boldsymbol{x}_j .$$

Requirements

- ✓ **Weighted sum** of the input (linear operation).
- ✓ **Explicit dependenc**y on the other words.
- ✓ **Large** receptive field.
- ✗ **Training asymmetry:** During training, weights at a large index won't be updated as much as weights at a small index.

Extended convolution

We would like to be able to do something like this:

$$\boldsymbol{y}_i = \sum_{j=1}^{L} w_j \boldsymbol{x}_j.$$

Requirements

- ✓ **Weighted sum** of the input (linear operation).
- ✓ **Explicit dependenc**y on the other words.
- ✓ **Large** receptive field.
- ✗ **Training asymmetry:** During training, weights at a large index won't be updated as much as weights at a small index.

Solution: Compute the weights $w_1, \ldots, w_L$ dynamically based on the input $\boldsymbol{x}$!

Dynamic weights

We would like something like this:

$$\boldsymbol{y}_i = \sum_{j=1}^{L} s(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{x}_j.$$

Any idea of a function s that could work?

Dynamic weights

We would like something like this:

$$\boldsymbol{y}_i = \sum_{j=1}^{L} s(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{x}_j.$$

Any idea of a function s that could work?

Scalar product!

$$s(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j.$$

$\Longrightarrow$ We get back to the original intuition that similar words should have similar vectors.

Query / key vectors

Input projection

To add expressiveness to the model, attention uses three different linear projections:

$$\begin{cases} \boldsymbol{q}_i & = \boldsymbol{Q}\boldsymbol{x}_i \in \mathbb{R}^d, \\ \boldsymbol{k}_j & = \boldsymbol{K}\boldsymbol{x}_j \in \mathbb{R}^d, \\ \boldsymbol{v}_j & = \boldsymbol{V}\boldsymbol{x}_j \in \mathbb{R}^d, \end{cases},$$

and $s_{ij} = \boldsymbol{q}_i^T \boldsymbol{k}_j$. Then $\boldsymbol{y}_i = \sum_{j=1}^{L} s_{ij} \boldsymbol{v}_j$.

- **Query:** the embedding we want to contextualize.
- **Key:** the embeddings we want to contextualize with.
- **Value:** the embeddings we want to output.

Attention weights

To make attention really expressive, we actually normalize the scalar products with a softmax:

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{l=1}^{L} \exp(s_{il})} \in \mathbb{R}^{L}, \tag{5}$$

- This ensures a normalized sum of the weights, $\boldsymbol{y}_i = \sum_{j=1}^{L} \alpha_{ij} \boldsymbol{v}_j$, has the same scale as the input.
- We have an non-negative weighted sum of the input. Non-negative weights are important for the model to learn to ignore irrelevant words.
- The softmax add a non-linearity to the model, important for learning.

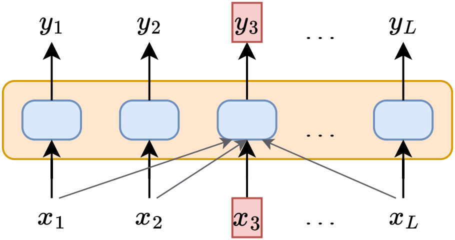

Figure 1: Each input is contextualized by the mean of **attention**.

Attention

$$\begin{cases} \boldsymbol{q}_i &= \boldsymbol{Q}\boldsymbol{x}_i \in \mathbb{R}^d, \\ \boldsymbol{v}_j &= \boldsymbol{V}\boldsymbol{x}_j \in \mathbb{R}^d, \\ \boldsymbol{k}_j &= \boldsymbol{K}\boldsymbol{x}_j \in \mathbb{R}^d, \end{cases} \quad \begin{cases} s_{ij} &= \boldsymbol{q}_i^T \boldsymbol{k}_j \in \mathbb{R},\ 1 \leq j \leq L, \\ \alpha_i &= \mathrm{Softmax}(\boldsymbol{s}_i) \in \mathbb{R}^L, \\ \boldsymbol{y}_i &= \sum_{j=1}^{L} \alpha_{ij} \boldsymbol{v}_j \in \mathbb{R}^d. \end{cases} \tag{6}$$

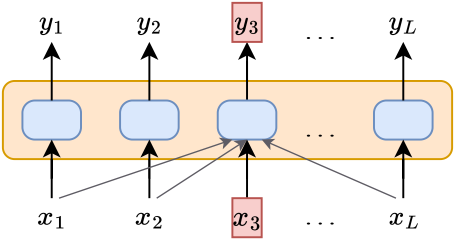

Figure 1: Each input is contextualized by the mean of **attention**.

Attention

$$\begin{cases} \boldsymbol{q}_i &= \boldsymbol{Q}\boldsymbol{x}_i \in \mathbb{R}^d, \\ \boldsymbol{v}_j &= \boldsymbol{V}\boldsymbol{x}_j \in \mathbb{R}^d, \\ \boldsymbol{k}_j &= \boldsymbol{K}\boldsymbol{x}_j \in \mathbb{R}^d, \end{cases} \quad \begin{cases} s_{ij} &= \boldsymbol{q}_i^T \boldsymbol{k}_j \in \mathbb{R},\ 1 \leq j \leq L, \\ \alpha_i &= \text{Softmax}(\boldsymbol{s}_i) \in \mathbb{R}^L, \\ \boldsymbol{y}_i &= \sum_{j=1}^L \alpha_{ij} \boldsymbol{v}_j \in \mathbb{R}^d. \end{cases} \tag{6}$$

$\implies$ The output $\boldsymbol{y}_i$ is simply a **non-negative weighted sum** of the input.

Standard tranformer model

Then we can build **deep networks** around this attention block.

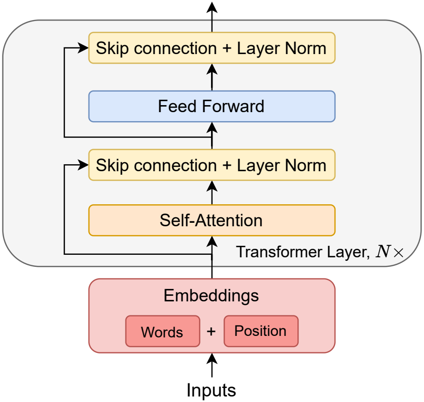

Figure 2: Standard stack of transformer layers.

Feed Forward

Feed Forward are used to make the model more expressive. They are basically two big linear layers with a non-linearity in between.

Typical Feed Forward

$$\begin{cases} \boldsymbol{z}_i & = \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_i + \boldsymbol{b}_1) \in \mathbb{R}^{4 \times d}, \\ \boldsymbol{y}_i & = \boldsymbol{W}_2 \boldsymbol{z}_i + \boldsymbol{b}_2 \in \mathbb{R}^{d}. \end{cases} \tag{7}$$

Layer norm and skip-connection

Layer norm

$$\begin{cases} \boldsymbol{\mu} &= \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{x}_i, \\ \boldsymbol{\sigma} &= \sqrt{\frac{1}{d}\sum_{i=1}^{d} (\boldsymbol{x}_i - \boldsymbol{\mu})^2}, \\ \boldsymbol{y}_i &= \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}. \end{cases} \tag{8}$$

Skip-connection

$$\boldsymbol{y}_i = \boldsymbol{x}_i + \mathrm{LayerNorm}(\mathrm{Attention}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L)). \tag{9}$$

Break

Break!

How should we use it in practice?

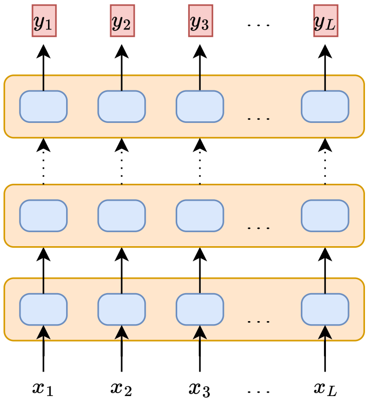

Figure 3: $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L \in \mathbb{R}^d$ are **deep contextualized words embeddings**.

Challenge: make these embeddings as expressive as possible.

Subsequent questions

How did we obtain such impressive performances in NLP?

Observations

1. LLMs can perform well on almost all classical NLP tasks, despite **not having being trained for it**.
2. LLMs **perform better than human annotators** on some tasks [6].
3. LLMs are **big**.

Subsequent questions

How did we obtain such impressive performances in NLP?

Observations

1. LLMs can perform well on almost all classical NLP tasks, despite **not having being trained for it**.
2. LLMs **perform better than human annotators** on some tasks [6].
3. LLMs are **big**.

Conclusion

- 1. and 2. implies that the model did not use specific annotated data.
- 3. means that the model used a lot of data.

Subsequent questions

How did we obtain such impressive performances in NLP?

Observations

1. LLMs can perform well on almost all classical NLP tasks, despite **not having being trained for it**.
2. LLMs **perform better than human annotators** on some tasks [6].
3. LLMs are **big**.

Conclusion

- 1. and 2. implies that the model did not use specific annotated data.
- 3. means that the model used a lot of data.

$\Longrightarrow$ LLMs work because they perform a lot of **self-supervised** learning.

Table of Contents

Why pre-training?

Huge expressive power of transformers [10]. $\Longrightarrow$ Leverage lots of data.

Why pre-training?

Huge expressive power of transformers [10]. $\Longrightarrow$ Leverage lots of data.

Good news

There are lots of text data.

Bad news

Few annotated data.

Why pre-training?

Huge expressive power of transformers [10]. $\Longrightarrow$ Leverage lots of data.

Good news

There are lots of text data.

- Common Crawl [11] **250 billions web pages**.
- The Pile [5] **825GiB** of texts from diverse sources (web, books, professional resources, etc).

Bad news

Few annotated data.

- GLUE (a very popular benchmark on text classification) is made of datasets between and **780 and 400K documents**.
- TriviaQA (answering questions about a given text) is made of **650K documents**.

What is a good-pretraining?

We have a lot of un-annotated data $\implies$ Self-supervised learning.

What is a good-pretraining?

We have a lot of un-annotated data $\implies$ Self-supervised learning.
Challenge: find a pre-training *close-enough* to the target tasks.

Pre-training formulation

$$\mathcal{D} = \{x_i\}_i \xrightarrow{\mathcal{T}} \hat{\mathcal{D}} = \{\tilde{x}_j, \tilde{y}_j\}_j,$$

where $\mathcal{T}$ is any transformation over a document.
The model is trained on a loss:

$$\min_{\theta} \mathcal{L}\left(\mathrm{LLM}_{\theta}(\tilde{x}),\, \tilde{y}\right).$$

Different pre-training

Since there are several tasks in NLP, there exists **different pre-training**.
We will focus on the main ones:

- Classification,
- Generation.

$\Longrightarrow$ All the most used models derive from one of those pre-trainings.

Pre-training for classification

Goal of text classification

Infer the **global meaning** of texts, through **words in their context**.

Pre-training for classification

Goal of text classification

Infer the **global meaning** of texts, through **words in their context**.

$\Longrightarrow$ Emphasize words contextualization during pre-training!

Pre-training for classification

Goal of text classification

Infer the **global meaning** of texts, through **words in their context**.

$\Longrightarrow$ Emphasize words contextualization during pre-training!

You have already done it!

Word2Vec as a pre-training

You constructed $\tilde{\mathcal{D}}$ by extracting positive and negative context.

- $\tilde{x} = \begin{cases} (w,\ C^{+}) \\ \text{or} \\ (w,\ C^{-}). \end{cases}$
- $\tilde{y} = \begin{cases} 1 & \text{if } C^{+}, \\ 0 & \text{otherwise.} \end{cases}$

Word2Vec as a pre-training

You constructed $\tilde{\mathcal{D}}$ by extracting positive and negative context.

- $\tilde{x} = \begin{cases} (w,\ C^+) \\ \text{or} \\ (w,\ C^-). \end{cases}$
- $\tilde{y} = \begin{cases} 1 & \text{if } C^+, \\ 0 & \text{otherwise.} \end{cases}$

Let's see how this can be pushed further with transformers.

BERT – Introduction

Let's now dive into bigger experiments: **BERT** [1].

- BERT was an important milestone.
- Impressive performance that yields to the **massive adoption of transformers**.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Avg -
Pre-GPT SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
ELMO++	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{base}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\textbf{BERT}_{\textbf{large}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: Results on GLUE dataset, 9% of relative amelioration on average on an extremely competitive dataset.

BERT – Introduction

Let's now dive into bigger experiments: **BERT** [1].

- BERT was an important milestone.
- Impressive performance that yields to the **massive adoption of transformers**.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Avg -
Pre-GPT SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
ELMO++	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{base}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\textbf{BERT}_{\textbf{large}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: Results on GLUE dataset, 9% of relative amelioration on average on an extremely competitive dataset.

$\Longrightarrow$ Let's go through the **technical details**.

BERT's pre-training

BERT's Masked Language Modeling (MLM)

Goal (Word2Vec++): predict the *word given the context*.

BERT's pre-training

BERT's Masked Language Modeling (MLM)

Goal (Word2Vec++): predict the *word given the context*.
$\Longrightarrow$ This is intuitively much harder than Word2Vec objective.

BERT's pre-training

BERT's Masked Language Modeling (MLM)

Goal (Word2Vec++): predict the *word given the context*.
$\Longrightarrow$ This is intuitively much harder than Word2Vec objective.

- Delete randomly 15% of tokens in x.
- Predict the deleted tokens.

BERT MLM illustration

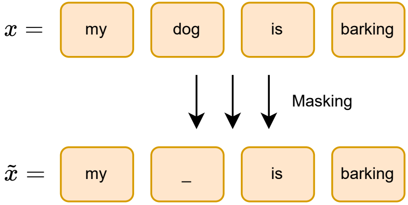

Figure 4: BERT masked language modeling.

Then we train the model to predict the right word:

$$
\begin{aligned}
\mathcal{L} &= -\log \mathrm{P}_\theta(\mathrm{dog} \mid \tilde{x}), \\
&= \sum_{\substack{w \in x \\ w \text{ is masked}}} -\log \mathrm{P}_\theta(w \mid \tilde{x})
\end{aligned}
$$

Other objectives?

That's it for the MLM pre-training objective.

Other objectives?

That's it for the MLM pre-training objective.

Why stop there? We can stack several objectives!

Other objectives?

That's it for the MLM pre-training objective.

Why stop there? We can stack several objectives!

Next sentence prediction.

Intuition: MLM operates at a token scale.
$\Longrightarrow$ Enhance the model's global understanding with **next sentence prediction**.

Next sentence prediction

- Extract a sentence s_1 from a document $x \in \mathcal{D}$.
- With 50% chance, take s_2 the sentence following s_1.
- With 50% chance, take s_2 a random sequence from $\mathcal{D}$.

Simply penalize the model:

$$\mathcal{L} = -\mathbf{1}_{s_2 \text{ follows } s_1} \log \mathrm{P}_\theta(s_1, s_2) - \mathbf{1}_{\text{random } s_2} \log(1 - \mathrm{P}_\theta(s_1, s_2)) \tag{10}$$

BERT in practice

Several questions arise in practice:

- How do we actually format the input?
- How do we indicate the model we deleted a word?
- How do we indicate a model what is the first sentence?

Let's visualize everything.

BERT – Input embeddings MLM

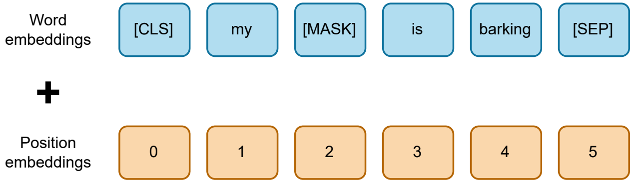

Figure 5: BERT input embeddings for MLM.

BERT uses **words and position embeddings**. There are **3 special tokens**.

- A special `[MASK]` token.
- A special token `[CLS]` that should retain sentence-level information.
- A special token indicating the end of the sequence `[SEP]`.

BERT – Next sentence prediction embeddings

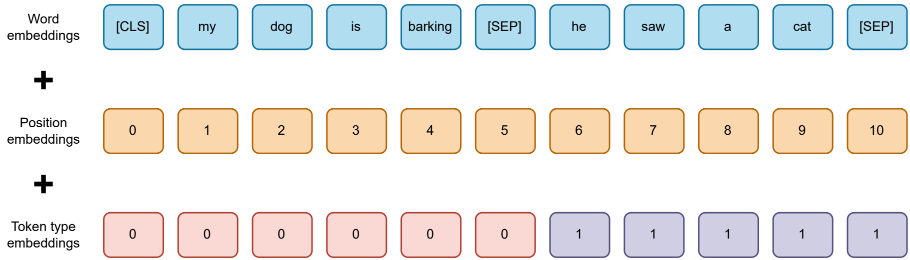

Figure 6: BERT input embeddings for next sentence prediction.

We use additional **token type** embeddings.

Online demo

Notebook.

Finetuning

Finetuning leverages internal representation as a backbone for classification.

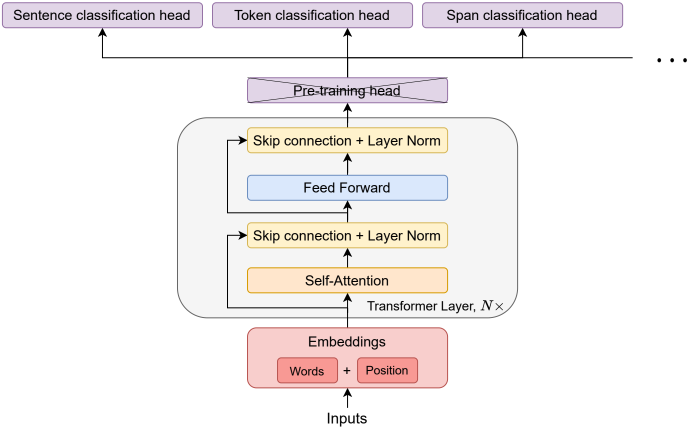

Figure 7: Switching from pretraining to finetuning.

Break!

Table of Contents

Reminders

Generative language models seek to maximize the log-likelihood over the dataset.

$$\max_{\theta} \sum_{x \in \mathcal{D}} \log \mathrm{P}_\theta(x).$$

But, since we are dealing with sequences of words, defining P_θ **should range over the whole set of sequences**, which is of cardinal $|\mathcal{V}|^L$.

When $\mathcal{V} \approx 30K$ and $L \approx 2K$, this is **higly unfeasible**.

Reminders

Generative language models seek to maximize the log-likelihood over the dataset.

$$\max_{\theta} \sum_{x \in \mathcal{D}} \log \mathrm{P}_{\theta}(x).$$

But, since we are dealing with sequences of words, defining P_{θ} **should range over the whole set of sequences**, which is of cardinal $|\mathcal{V}|^L$.

When $\mathcal{V} \approx 30K$ and $L \approx 2K$, this is **higly unfeasible**.

Instead, we choose to learn our probability distribution over the factorized form:

$$\mathrm{P}_{\theta}(x) = \prod_{i=1}^{L} \mathrm{P}_{\theta}(x_i \mid x_{<i}).$$

Generation

$$\log \mathrm{P}_\theta(x) = \sum_{i=1}^{L} \log \mathrm{P}_\theta(x_i \mid x_{<i}).$$

$\Longrightarrow$ The model learns to **predict the next tokens** given the previous ones.

This is clearly an **unsupervised** pre-training objective.

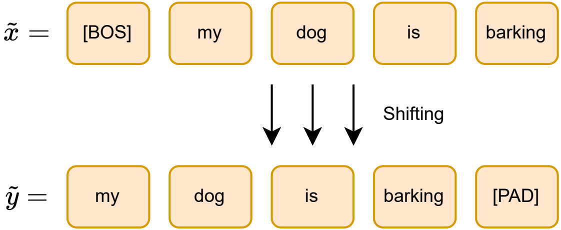

Figure 8: Generative models pre-training.

Architecture for generation

Can we still use the same bidirectional architecture?

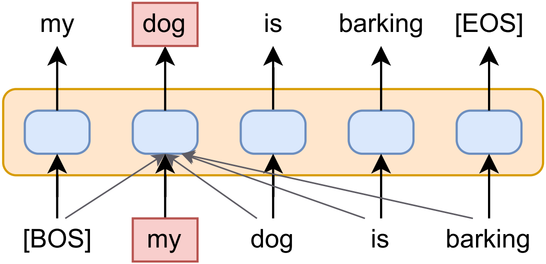

Figure 9: Bidirectional architecture.

Architecture for generation

Answer: No!

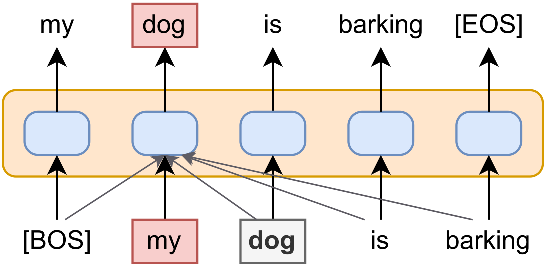

Figure 10: The transformer has access to dog to predict dog.

$\Longrightarrow$ We need another architecture.

Decoder

Attention in BERT

$$\begin{cases} s_{ij} &= \boldsymbol{q}_i^T \boldsymbol{k}_j \in \mathbb{R},\ 1 \leq j \leq \boldsymbol{L}, \\ \boldsymbol{\alpha}_i &= \text{Softmax}(\boldsymbol{s}_i) \in \mathbb{R}^{\boldsymbol{L}}, \\ \boldsymbol{y}_i &= \sum_{j=1}^{\boldsymbol{L}} \alpha_{ij} \boldsymbol{v}_j \in \mathbb{R}^d. \end{cases}$$

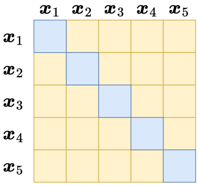

Figure 11: **Bidirectional** attention, tokens attend to every token.

Attention for generation

$$\begin{cases} s_{ij} &= \boldsymbol{q}_i^T \boldsymbol{k}_j \in \mathbb{R},\ 1 \leq j \leq \boldsymbol{i}, \\ \boldsymbol{\alpha}_i &= \text{Softmax}(\boldsymbol{s}_i) \in \mathbb{R}^{\boldsymbol{i}}, \\ \boldsymbol{y}_i &= \sum_{j=1}^{\boldsymbol{i}} \alpha_{ij} \boldsymbol{v}_j \in \mathbb{R}^d. \end{cases}$$

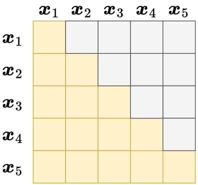

Figure 12: **Unidirectional** attention, tokens can only attend backward.

Decoder architecture

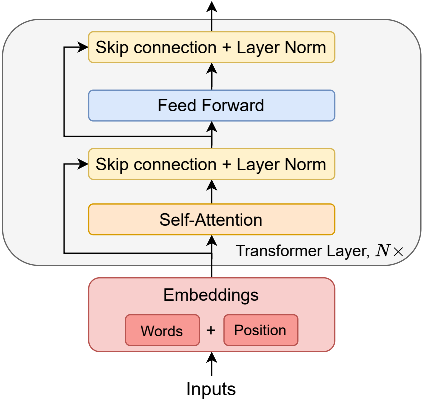

Figure 13: Decoder architecture.

Well-known decoder architecture

- GPT2 [3], GPT3 [4]
- LLama [9], LLama-2 [8], LLama-3, etc.
- etc.

$\Longrightarrow$ Demo!

Encoder-decoder

We saw bidirectional-**encoder** and causal **decoder**.

Encoder-decoder

We saw bidirectional-**encoder** and causal **decoder**.

Why don't use both?

Encoder-decoder models.

Encoder-decoder

We saw bidirectional-**encoder** and causal **decoder**.

Why don't use both?

Encoder-decoder models.

Encoder-decoder

Architectures tailored for conditional generation tasks.

- Give full access to x.
- Causal generation for y.

$$\mathrm{P}_\theta(y_i \mid y_{<i}, x).$$

Conditional generation

Conditional generation

Framing a problem as a condition generation task might be useful in a lot of tasks:

- translating text,
- summarizing a news article,
- answering a question over a text,
- etc.

Conditional generation

Conditional generation

Framing a problem as a condition generation task might be useful in a lot of tasks:

- translating text,
- summarizing a news article,
- answering a question over a text,
- etc.

$\implies$ It makes sense to give the model full access to a source x and generate the answer y based on this input x.

Encoder-decoder architecture

Encoder

x is encoded through a **bidirectional** transformer (BERT-like).

Decoder

y is processed through a **causal** transformer (GPT-like).

$\Longrightarrow$ How do the two communicate?

Encoder-decoder illustration

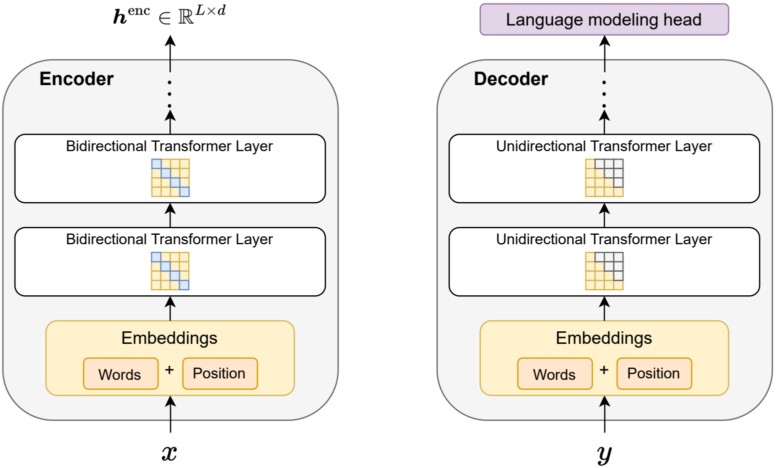

Figure 14: An encoder and a decoder.

Encoder-decoder illustration

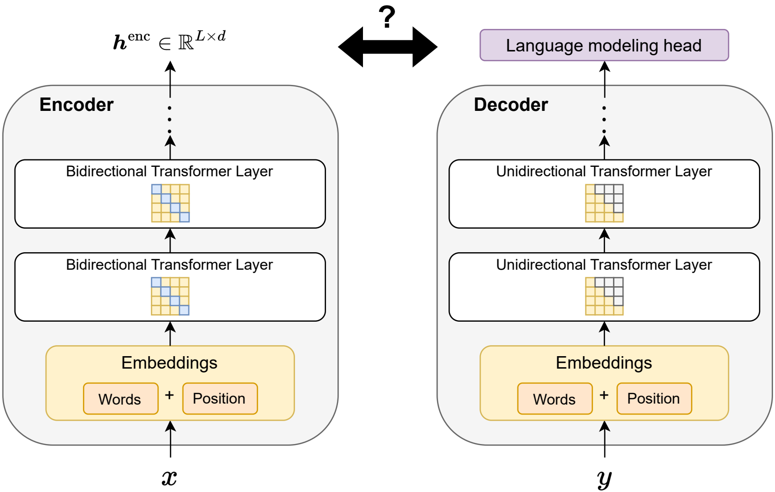

Figure 14: How can an encoder and a decoder communicate?

Cross-attention

Cross-attention: contextualization with the encoder output

Instead of computing the similarity of a token $\boldsymbol{x}_i$ with the other tokens $(\boldsymbol{x}_j)_{j \neq i}$, we compute the **similarity with the ouput of the encoder** ($\boldsymbol{z}$ is the decoder input):

$$\begin{cases} \boldsymbol{q}_i &= \boldsymbol{Q}\boldsymbol{z}_i, \\ \boldsymbol{v}_j &= \boldsymbol{V}\boldsymbol{h}_j^{\text{enc}}, \\ \boldsymbol{k}_j &= \boxed{\boldsymbol{K}\boldsymbol{h}_j^{\text{enc}}}, \end{cases} \qquad \begin{cases} s_{ij} &= \boldsymbol{q}_i^T \boldsymbol{k}_j \in \mathbb{R},\ 1 \leq j \leq L, \\ \boldsymbol{\alpha}_i &= \text{Softmax}(\boldsymbol{s}_i) \in \mathbb{R}^{\boldsymbol{i}}, \\ \boldsymbol{y}_i &= \sum_{j=1}^{L} \alpha_{ij} \boldsymbol{v}_j \in \mathbb{R}^d. \end{cases}$$

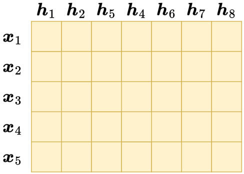

Figure 15: Tokens in the decoder attend to tokens from the encoder output.

Encoder-decoder with cross-attention

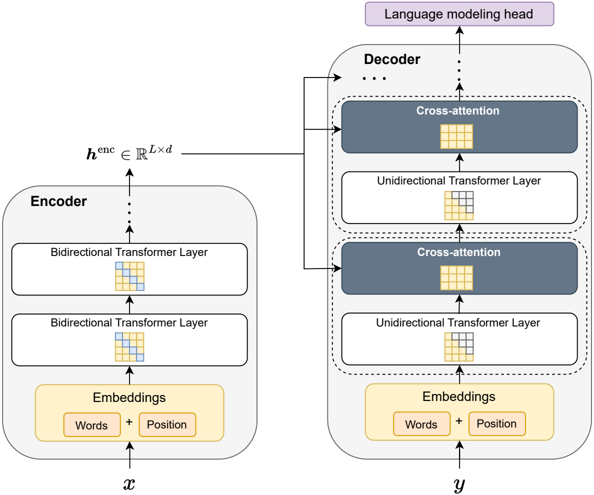

Figure 16: Encoder-decoder model with cross-attention layers.

Decoder layer for an encoder-decoder model

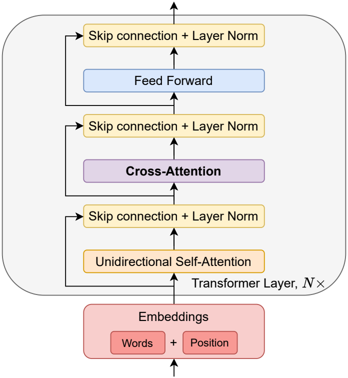

Figure 17: Decoder layer for an encoder-decoder model.

The decoder layers are extended with a **cross-attention** block. The encoder layers are not modified compared to encoder-only models (BERT).

Pre-training for encoder-decoders

Encoder-decoders have been used broadly in:

- Machine **translation** (state-of-the-art in the domain) [10],
- Text **summarization** (state-of-the-art also according to some evaluation) [2],
- Question answering [7],
- etc.

Except for machine translation, they relied on a pre-training.

BART

BART [2] is one of the most well-known encoder-decoder. For pre-training, the authors proposed a **denoising objective**.

BART's denoising objective

Several corruptions are made on the original text, and the goal is to retrieve the original one. It can be seen as a **generalization of BERT**:

- span masking,
- token deletion,
- sentence permutation.

BART on tokens infilling

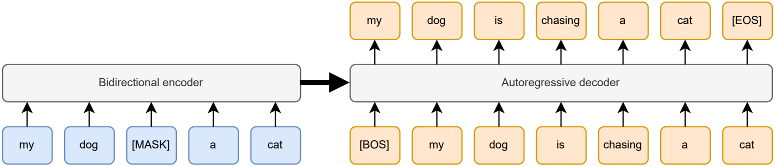

Figure 18: Denoising with BART.

Other models

Following BERT and BART, many papers proposed new pre-training objectives. To mention some of them:

- ELECTRA,
- ROBERTA,
- DEBERTA,
- T5,
- Pegasus,
- etc.

They all have their specificities but rely on the same ideas than BERT and BART.

Inference

Throughout the lessons we talked a lot about **training**.

But what about **inference**?

In the following part we are going to discuss the potential subtelties of inference in NLP.

Inference in classification

At train time

Classification models are trained with the MLE objective, i.e., they maximize $\mathrm{P}_\theta(y = \text{class} \mid x)$.

Inference in classification

At train time

Classification models are trained with the MLE objective, i.e., they maximize $\mathrm{P}_\theta(y = \text{class} \mid x)$.

At inference time

For an input x, the model gives a probability distribution over the classes $\mathrm{P}_\theta(\cdot \mid x)$.
Then you fix a decision rule, usually:

$$\hat{y} = \arg\max_y \mathrm{P}_\theta(y \mid x). \tag{11}$$

Inference in classification

At train time

Classification models are trained with the MLE objective, i.e., they maximize $\mathrm{P}_\theta(y = \text{class} \mid x)$.

At inference time

For an input x, the model gives a probability distribution over the classes $\mathrm{P}_\theta(\cdot \mid x)$.
Then you fix a decision rule, usually:

$$\hat{y} = \arg\max_y \mathrm{P}_\theta(y \mid x). \tag{11}$$

$\Longrightarrow$ For classification models, i.e., **encoders** in NLP, we do the same.

Inference in generation

At train time

For generative models, we maximize instead the factorized density: $\prod_{i=1}^{L} \mathrm{P}_\theta(y_i = w_i \mid w_{<i})$.

Inference in generation

At train time

For generative models, we maximize instead the factorized density: $\prod_{i=1}^{L} \mathrm{P}_\theta(y_i = w_i \mid w_{<i})$.

At inference time

Is the following decision rule still a good choice?

$$\hat{y} = \arg\max_{y} \mathrm{P}_\theta(y \mid x) = \arg\max_{y_1,\ldots,y_L} \prod_{i=1}^{L} \mathrm{P}_\theta(y_i \mid y_{<i}).$$

Inference in generation

At train time

For generative models, we maximize instead the factorized density: $\prod_{i=1}^{L} \mathrm{P}_\theta(y_i = w_i \mid w_{<i})$.

At inference time

Is the following decision rule still a good choice?

$$\hat{y} = \arg\max_{y} \mathrm{P}_\theta(y \mid x) = \arg\max_{y_1,\ldots,y_L} \prod_{i=1}^{L} \mathrm{P}_\theta(y_i \mid y_{<i}).$$

No! We are taking the arg max over $|\mathcal{V}|^L$ combinations, highly **intractable**.

Inference in generation

Workaround: approximate $\arg\max_{y_1,\ldots,y_L} \prod_{i=1}^{L} \mathrm{P}_\theta(y_i \mid y_{<i})$ with a **greedy algorithm**.

Simply:

- $\hat{y}_1 = \arg\max_{y_1} \mathrm{P}_\theta(y_1)$,
- $\hat{y}_2 = \arg\max_{y_2} \mathrm{P}_\theta(y_2 \mid \hat{y}_1)$,
- $\vdots$
- $\hat{y}_i = \arg\max_{y_i} \mathrm{P}_\theta(y_i \mid \hat{y}_{<i})$.

$\implies$ At each step, the $\arg\max$ is only performed over $|\mathcal{V}|$ possibilities.

Other inference methods

- **Beam search**: keep the k best sequences at each step.
- **Sampling**: sample from the distribution, using ancestral sampling. This can be parametrized with the temperature.
- **Top-k sampling**: sample from the k most probable tokens.
- **Top-p sampling**: sample from the smallest set of tokens whose cumulative probability exceeds a threshold p.

Other inference methods

- **Beam search**: keep the k best sequences at each step.
- **Sampling**: sample from the distribution, using ancestral sampling. This can be parametrized with the temperature.
- **Top-k sampling**: sample from the k most probable tokens.
- **Top-p sampling**: sample from the smallest set of tokens whose cumulative probability exceeds a threshold p.

Demo!

Table of Contents

Conclusion

Thank you!

References I

[1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[2] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].

[3] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[5] Leo Gao et al. *The Pile: An 800GB Dataset of Diverse Text for Language Modeling*. 2020. arXiv: 2101.00027 [cs.CL].

References II

[6] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. "ChatGPT outperforms crowd workers for text-annotation tasks". In: *Proceedings of the National Academy of Sciences* 120.30 (July 2023). DOI: 10.1073/pnas.2305016120. URL: https://doi.org/10.1073%2Fpnas.2305016120.

[7] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG].

[8] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[9] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[10] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

References III

[11] Common Crawl. *Common Crawl website*. URL: https://commoncrawl.org/.