ABSTRACT

SNP AUTO-CALLING USING ARTIFICIAL NEURAL NETWORKS

by
Damien Spivak

In recent years feedforward artificial neural networks (ANN) and their training algorithms have become an effective methodology for the construction of nonlinear systems that solve the statistical problem of classification. The ability of ANNs to solve this problem is highly germane to making progress in the refinement of DNA microarray analysis and techniques regarding this issue. This study attempts to deal with the classification of microarray data and the comparison and validation of simple feedforward ANNs in partitioning high dimensional data. In doing this the efficacy of using ANNs as a genotyping tool will be proven. Furthermore, it has been determined through extensive testing that the classification abilities of simple feedforward ANNs are at least comparable with that of SVMs.

SNP AUTO-CALLING USING ARTIFICIAL NEURAL NETWORKS

by
Damien Spivak

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computational Biology

Department of Computer Science

August 2005

APPROVAL PAGE

SNP AUTO-CALLING USING ARTIFICIAL NEURAL NETWORKS

Damien Spivak

Dr. Qun Ma, Thesis Advisor — Date
Assistant Professor of Computer Science, NJIT

Dr. Frank Shih, Committee Member — Date
Professor of Computer Science, NJIT

Dr. Barry Cohen, Committee Member — Date
Assistant Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Damien Spivak

Degree: Master of Science

Date: August 2005

Undergraduate and Graduate Education:

- Master of Science in Computational Biology,
  New Jersey Institute of Technology, Newark, NJ, 2005

- Bachelor of Arts,
  McGill University, Montreal, Quebec, Canada, 2002

Major: Computational Biology

To Evelyn,
the love of my life and to our many adventures ahead.

ACKNOWLEDGMENT

I am indebted to many people throughout my education as a graduate student. First and foremost I must thank Dr. Qun "Marc" Ma for being the most capable advisor a student could have and whose help made this thesis possible. I would also like to thank Kai Zhang whose help in creating the GenoIterANN package was indispensable. Furthermore, I would like to thank Dr. Barry Cohen and Dr. Frank Shih for their insight and help in preparing this thesis for publication. Lastly, I would like to thank my wife Evelyn and my family for having the patience to stand by me through everything.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent decades the use of artificial neural networks (ANN) has experienced a renaissance. ANNs have progressed greatly since their first introduction with the advent of better training methodologies and better understanding of their capabilities. With this progress has come the possibility of using the neural network methodology to treat a variety of statistical problems that present themselves in many different fields. One problem that is of considerable importance to many fields is classification. This issue marks an important intersection between statistics and molecular biology that is readily applicable to the methodology of neural networks.

With respect to classification, this study tests the relative efficacy of using single hidden layer, feedforward ANNs for the purpose of determining genotype callings in single nucleotide polymorphism (SNP) microarrays. This feedforward ANN takes as input the logarithmic intensities of two different signals and respond with a classification of either homozygous or heterozygous. The performance of the ANN methodology is benchmarked against the results of both support vector machines as well as the manual choice of a linear cutoff.

CHAPTER 2

A PRIMER TO FUNCTIONAL GENOMICS

This chapter introduces selective, basic concepts of functional genomics. Functional genomics is the study of observing the pattern of gene expression of a given cell type at a given time. The ultimate goal of this study to determine a given pattern of gene expression can be assigned causation for an aspect of the current state of the cell. However, the pathways that map from current gene expression to the current state of the cell are not simple, but rather involve a complex set of reactions. These pathways have only recently become tractable with the application of recent breakthroughs in microarray technologies. Microarrays along with multiplex reverse transcriptase polymerase chain reaction (RT-PCR) have provided a highly efficient and cost effective technique for genome analysis on a large scale.

The computational problems of creating an automated system which classifies the signals created by microarrays in order to determine the current state of the cell will be defined within these notions. The discussion in this chapter will help to explain the underlying biology which is the context in which the feedforward ANN methodology has been used as a diagnostic tool.

2.1 Central Dogma of Molecular Biology

The central dogma of molecular biology states that the flow of genetic information in a cell is from deoxyribonucleic acid (DNA) to ribonucleic acid (RNA) to protein (Crick 1970). Although powerful in importance, it is a considerable over-simplification of the process by which gene expression takes place. In order to give the process slightly more

detail it may be divided into three major steps: transcription, translation, and replication. Though the last of these steps, replication is of extreme importance, it is out of the scope of this study and hence this review will be limited to the first and to a much lesser extent the second of these three steps. Alternatively, a post-transcriptional event known as splicing must be added to this study's review of the central dogma, as it is relevant to the justification of using the ANN methodology.

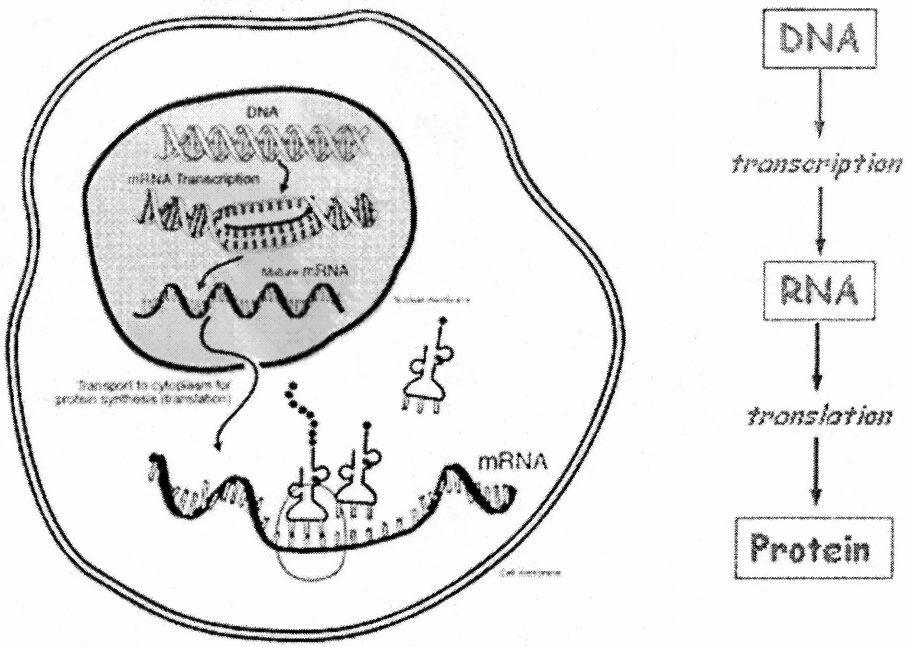

Figure 2.1 The central dogma of molecular biology (NCBI, 2005).

2.1.1 Transcription

Transcription is the process of making RNA from DNA. Essentially, both DNA and RNA are written in the same four letter alphabet of nucleotides: alanine (A), guanine (G), cytosine (C), and thymine (T) (uracil (U) in RNA) (Dickerson, 1983). As a result the

process of copying a gene from DNA to RNA simply transcribes the instructions for the production of protein (Brenner, 1961). In the eukaryotic cell DNA remains in the nucleus; polypeptide production, however, takes place in the cytoplasm. The instructions contained on the DNA molecule must be transcribed onto an RNA molecule, which moves out of the nucleus to serve as a template for protein production. This occurs through the operation of an enzyme known as RNA polymerase. Transcription does not change the language of the message being copied. Therefore the event of transcription does not add any complexity to the process of gene expression. However, an event that directly follows transcription known as splicing does (Weaver, 1961).

2.1.2 Posttranscriptional Phase: Splicing

In the late 1970's it was discovered that the DNA that composed genes, at least in eukaryotes, contained a larger number of nucleotides than the transcribed messenger RNA (mRNA) (Weaver, 2003). This, in turn, eventually led to the discovery that genes were composed of intervening sequences or introns which are not contained in the final mRNA sequence and expressed regions or exons which are contained in the final mRNA sequence (Sharp, 1994). Thus, the manner in which DNA gets transcribed to the final product of mRNA takes place in two steps. First, the gene is transcribed from DNA to an mRNA precursor known as heterogeneous RNA (hnRNA) which still contains intervening sequences. The hnRNA contains signals known as consensus sequences which determine where introns are to be spliced out. These signals allow hnRNA to form a complex with several proteins to become a structure that is known as the spliceosome. This spliceosome lines up the 5' and 3' ends of the preceding and following expressed

regions of an intervening sequence respectively so that they may form the final transcriptional product which is mRNA.

As an added complication, the implicit function which determines the mapping of genes composed of DNA to their directives composed of mRNA is not injective. Rather, approximately one in 20 genes may be spliced to form more than one mRNA sequence and may eventually be translated to form several different proteins.

2.2 Functional Genomics

The completion of the human genome project has been seen, by many, to mark the dawn of a new age in understanding for both the field of molecular biology and medicine. With this understanding come promises of a revolution in biotechnology. However, before any of these promises can come to fruition a number of hurdles must be surmounted. Foremost among these hurdles is the ability to interpret the function and behavior of the genes which compose the human genome (Weaver, 2003).

The area of research, known as functional genomics, is the field that has arisen to meet these challenges. One of the problems that functional genomics must solve is how to develop a cheap, high-throughput method for analyzing the gene expression of a cell at a given time. Some of the advancements in functional genomics, which have made inroads into solving these problems, are multiplex polymerase chain reaction (PCR), microarrays, and use of single nucleotide polymorphisms (SNPs) to determine the function of different genes (Hedge, 2000).

2.2.1 Polymerase Chain Reaction

PCR is a molecular cloning method which has greatly increased the ability to acquire samples of DNA from a cell (Weaver, 2003). The technique amplifies the amount of DNA found in a sample by using an enzyme known as DNA polymerase to create copies of selected regions of DNA. By creating a small segment of DNA (complementary to a region of interest), known as a primer, and adding it to a heated sample solution of DNA, it will create a duplex formation of primer and sample. The primer's function in this first step of PCR is twofold. First, it hybridizes to the sequence of interest, the region of the sample which is complementary to the primer. Second, it allows the DNA polymerase to begin creating a strand complementary to the region downstream of the primer binding site. After a single cycle of heating, and adding primer to a DNA sample solution, it is possible to double the quantity of a given region of DNA. Furthermore, this process can be repeated multiple times to achieve a desired yield as described by the formula below.

$$y = n \cdot 2^{x} \tag{2.1}$$

where n is the number of intial templates and x is the number of PCR cycles.

One of the many applications of PCR is to apply the technique to mRNA sequences. Often, it is useful to create a set of clones which represent one or more mRNA sequences present in a given cell at a given time. The name of this procedure is reverse transcriptase PCR (RT-PCR). The central part of RT-PCR is the synthesis of a complementary DNA, or cDNA, strand from an mRNA template. This synthesis is accomplished using the enzyme reverse transcriptase.

Another, more recent advancement in PCR technology is multiplexing, or the ability to amplify several fragments of DNA simultaneously. This technique is possible

by using more than one pair of primers in a standard PCR reaction. Typically, these multiplex PCR reactions are useful in genotyping applications where simultaneous analysis of multiple markers is required for the detection of pathogens. Unfortunately, multiplex assays can be tedious and time-consuming to establish, requiring lengthy optimization procedures in order to ensure accuracy.

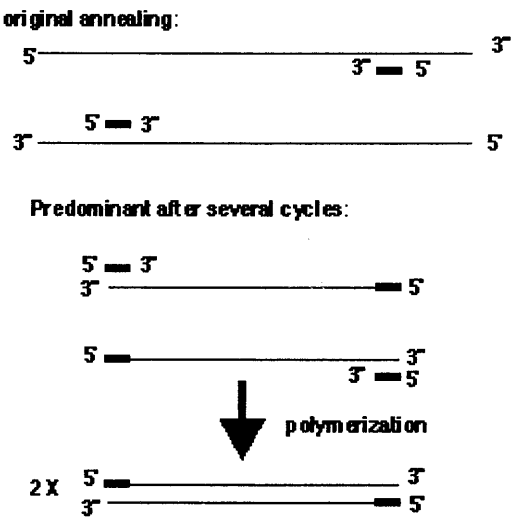

Figure 2.2 One cycle of a standard PCR reaction (NCBI, 2005).

2.2.2 Microarray Technology

In order to analyze the gene expression of a given cell at a given time, science has several methods which have been traditionally used in conjunction with the PCR reaction discussed in the previous section. Some these techniques include dot blots, direct sequencing or restriction fragment length polymorphism (RFLP) gel electrophoresis. However, these techniques often prove impractical due mainly to the desired size of many assays (Cui, 2002). Instead, a technology known as DNA microarray is used.

DNA microarrays are a technique in which a microarray is partitioned into several thousand cells. Each of these locations is printed with a small amount of cDNA which is

unique much like that of a dot blot. However, unlike a dot blot the spots of the DNA microarray are extremely small, only 100 – 150 μm, and the centers of the spots are only 200 – 250 μm apart (Weaver, 2003). This allows the size of an assay to be greatly reduced as well as permitting the running of several assays at one time. Once all the elements of an array have been spotted, the arrays are dried and the DNA sequences that have been placed at each location are covalently bonded by ultraviolet radiation to the surface of the chip (Hedge, 2000).

Once the preparation of a DNA microarray is complete, it is possible to run assays on the chip by spreading a sample solution of fluorescently labeled RNA across the surface of the chip. The labeled RNA in sample would then be able to hybridize to any DNA of complementary sequence. Once the RNA has been given adequate time to hybridize to the DNA on the chip, the chip is cleaned of any residue and can be analyzed.

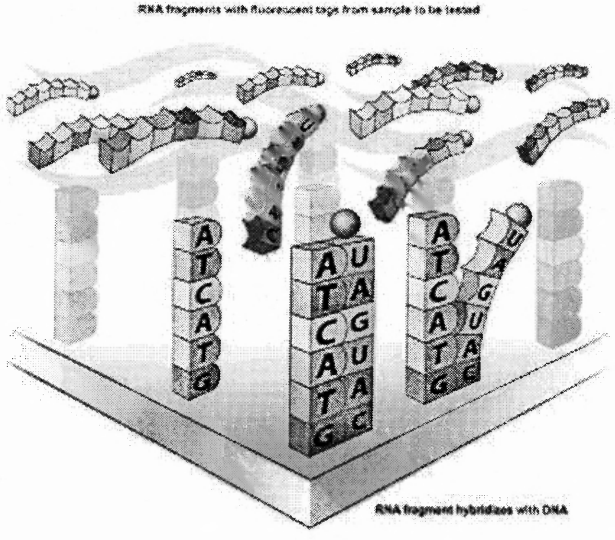

Figure 2.3 Enlarged region of a DNA microarray (NCBI, 2005).

The low level analysis of a DNA microarray is a reflection of the relative color intensities exhibited in a given cell of the array. In a simple experiment using DNA microarrays, two different fluorescent tags or channels may be used say in the colors red

and green. If microarray experiments were used to determine the signature gene expression of a particular phenotype for a cell, the experiment would designate one tag, say red, for the phenotype positive cells, and the other green, for phenotype negative cells. This then provides a simple way of determining what the gene expression pattern of a particular phenotype may be (Weaver, 2003). From a cursory glance an individual is able to see what genes are being expressed in the phenotype positive by those cells which are red and what genes are being expressed in the phenotype negative by those cells are green. If a particular cell is yellow then it can be surmised that to some extent both phenotypes express the given gene.

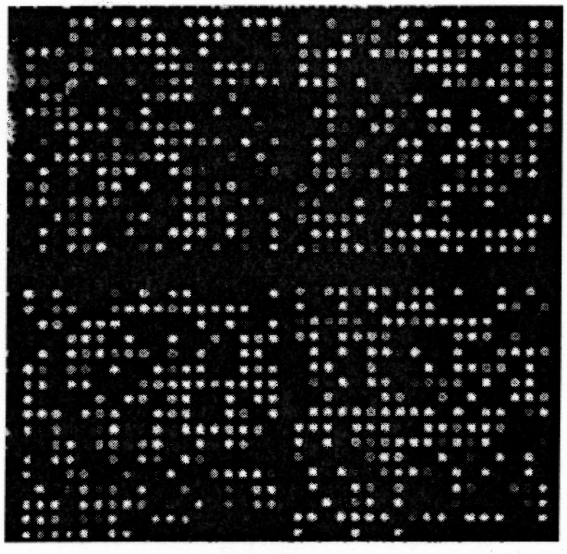

Figure 2.3 DNA microarray.

2.2.3 Single Nucleotide Polymorphisms

With the completion of the rough draft of the human genome sequence it is now possible to look for difference among individuals. These differences are often represented by single nucleotide polymorphisms (SNP). SNPs are DNA sequence variations that occur when a single nucleotide in a genome sequence is altered (Wang, 1998). For example, the sequence AAGGACT and the sequence ATGGACT could possibly represent the

sequences of an SNP. The majority of these SNPs have no effect on the functioning of a cell; however other SNPs are believed to be responsible for diseases, predisposition of diseases and responsiveness to particular drugs. By using technologies, such as the DNA microarrays previously discussed, it may now be possible to determine correlations between particular pathologies and signature gene expressions that are associated.

2.3 Determining Loss of Heterozygosity through Genotyping

Loss of heterozygosity (LOH), a concept initially described in 1971 by Knudson, is an indicator of tumor suppressor gene (TSG) inactivation in cancer cells (Knudson, 1971). LOH is the loss of one allele at a specific locus, caused by a mutation, or loss of a chromosome from a chromosome pair, resulting in abnormal hemizygosity. It can be detected by examining polymorphic markers in these regions. If heterozygous markers for a locus appear monomorphic because one of the alleles was deleted, then it can be assumed that LOH has occurred. When LOH occurs at a TSG where one of the alleles is abnormal, it results in a possible neoplastic transformation in of the cell.

However, cancer development is not necessarily as simple as the inactivation of one TSG. Rather it often involves TSG inactivation at several different loci. Therefore, the field of cancer research requires accurate genetic analysis techniques that detect LOH on a large scale. The development of this technique will permit the discovery of all TSGs which are affected by LOH and, allow a more comprehensive view of the disease cancer and its underlying mechanisms (Cui, 2002). Unfortunately, however, traditional methods of "one-marker one-assay" have been found to be largely inadequate for the goal of exhaustively identifying the TSGs. This is due to the fact that using these techniques on

the large scale required would necessitate large amounts of tissue sample deemed to be cancerous or pre-cancerous. Due to advances in biomedical diagnostic equipment, better screening techniques and preventative medicine the availability of these tissues are limited. Furthermore, the "one-marker one-assay" approach when used on large scale presents a problem with respect to both time and cost creating the need for a new innovative technique for determining genotype.

One possibility for this technique which averts the problems previously mentioned is to use the use the multiplex PCR reaction described in Section 2.2.1 in conjunction with a microarray assay described in Section 2.2.2. This allows at once resolves the problems associated with traditional methods by allowing the amplification and genotype testing for several genetic markers in one experiment.

CHAPTER 3

A PRIMER OF ARTIFICIAL NEURAL NETWORKS

This chapter will review the fundamentals of simple feedforward neural networks. In creating this review, the intention of this chapter is twofold. First, it will lay the groundwork for understanding the following chapter, which will formalize the problem of classification. Second, this review will serve to justify the choice of methodology which has been used in this study.

The description of artificial neural networks (ANN) can be divided into three major categories: network architecture, neuronal model, and optimization algorithm. This section will present a unified framework grounded in basic ANN theory. Subsequent discussions of ANN methodologies will be made in this context

3.1 Artificial Neural Networks

ANNs are systems which are formed out of many highly interconnected memoryless computing units (Bishop, 1995). This system can be represented as a weighted, directed graph in which the vertices are representative of the computing units and the edges are representative of the connections between these computing units, and the weights are the strengths of these connections. The pattern of interconnections that is formed from these edges and vertices is known as the architecture of the ANN (Figure 3.1).

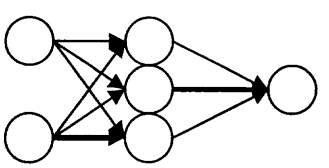

Figure 3.1 Topology of ANN as a weighted, fully connected, directed graph.

3.2 Network Architecture

It is convenient to think of an ANN as a series of layers. The computing elements that make up these layers often behave in a similar manner. The factors that guide this behavior are the activation function and the pattern of weighted connections in which it sends output and receives input. The uniformity in behavior is a result of the tendency for all elements in a given layer to contain the same activation function and for all elements in the same layer to have the same pattern of connections (Bishop, 1995). This may be observed in figure 3.1.

Most neural networks have an input layer where each unit is equal to an external input signal, an output layer from which the response of the ANN can be read, and possibly a number of hidden layers. Though all neural networks contain these key elements, the architecture of networks can vary greatly.

3.3 Computing Elements and Activation Functions

The fundamental unit of a neural network is the computing element. These computing elements are loosely based on the biological behavior of a neuron and thus are commonly referred to by that name (Sarle, 1997). These neurons relate the output value y to multiple inputs $\{x_1,...,x_d\}$, represented as a column vector $\underline{x}$. The extent to which each element of the column vector $\underline{x}$ contributes to the value of y is determined by a second column vector $\underline{w}$, or $\{w_1,...,w_d\}$. Thus, the processed input of any given neuron that is not part of the input layer is actually the dot product of the column vectors $\underline{w}$ and $\underline{x}$, represented as $\underline{x} \cdot \underline{w}$. The key determinate of the output value y, as previously mentioned,

is the activation function. As with the choice of network topology there are several activation functions that may be used (Fine, 1999).

The simplest of these functions is the identity function

$$f(x) = x \text{ for all } x. \tag{3.1}$$

This function is used for the computing elements in the input layer and occasionally in the output layer.

The second of these functions have a "thresh-holding" behavior and require a firing threshold τ and are commonly referred to as threshold functions. These functions are typically used in single-layer networks to convert net input, which is often continuous, to an output which is binary.

$$y = f\left(\sum_{1}^{d} w_i x_i - \tau\right). \tag{3.2}$$

Examples of functions which exhibits this "thresh-holding" behavior are the sign function

$$f(x) = sign(x - \tau) = \begin{cases} +1, x \geq \tau \\ -1, otherwise \end{cases}, \tag{3.3}$$

and the unit-step function

$$f(x) = U(x - \tau) = \begin{cases} 1, x \geq \tau \\ 0, otherwise \end{cases}. \tag{3.4}$$

Unfortunately, there are several limitations to the use of threshold functions in terms of what they are capable of computing, as well as their property of not being differentiable. Rather, sigmoid functions are often preferred for use in most neural networks (Fine, 1999). The two most common of these are the logistic function,

$$f(x)=\frac{1}{1+\exp(-x\sigma)}. \tag{3.4}$$

and the hyperbolic,

$$h(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}. \tag{3.5}$$

3.4 Optimization Algorithms

In addition to network architecture and choice of activation function the optimization algorithm is one of the major distinguishing characteristics of different neural networks. In general there are two categories in which most artificial neural networks may be placed: unsupervised and supervised.

Unsupervised networks are networks which attempt to group similar input vectors without the use of training data to instruct the network on what the typical output should be. The ANN however has its weights modified so that similar input vectors are assigned to the same output grouping or cluster (Anderson, 1997). Though unsupervised networks are a fascinating area of research, the focus of this study will based on those ANN which are considered to be supervised.

Supervised networks are perhaps the most common form of ANN. Supervised networks are defined by those networks which present a series of training input vectors, each with an associated target output vector.

$$T=\{(x_i,t_i),i=1:n\} \tag{3.6}$$

The weights of the network are then adjusted according to a learning algorithm. This is done in an effort to select a network η such that the output of the network $\underline{y}_i=\eta(\underline{x}_i,\underline{w}_i)$

approximates the desired output $\underline{t}_i$ for the input x_i. The metric for this approximation on the training set T is the sum squared error (SSE) function of the form

$$\varepsilon_T = \frac{1}{2}\sum_{i=1}^{n}\left\|y_i - t_i\right\|^2. \tag{3.7}$$

This metric is a function of the weight vector $\underline{w}$ since y_i is dependent on the parameters of the selected network η. There are other metrics which may be used to determine how well a selected network approximates the training set. However, due to its ease of optimization through calculus methods, the SSE function is preferred (Moller, 1993).

Hence, training an ANN is a nonlinear optimization problem in which the objective can summarized

$$\textbf{minimize } \varepsilon_T(\underline{w}) \textbf{ by choice of } \underline{w} \in W \subset \Re^p.$$

Unfortunately, the problem of finding minima is made difficult by the high dimensionality of most networks, where the dimensionality is a result of the number of the weight parameters of the network.

There are several learning algorithms that may be used to accomplish the task of adjusting the network's set of weights such as the Hebb rule, the perceptron learning rule and the delta rule for single layer networks (Anderson, 1997). There are also various backpropagation algorithms, such as the iterative gradient descent algorithm, and the scaled conjugant gradient algorithm, that are typically used in multi-layer neural networks. Of these several learning procedures, only the scaled conjugate gradient algorithms will be discussed in the context of this chapter.

3.4.1 Estimation of Error

The gradient decent methods determine the search directions and step sizes of the traversal of weight space using the information gained from the second order approximation given by

$$\varepsilon_T(\underline{w}) = \varepsilon_T(\underline{w}^0) + \underline{g}(\underline{w}^0)^T(\underline{w} - \underline{w}^0) + \frac{1}{2}(\underline{w} - \underline{w}^0)^T H(\underline{w}^0)(\underline{w} - \underline{w}^0) + o\left(\left\|\underline{w} - \underline{w}^0\right\|^2\right). \quad (3.8)$$

However, some gradient descent algorithms make the assumption of an approximating quadratic model given by

$$m(\underline{w}) = \varepsilon_T(\underline{w}^0) + \underline{g}(\underline{w}^0)^T(\underline{w} - \underline{w}^0) + \frac{1}{2}(\underline{w} - \underline{w}^0)^T H(\underline{w}^0)(\underline{w} - \underline{w}^0). \quad (3.9)$$

If this quadratic approximation about a given point $\underline{w}^0$ has a Hessian matrix which is positive definite, then it may be assumed that there is a unique minimum. Taking the gradient of this gives

$$\nabla m(\underline{w}) = g(\underline{w}^0) + H(\underline{w} - \underline{w}^0). \quad (3.10)$$

Setting the gradient to zero and solving for w^*, the minimizing weight configuration, it can be determined that

$$w^* = w^0 - H^{-1}\underline{g}. \quad (3.11)$$

With this 3.19 can be re-expressed to in terms of w^*

$$m(\underline{w}^*) = m(\underline{w}^0) + \frac{1}{2}\underline{g}(\underline{w}^0)^{\mathrm{T}} \mathrm{H}^{-1}\underline{g}(\underline{w}^0). \tag{3.12}$$

It is important to note that the calculation of the Hessian, though very useful in many cases, can be intractable (Fine, 1999).

3.4.2 Gradient Determination by Backpropagation

Backpropagation is an efficient technique for determining the gradient vector which is needed to implement many of the optimization algorithms (Anderson, 1997). It will be assumed that the application of the backpropagation algorithm as explained here will be exclusively used on a general network of simple feedforward topology using differentiable activation function and differentiable error function. The formula that will be within this section will be illustrated using a single hidden layer network with a layer of sigmoidal hidden units and a sum-squared activation function (3.7).

A feedforward neural network computes a weighted sum of its inputs in the form,

$$a_j = \sum_i w_{ji} z_i \tag{3.13}$$

where z_i is the activation of a unit, which sends a connection to unit j and w_{ji} is the weight associated with that connection. This weighted sum is calculated over all the units which are connected to unit j. The sum a_j is transformed by a non-linear activation function f to give the activation z_j of unit j in the form

$$z_j = f(a_j). \tag{3.14}$$

For clarity, it is important to note that if the z_i are input values in (3.8) then they will be denoted by x_i as well; if the value z_j is an output in (3.9) then it will be denoted y_k.

As part of the weight minimization process mentioned in the previous section, it is the effort of backpropagation to assign responsibility of error to weights in the network. It is convenient to relate the error function to the pattern which is being classified. Hence, it is important when using backpropagation to use error functions which can be written as a sum over all the patterns contained in the training set (Moller, 1993). Thus the equations of the form,

$$\varepsilon_T(\underline{w}) = \sum_1^n \varepsilon_m(\underline{w}), \text{ and} \tag{3.15}$$

$$\varepsilon_m(\underline{w}) = \frac{1}{2}(y_m - t_m)^2, \tag{3.16}$$

aid in the goal of determining the gradient of ε_T with respect to the weight of the network. Using (3.10) we can express these derivatives as sums over the training patterns of the derivatives for each pattern separately. Hence, for the rest of this section, the problem gradient determination will be for one pattern at a time.

Now consider the evaluation of the derivative of ε_m with respect to some weight w_{ij}. Note that ε_m depends on the weight w_{ji} only by the summed input a_j to unit j. Therefore, the chain rule can be applied to find the partial derivative to give

$$\frac{\partial \varepsilon_m}{\partial w_{ji}} = \frac{\partial \varepsilon_m}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial \varepsilon_m}{\partial a_j} z_i . \tag{3.17}$$

We supplement the notation with,

$$\delta_j \equiv \frac{\partial \varepsilon_m}{\partial a_j}, \tag{3.18}$$

where δ's are the rate of contribution to the error term ε_m by the excitation of a_j. This term is the error signal which is backpropagated from the output to the jth unit. Equation (3.12) can be re-expressed as

$$\frac{\partial \varepsilon_m}{\partial w_{ji}} = \delta_j z_i . \tag{3.19}$$

Equation (3.14) shows that the required derivative can be obtained by simply multiplying the value of z by the unit at the input end of the weight. As a result, in order to evaluate the derivatives, it is only necessary to calculate the value of δ_j for each hidden and output unit in the network, and then apply (3.14).

In calculating δ_k for the output units the evaluation is relatively simple. From the definition (3.13) we have

$$\delta_k \equiv \frac{\partial \varepsilon_m}{\partial a_k} = f'(a_k)\frac{\partial \varepsilon_m}{\partial y_k} \tag{3.20}$$

where (3.9) has been used with z_k is substituted by y_k.

In order to evaluate the δ's for hidden units, once again, the chain rule may be used to calculate the partial derivatives,

$$\delta_j \equiv \frac{\partial \varepsilon_m}{\partial a_j} = \sum_k \frac{\partial \varepsilon_m}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{3.21}$$

in which the sum is taken over all k to which j has connections. The units and connections are depicted in Figure 3.2.

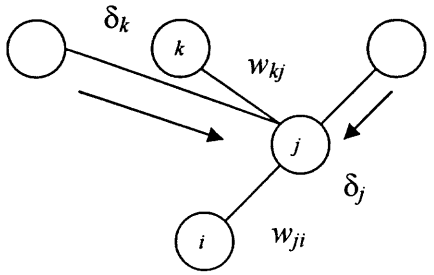

Figure 3.2 Illustration of the calculation of δ_j for the hidden unit j through backpropagation.

If definition (3.13) is substituted into equation (3.16) and applied to (3.8) and (3.9) the following backpropagation formula can be obtained

$$\delta_j = f'(a_j)\sum_k w_{kj}\delta_k \tag{3.22}$$

The backpropagation procedure for calculating ε_m be summarized as follows:

1. A forward pass of the training data, an input vector x^n, through the network to determine the node outputs using (3.8) and (3.9) to determine and the activation values of all hidden and output units.

2. Calculate the δ_k for the output units using (3.15).

3. Backpropagate the all δ's through (3.36) to obtain δ_j for units in the hidden layers of the network.

4. Using (3.14) to calculate all derivatives.

The derivative of the total error can then be calculated by using the summarized algorithm above for all patterns in the training set and then summing over all patterns using (3.10).

3.4.3 Generalized Optimization Strategy

The majority of optimization methods used to minimize the global error function are based on a common strategy (Fine, 1999). The minimization is given by a local iterative

gradient decent process. This process calculates an approximation of the error function for a given point in the weight space. This approximation is usually given by some first or second Taylor expansion of the function. This simple algorithm can be summarized as the following algorithm.

1. Choose initial weight vector $\underline{w}_1$ and set k=1.
2. Determine a search direction p_k and a step size α_k so that $\varepsilon(\underline{w}_k + \alpha_k) < \varepsilon(\underline{w}_k)$.
3. Update vector: $w_{k+1} = w_k + \alpha_k p_k$
4. If $\varepsilon(\underline{w}_k) \neq 0$ then set k = k + 1 and go to 2 else return $\underline{w}_{k+1}$ as the desired minimum.

To elaborate on this summarization, it useful to point out that this generalized algorithm involves two important independent steps. First, a search direction is determined to define the location of which the following point in weight space. Second, a decision is made on the distance traveled in this search direction to obtain the following point. The methods in which these two steps are calculated largely determine what type of algorithm will be used (Moller, 1993).

3.4.4 Conjugate Gradient Algorithm

In the conjugate gradient algorithm (CGA) the search direction is calculated recursively, as is the case with the generalized optimization strategy. However, the CGA has the added characteristic that each search direction is chosen so that it is non-interfering with any previously chosen search directions by using the second order information gained from the Hessian or estimations of it. This differs from other gradient descent algorithms which can undo progress made by previous choices in search direction with each newly calculated search direction. The search directions are selected s0 that each iteratively selected parameter value w_k, the current gradient g_k, is orthogonal to all previous search

directions $\underline{p}_1,\ldots,\underline{p}_{k-1}$. Therefore, at any given step of the algorithm, the value of the gradient (direction of steepest descent) is orthogonal to the linear subspace spanned by the previously calculated search directions (Kosko, 1992). As in the generalized algorithm, the new weight vector is calculated by

$$\underline{w}_{k+1} = w_k + \alpha_k \underline{p}_k = \underline{w}_0 + \sum_{i=0}^{k} \alpha_i \underline{p}_i . \tag{3.23}$$

This calculation is augmented with the optimal constraints on the search directions and weighting coefficients (α_i) such that the error function gradient ($\underline{g}_{k+1}$) calculated at the new location be orthogonal to all previous search directions,

$$(\forall \le k)\underline{g}_{k+1}^{\mathrm{T}} \underline{p}_i = 0 . \tag{3.24}$$

This set of search directions which conform to these constraints is also known as a conjugate system.

In order to understand how a conjugate system facilitates the locating of a local minimum on the error surface, it is helpful to consider the step from the starting point w_1 and the critical point w_* as being expressed as a linear combination $p_1,\ldots,p_n$

$$\underline{w}_* - \underline{w}_0 = \sum_{i=1}^{N} \alpha_i p_i . \tag{3.25}$$

By multiplying (3.25) by the term $p_j^{\mathrm{T}}\mathrm{H}$ and substituting g for $\mathrm{H}(\underline{w}_*-\underline{w}_0)$ using (3.10), evaluated at the critical point, gives

$$\begin{aligned} & p_j^T\left(-\underline{g} - \mathrm{H}w_0\right) = \alpha_j p_j^{\mathrm{T}} \mathrm{H} p_j \Rightarrow \\ & \alpha_j = \frac{p_j\left(-g - \mathrm{H}w_0\right)}{p_j^{\mathrm{T}}\mathrm{H}p_j} = \frac{-p_j^{\mathrm{T}} g}{p_j^{\mathrm{T}}\mathrm{H}p_j} \end{aligned} \tag{3.26}$$

It can be shown that by using (3.25) and (3.26) the local minimum can be found in N iterative steps, where N defines the number of dimensions in the weight space. In each of these iterative steps the intermediate points defined by $w_{k+1} = w_k + \alpha_k p_k$ are the minima for the quadratic error function $m(w)$ and are restricted to the k-plane π_k= $y_0+\alpha_1 p_1+\ldots+\alpha_k p_k$. In order to ensure that each of the search directions possesses the qualities of orthogonality and non-interference, it is necessary to introduce a scaling element to the newly defined search direction (Moller, 1993). The newly defined directions are defined by,

$$p_{k+1} = g_{k+1} + \beta_k p_k$$

$$\text{where } g_{k+1} = m'(\underline{w}_{k+}),\ \beta_k = \frac{|g_{k+1}|^2 - g_{k+1} g_k}{g_k^{\mathrm{T}} g_k} \tag{3.27}$$

With the elements provided by (3.23), (3.26) and (3.27) it is now possible to describe the CGA which will locate a critical point in a quadratic error surface. This algorithm proceeds as follows:

1. Select initial point $\underline{w}_0$, with values in the vector randomly chosen with high probability that they are small so that it does not saturate the network nodes.
2. Use backpropagation to evaluate the error function $E_0 = \varepsilon_T(\underline{w}_0)$ and the gradient of the error function $\underline{g}_0 = \nabla\varepsilon_T(\underline{w}_0)$.
3. Calculate the first search direction as $\underline{p}_0 = -\underline{g}_0$ and set the cycle index $k = 0$.
4. Calculate second order information: $s_k = \varepsilon_T''(\underline{w}_k)\underline{p}_k$, $\delta_k = p_k^{\mathrm{T}} s_k$.
5. Evaluate the step size α_k such that $\alpha_k = \frac{\mu_k}{s_k}$, where $\mu_k = p_k^{\mathrm{T}} g_k$
6. Update the weight vector: $w_{k+1} = w_k + \alpha_k p_k$, $g_k = -\varepsilon_T'(w_{k+1})$.
7. If k=1 equals the number dimensions N then restart the algorithm: $p_{k+1} = g_{k+1}$ else, create a new search direction:

   $\beta_k = \frac{|g_{k+1}|^2 - g_{k+1} g_k^{\mathrm{T}}}{\mu_k}$, $p_{k+1} = g_{k+1} + \beta_k p_k$.

8. If the new search direction $g_k \neq 0$ then set $k=k+1$ and go to step 2 else end and return the point w_{k+1}.

CHAPTER 4

DEVELOPMENT OF GenoIterANN

4.1 General Description of the Software

GenoIterANN is a menu driven software consisting of 62 files: 10 are directly involved in the creating the feedforward ANN, 20 are used in a supporting role for the network and the remaining 31 files are used to implement a support vector machine (SVM) classification system (the scope of this chapter will limit itself to the first files which pertain to creating the feedforward ANN and those files which support the network). These files have been created using the MATLAB software package. The feedforward network portion of the GenoIterANN software has been created as an open source toolbox which is implemented by GenoIterANN.

4.2 Basic Software Methodology

The impetus for the creation of the GenoIterANN software was a need for a rapid large-scale genotyping method. The specific genotyping example on which this software was based is the concept of loss of heterozygosity (LOH) as a means of creating an indicator for cancer detection (see section 2.2). The assay for determination of whether LOH has taken place is performed using a two-color SNP microarray in which there are three possible states: homozygous "C", homozygous "T", heterozygous "CT". Depending on the oligonucleotide sequence, there will be particular intensity signature that can be acquired by digitizing the fluorescence emitted from each location on the microarray assay. This intensity signature may be used to classify or "make a call" for a particular oligonucleotide sequence to determine its particular genotype. However, the ability to

make this system determine genotype relies on two sets of data. The first of these sets is a training set which samples the universal set of data and allows the network to learn the classification patterns of the universal set. The second of these data sets is the testing set which serves to validate and determine the accuracy of the network configuration which has been learned through training. It is on the basis of this very general summary which figure 4.1 describes and which will guide the remainder of this chapter.

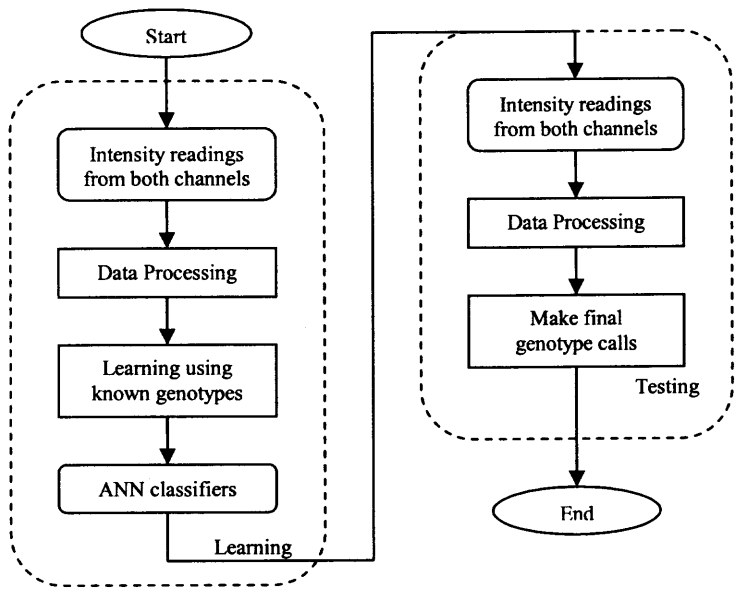

Figure 4.1 The basic methodologies for the GenoIterANN software

4.2.1 Data Preprocessing

At the beginning of the process described is the acquisition of the intensity readings from the aforementioned microarray assay. However, before the intensity readings may be used by the GenoIterANN software, the noise that is present in these readings must be removed. This removal is performed through noise subtraction and normalization. Noise subtraction is useful in eliminating background noise. On the other hand normalization is helpful to eliminate systematic error present in the signals that may arise from channel

arise from channel imbalance. Data preprocessing is particularly important in ensuring the success of data analysis particularly when data sets contain samples from multiple assays performed at different times. Data preprocessing ensures that all of the data points in a given data set have an equivalent weighting for analysis.

4.2.2 Automated Genotype-Calling

Once data preprocessing has been performed on the training set, the training set can be effectively used in training the feedforward ANN. The feedforward ANN uses the training set to develop the optimal classifiers that will make the most accurate calls on for a data point in a blind test. GenoIterANN does this by creating three independent feedforward ANNs with each network responsible for learning the signature associated with a particular genotype call. Each of these three networks will receive a matrix containing data points of 2 dimensions x and y with a particular genotype associated with each point. The procedures for how the feedforward ANNs use the training to learn is described in detail in chapter 3 and will be omitted from this chapter.

4.2.3 Iterative Refinement

Iterative refinement is based on the idea that the heterozygous data points should ideally have a linear best fit collinear to the equation $y - x = 0$. Unfortunately, however, this typically does not occur in most experiments. Rather, often is the case where the heterozygous data points lie unevenly about the line $y - x = 0$, since the intensities of red and green in spots of a microarray rarely have equal value. As a result, one of the features of the GenoIterANN software package is to adjust this imbalance by imposing an

artificial regularity constraint on the ANN classifiers [8]. This is achieved by applying linear regression analysis on the heterozygous data points following training. If the aforementioned constraint of congruence is not met, then it is assumed that there is systematic error in the training sample and the GenoIterANN software attempts to correct this error. The correction of the assumed systematic error in the data is accomplished by performing an iterative refinement process described by figure 4.2.

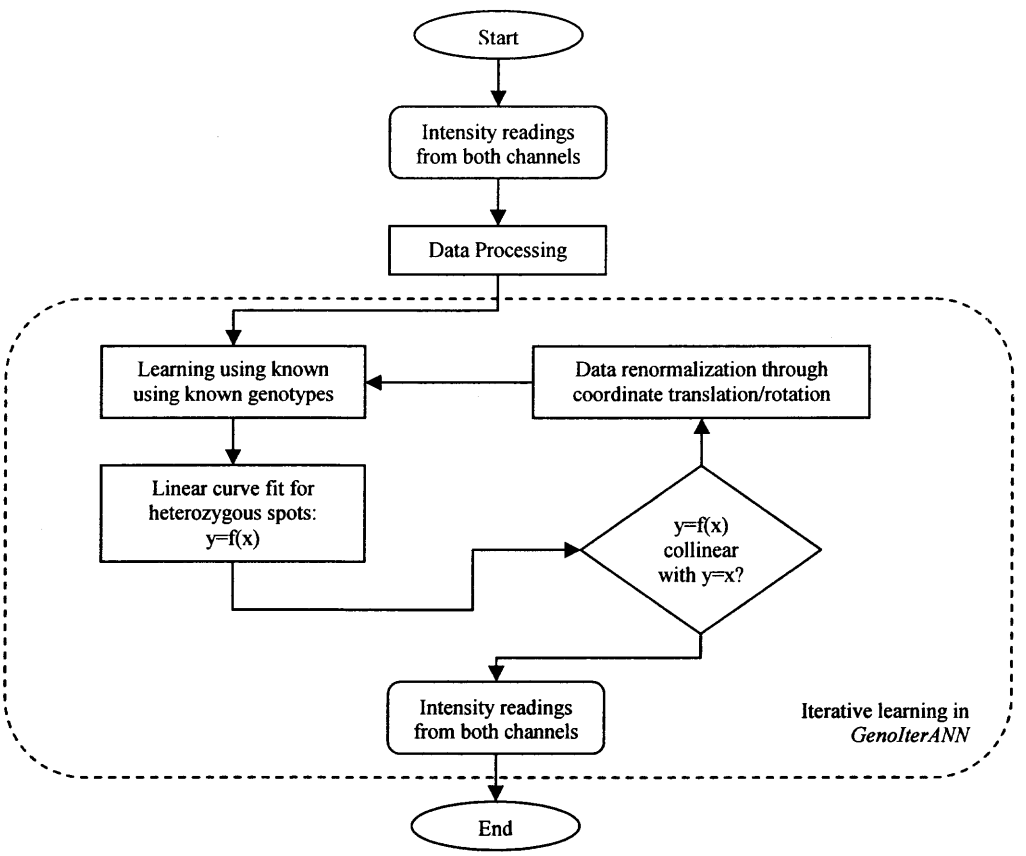

Figure 4.2 The methodology for GenoIterANN with iterative refinement [4].

4.3 GenoIterANN File Structure

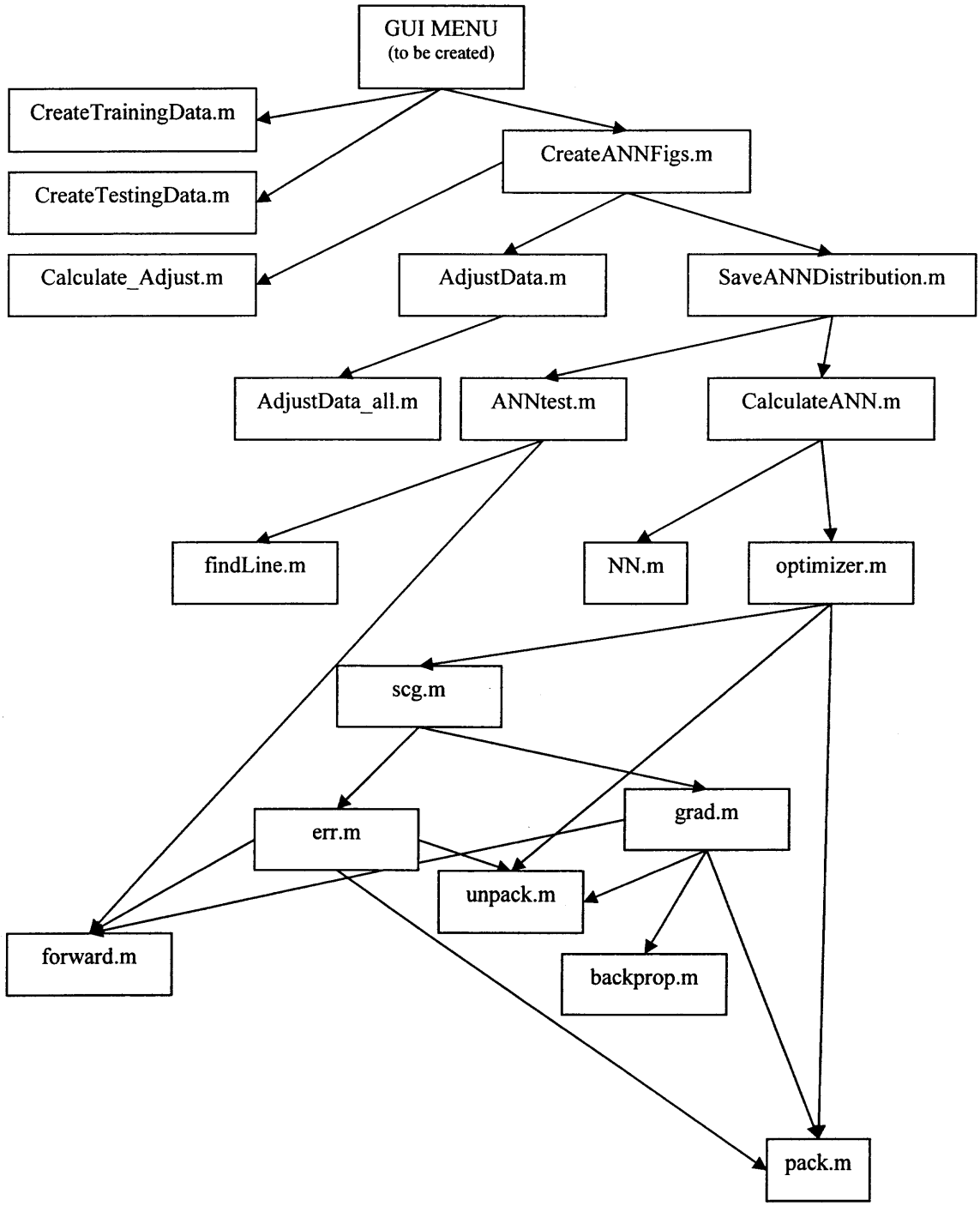

Figure 4.2 File structure for GenoIterANN

4.4 Software File Descriptions

4.4.1 *AdjustData.m*

Parameters:

C: intensity readings associated with a genotype call of homozygous "C"
T: intensity readings associated with a genotype call of homozygous "T"
H: intensity readings associated with a genotype call of heterozygous "CT"
p: rotation factor
p2: intercept point

Description:

This file is simply responsible for relaying each of the arrays containing intensity readings to the file *AdjustData_all.m* along with the rotation factor and the intercept point.

Files called: *AdjustData_all.m*

4.4.2 *AdjustData_all.m*

C: intensity readings associated with a genotype call of homozygous "C", "T", or "CT" depending on the parameter passed to the function *AdjustData_all.m*.
p: rotation factor
p2: intercept point for the first round

Parameters:

Description:

This file determines the

Files called: none

4.4.3 *ANNtest.m*

Parameters:

ANN1: FFANN trained to determine which intensity value are homozygous "C"
ANN2: FFANN trained to determine which intensity value are homozygous "T"
ANN3: FFANN trained to determine which intensity value are heterozygous "CT"

Description:

This file first makes a forward pass of the network to determine the calls for each of the data points in the testing file. Following this procedure the data points corresponding to homozygous "C", homozygous "T", and heterozygous "CT" are stored in stored in separate arrays. The arrays are then used to determine the slope and y-intercept of the lines which best partition the data points in the three arrays.

Files called: *forward.m*, *findLine.m*

4.4.4 *CalculateAdjust.m*

Parameters:

H: intensity readings associated with a genotype call of heterozygous "CT"

Description:

This file calculates the rotation factor and the intercept so that the heterozygous data points may be remapped so that the best fit of the heterozygous points are collinear with the line x = y.

Files called:

4.4.5 *CalculateANN.m*

Parameters:

C: intensity readings associated with a genotype call of homozygous "C"
T: intensity readings associated with a genotype call of homozygous "T"
H: intensity readings associated with a genotype call of heterozygous "CT"
nCTH: array containing the genotype calls where each element is 1, 2 or 3 corresponding to each of the possible genotype calls

method: unused parameter

Description:

This file creates three separate arrays that represent the known genotype calls for the training set: homozygous "C", homozygous "T", or heterozygous "CT". The size of each of these arrays is equal to the number of data points in the training set with each element being equal to either 1 or 0 with the same index as its respective intensity reading. In each of these arrays, a 1 signifies a positive call for genotype represented by the array and 0 a negative call. Along with these three arrays, a fourth array is created which contains all of the intensity readings.

The next step taken in the file is to set the configuration of the single hidden FFANN parameters. These parameters are the number of nodes in the input, hidden, and output layers, as well as the learning rate of the network. These parameters are then used to instantiate three independent FFANNs, each corresponding to the three different genotype calls. These three FFANNs are then subsequently trained with each of the FFANNs, taking as parameters the array containing the intensity readings, the array containing the genotype calls for which the particular FFANN is responsible for calling, as well as the number of cycles for which the FFANN will be trained.

Files called: *NN.m*, *optimizer.m*

4.4.6 *CreateANNfigs.m*

Parameters:

method: unused parameter

Description:

This file

Files called:

4.4.7 *backprop.m*

Parameters:

net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file
x: matrix of input vectors
z: matrix of hidden unit activations
deltas: matrix which estimates the gradient with respect to the error function of the output units

Description:

This file implements the standard backpropagation algorithm to determine the gradients of the standard error function (see 3.4.2).

Files called: none

4.4.8 *err.m*

Parameters:

w: matrix containing the weights of the FFANN
net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file
x: matrix containing all input values
t: matrix containing all target values

Description:

This file evaluates the error function.

Files called: *pack.m*, *unpack.m*

4.4.9 *forward.m*

Parameters:

net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file

x: matrix containing all input values

Description:

This file serves to propagate the values in the matrix ***x*** to determine the output of the FFANN for a given weight configuration.

Files called: none

4.4.10 *grad.m*

Parameters:

w: matrix containing weight and bias values
net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file
x: matrix containing all input values
t: matrix containing all target values

Description:

This file takes the weight configuration of the FFAN described by the object "net", the values in the input matrix "x", and the target matrix "t" with respect to the error function described by the err.m file. The gradient determination is performed using the backpropagation algorithm described by the file *backprop.m*.

Files called: *backprop.m, forward.m, pack.m, unpack.m*

4.4.11 *NN.m*

Parameters:

nin: number of input nodes in the FFANN
nhidden: number of hidden nodes in the FFANN
nout: number of output nodes in the FFANN

outfunc: type of output node used in the network
prior: corresponding to a zero-mean isotropic Gaussian prior with inverse variance with value PRIOR

Description:

This file creates a data structure which stores all the elements necessary to adequately describe an FFANN such as number of input nodes, number of hidden nodes, number of output nodes, type of output function, values of weights connecting layers, biases, and a prior gaussian.

Files called: none

4.4.12 *optimizer.m*

Parameters:

net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file
options: a vector which contains values which direct several procedures in the GenoIterANN software such as the number of cycles of training, toggle for displays, etc
x: matrix containing all input values
t: matrix containing all target values

Description:

This file calls the error minimizing function scaled, conjugate gradient, and defines which error metric will be used by this algorithm.

Files called: *pak.m, unpack.m, scg.m*

4.4.13 *pack.m*

Parameters:

net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file

Description:

This file simply places the weight and bias values from the net data structure and into a row vector so that it may be easier to apply mathematical operations to the weight and bias values.

Files called: none

4.4.14 *scg.m*

Parameters:

f: error function

x: weight vector

options:

gradf: defines chosen error function

Description:

This function implements the scaled conjugate gradient algorithm (see Section 3.4.4).

Files called: *pack.m*, *unpack.m*

4.4.15 *unpack.m*

Parameters:

net: data structure which contains all the information relevant to the FFANN and created by the *NN.m* file

w: matrix containing weight and bias values

Description:

This file updates the net data structure with the current values of the matrix containing the weights and biases.

Files called: none

CHAPTER 5

TESTING

The testing of GenoIterANN was performed using two sets of data. The first testing was performed using two artificial data sets, while the second set of data was performed using microarray data used in (Ma et al., 2004) provided by Dr. Honghua Li at the Cancer Institute of New Jersey.

5.1 Artificial Data Set

For the testing of GenoIterANN with artificial data sets, two sets of data files were created. The first of these files was a file containing training data which consisted of three columns of data. The second of these was a file containing testing data which consisted of three columns of data. For both of these files, the data points defined in the first two columns was generated by randomly choosing values between a given maximum and minimum values. A brief description of the file is provided below.

- **Artificialtrain1.txt**: Contains two columns of x and y coordinates and a third column with label called by a known cut-off function. This cut-off function is defined as y = 2 hence, $f(y)=\begin{cases} y \geq 2...1 \\ y < 2...0 \end{cases}$.
- **Artificialtrain2.txt:** Contains two columns of x and y coordinates and a third column with label called by a known cut-off function. This cut-off function is defined as $(x-3)^2+(y-3)^2=2$ hence, $f(x,y)=\begin{cases} (x-3)^2+(x-3)^2 \leq 2...1 \\ (x-3)^2+(x-3)^2 > 2...0 \end{cases}$.

For each of the data sets, the testing procedure used is as follows:

1. Train network with training file artificialtrain.txt and save the resulting network.
2. Using the trained network perform classification using the first two columns of the testing file artificialtest.txt.
3. Comparison of the labels predefined by the third column in the artificialtest.txt with the labels called by network.

5.1.1 Classification with Artificialtrain1.txt

File artificialtraining.txt consists of 200 data points which can be linearly separated. The purpose of the test is to verify the ANN training model created by GenotIterANN. The training with artificialtraining.txt file was done using the parameter values and a summary of the training result is provided in the Table 5.1.

Table 5.1 Summary of Network Characteristics used with Artificialtrain.txt.

Number of Input Nodes	2
Number of Hidden Nodes	9
Output Node type	Linear
Number of Iterations	300
Prior	0.01

Table 5.2 Summary of Artificialtrain1.txt

Number of data points	100
Number of red(1) labeled data points	20
Number of green(2) labeled data points	80

The plot generated by the training was used to classify the data points in artificialtrain1.txt. Figure 5.1 below shows the data points with the true calls being

observed from the color of the points. The red triangular labeled data points being classified as one and the green diamond labeled data points being classified as zero. Furthermore, the results of training the neural network can be observed by viewing the background of the plot. The background demonstrates a mapping of the function learned by the neural network. The range of this implicit function is depicted by all regions of the plot which is colored blue being mapped to the value of zero, and all regions of the plot which is colored gray being mapped to the value of one.

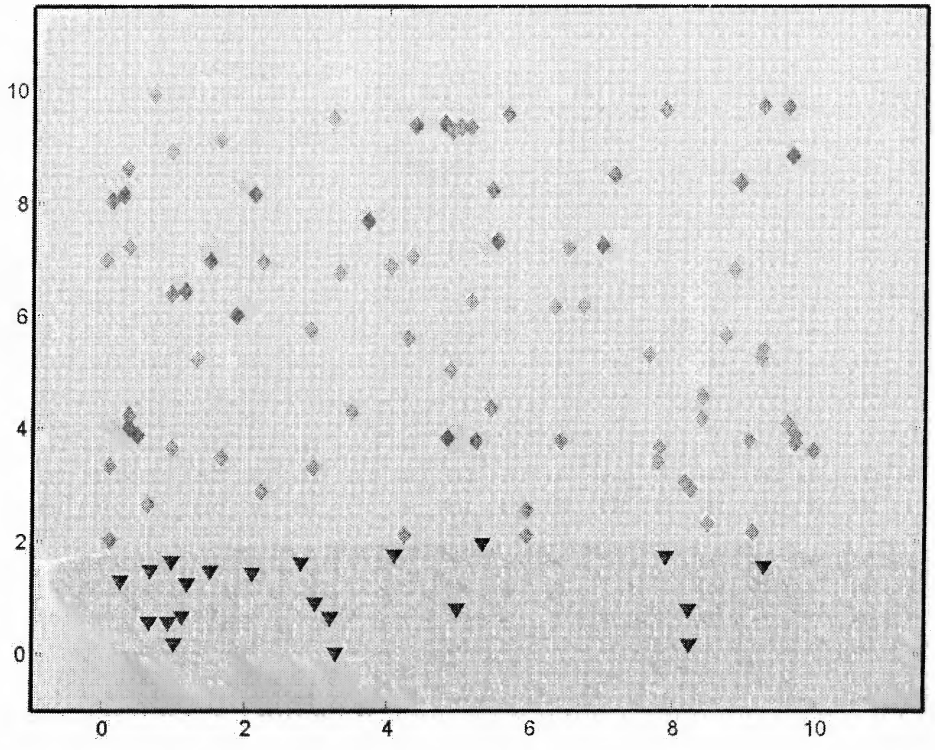

Figure 5.1 Data points with true calls plotted, from input file artificialtrain1.txt.

Table 5.3 Summary of Neural Network Training Results Using Artificialtrain1.txt

Number of correct calls:	100
Number of incorrect calls:	0
Accuracy:	1.00%(100/100)
Sum Squared Error:	.5354

5.1.2 Classification with artificialtrain2.txt

File artificialtraining.txt consists of 1000 data points which can be linearly separated. The training with artificialtraining.txt file was done using the parameter values and a summary of the training result is provided in the Table 5.1.

Table 5.4 Summary of the Network Characteristics with Artificialtrain2.txt

Number of Input Nodes	2
Number of Hidden Nodes	10
Output Node type	Linear
Number of Cycles	2500
Prior	0.01

Table 5.5 Summary of Artificialtrain2.txt

Number of data points	1000
Number of red(1) labeled data points	60
Number of green(2) labeled data points	940

The plot generated by the training was used to classify the data points in artificialtrain1.txt. Figure 5.1 below shows the data points with the true calls being observed from the color of the points. The red triangular labeled data points being classified as one and the green diamond labeled data points being classified as zero. Furthermore, the results of training the neural network can be observed by viewing the background of the plot. The background demonstrates a mapping of the function learned

by the neural network. The range of this implicit function is depicted by all regions of the plot which is colored blue being mapped to the value of zero and all regions of the plot which is colored gray being mapped to the value of one.

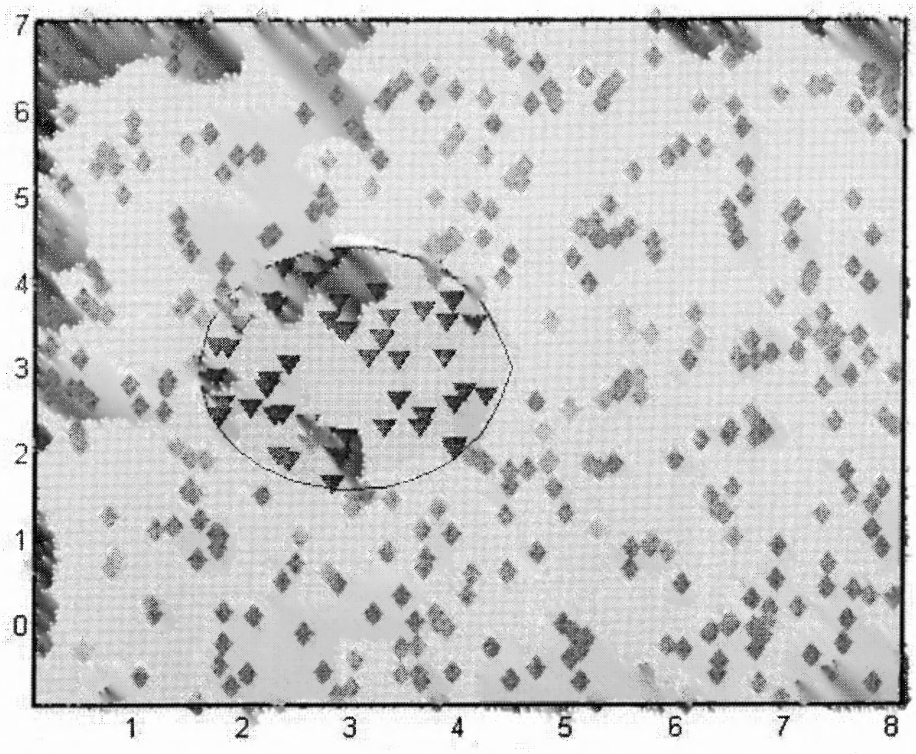

Figure 5.2 Data points with true calls plotted, from input file artificialtrain2.txt.

Table 5.6 Summary of neural network training results using artificialtrain2.txt.

Number of correct calls:	996
Number of incorrect calls	4
Accuracy:	0.996%(996/1000)
Sum Squared Error:	.6233

5.2 Microarray Data Set

The second set of data which was used in this study was done using microarray data. As microarray data was the impetus for the development of the GenoIterANN software it is appropriate that testing was done on a sample of microarray data.

5.2.1 Description of Data

The preparation of the microarray data was performed by extracting raw data from two files, train.txt and test.txt containing 2,447 and 4,439 data points respectively. Both of these files are used to create two data structures named TrainingData.mat and TestingData.mat. These data structures each contain nine fields. These data structures are summarized in table 5.7 and 5.8.

Table 5.7 TrainingData.mat data structure

Name	Description
C	Array<2x888> data points associated with homozygous "C" calls
T	Array<2x770> data points associated with homozygous "T" calls
H	Array<2x634> data points associated with homozygous "CT" calls
L	Array<2x136>
U	Array<2x19>
nCTH	Array<1x3> number of calls for C, T and H

5.2.2 Preparation of Data

The raw data that is collected from microarray experiments is given by two values. These values are determined through image processing, where the first value and second value are measure of the mean green intensity and mean red intensity values of a cell of a DNA microarray. As may be observed in Tables 5.7 and 5.8, the raw data is processed in

two steps. The first of these steps is applied by subtracting the background noise from the raw data to create a normalized set of values. In the second step of processing the data, the natural logarithm is applied to the normalized data. As a final step in the preprocessing of the data, all low intensity values are removed. As a result of preprocessing, the number of data points is changed to 2,293 and 4,419 for the training and testing, respectively. This processed data is then used directly by GenoIterANN.

5.2.2 Training of Classifiers

The GenoIterANN program, as previously described, goes through two phases. The first of these phases is the training phase in which the neural network learns to make accurate genotype callings using data which contains mapping of input values to targets which are known to be correct. However, in GenoIterANN this training process is modified. After training there is a curve fitting procedure which attempts to make the best fit of the heterozygous labeled data points. The GenoIterANN then attempts to correct the systematic error that is present in the data using the system of canonical classifiers. Following this, there is a validation process in which the trained neural network attempts a blind test with data where the targets are unknown.

Table 5.8 Summary of the network characteristics used with TrainingData.mat.

Number of Input Nodes	2
Number of Hidden Nodes	9
Output Node type	Linear
Number of Iterations	300
Prior	0.01

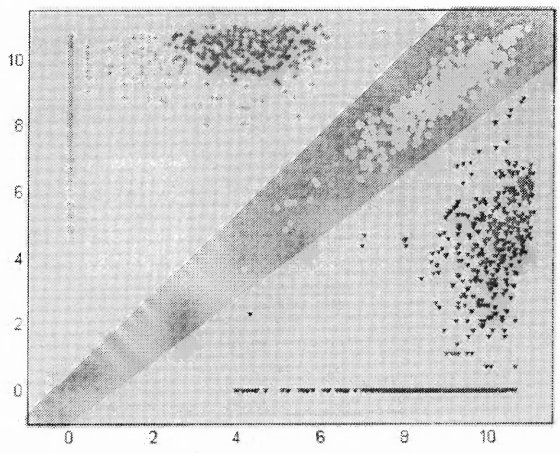

Figure 5.3 Training data with known genotypes, no iterations.

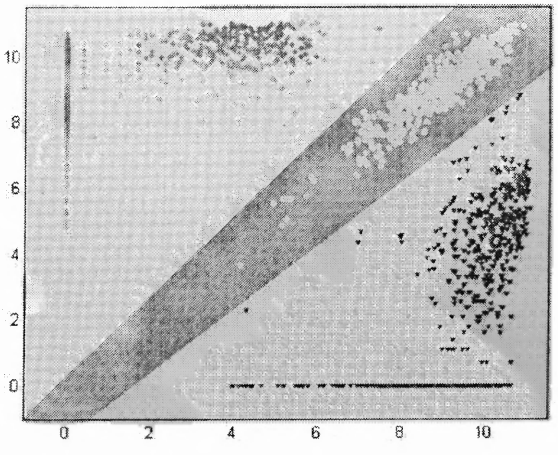

Figure 5.4 Training data with known genotype, one iteration.

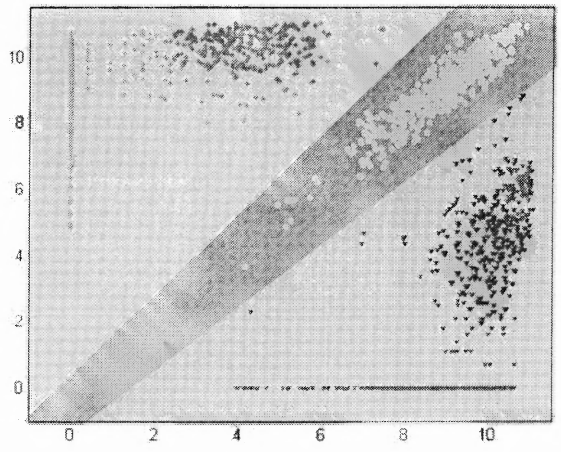

Figure 5.5 Training data with known genotypes, two iterations.

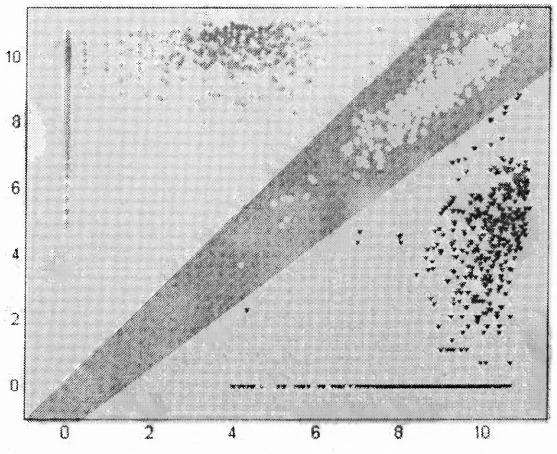

Figure 5.6 Training data with known genotypes, three iterations.

5.2.3 Testing with Variation of Internal Nodes

In the following section the study attempts to experiment with variation in the internal architecture of the feedforward ANN in GenoIterANN by using the data acquired from TrainingData.mat while altering the number of nodes in the hidden layer of the network.

Table 5.9 Summary of Results While Varying Nodes in the Hidden Layer

Number of nodes in hidden layer	**Line equations generated** $y = a_1x + a_2$
Homozygous "C"/Heterozygous "H" partition with 3 nodes	$a_1 = 1.2300$ $a_2 = 0.3400$
Heterozygous "H"/Homozygous "T" partition with 3 nodes	$a_1 = 0.9595$ $a_2 = -1.2848$
Homozygous "C"/Heterozygous "H" partition with 5 nodes	$a_1 = 1.2300$ $a_2 = 0.3500$
Heterozygous "H"/Homozygous "T" partition with 5 nodes	$a_1 = 0.9595$ $a_2 = -1.2800$
Homozygous "C"/Heterozygous "H" partition with 10 nodes	$a_1 = 1.2450$ $a_2 = -0.0979$
Heterozygous "H"/Homozygous "T" partition with 10 nodes	$a_1 = 0.9367$ $a_2 = -1.1832$
Homozygous "C"/Heterozygous "H" partition with 20 nodes	$a_1 = 1.2117$ $a_2 = 0.1089$
Heterozygous "H"/Homozygous "T" partition with 20 nodes	$a_1 = 0.9262$ $a_2 = -1.0679$

As a result of investigating the impact of variation of hidden nodes on the accuracy of GenoIterANN it may be concluded that varying the nodes has limited impact on the quality of the trained classifiers final calls. However, it maybe noted that in cursory tests using a less than ideal data set for validation it was found that the ANN classifiers worked best while using approximately 10 nodes in the hidden layer and one iterative refinement of the data set.

5.2.4 Concordance Rate

Under ideal circumstances, validation of the feed forward ANN methodology at the heart of GenoIterANN would be performed using data obtained from other independent

methods of genotyping such as direct sequencing or RFLP. However, this is impractical due to the time requirements imposed by these methods. Instead, in order to validate the accuracy of the classifiers we take advantage of the fact that classifiers should have the ability to genotype in both the sense and anti-sense directions. This ability provides the possibility of measuring accuracy by observing the concordance rate (Zhang, 2005). The concordance rate is defined as

$$R_{concord} = \frac{|A \cap B|}{|A|} \tag{5.1}$$

in which A and B represent the sense and anti-sense direction respectively.

In order to determine a concordance rate for GenoIterANN with minimal error two microarray data sets were used containing the same SNPs. However, the oligo-nucleotides contained on each microarray which correspond to each of SNPs being tested for are presented with opposing orientations. In an effort to make this measurement as accurate as possible, replicates of each SNP as well as other standard microarray experimental error minimization techniques are included in the two assays. This ensures that any experimental error that is may take place while collecting the data is minimized.

With two microarray assays conforming to this description the data that is extracted may be used in GenoIterANN procedure as testing data. The output of GenoIterANN using this testing data will provide the blind test necessary for validation of the GenoIterANN methodology. However, before this output can be used any conflicting calls among replicate SNPs must be resolved. This resolution is executed by using a rule which eliminates statistical outliers in the data while not completely discarding all data that may be obtained from conflicting calls. This rule is simply that any two data points which possess calls which lie in ranges which are estimated to be

contiguous (Figure 5.6, solid line) by the classifiers are averaged, while any that are not contiguous are discarded (Figure 5.6, dashed line). Once the usable data points have been extracted equation 5.1 maybe used to obtain the concordance rate.

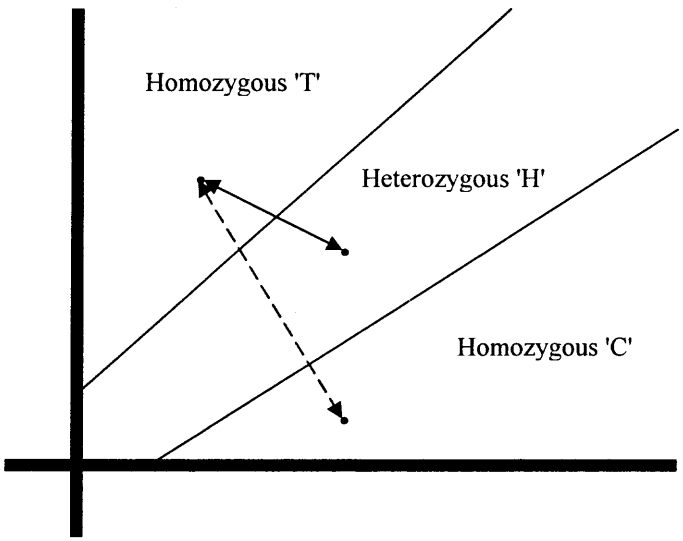

Figure 5.7 Rule for determining data point conflicts

In this study, two data sets were obtained from microarrays matching the specifications described (table 5.12). These data sets where then used in a blind test validation of the GenoIterANN procedure. The result of this validation test was a value of 88.5% concordance rate. The plot of the sense and anti-sense data points plotted over the classifier mappings is seen in Figure 5.7 and 5.8.

Table 5.10 Description of sense and anti-sense oriented validation sets.

Name	Description
BG_red	Red intensity values with background interference<1x1152>
BG-green	Green intensity values with background interference<1x1152>
numberTest	Number of data points <int>
Red	Red intensity values with interference subtracted <1x1152>
Green	Green intensity values with interference subtracted <1x1152>

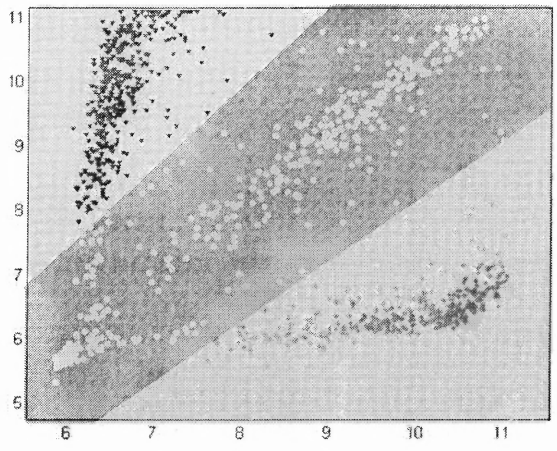

Figure 5.8 Concordance testing using test data with sense orientation.

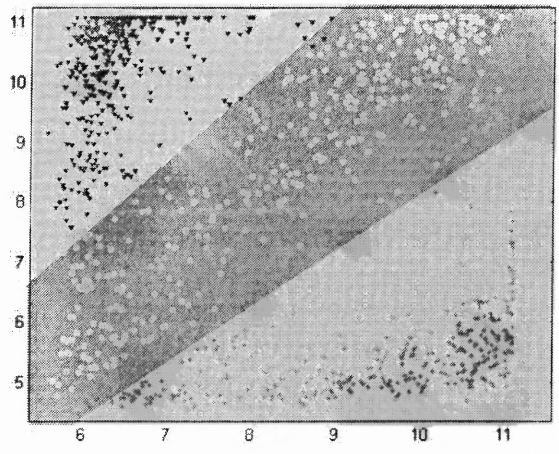

Figure 5.9 Concordance testing with anti-sense orientation.

CHAPTER 6

CONCLUSION

Using the high-throughput SNP genotyping methods, known as GenoIterANN, this study analyzed 4,600 SNPs covering 12 chromosomes in a minimum of 24 genomic were used (Zhang, 2005). These 4,600 SNPs were then classified by the GenoIterANN to determine the feasibility of an SNP auto-calling using an artificial neural network. In testing this feasibility it was requisite that a reasonable method for validating the results be determined.

Although it is ideal for SNP calling to be validated using methods such as RFLP and direct sequencing; it was not and often is not practical to use these methods. Most often it is because of time and financial constraints. Rather, in this study an approach for validating classification accuracy by measuring the concordance rate was introduced. Using this method it was found that the GenoIterANN had a concordance rate of approximately 98 percent.

In comparing the GenoIterANN algorithm to other competing methods, such as linear-cutoff and Support Vector Machine (SVM), it can be shown that GenoIterANN is a modest improvement over linear-cutoff and comparable to the accuracy of SVM. Thus, it has been demonstrated that the GenoIterANN algorithm is a simple, accurate and relatively efficient procedure for use as a practical SNP auto-calling method.

Further discussion of these results brings us to dealing with minimal sample sizes. If the number of heterozygous SNPs in a given sample is too small, data renormalization may be significantly skewed during the iterative genotype-calling procedure. This is

caused because the linear curve fit of points from a small sample can interfere with the classifiers ability to discern the true image of difference in the two channels. In order to avoid this problem, one may acquire data from several different microarray assays and process this data collectively rather than processing each assay one at a time. The technique was the actual method adopted in our of GenoIterANN's efficacy.

Also, further investigation of many of the SNPs which were erroneously classified by GenoIterANN has shown that about 60 data points are far from the decision boundaries. Most likely these data points are mistakes which have been caused by experimental error in the laboratory. If these 60 data points are omitted from our testing the concordance rate of GenoIterANN reaches over 99 percent.

In terms of future progress concerning the GenoIterANN algorithm, it is hoped that inroads will be made in both the convergence time required for training, and further minimization of incorrect calls made by the procedure. As a suggestion for solving the problems of convergence time, it may behoove future research to experiment using different error minimization algorithms as well as architectural pruning.

REFERENCES

1. Anderson, J.A. (1997). An Introduction to Neural Networks. Cambridge, MA: The MIT Press.

2. Bishop, C. (1995). Neural Networks for Pattern Recognition. New York, NY: Oxford University Press.

3. Bair,E., Tibshirani,R. (2004). "Machine learning methods applied to DNA microarray data can improve the diagnosis of cancer", *SIGKDD* Explorations, 5(2): 48-55.

4. Brenner, S., Jacob F., Meselson M. (1961). "An unstable intermediate carrying information from genes to ribosomes for protein synthesis". Nature 190:576-581.

5. Crick F. (1970). Central dogma of molecular biology, Nature, 227:561-563.

6. Cheng, B. and Titterington, D.M. (1994). "Neural Networks: A Review from a Statistical Perspective". Statistical Science, 9, 2-54.

7. Cui, X., Feiner, H., Li, H. (2002). "Multiplex Loss of Heterozygosity Analysis by Using Single or Very Few Cells", *Journal of Molecular Diagnostics*, 4(3):172-177.

8. Dickerson, R.E. (1983). "The DNA helix and how it reads". *Scientific American* 249(12): 94-111.

9. Fine, T.L. (1999). Feedforward neural network methodology. New York, NY: Springer.

10. Hegde,P., Qi,R. Abernathy,K., Gay,C., Dharap,S., Gaspard,R., Hughes,J.E., Snesrud,E., Lee,N., Quackenbush,J. (2000). "A concise guide to cDNA microarray analysis, *Biotechnique* 29(3): 548-550, 552-554, 556.

11. Knudson Jr. (1971). "AG: Mutation and cancer: statistical study of retinoblastoma". *Proc Natl Acad Sci.*

12. Masters, T. (1995) Advanced Algorithms for Neural Networks: A C++ Sourcebook, NY: John Wiley and Sons, ISBN 0-471-10588-0.

13. Ma, Q., Spivak, D., Zhang, K., Shih F., Li, H. (2005). "Genotype Calling Using Neural Networks". (Submitted).

14. Kosko, B. (1992), Neural Networks and Fuzzy Systems, Englewood Cliffs, N.J.: Prentice-Hall.

15. Moller, M.F. (1993). "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning". CiteSeer. Retrieved September 5, 2004 from World Wide Web: http://citeseer.ist.psu.edu/.

16. NCBI. (2005). Web: http://www.ncbinlm.nih.gov

17. NETLABS neural network software [computer software]. (1995). Birmingham, UK: Netlabs.

18. Pfiefer R., Scheier C. (1999). Understanding Intelligence. Cambridge MA: The MIT Press.

19. Ripley, B.D. (1996). Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press.

20. Sarle W.S. (1997). "Neural Net Device Research". Retrived May 2004. Web site: http://www.compamerica.com/nnfaq.

21. Sharp, P.A. (1994). "Split genes and RNA splicing". *Cell* 77:805-815.

22. Slomin, K. (2002). "From patterns to pathways: gene expression data analysis comes of age". *Nature Genetics Supplement.*

23. Wang, D.G., et al. (1998). "Large scale identification. Mapping, and genotyping of single-nucleotide polymorphisms in the human genome". *Science* 280:1077-82.

24. Wang,H,-Y., Luo,M., Li,H. (2005). "A genotyping system capable of simultaneously analyzing >1000 single nucleotide polymorphisms in a haploid genome". *Genome Research*(to appear).

25. Weaver, R.F. (2003). Molecular Biology. Boston, MA: McGraw Hill.

26. Weisstein, E.W. et al. "Conjugate Gradient Method." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/ConjugateGradientMethod.html

27. K. Zhang, M. Q Ma, H. -Y. Wang, M. Banerjee, A. Karmarker, F. Shih, J. Wang and H. Li, SNP auto-calling using support vector machines. Proc. of the 6th International Symposium on Computational Biology and Genome Informatics, Salt Lake City, July 21-26, 2005.