An OS Service for Efficient Data Communication in Large-Scale Machine Learning Applications

Anders Øland
anderso@cs.cmu.edu

Timothy Loffredo
timloff@gmail.com

Yimeng Zhang
yimengzh@cmu.edu

October 23, 2014

Abstract

Communication has become one of the most significant bottlenecks for modern large-scale machine learning applications, e.g. deep neural network, whether using a compute cluster with thousands of nodes, or a single multi-GPU machine. However, there's little OS-level support for fast and efficient communication prevalent in these scenarios. In this project, we plan to develop an OS service that applies quantization and other (possibly lossy) compression techniques to increase throughput of communication yet doesn't hurt the performance of learning. We'd like to explore appropriate compression schemes for different kinds of data, as well as designing an on-line algorithm to determine the best compression parameters dynamically. In our initial experiments with neural networks [1], we were able to reduce the communication overhead by nearly an order of magnitude, without slowing down the convergence, or hurting the generalization of the model.

1 Introduction

Large-scale distributed machine learning is becoming increasingly more important in modern computing. The ability to learn from massive amounts of data is extremely useful in both research and industry applications. In particular, deep learning has received an enormous amount of interest, and indeed companies like Google, Facebook, Amazon, and Baidu have all been investing heavily in this line of research. This involves processing terabytes of data, and training models with *billions* of parameters; even on the biggest and fastest computers in the world, this can take days, and even weeks — for one single model. A lot of attention has been given to the problem of improving the learning algorithms, in order to get good scaling. However, no published work (literally) exists on specifically tackling the primary bottleneck, which is *communication*.

In this project, we will investigate the effectiveness and usefulness of different methods for quantization and compression of the parameters of neural networks (NN) in a distributed setting. In this context, data (and possibly the model parameters) are split across different workers. Model parameters and intermediate products in learning need to be communicated across workers during learning. The end goal is to provide an OS service that can handle the communication involved in distributed deep learning effectively. Albeit, in this work, we will not actually focus on *implementing* said service, but rather we will propose *how* it could be implemented. That is, what quantization and compression algorithms are best suited for different types of data (e.g. weights or error gradients

in a NN), and different contexts (e.g. CPUs vs. GPUs, or Ethernet vs. InfiniBand), and what is the data that needs to be transmitted (e.g. Huffman codebooks, or quantization meta-data).

2 Methodology

2.1 Resources

In order to evaluate the effectiveness of the various versions of the service we propose, we will implement it in MATLAB. That way, this project can focus on experimenting with different quantization and compression methods rather than thrashing on implementation details. This project also requires a testing environment where we can measure the communication and performance metrics of a running system using our service. Anders, a member of our team, privately owns two machines with nine NVIDIA GPUs connected by a dual-channel InfiniBand interconnect. We plan on using these machines as our testbed.

2.2 Evaluation

In choosing a compression scheme, we must consider many factors including cost (complexity and execution time), memory usage, effectiveness (compression ratio), and impact on learning (the generalization error of the model). Obviously, the cost of encoding, sending, and decoding the data must be less than simply sending the data as is. Similarly, cutting down the cost of communication is meaningless if it affects the learning negatively, such that either the generalization error increases, or it takes considerably more iterations to converge (so that nothing is gained with respect to the total execution time).

Thus, we will evaluate our proposed choices of quantization and compression algorithms, for various relevant scenarios (use-cases), with respect to:

- Compression ratio
- Impact on learning
- Algorithmic complexity
- Memory usage

3 Goals

The 75% goal for this project (which we expect to achieve) is to evaluate three different methods for quantization with respect the measures mentioned in 2.2. For this part we will only use Huffman coding, standard NNs, and one standard dataset (remember, running these experiments is very time-consuming).

The 100% goal for this project (which is probably achievable) is to include results for convolutional neural networks, compare Huffman coding with gzip, and add an extra dataset. Furthermore, we will outline a possible communication protocol, and discuss (based on our results) what methods are best suited for various interesting real-life scenarios.

The 125% goal for this project (which is a stretch goal) is to add completely autonomous tuning of the transport parameters. That is, without knowing anything (or at least not knowing everything in advance) about the data, how can we choose the best quantization and compression scheme.

4 Final Paper Plan

In our final paper, we will first introduce the problem of data compression in neural networks, and discuss background and related work. Next, we will briefly outline the experimental setup, including what machines and environment were used. The gist of the paper will be concerned with describing and evaluating different methods for quantization and compression. We will provide experimental results, and conclude with a discussion about what schemes worked well and which ones did not.

Our early experiments will focus on trying out different compression and quantization schemes in order to learn their performance characteristics. Huffman codes with dynamically determined codebooks that are sent in addition to the data is likely to be the first experiment. We will run a neural network problem programmed in MATLAB on a single CPU or GPU, and test our Huffman compression scheme by sending weight data at every epoch to a "dummy" proxy process. We will measure the compression ratio, and the model accuracy.

Other possible experiments include Huffman codes with statically determined codebooks that are not sent with the data but are known at both ends, Lloyd's algorithm for quantization, and naive bitmask quantization. The results of these experiments will tell us which compression and quantization schemes are most effective for neural network-type data. It might be that a single algorithm that rises to the top as the obvious winner; or more likely, there will be several algorithms that form a reasonable set of trade-offs.

We will also try out our compression algorithms on different network architectures (e.g. convolutional neural networks) with different datasets. The results from these experiments will help us know the effectiveness and generality of different compression schemes for various learning scenarios.

Questions to be answered by our experiments:

- Is compression beneficial for a wide variety of (deep) learning applications?
- What are the appropriate compression schemes for applications with different characteristics?
- What is the overall reduction of communication overhead by compression? Is compression worthwhile considering the trade-off between the complexity of introducing it and its benefits for communication?

References

[1] Anders Øland and Bhiksha Raj. Reducing Communication Overhead in Distributed Learning by an Order of Magnitude (Almost). submitted to ICASSP 2015.