Lecture 24 — Large Language Models

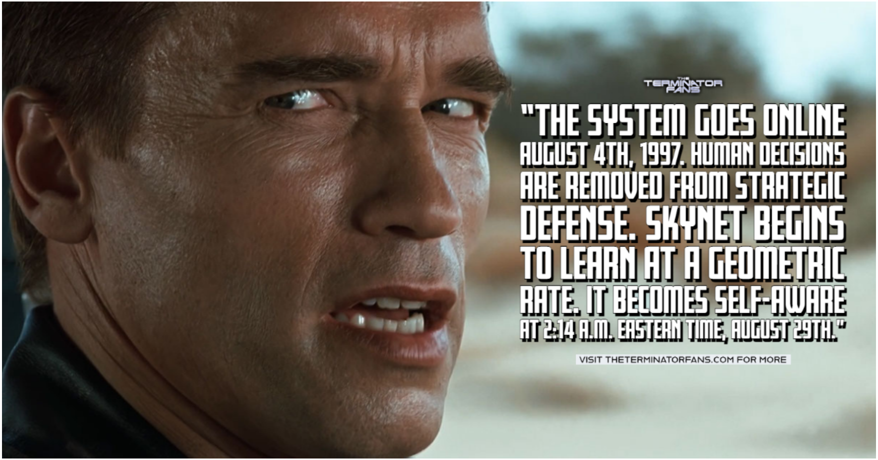
Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

September 19, 2023

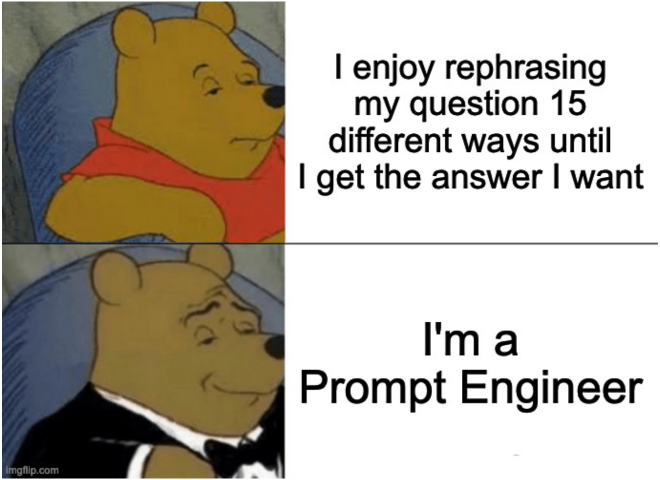
In the Beginning

In November of 2022, OpenAI introduced ChatGPT to the world...


THE TERMINATOR FANS
"THE SYSTEM GOES ONLINE AUGUST 4TH, 1997. HUMAN DECISIONS ARE REMOVED FROM STRATEGIC DEFENSE. SKYNET BEGINS TO LEARN AT A GEOMETRIC RATE. IT BECOMES SELF-AWARE AT 2:14 A.M. EASTERN TIME, AUGUST 29TH."
VISIT THETERMINATORFANS.COM FOR MORE

Reaction 2


STARTUP TECH SUPPOSED TO DISRUPT THEIR INDUSTRY OF CHOICE
CHATGPT
imgflip.com


I enjoy rephrasing my question 15 different ways until I get the answer I want
I'm a Prompt Engineer
imgflip.com


The "buy my course" starter pack
Is banned from PayPal
Doesn't want you to know that that his money really comes from selling his course
"I'm a mentor"
$2,000 Per Month SMMA
fulfillment by amazon
AliExpress
shopify
WooCommerce
facebook business
INSTAGRAM FOR BUSINESS
AT 16
"Not saturated"
"Are you tired of working a 9-5?"
"college is a waste of money"
$1,000 Per DAY PASSIVE INCOME
*not a rental
$5,000 ON SHOPIFY
WITH NO EXPERIENCE
"you gotta find a Niche"
Usally in his late teens
"Five easy steps"
Sets up an "us vs them" dynamic

Such large language models have existed before, but ChatGPT ended up a hit because it's pretty good at being "conversational".

This is referred to as Natural Language Processing (NLP).

Just because it gives you an answer, doesn't mean the answer is correct or true.

AI is subject to hallucinations.

Remember, The Law...

Legally and professionally, the engineer is responsible for understanding how a given tool works & verifying the output is reasonable & correct.

Prepared in Advance?

Part of what makes the GPT-3 and GPT-4 models better at producing output that matches our expectations is that it relies on pre-training.

This course is not one on neural networks, large language models, AI, or similar.

But performance is relevant here!

Parameters

One factor that matters in how good a model is: parameters.

Bigger is *usually* better...
  But requires more computational and memory resources.

We can maybe tune some options!

Let’s Try Some Optimization

This section is based on a guide from “Hugging Face”.

Hugging Face

You may have guessed by the placement of this topic in the course material that the GPU is the right choice for how to generate or train a large language model.

About that Name

Just... uh... don't confuse Hugging Face with Facehugger...

Why a GPU?

In this case we're talking about Transformers.

There are three main groups of optimizations that it does:

- Tensor Contractions
- Statistical Normalizations
- Element-Wise Operators

We also need to consider what's in memory.

Optimize What?

We can focus on how to generate a model that gives answers quickly...

Or we can focus on how to generate or train the model quickly.

Make a Fast Model

Use more space to reduce CPU usage, optimize for common cases, speculate...

Some of these are more fun than others: given a particular question, can you guess what the followup might be?

Make a Model Fast

Why would we customize some LLM?

Don't send your data to OpenAI...

Specialize for your workload.

Our first major optimization, and perhaps the easiest to do, is the batch size.

Let’s Review the Code

The code is a little too large for the slides so let’s go over it in a code editor.

The bert-large-uncased model is about 340 MB.

It’s uncased because it makes no distinction between capitals and lower-case letters, e.g., it sees “Word” and “word” as equivalent.

Let's Try Training

```
jzarnett@ecetesla0:~/github/ece459/lectures/live-coding/L24$ python3 dummy_data.py
Starting up. Initial GPU utilization:
GPU memory occupied: 0 MB.
Initialized Torch; current GPU utilization:
GPU memory occupied: 417 MB.
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at  bert-large-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
GPU memory occupied: 1705 MB.
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0;
7.43 GiB total capacity; 6.90 GiB already allocated; 16.81 MiB free; 6.90 GiB
reserved in total by PyTorch) If reserved memory is >> allocated memory try setting
max_split_size_mb to avoid fragmentation.  See documentation for Memory Management
and PYTORCH_CUDA_ALLOC_CONF
```

I asked `nvidia-smi`...

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.199.02   Driver Version: 470.199.02   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P4            Off  | 00000000:17:00.0 Off |                    0 |
| N/A   42C    P0    23W /  75W |      0MiB /  7611MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

7 611 MB is not enough for this model...

Problem is we don't have any bigger VRAM cards.

Fine, I Guess

What I actually did next was change to a smaller version of the model, `bert-base-uncased`.

It’s significantly smaller (110 MB) and something the card could handle.

Smaller Model Output

```
jzarnett@ecetesla0:~/github/ece459/lectures/live-coding/L24$ python3 dummy_data.py
Starting up. Initial GPU utilization:
GPU memory occupied: 0 MB.
Initialized Torch; current GPU utilization:
GPU memory occupied: 417 MB.
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at  bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use
 it for predictions and inference.
GPU memory occupied: 887 MB.
{'loss': 0.0028, 'learning_rate': 1.1718750000000001e-06, 'epoch': 0.98}
{'train_runtime': 109.6152, 'train_samples_per_second': 4.671,
'train_steps_per_second': 4.671, 'train_loss': 0.0027378778694355788, 'epoch': 1.0}
Time: 109.62
Samples/second: 4.67
GPU memory occupied: 3281 MB.
```

Results Table

Then I needed to experiment some more with batch size to find the ideal.

Batch Size	Time (s)	Samples/s	Memory Used (MB)	Utilization (%)
1	109.62	4.67	3 281	43.1
2	85.82	5.97	3 391	44.6
4	72.18	7.09	4 613	60.6
8	66.70	7.68	7 069	92.9

Unsurprisingly, choosing batch size of 9 leads to OOM.

Next Idea

We seem to be memory limited – let's see what we can do?

First idea: gradient accumulation.

Calculate gradients in small increments rather than for the whole batch.

Experimenting with Gradient Accumulation

Steps	Time (s)	Samples/s	Memory Used (MB)	Utilization (%)
1	66.06	7.75	7 069	92.9
2	63.96	8.01	7 509	98.7
4	62.81	8.15	7 509	98.7
8	62.65	8.17	7 509	98.7
16	62.42	8.20	7 509	98.7
32	62.44	8.20	7 509	98.7
128	62.20	8.23	6 637	87.2
1024	61.78	8.29	6 637	87.2
4096	62.16	8.24	6 637	87.2

I got Suspicious

I got suspicious about the 128 dropoff in memory usage and it made me think about other indicators – is it getting worse somehow?

The output talks about training loss...

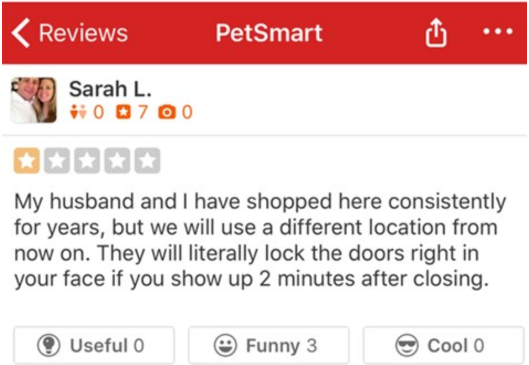
Gradient Accumulation Steps	**Loss**
1	0.029
2	0.070
4	0.163
8	0.169
16	0.447
32	0.445
128	0.435
1024	0.463
4096	0.014

Is that concerning?

Let’s Validate

We need to do some validation of how it’s going...

We’ll train and validate using some Yelp data.



More Code, More Review

The python code here is significantly different but we'll take a look.

I've skipped some intermediate results since at 9 minutes to calculate it takes a while to fill in all the values above.

Accuracy Results

GA Steps	Time (s)	Samples/s	Memory Used (MB)	Final Accuracy
1	538.37	5.56	7 069	0.621
8	501.89	5.98	7 509	0.554
32	429.70	6.98	7 509	0.347
1024	513.17	5.85	7 509	0.222

Interesting results – maybe a little concerning?

More Isn't Always Better

Increasing the gradient accumulation does change the effective batch size...

Batch size too large may mean less ability to generalize (overfitting).

But smaller isn't always better either; can underfit the data.

In the Yelp example, I get worse accuracy with batch size of 1 than 4, and 4 is worse than 8. There really is no magic number.

Gradient Checkpointing

Gradient Checkpointing: increase compute time to save memory.

Instead of saving activations, save only some, recompute the rest.

Does it work?

Gradient Checkpointing

Trying this out with batch size of 8, the total time goes from 66.70 to 93.07s and the memory from 7 069 down to 3 619 MB.

As expected, we got slower but used less memory.

Maybe it means we can increase the batch size?

Raising it to 16 means the time was 100.55s but still only 3 731 MB

Increasing the batch size a lot to finish faster might work eventually.

And no, using this checkpointing even with a batch size of `1` is not sufficient to run the `bert-large-uncased` model on `ecetesla0`.

And remember that excessively large batch sizes aren't good...

Mixed Precision, Data Preloading

Mixed Precision: Use less accurate types (16 vs 32-bit).

Data Preloading: Multithread to get data to GPU faster.

Other ideas: Mixture of Experts, bigger GPU, multiple-GPU...

Tradeoffs

- Accuracy for time?
- Memory for CPU?
- Err on the side of over- or under-fitting?

Tradeoffs

In the next few years, technologies and decision-making for this will become much more sophisticated.

But in the meantime, we can have a lot of fun experimenting and learning.