UNIVERSITAT ROVIRA I VIRGILI

Prediction with Back Propagation and Linear Regression

Neural and Evolutionary Computing
Practical Work

Celina Osorio Ochoa

Spring 2014

Content

Prediction with Back Propagation and Linear Regression

The objective of this practical work was to predict the power of the turbine of a hydro electrical plant, using back propagation and multiple linear regressions.

Introduction

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data (Lacey, 1998). It can provide a robust understanding of data and it has been applied extensively a predictive models for engineering and non-engineering domains (Parsons, Rizzo, & Buckwalter, 2004).

Development

The multi-linear regression was performed in Microsoft Excel Software using its functionality to make predictions; the back propagation algorithm was developed in Java Language. There are two main classes, Neural Network and Back Propagation, in the neural network class is all the structure of the network which is loaded from a file with the following format:

```
patterns 401                  //Number of patterns for training
patterns_prediction 49        //Number of patterns for validation
input_variables 4             //Number of input variables
values_to_predict 1           //Number of values to predict
layers 2                      //Number of layers in addition
                              //to the input and output layers
neurons_per_layer 9 3         //Neurons per added layers
epochs 2500                   //Number of epochs
repetitions 1                 //Number of repetitions
learning_rate 0.9             //Learning Rate
momentum 0.4                  //Momentum
# height over sea level, fall, net fall, flow, power
624.00 89.160 89.765 3.500 2512.85 //Input patterns and then
628.00 93.160 93.765 3.500 2583.79 //the output pattern
…
612.00 76.840 76.415 6.500 4450.58 //Last patterns in the
610.00 75.040 75.365 4.500 2893.53 //file are for validation
```

Neural Network Class

This class as it has been said contains the neural network structure, which is described by dynamic arrays of weights and thresholds. The array list represents the layer; the matrix inside each element of the array list (layer) represents the structure of the layer.

```
ArrayList <float[][]> weights;
ArrayList <float[]> thresholds;
```

Once the structure is loaded, the weights and thresholds are set to random values by using the Random class in methods *initializeThresholds()* and *initializeWeights()*. The main line of code is `weights.get(ℓ)[i][j] = (float)Math.random();` where ℓ represents the layer, *i* and *j* represent the weight which connects neuron *i* with neuron *j*.

Another thing made in this class is the scaling function, in order to improve the efficiency of the standard back propagation algorithm, a transformation of the input data to a range helps to escape of early trapping from premature saturation [ref]. This function, basically searches for the maximum and minimum value of each variable and then performs a transformation with these values in this way:

$$scaledPattern_{ij} = \frac{pattern_{ij} - minValue_j}{maxValue_j - minValue_j}$$

The same operation is made for the training patterns and also for the validation patterns, an example of the codification:

```
scaledPatterns[i][j]=(patterns[i][j]-Min[j])/(Max[j]-Min[j]);
```

Back Propagation Class

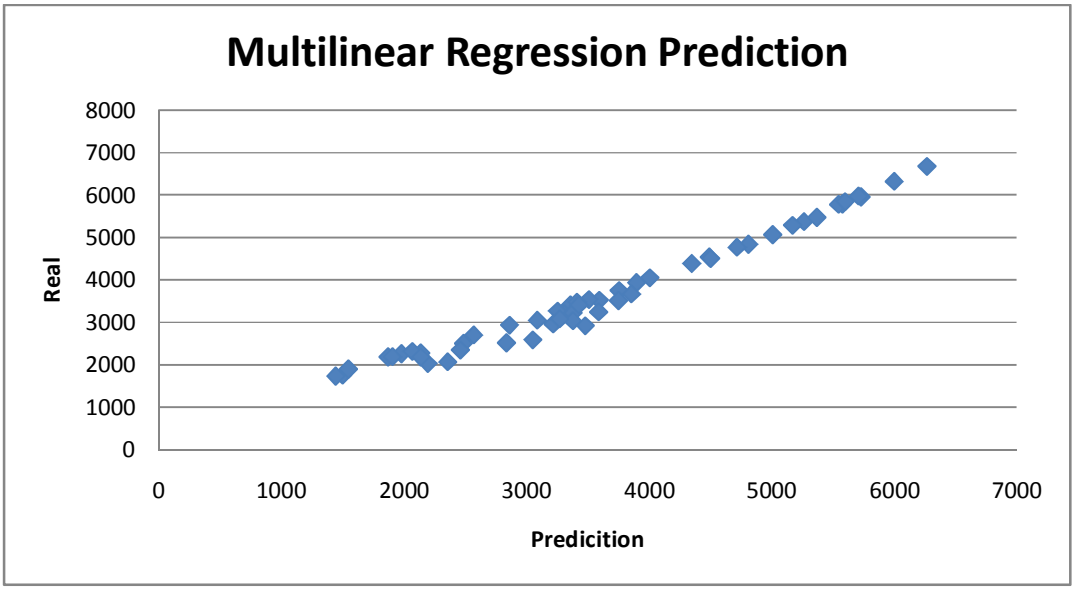
This is the class which contains the whole algorithm of back propagation; it has an instance of the Neural Network class with the file turbine.txt loaded. There is a method which calls the others and executes the algorithm in the proper order, this is **`computeAlgorithm()`**. Inside it, there are the following methods:

- **`backPropagation(float[] patternX, float[] patternZ)`**
  - This method receives an input pattern and an output pattern, and propagates the error.
- **`feedForward(float [] pattern)`**
  - This method receives an input pattern, in the first use of it the weights are set to random values.
- **`makePrediction()`**
  - Make some calls to **`feedForward(float [] pattern)`** with the data from validation set, and the weights obtained after the computation of the whole algorithm.
- **`updateWeights()`**
  - Based on the results obtained by the propagation of the error the weights are updated from random to sense values.
- **`computeAlgorithm()`**
  - It contains calls to the previous methods in a proper order.

Testing

Multi-linear Regression Testing

As it has been said, the multi-linear regression was calculated through the prediction function in Microsoft Excel Software. The results obtained are as expected and the total error of the prediction is **4.54%**, the Graphic 1 shows how the data behaves. The x-axis represents the predicted values and the y-axis represents the real data.



Graphic 1 Multi-linear Regression Prediction

Back Propagation Testing

Back propagation may have several input parameters with which it can achieve extremely different results; and also there is not a written guide on how to set the structure, how many neurons per layer or how many layers to use, of course there are recommendations but the best way to find the neural network structure that best suits our needs is to make a practical analysis.

It has been tried two structures of networks, both of them has the same input and output values, because in this practical work it is analyzed just one data set which contains the data of a hydro electrical plant. The first structure can be observed in the Image 1 and the second in the Image 2.

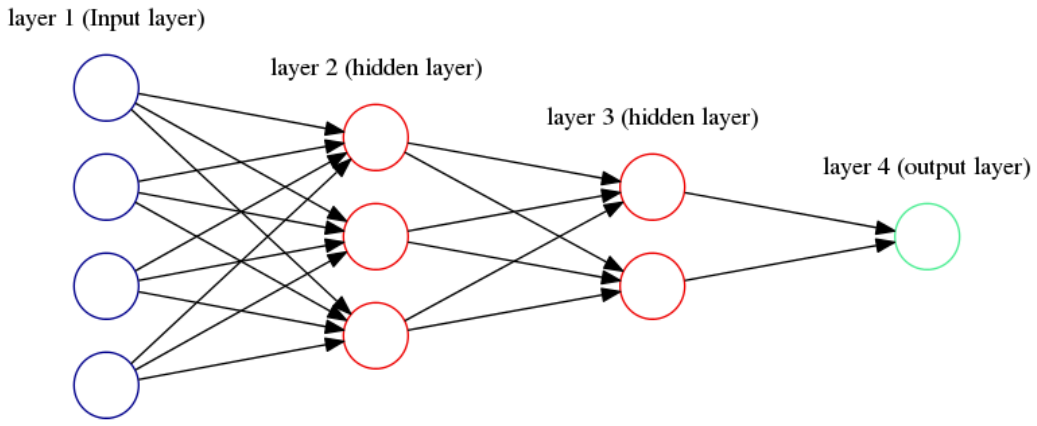


Image 1 First Tested Network Structure

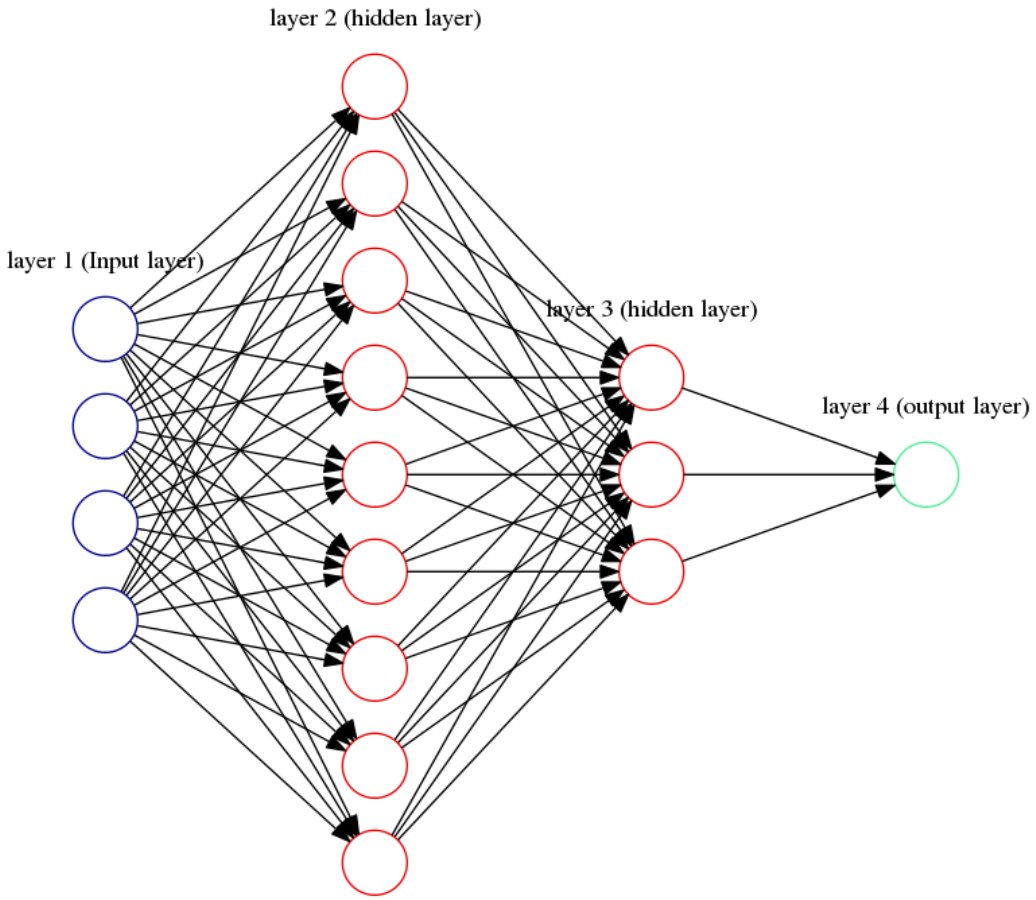
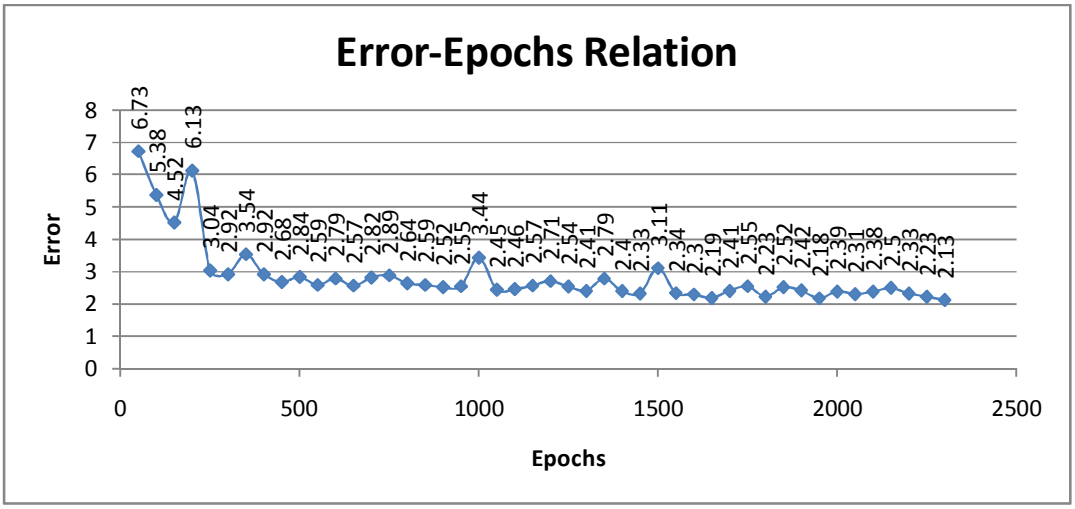


Image 2 Second Tested Network Structure

Test 1

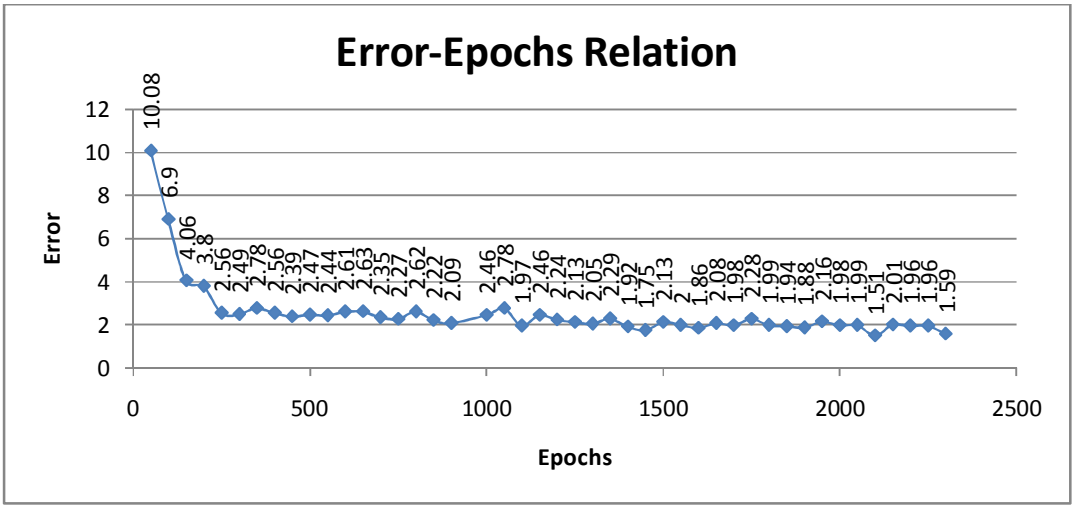
In the first test with both of them were performed 46 repetitions with different epochs each one, increasing in 50 units each time. The starting value of epoch was 50 and the last epoch value was 2300.

The results from the first network structure (FNS) are in the Graphic 2, this graphic does not show how the error behaves each epoch, and instead it shows the prediction error for each complete round of the algorithm with the number of epochs shown in the x-axis. As it can be seen, the error stabilizes around 2% and this is not the error expected (to analyze the three best results for FNS see Turbine-Results-Regression.xml – Sheet Best Prediction STR1, and all the predictions in Sheet Predictions STR1).

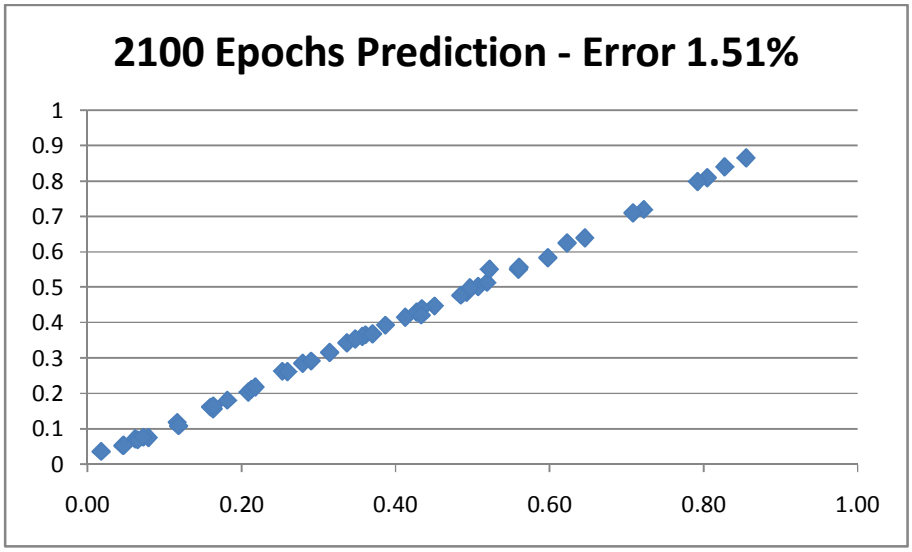


Graphic 2 Error Epochs Relation for First Network Structure

From the second network structure (SNS) were obtained better prediction results (Graphic 3), because in this case the error achieves values around 1.5%, as this networks seems to be better, were performed more tests. The best prediction error obtained in this test was the one in the Graphic 4 (to analyze the three best results for SNS see Turbine-Results-Regression.xml – Sheet Best Prediction STR2, and all the predictions in Sheet Predictions STR2).



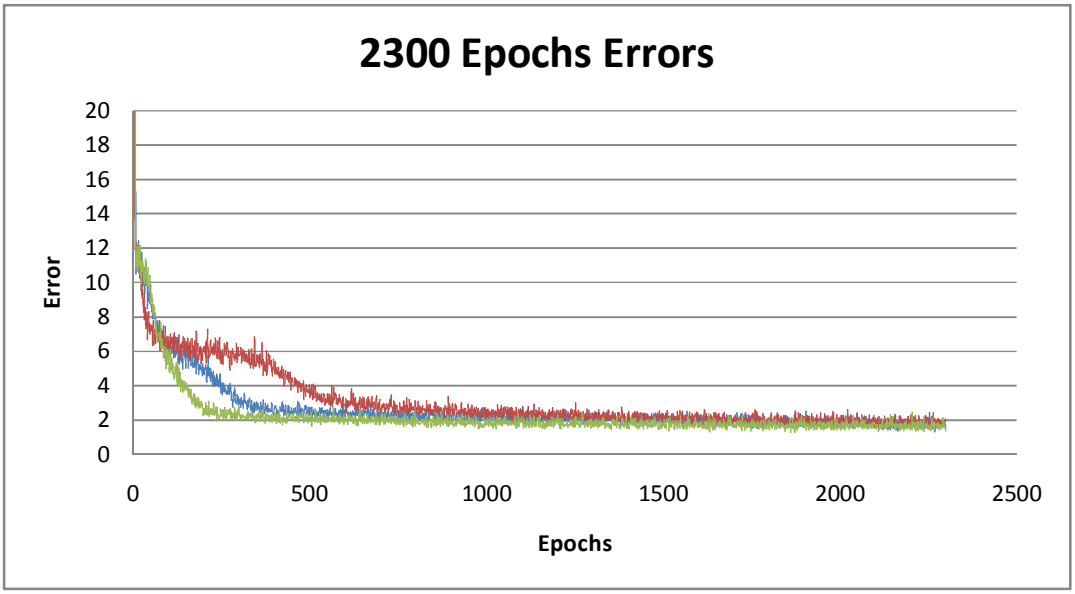
Graphic 3 Error Epochs Relation for Second Network Structure



Graphic 4 Prediction Error in 2100 epochs

Test 2

Once it has been chosen the SNS, it is important to observe if the network behaves the same way if the same value of epochs is used more than one execution, this time the Graphic 5 will show the error for each epoch, running three times with an epoch limit set in 2300. In the blue line execution, the best error value was 1.31% in the epoch 2270, the best error value for the red line was 1.54% in the epoch 2139 and the best error value for the green line was 1.29% in the epoch 1873, all of these errors for the training set. It can be observed that the best error values for training set were obtained in a different epoch value (See Turbine-Results-Regression.xlsx Sheet Fixed-Epochs). As a matter of solution to get the best error, the algorithm was modified to execute the whole algorithm until a desired error was achieved with the training set.



Graphic 5 2500 Epochs Errors

In the approach mentioned above, the condition to stop the algorithm is to get an error of 1.3% in the training set. For different executions it is achieved and with this approach, the testing error is always around 1.5% (See Turbine-Results-Regression.xlsx Sheet Best-Results-Non-Fixed-Epoch).

Conclusions

It was possible to compare results of multi-linear regression and back propagation in the same data set. As it has been stated in the report, the back propagation algorithm obtains better results than the regression.

The algorithm implemented in java gives efficient results for this data set; its development is relatively difficult if the formulas are not followed strictly.

It is important to observe graphically how the error behaves with different network structures and with different epochs, in this way better results can be achieved.

References

Gómez-Jiménez, S. (1994). *Multilayer Neural Networks: Learning Models and Applications.* Barcelona.

Lacey, M. (1998). Statistical Topics. New Haven, Conneticut, United States of America.

Parsons, T. D., Rizzo, A. A., & Buckwalter, J. G. (2004). Backpropagation and Regression: Comparative Utility for Neuropsychologists. *Journal of Clinical and Experimental Neuropsychology* , 95-104.