A PERFORMANCE BASELINE FOR MACHINERY CONDITION CLASSIFICATION BY NEURAL NETWORK

by

Roger Alan Nichols

Project report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Systems Engineering

APPROVED:

F. J. Ricci, Chairman

B. S. Blanchard, Jr

D. J. Schaefer

April, 1993

Blacksburg, VA

A PERFORMANCE BASELINE FOR MACHINERY CONDITION CLASSIFICATION BY NEURAL NETWORK

by

Roger Alan Nichols

Chairman: Dr F. J. Ricci

Department: Electrical Engineering

(ABSTRACT)

This project develops a set of multi-layered perceptron neural networks to serve as performance baselines for the classification of the material condition of a representative helicopter intermediate gearbox by advanced neural network models currently under development.

The first half of a collection of machinery condition sensor data recording induced faults in a TH-1L helicopter intermediate gearbox is used to develop candidate network configurations and the second half of the data collection to test the candidate networks. The data is derived from three accelerometer sensor channels. The network with the lowest average machinery condition classification error is chosen as the baseline network for that sensor channel and described in "C" computer program code.

The error in gearbox machinery condition classification for these networks ranges from 2.2% for the sensor channel five network to 7.9% for the channel 5+6+7 combined network.

CONTENTS

ILLUSTRATIONS

FIGURES page

ILLUSTRATIONS

TABLES

I.

INTRODUCTION

There is considerable interest today in developing a capability for the automatic monitoring and classification of machinery condition. The ability to do this would allow maintenance decisions to be made directly on the observed material condition rather than from a discovered failure or a tear-down-and-inspect conducted after a specified period of operation.

With an automatic monitoring system installed the problem remains of analyzing the signal output to determine the machinery condition. If the system has been fully characterized, algorithms could be developed for the analysis. On the other hand, a neural computer network could be developed and trained to do the classification, without algorithm development.

Taking the latter approach, a group of investigators from industry has designed a particular model of a neural network [7] which shows promise. The network has been applied to the problem of classifying the material condition of a helicopter intermediate gearbox with six possible conditions, normal operation or one of five common faults. The input data is in

the form of an 8-element material condition vector developed from gearbox accelerometer data.

A problem now arises in assessing the value of this new network model. How good is it? That is, how does it compare with a "basic" neural network? A baseline neural network is needed to serve as a comparison standard for the performance evaluation of the new network model and for others yet to be developed. This project develops a multi-layered perceptron neural network model to serve as this baseline network.

A short tutorial on neural computing is presented in Appendix A.

II.

PROJECT

Task 1. Develop a multi-layered perceptron neural network to serve as a baseline in the performance evaluation of other neural networks applied to the classification of the material condition of a helicopter intermediate gearbox. The perceptron network will accept an 8-element material condition vector and return a 6-element output vector identifying normal operation or one of five common faults of the gearbox. Network selection will be based on lowest percentage machinery condition classification error.

Task 2. Establish the performance of the selected baseline neural network.

III.

MATERIALS

COMPUTER PROGRAM

The "NeuralWorks Professional II/Plus" computer program was obtained from "NeuralWare, Inc." and loaded on the IBM personal computer in the EE Department laboratory at the Northern Virginia Graduate Center of VPI & SU. The perceptron back-propagation section of this program was used to generate the multi-layer neural networks examined as candidates for the baseline network.

The NeuralWare documentation includes a tutorial [8] on how to use the program, but some experimentation and sample network trials were necessary to understand the details as the NeuralWare program has rather extensive neural network capabilities. The perceptron network back-propagation section represents only one of the many neural network capabilities of this program.

SAMPLE DATA

An analog frequency signal data set was obtained from the US Naval Air Test Center, Patuxent River, Maryland. This data consists of machinery condition signals from a TH-1L helicopter intermediate gearbox in which predetermined faults were induced. The gearbox is described in reference [2] as consisting of a "duplex ball bearing set and a roller bearing which support the input pinion, and identical bearings supporting the output gear. The gear and pinion are spiral bevel and transfer power at an angle of 42 degrees."

The analog signals were recorded over a total of nine test runs and represent fault levels which could be found in an operating gearbox. That is, they are somewhere in the range between fault inception and gear box failure. Two runs record the no fault case, two reflect inner race faults on the output ball bearing, and two outer race faults on the input ball bearing. The effect of a rolling bearing fault on the input ball bearing, gear spalls on two teeth, and the loss of half of a gear tooth are recorded in one run each. The signals were derived from three accelerometers mounted at the output end of the gearbox and positioned radially to the axis of the shaft. One accelerometer was located near the ball bearing load zone (channel five), one near the roller bearing

load zone (channel six), and the other (channel seven) approximately 90 degrees from these two load zones.

To condition the test samples for neural network analysis the accelerometer signals were passed through an anti-aliasing filter with a cutoff frequency of 16 kHz and digitalized at a 46.880 kHz sampling rate. Discrete time domain signals formed from one-second samples were then transformed into the frequency domain by a 1024-point FFT and the power spectral density computed and normalized. Eight feature elements were established by averaging the power spectral density function over sixteen equally spaced bins, corresponding to 2.93 kHz each, and discarding bins 9-16 since the signal is real valued. The classification data may thus be viewed as consisting of the components of a low resolution normalized power spectral density function of a windowed accelerometer time history.

Machinery condition case identification was added to the eight feature elements in the form of a six-element vector representing the six possible gearbox conditions as follows:

100000 No fault. Normal gearbox operation.

010000 Inner race fault on output ball bearing.

001000 Outer race fault on input ball bearing.

000100 Rolling element fault on input ball bearing.

000010 Gear spalls on two teeth.

000001 Loss of 1/2 tooth.

The data available for network analysis consisted of 480 machinery condition vectors from each of three channels for a total of 1440 vectors. Of these, 180 x 3 = 540 were obtained from the first half of the one-second sample collection and were used as the network training library. The other 300 x 3 = 900 came from the last half of the one-second sample collection and formed the network testing library.

IV.

METHOD

The machinery condition library data were assembled in the required format for the NeuralWare program. For both the training and testing sets the input was arranged in the form of a 14-element vector consisting of the eight feature elements followed by the six-element machinery condition case identification vector. The 12-element output vector consisted of the six elements of the machinery condition identification followed by the six elements representing the network's determination of the machinery condition. In the training cycle the machinery condition vector identified the desired output for the back-propagation error determination. In the test phase it served as the case identifier for evaluation of the network output.

The NeuralWare program was directed to scale the input data to match the range of the non-linear transformation process of the network processing element (-1,1) and, in turn, to descale the network output.

One-, two-, and three-layer networks were developed, all taking the eight-element input vector and returning a six-element machinery condition output vector. The one-layer

network type consisted of an input of eight elements and an output layer of six active processing elements, an 8-6 layer format. The two-layer networks added a hidden layer (a layer internal to the network and not directly observable from outside the network) of seven processing elements to form an 8-7-6 layer format. The three-layer networks had two hidden layers. One set had two hidden layers of seven processing elements each (8-7-7-6). A second had one hidden layer of eight processing elements and one of seven (8-8-7-6). The third set had an 8-7-6-6 layer format.

All of the networks were of the fully connected, feed forward type. That is, all of the processing elements in one layer were fully connected to each of the processing elements in the next layer. There were no forward connections that skipped a layer. Nor were there any feedback loops. The back-propagation learning cycle utilized the same connections in the reverse direction.

In the perceptron neural network the input connections to the processing element are weighted. The processing element sums the weighted inputs, passes the sum through a non-linear transformation element and distributes the output to the connections of the next layer. The hyperbolic tangent was chosen as the non-linear transformation rather than the

sigmoid since the sigmoid has only a (0,1) range while the hyperbolic tangent has a larger (-1,1) range. The weights were initially set to small randomized values and adjusted in the learning, or training, phase via the back-propagation process.

The rate of learning, or amount of individual processing element weight change allowed per update, was varied at intervals in keeping with the theory that changes should be large at first and decrease as the network approaches convergence, the final trained state. This was accomplished by establishing a weight update, or learning, schedule based on the training cycle count. Values between 0 and 1 were assigned to the coefficient (C1) for the weight update term and to the coefficient (C2) for the momentum term of the back-propagation error distribution equation.

The basic learning schedule was that for the 8-7-6 network. For this C1 and C2 of the output layer were set to 0.2 and changed to 0.1 at the 1000 cycle count. For the hidden layer C1 was set to 0.8 and C2 to 0.2 with a change at the 1000 cycle count to 0.3 and 0.1, respectively. The input schedule was left at the default case of 0.9 and 0.6 for C1 and C2. This followed the NeuralWare program rule of thumb that the values of C1 and C2 should also decrease over the

layers from input to output. Alternate schedules with decreasing values for C1 and C2 specified at intervals out to a last change at 30,000 cycles were also set up.

The weight update intervals followed the normalized cumulative delta rule. The error of the individual processing element, as determined by the back-propagation error distribution process, was accumulated for the individual processing element over a fixed number of training cycles (epoch), that total divided by the epoch number, and the result applied to the individual processing element as a delta weight correction. This tended to smooth out wide weight oscillations and reduce the probability of a processing element locking up in a plus or minus one state. Epoch values from 16 (the default case) up to the number of sample vectors in the training set were tried. In the former case the graph of the RMS error of the output layer was very grainy. In the later, it lost all of its detail and much of its information.

Training consisted of presenting the network with randomly selected vectors from the training set and observing the RMS error, and distribution of processing element connection weights, of the output layer as the training progressed. Training was interrupted and that network configuration saved when the RMS error reached successive

error threshold values of 0.200, 0.175, 0.150, ... , 0.050 and when the training cycle count reached 55,000 to 110,000 depending on the apparent stability of the graph trace of the output layer RMS error. The RMS error value indicated that training was taking place by decreasing in value. However, the RMS error graph trace generally settled out at about 0.250 with irregular fluctuations of around plus/minus 0.05 and some of plus/minus 0.10 or so. It did not converge to zero as would be the case for perfect training.

The trained networks were then tested with the library of testing vectors and the output saved to file. The test result vectors were examined under the criterion of winner takes all. The output vector element with the highest numerical value was taken to be the network evaluation of the machinery condition, the "1" of the machinery condition case vector. There were no ties. The number of errors in case identification was determined and the percentage error calculated for each of the networks.

In all, 37 network configurations were evaluated for selection as a baseline. The 8-6 and 8-7-6 layer format networks were trained and tested for single channel evaluation with the network exhibiting the lowest percent classification error selected as the baseline for that channel, one each for

channels five, six, and seven. Similarly, 8-7-6, 8-8-7-6, 8-7-7-6, and 8-7-6-6 layer format networks were trained and tested for channel 5+6+7 combined with the one presenting the lowest percent classification error selected as the baseline.

The final step was to obtain the network description in "C" computer code from the NeuralWare program for each of the baseline. This code is a directly callable routine for a computer program. It is made up of the scaling/descaling factors and the transformation equation for each processing element of the network.

V.

RESULTS

The test results of the 37 trained networks are shown in Tables I-IV. Networks trained and tested with channel 5 data are grouped in Table I. Those networks using channel 6 data are in Table II, and those for channel 7 in Table III. Table IV contains the networks trained and tested with the complete data library, channel 5+6+7 combined.

The networks are identified by layer format and the point at which training was stopped on that particular network, either RMS error of the output layer or cycle count.

The "network error" column tabulates the network test results. The network figure in this column is the calculated percent error in classification for that network. The network with the lowest percent error is identified in the "remarks" column.

Table I. Channel 5 Network Training Results

NETWORK LAYER FORMAT	TRAINING STOPPED AT				NETWORK ERROR (%)	REMARKS
	INDICATOR		VALUE			
	RMS ERR(1)	CYCLE COUNT	RMS ERR(2)	CYCLE COUNT		
8-7-6	X		0.150	3056	6.22	
	X		0.125	6416	2.22	BASELINE STANDARD
	X		0.050	39,216	5.00	
		X	[0.09]	75,000	4.11	
		X	[0.15]	100,000	4.22	(3)
8-6	X		0.200	1936	4.00	
	X		0.150	5136	4.78	
		X	[0.18]	55,000	6.78	

Note: 1. RMS error of output layer (threshold value).
2. [ ] - Estimated from RMS error graph.
3. Alternate training schedule used. Epoch set to number of training library samples (180).

Table II. Channel 6 Network Training Results

NETWORK LAYER FORMAT	TRAINING STOPPED AT				NETWORK ERROR (%)	REMARKS
	INDICATOR		VALUE			
	RMS ERR(1)	CYCLE COUNT	RMS ERR(2)	CYCLE COUNT		
8-7-6	X		0.100	13,136	4.00	
	X		0.075	28,016	2.89	BASELINE STANDARD
	X		0.050	59,536	4.40	
		X	[0.07]	75,000	4.56	
8-6	X		0.200	4016	13.56	
	X		0.175	15,696	6.89	
		X	[0.37]	55,000	5.00	

Note: 1. RMS error of output layer (threshold value).
2. [ ] - Estimated from RMS error graph.

Table III. Channel 7 Network Training Results

NETWORK LAYER FORMAT	TRAINING STOPPED AT				NETWORK ERROR (%)	REMARKS
	INDICATOR		VALUE			
	RMS ERR(1)	CYCLE COUNT	RMS ERR(2)	CYCLE COUNT		
8-7-6	X		0.150	4976	6.00	
	X		0.125	11,696	5.33	
	X		0.075	39,856	5.00	BASELINE STANDARD
		X	[0.10]	75,000	7.00	
8-6	X		0.200	5776	6.67	
		X	[0.33]	55,000	6.44	

Note: 1. RMS error of output layer (threshold value).
2. [ ] - Estimated from RMS error graph.

Table IV. Channel 5+ 6+7 Combined Network Training Results

NETWORK LAYER FORMAT	TRAINING STOPPED AT				NETWORK ERROR (%)	REMARKS
	INDICATOR		VALUE			
	RMS ERR(1)	CYCLE COUNT	RMS ERR(2)	CYCLE COUNT		
8-7-6	X		0.175	16,976	9.41	
		X	[0.30]	35,000	8.22	
	X		0.150	44,016	9.63	
		X	[0.20]	66,000	7.89	BASELINE STANDARD
		X	[0.23]	101,000	8.52	
	X		0.125	12,656	11.26	(3)
	X		0.125	14,576	11.44	(3)
		X	[0.30]	50,000	11.33	(3)
8-8-7-6	X		0.175	31,056	12.15	
		X	[0.32]	51,000	12.11	
8-7-7-6	X		0.150	9936	9.48	
		X	[0.25]	55,000	9.00	
8-7-6-6	X		0.200	10,736	11.26	
	X		0.120	22,896	11.19	
	X		0.125	33,616	10.07	
		X	[0.32]	55,000	11.37	

Note: 1. RMS error of output layer (threshold value).
2. [ ] - Estimated from RMS error graph.
3. Alternate training schedule used.

VI.

DISCUSSION

It does not appear that one network would adequately serve as the baseline in all cases. For this reason four separate baseline networks have been identified, one for channel five, one for channel six, one for channel seven, and one for the combined case of all three sensors.

Of the various network layer formats the 8-7-6 format performed best. All of the baseline networks are of this format.

Analysis of the test results for each of the networks chosen as a baseline is shown in the form of a confusion matrix in Tables V-VIII. The confusion matrix relates the network classification decision (row) to the actual machinery condition (column) as a percentage of machinery condition case test samples. In these matrices the column values add to 100 percent.

Variations in learning schedule did not appear to improve training over that accomplished in the simple schedule used for most of the 8-7-6 networks. This had been noted in earlier trial runs and remained true in the recorded cases. This is probably a characteristic of the data as all of the

networks settled out at an RMS error for the output layer of about 0.250 with fluctuations of about plus/minus 0.05. Some, of course, had larger fluctuations than others.

Table V. Confusion Matrix For Channel 5 Baseline Standard Network

NETWORK CLASSIFICATION	FAULT					
CASE	1	2	3	4	5	6
1	99.5	1.0	5.5	-	-	-
2	-	97.5	-	-	-	-
3	0.5	1.0	94.5	2.0	1.0	-
4	-	-	-	98.0	-	-
5	-	0.5	-	-	99.0	-
6	-	-	-	-	-	100.0

Note: Matrix values in %.
Column 1-3, 200 test samples each.
Column 4-6, 100 test samples each.
Case 1 Normal gearbox operation.
Case 2 Inner race fault on output ball bearing.
Case 3 Outer race fault on input ball bearing.
Case 4 Rolling element fault on input ball bearing.
Case 5 Gear spalls fault.
Case 6 Half tooth fault.

Table VI. Confusion Matrix For Channel 6 Baseline Standard Network

NETWORK CLASSIFICATION	FAULT					
CASE	1	2	3	4	5	6
1	94.5	-	-	-	-	-
2	-	100.0	0.5	-	-	-
3	5.5	-	97.0	-	-	-
4	-	-	2.5	100.0	-	-
5	-	-	-	-	96.0	5.0
6	-	-	-	-	4.0	95.0

Note: Matrix values in %.
Column 1-3, 200 test samples each.
Column 4-6, 100 test samples each.
Case 1 Normal gearbox operation.
Case 2 Inner race fault on output ball bearing.
Case 3 Outer race fault on input ball bearing.
Case 4 Rolling element fault on input ball bearing.
Case 5 Gear spalls fault.
Case 6 Half tooth fault.

Table VII. Confusion Matrix For Channel 7 Baseline Standard Network

NETWORK CLASSIFICATION \ FAULT	1	2	3	4	5	6
1	92.5	-	10.0	-	-	1.0
2	-	97.5	-	-	-	-
3	6.5	-	90.0	-	-	-
4	1.0	2.0	-	96.0	-	-
5	-	0.5	-	4.0	100.0	-
6	-	-	-	-	-	99.0

Note: Matrix values in %.
Column 1-3, 200 test samples each.
Column 4-6, 100 test samples each.
Case 1 Normal gearbox operation.
Case 2 Inner race fault on output ball bearing.
Case 3 Outer race fault on input ball bearing.
Case 4 Rolling element fault on input ball bearing.
Case 5 Gear spalls fault.
Case 6 Half tooth fault.

Table VIII. Confusion Matrix For Channel 5+6+7 Combined Baseline Standard Network

NETWORK CLASSIFICATION \ FAULT	1	2	3	4	5	6
1	91.0	0.2	7.5	-	-	-
2	1.0	93.0	1.8	-	1.7	2.7
3	6.8	3.0	86.8	2.0	0.7	-
4	1.2	1.0	3.5	98.0	-	-
5	-	2.8	0.3	-	96.3	4.3
6	-	-	-	-	1.3	93.0

Note: Matrix values in %.
Column 1-3, 600 test samples each.
Column 4-6, 300 test samples each.
Case 1 Normal gearbox operation.
Case 2 Inner race fault on output ball bearing.
Case 3 Outer race fault on input ball bearing.
Case 4 Rolling element fault on input ball bearing.
Case 5 Gear spalls fault.
Case 6 Half tooth fault.

VII.

SUMMARY AND CONCLUSIONS

Four multi-layer perceptron neural networks have been developed to serve as baselines in the performance evaluation of other neural networks applied to the classification of the material condition of a TH-1L helicopter intermediate gearbox. Each network accepts an eight-element material condition vector and returns a six-element output vector identifying normal operation or one of five common faults of the gearbox. These networks are described in "C" programming code in Figures 1-4. They are all two-layered networks consisting of an eight element input, one hidden layer of seven processing elements, and an output layer of six processing elements. The processing elements utilize the hyperbolic tangent transformation process.

The performance of each of the baseline networks is shown in Table IX as a percent correct classification figure. As indicated in the summary of best case network classification performance (Table X), no one network performs best in all cases. Channel five (duplex ball bearing load zone sensor) is the best in overall case classification while channel six (roller bearing load zone sensor) is the best for a basic "good/bad" gearbox classification.

The classification error of the baseline for channel five (2.2%) may also be used in the system design process for an intermediate gearbox of the TH-1L helicopter. It will provide a value for network error in the system requirements allocation process.

The development of systems for the automatic monitoring and classification of machinery condition is progressing. As neural networks are developed for the new systems, the multi-layered perceptron network can be reconfigured to serve as a performance comparison standard for the machinery condition data analysis of those systems.

```
/*   Channel 5 Baseline Neural Network (Perceptron)   */

#if __STDC__
#define  ARGS(x) x
#else
#define  ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double tanh  ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES ***
*/

#if __STDC__
int NN_GBXC5( void *NetPtr, float Yin[8], float Yout[6] )
#else
int NN_GBXC5( NetPtr, Yin, Yout )
void  *NetPtr; /* Network Pointer (not used) */
float Yin[8], Yout[6];  /* Data */
#endif /* __STDC__ */
{
    float  Xout[23];  /* work arrays */
    long   ICmpT;  /* temp for comparisons */

    /* Read and scale input into network */
    Xout[2] = Yin[0] * (0.11925415) + (8.4536229);
    Xout[3] = Yin[1] * (0.032063101) + (3.9165558);
    Xout[4] = Yin[2] * (0.028125439) + (3.8805512);
    Xout[5] = Yin[3] * (0.019380783) + (2.5365472);
    Xout[6] = Yin[4] * (0.023219095) + (4.5342714);
    Xout[7] = Yin[5] * (0.024189645) + (4.2190369);
    Xout[8] = Yin[6] * (0.024215131) + (4.3061162);
    Xout[9] = Yin[7] * (0.026888224) + (5.1941597);
LAB110:

    /* Generating code for PE 0 in hidden layer */
    Xout[10] = (float)(-0.12192477) + (float)(-2.5802357) *
       Xout[2] + (float)(-2.7966828) * Xout[3] +
       (float)(0.60664678) * Xout[4] + (float)(0.73407787) *
       Xout[5] + (float)(1.4104127) * Xout[6] +
       (float)(-2.0937653) * Xout[7] + (float)(0.17737336) *
       Xout[8] + (float)(-0.18950854) * Xout[9];
    Xout[10] = tanh( Xout[10] );
```

Figure 1. Network Description of Channel 5 Baseline.

```
/* Generating code for PE 1 in hidden layer */
Xout[11] = (float)(-0.091275521) + (float)(-0.24187014) *
   Xout[2] + (float)(1.0646201) * Xout[3] +
   (float)(0.019783229) * Xout[4] + (float)(-0.55860305)
   * Xout[5] + (float)(-0.47621715) * Xout[6] +
   (float)(-0.1754806) * Xout[7] + (float)(0.062540799) *
   Xout[8] + (float)(0.055076472) * Xout[9];
Xout[11] = tanh( Xout[11] );

/* Generating code for PE 2 in hidden layer */
Xout[12]  = (float)(-0.16416407) + (float)(-0.31288245)
   Xout[2] + (float)(0.86501116) * Xout[3] +
   (float)(-1.3168422) * Xout[4] +  (float)(2.2203441) *
   Xout[5] + (float)(-3.103837) * Xout[6] +
   (float)(1.6822222) * Xout[7] + (float)(-2.5696313) *
   Xout[8] + (float)(1.218491) * Xout[9];
Xout[12] = tanh( Xout[12] );

/* Generating code for PE 3 in hidden layer */
Xout[13]  = (float)(-0.63875461) + (float)(-2.156086) *
   Xout[2] + (float)(0.26846516) * Xout[3] +
   (float)(0.66724205) * Xout[4] + (float)(0.014900664) *
   Xout[5] + (float)(3.5206127) * Xout[6] +
   (float)(0.015751587) * Xout[7] + (float)(1.9082733) *
   Xout[8] + (float)(0.72828406) * Xout[9];
Xout[13] = tanh( Xout[13] );

/* Generating code for PE 4 in hidden layer */
Xout[14]  = (float)(-0.90285945) + (float)(-0.94491607) *
   Xout[2] + (float)(0.21370468) * Xout[3] +
   (float)(-0.74377286) * Xout[4] + (float)(-0.13708515)
   * Xout[5] + (float)(-0.24019572) * Xout[6] +
   (float)(0.37222651) * Xout[7] + (float)(0.94192708) *
   Xout[8] + (float)(1.6515536) * Xout[9];
Xout[14] = tanh( Xout[14] );

/* Generating code for PE 5 in hidden layer */
Xout[15]  = (float)(-0.18870859) + (float)(-0.13018428) *
   Xout[2] + (float)(-0.82529896) * Xout[3] +
   (float)(-2.2477729) * Xout[4] + (float)(-1.4811893) *
   Xout[5] + (float)(2.5158043) * Xout[6] +
   (float)(1.4320252) * Xout[7] + (float)(1.5353054) *
   Xout[8] + (float)(0.9324953) * Xout[9];
Xout[15] = tanh( Xout[15] );
```

Figure 1. Network Description of Channel 5 Baseline (continued).

```
/* Generating code for PE 6 in hidden layer */
Xout[16]  = (float)(0.67647803) + (float)(-0.050713181) *
   Xout[2] + (float)(-1.7828685) * Xout[3] +
   (float)(-1.5177482) * Xout[4] + (float)(-1.0550293) *
   Xout[5] + (float)(-0.20017552) * Xout[6] +
   (float)(-2.4467273) * Xout[7] + (float)(1.5668499) *
   Xout[8] + (float)(-0.098661967) * Xout[9];
Xout[16] = tanh( Xout[16] );

/* Generating code for PE 0 in output layer */
Xout[17]  = (float)(-0.77599174) + (float)(-1.6550696) *
   Xout[10] + (float)(0.09909334) * Xout[11] +
   (float)(-0.25097385) * Xout[12] + (float)(-1.0970219)
   * Xout[13] + (float)(-0.14024074) * Xout[14] +
   (float)(-0.0094530974) * Xout[15] + (float)(0.62210542)
   * Xout[16];
 Xout[17] = tanh( Xout[17] );

/* Generating code for PE 1 in output layer */
Xout[18]  = (float)(-0.85951954) + (float)(0.086964965) *
   Xout[10] + (float)(0.13034572) * Xout[11] +
   (float)(1.4705071) * Xout[12] + (float)(0.79903674) *
   Xout[13] + (float)(0.65419638) * Xout[14] +
   (float)(0.67681533) * Xout[15] + (float)(-0.25681552)
   * Xout[16];
Xout[18] = tanh( Xout[18] );

/* Generating code for PE 2 in output layer */
Xout[19]  = (float)(-1.6221168) + (float)(0.39163128) *
   Xout[10] + (float)(-0.076865353) * Xout[11] +
   (float)(-0.76000553) * Xout[12] + (float)(0.9239049) *
   Xout[13] + (float)(0.34888852) * Xout[14] +
   (float)(0.54028714) * Xout[15] + (float)(1.5615762) *
   Xout[16];
Xout[19] = tanh( Xout[19] );

/* Generating code for PE 3 in output layer */
Xout[20]  = (float)(-1.2811481) + (float)(0.82453918) *
   Xout[10] + (float)(-0.4350099) * Xout[11] +
   (float)(0.50549358) * Xout[12] + (float)(-0.41205794)
   * Xout[13] + (float)(-0.10723666) * Xout[14] +
   (float)(-0.64274096) * Xout[15] + (float)(0.6829409) *
   Xout[16];
Xout[20] = tanh( Xout[20] );
```

Figure 1. Network Description of Channel 5 Baseline (continued).

```
    /* Generating code for PE 4 in output layer */
    Xout[21]  = (float)(-1.3161408) + (float)(0.30088821) *
       Xout[10] + (float)(0.58228457) * Xout[11] +
       (float)(-0.54354525) * Xout[12] + (float)(0.59864652)
       * Xout[13] + (float)(-0.034113362) * Xout[14] +
        (float)(-1.2387977) * Xout[15] + (float)(-0.24365871)
       * Xout[16];
    Xout[21] = tanh( Xout[21] );

    /* Generating code for PE 5 in output layer */
    Xout[22]  = (float)(-1.2855778) + (float)(0.034995534) *
       Xout[10] + (float)(-0.1504169) * Xout[11] +
       (float)(-0.53491175) * Xout[12] + (float)(-0.29670572)
       * Xout[13] + (float)(-0.53877664) * Xout[14] +
       (float)(0.87916631) * Xout[15] + (float)(-1.0131755) *
       Xout[16];
    Xout[22] = tanh( Xout[22] );

    /* De-scale and write output from network */
    Yout[0] = Xout[17] * (0.62499999) + (0.5);
    Yout[1] = Xout[18] * (0.62499999) + (0.5);
    Yout[2] = Xout[19] * (0.62499999) + (0.5);
    Yout[3] = Xout[20] * (0.62499999) + (0.5);
    Yout[4] = Xout[21] * (0.62499999) + (0.5);
    Yout[5] = Xout[22] * (0.62499999) + (0.5);
    return( 0 );
}
```

Figure 1. Network Description of Channel 5 Baseline (continued).

```
/*   Channel 6 Baseline Neural Network (Perceptron)    */

#if __STDC__
#define  ARGS(x) x
#else
#define  ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double tanh  ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES ***
*/

#if __STDC__
int NN_GBXC6( void *NetPtr, float Yin[8], float Yout[6] )
#else
int NN_GBXC6( NetPtr, Yin, Yout )
void  *NetPtr; /* Network Pointer (not used) */
float Yin[8], Yout[6];  /* Data */
#endif /* __STDC__ */
{
    float  Xout[23];  /* work arrays */
    long   ICmpT;  /* temp for comparisons */

    /* Read and scale input into network */
    Xout[2] = Yin[0] * (0.11925415) + (8.4536229);
    Xout[3] = Yin[1] * (0.032063101) + (3.9165558);
    Xout[4] = Yin[2] * (0.028125439) + (3.8805512);
    Xout[5] = Yin[3] * (0.019380783) + (2.5365472);
    Xout[6] = Yin[4] * (0.023219095) + (4.5342714);
    Xout[7] = Yin[5] * (0.024189645) + (4.2190369);
    Xout[8] = Yin[6] * (0.024215131) + (4.3061162);
    Xout[9] = Yin[7] * (0.026888224) + (5.1941597);
LAB110:

    /* Generating code for PE 0 in hidden layer */
    Xout[10]  = (float)(-1.2362289) + (float)(-0.60657102) *
       Xout[2] + (float)(-0.47484633) * Xout[3] +
       (float)(-0.60000658) * Xout[4] + (float)(3.160723) *
       Xout[5] + (float)(-0.09713877) * Xout[6] +
       (float)(-4.2901735) * Xout[7] + (float)(2.6266184) *
       Xout[8] + (float)(4.4436154) * Xout[9];
    Xout[10] = tanh( Xout[10] );
```

Figure 2. Network Description of Channel 6 Baseline.

```
/* Generating code for PE 1 in hidden layer */
Xout[11]  = (float)(-1.4230086) + (float)(2.8433883) *
   Xout[2] + (float)(1.0203904) * Xout[3] +
   (float)(-0.22847529) * Xout[4] + (float)(-0.14631639)
   * Xout[5] + (float)(1.4669383) * Xout[6] +
   (float)(0.31455684) * Xout[7] + (float)(-0.072774217)
   * Xout[8] + (float)(1.1773758) * Xout[9];
Xout[11] = tanh( Xout[11] );

/* Generating code for PE 2 in hidden layer */
Xout[12]  = (float)(0.19065374) + (float)(0.70065469) *
   Xout[2] + (float)(-0.58088535) * Xout[3] +
   (float)(-3.2595415) * Xout[4] + (float)(-2.3456919) *
   Xout[5] + (float)(0.93431687) * Xout[6] +
   (float)(-3.4129519) * Xout[7] + (float)(-0.4648515) *
   Xout[8] + (float)(4.2038856) * Xout[9];
Xout[12] = tanh( Xout[12] );

/* Generating code for PE 3 in hidden layer */
Xout[13]  = (float)(-4.7789106) + (float)(-1.1866335) *
   Xout[2] + (float)(0.68420517) * Xout[3] +
   (float)(1.7134892) * Xout[4] + (float)(-2.9007587) *
   Xout[5] + (float)(5.3810911) * Xout[6] +
   (float)(6.0290437) * Xout[7] + (float)(-2.0747983) *
   Xout[8] + (float)(-1.6726545) * Xout[9];
Xout[13] = tanh( Xout[13] );

/* Generating code for PE 4 in hidden layer */
Xout[14]  = (float)(0.092911489) + (float)(-0.022896653)
   * Xout[2] + (float)(-0.78834486) * Xout[3] +
   (float)(-1.6581595) * Xout[4] + (float)(-2.1321568) *
   Xout[5] + (float)(-1.8238212) * Xout[6] +
   (float)(-1.5646186) * Xout[7] + (float)(-0.80033237) *
   Xout[8] + (float)(-1.1865985) * Xout[9];
Xout[14] = tanh( Xout[14] );

/* Generating code for PE 5 in hidden layer */
Xout[15]  = (float)(3.3436232) + (float)(1.0822866) *
   Xout[2] + (float)(-2.487431) * Xout[3] +
   (float)(-3.803617) * Xout[4] + (float)(-3.2436604) *
   Xout[5] + (float)(-1.3830557) * Xout[6] +
   (float)(-0.47244304) * Xout[7] + (float)(0.23475614) *
   Xout[8] + (float)(1.2024601) * Xout[9];
Xout[15] = tanh( Xout[15] );
```

Figure 2. Network Description of Channel 6 Baseline (continued).

```
/* Generating code for PE 6 in hidden layer */
Xout[16]  = (float)(-0.047892947) + (float)(-3.0217869) *
   Xout[2] + (float)(-4.2551174) * Xout[3] +
   (float)(6.450623) * Xout[4] + (float)(-1.1553754) *
   Xout[5] + (float)(-3.9374387) * Xout[6] +
   (float)(-1.221519) * Xout[7] + (float)(-1.6893104) *
   Xout[8] + (float)(-0.80829781) * Xout[9];
Xout[16] = tanh( Xout[16] );

/* Generating code for PE 0 in output layer */
Xout[17]  = (float)(-0.29691812) + (float)(-0.29827064) *
   Xout[10] + (float)(0.070588581) * Xout[11] +
   (float)(0.31633878) * Xout[12] + (float)(-0.32860959)
   * Xout[13] + (float)(0.98738533) * Xout[14] +
   (float)(-0.32264856) * Xout[15] + (float)(-0.092328146)
   * Xout[16];
Xout[17] = tanh( Xout[17] );

/* Generating code for PE 1 in output layer */
Xout[18]  = (float)(-0.78631836) + (float)(0.29341441) *
   Xout[10] + (float)(0.38940176) * Xout[11] +
   (float)(1.066498) * Xout[12] + (float)(0.22149064) *
   Xout[13] + (float)(-0.83620095) * Xout[14] +
   (float)(0.125808) * Xout[15] + (float)(0.29269332) *
   Xout[16];
Xout[18] = tanh( Xout[18] );

/* Generating code for PE 2 in output layer */
Xout[19]  = (float)(-1.9202411) + (float)(-0.49193397) *
   Xout[10] + (float)(-0.64181268) * Xout[11] +
   (float)(-0.97986066) * Xout[12] + (float)(-0.38663357)
   * Xout[13] + (float)(0.92065704) * Xout[14] +
   (float)(1.1099949) * Xout[15] + (float)(-2.0247054) *
   Xout[16];
Xout[19] = tanh( Xout[19] );

/* Generating code for PE 3 in output layer */
Xout[20]  = (float)(-0.97902215) + (float)(0.12083197) *
   Xout[10] + (float)(0.28198731) * Xout[11] +
   (float)(-0.06923081) * Xout[12] + (float)(0.13878107)
   * Xout[13] + (float)(-1.3049214) * Xout[14] +
   (float)(0.088924833) * Xout[15] + (float)(1.6828625) *
   Xout[16];
Xout[20] = tanh( Xout[20] );
```

Figure 2. Network Description of Channel 6 Baseline (continued).

```
    /* Generating code for PE 4 in output layer */
    Xout[21]  = (float)(-1.0301117) + (float)(0.26369184) *
       Xout[10] + (float)(-0.056752473) * Xout[11] +
       (float)(-0.28621724) * Xout[12] + (float)(-1.3436009)
       * Xout[13] + (float)(0.23642857) * Xout[14] +
       (float)(-1.2173234) * Xout[15] + (float)(-0.03417607)
       * Xout[16];
    Xout[21] = tanh( Xout[21] );

    /* Generating code for PE 5 in output layer */
    Xout[22]  = (float)(-0.54304767) + (float)(-0.70715219) *
       Xout[10] + (float)(0.098667972) * Xout[11] +
       (float)(-0.55034256) * Xout[12] + (float)(0.81649381)
       * Xout[13] + (float)(0.58692271) * Xout[14] +
       (float)(-0.69770223) * Xout[15] + (float)(0.29006338)
       * Xout[16];
    Xout[22] = tanh( Xout[22] );

    /* De-scale and write output from network */
    Yout[0] = Xout[17] * (0.62499999) + (0.5);
    Yout[1] = Xout[18] * (0.62499999) + (0.5);
    Yout[2] = Xout[19] * (0.62499999) + (0.5);
    Yout[3] = Xout[20] * (0.62499999) + (0.5);
    Yout[4] = Xout[21] * (0.62499999) + (0.5);
    Yout[5] = Xout[22] * (0.62499999) + (0.5);
    return( 0 );
}
```

Figure 2. Network Description of Channel 6 Baseline (continued).

```
/*   Channel 7 Baseline Neural Network (Perceptron)    */

#if __STDC__
#define  ARGS(x) x
#else
#define  ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double tanh  ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES ***
*/

#if __STDC__
int NN_GBXC7( void *NetPtr, float Yin[8], float Yout[6] )
#else
int NN_GBXC7( NetPtr, Yin, Yout )
void  *NetPtr; /* Network Pointer (not used) */
float Yin[8], Yout[6];  /* Data */
#endif /* __STDC__ */
{
    float  Xout[23];  /* work arrays */
    long   ICmpT;  /* temp for comparisons */

    /* Read and scale input into network */
    Xout[2] = Yin[0] * (0.11925415) + (8.4536229);
    Xout[3] = Yin[1] * (0.032063101) + (3.9165558);
    Xout[4] = Yin[2] * (0.028125439) + (3.8805512);
    Xout[5] = Yin[3] * (0.019380783) + (2.5365472);
    Xout[6] = Yin[4] * (0.023219095) + (4.5342714);
    Xout[7] = Yin[5] * (0.024189645) + (4.2190369);
    Xout[8] = Yin[6] * (0.024215131) + (4.3061162);
    Xout[9] = Yin[7] * (0.026888224) + (5.1941597);
LAB110:

    /* Generating code for PE 0 in hidden layer */
    Xout[10]  = (float)(-0.85001844) + (float)(-1.0396758) *
       Xout[2] + (float)(-1.6306974) * Xout[3] +
       (float)(6.2906308) * Xout[4] + (float)(-0.33945805) *
       Xout[5] + (float)(3.6619363) * Xout[6] +
       (float)(1.8491207) * Xout[7] + (float)(2.7430284) *
       Xout[8] + (float)(-5.7645664) * Xout[9];
    Xout[10] = tanh( Xout[10] );
```

Figure 3. Network Description of Channel 7 Baseline.

```
/* Generating code for PE 1 in hidden layer */
Xout[11]  = (float)(-0.049457848) + (float)(-1.6015607) *
   Xout[2] + (float)(-5.939713) * Xout[3] +
   (float)(-1.5613074) * Xout[4] + (float)(1.608575) *
   Xout[5] + (float)(3.2847226) * Xout[6] +
   (float)(3.511672) * Xout[7] + (float)(-4.0099545) *
   Xout[8] + (float)(1.5449253) * Xout[9];
Xout[11] = tanh( Xout[11] );

/* Generating code for PE 2 in hidden layer */
Xout[12]  = (float)(-1.5290245) + (float)(-3.6121809) *
   Xout[2] + (float)(-1.6800071) * Xout[3] +
   (float)(4.3991647) * Xout[4] + (float)(2.7347767) *
   Xout[5] + (float)(0.75959486) * Xout[6] +
   (float)(-0.18560021) * Xout[7] + (float)(-2.3659115) *
   Xout[8] + (float)(1.4201123) * Xout[9];
Xout[12] = tanh( Xout[12] );

/* Generating code for PE 3 in hidden layer */
Xout[13]  = (float)(0.7640292) + (float)(-0.7441029) *
   Xout[2] + (float)(-3.3137417) * Xout[3] +
   (float)(-2.9614735) * Xout[4] + (float)(0.68064672) *
   Xout[5] + (float)(-0.46757856) * Xout[6] +
   (float)(1.7393887) * Xout[7] + (float)(-4.7351685) *
   Xout[8] + (float)(4.3163881) * Xout[9];
Xout[13] = tanh( Xout[13] );

/* Generating code for PE 4 in hidden layer */
Xout[14]  = (float)(0.036794752) + (float)(0.80874187) *
   Xout[2] + (float)(-2.8342977) * Xout[3] +
   (float)(-1.6920253) * Xout[4] + (float)(-6.4490356) *
   Xout[5] + (float)(-2.2223001) * Xout[6] +
   (float)(-0.18939279) * Xout[7] + (float)(0.97373468) *
   Xout[8] + (float)(0.43927357) * Xout[9];
Xout[14] = tanh( Xout[14] );

/* Generating code for PE 5 in hidden layer */
Xout[15]  = (float)(1.8314953) + (float)(2.8692045) *
   Xout[2] + (float)(-3.1761947) * Xout[3] +
   (float)(4.1439939) * Xout[4] + (float)(-1.6364973) *
   Xout[5] + (float)(-1.4868186) * Xout[6] +
   (float)(-1.5667051) * Xout[7] + (float)(3.3063626) *
   Xout[8] + (float)(-2.7897489) * Xout[9];
Xout[15] = tanh( Xout[15] );
```

Figure 3. Network Description of Channel 7 Baseline (continued).

```
/* Generating code for PE 6 in hidden layer */
Xout[16]  = (float)(-1.8766325) + (float)(-0.3451767) *
   Xout[2] + (float)(0.41630313) * Xout[3] +
   (float)(-2.0144601) * Xout[4] + (float)(0.33625582) *
   Xout[5] + (float)(2.0043108) * Xout[6] +
   (float)(3.9398975) * Xout[7] + (float)(0.30485672) *
   Xout[8] + (float)(-1.915418) * Xout[9];
Xout[16] = tanh( Xout[16] );

/* Generating code for PE 0 in output layer */
Xout[17]  = (float)(-0.70065403) + (float)(0.87089986) *
   Xout[10] + (float)(-1.1258541) * Xout[11] +
   (float)(-0.067218259) * Xout[12] + (float)(-0.24012817)
   * Xout[13] + (float)(1.1665372) * Xout[14] +
   (float)(-1.9874921) * Xout[15] + (float)(-0.97608429)
   * Xout[16];
Xout[17] = tanh( Xout[17] );

/* Generating code for PE 1 in output layer */
Xout[18]  = (float)(-1.8466408) + (float)(-0.80945539) *
   Xout[10] + (float)(0.78324491) * Xout[11] +
   (float)(1.0169119) * Xout[12] + (float)(-0.1595639) *
   Xout[13] + (float)(0.27367693) * Xout[14] +
   (float)(-1.036254) * Xout[15] + (float)(-0.56164873) *
   Xout[16];
Xout[18] = tanh( Xout[18] );

/* Generating code for PE 2 in output layer */
Xout[19]  = (float)(-2.1759548) + (float)(-0.95814037) *
   Xout[10] + (float)(0.70946145) * Xout[11] +
   (float)(-1.2249976) * Xout[12] + (float)(1.2662531) *
   Xout[13] + (float)(-0.584782) * Xout[14] +
   (float)(1.8446044) * Xout[15] + (float)(0.13933684) *
   Xout[16];
Xout[19] = tanh( Xout[19] );

/* Generating code for PE 3 in output layer */
Xout[20]  = (float)(-1.1233367) + (float)(0.28520119) *
   Xout[10] + (float)(0.23571797) * Xout[11] +
   (float)(0.94641453) * Xout[12] + (float)(0.014802053)
   * Xout[13] + (float)(1.1706237) * Xout[14] +
   (float)(-0.36040393) * Xout[15] + (float)(0.060552772)
   * Xout[16];
Xout[20] = tanh( Xout[20] );
```

Figure 3. Network Description of Channel 7 Baseline (continued).

```
    /* Generating code for PE 4 in output layer */
    Xout[21]  = (float)(-1.8035799) + (float)(0.82990104) *
       Xout[10] + (float)(-0.39371634) * Xout[11] +
       (float)(0.3639448) * Xout[12] + (float)(0.020903151) *
       Xout[13] + (float)(-1.4780852) * Xout[14] +
       (float)(0.7810728) * Xout[15] + (float)(-0.64493746) *
       Xout[16];
    Xout[21] = tanh( Xout[21] );

    /* Generating code for PE 5 in output layer */
    Xout[22]  = (float)(-0.23103279) + (float)(0.35826457) *
       Xout[10] + (float)(0.016775791) * Xout[11] +
       (float)(-0.26488703) * Xout[12] + (float)(0.28906751)
       * Xout[13] + (float)(0.14797242) * Xout[14] +
       (float)(-0.32592633) * Xout[15] + (float)(1.0858428) *
       Xout[16];
    Xout[22] = tanh( Xout[22] );

    /* De-scale and write output from network */
    Yout[0] = Xout[17] * (0.62499999) + (0.5);
    Yout[1] = Xout[18] * (0.62499999) + (0.5);
    Yout[2] = Xout[19] * (0.62499999) + (0.5);
    Yout[3] = Xout[20] * (0.62499999) + (0.5);
    Yout[4] = Xout[21] * (0.62499999) + (0.5);
    Yout[5] = Xout[22] * (0.62499999) + (0.5);
    return( 0 );
}
```

Figure 3. Network Description of Channel 7 Baseline (continued).

```
/*   Channel 5+6+7 Combined Baseline Neural Network
     (Perceptron) */

#if __STDC__
#define  ARGS(x) x
#else
#define  ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double tanh  ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES ***
*/

#if __STDC__
int NN_GBX567( void *NetPtr, float Yin[8], float Yout[6] )
#else
int NN_GBX567( NetPtr, Yin, Yout )
void  *NetPtr; /* Network Pointer (not used) */
float Yin[8], Yout[6];  /* Data */
#endif /* __STDC__ */
{
    float  Xout[23];  /* work arrays */
    long   ICmpT;  /* temp for comparisons */

    /* Read and scale input into network */
    Xout[2] = Yin[0] * (0.11925415) + (8.4536229);
    Xout[3] = Yin[1] * (0.032063101) + (3.9165558);
    Xout[4] = Yin[2] * (0.028125439) + (3.8805512);
    Xout[5] = Yin[3] * (0.019380783) + (2.5365472);
    Xout[6] = Yin[4] * (0.023219095) + (4.5342714);
    Xout[7] = Yin[5] * (0.024189645) + (4.2190369);
    Xout[8] = Yin[6] * (0.024215131) + (4.3061162);
    Xout[9] = Yin[7] * (0.026888224) + (5.1941597);
LAB110:

    /* Generating code for PE 0 in hidden layer */
    Xout[10]  = (float)(2.1650488) + (float)(0.51765656) *
       Xout[2] + (float)(-0.93372732) * Xout[3] +
       (float)(-1.8701128) * Xout[4] + (float)(-1.1336367) *
       Xout[5] + (float)(-2.210844) * Xout[6] +
       (float)(-2.8568017) * Xout[7] + (float)(0.30550599) *
       Xout[8] + (float)(3.2518566) * Xout[9];
    Xout[10] = tanh( Xout[10] );
```

Figure 4. Network Description of Channel 5+6+7 Combined Baseline.

```
/* Generating code for PE 1 in hidden layer */
Xout[11]  = (float)(-1.303616) + (float)(-4.0838604) *
   Xout[2] + (float)(-2.9945533) * Xout[3] +
   (float)(4.9566259) * Xout[4] + (float)(3.1236155) *
   Xout[5] + (float)(0.31473091) * Xout[6] +
   (float)(1.2816585) * Xout[7] + (float)(-2.1291025) *
   Xout[8] + (float)(1.7792628) * Xout[9];
Xout[11] = tanh( Xout[11] );

/* Generating code for PE 2 in hidden layer */
Xout[12]  = (float)(1.1715755) + (float)(1.2604314) *
   Xout[2] + (float)(-3.568207) * Xout[3] +
   (float)(-1.3073902) * Xout[4] + (float)(-4.2037044) *
   Xout[5] + (float)(1.5924199) * Xout[6] +
   (float)(4.6907048) * Xout[7] + (float)(-2.5169041) *
   Xout[8] + (float)(2.3589554) * Xout[9];
Xout[12] = tanh( Xout[12] );

/* Generating code for PE 3 in hidden layer */
Xout[13]  = (float)(-0.48920253) + (float)(0.011894092) *
   Xout[2] + (float)(1.6359543) * Xout[3] +
   (float)(-1.5948308) * Xout[4] + (float)(-1.1471063) *
   Xout[5] + (float)(-4.4292927) * Xout[6] +
   (float)(0.66382504) * Xout[7] + (float)(-0.45270401) *
   Xout[8] + (float)(-1.0215354) * Xout[9];
Xout[13] = tanh( Xout[13] );

/* Generating code for PE 4 in hidden layer */
Xout[14]  = (float)(1.1854466) + (float)(2.3103707) *
   Xout[2] + (float)(-0.43266118) * Xout[3] +
   (float)(-1.1943052) * Xout[4] + (float)(-1.3579001) *
   Xout[5] + (float)(-3.9594347) * Xout[6] +
   (float)(-1.0741928) * Xout[7] + (float)(2.5075767) *
   Xout[8] + (float)(-3.5812092) * Xout[9];
Xout[14] = tanh( Xout[14] );

/* Generating code for PE 5 in hidden layer */
Xout[15]  = (float)(-1.5507644) + (float)(2.6392918) *
   Xout[2] + (float)(1.3269755) * Xout[3] +
   (float)(-2.551749) * Xout[4] + (float)(-3.6663303) *
   Xout[5] + (float)(-2.3318245) * Xout[6] +
   (float)(-1.4067193) * Xout[7] + (float)(-0.18262224) *
   Xout[8] + (float)(2.4798441) * Xout[9];
Xout[15] = tanh( Xout[15] );
```

Figure 4. Network Description of Channel 5+6+7 Combined Baseline (continued).

```
/* Generating code for PE 6 in hidden layer */
Xout[16]  = (float)(0.68331832) + (float)(1.9184027) *
   Xout[2] + (float)(3.0481825) * Xout[3] +
   (float)(-5.5359893) * Xout[4] + (float)(-0.30242923) *
   Xout[5] + (float)(-1.4136549) * Xout[6] +
   (float)(0.47077999) * Xout[7] + (float)(-2.2177608) *
   Xout[8] + (float)(2.4806352) * Xout[9];
Xout[16] = tanh( Xout[16] );

/* Generating code for PE 0 in output layer */
Xout[17]  = (float)(-0.67697048) + (float)(-0.21998633) *
   Xout[10] + (float)(-1.4980452) * Xout[11] +
   (float)(-0.89661926) * Xout[12] + (float)(1.2544708) *
   Xout[13] + (float)(-0.20170347) * Xout[14] +
   (float)(0.89252609) * Xout[15] + (float)(-0.41504142)
   * Xout[16];
Xout[17] = tanh( Xout[17] );

/* Generating code for PE 1 in output layer */
Xout[18]  = (float)(-0.97790527) + (float)(0.67401952) *
   Xout[10] + (float)(1.4429724) * Xout[11] +
   (float)(-0.0509957) * Xout[12] + (float)(-0.19721894)
   * Xout[13] + (float)(-0.78762019) * Xout[14] +
   (float)(0.74004883) * Xout[15] + (float)(1.8177283) *
   Xout[16];
Xout[18] = tanh( Xout[18] );

/* Generating code for PE 2 in output layer */
Xout[19]  = (float)(-2.9606795) + (float)(0.84440678) *
   Xout[10] + (float)(-0.91007847) * Xout[11] +
   (float)(0.83775264) * Xout[12] + (float)(-0.75925416)
   * Xout[13] + (float)(0.94593042) * Xout[14] +
   (float)(-1.2261925) * Xout[15] + (float)(0.28896332) *
   Xout[16];
Xout[19] = tanh( Xout[19] );

/* Generating code for PE 3 in output layer */
Xout[20]  = (float)(-2.0208859) + (float)(0.8019042) *
   Xout[10] + (float)(1.2120522) * Xout[11] +
   (float)(0.6426028) * Xout[12] + (float)(-0.51600492) *
   Xout[13] + (float)(0.86542255) * Xout[14] +
   (float)(0.26224795) * Xout[15] + (float)(-0.38298726)
   * Xout[16];
Xout[20] = tanh( Xout[20] );
```

Figure 4. Network Description of Channel 5+6+7 Combined Baseline (continued).

```
    /* Generating code for PE 4 in output layer */
    Xout[21]  = (float)(-0.54617912) + (float)(-0.1051699) *
       Xout[10] + (float)(0.062350694) * Xout[11] +
       (float)(-1.1998838) * Xout[12] + (float)(-0.6690836) *
       Xout[13] + (float)(0.42717206) * Xout[14] +
       (float)(0.48723215) * Xout[15] + (float)(-0.83237219)
       * Xout[16];
    Xout[21] = tanh( Xout[21] );

    /* Generating code for PE 5 in output layer */
    Xout[22]  = (float)(-0.59073412) + (float)(-1.4769753) *
       Xout[10] + (float)(-0.16596904) * Xout[11] +
       (float)(0.72467691) * Xout[12] + (float)(0.20691243) *
       Xout[13] + (float)(-0.10412298) * Xout[14] +
       (float)(-0.20468295) * Xout[15] + (float)(0.13812968)
       * Xout[16];
    Xout[22] = tanh( Xout[22] );

    /* De-scale and write output from network */
    Yout[0] = Xout[17] * (0.62499999) + (0.5);
    Yout[1] = Xout[18] * (0.62499999) + (0.5);
    Yout[2] = Xout[19] * (0.62499999) + (0.5);
    Yout[3] = Xout[20] * (0.62499999) + (0.5);
    Yout[4] = Xout[21] * (0.62499999) + (0.5);
    Yout[5] = Xout[22] * (0.62499999) + (0.5);
    return( 0 );
}
```

Figure 4. Network Description of Channel 5+6+7 Combined Baseline (continued).

Table IX. Network Classifier Performance Summary

		CHANNEL			
		5	6	7	5+6+7 COMBINED
FAULT CASE	1	99.5	94.5	92.5	91.0
	2	97.5	100.0	97.5	93.0
	3	94.5	97.0	90.0	86.8
	4	98.0	100.0	96.0	98.0
	5	99.0	96.0	100.0	96.3
	6	100.0	95.0	99.0	93.0
	ALL	97.8	97.1	95.0	92.1
GOOD/BAD GEARBOX		98.4	98.8	96.0	96.3

Note: Values indicate % correct classification.

Table X. Best Case Network Classifier Performance

		CHANNEL			
		5	6	7	5+6+7 COMBINED
FAULT CASE	1	99.5	-	-	-
	2	-	100.0	-	-
	3	-	97.0	-	-
	4	-	100.0	-	-
	5	-	-	100.0	-
	6	100.0	-	-	-
	ALL	97.8	-	-	-
GOOD/BAD GEARBOX		-	98.8	-	-

Note: Values indicate % correct classification.

VIII.

REFERENCES

[1] Hecht-Nielsen, R., NEURALCOMPUTING, Addison-Wesley Publishing Co, Menlo Park, California , 1991.

[2] Hollins, M.L., "The Effects of Vibration Sensor Location in Detecting Gear and Bearing Defects," Technical Report, Naval Air Test Center, Patuxent River, MD, 1986.

[3] Hwang, J-N., Vlontzos, J.A., and Kung, S-Y., "A Systolic Neural Network Architecture for Hidden Markov Models," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 37 No. 12, pp 1967-1979, December 1989.

[4] Jurik, M., "Introduction to Neural Network Algolrithm," EW Technical Issues 90-10, Journal of Electronic Defense, Vol. 13 No. 10, October 1990.

[5] Khotanazad, A. and Lu, J-H., "Classification of Invariant Image Representations Using a Neural Network," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 38 No. 6, pp 1028-1038, June 1990.

[6] Lippman, R.P., "An Introduction to Computing With Neural Nets," IEEE Acoustics, Speech, and Signal Processing Magazine, Vol. 4 No. 2, pp 4-22, April 1987.

[7] Murray, J.J., "Machinery Condition Monitoring Using the CMAC Neural Network," Unpublished Industrial Working Document.

[8] ---, NEURAL COMPUTING, NeuralWorks Professional II/PLUS and NeuralWorks Explorer Programming Tutorial, NeuralWare Inc, 1991.

APPENDIX A

NEURAL COMPUTING

INTRODUCTION

"Neuralcomputing is the technological discipline concerned with information processing systems (that is, neural networks) that autonomously develop operational characteristics in adaptive response to an information environment." [1] It is a computer model which attempts to match the functionality of the brain based on our present understanding of biological nervous systems. It is based on the non-linear transfer function of the synaptic neural connections.

This approach to information processing differs from programmed computing in that it does not require algorithm (rule) development. Thus, it can be used for that type of problem where the algorithms are not known or would not be conducive to development.

The basic structure of neural computing is the neural network. This is an array of many simple non-linear processing elements or nodes connected by links with variable

weights to form a parallel processing structure of the Multiple Instruction Multiple Data (MIMD) type. The array of nodes creates a distributed memory structure which provides for a graceful degradation of performance in case of node failure.

The neural network is trained rather than logic programmed. Problem class examples are presented to the network and from them the network learns to abstract the underlying statistics of the problem class. Learning takes the form of adjusting the individual nodal link weights. These link weights are adjusted through a training/testing program to the point where the network output is within acceptable limits and then the neural network is declared production ready. It is the development of an efficient error correcting concept for the training program that has precipitated the resurgence of neural computing research and development.

Neural networks generally take the form of a software-callable procedure or subroutine within an algorithmic program. The network solves the core problem for the program. The program gathers and preprocesses the data, and then uses the output in further operations.

Real world applications of neural computing may be found in the areas of defense, finance, insurance, and the automobile industry. Of these most have been accomplished by application area or domain experts rather than by neural computing experts. Since only a relatively small set of neural networks have been adequately characterized, one normally finds that it is the domain expert who recognizes what problems can be solved with which neural network architecture.

The fields of expertise for neural networks are those based on the recognition of patterns within a data class. Thus, applications may be found in sensor processing, data analysis, control, and, of course, pattern recognition or classification. Neural network applications are also found in signal processing, image or data compression, and functional synthesis areas. Reference [5] discusses the use of a neural network in the classification of images while reference [3] develops the use of the neural network in hidden Markov model processes.

NETWORK ARCHITECTURE

The neural network is an array structure in the form of a directed graph. It consists of a set of nodes connected by

a set of directed line segments. Figure 5 [4] illustrates a typical neural network. The nodes of the network are the array processing elements and the links are the unidirectional connections joining them. The array columns of processing elements are referred to as the network layers (some authors call them slabs rather than layers). The network input is not considered to be a layer when counting the number of network layers since it does not contain processing elements. Those layers not directly accessible from outside the network are referred to as hidden layers. The network illustrated in Figure 5 has two layers, one of which is a hidden layer.

Thus, the typical neural network consists of a number of processing elements arranged into a sequence of layers with unidirectional links between the nodes of the successive layers. The connections may be full or random. That is, a node in one layer may be connected to each node in the previous layer or the nodal connections may be random. In addition, in some structures (e.g., competing structures) the nodes of a layer may also be connected to the other nodes of that layer. While there is no theoretical limit to the number of nodes in a network, in a practical sense one would not expect to see a network with more than a hundred or so nodes or more than three layers.

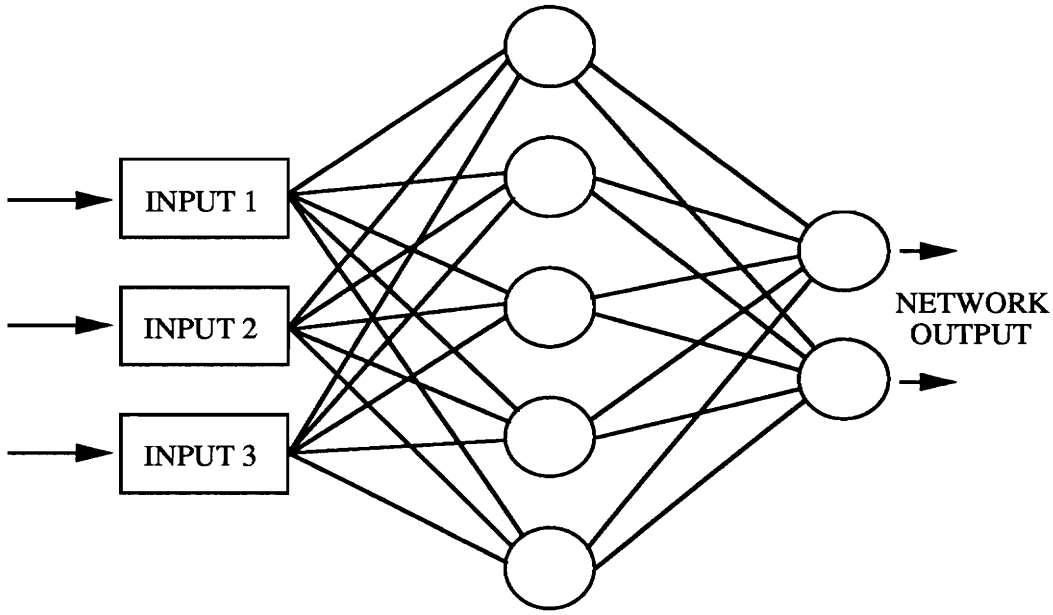


Figure 5. A Two-Layer Neural Network, Fully Connected.

The processing element of the network is patterned after the synapse-neuron connection of a biological nervous system. It receives signals from various sources, operates on the collection of input signals, and provides a single transformed output which may fan out as input signals to other processing elements. This operation may be continuous or episodic, triggered by an activate signal. A schematic representation of a processing element is shown in Figure 6. [1]

The operation performed by the processing element is usually that of taking the weighted sum of the inputs and transforming this sum through some transfer function. This is illustrated in Figure 7. [8] (The notation used is representative of that in the literature, but by no means universal.) The weight applied to each input represents the strength of that synaptic connection. The weights are set at random values ($-1 \leq w \leq 1$) and adjusted during the training cycle.

The transfer function is nonlinear. It could be a threshold logic element or a hard limiter. [6] It is commonly the sigmoid (0,1), the hyperbolic tangent (-1,1), or the modified hyperbolic tangent (-1,1) as presented below. It could also be the sine function.

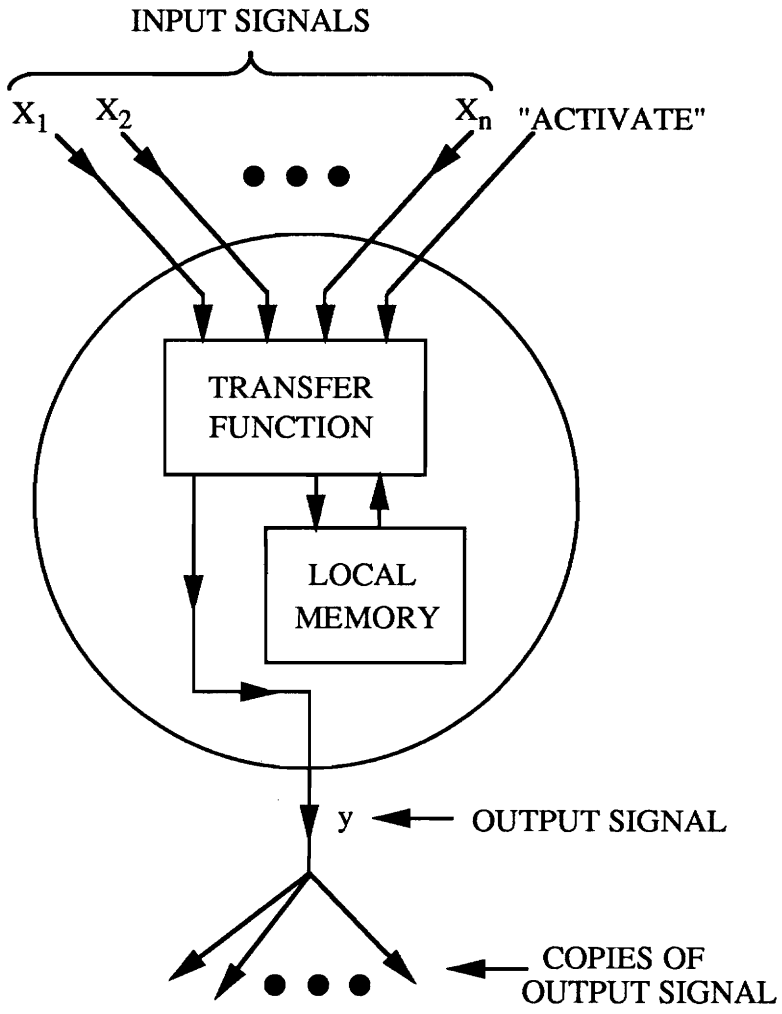
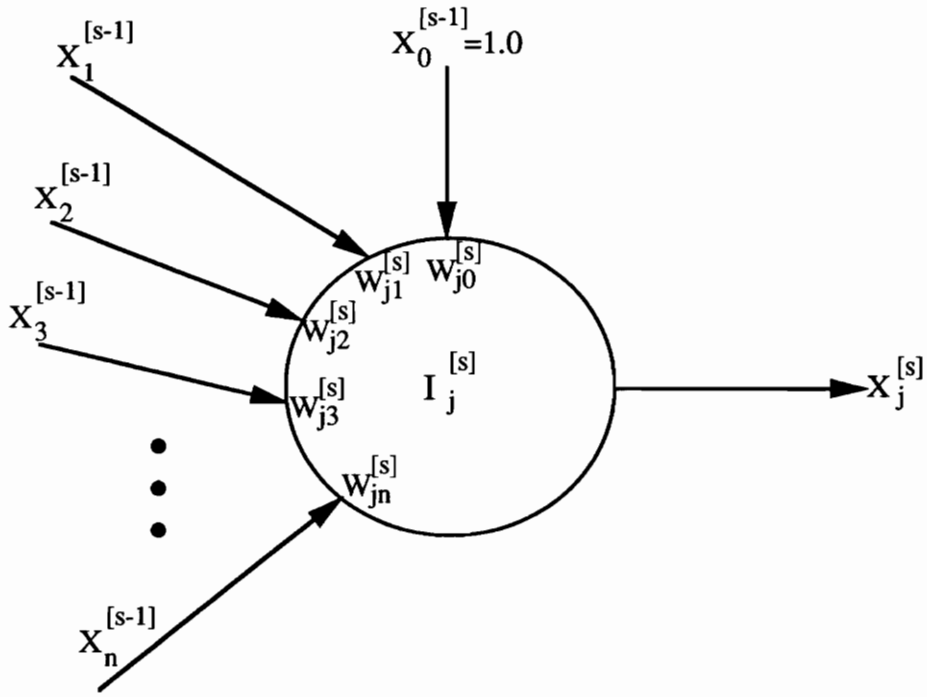


Figure 6. Schematic Representation of a Processing Element.



$x_j^{[s]}$ current output of jth neuron in layer s
$w_{ji}^{[s]}$ connection weight joining $x_i^{[s-1]}$ to $x_j^{[s]}$
$I_j^{[s]}$ weighted input sum to jth neuron in layer s

$$x_j^{[s]} = f\left[\sum_i \left(w_{ji}^{[s]} * x_i^{[s-1]}\right)\right]$$
$$= f\left(I_j^{[s]}\right)$$

Figure 7. Weighted Sum Processing Element

$$\text{Sigmoid: } f(z) = \frac{1}{1 + e^{-z}}$$

$$\text{Hyperbolic tangent: } f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\text{Modified hyperbolic tangent: } f(z) = \frac{e^{z} - 1}{e^{z} + 1}$$

NETWORK OPERATION

NEURAL COMPUTING [8] discusses several different applications, characterized by different neural networks. Some of these applications are Adaline and Madaline, Adaptive Resonance Theory, Bi-Directional Associative Memory, Brain-State-In-A-Box, Delta-Bar-Delta, Hamming, Hopfield, Perceptron, Self-Organizing Maps, Spatio-Temporal Pattern Recognition. All are interesting network applications. This discussion addresses the multi-layer Perceptron network, an extension of the original neural computing project, the Perceptron by Rosenblatt in 1957.

The perceptron network is a pattern classifier of the Gaussian maximum likelihood type except that it makes no assumptions concerning the shape of the underlying distributions. It focuses instead on the output error when

distributions overlap. The network can be trained to classify patterns of inputs, but the classes must be linearly separable. This is illustrated by Figure 8. [1] In this illustration the two classes are separated by a hyperplane determined by the nodal connection weight vector w. The linearly separable requirement may be removed if a multi-layer perceptron network is used. It can be proved by construction [6] that the three-layer perceptron can form arbitrarily complex decision regions when hard limiting, sigmoidal, or other non-linear transfer functions are used. The network can form regions as complex as those formed by nearest neighbor classifiers. Figure 5 represents a two-layer perceptron network. It is a fully connected, feed forward network.

Lippmann [6] gives some general concepts for determining the number of nodes to use in the network. He considers the number of nodes in the second layer in the worst case to be equal to the number of disconnected regions of the input distribution. The number of nodes in the first layer must typically provide three or more edges for each convex area generated by each second layer node. Jurik [4] advocates a somewhat different guideline for his signal classifier network example. He has an output node for each class and a "few" cells in the hidden layer. He then adjusts the number in this

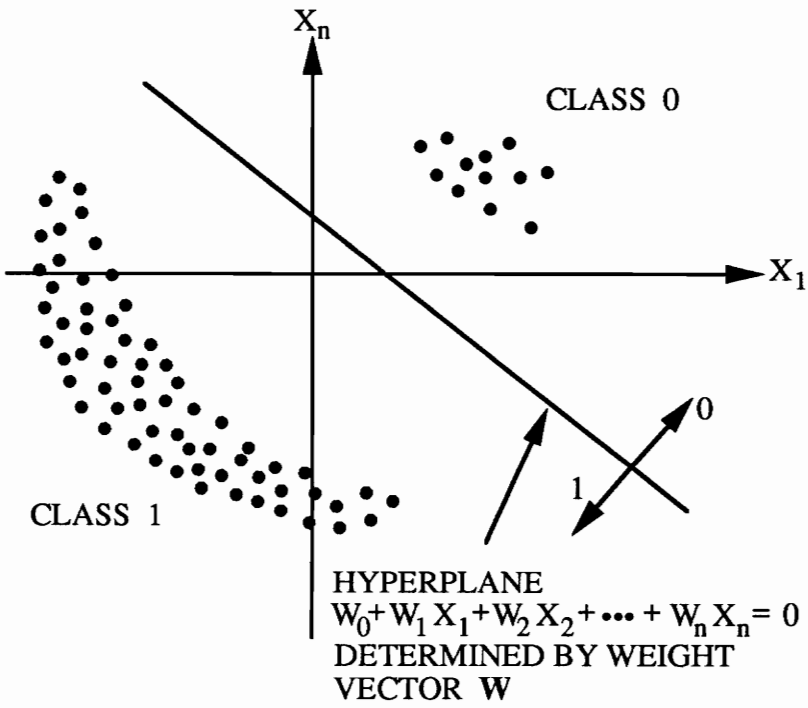


Figure 8. The Classification Problem

layer by trial. "If after 2000 cycles the network's output is still pretty bad, add another cell to the middle layer and try again."

The training cycle consists of presenting a training set example to the network input, determining the output error, adjusting the weights of the processing elements to reduce the error, and repeating the cycle through the training set. This is continued until the training set output is within an acceptable error range. The network is then tested with the test set. If the results are not acceptable, the network is modified and the cycle repeated. The technique used to reduce the output error, to adjust the processing element weights, in the multi-layer perceptron is called the "back-propagation" algorithm.

Back-propagation is a generalization of the least mean square algorithm. A descending gradient search technique is used to minimize the mean square difference between the desired and actual outputs. The global error is assumed to be distributed throughout the network in accordance with the individual processing element connecting weights, the local errors. The output error is back propagated through the network and used to adjust the nodal connection weights.

Assuming the global error to be E, the local node error is [8]

$$e_j^{[s]} = \partial E / \partial I_j^{[s]}$$

This may be expressed as

$$e_j^{[s]} = f'(I_j^{[s]}) * \sum_k \left(e_k^{[s+1]} * w_{kj}^{[s+1]} \right)$$
$$= x_j^{[s]} * (1 - x_j^{[s]}) * \sum_k \left(e_k^{[s+1]} * w_{kj}^{[s+1]} \right)$$

where the non-linear transfer function is the sigmoid. This is analogous to the summation term (Figure 7) used to forward propagate the input signal. In this case the [s+1] layer term is used instead of the [s-1] layer term.

The local processing element weight adjustment is

$$\Delta w_{ji}^{[s]} = -\beta * (\partial E / \partial w_{ji}^{[s]})$$

where Beta is the learning coefficient. This may be expressed as

$$\Delta w_{ji}^{[s]} = \beta * e_j^{[s]} * x_i^{[s-1]}$$

A different expression is needed for the local error of the output layer processing elements since it has no higher layer. If the least mean square is used as the error measure, the error is

$$E = 0.5 * \sum_k \left((d_k - o_k)^2 \right)$$

where k indexes the components of the desired (d) and observed (o) outputs. The local error for the output layer processing elements is given by

$$e_k^o = (d_k - o_k) * f'(I_k)$$

The back-propagation process may be speeded up by modifying the delta weight adjustment equation. One method is to add a small positive offset to the derivative of the sigmoid prior to scaling the local error, if scaling is used. This helps to prevent saturation, where learning would effectively cease. Another method is to add a momentum factor to the delta weight equation.

This changes the equation to

$$\Delta w_{ji}^{[s]} = \beta * e_j^{[s]} * (x_i^{[s-1]} + k * e_i^{[s-1]})$$

where k is the momentum term. Jurik [4] mentions that a more advanced algorithm is currently available called Back-Percolation. This is supposed to train networks 10 to 100 times faster than back-propagation. He does not, however, discuss it, but merely states that it is a more complex algorithm. This algorithm is not available in the NeuralWare Professional II/Plus program.

CONCLUSION

As Hecht-Nielsen points out in NEURALCOMPUTING [1] neural computing is a fundamentally different approach to information processing than that of programmed computing. The former is based on transformations and the latter on algorithms and rules. They are operationally complementary, but conceptually incompatible. Neural computing systems operate on information in a totally different way than the algorithmic, rule based way of programmed computer systems which includes artificial intelligence programming. The introduction of neural networks into the problem solution has often led to a major reduction

in overall system development time and, consequently, expense.

The neural computing field is still in its infancy. Some neural network capabilities and properties can be proven, but there is no conceptual understanding of just how the neural network does its job. There is much yet to be discovered and developed.