Distributed Neural Network

Akshay Naik
Indiana University Bloomington
Bloomington, Indiana
aunaik@iu.edu

Hardik Rakholiya
Indiana University Bloomington
Bloomington, Indiana
hrakholi@iu.edu

Jigar Madia
Indiana University Bloomington
Bloomington, Indiana
jmadia@iu.edu

ABSTRACT

In recent times, Neural Networks have become one of the most coveted form of Machine Learning technique due to its superior performance in patter recognition as well as classification tasks as compared to other techniques. However with the ever increasing need of data to train neural network efficiently, it has become more and more difficult to train a deep neural networks using large data sets. One solution which has been promoted is the use of Distributed Computing for performing processing of large scale data. Currently there are various environment available to create neural network in a distributed fashion. Thus, it is also important to understand how different distributed environments can lead to possible results and how well they perform. With this aim we present a study of three types of frameworks - Spark, TensorFlow and Harp, and compare the results side by side to see how well they perform.

KEYWORDS

Neural Networks, Distributed Computing, Apache Spark, Distributed TensorFlow, Harp-DAAL

1 INTRODUCTION

In the digital age, data has become one of the most important asset for any organization. From businesses to governments, almost everyone wants to tap into the data collected over a period of time to analyze how they have performed and how they will be performing in future. The first part is not that difficult since all you need to do is summarize and aggregate results to measure the performance. The problem comes with second part where you hope to analyze past data for predicting the future growth or losses. The field of pattern generation and prediction is where Machine Learning comes into picture. Machine Learning can help us solve this exact problem by helping us to create models which will predict an outcome given a set of scenario parameters. There are many techniques available for generating models like Decision Trees, Random Forest, Adaboost, SVM, Neural Networks etc which are known as classification models. Especially in recent years the reemergence of Neural Networks is significantly important since they have proved to give the best results for classification in many real world problems and have taken a central stage in today's world.

However these Neural Network models have their own complexity in terms of large data processing. To train these models, most of the times we need sufficiently large datasets which can be used to cover as many patterns as possible. The processing of these large datasets presents a problem which cannot be solved by standalone systems and thus we need to relay upon something called Distributed Systems. Distributed Systems in simple terms combine large number of small scale systems to create one big distributed environment which can divide and conquer any processing problem we have. It needs to be noted that we cannot simply combine many different machines into one big network and start processing data. We need to have special resource managers and schedulers which look after these resources and plan and optimize data processing in most efficient manner.

Distributed Systems offer speed, reliability, flexibility and availability to users when needed and thus have become increasingly important. Keeping this union of technologies in mind there has been an emergence of frameworks which are crafted to suit the requirements of exactly this job of running complex Machine Learning algorithms on Distributed Systems with minimum efforts and configuration setup like Apache Spark, TensorFlow and Harp-DAAL. These frameworks provide support to not only process data independently in distributed systems but also provide inbuilt frameworks for implementing Machine Learning algorithms like Neural Network in a distributed environment. This results into high speed model training and testing of the data sets to provide best possible accuracy. However given so many options it is also important to study how they perform on the same level and how they fare against each other so as to understand which framework provides most optimum results.

In this study we further explain what all related work has been done till now followed by a brief explanation and implementation details of these frameworks and the experiment results we have observed. We also drafted a set of conclusions which we drew based on the results we have got.

2 RELATED WORK

The problem of these frameworks is that there is no proper related work available on direct comparisons between them. We went to some articles, forums and documentations to understand how the frameworks are different. Since Harp-DAAL is a very recent framework it is still not used widely and as such does not offer any related work. We compare some of the key differences of Apache Spark and TensorFlow.

While Apache Spark is a data processing framework compared to TensorFlow which is a Machine Learning framework. The main purpose and design of Apache Spark is to undertake massive data processing on Distributed Systems through cluster and hardware optimization. On the other hand, TensorFlow relies on optimizing at the logical level through faster numerical computations, using graphs to map data flow etc. While Apache Spark considers hardware level optimization, TensorFlow relies more on software level optimization.

Also since Spark is a data processing framework, its support of Neural Networks is not as well established as TensorFlow. While TensorFlow provides an array of options to fine tune a Neural Network, Spark provides basic parameters in its ML library and not every possible tuning is provided. Based on these observations it has become a general trend to believe that while Spark can provide a better performance, accuracy is better on TensorFlow. With these pre-existing notions we would like to perform some actual experiments on these platforms and see how these assumptions hold up.

3 ARCHITECTURE AND IMPLEMENTATION

The basic building block of neural network is perceptron, each perceptron has a weight vector and bias assigned to it and it performs matrix multiplication of its weight vector with the input data to the perceptron and then it is passed through a non-linear function like logist or ReLu during forward propagation. For convenience of execution, the weight vector of perceptrons from one layer are concatenated to form weight matrix which is then multiplied with the input data of a particular layer. The output of the matrix multiplication is passed through a non-linear function to add non-linearity. During backward propagation the weights assigned to each perceptron is updated according to the loss calculated at the end of forward propagation. These are the main operations which are carried out while training a neural network.

Every distributed environment try to optimize the above mentioned computations during neural network training in different ways. For example, TensorFlow tries to optimize training at software level by optimizing numeric computations whereas Spark tries to optimize training at hardware level by using distributed batch processing.

In the following sections we will look at how Spark, Distributed TensorFlow and Harp-DAAL optimizes neural network training by using Distributed computing techniques.

3.1 Apache Spark[1]

Apache Spark in recent times is one of the most widely used analytical engine for processing large scale data at a tremendous speed. It is able to achieve a speed of 100 times faster than Hadoop Map-Reduce which is possible due to redesigned method of storing and processing data on a distributed environment.

Spark uses the concept of Resilient Distributed Datasets (RDD) for storing data in an efficient and fault tolerant method. In RDD's, the data is stored in memory in a distributed environment in partitions. Each operation on RDD is stored as well which makes it easy to replicate a RDD if it needs to be recreated in some other environment. Since RDD's are stored in memory it increases the performance of Spark compared to Hadoop where data is stored on disk and read-write operations are performed on it which are time consuming.

On a distributed environment, we can use Apache YARN resource manager to run Spark. Spark executors are run on a YARN container and the container running Spark Application Master controls the execution in different YARN nodes. The Spark Application Master requests resources from YARN resource manager and allocates tasks to each Spark executor contained at YARN Node Managers.Since YARN provides an efficient way of managing resources it provides an increase in SparkâĂŹs performance and the YARN configuration does not need to be changed for other applications as well.

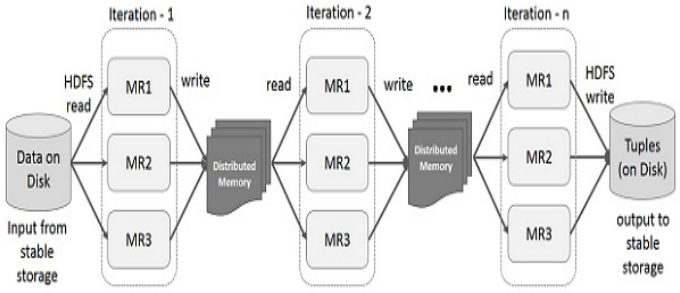

Figure 1: Spark RDD[6]

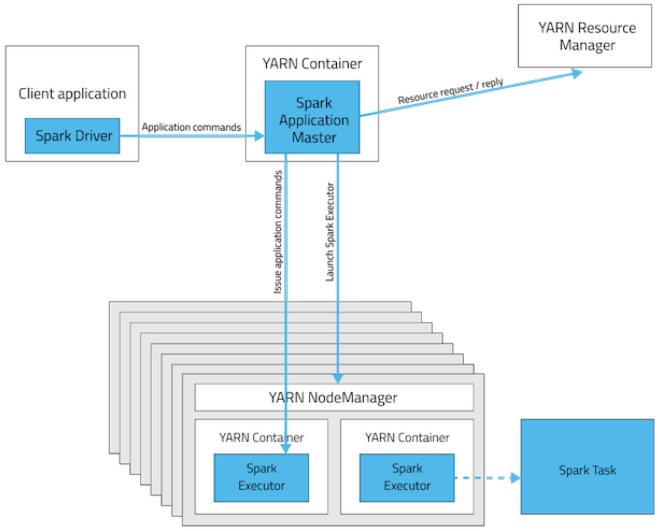

Figure 2: Spark-YARN Framework[2]

3.2 Distributed TensorFlow [3]

In TensorFlow, cluster is a set of tasks that participate in the distributed execution of a TensorFlow computation graph. Each task has associated TensorFlow server, which contains a master that can be used to create sessions, and a worker that executes operations in the graph. Each task runs a kind of job, i.e. either worker or parameter server.

We create a 'ClusterSpec' object and send it as an argument to 'Server' object which we use to start a task. The figure below represents one such example of cluster setting with two nodes.

For the purpose of our experimentation we are using two nodes for execution of distributed TensorFlow. We have incorporated model parallelism architecture [8] in TensorFlow to speed up the training time. In Model parallelism, the static graph is divided among the worker nodes and each worker node performs the computation specific to the part of the graph it contains.

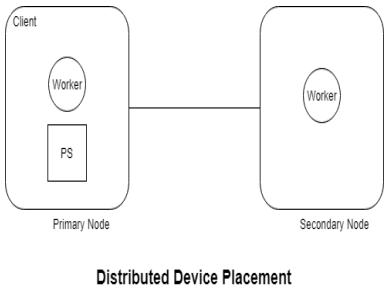

Figure 3: Distributed Device Placement

For example, if we have four layered network and two nodes then by default two layers will be assigned to each node in a round robin fashion unless specified the manner to distribute the network layers among the workers.

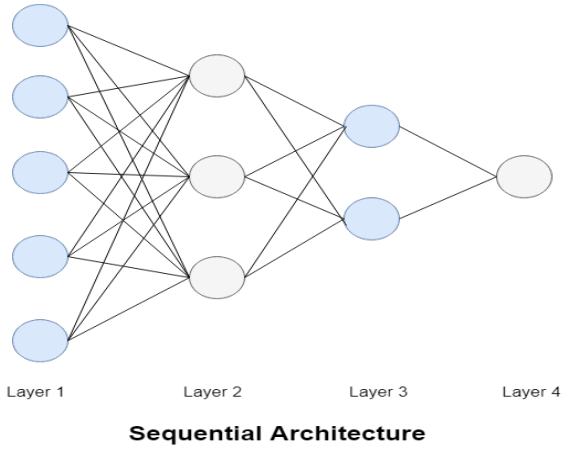

Figure 4: Sequential Neural Network Architecture

In order to carry out experiment we have planned we were not able to find an already implemented neural network in TensorFlow wherein we can change hyper-parameters like number of layers, number of neurons and number of epochs. Thus, we created a new code for TensorFlow implementation wherein we can change the above-mentioned hyper-parameters while running the code script. This will help us to compare TensorFlow to Harp and Spark with different parameter values

.

To run the code on Juliet server, we have installed TensorFlow for a particular user as we don't have super user permission and tested our code whether it is working or not.

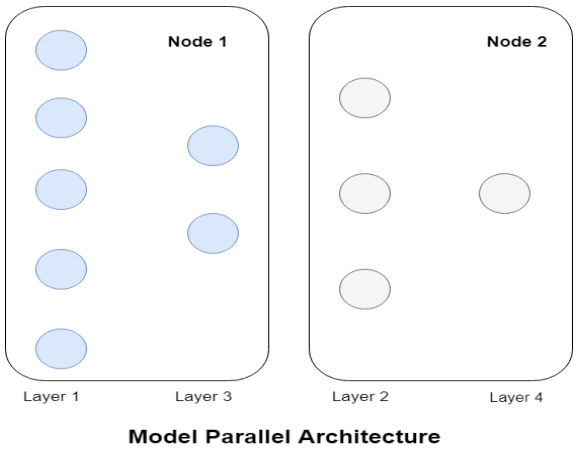

Figure 5: Model Parallel Neural Network Architecture

3.3 Harp-DAAL [5] [4]

Harp is IU's own high performance computing framework providing support for distributed data analytics and machine learning. The original harp framework only supports development in Java applications, which is a common choince of the Hadoop ecosystem. However, pure java applications still lack the support for high-end HPC architectures such as Intel's Xion Phi. This is where Intel's DAAL(Data Analytics Acceleration Library) comes into play. DAAL provides the users of well optimized building blocks for data analytics and machine learning applications on Intel's architectures. The application can exploit all of the hardware threads on multiple core platforms by invoking DAAL's native kernels. Running computation intensive workloads with DAAL is thus comparatively faster.

Harp-DAAL framework consists of two layers: a communication layer and a computation layer. The Harp framework acts as a communication layer which plug into the Hadoop ecosystem and handles the communication between the mapper nodes. DAAL on the other hand, acts as a computation layer which is highly optimized for running MPI-like operations handles all the computation on the mapper nodes.

Initially we ran our experiments without using DAAL which took hours to complete. So all the subsequent experiments were performed using Harp-DAAL. The implementation for Neural Network algorithm along with other experimental algorithms are present in ml module of harp's maven project. We, therefore, decided to use Harp-DAAL's native Neural Network algorithm in our experiments.

We used Apache Hadoop and Apache Yarn for running Harp. Hadoop's core part Hadoop Distributed File System provides an efficient and fault tolerant method to store data across nodes and Apache Yarn provides efficient way of managing node resources. It is also useful for job scheduling/monitoring. We have used 2 juliet nodes for our experiment which consists of 1 namenode(master) and 2 datanodes(slaves) in Yarn's terms. The algorithm ran for these 2 mappers with 16 threads on each node with maximum allocated

memory. The batch size was set to 50 with varying number of iterations.

We faced many issues during our experiments which significantly slowed us down. Initially the libJavaAPI.so of DAAL's library on the master branch was non-functional and thus crashed the application every time. This issue was also faced and handled when Akkas supplied another version of the file during lab sessions. We were then able to run kmeans algorithm but not neural network algorithm. We tried to debug it by ourselves and also raised the issue with Langshi. The test scripts for neural network algorithm were not updated until one week before thanksgiving break. We had pre-processed the data into CSV format as specified in the documentation by converting binary data into comma-separated int values. But Harp-DAAL did not seem to accept integer values. The problem was later resolved when Prof. Peng supplied his own data with float values. We were also not able to change the topology of neural network layers other than the default one since Harp-DAAL's output returned NaN values whenever we added a new layer into the topology. Not able to change the configuration of our neural network is the reason we have limited results with Harp-DAAL and were not able to compare it to Tensorflow and Spark. One other minor issue we faced was that Harp-DAAL did not exit the application gracefully when we tried to run the algorithm with 100 epochs although the algorithm had finished executing much earlier.

To keep consistency in all three platforms, we have implemented fully connected neural network instead of convolutional neural network which would have given us better performance. Also, we have used gradient descent optimizer instead of ADAM optimizer. We have used accuracy as evaluation criteria which is consistent across all the three frameworks.

4 EXPERIMENTS

We have structured our experiment in such a way that we can compare the accuracy as well as the computation time for training the neural network taken by each framework for different set of hyper-parameters setting.

4.1 Settings

We change hyper-parameters such as number of layers, number of neurons each layer and number of epoch and compare the accuracy as well as the training time required by each framework.

4.2 Input

We are using MNIST data-set as our input data which is a very well known data-set for Handwritten digits recognition task. This data-set was once used for the purpose of bench-marking neural network performance. It contains 60000 training images and 10000 test images of digits which can be classified into one of the ten classes, i.e. 0 to 9.

4.3 Results

Table 1: TensorFlow Results

Layers	Epochs	Time	Train Accuracy	Test Accuracy
50,25	10	46.18 s	97.64	96.13
	100	459.29 s	99.92	97.50
	300	1372.51 s	99.99	97.43
	500	2270.90 s	99.98	97.19
150,50,25	10	46.61 s	98.37	96.98
	100	475.44 s	100	98.10
	300	1439.70 s	99.99	97.99
300,150,50,25	10	54.16 s	98.11	96.86
	100	547.31 s	99.99	98.51
	300	1642.71 s	100	98.07

Looking at the TensorFlow results we can see that for each layer combination the training time increases drastically with increase in the number of epochs but the accuracy slightly changes. One reason for small change in accuracy with increase in epochs might be that the data distribution was very simple for TensorFlow to learn in few number of epochs, this can be seen from the accuracy we are getting with only 10 epochs.

Table 2: Spark Results

Layers	Epochs	Time	Train Accuracy	Test Accuracy
50,25	10	84.30 s	54.82	55.12
	100	592.70 s	92.60	92.40
	300	1741.27 s	97.21	95.17
	500	2844.10 s	98.70	95.21
150,50,25	10	204.12 s	35.13	34.44
	100	1602.40 s	90.20	90.03
	300	4742.67 s	97.34	95.30
300,150,50,25	10	423.36 s	10.53	10.46
	100	3834.22 s	86.58	86.85
	300	11587.23 s	96.35	94.76

Table 3: Harp-DAAL Results

Layers	Epochs	Time	Test Accuracy
784,40,2	10	130 s	99.42

Based on the above results we can see that Spark results are improving with increase in Epochs rather than layers. For ex. results of hidden layers 50,25 with 300 epochs is almost same as results of other increased hidden layers for same epochs but the time taken is proportional to increase in both epochs and hidden layers.

On comparing results for both the frameworks we can definitely observe that TensorFlow is performing much faster than Spark in training its Neural Network and also provides much better accuracies in many of the cases with less layers or less epochs.

4.4 Analysis of results

To analyze results better we decided to plot the results we got from testing on Spark and TensorFlow and see side by side what differences we can get from it. Let us first analyze the differences in accuracy of testing data.

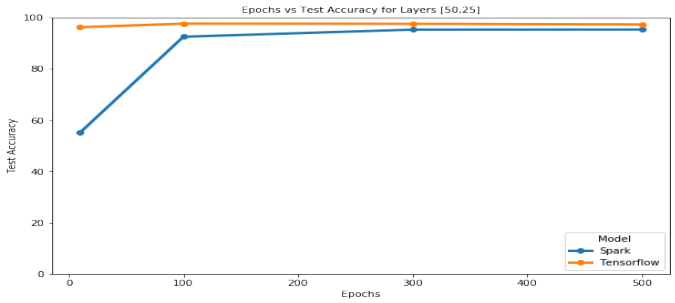

Figure 6: Accuracy's vs Epochs for Layers [50,25]

In the above graph we can see the results keeping layers constant and changing the epochs. We can see initially Spark gives very poor accuracy compared to TensorFlow when epochs are less but it gives almost similar accuracy when we increase the number of epochs. Let us see the results for same epoch values by adding an extra hidden layer.

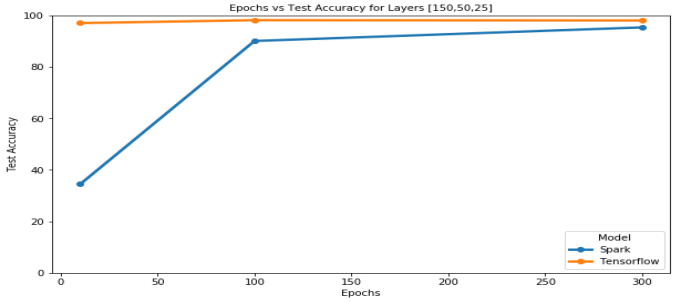

Figure 7: Accuracy's vs Epochs for Layers [150,50,25]

We see that on adding hidden layer the accuracy for epoch 10 dropped for Spark but it was almost same as previous test case for TensorFlow.

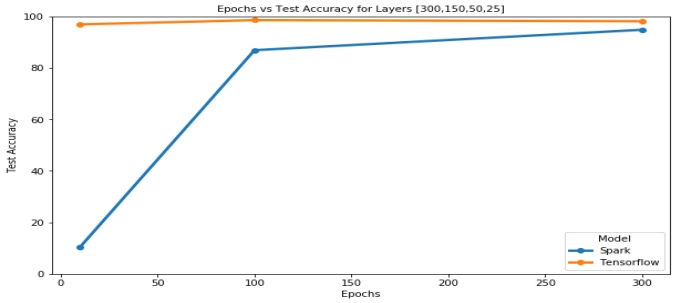

Figure 8: Accuracy's vs Epochs for Layers [300,150,50,25]

Again on adding hidden layer we see drop in accuracy for Spark when epoch is 10. Based on these results we can assume that Spark needs more number of iterations to get competitive results compared to TensorFlow which can perform really well even with less iterations.

Lets check how the results were in terms of time taken to train the model. The results are plotted in a similar manner with Training time in seconds replacing accuracy on y-axis.

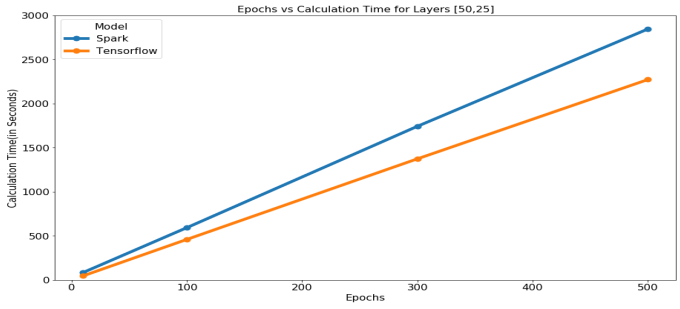

Figure 9: Time vs Epochs for Layers [50,25]

We can see that TensorFlow takes less time than Spark with increase in epochs.

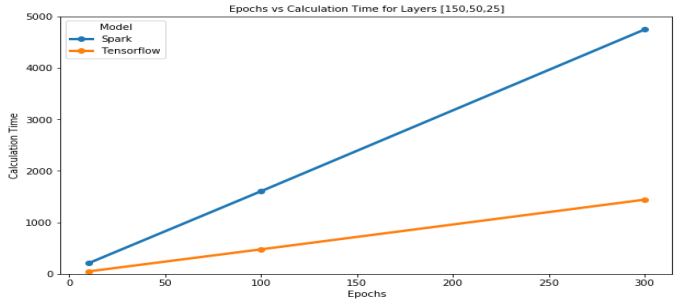

Figure 10: Time vs Epochs for Layers [150,50,25]

The difference between time taken increases with increase in number of layers and epochs.

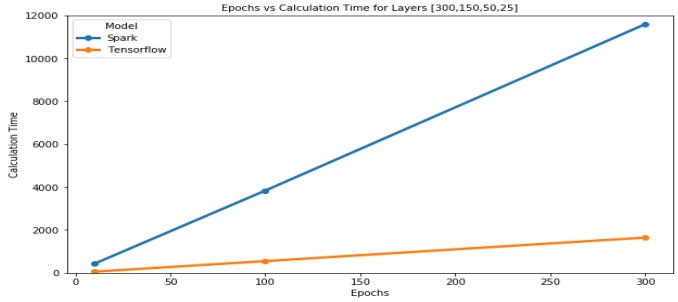

Figure 11: Time vs Epochs for Layers [300,150,50,25]

Based on the above graphs we can say TensorFlow outperforms Spark by a wide margin as we go on increasing epochs and hidden layers.

From the above results and our analysis we feel the wide difference in terms of time and accuracy is due to that fact that while Spark is good at working on distributed systems and it tries to increase speed of training the Neural Network by distributing the computations[7], but Distributed TensorFlow outperforms it due to the fact that it not only distributes computations but since being custom made for Machine Learning algorithms, it optimizes the numerical computations for training the model.

5 CONCLUSION AND FUTURE SCOPE

From our experiment, we can see that distributed TensorFlow outperforms Spark for Neural Network training.

In future, one interesting experiment to carry out will be comparison of Distributed TensorFlow with Spark Tensorflow. We feel that the Spark TensorFlow architecture is similar to Distributed TensorFlow between-graph replication architecture and would be interesting thing to compare their performance.

6 ACKNOWLEDGEMENTS

The authors would like to thank Prof. Judy Qiu, Prof. Bo Peng, Langshi Chen and Selahattin Akkas for their guidance in understanding concepts of how different distributed framework works from application level down to hardware level and also for providing us access to Juliet server to carry out our experiments.

REFERENCES

[1] [n. d.]. Apache Spark Documentation. https://spark.apache.org/docs/latest/
[2] [n. d.]. Apache Spark Resource Management and YARN App Models. https://blog.cloudera.com/blog/2014/05/apache-spark-resource-management-and-yarn-app-models/
[3] [n. d.]. Distributed TensorFlow. https://www.tensorflow.org/deploy/distributed
[4] [n. d.]. The Harp Java API. https://dsc-spidal.github.io/harp/api/
[5] [n. d.]. Harp Neural Network. https://dsc-spidal.github.io/harp/docs/
[6] [n. d.]. Spark RDD Tutorial. https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
[7] Saman Biookaghazadeh. [n. d.]. Decoding Apache Spark Neural Network. https://samanaghazadeh.wordpress.com/2016/10/14/apache-spark-neural-network-decoded/
[8] Kuo Zhang. [n. d.]. Data parallel and model parallel distributed training with Tensorflow. http://kuozhangub.blogspot.com/2017/08/data-parallel-and-model-parallel.html

7 INDIVIDUAL CONTRIBUTION

7.1 Akshay Naik

- Implemented Feed Forward Neural Network in Distributed TensorFlow from scratch
- Fine tuned hyper-parameters
- Experimented with various hyper-parameter and calculated accuracy and training time
- Researched about different distributed architecture for TensorFlow like model parallel, in-graph replication and between-graph replication. Finally implemented model parallel architecture
- Some of the problem faced during initial times was to understand how to run a neural network in distributed fashion in TensorFlow, which we were comfortable once the lab session related to TensorFlow was taken
- Compiled TensorFlow related report and contributed equally on other common sections of report with Jigar and Hardik

7.2 Hardik Rakholiya

- Understood Harp's high level architecture through its API and available documentation on Harp's website and Intel DAAL's website
- Applied the concepts learned in lab sessions to configure the Harp-DAAL
- Data pre-processing to make it compatible with Harp-DAAL Neural Network algorithm
- Debugged the source code when the algorithm failed to run. Raised issues with Langshi and Prof. Peng to get them resolved.
- Helped Jigar in running the PySpark implementation

7.3 Jigar Madia

- Studied Apache Spark architecture and framework for understanding the underlying concepts of RDD's and Neural Network implementations
- Implemented Neural Network on PySpark
- Configured Apache Spark with YARN
- Tested cases for Neural Netwoks with different parameters to optimize the accuracy and reduce time as much as possible
- Compiled Spark related topics in report and contributed equally on other common sections of report with Akshay and Hardik