GPU Profiling For Tensorflow Performance

Alexander Stein and Michael Alvarino

May 6, 2018

Abstract

Our final results, source code, and documents can all be found as a series of blog posts detailing our experiences at https://aistein.github.io/dlprof/. What follows is a high level description of our project which can also be found on the homepage to our blog posts.

1 Introduction

Historically deep neural networks were treated as interesting, but mostly theoretical, experiments due to the computational power needed to learn a model with so many parameters. Simpler models were proposed and tested, but nothing on the scale of what we have today. It is now common to find networks with orders of magnitude more trainable parameters than the size of the input space. Without the increase in computational power achieved by GPUs deep neural network research may have continued to lag. Now that GPUs and even some custom processing units (TPUs) have been optimized for highly concurrent computations how can we continue to make our deep neural networks faster?

This series of blog posts proposes ways to best utilize modern deep learning frameworks, specifically Tensorflow. It is the culmination of a final project for the course EE6040 at Columbia University by aistein and michaelAlvarino exploring common inefficiencies, easy gains, new features, and low level analysis that can help improve neural network training and inference times. Our intent was to produce a guide-book detailing our experience using (and even compiling!) Tensorflow to build, debug, and improve neural networks.

2 Background

The research of optimizing GPU computations for linear algebra, and for creating efficient learning algorithms is very rich. Unfortunately, very little research has been published with the practical goal of using a specific framework like Tensorflow and its ecosystem of tools to improve neural network performance. We were able to find some works which utilize low level profiling tools to optimze high level framework code, but unfortunately the examples we found provided neither source code nor clear guidelines for how best to use the frameworks available. Rather, they provided small improvements that seemed unrealistic when considering professional development standards. The nearest examples are those provided in Tensorflows performance page. From this page we took several select examples that we thought were especially useful or attainable for engineers with our resources and tested them on some ”real-world” use cases.

3 Our Approach

I quote real-world because we are not creating profit generating products or core business products, but rather taking networks that we are familiar with and applying the tools given to demonstrate improvements. As previously stated, our goals were to provide useful and achievable examples of optimizations and best practices for Tensorflow.

4 Experiments

One of the networks we use (specifically we use it to demonstrate input pipeline optimizations, the tfrecords api, and multi-gpu training) is based on the network described in Joint Deep Modeling of Users and Items Using Reviews for Recommendation. The original code base was part of a previous project written using a combination of keras for network modeling and python for input pipelines and is available here. The original convolutional model took on the order of 10s of minutes to train per epoch, but with our improvements we were able to bring that time down to 26 seconds in some cases.

In addition to this custom model, we use some pre-defined Tensorflow models often pulled and modified from the Tensorflow Examples repository. For example, when demonstrating the effect of data formats we turned to Tensorflow's mnist_deep.py example for a pre-existing convolutional network that could be optimized.