Lab 3 CSE 5194.01

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

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

1. Profile your distributed training experiments conducted in Lab #2 further to gain insights about performance. Profiling analysis should include:
2. Breakdown of computation and communication time
3. Identification of any specific communication pattern, (e.g. Broadcast, All-to-all, or something other)
4. Factors that affect the amount of computation. (e.g. batch size)
5. Factors that affect the amount of communication performed (e.g. model type)

**Breakdown of computation and communication time**

We profiled our models using the python built-in time module and wrapping critical training and communication code within our scripts. We break down the communication as the time required to broadcast the initial weights to the other distributed processes and the time required to share all the loss values with all processes in an all-to-all communication step when training for each batch. The training time is the time it takes to run 1 full epoch through the dataset and train the model minus the loss communication step and the prior setup of creating the model and loading in the dataset. Below is a table of our findings.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Gated CNN, batch size 32 | LSTM, batch size 32 | Gated CNN,  batch size 128 | LSTM, batch size 128 |
| initial broadcast | 0.15 | 0.33 | 0.24 | 0.25 |
| loss communication time | 286.82 | 329.23 | 59.98 | 74.21 |
| total communication time | 286.97 | 329.57 | 60.23 | 74.47 |
| training time | 555.02 | 609.02 | 189.35 | 195.73 |

Table 1: Training time profile over 1 epoch of training.

The models we used for the lab are a Gated Convolutional Neural Network (Gated CNN) [1] and a Long Short-Term Memory (LSTM) model with Attention [2] on the Word2Vec dataset.

**Identification of any specific communication patterns**

The initial broadcast takes a very little time penalty. The majority of communication cost is still within the distributed optimizer, which is introduced by horovod library.

**Factors that affect the amount of computation**

The table above shows that both models decreased significantly in training time and total communication time when a larger batch size was used showing that batch size is a significant factor in decreasing both computation time and communication time.

**Factors that affect the amount of communication performed**

Although the Gated CNN has many more layers than the LSTM model, the latter contains more parameters. When comparing the two model’s performance when the batch sizes are the same it is clear that the LSTM is taking a much longer time in both training/computation time and communication time. This is very likely due to the prior mentioned larger number of parameters it has compared to the Gated CNN which directly affects how much more bandwidth it needs to use to communicate its state throughout the training process as well as how much more computation is required for the model to train in general. Therefore, the model or specifically the number of parameters is an inversely correlating factor in determining both communication and computation time. Besides, batch size is also a critical factor. Larger batch size will make the communication time less.

1. Investigate opportunities for performance improvement for your chosen DL framework. This is an open research question, so you can explore different options to accelerate the training time.

A few example options are provided below:

* 1. Overlapping of communication and computation
  2. Using multiple CUDA streams for computation and communication
  3. Finding out the best set of hyper-parameters like learning rate to train quickly
  4. Using different communication libraries to improve performance (e.g. OpenMPI vs. MVAPICH2)

We explored several possibilities within the structure of pytorch + horovod + MPI.

**Overlapping**

Due to the limitation of horovod library, we cannot operate in some very low-level mechanism, such as the operating each thread’s communication and computation. However, this is not necessarily a disadvantage. The primary benefit of horovod is that we can make the distributed training work without manually setting up the tedious options. Without a good optimization strategy, the performance does not guarantee to be better than the default.

**Multiple CUDA streams**

For the same reason, horovod does not support manipulating CUDA streams. However, outside the horovod library in the native pytorch CUDA API, it does support CUDA streams setting. A CUDA stream is a linear sequence of execution that belongs to a specific devices. One normally does not need to create one explicitly: by default, every device will use their own default stream. For the case of our experiments, it is also the case.

**Hyper-parameters**

We experiment with different batch sizes on multiple nodes (2 and 4) on transformerXL model [3]–[5]. As indicated in lab2, the transformerXL is a relatively large model with 300 million parameters, and thus the communication overhead is very large when distributed training on multiple nodes. For 4 nodes, the model reached about 2000ms per batch and the loss was not converging at all. However, we find that the training will converge if we train the model with decreased batch size of 10.

Though the model is still spending too much time on communication, the model can converge to about 300 ppl and 5.8 loss and 372ppl and 5.92 loss for 2 node and 4 nodes respectively at both step 8000, which achieve pretty similar results compared with our single node GPU training. Although more experiments need to be conducted to make sure that batch size may be a factor affecting the result of distributed training, the model can certainly achieve better results with more hyperparameter tuning such as learning rate and warmup step.

Moreover, it was observed on other dataset, that adding “bind-to none” and “map-by slot” could decrease communication time, which could leave for further improvement of the model.

**MPI version**

The default MPI release we are using is mvapich2/2.3.1-gpu, which is the MPI release over InfiniBand and other high-speed interfaces [6]. We compared the Gated CNN model’s training performance over these two MPI releases and the results are shown in Table 2.

|  |  |  |
| --- | --- | --- |
|  | Mvapich2/2.3.1-gpu | Openmpi/4.0.1 |
| Total training time (secs) | 391 | 371 |
| Computation (secs) | 287 | 244 |
| Communication (secs) | 104 | 127 |

Table 2: Training performance's over the two MPI versions

In general, mvapich2 can achieve lower communication cost compared to vanilla openmpi 4.0.1; however, the overall computation time and the total training time are much smaller for openmpi 4.0.1, which is anti-intuitive from the experience with the results derived by Tensorflow. We think the reason is because the unique feature of pytorch, which is a define-by-run framework. Moreover, as stated on its Github page [7], horovod works the best with NCCL backend unless one has better communication reduce protocol, which could leave room for experiments to further improve communication time.

**Discussion**

Language modeling has been a crucial part of the training effective deep models. Deep neural networks such as BERT and XLnet have achieved SOTA results on several NLP tasks by practicing the pretraining+fine-tuning pipeline. However, the large amount of computation needed for effective pre-training process has been the major issue that hinders research process. Given the fact that models like transformer and convolutional neural networks can be easily parallelized on GPUs, building up an effective distributed model paradigm becomes very crucial when dealing with certain NLP tasks.

Moreover, there is still need for methodology for more effective distributed learning over large models; we have demonstrated that distributed training on could be done through many training packages such as horovod by adding a few lines to original single GPU implementations, such as GatedCNN. Also, distributed learning could be done on large models like transferXL with reasonable results from an accuracy perspective. For further improvements, one could develop a more efficient way distributed scheduler and optimizer for the implementation of transformerXL and more optimization could be done by using different communication library as well.

Reference:

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

[2] 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.

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

[4] T. Wolf *et al.*, “Transformers: State-of-the-art Natural Language Processing,” *arXiv Prepr. arXiv1910.03771*, 2019.

[5] Z. Yang, “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context,” 2019. [Online]. Available: https://github.com/kimiyoung/transformer-xl. [Accessed: 22-Oct-2019].

[6] H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, and D. K. Panda, “MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters,” *Comput. Sci. Dev.*, vol. 26, no. 3–4, p. 257, 2011.

[7] Horovod, “Horovod,” 2019. [Online]. Available: https://github.com/horovod/horovod. [Accessed: 03-Dec-2019].

1. Order in name alphabetically. [↑](#footnote-ref-1)