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 proposed to solve 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

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:

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

$$\min_{W,H \geq 0} \frac{1}{2} \|V - WH\|^2_F + \rho_1 \sum_{i,r} W_{ir} + \rho_2 \sum_{r,j} H_{rj}.$$  \hspace{1cm} (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 [9] 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 [7] 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 subproblem, a coordinate descent method can efficiently give a reasonably good solution for each subproblem 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 $W$:  
\[
\min_{W,H \geq 0} L(W, H) = \sum_{i,j} V_{ij} \log \left( \frac{V_{ij}}{(WH)_{ij}} \right) - V_{ij} + (WH)_{ij} \quad (3)
\]

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 6 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 $s$:

\[
\min_{s; W_{ir} + s \geq 0} g_{ir}^W(s) = f(W + sE_{ir}, H).
\quad (4)
\]

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

\[
g_{ir}^W(s) = \frac{1}{2} \sum_j (V_{ij} - (WH)_{ij} - sH_{ir})^2
= g_{ir}(0) + (g_{ir})'(0)s + \frac{1}{2}(g_{ir})''(0)s^2.
\quad (5)
\]

For convenience we define

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

and $G^H \equiv \nabla_H f(W, H) = WTW - WT^TV.
\quad (7)$

Then we have

\[
(g_{ir})'(0) = (G^W)_{ir} = (WHH^T - VH^T)_{ir}
\quad (8)
\]

\[
(g_{ir})''(0) = (HHT)_{rr}.
\quad (9)
\]

Since 3 is a one-variable quadratic function with the constraint $W_{ir} + s \geq 0$, we can solve it in closed form:

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

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

\[
g_{ir}^W(s) = \left( (WHH^T - VH^T)_{ir} + \rho_l \right) s + \frac{1}{2}(HHT)_{rr} s^2 + g_{ir}(0),
\]

and thus the one variable update becomes

\[
W_{ir} \leftarrow \max \left( 0, W_{ir} - ((WHH^T - VH^T)_{ir} + \rho_l)/(HHT)_{rr} \right).
\]

Compared to 3, the new $W_{ir}$ is shifted left to zero by the factor $\rho_l/(HHT)_{rr}$, 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
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 variable selection scheme, 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.

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:

$$(W^0, H^0) \rightarrow (W^1, H^0) \rightarrow (W^1, H^1) \rightarrow \cdots$$

(10)

Between each outer iteration are the following inner updates:

$$(W^i, H^i) \rightarrow (W^{i+1}, H^i) \rightarrow (W^{i+1}, H^{i+1}) \cdots$$

(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 2.4, the optimal update will be (9), and the function value will be decreased by

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

(12)

$D_{ir}^W$ measures how much we can reduce the objective value by choosing the coordinate $W_{ir}$. Therefore, if $D_{ir}^W$ can be maintained, we can greedily choose variables according to it. If we have $G^W$, we can compute $s^*$ by (8) and compute $D_{ir}^W$ 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

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

(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 (9), 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 $O(km)$, 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 (14). To select the next variable-to-update, we want to select the index $(i^*, r^*)$ that satisfies

$$(i^*, r^*) = \arg \max_{i,j} D_{ij}^W.$$  

(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 $q_i$ will remain the same except for one component $q_{ir}$. 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 $\{q_i \mid i = 1, \ldots, m\}$, 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

$$p_{ij}^{\text{init}} = \max_{i,j} D_{ij}^W.$$  

(15)

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

$$\max_{i,j} D_{ij}^W < \epsilon p_{ij}^{\text{init}},$$  

(16)

where $\epsilon$ is a small positive constant. Note that $\epsilon p_{ij}^{\text{init}}$ 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 we can iteratively update variables in the \( i \)th row and there is a variable that lead to maximum decrease in objective function) until the inner stopping condition is met:

\[
\max_j D^W_{ij} \leq \epsilon p^\text{init}. \tag{17}
\]

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 \( H^{\text{new}} \leftarrow 0 \)
- While (not converged)
  1. Compute \( P^{VH} \leftarrow P^{VH} + V(H^{\text{new}})^T \) according to the sparsity of \( H^{\text{new}} \)
  2. \( W^{\text{new}} \leftarrow 0 \)
  3. \( G^W \leftarrow W P^{HH} - P^{VH} \)
  4. \( S^W_{ir} \leftarrow \max(W_{ir} - G^W_{ir} / \epsilon^2, 0) - W_{ir} \) for all \( i, r \)
  5. \( D^W_{ir} \leftarrow \max(S^W_{ir})^2 \) for all \( i, r \)
  6. \( q_i \leftarrow \text{arg max}_j D^W_{ij} \) for all \( i = 1, \ldots, m \), and \( p^\text{init} \leftarrow \max_j D^W_{ij} \)
  7. For \( i = 1, \ldots, m \)
    1. \( s^W_{qi} \leftarrow S^W_{qi} \)
    2. \( P^{Wq_i} \leftarrow P^{Wq_i} + s^W_{qi} \) (Also do a symmetric update for \( P^{q_iW} \))
    3. \( W_{qi}^{\text{new}} \leftarrow W_{qi}^{\text{new}} + s^W_{qi} \)
    4. \( G^H_{qi} \leftarrow G^H_{qi} + s^W_{qi} \)
    5. \( S^H_{qi} \leftarrow \max(W_{qi} - G^H_{qi} / \epsilon^2, 0) - W_{qi} \) for all \( r = 1, \ldots, k \)
    6. \( D^H_{qi} \leftarrow \max(S^H_{qi})^2 \) for all \( r = 1, \ldots, k \)
    7. \( q_i \leftarrow \text{arg max}_j D^H_{ij} \)
  8. \( W \leftarrow W + W^{\text{new}} \)
  9. For updates to \( H \), repeats analogous steps to Step 4 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 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 (9) has two terms \( WHH^T + VH^T \) (in Algorithm 1) we use \( P^{VH}, P^{HH}, P^{WV} \) and \( P^{WW} \) to store matricies \( V^H \), \( H^H \), \( W^T V \) 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 \( V^H \), we can also store the updates in \( H^{\text{new}} \) so that \( H^{\text{new}} = H^{t+1} - H^t \), and then updates \( V^H \) by step 1. Assume that, on average, step 4 has \( t \) coordinate updates, then the number of nonzeroes in \( H^{\text{new}} \) is at most \( t \). With sparse matrix multiplication, step 4 costs \( O(mt) \). However, when \( H^{\text{new}} \) is dense, we should use a dense matrix multiplication for step 4 which costs \( O(mnk) \) 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 \( V^H \) dominates the complexity and the time cost per coordinate update is \( O(m) \). In this case, time 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:

Table 1: Time complexity analysis for Algorithm 1

<table>
<thead>
<tr>
<th>Dense V</th>
<th>Sparse V</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(\frac{sk}{t}) )</td>
<td>( O(\frac{mk^2}{t}) )</td>
</tr>
</tbody>
</table>

Descent when \( k < k^2 \), \( O(mk^2) \) will dominate the complexity. When \( k^2 \leq t \leq nk \), computational time for computing \( V^H \) dominates the complexity and the time cost per coordinate update is \( O(m) \). In this case, time 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:

\[
\begin{align*}
\begin{cases}
O(\frac{mk^2}{t}) & \text{if } k^2 > t \\
O(m) & \text{if } nk > t \geq k^2 \\
O(\frac{nk^2}{t}) & \text{if } nm > t \geq nk \\
O(k) & \text{if } t > nm
\end{cases}
\end{align*}
\]

When \( V \) is sparse, assume there are \( s \) nonzero elements in \( V \), the complexity for the computing \( V^H \) is modified when others remain the same.

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

To apply coordinate descent for solving NMF with KL-divergence \([5]\), we consider the one-variable sub-problem:

\[
h_{ir}(s) = L(W + sE_{ir}, H) \tag{18}
\]

\[
= \sum_{j=1}^{n} -V_{ij} \log \left( (W H)_{ij} + sH_{ir} \right) + sH_{ir} + \text{constant}.
\]

Here we discuss the updates to \( W \), the update rules for \( H \) can be derived similarly. Unlike the one-variable subproblem 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:

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

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:

\[
s \leftarrow \max\left(-W_{ir}, s - \frac{h'_{ir}(s)}{h''_{ir}(s)}\right), \tag{19}
\]

where the first derivative \( h'_{ir}(s) \) and the second derivative \( h''_{ir}(s) \) can be written as

\[
h'_{ir}(s) = \sum_{j=1}^{n} H_{rj} \left(1 - \frac{V_{ij}}{(WH)_{ij} + s H_{rj}}\right), \tag{20}
\]

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

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 \(-\frac{h'_{ir}(s)}{h''_{ir}(s)}\) 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 \( j = 1, \ldots, n \), the second derivative \( h''_{ir}(s) \) 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

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

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 \( W_{ir} \leftarrow W_{ir} + s^t \), the gradient for \( h_{ir}(s) \) will change for all \( t = 1, \ldots, k \). 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 \( j = 1, \ldots, n \) by

\[
(WH)_{ij} = (WH)_{ij} + s^t H_{rj} \quad \forall j, \tag{22}
\]

and then recompute \( h''_{ir}(0) \) for all \( t = 1, \ldots, k \) by (20).

Therefore maintaining \( h''_{ir}(0) \) 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.

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

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

3.5. 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

\[
\min_{W \geq 0} f(W, H) \text{ and } \min_{H \geq 0} f(W, H) \tag{23}
\]

until convergence. As mentioned in [19], 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

\[
\min_{h_{ij} \geq 0} \|V - Wh_{ij}\|^2, \text{ where } h_{ij} \text{ is the } i \text{th column of } H.
\]

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-
ently an inexact method FastHals has been proposed proposed\cite{Zhang}. 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 \cite{Zhang}. This convergence result holds for any inner stopping condition \(\epsilon < 1\), 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 \(\nabla f(W)\) 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 \(\nabla^P f(W)\) cannot be maintained within \(O(nmk)\) time after each update of \(W\), so the cost for each sub-problem is \(O(nmk^2)\), where the number of inner iterations, which is large compared to \(O(nmk)\) for square loss.

Our algorithm (CCD) spends \(O(nmk)\) time where \(d\) 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 \cite{Lee}. Experiments in Section 5 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

\[
\|w_r^*\| > 0, \|h_r^*\| > 0 \quad \forall r = 1, \ldots, k, \tag{24}
\]

then \((W^*, H^*)\) is a stationary point of \(\Phi\) (or \(\Phi\)).

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 \cite{Lee}. For KL-NMF, \(\Phi\) 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 \(\Phi\) 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 \cite{Lee} to prove the following theorem:

**Theorem 4**

Any limit point \((W^*, H^*)\) of CCD (or FastHals) is a stationary point of \(\Phi\) (or \(\Phi\)) 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 \(W, G^W, H H^T\) to C by MATLAB-C interface in Step 7. At the end of the loop, our C code returns \(W^{\text{new}}\) 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 \cite{Zhang}. The projected gradient for \(f(W, H)\), i.e., \(\nabla^P f(W, H)\) has two parts including \(\nabla^P f(W, H)\) and \(\nabla^P f(W, H)\), where

\[
\nabla^P_{W, H} f(W, H)_{ir} = \begin{cases} 
\frac{\partial}{\partial W_{ir}} f(W, H) & \text{if } W_{ir} > 0, \\
\min(0, \frac{\partial}{\partial W_{ir}} f(W, H)) & \text{if } W_{ir} < 0.
\end{cases}
\tag{25}
\]

\(\nabla^P_{W, H} f(W, H)\) 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/~cjlin/nmf/](http://www.cs.utexas.edu/~cjlin/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:

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.

<table>
<thead>
<tr>
<th>dataset</th>
<th>m</th>
<th>n</th>
<th>k</th>
<th>relative error</th>
<th>Time (in seconds)/FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GCD</td>
</tr>
<tr>
<td>Synth03</td>
<td>500</td>
<td>1,000</td>
<td>10</td>
<td>$10^{-4}$</td>
<td>0.6/0.7G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>$10^{-4}$</td>
<td>4.0/5.0G</td>
</tr>
<tr>
<td>Synth08</td>
<td>500</td>
<td>1,000</td>
<td>10</td>
<td>$10^{-4}$</td>
<td>0.21/0.11G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>$10^{-4}$</td>
<td>0.43/0.46G</td>
</tr>
<tr>
<td>CBCL</td>
<td>361</td>
<td>2,429</td>
<td>49</td>
<td></td>
<td>0.0410</td>
</tr>
<tr>
<td>ORL</td>
<td>10,304</td>
<td>400</td>
<td>25</td>
<td></td>
<td>0.0385</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0376</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0373</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0365</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0335</td>
</tr>
<tr>
<td>ORL</td>
<td>10,304</td>
<td>400</td>
<td>25</td>
<td></td>
<td>0.0332</td>
</tr>
</tbody>
</table>

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.cl.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 $\parallel V - WH \parallel_F^2/\parallel V \parallel_F^2$. 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 [9] 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 of $L(W,H)$ 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:

2. RCV1 [10]: An archive of newswire stories from Reuters Ltd. The original dataset has 781,265 documents and 47,236 features. Following the preprocessing in [13], we choose data from 15 random categories and eliminate unused features. However, our data is much larger than the one used in [13].
3. MNIST [12]: 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 dataset. We run CCD 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 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 $\nabla f(W,H)$, 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


10. APPENDIX

10.1 Lemma 1

Lemma 1

Assume GCD generates a sequence of \((W, H)\) pairs:

\[
(W^i, H^i), (W^2, H^1), (W^2, H^2), \ldots
\]

Suppose there exists a limit point \((\bar{W}, \bar{H})\). Then the function value globally converges to some value:

\[
\lim_{k \to \infty} f(W^k, H^k) = \lim_{k \to \infty} f(W^{k+1}, H^k) = f(\bar{W}, \bar{H}).
\]  

(27)

Proof. This can be simply proved by the fact that our algorithm monotonically decreases the function value.

10.2 Proof of Theorem 1

Assume the Newton method generates the following sequence:

\[
x^{i+1} = \max(0, x^i - \frac{f'(x^i)}{f''(x^i)}), \quad i = 0, 1, \ldots
\]

(28)

The gradient and the second derivative are:

\[
f'(x) = -\sum_i \frac{c_ib_i}{a_i+b_ix} + \sum_i b_i x
\]

(29)

\[
f''(x) = \sum_i \frac{c_ib^2_i}{(a_i+b_i x)^2} > 0
\]

(30)

First, we show that

\[
f'(x^i) \leq 0 \quad \forall i \geq 1.
\]

(31)

To prove (31), since \(f''(x)\) is positive and decreasing,

\[
f'(y) \leq f'(x) + (y-x) f''(x) \quad \forall x, y \geq 0.
\]

By setting \(y = x - \frac{f'(x)}{f''(x)}\) we have

\[
f'(x - \frac{f'(x)}{f''(x)}) < 0.
\]

Moreover, from (29) we have \(f'(0) < 0\). Therefore after one Newton step, \(x^{i+1}\) will be either \(x^i - \frac{f'(x^i)}{f''(x^i)}\) or 0. In either case (31) holds. Therefore after the sequence after the first iteration \(\{x^i\}_{i \geq 1}\) have the property that \(f'(x^i) < 0\) and, thus \(-\frac{f'(x^i)}{f''(x^i)} > 0\). So the sequence \(\{x^i\}_{i \geq 1}\) is monotonically increasing in a compact set \(f'(x) \leq 0, \{x^i\}\) then converges to a limit point, say \(x^*\). \(f'(x^*)\) can be either negative or zero. If \(f'(x^*) < 0\), since \(\lim_{i \to \infty} x_i = x^*\), when \(i\) is large enough we have the following two properties by continuity of first and second derivatives:

\[
\frac{f'(x_i)}{f''(x_i)} < \frac{f'(x^*)}{2f''(x^*)}
\]

(32)

\[
x^* - x^i < \frac{f'(x^*)}{2f''(x^*)}
\]

(33)

Combine (32) and (33) we have

\[
x^* - x^{i+1} = x^* - x^i + \frac{f'(x_i)}{f''(x_i)} < 0,
\]

which raises the contradiction with \(x^{i+1} \leq x^*\). So the Newton method converges to the optimal solution \(f'(x^*) = 0\).

10.3 Proof of Theorem 2

First, we look into our algorithm to see what properties we can use. In a sequence of updates from \((W^k, H^k)\) to \((W^{k+1}, H^k)\), at the beginning GCD computes the maximum function value reduction for one coordinate:

\[
v_0 = \max_{i,j,q} f(W^k, H^k) - f(W^k + qe_{ij}, H^k).
\]

From Section 2 GCD will choose this variable with objective function decreasing \(v_0\) no matter selecting the max element in \(D^W\) or selecting max value row by row. So we have

\[
f(W^k, H^k) - f(W^{k+1}, H^k) \geq v_0.
\]

(34)

GCD updates variables in \(W\) until the maximum possible function value reduction is less than \(\epsilon v_0\), that is, in the end of updates of variables of \(W\), we will get \(W^{k+1}\) with the following property:

\[
\max_{i,j,q} f(W^{k+1}, H^k) - f(W^{k+1} + qe_{ij}, H^k) < \epsilon v_0 \leq v_0,
\]

(35)

where \(\epsilon \in (0, 1)\). Combining (34) and (35)

\[
f(W^k, H^k) - f(W^{k+1}, H^k) > \max_{i,j,q} f(W^{k+1}, H^k) - f(W^{k+1} + qe_{ij}, H^k).
\]

(36)

We will use (36) for later proof.

For a sequence of updates on variables of \(H\), assume at the beginning the maximum function value decreasing is

\[
\bar{v}_0 = \max_{i,j,q} f(W^k, H^k) - f(W^k + p e_{ij}, H^k).
\]

(37)

By the same argument of (37), we have

\[
f(W^{k+1}, H^k) - f(W^{k+1} + q e_{ij}, H^k) < \bar{v}_0.
\]

(38)

With (37) we have

\[
f(W^{k+1}, H^{k+1}) \leq \min_{i,j,q} f(W^{k+1}, H^k + p e_{ij}).
\]

(39)

Next we will use (36) and (37) to finish our proof. Assume the sequence has a limit point \((W, \bar{H})\), without loss of generality, assume there is a subsequence

\[
(W^{k+1}, H^k)_{k \in K} \to (W, \bar{H}),
\]

(40)

(otherwise there will exist \((W^k, H^k) \to (W, \bar{H})\), and we can use the similar argument.)

Now if \((W, \bar{H})\) is not a stationary point, one of the following conditions will hold:

\[
\exists i, j, q \text{ such that } f(W + qe_{ij}, H) \leq f(W, \bar{H}) - \sigma
\]

(41)

\[
\exists i, j, q \text{ such that } f(W, H + pe_{ij}) \leq f(W, \bar{H}) - \sigma
\]

(42)

for some \(\sigma > 0\).

First, consider when (41) happens for some \(i, j\) and \(q\). From the continuity of \(f(\cdot)\) we know there exists a large enough constant \(R\) such that

\[
f(W^{k+1} + pe_{ij}, H^k) \leq f(W, \bar{H}) - \sigma/2
\]

\forall k > R \text{ and } k \in K
Thus
\[
f(W^{k+1}, H^k) - f(W^{k+1} + pe_{ij}, H^k) \geq
f(W^{k+1}, H^k) - f(\bar{W}, \bar{H}) + \sigma/2 \geq \sigma/2,
\]
where the second inequality comes from the monotonicity of the sequence generated by GCD. With (36) we have
\[
f(W^k, H^k) - f(W^{k+1}, H^k) > \sigma/2 \quad \forall k > R \text{ and } k \in K.
\]
However, this will make \( f(W^k, H^k) \to -\infty \) when \( k \to \infty \), a contradiction.

Second, when case (42) happens, by (39) we have
\[
f(W^{k+1}, H^{k+1}) \leq f(W^{k+1}, H^k + qe_{ij}) < f(W^{k+1}, H^k)
\]
when \( k \) large enough. By the continuity of \( f \) and the fact that \( \lim_{k \in K} (W^{k+1}, H^k) = (\bar{W}, \bar{H}) \), taking limit on both sides and using Lemma 1 we get
\[
f(\bar{W}, \bar{H} + qe_{ij}) = f(\bar{W}, \bar{H}),
\]
which raises the contradiction with (42).

Since both case (11) and (42) cannot happen, any limit point of our algorithm is a stationary point.