Transformers

Bryan Pardo
Deep Learning
Northwestern University
We’re ready for Transformers!
Pay Attention!

- Transformers introduced in 2017
- Use attention
- Do NOT use recurrent layers
- Do NOT use convolutional layers
- Hence the title of the paper that introduced them

Features of Transformers

• They use residual connections to allow deeper networks to fine-tune as appropriate

• They use attention in multiple places in both the encoder and decoder

• People often mistake them for automobiles
Step-by-step
The Input sequence

• Let’s build a representation that takes the input sequence and learns relationships between input tokens (words represented by their embeddings)

Embedding: 0 734 912 733 43 1

Input sentence: [START] Thinking machines think quickly [STOP]
Positional encoding

• In an RNN, the recurrence encodes the order implicitly.

• In a Transformer, relatedness between words is handled by self-attention.

• If we’re not using recursion to implicitly encode order, how does the system tell the difference between these two sentences?

Bob hands Maria the ball.

Maria hands Bob the ball.
Positional encoding

• Lots of ways to go
  e.g. just number the items.

• They chose to implicitly encode position by adding the values of a bunch of sinewaves to the embeddings

• Honestly, I’m not sure why they did it this way, rather than just appending an index number to the embedding
Positional encoding

\[ PE_{\text{pos}, 2i} = \sin(\text{pos}/10000^{2i}/d_{\text{model}}) \]

\[ PE_{\text{pos}, 2i+1} = \cos(\text{pos}/10000^{2i}/d_{\text{model}}) \]

- \( i \) = the index of a dimension in a single input embedding vector \( x \)
- \( d_{\text{model}} \) = the total number of dimensions in the embedding
- \( \text{pos} \) = the position in the sequence of the input embedding vector
A concrete example

For example, for word \( w \) at position \( pos \in [0, L - 1] \) in the input sequence \( w = (w_0, \ldots, w_{L-1}) \), with 4-dimensional embedding \( e_w \), and \( d_{model} = 4 \), the operation would be

\[
e'_w = e_w + \left[ \sin \left( \frac{pos}{10000^0} \right), \cos \left( \frac{pos}{10000^0} \right), \sin \left( \frac{pos}{10000^{2/4}} \right), \cos \left( \frac{pos}{10000^{2/4}} \right) \right]
\]

\[
e'_w = e_w + \left[ \sin (pos), \cos (pos), \sin \left( \frac{pos}{100} \right), \cos \left( \frac{pos}{100} \right) \right]
\]

where the formula for positional encoding is as follows

\[
PE(pos, 2i) = \sin \left( \frac{pos}{10000^{2i/d_{model}}} \right),
\]

\[
PE(pos, 2i + 1) = \cos \left( \frac{pos}{10000^{2i/d_{model}}} \right).
\]

with \( d_{model} = 512 \) (thus \( i \in [0, 255] \)) in the original paper.

Thanks for the example, anonymous contributor on Stack Overflow!
https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model
Some positional encodings visualized

https://nlp.seas.harvard.edu/2018/04/03/attention.html#decoder
Step-by-step
Self Attention: Query – Key – Value

http://jalammar.github.io/illustrated-transformer/
Self Attention: Query – Key – Value

• QUERY: Built from embedded input sequence
• KEY: Same as query
• VALUE: Same as query
• ATTENTION: “relatedness” between pairs of words
• CONTEXT: The sequence of context values

\[
\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V
\]
Multi-head Self Attention

- Each attention head will have unique learned weight matrices for the query (Q), key (K), and value (V), where \( i \) is the index of each attention head

\[
W_i^Q, W_i^K, W_i^V
\]

- There’s also output weight to learn for the multi-head layer

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O
\]

where \( \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \)
Here’s the figure from the paper
Self-attention: one head, all words
Self attention: Multiple heads, one word

Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.
Step-by-step

\[ FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \]

Two linear layers, with a ReLU in between
Step-by-step

One encoder block

Nx: The number of encoder blocks IN SEQUENCE
Step-by-step

• Run the encoder on the ENTIRE input language sequence.

• The decoder outputs tokens one-at-a-time, feeding the previous token output into the model to generate the next token.

• The next decoder output is conditioned on the ENTIRE sequence of encoder outputs + the previous decoder output.
Step-by-step
Step-by-step
Masked attention

- Don’t let the attention “look ahead” to sequence elements the system hasn’t generated yet
- Apply a “mask” matrix with 0 everywhere you’re not allowed to look and 1 everywhere else
- Do element-wise multiplication to the value vector

\[
\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \odot M
\]
3 variants of attention
That’s the whole model
Transformer encoding: parallelizable

Image from https://jalammar.github.io/illustrated-transformer/
Transformer decoding: autoregressive

Decoding time step: 1 2 3 4 5 6

INPUT: Je suis étudiant

OUTPUT

Embedding with time signal

Embeddings

Image from https://github.io/illustrated-transformer/
State-of-the-art language translation

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></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></td>
<td></td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
<td>39.92</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
<td>40.4</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td>40.4</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>41.29</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>
<tr>
<td></td>
<td></td>
<td>3.3 \cdot 10^{18}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.3 \cdot 10^{19}</td>
</tr>
</tbody>
</table>
OK I’m convinced….attention is great but...

• What if I’m doing (single) language modeling instead of translation?

• Do I really need this whole encoder-decoder framework?

• No. You don’t (more on this in a minute)
Training a good language model is...

• Slow

• Requires lots of data

• Not every task has enough labeled data

• Requires lots of computing resources

• Not everyone has enough computing resources
Let’s use TRANSFER LEARNING!

- Train a general model on some task using a huge dataset for a long time
- Fine-tune the trained model on a variety of related “downstream” tasks
- Allows re-use of architecture
- Allows training on tasks where there is relatively little data
Use half of a Transformer

• Get rid of the encoder

• Use the decoder as a language model

• Note that this is an autoregressive model
GPT: Generative Pre-Training

Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT was state of the art in 2018

<table>
<thead>
<tr>
<th>DATASET</th>
<th>TASK</th>
<th>SOTA</th>
<th>OURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNLI</td>
<td>Textual Entailment</td>
<td>89.3</td>
<td>89.9</td>
</tr>
<tr>
<td>MNLI Matched</td>
<td>Textual Entailment</td>
<td>80.6</td>
<td>82.1</td>
</tr>
<tr>
<td>MNLI Mismatched</td>
<td>Textual Entailment</td>
<td>80.1</td>
<td>81.4</td>
</tr>
<tr>
<td>SciTail</td>
<td>Textual Entailment</td>
<td>83.3</td>
<td>88.3</td>
</tr>
<tr>
<td>QNLI</td>
<td>Textual Entailment</td>
<td>82.3</td>
<td>88.1</td>
</tr>
<tr>
<td>RTE</td>
<td>Textual Entailment</td>
<td>61.7</td>
<td>56.0</td>
</tr>
<tr>
<td>STS-B</td>
<td>Semantic Similarity</td>
<td>81.0</td>
<td>82.0</td>
</tr>
<tr>
<td>QQP</td>
<td>Semantic Similarity</td>
<td>66.1</td>
<td>70.3</td>
</tr>
<tr>
<td>MRPC</td>
<td>Semantic Similarity</td>
<td>86.0</td>
<td>82.3</td>
</tr>
<tr>
<td>RACE</td>
<td>Reading Comprehension</td>
<td>53.3</td>
<td>59.0</td>
</tr>
<tr>
<td>ROCStories</td>
<td>Commonsense Reasoning</td>
<td>77.6</td>
<td>86.5</td>
</tr>
<tr>
<td>COPA</td>
<td>Commonsense Reasoning</td>
<td>71.2</td>
<td>78.6</td>
</tr>
<tr>
<td>SST-2</td>
<td>Sentiment Analysis</td>
<td>93.2</td>
<td>91.3</td>
</tr>
<tr>
<td>CoLA</td>
<td>Linguistic Acceptability</td>
<td>35.0</td>
<td>45.4</td>
</tr>
<tr>
<td>GLUE</td>
<td>Multi Task Benchmark</td>
<td>68.9</td>
<td>72.8</td>
</tr>
</tbody>
</table>
BERT
Bidirectional Encoder Representations from Transformers

Use the OTHER half of a Transformer

- Get rid of the decoder
- Use the encoder as a language model
## Input encoding to BERT

![Input encoding to BERT](image)

**Figure 2:** BERT input representation. The input embeddings are the sum of the token embeddings, segment embeddings and the position embeddings.
Masked word prediction training

• Randomly cover up a word in a sentence

Original sentence: My dog has fleas.

Training sentences:

[MASK] dog has fleas.
My [MASK] has fleas.
My dog [MASK] fleas.
My dog has [MASK].
Model learns to expect [MASK]. Here’s the fix.

- 80% of the time: Replace the word with the [MASK] token, e.g., my dog is hairy → my dog is [MASK]

- 10% of the time: Replace the word with a random word, e.g., my dog is hairy → my dog is apple

- 10% of the time: Keep the word unchanged, e.g., my dog is hairy → my dog is hairy. The purpose of this is to bias the representation towards the actual observed word.
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).
Language Tasks

**MNLI**  Multi-Genre Natural Language Inference is a large-scale, crowdsourced entailment classification task (Williams et al., 2018). Given a pair of sentences, the goal is to predict whether the second sentence is an *entailment*, *contradiction*, or *neutral* with respect to the first one.

**QQP**  Quora Question Pairs is a binary classification task where the goal is to determine if two questions asked on Quora are semantically equivalent (Chen et al., 2018).

**QNLI**  Question Natural Language Inference is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016) which has been converted to a binary classification task (Wang et al., 2018a). The positive examples are (question, sentence) pairs which do contain the correct answer, and the negative examples are (question, sentence) from the same paragraph which do not contain the answer.

**SST-2**  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).
## General Language Understanding Evaluation

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm) 392k</th>
<th>QQP 363k</th>
<th>QNLI 108k</th>
<th>SST-2 67k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</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>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</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>
</tr>
<tr>
<td>BERT\text{LARGE}</td>
<td><strong>86.7/85.9</strong></td>
<td><strong>72.1</strong></td>
<td><strong>92.7</strong></td>
<td><strong>94.9</strong></td>
</tr>
</tbody>
</table>
Bigger is better?

OpenAI: GPT3: 175 billion parameters

Microsoft: Turing-NLG: 20 billion parameters

NVIDIA: Megatron: 8 billion

OpenAI: GPT-2: 1.5 billion parameters

Google: BERT (Transformer): 300 million parameters

OpenAI: GPT (Transformer): 100 million parameters

Google: Seq2Seq LSTM: 65 million parameters
Bigger is better?

Human Brain: $10^{15}$ connections
https://www.nature.com/articles/d41586-019-02208-0

OpenAI: GPT3: 175 billion parameters

Microsoft: Turing-NLG: 20 billion parameters

NVIDIA: Megatron: 8 billion

OpenAI: GPT-2: 1.5 billion parameters

Google: BERT (Transformer): 300 million parameters

OpenAI: GPT (Transformer): 100 million parameters

Google: Seq2Seq LSTM: 65 million parameters