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.
They first introduce a multi-head attention module.

Scaled Dot-Product Attention

Multi-Head Attention

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

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

\[
H_i = \text{Attention} \left( QW_i^Q, KW_i^K, VW_i^V \right), \ i = 1, \ldots, h
\]

with

\[
W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \ W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \ W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \ W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}
\]
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.
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

\begin{align*}
PE_{t,2i} &= \sin \left( \frac{t \cdot 2i}{10,000 \cdot d_{\text{model}}} \right) \\
PE_{t,2i+1} &= \cos \left( \frac{t \cdot 2i+1}{10,000 \cdot d_{\text{model}}} \right).
\end{align*}
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 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.

“Original” Transformer (Vaswani et al., 2017).
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></td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td>24.6</td>
<td>39.2</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>25.16</td>
<td>40.46</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>26.36</td>
<td>41.29</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>41.8</td>
</tr>
</tbody>
</table>

(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)
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)
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. $\mathcal{F}$ = sub-layer, and LN = 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.
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 [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.

BERT (Devlin et al., 2018)
We note that in the literature the bidirectional Transformer networks (Vaswani et al., 2017) as well as excellent guides such as “The Annotated Transformer.”

Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses unidirectional self-attention. This critical difference between the pre-trained architectures is the main reason for the performance gap between BERT and GPT on downstream tasks.

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 the BERT model, we consider two sizes: BASE ($L=12$, $H=768$, $A=12$, Total Parameters=340M) and LARGE ($L=24$, $H=1024$, $A=16$, Total Parameters=1100M).

Because the use of self-attention heads in BERT is crucial, we start our experiments with $A=1, 2, 3, 4, 8, 10$.

To measure how much can be gained from different subset sizes, we consider two cases: $A=2, 3, 4, 8$ and $A=1, 2, 3, 4$.

In all cases we set the feed-forward/filter size to be $4H$, i.e., 3072 for the $H=768$ and 4096 for the $H=1024$.

We primarily report results on two model sizes: BASE (LARGE) and $A=16$, Total Parameters=340M).

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)
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)
Head 8-10
- **Direct objects** attend to their verbs
- 86.8% accuracy at the **dobj** relation

Head 8-11
- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the **det** relation

(Clark et al., 2019)
- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the **poss** relation

- **Passive auxiliary verbs** attend to the verb they modify
- 82.5% accuracy at the **auxpass** relation

(Clark et al., 2019)
Head 9-6
- **Prepositions** attend to their objects
- 76.3% accuracy at the *pobj* relation

Head 5-4
- **Coreferent** mentions attend to their antecedents
- 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent

(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.

(Radford, 2018)
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 \[ \text{LayerNorm}(x) \]. 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 (Radford, 2018)
“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/).

```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.
```
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:

<table>
<thead>
<tr>
<th>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: <strong>positive</strong>,</th>
</tr>
</thead>
<tbody>
<tr>
<td>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: <strong>negative</strong>,</td>
</tr>
<tr>
<td>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: <strong>physics</strong>,</td>
</tr>
<tr>
<td>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: <strong>mathematics</strong>,</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Setting PTB SOTA (Zero-Shot)</th>
<th>35.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3 Zero-Shot</td>
<td>20.5</td>
</tr>
</tbody>
</table>

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.

3.1.1 Language Modeling

We calculate zero-shot perplexity on the Penn Tree Bank (PTB) [MKM+94] dataset measured in [RWC+19]. 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 [PKL+16] 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. [BHT+20] reflect on the small 1.5% improvement achieved by a doubling of model size between two recent state of the art results ([SPP+19] 11

(Brown et al., 2020)
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.

3.1.1 Language Modeling

We calculate zero-shot perplexity on the Penn Tree Bank (PTB) [MKM+94] dataset measured in [RWC+19]. 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 [PKL+16] 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. [BHT+20] reflect on
the small 1.5% improvement achieved by a doubling of model size between two recent state of the art results ([SPP+19]
(Brown et al., 2020)

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

Figure G.45: Evaluation example for Arithmetic 2Dx

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

<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>

(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()])
```

OpenAI/Github's Copilot
```python
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
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: 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.

Input: Reading "War and Peace" doesn't scare me. I once had to review the code for my intern’s whole project on his 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 think I understand why me and my partner are such a good match. I get excited when I smell food and become super alert when the doorbell rings. My girlfriend Shelley becomes ecstatic when she sees a person who she just saw 5 hours ago.

Model Output: This joke is about dogs. Dogs get excited when they smell food and when the doorbell rings. The girlfriend is excited to see someone she just saw 5 hours ago, which is a trait of dogs.

Input: I tried 10,000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed 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.

(Chowdhery et al., 2022)
Inference Chaining

(Chowdhery et al., 2022)
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).
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.
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.
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).
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."

(Cheng 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)
Table 1: Details of Vision Transformer model variants.

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Hidden size $D$</th>
<th>MLP size</th>
<th>Heads</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-Base</td>
<td>12</td>
<td>768</td>
<td>3072</td>
<td>12</td>
<td>86M</td>
</tr>
<tr>
<td>ViT-Large</td>
<td>24</td>
<td>1024</td>
<td>4096</td>
<td>16</td>
<td>307M</td>
</tr>
<tr>
<td>ViT-Huge</td>
<td>32</td>
<td>1280</td>
<td>5120</td>
<td>16</td>
<td>632M</td>
</tr>
</tbody>
</table>

(Dosovitskiy et al., 2020)
Transfer accuracy [%]
Average-5
Transformer (ViT)
ResNet (BiT)
Hybrid
Total pre-training compute [exaFLOPs]
1 exaFLOPs \approx 1h \text{ RTX 3090}

(Dosovitskiy et al., 2020)
RGB embedding filters (first 28 principal components)

(Dosovitskiy et al., 2020)
(Dosovitskiy et al., 2020)
(Dosovitskiy et al., 2020)
The Swin Transformer (Liu et al., 2021) improves the ViT architectures through the use of hierarchical representation with local attention in shifting windows.
The DETR algorithm (Carion et al., 2020) combines a CNN and a transformer for object detection.

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

(Carion et al., 2020)
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.

DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP\textsubscript{S} but greatly improved AP\textsubscript{L}. We use torchscript Faster R-CNN and DETR models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

<table>
<thead>
<tr>
<th>Model</th>
<th>GFLOPS/FPS</th>
<th>#params</th>
<th>AP</th>
<th>AP\textsubscript{50}</th>
<th>AP\textsubscript{75}</th>
<th>AP\textsubscript{S}</th>
<th>AP\textsubscript{M}</th>
<th>AP\textsubscript{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>

(Carion et al., 2020)
Fig. 6: Visualizing decoder attention for every predicted object (images from COCO validation set). Predictions are made with baseline DETR model on a validation image.

Table 4: Effect of loss components on AP. We train two models turning off different loss components and observe that decoder attention is fairly local, meaning that it mostly attends to object extremities, to 28.7M, leaving only 10.8M in the transformer, performance drops by 2.3 AP, and the decoder only leads to a minor AP drop. All these models use learned output positional encodings in our model: spatial positional encodings and output positional encodings.

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 decoder. Learned embeddings are shared between all layers. Not using spatial pos. enc. leads to a significant drop in AP. Interestingly, passing them in none sine at attn. gives poor results on its own, but when combined with learned at attn. and GIoU loss, and observe that GIou improves AP.

There are two kinds of positional encodings.

FFN inside transformers can be seen as 1D convolutions on the set of points (formerly bounding boxes) to the extremities to extract the class and object boundaries. The encoder is able to separate individual instances. Predictions are made with baseline DETR model on a validation set. Predictions are made with DETR-DC5 model. Attention scores are coded with different colors for different objects. Decoder typically attends to object extremities, such as legs and heads. Best viewed in color.

Note: The table and figures are not fully transcribed due to the limitations of the image processing and transcription tools. The text is cut off and not fully visible in the image.
The End
References


