Transformers and Pre-trained Language Models

Danqi Chen

Princeton Language and Intelligence
Princeton University

July 15th, 2024

(Many slides are adapted from Princeton COS484 and Stanford CS224N course materials)
Our research focuses on **training**, **adapting** and **understanding** large language models.

See more at [https://www.cs.princeton.edu/~danqic/](https://www.cs.princeton.edu/~danqic/)
Lecture plan

Part I.  **Transformers**
Focus: innovations and key designs in neural architectures

30min coffee break

Part II.  **Pre-trained language models**
Focus: training objectives & data, downstream adaptations
Lecture plan

• **Fundamentals (70%)** - I will walk through the most important ideas in NLP and LLMs in the past 5+ years (Transformers, pre-training, in-context learning, RLHF, ...)

• **How do these ideas evolve and lead to state-of-the-art models? (15%)**
  - I will highlight recent improvements and developments

• **Cutting-edge research topics (15%)** - What research topics do we study in 2024? I will briefly discuss some of the works from my research group too
Part I. Transformers
Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar†
Google Research
nikip@google.com

Jakob Uszkoreit†
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez†
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser†
Google Brain
lukaszkaiser@google.com

Illia Polosukhin†‡
illia.polosukhin@gmail.com

(Vaswani et al., 2017)
What is attention?

**Neural Machine Translation by Jointly Learning to Align and Translate**

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho    Yoshua Bengio*
Université de Montréal

(Bahdanau et al., 2015)

- Attention is a technique to address the “bottleneck” issue in the seq2seq model, originally designed for machine translation
What is attention?

Attention is a technique to address the “bottleneck” issue in the seq2seq model, originally designed for machine translation.

Key idea: At each time step during decoding, focus on only a particular part of the source sentence.

- This depends on decoder’s current hidden state $h_{dec}^{t}$
- Usually implemented as a probability distribution over the hidden states of the encoder ($h_{enc}^{i}$)
Attention for seq2seq models
Attention learns the notion of **alignment**

“Which source words are more relevant to the current target word?”

Attention for seq2seq models

\[ h_{1}^{enc}, \ldots, h_{n}^{enc} \text{ and } h_{t}^{dec} \] are hidden states from encoder and decoder RNNs

- Encoder hidden states: \( h_{1}^{enc}, \ldots, h_{n}^{enc} \) (n: # of words in source sentence)
- Decoder hidden state at time \( t \): \( h_{t}^{dec} \)
- Attention scores:

\[ e^{t} = [g(h_{1}^{enc}, h_{t}^{dec}), \ldots, g(h_{n}^{enc}, h_{t}^{dec})] \in \mathbb{R}^{n} \]
- Attention distribution:

\[ \alpha^{t} = \text{softmax}(e^{t}) \in \mathbb{R}^{n} \]
- Weighted sum of encoder hidden states:

\[ o_{t} = \sum_{i=1}^{n} \alpha_{i}^{t} h_{i}^{enc} \in \mathbb{R}^{h} \]

Combine \( o_{t} \) and \( h_{t}^{dec} \) to predict next word
Attention as a soft, averaging lookup table

We can think of **attention** as performing fuzzy lookup in a **key-value store**.

**Lookup table**: a table of keys that map to values. The query matches one of the keys, returning its value.

**Attention**: The query matches to all keys softly to a weight between 0 and 1. The keys’ values are multiplied by the weights and summed.

(In the case of NMT, key = value)
Transformer encoder-decoder

- Transformer encoder + Transformer decoder: a replacement for seq2seq + attention based on RNNs
- First designed and experimented on NMT

(Vaswani et al., 2017)

Transformers (both encoders and decoders) have become the default neural architectures in modeling languages!
Transformer encoder-decoder

- Transformer encoder = a stack of encoder layers
- Transformer decoder = a stack of decoder layers

**Transformer encoder**: BERT, RoBERTa, ELECTRA

**Transformer decoder**: GPT-n, ChatGPT, Gemini, Claude, LLaMA, Mistral, …

**Transformer encoder-decoder**: T5, BART

- Key innovation: self-attention, multi-head
- Transformers don’t have any recurrence structures!

\[ h_t = f(h_{t-1}, x_t) \in \mathbb{R}^h \]

(Vaswani et al., 2017)
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

The Annotated Transformer

https://nlp.seas.harvard.edu/annotated-transformer/
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
General form of attention

• A more general form: use a set of keys and values \((k_1, v_1), \ldots, (k_n, v_n)\), \(k_i \in \mathbb{R}^{d_k}, v_i \in \mathbb{R}^{d_v}\), keys are used to compute the attention scores and values are used to compute the output vector

• Attention always involves the following steps:
  • Computing the attention scores \(e = g(q, k_i) \in \mathbb{R}^n\)
  • Taking softmax to get attention distribution \(\alpha\):
    \[
    \alpha = \text{softmax}(e) \in \mathbb{R}^n
    \]
  • Using attention distribution to take weighted sum of values:
    \[
    o = \sum_{i=1}^{n} \alpha_i v_i \in \mathbb{R}^{d_v}
    \]
Self-attention

• In NMT, query = decoder’s hidden state, keys = values = encoder’s hidden states

• Self-attention = attention from the sequence to itself

• Self-attention: let’s use each word in a sequence as the query, and all other words in the sequence as keys and values.

https://jalammar.github.io/illustrated-transformer/
Self-attention

Step #1: Transform each input vector into three vectors: query, key, and value vectors

\[ q_i = x_i W^Q \in \mathbb{R}^{d_q} \]
\[ k_i = x_i W^K \in \mathbb{R}^{d_k} \]
\[ v_i = x_i W^V \in \mathbb{R}^{d_v} \]

Note that we use row vectors here;
It is also common to write
\[ q_i = W^Q x_i \in \mathbb{R}^{d_q} \]
for \( x_i = \) a column vector

https://jalammar.github.io/illustrated-transformer/
Self-attention

Step #2: Compute pairwise similarities between keys and queries; normalize with softmax

For each $q_i$, compute attention scores and attention distribution:

$$\alpha_{i,j} = \text{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)$$

aka. “scaled dot product”

It must be $d_q = d_k$ in this case

Q. Why scaled dot product?

To avoid the dot product to become too large for larger $d_k$; scaling the dot product by $\frac{1}{\sqrt{d_k}}$ is easier for optimization
Self-attention

Step #3: Compute output for each input as weighted sum of values

$$h_i = \sum_{j=1}^{n} \alpha_{i,j} v_j \in \mathbb{R}^{d_v}$$

https://jalammar.github.io/illustrated-transformer/
What would be the output vector for the word “Thinking” approximately?

(A) $0.5v_1 + 0.5v_2$

(B) $0.54v_1 + 0.46v_2$

(C) $0.88v_1 + 0.12v_2$

(D) $0.12v_1 + 0.88v_2$

(C) is correct.

https://jalammar.github.io/illustrated-transformer/
Self-attention: matrix notations

\[ X \in \mathbb{R}^{n \times d_{in}} \text{ (n = input length)} \]

\[ Q = XW^Q, K = XW^K, V = XW^V \]

where \( W^Q \in \mathbb{R}^{d_{in} \times d_q}, W^K \in \mathbb{R}^{d_{in} \times d_k}, W^V \in \mathbb{R}^{d_{in} \times d_v} \)

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

Q: What is this softmax operation?
Multi-head attention

“The Beast with Many Heads”

- It is better to use multiple attention functions instead of one!
  - Each attention function (“head”) can focus on different positions.
- It gives the attention layer multiple “representation subspaces”

https://jalammar.github.io/illustrated-transformer/
Finally, we just concatenate all the heads and apply an output projection matrix.

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

- In practice, we use a reduced dimension for each head.
  \[W_i^Q \in \mathbb{R}^{d_{\text{in}} \times d_q}, W_i^K \in \mathbb{R}^{d_{\text{in}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{in}} \times d_v}\]
  \[d_q = d_k = d_v = d/m \quad d = \text{hidden size, } m = \# \text{ of heads}\]
  \[W^O \in \mathbb{R}^{d \times d_{\text{out}}}\]

- The total computational cost is similar to that of single-head attention with full dimensionality.

https://jalammar.github.io/illustrated-transformer/
Multi-head attention

“The Beast with Many Heads”

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

- We can think of multi-head attention (MHA) layer as an abstraction layer that maps a sequence of input vectors \( x_1, \ldots, x_n \in \mathbb{R}^{d_{in}} \) to a sequence of \( n \) vectors: \( h_1, \ldots, h_n \in \mathbb{R}^{d_{out}} \)
  
  If we stack multiple layers, usually \( d_{in} = d_{out} = d \)

- The same abstraction as RNNs - used as a drop-in replacement for an RNN layer
  \[
  h_t = f(Wh_{t-1} + Ux_t + b) \in \mathbb{R}^{d_{out}}
  \]

  Much easier to parallelize, more expensive to scale up to longer sequences!

https://jalammar.github.io/illustrated-transformer/
What does multi-head attention learn?

https://github.com/jessevig/bertviz
Transformers: roadmap

- Self-attention and multi-head self-attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder
- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
Adding nonlinearities: Feed-forward layers

- There are no element-wise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors.

- Simple fix: add a feed-forward network to post-process each output vector.

\[
\text{FFN}(x_i) = \text{ReLU}(x_i W_1 + b_1)W_2 + b_2
\]

\[W_1 \in \mathbb{R}^{d \times d_{ff}}, b_1 \in \mathbb{R}^{d_{ff}}\]
\[W_2 \in \mathbb{R}^{d_{ff} \times d}, b_2 \in \mathbb{R}^{d}\]

Usually, \(d_{ff} = 4d\)
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder
- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
Modeling order information: positional encoding

- Unlike RNNs, self-attention doesn’t build in order information, we need to encode the order of the sentence in our keys, queries, and values

- Solution: Add **positional embeddings** to the input embeddings: $p_i \in \mathbb{R}^d$ for $i = 1, 2, \ldots, n$

  $$x_i \leftarrow x_i + p_i$$

- **Sinusoidal positional embeddings**: sine and cosine functions of different frequencies:

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

  - **Pros**: Periodicity + can extrapolate to longer sequences
  - **Cons**: Not learnable
Modeling order information: positional encoding

- **Absolute positional embeddings**: let all $p_i$ be learnable parameters
  - $P \in \mathbb{R}^{d \times L}$ for $L = \text{max sequence length}$
- **Pros**: each position gets to be learned to fit the data
- **Cons**: can’t extrapolate to indices outside of max sequence length $L$
- **Examples**: BERT, GPT-1, GPT-2, GPT-3, OPT

(Devlin et al., NAACL 2019)
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
How to make Transformers work for **deep** NNs?

Add & Norm: $\text{LayerNorm}(x + \text{Sublayer}(x))$

**Residual connections** (He et al., CVPR 2016)

Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ ($i$ represents the layer)

$$X^{(i-1)} \xrightarrow{\text{Layer}} X^{(i)}$$

We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$, so we only need to learn “the residual” from the previous layer

$$X^{(i-1)} \xrightarrow{\text{Layer} +} X^{(i)}$$

This prevents the network from "forgetting" or distorting important information as it is processed by many layers.
How to make Transformers work for *deep* NNs?

Add & Norm:  \( \text{LayerNorm}(x + \text{Sublayer}(x)) \)

**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.

\[
y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \ast \gamma + \beta \quad \gamma, \beta \in \mathbb{R}^d \text{ are learnable parameters}
\]
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder
- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
Let’s put things together - Transformer encoder

From the bottom to the top:
• Input embedding
• Positional encoding
• A stack of Transformer encoder layers

Transformer encoder is a stack of $N$ layers, which consists of two sub-layers:
• Multi-head attention layer
• Feed-forward layer

$$x_1, \ldots, x_n \in \mathbb{R}^{d_{in}} \quad \rightarrow \quad h_1, \ldots, h_n \in \mathbb{R}^{d_{out}}$$
Let’s put things together - Transformer decoder

From the bottom to the top:
• Output embedding
• Positional encoding
• A stack of Transformer decoder layers
• Linear + softmax

Transformer decoder is a stack of $N$ layers, which consists of three sub-layers:
• Masked multi-head attention
• Multi-head cross-attention
• Feed-forward layer
• (w/ Add & Norm between sub-layers)
Masked multi-head self-attention

• Key: You can’t see the future text for the decoder!

• Solution: for every $q_i$, only attend to $\{(k_j, v_j)\}, j \leq i$  

How to implement this? Masking!

https://jalammar.github.io/illustrated-gpt2/
Masked multi-head self-attention

\[ q_i = x_i W^Q, k_i = x_i W^K, v_i = x_i W^V \]

\[ e_{i,j} = \frac{q_i \cdot k_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n \]

\[ \alpha_i = \text{softmax}(e_i) \]

**Efficient implementation:** compute attention as we normally do, mask out attention to future words by setting attention scores to \(-\infty\)

```
dot = torch.bmm(queries, keys.transpose(1, 2))
indices = torch.triu_indices(t, t, offset=1)
dot[::, indices[0], indices[1]] = float('-inf')
dot = F.softmax(dot, dim=2)
```
Masked multi-head self-attention

The following matrix denotes the values of \( \frac{q_i \cdot k_j}{\sqrt{d_k}} \) for \( 1 \leq i \leq n, 1 \leq j \leq n \) (\( n = 4 \))

\[
\begin{array}{cccc}
1 & 0 & -1 & -1 \\
1 & 1 & -1 & 0 \\
0 & 1 & 1 & -1 \\
-1 & -1 & 2 & 1 \\
\end{array}
\]

What should be the value of \( \alpha_{2,2} \) in masked attention?

(A) 0 
(B) 0.5 
(C) \( \frac{e}{2e + e^{-1} + 1} \) 
(D) 1

The correct answer is (B)
Multi-head cross-attention

Similar as the attention in seq2seq model!

Cross-attention between source and target sequence
Multi-head cross-attention

**Self-attention:**

\[ q_i = x_i W^Q, k_i = x_i W^K, v_i = x_i W^V \]

\[ e_{i,j} = \frac{q_i \cdot k_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n \]

\[ \alpha_i = \text{softmax}(e_i) \]

\[ h_i = \sum_{j=1}^{n} \alpha_{i,j} v_j \]

**Cross-attention:**

\[ \tilde{x}_1, \ldots, \tilde{x}_m : \text{hidden states from encoder} \]

\[ x_1, \ldots, x_n : \text{hidden states from decoder} \]

\[ q_i = x_i W^Q \quad i = 1, 2, \ldots, n \]

\[ k_j = \tilde{x}_j W^K, v_j = \tilde{x}_j W^V \quad \forall j = 1, 2, \ldots, m \]

\[ e_{i,j} = \frac{q_i \cdot k_j}{\sqrt{d_k}}, \forall j = 1, \ldots, m \]

\[ \alpha_i = \text{softmax}(e_i) \]

\[ h_i = \sum_{j=1}^{m} \alpha_{i,j} v_j \]

(always from the top layer)
Transformer encoder-decoder

\[ \text{softmax}(W_o h_i) \]
Training Transformer encoder-decoder models

The same as the way that we train seq2seq models!

- Training data: parallel corpus \{\(w_i^{(s)}, w_i^{(t)}\}\)
- Minimize cross-entropy loss:
  \[
  \sum_{t=1}^{T} - \log P(y_t | y_1, \ldots, y_{t-1}, w^{(s)})
  \]
  (denote \(w^{(t)} = y_1, \ldots, y_T\))
- Back-propagate gradients through both encoder and decoder

**Masked self-attention is the key!**

This can enable parallelizable operations while NOT looking at the future
Empirical results with Transformers

<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 [15]</td>
<td>23.75</td>
<td>39.2</td>
</tr>
<tr>
<td>Deep-Att + PosUnk [32]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNMT + RL [31]</td>
<td>24.6</td>
<td>39.92</td>
</tr>
<tr>
<td>ConvS2S [8]</td>
<td>25.16</td>
<td>40.46</td>
</tr>
<tr>
<td>MoE [26]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [32]</td>
<td></td>
<td>40.4</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [31]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>ConvS2S Ensemble [8]</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><strong>28.4</strong></td>
<td><strong>41.0</strong></td>
</tr>
</tbody>
</table>

(Vaswani et al., 2017)
Transformer-based language models

The backbone of large language models (e.g., GPT/ChatGPT, Gemini, LLaMA, …)
## Transformer architecture specifications

(Vaswani et al., 2017)

<table>
<thead>
<tr>
<th>Model Name</th>
<th>$n_{\text{params}}$</th>
<th>$n_{\text{layers}}$</th>
<th>$d_{\text{model}}$</th>
<th>$n_{\text{heads}}$</th>
<th>$d_{\text{head}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3 Small</td>
<td>125M</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>64</td>
</tr>
<tr>
<td>GPT-3 Medium</td>
<td>350M</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>GPT-3 Large</td>
<td>760M</td>
<td>24</td>
<td>1536</td>
<td>16</td>
<td>96</td>
</tr>
<tr>
<td>GPT-3 XL</td>
<td>1.3B</td>
<td>24</td>
<td>2048</td>
<td>24</td>
<td>128</td>
</tr>
<tr>
<td>GPT-3 2.7B</td>
<td>2.7B</td>
<td>32</td>
<td>2560</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>GPT-3 6.7B</td>
<td>6.7B</td>
<td>32</td>
<td>4096</td>
<td>32</td>
<td>128</td>
</tr>
<tr>
<td>GPT-3 13B</td>
<td>13.0B</td>
<td>40</td>
<td>5140</td>
<td>40</td>
<td>128</td>
</tr>
<tr>
<td>GPT-3 175B or “GPT-3”</td>
<td>175.0B</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>128</td>
</tr>
</tbody>
</table>

(Brown et al., 2020)
Transformers: pros and cons

• **Easier to capture long-range dependencies**: we draw attention between every pair of words!

• **Easier to parallelize**:

\[ Q = XW^Q \quad K = XW^K \quad V = XW^V \]

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

• **Are positional embeddings enough to capture positional information?**

  Otherwise self-attention is an unordered function of its input

• **Quadratic computation in self-attention**

  Can become very slow when the sequence becomes very long
Computational analysis of Transformers

Multi-head attention (MHA)

\[ Q = XW_Q, \quad K = XW_K, \quad V = XW_V \]

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

\[ n \times d \quad O(nd^2 + n^2d) \]

Feed-forward layers (FFN)

\[
\text{FFN}(x_i) = \text{ReLU}(x_i W_1 + b_1)W_2 + b_2
\]

\[ n \times d \quad O(nd^2) \]

Note: RNNs only require \( O(nd^2) \) time: \( h_t = f(W h_{t-1} + U x_t + b) \)

(assuming input dimension = hidden dimension = \( d \))
Computational analysis of Transformers

- For BERT-sized models ($n = 512$, $d = 768$, $d_{ff} = 4d$), 2/3 of parameters are FFNs.

- However, when sequence length becomes longer (e.g., $> 50,000$), the computation will be dominated by self-attention $O(n^2d)$
  - Numerous solutions have been proposed to address this issue
  - Long-context language modeling is still one of the most active research areas today

(Ganesh et al., 2020)
Transformers: roadmap

- Self-attention and multi-head attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder
- Advanced techniques: SwiGLU, rotary embeddings, pre-normalization, grouped query attention
- Architecture exploration beyond Transformers
Major modifications since original Transformers
SwiGLU activation

SwiGLU = Swish + GLU

SwiGLU(x, W, V, b, c, \( \beta \)) = Swish_\( \beta \)(xW + b) \otimes (xV + c)

Swish(x) = x \cdot \text{sigmoid}(\beta x)

https://azizbelaweid.substack.com/p/what-is-swiglu-how-to-implement-it

(Shazeer et al., 2020): GLU Variants Improve Transformer
SwiGLU activation

<table>
<thead>
<tr>
<th></th>
<th>Score Average</th>
<th>CoLA MCC</th>
<th>SST-2 Acc</th>
<th>MRPC F1</th>
<th>MRPC Acc</th>
<th>STSB PCC</th>
<th>STSB SCC</th>
<th>QQP F1</th>
<th>QQP Acc</th>
<th>MNLIm Acc</th>
<th>MNLImm Acc</th>
<th>QNLI Acc</th>
<th>RTE Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNReLU</td>
<td>83.80</td>
<td>51.32</td>
<td>94.04</td>
<td>93.08</td>
<td>90.20</td>
<td>89.64</td>
<td>89.42</td>
<td>89.01</td>
<td>91.75</td>
<td>85.83</td>
<td>86.42</td>
<td>92.81</td>
<td>80.14</td>
</tr>
<tr>
<td>FFNGeLU</td>
<td>83.86</td>
<td>53.48</td>
<td>94.04</td>
<td>92.81</td>
<td>90.20</td>
<td>89.69</td>
<td>89.49</td>
<td>88.63</td>
<td>91.62</td>
<td>85.89</td>
<td>86.13</td>
<td>92.39</td>
<td>80.51</td>
</tr>
<tr>
<td>FFNSwish</td>
<td>83.60</td>
<td>49.79</td>
<td>93.69</td>
<td>92.31</td>
<td>89.46</td>
<td>89.20</td>
<td>88.98</td>
<td>88.84</td>
<td>91.67</td>
<td>85.22</td>
<td>85.02</td>
<td>92.33</td>
<td>81.23</td>
</tr>
<tr>
<td>FFNGLU</td>
<td>84.20</td>
<td>49.16</td>
<td>94.27</td>
<td>92.39</td>
<td>89.46</td>
<td>89.46</td>
<td>89.35</td>
<td>88.79</td>
<td>91.62</td>
<td>86.36</td>
<td>86.18</td>
<td>92.92</td>
<td>84.12</td>
</tr>
<tr>
<td>FFNGeGLU</td>
<td>84.12</td>
<td>53.65</td>
<td>93.92</td>
<td>92.68</td>
<td>89.71</td>
<td>90.26</td>
<td>90.13</td>
<td>89.11</td>
<td>91.85</td>
<td>86.15</td>
<td>86.17</td>
<td>92.81</td>
<td>79.42</td>
</tr>
<tr>
<td>FFNBilinear</td>
<td>83.79</td>
<td>51.02</td>
<td>94.38</td>
<td>92.28</td>
<td>89.46</td>
<td>90.06</td>
<td>89.84</td>
<td>88.95</td>
<td>91.69</td>
<td>86.90</td>
<td>87.08</td>
<td>92.92</td>
<td>81.95</td>
</tr>
<tr>
<td>FFNSwGLU</td>
<td>84.36</td>
<td>51.59</td>
<td>93.92</td>
<td>92.23</td>
<td>88.97</td>
<td>90.32</td>
<td>90.13</td>
<td>89.14</td>
<td>91.87</td>
<td>86.45</td>
<td>86.47</td>
<td>92.93</td>
<td>83.39</td>
</tr>
<tr>
<td>FFNReGLU</td>
<td><strong>84.67</strong></td>
<td><strong>56.16</strong></td>
<td><strong>94.38</strong></td>
<td><strong>92.06</strong></td>
<td><strong>89.22</strong></td>
<td><strong>89.97</strong></td>
<td><strong>89.55</strong></td>
<td><strong>88.86</strong></td>
<td><strong>91.72</strong></td>
<td><strong>86.20</strong></td>
<td><strong>86.40</strong></td>
<td><strong>92.68</strong></td>
<td><strong>81.59</strong></td>
</tr>
</tbody>
</table>

[Raffel et al., 2019] 83.28 53.84 92.68 92.07 88.92 88.02 87.94 88.67 91.56 84.24 84.57 90.48 76.28

ibid. stddev. 0.235 1.111 0.569 0.729 1.019 0.374 0.418 0.108 0.070 0.291 0.231 0.361 1.393

SwiGLU($x, W, V, b, c, \beta$) = Swish$_\beta$(x[$\fbox{W}$] + b) $\otimes$ (x[$\fbox{V}$] + c)

Notes: there are 3 projection matrices (up_project, down_project, gate_project), $d_{ff}$ is reduced to $4d \times \frac{2}{3}$
Pre-normalization

RMSNorm normalization function

$$\bar{a}_i = \frac{a_i}{\text{RMS}(a)g_i}, \quad \text{where } \text{RMS}(a) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i^2}.$$  

(Zhang and Senrich, 2019)

Image: (Xiong et al., 2020)
Rotary positional embeddings

- **Relative positional embeddings** *(T5 uses this!)*:

  ![Self-Attention with Relative Position Representations](image)

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

  - Instead of focusing on absolute positions, relative positional embeddings concentrate on the **distances between pairs of tokens**
  - Incorporating this relative positional information into attention directly

*(Shaw et al., 2018) Self-Attention with Relative Position Representations*
Rotary positional embeddings

Unites both absolute and relative positional information

\[ f_{\{q,k\}}(x_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{g,k\}}^{(11)} \\ W_{\{g,k\}}^{(21)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix} \]

(Su et al., 2021) RoFormer: Enhanced Transformer with Rotary Position Embedding
Grouped query attention (GQA)

(Ainslie et al., 2023) GQA: Training generalized multi-query transformer models from multi-head checkpoints.
Architecture exploration beyond Transformers
Efficient Transformers

(Tay et al., 2020): Efficient Transformers: A Survey
Example: Performers

$L$: sequence length, $m << L$

**Low-rank decomposition**: Decompose $A$ as the product of $Q'$ and $K'$ (random projection of original keys and queries)

(Choromanski et al., 2020): Rethinking Attention with Performers
Example: Longformer / Big Bird

**Sparse attention:** only compute attention at particular positions

(a) Full $n^2$ attention  (b) Sliding window attention  (c) Dilated sliding window  (d) Global+sliding window

(Beltagy et al., 2020): Longformer: The Long-Document Transformer

(a) Random attention  (b) Window attention  (c) Global Attention  (d) BIGBIRD

(Zaheer et al., 2021): Big Bird: Transformers for Longer Sequences
Example: Transformer-XL

**Segment-level recurrence with state reuse:** hidden representations from previous segment will be cached as extended context (no back-propagation to those!)

(a) Training phase.

(b) Evaluation phase.

Research from my group

TRIME
Target token's embedding Positive in-batch memory
Other token embeddings Negative in-batch memory

Forward pass Back-propagation
prediction (target: “Apple”) similarity

encoder

Jobs became CEO of...

… works at Microsoft

… returned to Apple

… Jobs became CEO

… moves to Apple

AutoCompressors

use for language modeling summary vectors

LM

summary vectors

LM

summary tokens

randomly segmented input

(Zhong et al., EMNLP’22) Training Language Models with Memory Augmentation

(Chevalier et al., EMNLP’23) Adapting Language Models to Compress Contexts
Research from my group

CEPE

(Yen et al., ACL’24) Long-Context Language Modeling with Parallel Context Encoding

These architectures/techniques are generally applicable to both long-context modeling and retrieval augmentation!
Remarks on efficient Transformers

• A lot of exploration around 2019-2021: mostly approximation solutions of replacing the full quadratic attention. Few techniques have been adopted in state-of-the-art LLMs (exception: Mistral uses Sliding Window Attention to handle longer sequences).

![Diagrams showing Vanilla Attention and Sliding Window Attention](image)

You can still do such approximations to speed up inference though!

• On the other hand, many system-level advancements have been made to scale up Transformers to longer sequences without approximation e.g., FlashAttention (Dao et al., 2022)
(Dosovitskiy et al., 2021): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
Mixture of experts (MoEs)
Mixture of experts (MoEs)

(Zhong et al., 2024): Lory: Fully Differentiable Mixture-of-Experts for Autoregressive Language Model Pre-training
Part II. Pre-trained language models
Roadmap

- **The BERT era**: pre-training and fine-tuning
- **The GPT-3 era**: prompting and in-context learning
- **The ChatGPT era**: supervised instruction tuning and RLHF
The BERT era: pre-training and fine-tuning
BERT = Bidirectional Encoder Representations from Transformers

Input: a sequence of $n$ words
Output: a sequence of $n$ vectors aka. "contextualized word embeddings"
Each word doesn’t have a fixed vector as in (static) word embeddings

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin  Ming-Wei Chang  Kenton Lee  Kristina Toutanova
Google AI Language
{jacobdevlin, mingweichang, kentonl, kristout}@google.com

(Devlin et al., 2019)
Inside BERT: A Transformer encoder

- 12 or 24 layers of Transformer blocks
- Each block consists of a multi-head self-attention layer and a feedforward layer with residual connections
How is BERT pre-trained?

Two **pre-training** objectives:
- Masked language modeling (MLM)
- Next sentence prediction (NSP)

**Masked language modeling**

= mask out 15% of the inputs and then predict the masked words

<table>
<thead>
<tr>
<th>store</th>
<th>gallon</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>↑</td>
</tr>
</tbody>
</table>

the man went to the [MASK] to buy a [MASK] of milk

- Rather than *always* replacing the chosen words with [MASK], the data generator will do the following:
  - 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.

- Too little masking: too expensive to train
- Too much masking: not enough context
How is BERT pre-trained?

Two **pre-training** objectives:
- Masked language modeling (MLM)
- Next sentence prediction (NSP)

Next sentence prediction

Sample two segments of text (**segment A and segment B**) and predict whether the second segment is followed after the first one.

- 50% probability: a text segment of 512 tokens
- 50% probability: a text segment of 256 tokens, followed by another text segment of 256 tokens from a different document

The MLM loss and NSP loss are combined in pre-training
Remarks on BERT’s pre-training objectives

• Later on, (Joshi et al., 2019; Liu et al., 2019) find that the next-sentence prediction objective unnecessary - RoBERTa doesn’t use NSP at all.

• Understanding the role of masking rates (Why 15%?)

Should You Mask 15% in Masked Language Modeling?

Alexander Wettig* Tianyu Gao* Zexuan Zhong Danqi Chen
Department of Computer Science, Princeton University
{awettig,tianyug,zzhong,danqic}@cs.princeton.edu

(Wetting et al., 2023)

• The optimal masking rate should depend on model sizes and masking strategies
• Masking plays two distinct roles: corruption vs prediction
How is BERT used for downstream tasks?

The pre-trained BERT encoder can be used directly for downstream tasks with **minimal task-specific parameters**!
BERT for text classification

- Add a classifier on top of the [CLS] representation
- New parameters: $d \times |C|$, jointly trained with BERT parameters
  - $d =$ hidden dimension (e.g., 768)
  - $|C| =$ number of classes (e.g., 2)
Tesla was the fourth of five children. He had an older brother named Dane and three sisters, Milka, Angelina and Marica. Dane was killed in a horse-riding accident when Nikola was five. In 1861, Tesla attended the "Lower" or "Primary" School in Smiljan where he studied German, arithmetic, and religion. In 1862, the Tesla family moved to Gospić, Austrian Empire, where Tesla’s father worked as a pastor. Nikola completed "Lower" or "Primary" School, followed by the "Lower Real Gymnasium" or "Normal School."

Q: What language did Tesla study while in school?
A: German

(Rajpurkar et al., 2016): SQuAD: 100,000+ Questions for Machine Comprehension of Text
All the parameters are pre-trained except for a small number of task-specific parameters $w_{\text{start}}, w_{\text{end}}$

BERT has 110M or 330M parameters
Prompt-based fine-tuning

Task: sentiment classification

Input: “No reason to watch.”
Output: positive 👍 or negative 👎?

Label mapping

<table>
<thead>
<tr>
<th>Label</th>
<th>Example</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>great</td>
<td>✔</td>
</tr>
<tr>
<td>negative</td>
<td>terrible</td>
<td>✔</td>
</tr>
</tbody>
</table>

Fine-tuning

- 81.4% (32 examples)
- 93.5% (67k examples; BERT-base)

Prompt-based fine-tuning

- 92.7% (32 examples)

(Gao et al., 2021): Making Pre-trained Language Models Better Few-shot Learners
BERT: training cost

- BERT-base: 12 layers, \( n = 512, \ d = 768, \) 110M parameters
- BERT-large: 24 layers, \( n = 512, \ d = 1024, \) 330M parameters
- Trained on Wikipedia + BooksCorpus (3.3 billion tokens)
- Estimate: 6 days for BERT-base and 26 days for BERT-large on 8 Nvidia Titan-V GPUs (12Gb memory)

MosaicBERT: Pretraining BERT from Scratch for $20

https://www.mosaicml.com/blog/mosaicbert (8x A100-80Gb GPUs)
RoBERTa

- BERT is still under-trained
- Removed the next-sentence prediction objective
- Trained longer with 10x data & bigger batch sizes
- Pre-trained on 1,024 V100 GPUs for one day in 2019

<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 <strong>BOOKS + WIKI</strong></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><strong>BERT</strong>&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with <strong>BOOKS + WIKI</strong></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>
</tbody>
</table>

(Liu et al., 2019): RoBERTa: A Robustly Optimized BERT Pretraining Approach
ELECTRA provides a more **efficient** training method, because it predicts 100% of tokens (instead of 15%) every time.

(Clark et al., 2020): ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators
BERT vs GPT-1 models

- Unlike GPT-1 (autoregressive language models), BERT/RoBERTa/ELECTRA can’t generate text naturally!
- However, bidirectionality is important for natural language understanding tasks.

Why not combine the best of both worlds?

(Devlin et al., 2019)
T5: Text-to-text models

- T5 = Text-to-Text Transfer Transformer
- Transformer encoder-decoder architecture
  - Encoder: preserves bidirectionality
  - Decoder: good for generation!

Pre-training (always 15% span masking)

(Original text) Thank you for inviting me to your party last week.

(Inputs) Thank you <X> me to your party <Y> week.

(Targets) <X> for inviting <Y> last <Z>

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
T5: Text-to-text models

- T5 = Text-to-Text Transfer Transformer

Fine-tuning

All NLU tasks can be cast as text prediction tasks too!

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
T5: Text-to-text models

- T5 = Text-to-Text Transfer Transformer

T5 comes in different sizes:

- t5-small.
- t5-base.
- t5-large.
- t5-3b.
- t5-11b.

Training corpus:

C4 = Colossal Clean Crawled Corpus

A cleaned version of Common Crawl
Trained on 34B tokens

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
BART: another text-to-text model

(Lewis et al., 2020): BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension
The GPT-3 era: prompting and in-context learning
**GPT-3**

- From GPT-1 to GPT-2 to GPT-3
- **Transformer decoder** for autoregressive language modeling
- What’s new?
  - Parameters ↑ Data ↑
  - These models can start generating human-like text!
  - You don’t have to fine-tune these models - **prompting** works well!

---

**Language Models are Few-Shot Learners**

<table>
<thead>
<tr>
<th>Tom B. Brown*</th>
<th>Benjamin Mann*</th>
<th>Nick Ryder*</th>
<th>Melanie Subbiah*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jared Kaplan†</td>
<td>Prafulla Dhariwal</td>
<td>Arvind Neelakantan</td>
<td>Pranav Shyam</td>
</tr>
<tr>
<td>Girish Sastry</td>
<td>Amanda Askell</td>
<td>Sandhini Agarwal</td>
<td>Ariel Herbert-Voss</td>
</tr>
<tr>
<td>Gretchen Krueger</td>
<td>Tom Henighan</td>
<td>Rewon Child</td>
<td>Aditya Ramesh</td>
</tr>
<tr>
<td>Daniel M. Ziegler</td>
<td>Jeffrey Wu</td>
<td>Clemens Winter</td>
<td></td>
</tr>
<tr>
<td>Christopher Hesse</td>
<td>Mark Chen</td>
<td>Eric Sigler</td>
<td>Mateusz Litwin</td>
</tr>
<tr>
<td>Benjamin Chess</td>
<td>Jack Clark</td>
<td>Christopher Berner</td>
<td></td>
</tr>
<tr>
<td>Sam McCandlish</td>
<td>Alec Radford</td>
<td>Ilya Sutskever</td>
<td>Dario Amodei</td>
</tr>
</tbody>
</table>

*(Brown et al., 2020)*
GPT-1 vs GPT-2 vs GPT-3

https://medium.com/@YanAlx/

(Zha et al., 2023)
A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.

The incident occurred on the downtown train line, which runs from Covington and Ashland stations.

In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief.

“The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation,” said Tom Hicks, the U.S. Energy Secretary, in a statement. “Our top priority is to secure the theft and ensure it doesn’t happen again.”

The stolen material was taken from the University of Cincinnati’s Research Triangle Park nuclear research site, according to a news release from Department officials.
GPT-3’s in-context learning

• Before GPT-3, most use of pre-trained language models are through fine-tuning, on a reasonably-sized supervised dataset.
  • SST-2 has 67k examples, SQuAD has 88k (passage, answer, question) triples.

• GPT-3 shows that, with a very large autoregressive language model (175B parameters), the model can perform a task:
  • Using only a few examples  “Few-shot learning”
  • Without gradient updates:
    the examples are only provided in the context!  “In-context learning”

(Brown et al., 2020): Language Models are Few-Shot Learners
In-context learning

**Few-shot**

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1. Translate English to French:
2. sea otter => loutre de mer
3. peppermint => menthe poivrée
4. plush giraffe => girafe peluche
5. cheese => ................................
```
In-context learning

(Brown et al., 2020): Language Models are Few-Shot Learners
In-context learning

Input: 2014-06-01
Output: !06!01!2014!

Input: 2007-12-13
Output: !12!13!2007!

Input: 2010-09-23
Output: !09!23!2010!

Input: 2005-07-23
Output: !07!23!2005!

in-context examples

model completion

test example

http://ai.stanford.edu/blog/in-context-learning/
Understanding in-context learning

- **Hypothesis #1**: Transformers perform implicit gradient descent to update an “inner model”

- **Hypothesis #2**: Transformers learn tasks required for downstream applications during pre-training, and in-context demonstrations are only used to recognize which task is required.
Understand in-context learning

We disentangle In-context learning into two roles - task recognition (TR) vs task learning (TL)

• TR: recognizes the task from demonstrations and applies LLMs’ pre-trained priors

• TL: learns a new input-label mapping from demonstrations

• ICL performs both TR and TL, but TL emerges with larger models and more demonstrations

(Pan et al., 2023): What In-Context Learning “Lears” In-Context: Disentangling Task Recognition and Task Learning
Chain-of-thought (CoT) prompting

(Wei et al., 2022): Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
The ChatGPT era: Supervised instruction tuning and RLHF
From 2020’s GPT-3 to 2022’s ChatGPT

What’s new?

• Code training
• Supervised instruction tuning
• RLHF = Reinforcement learning from human feedback

https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1ab9e3e36fa1dc1
Post-training pipeline of LLMs

Training language models to follow instructions with human feedback

Long Ouyang*  Jeff Wu*  Xu Jiang*  Diogo Almeida*  Carroll L. Wainwright*

Pamela Mishkin*  Chong Zhang  Sandhini Agarwal  Katarina Slama  Alex Ray

John Schulman  Jacob Hilton  Fraser Kelton  Luke Miller  Maddie Simens

Amanda Askell†  Peter Welinder  Paul Christiano*†

Jan Leike*  Ryan Lowe*

OpenAI

The InstructGPT paper
Post-training pipeline of LLMs

Post-training pipeline of LLMs (Ouyang et al., 2022): Training language models to follow instructions with human feedback.
Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research.

"Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-78.
Supervised fine-tuning (SFT)

<table>
<thead>
<tr>
<th>Use-case</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brainstorming</td>
<td>List five ideas for how to regain enthusiasm for my career</td>
</tr>
<tr>
<td>Generation</td>
<td>Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.</td>
</tr>
<tr>
<td>Rewrite</td>
<td>This is the summary of a Broadway play:</td>
</tr>
<tr>
<td></td>
<td>&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td></td>
<td>{summary}</td>
</tr>
<tr>
<td></td>
<td>&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td></td>
<td>This is the outline of the commercial for that play:</td>
</tr>
<tr>
<td></td>
<td>&quot;&quot;&quot;&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use-case</th>
<th>(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation</td>
<td>45.6%</td>
</tr>
<tr>
<td>Open QA</td>
<td>12.4%</td>
</tr>
<tr>
<td>Brainstorming</td>
<td>11.2%</td>
</tr>
<tr>
<td>Chat</td>
<td>8.4%</td>
</tr>
<tr>
<td>Rewrite</td>
<td>6.6%</td>
</tr>
<tr>
<td>Summarization</td>
<td>4.2%</td>
</tr>
<tr>
<td>Classification</td>
<td>3.5%</td>
</tr>
<tr>
<td>Other</td>
<td>3.5%</td>
</tr>
<tr>
<td>Closed QA</td>
<td>2.6%</td>
</tr>
<tr>
<td>Extract</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

SFT data: only ~13k (written by labeller, not public)

(Ouyang et al., 2022): Training language models to follow instructions with human feedback
Collection of supervised fine-tuning data

- Repurposed from existing supervised datasets
- Human-written instructions and responses

Super-NaturalInstructions (Wang et al., 2022)
Also: FLAN, T0++
Collection of supervised fine-tuning data

- Response distilled from GPT models
- Instructions can be generated by GPT models too, e.g., Self-Instruct (Wang et al., 2023)

https://sharegpt.com/  Stanford Alpaca
Learning from human feedback

- Preference data: (instruction, winning response, losing response)

InstructGPT: 33k prompts, each with K (4~9) corresponding SFT model completions ranked by labellers

(Ouyang et al., 2022) Training language models to follow instructions with human feedback
Reinforcement learning from human feedback

Multiple stages in training
- Reward model training
- Sampling from policy model
- Policy model update

Multiple models involved
- Reward model
- Policy model
- Reference model

https://huggingface.co/blog/rlhf
# GPT-3 vs InstructGPT

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GPT</th>
<th>Supervised Fine-Tuning</th>
<th>InstructGPT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RealToxicity</strong></td>
<td>0.233</td>
<td>0.199</td>
<td>0.196</td>
</tr>
<tr>
<td><strong>TruthfulQA</strong></td>
<td>0.224</td>
<td>0.206</td>
<td>0.413</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GPT</th>
<th>Supervised Fine-Tuning</th>
<th>InstructGPT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hallucinations</strong></td>
<td>0.414</td>
<td><strong>0.078</strong></td>
<td>0.172</td>
</tr>
<tr>
<td><strong>Customer Assistant Appropriate</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[https://openai.com/index/instruction-following/](https://openai.com/index/instruction-following/)
Direct preference optimization (DPO)

Instead of training an explicit reward model, express reward in the form of policy model:

**Implicit reward expression:**

\[ r(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x) \]

**Bradley-Terry ranking objective:**

\[ \mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_i) \sim \mathcal{D}} \left[ \log \sigma(r_\phi(x, y_w) - r_\phi(x, y_i)) \right] \]

**DPO objective:**

\[ \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_i) \sim \mathcal{D}} \left[ \log \sigma \left( \beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_i \mid x)}{\pi_{\text{ref}}(y_i \mid x)} \right) \right] \]

(Rafailov et al., 2023) Direct Preference Optimization: Your Language Model is Secretly a Reward Model
Simple preference optimization (SimPO)

\[
\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}\left[ \log \sigma \left( \beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]
\]

\[
\mathcal{L}_{\text{SimPO}}(\pi_\theta) = -\mathbb{E}\left[ \log \sigma \left( \frac{\beta}{|y_w|} \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \frac{\beta}{|y_l|} \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} - \gamma \right) \right]
\]

- A simple length-normalized reward (reference-free!):

\[
r_{\text{SimPO}}(x, y) = \frac{\beta}{|y|} \log \pi_\theta(y \mid x) = \frac{\beta}{|y|} \sum_{i=1}^{|y|} \log \pi_\theta(y_i \mid x, y_{<i})
\]

- Introducing target reward margin in Bradley-Terry objective:

\[
p(y_w \succ y_l \mid x) = \sigma \left( r(x, y_w) - r(x, y_l) - [\gamma] \right)
\]

(Meng et al., 2024) SimPO: Simple Preference Optimization with a Reference-Free Reward
Simple preference optimization (SimPO)

(Meng et al., 2024) SimPO: Simple Preference Optimization with a Reference-Free Reward