# Non-negative Matrix Factorization with Applications to Handwritten Digit Recognition

Michael J. M. Mazack
Department of Scientific Computation
University of Minnesota - Twin Cities

December 15, 2009

## Introduction

In the last decade, non-negative matrix factorization (NMF) has become a widely used method for solving problems in data mining and pattern recognition. The NMF in its present state can be traced back to the work of Paatero and Tapper in 1994 at the University of Helsinki under the name, "positive matrix factorization" [1]. This technique was popularized by Lee and Seung in 1999 under its current name, "non-negative matrix factorization" [2]. In the last decade, Lee and Seung have continued to develop and publish algorithms for the computation of NMF [3]. Since 2005, sparse variants of NMF have become quite popular and have been successfully used in cancer class discovery [4] and microarray data analysis [5]. Recent work has focused on improving existing NMF algorithms by attempting to remove random initialization requirements. In this paper, we discuss dense and sparse algorithms for the computation of NMF, provide their MATLAB implementations, and examine the use of NMF in the context of handwritten digit recognition.

## NMF and its Properties

We begin our discussion of NMF with its definition and a few observations.

**Definition 1.** *Let $A \in \mathbb{R}^{m\times n}$ be matrix such that $a_{ij} \geq 0$ for all $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$ (henceforth, $A \geq 0$). Then for $k \leq \min\{m,n\}$ there exist $W \in \mathbb{R}^{m\times k} \geq 0$ and $H \in \mathbb{R}^{k\times n} \geq 0$ such that $A \approx WH$.*

The first observation to make from the definition is that the NMF gives an approximation for the matrix $A$ and the number $k \leq \min\{m,n\}$ determines the rank of the approximation. Also, in the case of $k = \min\{m,n\}$, it is possible to achieve the equality[1] $A = WH$.

The second observation is that the columns of $W$ form a $k$-dimensional approximation for the column space of $A$. To see this, let $x \in \mathbb{R}^n$ and right-multiply the NMF by $x$,

$$A \approx WH \quad \Rightarrow \quad Ax \approx WHx = Wy \quad , \quad y = Hx.$$

Third, we observe that the NMF is not unique. For any diagonal matrix $D \in \mathbb{R}^{k\times k}$ with stictly positive diagonal entries,

$$A \approx WH = WDD^{-1}H = (WD)(D^{-1}H) = W'H'.$$

Thus, the NMF is not unique.

[1]The achievement of equality in this case relies on the NMF algorithm used and whether the objective-function values converge to a global minimum.

# Methods for Computing Dense NMF

The NMF has been traditionally been thought of as a solution to the following optimization problem [3]

$$\text{Minimize} \quad f(W,H) = \frac{1}{2}\|A - WH\|_F^2 \quad \text{subject to:}$$
$$W \in \mathbb{R}^{m\times k} \geq 0,$$
$$H \in \mathbb{R}^{k\times n} \geq 0.$$

We now discuss the two most widely used methods for computing a dense NMF. We first examine the multiplicative update method, and later examine the method of alternating least squares.

## Multiplicative Update

The multiplicative update method approaches the optimization problem in terms of Karush-Kuhn-Tucker (KKT) conditions [3]. It begins with random $W \geq 0$ and random $H \geq 0$ and iterates the below a chosen number of times:

$$H_{ij} \leftarrow H_{ij}\frac{\left(W^TA\right)_{ij}}{\left(W^TWH\right)_{ij}} \quad , \quad W_{ij} \leftarrow W_{ij}\frac{\left(AH^T\right)_{ij}}{\left(WHH^T\right)_{ij}}.$$

This method has several problems in practice. For example, there is no guarantee that the denominators are nonzero, so a small positive number is usually added to prevent division by zero. Next, it is possible to converge to a saddle point or one of several local minima instead of the global minimum. For an extensive treatment and discussion on the multiplicative update method see [3].

A simple MATLAB implementation of multiplicative NMF is given below.

```
function [w, h] = nmf_mu(a, k, maxiter)

[m, n] = size(a);
w = rand(m, k);
h = rand(k, n);

for i = 1:maxiter
  h = h.*(w'*a)./(w'*w*h + 1e-9);
  w = w.*(a*h')./(w*h*h' + 1e-9);
end
```

## Alternating Least Squares

A better approach to solving the NMF optimization problem is by the method of alternating least squares. This method arises out of the fact that the optimization problem is convex in both $W$ and $H$, but not simultaneously. The algorithm starts with a random $W \geq 0$ and solves the below in the order given [5].

$$\min_H \|WH - A\|_F^2 \quad \text{such that} \quad H \geq 0,$$
$$\min_W \|H^TW^T - A^T\|_F^2 \quad \text{such that} \quad W \geq 0.$$

At each step, the algorithm replaces any negative values resulting from the least squares solution with zeros. This is what is called "non-negative least squares." For this reason, the alternating (non-negative) least squares NMF algorithm is often called NMF/ANLS.

A simple implementation of NMF/ANLS in MATLAB is given on the next page.

```
function [w, h] = nmf_anls(a, k, maxiter)

[m, n] = size(a);
w = rand(m, k);

for i = 1:maxiter
  % Solve for h.
  for j = 1:n
    h(:, j) = lsqnonneg(w, a(:, j));
  end

  % Solve for w'.
  for j = 1:m
    w(j, :) = lsqnonneg(h', a(j, :)')';
  end
end
```

The above implementation is the most basic means of solving NMF/ANLS and is rather inefficient. Many improvements have been made to the algorithm, most notably the projected gradient implementation by Lin in 2007 [6].

# Methods for Computing Sparse NMF

We now examine two popular methods for computing sparse NMF, namely SNMF/R and SNMF/L. The SNMF/R method allows for regulation of sparsity in the right factor $H$ while the SNMF/L method allows for regulation of sparsity in the left factor $W$. These two methods are very similar and both can be approached as a modified NMF/ANLS problem.

## SNMF/R

SNMF/R seeks to minimize the number of nonzero entries in $H$ while attempting to provide a $W$ such that $A \approx WH$ is a useful approximation. This is accomplished by the minimization of a new objective function [5],

$$f(W,H) = \frac{1}{2}\|A - WH\|_F^2 + \eta\|W\|_F^2 + \beta\sum_{j=1}^{n}\|H(:,j)\|_1^2.$$

The parameter $\beta$ is used to adjust the sparsity in $H$ while the parameter $\eta$ is used to preserve accuracy in $W$. A sparse $H$ is easily found by choosing a large value of $\beta$ and a small value of $\eta$. We can compute SNMF/R by solving the ANLS problem below where $e_{1\times k}$ is a vector of all ones[2].

$$\min_H \left|\left| \begin{pmatrix} W \\ \sqrt{\beta}e_{1\times k} \end{pmatrix} H - \begin{pmatrix} A \\ 0_{1\times n} \end{pmatrix} \right|\right|_F^2 \quad , \quad \text{such that} \quad H \geq 0.$$

$$\min_W \left|\left| \begin{pmatrix} H^T \\ \sqrt{\eta}I_k \end{pmatrix} W^T - \begin{pmatrix} A^T \\ 0_{k\times m} \end{pmatrix} \right|\right|_F^2 \quad , \quad \text{such that} \quad W \geq 0.$$

A MATLAB implementation is easily written by slightly modifying the one for NMF/ANLS.

[2]This ANLS problem is derived from exploiting properties of the Frobenius norm.

```
function [w, h] = snmfr(a, beta, eta, k, maxiter)

[m, n] = size(a);
w = rand(m, k);

for i = 1:maxiter
  % Solve for h.
  for j = 1:n
    h(:, j) = lsqnonneg([w ; sqrt(beta)*ones(1,k)], ...
                        [a(:, j) ; 0]);
  end

  % Solve for w'.
  for j = 1:m
    w(j, :) = lsqnonneg([h' ; sqrt(eta).*eye(k)], ...
                        [a(j, :)' ; zeros(k, 1)])';
  end
end

% We now have a sparse h.
h = sparse(h);
```

## SNMF/L

The SNMF/L seeks to minimize the number of nonzero entries in $W$ while attempting to provide $H$ such that $A \approx WH$ is a useful approximation. SNMF/L is derived in a manner very similar to that of SNMF/R. The objective function for SNMF/L [5] is

$$f(W,H) = \frac{1}{2}\|A - WH\|_F^2 + \eta\|H\|_F^2 + \beta\sum_{i=1}^{m}\|W(i,:)\|_1^2.$$

Similar to SNMF/R, the parameter $\beta$ is used to adjust the sparsity in $W$ while the parameter $\eta$ is used to preserve accuracy in $H$. The same rule of thumb for obtaining a sparse $H$ in SNMF/R applies to obtaining a sparse $W$ with SNMF/L. The NMF/ANLS problem for SNMF/L is below

$$\min_H \left|\left| \begin{pmatrix} W \\ \sqrt{\eta} I_k \end{pmatrix} H - \begin{pmatrix} A \\ 0_{k\times n} \end{pmatrix} \right|\right|_F^2 \quad , \quad \text{such that} \quad H \geq 0$$

$$\min_W \left|\left| \begin{pmatrix} H^T \\ \sqrt{\beta} e_{1\times k} \end{pmatrix} W^T - \begin{pmatrix} A^T \\ 0_{1\times m} \end{pmatrix} \right|\right|_F^2 \quad , \quad \text{such that} \quad W \geq 0.$$

MATLAB implementation:

```
function [w, h] = snmfl(a, beta, eta, k, maxiter)

[m, n] = size(a);
w = rand(m, k);

for i = 1:maxiter
  % Solve for h.
  for j = 1:n
    h(:, j) = lsqnonneg([w ; sqrt(eta).*eye(k)], ...
                        [a(:, j) ; zeros(k, 1)]);
```

```
  end

  % Solve for w'.
  for j = 1:m
    w(j, :) = lsqnonneg([h' ; sqrt(beta)*ones(1, k)], ...
                        [a(j, :)' ; 0])';
  end
end

% We now have a sparse w.
w = sparse(w);
```

---

# Handwritten Digit Recognition

## Background

The postal services of various countries rely on algorithms to automatically classify handwritten digits to quickly sort mail by postal codes using a machine. Some such algorithms are based on using low-rank approximations via singular-value decomposition (SVD) in conjunction with linear least squares methods [7]. The NMF provides an alternate way to obtain a low-rank approximation of a non-negative matrix and is worth investigating for this purpose.

The problem can be posed as the following: automatically classify a single, unknown, handwritten "test digit" using a database of known "training digits". Several databases are freely available, including a database from the United States Postal Service[3]. This database consists of 7291 grayscale images of training digits and 2007 grayscale images of test digits stored as $16 \times 16$ matrices with values between -1 (white) and 1 (black). We use this database in our tests after rescaling the values between 0 (white) and 1 (black). For a specific breakdown of the database see [8].

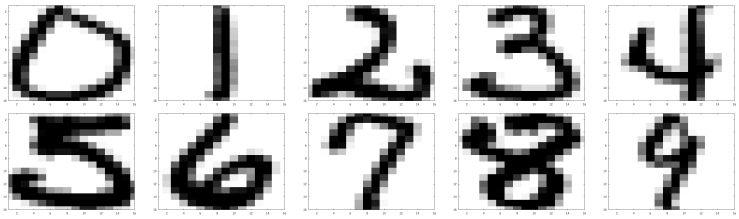

Figure 1: Several Images from the Database

## Classification Algorithm

We propose a classification algorithm using techniques similar to those done in [7]. That is, we begin by constructing the matrices $D_i$ for every $i \in \{0, 1, ..., 9\}$ whose columns consist of all digits of type $i$ from the training digit database. Each $16 \times 16$-pixel image has a corresponding and equivalent vector form in $\mathbb{R}^{256}$ formed by stacking the 16 columns.

$$D_5 = \begin{bmatrix} | & | & | & \dots & | \\ 5 & 5 & 5 & \dots & 5 \\ | & | & | & \dots & | \end{bmatrix} \quad D_5 \in \mathbb{R}^{256 \times 556}$$

Figure 2: Unrolled $16 \times 16$ matrices are stored as vectors in $\mathbb{R}^{256}$ which are columns of $D_5$.

[3]Available at: `http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html`

Once every $D_i$ matrix has been formed, we take the vector counterpart of a test digit $d \in \mathbb{R}^{256}$ and consider the least squares problem

$$\rho_i = \min_x \|D_i x - d\|_2^2.$$

Our goal is to classify $d$ by finding $\rho_i$ for every $i \in \{0, 1, ..., 9\}$ and classifying $d$ as the $i$ given by $\min_i\{\rho_i\}$. In principle, this can only be done for low-rank approximations of $D_i$ due to many of the $D_i$ matrices spanning $\mathbb{R}^{256}$. This encourages the use of NMF. Recalling that the columns of $W_{i_k}$ form a basis for the column space of $D_{i_k}$, we have

$$\begin{array}{rcccccl} D_i x & \approx & D_{i_k} x & = & W_{i_k} H_{i_k} x & = & W_{i_k} y \quad , \quad y = H_{i_k} x \\ \Rightarrow & \min\limits_x \|D_{i_k} x - d\|_2^2 & \approx & \min\limits_y \|W_{i_k} y - d\|_2^2. & & & \end{array}$$

The above allows for the use of $W_{i_k}$ in the classification algorithm instead of $D_{i_k}$. An intriguing idea is to use SNMF/L to obtain a sparse $W_{i_k}$ which can be exploited to solve the least squares problem at each step efficiently. This is considered and tested in the next section.

Below is a summary of the algorithm.

---

Let $i \in \{0, 1, ..., 9\}$.

Do once at startup:

- Form the $D_i$ matrices for every $i$.
- Compute the NMF (or SNMF/L) of each $D_i$ with a rank-$k$ approximation.

Let $d \in \mathbb{R}^{256}$ be a test digit to classify.

- For every $i$, compute $q_i = \min_y \|W_{i_k} y - d\|_2^2$.
- Compute $\min_i\{q_i\}$ and classify $d$ as an "$i$".

---

## NMF Classification Algorithm Test Results

Our tests yielded an average correct classification rate of 92.676% using a rank-10 NMF approximation of each training-digit matrix. This rate is comparable to the rate of 93.572% obtained in [8] for a rank-10 SVD approximation. See the table for specific results.

| Digit | Sample Size | Correct | Incorrect | Success Rate |
|---|---|---|---|---|
| 0 | 359 | 353 | 6 | 98.329% |
| 1 | 264 | 257 | 7 | 97.348% |
| 2 | 198 | 175 | 23 | 88.384% |
| 3 | 166 | 141 | 25 | 84.940% |
| 4 | 200 | 178 | 22 | 89.000% |
| 5 | 160 | 148 | 12 | 92.500% |
| 6 | 170 | 163 | 7 | 95.882% |
| 7 | 147 | 129 | 18 | 87.755% |
| 8 | 166 | 149 | 17 | 89.759% |
| 9 | 177 | 167 | 10 | 94.350% |

Average Success Rate: 92.676%.

The more interesting problem lies in using SNMF/L to force sparsity in each $D_i$. For these tests, we used a rank-10 approximation for each $D_i$ taking $\eta = 0.1$ and varying the values of $\beta$. The table gives the minimum and maximum number of nonzero entries over all $D_i$ and the success rate for each $\beta$. Note that each $D_i \in \mathbb{R}^{256 \times 10}$ consists of 2560 entries.

| $\beta$ | $\min(nnz(D_{i_k}))$ | $\max(nnz(D_{i_k}))$ | Success Rate |
|---|---|---|---|
| 0.01 | 542 | 1271 | 92.676% |
| 0.1 | 529 | 1199 | 91.179% |
| 1.0 | 269 | 1000 | 90.533% |
| 10.0 | 198 | 930 | 90.882% |
| 100.0 | 218 | 674 | 88.490% |
| 1000.0 | 157 | 411 | 84.853% |
| 10000.0 | 157 | 256 | 80.668% |

# Concluding Remarks

The NMF provides a way for approximating a non-negative matrix with some useful properties. In particular, NMF is a practical alternative to using the SVD to obtain low-rank approximations of non-negative matrices. As we have shown, NMF can be used for classifying handwritten digits and is competitive with SVD-based algorithms in terms of classification accuracy.

The more interesting result comes from forcing sparsity in the least-squares problem with SNMF/L. Forced sparsity results in increased performance, but reduces the accuracy significantly. Given that the initial data is dense, it is intriguing that forcing sparsity to the extent of nonzero entries totaling less than 10% of the matrix entries yielded any accurate results at all.

We also encourage the study of NMF, as it is a relatively recent phenomena with many applications and unanswered questions. Additionally, we encourage the use of SNMF/R and SNMF/L by placing the simple MATLAB implementations presented in this paper in the public domain.

# References


[1] P. Paatero and U. Tapper. Positive matrix factorization: a nonnegative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5:111126, 1994.

[2] D. Lee and H. Seung. "Learning the parts of objects by non-negative matrix factorization". *Nature* 401 6755:788-791, 1999.

[3] D. Lee and H. Seung "Algorithms for Non-negative Matrix Factorization." *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*. MIT Press. pp. 556-562, 2001.

[4] Y. Gao and G. Church. "Improving molecular cancer class discovery through sparse non-negative matrix factorization." *Bioinformatics*, 21:3970-3975, 2005.

[5] H. Kim and H. Park. "Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis." *Bioinformatics*, 23:1495-1502, 2007.

[6] C.-J. Lin. "Projected gradient methods for non-negative matrix factorization." *Neural Computation*, 19:2756-2779, 2007.

[7] B. Savas. "Analyses and Tests of Handwritten Digit Algorithms." Master's thesis, Mathematics Department, Linköping University, 2002.

[8] M. Mazack. "Algorithms for Handwritten Digit Recognition." Master's colloquium, Mathematics Department, Western Washington University, 2009.