

*Some neural network developers
always pick their favorite
network type for all applications.
Others can't decide. But choice
doesn't have to make life hard.*

by Maureen Caudill

Avoiding the Great Backpropagation Trap

Selecting the correct neural network model for a particular application is much like selecting dessert in a fancy restaurant. One diner—perhaps a self-confessed chocoholic—automatically selects the most chocolatey dessert, perhaps the double chocolate decadence supreme cheesecake with fudge sauce and chocolate sprinkles. Another diner dithers and wavers, trying to make a choice between what appears to be equally delectable options, and finally ends up with the same old scoop of vanilla ice cream just because it's a "safe" choice. In the first case, the diner loses out on the possibility of discovering an even better dessert; in the second, time and effort is wasted while much more interesting options are avoided. In more extreme cases, each of these diners may forego dessert entirely to avoid having to make an unacceptably risky decision.

Similarly, many beginning neural network users automatically select their favorite network type—usually backpropagation—for all applications. Others see the variety of

network choices available and vainly attempt to make a firm decision between them. Because they are not sure how to choose a new network model that they have not yet tried, they eventually settle for their tried-and-true network, again usually backpropagation. Alternatively, the entire problem may be given up as a hopeless task.

I realize this may come as a distinct shock to some people, but backpropagation, as excellent as it is for many tasks, is not the only good neural network for applications. In fact, a number of perfectly doable network applications cannot be solved by classical backpropagation networks. Several variations on backpropagation do exist, of course, but—believe it or not—many applications can be solved successfully without using gradient descent, or the delta rule. An alert and aware neural network application designer can avoid both these extremes by exercising a little judgment and care.

BE AWARE

Before you can use a network model to solve

an application, you must first know about that model. Thus the first step in breaking out of the Great Backpropagation Trap is to learn as much as you can about as many neural network models as possible, which means, at a minimum, regularly reading the pertinent literature. Many journals are now published regularly, including several from professional societies. Also look for conference proceedings in the engineering or science sections of any good university library; in these proceedings the quality of the papers varies greatly, but some real gems are presented at nearly every one. Be aware that the papers published in all these sources are research papers, so many of them are difficult to read and understand, particularly for a neophyte.

If reading research journals is not your thing, you can also try books. The mathematical sophistication needed varies greatly in the available books; check the AI or computer shelf at any good university or technical bookstore for specific titles. Many people have trouble understanding very technical books and articles. If you are one of these—and I suspect that this includes more people than are willing to admit it—you may want to consider taking a training course in neural networks. A proliferation of one- to five-day "short courses" are being taught these days. If you plan to attend one of these, look very carefully at the instructor's credentials and the course syllabus before signing the check for the tuition.

Try to avoid classes where an instructor may try to force a single network architecture on you as the panacea for all problems. In fact, stay away from courses taught by people who are known as promoters of a particular network architecture, unless you want to get an in-depth study of that architecture.

At least a half-day of instruction should be devoted to each kind of network discussed; a one-day course that promises to teach you eight network architectures will confuse you, not educate you.

Also make sure that the instructor won't force a single network architecture on you as the panacea for all problems. In fact, my advice is to stay away from courses taught by people who are known as "promoters" of a particular network architecture, unless you want to get an in-depth study of that architecture. Human nature being what it is, you are likely to be taught that the instructor's pet architecture encompasses all others or clearly outperforms all other networks at all problems. Of course, if you do want to learn about a particular network in depth, such a course would be a reasonable choice.

As an alternative, consider signing up for a university extension or night-school course. The problem here is that you really won't know much about the instructor's background in advance, particularly with regard to architecture biases. On the other hand, the course will likely be taught in the evenings or weekends over 10-12 weeks and will not interfere as much with your work schedule, not to mention being considerably cheaper than a typical short course. Also, you can often learn more if the material is spread over an entire quarter or semester than if it is concentrated in a single week.

No matter what kind of course you sign up for, make sure that it involves some kind of hands-on training—at least a term project—that allows you to work with a neural network simulator. Psychologists have shown that we remember only about 10%-20% of what we see and hear, but we retain about 90% of what we do. Forcing yourself to take on a project or work with a neural network simulator will help you really understand how they work much more than just listening to someone lecture. The point is that you must spend some time and effort educating yourself about a variety of network architectures before you can expect to be able to be more creative about your network choices.

ANALYZE YOUR PROBLEM

First ask yourself whether the application you have is truly a neural-network problem or not. Does an algorithm exist to solve the problem? Is the algorithm effective (is it reasonable to implement in terms of the time and hardware constraints you have)? Are there specific *if-then* rules that can be used to solve the problem? If so, is there a lucid, cooperative expert available who can help you define those rules? If the answer to any of these questions is yes, you don't have a neural-network problem, you have a programming or an expert-system problem.

While considering the application from this standpoint, consider also whether it can be broken into pieces, each of which is amenable to separate solutions. Perhaps you have a large application in which you need some digital signal processing, production rules, and pattern recognition. Plan to use ordinary programming techniques for the signal processing, an expert system to implement the rules, and a neural network to do just the pattern recognition part of the problem (assuming, of course, that the questions I raised are properly answered).

Neural networks should be used only to solve problems for which more traditional alternatives don't exist or are unsatisfactory in any way. "Unsatisfactory" can mean anything from too expensive (in time, money, hardware, or personnel) to run or develop, too large physically, too inaccurate, or just "too anything." If a perfectly satisfactory traditional alternative exists, forget about neural networks.

DEFINE THE PROBLEM

Now that you are educated about neural networks and have determined what pieces of the application are truly neural-network problems, you are faced with a bewildering variety of networks from which to choose for a given problem. How can you figure out which one is best for your application?

Each neural network architecture and training system is better at some kinds of problems than others. As you learn about a new network type, try to analyze it to determine what kinds of problems it would be able to solve. For example, backpropagation networks make wonderful mapping networks but may not be quite as good as counterpropagation networks for associative memory problems. Today's self-organizing networks are almost always restricted to categorization problems or, sometimes, associative memory problems.

One good way to begin is with the process of elimination. Consider your problem and ask yourself what specific information you expect to give the network as input and what you want the network's output to be. While you are defining these things, try to determine the general kind of application you are working on. These fall into several categories.

Mapping. A mapping problem is one in which an input pattern is associated with a particular output pattern. For example, the input pattern that consists of a pixel-image of a character may be mapped to the output pattern of that character's ASCII code. Or a pattern of sensor readings may map to a pattern of valve settings. Backpropagation and counterpropagation networks are very good at these problems, as well as functional-link nets.

Associative memory. An associative memory stores information by associating it with other information; recall is performed by providing the association and having the network produce the stored information. One way to distinguish an associative memory problem from a mapping problem (the dividing line between them is very fuzzy) is to ask yourself what you want the network to do with an input pattern on which it hasn't been trained. If you want the network to reproduce one of the output patterns you trained it with—to recall it, essentially—you probably have an associative memory problem. If you want the network to generate a new output, you likely have a mapping problem. These problems can be solved by a variety of networks, including backpropagation, counterpropagation, Kohonen, and crossbar networks.

Categorization. In a categorization problem, the inputs are to be clustered into categories. Typically the network is provided with an input pattern and it responds with the category to which the pattern belongs. Self-organizing networks are often excellent choices here, including Kohonen and adaptive resonance networks (as are Adaline and probabilistic nets).

Temporal mapping. This problem is just like any other mapping problem, except that the input data includes a consideration of time. For example, the input may be a time sequence of sensor readings that must map to an output pattern. Many process-

Neural networks should be used only to solve problems for which more traditional alternatives don't exist or are unsatisfactory in any way. "Unsatisfactory" can mean anything from too expensive to run or develop, too large physically, too inaccurate, or just "too anything." If a perfectly satisfactory traditional alternative exists, forget about neural networks.

control patterns are temporal mapping problems. Depending on circumstances, consider avalanches, backpropagation, and any recurrent network.

Image processing. I separate image-processing problems from other mapping problems primarily because of the huge dimensions of the raw input data. A high-resolution image can easily contain $1,024 \times 1,024$ pixels, or more than one million elements in the input vector. Because of the limited size of current neural networks, image processing problems require special preprocessing techniques and always must have the size of the input data reduced. Depending on your goals, consider a categorizing or mapping network.

TRAINING PROCEDURES

With the constraints that you now have on the scope of the problem, you should have some idea of what information you will be able to give the network. For each input pattern, do you know exactly what the network's output is supposed to be? If so, you should probably plan to use the most efficient training procedure, supervised training, in which you provide the network with the exact output you want it to learn to produce. If not, can you provide the network any kind of information about its performance? If this answer is yes, then you will likely want to use some form of "graded" training. (Graded training means that you provide the network with a measure of how

well or poorly it's doing without specifically telling it what the output should be. For example, you might tell a network that the level in a water tank is "too high" or "too low" without giving it specific information on how to set the valves its output controls.) If you have no information at all to give the network, you are automatically constrained to unsupervised training schemes.

Knowledge of the kind of training you plan to do will immediately help you sort out the possible networks you can use. For example, if you want to use unsupervised training, you have completely eliminated backpropagation networks. If you want to do supervised training, forget about Kohonen networks (except in the guise of a counterpropagation net) or adaptive resonance networks.

In many applications you will have only partial knowledge of the correct output for each input pattern, which opens up a wide variety of networks that may have unique architectures, modified training schemes, and so on. In fact, in your newly educated state, you may have an idea of how to modify some network slightly to solve the problem. If so, go for it—and when you're done, write it up as a paper in one of the journals.

GIVE IT A HINT

Neural networks are not afflicted with a macho mentality. Just the opposite, in fact. Most networks will do much better if they are provided with a hint about the problem at hand. Sometimes you don't know much about how to solve a particular problem, but there are many cases for which you have at least a clue about features or concepts that are very likely to be helpful. In these cases you should try to pass that information along to the network. This can be done in several ways.

First, you can preset the starting configuration of the network. In the case of a backpropagation network, for example, you can preset some (not all) of the weights to the middle-layer neurodes so that some of them search for specific features that you think are likely to be important in the input pattern. You can also build the backpropagation network with extra connections, such as direct input layer-to-output layer connections. In a Kohonen network you accomplish the same thing by carefully normalizing the weights and input patterns and setting the initial states of the weights appropriately. Other kinds of networks can be built with similar "prejudices." Of course, if your prejudices are wrong, you will probably make the network's task harder instead of easier, but if you are right, you may considerably shorten the training time needed.

A second method is based on response training, originally developed by Gary Josin

Knowledge of the kind of training you plan to do will immediately help you sort out the possible networks you can use.

of Neural Systems in Vancouver, B.C. Response training means that the results of the network's actions in this time period are provided as part of the inputs to the network in the next time period. The response inputs can be from direct sensor readings, or they can be computed based on the fundamental physics (or other determining science) of the problem. As an example, if the network is used to keep an airplane flying straight and level, the response input at time 2 might be computed from time 1's results, using Newton's laws of motion to determine the plane's status as a result of that output. In essence, you are providing the network with some level of knowledge of the laws of motion.

A third method is accomplished by giving the network a hint. This method adds an additional output to the network; the output is for a quantity that you think is important to the problem. In the straight-and-level flight problem, for example, you might have the network learn to generate the plane's height as an additional output, added to the more normal thrust and elevation outputs. Here again, you provide the network with the notion that the height—computed again by the laws of physics—is an important quantity that should be considered in solving the problem.

CHOICES, CHOICES

Finally, consider if your application has any special constraints that would affect the choice of network architecture. For example, must the network learn on-line? Does it have to achieve a particular speed either during training or processing? Does it have to learn all the time?

In each of these cases, the network of choice may not be backpropagation. Unless you have hardware assistance in the form of a special accelerator card (a 25 or 30MHz PC usually is not good enough) or are working with a commercial neural network chip, you will probably not be able to do real-time, on-line backpropagation training. In these cases, consider a network that uses Kohonen or Hebbian learning; the math is much simpler with significantly reduced computational loads.

What about the software simulator? If you plan to write your own, you can do anything you want. If you plan to use a commercial simulator, you have to live within the constraints of that software.

Are you trying to do a temporal mapping? If so, consider any one of the recurrent networks. But beware: Recurrent networks—except for the very simple crossbar recurrent networks—tend to be huge memory and processor hogs, even more so than backpropagation networks. You should definitely plan on training such networks on a Sun or

some other high-speed workstation—and the faster the better.

Do you want to do categorization? Consider a Kohonen network or even an ART1 network if you have high-speed hardware. Be careful of the preprocessing and other constraints for these networks, however.

Do you want to work with sensor data that is inherently unreliable? Consider using a fuzzifier on the input data before passing into the network. Is your network's output only approximate? You might want to think about building a fuzzy expert system to handle the estimates.

Do you need statistical or probabilistic mappings of the input data? In this case just about the only choice is the Kohonen network, which models the probability distribution function of the training data. The learning vector quantization form of a Kohonen network can also be used in statistical clustering analysis applications.

Are you doing image processing? Bone up on traditional image-processing techniques and plan to use them as much as you can. We can't build neural networks big enough to manage a one-million-element input pattern, so use the standard techniques to reduce the size of the input dramatically. Learn about Fourier transforms (with circle and wedge cuts to reduce the transform's dimensionality), tiling the input, edge and surface detection, and many more. Then consider one of the architectures that uses lateral inhibition, as well as the simpler

Consider if your application has any special constraints that would affect the choice of network architecture. If the network must learn on-line, needs to achieve a particular speed either during training or processing, or must learn all the time, the network of choice may not be backpropagation.

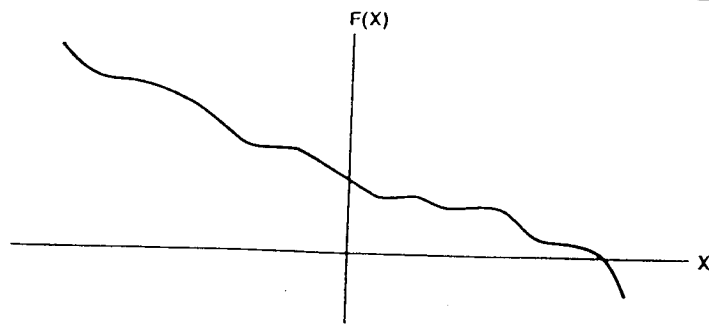
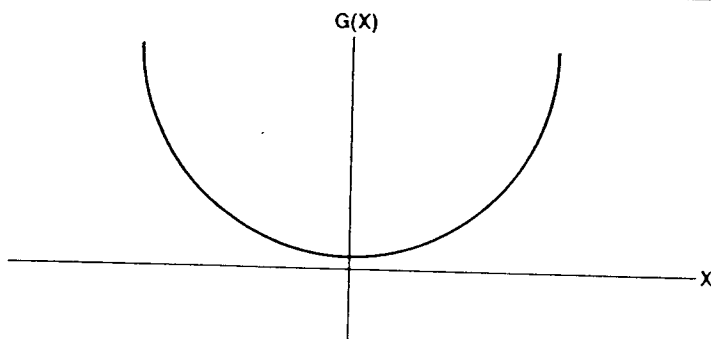


FIGURE 1. Monotonic function.

FIGURE 2. Nonmonotonic function.

backpropagation networks and its variations. Backpropagation is a wonderful network for many applications and is particularly useful for pattern-recognition tasks. Classic



backpropagation networks use supervised training, however, so you have to know the exact answer for each training pattern. It's not a particularly good choice for an associative memory application because it may tend to generate new or interpolated answers. It is especially bad at learning problems that are not "monotonic." For example, suppose you are trying to teach a backpropagation network the function shown in Figure 1. In this function, every time the argument increases, the function value decreases. This is monotonic behavior, and a backpropagation network can learn this problem quite handily. (It would do equally as well with a function that increases as the argument increases.) On the other hand, a function like that shown in Figure 2 is not monotonic; backpropagation will have more trouble learning it. (Try teaching a backprop network the square of a series of positive and negative integers. Then try teaching it the cube of the same numbers. The square is nonmonotonic; the cube is monotonic.) For multidimensional data it is frequently very difficult to determine monotonicity, but if you can do so, it will help you determine whether a simple backpropagation network will learn the problem easily. Backpropagation networks also don't cope well with training sets that are very large, either in terms of the size of

VME

REAL TIME NEURAL PROCESSING

ISA

- * 64 ANALOG INPUTS
- * 64 ANALOG OUTPUTS
- * COMPLETE CHIP CONTROL
- * DIGITAL I/O
- * MONITOR ALL INPUTS AND OUTPUTS

ETANN ULTIMA:

- BASED ON INTEL ETANN CHIP
- >2 BILLION XPS
- VME
- BOTH ANALOG AND DIGITAL I/O.
- MONITOR, CONTROL, AND INHIBIT ALL SIGNALS AND CHIP VALUES

0491E1-ISA:

- BASED ON INTEL ETANN CHIP
- >2 BILLION XPS
- ISA
- DIGITAL IN AND OUT
- MONITOR AND CONTROL ALL CHIP VALUES AND I/Os.
- COMPATIBLE WITH BRAIN MAKER

ETANN BASIC:

- BASED ON INTEL ETANN CHIP
- >2 BILLION XPS
- STAND ALONE
- ANALOG IN AND OUT

SLICE

SHIPMENT OF ISA AND VME CARDS WITH AMERICAN NEURALOGIX'S SLICE CHIP IN JANUARY

Rapid Imaging Inc

PHONE: 407-851-3163

5955 T. G. LEE BLVD, SUITE 150, ORLANDO, FLORIDA 32822

each input pattern or in the number of examples in the training set. They usually don't solve categorization problems well.

This article is not meant to be a backpropagation network bashing session—backprop remains the number one network of choice in applications because of its ease of use and reliable performance. But keep your mind open to other possibilities, too. Just because backpropagation can solve the problem doesn't mean that it's the best solution for your circumstances. Other constraints such as real-time learning or high-speed processing may interfere with it as the final system solution.

Just remember: When contemplating that tray of neural-network "goodies," be a little adventurous and try something you've never tasted before. With a thoughtful selection, and careful consideration of all the problem constraints, a new network architecture may satisfy your needs even better than that old favorite backpropagation. ☆

Maureen Caudill is author of *In Our Own Image* (Oxford University Press, 1992) and coauthor of three other books on neural networks from MIT Press. A San Diego consultant specializing in neural network technology, she is a frequent speaker and teaches in the University of California, San Diego's Extension certificate program on Intelligent Systems Technologies. She is a frequent contributor to *AI Expert* and other technical magazines.

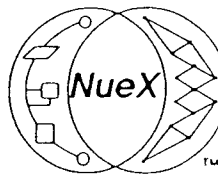
AI EXPERT IS LOOKING FOR A FEW GOOD ARTICLES

If you've got real-world experience developing and deploying AI solutions in commercial environments, *AI Expert* wants you.

We're looking for down-to-earth, practical insight aimed at professional system developers implementing knowledge-based solutions. If you've ever thought about writing for a worldwide audience, here's your chance. Send abstracts to:

Alan Zeichick, Editor
AI Expert
600 Harrison St.
San Francisco, Calif. 94107

Command Neural Networks and Expert Systems on the Mac®



Hybrid
Neural Network
Expert System
Environment

NueX™ combines the complementary power of neural networks and expert systems. Use *NueX* to explore neural and rule based computing. Build intelligent applications using the *NueX* network and rule editors. Add HyperCard® interface. Field proven software in applications ranging from remote sensing to structural monitoring, from financial forecasting to physiological monitoring.

NueX™ for the Macintosh® available for an introductory price of \$295.00. HyperCard 2.1 included. PC Windows® version available 2Q92. Demo version and government/university discounts available.

Charles River Analytics Inc.

55 Wheeler Street, Cambridge, MA 02138

Tel: (617) 491-3474

Fax: (617) 868-0780

Email: nuex@crasun.cra.com

Call for Free Demo
(617) 491-3474

N-TRAIN™

NEURAL NETWORK LIBRARIES, DEVELOPMENT TOOLS & SYSTEMS

Develop high-performance systems that *learn from experience*

N-TRAIN for C makes neural network application development quick and easy

- ... support for 32-bit compilers: large problems are easy to address
- ... algorithms coded in double-precision for reliable, numerically stable learning
- ... optimal use of co-processor means fast results—preliminary tests show N-TRAIN to be about 50% faster on large problems than BrainMaker Professional—the so-called "fastest" neural development shell
- ... clean programmer interface: logical, easy to use function calls make programming a pleasure
- ... fine control over detail: learning rates, transfer functions, etc. may be controlled both globally and on a per-layer basis

N-TRAIN neural shell for the 386/486 or Microway's 860 card is the tool of choice for neural systems development. Based on N-TRAIN for C, it is fast, reliable and powerful. Trains nets on difficult data, easily handles large problems, unlimited network size, allows for complete user control. As one user said:

"Truly trains and converges...I previously used [names 2 popular tools], but N-TRAIN is the best and fastest!" —Marc Chaikin ("Chaikin Oscillator")

ALSO: N-TRAIN Run-Time DLL to access nets from within any *Windows* language or application; *TradeNet* to access nets from within Omega Research's *TradeStation*, and other pretrained neural systems for financial markets.

For *free Guide to Information & Products* and our article "Developing Neural Network Forecasters..." (*Technical Analysis*, April 1992) contact:

SCIENTIFIC CONSULTANT SERVICES, INC.

20 Stagecoach Rd., Selden, NY 11784; (516) 696-3333

Ad prepared 10/92. Copyright (c) 1992 SCS, Inc. All rights reserved.

BUYER'S GUIDE TO NEURAL NET SOFTWARE

NEURAL NETWORK

SPECIAL REPORT

Compliments of Adaptive Solutions

INTRODUCING NEURAL NETS

Here's How They Work

PROGRAMMING IN C++

The Fastest Way to Fly

A NEURAL TOWER OF HANOI

Old Problem, New Solution

TIPS FOR DATA PREPARATION

Tricks for All Seasons

AVOIDING THE BACKPROP T

How to Structure a Network

ALGORITHMS & APP

Everything You Need to Know

U.S./Canada \$7.95



0 74470 77105 0

BROUGHT TO YOU BY



THE MAGAZINE OF ARTIFICIAL
INTELLIGENCE IN PRACTICE