CS391L HW2: Independent Component Analysis

Mingyo Seo
UT EID: ms84662
Email: mingyo@utexas.edu

***Abstract*—In this assignment, an overview of Independent Component Analysis (ICA) is presented and is applied to separate a mixture of sounds. A set of 5 different sources were taken and mixed to obtain a 5 different "microphone" recordings. The ICA algorithm was applied to retrieve the individual sounds and the retrieval quality was examined.**

I. Introduction

The problem of seperating a certain signal from a mixture of multiple signals can be applied to many areas. The simplest example of this problem is the cocktail party problem where we have recordings from different microphones of the same mixture of conversations. In this paper, in particular, we implemented an ICA to seperate voices from the mixture of conversations. We also studied the effects of parameters on the model's performance. The answers to the HW2 questions are included in the following sections.

- Fig. **??**: Visualization of base source sounds
- Fig. **??**: Visualization of mixed source sounds
- Fig. **??**: Visualization of recovered source sounds
- Fig. 1, 2: Correlation between orignal and recovered sounds

II. Method

A. Mixing matrix

In this assignment, a test data set U containing 5 sound sources and t number of samples is given. We aim to mix n sources from the given sounds to generate a pseudo microphone output X of m recordings where $m \geq n$. To do this, we introduce a mixing matrix A, as

$$X = AU \tag{1}$$

Here, X is a $m \times t$ matrix of the mixed sound signals.

B. Independent Component Analysis

Contrary to Principal Component Analysis which finds the primary features in the given dataset, the ICA method decomposes a mixture of signals into its individual sources.

The goal of the ICA method is to find the un-mixing $n \times m$ matrix W, as

$$WA = I. \tag{2}$$

This yields of restoring the orignial signal U, as

$$U = WX. \tag{3}$$

We use the maximum likelihood estimate to find $\hat{W}$, an estimate of W, and the gradient descent to approximate it. These methods are described in the following subsections.

C. Maximum Likelihood Estimation

There are several methods available in literature that can be used to calculate the unmixing matrix W, [] for example uses a SVD (singular value decomposition) methods to calculate the matrix W. There are advantages and disadvantages to the available approaches. For example, the SVD approach involves calculation of the Eigenvalues of XX^T which may be slow if the number of microphone inputs (m) is large. The Maximum Likelihood Estimate (MLE) of W is calculated by maximizing the parameters of W to match the mixed data X. To do this, the following definition is used for the likelihood of the mixture X:

$$p(x = X) = p_x(X) = p_u(U) \cdot |W| = \prod_{i=1} p_i(u_i)|W| \tag{4}$$

where, $p_u(U)$ is the probability density function of the independent source signals, and $p_i(u_i)$ is the probability density of the i_{th} source component

Since we have t samples of all mixtures in X, we can calculate $P_x(X)$ as:

$$L(W) = \prod_{j=1}^{t} \prod_{i=1}^{n} p_i(w_i^T x_j)|W| \tag{5}$$

where, x_j is the j^{th} column of X.
Thus our goal is to maximize $L(W)$ over all W, i.e.:

$$\max_{W \epsilon R^{(n,m)}} L(W) \tag{6}$$

Given that the probability density function is always positive, we can maximize $L(W)$ by maximizing the log likelihood function. The reason for maximizing log likelihood is because, taking log of $L(W)$ changes the product terms in W into summation terms, i.e.:

$$\ln(L(W)) = \sum_{j=1}^{t} \sum_{i=1}^{n} \ln(p_i(w_i^T x_j)|W|) \tag{7}$$

$$= \sum_{j=1}^{t} \sum_{i=1}^{n} \ln(p_i(w_i^T x_j) + \sum_{i=1}^{n} \ln(|W|) \tag{8}$$

$$\therefore \ln(L(W)) = \sum_{j=1}^{t} \sum_{i=1}^{n} \ln(p_i(w_i^T x_j) + t \ln(|W|) \tag{9}$$

We can simplify equation 9 further as:

$$\frac{1}{t} \ln(L(W)) = E\left[\sum_{i=1}^{n} \ln(p_i(w_i^T x_j))\right] + \ln(|W|) \tag{10}$$

where E[] is the expectation/mean observation.
Here we introduce the concept of the cumulative density function (cdf). The cumulative density function $g(X)$ is the integral of the pdf, and gives the net probability of the function having a value below X. Thus the pdf $p(X)$ is simply the derivative of $g(X)$. Thus equation 10 becomes,

$$\frac{1}{t}\ln\left(L(W)\right) = E\left[\ln\left(g'(WX)\right)\right] + \ln\left(|W|\right) \tag{11}$$

There are many algorithms that are available for maximizing the the log-likelihood function. Gradient descent algorithm is a popular approach to maximize $L(W)$. Since we maximize with respect to W, the gradient is taken accordingly, i.e.:

$$\begin{aligned}&\frac{1}{t}\frac{\partial}{\partial W}\ln(L(W))\\ &= E\left[\frac{\partial}{\partial W}(\ln\left(g'(WX)\right)X^T\right] + \frac{\partial}{\partial W}\ln\left(|W|\right)\\ &= E\left[\frac{\partial}{\partial W}(\ln\left(g'(WX)\right)X^T\right] + \left[W^T\right]^{-1}\end{aligned} \tag{12}$$

D. Gradient Descent

The gradient descent method is an iterative algorithm to find a function's minimum or maximum points. If the convex function is convex, the gradient descent converges to the global extremums. Otherwise, it converges to the local extremums. Proving the convexity of the log likelihood $L(W)$ is beyond the scope of this report, so it is not described in this report.

Gradient descent starts at an initial point $\hat{W}_0$ and updates it by moving to next points recursively along the gradient computed at previous points. To formulate the gradient descent method in terms of W, the estimate $\hat{W}_k$ of the W matrix after the k^{th} iteration is given as,

$$\begin{aligned}\hat{W}_{k+1} &= \hat{W}_k + \eta\cdot\left(\frac{1}{t}\frac{\partial}{\partial W}\ln(L(W))\right)_{W=\hat{W}_k}\\ &= \hat{W}_k + \eta\cdot\Delta W,\end{aligned} \tag{13}$$

where η is the *learning rate* of the gradient descent. Here, the initial point $\hat{W}_0$ is given randomly, and η should be adjusted to acheive convergence.

From Equation 12, the gradient term contains $\left[W^T\right]^{-1}$ which requires expensive computation for the inverse operation. Thus, we process the interation by multiplying it with W^TW, which preservers the convergence to the optimum, and avoids the inverse operation, as

$$\begin{aligned}\Delta W &= \frac{1}{t}\frac{\partial}{\partial W}\ln(L(W))W^T\\ &= E\left[\frac{\partial}{\partial W}(\ln\left(g'(WX)\right)X^T\right]W^TW + \left[W^T\right]^{-1}W^TW\\ &= \left(E\left[\left(\frac{\partial}{\partial W}(\ln\left(g'(WX)\right)\right)(WX)^T\right]W + W\right)_{W=\hat{W}_k}.\end{aligned} \tag{14}$$

The above formulation does not affect the optimality of equation 6. Decription for this is omitted for brevity of this report, and we encourage to refer the class notes.

To facilitate the gradient descent algorithm, we need to choose a globally differentiable cdf. In particular, we used the following cdf,

$$g(WX) = \frac{1}{1+e^{-WX}}. \tag{15}$$

It is differentiable for all WX and is bounded in $[0, 1]$. The derivative of g is given as

$$g'(WX) = g(WX)(1 - g(WX)). \tag{16}$$

Inserting Equation 16 into Equation 14 yields

$$\Delta W = \left(\left(E\left[(1-2g(WX))(WX)^T\right] + I\right)W\right)_{W=\hat{W}_k}. \tag{17}$$

To summarize the processes described above, the algorithm is implemented as:

1) Start with an intial point $\hat{W}_0$.
2) Compute matrix $Z_k = g(\hat{W}_kX)$
3) Compute $\Delta W = \left(E\left[(1-2Z_k)(\hat{W}_kX)^T\right] + I\right)\hat{W}_k$.
4) Update the estimation as $\hat{W}_{k+1} = \hat{W}_k + \eta\cdot\Delta W$
5) Return to Step 2 and repeat the processes until it reaches the maximum iteration.

III. Results

The classifier model of PCA and KNN methods described in Section II is implemented python scripts (3.8.5).

A. Model Training

We trained with the first 1000 samples of the training set, and the result eigenvalues are presented in descending order in Fig. **??**. The eigenvectors of the principal components are presented in Fig. **??**. For evaluation of the model, we chose the principal component $M = 15$ and KNN size $K = 5$, and the model achieves an accuracy of 84.35 %.

B. Batch size

The plot of accuracy of with different number of trainign samples is presented in Fig. **??**. In the experiment, we used $M = 24$, and $K = 5$. There is trade-off in using a larger training sample size: the accuracy increases as a training sample size increases but computation time also increases linearly.

C. Learning rate

The plot of the accuracy of with different KNNs is presented in Fig. **??**. In the experiment, we used $M = 24$, and 1000 training samples. The accuracy for inferring worsens as KNN size increases, the computation time does not change much.

IV. Summary

The classifier method of PCA and KNN for classifying digit character images are implemented in the assignment. For evaluation, we used the MNIST dataset. From the results of the experiments, a smaller KNN size benefits the accuracy. However, for the PCA size, the accuracy peaked at $M = 50$, This implies that enough number of feature is required for estimation but too many features distract the proper labels,

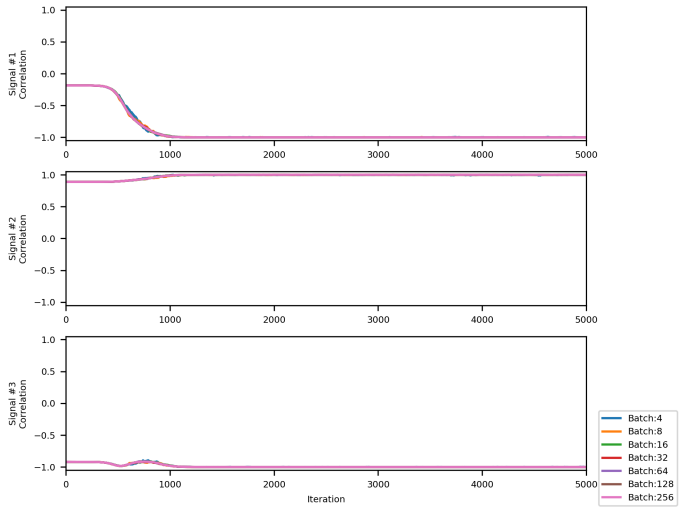

Fig. 1. Accuracy and computation time changes on training sample size: training sample sizes are chosen in [300, 500, 1000, 5000, 10000, 30000, 60000].

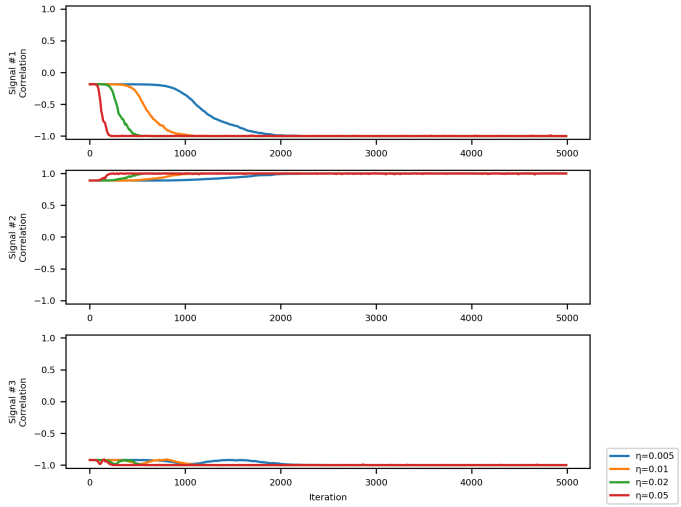

Fig. 2. Accuracy and computation time changes on KNN sizes: KNN sizes are chosen in [1, 3, 5, 10, 25].

which worsen the accuracy. On the other hand, a larger training size yields higher accuracy but also requires higher computation power.