

# **Automation of Engineering Design Aids using Neural Networks**

Siripong Malasri & Jittapong Malasri  
Christian Brothers University  
Memphis, Tennessee

Kriangsiri Malasri  
Georgia Institute of Technology  
Atlanta, Georgia

## **ABSTRACT**

Often in engineering design work, engineers refer to design aids such as look-up tables and graphical plots. In many cases, this information is obtained from experimental data for which no simple mathematical relationship is available. However, using such design aids is error-prone as well as difficult to smoothly integrate with other computerized applications.

Neural networks have been shown to be able to recognize patterns and have been applied towards projecting trends in data. This work investigates the use of a neural network for generating accurate input-output relationships based on reference design aids. It shows how one can go from a hard copy of a design chart to network training to stand-alone software. A stress concentration design aid is used as an example to demonstrate the procedure.

## **INTRODUCTION**

Engineering design work often requires the use of design aids such as look-up tables and graphical plots. These tables and plots are created from experimental data for which no simple mathematical relationship is available. Reading values from these tables/plots typically involves interpolation of the available data, which can be time-consuming and error-prone. In addition, it is difficult to smoothly integrate these design aids with other computerized applications.

Artificial neural networks have been used widely to recognize patterns and have been applied towards projecting trends in data. A simple example [1] to demonstrate the ability of a neural network to interpolate and extrapolate data is shown in Figure 1 and Table 1. This example shows that a neural network after proper training is able to fit a curve through a set of data points, to interpolate values between data points, and to extrapolate values beyond the given set of data points. The example was performed using an in-house computer program called BACKPROP, which is based on the commonly used backpropagation network model.

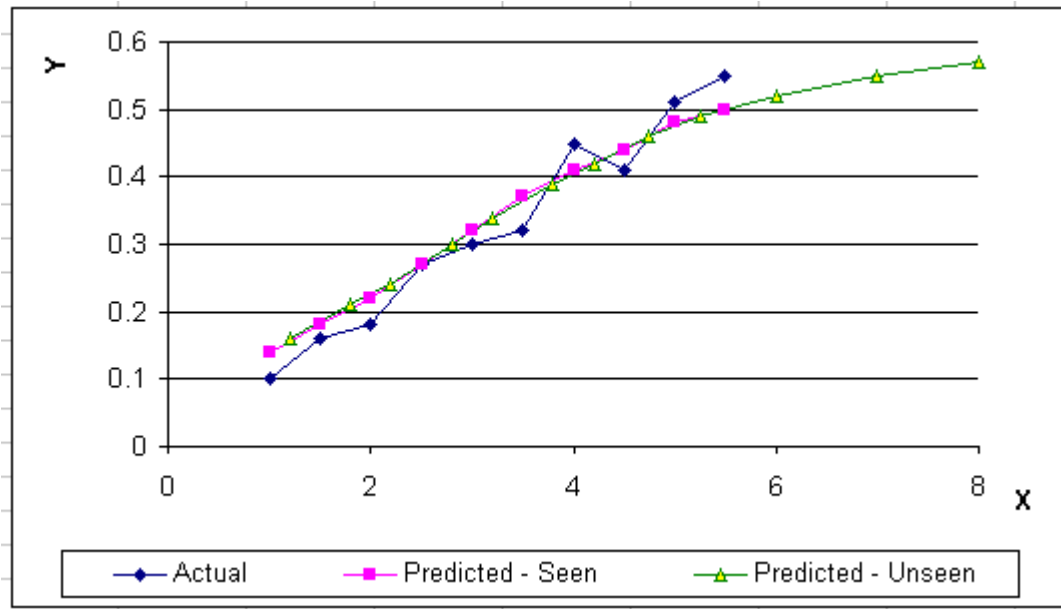


Figure 1. Trend determination, interpolation, and extrapolation using a backpropagation neural network.

Table 1. Data used in Figure 1.

Point	X	Y	BP Y	X	BP Y
1	1	0.1	0.14	1.2	0.16
2	1.5	0.16	0.18	1.8	0.21
3	2	0.18	0.22	2.2	0.24
4	2.5	0.27	0.27	2.8	0.3
5	3	0.3	0.32	3.2	0.34
6	3.5	0.32	0.37	3.8	0.39
7	4	0.45	0.41	4.2	0.42
8	4.5	0.41	0.44	4.75	0.46
9	5	0.51	0.48	5.25	0.49
10	5.5	0.55	0.5	6	0.52
				7	0.55
				8	0.57

Notes:

- $Y = \text{Actual } Y$
- $BP \ Y = \text{Computed } Y \text{ from a trained backpropagation network}$
- $X$  and  $Y$  values in the 2<sup>nd</sup> and 3<sup>rd</sup> columns are the training data set
- $BP \ Y$  values in the 4<sup>th</sup> column are predicted by a trained network
- $X$  and  $BP \ Y$  values in the 5<sup>th</sup> and 6<sup>th</sup> (last two) columns demonstrate the ability of the network to perform interpolation and extrapolation.

Using a similar technique as the above example, data can be read from a graphical design aid and a neural network can be trained to recognize the data pattern. This paper provides a step-by-step methodology for creating such an automated engineering design aid, using the calculation of notched beam stress concentration as an example.

## STRESS CONCENTRATION

When a structural member changes its cross section abruptly due to cuts, holes, etc., stress near the deformations increases significantly due to the concentration of stress in that area. Several methods can be used to quantitatively determine these stresses, including finite-element analysis, experimental procedures, and determining a stress concentration factor ( $C$ ) from a graphical design aid. Typical design engineers would choose the latter method if a design aid is available, due to its simplicity. Design aids have been developed for many situations and appear in various handbooks and textbooks. In this paper, a notched-beam stress concentration graphical design aid [2] was used as an example. Other design aids can be implemented in the same manner.

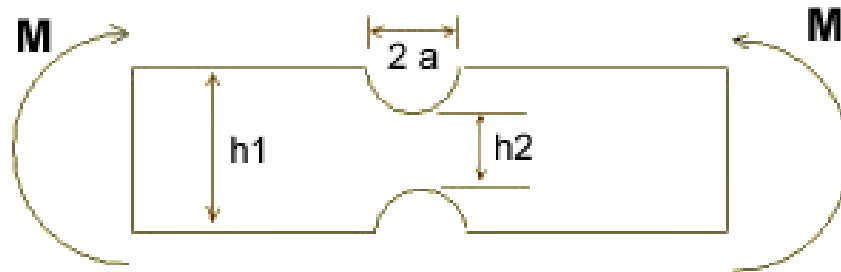


Figure 2. Notched beam.

Figure 2 shows a notched beam. The peak stresses at the two notches can be determined from the relation

$$\sigma = C \frac{Mc}{I}$$

where the peak bending stress  $\sigma$  is calculated using the bending moment  $M$ , distance from neutral axis  $c$ , cross-sectional moment of inertia  $I$ , and stress concentration factor  $C$ . For a notched beam,  $C$  is a function of the ratios  $a/h_2$  and  $h_1/h_2$ , where  $a$ ,  $h_1$ , and  $h_2$  denote the dimensions labeled in Figure 2.

## STEP-BY-STEP METHODOLOGY

### Data Preparation

Data was obtained using a stress concentration graphical design aid from a textbook [2], similar to the one reproduced in Figure 3. This plot gives the value of the stress concentration factor  $C$  (output) as a function of two inputs,  $a/h_2$  and  $h_1/h_2$ . As shown in Appendix A, a training data set was created from this information. Input-output pairs with “T” in the “Mark” column are used for training, while those with “C” and “V” are used for calibration and validation purposes. The calibration set is used during a training session to see how well the network has learned. It can also be used as the criterion to stop a training session. The validation set is used to evaluate the trained network’s performance. For this example, there are a total of 46 training pairs, 15 calibration pairs, and 15 validation pairs.

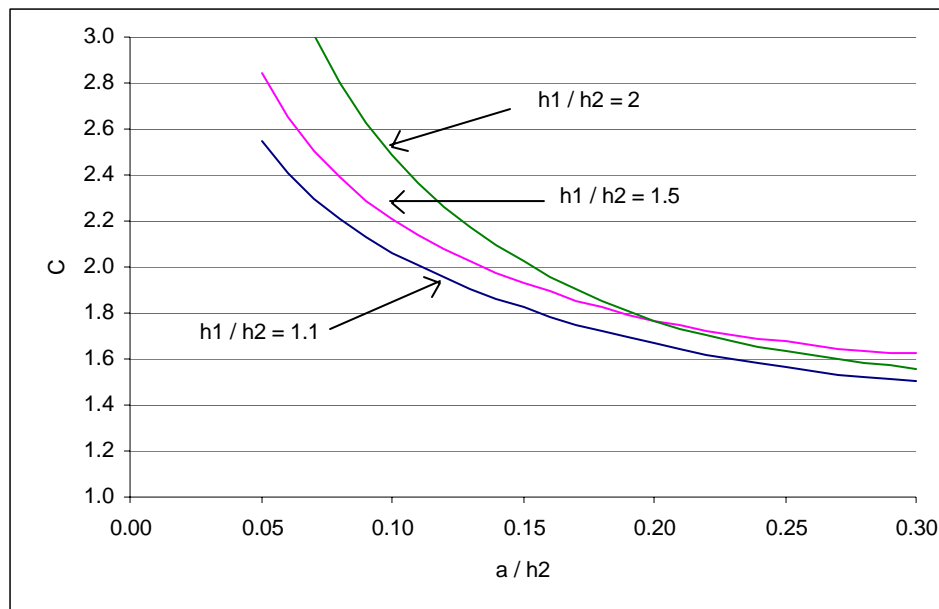


Figure 3. Example of a graphical design aid for determining the stress concentration factor  $C$  for a notched beam.

### Neural Network Formulation, Training, & Validation

The NeuroShell 2 software [3] was used to create and train a backpropagation network to reproduce the graphical relationship of Figure 3. Appendix B shows screenshots of the important steps in generating a network for this notched beam problem. As shown in Figure B.5, the network consists of 2 input neurons ( $a/h_1$  and  $h_1/h_2$ ), 8 hidden neurons, and 1 output neuron ( $C$ ). The training data was presented to the network on a rotational basis, with the network that gets the best results saved during the calibration. Figures B.6 and B.7 show excellent results from the network, with both  $R^2$  and  $r^2$  almost 1 and all

computed outputs within 5% of the actual values. Table B.1 compares the computed output from the network with the actual outputs; no significant errors can be found.

### Visual Basic Stand-alone Application Development

NeuroShell 2 can generate source code in C, Visual Basic, or a generic format for use with calculators, as shown in Appendix C. Visual Basic .NET [4] was used to create a user interface with two input fields ( $h_1/h_2$  and  $a/h_2$ ) and one output field ( $C$ ), as shown in Figure 4. The source code generated from NeuroShell 2 was attached to the “Start” button. A few lines of code were added to allow the program to read input from the two input fields and to direct the output to the output field. This stand-alone application can be obtained from the “Downloads” section at <http://www.cbu.edu/~pong/ce213/>. If an error message indicating a missing DLL file appears, the user probably needs to update Windows from Microsoft’s site at <http://windowsupdate.microsoft.com/>. Instructions on updating the OS are also given on the former page.

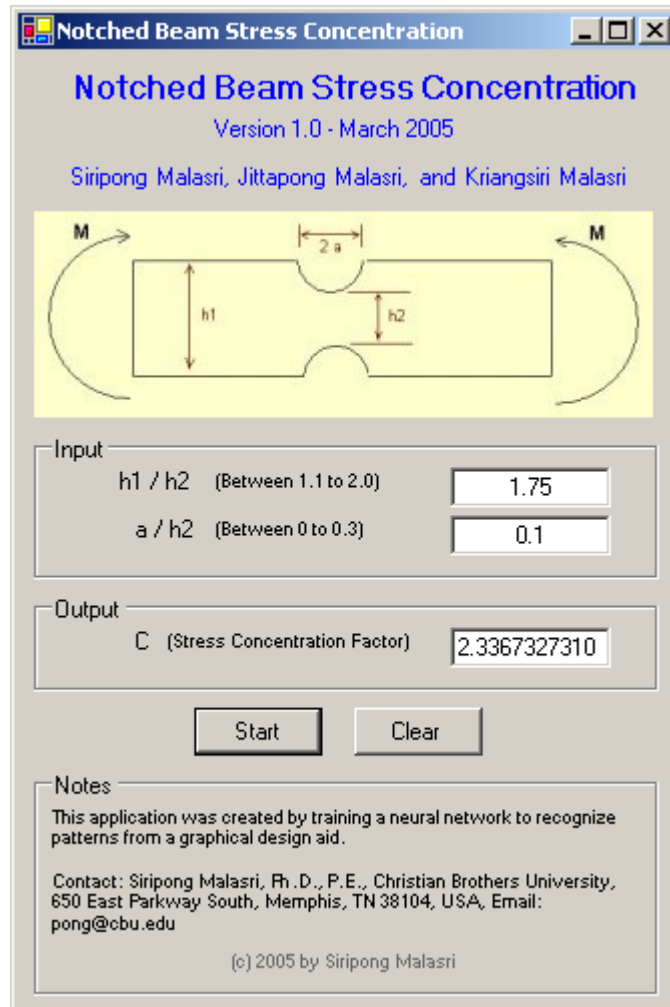


Figure 4. Stand-alone Visual Basic application.

## DISCUSSION & CONCLUSION

A backpropagation neural network can be easily trained for this particular problem and gives excellent results. NeuroShell 2's ability to generate source code in C or Visual Basic is extremely helpful in developing a separate stand-alone application. The application is an executable that can be run on any Windows computer without requiring the user to have a copy of NeuroShell 2, given that the operating system is updated with the needed components.

The generated source code could be inserted into a more comprehensive stress analysis computer program, which would create a seamless interface with that particular application. Ideally, such a stress analysis program would allow the user to easily handle a wide variety of cases without having to consult a traditional design aid at all.

In the future, the authors plan to train more networks to cover as many stress concentration cases as can be found in the literature.

## REFERENCES

1. Siripong Malasri and Kriangsiri Malasri, *Artificial Intelligence*, v3.0, digital class notes, 2003.
2. Anthony Bedford and Kenneth M. Liechti, *Mechanics of Materials*, Prentice-Hall, 2000.
3. *NeuroShell 2*, Release 4.0, Ward Systems Group, Inc., [www.wardsystems.com](http://www.wardsystems.com), 2000.
4. *Visual Basic .NET*, Microsoft Corp., [www.microsoft.com](http://www.microsoft.com), 2003.

## APPENDIX A - Training Data

Table A.1. Training data obtained from a graphical plot and tabulated for NeuroShell 2.

<b>h1/h2</b>	<b>a/h2</b>	<b>C</b>	<b>Mark</b>
1.10	0.05	2.55	T
1.10	0.06	2.41	T
1.10	0.07	2.30	T
1.10	0.08	2.21	C
1.10	0.09	2.13	V
1.10	0.10	2.06	T
1.10	0.11	2.00	T
1.10	0.12	1.95	T
1.10	0.13	1.91	C
1.10	0.14	1.86	V
1.10	0.15	1.82	T
1.10	0.16	1.79	T
1.10	0.17	1.75	T
1.10	0.18	1.72	C
1.10	0.19	1.69	V
1.10	0.20	1.67	T
1.10	0.21	1.64	T
1.10	0.22	1.62	T
1.10	0.23	1.60	C
1.10	0.24	1.58	V
1.10	0.25	1.57	T
1.10	0.26	1.55	T
1.10	0.27	1.53	T
1.10	0.28	1.52	C
1.10	0.29	1.51	V
1.10	0.30	1.51	T
1.50	0.05	2.84	T
1.50	0.06	2.65	T
1.50	0.07	2.51	C
1.50	0.08	2.39	V
1.50	0.09	2.29	T
1.50	0.10	2.21	T
1.50	0.11	2.14	T
1.50	0.12	2.08	C
1.50	0.13	2.03	V
1.50	0.14	1.98	T
1.50	0.15	1.93	T
1.50	0.16	1.89	T

<b>h1/h2</b>	<b>a/h2</b>	<b>C</b>	<b>Mark</b>
1.50	0.17	1.86	C
1.50	0.18	1.82	V
1.50	0.19	1.79	T
1.50	0.20	1.77	T
1.50	0.21	1.75	T
1.50	0.22	1.73	C
1.50	0.23	1.71	V
1.50	0.24	1.69	T
1.50	0.25	1.67	T
1.50	0.26	1.66	T
1.50	0.27	1.65	C
1.50	0.28	1.63	V
1.50	0.29	1.63	T
1.50	0.30	1.63	T
2.00	0.07	3.01	T
2.00	0.08	2.80	C
2.00	0.09	2.63	V
2.00	0.10	2.49	T
2.00	0.11	2.37	T
2.00	0.12	2.26	T
2.00	0.13	2.17	C
2.00	0.14	2.09	V
2.00	0.15	2.02	T
2.00	0.16	1.96	T
2.00	0.17	1.90	T
2.00	0.18	1.85	C
2.00	0.19	1.81	V
2.00	0.20	1.77	T
2.00	0.21	1.73	T
2.00	0.22	1.70	T
2.00	0.23	1.67	C
2.00	0.24	1.65	V
2.00	0.25	1.63	T
2.00	0.26	1.61	T
2.00	0.27	1.60	T
2.00	0.28	1.58	C
2.00	0.29	1.57	V
2.00	0.30	1.56	T

## APPENDIX B - Training a Network using NeuroShell 2

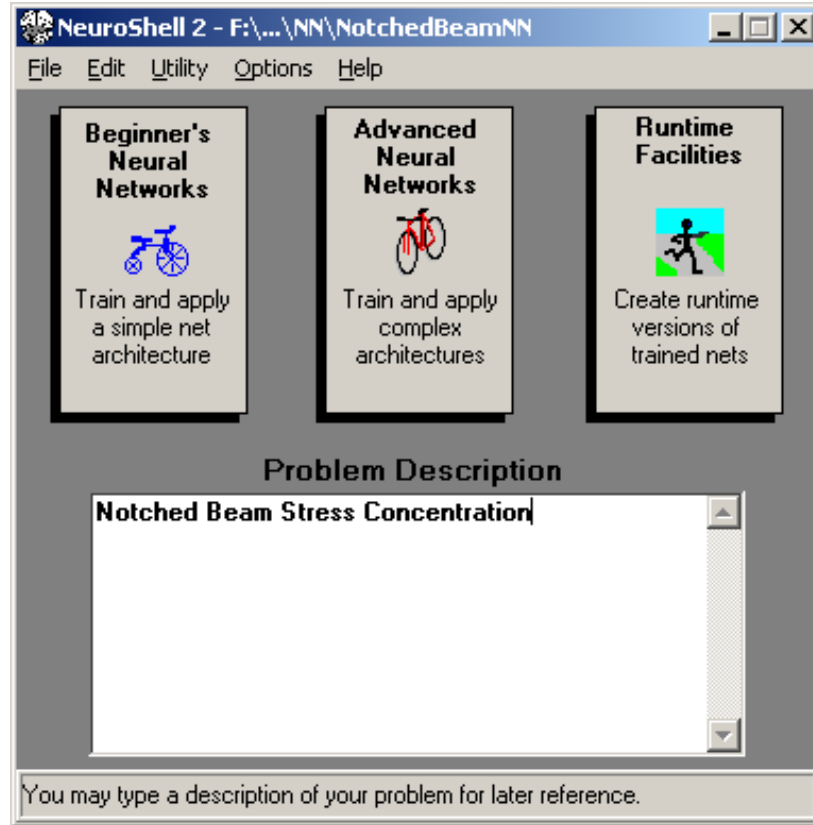


Figure B.1. Network creation. "Beginner's Neural Networks" was used in this example.

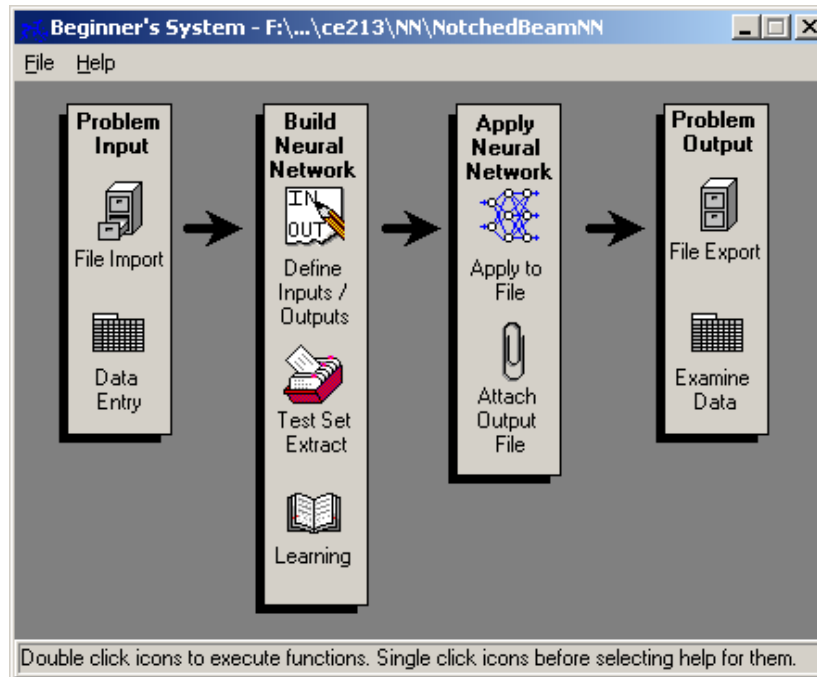


Figure B.2. Introduction of training data. "File Import" was used to import data from a spreadsheet file.

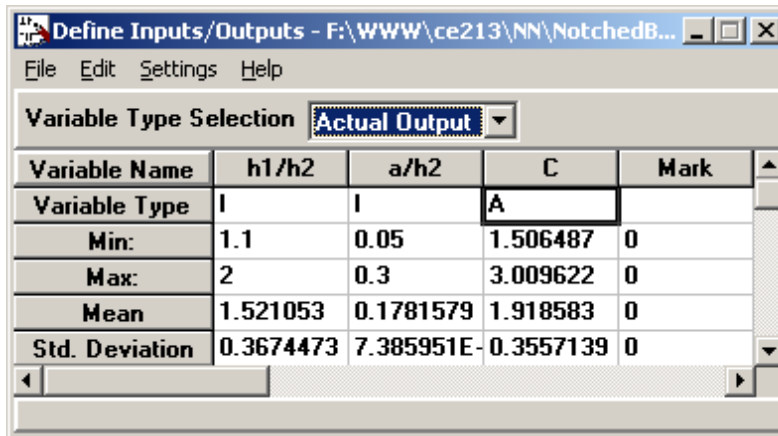


Figure B.3. Designating the data as inputs (I) or actual outputs (A).

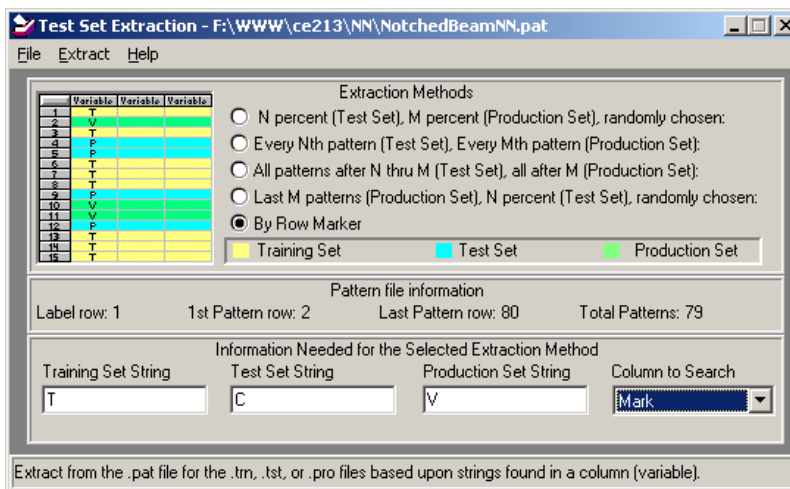


Figure B.4. Training, calibration, and validation data are identified with "T," "C," and "V" marks, respectively.

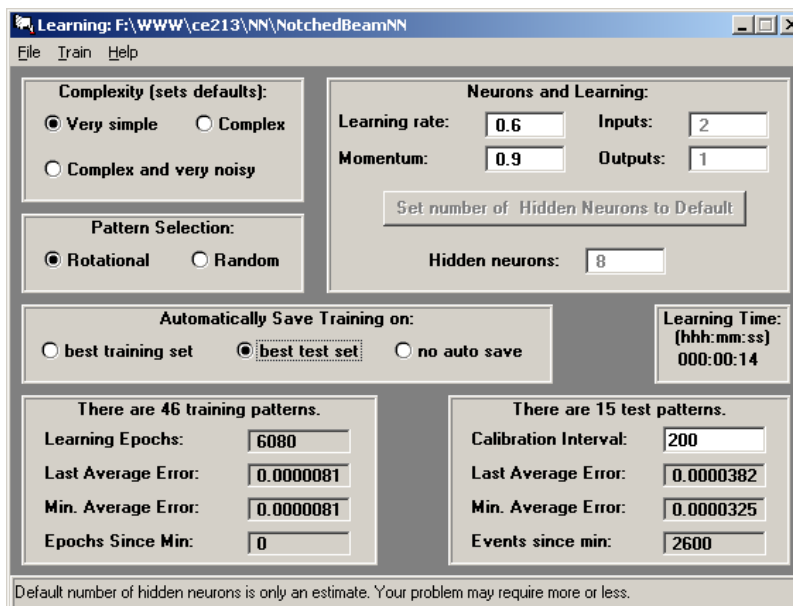


Figure B.5. Setting network parameters. This network was trained using mostly default settings.

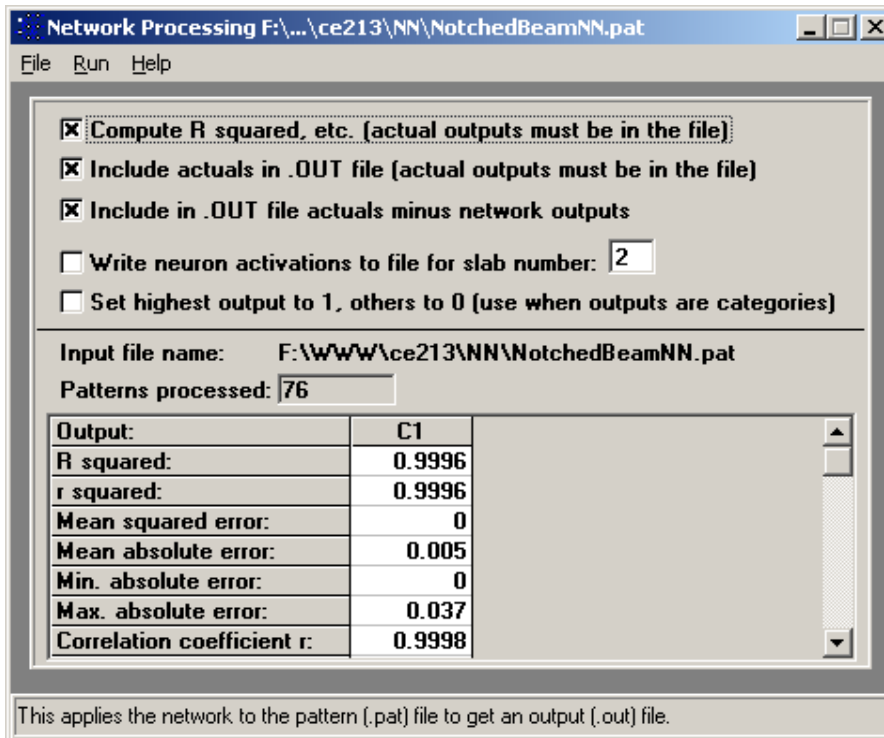


Figure B.6. The neural network's performance was assessed using the  $R^2$  metric.  $R^2$  values close to 1 indicate good correspondence with the training data.

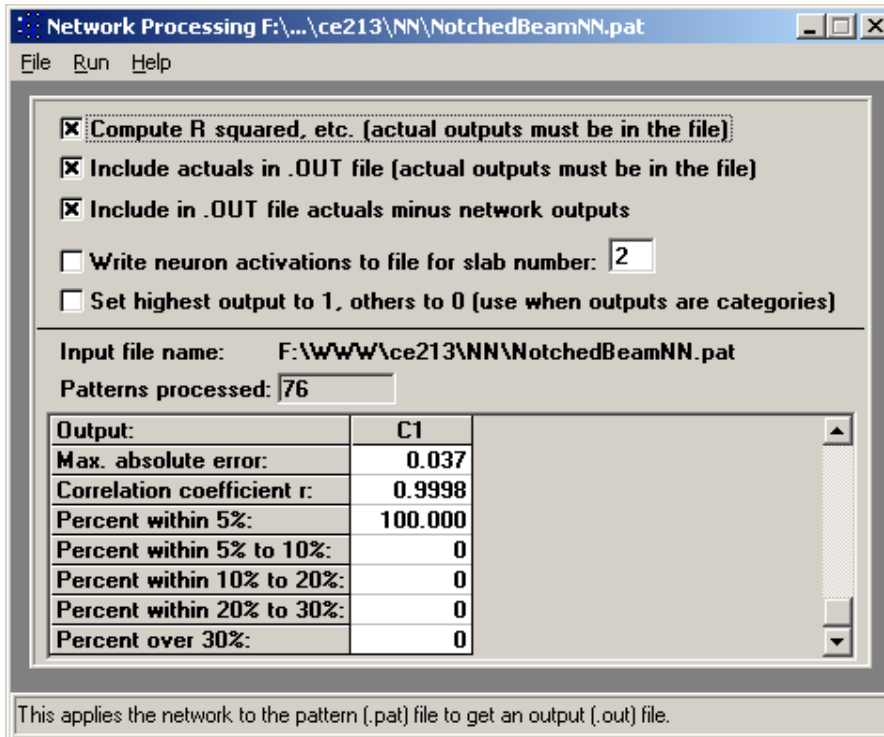


Figure B.7. The trained network is able to reproduce all cases within 5% error.

Table B.1. Comparison of actual outputs and computed outputs from the trained network.

<b>Actual</b>	<b>Network</b>	<b>Act-Net</b>	<b>Actual</b>	<b>Network</b>	<b>Act-Net</b>
2.55	2.53	0.01	1.66	1.66	0.00
2.41	2.41	0.00	1.65	1.65	0.00
2.30	2.31	-0.01	1.63	1.64	0.00
2.21	2.22	-0.01	1.63	1.63	0.00
2.13	2.14	-0.01	1.63	1.61	0.01
2.06	2.07	0.00	3.01	3.01	0.00
2.00	2.00	0.00	2.80	2.84	-0.04
1.95	1.95	0.01	2.63	2.66	-0.03
1.91	1.90	0.01	2.49	2.49	-0.01
1.86	1.85	0.01	2.37	2.36	0.00
1.82	1.81	0.01	2.26	2.25	0.01
1.79	1.78	0.01	2.17	2.16	0.01
1.75	1.75	0.00	2.09	2.09	0.01
1.72	1.72	0.00	2.02	2.02	0.00
1.69	1.69	0.00	1.96	1.96	0.00
1.67	1.67	0.00	1.90	1.90	0.00
1.64	1.64	0.00	1.85	1.85	0.00
1.62	1.62	0.00	1.81	1.81	0.00
1.60	1.60	0.00	1.77	1.77	0.00
1.58	1.58	0.00	1.73	1.73	0.00
1.57	1.57	0.00	1.70	1.70	0.00
1.55	1.55	0.00	1.67	1.68	0.00
1.53	1.54	0.00	1.65	1.65	0.00
1.52	1.52	0.00	1.63	1.63	0.00
1.51	1.51	0.00	1.61	1.61	0.00
1.51	1.51	0.00	1.60	1.59	0.00
2.84	2.84	0.01	1.58	1.58	0.00
2.65	2.66	-0.01	1.57	1.57	0.00
2.51	2.51	-0.01	1.56	1.56	0.00
2.39	2.39	0.00			
2.29	2.29	0.00			
2.21	2.21	0.00			
2.14	2.14	0.01			
2.08	2.07	0.01			
2.03	2.02	0.01			
1.98	1.97	0.01			
1.93	1.93	0.00			
1.89	1.89	0.00			
1.86	1.85	0.00			
1.82	1.82	0.00			
1.79	1.80	0.00			
1.77	1.77	0.00			
1.75	1.75	0.00			
1.73	1.73	0.00			
1.71	1.71	0.00			
1.69	1.69	0.00			
1.67	1.68	0.00			

## APPENDIX C - Developing a Visual Basic Stand-alone Application

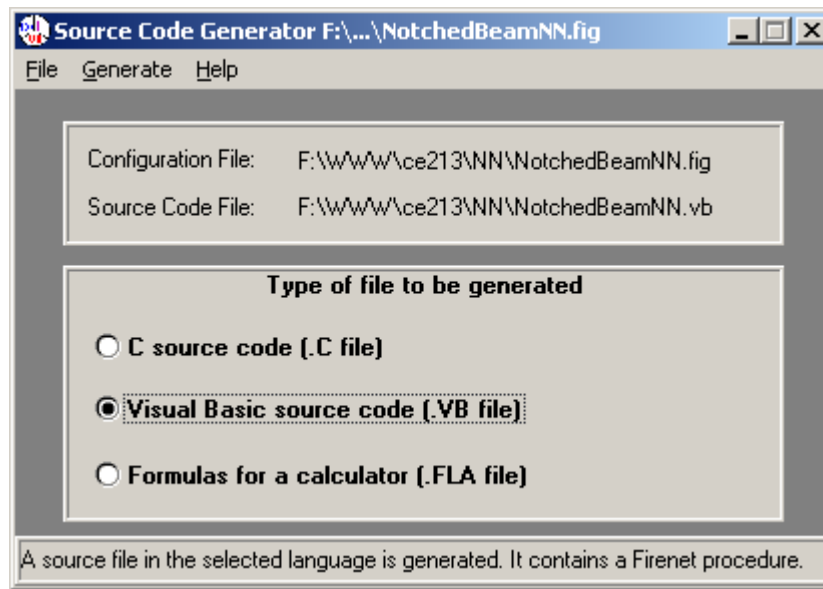


Figure C.1. NeuroShell 2 can generate source code in C, Visual Basic, and a generic format for use with calculators.

Table C.1. Visual Basic source code generated from NeuroShell 2.

```
' Insert this code into your VB program to fire the F:\WWW\ce213\NN\NotchedBeamNN network
' This code is designed to be simple and fast for porting to any machine.
' Therefore all code and weights are inline without looping or data storage
' which might be harder to port between compilers.
Sub Fire_NotchedBeamNN (inarray(), outarray())
  Dim netsum as double
  Static feature2(8) as double
  ' inarray(1) is h1/h2
  ' inarray(2) is a/h2
  ' outarray(1) is C
  if (inarray(1)<1.1) then inarray(1) = 1.1
  if (inarray(1)>2) then inarray(1) = 2
  inarray(1) = (inarray(1) - 1.1) / 0.9
  if (inarray(2)<0.05) then inarray(2) = 0.05
  if (inarray(2)>0.3) then inarray(2) = 0.3
  inarray(2) = (inarray(2) - 0.05) / 0.25
  netsum = -2.761628
  netsum = netsum + inarray(1) * 0.1916516
  netsum = netsum + inarray(2) * -3.577069
  feature2(1) = 1 / (1 + exp(-netsum))
  netsum = -1.504658E-02
  netsum = netsum + inarray(1) * 0.7065185
  netsum = netsum + inarray(2) * -3.172742
  feature2(2) = 1 / (1 + exp(-netsum))
  netsum = -1.407083
```

```

netsum = netsum + inarray(1) * 1.351259
netsum = netsum + inarray(2) * -2.85808
feature2(3) = 1 / (1 + exp(-netsum))
netsum = -3.309136
netsum = netsum + inarray(1) * 3.444713
netsum = netsum + inarray(2) * -15.29131
feature2(4) = 1 / (1 + exp(-netsum))
netsum = -3.895849
netsum = netsum + inarray(1) * -7.101685
netsum = netsum + inarray(2) * 4.234684
feature2(5) = 1 / (1 + exp(-netsum))
netsum = -5.930579
netsum = netsum + inarray(1) * 4.182865
netsum = netsum + inarray(2) * 2.499647
feature2(6) = 1 / (1 + exp(-netsum))
netsum = -0.1138547
netsum = netsum + inarray(1) * -5.1836
netsum = netsum + inarray(2) * -6.022686
feature2(7) = 1 / (1 + exp(-netsum))
netsum = -2.972661
netsum = netsum + inarray(1) * 0.1311532
netsum = netsum + inarray(2) * -4.111131
feature2(8) = 1 / (1 + exp(-netsum))
netsum = -1.64526
netsum = netsum + feature2(1) * 2.132756
netsum = netsum + feature2(2) * 2.229266
netsum = netsum + feature2(3) * 2.246875
netsum = netsum + feature2(4) * 5.741134
netsum = netsum + feature2(5) * -1.195331
netsum = netsum + feature2(6) * -0.8591065
netsum = netsum + feature2(7) * 0.6125866
netsum = netsum + feature2(8) * 2.114446
outarray(1) = 1 / (1 + exp(-netsum))
outarray(1) = 1.503135 * (outarray(1) - .1) / .8 + 1.506487
if (outarray(1)<1.506487) then outarray(1) = 1.506487
if (outarray(1)>3.009622) then outarray(1) = 3.009622
End Sub

```

*Note: The source code above was designed for Visual Basic 6.0. There are several changes in Visual Basic .NET. Currently, Visual Basic .NET does not support the "exp" function. The problem can be easily solved by defining an "exp" variable as 2.7182818, which is the value of e<sup>1</sup>. Then replace exp(-netsum) with exp<sup>^</sup>(-netsum).*

## **AUTHORS**

**Siripong Malasri** is a professor of civil engineering and the dean of engineering at Christian Brothers University. Dr. Malasri received his Ph.D. from Texas A&M University and is a registered professional engineer in the state of Tennessee. He has worked on artificial intelligence related areas since the mid-1980's. At CBU, he regularly teaches undergraduate mechanics courses as well as an artificial intelligence course in the engineering management graduate program. Occasionally he also teaches an undergraduate neural networks course for computer engineering and computer science majors.

**Jittapong Malasri** is a civil engineering junior at Christian Brothers University. He is an active member in the ASCE Student Chapter and Sigma Alpha Epsilon, holding the Vice President position in both organizations. He was the winner of the Water Resources Competition at the ASCE Southeast Regional Conference in 2003 and the Environmental Engineering Competition at the ASCE Deep South Regional Competition in 2004. This year Jittapong is on CBU's concrete canoe steel bridge teams.

**Kriangsiri Malasri** is a master's student in aerospace engineering at the Georgia Institute of Technology, with a specialization in flight mechanics and control. He has a strong interest in computer technologies as well and has done research on vision-based control of unmanned aerial vehicles. He plans to continue his education with a study of computer science, particularly artificial intelligence.