Large Language Models in Economics

Jesús Fernández-Villaverde[1]

August 7, 2023

[1]University of Pennsylvania

Some background

- These slides are available at `www.sas.upenn.edu/~jesusfv/LLM.pdf`. I will periodically update them.
- Also, for more general material on artificial intelligence and deep learning, check:
  1. `https://www.sas.upenn.edu/~jesusfv/teaching.html`.
  2. `https://youtu.be/ky5lTihM1U0`.
- The code examples use `Python+PyTorch`, but you can follow the arguments without any knowledge of `Python+PyTorch`.
- I will cite further references (many online!), but let me know if you want specific references.
- Thanks to many coauthors and students.

Outline

- A complete treatment of large language models (LLMs) and their application in economics deserves a whole semester of lecturing.
- Instead, I will focus on a few key ideas (e.g., what is a LLM?, transduction, embedding, and attention).
- I will go from the more general to the more technical:
    1. The revolution of LLMs.
    2. On the role of LLMs in economics.
    3. Text as data.
    4. Natural language processing.
    5. The transformer model.

The revolution of LLMs

What is a LLM?

- ChatGPT, a chatbot built on top of the GPT LLM released on November 28, 2022, has popularized deep learning models trained with a text corpus.

- Language models learn a probability distribution over language:

$$P(w_1, \ldots, w_m)$$

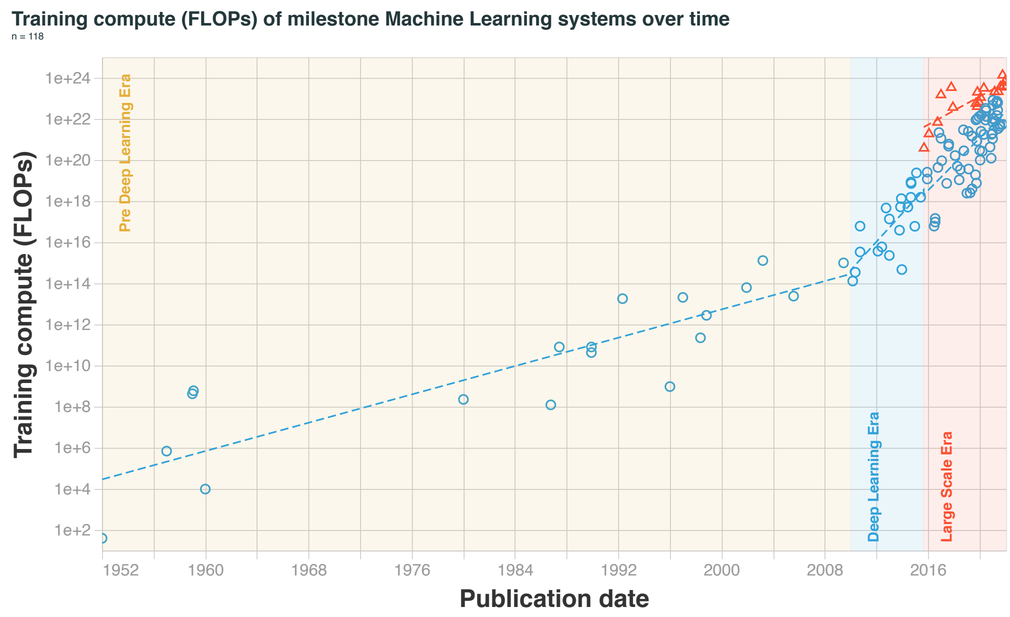
  For example, what is the most likely word after "European Central" in an article at the FT?

- A language model can use many different probability structures and not necessarily a deep neural networks (even if the latter have gained much popularity).

- Large in terms of training data (e.g., Common Crawl, Wikipedia, GitHub, ...) and parameters (e.g., PaLM has 540 billion parameters; GPT-4 rumored to have 1 trillion).

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

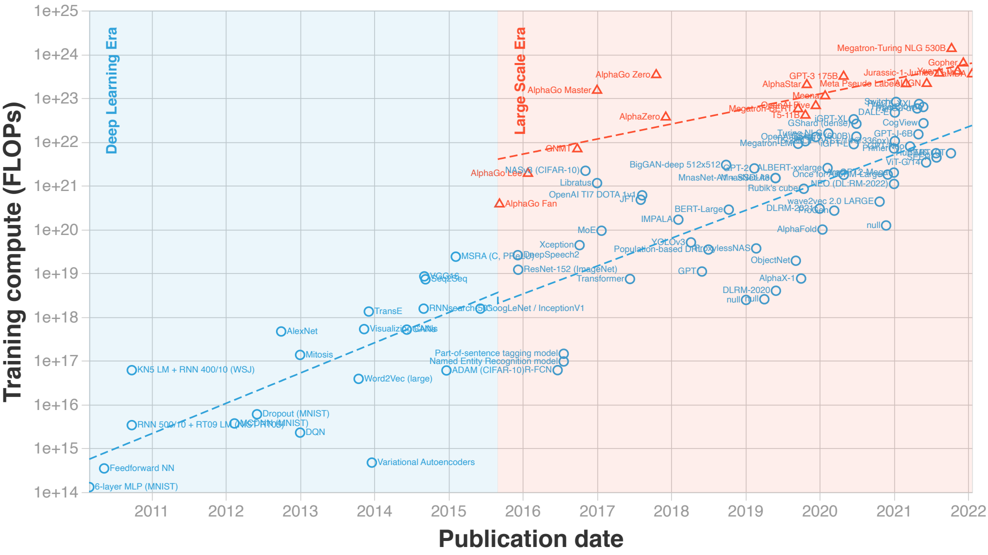
Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.



Training compute (FLOPs) of milestone Machine Learning systems over time
n = 118
Pre Deep Learning Era
Deep Learning Era
Large Scale Era
Training compute (FLOPs)
1e+24
1e+22
1e+20
1e+18
1e+16
1e+14
1e+12
1e+10
1e+8
1e+6
1e+4
1e+2
1952
1960
1968
1976
1984
1992
2000
2008
2016
Publication date

Training compute (FLOPs) of milestone Machine Learning systems over time

n = 99



Location, location, location

- Original contribution by Elman (1990): Finding Structure in Time.
- Key idea: exploit the location of words within a text.

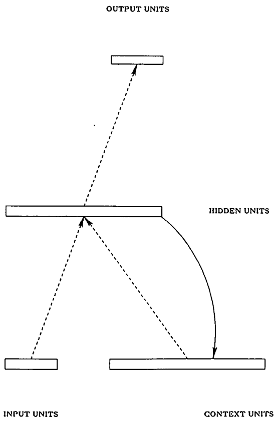


Figure 2. A simple recurrent network in which activations are copied from hidden layer to context layer on a one-for-one basis, with fixed weight of 1.0. Dotted lines represent trainable connections.

The transformers

- Why now?
- Conjunction of:
  1. A pathbreaking algorithmic revolution: transformer models based on self-attention (December 2017).
  2. GPUs: attention multiheads can run on separate GPUs openings.
  3. We have learned that we want to train LLMs according to power laws linking complexity and data. Hoffman *et al.*, 2022: for every doubling of model size the number of training tokens should also be doubled.
- This is the reason behind the "T" in `GPT` (generative pre-trained transformer).

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

NVIDIA Corp

NASDAQ: NVDA

Overview | Compare | Financials

Market Summary > NVIDIA Corp

454.69 USD

+ Follow

+391.97 (624.95%) ↑ past 5 years

Closed: Jul 14, 7:59 PM EDT • Disclaimer

After hours 456.54 +1.85 (0.41%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	465.83	Mkt cap	1.12T	CDP score	B
High	480.88	P/E ratio	236.30	52-wk high	480.88
Low	450.60	Div yield	0.035%	52-wk low	108.13

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Three kinds of LLM

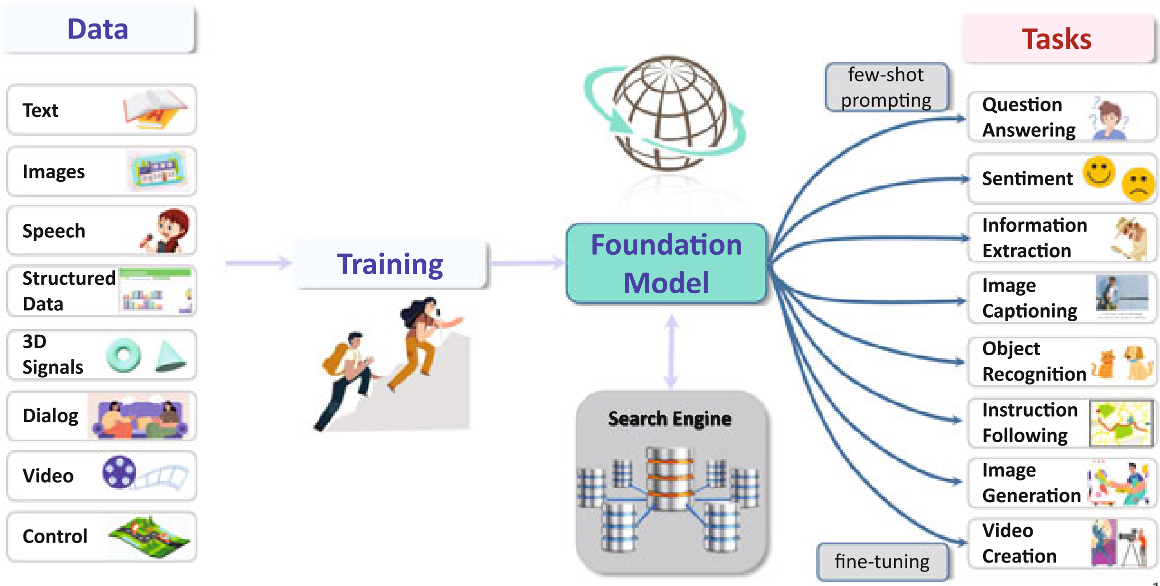
- Generic language models: predicting the next token. I will center on this one type.
- Instruction tuned.
- Dialog tuned: ChatGPT (the base model is hard to interact with).

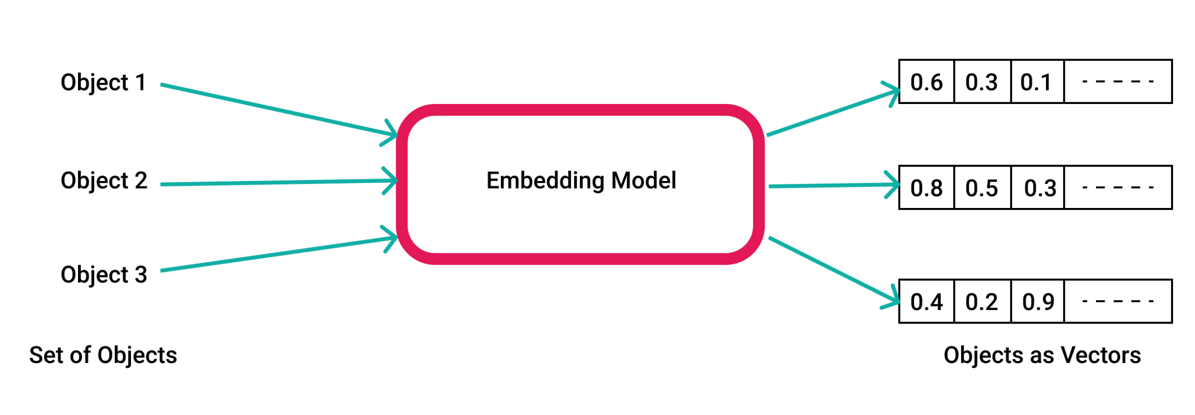
The uses of LLM

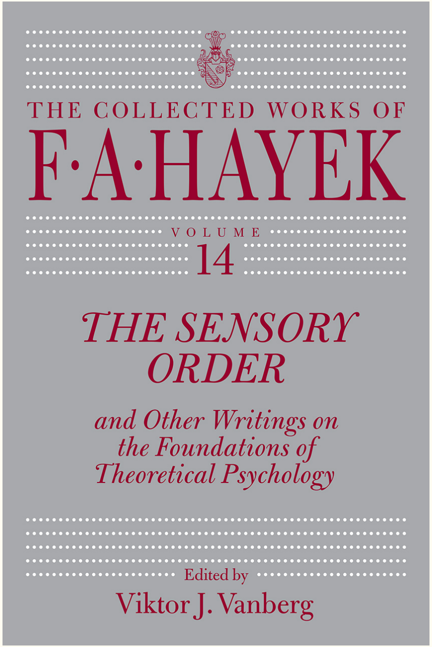
- All three types share that they are trained to tackle text-based tasks:
  1. Text classification.
  2. Text summarization (including sentiment analysis).
  3. Text generation (including translation and coding).
  4. Questions/Answers.
  5. Common sense reasoning.
- Because of these capabilities, we can consider LLMs as a part of generative AI: models capable of generating new content.
- This is the reason behind the "G" in `GPT` (generative pre-trained transformer).

Foundation models I

- Some authors are even talking about foundation models: instead of multiple pipelines for each task, we have a common one.
- Key reason: embedding.
- Adapted models and pluggings.
- Emerging properties we do not fully understand:
  1. For example, LLMs seem to have a theory of the mind.
  2. Related to old ideas in F.A. Hayek's The Sensory Order.


Data
Text
Images
Speech
Structured Data
3D Signals
Dialog
Video
Control
Training
Foundation Model
Search Engine
few-shot prompting
fine-tuning
Tasks
Question Answering
Sentiment
Information Extraction
Image Captioning
Object Recognition
Instruction Following
Image Generation
Video Creation


Object 1
Object 2
Object 3
Embedding Model
0.6 0.3 0.1 - - - - -
0.8 0.5 0.3 - - - - -
0.4 0.2 0.9 - - - - -
Set of Objects
Objects as Vectors


THE COLLECTED WORKS OF
F·A·HAYEK
VOLUME
14
THE SENSORY ORDER
and Other Writings on the Foundations of Theoretical Psychology
Edited by
Viktor J. Vanberg

Foundation models II

- How far away are we from human-level artificial general intelligence (AGI)? (what is intelligence anyway?).
- Questions:
    1. Hallucinations?
    2. Safety vs. accuracy?
    3. Existential risk from AI?
- https://munkdebates.com/debates/artificial-intelligence

On the Opportunities and Risks of Foundation Models

Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles Hamed Nilforoshan
Julian Nyarko Giray Ogut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*[1]

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

General vs. specialized LLM

- Either for general purpose or specialized corpora of documents.
- You can pre-train the LLM in a large dataset and adapt it to a smaller corpus (even zero-shot learning).
  - For example, all documents within the Fed, all the NBER working papers, all articles at the FT.
- This is the reason behind the "P" in `GPT` (generative pre-trained transformer).
- Parameter-efficient fine-tuning methods, prompt training, and supervised learning.
- Key idea: Transduction (particular$\rightarrow$particular) vs. induction (particular$\rightarrow$general$\rightarrow$particular).
- Related to the failure of the project of building a universal formal grammar in the 1970s (we will return to this point later on).

Language Models are Few-Shot Learners

Tom B. Brown* **Benjamin Mann*** **Nick Ryder*** **Melanie Subbiah***

Jared Kaplan† **Prafulla Dhariwal** **Arvind Neelakantan** **Pranav Shyam** **Girish Sastry**

Amanda Askell **Sandhini Agarwal** **Ariel Herbert-Voss** **Gretchen Krueger** **Tom Henighan**

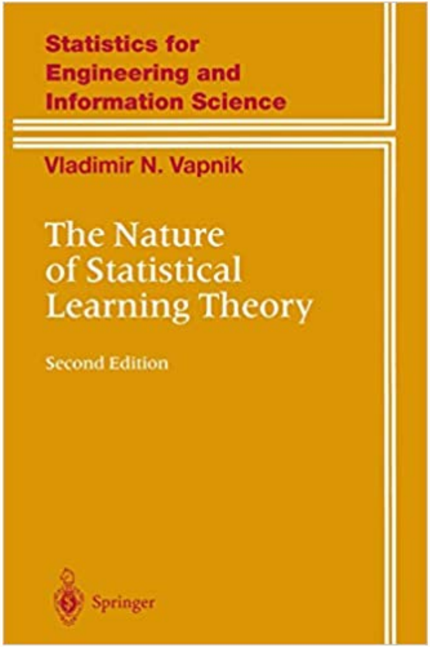
Rewon Child **Aditya Ramesh** **Daniel M. Ziegler** **Jeffrey Wu** **Clemens Winter**

Christopher Hesse **Mark Chen** **Eric Sigler** **Mateusz Litwin** **Scott Gray**

Benjamin Chess **Jack Clark** **Christopher Berner**

Sam McCandlish **Alec Radford** **Ilya Sutskever** **Dario Amodei**

OpenAI

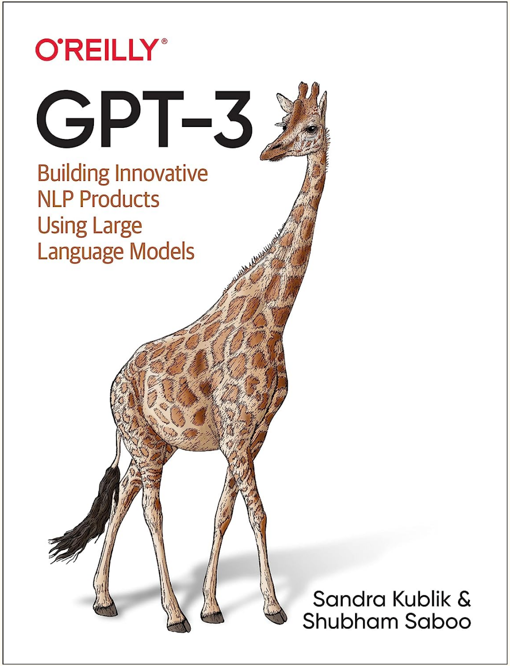

Statistics for
Engineering and
Information Science
Vladimir N. Vapnik
The Nature
of Statistical
Learning Theory
Second Edition
Springer

Chatbots vs. APIs

- You might have used the chatbot for `ChatGPT`. This is the reason behind the "Chat" in `ChatGPT` (chatbot for generative pre-trained transformer).
- However, for more systematic research, one can use APIs (application programming interfaces) and focus on prompt design:

```python
import openai

openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World
        {"role": "user", "content": "Where was it played?"}
    ]
)
```


O'REILLY
GPT-3
Building Innovative
NLP Products
Using Large
Language Models
Sandra Kublik &
Shubham Saboo

Life beyond ChatGPT

1. `ChatGPT`: designed for chatbots and conversational AI.
2. `Llama 2`: best open source model, trained on 1-1.4T tokens.
3. `Bard`: `Google`.
4. `LangChain`: designed for translation.
5. `Cohere`: designed for text classification, summarization, and sentiment analysis.
6. Many others.

ai.meta.com/resources/models-and-libraries/llama/

Meta AI

Research Blog Resources About

Large language model

Llama 2: open source, free for research and commercial use

We're unlocking the power of these large language models. Our latest version of Llama – Llama 2 – is now accessible to individuals, creators, researchers, and businesses so they can experiment, innovate, and scale their ideas responsibly.

Download the Model

huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

Spaces HuggingFaceH4 / open_llm_leaderboard like 4.07k Running App Files Community 159

🤗 Open LLM Leaderboard

📐 The 🤗 Open LLM Leaderboard aims to track, rank and evaluate LLMs and chatbots as they are released.

🤗 Anyone from the community can submit a model for automated evaluation on the 🤗 GPU cluster, as long as it is a 🤗 Transformers model with weights on the Hub. We also support evaluation of models with delta-weights for non-commercial licensed models, such as the original LLaMa release.

Other cool benchmarks for LLMs are developed at HuggingFace, go check them out: 🙋🤖 human and GPT4 evals, 🖥️ performance benchmarks

🟢: Base pretrained model – 🔶: Finetuned model – 🟦: Model using RL (read more details in "About" tab)

🏅 LLM Benchmark | 📝 About | 🚀 Submit here!

Select columns to show

- [x] Average ⬆️
- [x] ARC
- [x] HellaSwag
- [x] MMLU
- [x] TruthfulQA
- [ ] Type
- [ ] Hub License
- [ ] #Params (B)
- [ ] Hub ❤️
- [ ] Model sha

🔍 Search for your model and press ENTER...

Filter model types

(•) all ( ) 🟢 base ( ) 🔶 finetuned ( ) 🟦 RL-tuned

T	Model	Average ⬆️	ARC	HellaSwag	MMLU	TruthfulQA
	upstage/Llama-2-70b-instruct-v2	73	71.1	87.9	70.6	62.2
🔶	upstage/Llama-2-70b-instruct	72.3	70.9	87.5	69.8	61
🔶	stabilityai/StableBeluga2	71.4	71.1	86.4	68.8	59.4
🔶	augtoma/qCammel-70-x	71	68.3	87.9	70.2	57.5
🔶	jondurbin/airoboros-l2-70b-gpt4-1.4.1	70.9	70.4	87.8	70.3	55.2
🔶	TheBloke/llama-2-70b-Guanaco-QLoRA-fp16	70.6	68.3	88.3	70.2	55.7
🔶	upstage/llama-65b-instruct	70	68.9	86.4	64.8	59.7
🔶	stabilityai/StableBeluga1-Delta	68.7	68.2	85.9	64.8	55.8

huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

Spaces lmsys/chatbot-arena-leaderboard like 195 Running App Files Community 3

Leaderboard

| Vote | Blog | GitHub | Paper | Dataset | Twitter | Discord |

🏆 This leaderboard is based on the following three benchmarks.

- Chatbot Arena - a crowdsourced, randomized battle platform. We use 50K+ user votes to compute Elo ratings.
- MT-Bench - a set of challenging multi-turn questions. We use GPT-4 to grade the model responses.
- MMLU (5-shot) - a test to measure a model's multitask accuracy on 57 tasks.

💻 Code: The Arena Elo ratings are computed by this notebook. The MT-bench scores (single-answer grading on a scale of 10) are computed by fastchat.llm_judge. The MMLU scores are computed by InstructEval and Chain-of-Thought Hub. Higher values are better for all benchmarks. Empty cells mean not available.

Model	⭐ Arena Elo rating	📈 MT-bench (score)	MMLU	License
GPT-4	1206	8.99	86.4	Proprietary
Claude-1	1166	7.9	77	Proprietary
Claude-instant-1	1138	7.85	73.4	Proprietary
Claude-2	1135	8.06	78.5	Proprietary
GPT-3.5-turbo	1122	7.94	70	Proprietary
Vicuna-33B	1096	7.12	59.2	Non-commercial
Vicuna-13B	1051	6.57	55.8	Llama 2 Community
MPT-30B-chat	1046	6.39	50.4	CC-BY-NC-SA-4.0
WizardLM-13B-v1.1	1040	6.76	50	Non-commercial
Guanaco-33B	1038	6.53	57.6	Non-commercial
PaLM-Chat-Bison-001	1015	6.4		Proprietary
Vicuna-7B	1006	6.17	49.8	Llama 2 Community

On the role of LLMs in economics

How can LLMs change my workflow?

- Text as data by M. Gentzkow, B.T. Kelly, and M. Taddy: general introductory survey.
- Text algorithms in economics by E. Ash and S. Hansen: general introductory survey.
- A User's Guide to GPT and LLMs for Economic Research by K. Bryan: examples of how to use LLM in your daily research.
- Second half of `https://youtu.be/bZQun8Y4L2A` by A. Karpathy: nice tricks for good prompting.
- Language Models and Cognitive Automation for Economic Research by A. Korinek: application of LLM for ideation, writing, background research, data analysis, coding, and mathematical derivations.

How can I apply LLMs to learn about the economy?

- Hedonic prices and quality-adjusted price indices powered by AI by P. Bajari *et al.*: use product description text to predict product prices.
- Bloated Disclosures: Can ChatGPT Help Investors Process Financial Information? by A. Kim, M. Muhn, and V. Nikolaev: probe the economic usefulness of LLMs in summarizing complex corporate disclosures using the stock market as a laboratory.
- Asset Embeddings by X. Gabaix, R.S.J. Koijen, and M. Yogo: learn asset embeddings from investors' holdings data.
- Work2vec: Using language models to understand wage premia by S.H. Bana: uncover the premia associated with eight in-demand certifications.
- Out of One, Many: Using Language Models to Simulate Human Samples by L.P. Argyle *et al.*: using LLMs to synthesize data from undersample populations.

What are the effects of LLMs on the economy? (positive and normative)

- Large Language Models as Simulated Economic Agents: What Can We Learn from Homo Silicus? by J.J. Horton: LLM as an approximation of bounded-rational agents.
- Economics, Hayek, and Large Language Models by T. Cowen: a podcast about how LLM might change our conception of how economies work.
- Generative AI at Work by E. Brynjolfsson, D. Li, and L.R. Raymond.
- Preparing for the (Non-Existent?) Future of Work by A. Korinek and M. Juelfs.
- GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models by T. Eloundou, S. Manning, P. Mishkin, and D. Rock.
- Regulating Transformative Technologies by D. Acemoglu and T. Lensman.
- Power and Progress by D. Acemoglu and S. Johnson.

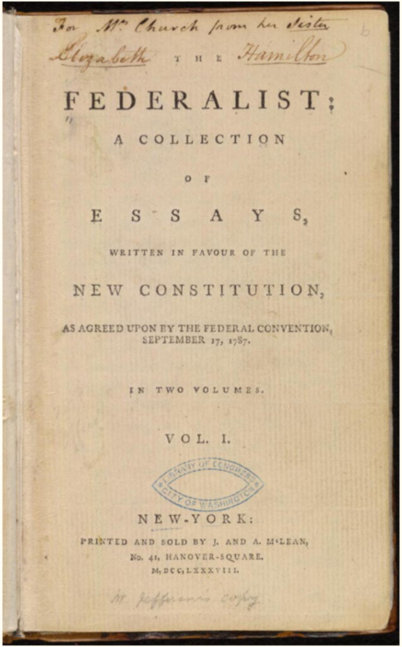
Text as data

Text is the new data

- Important for economics:
  1. Statements by policy makers.
  2. Political manifestos.
  3. Legal documents (court decisions, criminal records).
  4. Companies earning reports.
  5. Costumer complaints.
  6. Documents in libraries and archives.
  7. News, news commentary, and interviews.
  8. Verbal surveys.
  9. Opinion mining and sentiment analysis from social media.

How do we handle text?

- How do we use text in economic and statistical methods?
- Historically: reading the documents (or interviewing the authors)! But too slow, prone to errors and biases, and hard to replicate.
- Basic statistics: Inference in an Authorship Problem by Mosteller and Wallace (1963).
- Machine learning can help to extend the scope of text analysis.



For Mrs. Church from her Sister

Elizabeth THE Hamilton

FEDERALIST:

A COLLECTION

OF

ESSAYS,

WRITTEN IN FAVOUR OF THE

NEW CONSTITUTION,

AS AGREED UPON BY THE FEDERAL CONVENTION,
SEPTEMBER 17, 1787.

IN TWO VOLUMES.

VOL. I.

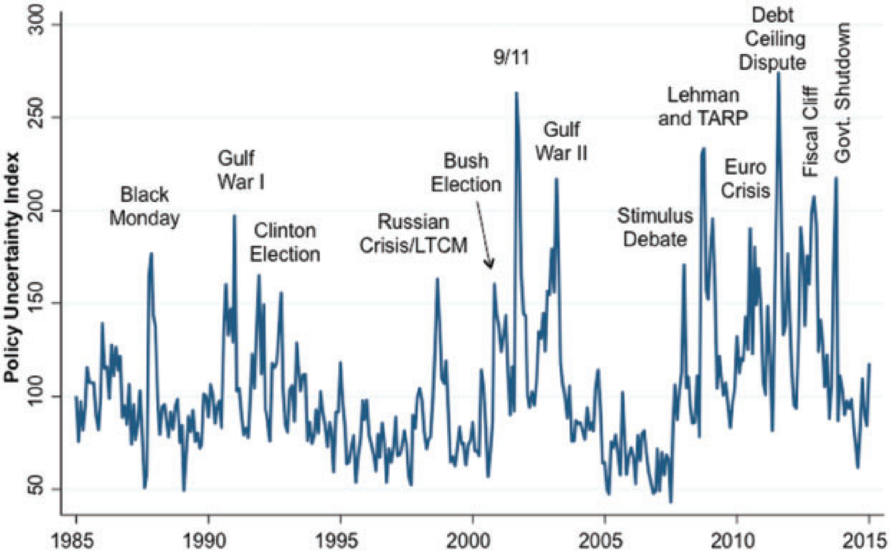
NEW-YORK:

PRINTED AND SOLD BY J. AND A. M'LEAN,
No. 41, HANOVER-SQUARE.
M,DCC,LXXXVIII.

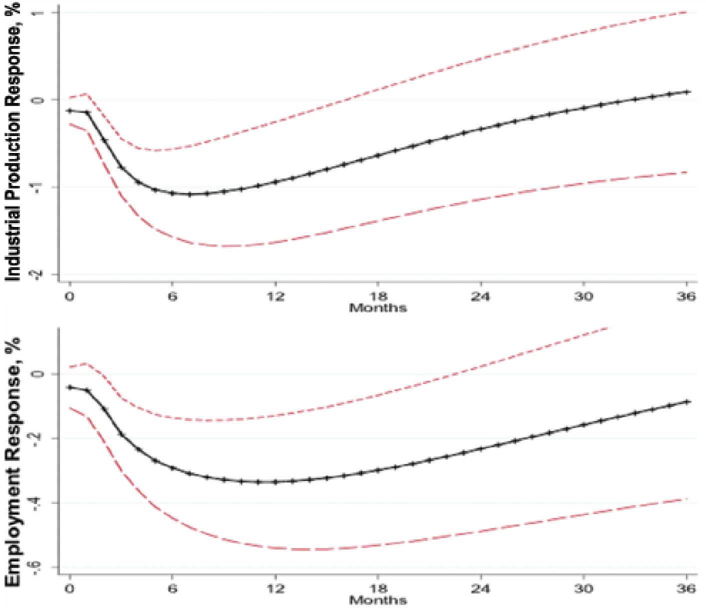
Mr. Jefferson's copy

Potential uses

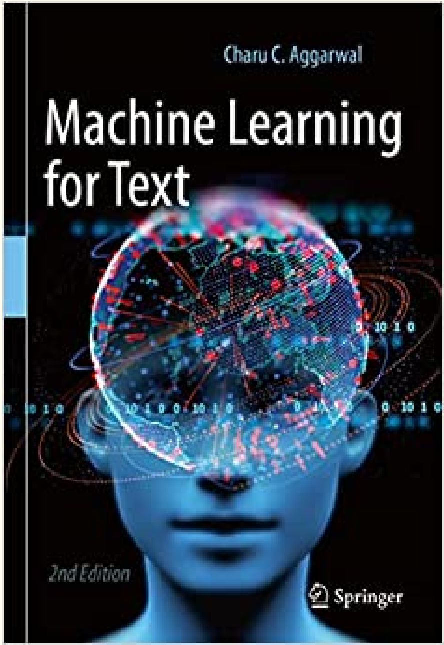
- Large area with many other applications in economics:
    1. Measurement.
    2. Prediction.
    3. Causality.

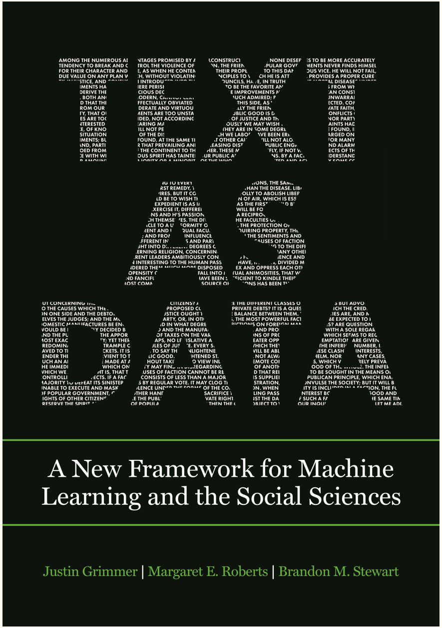


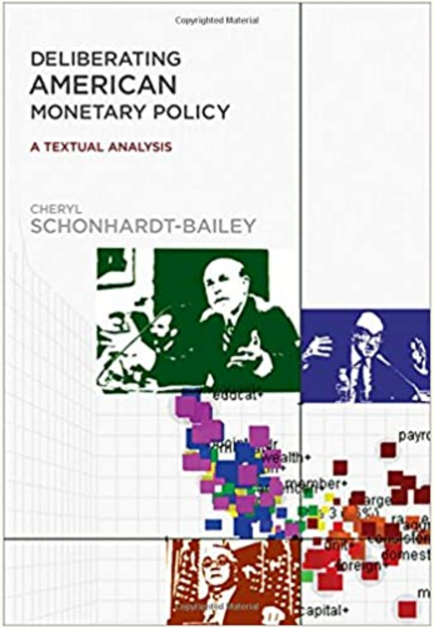
Index reflects scaled monthly counts of articles containing 'uncertain' or 'uncertainty', 'economic' or 'economy', and one or more policy relevant terms: 'regulation', 'federal reserve', 'deficit', 'congress', 'legislation', or 'white house'. The series is normalized to mean 100 from 1985-2009 and based on queries run on 2 February, 2015 for the USA Today, Miami Herald, Chicago Tribune, Washington Post, LA Times, Boston Globe, SF Chronicle, Dallas Morning News, NY Times, and the Wall Street Journal.



Industrial Production Response, %
1
0
-1
-2
0
6
12
18
24
30
36
Months
Employment Response, %
0
-.2
-.4
-.6
0
6
12
18
24
30
36
Months


Charu C. Aggarwal
Machine Learning for Text
2nd Edition
Springer

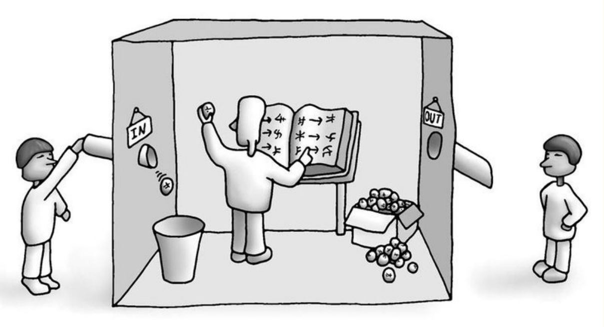

TEXT AS DATA
A New Framework for Machine Learning and the Social Sciences
Justin Grimmer | Margaret E. Roberts | Brandon M. Stewart

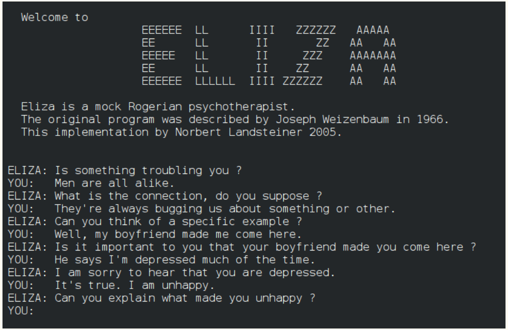

Copyrighted Material
DELIBERATING AMERICAN MONETARY POLICY
A TEXTUAL ANALYSIS
CHERYL SCHONHARDT-BAILEY

Natural language processing

Natural language processing

- Natural language processing (NLP): field specialized in how computers can deal with language as it appears in "natural" contexts (speech, text, ...).
- One of the very first applications of computers: Georgetown-IBM experiment in automatic translation in 1954.
- Classical NLP was based on symbolic rules (John Searle's Chinese room experiment and ELIZA) and Chomskyan theories of linguistics.
  - After some early success, the field stagnated.
- In comparison, modern NLP is built around statistical models.
  - Base of its recent success.


IN
OUT


Welcome to
EEEEEE LL   IIII   ZZZZZZ   AAAAA
EE   LL   II   ZZ   AA   AA
EEEEE   LL   II   ZZZ   AAAAAAA
EE   LL   II   ZZ   AA   AA
EEEEEE LLLLLL IIII ZZZZZZ   AA   AA
Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.
ELIZA: Is something troubling you ?
YOU: Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU: They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU: Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU: He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU: It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:

Modern language models refute Chomsky's approach to language

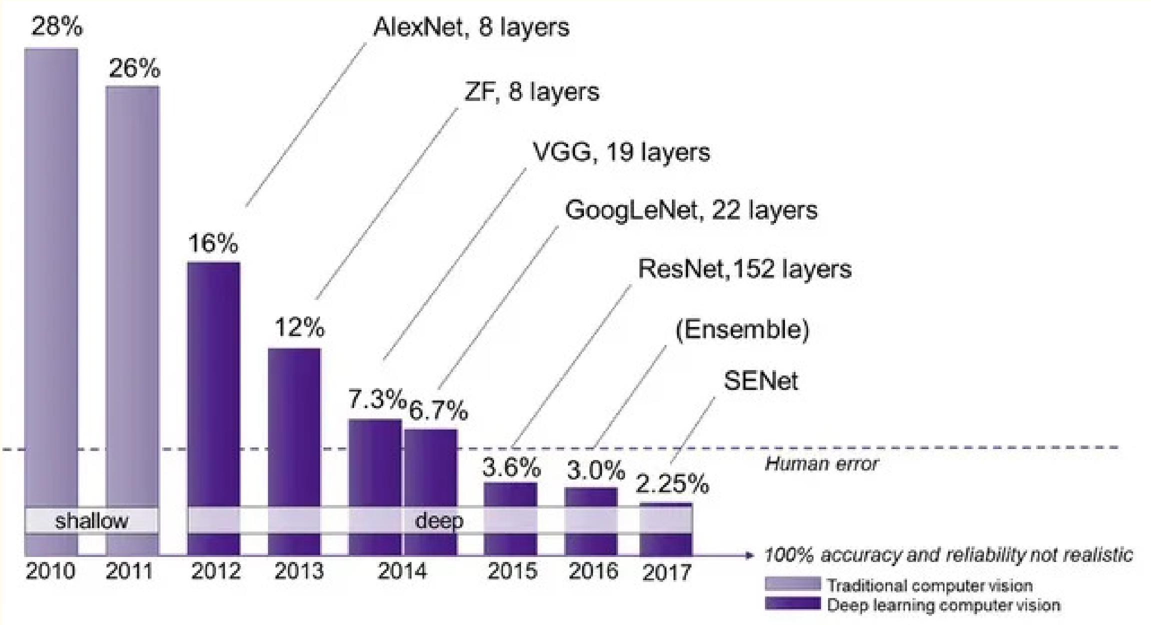
Steven T. Piantadosi[a,b]
[a]UC Berkeley, Psychology [b]Helen Wills Neuroscience Institute

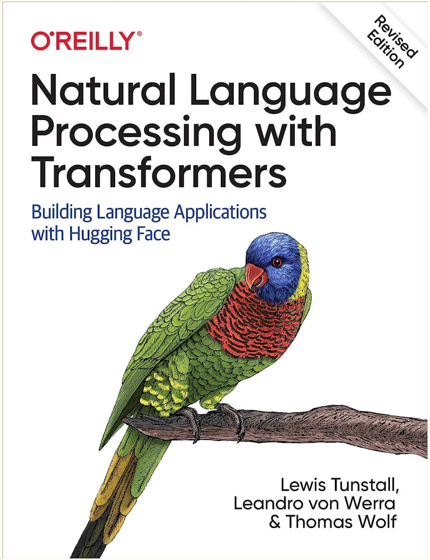
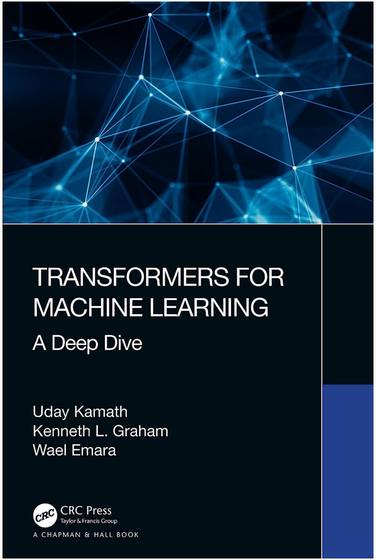
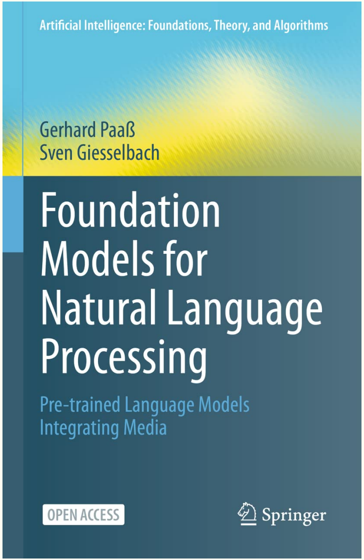
The rise and success of large language models undermines virtually every strong claim for the innateness of language that has been proposed by generative linguistics. Modern machine learning has subverted and bypassed the entire theoretical framework of Chomsky's approach, including its core claims to particular insights, principles, structures, and processes. I describe the sense in which modern language models implement genuine *theories* of language, including representations of syntactic and semantic structure. I highlight the relationship between contemporary models and prior approaches in linguistics, namely those based on gradient computations and memorized constructions. I also respond to several critiques of large language models, including claims that they can't answer "why" questions, and skepticism that they are informative about real life acquisition. Most notably, large language models have attained remarkable success at discovering grammar without using any of the methods that some in linguistics insisted were necessary for a science of language to progress.

The transformer model

The transformer model

- Deep convolutional neural networks, introduced in 2012, greatly impacted computer vision.
- But NLP (at the time, built around RNN and CNN) had lagged.
- Vaswani *et al.* (2017): Attention Is All You Need. Group of researchers affiliated with Google.
- 83,844 Google Scholar citations as of August 2, 2023.
- Transformers applied to other fields outside natural language processing (Visual transformers, DALL-E). In fact, anything that is set-to-set.
- Built around two ideas:
  1. (Self-)Attention.
  2. Encoder/decoder structure.


28%
26%
16%
12%
7.3%
6.7%
3.6%
3.0%
2.25%
AlexNet, 8 layers
ZF, 8 layers
VGG, 19 layers
GoogLeNet, 22 layers
ResNet,152 layers
(Ensemble)
SENet
Human error
shallow
deep
2010
2011
2012
2013
2014
2015
2016
2017
100% accuracy and reliability not realistic
Traditional computer vision
Deep learning computer vision


O'REILLY
Revised Edition
Natural Language Processing with Transformers
Building Language Applications with Hugging Face
Lewis Tunstall, Leandro von Werra & Thomas Wolf

TRANSFORMERS FOR MACHINE LEARNING
A Deep Dive
Uday Kamath
Kenneth L. Graham
Wael Emara
CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Artificial Intelligence: Foundations, Theory, and Algorithms
Gerhard Paaß
Sven Giesselbach
Foundation Models for Natural Language Processing
Pre-trained Language Models Integrating Media
OPEN ACCESS
Springer

Steps to build a transformer model

1. Formalizing text.
2. Text wrangling.
3. Tokenization.
4. Embedding.
5. Attention.
6. Output.
7. Training.
8. Extensions.

Step I: Formalizing text

Some terminology

- *Corpus*: the dataset under consideration (e.g., corporate reports, political speeches, statements, court decisions, newspaper articles, tweets, ...).
    - Third-declension neutral noun in Latin: nominative plural *corpora*.
- *Document*: each of the components of the corpus.
- *Terms*: each of the components of a document (usually words).
- *Ngrams*: Adjacent terms that we may want to handle together ("United States," "high unemployment").
- *Metadata*: covariates associated with each document (not always present).

What is text?

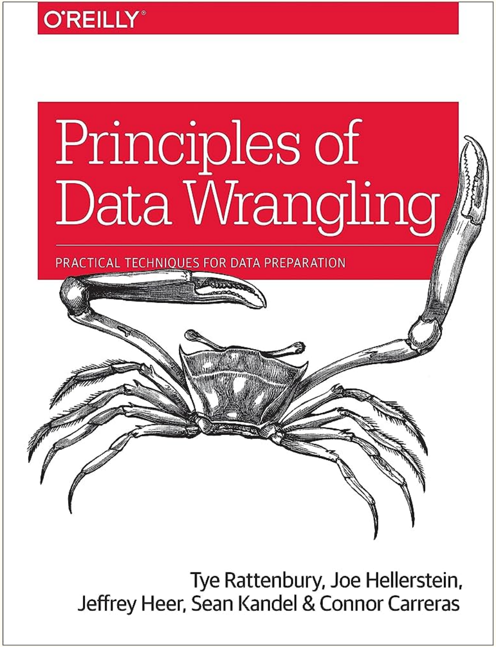
- Formally, a text is an ordered string of characters.
- Some of these may be from the Latin alphabet –'a', 'A' – but there may also be:

  1. Decorated Latin letters (e.g., ú).
  2. Non-Latin alphabetic characters (e.g., Chinese, Arabic, Hebrew).
  3. Punctuation (e.g., '!').
  4. White spaces, tabs, newlines.
  5. Numbers.
  6. Non-alphanumeric characters (e.g., '@').

Step II: Text wrangling

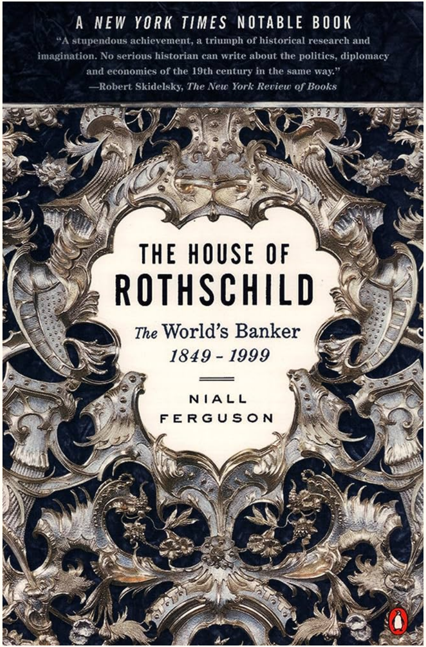
From files to databases, I

- First step is to *pre-process* strings to obtain a cleaner representation.
- This is often the "secret sauce of LLM."
- Rattenbury *et al.*, (2017) claim that between 50% and 80% of real-life data analysis is spent with data wrangling.
- Turning raw text files into structured databases is often a challenge:
  1. Separate metadata from text.
  2. Identify relevant portions of the text (paragraphs, sections, etc).
  3. Remove graphs and charts.
  4. Often, concerns about copyright, consent, safety, and privacy considerations.


O'REILLY
Principles of Data Wrangling
PRACTICAL TECHNIQUES FOR DATA PREPARATION
Tye Rattenbury, Joe Hellerstein,
Jeffrey Heer, Sean Kandel & Connor Carreras

From files to databases, II

- First step for non-editable files is conversion to an editable format, usually with optical character recognition (OCR) software.
- This is another potential application of deep learning.
- Check, for example: Shen *et al.* (2021), LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis.
- With raw text files, we can use regular expressions to identify relevant patterns.
- HTML and XML pages provide structure through tagging.
- If all else fails, manual extraction.


A NEW YORK TIMES NOTABLE BOOK
"A stupendous achievement, a triumph of historical research and imagination. No serious historian can write about the politics, diplomacy and economics of the 19th century in the same way."
—Robert Skidelsky, The New York Review of Books
THE HOUSE OF ROTHSCHILD
The World's Banker
1849 - 1999
NIALL
FERGUSON

Raw text files

The Quartz guide to bad data,
https://qz.com/572338/the-quartz-guide-to-bad-data/

I once acquired the complete dog licensing database for Cook County, Illinois. Instead of requiring the person registering their dog to choose a breed from a list, the creators of the system had simply given them a text field to type into. As a result this database contained at least 250 spellings of Chihuahua.

- Issues:
  1. Inconsistent spelling and historical changes.
  2. N/A, blank, or null values.
  3. 0 values (or -1 or dates 1900, 1904, 1969, or 1970).
  4. Text is garbled.
  5. Lines ends are garbled.
  6. Text comes from OCR.


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regular expressions I

- Regular expressions: sequence of characters that specifies a search pattern.
- You need to learn a programming language that manipulates regular expressions efficiently.
- About regular expressions in general:
  1. Tutorial: `https://www.regular-expressions.info/reference.html`.
  2. Online trial: `https://regexr.com/`.

Regular expressions II

- Modern programming languages have powerful regular expressions capabilities.
- In `Python`: `https://www.tutorialspoint.com/python/python_reg_expressions.htm`.
- In `R`: `https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf`.
    1. Key packages: `dplyr`, `stringr`, and `tidyr` part of `tidyverse`.
    2. In particular, learn to use the piping command from `dplyr` to make code more readable.
    3. Look also at `https://www.tidytextmining.com/` for text mining.

Step III: Tokenization

Tokenization

- Tokenization is splitting a raw character string into useful semantic pieces for processing called tokens.
- For example, we chop the string of characters:

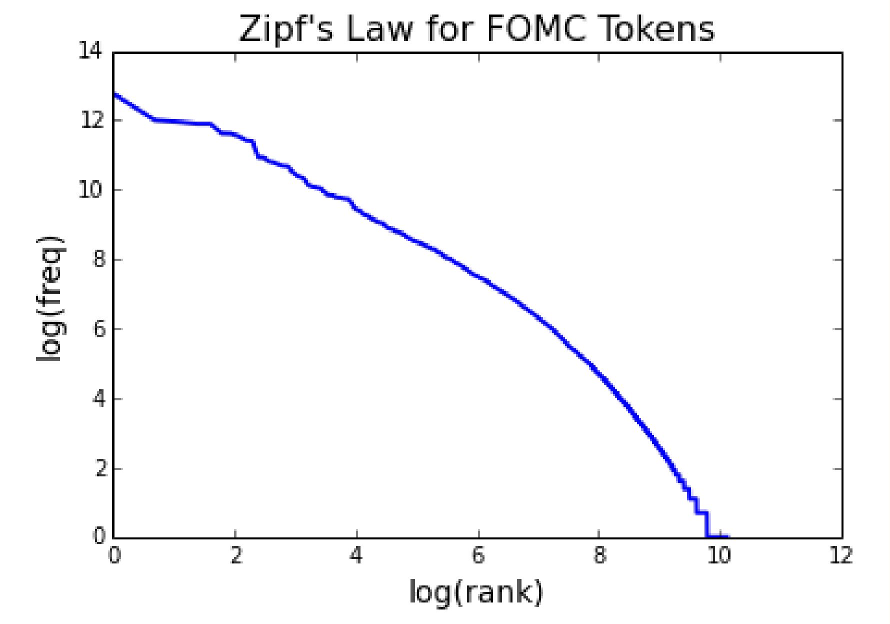
  "The European Central Bank is in Frankfurt"

  into

  "The", "European", "Central", "Bank", "is", "in", "Frankfurt".
- Often, tokens are words, but there may be characters, numbers, punctuation, and white spaces.
- Simple rules work well, but not perfectly. For example, splitting on white space and punctuation will separate hyphenated phrases, as in "risk-averse agent" and contractions, as in "aren't".
- While, in practice, one uses a specialized library for tokenization, it is important to understand tokenization in some more detail.

Vocabularies

- Tokenization relies on a vocabulary: a list of all allowed tokens.
- Oxford English dictionary $\approx$ 170k words in current use (vs. more than one mil ever used).
- We take advantage of that, in practice, we only use around 40k words (with a clear Zipf's law distribution). Other words are mapped into the 40k or masked as unknown.
- For specialized LLM, we might want to have specific vocabularies.
- How?
    1. Domain knowledge.
    2. Stop-words removal.
    3. Linguistic roots.
    4. Multi-word phrases.


Zipf's Law for FOMC Tokens
log(freq)
log(rank)
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12

Linguistic roots

- For many applications, the relevant information in tokens is their linguistic root, not their grammatical form (English in an inflected language). We may want to treat “prefer,” “prefers,” and “preferences” as equivalent tokens.

- Two options:

  1. *Stemming*: Deterministic algorithm for removing suffixes. Check, for example, `Porter stemmer`: `https://tartarus.org/martin/PorterStemmer/`.

     Stem need not be an English word: `Porter stemmer` maps ‘inflation’ to ‘inflat’.

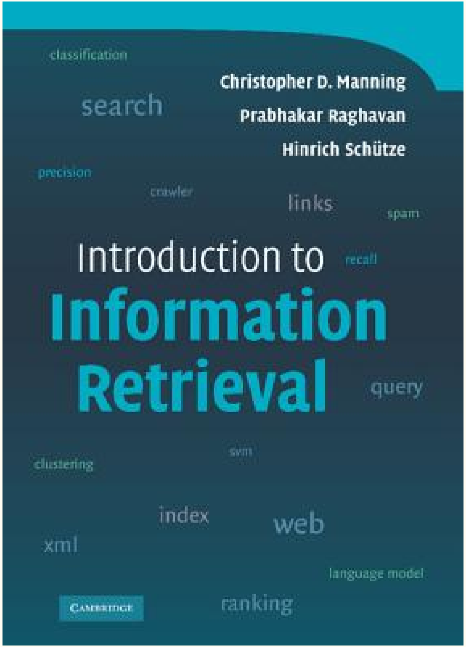
  2. *Lemmatizing*: Tag each token with its part of speech, then look up each (word, position) pair in a vocabulary to find the linguistic root.

     E.g., “saw” tagged as a verb would be converted to “see”, “saw” tagged as a noun left unchanged.

- A related transformation is *case-folding* each alphabetic token into lowercase. Not without ambiguity, e.g., “US” and “us” are each mapped into the same token.

Multi-word phrases

- Sometimes groups of individual tokens like "Banco de España" or "text mining" have a specific meaning.
- One ad-hoc strategy is to tabulate the frequency of all unique two-token (bigram) or three-token (trigram) phrases in the data and convert the most common into a single token.
- For example, in FOMC data, the most common bigrams include "interest rate," "labor market,"' "basi point"; most common trigrams include "feder fund rate," "real interest rate," "real gdp growth," "unit labor cost."


classification
Christopher D. Manning
search
Prabhakar Raghavan
Hinrich Schütze
precision
crawler
links
spam
Introduction to
recall
Information
Retrieval
query
clustering
svm
index
web
xml
language model
CAMBRIDGE
ranking

Tokenization in GPT-3

- GPT-3 tokenizer here: `https://platform.openai.com/tokenizer`.
- GPT-3 uses byte pair encoding (`https://github.com/openai/tiktoken`):
  1. Common words are a single token, less frequent words are represented by multiple tokens:

     "Enconding" is tokenized as "Enc" and "oding".
  2. Odd words are dropped.
- We assign every token an ID from a vocabulary with a total of 50257 tokens. For memory reasons, one may want to cap the vocabulary at $2^8 = 65536$ tokens.
- Example: "European Central Bank" $\rightarrow$ "European", "Central", "Bank" $\rightarrow [22030, 5694, 5018]$.
- More precisely, we represent each integer as a one-hot vector $w_{1\times 50227}$ with a 1 in the corresponding entry.

Step IV: Embedding

Embedding

- In natural language, words bundle in predictable patterns:

$$P(\text{Bank}|\text{European} + \text{Central}) \gg 0$$

$$\text{but}$$

$$P(\text{Giraffe}|\text{European} + \text{Central}) \approx 0$$

- This means we can use probabilities to generate predictions.
- We can capture this idea with an embedding: a representation of a token as a vector.
- We can estimate static embeddings with a simple logistic classifier (`Word2vec`).
- Useful for tasks such as document classification or sentiment analysis.
- However, static embeddings are not powerful enough for many interesting problems.
- We want more complex models that can incorporate contextual information.

Contextual embedding into vectors

- We take each token and embed it into a dense n-dim vector, to which we will add some context information.
- Why do we do this?
  1. Dimensionality reduction.
  2. More importantly: projection into a more informative space (interpretability?).
- Also, we usually do this in blocks of tokens: it will train the transformer to make predictions within the block.

Embedding in GPT-3

- GPT-3 uses input blocks of $m = 2048$ tokens (even if it needs to leave space empty): $B_{2048\times 50227}$ where each row is one token and a 12288-dim embedding.

- More concretely, we get a sequence-embeddings matrix:

$$E_{2048\times 12228} = B_{2048\times 50227} * W^{E}_{50227\times 12228}$$

  where $W^{E}_{50227\times 12228}$ is an embedding weight matrix (we will see later how we pick it).

- For example:

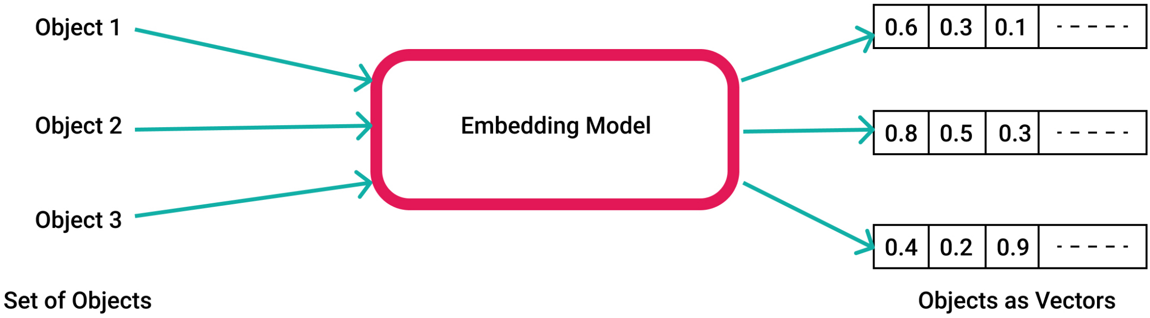
$$\text{“European”} \quad = [0.01, -0.99, \cdots, 0.34, 0.12]$$

Why is embedding key?

- Since we have a vector representation for each token, we can define standard vector operations by looking at the closest embedding:
    - Sum: Bank = European + Central

$$[0.03, -0.9, \cdots, 0.42, 0.36] \approx [0.01, -0.99, \cdots, 0.34, 0.12] + [0.02, 0.09, \cdots, 0.08, 0.24]$$

    - Subtraction: Frankfurt = European Commission + Brussels − European Central Bank.
- We can map any piece of information into an n-dimension vector. Whether there are tokens from text, pixels from photographs, Fourier weights from a recording, etc, is irrelevant → foundation models.


Object 1
Object 2
Object 3
Embedding Model
0.6 0.3 0.1 - - - - -
0.8 0.5 0.3 - - - - -
0.4 0.2 0.9 - - - - -
Set of Objects
Objects as Vectors

Encoding

- We mentioned before we want to incorporate context into our embedding.
- Distributional semantics: “A word is characterized by the company it keeps” (Firth, 1957).
- Think about the sentence: “I seat in the bank inside the bank office by the river bank where you bank.”
- We capture these relations by looking at the position of a token within a block: encoding.
- Quite different ways to do it, language-dependent (i.e., compare English, an analytic language, with Latin, a synthetic one!)

John Rupert Firth

Positional encoding in GPT-3

- We take the position of each token within the block $[0 - 2047]$ through 12288 sinusoidal functions, each with a different frequency.

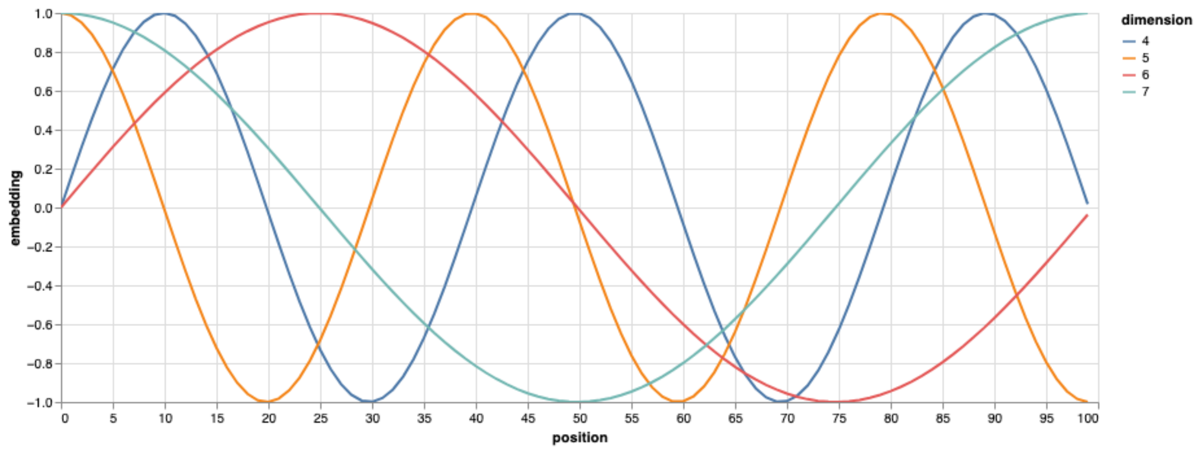
- Thus, we get a sequence-positional-encodings matrix:

$$S_{2048 \times 12228} = sin_{12288}(B_{2048 \times 50227})$$

  Extrapolate easily.

- We sum the sequence-embeddings matrix and sequence-positional-encodings matrix:

$$SE_{2048 \times 12228} = E_{2048 \times 12228} + S_{2048 \times 12228}$$


dimension
4
5
6
7
embedding
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
position

Step V: Attention

Attention

- Train the neural network to focus on some input data (e.g., some tokens) and lower the weights of other inputs by sharing communication among tokens.
- Mimics human cognition.
- Generalization of ideas floating since the 1990s (multiplicative modules, sigma pi units, and hyper-networks).
- Permutation invariant (unless we introduce positional encoding).
- Particularly easy to parallelize with GPUs because it avoids the previous approach of using recurrence.
- A more detailed introduction: https://www.youtube.com/watch?v=AIiwuClvH6k&ab_channel=GoogleDeepMind.

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.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Query, key, and value I

- We take a sequence $\mathbf{x} = (x_1, \ldots, x_m)$ of n-dim input vectors and produce a sequence $\mathbf{y} = (y_1, \ldots, y_m)$ of p-dim output vectors.
- p is the head size.
- With our previous example of GPT-3, $m = 2048$ and $n = 12288$.
- In a database, you have a query and obtain a value.
- Often, you want to have a key for each value.

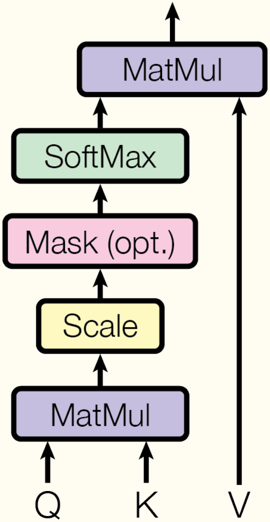
Query, key, and value II

- Every token emits a query ("what am I looking for?") and a key ("what do I contain?") vector.
- Three components:
  1. Q: query $Q_{m\times p} = \text{softmax}\left(SE_{m\times n}W^{Q}_{n\times p}\right)$.
  2. K: key $K_{m\times p} = \text{softmax}\left(SE_{m\times n}W^{K}_{n\times p}\right)$.
  3. V: value $V_{m\times p} = \text{softmax}\left(SE_{m\times n}W^{V}_{n\times p}\right)$.
- Importance matrix $\text{softmax}\left(QK^T\right)$ represents the relative importance of each token with respect to all others ("affinities").
- Then:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(QK^T\right)V$$

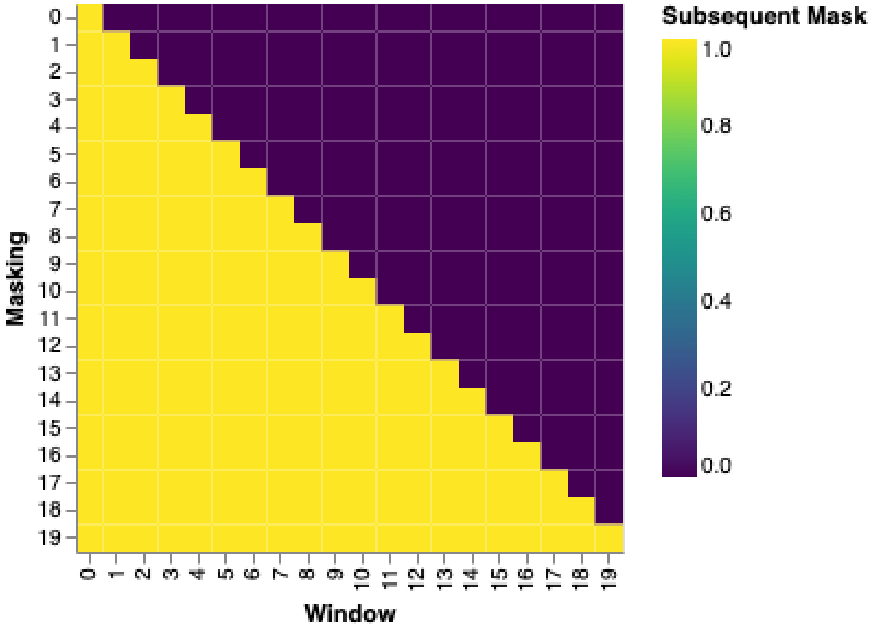
- You can think about $\text{Attention}(Q,K,V)$ as a refined embedding.

Scaled Dot-Product Attention

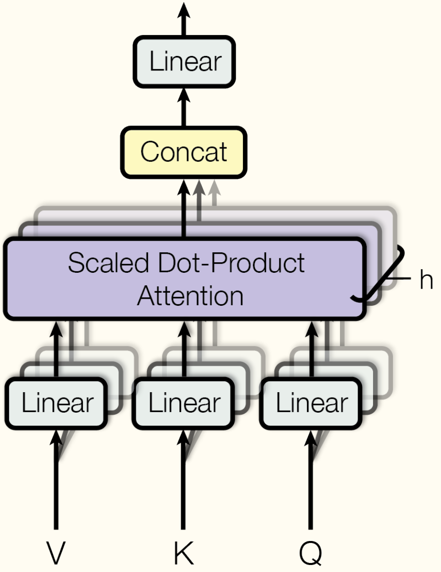


Scaling, masking, and multiheads

- Scaling: we can scale (QK^T) by $\sqrt{n}$ before applying softmax.
- Masking: some words in the input sequence are masked. Many possible reasons: GPT-3 to avoid having an encoder.
- Multiheads: We build multiple attention weights $W^{Q,r}, W^{K,r}, W^{V,r}$, where r is the index of the self-attention path.
- There is a simpler implementation of query, key, value: dot product.

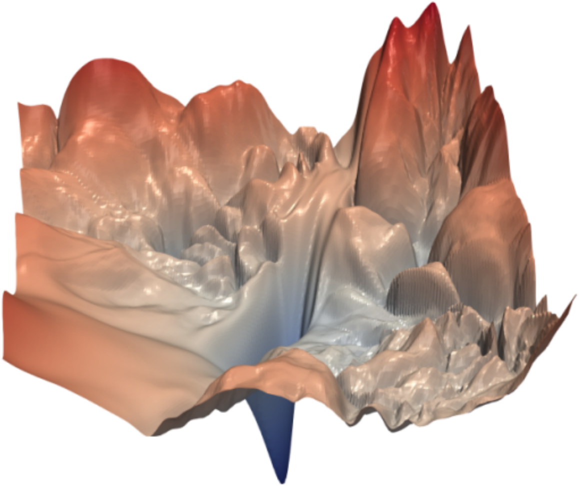

Subsequent Mask
1.0
0.8
0.6
0.4
0.2
0.0
Masking
Window
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Multi-Head Attention

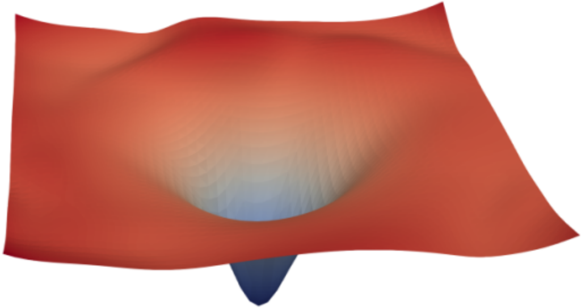


Scaling, masking, and multiheads in GPT-3

- In GPT-3, we have $p = 128$ and 96 attention weights for a total of 12228 (same as n).
- Also, we multiply by a new weight matrix W_O, add original SE, and normalize to get an output $\text{Attention}_{norm}(Q, K, V)_{2048\times 12228}$.
  1. Why sum? Skip connection (also known as a residual or shortcut connection).
  2. Why normalization?



(a) without skip connections



(b) with skip connections

Layer Normalization

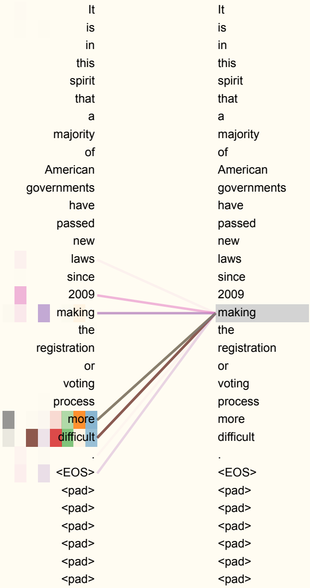
Jimmy Lei Ba
University of Toronto
jimmy@psi.toronto.edu

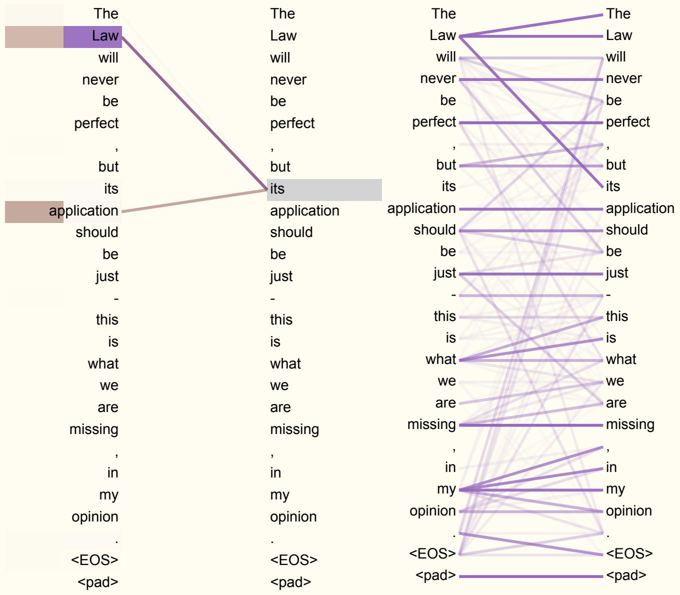
Jamie Ryan Kiros
University of Toronto
rkiros@cs.toronto.edu

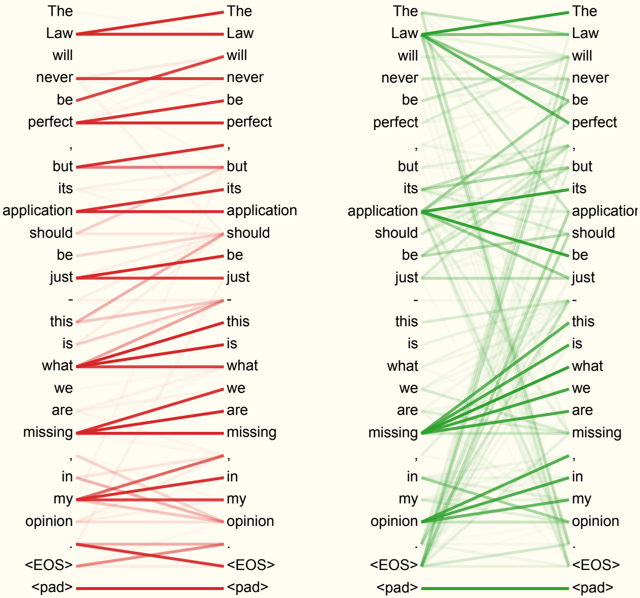
Geoffrey E. Hinton
University of Toronto
and Google Inc.
hinton@cs.toronto.edu

Abstract

Training state-of-the-art, deep neural networks is computationally expensive. One way to reduce the training time is to normalize the activities of the neurons. A recently introduced technique called batch normalization uses the distribution of the summed input to a neuron over a mini-batch of training cases to compute a mean and variance which are then used to normalize the summed input to that neuron on each training case. This significantly reduces the training time in feed-forward neural networks. However, the effect of batch normalization is dependent on the mini-batch size and it is not obvious how to apply it to recurrent neural networks. In this paper, we transpose batch normalization into layer normalization by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a *single* training case. Like batch normalization, we also give each neuron its own adaptive bias and gain which are applied after the normalization but before the non-linearity. Unlike batch normalization, layer normalization performs exactly the same computation at training and test times. It is also straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step. Layer normalization is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer normalization can substantially reduce the training time compared with previously published techniques.


It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>


The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>


The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>

Step VI: Output

Decoding

- Next, we pass the result $\text{Attention}_{norm}(Q, K, V)$ through a feed forward neural network with ReLUs to get $Y^F_{2048\times 12228}$.
    - Why? Forecasting.
- We sum $Y^E_{2048\times 12228} = \text{Attention}_{norm}(Q, K, V) + Y^F$ and normalize.
- Finally, we get Y_E with the inverse of our embedding weight matrix:

$$Y^E(W^E)^{(-1)}$$

- We apply softmax and select a word among the top-k probabilities.
- Also, we can use human alignment.

Step VII: Training

Practical implementation I

- GPT-3 uses 499 billion tokens in the full training data. The `Common Crawl` data set contains 410 of those.
- Loss function to select all the relevant weights: the average negative log-likelihood per token.
- Dropout.
- Powerful optimizer.
- Length of training vs. size of model and data.

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.

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

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

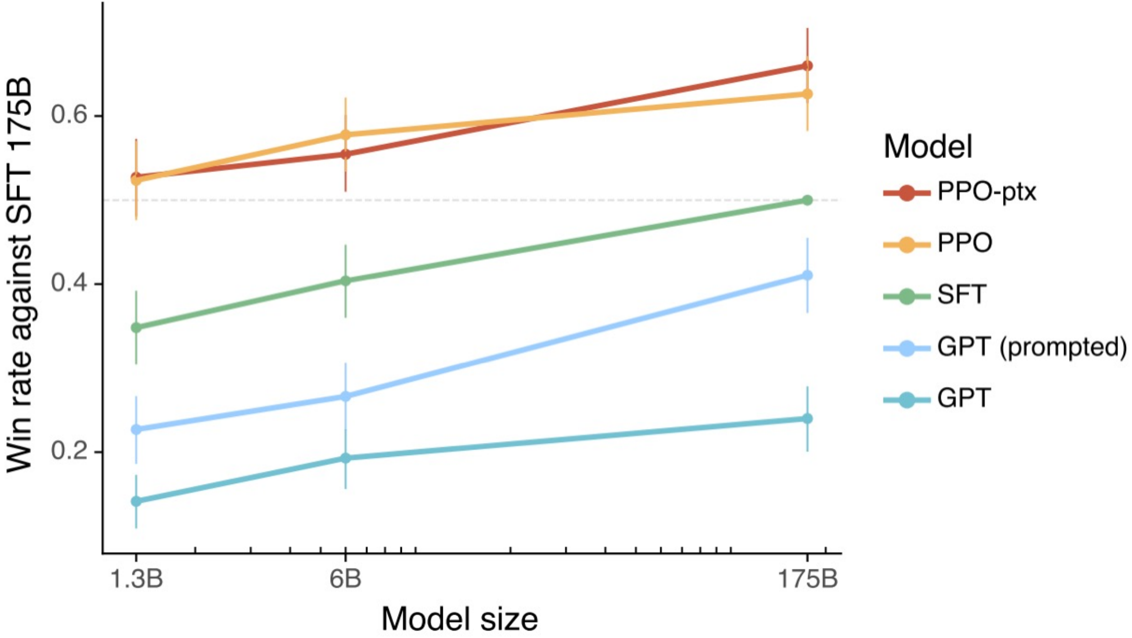
	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32			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

Practical implementation II

- Train and validation data.
- Different approaches:
  1. Supervised fine-tuning (SFT): The raw model is pre-trained on a large dataset and then trained on smaller but higher-quality datasets.
  2. Reinforcement Learning from Human Feedback (RLHF).
  3. Generating vs. ranking answers.
- Check `https://thegradient.pub/ai-is-domestification/`.

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM **can deploy this model**	1-100 GPUs days of training ex: Vicuna-13B **can deploy this model**	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude **can deploy this model**



Win rate against SFT 175B
0.6
0.4
0.2
1.3B
6B
175B
Model size
Model
PPO-ptx
PPO
SFT
GPT (prompted)
GPT

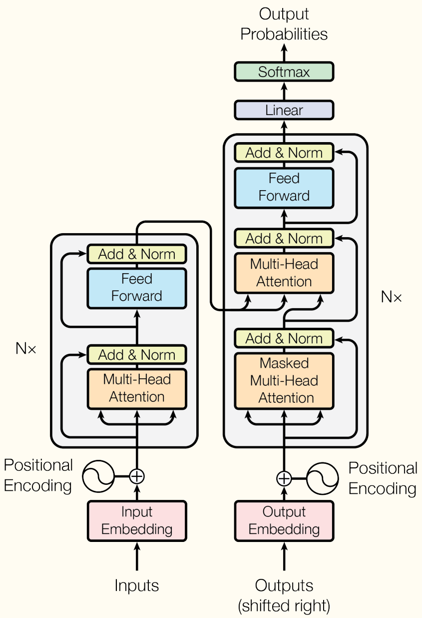
Code

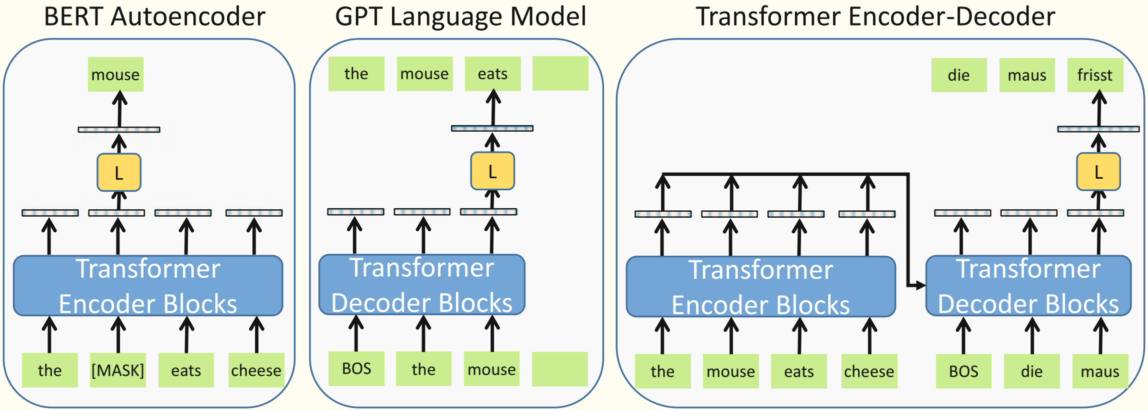
- `Python` + `PyTorch` allow for an easy implementation of this architecture.
- Code online:
  1. http://nlp.seas.harvard.edu/annotated-transformer/.
  2. https://www.youtube.com/watch?v=kCc8FmEb1nY and https://github.com/karpathy/nanoGPT.
- You want to run the code on GPUs.

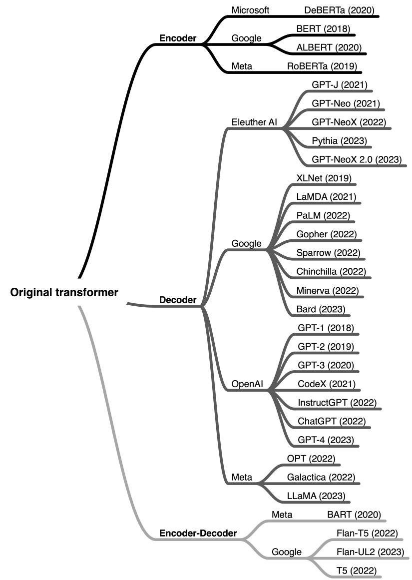
Step VIII: Extensions

Simplifying

- The original transformer architecture also has a decoder component. Why?
- It turns out we do not need an encoder or a decoder.
- We can dispense with one of the two.
  1. Autoencoders: BERT.
  2. Autoregressive language models: GPT.
- Cross-attention: Q's and K's come from outside sources of information.


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)


BERT Autoencoder
mouse
L
Transformer Encoder Blocks
the
[MASK]
eats
cheese
GPT Language Model
the
mouse
eats
L
Transformer Decoder Blocks
BOS
the
mouse
Transformer Encoder-Decoder
die
maus
frisst
L
Transformer Encoder Blocks
Transformer Decoder Blocks
the
mouse
eats
cheese
BOS
die
maus


Original transformer
Encoder
Microsoft
DeBERTa (2020)
Google
BERT (2018)
ALBERT (2020)
Meta
RoBERTa (2019)
Decoder
Eleuther AI
GPT-J (2021)
GPT-Neo (2021)
GPT-NeoX (2022)
Pythia (2023)
GPT-NeoX 2.0 (2023)
Google
XLNet (2019)
LaMDA (2021)
PaLM (2022)
Gopher (2022)
Sparrow (2022)
Chinchilla (2022)
Minerva (2022)
Bard (2023)
OpenAI
GPT-1 (2018)
GPT-2 (2019)
GPT-3 (2020)
CodeX (2021)
InstructGPT (2022)
ChatGPT (2022)
GPT-4 (2023)
Meta
OPT (2022)
Galactica (2022)
LLaMA (2023)
Encoder-Decoder
Meta
BART (2020)
Google
Flan-T5 (2022)
Flan-UL2 (2023)
T5 (2022)

Frontier work: QAs and Assistants

- QAs: either quote from a text or created from scratch.
- Hope to avoid domain knowledge.
- Design of assistants through prompt design and pre-train.
- Unfortunately, some of the details of frontier models are not public.
- But check: https://youtu.be/bZQun8Y4L2A.