Improving Convergence Rates in Matrix Factorization when Non-Zero Entries are Skewed

Avinandan Bose
180165
Department of Computer Science Engineering,IIT Kanpur
Mixed Mode Term Paper for Convex Optimization and Signal Processing (EE609)

***Abstract*— Stochastic Gradient Descent (SGD) is a popular algorithm used in several machine learning tasks, and a lot of research work has been done to parallelize SGD without the need of locking, ie. processors will have access to shared memory with the possibility of overwriting each other's updates.We present a method of permuting the observed entries in the problem of matrix factorization when the observed non-zero entries are skewed (ie. not missing uniformly at random) so that we get faster convergence rates than existing techniques when using lockfree parallelized SGD.**

I. INTRODUCTION

In the problem of matrix factorization, we are given a large n $\times$ m matrix M, where only a fraction of the entries indexed by O_{obs} are observed, and we try to find a low rank approximation of M, by writing it as M = RC^T where R is an n $\times$ r matrix and C is an m $\times$ r matrix, where r is the small rank approximation.

Mathematically, we want to optimize $\sum_{(i,j):(i,j)\in O_{obs}}$ (M_{ij} - $(RC^T)_{ij}$)^2.We begin with a random guess for matrices R and C, and then use an algorithm like SGD to get the optimum values for R and C which minimize the objective.

It is worth noting that the observed entries of the matrix M, may not be uniformly distributed,ie.the number of observed entries in a particular row or column of M is not uniform.Let's take the example of movie ratings by different users to motivate this further.On any movie streaming platform,all movies never have the same number of ratings.Factors such as popularity,date it was first aired etc influence the number of ratings it gets.Similarly, not all users watch the same number of movies, and it depends on how active the user is,how long he/she is subscribed to this platform etc.Most often, a huge chunk of the observed entries is confined to only a few users and few movies, and this observation is key to our method's performance.

II. BACKGROUND

A. HOGWILD

HOGWILD [1] is a lock-free parallelized SGD algoritm, which works very well on sparse datasets.The following definitions will be used through the rest of the paper. **Pool** : A step where τ number of processors are updating parameters from shared memory, without any locking,ie. different processors may overwrite the updates to a subset of parameters.Having Locking in place means that at one point in time,only one processor can write to the shared memory, but locking causes a huge time overhead, and HOGWILD proposes to solve just that. **Collision** : The phenomenon when two or more processors,try to overwrite each other's updates at the same time. In a pool with more than one processor, collision is possible.

HOGWILD is well suited to cases, in which the gradient due to a single data-point is non-zero in only a small subset of parameters. For our case, the gradient for the data point M_{ij} has non zero components only for R_i (the ith row of R) and C_j (the jth row of C).These just constitute 2r parameters of a total of (n + m)r parameter space.If the entries are sparse (exact quantitative details in the next paragraph), the collisions in a pool are very low, and thus HOGWILD performs well even without locking.

Let $|O_{obs}|$ be the total number of entries in M.We have the following quantities then :

Δ_r = max$_i$ (no. of entries in ith row of M) / $|O_{obs}|$.

Δ_c = max$_j$ (no. of entries in jth column of M) / $|O_{obs}|$.

Δ = max (Δ_r,Δ_c)

ρ = Δ_r + Δ_c.

Δ signifies the maximum fraction of parameters that can collide in a pool. ρ signifies the sparsity of the given matrix.

Ω = 2r , is the number of parameters that a single data-point depends upon.

Our square function is Lipschitz continuous differentiabile with Lipschitz constant *L*.The square function is also strongly convex with modulus c. Also the norm of the partial derivative (for data-point M_{ij}, the derivate with respect to R_i and C_j is always bounded by a value *M*.

x^* = [R^*,C^*] collectively describe the model parameters which minimize the objective.

Let τ denote the number of processors in a pool. Then with a constant step size

$$\gamma = \frac{\nu\epsilon c}{2LM^2\Omega(1+6\rho\tau+4\tau^2\Omega\Delta^{0.5})} \tag{1}$$

for some $\epsilon > 0$ and $\nu \in (0,1)$.Define $D_0 := ||x^0 - x^*||^2$ (where x^0 = [R^0,C^0] is the initial random guess for our parameters) and let k be an integer satisfying

$$k \geq \frac{2LM^2(1+6\tau\rho+6\tau^2\Omega\Delta^{0.5})log(\frac{LD_0}{\epsilon})}{c^2\nu\epsilon} \tag{2}$$

Then after k component updates of x, we have $\mathbf{E}[f(x^k) - f(x^*)] \leq \epsilon$.

The term $\frac{k}{\tau}$ denotes the number of pools we need to run before reaching close to the optimum objective, and thus also indicates the total time taken by the algorithm (since each pool does same number of parameter updates,hence taking nearly same time). From the expression, it is clear that lower the values of Δ and ρ, the faster the algorithm is.As mentioned in the introduction section above about the non uniformity of the way the observed entries are distributed and the fact that a major chunk of the entries is confined to only a few rows and columns, by just separating these rows and columns, we should be able to get a significant drop in the values of ρ and Δ. The following sub-section is crucial as to what we are going to do with the separated rows and columns with much higher density.

B. SSGD

Stratified SGD (SSGD) [2], is a novel technique, which creates a stratum Z = Z^1 U Z^2 U ... U Z^d (Stratification), where each substratum Z^i contains data-points sharing no parameters.For the matrix M, choosing those indices (i,j), such that (i - j) = constant, and grouping them in a substratum Z^k, ensures that for any two points a and b in Z^k,$i_a \neq i_b$ and $j_a \neq j_b$.Thus if points taken from a substratum are fed to a pool, we are bound not to have any collisions at all.

However the problem is that since the given matrix M is quite sparse, we will often have substrata which have much lesser than τ (the number of processors) points.But since the time taken by a pool, is independent of the number of data-points actually fed into the pool (as all updates in different processors happen simultaneously, having 1 data-point or τ data-points doesn't affect the time of computations by the pool), we will end up needing way more than $\frac{|O_{obs}|}{\tau}$ number of pool runs, just to cover all data-point's once.

If however our matrix is fairly dense, we can almost always ensure that each substratum has sufficient entries to fill τ processors.In the previous subsection, we talked about separating out a small subset of rows and columns which is fairly dense, thus stratifying this small subset of rows and columns to avoid collisions, will mean the chances of collisions in a pool are very low, and also we don't have sparsely filled pools, thereby saving time on both fronts.

III. PROPOSED METHOD

Given matrix M,we re-permute it's rows and columns such that the subset of rows and columns that are dense and contain a major chunk of the total entries is clubbed together (specifically the lower right corner).

A. Algorithm : Re-permuting rows and columns

Algorithm 1: Re-permuting rows and columns

Input: Matrix M and O_{obs}

Output: re-permuted copies of the matrix M and O_{obs} according to increasing row-wise and column-wise counts

1 Ord_r = argsort (row-wise count of non-zero elements in O_{obs})

2 Ord_c = argsort (column-wise count of non-zero elements in O_{obs})

3 **return** M[Ord_r,Ord_c],O_{obs}[Ord_r,Ord_c]

B. Effect on ρ and Δ after the re-purmutation

We separate out the subset of last d rows and columns from re-permuted M into a d $\times$ d matrix A = M[n-d:n,m-d:m]. For the remaining matrix, we calculate the Δ and ρ values.

Δ_r^* = max ($\max_i$ (no. of entries in ith row of M[0:n-d-1,0:m] , $\max_i$ (no. of entries in ith row of M[n-d:n,0:m-d-1]) / $|O_{obs}|$.

Δ_c^* = max ($\max_i$ (no. of entries in ith column of M[0:n:0:m-d-1] , $\max_i$ (no. of entries in ith column of M[0:n-d-1:m-d:m]) / $|O_{obs}|$.

Δ^* = max (Δ_r^*,Δ_c^*).

ρ^* = Δ_r^* + Δ_c^*.

Now,we notice that the since the rows and columns are arranged in increasing number of row-wise and column-wise count, $\Delta_r^* \leq \Delta_r$ and $\Delta_c^* \leq \Delta_c$,and thus $\rho^* \leq \rho$ and $\Delta^* \leq \Delta$.

Since,we will build a stratum for the separated sub-matrix A and feed one substratum at a time into the pool thus entries from this region will be collision free when fed to a pool and the only collisions are captured by the computed values of Δ^* and ρ^*, thus the same values of Δ^* and ρ^* also hold for the entire matrix M.

IV. THEORETICAL ANALYSIS OF WHEN THE PROPOSED METHOD WILL OUTPERFORM SSGD AND HOGWILD

Both SSGD and our method are special cases of HOGWILD with better values of ρ and Δ. From the HOGWILD paper we have the result (also stated in Section II A), after k component updates of x, we have $\mathbf{E}[f(x^k) - f(x^*)] \leq \epsilon$.We will now keep the value of ϵ constant across all the three methods and proceed to calculate the value of k (which is a function of ρ and Δ and thus different for different methods) for these three methods, and then also compute the time taken by them to complete k updates, and the final comparison will be based on the time taken by them to reach $\mathbf{E}[f(x^k) - f(x^*)] \leq \epsilon$.

The following are valid assumptions we make for our calculations. One iteration of a pool takes t_0 time, and this is independent of how many cores are filled with data as all the updates happen simultaneously.For a particular method on average out of τ cores, say a fraction f is occupied, then the time to complete k updates is $\mathbf{t} = \frac{kt_0}{f\tau}$.The values of k and f will be different for different methods to achieve $\mathbf{E}[f(x^k) - f(x^*)] \leq \epsilon$ and keeping ϵ fixed for all of them, we will compare the time taken ($\mathbf{t}$) by each method.

Also for fairness the the initial guess x^0 of the parameters is assumed to same for all three methods.

A. Calculations for HOGWILD

As mentioned in Section II A, we use the lower value of k, ie

$$k = \frac{2LM^2(1 + 6\tau\rho + 6\tau^2\Omega\Delta^{0.5})log(\frac{LD_0}{\epsilon})}{c^2\nu\epsilon} \quad (3)$$

More specifically since most of the factors are constants across different methods, we use

$$k = \mathbf{a}(1 + 6\tau\rho + 6\tau^2\Omega\Delta^{0.5}) \quad (4)$$

where **a** is a constant. Since the fraction f is 1 for HOGWILD (we can always send τ data-points in every pool), the required time

$$\mathbf{t} = \frac{\mathbf{a}(1 + 6\tau\rho + 6\tau^2\Omega\Delta^{0.5})}{\tau} \quad (5)$$

B. Calculations for SSGD

Since there is no collision in SSGD updates, the values for ρ and Δ are 0 (actually they are $\frac{2}{|O_{obs}|}$ and $\frac{1}{|O_{obs}|}$ by definition, but these are so small that they can be taken as 0 for practical purposes). The fraction f_1 is a number between 0 and 1 for SSGD. Thus the time taken by SSGD is

$$\mathbf{t} = \frac{\mathbf{a}}{f_1\tau} \quad (6)$$

C. Calculations for Proposed Method

Say we have d data-points in total. d_1 of them are separated and will be stratified. Therefore the fraction $f_2 = \frac{gd_1+(d-d_1)}{d}$ or f_2 = (1 - (1 - g) $\frac{d_1}{d}$).Thus when g (which is the fraction of processors occupied on average when data-points are sampled from the smaller denser sub-matrix) is close to 1 (which ideally is, as the area being stratified is dense), the value of f_2 tends to 1.

The required time

$$\mathbf{t} = \frac{\mathbf{a}(1 + 6\tau\rho^* + 6\tau^2\Omega(\Delta^*)^{0.5})}{f_2\tau} \quad (7)$$

D. Comparisons

For our Method to outperform HOGWILD, we should have

$$\frac{(1 + 6\tau\rho^* + 6\tau^2\Omega(\Delta^*)^{0.5})}{f_2} \leq (1 + 6\tau\rho + 6\tau^2\Omega\Delta^{0.5}) \quad (8)$$

For our method to outperform SSGD, we mus have

$$\frac{(1 + 6\tau\rho^* + 6\tau^2\Omega(\Delta^*)^{0.5})}{f_2} \leq \frac{1}{f_1} \quad (9)$$

Ideally, whenever the values of ρ^* and Δ^* are close to 0, and the value of f_2 is close to 1, our method outperforms HOGWILD and SSGD. In the following sections, we demonstrate a wide range of cases where our algorithm performs better and by how much is the performance gain.

V. Experiments on Synthetic Datasets

We conducted experiments on various kinds of synthetic data-sets (where the original matrix M is a product of 2 low rank matrices R and C),and we study 2 important factors which decide the relevance of our work : 1.Overall density of non-zero entries of matrix M 2.Distribution of the pockets of high density.

A. Overall density of non-zero entries

For the 4 scenarios presented below, how the high density pockets were distributed is kept constant.In 5000 $\times$ 5000 dimensional matrices, there are two types of zones based on density of non-zero entries, one of high density (having density **a**) and one of low density (having density **b**), and the high density zones can be separated into a 1500 $\times$ 1500 dimensional sub-matrix. We vary **a** and **b** and present the results in Fig. 1 and Fig. 2.

We found that when the overall density is very high, SSGD and our method have almost similar convergence.When the density is very low, all methods have similar performance.In between these two extremes, our method has a significant performance gain.

B. Distribution of pockets of high density

Now we keep the values of **a** and **b** constant throughout, but we vary how the high density pockets are distributed. More specifically, we keep the number of rows and columns containing these high density pockets constant, but vary the dimensions of the minimum sub-matrix they can be rearranged into. Let **d** denote the dimensions of these **d** $\times$ **d** sub-matrix. The results are plotted in Fig. 3 and Fig. 4.

We see that the smaller the dimensions of the sub-matrix into which the dense zones can be rearranged, the better our algorithm outperforms the others.This is accounted by the fact, the smaller **d** is, higher will be the fraction f_2, since we will have lower number of subsets of non-colliding data-points (for same number of total data-points), and thus reducing the chances of having very sparsely populated pools.

VI. Experiments on Real Dataset

We tried and contrasted our algorithm with HOGWILD and SSGD on the MovieLens 100K dataset, which has a movie-user ratings matrix, and 100,000 ratings are available for 1682 movies and 943 users. The plots of Fig. 5 represent the number of ratings per movie and per user, and we observe that a very large number of ratings belong to only a few movies and a few users. We choose **d** = 400 to form the sub-matrix that will be stratified, and then run the three methods, and the results in Fig. 6 show that our method outperforms both HOGWILD and SSGD.

VII. Summary

References

[1] Feng Niu, Benjamin Recht, Christopher Ré and Stephen J. Wright Hogwild!: A Lock-Free Approach to Parallelizing Stochastic In Advances in neural information processing systems, pages 693–701, 2011 Gradient Descent

[2] Rainer Gemulla, Peter J.Haas, Yannis Sismanis, Christina Teflioudi, Faraz Makari Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.

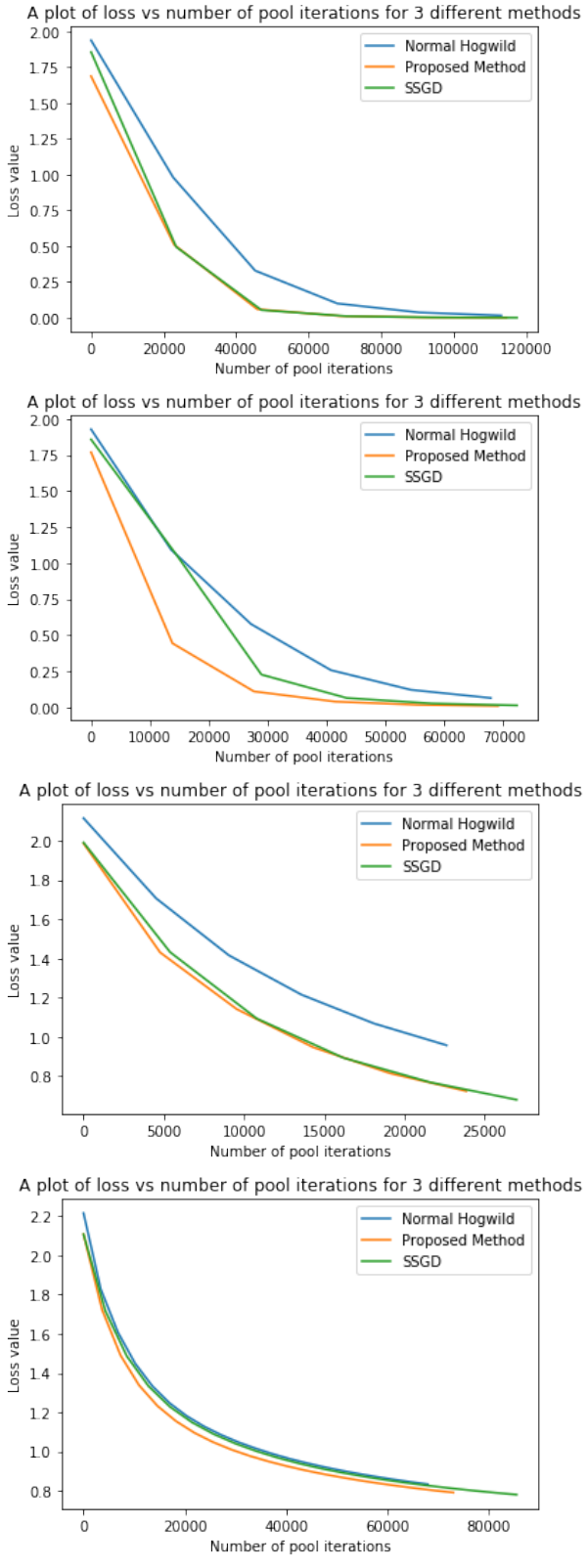

Fig. 1. Plot of loss vs number of pool iterations for the three different methods .The values for **a** and **b** in the plots are 1.**a** = 0.5 ,**b** = 0.05 2.**a** = 0.3,**b** = 0.03 3.**a** = 0.1,**b** = 0.01 2.**a** = 0.05,**b** = 0.01

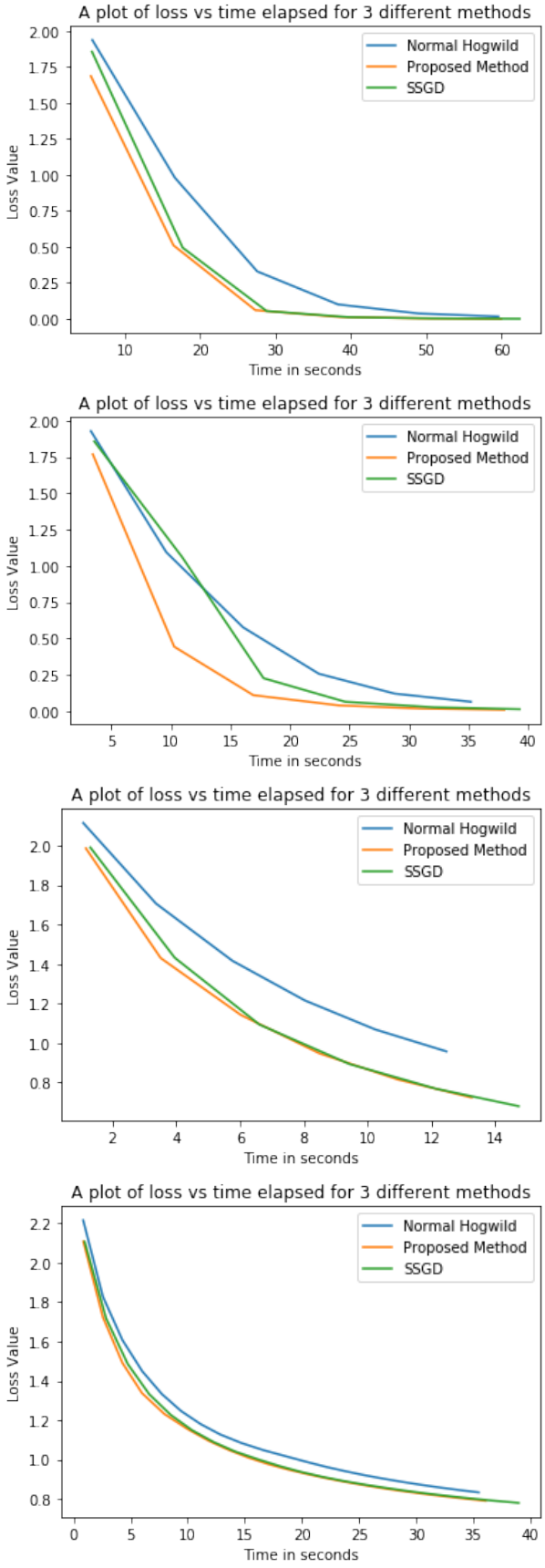

Fig. 2. Plot of loss vs time elapsed in seconds for the three different methods .The values for **a** and **b** in the plots are 1.**a** = 0.5 ,**b** = 0.05 2.**a** = 0.3,**b** = 0.03 3.**a** = 0.1,**b** = 0.01 2.**a** = 0.05,**b** = 0.01

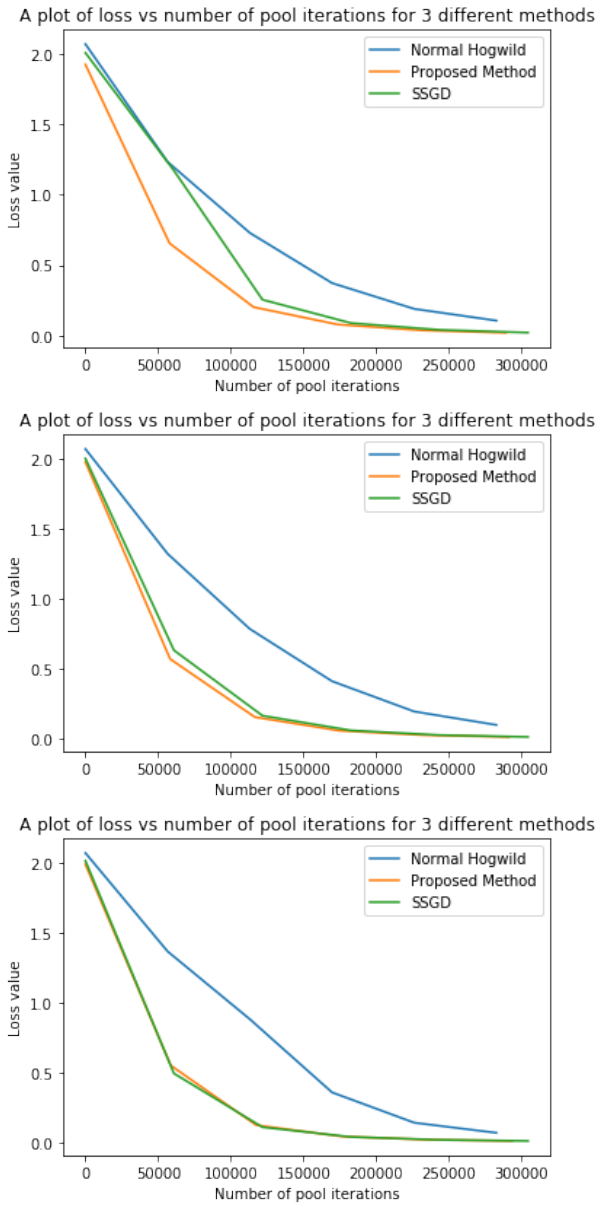

Fig. 3. Plot of loss vs number of pool iterations for the three different methods.The values for **d** are 1.**d** = 1500,2.**d** = 2000,3.**d** = 2500

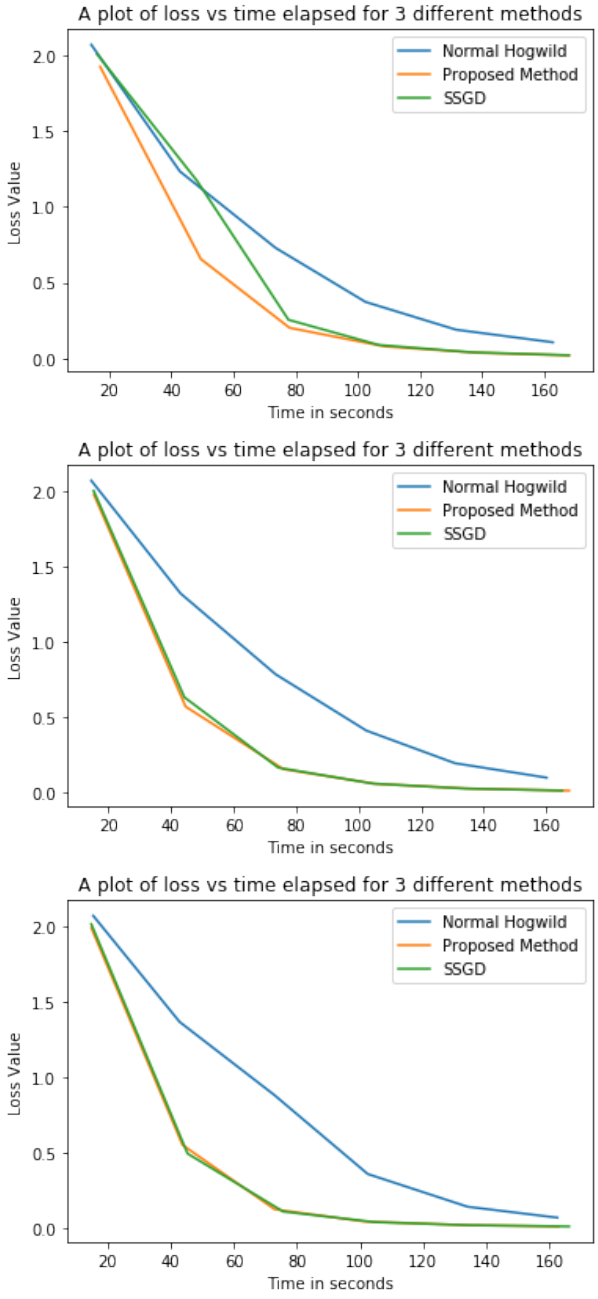

Fig. 4. Plot of loss vs time elapsed in seconds for the three different methods.The values for **d** are 1.**d** = 1500,2.**d** = 2000,3.**d** = 2500

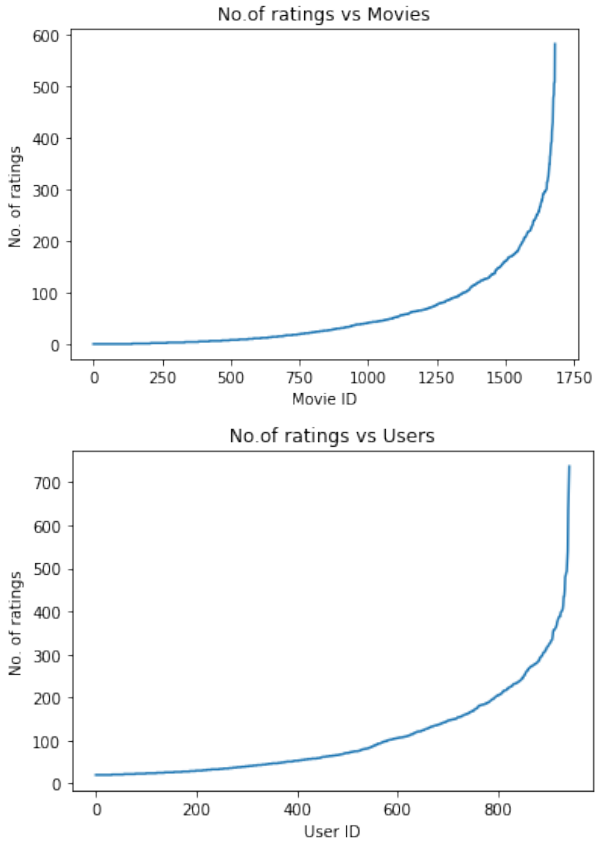

Fig. 5. Ratings Count Statistics, per Movie and per User.

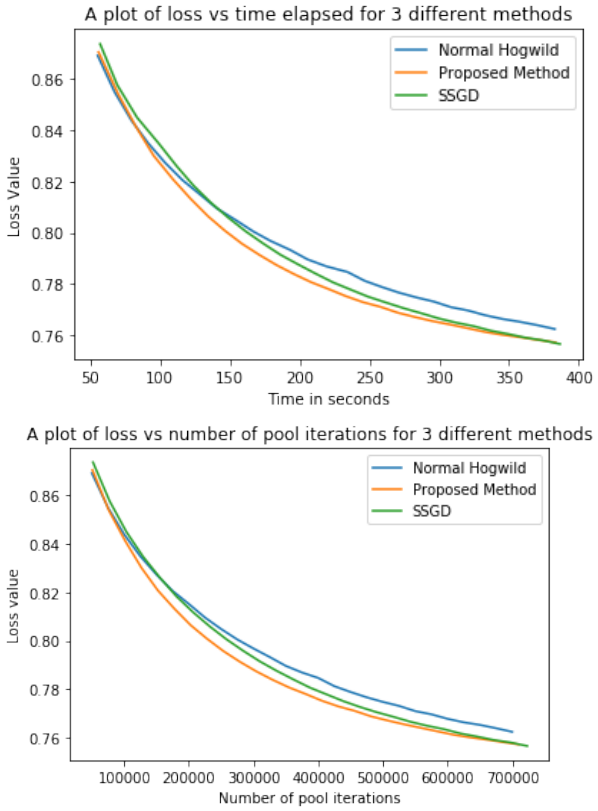

Fig. 6. Comparative performance of HOGWILD,our method and SSGD on MovieLens100K data-set.