Napkin Math For Finetuning

Jonathan Whitaker @johnowhitaker

Questions:

- What affects performance when fine tuning?
- How can I make it faster?
- When should I use LoRA? Quantization? GC? CPU Offloading?
- What’s the cheapest option?
- What’s the most accurate?
- What hardware should I use (cheapest? fastest?)
- How will changing [X] affect the training time?
- What batch size / context length / … is best?
- How do I figure out what settings to use

Links

https://johnowhitaker.dev << Me (@johnowhitaker most places)

https://github.com/AnswerDotAI/fsdp_qlora/blob/main/benchmarks_03_2024.md << My (and others) benchmarks for FSDP+QLoRA used in example

https://github.com/huggingface/transformers/issues/25572#issuecomment-1687749561 << Someone showing math for activations etc

https://docs.google.com/presentation/d/1Ye_6zeatCWkq-fx8A--yK34uwU8oC2YQtMSTV1DgkSI/edit?usp=sharing << These slides (??)

https://pytorch.org/tutorials/intermediate/optimizer_step_in_backward_tutorial.html << more use of memory viz plus an under-rated technique

The Good News

- We know how these models work
- We can do the maths
- We can do experiments

The Bad News

- Implementation details differ
- The maths can get hard
- “I don’t know what I’m doing!!!” << us, soon ;)

It's going to be OK!

The Plan

- Intro (done ✅)
- What happens during training/finetuning?
- Value of running experiments
- A small code example: instrumenting a ‘training step’
- Napkin math live-mathing to estimate memory use
- A case study
- Questions << ask throughout too!

Training Neural Networks

Through a model...

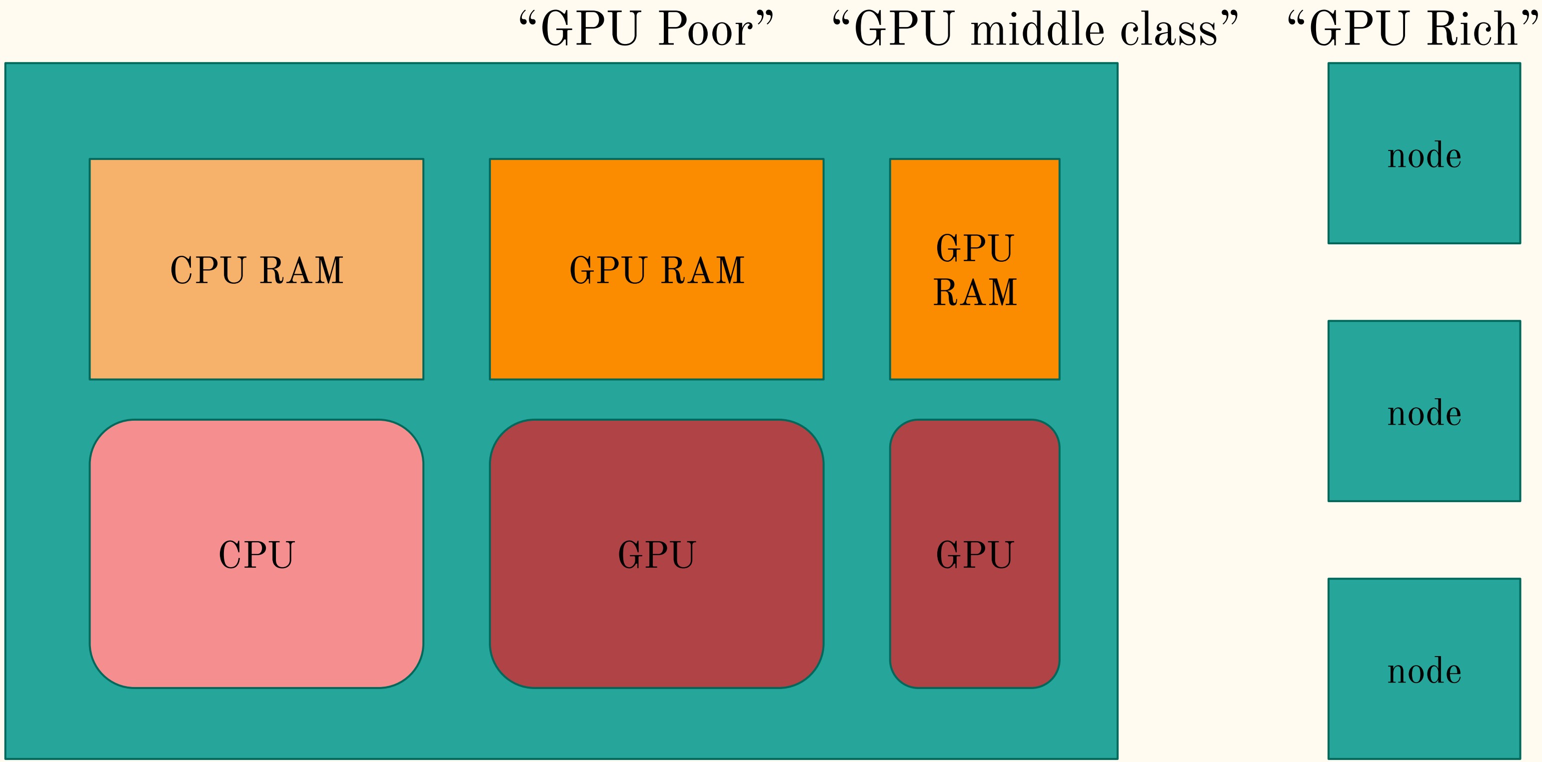
We feed some data...

To get an answer

Then update the model...

We measure how good it is.

On Computers



Training Neural Networks

(data takes up space)

We feed some data…

(This involves crunching lots of numbers)

(the model takes up space)

Through a model…

(storing gradients - more space)

(The intermediate results take up space)

To get an answer

(we compare to the right answer)

We measure how good it is.

(optimizer state - more space)

Then update the model…

(we trace back the gradients to find out how)

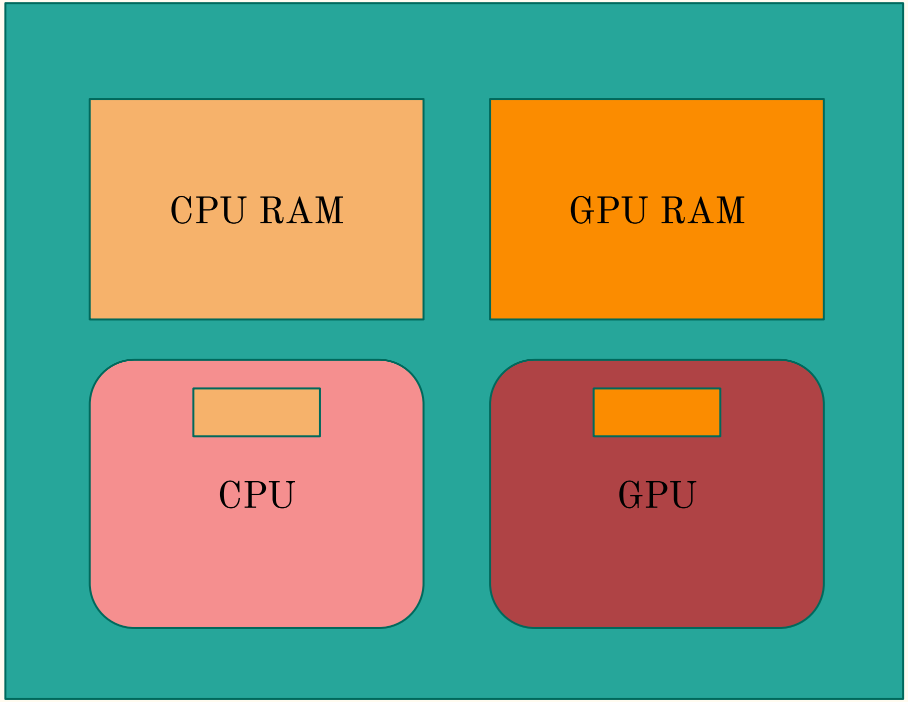
(This involves crunching lots of numbers)

What takes up time?

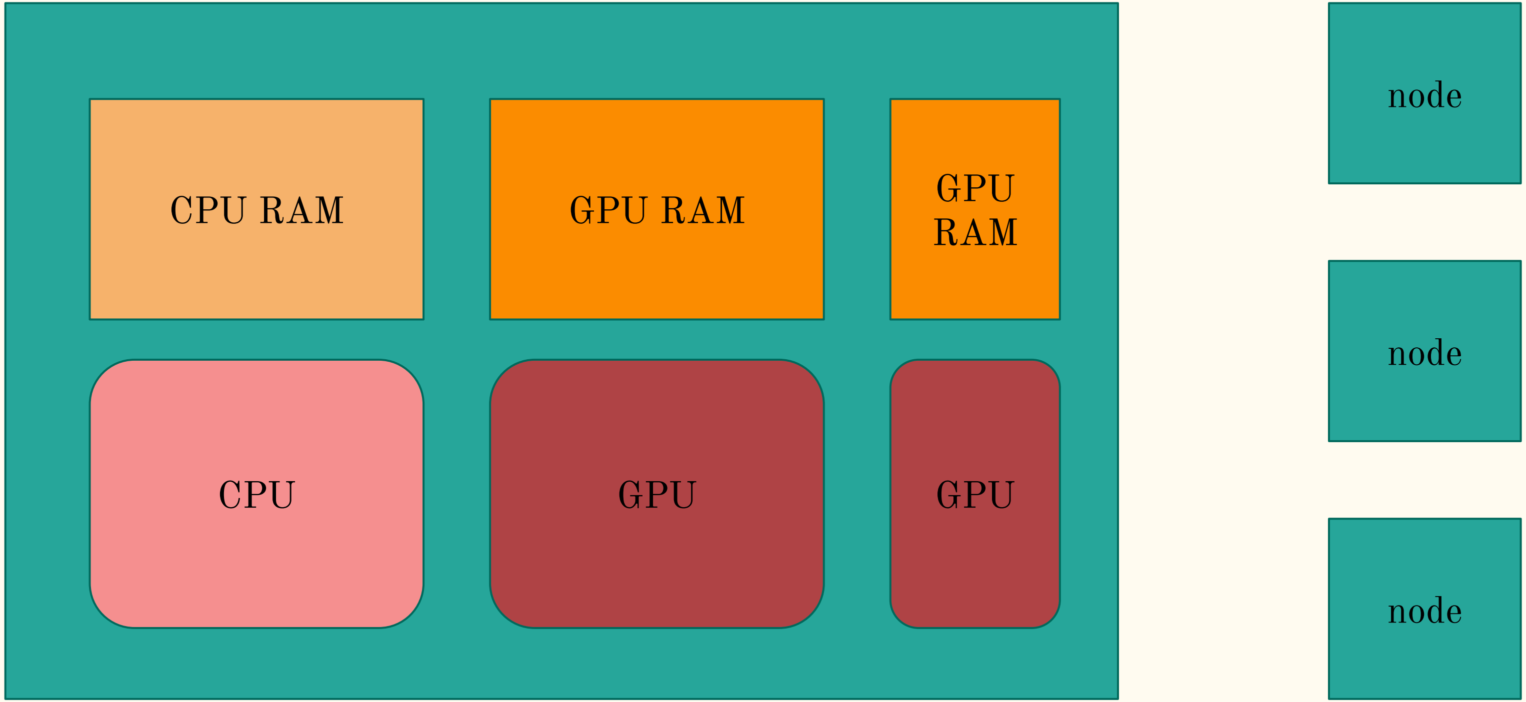
- Crunching the numbers
- Copying data around

Keep an eye on these two aspects

Why are we copying data around?



Why are we moving data around?



Goal: keep the GPU fed

Tricks at our disposal

- Flash attention and fancy kernels
  - Reducing intermediates and VRAM<->HBM with ‘fused kernels’, changing complexity
- Gradient Checkpointing AKA Activation Checkpointing
  - Reducing memory usage in exchange for a little more compute
- CPU Offloading
  - Storing some things on the CPU to free up GPU RAM for more data
- LoRA
  - Only training some parameters -> less gradients, small optimizer state
- Quantization
  - Reduces the space needed for weights (etc) but needs a little compute to dequantize

Running Experiments

Change one thing at a time, see what it does (thank you Modal for making it easy to test on H100s!)

```
gradient_accumulation_steps: 1
micro_batch_size: 32
num_epochs: 1
optimizer: adamw_torch
lr_scheduler: cosine
learning_rate: 0.0001
```

Will 2x batch size give 2x speedup?

Name (3 visualized)	Runtime
bs=64, load_in_4_bit	1m 35s
bs=64	1m 29s
bs=32	1m 36s

A simple example

Training a model on one GPU

- Forward Pass
- Backward Pass
- Effect of batch size
- Effect of context length
- Memory: constant vs scaling with data?
- Compute: how does it scale?

A Case Study, ‘Compute Bound’ vs ‘Memory Bound’

Questions?