OML Project

- Rupanshu — 2019475, Divyansh — 2019464

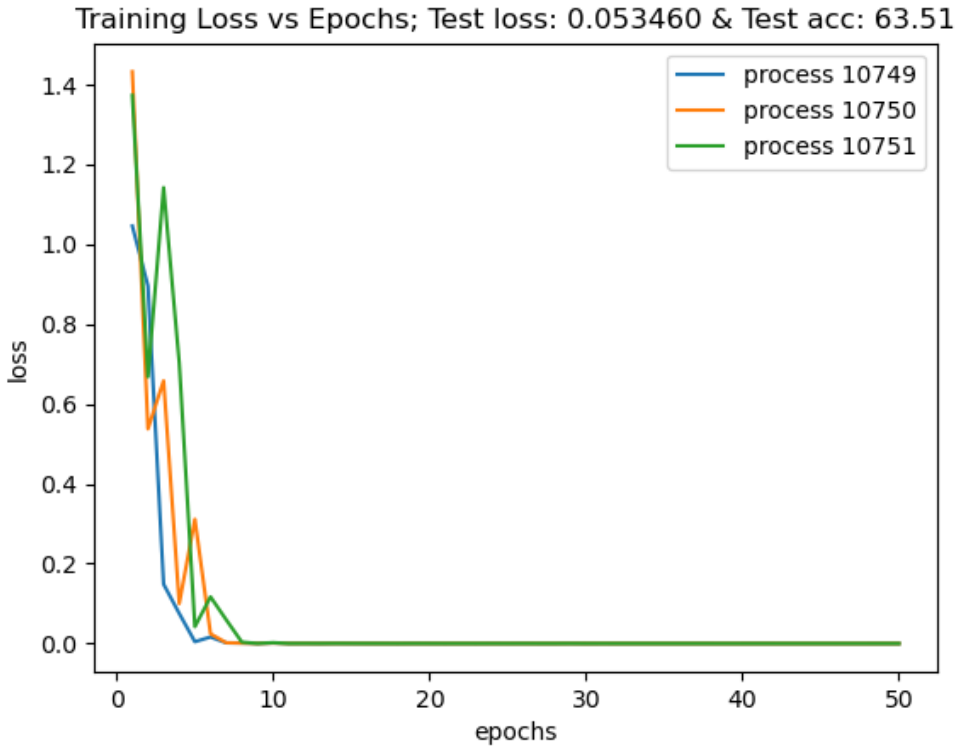
Hogwild

- Hogwild algorithm:

Algorithm 1 HOGWILD! update for individual processors

loop
  Sample e uniformly at random from E
  Read current state x_e and evaluate $G_e(x)$
  for $v \in e$ **do** $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$
end loop

- Hogwild is implemented in a shared memory setting on the CIFAR dataset in a distributed shared memory setting.
  - To achieve this, pytorch provides libraries such as `torch.multiprocessing` , several processes are spawned and optimize the shared model parameters. No synchronization is used.
  - SGD nesterov is used as an optimizer with the step function implemented without using pytorch.
  - The dataset was subsampled and divided in batch sizes of 64. The model was trained for 50 epochs.
  - Loss Criterion: Negative Likelihood loss `nn.NLLoss()`
- Loss plot for each process:
  - Num Processes: 3, Learning Rate: 0.01, Epochs: 50, Batch Size: 64, Momentum: 0.9



- Running Instruction:

```
python3 main.py

# usage: main.py [-h] [--batch-size BATCH_SIZE] [--num-epochs NUM_EPOCHS]
# [--momentum MOMENTUM] [--learning-rate LEARNING_RATE]
# [--display-interval DISPLAY_INTERVAL] [--num-processes NUM_PROCESSES
```

Local/K-step SGD

- Local SGD algorithm:

initialize $\widetilde{\mathbf{w}}_1$;

for $n = 1, ..., N$ **do**

Processor P_j, $j = 1, \ldots, P$ do concurrently:

set $\mathbf{w}_n^j = \widetilde{\mathbf{w}}_n$;

for $k = 1, ..., K$ **do**

randomly sample a mini-batch of size B_n and update:

$$\mathbf{w}_{n+k}^j = \mathbf{w}_{n+k-1}^j - \frac{\gamma_n}{B_n} \sum_{s=1}^{B_n} \nabla F(\mathbf{w}_{n+k-1}^j; \xi_{k,s}^j)$$

end

Synchronize $\widetilde{\mathbf{w}}_{n+1} = \frac{1}{P} \sum_{j=1}^{P} \mathbf{w}_{n+K}^j$;

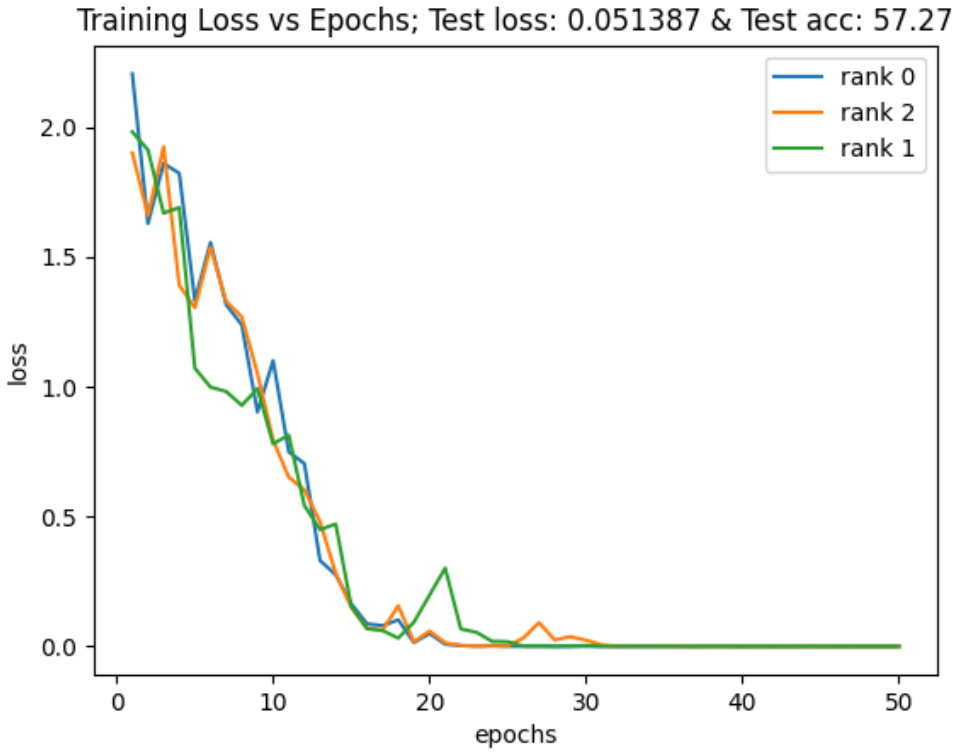
end

Algorithm 2: K-step average stochastic gradient descent algorithm

- Local SGD is implemented in a distributed settings where processes can be nodes in a networks.
  - To achieve this, pytorch provides libraries such as `torch.distributed`
    - Number of nodes is the `world_size`
    - Every process is identified by their `rank`
    - Data is partitioned between the processes without replacement
    - Each processes takes K-steps and then synchronizes and averages its weights with all the other nodes.
  - SGD nesterov is used as an optimizer with the step function implemented without using pytorch.

- The dataset was subsampled and divided in batch sizes of 64. The model was trained for 50 epochs.
- Loss Criterion: Negative Likelihood loss `nn.NLLoss()`

- Loss Plot for each processes:
  - Num Nodes: 3, Learning Rate: 0.01, Epochs: 50, Batch Size: 64, Momentum: 0.9, K: 3



- Running Instructions:

```
bash launch.sh [NUM_PROCESSES]
```