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.

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \\
\text{MultiHead}(Q, K, V) = \text{Concat} \left( H_1, \ldots, H_h \right) W^O
\]

with

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

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 \( W_i^K \), \( W_i^Q \), and \( W_i^V \). And there is one final processing \( W^O \) applied on the concatenated results of the multiple heads.
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

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

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

---

**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 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.

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).
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

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.

(François Fleuret, Deep learning / 13.3. Transformer Networks)
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.

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 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.
We note that in the literature the bidirectional Transformer follows each other in the corpus. Predicting whether two sentences do two tasks at the same time: predicting missing words and predicting whether two sentences follow each other in the corpus.

(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).

CoLA

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).

STS-B

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.

MRPC

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).

RTE

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

WNLI

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.
Notes

Once again, visualizing the attention matrices show that it connects a word with other words that help to get its meaning.
Large Language Models
GPT (Generative Pre-Training, Radford, 2018) is a decoder of a transformer trained for auto-regressive text generation.

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.

**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.
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.

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.

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.

The `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”.

François Fleuret

Deep learning / 13.3. Transformer Networks
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>
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+

Context → Q: What is 98 plus 45?
A: 143

Figure G.45: Evaluation example for Arithmetic 2Dx

Context → Q: What is 95 times 45?
A: 4275

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

Context → Q: What is 509 minus 488?
A: 21

Figure G.47: Evaluation example for Arithmetic 3D+

Context → Q: What is 556 plus 497?
A: 1053

Figure G.48: Evaluation example for Arithmetic 4D-

Context → Q: What is 6209 minus 3365?
A: 2844

Figure G.49: Evaluation example for Arithmetic 4D+

Context → Q: What is 9923 plus 617?
A: 10540

Figure G.50: Evaluation example for Arithmetic 5D

Context → Q: What is 40649 minus 78746?
A: -38097

Figure G.51: Evaluation example for Arithmetic 5D+

<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
# 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
```
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.

(Chowdhery et al., 2022)
Inference Chaining

I will explain how to make logical inferences:

1. John wakes up to the sound of his children giggling outside the door. They bust in and say "Happy Father's Day!" with a plate full of food. 2022 has been a great year. Can we infer that John was born before the inauguration of Barack Obama?
   Answer: we can infer that John is a father because of "his children" we can infer that the current year is 2022. Barack Obama was inaugurated in 2009, so it has been about 13 years since Obama was inaugurated. John’s children are old enough to speak, so if John were born after Obama’s inauguration, he would have been younger than 13 when he had children. So, we can infer that John was born before Obama’s inauguration. The answer is "yes".

2. Sam sees a piano in the subway stations. Laughing, his friends prod him to play, saying "Remember how amazing your rendition of Chopsticks was at the Christmas party?" Can we infer that John enjoyed in piano at Juilliard?
   Answer: No, we can infer that John played piano at Juilliard would be an expert pianist. "Chopsticks" is a very simple song to play on the piano. His friends were therefore likely being sarcastic when they told him to play, which would imply that Sam was not an expert pianist. So it is unlikely that Sam enjoyed in piano at Juilliard. The answer is "no".

Input: I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model instead. They thought they were humorous analogies, reversal-of-expectations). The inference chaining examples were also written by the authors.

Model Output: The input is "I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model instead. They thought they were humorous analogies, reversal-of-expectations). The inference chaining examples were also written by the authors." The model output is "I was supposed to start writing the paper at 5:00 PM. But then I started playing with a language model instead. They thought they were humorous analogies, reversal-of-expectations). The inference chaining examples were also written by the authors."
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)
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)

Notes

With large pre-training, Vision Transformers (ViT) performs better than Resnets adapted to “Big Transfer” (BiT).
1 exaFLOPs \(\sim\) 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.
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.
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.
The DETR algorithm (Carion et al., 2020) combines a CNN and a transformer for object detection.

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

(Carion et al., 2020)

Notes

A CNN converts the original image into a tensor of 1/32 the size, which is flatten as a sequence of tokens.

The sequence if 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. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP\(_S\) but greatly improved AP\(_L\). 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(_S)</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></td>
<td></td>
<td></td>
<td></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 image).

- Using $\ell_1$ without GIoU shows poor results. We only studied
- Importance of FFN.
- There are two kinds of positional encodings in our model: spatial positional encodings and output positional encodings.
- Predictions are made with baseline DETR model on a validation set.
- Table 3: Results for different positional encodings compared to the baseline (last row), totalling into a very significant +8.2/9.5 AP improvement between the first and last layer of the transformer.
- AP for most of the model performance, losing only 0.7 AP to the baseline with combined losses. Using $\ell_1$ gives poor results on its own, but when combined with GIoU improves AP$^\Delta_{\text{GIoU}}$.
- Our baseline (last row) combines both losses.
- Similarly to visualizing encoder attention, we visualize decoder attentions in the second and subsequent layers, the self-attention mechanism has separated instances via global attention, the decoder only needs to attend to object extremities to extract the class and object boundaries.
- 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 at the input gives poor results, but when combined with other positional encodings, it improves performance.
- AP for different positional encodings.

<table>
<thead>
<tr>
<th>Positional Encoding</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sine at attn.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learned at attn.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sine at input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learned at input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learned at input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learned at input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- For different positional encodings.
- FFN inside transformers can be seen as 1 convolution layer.
- Over the activations allows the model to inhibit duplicate predictions. We observe that decoder attention is fairly local, meaning that it mostly attends to the output elements, and thus it is prone to making multiple predictions for the same object. In the second and subsequent layers, the self-attention mechanism serves that the improvement brought by NMS diminishes as depth increases. At the last layers, we observe a small loss in AP as NMS incorrectly removes true positive predictions.
- The outputs after each decoder are compared objects out of the outputs of every decoder layer. We analyze the importance of positional encodings.
- The importance of positional encodings.
- The self-attention(430, 600) and self-attention(520, 450) are two kinds of positional encodings in our model: spatial positional encodings and output positional encodings.
References


