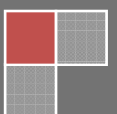
2009

Design: Genetic algorithms and neural networks

William Sayers (05025397)

William Keith Paul Sayers
Faculty of Advanced Technology
26/01/2009



Contents

1.0 – Introduction 4
1.1 – Program Requirements 4
1.1.1 – Genetic Algorithm Settings 4
1.1.2 – Back Propagation Settings 4
1.1.3 – Neural Network Training Results Display 5
1.1.4 – Genetic Algorithm Training Results Display 5
2.0 - Overall Design 6
2.1 – General Structure of Application 6
2.2 – Creation of DLL's 7
2.3 – User Interface Design and Programming in C# 7
2.4 – Communication of data between unmanaged and managed code 8
3.0 - User Interface 10
3.1 – Option input 10
3.2 – Graphical representation of Error 10
3.2.1 – ZedGraph .NET component 11
3.3 User interface implementation 12
4.0 - Back Propagation Neural Network 13
4.1 – Back Propagation Algorithm 13
4.2 – FANN Library 14
4.2.1 – FANN Data Structures 14
4.2.2 – FANN Methods (Nissen, Reference Manual) 14
4.3 – Back Propagation Neural Net Implementation 15
5.0 – Genetic Algorithm Neural Network 17
5.1 – Genetic Algorithm 17
5.1.1 – Elitism 17
5.1.2 – Crossover 17
5.1.3 – Mutation 18
5.2 Genetic Algorithm Implementation 18
5.2.1 Genetic Algorithm Class 18
6.0 – Problems Encountered 20
7.0 – Appendix I – Prototype Progress (Screenshots) 21
7.1 – Options Input 21
7.2 – Genetic Algorithm output 22

7.3 – Back Propagation Output 23
7.4 – Mean Squared Error Back Propagation 24
7.5 – Mean Squared Error Genetic Algorithm 25
8.0 – Appendix II – Licenses 26
8.1 – LGPL (FANN & ZedGraph license) 26
9.0 – Appendix III - Use Case Diagram 27
9.1 - Train Network (Genetic Algorithm) 27
9.1.1 – Pre-condition 27
9.1.2 – Post-condition 27
9.1.3 – Assumptions 27
9.1.4 – Triggers 27
9.2 – Train Network (Back Propagation) 28
9.2.1 – Pre-condition 28
9.2.2 – Post-condition 28
9.2.3 – Assumptions 28
9.2.4 – Trigger 28
9.3 – Show About 28
9.3.1 – Pre-condition 28
9.3.2 – Post-condition 28
9.3.3 – Assumptions 28
9.3.4 – Triggers 28
9.4 – Exit Program 28
9.4.1 – Pre-condition 28
9.4.2 – Post-condition 28
9.4.3 – Triggers 28
10.0 – Appendix IV - Works Cited 29

1.0 – Introduction

The purpose of this application will be to allow the user to run either a back-propagation trained neural network, or a genetic algorithm trained neural network and present the results of the network in such a fashion as to allow analysis and comparison of the training methods and their suitability in different situations.

1.1 – Program Requirements

The program must be able to:

- Run a genetic algorithm on a feed forward fully interconnected neural network to train the network.
- Run a back propagation algorithm on a similar network to train the network.
- Allow the user to set appropriate variables (see sections 1.1.1, 1.1.2) to adjust the execution of the learning algorithm selected for the neural network.
- Allow the user to select which learning algorithm they wish to use.
- Display the results of running that algorithm on a neural network in a meaningful fashion.

1.1.1 – Genetic Algorithm Settings

The genetic algorithm settings portion of the program must allow the user to adjust the mutation rate, the crossover rate and the number of generations to process with these parameters.

The mutation rate and the crossover rate should be adjustable via a probability parameter, which will influence how much mutation and crossover occurs, whilst still retaining the random aspects of both functions.

The number of generations should be alterable via a simple integer that the user can input.

- Mutation rate (percentage)
- Crossover rate (percentage)
- Generations to iterate through

1.1.2 – Back Propagation Settings

Back Propagation offers fewer parameters to adjust than a genetic algorithm performing the same task, as there is no randomness inherent in the algorithm that could accept user input.

Only one parameter is required to allow a back propagation algorithm to run, the number of iterations that it is allowed to run.

Other implementations of back propagation could allow parameters such as "momentum" which would be the amount added to each alteration decreased in inverse proportion to the amount of iterations completed, "desired bit error" in which the algorithm would continue to train until a certain target level of error is reached at which point the algorithm would halt, or "desired mean squared error" in which the algorithm continues to train until a certain mean squared error value is reached.

The implementation I plan to use in this application will be fairly basic (although not necessarily slow because of that) and probably require none of these parameters however.

- Iterations to run

1.1.3 – Neural Network Training Results Display

The training results should be shown both in the form of a text box (not editable by the user) and a graph displaying the mean squared error per iteration of the network or for chosen iterations.

The text box will show the output that would previously have been routed to the standard output (console window) during the testing phase, which will consist of a list of the mean standard error per iteration or at chosen iterations and the output from a test run of the neural network.

The graph will map mean squared error over iterations, to allow you to visualise the drop in error as the training algorithm progresses.

1.1.4 – Genetic Algorithm Training Results Display

The training results will be show again using a text box which is not user editable and a graph displaying the mean squared error of the best neural net the algorithm has produced per iteration.

The text box again shows the output that would previously have been routed to the console during the testing phase, which will consist of a list of the mean error of the best neural net the algorithm has produced per iteration.

The graph will map the mean squared error measurements over the iterations to allow you to visualise the way in which the genetic algorithm moves through the search space, dropping previous best candidates and replacing them as time goes on.

2.0 - Overall Design

2.1 – General Structure of Application

The application will be generally structured after a model view controller design pattern, with the two DLL's forming the model and the C# managing the controller and the view parts of the solution.

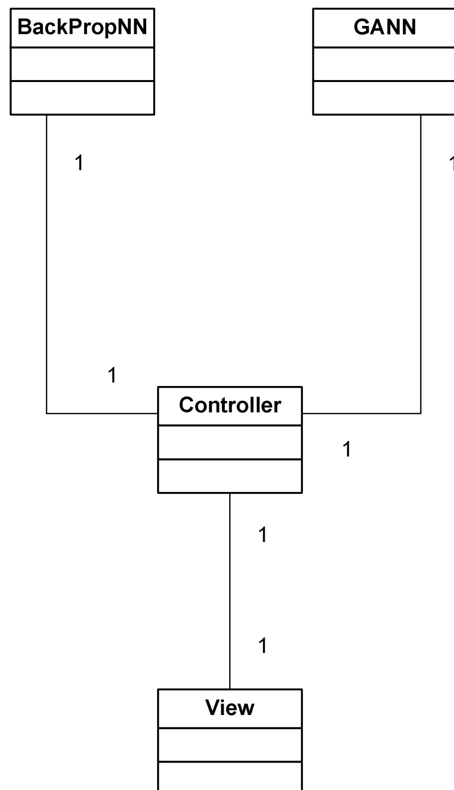


Figure 1 - Model view controller style implementation. BackPropNN and GANN will be implemented as dynamic link libraries to be called by C#

Controller and view will be implemented in C# as part of the graphical user interface, whereas BackPropNN and GANN will be implemented in C++ as dynamic link libraries that the C# user interface will load and use functions from.

The reasoning behind this approach is that C# lends itself very well to rapid prototyping and user interface programming whereas C++ is a very effective language for data manipulation at speed. The program could be completed entirely in C# or C++ but the end product would be less efficient in various ways (depending on the language) than simply using both languages and thus getting the best points of them both in a single application.

Rather than implementing the export of an object that will be used in C#, a series of functions will be implemented in the dynamic link libraries that allow the C# application to interact with the objects implemented in the dynamic link libraries.

2.2 – Creation of DLL's

When a function in C++ is compiled, the compiler "decorates" the function name. This is to allow function over-loading present in C++ applications to compile properly. However if we're going to call that function from another application (the function is in a dynamic link library) we need the function to not be decorated to enable us to call the correct function from the application.

This is achieved by means of the "extern"C"" statement before the function name and then the "__declspec(dllexport)" tells the compiler that this function is to be exported for use in a dynamic link library.

```
extern "C" __declspec(dllexport) int functionX (int X);
```

Once you have a dynamic link library compiled with your desired access functions declared in this manner, loading the functions and using them in C# is quite easily accomplished using the DllImport attribute like so:

```
 [DllImport(@"c:\mydll.dll")]
public static extern int functionX(int X);
```

The function can now be called by C# code in the same scope, in the same way as any other function.

```
int temp1 = 0;

int temp = function(temp1);
```

2.3 – User Interface Design and Programming in C#

C# in Microsoft visual studio .NET and above allows you to essentially "draw" the user interface onto the screen and then customize it using properties for each type of user interface object.

This makes it a very good choice for rapid application development and prototyping in general.

The basic layout of the user interface is designed to be along these lines:

<table>
<tr><td colspan="3">Menu Bar</td></tr>
<tr><td colspan="3">Tab Bar (switch between property entry and graph)</td></tr>
<tr><td rowspan="3">Genetic Algorithm Properties</td><td>Back propagation Properties</td><td rowspan="2">Algorithm Selection</td></tr>
<tr><td>Iterations to report</td></tr>
<tr><td>Dataset Selection</td><td>Run Button</td></tr>
<tr><td colspan="3">Textual output of the neural network training and testing.</td></tr>
</table>

Figure 2 - Graphical user interface design

The second tab will display a graph with the data collected from the run of the neural network training algorithm. This data will be collected inside the dynamic link libraries and then extracted and parsed by the C# application when needed. It will be in the form of an array of floating point variables that can be accessed via a function that takes the index as a parameter and returns the array element corresponding to that index.

2.4 – Communication of data between unmanaged and managed code

In terms of communicating data over the managed/unmanaged threshold there are several issues that need to be overcome.

Very basic types, such as floats, ints, or doubles, can for the most part be passed directly between unmanaged C++ and managed C# with no issues.

More complex types such as arrays, or strings, require some working around to function.

Fortunately a workaround is available for float arrays in a small scale application such as the prototype I'm building, you can simply have a function in your library that takes the index as a parameter and returns the element for that index. This also allows you to implement some protection to avoid going out of bounds in the unmanaged code.

For the strings, there is a further problem in that C++ by default stores strings in ANSI format, whereas C# being a more modern language stores strings in Unicode format.

The solution to this is to use wide character types in C++ store the string in C format as opposed to in a C++ object (i.e. an array of characters) and in C# accept the string as an integer pointer.

Once you have the string being passed to C# in the form of an integer pointer you can use the marshal class to convert the integer pointer to a C# Unicode string object.

3.0 - User Interface

The user interface for the application is to be developed (as previously mentioned) in C# using the visual studio 2008 design tools to create the user interface.

Interaction with the user interface from a programming perspective is then managed by clicking and dragging or altering properties (in design mode) or by altering or checking the values of properties from code at runtime.

3.1 – Option input

Most of the options in the user interface will be either input of a number, or a choice between one or two options (such as selection of the learning algorithm).

For the numerical input I plan to use masked text boxes which will allow me to set a mask property so that only numbers can be entered into those text boxes. Then in the code I will check that the numbers entered are values that will make sense to the libraries before passing the data onto the library indicated by the user choice to be incorporated into the generated object and run.

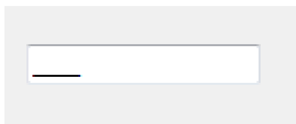


Figure 3 - A masked text box with max "0000"

For the choices between options I will use “RadioButton” objects, which allow the user to select one of the objects within a frame and only one. Upon selecting a new object all other objects in that frame have their “checked” property set to “false”.

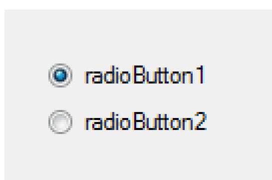


Figure 4 - A collection of radioButtons

3.2 – Graphical representation of Error

A graphical representation of the error could be hand-coded from scratch in C# but there are user controls to check this kind of thing.

Amongst the graphing controls researched and experimented with are:

- NPlot (Nplot Charting Library for .NET)
- GoogleChartSharp (GoogleChartSharp)
- ZedGraph (ZedGraph)

All three of which are referenced in section 10.0. ZedGraph is the control I have decided upon using, because it's both simple to use, easy to incorporate into the project and flexible in terms of displays (in case I wish to add more graphing options at a later stage).

3.2.1 – ZedGraph .NET component

"ZedGraph is a set of classes, written in C#, for creating 2D line and bar graphs of arbitrary datasets. The classes provide a high degree of flexibility -- almost every aspect of the graph can be user-modified. At the same time, usage of the classes is kept simple by providing default values for all of the graph attributes. The classes include code for choosing appropriate scale ranges and step sizes based on the range of data values being plotted." – ZedGraph Wiki

3.2.1.1 – Adding the ZedGraph control

To add the ZedGraph control to the project right click inside the "general" area of the toolbox in visual studio, select "choose items…" and then "browse". Navigate to the "zedgraph.dll" file and then you should see a "ZedGraphControl" option to be selected from the toolbox.

You also need to add the appropriate references to ZedGraph to your project.

Once ZedGraph has been added to your application the control can be manipulated by getting and setting properties from within code, similar to any other control object in .NET.

The "pointpairlist" data type will become available, which is basically a two dimensional list that you can add X and Y values too, ready for the graph to plot. In order to plot a graph, you first obtain a reference to the graph pane.

```
GraphPane myPane = zgc.GraphPane;
```

Once you have this reference, you can adjust the properties such as title, X axis title and Y axis title.

To add a line to the graph (more than one line per graph is entirely possible) you use the "AddCurve" function.

```
LineItem myCurve = myPane.AddCurve( "Porsche",
      list1, Color.Red, SymbolType.Diamond );
```

The first parameter is the title of the series or “curve”, the second parameter is the “PointPairList” data previously constructed, the third is the colour of the line and the fourth is the type of symbols to use to indicate data points.

Once you’ve set all relevant properties and added all the curves you desire, the axes can be calculated afresh by calling the function AxisChange function, which is part of the pane object and can thus be accessed via your reference to the pane.

```
zgc.AxisChange();
```

3.3 User interface implementation

C# is based largely upon an event driven model of programming, so several functions will likely end up modifying a single option, or a single variable. I will only bother to detail the larger functions I think I’ll need in this section.

- Parsetext
  This function will take in all the textual input from the masked text boxes and parse them into integers or return an appropriately descriptive error message.
- Checkvalues
  This function will take the integers previously returned by the above function and check they’re within sensible values for use as dynamic link library parameters.
- RunNN
  This function runs the neural network, with the training algorithm selected on the user interface and the data collected, checked and parsed by the two above functions.

When the “Run” button is clicked, the three functions are called in the order presented above, resulting in the correct neural network dynamic link library being run according to the input information.

4.0 - Back Propagation Neural Network

The back propagation is (in this version of the application) being implemented via the FANN library (Nissen, FANN) which is an open source library that implements multilayer feed forward neural networks, fully or partially interconnected, in C.

4.1 – Back Propagation Algorithm

Training by back propagation involves the following steps:

- Present a set of training data
- Compare the networks output to the desired output at each output neuron and calculate the error.
- Adjust the weight of the output neurons to lessen the error.
- Assign responsibility to each of the neurons in the previous level, based on the strength of the weights connecting them to the output neurons.
- Adjust the weight of the neurons to minimise the responsibility.

In order to use a back propagation training algorithm you must have a non-linear activation function for your artificial neuron. A commonly used function is the sigmoid function:

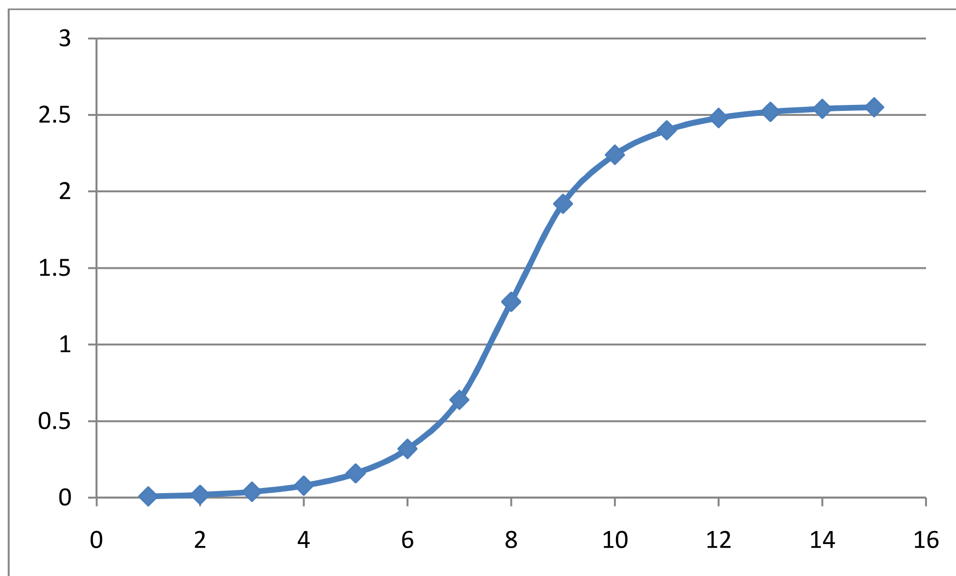


Figure 5 - A sigmoid function

Back propagation requires a differentiable activation function so that it can calculate the difference between a nodes output and the desired output for that node.

4.2 – FANN Library

The FANN library allows you to include a header file, call several functions to set properties and then run a back propagation neural network based on the properties you've set. It is a very simple and powerful open source solution distributed under the LGPL license.

4.2.1 – FANN Data Structures

4.2.1.1 - Neural network data structure (Nissen, struct fann)

This data structure contains all data required to create, train, run and generally operate the neural network.

4.2.1.2 - Neural network training data structure (Nissen, struct fann_train_data)

This data structure contains the data required to train the neural network on a given set of input and desired output values.

4.2.2 – FANN Methods (Nissen, Reference Manual)

This section briefly mentions the FANN methods that I intend to use in my implementation of a back propagation neural net. Methods I don't plan to use are not included in this section, but can be seen in the FANN reference manual (Nissen, Reference Manual), along with far more detailed descriptions of the functions being used.

- fann_create_standard
  Create a neural network with a specified number of layers and a specified number of nodes in each layer.
- fann_read_train_from_file
  Read a file from the hard disk that stores the training data for the current neural network.
- fann_set_activation_steepness_hidden
  Set the steepness of the hidden node activation functions.
- fann_set_activation_steepness_output
  Set the steepness of the output node activation functions.
- fann_set_activation_function_hidden
  Set the activation function (i.e. sigmoid, stepping) of the hidden layers.
- fann_set_activation_function_output
  Set the activation function (i.e. sigmoid, stepping) of the hidden layers.
- fann_set_training_algorithm
  Sets the training algorithm to use (there are various back-propagation based variants available).

- fann_init_weights

  Sets the weights initial values to minimise training time, according to the algorithm described by Nguyen & Widrow (Nguyen & Widrow, 1990).
- fann_train_epoch

  This function trains one epoch of the neural network with the training data given. An epoch is where each set of training data (inputs and an output) is considered exactly once.
- fann_test_data

  This function tests the network on the data given and calculates the MSE for the network on that data.
- fann_length_train_data

  This function returns the number of training patterns in the training data structure supplied.
- fann_run

  This function runs the neural network with the supplied data.
- fann_destroy_train

  This function destroys the training data structure in memory, freeing the memory for other applications to use.
- fann_destroy

  This function destroys the artificial neural network in memory, freeing the memory for other applications to use.

4.3 – Back Propagation Neural Net Implementation

The implementation will involve four functions exported from the unmanaged C++ dynamic link library. These functions are:

- runXORbp(iterations, report interval)

  Creates and runs the back propagation neural net, for the specified number of iterations, recording the MSE for every interval indicated by the second parameter.
- wchar_t output()

  Returns the output desired to be shown in the text box in the C# GUI.
- clear()

  Clears the memory structures and allows other applications to use the memory once more.
- float MSEdata(index)

  Returns the data from the array of floats storing the mean squared error at the index indicated.

It also will involve a function not to be exported, which will take all the parameters necessary to create and train a neural network.

- create_and_train_nn
  This function takes the input layers, output layers, hidden nodes, number of layers, desired error, number of iterations to run and interval of reports. Then trains and runs the neural network as indicated by the parameters.

Other data necessary to run the neural network will be stored as global variables, to allow easy access for the functions designed to pass this data to C# across the managed/unmanaged boundary.

- Vector MSE_vec
  This variable stores the mean standard error results to be returned to C++.
- outstr and *outch
  The variable "outstr" is the string object representation of the textual output data to be sent to the user interface. The variable "*outch" contains the C-string character array representation of the data to be sent to the user interface. The data must be sent as a character array to allow C# to correctly marshal the data into a format it understands.

5.0 – Genetic Algorithm Neural Network

Genetic algorithms are an attempt to copy the laws of natural selection to a certain extent in order to apply a random search algorithm to a search space.

The benefit of natural selection as a search algorithm is that you do not have to investigate all areas of the problem in order to come up with a good solution. Genetic algorithms are also parallelised by design, giving them an advantage over more traditional serialized search techniques.

5.1 – Genetic Algorithm

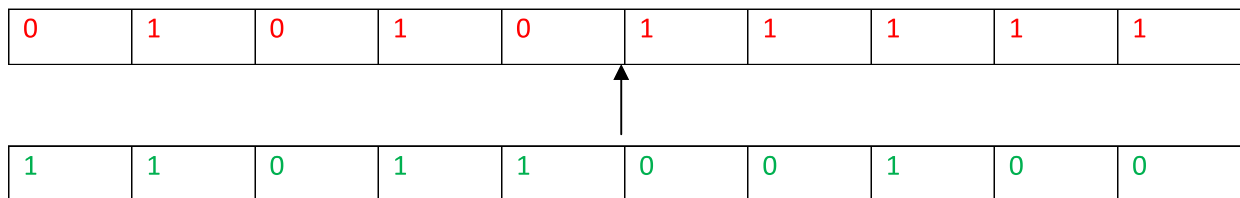
The algorithm to be used in the application will implement crossover, mutation and elitism.

5.1.1 – Elitism

Elitism is the idea of permanently retaining the best solution discovered so far, in order to avoid losing the best solution in a following generation. This usually increases the efficiency of the genetic algorithm and the quality of the eventual solution.

5.1.2 – Crossover

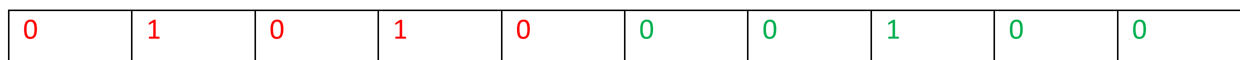
Crossover is standard in genetic algorithms and is a method by which new offspring can be generated.



0	1	0	1	0	1	1	1	1	1
1	1	0	1	1	0	0	1	0	0

Starting with two genomes, the crossover point is chosen, in this case the centre but it can be any point on the genomes.

The first part of the first genome and the second part of the second genome are then combined.



0	1	0	1	0	0	0	1	0	0

The genome produced by this method will combine characteristics of both the parent genomes, hopefully moving closer towards a solution.

5.1.3 – Mutation

Mutation adds randomness to the genetic algorithm; a mutation probability will be supplied as a parameter to the genetic algorithm. Each iteration the genomes will be checked and each bit in the genome will have a probability of being flipped or altered.

5.2 Genetic Algorithm Implementation

The neural network with a genetic learning algorithm will be a class based design, based on two classes. One class will handle the genetic algorithm methods and data and the other class will handle the neural network methods and data. The genetic algorithm implementation is being adapted from an online tutorial (Matthews).

The original implementation has no probability for crossover (crossover was effectively always one hundred percent), no recording of any data for later analysis, no allowance for altering usage variables (such as population size, number of generations, iterations to run), used mean error and not mean squared error in the fitness function (making it less comparable to the back propagation data used for analysis) and no elitism included in the solution.

The above mentioned functionality must be added to the tutorial code prior to my using it.

5.2.1 Genetic Algorithm Class

The genetic algorithm class needs to store the following information.

- Population Size
- Maximum number of generations to run
- Probability of crossover occurring
- Probability of mutation occurring
- Elitism error threshold
- Current best solution
- Report interval (in generations)
- List of the errors of the best solution

It requires functions to perform the following operations.

- Initialise and create all variables (constructor/s)
- Destroy any created variables (destructor/s)
- Run the genetic algorithm
- Sort the population by fitness
- Generate a new population

The population of the genome will be stored, instead of in the form of binary strings, in the form of arrays of floating point numbers. Mutation will be implemented by means of randomly adding or subtracting one from all the floats for one population member.

5.2.1.1 Running the Genetic Algorithm

The function to run the genetic algorithm will probably consist of nested loops, to run the training for each member of the population once for each generation (training function is contained in the neural net class and simply returns the difference between the desired answer and the actual answer as produced).

After running the training for the population members, sort according to fitness then generate a new population prior to the next generation being processed.

5.2.1.2 Sorting the population by fitness

Since the errors were generated in the previous process (running training for each population member) a simple bubble-sort can be used to get the population members in order of error.

5.2.1.3 Generate a new population

The generation of a new population, will involve looping through the current population, for each member of the population choosing two random "parents" in the current population.

If a random number between 0 and 100 is less than the given probability for crossover, generate a new population member out of the first half of the weights for the first selected parent and the second half of the weights for the second selected parent.

If another random number between 0 and 100 is less than the given probability for mutation, increase or decrease the value of the weights for the two selected "parents" by one or minus one.

5.2.1.4 Skewing towards optimal replacements

When choosing the two random parents one will be selected from the "top" of the genome and one will be selected from the "bottom" of the genome. The parent from the "bottom" will be the one to be replaced in the event of crossover. If no crossover takes place, both parents will be left same

6.0 – Problems Encountered

This section details the problems encountered and solved so far in development of the prototype.

Most of the problems were in the area of back-propagation. I wanted a smoothly curving back-propagation algorithm that did not oscillate wildly either prior too, or after minimising the error.

The back-propagation algorithms available with FANN are as follows (Nissen, Datatypes):

- FANN_TRAIN_INCREMENTAL

This is a standard back propagation algorithm where the weights are updated after each training pattern. This means that the weights are updated many times during a single epoch.

- FANN_TRAIN_BATCH

Standard back propagation algorithm, where the weights are updated after calculating the mean square error for the whole training set. This means that the weights are only updated once during an epoch.

- FANN_TRAIN_RPROP

This is a more advanced batch training algorithm which achieves good results for many problems.

- FANN_TRAIN_QUICKPROP

This algorithm is a batch training algorithm which achieves good results for many problems.

For my problem, standard back propagation training by batch (FANN_TRAIN_BATCH) provided the best and most consistent results.

7.0 – Appendix I – Prototype Progress (Screenshots)

7.1 – Options Input

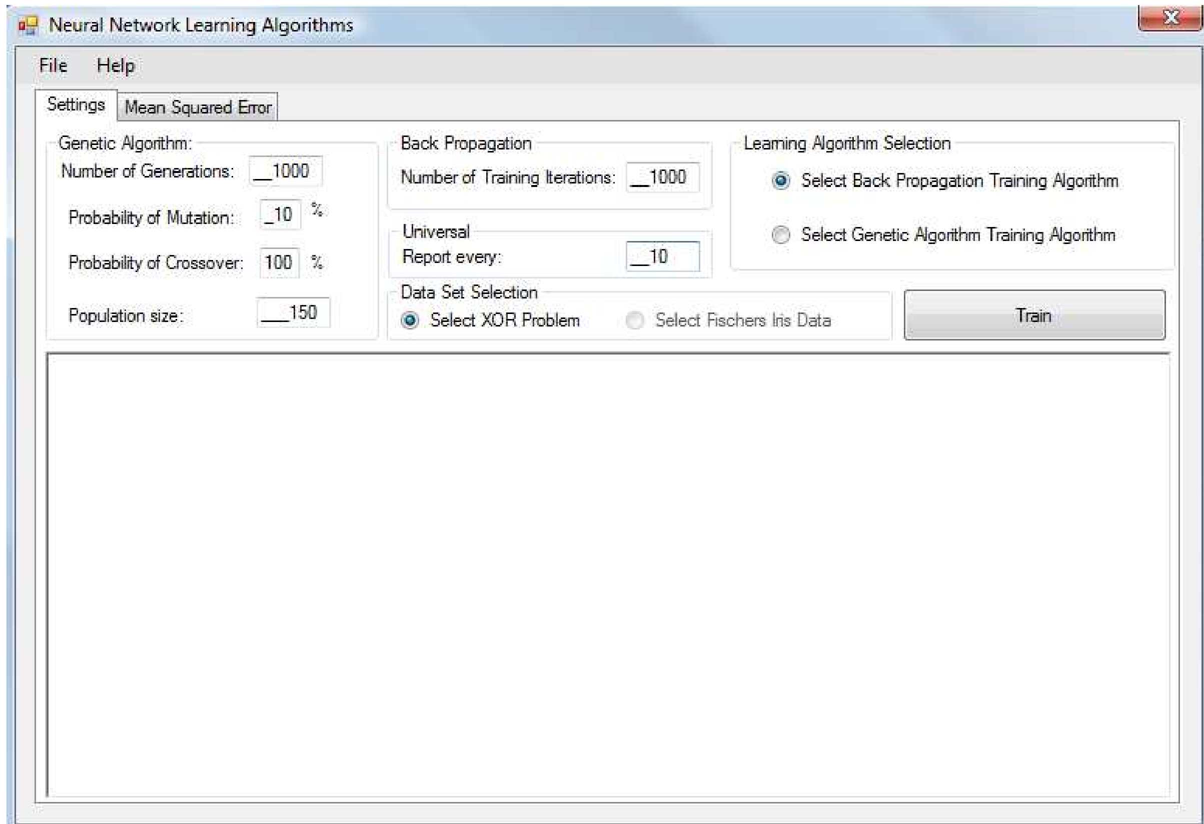


Figure 6 - Options input screen, with options filled in

7.2 – Genetic Algorithm output

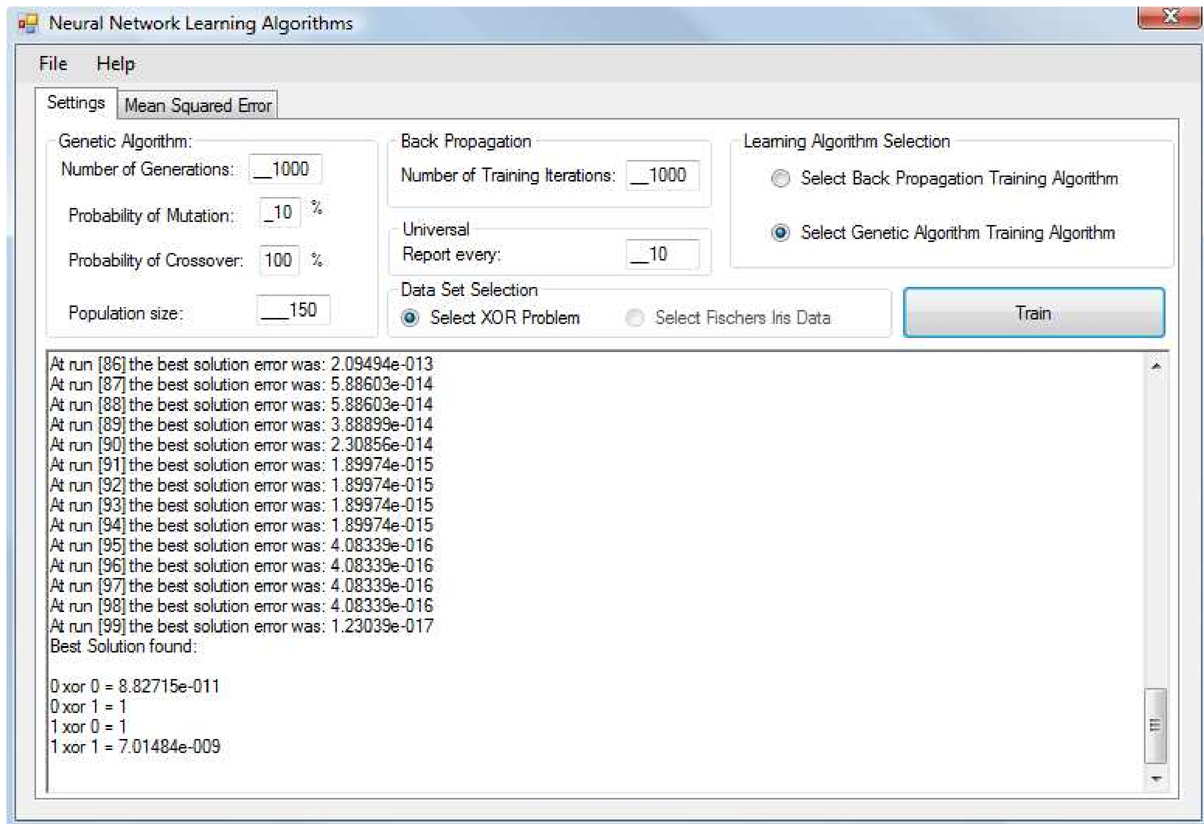


Figure 7 - Genetic Algorithm training algorithm output

7.3 – Back Propagation Output

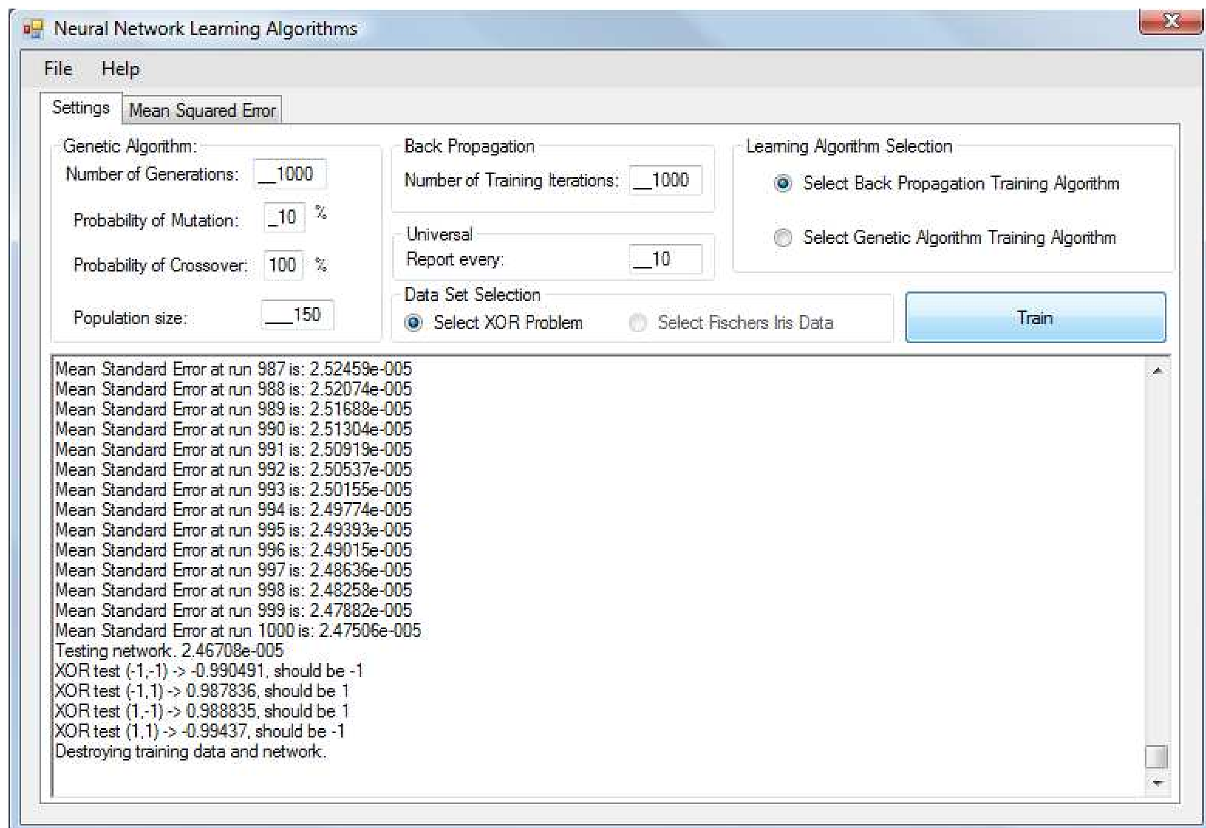


Figure 8 - Back propagation training algorithm output

7.4 – Mean Squared Error Back Propagation

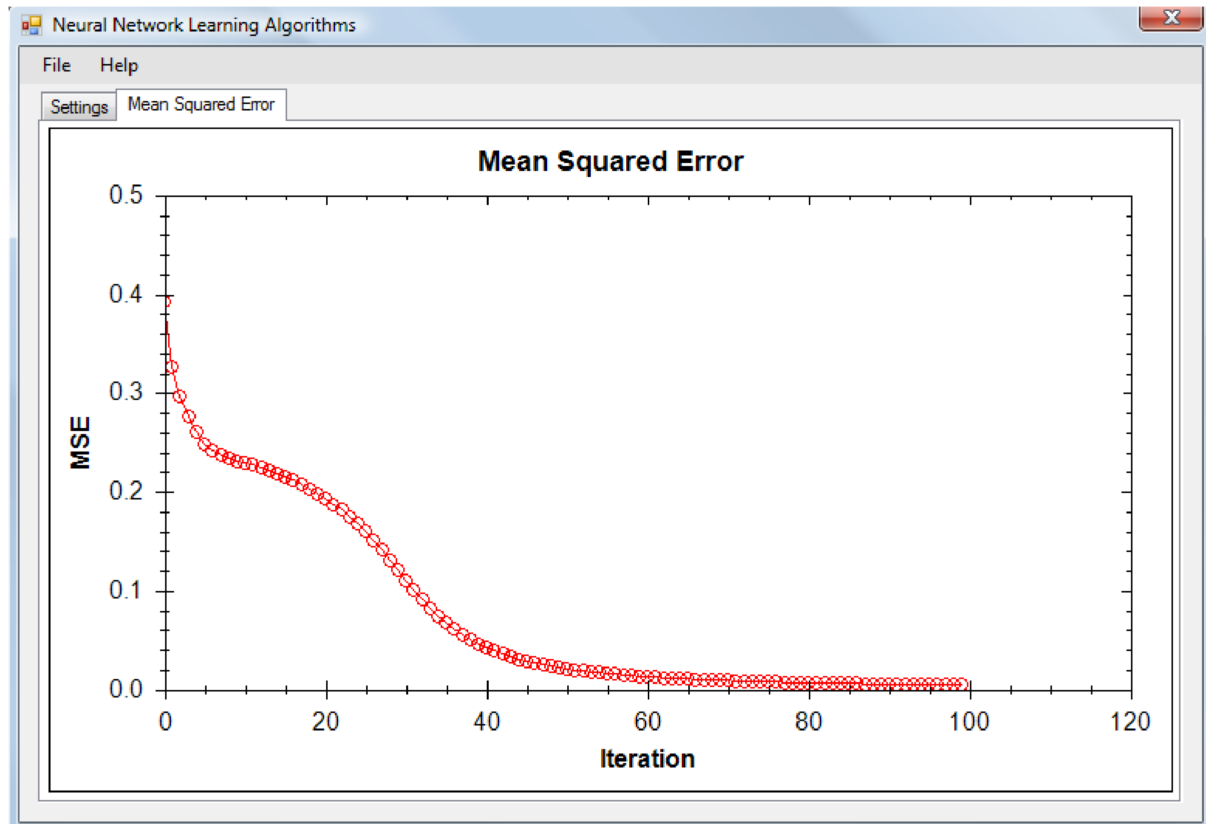


Figure 9 - Back propagation training graph

7.5 – Mean Squared Error Genetic Algorithm

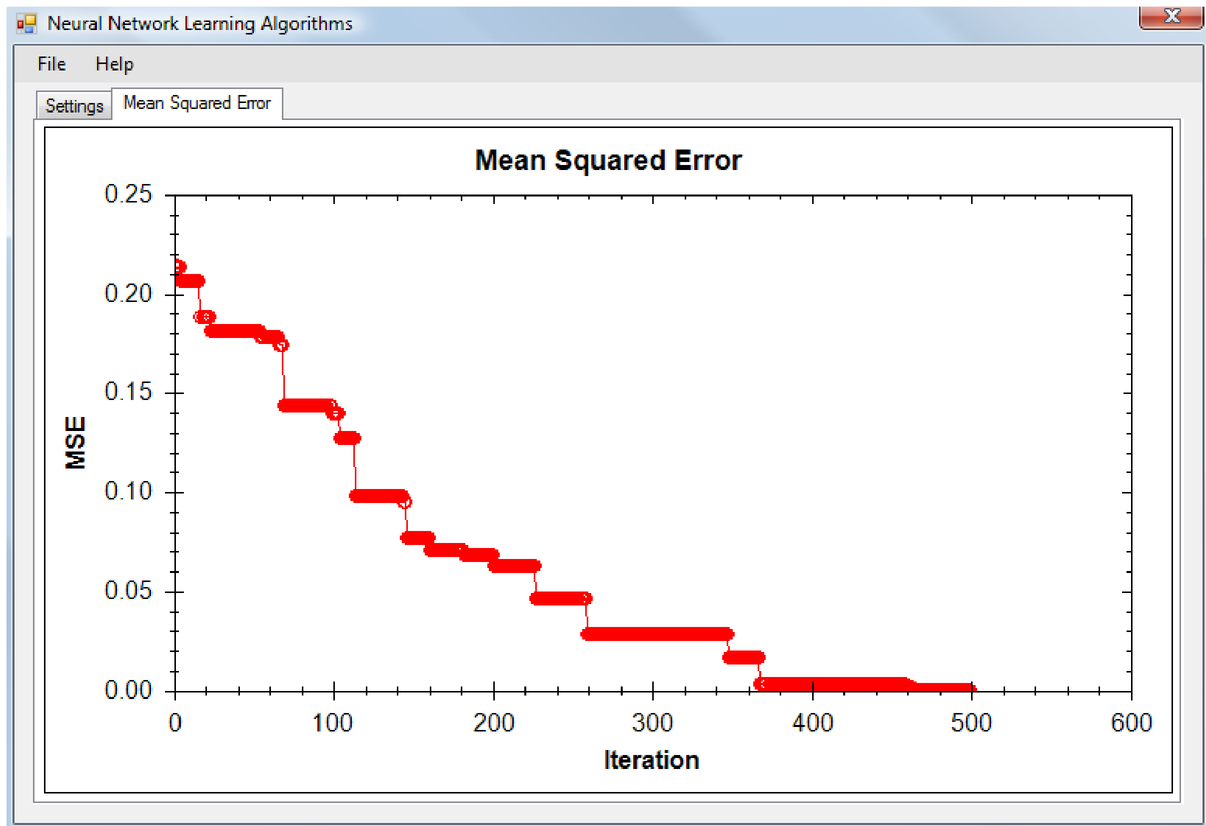


Figure 10 - Genetic algorithm training graph

8.0 – Appendix II – Licenses

8.1 – LGPL (FANN & ZedGraph license)

http://www.gnu.org/licenses/lgpl.html

9.0 – Appendix III - Use Case Diagram

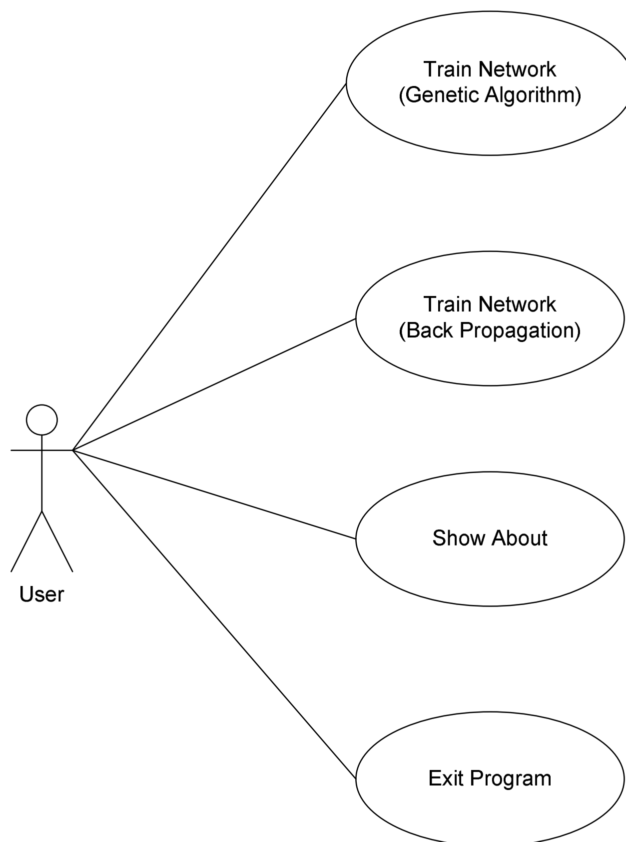


Figure 11 - Use case diagram

9.1 - Train Network (Genetic Algorithm)

9.1.1 – Pre-condition

All options have been input correctly (numbers within sane values).

9.1.2 – Post-condition

The neural network has been trained to achieve close to optimum values by a genetic algorithm.

9.1.3 – Assumptions

Neural network training data is available (hard coded or as part of a file).

Sufficient memory and time is available.

9.1.4 – Triggers

“Train” button is clicked by the user with “Genetic Training Algorithm” selected.

9.2 – Train Network (Back Propagation)

9.2.1 – Pre-condition

All options relevant to back propagation have been input correctly (numbers within sane values).

9.2.2 – Post-condition

The neural network has been trained to achieve close to optimum values by a back propagation algorithm.

9.2.3 – Assumptions

Neural network training data is available (hard coded or as part of a file).

Sufficient memory and time is available.

9.2.4 – Trigger

“Train” button is clicked by the user with “Back propagation training algorithm” selected.

9.3 – Show About

9.3.1 – Pre-condition

Application has been loaded.

9.3.2 – Post-condition

“About” form has been displayed on the screen.

9.3.3 – Assumptions

Sufficient memory is available to load the “About” form.

9.3.4 – Triggers

User clicks “About” on the help menu.

9.4 – Exit Program

9.4.1 – Pre-condition

The application has been loaded and neither neural network training algorithm is running.

9.4.2 – Post-condition

The application has exited memory.

9.4.3 – Triggers

The user clicks “Exit” on the file menu, or the user closes the main window.

10.0 – Appendix IV - Works Cited

GoogleChartSharp. (n.d.). Retrieved 21 01, 2009, from http://code.google.com/p/googlechartsharp/

Matthews, J. (n.d.). *generation5*. Retrieved 26 01, 2009, from XORGA: http://www.generation5.org/content/2000/xorga.asp

Nguyen, D., & Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosinginitial values of the adaptive weights. *Neural Networks, 1990., 1990 IJCNN International Joint Conference on* , 21-26.

Nissen, S. (n.d.). Retrieved 21 01, 2009, from FANN: http://leenissen.dk/fann/

Nissen, S. (n.d.). Retrieved 26 01, 2009, from Datatypes: http://leenissen.dk/fann/html/files/fann_data-h.html#fann_train_enum

Nissen, S. (n.d.). *Reference Manual*. Retrieved 21 01, 2009, from FANN: http://leenissen.dk/fann/html/files/fann-h.html

Nissen, S. (n.d.). *struct fann*. Retrieved 21 01, 2009, from FANN: http://leenissen.dk/fann/fann_1_2_0/r1597.html

Nissen, S. (n.d.). *struct fann_train_data*. Retrieved 21 01, 2009, from FANN: http://leenissen.dk/fann/fann_1_2_0/r1837.html

Nplot Charting Library for .NET. (n.d.). Retrieved 21 01, 2009, from http://netcontrols.org/nplot/wiki/

ZedGraph. (n.d.). Retrieved 21 01, 2009, from http://zedgraph.org/wiki/index.php?title=Main_Page