Training Neural Networks in Parallel

Jordan A. Ott

April 3, 2017

Abstract

In this paper we explore training neural networks as a parallel task. Our goal is to decrease the time required to train a neural network while maintaining the accuracy of the network. We will present a model of training neural networks through data parallelism. A single hidden layer network is used for this experiment, training on the MNIST dataset of handwritten digits.

1 Introduction

Multilayer Perceptrons (MLP), more commonly referred to as Feedforward Neural Networks have become extremely popular in recent years. They have been applied to nearly every domain and have been able to outperform previous models in a vast majority of cases. While extremely useful, neural networks still require thousands of training samples which result in a great amount of time and computation power to train a network. Thus the need to expedite this process is one of great importance.

There are two main approaches which can be taken to parallelize the training of a network. The first is to assign sections of the network across threads (See Figure 1). This way each thread handles the computation of the nodes it is responsible for and passes the results along to other nodes which are controlled by different threads.

The second approach is to create one network and instantiate copies of it across threads (See Figure 2). This way each thread receives its own network

and a batch of data to train on. Once all threads have finished working on their batch, their network parameters are averaged with all other threads to produce one master network. In this paper we explore the implementation of the latter.

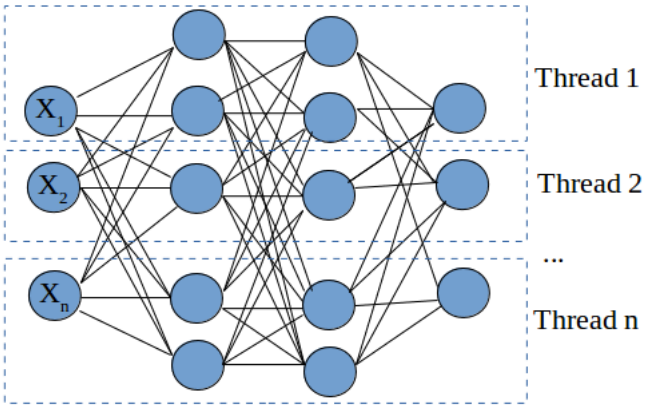

Figure 1: Section parallelization. In this model, groups of nodes in a network are assigned to a thread. The thread handles the computations of each node and passes results along to other nodes in other threads.

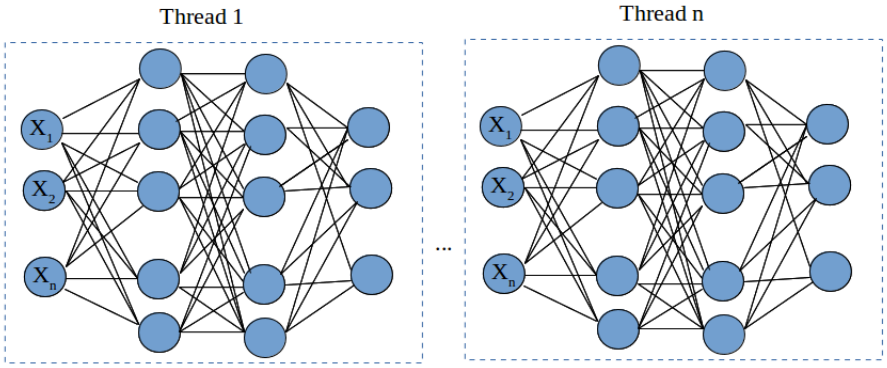

Figure 2: Network parallelization. In this model, each thread receives a copy of the network. Weights of each network are updated independently among threads and averaged at the end.

2 Method and Results

For simplicity we construct a neural network with a single hidden layer. Our dataset is the MNIST dataset of handwritten digits. For our network architecture we have 784 input nodes, corresponding to the twenty eight by twenty eight pixel dimension of the images. Our hidden layer has 128 fully connected nodes and our output layer has ten nodes corresponding to the digits zero through nine. We use 50,000 data points for training and the remaining 10,000 for testing. Our model was implemented in the C++ programming language, using OpenMP to parallelize our model. It is important to note that each time we train a model using this approach each thread receives 50,000/(number of threads). When our model is run we create n threads and assign each thread a network and its share of the training samples. Once the threads have finished we average the weights of each network to create one master network. This is shown by equation 1.

$$W_i = \sum_{i=1}^{n} \frac{w_i}{n} \tag{1}$$

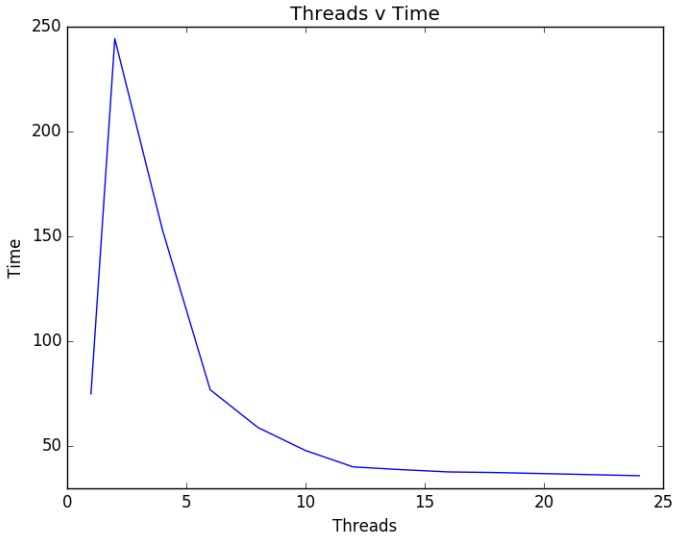

Figure 3: This represents the number of threads used for training and the amount of time, in minutes, the network takes to train.

Figure 3 shows the relationship between the number of threads used and the amount of time spent training. In this graph we see that it is not until we reach six threads or more that the distribution of work among threads overcomes the overhead cost of parallelizing the training.

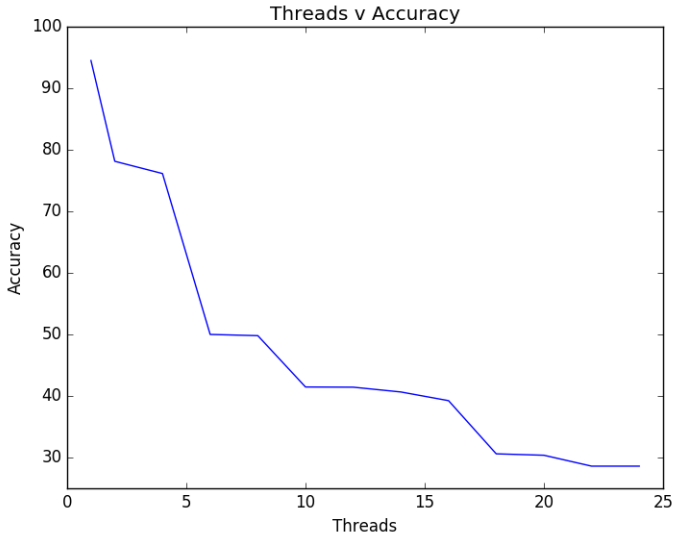

Figure 4: This represents the number of threads used for training and the accuracy of the network, in percentages.

Because parallelizing our model means distributing out chunks of the training data, each threads network received a much smaller amount of data to train on. The effect of this on the accuracy of the model can be in Figure 4. As we increase the number of threads the size of training data each thread receives gets smaller. This means each threads network goes through less parameter updates and is less likely to converge on an optimal solution.

However, an interesting finding is shown in Figure 5. Instead of averaging the weights across all threads we simply performed a sum of the weights across the threads (See Equation 2). When testing the model we observed roughly an increase of roughly ten percent across all tested models. We believe that further work in this area of combining network parameters after parallel training will yield better results. As our method has shown, a simple average across weights will not yield satisfactory results.

$$W_i = \sum_{i=1}^{n} w_i \tag{2}$$

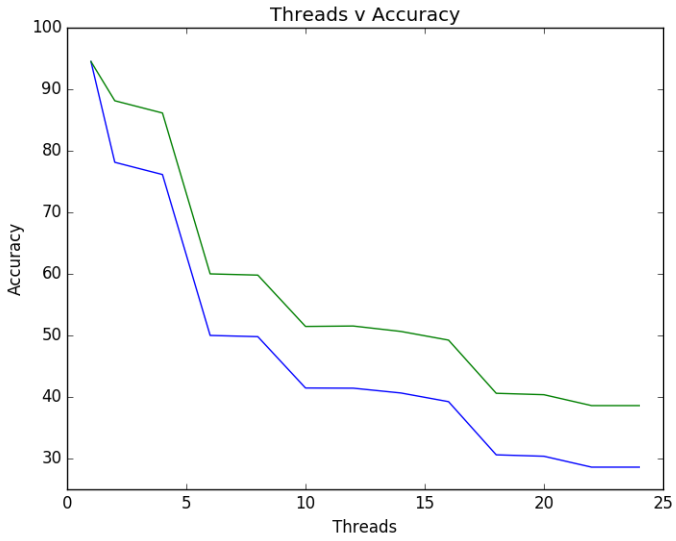

Figure 5: This represents the number of threads used for training and the accuracy of the network, in percentages. The green line represents when the weights are only summed at the end of training, not averaged.

3 Related Works

George Dahl et al. proposed a method to train neural networks on clustered systems [1]. They were able to produce a significant speed-up compared to parallel approaches. They also examined the cost of communication between training nodes.

Mark Pethick et al. showed compared the advantages and disadvantages of the two parallel training strategies, exemplar parallel and node parallel strategies [2]. They found that the experimental results are very consistent with the theoretical costs. They were then able to use the theoretical cost equations to predict which strategy would yield better results given a network size, a dataset size, and a number of processors.

4 Conclusion and Future Work

In this paper we explored training neural networks through data parallelism. We were able to greatly reduce the time required to train a network. However, due to our techniques of combining network parameters we were unable to maintain the original accuracy of the network. In future work we hope to explore better designed methods for combining parameters from each threads network once its training batch has completed.

Acknowledgements

This work was supported by a generous gift from Rene German and Chapman's Machine Learning and Assistive Technology Lab.

References

[1] G. Dahl, A. McAvinney, T. Newhall, *et al.*, "Parallelizing neural network training for cluster systems," in *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pp. 220–225, ACTA Press, 2008.

[2] M. Pethick, M. Liddle, P. Werstein, and Z. Huang, "Parallelization of a backpropagation neural network on a cluster computer," in *International conference on parallel and distributed computing and systems (PDCS 2003)*, 2003.