

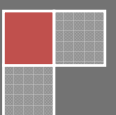
2008

Genetic Algorithms and Neural Networks

Supervised by: Colin W. Morris

This document covers the preliminary research towards a project to investigate the application of genetic algorithms with regards to replacing back propagation for the training of artificial neural networks.

William Keith Paul Sayers (05025397)
Computer Games Development: Final Year Project



Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

STATEMENT OF ORIGINALITY

SCHOOL OF COMPUTING

DEGREE SCHEME IN COMPUTING

LEVEL THREE PROJECT

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerised sources have been fully referenced, and I fully realise the consequences of plagiarising any of these sources.

Student Name (Printed)

Student Signature

Registered Scheme of Study

Date of Signing

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

Contents

1.0 - Neural Networks	4
1.1 - General description of neural networks	4
1.1.1 - The Human Brain	4
1.1.2 - Artificial neurons	5
1.2 - History of Neural Networks.....	6
1.3 - Advantages and disadvantages of Neural Networks	7
2.0 - Training by back-propagation	10
3.0 - Current Applications of Neural Networks	13
3.1 - Neural networks in medicine	13
3.2 - Neural Networks in business.....	13
3.3 - Object Trajectories.....	14
3.4 - Robot control	14
4.0 - Genetic Algorithms	15
4.1 - General Description of Genetic Algorithms	15
4.2 - History of Genetic Algorithms.....	15
4.3 - Advantages and Disadvantages of Genetic Algorithms.....	16
4.3.1 Advantages	16
4.3.2 Disadvantages.....	18
5.0 – Training with Genetic Algorithms	20
5.1 –Determining weight values with genetic algorithms	20
5.1.1 - Representation of a neural network within a genetic algorithm.....	21
5.2 Using genetic algorithms to determine neural network structure.....	21
5.2.1 – Representing neural network structure	23
6.0 - Combining C# and C++	25
7.0 – Application of the Research	26
8.0 Bibliography	28
9.0 - Appendix – Libraries Investigated.....	30

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

1.0 - Neural Networks

1.1 - General description of neural networks

1.1.1 - The Human Brain

The human brain is composed of many millions of nerve cells called “Neurons”. No one neuron makes decisions all by itself but the brain contains many millions of neurons which all act collectively to manipulate information presented to them.

A neuron is composed of a cell body, with many extensions called “dendrites” and one extension called an “axon”.

These extensions are “synapses”, they are where information enters the dendrite.

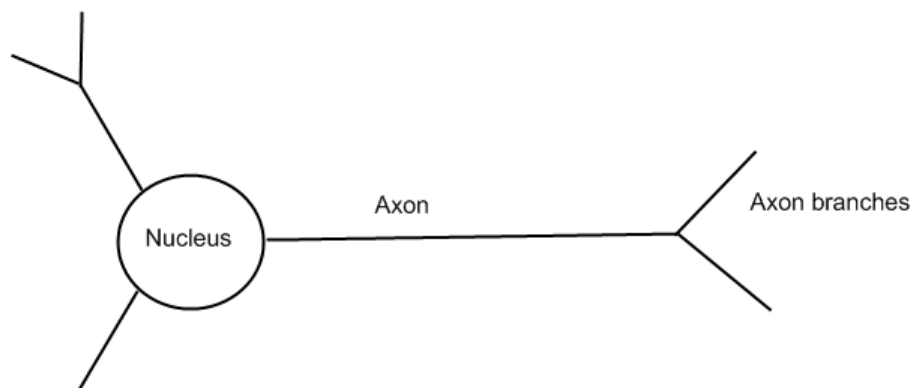


Figure 1 - A diagram of a neuron as it exists in the brain

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

If the input from the dendrites is large enough, an action potential is generated. This action potential then travels down the axon to the cells at the axon terminals (end of the axon branches) which may well be further neurons.

Neurons in the brain do not “fire” in increments or decrements, they either generate an action potential or they do not.

Synapses can be either excitatory or inhibitory. A signal arriving at an inhibitory synapse will tend the neuron towards not firing, whereas one arriving at an excitatory synapse will encourage the neuron to fire (Barber, 2006).

1.1.2 - Artificial neurons

So to construct an artificial neuron, we would need a similar model but represented in a data structure.

N inputs are required, plus a summing function and an activation function. The data will arrive at the inputs, they will be summed, then the activation function will fire or not fire based on the sum of $N_1, N_2 \dots N_i$.

The inputs would also require a weight for each input ($W_1, W_2 \dots W_i$) in order to effectively emulate the excitatory and inhibitory properties of synaptic pulses.

It would also be helpful to have an adjustable threshold processor, which could be referred to as bias (it would bias the artificial neuron towards firing, or not firing).

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

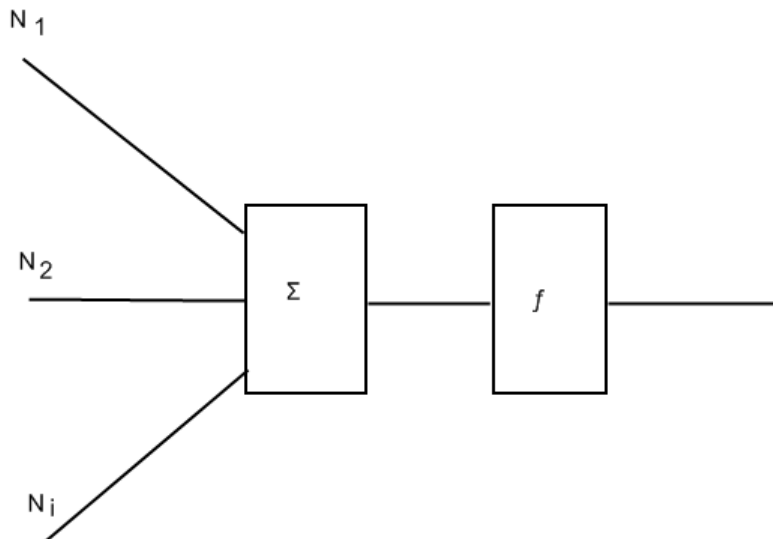


Figure 2 - A basic diagram of an artificial neuron

This model of a neuron is usually referred to as a “perceptron” (Barber, 2006).

1.2 - History of Neural Networks

Neural network research has its beginnings in psychology, the work of 19th century psychologists such as Freud, William James and others, contributed to the ideas that underpinned early neural network research (Blum, 1992).

McCulloch and Pitts formulated the first neural network model (Blum, 1992) which featured digital neurons but the neurons had no capability to learn.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

Frank Rosenblatt developed the perceptron model then, in 1957, however he was unable to come up with a reliable mathematically accurate mechanism for allowing multi-layer perceptrons to learn.

Various educational experimentations were made into usage and effectiveness of two layer perceptrons but the next major advancement in neural networks was not until 1974, when Werbos discovered the back-propagation algorithm (independently re-discovered by Parker in 1982).

Back-propagation allows for the training of multilayer perceptrons and in particular, perceptrons with N hidden layers, of N nodes.

Werbos discovered the algorithm whilst working on his doctoral thesis in statistics and called it "dynamic feedback", Parker discovered it independently in 1982 whilst performing graduate work at Stanford University in the United States of America and called it "learning logic" (Blum, 1992).

1.3 - Advantages and disadvantages of Neural Networks

The advantage of neural networks over more conventional methods is that they make possible or practical many things that with conventional solutions would be difficult or impossible.

They have a benefit over many problem solving techniques in that they are inherently parallel and thus can be implemented to run very effectively and efficiently on parallel hardware, in which situation they become an extremely fast as well as flexible tool.

The individual neurons have no time dependencies on each other and can as such each be run on a separate processor (or thread on a single processor) if desired without causing the problems often associated with such parallelism.

A good example of this is that neural networks have been implemented to run on architecture originally designed for processing three dimensional computer graphics, taking advantage of the massively parallel architecture of this hardware and the programmable

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

sections of the graphics pipeline (vector and fragment shaders) to produce extremely fast neural networks (Davis, 2001).

Neural networks excel at determining what data in a particular set is relevant, as irrelevant data will simply end up weighted so that it has zero or close to zero actual effect on the solution produced by the network.

One of the strengths of neural networks (based on the above) is that there is no need to establish before attempting problem solving what data is relevant and what data is not. We can simply present the neural network with all the data as opposed to following the circuitous and overly complex (by comparison) statistical approach:

1. Decide which data is relevant
2. Formulate a statistical model
3. Run model
4. Analyze
5. Build a system that incorporates what we've learnt

Another advantage of neural networks is that noisy data is not a real problem for them to learn to interpret; in fact noisy data can be used during training to try to make the post-training network as accurate as possible in as many situations as possible.

One disadvantage to neural networks can be that it is very difficult for a human being to analyse a trained neural network and describe why it may come up with one answer over another. This is a large disadvantage in areas such as medical diagnosis, where explaining why you have arrived at a particular diagnoses is an important part of the process.

Another (connected) disadvantage is since once a neural network is trained it is hard to tell why it is coming up with a particular solution, it can also be hard to tell how well (or badly) trained a neural network is.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

A classic example is that in the 1980's, the US military wanted to use a neural network to analyse images and determine whether they contained a concealed tank, for obvious purposes.

They collected one hundred images with a concealed tank, then a further one hundred images without. Fifty pictures of each set were set aside and the neural network was trained with the remaining one hundred images with and without tanks. After it had been trained, they tested it with the remaining fifty images that the neural network hadn't seen before and the neural network correctly identified most of the tanks.

The US military then tested the neural network with a further set of one hundred images, with and without tanks, only to find that on this round of testing, the network came up with apparently random answers.

Eventually someone noticed that in the original data set, all the images with tanks had been photographed on a cloudy day and all the images without on a sunny day.

Asked to differentiate between the two sets of pictures, the network had trained itself to recognise whether the sky was cloudy or not.

A further disadvantage of neural networks is that there is no formal methodology with which to choose the architecture, train the neural network, or even to verify that a neural network is correctly trained. A tool used to solve heuristic problems, requires largely heuristic construction and preparation.

The training times on neural networks can also be a disadvantage, although the author would point out, if the training time is still less than the time it would take to solve a problem without a neural network, a neural network is still the most efficient method to solve the problem.

Another possible criticism is that neural networks are extremely dependant on the quality and content of the original training data set. The more time spent training the neural network on quality data sets, the more accurate and in line with expectations and desires

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

the eventual result will be. This drawback is however one that is generally universal to computing (and many other areas) often referred to as “GIGO” or “Garbage in, Garbage out”. Neural networks make it more apparent however because (as in the previous US Military example) it can be difficult sometimes to decide what constitutes good data.

Fortunately since neural networks are (as has been mentioned above) good at ignoring irrelevant data, the usual best approach is to simply feed them as much data as can be obtained, covering as many relevant situations as possible; this has to be done within reason however, or it does exacerbate the problem with the time spent training the neural network.

The real strength of neural networks, in the opinion of the author, lies in their generalisation. When presented with data similar to data they’ve previously trained on, but not the same, they produce similar results. This is far more useful than it at first sounds and similar from a very high-level perspective to how the human brain and brains in general function.

2.0 - Training by back-propagation

In essence training by back propagation involves the 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:

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

1

$$s_c(x) = \frac{1}{1 + e^{-cx}}$$

The constant c in this function can be arbitrary and the value of c affects the curve of the function, which is similar to an S shape:

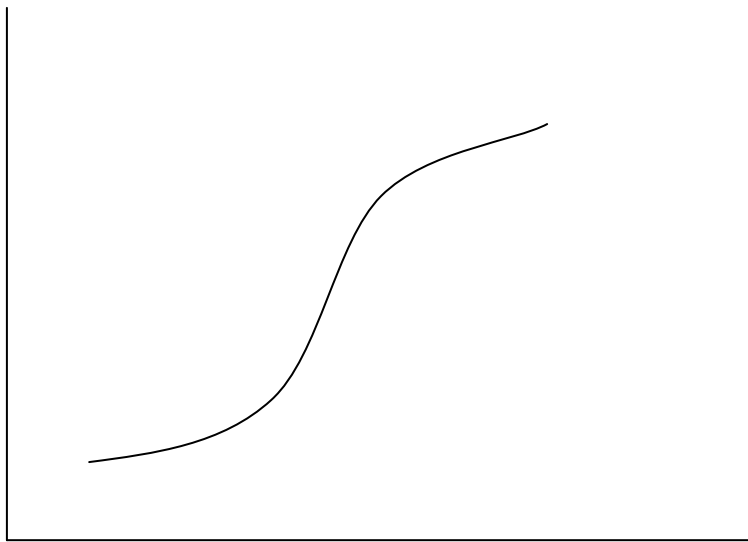


Figure 3 - A sigmoid curve

As you increase the value of c the curve moves closer to a step function as so:

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

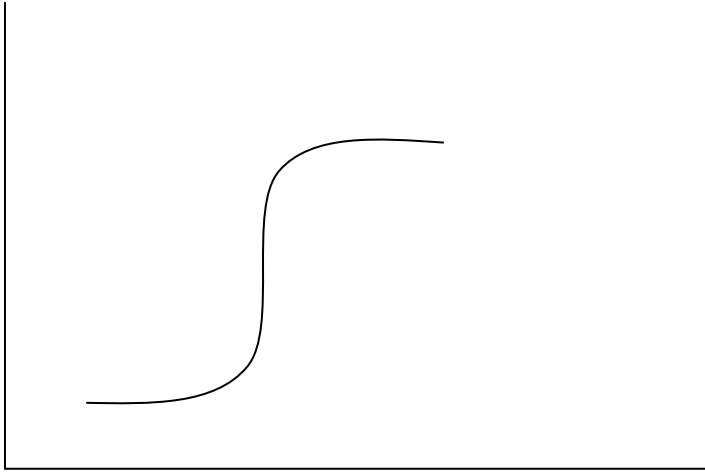


Figure 4 - A sigmoid curve, with c increased relative to the previous diagram.

The advantage of using a function like this is that it allows us to differentiate how close we came to the correct result, allowing us to apply our learning algorithm.

Using a differentiable function has its own issues associated with it however, since it is by the use of these functions that local optima become apparent in the search space.

Most of the neural network libraries researched (see section 9.0) implement a back-propagation solution and one of these libraries will be used in the project to implement a feed-forward back-propagation neural network, against which to compare the genetic algorithm trained neural network, since the focus of the project is not back-propagation but genetic algorithms.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

3.0 - Current Applications of Neural Networks

3.1 - Neural networks in medicine

There is a fairly large amount of research at the moment covering the applications of neural networks in medicine. One of the main areas being researched is using neural networks to recognise diseases/conditions via scans. This is particularly suitable work for neural networks because you don't need perfect examples of how to recognise the diseases and conditions, merely a variety of scans covering all possible permutations of the disease or condition.

You can also use the technology to model parts of the human biological system in order to better understand it, for example the human cardio-vascular system.

The main reasons for using a neural network to model the human cardio-vascular system are that a neural network is capable of adjusting itself to be relevant to a particular individual and also to adapt by itself to changes in that individual.

Research has been done in the 1980's into using neural networks to diagnose diseases however a clear problem with this approach is that when using a neural network it would be difficult to tell how a neural network has arrived at its diagnosis (Stergiou & Dimitrios, 1997).

3.2 - Neural Networks in business

Neural networks, being good at analysing patterns and predicting future trends, can fit very well into most business situations.

Neural networks have been used in business in the past for applications such as assisting the marketing control of airline seat allocations in the AMT (Airline Marketing Tactician).

A back-propagation neural net is integrated with the airline marketing tactician which monitored and recommended on booking for each flight, thus supplying information more or less directly linked to the airlines main income.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

Neural nets have also been used in credit scoring, mortgage screening and assessing possible borrowers (Stergiou & Dimitrios, 1997).

3.3 - Object Trajectories

Neural networks have also been applied to predicting the trajectory of a moving object in real time. The position, velocity and acceleration of the object being estimated by several neural networks using several of the most recent measurements of the object coordinates (Marshall & Srikanth, 1999).

3.4 - Robot control

Neural networks in robotics are usually applied to controlling manipulators (such as robotic arms in a car manufacturing plant).

There are four major problems that have to be solved in this kind of task (Artificial neural networks, 2008):

- Forward Kinematics
- Inverse Kinematics
- Dynamics
- Trajectory generation

The ultimate goal of course is to position the “effector” (I.e. a robotic hand or any other similar manipulator) in the appropriate position to grasp or otherwise manipulate an object or part of an object. When a robot arm is one hundred percent accurate that’s a fairly simple task (relatively speaking) involving the following steps (Artificial neural networks, 2008):

- Determine the target coordinates relative to the robot.
- Calculate the joint angles required for the arm to be in the appropriate position.
- Adjust the arm to the appropriate joint angles and close the effector.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

The question inevitably arises what a neural network is needed for in these circumstances. The answer is that a neural net allows a robotic arm to be much more flexible in its application and to adjust itself for its own wear and tear (Artificial neural networks, 2008).

4.0 - Genetic Algorithms

4.1 - General Description of Genetic Algorithms

Genetic algorithms attempt to copy natural laws to a certain extent in order to apply a random search pattern to a defined search space.

Natural selection (Darwin, 1859) very neatly avoids one of the larger problems involved in software design: how to specify in advance all the possible permutations of the original problem and how the program should react to those permutations.

By using similar techniques to natural selection it is possible to “breed” solutions to problems.

Most organisms evolve by means of sexual reproduction and natural selection (Darwin, 1859).

Sexual reproduction ensures mixing and recombination of genes. During the process chromosomes line up together and then cross over partway along their length, swapping genetic material, this mixing leads to much faster evolution than if off-spring simply copied the genetic material as it is in the parents.

Mutation plays a part in this as well, occasionally causing the genetic information, during this process, to alter slightly in a random way.

4.2 - History of Genetic Algorithms

Genetic algorithms first appeared on computers in the late 1950's and early 1960's, mostly designed to help evolutionary biologists' model aspects of natural evolution. By 1962 many researchers had independently developed evolution-inspired algorithms for function optimisation and machine learning but didn't get much follow up to their work.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

In 1965 one Ingo Rechenberg, then of the technical University of Berlin introduced a technique he named evolution strategy, although this technique involved no crossover, or indeed multiple genomes in a population.

In 1975, the publication of the book “Adaptation in Natural and Artificial Systems” occurred, by the author John Holland, built on papers both by himself and by other researchers. This book presents the concepts of adaptive digital systems using mutation, selection and crossover, simulating processes of biological evolution.

Also in 1975 a dissertation by Kenneth De Jong established the potential of genetic algorithms by showing that they perform well on a wide variety of data, including noisy and discontinuous data.

By the early 1980’s genetic algorithms were being applied to a wide range of subjects. At first this application was largely theoretical; however genetic algorithms soon moved into commercial territory and nowadays, are being applied to many problems in many different areas (Marczyk, 2004).

4.3 - Advantages and Disadvantages of Genetic Algorithms

4.3.1 Advantages

The primary advantage of genetic algorithms is their inherent parallelism. A lot of other algorithms are largely serial and can only explore a search-tree so far in any one direction before abandoning all progress and starting again from the beginning, or further up the hierarchy.

Genetic algorithms effectively explore many different branches of the tree at once and when a certain branch turns out to not be optimal, abandon that search, proceeding with other more likely candidates.

This leads to a large proportion of the tree being searched, with processing power exponentially dedicated to more promising avenues as the search progresses.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

The real advantage of parallelism however is that by evaluating the relatively small search-space that it does, a genetic algorithm is implicitly evaluating a much larger group of individuals, in the same way that the average response of a relatively small percentage of the population of any given country, can be used to fairly accurately predict national trends.

This allows a genetic algorithm to move towards the search-space with the most promising individuals in a timely fashion and then select the best member of that group. This is known as “Schema Theorem” (Marczyk, 2004).

Thanks to the parallelism that is a genetic algorithms main advantage, they are very well suited as a means of exploring search space too large to be effectively searched in a reasonable amount of time by other, more conventional, methods. Mostly these problems are defined as being “non-linear”. In a linear problem, each component’s fitness is individual and any improvement to individual components fitness is necessarily an improvement to the whole. Many real life problems are not like this, they tend towards non-linearity where altering one component to be more fit may make other components less fit and cause a ripple effect across the whole system.

This non-linearity results in a huge increase in the search space to be navigated, making genetic algorithms an effective way to search them, due to their strength in navigating large search spaces (Marczyk, 2004).

A third advantage of genetic algorithms is that they do not tend to be easily trapped by local optima, due again to the parallelism of their approach, the random nature of the various starting points of the initial population and the other methods they employ, such as crossover and mutation.

Without crossover, selection and mutation a genetic algorithm is metaphorically similar to a large number of parallel depth first algorithms, each searching their own space for the best solution. Selection allows the pruning of the least promising searches, crossover allows promising solutions to share their success and mutation allows for certain random changes in the local search space of a given solution.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

In infinite or very large search spaces it is hard to know whether the global optima have been reached, or merely very good local optima. However genetic algorithms tend to give good results compared to other search strategies.

One of the biggest strengths of genetic algorithms can at first glance appear to be their biggest weakness, which is: genetic algorithms have no prior knowledge about the problem they are trying to solve, they produce random changes to their candidates and then use the objective function to determine whether those changes are overall positive or negative.

This allows them to discover solutions that other algorithms may have over-looked, or never contained in their search space in the first place. A good example of this is the concept of negative feedback, rediscovered by genetic algorithms, but denied a patent for several years because it ran counter to established beliefs. It did, however, work (Marczyk, 2004).

4.3.2 Disadvantages

One of the disadvantages of a genetic algorithm is the necessity of representing the problem the genetic algorithm is trying to solve, in a form that the genetic algorithm can use.

Generally genetic algorithms use strings of binary, integer, or real-valued numbers, with each number representing some distinct part of the solution (Marczyk, 2004).

Another method is to use genetic programming, where the actual code of the program forms the genomes in the genetic algorithm and it can swap portions in or out, or adjust them.

Another disadvantage of genetic algorithms is that the fitness function is absolutely crucial to the development of a suitable solution for the problem. If the fitness function is poorly written, the genetic algorithm may end up solving an entirely different problem from the one that was originally intended.

Another issue is setting the correct mutation rate, population size, etc. If the mutation is too high, the system will never converge towards a suitable solution, however if it's too low, a

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

comparatively smaller segment of the search space will be covered and the eventual solution may take longer to reach or never be reached as a result. Likewise if the size of the population is too low, too little of the search space may be investigated to find the best possible solution (Marczyk, 2004).

A further problem is that fitness functions can be deceptive. If the solutions to various problems are in areas of the search space that would not be considered likely by the fitness function then genetic algorithms (and most other search techniques) are no better than a random search for finding the solution.

For example in the diagram below, the global optima is at the far right, however that is only one point, there is no slope leading to it, therefore no gradual increase can lead you to it, you either have the solution or you don't. In this example a properly constructed genetic algorithm would be likely to settle on the marginally less optimal, but altogether easier to find, local optima in the centre of the diagram, unless by random chance it discovered the single point with the most optimum solution.

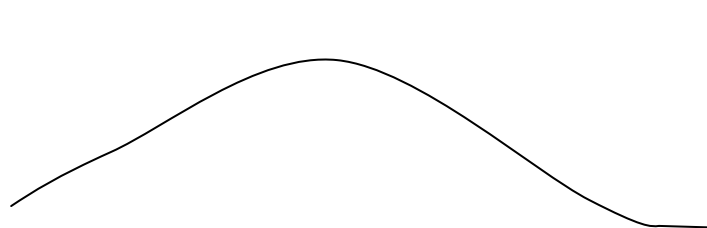


Figure 5 - Highly local optimums

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

A further problem is premature convergence: this occurs when a mutation early in the process produces a large leap in the fitness of that particular genome, which will then reproduce abundantly, lowering the diversity of the population and resulting in genetic algorithms possibly falling into the local optima that the mutation represents. There are various methods for solving this, including sigma scaling and Boltzmann selection (Mitchell, 1996).

Finally it is often recommended not to use genetic algorithms where analytical solutions exist. This is because analytical solutions, when they exist, usually produce more accurate results faster than a genetic algorithm is capable of (Marczyk, 2004).

5.0 – Training with Genetic Algorithms

5.1 – Determining weight values with genetic algorithms

Training neural networks can be thought of as problem solving, involving finding the optimum values for a set of real numbers (connection weights) which will produce the least error in the results from the neural network.

The error surfaces associated with these problems tend to be highly variable and contain many local optima, however the method of choice for training neural networks is usually back propagation (see section 2.0) which is a gradient descent algorithm and as such can be easily trapped in local optima (Branke, 1995). There are methods to try and avoid this problem when using back propagation, such as adding a “momentum” to the algorithm, to try to help it move beyond local optima; however these are not a complete solution (Blum, 1992).

Genetic algorithms on the other hand (as described in section 4.0) are good at avoiding getting stuck on local optima due to their searching several regions of search space simultaneously. Genetic algorithms also have the benefit that they do not place any restrictions whatsoever on the architecture of the neural network, since you’re merely

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

increasing the size of the genomes being worked with, not altering them besides that in any way.

In addition, the only information they require to train a neural network is an objective function as described in section 4.0, to test the appropriateness of a given genome.

5.1.1 - Representation of a neural network within a genetic algorithm

As a general rule, a neural network is represented within a Genetic algorithm, as a concatenation of all the weights in a string.

Since cross-over can disrupt genes far apart but is less likely to do so for genes far together, it can be helpful to keep functional units close together (I.e. place a neurons input and output weights side by side in the representation, place all the input weights of a node side by side and all nodes of a layer side by side).

One of the most important decisions when deciding how to represent a neural network within a genome is whether to use binary strings or real number strings.

Standard genetic algorithms use binary strings, however representing a large number of real valued numbers as a binary string, can lead to very large strings (thousands of bits) (Branke, 1995).

5.2 Using genetic algorithms to determine neural network structure

As well as using a genetic algorithm to evolve the weight set for a fixed-structure neural network, it is also possible to use a genetic algorithm to evolve a structure for a neural network.

The average neural network has an input layer, one or two hidden layers of an indeterminate number of nodes and one output layer. The input and output layer structure is dictated by the requirements of the problem at hand.

These are all connected, outputs to inputs, to form a feed forward network.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

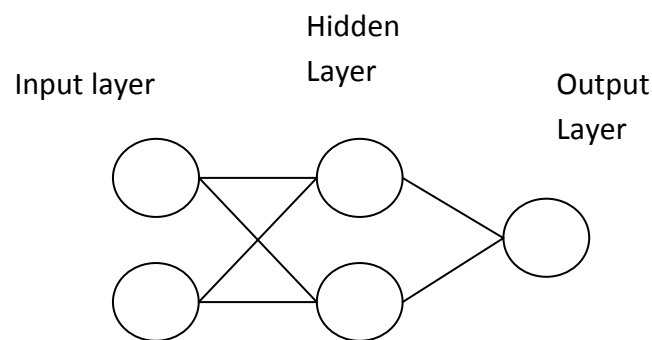


Figure 6 - A feed forward neural network

This architecture is then altered (number of nodes in each hidden layer, number of hidden layers) by trial and error and intuition to establish a neural network that performs well for the problem at hand.

However this is not necessarily the most efficient form for a network for any given problem and the trial and error method is certainly not the most efficient method for determining the correct architecture for an optimally performing neural network for the problem at hand.

If neural network architecture is too simplistic for the problem at hand, the neural network will never learn to solve the problem (or come close enough to suggest a solution).

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

On the other hand, if the neural network architecture is too complex, the neural network will learn fast but will generate too specific a solution and will not generalize between similar inputs. As mentioned in section 1.0, a neural networks generalisation is its main strength and not one to be given up.

Although network architecture plays such a large role in network performance, there is currently no method by which to establish, given a specific problem, a specific neural network topology to deal with that problem. Or even a method by which you can test, to see if your network topology is the most optimal solution for the problem at hand (Branke, 1995).

5.2.1 – Representing neural network structure

Representing the structure of a neural network within a genetic algorithm is not as straightforward as representing the weight values of a neural network within a genetic algorithm.

Ideally, a representation is required which can accommodate all networks that will work for the problem but none that will not, thus eliminating the searching of meaningless space and by definition, being able to contain the most optimum solution for the problem at hand.

There are basically two methods for representing neural network topologies within a genetic algorithm, the strong, direct, or low-level encoding and the weak, indirect, or high-level encoding (Branke, 1995).

Low-level encodings specify each connection individually, whereas high-level encodings group together connections and/or nodes.

In direct encoding only the connections are encoded into the genetic algorithm, thus meaning that in order to remove a node completely, you must remove all connections to or from that node completely.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

An exception would be made in cases where a genetic algorithm determines the number of hidden layers and the number of nodes per hidden layer and all the layers are assumed to be fully interconnected.

In one version of high level encoding of a neural network, the network is divided into areas and for each area the number of nodes and the density of connections to other nodes is determined (Branke, 1995).

Back propagation learning rate is also a parameter for one of these areas.

One of the reasons why alternatives to low-level encoding arose is the exponential rise in the number of connections to be mapped, as the size of the neural network increases. These mapping methods are therefore more suitable for small networks or networks with a small number of connections.

This could be useful, by limiting the potential size of the networks evolved and preventing overly large and complex networks from being generated but on the other hand, it's impossible to know whether these large and complex networks may be the most suitable way to solve the problem at hand.

As mentioned in section 5.2 neural networks that are too large will solve a problem fast but will not generalize well and therefore their usefulness is limited severely. Neural networks that are too small on the other hand, will not ever succeed in learning the problem. So a balance must be struck between speed of learning and generalisation exhibited in the trained network.

A problem with high level encoding is that regular networks are favoured but not every network is equally probable. This can lead to human bias entering the system and affecting the outcome, which is obviously undesirable.

This section largely drawn from information in: "Curved trajectory prediction using a self-organizing neural network" (Branke, 1995).

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

6.0 - Combining C# and C++

There are several methods for combining C++ code with managed C# code. The method I have chosen involves creating the C++ code as a dynamic link library, then using the code in the dynamic link library from within the C# user interface.

In order to do this you need to declare the functions you wish to export in C++ as such:

```
extern "C"
{
    __declspec(dllexport) int addition( int a, int b );
}
```

The “__declspec(dllexport)” before the functions in question (the functions to be exported) allows us to export these functions for usage by code outside the DLL, in much the same way as a “.def” file would, using more traditional methods.

This particular method of setting up a DLL and using it is Microsoft specific, however since I’m building a windows application, in Visual studio under windows, there’s no reason not to simplify things where using a Microsoft specific method does so.

Using the “extern “C”” as it has been, prevents C++ from mangling the functions names (to allow function over-loading) allowing them to be called as one would expect from within C#.

```
extern "C"
{
    __declspec(dllexport) int addition( int a, int b )
    {
        return( a + b );
    }
}
```

When you define your functions, you have to use the “__declspec(dllexport)” declaration once more, to ensure that they’re exported properly (Anonymous, 2007).

In C# in your application, you then need to import the functions as follows (DiLascia, 2002):

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

```
[DllImport("DLLTest.dll")]  
static extern int addition(int a, int b);
```

And then you can execute them as normal. Because calling functions in a dynamic link library will generate an exception if there's a problem, it can be good to use try and catch syntax, to prevent the program crashing and allow it to crash gracefully like so:

```
try {  
    int iadd = addition(5,6);  
    label2.Text = iadd.ToString();  
}  
catch (DllNotFoundException a) {  
    label2.Text = a.ToString();  
}  
catch (EntryPointNotFoundException a) {  
    label2.Text = a.ToString();  
}
```

When one of these two common exceptions is hit, this piece of code will catch the exception and note it in the text on label2. If all is as well, label2 should contain the text "11" as the answer to 5+6.

7.0 – Application of the Research

The research presented here is to be used to produce the deliverables for milestone 2 of my final year project: I plan to combine C# for the user interface and C++ for the neural network back-ends in order to create an efficient, powerful and fast solution.

Section 6.1 covers the combination of C# and C++ so I will not bother to cover it here, I plan to use the DLL method, with one or two DLL's linked into my C #user interface code.

The user interface will then have methods to adjust parameters such as the genetic algorithms population size, number of generations, etc. In order to test and see what kind of genetic algorithm learns and develops fastest. It will also allow you to specify the number of back-propagation iterations.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

This application will be put to use evaluating the difference in performance between a genetic algorithm and a neural network.

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

8.0 Bibliography

Anonymous. (2007). Retrieved 15 11, 2008, from http://www.codeguru.com/Cpp/Cpp/cpp_mfc/tutorials/article.php/c9855/

Artificial neural networks. (2008). Retrieved 06 11, 2008, from <http://www.learnartificialneuralnetworks.com/robotcontrol.html>

Barber, S. (2006). *AI: Neural network for beginners*. Retrieved 30 10, 2008, from Codeproject.com: http://www.codeproject.com/KB/recipes/NeuralNetwork_1.aspx ;
http://www.codeproject.com/KB/recipes/Backprop_ANN.aspx ;
http://www.codeproject.com/KB/cs/GA_ANN_XOR.aspx

Blum, A. (1992). *Neural Networks in C++*. Wiley Professional Computing.

Branke, J. (1995). *Curved trajectory prediction using a self-organizing neural network*. Retrieved 14 11, 2008, from <http://www.aifb.uni-karlsruhe.de/~jbr/Papers/GaNNOverview.ps.gz>

Darwin, C. (1859). *On the origin of the species by means of natural selection, or the preservation of favoured races in the struggle for life*. London: John Murray.

Davis, C. E. (2001). *Graphics processing unit computation of neural networks*. Retrieved 14 11, 2008, from UNM Computer Science: http://www.cs.unm.edu/~chris2d/papers/CED_thesis.pdf

DiLascia, P. (2002). Retrieved 15 11, 2008, from <http://msdn.microsoft.com/en-us/magazine/cc301501.aspx>

Marczyk, A. (2004). *Genetic algorithms and evolutionary computation*. Retrieved 10 11, 2008, from <http://www.talkorigins.org/faqs/genalg/genalg.html>

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

Marshall, J. A., & Srikanth, V. (1999). *Curved trajectory prediction using a self-organizing neural network*. Retrieved 14 11, 2008, from <http://www.cs.unc.edu/~marshall/WWW/PAPERS/curve9912.ps.gz>

Mitchell, M. (1996). *An introduction to genetic algorithms*. MIT Press.

Stergiou, C., & Dimitrios, S. (1997). *Neural Networks*. Retrieved 5 11, 2008, from http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Contents

Genetic Algorithms and Neural Networks

William Sayers

05025397

Supervised by Colin Morris

9.0 - Appendix - Libraries Investigated

- GALib
 - Matthew Wall
 - <http://lancet.mit.edu/ga/>
 - Accessed: 31/10/2008
- Libneural
 - D. Franklin
 - <http://iee.uow.edu.au/~daniel/software/libneural/>
 - Accessed: 31/10/2008
- Libann
 - Anonymous(1)
 - <http://www.nongnu.org/libann/index.html>
 - Accessed: 31/10/2008
- Flood
 - Roberto Lopez
 - <http://www.cimne.com/flood/>
 - Accessed: 31/10/2008
- Annie
 - Asim Shankar
 - <http://annie.sourceforge.net/>
 - Accessed: 31/10/2008
- FANN (Fast Artificial Neural Network Library)
 - Steffen Nissen
 - <http://leenissen.dk/fann/index.php>
 - Accessed: 31/10/2008