A WATER DEMAND FORECASTING SYSTEM

USING FUZZY LOGIC

By

YU-WEN LIN

Bachelor of Science

National Chung-Hsing University

Taiwan, R. O. C.

1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1998

OKLAHOMA STATE UNIVERSITY

A WATER DEMAND FORECASTING SYSTEM

USING FUZZY LOGIC

Thesis Approved:

H. Lu

Thesis Advisor

John Hatcliff

G. E. Hedrick

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Huizhu Lu, for helping me focus this study, for giving generously of her expertise, and for her patience. Also appreciation is extended to the other members of my committee, Dr. G. E. Hedrick and Dr. John Hatcliff, for their guidance, patience, and encouragement.

Appreciation is also expressed to Stillwater Research Station and City of Stillwater for their kindly help during the data preparation.

Special thanks are offered to Mrs. Gay Clarkson for her time and valuable help in proving the writing of the thesis.

My sincere thanks go to Ms. Wein-Pin Yeh for her effort of reviewing the thesis and emotional support.

Finally, special gratitude and love is expressed to my parents and my family for their patience and encouragement throughout my academic years. I wouldn't be able to reach this goal without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

Neural Networks and Fuzzy Logic have been attractive to researchers and engineers in the field of Artificial Intelligence (AI) for a little longer than a decade. Both approaches were introduced to emulate the way human beings learn (Neural Networks) and think (Fuzzy Logic). Neural Networks have made modeling problems easier by self-learning from a given set of training data. Successful applications using fuzzy logic and fuzzy systems also have emerged. In addition, the concept of fuzzy logic has been applied to other research fields, such as neural network training.

Although many people claim that Neural Networks have self-learning abilities for complicated problems, the degree of success varies from application to application. It is not easy to construct appropriate neural networks for the following reasons. First, when training parameters do not fit the problem, they may degrade the performance of the trained neural network. Secondly, the neural network's training time is not always acceptable. Finally, that the expert's expertise and experience cannot be incorporated into the neural network for certain problems. Constructing proper neural networks and training the network is more of an art than a science.

Dr. Lotfi A. Zadeh of the University of California at Berkeley introduced the concept of fuzzy sets in 1965. Fuzzy logic is based on fuzzy sets and offers a new mathematical model to express imperfection -- the way humans describe their experiences, expertise, thoughts, or reasoning from facts. Engineers and scientists then employ fuzzy logic to modeling problems, which are called fuzzy systems. Because it is a much easier and are intuitive approach to integrate engineers or scientists' experience and expertise to modeling problems, the applications of fuzzy systems have been adapted by some engineers in building control systems. Requirements for building well-performed fuzzy systems include well-defined antecedent and consequent membership functions, and inference rule base. That is, the performance of a fuzzy system is definitely determined by the experiences and expertise provided by engineers or experts. Compared to neural network systems, the main drawback of fuzzy systems is that they cannot learn from representation data, which, on the other hand, is the main advantage of neural network systems.

While seeking to improve the performance of neural networks and give fuzzy system self-intelligence, the fusion of neural network and fuzzy logic has become a very popular research field in recent years. The major research direction includes the following aspects:

(1) In order to improve the neural networks' performance, fuzzy neurons may be introduced to incorporate the expert's experience on neural networks' training procedure. Experienced scientists believe neural networks have the ability to learn faster and better establish fuzzy rules.

(2) Use neural network techniques to make fuzzy systems self-adaptive. Neural network techniques may be used to generate inference rule base, antecedent membership functions, and consequent membership functions from given data sets.

1.2 Research Objective

Though fuzzy logic is often used to design control systems rather than forecasting systems, it is reasonable to utilize fuzzy logic to model forecasting problems since both systems usually try to model the target system by applying existed knowledge to the problem. In addition, we often use observed data (inputs and outputs) in modeling procedures and try to give accurate predictions from future input for both systems.

In this research, we build a simple water demand forecasting system using fuzzy logic as the modeling method. But, instead of producing the membership functions and fuzzy rules intuitively, we utilize a program, FuNeGen, to generate the membership functions and fuzzy rules, and a neural network representing the fuzzy system. We study the generated neural network and fuzzy rules, and then we build and tune the simple water demand forecasting system.

The experimental data used in this research includes daily water flows from the Water Treatment Plant, City of Stillwater, and weather data (temperature) from Stillwater Research Station.

CHAPTER II

LITERATURE REVIEW

2.1 Neural Networks

Neural networks (NN), or artificial neural networks (ANN), are "an abstract simulation of a real nervous system that contains a collection of neuron units communicating with each other via axon connections." [Kung 93] The fundamental neural net model was proposed by McCulloch and Pitts in 1943 as a computational model called "nervous activity." The neuron proposed was a simple binary device with a fixed threshold to perform simple threshold logic. The model leads the work of John von Neumann, Marvin Minsky, Frank Rosenblatt, and many others, in the development of modern computer. Researchers believe that, by embedding an enormous number of simple neurons in an interacting nervous system, it is possible to provide computational power for very sophisticated information processing [Anderson 1972].

A simple typical neural network including an input layer, a hidden layer, and an output layer is shown in Figure 1.

Neural networks may have an unlimited number of hidden layers and, in each hidden layer, an unlimited number of neurons, while the number of neurons in input and output layers is most likely determined by the problem itself. Some researchers claimed that one hidden layer is good enough for all problems; however, it has not been proven true. Even if it is true, how to choose the number of neurons in the hidden layer is still an open question. The number of hidden layers and neurons are critical because they determine the network's performance. If too few neurons are placed, the network may not have enough "degree of freedom" to accurately model the target problem. On the other hand, if too many neurons or hidden layers are placed in the network, the training progress may become unacceptably slow. In addition, if the network is trained for a long time, it starts memorizing the specific training sets, including the imbedded noise, rather than developing a generalized model for the underlying problem. In practice, constructing a precise network requires more knowledge for neural networks than the underlying problem.

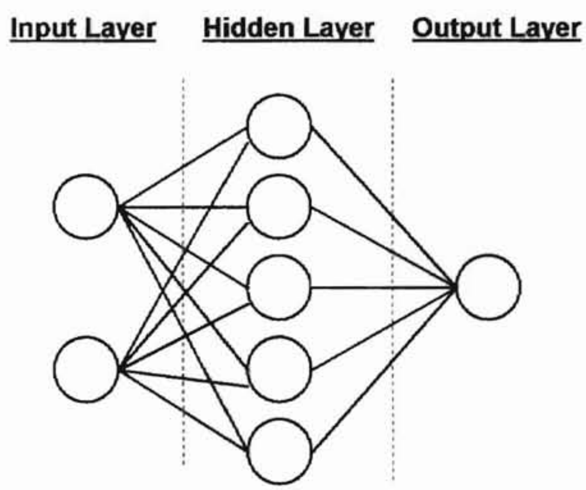


Figure 1 A typical neural network

The first learning procedure for neural networks was proposed by Rumelhart, Hinton and Williams [86]. With the goal of minimizing the error (measure of difference) between the actual output of the network and the desired output according to current input, the procedure repeatedly adjusts weights in the network. The procedure is called Back Propagation and still is one of the most popular learning algorithms for training neural networks. The Back Propagation learning algorithm is described below.

2.2 Back-Propagation Learning

The standard back-propagation algorithm is a simple version of gradient descent that aims to find a set of weights (weight space) which ensures that for each input vector the output vector gained from the trained network is the same as (or very close to) the desired output vector. The schematic diagram from [Kung 93] describing the back-propagation process is shown in Figure 2.

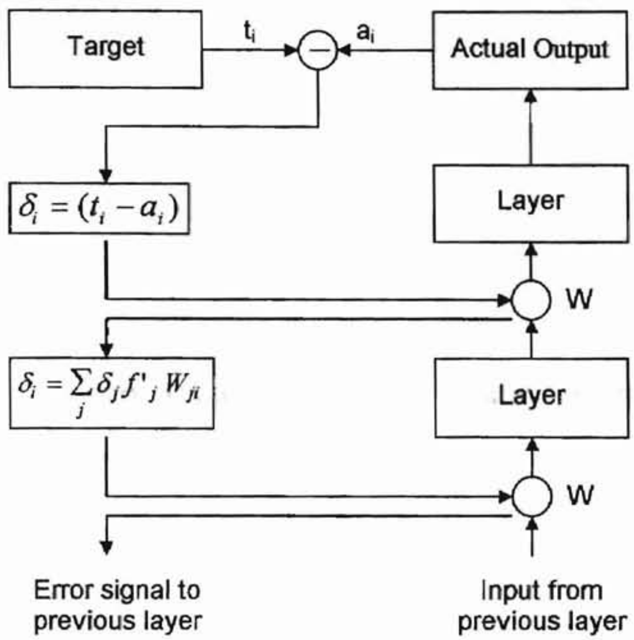


Figure 2 The schematic diagram for the back-propagation process

Note that δ_i is the error signal that feeds back to the previous layer, and f'_j is the first derivative of f_j, the activation function, which usually is a threshold function for firing a neuron's output. The general concept of back-propagation algorithm is to calculate error signal from each layer's output and then propagate the error signal to the previous layer for weights adjustment, so as to adjust the previous layer's output. A more detailed description of back-propagation algorithm is given below.

Every neuron in a neural network can have one or more than one input and one or more than one output. In the standard Back-Propagation learning algorithm, each neuron has at least one input and only one output, as described below.

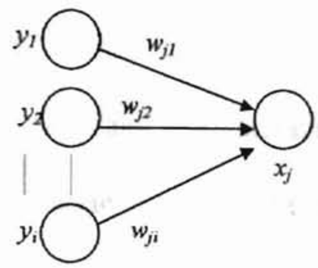


Figure 3: Calculation of total input for neuron x_j

As shown in Figure 3, the total input from neurons y_i to neuron x_j is a linear function

$$x_j = \sum_i y_i w_{ji} \text{ ,} \quad \text{(1)}$$

where w_{ji} is the weight from neuron y_i to neuron x_j.

The output for neuron x_j is a non-linear function of its total input, as shown below.

$$\frac{1}{1+e^{-x_j}} \quad \text{(2)}$$

The learning procedure aims to find a set of weights (weight space) which ensures that, for each input vector, the output vector gained from the trained network is the same as (or very close to) the desired output vector. The total error, E, is computed by following equation:

$$E = \frac{1}{2}\sum_p \sum_j (y_{j,p} - d_{j,p})^2 \text{ ,} \quad \text{(3)}$$

where p is an index over cases (input-output pairs, or patterns), j is an index over output units, y is the actual state (output) of an output unit, and d is its desired state (output).

The standard back-propagation algorithm involves two phases. In the first phase, the input vector is presented and propagated forward through the network to calculate the actual output value for each output neuron (unit). Then this output value is compared with the desired output and an error signal δ_{pj} is generated for each output unit j.

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}}, \qquad (5)$$

where E_p is the total error of pattern p, and

$$net_{pj} = \sum_i w_{ji} y_{pi} \;, \qquad (6)$$

where y_{pi} is the total output of neuron i over pattern p, and w_{ji} is the weight between neuron i and neuron j.

The second pass is a backward pass through the network. It passes the error signal to each unit in the network and an appropriate weight change is made. The weight change is defined by

$$\Delta w_{ji}(t+1) = \eta\, (\delta_{pj} y_{pi}) + \alpha\, \Delta w_{ji}(t), \qquad (7)$$

where the subscript t indexes the time, η is the learning rate, and α is a constant that determines the effect of past weight changes on the current direction of movement in weight space, also known as momentum.

The learning rate η is a constant used to determine how much the weight should change, while the momentum α is used to determine the relative contribution of the past weight change gradient to the current weight change.

The standard back-propagation algorithm, however, generally lacks the ability to produce an effective neural net for a given task within a reasonable time. [Hinton 87] [Fahlman 88] In some cases, the training time a network takes is too long to be accepted. [XWB 92] The first issue causing a slow training time is the learning rate η. It has to be small enough in order not to overshoot the goal [Fahlman 88]; however, the learning rate is a constant in standard back-propagation algorithm and can not be adjusted in the training session. To improve the training time, the momentum α was then introduced, as described in equation (7).

Some researchers also proposed other methods other than the momentum parameter. Weir [Weir 91] proposed a method. The method also adjusts the learning rate only, but in terms of the length and direction of the next step in the weight space towards the goal weight space.

Another major reason causing slow training time is that the activation functions in standard back-propagation algorithm are not changeable [XWB 92.] The sigmoid function,

$$a = \frac{1}{1+e^{-\sigma x}} \quad , \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

uses $\sigma = 1$ (σ is steepness parameter) for all inputs during the entire training session in standard back-propagation algorithm. The training process may be very slow for $\sigma = 1$, but may be overshooting the goal if σ *is* too large.

Researchers accumulate their experiences of how the learning rate and activation function can affect the training performance over different situations. When the fuzzy logic became more acceptable, they found that the embedded fuzzy logic rules, which dynamically adjusts the learning rate and activation function in training session, can greatly improve the training time. The concept of fuzzy logic is briefly described below.

2.3 Fuzzy Sets, Fuzzy Logic and Fuzzy Systems

Conventional binary logic is based on binary outputs, true and false in linguistic, or 1 and 0 in numeral. The technique is good enough to design modern computers, but not appropriate to describe all events in the real world, especially human experiences and expertise, which can not always be covered by the true-false cases. Therefore, fuzzy theory was introduced by Dr. Lotfi Zadeh to deal with the problem. Fuzzy theory involves fuzzy sets, fuzzy logic and is applied to fuzzy system. More detailed descriptions are given below.

2.3.1 Fuzzy Sets

Fuzzy set theory has a strong relationship with classical set theory. In classical set theory, a subset of **A** can be defined as a mapping from set **B** to set $\{1,0\}$,

$A: B \rightarrow \{1,0\}$

The mapping can be represented as ordered pairs. The first element of the ordered pair is an element from set **B**, and the second element from set $\{1,0\}$. That is, the possible mapping for each element in **B** is either 1 or 0, where value 1 is used to represent membership (true) and value 0 is used to represent non-membership (false).

In fuzzy set theory, the set **A** can be defined as a mapping from set **B** to interval [0,1], that is, the second element of the ordered pairs is a value between 0 and 1. Compared to classical set theory's membership-non-membership (true-false) mapping, the second element in fuzzy sets is used to represent the "degree of membership" for each element in set **B**. The value 1 represents a complete membership and the value 0 represents complete non-membership. In practice, membership functions are used to describe the relationship between an input and its correspond membership. In addition we can find that fuzzy set is a superset of classical set, or classical set is a special case in fuzzy set.

2.3.2 Fuzzy Logic

As that fuzzy set is a superset of classical set, fuzzy logic is a superset of conventional logic (binary logic, or Boolean logic). The results of conventional logic operation map to binary set {0,1}; the results of fuzzy logic operation map to fuzzy set that is in interval [0,1]. Operators commonly used in binary are used in fuzzy logic operation but have more complicate inter-operations. For example, the standard definitions in fuzzy logic are:

Truth (not x)	=	1.0 - Truth (x)
Truth (x and y)	=	Minimum (Truth(x), Truth(y))
Truth (x or y)	=	Maximum (Truth(x), Truth(y))

2.3.3 Fuzzy Systems

Systems that use fuzzy logic for reasoning data are called Fuzzy Systems. A fuzzy system involves collections of membership functions and inference rules. The rules can be represented as linguistic form, such as

If x is LOW and y is HIGH then z is MEDIUM

Dr. Lotfi Zadeh thinks that the fuzzy theory should be regarded as the process of "fuzzification" and as a methodology to generalize any specific theory from a crisp (discrete) to a continuous (fuzzy) form.

Fuzzy systems are mostly applied and known in designing control system, or fuzzy control system. Engineers may build a control system very easily without the process of mathematics modeling; instead, they use fuzzy theory to model the system in an intuitive manner. A typical fuzzy system is illustrated in Figure 2.

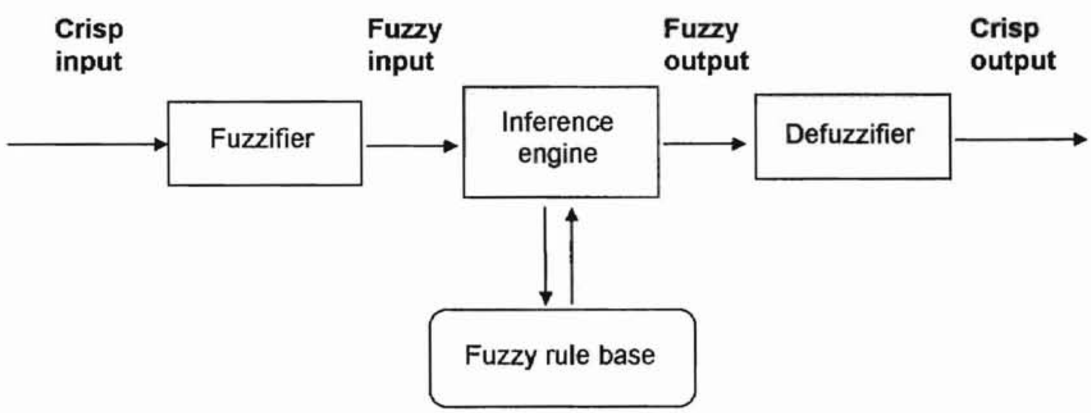


Figure 4: A typical fuzzy control system

As shown above, the crisp input is translated (or fuzzified) to fuzzy input with respect to the antecedent membership function. For example, a crisp input of 75 degrees in temperature can be represented as 80% HIGH and 15% VERY HIGH. Note that an input can have membership in more than one category. Then the fuzzy inputs are applied to the inference engine and consult all applied rules, which gives the result of one or more fuzzy outputs. Simple fuzzy rules can be:

IF Temp is HIGH Then Fan-Speed is HIGH, and

IF Temp is VERY HIGH Then Fan-Speed is VERY HIGH.

Here, according to our example, the fuzzy output is 80% HIGH and 15% VERY HIGH for fan speed. Therefore, the fuzzy output needs to be converted (defuzzified) to a crisp target output with respect to consequent membership functions. Then the result is sent to the fan speed controller.

Designing a fuzzy system is quite intuitive and fairly easy for experienced engineers and scientists. With relative small amounts of time, they can build up the antecedent membership functions, the fuzzy inference rules, and the consequent membership functions. However, for an unknown question, tests and experiments are still necessary to design proper membership functions and inference rules. Furthermore, a fuzzy system is inherently not self-adaptive to given input-output patterns.

2.4 Fusion of neural networks and Fuzzy Logic Techniques

2.4.1 Improve Neural Networks with Fuzzy Logic

The standard back-propagation algorithm can train neural networks and is quite easily to implement, hence most improvement works are focused on optimizing the standard back-propagation algorithm. The problems of standard back-propagation algorithm include high convergence time and the possibility of ending up in a local minimum. Approaches proposed to improve the quality of back-propagation algorithm were mostly for dynamic adaptation of network parameters, especially for the learning rate. Among them are global optimization approaches, which adjust learning rate and sometimes also momentum parameters for all weight in the network globally; and local optimization approaches, which assign each synapse a different learning rate and parameters are adapted locally in the network [HG 94]. The results are successful for some applications and the neural networks can be tuned to obtain excellent performance. Unfortunately, the tuned parameters are application dependent. Along with the empirical knowledge gained from the researches, however, the fuzzy rule base was built.

Fuzzy controlled dynamic adaptation of back-propagation algorithm was introduced in 1992 by several researchers [XWB 92] [ACMC 92]. Xu et. al. developed a Fuzzy Associative Memory (FAM) system and used 24 FAM rules to define a self-adjusting activation function and learning rate function. The FAM system was integrated in their neural network and back-propagation algorithm. The following section is adapted from [XWB 92] which briefly describes the FAM system.

(1) The learning rate function $C(E,t)$ is defined as a function of overall error, E, and training time, t, rather than a constant in standard back-propagation. The underlying principles of adjusting the function are briefly explained below:

(a) $C(E,t)$ should be large when the error E is considered big, indicating that the weights are far away from the desired ones; $C(E,t)$ should become small when the error E is very small, showing that the weights are modified close to the desired.

(b) If training time t is fairly short, $C(E,t)$ should be large to promote the learning speed of back-propagation. $C(E,t)$ should become small toward better convergence in the final stage when t is significant long.

(2) The self-adjusting activation function is defined by

$$S(E,t,a_i) = \frac{1}{1+e^{-\sigma(E,t)a_i}},$$

where a_i is the sum of weighted input to the neuron, and $\sigma(E,t)$ is a constant, 1, in standard back-propagation but called an accelerator determined by FAM systems.

The underlying adjusting principles of the function are briefly described below:

(a) If training time is relatively short and the error is quite big, the smaller numerical value of the accelerator $\sigma(E,t)$ can relax all inputs to the neurons being considered, so that the function $S(E,t,a_i)$ becomes flat and "soft" in order to allow the initial weights of related connections to adjust quickly and easily.

(b) When the error is very small and/or training time is very long, $\sigma(E,t)$ should be larger and $S(E,t,a_i)$ should become steep and "hard" to all inputs to the neurons, such that the weights are convergent toward the desired goals precisely.

The approach (called Fuzzy neural networks, or FNN) did show exciting results in improving learning speed compared to standard back-propagation (SBP) and back-propagation with self-adjusting learning rate function (BP+C). According to their reports, the FNN was more than ten times faster than the BP+C and almost 30 times faster than the SBP to converge to minimum error. Halgamuge et. al. [HMG 93] used the same approach with additional fuzzy rules and showed even better results than the FNN proposed by Xu. The additional statements of the used fuzzy rules presented by Halgamuge are described below.

"(1) A high error means being far away from the minimum. Hence the learning rate should be high.

(2) The change of the error (*CE* in short) from iteration to iteration is the most important and significant measure:

$$CE(t) = E(t) - E(t\text{-}1)$$

As long as it is high the learning rate can be increased quite safely. Negative *CE* indicates that the minimum has been past.

(3) Since the idea is to use a very large learning rate if possible, it is important to know when to decrease it again as to avoid overshot. The experiments have shown that the most reliable measure is the second change of error. Originally only the sign of this value was regarded. Positive sign means an increase in *CE* which in turn means it is safe to increase the learning rate. But it also proved beneficial to take the magnitude of this value into account because it contains information about the trend in *CE*: Even *CE* remains positive for consecutive iterations, its decrease gives an early hint that the minimum is being approached. To be independent of the magnitude of *CE* the *QCE* measure is introduced as a quotient computed as

$QCE = CE(t) / CE(t\text{-}1)$

Values of *QCE* smaller than 1 should lead to a decrease of the learning rate. Additionally, these statements are introduced under the assumption that the error surface for each weight can be approximated by parabola."

2.5 Neural Networks for Fuzzy Systems

Since neural network is capable of learning from given data set, it is natural to use neural networks to improve fuzzy systems. Among the proposed approaches, Halgamuge et. al. [HG 94] constructed a special multi-layer neural network for generating fuzzy systems that had showed great performance. The fuzzy-neural network (FNN) can be used to generate antecedent membership functions, consequent membership functions, fuzzy rule base, and defuzzification function from the given data set.

The FNN is implemented based on mapping fuzzy systems into a feed-forward type of neural networks. Major components of the FNN structure included Fuzzification section, Rule Generation section, and Defuzzification section. The system is briefly described by Halgamuge et al in [HG 94] as follows.

" (1) Rule neurons were used to implement the premise of fuzzy rules. Three types of rules can be created:

(a) simple rules with premises containing a single fuzzy variable

(b) conjunctive rules with many fuzzy variables in premises

(c) disjunctive rules with many fuzzy variables in premises

(2) The generated antecedent and consequent membership functions are sigmoidal functions, among all other options. The number of categories per input in antecedent membership functions was assigned by user. The FNN will try to reshape the function and reduce the number of categories.

(3) Linear neurons were used to generate consequent membership functions, rather than the sigmoid function.

(4) Instead of choosing the standard defuzzification method such as Center of Gravity (COG), the FNN utilized Customizable Basic Defuzzification Distributions (CBADD), which was said to have an optimizable approach with theoretical or application oriented considerations.

(5) Each generated rule is given a weight (or bias) and is used to compute the rule's strength.

(6) The crisp output is computed by function

$$O_i = a_i\left(\sum_{j=1}^{r} W_{ij} * K_j\right)$$

Where W_{ij} is the weight from connection of jth rule node to the ith output node, K_j is the rule strength, r is the number of rules, and a_i is the sigmoidal activation function." Note that the function differs from that used in the conventional neural network, which sums up the weights only. Here the rule strengths are considered as input also.

CHAPTER III

EXPERIMENTAL RESULTS

3.1 Data Used for Training and Testing

The data used in the research includes

1. Date (month and day),
2. Daily flow data from Stillwater Water Treatment Plant, and
3. Daily high and low temperatures data from Stillwater Research Station.

A diagram of the original data is shown in Figure 3. Compilations included 267 records from the training, file and 91 records from the testing file. Each record in the training and testing files has two lines, input vector and actual output for the first line and the second line respectively, arranged in the form shown below.

```
Month       Temp_High  Temp_Low   Prev_Flow
Flow
```

Where

`Month:` The number value of month, from 1 to 12

`Temp_High:` The high temperature of the day in °F

`Temp_Low:` The low temperature of the day in °F

`Prev_Flow:` The water flow of the previous day

Flow: The flow of the day

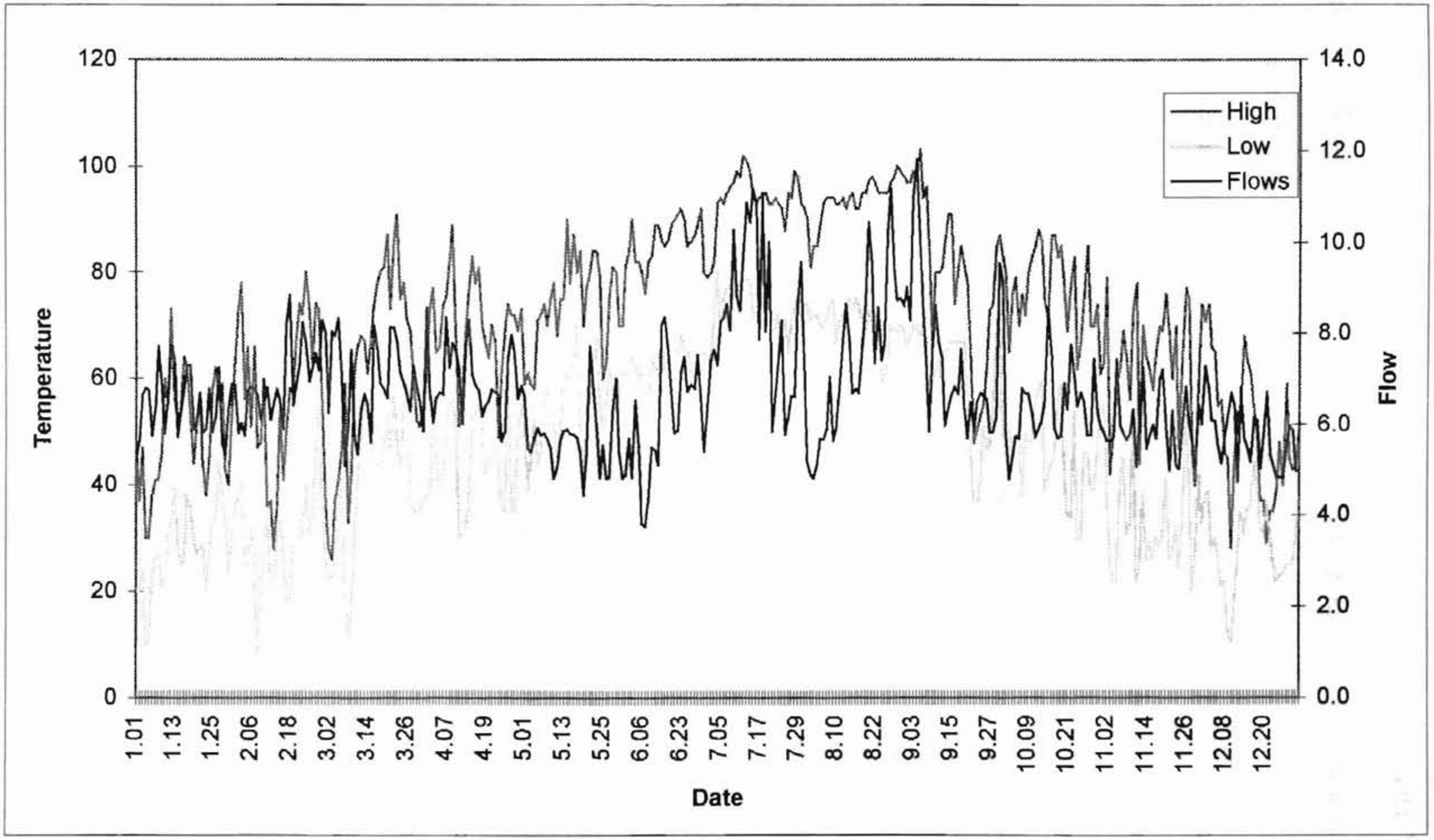


Figure 5: A brief view of the original data et. The left and right Y-axi indicate temperature and flow, re pectively.

The compiled experimental data set excludes the value of day because our training experiences showed degraded training performance when day is included. We include the previous flow in the data set. Because, according to our observations, it shows the influence on next day's flow and we want to prove it.

In addition, the input data is not normalized to domain [0,1]. The original values are preserved because the training program FuNeGen v1.1 does the normalization work internally. It is a great advantage for users to easily observe the generated output.

The FuNeGen program has many options for users to set and some of these options affect the training performance. Among all the options, the number of rules is the major cause that affects the performance. It can be easily understood as giving the training neural network sufficient space for all possible rules. It is like choosing an appropriate number of neurons for the target network. We consider this an advantage over constructing a conventional neural network. According to our experience, if all of them are effective to the target system, the number of rules should be about two times the number of inputs. We also find that the number of antecedent membership functions affects the training performance. But, since the FuNeGen program tunes the shape and number of membership functions, it is safe to set the number to 5, the maximum value allowed in FuNeGen.

3.2 Training Results

The training results extracted fuzzy rules, antecedent membership functions, and a neural network representing the fuzzy system.

3.2.1 The Extracted Fuzzy Rules and Discussions

The extracted fuzzy rules of the target neural network are shown below in the order of rule weight. Note that the number of rules and the number of antecedent membership functions were set and were decided by experiences from experiments.

Table I The extracted fuzzy rules

No	Rule Weights	Fuzzy Rules
1	1.163	IF (I1 mf4) THEN output0
2	-1.274	IF (I0 mf0) AND (i1 mf4) THEN NOT output0
3	0.788	IF (I0 mf2) AND (i1 mf1) THEN output0
4	-0.561	IF (I0 mf2) THEN NOT output0
5	-0.533	IF (I0 mf0) OR (i3 mf0) THEN NOT output0
6	-0.370	IF (I0 mf4) THEN NOT output0
7	-0.304	IF (I0 mf1) THEN NOT output0
8	0.278	IF (I1 mf3) OR (i2 mf4) THEN output0
9	0.240	IF (I0 mf2) OR (i2 mf2) THEN output0
10	-0.144	IF (I0 mf3) OR (i2 mf2) THEN NOT output0
11	-0.105	IF (I1 mf3) THEN NOT output0
12	-0.022	IF (I2 mf2) OR (i3 mf3) THEN NOT output0

The `I0`, `I1`, `I2`, and I3 represent Month, Temp_High, Temp_Low and Prev_Flow respectively. The classifier `mf0`, `mf1`, `mf2`, `mf3`, and `mf4` for membership functions can be usually translated to Very Low, Low, Medium, High, and Very High memberships respectively. Therefore, the extracted rules can be rewritten into more readable form. Note that a negative weight means the rule has negative effect on the output flow.

Table II The translated fuzzy rules

Rule	Rule Description		Action
1	If High Temperature is Very High	Then	Output = RS * 1.163
2	If Month is Very Low AND High Temperature is Very High	Then	Output = RS * -1.274
3	If Month is Medium AND High Temperature is Low	Then	Output = RS * 0.788
4	If Month is Medium	Then	Output = RS * -0.561
5	If Month is Very Low OR Previous Flow is Very Low	Then	Output = RS * -0.533
6	If Month is Very High Then	Then	Output = RS * -0.370
7	If Month is Low	Then	Output = RS * -0.304
8	If High Temperature is High OR Low Temperature is Very High	Then	Output = RS * 0.278
9	If Month is Medium OR Low Temperature is Medium	Then	Output = RS * 0.240
10	If Month is High OR Low Temperature is Low	Then	Output = RS * -0.144
11	If High Temperature is Low	Then	Output = RS * -0.105
12	If Low Temperature is low OR Previous Flow is High	Then	Output = RS * -0.022

RS : Rule strength

There are a few good results in the extracted fuzzy rules.

1. The first rule is a very convincing rule. It says that if the high temperature of a day is very high, then the output flow has the largest weight, 1.163. That reflects our assumption as well as the collected data very well.

2. Rules 4, 5, 6 and 7 are good examples too. Rule 4 says that if the month is either May, June, July or August, then the output flow has a negative weight of -0.561. Rule 5 says that if the month is very low, e.g., January, or the previous flow is very low, then the output flow has a negative weight –0.533. Rule 6 says that if the month is very high, e.g., December, then the output has a negative weight –0.370. Rule 7 says that if the month is low, e.g., April and May, then the output flow has a negative weight –0.304. These rules support the facts in our collected data that the lowest flow values occur in January, May and December.
3. Rule 10 and 11 also make sense. They both indicate that low temperature has negative effect on flow, though not very much.
4. Rule 12 somehow proves our reasonable guess – if the previous day's flow is high then the day's flow tends to be low. However, the rule's weight is the smallest, among all rules.

3.2.2 The Extracted Membership Functions

Figure 5 to Figure 8 are screen snaps of the extracted membership function for our inputs. They are generated and optimized by the FuNeGen program. In FuNeGen, the degree of membership is determined by following two sigmoidal functions.

$$\mu = \frac{1}{1+e^{-S(I-\alpha)}} \quad \text{.......} (9)$$

$$\mu = \frac{1}{1+e^{S(I-\alpha)}} \quad \text{.......} (10)$$

μ : Degree of membership

S : Steepness factor, or gradient factor

I : Normalized input

α : Shift factor

Equation (10) is actually a mirror of Equation (9). Suppose S is a positive number, Equation (9) represents a curve with rising edge, which can be used to describe the Very High membership function. On the contrary, Equation (10) represents a curve with falling edge that describes Very Low membership function, as shown in Figure 3.

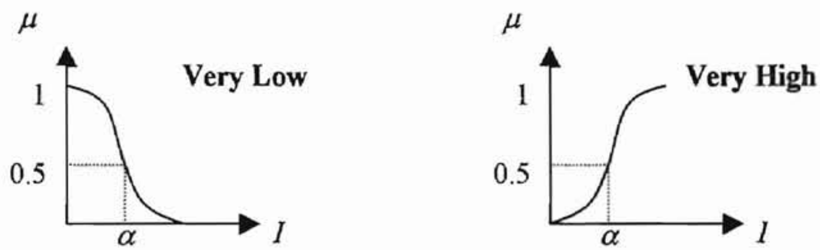


Figure 6: Antecedent Membership Functions for Very Low and Very High

Membership functions having both rising and falling edges, such as Medium, can be considered as a combination of Equations (9) and (10), as shown in Figure 4.

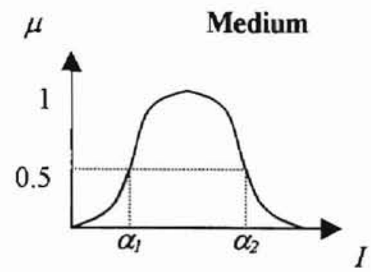


Figure 7: Antecedent Membership Function for Medium

Initially, the steepness factor S is 14.0. The shift factors α for membership functions, which can be found in FuNeGen's option settings, are listed below.

Table III The initial shift factors before optimization

(1)	**Very Low:**	0.15 (falling edge)
(2)	**Low:**	0.2 (rising edge) and 0.38 (falling edge)
(3)	**Medium:**	0.4 (rising edge) and 0.6 (falling edge)
(4)	**High:**	0.62 (rising edge) and 0.8 (falling edge)
(5)	**Very High:**	0.85 (rising edge)

FuNeGen optimizes membership functions by adjusting S and α, the steepness and shift factors.

Pursuant to the five shift factor values and the steepness factor, a set of membership functions are obtained by substituting each shift factor value, the α value, and the steepness factor, the S value, into Equations (9) and (10). The result graph is shown with gray curves in Figure 8 to Figure 11. The dark curves, on the other hand, show the optimized membership functions.

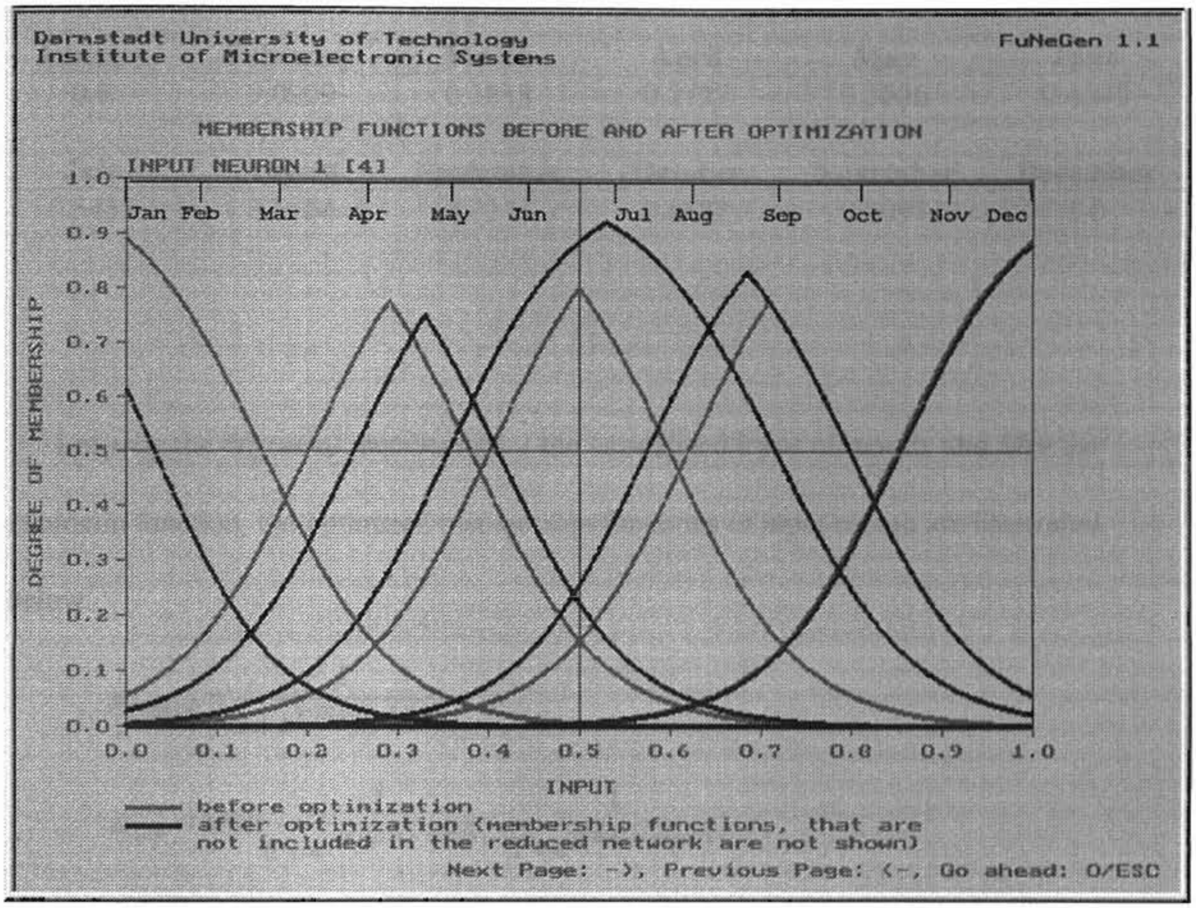


Figure 8: Membership functions vs Months (`IO`)

The initial membership functions vs input Months are shown with gray curves in Figure 8. The membership functions' shapes and scales are adjusted during optimizations. We can see that the degree of membership for months of Very Low (such as January and February) is lowered. In addition, the coverage of membership function for months of

Medium is wider than the initial setting, and makes August have a higher degree of membership in Medium Months. The normalized input values of Month are listed below.

Table IV Normalized input values of input Month (`I0`)

January	February	March	April	May	June
0.0	0.0909	0.1818	0.2727	0.3636	0.4545

July	August	September	October	November	December
0.5455	0.6364	0.7273	0.8182	0.9091	1.0

Let μ be the degree of membership, I the normalized input of month, and *Min* the minimum function, the optimized membership functions of input month are illustrated below.

$$\mu_{VeryLow} = \frac{1}{1+e^{15.0243(I-0.0312)}}$$

$$\mu_{Low} = Min(\frac{1}{1+e^{-14.0217(I-0.2518)}}, \frac{1}{1+e^{13.8484(I-0.410)}})$$

$$\mu_{Medium} = Min(\frac{1}{1+e^{-14.2724(I-0.3546)}}, \frac{1}{1+e^{13.3074(I-0.7174)}})$$

$$\mu_{High} = Min(\frac{1}{1+e^{-14.2850(I-0.5740)}}, \frac{1}{1+e^{14.0182(I-0.7996)}})$$

$$\mu_{VeryHigh} = \frac{1}{1+e^{-13.9762(I-0.8540)}}$$

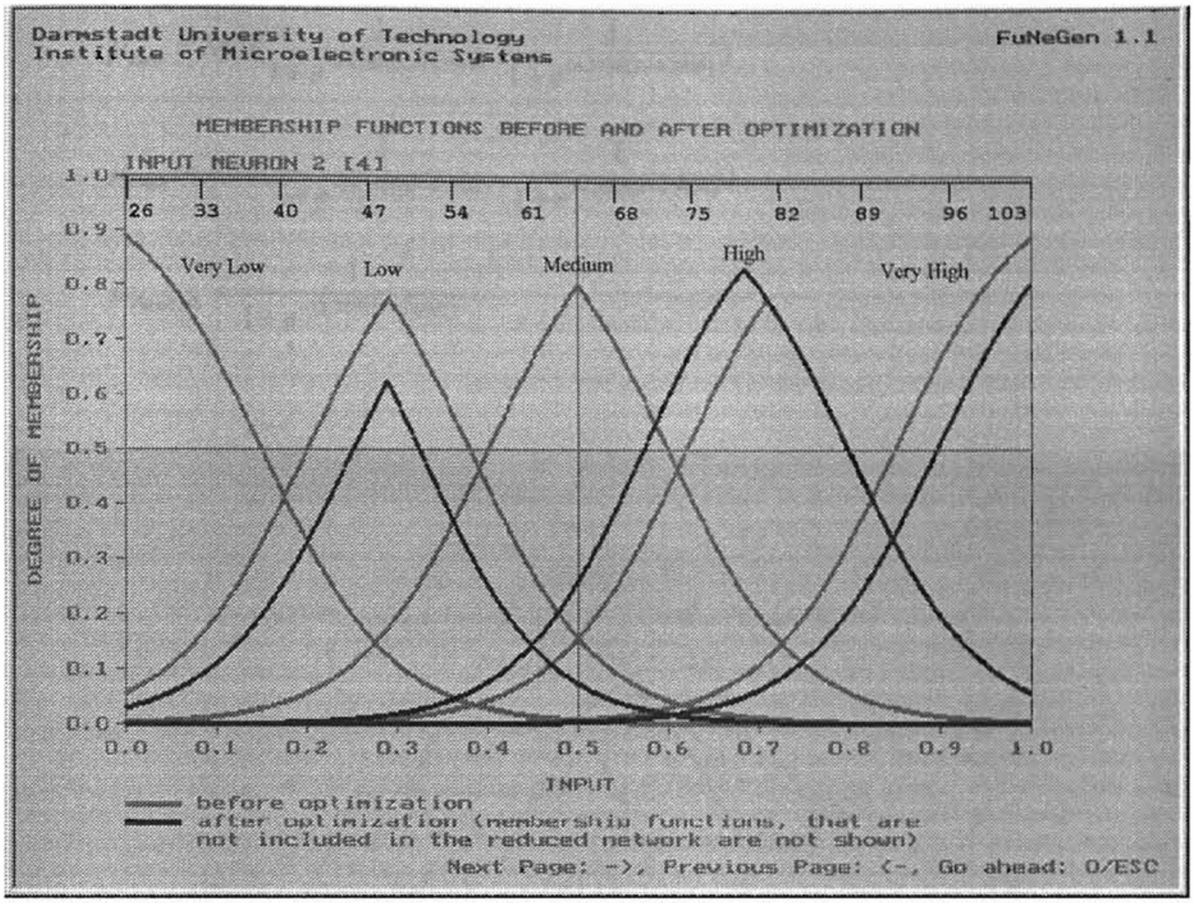


Figure 9: Membership functions vs High Temperatures

In Figure 9, the number of membership functions is reduced from five to three (Low, High, and Very High, see the dark curves) after optimization. The degree of membership for Low High-Temperature, the left curve, is lowered, which decreases the effect to the water flow when the high temperature is low. According to our input data, High-Temperature ranges from 103 °F to 26 °F. Therefore, the Low High-Temperature is around 49 °F ((103-26) * 0.3 + 26), and the High High-Temperature is around 78 °F ((103-26) * 0.68 + 26). The membership functions for High-Temperature are listed below.

$$\mu_{Low} = Min(\frac{1}{1+e^{-13.7816(I-0.2507)}}, \frac{1}{1+e^{14.3758(I-0.3245)}})$$

$$\mu_{High} = Min(\frac{1}{1+e^{-14.3732(I-0.5723)}}, \frac{1}{1+e^{14.0233(I-0.7994)}})$$

$$\mu_{VeryHigh} = \frac{1}{1+e^{-13.7773(I-0.8961)}}$$

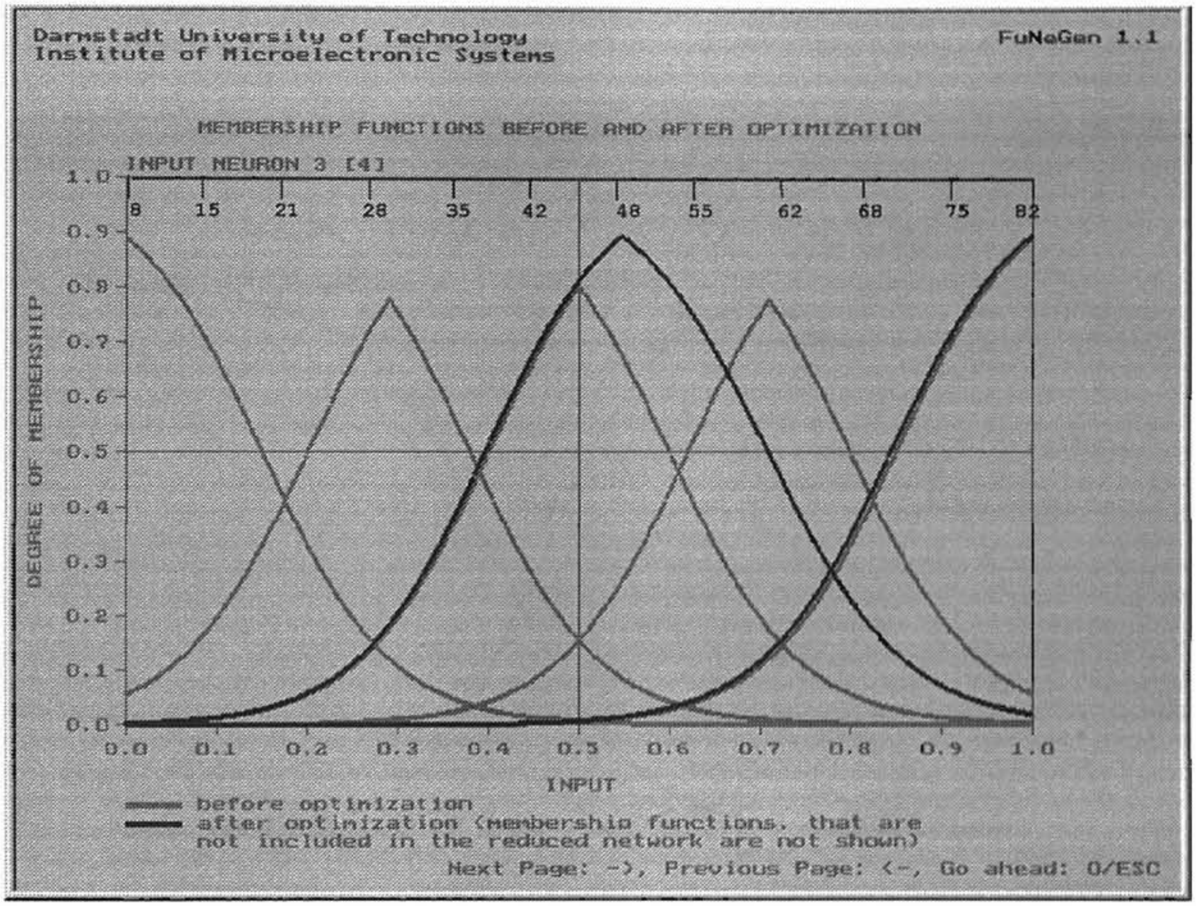


Figure 10: Membership functions vs Low Temperatures

In Figure 10, the number of membership functions is reduced to two (Medium and Very High) after optimization. The highest and lowest Low-Temperatures are 82 °F and 8 °F respectively, according to our data. The Medium Low-Temperature is around 50 °F and the highest degree of membership is about 0.9 by observation of the graph above or from the membership functions listed below.

$$\mu_{Medium} = Min(\frac{1}{1+e^{-14.0774(I-0.3953)}}, \frac{1}{1+e^{13.2717(I-0.7091)}})$$

$$\mu_{VeryHigh} = \frac{1}{1+e^{-13.8988(I-0.8439)}}$$

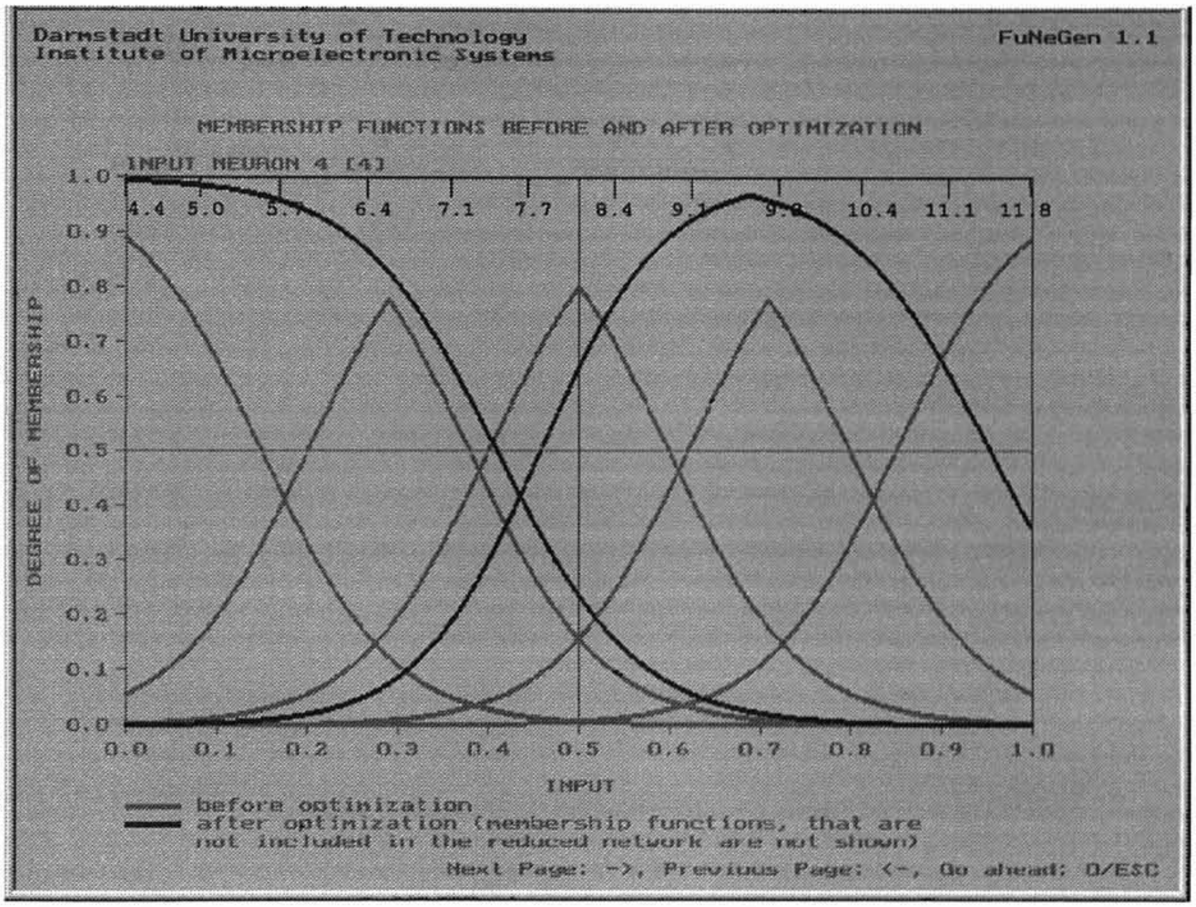


Figure 11: Membership functions Vs Previous Flows

In Figure 11, just like the previous one, the number of membership functions is reduced to two, Very Low and High. Figure 11 shows that the highest degree of membership is 1, full scale, for Very Low Previous-Flow, and close to 1 for High. This fact gives larger rule strength for input Previous Flow. According to the input data, the highest and lowest flow values are 11.803 and 4.356 respectively. The largest degree for High Previous-Flow is when the previous flow is around 9.5. We also find that the High membership function covers a wider area. That gives previous flow more influence on the rule strength. Again, the membership functions are listed below.

$$\mu_{VeryLow} = \frac{1}{1+e^{-13.0589(I-0.4101)}}$$

$$\mu_{High} = Min(\frac{1}{1+e^{-14.9231(I-0.4571)}}, \frac{1}{1+e^{12.9411(I-0.9545)}})$$

3.2.3 The Trained Neural Network

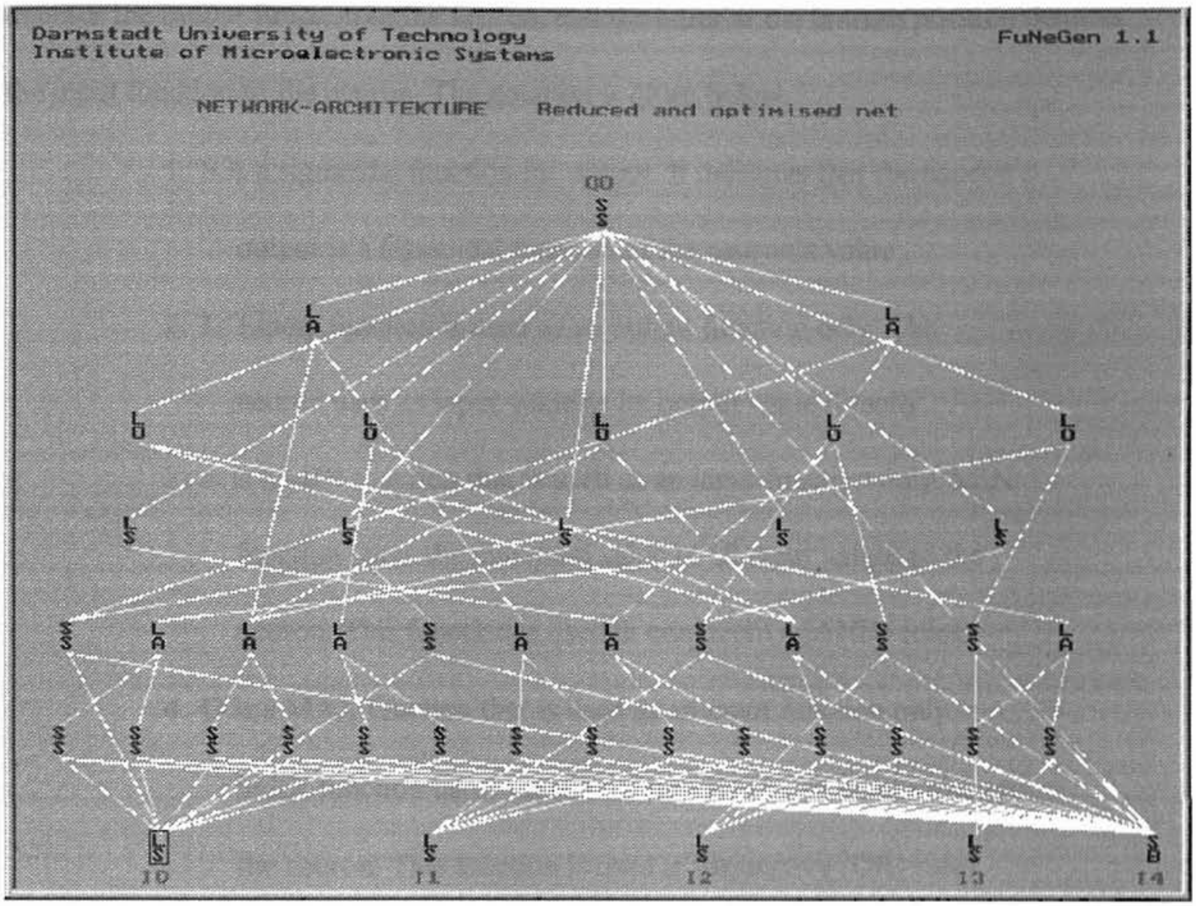


Figure 12: The trained neural network representing the fuzzy system

The diagram shown above is the trained and optimized neural network that represents the fuzzy system we have described. Excluding node I4, the network has 43 neurons spread over 7 layers. Notice that the right-most node, I4, in the bottom layer is not really an input neuron. In fact, node I4 stores a fixed value 1.0 and has edges (weights) to every neuron that use sigmoidal function as its output function. The edges from node I4 to those neurons store weights that actually are the shift factors, α values, of the sigmoidal functions.

In Figure 12 and 13, two capital letters among S, L, A, and O written in vertical are used to denote each neuron's input and output functions. The letter at the top position denotes the output function of the neuron, and the letter at the bottom position denotes the input function to the neuron. The notation is given below.

1. **S** is a sigmoidal function for output. It indicates that the neuron's output is a sigmoidal function of the neuron's value.
2. **L**: Linear function is used as an output function only. The neuron uses its input value as its output value directly.
3. **A** is a MIN function that is used as an input function only. MIN function takes the minimum value of all input values to the neuron. This function is used in conjunctive (AND) rules.
4. **O** is a MAX function that is used as an input function only. MAX function takes the maximum value of all input values to the neuron. This function is used in disjunctive (OR) rules.

We need to number the network's neurons before illustrating how the trained neural network represents the fuzzy system. The numbering counts from bottom layer to top layer and from left to right. For example, n(i,j) denotes the neuron positioned in layer i and node j. Hence, n(0,0) denotes the bottom leftmost neuron in the network. Figure 13 shows the resulting network.

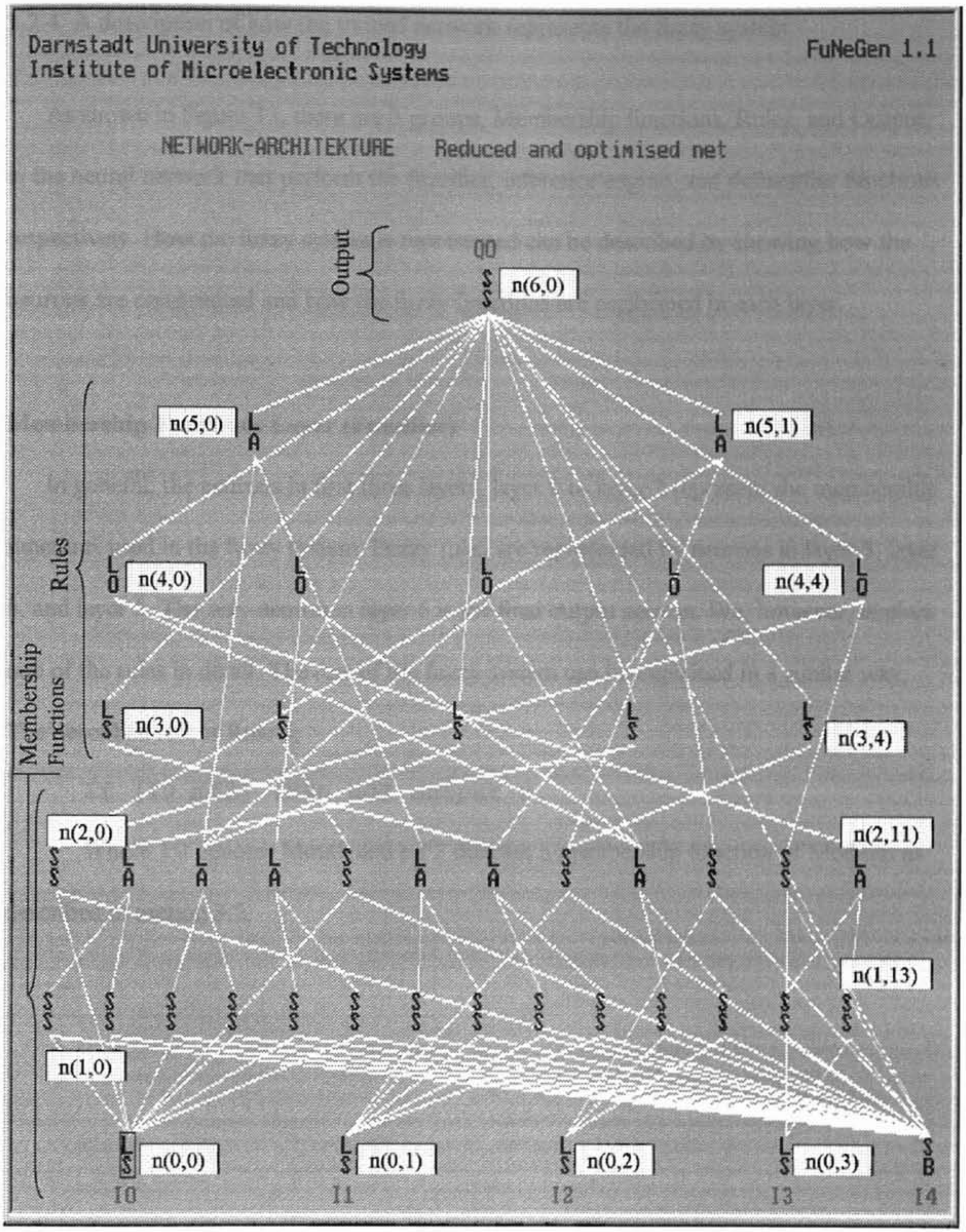


Figure 13: The trained neural network with neurons numbered and categorized

3.2.4 A description of how the trained network represents the fuzzy system

As shown in Figure 13, there are 3 groups, Membership functions, Rules, and Output, in the neural network that perform the fuzzifier, inference engine, and defuzzifier functions respectively. How the fuzzy system is represented can be described by showing how the neurons are constructed and how the fuzzy functions are performed in each layer.

Membership Functions Layer (Fuzzifier)

In general, the neurons in first three layers, layer 0 to layer 2 represent the membership functions used in the fuzzy system. Fuzzy rules are represented by neurons in layer 3, layer 4, and layer 5. The only neuron in layer 6 is the final output neuron. We, however, explain one of the rules in detail. The rest of the fuzzy system can be explained in a similar way. The described rule is Rule 4,

```
If (I0 mf2) THEN NOT output 0.
```

Where `I0` denotes Month and `mf2` denotes a membership function of Medium as described in section 3.2.

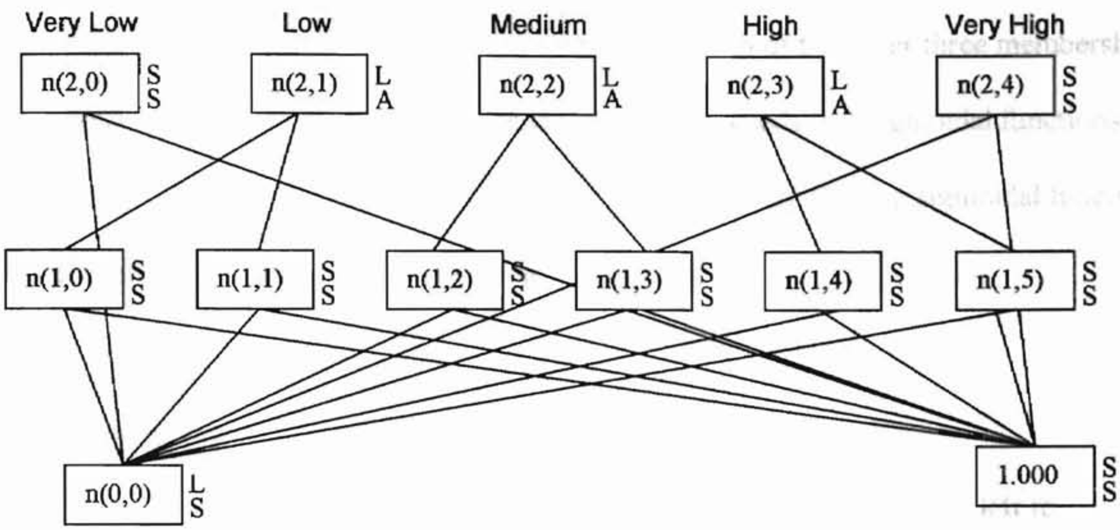


Figure 14: Neurons represent membership functions for Month (I0)

Recall that there are five membership functions used for input of months. The network has 11 neurons to implement the membership function. Very High and Very Low membership functions need only one neuron, respectively. The Low, Medium, and High membership functions need 3 neurons for each. The following table shows the neuron(s) used for each membership function for input Month.

Membership	Neuron(s)
Very Low	n(2,0)
Low	n(1,0), n(1,1), n(2,1)
Medium	n(1,2), n(1,3), n(2,2)
High	n(1,4), n(1,5), n(2,3)
Very High	n(2,4)

Very low and Very High membership functions need only one sigmoidal function each to determine the degree of membership for the input. Each of the other three membership functions (Low, Medium, and High), on the other hand, needs two sigmoidal functions and a MIN function. Therefore, two neurons are used to store the two sigmoidal function outputs and one neuron to store the minimum value of them.

Rules Layer (Inference engine)

Excluding the first 3 layers and the output neuron, there are 12 neurons left to represent the 12 fuzzy rules in our target fuzzy system. Each of the rules has either one (for simple rules) or two inputs (for conjunctive and disjunctive rules). For simple rules, the rule neurons use the degree of membership of its input neuron as the output, the rule strength. For conjunctive (AND) rules, the rule neurons use MIN input function to choose the minimum degree of membership from its input neurons and then use the value as the rule strength, RS. For disjunctive (OR) rules, the rule neurons use MAX input function to choose the maximum degree of membership from its input neurons and use the values as the rule strength, RS.

In our example, neuron n(3,2) is used to implement rule 4,

```
If (I0 mf2) THEN NOT output 0 (or output = RS * -0.561)
```

The following figure shows how it starts from `I0` to the output neuron.

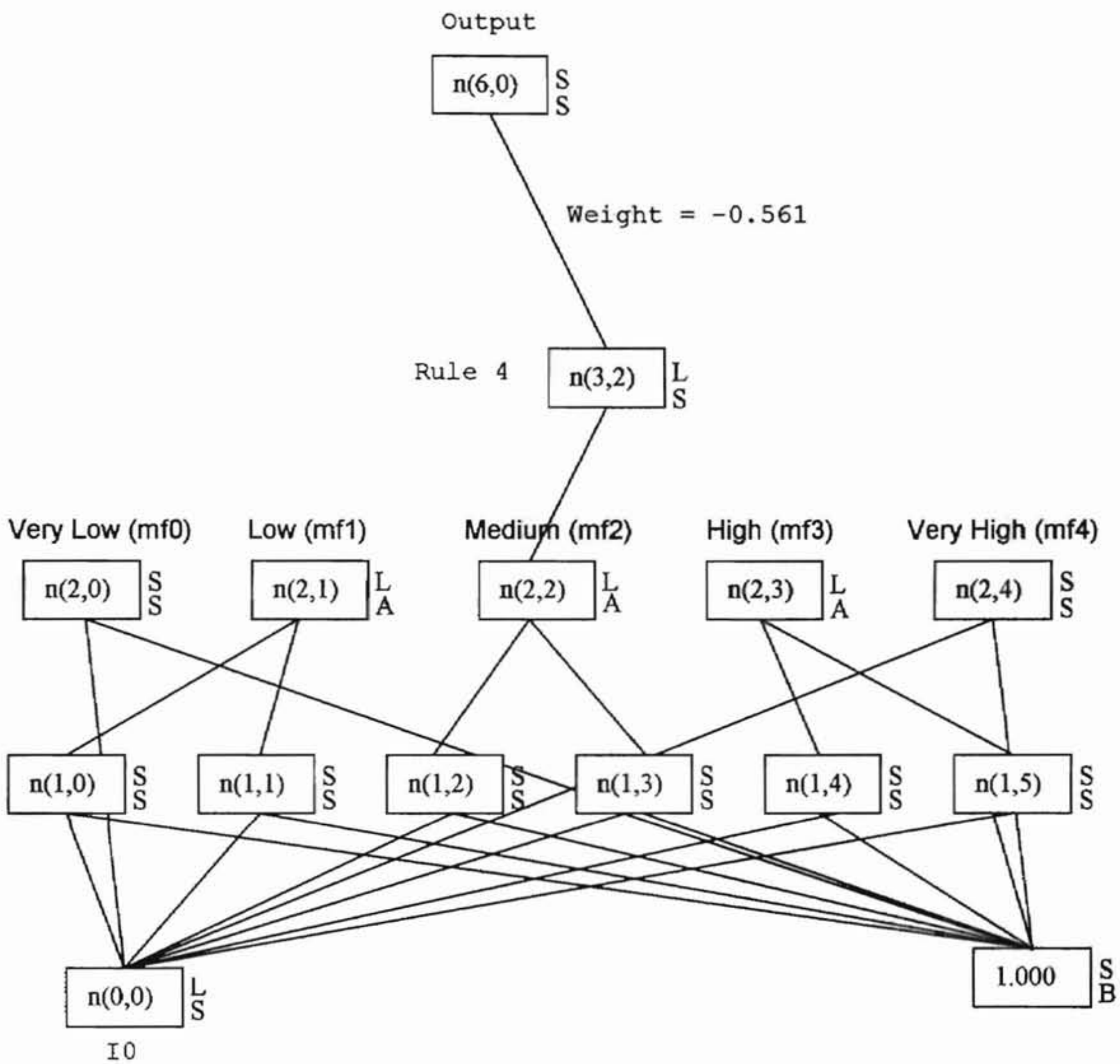


Figure 15 The diagram shows how the `I0` contribute to Rule 4 and the Rule 4 contribute to Output.

Since rule 4 is a simple rule, neuron n(3,2) has one input neuron only, node n(2,2), which represents membership function of medium for input 0.

Similarly, Neuron n(5,0) can be used as an example for conjunctive rule since its input function is a MIN (denoted as A) function. Neuron n(5,0) has two input neurons, `I0 mf2` and `i1 mf1`, and it is used to implement rule 3,

`If (I0 mf2) AND (i1 mf1) THEN output0.`

The understanding of the network architecture is the essence to implement a water demand forecasting system.

Output Layer (Defuzzifier)

The fuzzy system uses a different defuzzification scheme from conventional fuzzy systems. A weight value is assigned to each rule to determine how much the rule will affect the actual output. In the system, the actual output, neuron n(6,0), is the summation of rule strength times its assigned weight, where the summation is calculated over all inputs.

3.3 Test Results

The trained network is tested with a test file consisting of 91 known observations. According to the output generated from FuNeGen, the standard deviation of the error is 0.9697. The error is defined as the difference of actual value and target value, where the resulting value is the value generated by the fuzzy system and the target value is the observed value. Figure 16 shows the testing results. A list of the testing results is also attached in Appendix A.

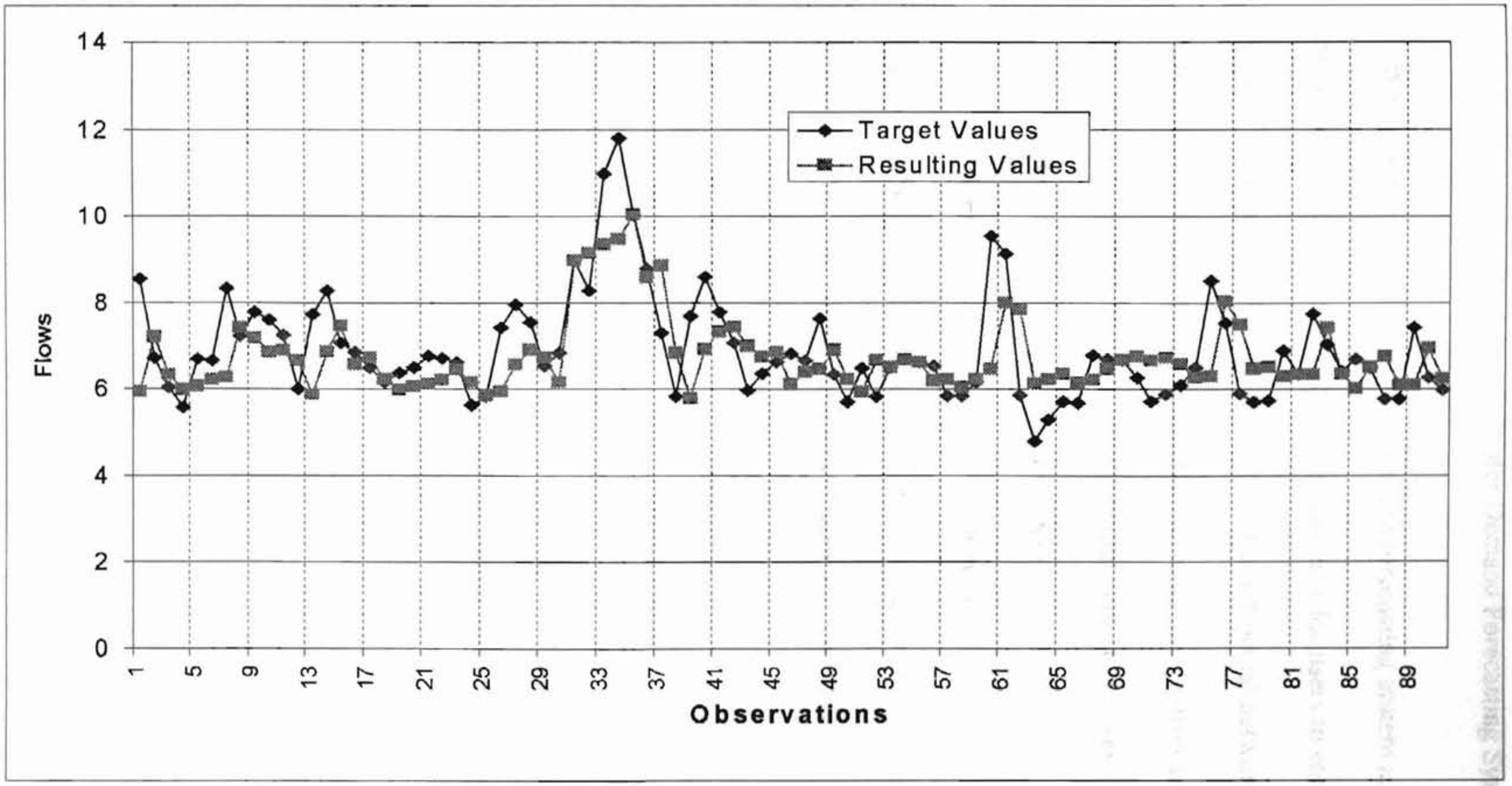


Figure 16: The testing results of the fuzzy system

3.4 A Simple Water Demand Forecasting System

We implemented a simple water demand forecasting system in Visual Basic 5.0 (VB5). The program includes an interactive user interface for users to enter data, as shown in Figure 17. The entered data are fed into the trained neural network. Then the interface displays the calculated result(s) including values in each neuron and the predicted water flow. The code implementing the trained neural network is a rewritten VB5 version. The original version comes from the C code generated by FuNeGen.

The implementation also includes a batch evaluation function to evaluate a set of test values, as shown in Figure 18. The function is used to test the performance of modified neural networks. That is, when the neural network is modified to reflect our thoughts, the function is provided for us to test it.

For future study of the water demand forecasting problem, the implementation also shows the neural network, the neurons' associated membership functions, rules, and the stored value in each neuron. In addition, the edges are grouped by colors for easier observation of the results of membership functions and rule strengths.

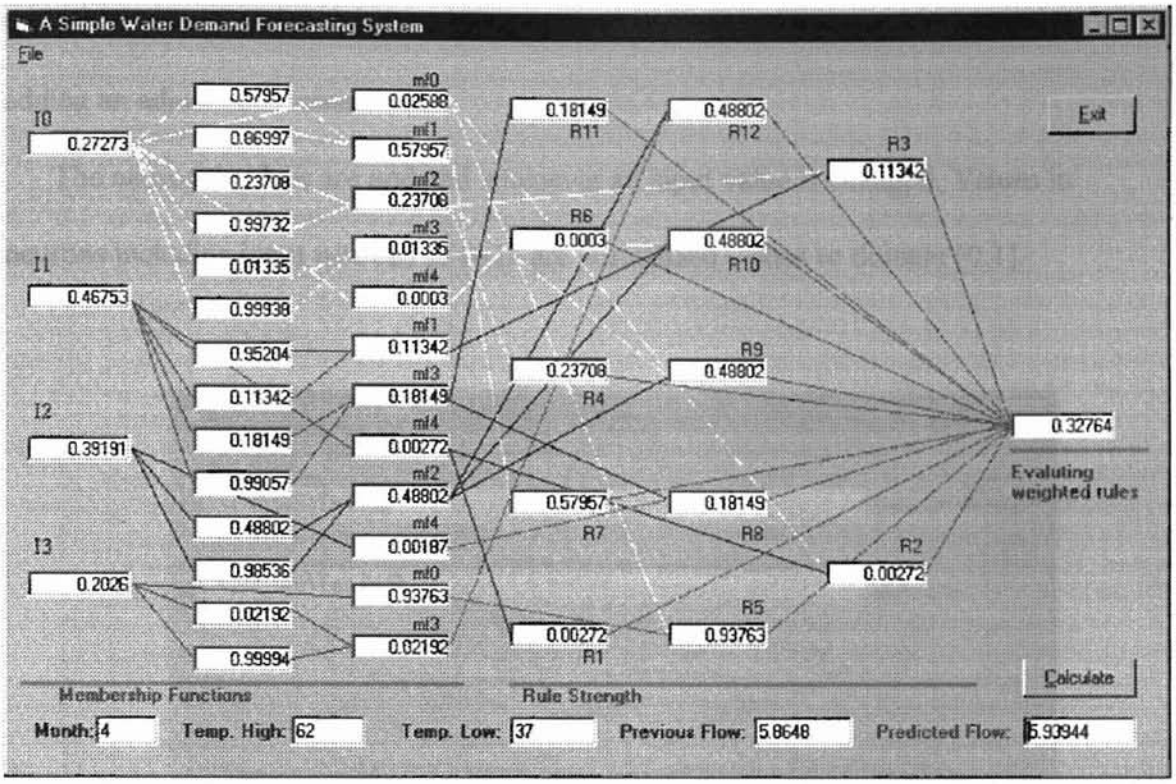


Figure 17: An example of the simple water demand forecasting system

The source VB5 code is listed in Appendix B. Please note that the last line in the function NN_Recall has been modified. The code generated by the FuNeGen seems not de-scale the output neuron right. The original and modified code is listed below.

Original: `Yout = Xout(42) * (12.411668) - (6.838333)`

Corrected: `Yout = Xout(42) * (12.411668) - (6.838333) + 8.71122`

The adjustment value is obtained by evaluating the differences between values calculated by the original code and values shown in FuNeGen's output file. Fortunately, the results show consistent differences in all comparisons. Therefore, we can correct it by adding an adjustment value.

The neurons' values are updated whenever an input value is changed. Values in neurons including input neurons I0 to I3 are normalized values to domain [0,1].

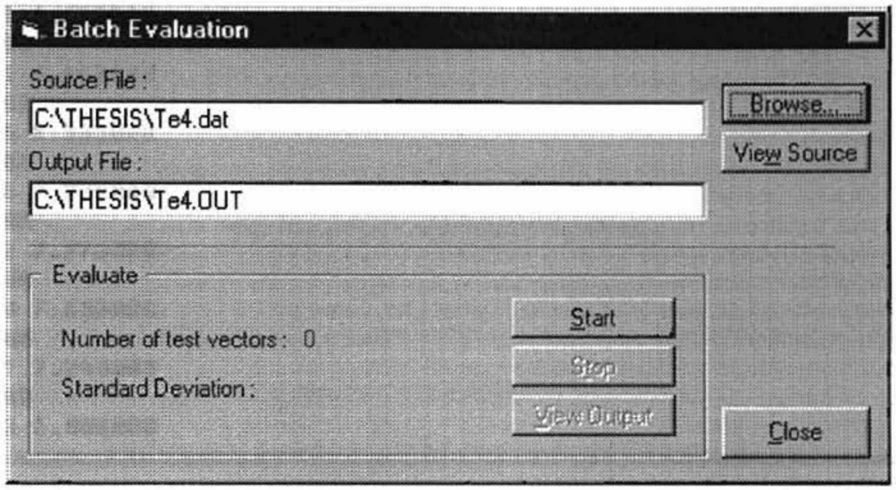
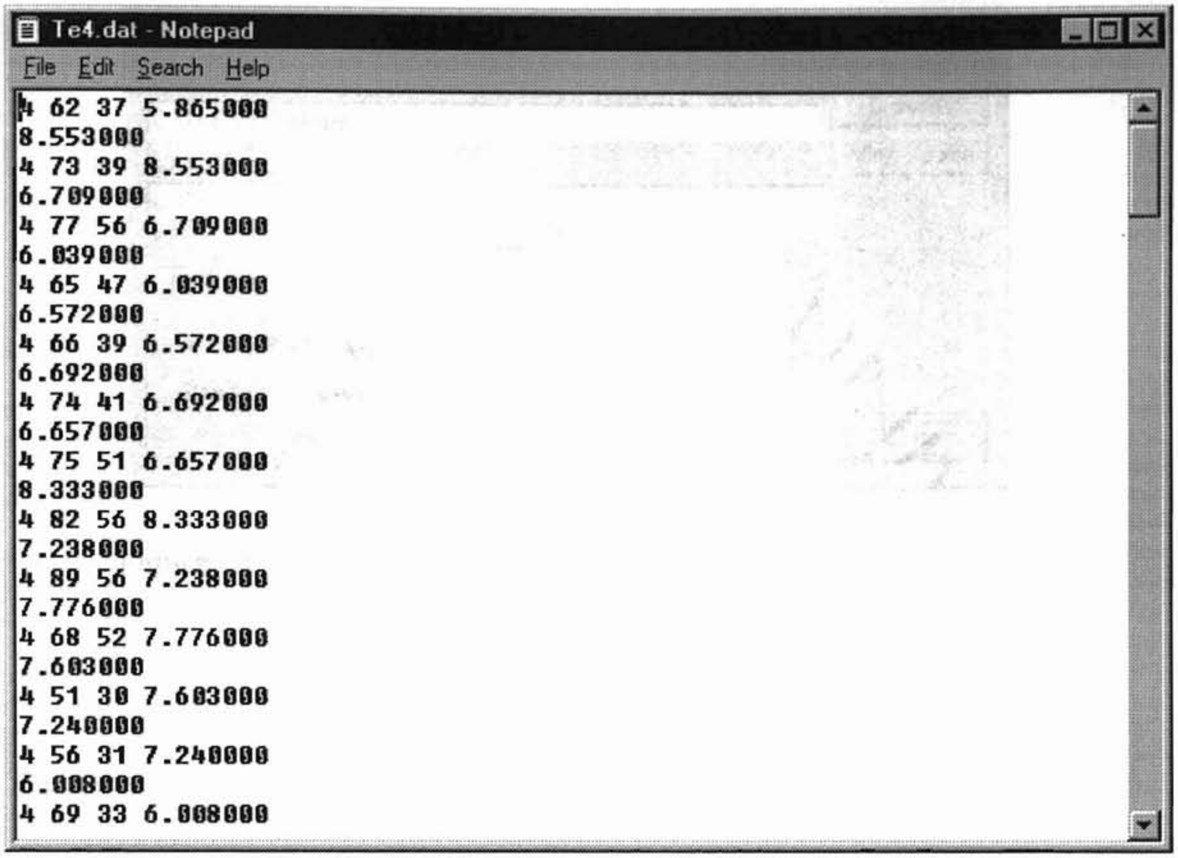


Figure 18 The Batch Evaluation window let user specify a test file and the desired output file.

The batch evaluation function can be invoked from menu item [File]/[Batch Evaluate]. The window shown in Figure 18 lets users specify the source file they want to batch test and the output file to store the resulting values of each evaluation. The source file is a collection of test vectors whose format is the same as the test file described in section 3.1. The Browse button lets users easily specify the file by using a dialog window. The View Source button lets users check the content of the source file they specified, as shown in Figure 19.



Te4.dat - Notepad
File Edit Search Help

```
4 62 37 5.865000
8.553000
4 73 39 8.553000
6.709000
4 77 56 6.709000
6.039000
4 65 47 6.039000
6.572000
4 66 39 6.572000
6.692000
4 74 41 6.692000
6.657000
4 75 51 6.657000
8.333000
4 82 56 8.333000
7.238000
4 89 56 7.238000
7.776000
4 68 52 7.776000
7.603000
4 51 30 7.603000
7.240000
4 56 31 7.240000
6.008000
4 69 33 6.008000
```

Figure 19 The window shows the content of the source file for testing.

The Start button is enabled when the Source File is valid. Default Output File name will be assigned automatically according to the Source File field. A warning will pop up if there is already a file there. When the evaluation is complete, the number of test vector and the standard deviation of resulting values will be shown, as shown in Figure 20.

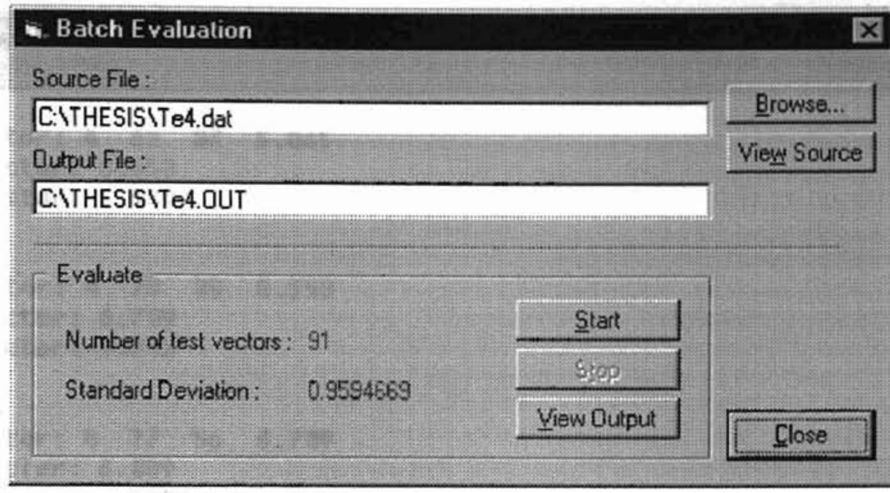


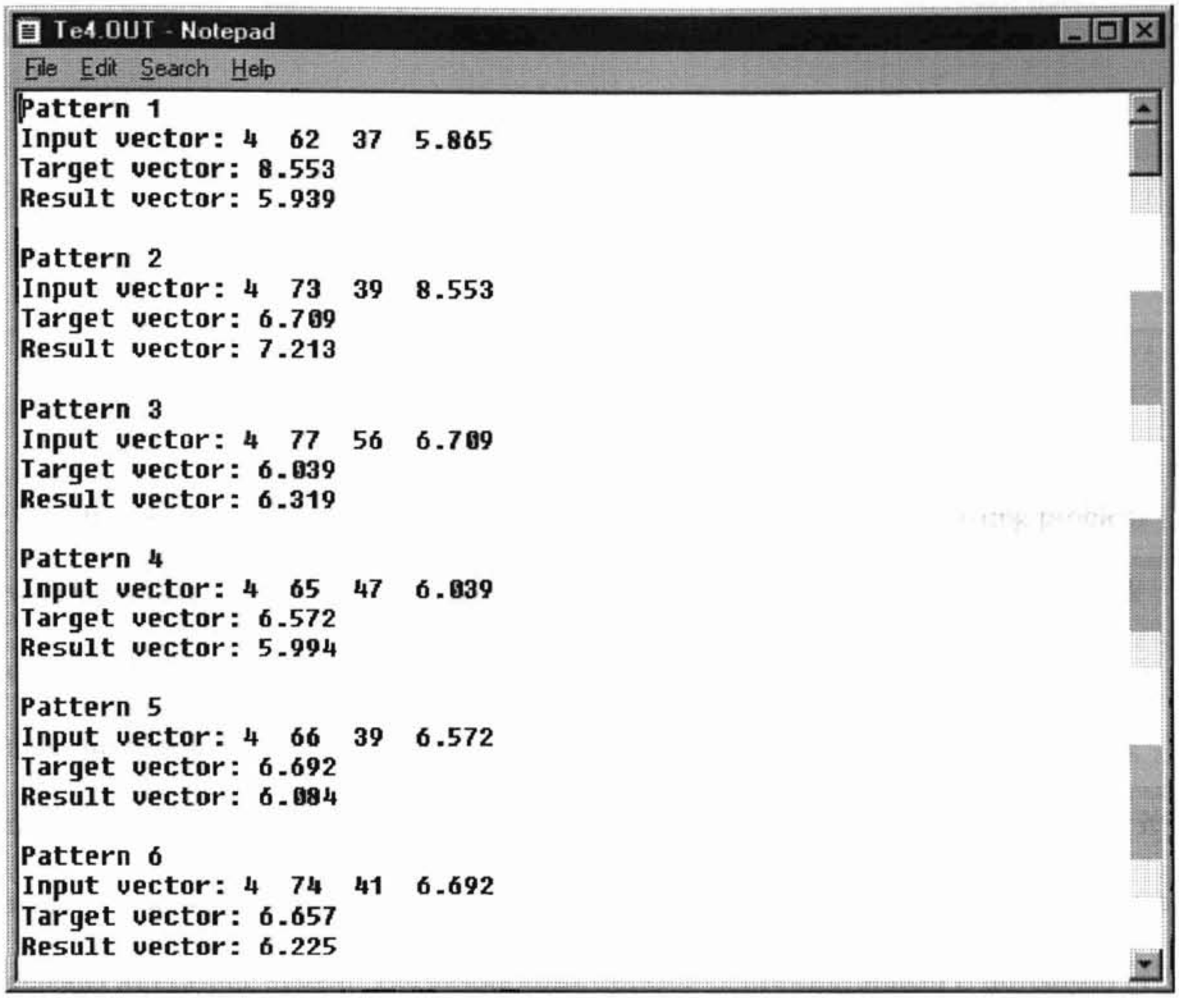
Figure 20 The window shows number of vectors evaluated and the resulting standard deviation.

The standard deviation is evaluated by the following formula.

$$StdDev = \sqrt{\frac{n\sum x^2 - (\sum x)^2}{n(n-1)}}$$

Where n is the number of test vectors, and x is the differences between target (the observed) and resulting values.

The View Output button will be enabled if the evaluation is good. Users may use this function to view the evaluation results of each test, the number of test vectors, and the standard deviation of the batch evaluation, as shown in Figure 21.



Te4.OUT - Notepad

File Edit Search Help

```
Pattern 1
Input vector: 4  62  37  5.865
Target vector: 8.553
Result vector: 5.939

Pattern 2
Input vector: 4  73  39  8.553
Target vector: 6.709
Result vector: 7.213

Pattern 3
Input vector: 4  77  56  6.709
Target vector: 6.039
Result vector: 6.319

Pattern 4
Input vector: 4  65  47  6.039
Target vector: 6.572
Result vector: 5.994

Pattern 5
Input vector: 4  66  39  6.572
Target vector: 6.692
Result vector: 6.084

Pattern 6
Input vector: 4  74  41  6.692
Target vector: 6.657
Result vector: 6.225
```

Figure 21 The View Output function let user view the test results.

CHAPTER IV

CONCLUSIONS AND DISCUSSIONS

This research presents an approach to modeling a water demand forecasting problem with neurofuzzy technique. The data is trained by a specially constructed neural network designed by Halgamuge. The trained neural network represents a fuzzy system, or neurofuzzy network, which we may use to observe the knowledge learned, or extracted, from the training data sets. We explain the structure of the trained neural network, how it represents a fuzzy system, and use the trained neural network to implement a simple water demand forecasting system. The simple water demand forecasting system is developed in Visual Basic 5.0 and can be run under Windows 95 or Windows NT. The program also provides a batch test function that can be used to evaluate the neural network's performance.

The results in Figure 15 have shown the neural network's abilities to model the water demand forecasting problem. In addition, the interpreted fuzzy system also demonstrates that it is possible to extract reasonable knowledge from sets of training data. The results, however, are not perfect because of the nature of complexity of the water demand problem. The training data we have at this time are not sufficient to build a perfect forecasting system. For example, daily population should be an important factor to water demand, but it is almost impossible to gather the daily population data.

Furthermore, it is possible to improve the performance of the trained network with fuzzy logic approach. That is, we may observe and modify the fuzzy system according to our understanding in the water demand problem. Unfortunately, we are not able to improve the performance by simply removing rules. The network should be retrained whenever the structure is altered to get optimized performance.

As the problems are met in modeling with neural network, the parameter setting is another important factor in which the designer has to take care, especially the number of rules setting. Currently there is no convenient and precise approach to determine the number of rules a fuzzy system should has, unless the problem has been well studied. In this research, we set the number of rules by experiments. We think the problem can be a major topic for future works.

REFERENCES

A. H. Al-Momani. *Model of Future Water Demand and Supply in Jordan - A Tool for Planning*. Ph. D. Dissertation, Oklahoma State University. 1988.

C. V. Altrock. *Fuzzy Logic & NeuroFuzzy Applications in Business & Finance*. Prentice Hall, NJ, 1997.

D. E. Rumelhart and J . L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA, 1986.

D. Plaut, S. Nowlan and G. Hinton. *Report CMU-CS-86-126: Experiments on Learning in Back Propagation*. Carnegie-Mellon University, 1986.

F. S. Wong, P. Z. Wang and T. H. Goh. Fuzzy Neural System for Decision Making. *International Joint on neural networks (IJCNN). Singapore 1991*. Vol. 2, pp. 1625-1637, 1991.

H. R. Berenji and P. Khedkar. Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. *IEEE Transactions on neural networks*, Vol. 3, No. 5, pp. 724-740, 1992.

H. Y. Xu, G. Z. Wang and C. B. Baird. A fuzzy neural networks Technique with Fast Back Propagation Learning. *International Joint Conference on neural networks*. June 1992.

L. H. Tsoukalas and E. N. Houstis. Neurofuzzy Motion Planners for Intelligent Robots, *Journal of Intelligent and Robotic Systems*, pp. 339-356, Vol. 19, 1997.

L. T. Wu. *F3MCNN : Fuzzy Minimum Mean Maximum Clustering neural network*. Ph. D. Dissertation, Oklahoma State University, 1995.

M. H. Lim, B. H. Gwee and T. H. Goh. Cause Associator Network for Fuzzily Deduced Conclusions in Process Control. *International Joint on neural networks (IJCNN). Singapore 1991*. Vol. 2, pp. 1248-1253, 1991.

N. A. Malik. *Application of Fuzzy Logic To Distress Analysis*. Thesis, Oklahoma State University, 1994.

R. A. Jacobs. Increased Rates of Convergence Through Learning Rate Adaptation. *IEEE Transactions on neural networks*, 1988.

S. K. Halgamuge and M. Glesner. Fuzzy Controlled Adaptive Gradient Descent Learning Methods. *European Congress on Fuzzy and Intelligent Technologies' 94*. Aachen, Germany, September, 1994.

S. K. Halgamuge and M. Glesner. Fuzzy Neural Fusion Techniques for Industrial Applications. *ACM Symposium on Applied Computing (SAC'94)*. Phoenix, March 1994.

S. K. Halgamuge and M. Glesner. neural networks in Designing Fuzzy Systems for Real World Applications. *Fuzzy Sets and Systems*, Vol. 65, No. 1, pp. 1-12, North Holland, 1994.

S. K. Halgamuge, A. Mari and M. Glesner. Fast Perceptron Learning by Fuzzy Controlled Dynamic Adaptation of Neural Parameters. ftp://obelix.microelectronic.e-technik.th-darmstadt.de/pub/neurofuzzy/saman.tar.Z , 1995

S. K. Halgamuge, W. Poechmueller, and M. Glesner. A Rule Based Prototype System for

Automatic Classification in Industrial Quality Control. *IEEE International Conference on neural networks' 93*. pp. 238-243. San Francisco, March 1993.

S. R. K. Kavuturu. *Water Demand Forecasting and Comparison of neural network Models*. Thesis, Oklahoma State University, 1993.

S. Tzafestas, S. Raptis and G. Stamou. A Flexible Neurofuzzy Cell Structure for General Fuzzy Inference, *Mathematics and Computer in Simulation*, pp. 219-233, Vol. 41 N3-4, July 1996.

S. Y. Kung. *Digital neural networks*. PTR Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

V. B. Rao and H. V. Rao. *C++ neural networks & Fuzzy Logic 2nd*. MIS:Press, New York, 1995.

Z. Q. Wu and C. J. Harris. A Neurofuzzy Network Structure for Modeling and State Estimation of Unknown Nonlinear Systems. *International Journal of Systems Science*, pp 335-345, Vol. 28, No. 4, 1997.

APPENDIX A

Testing Results

```
Pattern 1
Input vector:     3.9997  61.9975  37.0006   5.8648
Target vector:     8.5536
Actual vector:     5.9397

Pattern 2
Input vector:     3.9997  73.0008  38.9986   8.5531
Target vector:     6.7093
Actual vector:     7.2144

Pattern 3
Input vector:     3.9997  76.9971  55.9964   6.7093
Target vector:     6.0390
Actual vector:     6.3195

Pattern 4
Input vector:     3.9997  65.0005  46.9980   6.0390
Target vector:     6.5715
Actual vector:     5.9956

Pattern 5
Input vector:     3.9997  66.0015  38.9986   6.5722
Target vector:     6.6919
Actual vector:     6.0849

Pattern 6
Input vector:     3.9997  74.0018  40.9966   6.6921
Target vector:     6.6571
Actual vector:     6.2264

Pattern 7
Input vector:     3.9997  75.0028  51.0014   6.6571
Target vector:     8.3327
Actual vector:     6.2649

Pattern 8
Input vector:     3.9997  82.0021  55.9964   8.3327
Target vector:     7.2380
Actual vector:     7.4180

Pattern 9
Input vector:     3.9997  89.0014  55.9964   7.2380
Target vector:     7.7754
Actual vector:     7.1809

Pattern 10
Input vector:     3.9997  68.0035  52.0004   7.7757
Target vector:     7.6029
Actual vector:     6.8607

Pattern 11
Input vector:     3.9997  51.0019  30.0002   7.6029
Target vector:     7.2405
Actual vector:     6.8917

Pattern 12
Input vector:     3.9997  55.9992  30.9992   7.2402
Target vector:     6.0080
Actual vector:     6.6596

Pattern 13
Input vector:     3.9997  68.9968  32.9972   6.0077
Target vector:     7.7307
Actual vector:     5.8814

Pattern 14
Input vector:     3.9997  76.0038  36.0016   7.7310
Target vector:     8.2843
Actual vector:     6.8681

Pattern 15
Input vector:     3.9997  83.0031  52.9994   8.2843
Target vector:     7.0754
Actual vector:     7.4602
```

```
Pattern 16
Input vector:     3.9997  77.9981  52.9994   7.0749
Target vector:     6.8582
Actual vector:     6.5789

Pattern 17
Input vector:     3.9997  81.0011  54.9974   6.8582
Target vector:     6.7304
Actual vector:     6.5094

Pattern 18
Input vector:     3.9997  69.9978  51.0014   6.7301
Target vector:     6.1656
Actual vector:     6.2289

Pattern 19
Input vector:     3.9997  67.0025  42.0030   6.1649
Target vector:     6.3865
Actual vector:     5.9844

Pattern 20
Input vector:     3.9997  63.9995  44.0010   6.3861
Target vector:     6.5045
Actual vector:     6.0775

Pattern 21
Input vector:     3.9997  69.9978  45.0000   6.5052
Target vector:     6.7614
Actual vector:     6.1271

Pattern 22
Input vector:     3.9997  67.0025  46.9980   6.7606
Target vector:     6.7117
Actual vector:     6.2264

Pattern 23
Input vector:     3.9997  48.9999  44.0010   6.7122
Target vector:     6.6311
Actual vector:     6.4449

Pattern 24
Input vector:     3.9997  60.9965  36.0016   6.6311
Target vector:     5.6294
Actual vector:     6.1706

Pattern 25
Input vector:     3.9997  68.9968  35.0026   5.6302
Target vector:     5.8305
Actual vector:     5.8454

Pattern 26
Input vector:     3.9997  74.0018  60.9988   5.8313
Target vector:     7.4192
Actual vector:     5.9459

Pattern 27
Input vector:     3.9997  71.9998  35.0026   7.4190
Target vector:     7.9603
Actual vector:     6.5727

Pattern 28
Input vector:     3.9997  71.9998  35.0026   7.9603
Target vector:     7.5595
Actual vector:     6.9066

Pattern 29
Input vector:     3.9997  68.9968  49.0034   7.5590
Target vector:     6.5454
Actual vector:     6.7403

Pattern 30
Input vector:     3.9997  73.0008  54.9974   6.5462
Target vector:     6.8321
Actual vector:     6.1631
```

```
Pattern 31
Input vector:    9.0003  97.0017  66.0012   8.5911
Target vector:   8.9955
Actual vector:   8.9905

Pattern 32
Input vector:    9.0003  97.0017  67.9992   8.9947
Target vector:   8.2719
Actual vector:   9.1419

Pattern 33
Input vector:    9.0003  99.0037  69.9972   8.2716
Target vector:  10.9826
Actual vector:   9.3380

Pattern 34
Input vector:    9.0003  98.0027  69.9972  10.9823
Target vector:  11.8030
Actual vector:   9.4771

Pattern 35
Input vector:    9.0003 103.0000  67.9992  11.8030
Target vector:  10.0207
Actual vector:  10.0368

Pattern 36
Input vector:    9.0003  93.9987  69.9972  10.0209
Target vector:   8.6070
Actual vector:   8.8006

Pattern 37
Input vector:    9.0003  96.0007  67.0002   8.6067
Target vector:   7.2976
Actual vector:   8.8664

Pattern 38
Input vector:    9.0003  83.9964  59.0008   7.2983
Target vector:   5.8305
Actual vector:   6.8458

Pattern 39
Input vector:    9.0003  67.0025  59.9998   5.8313
Target vector:   7.6960
Actual vector:   5.7970

Pattern 40
Input vector:    9.0003  80.0001  60.9988   7.6960
Target vector:   8.6045
Actual vector:   6.9364

Pattern 41
Input vector:    9.0003  80.0001  64.0032   8.6053
Target vector:   7.7816
Actual vector:   7.3385

Pattern 42
Input vector:    9.0003  81.0011  59.0008   7.7824
Target vector:   7.4366
Actual vector:   7.0729

Pattern 43
Input vector:    9.0003  85.9984  65.0022   7.4368
Target vector:   5.9745
Actual vector:   6.9935

Pattern 44
Input vector:    9.0003  91.0034  66.0012   5.9742
Target vector:   6.3642
Actual vector:   6.7477

Pattern 45
Input vector:    9.0003  91.0034  67.0002   6.3637
Target vector:   6.6323
Actual vector:   6.8495
```

```
Pattern 46
Input vector:     9.0003  74.0018  67.0002   6.6318
Target vector:     6.8160
Actual vector:     6.1160

Pattern 47
Input vector:     9.0003  81.0011  67.0002   6.8157
Target vector:     6.6658
Actual vector:     6.4002

Pattern 48
Input vector:     9.0003  84.9974  67.0002   6.6661
Target vector:     7.6327
Actual vector:     6.4734

Pattern 49
Input vector:     9.0003  81.0011  67.0002   7.6327
Target vector:     6.3506
Actual vector:     6.9128

Pattern 50
Input vector:     9.0003  77.9981  57.0028   6.3511
Target vector:     5.7027
Actual vector:     6.2388

Pattern 51
Input vector:     9.0003  63.9995  46.9980   5.7032
Target vector:     6.4796
Actual vector:     5.9397

Pattern 52
Input vector:     9.0003  47.9989  35.0026   6.4799
Target vector:     5.8305
Actual vector:     6.6683

Pattern 53
Input vector:     9.0003  50.0009  35.0026   5.8313
Target vector:     6.5069
Actual vector:     6.5094

Pattern 54
Input vector:     9.0003  52.9962  40.9966   6.5067
Target vector:     6.6894
Actual vector:     6.6782

Pattern 55
Input vector:     9.0003  54.9982  51.0014   6.6899
Target vector:     6.6236
Actual vector:     6.6311

Pattern 56
Input vector:     9.0003  61.9975  46.9980   6.6229
Target vector:     6.5467
Actual vector:     6.2041

Pattern 57
Input vector:     9.0003  73.0008  46.9980   6.5469
Target vector:     5.8367
Actual vector:     6.2314

Pattern 58
Input vector:     9.0003  74.0018  54.9974   5.8372
Target vector:     5.8367
Actual vector:     6.0465

Pattern 59
Input vector:     9.0003  84.9974  67.9992   5.8372
Target vector:     6.1594
Actual vector:     6.2401

Pattern 60
Input vector:     9.0003  86.9994  72.0026   6.1589
Target vector:     9.5354
Actual vector:     6.4660
```

```
Pattern 61
Input vector:    10.0002  83.0031  49.0034   9.5346
Target vector:    9.1270
Actual vector:    7.9939

Pattern 62
Input vector:    10.0002  80.0001  52.0004   9.1273
Target vector:    5.8591
Actual vector:    7.8499

Pattern 63
Input vector:    10.0002  65.0005  51.0014   5.8588
Target vector:    4.8028
Actual vector:    6.1209

Pattern 64
Input vector:    10.0002  76.0038  45.9990   4.8028
Target vector:    5.2869
Actual vector:    6.2326

Pattern 65
Input vector:    10.0002  78.9991  49.0034   5.2869
Target vector:    5.7225
Actual vector:    6.3630

Pattern 66
Input vector:    10.0002  69.9978  50.0024   5.7218
Target vector:    5.6853
Actual vector:    6.1458

Pattern 67
Input vector:    10.0002  76.0038  37.9996   5.6853
Target vector:    6.7800
Actual vector:    6.2227

Pattern 68
Input vector:    10.0002  71.9998  50.0024   6.7800
Target vector:    6.6906
Actual vector:    6.4908

Pattern 69
Input vector:    10.0002  80.0001  47.9970   6.6906
Target vector:    6.6757
Actual vector:    6.6857

Pattern 70
Input vector:    10.0002  83.0031  47.9970   6.6757
Target vector:    6.2761
Actual vector:    6.7614

Pattern 71
Input vector:    10.0002  84.9974  55.9964   6.2758
Target vector:    5.7213
Actual vector:    6.6571

Pattern 72
Input vector:    10.0002  88.0004  58.0018   5.7210
Target vector:    5.8876
Actual vector:    6.7279

Pattern 73
Input vector:    10.0002  85.9984  59.9998   5.8871
Target vector:    6.0899
Actual vector:    6.5901

Pattern 74
Input vector:    10.0002  75.0028  38.9986   6.0897
Target vector:    6.4995
Actual vector:    6.2686

Pattern 75
Input vector:    10.0002  70.9988  42.0030   6.5000
Target vector:    8.5089
Actual vector:    6.3121
```

```
Pattern 76
Input vector:     10.0002  86.9994  44.0010   8.5092
Target vector:     7.5284
Actual vector:     8.0249

Pattern 77
Input vector:     10.0002  86.9994  45.0000   7.5277
Target vector:     5.9013
Actual vector:     7.5048

Pattern 78
Input vector:     10.0002  83.0031  57.0028   5.9013
Target vector:     5.6965
Actual vector:     6.4759

Pattern 79
Input vector:     10.0002  84.9974  59.0008   5.6972
Target vector:     5.7362
Actual vector:     6.5156

Pattern 80
Input vector:     10.0002  76.9971  43.0020   5.7359
Target vector:     6.8917
Actual vector:     6.3071

Pattern 81
Input vector:     10.0002  68.9968  34.0036   6.8917
Target vector:     6.3394
Actual vector:     6.3443

Pattern 82
Input vector:     10.0002  77.9981  34.0036   6.3399
Target vector:     7.7506
Actual vector:     6.3493

Pattern 83
Input vector:     10.0002  83.0031  53.9984   7.7503
Target vector:     7.0270
Actual vector:     7.4242

Pattern 84
Input vector:     10.0002  61.9975  30.0002   7.0272
Target vector:     6.4188
Actual vector:     6.3580

Pattern 85
Input vector:     10.0002  67.0025  30.0002   6.4188
Target vector:     6.6981
Actual vector:     6.0291

Pattern 86
Input vector:     10.0002  75.0028  45.9990   6.6981
Target vector:     6.5007
Actual vector:     6.5268

Pattern 87
Input vector:     10.0002  84.9974  45.9990   6.5007
Target vector:     5.7784
Actual vector:     6.7663

Pattern 88
Input vector:     10.0002  69.9978  38.9986   5.7776
Target vector:     5.7809
Actual vector:     6.0936

Pattern 89
Input vector:     10.0002  69.9978  38.9986   5.7814
Target vector:     7.4316
Actual vector:     6.0936

Pattern 90
Input vector:     10.0002  74.0018  45.9990   7.4309
Target vector:     6.2748
Actual vector:     6.9488
```

```
Pattern 91
Input vector:     10.0002  60.9965  54.9974    6.2751
Target vector:      5.9956
Actual vector:      6.2612

Results:
Number of test vectors: 91
Standard Deviations:
Output0:      0.9697
```

APPENDIX B

Code Listing

FUNE.VBP

Project Settings

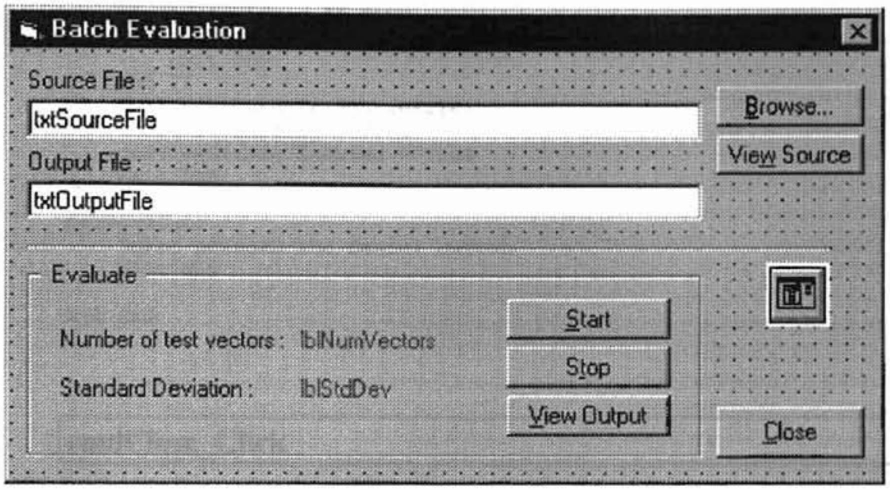
Type	Exe
IconForm	frmFune
Startup	frmFune
ExeName32	fune.exe
Command32	
Name	Project1
HelpContextID	0
CompatibleMode	0
MajorVer	1
MinorVer	0
RevisionVer	0
AutoIncrementVer	0
ServerSupportFiles	0
VersionCompanyName	OSU
CompilationType	0
OptimizationType	0
FavorPentiumPro(tm)	0
CodeViewDebugInfo	0
NoAliasing	0
BoundsCheck	0
OverflowCheck	0
FlPointCheck	0
FDIVCheck	0
UnroundedFP	0
StartMode	0
Unattended	0
ThreadPerObject	0
MaxNumberOfThreads	1

Project References

Reference	OLE Automation
Object	COMDLG32.OCX

BATCH.FRM

Mod Date Sun Feb 22 15:34:32 1998
Size 10706



Declarations

```
Attribute VB_Name = "frmBatchEvaluate"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Dim SourceFile As String
Dim OutputFile As String
Dim NumVectors As Integer
Dim ArraySize As Integer
Dim Diffs() As Single
Dim OutInFocus As Integer
Public StopFlag As Integer
```

Subroutines

cmdBrowseSource_Click

Qualifiers: Private

```
Private Sub cmdBrowseSource_Click()

    Dim Pos As Integer, tmpS As String

    ' Set CancelError is True
    CDialog.CancelError = True
```

```
    On Error GoTo ErrHandler
    ' Set flags
    'CommonDialog1.Flags = cdlOFNHideReadOnly
    ' Set filters
    CDialog.Filter = "All Files (*.*)|*.*|Data Files (*.dat)|*.dat"
    ' Specify default filter
    CDialog.FilterIndex = 2
    ' Display the Open dialog box
    CDialog.ShowOpen
    txtSourceFile = CDialog.FileName
    SourceFile = CDialog.FileTitle
    If SourceFile = "" Then
        cmdViewSource.Enabled = False
        Exit Sub
    End If
    Call MakeOutFile(SourceFile)
    cmdStart.Enabled = True
    cmdViewSource.Enabled = True
    Exit Sub

ErrHandler:
    'User pressed the Cancel button
    Exit Sub

End Sub
```

cmdClose_Click

Qualifiers: Private

```
Private Sub cmdClose_Click()

    Hide

End Sub
```

cmdStart_Click

Qualifiers: Private

```
Private Sub cmdStart_Click()

    Dim ret
    Dim InFileNum As Integer, OutFileNum As Integer
    Dim InFile As String, OutFile As String
    Dim M As Integer, HighTemp As Single
    Dim LowTemp As Single, PrevFlow As Single
    Dim Target As Single, tmpS As String
    Dim R As Integer, SDev

    cmdStart.Enabled = False
    cmdStop.Enabled = True
    cmdViewOutput.Enabled = False
    InFile = txtSourceFile
    OutFile = txtOutputFile
    StopFlag = False
    'check if source file exists
    If Dir(InFile) = "" Then
```

```
        ret = MsgBox("Source file not exist:" & vbCrLf & vbCrLf & InFile,
vbExclamation)
        Exit Sub
    End If
    InFileNum = FreeFile
    'On Error GoTo OpenError
    Open InFile For Input Access Read As #InFileNum
    OutFileNum = FreeFile
    Open OutFile For Output As #OutFileNum
    NumVectors = 0
    Do While Not EOF(InFileNum)
        If StopFlag Then
            cmdStop.Enabled = False
            cmdStart.Enabled = True
            Exit Sub
        End If
        Input #InFileNum, M, HighTemp, LowTemp, PrevFlow
        Input #InFileNum, Target
        NumVectors = NumVectors + 1
        Yin(0) = M: Yin(1) = HighTemp:
        Yin(2) = LowTemp: Yin(3) = PrevFlow
        R = NN_Recall(Yin())
        'check array size
        If NumVectors >= ArraySize Then
            ArraySize = ArraySize + 100
            ReDim Preserve Diffs(0 To ArraySize - 1)
        End If
        Diffs(NumVectors - 1) = Target - Yout
        Print #OutFileNum, "Pattern " & NumVectors
        tmpS = M & "  " & HighTemp & "  " & LowTemp & "  " & PrevFlow
        Print #OutFileNum, "Input vector: " & tmpS
        Print #OutFileNum, "Target vector: " & Target
        Print #OutFileNum, "Result vector: " & Format$(Yout, "#.000")
        Print #OutFileNum,
    Loop
    ReDim Preserve Diffs(0 To NumVectors - 1)
    lblNumVectors = NumVectors
    lblStdDev = StdDev(NumVectors, Diffs())
    cmdStop.Enabled = False
    cmdStart.Enabled = True
    cmdViewOutput.Enabled = True
    Close
    ret = MsgBox("Evaluation completed successfully")
    Exit Sub

OpenError:
    ret = MsgBox("Unable to open source file:" & vbCrLf & vbCrLf & InFile,
vbExclamation)
    Exit Sub

End Sub
```

cmdStop_Click

Qualifiers: Private

```
Private Sub cmdStop_Click()

    StopFlag = True

End Sub
```

cmdViewOutput_Click

Qualifiers: Private

```
Private Sub cmdViewOutput_Click()

    Dim RetVal As Double
    Dim S As String

    S = "notepad.exe " & txtOutputFile
    RetVal = Shell(S, 1)

End Sub
```

cmdViewSource_Click

Qualifiers: Private

```
Private Sub cmdViewSource_Click()

    Dim RetVal As Double
    Dim S As String

    S = "notepad.exe " & txtSourceFile
    RetVal = Shell(S, 1)

End Sub
```

Form_Load

Qualifiers: Private

```
Private Sub Form_Load()

    cmdStart.Enabled = False
    cmdStop.Enabled = False
    cmdViewOutput.Enabled = False
    txtSourceFile = ""
    txtOutputFile = ""
    lblNumVectors = 0
    lblStdDev = ""

    ArraySize = 150
    ReDim Diffs(ArraySize)

End Sub
```

txtOutputFile_Change

Qualifiers: Private

```
Private Sub txtOutputFile_Change()

    Dim tmpS As String, ret

    If OutInFocus Then
        Exit Sub
    End If

    tmpS = txtOutputFile
    If tmpS <> "" Then
        If Dir(tmpS) <> "" Then
            ret = MsgBox("The output file already exist, overwrite it?", _
                vbYesNo + vbInformation)
            If ret = vbYes Then
                cmdStart.Enabled = True
                Exit Sub
            Else
                txtOutputFile = ""
                cmdStart.Enabled = False
            End If
        End If
    End If

End Sub
```

txtOutputFile_GotFocus

Qualifiers: Private

```
Private Sub txtOutputFile_GotFocus()

    OutInFocus = True

End Sub
```

txtOutputFile_LostFocus

Qualifiers: Private

```
Private Sub txtOutputFile_LostFocus()

    OutInFocus = False

End Sub
```

Functions

MakeOutFile

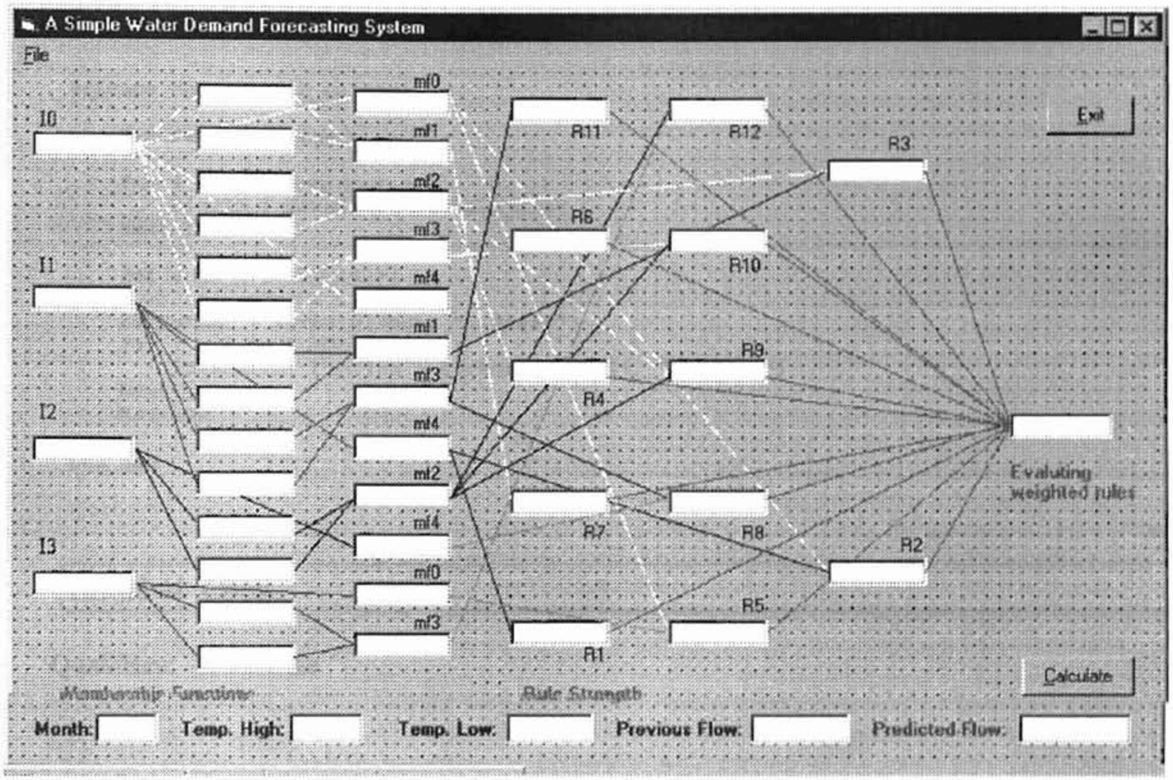
Qualifiers: Private

Arguments:	Source	String	By Ref.
Returns:	Variant		

```
Private Function MakeOutFile(Source As String)

    Dim Pos As Integer, tmpS As String

    Pos = InStr(SourceFile, ".")
    If Pos <> 0 Then
        OutputFile = Left(SourceFile, Pos - 1) & ".OUT"
    Else
        OutputFile = SourceFile & ".OUT"
    End If
    tmpS = Left$(txtSourceFile, Len(txtSourceFile) - Len(SourceFile))
    txtOutputFile = tmpS & OutputFile

End Function
```

FUNE.FRM

Mod Date Sun Feb 22 12:14:42 1998
Size 49726



Declarations

```
Attribute VB_Name = "frmFune"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Dim I As Integer
```

Menu

Caption	Shortcut	Name
&File		mnuFile
&Batch Evaluate		mnuBatchEvalute
		mnuSpe1
&Exit		mnuExit

Subroutines

cmdCalc_Click

Qualifiers: Private

```
Private Sub cmdCalc_Click()

    Dim ret As Integer

    For I = 0 To NumInput - 1
        Yin(I) = CSng(Val(txtInput(I)))
    Next I

    ret = NN_Recall(Yin())

    txtOutput = Format$(Yout, "0.#####")
    For I = 0 To NumNeurons - 1
        lblXout(I).Caption = Format$(Xout(I), "0.#####")
    Next I
    Refresh

End Sub
```

cmdExit_Click

Qualifiers: Private

```
Private Sub cmdExit_Click()

    End

End Sub
```

Form_Load

Qualifiers: Private

```
Private Sub Form_Load()

    For I = 0 To NumNeurons - 1
        lblXout(I).Caption = ""
    Next I
    For I = 0 To NumInput - 1
        txtInput(I) = ""
    Next I
    txtOutput = ""

End Sub
```

mnuBatchEvalute_Click

Qualifiers: Private

```
Private Sub mnuBatchEvalute_Click()

    frmBatchEvaluate.Show

End Sub
```

mnuExit_Click

Qualifiers: Private

```
Private Sub mnuExit_Click()

    End

End Sub
```

txtInput_GotFocus

Qualifiers:	Private		
Arguments:	Index	Integer	By Ref.

```
Private Sub txtInput_GotFocus(Index As Integer)

    txtInput(Index).SelStart = 0
    txtInput(Index).SelLength = Len(txtInput(Index))

End Sub
```

txtInput_LostFocus

Qualifiers:	Private		
Arguments:	Index	Integer	By Ref.

```
Private Sub txtInput_LostFocus(Index As Integer)

    Dim ret As Integer

    For I = 0 To NumInput - 1
        Yin(I) = CSng(Val(txtInput(I)))
    Next I
```

```
    ret = NN_Recall(Yin())

    txtOutput = Format$(Yout, "0.#####")
    For I = 0 To NumNeurons - 1
        lblXout(I).Caption = Format$(Xout(I), "0.#####")
    Next I
    Refresh

End Sub
```

FUNE.BAS

Mod Date	Mon Feb 23 08:35:26 1998
Size	7210

Declarations

```
Attribute VB_Name = "Module1"
Option Explicit
Public Const NumNeurons = 43
Public Const NumInput = 4
Public Xout(0 To NumNeurons - 1) As Double '  ''/* work arrays */
Public Yin(NumInput) As Double
Public Yout As Double
Dim I As Integer
```

Functions

Max

Qualifiers:	Public		
Arguments:	A	Double	By Ref.
	B	Double	By Ref.
Returns:	Double		

```
Function Max(A As Double, B As Double) As Double

    Max = IIf(A > B, A, B)

End Function
```

Min

Qualifiers:	Public		
Arguments:	A	Double	By Ref.
	B	Double	By Ref.
Returns:	Double		

```
Function Min(A As Double, B As Double) As Double

    Min = IIf(A > B, B, A)

End Function
```

StdDev

Qualifiers:	Public		
Arguments:	N	Integer	By Ref.
	P	Single	By Ref.
Returns:	Single		

```
Function StdDev(N As Integer, P() As Single) As Single
'Used to calculate standard deviation of an array
'StdDev = SQR( (n * sigma(x^2) - (sigma(x))^2 )) / (n*(n-1))

    Dim tmp1 As Double, tmp2 As Double
    Dim I As Integer
    '
    tmp1 = 0
    tmp2 = 0
    For I = 0 To N - 1
        tmp1 = tmp1 + P(I) * P(I)
        tmp2 = tmp2 + P(I)
    Next I
    tmp2 = tmp2 * tmp2
    StdDev = Sqr((N * tmp1 - tmp2) / (N * (N - 1)))

End Function
```

NN_Recall

Qualifiers:	Public		
Arguments:	Yin	Double	By Ref.
Returns:	Integer		

```
Public Function NN_Recall(Yin() As Double) As Integer
'The implementation of the trained neural network
'repsenting the fuzzy system, a simple water demand
'forecasting system. The standard deviation of this
'network is 0.959 for original version.

    For I = 0 To NumNeurons - 1
        Xout(I) = 0
    Next I

    '/* Read and scale input into network */

    Xout(0) = Yin(0) * (0.090909) + (-0.090909)
    Xout(1) = Yin(1) * (0.012987) + (-0.337662)
    Xout(2) = Yin(2) * (0.013514) + (-0.108108)
    Xout(3) = Yin(3) * (0.134282) + (-0.584933)

    '/* Neurons representing membership functions    */

    '/* Generating Code for PE 0 in layer 1 */
    'Prepare membership calculation for I0, Month
    Xout(4) = Xout(0) * (14.021662) + (-3.503057)
    Xout(4) = 1 / (1 + Exp(-Xout(4)))

    '/* Generating Code for PE 1 in layer 1 */
    Xout(5) = Xout(0) * (-13.848388) + (5.677543)
    Xout(5) = 1 / (1 + Exp(-Xout(5)))
```

```
'/* Generating Code for PE 2 in layer 1 */
Xout(6) = Xout(0) * (14.272412) + (-5.061203)
Xout(6) = 1 / (1 + Exp(-Xout(6)))

'/* Generating Code for PE 3 in layer 1 */
Xout(7) = Xout(0) * (-13.307358) + (9.547252)
Xout(7) = 1 / (1 + Exp(-Xout(7)))

'/* Generating Code for PE 4 in layer 1 */
Xout(8) = Xout(0) * (14.284948) + (-8.19891)
Xout(8) = 1 / (1 + Exp(-Xout(8)))

'/* Generating Code for PE 5 in layer 1 */
Xout(9) = Xout(0) * (-14.018159) + (11.208872)
Xout(9) = 1 / (1 + Exp(-Xout(9)))

'/* Generating Code for PE 6 in layer 1 */
'Prepare membership calculation for I1, High Temperature
Xout(10) = Xout(1) * (13.781557) + (-3.455154)
Xout(10) = 1 / (1 + Exp(-Xout(10)))

'/* Generating Code for PE 7 in layer 1 */
Xout(11) = Xout(1) * (-14.375848) + (4.664914)
Xout(11) = 1 / (1 + Exp(-Xout(11)))

'/* Generating Code for PE 8 in layer 1 */
Xout(12) = Xout(1) * (14.373196) + (-8.22622)
Xout(12) = 1 / (1 + Exp(-Xout(12)))

'/* Generating Code for PE 9 in layer 1 */
Xout(13) = Xout(1) * (-14.023259) + (11.210794)
Xout(13) = 1 / (1 + Exp(-Xout(13)))

'/* Generating Code for PE 10 in layer 1 */
'Prepare membership calculation for I2, Low Temperature
Xout(14) = Xout(2) * (14.077441) + (-5.565)
Xout(14) = 1 / (1 + Exp(-Xout(14)))

'/* Generating Code for PE 11 in layer 1 */
Xout(15) = Xout(2) * (-13.271733) + (9.410386)
Xout(15) = 1 / (1 + Exp(-Xout(15)))

'/* Generating Code for PE 12 in layer 1 */
'Prepare membership calculation for I3, Previous Flow
Xout(16) = Xout(3) * (14.923055) + (-6.821737)
Xout(16) = 1 / (1 + Exp(-Xout(16)))

'/* Generating Code for PE 13 in layer 1 */
Xout(17) = Xout(3) * (-12.941121) + (12.352075)
Xout(17) = 1 / (1 + Exp(-Xout(17)))

'/* Generating Code for PE 0 in layer 2 */
Xout(18) = Xout(0) * (-15.024256) + (0.469505)
Xout(18) = 1 / (1 + Exp(-Xout(18)))

'/* Generating Code for PE 1 in layer 2 */
Xout(19) = Min(Xout(4), Xout(5))

'/* Generating Code for PE 2 in layer 2 */
Xout(20) = Min(Xout(6), Xout(7))

'/* Generating Code for PE 3 in layer 2 */
Xout(21) = Min(Xout(8), Xout(9))

'/* Generating Code for PE 4 in layer 2 */
Xout(22) = Xout(0) * (13.976202) + (-11.935321)
Xout(22) = 1 / (1 + Exp(-Xout(22)))

'/* Generating Code for PE 5 in layer 2 */
Xout(23) = Min(Xout(10), Xout(11))

'/* Generating Code for PE 6 in layer 2 */
Xout(24) = Min(Xout(12), Xout(13))
```

```
'/* Generating Code for PE 7 in layer 2 */
Xout(25) = Xout(1) * (13.777318) + (-12.34531)
Xout(25) = 1 / (1 + Exp(-Xout(25)))

'/* Generating Code for PE 8 in layer 2 */
Xout(26) = Min(Xout(14), Xout(15))

'/* Generating Code for PE 9 in layer 2 */
Xout(27) = Xout(2) * (13.898801) + (-11.729348)
Xout(27) = 1 / (1 + Exp(-Xout(27)))

'/* Generating Code for PE 10 in layer 2 */
Xout(28) = Xout(3) * (-13.058938) + (5.356061)
Xout(28) = 1 / (1 + Exp(-Xout(28)))

'/* Generating Code for PE 11 in layer 2 */
Xout(29) = Min(Xout(16), Xout(17))


'/* Neurons representing rules                        */

'/* Generating Code for PE 0 in layer 3 */
'Rule 11
Xout(30) = Xout(24)

'/* Generating Code for PE 1 in layer 3 */
'Rule 6
Xout(31) = Xout(22)

'/* Generating Code for PE 2 in layer 3 */
'Rule 4
Xout(32) = Xout(20)
'Xout(32) = 0 'modified, 02/23/98, StdDev=0.972

'/* Generating Code for PE 3 in layer 3 */
'Rule 7
Xout(33) = Xout(19)

'/* Generating Code for PE 4 in layer 3 */
'Rule 1
Xout(34) = Xout(25)

'/* Generating Code for PE 0 in layer 4 */
'Rule 12
Xout(35) = Max(Xout(26), Xout(29))

'/* Generating Code for PE 1 in layer 4 */
'Rule 10
Xout(36) = Max(Xout(21), Xout(26))

'/* Generating Code for PE 2 in layer 4 */
'Rule 9
Xout(37) = Max(Xout(20), Xout(26))

'/* Generating Code for PE 3 in layer 4 */
'Rule 8
Xout(38) = Max(Xout(24), Xout(27))

'/* Generating Code for PE 4 in layer 4 */
'Rule 5
Xout(39) = Max(Xout(18), Xout(28))

'/* Generating Code for PE 0 in layer 5 */
'Rule 3
Xout(40) = Min(Xout(20), Xout(23))
'Xout(40) = 0 'modified, 02/23/98, StdDev=0.970

'/* Generating Code for PE 1 in layer 5 */
'Rule 2
Xout(41) = Min(Xout(18), Xout(25))


'/* Neurons evaluating weighted rules            */

'/* Generating Code for PE 0 in layer 6 */
```

```
    Xout(42) = Xout(41) * (-1.273858) + Xout(34) * (1.163262) _
             + Xout(40) * (0.787559) + Xout(33) * (-0.304031) _
             + Xout(32) * (-0.560732) + Xout(31) * (-0.370114) _
             + Xout(39) * (-0.533187) + Xout(38) * (0.278214) _
             + Xout(30) * (-0.104999) + Xout(37) * (0.239863) _
             + Xout(36) * (-0.143949) + Xout(35) * (-0.021554) _
             + (-0.066442)
    Xout(42) = 1 / (1 + Exp(-Xout(42)))

    '/* De-scale and write output from network */

    'Yout = Xout(42) * (12.411668) - (6.838333)
    Yout = Xout(42) * (12.411668) - (6.838333) + 8.71122

End Function
```

Procedure List

Procedure	Module	Returns	Arg	Type
cmdBrowseSource_Click	BATCH.FRM		(None)	(N/A)
cmdBrowseSource_Click	BATCH.FRM		(None)	(N/A)
cmdCalc_Click	FUNE.FRM		(None)	(N/A)
cmdClose_Click	BATCH.FRM		(None)	(N/A)
cmdExit_Click	FUNE.FRM		(None)	(N/A)
cmdStart_Click	BATCH.FRM		(None)	(N/A)
cmdStop_Click	BATCH.FRM		(None)	(N/A)
cmdViewOutput_Click	BATCH.FRM		(None)	(N/A)
cmdViewSource_Click	BATCH.FRM		(None)	(N/A)
Form_Load	BATCH.FRM		(None)	(N/A)
Form_Load	FUNE.FRM		(None)	(N/A)
MakeOutFile	BATCH.FRM	Variant	Source	String
Max	FUNE.BAS	Double	A B	Double Double
Min	FUNE.BAS	Double	A B	Double Double
mnuBatchEvalute_Click	FUNE.FRM		(None)	(N/A)
mnuExit_Click	FUNE.FRM		(None)	(N/A)
NN_Recall	FUNE.BAS	Integer	Yin	Double
StdDev	FUNE.BAS	Single	N P	Integer Single
txtInput_GotFocus	FUNE.FRM		Index	Integer
txtInput_LostFocus	FUNE.FRM		Index	Integer
txtOutputFile_Change	BATCH.FRM		(None)	(N/A)

txtOutputFile_GotFocus	BATCH.FRM	(None)	(N/A)
txtOutputFile_LostFocus	BATCH.FRM	(None)	(N/A)

Procedure Calling List

Procedure	Module	Calls	Module
cmdBrowseSource_Click	BATCH.FRM	MakeOutFile	BATCH.FRM
		MakeOutFile	BATCH.FRM
cmdCalc_Click	FUNE.FRM	NN_Recall	FUNE.BAS
cmdStart_Click	BATCH.FRM	NN_Recall	FUNE.BAS
		StdDev	FUNE.BAS
NN_Recall	FUNE.BAS	StdDev	FUNE.BAS
		Max	FUNE.BAS
txtInput_LostFocus	FUNE.FRM	NN_Recall	FUNE.BAS

Procedure Called By List

Procedure	Module	Called By	Module
MakeOutFile	BATCH.FRM	cmdBrowseSource_Click	BATCH.FRM
		cmdBrowseSource_Click	BATCH.FRM
Max	FUNE.BAS	NN_Recall	FUNE.BAS
NN_Recall	FUNE.BAS	cmdCalc_Click	FUNE.FRM
		cmdStart_Click	BATCH.FRM
		txtInput_LostFocus	FUNE.FRM
StdDev	FUNE.BAS	cmdStart_Click	BATCH.FRM
		NN_Recall	FUNE.BAS

2

VITA

Yu-Wen Lin

Candidate for the Degree of

Master of Science

Thesis: A WATER DEMAND FORECASTING SYSTEM USING FUZZY LOGIC

Major Field: Computer Science

Biographical:

Personal Data: Born in Tainan, Taiwan, Republic of China on August 16, 1966, the son of Yao-Shin Lin and Show-Chu Wu.

Education: Graduated from Chen-Kuo Senior High School, Taipei, Taiwan, Republic of China in June 1984; received Bachelor of Science in Agricultural Machinery Engineering from National Chung-Hsing University, Taichung, Taiwan, Republic of China in June 1988. Completed the requirements for the Master of Science degree at Oklahoma State University in July 1998.

Experience: Research Assistant of Computer Science Department at Oklahoma State University, January, 1997 to present; Teaching Assistant of Computer Science Department at Oklahoma State University, January, 1995 to December, 1997; Research Assistant of Biosystems and Agricultural Engineering Department, January, 1996 to August 1997;