Image separation routines

1. Introduction

This manual presents a set of MATLAB routines for linear and nonlinear image separation, based on the MISEP method [1-4]. More specifically, these routines were used to produce the results published in [4]. The set of routines is a modified version of the MISEP Toolbox, available at

http:// www.lx.it.pt/~lbalmeida/ica/mitoolbox.html,

and has a few additional routines for performing auxiliary functions.

Note: These routines are supplied only as a means to test the separation methods described in [4]. Although the author may occasionally be able to help users, no support is given. The routines contain code for implementing several options that are not covered in this manual (e.g. use of weight decay for regularization).

2. General description

The separation routines implement linear or nonlinear ICA, based on a training set of observations of mixture images. The separation is performed by a Multilayer Perceptron (MLP). In the case of linear separation, the MLP has a single layer of linear units, and thus simply performs a product by a matrix. In the case of nonlinear separation, the MLP has additional sigmoidal hidden units, and thus performs a nonlinear operation.

Figure 1 shows the structure of the network during training. See references [1-3] for the description of the training method. The separation MLP is block **F**. The separated components are y_1 and y_2. During the optimization (training) of the separation network, auxiliary MLPs (denoted ψ_i in the figure) are used.

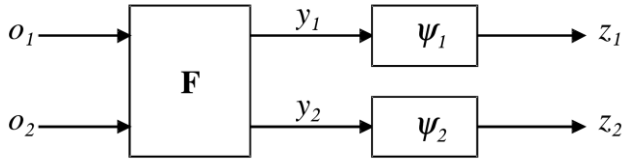

Figure 1. Structure of the separation network.

3. Files and variables

The files contained in the distribution are the following:

a) Main script files (intended to be called by the user)

`netparlin.m`	set parameters for training the linear separator used in [4]
`netparnonlin.m`	set parameters for training the nonlinear separator used in [4]
`selectdata.m`	randomly select data from mixture images, to form a training set
`netinit.m`	initialize MLPs for training
`train.m`	train the MLPs
`processimg.m`	process mixture images with the trained MLP
`saveweights.m`	save learned weights of the MLPs to a file
`loadweights.m`	load learned weights of the MLPs from a file
`intadjust.m`	find an optimally intensity-adjusted image, for the computation of quality measure Q_2 (see [4])
`kraskovmi.m`	estimate the mutual information, from samples of two jointly distributed random variables

b) Internal script files (intended for internal use by the toolbox)

`arrinit.m`	initialize various arrays for training
`back.m`	backpropagate
`compdist.m`	compute the estimated CDF of the components, for plotting
`compgrad.m`	compute the gradient of the cost function
`costderiv.m`	compute the cost function and its derivatives
`doepoch.m`	perform a training epoch
`forward.m`	propagate forward (compute network's outputs)
`normalize.m`	normalize the `obs` array to [0,1]
`plotdata.m`	produce plots during training
`reportresults.m`	report training progress
`testcost.m`	test whether the cost function has improved, and act accordingly
`wadapt.m`	adapt weights during training

The main variables and arrays that the user may need to access are the following:

`ntrain`	number of training patterns
`nepochs`	number of training epochs to be performed
`outputonlyepochs`	number of training epochs during which only the ψ MLPs are adapted

`linearonlyepochs`	number of training epochs during which the separation MLP (**F)** is constrained to be linear
`symepochs`	number of training epochs during which the separation MLP (**F)** is constrained to be symmetrical
`obs`	mixture observations (images); `obs(:,:,1)` should contain the first mixture image and `obs(:,:,2)` the second mixture image
`trpattern`	training patterns (array of size `2 * ntrain`, with one pattern per column); this array is created from the mixture observations (`obs`) by `selectdata.m`
`separimg`	separated images, produced by `processimg.m` (array, with the same size and structure as `obs`)

The following sections describe in more detail the basic operation of the routines.

4. Installation

To install the toolbox simply unzip the contents of the zip file to a directory of your choice.

5. Basic operation of the routines

5.1. Initializing and training the separation system

To use the separation routines you should first clear the MATLAB environment:

```
clear all
```

Then you need to place the mixture images in the `obs` array. This will normally be done by a pair of commands like

```
obs = imread('mixture1.bmp');
obs(:,:,2) = imread('mixture2.bmp');
```

Next, change to the directory where you've placed the separation routines:

```
cd <directory where you've placed the routines>
```

Then, to initialize parameters, give one of the commands

```
netparlin
```

or

```
netparnonlin
```

depending on whether you want to perform linear or nonlinear separation. You may also use your own initialization scripts, with your own parameter values, by editing `netparlin.m` or `netparnonlin.m` (see Section 6.1).

Among other things, the initialization scripts set the following parameters (see Section 3 for an explanation of their function):

```
ntrain
nepochs
outputonlyepochs
linearonlyepochs
symepochs
```

Next you need to randomly select pairs of mixture pixels to form the training set. For this, give the command

```
selectdata
```

The randomly selected patterns will be placed in the `trpattern` array. The number of training patterns that are selected is given by `ntrain`.

Then it is necessary to initialize the MLPs. This is performed by giving the command

```
netinit
```

The network can now be trained, by giving the command

```
train
```

Training will begin. For every training epoch, one line of output will be produced, showing the value of the cost function, the improvement in the cost function relative to the lowest cost found in previous epochs, and the number of the epoch. The cost function that is used is *minus* the objective function that is mentioned in [1-4], i.e. it is given by

$$-\frac{1}{\texttt{ntrain}} \sum_{k=1}^{\texttt{ntrain}} \log\det\left|\mathbf{J}^k\right|$$

Since the cost function is minus the objective function, it is minimized, and not maximized. This is due to historical reasons, since this package evolved from a supervised MLP training package in which the cost function was minimized.

In the first epoch, a very large improvement will be reported. This large value is a result of the internal cost function control procedure, and is not a sign of malfunction. From time to time, negative improvements may be reported, possibly during several consecutive epochs. This is taken care of by the internal cost function control procedure, and also is not a sign of malfunction. In such cases, positive improvements should reappear after a few epochs.

Every five epochs, a plot will be made. You should resize the plotting window so that the two leftmost plots are square. The leftmost plot shows a scatter plot of the current estimated components, designated as y_1 and y_2 in [1-4] and in Fig. 1 (y_1: horizontal axis; y_2: vertical axis). The center plot shows a scatter plot of the outputs of the ψ MLPs (these outputs are referred to as z_1 and z_2 in [1-4] and in Fig. 1). If ICA is successful, the distribution in this scatter plot should become approximately uniform in a square. The rightmost plots show the ψ_i functions, which are the estimated Cumulative Distribution Functions (CDFs) of y_1 and y_2 (ψ_1: top; ψ_2: bottom). These CDFs are rescaled from the interval [0,1] to [-1,1].

After performing the number of training epochs specified in `nepochs`, training will stop. You can resume training for a further `nepochs`, if you wish, by giving the `train` command again.

You can also interrupt training at any time, by pressing Ctrl-C. The state of the network will be preserved, and can be used for resuming training or for processing data.[1]

5.2. Processing images with the trained system

To process the mixture images contained in `obs` after training the system, give the command

```
processimg
```

The separated images will be placed in `separimg(:,:,1)` and `separimg(:,:,2)`, respectively.

5.3 Auxiliary routines

saveweights and loadweights

These routines respectively save and load the MLPs' weights to/from a file, in '.mat' format. Before using them you should set the variable `filename` to the name of the file to be used, e.g.

```
filename = 'test2weights.mat'
```

Besides saving/loading the weights, these routines also save/load the variables `nhidden` and `nextra`, so that the MLPs' structures remain consistent.

kraskovmi

This routine computes the mutual information estimate $I^{(1)}$, described in [5]. Its calling format is

```
mi = kraskovmi(x1,x2,k);
```

where `x1` and `x2` are row vectors containing samples of the two random variables forming the joint distribution whose mutual information is to be estimated, and `k` is the nearest-neighbor order used in the estimation algorithm (see [5] for details). The recommended value of `k` is between 2 and 4.

The samples in `x1` and `x2` should correspond to one another, i.e. `x1(1,1)` to `x2(1,1)`, `x1(1,2)` to `x2(1,2)`, etc., so that they represent the joint distribution whose mutual information is to be estimated.

intadjust

This routine finds an image with optimally adjusted intensities, for computing the quality measure Q_2 (see [4]). The calling format is

```
adjustedimage = intadjust(sourceimage,refimage);
```

where `sourceimage` is the image to be adjusted (usually the result of a separation) and `refimage` is the image which serves as reference for intensity adjustment

---

[1] Since I changed to MATLAB 7, it occasionally crashes with a runtime error if I press Ctrl-C when a script is running. This seems to be a bug of MATLAB 7, and not of the processing routines.

(usually a source image). The output is the result of applying the optimal monotonic intensity transformation $f(\cdot)$ to `sourceimage` (see [4] for details). The arguments `sourceimage` and `refimage` should be of the same size.

`imalign`

This routine performs the image alignment as described in the paper. See the initial comments in the code for a description of inputs and outputs. Note that the routine does not perform the increase of resolution of the images, needed for an accurate alignment, nor the decrease in resolution after the alignment. You should perform that yourself before and after calling the routine.

6. Changing settings

There is a number of parameters and other settings that you can change, to suit the system to your needs.

6.1. Changing parameters

All parameters are set in the files `netparXXX.m`. You may change parameters by editing one of these files or by creating new ones based on them.

The parameters that you may, most probably, want to change are:

`nhidden`	number of units in the hidden layer of the separating MLP, for each component. The total number of hidden units is the double of `nhidden`, since there are two components
`nextra`	number of hidden units in each of the ψ MPLs.
`ntrain`	number of training patterns
`nepochs`	number of training epochs per training run
`outputonlyepochs`	number of training epochs during which only the ψ MLPs are adapted
`linearonlyepochs`	number of training epochs during which the separation MLP (**F)** is constrained to be linear
`symepochs`	number of training epochs during which the separation MLP (**F)** is constrained to be symmetrical

6.2. Modifying the output produced during training

The output generated during training can be changed by editing two files. The output printed after each epoch is created by the script `reportresults.m`, and the plots are generated by the script `plotdata.m`.

You may eliminate the printed and/or plotted output by replacing these files with blank ones. Further control of the output is possible by editing these files, but that is

beyond the scope of this manual, because it would involve a detailed explanation of the various internal variables used by the routines.

7. License

The routines supplied here are copyright of Luis B. Almeida. Free permission is given for their use, solely for nonprofit research purposes. All other uses are prohibited, except if a specific license is obtained from the copyright owner.

8. References

[1] L. B. Almeida, “MISEP – an ICA method for linear and nonlinear mixtures, based on mutual information”, in *Proc. 2002 Int. Joint Conf. on Neural Networks*, Honolulu, Hawaii, 2002,
http://www.lx.it.pt/~lbalmeida/papers/AlmeidaIJCNN02.pdf

[2] L. B. Almeida, “MISEP – linear and nonlinear ICA based on mutual information”, *Journal of Machine Learning Research*, Vol. 4, December 2003, pp. 1297-1318,
http://www.jmlr.org/papers/volume4/almeida03a/almeida03a.pdf

[3] L. B. Almeida, “Linear and Nonlinear ICA Based on Mutual Information – the MISEP Method”, *Signal Processing*, Vol. 84, No. 2, February 2004, pp. 231-245,
http://www.lx.it.pt/~lbalmeida/papers/AlmeidaSigProc03.pdf

[4] L. B. Almeida, "Separating a Real-Life Nonlinear Image Mixture", *Journal of Machine Learning Research*, vol. 6, pp. 1199–1232, 2005,
http://www.lx.it.pt/~lbalmeida/papers/AlmeidaJMLR05.pdf

[5] A. Kraskov, H. Stögbauer and P. Grassberger, "Estimating Mutual Information", *Physical Review E*, vol. 69, pp. 066138, 2004,
http://arxiv.org/pdf/cond-mat/0305641