# Document Classification using Stochastic Gradient Descent in Local and Distributed Settings

**Sahil Deep** [*1]

## Abstract

In this assignment, Logistic Regression was implemented in two settings i.e local using Python and distributed using TensorFlow framework. The classifiers were trained using Stochastic Gradient Descent, which was implemented in Python in local mode and using Tensorflow framework in distributed setting. Distributed implementation comprised of three implementations i.e. Bulk Synchronous Parallel(BSP), Asynchronous and Stale Synchronous.

## 1. Preprocessing on Dataset

- Each document was converted into words using $wrds = re.findall('w+', document)$ as tokenizer.
- Words were converted to lower case.
- No stopword removal or stemming was done.
- As a result, the vocabulary consisted of 301404 words(for full training data).

## 2. Local Implementation

In local implementation used Python to implement L2-regularized logistic regression. Total number of epochs 50 for all local implementations. Various functions used for the purpose are as follows:-

- Decreasing function used : $\frac{\eta}{t^2}$ initial learning rate $\eta = 1$ and $t$ is the number of epochs.
- Increasing function used : $\eta.(1.1)^t$ where $\eta = 0.001$ is initial learning rate and $t$ is number of epochs.
- Constant learning rate of $0.25$.

The results for cross entropy loss is plotted in figure 1. Table 1 gives various values corresponding to the chosen function. After shuffling training data. Parameters passed to the local implementation of the code are <Vocabulary size > < Initial learning rate> <Regularization Coefficient> < Total number of iterations > <Size of training dataset>. The values are mentioned in run.sh file. The following steps were used in the local implementation:-

*Table 1.* Accuracy and Time in Local Setting

| METHOD | TRG TIME(S) | TEST TIME(S) | TRG ACC | TEST ACC |
|---|---|---|---|---|
| DECREASING | 26481.2 | 14.4 | 80 | 73.5 |
| INCREASING | 26384.5 | 8.9 | 79.7 | 73.9 |
| CONSTANT | 26304.3 | 10.2 | 79.3 | 72.1 |

- Let $k = 0$ and let $A$ and $B$ be empty hashtables. $A$ will record the value of $K$ last time $B[j]$ was updated.
- For $t = 1, ...., T$
  - For each example $x_i, y_i$:
    - Let k=k+1
    - For each non-zero feature of $x_i$ with index $j$ and value $x_j$:
      - If $j$ is not in $B$, set $B[j] = 0$.
      - If $j$ is not in $A$, set $A[j] = 0$.
      - Simulate the regularization updates that would have been performed for the $k - A[j]$ examples since the last time a non-zero $x_j$ was encountered by setting
      $$B[j] = B[j](1 - 2\lambda\mu)^{k-A[j]}$$
      - Set $B[j] = B[j] + \lambda(y - p)x_j$
      - Set $A[j] = k$
- For each parameter $\beta_1, ...., \beta_d$, set
$$B[j] = B[j].(1 - 2\lambda\mu)^{k-A[j]}$$
- Output parameters $\beta_1, ....., \beta_d$.

## 3. Distributed Implementation using Parameter Server

Code was written in python and run on Turing clusters different worker nodes using Distributed Tensorflow frame-

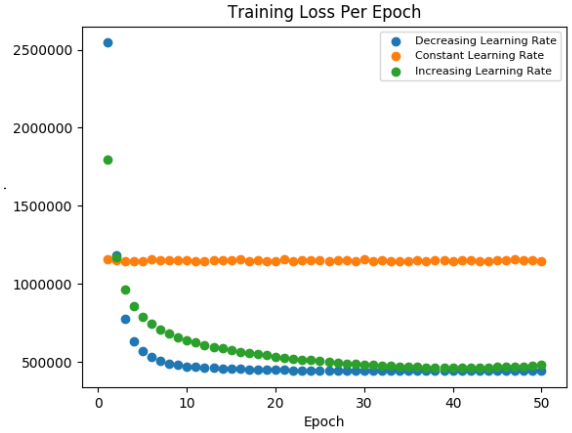


*Figure 1.*

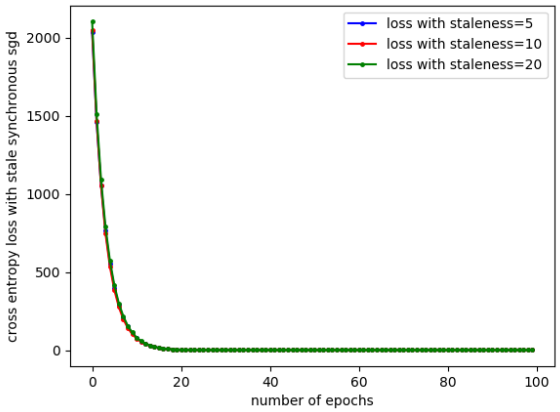


*Figure 2.*

*Table 2.* Accuracy and Time in Distributed Setting

| METHOD | TRG TIME | TEST TIME | TRG ACC | TEST ACC |
|---|---|---|---|---|
| BSP | 2399.8 | 0.91 | 75.6 | 72.5 |
| ASYNC | 1878.7 | 0.94 | 74.9 | 71.2 |
| SSP-5 | 1886.4 | 0.92 | 74.8 | 70.5 |
| SSP-10 | 1904.0 | 0.96 | 74.9 | 69.8 |
| SSP-20 | 1896.3 | 0.98 | 75.8 | 69.6 |

work. This framework was chosen as distributed graph learning can be done iteratively on Tensorflow synchronously and asynchronously. All the distributed training techniques follow the between- graph replication. This naturally employs that they are asynchronous and thus effort has been employed to synchronize them.

### 3.1. BSP

The Barrier Synchronization has been implementedby employing vector with one entry per user. If all the workers are in same iteration then proceeds with SGD else busy waits.This busy waiting involves reading the vector from the parameter server.

### 3.2. Stale Synchronous

This implementation also has a vector maintained in the parameter server. Each node verifies if its own iteration is off by the staleness factor from the minimum value in the vector. Depending upon the staleness factor it blocks or continues with its iteration. I have gone with staleness of $5, 10$ and $20$ respectively.

### 3.3. Asynchronous

It did not involve any barrier and hence each worker completed its iterations at different times and have no synchronization between themselves.

### 3.4. Plots

Various plots corresponding to the technique employed for BSP, SSP and Asynchronous setting are generated. Figure 3 gives the accuracy of various techniques employed as the number of epochs vary. Figure 4 gives the training loss of the three techniques employed.

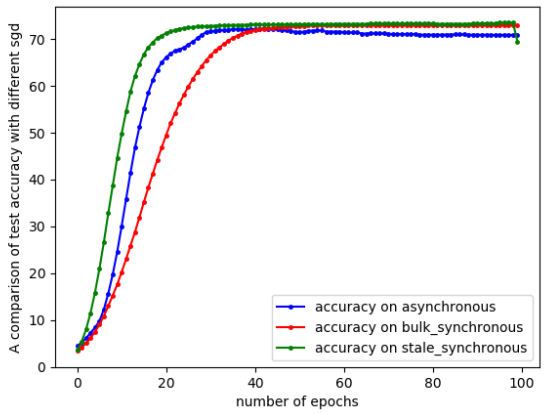


*Figure 3.*

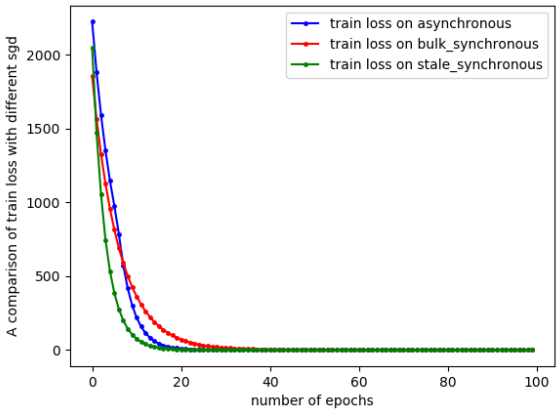


*Figure 4.*

## References

The following pdf http://www.cs.cmu.edu/~wcohen/10-405/assignments/hw3.pdf was used to implement SGD in local mode.
For Distributed implementation used the following reference and code was modified to suit our needs https://github.com/Jiankai-Sun/Distributed-TensorFlow-Example/tree/master/Logistic-Regression.