COMPSCI 527 — Homework 3

Due on September 29, 2016

Please refer to homework 1 and the class mechanics web page for homework policies and formatting/submission instructions. The submission checklist for this assignment is as follows:

`hw3.pdf`, `imagePCA.m`, `encode.m`, `decode.m`

Keep in mind that your code must also show up in `hw3.pdf` .

Image Gradients

Suppose that you are given a column vector g with $k > 1$ samples of a 1-dimensional Gaussian function, properly normalized, and a column vector d with k samples of the derivative of a 1-dimensional Gaussian function, properly normalized. For instance,

$$g = \begin{bmatrix} 0.0003 \\ 0.1065 \\ 0.7866 \\ 0.1065 \\ 0.0003 \end{bmatrix} \quad \text{and} \quad d = \begin{bmatrix} 0.0025 \\ 0.4951 \\ 0.0000 \\ -0.4951 \\ -0.0025 \end{bmatrix} .$$

Then, the derivative I_h of an image I in the horizontal direction can be computed as follows:

$$I_h = I * g * d^T \tag{1}$$

where the asterisk denotes convolution. Assume that I is much larger than the convolution kernels, and ignore image boundary effects in your answers (so it does not matter if your convolution is `'full'`, `'same'`, or `'valid'`).

1. Why are parentheses not needed in the expression for I_h above?

2. Which of the following two mathematically equivalent computations is computationally more efficient, and why?

$$\begin{aligned} I_h &= I * (g * d^T) \\ I_h &= (I * g) * d^T \end{aligned}$$

In your answer, assume that the image I is $n \times n$ with n much greater than k, and give the approximate cost in arithmetic operations per pixel of I for the two versions.

3. The *Laplacian* of an image $I(x, y)$ is defined as the image

$$L = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} .$$

Write an equation in the same style as equation (1) that use convolutions with vectors g and d and their transposes g^T and d^T to compute the Laplacian L of the image I.

4. Add parentheses to your expression to reflect an efficient ordering of the computation.

5. Let I be a three-dimensional, $n \times n \times n$ array of pixels, perhaps obtained by stacking one after another the n consecutive frames of a gray-level video sequence of frames that are sized n by n pixels each. With three or more dimensions, the notation with transposes used earlier is no longer sufficient, since there is no notation that transposes a vector "into the third dimension.". Instead, given vectors g and d as defined above, let g_1 be a $k \times 1 \times 1$ version of g, that is, let the k samples of g be arranged into a three-dimensional array of dimensions k, 1, and 1, in this order. Similarly, let g_2 be a $1 \times k \times 1$ version of g, and let g_3 be a $1 \times 1 \times k$ version of g. Define d_1, d_2, and d_3 similarly.

Write equations that use convolutions with vectors g_i and d_i (for $i = 1, 2, 3$) to compute the magnitude $\|\nabla I\|$ of the gradient of I **using the smallest possible number of arithmetic operations**, at the cost of extra variables. Write several simple equations rather than one complicated one. You will need a last equation without convolutions that computes the magnitude. You need not parenthesize your expressions.

SVD

It was stated in class that the unit vectors $\mathbf{v}_i$ that map into the axes of the ellipsoid $\mathbf{b} = A\mathbf{x}$ with $\|\mathbf{x}\| = 1$ are orthogonal to each other. The general proof of existence of the SVD given in an Appendix in the notes implies this result. In the problem below, you are asked to prove the same fact more directly for small matrices.

6. Prove by direct calculus that the columns $\mathbf{v}_1$ and $\mathbf{v}_2$ of the matrix V in the SVD $A = U\Sigma V^T$ of any 3×2 matrix A are orthogonal. **Hints:** Let $\mathbf{x}(\theta) = [\cos\theta, \sin\theta]^T$. Where does the function $f(\theta) = \|A\mathbf{x}(\theta)\|$ achieve its maximum and minimum values? It is OK to look up trigonometric identities as you study this function. Here are a few that may or may not be useful, depending on how you approach the problem:

$$\begin{aligned} \cos 2\alpha &= \cos^2\alpha - \sin^2\alpha \\ \sin 2\alpha &= 2\sin\alpha\cos\alpha \\ u\cos\alpha + v\sin\alpha &= \sqrt{u^2+v^2}\,\cos(\alpha - \phi) \end{aligned}$$

In the last expression,

$$\phi = \text{arctan}_2(v, u) = \begin{cases} \arctan(\frac{v}{u}) & \text{if } u > 0 \\ \pi + \arctan(\frac{v}{u}) & \text{if } u < 0 \\ \frac{\pi}{2} & \text{if } u = 0 \text{ and } v > 0 \\ -\frac{\pi}{2} & \text{if } u = 0 \text{ and } v < 0 \\ 0 & \text{if } u = 0 \text{ and } v = 0 \end{cases}$$

denotes the two-argument arc-tangent.

PCA

The MNIST handwritten digit database was developed by NYU's Yann LeCun in the late Nineties to provide a benchmark for software that recognizes isolated handwritten digits. The site `http://yann.lecun.com/exdb/mnist/` describes the database, which is contained in the four files in the `data` directory in the homework zip file.

If you make the `code` directory in the zip file your MATLAB working directory and type

```
data = readMNISTDatabase('../data');
```

then the resulting structure `data` will contain a set of 60,000 digit images, each annotated with a *label* that specifies what digit that image depicts. These images are said to be *manually annotated* with these labels. More specifically, the `data` structure has a field `data.image` with 60,000 gray-level images of size 28×28 pixels of type `uint8` arranged in a single array of size $28 \times 28 \times 60{,}000$. Each image shows a single handwritten digit between 0 and 9, properly sized and centered. There is also a field `data.label` with the corresponding 60,000 labels (each label is a `uint8` number between 0 and 9).

Your code may assume that all images are gray-level.

7. Write a MATLAB function with header

```
function [code, sigma2] = imagePCA(images, d, sampling)
```

that takes an $r \times c \times n$ array `images` of `uint8` images like `data.image`, an integer dimension `d` that is no greater than $m = rc$, and an optional, positive, integer sampling factor `sampling`. The function returns a data structure `code` with the result of performing the Principal Component Analysis (PCA) on the array `images`. The output vector `sigma2` collects the squares of the m (not `d`) singular values of the PCA.

More specifically, if you reshape all the images into an $m \times n$ array A of type `double`, the structure `code` has a field `code.centroid` with the centroid of all the images, and a field `code.decoder` with the $m \times d$ PCA matrix for the given images. The centroid is an m-dimensional column vector equal to the average of all the columns of A. The PCA matrix is the appropriate submatrix of the matrix U of left singular vectors of A. Processing all the images in `images` may be too expensive, so the argument `sampling` allows working with just `images(:, :, 1:sampling:end)` instead. For instance, if `sampling` is 60 (the default value) and an array of 60,000 images is provided, only 1,000 images are used to compute the PCA.

Each column of `code.decoder` has as many entries as each input image has pixels, and can therefore be reshaped into an image (with signed values). It is instructive to look at some of these images.

Hand in your code (both in the PDF file and as a separate MATLAB file) for `imagePCA`, as well as a single figure that shows all the 32 singular-vector images corresponding to the columns of `code.decoder` obtained by running the function on `data.image` with `d` = 32 and with the default value of `sampling`. Use the MATLAB `imagesc` and `axis` commands to display the images with the proper aspect ratio. Arrange the 32 images into a 6×6 array (with the last four entries empty) on the page, and label each with a well-visible integer between 1 and 32, so you know which image is which.

[Hint: You may either write the PCA code from scratch or use the MATLAB function `pca`. The former solution is easier, since `pca` uses shape conventions and terminology that are different from what we used in class. If you use `pca`, it is your responsibility to understand its input and output arguments precisely.]

8. Once the PCA `code` data structure for a data set is known, one can encode further images from the data set with the same procedure: Turn the new image into a vector `x`, subtract `code.centroid` from it, and compute the vector `y` of the coefficients that express `x` in the orthonormal basis given by the columns of `code.decoder`.

Write a MATLAB function with header

```
function y = encode(img, code)
```

that takes an image `img` and a `code` as produced by `imagePCA` and returns the `d`×`1` PCA vector `y` for `img`. Sizes of `img` and the fields of `code` must be compatible, but your code need not check that this is the case. [Hint: remember to cast `img` to `double`.]

Hand in your code (both in the PDF file and as a separate MATLAB file). No figures yet.

9. A vector `y` produced by `encode` is a compact, approximate representation of the original image `img`. The image approximation can be recreated from `y` by linearly combining the columns of `code.decoder` with the coefficients in `y`, adding `code.centroid` to the result, and reshaping the result into an image of the appropriate size.

Write a MATLAB function with header

```
function img = decode(y, code, sz)
```

that takes a PCA vector `y` as produced by `code`, the structure `code` itself, and the size `sz` of the original image. The function returns the corresponding `uint8` image `img`. Same *caveat* for sizes as for `encode`.

Hand in your code (both in the PDF file and as a separate MATLAB file). No figures yet.

10. Use the instruction

```
[~, which] = unique(data.label);
```

to obtain a vector `which` with one image index for each digit from 0 to 9. These images are used here as samples to show the effects of PCA.

For $d = 2, 4, 8, 16, 32$, compute the reconstruction of the ten sample images from their PCA with d components. [Hint: You only need to `encode` once. Just pass parts of the resulting code to `decode`. Remember to cast `img` to `uint8` before returning it.]

Hand in a single figure with the ten original sample images, and then five more figures, each with the resulting reconstruction for a different value of d. Arrange the 10 images in each figure into a 3×4 array (with the last two entries empty), and label each with a well-visible integer between 0 and 9. Show clearly which figure is for which value of d. Also give a table that for each value of d shows the fraction (a number between 0 and 1) of total variance captured by the PCA. State how you computed the entries in the table, but do not show your code.