CSCI-503 Project Proposal

Palash Goyal, 6803256849
Nitin Kamra, 4873546495
Vasileios Zois, 7489325758
Sungyong Seo, 7897906462

April 4, 2016

1 Background

Artificial neural networks are a powerful machine learning tools used in many applications including but not limited to search engines, spam and fraud detection, image classification, diagnostic medicine applications and stock market prediction.

Prior to application, neural networks undergo a training phase which is known to be computationally intensive. This is primarily because the prevailing neural network architectures are implemented using several hidden layers, with each one consisting of thousands to millions of neurons in order to generalize well on diverse inputs. In this case, the resulting number of parameters that need to be trained are in the order of millions. Furthermore, achieving high accuracy, requires considering a large number of training examples (usually in the order of millions). For this reason training a neural network can be also considered a data intensive operation. This calls for efforts to parallelize the training process on multi-core and many-core architectures while emphasizing at utilizing the maximum available bandwidth.

Currently Gradient Descent is the most commonly used optimization algorithm used to train neural networks in supervised settings. It is implemented in a layerwise-recursive fashion which is termed Backpropagation in the context of neural networks.

2 Problem Description

In this project, we consider the task of training feedforward neural networks for classification tasks using the conventional backpropagation algorithm. Feedforward neural networks act as function approximators in such tasks.

More formally, given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1:N}$ with data points $x_i \in \mathbb{R}^M$ and scalar labels $y_i \in \mathbb{R}$, the classification task involves making an accurate label prediction $\hat{y}$ on a previously unseen data point x. We approximate the label as a function of the datapoint using a feedforward neural network with parameters $\theta = \{\theta_k\}_{k=1:K}$ as follows: $y \approx \hat{y} = f(x; \theta)$. The neural network learns

the function f from the training data $\mathcal{D}$ by tuning its weights (θ) to minimize a pre-specified loss function, for instance, the Mean-Squared Error loss: $\mathcal{L}_{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i;\theta))^2$.

This minimization is generally carried out using gradient descent, which in its most naive form is implemented as follows:

1. Initialize all weights (θ) randomly with small values close to 0.
2. Repeat until convergence {

$$\theta_k := \theta_k - \alpha\frac{\partial\mathcal{L}_{MSE}}{\partial\theta_k} \qquad \forall k \in \{1, 2, ..., K\}$$

   }

Note that the derivative of the loss function generally takes the following form:

$$\frac{\partial\mathcal{L}_{MSE}}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i;\theta))\frac{\partial f(x_i;\theta)}{\partial\theta_k}$$

Since this requires a summation over all the training examples, the gradient computation is the biggest bottleneck for many supervised learning tasks on huge data sets.

On the other hand, having the gradient as a sum of partial gradients with respect to individual training examples opens up the possibility of parallelizing the gradient computation efficiently. This can be done by distributing training examples across various processors/machines and letting them compute a partial gradient over their own training examples and then summing up these partial gradients to get the actual gradient. A good approximation of this process can be obtained by using a mini-batch of training examples per iteration, instead of using the full dataset.

3 Proposed Work

Given the recent interest in neural network training and the complexity associated with it, we wish to study techniques that enable parallel training on multi-core and many-core architectures. Our plan is to implement the parallelized training of neural networks using mini-batch gradient descent. We are aiming to complete the following tasks by the end of the semester:

- Serial implementation of Mini-batch Gradient Descent(MGD) in C++
- Parallel implementation of MGD in C++ using OpenMP
- Comparison between serial MGD, parallel MGD with OpenMP
- Parallel MGD using Cuda-C on a GPU
- Comparison between serial MGD, parallel MGD (Cuda)

To test the above implementations, we will use the following datasets:

- **MNIST** - This is a database of handwritten digits that is commonly used for training various image processing systems. It has 60,000 images each of size 28×28 pixels. The task is to classify the digit in the image.

- **KDD Cup 1999** - This is a data set designed for intrusion detection (distinguishing bad network connections from good ones). It has about 4 million data points each of which has 42 features.

4 Further extensions (Optional)

If time permits, we shall also implement some extra steps for a better comparison. Specifically we will implement MGD in Theano, which is a Deep Learning library with a Python interface and do a performance comparison with our code:

- Parallel implementation of MGD in Theano
- Parallel MGD in Theano using a GPU
- Comparison between our code and Theano implementations of MGD (both CPU and GPU)

Note that since Theano is a state of the art Deep Learning library which performs many other optimizations apart from just parallelizing Neural Network training, we do not expect our code to run faster than Theano. Instead the comparison will be done to see how close we can get to the current state of the art by simply parallelizing the Neural Network training.