Sam Cantor
June 12th 2019

1. Introduction/Abstract

The goal of my individual research at DCP is to establish a system for distributing machine/deep learning tasks on the platform, with a focus on developing a program to train convolutional neural networks on the platform for image/video processing. As datasets and machine learning models increase in size and complexity, access to computing power to train these models becomes vital. In theory, the ability to use DCP's platform for machine learning would drastically reduce the amount of computing time for training. The decision to research CNNs and image processing was because of its relevance in the field of machine learning, and a general framework could be constructed based off CNN training to accommodate for many classes of neural networks First, a fundamental understanding of how CNNs work on a low level must be established. This will allow me to reconstruct a neural network in javascript and be able to optimize it for DCL. Then, research must be done on previous methods to distribute or parallelize algorithms for deep learning. Finally, testing will be used to determine if the implementation of the program in DCL will speed up the training of models.

2. Convolutional Neural Networks

CNNs, or ConvNets, are one of the most relevant and applicable classes of deep learning neural networks used today, and are suitable for image processing and classification. It takes in images as inputs, and assigns biases and weights to image features in order to differentiate them. As mentioned by Sumit Saha: "The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction".

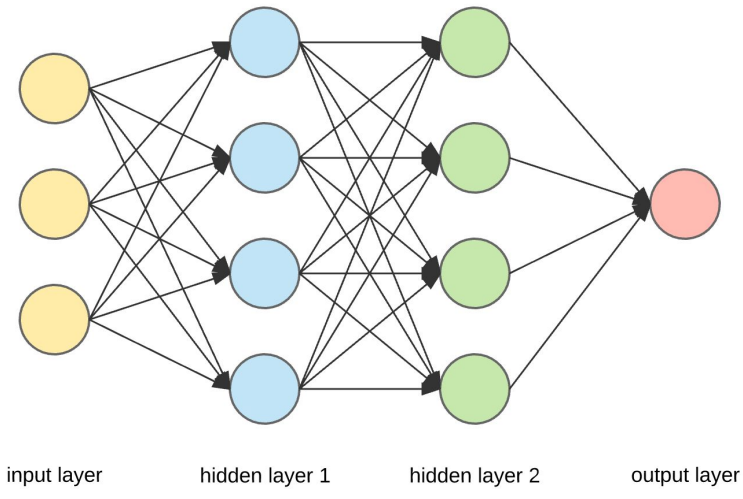

Figure 1: Simple Neural Network

2.1 Forward Propagation

A neural network at a low level takes the values from input neurons, or nodes, and sends them through a network of hidden nodes by applying functions to them, until they reach the final output layer. The functions also include individual weights and biases that will be updated to improve the accuracy of the model. These are also known as kernels. Fundamentally, convolutional neural networks operate in the same manner. The image is sent through multiple filters and fully connected networks until output predictions can be made. The first step in training a CNN is to initialize random weight and bias values to each filter. The image will then be split up into three color channels and sent through the feature extraction part of the network. Feature extraction consists of multiple layers, including convolution, ReLU, and pooling. The convolution and ReLU layers apply filters to the image to create a filter map and replace negative values with zero. Pooling reduces the dimensionality of the filter map while retaining the most important information.

After multiple iterations of these layers are applied, the data gets sent through a fully connected network, simplified by **Figure 1**. At the last layer, output predictions are determined for each class (i.e dog, cat, tree). For example, let's say the image was a tree. Then, the target values would be (0,0,1). Possible output values could be (0.2, 0.4, 0.4). This data will then be used in backpropagation to adjust the biases and weights in order to achieve a more accurate result.

2.2 Backpropagation

Using the output predictions generated in the forward propagation phase of the network, the filter weights and biases will be adjusted in an attempt to improve the accuracy of the model. First, the error of the output predictions with respect to the target values must be calculated. Ujjwal Karn states the error can be calculated using the following equation:

$$Total\ Error = \sum \tfrac{1}{2}(target\ probability - output\ probability)^2$$

The total error can be used to calculate the *Gradient* with respect to each weight. *Gradient Descent* is then used to update the filter weights and biases to improve the model for future inputs. Gradient Descent represents the loss function of the algorithm, and the goal is to minimize this.

2.2.1 Stochastic Gradient Descent

According to a paper by Hedge and Usmani (2016), one of the most computationally expensive steps of training a neural network is computing the gradient of loss for the images in the data-set with respect to all the parameters in the model. They claim that this is why it is necessary to use stochastic gradient descent. As noted in the paper, this method is used for every subset of the database until all images have been used up, also known as an epoch. Previous methods to parallelize and distribute SGD will be evaluated later on.

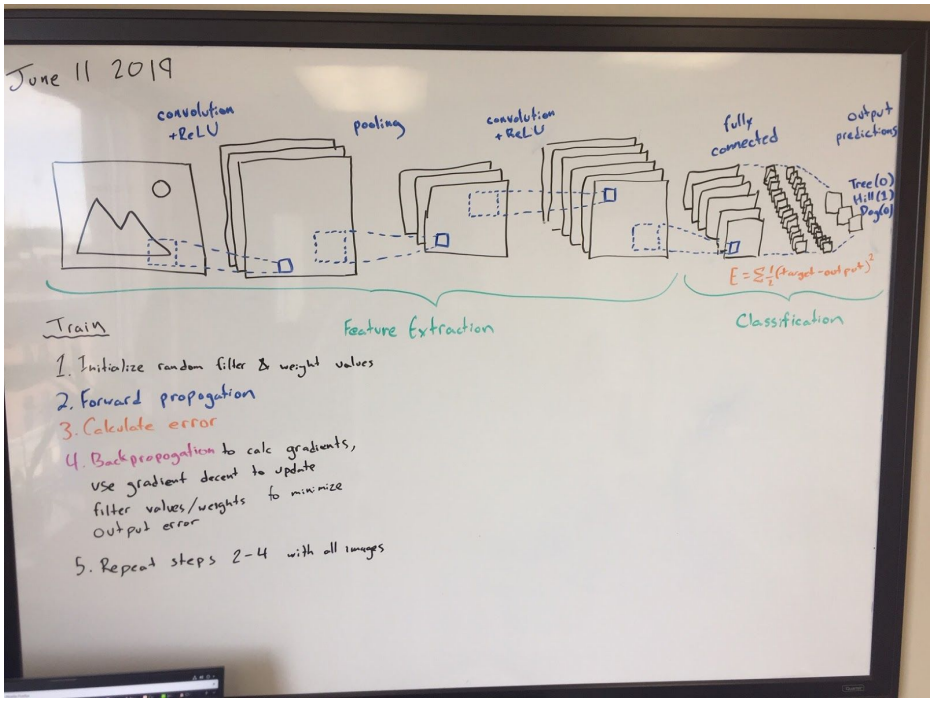

Figure 2: My illustration of a low level convolutional neural network and how to train one

3. Parallel and Distributed Methods

There are many different ways to approach distributed and parallelized computing. The first includes local training. Recent development in deep learning optimization, such as multi-core processing and GPU computation has drastically improved the computation time of training models. Methods for distributing work in this fashion will be looked into to determine if it can be expanded to seperate computers. The problem with some methods for local training is that they assume frequent communication can be established. The way DCP operates, small tasks should be given to workers with sufficient information to compute the task without referring back to the master computer (until return is called). Therefore, methods to parallelize deep neural networks so that computers can locally compute tasks is desired.

3.1 Data Parallelism

Data parallelism is a useful technique for training neural networks when the dataset is too large to fit on one computer (or training the entire dataset with one computer would take too long). With this method, data could be split up and sent to workers on the DCP network. This would allow machines to train the model locally, but only with a subset of the full data-set (Hegde, 2016). There are two ways to update the parameters of the master model using data parallelism: synchronous update and asynchronous update.

3.1.1 Synchronous Update

Synchronization in this context refers to the computation of loss-gradients. As mentioned earlier, evaluating the loss function of a model is assumed to be the most computationally expensive part of training a neural network, so it will be the main focus of research. Synchronous update methods wait for all loss-gradients in a data subset to be computed before moving on. In addition, before performing another iteration, or epoch of the dataset, all computers must return their tasks. This introduces a bottleneck in DCP, and unless each computer is benchmarked before sending it a task, this could make the program inefficient. Therefore, looking more into asynchronous update methods will be the focus of current research.

3.1.1.1 Parallel SGD

Proposed by Zinkevich and other researchers at Yahoo! Labs in 2010. This method of parallelizing stochastic gradient descent seems implementable, but due to the bottleneck of synchronous update methods described above, further research at this time will not take place.

3.1.1.2 Alternating Direction Method of Multipliers SGD (ADMM)

As described by Stephen Boyd in 2011, it proposes another method to update weights using SGD in a parallelized synchronous manner. The computation time of Parallel SGD and ADMM are the same, with complexity of $O(p * log(1/\varepsilon))$ per machine, where p is the size of the parameter, to achieve an error less than ε.

3.1.2 Asynchronous Update

With asynchronous methods, workers on the DCP network could locally train their subset of data on the full model, and when the computation is complete, return their trained model to the master so that the filter weights and biases can be updated. Unfortunately this is not as simple as just adding the returned weights to the master model. There have been multiple proposed methods to parallelize stochastic gradient descent in an asynchronous manner, and they will be explored and evaluated for use in DCP.

3.1.2.1 Downpour SGD

Developed by Jeffrey Dean and others at Google in 2012. As stated in the *Large Scale Distributed Deep Networks* paper by Dean: “Within this framework, we have developed the algorithm downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas”. After evaluation (Hegde, 2016), the computation time of downpour SGD is seen to be similar to ADMM and Parallel SGD, proving that it is a suitable method for use in DCP. Further research on how to implement this method will be conducted.