Alternative Neural Network Training Methods

Vincent W. Porto, Orincon Corporation
David B. Fogel and Lawrence J. Fogel, Natural Selection, Inc.

THIS ARTICLE INVESTIGATES THREE POTENTIAL NEURAL NETWORK TRAINING ALGORITHMS IN PROCESSING ACTIVE SONAR RETURNS. ALTHOUGH ALL THREE METHODS GENERATE REASONABLE PROBABILITIES OF DETECTION AND FALSE ALARM IN DISCRIMINATING BETWEEN MAN-MADE OBJECTS AND BACKGROUND EVENTS, THE STOCHASTIC TRAINING METHODS OF SIMULATED ANNEALING AND EVOLUTIONARY PROGRAMMING OUTPERFORM BACK PROPAGATION.

NEURAL NETWORKS ARE PARALlel structures made up of nonlinear processing nodes that are connected by fixed or variable weights. They can provide arbitrarily complex measurable decision mappings and are well suited for use in pattern recognition. Neural network paradigms generally fall into two categories: *unsupervised* and *supervised learning*.

Unsupervised learning networks use the neural network topology as a self-organizing clusterer where decision regions are created with respect to the similarity of the input exemplars to other previously observed exemplars. The network adapts its outputs to minimize a function of the spacing between the elements in each organized cluster of data points. This paradigm requires no a priori clustering information.

Supervised learning networks operate on input exemplars that are associated with a desired output. Such learning involves iteratively training the neural network to approximate this mapping. System designers often use multilayer perceptrons[1] (see Figure 1) in supervised learning applications because they can perform arbitrarily complex decision mapping.[2] While this article restricts itself to supervised learning networks, the stochastic training methods investigated also apply to other topologies and unsupervised algorithms.

Multilayer perceptrons allow for continuous-valued inputs and outputs. The internal layers transform the input vectors into the output space. Internal weights and bias terms define the output of the network given the presented exemplar. This mapping of input exemplars on the network topology defines a response surface over an n-dimensional hyperspace where there are n weight and bias terms to be adapted. Various algorithms[3] can be used to search for the set of weights and biases that minimize selected functions of the error between the actual and desired outputs (such as the mean squared error).

Typical response surfaces often possess local minima. Optimization techniques based on gradient descent may stagnate at these potentially suboptimal solutions, rendering the network incapable of sufficient performance. Second-order Newton and quasi-Newton methods may also fall prey to such entrapment.

Many tricks have been invented for avoiding this problem, such as restarting with a new random set of weights, training with noisy exemplars, and perturbing the weights when they appear to prematurely converge. While these methods may lead to improved solutions, there is no guarantee that such minima will not also be only locally optimal. Further, the same suboptimal solution may

0885-9000/95/$4.00 © 1995 IEEE

be rediscovered, leading to fruitless oscillatory training behavior.

Stochastic techniques offer an alternative to conventional gradient methods. Both simulated annealing[4] and simulated evolution[5] can serve to generate weight and bias sets. Neither suffers from entrapment in local minima (that is, they have asymptotic global, as opposed to only local, convergence properties).

This article assesses three methods of training neural networks, (1) back propagation, (2) simulated annealing, and (3) evolutionary programming, in terms of their performance in distinguishing between sonar returns that reverberate from a man-made metal sphere and those caused by sea mounts, fish and plant life, background noise, or similar anomalies. Although we focus on sonar signal processing, the results extend to more general applications of pattern classification.

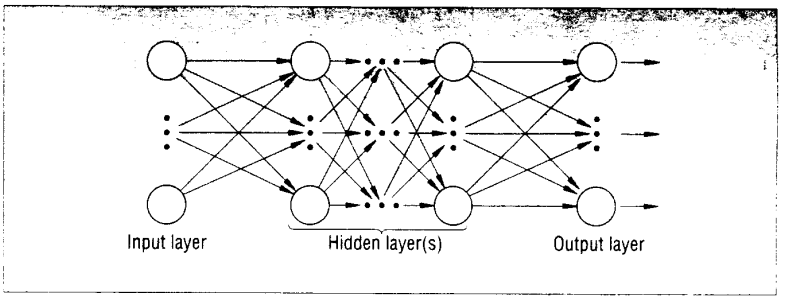

Figure 1. The architecture for a multilayer perceptron. Each node connects to all nodes in the forward layer. The number of input nodes corresponds to the dimensionality of the exemplars and the number of output nodes corresponds to the number of distinct classes. Each node processes the weighted input of all nodes in the previous layer. Not explicitly shown in the figure is a bias term that is also associated with each node.

Training algorithms

Back propagation is a familiar neural network training method, and simulated annealing and evolutionary programming can also be used to optimize networks.

Back propagation. This gradient descent technique uses classification errors to adjust weights and bias terms. After receiving a set of exemplars, this method uses the difference between the output and the desired activations to adjust the weights between each neural layer. Specifically:

$$\Delta_p \omega_{ji} = \eta \delta_{pj} o_{pi}$$

where $\Delta_p \omega_{ji}$ is the change in the weight between node i and node j, η is a scaling constant, and δ_{pj} is the negative partial derivative of the squared error between the target and the output for the pth pattern with respect to the sum of the weighted outputs at node i. Therefore,

$$\delta_{pj} = -\partial E_p / \partial net_{pj}$$

$$E_p = 0.5 \sum (t_{pj} - o_{pj})^2$$

$$net_{pj} = \sum w_{ji} o_{pi}$$

where t_{pj} is the target value for the jth node given the pth exemplar and o_{pj} is the output for the jth node given the pth exemplar (or is an input value if the jth node is an input node). Rumelhart et al.[6] indicates that the value of δ_{pj} can be calculated as

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj})$$

where $f'_j(net_{pj})$ is the derivative of the activation function associated with the jth node if the node is an output unit, or

$$\delta_{pj} = f'_j(net_{pj}) \sum \delta_{pk} w_{kj}$$

whenever the unit is not an output unit (hidden node). Here k refers to nodes in the forward layer for which the values of δ_{pk} have already been determined. Rumelhart[6] contains some notational errors and omissions; the reader may wish to examine Rogers and Kabrisky[7] for an alternative description of back propagation.

A crucial problem with this method is the requirement for infinitesimally small step sizes (learning rates) to guarantee asymptotic convergence to a minimum point. In practice, computational time limitations preclude this possibility. Most applications estimate an appropriate step size, which often leads either to extremely slow convergence or to oscillatory behavior without convergence. If the step size is too small, the total number of iterations required will become prohibitive. With too large a step size, the process may not converge to any minima at all. Newton and quasi-Newton (quadratic) methods use higher-order derivative information (or approximations thereof) to gain increased rates of convergence. But this requires substantially more computational time per epoch or iteration. Also, these methods are still susceptible to problems in selecting an appropriate step size.

Simulated annealing. As a stochastic process,[8] this generalized Monte Carlo technique uses a monotonically decreasing variance controlled by the temperature-annealing schedule. The annealing algorithm will converge to the global optimum on a function containing multiple extrema if the temperature falls according to an appropriate cooling schedule.[9]

The annealing process is analogous to a ball rolling in a box containing a nonlinear surface topology—one with valleys and moguls. To set the ball in motion, we shake the box, decreasing the variance over time. Our shaking must be sufficiently violent to enable the ball to escape from local minima, yet sufficiently gentle that it does not drive the ball away from the global minimum. Only by continuously decreasing the amplitude of the shaking can we accomplish this.

Specifically, following Kirkpatrick et al.[8] and Bohachevsky et al.,[10] we can implement a basic annealing algorithm as follows. Suppose $E(\boldsymbol{x})$ is the total error over all patterns given the n-dimensional weight set $\boldsymbol{x}$.

1) Let $\boldsymbol{x_0}$ be an arbitrary starting point (either specified or selected at random).
2) Calculate $E(\boldsymbol{x_0})$.
3) If $E(\boldsymbol{x_0}) < \varepsilon$, halt.
4) Generate n independent standard normal variates $Y_1, \ldots, Y_n$ and compute the components of $\boldsymbol{U}$: $U_i = Y_1/(Y_1^2 + \ldots + Y_n^2)^{0.5}$, $i = 1, \ldots, n$.
5) Set $\boldsymbol{x^*} = \boldsymbol{x_0} + (\Delta r)\boldsymbol{U}$.
6) Calculate $E(\boldsymbol{x^*})$.
7) If $E(\boldsymbol{x^*}) \le E(\boldsymbol{x_0})$, set $\boldsymbol{x_0} = \boldsymbol{x^*}$, if $E(\boldsymbol{x^*}) < \varepsilon$, halt, otherwise go to step 4.
8) If $E(\boldsymbol{x^*}) > E(\boldsymbol{x_0})$, set $p = \exp(-(E(\boldsymbol{x^*}) - E(\boldsymbol{x_0}))/t)$
   a) Generate a uniform [0,1] variate V.
   b) If $V \ge p$, go to step 4.
   c) If $V < p$ set $\boldsymbol{x_0} = \boldsymbol{x^*}$, go to step 4.

The values for Δr and t are determined empirically.

If we hold the temperature t (analogous to

the shaking motion) constant, the annealing algorithm reverts to a pure Monte Carlo method. At $t = 0$, the method approximates a gradient descent algorithm. As noted earlier, t is generally a decreasing function of time. If the temperature falls too quickly, the system will tend to freeze in local optima. But slow cooling schedules result in inefficient searches.

For the current research, we set the temperature inversely proportional to the iteration number. Further, rather than use a uniform distribution for generating new trials, we employed a Cauchy probability density function. This continuous distribution allows for occasional large jumps, providing additional ability to escape from local minima.

Evolutionary programming.[11] This stochastic optimization algorithm method can discover suitable solutions to complex problems. In training a neural network, the algorithm starts with an original population (parents) of weight sets that it evaluates by examining the corresponding outputs from the neural network given the set of exemplars. It can apply any payoff function as required from the operational costs of the various correct and incorrect classifications. For the experiments in this article, we restricted attention to the mean squared error over all patterns.

Random mutation of these parents creates new solutions. Specifically, each weight and bias term is perturbed by a Gaussian random variable with zero mean and a variance equal to the parent's mean squared error. (Fogel[12] describes other more sophisticated methods for generating offspring.) The algorithm scores offspring weights the same way it did for their parents. For the new parents, it probabalistically selects the most suitable solutions for the task at hand.

The algorithm performs this selection by requiring each weight set to compete against a selected number (for example, 10) of other randomly chosen weight sets. If the error score of the former set is lower than or equal to the chosen opponent in each competition, that weight set receives a "win." The algorithm retains those weight sets with the most wins as parents of the next generation. It repeats this procedure until either it achieves a solution of sufficient quality or exhausts the available computer time.

Specifically, following Fogel,[12] we generally implement the most basic algorithm as follows. Given $E(\boldsymbol{x})$, the error function for any n-dimensional vector of weights and biases defining a candidate network:

1) Select an initial population of P candidate vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_P$, where each vector comprises n floating point values. The distribution for these initial values is typically uniform across [–0.5, 0.5] but other choices such as Gaussian are also reasonable.
2) Calculate $E(\boldsymbol{x}_i)$, $i=1, ..., P$.
3) From each parent vector $\boldsymbol{x}_i$, $i=1, ..., P$, create a corresponding offspring vector $\boldsymbol{x}_i$, $i=P+1, ..., 2P$, by adding a Gaussian random variable with mean zero and variance proportional to $E(\boldsymbol{x}_i)$ to every component (weight and bias).
4) Calculate $E(\boldsymbol{x}_i)$, $i=P+1, ..., 2P$.
5) Conduct a series of c pairwise competitions between each vector and randomly selected opponents. Let $\boldsymbol{x}_i$ be the vector being conditioned upon. Let $\boldsymbol{x}_0$ be the randomly selected opponent vector. Assign a "win" to $\boldsymbol{x}_i$ if $E(\boldsymbol{x}_i) \leq E(\boldsymbol{x}_0)$. Rank all of the vectors by their associated number of wins [0, ..., c].
6) Select those vectors with the most wins to be parents for the next generation.
7) If the available time has been exhausted or a sufficient solution has been discovered then halt, else proceed to step 3 and continue.

GENETIC ALGORITHMS DIFFER FROM OTHER EVOLUTIONARY ALGORITHMS IN THAT THIS APPROACH EMPHASIZES THE USE OF SPECIFIC OPERATORS, SUCH AS CROSSOVER, THAT MIMIC THE FORM OF GENETIC TRANSFER IN BIOTA.

Other methods of simulating evolution are available, including genetic algorithms[13] and evolution strategies.[14] The differences between evolutionary programming and evolution strategies are minor, so we should expect comparable performance. Genetic algorithms differ from other evolutionary algorithms in that this approach emphasizes the use of specific operators, such as crossover, that mimic the form of genetic transfer in biota. While there may be specific circumstances for which such operators are especially appropriate, evidence suggests that they are not required for successful optimization and that their use can be detrimental, providing inferior performance to evolutionary computations that do not include crossover (for example, see Fogel[12] and Fogel and Stayton[15]). Further, crossover may not be especially appropriate for application to neural structures,[16] although it has been applied with success.[17] We leave comparisons between evolutionary programming and genetic algorithms on the problem of concern addressed in this article for future research.

Method and materials

Previous research in neural network pattern classification shows the feasibility of using active sonar returns for the automated detection of marine objects of interest.[18] This research indicated that active returns contained sufficient information to successfully differentiate man-made metal spheroid objects from naturally occurring objects, such as land masses and rocks, and background reverberation.

Continuous transmission frequency modulation (CTFM) active sonar units transmit a continuous sawtooth swept-frequency waveform and receive reflected returns, represented by time-delayed versions of the transmitted signal from objects within the field of view. The received array output signal is heterodyned with the current transmitted acoustic waveform and low-pass filter to produce a baseband return signal in the 500 Hz to 3 kHz frequency range. The frequency of a return is directly proportional to the range of a reflecting object. We sampled the basebanded analog signal output from the sonar unit at 7,810 Hz.

The data sets used for the current experiments came from a test conducted in the Pacific coastal waters near Catalina Island, offshore from Los Angeles. The sonar (operated at a 500-yard maximum range setting) was towed through a field containing four metal spheres, three feet in diameter, tethered at varying depths (5, 15, and 25 feet). These test sets contained a large number of varied, brief events. Augmenting this data set was a second data set consisting of similar objects placed in a field with the sonar unit operating on longer range settings (1,500 yards maximum).

The test used multiple time series pulse return exemplars for each of the output classes. To provide an alternate output class for the classifier, the test included template exemplars containing representative samples of land masses, large rocks, and background and ship wake reverberation. This allowed for training against potential clutter. The comprehensive data set included over 475 specific returns.

Parametric functions of spectral transforms of the CTFM sonar data served as a preprocessing feature extraction technique. We generated high-resolution spectra of the time-series samples using 512-point fast-Fourier transforms of the input data. A Hamming window function provided spectral windowing.

Figure 2 presents a sample of raw data. This provided 256 spectral bins of width 15.25 Hz, corresponding to 70 inches in range resolution (210 inches in the second data set). Again, we analyzed the data for the peak power over all spectral bins, this time using sliding window sums over eight contiguous bins for detection and centering of the spectral energy from the return. We examined these eight raw spectral bin amplitudes for use as inputs to the neural network. Further, we derived four different measures of spectral "purity," which we used as simple parametric functions on these eight spectral bins.

Two ratios provided a measure of the return's bandwidth. These are the ratios of peak magnitude to its two adjacent bin magnitudes and to the censored mean of the background magnitudes (remove the peak magnitude then average over the remaining bins). These ratios also encode the relative signal-to-noise information. A strong return at a single frequency from a strong spectral reflector, such as one of the spherical metal objects, differs noticeably from that of a rock whose spectral return, while perhaps greater in total energy, spreads over a larger number of adjacent bins.

Third was the ratio of the spectral peak magnitude plus the magnitude of the two adjacent side bins to the censored mean level. Last came the ratio of the spectral peak magnitude plus the two adjacent side bin magnitudes to the sum of the magnitudes of the next two adjacent side bins (those bins on either side of the two bins immediately adjacent to the peak). These four spectral ratios effectively encode normalized signal-to-noise and slope information around the peak spectral magnitude.

A three-layer perceptron, containing two hidden layers with a single output node, served for all classification tests. The input feature space consisted of four input nodes mapping the spectral ratio input features as described just now. We deemed two hidden layers with four nodes in each layer sufficient for these experiments. The single output neuron served to create a binary discrimination region for differentiating between man-made metal sphere returns and those originating from natural objects or background reverberation. This simple architecture has been proven capable of learning appropriate discrimination functions with this data set.[18]

Our test implemented both continuous-valued inputs and outputs. As Figure 3 shows, it used a conventional fan-in nodal architecture calculating weighted sums of the inputs with its output passing through a sigmoid function ranging over [0,1].

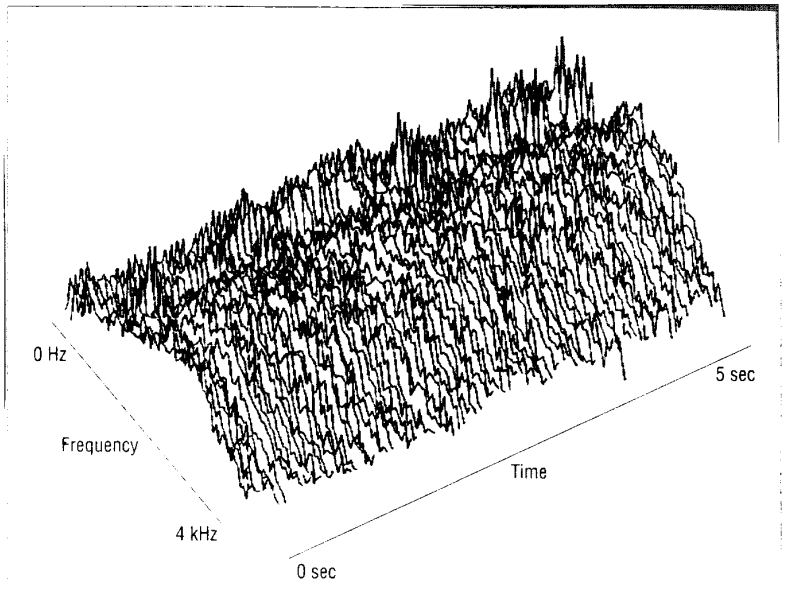

Figure 2. A spectral display of a sample-active sonar return. Frequency ranges from 0 to 4 kHz. Time ranges from 0 to 5 seconds.

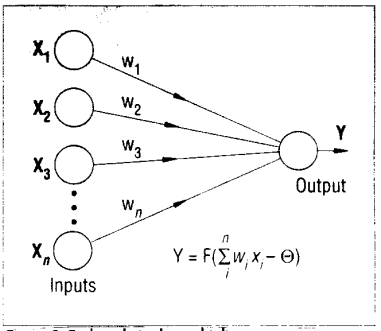

Figure 3. Each node in the multiplier perceptron performs the dot product between the associated weight vector and input vector, subtracts a variable threshold and passes the result through the sigmoid filter, $F(x)=(1+\exp(-x))^{-1}$.

Experiments

Proper classifier design requires sufficient exemplars that are representative of the underlying distributions. Such exemplars allow for the creation of transformation mappings from the input feature space to the desired output space, and are a prerequisite for robust classification. We randomly divided the available exemplars into nonoverlapping training and test sets. The training set consisted of 238 exemplars comprising 101 sphere return exemplars, 66 land mass and rock return exemplars, and 71 background reverberation exemplars. This set provided a sufficient number of training exemplars with an adequate number of potential clutter signals to train against. We reserved the remaining 237 exemplars for independent testing. Visual observation of these data indicated overlapping pairwise distributions in all of the feature space dimensions (Figure 4, next page). The data mappings were not simply convex regions that simple network topologies could learn.

Exemplars from the three output classes

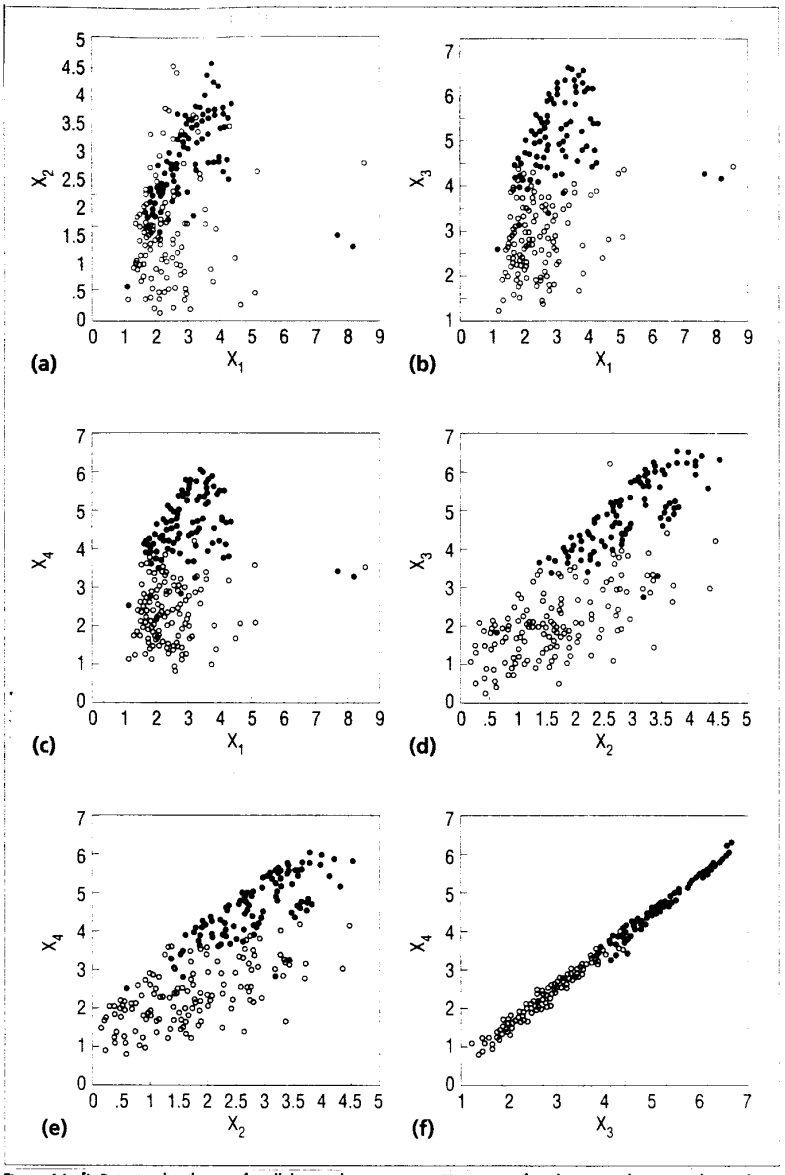

Figure 4 (a-f). Pairwise distributions for all data in the training set. X_1 = ratio of peak magnitude to two adjacent bin magnitudes; X_2 = ratio of peak magnitude to censored mean of background magnitudes; X_3 = ratio of peak magnitude plus two adjacent bins to censored mean; X_4 = ratio of peak magnitude plus two adjacent bins to the sum of the magnitudes of the next two adjacent side bins. Solid circles indicate returns from metal spheres. Open circles indicate returns from natural sources or background reverberation.

fell into two hypothesis classes: (1) returns from the metal spheres, and (2) returns from the natural objects and other background. In each case, after training the network, we processed the independent test data set through the network to ascertain the accuracy of the classification methods. Sampling of the single-output neuron used a threshold of 0.5 for output selection.

Figure 5 shows a typical learning rate for back propagation. Using a fixed step size of 0.08, we executed 20 independent random trials. We considered training to be adequate when the output RMS error over all exemplars was less than 0.08, which typically occurred after 500 to 800 epochs. Table 1 indicates the best- and worst-case performance on the training and test sets using back propagation. Note that the worst-case performance on the training and test sets indicates entrapment in a local minimum.

We implemented simulated annealing to train the previous network topology using the same data sets. The exhibited learning behavior was not monotonic and substantially slower in convergence than back propagation (Figure 6). But the annealing algorithm never became entrapped in any of the local minima that trapped back propagation. Table 2 shows the best- and worst-case performance over 20 trials using this training technique. We stopped training runs when either the total RMS output error was less than 0.08 or 5,000 iterations were completed.

We implemented evolutionary programming as a training method for the same network topology. In training, we initialized a population of 50 parent vectors in accordance with a standard normal distribution (zero mean and unit variance). All vectors were scored with respect to the sum of the squared error between the target output (1.0 for the spheres, 0.0 otherwise) and the actual network output determined by the vector of 45 (network) weights and biases.

We set the variance for mutation equal to the mean squared error of the parent and retained the best 50 vectors to serve as parents for the next generation in accordance with the probabilistic formulation described earlier. We halted the evolution after 500 generations. Figure 7 depicts a typical learning rate. Table 3 shows best- and worst-case results over 20 trials.

Approximately 500 generations were necessary for accurate pattern discrimination performance with an output RMS error of 0.08 or less. These tests indicate an empirical probability of correct classification ranging between 0.94 and 0.98 for the metal spheres and a probability of misclassification ranging between 0.015 and 0.074 for the other classes, given a 0.5 output threshold. In all cases, the output neuron activation levels typically ranged from 0.6 to 0.99 for the metal sphere and 0.02 to 0.25 for the other combined class (natural and background).

THE EXPERIMENTAL EVIDENCE indicates the suitability for processing active sonar returns through neural networks to discriminate between man-made spherical objects and background events. All three

training algorithms provided reasonable probabilities of detection and false alarm. Back propagation was not as consistent as the stochastic optimization techniques and repeatedly stalled at suboptimal weight sets that did not yield satisfactory results.

The typical back-propagation algorithm is limited to multilayer perceptrons. Designers can construct other search techniques for various network topologies, but must address specific considerations in each case. In contrast, they can implement stochastic optimization algorithms such as simulated annealing and evolutionary programming to optimize the weighted interconnections of any generalized network for any payoff function. Feedback and feedforward loops can extend over more than one layer without computational difficulties. The physical realization of the network does not constrain the stochastic training algorithms, and indeed, some efforts have been made to evolve both the weights and the topology of a network simultaneously.[12,16,19]

The response (error) surfaces generated in real-world neural network pattern classification problems are typically pocked with multiple optima. While a gradient technique such as back propagation is guaranteed to find locally optimal solutions if the step size tends to zero, these local solutions may fail to provide satisfactory performance on the given training set. Often, additional nodes are added to the network until the training algorithm discovers a suitable solution. But the resulting network can be severely overdefined.

Any training data can be correctly classified if the network has sufficient degrees of freedom. Such a network may not perform well on new data taken independently from the training data. Stochastic methods such as simulated annealing and evolutionary programming can overcome local optima and result in smaller and effective networks which may be more robust.

Other research has offered annealing and evolutionary optimization methods that should generate faster converging algorithms than the simplified versions presented here.[12,20] If implemented on a highly parallel processing computer, the computational costs for both simulated annealing and evolutionary programming are roughly equivalent, but the runtime performance from evolutionary training may be better as the standard annealing algorithm does not take advantage of a population of contending solutions.

Table 1. The best- and worst-case classification performance using back propagation over 20 trials. The first number indicates the number of correct classifications. The second number indicates the total number of events in that class.

	Best		Worst	
	Training set	Test set	Training set	Test set
Metal sphere	98/101	98/101	65/101	77/101
Background	127/137	131/136	100/137	103/136

Table 2. The best- and worst-case classification performance using simulated annealing over 20 trials. The first number indicates the number of correct classifications. The second number indicates the total number of events in that class.

	Best		Worst	
	Training set	Test set	Training set	Test set
Metal sphere	98/101	98/101	97/101	98/101
Background	127/137	131/136	127/137	131/136

Table 3. The best- and worst-case classification performance using evolutionary programming over 20 trials. The first number indicates the number of correct classifications. The second number indicates the total number of events in that class.

	Best		Worst	
	Training set	Test set	Training set	Test set
Metal sphere	94/101	95/101	96/101	99/101
Background	127/137	134/136	130/137	126/136

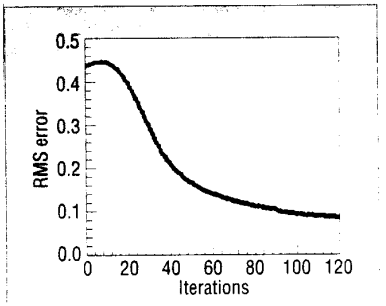

Figure 5. A typical learning rate for back propagation.

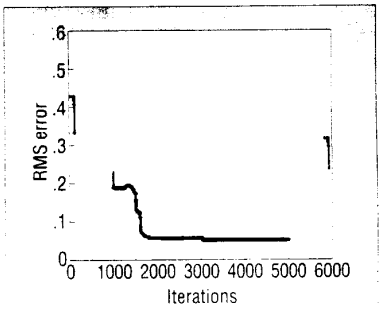

Figure 6. A typical learning rate for simulated annealing.

Acknowledgments

The authors would like to thank Peter J. Angeline, Thomas Bäck, Patrick K. Simpson, and four anonymous referees for their comments and criticisms of this article.

References

1. F. Rosenblatt, "The Perceptron, a Perceiving and Recognizing Automaton," Project PARA, Cornell Aeronautical Laboratory, Report 85-640–1, Buffalo, NY, 1958.
2. L.K. Jones, "Constructive Approximations for Neural Networks by Sigmoidal Functions," *Proc. IEEE,* Vol. 78, 1990, pp. 1586–1589.

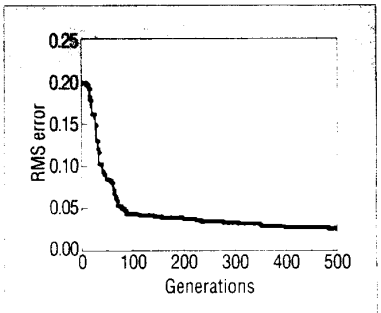

Figure 7. A typical learning rate for evolutionary programming.

3. P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,* doctoral dissertation, Harvard, Cambridge, Mass., 1974.

4. D. Ackley, G. Hinton, and T. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, Vol. 9, 1985, pp. 147–169.

5. D.B. Fogel, L.J. Fogel, and V.W. Porto, "Evolving Neural Networks," *Biological Cybernetics*, Vol. 63, 1990, pp. 487–493.

6. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, D.E. Rumelhart and J.L. McClelland, eds., MIT Press, Cambridge, Mass., 1986, pp. 318–362.

7. S.K. Rogers and M. Kabrisky, *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*, SPIE Optical Engineering Press, Bellingham, Wash., 1991.

8. S. Kirkpatrick, C.D. Gellatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 22, 1983, pp. 671–680.

9. S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721–741.

10. I.O. Bohachevsky, M.E. Johnson, and M.L. Stein, "Generalized Simulated Annealing for Function Optimization," *Technometrics*, Vol. 28, 1986, pp. 209–217.

11. L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley, New York, 1966.

12. D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence,* IEEE Press, Piscataway, N.J., 1995.

13. K.A. De Jong, "Genetic Algorithms: A 25-Year Perspective," *Computational Intelligence: Imitating Life*, J.M. Zurada, R.J. Marks, and C.J. Robinson, eds., IEEE Press, 1994, pp. 125–134.

14. H.-P. Schwefel, *Evolution and Optimum Seeking*, Springer-Verlag, Berlin, 1995.

15. D.B. Fogel and L.C. Stayton, "On the Effectiveness of Crossover in Simulated Evolutionary Optimization," *BioSystems*, Vol. 32, No. 3, 1994, pp. 171–182.

16. P.J. Angeline, G.M. Saunders, and J.B. Pollack, "An Evolutionary Algorithm That Constructs Recurrent Neural Networks," *IEEE Trans. Neural Networks*, Vol. 5, 1994, pp. 54–65.

17. J.D. Schaffer, "Combinations of Genetic Algorithms with Neural Networks or Fuzzy Systems," *Computational Intelligence: Imitating Life,* J.M. Zurada, R.J. Marks, and C.J. Robinson, eds., IEEE Press, 1994, pp. 371–382.

18. V.W. Porto, "Detection of Undersea Objects Using Neural Networks," *Proc. 23rd Asilomar Conf. on Signals, Systems and Computers*, Vol. 1, R.R. Chen, ed., Maple Press, San Jose, Calif., 1989, pp. 376–380.

19. J.R. McDonnell and D. Waagen, "Evolving Recurrent Perceptrons for Time-Series Modeling," *IEEE Trans. Neural Networks*, Vol. 5, 1994, pp. 24–38.

20. L. Ingber and B. Rosen, "Genetic Algorithms and Very Fast Simulated Annealing—A Comparison," *Math. and Comp. Mod.*, Vol. 16, No. 11, 1992, pp. 87–100.

Vincent W. Porto is a senior principal engineer at Orincon Corporation in San Diego, California, where his research interests include neural networks, evolutionary computation, and sonar and image processing. He received the BS in mathematics from the University of California at San Diego in 1981, and has completed additional graduate work at UCSD. He has served as the finance chair for each of the annual conferences on evolutionary programming (1992-1995) and was a founding officer the of the Evolutionary Programming Society in 1991. He can be reached at: Orincon Corp., 9363 Towne Centre Dr., San Diego, CA, 92121; porto@orincon.com.

David B. Fogel is chief scientist of Natural Selection, Inc., in La Jolla, California. His research interests include the theoretical foundations of evolutionary computation and the application of evolutionary optimization to real problems in medicine, industry, and defense. He received the PhD in engineering sciences from UCSD in 1992, and has subsequently taught undergraduate and graduate courses in evolutionary computation, stochastic processes, and statistical process control at UCSD. He is a senior member of the IEEE, serves as the chairman of the IEEE Neural Network Council's Committee on Evolutionary Computation, and is the technical chairman for the 1995 IEEE Conference on Evolutionary Computing to be held in Perth, Australia, Nov. 1995. He is also associate editor for *BioSystems* and the *IEEE Transactions on Neural Networks*, and a member of the editorial boards of the journals *Evolutionary Computation* and *Fuzzy Sets & Systems*. Dr. Fogel is the author of *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (IEEE Press, 1995). He can be reached at: Natural Selection, Inc., 1591 Calle De Cinco, La Jolla, CA, 92037; fogel@sunshine.ucsd.edu.

Lawrence J. Fogel is president of Natural Selection, Inc. in La Jolla, California. His research interests include the engineering potential of evolutionary programming as well as the evolution of human intelligence and consciousness. He received the PhD in engineering from UCLA in 1964. He conducted much of the first research in evolutionary programming in the early 1960s and is co-author of *Artificial Intelligence through Simulated Evolution*, published by John Wiley, 1966. He was the founding editor of the *Journal of Cybernetics* (1970) and is a member of the editorial boards of *Evolutionary Computation* and *BioSystems*. He serves as the general chairman of the Fifth Annual Conference on Evolutionary Programming, to be held in San Diego, CA, Feb. 1996. He can be reached at: Natural Selection, Inc., 1591 Calle De Cinco, La Jolla, CA,. 92037; fogel@superc.nosc.mil.