Language Technology
http://cs.lth.se/edan20/
Chapter 12: Transformers

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

September 22, 2022

Transformers

After feedforward and recurrent networks, transformers are a third form of networks (in fact a kind of feedforward):

- An architecture proposed in 2018 based on the concept of **attention**
- Consists of a smart pipeline of matrices
- Transformers are trained with a **masked language model**
- They can learn complex lexical relations

Using Transformers

Goals of transformers:

- Encapsulate a massive amount of knowledge.
- In consequence trained on very large corpora
- Sometimes marketed as the ImageNet moment (See `https://ruder.io/nlp-imagenet/`)

Transformers in practice:

- Large companies train transformers on colossal corpora, the pretrained models, requiring huge computing resources (https://arxiv.org/pdf/1906.02243.pdf)
- Mere users:
  - Reuse the models in applications
  - Fine-tune some parameters in the downstream layers

The Concept of Attention

Reference paper: Attention Is All You Need by Vaswani et al (2017)
Link: `https://arxiv.org/pdf/1706.03762.pdf`
Implementation in PyTorch:
`https://nlp.seas.harvard.edu/2018/04/03/attention.html`

Contextual Embeddings

Embeddings we have seen so far do not take the context into account
Attention is a way to make them aware of the context.
Consider the sentence:

> *I must go back to my ship and to my crew*
> *Odyssey, book I*

The word *ship* can be a verb or a noun with different meanings, but has only one GloVe embedding vector
Self-attention will enable us to compute contextual word embeddings.

Self-Attention

In the paper *Attention is all you need*, Vaswani et al. (2017) use three kinds of vectors, queries, keys, and values. Here we will use one type corresponding to GloVe embeddings.
We compute the attention this way:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}})\mathbf{V},$$

where d_k is the dimension of the input. The softmax function is defined as:

$$\text{softmax}(x_1, x_2, \ldots, x_j, \ldots, x_n) = (\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}})$$

The meaning of $\mathbf{QK}^\intercal$

$\mathbf{QK}^\intercal$ is the dot product of the GloVe vectors. It will tell us the similarity between the words

This is analogous to cosine similarity:

	i	must	go	back	to	my	ship	and	to	my	crew
i	1.00	0.75	0.86	0.76	0.73	0.90	0.35	0.65	0.73	0.90	0.42
must	0.75	1.00	0.85	0.68	0.87	0.69	0.42	0.69	0.87	0.69	0.45
go	0.86	0.85	1.00	0.84	0.84	0.81	0.41	0.68	0.84	0.81	0.49
back	0.76	0.68	0.84	1.00	0.83	0.76	0.49	0.77	0.83	0.76	0.51
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
ship	0.35	0.42	0.41	0.49	0.54	0.38	1.00	0.46	0.54	0.38	0.78
and	0.65	0.69	0.68	0.77	0.86	0.63	0.46	1.00	0.86	0.63	0.49
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
crew	0.42	0.45	0.49	0.51	0.51	0.44	0.78	0.49	0.51	0.44	1.00

Vaswani's attention score

The attention scores are scaled and normalized by the softmax function.

$$\text{softmax}(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}),$$

	i	must	go	back	to	my	ship	and	to	my	crew
i	0.36	0.05	0.07	0.05	0.04	0.19	0.01	0.02	0.04	0.19	0.01
must	0.14	0.20	0.10	0.06	0.11	0.10	0.03	0.05	0.11	0.10	0.02
go	0.18	0.09	0.14	0.09	0.08	0.13	0.02	0.04	0.08	0.13	0.02
back	0.14	0.05	0.09	0.19	0.08	0.12	0.03	0.06	0.08	0.12	0.03
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
ship	0.03	0.03	0.03	0.04	0.05	0.03	0.55	0.03	0.05	0.03	0.13
and	0.10	0.08	0.07	0.10	0.12	0.09	0.04	0.15	0.12	0.09	0.04
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
crew	0.06	0.05	0.05	0.06	0.05	0.06	0.21	0.04	0.05	0.06	0.31

Attention

We use these scores to compute the attention.

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{Q}) = \text{softmax}(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}})\mathbf{V},$$

For *ship:*

```
attention_ship = (0.03 * embeddings_dict['i'] +
                  0.03 * embeddings_dict['must'] +
                  0.03 * embeddings_dict['go'] +
                  0.03 * embeddings_dict['back'] +
                  0.04 * embeddings_dict['to'] +
                  0.05 * embeddings_dict['my'] +
                  0.55 * embeddings_dict['ship'] +
                  0.03 * embeddings_dict['and'] +
                  0.05 * embeddings_dict['to'] +
                  0.03 * embeddings_dict['my'] +
                  0.13 * embeddings_dict['crew'])
```

where the *ship* vector received 13% of its value from *crew*

Code Example

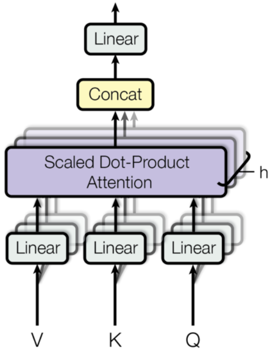
Experiment: Jupyter Notebook: `https://github.com/pnugues/edan95/blob/master/programs/4.4-attention.ipynb`
(First part of the notebook)

Multihead Attention

This attention is preceded by dense layers:
If $\mathbf{X}$ represents complete input sequence (all the tokens), we have:

$$\begin{aligned}\mathbf{Q} &= \mathbf{X}\mathbf{W}_Q,\\ \mathbf{K} &= \mathbf{X}\mathbf{W}_K,\\ \mathbf{V} &= \mathbf{X}\mathbf{W}_V.\end{aligned}$$

And followed by another dense layer.
In addition, most architectures have parallel attentions, where the outputs (called heads) are concatenated (multihead)



From *Attention is all you need*, Vaswani et al. (2017)

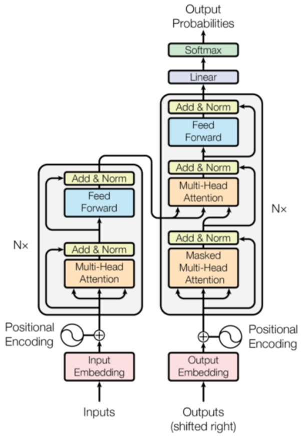
Code Example

Keras has an implementation of this architecture with the `MultiHeadAttention()` layer.
Experiment: Jupyter Notebook: `https://github.com/pnugues/edan95/blob/master/programs/4.4-attention.ipynb`
(Second part of the notebook)

Transformers

Transformers are architectures, where:

1. The first part of the layer is a multihead attention;
2. We reinject the input to the attention output in the form of an addition:
$$\mathbf{X} + \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{Q}).$$
This operation is called a skip or residual connection, which improves stability.
3. The result in then normalized per instance, i.e. a unique sequence, defined as:
$$x_{i,j_{norm}} = \frac{x_{i,j} - \bar{x}_{i,.}}{\sigma_{x_{i,.}}}.$$
4. It is followed by dense layers.



Left part, from *Attention is all you need*, Vaswani et al. (2017)

Training Transformers

Transformers, such as BERT, are often trained on masked language models with two tasks:

1. For a sentence, predict masked words: We replace 15% of the tokens with a specific mask token and we train the model to predict them. This is just a cloze test;
2. For a pair of sentences, predict if the second one is the successor of the first one;

Taking the two first sentences from the *Odyssey*:

> *Tell me, O Muse, of that ingenious hero who travelled far and wide after he had sacked the famous town of Troy.*
> *Many cities did he visit, and many were the nations with whose manners and customs he was acquainted;*

Masked language models

We add two special tokens: `[CLS]` at the start of the first sentence and `[SEP]` at the end of both sentences, and the token `[MASK]` to denote the words to predict.

We would have for the first task:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP]*

For the second task, we would have as input:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP] Many cities did he [MASK visit], and many were the [MASK nations] with whose manners [MASK and] customs he was acquainted; [SEP]*

where the prediction would return that the second sentence is the next one (as opposed to random sequences)

Positional Embeddings

BERT (the first transformer, Devlin et al. (2019)) maps each token to three embedding vectors:

- the token embedding,
- the position of the token in the sentence (positional embeddings), and
- the segment embeddings (we will skip this part).

The three kinds of embeddings are learnable vectors.
In the BERT base version, each embedding vector has 768 dimensions.
Let us consider two sentences simplified from the *Odyssey:*

Tell me of that hero. Many cities did he visit.

Input:	[CLS]	Tell	me	of	that	hero	[SEP]	Many	cities	did	he	visit	[SEP]
Token	$\mathbf{E}_{\texttt{[CLS]}}$	$\mathbf{E}_{\text{tell}}$	$\mathbf{E}_{\text{me}}$	$\mathbf{E}_{\text{of}}$	$\mathbf{E}_{\text{that}}$	$\mathbf{E}_{\text{hero}}$	$\mathbf{E}_{\texttt{[SEP]}}$	$\mathbf{E}_{\text{many}}$	$\mathbf{E}_{\text{cities}}$	$\mathbf{E}_{\text{did}}$	$\mathbf{E}_{\text{he}}$	$\mathbf{E}_{\text{visit}}$	$\mathbf{E}_{\texttt{[SEP]}}$
Segment	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$
Position	$\mathbf{E}_0$	$\mathbf{E}_1$	$\mathbf{E}_2$	$\mathbf{E}_3$	$\mathbf{E}_4$	$\mathbf{E}_5$	$\mathbf{E}_6$	$\mathbf{E}_7$	$\mathbf{E}_8$	$\mathbf{E}_9$	$\mathbf{E}_{10}$	$\mathbf{E}_{11}$	$\mathbf{E}_{12}$

Model size

Transformers are trained on large corpora like the colossal clean crawled corpus (https://arxiv.org/abs/2104.08758) and encapsulate semantics found in text in the form of numerical matrices.
This results in large models (Devlin et al., 2019):

> *In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. We primarily report results on two model sizes: BERT$_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M) and BERT$_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M).*

Transformers can then act as pre-trained models for a variety of tasks.
See the list from Huggingface
Finally, an interesting reading: https://sayakpaul.medium.com/an-interview-with-colin-raffel-research-scientist-at-google-5

Code Example

Experiment: Jupyter Notebook: `https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part03_transformer.ipynb` from Chollet's book, first part up to positional embeddings

Code Example

Experiment: Jupyter Notebook: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter11_part03_transformer.ipynb from Chollet's book, second part from positional embeddings

As a personal work and to gain a deeper understanding, you can read a tutorial in PyTorch:
https://nlp.seas.harvard.edu/2018/04/03/attention.html