Distributed Logistic Regression for Multiclass Classification

Suraj Panwar, Mtech DESE, 14644

1. Introduction

The project undertakes the task of implementation of Logistic Regression both on a local setting as well as a distributed implementation on multiple nodes using Tensorflow. The dataset used for this project is extracted from DBPedia which is a 50 class classification problem with multiple labels per example.

The random guessing accuracy for this dataset considering only 1 class per example is 2% and if we consider the possibility of multiple labels, this further decreases. This is taken as the baseline accuracy for this project. Due to having multiple labels the algorithm classification becomes a difficult task, however, during the classification we consider the most probable label provided by the Logistic Regression algorithm and if it is one of the true classes it is considered as the correct classification.

This project also explores the application of Tensrflow Distributed architecture framework for the task of the Logistic Regression Implementation by assigning the task of creating Parameter Server and worker nodes and allocating work between them for processing. This significantly improves the computation speed of the algorithm and provides better results.

All the mentioned results have been recorded and mentioned in this report along with possible explanations of the cause. The algorithms are however compared mostly on the grounds of test accuracy and execution time.

2. Logistic Regression

2.1. Local Logistic Regression

The local Logistic Regression has been implemented on a personal laptop with no distributed implementation. The main training corpus has been the DBPedia from which the test and dev sets have also been extracted. The model has been implemented using L2 regularization to increase the model performance. The model is updated using performance on the development set and finally the accuracy is evaluated on the test dataset provided.

The probability of a class for a given sample have been modeled using a multiclass Logistic Regression model or a Softmax model with output 50 output classes the model gives a probability estimate of the example belonging to a specific class. The class with the highest probability is assigned as the class label. If the class label is one of the given labels, the sample is considered as correctly classified.

2.2. Observations

The Logistic regression model was made for different types of learning architectures using increasing, decreasing and a constant learning rate. The model was also optimized for different values of training epochs, learning rate and regularization constant. Stochastic gradient descent was used for training the model as well as batch gradient descent, however only Stochastic Gradient Descent accuracy has been reported here.

Two embedding methods were also tested for the performance comparison, both Doc2Vec embedding and tf-idf vectorization was tested for measuring accuracy of the mode. The embedding size of the documents used here was fixed at 300 length, however the embedding length was also varied to note the effect of the embedding for the output accuracy. The embedding size checked were 100, 500, 300 and 1000 even though the 1000 and 500 resulted in increase in the model accuracy, the increase was only marginal compared to the 300 embedding length, but resulted in significant increase in the processing time. So we have used a 300 length embedding for the rest of the assignment. Among the two methods, tf-idf vectorization gave significantly better results then the Doc2Vec embedding because of the limited training corpus size and hence has been used in the rest of the assignment.

For the train and the test time reporting the inbuilt time module was used and the time for 10 epochs was averaged over two experiments. The reported time for testing and training set evaluation has been reported in the table.

The train and the test accuracies, the hyper parameter and the embedding type is reported in the given table and has been optimized on the development set for the optimal performance. These hyper parameters are then used throughout the rest of the assignment when we explore the distributed implementation of the the Logistic Regression.

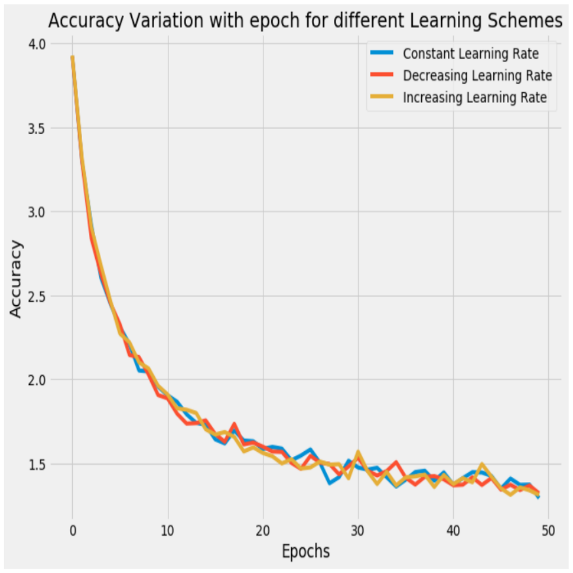

Figure 1. Accuracy vs Epoch plot for different learning rate schemes

Features	Optimal
Learning Scheme	Constant
Learning Rate	1e-1
Decay/Growth rate	1e-6
Embedding	Tf-IDF
Embedding Size	300
Regularization	L2 Norm
Reg. Constant	1e-2
Train Accuracy	82.71%
Test Accuracy	79.72%
Train Time	412 sec
Test Time	56 sec

Table 1. Logistic Regression on Local Machine.

2.3. Distributed Logistic Regression

2.3.1. Framework

For distributed implementation of Logistic Regression we need a framework which could create a parameter server and worker nodes, on which the task at hand can be distributed for faster execution. For this purpose I am using Tensorflow as it allows to easily create parameter servers and worker nodes and is able to easily distribute work among its worker nodes. Tensorflow also allows functionality to easily switch between synchronous as well as asynchronous modes and thus is further helpful in our model here.

2.3.2. BSP SGD setup

The BSP SGD model was trained in Tensorflow using the hyper parameters decided on the local implementation. The number of workers decided on this task was 2, with one parameter server hosted on 3 nodes of the Turing cluster. Besides running the model on Turing nodes the model was also tested on local machine with different ports and the performance was in sync with the previous one.

Features	Optimal
Worker Nodes	2
Parameter Server	1
Learning Rate	1e-1
Decay/Growth rate	1e-6
Embedding	Tf-IDF
Embedding Size	300
Regularization	L2 Norm
Reg. Constant	1e-2
Train Accuracy	80.39%
Test Accuracy	76.26%
Train Time	296 sec
Test Time	42 sec

Table 2. Logistic Regression using BSP SGD.

As before the training time and the test time were noted by averaging the output of 2 different executions to increase the consistency of the model. The number of replica model can always be changed by changing the replica optimizers.

The performance of the BSP model was in sync with the local execution however, it excelled in the area of training time and the test time requirement.

As a BSP SGD combines the model of all different workers in regular workers, the processing speed of the BSP SGD is comparatively lesser than the counter part of Asynchronous training.

2.3.3. Asynchronous SGD

Asynchronous SGD has also been trained on Tensorflow using the Turing clusters nodes for execution. The setup consists of the hyper parameters tunes in the local Logistic Regression setting and then distributes the work between variable worker nodes and 1 parameter server.

The model trained is completely asynchronous in the fact that the gradient update can take place at any moment without the reference of the other workers and hence has better speed improvements than synchronous processes. The training time and test time along with the accuracy obtained with the model for 2 number of workers is also tabulated in this report.

The variation of the training loss along with the number of worker nodes was also plotted for identifying the relation of the loss with worker numbers. The result has also been plotted. The general body for creating the Distributed archi-

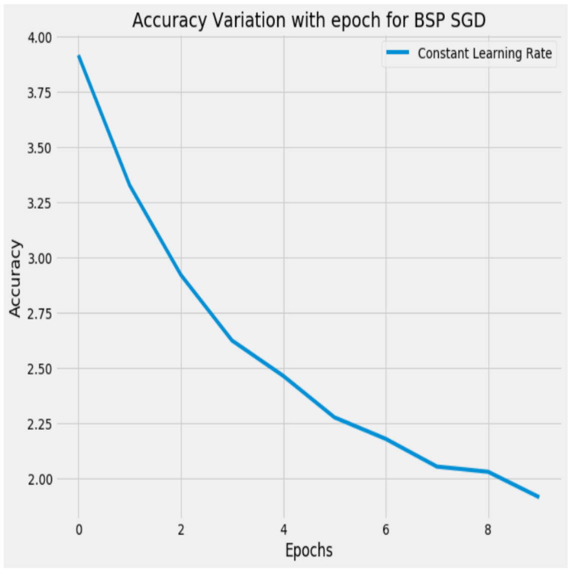

Figure 2. Accuracy vs Epoch plot for BSP SGD

Features	Optimal
Worker Nodes	2
Parameter Server	1
Learning Rate	1e-1
Decay/Growth rate	1e-6
Embedding	Tf-IDF
Embedding Size	300
Regularization	L2 Norm
Reg. Constant	1e-2
Train Accuracy	79.38%
Test Accuracy	77.44%
Train Time	237 sec
Test Time	35 sec

Table 3. Logistic Regression using Asynchronous SGD.

tecture has been made with help taken from the Tensorflow website for Distributed Computing. From the results obtained it also seems that in general model converge faster with the increase in the worker nodes.

2.3.4. Bounded Asynchronous (SSP) SGD

The Bounded Asynchronous (SSP) SGD is also implemented in Tensorflow, but here instead of giving delay in time for the gradients to accumulate and turn stale, we have used a gradient measure where at each step the optimizer collects 50 gradients before applying to variables. This can then result in resulting in 50 gradient stale limit.

The variation of the gradient staleness parameter is varied to observe the effect of this parameter on training of the model. In general this model performs better than the BSP SGD but

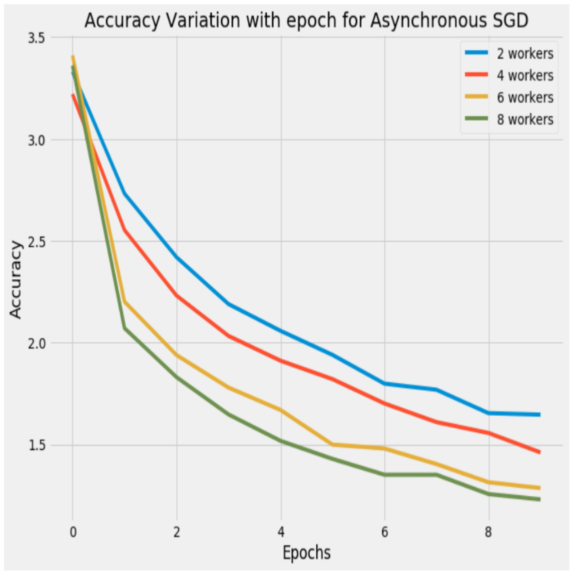

Figure 3. Accuracy vs Epoch plot for Asynchronous SGD

lacks in comparison to complete Asynchronous SGD. The performance was compared for a gradient staleness delay of 50 and for 10 epochs. The worker for the model comparison was a 2 worker model with 1 parameter server.

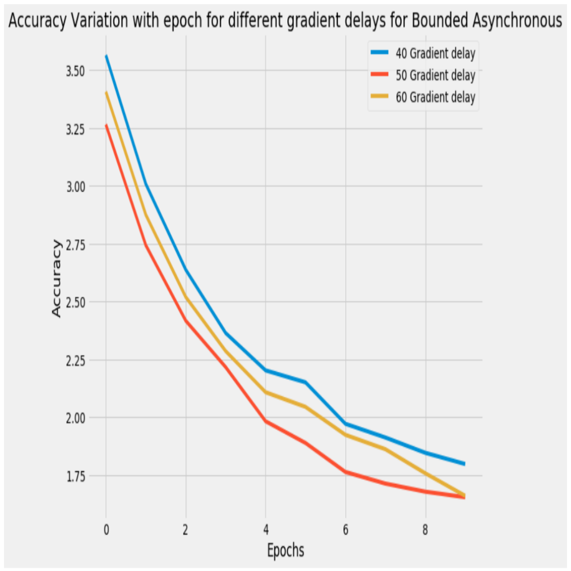

Figure 4. Accuracy vs Epoch plot for SSP for different gradient staleness.

2.4. Model Comparison

The final model comparison can be seen in the given plot. From the plot it seems that the Asynchronous SGD is the fastest of the algorithms proposed followed by Bounded SGD for gradient staleness of 50 and then BSP SGD. For

Features	Optimal
Gradient Delay	50
Worker Nodes	2
Parameter Server	1
Learning Rate	1e-1
Decay/Growth rate	1e-6
Embedding	Tf-IDF
Embedding Size	300
Regularization	L2 Norm
Reg. Constant	1e-2
Train Accuracy	79.38%
Test Accuracy	77.44%
Train Time	237 sec
Test Time	35 sec

Table 4. Logistic Regression using Asynchronous SGD.

all the model testing was performed on 2 worker and 1 parameter server configuration. However, in Asynchronous nature of the model training the resource utilization was highest in the configuration so we can take it as a interchange of performance vs resource utilization.

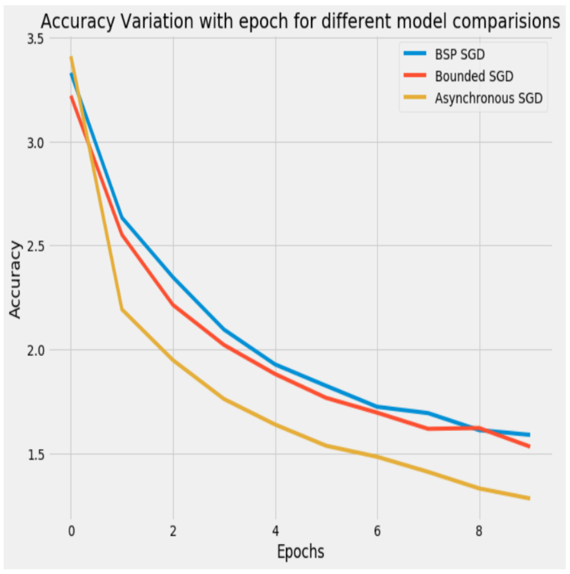

Figure 5. Accuracy vs Epoch plot for different distributed configurations.

2.5. Resources Used

The main help taken for this project was from the Tensorflow Distributed computing website for code help as well as various blogs.

3. Github Resource

The codes for all the above executions have been uploaded at Github at :

https://github.com/Suraj-Panwar/Distributed-Logistic-Regression.git