Week 11: Introduction to Large Language Models

MY360/459 Quantitative Text Analysis

https://lse-my459.github.io/

Friedrich Geiecke

1. Introduction and Foundations
2. Quantifying Texts
3. Exploiting Word Meanings
4. Classifying Texts into Categories
5. Scaling Latent Traits Using Texts
6. *Reading Week*
7. Text Similarity and Clustering
8. Probabilistic Topic Models
9. Methods Review and Neural Network Fundamentals
10. Static Word Embeddings
11. Introduction to Large Language Models

Today

- Introduction to large language models (LLMs)
- At a fundamental level, current LLMs such as ChatGPT are simply autoregressive functions which iteratively predict the next word/token, starting with a sequence of input words/tokens

this is the LLM at the very basic level

$$\longrightarrow Pr(x_{T+1}|x_T, \ldots, x_1)$$

at some point the model will predict end of statement model

- The aim of today's lecture is to build a better understanding of these functions and how they work
- Given the time constraint of one lecture, different topics can only be discussed relatively briefly, but many resources for further study will be provided

Outline

- Tokenisation
- Fundamental architecture
- Pretraining
- Posttraining

Outline

- Tokenisation
- Fundamental architecture
- Pretraining
- Posttraining

From text to numbers

- Before we begin with language models, let's briefly review some basics in tokenisation
- How do we move from texts to numerical representations that are processed by the neural networks?

An imaginary dataset of four short patents

- Document 1: “invention”
- Document 2: “a good invention”
- Document 3: “a better invention”
- Document 4: “a less good invention”

Tokenisation - stylised example

- For simplicity, assume we tokenise at the word level, then there are five tokens in total: "a", "invention", "better", "good", "less"

- A tokenisation dictionary allows to map strings to integers and back

How do you move from text to numerical representation?

Tokenisation is the maping from integers to the words

```
{
    0: "a":
    1: "invention":
    2: "better"
    3: "good"
    4: "less"
}
```

ChatGPT gets a sequence of numbers as input and then predicts the next number

- Assuming this tokeniser, the document "a good invention" becomes `[0, 3, 1]`

Text encoding review

- As discussed in a previous week of the course, computers encode characters as sequences of bits (8 bits are one byte)
- Internally, a long text is therefore simply represented as a sequences of ones and zeros
- A common encoding system is e.g. Unicode which uses variable byte-lengths to encoder characters

Modern tokenisers

That is why tokens are parts of the words
- it bundles

- Idea of of modern byte pair encoding (BPE) tokenisers: Bundle commonly occurring consecutive bytes into tokens
- The current tokeniser of language models such as *GPT-4o* is *o200k_base* from the library *tiktoken*, it has ca. $\sim$200,000 unique tokens
- For example, "london school of economics" becomes $[75, 8552, 3474, 328, 48229]$
- `https://tiktokenizer.vercel.app/?model=o200k_base`

lower case 'london' is not the same token as 'London'

Discussions

- Tokenisation is an approximate process and improvements are an active area of research
- Some commonly discussed failure modes of LLMs are linked to tokenisation

counting a letter is not feasible for a model that
letters are not unit of operation for that model
-- its not working on character level but token

- For example, tasks like counting the letters in words or swapping first and last characters are tricky if the unit of operation are in fact tokens instead of characters themselves (illustration in lecture)

Tokens and embeddings

- Models like ChatGPT input and do not output words, but token ids
- The text prompt is transformed into tokens and the model's response from tokens back to text
- Inside the model, tokens are represented as embedding vectors
- Current neural network architectures commonly learn and update context-dependent token embeddings as a byproduct of training

These are vectors which is good news for us

Outline

- Tokenisation
- Fundamental architecture
- Pre-training
- Post-training

Neural language models

- Research on autoregressive neural language models is not new: Bengio et al. (2003) predicted the next word given a (short) sequence of previous words $Pr(x_{T+1}|x_T, ..., x_1)$

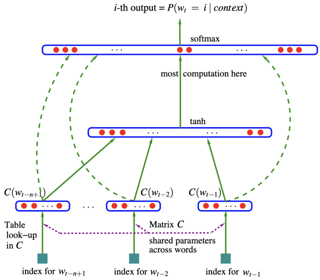
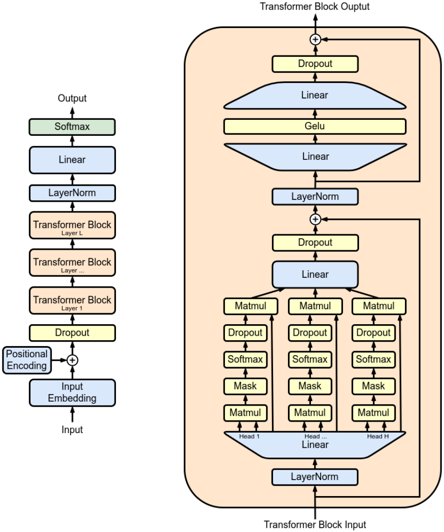


Figure 1 from Bengio et al. (2003)

Large language models

- Goal: Build and train $Pr(x_{T+1}|x_T, \ldots, x_1)$ using current methods
- Today's LLMs are based on the transformers which were pioneered in Vaswani et al. (2017)
- Some current frontier models parametrise $Pr(x_{T+1}|x_T, \ldots, x_1)$ with likely close to a trillion parameters
- The context which can be taken into account by their attention mechanisms, $x_T, \ldots, x_1$, can currently be up to $\sim$2,000,000 tokens in the case of e.g. Gemini Pro
- In the following, we discuss a simplified version broadly along the lines of the architecture of GPT-2 (Radford et al., 2019)

∼GPT-2 architecture



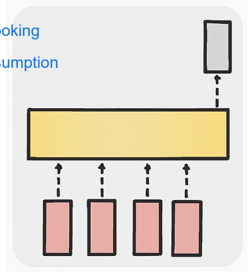
From `https://en.wikipedia.org/wiki/Generative_pre-trained_transformer`

Example

Greedy algorithm -- that is not forward looking

- current work is looking to relax that assumption

autoregressive model



- As an example, consider the sentence “Language models are interesting” and assume tokenisation at the word level
- Abstractly, we would like to learn a function (orange) that predicts (the probability of) the next token (grey)
- Today, these functions are transformers

Key components

x1 x2 x3 x4

language models are interesting

- Transformers have two key components:
  1. Attention mechanism
  2. Feed-forward neural network
- Building on our previous discussion of simple neural networks, the following slides discuss a forward pass through a simple (decoder) transformer

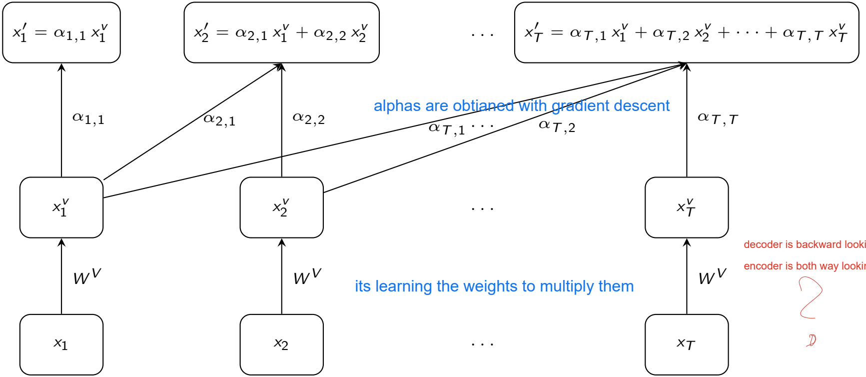
Example

- Input sentence: "Language models are interesting"
- Using the *tiktoken o200k_base* tokeniser, it would be split into an input sequence of $T = 4$ tokens: $[17238, 7015, 553, 9559]$
- The initial input embeddings simply come from a lookup table where we find (at first randomly initialised) embeddings, so we obtain the vectors $x_{17238}, x_{7015}, x_{553}, x_{9559}$
- For each of these vectors, we add a positional encoding: $x_t = x_{token\ id} + x_t^{pos}$
- The resulting embeddings x_1, x_2, x_3, x_4 enter the first attention block

Causal/masked self-attention

initalise w randomly

attention block



alphas are obtianed with gradient descent

its learning the weights to multiply them

decoder is backward looking

encoder is both way looking

- Given a value weight matrix W^V, x_t^v is a linear transform: $x_t^v = W^V x_t$
- x's after applying attention are simple weighted averages over these x^vs **earlier in the sequence**

  backward looking

  backward looking attention -- if it also looked at x4 it woul be full attention
- Attention could be fully connected, but is backward-looking here
- Weights $\alpha(W^Q, W^K, x)$ are actually functions of so-called query and key matrices (How?) which are learned through gradient descent alongside W^V

Feedforward neural network

- The second key component of the transformer is a simple feed forward network/MLP

- This adds a flexible non-linear transformation of embeddings
- Unlike in the attention block, each embedding vector x'_t is processed separately
- Combined weight matrices and biases of the network are denoted as W^{FF}, b^{FF} for easier notation

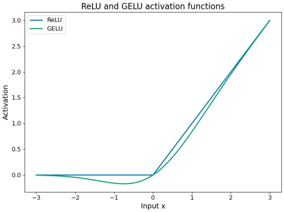
ReLU and GELU

- Transformers often use a GELU activation
- ReLU discussed previously: $f(x) = \max(0, x)$
- GELU:

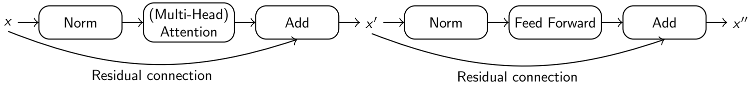
$$f(x) = x\,\Phi(x) \approx 0.5\,x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715\,x^3\right)\right]\right),$$

where $\Phi(x)$ is just the CDF of a standard normal distribution

- Retains small negative values and can lead to improved gradient flow



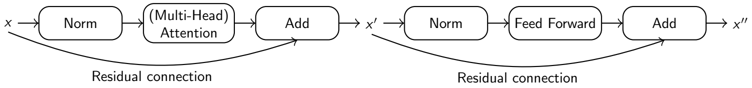
Combined into one transformer block



- Add norms and residual connections between x and x' or x' and x'' that were previously omitted
- Multi-head attention: Apply several attention heads in parallel and stack output vectors
- Each block has its own W_h^V, W_h^Q, W_h^K for all heads h, and W^{FF}, b^{FF} for the feed forward network
- For the next block, we simply set $x \leftarrow x''$

Forward pass on one slide

- Input: "Language models are interesting"
- Encode and look up static input embeddings: $x_{17238}, x_{7015}, x_{553}, x_{9559}$
- Add positional encoding: $x_t = x_{token\ id} + x_t^{pos}$
- Apply $N\times$ transformer blocks



- In the last block, transform the last embedding of a sequence into a vector with ∼200,000 dimensions (if we used the *o200k_base* tokeniser): $W^{fully\ connected} \cdot x_T^N = logits$
- Apply softmax to logits and sample next token

Training

- If we randomly initialise the parameters across the transformer, we could now make a (bad) prediction of the next token following an input sentence
- Parameters/weights across the transformer can be learned through gradient descent minimising a loss function

Outline

- Tokenisation
- Fundamental architecture
- Pretraining
- Posttraining

Pretraining

- Training of the LLM currently starts with a so-called pretraining stage, where the model is trained on very large amounts of curated internet text data to predict the next token
- Conceptually, predicting the next token is a classification problem and the loss function that is minimise with gradient descent is the cross-entropy loss discussed in previous weeks

Pretraining datasets

pretraining data is iteratively tested on high quality data, e.g. Wikipedia

perfect recall on all the data it has seen before

- Teams of researchers work only on assembling suitably high-quality internet datasets for pretraining
- One public example is `https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1`

After training, the model is able to generate texts in inference

- Step 1:
  - **Input (initial prompt):** [17238, 7015, 553, 9559] - "Language models are interesting"
  - **Sampled token:** [53081] - "because"
- Step 2:
  - **Input:** [17238, 7015, 553, 9559, 53081] - "Language models are interesting because"
  - **Sampled token:** [33574] - "they"
- Step 3:
  - **Input:** [17238, 7015, 553, 9559, 53081, 33574] - "Language models are interesting because they"
  - **Sampled token:** [553] - "are"
- etc.

Discussion

- So far we have obtained a base model which can auto-complete and simulate internet texts
- This pretraining stage can be conceptualised as a kind of lossy compression of internet text data into the model's floating point parameter values
- As an example, a Llama 2 model variant with $\sim$70 billion parameters was trained on $\sim$10 terabytes of internet text data. Its resulting parameters can be stored in a $\sim$140GB file
- Training required $\sim$6,000 GPUs for $\sim$12 days and may have roughly implied a total cost in the order of $2M (see Karpathy, 2023a, for further discussion)
- Llama variants allow to prompt base as well as instruct models (illustration in lecture)

Scaling laws

- Albeit not the eventual task, the loss of the base model turns out strongly correlated with many downstream capabilities
- It tends to decrease in compute, dataset size, and parameters

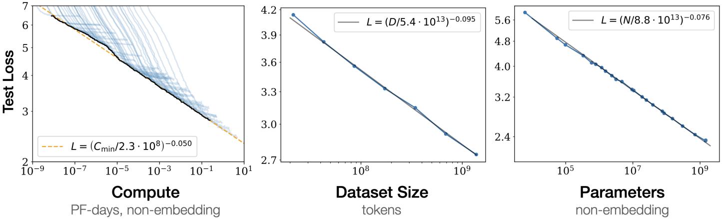


Figure 1 from Kaplan et al. (2020)

- One key question for the current AI paradigm is how long these laws may hold

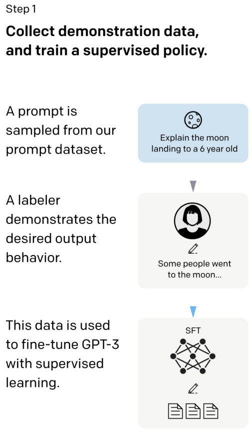
Outline

- Tokenisation
- Fundamental architecture
- Pretraining
- Posttraining
    - Supervised fine-tuning
    - Reinforcement learning

Supervised fine-tuning (SFT)

- The pretraining stage stored vast amounts of information in the model's parameter values
- How can we make the model answer questions instead of simulating internet documents?
- Repeat the same training objective – next token prediction – on much smaller sets of text containing prompts and answers

Example: SFT in Ouyang et al. (2022)



Part of Figure 2 from Ouyang et al. (2022)

Example: SFT in Ouyang et al. (2022)

- Three different GPT-3 models discussed in paper: 1.3B, 6B, and 175B parameters
- Dataset of 13k training prompts (from the API and labeler written)
- Create different types of prompt-answer pairs: Plain (labelers come up with task), few-shot (labelers give examples in prompt), user-based (labelers create prompt-answers matching desired use cases submitted to API + write answers to general prompts from API sandbox)
- The dataset is much smaller than in pretraining, but the model is able to generalise following instructions

Alignment

- This approach does not just train the model to respond to questions, it is also used to ethically align it, to make it refuse answering certain questions, etc.
- For an example of open-source SFT dataset, see e.g.: `https://huggingface.co/datasets/allenai/tulu-3-sft-mixture`
- Several challenges remain more broadly, e.g.:
  - Alignment to preferences in particular instructions, but not necessarily to others
  - Aligned models might still be tricked

Discussion

- Recent variants increasingly use synthetic data with AI-generated answers (Bai et al., 2022; Lee et al., 2023), but curation of SFT datasets is key and involves humans
- Users can fine-tune open-weight LLMs locally (subject to sufficient compute) and some providers of proprietary LLMs also allow users to fine-tune their LLMs via APIs given own prompt-answer pairs
- When using LLMs for own applications in research and other work, a common decision is between further fine-tuning or what is referred to as in-context learning
- By updating the model parameters, fine-tuning is fundamentally different to in-context learning

In-context learning

- Without updating the parameters, information can be shared with the model simply as part of the context/prompt
- Via its attention mechanism, it will take that information into account while generation answers
- This will change answers, but not through changing the underlying parameters/weights of the model
  - Zero-shot prompting: Give only the instructions in the prompt
  - One-shot prompting: Give one example as part of the prompt
  - Few-shot prompting: Give a few examples as part of the prompt

In-context learning - further details

- Models like ChatGPT have been instruction-trained with SFT and other approaches, however, it can still be very helpful to provide further examples in the prompt
- To give a stylised examples, in-context learning can even allow base models to respond as assistants in some highly simplified cases without SFT (illustration in lecture)

```
You are a helpful AI assistant answering questions.
User question: What is the abbreviation of the
London School of Economics and Political Science?
Your answer: LSE
User question: When was the LSE founded?
Your answer: 1895
User question: Why is the sky blue?
Your answer:
```

Hallucinations

- Their structure of simply predicting the next token makes LLMs vulnerable to factual inaccuracies
- Recall of information from their parameter values is limited
- Providing additional information as part of the context/prompt can therefore increase accuracy of answers
- This is essentially what retrieval augmented generation (RAG) does: Find relevant texts for a user question in a database and add them to the context of a chat LLM. The LLM can now respond more accurately with the added information in the content
- Another current approach to reduce hallucinations in LLMs is outlined on the next slide

Improving factual accuracy

- Some neurons in a model will likely indicate that it is uncertain about a question, but it has been trained to confidently reply through SFT examples by annotators which it now simulates
- The fundamental challenge here is to find out where knowledge boundaries of a particular model lie
- Approaches like the following can help to increase factual accuracy of LLM output:
  - Collect questions based on internet texts and see through repeated sampling of responses if the model knows the answer (illustration in lecture)
  - For those questions where the model gives wrong answers, manually write a response saying that is unaware of the answer
  - Then fine-tune the model on those question-answer pairs to teach it examples of responses if it does not know an answer

Tool use

- LLMs have knowledge cut-offs in their underlying trainning data, are not very good at counting, etc.
- Another way to improve their capabilities is *tool use*, e.g. running Python code, using a calculator, searching the web
- Models emit special token to use tools (illustration in lecture)
- Again this is taught to the model through examples of prompt-answer pairs in fine-tuning
- It is also linked to the previous discussion, because the model may use tools if it does not know answers

Good practices for SFT data to improve reasoning

- Lastly, SFT can also improve reasoning capabilities of models
- The key is to distribute the required computation over several tokens
- Good prompt-answer example to challenging questions therefore write out reasoning over several steps before arriving at the final answer

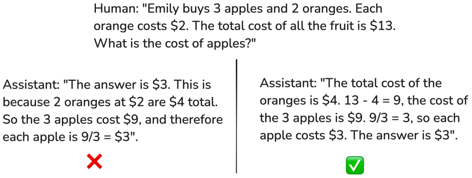


Figure from Karpathy (2025)

Reasoning discussion

- This is why ChatGPT and other LLMs respond with lists of steps to prompts
- It is not just to make answers more easily understandable for you, but they imitate response styles of annotators who wrote their responses in this way to help the model distribute its answers over many tokens
- But what are the best reasoning tokens for a model? They do not necessarily have to be the best for a human anotator who is curating the SFT data
- This is where a current frontier in the field lies, using methods from reinforcement learning to let the model discover own ways of reasoning

Outline

- Tokenisation
- Fundamental architecture
- Pretraining
- Posttraining
  - Supervised fine tuning
  - Reinforcement learning

Before we continue: Reinforcement learning (RL) basics

Figure: Based on Human-level control through deep reinforcement learning Mnih et al. (2015) and on Mnih et al. (2016), Screenshot from: https://youtu.be/xXP77QiHFTs

- Examples of RL applications: Playing Go, Chess, Shogi, or Starcraft (Silver et al., 2017, 2018; Vinyals et al., 2019), robotics (Abbeel et al., 2007), nuclear fusion (Degrave et al., 2022), LLMs such as the GPTs or DeepSeek (Ouyang et al., 2022; Guo et al., 2025)

More abstractly

- Through experimentation, the agent learns a policy function $\pi(A_t|S_t)$ to pick actions that maximise future rewards
- Observe state, choose action, observe reward and new state, chose new action, observe new state and reward, etc.

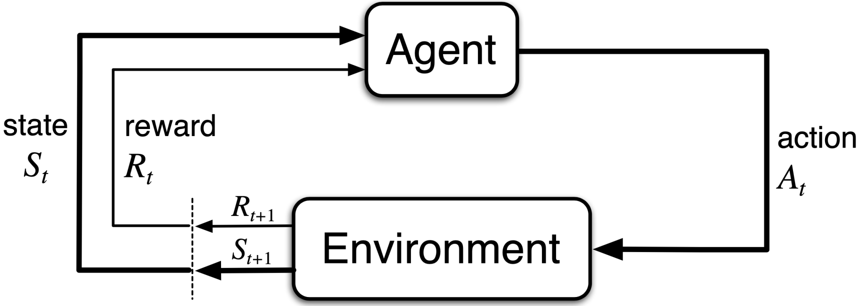


Figure: From Sutton and Barto (2020)

Language models as RL policy functions

- The language model $Pr(x_{T+1}|x_T, \ldots, x_1)$ can be conceptualised and relabelled as a policy function $\pi(A_t|S_t)$!
- It's state S_t are the prompt tokens, its action A_t the next token, the next state are prompt and the answer token combined $S_{t+1} = \{S_t, A_t\}$, etc.
- Abstractly, this is the same structure as in the Atari games or in Chess, Go, etc.
- To further train the LLM with RL, we are only missing rewards

Language models as RL policy functions

- Option 1: Train a separate reward model that predicts how much different LLM answers would be liked by humans (Ouyang et al., 2022). Issues: RL will learn to exploit inconsistencies in this reward model
- Option 2: Instead, ask the LLM to attempt problems with known answer such as in coding or math, and give a reward if the true answer is reached. Then train the LLM to reinforce those answers which solved problems
- In a nutshell, Option 2 is how current reasoning models such as DeepSeek are trained

Discussion

- LLMs has been initialised with pretraining and SFT to acquire world knowledge and some capabilities in question answering
- RL then refines the function $Pr(x_{T+1}|x_T, \ldots, x_1)$ to predict better-suited tokens for answering some complex questions in math or coding
- RL is a kind of advanced search algorithm with capabilities to find surprising actions, e.g. some previously unknown strategies in chess for AlphaZero (Silver et al., 2018)
- This is now applied to reasoning by choosing the next token in language instead of the next move in a game

Surprising findings in reasoning traces

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: <think>
To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both $\cdots$
$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$
Rearrange to isolate the inner square root term:
$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$
...
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$
We started with the equation:
$\sqrt{a - \sqrt{a + x}} = x$
First, let's square both sides:
$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$
Next, I could square both sides again, treating the equation: $\cdots$
...

Table 3 | An interesting "aha moment" of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

From Guo et al. (2025)

Reasoning tokens/time are increasing throughout RL training of LLM

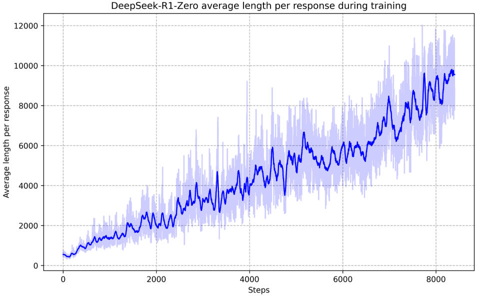


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

From Guo et al. (2025)

Discussion

- Illustration with classic example of Bayes' theorem in lecture
- Many open questions remain such as if reasoning capabilities from coding and math can generalise to and further improve domains with less clear answers/rewards such as summarisation etc.
- One area of current work is to create large high-quality datasets with questions across fields that have verifiable answers

Coding

- 01-llms-via-apis.Rmd

References for the slides and resources for further study

- Detailed applied lecture on large language models (Karpathy, 2025): `https://www.youtube.com/watch?v=7xTGNNLPyMI`
- Visualisation of the GPT model and attention mechanism (Sanderson, 2024): `https://www.youtube.com/watch?v=wjZofJX0v4M`
- Written introductions to transformers: `https://://jalammar.github.io/illustrated-transformer/` or `https://peterbloem.nl/blog/transformers`
- Full lecture series on building neural networks and language models from scratch (Karpathy, 2024): `https://www.youtube.com/watch?v=VMj-3S1tku0&list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ`

References I

Abbeel, Pieter, Adam Coates, Morgan Quigley, and Andrew Y. Ng, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," in B. Schölkopf, J. C. Platt, and T. Hoffman, eds., *Advances in Neural Information Processing Systems 19*, MIT Press, 2007, pp. 1–8.

Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon et al., "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, 2003, *3* (Feb), 1137–1155.

Choromanski, Krzysztof, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser et al., "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

Dao, Tri, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

References II

—, **Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré**, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in neural information processing systems*, 2022, *35*, 16344–16359.

Degrave, Jonas, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas et al., "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, 2022, *602* (7897), 414–419.

Guo, Daya, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

Kaplan, Jared, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

References III

Karpathy, Andrej, "Lecture: Intro to Large Language Models," 2023. https://www.youtube.com/watch?v=zjkBMFhNj_g.

— , "Stanford CS25: Introduction to Transformers w/ Andrej Karpathy," 2023. https://www.youtube.com/watch?v=XfpMkf4rD6E.

— , "Lecture series: Neural networks: Zero to hero," 2024. https://www.youtube.com/playlist?list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ.

— , "Lecture: Deep Dive into LLMs like ChatGPT," 2025. https://www.youtube.com/watch?v=7xTGNNLPyMI.

Lee, Harrison, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi, "Rlaif: Scaling reinforcement learning from human feedback with ai feedback," *arXiv preprint arXiv:2309.00267*, 2023.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv e-prints*, Feb 2016, p. arXiv:1602.01783.

References IV

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, "Human-Level Control through Deep Reinforcement Learning," *Nature*, feb 2015, *518* (7540), 529–533.

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray et al., "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, 2022, *35*, 27730–27744.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever et al., "Language models are unsupervised multitask learners," *OpenAI blog*, 2019, *1* (8), 9.

Sanderson, Grant, "Course: Neural networks," 2024. `https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi`.

References V

Shah, Jay, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao, "Flashattention-3: Fast and accurate attention with asynchrony and low-precision," *Advances in Neural Information Processing Systems*, 2024, *37*, 68658–68685.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton et al., "Mastering the game of go without human knowledge," *Nature*, 2017, *550* (7676), 354.

__, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel et al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, 2018, *362* (6419), 1140–1144.

Sutton, Richard S and Andrew G Barto, *Reinforcement learning: An introduction (2nd edition)*, MIT press, 2020.

References VI

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, 2017, *30.*

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, 2019, *575* (7782), 350–354.

Appendix

Dot product attention: Details on queries and keys

Back

- To compute e.g. $\alpha_{T,1}$:
  1. **Query vector:** Compute the query for token T
  $$x_T^Q = W^Q x_T$$
  2. **Key vectors:** For every token $t = 1, \dots, T$, compute
  $$x_t^K = W^K x_t$$
  3. **Attention weight:** Calculate
  $$\alpha_{T,1} = \frac{\exp\left(\frac{x_T^Q \cdot x_1^K}{\sqrt{d}}\right)}{\sum_{t=1}^{T} \exp\left(\frac{x_T^Q \cdot x_t^K}{\sqrt{d}}\right)}$$
  where d is the dimension of the key and query vectors
- All other weights $\alpha_{t,n}$ are compute analogously
- Attention block is characterised by W^V, W^Q, W^K

Dot product attention: Intuition

- Query: What information is a token seeking
- Key: What information is a token offering
- The dot product is a measure of similarity (scaled by vector lengths instead, it would have been the cosine similarity definition!)

Challenges

- Expensive in terms of computation and memory due to quadratic complexity - need to compute $T \times T$ attention weights (in case of fully connected attention)
- This makes it difficult to scale to very long long context windows
- More recent variants: Improve memory access (e.g. Dao et al., 2022; Dao, 2023; Shah et al., 2024), decrease quadratic complexity (e.g. Choromanski et al., 2020)