High Performance LLMs From First Principles (2024)

Session 1
https://github.com/rwitten/HighPerfLLMs2024

rwitten@

Goal: learn how to achieve **high performance** for LLMs

Program

Predict

Profile

Program (write code in Jax)

Predict (roofline on napkin or spreadsheet)

Profile (run code, compare to predictions)

This class: we will build an LLM from scratch in Jax that achieves high performance for training and inference.

(But the concepts apply everywhere! All NNs, all High Performance Computing, all hardware and all frameworks!)

#1 feedback last time:
“Things made sense when you were talking but then I forgot”

Lesson: there will be exercises between classes!

My Asks

Please ask lots of questions! Just raise your hand or speak up!

If there are topics you're interested in, message me between sessions.

Join the discord! https://discord.gg/2AWcVatVAw

Do the exercises! Give feedback, ask questions!

Website: https://github.com/rwitten/HighPerfLLMs2024

My Background

- Lots of work with accelerators over the years!
  - Started with TensorFlow in 2016 and PyTorch in 2017
  - Plenty of ML but some more exotic things as well:
    - TorchScript and TensorRT and even a little MXNet over the years!
    - “General-purpose GPU” – using GPUs for high performance, non-ML work.
    - On-device ML for robotics (Jetson series)
  - Trainium Compiler at AWS
- Now:
  - At Cloud TPU, driving development for MaxText and related projects:
    - “Largest Job” publicly demonstrated
    - “Accurate Quantized Training” (1.4x speedup by matmuls in int8)
  - In Jax!

Your Background

- Comfortable programming in Python
- Some ML experience (can be TensorFlow, PyTorch, Jax, another framework, or from scratch)!
- Access to a machine with either Cloud TPUs or Nvidia GPUs.
    - If you're using Nvidia GPUs, you'll need to be a little scrappier in getting set up, etc.
    - (You can even follow along in Jax CPU if you're a bit brave! If do you, reach out to me, I'm happy to advise.)

What We'll Cover

- Over ~10 sessions
  - Jax LLM Implementation From Scratch
  - Single Chip Rooflines And Compilation
  - Distributed Computing via Sharding
  - LLM Training – what happens under the hood, rooflines, sharding, optimization
  - LLM Inference – what happens under the hood, rooflines, sharding, optimization
  - Attention Deep Dive – flash, vLLM, continuous batching, etc.
  - Covered Throughout:
    - ML: Quantization, Checkpointing, Data Loading, Numerics
    - Practical Tips: Debugging, Overlapping Jax Kernels
    - Larger scale: Goodput
    - Fancy stuff: Ahead of Time Compilation
    - Going deeper: shard map, pallas.

Why Jax?

- (Hopefully you'll be convinced after the class.)
- Jax
  - Pythonic, flexible and beautiful framework.
  - Easiest way to write performant code! End-to-end compilation and optimization!
  - Incredibly powerful multi-chip and multi-host story.
  - Separation between hardware and software means Jax produces high performance code on CPU, Nvidia GPU and TPU.
    - And we hope more platforms in the future!
  - Compare cutting edge codebases!
    - https://github.com/google/maxtext
    - https://github.com/NVIDIA/Megatron-LM
  - Key insight: Jax was the only framework invented after end-to-end compilation became state of the art. (And you'll learn why that matters soon!)

Today: Program

Predict

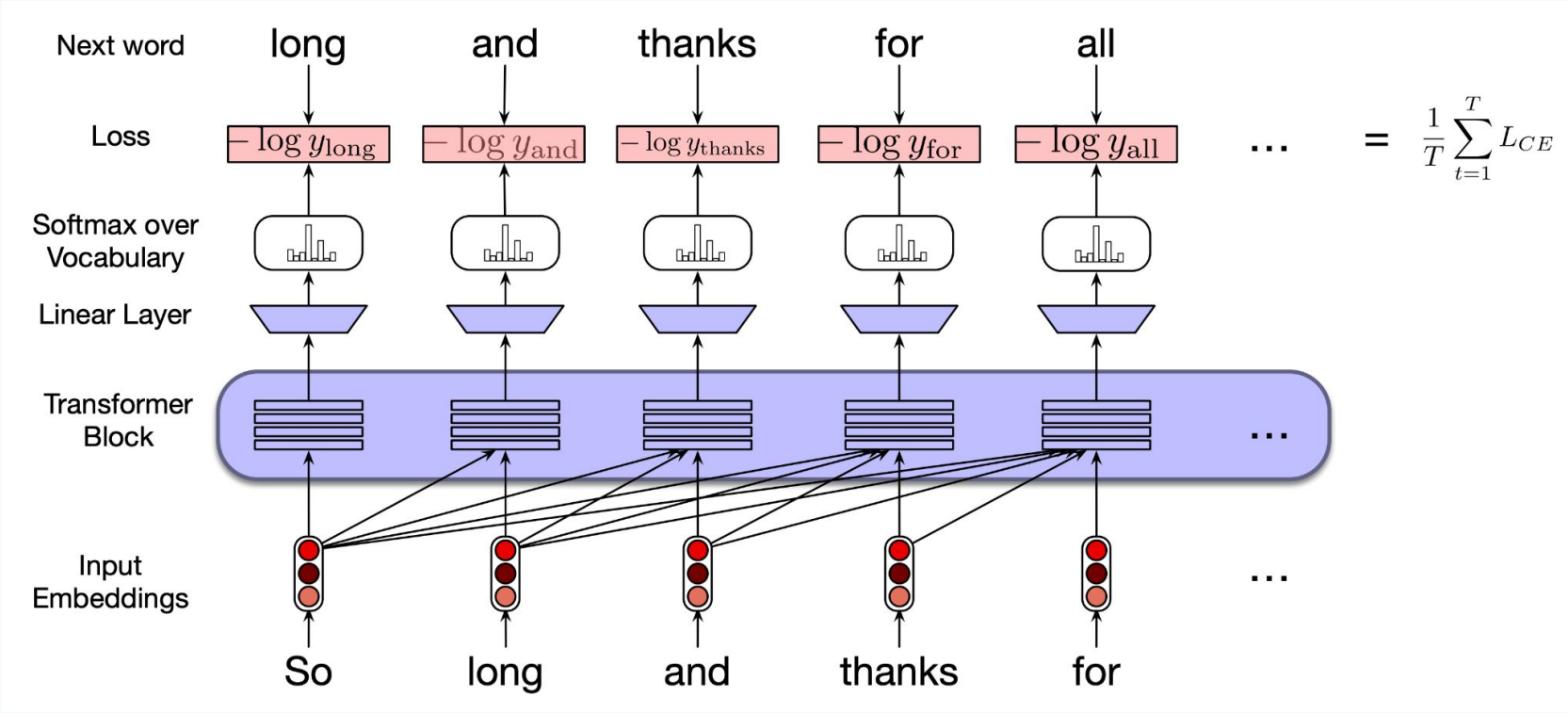
Profile

Let’s Write An LLM and Train Loop!

Introduction To Transformer – Components

- Slightly simplified today!
- Define Task
- Explain:
  - Embedding
  - Positional encoding
  - Feed-forward and activation
  - Attention
  - Unembedding
  - Loss
- Not covering:
  - Normalization

Introduction To LLM – Task



- Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright © 2023. All rights reserved. Draft of January 7, 2023.

Introduction To LLM – Data loading

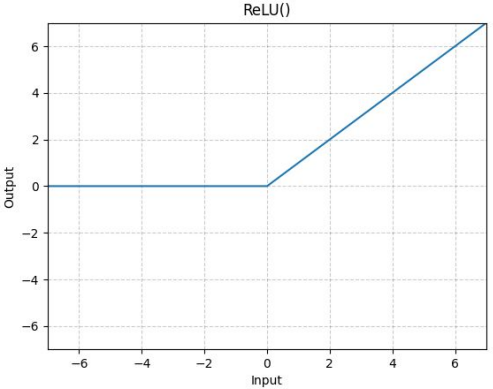
- First:
  - we get an input
    - [“So”, “long”, “and”, “thanks”, “for”]
  - And want to output
    - [“long”, “and”, “thanks”, “for”, ?]

Introduction To LLM – Tokenization & Embedding

- We get an input [“So”, “long”, “and”, “thanks”, “for”]
- We need to tokenize this. Think about word to number:
  - [17, 13118, 6, 19912, 4]
  - (In our examples, we will do next character prediction using old school ASCII 256 character code.)
  - (Different dictionaries are different but frequently the vocab is ~100K)
- And then we need to “embed”
  - This is a lookup table – each index gets its own vector.
  - This embedding table is [VOCAB, EMBED]

Introduction To LLM – Multilayer Perceptron (MLP)

- Each block has as input of
  - BATCH, SEQUENCE, EMBED
- FF >> EMBED (Typically FF/EMBED = 4. We'll learn more about this in the future.)
- Now we need to do a matrix multiply to:
  - An (EMBED, FF) matrix
  - (BATCH, SEQUENCE, FF) = (BATCH, SEQUENCE, EMBED) * (EMBED, FF)
- Next a nonlinearitiy – RELU
- Now we need to do a matrix multiply to:
  - An (FF, EMBED) matrix
  - (BATCH, SEQUENCE, EMBED) = (BATCH, SEQUENCE, FF) * (FF, EMBED)
- Next a nonlinearitiy – RELU #at least for our simplified setup
- See ReLU plot (thanks to PyTorch for the picture)



Introduction To LLM – Attention

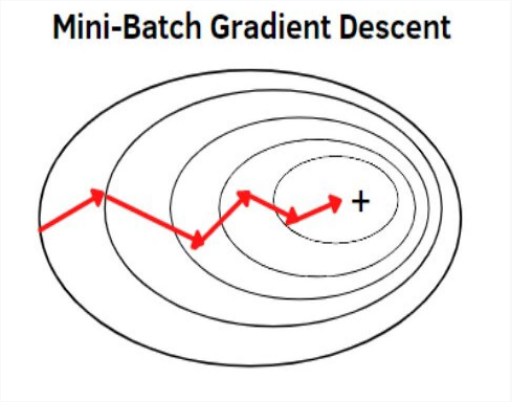
- Everything we've seen is separable across tokens – i.e. tokens don't see each other.
- Instead of "So long and thanks" -> "for" this would be "thanks" -> "for".
    - (Sometimes this would be called a unigram model)
- Attention is the solution.
- We aren't going to explain that. So I will just skip it today! We'll give attention it's whole own section!

Introduction To LLM – Output

- Now we have an output of [BATCH, SEQUENCE, EMBED]
- Now we go through another [EMBED, VOCAB] matrix
- Map to [BATCH, SEQUENCE, VOCAB]
- We need to score how well [BATCH, SEQUENCE, VOCAB] matches the desired output.
    - Which is [BATCH, SEQUENCE]
- This is cross entropy!
    - p(guesses, j) = exp(guesses[j]) / sum(guesses(l))

Introduction To LLM – Train Loop

- We use Optax to get an optimizer.
  - For every parameter, most modern optimizers have 2 extra params.
- Then we get loss and gradients (thank you Jax for grad).
- And we make an optimizer update!



Picture: https://statusneo.com/efficientdl-mini-batch-gradient-descent-explained/

Let’s Write An LLM and Train Loop! – Done

Thanks!
Ping me (rwitten@google.com) with feedback, suggested topics, etc!