Ethan Hyde, Eric Edwards

CS431

16 October 2023

Project Proposal

Summary:

For our project, we will be implementing a 3 layer neural network, taking input from the MNIST dataset and training the model to recognize handwriting patterns over time. First, we will be creating a serial implementation of a neural network, processing images iteratively with no parallelization, while using AdaGrad/stochastic gradient descent algorithms. Next, we will use the Math Kernel Library (MKL) from Intel, and compare the learning rate of our model with the MKL implementation to the learning rate from gradient descent algorithm implementation. Next, using OpenMP, we will parallelize these approaches in hope to increase the learning rate while taking advantage of the processing capabilities from the Talapas computer.

Background:

In principle, a neural network can be seen as an adjustable vector function. By training the neural network, we allow it to adjust the weights and biases of the function itself, which would result in a more intelligent model. To do this, we would start by feeding an image from the MNIST dataset, which would then be converted into a 2D binary array. Each image in the dataset has an expected output value, so the learning is supervised in a way. We can see how the outcome of our neural network differs from the expected outcome by deriving a *Cost Function*. This function would tell us how correct or incorrect our network was on a particular sample, and provide us with some insight on how we might be able to change the neural network to improve it. An example of a cost function we might use is the Mean Squared Error, which is

$C = \frac{1}{N}\sum_{i=1}^{N}(y_i - exp_i)^2$ which takes the average over the results and the difference between the estimate of y and the actual value of y. The square allows us to make the function a quadratic.

The first optimization algorithm we will be using is called AdaGrad. This is specifically designed to handle sparse data machine learning. It works by adapting the learning rate of all the network's parameters by scaling them interactively and proportional to the square root of the sum of all the historical squared values. Each time the algorithm looks at the slope of the gradient, it asks how steep it is, then squares that answer. If the accumulated square gradient value is very large, then its square root will be large, meaning there's a smaller learning rate for that parameter. If that number is small, then there will be a quicker learning rate for that parameter. Adagrad first works by initializing all the model parameters, and setting a learning rate, denoted as η . We then will initialize a gradient accumulation vector G which keeps track of the sum of the squared gradients for each vector. Typically, people will compute the gradient by using backpropagation, and updating the parameters accordingly.

The next algorithm we are using will be stochastic gradient descent. This is more efficient than the gradient descent algorithm, as it picks one example (known as a mini-batch) at a time from the dataset and updates the gradient based on this one example. With each epoch over the MNIST dataset, we would shuffle the images and initialize the weights and biases of the neural network. For each mini-batch we would do a forward pass, computing the output and input the mini-batch into the neural network. The cost function would then compute the loss and see how accurate we were, then we will use backpropagation to compute the gradient of the loss function for each weight and bias. Eventually, the model will become more intelligent and after a certain number of epochs, we will be able to evaluate the model and make changes as needed.

Challenge:

As MNIST is a fairly simple dataset, it might be hard to find a substantial difference between the different optimization algorithms. There's many models out there that are able to achieve an accuracy rate greater than 95%. However, the simplicity of this dataset might assist us with writing a neural network for the first time, as using a complex dataset may prove to be more challenging than anticipated. Another problem we may face could be reproducibility of results. For example, we may get a high accuracy rating on one training cycle, then when we shuffle the dataset, a completely different accuracy rating. It's important that the model gains an intelligence rating that is consistent across all tests.

Resources:

As stated earlier, we will be utilizing:

1. The Talapas supercomputer - the AdaGrad algorithm shines with high computational resources, so this will be a very useful tool for implementing a fast and precise neural network.
2. The Math Kernel Library by Intel, we'll use this for optimizing the matrix multiplication optimizations as a way to compare to our from-scratch parallelization.
3. We will utilize our neural network research from various learning sources to build our from-scratch neural network framework.
4. We will implement our optimizations based on academic pseudocode, i.e. https://optimization.cbe.cornell.edu/index.php?title=AdaGrad
5. We will use the OpenMP library to parallelize the neural network computations.

Goals:

Our goal in this project is to create a fast, accurate neural network from scratch using parallelization techniques to speed up computations. An initial goal will be to be able to create parallelization optimization that is at least 75% as efficient as the MKL optimizations. We want to determine whether AdaGrad or Stochastic Gradient Descent performs better both with high computation power and parallelization implemented.

Timeline:

Week 1 (10/16-10/20):

- Neural network research, optimization method research

Week 2 (10/23-10/27):

- Build serial neural network framework

Week 3 (10/30-11/3):

- Implement optimization algorithms

Week 4 (11/6-11/10):

- Parallelize code using OpenMP

Week 5 (11/13-11/17):

- Parallelize code using MKL

Week 6 (11/20-11/24):

- Testing and report

Week 7 (11/27-12/1):

- Presentation of project