Accelerating Neural Network Training using Beowulf Cluster Computer

Héctor Mejía, Franz Guzmán

December 2019

1 Introduction

It is undoubtedly true that modern society consumes huge volumes of data to maintain its daily activities, and the development of information and communication systems serve this purpose well. With the booming of artificial intelligence, academia, business and societal fields were revolutionized, because it provided tools never seen before to make automatic prediction, recommendation, classification and decision tasks possible without the intervention of humans. However, the problem in this setting is that normally, computation for real life data to make those tasks involve expensive data-centers and highly specialized infrastructure, which makes technology prototyping and development very difficult. The aim of this work is to take advantage of open-source software for deep learning, message passing frameworks, Linux secure shell and homogeneous, relatively cheap hardware to develop a cluster computer with the sole purpose of speeding up neural network training and prediction, while still achieving a prediction accuracy level similar to that of a sequential algorithm.

First we provide information involving the project in the background section, then we state the dataset used for this experiment. After it, we explain the computer used for the sequential training algorithm as well as the cluster computer setting used for the parallel version of the algorithm in the hardware setup sections. Later we introduce the neural network model architecture, hyper-parameters and synchronization methods for parallel training in the neural network training section. Finally, we provide results for both sequential and parallel versions of the algorithm and discussion over the feasibility of the cluster and the results aforementioned.

2 Background

The development of this work involved a wide range of concepts of computer science, operating systems, artificial intelligence and optimization techniques.

2.1 General Concept of distributed systems

A distributed system is a collection of processors that do not share memory or a computer clock. Instead, each processor has its own local memory and they communicate with one another through a communication network using message passing.[1]

2.2 Operating systems and distributed system structures

An operating system is an intermediary program between the user and the computer hardware that provides environments in which a user can execute programs efficiently. There are some aspects that encompass a distributed operating system, but for the purposes of this work, we focus only on data, computation and process migration mechanisms. For a very basic distributed system, we generally need to have copies of the parallel program on every node we want to execute. One approach to data migration is the Network File System Protocol (NFS). This protocol allow users to access and update files on remote computers as though they were on the local computer. A concrete implementation is a client/server application over a TCP/IP network to allow remote file access. [1]

Then when data availability has taken into account, the system must be able to transfer computation across its nodes. There are multiple approaches to computation migration. One of them is through messages. In here, the invoking program sends a message to a process and relies on the process itself and the underlying infrastructure to select and invoke the actual code to run. [2]

2.3 Artificial Neural Networks

Artificial Neural Networks are computational systems inspired the cognitive function of the human brain. The system learns based on examples without being programmed with an initial set of rules. Those examples are processed in the network's internal layers of nodes. Each node is given a relative weight analogous to synaptic strengths in biological brains to determine the importance of each node for the output. The output layer correspond to a single node for prediction tasks or multiple nodes to return probabilities of being one class or another for classification tasks. ANNs use a propagation function that maps the output of some hidden layer and its weigths, to the input of the next layer as a weighted sum. A bias term is often added. ANNs are able to adjust their inner weights to provide optimal solutions, given enough example data.

One particular type of artificial neural network is the convolutional neural network, which is widely used for classification tasks given images and for computer vision. It makes use of hidden convolution layers in which a window convolves on the output of the predecessor layer to extract automatic features for the next layer. [3]

3 Hardware setup

For the development of this project three computers connected to a switch by Ethernet cables were used to develop the cluster. All three computers were of the exact same model. For the execution of the sequential version of the program, one of those computers was used. Specifications for the machines are as follows:

- Motherboard: HP Placa base 82A2 KBC Version 06.15
- CPU: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
- RAM memory: Samsung M378A1K43CB2-CRC 8GiB
- Storage: Seagate ATA Disk ST1000DM003-1SB1 1TB Partitioned
  Partition : Linux partition EXT4 50GiB mounted on /
  Partition 2: Linux partition EXT4 407GiB mounted on /home
- Network: RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller 1Gbit/s
- Network PCI: Intel PCI bridge 200 Series PCH PCI Express Root Port

Specification on the network switch are:

- Catalyst 2960-X Series 1Gbit/s

4 Neural Network model and training

The architecture design of the neural network include two convolutional layers. The first accepts grayscale images and returns images with 6 channels of data, with a window size of 3x3 pixels. The second accepts images with 6 channels and returns images with 16 channels, with a window size of 3x3 pixels also. Two max-pooling layers with windows of size 2x2 pixels each proceeding a convolution layer. Three fully connected layers with 120, 84 and 10 nodes, respectively; with the first two having ReLU activation functions, defined as:

$$f(x) = max(0, x) \tag{1}$$

and the last one having a softmax function:

$$f(x_i) = \frac{exp(x_i)}{\Sigma_j exp(x_j)} \tag{2}$$

4.1 Gradient descent

Neural network learning algorithms explicitly minimize a cost function. In the case of the stochastic gradient function. Let us consider a function, $J(z, w)$, the loss function, that measures the cost incurred by a system defined by some parameter w processing an observation z.[4] Learning consists of finding the parameter w^* that minimizes the following cost:

$$C(w) = E(J(z,w)) = \int J(z,w)dp(z) \tag{3}$$

If the training set is finite, the distribution $dP(z)$ is discrete. Now the algorithm to minimize (3) is the following:

$$\begin{aligned} dP(z) &= \sum_{i=1}^{N} \frac{1}{N}\delta(z - z_i) \\ C(w) &= \frac{1}{\sum}{}_{i=1}^{N} J(zi_i, w) \end{aligned} \tag{4}$$

Each iteration of the stochastic gradient algorithm consists in drawing an example z at random and applying the parameter update rule:

$$w_{t+1} = w_t \epsilon_t \Delta_w J(z, w_t) \tag{5}$$

4.2 Cross entropy loss function

The loss function used in the ANN is cross-entropy. It is based on the notion of cross-entropy, which is a measure of distance between a true distribution p and an estimated distribution q [5], and can be defined as:

$$H(p,q) = -\sum_x p(x)\log(q(x)) \tag{6}$$

Using this measure, the cross-entropy loss can be defined as follows:

$$L_i = -f_{ai} + \log(\sum_j e_j^f) \tag{7}$$

The loss function is useful to track during training is the loss, as it is evaluated on the individual batches during the forward pass.

5 Overview of the dataset and experimental procedure

The dataset used for this work is the MNIST dataset. Corresponds to 28x28 single channel images of handwritten numbers from 0 to 9. The following picture shows a sample of the images: To evaluate the performance of the cluster, 12 experiments were set. Each experiment includes an increasing number of processor used for the computation, starting from the sequential algorithm, meaning that a single processor is employed, up to a parallel run with 12 processors, all 4 from each machine. The sequential results are set as a baseline to be compared

Figure 1: Sample batch from training data

with the parallel runs. Then loss values, execution time and accuracy are retrieved from the experiments. An additional metric called speedup is computed, to know how many times faster the parallel algorithm is, over the sequential, using:

$$speedup(t_{parallel}, t_{sequential}) = \frac{t_{sequential}}{t_{parallel}} \tag{8}$$

Finally, results are compared and plotted.

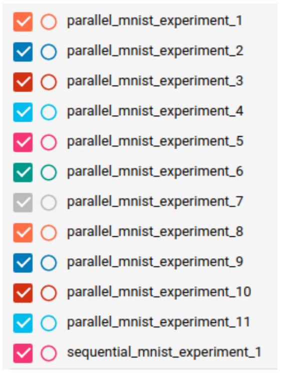
6 Results

Results from execution time and speedup given a number of processors are as follows:

No. Processors	Time	Speedup
1	75.079	1
2	62.33	1.205
3	54.09	1.388
4	56.31	1.333
5	47.39	1.584
6	39.77	1.888
7	36.39	2.063
8	49.7	1.511
9	31.84	2.358
10	34.56	2.172
11	28.82	2.605
12	27.31	2.749

Table 1: Results of execution time and speedup given processors

These results are also plotted, along with loss values and PR curve. Legend is provided to recognize each color in each graph. Note that for each experiment index, the number of processors is $index + 1$. So parallel experiment 1 uses 2 processors, and so forth.



✓ parallel_mnist_experiment_1
✓ parallel_mnist_experiment_2
✓ parallel_mnist_experiment_3
✓ parallel_mnist_experiment_4
✓ parallel_mnist_experiment_5
✓ parallel_mnist_experiment_6
✓ parallel_mnist_experiment_7
✓ parallel_mnist_experiment_8
✓ parallel_mnist_experiment_9
✓ parallel_mnist_experiment_10
✓ parallel_mnist_experiment_11
✓ sequential_mnist_experiment_1

Figure 2: Legend for all the plots

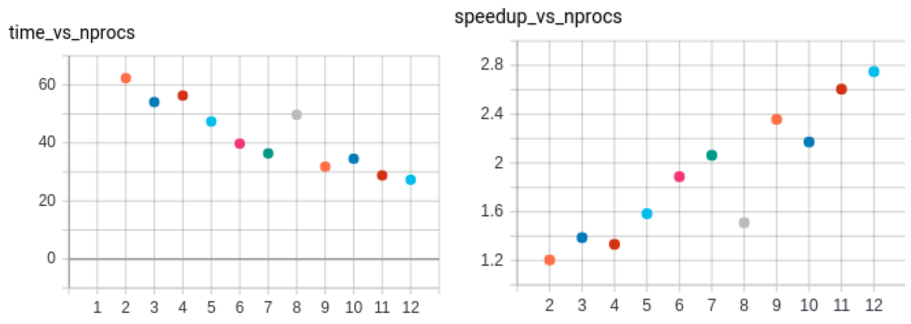


Figure 3: Time vs nprocs (left plot) speedup vs nprocs (right plot)

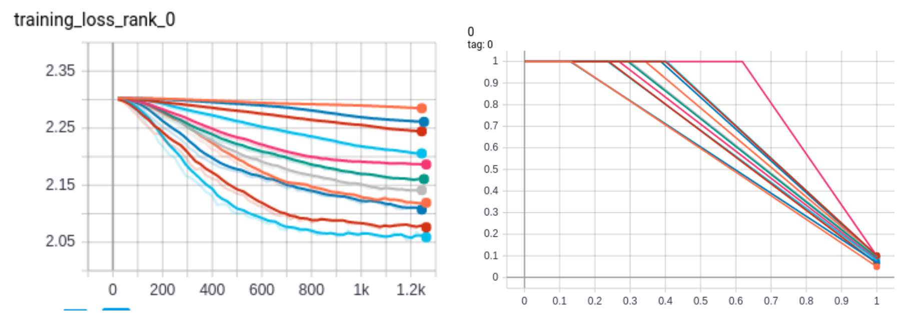


Figure 4: Loss Values for each experiment (left plot) PR curves for each experiment (right plot)

7 Discussion

Our results have shown, as seen in table (1), that the training time in distributed are better than in sequential with only 1 processor. This data is also plotted in figure (3). We can see that the time it takes to train after increasing the number of processor is in a linear relation. This is also seen in the speedup of the training.

The next two figures (4) tell us more about the learning rate of the NN and the accuracy of the same. It is notable that in rank 0 with 12 processor we have a desired good learning rate as it is not in linear relation. On the other hand the figure on the right side it's called a Precision-Recall curves and plot the positive predictive value (PPV, y-axis) against the true positive rate (TPR, x-axis). And it implies that the sequential algorithm might be better at $k = 0.6$ as it has a precision of 1. Then it's followed by the distributed algorithm of 12 processors on $k = 0.4$. And the worst achieved precision is from the distributed with 1 processor.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating Systeme Concepts.* 2005.

[2] B. R. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, "Introduction to Parallel Programming," in *Heterogeneous Computing with OpenCL*, pp. 1–13, Elsevier, 2013.

[3] Y.-S. Park and S. Lek, "Artificial Neural Networks," pp. 123–140, 2016.

[4] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[5] A. Bhardwaj, W. Di, and J. Wei, *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling.* Packt Publishing, 2018.