1. Some simple linear algebra

a. Show that $vec(a \cdot b^T) = b \otimes a$ where a and b are both vectors

First let's define a and b as column vectors with m entries:

$$a \in \mathbb{R}^{mx1}$$

$$b \in \mathbb{R}^{mx1}$$

More concretely:

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Then:

$$a \cdot b^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 & \dots & b_m \end{bmatrix} = \begin{bmatrix} a_1b_1 & a_1b_2 & & a_1b_m \\ a_2b_1 & a_2b_2 & & a_2b_m \\ a_3b_1 & a_3b_2 & \dots & a_3b_m \\ \vdots & \vdots & & \vdots \\ a_mb_1 & a_mb_2 & & a_mb_m \end{bmatrix}$$

Another way of writing this is:

$$a \cdot b^T = \begin{bmatrix} | & | & & | \\ a \cdot b_1 & a \cdot b_2 & \dots & a \cdot b_m \\ | & | & & | \end{bmatrix} \textit{where each column j is simply the vector a scaled by } b_j$$

In this form it is easy to see that:

$$vec(a \cdot b^T) = \begin{bmatrix} a \cdot b_1 \\ a \cdot b_2 \\ \vdots \\ a \cdot b_m \end{bmatrix}$$

We know that

$$b \otimes a = \begin{bmatrix} a \cdot b_1 \\ a \cdot b_2 \\ \vdots \\ a \cdot b_m \end{bmatrix} = vec(a \cdot b^T)$$

b. Show that $vec(A)^T \cdot vec(B) = tr(A^T \cdot B)$ for two matrices of the same order

First let's define a and b as matrices of the same order:

$A \in \mathbb{R}^{mxn}$

$B \in \mathbb{R}^{mxn}$

For me, it made the notation much simpler if I thought of a and b as a collection of column vectors. That means for both a and b, I have n column vectors, each of which has m entries. I also define A^T similarly:

$$A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix} \qquad B = \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_n \\ | & | & & | \end{bmatrix} \qquad A^T = \begin{bmatrix} - & {a_1}^T & - \\ - & {a_2}^T & - \\ - & \vdots & - \\ - & {a_n}^T & - \end{bmatrix}$$

Just to be concrete:

$a_i \in \mathbb{R}^{mx1} \, for \, i \in \{1, 2, \dots, n\}$

$b_i \in \mathbb{R}^{mx1} \, for \, i \in \{1, 2, \dots, n\}$

Now it is easy to see that the left-hand side breaks down as follow:

$$vec(A) = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \qquad vec(B) = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$vec(A)^T = \begin{bmatrix} {a_1}^T & {a_2}^T & \cdots & {a_n}^T \end{bmatrix}$$

$$vec(A)^T \cdot vec(B) = \begin{bmatrix} {a_1}^T & {a_2}^T & \cdots & {a_n}^T \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = {a_1}^T \cdot b_1 + {a_2}^T \cdot b_2 + \cdots + {a_n}^T \cdot b_n$$

We can look at the right-hand side now:

$$A^T \cdot B = \begin{bmatrix} - & {a_1}^T & - \\ - & {a_2}^T & - \\ - & \vdots & - \\ - & {a_n}^T & - \end{bmatrix} \cdot \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} {a_1}^T \cdot b_1 & {a_1}^T \cdot b_2 & \cdots & {a_1}^T \cdot b_n \\ {a_2}^T \cdot b_1 & {a_2}^T \cdot b_2 & \cdots & {a_2}^T \cdot b_n \\ \vdots & \vdots & \vdots & \vdots \\ {a_n}^T \cdot b_1 & {a_n}^T \cdot b_2 & \cdots & {a_n}^T \cdot b_n \end{bmatrix}$$

$$\boldsymbol{tr}(A^T \cdot B) = {a_1}^T \cdot b_1 + {a_2}^T \cdot b_2 + \cdots + {a_n}^T \cdot b_n = vec(A)^T \cdot vec(B)$$

If X is a real matrix, then the matrix $X \cdot X^T$ is positive semidefinite.

First, let's define the matrix X to be $X \in \mathbb{R}^{m \times n}$

To test if a matrix H ($\mathbb{R}^{m \times n}$) is positive semidefinite, we need to prove that for any vector $z \in \mathbb{R}^{n \times 1}$,

$z^T (H) z \geq 0$

Okay, in this case $H = X \cdot X^T$, so we need to prove the following:

$z^T (X \cdot X^T) z \geq 0$ (1)

First, let's define

$y = X^T \cdot z$

Then

$y^T = z^T \cdot X$

Okay, now let's substitute y into (1):

$y^T y = \|y\|_2^2 = z^T (X \cdot X^T) z \geq 0$

A probability problem

What we want to find:

$P(+\,Cancer \mid +\,Test)$

What we know:

$P(+\,Cancer) = 0.007$

$P(-\,Cancer) = 0.993$

$P(+\,Test \mid +Cancer) = 0.9$

$P(+Test \mid -Cancer) = 0.08$

Well, looks like we need to use Bayes Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In our case, this becomes:

$$P(+\,Cancer \mid +\,Test) = \frac{P(+\,Test \mid +Cancer)P(+\,Cancer)}{P(+Test)}$$

We have pretty much everything we need, EXCEPT $P(+Test)$. I found this by filling out the matrix:

$$\begin{bmatrix} & +Cancer & -Cancer & \\ +Test & ? & ? & ? \\ -Test & ? & ? & ? \\ & 0.007 & 0.993 & \end{bmatrix}$$

Let's solve the joint probabilities first:

$P(+Cancer, +Test) = P(+Test|+Cancer)P(+Cancer) = \ 0.9 * 0.007 \ = \ 0.0063$

$P(+Cancer, -Test) = 0.007 - 0.0063 = 0.0007$

$P(-Cancer, +Test) = P(+Test|-Cancer)P(-Cancer) = \ 0.08 * 0.993 = 0.07944$

$P(-Cancer, -Test) = 0.993 - 0.07944 = 0.91356$

And then $P(+Test)$ & $P(-Test)$

$P(+Test) = P(+Cancer, +Test) + P(-Cancer, +Test) = \ 0.0063 + 0.07944 \ = \ 0.08574$

$P(-Test) = P(+Cancer, -Test) + P(-Cancer, -Test) = 0.0007 + 0.91356 \ = \ 0.91426$

$$\begin{bmatrix} & +Cancer & -Cancer & \\ +Test & 0.0063 & 0.07944 & 0.08574 \\ -Test & 0.0007 & 0.91356 & 0.91426 \\ & 0.007 & 0.993 & \end{bmatrix}$$

Now let's get back to solving Bayes rule:

$$P(+\,Cancer \mid +Test) = \frac{P(+\,Test \mid +Cancer)P(+\,Cancer)}{P(+Test)} = \frac{0.9 * 0.007}{0.0063} = 0.0735 = 7.35\%$$

Stats operations using linear algebra

You are given K grayscale images of size M by N. Each image is represented as a column vector and they are all presented to you as an (M×N) by K matrix.

1a. Use a linear algebra expression that computes the mean image as an M by N matrix. Don't use summations, don't use for loops. You can freely construct any arbitrary matrix to use in any expression that you need. Hint: vec-transpose

First, let's think about what the data looks like. Let's call it D:

$$D = \begin{bmatrix} | & & | \\ IMG\ 1 & \cdots & IMG\ K \\ | & & | \end{bmatrix}, \qquad D \in \mathbb{R}^{(M*N)\times K}$$

Every column in this data matrix is a vectorized image. Each row represents a pixel value. To find the mean image, what we need to do is average across the rows of this data matrix. To do this, I performed the following linear operation on D:

$$\mu_{vec} = D \cdot \left(\frac{1}{K}\mathbf{1_K}\right), \mathbf{1_K}\ is\ a\ column\ vector\ of\ 1's\ with\ K\ entries$$

We end up with $\mu_{vec} \in \mathbb{R}^{(m*n)\times 1}$. Each entry in this vector is essentially the average pixel for each pixel across all images. To get this into the mean image form, we need to reshape this vector into the actual image shape which we can do with the vec-transpose

$$\mu_{IMG} = [\mu_{vec}]^{(M)}$$

In one line: $\mu_{IMG} = \left[D \cdot \left(\frac{1}{K}\mathbf{1_K}\right)\right]^{(M)}, \mathbf{1_K}\ is\ a\ column\ vector\ of\ 1's\ with\ K\ entries$

1b. Obtain a 2×2 covariance matrix of the average of the top half of all the images (not top half of the vector representation) with the average of the bottom half.

To summarize the steps of this problem:

1. Find the mean value of every image's top half and bottom half. This should result in 2*K entries, 2 per image.
2. Find the mean value of the top and bottom half across all images. This should result in 2 entries.
3. Subtract the mean value of the top across all images (found in step 2) from every mean value of the top half of each image. Do the same thing for the bottom.

We should be able to perform these steps in the following way. First, let's construct a matrix T as follows:

$$T = \frac{2}{(M \times N)}\left(\mathbf{1_N}^{\boldsymbol{T}} \otimes \boldsymbol{I_2} \otimes \mathbf{1}_{\frac{\boldsymbol{M}}{\mathbf{2}}}{}^{\boldsymbol{T}}\right)$$

When I multiply my input matrix D on the left with this transform, I get out another matrix, $\mu_{top/bottom,images}$, with as many columns as images. Each column in this matrix has two rows, the top entry containing the average of the top of that image, and the bottom entry containing the average of the bottom of that image:

$$\mu_{top/bottom,images} = \begin{bmatrix} \mu_{top,image\ 1} & \mu_{top,image\ 2} & \cdots & \mu_{top,image\ K} \\ \mu_{bottom,image\ 1} & \mu_{bottom,image\ 2} & \cdots & \mu_{bottom,image\ K} \end{bmatrix},\ \mu_{top/bottom,images} \in \mathbb{R}^{2\times K}$$

Next, I want to average $\mu_{top/bottom,images}$, across its rows to get a new vector, $\mu_{top/bottom}$, which has exactly 2 entries. The top entry is the average of all the top halves across all images. The bottom entry is the average of all the bottom halves across all images. I do this as following:

$$\mu_{top/bottom} = \mu_{top/bottom,images} \cdot \left(\frac{1}{K}\mathbf{1_K}\right), \mathbf{1_K}\ is\ a\ column\ vector\ of\ 1's\ with\ K\ entries, \mu_{top/bottom,} \in \mathbb{R}^{2\times 1}$$

Now I need to subtract $\mu_{top/bottom}$ from every column in $\mu_{top/bottom,images}$ to get a new matrix μ^o:

$$\mu^o = \mu_{top/bottom,images} - \mathbf{1}_{\boldsymbol{K}}^{\boldsymbol{T}} \otimes \mu_{top/bottom}$$

We are now ready to compute the covariance.

$$\Sigma = \frac{1}{k-1}(\mu^o \mu^{oT})$$

Now your images are colored and packed in a M by N by 3 by K tensor. The dimension of size 3 contains the red, green and blue channel.

2a. Like above, show the expression for the mean image over all channels.

$$\mu_{img} = \left(\left(\frac{1}{K}\,\mathbf{1}_{\boldsymbol{K}}^{\boldsymbol{T}} \otimes \frac{1}{3}\,\mathbf{1}_{\boldsymbol{3}}^{\boldsymbol{T}} \otimes I_N \otimes I_M\right) vec(D)\right)^{(M)}$$

This transform looks very similar to the color mixing transform from the slides, and indeed it is. We can see what each component is doing. First let's look at the Kronecker products. Each of these products allow us to perform operations across the various dimensions of our data. This combination does the following:

$\frac{1}{K}\,\mathbf{1}_{\boldsymbol{K}}^{\boldsymbol{T}}$: *this will average across all images*

$\frac{1}{3}\,\mathbf{1}_{\boldsymbol{3}}^{\boldsymbol{T}}$: *this will average across all channels*

I_N: *this will do nothing along the columns*

I_M: *this will do nothing along the rows*

When we multiply a vectorized D on the left with this matrix, we get a vectorized version of the mean across all image across and across all channels. To get the mean image, we reshape using vec-transpose.

2b. Do that again but compute the mean image of only the red channel.

$$\mu_{red} = \left(\left(\frac{1}{K}\,\mathbf{1}_{\boldsymbol{K}}^{\boldsymbol{T}} \otimes [1\ 0\ 0] \otimes I_N \otimes I_M\right) vec(D)\right)^{(M)}$$

The logic is pretty much the same as above where each component of the Kronecker products performs some operation on each of the dimensions.

$\frac{1}{K}\,\mathbf{1}_{\boldsymbol{K}}^{\boldsymbol{T}}$: *this will average across all images*

$[1\ 0\ 0]$: *this is a selection operator to select the red channel*

I_N: *this will do nothing along the columns*

I_M: *this will do nothing along the rows*

When we multiply a vectorized D on the left with this matrix, we get a vectorized version of the mean across all image across and across only the red channels. To get the mean image, we reshape using vec-transpose.

Problem 4. Signal Processing is Linear Algebra

As promised in class, matrix multiplies can perform all kinds of linear operations. Here you will have to implement a spectrogram using only a matrix multiplication. Your input sound is a column vector x. Describe exactly how you would construct a matrix A, such that the product A·x will produce the vec(·) of the complex spectrogram coefficients. Your transform should have a DFT size of 1024, a hop size of 512, and will use a Hann window. Make a plot of the absolute value of the spectrogram matrix and allow me to marvel at its beauty.

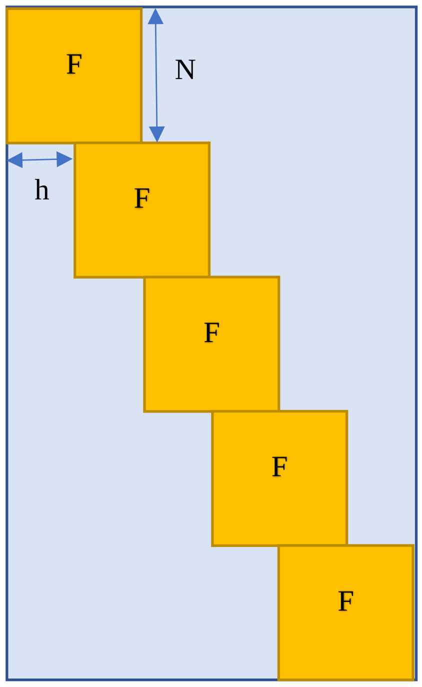
First let's just construct the basic DFT matrix with no windowing:

$$F_N^{jk} = \frac{1}{\sqrt{N}} e^{ijk\frac{2\pi}{N}} = \frac{1}{\sqrt{N}}\left(\cos\frac{jk2\pi}{N} + i\sin\frac{jk2\pi}{N}\right)$$

$$F = \begin{bmatrix} F_N^0 & F_N^0 & F_N^0 & & F_N^0 \\ F_N^0 & F_N^1 & F_N^2 & & F_N^{(n-1)} \\ F_N^0 & F_N^2 & F_N^4 & \cdots & F_N^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ F_N^0 & F_N^{(n-1)} & F_N^{2(n-1)} & & F_N^{(n-1)^2} \end{bmatrix}$$

Now to get actual matrix we want, we need to apply this 1024-point DFT to chunks of our vectorized signal. To do that, we create a matrix by essentially tiling this matrix F to get F' where h and N are used to determine how much to move the matrix F left and right before copying into F'. This picture tries to explain this:

F' =



Now to explain how this works. For example, let's say I only tiled once. Then my F' will look essentially like the DFT matrix. If our input signal has the same points as our DFT, we can multiply our vectorized data with this DFT matrix and see the distribution of frequencies that make up the data.

Now we add a hop, say in this case h=512. Concretely, say our input signal is 2048 points. Say we have chosen a 1024-point DFT with a hop of 512 points. If we construct a matrix F’ using the above logic, we will have a matrix that ends up with dimensions 4096x2048. Each row of this matrix is essentially a sinusoid at a frequency and because of the tiling pattern, we are multiplying this sinusoid with different segments of the signal. In this case, we split the segment into four chunks (the first 512 points, second 512 points, third 512 points, fourth 512 points) and run the DFT on each.

Now to get the vec(.) which is the vectorized DFT coefficients, we can simply run

$$DFT_{coef} = F'A$$