Contents

1 Problem: 1

1.1 Ring Reduce and Ring All Reduce

- These two parts have been implemented in the Python files- ring reduce and ring all reduce.

1.2 Describe how did I perform above two exercises. Compare the accuracy of the trained model and the training time.

- **Explanation of Part-a**

  (i) Data Preprocessing: The MNIST dataset was used(as mentioned in the question), which consists of images of handwritten digits. For compatibility with the LeNet-5 architecture, images were padded. The dataset was converted into tensor format for normalization & PyTorch compatibility and was divided into training(100 images which are split into four equal subsets) and testing.

  (ii) Model Architecture: The LeNet-5 model was implemented as a convolutional neural network (CNN) with the following layers:

  * First convolutional layer - 6 filters of size 5×5, which uses a Tanh activation function.
  * An average pooling layer - A kernel size of 2×2 and a stride of 2 for feature maps.
  * Second convolutional layer - 16 filters of size 5×5, which again uses a Tanh activation function.
  * Another average pooling layer - Same as before.
  * Third convolutional layer - 120 filters of size 5×5, which again uses a Tanh activation function.
  * The output of the final convolutional layer - Flattened and passed through a fully connected (FC) layer with 84 neurons, using another Tanh activation function.
  * A second fully connected layer - 84 neurons to 10 output classes.

  CrossEntropy loss function was used to train the model and Stochastic Gradient Descent (SGD) is optimizing it with a learning rate of 0.01.

  (iii) Data Parallelism Implementation(all reduce): The training process is using PyTorch's Distributed Data Parallel (DDP) framework for parallel training. The steps are as follows:

  * Spliting jobs into equal four processes, each assigned to one of the four subsets.
  * Each process initialized a copy of model and loading its subset using PyTorch.
  * To enable distributed training in PyTorch, the environment variables `MASTER_ADDR` and `MASTER_PORT` are set to define the master node's address and communication port. These ensure proper coordination between multiple processes during training.
  * During training, each process was computing gradients independently which were synchronized across processes using `torch.distributed.all_reduce`.
  * Gradient synchronization is done after each process updates it's local parameters.
  * The distributed training follows a data parallelism approach which is managed using `torch.distributed` package of Pytorch (using 'gloo' backend for inter-process communication).

  (iv) Performance Metrics Collection: The following metrics were recorded at each epoch while training:

  * Average training loss for each process.
  * Training accuracy.
  * The total training time per process.

- **Explanation of Part-b**

  (i) Data Parallelism Implementation(ring all reduce):

  * The previous implementation used **all reduce** approach for gradient synchronization while this uses **ring all reduce**, where multiple processes work on different data subsets while maintaining synchronization using PyTorch. Steps for ring all reduce are as follows:
    1. Initialize the distributed process group.

2. Divide the gradients into chunks.
3. Perform the scatter-reduce operation using non-blocking send and receive operations.
4. Perform the all-gather operation to distribute the reduced gradients.
5. Average the gradients and update the model.

(ii) Changes in Data Loading Strategy:

* Assigning subsets to processes using PyTorch's DistributedSampler to automatically partition the dataset among different processes.
* This ensures equal workload distribution and prevents redundant data processing.

Rest of the settings are same as all reduce implementation.

- **Comparison of Part a and b**

(i) Accuracy Comparison

* In Part (a), which is **all reduce**, the training accuracy starts at around 9% and gradually increases, reaching approximately 66% by the 100th epoch.
* Whereas Part (b), which is **ring-all reduce**, has a much faster improvement in accuracy, and end up with final average accuracy of 91%.
* The accuracies' trend of both the implementations show that **ring-all reduce** model is learning more effectively in early training stages than **all reduce**

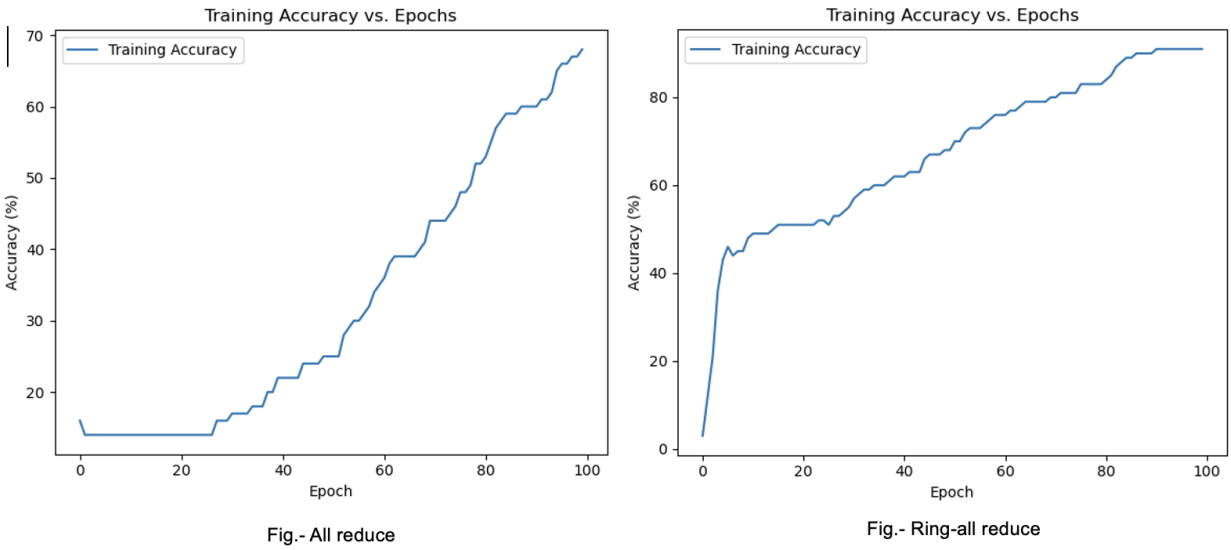

Fig.- All reduce

Fig.- Ring-all reduce

Figure 1: Accuracy vs Epochs

(ii) Training Loss and Convergence

* Both the models are showing a decreasing trend in the loss but **ring-all reduce** demonstrates a considerably quicker reduction in loss compared **all reduce**.
* In **all reduce**, the initial loss is approximately 2.3 and decreases to around 1.4 by the final epoch. This shows that the decline in loss is steady but not very sharp.
* In **ring-all reduce**, the initial loss is slightly lower which is around 2.25 but it decreases more aggressively, reaching close to 0.6 by the 100th epoch.
* The faster convergence of Part (b) indicates that it learns more efficiently and potentially due to improved architecture and better hyperparameters.

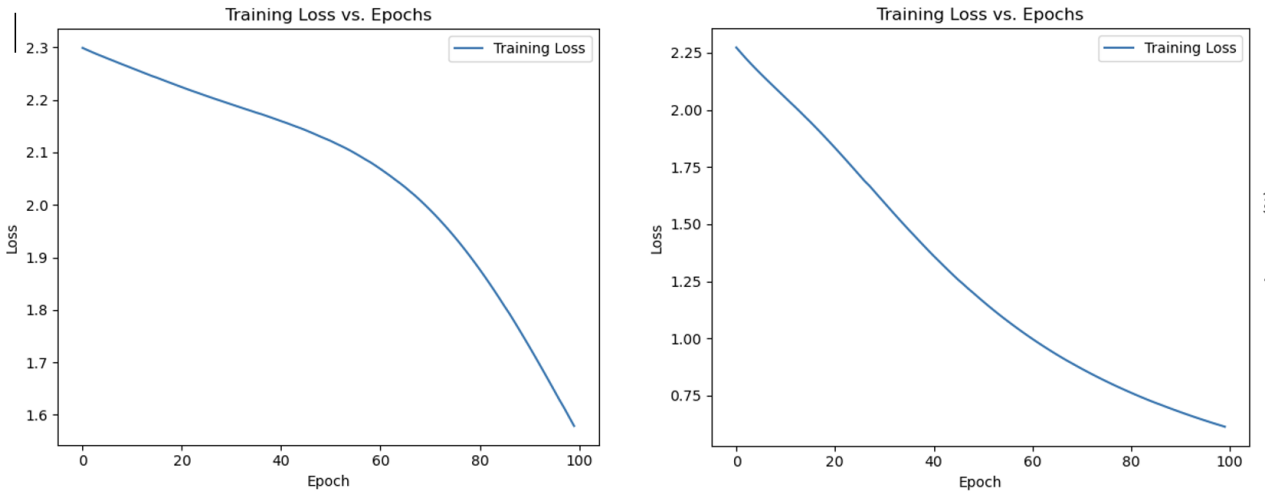

Figure 2: Loss vs Epochs

(iii) Training Time

* The training time of both models for current run - **all reduce** is taking 3.03 seconds and **ring-all reduce** 2.80 seconds. This indicates that part (b) is performing better in aspects of training accuracy, training time and improvement over loss.
* The dataset used in both cases can significantly influence the results, certain datasets may be easier to learn, leading to the faster accuracy improvement.

```
Rank 0, Epoch 98, Loss: 1.4445, Accuracy: 72.00%
Rank 2, Epoch 98, Loss: 1.5006, Accuracy: 72.00%
Rank 3, Epoch 98, Loss: 1.4609, Accuracy: 68.00%
Rank 1, Epoch 98, Loss: 1.4999, Accuracy: 52.00%
Rank 2, Epoch 99, Loss: 1.4825, Accuracy: 72.00%
Rank 0, Epoch 99, Loss: 1.4247, Accuracy: 72.00%
Rank 1, Epoch 99, Loss: 1.4820, Accuracy: 52.00%
Rank 3, Epoch 99, Loss: 1.4436, Accuracy: 68.00%
Rank 3, Epoch 100, Loss: 1.4265, Accuracy: 68.00%
Rank 2, Epoch 100, Loss: 1.4645, Accuracy: 72.00%
Rank 0, Epoch 100, Loss: 1.4052, Accuracy: 72.00%
Rank 1, Epoch 100, Loss: 1.4643, Accuracy: 52.00%
Rank 2, Training Time: 3.03 seconds
Rank 3, Training Time: 3.03 seconds
Rank 1, Training Time: 3.03 seconds
Rank 0, Training Time: 3.03 seconds
Averaged Train Losses: [2.3190438747406006, 2.3154043555259705, 2.311843991279602, 2.3083505630493164, 2.304912090301
5137, 2.3015183806419373, 2.2981602549552917, 2.2948291301727295, 2.2921221256256104, 2.288825750350952, 2.2855518460
273743, 2.282266139984131, 2.279024839401245, 2.2757246494293213, 2.2724117636680603, 2.269082009792328, 2.2657312750
816345, 2.2623568177223206, 2.258955180644989, 2.2555335760116577, 2.252069652080536, 2.248570442199707, 2.2450342178
344727, 2.2414584159851074, 2.237840950489044, 2.2341794967651367, 2.2304720878601074, 2.2267160415649414, 2.2230663838
95606995, 2.219208240509033, 2.215293288230896, 2.2113178372383118, 2.207376778125763, 2.2032703161239624, 2.19909077
88276672, 2.1948324441909979, 2.1904895305633545, 2.1860556602478027, 2.1815239787101746, 2.1768869757652283, 2.172136
425971985, 2.1672635674476624, 2.162259817123413, 2.1571150422096252, 2.1521793603897095, 2.1473053693771362, 2.14170
43805122375, 2.1359214782714844, 2.129961311817169, 2.123779594898224, 2.1173800826072693, 2.1107505559921265, 2.1043
225526809692, 2.0977675318717957, 2.091073989868164, 2.083499014377594, 2.0755918622016907, 2.0673800110816956, 2.059
239387512207, 2.050395578145981, 2.0418331623077393, 2.032314211130142, 2.0224314630031586, 2.012634813785553, 2.0020
041465759277, 1.9909804463386536, 1.979555904865265, 1.9677417278289795, 1.9561885595321655, 1.9446581304073334, 1.93
16338300704956, 1.9181680083274841, 1.9046920835971832, 1.8916971683502197, 1.8785884380340576, 1.8635331690311432, 1
.849031329154968 3, 1.8342526257038116, 1.8185290098190308, 1.8020019829273224, 1.785133183002472, 1.7679515779018402,
 1.750480741262436, 1.7327476143836975, 1.7147763073444366, 1.6968246698379517, 1.6784626841545105, 1.659952431917190
6, 1.6413249969482422, 1.6226114928722382, 1.603842705488205, 1.585049360990524 3, 1.5664933919906616, 1.5477323830127
716, 1.5290318131446838, 1.5120707154273987, 1.494555652141571, 1.476458489894867, 1.4581830203533173, 1.440134137868
8812]
Averaged Train Accuracies: [9.0, 9.0, 9.0, 9.0, 10.0, 10.0, 9.0, 13.0, 15.0, 18.0, 17.0, 17.0, 17.0, 17.0, 16.0, 17.0
, 16.0, 16.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 26.0, 27.0, 28.0, 29.0, 30.0, 30.0, 30.0, 29.0, 29.0, 32.0, 31.0, 3
1.0, 30.0, 30.0, 30.0, 30.0, 30.0, 29.0, 29.0, 29.0, 29.0, 29.0, 31.0, 32.0, 32.0, 32.0, 33.0, 35.0, 35.0, 36.0, 36.0
, 38.0, 39.0, 41.0, 43.0, 43.0, 43.0, 46.0, 46.0, 47.0, 48.0, 49.0, 50.0, 52.0, 52.0, 52.0, 52.0, 54.0, 55.0, 56.0, 5
6.0, 59.0, 59.0, 60.0, 60.0, 61.0, 61.0, 61.0, 61.0, 62.0, 62.0, 63.0, 64.0, 64.0, 64.0, 64.0, 65.0, 65.0, 65.0, 65.0
, 65.0, 65.0, 66.0, 66.0, 66.0, 66.0]
Average Training Time: 3.03 seconds
Final Training Accuracy: 66.00%
```

Figure 3: Loss vs Epochs

Figure 4: Loss vs Epochs

References

[1] M. Bosi et al., "Convolutional coding: An overview and new results," in *IEEE Transactions on Communications*, vol. 46, no. 9, pp. 1092-1103, Sept. 1998. [Online]. Available: