

# Fast Coordinate Descent Methods with Variable Selection for Non-negative Matrix Factorization

Cho-Jui Hsieh  
Dept. of Computer Science  
University of Texas at Austin  
Austin, TX 78712-1188, USA  
cjhsieh@cs.utexas.edu

Inderjit S. Dhillon  
Dept. of Computer Science  
University of Texas at Austin  
Austin, TX 78712-1188, USA  
inderjit@cs.utexas.edu

## ABSTRACT

Nonnegative Matrix Factorization (NMF) is an effective dimension reduction method for non-negative dyadic data, and has proven to be useful in many areas, such as text mining, bioinformatics and image processing. NMF is usually formulated as a constrained non-convex optimization problem, and many algorithms have been developed for solving it. Recently, a coordinate descent method, called *FastHals* [3], has been proposed to solve least squares NMF and is regarded as one of the state-of-the-art techniques for the problem. In this paper, we first show that *FastHals* has an inefficiency in that it uses a cyclic coordinate descent scheme and thus, performs unneeded descent steps on unimportant variables. We then present a variable selection scheme that uses the gradient of the objective function to arrive at a new coordinate descent method. Our new method is considerably faster in practice and we show that it has theoretical convergence guarantees. Moreover when the solution is sparse, as is often the case in real applications, our new method benefits by selecting important variables to update more often, thus resulting in higher speed. As an example, on a text dataset RCV1, our method is 7 times faster than *FastHals*, and more than 15 times faster when the sparsity is increased by adding an L1 penalty. We also develop new coordinate descent methods when error in NMF is measured by KL-divergence by applying the Newton method to solve the one-variable sub-problems. Experiments indicate that our algorithm for minimizing the KL-divergence is faster than the Lee & Seung multiplicative rule by a factor of 10 on the CBCL image dataset.

## Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning; G.1.6 [Numerical Analysis]: Optimization

## General Terms

Algorithms, Performance, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'11, August 21–24, 2011, San Diego, California, USA.  
Copyright 2011 ACM 978-1-4503-0813-7/11/08 ...\$10.00.

## 1. INTRODUCTION

Non-negative matrix factorization (NMF) ([20, 14]) is a popular matrix decomposition method for finding non-negative representations of data. NMF has proved useful in dimension reduction of image, text, and signal data. Given a matrix  $V \in \mathbb{R}^{m \times n}$ ,  $V \geq 0$ , and a specified positive integer  $k$ , NMF seeks to approximate  $V$  by the product of two non-negative matrices  $W \in \mathbb{R}^{m \times k}$  and  $H \in \mathbb{R}^{k \times n}$ . Suppose each column of  $V$  is an input data vector, the main idea behind NMF is to approximate these input vectors by nonnegative linear combinations of nonnegative “basis” vectors (columns of  $W$ ) with the coefficients stored in columns of  $H$ .

The distance between  $V$  and  $WH$  can be measured by various distortion functions — the most commonly used is the Frobenius norm, which leads to the following minimization problem:

$$\min_{W, H \geq 0} f(W, H) \equiv \frac{1}{2} \|V - WH\|_F^2 = \frac{1}{2} \sum_{i,j} (V_{ij} - (WH)_{ij})^2 \quad (1)$$

To achieve better sparsity, researchers ([7, 22]) have proposed adding regularization terms, on  $W$  and  $H$ , to (1). For example, an L1-norm penalty on  $W$  and  $H$  can achieve a more sparse solution:

$$\min_{W, H \geq 0} \frac{1}{2} \|V - WH\|_F^2 + \rho_1 \sum_{i,r} W_{ir} + \rho_2 \sum_{r,j} H_{rj}. \quad (2)$$

One can also replace  $\sum_{i,r} W_{ir}$  and  $\sum_{r,j} H_{rj}$  by Frobenius norm of  $W$  and  $H$ . Note that usually  $m \gg k$  and  $n \gg k$ . Since  $f$  is non-convex, we can only hope to find a stationary point of  $f$ . Many algorithms ([15, 5, 1, 23]) have been proposed for this purpose. Among them, Lee & Seung’s multiplicative-update algorithm [15] has been the most common method. Recent methods follow the Alternating Nonnegative Least Squares (ANLS) framework suggested by [20]. This framework has been proved to converge to stationary points by [6] as long as one can solve each nonnegative least square sub-problem exactly. Among existing methods, [18] proposes a projected gradient method, [11] uses an active set method, [10] applies a projected Newton method, and [12] suggests a modified active set method called *Block-Pivot* for solving the non-negative least squares sub-problem. Very recently, [19] developed a parallel NMF solver for large-scale web data. This shows a practical need for further scaling up NMF solvers, especially when the matrix  $V$  is sparse, as is the case for text data or web recommendation data.

Coordinate descent methods, which update one variable at a time, are efficient at finding a reasonable solution efficiently, a property that has made these methods successful in large scale classification/regression problems. Recently,

a coordinate descent method, called **FastHals** [3], has been proposed to solve the least squares NMF problem (1). Compared to methods using the ANLS framework which spend significant time in finding an exact solution for each sub-problem, a coordinate descent method can efficiently give a reasonably good solution for each sub-problem and switch to the next round. Despite being a state-of-the-art method, **FastHals** has an inefficiency in that it uses a cyclic coordinate descent scheme and thus, may perform unneeded descent steps on unimportant variables. In this paper, we present a variable selection scheme that uses the gradient of the objective function to arrive at a new coordinate descent method. Our method is considerably faster in practice and we show that it has theoretical convergence guarantees, which was not addressed in [3]. We conduct experiments on large sparse datasets, and show that our method generally performs 2-3 times faster than **FastHals**. Moreover when the solution is sparse, as is often the case in real applications, our new method benefits by selecting important variables to update more often, thus resulting in higher speed. As an example, on a text dataset RCV1, our method is 7 times faster than **FastHals**, and more than 15 times faster when the sparsity is increased by adding an L1 penalty.

Another popular distortion measure to minimize is the KL-divergence between  $V$  and  $WH$ :

$$\min_{W, H \geq 0} L(W, H) = \sum_{i,j} V_{ij} \log\left(\frac{V_{ij}}{(WH)_{ij}}\right) - V_{ij} + (WH)_{ij} \quad (3)$$

The above problem has been shown to be connected to Probabilistic Latent Semantic Analysis (PLSA) [4]. As we will point out in Section 3, problem (3) is more complicated than (1) and algorithms in the ANLS framework cannot be easily extended to solve (3). For this problem, [3] proposes to solve by a sequence of local loss functions — however, a drawback of the method in [3] is that it will not converge to a stationary point of (3). In this paper we propose a cyclic coordinate descent method for solving (3). Our new algorithm applies the Newton method to get a better solution for each variable, which is better compared to working on the auxiliary function as used in the multiplicative algorithm of [15]. As a result, our cyclic coordinate descent method converges much faster than the multiplicative algorithm. We further provide theoretical convergence guarantees for cyclic coordinate descent methods under certain conditions.

The paper is organized as follows. In Section 2, we describe our coordinate descent method for least squares NMF (1). The modification for the coordinate descent method for solving KL-NMF (3) is proposed in Section 3. Section 4 discusses the relationship between our methods and other optimization techniques for NMF, and Section 5 points out some important implementation issues. The experiment results are given in Section 6. The results show that our methods are more efficient than other state-of-art methods.

## 2. COORDINATE DESCENT METHOD FOR LEAST SQUARES NMF

### 2.1 Basic update rule for one element

The coordinate descent method updates one variable at a time until convergence. We first show that a closed form update can be performed in  $O(1)$  time for least squares NMF if certain quantities have been pre-computed. Note that in this section we focus on the update rule for variables in  $W$  — the update rule for variables in  $H$  can be derived similarly.

Coordinate descent methods aim to conduct the following one variable updates:

$$(W, H) \leftarrow (W + sE_{ir}, H)$$

where  $E_{ir}$  is a  $m \times k$  matrix with all elements zero except the  $(i, r)$  element which equals one. Coordinate descent solves the following one-variable subproblem of (1) to get  $s$ :

$$\min_{s: W_{ir} + s \geq 0} g_{ir}^W(s) \equiv f(W + sE_{ir}, H). \quad (4)$$

We can rewrite  $g_{ir}^W(s)$  as

$$\begin{aligned} g_{ir}^W(s) &= \frac{1}{2} \sum_j (V_{ij} - (WH)_{ij} - sH_{rj})^2 \\ &= g_{ir}^W(0) + (g_{ir}^W)'(0)s + \frac{1}{2} (g_{ir}^W)''(0)s^2. \end{aligned} \quad (5)$$

For convenience we define

$$G^W \equiv \nabla_W f(W, H) = WHH^T - VH^T. \quad (6)$$

$$\text{and } G^H \equiv \nabla_H f(W, H) = W^TWH - W^TV. \quad (7)$$

Then we have

$$\begin{aligned} (g_{ir}^W)'(0) &= (G^W)_{ir} = (WHH^T - VH^T)_{ir} \\ \text{and } (g_{ir}^W)''(0) &= (HH^T)_{rr}. \end{aligned}$$

Since (5) is a one-variable quadratic function with the constraint  $W_{ir} + s \geq 0$ , we can solve it in closed form:

$$s^* = \max\left(0, W_{ir} - (WHH^T - VH^T)_{ir} / (HH^T)_{rr}\right) - W_{ir}. \quad (8)$$

Coordinate descent algorithms then update  $W_{ir}$  to

$$W_{ir} \leftarrow W_{ir} + s^*. \quad (9)$$

The above one variable update rule can be easily extended to NMF problems with regularization as long as the gradient and second order derivative of sub-problems can be easily computed. For instance, with the L1 penalty on both  $W$  and  $H$  as in (2), the one variable sub-problem is

$$\bar{g}_{ir}^W(s) = \left( (WHH^T - VH^T)_{ir} + \rho_1 \right) s + \frac{1}{2} (HH^T)_{rr} s^2 + \bar{g}_{ir}^W(0),$$

and thus the one variable update becomes

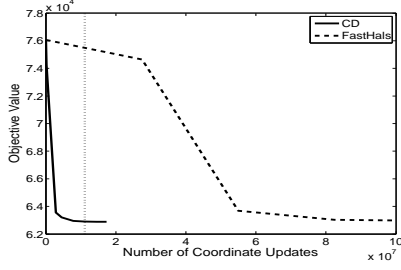
$$W_{ir} \leftarrow \max\left(0, W_{ir} - ((WHH^T - VH^T)_{ir} + \rho_1) / (HH^T)_{rr}\right).$$

Compared to (8), the new  $W_{ir}$  is shifted left to zero by the factor  $\frac{\rho_1}{(HH^T)_{rr}}$ , so the resulting  $W_{ir}$  generally has more zeros.

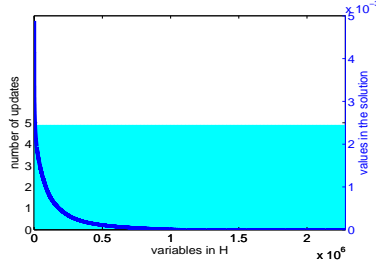
A state-of-the-art cyclic coordinate descent method called **FastHals** is proposed in [3], which uses the above one variable update rules.

### 2.2 Variable selection strategy

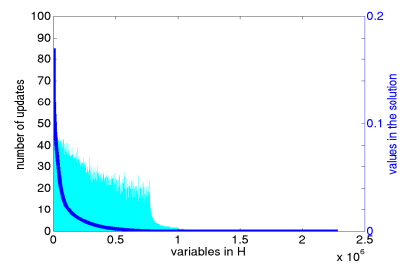
With the one variable update rule, **FastHals** [3] conducts a cyclic coordinate descent. It first updates all variables in  $W$  in cyclic order, and then updates variables in  $H$ , and so on. Thus the number of updates performed for each variable is exactly the same. However, an efficient coordinate descent method should update variables with frequency proportional to their “importance”. In this paper, we exhibit a carefully chosen variable selection scheme (with not too much overhead) to obtain a new coordinate descent algorithm that has the ability to select variables according to their “importance”, thus leading to significant speedups. Similar strategies has been employed in Lasso or feature selection problems [17, 21]. To illustrate the idea, we take a text dataset



(a) Coordinate updates versus objective value



(b) The behavior of FastHals



(c) The behavior of GCD

Figure 1: Illustration of our variable selection scheme. Figure 1(a) shows that our method GCD reduces the objective value more quickly than **FastHals**. With the same number of coordinate updates (as specified by the vertical dotted line in Figure 1(a)), we further compare the distribution of their coordinate updates. In Figure 1(b) and 1(c), the X-axis is the variables of  $H$  listed by descending order of their final values. The solid line gives their final values, and the light blue bars indicate the number of times they are chosen. The figures indicate that **FastHals** updates all variables uniformly, while the number of updates for GCD is proportional to their final values, which helps GCD to converge faster.

RCV1 with L1 penalty as an example. In Figures 1(b) and 1(c) the variables of the final solution  $H$  are listed on the X-axis — note that the solution is sparse as most of the variables are 0. Figure 1(b) shows the behavior of **FastHals**, which clearly shows that each variable is chosen uniformly. In contrast, as shown in Figure 1(c), by applying our new coordinate descent method, the number of updates for the variable is roughly proportional to their final values. For most variables with final value 0, our algorithm will never pick them to update. Therefore our new method focuses on nonzero variables and reduces the objective value more efficiently. Figure 1(a) shows that we can attain a 10-fold speedup by applying our variable selection scheme.

We now show that with an appropriate data structure, we can select variables that lead to maximum decrease of objective function in  $O(k + \log m)$  time with a pre-computed gradient. However,  $O(\log m)$  can be expensive in practice, so we further provide an alternate efficient row-based variable selection scheme that needs  $O(k)$  time per update.

Before presenting our variable selection method, we first introduce our framework. Similar to **FastHals**, our algorithm switches between  $W$  and  $H$  in the outer iterations:

$$(W^0, H^0) \rightarrow (W^1, H^0) \rightarrow (W^1, H^1) \rightarrow \dots \quad (10)$$

Between each outer iteration are the following inner updates:

$$(W^i, H^i) \rightarrow (W^{i,1}, H^i) \rightarrow (W^{i,2}, H^i) \dots \quad (11)$$

Later we will discuss the reason why we have to focus on  $W$  or  $H$  for a sequence of updates.

Suppose we want to choose variables to update in  $W$ . If  $W_{ir}$  is selected, as discussed in Section 2.1, the optimal update will be (8), and the function value will be decreased by

$$D_{ir}^W \equiv g_{ir}^W(0) - g_{ir}^W(s^*) = -G_{ir}^W s^* - \frac{1}{2}(HH^T)_{rr}(s^*)^2. \quad (12)$$

$D_{ir}^W$  measures how much we can reduce the objective value by choosing the coordinate  $W_{ir}$ . Therefore, if  $D^W$  can be maintained, we can greedily choose variables according to it. If we have  $G^W$ , we can compute  $s^*$  by (8) and compute  $D^W$  by (12), and so an element of  $D^W$  can be computed in  $O(1)$  time. At the beginning of a sequence of  $W$ 's updates, we can precompute  $G^W$ . The details will be provided in Section 2.3. Now assume we already have  $G^W$ , and  $W_{ir}$  is updated to  $W_{ir} + s^*$ . Then  $G^W$  will remain the same except for the  $i$ th row, which will be replaced by

$$G_{ij}^W \leftarrow G_{ij}^W + s^*(HH^T)_{rj} \quad \forall j = 1, \dots, k. \quad (13)$$

Using (13), we can maintain  $G^W$  in  $O(k)$  time after each variable update of  $W$ . Thus  $D^W$  can also be maintained in  $O(k)$  time.

However, maintaining  $G^H$  at each inner update of  $W$  is more expensive. From (7), when each element of  $W$  is changed, the whole matrix  $G^H$  will be changed, thus every element of  $D^H$  may also change. So the time cost for maintaining  $D^H$  is at least  $O(kn)$ , which is too much compared to  $O(k)$  for maintaining  $D^W$ . This is the reason that we follow the alternative minimization scheme and restrict ourselves to either  $W$  or  $H$  for a sequence of inner updates.

After the  $i$ -th row of  $G^W$  is updated, we can immediately maintain the  $i$ -th row of  $D^W$  by (12). To select the next variable-to-update, we want to select the index  $(i^*, r^*)$  that satisfies  $(i^*, r^*) = \arg \max_{i,r} D_{ir}^W$ . However, a brute force search through the whole matrix  $D^W$  will require  $O(mk)$  time. To overcome this, we can store the largest value  $v_i$  and index  $q_i$  for each row of  $D^W$ , i.e.,

$$q_i = \arg \max_j D_{ij}^W, \quad v_i = D_{i,q_i}^W. \quad (14)$$

As in (13), when  $W_{ir}$  is updated, only the  $i$ th row of  $G^W$  and  $D^W$  will change. Therefore the vector  $\mathbf{q}$  will remain the same except for one component  $q_i$ . Since the  $i$ th row of  $D^W$  contains  $k$  elements and each element can be computed in constant time, we can recalculate  $q_i$  in  $O(k)$  time. Each time we only change the largest value in  $\{q_i \mid i = 1, \dots, m\}$ , therefore we can store these values using a heap data structure so that each retrieval and re-calculation of the largest value can be done in  $O(\log m)$  time. This way the total cost for one update will be  $O(k + \log m)$ .

A stopping condition is needed for a sequence of updates. At the beginning of updates to  $W$ , we can store

$$p^{\text{init}} = \max_{i,j} D_{ij}^W. \quad (15)$$

Our algorithm then iteratively chooses variables to update until the following stopping condition is met:

$$\max_{i,j} D_{ij}^W < \epsilon p^{\text{init}}, \quad (16)$$

where  $\epsilon$  is a small positive constant. Note that (16) will be satisfied in a finite number of iterations as  $f(W, H)$  is lower bounded, and so the minimum for  $f(W, H)$  with fixed  $H$  is achievable. A small  $\epsilon$  value indicates each sub-problem is

solved to high accuracy, while a larger  $\epsilon$  value means our algorithm switches more often between  $W$  and  $H$ . We choose  $\epsilon = 0.001$  in our experiments.

In practice, this method can work when  $k$  is large. However, when  $k \ll m$ , the  $\log m$  term in the update cost will dominate. Moreover, each heap operation costs more time than a simple floating point operation. Thus we further design a more efficient row-based variable selection strategy.

First, we have an observation that when one  $W_{ir}$  is updated, only the  $i$ th row of  $D^W$  will be changed. Therefore if we update variables in the  $i$ th row and there is a variable in the  $j$ th row with larger potential decrease  $D_{jr}^W$ , we will not change  $D_{jr}^W$ . Choosing the largest  $D^W$  value in one row costs  $O(k)$ , which is cheaper than  $O(\log m)$  when  $k$  small. Therefore we can iteratively update variables in the  $i$ th row (that lead to maximum decrease in objective function) until the inner stopping condition is met:

$$\max_j D_{ij}^W < \epsilon p^{init}. \quad (17)$$

Our algorithm then update variables in the  $(i+1)$ -st row, and so on. Since changes in other rows will not affect the  $D^W$  values in the  $i$ th row, after our algorithm sweeps through row 1 to row  $m$ , (17) will be met for each row, thus the stopping condition (16) will also be met for the whole  $W$ .

### 2.3 Time complexity analysis

---

**Algorithm 1** GCD for least squares NMF

---

- Given:  $V, k, \epsilon$  (typically,  $\epsilon = 0.001$ )
  - Output:  $W, H$
  - Compute  $P^{VH} = VH^T$ ,  $P^{HH} = HH^T$ ,  $P^{WV} = W^T V$ ,  $P^{WW} = W^T W$
  - Initialize  $H^{new} \leftarrow 0$
  - While (not converged)
    1. Compute  $P^{VH} \leftarrow P^{VH} + V(H^{new})^T$  according to the sparsity of  $H^{new}$
    2.  $W^{new} \leftarrow 0$
    3.  $G^W \leftarrow W P^{HH} - P^{VH}$
    4.  $S_{ir}^W \leftarrow \max(W_{ir} - \frac{G_{ir}^W}{P_{rr}^{HH}}, 0) - W_{ir}$  for all  $i, r$ .
    5.  $D_{ir}^W \leftarrow -G_{ir}^W S_{ir}^W - \frac{1}{2} P_{rr}^{HH} (S_{ir}^W)^2$  for all  $i, r$ .
    6.  $q_i \leftarrow \arg \max_j D_{ij}^W$  for all  $i = 1, \dots, m$ , and  $p^{init} \leftarrow \max_i D_{i, q_i}^W$
    7. For  $i = 1, 2, \dots, m$ 
      - While  $D_{i, q_i}^W < \epsilon p^{init}$ 
        - 7.1.  $s^* \leftarrow S_{i, q_i}^W$
        - 7.2.  $P_{q_i, :}^{WW} \leftarrow P_{q_i, :}^{WW} + s^* W_{q_i, :}$ . (Also do a symmetric update for  $P_{:, q_i}^{WW}$ )
        - 7.3.  $W_{i, q_i}^{new} \leftarrow W_{i, q_i}^{new} + s^*$
        - 7.4.  $G_{i, :}^W \leftarrow G_{i, :}^W + s^* P_{q_i, :}^{HH}$
        - 7.5.  $S_{ir}^W \leftarrow \max(W_{ir} - \frac{G_{ir}^W}{P_{rr}^{HH}}, 0) - W_{ir}$  for all  $r = 1, \dots, k$ .
        - 7.6.  $D_{ir}^W \leftarrow -G_{ir}^W S_{ir}^W - \frac{1}{2} P_{rr}^{HH} (S_{ir}^W)^2$  for all  $r = 1, \dots, k$ .
        - 7.7.  $q_i \leftarrow \arg \max_j D_{ij}^W$ .
    8.  $W \leftarrow W + W^{new}$
    9. For updates to  $H$ , repeats analogues steps to Step 1 through Step 8.
- 

Our coordinate descent method with variable selection strategy can be summarized in Algorithm 1. We call our new coordinate descent algorithm GCD– Greedy Coordinate

Table 1: Time complexity analysis for Algorithm 1. We focus on time cost for one complete update of  $W$ . Here  $t$  is average number of inner updates, and  $s$  is number of nonzeros in  $V$ .

	dense $V$	sparse $V$
Compute matrix $VH^T$ (Step 1)	$\min(O(mt), O(nmk))$	$\min(O(\frac{st}{n}), O(sk))$
Initialize gradient/decreasing matrix (Steps 3 to 6)	$O(mk^2)$	$O(mk^2)$
Update inner product (Steps 7.2)	$O(tk)$	$O(tk)$
Update gradient and decreasing matrix (Step 7.4 to 7.7)	$O(tk)$	$O(tk)$

Descent since we take a greedy step of maximum decrease in objective function. The one-variable update and variable selection strategy are in the inner loop (step 7) of Algorithm 1. Before the inner loop, we have to initialize the gradient  $G^W$  and objective function decrease matrix  $D^W$ .  $G^W$  in (6) has two terms  $WHH^T$  and  $VH^T$  (in Algorithm 1, we use  $P^{VH}$ ,  $P^{HH}$ ,  $P^{WV}$  and  $P^{WW}$  to store matrices  $VH^T$ ,  $HH^T$ ,  $W^T V$  and  $W^T W$ ). For the first term, since  $HH^T$  is maintained in an analogous way to step 7.2), computing  $W(HH^T)$  costs  $O(mk^2)$  time. For the second term  $VH^T$ , we can also store the updates in  $H^{new}$  so that  $H^{new} = H^{i+1} - H^i$ , and then updates  $VH^T$  by step 1. Assume that, on average, step 7 has  $t$  coordinate updates, then the number of nonzeros in  $H^{new}$  is at most  $t$ . With sparse matrix multiplication, step 1 costs  $O(mt)$ . However, when  $H^{new}$  is dense, we should use a dense matrix multiplication for step 1, which costs  $O(nmk)$  and is independent of  $t$ .

Table 1 summarizes the cost of one outer iteration (step 1 to 9). To get the amortized cost per coordinate update, we can divide the numbers by  $t$ . We first consider when  $V$  is dense. When  $t < k^2$ ,  $O(mk^2)$  will dominate the complexity. When  $k^2 \leq t \leq nk$ , computational time for computing  $VH^T$  dominates the complexity and the time cost per coordinate update is  $O(m)$ . In this case, time cost per update is the same as **FastHals**. When  $t$  is larger than  $nk$ , the time cost per update will decrease. We summarize the amortized cost per update as below:

$$\begin{cases} O(\frac{mk^2}{t}) & \text{if } k^2 > t \\ O(m) & \text{if } nk > t \geq k^2 \\ O(\frac{nmk}{t}) & \text{if } nm > t \geq nk \\ O(k) & \text{if } t > nm \end{cases}$$

When  $V$  is sparse, assume there are  $s$  nonzero elements in  $V$ , the complexity for the computing  $VH^T$  is modified while others remain the same.

### 3. COORDINATE DESCENT METHOD FOR NMF WITH KL-DIVERGENCE

To apply coordinate descent for solving NMF with KL-divergence (3), we consider the one-variable sub-problem:

$$\begin{aligned} h_{ir}(s) &= L(W + sE_{ir}, H) \\ &= \sum_{j=1}^n -V_{ij} \log \left( (WH)_{ij} + sH_{rj} \right) + sH_{rj} + \text{constant}. \end{aligned} \quad (18)$$

Here we discuss the updates to  $W$ , the update rules for  $H$  can be derived similarly. Unlike the one-variable sub-problem of least squares NMF (5), minimizing  $h_{ir}(s)$  has no closed form solution. Thus **FastHals** fails to derive a coordi-

nate descent method for KL-divergence. Instead of solving the one-variable sub-problem  $h_{ir}(s)$ , **FastHals** in [3] minimizes the following one-variable problem for each update:

$$\bar{h}_{ir}(s) = \sum_j -(V_{ij} - \sum_{t \neq r} W_{it} H_{tj}) \log(s H_{rj}) + s H_{rj}.$$

The above problem has a closed form solution, but the solution is different from the one-variable sub-problem (18), thus **FastHals** solves a different problem and may converge to a different final solution. Therefore we can say that application of coordinate descent to solve NMF with KL-divergence has not been studied before.

In this section, we propose the use of Newton's method to solve the sub-problems. Since  $h_{ir}$  is twice differentiable, we can iteratively update  $s$  by Newton direction:

$$s \leftarrow \max(-W_{ir}, s - h'_{ir}(s)/h''_{ir}(s)), \quad (19)$$

where the first term comes from the non-negativity constraint. The first derivative  $h'_{ir}(s)$  and the second derivative  $h''_{ir}(s)$  can be written as

$$h'_{ir}(s) = \sum_{j=1}^n H_{rj} \left( 1 - \frac{V_{ij}}{(WH)_{ij} + s H_{rj}} \right). \quad (20)$$

$$h''_{ir}(s) = \sum_{j=1}^n \frac{V_{ij} H_{rj}^2}{((WH)_{ij} + s H_{rj})^2}. \quad (21)$$

For KL-divergence, we need to take care of the behavior when  $V_{ij} = 0$  or  $(WH)_{ij} = 0$ . For  $V_{ij} = 0$ , by analysing the asymptotic behavior,  $V_{ij} \log((WH)_{ij}) = 0$  for all positive values of  $(WH)_{ij}$ , thus we can ignore those entries. Also, we do not consider the case that one row of  $V$  is entirely zero, which can be removed by preprocessing. When  $(WH)_{ij} + s H_{rj} = 0$  for some  $j$ , the Newton direction  $-h'_{ir}(s)/h''_{ir}(s)$  should be infinity. In this situation we reset  $s$  so that  $W_{ir} + s$  is a small positive value and restart the Newton method. When  $H_{rj} = 0$  for all  $j = 1, \dots, n$ , the second derivative (21) is zero. In this case  $h_{ir}(s)$  is constant, thus we do not need to change  $W_{ir}$ .

To execute Newton updates, each evaluation of  $h'_{ir}(s)$  and  $h''_{ir}(s)$  takes  $O(n)$  time for the summation. For general functions, we often need a line search procedure to check sufficient decrease from each Newton iteration. However, as we now deal with the special function (18), we prove the following theorem to show that Newton method without line search converges to the optimum of each sub-problem:

**Theorem 1**

If a function  $f(x)$  with domain  $x \geq 0$  can be written in the following form

$$f(x) = -c_i \sum_{i=1}^l \log(a_i + b_i x) + \sum_j b_j x,$$

where  $a_i > 0, b_i, c_i \geq 0 \forall i$ , then the Newton method without line search converges to the global minimum of  $f(x)$ .

The proof based on the strict convexity of log function is in the Appendix of our technical report [9]. By the above Theorem, we can iteratively apply (19) until convergence. In practice, we design the following stopping condition for Newton method:

$$|s^{t+1} - s^t| < \epsilon |W_{ir} + s^t| \text{ for some } \epsilon > 0,$$

where  $s^t$  is the current solution and  $s^{t+1}$  is obtained by (19).

As discussed earlier, variable selection is an important issue for coordinate descent methods. After each inner update

---

**Algorithm 2** CCD for NMF with KL-divergence

---

1. Given:  $V, k, W, H, \epsilon$  (typically,  $\epsilon = 0.5$ )
  2. Output:  $W, H$
  3.  $P^{WH} \leftarrow WH$ .
  4. While (not converged)
    - 4.0.1. For  $i = 1, \dots, m$  (updates in  $W$ )
      - For  $r = 1, \dots, k$ 
        - While 1
          - \* Compute  $s$  by (19).
          - \*  $w^{\text{old}} = W_{ir}$ .
          - \*  $W_{ir} \leftarrow W_{ir} + s$ .
          - \* Maintain  $(WH)_{i, \cdot}$  by (22).
          - \* If  $|s| < \epsilon w^{\text{old}}$ , Break
    - 4.0.2. For updates to  $H$ , repeats steps analogous to Step 4.0.1
- 

$W_{ir} \leftarrow W_{ir} + s^*$ , the gradient for  $h_{it}(s)$  will change for all  $t = 1, \dots, k$ . As mentioned in Section 2, for NMF with square loss we can update the gradient for one row of  $W$  in  $O(k)$  time. However, there is no easy way to update the values together for KL-divergence. To maintain the gradient, we need to update  $(WH)_{ij}$  for all  $j = 1, \dots, n$  by

$$(WH)_{ij} = (WH)_{ij} + s^* H_{rj} \quad \forall j, \quad (22)$$

and then recompute  $h'_{it}(0)$  for all  $t = 1, \dots, k$  by (20). Therefore maintaining  $h'_{it}(0)$  will take  $O(kn)$  time. This is expensive compared to the time cost  $O(n)$  for updating one variable, so we just update variables in a cyclic order. Thus our method for KL-divergence is a Cyclic Coordinate Descent (CCD). Notice that to distinguish our method for KL-divergence with **FastHals** (cyclic coordinate descent for least squares NMF), through this paper CCD indicates our method for KL-divergence.

In summary, our algorithm chooses each variable in  $W$  once in a cyclic order, minimizing the corresponding one-variable sub-problem, and then switches to  $H$ . Each Newton update takes  $O(n)$  time, so each coordinate update costs  $O(nd)$  time where  $d$  is the average number of Newton iterations. Algorithm 2 summarizes the details.

## 4. CONVERGENCE PROPERTIES AND RELATIONS WITH OTHER METHODS

### 4.1 Methods for Least Squares NMF

For NMF,  $f(W, H)$  is convex in  $W$  or  $H$  but not simultaneously convex in both  $W$  and  $H$ , so it is natural to apply alternating minimization, which iteratively minimizes the following two problems

$$\min_{W \geq 0} f(W, H) \text{ and } \min_{H \geq 0} f(W, H) \quad (23)$$

until convergence. As mentioned in [10], we can categorize NMF solvers into two groups: exact methods and inexact methods. Exact methods are guaranteed to achieve optimum for each sub-problem (23), while inexact methods only guarantee decrease in function value. Each sub-problem for least squares NMF can be decomposed into a sequence of non-negative least square (NNLS) problems. For example, minimization with respect to  $H$  can be decomposed into  $\min_{h_i \geq 0} \|V - W h_i\|^2$ , where  $h_i$  is the  $i$ th column of  $H$ . Since the convergence property for exact methods has been proved in [6], any NNLS solver can be applied to solve NMF in an alternating fashion. However, as mentioned before, very re-

cently an inexact method **FastHals** has been proposed [3]. The success of **FastHals** shows that exactly solving sub-problems may slow down convergence. This makes sense because when  $(W, H)$  is still far from optimum, there is no reason in paying too much effort for solving sub-problems exactly. Since GCD does not solve sub-problems exactly, it can avoid paying too much effort for each sub-problem. On the other hand, unlike most inexact methods, GCD guarantees the quality of each updates by setting a stopping condition and, thus converges faster than inexact methods.

Moreover, most inexact methods do not have a theoretically convergence proof, thus the performance may not be stable. In contrast, we prove that GCD converges to a stationary point by the following theorem:

**Theorem 2**

For least squares NMF, if a sequence  $\{(W_i, H_i)\}$  is generated by GCD, then every limit point of this sequence is a stationary point.

The proof can be found in the appendix of our technical report [9]. This convergence result holds for any inner stopping condition  $\epsilon < 1$ , thus it is different from the proof for exact methods, which assumes that each sub-problem is solved exactly. It is easy to extend the convergence result for GCD to regularized least squares NMF.

**4.2 Methods for NMF with KL divergence**

NMF with KL divergence is harder to solve compared to square loss. Assume an algorithm applies an iterative method to solve  $\min_{W \geq 0} f(W, H)$ , and needs to compute the gradient at each iteration. After computing the gradient (6) at the beginning, least squares NMF solvers can maintain the gradient in  $O(mk^2)$  time when  $W$  is updated. So it can do many inner updates for  $W$  with comparatively less effort  $O(mk^2) \ll O(nmk)$ . Almost all least squares NMF solvers take advantage of this fact. However, for KL-divergence, the gradient (20) cannot be maintained within  $O(nmk)$  time after each update of  $W$ , so the cost for each sub-problem is  $O(nmkt)$  where  $t$  is the number of inner iterations, which is large compared to  $O(nmk)$  for square loss.

Our algorithm (CCD) spends  $O(nmk\bar{d})$  time where  $\bar{d}$  is the average number of Newton iterations, while the multiplicative algorithm spends  $O(nmk)$  time. However, CCD has a better solution for each variable because we use a second order approximation of the actual function, which is better compared to working on the auxiliary function as used in multiplicative algorithm proposed by [15]. Experiments in Section 6 show CCD is much faster.

In addition to the practical comparison, the following theorem proves CCD converges to a stationary point under certain condition. **FastHals** for least squares NMF can also be covered by this theorem because it is also a cyclic coordinate descent method.

**Theorem 3**

For any limit points  $(W^*, H^*)$  of CCD (or **FastHals**), assume  $w_r^*$  is the  $r$ th column of  $W^*$  and  $h_r^*$  is the  $r$ th row of  $H^*$ , if

$$\|w_r^*\| > 0, \|h_r^*\| > 0 \quad \forall r = 1, \dots, k, \quad (24)$$

then  $(W^*, H^*)$  is a stationary point of (3) (or (1)).

With the condition (24), the one-variable sub-problems for the convergence subsequence are strictly convex. Then the proof follows the proof of Proposition 3.3.9 in [2]. For KL-NMF, (24) is violated only when corresponding row/columns

of  $V$  are all zero (otherwise the objective value will be infinity). Therefore, zero row/columns of  $V$  can be removed by preprocessing to ensure the convergence of CCD. For least squares NMF, usually (24) holds in practice so that **FastHals** also converges to a stationary point.

For CCD with regularization on both  $W$  and  $H$ , each one-variable sub-problem becomes strictly quasiconvex, thus we can apply Proposition 5 in [6] to prove the following theorem:

**Theorem 4**

Any limit point  $(W^*, H^*)$  of CCD (or **FastHals**) is a stationary point of (3) (or (1)) with  $L1$  or  $L2$  regularization on both  $W$  and  $H$ .

**5. IMPLEMENTATION ISSUES**

**5.1 Implementation with MATLAB and C**

It is well known that MATLAB is very slow in loop operations and thus, to implement GCD, the “for loop” in Step 7 of Algorithm 1 is slow in MATLAB. To have an efficient implementation, we transfer three matrices  $W, G^W, HH^T$  to C by MATLAB-C interface in Step 7. At the end of the loop, our C code returns  $W^{new}$  back to the main MATLAB program. Although this implementation gives an overhead to transfer  $O(nk + mk)$  size matrices, our algorithm still outperforms other methods in experiments. In the future, it is possible to directly use BLAS3 library to have a faster implementation in C.

**5.2 Stopping Condition**

The stopping condition is important for NMF solvers. Here, we adopt projected gradient as stopping condition as in [18]. The projected gradient for  $f(W, H)$ , i.e.,  $\nabla^P f(W, H)$  has two parts including  $\nabla_W^P f(W, H)$  and  $\nabla_H^P f(W, H)$ , where

$$\nabla_W^P f(W, H)_{ir} \equiv \begin{cases} \frac{\partial}{\partial w_{ir}} f(W, H) & \text{if } W_{ir} > 0, \\ \min(0, \frac{\partial}{\partial w_{ir}} f(W, H)) & \text{if } W_{ir} = 0. \end{cases} \quad (25)$$

$\nabla_W^P f(W, H)$  can be defined in a similar way. According to the KKT condition,  $(W^*, H^*)$  is a stationary point if and only if  $\nabla^P f(W^*, H^*) = 0$ , thus we can use  $\nabla^P f(W^*, H^*)$  to measure how close we are to a stationary point. We stop the algorithm after the norm of projected gradient satisfies the following stopping condition:

$$\|\nabla^P f(W, H)\|_F^2 \leq \epsilon \|\nabla^P f(W^0, H^0)\|_F^2,$$

where  $W^0$  and  $H^0$  are initial points.

**6. EXPERIMENTS**

In this section, we compare the performance of our algorithms with other NMF solvers. All sources used for our comparisons are available at <http://www.cs.utexas.edu/~cjhsieh/nmf>. All the experiments were executed on 2.83 GHz Xeon X5440 machines with 32G RAM and Linux OS.

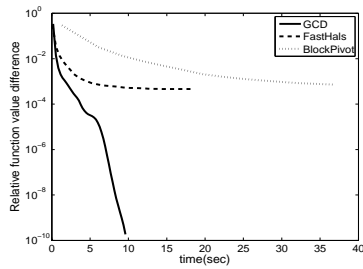
**6.1 Comparison on dense datasets**

For least squares NMF, we compare GCD with three other state-of-the-art solvers:

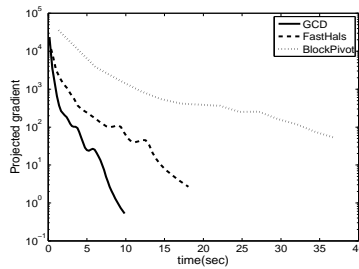
1. **ProjGrad**: the projected gradient method in [18]. We use the MATLAB source code at <http://www.csie.ntu.edu.tw/~cjlin/nmf/>.
2. **BlockPivot**: the block-pivot method in [12]. We use the MATLAB source code at <http://www.cc.gatech.edu/~hpark/nmfsoftware.php>.

Table 2: The comparisons for least squares NMF solvers on dense datasets. For each method we present time/FLOPs (number of floating point operations) cost to achieve the specified relative error. The method with the shortest running time is boldfaced. The results indicate that GCD is most efficient both in time and FLOPs.

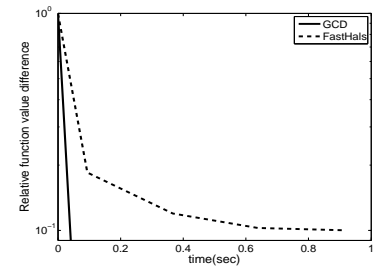
dataset	$m$	$n$	$k$	relative error	Time (in seconds)/FLOPs			
					GCD	FastHals	ProjGrad	BlockPivot
Synth03	500	1,000	10	$10^{-4}$	<b>0.6/0.7G</b>	2.3/2.9G	2.1/1.4G	1.7/1.1G
			30	$10^{-4}$	<b>4.0/5.0G</b>	9.3/16.1G	26.6/23.5G	12.4/8.7G
Synth08	500	1,000	10	$10^{-4}$	<b>0.21/0.11G</b>	0.43/0.38G	0.53/0.41G	0.56/0.35G
			30	$10^{-4}$	<b>0.43/0.46G</b>	0.77/1.71G	2.54/2.70G	2.86/1.43G
CBCL	361	2,429	49	0.0410	<b>2.3/2.3G</b>	4.0/10.2G	13.5/14.4G	10.6/8.1G
				0.0376	<b>8.9/8.8G</b>	18.0/46.8G	45.6/49.4G	30.9/29.8G
				0.0373	<b>14.6/14.5G</b>	29.0/75.7G	84.6/91.2G	51.5/53.8G
ORL	10,304	400	25	0.0365	<b>1.8/2.7G</b>	6.5/14.5G	9.0/9.1G	7.4/5.4G
				0.0335	<b>14.1/20.1G</b>	30.3/66.9G	98.6/67.7G	33.9/38.2G
				0.0332	<b>33.0/51.5G</b>	63.3/139.0G	256.8/193.5G	76.5/82.4G



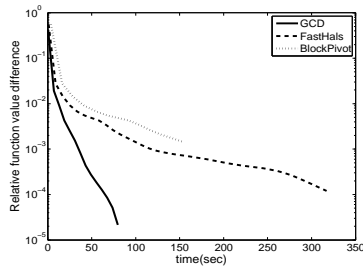
(a) Objective value for Yahoo-News dataset



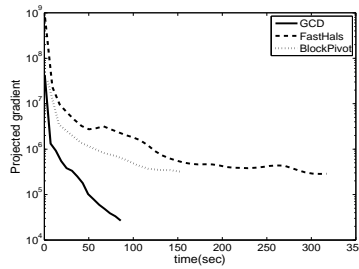
(b) Projected gradient for Yahoo-News dataset



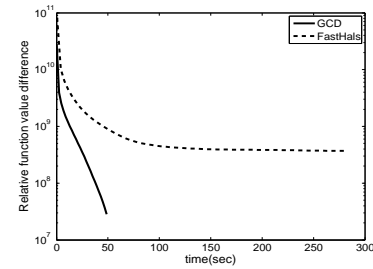
(c) L1 regularized objective value for Yahoo-News dataset, with  $\rho_1 = 10, \rho_2 = 20$



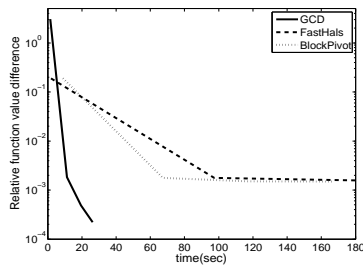
(d) Objective value for MNIST dataset



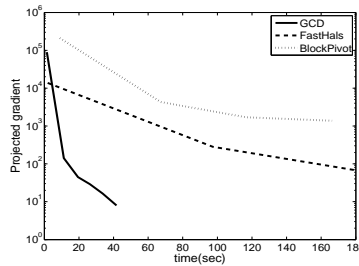
(e) Projected gradient for MNIST dataset



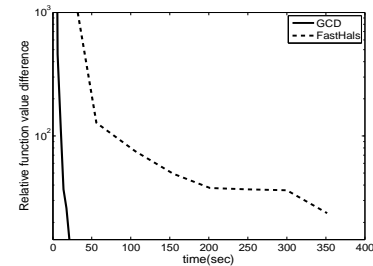
(f) L1 regularized objective value for MNIST dataset with  $\rho_1 = 50,000, \rho_2 = 100,000$



(g) Objective value for RCV1 dataset



(h) Projected gradient for RCV1 dataset



(i) L1 regularized objective value for RCV1 dataset with  $\rho_1 = 0.005, \rho_2 = 0.05$

Figure 2: Time comparison for large sparse datasets. The result indicate that GCD is both faster and converges to better solutions.

3. **FastHals**: Cyclic coordinate descent method in [3]. We implemented the algorithm in MATLAB.

For GCD, we set the inner stopping condition  $\epsilon$  to be 0.001. We test the performance on the following dense datasets:

1. Synthetic dataset: Following the process in [12], we generate the data by first randomly creating  $W$  and  $H$ , and then compute  $V = WH$ . We generate two datasets Synth03 and Synth08, the suffix numbers indicate 30% or 80% variables in solutions are zeros.
2. CBCL image dataset: <http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html>
3. ORL image dataset: <http://www.c1.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

We follow the same setting as in [8] for CBCL and ORL datasets. The size of the datasets are summarized in Table 2. To ensure a fair comparison, all experimental results in this paper are the average of 10 random initial points.

Table 2 compares the CPU time for each solver to achieve the specified relative error defined by  $\|V - WH\|_F^2 / \|V\|_F^2$ . For synthetic dataset, since the exact factorization exists, all the methods can achieve very low objective function value. From Table 2, we can conclude that GCD is two to three times faster than **BlockPivot** and **FastHals** on dense datasets.

As mentioned in Section 5, we implement part of GCD in C. To have a fair comparison, the FLOPs (number of Floating Point Operations) is listed in Table 2. **FastHals** is competitive in time but slow in FLOPs. This is because it fully utilizes dense matrix-multiplication operations, which is efficient in MATLAB.

For NMF with KL divergence, we compare the performance of our cyclic coordinate descent method (CCD) with the multiplicative update algorithm (**Multiplicative**) proposed in [15]. As mentioned in Section 3, the method proposed in [3] solves a different formulation, so we do not include it in our comparisons. Table 3 shows the runtime for CCD and **Multiplicative** to achieve the specified relative error. For KL-divergence, we define the relative error to be the objective value  $L(W, H)$  in (3) divided by  $\sum_{i,j} V_{ij} \log(\frac{V_{ij}}{(\sum_j V_{ij})/n})$ , which is the distance between  $V_{ij}$  and the uniform distribution for each row. We implement CCD in C and **Multiplicative** in MATLAB, so we also list the FLOPs in Table 3. Table 3 shows that CCD is 2 to 3 times faster than **Multiplicative** at the beginning, and can be 10 times faster to get a more accurate solution. If we consider FLOPs, CCD is even better.

## 6.2 Comparison on sparse datasets

In Section 6.1, **BlockPivot**, **FastHals**, and GCD are the three most competitive methods. To test their scalability, we further compare their performances on large sparse datasets. We use the following sparse datasets:

1. Yahoo-News (K-Series): A news articles dataset.
2. RCV1 [16]: An archive of newswire stories from Reuters Ltd. The original dataset has 781,265 documents and 47,236 features. Following the preprocessing in [18], we choose data from 15 random categories and eliminate unused features. However, our data is much larger than the one used in [18].
3. MNIST [13]: A collection of hand-written digits.

The statistics of the datasets are summarized in Table 4. We set  $k$  according to the number of categories for each datasets. We run GCD and **FastHals** with our C implementations with sparse matrix operations, and for **BlockPivot** we use the author’s code in MATLAB with sparse  $V$  as input.

In Figure 2(a), 2(d), 2(g), we show the CPU time for the 3

Table 3: Time comparison results for KL divergence. \* indicates the specified objective value is not achievable. The results indicate CCD outperforms **Multiplicative**

dataset	$k$	relative error	Time (in seconds)/FLOPs	
			CCD	<b>Multiplicative</b>
Synth03	10	$10^{-3}$	<b>11.4/5.2G</b>	34.0/68.1G
		$10^{-5}$	<b>14.8/6.8G</b>	144.2/240.6G
	30	$10^{-3}$	<b>121.1/58.7G</b>	749.5/2057.4G
		$10^{-5}$	<b>184.32/89.3G</b>	7092.3/18787.8G
Synth08	10	$10^{-2}$	<b>2.5/1.7G</b>	30.3/71.6G
		$10^{-5}$	<b>13.0/8.8G</b>	*
	30	$10^{-2}$	<b>22.6/11.2G</b>	46.0/93.9G
		$10^{-5}$	<b>56.8/27.7G</b>	*
CBCL	49	0.1202	<b>38.2/18.2G</b>	<b>21.2/64.1G</b>
		0.1103	<b>123.2/58.4G</b>	562.6/781.3G
		0.1093	<b>166.0/78.7G</b>	3266.9/2705.4G
ORL	25	0.3370	<b>73.7/35.0G</b>	165.2/336.3G
		0.3095	<b>253.6/117.0G</b>	902.2/1323.0G
		0.3067	<b>370.2/177.5G</b>	1631.9/3280.2G

Table 4: Statistics of data sets.  $k$  is the value of reduced dimension we use in the experiments.

Data set	$m$	$n$	#nz	$k$
Yahoo-News	21,839	2,340	349,792	20
MNIST	7,80	60,000	8,994,156	10
RCV1(subset)	31,025	152,120	7,817,031	15

methods to reduce the objective value of least squares NMF, versus with logarithmically decreasing values of  $(f(W, H) - f^*)/f^*$ , where  $f^*$  denote the lowest average objective value, for 10 initial points, of the 3 methods.

Compared to **FastHals** and **BlockPivot**, GCD converges to a local optimum with lower objective value. This is important for nonconvex optimization problems because we can find a better local optimum. To further compare the speed of convergence to local optimums, we check the projected gradient  $\nabla^P f(W, H)$ , which measures the distance between current solution and stationary points. The results are in Figure 2(b), 2(e) and 2(h). The figures indicate that GCD converges to the stationary point in lesser CPU time.

We further add L1 penalty terms as in (2). We only include GCD and **FastHals** in the comparison because **BlockPivot** does not provide the same regularized form in their package. Figures 2(c), 2(f) and 2(i) compare the methods for reducing the objective value of (2). For this comparison we choose the parameters  $\lambda_1$  and  $\lambda_2$  so that on average more than half the variables in the solution of  $W$  and  $H$  are zero. The figures indicate that GCD achieves lower objective function value than **FastHals** in MNIST, and for Yahoo-News and RCV1, GCD is more than 10 times faster. This is because GCD can focus on nonzero variables while **FastHals** updates all the variables at each iteration.

## 7. DISCUSSION AND CONCLUSIONS

In summary, we propose coordinate descent methods that do variable selection for solving least squares NMF and KL-NMF. Our methods have theoretical guarantees and are efficient on real-life data. The significant speedups on sparse data show a potential to apply NMF to larger problems. In the future, our method can be extended to solve NMF with missing values or other matrix completion problems.

## 8. ACKNOWLEDGEMENTS

This research was supported by NSF grant CCF-0728879.



## 9. REFERENCES

- [1] M. Berry, M. Browne, A. Langville, P. Pauca, and R. Plemmon. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, 2007. Submitted.
- [2] D. Bersekas and J. Tsitsiklis. Parallel and distributed computation. *Prentice-Hall*, 1989.
- [3] A. Cichocki and A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transaction on Fundamentals*, E92-A(3):708–721, 2009.
- [4] E. Gaussier and C. Goutte. Relation between PLSA and NMF and implications. *28th Annual International ACM SIGIR Conference*, 2005.
- [5] E. F. Gonzales and Y. Zhang. Accelerating the Lee-Seung algorithm for non-negative matrix factorization. Technical report, Department of Computational and Applied Mathematics, Rice University, 2005.
- [6] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Operations Research Letters*, 26:127–136, 2000.
- [7] P. O. Hoyer. Non-negative sparse coding. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 557–565, 2002.
- [8] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.
- [9] C.-J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. *Department of Computer Science TR-11-06, University of Texas at Austin*, 2011.
- [10] D. Kim, S. Sra, and I. S. Dhillon. Fast Newton-type methods for the least squares nonnegative matrix approximation problem. *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 343–354, 2007.
- [11] J. Kim and H. Park. Non-negative matrix factorization based on alternating non-negativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.
- [12] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. *Proceedings of the IEEE International Conference on Data Mining*, pages 353–362, 2008.
- [13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. MNIST database available at <http://yann.lecun.com/exdb/mnist/>.
- [14] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
- [15] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.
- [16] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
- [17] Y. Li and S. Osher. Coordinate descent optimization for  $l_1$  minimization with application to compressed sensing; a greedy algorithm. *Inverse Probl. Imaging*, 3(3):487–503, 2009.
- [18] C.-J. Lin. Projected gradient methods for non-negative matrix factorization. *Neural Computation*, 19:2756–2779, 2007.
- [19] C. Liu, H. chih Yang, J. Fan, L.-W. He, and Y.-M. Wang. *Distributed Non-negative Matrix Factorization for Web-Scale Dyadic Data Analysis on MapReduce*. 2010.
- [20] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error. *Environmetrics*, 5:111–126, 1994.
- [21] S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003.
- [22] J. Piper, P. Pauca, R. Plemmons, and M. Giffin. Object characterization from spectral data using nonnegative factorization and information theory. In *Proceedings of AMOS Technical Conference*, 2004.
- [23] R. Zdunek and A. Cichocki. Non-negative matrix factorization with quasi-newton optimization. *Eighth International Conference on Artificial Intelligence and Soft Computing, ICAISC*, pages 870–879, 2006.