Due: February 24th, 2015 **Homework 2** CPSC 476

In this assignment you will implement the FastICA algorithm and apply it to several example datasets. Following is a review of ICA, that fills in a few of the missing steps in the posted tutorial.

The ICA Problem

Say we have a set of d independent random variables s_j ($j \in \{1, \ldots, d\}$) with non-Gaussian distributions, and x_i ($i \in \{1, \ldots, d\}$) are linear combinations of the s_j:

$$x_i = \sum_{j=1}^{d} a_{ij} s_j \tag{1}$$

Let $\mathbf{s} = (s_1, \ldots, s_d)^\mathsf{T}$ and $\mathbf{x} = (x_1, \ldots, x_d)^\mathsf{T}$ be random vectors formed from the s_j and x_i respectively. Then (1) in matrix vector notation is equivalent to

$$\mathbf{x} = A\mathbf{s}.$$

If we have n samples of $\mathbf{x}$, we can stack them side-by-side to form a $d \times n$ data matrix X whose columns consist of the samples of $\mathbf{x}$ we have observed. Corresponding to X is the set of samples of $\mathbf{s}$ that were mixed to form each column of X. Combining these similarly into a matrix S, we can write

$$X = AS.$$

We can think of the rows of S as d independent "source" signals, mixed via a weighted sum by the mixing matrix A, which lead to the observed signals in the rows of X (each row of A providing the set of weights responsible for the corresponding row of X). For instance, the source signals may be audio clips, and the observed signals may be weighted combinations of the audio clips. We might like to recover the original, unmixed clips from the mixed versions.

A slightly different perspective considers each column of A to be one "element" of a scene (e.g., an image), and these elements combine additively to form the full scene. In this case, each column of the data X might be an image, formed from a weighted combination of the elements (columns of A), with the weights for each image (columns of S) independent of one another (e.g., knowing how much one element is present in an image doesn't tell us about how much any of the others are present). More formally, the columns of A form a basis for the space of images in such a way that the coefficients needed to represent the images are independent of one another.

From either perspective, we need to recover A (or A^{-1}). Is this possible, given only X? Perhaps surprisingly, the answer is yes, up to a few (largely inconsequential) ambiguities. The key insight for our approach is provided by the central limit theorem, which states that the distribution of the sum of a set of independent random variables converges to a normal distribution as the number of variables goes to infinity. In general, this means that a sum of independent random variables is "more Gaussian" than each variable individually (assuming none of the variables are themselves Gaussian). For the ICA problem, the rows of X look more like samples from a Gaussian than the rows of S (whenever A performs mixing, i.e., isn't diagonal). Thus, we just need to find a matrix W such that WX has rows that are as non-Gaussian as possible (and linearly independent). The best W will of course be A^{-1}, so that $WX = WAS = A^{-1}AS = S$.

Measuring Non-Gaussianity

How can we measure non-Gaussianity? A common measure is kurtosis, which intuitively speaking measures the "peakiness" of a distribution. Kurtosis for a zero mean random variable Y is given by

$$\text{kurt}[Y] = \text{E}\big[Y^4\big] - 3\,\text{E}\big[Y^2\big]^2,$$

and the Gaussian distribution has a kurtosis of 0 (achieving this desirable property is part of what makes the definition a bit messy). Using the absolute value or the square of kurtosis provides a measure of deviation from Guassianity (although there are some non-Gaussian distributions with a kurtosis of 0, these are rarely encountered in practice). A drawback to the use of kurtosis, at least when approximated from sampled data, is its sensitivity to outliers.

Another measure of non-Gaussianity relies on results from information theory. The entropy of a random variable is a measure of the expected "information content" of the random variable. The information content associated with a given event e is defined to be

$$I(e) = \log\left(\frac{1}{P(e)}\right) = -\log(P(e)), \tag{2}$$

where $P(e)$ is the probability of event e. Some intuitive motivation for this definition can be found in recognizing that infrequent events should have higher information content, and the information content associated with the occurrence of two independent events e_1 and e_2 (associated with two random variables) should follow $I(e_1 \wedge e_2) = I(e_1) + I(e_2)$. Entropy is thus

$$H(Y) = \text{E}[I(Y)] = -\sum_y P(y)\log P(y).$$

By Shannon's source coding theorem, the entropy of a random variable provides a lower

bound on the expected codeword length of an optimal code used to compress a sequence of independent samples of the variable (random variables with higher entropy are harder to compress). This wonderful result means you can upper-bound the entropy of a set of data by using a compression algorithm, e.g., `zip` or similar!

For continuous random variables, an analogous formula ("differential entropy") holds, replacing the sum with an integral and the probability mass function with a probability density function. This is usually denoted $h(Y)$, and has a neat relation to discrete entropy: the discrete entropy of an n-bit quantization of a continuous random variable Y is approximately $h(Y) + n$.

How does this relate to measuring non-Gaussianity? It can be shown that the differential entropy of a Gaussian random variable upper-bounds the entropy for all other random variables with the same variance. In other words:

$$J(Y) = h(Y_{\mathrm{G}}) - h(Y) \geq 0,$$

where Y_{G} is a Gaussian random variable with the same variance as Y. $J(Y)$ is called the "negentropy" of Y, and provides a measure of deviation from Gaussianity.

Robust estimators of negentropy have been developed. In particular, it can be shown that

$$J(Y) \propto (\mathrm{E}[G(Y)] - \mathrm{E}[G(Z)])^2,$$

where Z is assumed to be Gaussian with zero mean and unit variance, Y is similarly assumed to have zero mean and unit variance, and G is some nonquadratic function. The assumptions on Y can be enforced in practice, as we will see below, and the proportionality doesn't matter since we simply want to maximize J. By picking $G(y)$ so that it doesn't grow too quickly as y gets large, we obtain a non-Gaussianity measure significantly better than the sample kurtosis.

Preprocessing

Enforcing the assumption that our data is zero mean is easy: simply subtract the mean across the columns from our data matrix X. $\hat{X} = X - \overline{\mathbf{x}}\mathbb{1}^{\mathsf{T}}$ is zero mean ($\overline{\mathbf{x}} = \frac{1}{n}X\mathbb{1}$ being the mean across the columns of X, and $\mathbb{1}$ the all ones vector), so our recovered $\hat{S} = W\hat{X}$ will be as well (mixing zero mean signals results in zero mean signals), and then $S = \hat{S} + W\overline{\mathbf{x}}\mathbb{1}^{\mathsf{T}}$ (i.e., we can add back the mean at the end).

Enforcing the unit variance assumption is slightly trickier. Once our data is zero mean, we

can estimate the covariance between x_i and x_j using the sample covariance matrix

$$\mathrm{E}[(x_i - \mu_i)(x_j - \mu_j)] = \mathrm{E}[x_i x_j] \approx \left(\frac{1}{n-1}\hat{X}\hat{X}^\mathsf{T}\right)_{ij}.$$

We'd like to transform our data in such a way that the transformed data is decorrelated with unit variance, i.e., $\tilde{X} = M\hat{X}$ and $\frac{1}{n-1}\tilde{X}\tilde{X}^\mathsf{T} = I$. Finding an M that does this (a "whitening matrix") turns out to be fairly easy. Recall (or be informed) that we can use the singular value decomposition of $\hat{X} = U\Sigma V^\mathsf{T}$ to uniquely express it as the product of an orthogonal matrix U ($d \times d$), a diagonal matrix Σ ($d \times d$), and another orthogonal matrix V ($n \times d$). Note that the dimensions given here are only valid when $n \geq d$, i.e., there are at least as many samples as dimensions—in general, Σ will be $d \times n$ and V will be $n \times n$, but since all of the rightmost $(n-d)$ columns of Σ are nothing but zeros when $n \geq d$, we can pitch them along the rightmost $(n-d)$ columns of V (bottom rows of V^T) without losing anything. In MATLAB, you can get this version of the SVD using `svd(X, 'econ')`. So,

$$\frac{1}{n-1}\tilde{X}\tilde{X}^\mathsf{T} = \frac{1}{n-1}M\hat{X}\hat{X}^\mathsf{T}M^\mathsf{T} = \frac{1}{n-1}MU\Sigma V^\mathsf{T}V\Sigma U^\mathsf{T}M^\mathsf{T} = \frac{1}{n-1}MU\Sigma^2 U^\mathsf{T}M^\mathsf{T}.$$

Letting $M = \sqrt{n-1}\,\Sigma^{-1}U^\mathsf{T}$ (where for Σ^{-1} we just invert its nonzero entries) we get

$$\frac{1}{n-1}MU\Sigma^2 U^\mathsf{T}M^\mathsf{T} = \Sigma^{-1}U^\mathsf{T}U\Sigma^2 U^\mathsf{T}U\Sigma^{-1} = I$$

since $U^\mathsf{T}U = I$. Applying M to our data gives, $\tilde{X} = M\hat{X} = \sqrt{n-1}\Sigma^{-1}U^\mathsf{T}U\Sigma V^\mathsf{T} = \sqrt{n-1}V^\mathsf{T}$, our "whitened" data matrix, and we can easily undo the effects of whitening by applying $M^{-1} = \frac{1}{\sqrt{n-1}}U\Sigma$.

Whitening the data makes life much easier. Observe that $\tilde{X} = MX = MAS = \tilde{A}S$, and $I = \frac{1}{n-1}\tilde{X}\tilde{X}^\mathsf{T} = \tilde{A}\big(\frac{1}{n-1}SS^\mathsf{T}\big)\tilde{A}^\mathsf{T}$. Since the s_j are assumed to be independent, we already know SS^T is close to diagonal when n is large. If we assume the s_j have unit variance, then $\frac{1}{n-1}SS^\mathsf{T} \approx I$. This assumption is reasonable, since we can't estimate the variance of the s_j anyway—any change in the variance of an s_j can be absorbed by scaling the corresponding column of A, so that no change in the mixed data X is observed. Under this assumption, we're left with $I = \tilde{A}\big(\frac{1}{n-1}SS^\mathsf{T}\big)\tilde{A}^\mathsf{T} \approx \tilde{A}\tilde{A}^\mathsf{T}$, showing that the mixing matrix that gave rise to our whitened version of the data is orthogonal. This means we only need to recover $\tilde{W} = \tilde{A}^{-1} = \tilde{A}^\mathsf{T}$, an orthogonal matrix. Once we have $\tilde{W}$, we can easily get the original W and A, since:

$$\begin{aligned} \tilde{W} = \tilde{A}^{-1} = (MA)^{-1} = A^{-1}M^{-1} \quad &\Rightarrow \quad W = A^{-1} = \tilde{W}M, \\ \tilde{W}^\mathsf{T} = \tilde{A} = MA \quad &\Rightarrow \quad A = M^{-1}\tilde{W}^\mathsf{T}. \end{aligned}$$

Deriving FastICA

To summarize, with $Y = \tilde{W}\tilde{X}$, we want to maximize $J(Y) \propto (\mathrm{E}[G(Y)] - \mathrm{E}[G(Z)])^2$, with respect to $\tilde{W}$ (subject to the constraint that $\tilde{W}$ is an orthogonal matrix). We can do this one row of $\tilde{W}$ at a time. We'd like to end up with $\tilde{W} = \tilde{A}^{-1} = \tilde{A}^{\mathsf{T}}$.

Let $\mathbf{w}^{\mathsf{T}}$ be a row of $\tilde{W}$. We must maximize $J(\mathbf{w}^{\mathsf{T}}\tilde{X}) = \left(\frac{1}{n}G(\mathbf{w}^{\mathsf{T}}\tilde{X})\mathbb{1} - \mu_{G(Z)}\right)^2$ subject to the constraint that $\|\mathbf{w}\| = 1$ (here we assume G gets applied pointwise to the vector argument, and we take the sample mean by multiplying against $\frac{1}{n}\mathbb{1}$). This is easily done using the method of Lagrange multipliers. In short, we can travel within our set of constrained $\mathbf{w}$ and improve $J(\mathbf{w}^{\mathsf{T}}\tilde{X})$ whenever $\nabla_{\mathbf{w}} J(\mathbf{w}^{\mathsf{T}}\tilde{X})$, the direction of maximal increase in J with respect to $\mathbf{w}$, isn't orthogonal to our constraint set (because if it's not orthogonal, we can go at least a little bit "uphill" without violating the constraint). We can rewrite our constraint $F(\mathbf{w}) = \mathbf{w}^{\mathsf{T}}\mathbf{w} - 1 = 0$, which makes it clear the constraint is just a level set of F, and so the direction orthogonal to it is just the gradient $\nabla F(\mathbf{w})$. So our (local) optima of $J(\mathbf{w}^{\mathsf{T}}\tilde{X})$ occur when these two gradients are parallel (though not necessarily the same length), which we can write:

$$
\begin{aligned}
\nabla F(\mathbf{w}) &\propto \nabla_{\mathbf{w}} J(\mathbf{w}^{\mathsf{T}}\tilde{X}) \\
&\Updownarrow \\
\nabla(\mathbf{w}^{\mathsf{T}}\mathbf{w} - 1) &\propto \nabla_{\mathbf{w}}\left(\frac{1}{n}G(\mathbf{w}^{\mathsf{T}}\tilde{X})\mathbb{1} - \mu_{G(Z)}\right)^2 \\
&\Updownarrow \\
2\mathbf{w} &\propto 2\left(\frac{1}{n}G(\mathbf{w}^{\mathsf{T}}\tilde{X})\mathbb{1} - \mu_{G(Z)}\right)\frac{1}{n}\tilde{X}G'(\tilde{X}^{\mathsf{T}}\mathbf{w}) \\
&\Updownarrow \\
\beta\mathbf{w} &= \frac{1}{n}\tilde{X}g(\tilde{X}^{\mathsf{T}}\mathbf{w}) \qquad (g = G', \text{ consolidate scalars into } \beta) \\
&\Updownarrow \\
0 &= \frac{1}{n}\tilde{X}g(\tilde{X}^{\mathsf{T}}\mathbf{w}) - \beta\mathbf{w} \qquad (3)
\end{aligned}
$$

The transition from the second to the third line might not be obvious at first glance, but it just involves applications of matrix calculus generalizations of the product rule and chain rule (using denominator layout notation and the fact that G is applied pointwise).

We're now left with an equation (3) in $\mathbf{w}$ that we'd like to solve. Call the right hand side of (3) $L(\mathbf{w})$. We want to find roots of L. A common iterative technique for finding roots of a function is Newton's method, in which we repeatedly approximate the function at a point

linearly, find the root of the linear approximation, and then repeat using the solution as a new point until we converge. For a function of a single variable $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$, this looks like:

$$\begin{aligned} 0 &= f(x_t) + f'(x_t)(x_{t+1} - x_t) \\ &\Updownarrow \\ x_{t+1} &= x_t - \frac{f(x_t)}{f'(x_t)}. \end{aligned}$$

In the multivariable case, the picture is identical, but the update involves the Jacobian of f, J_f (matrix of its partial derivatives):

$$\mathbf{x}_{t+1} = \mathbf{x}_t - J_f^{-1}(\mathbf{x}_t) f(\mathbf{x}_t).$$

The Jacobian of L is given by

$$J_L(\mathbf{w}) = \frac{1}{n} \tilde{X} \operatorname{diag}\Big(g'(\tilde{X}^\mathsf{T}\mathbf{w})\Big) \tilde{X}^\mathsf{T} - \beta I,$$

where $\operatorname{diag}(\mathbf{v})$ creates a matrix with $\mathbf{v}$ along the main diagonal. Inverting J_L isn't difficult analytically, but can be reasonably expensive computationally depending on the size of the data matrix. The first term in the expression is equivalent to $\frac{1}{n}\sum_i^n \mathbf{x}_i \mathbf{x}_i^\mathsf{T} g'(\mathbf{w}^\mathsf{T}\mathbf{x}_i)$, which approximates $\mathrm{E}[\mathbf{x}\mathbf{x}^\mathsf{T} g'(\mathbf{w}^\mathsf{T}\mathbf{x})]$ (here $\mathbf{x}_i$ is the i^{th} whitened sample of the data). Since we've whitened the data, we can make a simplifying assumption, namely that $\mathrm{E}[\mathbf{x}\mathbf{x}^\mathsf{T} g'(\mathbf{w}^\mathsf{T}\mathbf{x})] \approx \mathrm{E}[\mathbf{x}\mathbf{x}^\mathsf{T}]\,\mathrm{E}[g'(\mathbf{w}^\mathsf{T}\mathbf{x})] = \mathrm{E}[g'(\mathbf{w}^\mathsf{T}\mathbf{x})] I \approx \Big(\frac{1}{n} g'(\mathbf{w}^\mathsf{T}\tilde{X})\mathbb{1}\Big) I$. The intuitive appeal is clear, although showing that we still converge to the right locations requires a proof.

So, approximating $J_L(\mathbf{w})^{-1}$ by $\Big(\frac{1}{n} g'(\mathbf{w}^\mathsf{T}\tilde{X})\mathbb{1} - \beta\Big)^{-1} I$, we have the (approximate) Newton iteration

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\tilde{X} g(\tilde{X}^\mathsf{T}\mathbf{w}_t) - \beta \mathbf{w}_t}{g'(\mathbf{w}_t^\mathsf{T}\tilde{X})\mathbb{1} - \beta} \\ &\Updownarrow \\ \Big(\beta - g'(\mathbf{w}^\mathsf{T}\tilde{X})\mathbb{1}\Big)\mathbf{w}_{t+1} &= \beta \mathbf{w}_t - \Big(g'(\mathbf{w}_t^\mathsf{T}\tilde{X})\mathbb{1}\Big)\mathbf{w}_t + \tilde{X} g(\tilde{X}^\mathsf{T}\mathbf{w}_t) - \beta \mathbf{w}_t \\ &\Updownarrow \\ \mathbf{w}_{t+1} &\propto \tilde{X} g(\tilde{X}^\mathsf{T}\mathbf{w}_t) - \Big(g'(\mathbf{w}_t^\mathsf{T}\tilde{X})\mathbb{1}\Big)\mathbf{w}_t. \end{aligned}$$

Since we know $\mathbf{w}$ should be unit length, and the above update gives us the direction (we know it's proportional to the right hand side), we can simply do the update in two steps:

1. $\mathbf{w}_{t+1}^{+} = \tilde{X} g(\tilde{X}^{\mathsf{T}} \mathbf{w}_t) - \Big(g'(\mathbf{w}_t^{\mathsf{T}} \tilde{X}) \mathbb{1} \Big) \mathbf{w}_t$
2. $\mathbf{w}_{t+1} = \mathbf{w}_{t+1}^{+} / \left\| \mathbf{w}_{t+1}^{+} \right\|$

This can be iterated until a desired convergence is achieved: if the dot product between $\mathbf{w}_{t+1}$ and $\mathbf{w}_t$ is sufficiently close to 1 (or -1, since the sign of the independent components is arbitrary), say $|\mathbf{w}_{t+1}^{\mathsf{T}} \mathbf{w}_t| > 1 - 10^{-9}$, no more iterations are necessary. Of course, since Newton's method isn't guaranteed to converge, putting a maximum limit on the number of iterations is a good idea (cleverer approaches that detect slow convergence and modify the size of the update step are also possible).

Multiple Components

What about recovering multiple independent components? Because we're trying to find $\tilde{W} = \tilde{A}^{\mathsf{T}}$, the unmixing matrix for the whitened data, we know all of the rows are orthogonal to one another. So once we've recovered one row, we can search for additional rows in the subspace orthogonal to the first row. This entails only a small change to the algorithm above. If the weight vectors we have recovered so far are stored in the rows of $\tilde{W}_*$, we can compute the length of the projection of $\mathbf{w}_{t+1}^{+}$ onto all previous recovered weight vectors by $\tilde{W}_* \mathbf{w}_{t+1}^{+}$. We can use the result to subtract off the components of $\mathbf{w}_{t+1}^{+}$ lying in the directions of previous weight vectors. To do this, simply compute

$$\mathbf{w}_{t+1}^{+} - \tilde{W}_*^{\mathsf{T}} \tilde{W}_* \mathbf{w}_{t+1}^{+}.$$

This scales all the previous weight vectors by the length of $\mathbf{w}_{t+1}^{+}$'s projection onto them, sums them up, and subtracts from $\mathbf{w}_{t+1}^{+}$. You can verify the result is indeed orthogonal to all previous weight vectors by computing

$$\tilde{W}_* \Big(\mathbf{w}_{t+1}^{+} - \tilde{W}_*^{\mathsf{T}} \tilde{W}_* \mathbf{w}_{t+1}^{+} \Big) = \tilde{W}_* \mathbf{w}_{t+1}^{+} - \tilde{W}_* \tilde{W}_*^{\mathsf{T}} \tilde{W}_* \mathbf{w}_{t+1}^{+} = \tilde{W}_* \mathbf{w}_{t+1}^{+} - \tilde{W}_* \mathbf{w}_{t+1}^{+} = 0,$$

since $\tilde{W}_* \tilde{W}_*^{\mathsf{T}} = I$ (note that $\tilde{W}_*$ has fewer rows than columns so $\tilde{W}_*^{\mathsf{T}} \tilde{W}_* \neq I$). To avoid computing $\tilde{W}_*^{\mathsf{T}} \tilde{W}_*$ repeatedly, you can maintain a projection matrix P initialized to 0, and update it whenever you recover an additional weight vector by $P_{\text{new}} = P_{\text{prev}} + \mathbf{w}\mathbf{w}^{\mathsf{T}}$. Convince yourself that $P = \tilde{W}_*^{\mathsf{T}} \tilde{W}_*$!

1 Implement FastICA. Your implementation should have the MATLAB signature

```
[W, A] = fastica(data, numics)
```

where `W` and `A` are the recovered unmixing and mixing matrices, respectively, and `numics` specifies how many independent components to recover. The algorithm has four steps, all described (somewhere) above:

1. Demean the data
2. Whiten the data (remember to use `svd(data, 'econ')`)
3. Find `numics` rows of $\tilde{W}$ for the whitened data
4. Recover W and A from $\tilde{W}$

For g (the contrast function), use:

$$g(u) = \tanh(u)$$
$$g'(u) = 1 - \tanh^2(u)$$

For the stopping criteria, use $|\mathbf{w}_{t+1}^{\mathsf{T}}\mathbf{w}_t| > 1 - 10^{-9}$ and a maximum of 1000 iterations per component.

There are several example datasets included, which are stored in `data.mat`. When you load this in MATLAB (`load data` from the homework folder), you will get three variables:

`artificial` An artifical data set for testing purposes. Make sure your algorithm works well on this before moving on to the other data sets. The recovered independent component signals can be plotted via `plot(W(i,:)*artificial)`, where `W` is the unmixing matrix and `i` (1 or 2) is the independent component you want to recover. The components will look like a sin wave and a sawtooth wave (you'll know when you get it right). Another good test is to make sure `var(W(i,:)*artificial)` is 1.

`audio1/2/3` Some stereo audio clips. You can play e.g. the first channel of the first clip via `soundsc(audio1(1,:), 44100)`. Note the 44100 (sampling rate of the audio clips). Don't forget this or it will play veeerrry slooooowwwly. Listen carefully to the first clip (preferably with headphones)... note the presence of guitar, drum, and bass in both

channels. When you've run ICA on this clip, you can play the unmixed channels with `soundsc(W*audio1(1,:), 44100)` and `soundsc(W*audio1(2,:), 44100)`. What do you observe? What about this audio clip is conducive to the ICA formulation? Run on the other clips as well. How do the results compare? Why for example does ICA not seem to work on `audio3`? Bonus points of undetermined value: identify the music.

`patches` A set of 40000 12×12 natural image patches. Once you've run ICA on these (which might take a little bit—limit your `numics` to 30 or so), you can visualize the ICA basis functions (columns of A) using the `displaycolumns` function (e.g., `displaycolumns(A)`). You can similarly visualize the corresponding filters (columns of W^{T}). What do you observe?

These datasets are all you are required to run on, but if you have another dataset in mind that you think matches the ICA problem setup well, please feel free to run on that too! Submit a brief writeup describing your results, including a figure from `displaycolumns(A)` after running on the natural image patches, as well as your `fastica.m` file.