MLP Neural Networks Using Octave NN Package

Nung Kion, Lee

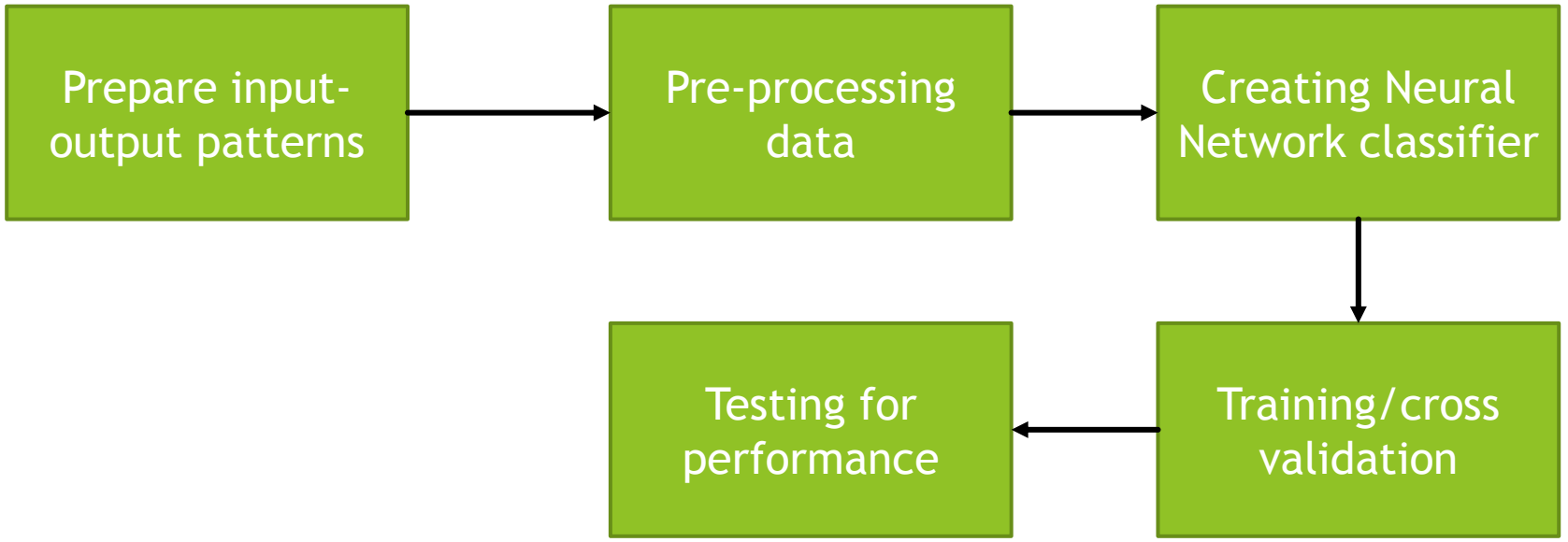
Faculty of Cognitive Sciences and Human Development

UNIVERSITI MALAYSIA SARAWAK

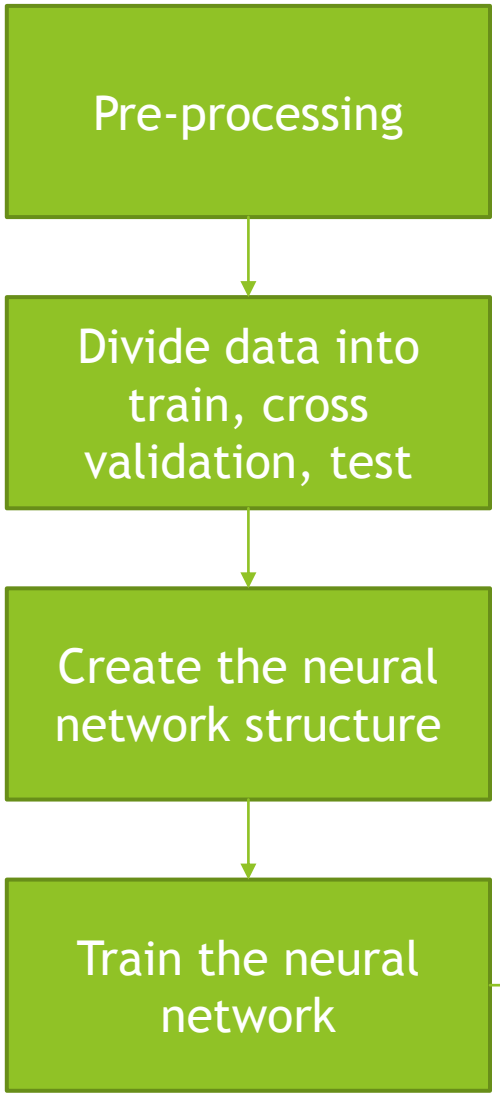
Introduction

- Octave provides a simple neural network package to construct the Multilayer Perceptron Neural Networks which is compatible (partially) with Matlab.
- Only feedforward backprogation neural network is implemented.
- Only one training algorithm is available (the Levenberg-Marquardt)

Steps of Using Neural Networks as Classifier



Steps in creating NN in Octave



- standardize inputs and outputs (if necessary).
- E.g. normalization
- prestd function

Use subset function or own coding

newff function

train function

Predict class of inputs of test dataset

sim function

Take Note!!!

- In Octave, the training, test, or validation data must be prepared in columns. That is the rows are features and the columns are training input patterns.
- The target patterns/desired output of corresponding to input patterns must be arranged in columns as well.

Training data input-output vectors.

ID	Sep_len	Sep_wid	Pet_len	Pet_wid	Type 1	Type 2	Type 3
1	5.1	3.5	1.4	0.2	1	0	0
2	4.9	3	1.4	0.2	1	0	0
3	4.7	3.2	1.3	0.2	1	0	0
4	4.6	3.1	1.5	0.2	1	0	0
5	5	3.6	1.4	0.2	1	0	0
6	6.3	3.3	4.7	1.6	0	1	0
7	4.9	2.4	3.3	1	0	1	0
8	6.6	2.9	4.6	1.3	0	1	0
9	5.2	2.7	3.9	1.4	0	1	0
10	5	2	3.5	1	0	1	0
11	5.9	3	4.2	1.5	0	1	0
12	6	2.2	4	1	0	1	0
13	6.4	2.8	5.6	2.1	0	0	1
14	7.2	3	5.8	1.6	0	0	1
15	7.4	2.8	6.1	1.9	0	0	1
16	7.9	3.8	6.4	2	0	0	1
17	6.4	2.8	5.6	2.2	0	0	1
18	6.3	2.8	5.1	1.5	0	0	1
19	6.1	2.6	5.6	1.4	0	0	1
20	7.7	3	6.1	2.3	0	0	1

Input-output patterns format for Octave NN package

Input				Target		
I1	I2	I3	I4	T1	T2	T3
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	1	0	0
4.7	3.2	1.3	0.2	1	0	0
4.6	3.1	1.5	0.2	1	0	0
5	3.6	1.4	0.2	1	0	0
6.3	3.3	4.7	1.6	1	0	0
4.9	2.4	3.3	1	1	0	0
6.6	2.9	4.6	1.3	1	0	0
5.2	2.7	3.9	1.4	1	0	0
5	2	3.5	1	1	0	0

Examples of Iris data

Input

Target

I1	5.1	4.9	4.7	4.6	5.0	5.4	4.6	5.0	4.4	4.9
I2	3.5	3.0	3.2	3.1	3.6	3.9	3.4	3.4	2.9	3.1
I3	1.4	1.4	1.3	1.5	1.4	1.7	1.4	1.5	1.4	1.5
I4	0.2	0.2	0.2	0.2	0.2	0.4	0.3	0.2	0.2	0.1
T1	1	1	1	1	1	1	1	1	1	1
T2	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0

Input output pairs for Octave NN

Exercise 1

- Download the iris.data from Morpheus and extract the input and output patterns into inputdata and outputdata respectively.
- Prepare the inputdata and outputdata in the format suitable for Octave NN input.

Min_Max Function

- Pulangkan nilai minimum dan maximum setiap baris dalam matrix untuk kegunaan fungsi newff.

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);

Description:

RxN: R x N matrix of min and max values for R input elements with N columns

Example:

$$\begin{bmatrix} 1 & 11 \\ 0 & 21 \end{bmatrix} = min\_max \begin{bmatrix} 3 & 1 & 3 & 5 & 11 \\ 12 & 0 & 21 & 8 & 6 \end{bmatrix} \quad (2.1)$$

Exercise 2

- Use the min_max function to determine the min and max value of each feature value of the iris dataset you have loaded into Octave workspace in Exercise 1.

- Record your result.

subset function

- Use to divide dataset into train, test and cross-validation.
- Recall that cross-validation data is used during training to achieve generalization.
- Reference: http://octave.sourceforge.net/nnet/function/subset.html

- Syntax:

  [mTrain, mTest, mVali] = subset (mData,nTargets,iOpti,fTest,fVali)

  mData – the complete training data with input and output values. E.g. in iris data, the mData is a 7 x 150 matrix (4 inputs and 3 outputs).

  nTargets – the number of outputs (e.g. 3 in iris data).

  iOpti – randomize the columns of data (permute). 0- No; 1 or 2 – Yes (see ref)

  fTest – percentage of data use for testing (default is 1/3 of data)

  fVali – percentage of data use for cross-validation (default is 1/6 of data).

Exercise 3

Note that: the value of fTest + fVali < 1. If fTest + fVali = 1, then no data for training. It is suggested to use the default.

E.g.: [mTrain, mTest, mVal] = subset(mData,1);

returns three subsets mTrain – 0.5, mTest – 1/3 and mVal – 1/6 of the mData without random shuffling. mData has one output value.

Load the iris.data into mData variable. Create train, test, validation subsets with 0.6, 0.2, 0.1 percentage. Verify your subsets using the size command on the produced subsets.

Standardization of inputs

- Read the following articles to understand why standardization of input feature values.
  1. http://stackoverflow.com/questions/4674623/why-do-we-have-to-normalize-the-input-for-an-artificial-neural-network
  2. http://stats.stackexchange.com/questions/7757/data-normalization-and-standardization-in-neural-networks
  3. ftp://ftp.sas.com/pub/neural/FAQ2.html#A_std
- Standardization converts feature values to have zero mean and standard deviation is 1.
- We can standardize both input feature values and/or target values depending on the data used.
- The prestd is available by Octave for standardization which syntax can be found at

http://dali.feld.cvut.cz/ucebna/matlab/toolbox/nnet/prestd.html?cmdname=prestd

Exercise 4

- Standardize the input values of iris data you have loaded.
- Check the result for correctness.

Creating MLP neural networks

The MLP NN implemented by Octave is very limited. It only support the Levenberg-Marquardt (LM) backpropagation training algorithm, not the gradient descent method discussed in the class.

See http://nopr.niscair.res.in/bitstream/123456789/8460/1/IJEMS%2012(5)%20434-442.pdf

for LM training method.

Syntax for creating MLP NN

net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

Description

PR - R x 2 matrix of min and max values for R input elements.

Si - Size of ith layer, for Nl layers.

TFi - Transfer function of ith layer, default = 'tansig'.

BTF - Backprop network training function, default = 'trainlm'.

BLF - Backprop weight/bias learning function, default = 'learngdm'.

PF - Performance function, default = 'mse'. Means square error.

and returns an N layer feed-forward backprop network.

The transfer functions TFi can be any differentiable transfer function such as tansig, logsig, or purelin.

The net is a struct data type (e.g. struct in c++).

Newff example

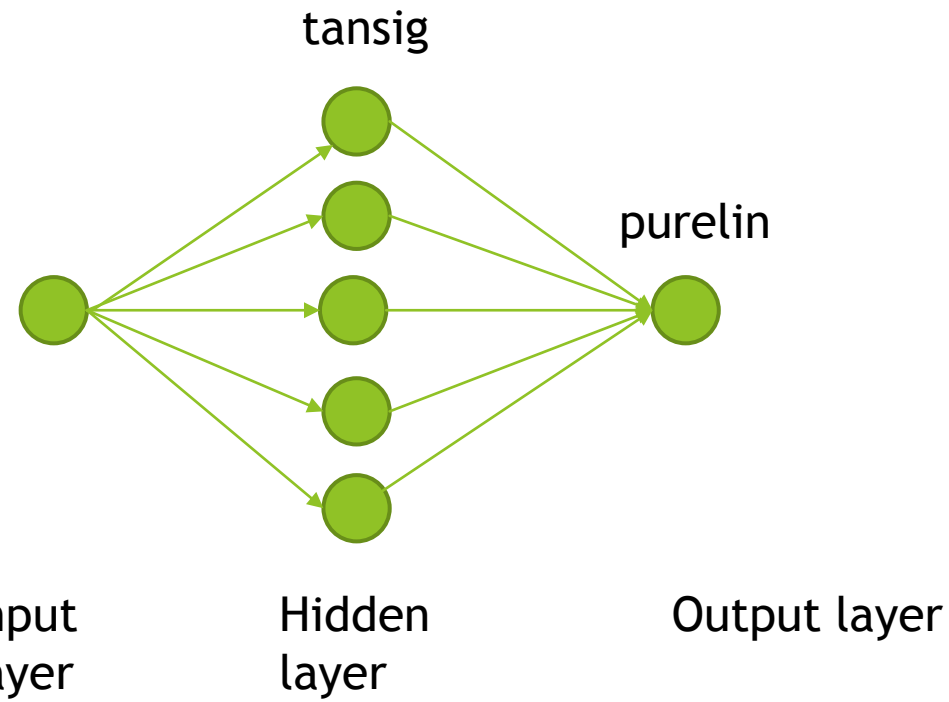
- Here is a problem consisting of inputs P and targets T that we would like to solve with a neural-network.

```
P = [0 1 2 3 4 5 6 7 8 9 10];  #one input with eleven examples
T = [0 1 2 3 4 3 2 1 2 3 4];  #one output
```

- Here a two-layer feed-forward network is created (one hidden layer, one output layer). The network's input ranges from [0 to 10]. The first layer has five tansig neurons, the second layer has one purelin neuron. The trainlm network training function is to be used.

```
MLPnet = newff([0 10],[5 1],{'tansig' 'purelin'});  or
MLPnet = newff([0 10],[5 1],{'tansig' 'purelin'},’trainlm’,’learngdm’,’mse’);
```

- MLPnet = newff([0 10],[5 1],{'tansig' 'purelin'});



Pay attention to which activation function to be used for output neurons. Different activation functions have different output range values. E.g. tansig is -1 to 1 and logsig is 0 to 1.

Training the neural network

[net,tr,out,E] = train(MLPnet,mInputN,mOutput,[],[],VV);

left side arguments:

net: the trained network of the net structure MLPnet

right side arguments:

MLPnet : the untrained network, created with newff

mInputN: normalized input matrix

mOutput: output matrix (normalized or not)

[] : unused parameter

[] : unused parameter

VV : validation data as struct. VV has two members:

VV.P and VV.T. VV.P is the validation input data. VV.T is the validation target output.

Training neural network (continue)

Example:

- Suppose using Iris data.

[Traindata, Testdata, Validation] = subset(Inputdata, 3,1, 0.4, 0.1);

VV.P = Validation(1:4, :); #first four rows are input features

VV.T = Validation(5:7, :); #5th to 8 rows are target output values

trainsubset = Traindata(1:4, :);

traintarget = Traindata(5:7, :);

trainednet = train(MLPNet, trainsubset, traintarget, [], [], VV);

Training will stop once one of the stopping criteria is met (see next slides).

Training neural network (continue)

- The training parameters are stored in the neural network trainParam struct.
- Default values for trainParam are:

```
net.trainParam.epochs = 100;   #number of epochs
net.trainParam.goal = 0;   #minimum means square error to achieve
net.trainParam.max_fail = 5;
net.trainParam.min_grad = 1.0000e-010; #gradient of the mse graph
net.trainParam.show = 50;  #show mse every 50 epochs
net.trainParam.time = Inf;  #maximum training time in seconds
net.show  = 50; #how frequent the mse is displayed in training progress graph?
```

max_fail: Validation vectors are used to stop training early if the network performance on the validation vectors fails to improve or remains the same for max_fail epochs in a row.

Training neural networks (continue)

- Training stops when any of these conditions occurs:
  - The maximum number of epochs (repetitions) is reached.
  - The maximum amount of time is exceeded.
  - Performance is minimized to the goal.
  - The performance gradient falls below min_grad.
  - Validation performance has increased more than max_fail times since the last time it decreased (when using validation).

Training Neural Networks (continue)

You should change the trainParam before using the train function to be effective.

E.g.:

MLPNet.trainParam.epochs = 500; #change maximum epochs to 500

MLPNet.trainParam.goal = 0.01; #stop training if mse is <= 0.01

trainednet = train(MLPNet, trainsubset, traintarget, [], [], VV);

Test trained neural networks

- To test the trained neural network use the sim function
- Syntax:

***netoutput* = sim (*net*, *mlnput*)**

net : trained neural network using train function

mlnput: standardized test input values

Exercise 5

- Download the mlpnn.m file from Morpheus and run the program in Octave.
- Please also download the input data file mData.txt.
- The input data has 13 columns: 12 inputs and 1 output.
- Put the mData.txt and mlpnn.m in the same folder.
- Read through the codes carefully and modify the training parameters.

1. Change the neural networks number of hidden neurons.
2. Run the network several times and observe the training progress graph.
3. Change the trainParam with different values. Make sure suitable values are used.

Exercise 6 – Using Iris.data

- Create a suitable neural network for the Iris.data file.
- Create suitable train, test, validation subsets.
- Train the neural networks using suitable parameters.
- Determine the accuracy of the neural network you have created.