A neural network toolbox for Octave
User's Guide
Version: 0.1.3

Michel D. Schmid

July 23, 2007

Contents

Chapter 1

Introduction

1.1 Compatibility to Matlab's ™Neural Network Toolbox

The compatibility is one of the strongest targets during developing this toolbox. If I have to develope an incompatibility e.g. in naming the functions, it will be descriped in this documentation. Even though it should be clear that I can't make a one to one copy. First, the m-files are copyrighted and second, octave doesn't yet support the object oriented-programming techonology.

If you find a bug, any not described incompatibility or have some suggestions, please write me at michaelschmid@users.sourceforge.net. This will help improving this toolbox.

1.2 Version numbers

The first number describes the major release. Version number V1.0 will be the first toolbox release which should have the same functions like the Matlab R14 SP3 neural network Toolbox.

The second number defines the finished functions. So to start, only the MLPs will realised and so this will be the number V0.1.0.

The third number defines the status of the actual development and function. V0.1.0 means a first release with MLP. Actually it works only with Levenberg-Marquardt algorithm and Mean-Square-Error as performance function.

1.3 Known incompatibilities

1.3.1 Function names

minmax

minmax is in this toolbox called *min_max*. This is because Octave already has a function whichs name is *minmax*. This is a c file and the functions *min* and *max* are therein realized.

Chapter 2

Neural Network Toolbox for Octave

This chapter describes all functions available in the neural network toolbox of Octave.
Eventhough it will be as compatible as possible to the one of MATLAB(TM).

2.1 Available Functions

2.1.1 min_max

min_ max get the minimal and maximal values of an training input matrix. So the dimension of this matrix must be an RxN matrix where R is the number of input neurons and N depends on the number of training sets.

Syntax:

mMinMaxElements = min_max(RxN);

2.1.2 newff

newff is the short form for ***new feed forward network***. This command creates a feed-forward backpropagation network structure.

Syntax:

net = newff(Rx2,[S1 S2 ... SN],{TF1 TF2 ... TFN},BTF,BLF,PF)

Description:

Rx2: R x 2 matrix of min and max values for R input elements
Si: Size of ith layer, for N layers
TFi: Transfer function of ith layer, default = "tansig"
BTF: Backpropagation network training function, default = "trainlm"
BLF: Backpropagation weight/bias learning function, NOT USED, is only for MATLAB(TM) compatibility
PF: Performance function, default = "mse"

Examples:

net = newff(Rx2,[2 1])
net = newff(Rx2,[2 1],{"tansig","purelin"});
net = newff(Rx2,[2 1],{"tansig","purelin"},"trainlm");

net = newff(Rx2,[2 1],{"tansig","purelin"},"trainlm","notUsed","mse");

Comments:
In this version, you can have as much output neurons as you want. The same with the number of hidden layers. This means you can have one input layer, unrestricted number of hidden layers and one output layer. That's it.

2.1.3 saveMLPStruct

This is an additional function which doesn't exist in the neural network toolbox of MathWorks (TM). To see the network structure, you can use this command and save the complete structure to a file. Open this file and you have the same view like you would open the *network type* of MATLAB(TM).

Syntax:

saveMLPStruct(net,"initNetwork.txt");

2.1.4 sim

Syntax:

simout = sim(....);

2.1.5 train

Syntax:

net = train(....);

2.2 Transfer functions

2.2.1 logsig

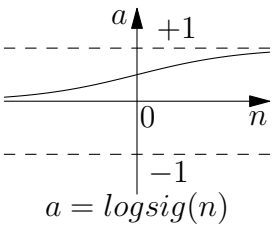

Figure 2.1: Log-Sigmoid transfer function

Figure 2.2: Log-Sigmoid transfer function logo

2.2.2 purelin

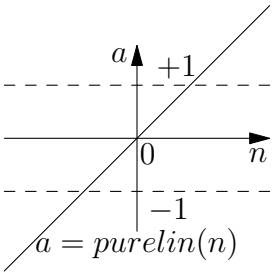

Figure 2.3: Linear transfer function

Figure 2.4: Linear transfer function logo

2.2.3 tansig

I solved all of my real life problems with this transfer function if a non-linear function was used. In [4] page 2-6 the tansig is defined as in equation (2.1). A look on the MathWorks homepage with the keyword tansig will show that tansig is programed as in equation (2.2). IS THIS A COPYRIGHT PROBLEM?

$$a = \frac{e^n - e^{-n}}{e^n + e^{-n}} \tag{2.1}$$

$$a = \frac{2}{(1 + e^{-2*n}) - 1} \tag{2.2}$$

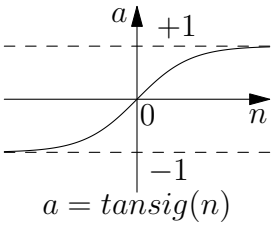

Figure 2.5: Tansig transfer function

Figure 2.6: Tansig transfer function logo

Chapter 3

Examples

3.1 Example 1

You can find this example in the *tests/MLP* directory of each release or from the subversion repository. I will do (more or less) a line by line walkthrough, so after this should be everything clear. I assume that you have some experience with multilayer perceptrons.

3.1.1 Introduction

Our problem can be solved with a monotonically increasing or decreasing surface. An input vector **p** (with 9 values) should be mapped onto one output value. Because we know that it can be solved with a monotonically increasing or decreasing surface, we can choose a 9-1-1 multi-layer perceptron (short: MLP). This means an MLP with 9 input neurons, only 1 hidden neuron and with 1 output neuron.

3.1.2 Code m-file

```
00001 ## Copyright (C) 2006 Michel D. Schmid  <michaelschmid@users.sourceforge.net>
00002 ##
00003 ##
00004 ## This program is free software; you can redistribute it and/or modify it
00005 ## under the terms of the GNU General Public License as published by
00006 ## the Free Software Foundation; either version 2, or (at your option)
00007 ## any later version.
00008 ##
00009 ## This program is distributed in the hope that it will be useful, but
00010 ## WITHOUT ANY WARRANTY; without even the implied warranty of
00011 ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00012 ## General Public License for more details.
00013 ##
00014 ## You should have received a copy of the GNU General Public License
00015 ## along with this program; see the file COPYING.  If not, write to the Free
00016 ## Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
00017 ## 02110-1301, USA.
00018
00019 ## Author:  Michel D. Schmid
00020
00021
00022 ## load data
00023 mData = load("mData.txt","mData");
00024 mData = mData.mData;
```

```
00025 [nRows, nColumns] = size(mData);
00026     # this file contains 13 columns.
00027     # The first 12 columns are the inputs
00028     # the last column is the output,
00029     # remove column 4, 8 and 12!
00030     # 89 rows.
00031
00032
00033 mOutput = mData(:,end);
00034 mInput = mData(:,1:end-1);
00035 mInput(:,[4 8 12]) = []; # delete column 4, 8 and 12
00036
00037 ## now prepare data
00038 mInput = mInput';
00039 mOutput = mOutput';
00040
00041 # now split the data matrix in 3 pieces, train data, test data and validate data
00042 # the proportion should be about 1/2 train, 1/3 test and 1/6 validate data
00043 # in this neural network we have 12 weights, for each weight at least 3 train sets..
00044 # (that's a rule of thumb like 1/2, 1/3 and 1/6)
00045 # 1/2 of 89 = 44.5; let's take 44 for training
00046 nTrainSets = floor(nRows/2);
00047 # now the rest of the sets are again 100%
00048 # ==> 2/3 for test sets and 1/3 for validate sets
00049 nTestSets = (nRows-nTrainSets)/3*2;
00050 nValiSets = nRows-nTrainSets-nTestSets;
00051
00052 mValiInput = mInput(:,1:nValiSets);
00053 mValliOutput = mOutput(:,1:nValiSets);
00054 mInput(:,1:nValiSets) = [];
00055 mOutput(:,1:nValiSets) = [];
00056 mTestInput = mInput(:,1:nTestSets);
00057 mTestOutput = mOutput(:,1:nTestSets);
00058 mInput(:,1:nTestSets) = [];
00059 mOutput(:,1:nTestSets) = [];
00060 mTrainInput = mInput(:,1:nTrainSets);
00061 mTrainOutput = mOutput(:,1:nTrainSets);
00062
00063 [mTrainInputN,cMeanInput,cStdInput] = prestd(mTrainInput);# standardize inputs
00064
00065 ## comments:  there is no reason to standardize the outputs because we have only
00066 # one output ...
00067
00068 # define the max and min inputs for each row
00069 mMinMaxElements = min_max(mTrainInputN); # input matrix with (R x 2)...
00070
00071 ## define network
00072 nHiddenNeurons = 1;
00073 nOutputNeurons = 1;
00074
00075 MLPnet = newff(mMinMaxElements,[nHiddenNeurons nOutputNeurons],\
00076         {"tansig","purelin"},"trainlm","","mse");
00077 ## for test purpose, define weights by hand
00078 MLPnet.IW{1,1}(:)  = 1.5;
```

```
00079 MLPnet.LW{2,1}(:)  = 0.5;
00080 MLPnet.b{1,1}(:)  = 1.5;
00081 MLPnet.b{2,1}(:)  = 0.5;
00082
00083 saveMLPStruct(MLPnet,"MLP3test.txt");
00084
00085 ## define validation data new, for matlab compatibility
00086 VV.P = mValiInput;
00087 VV.T = mValliOutput;
00088
00089 ## standardize also the validate data
00090 VV.P = trastd(VV.P,cMeanInput,cStdInput);
00091
00092 [net] = train(MLPnet,mTrainInputN,mTrainOutput,[],[],VV);
00093
00094 # make preparations for net test and test MLPnet
00095 #  standardise input & output test data
00096 [mTestInputN] = trastd(mTestInput,cMeanInput,cStdInput);
00097
00098 [simOut] = sim(net,mTestInputN);
00099 simOut
```

3.1.3 Walkthrough

Till line number 0023 there is realy nothing interesting.
On line 0023 & 0024 data will be loaded. This data matrix contains 13 columns. Column 4, 8 and 12 won't be used (this is because the datas are of a real world problem). Column 13 contains the target values. So on the lines 0049 till 0051 this will be splittet into the corresponding peaces. A short repetition about the datas: Each line is a data set with 9 input values and one target value. On line 0038 and 0039 the datas are transposed. So we have now in each column one data set.

Now let's split the data matrix again in 3 pieces. The biggest part is for training the network. The second part for testing the trained network to be sure it's still possible to generalize with the net. And the third part, and the smallest one, for validate during training. This splitting happens on the lines 0041 till 0061.

Line 0063 is the first special command from this toolbox. This command will be used to pre-standardize the input datas. Do it ever! Non linear transfer functions will squash the whole input range to an small second range e.g. the transfer function *tansig* will squash the datas between -1 and +1.

On line 0069 the next toolbox command will be used. This command min_max creates a $Rx2$ matrix of the complete input matrix. Don't ask me for what MATLAB(TM) this is using. I couldn't figure out it. One part is the number of input neurons, but for this, the range would not be needed. Who cares ;-)

Now it's time to create a structure which holds the informations about the neural network. The command **newff** can do it for us. See the complete line and actually, please use it only on this way, each other try will fail! This means, you can change the number of input neurons, the number of hidden neurons and the number of output neurons of course. But don't change the transfer functions, the train algorithm or the performance function. And don't change the number of hidden layers.

saveMLPStruct on line 0083 is a command which doesn't exist in MATLAB(TM). This will save the structure with the same informations you can see in MATLAB(TM) if you try to open the net-type.

The validation part on line 0086 & 0087 is important. The naming convention is for MATLAB(TM) compatibility. For validate, you have to define a structure with the name **VV**. Inside this structure you have to define actually **VV.P** & **VV.T** for validate inputs and validate targets. Bye the way, you have to pre-standardize them like the training input matrix. Use for this the command **trastd** like on line 0090.

train is the next toolbox command and of course one of the most important. Please also use this command like on line 0092. Nothing else will work.

The second last step is to standardize again datas. This time the test datas. See line 0096 for this and the last step. Simulate the network. This can be done with the command **sim**. This will be a critical part if someone else will write a toolbox with this command name!

I hope this short walkthrough will help for first steps. In next time, I will try to improve this documentation and of course, the toolbox commands. But time is realy rare.

Bibliography

[1] John W. Eaton

GNU Octave Manual, Edition 3, PDF-Version, February 1997

[2] The MathWorks, Inc.

MATLAB Help, MATLAB Version 7.1 (R14SP3), Neural Network Toolbox Version 4.0.6 (R14SP3)

[3] Christopher M. Bishop

Neural Networks for Pattern Recognition, OXFORD University Press, Great Clarendon Streed, Oxford OX2 6DP, ISBN 0-19-853864-2, 2002

[4] Martin T. Hagen, Howard B. Demuth, Mark H. Beale

NEURAL NETWORK DESIGN, PWS Publishing Company, 20 Park Plaza, Boston, MA 02116-4324, ISBN 053494332-2, 1996