Large Language Models (LLMs) - Data and Models

Rowel Atienza, PhD

University of the Philippines

github.com/roatienza

2023

https://stanford-cs324.github.io/winter2022

Data for LLMs

- Must be diverse
- Sources : Web
    - Common Crawl – free and open, 320TB
- Other sources: Private datasets

Representation

- Web data – mostly from young people and developed countries
- GPT2 – trained on Reddit and WebText; most contributors are men
- Wikipedia – at most 15% contributors are women

WebText

- Diverse but high quality
- News, wikipedia, fiction
- 40GB of text

OpenWebText - OpenAI

- Extracted all the URLs from the Reddit submissions dataset.
- Used Facebook’s fastText to filter out non-English.
- Removed near duplicates.
- End result is 38 GB of text.
- Small toxicity content

Colossal Clean Crawled Corpus (C4)

- Started with April 2019 snapshot of Common Crawl (1.4 trillion tokens)
- Removed “bad words”
- Removed code (“{“)
- langdetect to filter out non-English text
- Resulted in 806 GB of text (156 billion tokens)

GPT3 Dataset

- Downloaded 41 shards of Common Crawl (2016-2019).
- 13-gram deduplication
- No benchmark datasets
- Expanded data sources (WebText2, Books1, Books2, Wikipedia).

The Pile – Eleuther AI

- A dataset for language modeling, where the key idea is to source it from smaller high-quality sources (academic + professional sources).
- 825 GB English text
- 22 high-quality datasets

Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3†	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
FreeLaw	51.15 GiB	6.12%	1.5	76.73 GiB	15.06 KiB
Stack Exchange	32.20 GiB	5.13%	2.0	64.39 GiB	2.16 KiB
USPTO Backgrounds	22.90 GiB	3.65%	2.0	45.81 GiB	4.08 KiB
PubMed Abstracts	19.26 GiB	3.07%	2.0	38.53 GiB	1.30 KiB
Gutenberg (PG-19)†	10.88 GiB	2.17%	2.5	27.19 GiB	398.73 KiB
OpenSubtitles†	12.98 GiB	1.55%	1.5	19.47 GiB	30.48 KiB
Wikipedia (en)†	6.38 GiB	1.53%	3.0	19.13 GiB	1.11 KiB
DM Mathematics†	7.75 GiB	1.24%	2.0	15.49 GiB	8.00 KiB
Ubuntu IRC	5.52 GiB	0.88%	2.0	11.03 GiB	545.48 KiB
BookCorpus2	6.30 GiB	0.75%	1.5	9.45 GiB	369.87 KiB
EuroParl†	4.59 GiB	0.73%	2.0	9.17 GiB	68.87 KiB
HackerNews	3.90 GiB	0.62%	2.0	7.80 GiB	4.92 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails†	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB
The Pile	**825.18 GiB**			**1254.20 GiB**	**5.91 KiB**

Modeling

Language Model

- Probability distribution over sequences of tokens
- The LM:

$$p(x_1, x_2, \dots, x_L) = p \in [0,1]$$

- LM using Prompt to Completion

$$p(x_1, x_2, \dots, x_L) =$$

$$p(x_1, x_2, \dots, x_P)p(x_{P+1}|x_1, x_2 \cdots x_P) \cdots p(x_L|x_1, x_2 \cdots x_{L-1})$$

Where $x_1, x_2, \dots, x_L \in \mathcal{V}$. L is the sequence length. Assumption is there an existing a vocabulary $\mathcal{V}$

Language Model

Goal: Learn $p(x_1, x_2, \ldots, x_L)$ from data (eg. The Pile)

Tokenization

Tokenization: how a string is split into tokens.

Model Architecture

Model: Estimates $p(x_1, x_2, \dots, x_L)$ from data represented as tokens.

Tokenization

- Language comes as a string
  - *Language*: “the cat sat on the mat”
- A tokenizer converts a string into tokens
  - *Tokens*: “the cat sat on the mat” → $[the \quad cat \quad sat \quad on \quad the \quad mat]$
- Easiest :

```
>>> text = "the cat sat on the mat"
>>> text.split(" ")
['the', 'cat', 'sat', 'on', 'the', 'mat']
```

Tokenization

- However, in some languages (and some words in English), spaces do not separate words

Good Tokenizer

- Not too many tokens (extreme: characters or bytes), or else the sequence becomes difficult to model.
- Not too few tokens, or else there won’t be parameter sharing between words (e.g., should mother-in-law and father-in-law be completely different)?
- Each token should be a linguistically or statistically meaningful unit.

Byte Pair Encoding (BPE) [Sennrich et al, 2015]

- Learning the tokenizer. Intuition: start with each character as its own token and combine tokens that co-occur a lot.
- Input: a training corpus (sequence of characters).
- Initialize the vocabulary $x_1, x_2, \dots, x_L \in \mathcal{V}$
- While we want to still grow $\mathcal{V}$:
- Find the pair of elements $x_i, x_j \in \mathcal{V}$ that co-occur the most number of times.
- Replace all occurrences of x_i, x_j with a new symbol x_{ij}.
- Add x_{ij} to $\mathcal{V}$.

Example

- Data: $[t\ h\ e\ \_\ c\ a\ t], [t\ h\ e\ \_\ c\ a\ p], [t\ h\ e\ \_\ b\ a\ t]$
- New tokens:
  - $th\ - 3\times$
  - $the\ - 3\times$
  - $ca - 2\times$
  - $at\ - 2\times$
- Updated $\mathcal{V}$: $t, h, e, c, a, t, p, b, th, the, ca, at$
- $Tokenize([t\ h\ e\ \_\ c\ a\ t]) = [the\ \_\ ca\ t]$

Unicode

- 144,697 of Unicode characters.
- Solution: Run on byte equivalent of characters

SentencePiece (Unigram Model) [Kudo 2018]

$$p(x_{1:L}) = \prod_{(i,j)\in\mathcal{T}} p(x_{i:j})$$

- Training: $[ababc]$
- Tokenization $\mathcal{T} = \{(1,2),(3,4),(5,5)\}$
- Vocabulary $\mathcal{V} = \{ab, c\}$
- Likelihood $p(x_{1:5}) = \frac{2}{3}\times\frac{2}{3}\times\frac{1}{3} = \frac{4}{9}$

SentencePiece (Unigram Model) [Kudo 2018]

- Fully reversible (lossless) – tokenized strings can be detokenized w/o ambiguity.
  - `detokenize(tokenize(‘I love New York.')) == 'I love New York.'`
- Uses both BPE and unigram for tokenization
- Uses Unicode characters, including whitespace, and using a consistent encoding and decoding scheme to preserve all the information needed to reproduce the original text
- Fast, self-contained, language independent

https://github.com/google/sentencepiece

https://medium.com/codex/sentencepiece-a-simple-and-language-independent-subword-tokenizer-and-detokenizer-for-neural-text-ffda431e704e

SentencePiece Algorithm

1. Start with a "reasonably big" seed vocabulary $\mathcal{V}$.
   - $\mathcal{V}$: $[ababc] \rightarrow [abab\ aba\ ab\ a\ b\ c]$
2. Repeat:
   a) Given $\mathcal{V}$, optimize $p(x)$ and $\mathcal{T}$ using the EM algorithm.
   b) Compute $loss(x)$ for each token $x \in \mathcal{V}$ capturing how much the likelihood would be reduced if x were removed from $\mathcal{V}$.
   $[ab\ ab\ c], [a\ b\ a\ b\ c], [aba\ b\ c], [abab\ c]$
   c) Sort by loss and keep the top 80% tokens in $\mathcal{V}$.

SentencePiece vs BPE

- GPT-2 and GPT-3 used BPE, vocabulary size of 50K
- Jurassic used SentencePiece with vocabulary size of 256K
- Impact:
  - Given the same string, Jurassic requires 28% fewer tokens than GPT-3, so it is 1.4x faster
  - Both Jurassic and GPT-3 use the same context size (2048), so one can feed in 39% more text into the prompt.

Example

- GPT-3 BPE (9 tokens):
    - [Ab, raham, ␣Lincoln, ␣lived, ␣at, ␣the, ␣White, ␣House, .]
- Jurassic SentencePiece (4 tokens):
    - [Abraham␣Lincoln, ␣lived, ␣at␣the␣White␣House, .]

LM Types

LM Types

- Encoder Only – BERTs
- Decoder Only – GPTs
- Encoder-Decoder – BART, T5

Contextual Embeddings

- Embedding: ϕ: $\mathcal{V}^L \to \mathbb{R}^{d \times L}$
  - Tokens to features
- Embedding: $x_{1:L} = [x_1, x_2, \ldots, x_L] \xrightarrow{\phi} [\phi(x_1), \phi(x_2), \ldots, \phi(x_L)]$
- Example:

$$x_{1:3} = [ab, ab, c] \xrightarrow{\phi} [\phi(ab), \phi(ab), \phi(c)] = \left[\begin{bmatrix} 1. \\ -1. \\ 0. \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 1. \\ 1. \end{bmatrix} \right]$$

$$= \phi(x_{1:3})$$

Encoder Only (BERT, RoBERTa)

- Can produce contextual embedding: $x_{1:L} \overset{\phi}{\rightarrow} \phi(x_{1:L})$
- Application 1: Sentiment classification
  - [[CLS],the,movie,was,great]⇒positive.
- Application 2: Natural language inference
  - [[CLS],all,animals,breathe,[SEP],cats,breathe]⇒entailment.
- Pro: bidirectional dependency
- Con: Cannot generate text
- Loss: Masked Language Modelling

Decoder Only (GPTs) – Autoregressive Models

- Can produce contextual embedding of the prompt: $x_{1:P} \xrightarrow{\phi} \phi(x_{1:P})$ and the distribution over the next tokens $p(x_{P+1:L})$ or to the completion.

$$x_{1:P} \Rightarrow \phi(x_{1:P}) p(x_{P+1:L} | \phi(x_{1:P}))$$

- Application: Text Generation
  - [[CLS],the,movie,was]⇒great
- Pro: Can naturally generate text completions.
- Con: Unidirectional dependency (causal)
- Loss: Maximum Likelihood

Encoder-Decoder (T5, BART)

- Can produce contextual embedding: $x_{1:L} \overset{\phi}{\rightarrow} \phi(x_{1:L})$ and generate new output : $y_{1:L}$.

$$x_{1:L} \Rightarrow \phi(x_{1:L}) p(y_{1:L} | \phi(x_{1:L}))$$

- Application: Table to text conversion
- Pro: bidirectional dependency.
- Con: can generate new outputs
- Loss: Ad-hoc training objectives

General Algorithm for LMs

1. Given an input string: $\boldsymbol{x}$
2. $\boldsymbol{y}_{tok} = Tokenize(\boldsymbol{x})$
3. $\boldsymbol{y}_{emb} = Embed(\boldsymbol{y}_{tok})$
4. $\boldsymbol{y}_{ctx} = ContextEmbed(\boldsymbol{y}_{emb})$
5. $\boldsymbol{y}_{seq} = SequenceModel(\boldsymbol{y}_{ctx})$

General Algorithm for LMs

1. Given an input string: $\boldsymbol{x}$
2. $Tokenize(\boldsymbol{x})$ - Tokenizer (eg sentencepiece, HF)
3. $Embed(\boldsymbol{y}_{tok})$ - `nn.Embedding()`
4. $ContextEmbed(\boldsymbol{y}_{emb})$ - `nn.Transformer()`
5. $SequenceModel(\boldsymbol{y}_{ctx})$ - HF

End