A Comparison of Training Techniques: ADALINE, Back Propagation and Genetic Algorithms

Wei Yang

A Major Report

In

The Department

Of

Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2000

0-612-47857-2

ABSTRACT

A Comparison of Training Techniques:

ADALINE, Back Propagation and

Genetic Algorithms

Wei Yang

This study is in the area of neural network architectures and training algorithms. The emphasis is placed on the comparisons of ADALINE, Back Propagation and Genetic Algorithms in training neural networks. Concrete examples are developed to illustrate and reveal the fundamental theories of neural networks, and demonstrate the strengths and weaknesses of ADALINE, Back Propagation and Genetic Algorithms. An object-oriented approach is applied in the overall analysis and design of the neural network architectures. The Object–oriented programming with C++ is used to facilitate the development and implementation of the neural network architectures and training algorithms.

Acknowledgements

I have been fortunate to have an extremely kind and considerate advisor, Dr. Peter Grogono, who gives me a lot of helps in guiding me throughout the project. I want to say thanks to Dr. Peter Grogono.

I would also like to thank Dr. Rajjan Shinghal for his time and helps in examining this paper.

1.INTRODUCTION 1

1.1 SCOPE 1

2. NEURAL NETWORKS PARADIGM 2

2.1 BIOLOGICAL BASIS FOR NEURAL NETWORKS 2
2.2 NEURAL NETWORK 3
2.2.1 ABSTRACTION OF NEURAL NETWORKS 3
2.2.2 TYPES OF NEURAL NETWORKS 5

3. OBJECT ORIENTED DESIGN & IMPLEMENTATION OF NEURAL NETWORKS .. 8

3.1 MATHEMATICAL ABSTRACTION OF NEURAL NETWORK 8
3.1.1 NEURAL NETWORK FORMALIZATION 9
3.1.2 PATTERN SET FORMALIZATION 9
3.1.3 NETWORK PARAMETER FORMALIZATION 9
3.1.4 NETWORK TOPOLOGY FORMALIZATION 10
3.2 DESIGN AND IMPLEMENTATION OF NEURAL NETWORK WITH CLASSES IN C++ 10
3.2.1 CLASS INTERFACE AND RESPONSIBILITIES 12

4. IMPLEMENTING ADALINE NEURAL NETWORK 15

4.1 OBJECT ORIENTED IMPLEMENTATION OF ADALINE NETWORK 15
4.1.1 ADALINE NETWORK NODE CLASS 15
4.1.2 ADALINE LINK CLASS 20
4.2 TRAINING ADALINE NETWORK WITH EXAMPLE OF OPEN-TO-BUY 20
4.2.1 OPEN-TO-BUY EXAMPLE DESCRIPTION 20
4.2.2 SELECTING ADALINE TRAINING SET FOR OPEN-TO-BUY EXAMPLE 22
4.2.3 CREATING ADALINE TRAINING SET FOR OPEN-TO-BUY EXAMPLE 22
4.2.4 CONSTRUCTING ADALINE NETWORK 24
4.2.5 TRAINING THE ADALINE NETWORK TO SOLVE THE OPEN-TO-BUY PROBLEM 24
4.3 ANALYSIS OF ADALINE WITH EXAMPLE OF XOR PROBLEM 31

5.IMPLEMENTING BACKPROPAGATION NEURAL NETWORK 33

5.1 OBJECT-ORIENTED IMPLEMENTATION OF BACKPROPAGATION NETWORKS 34
5.1.1 BACKPROPAGATION NEURAL NETWORK NODE CLASSES 34
5.1.2 BACKPROPAGATION NEURAL NETWORK LINK CLASS 36
5.2 APPLICATION OF BP WITH EXAMPLE -- XOR PROBLEM REVISITED 36
5.3 SOLVING XOR PROBLEM BY USING BACKPROPAGATION NEURAL NETWORK 38
5.3.1 XOR PROBLEM TRAINING SET 38
5.3.2 CREATING TRAINING SET FOR XOR PROBLEM 38
5.3.3 CONSTRUCTING BP NETWORK FOR XOR PROBLEM 38
5.3.4 TRAINING THE BP NETWORK TO SOLVE THE XOR PROBLEM 40

5.3.4.1 Training Stage.......... 40
5.3.4.2 Save Training Results Stage:.......... 42
5.3.4.3 Load and Run Stage.......... 43
5.3.4.4 Outputs.......... 44

6. SUMMARY OF COMPARISON BETWEEN ADALINE AND BP.......... 47

7. GENETIC ALGORITHMS.......... 48

7.1 BASICS OF GENETIC ALGORITHMS.......... 48
7.2 OO ANALYSIS FOR GENETIC ALGORITHMS.......... 49
7.3 DESIGN AND IMPLEMENT GENETIC ALGORITHM CLASSES IN C++.......... 50

8. TRAINING RECURRENT NEURAL NETWORKS WITH GENETIC ALGORITHMS.......... 53

8.1 OBJECT ORIENTED ANALYSIS OF RECURRENT NEURAL NETWORK WITH GA.......... 53

9. USING GA TO TRAIN RECURRENT NEURAL NETWORK FOR XOR PROBLEM 55

9.1 EXECUTION OUTPUT FOR THE XOR PROBLEM WITH GENETIC ALGORITHM.......... 56

10.SUMMARY OF COMPARISON BETWEEN BP AND GA.......... 58

REFERENCES.......... 60

APPENDIX SOURCE LISTINGS AND OUTPUT FILES.......... 62

Figure 2-1 Conceptual diagram of a neuron 2
Figure 2.2 An abstract neuron 4
Figure 2.3 Functional model of a neural network 5
Figure 2.4 ADALINE and Backpropagation neural network schematics. 7
Figure 2.5 A Recurrent network with two hidden-layer neurons 7
Figure 3.1 Class diagram for neural network nodes and links 11
Figure 4.1 Class hierarchy for Base_Node, Input_Node *and* Bias_Node 16
Figure 4.2 Class hierarchy for Base_Node, Feed_Forward_Node, ADALINE_Node 18
Figure 4.3 Input patterns for "Open-To-Buy", which is in the Rectangle region 21
Figure 4.4 Training set for Open-To-Buy example (file "OTBTrnSet.txt") 23
Figure 4.5 Structure of constructed ADALINE network to solve Open-To-Buy example. 24
Figure 4.6 Partial outputs for the training set presented to ADALINE network 26
Figure 4.7 Number of iterations the training set has been presented to the ADALINE .. 27
Figure 4.8 Outputs of the running result of ADALINE network (file OTB.out*)* 30
Figure 4.9 Input pattern for XOR problem 31
Figure 4.10 All possible functions for XOR input patterns 31
Figure 4.11 Separations of input space corresponding to OR and AND 32
Figure 5.1 The basic structure of a Backpropagation neural network 33
Figure 5.2 Class hierarchy for Backpropagation(BP) Neural Network nodes 34
Figure 5.3 XOR Problem training Set 38
Figure 5.4 Backpropagation neural network for XOR problem 39
Figure 5.5 Sigmoid Function and the output mapping 40
Figure 5.6 Stored Data for Nodes 42
Figure 5.7 Stored Data for Links 43
Figure 5.8 Training output for XOR problem 45
Figure 5.9 Running results output for XOR problem 46
Figure 7.1 Class diagram for genetic algorithm architecture 49
Figure 8.2 The class diagram for recurrent neural network 54
Figure 9.1 Recurrent neural network for XOR problem 55
Figure 9.2 Output of start of an XOR genetic training experiment 56
Figure 9.3 Output for generation 5 for XOR genetic training experiment 56
Figure 9.4 Output for generation 75 for XOR genetic training experiment 57
Figure 9.5 Output for generation 1000 for XOR genetic training experiment 57

1.Introduction

My project focuses on:

- Object oriented design and implementation of Neural Networks;
- Neural Networks with and without hidden layers (ADALINE vs. Backpropagation Algorithms);
- Neural Networks with and without feedback (recurrent vs. non-recurrent Neural Network);
- Applying Genetic Algorithm in solving Neural Network problems;
- Comparisons between ADALINE and Backpropagation, and between Backpropagation and Genetic Algorithm.

1.1 Scope

The purpose of this project is to study the common neural network architectures, find out their strength and weakness with implementation of practical examples, using Object Oriented methodology in C++. Neural networks can be formalized, realized as objects, and used in countless applications. In this project, we are mainly interested in the study of the foundations from which any neural network architecture can be constructed, and relatively detailed studies of several neural network architecture such as ADALINE, Back propagation (BP), and recurrent neural networks and their comparisons as well.

We are going to develop several concrete examples to illustrate and reveal the fundamental theory, major difference and possible incorporations among these neural network architectures and algorithms.

2. Neural Networks Paradigm

2.1 Biological Basis for Neural Networks

Studies over the past few decades have shed some light on the construction and operation of our brains and nervous systems. All living organisms are made up of cells. The basic building blocks of the nervous system are nerve cells, called *neurons*. The major components of a neuron include a central cell body, dendrites, and an axon. Figure 2.1 shows a conceptual diagram of a neuron; it is a sketch of only one representation of a neuron. The signal flow goes from left to right, from the dendrites, through the cell body, and out through the axon. The signal from one neuron is passed on to another by means of a connection between the axon of the first and a dendrite of the second. This connection is called a *synapse*. Axons often synapse onto the trunk of a dendrite, but they can also synapse directly onto the cell body.

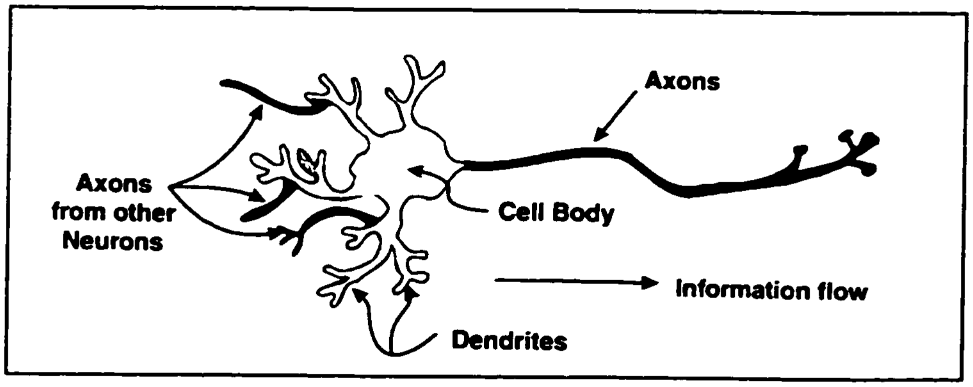


Figure 2-1 Conceptual diagram of a neuron

2.2 Neural Network

A Neural network is an information processing system that has certain performance characteristics in common with biological neural networks. The neural networks have been developed as generalizations based on the assumptions that:

- Information processing occurs at many simple elements called neurons (or nodes);
- Signals are passed between neurons over connection links;
- Each connection link has an associated weight, which, in a typical neural network, multiplies the signal transmitted;
- Each neuron applies an activation function (usually nonlinear) to its network inputs to determine its output signal;

A neural network is characterized by:

- Its pattern of connections between the neurons (called its architecture);
- Its method of determining the weights on the connections (called its training, or learning, algorithm);
- Its activation function.

Neural networks can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

2.2.1 Abstraction of Neural Networks

In general, a neural network is a collection of simple, analog signal processors, connected through *links*. Schematically, a neural network is represented in the form of a directed graph, where the nodes represent the *processing elements*, the arcs represent the modulating connections, and arrowheads on the arcs indicate the normal direction of signal flow. Processing elements are larger structures known as *layers* where each layer performs an analog integration of its inputs to determine its *activation value.*

Figure 2.2 illustrates the typical model of a neural network processing unit (also called *neuron* or *node*). In this figure, it shows the structure of an abstract neuron with n inputs. Each input channel i can transmit a real value Xi. The *primitive function f* computed in the body of the abstract neuron can be selected according to purpose for the specific network architectures. Usually the input channels have an associated weight, which means that the incoming information Xi is multiplied by the corresponding weight Wi. The transmitted information is integrated at the neuron (usually just by adding the different signals) and the primitive function is then evaluated.

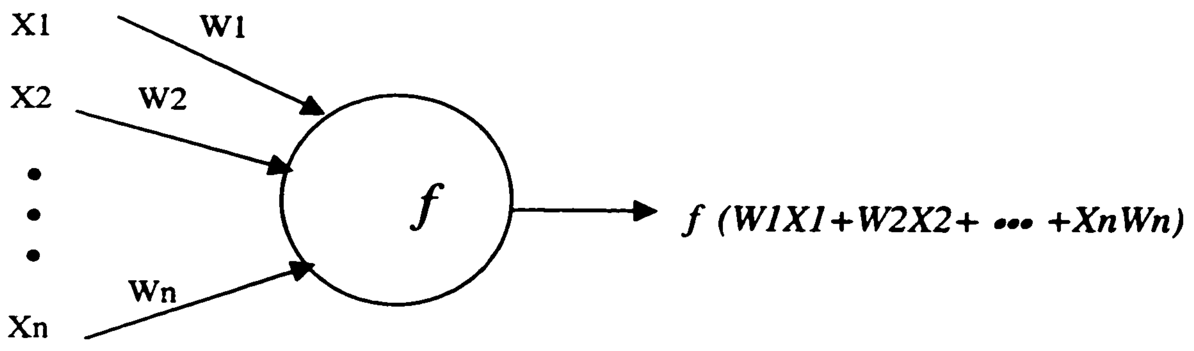


Figure 2.2 An abstract neuron

If we conceive of each node in a neural network as a primitive function capable of transforming its input into a precisely defined output, then neural networks are nothing but networks of primitive functions. Different models of neural networks differ mainly in the assumptions about the primitive functions used, the interconnection pattern, and the timing of the transmission of information.

Typical neural networks have the structure shown in Figure 2.3. The network can be thought of as a function Φ which is evaluated at the point *(x, y, z)*. The node implements the primitive functions *f1, f2, f3, f4* which are combined to produce Φ. The function Φ implemented by a neural network will be called the *network function*. Different selections of the weights α1, α2,...,α5 produce different network functions. Therefore, three elements are particularly important in any model of neural networks:

- The structure of the nodes;
- The topology of the network;
- The learning algorithm used to find the weights of the network.

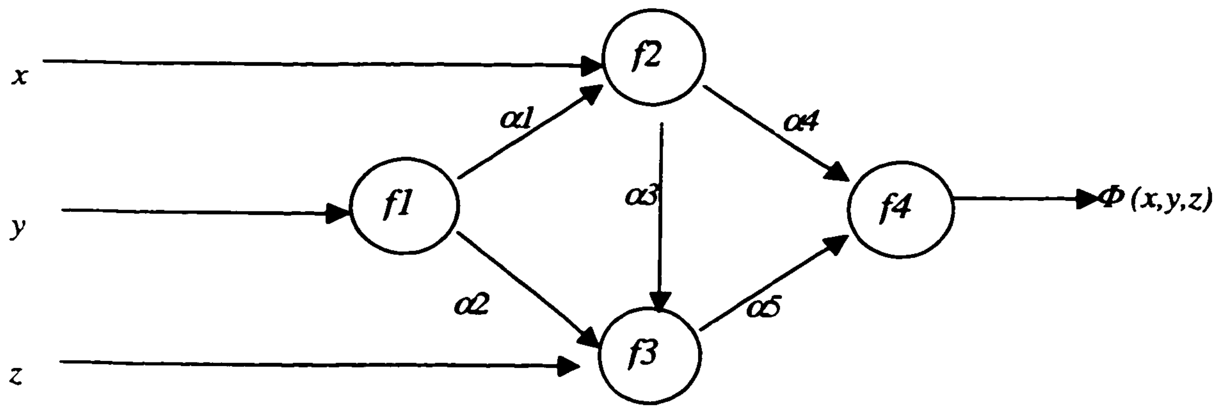
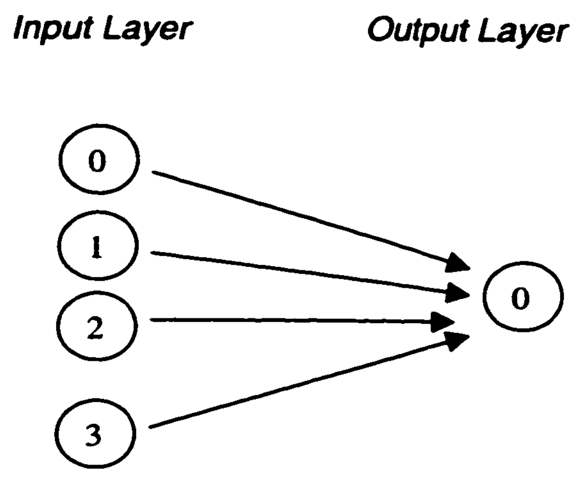


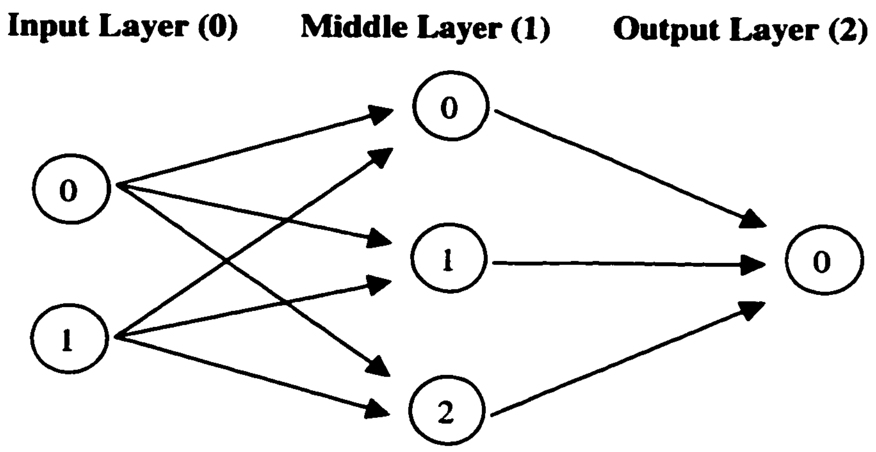
Figure 2.3 Functional model of a neural network

2.2.2 Types of Neural networks

There are many types of neural network architectures and associated algorithms such as *ADALINE, Backpropagation (BP), Self-Organizing, BAM (Bi-directional Associative Memory),* and so on. They can also be classified as either feed-forward or recurrent networks (i.e. nets without and with feedkacks). These algorithms are also called network architectures as every algorithm has its associated network topology and links. In this project, we mainly concentrate on ADALINE, Backpropagation and recurrent neural networks, and exploit the differences in the networks with or without back propagation, and with or without feedback. Figure 2.4 illustrates the ADALINE and Backpropagation neural network schematics.



ADALINE Schematic



Multi-layered schematic

Figure 2.4 ADALINE and Backpropagation neural network schematics.

Figure 2.5 illustrates the recurrent network with two hidden-layer neurons whose outputs feed back to each neuron in the hidden layer (not just feeding forward to the output-layer neurons). Problems with continuous input data require neural networks with feedback loops between neuron layers.

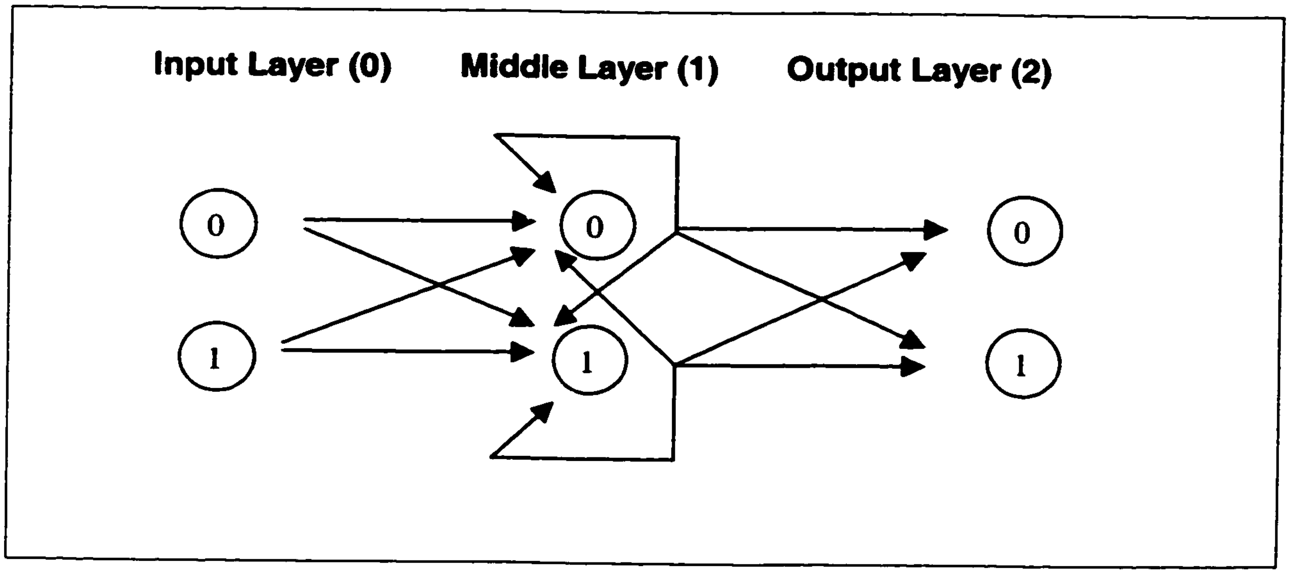


Figure 2.5 A Recurrent network with two hidden-layer neurons

3. Object Oriented Design & Implementation of Neural Networks

Observing that although different neural network may have different functionality of the nodes (neurons) and the links, the structure or topology of each neural network is based on a common network model. This commonality makes neural networks ideal for object-oriented implementations.

In this project, we use objected-oriented design with object-oriented programming implementation in C++. To create a set of base classes capable of implementing several kind of neural network architectures, characteristics common to each architecture must be recognized. The common attributes can be generalized as the fact that every neural network architecture is composed of a set of interconnected nodes that accept input, processes this information in some way, and produce an output which may be passed to other nodes.

3.1 Mathematical Abstraction of Neural Network

A neural-network formalization is simply a mathematical representation of a neural network model. We apply the generic neural-network formalizations which were introduced by Fiesler (1992) and extended by Rogers and Satyadas (1994) to formalize the ADALINE and Backpropagation neural networks.

3.1.1 Neural Network Formalization

A neural network can be defined with the following 3-tuple:

Neural Network $NN = (S, P, T)$

where

- S is the pattern set, used for training, testing or operating the network;
- P is the network parameters;
- T is the Network's topology.

3.1.2 Pattern Set Formalization

The formalized pattern set can be expressed as:

Pattern Set $S = \{I, O\}$

where I is a set of input patterns, O is a set of corresponding desired output patterns. The input and output pattern sets can also be further decomposed as:

$$I = \{P_{kj}\}$$

where k is input number and j is the input pattern *component*(a group of inputs in an input pattern).

$$O = \{O_{kj}\}$$

where k is the desired output pattern and j is the desired output pattern component.

3.1.3 Network Parameter Formalization

The parameter set (P) in NN= (S, P, T) contains various parameters used in the neural network's training, testing and running. The parameter set would include *learning rate*, *momentum term* (a factor, which can take values between 0 to 1, used for updating weights), maximum number of *training iterations, tolerance* etc.

3.1.4 Network Topology Formalization

The last element in neural network NN = *(S, P, T)* is its topology *T*. It defines the framework (*F*) and interconnecting link (*L*) between the network nodes.

Topology $T = (F, L)$

Where the framework (*F*) is the set of layers in the network.

Framework $F = \{C1, C2, \ldots, Cn\}$

Where the layer *Ci* is a set of nodes (*n*), each identified by its layer (*i*) and its position (*j*) within its layer.

Layer $Ci = \{n_{i,j}\}$

3.2 Design and Implementation of Neural Network with Classes in C++

Design of the formalized neural network is generalized with two base classes. One base class represents the node (or neuron), and one presents link (or connection) of the general neural network. We name them as Base_Node class and Base_Link class. The Class diagrams are shown in Figure 3.1. These two classes are interdependent and all other classes which implement different algorithms are derived directly or indirectly from them.

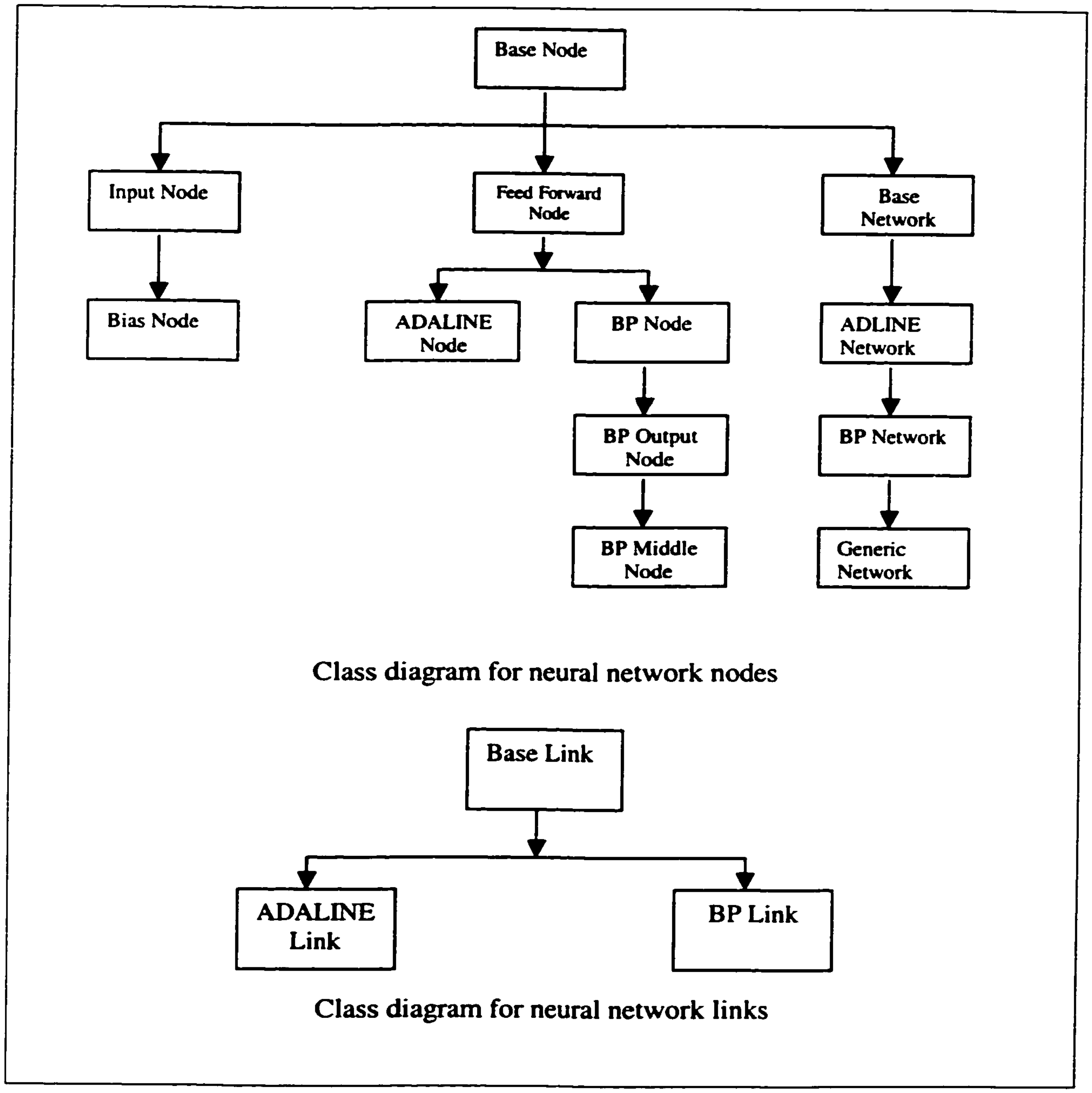


Figure 3.1 Class diagram for neural network nodes and links

In these class diagrams, we omit some common neural network architectures, such as, SON (Self-Organizing Network) and BAM (Bi-directional Associative Memory), which are not of interest for this project.

3.2.1 Class Interface and Responsibilities

1) Class Base_Node

Responsibility:

Class Base_Node is the neural network node which manages the receiving of input values and distribution of the processed information to other nodes in network. One of the primary responsibilities of the Base_Node class is to maintain connectivity between the network nodes. This task is implemented by using doubly linked list LList objects through their two-way pointers. As the Base_Node class is a higher abstraction of neural network node, it defines several virtual functions which can be overridden by its child classes.

Class Interface:

```
class Base_Node {
    private:
        static int ticket;
    protected:
        int id;             // Identification Number
        double *value;      // Value (s) stored by this node
        int value_size;     // Number of Values stored by this node
        double *error;      // Error value (s) stored by this node
        int error_size;     // Number of Error values stored by this node
        LList in_links;     // List for input links
        LList out_links;    // List for output links
    public:
        Base_Node ( int v_size=1, int e_size=1) ;    // Constructor
        ~Base_Node ( void) ;                         // Destructor
        LList *In_Links ( void) ;
        LList *Out_Links ( void) ;
        virtual void Run ( int mode=0) ;
        virtual void Learn ( int mode=0) ;
        virtual void Load ( ifstream &infile)  ;
        virtual void Save ( ofstream &outfile)  ;
        inline virtual double Get_Value ( int id=NODE_VALUE)  ;
        inline virtual void Set_Value ( double new_val, int id=NODE_VALUE)  ;
        inline virtual double Get_Error ( int id=NODE_ERROR)  ;
        inline virtual void Set_Error (  double new_val, int id=NODE_ERROR)  ;
        inline int Get_ID ( void) ;
        inline virtual char *Get_Name ( void) ;
        void Create_Link_To ( Base_Node &to_node, Base_Link *link) ;
        virtual void Print ( ofstream &out) ;

        friend void Connect ( Base_Node &from_node, Base_Node &to_node,
                     Base_Link *link) ;
        friend void Connect ( Base_Node &from_node, Base_Node &to_node,
                     Base_Link &link) ;
        friend void Connect ( Base_Node *from_node, Base_Node *to_node,
                     Base_Link *link) ;
        friend int Disconnect ( Base_Node *from_node, Base_Node *to_node) ;

        friend double Random ( double lower_bound, double upper_bound) ;
    };
```

2) Class Base_Link

Responsibility:

Each link object helps to maintain connectivity in the network by storing pointer to a source node object and to a destination node object. The pointer to Base_Node object is of the type of "pointer to Base_Node". The Base_Link class also provides a value set so that values can be maintained in the link objects. This ability is necessary for most Neural network models since the values or weights associated with the links are significant to the network state.

Class Interface:

```
class Base_Link {
    private:
        static int ticket;
    protected:
        int id;                   // ID number for link
        double *value;            // Value ( s) for Link
        Base_Node *in_node;       // Node instance link is comming from
        Base_Node *out_node;      // Node instance link is going to
        int value_size;
    public:
        Base_Link ( int size=1) ;  // Constructor
        ~Base_Link ( void) ;        // Destructor for Base Links
        virtual void Save ( ofstream &outfile) ;
        virtual void Load ( ifstream &infile) ;
        inline virtual double Get_Value ( int id=WEIGHT) ;
        inline virtual void Set_Value ( double new_val, int id=WEIGHT) ;
        inline virtual void Set_In_Node ( Base_Node *node, int id) ;
        inline virtual void Set_Out_Node ( Base_Node *node, int id) ;
        inline virtual Base_Node *In_Node ( void) ;
        inline virtual Base_Node *Out_Node ( void) ;
        inline virtual char *Get_Name ( void) ;
        inline virtual void Update_Weight ( double new_val) ;
        inline int Get_ID ( void) ;
        inline virtual double In_Value ( int mode=NODE_VALUE) ;
        inline virtual double Out_Value ( int mode=NODE_VALUE) ;
        inline virtual double In_Error ( int mode=NODE_ERROR) ;
        inline virtual double Out_Error ( int mode=NODE_ERROR) ;
        inline virtual double Weighted_In_Value ( int mode=NODE_VALUE) ;
        inline virtual double Weighted_Out_Value ( int mode=NODE_VALUE) ;
        inline virtual double Weighted_In_Error ( int mode=NODE_VALUE) ;
        inline virtual double Weighted_Out_Error ( int mode=NODE_VALUE) ;
        inline virtual int Get_Set_Size ( void) ;
};
```

4. Implementing ADALINE Neural Network

The ADALINE typically uses bipolar (1 or –1) activation for its input signals and its target output (although it is not restricted to such values). The weights on the connections from the input units to the ADALINE are adjustable. The ADALINE also has a *bias,* which acts like an adjustable weight on a connection from a unit whose activation is always 1. In general, an ADALINE can be trained using the *delta rule*. This learning rule minimizes the mean squared error between the activation and the target value. This allows the net to continue learning on all training patterns, even after the correct output value is generated (if a threshold function is applied) for some patterns. After training, if the net is being used for pattern classification in which the desired output is either a +1 or a –1, a threshold function is applied to the net input to obtain the activation. If the net input to the ADALINE is greater than or equal to 0, its activation is set to 1; otherwise it is set to –1. Any problem for which the input patterns corresponding to the output value +1 are linearly separable from input pattern corresponding to the output value –1 can be modeled successfully by an ADALINE unit.

4.1 Object Oriented Implementation of ADALINE Network

4.1.1 ADALINE Network Node Class

In ADALINE neural network, there are three basic types of nodes, i.e. *Input_Node, Bias_Node, ADALINE_Node* (note we only name output node as ADALINE_Node). Recall the class hierarchy shown in Figure 3.1. The Input_Node of ADALINE network can be derived from Base_Node, and furthermore the Bias_Node of ADALINE network can be derived from Input_Node as it is a special kind of input node whose value is always a constant. This class hierarchy is shown again in the Figure 4.1.

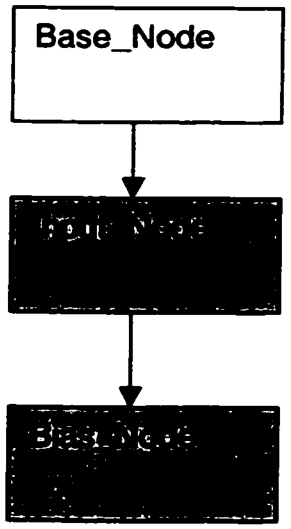


Figure 4.1 Class hierarchy for Base_Node, Input_Node and Bias_Node

The Input_Node and Bias_Node can be applied in many other neural network architectures. They are not the unique features of ADALINE network, so we generalize them as common nodes for neural network and define them in head file "common.h". It can be used for a general purpose for neural network architectures. The next stage is to design the output node class, which is a unique feature of ADALINE neural network. We name it ADALINE_Node, and put its class declaration in a separate header file.

The class definitions and interfaces are as following.

```
class Input_Node : public Base_Node    // The Input Node class is a generic
{                                       // Input Node. It can be used with
                                        // most networks.
        public:
            Input_Node ( int size=1) : Base_Node ( size,size)
            {
                    for (int i=0; i<size; i++)
                    {
                        error[i]=0.0;
                        value[i]=0.0;
                    };
            };
            virtual char *Get_Name ( void)
            {
                static char name[]="INPUT_NODE";
                return name;
            };
        };


// The Bias Node Class is a node that always produces the same output.
// The Bias Node's default output is 1.0
class Bias_Node : public Input_Node
{
        public:
            Bias_Node ( double bias=1.0) : Input_Node ( 1)      // Constructor
                { value[0]=bias; };
            virtual void Set_Value ( double value, int id=0) {}; // Disable Set_Value
            virtual double Get_Value ( int id=0) { return value[0]; };
            virtual char *Get_Name ( void)
            {
                static char name[]="BIAS_NODE";
                return name;
            };
        };
```

However the ADALINE_Node has its unique feature, and should be defined according to the ADALINE network architecture. As the ADALINE_Node is a kind of feedforward node, we can derive it from Feed_Forward_Node. This class hierarchy is shown in Figure 4.2.

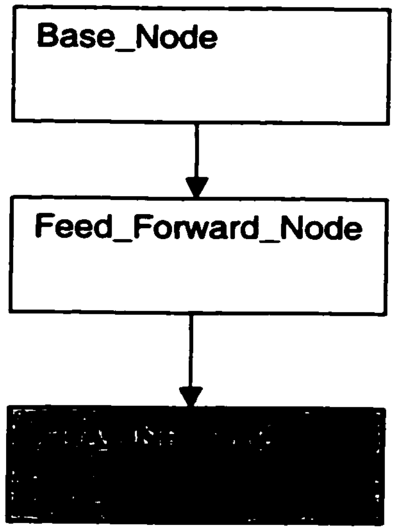


Figure 4.2 Class hierarchy for Base_Node, Feed_Forward_Node and ADALINE_Node

As we know, in addition to ADALINE network, Backpropagation network is also a kind of feed forward neural network. They share the common features in this aspect. That is why we need to develop a Feed_Forward_Node class. The definition and interface of Feed_Forward_Node class is in file "base.h" and is shown as following:

```
// This derived class provides a generic feed-forward neural-network node which
// can be used by the ADALINEs and Backprop networks.
class Feed_Forward_Node : public Base_Node
        {
        protected:
            virtual double Transfer_Function ( double value) ;

        public:
            Feed_Forward_Node ( int v_size=1, int e_size=1) ; // Constructor
            virtual void Run ( int mode=0) ;
            virtual char *Get_Name ( void) ;
        };
```

The ADALINE_Node class can be derived from Feed_Forward_Node class. The definition and interface of ADALINE_Node class is in file "adaline.h" and is shown as following:

```
// ADALINE processing Node
class ADALINE_Node : public Feed_Forward_Node       {
    protected:
        virtual double Transfer_Function ( double value) ;

    public:
        ADALINE_Node ( void) ;
        ADALINE_Node ( double lr) ;
        virtual void Learn ( int mode) ;
        virtual char *Get_Name ( void) ;
    };
```

For ADALINE_Node class, we need to override the Transfer_Function as ADALINE uses a *threshold function f(x).*

$$f(x) = -1.0 \text{ if } x < 0 \text{ otherwise } f(x) = 1.0$$

We will do a similar override of this function by a Sigmoid function for Backpropagation Neural network later.

Now we summarize our design and implementation issues of the ADALINE network nodes. In design, we derive Input_Node class directly from Base_Node class, and derive Bias_Node from Input_Node. However, we derive ADALINE_Node from Feed_Forward_Node (which is derived from Base_Node). In implementation, we put the declarations of Input_Node class and Bias_Node in one header file "common.h", which can also be used for other kind of neural network architectures, and put the declaration of ADALINE_Node in "adaline.h", which serves only for ADALINE network.

4.1.2 ADALINE Link Class

ADALINE Link Class can be derived from Base_Link class. In the constructor, the weight value is initialized by a random number ranging from –1.0 to 1.0. The class interface is as following.

```
class ADALINE_Link : public Base_Link          // Link for ADALINE Node
  {
   public:
      ADALINE_Link ( void) ;
      virtual void Save ( ofstream &outfile) ;
      virtual void Load ( ifstream &infile) ;
      virtual char *Get_Name ( void) ;
      };
```

4.2 Training ADALINE Network with Example of Open-To-Buy

4.2.1 Open-To-Buy Example Description

To demonstrate the ADALINE network's capability of learning, we suppose that there is a Supply Chain store. The purchasing department maintains the budget available for annual or monthly purchasing, which we call "Open-To-Buy". The three major elements influencing the Open-To-Buy are Company's net income or loss (here call it Company Income), funds from the financing institutions or the loan being paid back (here call it Banking Amount), and a certain amount money to be reserved as the company's contingency (here call it Contingency). The Contingency is a constant value. The Company Income and Banking Amount has different factors affecting the Open-To-Buy, we assume that the factors are as followings:

4/3*CompIncome, 1/2 * BankingAmount, 1*Contingency

For easy processing, we assume that any input values for CompIncome and BankingAmount are being divided by a certain constant and thus be restricted to the range of (-1.0 ~ +1.0), and the Contingency is just a constant 1/3. So:

CompIncome = (- 3/2 * BankingAmount + 1) /4

We say the "Open-To-Buy" can only be opened when the company has a total net income which is greater or equal to (- 3/2 * BankingAmount + 1) /4, otherwise it will be closed.

CompIncome >= (- 3/2 * BankingAmount + 1) /4 Open

CompIncome < (- 3/2 * BankingAmount + 1) /4 Closed.

We are going to train the ADALINE network with a limited number of data points of inputs to solve this problem. Training the ADALINE network involves presenting the input patterns in the training set to the network, running the network to get an output, and adjusting the link values, that is the weights, if the network produces a value other than the desired output. Training is completed when the number of training set patterns the network gets wrong is below a predetermined tolerance. In Open-To-Buy problem, the tolerance is 0. Recall that we chose only the input values with absolute value less than 1. This equation can be shown as in figure 4.3.

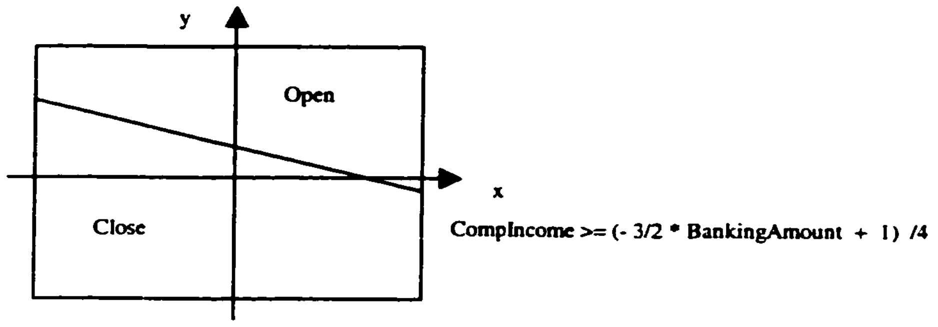


Figure 4.3 Input patterns for "Open-To-Buy", which is in the Rectangle region

4.2.2 Selecting ADALINE Training Set for Open-To-Buy Example

We use a random number generator to get values of the two inputs x and y, further we restrict these values to be in the range of from –1.0 to +1.0.

Inputs:	*InputX=randomly chosen value from –1.0 to +1.0* *(represents BankingAmount))*	
	InputY=randomly chosen value from –1.0 to +1.0 (represents CompIncome)	
Threshold Value:	*THValue = (- 3/2InputX +1) /4*	
Desired Outputs:	-1.0	if *InputY < THValue* *(represents Open-To-Buy should be closed)*
	1.0	if *InputY >= THValue* *(represents Open-To-Buy is open)*
Tolerance:	0	

4.2.3 Creating ADALINE training Set for Open-To-Buy Example

We write a C++ program to generate the training set by using the random number generator.

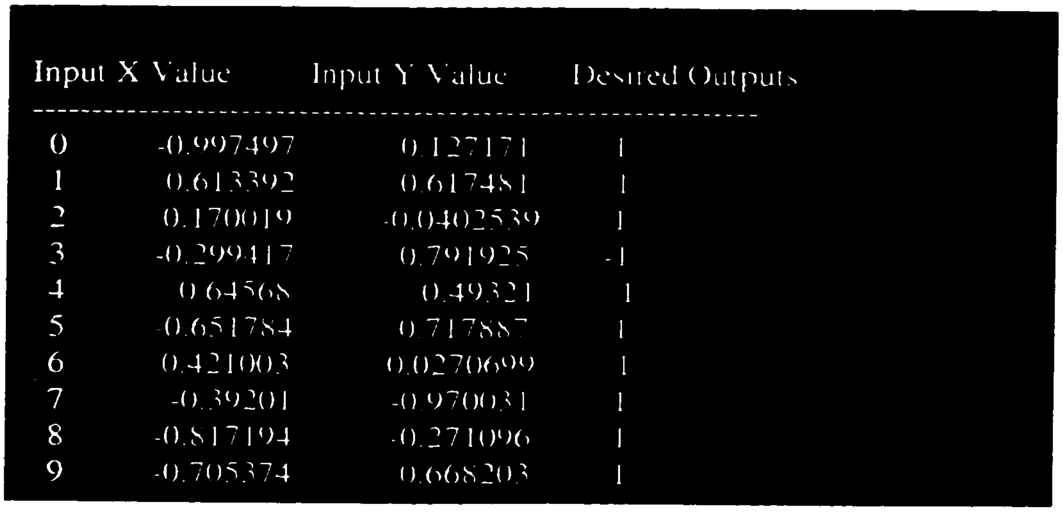
<u>CODES FOR GENERATING TRAINING SET:</u>

```
void main () {

        double InputX, InputY, THValue, DesiredOutput;
        ofstream fout ( "LinearTrnSet1.txt") ;
/*        fout<<"  "<<"    Input X Value"<< "         ";
          fout <<Input Y Value          ";
          fout<<"Desired Outputs"<<endl;
          fout<<"--------------------------------------------------------------"<<endl;
*/    //This part just shows the output format, and will not be included
      //in the real training set file, so it is commented out.

        for ( int i=0; i<250; i++) {

                //RAND_MAX=32767 which is the max integer value
                InputX= ( (double) rand () / ( double) RAND_MAX) *2.0-1.0;
                InputY= ( (double) rand () / (double) RAND_MAX) *2.0-1.0;
                THValue= ( (-3.0/2.0) *InputX+1) /4;
```

```
            if ( InputY<THValue)
                    DesiredOutput=1;
            else
                    DesiredOutput=-1;
            fout<<setiosflags ( ios::right)  <<setw ( 3)  <<i<<" ";
            fout<<setw ( 18)  <<InputX <<setw ( 20)  <<InputY;
            fout<<setw ( 10)  <<DesiredOutput<<endl;
    }
    fout.close ()  ;
}
```

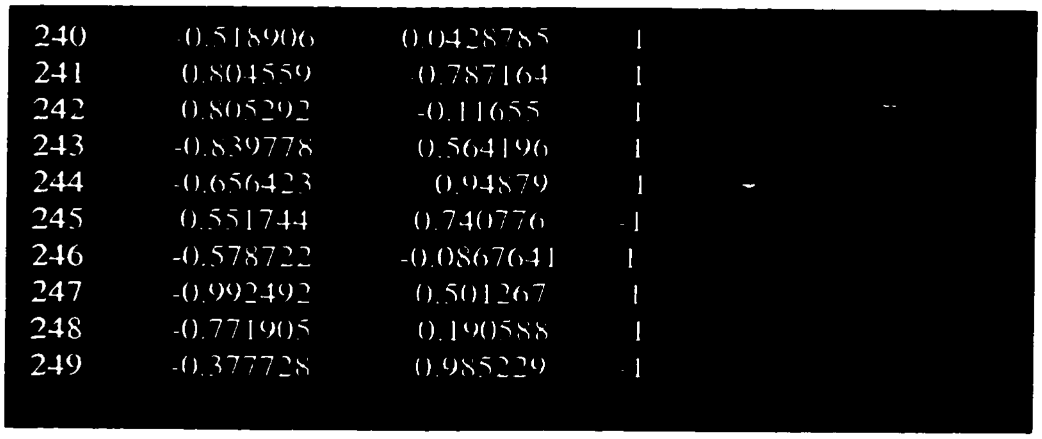
we select 250 pairs of x, y inputs as training patterns for our example. The training set is stored in the file "OTBTrnSet.txt", the partial output is as following:

GENERATED TRAINING SET:



```
Input X Value        Input Y Value      Desired Outputs
----------------------------------------------------------
 0        -0.997497         0.127171        1
 1         0.613392         0.617481        1
 2         0.170019        -0.0402539       1
 3        -0.299417         0.791925       -1
 4         0.64568           0.49321         1
 5        -0.651784         0.717887        1
 6         0.421003         0.0270699       1
 7         -0.39201         -0.970031       1
 8        -0.817194        -0.271096        1
 9        -0.705374         0.668203        1
```

... ... Inputs from 10 to 239 are omitted ... ...



```
240       -0.518906         0.0428785       1
241        0.804559        -0.787164        1
242        0.805292        -0.11655         1
243       -0.839778         0.564196        1
244       -0.656423          0.94879        1
245        0.551744         0.740776       -1
246       -0.578722        -0.0867641       1
247       -0.992492         0.501267        1
248       -0.771905         0.190588        1
249       -0.377728         0.985229       -1
```

Figure 4.4 Training set for Open-To-Buy example (file "OTBTrnSet.txt")

4.2.4 Constructing ADALINE network

To solve this Open-To-Buy problem, we should construct an ADALINE network. In this example we chose 4 nodes, i.e. two input nodes, one bias node and one output node. Recall that these ADALINE Nodes are subclass of Base Nodes. The structure constructed for this example is as in Figure 4.5.

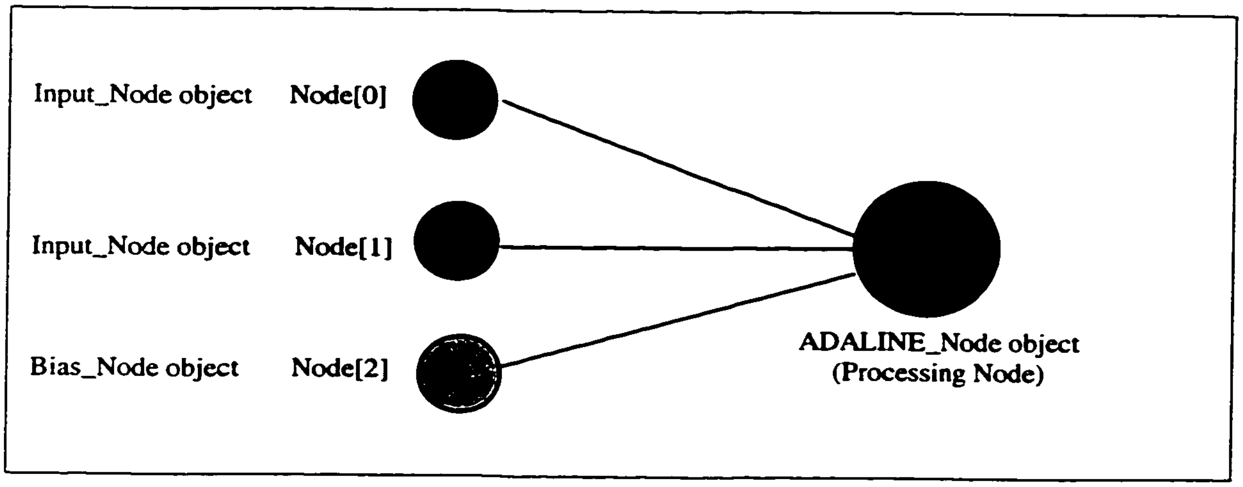


Figure 4.5 Structure of constructed ADALINE network to solve Open-To-Buy example

4.2.5 Training the ADALINE Network to Solve the Open-To-Buy Problem

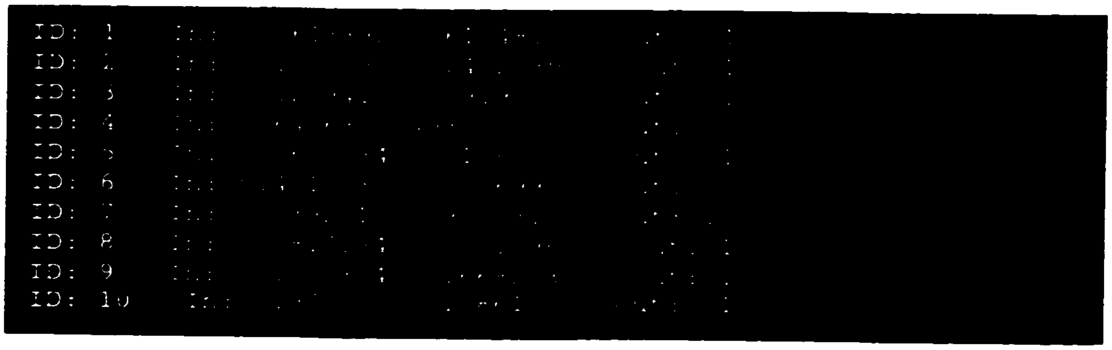
Training ADALINE network involves several steps:

1) Presenting the training set input patterns to the network
2) Running the network to get an output, and adjusting the link values if the network produce a value which is not as desired
3) Training is complete when the number of the training set patterns the network gets wrong is below the tolerance

```
// Train ADALINE
int iteration=0;
int good=0;

// Train until all patterns are good
while (good<250) {
   good=0;
   for (int i=0; i<250; i++)  {
       Node[0]->Set_Value ( data[i]->In ( 0) )  ;   // Set Input Node Values
       Node[1]->Set_Value ( data[i]->In ( 1) )  ;

       Node[3]->Run ()  ;                          // Run ADALINE Node

      if (data[i]->Out ( 0)  !=Node[3]->Get_Value ())
      {  // If ADALINE produced an error, then perform learning function
         Node[3]->Learn ()  ;
         Break
      }
      else good++;
   }
 cout << iteration << ".   " << good << "/250"  << endl;
 iteration++
}
```

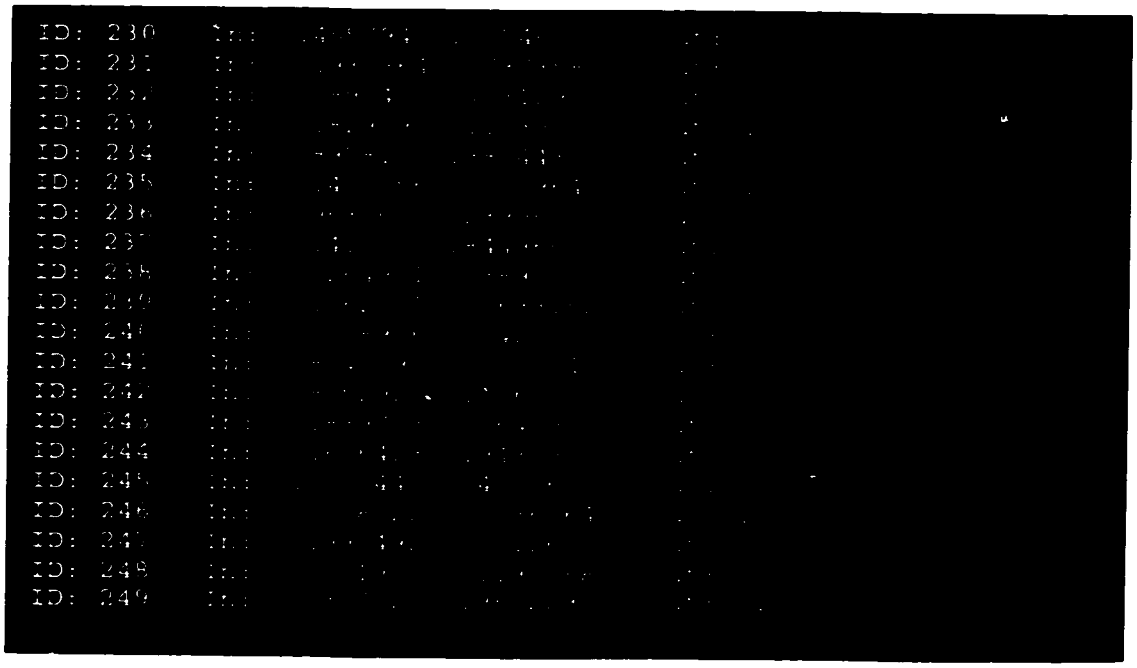
After the network is successfully trained, it can solve the proposed problem with desired results.

Partial outputs for the training set presented to ADALINE network:



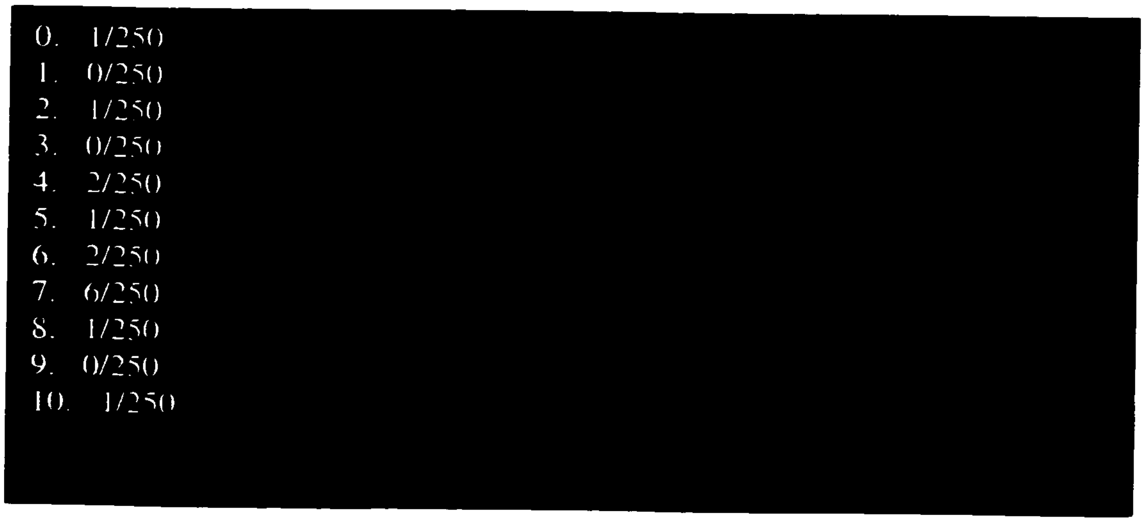
```
ID: 1    [illegible]
ID: 2    [illegible]
ID: 3    [illegible]
ID: 4    [illegible]
ID: 5    [illegible]
ID: 6    [illegible]
ID: 7    [illegible]
ID: 8    [illegible]
ID: 9    [illegible]
ID: 10   [illegible]
```

… … Inputs 11 to 229 are omitted … …



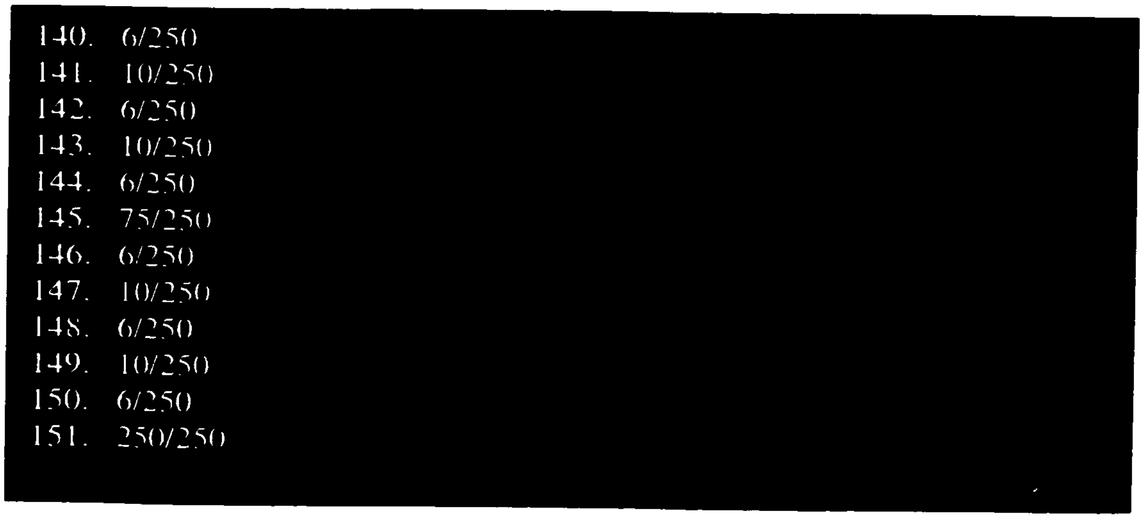
```
ID: 230  [illegible]
ID: 231  [illegible]
ID: 232  [illegible]
ID: 233  [illegible]
ID: 234  [illegible]
ID: 235  [illegible]
ID: 236  [illegible]
ID: 237  [illegible]
ID: 238  [illegible]
ID: 239  [illegible]
ID: 240  [illegible]
ID: 241  [illegible]
ID: 242  [illegible]
ID: 243  [illegible]
ID: 244  [illegible]
ID: 245  [illegible]
ID: 246  [illegible]
ID: 247  [illegible]
ID: 248  [illegible]
ID: 249  [illegible]
```

Figure 4.6 Partial outputs for the training set presented to ADALINE network (file OTB.out)



```
0.  1/250
1.  0/250
2.  1/250
3.  0/250
4.  2/250
5.  1/250
6.  2/250
7.  6/250
8.  1/250
9.  0/250
10.  1/250
```

…… … Inputs 11 to 139 are omitted …… …



```
140.  6/250
141.  10/250
142.  6/250
143.  10/250
144.  6/250
145.  75/250
146.  6/250
147.  10/250
148.  6/250
149.  10/250
150.  6/250
151.  250/250
```

Figure 4.7 Number of iterations the training set has been presented to the ADALINE network (file OTB.out)

Figure 4.7 is the partial output generated by the ADALINE network. The first integer is the number of iterations the training set has been presented to the ADALINE network during the training. From the above output result, we can find the total number of iterations is 151, i.e. the learning of the network stops at 151st iterations. The second number contains the number of patterns the ADALINE network has correctly learned after an iteration of the pattern set. We can see from the output, once this number becomes 250, the training stage finishes. We can also see that this number is not simply approaching 250 throughout iterations, this is due to the delta rule applied to the ADALINE network training. The delta rule uses the error produced by the ADALINE to correct link values so the correct answer will be produced next time the input pattern is presented. Unfortunately, when the link value is adjusted to give the correct output for the current input pattern, it may cause other input patterns that were producing correct outputs to produce incorrect outputs.

After the training, the training results containing the Nodes and Links for ADALINE network is saved into the disk file. In Adaline1.net file, each data is mapping to a specific data member of ADALINE Nodes and Links. The following file is created through training the ADALINE network, in which the information of trained results is stored. This file will be further loaded into the memory again to initialize the ADALINE nodes and links for solving the Open-To-Buy problem.

1) Information stored for ADALINE Nodes in file "Adaline1.net"

Content in file	*description*
0	id
1	value_size
-0.377728	value[i]
1	error_size
0	error[i]
1	id
1	value_size
0.985229	value[i]
1	error_size
0	error[i]

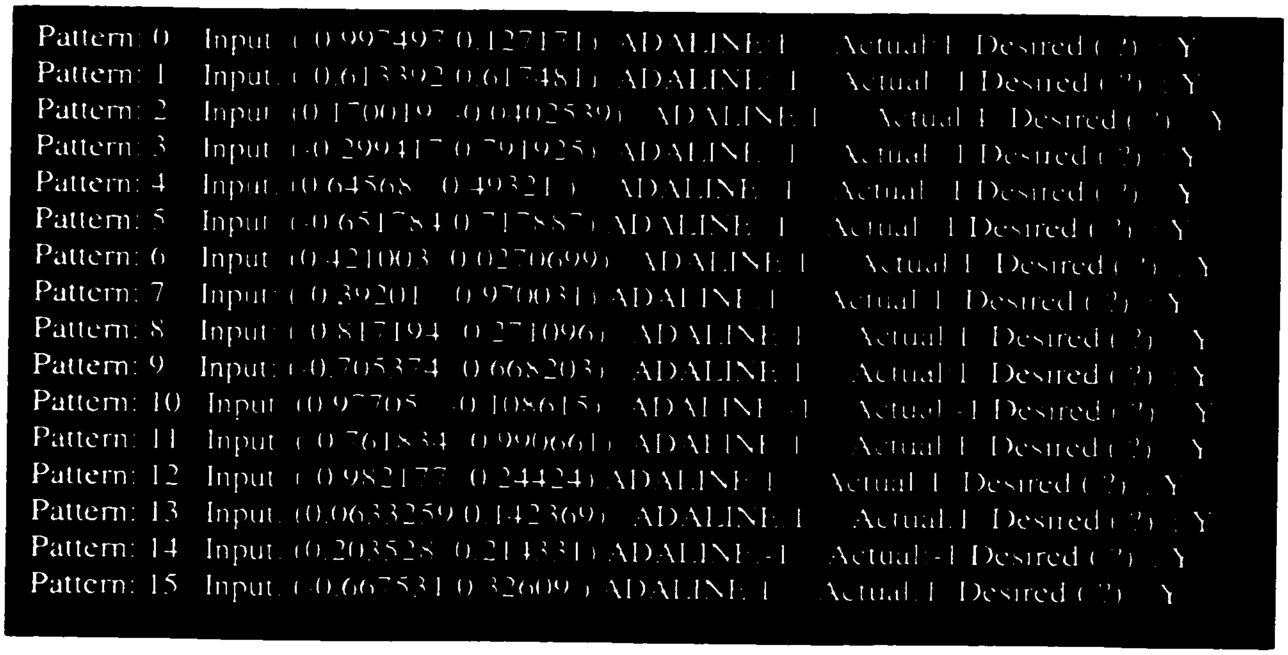
```
2                      id
1                      value_size
1                      value[i]
1                      error_size
0                      error[i]

3                      id
2                      value_size
-1  0.45               value[i]
1                      error_size
2                      error[i]
```

2) Information Stored for ADALINE Links in file Adaline1.net

Link Id	*value[WEIGHT]*	*In_Node Id*	*Out_Node Id*
0	-4.02549078222297	0	3
1	-4.10841864101352	1	3
2	2.08660847804193	2	3

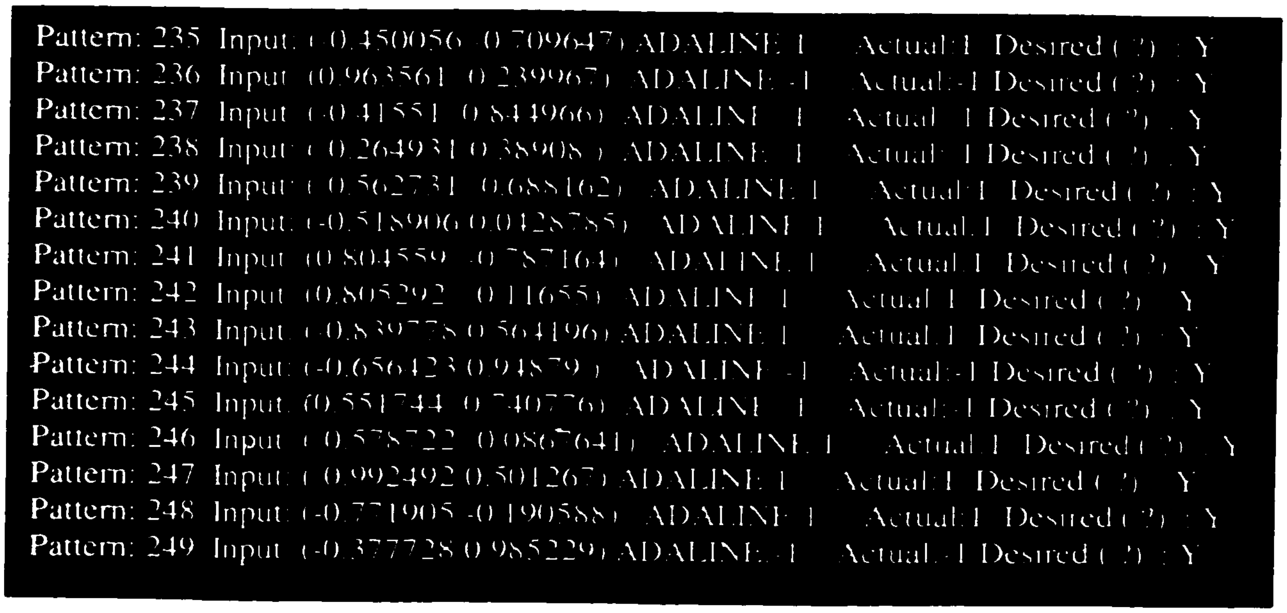
After the information necessary for the ADALINE network Nodes and Links is stored in the disk file with the contents shown above. An ADALINE network can be created in the memory first and subsequent!y be loaded by retrieving the data from the adaline1.net file.

The next step is to run the ADALINE network. The running results are shown in the following outputs.



```
Pattern: 0   Input: (0.997497 0.127171)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 1   Input: (0.613392 0.617481)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 2   Input: (0.170019 -0.0402539)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 3   Input: (-0.299417 0.791925)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 4   Input: (0.64568  0.49321 )  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 5   Input: (-0.651784 0.717887) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 6   Input: (0.421003  0.0270699)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 7   Input: ( 0.39201   0.970031) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 8   Input: ( 0.817194  0.271096)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 9   Input: (-0.705374  0.668203)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 10  Input: (0.97705  -0.108615)  ADALINE: -1   Actual: -1 Desired (?) : Y
Pattern: 11  Input: ( 0.761834  0.990661)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 12  Input: ( 0.982177  0.24424) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 13  Input: (0.0633259 0.142369)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 14  Input: (0.203528  0.214331) ADALINE: -1   Actual: -1 Desired (?) : Y
Pattern: 15  Input: (-0.667531 0.32609 ) ADALINE: 1   Actual: 1  Desired (?) : Y
```

……… **Patterns 16 to 234 are omitted** ……



```
Pattern: 235  Input: (-0.450056 -0.709647) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 236  Input: (0.963561  0.239967)  ADALINE: -1   Actual: -1 Desired (?) : Y
Pattern: 237  Input: (-0.41551  0.844966)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 238  Input: ( 0.264931 0.38908 )  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 239  Input: ( 0.562731  0.688162)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 240  Input: (-0.518906 0.0128785)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 241  Input: (0.804559  -0.787164)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 242  Input: (0.805292   0.11655)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 243  Input: (-0.839778 0.564196) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 244  Input: (-0.656423 0.94879 )  ADALINE: -1   Actual: -1 Desired (?) : Y
Pattern: 245  Input: (0.551744  0.740776)  ADALINE: 1   Actual: -1 Desired (?) : Y
Pattern: 246  Input: ( 0.578722  0.0867641)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 247  Input: ( 0.992492 0.501267) ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 248  Input: (-0.771905 -0.190588)  ADALINE: 1   Actual: 1  Desired (?) : Y
Pattern: 249  Input: (-0.377728 0.985229) ADALINE: -1   Actual: -1 Desired (?) : Y
```

Figure 4.8 Outputs of the running result of ADALINE network (file OTB.out)

4.3 Analysis of ADALINE with Example of XOR Problem

The XOR problem is an interesting example because it reveals the capability of the neural network architectures in solving problems. The XOR logic problem has a very straight-forward input pattern:

Input	*Pattern*	*Desired Output*
X	Y	
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.9 Input pattern for XOR problem

Taking the functions of the two variables as an example we can gain some insight into this problem. The following figure shows all 16 possible Boolean functions of two variables *f0 to f15*. Each column *fi* shows the value of the function for each combination of the two variables *x1* and *x2*. The function *f0*, for example, is the zero function whereas *f14* is the OR logic function.

x1	*x2*	*f0*	*f1*	*f2*	*f3*	*f4*	*f5*	*f6*	*f7*	*f8*	*f9*	*f10*	*f11*	*f12*	*f13*	*f14*	*f15*
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 4.10 All possible functions for XOR input patterns

From our proceeding example, we are clearer about the ADALINE network algorithm. The ADALINE-computable function are those for which the points whose function value is 0 can be separated from the points whose function value is 1 using a line. Figure 4.11 shows two possible separations to computer the OR and AND functions.

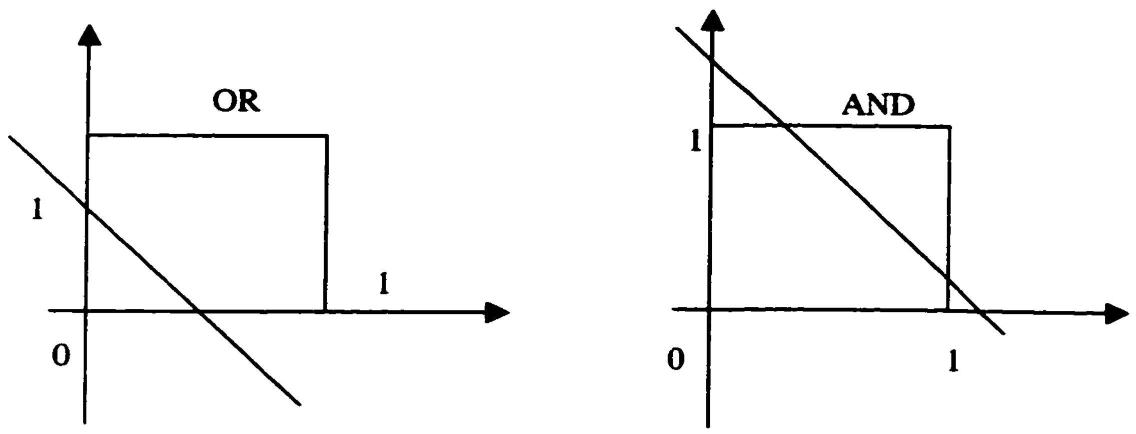


Figure 4.11 Separations of input space corresponding to OR and AND

It is clear that two of the functions in Figure 4.10 can not be computed in this way. They are the function XOR and Identity (*f6 and f9*). It is intuitively evident that no line can produce the necessary separation of the input space. This can also be shown analytically. Let w1 and w2 be the weights of a ADALINE with two inputs, T is its threshold. If the ADALINE compute the XOR functions, the following 4 inequalities must be fulfilled:

x1 = 0	x2 = 0	w1x1 + w2x2 = 0	→	0<T
x1 = 1	x2 = 0	w1x1 + w2x2 = w1	→	w1>=T
x1 = 0	x2 = 1	w1x1 + w2x2 = w2	→	w2>=T
x1 = 1	x2 = 1	w1x1 + w2x2 = w1+W2	→	w1+w2<T

Since T is positive, according to the first inequality, w1 and w2 are positive too, according to the second and the third inequalities. If we combine the second and the third inequalities, we can get w1+w2>=T. This is a contradiction with the last inequality w1+w2<T . This contradiction implies that no ADALINE capable of computing the XOR function exists. This kind of property is usually called as *Linear Separatability*. So XOR is not linear separable, thus it can not be solved by ADALINE. However, We will show that the Backpropagation network algorithm can easily solve the XOR problem. We will revisit the XOR problem later.

5.Implementing Backpropagation Neural Network

The backpropagation neural network offers a mechanism which is based on finding the outputs at the last (or output layer) of the neural network and calculating the errors or differences between the desired outputs and the current outputs. When the output is different from the desired output, changes are made in the weights, in proportion to the error between the desired output and the actual output. In Backpropagation Neural Networks, the algorithm involves making the corrections to the weights from the last-but-one layer to the last layer first, then using the calculations involved in these corrections as the basis for calculating the corrections for the next layer back... until the input layer is reached. That is the unique feature of Backpropagation Neural Network with other multi-layer networks. The basic structure of a Backpropagation Neural Network is shown in the Figure 5.1.

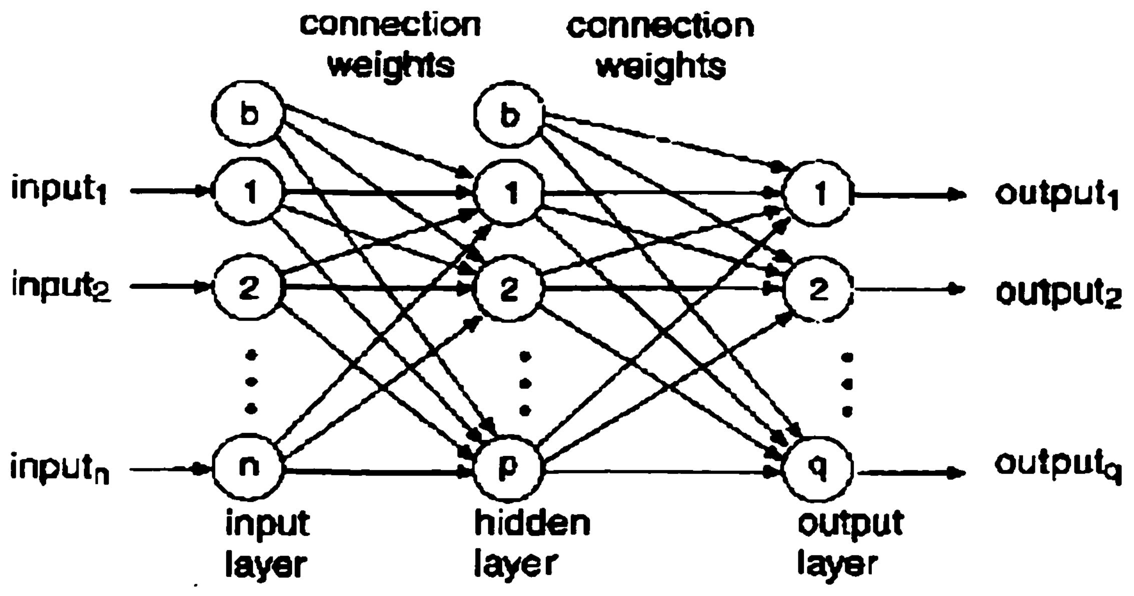


Figure 5.1 The basic structure of a Backpropagation neural network

5.1 Object-Oriented Implementation of BackPropagation Networks

5.1.1 Backpropagation Neural Network Node Classes

In designing our Backpropagation Neural Network class, we will make the use of the existing classes. The Input_Node used for ADALINE network can be used here for Backpropagation Neural Network without any modification. Similar to ADALINE_Node, the other kinds of nodes of Backpropagation Neural Network can also be derived from Feed_Forward_Node class. We show this class hierarchy in the following diagram.

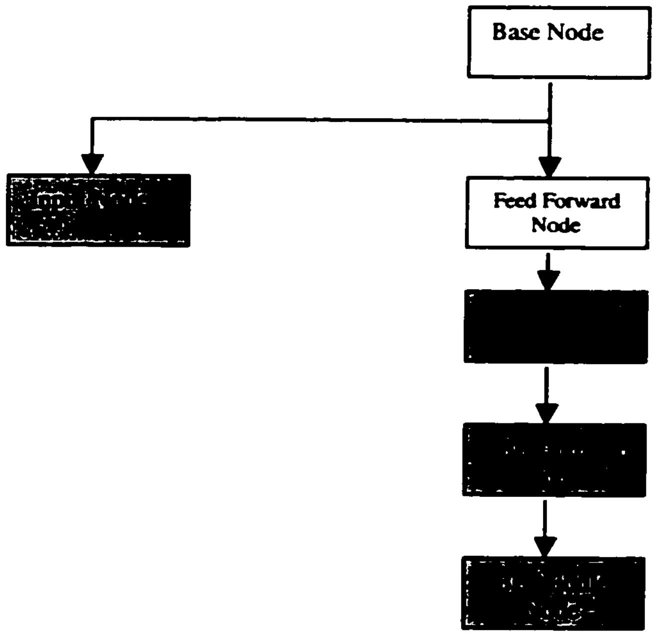


Figure 5.2 Class hierarchy for Backpropagation(BP) Neural Network nodes

The class definitions and interfaces are listed as following.

Class BP_Node

```
class BP_Node : public Feed_Forward_Node
{
    protected:
        virtual double Transfer_Function (  double value)  ;

    public:
        BP_Node (  int v_size=1, int e_size=0)  ;
                                  // Default of 1 value set member (NODE_VALUE)
        virtual char *Get_Name (  void)  ;
};
```

Class BP_Output_Node

```
class BP_Output_Node : public BP_Node
{
    public:
        BP_Output_Node (  double lr, double mt, int v_size=3, int e_size=1)  ;
                // default of 3 value set members (NODE_VALUE,
                //LEARNING_RATE, MOMENTUM)
                // default of 1 error set member (NODE_ERROR)
    protected:
        virtual double Compute_Error (  int mode=0)  ;
        virtual void Learn (  int mode=0)  ;
        virtual char *Get_Name (  void)  ;
};
```

Class BP_Middle_Node

```
class BP_Middle_Node : public BP_Output_Node
{
    public:
        BP_Middle_Node (  double lr, double mt, int v_size=3, int e_size=1)  ;
                // default of 3 value set members (NODE_VALUE,
                //LEARNING_RATE, MOMENTUM)
                // default of 1 error set member (NODE_ERROR)
        virtual char *Get_Name (  void)  ;

    protected:
        virtual double Compute_Error (  int mode=0)  ;
};
```

5.1.2 Backpropagation Neural Network Link Class

The back propagation link class can be derived from **Base_Link**. It should define enough value set members in the link objects to hold the link value (weight) and the link value's previous change (*delta*). It should also override the **save()** function. The class definitions and interfaces are listed as following.

Class BP_Link

```
class BP_Link : public Base_Link
{
    public:
        BP_Link ( int size=2) ;
                        // default of 2 link value set members (WEIGHT, DELTA)
        virtual void Save ( ofstream &outfile) ;
        virtual void Load ( ifstream &infile) ;
        virtual char *Get_Name ( void) ;
        virtual void Update_Weight ( double new_val) ;
};
```

5.2 Application of BP with Example – XOR Problem Revisited

We revisit the XOR problem encountered in our ADALINE network to study how the Backpropagation Neural Network objects can be used to solve XOR problem, whereas ADALINE can not.

We saw that ADALINE essentially carries all logical functions AND, OR, or NOT. So, If some function beyond the capability of an ADALINE can be expressed as a combination of these basic logical functions, then it could be implemented by using more than one ADALINEs. For instance, we could say that the XOR function differs from the OR function in that it excludes the case of both inputs being on. So we can say that XOR is simply:

(X1 OR X2) AND (NOT (X1 AND X2))

So, if we get one ADALINE to do the OR (X1 OR X2), others to do the NOT (X1 AND X2), and another to do the AND of these two, then XOR can be implemented. This gives us an insight of developing a multi-layer neural network to solve the problem which can not be solved by a ADALINE network.

As we mentioned earlier, the transfer function is a nonlinear function that, when applied to the net input of a node, determines the output of that node. Its domain must generally be all real numbers, as there is no theoretical limit to what the net input can be (however, in practice we can easily limit the input by limiting the weights, and often do). The range of the transfer function (i.e. values it can output) is usually limited. The most common limits are 0 to 1, while some range from –1 to 1. The ADALINE network uses a simple threshold function. If the weighted sum of inputs is less than the threshold, the output is 0. Otherwise the output is 1. To facilitate the BP algorithm, we shall introduce the Sigmoid function. In our example we use the most commonly employed sigmoid function:

$$f(x) = 1/(1+e^{-x})$$

5.3 Solving XOR Problem by Using Backpropagation Neural Network

5.3.1 XOR Problem training Set

X	Y	Output
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5.3 XOR Problem training Set

5.3.2 Creating training Set for XOR Problem

The following code segment is for creating the training set to be presented to BP network for training

```
// Create Training Set - XOR problem

Pattern *data[4];
                          // sizes  id   input     output
                          // -----  --   -----     -----
data[0]=new Pattern ( 2,1,    1,  0.0,0.0,  0.0) ;
data[1]=new Pattern ( 2,1,    2,  0.0,1.0,  1.0) ;
data[2]=new Pattern ( 2,1,    3,  1.0,0.0,  1.0) ;
data[3]=new Pattern ( 2,1,    4,  1.0,1.0,  0.0) ;
```

5.3.3 Constructing BP Network for XOR Problem

To solve XOR problem with BP algorithm, we need to construct a Backpropagation neural network with one input layer, one output layer and one hidden layer (middle layer). The topology of our BP neural network is shown as Figure 5.4.

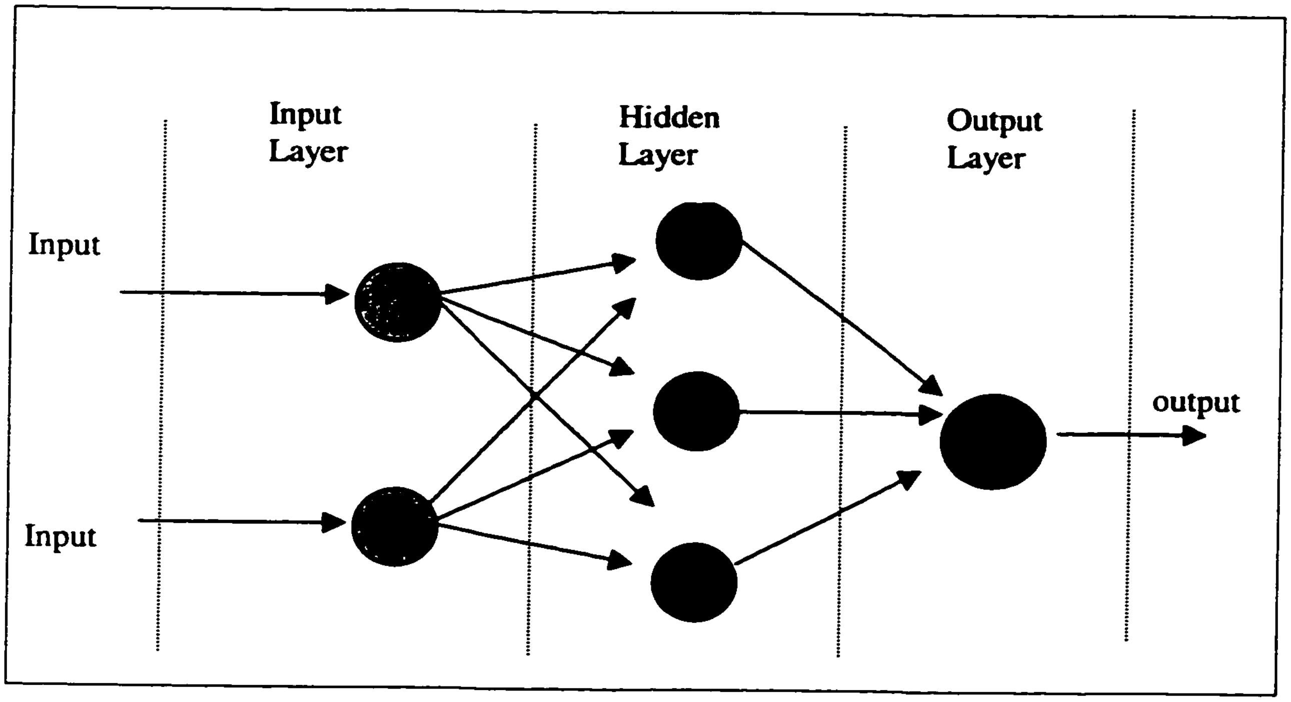


Figure 5.4 Backpropagation neural network for XOR problem

To realize this topology, the C++ implementation is:

```
// Create BP Network
    Base_Node *Node[6];
    Base_Link *Link[9];

    //1) create nodes

    // Input layer
    Node[0]=new Input_Node;
    Node[1]=new Input_Node;

    // Middle layer nodes
    // Learning rate 0.4, Momentum 0.9
    Node[2]=new BP_Middle_Node ( 0.4, 0.9) ;
    Node[3]=new BP_Middle_Node ( 0.4, 0.9) ;
    Node[4]=new BP_Middle_Node ( 0.4, 0.9) ;
    // Output layer node
    Node[5]=new BP_Output_Node ( 0.4, 0.9) ;

    //2) Create Links—connecting

    // Create Links for Network
    for (int i=0; i<9; i++)
        Link[i]=new BP_Link () ;
```

```
// Connect Network with links
int curr=0; for (i=2; i<=4; i++)
    for (int j=0; j<=1; j++)
          Connect ( Node[j],Node[i],Link[curr++]) ;
    for (int j=2; j<=4; j++)
         Connect ( Node[j],Node[5],Link[curr++]) ;
```

5.3.4 Training the BP network to Solve the XOR Problem

5.3.4.1 Training Stage

The BP network should be trained until all patterns in the training set produce an error less than 50%. Since the XOR problem has a binary output, an error tolerance of less than 50% is adequate to determine a correct output. We can see it clearer from the plot of sigmoid function, the desire outputs could take values of 0.0 or 1.0, the sigmoid function can generate two categories of outputs: one has the values ranging from 0.0 to 0.5 and the another has the values ranging from 0.5 to 1.0. These two categories can map the desired output of 1.0 and 0.0. For example, if the network generates an output of 0.34567 for input pattern (0.0, 0.0), the error is (0.34567-0.0 = 0.34567) < 0.5 , this output is fallen into the first category, can be mapped to the desired output 0.0, thus should be acceptable.

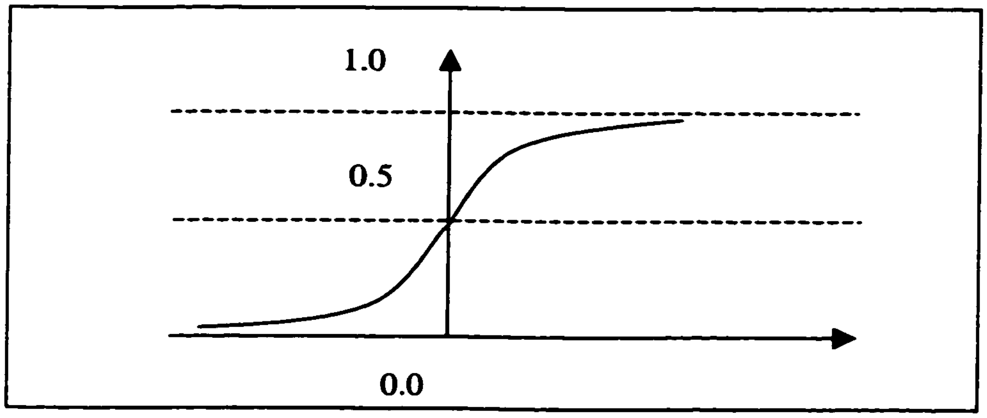


Figure 5.5 Sigmoid Function and the output mapping

Training a BP network involves several steps:

1) Set input values
2) The forward pass then is performed by executing the Run () operation on each node in the middle and output layers.
3) When the forward pass is complete, the desired output for the output node is loaded with the Set_error (..) operation
4) The backward pass is performed by excuting the Learn () operation on the nodes in the output and middle layers.

The detailed implementation of training is shown as following.

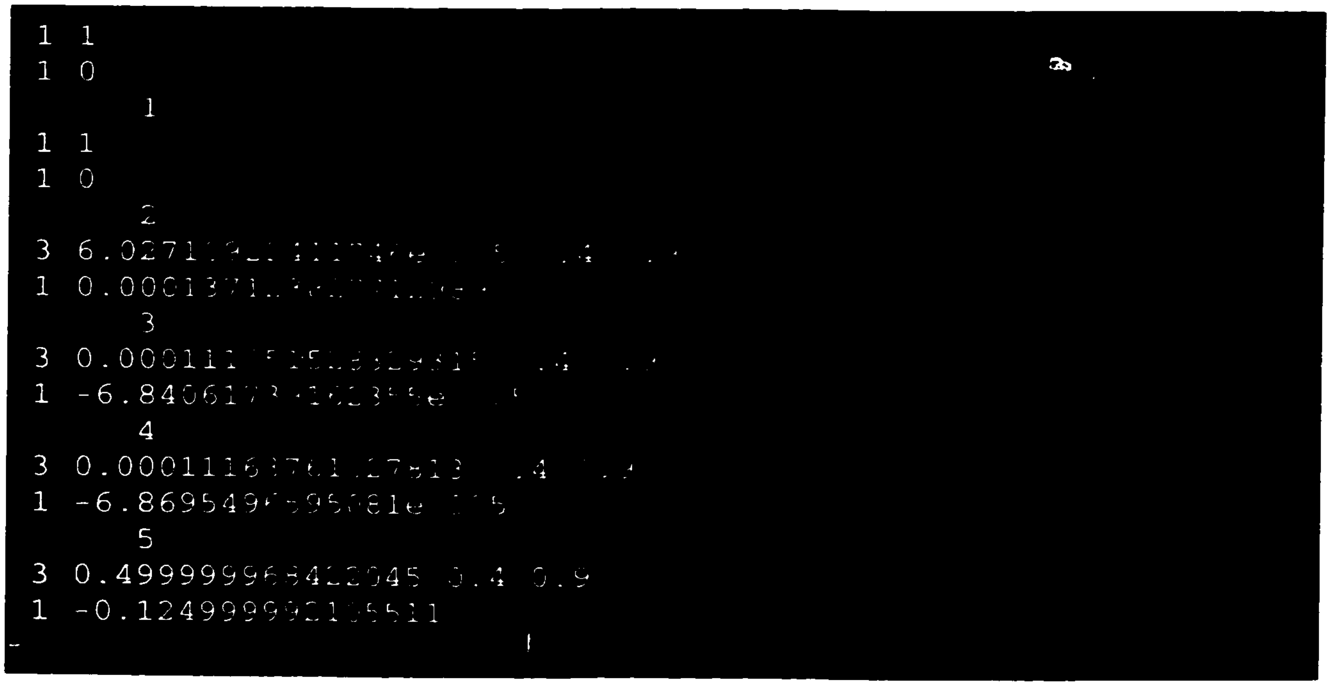
```
// Train Backprop Network
    long iteration=0;
    int good=0;
    double tolerance=0.5;
    double total_error;

    while (good<4)  // Train until all patterns are correct
        {
        good=0;
        total_error=0.0;

        for (int i=0; i<4; i++)
            {
            Node[0]->Set_Value ( data[i]->In ( 0) ) ;  // Set Input Node values
            Node[1]->Set_Value ( data[i]->In ( 1) ) ;

            for (int j=2; j<=5; j++)                   // Forward Pass
                Node[j]->Run () ;

            Node[5]->Set_Error ( data[i]->Out ( 0) ) ; // Set Error Values

            for (j=5; j>=2; j--)                       // Backward Pass
                Node[j]->Learn () ;

            if (fabs ( Node[5]->Get_Value () -data[i]->Out ( 0) ) <tolerance)
                good++;
        total_error+=fabs ( Node[5]->Get_Error ()) ;
        }
                                            // Print status every 1000 iterations
    if (iteration%1000==0) cout << iteration << ".  " << good << "/4"
                << "   Error: " << setprecision ( 15) << total_error << endl;
    iteration++;
    }
```

5.3.4.2 Save Training Results Stage:

After the training the BP network, the training result should be stored in a disk file.

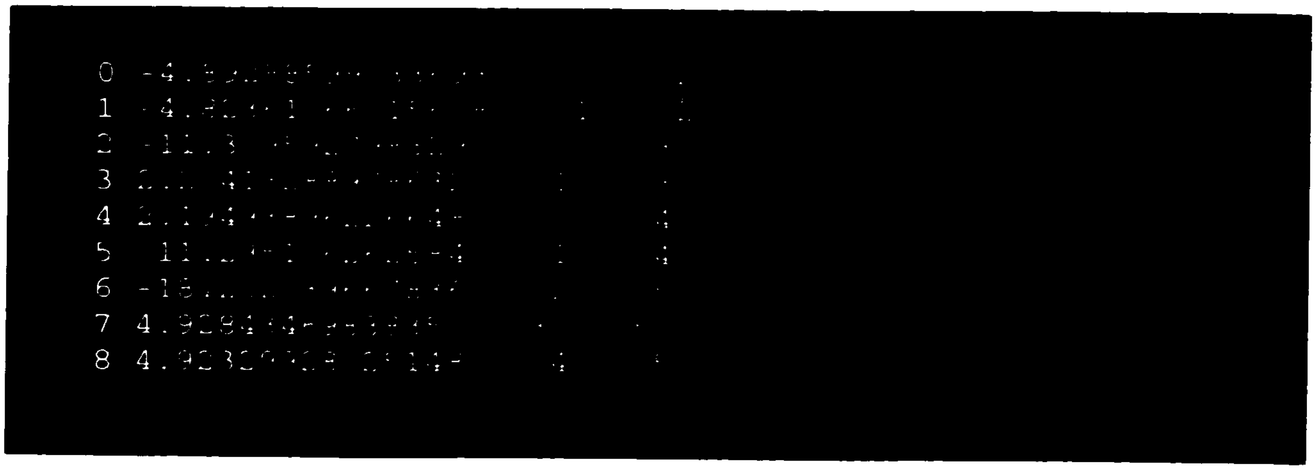
```
// Save Backprop
    ofstream outfile ( "XOR-BP.net")  ;
    for (i=0; i<6; i++)          // Save Nodes
        Node[i]->Save ( outfile)  ;

    for (i=0; i<9; i++)          // Save Links
        Link[i]->Save ( outfile)  ;

    outfile.close ()  ;
```

The content of the saved disk file "XOR-BP.net" contains the data for the BP network Nodes and Links:



```
1 1
1 0
     1
1 1
1 0
     2
3 6.0271[illegible]e[illegible] [illegible].4 [illegible]
1 0.0001371[illegible]e[illegible]
     3
3 0.000111[illegible] [illegible].4 [illegible]
1 -6.840617[illegible]e[illegible]
     4
3 0.00011163761[illegible]7513 [illegible].4 [illegible]
1 -6.869549[illegible]95[illegible]81e[illegible]5
     5
3 0.499999968412[illegible]045 0.4 0.9
1 -0.12499999210[illegible]511
```

Figure 5.6 Stored Data for Nodes



```
0 -4.[illegible]
1 [illegible]4.[illegible]
2 -11.3[illegible]
3 [illegible]
4 2.[illegible]
5  11.[illegible]
6 -1[illegible]
7 4.928[illegible]
8 4.923[illegible]
```

Figure 5.7 Stored Data for Links

5.3.4.3 Load and Run Stage

Before loading and running the BP network for the XOR problem, the Nodes and Links objects should be created first.

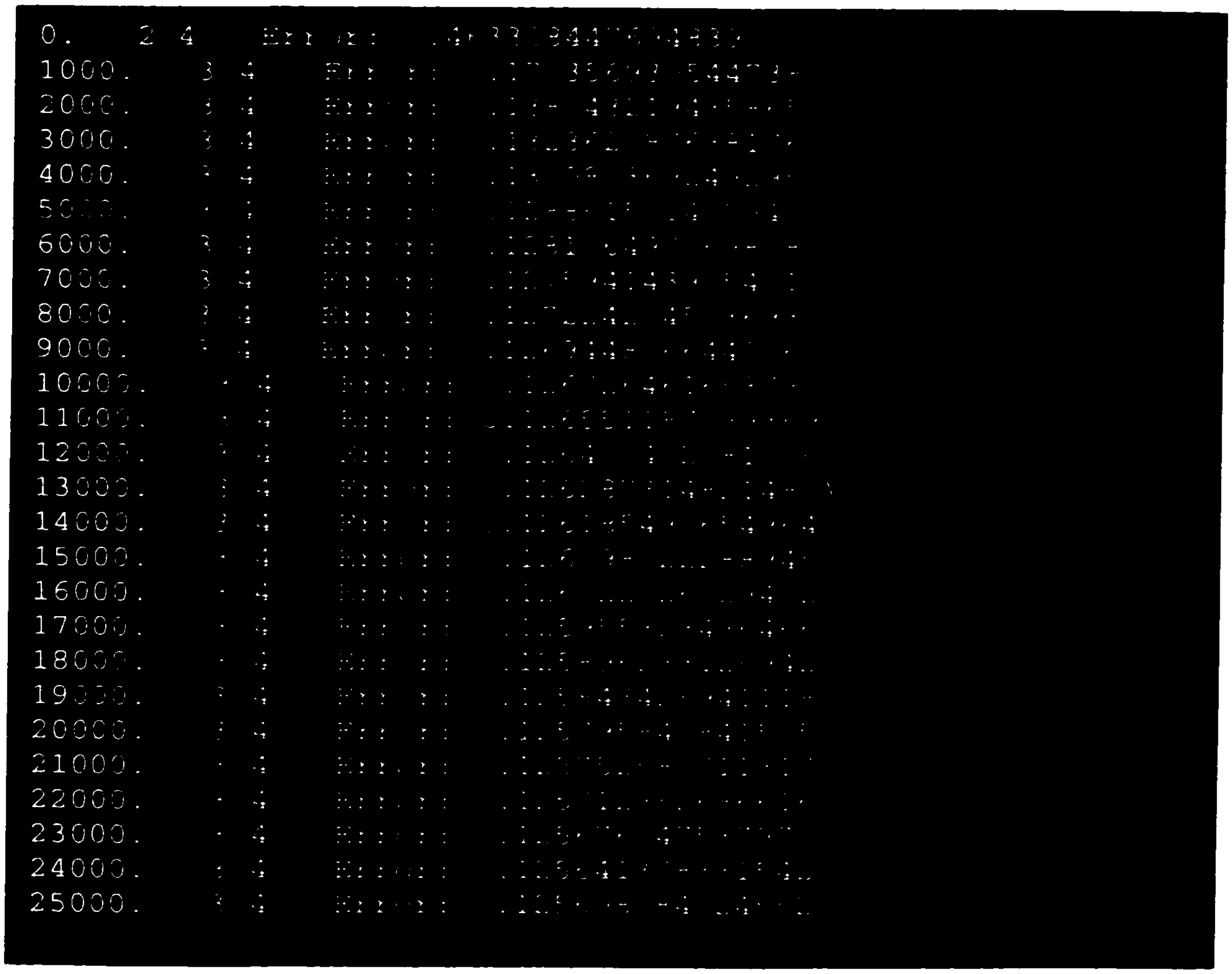
```
// Create Backprop
   Node[0]=new Input_Node;
   Node[1]=new Input_Node;
   Node[2]=new BP_Node;             // Only BP_Nodes are required
   Node[3]=new BP_Node;             // since there will be no learning
   Node[4]=new BP_Node;
   Node[5]=new BP_Node;

   for (i=0; i<9; i++)         // Create Links for Network
      Link[i]=new BP_Link () ;

   curr=0;                     // Connect Network
   for (i=2; i<=4; i++)
      for (int j=0; j<=1; j++)
         Connect ( Node[j],Node[i],Link[curr++]) ;

   for (j=2; j<=4; j++)
      Connect ( Node[j],Node[5],Link[curr++]) ;
```

Then the BP network data is loaded from the disk file XOR-Bp.net, and the running BP network is performed.

```
// Load Backprop
   ifstream infile ( "XOR-BP.net") ;
   for (i=0; i<6; i++)
        Node[i]->Load ( infile) ;

   for (i=0; i<9; i++)
      Link[i]->Load ( infile) ;

   infile.close () ;

// Run Backprop Network
   for (i=0; i<4; i++)
      {
      Node[0]->Set_Value ( data[i]->In ( 0) ) ; // Set Input Node Values
      Node[1]->Set_Value ( data[i]->In ( 1) ) ;

      for (int j=2; j<=5; j++) // Forward Pass
         Node[j]->Run () ;

      double out=Node[5]->Get_Value () ;      // Get output layer's output

      cout << "Pattern: " << setw ( 3) << i << "  Input: ("
           << data[i]->In ( 0) << ","
           << data[i]->In ( 1)
           << ")  Backprop: (" << out
           << ") Actual: (" << data[i]->Out ( 0) << ")  " << endl;
      }
```

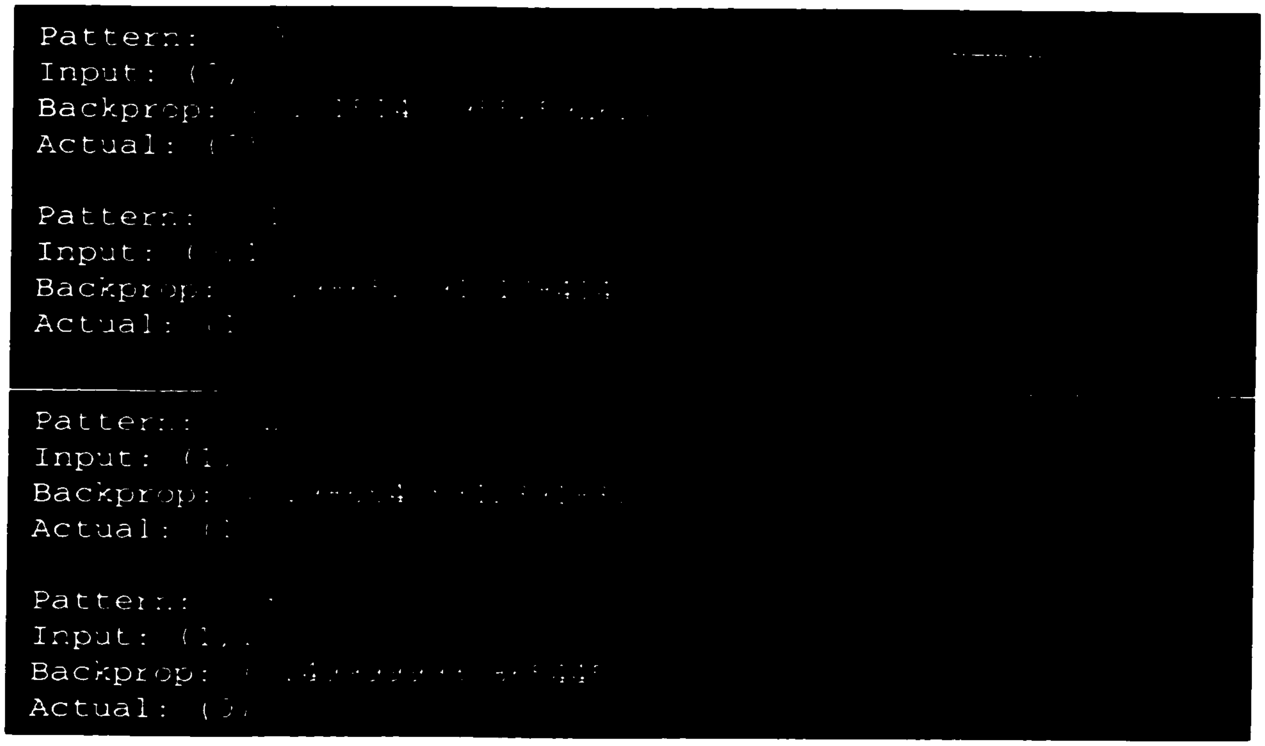
5.3.4.4 Outputs

The output training and running of BP network to solve the XOR problem is in file "XOR-BP.out". It contains the output reflecting the training process and the final running result. In figure 5.8, the output is displayed every 1000 iterations, so we did not get the output of the iteration which the training is stopped. We modify the condition of displaying outputs and get the outputs of the last two iterations: (25743. 3/4 Error: 0.125583623778577) and (25744. 4/4 Error: 0.125583590719234). We find both the training and the running results are satisfactory.



```
0.       2 4    Error:  [illegible]
1000.      3 4    Error:  [illegible]
2000.      3 4    Error:  [illegible]
3000.      3 4    Error:  [illegible]
4000.      3 4    Error:  [illegible]
5000.      [illegible] 4    Error:  [illegible]
6000.      3 4    Error:  [illegible]
7000.      3 4    Error:  [illegible]
8000.      3 4    Error:  [illegible]
9000.      3 4    Error:  [illegible]
10000.      [illegible] 4    Error:  [illegible]
11000.      [illegible] 4    Error:  [illegible]
12000.      3 4    Error:  [illegible]
13000.      3 4    Error:  [illegible]
14000.      3 4    Error:  [illegible]
15000.      [illegible] 4    Error:  [illegible]
16000.      [illegible] 4    Error:  [illegible]
17000.      [illegible] 4    Error:  [illegible]
18000.      [illegible] 4    Error:  [illegible]
19000.      3 4    Error:  [illegible]
20000.      3 4    Error:  [illegible]
21000.      [illegible] 4    Error:  [illegible]
22000.      [illegible] 4    Error:  [illegible]
23000.      [illegible] 4    Error:  [illegible]
24000.      3 4    Error:  [illegible]
25000.      3 4    Error:  [illegible]
```

Figure 5.8 Training output for XOR problem



```
Pattern: [illegible]
Input: ([illegible]
Backprop: [illegible]
Actual: ([illegible]

Pattern: [illegible]
Input: ([illegible]
Backprop: [illegible]
Actual: ([illegible]

Pattern: [illegible]
Input: (1[illegible]
Backprop: [illegible]
Actual: ([illegible]

Pattern: [illegible]
Input: (1,[illegible]
Backprop: [illegible]
Actual: ([illegible]
```

Figure 5.9 Running results output for XOR problem

6. Summary of Comparison Between ADALINE and BP

Generally speaking, we can say Backpropagation Neural Network is an extension of ADALINE network. The major differences of the architectures between them lie in the hidden layer. A BP network can have one or more hidden layers . There are several major differences between ADALINE and BP network.

- Linear separability

From XOR problem, we can see that the layered structure of BP network allows it to escape the ADALINE's linear separability limitation making it a much more powerful tool.

- Output

From our XOR problem and Open-to-Buy example, we can see that ADALINE can only give a binary output either -1 or 1 (may be 0, 1) as shown in Figure 4.8. However, a BP network does not have this limitation. It can have any number of outputs whose values fall within a continuous range. So a BP network is ideal for solving the problems involving classification, projection, interpretation and generalization.

- Inputs mappings

A single layer ADALINE network is severely limited in the mappings it can learn; a multi-layer BP network can learn any continuous mapping to an arbitrary accuracy. So of BP networks can be applied to solve problems in many areas. Applications using such networks generally involve mapping a given set of inputs to a specified set of target outputs.

7. Genetic Algorithms

7.1 Basics of Genetic Algorithms

Genetic algorithms (GAs) are search algorithms that reflect in a primitive way some of the processes of natural evolution. As such, they are analogous to neural networks' status as primitive approximations to biological neural processing.

Genetic algorithms are inspired by the study of genetics; they borrow terms from genetics and simulate the adaptive behavior of biological systems. The smallest data item that encodes information is the gene. Problem information is encoded by a sequence of genes; this ordered collection is the chromosome. The position of a specific gene in a chromosome is the locus. We allow the information in genes to mutate, that is, the information can occasionally change randomly. Mutations usually involve small changes. Another type of variation is the crossover, in which a random locus is chosen where two genes are split; the two genes then exchange the portion occurring after the locus. The formula specifying how well a given chromosome solves the formulated problem is called the fitness. Fitness is represented by a function of one argument (a chromosome) that returns a fitness value. The population is a set of chromosomes with initial random values that are used to solve a problem. The number of generations is the number of times the population is varied and the fitness values calculated in solving a problem. A complete description with directions for mutations, crossovers, and the number of generations is called an experiment.

The series of operations carried out when implementing a GA paradigm is:

1. Initialize the population,
2. Calculate fitness for each individual in the population,
3. Reproduce selected individuals to form a new population,
4. Perform crossover and mutation on the population, and
5. Loop to step 2 until some condition is met.

7.2 OO Analysis for Genetic Algorithms

To model the Genetic Algorithms, we set up four basic class, bit_vector, gene_sequence, chromosome and genetic_experiment. The relationships among these classes are in the forms of aggregation and "is-a". Chromosome is a subclass of gene_sequence, and gene_sequence is a subclass of bit_vector class. Genetic_experiment consists of chromosome. The class diagram is shown in Figure 7.1.

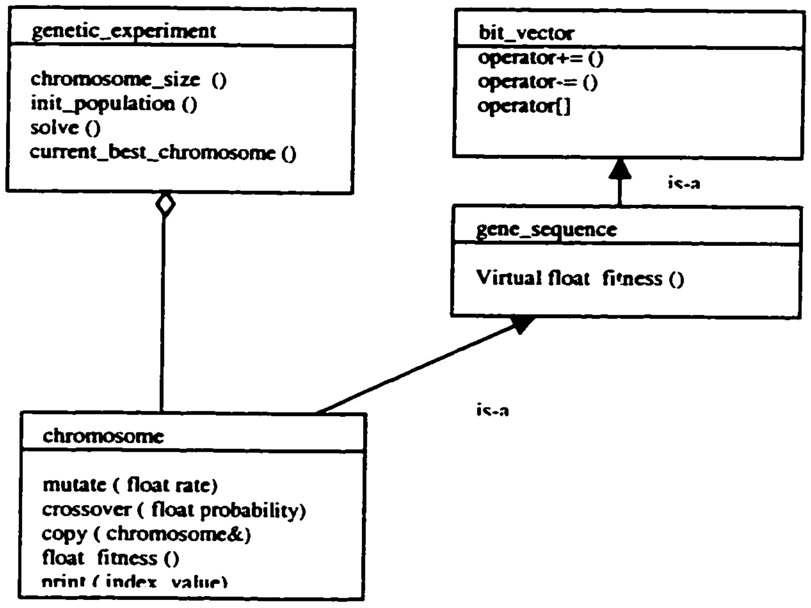


Figure 7.1 Class diagram for genetic algorithm architecture

7.3 Design and Implement Genetic Algorithm Classes in C++

The class responsibilities and interface definitions are as followings.

Class gene_sequence:

In Genetic Algorithms, individual gene in a chromosome is usually represented by a single bit. Since bit_vector objects are useful for other applications besides genetic algorithms, we design a separate class that supports setting and testing individual bits in a compact vector of bits.

```
class gene_sequence : public bit_vector
{
 public:
   gene_sequence ( int size = 1)  ;
  ~gene_sequence ()  ;
   virtual float fitness () = 0;  // abstract base class
};
```

Class chromosome:

Class chromosome is derived from class gene_sequence, adding behavior for mutation and genetic crossovers.

```
class chromosome : public gene_sequence {
 public:
   chromosome ( int size = 1)  ;
   chromosome ( chromosome & chromosome_to_copy)  ;
  ~chromosome ()  ;
   void mutate ( float mutation_rate)  ;
   void copy ( chromosome & chromosome_to_copy)  ;
   // perform "in place" without creating a new chromosome:
   void crossover ( chromosome & other_chromosome)  ;
   // Define the following member function in your application:
   float fitness ()  ;
   void print ( int index = 0, float fitness = -999.9)  ;
 private:
   RandomSequence random_seq;
};
```

Class genetic_experiment:

The class genetic_experiment controls a set of chromosomes during an experiment, including finding a single chromosome with the largest fitness value.

```
class genetic_experiment {

 public:
   genetic_experiment ( int chrom_size = -1,
                     int population_size = -1,
                     float mutation_rate = 25.0,  // 25 %
                     float crossover_probability = 0.3)  ;
  ~genetic_experiment ()  ;
   int chromosome_size () { return size_of_chromosome; }
   int number_of_chromosomes ()  ;
   virtual void initialize_population ()  ;
   void solve ()  ;
   float current_best_fitness ()  ;
   chromosome * const current_best_chromosome ()  ;
 private:
   chromosome **chromosomes;
   int size_of_population;
   int size_of_chromosome;
   int current_best_chromosome_index;
   float mut_rate;
   float cross_prob;
   float *fitnesses;
   void sort_by_fitness ()  ;
   RandomSequence randoms;
};
```

Class bit_vector

```
class bit_vector {
 public:
   bit_vector ( int num_bits = 1)  ;
  ~bit_vector ()  ;
   unsigned int operator[] ( int bit_index)  ;
   int number_of_on_bits ()  ;
   void operator+= ( int bit_to_turn_on)  ;
   void operator-= ( int bit_to_turn_off)  ;
   int operator== ( bit_vector &other)  ;
   void set_to_zero ()  ;
   void set_to_one ()  ;
   int size ()
   {
     return number_of_bits;
```

```
    }
  protected:
    unsigned int * data;
    int number_of_bits;
    int number_of_ints;
};
```

The detailed implementation of Bit_vector class is included in source listings in the Appendix. Bit_vector class will be used later for our recurrent network class (Figure 2.5), whereas, in this project, we will concentrate on the paradigm of how genetic algorithm can be applied for training recurrent neural network. We will then model the neural network in genetic algorithm architecture. In this project, we only study the genetic algorithm applied to recurrent neural network, rather than studying the general genetic algorithm and thus the detailed implementation for Genetic_experiment, Chromosome and Gene_sequence classes are omitted. The classes needed for training recurrent neural networks will be developed in later chapter.

8. Training Recurrent Neural Networks with Genetic Algorithms

Problems with continuous input data require neural networks with feedback loops between neuron layers, which are called *recurrent neural networks*. We will study how to search for a set of weights for a recurrent neural network by treating the set of weights as a chromosome, and use a genetic algorithm to find a fit chromosome that minimizes the error for a given set of neural network training data. A typical recurrent neural network is shown as figure 2.5.

8.1 Object Oriented Analysis of Recurrent Neural Network with GA

We model the neural network by applying the GA algorithm along with the feed back feature. This is a totally different architecture from that seen till now, so we need to define the classes based on this architecture rather than simply using the Base classes for feed forward ADALINE. There are three class that should be defined, they are genetic_recurrent_network class, RandomSequence class and bit_vector class. They are derivations (however we do not implement the inheritance here) of the more general Genetic Algorithm classes defined in chapter 7 (Figure 7.1). For instance, in Figure 7.1, the member function fitness() for gene class is a virtual function, and should be overridden by chromosome's fitness() member function, where the fitness() member function for the class genetic_recurrent_network is not an override function, but it is a application dependent one. The class diagram is shown in Figure 8.2.

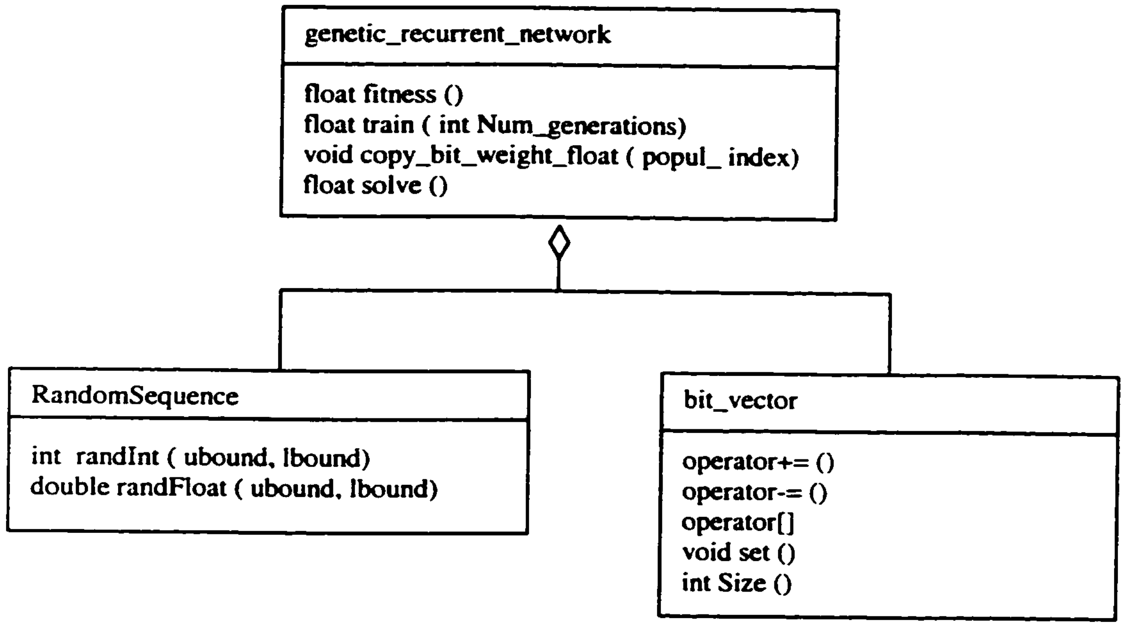

genetic_recurrent_network
float fitness ()
float train (int Num_generations)
void copy_bit_weight_float (popul_ index)
float solve ()
RandomSequence
int randInt (ubound, lbound)
double randFloat (ubound, lbound)
bit_vector
operator+= ()
operator-= ()
operator[]
void set ()
int Size ()

Figure 8.2 The class diagram for recurrent neural network

Figure 8.1 shows the class diagram for class genetic_recurrent_network. An instance of this class contains static allocation of two-dimensional weight arrays and a set of single large bit vectors (chromosomes). The population is a set of these large single bit vectors. The public member function copy_bit_weights_to_floats() unpacks the bit vector for a specified member of the chromosome population into the two dimensional floating point bit arrays. The genetic learning is implemented by genetic crossover and mutation on these bit vectors. The fitness member function evaluates each chromosome by using the two dimensional weight arrays as temporary storage to implement a neural network for each member of the chromosome population. We reuse a single set of two dimensional floating point weight arrays to evaluate all chromosomes in the population.

9. Using GA to Train Recurrent Neural Network for XOR Problem

To support genetic learning for XOR problem, we need to map a single-bit vector into the set of two-dimensional floating-point weight arrays. The size of the neural network is set by the following constant definitions:

```
const int NUM_INPUTS  = 2;  // number of input nodes is two
const int NUM_HIDDEN  = 6; //number of  nodes in hidden layer is six
const int NUM_OUTPUTS = 1; //number of output node is one.
```

The topology of the neural network is set up by defining:

```
#define USE_I_TO_O    //input to output nodes
#define USE_H_TO_H   //hidden to hidden nodes
#define USE_O_TO_H   //output to hidden nodes
```

The recurrent neural network with above macro definition is shown in the following Figure.

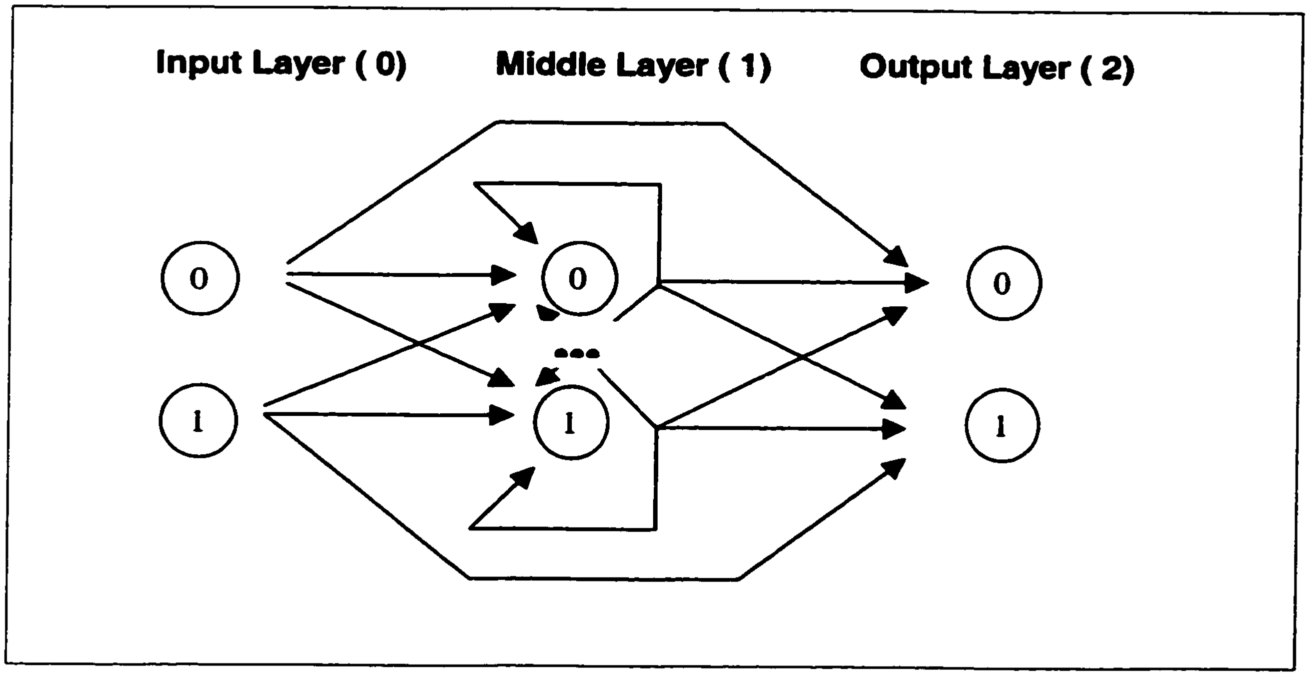


Figure 9.1 Recurrent neural network for XOR problem

If we just want implement the topology as in Figure 8.1, i.e. without the connection from input node to output node directly. We can define the second two macros without the first one:

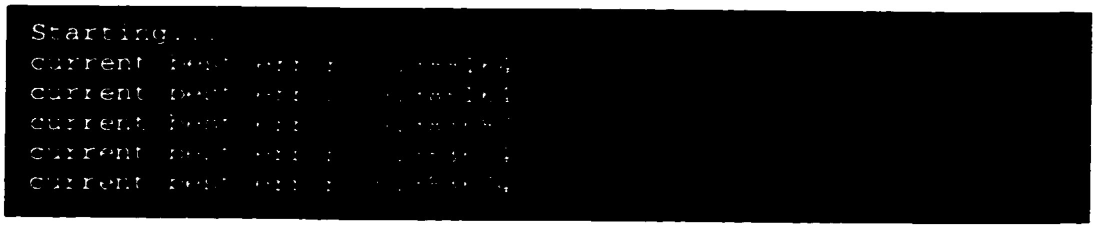
```
#define USE_H_TO_H   //hidden to hidden nodes
#define USE_O_TO_H   //output to hidden nodes
```

(no connection from input to output directly)

9.1 Execution Output for the XOR Problem with Genetic Algorithm

The followings are partial output of the XOR problem.

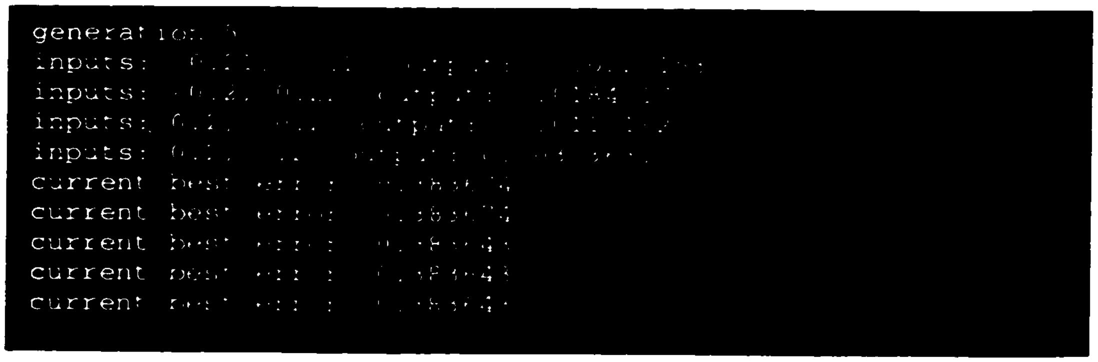
This output shows the start of an XOR genetic training experiment.



```
Starting...
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
```

Figure 9.2 Output of start of an XOR genetic training experiment

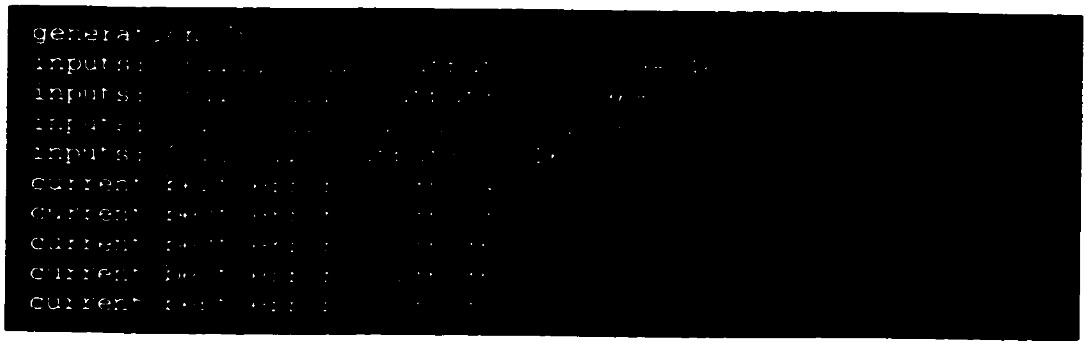
After 5 generations, the error is still large, and the best set of neural network weights in the chromosome population can not solve the XOR problem.



```
generation 5
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
current best err : [illegible]
```

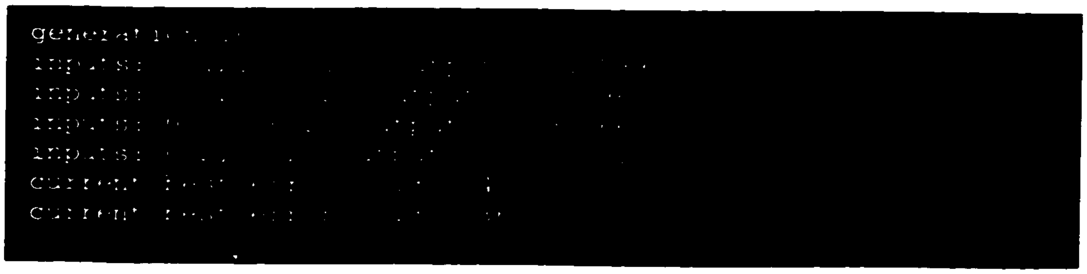
Figure 9.3 Output for generation 5 for XOR genetic training experiment

In this part of output, after 75 generation the error is smaller, but the best set of weights in the chromosome population still has not learnt to map (1, 1) to (0). The last pattern of inputs are 0.2 and 0.2, the output is 0.0160713.



```
generation [illegible]
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
current [illegible]
current [illegible]
current [illegible]
current [illegible]
current [illegible]
```

Figure 9.4 Output for generation 75 for XOR genetic training experiment



```
generation [illegible]
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
inputs: [illegible]
current [illegible]
current [illegible]
```

Figure 9.5 Output for generation 1000 for XOR genetic training experiment

In above figures of output, it shows that the best set of weights after 1000 generations has improved very little. We can see that although Genetic Algorithm is a tool to solve neural network problems, it is not efficient to solve the XOR problem. This result shows that the network required to solve the XOR problem is a simple, non-recurrent, three-layer network, i.e. the Backpropagation Algorithm we used in the proceeding XOR example. So in practice, we would not recommend using GA to solve the XOR problem.

10.Summary of Comparison between BP and GA

Genetic algorithm forms a good basis for solving some learning, optimization, and search, problems. Genetic algorithms are particularly appropriate for quickly finding good solutions to problems, but not necessarily the best solution for a given problem specification.

The advantage in using genetic algorithms lies in the ability to design arbitrarily connected neural networks, and train them, without requiring a formal scheme for adapting the connection weight values for a set of training data.

Clearly, the sample XOR program with GA do a poor job at training simple (non-recurrent) neural networks compared with the backward error propagation algorithm. However, using Genetic Algorithms to train highly recurrent neural networks looks promising.

In this project, the example we developed does not challenge the abilities of the genetic algorithm in training recurrent neural network. It just demonstrates that genetic algorithms can be applied to train recurrent neural network.

Genetic Algorithms are a kind of evolutionary computing, Backpropagation is a common algorithm for neural network computing. They can be combined together to make a more efficient way to solve neural network problems in the following ways.

- Using genetic algorithms to evolve optimal values for neural network parameters such as the number of nodes, connectivity, weights, learning rate, and momentum. Genetic algorithms can offer a mechanism to find out an optimal topology through neural network evolution.
- Using neural networks to obtain good quality initial populations for genetic algorithms.
- Using Backpropagation with genetic learning to decrease training times

- Using genetic algorithms and neural networks in parallel to solve the same problem, in order to increase the confidence in the correctness of the solutions obtained independently by each method.

References

[1] Allman, W.F. *Inside the Neural Network Revolution.* Bantam Books, New York, NY. 1989.

[2] Albrecht, R.F., C.R. Reeves, and N.C. Steele, *Artificial Neural Nets and Genetic Algorithms,* Springger-Verlag, Vienna. 1993

[3] Aleksander, I., and H. Morton, *An Introduction to Neural Computing,* Chapman and Hall, London. 1990

[4] Blum, A. *Neural Networks in C++ - An object-Oriented Framework for Building Connectionist Systems.* John Wiley & Sons. New York, NY 1992

[5] Booch, Grady. *Object-Oriented Analysis and Design with Applications.* Addison-Wesley. 1994

[6] Davis, L., Ed. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, NY. 1991

[7] Eberhart R., Simpson P. and Dobbing R. *Computational Intelligence PC Tools.* AP Professional, New York, NY. 1996

[8] Fiesler, E. et al. *Layer Based Neural Network Formalization.* In *Artificial Neural Networks II,* vol.1, pages 329-332. North-Holland/Elsevier Science Oublishers, Amsterdam, The Netherlands. 1992.

[9] Landau L.J. and Taylor J.G. *Concepts for Neural Networks – A Survey.* Springer-Verlag, New York. 1998

[10] Masters, T. *Practical Neural Network Recipes in C++*. Academic Press, San Diego, CA 1993

[11] Rao, Valluru and Hayagriva V. *C++ Neural Networks and Fuzzy Logic*. MIS Press. 1993

[12] Simpson, Patrick K. Artificial Neural System – Foundations, Paradigms, Applications and implementation. Pergamon Press, London. 1990

[13] Rogers, Samuel Joe and Antony Satyadas. *Applying a Neural Network Formalism*. Department of Computer Science, The University of Alabama, Tuscaloosa, Alabama. 1994

[14] Werbos, P. *The Roots of Backpropagation*. J. Wiley & Sons, New York, NY. 1994

[15] Whitley, D. *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA 1993

[16] White, H. Artificial Neural Networks – Approximation and Learning Theory, Blackwell, Oxford. 1992

Appendix Source Listings and Output Files

```
//*****************************************************************************
// OTB-trn-set.cpp
// program to generate training set for Open-To-Buy example
//*****************************************************************************

#include<iostream.h>
#include<fstream.h>
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<iomanip.h>

void main(){

        double InputX, InputY, VTValue, DesiredOutput;
        ofstream fout("OTBTrnSet.txt");
/*  fout<<"  "<<"   Input X Value"<< "      Input Y Value"<<"     Desired Outputs"<<endl;
        fout<<"-----------------------------------------------------------------"<<endl;
*/

        for(int i=0; i<250; i++){

                //RAND_MAX=32767 which is the max integer value


                InputX=( (double)rand()/(double)RAND_MAX )*2.0-1.0;
                InputY=( (double)rand()/ (double)RAND_MAX)*2.0-1.0;
                VTValue=((-3.0/2.0)*InputX+1)/4;

                if(InputY<VTValue)
                        DesiredOutput=1;
                else
                        DesiredOutput=-1;

                fout<<setiosflags(ios::right)<<setw(3)<<i<<"
"<<setw(18)<<InputX<<setw(20)<<InputY<<setw(10)<<DesiredOutput<<endl;
        }


        fout.close();
}
```

```
//****************************************************************************
// File base.h
//
// This header file contains the base classes for the neural network nodes
// and links. It aslo contains the commonly used doubly linked list class
//****************************************************************************
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<iostream.h>
#include<iomanip.h>
#include<fstream.h>
//-----------------------------------------------------------------
#ifndef BASE
#define BASE

#define NODE_VALUE 0
#define LEARNING_RATE 1
#define NODE_ERROR 0
#define WEIGHT 0

class Base_Node; // Forward declaration so links can use the base node type
class Base_Link  // Base Neural-Network Link class
   {
   private:

      static int ticket;

   protected:
      int id;              // ID number for link
      double *value;          // Value(s) for Link
      Base_Node *in_node;      // Node instance link is comming from
      Base_Node *out_node;      // Node instance link is going to
      int value_size;

   public:
      Base_Link( int size=1 );     // Constructor
      ~Base_Link( void );        // Destructor for Base Links
      virtual void Save( ofstream &outfile );
      virtual void Load( ifstream &infile );
      inline virtual double Get_Value( int id=WEIGHT );
      inline virtual void Set_Value( double new_val, int id=WEIGHT);
      inline virtual void Set_In_Node( Base_Node *node, int id );
      inline virtual void Set_Out_Node( Base_Node *node, int id );
      inline virtual Base_Node *In_Node( void );
      inline virtual Base_Node *Out_Node( void );
      inline virtual char *Get_Name( void );
      inline virtual void Update_Weight( double new_val );
      inline int Get_ID( void );
      inline virtual double In_Value( int mode=NODE_VALUE );
      inline virtual double Out_Value( int mode=NODE_VALUE );
      inline virtual double In_Error( int mode=NODE_ERROR );
      inline virtual double Out_Error( int mode=NODE_ERROR );
      inline virtual double Weighted_In_Value( int mode=NODE_VALUE );
      inline virtual double Weighted_Out_Value( int mode=NODE_VALUE );
      inline virtual double Weighted_In_Error( int mode=NODE_VALUE );
```

```
        inline virtual double Weighted_Out_Error( int mode=NODE_VALUE );
        inline virtual int Get_Set_Size( void );
        inline virtual void Epoch( int mode=0 );
    };
//-----------------------------------------------------------------
class LList                                    // Linked-List Support Class
    {
    private:

        struct NODE
            {
            NODE *next, *prev;
            Base_Link *element;
            };

        NODE *head,*tail,*curr;
        int count;

    public:
        LList( void );
        ~LList( void );
        int Add_To_Tail( Base_Link *element );
        int Add_Node( Base_Link *element );
        int Del_Node( void );
        int Del( Base_Link *element );
        int Find( Base_Link *element );
        inline void Clear( void );
        inline int Count( void );
        inline void Reset_To_Head( void );
        inline void Reset_To_Tail( void );
        inline Base_Link *Curr( void );
        inline void Next( void );
        inline void Prev( void );
    };
//-----------------------------------------------------------------
Base_Link::Get_Set_Size( void )
    {
    return value_size;
    }
//-----------------------------------------------------------------
Base_Link::Base_Link( int size )        // Constructor
    {
    id=++ticket;
    value_size=size;
    if (value_size<=0) value=NULL;
    else value=new double[value_size];
    for (int i=0; i<value_size; i++)    // initialize value set to zero
        value[i]=0.0;
    in_node=out_node=NULL;
    };
//-----------------------------------------------------------------
Base_Link::~Base_Link( void )           // Destructor for Base Links
    {
    if (value_size>0) delete[] value;
    };
//-----------------------------------------------------------------
```

```
void Base_Link::Save( ofstream &outfile )
   {
   outfile << id << endl;
   outfile << value_size;          // Store value set
   if (value) delete []value;
   value=new double[value_size];
   for (int i=0; i<value_size; i++)
      outfile << " " << setprecision(18) << value[i];
   outfile << endl;
   };
//----------------------------------------------------------------
void Base_Link::Load( ifstream &infile )
   {
   infile >> id;
   infile >> value_size;
   if (value) delete []value;
   value=new double[value_size];      // Load value set
   for (int i=0; i<value_size; i++)
      infile >> value[i];
   };
//----------------------------------------------------------------
double Base_Link::Get_Value( int id ) { return value[id]; };
//----------------------------------------------------------------
void Base_Link::Set_Value( double new_val, int id) { value[id]=new_val; };
//----------------------------------------------------------------
void Base_Link::Set_In_Node( Base_Node *node, int id ) { in_node=node; };
//----------------------------------------------------------------
void Base_Link::Set_Out_Node( Base_Node *node, int id ) { out_node=node; };
//----------------------------------------------------------------
Base_Node *Base_Link::In_Node( void ) { return in_node; };
//----------------------------------------------------------------
Base_Node *Base_Link::Out_Node( void ) { return out_node; };
//----------------------------------------------------------------
char *Base_Link::Get_Name( void )
   {
   static char name[]="BASE_LINK";
   return name;
   };
//----------------------------------------------------------------
void Base_Link::Update_Weight( double new_val )
   { value[WEIGHT]+=new_val; };
//----------------------------------------------------------------
int Base_Link::Get_ID( void ) { return id; };
//----------------------------------------------------------------
void Base_Link::Epoch( int code ) { };
//----------------------------------------------------------------
int Base_Link::ticket=-1;  // This static variable is shared by all
                           // links derived from the base link class. Its
                           // purpose is to give each link created from
                           // the base_link class a unique identification
                           // number.
//----------------------------------------------------------------
class Base_Node          // Base Neural-Network Node
   {
   private:
      static int ticket;
```

```
  protected:
     int id;          // Identification Number
     double *value;    // Value(s) stored by this node
     int value_size;   // Number of Values stored by this node
     double *error;    // Error value(s) stored by this node
     int error_size;   // Number of Error values stored by this node

     LList in_links;   // List for input links
     LList out_links;  // List for output links

  public:
     Base_Node( int v_size=1, int e_size=1 );   // Constructor
     ~Base_Node( void );                     // Destructor
     LList *In_Links( void );
     LList *Out_Links( void );
     virtual void Run( int mode=0 );
     virtual void Learn( int mode=0 );
     virtual void Epoch( int code=0 );
     virtual void Load( ifstream &infile );
     virtual void Save( ofstream &outfile);
     inline virtual double Get_Value( int id=NODE_VALUE );
     inline virtual void Set_Value( double new_val, int id=NODE_VALUE );
     inline virtual double Get_Error( int id=NODE_ERROR );
     inline virtual void Set_Error( double new_val, int id=NODE_ERROR );
     inline int Get_ID( void );
     inline virtual char *Get_Name( void );
     void Create_Link_To( Base_Node &to_node, Base_Link *link );
     virtual void Print( ofstream &out );

     friend void Connect( Base_Node &from_node, Base_Node &to_node,
                 Base_Link *link );
     friend void Connect( Base_Node &from_node, Base_Node &to_node,
                 Base_Link &link );
     friend void Connect( Base_Node *from_node, Base_Node *to_node,
                 Base_Link *link );
     friend int Disconnect( Base_Node *from_node, Base_Node *to_node);

     friend double Random( double lower_bound, double upper_bound );
  };
//----------------------------------------------------------------
Base_Node::Base_Node( int v_size, int e_size )    // Constructor
   {
   id=++ticket;
   if (v_size<=0) { value_size=0; value=NULL; } // Create value storage
   else
      {
      value_size=v_size;
      value=new double[v_size];
      }

   for (int i=0; i<v_size; i++)
        value[i]=0.0;

   if (e_size<=0) { error_size=0; error=NULL; } // Create error storage
   else
```

```
        {
        error_size=e_size;
        error=new double[e_size];
        }

    for (i=0; i<e_size; i++)                    // Set all errors to zero
        error[i]=0.0;
    };
//----------------------------------------------------------------
Base_Node::~Base_Node( void )                       // Destructor
    {
    if (value_size>0) delete[] value;
    if (error_size>0) delete[] error;
    };
//----------------------------------------------------------------
LList *Base_Node::In_Links( void ) { return &in_links; };
//----------------------------------------------------------------
LList *Base_Node::Out_Links( void ) { return &out_links; };
//----------------------------------------------------------------
void Base_Node::Run( int mode ) { };
//----------------------------------------------------------------
void Base_Node::Learn( int mode ) { };
//----------------------------------------------------------------
void Base_Node::Epoch( int code ) { };
//----------------------------------------------------------------
void Base_Node::Load( ifstream &infile )
    {
    infile >> id;
    infile >> value_size;
    if (value) delete []value;
    value=new double[value_size];        // Load value set
    for (int i=0; i<value_size; i++)
        infile >> value[i];
    infile >> error_size;
    if (error) delete []error;          // Load error set
    error=new double[error_size];
    for (i=0; i<error_size; i++)
        infile >> error[i];
    };
//----------------------------------------------------------------
void Base_Node::Save( ofstream &outfile)
    {
    outfile << setw(4) << id << endl;
    outfile << value_size;           // store value set
    for (int i=0; i<value_size; i++)
        outfile << " " << setprecision(18) << value[i];
    outfile << endl;
    outfile << error_size;           // store error set
    for (i=0; i<error_size; i++)
        outfile << " " << setprecision(18) << error[i];
    outfile << endl;
    };
//----------------------------------------------------------------
double Base_Node::Get_Value( int id ) { return value[id]; };
//----------------------------------------------------------------
void Base_Node::Set_Value( double new_val, int id ) { value[id]=new_val; };
```

```
//-----------------------------------------------------------------
double Base_Node::Get_Error( int id ) { return error[id]; };
//-----------------------------------------------------------------
void Base_Node::Set_Error( double new_val, int id ) { error[id]=new_val; };
//-----------------------------------------------------------------
int Base_Node::Get_ID( void ) { return id; };
//-----------------------------------------------------------------
char *Base_Node::Get_Name( void )
   {
   static char name[]="BASE_NODE";
   return name;
   };
//-----------------------------------------------------------------
void Base_Node::Create_Link_To( Base_Node &to_node, Base_Link *link )
    {
    out_links.Add_To_Tail(link);
    to_node.In_Links()->Add_To_Tail(link);
    link->Set_In_Node(this, id);
    link->Set_Out_Node(&to_node, to_node.Get_ID());
    };
//-----------------------------------------------------------------
void Connect( Base_Node &from_node, Base_Node &to_node, Base_Link *link )
    {
    from_node.Create_Link_To(to_node,link);
    };
//-----------------------------------------------------------------
void Connect( Base_Node &from_node, Base_Node &to_node, Base_Link &link )
    {
    from_node.Create_Link_To(to_node,&link);
    };
//-----------------------------------------------------------------
void Connect( Base_Node *from_node, Base_Node *to_node, Base_Link *link )
    {
    from_node->Create_Link_To(*to_node,link);
    };
//-----------------------------------------------------------------
int Disconnect( Base_Node *from_node, Base_Node *to_node ) // Remove link
   {
   LList *out_links=from_node->Out_Links();
   int flag=0;
   out_links->Reset_To_Head();
   for (int i=0; i<out_links->Count(); i++)  // for each output link
      {
      if (out_links->Curr()->Out_Node()==to_node)
         {
         flag=1;
         break;
         }
      out_links->Next();
      }

   if (flag==1)      // link exists, delete it from both nodes
      {
      out_links->Curr()->Out_Node()->In_Links()->Del(out_links->Curr());
      out_links->Del_Node();
      return 1;
```

```
        }
    else
        return 0;      // link not found
    };
//-----------------------------------------------------------------
double Random( double lower_bound, double upper_bound ) // Generate Random Number
    {
    return ((double)((rand()%RAND_MAX))/(double)RAND_MAX)*
           (upper_bound-lower_bound)+lower_bound;
    };
//-----------------------------------------------------------------
void Base_Node::Print( ofstream &out )
    {
    out << "Node ID: " << id << "   Node Name: " << Get_Name() << endl;
    out <<  "Value Set: ";
    for (int i=0; i<value_size; i++)
        out << value[i] << " ";
    out << endl;
    out <<  "Error Set: ";
    for (i=0; i<error_size; i++)
        out << error[i] << " ";
    out << endl;

    in_links.Reset_To_Head();
    for (i=0; i<in_links.Count(); i++)
        {
        out << "   In Link ID : " << in_links.Curr()->Get_ID()
             << "  Link Name: " << in_links.Curr()->Get_Name()
           << "  Source Node: " << in_links.Curr()->In_Node()->Get_ID()
           << "  Value Set: ";
        for (int j=0; j<in_links.Curr()->Get_Set_Size(); j++)
             out << in_links.Curr()->Get_Value(j) << " ";
        out << endl;
        in_links.Next();
        }

        out_links.Reset_To_Head();
        for (i=0; i<out_links.Count(); i++)
            {
            out << "   Out Link ID: " << out_links.Curr()->Get_ID()
               << "  Link Name: " << out_links.Curr()->Get_Name()
               << "  Dest Node  : " << out_links.Curr()->Out_Node()->Get_ID()
               << "  Value Set: ";
            for (int j=0; j<out_links.Curr()->Get_Set_Size(); j++)
                 out << out_links.Curr()->Get_Value(j) << " ";
            out << endl;
            out_links.Next();
            }
       out << endl;
       }
//-----------------------------------------------------------------
int Base_Node::ticket=-1;  // This static variable is shared by all
                           // links derived from the base link class. Its
                           // purpose is to give each link created from
                           // the base_link class a unique identification
                           // number.
```

```
//--------------------------------------------------------------
double Base_Link::In_Value( int mode )          // These Base_Link members
          {return in_node->Get_Value(mode);}; // must be defined after the
double Base_Link::Out_Value( int mode )         // Base_Node class because
          {return out_node->Get_Value(mode);};// they reference specific
double Base_Link::In_Error( int mode )          // Base_Node Members.
          { return in_node->Get_Error(mode);};
double Base_Link::Out_Error( int mode )
          { return out_node->Get_Error(mode);};
double Base_Link::Weighted_In_Value( int mode )
          { return in_node->Get_Value(mode)*value[WEIGHT];};
double Base_Link::Weighted_Out_Value( int mode )
          { return out_node->Get_Value(mode)*value[WEIGHT];};
double Base_Link::Weighted_Out_Error( int mode )
          { return out_node->Get_Error(mode)*value[WEIGHT];};
double Base_Link::Weighted_In_Error( int mode )
          { return in_node->Get_Error(mode)*value[WEIGHT];};

//--------------------------------------------------------------
class Feed_Forward_Node : public Base_Node  // This derived class provides
       {                           // a generic feed-forward
                                   // neural-network node which
                                   // can be used by the ADALINEs
                                   // and Backprop networks.

       protected:
          virtual double Transfer_Function( double value );

       public:
          Feed_Forward_Node( int v_size=1, int e_size=1 ); // Constructor
          virtual void Run( int mode=0 );
          virtual char *Get_Name( void );
       };
//--------------------------------------------------------------
double Feed_Forward_Node::Transfer_Function( double value ) { return value;};
//--------------------------------------------------------------
Feed_Forward_Node::Feed_Forward_Node( int v_size, int e_size ):Base_Node(v_size,e_size){};
//--------------------------------------------------------------
void Feed_Forward_Node::Run( int mode )
   {
   in_links.Reset_To_Head();
   double total=0.0;
   int cnt=in_links.Count();
   for (int i=0; i<cnt; i++)  // For each node's input link
      {
      total+=in_links.Curr()->Weighted_In_Value();
      in_links.Next();
      }
   value[mode]=Transfer_Function(total);
   };
//--------------------------------------------------------------
char *Feed_Forward_Node::Get_Name( void )
   {
   static char name[]="FEED_FORWARD_NODE";
   return name;
   };
```

```
//---------------------------------------------------------------------
class Base_Network : public Base_Node     // Base Network Node
    {
    protected:
        int num_nodes;              // Number of nodes in Network
        int num_links;              // Number of links in Network
        Base_Node **node;           // Array of base nodes
        Base_Link **link;           // Array of base links

        virtual void Create_Network( void );
        virtual void Load_Inputs( void );
        virtual void Save_Nodes_Links( ofstream &outfile );
        virtual void Load_Nodes_Links( ifstream &infile );

    public:
        Base_Network( void );       // Constructor
        ~Base_Network( void );      // Destructor
        virtual void Epoch( int code=0 );
        virtual void Print( ofstream &outfile );
        virtual char *Get_Name( void );
        };
//---------------------------------------------------------------------
void Base_Network::Create_Network( void ) { };
//---------------------------------------------------------------------
void Base_Network::Load_Inputs( void ) { };
//---------------------------------------------------------------------
void Base_Network::Save_Nodes_Links( ofstream &outfile )
    {
    outfile << num_nodes << endl;
    outfile << num_links << endl;
    for (int i=0; i<num_nodes; i++)    // Store all nodes
        node[i]->Save(outfile);
    for (i=0; i<num_links; i++)        // Store all links
        link[i]->Save(outfile);
    };
//---------------------------------------------------------------------
void Base_Network::Load_Nodes_Links( ifstream &infile )
    {
    infile >> num_nodes;
    infile >> num_links;
    Create_Network();
    for (int i=0; i<num_nodes; i++)    // Load all nodes
        node[i]->Load(infile);
    for (i=0; i<num_links; i++)        // Load all links
        link[i]->Load(infile);
    }
//---------------------------------------------------------------------
Base_Network::Base_Network( void ) : Base_Node(0,0)  // Constructor
    {
    num_nodes=0;
    num_links=0;
    node=NULL;
    link=NULL;
    };
//---------------------------------------------------------------------
Base_Network::~Base_Network( void )        // Destructor
```

```
    {
    if (node!=NULL)
        {
        for (int i=0; i<num_nodes; i++)  // Free all nodes
            delete node[i];
        for (i=0; i<num_links; i++)      // Free all links
            delete link[i];
        delete []node;
        delete []link;
        }
    };
//-----------------------------------------------------------------
void Base_Network::Print( ofstream &outfile )
    {
    for (int i=0; i<num_nodes; i++)      // Print each node in network
        node[i]->Print(outfile);
    };
//-----------------------------------------------------------------
char *Base_Network::Get_Name( void )
    {
    static char name[]="BASE_NETWORK";
    return name;
    };
//-----------------------------------------------------------------
void Base_Network::Epoch( int code )
    {
    for (int i=0; i<num_nodes; i++)  // Run Epoch for each node in network
        node[i]->Epoch( code );

    for (i=0; i<num_links; i++)      // Run Epoch for each link in network
        link[i]->Epoch( code );
    }
//-----------------------------------------------------------------
// Linked-List Constructor

LList::LList( void )
    {
    curr=head=tail=NULL;
    count=0;
    }
//-----------------------------------------------------------------
// Linked-List Destructor

LList::~LList( void )  { Clear(); }
//-----------------------------------------------------------------
int LList::Count( void ) { return count; }
//-----------------------------------------------------------------
// Clear out the contents of a list

void LList::Clear( void )
    {
    NODE *i=head, *temp;

    while (i!=NULL)
        {
        temp=i;
```

```
      i=i->next;
      delete temp;
      }
   curr=head=tail=NULL;
   count=0;
   }
//-----------------------------------------------------------------
// Add an element to the tail of a list

int LList::Add_To_Tail( Base_Link *element )
   {
   curr=NULL;
   return Add_Node( element );
   }
//-----------------------------------------------------------------
// This function add a node before the node curr points to.  If curr is
// NULL then the node is added to the tail of the list.

int LList::Add_Node(  Base_Link *element )
   {
   NODE *temp=new NODE;
   if (temp==NULL)
      {
      cout << "Unable to allocate Node..." << endl;
      }

   temp->element=element;

   if (temp==NULL) return 0;

   if (curr==NULL)          // Add to tail of list
      {
      temp->prev=tail;
      temp->next=NULL;
      if (tail==NULL)       // Empty list
         {
         head=temp;
         tail=temp;
         }
      else
         {            // Non-Empty list
         tail->next=temp;
         tail=temp;
         }
      }
   else if (curr==head)      // Add as head of list
      {
      temp->prev=NULL;
      temp->next=head;
      if (head==NULL)       // Empty List
         {
         head=temp;
         tail=temp;
         }
      else
         {
            head->prev=temp;  // Non-Empty List
            head=temp;
            }
        }
    else
        {                    // Add to Middle of List
        temp->prev=curr->prev;
        temp->next=curr;
        curr->prev->next=temp;
        curr->prev=temp;
        }
    count++;
    return 1;
    }
//------------------------------------------------------------
// Function to verify existence of element in list and returns position

int LList::Find( Base_Link *element )
    {
    NODE *temp=head;
    int cnt=1;
    curr=NULL;
    while (temp!=NULL)
        {
        if (temp->element==element)
            {
            curr=temp;
            return cnt;
            }
        cnt++;
        temp=temp->next;
        }
    return 0;
    }
//------------------------------------------------------------
// Deletes an element anywhere in a list (first occurence)

int LList::Del( Base_Link *element )
    {
    if (!Find(element)) return 0;
    return Del_Node();
    }
//------------------------------------------------------------
// This function deletes the current node in the list (curr)

int LList::Del_Node( void )
    {
    if (curr==NULL) return 0;  // If list is empty, do nothing

    delete curr->element;      // Free link object

    if (curr==head)            // Delete first node in list
        {
        if (head==tail) tail=NULL;
        else head->next->prev=NULL;
        head=curr->next;
        }
    else if (curr==tail)      // Delete last node in list
        {
        tail->prev->next=NULL;
        tail=curr->prev;
        }
    else                  // Delete node in middle of list
        {
        curr->next->prev=curr->prev;
        curr->prev->next=curr->next;
        }

    delete curr;
    curr=NULL;
    count--;
    return 1;
    }
//------------------------------------------------------------------
// Resets current node position (curr) to head of list

void LList::Reset_To_Head( void ) { curr=head; }
//------------------------------------------------------------------
// Resets current node position (curr) to head of list

void LList::Reset_To_Tail( void ) { curr=tail; }
//------------------------------------------------------------------
// Returns the Current element pointed to in the Container

Base_Link *LList::Curr( void )
    {
    if (curr==NULL) return NULL;
    else return curr->element;
    };
//------------------------------------------------------------------
// Advances current node

void LList::Next( void )
    {
    if (curr->next==NULL) curr=head;
    else curr=curr->next;
    }
//------------------------------------------------------------------
// Move the current node backwards

void LList::Prev( void )
    {
    if (curr->prev==NULL) curr=tail;
    else curr=curr->prev;
    }

#endif BASE
```

```
//***************************************************
// File  pattern.h - Input/Output Pattern Class
//***************************************************
#ifndef PATTERN
#define PATTERN

#include<stdio.h>
#include<stdlib.h>
#include<fstream.h>
#include<stdarg.h>

class Pattern
   {
   private:
         double *in_set;       // Pointer to input pattern array
         double *out_set;      // Pointer to output pattern array
         int id;               // Pattern Identification Number
         int in_size, out_size;  // Input and Output pattern sizes

   public:
         Pattern( int in, int out );
         Pattern( int in, int out, int data_id);
         Pattern( int in, int out, int data_id,
                  double *in_array, double *out_array );
         Pattern( int in, int out, ifstream &infile );

         ~Pattern( void );

         virtual inline double In(int id);
         virtual inline double Out(int id);

         virtual inline void Set_In( int id, double value );
         virtual inline void Set_Out( int id, double value );

         virtual inline int In_Size( void );
         virtual inline int Out_Size( void );

         virtual void Save( ofstream& outfile );
         virtual void Load( ifstream &infile );

         virtual void Print( void );
         virtual inline int Get_ID( void );

         virtual void Copy( Pattern &in );
   };
//-----------------------------------------------------------------------
// Constructor
Pattern::Pattern( int in, int out )
         {
         in_size=in;
         out_size=out;
         in_set=new double[in_size];
         out_set=new double[out_size];
         };
//---------------------------------------------------------------------//
```

```
 Pattern::Pattern( int in, int out, int data_id, ... )
              {
              in_size=in;
              out_size=out;
              in_set=new double[in_size];
              out_set=new double[out_size];

              id=data_id;
              va_list vl;

              va_start(vl, data_id );
              for (int i=0; i<in_size; i++)
                   in_set[i]=va_arg(vl,double);

              for (i=0; i<out_size; i++)
                   out_set[i]=va_arg(vl,double);

              va_end( vl );
              };
//--------------------------------------------------------------------

 Pattern::Pattern( int in, int out, int data_id,
             double *in_array, double *out_array )
          {
          in_size=in;
          out_size=out;
          id=data_id;

          in_set=new double[in_size];
          for (int i=0; i<in_size; i++)
              in_set[i]=in_array[i];

          out_set=new double[out_size];
          for (i=0; i<out_size; i++)
              out_set[i]=out_array[i];
          };
//--------------------------------------------------------------------
Pattern::Pattern( int in, int out, ifstream &infile )
   {
   in_size=in;
   out_size=out;
   infile >> id;

   in_set=new double[in_size];
   out_set=new double[out_size];

   Load(infile);
   };
//--------------------------------------------------------------------
// Destructor - Release in/out arrays

Pattern::~Pattern( void )
   {
   if (in_set) delete []in_set;
   if (out_set) delete []out_set;
   };
```

```
//-------------------------------------------------------------------
// Function to get Input Pattern array values

double Pattern::In(int id) { return in_set[id]; };

//-------------------------------------------------------------------
// Function to get Output Pattern array values

double Pattern::Out(int id) { return out_set[id]; };

//-------------------------------------------------------------------
// Function to set input pattern value

void Pattern::Set_In( int id, double value ) { in_set[id]=value; };

//-------------------------------------------------------------------
// Function to set output pattern value

void Pattern::Set_Out( int id, double value ){ out_set[id]=value; };

//-------------------------------------------------------------------
// Function to return input pattern size

int Pattern::In_Size( void ) { return in_size;};

//-------------------------------------------------------------------
// Function to return output pattern size

int Pattern::Out_Size( void ) { return out_size;}

//-------------------------------------------------------------------
// Function to return pattern identification number

int Pattern::Get_ID( void ) { return id; };

//-------------------------------------------------------------------
// Function to save pattern to disk

void Pattern::Save( ofstream& outfile )
     {
     outfile << id << "\t";
     for (int i=0; i<in_size; i++)
     outfile << in_set[i] << "\t";

     for ( i=0; i<out_size; i++)
        {
        outfile << out_set[i];
        if (i!=out_size-1) outfile << '\t';
        }

     outfile << endl;
     };
//-------------------------------------------------------------------
// Function to Load a pattern from disk

void Pattern::Load( ifstream &infile )
```

```
     {
     for (int i=0; i<in_size; i++)
         infile >> in_set[i];

     for (i=0; i<out_size; i++)
         infile >> out_set[i];

     char ch;
     ch=infile.peek();
     while (ch=='\n' || ch==EOF)
          {
          ch=infile.get();
          if (ch==EOF) break;
          ch=infile.peek();
          }
     };
//---------------------------------------------------------------------
// Function to print pattern

void Pattern::Print( void )
     {
     cout << "ID: " << id << "   In: ";
     for (int i=0; i<in_size; i++)
          cout << in_set[i] << " ";
     cout << "   Out: ";
     for (i=0; i<out_size; i++)
          cout << out_set[i] << " ";
     cout << endl;
     }
//---------------------------------------------------------------------
// Function to copy pattern

void Pattern::Copy( Pattern &orig )
    {
    int i;

    if (orig.In_Size()==in_size)
        for (i=0; i<in_size; i++)
            in_set[i]=orig.In(i);

    if (orig.Out_Size()==out_size)
        for (i=0; i<out_size; i++)
            out_set[i]=orig.Out(i);
    }


#endif
```

```
//**************************************************
// File common.h
//
// This header file contains some commonly used node types
//
//**************************************************

#include"base.h"

#ifndef COMMON
#define COMMON

//----------------------------------------------------------------------

class Input_Node : public Base_Node      // The Input Node class is a generic
      {                          // Input Node.  It can be used with
                                 // most networks.
      public:
         Input_Node( int size=1 ) : Base_Node(size,size)  // Default of one value
            {                              // set member (NODE_VALUE)
            for (int i=0; i<size; i++)            // and one error set
                  {                        // member (NODE_ERROR)
                  error[i]=0.0;
                  value[i]=0.0;
                  };
            };

         virtual char *Get_Name( void )
            {
            static char name[]="INPUT_NODE";
            return name;
            };
      };

//----------------------------------------------------------------------

class Bias_Node : public Input_Node    // The Bias Node Class is a node that
      {                        // always produces the same output.
                               // The Bias Node's default output is 1.0
      public:
         Bias_Node( double bias=1.0 ) : Input_Node(1)       // Constructor
            { value[0]=bias; };
         virtual void Set_Value( double value, int id=0 ){ }; // Disable Set_Value
         virtual double Get_Value( int id=0 ){ return value[0]; };

         virtual char *Get_Name( void )
            {
            static char name[]="BIAS_NODE";
            return name;
            };
      };

//----------------------------------------------------------------------

#endif
```

```
//*****************************************************************************
// File  adaline.h
//
// This header file contains the ADALINE classes
//
//*****************************************************************************
#ifndef ADALINE
#define ADALINE

#include"base.h"
#include"common.h"
#include"pattern.h"

//-----------------------------------------------------------------------------
class ADALINE_Node : public Feed_Forward_Node    // ADALINE processing Node
   {
   protected:
      virtual double Transfer_Function( double value );

   public:
      ADALINE_Node( void );
      ADALINE_Node( double lr );
      virtual void Learn( int mode );
      virtual char *Get_Name( void );
   };
//-----------------------------------------------------------------------------
double ADALINE_Node::Transfer_Function( double value )  // Threshold
   {                                  // Transfer Function
   if (value<0) return -1.0;
   else return 1.0;
   };
//-----------------------------------------------------------------------------
ADALINE_Node::ADALINE_Node( void ): Feed_Forward_Node(2,1) { }; // Constructor
//-----------------------------------------------------------------------------
ADALINE_Node::ADALINE_Node( double lr ):Feed_Forward_Node(2,1)  // Constructor with
   {                                       // Learning Rate
   value[LEARNING_RATE]=lr;                          // specified
   };
//-----------------------------------------------------------------------------
void ADALINE_Node::Learn( int mode )   // ADALINE learning function
   {
   error[NODE_ERROR]=value[NODE_VALUE]*-2.0;
   Base_Link *link;
   in_links.Reset_To_Head();
   int cnt=in_links.Count();
   double delta;
   for (int i=0; i<cnt; i++)  // for each input link
      {
      link=in_links.Curr();
      delta=value[LEARNING_RATE]*link->In_Value()*error[NODE_ERROR]; // Detla rule
      link->Update_Weight(delta);
      in_links.Next();
      }
   };
//-----------------------------------------------------------------------------
char *ADALINE_Node::Get_Name( void )
```

```
    {
    static char name[]="ADALINE_NODE";
    return name;
    };
//------------------------------------------------------------------------
class ADALINE_Link : public Base_Link          // Link for ADALINE Node
   {
   public:
      ADALINE_Link( void );
      virtual void Save( ofstream &outfile );
      virtual void Load( ifstream &infile );
      virtual char *Get_Name( void );
      };
//------------------------------------------------------------------------
ADALINE_Link::ADALINE_Link( void ) : Base_Link()  // Constructor
   {
   value[WEIGHT]=Random(-1.0,1.0);          // Automatically initialized
   }                                // weight value to random number
//------------------------------------------------------------------------
void ADALINE_Link::Save( ofstream &outfile )
   {
   outfile << setw(4) << id <<  " " << setprecision(18)
        << value[WEIGHT] << " " << setw(4) << In_Node()->Get_ID() << " "
        << setw(4) << Out_Node()->Get_ID() << endl;
   }
//------------------------------------------------------------------------
void ADALINE_Link::Load( ifstream &infile )
   {
   infile >> id;
   infile >> value[WEIGHT];
   int dummy;
   infile >> dummy;  // Skip over node IDs
   infile >> dummy;
   }
//------------------------------------------------------------------------
char *ADALINE_Link::Get_Name( void )
   {
   static char name[]="ADALINE_LINK";
   return name;
   };
//------------------------------------------------------------------------

#endif ADALINE
```

```
//**************************************************************************************
// File OTBTrnSet.txt contains the outputs of the training set to be present to ADALINE network
// for solving the Open-To-Buy example

//**************************************************************************************
 0          -0.997497           0.127171        1
 1          -0.613392           0.617481       -1
 2           0.170019          -0.0402539       1
 3          -0.299417           0.791925       -1
 4           0.64568            0.49321        -1
 5          -0.651784           0.717887       -1
 6           0.421003           0.0270699       1
 7          -0.39201           -0.970031        1
 8          -0.817194          -0.271096        1
 9          -0.705374          -0.668203        1
10           0.97705           -0.108615       -1
11          -0.761834          -0.990661        1
12          -0.982177          -0.24424         1
13           0.0633259          0.142369        1
14           0.203528           0.214331       -1
15          -0.667531           0.32609         1
16          -0.0984222         -0.295755        1
17          -0.885922           0.215369        1
18           0.566637           0.605213       -1
19           0.0397656         -0.3961          1
20           0.751946           0.453352       -1
21           0.911802           0.851436       -1
22           0.0787072         -0.715323        1
23          -0.0758385         -0.529344        1
24           0.724479          -0.580798        1
25           0.559313           0.687307       -1
26           0.993591           0.99939        -1
27           0.222999          -0.215125        1
28          -0.467574          -0.405438        1
29           0.680288          -0.952513        1
30          -0.248268          -0.814753        1
31           0.354411          -0.88757         1
32          -0.982421           0.83758        -1
33          -0.448225          -0.454207        1
34           0.175817           0.382366       -1
35           0.675222           0.452986       -1
36          -0.0301218         -0.589282        1
37           0.487472          -0.0630818       1
38          -0.0840785          0.898312       -1
39           0.488876          -0.783441        1
40           0.198096          -0.22953         1
41           0.470016           0.217933       -1
42           0.14481           -0.277322        1
43          -0.69689           -0.549791        1
44          -0.149693           0.605762       -1
45           0.0342112          0.97998        -1
46           0.503098          -0.308878        1
47          -0.662038           0.314615        1
48          -0.0162053         -0.872921        1
49           0.399518           0.00961333      1
50          -0.705008           0.899167       -1
51          -0.716849           0.810236       -1
52           0.385784          -0.393902        1
53          -0.146886          -0.859249        1
54           0.933226           0.366375       -1
55          -0.693533           0.754509       -1
56           0.643361           0.164098       -1
57          -0.617298          -0.644215        1
58           0.634388          -0.0494705       1
59          -0.688894           0.00784326      1
60           0.464034          -0.188818        1
61          -0.44084            0.137486        1
62           0.364483           0.511704       -1
63           0.443831          -0.0494095       1
64          -0.75396           -0.264382        1
65           0.669362          -0.929807        1
66           0.0340281          0.325968       -1
67          -0.147557          -0.790643        1
68           0.898679           0.842769       -1
69           0.0990936         -0.308023        1
70          -0.0565508         -0.250038        1
71           0.69396           -0.366253        1
72          -0.0878018         -0.456221        1
[illegible]  [illegible]65941  -0.404401        1
74           0.478378           0.134556       -1
75          -0.60802            0.522629       -1
76           0.678884          -0.204688        1
77           0.00180059         0.780328       -1
78          -0.945067           0.989257       -1
79           0.145177          -0.898984        1
80           0.0626545         -0.611866        1
81           0.686087           0.253517       -1
82           0.315226          -0.604297        1
83           0.684317          -0.753349        1
84          -0.780145           0.486251        1
85          -0.371868           0.882138       -1
```

86	-0.427839	-0.327372	1
87	-0.719474	0.46617	1
88	0.66924	0.415998	-1
89	0.200476	0.49443	-1
90	-0.494552	-0.711051	1
91	-0.996765	-0.877987	1
92	0.612476	0.705252	-1
93	-0.578845	-0.768792	1
94	0.106418	-0.971496	1
95	-0.772454	-0.0909757	1
96	0.50444	0.372295	-1
97	0.0868862	-0.852229	1
98	-0.12656	-0.596118	1
99	0.392438	-0.419294	1
100	-0.126621	-0.535142	1
101	0.155736	0.065157	1
102	0.257363	-0.679617	1
103	0.00827052	0.926084	-1
104	0.391522	0.849605	-1
105	-0.620106	-0.328104	1
106	-0.6433	0.990356	-1
107	-0.0851161	0.996033	-1
108	-0.804987	0.250343	1
109	-0.811213	-0.124546	1
110	0.863033	-0.903134	1
111	0.789239	-0.419965	1
112	-0.545396	0.538133	-1
113	-0.178564	-0.596057	1
114	0.256142	0.208289	-1
115	-0.0967742	-0.0672933	1
116	0.195654	0.269448	-1
117	0.709586	0.657582	-1
118	0.24955	0.441816	-1
119	0.131504	-0.249733	1
120	-0.631458	0.475814	1
121	0.110263	0.810175	-1
122	-0.514267	-0.62212	1
123	0.209449	0.397015	-1
124	0.169225	-0.297403	1
125	-0.0110782	-0.839228	1
126	0.481491	0.224097	-1
127	0.240761	0.382244	-1
128	0.609058	-0.701773	1
129	0.152074	0.735466	-1
130	0.823115	0.229408	-1
131	0.455367	-0.913572	1
132	0.335551	0.953063	-1
133	-0.369976	0.138401	1
134	-0.388348	-0.65215	1
135	-0.782891	0.73809	-1
136	0.702445	0.488632	-1
137	-0.690237	-0.346171	1
138	-0.841304	-0.846797	1
139	0.281961	0.640004	-1
140	0.0901822	-0.103488	1
141	-0.182043	-0.402509	1
142	-0.0688803	0.00241096	1
143	-0.694693	-0.353923	1
144	0.475997	-0.372234	1
145	0.653371	0.918149	-1
146	0.746696	0.450056	-1
147	-0.399884	0.887997	-1
148	-0.745537	-0.868526	1
149	0.569933	0.0491653	-1
150	0.219275	0.912229	-1
151	-0.855464	0.751274	-1
152	0.307718	-0.355754	1
153	-0.790399	0.0101016	1
154	-0.545824	-0.419416	1
155	0.839961	0.102329	-1
156	0.325602	-0.770928	1
157	-0.0149236	-0.241737	1
158	-0.00637837	0.586718	-1
159	0.0185247	-0.235267	1
160	0.376324	0.0643025	1
161	0.212561	-0.209632	1
162	-0.98822	0.415754	1
163	-0.798761	0.246132	1
164	0.726493	-0.0169988	-1
165	0.494675	-0.00619526	1
166	-0.239784	0.570727	-1
167	0.105625	-0.285806	1
168	0.911435	0.261696	-1
169	-0.64684	-0.251503	1
170	-0.736747	0.486557	1
171	0.903439	0.223975	-1
172	-0.944334	-0.340312	1
173	-0.88818	0.27842	1
174	-0.736747	0.694143	-1
175	0.728629	0.193762	-1
176	0.443281	0.707938	-1
177	-0.970641	-0.747063	1
178	0.415815	0.234291	-1
179	-0.564867	-0.868099	1
180	-0.66216	0.248207	1

181	-0.318033	-0.361187	1
182	-0.26487	0.322001	1
183	0.604785	0.613758	-1
184	0.0530717	0.222205	1
185	0.596362	0.801202	-1
186	-0.710379	0.260353	1
187	-0.195166	-0.492599	1
188	-0.72692	0.710379	-1
189	-0.867672	-0.144383	1
190	0.146702	-0.395428	1
191	0.0961028	-0.548875	1
192	-0.3773	-0.778741	1
193	0.616077	-0.730583	1
194	-0.431501	0.57622	-1
195	0.79046	0.579272	-1
196	0.487594	0.230445	-1
197	-0.277749	0.713309	-1
198	-0.543016	0.727165	-1
199	-0.541124	-0.5009	1
200	0.0848109	0.969665	-1
201	-0.892392	-0.837153	1
202	0.0493484	-0.146397	1
203	-0.810663	-0.482406	1
204	0.783074	-0.534471	1
205	-0.7069	-0.749809	1
206	0.863277	-0.839778	1
207	-0.90582	-0.882565	1
208	-0.327189	0.829402	-1
209	-0.202795	-0.134434	1
210	0.892331	0.674368	-1
211	0.068453	0.684194	-1
212	0.387066	-0.204627	1
213	-0.481674	-0.991333	1
214	0.0511795	0.909604	-1
215	-0.202612	-0.517808	1
216	0.171117	-0.489731	1
217	0.368023	0.890561	-1
218	-0.129002	0.78045	-1
219	-0.985656	0.881954	-1
220	0.203101	0.572314	-1
221	0.153356	-0.71514	1
222	-0.555345	-0.233985	1
223	-0.991455	-0.164159	1
224	-0.835505	0.319864	1
225	0.710196	-0.870296	1
226	0.62212	0.324137	-1
227	0.382977	0.605396	-1
228	0.0602741	0.371258	-1
229	-0.714469	0.379009	1
230	0.455794	0.555467	-1
231	-0.937864	0.737358	-1
232	0.289041	0.413129	-1
233	-0.829096	0.103977	1
234	0.89581	-0.882443	1
235	-0.450056	-0.709647	1
236	0.963561	0.239967	-1
237	-0.41551	0.844966	-1
238	-0.264931	0.38908	-1
239	-0.562731	-0.688162	1
240	-0.518906	0.0428785	1
241	0.804559	-0.787164	1
242	0.805292	-0.11655	1
243	-0.839778	0.564196	1
244	-0.656423	0.94879	-1
245	0.551744	0.740776	-1
246	-0.578722	-0.0867641	1
247	-0.992492	0.501267	1
248	-0.771905	-0.190588	1
249	-0.377728	0.985229	-1

```
//*******************************************************************************
// Filename: adaline.cpp
// This file conatins the codes for training, running the ADALINE network to
// solve Open-To-Buy example
//*******************************************************************************
#include<stdio.h>
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<assert.h>
#include"adaline.h"
#include"common.h"

void main( void )

   {
   srand(1);
   int i;

// Load Training Set
   Pattern *data[250];
   ifstream infile("OTBTrnSet.txt");

   if (infile.fail())
      {
      cout << "Unable to open pattern file..." << endl;
      exit(0);
      }

   for (i=0; i<250; i++)
      {
     data[i]=new Pattern(2,1,infile);
     data[i]->Print();
      }
   infile.close();

// Create ADALINE
   Base_Node *Node[4];
   Base_Link *Link[3];

   Node[0]=new Input_Node;          // Create Nodes for Network
   Node[1]=new Input_Node;
   Node[2]=new Bias_Node;
   Node[3]=new ADALINE_Node( 0.45 );  // ADALINE node with learning rate of 0.45

   Link[0]=new ADALINE_Link;         // Create Links for Network
   Link[1]=new ADALINE_Link;
   Link[2]=new ADALINE_Link;

   Connect(Node[0],Node[3], Link[0]); // Connect Network
   Connect(Node[1],Node[3], Link[1]);
   Connect(Node[2],Node[3], Link[2]);

// Train ADALINE
   int iteration=0;
int good=0;
while (good<250)              // Train until all patterns are good
      {
      good=0;
      for (int i=0; i<250; i++)
            {
            Node[0]->Set_Value(data[i]->In(0));   // Set Input Node Values
            Node[1]->Set_Value(data[i]->In(1));

            Node[3]->Run();                 // Run ADALINE Node

            if (data[i]->Out(0)!=Node[3]->Get_Value()) // If ADALINE
               {                            // produced an error
               Node[3]->Learn();                 // error, then
               break;                        // perform
               }                            // learning funct.
            else good++;
            }
      cout << iteration << ".   " << good << "/250"  << endl;
      iteration++;
      }


// Save ADALINE
   ofstream outfile("adaline1.net");

   for (i=0; i<4; i++)              // Save Nodes
        Node[i]->Save(outfile);
   for (i=0; i<3; i++)              // Save Links
         Link[i]->Save(outfile);
   outfile.close();

   for ( i=0; i<4; i++)             // Destroy Network
        delete Node[i];
   for (i=0; i<3; i++)
         delete Link[i];


// Create ADALINE
   Node[0]=new Input_Node;                // Create Nodes
   Node[1]=new Input_Node;
   Node[2]=new Bias_Node;
   Node[3]=new ADALINE_Node;

   Link[0]=new ADALINE_Link;               // Create Links
   Link[1]=new ADALINE_Link;
   Link[2]=new ADALINE_Link;

   Connect(Node[0],Node[3], Link[0]);       // Connect Network
   Connect(Node[1],Node[3], Link[1]);
   Connect(Node[2],Node[3], Link[2]);


// Load ADALINE
   infile.open("adaline1.net");
   for (i=0; i<4; i++)              // Load Nodes
```

```
        Node[i]->Load(infile);
    for (i=0; i<3; i++)                    // Load Links
          Link[i]->Load(infile);
    infile.close();


// Run ADALINE

    char AsDesired;
    for (i=0; i<250; i++)
        {
        Node[0]->Set_Value(data[i]->In(0));    // Set Input Node values
        Node[1]->Set_Value(data[i]->In(1));

        Node[3]->Run();                      // Run ADALINE node


        AsDesired='N';

        if(Node[3]->Get_Value()==data[i]->Out(0))
                             AsDesired='Y';
                   cout<<setiosflags(ios::left);
        cout<< "Pattern: " << setw(3) << i << "  Input: ("<<setw(10)
              << data[i]->In(0) <<setw(10)<<setiosflags(ios::right)
              << data[i]->In(1) << ")   ADALINE:"<<setw(10)
              << setw(3) << Node[3]->Get_Value() << "   Actual:"<<setw(10)
              << setw(3) << data[i]->Out(0) <<"Desired(?): "<<setw(4)<<AsDesired<< endl;
        }

// Destroy Network
   for ( i=0; i<4; i++)
        delete Node[i];
    for (i=0; i<3; i++)
        delete Link[i];
    }
```

```
//************************************************************************
// File adaline1.net   which contains the training  results saved, and to be used for loading the
//data of  nodes and links  needed for adaline network to run to solve the Open-To-Buy example
//************************************************************************
0   1    -0.377728          1       0
1   1     0.985229          1       0
2   1     1                       1       0
3   2     1         0.45        1       2

    0  -3.53366265222296    0    3
    1   -8.12242215101353    1    3
    2    2.08660847804193    2    3
```

```
//************************************************************************
// File OTB.out contains the outputs of  presenting training set to ADALINE,
// the number of iterations the training set has been presented to the
//ADALINE network during the training, and the running results for
//the Open-To-Buy example

//************************************************************************
ID: 0    In: -0.997497 0.127171   Out: 1
ID: 1    In: -0.613392 0.617481    Out: -1
ID: 2    In: 0.170019 -0.0402539   Out: 1
ID: 3    In: -0.299417 0.791925    Out: -1
ID: 4    In: 0.64568 0.49321       Out: -1
ID: 5    In: -0.651784 0.717887    Out: -1
ID: 6    In: 0.421003 0.0270699    Out: 1
ID: 7    In: -0.39201 -0.970031    Out: 1
ID: 8    In: -0.817194 -0.271096    Out: 1
ID: 9    In: -0.705374 -0.668203    Out: 1
ID: 10    In: 0.97705 -0.108615    Out: -1
ID: 11    In: -0.761834 -0.990661    Out: 1
ID: 12    In: -0.982177 -0.24424    Out: 1
ID: 13    In: 0.0633259 0.142369    Out: 1
ID: 14    In: 0.203528 0.214331    Out: -1
ID: 15    In: -0.667531 0.32609    Out: 1
ID: 16    In: -0.0984222 -0.295755    Out: 1
ID: 17    In: -0.885922 0.215369    Out: 1
ID: 18    In: 0.566637 0.605213    Out: -1
ID: 19    In: 0.0397656 -0.3961    Out: 1
ID: 20    In: 0.751946 0.453352    Out: -1
ID: 21    In: 0.911802 0.851436    Out: -1
ID: 22    In: 0.0787072 -0.715323    Out: 1
ID: 23    In: -0.0758385 -0.529344    Out: 1
ID: 24    In: 0.724479 -0.580798    Out: 1
ID: 25    In: 0.559313 0.687307    Out: -1
ID: 26    In: 0.993591 0.99939   Out: -1
ID: 27    In: 0.222999 -0.215125    Out: 1
ID: 28    In: -0.467574 -0.405438    Out: 1
ID: 29    In: 0.680288 -0.952513    Out: 1
ID: 30    In: -0.248268 -0.814753    Out: 1
ID: 31    In: 0.354411 -0.88757    Out: 1
ID: 32    In: -0.982421 0.83758    Out: -1
ID: 33    In: -0.448225 -0.454207    Out: 1
ID: 34    In: 0.175817 0.382366    Out: -1
ID: 35    In: 0.675222 0.452986    Out: -1
ID: 36    In: -0.0301218 -0.589282    Out: 1
ID: 37    In: 0.487472 -0.0630818    Out: 1
ID: 38    In: -0.0840785 0.898312    Out: -1
ID: 39    In: 0.488876 -0.783441    Out: 1
ID: 40    In: 0.198096 -0.22953    Out: 1
ID: 41    In: 0.470016 0.217933    Out: -1
ID: 42    In: 0.14481 -0.277322    Out: 1
ID: 43    In: -0.69689 -0.549791    Out: 1
ID: 44    In: -0.149693 0.605762    Out: -1
ID: 45    In: 0.0342112 0.97998    Out: -1
ID: 46    In: 0.503098 -0.308878    Out: 1
ID: 47    In: -0.662038 0.314615    Out: 1
ID: 48    In:    [illegible] -0.672921    Out: 1
ID: 49    In: 0.399518 0.00961333    Out: 1
ID: 50    In: -0.705008 0.899167    Out: -1
ID: 51    In: -0.716849 0.810236    Out: -1
ID: 52    In: 0.385784 -0.393902    Out: 1
ID: 53    In: -0.146886 -0.859249    Out: 1
ID: 54    In: 0.933226 0.366375    Out: -1
ID: 55    In: -0.693533 0.754509    Out: -1
ID: 56    In: 0.643361 0.164098    Out: -1
ID: 57    In: -0.617298 -0.644215    Out: 1
ID: 58    In: 0.634388 -0.0494705    Out: 1
ID: 59    In: -0.688894 0.00784326    Out: 1
ID: 60    In: 0.464034 -0.188818    Out: 1
```

```
ID: 61    In: -0.44084 0.137486    Out: 1
ID: 62    In: 0.364483 0.511704    Out: -1
ID: 63    In: 0.443831 -0.0494095    Out: 1
ID: 64    In: -0.75396 -0.264382    Out: 1
ID: 65    In: 0.669362 -0.929807    Out: 1
ID: 66    In: 0.0340281 0.325968    Out: -1
ID: 67    In: -0.147557 -0.790643    Out: 1
ID: 68    In: 0.898679 0.842769    Out: -1
ID: 69    In: 0.0990936 -0.308023    Out: 1
ID: 70    In: -0.0565508 -0.250038    Out: 1
ID: 71    In: 0.69396 -0.366253    Out: 1
ID: 72    In: -0.0878018 -0.456221    Out: 1
ID: 73    In: 0.965941 -0.404401    Out: 1
ID: 74    In: 0.478378 0.134556    Out: -1
ID: 75    In: -0.60802 0.522629    Out: -1
ID: 76    In: 0.678884 -0.204688    Out: 1
ID: 77    In: 0.00180059 0.780328    Out: -1
ID: 78    In: -0.945067 0.989257    Out: -1
ID: 79    In: 0.145177 -0.898984    Out: 1
ID: 80    In: 0.0626545 -0.611866    Out: 1
ID: 81    In: 0.686087 0.253517    Out: -1
ID: 82    In: 0.315226 -0.604297    Out: 1
ID: 83    In: 0.684317 -0.753349    Out: 1
ID: 84    In: -0.780145 0.486251    Out: 1
ID: 85    In: -0.371868 0.882138    Out: -1
ID: 86    In: -0.427839 -0.327372    Out: 1
ID: 87    In: -0.719474 0.46617    Out: 1
ID: 88    In: 0.66924 0.415998    Out: -1
ID: 89    In: 0.200476 0.49443    Out: -1
ID: 90    In: -0.494552 -0.711051    Out: 1
ID: 91    In: -0.996765 -0.877987    Out: 1
ID: 92    In: 0.612476 0.705252    Out: -1
ID: 93    In: -0.578845 -0.768792    Out: 1
ID: 94    In: 0.106418 -0.971496    Out: 1
ID: 95    In: -0.772454 -0.0909757    Out: 1
ID: 96    In: 0.50444 0.372295    Out: -1
ID: 97    In: 0.0868862 -0.852229    Out: 1
ID: 98    In: -0.12656 -0.596118    Out: 1
ID: 99    In: 0.392438 -0.419294    Out: 1
ID: 100    In: -0.126621 -0.535142    Out: 1
ID: 101    In: 0.155736 0.065157    Out: 1
ID: 102    In: 0.257363 -0.679617    Out: 1
ID: 103    In: 0.00827052 0.926084    Out: -1
ID: 104    In: 0.391522 0.849605    Out: -1
ID: 105    In: -0.620106 -0.328104    Out: 1
ID: 106    In: -0.6433 0.990356    Out: -1
ID: 107    In: -0.0851161 0.996033    Out: -1
ID: 108    In: -0.804987 0.250343    Out: 1
ID: 109    In: -0.811213 -0.124546    Out: 1
ID: 110    In: 0.863033 -0.903134    Out: 1
ID: 111    In: 0.789239 -0.419965    Out: 1
ID: 112    In: -0.545396 0.538133    Out: -1
ID: 113    In: -0.178564 -0.596057    Out: 1
ID: 114    In: 0.256142 0.208289    Out: -1
ID: 115    In: -0.0967742 -0.0672933    Out: 1
ID: 116    In: 0.195654 0.269448    Out: -1
ID: 117    In: 0.709586 0.657582    Out: -1
ID: 118    In: 0.24955 0.441816    Out: -1
ID: 119    In: 0.131504 -0.249733    Out: 1
ID: 120    In: -0.631458 0.475814    Out: 1
ID: 121    In: 0.110263 0.810175    Out: -1
ID: 122    In: -0.514267 -0.62212    Out: 1
ID: 123    In: 0.209449 0.397015    Out: -1
ID: 124    In: 0.169225 -0.297403    Out: 1
ID: 125    In: -0.0110782 -0.839228    Out: 1
ID: 126    In: 0.481491 0.224097    Out: -1
ID: 127    In: 0.240761 0.382244    Out: -1
ID: 128    In: 0.609058 -0.701773    Out: 1
ID: 129    In: 0.152074 0.735466    Out: -1
ID: 130    In: 0.823115 0.229408    Out: -1
ID: 131    In: 0.455367 -0.913572    Out: 1
ID: 132    In: 0.335551 0.953063    Out: -1
ID: 133    In: -0.369976 0.138401    Out: 1
ID: 134    In: -0.388348 -0.65215    Out: 1
ID: 135    In: -0.782891 0.73809    Out: -1
ID: 136    In: 0.702445 0.488632    Out: -1
ID: 137    In: -0.690237 -0.346171    Out: 1
ID: 138    In: -0.841304 -0.846797    Out: 1
ID: 139    In: 0.281961 0.640004    Out: -1
ID: 140    In: 0.0901822 -0.103488    Out: 1
ID: 141    In: -0.182043 -0.402509    Out: 1
ID: 142    In: [illegible] 0.00241096    Out: 1
ID: 143    In: -0.694093 -0.353923    Out: 1
ID: 144    In: 0.475997 -0.372234    Out: 1
ID: 145    In: 0.653371 0.918149    Out: -1
ID: 146    In: 0.746696 0.450056    Out: -1
ID: 147    In: -0.399884 0.887997    Out: -1
ID: 148    In: -0.745537 -0.868526    Out: 1
ID: 149    In: 0.569933 0.0491653    Out: -1
ID: 150    In: 0.219275 0.912229    Out: -1
ID: 151    In: -0.855464 0.751274    Out: -1
ID: 152    In: 0.307718 -0.355754    Out: 1
ID: 153    In: -0.790399 0.0101016    Out: 1
ID: 154    In: -0.545824 -0.419416    Out: 1
ID: 155    In: 0.839961 0.102329    Out: -1
```

```
ID: 156    In: 0.325602 -0.770928     Out: 1
ID: 157    In: -0.0149236 -0.241737     Out: 1
ID: 158    In: -0.00637837 0.586718     Out: -1
ID: 159    In: 0.0185247 -0.235267     Out: 1
ID: 160    In: 0.376324 0.0643025     Out: 1
ID: 161    In: 0.212561 -0.209632     Out: 1
ID: 162    In: -0.98822 0.415754     Out: 1
ID: 163    In: -0.798761 0.246132     Out: 1
ID: 164    In: 0.726493 -0.0169988     Out: -1
ID: 165    In: 0.494675 -0.00619526     Out: 1
ID: 166    In: -0.239784 0.570727     Out: -1
ID: 167    In: 0.105625 -0.285806     Out: 1
ID: 168    In: 0.911435 0.261696     Out: -1
ID: 169    In: -0.64684 -0.251503     Out: 1
ID: 170    In: -0.736747 0.486557     Out: 1
ID: 171    In: 0.903439 0.223975     Out: -1
ID: 172    In: -0.944334 -0.340312     Out: 1
ID: 173    In: -0.88818 0.27842     Out: 1
ID: 174    In: -0.736747 0.694143     Out: -1
ID: 175    In: 0.728629 0.193762     Out: -1
ID: 176    In: 0.443281 0.707938     Out: -1
ID: 177    In: -0.970641 -0.747063     Out: 1
ID: 178    In: 0.415815 0.234291     Out: -1
ID: 179    In: -0.564867 -0.868099     Out: 1
ID: 180    In: -0.66216 0.248207     Out: 1
ID: 181    In: -0.318033 -0.361187     Out: 1
ID: 182    In: -0.26487 0.322001     Out: 1
ID: 183    In: 0.604785 0.613758     Out: -1
ID: 184    In: 0.0530717 0.222205     Out: 1
ID: 185    In: 0.596362 0.801202     Out: -1
ID: 186    In: -0.710379 0.260353     Out: 1
ID: 187    In: -0.195166 -0.492599     Out: 1
ID: 188    In: -0.72692 0.710379     Out: -1
ID: 189    In: -0.867672 -0.144383     Out: 1
ID: 190    In: 0.146702 -0.395428     Out: 1
ID: 191    In: 0.0961028 -0.548875     Out: 1
ID: 192    In: -0.3773 -0.778741     Out: 1
ID: 193    In: 0.616077 -0.730583     Out: 1
ID: 194    In: -0.431501 0.57622     Out: -1
ID: 195    In: 0.79046 0.579272     Out: -1
ID: 196    In: 0.487594 0.230445     Out: -1
ID: 197    In: -0.277749 0.713309     Out: -1
ID: 198    In: -0.543016 0.727165     Out: -1
ID: 199    In: -0.541124 -0.5009     Out: 1
ID: 200    In: 0.0848109 0.969665     Out: -1
ID: 201    In: -0.892392 -0.837153     Out: 1
ID: 202    In: 0.0493484 -0.146397     Out: 1
ID: 203    In: -0.810663 -0.482406     Out: 1
ID: 204    In: 0.783074 -0.534471     Out: 1
ID: 205    In: -0.7069 -0.749809     Out: 1
ID: 206    In: 0.863277 -0.839778     Out: 1
ID: 207    In: -0.90582 -0.882565     Out: 1
ID: 208    In: -0.327189 0.829402     Out: -1
ID: 209    In: -0.202795 -0.134434     Out: 1
ID: 210    In: 0.892331 0.674368     Out: -1
ID: 211    In: 0.068453 0.684194     Out: -1
ID: 212    In: 0.387066 -0.204627     Out: 1
ID: 213    In: -0.481674 -0.991333     Out: 1
ID: 214    In: 0.0511795 0.909604     Out: -1
ID: 215    In: -0.202612 -0.517808     Out: 1
ID: 216    In: 0.171117 -0.489731     Out: 1
ID: 217    In: 0.368023 0.890561     Out: -1
ID: 218    In: -0.129002 0.78045     Out: -1
ID: 219    In: -0.985656 0.881954     Out: -1
ID: 220    In: 0.203101 0.572314     Out: -1
ID: 221    In: 0.153356 -0.71514     Out: 1
ID: 222    In: -0.555345 -0.233985     Out: 1
ID: 223    In: -0.991455 -0.164159     Out: 1
ID: 224    In: -0.835505 0.319864     Out: 1
ID: 225    In: 0.710196 -0.870296     Out: 1
ID: 226    In: 0.62212 0.324137     Out: -1
ID: 227    In: 0.382977 0.605396     Out: -1
ID: 228    In: 0.0602741 0.371258     Out: -1
ID: 229    In: -0.714469 0.379009     Out: 1
ID: 230    In: 0.455794 0.555467     Out: -1
ID: 231    In: -0.937864 0.737358     Out: -1
ID: 232    In: 0.289041 0.413129     Out: -1
ID: 233    In: -0.829096 0.103977     Out: 1
ID: 234    In: 0.89581 -0.882443     Out: 1
ID: 235    In: -0.450056 -0.709647     Out: 1
ID: 236    In: 0.963561 0.239967     Out: -1
ID: 237    In: -0.41551 0.844966     Out: -1
ID: 238    In: -0.264931 0.38908     Out: -1
ID: 239    In: -0.562731 -0.688162     Out: 1
ID: 240    In: -0.518906 0.0428785     Out: 1
ID: 241    In: 0.804559 -0.787164     Out: 1
ID: 242    In: 0.805292 -0.11655     Out: 1
ID: 243    In: -0.839778 0.564196     Out: 1
ID: 244    In: -0.656423 0.94879     Out: -1
ID: 245    In: 0.551744 0.740776     Out: -1
ID: 246    In: -0.578722 -0.0867641     Out: 1
ID: 247    In: -0.992492 0.501267     Out: 1
ID: 248    In: -0.771905 -0.190588     Out: 1
ID: 249    In: -0.377728 0.985229     Out: -1
0.    1/250
```

1.	0/250
2.	1/250
3.	0/250
4.	2/250
5.	1/250
6.	2/250
7.	6/250
8.	1/250
9.	0/250
10.	1/250
11.	10/250
12.	2/250
13.	6/250
14.	1/250
15.	0/250
16.	1/250
17.	10/250
18.	2/250
19.	6/250
20.	10/250
21.	6/250
22.	1/250
23.	6/250
24.	10/250
25.	6/250
26.	10/250
27.	6/250
28.	10/250
29.	6/250
30.	1/250
31.	6/250
32.	10/250
33.	6/250
34.	10/250
35.	6/250
36.	1/250
37.	6/250
38.	10/250
39.	6/250
40.	10/250
41.	6/250
42.	1/250
43.	6/250
44.	10/250
45.	6/250
46.	10/250
47.	6/250
48.	75/250
49.	6/250
50.	10/250
51.	6/250
52.	10/250
53.	6/250
54.	75/250
55.	6/250
56.	10/250
57.	6/250
58.	10/250
59.	6/250
60.	75/250
61.	6/250
62.	10/250
63.	6/250
64.	10/250
65.	6/250
66.	58/250
67.	1/250
68.	10/250
69.	6/250
70.	10/250
71.	6/250
72.	184/250
73.	1/250
74.	10/250
75.	6/250
76.	120/250
77.	1/250
78.	120/250
79.	1/250
80.	120/250
81.	1/250
82.	120/250
83.	1/250
84.	84/250
85.	1/250
86.	120/250
87.	1/250
88.	87/250
89.	1/250
90.	87/250
91.	1/250
92.	6/250
93.	10/250
94.	6/250
95.	10/250

```
96.   6/250
97.   1/250
98.   6/250
99.   10/250
100.   6/250
101.   10/250
102.   6/250
103.   1/250
104.   6/250
105.   10/250
106.   6/250
107.   10/250
108.   6/250
109.   14/250
110.   6/250
111.   14/250
112.   6/250
113.   10/250
114.   6/250
115.   75/250
116.   6/250
117.   10/250
118.   6/250
119.   10/250
120.   6/250
121.   75/250
122.   6/250
123.   10/250
124.   6/250
125.   75/250
126.   6/250
127.   10/250
128.   6/250
129.   10/250
130.   6/250
131.   164/250
132.   6/250
133.   75/250
134.   6/250
135.   10/250
136.   6/250
137.   10/250
138.   6/250
139.   75/250
140.   6/250
141.   10/250
142.   6/250
143.   10/250
144.   6/250
145.   75/250
146.   6/250
147.   10/250
148.   6/250
149.   10/250
150.   6/250
151.   250/250
Pattern: 0    Input: (-0.997497 0.127171  )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 1    Input: (-0.613392 0.617481  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 2    Input: (0.170019  -0.0402539)    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 3    Input: (-0.299417 0.791925  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 4    Input: (0.64568   0.49321   )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 5    Input: (-0.651784 0.717887  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 6    Input: (0.421003  0.0270699 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 7    Input: (-0.39201  -0.970031 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 8    Input: (-0.817194 -0.271096 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 9    Input: (-0.705374 -0.668203 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 10   Input: (0.97705   -0.108615 )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 11   Input: (-0.761834 -0.990661 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 12   Input: (-0.982177 -0.24424  )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 13   Input: (0.0633259 0.142369  )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 14   Input: (0.203528  0.214331  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 15   Input: (-0.667531 0.32609   )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 16   Input: (-0.0984222-0.295755 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 17   Input: (-0.885922 0.215369  )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 18   Input: (0.566637  0.605213  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 19   Input: (0.0397656 -0.3961   )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 20   Input: (0.751946  0.453352  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 21   Input: (0.911802  0.851436  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 22   Input: (0.0787072 -0.715323 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 23   Input: (-0.0758385-0.529344 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 24   Input: (0.724479  -0.580798 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 25   Input: (0.559313  0.687307  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 26   Input: (0.993591  0.99939   )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 27   Input: (0.222999  -0.215125 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 28   Input: (-0.467574 -0.405438 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 29   Input: (0.680288  -0.952513 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 30   Input: (-0.248268 -0.814753 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 31   Input: (0.354411  -0.88757  )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 32   Input: (-0.982421 0.83758   )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 33   Input: (-0.448225 -0.454207 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 34   Input: (0.175817  0.382366  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 35   Input: (0.675222  0.452986  )    ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 36   Input: (-0.0301218-0.589282 )    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 37   Input: (0.487472  -0.0630818)    ADALINE:1     Actual:1  Desired(?): Y
Pattern: 38   Input: (-0.08407850.898312  )    ADALINE:-1    Actual:-1 Desired(?): Y
```

```
Pattern: 39    Input: (0.488876  -0.783441 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 40    Input: (0.198096  -0.22953  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 41    Input: (0.470016  0.217933  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 42    Input: (0.14481   -0.277322 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 43    Input: (-0.69689  -0.549791 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 44    Input: (-0.149693 0.605762  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 45    Input: (0.0342112 0.97998   )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 46    Input: (0.503098  -0.308878 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 47    Input: (-0.662038 0.314615  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 48    Input: (-0.0162053-0.872921 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 49    Input: (0.399518  0.00961333)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 50    Input: (-0.705008 0.899167  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 51    Input: (-0.716849 0.810236  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 52    Input: (0.385784  -0.393902 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 53    Input: (-0.146886 -0.859249 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 54    Input: (0.933226  0.366375  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 55    Input: (-0.693533 0.754509  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 56    Input: (0.643361  0.164098  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 57    Input: (-0.617298 -0.644215 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 58    Input: (0.634388  -0.0494705)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 59    Input: (-0.688894 0.00784326)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 60    Input: (0.464034  -0.188818 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 61    Input: (-0.44084  0.137486  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 62    Input: (0.364483  0.511704  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 63    Input: (0.443831  -0.0494095)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 64    Input: (-0.75396  -0.264382 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 65    Input: (0.669362  -0.929807 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 66    Input: (0.0340281 0.325968  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 67    Input: (-0.147557 -0.790643 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 68    Input: (0.898679  0.842769  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 69    Input: (0.0990936 -0.308023 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 70    Input: (-0.0565508-0.250038 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 71    Input: (0.69396   -0.366253 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 72    Input: (-0.0878018-0.456221 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 73    Input: (0.965941  -0.404401 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 74    Input: (0.478378  0.134556  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 75    Input: (-0.60802  0.522629  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 76    Input: (0.678884  -0.204688 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 77    Input: (0.001800590.780328  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 78    Input: (-0.945067 0.989257  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 79    Input: (0.145177  -0.898984 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 80    Input: (0.0626545 -0.611866 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 81    Input: (0.686087  0.253517  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 82    Input: (0.315226  -0.604297 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 83    Input: (0.684317  -0.753349 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 84    Input: (-0.780145 0.486251  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 85    Input: (-0.371868 0.882138  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 86    Input: (-0.427839 -0.327372 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 87    Input: (-0.719474 0.46617   )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 88    Input: (0.66924   0.415998  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 89    Input: (0.200476  0.49443   )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 90    Input: (-0.494552 -0.711051 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 91    Input: (-0.996765 -0.877987 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 92    Input: (0.612476  0.705252  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 93    Input: (-0.578845 -0.768792 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 94    Input: (0.106418  -0.971496 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 95    Input: (-0.772454 -0.0909757)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 96    Input: (0.50444   0.372295  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 97    Input: (0.0868862 -0.852229 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 98    Input: (-0.12656  -0.596118 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 99    Input: (0.392438  -0.419294 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 100   Input: (-0.126621 -0.535142 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 101   Input: (0.155736  0.065157  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 102   Input: (0.257363  -0.679617 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 103   Input: (0.008270520.926084  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 104   Input: (0.391522  0.849605  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 105   Input: (-0.620106 -0.328104 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 106   Input: (-0.6433   0.990356  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 107   Input: (-0.08511610.996033  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 108   Input: (-0.804987 0.250343  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 109   Input: (-0.811213 -0.124546 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 110   Input: (0.863033  -0.903134 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 111   Input: (0.789239  -0.419965 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 112   Input: (-0.545396 0.538133  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 113   Input: (-0.178564 -0.596057 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 114   Input: (0.256142  0.208289  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 115   Input: (-0.0967742-0.0672933)   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 116   Input: (0.195654  0.269448  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 117   Input: (0.709586  0.657582  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 118   Input: (0.24955   0.441816  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 119   Input: (0.131504  -0.249733 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 120   Input: (-0.631458 0.475814  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 121   Input: (0.110263  0.810175  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 122   Input: (-0.514267 -0.62212  )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 123   Input: (0.209449  0.397015  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 124   Input: (0.169225  -0.297403 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 125   Input: (-0.0110782-0.839228 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 126   Input: (0.481491  0.224097  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 127   Input: (0.240761  0.382244  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 128   Input: (0.609058  -0.701773 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 129   Input: (0.152074  0.735466  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 130   Input: (0.823115  0.229408  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 131   Input: (0.455367  -0.913572 )   ADALINE:1      Actual:1  Desired(?): Y
Pattern: 132   Input: (0.335551  0.953063  )   ADALINE:-1     Actual:-1 Desired(?): Y
Pattern: 133   Input: (-0.369976 0.138401  )   ADALINE:1      Actual:1  Desired(?): Y
```

```
Pattern: 134  Input: (-0.388348 -0.65215  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 135  Input: (-0.782891 0.73809   )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 136  Input: (0.702445  0.488632  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 137  Input: (-0.690237 -0.346171 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 138  Input: (-0.841304 -0.846797 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 139  Input: (0.281961  0.640004  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 140  Input: (0.0901822 -0.103488 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 141  Input: (-0.182043 -0.402509 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 142  Input: (-0.06888030.00241096)  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 143  Input: (-0.694693 -0.353923 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 144  Input: (0.475997  -0.372234 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 145  Input: (0.653371  0.918149  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 146  Input: (0.746696  0.450056  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 147  Input: (-0.399884 0.887997  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 148  Input: (-0.745537 -0.868526 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 149  Input: (0.569933  0.0491653 )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 150  Input: (0.219275  0.912229  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 151  Input: (-0.855464 0.751274  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 152  Input: (0.307718  -0.355754 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 153  Input: (-0.790399 0.0101016 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 154  Input: (-0.545824 -0.419416 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 155  Input: (0.839961  0.102329  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 156  Input: (0.325602  -0.770928 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 157  Input: (-0.0149236-0.241737 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 158  Input: (-0.006378370.586718  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 159  Input: (0.0185247 -0.235267 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 160  Input: (0.376324  0.0643025 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 161  Input: (0.212561  -0.209632 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 162  Input: (-0.98822  0.415754  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 163  Input: (-0.798761 0.246132  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 164  Input: (0.726493  -0.0169988)  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 165  Input: (0.494675  -0.00619526)  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 166  Input: (-0.239784 0.570727  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 167  Input: (0.105625  -0.285806 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 168  Input: (0.911435  0.261696  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 169  Input: (-0.64684  -0.251503 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 170  Input: (-0.736747 0.486557  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 171  Input: (0.903439  0.223975  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 172  Input: (-0.944334 -0.340312 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 173  Input: (-0.88818  0.27842   )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 174  Input: (-0.736747 0.694143  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 175  Input: (0.728629  0.193762  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 176  Input: (0.443281  0.707938  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 177  Input: (-0.970641 -0.747063 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 178  Input: (0.415815  0.234291  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 179  Input: (-0.564867 -0.868099 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 180  Input: (-0.66216  0.248207  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 181  Input: (-0.318033 -0.361187 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 182  Input: (-0.26487  0.322001  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 183  Input: (0.604785  0.613758  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 184  Input: (0.0530717 0.222205  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 185  Input: (0.596362  0.801202  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 186  Input: (-0.710379 0.260353  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 187  Input: (-0.195166 -0.492599 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 188  Input: (-0.72692  0.710379  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 189  Input: (-0.867672 -0.144383 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 190  Input: (0.146702  -0.395428 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 191  Input: (0.0961028 -0.548875 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 192  Input: (-0.3773   -0.778741 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 193  Input: (0.616077  -0.730583 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 194  Input: (-0.431501 0.57622   )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 195  Input: (0.79046   0.579272  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 196  Input: (0.487594  0.230445  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 197  Input: (-0.277749 0.713309  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 198  Input: (-0.543016 0.727165  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 199  Input: (-0.541124 -0.5009   )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 200  Input: (0.0848109 0.969665  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 201  Input: (-0.892392 -0.837153 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 202  Input: (0.0493484 -0.146397 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 203  Input: (-0.810663 -0.482406 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 204  Input: (0.783074  -0.534471 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 205  Input: (-0.7069   -0.749809 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 206  Input: (0.863277  -0.839778 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 207  Input: (-0.90582  -0.882565 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 208  Input: (-0.327189 0.829402  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 209  Input: (-0.202795 -0.134434 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 210  Input: (0.892331  0.674368  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 211  Input: (0.068453  0.684194  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 212  Input: (0.387066  -0.204627 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 213  Input: (-0.481674 -0.991333 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 214  Input: (0.0511795 0.909604  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 215  Input: (-0.202612 -0.517808 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 216  Input: (0.171117  -0.489731 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 217  Input: (0.368023  0.890561  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 218  Input: (-0.129002 0.78045   )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 219  Input: (-0.985656 0.881954  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 220  Input: (0.203101  0.572314  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 221  Input: (0.153356  -0.71514  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 222  Input: (-0.555345 -0.233985 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 223  Input: (-0.991455 -0.164159 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 224  Input: (-0.835505 0.319864  )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 225  Input: (0.710196  -0.870296 )  ADALINE:1    Actual:1  Desired(?): Y
Pattern: 226  Input: (0.62212   0.324137  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 227  Input: (0.382977  0.605396  )  ADALINE:-1   Actual:-1 Desired(?): Y
Pattern: 228  Input: (0.0602741 0.371258  )  ADALINE:-1   Actual:-1 Desired(?): Y
```

```
Pattern: 229  Input: (-0.714469 0.379009  )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 230  Input: (0.455794  0.555467  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 231  Input: (-0.937864 0.737358  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 232  Input: (0.289041  0.413129  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 233  Input: (-0.829096 0.103977  )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 234  Input: (0.89501   -0.882443 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 235  Input: (-0.450056 -0.709647 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 236  Input: (0.963561  0.239967  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 237  Input: (-0.41551  0.844966  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 238  Input: (-0.264931 0.38908   )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 239  Input: (-0.562731 -0.688162 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 240  Input: (-0.518906 0.0428785 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 241  Input: (0.804559  -0.787164 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 242  Input: (0.805292  -0.11655  )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 243  Input: (-0.839778 0.564196  )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 244  Input: (-0.656423 0.94879   )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 245  Input: (0.551744  0.740776  )   ADALINE:-1    Actual:-1 Desired(?): Y
Pattern: 246  Input: (-0.578722 -0.0867641)   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 247  Input: (-0.992492 0.501267  )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 248  Input: (-0.771905 -0.190588 )   ADALINE:1     Actual:1  Desired(?): Y
Pattern: 249  Input: (-0.377728 0.985229  )   ADALINE:-1    Actual:-1 Desired(?): Y
```

```
//*************************************************************************
// file backprop.h  -  Backprop base classses
//*************************************************************************
#ifndef BACKPROP
#define BACKPROP

#include"base.h"
#include"common.h"
#define DELTA 1
#define MOMENTUM 2
//-----------------------------------------------------------------
class BP_Node : public Feed_Forward_Node
   {
   protected:
      virtual double Transfer_Function( double value );

   public:
      BP_Node( int v_size=1, int e_size=0 ); // Default of 1 value set member (NODE_VALUE)
      virtual char *Get_Name( void );
   };
//-----------------------------------------------------------------
class BP_Link : public Base_Link
   {
   public:
      BP_Link( int size=2 ); // default of 2 link value set members (WEIGHT, DELTA)
      virtual void Save( ofstream &outfile );
      virtual void Load( ifstream &infile );
      virtual char *Get_Name( void );
      virtual void Update_Weight( double new_val );
   };
//-----------------------------------------------------------------
class BP_Output_Node : public BP_Node
   {
   public:
      BP_Output_Node( double lr, double mt, int v_size=3, int e_size=1 );
                  // default of 3 value set members (NODE_VALUE, LEARNING_RATE, MOMENTUM)
                  // default of 1 error set member (NODE_ERROR)
   protected:
      virtual double Compute_Error( int mode=0 );
      virtual void Learn( int mode=0 );
      virtual char *Get_Name( void );
   };
//-----------------------------------------------------------------
class BP_Middle_Node : public BP_Output_Node
   {
   public:
      BP_Middle_Node( double lr, double mt, int v_size=3, int e_size=1 );
                  // default of 3 value set members (NODE_VALUE, LEARNING_RATE, MOMENTUM)
                  // default of 1 error set member (NODE_ERROR)

      virtual char *Get_Name( void );

   protected:
      virtual double Compute_Error( int mode=0 );
   };
//-----------------------------------------------------------------
```

```
double BP_Node::Transfer_Function( double value )
   {
   return 1.0/(1.0+exp(-value));        // Sigmoid Function
   };
//----------------------------------------------------------------
BP_Node::BP_Node( int v_size, int e_size ): Feed_Forward_Node(v_size,e_size) { };
//----------------------------------------------------------------
char *BP_Node::Get_Name( void )
   {
   static char name[]="BP_NODE";
   return name;
   };
//----------------------------------------------------------------
BP_Link::BP_Link( int size ) : Base_Link(size)
   {
   value[WEIGHT]=Random(-1.0,1.0);  // Weight random value between -1.0 and 1.0
   value[DELTA]=0.0;              // Initialize previous change to 0.0
   }
//----------------------------------------------------------------
void BP_Link::Save( ofstream &outfile )
   {
   outfile << setw(4) << id << " " << setprecision(18)
           << value[WEIGHT] << " " << setw(4)
           << In_Node()->Get_ID() << " "
           << setw(4) << Out_Node()->Get_ID() << endl;
   }
//----------------------------------------------------------------
void BP_Link::Load( ifstream &infile )
   {
   infile >> id;
   infile >> value[WEIGHT];
   int dummy;
   infile >> dummy;  // Skip over connecting node IDs
   infile >> dummy;
   }
//----------------------------------------------------------------
char *BP_Link::Get_Name( void )
   {
   static char name[]="BP_LINK";
   return name;
   };
//----------------------------------------------------------------
void BP_Link::Update_Weight( double new_val )
   {
   double momentum=Out_Node()->Get_Value(MOMENTUM);
   value[WEIGHT]+=new_val+(momentum*value[DELTA]);  // Update weight with current change
                                                     // and percent of last change
   value[DELTA]=new_val;                             // Store current change for next time
   }
//----------------------------------------------------------------
BP_Output_Node::BP_Output_Node( double lr, double mt, int v_size, int e_size ):
                 BP_Node(v_size,e_size)
   {
   value[LEARNING_RATE]=lr;
   value[MOMENTUM]=mt;
   };
```

```
//------------------------------------------------------------
double BP_Output_Node::Compute_Error( int mode )
   {
   return value[NODE_VALUE]*(1.0-value[NODE_VALUE])*  // Compute output node error
          (error[NODE_ERROR]-value[NODE_VALUE]);
   };
//------------------------------------------------------------
void BP_Output_Node::Learn( int mode )
   {
   double delta;
   error[NODE_ERROR]=Compute_Error();

   in_links.Reset_To_Head();
   Base_Link *link;
   int cnt=in_links.Count();

   for (int i=0; i<cnt; i++)  // For each input link
       {
       link=in_links.Curr();
       delta=value[LEARNING_RATE]*error[NODE_ERROR]*link->In_Value(); // Delta Rule
       link->Update_Weight(delta);
       in_links.Next();
       }
   };
//------------------------------------------------------------
char *BP_Output_Node::Get_Name( void )
   {
   static char name[]="BP_OUTPUT_NODE";
   return name;
   };
//------------------------------------------------------------
BP_Middle_Node::BP_Middle_Node( double lr, double mt, int v_size, int e_size ):
                    BP_Output_Node(lr,mt,v_size, e_size){ };
//------------------------------------------------------------
char *BP_Middle_Node::Get_Name( void )
   {
   static char name[]="BP_MIDDLE_NODE";
   return name;
   };
//------------------------------------------------------------
double BP_Middle_Node::Compute_Error( int mode )
   {
   double total=0;
   out_links.Reset_To_Head();
   int cnt=out_links.Count();
   for (int i=0; i<cnt; i++)  // For each of the node's output links
      {
      total+=out_links.Curr()->Weighted_Out_Error();
      out_links.Next();
      }
   return value[NODE_VALUE]*(1.0-value[NODE_VALUE])*total;
   };
//------------------------------------------------------------

#endif
```

```
//******************************************************************************
// File backpro1.cpp
//******************************************************************************
#include<stdio.h>
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<assert.h>
#include<time.h>
#include<stdlib.h>

#include"backprop.h"
#include"common.h"
#include"base.h"
#include"pattern.h"

void main( void )
   {
   srand(1);

// Create Training Set - XOR problem
   Pattern *data[4];

               // sizes id   input    output
               // -----  --   -----    ------
   data[0]=new Pattern(2,1,  1,  0.0,0.0,  0.0 );
   data[1]=new Pattern(2,1,  2,  0.0,1.0,  1.0 );
   data[2]=new Pattern(2,1,  3,  1.0,0.0,  1.0 );
   data[3]=new Pattern(2,1,  4,  1.0,1.0,  0.0 );

// Create Backprop Network
   Base_Node *Node[6];
   Base_Link *Link[9];

   Node[0]=new Input_Node;                      // Input layer nodes
   Node[1]=new Input_Node;
   Node[2]=new BP_Middle_Node( 0.4, 0.9 );      // Middle layer nodes
   Node[3]=new BP_Middle_Node( 0.4, 0.9 );      //   Learning rate 0.4, Momentum 0.9
   Node[4]=new BP_Middle_Node( 0.4, 0.9 );
   Node[5]=new BP_Output_Node( 0.4, 0.9 );      // Output layer node

   for (int i=0; i<9; i++)           // Create Links for Network
      Link[i]=new BP_Link();

   int curr=0;                     // Connect Network with links
   for (i=2; i<=4; i++)
      for (int j=0; j<=1; j++)
         Connect(Node[j],Node[i],Link[curr++]);

   for (int j=2; j<=4; j++)
       Connect(Node[j],Node[5],Link[curr++]);


// Train Backprop Network
   long iteration=0;
```

```
     int good=0;
     double tolerance=0.5;
     double total_error;

     while (good<4)     // Train until all patterns are correct
        {
        good=0;
        total_error=0.0;

        for (int i=0; i<4; i++)
           {
           Node[0]->Set_Value(data[i]->In(0));  // Set Input Node values
           Node[1]->Set_Value(data[i]->In(1));

           for (int j=2; j<=5; j++)            // Forward Pass
              Node[j]->Run();

           Node[5]->Set_Error(data[i]->Out(0));  // Set Error Values

           for (j=5; j>=2; j--)               // Backward Pass
               Node[j]->Learn();

           if (fabs(Node[5]->Get_Value()-data[i]->Out(0))<tolerance)
               good++;

           total_error+=fabs(Node[5]->Get_Error());
           }

                                          // Print status every 1000 iterations
        if (iteration%1000==0) cout << iteration << ".   " << good << "/4"
                   << "   Error: " << setprecision(15) << total_error << endl;
        iteration++;
        }

// Save Backprop
    ofstream outfile("XOR-BP.net");
    for (i=0; i<6; i++)           // Save Nodes
       Node[i]->Save(outfile);

    for (i=0; i<9; i++)           // Save Links
       Link[i]->Save(outfile);

    outfile.close();

// Destroy Network
    for ( i=0; i<6; i++)
        delete Node[i];
    for (i=0; i<9; i++)
       delete Link[i];

// Create Backprop
    Node[0]=new Input_Node;
    Node[1]=new Input_Node;
    Node[2]=new BP_Node;              // Only BP_Nodes are required
    Node[3]=new BP_Node;              // since there will be no learning
    Node[4]=new BP_Node;
    Node[5]=new BP_Node;

    for (i=0; i<9; i++)              // Create Links for Network
       Link[i]=new BP_Link();

    curr=0;                       // Connect Network
    for (i=2; i<=4; i++)
       for (int j=0; j<=1; j++)
           Connect(Node[j],Node[i],Link[curr++]);

    for (j=2; j<=4; j++)
         Connect(Node[j],Node[5],Link[curr++]);

// Load Backprop
    ifstream infile("XOR-BP.net");
    for (i=0; i<6; i++)
         Node[i]->Load(infile);

    for (i=0; i<9; i++)
       Link[i]->Load(infile);

    infile.close();

// Run Backprop Network
    for (i=0; i<4; i++)
        {
        Node[0]->Set_Value(data[i]->In(0)); // Set Input Node Values
        Node[1]->Set_Value(data[i]->In(1));

       for (int j=2; j<=5; j++)           // Forward Pass
          Node[j]->Run();

       double out=Node[5]->Get_Value();    // Get output layer's output

       cout << "Pattern: " << setw(3) << i << "  Input: ("
            << data[i]->In(0) << ","
            << data[i]->In(1)
            << ")  Backprop: (" << out
            << ")   Actual: (" << data[i]->Out(0) << ")" << endl;
       }

// Destroy Network
    for ( i=0; i<6; i++)
        delete Node[i];
    for (i=0; i<9; i++)
       delete Link[i];
    }
```

```
//******************************************************************************
 //  File "XOR-BP.net" contains the  data for the BP network Nodes and Links for solving
 //  XOR problem with Backpropagation neural   network
//******************************************************************************
1 1
1 0
        1
1  1
1  0
        2
3  6.02710922411746e-005      0.4       0.9
1  0.000137123827712989
    3
3  0.000111051528329315       0.4       0.9
1 -6.84061739162355e-005
      4
3 0.00011163761027813 0.4 0.9
1 -6.8695496595081e-005
   5
3 0.499999968422045 0.4 0.9
1 -0.124999992105511

   0 -4.89288599033693    0    2
   1 -4.82361066015608    1    2
   2 -11.309592798329    0    3
   3 2.20413288978632    1    3
   4 2.19493896226648    0    4
   5 -11.295106262584    1    4
   6 -18.202039667836    2    5
   7 4.9284346983835    3    5
   8 4.92329928025148    4    5


//********************************************************************************
// File  XOR-BP.out contains the training and running results output for XOR problem
//********************************************************************************
0.   2/4   Error: 0.463308447694839
1000.   3/4   Error: 0.170356930544738
2000.   3/4   Error: 0.138043219435865
3000.   3/4   Error: 0.132362087638176
4000.   3/4   Error: 0.130080369249295
5000.   3/4   Error: 0.128861501475946
6000.   3/4   Error: 0.128106497769808
7000.   3/4   Error: 0.127594143905471
8000.   3/4   Error: 0.127224204503693
9000.   3/4   Error: 0.126944806644006
10000.   3/4   Error: 0.126726461653785
11000.   3/4   Error: 0.126551187089369
12000.   3/4   Error: 0.126407402081706
13000.   3/4   Error: 0.12628731482148
14000.   3/4   Error: 0.126185490654964
15000.   3/4   Error: 0.126098022288945
16000.   3/4   Error: 0.126022026029402
17000.   3/4   Error: 0.125955323486466
18000.   3/4   Error: 0.125896233126942
19000.   3/4   Error: 0.125843428941118
20000.   3/4   Error: 0.125795840841505
21000.   3/4   Error: 0.125752580711317
22000.   3/4   Error: 0.12571288268661б
23000.   3/4   Error: 0.12567604753797
24000.   3/4   Error: 0.125641378331542
25000.   3/4   Error: 0.125608084024552
Pattern:   0  Input: (0,0)   Backprop: (0.0151400955259262)    Actual: (0)
Pattern:   1  Input: (0,1)   Backprop: (0.986527917178414)    Actual: (1)
Pattern:   2  Input: (1,0)   Backprop: (0.986540531239183)    Actual: (1)
Pattern:   3  Input: (1,1)   Backprop: (0.499999930865445)    Actual: (0)
```

```
//*************************************************************************
// File: bit_vec.h    used for training recurrent network using GA
//
//*************************************************************************

#ifndef bit_vec_h
#define bit_vec_h

#include <iostream.h>

const BITS_PER_INT = 8 * sizeof(unsigned int);


class bit_vector {
 public:
   bit_vector(int num_bits = 1);
  ~bit_vector();
   unsigned int operator[](int bit_index);
   int number_of_on_bits();
   void operator+=(int bit_to_turn_on);
   void operator-=(int bit_to_turn_off);
   int operator==(bit_vector &other);
   void set_to_zero();
   void set_to_one();
   int size()
   {
    return number_of_bits;
   }

 protected:
   unsigned int * data;
   int number_of_bits;
   int number_of_ints;

};
#endif
```

```
//**************************************************************************
// File: bit_vect.cpp
//**************************************************************************

#include "bit_vect.h"

bit_vector::bit_vector(int num_bits)
{
   number_of_ints = (num_bits + (BITS_PER_INT - 1)) / BITS_PER_INT;
   number_of_bits = num_bits;
   data = new unsigned int[number_of_ints];
   set_to_zero();
}


bit_vector::~bit_vector()
{
   delete [] data;
}


unsigned int bit_vector::operator[](int bit_index)
{
   return ((data[bit_index / BITS_PER_INT] &
          (1 << (bit_index % BITS_PER_INT)))
      != 0);
}


int bit_vector::number_of_on_bits()
{
   int ret_val = 0;
   for (int i=0; i<number_of_bits; i++)
      if (data[i / BITS_PER_INT] & (1 << (i % BITS_PER_INT)))
        ret_val++;
   return ret_val;
}



extern "C" { void exit(int); };

void bit_vector::operator+=(int bit_to_turn_on)
{
#ifndef FAST
   if (bit_to_turn_on < 0 || bit_to_turn_on >= number_of_bits)
   {
      cerr << "error: bit_to_turn_on=" << bit_to_turn_on << "\n";
      exit(1);
   }
#endif
   data[bit_to_turn_on / BITS_PER_INT] |=     (1 << (bit_to_turn_on % BITS_PER_INT));
}

void bit_vector::operator-=(int bit_to_turn_off)
{
```

```
#ifndef FAST
   if (bit_to_turn_off < 0 || bit_to_turn_off >= number_of_bits)
   {
      cerr << "error: bit_to_turn_off=" << bit_to_turn_off << "\n";
      exit(1);
   }
#endif
   data[bit_to_turn_off / BITS_PER_INT] &=
      (~(1 << (bit_to_turn_off % BITS_PER_INT)));
}


int bit_vector::operator==(bit_vector &other)
{
#ifndef FAST
   int size_other = other.size();
   if (number_of_bits != size_other)
   {
      cerr << "bit_vector::operator== size mismatch\n";
      exit(1);
   }
#endif
   for (int i=0; i<number_of_bits; i++)
   {
      if ((*this)[i] != other[i])
      {
         return 0;
      }
   }
   return 1;
}


void bit_vector::set_to_zero()
{
   for (int i=0; i<number_of_ints; i++)
      data[i] = 0;
}

void bit_vector::set_to_one()
{
   for (int i=0; i<number_of_ints; i++)
      data[i] = 1;
}
```

```
//**************************************************************************
// File: randseq.h
//**************************************************************************

#ifndef randseq_h
#define randseq_h

#define randomize() srand((unsigned)time(NULL))
#define random(num) (rand()%(num))

class RandomSequence
{
  public:
    RandomSequence(float randomSeed = 10.0);
    int    rndINT(int low, int high);
    double rndFP(double low, double high);

  private:
    void   next_random_sequence();
    double rand_util();
    float  randomseed;
    int    random_counter;
    int    rndcalcflag;
    double oldrand[55];
};

#endif



//**************************************************************************
// File: randseq.cpp   utility class to generate random integers and floating point numbers
//**************************************************************************
#define USE_LIBRARY 1

#include "randseq.h"
#include <math.h>

#include<time.h>
#define random(num)   (rand()%(num))
#define randomize()   srand((unsigned)time(NULL))



#ifdef USE_LIBRARY
#include <stdlib.h>
#endif

static float initialSeed = 0.7123152;

RandomSequence::RandomSequence(float randomSeed)
{
  if ( randomSeed < 0.0) randomSeed = - randomSeed;
  if ( randomSeed > 0.999)  randomSeed *= 0.1;
  if ( randomSeed > 0.999)  randomSeed *= 0.1;
  if (randomSeed >  0.999 || randomSeed < 0.01)
  {
    initialSeed *= 0.9842134;
    randomSeed = initialSeed;
    if (initialSeed < 0.1)  initialSeed *= 0.99991;
  }

#ifdef USE_LIBRARY
  randomize();
  float rr = rand();
  rr = rr / 70000.0;
  initialSeed = rr;
#endif

  int ran_index, ii;
  double new_random, prev_random;

  oldrand[54] = randomSeed;
  for (int k=0; k<54; k++)  oldrand[k] = 0.1 + (float)k * 0.016;
  new_random = 0.000000001;
  prev_random = randomSeed;
  for(ran_index = 1 ; ran_index <= 540; ran_index++)
  {
   ii = (21*ran_index)%54;
   oldrand[ii] = new_random;
   new_random = prev_random-new_random;
   if(new_random<0.0) new_random = new_random + 1.0;
   prev_random = oldrand[ii];
  }

  next_random_sequence();
  next_random_sequence();
  next_random_sequence();
  next_random_sequence();

  random_counter = 0;

}
void RandomSequence::next_random_sequence()
/* Create next batch of 55 random numbers */
{
   int ran_index;
   double new_random;

   for(ran_index = 0; ran_index < 24; ran_index++)
   {
      new_random = oldrand[ran_index] - oldrand[ran_index+31];
      if(new_random < 0.0) new_random = new_random + 1.0;
      oldrand[ran_index] = new_random;
   }
   for(ran_index = 24; ran_index < 55; ran_index++)
   {
      new_random = oldrand [ran_index] - oldrand [ran_index-24];
      if(new_random < 0.0) new_random = new_random + 1.0;
      oldrand[ran_index] = new_random;
   }
```

```
}

int RandomSequence::rndINT(int low, int high)
{
   int ret_val;
   if(low >= high) {
      ret_val = low;
   } else {
      ret_val = (int)((rand_util() * (high - low + 1)) + low);
      if(ret_val > high) ret_val = high;
   }
   return(ret_val);
}

double RandomSequence::rndFP(double low, double high)
{
   return (rand_util() * (high - low)) + low;
}

double RandomSequence::rand_util()
{
#ifndef USE_LIBRARY
   random_counter++;
   if(random_counter >= 55)
   {
      random_counter = 1;
      next_random_sequence();
   }
   return  oldrand[random_counter];
#else
   return ((float)(random(10000))) * 0.0001;
#endif
}


//************************************************************
// File: genetic.cpp    program for solving XOR problem by using
// genetic algorithm to train recurrent neural network
//************************************************************
#include "bit_vect.h"
#include "randseq.h"
#include <math.h>
#include <iostream.h>

const float MIN_VAL = -10.0;
const float MAX_VAL =  10.0;

const int NUM_INPUTS  = 2;
const int NUM_HIDDEN  = 6;
const int NUM_OUTPUTS = 1;
const int POPULATION = 500;
```

```
//#define USE_I_TO_O
#define USE_H_TO_H
#define USE_O_TO_H

float sigmoid(float x)
{
  return (1.0 / (1.0 + exp(-x))) - 0.5;
}

//-----------------------------------------------------------
//      genetic_recurrent_network  class
//-----------------------------------------------------------

class genetic_recurrent_network
{
 public:
   genetic_recurrent_network(int bits_per_value);
  ~genetic_recurrent_network();
   float fitness(); // for weights copied to float weight arrays
   float train(int number_of_generations); // return best fitness
   void copy_bit_weights_to_floats(int population_index);
   float solve();

 private:
   int bit_size;
   int int_range;
   int num_bits_per_chromosome;
   bit_vector **population;
   // float storage for weights (a chromosome with a specified
   // population index can be copied into these arrays to make
   // writing the fitness() function easier):
   float i_to_h[NUM_INPUTS][NUM_HIDDEN];
#ifdef USE_I_TO_O
   float i_to_o[NUM_INPUTS][NUM_OUTPUTS];
#endif
#ifdef USE_H_TO_H
   float h_to_h[NUM_HIDDEN][NUM_HIDDEN];
   float hidden_sums[NUM_HIDDEN];
#endif
#ifdef USE_O_TO_H
   float o_to_h[NUM_OUTPUTS][NUM_HIDDEN];
#endif
   float h_to_o[NUM_HIDDEN][NUM_OUTPUTS];

   float inputs[NUM_INPUTS];
   float hidden[NUM_HIDDEN];
   float outputs[NUM_OUTPUTS];

   void propagate();

   float *fitnesses;

   void initialize_population();
   void sort_by_fitness();

   RandomSequence ran_seq;
```

```
};


static int powers_of_two[] = {1,2,4,8,16,32,64,128,256,512,1024,2048,
                    4096, 8192, 16384, 32768};

genetic_recurrent_network::genetic_recurrent_network(int bits_per_value)
{
   bit_size = bits_per_value;
   int_range = powers_of_two[bit_size];
   population = new bit_vector * [POPULATION];
   num_bits_per_chromosome =
     bit_size *(NUM_INPUTS*NUM_HIDDEN
#ifdef USE_H_TO_H
          + NUM_HIDDEN*NUM_HIDDEN
#endif
#ifdef USE_I_TO_O
          + NUM_INPUTS*NUM_OUTPUTS
#endif
#ifdef USE_O_TO_H
          + NUM_OUTPUTS*NUM_HIDDEN
#endif
          + NUM_HIDDEN*NUM_OUTPUTS);
   for (int i=0; i<POPULATION; i++)
      population[i] = new bit_vector(num_bits_per_chromosome);
   fitnesses = new float[POPULATION];

   initialize_population();
   for (i=0; i<POPULATION; i++)
   {
     copy_bit_weights_to_floats(i);
     fitnesses[i] = fitness();
   }
   sort_by_fitness();
}


genetic_recurrent_network::~genetic_recurrent_network()
{
   delete [] population;
   delete [] fitnesses;
}
//-------------------------------------------------------------------
//      This function converts a specified chromosome (bit vector)
//      in the population to the floating point weights.
//-------------------------------------------------------------------
void genetic_recurrent_network::copy_bit_weights_to_floats(int population_index)
{
   bit_vector *b = population[population_index];
   int start_bit = 0;
   int i, j, k, val;
   float f_val;

   // input to hidden weights:
   for (i=0; i<NUM_INPUTS; i++)
   {
        for (j=0; j<NUM_HIDDEN; j++)
        {
            val = 0;
            for (k=0; k<bit_size; k++)
                if ((*b)[k+start_bit])
                    val += powers_of_two[bit_size - k - 1];
            f_val = val;
            f_val = (f_val / ((float)int_range))
                  * (MAX_VAL - MIN_VAL) + MIN_VAL;
            start_bit += bit_size;
            i_to_h[i][j] = f_val;
        }
    }

#ifdef USE_I_TO_O

    // input to output weights:
    for (i=0; i<NUM_INPUTS; i++)
    {
        for (j=0; j<NUM_OUTPUTS; j++)
        {
            val = 0;
            for (k=0; k<bit_size; k++)
                if ((*b)[k+start_bit])
                    val += powers_of_two[bit_size - k - 1];
            f_val = val;
            f_val = (f_val / ((float)int_range))
                  * (MAX_VAL - MIN_VAL) + MIN_VAL;
            start_bit += bit_size;
            i_to_o[i][j] = f_val;
        }
    }

#endif

#ifdef USE_H_TO_H

    // hidden to hidden weights:
    for (i=0; i<NUM_HIDDEN; i++)
    {
        for (j=0; j<NUM_HIDDEN; j++)
        {
            val = 0;
            for (k=0; k<bit_size; k++)
                if ((*b)[k+start_bit])
                    val += powers_of_two[bit_size - k - 1];
            f_val = val;
            f_val = (f_val / ((float)int_range))
                  * (MAX_VAL - MIN_VAL) + MIN_VAL;
            start_bit += bit_size;
            h_to_h[i][j] = f_val;
        }
    }

#endif
```

```
#ifdef USE_O_TO_H

    // output to hidden weights:
    for (i=0; i<NUM_OUTPUTS; i++)
    {
       for (j=0; j<NUM_HIDDEN; j++)
       {
          val = 0;
          for (k=0; k<bit_size; k++)
             if ((*b)[k+start_bit])
                val += powers_of_two[bit_size - k - 1];
          f_val = val;
          f_val = (f_val / ((float)int_range))
               * (MAX_VAL - MIN_VAL) + MIN_VAL;
          start_bit += bit_size;
          o_to_h[i][j] = f_val;
       }
    }

#endif
    // hidden to output weights:
    for (i=0; i<NUM_HIDDEN; i++)
    {
       for (j=0; j<NUM_OUTPUTS; j++)
       {
          val = 0;
          for (k=0; k<bit_size; k++)
             if ((*b)[k+start_bit])
                val += powers_of_two[bit_size - k - 1];
          f_val = val;
          f_val = (f_val / ((float)int_range))
               * (MAX_VAL - MIN_VAL) + MIN_VAL;
          start_bit += bit_size;
          h_to_o[i][j] = f_val;
       }
    }
}

////----------------------------------------------------------
//      Member function initialize_population
////----------------------------------------------------------
void genetic_recurrent_network::initialize_population()
{
    for (int i=0; i<POPULATION; i++)
    {
       for (int j=0; j<num_bits_per_chromosome; j++)
       {
          int index = ran_seq.rndINT(0, num_bits_per_chromosome - 1);
          (*population[i]) += index;
       }
    }
}
//----------------------------------------------------------------
//      Member function sort_by_fitness
//----------------------------------------------------------------
void genetic_recurrent_network::sort_by_fitness()
```

```
{
   float x;
   bit_vector *b;
   for (int i=0; i<POPULATION; i++)
   {
      for (int j=(POPULATION - 2); j>=i; j--)
      {
         if (fitnesses[j] > fitnesses[j+1])
         {
            x = fitnesses[j];
            b = population[j];
            fitnesses[j] = fitnesses[j+1];
            population[j] = population[j+1];
            fitnesses[j+1] = x;
            population[j+1] = b;
         }
      }
   }
#if 0
   cout << "fitnesses: ";
   for (i=0; i<POPULATION; i++) cout << fitnesses[i] << " ";
   cout << "\n";
#endif
}

////-------------------------------------------------------
//       Member function solve
////-------------------------------------------------------
float genetic_recurrent_network::solve()
{
   int i = 0, j;
   for (j=3*POPULATION/4 + 1; j<POPULATION; j++)
   {  // copy chromosome # i to # j:
      for (int k=0; k<num_bits_per_chromosome; k++)
      {
         if ((*population[i])[k])
            (*population[j]) += k;
         else
            (*population[j]) -= k;
      }
      i++;
      if (i >= (POPULATION/4 + 1))  i = 0;
   }

   // crossovers:
   for (j=0; j<POPULATION/5; j++)
   {
      int chrom_1 = ran_seq.rndINT(POPULATION/10+1, POPULATION - 1);
      int chrom_2 = ran_seq.rndINT(POPULATION/10+1, POPULATION - 1);
      int crossover_location =
         ran_seq.rndINT(1,num_bits_per_chromosome/bit_size);
      crossover_location *= bit_size;
      for (int k=0; k<crossover_location; k++)
      {
         int save = (*population[chrom_2])[k];
         if ((*population[chrom_1])[k])
```

```
            (*population[chrom_2]) += k;
        else
            (*population[chrom_2]) -= k;
        if (save)
            (*population[chrom_1]) += k;
        else
            (*population[chrom_1]) -= k;
    }
    i++;
}

// mutations:

for (j=5; j<POPULATION; j++)
{
    if (ran_seq.rndFP(0.0, 100.0) < 10.0)
    {
        int location = ran_seq.rndINT(0, num_bits_per_chromosome - 1);
        if ((*population[j])[location])
            (*population[j]) -= location;
        else
            (*population[j]) += location;
    }
}
for (j=7*POPULATION/8; j<POPULATION; j++)
{
    for (int m=0; m<num_bits_per_chromosome/4+2; m++)
    {
        if (ran_seq.rndFP(0.0, 100.0) < 75.0)
        {
            int location = ran_seq.rndINT(0, num_bits_per_chromosome - 1);
            if ((*population[j])[location])
                (*population[j]) -= location;
            else
                (*population[j]) += location;
        }
    }
}

for (i=0; i<POPULATION-1; i++)
{
    for (j=i+1; j<POPULATION; j++)
    {
        if ((*population[i]) == ((*population[j])))
        {
            int mutation_location =
              ran_seq.rndINT(0, num_bits_per_chromosome - 1);
            if ((*population[j])[mutation_location])
                (*population[j]) -= mutation_location;
            else
                (*population[j]) += mutation_location;
        }
    }
}

// Evaluate the fitness values of all chromosomes in the population:
    for (i=0; i<POPULATION; i++)
    {
      copy_bit_weights_to_floats(i);
      fitnesses[i] = fitness();
    }
    sort_by_fitness();

    return fitnesses[0];
  }

////-------------------------------------------------------
//      Member function propagate
//
//      This function does not directly use any chromosomes
//      in the current population.  It is used to propagate the
//      current input neuron activation values through the network
//      using the current weight set (as defined by member function
//      copy_bit_weights_to_floats for a specified chromosome).
////-------------------------------------------------------
void genetic_recurrent_network::propagate()
{
    for (int h=0; h<NUM_HIDDEN; h++)
    {
      hidden[h] = 0.0;
#ifdef USE_H_TO_H
      hidden_sums[h] = 0.0;
#endif
    }

#ifdef USE_H_TO_H
    for (int cycle=0; cycle<3; cycle++)
#endif
    {
      for (h=0; h<NUM_HIDDEN; h++)
      {
        for (int i=0; i<NUM_INPUTS; i++)
        {
          hidden[h] += inputs[i] * i_to_h[i][h];
        }
#ifdef USE_H_TO_H
        for (int h2=0; h2<NUM_HIDDEN; h2++)
        {
          hidden_sums[h] += hidden[h2] * h_to_h[h2][h];
        }
#endif
#ifdef USE_O_TO_H
        for (int o2=0; o2<NUM_OUTPUTS; o2++)
        {
          hidden_sums[h] += outputs[o2] * o_to_h[o2][h];
        }
#endif

#ifndef USE_H_TO_H
        hidden[h] = sigmoid(hidden[h]);
#else
        hidden[h] = sigmoid(hidden_sums[h]);
#endif
      }

      for (int o=0; o<NUM_OUTPUTS; o++)
         outputs[o] = 0.0;
      for (o=0; o<NUM_OUTPUTS; o++)
      {
         for (h=0; h<NUM_HIDDEN; h++)
         {
            outputs[o] += hidden[h] * h_to_o[h][o];
         }

#ifdef USE_I_TO_O
         for (int i=0; i<NUM_INPUTS; i++)
         {
            outputs[o] += inputs[i] * i_to_o[i][o];
         }
#endif
      }
      for (o=0; o<NUM_OUTPUTS; o++)
      {
         outputs[o] = sigmoid(outputs[o]);
      }
   }
}

////----------------------------------------------------------
//       Member function train
//
//       This function creates a specific number of new
//       generations in the population and uses member
//       function solve to evaluate each chromosome in each
//       population.
////----------------------------------------------------------
float genetic_recurrent_network::train(int num_generations)
{
   float best_error;
   for (int i=0; i<num_generations; i++)
   {
      best_error = solve();
      cout << "error =" << best_error << "\n";
   }
   return best_error;
}


// APPLICATION SPECIFIC FITNESS FUNCTION:

inline float val_s[illegible]oat x)
{
   return x * x;
}

int debug = 0;

//          XOR test problem setup:
```

```
float genetic_recurrent_network::fitness()
{
  // test with a fitness for xor function:
  static float in_1[] = {-0.21, -0.2,  0.2,  0.2};
  static float in_2[] = {-0.2,  0.22, -0.2,  0.2};
  static float out[]  = {-0.201, 0.21,  0.19, -0.192};

  float error = 0.0;
  for (int i=0; i<4; i++)
  {
    inputs[0] = in_1[i];
    inputs[1] = in_2[i];
    propagate();
    error += val_sq(outputs[0] - out[i]);
    if (debug)
    {
      cout << "inputs: " << inputs[0] << ", " << inputs[1]
          << "  output: " << outputs[0] << "\n";
    }
  }
  return sqrt(error);
}

//------------------------------------------------------------
//   main program
//------------------------------------------------------------
void main()
{
        int iteration;
  genetic_recurrent_network GeneticNet(12);
  cout << "First round\n";
  for ( iteration=0; iteration<10000; iteration++)
  {
    debug = 0;
    GeneticNet.train(5);
    debug = 1;
    GeneticNet.copy_bit_weights_to_floats(0);
    cout << "\n\ngeneration " << 5*(iteration + 1) << "\n";
    GeneticNet.fitness();
  }
}
```

```
//*************************************************************************
// File genetic.out        patial  outputs of the results of solving XOR problem by using
// genetic algorithm to recurrent neural network
//*************************************************************************
first round
error =0.37144
error =0.37144
error =0.37144
error =0.37144
error =0.37144


after generation 5
inputs: -0.21, -0.2  output: -0.0284544
inputs: -0.2, 0.22  output: 0.0832061
inputs: 0.2, -0.2  output: -0.00502013
inputs: 0.2, 0.2  output: 0.0405635
error =0.37144
error =0.37144
error =0.370686
error =0.370686
error =0.370686


generation 10
inputs: -0.21, -0.2  output: -0.0175243
inputs: -0.2, 0.22  output: 0.0845595
inputs: 0.2, -0.2  output: -0.00671278
inputs: 0.2, 0.2  output: 0.0300667
error =0.370686
error =0.370464
error =0.370382
error =0.368328
error =0.366677

-    -    -
generation 215
inputs: -0.21, -0.2  output: -0.00651032
inputs: -0.2, 0.22  output: 0.0522734
inputs: 0.2, -0.2  output: 0.0356941

inputs: 0.2, 0.2  output: 0.013072
error =0.358565
error =0.358565
error =0.358565
error =0.358549
error =0.358549


generation 220
inputs: -0.21, -0.2  output: 0.000575993
inputs: -0.2, 0.22  output: 0.0521467
inputs: 0.2, -0.2  output: 0.0356052
inputs: 0.2, 0.2  output: 0.0059123
error =0.358528
error =0.358528
error =0.358528
error =0.358528
error =0.358528

-    -    -
generation 1000
inputs: -0.21, -0.2  output: -0.22893
inputs: -0.2, 0.22  output: 0.052968
inputs: 0.2, -0.2  output: 0.0325668
inputs: 0.2, 0.2  output: -0.0360713
error =0.350547
error =0.350536
```