Profiling and Optimising ML Workfloads

Topics

- Setup
- GPU Utilization and Kernel Efficiency
- CPU-GPU Balance
- I/O and Storage Bottlenecks
- Hyperparameter Choices Affecting Performance

Setup

A Toy Example

- Train a ViT on Flowers102 dataset

- Start by cloning the example repo into your project subdirectory
- ssh login

```
cd /bask/projects/v/vjgo8416-hpc2511/$USER
```

```
cd hpc-training-nov-2025/4-Profiling
```

`git pull` (to allow for last min changes!)

- Future commands can be copy and pasted from README (locally or https://github.com/lannelin/pytorch profiling/tree/25_11)

water lily (72)

desert-rose (84)

gazania (70)

wild pansy (51)

oxeye daisy (48)

columbine (83)

sword lily (42)

orange dahlia (58)

barbeton daisy (40)

Setting up an environment

- Launch an interactive job

```
srun --export=USER,HOME,PATH,TERM --account vjgo8416-hpc2511 --qos=turing --nodes=1-1 --cpus-per-gpu=36 --reservation
vjgo8416-hpc2511 --gres=gpu:1 --time=1:0:0 --pty /bin/bash
```

- Load the necessary modules

```
module load Python/3.11.3-GCCcore-12.3.0
```

- Use the virtual environment I've already made and installed a copy of the repo into

```
source $PROJECT_DIR/profiling_shared/env/bin/activate

export PROJECT=vjgo8416-hpc2511
export PROJECT_DIR=/bask/projects/v/vjgo8416-hpc2511

cd $PROJECT_DIR/$USER/hpc-training-nov-2025/4-Profiling
```

- Using shared data (inspect config)
- Set environment variable to allow us to use shared huggingface models (rather than redownload)

```
export TORCH_HOME=/bask/projects/v/vjgo8416-training25/profiling_shared/torch_cache
```

Training a model

- You monitored the GPU in previous session with `nvidia-smi`. Let’s do that again from a separate window/terminal
  `ssh <USER>@login.baskerville.ac.uk` or use tensorboard and skip the ssh/squeue/srun bit

  ```
  squeue # get JOB ID

  srun --pty --overlap --jobid YOUR_JOBID bash

  nvidia-smi -l 1
  ```

- Back in tab1, train a model for just a single epoch

  ```
  python runner.py fit -c configs/training.yaml --trainer.logger.name test --trainer.max_epochs 2
  ```

  (this should take a minute or two)

What just happened?

Go to training config at https://github.com/lannelin/pytorch-profiling/blob/25_11/configs/training.yaml ⚡

Review GPU use from nvidia-smi

Note that we wrote a model checkpoint out into the logs dir
(check out disk space with `du -sch ./logs`)

Tensorboard…

What’s happening during training?

Roughly:

1. A **batch** of data is “passed forward” through the model (a series of operations involving the model **weights**).
2. The result for each item in the batch is compared to its expected value (the labels), this gives us the **loss**.
3. **Activations** are stored in the forward pass and used in a backward pass to compute **gradient** of the **loss** function with respect to the model **weights**.
4. We use the **gradient** and the **learning rate** to update the **weights**

What just happened… in Tensorboard?

- (the Baskerville application doesn’t have the plugin we’ll be using, unfortunately)

- Assign port numbers to the room*.
- Start tensorboard:

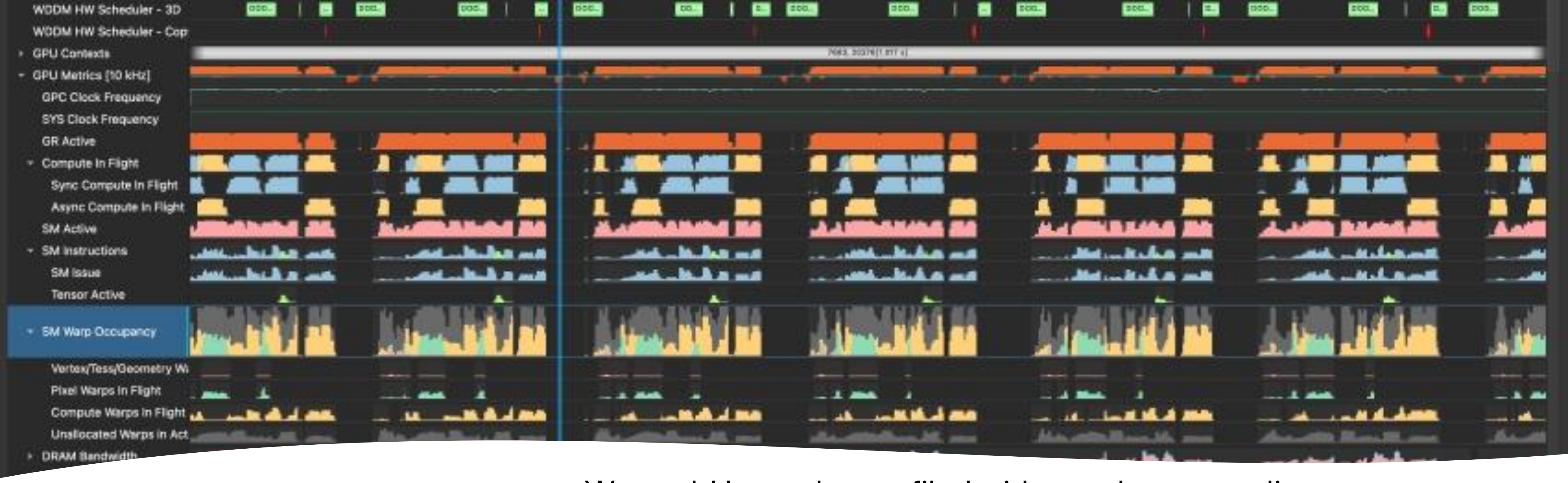
```
port=<my_port>
tensorboard --port ${port} --host $(hostname).cluster.baskerville.ac.uk --logdir $(pwd)/logs
```

- In your browser, go to tensorboard at: https://portal.baskerville.ac.uk/rnode/bask-<hostname>.cluster.baskerville.ac.uk/<my_port>/

Can try training a model on cpu for comparison (note aten::copy_)

```
python runner.py fit -c configs/training.yaml --trainer.logger.name cpu --trainer.max_epochs 1 --trainer.accelerator cpu
```

* We’ll each need to use a different port and we’ll assign this manually in the session.



Nsight Systems/Compute

https://developer.nvidia.com/nsight-systems

- We could have also profiled with nsys by prepending our command with the nsys command below.

```
nsys profile -w true -t cuda,nvtx,osrt,cudnn,cublas -s cpu --capture-range=cudaProfilerApi --capture-range-end=stop  --cudabacktrace=true -x true -o my_profile <python command> --trainer.profiler.emit_nvtx true
```

- Incompatible GCC for this session
- ~~Demo of Nsys~~

GPU Utilization and Kernel Efficiency

What’s using my GPU memory?

Model weights

+ **data** in batch

+ **activations** (num activations is some function of data size and num weights)

+ **gradients** (dependent on num weights)

+ **optimizer states**

+ any other temporary variables

+ any prefetching

NB: if we’re not doing the backwards pass (i.e. we’re doing inference) we don’t need the activations, gradients, and optimizer states!

Mixed Precision

- Manual – see snippet (and check ckpt exists first)

- Easy with lightning Trainer:

```
diff configs/training_bf16.yaml configs/training.yaml
```

With `nvidia-smi` running:

```
python runner.py fit -c configs/training_bf16.yaml --trainer.logger.name bf16
--trainer.max_epochs 2
```

Torch: inference_mode

- model.eval() vs inference_mode ?
- model.eval() for things like turning off dropout and batch norm
- inference_mode analogous to no_grad, locally disable gradients

- With nvidia-smi running, run model inference snippets with and without inference_mode

GPU Utilization and Kernel Efficiency - Summary

- Monitor GPU utilization (e.g., nvidia-smi, Nsight Systems).
- High GPU utilization doesn’t always mean efficient execution—kernel launch overhead and small batch sizes can waste cycles.
- Mixed precision training (FP16) can improve throughput and reduce memory footprint.
- Different requirements at training and inference

CPU-GPU Balance

How does the data get loaded?

In PyTorch we use `Dataset`s and `DataLoader`s.
(https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

- A Dataset stores the samples and labels
- A DataLoader wraps a Dataset and allows us to iterate over the dataset and makes batching easy

- In short, these two classes allow us to get data items and collate them into batches

DataLoader::num_workers

What does the num_workers parameter do?

We’ll illustrate its behaviour by adding a sleep to our data transformations. This sleep will run for every *item* in every batch.

We’ve defined this behaviour in `configs/sleepy_data.yaml` based on code in `src/pytorch_profiling/utils/transforms.py`.

With nvidia-smi running:

```
python runner.py fit -c configs/training.yaml --trainer.logger.name 0workers --trainer.max_epochs 2  --data.num_workers 0
```

```
python runner.py fit -c configs/training.yaml --trainer.logger.name 1workers --trainer.max_epochs 2  --data.num_workers 1
```

```
python runner.py fit -c configs/training.yaml --trainer.logger.name 2workers --trainer.max_epochs 2  --data.num_workers 2
```

```
python runner.py fit -c configs/training.yaml --trainer.logger.name 16workers --trainer.max_epochs 2  --data.num_workers 16
```

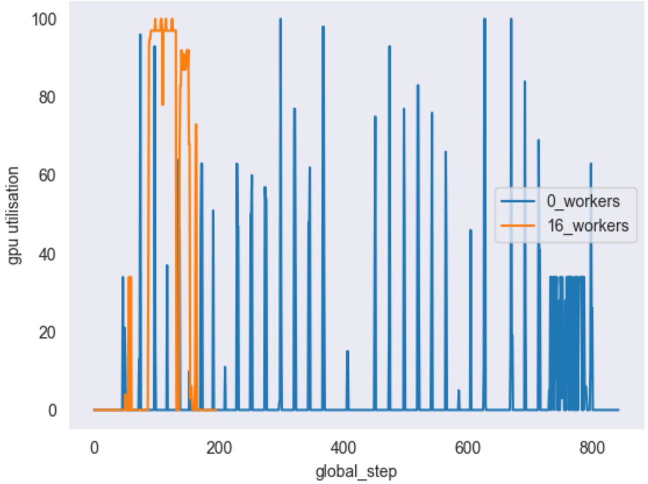
DataLoader::num_workers (2)

Using num_workers=0 (i.e. the data loading is handled by the main process), we can see that training runs slowly.

Using num_workers=1 (i.e. we have a single dedicated worker for data loading), we can still see that training runs slowly.

Using num_workers=2 (i.e. we have two dedicated workers for data loading), we can see that training has sped up!

Using num_workers=16 (i.e. we have 16 dedicated workers for data loading), we can see that training ha dramatically sped up.



DataLoader::num_workers (3)

Why did that make a difference?

Because our bottleneck was in our data transformation process and increasing num_workers allows us to load data, and perform this transformation, in **parallel**

Increasing num_workers can speed up our training when the model is consuming data (forward and backward pass) faster than the DataLoader is providing new batches.

Typically, the GPU is our expensive resource, we want to use it effectively and keep it fed.

Should we always just set this to maximum recommended i.e. nproc-1?

Might be wasteful - are we using CPU for anything else?

What if our data loading was limited by IO?

This might not have helped!

CPU-GPU Balance - Summary

- Importance of CPU-side preprocessing and how slow augmentation or feature extraction can starve the GPU.
- Use profiling tools to show CPU bottlenecks (e.g., PyTorch Profiler, Nsight Systems).
- Demonstrated how increasing num_workers in DataLoader can help.

IO and Storage Bottlenecks

Network vs local disk

- What comparative performance of real-terms read/write speeds (inc. network overhead) do we expect from:

1. /bask
2. /scratch-global
3. /tmp (local scratch)

Why is it helpful to know this?

We want to avoid bottlenecks and slow read/write from disk can be one of these. We should use the appropriate disk for the task.

Where in a deep learning pipeline might we encounter a read/write bottleneck?

I/O and Storage Bottlenecks - Summary

- Reading large datasets from network storage can throttle performance.
- Consider caching strategies or using local SSDs.
- Use of async data loading and prefetching.

Hyperparameter Choices Affecting Performance

Batch size vs learning rate Warning: handwavy. Yes, even more than before

What batch size should I use?

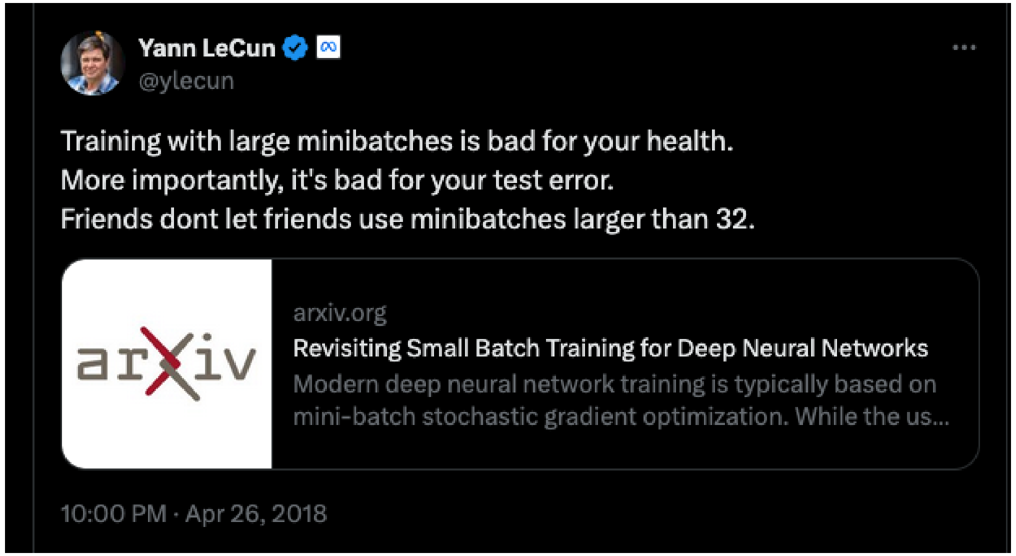
Often the answer is “as big as you can”. This often means the highest you can set before you run out of GPU memory.

This is debatable.

Bigger provides a better estimate of the true gradient but smaller has been shown to have nice regularising effects.

If I increase the batch size, do I need to do anything else?

Consider your learning rate.
Better estimate -> bigger step?



Yann LeCun
@ylecun

Training with large minibatches is bad for your health.
More importantly, it's bad for your test error.
Friends dont let friends use minibatches larger than 32.

arxiv.org
Revisiting Small Batch Training for Deep Neural Networks
Modern deep neural network training is typically based on mini-batch stochastic gradient optimization. While the us...

10:00 PM · Apr 26, 2018

Note: this is from **2018 and was demonstrated on a relatively small image dataset.**
In 2021 the ViT paper (model used today) used a batch size of 4096.
We’re now seeing batch sizes of up to 4M reported in the Llama 3 paper.

Other tips and tricks

There are a bunch of optimisations/shortcuts/tricks you can use to speed up training, reduce GPU memory usage, etc. Many are included in training recipes by default. Some examples (at wildly different scales):

- Quantization
- Gradient accumulation (simulates a larger batch size)
- LoRA (or other PEFT methods)
- Any others to discuss?

Hyperparameter Choices Affecting Performance - Summary

- Batch size, sequence length, and model depth impact memory and compute.
- Trade-offs between throughput and convergence speed.

End