Neural Network Parallelization

Mason Olsen, Callista West, and Patrick Thomasma

University of Oregon

October 11, 2021

Abstract

This is a project proposal by MPC Solutions on Parallelization of Neural Network training. In this proposal, there will be our motivation behind choosing this topic, areas and problems to be explored, possible directions of investigation, and our expected results.

I. Motivation

With our group having different programming experiences, all members expressed interest in learning more about parallel programming within machine learning. After focusing on machine learning, we found curiosity in neural networks. Neural networks, named after the process that biological neurons in the human brain use, are a subset of machine learning and a major part of deep learning algorithms. Neural networks, created with node layers, contain three types of layers: input layer, hidden layers, and an output layer. The nodes are connected to one another, to allow sending data to a deep layer of the network when the threshold value of any node is exceeded.

Additionally, there are numerous types of neural networks but feed forward neural networks are the most basic and allow extra layers without too much consideration of computation time. Neural networks are also useful tools within artificial intelligence. However, feed forwarding has disadvantages such as its inability to be used with sequential data and difficulties with image data. Some other neural network types include convolution neural networks (CNN), recurrent neural networks, and transformers. CNNs, while seen as automatic feature extractors from images, have their own downfalls such as object detection. This project will allow our group to further investigate the other types of neural networks and find a dataset with the correct neural network type to parallelize.

Accuracy for neural networks takes time, requiring training data to improve speed and accuracy in order to sort data at a high velocity. Each individual node is made up of input data, weights, a threshold, and an output. No weight is assigned to a node until the input layer has been determined. Each node's weight helps determine significance, where the larger weights carry more importance to the output. Using specific formulas to sum the total weights, the output will activate the node and pass the data to the next layer in the network if the threshold were to be exceeded.

With the preliminary research our group has done, neural networks alone have caught our interest. Aspects of the formulas involved in creating weights, thresholds, and outputs have helped us better understand neural networks,

but have left us to further explore neural networks with parallel programming. This project topic serves as an opportunity to decrease the computing power used by neural networks. Efficient neural networks are used in daily human life such as Google's search algorithm, one of the most well known neural networks. Through further investigation, research, and experimenting, MPC Solutions looks forward to exploring methods to parallelize a neural network.

II. Areas and Problems to be Explored

Neural networks are a major component of the internet that takes into account the activities that people will do in their everyday lives. A prime example of neural networks is google's search engine and how it will extrapolate data gathered through a users activities and be able to try and predict the users behavior in their searching patterns. Keeping this in mind it's not surprising to learn that Neural networking takes up a lot of computing power and a common way in order to utilize this massive computing power is to distribute computations between different processors and perform those computations simultaneously. Methods used are Model parallelism and data parallelism which is the most used technique for parallelization and its scale is corresponded to the batch size which is the number of training examples used to compute an update to the neural network. This type of parallelization has limits however since while the parallelization is satisfactory for most workloads with today's tech any workload that moves beyond that will not benefit from batch sizes that exceed 1024. Usually in order to more effectively utilize additional data parallelism the simple way to produce a speedup is to increase the batch size but with more advanced and state of the art hardware the benefits start to diminish, some optimization algorithms may be able to extend the scaling regime across many different models. The most interesting field to investigate would be how to find optimization algorithms so hardware can benefit with a bigger batch size and how to effectively use workloads that would be able to use the maximum amount of computing power possible with the available hardware, also to find a consistency between the width and the depth of the network even though many will have different notions of what the width and the depth could be.

III. Possible Directions of Investigation

The first thing we will do is select a dataset from Kaggle. Kaggle, a website with free access to a repository of community data and code, provides many datasets used for machine learning. With this data, the next direction for our project is to first create a neural network without any HPC techniques. This model will serve as a baseline model to compare different models that do have HPC techniques. We will be able to use the baseline model to draw conclusions when evaluating our work throughout and at the end of our project.

The algorithm we will use for training the neural network will most likely be a form of gradient descent. Gradient descent is a first-order iterative optimization algorithm. Gradient descent is used in machine learning to find the values of a function's parameters. With this algorithm, there are two areas of parallelization that can be explored. The first area

to explore is parallelizing the computation of minimizing the loss function in order to get the most accurate values for the weights and bias of each layer. This computation requires taking the derivative of the loss function with respect to each weight, plugging in those weights into their respective loss function with the parameters that correspond to those weights from the data sample, calculating the step size, then using that step size to calculate the new weights. This process is repeated until the change between the old weights and the new weights has reached an optimal minimum. For data samples with many parameters, this will require many of the same computation for all the different parameters. Thus it should be able to be parallelized. For large amounts of data samples, this will require many passes through the data set in order for the gradient descent to arrive at optimal weights and biases. The most likely way to increase the efficiency of these computations is to divide them up amongst multiple processes, having those processes calculate their respective gradients, and then aggregating those gradients across the multiple processes. This act of distributing the gradient calculations across multiple processes is called data parallelization which will be a large part in our project.

The second area of parallelization we will explore is how to parallelize gradient descent through the multiple layers. Neural networks use feedforwarding and backpropagation. Feedforwarding is sending a data sample through the neural network to get an estimate of its output. The error of the output is then backpropagated to update the weights and biases accordingly. In other words, the feedforwarding is used to create a prediction and the backpropagation is used to tune the weights and biases of each layer through the network to make that prediction more accurate. We have found that there is such thing as Parallel Back Propagation. This involves parallelizing the backpropagation that occurs in gradient descent through the multiple layers. Whether or not this improves speed or efficiency, will be researched in our project.

Finally, we will explore how to organize the data such that the constant memory fetches needed to train a neural network are optimized. Neural networks require large amounts of data to properly train the weights and biases of each layer, thus the organization of this data is crucial to efficient training. Some areas to explore in data parallelism for neural networks are how to efficiently distribute training samples across multiple processes and then combining the computed gradients. We will also want to explore how to organize the training data in memory so that either the number of fetches are less or the fetches are executed quicker.

IV. Expected Results

When we are able to implement our version of parallelization of the neural net we are hoping that the relationship between batch size and speedup will be consistent and that the benefits of the parallelization will benefit any kind of machine. Also, we will want the parallelization to be able to strongly scale with the neural net problem and to see if our optimizer will be able to perfectly scale with the batch size on any machine. A common problem with neural net parallelism is that when batch size is increased, the benefits of having a state of the art machine will diminish. Hopefully, the optimizer that we will implement can try to alleviate this problem for a bit and be able to speed up the processes

of the neural net for a good result. Our exact method for measuring speed and execution time is undecided at the moment, but with experimenting timing options, we will find the best method to measure speedups.