Deep learning

13.3. Transformer Networks

François Fleuret
https://fleuret.org/dlc/
Vaswani et al. (2017) proposed to go one step further: instead of using attention mechanisms as a supplement to standard convolutional and recurrent operations, they designed a model combining only attention layers. They designed this “transformer” for a sequence-to-sequence translation task, but it is currently key to state-of-the-art approaches across NLP tasks.

Notes
The standard practice is to train a transformer in a non-supervised manner on large unlabeled datasets such as Wikipedia—or re-use a pre-trained transformer—and then fine tune it in a supervised manner for tasks which require a ground truth such as sentiment analysis.
They first introduce a multi-head attention module.

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, \ldots, H_h) W^O$$

$$H_i = \text{Attention} \left( QW^O_i, KW^K_i, VW^K_i \right), \ i = 1, \ldots, h$$

with

$$W^O_i \in \mathbb{R}^{d_{\text{model}} \times d_k}, \ W^K_i \in \mathbb{R}^{d_{\text{model}} \times d_k}, \ W^K_i \in \mathbb{R}^{d_{\text{model}} \times d_v}, \ W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

Notes

The “scaled dot-product attention” (left) is very close to the attention module we saw in lecture 13.2. “Attention Mechanisms”, with the addition of an optional masking (in pink). This may be useful when such a module is used for a generative auto-regressive operation and the attention should be causal, looking only to the past.

The attention is a function of the keys, queries, and values. The only difference with what was seen in the previous course is that the attention matrix is rescaled with the dimension of the embedding, which matters quite a lot.

In the multi-head attention, each head $h$ has its own processing of the input keys, queries, and values through respectively $W^K_i$, $W^O_i$, and $W^K_i$. And there is one final processing $W^O$ applied on the concatenated results of the multiple heads.
Their complete model is composed of:

- An encoder that combines $N = 6$ modules each composed of a multi-head attention sub-module, and a [per-component] one hidden-layer MLP, with residual pass-through and layer normalization.
- A decoder with a similar structure, but with causal attention layers to allow for regression training, and additional attention layers that attend to the layers of the encoder.

Positional information is provided through an additive positional encoding of same dimension $d_{model}$ as the internal representation, and is of the form

$$PE_{t, 2i} = \sin \left( \frac{t}{10,000^{2i/d_{model}}} \right)$$

$$PE_{t, 2i+1} = \cos \left( \frac{t}{10,000^{2i+1/d_{model}}} \right).$$

---

**Notes**

Contrary to what we previously saw with the concatenated binary positional encoding, here the position is provided as additive encoding, where $t$ is the position in the sequence, and $2i$ and $2i + 1$ the dimension.
3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of \( N = 6 \) identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection \([11]\) around each of the two sub-layers, followed by layer normalization \([1] \). That is, the output of each sub-layer is \( \text{LayerNorm}(x + \text{Sublayer}(x)) \), where \( \text{Sublayer}(x) \) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension \( d_{\text{model}} = 512 \).

Decoder: The decoder is also composed of a stack of \( N = 6 \) identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position \( i \) can depend only on the known outputs at positions less than \( i \).

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Notes

This is a depiction of the standard transformer architecture for sequence-to-sequence translation. It consists of an encoder (left part) and a decoder (right part). Both are a stack of \( N = 6 \) modules.

Each token (subword) of the input sequence is encoded with a look-up table to get its embedding of dimension \( d \), so that the input is a tensor of size \( T \times d \). Then the positional encoding of same size is added to it.

Each of the \( N \) modules of the encoder is composed of a multi-head self-attention operation followed by a “feed forward” operation that applies a one hidden layer perceptron at every position of the sequence separately. This can be implemented with \( 1 \times 1 \) convolutions. Both the self-attention and the feed-forward are combined with residual pass-through.

The decoder is an auto-regressive model, and each of its module has a multi-head self-attention operation, then an attention that attends to the encoder, and a feed-forward operation. The self-attention is masked to make it causal, i.e. it takes into account only the part of the sequence already generated. The attention to the encoder is not masked but its keys and values are functions of the outputs of the corresponding module in the encoding stack.
The architecture is tested on English-to-German and English-to-French translation using the standard WMT2014 datasets.

- English-to-German: 4.5M sentence pairs, 37k tokens vocabulary.
- English-to-French: 36M sentence pairs, 32k tokens vocabulary.
- 8 P100 GPUs (150 TFlops FP16), 0.5 day for the small model, 3.5 days for the large one.
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-DE</td>
<td>EN-FR</td>
</tr>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
<td>1.0 \cdot 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td>39.2</td>
<td>1.4 \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
<td>9.6 \cdot 10^{18}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
<td>1.5 \cdot 10^{20}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>2.0 \cdot 10^{19}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>40.4</td>
<td>8.0 \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>1.1 \cdot 10^{21}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>1.2 \cdot 10^{21}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>37.3</td>
<td>3.3 \cdot 10^{18}</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>2.3 \cdot 10^{19}</td>
</tr>
</tbody>
</table>

(Vaswani et al., 2017)

Notes

The standard metric in natural language processing is the Bilingual Evaluation Understudy Score (BLEU) score which aims at evaluating a generated sequence to a reference sentence. The BLEU score ranges between 0 (perfect mismatch) and 1 (perfect match).
The Law will never be perfect, but its application should be just — this is what we are missing, in my opinion.

Notes
On the left is a visualization of the attention as computed by one head of the layer 5 of the encoder.

On the right the attention given by the word "its" for two different heads is on "law" and "application" which provides help for gender and grammatical issues.

(Vaswani et al., 2017)
The Law will never be perfect, but its application should be just—this is what we are missing, in my opinion.

(Vaswani et al., 2017)

Notes

Two other heads also in layer 5.
The Universal Transformer (Dehghani et al., 2018) is a similar model where all the blocks are identical, resulting in a recurrent model that iterates over consecutive revisions of the representation instead of positions.

Additionally the number of steps is modulated per position dynamically.
Transformer self-training and fine-tuning for NLP
The transformer networks were introduced for translation, and trained with a supervised procedure, from pairs of sentences.

However, as for word embeddings, they can be trained in an unsupervised manner, for auto-regression or as denoising auto-encoders, from very large data-sets, and fine-tuned on supervised tasks with small data-sets.

---

**Notes**

A transformer [pre-]trained in an unsupervised manner for the task of predicting a token: for auto-regression, the input is the sentence up to the token to predict, for mask language modeling, the input is a full sentence with some tokens replaced by a "mask" token. No ground truth is required for those tasks.

As for word embedding, training a transformer model like this allows to capture statistical structures in the text and provide an extremely good representation for more sophisticated tasks which can only be trained in a supervised manner with only small datasets available.
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

(Devlin et al., 2018)
GPT (Generative Pre-Training, Radford, 2018) is a transformer trained for auto-regressive text generation.

3.3 Task-specific input transformations

For some tasks, like text classification, we can directly fine-tune our model as described above. Certain other tasks, like question answering or textual entailment, have structured inputs such as ordered sentence pairs, or triplets of document, question, and answers. Since our pre-trained model was trained on contiguous sequences of text, we require some modifications to apply it to these tasks.

Previous work proposed learning task specific architectures on top of transferred representations [44]. Such an approach re-introduces a significant amount of task-specific customization and does not use transfer learning for these additional architectural components. Instead, we use a traversal-style approach [52], where we convert structured inputs into an ordered sequence that our pre-trained model can process. These input transformations allow us to avoid making extensive changes to the architecture across tasks. We provide a brief description of these input transformations below and Figure 1 provides a visual illustration. All transformations include adding randomly initialized start and end tokens (⟨⟩).

Textual entailment

For entailment tasks, we concatenate the premise \( p \) and hypothesis \( h \) token sequences, with a delimiter token (\$) in between.

Similarity

For similarity tasks, there is no inherent ordering of the two sentences being compared. To reflect this, we modify the input sequence to contain both possible sentence orderings (with a delimiter in between) and process each independently to produce two sequence representations which are added element-wise before being fed into the linear output layer.

Question Answering and Commonsense Reasoning

For these tasks, we are given a context document \( z \), a question \( q \), and a set of possible answers \( \{ a_k \} \). We concatenate the document context and question with each possible answer, adding a delimiter token in between to get \[ z; q; \$; a_k \]. Each of these sequences are processed independently with our model and then normalized via a softmax layer to produce an output distribution over possible answers.

Notes

Note that GPT model is inherently causal, so only carries information forward, and consists of 12 modules as opposed to 6 for the original transformer.

The tasks GPT can be fine-tuned on are:

- Classification: for instance for sentiment analysis, when the input is a comment, and the task is to predict whether it is positive or negative.
- Entailment: given a premise and a hypothesis, the task is to predict whether the hypothesis is implied by the premise.
- Similarity: the task is to predict if two pieces of text have the same meaning.
- Multiple choice: the task is to predict the correct answer.
“GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data.”

(Radford et al., 2019)

The GPT-3 model has 175B parameters and was trained on 300B tokens from various sources (Brown et al., 2020).
We can use HuggingFace’s pre-trained models (https://huggingface.co/).

```python
import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
model.eval()
tokens = tokenizer.encode('Studying Deep-Learning is')

for k in range(100):  # no more than 100 tokens
    outputs = model(torch.tensor([tokens])).logits
    next_token = torch.argmax(outputs[0, -1])
    print(k, next_token)
    tokens.append(next_token)
    if tokenizer.decode([next_token]) == '.': break

print(tokenizer.decode(tokens))

prints

Studying Deep-Learning is a great way to learn about the world around you.
```

---

**Notes**

HuggingFace is a company that develops and distributes open-source implementations of language models. The transformers can be installed using the pip Python packets management system with

```
pip install transformers
```

The piece of code in the slide loads a GPT-2 model and generates the end of a sentence given “Studying Deep-Learning is” as beginning.

`tokenizer` can take as input a sequence of strings to produce a sequence of token, or the opposite, take as input a sequence of tokens and produces the corresponding sequence of words.

In this example, the generative procedure picks at each iteration the word with the maximum probability, but stochastic sampling could also be used.

The generated sentence could make sense in other situations (e.g. replacing “Deep-Learning” with another topic of studies), but is grammatically correct and consistent with “studying”.

Large models of this class have been shown to exhibit some “zero shot learning” capabilities when they are properly “primed” (Brown et al., 2020).

For instance using HuggingFace’s gpt2-xl model with 1.6B parameters, we can get these sentence completions, where the priming text is between <>:

<Cherry is red, lettuce is green, lemon is> yellow, and orange is blue.
<Cherry is sweet, lettuce is bland, lemon is> sour, and orange is bitter.
<Cherry is a fruit, lettuce is a vegetable, lemon is> a fruit, and so on.
BERT (Bidirectional Encoder Representation from Transformers, Devlin et al., 2018) is a transformer pre-trained with:

- Masked Language Model (MLM), that consists in predicting [15% of] words which have been replaced with a “MASK” token.
- Next Sentence Prediction (NSP), which consists in predicting if a certain sentence follows the current one.

It is then fine-tuned on multiple NLP tasks.
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

(Devlin et al., 2018)

Notes

BERT is trained in an unsupervised manner to do two tasks at the same time: predicting missing words and predicting whether two sentences follow each other in the corpus.
Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

(Devlin et al., 2018)
Notes

Once again, visualizing the attention matrices show that it connects a word with other words that help to get its meaning.

(Clarke et al., 2019)
Attention in computer vision
Wang et al. (2018) proposed an attention mechanism for images, following the model from Vaswani et al. (2017).

\[ y = \text{softmax} \left( (W_\theta x)^\top (W_\Phi x) \right) W_g x. \]
They insert “non-local blocks” in residual architectures and get improvements on both video and images classification.

Figure 2. A spacetime non-local block. The feature maps are shown as the shape of their tensors, e.g., $T \times H \times W \times 1024$ for 1024 channels (proper reshaping is performed when noted). “⊙” denotes matrix multiplication, and “⊕” denotes element-wise sum. The softmax operation is performed on each row. The blue boxes denote $1 \times 1 \times 1$ convolutions. Here we show the embedded Gaussian version, with a bottleneck of 512 channels. The vanilla Gaussian version can be done by removing $\theta$ and $\phi$, and the dot-product version can be done by replacing softmax with scaling by $1/N$.

(Wang et al., 2018)

Notes

The input is a video sequence of $T$ frames of size $H \times W$. Each pixel location is represented by a feature vector of dimension 1024, and the input is of size $T \times H \times W \times 1024$.

The query, key, and value tensors are obtained by one-by-one convolution from the input. And a final one-by-one convolution project the output of the attention mechanism to the original input size.

Similarly to the transformer for natural language understanding, there is a residual pass-through.
Figure 3. Examples of the behavior of a non-local block in res3 computed by a 5-block non-local model trained on Kinetics. These examples are from held-out validation videos. The starting point of arrows represents one $x_i$, and the ending points represent $x_j$. The 20 highest weighted arrows for each $x_i$ are visualized. The 4 frames are from a 32-frame input, shown with a stride of 8 frames. These visualizations show how the model finds related clues to support its prediction.

(Wang et al., 2018)
Ramachandran et al. (2019) replaced convolutions with local attention.

\[
y_{i,j} = \sum_{(a,b) \in \mathcal{V}(i,j)} W_{i-a,j-b} x_{a,b} \quad \text{(Convolution)}
\]

\[
y_{i,j} = \sum_{(a,b) \in \mathcal{V}(i,j)} \text{softmax}_{a,b} \left( (W_Q x_{i,j})^\top (W_K x_{a,b}) \right) v_{a,b} \quad \text{(Local attention)}
\]

Figure 2: An example of a 3 x 3 convolution. The output is the inner product between the local window and the learned weights.

Figure 3: An example of a local attention layer over spatial extent of \( k = 3 \).
Table 1: ImageNet classification results for a ResNet network with different depths. Baseline is a standard ResNet, Conv-stem + Attention uses spatial convolution in the stem and attention everywhere else, and Full Attention uses attention everywhere including the stem. The attention models outperform the baseline across all depths while having 12% fewer FLOPS and 29% fewer parameters.

Figure 5: Comparing parameters and FLOPS against accuracy on ImageNet classification across a range of network widths for ResNet-50. Attention models have fewer parameters and FLOPS while improving upon the accuracy of the baseline.

(Ramachandran et al., 2019)
“A fully attentional network based off of the proposed stand-alone local self-attention layer achieves competitive predictive performance on ImageNet classification and COCO object detection tasks while requiring fewer parameters and floating point operations than the corresponding convolution baselines.”

(Ramachandran et al., 2019)
Cordonnier et al. (2020) showed that provided with proper positional encoding multi-head multiplicative attention layers can encode convolutions with filter support of size the number of heads:

“A multi head self-attention layer with $N_h$ heads of dimension $D_h$, output dimension $D_{out}$ and a relative positional encoding of dimension $D_p \geq 3$ can express any convolutional layer of kernel size $\sqrt{N_h} \times \sqrt{N_h}$ and $\min(D_h, D_{out})$ output channels.”

(Cordonnier et al., 2020)
Figure 5: Attention probabilities of each head (column) at each layer (row) using learned relative positional encoding without content-based attention. The central black square is the query pixel. We reordered the heads for visualization and zoomed on the 7x7 pixels around the query pixel.

Figure 6: Attention probabilities for a model with 6 layers (rows) and 9 heads (columns) using learned relative positional encoding and content-content based attention. Attention maps are averaged over 100 test images to display head behavior and remove the dependence on the input content. The black square is the query pixel. More examples are presented in Appendix A.

(Cordonnier et al., 2020)
References


