Review for Exam 2

Erik G. Learned-Miller
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

December 6, 2012

Abstract

This document reviews material you will need for Exam 2. In addition, you should also cover all of the material for Exam 1 from the previous review sheet.

1 Basic notation

In this document, I will use square bracket notation for representing functions that are defined over the integers. For example,

$$f[x]$$

is a function that can take values for $x = \{..., -2, -1, 0, 1, 2, ...\}$. So $f[3]$, $f[0]$, and $f[-1000]$ are meaningful, but $f[0.5]$ is not. For functions taking values over the real numbers, I will use the standard parentheses notation. That is, $f(x)$ is defined for any value of x that is a real number.

2 A bit of vector math

You will need to understand certain basic results about vectors. For the purposes of this class, you can think of a vector as just a collection of numbers, like in Matlab. For example, a vector of length 3, also known as a vector $\mathbf{v}$ in three dimensions, or a 3-dimensional vector, might be something like

$$\mathbf{v} = \begin{bmatrix} 12 \\ -3 \\ 4 \end{bmatrix}.$$

2.1 Vector magnitudes

The magnitude of a vector is a simple concept. You can think about it as the length of the vector, or equivalently, as the distance from the point specified by the vector to the origin of the coordinate system, that is, the distance between the vector and the zero vector. Using the example of $\mathbf{v}$ defined above, the magnitude of $\mathbf{v}$ is written

$$||\mathbf{v}|| = \sqrt{12^2 + (-3)^2 + 4^2} = 13.$$

2.2 Unit vectors

A unit vector is just a vector whose length is 1, like

$$\mathbf{w} = \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix}.$$

To convert a vector whose length is *not* 1 into a unit vector that is pointing in the same direction (whose components have the same *relative magnitude* as the original vector), just divide each original component by the magnitude of the vector. For our example $\mathbf{v}$ from above:

$$\mathbf{u} = \mathbf{v}/13 = \begin{bmatrix} \frac{12}{13} \\ \frac{-3}{13} \\ \frac{4}{13} \end{bmatrix}.$$

You can verify for yourself that $\mathbf{u}$ has a magnitude of 1.

2.3 Dot products of vectors

The dot product of two vectors is just the sum of the product of the corresponding components. Let

$$\mathbf{t} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.$$

Then the dot product of $\mathbf{t}$ and $\mathbf{v}$ is

$$\mathbf{t} \cdot \mathbf{v} = (2)(12) + (1)(-3) + (3)(4) = 33.$$

If the components of $\mathbf{v}$ are $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$, and the components of $\mathbf{t}$ are $\mathbf{t}_1$, $\mathbf{t}_2$, and $\mathbf{t}_3$, then the dot product can be written

$$\mathbf{t} \cdot \mathbf{v} = \sum_{i=1}^{3} (\mathbf{t}_i)(\mathbf{v}_i).$$

2.4 Angle between unit vectors

When the bases of two vectors are put together (for example, if they are both at the origin), then an angle is formed between them. Let's call it θ. **For unit vectors**, the following formula gives the relationship between the angle θ and the two vectors $\mathbf{u}$ and $\mathbf{v}$:

$$\cos(\theta) = \mathbf{u} \cdot \mathbf{v}. \tag{1}$$

If the two vectors being compared **are not unit vectors**, then they have to be converted to unit vectors before the formula works. Consider two vectors $\mathbf{y}$ and $\mathbf{z}$ that are not unit vectors. To find the angle between them, normalize them first:

$$\cos(\theta) = \frac{\mathbf{y}}{||\mathbf{y}||} \cdot \frac{\mathbf{z}}{||\mathbf{z}||}. \tag{2}$$

Note: these formulas work for vectors with any number of components, whether it is 2, 3, 5, or a hundred thousand components!

2.5 Vector that maximizes the dot product

Given a unit vector $\mathbf{v}$, which other unit vector maximizes the dot product with $\mathbf{v}$? To answer this question very simply, first ask, "What angle θ maximizes the cosine function (from equation 1)?" The answer is that the cosine function is maximized when $\theta = 0$ and $\cos(0) = 1$. Thus, to maximize the dot product between two unit vectors, the angle between them should be 0! This implies that the unit vector which maximizes the dot product with a unit vector $\mathbf{v}$ is the vector $\mathbf{v}$ itself! In other words:

$$\mathbf{v} \cdot \mathbf{v} \geq \mathbf{v} \cdot \mathbf{w},$$

for any unit vector $\mathbf{w}$.

2.6 Exercises

Here are simple questions to test your knowledge:

1. What's the magnitude of a unit vector?
2. What's the maximum possible value of the dot product between two unit vectors? (Answer: it's the same as the maximum value of the cosine function.)
3. What's the maximum possible value of the dot product between two vectors if their magnitudes are 2 and 5 respectively. (Answer: multiply both sides of equation 2 by the magnitudes of $\mathbf{y}$ and $\mathbf{z}$ and reason from there.)

2.7 Images as vectors

Consider a grayscale image that has 100 rows and 100 columns. There are 10,000 pixels in the image. You can think of this as a 10,000 component vector. Now consider two images I and J. If you think of them as two 10,000 component vectors, you can compute the dot product between them. If we normalize the two vectors, and take their dot product, what is the maximum possible value? (Answer: 1.0, the maximum of the cosine function.)

Suppose we have taken an image I, turned it into a vector, and normalized it, but dividing each element of the vector the magnitude of I. What other normalized image vector would maximize the dot product with the normalized I? The answer, drawing from section 2.5, is *the same image*. This the mathematical basis for the tracker that we built in the correlation based tracking assignment.

3 Filtering and Convolution

Consider a function $f[x]$ defined over the integers, where

- f[0]=1,
- f[1]=2,
- f[2]=1,

and $f[i] = 0$ for all other values of i, both positive and negative.

- Be able to plot $f[i]$, with appropriate labels on the axes.
- Be able to plot $f[i] + 1$.
- Be able to plot $f[i + 1]$.
- Be able to plot $f[i + 1] + 1$.
- Be able to plot $f[i - 1]$.

- Be able to plot $2f[i]$, $2f[i+1]$, $2f[i]+1$.
- Be able to plot $f[i]+f[i-5]$.
- Be able to plot $f[i]+f[i-1]$.
- Be able to plot $2f[i]+3f[i-1]+f[i-3]+2f[i-4]$.
- Finally, be able to plot

$$\sum_{t=0}^{4} g[t]f[i-t], \tag{3}$$

  where $g[t]$ is defined by

  - $g[0]=1$,
  - $g[1]=4$,
  - $g[2]=2$,
  - $g[3]=0$,
  - $g[4]=3$.

The expression in Equation 3 is called the *convolution* of the discrete function f with the function g, and can be written as

$$f[i] \otimes g[i].$$

3.1 Properties of convolution

1. Commutativity: $f[i] \otimes g[i] = g[i] \otimes f[i]$.
2. Associativity: $f[i] \otimes (g[i] \otimes h[i]) = (f[i] \otimes g[i]) \otimes h[i]$.
3. Linearity: $f[i] \otimes (g[i] + h[i]) = f[i] \otimes g[i] + f[i] \otimes [h[i]$.

3.2 Two-dimensional convolution

If I defined a two-dimensional function $f[i,j]$ over the discrete grid, you should be able to understand the meaning of functions like $f[i-1,j-3]$ just as in the one-dimensional case above. (The offsets -1 and -3 shift the function one unit to the right and three units up.) Putting these together, you should be able to understand expressions such as

- $3f[i-1,j-3]-5f[i-2,j-2]$, and
- $$\sum_{s=0}^{5}\sum_{t=0}^{5} g[s,t]f[i-s,j-t], \tag{4}$$

  which is the two-dimensional convolution of f with some other function g.

3.3 The relationship of convolution to vision

Convolution is often used in computer vision to model certain *processes* which occur in image formation. Two of the most common processes are:

- blurring due to lack of focus, or blurring due to atmospheric effects (as in astronomy,
- blurring due to motion of the camera

In either of these cases, the easiest way to start thinking about convolution is to think about what would happen to the photograph of a single point of light when the entire rest of the image was black. For example, think of the photo, on a very dark night of a star which is visible between some clouds, such that there is really only one point of light in the whole photograph. If the camera is out of focus, then this single point of light will appear as a smudge, known as the *point spread function*. If there happen to be two stars visible that are far enough apart, then there will be two copies of the point spread function, one at the location of each of the stars. This is how to think about convolution. The imaging process effectively "drops" a copy of the point spread function (which is the same thing as the convolution kernel) at each point of light in the image. The *brightness* of the smudge is exactly proportional to the brightness of the original star.

Of course, if the stars are close together, their smudges (or point spread functions will overlap), and we may not be able to discern in the blurred image that there were two stars there to begin with. Hence, the convolution of a complicated image will end up looking more just like a blur than a bunch of copies of a point spread function.

4 Filtering

Filtering or correlating the image with a kernel is very closely related to convolution, but conceptually, it is used in a very different way. Filtering is generally used to *analyze what is in the image* rather than *to model the process of image formation*, which is what convolution is frequently used for.

For example, filtering is often used for detection of an object that one has stored in memory. By filtering an image with an image of an object, we hope to find the patch of the image most similar to that patch (in some sense). Closely related to this is the idea of finding edges in images. If we want to find the places in the image where there are vertical edges, we filter the image with a convolution kernel that looks like a veritcal edge. Any region which responds strongly must have a patch that looks something like a vertical edge.

4.1 How to filter an image

Given an image I with N pixels and another image J of *exactly the same size*, then the result k of filtering I with J is simply the dot product of the two

images:

$$k = \sum_{i=1}^{N} I_i \times J_i,$$

where I_i and J_i are the pixel values in each image.

To filter an image I with a smaller image J, repeat the process described above for each patch of I that is the same size as image J by "sliding" the filter window across each possible position in I.

When you filter a larger image I with a smaller patch J, you will generally end up with a smaller image as a result. For example if I is 7x7 and J is 3x3, then the final image will be 5x5, because there are exactly 25 locations where the entire filter kernel J will fit within the image I. If it is important to make the final image the same size as the original image, one can allow the filter to extend outside the image I. To do this, one has to invent values for the missing pixels outside the image I. Usually, a value of 0 is used.

4.2 Some very common filter kernels

We have mentioned a variety of common filter kernels in class. Here is a review.

4.2.1 Averaging filter

The following filters compute the average brightness value of the patch that they are filtered with:

$$f_{3x3} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$f_{5x5} = \begin{bmatrix} \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \end{bmatrix}$$

The first one averages over a 3x3 neighborhood. The second over a 5x5 neighborhood.

4.2.2 Weighted average filter

Instead of each pixel contributing equally to the average, we can have pixels near the middle of the filter contribute more and pixels near the edge of the filter contribute less, as in

$$f_{Gauss} = \begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix}$$

While you don't need to memorize the specific values of such a filter, things you should notice include

- The values are all positive,
- The values in the middle are largest, decreasing away from the middle,
- The values sum to 1.0,
- They are symmetric about the center.

If you plotted the filter as a surface it would look approximately like a two-dimensional Gaussian (or normal) distribution. This is the kind of filter that one uses to produce the smoothed (or blurred) images used in a Gaussian pyramid (such as the one used in the SIFT descriptor).

4.2.3 Edge filters

Some filters, which look like little "mini-edges" if you show them as an image, are good for detecting edges. The following filters can be used to find horizontal edges in images, vertical edges, and diagonal edges, respectively:

$$f_{vert} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

$$f_{hor} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

$$f_{diag} = \begin{bmatrix} 0 & 0.5 & 1 \\ -0.5 & 0 & 0.5 \\ -1 & -0.5 & 0 \end{bmatrix}.$$

4.2.4 Partial derivative filters

In class, we discussed ways of estimating the gradient of an image at each point. This is the direction in which an image is changing the brightness the fastest, and can be written as a vector of the partial derivatives:

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \\ \frac{\partial I}{\partial y} \end{bmatrix}.$$

To compute a "discrete" approximation to the partial derivatives you can use the following filters:

$$f_{\partial x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix},$$

and

$$f_{\partial y} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}.$$

Alternatively, you could use the vertical edge filter (x-derivative) and horizontal edge filter (y-derivative) defined above to compute approximations to the partial derivatives.

4.2.5 The trivial "identity" filter

You should be able to figure out what the following filter does. If not, you don't understand filters yet:

$$f_{ident} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

5 SIFT features

Review the slides on SIFT features using the link on the course web site (Nov. 19 lecture).

The key high-level ideas of the SIFT descriptor are

- First, we find points of interest, known as "keypoints" in the image. These will be *local extrema* (minima or maxima) of the Difference-of-Gaussian image pyramid (discussed below).
- Some of these local extrema points are thrown out because they are unstable. This means that a very small change to the image (changing one pixel by one brightness value, for example), may change whether the point is a local extremum. Points in smooth regions of the image and along "ridges" (like images of folding curtains) are typically the types of points that are thrown out.
- After a keypoint is found (it is at a local extremum and it is not unstable), its "scale" is defined to be the level of the image pyramid in which it was found. If it was found in the blurriest level of the image pyramid, it will have a large scale. If it was found in the sharpest level of the image pyramid, it will have a small scale.
- Then a keypoint *orientation* is assigned. This is the, or one of the, dominant orientations in a patch around the keypoint. The dominant orientation is found by looking at a histogram of the gradient orientations in the patch, and picking the orientations with the most values in the histogram.
- Finally, with keypoints that have scales and orientations, we put a set of 4x4 bins down at the given orientation and scale, and build sixteen

histograms of local gradient magnitudes. Since each histogram has 8 bins, this will give us a total of 4x4x8 values for on SIFT *descriptor*.

You should understand all of the above points about SIFT keypoints and descriptors. Now, here are some additional details you need to understand to apply SIFT features.

5.0.6 Scale space and difference-of-Gaussian scale space

The scale space of an image is a sequence of successively more blurred copies of an image. By starting with an image I and letting this be layer 1, which we can call I_1 of the scale space, we can form layer 2 of scale space by filtering the image with a Gaussian kernel (see weighted average filter in filtering section). Since a Gaussian filter is symmetric, the resulting of filtering with the kernel and convolving with it (by flipping the kernel vertically and horizontall) is equivalent. Thus we can think about this as Gaussian convolution rather than Gaussian filtering, which makes it easier to write:

$$I_2 = I_1 \otimes f_{Gauss}.$$

You can blur more or less depending upon the spread of your Gaussian kernel, but you can just think of using the kernel given above. In general,

$$I_{k+1} = I_k \otimes f_{Gauss}.$$

The set of images $I_1, I_2, ..., I_K$ is called a Gaussian scale space of K levels. The difference-of-Gaussian scale space is formed by taking differences of successive images in the Gaussian scale space:

$$D_1 = I_2 - I_1,$$

and more generally,

$$D_k = I_{k+1} - I_k.$$

The set of images $D_1, D_2, ..., D_{K-1}$ is called a difference-of-Gaussian scale space of $K - 1$ levels.

5.0.7 Local extrema

A pixel in the stack of images $D_1, D_2, ..., D_{K-1}$ is a *local extremum* if it is larger thanits 26 neighbors or smaller than its 26 neighbors.

5.0.8 Magnitude and Orientation of a gradient

Using the partial derivative filters described in the filtering section above, you can compute the partial derivatives of an image at each point, and hence form the gradient of the image at each point. You can compute the magnitude of the gradient simply by using the vector magnitude formula from section 2.1.

Assuming the magntiude is not zero, you can compute the gradient orientation as

$$\theta = \arctan(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x}).$$

That is, θ is the angle whose tangent is the y-partial over the x-partial. If you don't understand the arctangent function, you need to look it up on your own.

5.0.9 SIFT uses

SIFT was specifically defined to create descriptors for image points such that if the same object were seen again, the descriptor for points on that object would be highly similar to the previous view of that object.

6 RANSAC

Given a bunch of SIFT keypoints and descriptors in one image, and a bunch of SIFT keypoints in another image, we can draw lines between descriptors that are very similar to each other (Euclidean distance betwen two SIFT vectors is less than some ϵ). However, some of these matches will be incorrect. THe purpose of RANSAC is to find a transformation that puts as many points from one image into correpondence with the other image as possible. I ask you to refer to the slides on the course web site to review RANSAC (slides from Nov. 21).

You should be generally familiar with how to use RANSAC with SIFT features to build an image panorama, as discussed in class.

7 Face Detection and Face Recognition

Suppose you read an article with a quotation from the CEO of a new face recognition company that says, "Our algorithm achieves 99.9% in face recognition." Why is this a meaningless statement? Consider the following factors:

- What is the number of people who are considered as potential identities of a face?
- Is the face under even lighting, or arbitrary lighting?
- Is the subject asked to pose for the picture, or are the pictures candid?
- How many training examples are given for each face?

These are just some of the reasons a single number is meaningless. Be prepared to discuss other reasons.

Discuss why face recognition might not yet be practically applied when looking for terrorists in an airport. How is the deployment of face recognition technology dependent upon the costs of the errors that might be made? I discussed these issues in class. You can review them in the "Intro to face slides" (Nov. 26 lecture).

7.1 Face Detection

1. Be familiar with the "Face in the Beans" slide (from "Intro to face slides" on course web page.) What does it illustrate about face detection in humans?

2. Discuss some of the subtleties in defining face detection. What is a true positive? A true negative? A false positive? A false negative? Why do we need a very, very, very low rate of false positives?

3. How is face detection posed as a classification problem? What are the classes?

4. Give a plausible number of regions that must be evaluated for a face detector and a typical image size.

7.1.1 Boosting

Understand the following elements of the "Boosting" algorithm as presented in the slides from Dec. 3.

- Boosting is a discriminative classification method which means that it uses training data to try to form regions of feature space that belong to one class or another class. THe example in the lecture slides is a two-dimensional feature space, meaning that it tries to break up the plane (a two-dimensional region) into areas that belong to the class "red" or "blue". In face detection, we may use a much larger number of features, but the same principles apply. A good exercise is to make sure you understand how Boosting would work with 3 features instead of 2 features.

- Boosting uses "weak learners", which are often based on computing a single feature. For example, you could build a gender classifier by asking whether someone's hair was longer than 6 inches. It wouldn't be a very good classifier, but as long as it is right more than 50% of the time, it is called a weak learner or a weak classifier. Boosting combines weak learners into a single better classifier.

- In each stage of Boosting, the examples that the previous learner got wrong are "re-weighted" and a new weak learner is found that tries to do better on the reweighted examples. You can think of reweighting as adding extra examples of the data points that were classfied incorrectly.

- After some number of weak learners are built, Boosting makes a final decision by adding together the weak learners in a voting scheme. Whichever class gets the most votes from all of the weak learners is the final decision.

7.1.2 Haar features and integral images

The rectangular features used in the Viola-Jones face detector are called Haar features. Many Haar features can be computed efficiently in a single image by first computing the *integral image*. You should understand the principles of integral images. If you can't remember how they work, refer to your notes and the lecture slides (from Dec. 3).

7.1.3 The cascade of classifiers

The Viola-Jones face detector is very fast because it uses a "cascade" of classifiers. An image region is only detected as a face if it makes through *all of the classifiers in the cascade.* The way it is designed, the Viola-Jones detector can reject most patches in an image as being highly unlikely to be faces by putting them through a very very simple classifier that comes at the beginning of the cascade. It can then spend more effort on the small number of images that come later in the cascade by applying more complicated classifiers to those patches.

8 Acromegaly and morphable models

Be able to answer these questions. You can refer to the slides for Dec. 5.

- What is acromegaly and what does it have to do with face recognition?
- What does it mean to screen for a medical condition?
- Why did I decide to use a morphable model to help screen for acromegaly instead of using a procedure which analyzed the location of various features within the image?

You do not need to understand principal components analysis or the details of the morphable model for the final.