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 models combining only attention layers.

They designed this “transformer” for a sequence-to-sequence translation task, but it is currently key to state-of-the-art approaches across NLP tasks.
They first introduce a multi-head attention module.

Their complete model is composed of:

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

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

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

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

The architecture is tested on English-to-German and English-to-French translation using the standard WMT2014 datasets.

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

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-DE</td>
<td>EN-FR</td>
</tr>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
<td>1.0 · 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td>24.6</td>
<td>2.3 · 10^{19}</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>25.16</td>
<td>9.6 · 10^{18}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>26.03</td>
<td>2.0 · 10^{19}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.36</td>
<td>7.7 · 10^{19}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>26.30</td>
<td>1.8 · 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.36</td>
<td>1.1 · 10^{21}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>1.2 · 10^{21}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>3.3 · 10^{18}</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>2.3 · 10^{19}</td>
</tr>
</tbody>
</table>

(Vaswani et al., 2017)
The Universal Transformer (Dehghani et al., 2018) is a similar model where all the blocks are identical, resulting in a recurrent model that iterates over consecutive revisions of the representation instead of positions.

Additionally the number of steps is modulated per position dynamically.
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.
Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for the training data. For similarity tasks, there is no inherent ordering of the two sentences being compared. 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 is trained on contiguous sequences of text, we require some modifications to apply it to these tasks.

For these tasks, we are given a context, a question \( q \), and a hypothesis \( h \). We concatenate the document context and two independent copies of the question and hypothesis. We then split each copy of the question and hypothesis into two parts and process them independently to produce two sequence representations. These representations are added element-wise before being fed into the linear output layer.

Training took 4 days to complete. The LM masking is used in BERT and is applied to all tokens except for the first and the last token in a sequence.

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

(Devlin et al., 2018)

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

(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 install implementations of the various flavors of transformers from HuggingFace (https://huggingface.co/)

```
pip install transformers
```

and use pre-trained models as we did for vision.
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(11):
    outputs, _ = model(torch.tensor([tokens]))
    next_token = torch.argmax(outputs[0, -1])
    tokens.append(next_token)

print(tokenizer.decode(tokens))

prints

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

BERT (Bidirectional Encoder Representation from Transformers, Devlin et al., 2018) is a transformer pre-trained with:

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

It is then fine-tuned on multiple NLP tasks.
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different 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)
### Table

| Head 9-6 | - Prepositions attend to their objects  
|          | - 76.3% accuracy at the pobj relation |

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

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

| Head 7-6 | - Possessive pronouns and apostrophes attend to the head of the corresponding NP  
|           | - 80.5% accuracy at the poss 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)

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

\[
y = \text{softmax} \left( (W_\theta x)^\top (W_\phi x) \right) W_g x.
\]

They insert “non-local blocks” in residual architectures and get improvements on both video and images classification.

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

(Wang et al., 2018)
The problem of long range interactions has been tackled in sequence modeling through the use of attention. Attention has been successfully applied in various applications, including text-to-speech [36] and generative sequence models [37, 38]. Several efforts have been made to incorporate attention into vision models instead of acting as an augmentation to convolution. To this end, we develop a simple mechanism for improving vision models.

### 2.1 Convolutions

Convolutional neural networks (CNNs) are typically employed with small neighborhoods (i.e. kernel size $k=3$). To increase the receptive field of a convolutional layer, one can employ average pooling with stride $d=2$, resulting in a region with shape $((i,j)-(i,j/k))$.

Following [46] we perform spatially fully-learned attention to each position $ij$. The relative distance is factorized across $(i,j)$, resulting in a region with shape $((i,j)-(i,j/k))$.

Relative positional embeddings [51, 46] can result in significantly better accuracies. Instead, attention provides a general mechanism for improving vision models.

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

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

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

(Ramachandran et al., 2019)

![Figure 2](image2.png)

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

![Figure 3](image3.png)

Figure 3: An example of a non-local block in res$_3$ computed by a 5-block non-local model trained on Kinetics. These example videos are from held-out validation videos. The starting point of arrows represents one $x_i$, and the ending points represent $x_j$. The 20 highest weighted arrows for each $x_i$ are visualized. The 4 frames are from a 32-frame input, shown with a stride of 8 frames. These visualizations show how the model finds related clues to support its prediction.

(Wang et al., 2018)
<table>
<thead>
<tr>
<th></th>
<th>FLOPS (B)</th>
<th>Params (M)</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>4.7</td>
<td>13.7</td>
<td>74.5</td>
</tr>
<tr>
<td>Conv-stem + Attention</td>
<td>4.5</td>
<td>10.3</td>
<td>75.8</td>
</tr>
<tr>
<td>Full Attention</td>
<td>4.7</td>
<td>10.3</td>
<td>74.8</td>
</tr>
</tbody>
</table>

Table 1: ImageNet classification results for a ResNet network with different depths. Baseline is a standard ResNet, Conv-stem + Attention uses spatial convolution in the stem and attention everywhere else, and Full Attention uses attention everywhere including the stem. The attention models outperform the baseline across all depths while having 12% fewer FLOPS and 29% fewer parameters.

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

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

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

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

(Cordonnier et al., 2020)
References


