

***Whatever neural net model you choose, the general approach will be the same. Still, a network can learn more easily from some representations than others***

**by Jeannette Lawrence**

# Data Preparation for a Neural Network

**A**re you preparing to implement neural-net application ideas? It's important to assess the available data properly before making any project commitments. The data may need to be converted into another form to be meaningful to a neural network. (I'll treat the neural network as a black box and concentrate instead on processing your data into and out of a neural network.)

Regardless of which neural network model you choose, the general approach to solving your problem will be the same. Understanding the problem well enough to know what kind of information is relevant is the key to a successful design. How your data is represented and translated also plays an important role in the network's ability to grasp the problem. I will explain why a network can learn more easily from some representations than others. Certain kinds of data (for example, the time-oriented data used in such problems as financial forecasting) pose additional challenges.

Your data may be continuous-valued or

binary. Sometimes it can be represented either way—as a single continuous value or as a set of ranges that are assigned binary values, such as the temperature of food. This data could be represented by the actual temperature or as one of four possible values: frozen, chilled, room temperature, or hot. When you have naturally occurring groups, the binary categories are often the best method for making correlations. When the values are very continuous, artificially breaking them up into groups can be a mistake. It is difficult for the network to learn examples that have values on or near the border between two groups.

A common mistake is using continuous-valued inputs to represent unique concepts. For example, you may think it's perfectly reasonable to represent the months of the year as a number from 1 to 12. However, the neural network will presume such data to be continuous-valued and as having "more or less" or "better or worse" qualities. Since the month July (7) is not more or better than June (6), individual inputs are

required for each month. Zip codes, bar codes, and marital status are more examples of data that require more than one input.

Often, the choice between binary and continuous is not so simple. If your data is fairly continuous but not evenly distributed over the entire range, the best representation can be tricky. Consider a network that predicts the savings level of individuals based upon demographics and personal history. One of the inputs might be the person's education level, and the value could range from zero to 20 years. Natural groupings occur around "graduation" years that support representing the data in ranges such as less than 12, 12–15, and 16–20 years. A person with 14 years of education would be transformed into a 1 for the 12–15 group and a 0 for the others. But if significant differences occur within a group, they will be lost. If you used one continuous-valued input representing the actual number of years, the neural network might have trouble. If a significant difference in the effect on savings between having a high-school diploma and not having one occurs, the network may not pick it up, because 12 and 11 look very similar considering the range of 0–20. The best representation may be a combination of the two, such as groups with continuous-valued rather than binary input. You may need to try several representations to see which one works best.

### ACTUAL VS. CHANGES

An important decision in representing continuous-valued data is whether to use actual amounts or changes in amounts. Some data such as the Dow Jones industrial average has a tendency to shift over time. (Several years ago, it might have been around 1,600; someday soon, it could be 3,200.) The change in the Dow from month to month

stays within a range of +/– 100, so it's better to use the change than the amount.

Another reason for using changes in amounts is that the smaller the range, the more meaningful small-value differences are to the network. Suppose the value of the Dow could be anything from 1,600–3,200. If we used an actual value, the numbers 2,205 for one day and 2,208 for the next day would look very similar, because there's only a difference of three between them in a range of 1,600. Using the change (a value of +3 for the second day) is easier for the network to appreciate, because it is a larger percentage of the total possible range of –100 to +100.

You must also consider whether to describe the information as unique items (such as retriever, terrier, or shepherd) or as a set of qualities (such as large, black, or short-haired). Information that can be exclusively categorized as one of several possible items is called a nondistributed representation. You would assign one neuron to each exclusive quality, and the data will be either a "true" or "false" (1 or 0) for each. A drawback to using nondistributed information is that a reasonably sized network can store a very limited number of unique patterns.

Information is deemed distributed when the qualities defining a unique pattern are spread out over many pieces of information. For example, a purple object can be thought of as being half red and half blue. By using three primary color inputs (red, blue, and yellow), many possible color combinations can be represented without adding neurons. A distributed input scheme reduces the number of neurons needed to represent a large number of patterns that share common qualities. It can enhance the generalization ability as well.

The problem, however, with using a dis-

FIGURE 1.  
Questionnaire.

QUESTIONNAIRE		
1. NAME:		
2. ADDRESS: (STREET, CITY)		
3. ADDRESS: (STATE, ZIP)		
4. SEX: (M OR F)		
5. AGE:		
6. YEARS OF SCHOOL COMPLETED:		
7. MARITAL STATUS: (MARRIED/SINGLE/DIVORCED)		
8. NUMBER OF DEPENDENTS:		
9. ANNUAL INCOME: \$	/YR	
10. MORTGAGE OR RENT PAYMENTS: \$	/MO	
11. OTHER DEBTS: \$	TOTAL	
12. PHONE NUMBER:		

#### ADDITIONAL DATA GATHERED FROM SALES ATTEMPT:

13. PHONE INTEREST: 1-NONE, 2-VERY LITTLE, 3-SOME, BUT DECLINED PRESENTATION,  
4-INTERESTED IN ATTENDING PRESENTATION
14. SHOWED UP FOR SALES PRESENTATION: YES/NO
15. SHOWED INTEREST AT PRESENTATION: 1-NONE, 2-VERY LITTLE, 3-SOME, BUT  
DECLINED PURCHASE, 4-APPLIED FOR PURCHASE
16. BOUGHT TIME SHARE CONDO: YES/NO

tributed approach for the output is that often the network output must be decoded twice: first, from neuron activations to the distributed qualities and second, to the non-distributed identification.

For example, if color were distributed, an output pattern of .2 blue, .7 yellow, and .4 red would have to be decoded again by some external observer or program to brown (unless the output is supposed to be a paint formula). A network with a distributed output layer also has less learning capacity precisely because it has fewer connections.

A few advantages of a distributed output network are that it uses fewer neurons in the output and the hidden layers, has fewer connections, does less computation, and runs faster. Also, the more outputs you have, the harder it is to train the neural network to be as accurate with all of the output neurons. Generally speaking, neural networks with a greater number of inputs than outputs do better.

### DATA TRANSFORMATION EXAMPLE

Let's see how pieces of data are transformed into data for a neural network. This neural network predicts the likelihood of a person buying a time-shared condominium in a nearby vacation resort, so that sales attempts can be focused on good prospects.

The data comes from two sources: questionnaires filled out by residents of a metropolitan area near the resort and the results of sales efforts on a test group of residents who filled out questionnaires. The people are enticed to fill out the form (Figure 1 and Table 1) with an offer of a chance to win a TV or vacation. (In fine print, of course, is the notification that they must attend a sales presentation to qualify for the vacation drawing.) Once the form is completed, the resident is asked to attend the time-share condo presentation.

The four sales attempt indicators were added together to indicate an overall likelihood of buying. These items were weighted differently before summing:

- Interest on phone: 1–4 points
- Attended presentation: 0 or 10 points
- Showed interest: 0–50 points
- Bought condo: 0 or 100 points

*Total possible: 1–164 points*

To put this data into the network, we need two things: an input layer of neurons and an encoding scheme. The encoding algorithm's function is to take our input data and convert it into a form suitable for presenting to the network. To accept and understand solutions, we need two more items: an output layer of neurons and a decoding scheme. In some models, a single layer serves as input and output. These models are generally associative or memory types. The decoding algorithm takes the values of the output layer

ITEM # DESCRIPTION	DATA USED, RANGE	TYPE OF DATA
1. Name	not used	n/a
2. Street, city	not used	n/a
3. State, zip	Zip, last three digits	symbolic
4. Sex	m/f	symbolic
5. Age	18–100	continuous number
6. School	0–20	continuous number or ranges
7. Marital	m/s/d	symbolic
8. Dependents	0–10	continuous number
9. Income	5,000–500,000	continuous number
10. Home payments	250–5,000	continuous number
11. Other debts	0–200,000	continuous number
12. Phone	not used	n/a
13. Interest on phone	1–4 points	continuous number
14. Attended presentation	yes/no	binary number
15. Showed interest	1–4 points	continuous number
16. Bought condo	yes/no	binary number

neurons and converts them into a meaningful answer.

Encoding and decoding algorithms are neural-network specific, but some guiding principles can be applied to all. Neurons operate with numeric inputs and outputs that correspond to the firing rates, or activation values, of neurons. Your data may already exist as something other than numbers that fall within the range neurons understand (usually 0 to 1, or -1 to +1). It may exist as symbols (words such as male/female), larger or smaller numbers, or even pictures. The input encoding must normalize your raw data, such as turn it into a sequence of numeric values that the network can understand. Each number in the sequence is assigned to a particular neuron in the input layer. The output decoding must do the opposite; it must take a sequence of numbers that corresponds to the output neurons' values and turn them into whatever form is required for the final output. Many neural-network programs will automate this task for you.

If your numeric data has a natural range that is something other than the neuron's operating range, you must normalize it. For example, if the input data could be anything from 49 to 174, the total range is 125. An input value of 92 is 43/125 of the range, or .344 on a scale of 0 to 1. Some neural-network programs, such as BrainMaker, will do this normalization for you automatically.

Figure 2 and Table 2 depict the neural network that results from the collection of sales prediction data and a sample input pat-

**TABLE 1.**  
Questionnaire.

INPUTS	RAW DATA	TRANSFORMED (for a 0-1 range)	NORMALIZED
Zip	90131	131	1 (other Zip inputs are 0)
Sex	m	male	1 (female input is 0)
Age	27	27	.1098
School	17	17	.85
Marital	m	married	1 (single and div. are 0)
Dependents	2	2	.2
Income	35,000	35,000	.0606
Home payments	1,750	1,750	.3158
Other debts	5,000	5,000	.025
OUTPUTS	RAW DATA	TRANSFORMED	NORMALIZED (for a 0-1 range)
Interest phone	very=4 +		
Attended present.	yes=10 +		
Interest present.	very=50 +		
Bought condo	no=0 =		
Total Likelihood	62 points	.3780	

**TABLE 2.**  
Resulting neural network.

tern with raw and normalized data. Notice that the zip codes require a separate input for each area, even though zip codes appear to be continuous-valued numbers at first glance. It makes sense for the output to be renormalized for a 0–100 value, which would represent percentage of likelihood of buying a condo. The sample pattern shown would be translated into a 38% probability of purchase. It could be decided that this person should be contacted in a few years.

#### TYPES OF DATA

Let's talk about some very special types of data and how they should be handled. But first, we need to discuss how a neural network sees trending data or time-oriented

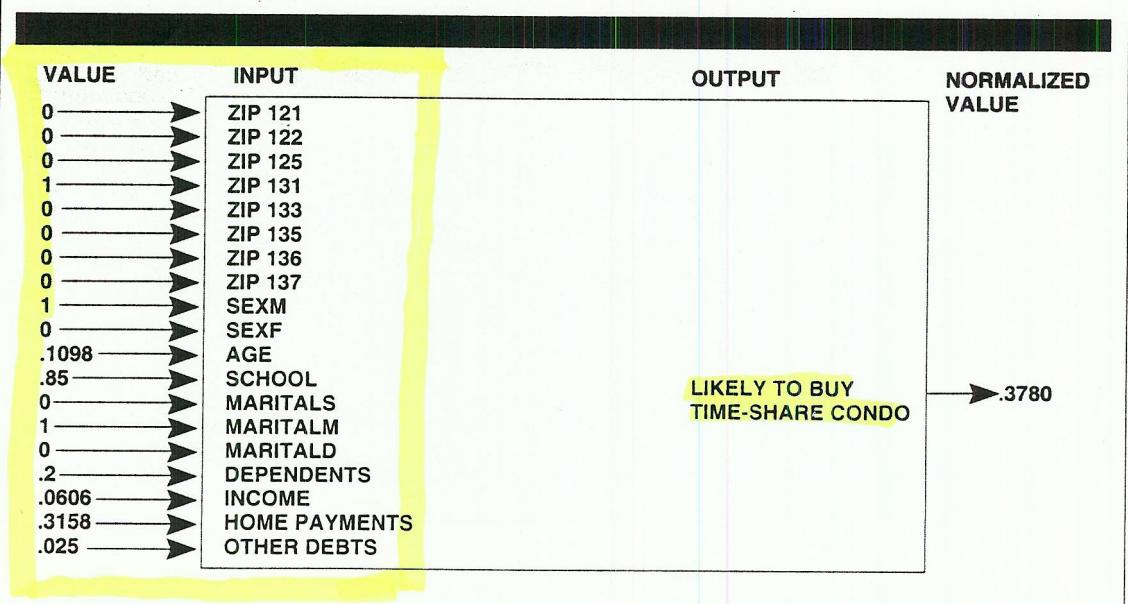
data such as financial forecasting data. A few neural-network models have some sustaining memory of previous data, but most don't. Most networks consider all the examples but only one example at a time. No explicit memory of the example has been previously seen. Therefore, you cannot simply present your data in a sequential order (first Monday's data, then Tuesday's data, and so on) and expect it to find the trend.

When you think about a time-oriented problem such as predicting tomorrow's stock price, you consider how the stock price has been changing each day for the last week or two (or some similar time period). Similarly, a neural network needs a lot of historical data. Each training example has some input data that spans several time periods and the output is the data for the next time period. Figure 3 shows an example financial neural network.

When the information is pictorial, the data has a wide range of possibilities at every point in the picture and is best suited to a nondistributed scheme. For example, in a black-and-white picture, each input neuron receives a number representing the intensity of one pixel (picture element) of the visual field.

Translation of picture images is more complicated but generally involves numeric values that correspond to a light/dark level for each pixel (dot) in the picture. The biggest problem with pictures is that often too many pixels are used to train a neural network in a reasonable amount of time. If a camera image is 1,024 pixels on a side, the number of necessary input neurons is more than a million. If you assume that your software could handle that amount, it could take months or years to train. The problem size should be reduced to something along the order of a few thousand inputs to train in any realistic amount of time. This reduc-

**FIGURE 2.**  
Resulting neural network.



tion can be done by using only the area of the picture containing some important feature, by tiling, or by performing a fast Fourier transform on the data. Tiling means grouping some adjacent pixels (a 16-by-16 square in this case) into a single value. A Fourier transform is a mathematical process that finds the frequencies of things.

One approach for feature extraction uses Fourier descriptors of the items to be recognized. For example, an application developed with BrainMaker reads a chemical drawing (comprised of characters and graphics) and translates it into a chemically-structured database. The chemical drawings are read into a PC from a scanner; some mathematical processing is performed to provide Fourier descriptors that are then fed into a neural network for recognition and translation into bonds and atomic symbols. Characters and graphics have frequency magnitude and phase "signatures" that can be recognized by the neural network. The neural network's output is formatted into a connection table and transmitted to a host computer database.

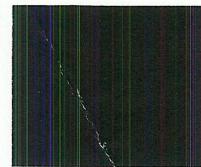
Fourier analysis of waveforms can also be useful. An example application developed with BrainMaker involves the analysis of sound waves obtained by nondestructive testing of concrete to determine whether voids exist. A large metal object strikes the concrete, and a nearby transducer picks up the reflected sound waves. The frequency content of the digitally recorded waveform

is then obtained using the fast Fourier transform technique. A single large amplitude peak at the frequency corresponds to multiple reflections of the pulse between the top and bottom plate surfaces. If a flaw occurs, the peak shows up at other frequencies. The value presented to each input neuron represents the amplitude at a particular frequency range. In this way, the neural network can "see" the flaw, which is also visible on the waveform. Similar processing can be done with EKGs and other electronic signals.

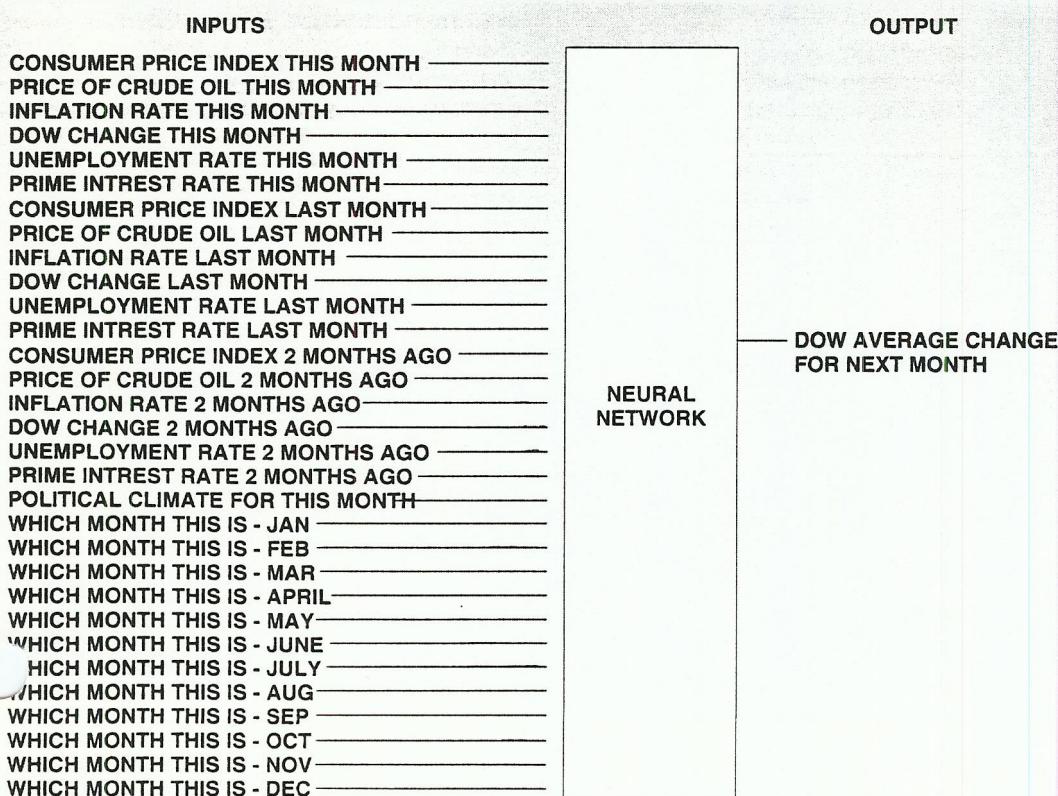
## DATA TYPE AND QUANTITY

Some important training considerations can affect the type and quantity of data you collect. Let's examine a few of these factors.

**Choice of data.** You don't need to determine formulas or rules to train a neural network. All you need to know is which kinds of information are important in solving a problem. If you're unsure, include it. A neural network can learn to ignore inputs that have little or nothing to do with the problem, provided you supply enough examples. It is rarely the case that too many kinds of data are used. More often, not enough data is used and correlations become difficult to find. When not enough kinds of data exist to make proper associations, the training time may become excessive. This situation can be evident with back-propagation networks when a very large number of hidden neurons is required to train the network. The danger with too many hidden neurons



**FIGURE 3.**  
Example financial neural network.



is memorization. This method is symptomatic of a network that trains well but tests poorly on new data.

**Data organization.** A big difference in how data gets organized occurs between supervised models in which the training data includes associated outputs (known answers) and unsupervised models, where it does not. If prediction, evaluation, or generalization is your goal, use a supervised model.

Supervised neural networks basically learn to associate one set of input data with a different set of output. For example, a neural network can associate a rise in GM stock price with a decrease in the price of steel. To predict the price of GM stock, you would specify the price of steel and other economic indicators as the input to the neural network. Naturally, you'd collect lots of historical data in which the indicators were paired with the corresponding GM stock price. Unsupervised models, such as Kohonen networks, are best applied to classification or recognition types of problems.

For example, you could store some descriptions of criminals (albeit a rather limited number). All the available data is input for training. When a new criminal case comes in with a partial suspect description, the neural network would look at the description and output the stored criminal that it most closely resembles.

**Data collection.** With a supervised model, the more examples you can collect for training and testing, the better. You must

have enough examples of a sufficient variety for training that the network will be able to make valid correlations for unfamiliar cases. The variety must include a good distribution of possible inputs and outputs. If your network will perform an evaluation such as the flight-readiness of an airplane, you must include examples of good and bad results in fairly even proportions. If you include 1,000 examples of the plane being ready and 10 of it not, the network will have a lot of trouble learning those 10. Even if it does learn them, when running it may tend to predict that the plane is ready more than it should.

You must have an ample training set so that you can set aside some amount for testing the neural network. Having the neural network learn a training set perfectly is not important, but having the network able to provide correct answers for data it has never seen is. A random sampling of examples should be set aside for testing. If you have very few examples with which to begin, you can use the "leave-k-out" procedure to train several networks each with a different subset of most of the examples; then test each with a different subset. Leaving a different set of examples (k) out of the training and subsequently testing on a different set will greatly enhance your ability to assess the network's effectiveness and may show you where more examples are needed. It will also give you a good idea of whether a network trained with all of the examples can generalize well.

If you don't have very many examples available, it may be all right to create a training set of data. If you're working with a supervised network (most commonly the case), the output for each pattern must be correctly evaluated by an expert, or it can be produced by a simulator. No one wants to look at all the generated patterns, so start by rating the obvious ones. Several experts can rate the examples. A single network might be trained on the sum total of what everyone thinks, or one might be trained for each person's opinion to see which gives the best results after training.

If you decide to fabricate examples, try to find the "border" patterns (examples in which the output just begins to be different). An example of a border pattern for a neural network that determines if the majority of inputs are true is the pattern in which the number of true inputs is one less than half the number of input units, such as seven of 15 inputs are true. The output for this pattern is false because a majority of inputs are not true. A minor change in the input pattern (only one more input made true) changes the output to true; hence it is a border pattern. Research has shown that the failure rate of a trained neural network decreases rapidly as the number of border

## **A big difference in how data gets organized occurs between supervised models in which the training data includes associated outputs (known answers) and unsupervised models, where it does not.**

training patterns used increases.

A manufactured training set using both border patterns and diverse-valued training patterns is substantially better. You can easily generate a list of patterns using a simple computer program. The number of possibilities for any continuous-valued input is simply limited by using a certain meaningful interval, such as every five thousand dollars in the range of the price of a home.

Randomly chosen training patterns are not the best, because any one set may arbitrarily emphasize the wrong conceptual points. The most easily identifiable patterns must be included for the network to learn the basics. There is no guarantee that a randomly chosen training set will include these basic patterns in the proper proportions.

Data that covers too long a time span can introduce problems. It is important to consider whether changes that have occurred, such as the introduction of new equipment, are related to a change in results. When the behavior has changed over time, you should limit the collection to a time period of similar behavior.

For example, the strongest influences on the value of the yen today may not be the same as those of 10 years ago, or a hospital may have acquired some new equipment two years ago that substantially improved treatment results. Designers of financial-ecasting neural networks often retrain their networks, throwing out the oldest data and adding newly collected examples to their training set. In the hospital case, you could still use all the data if you added an input that indicated if the equipment was present or not.

With most unsupervised models, the number of examples that can be stored is limited by the number of neurons. When too many examples are used, they begin to interfere with one another, and the network's recall ability diminishes. Autocorrelators (also called Hopfield nets) are able to store and perfectly recall at most  $n / (2\log n + \log \log n)$  number of patterns with  $n$  number of neurons.

There is much to consider when you begin developing a database for a neural network. The format of the data (continuous, binary, and so on) and the range of values represented (border and diverse-valued patterns) require careful attention. Remember, neural networks, like most computer programs, are very literal. They only understand what you say, not what you mean. ☆

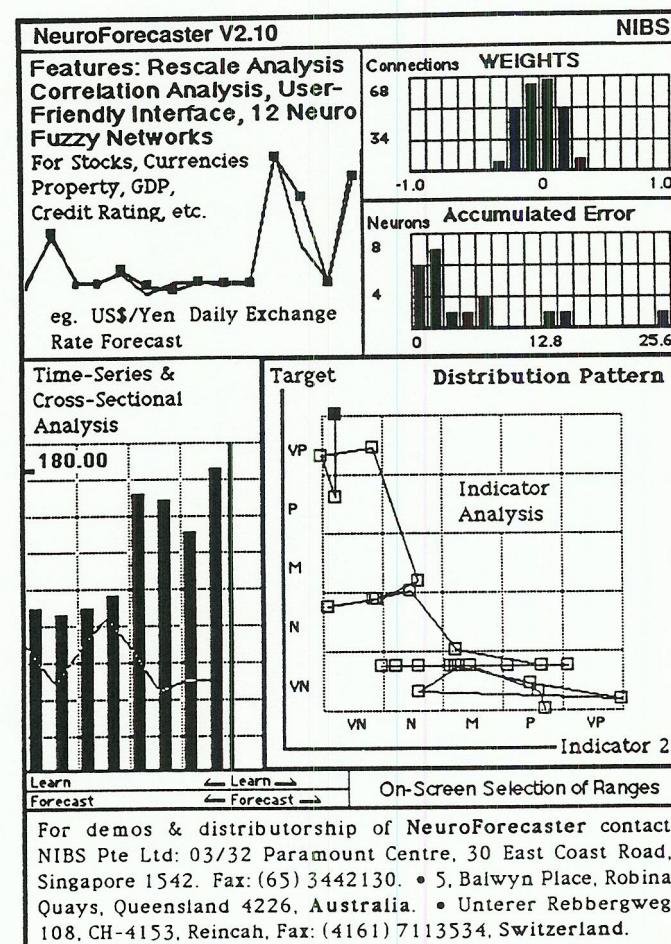
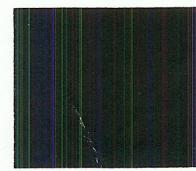
#### SUGGESTED READING

Hmad, S., and G. Tesauro. "Scaling and Generalization in Neural Networks: A Case Study," *Advances in Neural Information Processing Systems*, Vol. 1, 1989.

Grunlend, G.H. "Fourier Process for Hand Printed Character Recognition," *IEEE Transactions on Computers*, Vol. C21-2, pp. 195-201, Feb. 1972.

Sklansky, J., and G.N. Wassel. *Pattern Classifiers and Trainable Machines*. New York, N.Y.: Springer-Verlag, 1981.

**Jeannette Lawrence** has worked at California Scientific Software in the public relations, technical publications, and technical support areas. She is the author of *Introduction to Neural Networks* (California Scientific Software, 1991) and *Systhema Verlag-GMBH* (1991).



# Preparing the right data diet for training neural networks

**A**

As many IEEE Spectrum readers are doubtless aware, neural networks are a good way to interrelate nonlinear variables in a robust manner. The simplex method for optimization is not nearly as effectual, and neither are the various statistical methods for classifying and associating data and predicting results. The reason, in a nutshell, is that neural networks are put through a training phase, during which they can automatically fine-tune themselves as often as proves necessary to get the desired performance.

Of course, the old adage "garbage in...garbage out" applies as much to neural networks as it does to all other data-processing applications. If the training data set (the collection of input data and its associated correct output data) is not thoughtfully chosen, the resulting network is unlikely to hold up well in an industrial environment.

So it is hardly surprising that massaging the set of training data consumes some 80 percent of the engineering time spent getting a real-world neural network up and running—that is, getting it to converge under a broad enough range of conditions to be deployed with confidence in a production situation. If that data preparation is done systematically, much time can be saved and a more robust end-product can be obtained.

Here is a nine-step process that experience (the author's) has shown can enhance the probability of obtaining a learning convergence robust enough for industrial use.

## 1. Convert to comma-separated variables.

In developing a neural network system, the more the designer knows about the behavior of the data variables, or fields, with respect to one another, the better. In many cases, the statistical facilities built into a neural network software package will not yield sufficient information, in which case the use of a combination of tools may be desirable—for example, NeuralWare's DataSculptor, Microsoft's Excel, and a good database package, like DataEase. (You need DataEase only if you have more than 20 variables.) Formatting data sets as comma-separated variables (CSV) makes switching between DataSculptor and Excel effortless.

Actually, Excel and DataSculptor also work well with space-separated values, provided that scientific notation is used. The key requirement is to have a fixed width for each field.

## Defining terms

**Back propagation:** a learning scheme under which a multilayer neural network is organized for pattern recognition or classification utilizing an external teacher, such as sample data.

**Paradigm:** a framework that forms the basis of an algorithm; a theoretical thought process that arrives at a particular conclusion;

an approach or methodology that formulates an answer.

**Quintile:** any of the four values that divide the items of a frequency distribution into five classes.

**Six sigma:** a reference to the  $\pm 3\sigma$  range around the mean of a normal (gaussian) distribution, where  $\sigma$  is the standard deviation. The six-sigma range may be conveniently broken into five parts: too low, low normal,

normal (from  $-\sigma$  to  $+\sigma$ ), high normal, and too high.

**State of being:** any nonranked categorization—such as the classes true and false or red, blue, and green—where no class has any ordinal value above another.

**Universe:** the entire set of available samples describing the environment (although time and hardware limitations often dictate using less than the entire set).

Karla Yale, Yale Systems Inc.

## 2. Define project goals and be sure to gather enough data.

How best to configure the data depends to some extent on how the statement of the design objective is framed. The more precise and specific the objective statement, the easier it will be to see what neural network paradigm to use, what data to supply, and how best to present it [Table 1].

The sample size must be large enough to provide acceptably high confidence that the neural network will converge. To find this size, apply the number of variables needed to satisfy the objective statement in any standard t statistic and F distribution tables.

## 3. Scale the coded variables.

A key question to ask about the data in the samples is: are they measurable values, or are they states of being? If the former, it is highly desirable to map them to a limited range—preferably between 1 and 2. Why? For one thing, keeping the range of the measurable values as tight as possible reduces the chances that digital roundoff will kill a convergence. Also, the learning phase might converge and stay rooted at a local minimum instead of a more nearly optimal place.

As for preferring the range 1–2 over, say, 0–1, neural network matrices transform in the manner of eigenvectors, using standard matrix manipulations. In the course of those transformations, a matrix division usually proves necessary, for which reason it is best to avoid zero and its environs. This is all the more important when sample size is large and/or variables are numerous, because a great many iterations will probably be needed before the system finally converges. For a convergence that takes 11 hours, the last thing you want is to have the subroutines autoscale the data during each and every iteration. Not only does that waste time, but you may wind up with scaling at odds with the physics of the environment being modeled.

For classified variables, ask whether the information is classified for reasons of sequence, priority, or perhaps because it is the result of some physical measurement. In such cases of ordinal variables, the value of the variable is meaningful in itself and the rankings of the samples should be retained by any selected scaling process.

In the second instance, which involves cardinal variables, the values are merely numeric labels chosen perhaps for brevity or to correspond with existing categories recognized by a com-

## Examples of statements of design objectives

Statement	Shortcoming	Improved statement	Type of system
In manufacturing semiconductor devices—minimize wafer waste	This problem is too nonlinear to work well with the simplex method implied here	Generalize the chip layouts to an arbitrary set of shapes with a specified number of degrees of freedom  COMMENT: given the time constraints of development and processing, probably the optimum solution will not be found, but average chip size will be reduced	Genetic algorithm
In circuit design—reduce simulation run time by picking random seeds for initial conditions	This classical statistical approach does not recognize that the best starting values may not be known	Reduce the run time required for design simulation by developing a genetic-algorithm-based preprocessor	Genetic algorithm
In marketing a new product—determine the characteristics of a customer for it	The cumulative-frequency approach implicit here means the median customer will be profiled and many rare but profitable opportunities will be missed	Generalize the propensity for a customer to buy another product  COMMENT: the goal is to be able to forecast when this propensity increases	Back-propagation neural network
Maximize profits	This bad objective statement can be improved by using the simplex method, but it still will not yield valuable information it could have delivered	Identify the best sales prospects in terms of a propensity for generating profits and high interest in a product option  COMMENT: this improved objective statement finds profitable niches that may not be the most profitable and—without re-optimization—also finds the product options of interest	Back-propagation neural network
To predict chip quality before manufacture—apply statistical process control (SPC)	SPC looks at individual quality measurements, so is less robust than neural network technology, which interrelates all variables	Use a back-propagation neural network to predict chip quality before manufacture  COMMENT: knowing more about the quantified relationship of materials and process will cut waste, improve line efficiency; also, after as few as two stations on an assembly line, the neural network may be able to spot imminent quality problems, whereas SPC would find the individual measurements still within acceptable limits	Back-propagation neural network

puterized database. In such a case, it would be misleading to give them a ranking, or priority. A better way to represent these cardinal variables is as vector states in the multidimensional network space. For example, the variables 1, 2, and 3, may represent the colors red, green, and blue, with the fact that the number 3 is greater than the number 2 having no significance at all.

If the network converges, the results will correspond more accurately to the physics of the environment bounded by the objective statement when the vector state representation is used. Thus, the numbers 1 through 3 in the example will be more useful if converted into the vectors (1, 0, 0), (0, 1, 0), and (0, 0, 1).

### 4. Use sampling techniques to generate data sets.

If an application uses a total of, say, four independent variables, then obtaining a 95 percent confidence level around convergence requires a learning data set with about 100 unique samples for each independent variable—90 for training and 10 for testing. The number is based on the F distribution, which does not change very dramatically between 100 and 1000 samples. The more unique samples in the training data set, however, the more robust the network.

So, for the example of four input variables, a 95 percent confidence level around convergence requires a total of at least 400 samples. If more than 400 data samples are available, select randomly from among them. For example, if 4000 samples are available, choose every 10th sample, unless there is something about the data that would be skewed as a result of that pick frequency. Save a random 90 percent of the samples into the training data

set. Place the remaining 10 percent into the testing data set.

What if the situation is reversed and there are too few samples? If a model for production is needed, one of the three following approaches may be taken:

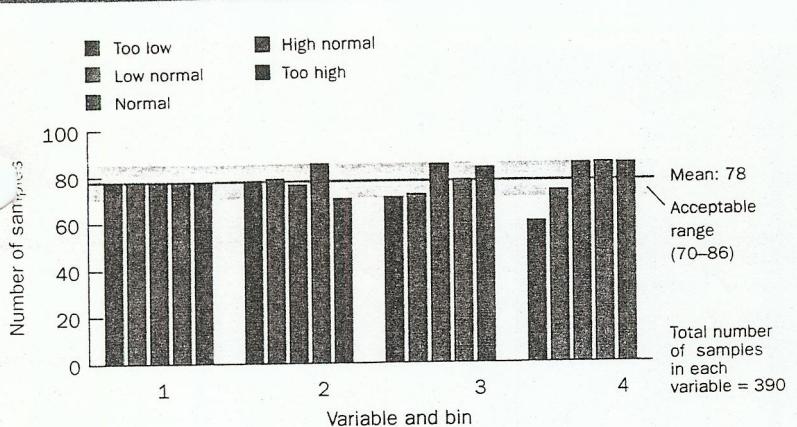
- Turn the objective statement into an optimization that calls for a genetic algorithm.
- In situations such as starting a new product line, there will be very few samples for the condition of test failure. To cope with the relative paucity of samples for one state, try turning the objective statement into a prediction that calls for the next best value. Because there has to be a band of acceptable values around the predicted value, this approach is not as robust as having actual samples for all states. It is, however, more robust than traditional statistical approaches because of the ability of neural networks to model nonlinear relationships.
- For situations in which enough time, money, and expertise are available, convert the model into a fuzzy logic objective.

### 5. Provide scatter plots for the learning data set.

To retain the natural distribution of the data, find out what transformation fits best. To do that, make scatter plots of the data, and then go on to Step 6.

### 6. Transform variables.

Let's say you have the scatter plots of the universe of 4000 samples and some minimal description of the data—variable name, format, examples, and a description or definition. To pick an initial transformation function for each variable, examine the



[1] This first pass at creating a training data set comes close, but gets no cigar. The problem: the fourth variable has too few samples in the "Too low" bin. In this example, which is not the same as the one on p. 65, it has been decided to use 390 samples of each variable and to assign each sample to one of five bins. Therefore, a perfect distribution would have 78 samples in every bin [first variable]. If deviations of up to  $\pm 10$  percent from perfection are allowed, 70–86 samples are acceptable [colored bar]. But the 60 samples shown in the "Too low" bin for the fourth variable is too low.

[2] This distribution is acceptably uniform, as the number of samples in each of the bins is within  $\pm 10$  percent of perfect uniformity

shape of the scatter plot for its universe, and pick a transformation function with a similar shape. For example, if the scatter plot has a frequency to it, select a periodic function, like a sine or tangent. If it has a huge range, try a log function. If the scatter plot slopes upward, perhaps an exponential function will work best.

Then transform the variables, and generate the scatter plots of the transformed results. Compare the scatter plots of the transformed learning (training plus testing) data set with those of the universe. In the foregoing example, the universe is 4000, with the training data set approximately 360 samples. If the scatter plots are not similar, make another pass at the random selection process. Keep iterating Steps 4, 5, and 6 until the sample pattern is similar to that of the universe.

## 7. Create a uniform distribution of the learning data set.

Figure 1 shows an example of a not-quite-adequate distribution of the training variables for a somewhat different situation: 390 samples of each of four variables. In this example, the number of samples of the fourth variable in the "Too low" bin is more than 10 percent below the target value of 78. The first variable is an example of perfection and rarely, if ever, occurs.

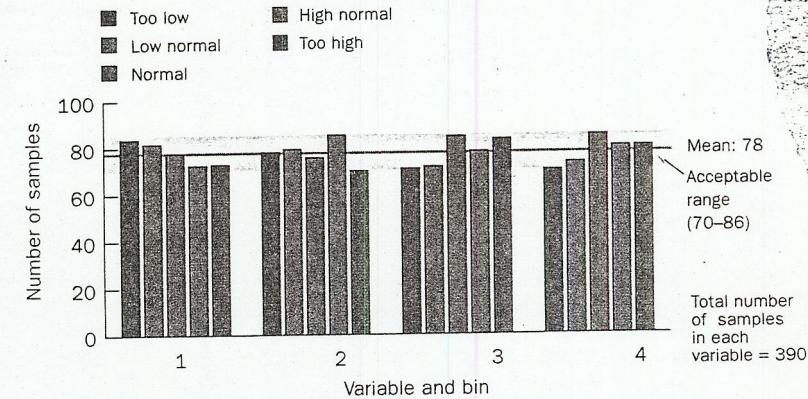
This type of distribution, if it does converge, can generate extremely flexible neural networks—networks flexible enough to work in applications other than the one one that spawned them.

Remember, in this step the goal is to create the learning data set, which has nothing to do with the distribution of the sample in the real world. The idea is to create a data set that will train the neural network to perform well under unusual conditions as well as in more common ones.

## 8. If necessary, recheck the database.

If necessary—that is, if there are any gaps in the uniform distribution—recheck the entire database. Take another look at the distribution of the fourth variable [Fig. 1, again]. There is a definite gap on the low side. Recheck the universe to see if there are any samples below the lowest quintile (below  $-2\sigma$ ). If there are a few, pull them into your pseudorandom sample.

You will want to replicate these samples so that they are uniformly distributed. This may seem illogical at first, but it makes sense when you remember that the network has to recognize the unique as well as the commonplace. This has nothing to do with it learning that a certain case happens only rarely. The network has to be well rounded enough to recognize atypical situations.



## 9. Recheck the uniform distribution.

Repeat Step 8 as necessary until the distribution becomes uniform across all variables. An acceptably distributed sample is shown in Fig. 2.

These steps are admittedly not very exciting. If you want an industrial-strength system, however, they are essential. ♦

## To probe further

*Introductory Statistics* by Thomas H. Wonnacott and Ronald J. Wonnacott (John Wiley & Sons, New York, 1990) contains F distribution tables, sigma (used for defining variance), and other statistical measurements of fit.

Among the many neural network packages currently available are: Adaptive Solutions; BrainMaker; CubiCalc Rule Maker; CubiQuick; DataEngine; Fuzzy flex; Fuzzy Judgment Maker; Fuzzy Knowledge Builder; Fuzzy Logic Toolbox; Fuzzy Thought Amplifier; G2; GURU; IDIS-PM; HNC; HNet; LevelN5 Quest; Nestor; NeuCOP; NeuFrame; NeuFuzzy; Neural Net Toolbox; NeuralWorks Explorer; NeuralWorks Predict; NeuralWorks Professional II/Plus v5.2; NeuronON-Line; NeuroShell 2; O INCA; and OWL.

The author's own World Wide Web site, [www.yale-systems.com](http://www.yale-systems.com), gives several descriptive examples of successful implementations. Visitors to the site may also request slide shows of other applications.

Another valuable Web site, [www.neuralware.com](http://www.neuralware.com), explains the NeuralWare DataSculptor and Professional II/Plus products, which handle back-propagation and fuzzy-logic paradigms, among others.

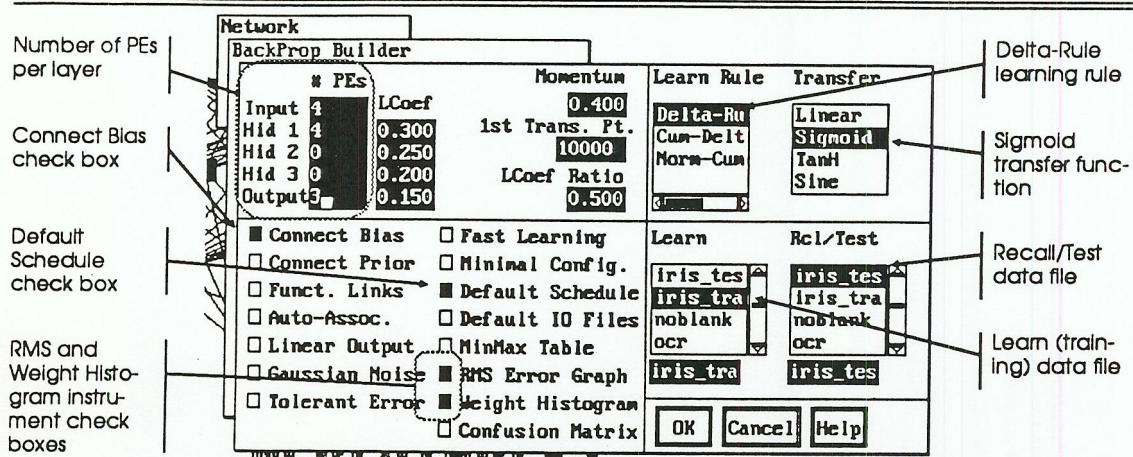
## About the author

Karla Yale (SM) is the president of Yale Systems Inc., Columbus, Ind., an engineering consulting company focused on artificial intelligence solutions for industry. Earlier, she worked for General Electric Co., where her duties included strategic planning and electronic design engineering. While with General Electric, Yale served as chairman of the Daytona Beach Chapter of the IEEE. She may be reached via e-mail at [k.yale@ieee.org](mailto:k.yale@ieee.org).

**Rule-of-thumb 1:** The more complex the relationship between the input data and the desired output, the more PEs are normally required in the hidden layer.

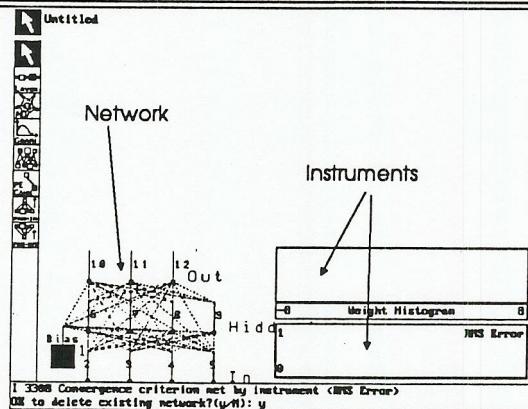
**Rule-of-thumb 2:** You can use the amount of training data available to set an upper bound for the number of PEs in the hidden layer. To calculate this upper bound use the following formula:

$$\frac{\text{cases}}{10 \times (m + n)} = h$$



Make the appropriate selections in the BackProp Builder dialog box.

A network and a pair of instruments similar to these will appear on your screen.



clutters the graphical display; hence O'INCA displays the neuron bias in its intuitive form as an attribute of the Neuron object.)

In some cases when a problem requires mapping of data beyond [-1,1] range, a *linear* activation can be used in the output units (alternate option is to normalize the neural net's input/output data). Also, note that activation functions approximating a "hard limiter" may be generated by using large Gain values for the Sigmoid activation function type.

### Neural Network Architecture

O'INCA version 1.0 supports multi-layered, feed-forward (MLFF) neural network architecture. Layers in between the input and output layers are called hidden layers. If the input layer is viewed as the "lowest" layer and output layer the "highest" layer, a neuron can only connect to another in a higher layer. Input layer neurons do not have fan-in links and output layer neurons have no fan-out links. Lateral connections between neurons in the same layer and feedback connections from neurons in the higher layer to lower layer neurons are not permitted in this architecture.

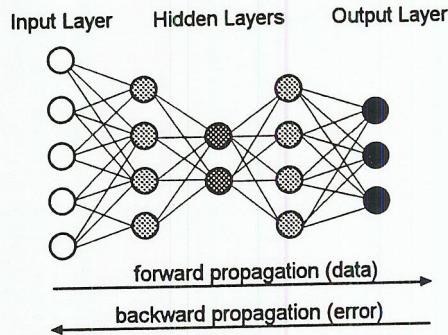


Figure 2-11: Multi-Layered Feed-Forward (MLFF) Nerual Network

Theoretically, only one hidden layer (with sufficient number of units) is required for approximating an arbitrarily complex continuous function, and hence the MLFF neural nets are referred to as universal approximators. However, in practice, two or three hidden layers are sometimes used to realize compact architecture or to accelerate the training process. In some experimental or "hand-crafted" networks

clutters the graphical display; hence O'INCA displays the neuron bias in its intuitive form as an attribute of the Neuron object.)

In some cases when a problem requires mapping of data beyond [-1,1] range, a *linear* activation can be used in the output units (alternate option is to normalize the neural net's input/output data). Also, note that activation functions approximating a "hard limiter" may be generated by using large **Gain** values for the Sigmoid activation function type.

### Neural Network Architecture

O'INCA version 1.0 supports multi-layered, feed-forward (MLFF) neural network architecture. Layers in between the input and output layers are called **hidden layers**. If the input layer is viewed as the "lowest" layer and output layer the "highest" layer, a neuron can only connect to another in a higher layer. Input layer neurons do not have fan-in links and output layer neurons have no fan-out links. *Lateral* connections between neurons in the same layer and feedback connections from neurons in the higher layer to lower layer neurons are not permitted in this architecture.

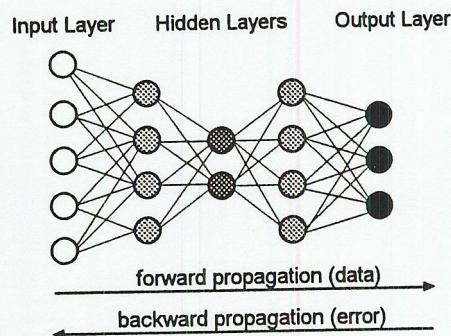


Figure 2-11: Multi-Layered Feed-Forward (MLFF) Neural Network

Theoretically, only one hidden layer (with sufficient number of units) is required for approximating an arbitrarily complex continuous function, and hence the MLFF neural nets are referred to as universal approximators. However, in practice, two or three hidden layers are sometimes used to realize compact architecture or to accelerate the training process. In some experimental or "hand-crafted" networks

additional intermediate layers are necessary. O'INCA allows up to ten *Ten layers* hidden layers for special designs.

Determination of the number of hidden layer neurons is a non-trivial problem, especially when searching for a [near] optimal configuration. Even if the resources (number of neurons & connections) needed to perform a particular type of computation can be estimated, the number of neurons required to learn the problem may be different. However, some guidelines can be used to approximate the initial number of hidden layer neurons.

For pattern classification problems with  $m$  Input neurons and  $n$  Output neurons, start with approximately  $(m \cdot n)^{1/2}$  hidden neurons (with only one hidden layer). If  $m$  and  $n$  are large,  $(m+n)^{1/2}$  can be used. For function approximating types of problems,  $m$  and  $n$  are typically very small, so begin with  $(m+n)^2$  neurons and add more if additional accuracy is required. For associative memory/data compression ( $m-q-m$ ) types of problems, use  $q$  units for which  $m/q$  provides the desired compression ratio. Note that these formulae are for an initial crude estimate only -- the number of required hidden units also depends on the complexity of the problem, the number of training patterns, and the "quality" of the training data (whether noisy or clean, etc.). Typically, the larger the training set and the more complex the relationship between input/output data, additional hidden neurons will be required.

For good generalization performance after training, it is important to minimize the number of hidden neurons. Note that for each additional hidden neurons,  $(m+n)$  additional degrees of freedom (weights) are provided to the network, and care must be taken to avoid "overfitting" on the training data. Most practical problems can be solved with only one hidden layer. If two hidden layers are used, choose the initial number of neurons in the second hidden layer to be at least 50% less than the number in the first hidden layer to increase the level of data abstraction.

After the initial design of the neural network is trained and tested, multiple iterations of changing the architecture and/or data set may be required to complete the design of the neural network to desired performance level.

### Training Algorithms

Training an MLFF neural net is accomplished by a supervised method. This means that "teach" signals are provided during training to indicate