A neural network toolbox for Octave
User's Guide
Version: 0.0.2-39

Michel D. Schmid

February 17, 2007

Contents

Chapter 1

Introduction

1.1 Compatibility to Matlab's ^{TM}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 deveoplemnt and function. V0.0.2-39 means no release with MLP, actually, everything under construction... ;-D.

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 in Octave.

Eventhough it will be as compatible as possible to the one of MATLAB(TM). STRONG LIMITS ARE STILL used in this toolbox. It's first version 0.0.2-39 ...

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 structure which holds informations about the neural network structure.

Syntax:

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

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(....);

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 multilayer 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
00002 ##
00003 ## This file is part of octnnettb.
00004 ##
00005 ## octnnettb is free software; you can redistribute it and/or modify it
00006 ## under the terms of the GNU General Public License as published by
00007 ## the Free Software Foundation; either version 2, or (at your option)
00008 ## any later version.
00009 ##
00010 ## octnnettb is distributed in the hope that it will be useful, but
00011 ## WITHOUT ANY WARRANTY; without even the implied warranty of
00012 ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00013 ## General Public License for more details.
00014 ##
00015 ## You should have received a copy of the GNU General Public License
00016 ## along with octnnettb; see the file COPYING.  If not, write to the Free
00017 ## Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
00018 ## 02110-1301, USA.
00019
00020 ## This is a test to train a 9-1-1 MLP (was a real project).
00021
00022 ## author:  mds
00023 ## e-mail:  michaelschmid@users.sourceforge.net
00024
```

```
00025
00026 ## for debug purpose only
00027 global DEBUG = 0;
00028 ## comments to DEBUG:
00029 # 0 or not exist means NO DEBUG
00030 # 1 means, DEBUG, write to command window
00031 # 2 means, DEBUG, write to files...
00032
00033
00034 ## load data
00035
00036 mData = load("mData.txt","mData");
00037 mData = mData.mData;
00038 [nRows, nColumns] = size(mData);
00039 # this file contains 13 columns.  The first 12 columns are the inputs
00040 # the last column is the output
00041 # remove column 4, 8 and 12
00042 # 89 rows
00043
00044 ## first permute the whole matrix in row wise
00045 ## this won't be used right now for debug and test purpose
00046 # order = randperm(nRows);
00047 # mData(order,:)  = mData;
00048
00049 mOutput = mData(:,end);
00050 mInput = mData(:,1:end-1);
00051 mInput(:,[4 8 12]) = []; # delete column 4, 8 and 12
00052
00053 ## now prepare data
00054 mInput = mInput';
00055 mOutput = mOutput';
00056
00057 # now split the data matrix in 3 pieces, train data, test data and validate
00058 data
00059 # the proportion should be about 1/2 train, 1/3 test and 1/6 validate data
00060 # in this neural network we have 12 weights, for each weight at least 3
00061 train sets..
00062 # (that's a rule of thumb like 1/2, 1/3 and 1/6)
00063 # 1/2 of 89 = 44.5; let's take 44 for training
00064 nTrainSets = floor(nRows/2);
00065 # now the rest of the sets are again 100%
00066 # ==> 2/3 for test sets and 1/3 for validate sets
00067 nTestSets = (nRows-nTrainSets)/3*2;
00068 nValiSets = nRows-nTrainSets-nTestSets;
00069
00070 mValiInput = mInput(:,1:nValiSets);
00071 mValliOutput = mOutput(:,1:nValiSets);
00072 mInput(:,1:nValiSets) = [];
00073 mOutput(:,1:nValiSets) = [];
00074 mTestInput = mInput(:,1:nTestSets);
00075 mTestOutput = mOutput(:,1:nTestSets);
00076 mInput(:,1:nTestSets) = [];
00077 mOutput(:,1:nTestSets) = [];
00078 mTrainInput = mInput(:,1:nTrainSets);
```

```
00079 mTrainOutput = mOutput(:,1:nTrainSets);
00080
00081 [mTrainInputN,cMeanInput,cStdInput] = prestd(mTrainInput);# standardize
00082                                       inputs
00083
00084 ## comments:  there is no reason to standardize the outputs because we have
00085 only
00086 # one output ...
00087
00088 # define the max and min inputs for each row
00089 mMinMaxElements = min_max(mTrainInputN); % input matrix with (R x 2)...
00090
00091 ## define network
00092 nHiddenNeurons = 1;
00093 nOutputNeurons = 1;
00094
00095 MLPnet = newff(mMinMaxElements,[nHiddenNeurons nOutputNeurons],{"tansig",
00096          "purelin"},"trainlm","learngdm","mse");
00097 ## for test purpose, define weights by hand
00098 MLPnet.IW{1,1}(:)  = 1.5;
00099 MLPnet.LW{2,1}(:)  = 0.5;
00100 MLPnet.b{1,1}(:)  = 1.5;
00101 MLPnet.b{2,1}(:)  = 0.5;
00102
00103 saveMLPStruct(MLPnet,"MLP3test.txt");
00104 #disp("network structure saved, press any key to continue...")
00105 #pause
00106
00107 ## define validation data new, for matlab compatibility
00108 VV.P = mValiInput;
00109 VV.T = mValliOutput;
00110
00111 ## standardize also the validate data
00112 VV.P = trastd(VV.P,cMeanInput,cStdInput);
00113
00114 #[net,tr,out,E] = train(MLPnet,mInputN,mOutput,[],[],VV);
00115 [net] = train(MLPnet,mTrainInputN,mTrainOutput,[],[],VV);
00116 # saveMLPStruct(net,"MLP3testNachTraining.txt");
00117 # disp("network structure saved, press any key to continue...")
00118 # pause
00119
00120 # % make preparations for net test and test MLPnet
00121 #     % standardise input & output test data
00122 [mTestInputN] = trastd(mTestInput,cMeanInput,cStdInput);
00123
00124 [simOut] = sim(net,mTestInputN);%,Pi,Ai,mTestOutput);
00125 simOut
```

3.1.3 Walkthrough

Till line number 0036 there is realy nothing interesting.
On line 0036 & 0037 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 0054 and 0055 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 0067 till 0079.

Line 0081 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 0089 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.

Lines 0098 till 0101 aren't needed normaly. This is only for testing purpose. It's the way you can change the network weights by hand. *newff* will initialize them randomly.

saveMLPStruct on line 0103 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 0108 & 0109 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 0112.

train is the next toolbox command and of course one of the most important. Please also use this command like on line 0115. Nothing else will work.

The second last step is to standardize again datas. This time the test datas. See line 0122 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