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 composed of attention layers only.

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.

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 $W_i^K$, $W_i^Q$, and $W_i^V$.

And there is one final processing $W_O$ applied on the concatenated results of the multiple heads.

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{Q K^\top}{\sqrt{d_k}} \right) V
\]

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

with

\[
W_i^Q \in \mathbb{R}^{d_{\text{mod}} \times d_k}, \quad W_i^K \in \mathbb{R}^{d_{\text{mod}} \times d_k}, \quad W_i^V \in \mathbb{R}^{d_{\text{mod}} \times d_v}, \quad W^O \in \mathbb{R}^{hd_v \times d_{\text{mod}}}
\]
Their complete Transformer model is composed of:

- An encoder that combines $N = 6$ modules, each composed of a multi-head attention sub-module, and a [per-token] 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 encoder final keys and values.

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

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

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

---

**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 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.
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).

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>EN-DE BLEU</th>
<th>EN-FR BLEU</th>
<th>EN-DE Training Cost (FLOPs)</th>
<th>EN-FR Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
<td>39.2</td>
<td>1.0 \cdot 10^{20}</td>
<td></td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td>24.6</td>
<td>39.92</td>
<td>2.3 \cdot 10^{19}</td>
<td>1.4 \cdot 10^{20}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>40.56</td>
<td>2.0 \cdot 10^{19}</td>
<td>1.2 \cdot 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>26.30</td>
<td>40.4</td>
<td>8.0 \cdot 10^{20}</td>
<td></td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.36</td>
<td>41.16</td>
<td>1.8 \cdot 10^{20}</td>
<td>1.1 \cdot 10^{21}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>41.29</td>
<td>7.7 \cdot 10^{19}</td>
<td>1.2 \cdot 10^{21}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
<td>3.3 \cdot 10^{18}</td>
<td></td>
</tr>
<tr>
<td>Transformer (big)</td>
<td><strong>28.4</strong></td>
<td><strong>41.8</strong></td>
<td><strong>2.3 \cdot 10^{19}</strong></td>
<td></td>
</tr>
</tbody>
</table>

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

(Vaswani et al., 2017)

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.
The Law will never be perfect, but its application should be just—this is what we are missing, in my opinion.

Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

(Vaswani et al., 2017)

Notes

Two other heads also in layer 5.
Standard transformers now combine differently the residual connection and the normalization (Wang et al., 2019).

![Figure 1: Examples of pre-norm residual unit and post-norm residual unit](image)

Figure 1: Examples of pre-norm residual unit and post-norm residual unit. $F = \text{sub-layer}$, and $LN = \text{layer normalization}$.

(Wang et al., 2019)
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 a 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.
BERT (Bidirectional Encoder Representation from Transformers, Devlin et al., 2018) is an encoder of 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.
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 \[ \text{LayerNorm}(x + \text{Sublayer}(x)) \] around each of the two sub-layers, followed by layer normalization. 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.
We note that in the literature the bidirectional Transformer architecture has been widely used for natural language inference (Conneau et al., 2018; Radford et al., 2018; Dai and Le, 2015). There has also been work showing effective transfer learning from supervised tasks with large datasets, such as computer vision research (Deng et al., 2009; Yosinski et al., 2014). As a running example for this section, we will omit an exhaustive background description of Transformers and refer readers to Vaswani et al. (2017) as well as excellent guides such as “The Annotated Transformer.”

In this work, we denote the number of layers as $L$, the hidden size as $H$, and the number of self-attention heads as $A$. For BERT BASE and BERT LARGE, we take $L=12$, $H=768$, and $A=12$, with total parameters $340$ million. For BERT XXL, we take $L=24$, $H=1024$, and $A=20$, with total parameters $1103$ million. Table 1 contains the models we consider and their specifications. Digits in parentheses indicate the model size as OpenAI GPT for comparison purposes.

The objective of the pre-training procedure is to fine-tune models pre-trained with ImageNet (Deng et al., 2009; Yosinski et al., 2014). A distinctive feature of BERT is its unified architecture across different tasks. There is mini-batch training with unlabeled data over different pre-training tasks. For fine-tuning, the model is trained on unlabeled data over the same tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The features are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize the downstream tasks. During fine-tuning, all parameters are fine-tuned.

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 downstream 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.
The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment (Socher et al., 2013).

The Corpus of Linguistic Acceptability is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically “acceptable” or not (Warstadt et al., 2018).

The Semantic Textual Similarity Benchmark is a collection of sentence pairs drawn from news headlines and other sources (Cer et al., 2017). They were annotated with a score from 1 to 5 denoting how similar the two sentences are in terms of semantic meaning.

Microsoft Research Paraphrase Corpus consists of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent (Dolan and Brockett, 2005).

Recognizing Textual Entailment is a binary entailment task similar to MNLI, but with much less training data (Bentivogli et al., 2009).

Winograd NLI is a small natural language inference dataset (Levesque et al., 2011). The GLUE webpage notes that there are issues with the construction of this dataset, and every trained system that’s been submitted to GLUE has performed worse than the 65.1 baseline accuracy of predicting the majority class. We therefore exclude this set to be fair to OpenAI GPT. For our GLUE submission, we always predicted the majority class.

Note that we only report single-task fine-tuning results in this paper. A multitask fine-tuning approach could potentially push the performance even further. For example, we did observe substantial improvements on RTE from multi-task training with MNLI.

https://gluebenchmark.com/faq

(François Fleuret)
**Notes**

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

*(Clark et al., 2019)*
Large Language Models
GPT (Generative Pre-Training, Radford, 2018) is a decoder of 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 (⟨s⟩, ⟨e⟩).

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.

4 Experiments

4.1 Setup

Unsupervised pre-training

We use the BooksCorpus dataset [71] for training the language model. It contains over 7,000 unique unpublished books from a variety of genres including Adventure, Fantasy, and Romance. Crucially, it contains long stretches of contiguous text, which allows the generative model to learn to condition on long-range information. An alternative dataset, the 1B Word Benchmark, which is used by a similar approach, ELMo [44], is approximately the same size.
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 \[\text{LayerNorm}(x + \text{Sublayer}(x))\] around each of the two sub-layers, followed by layer normalization. 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 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.
“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)
We can use HuggingFace's pre-trained models ([https://huggingface.co/](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])
    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 packages 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 GPT have been shown to exhibit some “few shot learning” capabilities when they are properly “primed” (Brown et al., 2020).

For instance using Hugging Face’s gpt2 model with 120M parameters, we can get these sentence completions, where the generated parts are in bold:

| I: I love apples, O: positive, I: music is my passion, O: positive, I: my job is boring, O: negative, I: frozen pizzas are awesome, O: **positive**, |
| I: I love apples, O: positive, I: music is my passion, O: positive, I: my job is boring, O: negative, I: frozen pizzas taste like cardboard, O: **negative**, |
| I: water boils at 100 degrees, O: physics, I: the square root of two is irrational, O: mathematics, I: the set of prime numbers is infinite, O: mathematics, I: gravity is proportional to the mass, O: **physics**, |
| I: water boils at 100 degrees, O: physics, I: the square root of two is irrational, O: mathematics, I: the set of prime numbers is infinite, O: mathematics, I: squares are rectangles, O: **mathematics**, |
Figure 3.1: Smooth scaling of performance with compute. Performance (measured in terms of cross-entropy validation loss) follows a power-law trend with the amount of compute used for training. The power-law behavior observed in [KMH+20] continues for an additional two orders of magnitude with only small deviations from the predicted curve. For this figure, we exclude embedding parameters from compute and parameter counts.

Table 3.1: Zero-shot results on PTB language modeling dataset. Many other common language modeling datasets are omitted because they are derived from Wikipedia or other sources which are included in GPT-3’s training data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Zero-Shot</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTB</td>
<td>GPT-3</td>
<td>20.50</td>
</tr>
<tr>
<td>LAMBADA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.1 Language Modeling

We calculate zero-shot perplexity on the Penn Tree Bank (PTB) [MKM94] dataset measured in [RWC19]. We omit the 4 Wikipedia-related tasks in that work because they are entirely contained in our training data, and we also omit the one-billion word benchmark due to a high fraction of the dataset being contained in our training set. PTB escapes these issues due to predating the modern internet. Our largest model sets a new SOTA on PTB by a substantial margin of 15 points, achieving a perplexity of 20.50. Note that since PTB is a traditional language modeling dataset it does not have a clear separation of examples to define one-shot or few-shot evaluation around, so we measure only zero-shot.

3.1.2 LAMBADA

The LAMBADA dataset [PKL16] tests the modeling of long-range dependencies in text – the model is asked to predict the last word of sentences which require reading a paragraph of context. It has recently been suggested that the continued scaling of language models is yielding diminishing returns on this difficult benchmark. [BHT20] reflect on the small 1.5% improvement achieved by a doubling of model size between two recent state of the art results ([SPP+19]).

The GPT-3 model has 175B parameters and is trained on 300B tokens from various sources (Brown et al., 2020). The Pathways Language Model (PaLM) has 540B parameters and is trained on 780B tokens (Chowdhery et al., 2022).
<table>
<thead>
<tr>
<th>Setting</th>
<th>2D+</th>
<th>2D-</th>
<th>3D+</th>
<th>3D-</th>
<th>4D+</th>
<th>4D-</th>
<th>5D+</th>
<th>5D-</th>
<th>2Dx</th>
<th>1DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3 Zero-shot</td>
<td>76.9</td>
<td>58.0</td>
<td>34.2</td>
<td>48.3</td>
<td>4.0</td>
<td>7.5</td>
<td>0.7</td>
<td>0.8</td>
<td>19.8</td>
<td>9.8</td>
</tr>
<tr>
<td>GPT-3 One-shot</td>
<td>99.6</td>
<td>86.4</td>
<td>65.5</td>
<td>78.7</td>
<td>14.0</td>
<td>14.0</td>
<td>3.5</td>
<td>3.8</td>
<td>27.4</td>
<td>14.3</td>
</tr>
<tr>
<td>GPT-3 Few-shot</td>
<td>100.0</td>
<td>98.9</td>
<td>80.4</td>
<td>94.2</td>
<td>25.5</td>
<td>26.8</td>
<td>9.3</td>
<td>9.9</td>
<td>29.2</td>
<td>21.3</td>
</tr>
</tbody>
</table>

Figure G.44: Evaluation example for Arithmetic 2D+

Figure G.45: Evaluation example for Arithmetic 2Dx

Figure G.46: Evaluation example for Arithmetic 3D-

Figure 3.10: Results on all 10 arithmetic tasks in the few-shot settings for models of different sizes. There is a significant jump from the second largest model (GPT-3 13B) to the largest model (GPT-3 175), with the latter being able to reliably accurate 2 digit arithmetic, usually accurate 3 digit arithmetic, and correct answers a significant fraction of the time on 4-5 digit arithmetic, 2 digit multiplication, and compound operations. Results for one-shot and zero-shot are shown in the appendix.

Table 3.9: Results on basic arithmetic tasks for GPT-3 175B. Setting 2D+ 2D- 3D+ 3D- 4D+ 4D- 5D+ 5D- 2Dx 1DC

First we evaluate GPT-3 in the few-shot setting, for which results are shown in Figure 3.10. On addition and subtraction, GPT-3 displays strong proficiency when the number of digits is small, achieving 100% accuracy on 2 digit addition, 98.9% at 2 digit subtraction, 80.2% at 3 digit addition, and 94.2% at 3-digit subtraction. Performance decreases as the number of digits increases, but GPT-3 still achieves 25-26% accuracy on four digit operations and 9-10% accuracy on five digit operations, suggesting at least some capacity to generalize to larger numbers of digits. GPT-3 also achieves 29.2% accuracy at 2 digit multiplication, an especially computationally intensive operation. Finally, GPT-3 achieves 21.3% accuracy at single digit combined operations (for example, 9*(7+5)), suggesting that it has some robustness beyond just single operations.

As Figure 3.10 makes clear, small models do poorly on all of these tasks – even the 13 billion parameter model (the second largest after the 175 billion full GPT-3) can solve 2 digit addition and subtraction only half the time, and all other operations less than 10% of the time.

One-shot and zero-shot performance are somewhat degraded relative to few-shot performance, suggesting that adaptation to the task (or at the very least recognition of the task) is important to performing these computations correctly. Nevertheless, one-shot performance is still quite strong, and even zero-shot performance of the full GPT-3 significantly outperforms few-shot learning for all smaller models. All three settings for the full GPT-3 are shown in Table 3.9, and model capacity scaling for all three settings is shown in Appendix H.

To spot-check whether the model is simply memorizing specific arithmetic problems, we took the 3-digit arithmetic problems in our test set and searched for them in our training data in both the forms "<NUM1> + <NUM2> =" and "<NUM1> * <NUM2> =". There were no such problems in the training data.

(Brown et al., 2020)
```python
X = torch.randn(1, 3, 768, 768)
# Print all values of X higher than its median
print(X[X > X.median()])

class Downsample(nn.Module):
    def __init__(self, factor, channel_out, drop_prob):
        super().__init__()
        self.downsample_conv = nn.Conv2d(64, 64, kernel_size=factor, stride=factor, groups=64)
        self.enlarge_conv = nn.Sequential(
            nn.Dropout2d(drop_prob),
            nn.Conv2d(64, channel_out, kernel_size=3, padding=1),
            norm_layer(channel_out),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        x = self.downsample_conv(x)
        x = self.enlarge_conv(x)
        return x
```

OpenAI/Github’s Copilot

François Fleuret

Deep learning / 13.3. Transformer Networks
Input: I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model for 10 minutes. 10 minutes later, it’s suddenly 9:30 PM!
Model Output: This joke is about how time flies when you’re having fun. The person was supposed to start writing a paper at 5:00 PM, but they started playing with a language model instead. They thought they were only playing with it for 10 minutes, but it was actually 4.5 hours.

Input: Jennifer looked out her window and sees a really cool cloud below the commercial airplane. Clouds are usually below airplanes, so Jennifer is probably on an airplane. The answer is “yes”.

Input: The problem with kleptomania is that they always take things literally.
Explanation: This joke is wordplay. Someone who “takes things literally” is someone who doesn’t fully understand social cues and context, which is a negative trait. But the definition of kleptomania is someone who literally takes things.

Input: I tried 10,000 random restarts of my neural network, but I was accused of overfitting. I guess no good deed goes unpunished.
Model Output: This joke is a pun. A neural network is a computer program that can learn from data. A “seed” is a number that is used to initialize a random number generator. A “good seed” is a number that produces a good random number generator. The phrase “no good deed goes unpunished” means that if you do something good, you will be punished for it.

Input: What’s the difference between a zebra and an umbrella? One is a striped animal related to horses, another is a device you use to stop the rain from falling on you.
Model Output: This joke is an anti-joke. The joke is that the answer is obvious, and the joke is that you were expecting a funny answer.

Input: Reading “War and Peace” doesn’t scare me. I once had to review the E1 for my intern’s whole project on their final day!
Model Output: “War and Peace” is a very long book, so reading it is a daunting task. However, the speaker is saying they had to review the code for their intern’s project on their final day, which is even more daunting.

Input: I was going to fly to visit my family on April 6th. My mom said “Oh great, your step dad’s poetry reading is that night!” So now I’m flying in on April 7th.
Model Output: The joke is that the speaker’s mother is trying to get them to go to their step dad’s poetry reading, but the speaker doesn’t want to go, so they are changing their flight to the day after the poetry reading.
I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model for 10 minutes. I tried 10,000 random restarts of my neural network, but I was still not getting anywhere. I thought I might as well try something else.

**Inference Chaining**

**Input:** I found out my grandma was in the hospital. I felt a particular color. When someone cut me off in traffic, I felt a different color. What is the most likely color I would see if I combined these two colors?

**Model Output:** The color that I felt when I found out my grandma was in the hospital was likely a sad color, like red. The color that I felt when someone cut me off in traffic was likely an angry color, like red. If I combined these two colors, I would likely see purple. The answer is "purple".

**Input:** Michael is at that really famous museum in France looking at its most famous painting. However, the artist who made this painting just made two of his favorite cartoon characters from his childhood, what was the country of origin of the thing that the cartoon character usually holds in his hand?

**Model Output:** The most famous painting in the Louvre is the Mona Lisa. The artist who made the Mona Lisa is Leonardo da Vinci. Leonardo da Vinci is also the name of the main character in the cartoon Teenage Mutant Ninja Turtles. Leonardo da Vinci is from Italy. The thing that Leonardo da Vinci usually holds in his hand is a katana. The country of origin of the katana is Japan. The answer is "Japan".

**Input:** Guido was writing a Python program. He keeps getting confused by variable scope and looking it up on StackOverflow. Is it possible that the Guido in the prompt refers to Guido van Rossum?

**Model Output:** Guido van Rossum is the creator of Python. He would not need to look up variable scope on StackOverflow. The answer is "no".

**Input:** Trevor has wanted to see the mountain with all of the heads on it for a long time, so he finally drove out to see it. What is the capital of the state that is directly east of the state that Trevor is currently in?

**Model Output:** The mountain with all of the heads on it is Mount Rushmore. Mount Rushmore is in South Dakota. The state directly east of South Dakota is Minnesota. The capital of Minnesota is St. Paul. The answer is "St. Paul".

**Input:** Jennifer looked out her window and sees a really cool cloud below her. She unbuckles her seatbelt and heads to the bathroom. Is Jennifer probably traveling more than 300 miles per hour relative to the earth?

**Model Output:** 300 miles per hour is about 480 km/h. This is about the speed of a commercial airplane. Clouds are usually below airplanes, so Jennifer is probably on an airplane. The answer is "yes".

---

*(Chowdhery et al., 2022)*

François Fleuret

Deep learning / 13.3. Transformer Networks

28 / 42
Summary:

- The original Transformer (Vaswani et al., 2017) combines an encoder composed of blocks each made of a self-attention layer, and a per-token MLP, and a decoder composed of blocks each made of a causal self-attention layer, a cross attention layer, and a per-token MLP.
- BERT (Devlin et al., 2018) has the structure of the Transformer’s encoder.
- GPT (Radford, 2018; Radford et al., 2019) has the structure of the Transformer’s decoder without cross-attention.
- A model can be self-trained to predict masked words (BERT), or for auto-regression (GPT), and fine-tuned on downstream tasks.
- Special tokens can separate parts of inputs (e.g. question / answer) or indicate the output token used for prediction (e.g. sentiment analysis).
- These models scale extremely well to 100s of billions of tokens and parameters (Kaplan et al., 2020)
- Auto-regressive language models can be primed to solve with remarkable accuracy zero-shot learning tasks (Brown et al., 2020; Chowdhery et al., 2022).
Vision Transformers
As in NLP, attention mechanisms in vision allow models to leverage long-term dependencies that would require many convolutional layers, e.g. for Self-Attention Generative Adversarial Networks (SAGANs):

"The self-attention module is complementary to convolutions and helps with modeling long range, multi-level dependencies across image regions. Armed with self-attention, the generator can draw images in which fine details at every location are carefully coordinated with fine details in distant portions of the image."

(Zhang et al., 2018)
The Vision Transformer (ViT, Dosovitskiy et al. 2020) is a very simple architecture for image classification.

“Inspired by the Transformer scaling successes in NLP, we experiment with applying a standard Transformer directly to images, with the fewest possible modifications. To do so, we split an image into patches and provide the sequence of linear embeddings of these patches as an input to a Transformer. Image patches are treated the same way as tokens (words) in an NLP application. We train the model on image classification in supervised fashion.”

(Dosovitskiy et al., 2020)
Vision Transformer (ViT)

Transformer Encoder

(Dosovitskiy et al., 2020)
With large pre-training, Vision Transformers (ViT) performs better than Resnets adapted to “Big Transfer” (BiT).

(Dosovitskiy et al., 2020)
1 exaFLOPs $\approx$ 1h RTX 3090

(Dosovitskiy et al., 2020)
Notes

The left figure shows filters learned to encode the patches. They exhibit the expected structure of an image linear basis.
The right image shows for every one of the $7 \times 7$ patch positions the similarity of its positional encoding with the positional encodings of all other patches. The learned encodings reflect the rows and columns, and more generally the 2d structure of the image lattice.

(Dosovitskiy et al., 2020)
Notes

This scatter plot shows for every layer and every head the average distance in pixels between a patch and the patches it attends too. Early layers have very diverse “attention distances”, while the last one tend to have long distance attention only.

(Dosovitskiy et al., 2020)
Notes

This picture is obtained by looking at the attention of the output token and then going backward through layers, multiplying by the attention averaged across heads for each.
The Swin Transformer (Liu et al., 2021) improves the ViT architectures through the use of hierarchical representation with local attention in shifting windows.

(Liu et al., 2021)
The DETR algorithm (Carion et al., 2020) combines a CNN and a transformer for object detection.

\[
\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[ -\log \tilde{p}_{\sigma(i)}(c_i) + \mathds{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_i(i)) \right]
\]

(Carion et al., 2020)

**Notes**

A CNN converts the original image into a tensor of \(1/32\) the size, which is flattened as a sequence of tokens. The sequence is then fed into a Transformer with a non-causal feed-forward decoder (as opposed to the standard auto-regressive one). The maximum number of detections is specified by the number initial “object queries” given to the decoder. The final read-out is done with a per-token MLP that maps the internal feature dimension to the box coordinates, and a linear layer that maps the internal feature representation to the class (+ “background”) logits. The loss is computed for an optimal matching \(\sigma\) computed with the Hungarian algorithm between the ground truth and the \(N\) predicted detections.
Table 1: Comparison with Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for Faster R-CNN models in Detectron2 [50], the middle section shows results for Faster R-CNN models with GIoU [38], random crops train-time augmentation, and the long training schedule.

<table>
<thead>
<tr>
<th>Model</th>
<th>GFLOPS/FPS</th>
<th>#params</th>
<th>AP</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>AP_{S}</th>
<th>AP_{M}</th>
<th>AP_{L}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster RCNN-DC5</td>
<td>320/16</td>
<td>166M</td>
<td>39.0</td>
<td>60.5</td>
<td>42.3</td>
<td>21.4</td>
<td>43.5</td>
<td>52.5</td>
</tr>
<tr>
<td>Faster RCNN-FPN</td>
<td>180/26</td>
<td>42M</td>
<td>40.2</td>
<td>61.0</td>
<td>43.8</td>
<td>24.2</td>
<td>43.5</td>
<td>52.0</td>
</tr>
<tr>
<td>Faster RCNN-R101-FPN</td>
<td>246/20</td>
<td>60M</td>
<td>42.0</td>
<td>62.5</td>
<td>45.9</td>
<td>25.2</td>
<td>45.6</td>
<td>54.6</td>
</tr>
<tr>
<td>Faster RCNN-DC5+</td>
<td>320/16</td>
<td>166M</td>
<td>41.1</td>
<td>61.4</td>
<td>44.3</td>
<td>22.9</td>
<td>45.9</td>
<td>55.0</td>
</tr>
<tr>
<td>Faster RCNN-FPN+</td>
<td>180/26</td>
<td>42M</td>
<td>42.0</td>
<td>62.1</td>
<td>45.5</td>
<td>26.6</td>
<td>45.4</td>
<td>53.4</td>
</tr>
<tr>
<td>Faster RCNN-R101-FPN+</td>
<td>246/20</td>
<td>60M</td>
<td>44.0</td>
<td>63.9</td>
<td>47.8</td>
<td>27.2</td>
<td>48.1</td>
<td>56.0</td>
</tr>
<tr>
<td>DETR</td>
<td>86/28</td>
<td>41M</td>
<td>42.0</td>
<td>62.4</td>
<td>44.2</td>
<td>20.5</td>
<td>45.8</td>
<td>61.1</td>
</tr>
<tr>
<td>DETR-DC5</td>
<td>187/12</td>
<td>41M</td>
<td>43.3</td>
<td>63.1</td>
<td>45.9</td>
<td>22.5</td>
<td>47.3</td>
<td>61.1</td>
</tr>
<tr>
<td>DETR-R101</td>
<td>152/20</td>
<td>60M</td>
<td>43.5</td>
<td>63.8</td>
<td>46.4</td>
<td>21.9</td>
<td>48.0</td>
<td>61.8</td>
</tr>
<tr>
<td>DETR-DC5-R101</td>
<td>253/10</td>
<td>60M</td>
<td>44.9</td>
<td>64.7</td>
<td>47.7</td>
<td>23.7</td>
<td>49.5</td>
<td>62.3</td>
</tr>
</tbody>
</table>

(Obin et al., 2020)
Fig. 6: Visualizing decoder attention for every predicted object (images from COCO validation set image).

Table 4: Effect of loss components on AP. We train two models turning off different components at each decoder layer by evaluating the objects that would be predicted at each stage of the decoding (Fig. 4). Both AP and AP50 improve after every layer, totalling into a very significant +8.2/9.5 AP improvement between the first and last layer. With its set-based loss, DETR does not need NMS by design. To verify this we run a standard NMS procedure with default parameters [50] for the last layer. Using ℓ1 without GIoU shows poor results. We only studied

Table 3: Results for different positional encodings compared to the baseline (last row), which has fixed sine pos. encodings passed at every attention layer in both the encoder and the decoder. Learned embeddings are shared between all layers. Not using spatial positional encodings leads to a significant drop in AP. Interestingly, passing them in separate individual instances. Predictions are made with baseline DETR model on a validation set.

Fig. 3: Encoder self-attention for a set of reference points. The encoder is able to separate individual instances. Predictions are made with baseline DETR model on a validation set.

François Fleuret

Deep learning / 13.3. Transformer Networks
References


