Received: 00 Month 0000 | Revised: 00 Month 0000 | Accepted: 00 Month 0000

DOI: xxx/xxxx

ASSIGNMENT

Independent Component Analysis Using Maximum Likelihood†

Joel Walsh*

[1]STEM Education, University of Texas at Austin, Texas, United States

Correspondence
*Joel Walsh. Email: joelawalsh@utexas.edu

Summary

Independent Component Analysis (ICA) is an algrothm used for Blind Source Seperation (BSS) This implementation uses a data set of five sounds, and a method of maximum likelihood estimation pioneered by Bell and Sejnowski (1995).

KEYWORDS:
ICA, Blind Source Seperation, gradient ascent, maximum likelihood

1 | INTRODUCTION

Independent Component Analysis is a method that can best be motivated by the "Cocktail Party Problem". Imagine yourself in a crowded room. You suddenly hear a voice that you recognize, amidst the white noise of chatter and clanking glass. Some would argue that you are conditioned to pick out non-Guassian signals, and that your brain is using that to discern individual voices. The data set used for this analysis are a collection of 4.4 second long sound clips. Four are common noises such as a drill or clapping. One is the fictional character Homer Simpson. Our task was to use a random matrix of weights to mix the signals, and then use ICA to discern the inverse of these weights. This inverse could be applied to a mixture matrix to "unmix" them. To find this inverse, A simple gradient descent model is used to infer the inverse of the weights function.

2 | METHODS

As far as preprocessing goes, I first centered and whitened (sphering) the data. Centering caused some problems within the whitening function, which required a square root at a certain portion. I removed the centering function, and after examining the data, determined that it must have already been centered. Re-centering may have caused some values to veer just a touch negative, even though the mean should have technically been zero.

To start the gradient portion, I created a function that produced the actual gradient update. I used the derivation of Bell and Sejnowski's method based on the professor's notes. The original equation,

$$W := W + \alpha \left(\begin{bmatrix} 1 - 2g\left(w_1^T x^{(i)}\right) \\ 1 - 2g\left(w_2^T x^{(i)}\right) \\ \vdots \\ 1 - 2g\left(w_n^T x^{(i)}\right) \end{bmatrix} x^{(i)^T} + \left(W^T\right)^{-1} \right) \tag{1}$$

Can be derived into the friendlier form,

$$\nabla \mathbf{W} = \boldsymbol{\eta} \left(\mathbf{I} + (\mathbf{1} - \mathbf{2Z})\mathbf{Y}^{\mathbf{T}}\right) \mathbf{W} \tag{2}$$

through a bit of algebra. So the bulk of the programming focused on acheiving a numerically stable way to calulate $\nabla \mathbf{W}$.

†

[0]**Abbreviations:** ANA, anti-nuclear antibodies; APC, antigen-presenting cells; IRF, interferon regulatory factor

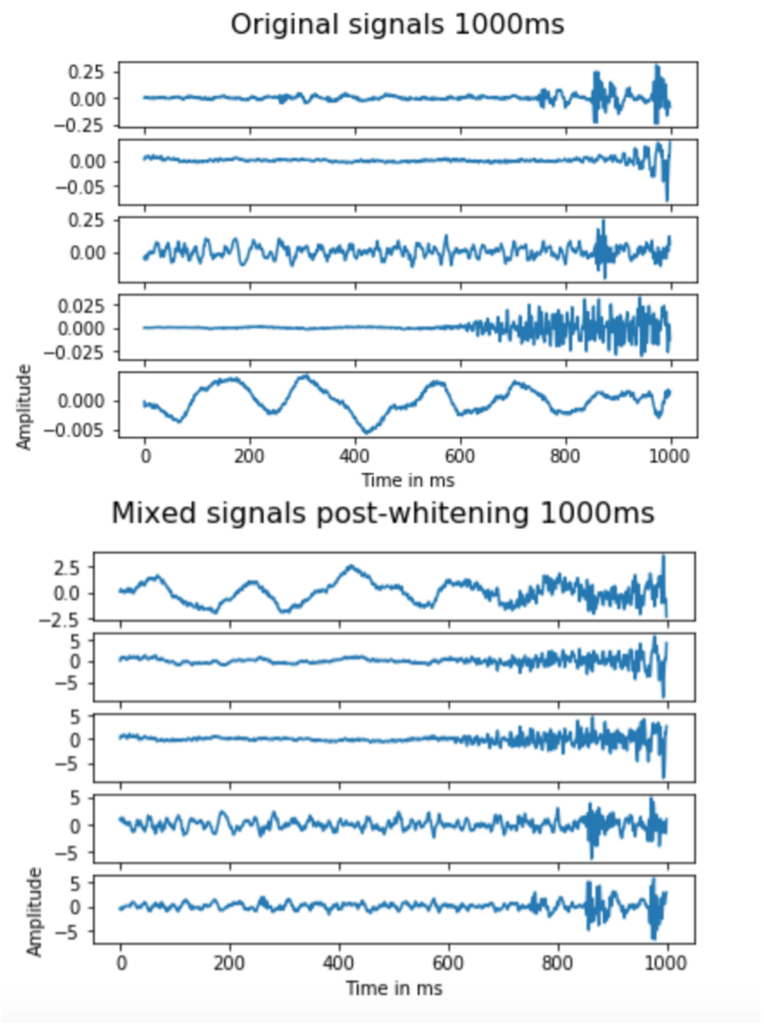


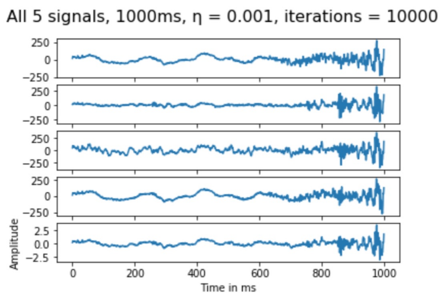
FIGURE 1 Original and mixed signals.

The value of η was built into the function "gradupdate". Within this function I had a number of problems. I initially used math.exp to calculate the sigmoid. I ran into many numerical problems with this, in the form of either vanishing or exploding gradients. I tried a number of other activation functions, but eventually switched from the built-in math function to using numpy's exp function, which was initially more stable. I then attempted to institute a sort of defacto floor and ceiling on the values within the gradient, but those methods provided unreliable. I eventually determined that the Z matrix formulation was not the problem, but that the (1-2*Z) part of my calculation was causing gradients to explode or vanish. I experienced some success curtailing this by changing to (1.-2.*Z) and initializing Z as a matrix of zeroes before filling in values using the sigmoid function, rather than as an empty array. Eventually I acheived some semblance of numerical stability. At each point I experimented with different η values.

This stability was short-lived however. After swithiching from numpy's matmul method to dot for matrix multiplication, the gradient problems persisted. The best results eventually came from a practice called "gradient clipping", whereby each gradient is multiplied by η), but also divided by the l2 norm of the gradient. This initially sets a floor or ceiling on how high or low the gradient can get.

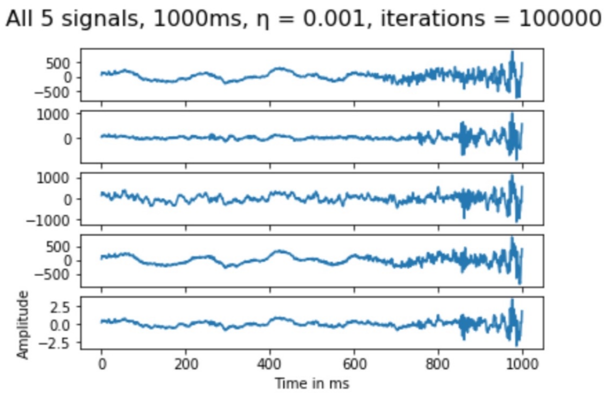
$$\mathbf{W} \leftarrow \frac{\eta \mathbf{W}}{||\mathbf{W}||} \tag{3}$$

This practice gave the best numerical stability. There is some empirical evidence that this can also improve learning times, although no theoretical explanation has presented itself. (Zhang et al, 2020)



```
All 5 signals, 1000ms, η = 0.001, iterations = 10000  Pearson Correlations,  i = original, j = mixed
[[-0.24877768 -0.80807957 -0.31386005 -0.23358416 -0.51673523]
 [ 0.46864982  0.50509131  0.51988101  0.49054143  0.52577182]
 [-0.13741422 -0.08684235 -0.56639455 -0.06822211 -0.21297793]
 [-0.42443173  0.02003444 -0.26756292  0.29827933 -0.16386448]
 [-0.71780016 -0.12983925 -0.4939242  -0.75085074 -0.59712601]]
```

FIGURE 2 10,000 iterations.



```
All 5 signals, 1000ms, η = 0.001, iterations = 100000  Pearson Correlations,  i = original, j = mixed
[[-0.24877768 -0.80807957 -0.31386005 -0.23358416 -0.5005354 ]
 [ 0.46864982  0.50509131  0.51988101  0.49054143  0.53770017]
 [-0.13741422 -0.08684235 -0.56639455 -0.06822211 -0.23544984]
 [-0.42443173  0.02003444 -0.26756292  0.29827933 -0.15652823]
 [-0.71780016 -0.12983925 -0.4939242  -0.75085074 -0.59446098]]
```

FIGURE 3 100,000 iterations.

3 | RESULTS

I chose to focus on the first 1000 milliseconds in order to make computation time feasible. I then mixed the data with random values from -10 to 10. To measure the match from the unmixed signal to the original, I used Pearson's correlation. A value of -1 or 1 would suggest perfect correlation between the symbols. Because ICA sometimes produces waves in different orders, I set up a matrix of Pearson correlations to determine which signal in the unmixed version likely correlated with the signal in the original signal. From this matrix it appears that the additional iterations did help some of the unmixed signals to approach a 0.9 Pearson's correlation threshold, albeit not quite there.

4 | SUMMARY

Like with any gradient ascent or descent algorithm, a number of numerical issues can arise due to the use of differing packages for matrix operations. Numpy's matmul method appeared to be the culprit for some of the difficulties with vanishing and exploding gradients. Gradient clipping seemed to be the best fix for these troubles. Some signals were easier to sepearate than others, such as with the second sound in the original set, although the reason for this was not clear.

How to cite this article: Walsh J. (2020), Independent Component Analysis using Maximum Likelihood Estimation