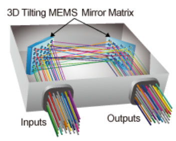
TCS for LLMs

Dean Foster (@foster) and Dhruv Madeka (maded@)

November 5, 2023



Real LLM discussion involve hardware

What makes modern LLMs work:

- GPUs
- cache efficient access
- bandwidth between caches
- communication between devices and instances

Real LLM discussion involve hardware

What makes modern LLMs work:

- GPUs
- cache efficient access
- bandwidth between caches
- communication between devices and instances
- ⋮
- Lasers

Real LLM discussion involve hardware

What makes modern LLMs work:

- GPUs
- cache efficient access
- bandwidth between caches
- communication between devices and instances
- ⋮
- Lasers

It's hardware, hardware, and more hardware

What can theory add?

Examples of cool theory:

1. "Auto-Regressive Next-Token Predictors are Universal Learners"
2. "SGD learning on neural networks: leap complexity and saddle-to-saddle dynamics"
3. saddle point escape
4. Many papers on two layer network theory
5. Many paper on the first step of SGD
6. μP
7. Matyroshka

And only 6 and 7 offer practical advice

Our goal: Useful theory

This talk: useful TCS for LLMs

I'll present 3 short ideas with implications for real NNs:

1. complexity of chain of thought
2. trap door functions
3. statistical degrees of freedom

Idea #1:

Chain of thought

Bad question:

Is $\sqrt{2\pi} \stackrel{?}{>} e$?

Good question:

Work out both sides of $\sqrt{2\pi} \stackrel{?}{>} e$, then say if it is true.

Best question:

Take a deep breath and work out both sides of $\sqrt{2\pi} \stackrel{?}{>} e$, then say if it is true.

Theorem (Merrill and Sabharwal 2023)

An LLM can only answer questions in TC(0) if asked directly for the answer. (arxiv)

Theorem (Merrill and Sabharwal 2023)

An LLM can only answer questions in TC(0) if asked directly for the answer. (arxiv)

Theorem (Daniel Hsu 2023)

An transformer LLM can answer the “two sum” problem, but to answer a “three sum” requires it to be extremely wide. (personal communications)

Theorem (Merrill and Sabharwal 2023)

An LLM can only answer questions in TC(0) if asked directly for the answer. (arxiv)

Theorem (Daniel Hsu 2023)

An transformer LLM can answer the “two sum” problem, but to answer a “three sum” requires it to be extremely wide. (personal communications)

Theorem (F. and Madeka 2023, Folk theorem 2024)

Using chain of thought reasoning, an LLM can solve any problem in PSPACE.

Implication #1:

Feed the out of one NN into another NN during training

Tiered model

- Bottom tier:
  - training: usual transformer model
  - Generates “roll outs” (starting every 50 words or so)

Tiered model

- Bottom tier:
  - training: usual transformer model
  - Generates “roll outs” (starting every 50 words or so)
- Middle tiers:
  - training: Using history and rollout, predict next word
  - generates new roll outs

Tiered model

- Bottom tier:
  - training: usual transformer model
  - Generates “roll outs” (starting every 50 words or so)
- Middle tiers:
  - training: Using history and rollout, predict next word
  - generates new roll outs
- Top tier:
  - Reads all roll outs and history
  - training: predictions the next word
  - inference: uses predictions to generate actual word

Idea #2:

One way functions

One way functions

A one way function is one where $f(x)$ is easy to compute, but $f^{-1}(y)$ is hard to compute.
Examples:

- Cryptography
- Effectively random functions
- P vs NP

Causal mask

We process words sequentially in a transformer LLM.

- Not as extreme as say in a LSTM
- Still, all values are "time stamped"
  - Every node in a transformer has a time stamp
  - It only depends on tokens that came before that time stamp
- Say more...

Extremely small embedding

Theorem

Suppose each layer i has nodes t such that $N_{i,t} = f(N_{i-1,t}, N_{i-1,t-1})$. Suppose further that $N_{i,t} \in R^1$. Then there exists polynomials with low complexity that take exponential computation under this restriction.

Extremely small embedding

Theorem

Suppose each layer i has nodes t such that $N_{i,t} = f(N_{i-1,t}, N_{i-1,t-1})$. Suppose further that $N_{i,t} \in R^1$. Then there exists polynomials with low complexity that take exponential computation under this restriction.

Example of what is going on:

- $b_t = b_{t+1}^2$
- Easy to compute from right to left
- takes one multiply at each step
- computing left to right requires raising to power x^{2^T}

Harder example:

- $b_t = \alpha_t + \beta_t b_{t+1} + \gamma_t b_{t+1}^2$
- Easy to compute from right to left
- computing left to right is a very complex polynomial

Extremely small embedding

Theorem

Suppose each layer i has nodes t such that $N_{i,t} = f(N_{i-1,t}, N_{i-1,t-1})$. Suppose further that $N_{i,t} \in R^1$. Then there exists polynomials with low complexity that take exponential computation under this restriction.

Key point:

- It needs to have a small context window
- Any fixed size will have hard examples
- We can compute R2L easy, but L2R is hard

Extremely small embedding

Theorem

Suppose each layer i has nodes t such that $N_{i,t} = f(N_{i-1,t}, N_{i-1,t-1})$. Suppose further that $N_{i,t} \in R^1$. Then there exists polynomials with low complexity that take exponential computation under this restriction.

How to attack the theorem:

- copy all data to the time t
- do all the computation
- Now as easy as R2L, but requires a huge embedding dimension

Implication #2:

"encoder" plus transformer network

Insert description here

Idea #3:

Statistical batch vs computational batch

Statistics independence

Palm masked out the first 10% of their tokens in every batch.

- Worked with a batch of 2000 tokens
- $Y_1, \ldots, Y_{t-1}$ used to predict Y_t
- But only for $t = 201, 202, \ldots, 2000$
- First 200 tokens not predicted in this batch

Statistics independence

Our encoder / decoder trick:

- Encodes 9000 tokens
- predicts next 1000 tokens
- First 9000 not predicted in this batch

Implication #3:

Stride length $\neq$ window length

Statistics independence

Trick to use more data:

- L = batch size
- S = "stride" (the number of predictions made)
- Use batchs 0, L

Summary

I present:

1. complexity of chain of thought
2. trap door functions
3. degrees of freedom

Summary

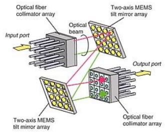
I present:

1. complexity of chain of thought
2. trap door functions
3. degrees of freedom

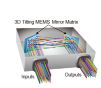
This argued for the following modifications to LLM foundation models:

- rollout aware network
- An encoder-decoder model {**Dean:** *What word do we use to replace encoder?*}
- better sampling of tokens

THANKS!



Pointers (we'll drop this .pdf in the chat)



MEMS

- Big bench: 100s of hard problems.
- PaLM and PaLM2 solve BigBench and professional exams
- Magical hour talk (Sebastien Bubeck)
- Magical 15 minute talk (Kahn of Kahn academy)
- Prompt Engineering (Andrew Ng's Prompt engineering)
- nanoGPT on github (build an LLM from scratch in 2 hours)

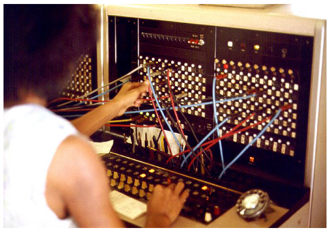
LLM requirements:
- Compute: 1000s of GPUs

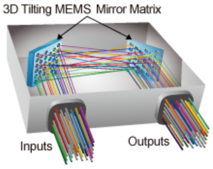
LLM requirements:
- Compute: 1000s of GPUs

(Need about 1000 to 10,000 A100s or H100s for 3 or 4 months.
So maybe 3 million dollars up to 100 million dollars.)

LLM requirements:
- Compute: 1000s of GPUs
- Communication: TBytes/s

LLM requirements:
- Compute: 1000s of GPUs
- Communication: TBytes/s





MEMS

LLM requirements:
- Compute: 1000s of GPUs
- Communication: TBytes/s
- Code: about 1000 lines

LLM requirements:
- Compute: 1000s of GPUs
- Communication: TBytes/s
- Code: about 1000 lines

nanoGPT github/video

LLM requirements:

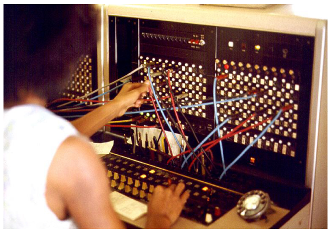
- Compute: 1000s of GPUs
- Communication: TBytes/s
- Code: about 1000 lines

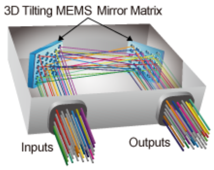
LLM find patterns

$$
\begin{aligned}
\overline{L}(\text{random guessing}) &= 15 = \log_2(60,000) \\
\overline{L}(\text{unigrams word frequency}) &= 11.7 = \log_2(3300) \\
\overline{L}(\text{bigrams (aka Markov)}) &= 8.8 = \log_2(500) \\
\overline{L}(\text{gzip (LZ compression)}) &= 8.2 = \log_2(300) \\
\overline{L}(\text{small LLM}) &= 7.5 = \log_2(200) \\
\overline{L}(\text{Humans}) &\approx 4 \\
\overline{L}(\text{LLM}) &= 3.6 = \log_2(12)
\end{aligned}
$$

(All in bits per token. I did the small LLM. Shannon, Cover/King did the human subjects estimation.)

Point to point is faster than packets





MEMS