15-386/686 Neural Computation
Problem Set 2: Hebbian Learning

Credit: 12 points.
Out: 2/8, 2023.
Due: 11:59 p.m. 2/22, 2023.

Collaboration and submission guidelines: You are allowed to collaborate with one partner on Part 1 and Part 2 of this problem set but not the creative exploration part (Part 3). Submit your assignment to Gradescope and CANVAS (zip file of report and codes). If you work in a team, put down your partner's name on the first page as well. Report should be type-written. Submit top level code for each part as part1.m, part2.m and part3.m along with all supporting scripts, functions, and .mat files. Running the top level code should generate all the results and figures included in your report. In your zip file submission, you should include your report, the codes and generated results (graphs).

Since there are aspects of image data manipulation that might be difficult for people who are not familiar with Matlab, we have made available example codes in the starter_code folder. By filling in a few lines correctly in each of these parts (see comments in the codes), you should have the basic program running. If you have limited coding experience, we recommend that you start the homework early so that you have enough time to debug and ask TA for help.

Part 1. Principal component analysis of natural images (6 points)

We are giving you a step-by-step guide for computing principal components, which is done in just a few lines in Matlab! **Note: part1_hint.m in the assignment folder provides most of the code and leaves some lines to be filled. There are multiple ways to do principal component analysis in Matlab. You can write your own codes and use your own way to compute the principal components as long as you can answer the questions.**

Given an 8 x 8 8-bit greyscale image, there is a total of 256^{64} possible images. Most of these images are just noise. Natural outdoor images live in a very small subspace in this 64-dimensional space. Recall from lecture that the Hebbian learning rule allows neurons to learn correlations in the input patterns in a similar fashion as conducting principal component analysis. In this assignment, we will explore how neurons in the brain can learn a set of features to code natural outdoor images efficiently.

It has been shown that Hebbian learning using the Oja's rule is equivalent to performing principal component analysis. Principal component analysis, or PCA, is a dimensionality reduction technique that allows you to represent signals that live in high dimensions in lower dimensions. An 8 x 8 image patch lives in a 64-dimensional space and are normally represented by 64 basis vectors. Suppose we want to find the smallest number of neurons to represent an image patch in natural image, we can use PCA or Hebbian learning with Oja rule.

Suppose we have 10 neurons $X_1, ...X_{10}$, then an 8x8 image patch (Y) is approximated by a linear combination of the chosen basis vectors ($\hat{Y}$):

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

This approximation reduces the representational dimensions from 64 to 10.

In order to select a set of best 10 basis vectors for this approximation, we choose the vectors 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.

$$\min \sum_{j=1}^{n} \|Y_j - \hat{Y}_j\|^2 = \min \sum_{j=1}^{n} \|Y_j - (a_{1,j} X_1 + a_{2,j} X_2 + ... + a_{10,j} X_{10})\|^2$$

The set of X_i that minimize this expression are given by the first 10 principal components of the sample covariance matrix, $\hat{C}$, which are the 10 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) Extract 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.

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)`. X should be 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 $\hat{C}$ by following the steps below.

(a). Use the command svd to perform eigenvector decomposition. The matlab function 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 $a * a = 1$.

(b). Now that we have the eigenvectors, we want to display them as 8x8 image patterns. To do so, reshape individual 64x1 vectors back into an 8x8 patch. You only need to do so for the top 10 principal components. Recall that the top principal components correspond to the eigenvectors with the greatest eigenvalues.

To display these patterns as images, you may 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. These 8 x 8 image patterns should resemble those shown in the lecture slides (ADD LECTURE AND SLIDE NUMBER).

Note that SVD actually returns all 64 principal components. You have to pick the top 10 based on the magnitude of the eigenvalues.

3. Answer the following questions:

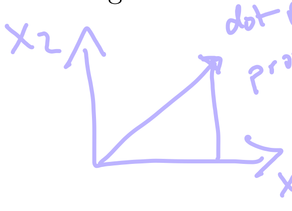
(a). (1 point) Plot 64 eigenvalues in descending order. Plot the first 10 principal components (eigenvectors) as 10 image patches in two rows. Be sure to use contrast normalization so you can see the images. Save these components, as we will evaluate them against what Hebbian learning rule will give us in Part 2.

What is the minimum number of principal components required to capture 90 percent of the variances in the data? What is the minimum number of principal components required to capture 99 percent of the variances in the data? Here, we assume the eigenvalue of the principal component corresponds to the variance captured by the principal component. The trace of matrix S (sum along the diagonal) is the total amount of variance. We will verify this assumption in (d).

(b). (1 point) Now, we have learned the principal components from the samples of 10 images. We want to see how ell they can be used to represent the test image that is not in the training set. Divide im11.tif into non-overlapping 8 x 8 chunks or blocks. 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:

dot product is projectio



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

where $a1, j$ is the projection of image j on principal component X_1.

Print out the synthesized image based on the first 10 principal components only. Also print out the images based on just the first 1 PC, the first 6 PCs and the first 8 PCs as well. How do the reconstructed images look? Reconstruct the image using the number of principal components required for capturing 90 percent percent of the variance (see(a) –how does that synthesized image look? Repeat this for the number of principal components for capturing 99 percent of the variance. Include all synthesized images and the original image in your report, report the number of principal components required to capture 90 and 99 percentage of the variances respectively.

(c). (1 point) Convert both the original images and the synthesized images into uint8 (one byte) type, i.e. the pixel value ranging from 0 to 255 (hint: look up the function `uint8`). Compute and report the average error per pixel of each of the synthesized images in (b) with the original image using the following equation:

$$MPE = mean_i \frac{\sum_{n,m} |\hat{Y}_i(n,m) - Y_i(n,m)|}{\sum_{n,m} |Y_{n,m}(t)|}$$

where i is index of an image patch, $mean_i$ is the mean computed over all image patches in the image. m and n are indexes of the 2D array of each image patch, ranging from 1 to 8 pixels in our case. $\hat{Y}$ is the synthesized images generated in the previous question.

(d) (1 point) Consider the 10 PC case. Plot a histogram of the projection of 1000 samples draw from the 10 training images onto each one of the top 3 PCs. Report the three histograms. Are these distributions Gaussians? Compute and report the standard deviations. What is the relationship between the standard deviation and the eigenvalues in the S matrix associated with the corresponding eigenvector (Principal component) returned by SVD above in (a) ?

(e) (1 point) In (d), you obtain three corresponding sets of 1000 coefficients, one for each PC. Compute the Pearson correlation between each of the three possible pairs of the three sets of coefficients. What do you observe? Is this consistent with your expectation?

(f) (1 point) Compute the first 10 principal components of the natural images, but now using image patches of size 16 $\times$ 16. Plot the 64 of the eigenvectors with the highest eigenvalues in descending order. Plot the first 10 principal components as 10 image patches in two rows. Be sure to use contrast normalization so you can see the images. Compare and contrast with the results in (a). How many principal components are needed to capture 99 % of the variance? Which is more efficient (use fewer numbers) – 8 x 8 PCs vs 16 x 16 PCs?

Part 2: Hebbian learning of natural images (5 points)

Note: We have provided you with part2_hint.m in the assignment folder.

Hebbian learning involves adjusting weights between learning nodes to better represent the relationship between them. This rule can be summarized as "cells that fire together, wire together" and is often modeled as:

$$\Delta \mathtt{w_i} = \eta \mathtt{x_i y}$$

Where the change in the strength of the synaptic connection, w_i, is equal to the product of the learning rule (η), the input (x_i), and the post synaptic response (y). However, this learning rule is unstable, as the weights will grow without bound. Different learning rules have been proposed (see exercise in Section (d) below). For example, Oja proposed a quadratic decay term as follows:

$$\Delta \mathtt{w_{ij}} = \eta(\mathtt{y_j x_i} - \mathtt{y_j^2 w_{ij}})$$

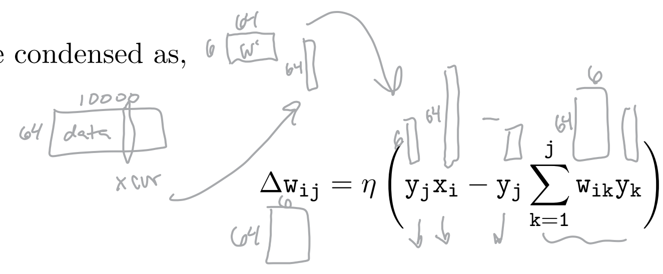
Furthermore, Oja's rule can be applied sequentially, which is called the Sanger's rule. First, compute the first neuron's weight and subtracted what this neuron can explain from the original signals. Then, learn the second neuron's weight to model the residuals and repeat the same process to obtain the subsequent neurons as specified by the following equation. This can be shown to perform principal component analysis (with a sequential algorithm), for neuron j,

as

$$\Delta \mathtt{w}_{\mathtt{ij}} = \eta \left(\mathtt{y}_{\mathtt{j}}(\mathtt{x}_{\mathtt{i}} - \sum_{\mathtt{k}=1}^{\mathtt{j}-1} \mathtt{w}_{\mathtt{ik}}\mathtt{y}_{\mathtt{k}})) - \mathtt{y}_{\mathtt{j}}^2 \mathtt{w}_{\mathtt{ij}}) \right)$$

which can be condensed as,

$$\Delta \mathtt{w}_{\mathtt{ij}} = \eta \left(\mathtt{y}_{\mathtt{j}}\mathtt{x}_{\mathtt{i}} - \mathtt{y}_{\mathtt{j}} \underbrace{\sum_{\mathtt{k}=1}^{\mathtt{j}} \mathtt{w}_{\mathtt{ik}}\mathtt{y}_{\mathtt{k}}} \right)$$



In this problem, we will use Sanger's rule to perform principal component analysis on the same data as in part 1 and then compare the two sets of results.

1. Compute the mean of the sample patches then subtract the mean from each patch. Keep the data in the form of a $64 \times n$ matrix.

2. Use Sanger's rule to learn the first 6 principal components of the data. Use $\eta = 2 * 10^{-10}$. Use the following instructions as a guide:

(i) Initialize your weights to random values using `w = randn(N, L)` where N is the dimensionality of your data and L is the number of principal components you want to find.

(ii) Compute the neurons' output for all data. This can be done in one line: `Y = w'*X` where X is the matrix of sample patches after subtracting the mean.

(iii) Compute Δ`w` using the Sanger's rule equation given above (note that this can be sped up with vectorization).

(iv) Update the weight vector. Again this can be done in a single line: `w = w + dw`

(v) This is an iterative method so repeat steps (ii) - (v) until `w` converges. Each column of `w` will eventually become a principal component of your sample patch data. It may be helpful to display the columns of `w` every once in a while so you can watch how `w` changes and check that the code is working correctly.

Depending on your implementation, this code may become very computationally expensive and take a while to run. Try to avoid using `for` loops wherever possible in order to save time. A heavily optimized code can still take 10 minutes to run. It is advisable to display each component once it is learned so you can monitor the progress of the learning process, and catch bugs early.

After your code converges, plot these principal components as 8x8 patches and include it in your report. Compare them with the results from part 1. Are there any differences?

To save your time, you only need to learn the first 6 components (neurons).

What to hand in:

(a) (2 points) As in part 1, include the first 6 principal components (each in the form of 8 x 8 image patches) and the plots in your report. Do the Sanger rule learn the same principal components as that obtained by SVD in Part 1? Compare the two sets of components. What are the differences, if any?

(b) (1 point) Reconstruct the image im11.tif using these 6 neurons' receptive fields as in part 1. Do not forget to add back the mean if you are reusing the patches. Compare with the results reconstructed from the first 6 principal components obtained in part 1—is the reconstruction better or worse, qualitatively?.

(c) (1 point) Repeat Part 1c for the image synthesized from 6 components. Calculate the mean error between the reconstructed image based on the 6 principal components and the original image. Compare with the results of Part 1c—is the reconstruction better or worse, quantitatively?

(d) (1 point). Provide a general discussion, reflection and observations on what you learned in Part 1 and Part 2 so far. Can you explain intuitively what the neurons are learning ? And why that is a good thing for neurons to learn? There is no fixed answer, but think about what we discussed in the lectures, use your imagination and logical reasoning. Limit yourself to half an page.

Part 3: Creative Exploration (1 points) (Potential Bonus: up to 2 points) Here, you should explore the topics or methods covered in this problem set, i.e. the use of PCA or Hebbian Learning, or on topics related to synaptic plasticity in general such as STDP, or other firing-rate based learning rules such as BCM rule. This exploration should begin with a question, and then you perform an experiment to answer this question. The project could involve carrying out a computational experiment or carrying out in-depth research to answer this question. Limit the write-up to this part to no more than 2 page. Exceptionally outstanding and creative work will be rewarded with bonus points, up to 2 points.