Transformer, BERT

Milan Straka

April 24, 2023
For some sequence processing tasks, *sequential* processing (as performed by recurrent neural networks) of its elements might be too restrictive.

Instead, we may want to be able to combine sequence elements independently on their distance. Such processing is allowed in the **Transformer** architecture, originally proposed for neural machine translation in 2017 in *Attention is All You Need* paper.
Transformer

Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762
Transformer – Self-Attention

Assume that we have a sequence of $n$ words represented using a matrix $X \in \mathbb{R}^{n \times d}$.

The attention module for a queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$ and values $V \in \mathbb{R}^{n \times d_v}$ is defined as:

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

The queries, keys and values are computed from the input word representations $X$ using a linear transformation as

$$Q = X W^Q$$
$$K = X W^K$$
$$V = X W^V$$

for trainable weight matrices $W^Q, W^K \in \mathbb{R}^{d \times d_k}$ and $W^V \in \mathbb{R}^{d \times d_v}$. 
Transformer – Self-Attention

Input

Embedding

Thinking

X₁

Machines

X₂

Queries

q₁

q₂

Keys

k₁

k₂

Values

v₁

v₂

W_q

W_k

W_v

http://jalammar.github.io/images/t/transformer_self_attention_vectors.png
Transformer – Self-Attention

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 (\sqrt{d_k})$

Softmax

Softmax $X$

Value

Sum

http://jalammar.github.io/images/t/self-attention-output.png
Transformer – Self-Attention

\[
\begin{align*}
X \times W^Q &= Q \\
X \times W^K &= K \\
X \times W^V &= V
\end{align*}
\]

\[
softmax\left(\frac{Q K^T}{\sqrt{d_k}}\right) = Z
\]


Multihead attention is used in practice. Instead of using one huge attention, we split queries, keys and values to several groups (similar to how ResNeXt works), compute the attention in each of the groups separately, concatenate the results and multiply them by a matrix $W^O$.

Scaled Dot-Product Attention

Multi-Head Attention

Figure 2 of “Attention Is All You Need”, https://arxiv.org/abs/1706.03762
Transformer – Multihead Attention

ATTENTION HEAD #0

\[ Q_0 \]

\[ K_0 \]

\[ V_0 \]

\[ W_0^Q \]

\[ W_0^K \]

\[ W_0^V \]

ATTENTION HEAD #1

\[ Q_1 \]

\[ K_1 \]

\[ V_1 \]

\[ W_1^Q \]

\[ W_1^K \]

\[ W_1^V \]

http://jalammar.github.io/images/t/transformer_attention_heads_qkv.png
Transformer – Multihead Attention

1) Concatenate all the attention heads

\[ Z_0 \  Z_1 \  Z_2 \  Z_3 \  Z_4 \  Z_5 \  Z_6 \  Z_7 \]

2) Multiply with a weight matrix \( W^o \) that was trained jointly with the model

\[ x \]

3) The result would be the \( Z \) matrix that captures information from all the attention heads. We can send this forward to the FFNN

\[ \begin{align*}
\text{http://jalammar.github.io/images/t/transformer_attention_heads_z.png} \\
\text{http://jalammar.github.io/images/t/transformer_attention_heads_weight_matrix_o.png}
\end{align*} \]
Transformer – Multihead Attention

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^o to produce the output of the layer

* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one

Why Attention

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. $n$ is the sequence length, $d$ is the representation dimension, $k$ is the kernel size of convolutions and $r$ the size of the neighborhood in restricted self-attention.

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
<th>Maximum Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2 \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n \cdot d^2)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Convolutional</td>
<td>$O(k \cdot n \cdot d^2)$</td>
<td>$O(1)$</td>
<td>$O(\log_k(n))$</td>
</tr>
<tr>
<td>Self-Attention (restricted)</td>
<td>$O(r \cdot n \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(n/r)$</td>
</tr>
</tbody>
</table>

Table 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762
The self-attention is complemented with FFN layers, which is a fully connected ReLU layer with four times as many hidden units as inputs, followed by another fully connected layer without activation.

**Original “Post-LN” configuration**

**Improved “Pre-LN“ configuration since 2020**
Transformer – Post-LN Configuration including Residuals

Add & Normalize

Feed Forward

Feed Forward

Add & Normalize

LayerNorm(\(X + Z\))

Self-Attention

X1

POSitional Encoding

X2

http://jalammar.github.io/images/t/transformer_residual_layer_norm_2.png
Transformer – Pre-LN Configuration

Transformer Encoder Layer

Layer normalization

Self-Attention

Layer normalization

Positional embedding of 1

$\mathbf{x}_1$

Positional embedding of 2

$\mathbf{x}_2$

Positional embedding of $N$

$\mathbf{x}_N$

$\vdots$

$L$ times

$\vdots$

$L$ times

$\vdots$

$L$ times

$\vdots$

$L$ times

$\vdots$

$L$ times

$\vdots$
Transformer – Decoder

Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762
Masked Self-Attention

During decoding, the self-attention must attend only to earlier positions in the output sequence.

This is achieved by masking future positions, i.e., zeroing their weights out, which is usually implemented by setting them to $-\infty$ before the softmax calculation.

Encoder-Decoder Attention

In the encoder-decoder attentions, the queries come from the decoder, while the keys and the values originate from the encoder.
Transformer – Positional Embedding

![Transformer Diagram](http://jalammar.github.io/images/t/transformer_positional_encoding_vectors.png)

- **ENCODER #0**
- **ENCODER #1**
- **DECODER #0**
- **DECODER #1**

**EMBEDDING WITH TIME SIGNAL**

- $x_1$
- $x_2$
- $x_3$

**POSITIONAL ENCODING**

- $t_1$
- $t_2$
- $t_3$

**EMBEDDINGS**

- $x_1$
- $x_2$
- $x_3$

**INPUT**

- *je*
- *suis*
- *étudiant*
Positional Embeddings

We need to encode positional information (which was implicit in RNNs).

- Learned embeddings for every position.
- Sinusoids of different frequencies:

\[
\begin{align*}
\text{PE}_{(pos, 2i)} &= \sin \left( \frac{pos}{10000^{2i/d}} \right) \\
\text{PE}_{(pos, 2i+1)} &= \cos \left( \frac{pos}{10000^{2i/d}} \right)
\end{align*}
\]

This choice of functions should allow the model to attend to relative positions, since for any fixed \( k \), \( \text{PE}_{pos+k} \) is a linear function of \( \text{PE}_{pos} \), because

\[
\begin{align*}
\text{PE}_{(pos+k, 2i)} &= \sin \left( \frac{(pos + k)}{10000^{2i/d}} \right) \\
&= \sin \left( \frac{pos}{10000^{2i/d}} \right) \cdot \cos \left( \frac{k}{10000^{2i/d}} \right) + \cos \left( \frac{pos}{10000^{2i/d}} \right) \cdot \sin \left( \frac{k}{10000^{2i/d}} \right) \\
&= \text{offset}_{(k, 2i)} \cdot \text{PE}_{(pos, 2i)} + \text{offset}_{(k, 2i+1)} \cdot \text{PE}_{(pos, 2i+1)}.
\end{align*}
\]
Transformer – Positional Embeddings

Positional embeddings, 16 tokens, dimension 512

Token positions

Embedding dimensions

Transformer models:
- SelfAtt
- PosEmbed
- Training
- ELMo
- BERT
- mBERT
- RoBERTa
- OneToRuleThemAll
Transformer – Positional Embeddings

Positional embeddings, 64 tokens, dimension 512

Embedding dimensions

Token positions

0 64 128 192 256 320 384 448

0 8 16 24 32 40 48 56

0 0.5 1.0

-0.5 -1.0
Transformer – Positional Embeddings

Positional embeddings, 512 tokens, dimension 512

Token positions

Embedding dimensions

0 64 128 192 256 320 384 448

0 64 128 192 256 320 384 448

0 0.5 1.0

-0.5

-1.0
Transformer – Training

Regularization
The network is regularized by:

- dropout of input embeddings,
- dropout of each sub-layer, just before it is added to the residual connection (and then normalized),
- label smoothing.

Default dropout rate and also label smoothing weight is 0.1.

Parallel Execution
Because of the *masked attention*, training can be performed in parallel.
However, inference is still sequential.
Transformer – Training

Optimizer
Adam optimizer (with $\beta_2 = 0.98$, smaller than the default value of 0.999) is used during training, with the learning rate decreasing proportionally to inverse square root of the step number.

Warmup
Furthermore, during the first warmup_steps updates, the learning rate is increased linearly from zero to its target value.

$$\text{learning\_rate} = \frac{1}{\sqrt{d_{\text{model}}}} \min\left(\frac{1}{\sqrt{\text{step\_num}}}, \frac{\text{step\_num}}{\text{warmup\_steps}} \cdot \frac{1}{\sqrt{\text{warmup\_steps}}}\right).$$

In the original paper, 4000 warmup steps were proposed.
Wordpieces were constructed using BPE with a shared vocabulary of about 37k tokens.

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>2.3 \cdot 10^{19}</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>25.16</td>
<td>9.6 \cdot 10^{18}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>26.03</td>
<td>2.0 \cdot 10^{19}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.36</td>
<td>7.7 \cdot 10^{19}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td>8.0 \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>1.8 \cdot 10^{20}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>7.7 \cdot 10^{19}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>3.3 \cdot 10^{18}</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>2.3 \cdot 10^{19}</td>
</tr>
</tbody>
</table>
The PPL is \textit{perplexity per wordpiece}, where perplexity is $e^{H(P)}$, i.e., $e^{\text{loss}}$ in our case.
Main Takeaway

Generally, Transformer provides more powerful sequence-to-sequence architecture and also sequence element representation architecture than RNNs, but requires substantially more data.
At the end of 2017, a new type of *deep contextualized* word representations was proposed by Peters et al., called ELMo, *Embeddings from Language Models*.

The ELMo embeddings were based on a two-layer pre-trained LSTM language model, where a language model predicts following word based on a sentence prefix. Specifically, two such models were used, one for the forward direction and the other one for the backward direction.

![Embedding of “stick” in “Let’s stick to” - Step #1](http://jalammar.github.io/images/elmo-forward-backward-language-model-embedding.png)
To compute an embedding of a word in a sentence, the concatenation of the two language model's hidden states is used.

Pre-trained ELMo embeddings substantially improved several NLP tasks.
A year later after ELMo, at the end of 2018, a new model called BERT (standing for Bidirectional Encoder Representations from Transformers) was proposed. It is nowadays one of the most dominating approaches for pre-training word embeddings and for paragraph and document representations.
The BERT model computes contextualized representations using a bidirectional Transformer architecture.

Figure 3 of “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, https://arxiv.org/abs/1810.04805
The baseline BERT base model consists of 12 Transformer layers:

The bidirectionality is important, but it makes training difficult.
The input of the BERT model is a sequence of subwords, namely their identifiers. This input represents two so-called sentences, but they are in fact pieces of text with hundreds of subwords (512 maximum in total). The first token is a special CLS token and every sentence is ended by a SEP token.

Every subword representation is a sum of:

- trainable subword embeddings,
- trainable positional embeddings (not the sinusoidal embeddings, but I do not know why),
- trainable segment embeddings, which indicate if a token belongs to a sentence A (inclusively up to its SEP token) or to sentence B.
The BERT model is pretrained using two objectives:

- **masked language model** – 15% of the input words are **masked**, and the model tries to predict them (using a head consisting of a fully connected layer with softmax activation);
  - 80% of them are replaced by a special **MASK** token;
  - 10% of them are replaced by a random word;
  - 10% of them are left intact.

- **next sentence prediction** – the model tries to predict whether the second *sentence* followed the first one in the raw corpus (using a head that on top of the **CLS** output adds a fully connected layer with tanh activation (**pooler**), followed by a softmax-activated fully connected layer with two outputs).
  - 50% of the time the second sentence is the actual next sentence;
  - 50% of the time the second sentence is a random sentence from the corpus.

**Pre-training**

*Figure 1 of “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”,* [https://arxiv.org/abs/1810.04805](https://arxiv.org/abs/1810.04805)
For pre-training, English BookCorpus (800M words) and Wikipedia (2,500M words) are used, with a 30k WordPieces vocabulary.

Batch size is 256 sequences, each 512 subwords, giving 128k tokens per batch. Adam with learning rate 1e-4 is used, with linear learning rate warmup for the first 10k steps, followed by a linear learning rate decay to 0. Standard momentum parameters are used, and $L^2$ weight decay of 0.01 is utilized.

Dropout of 0.1 on all layers is used, and GELU activation is used instead of ReLU.

Furthermore, because longer sequences are quadratically more expensive, first 90% of the pre-training is performed on sequences of length 128, and only the last 10% use sequences of length 512.

Two variants are considered:

- BERT base with 12 layers, 12 attention heads and hidden size 768 (110M parameters),
- BERT large with 24 layers, 16 attention heads and hidden size 1024 (340M parameters).
ReLU multiplies the input by zero or one, depending on its value. Dropout stochastically multiplies the input by zero or one.

Both these functionalities are merged in Gaussian error linear units (GELUs), where the input value is multiplied by $m \sim \text{Bernoulli}(\Phi(x))$, where $\Phi(x) = P(x' \leq x)$ for $x' \sim N(0, 1)$ is the cumulative density function of the standard normal distribution.

The GELUs compute the expectation of this value, i.e.,

$$\text{GELU}(x) = x \cdot \Phi(x) + 0 \cdot (1 - \Phi(x)) = x\Phi(x).$$

GELUs can be approximated using (no need to remember this):

$$0.5x \left( 1 + \tanh \left[ \sqrt{2/\pi}(x + 0.044715x^3) \right] \right) \quad \text{or} \quad x\sigma(1.702x).$$

Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

*Figure 1 of "Gaussian Error Linear Units (GELUs)".*
https://arxiv.org/abs/1606.08415
The pre-trained BERT model can be finetuned on a range of tasks:

- **sentence element representation**
  - PoS tagging
  - named entity recognition
  - ...

- **sentence representation**
  - text classification

- **sentence relation representation**
  - textual entailment, aka natural language inference (the second sentence is *implied by/contradicts/has no relation to* the first sentence)
  - textual similarity
  - paraphrase detection
  - natural language inference

Figure 4 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

Figure 4 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805
For finetuning, dropout 0.1 is used, usually very small number of epochs (2-4) suffice, and a good learning rate is usually one of 5e-5, 3e-5, 2e-5.

Table 1 of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT\text{BASE}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT\text{LARGE}</td>
<td>\textbf{86.7/85.9}</td>
<td>\textbf{72.1}</td>
<td>\textbf{92.7}</td>
<td>\textbf{94.9}</td>
<td>\textbf{60.5}</td>
<td>\textbf{86.5}</td>
<td>\textbf{89.3}</td>
<td>\textbf{70.1}</td>
<td>\textbf{82.1}</td>
</tr>
</tbody>
</table>

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.
Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

Table 3: SQuAD 2.0 results. We exclude entries that the SWAG paper.

Figure 5: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for $k$ steps. The $x$-axis is the value of $k$.

Table 8: Ablation over different masking strategies.

<table>
<thead>
<tr>
<th>Masking Rates</th>
<th>MNLI Dev Set Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev Set Results</td>
</tr>
<tr>
<td></td>
<td>Fine-tune</td>
</tr>
<tr>
<td>80% 10% 10%</td>
<td>84.2</td>
</tr>
<tr>
<td>100% 0% 0%</td>
<td>84.3</td>
</tr>
<tr>
<td>80% 0% 20%</td>
<td>84.1</td>
</tr>
<tr>
<td>80% 20% 0%</td>
<td>84.4</td>
</tr>
<tr>
<td>0% 20% 80%</td>
<td>83.7</td>
</tr>
<tr>
<td>0% 0% 100%</td>
<td>83.6</td>
</tr>
<tr>
<td>100% 0% 0%</td>
<td>95.4</td>
</tr>
<tr>
<td>80% 0% 20%</td>
<td>95.2</td>
</tr>
<tr>
<td>80% 20% 0%</td>
<td>95.2</td>
</tr>
<tr>
<td>0% 20% 80%</td>
<td>94.8</td>
</tr>
<tr>
<td>0% 0% 100%</td>
<td>94.9</td>
</tr>
</tbody>
</table>

**Table 8** of "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", https://arxiv.org/abs/1810.04805
**Multilingual BERT**

The Multilingual BERT is pre-trained on 102-104 largest Wikipedias, including the Czech one. There are two versions, the *cased* one has WordPieces including case, and the *uncased* one with subwords all in lower case and *without diacritics*.

Even if only very small percentage of input sentences were Czech, it works surprisingly well for Czech NLP.

Furthermore, without any explicit supervision, mBERT is able to represent the input languages in a *shared* space, allowing cross-lingual transfer.
Consider a reading comprehension task, where for a given paragraph and a question an answer needs to be located in the paragraph. Then training the model in English and then directly running it on a different language works comparably to translating the data to English and then back.


<table>
<thead>
<tr>
<th>F1 / EM</th>
<th>en</th>
<th>es</th>
<th>de</th>
<th>ar</th>
<th>hi</th>
<th>vi</th>
<th>zh</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Large</td>
<td>80.2 / 67.4</td>
<td>-</td>
<td>64.3 / 46.6</td>
<td>57.9 / 44.3</td>
<td>-</td>
<td>45.7 / 29.8</td>
<td>-</td>
</tr>
<tr>
<td>Multilingual-BERT</td>
<td>77.7 / 65.2</td>
<td>60.2 / 46.3</td>
<td>57.9 / 44.3</td>
<td>45.7 / 29.8</td>
<td>43.8 / 29.7</td>
<td>51.7 / 38.6</td>
<td>57.5 / 37.3</td>
</tr>
<tr>
<td>XLM</td>
<td>74.9 / 62.4</td>
<td>66.0 / 49.8</td>
<td>62.2 / 47.6</td>
<td>54.8 / 36.3</td>
<td>48.8 / 27.3</td>
<td>61.4 / 41.8</td>
<td>61.1 / 39.6</td>
</tr>
<tr>
<td>Translate test, BERT-L</td>
<td>-</td>
<td>65.4 / 44.0</td>
<td>57.9 / 41.8</td>
<td>33.6 / 20.4</td>
<td>23.8 / 18.9*</td>
<td>58.2 / 33.2</td>
<td>44.2 / 20.3</td>
</tr>
<tr>
<td>Translate train, M-BERT</td>
<td>-</td>
<td>53.9 / 37.4</td>
<td>62.0 / 47.5</td>
<td>51.8 / 33.2</td>
<td>55.0 / 40.0</td>
<td>62.0 / 43.1</td>
<td>61.4 / 39.5</td>
</tr>
<tr>
<td>Translate train, XLM</td>
<td>-</td>
<td>65.2 / 47.8</td>
<td>61.4 / 46.7</td>
<td>54.0 / 34.4</td>
<td>50.7 / 33.4</td>
<td>59.3 / 39.4</td>
<td>59.8 / 37.9</td>
</tr>
</tbody>
</table>
The next sentence prediction was originally hypothesized to be an important factor during training of the BERT model, as indicated by ablation experiments. However, later experiments indicated removing it might improve results.

The RoBERTa authors therefore performed the following experiments:

- SEGMENT-PAIR: pair of segments with at most 512 tokens in total;
- SENTENCE-PAIR: pair of natural sentences, usually significantly shorter than 512 tokens;
- FULL-SENTENCES: just one segment on input with 512 tokens, can cross document boundary;
- DOC-SENTENCES: just one segment on input with 512 tokens, cannot cross document boundary.

<table>
<thead>
<tr>
<th>Model</th>
<th>SQuAD 1.1/2.0</th>
<th>MNLI-m</th>
<th>SST-2</th>
<th>RACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our reimplementation (with NSP loss):</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEGMENT-PAIR</td>
<td>90.4/78.7</td>
<td>84.0</td>
<td>92.9</td>
<td>64.2</td>
</tr>
<tr>
<td>SENTENCE-PAIR</td>
<td>88.7/76.2</td>
<td>82.9</td>
<td>92.1</td>
<td>63.0</td>
</tr>
<tr>
<td>Our reimplementation (without NSP loss):</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FULL-SENTENCES</td>
<td>90.4/79.1</td>
<td>84.7</td>
<td>92.5</td>
<td>64.8</td>
</tr>
<tr>
<td>DOC-SENTENCES</td>
<td>90.6/79.7</td>
<td>84.7</td>
<td>92.7</td>
<td>65.6</td>
</tr>
<tr>
<td>BERT\textsubscript{BASE}</td>
<td>88.5/76.3</td>
<td>84.3</td>
<td>92.8</td>
<td>64.3</td>
</tr>
<tr>
<td>XLNet\textsubscript{BASE} (K = 7)</td>
<td>−/81.3</td>
<td>85.8</td>
<td>92.7</td>
<td>66.1</td>
</tr>
<tr>
<td>XLNet\textsubscript{BASE} (K = 6)</td>
<td>−/81.0</td>
<td>85.6</td>
<td>93.4</td>
<td>66.7</td>
</tr>
</tbody>
</table>

Table 2 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692
RoBERTa – Larger Batches

BERT is trained for 1M steps with a learning rate of 1e-4.

The RoBERTa authors also considered larger batches (with linearly larger learning rate).

<table>
<thead>
<tr>
<th>bsz</th>
<th>steps</th>
<th>lr</th>
<th>ppl</th>
<th>MNLI-m</th>
<th>SST-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1M</td>
<td>1e-4</td>
<td>3.99</td>
<td>84.7</td>
<td>92.7</td>
</tr>
<tr>
<td>2K</td>
<td>125K</td>
<td>7e-4</td>
<td>3.68</td>
<td>85.2</td>
<td>92.9</td>
</tr>
<tr>
<td>8K</td>
<td>31K</td>
<td>1e-3</td>
<td>3.77</td>
<td>84.6</td>
<td>92.8</td>
</tr>
</tbody>
</table>

Table 3: Perplexity on held-out training data (ppl) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (bsz). We tune the learning rate (lr) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

Table 3 of “RoBERTa: A Robustly Optimized BERT Pretraining Approach”, https://arxiv.org/abs/1907.11692
The RoBERTa model, Robustly optimized BERT approach, is trained with dynamic masking, FULL-SENTENCES without NSP, large 8k minibatches and byte-level BPE with 50k subwords.

<table>
<thead>
<tr>
<th>Model</th>
<th>data</th>
<th>bsz</th>
<th>steps</th>
<th>SQuAD (v1.1/2.0)</th>
<th>MNLI-m</th>
<th>SST-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoBERTa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>16GB</td>
<td>8K</td>
<td>100K</td>
<td>93.6/87.3</td>
<td>89.0</td>
<td>95.3</td>
</tr>
<tr>
<td>+ additional data (§3.2)</td>
<td>160GB</td>
<td>8K</td>
<td>100K</td>
<td>94.0/87.7</td>
<td>89.3</td>
<td>95.6</td>
</tr>
<tr>
<td>+ pretrain longer</td>
<td>160GB</td>
<td>8K</td>
<td>300K</td>
<td>94.4/88.7</td>
<td>90.0</td>
<td>96.1</td>
</tr>
<tr>
<td>+ pretrain even longer</td>
<td>160GB</td>
<td>8K</td>
<td>500K</td>
<td><strong>94.6/89.4</strong></td>
<td><strong>90.2</strong></td>
<td><strong>96.4</strong></td>
</tr>
<tr>
<td>BERT_\text{LARGE}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>13GB</td>
<td>256</td>
<td>1M</td>
<td>90.9/81.8</td>
<td>86.6</td>
<td>93.7</td>
</tr>
<tr>
<td>XLNet_\text{LARGE}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>13GB</td>
<td>256</td>
<td>1M</td>
<td>94.0/87.8</td>
<td>88.4</td>
<td>94.4</td>
</tr>
<tr>
<td>+ additional data</td>
<td>126GB</td>
<td>2K</td>
<td>500K</td>
<td>94.5/88.8</td>
<td>89.8</td>
<td>95.6</td>
</tr>
</tbody>
</table>

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of BERT\text{LARGE}. Results for BERT\text{LARGE} and XLNet\text{LARGE} are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

Table 4 of “RoBERTa: A Robustly Optimized BERT Pretraining Approach”, https://arxiv.org/abs/1907.11692
**RoBERTa Results**

<table>
<thead>
<tr>
<th>Model</th>
<th>SQuAD 1.1 EM</th>
<th>SQuAD 1.1 F1</th>
<th>SQuAD 2.0 EM</th>
<th>SQuAD 2.0 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single models on dev, w/o data augmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>84.1</td>
<td>90.9</td>
<td>79.0</td>
<td>81.8</td>
</tr>
<tr>
<td>XLNet&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>89.0</td>
<td>94.5</td>
<td>86.1</td>
<td>88.8</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>88.9</td>
<td>94.6</td>
<td>86.5</td>
<td>89.4</td>
</tr>
<tr>
<td>Ensembles on test (from leaderboard as of July 25, 2019)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALICE</td>
<td>88.2/87.9</td>
<td>95.7</td>
<td>90.7</td>
<td>83.5</td>
</tr>
<tr>
<td>MT-DNN</td>
<td>87.9/87.4</td>
<td>96.0</td>
<td>90.3</td>
<td>89.9</td>
</tr>
<tr>
<td>XLNet</td>
<td>90.2/89.8</td>
<td>98.7</td>
<td>90.7</td>
<td>90.6</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>90.8/90.2</td>
<td>98.9</td>
<td>90.2</td>
<td>88.2</td>
</tr>
</tbody>
</table>

Table 5: Results on GLUE. All results are based on a 24-layer architecture. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of single-task models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

Table 6 of "RoBERTa: A Robustly Optimized BERT Pretraining Approach", https://arxiv.org/abs/1907.11692

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Middle</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single models on test (as of July 25, 2019)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>72.0</td>
<td>76.6</td>
<td>70.1</td>
</tr>
<tr>
<td>XLNet&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>81.7</td>
<td>85.4</td>
<td>80.2</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>83.2</td>
<td>86.5</td>
<td>81.3</td>
</tr>
</tbody>
</table>

Table 7: Results on the RACE test set. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Yang et al. (2019).
The Transformer architecture has been spreading and has become almost omnipresent in tasks where data is available in abundance, in which case it usually delivers superior performance.

### Language Models

- **GPT**: Transformer decoder-only model, 2018, ~150M parameters
- **GPT-2**: Transformer decoder-only model, 2019, ~1.5B parameters
- **GPT-3**: Transformer decoder-only model, May 2020, ~175B parameters
- **GPT-4**: Transformer decoder-only model, March 2023, ???; $100M
- **Gopher**: Transformer decoder-only model, Dec 2021, ~280B parameters
- **XGLM**: multilingual Transformer decoder-only model, Dec 2021, ~7.5B parameters
  - 30 languages from 16 language families
- **Megatron-Turing NLP**: Transformer decoder-only model, Jan 2022, ~530B parameters
  - trained using 2240 A100
- **PaLM**: Transformer decoder-only model, Apr 2022, ~580B parameters
  - trained using 6144 TPUv4; 22% training data non-English
PaLM Jokes Explanations

I will explain these jokes:
(1) The problem with kleptomanics 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.
(2) Always borrow money from a pessimist. They'll never expect it back.
Explanation: Most people expect you to pay them back when you borrow money, however a pessimist is someone who always assumes the worst, so if you borrow money from them, they will expect that you won’t pay them back anyways.

**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:** 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 the day after the poetry reading.

**Input:** Reading "War and Peace" doesn’t scare me. I once had to review the CL 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.

---

**Figure 19 of “PaLM: Scaling Language Modeling with Pathways”, https://arxiv.org/abs/2204.02311**
Various pre-trained encoder-decoder models are available:

- BART, 2019, ~200M parameters
- T5, Oct 2019, up to ~11B parameters
  - REALM, Feb 2020, ~330M, uses explicit retrieval from a large knowledge base
- mT5, Oct 2020, ~100 languages, up to ~13B parameters
  - sizes small (300M), base (582M), large (1.23B), xl (3.74B), xxl (12.9B)
- ByT5, May 2021, byte-based, ~100 languages, same sizes as mT5

*Figure 1 of “REALM: Retrieval-Augmented Language Model Pre-Training”, https://arxiv.org/pdf/2002.05202.pdf*
In Oct 2020, an influential paper

*An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*

proposed processing of images using ViT a variant of the Transformers architecture (Visual Transformer)
The ViT architecture surpasses convolutional models like EfficientNet when pre-trained on very large data (~300M images); however, training only on ImageNet1k delivered worse results (77.9% top-1 accuracy).

An improved training with a variety of augmentation (DeiT architecture, Dec 2020) resulted in performance close to EfficientNet when trained only on ImageNet1k data (83.1% vs 84.7% top-1 accuracy).

_DeiT III: Revenge of the ViT_ (Apr 2022) has presented simplified training procedure, achieving results analogous to EfficientNetV2 on ImageNet1k (85.2% vs 85.7% top-1 accuracy) and ImageNet21k (87.2% vs 87.3% top-1 accuracy).
When data is limited ("only" 1M images), the best approach to train a ViT seems to be a BERT-like masking, which was proposed in Nov 2021 paper "Masked Autoencoders Are Scalable Vision Learners".

This MAE architecture reaches 86.9% top-1 accuracy on ImageNet1k-only training on images of size 224, and 87.8% on images of size 448.
As of April 2023, the current best model on ImageNet is BASIC-L trained with Lion optimizer. The image encoder in BASIC-L is CoAtNet-7, an architecture combining MBConv in first stages and relative pre-activated 2D self-attention in later stages, with GELU activations everywhere. The image encoder has 2.4B parameters, it is trained on 6.6B noisy image-text pairs using a batch size of 65536 images in ~7k TPUv4/days, and achieves 91.1% top-1 accuracy.

It also achieves 88.3% zero-shot accuracy on ImageNet, i.e., when no ImageNet training data is used for training nor finetuning.
Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call object queries, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.
The encoder uses fixed sine positional encodings added to every self-attention layer. The $x$ and $y$ axes are encoded independently and concatenated.

During training, we pair the predictions and gold objects (padded with “no object”s to the same length) using a maximum-weight bipartite matching algorithm – the Hungarian algorithm.