Machine Learning with Large Datasets
Implementing logistic regression with parameter servers

Shubham Bansal(14666)

In this assignment, you will implement and evaluate a L2-regularized logistic regression in the standalone and distributed setting using Parameter Server.

1. Data Preprocessing

We have preprocessed input documents and output labels seperately for each of training,development and test data to obtain them in the desired format for model training.

Input Documents:

- Cleaning of html links
- Removal of punctuation marks
- Removing alphanumeric/numeric words
- Converting to lowercase
- Removing words with length less than or equal to 2
- Removal of stopwords
- Stemming using snowball stemmer

Further, We have used BOW representation of each document. Before applying BOW, we have created vocabulary and have kept words with wordcount between 100 and 10000. Also, corresponding to each word, word to index mapping has been done. **We observed significant improvement in performance with with stemming. Total vocab size is 7625.**

Output Labels:

For each data point, output label have been transformed into multilabel format and further divided by counts of true labels corresponding to it to convert it to probability distribution.

Trainable Parameters:
Since, total number of classes are 50. Hence we have total 38125 trainable parameters with logistic regression.

Table 1. Train/test time vs type of learning rate

LEARNING RATE	DECAY RATE	EPOCHS	TRAIN TIME	TEST TIME
CONSTANT	1	100	878S	1.03S
DECAYING	0.96	100	904S	1.02S
INCREASING	1.04	100	908S	1.02S

2. Local Logistic Regression

The local implementation has been experimented on LEAP lab cluster on a single node with no parallelization support.
Worker: ["10.1.1.252:2224"] (node 2)
hyperparameters:
No of epochs : 150
Batch size: 5000
L2 Regularization constant : 0.001
starting learning rate: 0.002
Experiment:
We have experimented with learning rate in local settings. To observe the effect, training has been run for 150 epochs and plot of cross entropy training loss vs epochs is shown in Figure 1.
Also, a plot of test set accuracy with number of epochs has been shown in Figure 2. Table 1 compares the training time and test time performances with constant, decaying and increasing learning rate. Starting learning rate has been kept at **0.002**.

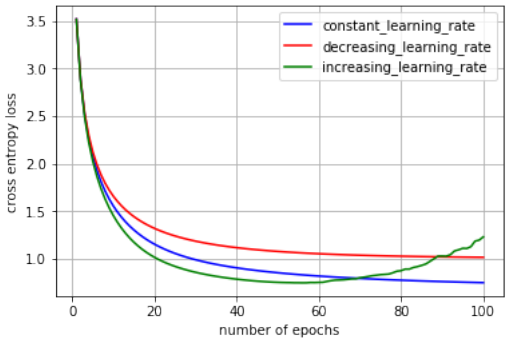

Figure 1. Cross Entropy Training Loss vs Number of Epochs

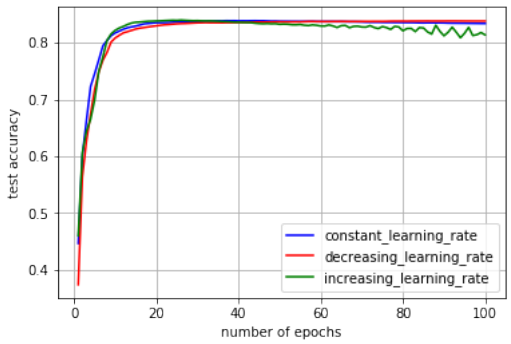

Figure 2. Test accuracy vs Number of Epochs

3. Distributed Logistic Regression

We have used tensorflow distributed framework which enables us to create a cluster of servers and then distribute computation graph among them. Also, it provides with very abstract way to train in both synchronous and asynchronous manner without worrying about communication among servers and distribution of data.**Between graph replication** approach has been used where every worker has the copy of computation graph. For all our experiments, we have kept a single parameter servers and number of worker nodes are varied. Cluster details are as follows:

CLUSTER: LEAP LAB, EE, IISc
Parameter server: ["10.1.1.254:2225"] (node 0)
Worker 0: ["10.1.1.253:2223"] (node 1)
Worker 1: ["10.1.1.252:2224"] (node 2)
Worker 2: ["10.1.1.251:2222"] (node 3)

HYPERPARAMTERS:
No of epochs : 100
Batch size: 5000
L2 Regularization constant : 0.001
learning rate: exponential decay of 0.96 with starting learning rate of 0.002

3.1. ASYNCHRONOUS PARALLEL

In this approach of training, each worker(replica of the graph) has an independent training loop that executes without coordination. Gradients are calculated and applied immediately without waiting for other workers.
Experiment:
For the case of asynchronous parallel, I have experimented with number of workers and plot of cross entropy training loss vs number of epochs has been shown in Figure 3. Also, training time, testing time and accuracy have been compared in Table 2.

Table 2. Train/test time and Test accuracy vs Number of workers for ASP

	2 WORKERS	3 WORKERS
WORKER0 TRAIN TIME	608S	598S
WORKER1 TRAIN TIME	1073S	1060S
WORKER2 TRAIN TIME	-	1207S
TEST TIME	0.810S	0.815S
TEST ACCURACY	0.84	0.84

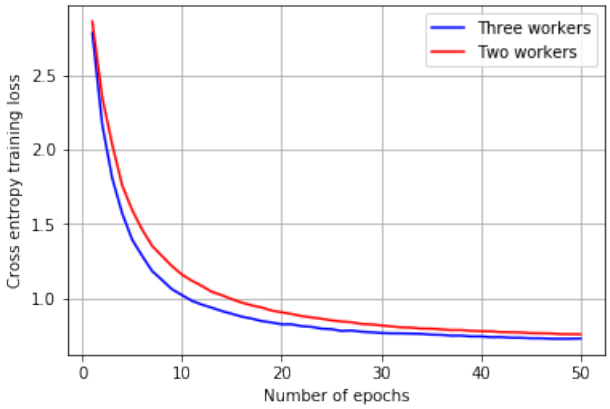

Figure 3. Cross entropy training loss vs Number of Epochs for ASP

3.2. BULK SYNCHRONOUS PARALLEL

The gradients are aggregated after each step and then applied to paramters in parameter server. Figure 4 shows the plot of cross entropy training loss vs number of epochs for 2 workers. Also, Table 3 reports the test accuracy,test time and compares the training time of various workers.

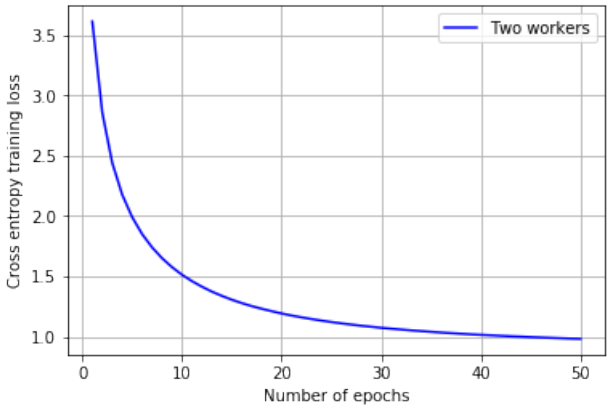

Figure 4. Cross entropy training loss vs Number of Epochs for BSP

Table 3. Train/test time and Test accuracy vs Number of workers for BSP

	2 WORKERS
WORKER0 TRAIN TIME	1180S
WORKER1 TRAIN TIME	1210S
TEST TIME	0.82S
TEST ACCURACY	0.84

Table 4. Train/test time and Test accuracy vs Number of workers for BSP

STALENESS	32	8
WORKER0 TRAIN TIME	591S	584S
WORKER1 TRAIN TIME	1188S	870S
TEST TIME	0.81S	0.82S
TEST ACCURACY	0.84	0.84

3.3. STALE SYNCHRONOUS PARALLEL

In this approach, if any of the workers go ahead of other by certain number of steps, the gradient update is dropped for that worker. Number of steps is a hyper parameter and is known as staleness.

Experiment:

We have varied staleness for various values of 8 and 32. Figure 5 compares the plot of cross entropy training loss vs number of epochs for 2 workers and varying staleness.Also, plot has been shown only for 10 epochs to comment on convergence in observations. Also, test accuracy,test time and compares the training time of various workers has been reported in Table 4 for different staleness.

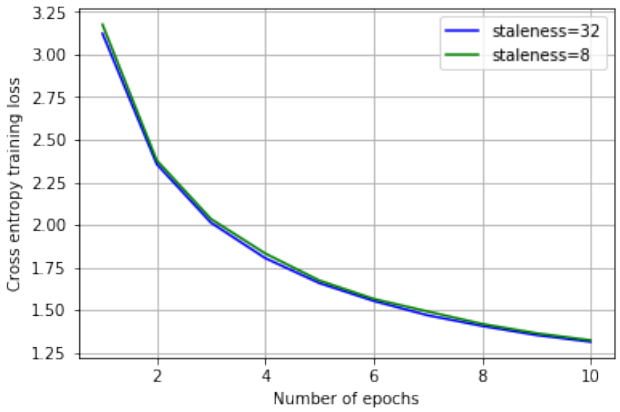

Figure 5. Cross entropy training loss vs Number of Epochs for SSP

Figure 6 compares the plot of cross entropy training loss vs number of epochs for various training modes discussed above.

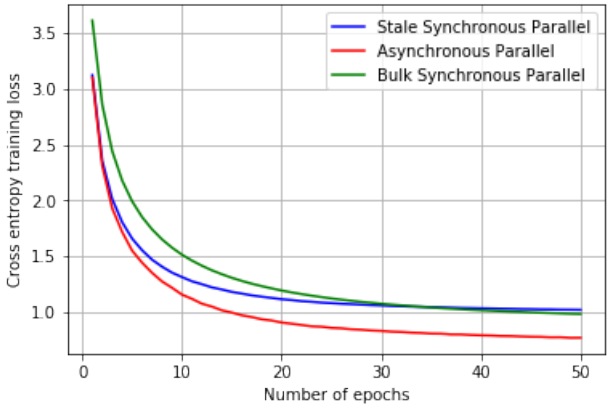

Figure 6. Cross entropy training loss vs Number of Epochs

3.4. OBSERVATTIONS

Following are the observations from the experiment.

- **Preprocessing**: Using stemmization and other preprocessing techniques, we were able to improve the test set accuracy from 74% to 84%.
- **Learning rate**: We observed that convergence rate of decaying learning rate is lower than constant learning rate. Also, training loss diverges after certain number of epochs in case of increasing learning rate.
- **Local training time vs Distributed training time**: We performed our local implementation on node 2 and distributed implementation on node 2 and node 3. We observed, node 3 was running very slow as compared to node 2. But when compared among time taken for node2, we observed distributed implementation to be 1.43 times faster than the local implementation for 100 epochs.
- **Number of workers in ASP mode**: We observed that convergence becomes faster with increase of number of workers as expected due to parallelization.
- **Varying staleness in SSP mode**: We observed that convergence with staleness of 32 marginally faster than staleness of 8 during the initial epochs. This might have happened because of the fact that gradient drops increases with decrease in staleness. Hence, there are less parameter updates corresponding to same number of epochs.
- **Comparison among ASP,BSP and SSP**: Since, there are highest number of parameter updates in ASP mode.

Hence, loss decreases at the fastest rate. In case of SSP, we drop the update as per the staleness factor specified. Hence, there are relatively lesser number of parameter updates. In case of BSP, we aggregate and then update and hence, has least number of parameter updates. Since, updates were not noisy in our case and hence, we achieved faster convergence with ASP. But, that may not be true in every case. Hence, SSP is the best mode of training. Also, we observed that worker 0 being relatively faster in other modes was slowed down in BSP mode to incorporate gradient aggregation.