Lab 1 CSE 5194.01

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

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

We arrange the ordering of questions so that relevant questions are close to each other. Followed by each question, the corresponding author(s) name is attached; report is merged and written up by Luyu.

1. (20 points) Run MNIST and CIFAR10 training on a single node (CPU and GPU) using the DL framework assigned to your team. (Subash)

The script is hosted in the Lab GitHub [1].

1. (5 points) What are the performance trends for MNIST and CIFAR10 for CPU and GPU? (Subash)

|  |  |  |
| --- | --- | --- |
| Accuracies | MNIST | CIFAR10 |
| CPU | .99 or 99% | .40 or 40% |
| GPU | .97 or 97% | .47 or 47% |

Table Average accuracy for each epoch

|  |  |  |
| --- | --- | --- |
| Training Time | MNIST | CIFAR10 |
| CPU | 48 Sec | 19.4762 Sec |
| GPU | 17.995 sec | 16.68 Sec |

Table Average epoch time for each epoch

These experiments were run with a batch size of 64 and the same optimizers. While MNIST shows higher accuracies, CIFAR10 is faster to train.

1. (5 points) What DNN model works best on CPU vs. GPU and why? (Subash)

It seems that in both the CPU and the GPU the smaller model works faster although in general the CPU is much more affected by larger models that the GPU. The models that I used are very similar to each other in terms of layers but the number of parameters in the MNIST model is much larger than the parameters in the CIFAR10 model. The assumption for why the GPU times are similar could be due to how parallelization is much more prevalent in this pipeline since there are many cores of the GPU whereas the CPU cannot parallelize as much and is greatly affected by the added parameters and model size.

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 chose the gated convolutional networks [2]: 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 [3], which was developed from the methods from the paper and later adjusted by Luyu Liu [4].

The other model we choose is transformer-XL [5], [6], which builds on the original transformer model that has been very popular in the field of natural language processing. Transformer model adds a recurrent layer to in the attention layer to further enhance to the model’s capability of predicting variable length input and long-range dependencies.

1. (50 points) Perform the above experiments on a single node (CPU and GPU) using the DL framework assigned to your team. For the single node CPU run, you can vary the number of cores and number of threads in the application. For GPU runs, there is no need to vary the number of threads, instead, use different batch sizes to find out the best performing batch size. (Luyu & Zicong)

For the gated convolutional networks, we conducted corresponding CPU experiments on Owen clusters and GPU experiments on Pitzer clusters for queuing time and computation time purposes. For the experiment of both CPU and GPU, we chose three values for the batch size: 32, 64, and 80. For the number of threads, we chose 7, 14 and 28 for CPU training. We will run 5 epoch for this model. For the transformer-XL, we adopted the same parameters (except CPUs of 7 cores) for the training. Each experiment’s results are shown in Table 3, Table 4, Table 5, Table 6, and Table 7.

1. (5 points) Mention the scale and size of your dataset and explain why you chose it? (Luyu & Zicong)

The scale of Word2Vec is rather large in terms of the quantity of the words; however, its overall size is the smallest of all three datasets. We wanted to choose the ImageNet datasets since we all have a background on the image processing with DNN, however, the size of the dataset is so large that it takes so long to download and unzip, considering both the instable speed of the network in the environment and the difficulties to share a large dataset between different accounts in the OSC environment. As a result, we chose Word2Vec for its abundancy and relative small volume.

Moreover, the dataset consists of many vocabularies, each of which is followed by a paragraph explaining the word. The model is trained on LM objects, which means the model is trained to learn to predict the next word given previous word sequence. As a result, accuracy and perplexity are then used as metrics to measure the quality of the model after certain amount of training.

1. (5 points) What is the best DNN architecture for your chosen dataset in terms of “accuracy” and in terms of “training time”? Describe if you find a suitable tradeoff between these two metrics. (Luyu & Zicong)

**Gated convolutional network model (Luyu)**

The gated convolutional network model is special for its innovative nature: it use convolutional network to train text dataset. However, it also shows non-trivial performance. For the training time, the model converged very fast and the test accuracy stabilized around 42% to 43% for most of the time. Table 3 shows the final accuracy after 5 epoch of each experiment under different hyperparameters and Table 4 shows the average training time of all 5 epoch for CPU experiments.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 3 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 4 Average epoch time among all epochs for each CPU experiment

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

Table 5 Accuracy and training time of GPU experiments

**Transformer-XL (Zicong)**

Transformer-XL is the SOTA model on this dataset. Although the best result achieved by the based model stated in the paper is about 23 in perplexity, we cannot manage to achieve the same results due to hardware limitations and lack of gradient accumulation techniques that helps to put larger batch under limited memory. Overall, there is an increasing performance as we increased batch size, which is reasonable given that the best result was trained using batch size of 224. Despite the fact that Transformer-XL took longer time and required GPU for the training, the model has achieved relatively much better results.

The training results after 20hs is shown in Table 6 and Table 7. Due to the large computational overhead and relatively poor performance of Owens CPUs, we cannot finish even a single epoch. However, we calculate the finished steps in a same time. We choose the smaller batch size when comparing training speed between CPU and GPU because the model has about 3 hundred million parameters and the training will be very inefficient where the model become too heavy to train for CPU.

|  |  |  |  |
| --- | --- | --- | --- |
| Batch Size: 16 | CPU(14 cores) | CPU(28 cores) | GPU(single) |
| Training Steps | 20000 | 23000 | 210000 |

Table Speed comparison between CPU and GPU

|  |  |  |  |
| --- | --- | --- | --- |
| Single GPU | Accuracy | Perplexity | Loss |
| Batch size: 32 | 0.59 | 171 | 5.15 |
| Batch size: 64 | 0.69 | 108 | 4.68 |
| Batch size: 80 | 0.72 | 96 | 4.57 |

Table Performance comparison between different batch sizes (on GPU)

The paper trains the model using 4 NVidia GPUs, which is not feasible for our experiment. We are using a single V100 GPUs and are observing various diverging training phase. Thus, we take the best performance during the training and record as the result shown above for each different batch.

**Comparison**

In terms of training time, gated convolutional network (GCN) will overwhelmingly outperform the transformer-XL (TXL) model. This is because the drastic difference in the trainable parameters: GCN model only has 3 million parameters while TXL has 3 hundred million parameters.

In terms of performance, due to the advanced design of the TXL model and the innovative nature of the GCN model, TXL’s accuracy (59% - 72%) is significantly higher than GCN’s (40% - 43%). However, GCN’s results are also non-trivial.

1. (5 points) Is the GPU always better than CPU for training with larger dataset? Explain your observation. (Luyu & Zicong)

In terms of training time, GPU shows an overwhelming advantage over CPU clusters, which can be 10 times faster than CPU. This shows the powerful lead of GPU. However, in terms of final accuracy, we can see very close accuracy for GPU and CPU; for some cases, CPU with 28 threads even outperforms GPU for batch size = 80. As for transformer-XL, GPU is always preferred and have achieved better results solely due to the demands of computation resource. GPU would be optimal given large models and more number of parameters.

A very interesting phenomenon with gated convolutional network model is that with batch size = 32, CPU with more threads and GPU diverged and achieved much worse performance than CPU. This could be because the optimizer the model uses (Adadelta) is not suitable for small batch. Another possibility is because of the model per se: when running batch size = 1024 after epoch 40, although the loss is steadily decreasing, the model’s test accuracy decreases. This is a very clear signal of overfitting. However, again, we cannot be sure since we did not train the model for more than 100 hours like the paper.

1. (5 points) For the single-node CPU run, do you see any trend when varying the number of cores and number of threads in the application? Explain your observation. (Luyu & Zicong)

The GCN model shows a very interesting pattern: CPU with 14 cores shows better performance than both 28 cores and 7 cores. Larger amount of CPU cores can provide more computational power, however, due to larger communication overhead between different cores, the overall performance will be worse; for smaller amount of CPU cores, the computational power becomes the bottleneck. As a result, for this gated convolutional model, it is the best to choose medium amount, compared to too large or too few amount of CPU cores.

For the TXL model, doubling the cores will not achieve 100% scale-up performance, which only improve the performance by 15%. This is of course caused by the same communication overhead with more cores. Due to the limited CPU cores on a single OSC node and the difficulties of large model’s training on CPUs, it is very hard to predict the scale-up performance with more cores.

Reference:

[1] S. Chebolu, “Lab1 MNIST and CIFAR10 code,” 2019. [Online]. Available: https://github.com/luyuliu/CSE-5194/blob/master/labs/lab1/subash/Lab1\_Training.py.

[2] 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.

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

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

[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] Z. Dai *et al.*, “Transformer-xl: Attentive language models beyond a fixed-length context,” *arXiv Prepr. arXiv1901.02860*, 2019.

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