Week 13 Lab: Profiling & Optimization

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Objectives

1. **Profiling**: Identify performance bottlenecks using PyTorch Profiler.
2. **Mixed Precision**: Implement AMP to speed up training.
3. **Data Loading**: Optimize DataLoader performance.
4. **Pruning**: Apply structured and unstructured pruning.

Prerequisites:

- GPU Runtime (Colab/Kaggle recommended).
- `pip install torch torchvision tensorboard`

Exercise 1: The Baseline (30 min)

Setup:

Create `baseline.py`. Train a ResNet-18 on CIFAR-10.

Use standard settings: `batch_size=32`, `num_workers=0`, `Float32`.

Task:

1. Measure **Throughput**: Images/second.
2. Measure **Peak Memory**: `torch.cuda.max_memory_allocated()`.
3. Record these baselines.

Exercise 2: Profiling (45 min)

Task:

Wrap your training loop in the PyTorch Profiler.

```
from torch.profiler import profile, record_function, ProfilerActivity

with profile(activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA], record_shapes=True) as prof:
    for i, batch in enumerate(dataloader):
        if i >= 10: break
        train_step(batch)
        prof.step()

print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

Analyze:

- Is most time spent on CPU (DataLoader) or GPU (Compute)?
- Are there gaps in the GPU timeline?

Exercise 3: Optimizing Data Loading (30 min)

Task:

Modify `DataLoader`:

1. Increase `num_workers` (try 2, 4, 8).
2. Enable `pin_memory=True`.
3. Enable `persistent_workers=True`.

Measure:

- Did the throughput increase?
- Did the "CPU time" in profiler decrease?

Exercise 4: Automatic Mixed Precision (AMP) (45 min)

Task:

Integrate `torch.cuda.amp` into your training loop.

1. Initialize `GradScaler`.
2. Wrap forward pass in `with autocast():`.
3. Use `scaler.scale(loss).backward()` and `scaler.step()`.

Measure:

- New Throughput (Images/sec).
- New Memory usage.
- **Expected Result**: ~2x speedup, ~50% memory reduction.

Exercise 5: Pruning (30 min)

Task:

Prune the trained model.

```
import torch.nn.utils.prune as prune

# Prune 30% of connections in all Conv2d layers
for name, module in model.named_modules():
    if isinstance(module, torch.nn.Conv2d):
        prune.l1_unstructured(module, name='weight', amount=0.3)
```

Verify:

- Check sparsity: `print(torch.sum(module.weight == 0))`
- Measure inference speed (CPU vs GPU). Note: Unstructured pruning might not speed up standard GPU inference!

Submission

Deliverables:

1. `optimization_report.md` :
    - Table comparing Baseline vs Optimized (Throughput, Memory).
    - Screenshots of Profiler trace.
2. `optimized_train.py` : The final script with AMP + Workers.
3. `pruning_demo.py` : Script showing sparsity calculation.

Challenge: Try `torch.compile(model)` (PyTorch 2.0) and report the speedup!