Lab 2 CSE 5194.01

Luyu Liu, Subash Chebolu, Zicong Zhang[[1]](#footnote-1)

{liu.6544, chebolu.1, zhang.5157}@osu.edu

Disclaimer: Due to the limit number of group members, after consulting Dr. Panda, we change the number of models from 3 to 2. After the change, it is much more reasonable to distribute and balance the workloads for each group member.

1. (20 points) Investigate one or two options for running distributed (multi-node) training for your assigned Deep Learning framework and explain your understanding by highlighting major components/technologies needed for it. a. Write a paragraph and draw a block diagram to explain your understanding. b. If there are multiple choice, list them down and highlight which approach have you used. E.g. MPI or gRPC or a combination of both for TensorFlow

PyTorch’s main in-built distributed implementation is in the package DataDistributedParallel which allows users to quickly set up and run in a distributed environment. The DataDistributedParallel package relies on a lower level implementation from the torch.Distributed package. The torch.Distributed package is where users can define algorithms for communicating during the session.

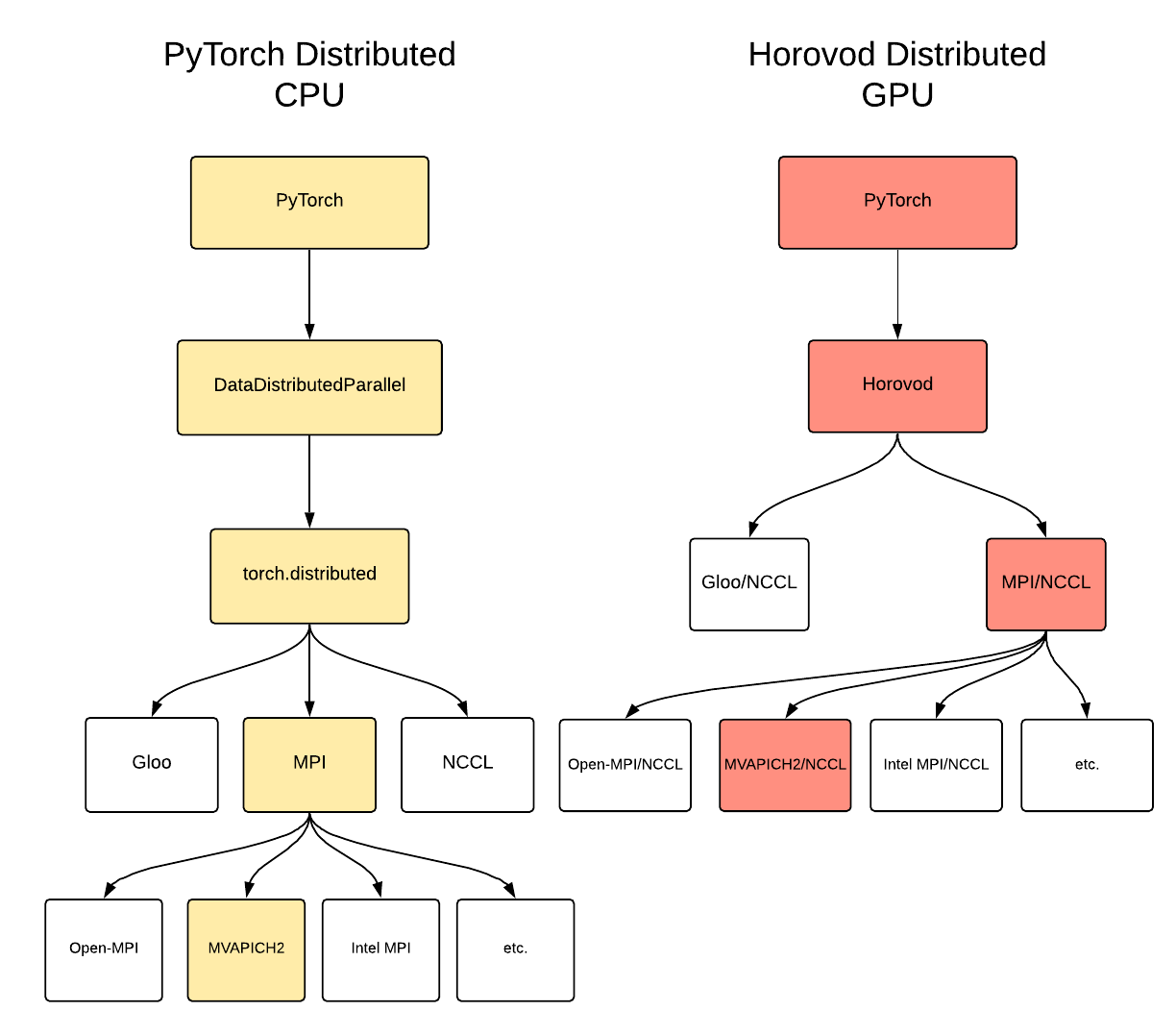


Figure 1 PyTorch distributed and Horovod distributed environment

The torch.Distributed package currently supports 3 backend communication frameworks: facebook’s Gloo, MPI’s various flavors, and NCCL. Figure 1 shows a hierarchical structure flow chart for both environment.

Although this was sufficient for our single-node and multi-node CPU training our team utilized a 3rd party framework for our GPU training called Horovod. This framework’s goal is to allow for easy and efficient transformation from a single-node GPU session to a multi-node GPU session. With minimal code changes, this framework allows for quick scaling across distributed GPU clusters. This framework uses MPI and NCCL in tandem for communication but MPI can be switched out for Gloo if necessary.

For our training, we tried PyTorch’s DataDistributedParallel package with an MVAPICH2 MPI flavor backend for the CPU training; but it needs higher privilege to install the required package and it is too esoteric to us. Instead, we used the 3rd party Horovod framework with MVAPICH2 flavor MPI + NCCL backend for GPU training.

1. Choose two different DNN models available for your chosen dataset. a. AlexNet, ResNet, and MobileNet for ImageNet dataset b. Use any model for Word2Vec that you can find for your assigned DL framework. c. Linear Model, Linear Model with Crosses, and DNN for Criteo Click Logs. (Luyu & Zicong)

For this experiment, we chose the Word2Vec 1- billion words dataset.

1. Choose two different DNN models available for your chosen dataset. a. AlexNet, ResNet, and MobileNet for ImageNet dataset b. Use any model for Word2Vec that you can find for your assigned DL framework. c. Linear Model, Linear Model with Crosses, and DNN for Criteo Click Logs. (Luyu & Zicong)

For the first training model, we still chose the gated convolutional networks [1] as the lab 1: a major contribution of the model is that it was a new attempt to use a non-recurrent approach to attain near performance as recurrent neural networks. The model is based on convolutional networks with a gating mechanism. The code is originated from a GitHub repository [2], which was developed from the methods from the paper and later adjusted and parallelized by Luyu Liu [3].

The other model we choose is Long short-term memory model (LSTM) with attention [4], [5]. LSTM is discussed in the class: it is a variant of RNN and have good performance to deal with the long sequence. Besides, the model has an attention mechanism, which will highlight the important information from the contextual information by setting weight [6]. We changed the learning rate to 0.001 which will take more time but more suitable for LSTM.

1. (20 points) Run the experiments for each DNN using a single node (If you have these numbers from Lab #1, please re-use these to save SUs. Otherwise, you can re-run these experiments.)
2. Run the GPU version

The GPU version is run on the Owens P100 node with one node.

1. Run the CPU version

The CPU version is run on the Owens CPU cluster with 28 cores in one node.

1. Report throughput (training time) in terms of samples/second or images/second for both CPU and GPU
2. Vary the batch size and find out the best batch size that gives you the highest throughput.

**[Gated CNN] (Luyu&Zicong)**

|  |  |  |
| --- | --- | --- |
|  | accuracy | training time |
| Batch size = 32 | 0.1077 | 538 |
| Batch size = 64 | 0.4328 | 291 |
| Batch size = 80 | 0.4323 | 273 |

Table 1 Average epoch time among all epochs for each GPU experiment

|  |  |  |  |
| --- | --- | --- | --- |
|  | CPU with 7 threads | CPU with 14 threads | CPU with 28 threads |
| Batch size = 32 | 0.4018 | 0.4021 | 0.0898 |
| Batch size = 64 | 0.4297 | 0.4303 | 0.4295 |
| Batch size = 80 | 0.4341 | 0.4303 | 0.4353 |

Table 2 Final accuracy among all epochs for each CPU experiment

|  |  |  |  |
| --- | --- | --- | --- |
|  | CPU with 7 threads | CPU with 14 threads | CPU with 28 threads |
| Batch size = 32 | 4109.6 | 3789.2 | 5235 |
| Batch size = 64 | 3683.4 | 2178.6 | 3185.2 |
| Batch size = 80 | 2044.4 | 1843.6 | 2038.6 |

Table 3 Average epoch time among all epochs for each CPU experiment

The results show that batch size = 80 is the best option. We will continue our training with this batch size.

**[LSTM with attention] (Luyu)**

|  |  |  |
| --- | --- | --- |
|  | accuracy | training time (s) |
| Batch size = 32 | 0.1952 | 925 |
| Batch size = 64 | 0.1951 | 569 |
| Batch size = 80 | 0.1951 | 464 |

Table 4 Average epoch time among all epochs for each GPU experiment with one node

The accuracy of the model is not increasing significantly with the smaller batch size. So, considering performance-wise improvement, we chose batch size = 80.

1. (40 points) After you have the data for the best batch size for a single node, run the experiment with the best batch size for 1, 2, 4, and 8 nodes.
2. Run for CPU only

The GPU version is run on the Owens P100 node with 2-8 node. Considering all the time and computational budget, instead of running the code on one node again, we use the old data in the last lab.

1. Run for GPU only

The CPU version is run on the Owens CPU cluster with 28 cores in 1-8 node.

1. Report the throughput for both versions
2. Report the scalability/speedup for multiple nodes by creating a graph that presents images/second on the y-axis and #nodes on x-axis.

**[Gated CNN] (Luyu)**

The statistics of each experiments are shown below, both for GPU and CPU.

|  |  |  |
| --- | --- | --- |
| # Node | accuracy | training time (s) |
| 1 | 0.4114 | 273/278 |
| 2 | 0.4015 | 328 |
| 4 | 0.4042 | 197 |
| 8 | 0.3910 | 101 |

Table 5 benchmark for each multi-node GPU experiment

|  |  |  |
| --- | --- | --- |
| # Node | accuracy | training time (s) |
| 1 | 0.4263 | 2038 |
| 2 | 0.4085 | 4043 |
| 4 | 0.4002 | 2105 |
| 8 | 0.4012 | 1132 |

Table 6 benchmark for each multi-node CPU experiment

Figure 2 Scalability of GPU experiments with 2, 4, and 8 distributed nodes (blue: actual training time; gray: performance loss compared to the ideal scenario)

Figure 3 Scalability of GPU experiments with 2, 4, and 8 distributed nodes (blue: actual training time; gray: performance loss compared to the ideal scenario)

**[LSTM with attention] (Luyu)**

|  |  |  |
| --- | --- | --- |
| # Node | accuracy | training time (s) |
| 1 | 0.1951 | 464 |
| 2 | 0.1951 | 459 |
| 4 | 0.1952 | 256 |
| 8 | 0.1951 | 139 |

Table 6 benchmark for each multi-node GPU experiment for LSTM

|  |  |  |
| --- | --- | --- |
| # Node | accuracy | Training speed (sample/s) |
| 1 | 0.1951 | 159 |
| 2 | Job Killed | Job Killed |
| 4 | Job Killed | 4.4 |
| 8 | Job Killed | Job Killed |

Table 7 benchmark for each multi-node CPU experiment for LSTM

1. (20 points) What can you conclude from this study? Write a few paragraphs to explain your results and the insights in-depth.

**[Gated-CNN] (Luyu)**

**Training time.** The trend of the training time is generally decreasing, which is intuitive due to the effective of parallelism training. However, from 1 node to 2 nodes, the performance will actually become worse. This could be because of the initial overhead of horovod training, since single node training does not require synchronization and library setup. Also, we trained all the model based on the very first epoch, which is reported to have unique and irregular pattern and generally longer training time. Comparing CPU with GPU, GPU still dominates the CPU in terms of performance for more than 10 times.

**Scalability.**  Due to the special feature of node = 1 and epoch 1, the time is not comparable. Consequently, we compare the scalability based on node =2. The scalability of CPU and GPU experiments is shown in Figure 1 and Figure 2. The general trend is that horovod cannot achieve 100% ideal performance improvement.

Comparing CPU with GPU, we can also see the difference: CPU’s performance loss is much higher than GPU’s. This is because besides the communication and synchronization between nodes, CPU nodes moreover have overhead within. Also, we noticed that GPU is larger than CPU in terms of the proportion of performance loss. This is probably due to the relatively constant overhead of the communication, thus making the proportion in the less time-consuming GPU experiments larger.

**Accuracy.** The trend of the accuracy does not have a clear pattern due to small training epoch. However, the non-parallel versions have the largest version.

**[LSTM with attention] (Luyu)**

**Training time.** The trend of the training time is generally decreasing, which is intuitive due to the effective of parallelism training. However, for CPU multi-node, we found that most of the experiments cannot be finished within the predefined wall-time of 2 hours (7200 seconds). The main reason can be the inefficiency of CPU devices and the large communication overhead within the node; also, compared with gated CNN, the LSTM may have more dependencies which limits its performance.

**Scalability.**  Although the performance of node = 2 is better than node =1, because of the disproportionate scalability, we will compare the scalability based on node =2. The scalability of GPU experiments is shown in Figure 4. The general conclusion is that horovod cannot achieve 100% ideal performance improvement but the performance loss is not very large.

Comparing CPU with GPU, we can also see the difference: First, the LSTM’s node = 2 experiment’s performance is better than node =1, while Gated CNN is otherwise. This could be because LSTM with attention model is much simpler than Gated CNN and has fewer parameters, which subsequently make its setup overhead less. Also, CPU’s performance loss is much higher than GPU’s. This is because besides the communication and synchronization between nodes, CPU nodes moreover have overhead within. Also, we noticed that GPU is larger than CPU in terms of the proportion of performance loss. This is probably due to the relatively constant overhead of the communication, thus making the proportion in the less time-consuming GPU experiments larger.

**Accuracy.** The model could reach a performance limit considering low epoch time and learning rate setting. So, the trend of the accuracy does not have a clear pattern.

Figure 4 Scalability of GPU experiments with 2, 4, and 8 distributed nodes (blue: actual training time; gray: performance loss compared to the ideal scenario)

Reference:

[1] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 933–941.

[2] J. Ohmura, “Gated-Convolutional-Networks,” 2019. [Online]. Available: https://github.com/jojonki/Gated-Convolutional-Networks. [Accessed: 12-Oct-2019].

[3] L. Liu, “CSE-5194,” 2019. [Online]. Available: https://github.com/luyuliu/CSE-5194.

[4] Y. Wang, M. Huang, and L. Zhao, “Attention-based LSTM for aspect-level sentiment classification,” in *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2016, pp. 606–615.

[5] Adbruijn, “offensive-text-classification,” 2019. [Online]. Available: https://github.com/adbruijn/offensive-text-classification.

[6] G. Liu and J. Guo, “Bidirectional LSTM with attention mechanism and convolutional layer for text classification,” *Neurocomputing*, vol. 337, pp. 325–338, 2019.

1. In first name alphabetical ordering [↑](#footnote-ref-1)