Journal of the Korean Nuclear Society
Volume 36, Number 3, pp.219~228, June, 2004

Study on the Self Diagnostic Monitoring System for an Air-Operated Valve : Algorithm for Diagnosing Defects

Wooshik Kim
Sejong University
98 Gunja-dong, Gwangjin-gu, Seoul, Korea

Jangbom Chai and Hyunwoo Choi
Ajou University
San 5, Wonchun-dong, Yeongtong-gu, Suwon, Korea
wskim@sejong.ac.kr

(Received January 21, 2003)

Abstract

[1] and [2] present an approach to diagnosing possible defects in the mechanical systems of a nuclear power plant. In this paper, by using a fault library as a database and training data, we develop a diagnostic algorithm 1) to decide whether an Air Operated Valve system is sound or not and 2) to identify the defect from which an Air-Operated Valve system suffers, if any. This algorithm is composed of three stages: a neural net stage, a non-neural net stage, and an integration stage. The neural net stage is a simple perceptron, a pattern-recognition module, using a neural net. The non-neural net stage is a simple pattern-matching algorithm, which translates the degree of matching into a corresponding number. The integration stage collects each output and makes a decision. We present a simulation result and confirm that the developed algorithm works accurately, if the input matches one in the database.

Key Words : AOV, symptom, neural net, pattern recognition, pattern matching

1. Introduction

In a nuclear power plant, vibration, contamination, unplanned cease, and malfunction of components may cause a gradual degradation or even an abrupt shutdown of operations. To increase the safety and the stability of a typical nuclear power plant, it is necessary to monitor constantly the components that influence plant' s system performance and to diagnose properly any defects that said components might have. Air Operated Valve (AOV) systems are indispensable components in typical nuclear power plants. In addition, the Nuclear Regulatory Commission (NRC) recommends the use of AOVs in testing and monitoring a plant' s system performance. In this paper, we propose a diagnostic algorithm that can monitor and diagnose possible defects that the AOV system may suffer.

This paper is organized as follows. First, we introduce the overall structure of the algorithm. Then, we explain the three stages that compose

the algorithm: the neural net stage, the non-neural net stage, and the integration stage. Finally, we present our simulation results.

2. Methodology

2.1. Overall Structure of the Defect Decision System

The overall structure of the system for detecting and identifying defects in an AOV system is illustrated in Figure 1. This system works as follows. In various locations of the AOV system, sensors are placed that constantly measure signals and produce data. A data processor collects and processes these (analog or digital) data to extract features. These features are eventually transformed into arrow patterns and sent to a decision processor. The decision processor analyzes correlation with pre-processed arrow patterns that are stored in a database and identifies the current states of the AOV system. The decision processor, using the correlation between the inputted arrow patterns and the database, 1) determines whether the AOV system is sound or impaired and 2) identifies the type of defects that exist, if any. The most important components involved are the data processor and the decision processor. In [3], the data processor is considered. In this paper, we consider the decision processor.

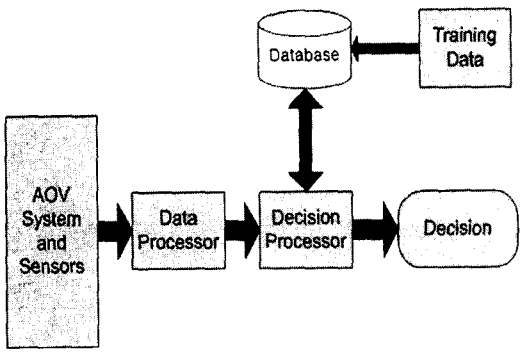

Fig. 1. Overall Block Diagram of the System for Detecting and Identifying Defects in an AOV System

2.2. Extraction of Standard Parameters of the AOV

2.2.1. Symptoms in AOV System

AOV systems can incur many defects [3]. In this study, well-known defects were purposely introduced to the AOVs in a plant to see how the power plant system would react. The experimental procedures were the same as those for the baseline experiment, except that the AOV had defective components. For each defect, the defective levels are controlled to ascertain the observability of the measured signals. Most defects have unique patterns but some defects have similar reaction patterns. When taken together, all the defects demonstrate 12 main symptoms (18 symptoms when including sub-symptoms). Based on the defects and the reaction each initiates, a library of fault symptoms was constructed and is shown in Table 1 [3].

2.2.2 Processing of Inputs

In [3], for each of the symptoms, we derived a series of arrow patterns. In Table 2, we show the arrow patterns of the symptom "Leakage at Position A," as an example. If leakage occurs at position A, then the data from each sensor change, as do all the values of the parameters derived from the sensor data. These changes of parameters are analyzed and represented as arrows, according to the change patterns, as shown in Table 2. The meanings of the arrows are listed in Table 3. For example, the arrow patterns of the second row i.e., 2-1 to 2-14 show the change of parameters that are given in Table 2 in [3].

Table 1 Fault Library for Defect an AOV System

No.		List
1		Restricted supplied air
2	2-1	Zero setting point of the E/P transducer (↑)
	2-2	Zero setting point of the E/P transducer (↓)
3	3-1	Span of the E/P transducer (↑)
	3-2	Span of the E/P transducer (↓)
4		Leakage at position A
5		Clogging at position A
6	6-1	Initial response point of the positioner (↑)
	6-2	Initial response point of the positioner (↓)
7		Stuck feedback linkage arm
8		Leakage at position B
9		Clogging at position B
10	10-1	Actuator spring preload (↑)
	10-2	Actuator spring preload (↓)
11	11-1	Packing load (↑)
	11-2	Packing load (↓)
12	12-1	Stiffness of the feedback spring (↑)
	12-2	Stiffness of the feedback spring (↓)

These arrow patterns are used as inputs to the algorithm for the diagnoses of defects. These arrow patterns present the data trends clearly. However, they are not easy to handle mathematically. First, these arrow patterns must be translated into corresponding numbers. Table 3 shows the rules governing the translation of arrow patterns into corresponding numbers. Here "b" is a fixed number that is used to differ between a horizontal arrow and a 'don' t care' or a 'no change' state. The number represented by b in Table 3 can be either a large number or an extremely small one.

Using Table 3, we develop a vector of numbers for each symptom arrow pattern. For example, for the symptom in Table 2, the vector of the input pattern is given as

$$
\begin{aligned}
P1 = [&b;-1;-1;b;b;b;b;b;b;b;b;b;b; \\
&b;-1;-1;b;b;b;b;b;1;-1;1;1;-1;1;1; \\
&b;b;b;1;1;b;b;b;b;b;b;b;b;1; \\
&1;1;1;-1;-1;-1;-1;-1;b;b; \\
&b;b;b;b;1;1;-1;-1;-1;-1;1;-1; \\
&-1;1;b;b;\ b;b;b;b;b;b;b;b;b;b;b;b;]
\end{aligned}
\tag{1}
$$

For each symptom, some items are more important than others. To exploit the importance,

Table 2. Arrow Patterns of the Symptom "Leakage at Position A"

Item	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10	1-11	1-12		
	↗↘	↘*	↘*	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘		
Item	2-1	2-2	2-3	2-4	2-5	2-6	2-7	2-8	2-9	2-10	2-11	2-12	2-13	2-14
	↗↘	↘*	↘*	↗↘	↗↘	↗↘	↗↘	↗	↘*	↗*	↗*	↘	↗*	↗*
Item	3-1	3-2	3-3	3-4	3-5	3-6	3-7	3-8	3-9	3-10	3-11	3-12	3-13	3-14
	↗↘	↗↘	↗↘	↗*	↗*	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗*
Item	3-15	3-16	3-17	3-18	3-19	3-20	3-21	3-22	3-23	3-24				
	↗*	↗*	↗*	↘*	↘*	↘*	↘*	↘*	↗↘	↗↘				
Item	4-1	4-2	4-3	4-4	4-5	4-6	4-7	4-8	5-1	5-2	5-3	5-4	5-5	5-6
	↗↘	↗↘	↗↘	↗↘	↗*	↗*	↘*	↘*	↘*	↘*	↗	↘*	↘*	↗
Item	6-1	6-2	6-3	6-4	6-5	6-6	6-7	6-8	6-9	6-10	7-1	7-2	7-3	7-4
	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘	↗↘

Table 3. Arrow Patterns and Their Corresponding Numbers

Arrow Pattern	Corresponding number	Meaning
↗↘	b	Don't care
↗	+1	Increase
→	0	No change
↘	-1	Decrease
↗→	b	No change or Increase
→↘	b	No change or Decrease

we put an asterisk mark (*) beside the important arrow patterns. These marks can also be implemented numerically with a mask function. For the symptom in Table 2, for example, the mask function is given as

$$
\begin{aligned}
M1 = [\ & 0;1;1;0;0;0;0;0;0;0;0;0;0; \\
& 0;1;1;0;0;0;0;0;1;1;1;0;1;1; \\
& 0;0;0;1;1;0;0;0;0;0;0;0;0;1; \\
& 1;1;1;1;1;1;1;1;0;0; \\
& 0;0;0;0;1;1;1;1;1;1;0;1; \\
& 1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;]; ?
\end{aligned} \tag{2}
$$

For all the symptoms imaginable, these vectors are prepared, stored as a database, and used as references for developing an algorithm for the identification of defects in the AOV system. Please refer to [3] for more information on the arrow patterns for each symptom.

2.3. Development of a Symptom Decision Algorithm

2.3.1. Overall Structure of the Algorithm

In Figure 2, we present a block diagram of the developed decision processor illustrated in Figure 1. In figure 2, two pattern recognition methods are used: one is a simple pattern-matching method, and the other is a neural net method. When a series of arrow patterns, which are extracted from the sensor data, are inputted to the system, the Neural Net algorithm identifies the symptom that matches the input patterns. In addition, the Non-Neural Net algorithm compares the input patterns with those of known symptoms stored in the database, calculates the degree of match, and lists possible symptoms in rank order. The system finally compares these two results, finds a common symptom, and makes a decision.

2.3.2. Pattern Recognition Module Based on Neural Network Algorithm

Neural networks are an area of research that studies the structure of neurons in a human body, especially in the human brain and the nervous system [4]. Neural networks mimic these human systems to obtain similar results as those of human beings when simulating similar work with computers. Among other application areas, neural networks have been found to work very well in

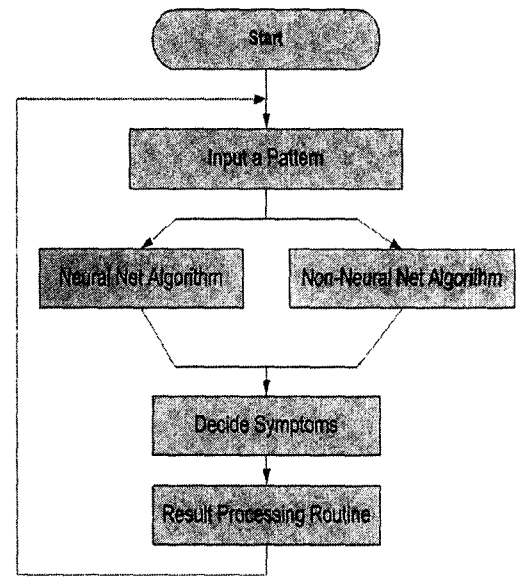

Fig. 2. Block Diagram of a Symptom Decision System

solving pattern recognition problems. In this paper, we develop a pattern recognition algorithm using a neural network.

In Several neural network models have been proposed, including single layer perceptron, Madaline, Hopfield, and multi-layer perceptron [4]. The single layer perceptron, one of the oldest neural network models, has two main advantages: a simple structure and good performance with linear dichotomy problems; it is used to solve simple character-recognition problems [4]. The multi-layer perceptron, using a back propagation algorithm, generally performs better than the single layer perceptron; however, it has longer training periods and more complex structures [4]. Adalines and Madalines are linear adaptive neural networks and are used in adaptive signal processing, or in modulation and demodulation systems, rather than in recognition problems. Hopfield networks are unsupervised networks and have self-organizing structures. Hopfield networks have two main limitations: symmetric coefficients and instability; they are usually used in building associative memories and in solving optimization problems. Other examples of neural networks are the Kohonen network, Cognitron, Neocognitron, and various Fuzzy Neural Networks [4]. All of these networks have very complex structures and a discussion of them is beyond the scope of this paper. In this study, we use a very simple neural-net model, a single layer perceptron model, in developing a pattern-recognition algorithm [4, 5].

2.3.2.1. Architecture of the Neural Network Algorithm

The structure of the single layer perceptron that we are using in this paper is shown in Figure 3. In figure 3 there are R inputs and S outputs; between these, there are a middle layer and transfer functions. The most popular transfer functions are hard limiters, linear functions, and sigmoid functions. Hard limiters are the mathematical incarnations of synaptic functions, which send

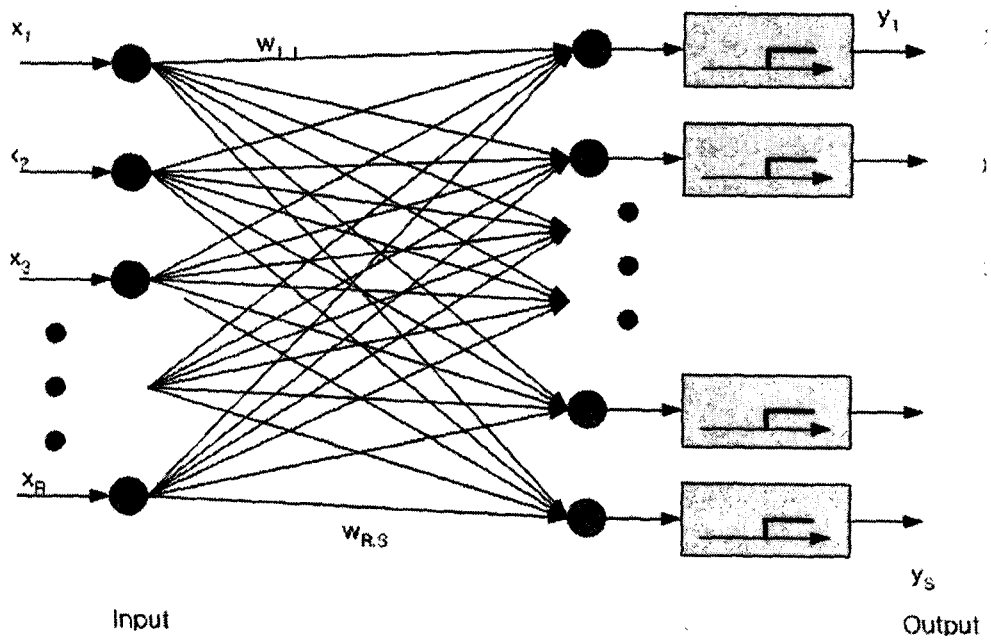

Fig. 3. A Simple Neural Network Model

synaptic pulses (i.e., the output) if the sum of all the inputs exceeds a certain threshold and otherwise do not. Because of this characteristic, hard limiters are usually used in decision components and are among the most widely used functions in pattern recognition problems. They are not, however, differentiable, which is essential for some neural nets, such as back-propagation multi-layer perceptrons. In these cases, sigmoid functions are more preferable, since they look similar to hard limiters but are differentiable. Linear functions are usually used in continuous neural net problems, such as adaptive signal processing, adaptive control, and modulation.

Mathematically, the relationship between the inputs and the outputs of the neural nets can be written as

$$\bar{y} = f(W\bar{x} + \bar{b}) \tag{3}$$

This equation is basically a matrix equation. $\bar{W}$ represents a coefficient weight matrix and has dimensions of $S \times R$. $\bar{b}$ represents the bias vector, and has dimensions of $S \times 1$. And $\bar{x}$ represents the input vector of dimension $R \times 1$. This equation, component by component, is written as follows

$$y_j = f(\sum_{i=1}^{R} w_{i,j}x_i + b_j), \quad j = 1, 2, \ldots, S \tag{4}$$

Here, $\omega_{i,j}$'s are the (i, j) components of the coefficient matrix, x_i the input, and b_j is the j^{th} component of the bias. f represents a transfer function and, in this paper, we use a hard limiter. The transfer function of the hard limiter is given as

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{5}$$

The input vector x_i's are provided by multiplying the pattern vector, P, and mask vector, M, element by element. For example, for the symptom 'Leakage at the Position A' as in Table 2, the No. 4 symptom in the table, from the equations (1) and (2), the input vector $\bar{x}$ is given as

$$\bar{\chi} = [P_k M_k] \tag{6}$$
$$= [0;-1;-1;0;0;0;0;0;0;0;0;0;$$
$$0;-1;-1;0;0;0;0;0;-1;1;1;0;1;1;$$
$$0;0;0;1;1;0;0;0;0;0;0;0;0;1;$$
$$1;1;1;-1;-1;-1;-1;-1;0;0;$$
$$0;0;0;0;1;1;-1;-1;-1;-1;0;-1;$$
$$-1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;]^t$$

If the input to the Neural Network system is $\bar{x}$, then all the output y_j's should be zero except $\bar{y}$ which should be 1. If we rewrite this output vector , then it should be a vector and is given as

$$\bar{y} = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^t. \tag{7}$$

Continuing with this procedure, we can find all 18 pairs of $(\bar{x}, \bar{y})$'s for all of the symptoms given in Table 2 and we can find the weighting matrix by training. The coefficient matrix can be derived from a series of either supervised or unsupervised training. These training approaches use various adaptation processes and update the weight matrix automatically and adaptively.

2.3.2.2. Non-neural Net Method: Simple Pattern Matching

The non-neural net algorithm is a simple pattern-matching method. This algorithm works as follows. The algorithm compares the input pattern with the arrow patterns of each symptom in the fault library, component by component, and calculates scores according to the degree to which input patterns match arrow patterns of each symptom. The score are devised to be highest if the input pattern precisely matches that of the pre-stored symptom. For example, for the arrow

with the asterisk (*) mark, if the arrow matches exactly, then the algorithm scores the input 5 points. If the input and stored arrow pattern are different by 45 degrees, then the algorithm scores the input 3 points. If the arrow pattern does not match at all, then the algorithm scores 1 point. For the case of the unmarked arrow, the algorithm scores 2 points for an exact match and 0 points otherwise. The formula that calculates the total number is

$$Score_j = \sum_{i=1}^{R} g(x_i, p_{i,j}) \tag{8}$$

Here, g represents the function of scoring the degree of matching, x_i, the i_th component of input pattern, and the i_th component of the j_th symptom. The next table shows the maximum possible values of the matching degree that each of the symptoms can have. Using this table, we calculate matching percentages using the following equation:

$$\text{Matching percentage}\ (\%) = \frac{Score_j}{Max_j} \times 100(\%) \tag{9}$$

This equation uses the maximum values of the table 4 and shows the calculated degree of matching.

3. Simulation and Results

Simulation has been done with Matlab using the Neural Net Toolbox.

3.1. Training

The Neural Net Toolbox in Matlab has several algorithms in it [6, 7]. Among these, 'adapt' and 'train' are most frequently used functions. In this simulation, we used the 'adapt' function to get neural net parameters, such as coefficients and biases. The 'adapt' function is a kind of incremental training method and is used mostly in the implementation of dynamic systems, such as designing adaptive filters or static systems, including pattern recognition [7]. A standard form of th

$$[net, a, e, pf] = adapt(net, P, T) \tag{10}$$

Table 4. Maximum Number of Scores that a Symptom Can Have

Symptom number	Maximum value	Symptom number	Maximum value	Symptom number	Maximum value
1	138	5	150	10-1	102
2-1	174	6-1	162	10-2	110
2-2	166	6-2	162	11-1	106
3-1	154	7	134	11-2	106
3-2	154	8	162	12-1	130
4	190	9	154	12-2	130

The input parameters used are as follows:

- R: Number of inputs and also the number of sensors. In this paper, we used 75 parameters and so, R is 75.
- S: Number of outputs. This matches the number of symptoms. We used 12 symptoms (18 when including sub-symptoms).
- net: This is a variable that has various neural network parameters. This stores all the parameters in the necessary memory space. Also, the results obtained after the training are also stored and outputted.
- P: The input data to be used in the training. There are S number of R×1 neural network training vectors.
- T: Represents the target value of the neural network and can be an S×S identity matrix.

The output parameters are as follows:

- net: Undated network values
- Y: Output of the network.
- E: Error.

The training data are prepared as follows. Based

on the arrow patterns of the 18 symptoms and sub-symptoms derived in [3], we prepared a pattern vector, P, and a mask vector, M. By multiplying these two vectors, the training data are prepared. With the 'adapt' function in Matlab, we calculate neural net parameters, such as the coefficients of the weighting matrix and the biases. To obtain a satisfactory result in this simulation, 60 training epochs were completed.

3.2. Simulation

The simulation has 3 stages. The first stage is a neural net stage, the second is a non-neural net stage, and the last stage is the integration of the results of the first two stages. In the first stage, we formulated a neural net algorithm and identified symptoms with the training data. The neural net algorithm developed here works well if there is no noise in the input pattern. If there is noise in the input pattern, then there is possibility that the algorithm may not yield correct results. To circumvent this problem, in the second stage, we calculated the degree of matching with a non-neural net algorithm. As described in section 2, this is a simple pattern-matching algorithm. In the last stage, we compared these two results and selected a common part of them to make a final decision.

3.3. Results

Figure 4 shows a picture of the monitor screen when the algorithm finished the calculation. In this example, we used No. 4 symptom as an input. As

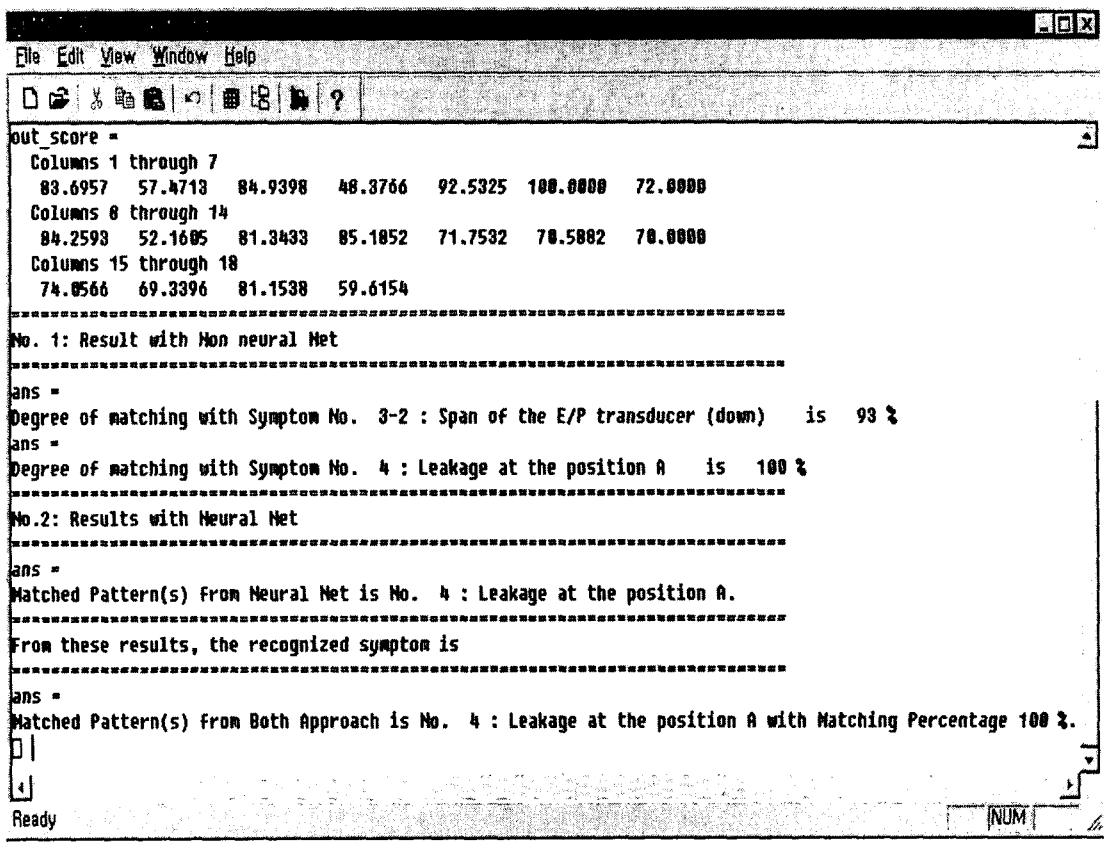

Fig. 3. A Simple Neural Network Model

Table 5. Summary of Simulation Results

No.		Input Pattern (Symptom)	Result w/ Non-neural net Approach [Symptom # (Matching Percentage)]	Result w/ Neural Net	Final Decision
1		Restricted supplied air	1(100)	1	1
2	2-1	Zero setting point of the E/P transducer (↑)	2-1(100), 3-1(91)	2-1	2-1
	2-2	Zero setting point of the E/P transducer (↓)	2-2(100), 3-2(92)	2-2	2-2
3	3-1	Span of the E/P transducer (↑)	3-1(100)	3-1	3-1
	3-2	Span of the E/P transducer (↓)	3-2(100)	3-2	3-2
4		Leakage at the position	3-2(93), 4(100)	4	4
5		Clogging at the position A	5(100)	5	5
6	6-1	Initial response point of the positioner (↑)	3-2(90), 6-1(100), 7(93), 8(92), 12-1(93)	6-1	6-1
	6-2	Initial response point of the positioner (↓)	3-1(90), 6-2(100), 12-2(93)	6-2	6-2
7		Stuck feedback linkage arm	7(100)	7	7
8		Leakage at the position B	6-1(97), 8(100),12-1(91)	8	8
9		Clogging at the position B	9(100)	9	9
10	10-1	Actuator spring preload (↑)	10-1(100)	10-1	10-1
	10-2	Actuator spring preload (↓)	10-2(100)	10-2	10-2
11	11-1	Packing load (↑)	11-1(100)	11-1	11-1
	11-2	Packing load (↓)	11-2(100)	11-2	11-2
12	12-1	Stiffness of the feedback spring (↑)	12-1(100)	12-1	12-1
12-2		Stiffness of the feedback spring (↓)	12-2(100)	12-2	12-2

expected, the result of the non-neural net algorithm shows that the symptom No. 4 matches exactly with a matching percentage of 100%. We also show the matching percentages of some of other symptoms as well, on the screen. For example, No. 3-2 symptom, which is similar to symptom No. 4, has a matching score of 94 %. All others that do not appear on the screen have matching percentages of less than 90% and so are not of importance. The result of the neural net algorithm shows that the input matches symptom No. 4 exactly. In the integration stage, a final decision is made if the two algorithm have the same result. In this case, the algorithm identifies symptom No. 4, which is the correct one.

Table 5 summarizes all the results after the algorithm was run for all the symptoms. The results from the non-neural net algorithm shows that the neural net algorithm works very well and has accurate results. In addition, the results from the non-neural net algorithm show that the algorithms work reasonable well. In general, we can say that the developed algorithm works very well for finding symptoms if the input pattern exactly matches that of any pre-given, assumed symptom. However, some symptoms show very similar patterns as other symptoms, and can have a matching percentage of as much as a 97%. Since this situation may cause a malfunction of the algorithm, we can say that it is desirable either to combine similar symptoms together or to find better parameters for future work.

4. Conclusions

In this paper, we developed a diagnostic system that (1) can determine whether an AOV system is sound or malfunctioning and, (2) can specify the defect from which the AOV system suffers. This system is composed of three stages: a neural net algorithm, a non-neural net algorithm, and an integration stage. For the neural net algorithm, we used a simple perceptron model. For the non-neural net algorithm, we used a simple pattern-matching algorithm that translates the degree of matching into a corresponding percentage. Based on the results of these two algorithms, the integration stage produces a final decision and specifies the class of defect. In our simulation, we used test patterns as inputs and determined whether the algorithm works accurately. We found that the system accurately specifies a symptom corresponding to that of the input.

In this paper, we used a single layer perceptron, that is, a neural network with the simplest structure. This algorithm works well with problems such as linear dichotomy; however, it does not generally obtain good results with nonlinear problems. The sensitivity to noise and the instability that were observed in this study appear to be caused mainly from the simplicity of the structure. The problems addressed in this paper are expected to be solved, if we use a more complex and complicated algorithm, such as a multi-layer perceptron, using back propagation, a Hopfield net, a Kohonnen net, or fuzzy neural networks. Future work necessitates an upgrade to a more powerful neural net stage and the application of the algorithm to an AOV system in an actual nuclear power plant.

Acknowledgement

This work was carried out under the NERI program by MOST of Korea.

References

1. L.J. Bond et al., "On-line Intelligent Self-Diagnostic Monitoring for Next Generation Nuclear Power Plants", NPIC&HMIT 2000, 1~10, Washington DC (2000).
2. Don B. Jarrell et al., "Prognostic and Condition Based Maintenance (CBM) - A Scientific Crystal Ball", Plant reliability for now and future Ⅱ, *ICAPP paper* 1194, 1~7 (2002).
3. Jangbom Chai, et al., "Study on the Self Diagnostic Monitoring System for an Air-Operated Valve : Development of a Fault Library", *Journal of Korea Nuclear Society*, Submitted for publication.
4. Daesoo Kim, "Neural Net Theory and its Applications", High Tech Info, Seoul (2001).
5. Kwanhyung Lee and et. al, "The Principles of the Artificial Intelligence and its Real Examples", TongYong Press, Seoul (1998).
6. Hyunyub Lee, Kyungil Moon, "Fuzzy and Neuro Using Matlab", Ajin, Seoul (1999).
7. Howard Demuth, Mark Beal, "Neural Network Toolbox For Use with Matlab" User' s Guide Version 3.0, MathWorks, Inc. (1998).