HW Assignment 3 (due by 9:00am on Feb 28)

1 Theory (100 points)

1. **[Linear Auto-Encoder, 40 points]** Consider a linear-encoder i.e., a 3 layer neural network with \( k \) hidden units in which (1) the output units are trained to replicate the input values, and (2) the activation function is the identity function. Assume the network is trained to minimize the sum-of-square-errors on the training dataset (no regularization or sparsity terms).

   (a) Derive the equations for the forward propagation algorithm. Use the notation introduced in class.

   (b) Derive the equations for the backpropagation algorithm. Use the notation introduced in class.

2. **[Whitening, 20 points]** Let \( X \) be a dataset of \( m \) \( D \)-dimensional samples and \( Y \) be the PCA whitening of \( X \). Let \( R \) be an arbitrary orthogonal matrix. Prove that the sample covariance matrix of the rotated data \( RY \) is the identity matrix.

3. **[Covariance Matrix, 20 points]** Let \( X \) be a dataset of \( m \) \( D \)-dimensional samples and \( \Sigma \) its sample covariance matrix. Prove that all eigenvalues of \( \Sigma \) are positive real numbers.

4. **[PCA, 20 points]** Prove that PCA is invariant to the scaling of the data, i.e. it will return the same eigenvectors regardless of the scaling of the input. More formally, if you multiply each feature vector \( x^{(i)} \) by some positive number (thus scaling every feature in every training example by the same number), PCA’s output eigenvectors will not change.

2 Implementation (100 points)

Download the skeleton code from http://ace.cs.ohio.edu/~razvan/courses/dl6890/hw/hw03.zip. Implement the PCA, PCA whitening, and ZCA whitening, following the steps explained in this section. Make sure that you organize your code in folders as shown in the table below.

```
dl6890
  hw03
    pca_2d.py
    pca_image.py
    figure_xx.jpg
    displayNetwork.py
    IMAGES_RAW.mat
    sampleIMAGESRAW.m
    pcaData.txt
```

Write code only in the 2 files indicated in bold. Save each figure displayed by the code into a jpg file `figure_xx.jpg`. The first part of the exercise should generate 6 figures.
figure_01.jpg to figure_06.jpg, whereas the second part of the exercise should generate 5 figures figure_07.jpg to figure_11.jpg.

2.1 PCA and Whitening in 2D (50 points)

Coding effort: my implementation has 9 lines of code

In this exercise you will implement PCA, PCA whitening and ZCA whitening, as described in the Lecture 3. The only file you need to modify is pca_2d.py. Implementing this exercise will make the next exercise significantly easier to understand and complete.

Step 0: Load data: The starter code contains code to load 45 2D data points. When plotted using the scatter function, the results should look like in Figure 1(a).

Step 1: Implement PCA: In this step, you will implement PCA to obtain xRot, the matrix in which the data is ”rotated” to the basis made up of the principal components. You should make use of NumPy’s svd() function here.

Step 1a: Finding the PCA basis: Find $u_1$ and $u_2$, and draw two lines in your figure to show the resulting basis on top of the given data points. Your figure should look like in Figure 1(b).

Step 1b: Check xRot: Compute xRot, and use the NumPy scatter() function to check that xRot looks as it should, which should be something like in Figure 2(a).

Step 2: Dimensionality reduction: In the next step, set k, the number of components to retain, to be 1. Compute the resulting xHat and plot the results, as in Figure 2(b).

Step 3: PCA Whitening: Implement PCA whitening using the formula from Lecture 3. Plot xPCAWithe, and verify that it looks like in Figure 3(a).

Step 4: ZCA Whitening: Implement ZCA whitening and plot the results. The results should look like in Figure 3(b).
2.2 PCA and Whitening on natural images (50 points)

Coding effort: my implementation has 19 lines of code

In this exercise, you will implement PCA, PCA whitening and ZCA whitening, and apply them to image patches taken from natural images. The only file you need to modify is pca_image.py.

Step 0a: Load data: The starter code contains code to load a set of natural images and sample 12x12 patches from them. The raw patches will look something like in Figure 4(a). These patches are stored as column vectors in the 144 x 10,000 array x.

Step 0b: Zero mean the data: First, for each image patch, compute the mean pixel value and subtract it from that image, this centering the image around zero. You should compute a different mean value for each image patch.

Step 1a: Implement PCA: In this step, you will implement PCA to obtain xRot, the matrix in which the data is ”rotated” to the basis comprising the principal components. Note that in this part of the exercise, you should not whiten the data.
**Step 1b: Check covariance**: To verify that your implementation of PCA is correct, you should check the covariance matrix for the rotated data $x_{rot}$. PCA guarantees that the covariance matrix for the rotated data is a diagonal matrix (a matrix with non-zero entries only along the main diagonal). Implement code to compute the covariance matrix and verify this property. One way to do this is to compute the covariance matrix, and visualise it using the SciPy function `misc.imsave()`. The image should show a white diagonal line against a dark background. For this dataset, because of the range of the diagonal entries, the diagonal line may not be apparent, but this trick of visualizing using `imsave()` will come in handy later in this exercise.

**Step 2: Find number of components to retain**: Next, choose $k$, the number of principal components to retain. Pick $k$ to be as small as possible, but so that at least 99% of the variance is retained. In the step after this, you will discard all but the top $k$ principal components, reducing the dimension of the original data to $k$. Write down the value of $k$ in your report.

**Step 3: PCA with dimensionality reduction**: Now that you have found $k$, compute $\tilde{x}$, the reduced-dimension representation of the data. This gives you a representation of each image patch as a $k$ dimensional vector instead of a 144 dimensional vector. If you are training a sparse autoencoder or other algorithm on this reduced-dimensional data, it will run faster than if you were training on the original 144 dimensional data.

To see the effect of dimensionality reduction, go back from $\tilde{x}$ to produce the matrix $\hat{x}$, the dimension-reduced data but expressed in the original 144 dimensional space of image patches. Visualise $\hat{x}$ and compare it to the raw data, $x$, as shown in Figure 4. You will observe that there is little loss due to throwing away the principal components that correspond to dimensions with low variation. For comparison, you may also wish to generate and visualise $\hat{x}$ for when only 90% of the variance is retained.

![Figure 4: (a) Raw patches; (b) PCA projected images, 99% variance.](image-url)
Step 4a: Implement PCA with whitening and regularization: Now implement PCA with whitening and regularization to produce the matrix $x_{\text{PCAWhite}}$. Use $\epsilon = 0.1$.

Step 4b: Check covariance: Similar to using PCA alone, PCA with whitening also results in processed data that has a diagonal covariance matrix. However, unlike PCA alone, whitening additionally ensures that the diagonal entries are equal to 1, i.e. that the covariance matrix is the identity matrix.

That would be the case if you were doing whitening alone with no regularization. However, in this case you are whitening with regularization, to avoid numerical problems associated with small eigenvalues. As a result of this, some of the diagonal entries of the covariance of your $x_{\text{PCAwhite}}$ will be smaller than 1. To verify that your implementation of PCA whitening with and without regularization is correct, you can check these properties. Implement code to compute the covariance matrix and verify this property. As earlier, you can visualise the covariance matrix with the SciPy function `misc.imsave()`.

Step 5: ZCA whitening: Now implement ZCA whitening to produce the matrix $x_{\text{ZCAWhite}}$. Visualize $x_{\text{ZCAWhite}}$ and compare it to the raw data, $x$, as shown in Figure 5. You should observe that whitening results in, among other things, enhanced edges. Try repeating this with $\epsilon$ set to 1, 0.1, and 0.01, and see what you obtain. The example in Figure 5 was obtained with $\epsilon = 0.1$.

![Figure 5: (a) Raw patches; (b) ZCA whitened images.](image)

3 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw03.zip file that contains the hw03 folder in which you change code only in the 2 required files.

On a Linux system, creating the archive can be done using the command:

```
$ zip -r hw03.zip hw03
```
Please observe the following when handing in homework:

1. Structure, indent, and format your code well.

2. Use adequate comments, both block and in-line to document your code.

3. On the theory assignment, clear and complete explanations and proofs of your results are as important as getting the right answer.

4. Make sure your code runs correctly when used in the directory structure shown above.