report

February 11, 2025

1 Report

Jakub Józefowicz

JJ395253

1.1 Implementation overview

1.1.1 Context

- Everything was done on the eden cluster on dgx-a100 nodes https://hpc.mini.pw.edu.pl/description/
  - This means I didn't have access to Krzysztof Ciebera's c4 dataset, so I just streamed if off huggingface. This can occasionally give a 504, I handed it in using the pl-grid datasets but that doesn't work set `is_plgrid` to false in `scripts/`generate_scripts.ipynb' and run the notebook to regenerate the scripts, then copy them over into project root. Same for account values
  - The interconnect between nodes is infiniband, but implements IBoIP, if that isn't the cse on PL-grid then the cpu backend will have to be changed from gloo to mpi.
  - The interface used might also have to be specified for dist. comms on some clusters
- I didn't bother optimisng NCCL env. variables, but I included them commented out in the scripts
- `OMP_NUM_WORKERS` could also be increased maybe
- Testing locally the script does seem to use all cores available

1.1.2 ArgParse

- Argument Parser was done in the ordinary fashion; I added some bounds checking for negative learning rates etc.

1.1.3 Monitoring

- Monitoring was added as Neptune runs with everything in the base namespace including the model params
- If you would like the scripts to include monitoring during grading then an api key has to be provided, the scripts can be regenerated with `generate_scripts.ipynb`

1.1.4 Torchrun

- one task per node, srun is using in combination with torchrun

- The master address was set according to a comment someone made in this gist: https://gist.github.com/TengdaHan/1dd10d335c7ca6f13810fff41e809904
  - hostname seems to work, but if not then one could srun one node to get it's ip
- The master port is randomly generated and is ≥ 10000
- The c10d backend is used, with `--standalone` used for the basic torchrun

1.1.5 FSDP

- Used `transformer_auto_wrap_policy` on the `Block` module provided
- Mixed precision is set to `bf16` in the manner requested if it is available on the node, this info is logged

1.1.6 2 GPUs

- No substantial changes required for the training scripts. Just change num nodes in batch and torchrun adjusting the number of gpus per node
- The dataloading was done with `split_dataset_by_node` provided by huggingface; `DistributedSampler` doesn't work with `IterableDataset`

1.1.7 Cosine Learning

- Just used a `SequentialLR` for `(LinearLR, CosineAnnealingLR)`
  - `T_max` was set to `0.99 * train_steps`
  - No warm restarts were used, since they didn't seem to be used in the optimal compute paper

1.1.8 Optimal Training steps

- Calculated below
- I just used `D=20N`, I don't see anything more in the paper that has to be extrapolated
- This was the only part I ran on two nodes
- Used `#SBATCH --array=2-3`

1.1.9 Saving and loading the model

- Ran it with dependecies

```
jjozefowicz@eden:~/git/bml-big-2$ sbatch sbatch_save.sub
Submitted batch job 977273
jjozefowicz@eden:~/git/bml-big-2$ sbatch -d afterok:977273 sbatch_load.sub
Submitted batch job 977274
```

- I used `from torch.distributed.checkpoint.state_dict import get_state_dict, set_state_dict` instead of the FSDP equivalent since the latter is deprecated and it was suggested to use the former instead.
- I save:
  - Model params
  - Optimizer state
  - Scheduler state
  - No. of steps ran

- It seems like for huggingface datasets you have to just skip the first n samples instead of setting some dataloader state

1.2 Calculating number of train steps

We just calculate the number of parameters manually and then multiply by 20 from what I understand. I don't see what else we need from the publication if we are given $D = 20N$.

[ ]:
```
vocab_size = 50257
d_model = embed_dim = 256
sequence_length = max_length = 256
num_layers = 4
num_heads = 4
# embedding
embeds = (vocab_size * embed_dim) + (sequence_length * embed_dim)
# transformer blocks
## fully connected
layer_norm = 2 * d_model
ff_in = d_model * (4 * d_model) + (4 * d_model)  # bias
ff_out = (4 * d_model) * d_model + d_model  # bias
ff = layer_norm + ff_in + ff_out
## self-attention
attn_in_proj = d_model * (3 * d_model)  # no bias
attn_out_proj = d_model * d_model  # no bias
attn = (
    attn_in_proj + attn_out_proj
)  # it seems like d_model of a head is d_model/num_heads
blocks = num_layers * (ff + attn)
# final head
head = d_model * vocab_size
# In total we get
N = embeds + blocks + head
D = 20 * N
print(f"N: {N}, D:{D}")
```

```
N: 28950016, D:579000320
```

So now for number of steps

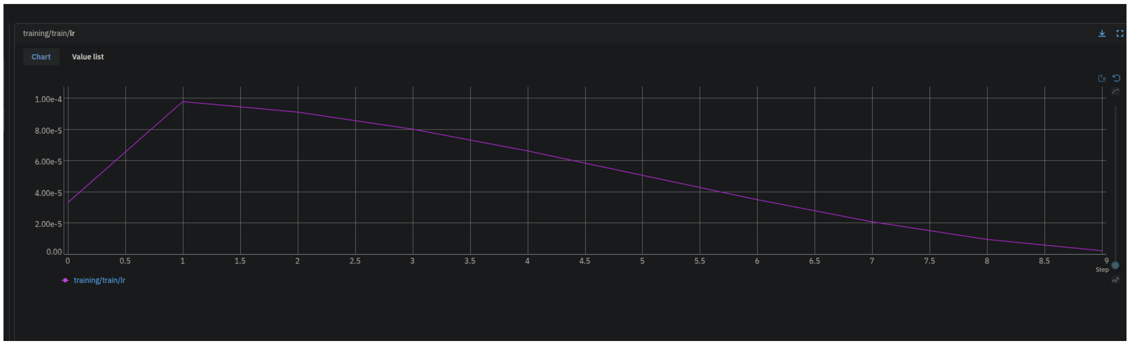
[7]:
```
batch_size = 256
per_step = batch_size * sequence_length

num_steps = round(D / per_step)
print(f"Num steps: {num_steps}")
```

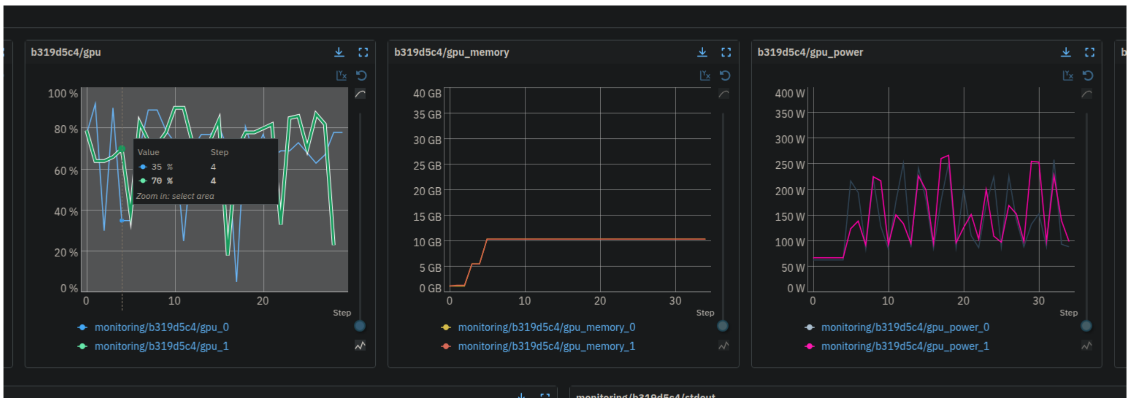
```
Num steps: 8835
```

1.3 Results

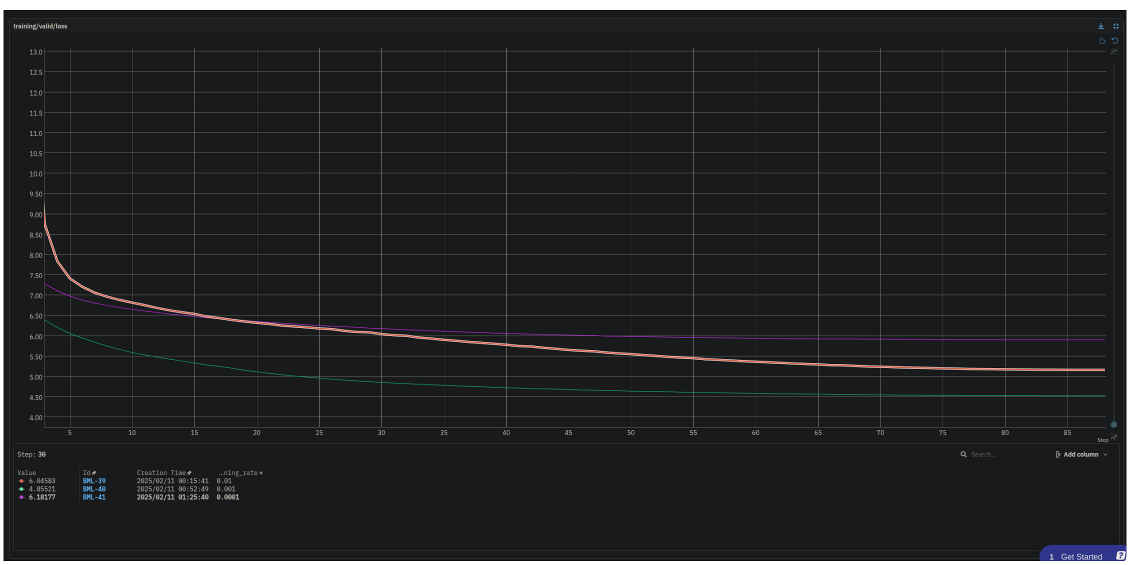
1.3.1 Cosine scheduler



1.3.2 Multi GPU



1.3.3 Learning Rate Comparison



1.3.4 Checkpoint Save-Load