Distributed MultiClass Multilable Logistic Regression

Dola Ram, Mtech DESE, 15197

1. Introduction

The project was to implementation of Logistic Regression both on a local setting as well as a distributed. Implementation was done 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. For classification of multilabel examples 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 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. Softmax model was used for the classification of multiclass multilabel classification. The local Logistic Regression has been implemented on a personal computer with no distributed implementation.

2.2. Distributed Logistic Regression

Framework used here is 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. Bulk Synchronous Parallel(BSP)

The bulk synchronous parallel (BSP) algorithm is often referred to as model averaging. In this model, data are distributed across multiple workers. Each worker updates its local model replica independently using its own portion of data with SGD. Periodically the local models are averaged and the generated global model is synchronized across workers. Worker node on task was 3 and parameter server was 1. As before the training time and the test time were noted by averaging the output of 3 different executions to increase the consistency of the model. The number of replica model can always be changed by changing the replica optimizers.

Table 1. Logistic Regression using BSP SGD.

LOCAL	MAPREDUCE
WORKER NODES	3
PARAMETER SERVER	1
LEARNING RATE	0.1
DECAY/GROWTH RATE	0.000001
EMBEDDING	TF-IDF
TRAIN ACCURACY	79.85%
TEST ACCURACY	73.86%
TRAIN TIME	570 SEC
TEST TIME	60 SEC

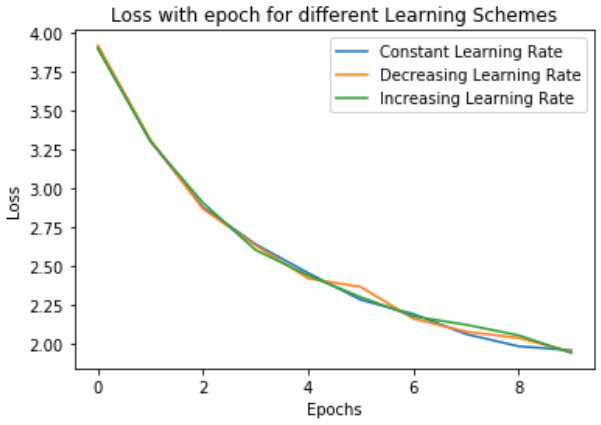

Figure 1. Loss vs Epoch plot for Asynchronous SGD.

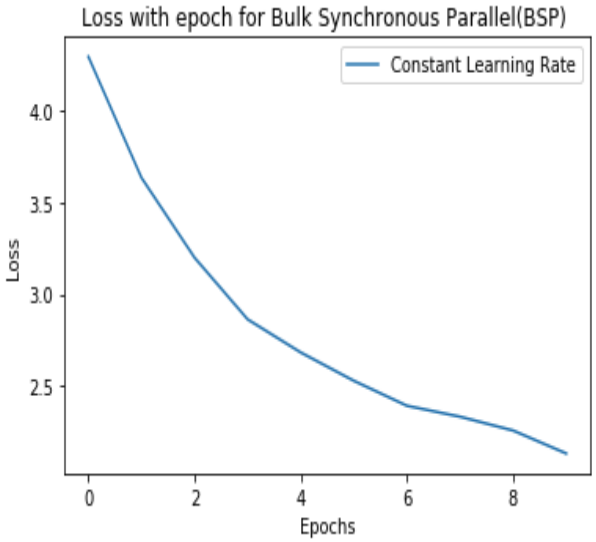

Figure 2. Loss vs Epoch plot for BSP SGD

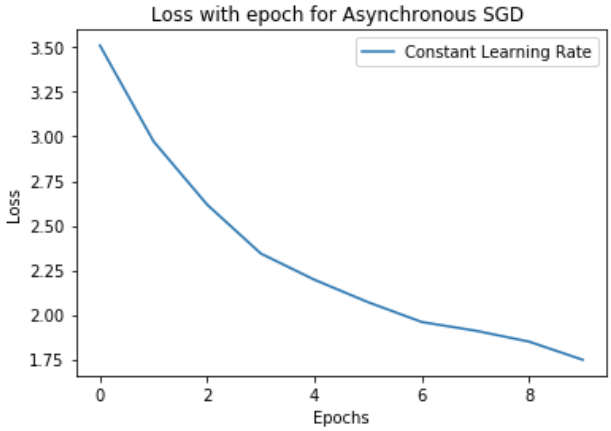

Figure 3. Loss vs Epoch plot for Asynchronous SGD.

3. AsyncronousS SGD

ASGD uses 1 parameter sever and 3 local workers. Each worker independently and asynchronously pulls the latest global model w from the parameter server, computes the gradient grad w with a new minibatch, and sends it to the parameter server. The parameter server always keeps the current model. Asynchronous SGD has also been trained on Tensorflow using the Turing clusters nodes for execution. 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 general body for creating the Distributed archi- 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.

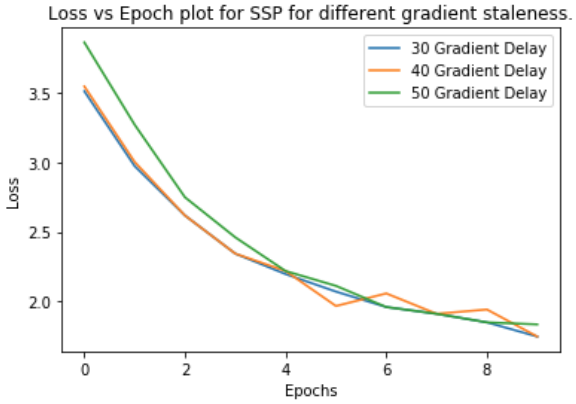

Figure 4. Loss vs Epoch plot for Asynchronous SGD.

Table 2. Logistic Regression using Asynchronous SGD.

LOCAL	MAPREDUCE
WORKER NODES	3
PARAMETER SERVER	1
LEARNING RATE	0.1
DECAY/GROWTH RATE	0.000001
EMBEDDING	TF-IDF
TRAIN ACCURACY	77.05%
TEST ACCURACY	72.76%
TRAIN TIME	536 SEC
TEST TIME	52 SEC

4. Stale Synchronous(SSP) SGD

As SSP can be view as another way to weaken the synchronization for the pushing, it can be potentially combined into one algorithm by replacing the partial pushing with SSP. Stale Synchronous 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 40 gradients before applying to variables. This can then result in resulting in 40 gradient stale limit.

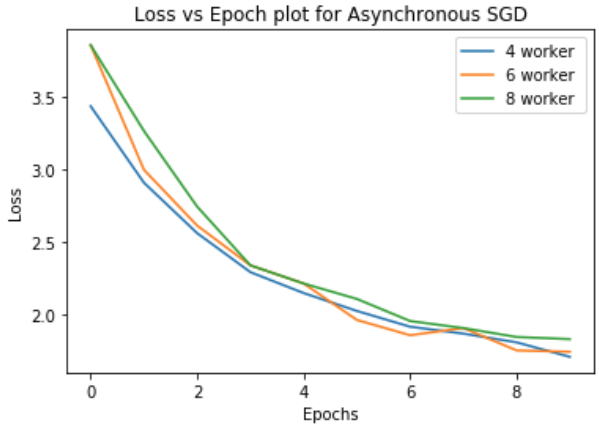

Figure 5. Loss vs Epoch plot for SSP for different gradient staleness.

5. Model Comparison

From the plot it seems that the Asynchronous SGD is the fastest of the algorithms proposed followed by Bounded SGD for gradient staleness of 40 and then BSP SGD. For all the model testing was performed on 3 worker and 1 parameter server configuration.

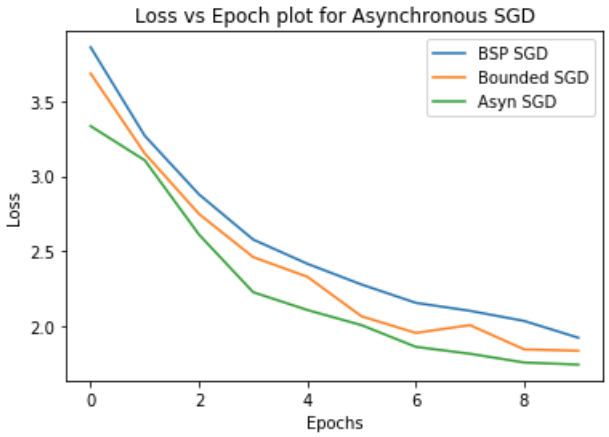

Figure 6. Loss vs Epoch plot for different distributed configurations.

6. Resources Used

Distributed implimentation was build upon the code provided on the tensorflow website https://www.tensorflow.org/deploy/distributed

7. Github Resource

Project program files along with the required resources have been uploaded at github at the following link:

https://github.com/dolaram/Distributed_Multiclass_multilabel_Logistic_Regression.git