Vignesh Ravindranath
vgr325

Independent Component Analysis (ICA)

1 Overview

The purpose of this assignment is to build an algorithm to successfully separate mixed signals (blind source separation). In other words, if we were to mix some source signals **U (n,t)**, where **n** is the number of signals and **t** is the length of each signal, with a mixing matrix **A (m,n)**, where **m** is the number of mixed signals, we would be left with a mixed signals **X (m,t)** = **AU**. Our goal is to recover the source signals (**U**) with minimal information about the source data (**U**). More formally, we are looking for an unmixing matrix **W (n,m)** = $\mathbf{A}^{-1}$, such that **Y = WX** is the recovered signal. Independent component analysis, or ICA, is one such method to recover data with minimal or no information about the source data.

2 Method

First we can characterize the distributions of each source signal. Suppose each source signal U_j can be described by a density p_U; the joint distribution of the source **U** is given by:

$$p(U) = \prod_{j=1}^{n} p_U(U_j).$$

Since the mixed signal **X** is simply a linear transformation of the original signal **U**, we can expect the distribution of the mixed signal **X** to be similar to the distribution of the original signal **U**. The distribution of the mixed signal **X** is given by:

$$p(X) = \prod_{j=1}^{n} p_U(WX) \cdot |W|,$$

where $\mathbf{W} = \mathbf{A}^{-1}$.

As we can see from the distribution above, we have two unknowns: **W** and p_U. We can *guess* as to what p_U may be by first defining a cumulative distribution function (cdf) for the original signal **U**. Then, the distribution p_U is simply the derivative of that cdf. According to material by Andrew Ng [1], the sigmoid function $g(X) = 1/(1 + \exp(-X))$ is a reasonable *default* cdf; thus, $p_U(X) = g'(X)$. Any distribution can be used as long as it is non-Gaussian. Now, the only unknown is **W**.

To solve for **W**, we can use gradient ascent to maximize the log likelihood of **W** (substitute g'(X) for p_U and take the log of the distribution above):

$$l(W) = \sum \left(\sum \log g'(WX) + \log|W| \right).$$

Then, the gradient ascent algorithm can be described as:

$$W := W + \eta(I * t + (1 - 2 * Z) * Y^T) * W,$$

where η is the learning rate (1e-6), t is the length of each source signal, $Y = WX$ is the *recovered* data, and $Z = g(Y)$. After a set number of iterations (*R_max*=1000), we will have an approximate unmixing matrix **W**, such that **Y = WX**.

3 Experiments

The three variables I will be considering in this homework are: η (learning rate), *R_max* (number of iterations to update W), and **m** (the number of mixed signals). I will use the *numpy.linalg.norm* function and the Pearson correlation coefficient (*correlation_coefficient*) to determine how close the recovered signal **Y** is from the original signal **U**.

From initial trial-and-error, it was found large learning rates (>1e-6) would lead to an overflow at any iteration above 100; so, I tested learning rates ≤1e-6. Moreover, I noticed the ICA algorithm struggled when it had to recover a larger number of signals. **Table 1** below indicates the correlation coefficient found for *n* recovered signals (*sounds.mat*), where *n* is between 2 and 5, *R_max*=1000, and **m=n**.

Table 1

n	Correlation Coefficient	Best η
2	0.9764	1e-6
3	0.7349	1e-7
4	0.5325	1e-8
5	0.3631	1e-9

The correlation coefficients seem to decrease with more signals. Moreover, smaller learning rates were required to obtain (relatively) better correlation coefficients. Since a learning rate of 1e-6 recovered approximately 97% of the first two signals, I will be using a learning rate of 1e-6 for future experiments.

Next, I found that the number of iterations (1000, 10000, 100000) produced very similar results and errors (*numpy.linalg.norm* and *correlation_coefficient*). The only difference between iterations was the run time to loop through each value of *R_max*; larger values of *R_max* took longer time – 5sec, 60sec, and 300sec respectively. So, the learning rate was set to 1e-6 and *R_max* was set to 1000.

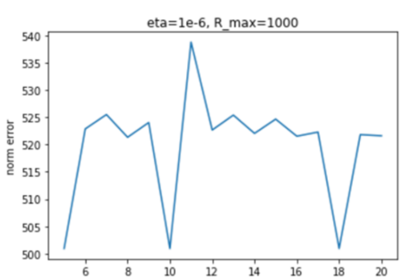

Figure 1: Norm error (numpy.linalg.norm) for values of **m** ranging from 5 to 20

Lastly, I experimented with the number of mixed signals **m** produced by the mixing matrix. **Figure 1** illustrates the error between the recovered and original signals. As seen by **Figure 1**, there is no clear pattern as to what values of **m** are optimal to reduce the recovered signal error. The lowest error was found when **m = n = 5** (*sounds.mat*), and so for future experiments, **m** will be set to **n**.

After setting the learning rate to 1e-6, *R_max* to 1000, and **m** to **n**, I mixed and recovered the signals provided by the *sounds.mat* file. **Figure 2** illustrates the original, mixed, and recovered signals.

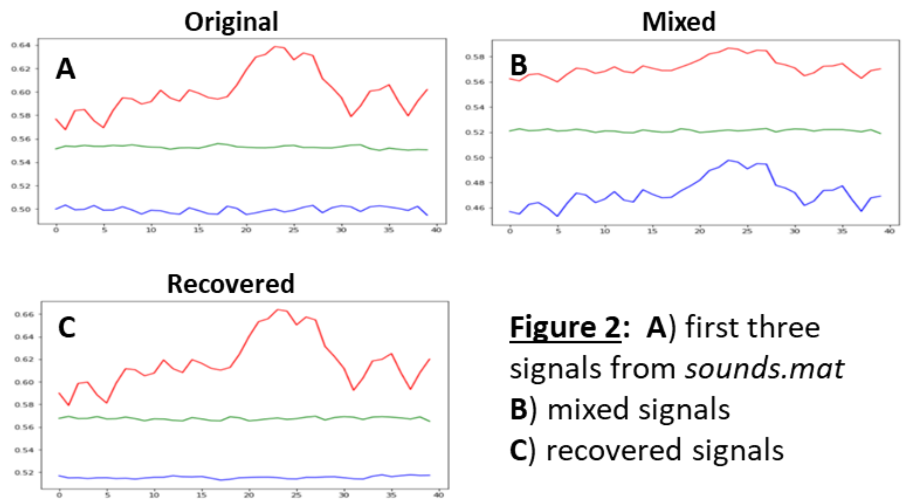

Figure 2: **A**) first three signals from *sounds.mat*
B) mixed signals
C) recovered signals

As seen by **Figure 2**, the original and recovered signals look comparable. However, there were more promising results for the data provided by the *icaTest.mat* file. **Figure 3** illustrates the original, mixed, and recovered signals.

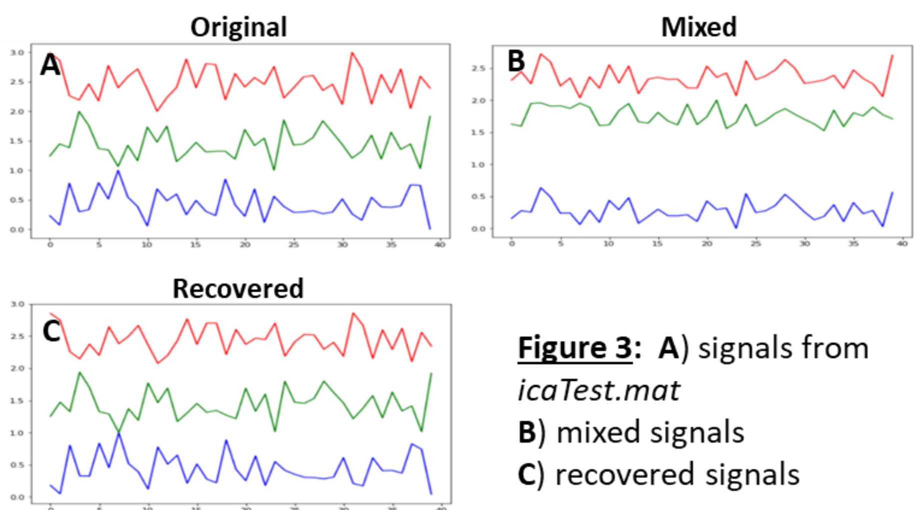

Figure 3: **A**) signals from *icaTest.mat*
B) mixed signals
C) recovered signals

The original and recovered signal look very similar, and this is supported by the high correlation coefficient (**0.9796**).

4 Results and Discussion

Overall, the ICA algorithm was able to successfully recover approximately 98% of the ICA test data, with a learning rate of 1e-6, an *R_max* of 1000, and with **m=n**. The program was prone to overflowing, thus requiring a very small learning rate. The number of iterations did not seem to improve the quality of the recovered images, indicating **W** must be converging at some *R_max* < 1000. I observed the recovered signals would not always be in the same order as the original data. With further investigation, I found after rearranging the recovered signals to match the order of the original signals, the correlation coefficient would increase significantly; however, the *numpy.linalg.norm* function would be consistent with its results. The ICA algorithm was an interesting algorithm to learn and seemed to recover original data with limited information.

5 References

a) time library – time
b) scipy library – io.loadmat
c) numpy library – linalg.norm
d) matplotlib library – pyplot

[1] http://cs229.stanford.edu/notes2020spring/cs229-notes11.pdf