LLM Inference

Zdeněk Kasner

21 March 2024
Lesson plan

1. **Discussion** (10 min)
2. **(L)LM zoo** (5 min)
3. **Transformer inference** visualization (20 min)
4. **Decoding** algorithms (30 min)
5. **Text generation** hands-on (20 min)
Discussion
1. How many trained language models do you estimate to be publicly available?

2. What is the difference between inference, generation, and decoding?

Discussion time!
Warm-up: identify what is not a language model

BART  BERT  BigBird  ERNIE

HOMER  Optimus  MARGE  Megatron
Warm-up: identify what is **not** a language model

- BART
- **BERT**
- **BigBird**
- ERNIE

HOMER

Optimus

**MARGE**

Megatron

**not yet**
LANGUAGE MODELS ARE HOMER SIMPSON! Safety Re-Alignment of Fine-tuned Language Models through Task Arithmetic

Rishabh Bhardwaj

1 Singapore University of Technology and Design, 2 Nanyang Technological University

Abstract

Aligned language models face a significant limitation as their fine-tuning often results in compromised safety. To tackle this, we propose a simple method RESTA that performs LLM safety realignment. RESTA stands for Reestoring Safety through Task Arithmetic. At its core, it involves a simple arithmetic addition of a safety vector to the weights of the compromised model. We demonstrate the effectiveness of RESTA in both parameter-efficient and full fine-tuning, covering a wide range of downstream tasks, including natural language processing tasks.
Warm-up: identify what is **not** a language model

- Alpaca
- Camel
- Falcon
- Flamingo
- Koala
- Llama
- Orca
- Vicuna

Images from https://creazzilla.com/
Warm-up: identify what is **not** a language model

- Alpaca
- Camel
- Falcon
- Flamingo
- Koala
- Llama
- Orca
- Vicuna

*images from https://creazilla.com/*
Some numbers

- **555,743 models** uploaded on [HuggingFace](https://huggingface.co) (2024/03/18)
  - includes finetuned / scaled variants of the same base model
- **123 models** on [AlpacaEval Leaderboard](https://alpacaeval.com)
  - mostly instruction-tuned models
- **73 models** in the [LMSYS Chatbot Arena](https://lmsys.com)
  - mostly models finetuned for chat
- **1 model platform** to rule them all everyone knows (ChatGPT)
- See also [https://github.com/Hannibal046/Awesome-LLM](https://github.com/Hannibal046/Awesome-LLM)
LLM evolutionary tree

source: https://arxiv.org/abs/2304.13712
Transformer Inference
Inference
The concept of using a trained model for making predictions on new data (for classification, sequence tagging, text generation, ...).

Generation
The process of using a trained model for producing a sequence of tokens.

Decoding
The algorithm of turning the model’s internal representation into a sequence of tokens.
Transformer inference

https://bbycroft.net/llm

https://github.com/bbycroft/llm-viz

EQUATIONS

CODE

DIAGRAMS

ANIMATED 3D VISUALIZATIONS
Text Generation
Autoregressive decoding

Where are we:

- **Task:** Generating a **sequence of tokens**.
- **Tool:** A language model (LM) giving us a **probability distribution** over the vocabulary for a given prefix.
- **Method:** Feed the sequence prefix in the LM → Select the next token → Append the token to the prefix → Repeat.

= **Autoregressive decoding**
Autoregressive decoding

\[ t=1 \]

\[ \text{<BOS>} \quad \text{I} \]

\[ P(y_t | \text{<BOS>}, "I") \]

- a
- aardwark
- am
- ...
- I
- ...
- the
- walrus
- ...
- zyzzyva

which token to select?
Autoregressive decoding

\[ P(y_t | \langle BOS \rangle, \text{"I"}) \]

the most probable one?

The diagram shows the process of autoregressive decoding with a Transformer model. The first step is the initial state \( \langle BOS \rangle \), which is followed by the word "I". The Transformer then predicts the next word, and the process continues with subsequent words like "aardwark", "am", "walrus", "zyzzyva", and so on, until the most probable word is selected.
Autoregressive decoding

P(y_t | <BOS>, “I”, “am”)

Transformer

a
aardwark

am

... I

... the

walrus

... zyzzyva

t=2

<BOS>

I

am
Autoregressive decoding

\[ P(y_t | \langle \text{BOS} \rangle, \text{"I"}, \text{"am"}, \text{"the"}) \]

\[ t=3 \]

\[
\begin{align*}
\langle \text{BOS} \rangle & \rightarrow \\
\text{I} & \rightarrow \\
\text{am} & \rightarrow \\
\text{the} & \rightarrow \\
\end{align*}
\]

Transformer
Autoregressive decoding

t=4

<BOS> → I → am → the → walrus

Transformer
Have we generated **the most probable** sequence?

Do we **want** to generate the most probable sequence?
Decoding Algorithms
Decoding algorithms

**Finding the most probable sequence**
- MAP decoding
- Greedy search
- Beam search

**Minimizing unwanted behavior**
- MBR decoding

**Sampling a random sequence**
- Top-k sampling
- Top-p (nucleus) sampling
- Mirostat
- Typical sampling
Exact Inference = Maximum a posteriori (MAP) decoding

- Finding the most probable sequence (=mode of the LM distribution) given the step-wise factorization of sequence probability:

\[ y^* = \arg \max_{y \in \mathcal{Y}} P(y) = \arg \max_{y \in \mathcal{Y}} \prod_{i=1}^{t} P(y_i | y_1, \ldots, y_{i-1}) \]

- Intractable (exponential search space) → approximation algorithms
- The mode may even not be a good solution! ([1], [2])
  - e.g. an empty sequence
Greedy decoding

- Selecting the **most probable token** in each step $t$:
  
  \[ y_t = \arg \max_{y_t \in \mathcal{V}} P(y_t | y_1, \ldots, y_{t-1}) \]

- Very fast, often works satisfactorily (especially with LLMs)
- Non-parameteric
Beam search

- Parameter **k: number of sequences**
- Each step $t$:
  - Extend the sequences from the step $t-1$ with all possible tokens.
  - Select the $k$ most probable sequences for the step $t+1$.
- Tuning $k$:
  - $k=1$ == greedy decoding
  - larger $k$ → slower algorithm
  - $k>1$ allows re-ranking results
Minimum Bayes Risk (MBR) Decoding

- Selecting the sequence most similar to other sequences = “consensus decoding”

\[ y^* = \arg \max_{y_k \in \mathcal{Y}} \sum_{y_\ell \in \mathcal{Y} \setminus y_k} \text{sim}(y_k, y_\ell) \]

- Useful for minimizing pathological behavior
- Intractable → we need a sampling algorithm
- Application in automatic speech recognition and machine translation

source: Minimum Bayes Risk Decoding
Top-k sampling

- Selecting the token in each step randomly from $k \in \{1, \ldots, |V|\}$ most probable tokens
- The truncated distribution is re-weighed using softmax
- The shape of distribution can be adjusted using the temperature $T$:

$$\text{softmax}(y_i) = \frac{e^{y_i/T}}{\sum_{j \in V_{\text{top}-k}} e^{y_j/T}}$$

$t = 1$

prefix = “The”  
→ sampling from \{nice, dog, car, woman, guy, man\}

cum.prob. = 0.68

$t = 2$

prefix = “The car”  
→ sampling from \{drives, is, turns, stops, down, a\}

cum.prob. = 0.99
Top-p (nucleus) sampling

- Similar to top-k sampling, but with a variable $k$ in each step.
- Sampling from **"nucleus"**: set of the most probable tokens with combined probability summing to $p \in (0, 1]$
- The number of selected tokens is related to the “peakiness” of the distribution.

---

**prefix** = “The”

→ sampling from {nice, dog, car, woman, guy, man, people, big, house}

**cum.prob.** = 0.94

($>0.9$)

$t = 1$

---

**prefix** = “The car”

→ sampling from {drives, is, turns}

**cum.prob.** = 0.97

($>0.9$)

$t = 2$
Mirostat

- Aims to eliminate repetition and incoherent text in stochastic algorithms
- Adapting the $k$ parameter based on the desired text perplexity (“mirum” = surprise, “stat” = control)
- Parameters:
  - $\tau$ (tau) - the target perplexity
  - $\eta$ (eta) - learning rate

---

**Algorithm 1:** Adaptive top-$k$ sampling for perplexity control

```
Target cross entropy $\tau$, maximum cross entropy $\mu = 2 \cdot \tau$, learning rate $\eta$

while more words are to be generated do
    Compute $\hat{s}$ from (40): $\hat{s} = \frac{\sum_{i=1}^{N-1} t_i b_i}{\sum_{i=1}^{N-1} t_i}$
    Compute $k$ from (41): $k = \left( \frac{2\mu}{1-N-\tau} \right)^{\frac{1}{2}}$
    Sample the next word $X$ using top-$k$ sampling
    Compute error: $e = \mathcal{G}(X) - \tau$
    Update $\mu$: $\mu = \mu - \eta \cdot e$
end
```
(Locally) typical sampling

- Decodes text so that in each step, its perplexity is close to the perplexity of the model
  - Similar to Mirostat, but dynamic: the perplexity is not pre-specified
- Information theory: typical messages are the messages that we would expect from the process

\[ p(H) = 0.75 \quad H H H H \rightarrow \text{most probable sequence} \]
\[ p(T) = 0.25 \quad H T H H \rightarrow \text{typical sequence} \]
Other parameters

- **repetition_penalty** - discounting the scores of previously generated tokens
- **length_penalty** - promoting shorter / longer sequences in beam search
- ...and many more, see [HF docs](#) 

(we went through the most important ones, though)
Text Generation - hands-on
Text generation starter kit
Demo time

https://huggingface.co/docs/transformers/llm_tutorial
https://mlabonne.github.io/blog/posts/2023-06-07-Decoding_strategies.html
Bonus: Beyond Autoregressive Decoding
Bonus: Beyond Autoregressive Decoding

- **Non-autoregressive decoding**
  - [1]
  - can be parallelized
  - needs to predict output length
  - weaker links between output tokens
Bonus: Beyond Autoregressive Decoding

- middle ground between autoregressive and non-autoregressive decoding

Insertion Transformer [1][2]
**Bonus: Beyond Autoregressive Decoding**

- **Algorithm for sequence alignment**
- **Useful for automatic speech recognition**
- **Can be stacked on top of the non-autoregressive decoder**

---

**Connectionist Temporal Classification (CTC) layer [1][2]**
Bonus: Reverse-Engineering Decoding Strategies
Reverse-Engineering Decoding Strategies

Blackbox Access to a Language Generation System

Daphne Ippolito
dei@google.com

Nicholas Carlini
ncarlini@google.com katherinelee@google.com

Milad Nasr
miladnasr@google.com

Yun William Yu
yywu@math.toronto.edu

Abstract

Neural language models are increasingly deployed into APIs and websites that allow a user to pass in a prompt and receive generated text. Many of these systems do not reveal generation parameters. In this paper, we present methods to reverse-engineer the decoding method used to generate text (i.e., top-k or nucleus sampling). Our ability to discover which decoding strategy was used has implications for detecting generated text. Additionally, the process of discovering the decoding method makes it easier to detect whether a writing sample was generated by a language model or else was human-written (Ippolito et al., 2020). As generated text proliferates on the web, in student homework, and elsewhere, this disambiguation is becoming increasingly important.

Concurrent work to ours by Naseh et al. (2023) has developed similar strategies for detecting decoding

Stealing the Decoding Algorithms of Language Models

Ali Naseh
University of Massachusetts Amherst
Amherst, Massachusetts, USA
anaseh@cs.umass.edu

Mohit Iyyer
University of Massachusetts Amherst
Amherst, Massachusetts, USA
miyyer@cs.umass.edu

Kalpesh Krishna
University of Massachusetts Amherst
Amherst, Massachusetts, USA
kalpesh@cs.umass.edu

Anirou Hamansadr
University of Massachusetts Amherst
Amherst, Massachusetts, USA
amir@cs.umass.edu

ABSTRACT

A key component of generating text from modern language models (LM) is the selection and tuning of decoding algorithms. These algorithms determine how to generate text from the internal probability distributions generated by the LM. The process of choosing a decoding algorithm and tuning its hyperparameters takes significant time, manual effort, and computation, and it also requires extensive human evaluation. Therefore, the identity and hyperparameters of such decoding algorithms are considered to be extremely valuable to their owners. In this paper, we show, for the first time, that an adversary with typical API access to an LM can steal the type and hyperparameters of its decoding algorithms at very low monetary costs. Our attack is effective against popular LMs used in text generation APIs, including GPT-2, GPT-3 and GPT-Neo. We demonstrate the feasibility of stealing such information with only a few dollars, e.g., $98.8, $1, $4, and $46 for the four versions of GPT-3.

https://aclanthology.org/2023.inlg-main.28/

Bonus: Reverse-Engineering Decoding Strategies

- With **API access**, answers to certain questions (dice rolls, months, ...) can be used to **estimate the \( k \) and \( p \) parameters** of the stochastic algorithms

> *ChatGPT only returns 14 of the 20 options for a 20-sided dice roll*

- Distinguishing between **top-k** and **top-p**: If two prompts yield very different predictions of \( k \), then top-k is probably not used.

- With access to **model’s full distribution**, we can distinguish also between other algorithms (greedy vs. beam search vs. top-k vs. top-p ...)

NPFL140 - LLM Inference
• Huggingface models
• Awesome LLM: curated list of resources
• Transformer inference: 3D visualization
• Huggingface decoding algorithms overview
• Huggingface text generation strategies (includes a few extra ones)
• Common pitfalls when generating text with LLMs
• Visualizing decoding strategies
• Minimum Bayes Risk decoding
Further reading

- **On Decoding Strategies for Neural Text Generators (Wiher et al., 2022)**
  - Language generation tasks vs. decoding strategies.

- **If beam search is the answer, what was the question? (Meister et al., 2020)**
  - Why does beam search work so well?

- **Understanding the Properties of Minimum Bayes Risk Decoding in Neural Machine Translation (Muller and Sennrich, 2021)**
  - When can MBR be useful?