Neural Networks
Course Project - Terrain Classification

Vicente Bosch Campos †
viboscam@posgrado.upv.es

March 11, 2011

Contents

List of Figures

1 Introduction

The application of Multilayer Perceptron and Support Vector Machines to uni - dimensionally multi-spectral representation of a 3x3 pixel matrix of an image section to recognize terrain type of the central pixel is considered in this course project.

The use of Pattern Recognition and Image Analysis is quite established for the task of satellite image classification. In this course project we consider the classification of terrain types.

1.1 Data

A Landsat MSS image consists of four digital images of the same zone using different spectral bands. Two of the bands are in the visible region (corresponding approximately to green and red regions of the visible spectrum) while the other two are in the (near) infra-red. Each pixel is represented as a 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is approximately 80m x 80m. Each image contains 2340 x 3380 such pixels.

The database used in the course project is a sub-area of a scene, consisting of 82 x 100 pixels. Each line of data corresponds to a 3x3 square neighborhood of pixels completely contained within the 82x100 sub-area. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3x3 neighborhood and a number indicating the classification label of the central pixel. As per the original database the number is a code for the following classes:

1. red soil
2. cotton crop
3. grey soil
4. damp grey soil

5. soil with vegetation stubble
6. mixture class (all types present)
7. very damp grey soil

Further preprocessing has been performed to the data provided to us. It has been normalized and as there are no records of the 6th type the classes label have been moved so that type 7 is represented by the label 6.

We have been provided with two data partitions:

- sat6c.tra.norm: Training set formed by 2000 records with the following composition:
  - Class 1: 471
  - Class 2: 220
  - Class 3: 440
  - Class 4: 197
  - Class 5: 213
  - Class 6: 459
- sat6c.public.tst.norm: Public test set containing of 1000 records with the following composition:
  - Class 1: 241
  - Class 2: 92
  - Class 3: 210
  - Class 4: 96
  - Class 5: 106
  - Class 6: 255

1.2 Scope

As part of the course project we will train classifiers for the landstat task described in the above sections. To be more precise we will use the following techniques:

- Multilayer Perceptron:
  - Back propagation with momentum
  - Quick propagation
  - Radial Networks

In order to evaluate the classifiers trained they will be sent to an external oracle containing a private data set in order to determine the test error with an independent data set.

2 Development

In this section we will describe the software framework developed in order to perform the experimentations for the task. The framework developed allows us to load, format and perform transformations with specific data sets and also launch classifier training with different classifiers and obtain mean test set classification error with cross validation.

2.1 Software Structure

The software is structured into the following classes:

DataPattern: The DataPattern class is defined in the *./lib/data_pattern.rb* file. It encapsulates a sample record of class containing the actual data (can be in numeric or string representations) and the class label. It allows us to perform the following operations:

- Order to records by label value.
- Iterate over the different values of the vector.
- Duplicate the record.

DataSet: The DataSet class can be found in the *./lib/data_set.rb* file. It represents a whole set of records of a task allowing us to perform the following tasks over them:

- Read the original data from a file and apply specific format to them to prepare them for the ML tool. e.g. Uniform the chain length for a Chromosome classification task.
- Write them to file using an specific format for a ML toolset.
- Partition the set into blocks ensuring correct representation and number of each class in the block.
- Combine different sets in an additive manner.
- Sort the set by class label.

Snns: The Snns class can be found in the *./lib/snns.rb* file. It is an object wrapper for the Batchman snns interface and allows us to perform the following tasks from the framework:

- The class allows as input an experiment object, it automatically generates an specific batchman script file and runs it.
- Permits us to programmatically access to the results of the .res files when reviewed with the analyze tool.

SnnsFileWriter: The SnnsFileWriter class can be found in the *./lib/snns.rb* file. The file writer is in charge of performing header and footer special formatting for the toolset chosen. In this case it allows us to call mkhead from the framework to prepare the pattern files.

SnnsPatternWriter: It can be found in *./lib/snns.rb.* The pattern writer in in charge of performing special formatting to the patterns required for correct usage on the desired toolset. In this case it ensures there is a space between each dimension value and ensures there is a line space between the vector and the class label.

Parameter: Coded in *./lib/parameter.rb.* The class allows us to define a parameter and a range of values of variation. The values can be a set of strings, if for example we want to vary between different neural network files or a numeric value, to study an specific range of the momentum parameter.

Experiment: Present in the file *./lib/experiment.rb.* It is an specific configuration, value set, of the parameters that will be executed on the chosen ML tool set and the value will be averaged over cross experimentation.

Study: contained in *./lib/study.rb.* It generates the set of experiments by performing combinations of each of the values for each parameter described and runs them writing the results and studies performed in specific files.

2.2 Library Options

With the defined software structure we can easily perform studies on an indicated task by just having to write the specific read and write formatters for the task. Next we will show a short example of the framework scripts used to train neural networks for the chromosome classification task:

Listing 1: chromosome.rb

```ruby
require 'ap'
require_relative '../lib/data_set'
require_relative '../lib/snns'
require_relative '../lib/study'
require_relative '../lib/chromosome_formatters'


data_folder="../test/chromosome_data/"
results_folder="./prac1"

#Create Data Set
set = MachineLearning::DataSet.new(data_folder+"cromos",1,:string)
#Create and set specific formatters for the task
set.file_writer= MachineLearning::SnnsFileWriter.new
set.line_writer = MachineLearning::SnnsPatternWriter.new
set.line_formatter = MachineLearning::ChromosomePatternFormatter.new
#Load data
set.read_data
#Set the params for the study
study_params = Array.new
#Learning technique
algo_param = MachineLearning::Parameter.new(:algo)
algo_param.fix_set(["BackpropMomentum"])
study_params << algo_param
#Original base model
ini_network=MachineLearning::Parameter.new(:in_model)
ini_network.fix_set([data_folder+"chr_1177_30_22.net"])
study_params << ini_network

#Learning parameter will range from 0.2 to 0.5 in 0.1 Increments
learning_param = MachineLearning::Parameter.new(:learning)

learning_param.numeric_range(0.2,0.5,0.1)

study_params << learning_param

#Momentum parameter will range from 0.1 to 0.5 in 0.1 increments
momentum_param = MachineLearning::Parameter.new(:momentum)

momentum_param.numeric_range(0.1,0.5,0.1)

study_params << momentum_param

#Creates the study indicating where to set the results, the
#learning technique SNNS, the parameters, num cross validation blocks
study = MachineLearning::Study.new
            (results_folder,:snns,study_params,set,5,"results.txt","study_list.txt")

#Generates the different combinations of all of the parameters indicated hence
#creating the different experiments
study.generate_experiments

#Runs the actual experiments
study.run
```

2.3 Software Output

The software generates the following output files that will help us out for the review of the classification training:

Study List: Is a detailed print out indicating the parameters used in the experiment as well as the pattern files and output files in case the experiment wants to be reviewed in detail:

```
--------------------------------------------------------------------------------------
1
{:algo=>"Quickprop", :in_model=>"../test/theory_data/trr_36_20_6.net", :learning=>0.15000000000000002, :max_growth=>0.
:id=>1, :test_pat=>"./QP30_20_6/20110209-000607_test.pat", :val_pat=>"./QP30_20_6/20110209-000607_val.pat",
:train_pat=>"./QP30_20_6/20110209-000607_train.pat", :out_model=>"./QP30_20_6/20110209-000607Quickprop.net",
:test_res=>"./QP30_20_6/20110209-000607test.res", :val_res=>"./QP30_20_6/20110209-000607val.res"}
--------------------------------------------------------------------------------------
```

Result File: For each of the parameter combinations a result is printed out containing the mean error, wrong and right classification % for the validation and test set over the cross training.

Log files: Contains an execution log of the Batchman execution with the training and validation MSE.

Bat files: Automatically generated Batchman script for the experiment.

Snns standard files: It also generates and saves the following SNNS files:

- Result files of the validation and test sets.
- Pattern files used in the training.
- Resulting neural network.

3 Experimental Results

3.1 Neural Networks

For the Neural Networks experiments a cross validation over 10 blocks will be performed for each experiment. The data will be composed of the training plus the public data set given.

Once selected the best result of an specific range/parameter/topology study the parameters are repeated in a 30 block cross validation sending one of the trained networks sent to the oracle to obtain private test data set classification result.

3.1.1 Back Propagation with Momentum

As it is dependent of the task all networks tested where configured with 36 input neurons and 6 output neurons connected in feed forward with out shortcuts. For the selection of the best hidden layer configuration, the following where tested:

- One hidden layer: 15,20,30,40,50,60,70,80,90 and 100 neurons.
- Two hidden layers: 10-10 neurons in each layer.

For each topology a study was launched varying the learning factor from 0.1 to 0.6 in increments of 0.05 and for each learning factor value the momentum was varied from 0.1 to 0.5 in 0.1 increments. Hence for each topology 55 different training set factors where tested, considering the 10 block cross validation and the 11 different topologies: 6050 networks where trained.

Next we will show a set of graphics to summarize the results obtained:

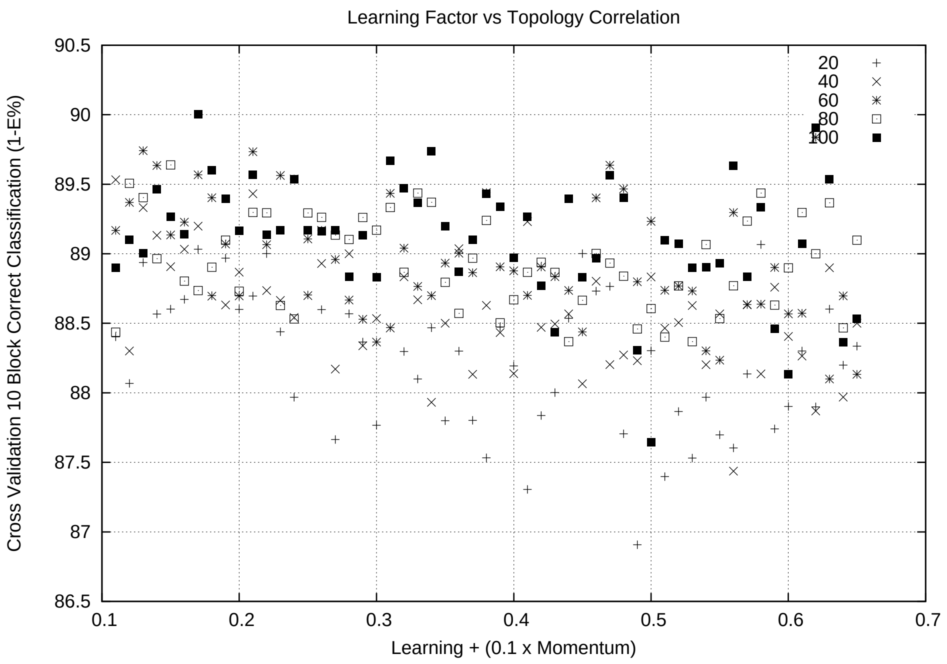

Figure 1: Plot shows the neural network classification right classifications (1 - E%) for the network topologies as a function of the training factors (Learning+(0.1xMomentum)) for the terrain classification task. Each plotted point is obtained by performing a 10 block cross validation over the given normalized database.

As we have seen in the previous graph there is no clear tendency of the effect of the learning factor on each topology but we can see that as we increase the number of neurons in the hidden layer we improve on the % of correctly classified patterns. This can be more clearly seen in the following graph:

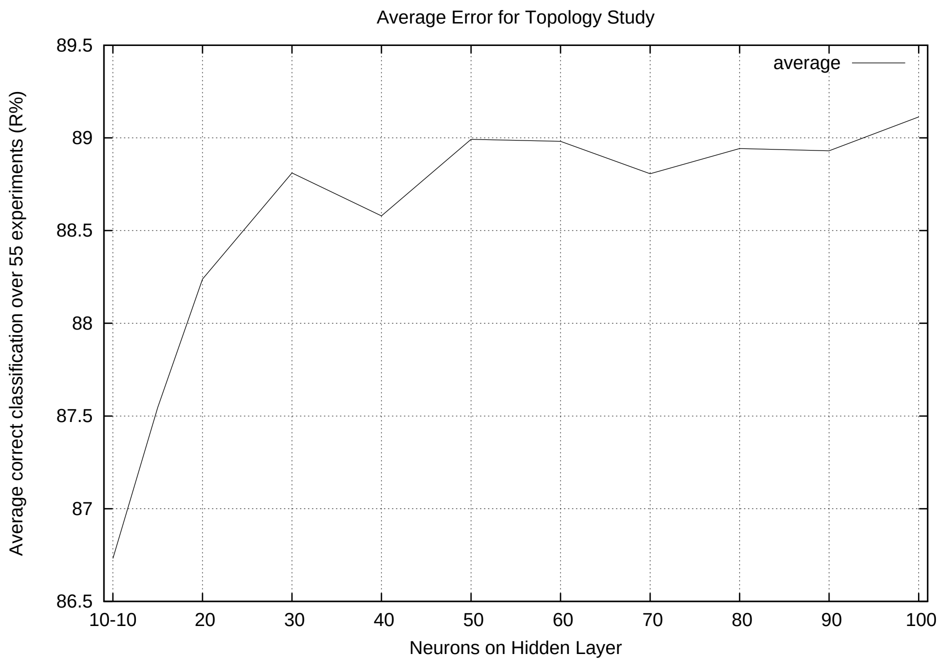

Figure 2: Plot shows the neural network classification right classifications (1 - E%) for the network topologies as a function of the number of neurons on the hidden layer for the terrain classification task. Each plotted point is obtained by performing an average of 55 experiments computed over 10 block cross validation over the given normalized database.

Hence the increase of neurons on the hidden layer seems to be beneficial for the task. In order to review the effect of the momentum the data was plotted giving more weight in the correlation the momentum factor:

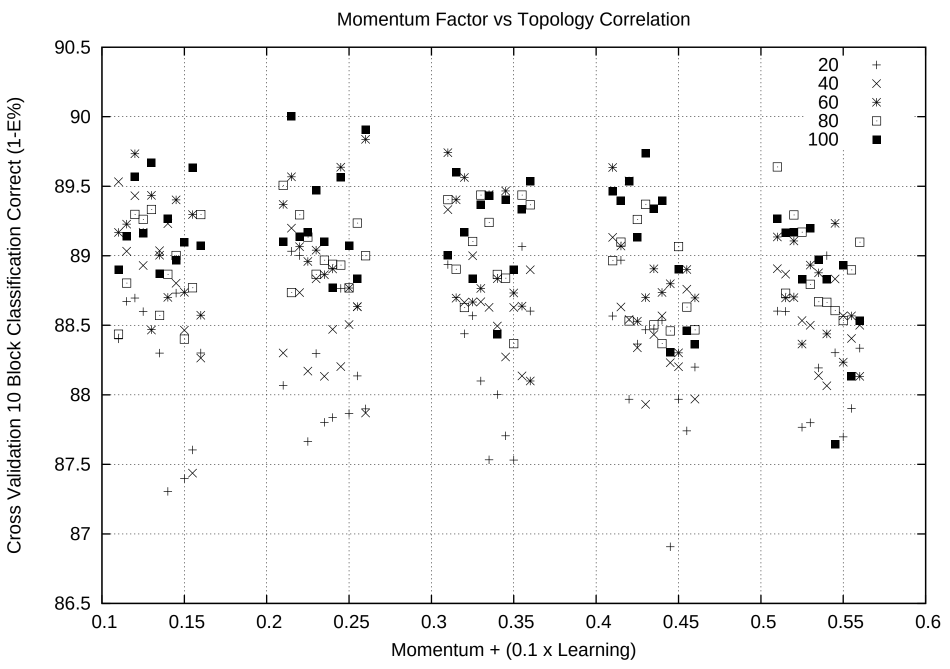

Figure 3: Plot shows the neural network classification right classifications (1 - E%) for the network topologies as a function of the training factors (Momentum+(0.Learning)) for the terrain classification task. Each plotted point is obtained by performing a 10 block cross validation over the given normalized database.

Again we can see no real tendency of how the momentum factor affects the different topologies.

3.1.2 Quick Propagation

The same set of topologies where trained with the Quick propagation training algorithm in order to review if it could improve the Back propagation classification results.

An initial study training all topologies was launched in order to review the best topology for this Algorithm. For all experiments the learning factors where set to:

- Learning: 0.3
- Max growth: 0.5

We can see in the previous graph that the one hidden layer neural network behave significantly better than the two hidden layer networks. Among the one hidden layer networks there is not a big difference in terms of correctly classified patterns but the neural network with 80 neurons seems to behave better; we will use this one for the in depth study of the learning parameters.

Once selected the best topology we performed a study of the quick propagation learning parameters by varying the learning parameter from 0.1 to 0.6 in 0.05 increments and for each learning factor value varying the maximum growth from 0.1 to 0.8 in 0.1 increments.

The best result obtained was with a learning factor of 0.1 and a training factor of 0.7.

Table 1: Test set comparison

Cross validation	Oracle result
16.564	15.50

3.1.3 Radial Networks

For Radial Networks a different set of networks had to be created as the radial basis learning requires to have a radial function as the activation function of the hidden layer.

In our study we created networks of: 20,40,60,80 & 100 neurons on the hidden layer. For all of them the gaussian activation function was selected.

Radial based training was significantly more complex to perform on the SNNS. A combination of three initialization functions had to be mixed up. Varying the default initialization or training parameter values from the ones explained in the SNNS manual caused the training algorithm to have numeric issues and not converge. Hence for this training algorithm only the topologies where varied.

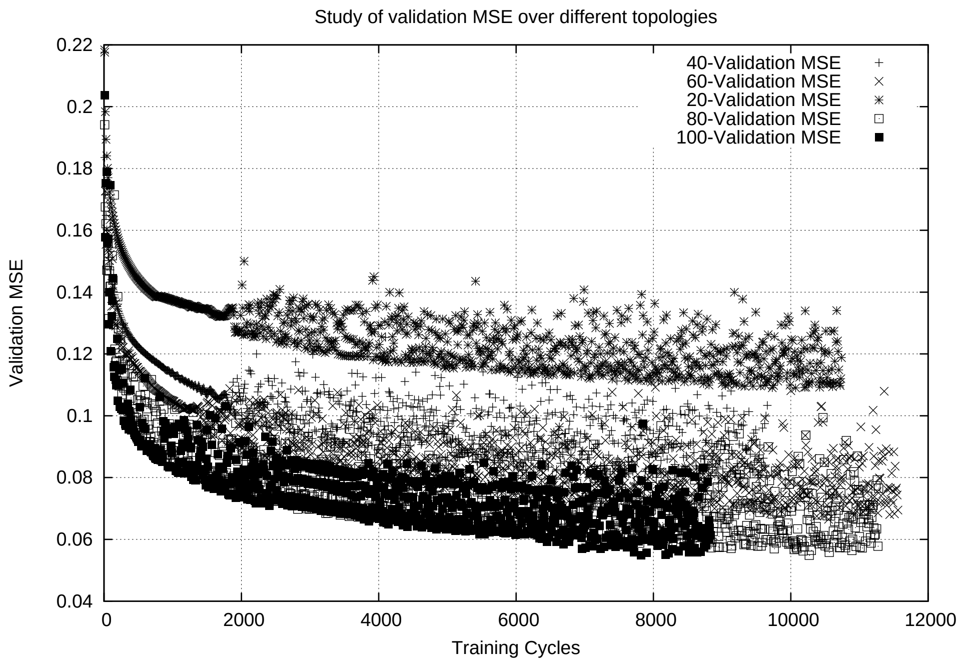

Figure 4: Plot shows the validation MSE for the radial network topologies as a function of the number of cycles trained for the terrain classification task.

The above plot is not very clarifying but we can conclude that as we increase the number of neurons we are getting a better generalization of the validation set but it must also be noted that the algorithm does not present a convergence and hence the results obtained might not be very repeatable.

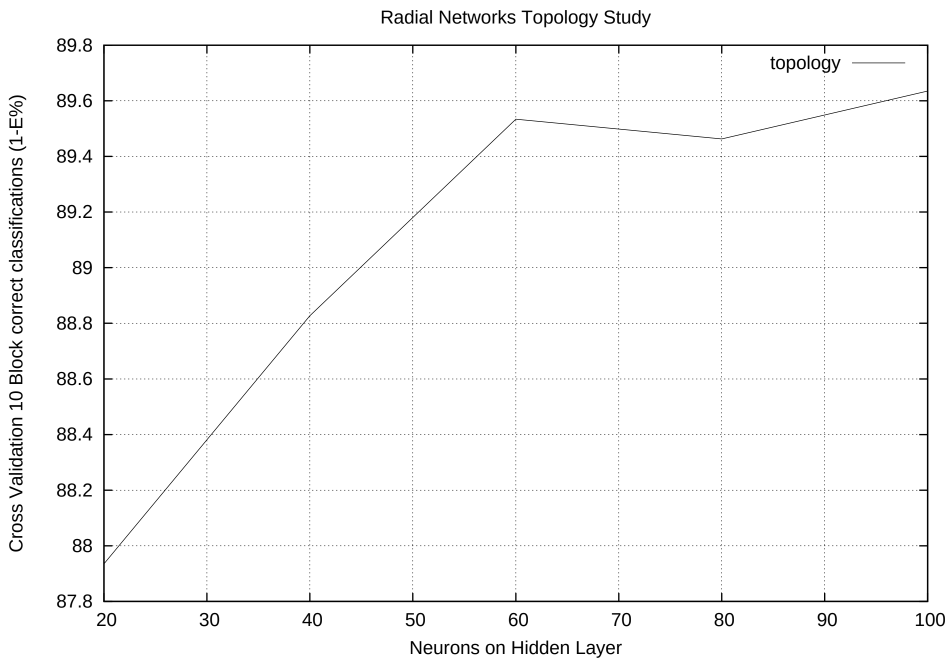

Figure 5: Plot shows the neural network classification right classifications (1 - E%) for the radial network topologies as a function of the number of neurons on the hidden layer for the terrain classification task. Each plotted point is obtained by performing a 10 block cross validation over the given normalized database.

As we have seen in the graph above the best neural network is of 100 neurons on the hidden layer but when sending the best trained network to the private test set oracle the best result was obtained with 40 neurons on the hidden layer.

4 Conclusions

As we have seen in the previous tasks the best results have been obtained for the Radial based learning. The best results for each algorithm in the private test set has been as follows:

Table 2: Private test set - best error results for training algorithms

Back-propagation	Quick-propagation	Radial Network
10.40	15.50	9.60

We have obtained the best results for Back-propagation and Radial based learning, being our radial network the network with least classification task in the board.

In future work it would be interesting to use the SVM software that has been integrated in the framework to study this task.