Lab 5 - Non-negative Matrix Factorization

Submission Instructions

- Submission Deadline: Wednesday (3rd October 2018) at 1.30pm.
- Please submit the lab in any one of these forms (we *will* be strict about students adhering to these formats -

1. A completed `Lab_5.mlx` with the corresponding `nmf.m`, `ssnmf.m`, `compute_objective.m`, `compute_objective_ss.m` files.

2. A completed `Lab_5.m` file **and** an exported PDF with the corresponding `.m` files as above. To export your `.m` file to a PDF, use the Publish Tab in the Editor. It is **necessary** to label all the figures you include, if they have not been labelled in the helper code already.

The ORL Faces Dataset

We use the ORL Face database for this assignment, which consists of 400 images for 40 people, each of size 112×92. These images were taken at different times, with varying lighting and for different facial expressions. All faces are in an upright position with a frontal view, with a slight left-right rotation. To use this dataset, we perform some pre-processing on them, listed here -

1. We use only the train split for this data (the first 9 images per person).
2. We construct a data matrix `matX` of size 10304×360 by flattening all the faces.
3. `matX` is divided by the max value present in the images to normalize the data and avoid overflow issues, giving us the final data matrix V.

```
clear all; close all;
d1 = 112; d2 = 92; d = d1*d2;
num_images = 9; num_people = 40;
images = cell(num_people, num_images);
matX = zeros(d, num_people*num_images);

count = 1;
for i = 1:num_people
    for j = 1:num_images
        % filename = sprintf('/your/local/path/to/orl_faces/Train/s%i/%i.pgm', i, j);
        filename = sprintf('./../orl_faces/Train/s%i/%i.pgm', i, j);
        img = double(imread(filename));

        matX(:,count) = reshape(img, d, 1);
        count = count+1;
    end
end

V = matX/max(matX(:));
```

Performing NMF

To perform NMF, we want to decompose the matrix $V = BW$. To do so, we'll follow the following steps

1. Create an NMF function `nmf(V, rank, max_iter, lambda)`
2. Initialize B and W randomly, and make sure W has unit-sum columns (each column should sum to 1).

3. Calculate the initial objective (as seen in the lecture for KL Divergence). It will be helpful to define a function `compute_objective(V, W, B)` that returns the objective value.
4. Perform the iterations $B = B \otimes \frac{\left(\frac{V}{BW}\right)W^T}{1W^T}$ and $W = W \otimes \frac{B^T\left(\frac{V}{BW}\right)}{B^T 1}$, where $\otimes$ specifies element-wise multiplication and all divisions are element-wise division.
5. Calculate the new objective function value.
6. Stopping Criteria: Stop when the absolute difference of objective values is smaller than or equal to λ (or) the max number of iterations has been reached.

Notes -

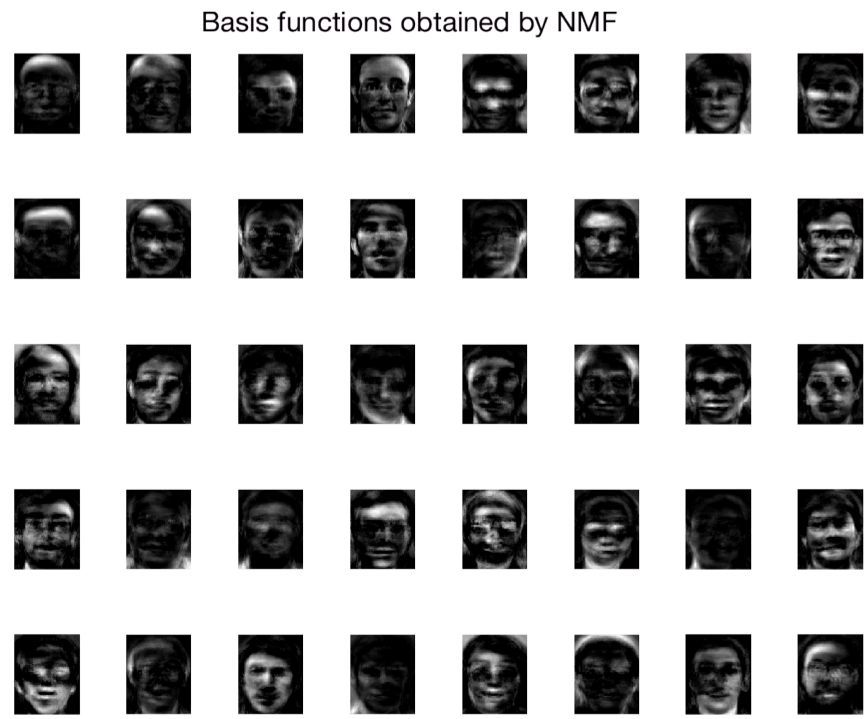
- Boilerplate for `nmf(V, rank, max_iter, lambda)` has been provided in `nmf.m`.
- Boilerplate for `compute_objective(V, W, B)` has been provided in `compute_objective.m`
- The notation $1W^T$ and B^T1 are another way of writing the sum of each column of W and B. What these denominator terms are doing are normalizing the columns of W and B such that they have unit-sum. You should ensure the columns of your B and W normalize to 1.

Validation on the ORL Faces Dataset

Step 1: Output the new bases and weights and plot Calling your NMF function on the data matrix X with parameters `rank = 40`, `max_iter=500`, and `lambda=0.001` will look something like

```
[B, W, obj, k] = nmf(V, 40, 500, 0.001);
```

```
figure;
suptitle('Basis functions obtained by NMF');
for k = 1:40
  subplot(5, 8, k);
  imagesc(reshape(B(:,k), d1, d2));
  colormap gray; axis image off;
end
```



Step 2: Compare your results with MATLAB's predefined NMF function

```
opt = statset('MaxIter', 500, 'Display', 'final');
[B, W] = nnmf(V, 40, 'options', opt, 'algorithm', 'mult');
```

```
figure;
suptitle('Basis functions obtained by MATLAB NMF Function');
for k = 1:40
  subplot(5, 8, k);
  imagesc(reshape(B(:,k), d1, d2));
  colormap gray; axis image off;
end
```

Performing NMF with added sparsity constraints

The process for performing sparse NMF is the same as above, with a few changes to Step 4 (the update rules).

1. Create a sparse NMF function `ssnmf(V, rank, max_iter, lambda, alpha, beta)`
2. Initialize B and W randomly, and make sure W has unit-sum columns (each column should sum to 1).
3. Calculate the initial objective. It will be helpful to define a new function for sparse NMF, `compute_objective_ss(V, W, B, alpha, beta)` that returns the objective value.
4. Perform the iterations $B = B \otimes \dfrac{\left(\dfrac{V}{BW}\right)W^T}{1W^T + \beta}$ and $W = W \otimes \dfrac{B^T\left(\dfrac{V}{BW}\right)}{B^T 1 + \alpha}$, where $\otimes$ specifies element-wise multiplication and all divisions are element-wise division.

5. Calculate the new objective function value.
6. Stopping Criteria: Stop when the absolute difference of objective values is smaller than or equal to λ (or) the max number of iterations has been reached.

Notes -

- Boilerplate for `ssnmf(V, rank, max_iter, lambda)` has been provided in `ssnmf.m`
- Boilerplate for `compute_objective_ss(V, W, B, alpha, beta)` has been provided in `compute_objective_ss.m`
- Here, the notation $1W^T + \beta$ and $B^T 1 + \alpha$ are normalizing the columns of W and B by the sum of column elements plus β, and the sum of column elements plus α respectively. You should ensure you perform this normalization.

Validation on the ORL Faces Dataset

Step 1: As before, output the new bases and weights and plot them by calling your SSNMF function on the data matrix V with parameters `rank = 40`, `max_iter=500`, `lambda=0.001`, `alpha=100`, and `beta=1` will look something like

```
[B, W, obj, k] = ssnmf(V, 40, 500, 0.001, 100, 1);
```

```
figure;
suptitle('Basis functions obtained by Sparse NMF');
for k = 1:40
  subplot(5, 8, k);
  imagesc(reshape(B(:,k), d1, d2));
  colormap gray; axis image off;
end
```

Basis functions obtained by Sparse NMF