ATTENTION IS ALL YOU NEED
INTRODUCTION
Sequence Transduction

• Convert a *sequence* to another *sequence*
Past Works – RNNs

• Sequential nature disallows parallelization within training samples
• Long distance between input and output
Past Works – Attention

• Allow modelling dependencies disregarding distance
• Used with RNNs or CNNs
Past Works – CNN

• Allow parallelism
• Difficult to learn dependencies between distant positions
Our Work – Transformer

• Rely entirely on *self-attention* to compute representations
• Allows for greater parallelization and short path lengths
Sneak Peek

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>
MODEL ARCHITECTURE
MODEL ARCHITECTURE
ENCODER AND DECODER STACKS
Encoder-Decoder Structure

INPUT: Je suis étudiant

OUTPUT: I am a student
Stacked Layers

\[ N = 6 \]

INPUT \( \text{Je suis étudiant} \)

OUTPUT \( \text{I am a student} \)
Encoder Sub-layers
Decoder Sub-layers

Encoder
- Feed Forward
- Self-Attention

Decoder
- Feed Forward
- Encoder-Decoder Attention
- Self-Attention
Residual Connections

- Each sub-layer has residual connection
Layer Normalization
What is 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”

• There is additive and dot-product attention
  • Dot-product attention is more efficient
Dot-Product Attention

\[ A(Q, K, V) = \text{softmax}(QK^T)V \]
Scaled Dot-Product Attention

\[ A(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \]
Scaled Dot-Product Attention
Scaled Dot-Product Attention: Vectors

• “Mapping a **query** and a set of **key-value pairs** to an output”
  • \( \text{query, key} \in \mathbb{R}^{d_k} \)
  • \( \text{value} \in \mathbb{R}^{d_v} \)

1. Compute dot product of query with all keys
2. Divide each by \( \sqrt{d_k} \)
3. Apply softmax to get weights of values
Scaled Dot-Product Attention: Vectors
Scaled Dot-Product Attention: Vectors

Input
Embedding
Queries
Keys
Values
Score

Thinking

Machines

x₁

q₁

k₁

v₁

q₁ • k₁ = 112

x₂

q₂

k₂

v₂

q₁ • k₂ = 96
Scaled Dot-Product Attention: Vectors

Input
Embedding
Queries
Keys
Values
Score
Divide by $\sqrt{d_k}$
Softmax

Thinking
$x_1$
$q_1$
$k_1$
$v_1$
$q_1 \cdot k_1 = 112$
14
0.88

Machines
$x_2$
$q_2$
$k_2$
$v_2$
$q_1 \cdot k_2 = 96$
12
0.12

MatMul
SoftMax
Mask (opt.)
Scale
MatMul

Q
K
V
Scaled Dot-Product Attention: Vectors
Scaled Dot-Product Attention: Matrix Form

\[ X W^Q = Q \]
\[ X W^K = K \]
\[ X W^V = V \]

\[ \text{MatMul} \]
\[ \text{SoftMax} \]
\[ \text{Mask (opt.)} \]
\[ \text{Scale} \]
\[ \text{MatMul} \]

\[ Q \]
\[ K \]
\[ V \]
Scaled Dot-Product Attention: Matrix Form

$$softmax\left(\frac{Q K^T}{\sqrt{d_k}}\right) \times V = Z$$
Decoder Attention: One step
Decoder Attention: Remainder

Decoding time step: 1 2 3 4 5 6

ENCODERS

K_{encdec}  V_{encdec}

Linear + Softmax

DECODERS

EMBEDDING WITH TIME SIGNAL

EMBEDDINGS

INPUT: Je suis étudiant

PREVIOUS OUTPUTS
Visualization of Attentions
What’s Missing from Self-Attention?

• Self-attention is just a weighted average of values
• Can we use different linear transformations like CNN filters?
Multi-Head Attention

$h = 8$
Multi-Head Attention
Multi-Head Attention: 8 Heads

Calculating attention separately in eight different attention heads

X

Thinking Machines

ATTENTION HEAD #0

Z₀

ATTENTION HEAD #1

Z₁

ATTENTION HEAD #7

Z₇
Multi-Head Attention: Concatenation

1) Concatenate all the attention heads

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

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

\[ Z \]
Multi-Head Attention: Details

• Each $Q, K, V$ matrices are randomly initialized
• To have similar computational cost as single-head attention, we use smaller $d_k = d_v = \frac{d_{model}}{h} = 64.$
Attention: Recap

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
Applications of Attention

1. Encoder-Decoder Attention
   • Query from previous decoder layer
   • Key, Value from encoder output

2. Encoder Self-Attention
   • Query, Key, Value from previous encoder layer

3. Decoder Self-Attention
   • Query, Key, Value from previous decoder layer
   • Prevent leftward information flow through masking
Masking for Decoder Self-Attention

- Decoder cannot look at subsequent positions
- Decoder attention should be masked
Applications of Attention
MODEL ARCHITECTURE
POSITION-WISE FEED-FORWARD NETWORKS
Position-wise Feed-Forward Networks

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

- Same linear transformation across different positions
- Different parameters for each layer
MODEL ARCHITECTURE
EMBEDDINGS AND SOFTMAX
Decoder Softmax

Which word in our vocabulary is associated with this index?

Get the index of the cell with the highest value (argmax)

\[ \log \text{probs} \]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[ \text{vocab}_{\text{size}} \]

\[ \text{am} \]

\[ 5 \]

\[ \log \text{its} \]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[ \text{vocab}_{\text{size}} \]

Softmax

Decoder stack output

Linear
MODEL ARCHITECTURE
POSITIONAL ENCODING
Rationale

• Transformer uses neither recurrence nor convolution
• Need to inject positional information

→ Add Positional Encoding (Learned or Fixed)

\[
P E_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})
\]
\[
P E_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})
\]
Positional Encoding
WHY SELF-ATTENTION
Comparison with Previous Methods

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>
Interpretability
BLEU Scores

**English German Translation quality**

- GNMT (RNN)
- Conv2S (CNN)
- SliceNet (CNN)
- Transformer

**English French Translation Quality**

- GNMT (RNN)
- Conv2S (CNN)
- Transformer
## BLEU Scores and Training Cost

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 \cdot 10^{20}</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>2.3 \cdot 10^{19}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
<td>9.6 \cdot 10^{18}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>2.0 \cdot 10^{19}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td></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><strong>28.4</strong></td>
<td>2.3 \cdot 10^{19}</td>
</tr>
</tbody>
</table>
## Variations

<table>
<thead>
<tr>
<th></th>
<th>$N$</th>
<th>$d_{\text{model}}$</th>
<th>$d_{ff}$</th>
<th>$h$</th>
<th>$d_{k}$</th>
<th>$d_{v}$</th>
<th>$P_{\text{drop}}$</th>
<th>$\epsilon_{ls}$</th>
<th>train steps</th>
<th>PPL (dev)</th>
<th>BLEU (dev)</th>
<th>params $\times 10^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>6</td>
<td>512</td>
<td>2048</td>
<td>8</td>
<td>64</td>
<td>64</td>
<td>0.1</td>
<td>0.1</td>
<td>100K</td>
<td>4.92</td>
<td>25.8</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(A)</td>
<td>1</td>
<td>512</td>
<td>512</td>
<td></td>
<td></td>
<td>5.29</td>
<td>24.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>128</td>
<td>128</td>
<td></td>
<td></td>
<td>5.00</td>
<td>25.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>32</td>
<td>32</td>
<td></td>
<td></td>
<td>4.91</td>
<td>25.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>16</td>
<td>16</td>
<td></td>
<td></td>
<td>5.01</td>
<td>25.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(B)</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.16</td>
<td>25.1</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.01</td>
<td>25.4</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>(C)</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.11</td>
<td>23.7</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.19</td>
<td>25.3</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.88</td>
<td>25.5</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.75</td>
<td>24.5</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.66</td>
<td>26.0</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024</td>
<td></td>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.12</td>
<td>25.4</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4096</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.75</td>
<td>26.2</td>
<td>90</td>
</tr>
<tr>
<td>(D)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.77</td>
<td>24.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.95</td>
<td>25.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.67</td>
<td>25.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.47</td>
<td>25.7</td>
<td></td>
</tr>
<tr>
<td>(E)</td>
<td></td>
<td>positional embedding instead of sinusoids</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.92</td>
<td>25.7</td>
<td></td>
</tr>
<tr>
<td>big</td>
<td>6</td>
<td>1024</td>
<td>4096</td>
<td>16</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td>300K</td>
<td>4.33</td>
<td>26.4</td>
<td>213</td>
</tr>
</tbody>
</table>
# Generalization to other tasks

<table>
<thead>
<tr>
<th>Parser</th>
<th>Training</th>
<th>WSJ 23 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vinyals &amp; Kaiser el al. (2014) [37]</td>
<td>WSJ only, discriminative</td>
<td>88.3</td>
</tr>
<tr>
<td>Petrov et al. (2006) [29]</td>
<td>WSJ only, discriminative</td>
<td>90.4</td>
</tr>
<tr>
<td>Zhu et al. (2013) [40]</td>
<td>WSJ only, discriminative</td>
<td>90.4</td>
</tr>
<tr>
<td>Dyer et al. (2016) [8]</td>
<td>WSJ only, discriminative</td>
<td>91.7</td>
</tr>
<tr>
<td>Transformer (4 layers)</td>
<td>WSJ only, discriminative</td>
<td>91.3</td>
</tr>
<tr>
<td>Zhu et al. (2013) [40]</td>
<td>semi-supervised</td>
<td>91.3</td>
</tr>
<tr>
<td>McClosky et al. (2006) [26]</td>
<td>semi-supervised</td>
<td>92.1</td>
</tr>
<tr>
<td>Vinyals &amp; Kaiser el al. (2014) [37]</td>
<td>semi-supervised</td>
<td>92.1</td>
</tr>
<tr>
<td>Transformer (4 layers)</td>
<td>semi-supervised</td>
<td>92.7</td>
</tr>
<tr>
<td>Luong et al. (2015) [23]</td>
<td>multi-task</td>
<td>93.0</td>
</tr>
<tr>
<td>Dyer et al. (2016) [8]</td>
<td>generative</td>
<td>93.3</td>
</tr>
</tbody>
</table>
THANK YOU FOR YOUR ATTENTION
Resources

Posts

• *The Illustrated Transformer* by Jay Alammar
• *Paper Dissected: “Attention is All You Need” Explained* by Keita Kurita
• *The Transformer – Attention is all you need* by Michal Chromiak
• *Transformer* by Jakob Uszkoreit (Google AI Blog)

Video

• *Tensor2Tensor Transformers* by Lukasz Kaiser
Resources

Code
• Tensor2Tensor (Official)
• Tensor2Tensor Colab (Official)
• Annotated Transformer (Harvard NLP)

Slides
• End-to-End AI