IRIS DATA ANALYSIS USING BACK PROPAGATION NEURAL NETWORKS

Sean Van Osselaer
Murdoch University, Western Australia

ABSTRACT

This project paper refers to experiments towards the classification of Iris plants with back propagation neural networks (BPNN). The problem concerns the identification of Iris plant species on the basis of plant attribute measurements. The paper outlines background information concerning the problem, making reference to statistics and value constraints identified in the course of the project. There is an outline of the algorithm of techniques used within the project, with descriptions of these techniques and their context. A discussion concerning the experimental setup is included, describing the implementation specifics of the project, preparatory actions, and the experimental results. The results generated by the networks constructed are presented, with the results being discussed and compared towards identification of the fittest architecture for the problem constrained by the data set. In conclusion, the fittest architecture is identified, and a justification concerning its selection offered.

Keywords : Iris, back propagation neural network, BPNN

INTRODUCTION

This project paper is related to the use of back propagation neural networks (BPNN) towards the identification of iris plants on the basis of the following measurements: sepal length, sepal width, petal length, and petal width. There is a comparison of the fitness of neural networks with input data normalised by column, row, sigmoid, and column constrained sigmoid normalisation. Also contained within the paper is an analysis of the performance results of back propagation neural networks with various numbers of hidden layer neurons, and differing number of cycles (epochs). The analysis of the performance of the neural networks is based on several criteria: incorrectly identified plants by training set (recall) and testing set (accuracy), specific error within incorrectly identified plants, overall data set error as tested, and class identification precision. The fittest network architecture identified used column normalisation, 40000 cycles, 1 hidden layer with 9 hidden layer neurons, a step width of 0.15, a maximum non-propagated error of 0.1, and a value of 1 for the number of update steps.

BACKGROUND

This project makes use of the well known Iris dataset, which refers to 3 classes of 50 instances each, where each class refers to a type of Iris plant. The first of the classes is linearly distinguishable from the remaining two, with the second two not being linearly separable from each other. The 150 instances, which are equally separated between the 3 classes, contain the following four numeric attributes: sepal length and width, petal length and width. A sepal is a division in the calyx, which is the protective layer of the flower in bud, and a petal is the divisions of the flower in bloom. The minimum values for the raw data contained in the data set are as follows (measurements in centimetres): sepal length (4.3), sepal width (2.0), petal length (1.0), and petal width (0.1). The maximum values for the raw data contained in the data set are as follows (measurements in centimetres): sepal length (7.9), sepal width (4.4), petal length (6.9), and petal width (2.5). In addition to these numeric attributes, each instance also includes an identifying class name, each of which is one of the following: Iris Setosa, Iris Versicolour, or Iris Virginica.

ALGORITHM OF TECHNIQUE USE

Data set construction

This project uses a two data set approach. The first of these sets is the training set, which is used for the actual training of the network, and for the determination of the networks recall ability. The second data set is the testing data set, which is not used in the training process, and is used to test the networks level of generalisation. This is done through the analysis of the accuracy achieved through testing against this set.

Normalisation

Normalisation of input data is used for ranging of values within an acceptable scope, and region. There are many mechanisms towards this end, four of which are analysed in this project. The four techniques used are column, row, sigmoid, and column constrained sigmoid normalisation.

Back propagation neural network (BPNN)

This project uses various back propagation neural networks (BPNN). BPNN use a supervised learning mechanism, and are constructed from simple computational units referred to as neurons. Neurons are connected by weighted links that allow for communication of values. When a neuron's signal is transmitted, it is transmitted along all of the links that diverge from it.

These signals terminate at the incoming connections with the other neurons in the network. The typical architecture for a BPNN is illustrated in Figure 1.

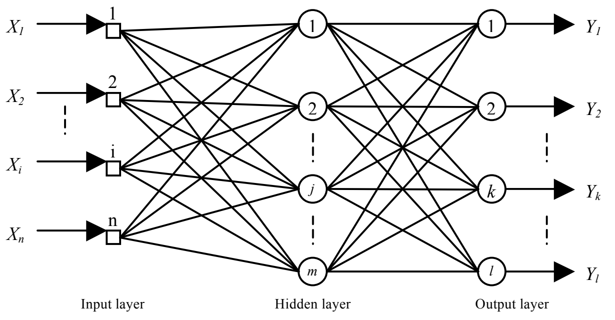

Figure 1. The architecture of a BPNN

In a BPNN, learning is initiated with the presentation of a training set to the network. The network generates an output pattern, and compares this output pattern with the expected result. If an error is observed, the weightings associated with the links between neurons are adjusted to reduce this error. The learning algorithm utilized has two stages. The first of these stages is when the training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output layer results are generated. Then, if the results differ from the expected, an error is calculated, and then transmitted backwards through the network to the input layer. It is during this process that the values for the weights are adjusted to reduce the error encountered. This mechanism is repeated until a terminating condition is achieved.

The algorithm towards the training of the network is as follows, and is adapted from "Artificial Intelligence – A Guide to intelligent Systems" by M. Negnevitsky [1]:

- Step 1: Initialisation

The weights and threshold values for the network are assigned values that are uniformly distributed over a small range eg: determined using the Haykin approach identified by Eq1.

$$\textbf{Eq1.}\quad \left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i} \right)$$

Where Fi is equivalent to the number of inputs to a neuron i. The initialisation of the weights mentioned is performed on each neuron within the network individually.

- Step 2: Activation

It is at this point that input values from a training set are presented to the networks input layer neurons, and the expected output values that are declared within the set qualified.

The networks hidden layer neurons then calculate their outputs. The calculation involved in this process is described by Eq2.

$$\textbf{Eq2.}\quad Y_j(P) = \text{sigmoid}\left[\sum_{i=1}^{n} X_i(P) \times W_{ij}(P) - \theta_j \right]$$

Where P is the P th training pattern in the training set, n is the number of inputs of the j th neuron in the hidden layer, X_i is an input into a neuron, and Y_j is the resulting output. The W_{ij} is the weighting, and the θ_j is the threshold value. The "sigmoid" in the above refers to the sigmoid function that is defined in Eq3.

$$\textbf{Eq3.}\quad Y_{\text{sigmoid}} = \frac{1}{1 + e^{-x}}$$

When it comes to the calculation of the eventual outputs of the output layer neurons, the calculation described by Eq4 is used towards the development of said outputs.

$$\textbf{Eq4.}\quad Y_k(P) = \text{sigmoid}\left[\sum_{j=1}^{m} X_{jk}(P) \times W_{jk}(P) - \theta_k \right]$$

Where P is the P th training pattern in the training set, m is the number of inputs of the k th neuron in the output layer, X_{jk} is an input into a neuron, and Y_k is the resulting output. The W_{jk} is the weighting, and the θ_k is the threshold value. The "sigmoid" in the above refers to the sigmoid function that is defined in Eq3.

- Step 3: Update weights

This is the step in which the weights of the BPNN are updated through the process of propagating backwards the errors related to the output neuron results. The calculation of the error gradient for the output neurons is calculated through Eq5.

$$\textbf{Eq5.}\quad \delta_k(P) = Y_k(P) \times [1 - Y_k(P)] \times e_k(P)$$

$$e_k(P) = Y_{d,k}(P) - Y_k(P)$$

Where: $$y_k(P) = \frac{1}{1 + \exp[-X_k(P)]}$$

δ_k is the error gradient for the k^{th} output neuron and P is the P^{th} pattern file.

With the weight adjustments for the output neurons being calculated by Eq6.

$$\textbf{Eq6.}\quad W_{jk}(P+1) = W_{jk}(P) + \Delta W_{jk}(P)$$

Where: $$\Delta W_{jk}(P) = \alpha \times Y_j(P) \times \delta_k(P)$$

The calculation of the error gradient for the hidden neurons is calculated through Eq7.

$$\textbf{Eq7.}\quad \delta_j(P) = Y_j(P) \times [1 - Y_j(P)] \times \sum_{k=1}^{l} \delta_k(P)\, W_{jk}(P)$$

With the weight adjustments for the output neurons being calculated by Eq8.

$$\textbf{Eq8.}\ W_{ij}(P+1) = W_{ij}(P) + \Delta W_{ij}(P)$$
$$\text{Where: } \Delta W_{ij}(P) = \alpha \times X_i(P) \times \delta_j(P)$$

- **Step 4: Iteration**

Increment the value of P by 1, and return to the second step of the process. This iterative process is conditional upon a terminating condition, if the terminating condition is realised, the training is complete, and the algorithm terminates.

DISCUSSION AND RESULTS OF EXPERIMENTAL SET UP

Data set construction

This projects initial activity was towards the separation of the consolidated data set, into a training set, and a testing set. It was decided to use a $^2/_3$ training $^1/_3$ testing approach, taking into consideration equal separation of representation between the classes within the sets. To this end, the training set was constructed of the first 2 of every 3 patterns contained in a class, with the remaining being allocated to the testing set. This resulted is a 102 pattern training set, and a 48 pattern testing set.

Normalisation

For the purpose of the project, four normalisation techniques were analysed, with the acceptable range for the normalised values being set to an approximation of 0.01(inclusive) to 0.99(exclusive).

- **Column normalisation**

In this normalisation technique, the largest value identified for an attribute is used as a divisor. For the project, a Java program was developed to generate a multiplier value to range the data to an approximation of, but not outside of the threshold range. An adding component was also generated by the program towards moving the ranged data into the appropriate region. The implementation was evaluated as shown in Eq9.

$$\textbf{Eq9.}\ xi = \left(\frac{xi}{\max(x) \times \text{multiplier}(x)} \right) + \text{adder}(x)$$

Where xi is an instance of x, max(x) is the largest identified value for x, multiplier(x) is the coefficient developed for ranging of the data, and adder(x) is the adding component developed for placing the ranged data into the appropriate region.

In the implementation of the column normalisation, the multipliers and adders shown in Table I were evaluated.

COLUMN NORMALISATION

	Multiplier	Adder
Sepal length	0.46999999999999953	-1.1479999999999844
Sepal width	0.5599999999999996	-0.8010000000000006
Petal length	0.8799999999999999	-0.1540000000000001
Petal width	0.99	-0.030000000000000002

Table I. The multipliers and adders evaluated for the column normalisation

Table II is a summary of the results of the column normalisation.

COLUMN NORMALISATION SUMMARY		
	Min Norm	Max Norm
Sepal length	0.0101	0.9797
Sepal width	0.0107	0.9847
Petal length	0.0107	0.9824
Petal width	0.0104	0.9801

Table II: summary of the results of the column normalisation

- **Row normalisation**

This technique involves using the combined total of the attributes across a pattern as a divisor. For the project, the multiplying coefficient and adding component where generated by a Java program for each row individually, as outlined in Eq10.

Eq10.

$$x = \left(\text{multiplier}(\text{row}(x)) \times \left(\frac{x}{\text{total}(\text{row}(x))} \right) \right) + \text{adder}(\text{row}(x))$$

Where x is a value in a row, multiplier (row(x)) is the coefficient developed for ranging the rows values, total(row(x)) is the total of the values in the row, and adder(row(x)) is the adding component developed for placing the data into the appropriate region.

In the implementation of the row normalisation, the multiplier and adder were generated row by row, with the highest value assigned across the data set being 0.9899, and the lowest 0.01.

- **Sigmoid normalisation**

This normalisation method involves using the sigmoid function, identified in Eq3, on the data values so as to "squash" the values into the appropriate range. The Java program developed for the project was used to generate a multiplying coefficient, and an adding component for the function. This was developed in an effort to spread the range of the values to better fill the range defined by the thresholds, and to move the values into the appropriate region. The sigmoid function was implemented as illustrated in Eq11.

$$\textbf{Eq11.}\ x = \text{multiplier} \times \left(\frac{1}{1+e^{-x}} \right) + \text{adder}$$

Where x is the value to be normalised, multiplier is the coefficient developed for ranging of the values, and adder is the adding component for placing the data into the appropriate region.

In the implementation of the sigmoid normalisation, the multiplier was determined to be 2.0599999999999996, and the adder to be -1.0709999999999928. Table III is a summary of the results.

SIGMOID NORMALISATION SUMMARY		
	Min Norm	Max Norm
Sepal length	0.9614	0.9882
Sepal width	0.7434	0.9640
Petal length	0.4350	0.9869
Petal width	0.0105	0.8327

Table III: summary of the results of the sigmoid normalisation

- ### Column constrained sigmoid normalisation

This normalisation technique is essentially a modification of the aforementioned sigmoid normalisation function implementation. However, in this implementation, the program developed coefficient and adding component are developed by attribute, rather than over the entire data set. The implementation is outlined in Eq12.

$$\textbf{Eq12.}\quad \mathrm{xi} = \mathrm{multiplier(x)} \times \left(\frac{1}{1+e^{-\mathrm{xi}}} \right) + \mathrm{adder(x)}$$

Where xi is an instance of x, multiplier(x) is the coefficient developed for ranging of the values of x, and adder(x) is the adding component developed for placing the ranged values of x into the appropriate region.

In the implementation of the column constrained sigmoid normalisation, the multipliers and adders shown in table IV were evaluated.

COLUMN CONSTRAINED SIGMOID NORMALISATION		
	Multiplier	Adder
Sepal length	75.21000000000157	-74.1929999999902
Sepal width	9.13999999999985	-8.040000000000983
Petal length	3.6499999999999657	-2.6579999999998183
Petal width	2.4499999999999913	-1.2759999999999703

Table IV. The multipliers and adders determined in the sigmoid normalisation

Table V is a summary of the results of the column constrained sigmoid normalisation.

COLUMN CONSTRAINED SIGMOID NORMALISATION SUMMARY		
	Min Norm	Max Norm
Sepal length	0.0102	0.9891
Sepal width	0.0105	0.9891
Petal length	0.0104	0.9883
Petal width	0.0102	0.9881

Table V: summary of the results of the column constrained sigmoid normalisation

Testing

In the analysis of the performance of the neural networks, several measuring criteria were utilized. The number of patterns that were incorrectly identified in the training set (recall), and testing set (accuracy) were evaluated. The class identification precision was used in conjunction with the recall/accuracy, and was computed by dividing the expected number of identified patterns for a class, by the value of patterns that were actually identified as being of that class. The overall set error as tested was also used, and was evaluated through Eq13.

$$\textbf{Eq13.}\quad \mathrm{Error} = \left\{ \sum_{i=1}^{m} \begin{array}{l} \mathit{Error} + X_i - T_i, X_i = 1 \\ \mathit{Error} + X_i + T_i, X_i \neq 1 \end{array} \right.$$

Where m is the number of patterns in a data set, X_i is the expected result for an output neuron i, and T_i is the actual output from that output neuron.

The specific error within incorrectly identified patterns was also used as a gauge for network performance, with the measure being the values generated from the networks output neurons during testing. These values were used as a point of comparison between networks.

METHODOLOGY TOWARDS IDENTIFYING THE NETWORK ARCHITECTURE AND LEARNING PARAMETERS

The three classes of Iris were allocated bit string representation as shown in table VI.

Classifications	Bit string
Iris-setosa	1 0 0
Iris-versicolor	0 1 0
Iris-virginica	0 0 1

Table VI. Classifications and their bit string representation

The bit strings represent the expected activation of the output neurons in regards to a pattern, and are used in the training, and the testing of the networks.

Stage 1

The networks constructed during the first stage of the network architecture development where constructed using the four input, and three output neurons necessitated by the problem domain, and networks with hidden layer neurons numbering 3, 4, 5, 6, 7, 8, and 9. The initial value for the cycles was 100, with the other variables being left at the JavaNNS defaults. The column normalised values proved to be the fittest in the comparison of the normalisation techniques. The poor results from the analysis of these initial networks lead to the decision to increase the number of cycles.

The value for the cycles was increased to 1000. The results showed improvement, with the plotting of the misidentified patterns, from both the testing and training sets against the number of hidden nodes demonstrating an improvement of the results with an increase in the number of hidden neurons, as shown in Figure 2.

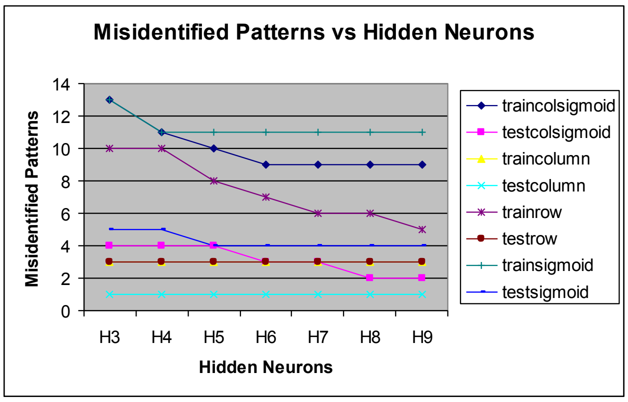

Figure 2. Misidentified Patterns vs Hidden Neurons

Plotting of the training and testing sets overall data set errors also demonstrated improvement of results with increases in the number of hidden layer neurons. Some informal networks generated for the purpose of identifying an appropriate number of cycles identified 70000 as a point of improvement under the constraints of nine hidden neurons, and column normalisation.

Stage 2

The second stage of the network architecture development consisted of the construction of networks for each of the normalisation techniques with 3, 6 and 9 hidden layer neurons, utilizing 10,000/ 20,000/ …/ 80,000 cycles. At this stage the networks continued to be developed with the default values from JavaNNS. The results derived from the fittest examples from each normalisation technique were plotted against each other so as to identify the normalisation technique with the best performance. The normalisation technique with the best results was demonstrated to be that of column normalisation as shown in Figure 3.

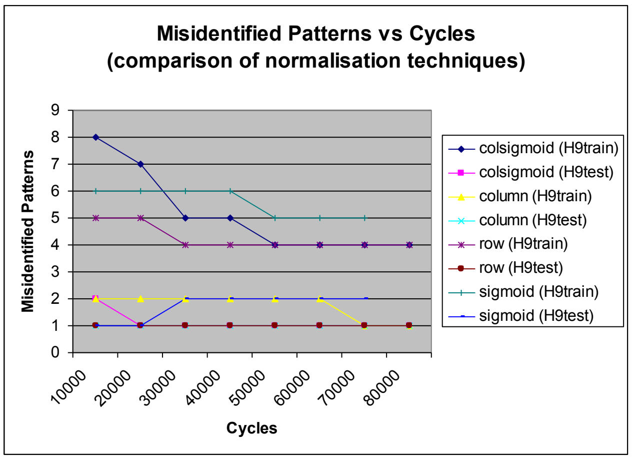

Figure 3. Misidentified Patterns vs Cycles
A comparison of the best stage 2 results for the different normalisation techniques

Stage 3

The third and final stage towards the development of the network architecture involved the generation of networks with 8, 9, 10, 12, and 15 hidden neurons, and step widths of 0.1, 0.15, and 0.2. Each network was trained over 40000, 80000, and 120000 cycles, with the maximum non-propagated error set at 0.1, and the number of update steps to 1. The results from the analysis of the results, as shown in Table VII, were used towards the identification of the fittest network for the iris classification problem.

Step Width	**0.2**			**0.15**			**0.1**		
Cycles (*10000)	4	8	12	4	8	12	4	8	12
Misidentified patterns in:									
Train H8	2	2	1	2	1	1	1	1	1
Test H8	1	1	1	1	1	1	1	1	1
Train H9	2	1	1	1	1	1	1	1	1
Test H9	1	1	1	1	1	1	1	1	1
Train H10	2	1	1	1	1	1	1	1	1
Test H10	1	1	1	1	1	1	1	1	1
Train H12	2	1	1	1	1	1	1	1	1
Test H12	1	1	1	1	1	1	1	1	1
Train H15	1	1	1	1	1	1	1	1	1
Test H15	1	1	1	1	1	1	1	1	1

Table VII. A comparison between the misidentified patterns from networks with differing numbers of hidden layer (H), and step width.

Conclusions

The fittest network architecture used column normalisation, 40000 cycles, 1 hidden layer with 9 hidden neurons, a step width of 0.15, a maximum non-propagated error of 0.1, and a value of 1 for the number of update steps.

The reasoning for the selection of the 9 hidden neurons was based in part on the following text from “Grouping parts with a neural network” by Chung, Y and Kusiak, A [2]:

> Hecht-Nielson (16) found that 2N+1 hidden neurons, where N is the number of inputs, are required for a one-hidden-layer back propagation network.

As the number of inputs is four, the required number of hidden neurons was calculated to be nine. Other contributing factors towards the decision to restrict the hidden layer to 9 hidden neurons were identified. One of these factors related to the rate of change in the overall set errors. The overall set errors for the networks with 8, 9, and 10 hidden neurons, shown in table VIII, shows a marked deceleration in the depreciation of the error. This particular example is in the context of a step width of 0.15, a non-propagated error of 0.1, and a value of 1 for the number of update steps.

Cycles	**Train**	**Test**

	H8	
40000	45.8743	21.8767
80000	45.2341	21.6376
120000	44.9451	21.5327
	H9	
40000	45.5431	21.7035
80000	44.9928	21.504
120000	44.6686	21.3828
	H10	
40000	45.3209	21.588
80000	44.8112	21.398
120000	44.4725	21.2682

Table VIII. A comparison between differing numbers of hidden neuron (H), and the associated overall set errors (in the context of the number of cycles).
Note: Step width of 0.15, a non-propagated error of 0.1, and a number of update steps of 1

Another reason for the decision to use nine hidden neurons is in reference to the specific error of the patterns incorrectly identified. Table IX shows the error for the incorrectly identified training pattern in the networks with 9 and 10 hidden neurons. Due to the increase in all of the output neurons values, it was judged to be inefficient to increase the number of hidden neurons beyond nine. The network with 8 hidden neurons was discarded due to a poorer performance in the identification of the patterns under the constraints of a 40000 cycle training regime.

Cycles of 40000, step width of 0.15, a non-propagated error of 0.1, and a number of update steps of 1			
Hidden neurons: 9			
	Pattern: 57 is incorrect!!!!!!!!!!!		
Expected	0 1 0		
Actual	0 0 1		
Results as generated	0.00212	0.41811	0.80217
Hidden neurons: 10			
	Pattern: 57 is incorrect!!!!!!!!!!!		
Expected	0 1 0		
Actual	0 0 1		
Results as generated	0.00213	0.41825	0.80385

Table IX. The error within the incorrectly identified training pattern of the networks with 9 and 10 hidden neurons
Note: Step width of 0.15, a non-propagated error of 0.1, and a number of update steps of 1

The step width of 0.15 was decided upon as a result of the comparison between the results achieved under the conditions of a step width of 0.1, 0.15, and 0.2. The results, shown in Table X, demonstrate that the overall set error was least in the vicinity of the step width of 0.15.

Step width	H9	
0.1	45.8052	21.7768
0.15	45.5431	21.7035
0.2	45.7166	21.8375

Table X. A comparison between step widths of 0.1, 0.15, and 0.2 in the 9 neuron hidden layer neural network

This network has been developed towards being able to identify Iris classifications on the basis of the attributes supplied. It is assumed that the data set is a fair reflection of the populations they represent, and that the efficiency of the network structure, as well as the efficiency of the training, are of importance.

ACKNOWLEDGEMENTS:

- Iris Plants Database
  Creator: R.A. Fisher
  Donor: Michael Marshall

REFERENCES:

[1] Negnevitsky, M. *Artificial Intelligence: A Guide to Intelligent Systems*, First Edition, Harlow, Pearson Eduction, 2002, Page 175-178.

[2] Chung, E. Kusiak, A. *Grouping parts with a neural network*, "Journal of Manufacturing Systems[Full Text], volume 13, issue 4, Page 262, Available ProQuest, 02786125 , 20-04-2003".