

MI-Based Linear and Nonlinear ICA Toolbox

Manual

1. Introduction

This manual presents a MATLAB toolbox for linear and nonlinear Independent Component Analysis (ICA) and Blind Source Separation (BSS), based on the minimization of the Mutual Information (MI) of the extracted components. The manual covers the use of the toolbox. The description of the separation method can be found in references [1-3].

2. General description

The toolbox implements linear or nonlinear ICA, based on a training set of observations \mathbf{x} .¹ 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 hidden sigmoidal units, and thus performs a nonlinear operation.

During the optimization (training) of the separation network, auxiliary MLPs are used. See Fig. 1 for the structure of the network during training, and references [1-3] for the description of the training method.

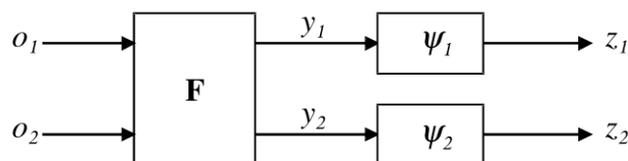


Fig. 1 – Structure of the network used by the toolbox.

3. Toolbox files and variables

The files contained in the toolbox are the following:

- a) Main script files (intended to be called by the user)
 - `netpar.m` set toolbox parameters
 - `generate.m` generate artificial mixture, for test purposes
 - `netinit.m` initialize the system for training
 - `train.m` train the separation system
 - `processdata.m` process test data with the trained system

¹ In this manual, bold lowercase characters denote vectors and bold uppercase letters denote matrices; Courier text denotes MATLAB commands, outputs, variables, etc.

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

<code>arrinit.m</code>	initialize various arrays for training
<code>back.m</code>	backpropagate
<code>compdist.m</code>	compute the estimated CPF of the components
<code>compgrad.m</code>	compute the gradient of the cost function
<code>costderiv.m</code>	compute the cost function and its derivatives
<code>doepoch.m</code>	perform a training epoch
<code>forward.m</code>	propagate forward (compute network's outputs)
<code>plotdata.m</code>	perform various plots during training
<code>reportresults.m</code>	report training progress
<code>testcost.m</code>	test whether the cost function has improved
<code>wadapt.m</code>	adapt weights during training

The main variables and arrays used by the toolbox, which the user may need to access, are the following:

<code>ninputs</code>	number of components of the observations
<code>ntrain</code>	number of training patterns
<code>nepochs</code>	number of desired training epochs
<code>trpattern</code>	training patterns (array of size <code>ninputs</code> x <code>ntrain</code> , one pattern per column)
<code>separpattern</code>	result of separation of <code>trpattern</code> (array, same size and structure as <code>trpattern</code>)
<code>mixeddata</code>	test data to be "separated" by the trained network (array, <code>ninputs</code> x <any number of columns>, one pattern per column).
<code>separadata</code>	result of separation of <code>mixeddata</code> (array, same size and structure as <code>mixeddata</code>)

The following sections describe in more detail the basic operation of the toolbox and functions of the main commands.

4. Installation

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

5. Basic toolbox operation

5.1. Initializing and training the system

To use the toolbox, you should first give, within MATLAB, the commands

```
cd <directory where the toolbox files reside>
clear all
```

and then the command

```
netpar
```

to initialize the toolbox parameters. Among other things, this command sets the following values (see below and Section 3 for an explanation of their function):

```
ninputs = 2
ntrain  = 1000
nepochs = 500
```

Next you need to place the training data in the `trpattern` array. This array should contain one pattern per column, with as many columns as training patterns. You should also set `ninputs` to the number of components of each training pattern (number of observed mixture components) and `ntrain` to the number of training patterns. You'll probably want to edit the file `netpar.m` to reflect the values that you normally use.

If you just want to test the toolbox with automatically generated data, instead of setting `trpattern` as described above, simply give the command

```
generate
```

This will place in `trpattern` a nonlinear mixture of two supergaussian random sources, with `ntrain` patterns.

Once `trpattern`, `ninputs` and `ntrain` are set, 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 will be output, 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. Note that the cost function is *minus* the cost function indicated in [1-3] and is minimized, instead of being maximized. This is due to historical reasons, because this toolbox 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, both in the initial training and if you've resumed training (see below for how to resume training). 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 y_1 and y_2 (y_1 – horizontal axis, y_2 – vertical axis). The center plot shows a scatter plot of the outputs of the ψ_i MLPs (i.e. of z_1 and z_2). If ICA is successful, the distribution in this scatter plot should become approximately uniform within a square. The rightmost plots show the functions ψ_1 and ψ_2 , i.e. the estimated Cumulative Probability Functions (CPFs) of y_1 and y_2 (ψ_1 – top, ψ_2 – bottom).

After 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 test data.²

5.2. Processing data with the trained system

To process data after training the system, first place the data to be processed in an array called `mixeddata` (the data should be formatted one pattern per column, with a number of columns equal to the number of patterns to be processed). Then give the command `processdata`. The results of processing will be placed in the array `separadata`, which will be of the same size as `mixeddata`.

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 file `netpar.m`. You may change parameters by editing this file and then giving the command

```
netpar
```

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

<code>ninputs</code>	number of sources and of observations per pattern.
<code>noutputs</code>	should be equal to <code>ninputs</code> .
<code>nhidden</code>	number of units in the hidden layer of the separating MLP, per component.
<code>nextra</code>	number of hidden units in each of the ψ MLPs.
<code>ntrain</code>	number of training patterns.
<code>nepochs</code>	number of epochs performed by each <code>train</code> command.
<code>ndisp</code>	number of epochs per output plot.
<code>initialepochs</code>	number of initial epochs, during which only the ψ MLPs are adapted (the F MLP remains fixed during these epochs).

² In my current version of MATLAB, for the training to resume after being interrupted by Ctrl-C, it is necessary to first close the figure that was created by the program. The program will open a new figure when it is resumed. This problem seems to result from a bug in MATLAB.

6.2. Modifying the output

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 functions performed by the various internal variables of the toolbox. Some ways to change the output plots are described in the comments of `plotdata.m`.

7. License

This toolbox is copyright of Luis B. Almeida. Free permission is given for its use, solely for nonprofit research purposes. Any other use is prohibited, unless 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>