Nonnegative Matrix Factorization
Supplemental material #2 to
Multi-Scale Label-Map Extraction for Texture Synthesis

Yitzchak David Lockerman[1], Basile Sauvage[2], Rémi Allègre[2], Jean-Michel Dischler[2], Julie Dorsey[1], Holly Rushmeier[1]
[1] Yale University, Computer Graphics Group, New Haven, USA
[2] ICube, Université de Strasbourg, CNRS, France

This suplemental material details our solution to the Nonnegative Matrix Factorization (NMF) problem

$$\begin{aligned} \underset{F,G}{\operatorname{argmin}} \quad & \|D^s - FG\|^2 \\ s.\,t. \quad & F \text{ and } G \text{ are left stochastic} \\ & \text{(non-negative with all columns adding up to 1)} \end{aligned} \tag{1}$$

where

- D is a large $n \times n$ sparse stochastic matrix (known).
- s is the number of steps (known).
- F is $n \times r$ and G is $r \times n$ (unknown).
- $r < n$ is the (unknown) rank of the factorization, i.e. the number of labels in our labeling problem.

We will present several algorithms of increasing complexity that compute r, F, and G from D and s. Each one will provide either a speed or accuracy improvement to its predecessor, but will require said predecessor to preform some of the calculations. The final algorithm presented (Algorithm 8) will call all of our other algorithms as subroutines. When we solve a NMF problem in the main paper, we use Algorithm 8.

1 Approximation error

Since the NMF problem in NP-hard we look for an approximate (sub-optimal) solution. The error of a solution (F, G) is given by $\|D^s - FG\|^2$ where matrix norm is Froebenius norm. However we must avoid creating a temporary dense $n \times n$ matrix D^s, as it would quickly become the bottleneck of the algorithm as n grows. We therefore rewrite:

$$\begin{aligned} \|D^s - FG\|^2 &= \operatorname{Tr}\left((D^s)^T D^s\right) + \operatorname{Tr}\left(G^T F^T F G\right) \\ &\quad - \operatorname{Tr}\left((D^s)^T F G\right) - \operatorname{Tr}\left(G^T F^T D^s\right) \\ &= \operatorname{Tr}\left((D^s)^T D^s\right) + \operatorname{Tr}\left(F^T F G G^T\right) \\ &\quad - \operatorname{Tr}\left((F^T D^s)^T G\right) - \operatorname{Tr}\left(F^T D^s G^T\right) \end{aligned}$$

Note that the first term is constant for a fixed D so it can be neglected. Also use $A \cdot B = \operatorname{Tr}(A^T B)$ the sum of elements of entry-wise product. In the end (F, G) must minimize the (normally negative) value

$$e(F,G) = \left(F^T F\right) \cdot \left(G G^T\right) - \left(F^T D^s\right) \cdot G - F \cdot \left(D^s G^T\right) \tag{2}$$

where proper order of matrix multiplication ensures no large dense matrices have to be calculated.

2 Initial factorization

To begin with, Algorithm 1 is our simplest heuristic. However, it is the only one that leads to a determination of the number of labels; a critical advantage of our method. The algorithm select the best result among a number of factorizations which are quickly generated using algorithms 2 and 3.

```
**Input:** sparse stochastic matrix $D$
**Input:** number of steps $s$
**Input:** number of runs $b$
**Output:** factorization $(F, G)$
**Output:** number of labels $r$

**repeat**
    $F \leftarrow$ Algorithm 2
    $G \leftarrow$ Algorithm 3
    **if** *it is the best attempt so far* **then** save $(F, G)$
**until** *no improvement for $b$ attempts*
$r \leftarrow$ number of columns of F

**return** $(F, G)$ and $r$
```

Algorithm 1: Computation of an initial factorization. The "best attempt" and "improvement" are measured by equation (2) and, unless otherwise noted, we used $b = 50$ runs in practice.

```
**Input:** $D$ and $s$
**Output:** $F$

list $\mathcal{L} \leftarrow$ all superpixels
matrix $F$ is empty
**while** $\mathcal{L} \neq \emptyset$ **do**
    $j \leftarrow$ random SP in $\mathcal{L}$
    $f \leftarrow D^s \delta_j$
    concatenate the vector $f$ as a new column of $F$
    **for** $i \in \mathcal{L}$ **do**
        **if** $(Df)_i < f_i$ **then** remove $i$ from $\mathcal{L}$
    **end**
**end**

**return** $F$
```

Algorithm 2: Solving for F.

Algorithm 2 is a heuristics that computes F and the number of labels (i.e. the number of columns). It is based on the (ideal) assumption that every superpixel (SP) j should be assigned as single label $c(j)$ so $G\delta_j \approx \delta_{c(j)}$. As a consequence $D^s\delta_j \approx FG\delta_j \approx F\delta_{c(j)}$, that is to say the probability map $D^s\delta_j$ is equal to column $c(j)$ in F. We use this idea in algorithm 2: we iteratively concatenate columns $f = D^s\delta_j$ to F until every superpixel i has a "good" label candidate. We consider that label $c(j)$ is a good candidate for i if an extra step D on f decreases its similarity with j: the intuition

```
Input: $D$, $s$, $F$
Output: $G$

$\lambda \leftarrow [1, \cdots, 1]^T$
while $norm$ decreases by at least $10^{-8}$ do
    Solve nnls for $G$: $2\left(F^TF\right)G = 2F^TD^s - \mathbf{1}\lambda^T$
    Normalize $G_{cj} \leftarrow G_{cj} / \left(\sum_k G_{kj}\right), \forall c, j$
    Solve least squares for $\lambda$: $\mathbf{1}\lambda^T = 2F^TD^s - 2\left(F^TF\right)G$
    $norm \leftarrow \left\|2\left(F^TF\right)G - 2F^TD^s + \mathbf{1}\lambda^T\right\|$;
end

return $G$
```

Algorithm 3: Solving for G. It uses a regular dense non-negative least squares (nnls) solver, a dense least squares solver, and Lagrange multipliers λ. Complexity is $O(nr^2)$.

is that, starting at δ_j, D first floods $c(j)$ (similarity increases for i) and then leaks outside $c(j)$ (similarity decreases for i).

Algorithm 3 solves the equation $D^s = FG$ for G in a least squares sense, using Lagrange multipliers. In particular λ is the multiplier for the constraint that $\mathbf{1}^TG = \mathbf{1}^T$. At its heart, the algorithm leaps between two optimizations: solving for G in a nonnegative least square sense, and solving for λ in a least squares sense.

3 Local improvement with gradient descent

While the factorization provided by Algorithm 2 is a good starting point, its error (measured by Equation (2)) is far from optimal. Indeed its advantages are (i) it computes the number of labels, and (ii) it tends to be "close" to near optimal solutions. We take advantage of this property by (i) fixing r, and (ii) use (F, G) as starting point for a gradient descent (see Algorithm 4).

```
Input: $D$ and $s$
Output: $(F, G)$ : the factorization
Output: $\epsilon$ : the error value

$(F_0, G_0, r) \leftarrow$ Algorithm 1 with $b = 50$
$(F, G, \epsilon) \leftarrow$ Algorithm 5 (gradient descent)
return $F$, $G$, and $\epsilon$
```

Algorithm 4: Our basic solution of our NMF algorithm. We get a starting point form Algorithm 1 and then descend from that point using Algorithm 5.

Our gradient descent algorithm (Algorithm 5) is specifically designed to work with powers of sparse matrices while being linear in n (at the expense of being quadratic in r). The algorithm lifts the stochastic matrices F and G to non-stochastic matrices $\mathbb{F}$ and $\mathbb{G}$. The derivatives of $\|D^s - FG\|^2$ with respect to F, G, $\mathbb{F}$ and $\mathbb{G}$ are given by

$$\Delta F = \nabla_F \|D^s - FG\|^2 = D^sG^T - F\left(GG^T\right) \quad (3)$$

$$\Delta G = \nabla_G \|D^s - FG\|^2 = F^TD^s - \left(F^TF\right)G \quad (4)$$

and for any $1 \le k \le n$, $1 \le l \le r$

$$\Delta\mathbb{F}_{kl} = \frac{\partial \|D^s - FG\|^2}{\partial \mathbb{F}_{kl}} = \frac{\left(\sum_i \mathbb{F}_{il}\right)\Delta F_{kl} - \sum_i \mathbb{F}_{il}\Delta F_{il}}{\left(\sum_i \mathbb{F}_{il}\right)^2} \quad (5)$$

```
Input: $D$ and $s$
Input: $(F_0, G_0)$ the initial factorization
Input: $h$ initial step size (we used $h = 2000$)
Input: $max\_it$ stopping criterion on # iterations (we used
       $max\_it = 40000$)
Input: $\delta$ stopping criterion on error decrease (we used $\delta = 10^{-20}$)
Output: $(F, G)$ the final factorization
Output: $\epsilon$ the error value

$F$ and $\mathbb{F} \leftarrow F_0$
$G$ and $\mathbb{G} \leftarrow G_0$
$\epsilon \leftarrow e(F, G)$
while # iterations $< max\_it$ do
    $\mathbb{F}' \leftarrow [\mathbb{F} + h\Delta\mathbb{F}]_+$ using equations (3) and (5)
    $\mathbb{G}' \leftarrow [\mathbb{G} + h\Delta\mathbb{G}]_+$ using equations (4) and (6)
    $F' \leftarrow$ normalize columns of $\mathbb{F}'$
    $G' \leftarrow$ normalize columns of $\mathbb{G}'$
    $\epsilon' \leftarrow e(F', G')$
    if $\epsilon - \epsilon' < 0$ then $h \leftarrow hc_1$

    else if $\epsilon - \epsilon' > c_2$ then $h \leftarrow hc_2/(\epsilon - \epsilon')$

    else if $|\epsilon - \epsilon'| < \delta$ then STOP

    else $(F, G, \mathbb{F}, \mathbb{G}, \epsilon) \leftarrow (F', G', \mathbb{F}', \mathbb{G}', \epsilon')$

end

return $F, G, \epsilon$
```

Algorithm 5: Gradient descent algorithm for nonnegative matrix factorization of stochastic matrices. Here, $[x]_+$ is x if x is positive, otherwise it is zero. Error measure is given by equation (2). The test $\epsilon - \epsilon' < 0$ prevents overshooting a minimum (we used $c_1 = .5$) The test $\epsilon - \epsilon' > c_2$ avoids to descend too quickly (we used $c_2 = 5$)

and for any $1 \leq k \leq r, 1 \leq l \leq n$

$$\Delta \mathbb{G}_{kl} = \frac{\partial \|D^s - FG\|^2}{\partial \mathbb{G}_{kl}} = \frac{\left(\sum_i \mathbb{G}_{il}\right) \Delta G_{kl} - \sum_i \mathbb{G}_{il} \Delta G_{il}}{\left(\sum_i \mathbb{G}_{il}\right)^2} \tag{6}$$

4 Convergence speed-up by projection on a sub-set of variables

When n is extremely large (more than $10,000$), the gradient descent algorithm becomes inefficient. To speed up the convergence we exploit an other observation: when $r \ll n$, the system is overdetermined. Our idea here is to project the problem on a sub-set of variables of size $m < n$, solve the sub-problem of size m, and derive a solution for the remaining $n - m$ variables.

Let $S \subset \{1, \dots, n\}$ be a sub-set of size m. For clarity we assume the rows and columns in D have been permuted so that $S = \{1, \dots, m\}$. Let $O = \{m + 1, \dots, n\}$ be the complement.

We can rewrite the matrices block-wise:

$$D = \begin{bmatrix} D_{SS} & D_{SO} \\ D_{OS} & D_{OO} \end{bmatrix}; \quad F = \begin{bmatrix} F_S \\ F_O \end{bmatrix}; \quad G = \begin{bmatrix} G_S & G_O \end{bmatrix} \tag{7}$$

We then rewrite the problem $D^s = FG$ block-wise:

$$\begin{bmatrix} D^s_{SS} & D^s_{SO} \\ D^s_{OS} & D^s_{OO} \end{bmatrix} = \begin{bmatrix} F_S G_S & F_S G_O \\ F_O G_S & F_O G_O \end{bmatrix} \tag{8}$$

The idea is to solve for the first 3 blocks only because they contain $\Theta(mn)$ elements, allowing them to be stored explicitly in dense matrix. The large block $D^s_{OO} = F_O G_O$ is ignored. Algorithm 6 works as follows:

- $D^s_{SS} = F_S G_S$ is a NMF problem which is small enough to be solved by Algorithm 4. However, the columns of D^s_{SS} need be normalized so as to add up to 1; the resulting F_S is then rescaled.
- $D^s_{SO} = F_S G_O$ can be solved for G_O using Algorithm 3.
- $D^s_{OS} = F_O G_S$ can be solved for F_O using a similar algorithm, modified to ensure that the rows of the full F sums to one (see Algorithm 7).

The blocks of D^s can be computed efficiently through the recurrence relationship, for $1 < i \leq s$:

$$\begin{aligned} D^i_{SS} &= D_{SS} D^{i-1}_{SS} + D_{SO} D^{i-1}_{OS} = D^{i-1}_{SS} D_{SS} + D^{i-1}_{SO} D_{OS} \\ D^i_{OS} &= D_{OS} D^{i-1}_{SS} + D_{OO} D^{i-1}_{OS} \\ D^i_{SO} &= D^{i-1}_{SS} D_{SO} + D^{i-1}_{SO} D_{OO} \end{aligned} \tag{9}$$

As detailed in Algorithm 6, this procedure first needs m to be chosen such that $r < m$. We start by defining r using Algorithm 1 with a small number b of runs because we only care about r. In simulated data $m = 10r$ seems to be sufficient and we use $m = 30r$ in practice. Then we run the projected algorithm if $m < n$ otherwise the gradient descent is sufficient. This algorithm is much faster than gradient descent when n is large. However it does not perfectly reach the local minimum, so in the end we run a gradient descent which converges in substantially fewer iterations.

```
Input: $D$ and $s$
Output: $(F, G)$ : the factorization
Output: $\epsilon$ : the error value

$r \leftarrow$ Algorithm 1 with $b = 2$
if $m = 30r \geq n$ then run Algorithm 4 and return

$S \leftarrow$ select $m$ disjoint random indexes
$O \leftarrow$ complement of $S$
Extract $D_{SS}$, $D_{SO}$ and $D_{OS}$ from $D$
Compute $D^s_{SS}$, $D^s_{SO}$ and $D^s_{OS}$ using equation (9)
 $\Lambda \leftarrow diag\left(\mathbf{1}^T D^s_{SS}\right)$
$(F_S, G_S) \leftarrow$ Solve $F_S G_S \approx \Lambda^{-1} D_{SS}$ (Algorithm 4)
$F_S \leftarrow \Lambda F_S$
$G_O \leftarrow$ Solve $D^s_{OS} \approx F_S G_O$ (Algorithm 3)
$F_O \leftarrow$ Solve $D^s_{OS} \approx F_O G_S$ (Algorithm 7 with $v = \mathbf{1} - F_S^T \mathbf{1}$)
$(F, G, \epsilon) \leftarrow$Algorithm 5 initialized at ( $\begin{bmatrix} F_S \\ F_O \end{bmatrix}$ , $\begin{bmatrix} G_S & G_O \end{bmatrix}$ )

return $(F, G, \epsilon)$
```

Algorithm 6: Projection algorithm for nonnegative matrix factorization of stochastic matrices.

```
Input: $D$, $s$, $G$, and $v$
Output: $F$

$\lambda \leftarrow [1, \cdots, 1]^T$
while norm decreases by at least $10^{-8}$ do
    Solve nnls for $F$: $2\left(GG^T\right) F = 2D^s G^T - \mathbf{1}\lambda^T$
      $F_{jc} \leftarrow v_c F_{jc} / \left(\sum_i F_{ic}\right), \forall c, j$
    Solve least squares for $\lambda$: $\mathbf{1}\lambda = 2D^s G^T - 2F\left(GG^T\right)$
    $norm \leftarrow \left\|2F\left(GG^T\right) - 2D^s G^T + \mathbf{1}\lambda^T\right\|$
end

return $F$
```

Algorithm 7: Solving for part of F. It uses a regular dense nonnegative least squares (nnls) solver, a dense least squares solver, and Lagrange multipliers λ. Complexity is $O(nr^2)$.

5 Boosted algorithm

The algorithms we have presented so far rely on random starting values. If these values turn out to be poor, the results of the algorithm will be poor as well. As such, we run the full algorithm set multiple times (5 times in practice). We then combine them using a custom boosting framework detailed in Algorithm 8.

At each run we obtain a pair (F, G). The results are combined into larger matrices:

$$\mathbf{F} = \begin{bmatrix} F_0 & F_1 & \dots & F_t \end{bmatrix} \quad \text{and } \mathbf{G} = \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_t \end{bmatrix} \tag{10}$$

which are normalized into stochastic matrices. Intuitively, $\mathbf{F}$ and $\mathbf{G}$ are factorizations of the the diffusion matrix with overlapping textures. We can measure the overlap by looking at the correlation between the different texture nodes given by $\mathbf{W} = \mathbf{GF}$. We can remove the overlap and produce a final texture set by preforming the factorization $\mathbf{W}^2 = \mathcal{FG}$ using Algorithm 4 with $b = 5$. Finally, we produce our final $(F, G) = (\mathbf{F}\mathcal{F}, \mathcal{G}\mathbf{G})$.

```
**Input:** $D$ and $s$
**Output:** $(F, G)$

$\mathbf{F} \leftarrow n \times 0$ matrix
$\mathbf{G} \leftarrow 0 \times n$ matrix
for $1 \dots 5$ do
    $(F, G) \leftarrow$ Algorithm 6
    Append the columns of $F$ to $\mathbf{F}$
    Append the rows of $G$ to $\mathbf{G}$
end
$\mathbf{W} \leftarrow \mathbf{GF}$
$(\mathcal{F}, \mathcal{G}) \leftarrow$ Solve $\mathbf{W}^2 = \mathcal{F}\mathcal{G}$ using Algorithm 4
$F \leftarrow \mathbf{F}\mathcal{F}$
$G \leftarrow \mathcal{G}\mathbf{G}$

return $(F, G)$
```
Algorithm 8: Boosted algorithm for nonnegative matrix factorization of stochastic matrices.