PKU Summer Lectures on Neural Computation
Exercise 2: Principal Component Analysis – Hebbian Learning

Credit: 1/4 of the total exercise credit
Out: 7/16, 2019.
Due: 7/20, (Sat), 2019.

Grading and policy:

The same policy also applies to all the four exercises, including exercise 1 due Friday by class.

Part 1: You are free to work with your classmates so that you can help each other, essentially to learn as much Matlab as you could in two days to do this assignment. Part 1 will count for 80 % of the credit of this assignment. For this part, you only need to submit your one-page write-up.

Part 2: You should come up with your idea and do it by yourself. You can work with one or two students, but each one should work on a different idea. As the grade will be based on how novel and how interesting your idea and implementation is. For example, if three students submitted the same idea, then the novelty credit will be divided by 3. For this part, you need to submit all your codes and data, so that we can run it and get the same answers.

Part 1. Principal component analysis of natural images

We give you a step-by-step guide to computing principal components, which really is done in a few lines in Matlab! **Note: We have provided you with part1_.m solution, so your main mission is just learn how to run it, change parameters, experiment and make some observations. Make at least 3 interesting observations to earn you the full credit for this part.**

For an 8 x 8 8-bit greyscale image patch, if all the pixels are allowed to vary independently, there are 256^{64} possible images. Most of these images are just noise. Natural images live in a very small subspace in this 64 dimensional space. Hebbian learning rule allows neurons to learn correlated structures in the input patterns via principal component analysis to a first approximation. As a result, neurons in the brain can code images more efficiently by learning a set of features to code them efficiently. In this exercise, we will study how Hebbian learning rule can discover the most important 'letters' in the alphabet for representing visual images.

The Oja rule for implementing Hebbian learning is effectively implementing the principal component analysis, which is a well-studied statistical modeling technique. Note, given the time constrained, we will skip the explicit implementation of Oja rule or Sanger's rule but you can try that for part of Part 2 of your assignment. Principal component analysis allows you to discover the important dimensions of signal variations in the input so that you can represent the signals using fewer dimensions. For example, a 64 dimensional vector space normally requires 64 basis to represent any arbitrary signal in this 64 dimensional space. We would like to determine the top ten basis vectors, $X_1 - X_{10}$ for representing any arbitrary signal in this 64 dimensional space i.e. any 8x8 image patches in natural outdoor imagery. In this representation, an 8x8 image patch is approximated

by a linear combination of the chosen bases vectors:

$$Y \approx \hat{Y} = a_1 X_1 + a_2 X_2 + ... + a_{10} X_{10}$$

where, Y is the original 8x8 patterns. $\hat{Y}$ is the approximation of Y.

This approximation reduces the representational size from 64 pixel values to a few values a_i coefficients (a relatively small amount of additional memory is required for storing the basis vectors once), achieving dimensional reduction.

Now, to select the best set of basis vectors for this approximation, we choose the X_i's that minimize the sum of squared error over the conglomerate of n 8 x 8 subregions, Y_j, collected from a set of representative outdoor images.

$$\min \sum_{j=1}^{n} \|Y_j - \hat{Y}_j\|^2 = \min \sum_{j=1}^{n} \|Y_j - (a_{1,j} X_1 + a_{2,j} X_2 + ... + a_{5j} X_5)\|^2$$

The set of X_i that minimize this expression are given by the first 5 principal components of the sample covariance matrix, $\hat{C}$, which are the 5 eigenvectors with the largest eigenvalues

$$\hat{C} = \frac{1}{n} \sum_{j=1}^{n} (y_j - \bar{m}_y)(y_j - \bar{m}_y)^T \qquad \bar{m}_y = \frac{1}{n} \sum_{j=1}^{n} y_j$$

where y_j is a 64 x 1 vector obtained by stacking the columns of the 8x8 image patch, Y_j, on top of each other. $\bar{m}_y$ is the 64 x 1 vector which is the sample mean of all the image patches. (The superscript "T" indicates the vector transpose operation).

In this exercise, we will provide a set of 10 "representative" images of natural scenes: im1.tif - im10.tif. Your job is to compute the basis vectors $X_1, ..., X_{10}$, from this data set using principal component analysis. This will involve performing the following steps:

1. Compute the sample covariance matrix using the following steps:

(a) Collect 500 8x8 sample patches from each of the provided natural scenes and store them as a 64xn matrix where n is the total number of patches. To do this, convert each patch into a 64x1 column vector and concatenate all the patch vectors together. This can be done using the provided extract_patches.m code which randomly selects patches from a given image.

To load an image use the command imread.

```
img = imread('im.1.tif');
```

This will load the image im.1.tif into a matlab matrix. Note that the type of this matrix probably will probably be of type uint8 and doing math operations on a matrix of this type will result in

rounding and clipping. The provided function extract_patches, as a byproduct, will convert these images to double for you so you will not have to worry about this fact.

To save an image use the command imwrite. Given image data in variable img, if we wish to save it as a png file we would invoke the command as:

```
imwrite(img, 'img.png', 'PNG');
```

Other formats are also supported such as jpeg and tiff. Inspect the help output for the command to see additional formats and options.

To extract random 500 8x8 patches from an image, call `X = extract_patches( img, 8, 500)` where X is a 64x500 matrix.

Make sure to save these patches because they will be used in following problems.

(b) Calculate the sample covariance matrix using the MATLAB command cov. Make sure to use the flag $\hat{C}$ `= cov(X, 1)` in order to normalize by n rather than the MATLAB default of n+1.

2. Compute the principal components (eigenvectors with the largest eigenvalues) of $\hat{C}$. Print these patterns and indicate their ordering and corresponding eigenvalues.

(a). The matlab routine for eigenvector decomposition does not arrange eigenvectors in descending order of eigenvalue magnitude. However, the singular value decomposition routine (svd) puts the singular vectors in the order of decreasing singular value. The svd of $\hat{C}$ is given by the matrix product:

$$USV^T$$

and is equivalent to the eigenvector decomposition. The columns of U (or the rows of V), correspond to the eigenvectors of $\hat{C}$ and the singular values in S correspond to the eigenvalues of $\hat{C}$.

Note that in general, given a $n \times k$ matrix X, there is a unique decomposition of $X = USV^T$, where U is a $n \times r$ column-orthogonal matrix, V^T is a $r \times k$ orthogonal matrix, and S is a $r \times r$ diagonal matrix with the elements in the main diagonal being nonnegative and in decreasing order. r is the rank of the matrix X. Note: if a and b are orthonormal column vectors, then $a * b^T = 0$, and $a * a = 1$.

Note in this case, $n = 64$ and $n = k = r$.

(b). The eigenvectors will be computed as 64x1 column vectors. These vectors will have to be unstacked to express and display them as 8x8 patterns. To change a 64x1 matrix A to an 8x8 patch, you would use `reshape(A, 8, 8)`. To display these patterns as images, you have to enlarge the patterns (by at least a factor of 4) so that the image is a 64x64 image for easier visualization. It may also be necessary to specify the actual range of intensity variation, e.g. `imshow(eig2,[-0.5,0.5])` or use `imagesc` and `colormap gray`. You may want to use the following contrast normalization trick

```
c = c - min(min(c)); %subtract each pixel by the global min of the image.
c = c / max(max(c)); %divide each resulting pixel by the max of the image.
im = c;
imshow(im);
```

to normalize your images. Essentially, the 8 x 8 image patterns should resemble those shown in the lecture slides.

Note that SVD actually return to you all 64 principal components. The one with the highest eigenvalues are the ones that capture the most variance.

3. Suggested Questions:

The program will plot 64 eigenvalues in descending order. Plot the 64 principal components as image patches in descending order left to right, top to bottom. Be sure to use contrast normalization so you can see the images. Are 5 principal components good enough for reconstruction? How many do you actually need to have good quality reconstruction? To answer this question, we do the following,

Divide im11.tif into 8 x 8 non-overlapping block of images. Colon notation can be used to extract each sub-matrix from the large image. For example, A(1 : 8; 1 : 8) selects the 8x8 upper left hand corner of the matrix A. Another method would be to use the MATLAB command blckproc or blockproc if you have access to the Signal Processing Toolbox. Project each image patch j onto the principal components (PC basis) to obtain the coefficients $a_{1,j}...a_{10,j}$, by taking the dot product between the image patch with each PC.

Now, you can synthesize each image patch back based on its projected coefficients using

$$\hat{Y_j} = (a_{1,j}X_1 + a_{2,j}X_2 + ... + a_{10,j}X_{10})$$

To see that, you could print out the synthesized image based on the first 5 principal components only. The program does that now, you can change the number of components used to explore. This would represent a compression of 64 (because each patch has 8 x 8 pixels) to 10 principal component values. In your report, make some observations of what the program produces and submit the reconstruction as your part 1 answer.

Part 2: Competition (20 %) Now, I have taken out questions from our original assignment, this gives you the freedom to ask questions yourself and answer them. You can do that for this part, but there will probably be signifiant overlap with your classmates. One question could be whether the learned principal components from one image is transferable to other images. PCA can be applied to many problems to extract important correlational structures from the data. For this code, an obvious thing to do is to try it on some different sets of data and see how the principal components change as a function of the data. You can also use it to sound signals, on financial data, sky is the limit. Try to ask some interesting questions, and come up with some interesting discoveries. You are allowed to submit multiple "papers", up to 3, each is limited to one page in write-up, with codes and data set attached.