AN EXTENDED CASCADE CORRELATION

NEURAL NETWORK

By

YULEI BAI

Bachelor of Science
Northwest University
Xi'an, P. R. China
1985

Master of Science
Research Institute of
Petroleum Exploration and Development
Beijing, P. R. China
1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirement for
the Degree of
MASTER OF SCIENCE
May, 2002

AN EXTENDED CASCADE CORRELATION

NEURAL NETWORK

Thesis Approved:

J. Chandler

Thesis Adviser

R. E. Hedrick

Rebar Chen

Timothy J. Pettibone

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my thesis advisor, Dr. John P. Chandler, for his constructive guidance in choosing the topic of this thesis, constant inspiration, and valuable time throughout my thesis work. My sincere appreciation also extends to other committee members, Dr. George E. Hedrick and Dr. Debao Chen, for their helpful advisement, suggestions and valuable time.

In addition, I would like to give my special appreciation to my wife, Qiaoling Li, for her strong encouragement, support and understanding throughout my study in Oklahoma State University. Thanks also go to my parents for their love and encouragement.

Finally, I would like to thank the Department of Computer Science for supporting me during the time of my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1. INTRODUCTION

1.1 Multi-Layer Feedforward Neural Networks

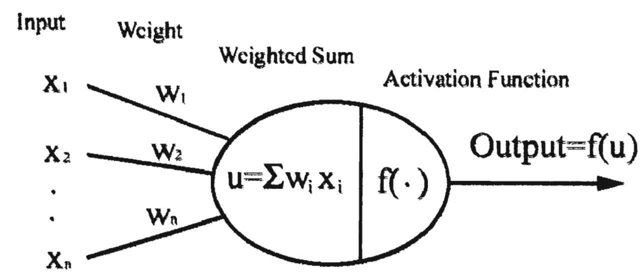
Multi-layer feed-forward networks, usually called Multi-Layer Perceptrons or MLPs, are the most common form of artificial neural networks (ANNs). Typically a MLP consists of several hidden layers of fully-connected artificial neurons (Figure 1.1). Each neuron (or node) has a bias input and an activation function associated with it.

The connections between neurons are represented by weights. The output of each neuron is the output of the activation function which takes as its input the weighted sum of all inputs to the neuron plus a weighted bias. The outputs of the neurons in one layer are all linked to each of the neurons in the next layer, except for the output layer of the network.

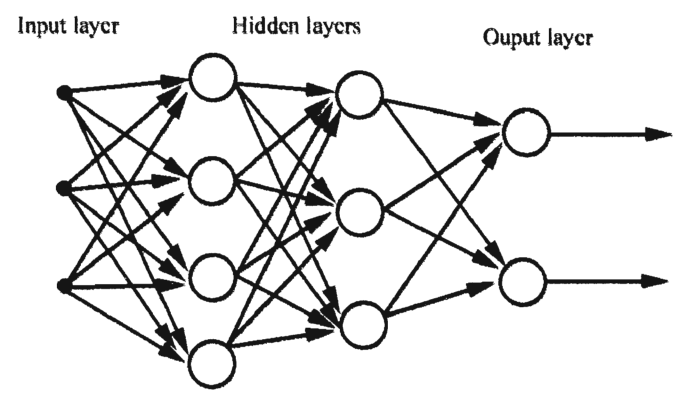
A MLP is capable of learning from examples. When input signals are fed in, the computation of the network is carried out on a layer-by-layer basis until the outputs of the network have been produced (forward pass). The result from this forward pass is compared with the desired output and error estimates are computed for the output nodes. This process repeats until the network goes through all the training examples. After this stage, the network is used to compute output for other unseen example inputs based on its generalization of the training examples.

ANNs are especially useful to solve problems whose underlying rules are unknown or difficult to be explicitly represented. Another important feature of ANNs is that a network with the same architecture can be used to solve different kinds of problems.

MLPs are very popular in, but not limited to, the domains of pattern classification and function approximation.



(a)



(b)

Figure 1.1: An artificial neuron (a) and a multi-layer feed-forward neural network (b)

1.2 Optimization of Network Architecture

Apart from the problems of choosing the training algorithm, the first major decision is to determine the optimal architecture for the given problem before training begins. The architectural factors include the numbers of hidden layers and the numbers of hidden nodes in these layers.

It has been known that a two-layer MLP network (one hidden layer) with enough hidden units can approximate any continuous function to any degree of accuracy [4] [7][9]. However, there is no theoretic solution for determining how many hidden units are sufficient for a given problem. If the number of nodes is too small the network may fail to generalize well (underfitting) between different inputs. On the other hand if the number of nodes and layers is too large then the network may be very slow in training and too closely approximate the training data (overfitting, i.e. exactly fit the noise).

The bad generalization is partly due to the insufficient representational capacity of the network (because of the finite size of the network used) and partly due to insufficient information about the target function because of a finite number of examples [16]. In the case of underfitting, the network cannot learn the correct representation for the given problem. In overfitting the network tends to memorize the details of all examples seen and is most unlikely to perform well when novel or noisy examples presented. Both underfitting and overfitting show the mismatch of the complexity between the problem being solved and the network used [11] [22].

It is reasonable to understand that the performance of a MLP network with fixed architecture may vary from problem to problem. In other words, an optimal architecture found for one problem may not be optimal for another problem. It is desirable for a network to have the ability to adjust its architecture towards an optimal structure during learning.

There are various approaches used in the area of network architecture selection [18]. Ad hoc or trial-and-error methods are based on past experience and knowledge of the problem. Several networks of different architectures are experimented with and the results are examined. Only the network that gives the best results is selected. This approach is useful when a-priori knowledge of the problem is available. But it is not so useful in most cases where neural networks are used. On the other hand, the architecture of the net cannot be modified during training. An alternative is to find the optimal structure of a neural network by using Genetic Algorithms [23] or other methods [3]. These methods seem computationally intensive before the optimal architecture is found. Another popular approach is to use Dynamic Learning Algorithms for optimization of the network architecture.

Dynamic Learning Algorithms are characterized by growing (or constructive, additive) and/or pruning (or destructive, subtractive) processes that automatically modify network topology during learning from a set of examples.

In constructive algorithms [2] [10], an initial network is developed with a small number of hidden nodes, and more nodes can be added during training in order to produce more accurate results (growing the network to minimize the error). The final topology and size of the network are dynamically determined by the algorithm and are a function of the set of examples and of the learning parameters. Constructive algorithms have the inherent advantage of rapid training in addition to finding both the architecture and weights. The potential disadvantage is that they may create over-complex networks. Although only a few constructive algorithms have actually been used in real applications, the Cascade Correlation network (CasCor) proposed by Fahlman [6] and its variants have been successfully used in many applications and is implemented in most large neural network simulator programs [20].

Pruning algorithms take the opposite strategy to growing algorithms. They start with a more complicated network and eliminate weights and nodes based on the analysis of relevance between weights [8] [12] [15] [17] [21] in order to reduce the complexity of the network.

The constructive approaches have several advantages over pruning algorithms [10]:

1) They are straightforward to specify an initial network with small size. For pruning, one does not know in practice how big the initial network should be.
2) Constructive algorithms always search for small network solutions first, and thus tend to be computationally economical and find a smaller network than a pruning algorithm in which the majority of training time is spent on larger networks than necessary.
3) The pruning process may introduce larger errors, especially when many are to be pruned. Smaller weights may have important impact on generalization [1].

In this paper, constructive algorithms are used to deal with the modification of the network architecture during training.

1.3 The Purpose of the Paper

The aim of this paper is to propose an extended Cascade-Correlation network (CasCor) that can be trained by constructive algorithms. The Cascade-Correlation network builds the net by adding nodes in one dimension. The proposed network architecture (XCAS) is based on the CasCor network but allows addition of nodes in two dimensions.

The goal of the new XCAS network is to reduce number of weights needed to resolve the given problems compared with the CasCor network. The network topology and weights will be automatically determined during training. The CasCor network and the proposed network, as well as the learning algorithm, will be introduced and described in Chapter 2. Simulation results on regression problems, and comparisons between CasCor and XCAS networks will be presented in Chapter 3.

Chapter 2. AN EXTENDED CASCOR NETWORK

In this chapter, we will describe features of the CasCor network first, then propose an extended CasCor network architecture (XCAS), finally, introduce the training algorithm used in both the CasCor and XCAS networks.

2.1 The Cascade-Correlation Architecture (CasCor)

The Cascade-Correlation network [6] is characterized by the *cascade architecture* in which hidden nodes are added to the net one at a time and each node added becomes a one-unit layer of the net (Figure 2.1). Another feature is *weight-freezing*: once a new hidden node has been added to the net, its incoming connection weights are frozen and do not change in later training. The training algorithm that creates and install the new hidden nodes will be described in later section.

Every output unit receives connections from a bias unit, all original inputs and all hidden nodes with corresponding adjustable weights. The bias unit provides a constant value of +1. Every hidden unit receives connections from the bias unit, all original inputs and all previous existing hidden nodes (Figure 2.1). Output units may employ linear or non-linear activation functions according to the problems.

As illustrated in Figure 2.2, CasCor begins with no hidden nodes. After the output connections have been trained, new hidden nodes can be added to the network one-by-one. The hidden node's input weights are frozen at the time the node is added to the net; only the output connections are trained repeatedly. The cycle of adding a node repeats until certain stop criteria are met. Thus the network topology and weights are determined automatically during the training.

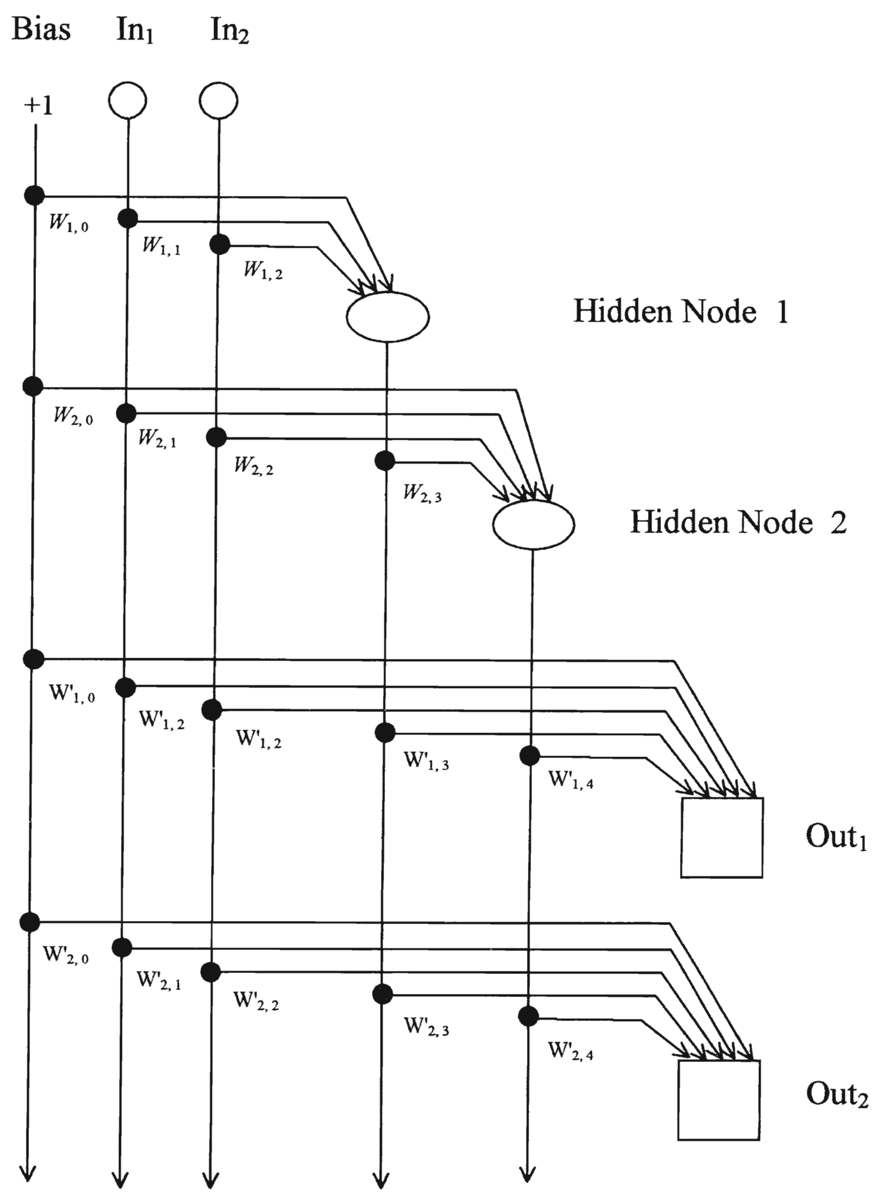


Figure 2.1: Diagram showing all connections in a CasCor network with two input units, two output units and two hidden units. For clarity of showing addition of hidden nodes (see Figure 2.2), several connections were lumped into one in the diagram drawn by Falhman (1990).

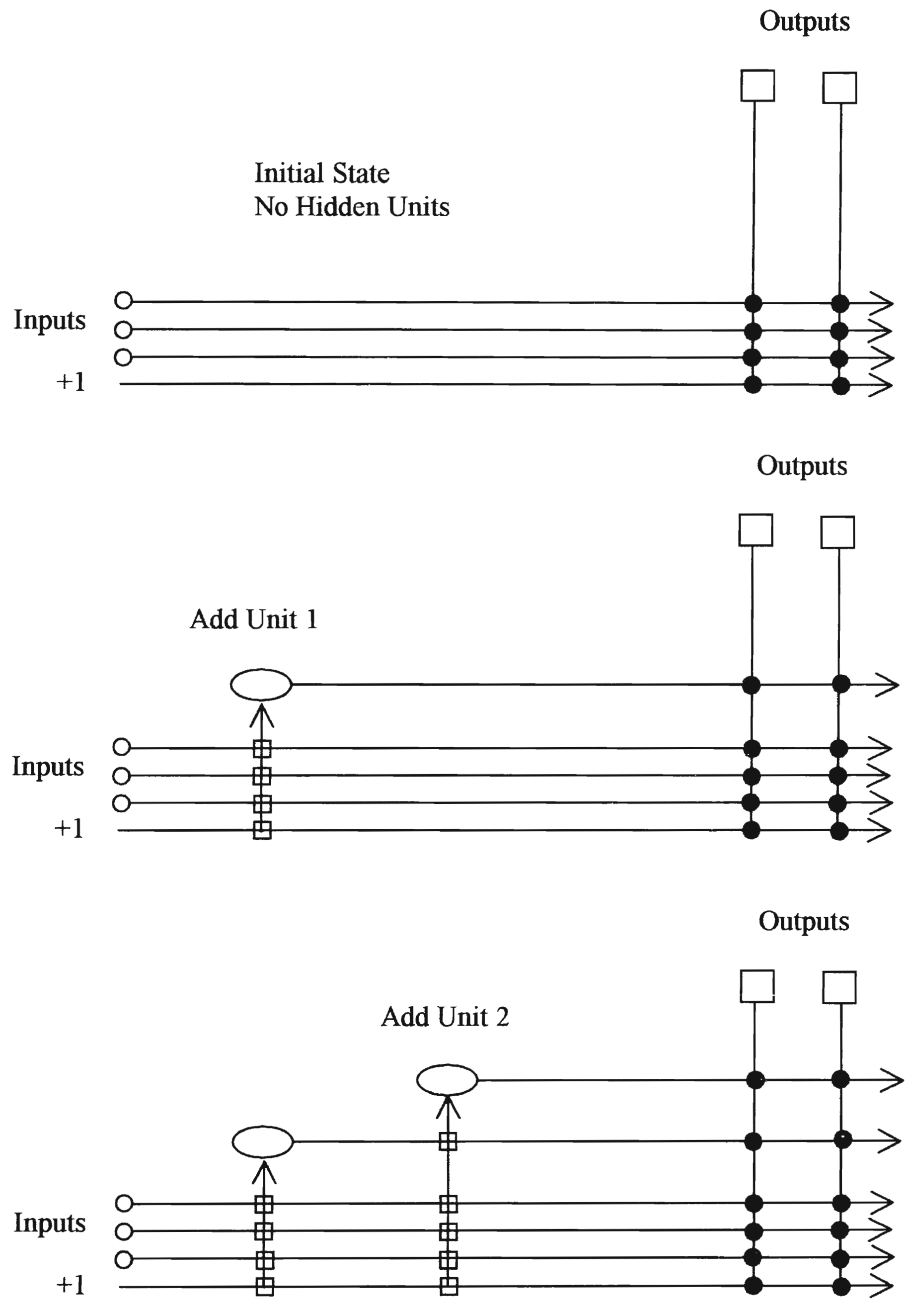


Figure 2.2: The CasCor network architecture. The vertical lines sum all incoming activation. Boxed connections are frozen, output connections (filled circles) are trained repeatedly (after Fahlman, 1990).

The CasCor network has two advantages over classic MLPs:

1) No need to guess about network topology in advance. Network size (number of layers and number of nodes in each layer) is automatically determined in the course of training.

2) No need to back-propagate error signals through the connections of the network. This means faster training.

However, the CasCor may produce very deep networks and lead to high fan-in to the hidden nodes. The total number of weights N_w in the resultant network will be large in this case. Weights (free or independent parameters) include all adjustable parameters that are associated with connections.

Let N be the number of hidden nodes in the net. p and q are the numbers of original inputs and outputs, which are constant for the given problem. The i-th hidden node has $(i-1)$ connections received from pre-existing hidden nodes and $(1+p)$ connections from original inputs and the bias node. Each output node has $(N+p+1)$ connections, so we have:

$$N_w = \sum_{i=1}^{N}(i+p) + q(N+p+1) = \frac{1}{2}N^2 + (p+q+\frac{1}{2})N + (1+p)q$$

This indicates the total number of weights is $\Theta(N^2)$, where N is the number of hidden nodes ultimately needed to solve the problem. It is obvious that both the depth of the net (or propagation delay) and the maximum fan-in of the hidden nodes are $\Theta(N)$.

In order to minimize the network depth and the fan-in of the hidden nodes, one approach proposed by [19] is to generate a strictly layered structure in which each layer has the same fixed number of connections (Figure 2.3). However, their test results showed the number of weights needed is close to or larger than that of CasCor network for the same problem. Another problem with their modified architecture is that the number of nodes in each layer must be set before training and can only be evaluated via heuristics and trial-and-error on the problem at hand. This problem should not appear in CasCor network family.

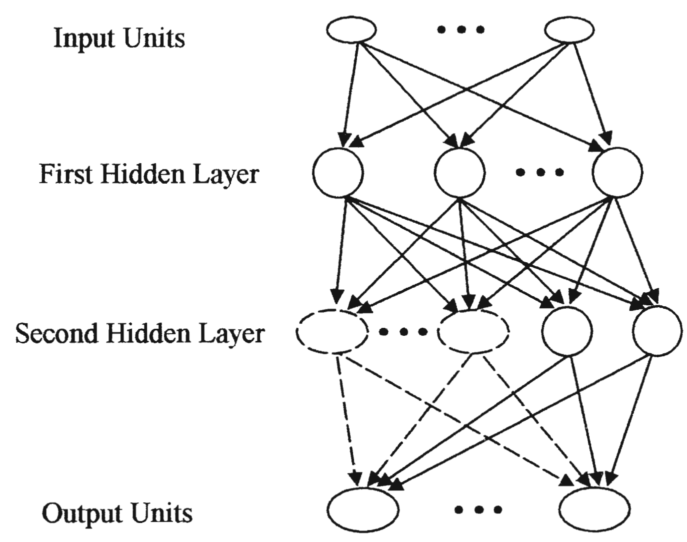


Figure 2.3: Strictly layered cascaded architecture. Each hidden layer has the fixed number of nodes. Old output units (dotted ellipses) are collapsed into the next layer (after [19]).

We take a different approach to achieve the same purpose. Our network architecture described in next section is designed to reduce the total number of weights but also reduce the network depth and fan-in of the hidden nodes.

2.2 Extension of CasCor Network Architecture

An important feature of the CasCor network is the way it adds the new node to the net. In effect, each node added forms a new layer in the net and the network grows in one dimension, thus the depth of the net keep increasing when more nodes are added.

Our idea is to allow the network to grow in two dimensions in the course of adding nodes (Figure 2.4). This kind of architecture is an extension of the CasCor network so we name it the XCAS network.

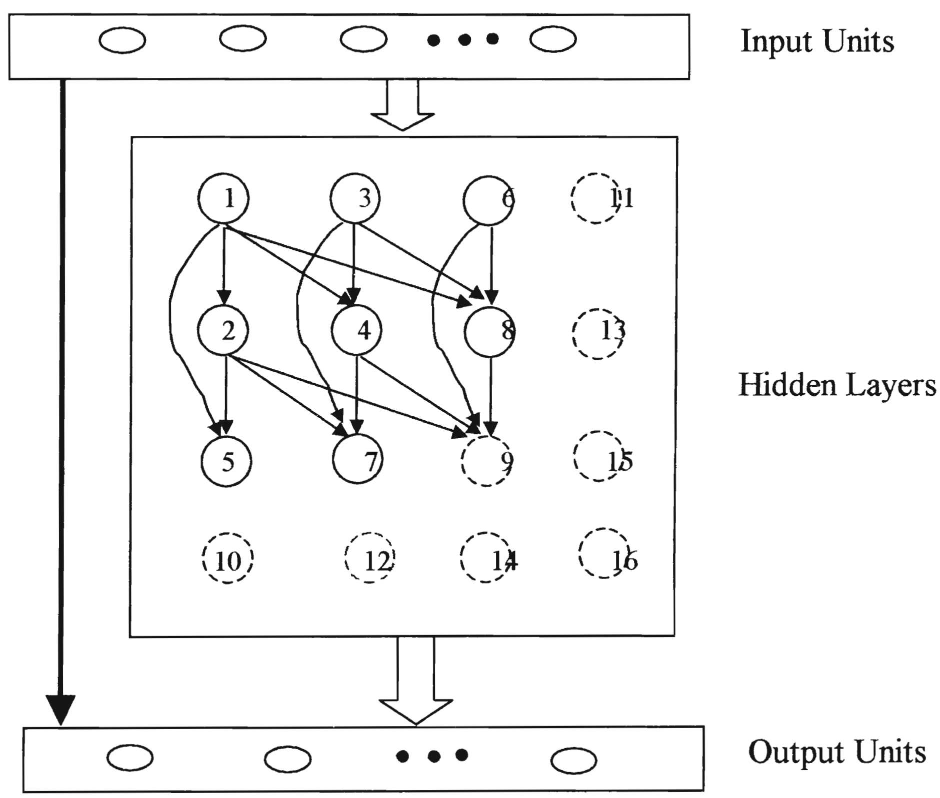


Figure 2.4: The proposed network architecture (XCAS). There are shortcut connections from input units to output units (shown by the arrowed line on the left side), from hidden units to hidden units (curved and arrowed line). Every node in the hidden layers also receives connections from all input units (block arrow). Every output unit also receives connections from all the hidden nodes in the network (block arrow). The number in the circle represents the order for addition of that node. Dashed circles represent nodes to be added later.

The network topology can be taken as an $m \times n$ matrix, where m, n are network layout parameters defined by user: m, n are the maximum number of layers (or rows) and the maximum number of columns permitted respectively. The CasCor is a special case of the XCAS for $n = 1$. Addition of nodes is to fill an empty $m \times n$ matrix according to a certain rule.

2.2.1 Addition of Nodes

Like the CasCor network, nodes are added to the net one at a time and their connection weights are frozen once added. For simplicity, we denote $node(i, j)$ as the node in the i-th row and j-th column of the network layout matrix.

1) *Symmetric addition*:

In an $n \times n$ network layout, nodes in the i-th row and i-th column of the network layout matrix are added symmetrically relative to diagonal nodes. When $node(i, k)$ is added, $node(k, i)$ is the next node to be added. This process repeats from $k = 1$ until $k = i$. After all positions in the i-th row and i-th column of the matrix are occupied, nodes are added to the next row and column in the same way when necessary.

The network will extend outwards along the diagonal of the matrix and keep as square a shape as possible (Figure 2.4). When N nodes have been added to the net, the depth is roughly the square root of N.

2) *Row-wise or Column-wise addition*:

In an $m \times n$ network layout, nodes are added symmetrically first on an $n' \times n'$ sub-matrix where $n' = \min(m, n)$, then added row after row if $m > n$, or column after column if $n > m$.

The following procedure determines the order according to which *node*(i, j) is added given an $m \times n$ network layout. The results are stored in two-dimensional arrays of integer *neuron* and *neuseq*.

```
Init_NetLayout(m, n, neuron, neuseq)
  INTEGER m, n, neuron(m, n), neuseq(m×n, 2)
  k =1
  min_mn = MIN( m, n )
  DO  i =1,  min_mn
      DO  j =1, min_mn
          neuron( i, j) = k
          neuseq( k,1) = i
          neuseq( k, 2) = j
          k = k  + 1
          IF ( i  ≠ j ) THEN
              neuron( j, i) = k
              neuseq( k, 1) = j
              neuseq( k, 2) = i
              k = k + 1
          END IF
```

```
        END DO
    END DO
    IF ( n > min_mn ) THEN
        DO  j = ( min_mn + 1 ),  n
            DO  i = 1,  m
                neuron( i, j ) = k
                neuseq( k, 1) = i
                neuseq( k, 2) = j
                k = k + 1
            END DO
        END DO
    END IF
    IF ( m > min_mn ) THEN
        DO  i = ( min_mn + 1), m
            DO  j = 1,  n
                neuron( i,  j) = k
                neuseq( k, 1) = i
                neuseq( k, 2) = j
                k = k + 1
            END DO
        END DO
    END IF
    END Init_NetLayout
```

The row index and column index for the k-th node to be added are given by *neuseq*(k, 1) and *neuseq*(k, 2) respectively. The *node*(i, j) will be the k-th node to be added for k = *neuron*(i, j).

There may be many strategies to explore for addition of hidden nodes in the XCAS network. We only discuss the schemes described above in this paper.

2.2.2 Connections

Each node in the net can receive forward pass connections from previous adjacent layer, shortcut connections from hidden nodes and from original inputs. Nodes in the first layer receive connections only from original inputs.

In the XCAS network, *node*(i, j) receives connections from:

1) *node*(i-1, k) for k = 1 to j;

   Those are nodes from the previous adjacent layer but with column index $\leq j$; number of forward pass connections = j; $i > 1$;

2) *node*(k, j) for k = 1 to i-2;

   Those are nodes from previous layers but in the same column as *node*(i, j); number of shortcut connections from hidden nodes = i-2;

3) shortcut connections from original inputs;

   number of shortcut connections from original inputs = number of inputs p;

Each node in the same column has the same pattern of connections as that of the CasCor network, but receives more connections from adjacent layers (constant 1 for CasCor). In actual implementation, shortcut connections from hidden nodes and/or original inputs can be enabled or disabled by the user, so XCAS is flexible for experimenting different connection schemes with the same network layout.

For $node(i, j)$, the number of weights $N_w(i, j)$ can be expressed as:

$N_w(i, j) = p + 1$ for $i = 1$;

$N_w(i, j) = j + p + 1$ for $i = 2$;

$N_w(i, j) = i + j + p - 1$ for $i > 2$;

where p is the number of original inputs.

Now we can calculate the total number of weights N_w assuming that the final network layout is $n \times n$ with all connections enabled as described above.

The total number of hidden nodes is $N = n^2$ in this case. Let M_1, M_2, M_3 be the total numbers of forward pass connections, shortcut connections from hidden nodes and from original inputs respectively. We have:

$$M_1 = \sum_{i=1}^{n} \sum_{j}^{n} (1 + p) = (1 + p) n^2$$

$$M_2 = \sum_{i=1}^{n-1} \sum_{j=1}^{n} j = \frac{1}{2} \cdot n(n^2 - 1) \qquad \text{for } i > 1$$

$$M_3 = \sum_{i=1}^{n-2} \sum_{j=1}^{n} i = \frac{1}{2} \cdot n(n+1)(n-2) \qquad \text{for } i > 2$$

The total number of weights for output nodes = $q(1 + p + n^2)$. So

$$\begin{aligned} N_w &= M_1 + M_2 + M_3 + q(1 + p + n^2) \\ &= n^3 + (p + q + \frac{1}{2}) n^2 - \frac{3}{2} n + q(1 + p) \\ &= N\sqrt{N} + (p + q + \frac{1}{2}) N - \frac{3}{2} \sqrt{N} + q(1 + p) \end{aligned}$$

This indicates the total number of weights is $\Theta(N^{3/2})$.

In comparisons with the CasCor network as seen in Table 2-1, the total number of weights, depth of the net and maximum fan-in of the hidden nodes are reduced significantly and also grow much slower when the number of hidden nodes become large.

Table 2-1: Growth rates of architectural parameters with number of hidden nodes

PARAMETERS	CasCor	XCAS
Depth of the network	$\Theta(N)$	$\Theta(N^{1/2})$
Maximum fan-in of the hidden nodes	$\Theta(N)$	$\Theta(N^{1/2})$
Total number of weights	$\Theta(N^2)$	$\Theta(N^{3/2})$

2.3 Learning Algorithm and Implementation

The learning algorithm used in CasCor network is also suitable for XCAS network although XCAS network has different architecture from CasCor. Another reason is that better and unbiased comparisons can be made between CasCor and XCAS if we employ the same algorithm for experimenting. For clarity, we describe here the main features of CasCor learning algorithm in pseudocode.

2.3.1 Objective Functions

In the CasCor learning algorithm, training cycle is divided into two phases: *input training* and *output training*.

Input training trains input weights of the candidate node by *maximizing* S, which is the covariance of the candidate's activation and the residual error developed before the candidate is added to the net:

$$S = \sum_{o} \left| \sum_{p} (C_p - \overline{C})(E_{p,o} - \overline{E_o}) \right|$$

where C_p is the activation of the candidate for pattern p, $\overline{C}$ is the activation of the candidate averaged over the set of all training patterns, $E_{p,o}$ is the residual error observed for pattern p at output unit o, and $\overline{E_o}$ is the average linear deviation at output unit o.

The partial derivative of S with respect to w_j is given by

$$\frac{\partial S}{\partial w_j} = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E_o}) f'_p \cdot in_{j,p}$$

where σ_o is the sign of the covariance for the candidate at output unit o, f'_p is the derivative for pattern p of the candidate's activation function with respect to its sum of inputs, and $in_{j,p}$ is the input to the candidate node for the pattern p and associated with w_j.

During the input training, there are no real connections from the candidate to output nodes because the candidate does not pass its activation to output nodes at this time. For this reason, a pool of several candidates can be trained independently at the same time and only the best one will be selected into the network. The covariance developed by a candidate during training depends on the random initialization of its input weights. The use of a pool of candidates thus greatly reduces the chance that a bad candidate caused by bad weight initialization will be added to the network.

Output training trains the weights of the output nodes by *minimizing* the squared error function E. E and its derivatives with respect to the weights of output nodes are

$$E = \frac{1}{2}\sum_p \sum_o (y_{p,o} - t_{p,o})^2$$

$$\frac{\partial E}{\partial w_{o,j}} = \sum_p \sum_o (y_{p,o} - t_{p,o}) f'_{p,o} \cdot in_{p,j}$$

where $y_{p,o}$ is the actual net output at output unit o for pattern p. $t_{p,o}$ is the corresponding target value, $f'_{p,o}$ is the derivative for pattern p of the output node's activation function with respect to its sum of inputs, and $in_{p,j}$ is the input to the output node for pattern p and associated with w_{oj}. For regression problems, output nodes usually use identity activation function such that $f'_{p,o} = 1$, so computation of derivatives is simpler.

2.3.2 Learning Algorithm

The training starts with no hidden nodes, so output training is performed first. If the criteria are met, the network ends in a network without any hidden nodes; otherwise, hidden nodes are trained and added to the network until some criteria are satisfied.

The main training procedure works as follows:

```
Train_Net
    FirstTime = TRUE
    REPEAT
        IF ( NOT FirstTime) THEN
```

```
            Input_Training
            FirstTime = FALSE
        END IF
        Output_Training
    UNTIL ( END_TRAIN_NET )
```

The termination criteria *END_TRAIN_NET* can be:

(i) Error measure value $\leq$ Error tolerance ε

(ii) Number of hidden nodes $>$ Permitted number of hidden nodes

The error tolerance ε and the permitted number of hidden nodes are set by the user. The training stops when either of conditions (i) and (ii) is met.

The error measure we used here can be squared error (SQE), mean squared error (MSE), square root of mean squared error (RMSE), normalized mean squared error (NMSE), squared error percentage (SQEP) or error index (EIDX). All those measures are based on squared error. Their definitions are listed in Appendix A.

The input training works as the follows:

```
Input_Training
  Initialize_Candidates
  Evaluate_Covariance_S
  REPEAT
      Compute_Derivatives
      Update_Candidate_Weights
      Evaluate_Covariance_S
```

UNTIL (*END_INPUT_TRAINING*)

The termination criteria *END_INPUT_TRAINING* include:

(i) Epochs trained > Permitted maximum epochs

(ii) Change rates in covariance values ≤ Change threshold ε

Input training stops when either of the above conditions is satisfied.

An epoch is defined as one pass through the entire set of training examples. Condition (ii) means progress stagnation of input training where the highest covariance value produced by any of the candidates has not changed by greater than a threshold value ε in the last *k* epochs. So there are three parameters selected by the user for the termination of input training: *permitted maximum epochs*, *input change threshold* ε and *patience parameter k*.

Initialize_Candidates : randomly initializes the weights of all candidates. The number of candidates used is set by user.

Evaluate_Covariance_S : computes values of the objective function S, i.e. covariance values for each candidates, and the signs of the covariance values that will be used in computation of derivatives.

Compute_Derivatives : computes derivatives with respect to the weights of candidates.

Update_Candidate_Weights : updates the weights of the candidates in order to maximize covariance.

After the termination of input training, the candidate with the highest covariance value is installed into the network and connected to the output nodes. The rest of the candidates are discarded. The weights of the output nodes associated with the newly installed node are also initialized with small values, the sign of which is the inverse of the covariance with the respective output unit.

The output training is similar to input training. In this phase, all the weights of hidden nodes currently in the network are frozen. A backward propagation of error through the hidden nodes is unnecessary, so output training is just like training a network without any hidden nodes but with additional input units.

```
Output_Training
    Compute_Error_Derivatives
    REPEAT
        Update_Output_Weights
        Compute_Error_Derivatives
    UNTIL ( END_OUTPUT_TRAINING )
```

The termination criteria *END_OUTPUT_TRAINING* are similar to those in input training:

(i) Error measure value $\leq$ Error tolerance ε

(ii) Epochs trained > Permitted maximum epochs

(iii) Change rate in error $\leq$ Change threshold ε'

Three user-selectable parameters for output training termination are: *permitted maximum epochs*, *patience parameter k* and *output change threshold* ε'.

Compute_Error_Derivatives : computes residual error and the derivatives with respect to the weights of the output nodes.

Update_Output_Weights : updates the weights of the output nodes in order to minimize *squared error.*

2.3.3 Implementation

The learning algorithm for the XCAS network, as described in the last section, has been implemented in FORTRAN 77.

When dealing with the learning algorithm, we did not mention a specific algorithm for updating weights. In fact, any existing algorithms for updating weights should work in our network. What most concerns us is whether the XCAS network can learn or not. In actual implementation, we used the quickprop algorithm proposed by Fahlman [24], which was also used in CasCor network.

The activation functions used and the corresponding derivatives are

1) *Identity Function*

$f(x) = x;$

$f'(x) = 1.0;$

2) *Asymmetrical Sigmoid Function*

$f(x) = 1.0/(1.0+\exp(-x));$ $\qquad 0 < f(x) < 1.0$

$f'(x) = f(1.0 - f);$

3) *Symmetrical Sigmoid Function*

$f(x) = 1.0/(1.0+\exp(-x)) - 0.5;$ $\qquad -0.5 < f(x) < 0.5$

$f'(x) = 0.25 - f^2;$

4) *Hyperbolic Tangent Sigmoid Function*

$$f(x) = (1.0 - \exp(-x)) / (1.0 + \exp(-x)); \qquad -1.0 < f(x) < 1.0$$

$$f'(x) = 1.0 - f^2;$$

All those functions can be used for output units. The identity function is usually not used for hidden units.

Chapter 3. TEST RESULTS ON REGRESSION PROBLEMS

In this Chapter we will perform experimentation with XCAS network and present test results and comparisons between the CasCor and XCAS.

3.1 Setup

In order to get consistent test results, common user-selectable parameters in the training algorithm were specified on an identical basis for the CasCor and the XCAS network.

Permitted maximum epochs for input and output training = 100;

Patience parameter for input and output training = 8;

Number of candidates = 8;

Learning rate for input training = 0.75;

Learning rate for output training = 0.35;

Maximum growth factor for learning rate = 1.75;

Input change threshold = 0.03;

Output change threshold = 0.01;

We used the symmetrical sigmoid function with output range between - 0.5 and + 0.5 for all the hidden units, and identity function for all the output units. The weights of the candidate nodes were randomly initialized between - 0.5 and + 0.5. The network layout for the XCAS was chosen as n by n (square layout) for the test although other layouts can be explored.

3.2 Regression Problems

We carried out experiments with the CasCor and XCAS networks on the following five non-linear functions, which had been used in [25] :

(1) *Simple Interaction Function*

$$f(x_1, x_2) = 10.391(0.36 + (x_1 - 0.4) \cdot (x_2 - 0.6))$$

(2) *Radial Function*

$$f(x_1, x_2) = 24.234 \cdot r^2 (0.75 - r^2); \qquad r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2$$

(3) *Harmonic Function*

$$f(x_1, x_2) = 42.659(0.1 + y_1 (0.05 + y_1^4 - 10 y_1^2 y_2^2 + 5 y_1^4));$$
$$y_1 = (x_1 - 0.5); \qquad y_2 = (x_2 - 0.5);$$

(4) *Additive Function*

$$f(x_1, x_2) = 1.3356(1.5(1 - x_1) + e^{2x_1 - 1} \sin(3\pi (x_1 - 0.6)^2) + e^{3(x_2 - 0.5)} \sin(4\pi (x_2 - 0.9)^2))$$

(5) *Complicated Interaction Function*

$$f(x_{1,} x_2) = 1.9(1.35 + e^{x_1} \sin(13(x_1 - 0.6)^2) \cdot e^{-x_2} \sin(7 x_2))$$

For simplicity, we refer to the above functions as F1, F2, F3, F4 and F5 respectively in simulation. All the five functions have two inputs and single output. Experiments were also made on combinations of the five functions for the purpose of testing the networks on problems with multiple outputs and increased complexity. The two test groups used

Group-I: each of the five functions is treated as an independent problem;

Group-II: the first 2, 3, 4, and 5 of the five functions are combined into four problems with different number of outputs:

CF2: F1 and F2 as the outputs of the network;

CF3: F1, F2, and F3 as the outputs of the network;

CF4: F1, F2, F3 and F4 as the outputs of the network;

CF5: F1, F2, F3, F4 and F5 as the outputs of the network;

Group-II is also used to test the learning ability of the networks when several independent problems are put together for training. If there exist the similarities to a great degree between the independent problems, we expect the networks should be able to take advantage of those similarities, and the total numbers of hidden nodes and weights used would be less than the sum of those for independent learning.

The training and test data sets are generated on a regular grid with the same range [0, 1] for the two inputs of the five functions. The abscissa values of the two independent variables for the training data set are sampled as

$$x_{i,j} = \frac{j-1}{n-1}; \qquad \text{for } i = 1, 2; \quad j = 1, 2, \ldots, n;$$

Similarly, the two inputs for the test data set are sampled as

$$x_{1,1} = 0;$$

$$x_{1,j} = \frac{j-1}{n-1} - \frac{1}{2(n-1)}; \qquad j = 2, 3, \ldots, n;$$

$$x_{2,j-1} = \frac{j-1}{n-1} - \frac{1}{2(n-1)}; \qquad j = 2, 3, \ldots, n;$$

So, the number of examples produced is n^2 for the training data set and $n(n-1)$ for the test set. The test set is independent of the training data set. We assumed n =15 in our simulation, thus obtained 225 examples for the training data set and 210 examples for the test set. We used the same set of input data pairs $\{(x_{1,j}, x_{2,j})\}$ generated for the experiments with all the five functions.

3.3 Test Results and Comparisons

The goal of our testing is to explore the learning ability of the XCAS network and assess its performance compared with the CasCor network. We carried out 10 runs on each problem with different random seeds for initialization of the candidates.

Table 3-1: Random seeds used in initialization of weights

Trial No.	1	2	3	4	5	6	7	8	9	10
Seed	7	48	77	173	231	378	455	571	601	737

We used the error index (EIDX) defined in Appendix A as the error measure in the training, and the threshold value 0.1 for stopping the training for all experiments. The final error is reported in squared error percentage (SQEP) over the set of all training examples.

3.3.1 Test Results on Group-I Problems

The average number of hidden nodes used by the CasCor and XCAS networks (Figure 3.1) shows that the complexities of the problems remain the same for both networks, in other words, the problem that is difficult for the CasCor network to learn is still difficult for the XCAS network. Generally, XCAS needs a few more nodes on each problem than CasCor, but needs much more nodes to learn the harmonic function (F3), which is the most difficult problem for both networks. The number of hidden nodes in the final network is a good indicator of the complexity of the problem, but not a good measure for comparisons between networks of different architectures and connections.

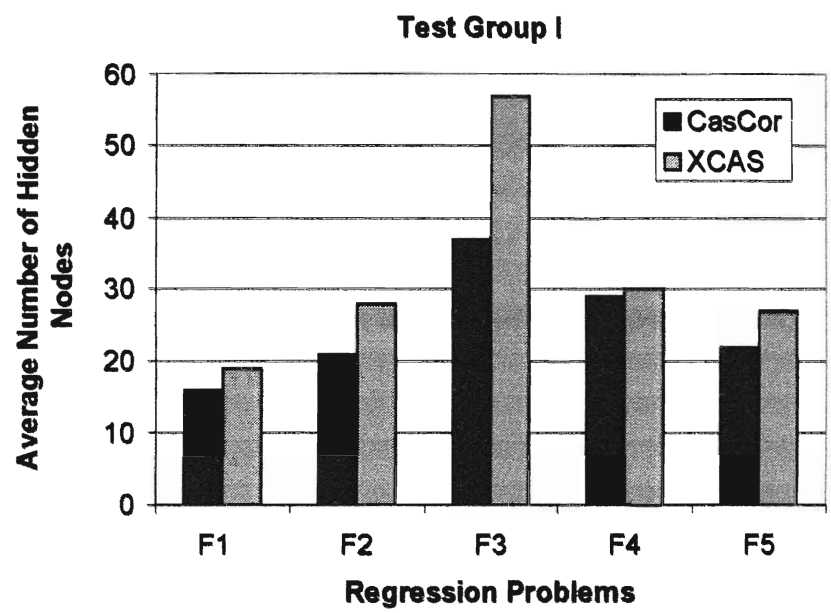


Figure 3.1: Average number of hidden nodes (Group-I)

As we have seen, the XCAS uses more nodes than the CasCor, however, it uses fewer weights than CasCor (Figure 3.2). The percentages reduced on the total number of weights by XCAS against CasCor range from 25.6% to 54.4%, at least 31% for F1, F3, F4 and F5, or in other words, XCAS only needs about 45% ~ 75% of the total number of weights used by CasCor on the same problem.

The results obtained here indicate XCAS is able to learn all of the problems tested and is also more efficient in usage of weights than the CasCor.

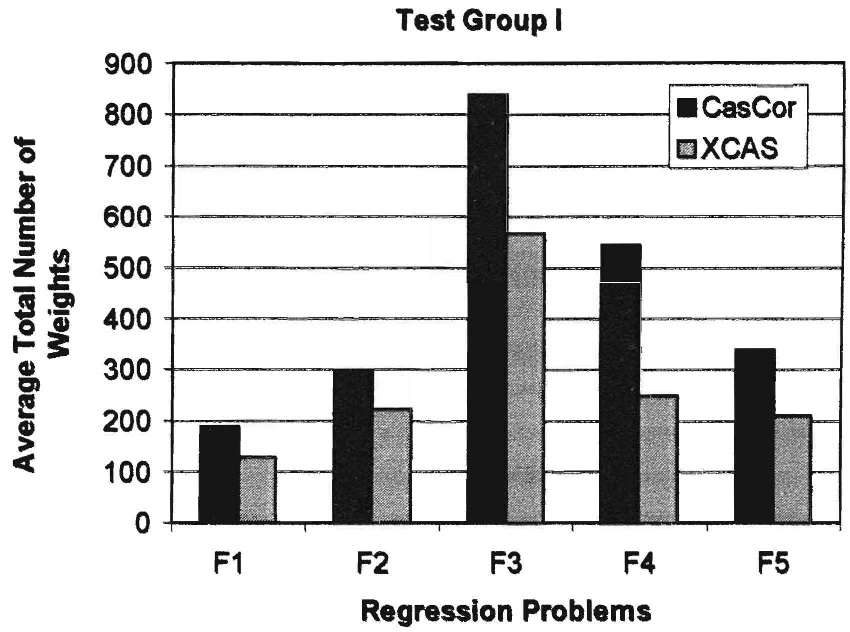


Figure 3.2: Average total number of weights (Group-I)

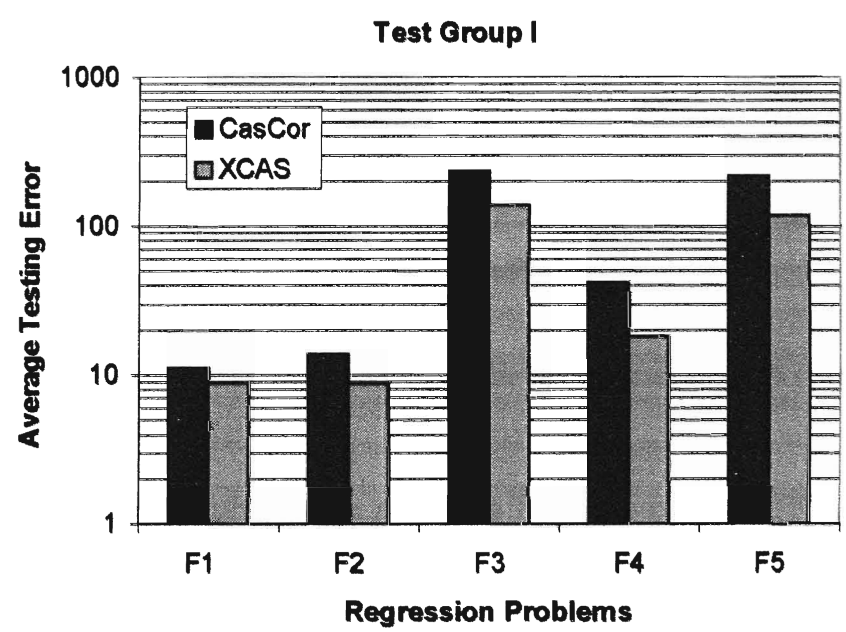


Figure 3.3: Average squared error percentage on the test set (Group-I)

The average errors yielded by CasCor on the test set (Figure 3.3) are 1.3 ~ 2.3 times as large as those produced by XCAS. In other words, XCAS is 1.3 ~ 2.3 times better than the CasCor by the ability of generalization, which is measured by the performance on the test set. The results here suggest that the network with a lesser number of weights tends to have better generalization. Both the CasCor and XCAS networks produced larger errors on problems F3 and F5 than on other problems (see Figure 3.3).

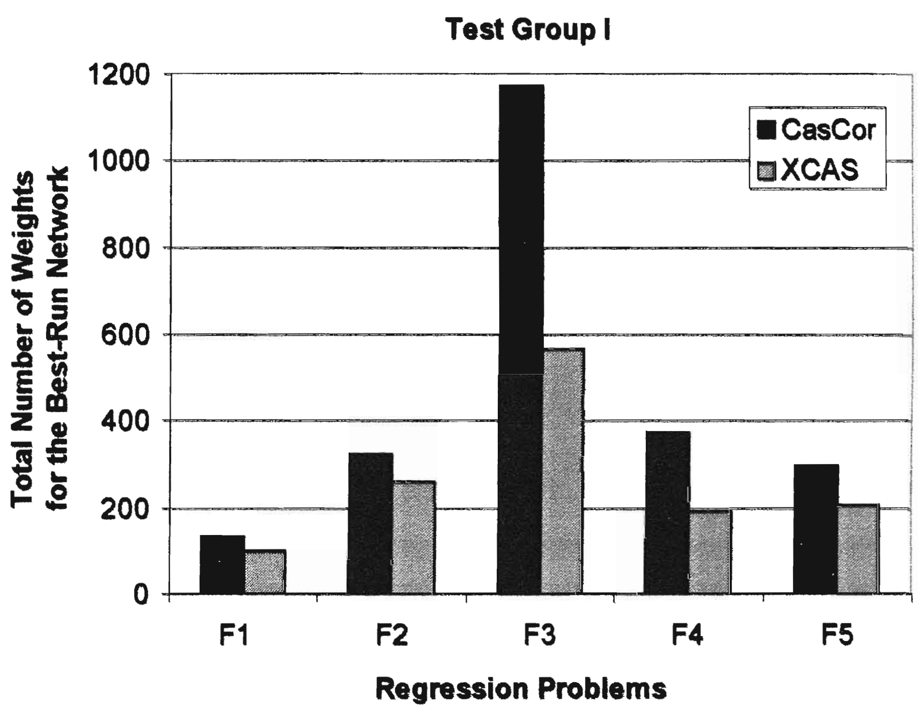


Figure 3.4: Total number of weights for the best-run network (Group-I)

The total numbers of weights in the networks from the best run (Figure 3.4) which gives the minimum error over the training data set, show the same trends indicated by the averages (see Figure 3.2). This suggests that the best-run network topology is close to the average network obtained over all runs. The testing error yielded by the best-run network (Figure 3.5) is generally scaled down in magnitude compared with the average value (see Figure 3.3) for both networks, but it becomes scaled up about 3 times on F3 for the

CasCor and 2 times on F5 for the XCAS. The XCAS performs much better than the CasCor for the most difficult problem, F3. The best-run networks do not definitely produce lower testing error than the average testing error.

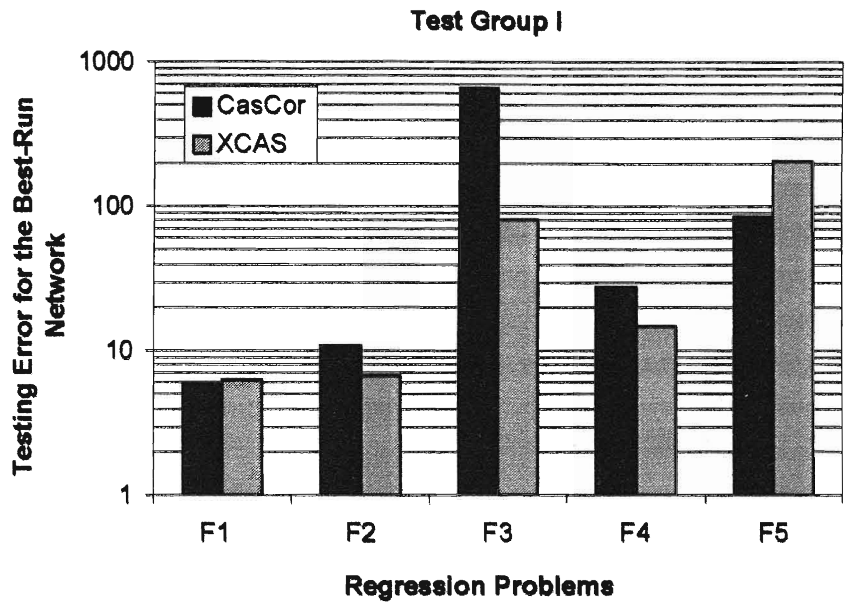


Figure 3.5: Squared error percentage on the test set for the best-run network (Group-I).

The simulation results on the test Group-I demonstrate that XCAS is able to learn all of the problems tested and uses a lesser number of weights than CasCor, and also has better performance on the test set in general.

3.3.2 Test Results on Group-II Problems

All the problems in Group-II have numbers of outputs varying from 2 to 4 although they have the same number of inputs. The average number of hidden nodes (Figure 3.6) keeps increasing with the increased number of outputs. XCAS still needs a few more nodes on each problem compared with CasCor.

It is interesting to note that the maximum number of hidden nodes used for Group-II problems is less than 70, about 10 more nodes than that for Group-I problems (Figure 3.1). This implies that both networks are able to take advantage of the similarities between those problems, thus the number of nodes needed for learning them together is less than the sum of those for learning them independently.

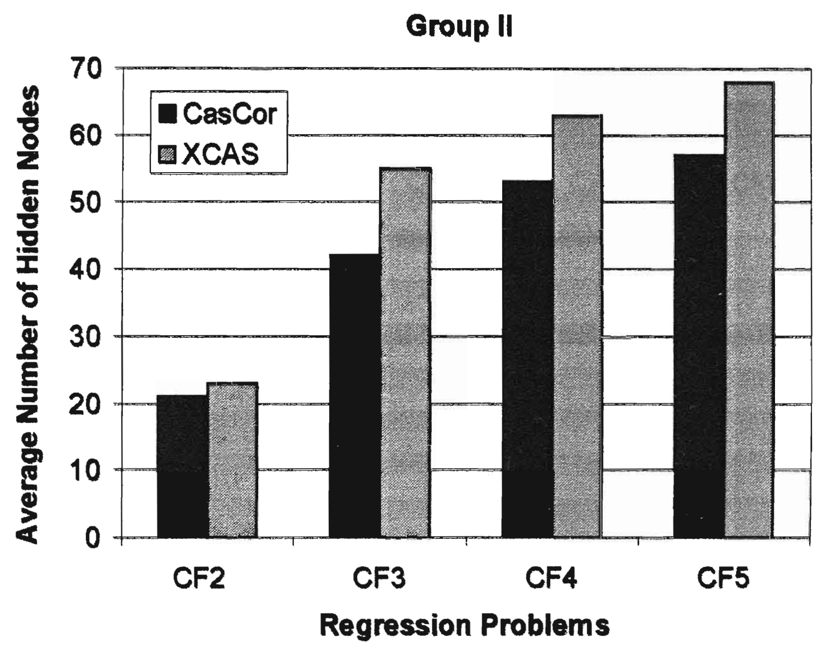


Figure 3.6: Average number of hidden nodes (Group-II).

As to Group-I problems, the XCAS network still needs a smaller number of weights to learn Group-II problems (Figure 3.7). The percentages of the average number of weights reduced by XCAS against CasCor range from 39.4% to 52.8%. In other words, XCAS only needs about 47% ~ 61% of the total weights used by CasCor for the Group-II problems. The results we obtained until now confirm that XCAS is able to learn complex problems as CasCor does but uses many fewer weights.

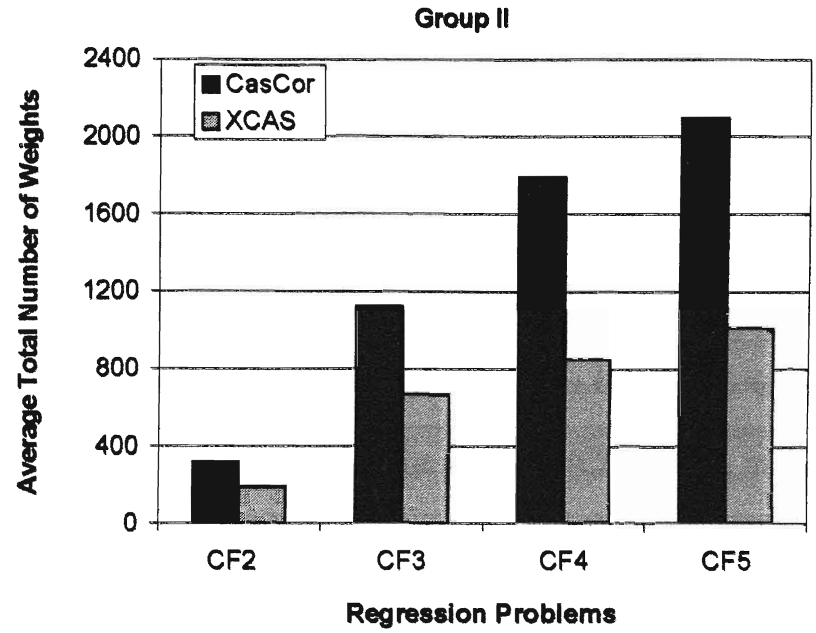


Figure 3.7: Average total number of weights (Group-II).

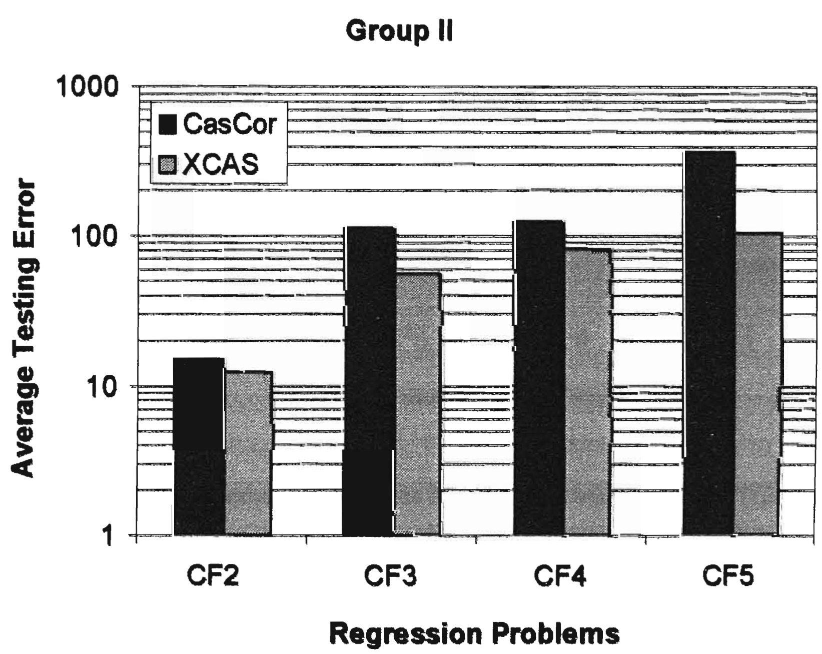


Figure 3.8: Average squared error percentage on the test set (Group-II).

The testing errors produced by CasCor are 1.2 ~ 3.4 times as large as those by XCAS (Figure 3.8), indicating that XCAS has better performances on the test set in general although fewer weights are used.

The total numbers of weights in the best-run networks (Figure 3.9) show a similar trend to that indicated by the averages (see Figure 3.7). The percentages of the total number of weights reduced by XCAS against CasCor are 31.0% ~ 42.8% for the first two problems (CF2, CF3), and at least 56.0% for the last two (CF4, CF5), indicating that the XCAS network uses many fewer weights than the CasCor network when the complexity of the problem increases.

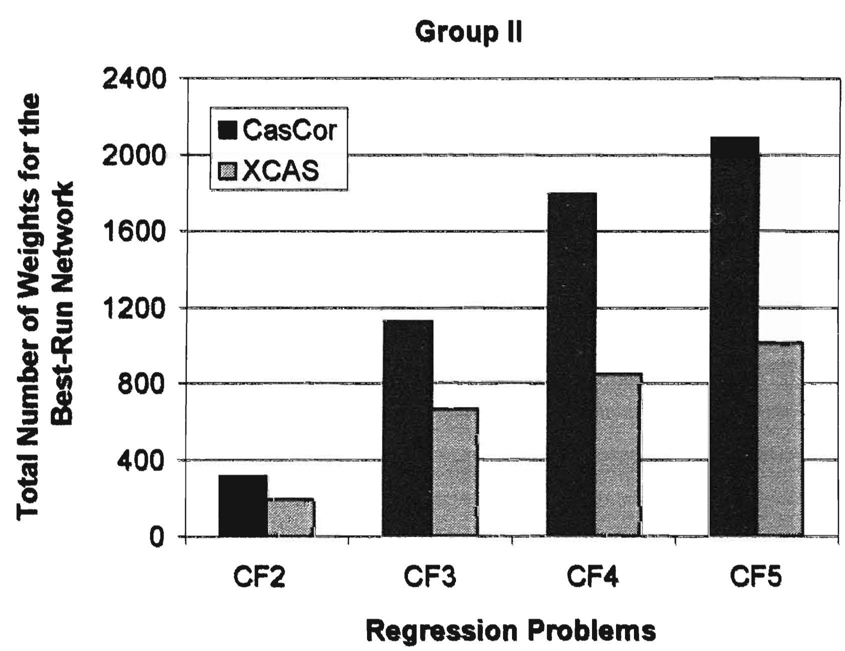


Figure 3.9: Total number of weights for the best-run network (Group-II)

As to the best-run networks, XCAS network still shows better performance on the test set than CasCor (Figure 3.10) in general, especially for the last two problems (CF4, CF5) which are more difficult than the first two problems.

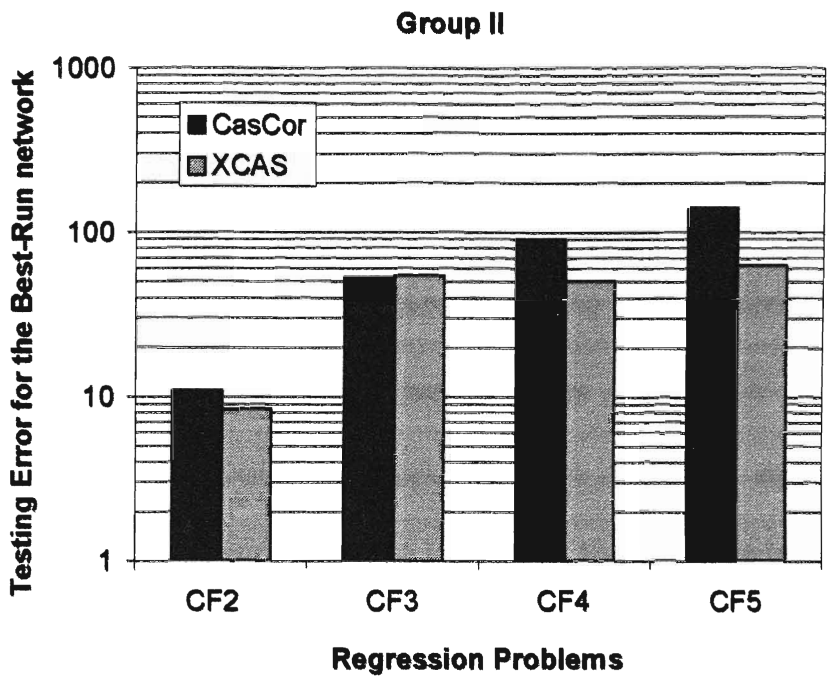


Figure 3.10: Squared error percentage on the test set for the best-run network (Group-II)

The test results on the Group-I and Group-II problems demonstrate that the XCAS network is able to learn all of regression problems tested, and has better performance on the test set than CasCor in general but uses fewer weights.

Chapter 4. CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

This study proposes a new network architecture (XCAS) based on extension of the Cascade-Correlation network (CasCor). Theoretically the total number of weights in the final network grows with the number of hidden nodes N as $N\sqrt{N}$ for the proposed network and as N^2 for the CasCor network. The test results on regression problems confirm that the proposed network is able to learn all of the problems tested, and not only use fewer weights but also exhibit better performance on the test set in general as opposed to the CasCor network. On the average, the percentages of the total number of weights reduced by XCAS against CasCor range from 25% to 55% for the corresponding problems tested, and increase with the number of hidden nodes used.

4.2 Recommendation for Future Work

Further investigations could be done in the following places:

(1) Use direct error minimization instead of the covariance maximization in training the hidden nodes so that the network is more suitable for regression problems.

(2) Allow hidden nodes to be of different types of activation functions. The network with variable types of hidden nodes may be more flexible for various and complicated problems.

(3) Use *arctan(x)* instead of a logistic or *tanh(x)* activation function.

(4) Extend the two-dimensional XCAS architecture to three or more dimensions.

REFERENCES

[1] Bartlett, P. L., "For Valid Generalization the Size of the Weights is More Important than the Size of the Network", in *Advances in Neural Information Processing Systems*, vol. 9, p. 134, The MIT Press, 1997.

[2] Amaldi, E. and E. Guenin, "Two Constructive Methods for Designing Compact Feedforward Networks of Threshold Units", *International Journal of Neural Systems*, vol. 8, Nos. 5 & 6, pp. 629-645, 1997.

[3] Chen, K., Liping Yang , Xiang Yu and Huisheng Chi, "A Self-Generating Modular Neural Network Architecture for Supervised Learning", *Neurocomputing -- An International Journal*, vol. 16, No. 1, pp. 33-48, 1997.

[4] Cybenko, G., "Approximation by Superposition of a Sigmoidal Function", *Mathematics of Control, Signals, and Systems*, vol. 2, No. 4, pp. 303-314, 1989.

[5] Duch, W. and N. Jankowski, "Survey of Neural Transfer Functions", *Neural Computing Surveys*, vol. 2, pp. 163-213, 1999.

[6] Fahlman, S.E. and C. Lebiere, "The Cascade-Correlation Learning Algorithm", Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[7] Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, vol. 2, No. 3, pp. 183-192, 1989.

[8] Hassibi, B. and D.G Stork. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon", in *Advances in Neural Information Procesing Systems*, vol. 5, pp. 164-171, Morgan Kaufmann, San Mateo, CA, 1993.

[9] Hornik, K., Stinchombe, M., White, H. "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, vol. 2, No. 5, pp. 359-366, 1989.

[10] Kwok, T.-Y. and D.-Y. Yeung, "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems", *IEEE Transactions on Neural Networks*, vol. 8, No. 3, pp. 630-645, 1997.

[11] Lawrence, S., C. L. Giles and A. C. Tsoi, "What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation", Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, 1996.

[12] LeCunn, Y., J.S. Denker, and S.A. Solla, "Optimal Brain Damage", in *Advances in Neural Information Processing Systems*, vol. 2, pp. 598-605, Morgan Kaufmann, San Mateo, CA, 1990.

[13] Littmann, E. and H. Ritter, "Cascade Network Architectures", in *Proceedings of the International Joint Conference on Neural Networks*, Baltimore, MD, USA, vol. 2, pp. 398-404, June 1992.

[14] Littmann, E. and H. Ritter, "Cascade LLM Networks", in *Artificial Neural Networks*, vol. 2, pp. 253-257, Elsevier Science Publishers B.V., 1992.

[15] Mozar, M.C., "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment", in *Advances in Neural Information Proceeding Systems*, vol. 1, pp. 107-115, Morgan Kaufmann, San Mateo, CA, 1989.

[16] Niyogi, P., & Girosi, F., "On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions", *Neural Computation*, vol. 8, No. 4, pp. 819-842, 1996.

[17] Van de Laar, P, and Heskes, T., "Pruning Using Parameter and Neuronal Metrics", *Neural Computation*, vol. 11, No. 4, pp. 977-993, 1999.

[18] Park, Y.R., Murray, T.J., & Chen, C., "Predicting Sun Spots Using a Layered Perceptron Neural Network", *IEEE Transactions on Neural Networks*, vol. 7, No. 2, pp. 501-505, 1996.

[19] Phatak, D. S. and Koren, I., "Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture", *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 930-935, 1994.

[20] Prechelt, L., "Investigation of the CasCor Family of Learning Algorithms", *Neural Networks*, vol. 10, No. 5, pp. 885-896, 1997.

[21] Reed, R., "Pruning Algorithms- A Survey", *IEEE Transactions on Neural Networks*, vol. 4, No. 5, pp. 741-747, 1993.

[22] Weigend, A., "On Overfitting and the Effective Number of Hidden Units", in *Proceedings of the 1993 Connectionist Models Summer School*, pp. 335-342, Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.

[23] Yao, X., "Evolving Artificial Neural Networks", *Proceedings of the IEEE*, vol. 87, No. 9, pp. 1423-1447, 1999.

[24] Fahlman, S. E., "An Empirical Study of Learning Speed in Back-Propagation Networks", Technical Report CMU-CS-88-162, Carnegie Mellon University, 1988.

[25] Hwang, J.N., S.R. Lay, M. Maechler, D. Martin, and J. Schimert, "Regression Modeling in Back-Propagation and Projection Pursuit Learning", *IEEE Transactions on Neural Networks*, vol. 5, No. 3, pp. 342-353, 1994.

APPENDICES

Appendix A--Error Measures

SQE: squared error;

MSE: mean squared error;

RMSE: square root of mean squared error;

NMSE: normalized mean squared error;

SQEP: squared error percentage;

EIDX: error index;

m : number of training examples or patterns;

n : number of outputs or dimensions of the output vector;

$y_{i,j}$: actual output of the network for the pattern i at output unit j;

y_{max} : maximum value of the actual outputs of the network;

y_{min} : minimum value of the actual outputs of the network;

$t_{i,j}$: target or desired output for the pattern i at output unit j;

$\bar{t}$: average target or desired outputs over the set of all training patterns;

(1) $$\text{SQE} = \sum_{i}^{m}\sum_{j}^{n}(y_{i,j} - t_{i,j})^2$$

(2) $$\text{MSE} = \frac{1}{m \cdot n}\sum_{i}^{m}\sum_{j}^{n}(y_{i,j} - t_{i,j})^2 = \frac{1}{m \cdot n} \cdot \text{SQE}$$

(3) $$\text{RMSE} = \sqrt{\frac{1}{m \cdot n} \cdot \sum_{i}^{m} \sum_{j}^{n} (y_{i,j} - t_{i,j})^2} = \sqrt{\text{MSE}}$$

(4) $$\text{NMSE} = \frac{y_{max} - y_{min}}{m \cdot n} \cdot \sum_{i}^{m} \sum_{j}^{n} (y_{i,j} - t_{i,j})^2 = (y_{max} - y_{min}) \cdot \text{MSE}$$

(5) $$\text{SQEP} = 100 \cdot \frac{y_{max} - y_{min}}{m \cdot n} \cdot \sum_{i}^{m} \sum_{j}^{n} (y_{i,j} - t_{i,j})^2 = 100 \cdot (y_{max} - y_{min}) \cdot \text{MSE}$$

(6) $$\text{EIDX} = \sqrt{\frac{1}{m \cdot n} \cdot \sum_{i}^{m} \sum_{j}^{n} (y_{i,j} - t_{i,j})^2} \Bigg/ \sqrt{\frac{1}{m \cdot n - 1} \cdot \sum_{i}^{m} \sum_{j}^{n} (t_{i,j} - \bar{t})^2}$$

$$= \text{RMSE} \Bigg/ \sqrt{\frac{1}{m \cdot n - 1} \cdot \sum_{i}^{m} \sum_{j}^{n} (t_{i,j} - \bar{t})^2}$$

Appendix B--Tables of Test Results

Simulations were performed on two test groups (see Chapter 3.2) for the CasCor and XCAS networks. Group-I includes five problems labeled as F1, F2, F3, F4 and F5. Group-II includes four problems labeled as CF2, CF3, CF4 and CF5, which are combinations of problems from Group-I.

The numbers of training examples and testing examples are 225 and 210 respectively, and are held constant for all the problems in Group-I and Group-II. Each problem was executed ten times on the CasCor and XCAS. The averages are made over ten runs.

Table A-1: Test results on Group-I for the CasCor network

Problem Label	Item		Average	Min	Max	Max-Min	Standard Deviation	Best Run
F1	Number of Hidden Nodes		15.9	12	20	8	2.8	13
	Total Number of Weights		188.6	117	273	156	54.6	133
	Training Error	SQEP	6.87	6.51	7.15	0.64	0.019	6.51
		RMSE	0.1088	0.1084	0.1091	0.0006	0.0002	0.1087
	Testing Error	SQEP	11.09	5.81	34.55	28.74	8.49	5.81
		RMSE	0.1362	0.1043	0.2602	0.1559	4.61	0.1043
F2	Number of Hidden Nodes		21.0	17	24	7	1.9	22
	Total Number of Weights		298.6	207	375	168	45.2	322
	Training Error	SQEP	3.62	3.51	3.74	0.23	0.009	3.51
		RMSE	0.1006	0.0996	0.1008	0.0011	0.0003	0.1003
	Testing Error	SQEP	13.77	7.19	44.84	37.65	1.162	10.83
		RMSE	0.1787	0.1383	0.3180	0.1796	0.0555	0.1679
F3	Number of Hidden Nodes		37.4	33	45	12	3.3	45
	Total Number of Weights		838.1	663	1173	510	139.7	1173
	Training Error	SQEP	12.81	12.64	13.08	0.44	0.013	12.64
		RMSE	0.1227	0.1223	0.1229	0.0006	0.0002	0.1226
	Testing Error	SQEP	237.02	116.20	653.46	537.26	16.74	653.46
		RMSE	0.5298	0.3850	0.9498	0.5648	0.1774	0.9498
F4	Number of Hidden Nodes		29.3	20	36	16	4.7	24
	Total Number of Weights		544.9	273	777	504	149.5	375
	Training Error	SQEP	5.46	5.25	5.75	0.50	0.015	5.25
		RMSE	0.1029	0.1022	0.1031	0.0009	0.0002	0.1029
	Testing Error	SQEP	41.98	20.17	64.95	44.79	1.661	27.32
		RMSE	0.2892	0.2050	0.3644	0.1594	0.0582	0.2385
F5	Number of Hidden Nodes		22.4	17	30	13	3.6	21
	Total Number of Weights		338.2	207	558	351	98.1	297
	Training Error	SQEP	43.82	42.81	44.77	1.95	0.07	42.81
		RMSE	0.2105	0.2086	0.2190	0.0023	0.0007	0.2106
	Testing Error	SQEP	217.26	83.84	544.27	460.43	13.109	83.84
		RMSE	0.4580	0.2981	0.7511	0.4531	0.1254	0.2981

Table A-2: Test results on Group-I for the XCAS network

Problem Label	Item		Average	Min	Max	Max-Min	Standard Deviation	Best Run
F1	Number of Hidden Nodes		18.7	15	23	8	2.3	15
	Total Number of Weights		129.3	99	170	71	19.7	99
	Training Error	SQEP	6.85	6.66	7.19	0.58	0.017	6.60
		RMSE	0.1090	0.1087	0.1091	0.0004	0.0001	0.1091
	Testing Error	SQEP	8.78	6.28	15.51	9.23	0.313	6.28
		RMSE	0.1240	0.1069	0.1628	0.0558	0.0198	0.1069
F2	Number of Hidden Nodes		28.3	21	33	12	4.1	32
	Total Number of Weights		222.1	149	272	123	41.1	260
	Training Error	SQEP	3.62	3.46	3.72	0.25	0.010	3.46
		RMSE	0.1007	0.0997	0.1008	0.0010	0.0003	0.0997
	Testing Error	SQEP	8.65	3.50	20.23	16.73	0.460	6.67
		RMSE	0.1505	0.1003	0.2343	0.1340	0.0358	0.1366
F3	Number of Hidden Nodes		56.7	46	70	24	7.4	57
	Total Number of Weights		567.8	426	749	323	98.6	565
	Training Error	SQEP	12.84	12.59	13.10	0.50	0.017	12.59
		RMSE	0.1228	0.1227	0.1229	0.0002	0.0001	0.1229
	Testing Error	SQEP	138.21	79.91	426.93	347.02	10.614	79.91
		RMSE	0.4125	0.3276	0.7470	0.4194	0.1313	0.3276
F4	Number of Hidden Nodes		30.2	13	47	34	10.1	25
	Total Number of Weights		248.5	81	441	360	106.4	193
	Training Error	SQEP	5.41	5.26	5.63	0.37	0.011	5.26
		RMSE	0.1031	0.1031	0.1032	0.0001	0.0000	0.1031
	Testing Error	SQEP	18.15	7.82	45.11	37.28	1.088	14.77
		RMSE	0.1867	0.1248	0.2990	0.1741	0.0490	0.1772
F5	Number of Hidden Nodes		26.9	21	33	12	4.4	27
	Total Number of Weights		208.8	149	272	123	44.5	206
	Training Error	SQEP	43.44	41.90	44.81	2.90	0.099	41.90
		RMSE	0.2107	0.2102	0.2109	0.0007	0.0002	0.2107
	Testing Error	SQEP	117.20	66.27	250.44	184.17	5.368	250.44
		RMSE	0.3427	0.2623	0.5159	0.2536	0.0729	0.5159

Table A-3: Test results on Group-II for the CasCor network

Problem Label	Item		Average	Min	Max	Max-Min	Standard Deviation	Best Run
CF2	Number of Hidden Nodes		20.7	18	23	5	1.9	18
	Total Number of Weights		315.0	249	374	125	47.5	249
	Training Error	SQEP	9.64	9.34	10.02	0.67	0.020	9.34
		RMSE	0.1261	0.1258	0.1262	0.0004	0.0001	0.1262
	Testing Error	SQEP	15.01	9.42	47.49	38.07	1.149	10.67
		RMSE	0.1528	0.1255	0.2949	0.1694	0.0504	0.1357
CF3	Number of Hidden Nodes		41.9	37	46	9	2.8	40
	Total Number of Weights		1120.9	897	1320	423	134.1	1029
	Training Error	SQEP	16.74	16.28	17.14	0.85	0.027	16.28
		RMSE	0.1397	0.1392	0.1399	0.0006	0.0002	0.1397
	Testing Error	SQEP	111.48	47.22	262.45	215.23	7.191	52.42
		RMSE	0.3468	0.2334	0.5535	0.3201	0.1065	0.2500
CF4	Number of Hidden Nodes		53.3	46	63	17	5.4	54
	Total Number of Weights		1792.0	1369	2406	1037	329.6	1821
	Training Error	SQEP	16.71	16.39	16.88	0.49	0.019	16.39
		RMSE	0.1406	0.1403	0.1408	0.0005	0.0002	0.1403
	Testing Error	SQEP	122.28	88.17	223.56	135.39	4.633	89.93
		RMSE	0.3710	0.3175	0.5380	0.2205	0.0710	0.3302
CF5	Number of Hidden Nodes		57.3	54	59	5	1.9	59
	Total Number of Weights		2088.0	1878	2198	320	120.7	2198
	Training Error	SQEP	26.94	26.26	27.57	1.30	0.042	26.26
		RMSE	0.1595	0.1594	0.1596	0.0002	0.0000	0.1596
	Testing Error	SQEP	352.69	139.82	919.4	779.58	28.691	139.82
		RMSE	0.5374	0.3655	0.9472	0.5871	0.2140	0.3655

Table A-4: Test results on Group-II for the XCAS network

Problem Label	Item		Average	Min	Max	Max-Min	Standard Deviation	Best Run
CF2	Number of Hidden Nodes		22.5	18	26	8	2.1	21
	Total Number of Weights		190.6	142	231	89	23.8	173
	Training Error	SQEP	9.69	9.39	9.89	0.49	0.014	9.39
		RMSE	0.3114	0.3066	0.3145	0.0079	0.0023	0.3066
	Testing Error	SQEP	12.24	8.30	31.91	23.61	0.768	8.38
		RMSE	0.3389	0.2881	0.5649	0.2768	0.0916	0.2896
CF3	Number of Hidden Nodes		55.4	50	62	12	3.9	50
	Total Number of Weights		664.2	589	774	185	59.4	589
	Training Error	SQEP	16.52	16.22	17.01	0.78	0.025	16.22
		RMSE	0.1397	0.1394	0.1399	0.0004	0.0000	0.1399
	Testing Error	SQEP	55.11	43.59	70.26	26.67	0.691	54.34
		RMSE	0.2546	0.2304	0.2847	0.0543	0.0138	0.2556
CF4	Number of Hidden Nodes		62.6	59	71	12	3.8	60
	Total Number of Weights		846.2	781	986	205	64.4	800
	Training Error	SQEP	16.90	16.55	17.17	0.62	0.019	16.55
		RMSE	0.1407	0.1405	0.1408	0.0002	0.0001	0.1408
	Testing Error	SQEP	81.70	44.44	270.99	226.55	6.803	50.23
		RMSE	0.2942	0.2291	0.5506	0.3215	0.0950	0.2427
CF5	Number of Hidden Nodes		67.8	62	75	13	3.8	65
	Total Number of Weights		1005.9	904	1140	236	66.1	963
	Training Error	SQEP	26.73	25.55	27.42	1.86	0.051	25.55
		RMSE	0.1569	0.1594	0.1597	0.0002	0.0000	0.1596
	Testing Error	SQEP	104.22	63.1	218.23	155.14	4.462	63.10
		RMSE	0.3071	0.2450	0.4586	0.2135	0.0608	0.2450

Appendix C--Program Source Code

XCAS network was implemented in FORTRAN77. Five files in plain text format must be created before the program is executed.

1) Network Configuration File: storing setup parameters and weights.

Example:

```
FIRST LINE
REM  NETSTA, NETLAY, NETCOL, HNUTYP, ONUTYP, ISHORT, HSHORT
        0        16        16        2        0        1        1
REM  HMXEPC, HBONUS, HCANDS, OMXEPC, OBONUS
        100        8        8        100        8
REM  HLRNRT, HMXLRN, HTHRES, WRANGE
        0.75        1.75        0.03        0.5
REM  OLRNRT, OMXLRN, OTHRES, ODECAY, ETOLER
        0.35        1.75        0.01        0.01        0.1
REM  NUMLAY, NUMNEU, ERNORM, PERCNT , ERRMSR
        0        0        1        0        5
REM  INPUTS, OUTPTS, VLDMOD,  VLDVAL, NSEEDS
        0        0        1        0        3
REM  RANDOM SEEDS
 77
 139
 354
```

The first line will be replaced by the program with the training data file name after training is finished. Lines starting with REM provide names for values appeared below and should not be removed. Values in boldface here are set by user, and must be separated by at least one space. Weights of the trained network will be appended to this file.

NETSTA: should be 0 when the network is going to be trained and non-zero when trained.
NETLAY: maximum number of layers permitted by user.
NETCOL: maximum number of columns permitted by user.
HNUTYP: types of hidden nodes represented by activation functions. Valid values: 1~3.
ONUTYP: types of output nodes. Valid values: 0 ~ 3.
ISHORT: enable (set to 1) or disenable (set to 0) shortcut connections to original inputs.
HSHORT: enable (set to 1) or disenable (set to 0) shortcut connections to hidden nodes.

HMXEPC: maximum epochs permitted for training a hidden node.
HBONUS: patience parameter for training a hidden node.
HCANDS: number of candidates used for training a hidden node.
OMXEPC: maximum epochs permitted for training an output node.

OBONUS: patience parameter for training an output node.

HLRNRT: learning rate for training hidden nodes, usually 0.5 ~ 2.5.
HMXLRN: maximum learning factor for training hidden nodes, usually 1.0 ~ 2.5.
HTHRES: threshold of change rate for training hidden nodes, usually 0.02 ~ 0.05.
WRANGE: weights will be initialized randomly between – WRANGE and + WRANGE.

OLRNRT: learning rate for training output nodes, usually 0.5 ~ 2.5.
OMXLRN: maximum learning factor for training hidden nodes, usually 1.0 ~ 2.5.
OTHRES: threshold of change rate for training output nodes, usually 0.01 ~ 0.05.
ODECAY: decay factor for updating weights of output nodes, usually < 0.05
ETOLER: error tolerance for training the network. The value depends on the error measure used.

NUMLAY: number of layers in the final network trained.
NUMNEU: number of hidden nodes in the final network trained.
ERNORM: specifies whether the error is expressed in normalized form (1) or not (0).
PERCNT: specifies whether the error is expressed in percentages (1) or not (0).
ERRMSR: specifies what kind of error measure will be used.
=1, mean squared error (MSE).
=2, square root of mean squared error (RMSE).
=3, error index (EIDX).
The definitions for the above measures are listed in Appendix A.

INPUTS: number of inputs used for the network, determined by training data file.
OUTPTS: number of outputs used for the network, determined by training data file.
VLDMOD: mode for selecting validation set from training data file.
VLDVAL: value associated with VLDMOD. Valid values depend on VLDMOD.
VLDMOD =1, create validation set from whole training data, starting from VLADVAL+1 to N, where N is the number of training examples.
VLDMOD =2 ~ 4, example j is in validation set if (j mod VLDMOD = VLDVAL).

NSEEDS: number of seeds (positive integers) for random number generator.

2) Training Data File: training data (training set + validation set) and test set.

Exmple:

```
BOOL_INPUTS=   0
REAL_INPUTS=   2
BOOL_OUTPUTS=  0
REAL_OUTPUTS=  1
TRAIN_SET=   225
VALID_SET=     0
TEST_SET=    210
  0.00000  0.00000  8.53180
  0.00000  0.07142  8.06264
  0.00000  0.14286  6.93275
  0.00000  0.21429  5.50860
  0.00000  0.28571  4.09001
  0.00000  0.35714  2.91015
  ..............................
```

The header (7 lines in total) includes information about attributes of input(s) and output(s), sizes of training set, validation set and test set. There must be at least one space between equal signs and values assigned. Each line in the data section includes input(s) and desired output(s) (shown in boldface in the above example). Data items are separated by at least one space. In data section, the training set comes first, then the validation set and test set. There should be no blank lines in data section.

3) Report file: an empty file used for storing statistical results about the network training. Information includes:

HNUS: number of hidden nodes.
HWTS: number of weights for hidden nodes.
TWTS: total number of weights (for hidden and output nodes).
IEPC: epochs used for input training.
OEPC: epochs used for output training.
TEPC: total number of epochs used in network training.
MSE: mean squared error.
RMS: square root of mean squared error.
VAR: variance of actual network outputs.
STD: standard deviation of actual network outputs.
IEW: EPCW for input training.
OEW: EPCW for output training.

$EPCW = (\Sigma K_j \cdot N_j)/1000$
K_j = weight-updating epochs for node j;
N_j = number of weights for node j;
EPCW is a measure of times spent on training the nodes in the network.

4) Input Data File: inputs to a trained network.

Example:

```
2          24
6.23460   3.02925
5.93771   3.32448
0.00000   0.14286
5.64083   3.40775
......................
```

The first line gives number of inputs and number of data lines.

5) Output File: an empty file used for storing network outputs

```
CCCCCCCCCCCCC      XCAS NEURAL NETWORK      CCCCCCCCCCCCCCCCCCCCCCCCCC
C PROGRAMMED IN FORTRAN77, COMPILABLE UNDER G77 IN UNIX SYSTEM
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C NAMING CONVENTION:
C IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
CCCCCCCCCCCCCCCCCCCC    CONSTANTS CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C USED IN ALLOCATION OF STORAGE, MUST BE SET BEFORE COMPILED
C ALSO COPIED TO COMMON /MAXCON/ BLOCKS
C   MXI: MAXIMUM NUMBER OF INPUTS
C   MXO: MAXIMUM NUMBER OF OUTPUTS
C   MXE: MAXIMUM NUMBER OF TRAINING EXAMPLES
C   MXT: MAXIMUM NUMBER OF RANDOM SEEDS ( ONE SEED USED IN EACH TRIAL)
C   MXD: MAXIMUM NUMBER OF CANDIDATES USED IN INPUT TRAINING
C   MXL: MAXIMUM NUMBER OF LAYERS
C   MXC: MAXIMUM NUMBER OF COLUMNS
C
CCCCCCCCCCCCCCCCCCCCCCC    VARIABLES   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX
C     = NUMBERS OF INPUTS, OUTPUTS, TRAINING EXAMPLES,TEST EXAMPLES,
C       VALIDATION EXAMPLES  RESPECTIVELY
C
C
C USER SELECTABLE PARAMETERS:
C   NTRIAL= NUMBER OF TRIALS = NUMBER OF RANDOM SEEDS
C   WRANGE= THE RANGE FOR RANDOM INITIALIZATION OF WEIGHTS
C   NETLAY= MAXIMUM NUMBER OF LAYERS PERMITTED <= MXL
C   NETCOL= MAXIMUM NUMBER OF COLUMNS PERMITTED <= MXC
C   MODVAL= MODE FOR SELECTING VALIDATION SET FORM TRAING EXAMPLES
C   MODREM= REMAINDER VALUE ASSOCIATED WITH MODVAL
C   NOXINP= ENABLE OR DISABLED SHORTCUT CONNECTIONS TO INPUTS
C   NOVERT= ENABLE OR DISABLED SHORTCUT CONNECTIONS TO HIDDEN NODES
C   IDXMSR= INDEX FOR CHOOSING AN ERROR MEASURE FOR OUTPUT TRAINING
C
C FOR INPUT TRAINING
C   MXEPOC= MAXIMUM NUMBER OF TRAINING EPOCHS PERMITTED
C   NBONUS= PATIENCE PARAMETER FOR INPUT TRAINING
C   NTRANS= TYPE OF ACTIVATION FUNCTION FOR HIDDEN NODES
C   ALPHA = LEANING RATE
C   BETA  = MAXIMUM LEARNING FACTOR
C   GAMMA = MOMENTUM
C   SHRINK= SHRINK FACTOR = BETA/(BETA+1.0)
C   THRESH= THRESHHOLD OF INPUT CHANGE RATE
C   NUMCND= NUMBER OF CANDIDATES USED
C
C FOR OUTPUT TRAINING
C   MXEPCO= MAXIMUM NUMBER OF TRAINING EPOCHS PERMITTED
C   NEUO  = TYPE OF ACTIVATION FUNCTION FOR OUTPUT UNITS
C   NEPCO = OUTPUT TRAINING EPOCHS USED
C   NBONO = PATIENCE PARAMETER FOR OUTPUT TRAINING
C   ALPHO, BETO, GAMMO, SHNKO, THREO
C         = LEANING RATE, MAXIMUM LEARNING FACTOR, MOMENTUM,
C           SHRINK FACTOR, THRESHHOLD OF INPUT CHANGE RATE FOR OUTPUT
C           TRAINING
C   DECAYO= DECAY FACTOR (USED IN QUICKPROP)
C   ERRTHR= THRESHHOLD VALUE FOR STOPING TRAINING THE NET
```

```
C
C OTHER VARIABLES:
C   NBESTC= THE INDEX OF THE BEST CANDIDATE
C   NEPOCH= INPUT TRAINING EPOCHS USED
C   NEPCO = OUTPUT TRAINING EPOCHS USED
C   BSTSCR= THE BEST SCORE OF CANDIDATES
C
C FILE NAMES:
C   NETFNM= NETWORK CONFIGRATION FILE, STORING THE SETUP PARAMETERS AND
C           WEIGHTS OF TRAINED NETWORK
C   RPTFNM= REPORT FILE THAT GIVES STATISTICAL RESULTS ABOUT TRAINING
C   TRNFNM= TRAINING DATA FILE (ALL TRAINING EXAMPLES)
C   RUNFNM= INPUT DATA FILE FOR RUNNING A TRAINED NETWORK
C   OUTFNM= OUTPUT FILE FOR A TRAINED NETWORK
C
CCCCCCCCCCCCCCCCCCCCCCCCC     ARRAYS      CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   NEUR(I,J): STORING THE SEQUENTIAL ORDER FOR ADDING THE NODE IN THE
C              I-TH LAYER, J-TH COLUMN
C
C   NEUS(K,2): IF K= NEUR(I,J),THEN I= NEUS(K,1),J= NEUS(K,2)
C   XINP(*,K): INPUT VECTOR OF THE K-TH EXAMPLE
C   DOUT(*,K): TARGET OUTPUT VECTOR FOR THE K-TH INPUT EXAMPLE
C   YOUT(*,K): ACTUAL NETWORK OUTPUT FOR THE K-TH INPUT EXAMPLE
C   HOUT(I,J,K): THE OUTPUT OF HIDDEN NODE(I,J) FOR THE K-TH EXAMPLE
C   HWTS(*,I,J): WEIGHTS OF THE NODE(I,J)
C   OWTS(*,J): WEIGHTS OF THE OUTPUT NODE J
C   OSLP(*,J): DERIVATIVES W.R.P TO WEIGHTS OF THE OUPUT NODE J
C   OPSL(*,J): THE PREVIOUS OSLP(*,J)
C   ODWT(*,J): CHANGES IN WEIGHTS OF OUPUT NODE J
C   ERRO(J,K): RESIDUAL ERROR PRODUCED AT OUPUT NODE J FOR EXAMPLE K
C   CDOU(J): AVERAGE OUPUTS OF THE CANDIDATE J OVER ALL THE EXAMPLES
C   CCOR(J,I,2): COVARIANCE VALUES OF CANDIDATE I AT OUPUT NODE J
C   CWTS(*,J): WEIGHTS OF THE CANDIDATE J
C   SLOP(*,J): DERIVATIVES W.R.P TO WEIGHTS OF THE CNADIDATE J
C   PSLP(*,J): PREVIOUS SLOP(*,J)
C   DWTS(*,J): CHANGES IN WEIGHTS OF THE CANDIDATE J
C   TSTH(I,J): TEMPORARY ARRAY STORING THE OUTPUTS OF NODE(I,J) FOR
C              SINGLE INPUT EXAMLE
C
C   MRSEED(*): STORING RANDOM SEEDS
C   NETFIG(N,*): INFORMATION ABOUT FINAL NETWORK ARCHITECTURE OBTAINED
C                AT N-TH RUN
C   TRNERR(N,*): INFORMATION ABOUT TRANING ERROR FOR THE N-TH RUN
C   TSTERR(N,*): INFORMATION ABOUT TESTING ERROR FOR THE N-TH RUN
C   VLDERR(N,*): INFORMATION ABOUT ERROR ON VALIDATION SET FOR THE N-TH
C                RUN
C
CCCCCCCCCCCCCCCCCC SUBROUTINES AND FUNCTIONS CCCCCCCCCCCCCCCCCCCCCCCCC
C  NAME OF SUBROUTINE OR FUNCTION
C      IF FOLLOWED BY { ...}:
C        SUBROUTINES CALLED (PREFIXED WITH CALL)AND/OR FUNCTIONS CALLED
C      ELSE : NOT CALL OTHER SUBROUTINES/FUNCTIONS
C
C
C  MAIN {CALL ININET,CALL SETNET,CALL RDPROB,CALL SETNUR,CALL
C        TRAIN,CALL GETSTA,CALL WRTNET,CALL TEST,CALL SHOWER,CALL
```

```
C       REPORT,CALL RUNNET}
C
C  SUBROUTINE ADJCOR
C  SUBROUTINE CNDSLP{FPRIME,OUTHNU,NUMHWT}
C  SUBROUTINE COMERR{MARKOP,OPRIME}
C  SUBROUTINE COREPC{CALL ADJCOR,OUTHNU}
C  SUBROUTINE ERRSTA
C  SUBROUTINE GETERV{CALL GETSTD}
C  SUBROUTINE GETSTA{CALL GETERV,NCONEX}
C  SUBROUTINE GETSTD{MARKOP}
C  SUBROUTINE HNUPAS{OUTHNU}
C  SUBROUTINE ININET
C  SUBROUTINE INITNN
C  SUBROUTINE OUTPAS{FTRANS}
C  SUBROUTINE OWTNEW
C  SUBROUTINE QKPROP
C  SUBROUTINE REPORT{CALL ERRSTA,CALL SHOWER}
C  SUBROUTINE RDHEAD
C  SUBROUTINE RDPROB{CALL GETSTD}
C  SUBROUTINE RUNNET{CALL HNUPAS,CALL OUTPAS}
C  SUBROUTINE SETNUR
C  SUBROUTINE SETANN{CALL SETNUR,CALL INITNN}
C  SUBROUTINE SETNET{CALL RDHEAD,CALL SETANN,NUMHWT}
C  SUBROUTINE SHOWER
C  SUBROUTINE TEST{CALL HNUPAS, CALL OUTPAS,CALL GETERV}
C  SUBROUTINE TRAIN{CALL OWTNEW,MTROUT,MTRINP,RANDOM}
C  SUBROUTINE TRNOUT{CALL OUTPAS,CALL COMERR,CALL GETERV,CALL UPOWTS}
C  SUBROUTINE UPHWTS{CALL QKPROP}
C  SUBROUTINE UPOWTS{ CALL QKPROP}
C  SUBROUTINE WRTNET{CALL WTHEAD,NUMHWT}
C  SUBROUTINE WTHEAD
C
C  FUNCTION FPRIME
C  FUNCTION FTRANS
C  FUNCTION MARKOP
C  FUNCTION MTRINP{CALL CNDSLP,CALL UPHWTS,CALL COREPC,CALL ADJCOR,
C                  NUMHWT, RANDOM,OUTHNU}
C  FUNCTION MTROUT{CALL TRNOUT}
C  FUNCTION NCONEX
C  FUNCTION NUMHWT
C  FUNCTION OPRIME
C  FUNCTION OUTHNU{NUMHWT,FTRANS}
C  FUNCTION RANDOM
C
CCCCCCCCCCCCCCCC       MAIN  PROGARM    CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCC
      PROGRAM XCAS
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      PARAMETER(MXI=36,MXO=5,MXE=4400,MXT=20,MXD=8,MXS=10)
      PARAMETER(MXL=16,MXC=16)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EPCWTS
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
```

```
      COMMON /STATIS/ TSQE,SQER,VSQE,VEIX,VDSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORMER,NEUO,NCENT,NRESO
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,WTSCRO
      COMMON /NVAMOD/ MODVAL, MODREM
      COMMON /MODNEX/ NOXINP, NOVERT
      COMMON /NETTOP/ NETLAY,NETCOL
      CHARACTER*30 NETFNM,TRNFNM,RUNFNM,OUTFNM,RPTFNM
C
      DIMENSION NEUR(MXL,1+MXC)
      DIMENSION NEUS(MXL*MXC,2)
      DIMENSION XINP(MXI,MXE)
      DIMENSION DOUT(MXO,MXE)
      DIMENSION YOUT(MXO,MXE)
      DIMENSION HOUT(MXL,MXC,MXE)
      DIMENSION HWTS(1+MXI+MXL+MXC,MXL,MXC)
C
      DIMENSION OWTS(1+MXI+MXL*MXC,MXO)
      DIMENSION OSLP(1+MXI+MXL*MXC,MXO)
      DIMENSION OPSL(1+MXI+MXL*MXC,MXO)
      DIMENSION ODWT(1+MXI+MXL*MXC,MXO)
      DIMENSION ERRO(MXO,1+MXE)
C
      DIMENSION CDOU(MXD)
      DIMENSION CCOR(MXO,MXD,2)
      DIMENSION CWTS(1+MXI+MXL+MXC,MXD)
      DIMENSION SLOP(1+MXI+MXL+MXC,MXD)
      DIMENSION PSLP(1+MXI+MXL+MXC,MXD)
      DIMENSION DWTS(1+MXI+MXL+MXC,MXD)
C
      DIMENSION TSTH(MXL,MXC)
C
      DIMENSION MRSEED(1+MXT)
      DIMENSION NETFIG(MXT,6)
      DIMENSION TRNERR(MXT,6)
      DIMENSION TSTERR(MXT,6)
      DIMENSION VLDERR(MXT,6)
C
      NZ=6
      NS=MXT
C  NET CONFIGRATION FILE NAME: UNIT=30
      NETFNM=''
C  TRAINING DATA FILE NAME: UNIT=31
      TRNFNM=''
C  RUNNING SET FILE NAME:  UNIT=32
      RUNFNM=''
C  NET OUTPUT FILE NAME:  UNIT=33
      OUTFNM=''
C  TRAINING AND TESTING REPORT FILE NAME:  UNIT=34
      RPTFNM=''
      NRPTFL=34
C
       WRITE(*,*)'Storage Limits For Network Layout:'
       WRITE(*,*)'Max_Layer= ',MXL, '   Max_Column= ',MXC
       WRITE(*,*)
C
      CALL ININET
```

```
      WRITE(*,*)'Enter configration file name:'
      READ(*,*)NETFNM
      CALL SETNET(HWTS,OWTS,MRSEED,NS,NEUR,NEUS,NETFNM,
     &                          MXI,MXL,MXC,MXO,MXD)
C
CCCCCC
      IF(NETSTA .EQ. 0) THEN
         WRITE(*,*)'You Are Going To Train The Net!'
         WRITE(*,*)
         WRITE(*,*)'Enter training_data, report file name:'
         READ(*,*)TRNFNM,RPTFNM
         CALL RDPROB(XINP,DOUT,TRNFNM,MXI,MXO,MXE)
         BSTERR=1.0D30
         DO 400 JSEED=1,NTRIAL
            CALL SETNUR(NEUR,MXL,MXC)
            NRSEED=MRSEED(JSEED+1)
            WRITE(*,*)'TRIAL ',JSEED,'  SEED= ',NRSEED
CCCCCC
             CALL TRAIN(XINP,HOUT,YOUT,DOUT,ERRO,HWTS,OWTS,ODWT,OSLP,
     &                 OPSL,CWTS,DWTS,SLOP,PSLP,CCOR,CDOU,NEUS,NEUR,
     &                 MXI,MXL,MXC,MXO,MXE,MXD)
C
            CALL GETSTA(NETFIG,NEUR,YOUT,TRNERR,VLDERR,NS,NZ,JSEED
     &                ,MXL,MXC,MXO,MXE)
C
            IF(VSQE .LT. BSTERR) THEN
               MRSEED(1)=JSEED
C
               CALL WRTNET(HWTS,OWTS,MRSEED,NS,NEUR,NETFNM,TRNFNM,
     &                      MXI,MXL,MXC,MXO)
C
               BSTERR=VSQE
            END IF
C
            CALL TEST(XINP,DOUT,YOUT,OWTS,TSTH,HWTS,NEUR,
     &                      TSTERR,NS,NZ,JSEED,MXI,MXL,MXC,MXO,MXE)
C
            WRITE(*,*)
           CALL SHOWER(NETFIG,TRNERR,VLDERR,TSTERR,NS,NZ,JSEED,0)
            WRITE(*,*)
400     CONTINUE
C
        OPEN(UNIT=NRPTFL,FILE=RPTFNM,STATUS='OLD')
C
        CALL REPORT(NETFIG,TRNERR,VLDERR,TSTERR,MRSEED,NS,NZ,
     &              NRPTFL,TRNFNM)
C
        CLOSE(NRPTFL)
         WRITE(*,*)'Configration file: ',NETFNM
         WRITE(*,*)'Training data file:',TRNFNM
C        WRITE(*,*)'Testing  set file: ',TSTFNM
         WRITE(*,*)'Trn & Tst  report: ',RPTFNM
CCCCCC
      ELSE
        WRITE(*,*)'You Are Going To Run On The Trained Net!'
        WRITE(*,*)
C
```

Table 9. LS Means for Production Characteristics by Treatment (Adjfat=0.4).

		Treatment (Frame Size X Muscle Score)					
Trait	Units	Small No.1	Small No.2	Med. No.1	Med. No.2	Large No.1	Large No.2
Purchase Weight of Cattle	Pounds	465.758	458.808	459.54	454.334	470.154	470.59
Standard Error		6.004	6.247	4.327	4.104	4.647	5.802
Backgrounding ADG	Pounds/Day	0.105	0.215	0.139	0.52	0.183	0.686
Standard Error		0.290	0.302	0.209	0.198	0.224	0.280
Pasture ADG	Pounds/Day	2.434	2.491	2.709	2.56	2.546	2.849
Standard Error		0.133	0.138	0.096	0.091	0.105	0.128
Feedlot ADG	Pounds/Day	3.252	3.746	3.723	3.396	3.387	3.646
Standard Error		0.157	0.163	0.113	0.107	0.121	0.152
Feed Efficiency In Feedlot	Feed/Gain in Pounds	6.651^{a}	6.828^{a}	6.881^{a}	7.477^{b}	7.922^{c}	7.673^{bc}
Standard Error		0.119	0.124	0.086	0.081	0.092	0.115
Days Fed in Feedlot	Days	105.384^{a}	106.545^{a}	121.216^{a}	137.2^{b}	152.307^{b}	141.93^{b}
Standard Error		4.939	5.139	3.560	3.376	3.823	4.773
Harvest Weight	Pounds	1064.345^{a}	1128.108^{ab}	1215.472^{bc}	1237.64^{c}	1289.412^{cd}	1336.958^{d}
Standard Error		24.543	25.539	17.689	16.776	18.998	23.717

a,b,c,d Means in the same row for the same item with a different superscript letter differ (P>.05).

```
              NEUR(NL,0)=1
            ELSE
              READ(NFILE,*)  NEUR(NL,0)
            END IF
              MAXCOL=MAX(MAXCOL,NEUR(NL,0))
100         CONTINUE
C
            IF(MAXCOL .GT. NETCOL) THEN
        WRITE(*,*)'Number of columns mismatched with network layout!'
              NFLAG=1
              GO TO 600
            END IF
C
C  READ WEIGHTS OF HIDDEN UNITS
            READ(NFILE,*)
            DO 300 NL=1, NUMLAY
               DO 200 NC=1,NEUR(NL,0)
                 NWTS=NUMHWT(NL,NC,NUMINP,NBX,NBC,NBV)+1
                 READ(NFILE,*)
                 DO 160 NW=1,NWTS
                    READ(NFILE,700)HW(NW,NL,NC)
160              CONTINUE
200            CONTINUE
300         CONTINUE
C
         END IF
C
C READ WEIGHTS OF OUTPUT UNITS
         READ(NFILE,*)
         NWTS=1+NUMINP+NUMNEU
         DO 500 NOUT=1,NUMOUT
           READ (NFILE,*)
           DO 400 NW=1,NWTS
             READ(NFILE,700) OW(NW,NOUT)
400        CONTINUE
500      CONTINUE
C
      END IF
C
600   CLOSE(NFILE)
      IF(NFLAG .GT. 0)STOP
700   FORMAT(1X,D24.15)
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C READ HEADER OF CONFIGRATION FILE
C  --GET SETUP PARAMETRS
CCCCCC
      SUBROUTINE RDHEAD(NFILE,NCODE,MXI,MXL,MXC,MXO,MXCND)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NCANDP/ MXEP,NBON,IDXMSR,NETSTA,METHOD,NUTYPH,NRSEED
      COMMON /STATIS/ SSQE,STDD,VMSE,VDEV,VSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TRMSE,TRDEV,TRSTD,VLDCOE,TRNCOE
      COMMON /NTRAIN/ NRETUR,NCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
```

```
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EPCWTH
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORM,NUTYPO,NCENT,NRESO
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,EPCWTO
      COMMON /NVAMOD/ MODV, MODR
      COMMON /MODNEX/ NOX, NOV
      COMMON /NETTOP/ NETLAY,NETCOL
C
      CHARACTER*30 NETF,TRNFNM,CH*80
C
      NCODE=0
      NETLAY=0
      NETCOL=0
C
      READ(NFILE,*)
      READ(NFILE,*)
      READ(NFILE,*)NETSTA,NETLAY,NETCOL,NUTYPH,NUTYPO,NOX,NOV
      READ(NFILE,*)
      READ(NFILE,*)MXEP,NBON,NCND,MXEPCO,NBONO
      READ(NFILE,*)
      READ(NFILE,*)ALPHA,BETA,THRESH,WRANGE
      READ(NFILE,*)
      READ(NFILE,*)ALPHO,BETO,THREO,DECAYO,ERRTHR
      READ(NFILE,*)
      READ(NFILE,*)NUMLAY,NUMNEU,NORM,NCENT,IDXMSR
      READ(NFILE,*)
      READ(NFILE,*)NUMINP,NUMOUT,MODV,MODR,NTRIAL
C
      IF((NETCOL .GT. MXC) .OR. (NETLAY .GT. MXL)) THEN
            WRITE(*,*)'Storage(',MXL,' BY ', MXC,' ) not enough for ',
     &              NETLAY, ' by ',NETCOL,'  network topology!'
          NCODE=1
          RETURN
      END IF
C
      DECAYO=DECAYO/100.0
      IF(NBONO .GT. MXEPCO) NBONO=8
      IF(NBONO .LT. 2) NBONO=4
C
      IF((IDXMSR .LT. 0) .OR. (IDXMSR .GT. 6)) THEN
         IDXMSR=5
         ERRTHR=0.2
         WRITE(*,*)'Invalid range of IDXMSR (1~6),'
         WRITE(*,*)'Default IDXMSR=5 and ERRTHR=0.2 used.'
         WRITE(*,*)
      END IF
C
      IF(NOX .NE. 0)NOX=1
      IF(NOV .NE. 0)NOV=1
C  CHECK CONFIG. FILE
C
      IF(NETSTA .NE. 0) THEN
        NCODE=1
        IF(NUMLAY .GT. NETLAY ) THEN
      WRITE(*,*)'Number of layers mismatched with network layout!'
        ELSE IF(NUMINP .GT. MXI) THEN
          WRITE(*,*)'Number of inputs over storage limit!'
        ELSE IF(NUMOUT .GT. MXO) THEN
```

```
            WRITE(*,*)'Number of outputs over storage limit!'
          ELSE IF(NCND .GT. MXCND) THEN
            WRITE(*,*)'Number of candidates over storage limit!'
          ELSE
            NCODE=0
          END IF
        ELSE
          NUMLAY=0
        END IF
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   SET UP NETWORK ARCHITECTURE
C   --SETUP OF CONNECTIONS
C   --SEQUENCE FOR ADDING NODES
CCCCCC
      SUBROUTINE SETANN(NEURON,NEUSEQ,MXL,MXC)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NETTOP/ NETLAY,NETCOL
C
      DIMENSION NEURON(MXL,0:MXC),NEUSEQ(MXL*MXC,2)
CCCCCC
      CALL SETNUR(NEURON,MXL,MXC)
C
      IF(NETCOL .EQ. 1) THEN
C FOR CASCOR NETWORK
        DO 100  NL=1,MXL
          NEURON(NL,1)=NL
          NEUSEQ(NL,1)=NL
          NEUSEQ(NL,2)=1
100     CONTINUE
      ELSE
C FOR XCAS NETWORK
        CALL INITNN(NEURON,NEUSEQ,MXL,MXC)
      END IF
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C ZERO THE ARRAY OF NETWORK LAYOUT
CCCCCC
      SUBROUTINE SETNUR(NEURON,MXL,MXC)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      DIMENSION NEURON(MXL,0:MXC)
C
      DO 100 NL=1,MXL
          NEURON(NL,0)=0
100   CONTINUE
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  INTIALIZE ARRAY NEUR AND NEUS
```

```
C  --SET SEQUENCE ACCORDING TO WHICH THE NEURON WILL BE ADDED
C  --SYMETRIC ADDITION OF NODES RELATIVE TO DIAGONAL OF THE MATRIX
CCCCCC
      SUBROUTINE INITNN(NEURON,NEUSEQ,MXL,MXC)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NETTOP/ NETLAY,NETCOL
C
      DIMENSION NEURON(MXL,0:MXC),NEUSEQ(MXL*MXC,2)
C
      MINSQR=MIN(NETLAY,NETCOL)
      K=1
      DO 200   I=1, MINSQR
         DO 100  J=1, I
             NEURON(I,J)=K
             NEUSEQ(K,1)=I
             NEUSEQ(K,2)=J
             K=K+1
           IF (I .NE. J) THEN
             NEURON(J,I)=K
             NEUSEQ(K,1)=J
             NEUSEQ(K,2)=I
             K=K+1
           END IF
100     CONTINUE
200   CONTINUE
C
      IF(NETCOL .GT. MINSQR) THEN
        DO 400  J=(MINSQR+1), NETCOL
           DO 300 I=1, NETLAY
             NEURON(I,J)=K
             NEUSEQ(K,1)=I
             NEUSEQ(K,2)=J
             K=K+1
300        CONTINUE
400    CONTINUE
      END IF
C
      IF(NETLAY .GT. MINSQR) THEN
        DO 600  I=(MINSQR+1), NETLAY
           DO 500 J=1, NETCOL
             NEURON(I,J)=K
             NEUSEQ(K,1)=I
             NEUSEQ(K,2)=J
             K=K+1
500        CONTINUE
600    CONTINUE
      END IF
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C COMPUTE NUMBER OF WTS+BIAS FOR A FINAL NETWORK
C  NFIG=ARRAY STORING NUMBER OF NODES IN EACH LAYER
C  NDIM=DIM. OF NFIG
C  NX=NUMBER OF INPUTS
```

```
C  MCX=1, SHORTCUT CONNECTIONS TO INPUTS ENABLED, OTHERWISE = 0
C  MCV=1, VERTICAL SHORTCUT CONNECTIONS TO PREVIOUS NODES, OTHERWISE =
0
CCCCCC
      FUNCTION NCONEX(NFIG,NDIM,NX,MCX,MCV)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      DIMENSION NFIG(NDIM)
C
        NCONEX=0
      DO 500 NL=1,NDIM
         NUMC=NFIG(NL)
         IF(NUMC .LE. 0) RETURN
         MCCX=MCX
         MCCC=1
         MCCV=0
         IF(NL .EQ. 1) THEN
           MCCX=1
           MCCC=0
             ELSE
               IF(NL .GT. 2)MCCV=MCV
         END IF
C
             DO 400 NC=1,NUMC
           NODEWS=NX*MCCX+NC*MCCC+(NL-2)*MCCV+1
           NCONEX=NCONEX+NODEWS
400      CONTINUE
C
500   CONTINUE
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C  SUBROUTINE FOR TRAINING THE NETWORK
CCCCCC
      SUBROUTINE TRAIN(X,H,Y,D,ER,HW,OW,ODW,OS,OPS,CW,DW,S,PS,
     &                 CC,CD,NEUS,NEUR,MXI,MXL,MXC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EPCWTS
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,MSRO,NEUO,METHO,NRETO
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,WTSCRO
      COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
      COMMON /NETTOP/ NETLAY,NETCOL
C
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE),Y(MXO,MXE),D(MXO,MXE)
      DIMENSION ER(MXO,1+MXE),HW(1+MXI+MXL+MXC,MXL,MXC),CD(MXD)
      DIMENSION OW(1+MXI+MXL*MXC,MXO),ODW(1+MXI+MXL*MXC,MXO)
      DIMENSION OS(1+MXI+MXL*MXC,MXO),OPS(1+MXI+MXL*MXC,MXO)
      DIMENSION CW(1+MXI+MXL+MXC,MXD),DW(1+MXI+MXL+MXC,MXD)
      DIMENSION S(1+MXI+MXL+MXC,MXD),PS(1+MXI+MXL+MXC,MXD)
      DIMENSION NEUS(MXL*MXC,2),NEUR(MXL,1+MXC),CC(MXO,MXD,2)
```

```
C
      NUMNEU=0
      NUMLAY=0
      NEPCO=0
      NEPOCH=0
      EPCWTS=0.0
      WTSCRO=0.0
      NCODE=0
      WSPAN=2.0*WRANGE
      NOWTS=1+MXI+MXL*MXC
      MAXNOD=NETLAY*NETCOL
C
      DO 140 JOU=1,NUMOUT
        DO 110 JCND=1,NUMCND
              CC(JOU,JCND,1)=0.0
          CC(JOU,JCND,2)=0.0
110     CONTINUE
C
        DO 120 JW=1,NOWTS
           OS(JW,JOU)=0.0
           OPS(JW,JOU)=0.0
           ODW(JW,JOU)=0.0
           OW(JW,JOU)=0.0
           IF(JW .LE. (1+NUMINP)) THEN
                 OW(JW,JOU)=WSPAN*RANDOM(NRSEED)
           END IF
120     CONTINUE
140   CONTINUE
CCCCCC
200   IF (NUMNEU .LT. MAXNOD) THEN
C
C OUTPUT TRAINING
        NCODE=MTROUT(X,H,Y,D,ER,OW,ODW,OS,OPS,NEUS,NEUR,
     &             MXI,MXL,MXC,MXO,MXE)
CCCCCC
            IF(NCODE .EQ. 1) THEN
          WRITE(*,*)'WIN!!!'
              GO TO 600
        END IF
        WRITE(*,700)'T_SQE:',TSQE,'V_SQE:',VSQE,'ErrVal:',ERRVAL
        WRITE(*,800)'H_Layers:',NUMLAY,'H_Nodes:',NUMNEU
            WRITE(*,*)
C
C INPUT TRAINING
        NCODE= MTRINP(X,H,HW,CC,CD,CW,DW,S,PS,ER,NEUS,NEUR,
     &              MXI,MXL,MXC,MXO,MXE,MXD)
CCCCCC
        CALL OWTNEW(OW,CC,MXI,MXL,MXC,MXO,MXD)
C
        GO TO 200
      END IF
C
        NCODE=MTROUT(X,H,Y,D,ER,OW,ODW,OS,OPS,NEUS,NEUR,
     &             MXI,MXL,MXC,MXO,MXE)
C
      WRITE(*,*)'OUT OF HIDDEN NODES !'
600   WRITE(*,700)'T_SQE:',TSQE,'V_SQE:',VSQE,'ErrVal:',ERRVAL
```

```
      WRITE(*,800)'H_Layers:',NUMLAY,'H_Nodes:',NUMNEU
      WRITE(*,*)
700   FORMAT(A,3X,F12.4,5X,A,F12.4,3X,A,F12.4)
800   FORMAT(A,I12,3X,A,I12)
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  MAIN FUNCTION FOR OUTPUT TRAINING
CCCCCC
      FUNCTION MTROUT(X,H,Y,D,ER,OW,ODW,OS,OPS,NEUS,NEUR,
     &                MXI,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,MSRO,NEUO,METHO,NRETO
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,EPWTSO
      COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
C
      DIMENSION ER(MXO,0:MXE),NEUR(MXL,1+MXC),NEUS(MXL*MXC,2)
      DIMENSION X(MXI,MXE),Y(MXO,MXE),D(MXO,MXE),H(MXL,MXC,MXE)
      DIMENSION OW(1+MXI+MXL*MXC,MXO),OS(1+MXI+MXL*MXC,MXO)
      DIMENSION ODW(1+MXI+MXL*MXC,MXO),OPS(1+MXI+MXL*MXC,MXO)
C
      MTROUT=3
      NEPC=0
      NFIRST=1
      PREERR=0.0
      NQUIT=MXEPCO
      MEWTS=0
      NWTS=1+NUMINP+NUMNEU
      DO 500 NEPC=1, MXEPCO
        CALL TRNOUT(X,H,Y,D,ER,OW,ODW,OS,OPS,NEUS,NEUR,
     &               MXI,MXL,MXC,MXO,MXE)
C
              NEPCO=NEPCO+1
              MEWTS=MEWTS+1
         IF(ERRVAL .LE. ERRTHR) THEN
               MTROUT=1
               GO TO 600
         ELSE
           IF(NFIRST .EQ. 1) THEN
             NFIRST=0
             PREERR=TSQE
           ELSE IF(ABS(TSQE-PREERR) .GT. (PREERR*THREO) )THEN
               NQUIT=NEPC+NBONO
               PREERR=TSQE
           ELSE
             IF(NQUIT .LT. NEPC) THEN
               MTROUT=2
               GO TO 600
             END IF
           END IF
         END IF
500   CONTINUE
```

```
600    EPWTSO=EPWTSO+DBLE(MEWTS*NWTS)/1000.0
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C VECTOR OUTPUT OF THE NET FOR THE K-TH EXMPLE
C AFTER ALL OUTPUTS OF HIDDEN UNITS ARE OBTAINED
C X=XINP
C H=HOUT(1,1,k)
C OW=OWTS
CCCCCC
       SUBROUTINE OUTPAS(KTH,X,H,Y,OW,NEUR,MXI,MXL,MXC,MXO,MXE)
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
       COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
       COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
       COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,MSRO,NEUO,METHO,NRETO
       COMMON /NETTOP/ NETLAY,NETCOL
C
       DIMENSION NEUR(MXL,0:MXC),X(MXI,MXE),Y(MXO,MXE),H(MXL,MXC)
       DIMENSION OW(0:(MXI+MXL*MXC),MXO)
C
       DO 600 JO=1,NUMOUT
           SUM=OW(0,JO)
         DO 100  JW=1,NUMINP
            SUM=SUM+X(JW,KTH)*OW(JW,JO)
100      CONTINUE
C
         IF(NUMNEU .GT. 0) THEN
           DO 500 M=1,NETLAY
              NCOL=NEUR(M,0)
              IF(NCOL .GT. 0) THEN
                DO 400 N=1,NCOL
                  JW=NUMINP+NEUR(M,N)
                  SUM=SUM+H(M,N)*OW(JW,JO)
400             CONTINUE
              END IF
500        CONTINUE
         END IF
550      Y(JO,KTH)=FTRANS(SUM,NEUO)
600    CONTINUE
C
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  COMPUTE VALUE OF STOPPING CONDITION
CCCCCC
       SUBROUTINE GETERV(Y,KX1,KX2,ERRV,NERR,MODSTD,MODTRN,
     &                                       MODV,MODR,MXO,MXE)
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
       COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
       COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
       COMMON /STATIS/ TSQE,SQER,VSQE,VEIX,VDSTD,ERRVAL,ERRTHR
       COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
       COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORMER,NEUO,NCENT,NRESO
```

```
C
      DIMENSION Y(MXO,MXE),ERRV(NERR),R(16)
C
      NV=0
      NB=6
      NR=16
      PERCNT=1.0
      COEV=1.0
      COVT=1.0
      VMSE=VSQE/(NVLDEX*NUMOUT)
      TMSE=TSQE/(NUMEXM*NUMOUT)
      IF(NCENT .EQ. 1)PERCNT=100.0
C
      ERRV(2)=SQRT(TMSE)
      ERRV(NB+2)=SQRT(VMSE)
      ERRV(5)=SQRT(TMSE)/VDSTD
      ERRV(NB+5)=SQRT(VMSE)/VDSTD
C
      IF(MODSTD .EQ. 1 )THEN
          CALL  GETSTD(Y,MXO,MXE,1,NUMOUT,KX1,KX2,MODV,MODR,R,NR,NV)
          VLDCOE=(R(NB+5)-R(NB+6))
          COEV=1.0+NORMER*(VLDCOE-1.0)
          ERRV(NB+3)=R(NB+1)
          ERRV(NB+4)=SQRT(ERRV(NB+3))
          ERRV(NB+6)=VMSE/R(NB+2)
C
          IF(MODTRN .EQ. 1) THEN
              TRNCOE=(R(5)-R(6))
              COET=1.0+NORMER*(TRNCOE-1.0)
              ERRV(3)=R(1)
              ERRV(4)=SQRT(ERRV(3))
              ERRV(6)=VMSE/R(NB+2)
          END IF
      END IF
C
      VMSE=VMSE*COEV
      ERRV(NB+1)=VMSE*PERCNT
      IF(MODTRN .EQ. 1) THEN
        TMSE=TMSE*COET
        ERRV(1)=TMSE*PERCNT
      END IF
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   OUTPUT TRAINING FOR ONE EPOCH
CCCCCC
      SUBROUTINE TRNOUT(X,H,Y,D,ER,OW,ODW,OS,OPS,NEUS,NEUR,
     &                  MXI,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORMER,NEUO,NCENT,NRESO
      COMMON /NCANDP/ MXEPC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /NVAMOD/ MODV,MODR
C
```

```
      DIMENSION ER(MXO,0:MXE),NEUR(MXL,1+MXC),NEUS(MXL*MXC,2),ERRV(16)
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE),Y(MXO,MXE),D(MXO,MXE)
      DIMENSION OW(0:(MXI+MXL*MXC),MXO),OS(0:(MXI+MXL*MXC),MXO)
      DIMENSION ODW(1+MXI+MXL*MXC,MXO),OPS(1+MXI+MXL*MXC,MXO)
C
      TSQE=0.0
      SQER = 0.0
      VSQE=0.0
      MODSTD = 1
      NB=6
      NERR=16
C
      IF((IDXMSR .EQ.5) .OR.(IDXMSR .EQ.2)) THEN
             MODSTD=0
      ELSE
           IF(IDXMSR .EQ. 1 )MODSTD=NORMER
      END IF
C
      DO 200 JOU=1,NUMOUT
         ER(JOU,0)= 0.0
         DO  100 JW=0,MXI+MXL*MXC
            OS(JW,JOU)=0.0
100      CONTINUE
200   CONTINUE
C
      DO 500 KTH=1,NUMEXM
        CALL OUTPAS(KTH,X,H(1,1,KTH),Y,OW,NEUR,MXI,MXL,MXC,MXO,MXE)
        CALL COMERR(KTH,X,H,Y,D,OS,ER,NEUS,MXI,MXL,MXC,MXO,MXE)
500   CONTINUE
C
      KX1=1
      KX2=NUMEXM
      CALL GETERV(Y,KX1,KX2,ERRV,NERR,MODSTD,0,MODV,MODR,MXO,MXE)
      ERRVAL=ERRV(NB+IDXMSR)
      IF(ERRVAL .GT. ERRTHR) THEN
         CALL  UPOWTS(OW,ODW,OS,OPS,MXI,MXL,MXC,MXO)
      END IF
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   COMPUTE ERRORS AT K-TH EXAMPLE
CCCCCCC
      SUBROUTINE COMERR(K,X,H,Y,D,OS,ER,NEUS,MXI,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,MSRO,NEUO,METHO,NRETO
      COMMON /NVAMOD/ MODV,MODR
C
      DIMENSION X(MXI,MXE),Y(MXO,MXE),D(MXO,MXE),H(MXL,MXC,MXE)
      DIMENSION OS(0:(MXI+MXL*MXC),MXO),NEUS(MXL*MXC,2),ER(MXO,0:MXE)
CCCCCC
```

```
        MARKV=MARKOP(K,MODV,MODR)
        SQE=0.0
C
        DO 400 JOU=1,NUMOUT
          DIF = Y(JOU,K) - D(JOU,K)
          ER(JOU,K)=DIF
          EFP= DIF*OPRIME(Y(JOU,K),NEUO)
          ER(JOU,0)= ER(JOU,0)+EFP
              SQE=SQE+(DIF*DIF)
          SQER = SQER+(EFP*EFP)
          OS(0,JOU)=OS(0,JOU)+EFP
C
          DO 200 JW=1,NUMINP
            VAL=(X(JW,K)*EFP)
            OS(JW,JOU)= OS(JW,JOU)+VAL
200       CONTINUE
C
          IF (NUMNEU .GT. 0) THEN
           DO 300 NNODES=1,NUMNEU
              JW=NUMINP+NNODES
              IROW=NEUS(NNODES,1)
              JCOL=NEUS(NNODES,2)
              OS(JW,JOU)= OS(JW,JOU)+H(IROW,JCOL,K)*EFP
300        CONTINUE
          END IF
400     CONTINUE
C
        TSQE =TSQE+SQE
        IF(MARKV .EQ. 1)VSQE=VSQE+SQE
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  UPDATING WEIGHTS OF OUTPUT UNITS
CCCCCC
      SUBROUTINE UPOWTS(OW,ODW,OS,OPS,MXI,MXL,MXC,MXO)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /PTROUT/ A,B,GAMMO,SF,THREO,DECAYO,WTSCRO
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
C
      DIMENSION OW(1+MXI+MXL*MXC,MXO),OS(1+MXI+MXL*MXC,MXO)
      DIMENSION ODW(1+MXI+MXL*MXC,MXO),OPS(1+MXI+MXL*MXC,MXO)
C
      MAXWTS=1+MXI+MXL*MXC
      NWTS=1+NUMINP+NUMNEU
      EPS=A/NUMEXM
      SF=B/(1.0+B)
      DO 400  JOU=1,NUMOUT
          DO  300  JTHW=1,NWTS
C UPDATE WTS USING QKPROP
           CALL QKPROP(JTHW,MAXWTS,OW(1,JOU),ODW(1,JOU),
     &                 OPS(1,JOU),OS(1,JOU),EPS,B,SF,DECAYO)
300       CONTINUE
400    CONTINUE
C
```

```
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  MAIN FUNCTION FOR INPUT TRAINING
CCCCCC
      FUNCTION MTRINP(X,H,HW,CC,CD,CW,DW,S,PS,ER,NEUS,NEUR,
     &                 MXI,MXL,MXC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /PTRAIN/ A,B,GAMMA,SF,BSTSCR,THRESH,EPCWTS
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /WTSVAR/ WRANGE,BSTERR,TRMSE,TRDEV,TRSTD,VLDCOE,TRNCOE
C
      DIMENSION ER(MXO,0:MXE),NEUS(MXL*MXC,2),NEUR(MXL,0:MXC)
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE),CD(MXD),CC(MXO,MXD,2)
      DIMENSION CW(1+MXI+MXL+MXC,MXD),DW(1+MXI+MXL+MXC,MXD)
      DIMENSION S(1+MXI+MXL+MXC,MXD),PS(1+MXI+MXL+MXC,MXD)
      DIMENSION HW(1+MXI+MXL+MXC,MXL,MXC)
C
      MTRINP=3
      PSCORE=0.0
      NQUIT= MXEPOC
      NFIRST= 1
      NBX=0
      NBC=0
      NBV=0
      NW=1+MXI+MXL+MXC
      IROW=NEUS(NUMNEU+1,1)
      JCOL=NEUS(NUMNEU+1,2)
      NWTS=NUMHWT(IROW,JCOL,NUMINP,NBX,NBC,NBV)+1
      WSCALE=2.0*WRANGE
C
      DO 160 JCND=1, NUMCND
           CD(JCND)=0.0
        DO 140 JW=1,NW
         CW(JW,JCND )=WSCALE*RANDOM(NRSEED)
         IF(JW .GT. NWTS)CW(JW,JCND )=0.0
         DW(JW,JCND)=0.0
         S(JW,JCND)=0.0
         PS(JW,JCND)=0.0
140     CONTINUE
C
        DO 150 JOU=1,NUMOUT
              CC(JOU,JCND,1)=0.0
              CC(JOU,JCND,2)=0.0
150     CONTINUE
160   CONTINUE
C
      DO  170 JOU=1,NUMOUT
         ER(JOU,0)=ER(JOU,0)/NUMEXM
170   CONTINUE
C
      CALL COREPC(X,H,CD,CC,CW,ER,NEUS,MXI,MXL,MXC,MXO,MXE,MXD)
C
```

```
       NODEPC=0
       DO 400 NEPC=1,MXEPOC
          CALL CNDSLP(X,H,CC,CD,CW,S,ER,NEUS,
     &                 MXI,MXL,MXC,MXO,MXE,MXD)
C
          CALL UPHWTS(X,H,CC,CD,CW,DW,S,PS,ER,NEUS,NWTS,
     &                 MXI,MXL,MXC,MXO,MXE,MXD)
C
          CALL ADJCOR(ER,CD,CC,MXO,MXE,MXD)
C
          NODEPC=NODEPC+1
          NEPOCH=NEPOCH+1
          IF(NFIRST .EQ. 1) THEN
            NFIRST=0
            PSCORE= BSTSCR
          ELSE IF(ABS(BSTSCR- PSCORE) .GT. (PSCORE*THRESH)) THEN
            PSCORE= BSTSCR
            NQUIT= NEPC + NBONUS
          ELSE
            IF(NQUIT .LT. NEPC ) THEN
              DO 200 JW=1,NWTS
                   HW(JW,IROW,JCOL)=CW(JW,NBESTC)
200           CONTINUE
C
              DO 300 K=1,NUMEXM
                  VAL=OUTHNU(X(1,K),NUMINP,H(1,1,K),CW(1,NBESTC),
     &                         IROW,JCOL,MXI,MXL,MXC)
C
                H(IROW,JCOL,K)=VAL
300           CONTINUE
              MTRINP=2
              GO TO 500
            END IF
          END IF
C
400    CONTINUE
C
500    NEUR(IROW,0)=NEUR(IROW,0)+1
       NUMLAY=MAX(NUMLAY,IROW)
         EPCWTS=EPCWTS+DBLE(NODEPC*NWTS)/1000.0D0
C
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  OBTAIN INITIAL VALUES OF NEW WEIGHTS OF OUTPUT UNITS
C  USING COVARIANCE VALUES
CCCCCC
       SUBROUTINE OWTNEW(OW,CC,MXI,MXL,MXC,MXO,MXD)
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
       COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
       COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
C
       DIMENSION CC(MXO,MXD,2),OW(0:(MXI+MXL*MXC),MXO)
C
       WM=1.0/(1+NUMINP+NUMNEU)
```

```
      NUMNEU=NUMNEU+1
      JW=NUMINP+NUMNEU
      DO 300 NOUT=1,NUMOUT
        OW(JW,NOUT) = CC(NOUT,NBESTC,1)*WM
300   CONTINUE
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   GET NUMBER OF WEIGHTS (NOT INCLUDING BIAS) FOR NEURON(IROW,JCOL)
C   NBX,NBC,NBV: BASE VALUES FOR INDEXING WEIGHTS W.R.P TO ORIGINAL
C   INPUTS, HIDDEN NODES IN THE (IROW-1) LAYER AND IN THE JCOL-TH
COLUMN
CCCCCC
      FUNCTION NUMHWT(IROW,JCOL,NUMINP,NBX,NBC,NBV)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /MODNEX/ NOX, NOV
C
      NW=NUMINP*NOX+JCOL
      NBX=0
      NBC=NUMINP*NOX
      NBV=-1
      IF(IROW .GT. 2) THEN
        NW=NW+(IROW-2)*NOV
        IF(NOX .EQ. 0) NBX=-1
        IF(NOV .EQ. 0)NBV=-1
          IF(NOV .EQ. 1)NBV=NUMINP*NOX+JCOL
      ELSE IF(IROW .EQ. 2) THEN
        IF(NOX .EQ. 0)NBX=-1
      ELSE
        NW=NUMINP
        NBC=-1
      END IF
      NUMHWT=NW
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    OUPUT OF THE NEURON(IROW,JCOL) FOR THE K-TH EXAMPLE
C    X=XINP(1,K)
C    NIN=NUMINP
C    H=HOUT(1,1,K)
C    HW=HWTS(1,IROW,JCOL)/CWTS(1,ICAND)
CCCCCC
      FUNCTION OUTHNU(X,NIN,H,HW,IROW,JCOL,MXI,MXL,MXC)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
C
      DIMENSION X(MXI),H(MXL,MXC),HW(0:(MXI+MXL+MXC))
C
      NBC=0
      NBX=0
      NBV=0
      NWTS=NUMHWT(IROW,JCOL,NIN,NBX,NBC,NBV)
```

```
          S=HW(0)
C  SUM OF WEIGHTED INPUTS
      IF(NBX .GE. 0) THEN
        DO 100  I=1,NIN
          S=S+X(I)*HW(NBX+I)
100     CONTINUE
      END IF
C
C  SUM OF WEIGHTED OUTPUTS OF NODES IN THE PREVIOUS LAYER
       IF(NBC .GE. 0) THEN
          DO 300  J=1,JCOL
            S=S+H(IROW-1,J)*HW(NBC+J)   `
300       CONTINUE
       END IF
C
C  SUM OF WEIGHTED OUTPUTS OF NODES IN THE JCOL COLUMN
      IF(NBV .GE. 0) THEN
         DO 200  I=1,IROW-2
            S=S+H(I,JCOL)*HW(NBV+I)
200      CONTINUE
      END IF
C
      OUTHNU=FTRANS(S,NTRANS)
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C COVARIANCE VALUES OF CANDIDATES AFTER ONE EPOCH OF INPUT TRAINING
CCCCCC
      SUBROUTINE COREPC(X,H,CD,CC,CW,ER,NEUS,MXI,MXL,MXC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
C
      DIMENSION ER(MXO,0:MXE),NEUS(MXL*MXC,2)
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE)
      DIMENSION CD(MXD),CC(MXO,MXD,2),CW(1+MXI+MXL+MXC,MXD)
C
      IR=NEUS(NUMNEU+1,1)
      JC=NEUS(NUMNEU+1,2)
      DO 300 K=1,NUMEXM
C
        DO 200 JCND=1,NUMCND
C
         VAL=OUTHNU(X(1,K),NUMINP,H(1,1,K),CW(1,JCND),
     &                              IR,JC,MXI,MXL,MXC)
C
C SUM OF CANDIDATE'S OUTPUT, USED IN COMPUTING COVARINCE VALUE
         CD(JCND) = CD(JCND)+VAL
C
          DO 180 JOU=1,NUMOUT
           CC(JOU,JCND,2)= CC(JOU,JCND,2)+VAL*ER(JOU,K)
180       CONTINUE
C
200    CONTINUE
C
```

```
300   CONTINUE
C
      CALL ADJCOR(ER,CD,CC,MXO,MXE,MXD)
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  ADJUST COVARINCE VELUES OF CANDIDTATES
C   CD=CDOU(1,0)
C   ER=ERRO(1,0)
CCCCCC
      SUBROUTINE ADJCOR(ER,CD,CC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EPCWTS
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /STATIS/ SSQE,SQER,SUMY,SMYY,VSTD,ERRVAL,ERRTHR
C
      DIMENSION CC(MXO,MXD,2),CD(MXD),ER(MXO,0:MXE)
CCCCCC
      NBESTC=0
      BSTSCR= 0.0
      DO 200 JCND=1,NUMCND
        CBAR = CD(JCND)/NUMEXM
        COR = 0.0
        SCORE= 0.0
C
        DO 100 JOU=1,NUMOUT
C NOMALIZE COVARIANCE VALUES
          COR = (CC(JOU,JCND,2) - ER(JOU,0)*CBAR)/SQER
          CC(JOU,JCND,1) = COR
          CC(JOU,JCND,2) = 0.0
          SCORE=SCORE+ABS(COR)
100     CONTINUE

        CD(JCND)= 0.0
        IF(SCORE .GT. BSTSCR) THEN
           BSTSCR = SCORE
           NBESTC=JCND
        END IF
200   CONTINUE
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  UPDATE WEIGHTS OF HIDDEN NODES
CCCCCC
      SUBROUTINE UPHWTS(X,H,CC,CD,CW,DW,S,PS,ER,NEUS,NWTS,
     &                  MXI,MXL,MXC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /PTRAIN/ A,B,GAMMA,SF,BSTSCR,THRESH,EPCWTS
```

```
C
      DIMENSION ER(MXO,0:MXE),NEUS(MXL*MXC,2)
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE),CC(MXO,MXD,2),CD(MXD)
      DIMENSION CW(1+MXI+MXL+MXC,MXD),DW(1+MXI+MXL+MXC,MXD)
      DIMENSION S(1+MXI+MXL+MXC,MXD),PS(1+MXI+MXL+MXC,MXD)
C
      NW=1+MXI+MXL+MXC
C
      EPS=A/(NUMEXM*NWTS)
      DEC=0.0
      SF=B/(1.0+B)
      DO 500  JCND=1,NUMCND
          DO  400  JW=1,NWTS
C
          CALL QKPROP(JW,NW,CW(1,JCND),DW(1,JCND),
     &              PS(1,JCND),S(1,JCND),EPS,B,SF,DEC)
C
400       CONTINUE
500   CONTINUE
C
          END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C COMPUTE PARTIAL DERIVATIVES OF COVARIANCE W.R.T TO WEIGHTS
C FOR EACH CANDIDATE
CCCCCC
      SUBROUTINE CNDSLP(X,H,CC,CD,CW,S,ER,NEUS,MXI,MXL,MXC,MXO,MXE,MXD)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /STATIS/ SSQE,SQER,SUMY,SMYY,VSTD,ERRVAL,ERRTHR
C
      DIMENSION ER(MXO,0:MXE),NEUS(MXL*MXC,2),CD(MXD)
      DIMENSION X(MXI,MXE),H(MXL,MXC,MXE),CC(MXO,MXD,2)
      DIMENSION CW(1+MXI+MXL+MXC,MXD),S(0:(MXI+MXL+MXC),MXD)
C
      NBC=0
      NBX=0
      NBV=0
      IROW=NEUS(1+NUMNEU,1)
      JCOL=NEUS(1+NUMNEU,2)
      NWTS=NUMHWT(IROW,JCOL,NUMINP,NBX,NBC,NBV)
C
      DO 400 K=1, NUMEXM
         DO 300 JCND=1,NUMCND
           CHANGE= 0.0
           DELTA=0.0
           VAL=OUTHNU(X(1,K),NUMINP,H(1,1,K),CW(1,JCND),
     &                          IROW,JCOL,MXI,MXL,MXC)
C
           CD(JCND) = CD(JCND)+VAL
           FP=FPRIME(VAL,NTRANS)
C
           DO 100 JOU=1,NUMOUT
             DIR=1.0
```

```
              IF(CC(JOU,JCND,1) .LT. 0.0)DIR=-1.0
C NORMALIZED  DELTA: DELTA=-DIR*FP*(ER(JOU,K) -ER(JOU,0))/SQER
              DELTA=-DIR*FP*(ER(JOU,K) -ER(JOU,0))
              CHANGE=CHANGE+DELTA
              CC(JOU,JCND,2)=CC(JOU,JCND,2)+VAL*ER(JOU,K)
100         CONTINUE
C
C  W.R.P TO THE BIAS NODE
             S(0,JCND)= S(0,JCND)+CHANGE
C
C  W.R.P TO THE NODES IN THE PREVIOUS LAYER
            IF(NBC .GE. 0) THEN
               DO 200  JW=1,JCOL
                 S(JW+NBC,JCND)=S(JW+NBC,JCND)+CHANGE*H(IROW-1,JW,K)
200            CONTINUE
            END IF
C
C  W.R.P TO INPUTS
            IF(NBX .GE. 0) THEN
              DO 220 JW=1,NUMINP
                S(NBX+JW,JCND)=S(NBX+JW,JCND)+CHANGE*X(JW,K)
220           CONTINUE
            END IF
C
C  W.R.P TO THE NODES IN THE SAME COLUMN
            IF(NBV .GT. 0) THEN
               DO 240  JW=1, IROW-2
                 S(NBV+JW,JCND)=S(NBV+JW,JCND)+CHANGE*H(JW,JCOL,K)
240            CONTINUE
            END IF
C
300       CONTINUE
400     CONTINUE
C
         END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C TEST TRAINED NET ON TEST SET
CCCCCC
      SUBROUTINE TEST(X,D,Y,OW,H,HW,NEUR,TSER,NS,NZ,JS,
     &                MXI,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /STATIS/ SSQE,SQER,SUMY,SMYY,VSTD,ERRVAL,ERRTHR
C
      DIMENSION NEUR(MXL,1+MXC),X(MXI,MXE),Y(MXO,MXE),R(16)
      DIMENSION D(MXO,MXE),H(MXL,MXC),OW(1+MXI+MXL*MXC,MXO)
      DIMENSION HW(1+MXI+MXL+MXC,MXL,MXC),TSER(NS,NZ),ERRV(16)
CCCCCC
      SSQE=0.0
C COMPUTE OUTPUTS OF NEURON FOR THE K-TH EXAMPLE
      DO 200 K=NUMEXM+1, NUMEXM+NTSTEX
        CALL HNUPAS(X(1,K),H,HW,NEUR,MXI,MXL,MXC)
```

```
        CALL OUTPAS(K,X,H,Y,OW,NEUR,MXI,MXL,MXC,MXO,MXE)
C
        DO 100 I=1,NUMOUT
          DXY=Y(I,K)-D(I,K)
          SSQE=SSQE+(DXY*DXY)
100     CONTINUE
C
200   CONTINUE
C
      KX1=NUMEXM+1
      KX2=NUMEXM+NTSTEX
      NEXM=NUMEXM
      STDD=TDSTD
      NUMEXM=NTSTEX
      TDSTD=TSDSTD
      NR=16
      CALL  GETERV(Y,KX1,KX2,ERRV,NR,1,1,1,0,MXO,MXE)
      DO 300 K=1,6
        TSER(JS,K)=ERRV(K)
300   CONTINUE
C
      NUMEXM=NEXM
      TDSTD=STDD
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  INITIALIZE PARAMETERS OF THE NET
CCCCCC
      SUBROUTINE ININET
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NCANDP/ MXEPOC,NBONUS,IDXMSR,NUMCOD,METHOD,NTRANS,NRSEED
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,DTOLER
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /STATIS/ SSQE,STDD,VMSE,VDEV,VSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TRMSE,TRDEV,TRSTD,VLDCOE,TRNCOE
C
C  /NUMVAR/
      NUMINP=0
      NUMOUT=0
      NUMEXM=0
      NTSTEX=0
      NRUNEX=0
      NTRIAL=0
C /NCANDP/
      MXEPOC=100
      NBONUS=8
      IDXMSR=1
CCCCCC   NETSTA=0, TRAIN, 1,RUN,
      NETSTA=0
      METHOD=0
      NTRANS=2
      NRSEED=13
C /PTRAIN/
      ALPHA=0.75
      BETA=1.75
```

```
      GAMMA=0.95
      SHRINK=BETA/(1.0+BETA)
      BSTSCR=0.0
      THRESH=0.03
      DTOLER=0.0
C
C /NTRAIN/
      NRETUR=0
      NUMCND=8
      NEPOCH=0
      NBESTC=0
      NUMLAY=0
      NUMNEU=0
C /STATIS/
      SSQE=0.0
      STDD=0.0
      VMSE=0.0
      VDEV=0.0
      ERRVAL=0.0
      VSTD=0.0
      ERRTHR=0.0
C /WTSVAR/ WRANGE,BSTFVU
      WRANGE=0.5
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C WRITE HEADER OF CONFIGRATION FILE
CCCCCC
      SUBROUTINE WTHEAD(NFILE,TRNFNM)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NCANDP/ MXEP,NBON,IDXMSR,NETSTA,METHOD,NUTYPH,NRSEED
      COMMON /STATIS/ SSQE,STDD,VMSE,VDEV,VSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TRMSE,TRDEV,TRSTD,VLDCOE,TRNCOE
      COMMON /NTRAIN/ NRETUR,NCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EPCWTH
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORM,NUTYPO,NCENT,NRESO
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,EPCWTO
      COMMON /NVAMOD/ MODV, MODR
      COMMON /MODNEX/ NOX, NOV
      COMMON /NETTOP/ NLAY,NCOL
      CHARACTER*30 TRNFNM,CH*80
C
      NETSTA=1
      WRITE(NFILE,'(A)')TRNFNM
      CH='REM  NETSTA, NETLAY, NETCOL, HNUTYP, ONUTYP, ISHORT, HSHORT'
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,900)NETSTA,NLAY,NCOL,NUTYPH,NUTYPO,NOX,NOV
CCCCCC
      CH='REM  HMXEPC, HBONUS, HCANDS, OMXEPC, OBONUS'
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,900)MXEP,NBON,NCND,MXEPCO,NBONO
      CH='REM  HLRNRT, HMXLRN, HTHRES, WRANGE'
      WRITE(NFILE,'(A)')CH
```

```
      WRITE(NFILE,850)ALPHA,BETA,THRESH,WRANGE
      CH='REM  OLRNRT, OMXLRN, OTHRES, ODECAY, ETOLER'
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,850)ALPHO,BETO,THREO,DECAYO*100.0,ERRTHR
      CH='REM  NUMLAY, NUMNEU, ERNORM, PERCNT , ERRMSR'
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,900)NUMLAY,NUMNEU,NORM,NCENT,IDXMSR
      CH='REM  INPUTS, OUTPTS, VLDMOD,  VLDVAL, NSEEDS'
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,900)NUMINP,NUMOUT,MODV,MODR,NTRIAL
C
850   FORMAT(4X,10(F7.4,1X))
900   FORMAT(4X,10(I7,1X))
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  UPDATE NET CONFIGRATION FILE
C  --SETUP PARAMETERS
C  --RANDOM SEEDS FOR TRAING THE NET
C  --SAVE WEIGHTS OF THE BEST-RUN NETWORK
CCCCCC
      SUBROUTINE WRTNET(HW,OW,NSEED,NS,NEUR,NETFNM,TRNFNM,
     &                  MXI,MXL,MXC,MXO)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NCANDP/ MXEP,NBON,IDXMSR,NETSTA,METHOD,NTRANS,NRSEED
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NTRAIN/ NRETUR,NCND,NEPOCH,NBESTC,NUMLAY,NUMNEU
      COMMON /NETTOP/ NETLAY,NETCOL
C
      DIMENSION NEUR(MXL,0:MXC),HW(1+MXI+MXL+MXC,MXL,MXC)
      DIMENSION OW(1+MXI+MXL*MXC,MXO),NSEED(0:NS)
      CHARACTER*30 NETFNM,TRNFNM,CH*80
C
      NBC=0
      NBX=0
      NBV=0
      NFILE=30
      DERR=0.0
      NETSTA=1
      OPEN(UNIT=NFILE,FILE=NETFNM,STATUS='OLD')
      CALL WTHEAD(NFILE,TRNFNM)
      CH='REM  RANDOM SEEDS'
      WRITE(NFILE,'(A)') CH
C
      DO 120 NT=1,NTRIAL
         WRITE(NFILE,*) NSEED(NT)
120   CONTINUE
C
      CH='REM  BEST SEED No. AND SEED VALUE'
      NRSEED=NSEED(0)
      WRITE(NFILE,'(A)')CH
      WRITE(NFILE,'(2I10)') NRSEED,NSEED(NRSEED)
C
      IF(NUMLAY .GT. 0) THEN
```

```
          CH='REM  NUMBERS OF HIDDEN NODES'
          WRITE(NFILE,'(A)') CH
C
          IF (NETCOL .EQ. 1) THEN
             WRITE(NFILE,*) NUMNEU
          ELSE
             DO 100 NL=1,NUMLAY
                WRITE(NFILE,*) NEUR(NL,0)
100          CONTINUE
          END IF
C
            CH='REM  WEIGHTS OF HIDDEN NODES'
          WRITE(NFILE,'(A)') CH
          DO 300 NL=1, NUMLAY
             DO 200 NC=1,NEUR(NL,0)
               NWTS=NUMHWT(NL,NC,NUMINP,NBX,NBC,NBV)+1
               WRITE(NFILE,800) NL,NC
               DO 160 NW=1,NWTS
                  WRITE(NFILE,700) HW(NW,NL,NC)
160            CONTINUE
200          CONTINUE
300       CONTINUE
CCCCCC
       END IF
C
       CH='REM  WEIGHTS OF OUTPUT NODES'
       WRITE(NFILE,'(A)') CH
       NWTS=1+NUMINP+NUMNEU
C
       DO 500 NOUT=1,NUMOUT
          WRITE(NFILE,800)NOUT
          DO 400 NW=1,NWTS
             WRITE(NFILE,700) OW(NW,NOUT)
400       CONTINUE
500    CONTINUE
C
       CLOSE(NFILE)
700    FORMAT(1X,D24.15)
800    FORMAT(I5,2X,I5)
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  OPERATION CODE SELECTING VALIDATION SET FROM TRAINING DATA
CCCCCC
       FUNCTION MARKOP(K,MODV,MODR)
C
       IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
       IF((MODV .GT. 1) .AND. (MODR .GE. MODV))MODR=0
       MARKOP=0
       IF(MODV .EQ. 1) THEN
          IF(K .GT. MODR) MARKOP=1
       ELSE
          IF(MOD(K,MODV) .EQ. MODR) MARKOP=1
       END IF
C
       END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C COMPUTE VARIANCE AND STANDARD DEVIATION FOR A GIVEN SET OF EXAMPLE
C R(I): I=1, VARIANCE
C         2, STANDARD DEVIATION
C         3, SUM(X^2)
C         4, SUM(X)
C         5, MAX(X)
C         6, MIN(X)
C    FOR ALL TRAINING EXAMPLES (I=1~6) AND VALIDATION SET (I=7~12)
C NT=COUNT OF TOTAL SAMPLES
C NV=COUNT OF VALIDATION SAMPLES
CCCCCC
      SUBROUTINE GETSTD(X,MDY,MDX,NY1,NY2,NX1,NX2,MDV,MDR,R,NR,NV)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      DIMENSION X(MDY,MDX),R(NR)
C
      NBV=6
      DO 100 I=1,NR
          R(I)=0.0
100   CONTINUE
      TMX=-1.0D30
      TMN=-TMX
      VMX=TMX
      VMN=TMN
      NV=0
      NT=0
      NY=NY2-NY1+1
      TCOE=0.0
      VCOE=0.0
C
      DO 500 K=NX1,NX2
         NCODE=MARKOP(K,MDV,MDR)
         SMX=0.0
         SQX=0.0
         DO 400 J=NY1,NY2
            SQX=SQX+X(J,K)*X(J,K)
            SMX= SMX+X(J,K)
            TMX=MAX(TMX,X(J,K))
            TMN=MIN(TMN,X(J,K))
C
            IF(NCODE .EQ. 1) THEN
              VMX=MAX(VMX,X(J,K))
              VMN=MIN(VMN,X(J,K))
            END IF
400      CONTINUE
C
         NT=NT+1
         TCOE=DBLE(NT-1)/DBLE(NT)
         R(3)=R(3)*TCOE+SQX/NT
         R(4)=R(4)*TCOE+SMX/NT
         IF(NCODE .EQ. 1) THEN
            NV=NV+1
            VCOE=DBLE(NV-1)/DBLE(NV)
            R(NBV+3)=R(NBV+3)*VCOE+SQX/NV
```

```
            R(NBV+4)=R(NBV+4)*VCOE+SMX/NV
          END IF
C
500     CONTINUE
C
        IF(NT .GT. 1) THEN
          TCOE=DBLE(NT)/DBLE(NY*NT-1)
          R(1)=(R(3)-R(4)*(R(4)/NY))*TCOE
          R(2)=SQRT(R(1))
          R(5)=TMX
          R(6)=TMN
        END IF
C
        IF(NV .GT. 1) THEN
          VCOE=DBLE(NV)/DBLE(NY*NV-1)
          R(NBV+1)=(R(NBV+3)-R(NBV+4)*(R(NBV+4)/NY))*VCOE
          R(NBV+2)=SQRT(R(NBV+1))
          R(NBV+5)=VMX
          R(NBV+6)=VMN
        END IF
C
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   READ TRAINING DATA FROM INPUT FILE
CCCCCC
        SUBROUTINE RDPROB(X,D,FNM,MXI,MXO,MXE)
C
        IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
        COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
        COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
        COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
        COMMON /NVAMOD/ MODV, MODR
C
        DIMENSION X(MXI,MXE),D(MXO,MXE),TEMP(MXI+MXO),R(16)
        CHARACTER*30 FNM, CH*256
CCCCCC
        NFILE=31
        NUM=0
        NFLAG=0
        NLEN=0
        NUMEXM=0
        NTSTEX=0
        NVLDEX=0
        NTRNEX=0
C
        OPEN(UNIT=NFILE,FILE=FNM,STATUS='OLD')
C
        READ(NFILE,*)CH, NUMINP
        READ(NFILE,*)CH, NUM
        NUMINP=NUMINP+NUM
        READ(NFILE,*)CH, NUMOUT
        READ(NFILE,*)CH, NUM
        NUMOUT=NUMOUT+NUM
        READ(NFILE,*)CH, NTRNEX
        READ(NFILE,*)CH, NVLDEX
```

```
      READ(NFILE,*)CH, NTSTEX
      NUMEXM=NTRNEX+NVLDEX
      NUM=NUMEXM+NTSTEX
C
      IF(MODV .EQ. 1) THEN
          MODR=NTRNEX
          IF(MODR .GE. NUMEXM)  THEN
            MODR=0
            NVLDEX=NUMEXM
          END IF
      ELSE
        IF((MODV .GT. 4) .OR. (MODV .LT. 2) .OR.(MODR .LT. 0)
     &           .OR. (MODR .GT. MODV) ) THEN
           WRITE(*,*) 'Invalid value for VLDMOD or VLDVAL;'
           WRITE(*,*) 'Default VLDMOD=3, VLDVAL=2  assumed.'
         MODV=3
         MODR=2
        END IF
      END IF
C
      IF((NUM .GT. MXE) .OR. (NUMINP .GT. MXI)
     &                      .OR.(NUMOUT .GT. MXO)) THEN
          WRITE(*,*) 'Storage not enough for training data;'
        NFLAG=1
      END IF
C
      IF(NFLAG .GT. 0) THEN
         CLOSE(NFILE)
         STOP 'Reading data terminated!'
      END IF
        NLEN=NUMINP+NUMOUT
CCCCCC
      DO 500 K=1,NUM
        READ(NFILE,*)(TEMP(J),J=1,NLEN)
C
        DO 100 NIN=1,NUMINP
            X(NIN,K)=TEMP(NIN)
100     CONTINUE
C
        DO 200 NOUT=1,NUMOUT
           D(NOUT,K)=TEMP(NUMINP+NOUT)
200     CONTINUE
C
500   CONTINUE
C
      CLOSE(NFILE)
C
      NR=16
      NB=6
      CALL GETSTD(D,MXO,MXE,1,NUMOUT,1,NUMEXM,MODV,MODR,R,NR,NV)
      NVLDEX=NV
      TDSTD=R(2)
      VDSTD=R(NB+2)
C
      NX1=NUMEXM+1
      NX2=NUMEXM+NTSTEX
      CALL GETSTD(D,MXO,MXE,1,NUMOUT,NX1,NX2,1,NX2+1,R,NR,NV)
```

```
      TSDSTD=R(2)
700   FORMAT(40(1X,D24.15))
      END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  RUN TRAINED NET ON NEW INPUT DATA
C X=XINP
C Y=YOUT
CCCCCC
      SUBROUTINE RUNNET(X,Y,H,HW,OW,NEUR,FNMIN,FNMOUT,
     &                  MXI,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
C
      DIMENSION NEUR(MXL,1+MXC),X(MXI,MXE),Y(MXO,MXE),H(MXL,MXC)
      DIMENSION OW(1+MXI+MXL*MXC,MXO),HW(1+MXI+MXL+MXC,MXL,MXC)
      CHARACTER*30 FNMIN, FNMOUT
CCCCCC
      NFX=32
      NFY=33
      NINP=0
      OPEN(UNIT=NFX,FILE=FNMIN,STATUS='OLD')
      READ(NFX,*)NINP,NRUNEX
C
      IF(NINP .NE. NUMINP) THEN
         WRITE(*,*)'Mismatched number of inputs!'
         CLOSE(NFX)
         STOP
      END IF
C
C  FORWARD PASS: GET OUTPUTS OF THE HIDDEN AND OUTPUT UNITS
C
      OPEN(UNIT=NFY,FILE=FNMOUT,STATUS='OLD')
      WRITE(NFY,*)NINP,'    ', NRUNEX
C
      NX=1
      DO 100 K=1,NRUNEX
        READ(NFX,*)(X(I,NX),I=1,NUMINP)
        CALL HNUPAS(X(1,NX),H,HW,NEUR,MXI,MXL,MXC)
        CALL OUTPAS(NX,X,H,Y,OW,NEUR,MXI,MXL,MXC,MXO,MXE)
        WRITE(NFY,700)(Y(J,NX),J=1,NUMOUT)
100   CONTINUE
C
C
      CLOSE(NFX)
      CLOSE(NFY)
      WRITE(*,*)'Ouputs stored in file  '//FNMOUT
700   FORMAT(40(1X,D24.15))
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C COMPUTE OUTPUTS OF THE HIDDEN UNITS FOR SINGLE INPUT VECTOR
CCCCCC
      SUBROUTINE HNUPAS(TX,TH,HW,NEUR,MXI,MXL,MXC)
```

```
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NETTOP/ NETLAY,NETCOL
C
      DIMENSION TX(MXI),TH(MXL,MXC)
      DIMENSION NEUR(MXL,0:MXC),HW(1+MXI+MXL+MXC,MXL,MXC)
CCCCCC
      DO 120 I=1,NETLAY
         NCOL=NEUR(I,0)
        IF(NCOL .GT. 0) THEN
           DO 110 J=1,NCOL
             TH(I,J)=OUTHNU(TX,NUMINP,TH,HW(1,I,J),I,J,MXI,MXL,MXC)
110        CONTINUE
        ELSE
           RETURN
        END IF
C
120   CONTINUE
C
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C SHOW STATISTICAL RESULTS OF TRAINING
CCCCCC
      SUBROUTINE SHOWER(NTS,TR,VR,SR,NS,NZ,JS,NF)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NTRNOU/ MXEPCO,NEPCO,NBONO,NORMER,NEUO,NCENT,NRESO
C
      DIMENSION NTS(NS,NZ),TR(NS,NZ),VR(NS,NZ),SR(NS,NZ)
      CHARACTER*5 CHU(6),CHT(6),CHV(6)
      CHARACTER CH1*17,CH2*17,CH3*17,C1*18,C2*11
C
      CHT(1)='MSE: '
      CHT(2)='VAR: '
      CHT(3)='IEW: '
      CHT(4)='OEW: '
      CHT(5)='EIX: '
      CHT(6)='FVU: '
C
      CHV(1)='MSE: '
      CHV(2)='RMS: '
      CHV(3)='VAR: '
      CHV(4)='STD: '
      CHV(5)='EIX: '
      CHV(6)='FVU: '
C
      CHU(1)='HNUS:'
      CHU(2)='HWTS:'
      CHU(3)='TWTS:'
      CHU(4)='IEPC:'
      CHU(5)='OEPC:'
      CHU(6)='TEPC:'
C
      C1=' actual error'
```

```
        C2=''
       IF(NORMER .EQ. 1)C1=' normalized error '
        IF(NCENT .EQ. 1) C2=' percentage'
       CH1='On Training  Data'
        CH2='On Validation Set'
        CH3='      On Test Set'
C
       IF(NF .EQ. 0) THEN
        WRITE(*,*)'TRIAL No:',JS
         WRITE(*,*)'SQUARED ERROR: '//C1//C2
C        WRITE(*,*)
        WRITE(*,700)CH1,CH2,CH3
       ELSE
        WRITE(NF,*)'TRIAL No:',JS
        WRITE(NF,*)'SQUARED ERROR: '//C1//C2
C         WRITE(NF,*)
        WRITE(NF,700)CH1,CH2,CH3
       END IF
C
       DO 200 J=1,6
       IF(NF .EQ. 0) THEN
          WRITE(*,800)CHU(J),NTS(JS,J),CHT(J),TR(JS,J),CHV(J),
     &           VR(JS,J),CHV(J),SR(JS,J)
       ELSE
          WRITE(NF,800)CHU(J),NTS(JS,J),CHT(J),TR(JS,J),CHV(J),
     &           VR(JS,J),CHV(J),SR(JS,J)
       END IF
C
200    CONTINUE
C
700    FORMAT(18X,A,2(2X,A))
800    FORMAT(A,1X,I10,3(3X,A,F11.4))
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  GATHER THE STATISTICAL PARAMETERS ABOUT THE NET TRAINED
C  --NUMBER OF HIDDENS, WEIGHTS, EPOCHS
C  --ERRORS ON TRAINING DATA AND VALIDATION SET
CCCCCC
      SUBROUTINE GETSTA(NTST,NEUR,Y,TNER,VAER,NS,NZ,JS,MXL,MXC,MXO,MXE)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NVLDEX,NTRIAL
      COMMON /PTRAIN/ ALPHA,BETA,GAMMA,SHRINK,BSTSCR,THRESH,EWTSIN
      COMMON /NTRAIN/ NRETUR,NUMCND,NEPCIN,NBESTC,NUMLAY,NUMNEU
      COMMON /PTROUT/ ALPHO,BETO,GAMMO,SHNKO,THREO,DECAYO,EWTSOU
      COMMON /NTRNOU/ MXEPCO,NEPCOU,NBONO,MSRO,NEUO,METHO,NRESO
      COMMON /MODNEX/ NOX, NOV
      COMMON /NVAMOD/ MODV, MODR
C
      DIMENSION NEUR(MXL,0:MXC),NTST(NS,NZ),TNER(NS,NZ),VAER(NS,NZ)
      DIMENSION R(16),ERRV(16),Y(MXO,MXE)
C
      NB=6
      NR=16
      NHWTS=NCONEX(NEUR(1,0),MXL,NUMINP,NOX,NOV)
```

```
      NYWTS=(1+NUMINP+NUMNEU)*NUMOUT
C
      KX1=1
      KX2=NUMEXM
      CALL GETERV(Y,KX1,KX2,ERRV,NR,1,1,MODV,MODR,MXO,MXE)
C
      DO 100  K=1,6
        VAER(JS,K)=ERRV(NB+K)
        TNER(JS,K)=ERRV(K)
100   CONTINUE
C
      TNER(JS,3)=EWTSIN
      TNER(JS,4)=EWTSOU
      NTST(JS,1)=NUMNEU
      NTST(JS,2)=NHWTS
      NTST(JS,3)=NHWTS+NYWTS
      NTST(JS,4)=NEPCIN
      NTST(JS,5)=NEPCOU
      NTST(JS,6)=NEPCIN+NEPCOU
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  OBTAIN STATSTICAL RESULTS OF TRAINING AND TESTING
CCCCCC
      SUBROUTINE ERRSTA(NETTST,AERR,NS,NZ,MS,MODE,IDXMIN,NBST,NF)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      DIMENSION AERR(NS,NZ),NETTST(NS,NZ)
      CHARACTER*4 TITLE(6),CH*80
CCCCCC
      AVP=0.0
      TITLE(1)='MSE '
      TITLE(2)='RMS '
      TITLE(3)='VAR '
      TITLE(4)='STD '
      TITLE(5)='EIX '
      TITLE(6)='FVU'
C
      IF(MODE .EQ. 0) THEN
         WRITE (NF,*)'STATS ON VALIDATION SET:'
      ELSE IF(MODE .EQ. 1) THEN
         WRITE (NF,*)'STATS ON TEST SET:'
      ELSE
         WRITE (NF,*)'STATS ON TRAINING DATA '
         TITLE(2)='VAR '
         TITLE(3)='IEW '
         TITLE(4)='OEW '
      END IF
C
      CH='          AVERAGE         MIN          MAX'
     &          //'         MX-MN  AVPOS        STD_DEV'
      WRITE (NF,'(A)')CH
CCCCCC
       NBST=1
       VALUE=0.0
```

```
C
      DO 550 JE=1,NZ
         SUM=0.0
         DEV=0.0
         VMIN=AERR(1,JE)
         VMAX=AERR(1,JE)
C
         DO 500 JS=1,MS
            VALUE=AERR(JS,JE)
            SUM=SUM+VALUE
            DEV=DEV+VALUE*VALUE
            IF(VALUE .LT. VMIN) THEN
               VMIN=VALUE
               IF(IDXMIN .EQ. JE) NBST=JS
            END IF
            IF(AERR(JS,JE) .GT. VMAX)VMAX=AERR(JS,JE)
500      CONTINUE
C
         SUM=SUM/MS
         DEV=(DEV-SUM*SUM*MS)/(MS-1)
         DEV=SQRT(DEV)
         AVP=(SUM-VMIN)/(VMAX-VMIN)
         WRITE (NF,900) TITLE(JE),SUM,VMIN,VMAX,VMAX-VMIN,AVP,DEV
550   CONTINUE
CCCCCC
C
       WRITE(NF,*)
       IF(MODE .EQ. 0 ) THEN
       WRITE(NF,*)'BEST NET BY '//TITLE(IDXMIN)//' ON TRIAL No:',NBST
        WRITE(NF,*)
       END IF
900   FORMAT(A,1X,4F12.4,1X,F6.2,F14.4)
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  SUMMARY REPORT FOR TRAINING AND TESTING
C  THE STAT. RESULTS WRITTEN INTO FILE
CCCCCC
      SUBROUTINE REPORT(NTST,TNER,VAER,TSER,MSED,NS,NZ,NF,TRNFNM)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      COMMON /NUMVAR/ NUMINP,NUMOUT,NUMEXM,NTSTEX,NRUNEX,NTRIAL
      COMMON /NCANDP/ MXEP,NBON,IDXMSR,NUMCOD,METHOD,NTRANS,NRSEED
      COMMON /STATIS/ TSQE,SQER,VSQE,VVAR,VDSTD,ERRVAL,ERRTHR
      COMMON /WTSVAR/ WRANGE,BSTERR,TVAR,TDSTD,TSDSTD,VCOE,TCOE
C
      DIMENSION NTST(NS,NZ),TNER(NS,NZ),TSER(NS,NZ)
      DIMENSION VAER(NS,NZ),MSED(0:NS)
      CHARACTER*4 TA(6),CH*80,TRNFNM*30
CCCCCC
      TA(1)='HNUS'
      TA(2)='HWTS'
      TA(3)='TWTS'
      TA(4)='IEPC'
      TA(5)='OEPC'
      TA(6)='TEPC'
```

```
C
      CH='XCAS NETWORK'
      IF(MXC .EQ. 1) CH='CASCOR NETWORK'
C
      WRITE(NF,*) CH
      CALL WTHEAD(NF,TRNFNM)
      WRITE (NF,*)
      WRITE(NF,*)'TARGET_STD FOR TRAINING DATA    ',TDSTD
      WRITE(NF,*)'TARGET_STD FOR VALIDATION SET   ',VDSTD
      WRITE(NF,*)'TARGET_STD FOR TEST SET         ',TSDSTD
      WRITE(NF,*)
C
      AVP=0.0
      WRITE (NF,*)
      CH='          AVERAGE          MIN          MAX'
     &           //'          MX-MN  AVPOS          STD_DEV'
      WRITE (NF,'(A)')CH
CCCCCC
      DO 200 K=1,NZ
        SUM=0.0
        MINV=NTST(1,K)
        MAXV=NTST(1,K)
        DEV=0.0
        VAL=0.0
C
        DO 100 JS=1,NTRIAL
          VAL=DBLE(NTST(JS,K))
          SUM=SUM+VAL/NTRIAL
          DEV=DEV+VAL*(VAL/(NTRIAL-1))
          MAXV=MAX( MAXV, NTST(JS,K) )
          MINV=MIN( MINV, NTST(JS,K) )
100     CONTINUE
C
        AVP=(SUM-MINV)/DBLE(MAXV-MINV)
        DEV=DEV-NTRIAL*SUM*(SUM/(NTRIAL-1))
        DEV=SQRT(DEV)
        WRITE (NF,900) TA(K),SUM,MINV,MAXV,MAXV-MINV,AVP,DEV
C
200   CONTINUE
C
      NETBST=0
      WRITE (NF,*)
      CALL ERRSTA(NTST,VAER,NS,NZ,NTRIAL,0,1,NBST,NF)
        NETBST=NBST
      CALL ERRSTA(NTST,TSER,NS,NZ,NTRIAL,1,IDXMSR,NBST,NF)
      CALL ERRSTA(NTST,TNER,NS,NZ,NTRIAL,2,IDXMSR,NBST,NF)
C
      WRITE(NF,*)
      WRITE(NF,*)'===========STATS ON BEST NET====================='
      CALL SHOWER(NTST,TNER,VAER,TSER,NS,NZ,NETBST,NF)
      WRITE(NF,*)'================================================'
      WRITE(NF,*)
      DO 300 NT=1,NTRIAL
       CALL SHOWER(NTST,TNER,VAER,TSER,NS,NZ,NT,NF)
       WRITE(NF,*)
300   CONTINUE
C
```

```
900    FORMAT(A,1X,F12.1,3I12,1X,F6.2,F14.1)
       END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  RETURN A RANDOM NUMBER BETWEEN -0.5 AND 0.5
C  REFERENCE: "A PORTABLE RANDOM NUMBER GENERATOR FOR USE IN SIGNAL
C                 PROCESSING", SANDIA NATIONAL LABORATORIES TECHNICAL
C                 REPORT, BY S. D. STEARNS.
CCCCCC
      FUNCTION RANDOM(N)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      N=2045*N+1
      N=N-(N/1048576)*1048576
      RANDOM=(N+1)/1048577.0-0.5D0
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   COMPUTE OUTPUT OF HIDDEN NEURON
C   NTRANS=1:
C          1: LOGISTIC SIGMOID,             0<Y<1.0
C                Y(X)=1.0/(1+EXP(-X))
C          2: SYMETRIC LOG-SIGMOID,       -0.5<Y<0.5
C                Y(X)=1.0 /(1.0 + EXP(-X) - 0.5
C          3: HYPERBOLIC TANGENT SIGMOID, -1.0<Y<1.0
C                Y(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
CCCCCC
      FUNCTION FTRANS(S,NTRANS)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      Y=0.0
      IF(NTRANS .EQ. 0) THEN
       Y=S
      ELSE IF(NTRANS .LT. 3) THEN
CCCCCC--------------LOGISTIC SIGMOID:  0<Y<1.0--------------
         IF(S .LT. -15.0) THEN
            Y=0.0
         ELSE IF(S .GT. 15.0) THEN
            Y=1.0
         ELSE
            Y=1.0/(1+EXP(-S))
         END IF
CCCCCC--------------SYMETRIC LOG-SIGMOID: -0.5<Y<0.5---------
         IF(NTRANS .EQ. 2) Y=Y-0.5
C
      ELSE
CCCCCC-----HYPERBOLIC TANGENT SIGMOID, -1.0<Y<1.0------------
         IF(S .LT. -8.0) THEN
            Y=-1.0
         ELSE IF (S .GT. 8.0) THEN
            Y=1.0
         ELSE
            Y=EXP(-2.0*S)
            Y=(1.0-Y)/(1.0+Y)
```

```
         END IF
      END IF
C
      FTRANS=Y
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   COMPUTE DERIVATIVE OF ACTIVATION FUNCTION FOR A HIDDEN NODE
CCCCCC
      FUNCTION FPRIME(F,NTRANS)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      FP=0.0
      IF( NTRANS .EQ. 0 ) THEN
         FP=1.0
      ELSE IF( NTRANS .EQ. 1 ) THEN
         FP=F*(1.0-F)
      ELSE IF(NTRANS .EQ. 2) THEN
         FP=0.25-F*F
      ELSE
         FP=1.0-F*F
      END IF
C
      FPRIME=FP
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   COMPUTE DERIVATIVE OF ACTIVATION FUNCTION FOR OUTPUT NODE
C
CCCCCC
      FUNCTION OPRIME(F,NTRANS)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      FP=0.0
      IF( NTRANS .EQ. 0 ) THEN
         FP=1.0
      ELSE IF( NTRANS .EQ. 1 ) THEN
         FP=F*(1.0-F)+0.1
      ELSE IF(NTRANS .EQ. 2) THEN
         FP=0.25-F*F+0.1
      ELSE
         FP=1.0-F*F+0.1
      END IF
C
      OPRIME=FP
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   QUICKPROP (MINIMIZING) FOR UPDATING WEIGHTS, BY SCOTT FAHLMAN, 1990
C  I: THE I-TH WEIGHTS
C  HW: WEIGHTS OF HIDDEN UNITS
C  PS: PREVIOUS SLOPE
C  S:  CURRENT SLOPE
C  DW: PREVIOUS DELTA W
C  A:  LEARING RATE (0.1~1.0, 0.6 IS OK)
```

```
C  B:  MAX LEARING RATE(0.8~2.0)
C  SF:  SHRINK FACTOR =B/(1.0+B)
C  DEC: DECAY FACTOR (.0001 FOR OUTPUT UPDATING)
CCCCCC
      SUBROUTINE QKPROP(I,MXW, CW,DW,PS,S, A,B,SF,DEC)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
C
      DIMENSION CW(MXW),PS(MXW),S(MXW),DW(MXW)
CCCCCC
      D=DW(I)
      SL=S(I)+D*DEC
      PSL=PS(I)
      DX=0.0
CCCCCC
      IF(D  .LT. 0.0) THEN
C LAST STEP WAS NEGATIVE
C
        IF(SL  .GT. 0.0) DX=-A*SL
C
        IF(SL  .GE. (PSL*SF)) THEN
           DX=DX+B*D
        ELSE
           DX=DX + D*SL/(PSL - SL)
        END IF
CCCCCC
      ELSE IF(D  .GT.  0.0) THEN
         IF(SL  .LT. 0.0) DX=-A*SL
C
         IF(SL  .LE. (PSL*SF)) THEN
            DX=DX+B*D
         ELSE
           DX=DX + D*SL/(PSL - SL)
         END IF
C
      ELSE
         DX=-A*SL
      END IF
C
      DW(I) = DX
      CW(I) = CW(I) + DX
      PS(I) = SL
      S(I) = 0.0
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

VITA

YULEI BAI

Candidate for the Degree of

Master of Science

Thesis: AN EXTENDED CASCADE CORRELATION NEURAL NETWORK

Major Field: Computer Science

Biographical:

Education: Received the Bachelor of Science in Petroleum-Geology from Northwest University, Xi'an, China, in July 1985; received Master degree in Petroleum Geology from Research Institute of Petroleum Exploration and Development, Beijing, China, in July 1989. Completed the requirements for Master of Science at Oklahoma State University in May 2002.

Professional Experience: Employed by Research Institute of Petroleum Exploration and Development, Beijing, China, as a Research Geologist, August 1989 to September 1997;