Parallelizing Neural Networks with OpenMP and CUDA

A Comparison Of Implementations and Optimizations

Luis Guzman
University of Oregon
lguzman@uoregon.edu

Steven Walton
University of Oregon
swalton2@uoregon.edu

October 9, 2020

1 Background

Neural Networks are becoming increasingly popular within computing, with work affecting almost all sectors of computing. Because of this many researchers have devoted a lot of time optimizing these algorithms. Deep Neural Networks have progressed from being academic curiosities to being a cornerstone of many modern technologies. Because of this researchers are constantly trying to optimize for different accelerator technologies and platforms. Even long before accelerators many were trying to parallelize neural networks [2]. While the new resurgence of machine learning partially came from becoming accelerators like GPUs there are still many innovations for CPUs, such as Facebook's new speech-to-text (stt) system that works on CPUs [3] which is aimed at home assistants that can't use large and power hungry accelerators. Algorithms are getting larger and there is continued efforts to apply machine learning techniques to simpler hardware. Thus for the foreseeable future optimizing machine learning algorithms on various platforms will be an active area of research.

2 Speeding Up Neural Architectures

There are many areas that neural architectures can be sped up. Many different calculations can be performed in parallel as they are not dependent on one another. This can be from the feed forward section to the backwards propagation. There are also parallel techniques for running the same model on different machines. There are techniques for data parallelism like batch normalizing [4] that also improve performance, which is arguably a speedup too (speedup to a certain convergence value).

The most computationally intensive part of the neural network is what is referred to backwards propagation. A neural network simply has a set of priors, data is fed in, an answer is given, and then you update the priors based on how wrong the answer is from the true solution. The way to update these priors is through backwards propagation. Fundamentally the main concept is performed by using a gradient descent technique, where we use the gradient to take a step in the direction towards a better solution. Being the most computationally heavy section of the network, this has become the most heavily optimized. Many have used optimized libraries like cuBLAS to accelerate training [7] while others have developed new optimization algorithms [5].

3 Proposed Work

Within this work we propose to test and evaluate different levels of parallelization within a neural network. We will investigate the effects of different acceleration techniques, including OpenMP and CUDA. For testing we will compare our network's performance and speed to that of PyTorch [6], an optimized python library for creating neural networks. We will test against both PyTorch's CPU and GPU implementations. To perform an accurate analysis we will compare how different parts of the network are parallelized or accelerated and thus determine which sections of the network are benefit the most from paralleliza-

tion. We will also profile the code to determine where sections of code need the most speed up. While we do not expect to match the optimization that PyTorch has, we can compare naive implementations as well as optimized libraries against professional software that is commonly used in industry. By comparing both CPU, serial and parallel, and GPU implementations we can also gain a better understanding of how neural networks can be optimized for different architectures. For example, GPUs are great at performing matrix multiplications while CPUs are more generalized compute units.

Much of this work will focus on comparing naive implementations vs optimal solutions, such as the Coppersmith-Winnograd Algorithm [1]. Matrix (and tensor) multiplication is the backstone of many parts of these neural networks, and comparing naive implementations to optimized algorithms and optimized software packages gives insight into how the optimized packages work. By profiling the code we can also get a better insight into which areas need to be improved the most and we can quantify how much these types of algorithms affect neural networks.

The next most important aspect if to parallelize the gradient descent methods. There has been work in parallelizing Stochastic Gradient Descent (SGD) [8], as well as other algorithms like Adam [5]. One simple way to parallelize all of these algorithms is by performing batching, which we will explore in this work.

Furthermore, in this work we will do continued research, specifically into the optimizations and parallelized methods that PyTorch does. At this point we do not know much about these algorithms but expect that expertise and interest to grow as the project continues.

4 Deliverables

In this work we shall produce source code that performs different optimizations techniques in a serialized form, with OpenMP, with CUDA, with PyTorch's CPU implementation, as well as with Pytorch's GPU implementation. We shall also provide graphs that illustrate these differences for different network sizes. For this work we will also summarize our findings as well as further research and deliver this report by the end of the term.

References

[1] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. Computational algebraic complexity editorial.

[2] A. I. Cristea and T. Okamoto. A parallelization method for neural networks with weak connection design. In C. Polychronopoulos, K. Joe, K. Araki, and M. Amamiya, editors, *High Performance Computing*, pages 397–404, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[3] Q. He, T. Koehler, A. D'Avirro, and C. Gupta. A highly efficient, real-time text-to-speech system deployed on cpus, May 2020.

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[7] X. Sierra-Canto, F. Madera-Ramirez, and V. Uc-Cetina. Parallel training of a back-propagation neural network using cuda. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 307–312, 2010.

[8] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, page 2595–2603, Red Hook, NY, USA, 2010. Curran Associates Inc.