Deep Neural Network with Raspberry Pies

Triton

***Abstract*—(Neeed to add this)**

I. Introduction

In the last few years, the deep learning field has seen tremendous advancement that enables the creation of many life-changing technologies including autonomous driving, self-taught AI, and pattern recognition. This progress is only possible due to the availability of high-performance GPUs which excel at solving multiple floating pointing problems simultaneously. According to PassMark Software, a crowd-sourced GPU benchmark database, the current flagship GPU is approximately forty thousand timers faster than the flagship GPU from 20 years ago (Citation). Unfortunately, as the interest in the deep learning field increases, so did the demand for high-performance flagship GPU and as such, the price of flagship GPU has increased exponentially. Today, the high-end flagship GPU is 4 times as expensive as the flagship GPU from 20 years ago (Citation) and this price increase makes it difficult for average consumers to obtain necessary hardware for experimenting with the deep neural network at home. As such, this paper is intended to explore the possibility of using a distributed system of relatively cheap hardware to solve a deep learning problem. This paper will start by covering the history of deep learning including its initial hurdles, its rise, and the current obstacles. Then, the paper provides an overview of how deep learning works such as forward propagation and backward propagation and techniques for parallelizing deep learning problems. Furthermore, this paper will discuss the implementation of one of the parallelization techniques as well as its setup, advantages, and limitations. Finally, the last part of the paper will contain the authors' thoughts, recommendations, and discussions on future works.

II. History of Deep Learning

Since its initial conception in 1967 by Alexey Ivakhnenko, two major hurdles of applying deep neural networks in solving real-life problems are hardware limitations and data limitations. To start with, a majority of hardware designed at that time was focused on single-core performance because that is what a majority of applications required (Citation). Thus, there was no incentive for developing hardware that has more than a single core. Unfortunately, this single core focus hardware is terrible for solving neural network problems because each node in a layer has to be calculated linearly before it can proceed to the next layer. Second, since the internet wasn't widespread at the time, any dataset that needed to train a neural network would need to be collected manually. This process is prohibitively expensive and thus, good datasets are hard to come by (Citation). This hardware restriction and data constraint stifled any advancement in the field for many years to come.

The rise of deep learning, in the last few years, is the result of the widespread availability of high-performance GPU which was primarily designed for rendering complex and dynamic scenes in real-time (Citation). Interestingly, rendering dynamic scenes and training neural networks require similar forms of hardware solutions which compose of many fast floating-point calculating cores. Furthermore, with the rise of the internet and the way people interact with it, it has become easier and cheaper than ever to collect a massive amount of datasets (Citation). These two factors fuel the deep learning revolution that occurs today and neural networks are being adapted to solve problems that were considered too complicated for a computer to solve.

However, as the deep learning field continues to make massive progress, a new problem is rearing its heads and threatening the potential growth of the field. Today, the general conscience on how to improve a neural network is to make it wider and deeper which, in theory, should increase the network representative power. And, as the network gets more complications, it also requires more datasets to train properly. As a result, the current state of the art neural networks is composed of hundreds of nodes and layers and require millions of datasets to train. This exponential increase in the volume of information means that neural networks of today have outgrown the information capacity of most modern hardware. To simply put it, neural networks are getting too large for a majority of GPU to represent and its training datasets are too big for most computers to store in their memories (Citation). To address the hardware restriction described above, many researchers have turned to the distributed system as a possible solution. The common setup for such a system is a cluster of one to many computers where each computer has as many GPUs as the computer's PCI-E bandwidth allows which is around two to four. Even though such a setup has shown promising results, its cost is undoubtedly high and as such, it will be a great barrier of entry for most consumers which, ultimately, could harm the future growth of deep learning as a field.

III. Deep Learning Basic

At its fundamental idea, a deep neural network is a universal approximator which means that it can reproduce any function by just looking at said function inputs and outputs. Deep neural networks can accomplish this because of its structure which consists of multiple layers of neurons, weights, and biases. (Citation) For a neural network to approximate any function, it first needs to go through a training process where its weights need to be adjusted based on how accurate its predicted output

is compared to the true outputs. This training process consists of two distinct phases: forward propagation and backward propagation.

To start with, the main purpose of the forward propagation phase is to see what kind of outputs are produced by a network given some inputs. This process starts by feeding inputs into the first layer and forwards outputs from such layers to the next layers. Inside the layer, neurons are responsible for taking outputs of previous layer neurons and multiplying them with their respective connection weights. Then, the neuron accumulates the resulting values and passes the sum into some kind of activation function to produce the output for this neuron. It is worth noting that the activation function is necessary for deep neural networks in order to introduce some form of nonlinearity into the network which will enable the network to approximate nonlinear function. (Citation). This process of passing outputs from one layer to the next is repeated until the final output is achieved.

Initially, predicted outputs produced by the network will be wildly inaccurate when compared to the expected outputs because weights are randomly generated. However, those inaccuracies or errors do offer a hint on how to adjust the network's weights to improve its performance. The main objective of the backward propagation phase is to adjust the network's weights such that its errors are minimized and it accomplishes this with the help of the gradient descent algorithm. First off, predicted outputs and expected outputs are compared to produce an error value called loss. Then, the loss value is propagated backward throughout the network where, at each layer, gradient, or amount in which each weight contributed to the loss value gets calculated. Finally, each weight gets adjusted with their respective gradient and the entire process repeats itself until the loss is minimized (Citation).

At first glance, once would assume that the process of forward propagation and backward propagation are repeated for each pair of input and output and that would produce a well-trained network but it is not necessarily the case.(Citation) The major problem that arises from adjusting weights for every dataset is the instability that would be introduced into the network which increases the chance of the network to converge on a suboptimal solution. Thus, it is recommended to train a network in batches of datasets which entails performing forward propagation steps for multiple datasets, average their losses, and use the average loss to perform backward propagation.

IV. Parallelization of Deep Learning

The consensus on how to deal with neural networks requiring more resources than what modern hardware can provide is through the use of distributed systems. However, in order for neural networks to be able to run on a distributed system, its structure needs to be changed. Currently, there are two types of parallelization that are recommended for a neural network and they are model parallelization and data parallelization.

To start with, as network networks get more complicated, its hardware requirement is also increased. Thus, the goal of model parallelization is to address the scenario where networks are too deep or twice to be represented in a single GPU. This methodology calls for the slicing of networks into smaller groups of layers where each layer chuck can be loaded onto a separate GPU. Then, the training process occurs similarly to how a network gets trained on a single GPU but instead of passing values from one layer to another, values are passed from one GPU to another GPU. (Citation)

Unfortunately, as neural network complexity increases, the amount of datasets required to train such a network also increases exponentially which usually exceeds the capability of most computers to hold. As such, data parallelization aims to address this issue by splitting datasets into small chunks and distributing them onto multiple nodes. Then, each node processes their chunk of datasets independently and forward the average loss to either a central node for centralized architecture or the next node for map-reduce architecture. Finally, the accumulated loss gets averaged out based on the number of nodes in the cluster and the averaged loss gets propagated back onto all nodes where their weights get adjusted by their respective gradient. (Citation)

V. Setup

(-Hardwares and Softwares used-)

VI. Result

(-Discuss the result including scalabiity, cost, problem, ..etc.-)

VII. Recommandations

(-Discuss what you would do for next research such as getting different hardware or software.-)