Natural Language Processing with Deep Learning
CS224N/Ling284

Anna Goldie
Lecture 8: Transformers

Adapted from slides by Anna Goldie, John Hewitt
Lecture Plan

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers
Lecture Plan

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers
Transformers: Is Attention All We Need?

• Last lecture, we learned that attention dramatically improves the performance of recurrent neural networks.
• Today, we will take this one step further and ask Is Attention All We Need?
Transformers: Is Attention All We Need?

- Last lecture, we learned that attention dramatically improves the performance of recurrent neural networks.
- Today, we will take this one step further and ask Is Attention All We Need?
- Spoiler: Not Quite!
Transformers Have Revolutionized the Field of NLP

By the end of this lecture, you will deeply understand the neural architecture that underpins virtually every state-of-the-art NLP model today!

[Diagram of Transformer architecture, courtesy of Paramount Pictures]
Great Results with Transformers: Machine Translation

First, Machine Translation results from the original Transformers paper!

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU EN-DE</th>
<th>BLEU EN-FR</th>
<th>Training Cost (FLOPs) EN-DE</th>
<th>Training Cost (FLOPs) EN-FR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
<td></td>
<td>1.0 \cdot 10^{20}</td>
<td></td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td></td>
<td>39.2</td>
<td>2.3 \cdot 10^{19}</td>
<td>1.4 \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
<td>39.92</td>
<td>2.3 \cdot 10^{19}</td>
<td>1.4 \cdot 10^{20}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
<td>40.46</td>
<td>9.6 \cdot 10^{18}</td>
<td>1.5 \cdot 10^{20}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>40.56</td>
<td>2.0 \cdot 10^{19}</td>
<td>1.2 \cdot 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td>40.4</td>
<td>8.0 \cdot 10^{20}</td>
<td></td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>41.16</td>
<td>1.8 \cdot 10^{20}</td>
<td>1.1 \cdot 10^{21}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td><strong>41.29</strong></td>
<td>7.7 \cdot 10^{19}</td>
<td>1.2 \cdot 10^{21}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
<td><strong>3.3 \cdot 10^{18}</strong></td>
<td></td>
</tr>
<tr>
<td>Transformer (big)</td>
<td><strong>28.4</strong></td>
<td><strong>41.8</strong></td>
<td>2.3 \cdot 10^{19}</td>
<td></td>
</tr>
</tbody>
</table>

[Test sets: WMT 2014 English-German and English-French] [Vaswani et al., 2017]
Great Results with Transformers: SuperGLUE

SuperGLUE is a suite of challenging NLP tasks, including question-answering, word sense disambiguation, coreference resolution, and natural language inference.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Model</th>
<th>URL</th>
<th>Score</th>
<th>BoolQ</th>
<th>CB</th>
<th>COPA</th>
<th>MultiRC</th>
<th>ReCoRD</th>
<th>RTE</th>
<th>WIC</th>
<th>WSC</th>
<th>AX-b</th>
<th>AX-g</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JDExplore d-team</td>
<td>Vega v2</td>
<td>![]</td>
<td>91.3</td>
<td>90.5</td>
<td>98.6</td>
<td>99.2</td>
<td>99.4</td>
<td>88.2/62.4</td>
<td>94.4/93.9</td>
<td>96.0</td>
<td>77.4</td>
<td>98.6</td>
<td>-0.4</td>
</tr>
<tr>
<td>2</td>
<td>Liam Fedus</td>
<td>ST-MoE-32B</td>
<td>![]</td>
<td>91.2</td>
<td>92.4</td>
<td>96.9</td>
<td>98.0</td>
<td>99.2</td>
<td>89.6/65.8</td>
<td>95.1/94.4</td>
<td>93.5</td>
<td>77.7</td>
<td>96.6</td>
<td>72.3</td>
</tr>
<tr>
<td>3</td>
<td>Microsoft Alexander v-team</td>
<td>Turing NLR v6</td>
<td>![]</td>
<td>90.9</td>
<td>92.0</td>
<td>95.9</td>
<td>97.6</td>
<td>98.2</td>
<td>88.4/63.0</td>
<td>96.4/95.9</td>
<td>94.1</td>
<td>77.1</td>
<td>97.3</td>
<td>67.8</td>
</tr>
<tr>
<td>4</td>
<td>ERNIE Team - Baidu</td>
<td>ERNIE 3.0</td>
<td>![]</td>
<td>90.6</td>
<td>91.0</td>
<td>98.6</td>
<td>99.2</td>
<td>97.4</td>
<td>88.6/63.2</td>
<td>94.7/94.2</td>
<td>92.6</td>
<td>77.4</td>
<td>97.3</td>
<td>68.6</td>
</tr>
<tr>
<td>5</td>
<td>Yi Tay</td>
<td>PaLM 540B</td>
<td>![]</td>
<td>90.4</td>
<td>91.9</td>
<td>94.4</td>
<td>96.0</td>
<td>99.0</td>
<td>88.7/63.6</td>
<td>94.2/93.3</td>
<td>94.1</td>
<td>77.4</td>
<td>95.9</td>
<td>72.9</td>
</tr>
<tr>
<td>6</td>
<td>Zirui Wang</td>
<td>T5 + UDG, Single Model (Google Brain)</td>
<td>![]</td>
<td>90.4</td>
<td>91.4</td>
<td>95.8</td>
<td>97.6</td>
<td>98.0</td>
<td>88.3/63.0</td>
<td>94.2/93.5</td>
<td>93.0</td>
<td>77.9</td>
<td>96.6</td>
<td>69.1</td>
</tr>
<tr>
<td>7</td>
<td>DeBERTa Team - Microsoft</td>
<td>DeBERTa / TuringNLRv4</td>
<td>![]</td>
<td>90.3</td>
<td>90.4</td>
<td>95.7</td>
<td>97.6</td>
<td>98.4</td>
<td>88.2/63.7</td>
<td>94.5/94.1</td>
<td>93.2</td>
<td>77.5</td>
<td>95.9</td>
<td>66.7</td>
</tr>
<tr>
<td>8</td>
<td>SuperGLUE Human Baselines</td>
<td>SuperGLUE Human Baselines</td>
<td>![]</td>
<td>89.8</td>
<td>89.0</td>
<td>95.8</td>
<td>98.9</td>
<td>100.0</td>
<td>81.8/51.9</td>
<td>91.7/91.3</td>
<td>93.6</td>
<td>80.0</td>
<td>100.0</td>
<td>76.6</td>
</tr>
<tr>
<td>9</td>
<td>T5 Team - Google</td>
<td>T5</td>
<td>![]</td>
<td>89.3</td>
<td>91.2</td>
<td>93.9</td>
<td>96.8</td>
<td>94.8</td>
<td>88.1/63.3</td>
<td>94.1/93.4</td>
<td>92.5</td>
<td>76.9</td>
<td>93.8</td>
<td>65.6</td>
</tr>
<tr>
<td>10</td>
<td>SPoT Team - Google</td>
<td>Frozen T5 1.1 + SPoT</td>
<td>![]</td>
<td>89.2</td>
<td>91.1</td>
<td>95.8</td>
<td>97.6</td>
<td>95.6</td>
<td>87.9/61.9</td>
<td>93.3/92.4</td>
<td>92.9</td>
<td>75.8</td>
<td>93.8</td>
<td>66.9</td>
</tr>
</tbody>
</table>

[Test sets: SuperGLUE Leaderboard Version: 2.0] [Wang et al., 2019]
Great Results with Transformers: Rise of Large Language Models!

Today, Transformer-based models dominate LMSYS Chatbot Arena Leaderboard!

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>Arena Elo</th>
<th>95% CI</th>
<th>Votes</th>
<th>Organization</th>
<th>License</th>
<th>Knowledge Cutoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPT-4-Turbo-2024-04-09</td>
<td>1258</td>
<td>+4/-4</td>
<td>26444</td>
<td>OpenAI</td>
<td>Proprietary</td>
<td>2023/12</td>
</tr>
<tr>
<td>1</td>
<td>GPT-4-1106-preview</td>
<td>1253</td>
<td>+3/-3</td>
<td>68353</td>
<td>OpenAI</td>
<td>Proprietary</td>
<td>2023/4</td>
</tr>
<tr>
<td>1</td>
<td>Claude_3_Opus</td>
<td>1251</td>
<td>+3/-3</td>
<td>71500</td>
<td>Anthropic</td>
<td>Proprietary</td>
<td>2023/8</td>
</tr>
<tr>
<td>2</td>
<td>Gemini_1.5_Pro_API-0409-Preview</td>
<td>1249</td>
<td>+4/-5</td>
<td>22211</td>
<td>Google</td>
<td>Proprietary</td>
<td>2023/11</td>
</tr>
<tr>
<td>3</td>
<td>GPT-4-0125-preview</td>
<td>1248</td>
<td>+2/-3</td>
<td>58959</td>
<td>OpenAI</td>
<td>Proprietary</td>
<td>2023/12</td>
</tr>
<tr>
<td>6</td>
<td>Meta_Llama_3_70b_Instruct</td>
<td>1213</td>
<td>+4/-6</td>
<td>15809</td>
<td>Meta</td>
<td>Llama 3 Community</td>
<td>2023/12</td>
</tr>
<tr>
<td>6</td>
<td>Bard (Gemini Pro)</td>
<td>1208</td>
<td>+7/-6</td>
<td>12435</td>
<td>Google</td>
<td>Proprietary</td>
<td>Online</td>
</tr>
<tr>
<td>7</td>
<td>Claude_3_Sonnet</td>
<td>1201</td>
<td>+4/-2</td>
<td>73414</td>
<td>Anthropic</td>
<td>Proprietary</td>
<td>2023/8</td>
</tr>
</tbody>
</table>

[Chiang et al., 2024]
Transformers Even Show Promise Outside of NLP
Transformers Even Show Promise Outside of NLP

Protein Folding

[Jumper et al. 2021] aka AlphaFold2!
Transformers Even Show Promise Outside of NLP

### Protein Folding

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ours-JFT (ViT-H/14)</th>
<th>Ours-JFT (ViT-L/16)</th>
<th>Ours-121k (ViT-L/16)</th>
<th>Bit-L (ResNet152x4)</th>
<th>Noisy Student (EfficientNet-L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td>88.55 ± 0.04</td>
<td>87.79 ± 0.03</td>
<td>85.30 ± 0.02</td>
<td>87.54 ± 0.02</td>
<td>88.4/88.5</td>
</tr>
<tr>
<td>ImageNet Ranzl</td>
<td>90.72 ± 0.05</td>
<td>90.54 ± 0.03</td>
<td>89.62 ± 0.05</td>
<td>90.54</td>
<td>90.55</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>99.50 ± 0.06</td>
<td>99.42 ± 0.03</td>
<td>99.15 ± 0.03</td>
<td>99.27 ± 0.06</td>
<td></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>94.55 ± 0.04</td>
<td>93.90 ± 0.01</td>
<td>93.25 ± 0.05</td>
<td>93.51 ± 0.06</td>
<td></td>
</tr>
<tr>
<td>Oxford-JFT Pets</td>
<td>97.56 ± 0.02</td>
<td>97.32 ± 0.11</td>
<td>94.67 ± 0.15</td>
<td>96.62 ± 0.23</td>
<td></td>
</tr>
<tr>
<td>Oxford Flowers-102</td>
<td>99.69 ± 0.02</td>
<td>99.74 ± 0.08</td>
<td>99.61 ± 0.02</td>
<td>99.63 ± 0.03</td>
<td></td>
</tr>
<tr>
<td>VTAB (19 tasks)</td>
<td>77.63 ± 0.33</td>
<td>76.29 ± 0.48</td>
<td>72.73 ± 0.21</td>
<td>75.29 ± 1.70</td>
<td></td>
</tr>
<tr>
<td>TPU/s3-core-days</td>
<td>2.0k</td>
<td>0.65k</td>
<td>0.25k</td>
<td>9.9k</td>
<td>12.3k</td>
</tr>
</tbody>
</table>

[Jumper et al. 2021] aka AlphaFold2!
Transformers Even Show Promise Outside of NLP

Protein Folding

Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

<table>
<thead>
<tr>
<th></th>
<th>Ours-JFT (ViT-H/14)</th>
<th>Ours-JFT (ViT-L/16)</th>
<th>Ours-L21k (ViT-L/16)</th>
<th>BIT-L (ResNet152x4)</th>
<th>Noisy Student (EfficientNet-L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td>88.55 ± 0.01</td>
<td>87.76 ± 0.03</td>
<td>85.30 ± 0.02</td>
<td>87.54 ± 0.02</td>
<td>88.4 ± 0.5</td>
</tr>
<tr>
<td>ImageNet Rand.</td>
<td>90.72 ± 0.05</td>
<td>90.54 ± 0.01</td>
<td>88.62 ± 0.05</td>
<td>90.54</td>
<td>90.55</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>99.50 ± 0.06</td>
<td>99.42 ± 0.03</td>
<td>99.15 ± 0.03</td>
<td>99.37 ± 0.02</td>
<td>—</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>94.55 ± 0.01</td>
<td>93.90 ± 0.01</td>
<td>93.25 ± 0.05</td>
<td>93.51 ± 0.06</td>
<td>—</td>
</tr>
<tr>
<td>Oxford-JFT Pts.</td>
<td>97.56 ± 0.03</td>
<td>97.32 ± 0.11</td>
<td>94.67 ± 0.15</td>
<td>96.62 ± 0.23</td>
<td>—</td>
</tr>
<tr>
<td>Oxford Flowers-102</td>
<td>98.68 ± 0.02</td>
<td>99.74 ± 0.09</td>
<td>99.61 ± 0.02</td>
<td>99.63 ± 0.03</td>
<td>—</td>
</tr>
<tr>
<td>VTAB (19 tasks)</td>
<td>77.63 ± 0.23</td>
<td>76.29 ± 0.61</td>
<td>72.73 ± 0.21</td>
<td>76.29 ± 1.70</td>
<td>—</td>
</tr>
<tr>
<td>TPU/v1-core-days</td>
<td>2.0k</td>
<td>0.65k</td>
<td>0.22k</td>
<td>9.9k</td>
<td>12.5k</td>
</tr>
</tbody>
</table>

ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

---

[Jumper et al. 2021] aka AlphaFold2!
Scaling Laws: Are Transformers All We Need?

- With Transformers, language modeling performance improves smoothly as we increase model size, training data, and compute resources in tandem.
- This power-law relationship has been observed over multiple orders of magnitude with no sign of slowing!
- If we keep scaling up these models (with no change to the architecture), could they eventually match or exceed human-level performance?

[Kaplan et al., 2020]
Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers
As of last lecture: recurrent models for (most) NLP!

- Circa 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM: (for example, the source sentence in a translation)

- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.

- Use attention to allow flexible access to memory
Why Move Beyond Recurrence?
Motivation for Transformer Architecture

The Transformers authors had 3 desirata when designing this architecture:

1. Minimize (or at least not increase) computational complexity per layer.
2. Minimize path length between any pair of words to facilitate learning of long-range dependencies.
3. Maximize the amount of computation that can be parallelized.

[Vaswani et al., 2017]
1. Transformer Motivation: Computational Complexity Per Layer

When sequence length \( n \ll \text{representation dimension} \cdot d \), complexity per layer is lower for a Transformer compared to the recurrent models we’ve learned about so far.

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_e(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 the Transformer paper.

[Vaswani et al., 2017]
2. Transformer Motivation: Minimize Linear Interaction Distance

- RNNs are unrolled “left-to-right”.
- It encodes linear locality: a useful heuristic!
  - Nearby words often affect each other’s meanings

- **Problem:** RNNs take $O(\text{sequence length})$ steps for distant word pairs to interact.
2. Transformer Motivation: Minimize Linear Interaction Distance

• $O(\text{sequence length})$ steps for distant word pairs to interact means:
  • Hard to learn long-distance dependencies (because gradient problems!)
  • Linear order of words is “baked in”; we already know sequential structure
doesn't tell the whole story...

The chef who …

Info of chef has gone through $O(\text{sequence length})$ many layers!
3. Transformer Motivation: Maximize **Parallelizability**

- Forward and backward passes have $O(\text{seq length})$ unparallelizable operations
  - GPUs (and TPUs) can perform many independent computations at once!
  - But future RNN hidden states can’t be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!
  - Particularly problematic as sequence length increases, as we can no longer batch many examples together due to memory limitations

Numbers indicate min # of steps before a state can be computed
High-Level Architecture: Transformer is all about (Self) Attention

• To recap, attention treats each word’s representation as a query to access and incorporate information from a set of values.
  • Last lecture, we saw attention from the decoder to the encoder in a recurrent sequence-to-sequence model
  • Self-attention is encoder-encoder (or decoder-decoder) attention where each word attends to each other word within the input (or output).

All words attend to all words in previous layer; most arrows here are omitted
Computational Dependencies for Recurrence vs. Attention

RNN-Based Encoder-Decoder Model with Attention

Transformer-Based Encoder-Decoder Model
Computational Dependencies for Recurrence vs. Attention

RNN-Based Encoder-Decoder Model with Attention

Transformer Advantages:
- Number of unparallelizable operations does not increase with sequence length.
- Each "word" interacts with each other, so maximum interaction distance is $O(1)$. 

Transformer-Based Encoder-Decoder Model
Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers
The Transformer Encoder-Decoder [Vaswani et al., 2017]

In this section, you will learn exactly how the Transformer architecture works:
• First, we will talk about the Encoder!
• Next, we will go through the Decoder (which is quite similar)!
Encoder: Self-Attention

Self-Attention is the core building block of Transformer, so let's first focus on that!
Intuition for Attention Mechanism

- Let's think of attention as a "fuzzy" or approximate hashtable:
  - To look up a value, we compare a query against keys in a table.
  - In a hashtable (shown on the bottom left):
    - Each query (hash) maps to exactly one key-value pair.
  - In (self-)attention (shown on the bottom right):
    - Each query matches each key to varying degrees.
    - We return a sum of values weighted by the query-key match.
Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word $x_i$, calculate its query, key, and value.
  \[
  q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i
  \]

- Step 2: Calculate attention score between query and keys.
  \[
  e_{ij} = q_i \cdot k_j
  \]

- Step 3: Take the softmax to normalize attention scores.
  \[
  \alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}
  \]

- Step 4: Take a weighted sum of values.
  \[
  \text{Output}_i = \sum_j \alpha_{ij} v_j
  \]
Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in $X$, calculate queries, keys, and values.
  \[
  Q = XW^Q \quad K = XW^K \quad V = XW^V
  \]

- Step 2: Calculate attention scores between query and keys.
  \[
  E = QK^T
  \]

- Step 3: Take the softmax to normalize attention scores.
  \[
  A = \text{softmax}(E)
  \]

- Step 4: Take a weighted sum of values.
  \[
  \text{Output} = \text{softmax}(QK^T)V
  \]

  \[
  \text{Output} = AV
  \]
What We Have So Far: (Encoder) Self-Attention!
But attention isn't quite all you need!

- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

$$m_i = MLP(output_i)$$

$$= W_2 \ast \text{ReLU}(W_1 \times output_i + b_1) + b_2$$
But how do we make this work for deep networks?

Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention
Training Trick #1: Residual Connections [He et al., 2016]

- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

\[ x_\ell = F(x_{\ell-1}) + x_{\ell-1} \]

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.

Residual connections are also thought to smooth the loss landscape and make training easier!
Training Trick #2: Layer Normalization [Ba et al., 2016]

- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.
- **Solution:** Reduce variation by normalizing to zero mean and standard deviation of one within each layer.

Mean: $\mu^\ell = \frac{1}{H} \sum_{i=1}^{H} a_i^\ell$

Standard Deviation: $\sigma^\ell = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^\ell - \mu^\ell)^2}$

$$x'^\ell = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$
Training Trick #2: Layer Normalization [Ba et al., 2016]

An Example of How LayerNorm Works (Image by Bala Priya C, Pinecone)

Mean: \( \mu^l = \frac{1}{H} \sum_{i=1}^{H} \alpha_i^l \)  
Standard Deviation: \( \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (\alpha_i^l - \mu^l)^2} \)

\( x^{l'} = \frac{x^l - \mu^l}{\sigma^l + \epsilon} \)
Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality $d_k$

Quick Statistics Review:
- Mean of sum = sum of means = $d_k \times 0 = 0$
- Variance of sum = sum of variances = $d_k \times 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!

Updated Self-Attention Equation:

$$\text{Output} = \text{softmax}(QK^T / \sqrt{d_k})V$$
Major issue!

• We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
• Consider this sentence:
  • "Man eats small dinosaur."

\[ \text{Output} = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \]
Major issue!

- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
  - "Man eats small dinosaur."
- Wait a minute, order doesn't impact the network at all!
- This seems wrong given that word order does have meaning in many languages, including English!
Solution: Inject Order Information through Positional Encodings!
Fixing the first self-attention problem: sequence order

- Since self-attention doesn’t build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each sequence index as a vector
  \[ p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2, \ldots, T\} \] are position vectors
- Don’t worry about what the \( p_i \) are made of yet!
- Easy to incorporate this info into our self-attention block: just add the \( p_i \) to our inputs!
- Let \( \tilde{v}_i, \tilde{k}_i, \tilde{q}_i \) be our old values, keys, and queries.

\[
\begin{align*}
  v_i &= \tilde{v}_i + p_i \\
  q_i &= \tilde{q}_i + p_i \\
  k_i &= \tilde{k}_i + p_i 
\end{align*}
\]

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...
Position representation vectors through sinusoids (original)

- **Sinusoidal position representations**: concatenate sinusoidal functions of varying periods:

\[
p_i = \begin{bmatrix}
\sin(i/10000^{2*1/d}) \\
\cos(i/10000^{2*1/d}) \\
\vdots \\
\sin(i/10000^{2*d/2/d}) \\
\cos(i/10000^{2*d/2/d})
\end{bmatrix}
\]

- Pros:
  - Periodicity indicates that maybe “absolute position” isn’t as important
  - Maybe can extrapolate to longer sequences as periods restart
- Cons:
  - Not learnable; also the extrapolation doesn’t really work
Extension: Self-Attention w/ Relative Position Encodings

**Key Insight:** The most salient position information is the relationship (e.g. “cat” is the word before “eat”) between words, rather than their absolute position (e.g. “cat” is word 2).

**Original Self-Attention Output:**

\[
z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V)
\]

where

\[
\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}
\]

\[
e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}
\]

**Relation-Aware Self-Attention Output:**

\[
z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a^V_{ij})
\]

where

\[
\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}
\]

\[
e_{ij} = \frac{x_i W^Q(x_j W^K + a^K_{ij})^T}{\sqrt{d_z}}
\]

\[
a^K_{ij} = w^K_{\text{clip}(j-i,k)}
\]

\[
a^V_{ij} = w^V_{\text{clip}(j-i,k)}
\]

\[
\text{clip}(x, k) = \max(-k, \min(k, x))
\]

**Table and Equations From [Shaw et al., 2018]**
Multi-Headed Self-Attention: k heads are better than 1!

- **High-Level Idea:** Let’s perform self-attention multiple times in parallel and combine the results.

[Vaswani et al. 2017]
The Transformer Encoder: **Multi-headed Self-Attention**

- What if we want to look in multiple places in the sentence at once?
  - For word $i$, self-attention “looks” where $x_i^T Q^T K x_j$ is high, but maybe we want to focus on different $j$ for different reasons?
- We’ll define **multiple attention “heads”** through multiple Q,K,V matrices

- Let $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times d_h}$, where $h$ is the number of attention heads, and $\ell$ ranges from 1 to $h$.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}(XQ_\ell K_\ell^T X^T) \ast XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
  - Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to “look” at different things, and construct value vectors differently.

Credit to https://jalammar.github.io/illustrated-transformer/
Yay, we've completed the Encoder! Time for the Decoder...
Decoder: Masked Multi-Head Self-Attention

• **Problem:** How do we keep the decoder from “cheating”? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
Decoder: Masked Multi-Head Self-Attention

• **Problem:** How do we keep the decoder from “cheating”? If we have a language modeling objective, can't the network just look ahead and "see" the answer?

• **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.
Masking the future in self-attention

- To use self-attention in decoders, we need to ensure we can’t peek at the future.

- At every timestep, we could change the set of keys and queries to include only past words. (Inefficient!)

- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

\[
e_{ij} = \begin{cases} q_i^\top k_j, & j < i \\ -\infty, & j \geq i \end{cases}
\]
Decoder: Masked Multi-Headed Self-Attention
Encoder-Decoder Attention

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \ldots, h_T$ be output vectors from the Transformer encoder; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer decoder, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the encoder (like a memory):
  - $k_i = K h_i, v_i = V h_i$.
- And the queries are drawn from the decoder, $q_i = Q z_i$.  

\[ x_i \in \mathbb{R}^d \]
Decoder: Finishing touches!
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!
Recap of Transformer Architecture
Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers
What would we like to fix about the Transformer?

- **Quadratic compute in self-attention (today):**
  - Computing all pairs of interactions means our computation grows \textit{quadratically} with the sequence length!
  - For recurrent models, it only grew linearly!

- **Position representations:**
  - Are simple absolute indices the best we can do to represent position?
  - As we learned: Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]
  - Rotary Embeddings [Su et al., 2021]
Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **Linformer** [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys
Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*

- For example, **BigBird** [Zaheer et al., 2021]

**Key idea:** replace all-pairs interactions with a family of other interactions, like *local windows, looking at everything*, and *random interactions.*
"Surprisingly, we find that most modifications do not meaningfully improve performance."
Parting remarks

• Yay, you now understand Transformers!
• Next class, we will see how pre-training can take performance to the next level!
• Good luck on assignment 4!
• Remember to work on your project proposal!