Global Optimization for Neural Network Training *

YI SHANG and BENJAMIN W. WAH
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801
{*shang, wah*}*@manip.crhc.uiuc.edu*

June 24, 1996

Abstract

In this paper, we study various supervised learning methods for training feed-forward neural networks. In general, such learning can be considered as a nonlinear global optimization problem in which the goal is to minimize a nonlinear error function that spans the space of weights using heuristic strategies that look for global optima (in contrast to local optima). We survey various global optimization methods suitable for neural-network learning, and propose the *NOVEL method*, a novel global optimization method for nonlinear optimization and neural network learning. By combining global and local searches, we show how *NOVEL* can be used to find a good local minimum in the error space. Our key idea is to use a user-defined trace that pulls a search out of a local minimum without having to restart it from a new starting point. Using five benchmark problems, we compare *NOVEL* against some of the best global optimization algorithms and demonstrate its superior improvement in performance.

1 Introduction

In this paper, we study various methods for the supervised learning of feed-forward neural networks. These networks perform mappings from an input space to an output space. In spite of different activation functions of neurons and connection structures, output O of a neural network can be defined as a function of inputs X and connection weights W: $O = \phi(X, W)$, where ϕ represents a mapping function.

Supervised learning involves finding a good mapping function that maps training patterns correctly as well as to generalize the mapping found to test patterns not seen in training. This is done by adjusting weights W on links while fixing the topology and activation function. In other words, given a set of training patterns of input-output pairs $\{ (I_1, D_1), (I_2, D_2), \cdots, (I_m, D_m) \}$ and an error function $\epsilon(W, I, D)$, learning strives to minimize learning error $E(W)$:

$$\min_W E(W) = \min_W \sum_{i=1}^{m} \epsilon(W, I_i, D_i). \tag{1}$$

One popular error function is the squared-error function in which $\epsilon(W, I_i, D_i) = (\phi(I_i, W) - D_i)^2$. Since $E(W) \geq 0$ for a given set of training patterns, if there exists W' such that $E(W') = 0$, then W' is a global minimum; otherwise, the W that gives the smallest $E(W)$ is the global minimum. The quality of a learned

*This research was supported in part by National Science Foundation Grant MIP 92-18715 and in part by Joint Services Electronics Program Contract N00014-90-J-1270.

Programs developed for this paper can be accessed through the World-Wide Web at http://manip.crhc.uiuc.edu.

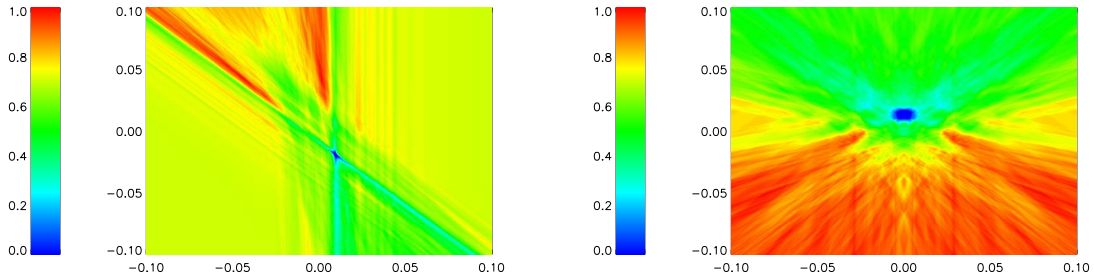

Figure 1: Two dimensional projections of the 33-dimensional error surface for a five hidden-unit 33-weight feed-forward neural network with sigmoidal activation function. The terrain is around a solution found by *NOVEL* to solve the two-spiral problem.

network is measured by its error on a given set of training patterns and its (generalization) error on a given set of test patterns.

In the form represented in (1), supervised learning can be considered as an unconstrained nonlinear minimization problem in which the objective function is defined by (1), and the search space is defined by the space of the weights.[1] Unfortunately, the terrain modeled by the error function in its weight space can be extremely rugged and has many local minima. This phenomenon is illustrated in Figure 1 that shows two contour plots of the error surface around a local minimum along two pairs of dimensions. The network here has been trained to solve the two-spiral problem (to be discussed in Section 4). Obviously, a search method that cannot escape from a local minimum will have difficulty in finding a solution that minimizes (1).

Many learning algorithms find their roots in function-minimization algorithms that can be classified into local minimization and global minimization. Local minimization algorithms, such as gradient-descent, are fast but usually converge to local minima. In contrast, global minimization algorithms have heuristic strategies to help escape from local minima.

There are many benefits in using smaller neural networks. First, they are less costly to implement and are faster, both in hardware and in software. Second, they generalize better because they avoid over-fitting the weights to the training patterns. In general, more unknown parameters (weights) induce more local minima in the error surface. Hence, the error surface of smaller networks can be very rugged and have few good solutions, making it difficult for a local minimization algorithm to find a good solution from a random starting point. This phenomenon also explains why a gradient-based local search method, such as back-propagation, can find a converged network when the number of weights is large, but have difficulty otherwise. To overcome this problem, more powerful global search methods are needed.

In this paper, we propose a novel global minimization method called the *NOVEL method*, and demonstrate its superior performance on neural network learning problems. Our major goals are to improve the neural networks learned for an application using the same amount of time as in other algorithms, as well as to find smaller networks using more time. In Section 2, we summarize previous work on unconstrained nonlinear minimization methods, and discuss their applications in neural network learning. In Section 3, we present the framework and components of *NOVEL*. To illustrate the minimization process, we show in Section 4 the learning of a neural network for solving the two-spiral problem, and compare *NOVEL* with some of the best global minimization algorithms. In Section 5, we evaluate the performance of *NOVEL* by applying it to four other benchmark problems. Finally, conclusions are drawn in Section 6.

[1] Without loss of generality, we consider minimization problems in this paper.

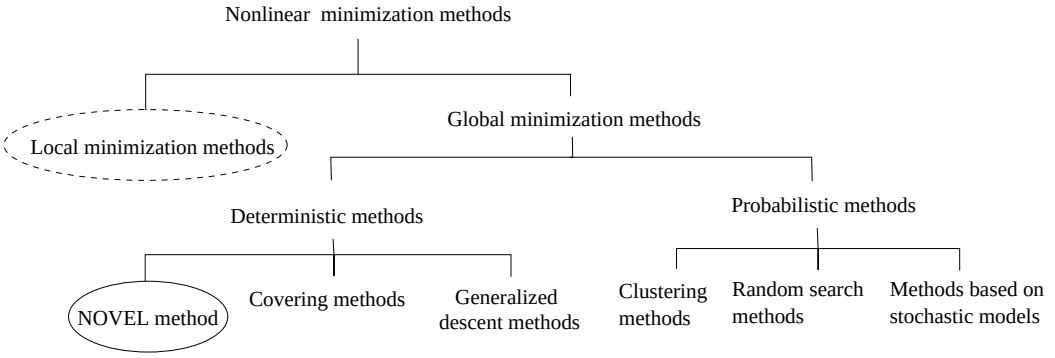

Figure 2: Classification of unconstrained nonlinear continuous global minimization methods. See side-bar for explanation of terminologies.

2 Methods for Nonlinear Unconstrained Minimization

Learning the weights of a feed-forward neural network can be considered as solving an unconstrained continuous nonlinear minimization problem. In the past, many techniques have been developed for solving similar problems in other disciplines. In this section, we summarize the key results in this area.

The task in solving an unconstrained continuous nonlinear minimization problem is to find assignments to its variables so that the given objective function is minimized. These problems are classified into uni-modal and multi-modal, depending on the number of local minima in the space of the objective function.

Neural network learning, in general, is a multi-modal nonlinear minimization problem with many local minima. Our study of $E(W)$ reveals the following features: (a) Flat regions may mislead gradient-based methods; (b) There may be many local minima that trap gradient-based methods; (c) Deep but suboptimal valleys may trap any search method; (d) Gradients may differ by many orders of magnitude, making it difficult to use gradients in any search method. A good search method should, therefore, have mechanisms (a) to use gradient information to perform local search (and be able to adjust to changing gradients) and (b) to escape from a local minimum after getting there.

Search methods can be classified into local minimization and global minimization. Local minimization algorithms, such as gradient-descent and Newton's method, find local minima efficiently and work best in uni-modal problems. Global minimization methods, in contrast, employ heuristic strategies to look for global minima and do not stop after finding a local minimum [1, 2].

Many local minimization methods have been applied to learning of feed-forward neural networks [3, 4]. Examples include back-propagation (*BP*), conjugate-gradient and quasi-Newton's methods. Local minimization algorithms have difficulties when the surface is flat (gradient close to zero), or when gradients can be in a large range, or when the surface is very rugged. When gradients can vary greatly, the search may progress too slowly when the gradient is small and may over-shoot when the gradient is large. When the error surface is rugged, a local search from a randomly chosen starting point will likely converge to a local minimum close to the initial point and a solution worse than the global minimum. Moreover, these algorithms require choosing some parameters, as incorrectly chosen parameters may result in slow convergence.

To overcome the deficiencies in local-search methods, global minimization methods have been developed. Figure 2 classifies unconstrained nonlinear global minimization algorithms. (See side-bar for further explanation.) These algorithms can be classified into probabilistic and deterministic. They use local search to determine local minima, and focus on bringing the search out of a local minimum once it gets there.

In the past, very few deterministic methods have been developed, most of which apply deterministic heuristics (such as modifying the trajectory in covering methods and adding penalties in penalty-based methods) to bring a search out of a local minimum. Other methods, like covering methods, partition a search space into subspaces before searching. All these methods do not work well when the search space is too large for deterministic methods to cover adequately.

Existing global minimization methods

Covering methods — These methods detect subregions not containing the global minimum and exclude them from further consideration. In general, this approach is useful for problems requiring solutions with guaranteed accuracy. These methods can be computationally expensive, as computation time increases dramatically as problem size increases.

Generalized descent methods — (a) Trajectory methods modify the differential equations describing the local descent trajectory. Their major disadvantage is the large number of function evaluations spent in unpromising regions. (b) Penalty methods prevent multiple determination of the same local minima by modifying the objective function, namely, by introducing a penalty term relating each local minimum found to an auxiliary function. Their problem is that as more local minima are found, the auxiliary function becomes rather flat, and the modified objective function becomes more difficult to minimize.

Clustering methods — Clustering analysis is used to prevent redetermination of already known local minima. There are two strategies for grouping the points around a local minimum: (a) retain only points with relatively low function values; (b) push each point towards a local minimum by performing a few steps of a local search. They do not work well when the terrain is very rugged.

Random search methods — These include pure random search, single-start, multi-start, random line search, adaptive random search, partitioning into subsets, replacing the worst point, evolutionary algorithms, and simulated annealing. They are simple to realize and perform well for some applications. However, they usually have many parameters that are problem-specific, leading to low efficiency when improperly applied.

Methods based on stochastic models — Most of these methods use random variables to model unknown values of an objective function. One example is the Bayesian method, which is based on a stochastic function and minimizes the expected deviation of the estimate from the real global minimum. Although very attractive theoretically, they are too expensive to be applied to problems with more than twenty variables. Further, they approximate the objective function in the average sense, which does not help when the goal is to find the minimum solution.

On the other hand, probabilistic global minimization methods rely on probability to make decisions. The simplest probabilistic algorithm uses restarts to bring a search out of a local minimum when little improvement can be made locally. This is used in learning methods such as BP. More advanced methods rely on probability to indicate whether a search should ascend from a local minimum (like in simulated annealing when it accepts up-hill movements). Other stochastic methods rely on probability to decide which intermediate points to interpolate as new starting points (like in random recombinations and mutations in evolutionary algorithms). All these algorithms are weak in either their local or their global search. For instance, gradient information useful in local search is not used well in simulated annealing and evolutionary algorithms. In contrast, gradient-descent algorithms with multi-starts are weak in global search.

Other probabilistic methods rely on sampling to determine the terrain and to decide where to search. Such strategies may fail when the terrain is very rugged or when the search gets trapped in a deep but suboptimal valley. This happens in clustering methods, whose performance is similar to that of random restarts when the terrain is rugged. Bayesian methods, on the other hand, do not work well because most of the samples they collect randomly from the error surface are close to the average error value, and these samples are inadequate to model the behavior at minimal points. Further, they are very expensive computationally and are usually not applicable for problems with over twenty variables.

Up to today, general nonlinear (global or local) minimization algorithms can at best find good local minima of a multi-modal function. Only in cases with very restrictive assumptions, such as Lipschitz condition, algorithms with guaranteed accuracy can be constructed.

In the next section, we propose a new global minimization method, called the *NOVEL method*, and its application to neural network learning. The method is unique because it has a deterministic mechanism to bring a search out of a local minimum.

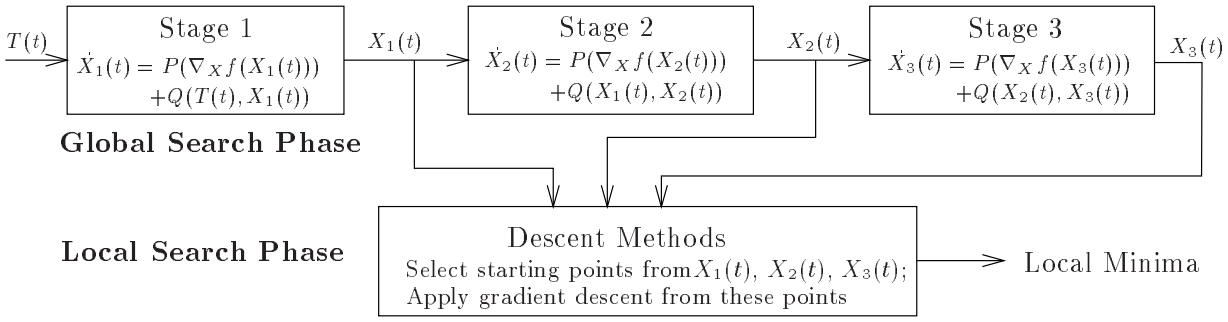

Figure 3: Framework of the *NOVEL* method. (See Section 3.2 for an explanation of the equations.)

3 NOVEL: A Novel Global Optimization Method

In this section, we describe the *NOVEL method*, a hybrid global- and local-search method. Our method is a trajectory-based method that relies on an *external* force to pull the search out of a local minimum, and employs local descents to locate local minima. It has three features: exploring the solution space, locating promising regions, and finding local minima. In exploring the solution space, the search is guided by a continuous terrain-independent *trace* that does not get trapped in local minima. In locating promising regions, *NOVEL* uses local gradient to attract the search to a local minimum but relies on the trace to pull it out of the local minimum once little improvement can be found. Finally, *NOVEL* selects one initial point for each promising local region and uses them as initial points for a descent algorithm to find local minima.

NOVEL is efficient in the sense that it tries to first identify good starting points before applying a local search. This avoids repeatedly determining unpromising local minima as in multi-start algorithms, and avoids computationally expensive descent algorithms from random starting points.

3.1 Framework of the NOVEL method

NOVEL has two phases: global-search phase and local-search phase (see Figure 3). The goal of the global-search phase is to identify regions containing local minima, whereas the goal of the local-search phase is to actually find the local minima.

In the global-search phase, there are a number of bootstrapping stages. (Three stages are shown in Figure 3.) The dynamics in each stage is represented by an ordinary differential equation. A stage is coupled to the next stage by feeding its output trajectory as the trace function of the next stage, with a user-supplied trace function as the input trace function of the first stage. Interpolations are performed when the input trace supplied by the previous stage is not a continuous function.

In general, the equations in each stage of the global-search phase can be different. In earlier stages, more weight can be placed on the trace function, allowing the resulting trajectory to explore more regions. In later stages, more weight can be placed on local descents, allowing the trajectory to descend deeper into local basins. Note that all the equations in the global-search phase can be combined into a single equation before being solved. We did not do so because each trajectory may identify new starting points that lead to better local minima. We present more details of the global-search phase in the next subsection.

In the local-search phase, a traditional descent method, such as gradient descent, conjugate gradient or Quasi-Newton's method, is applied to find local minima. Initial points for the local search are selected based on trajectories output by the global-search phase. Two heuristics can be applied in selecting initial points: use the best solutions in periodic time intervals as initial points, or use the local minima in the trajectory in each stage as initial points. In our experiments, we have used the first alternative, as the error terrain in neural network learning is very rugged and the second alternative will result in too many initial points.

Let us illustrate the global-search stage of *NOVEL* using a simple example based on Levy's No. 3

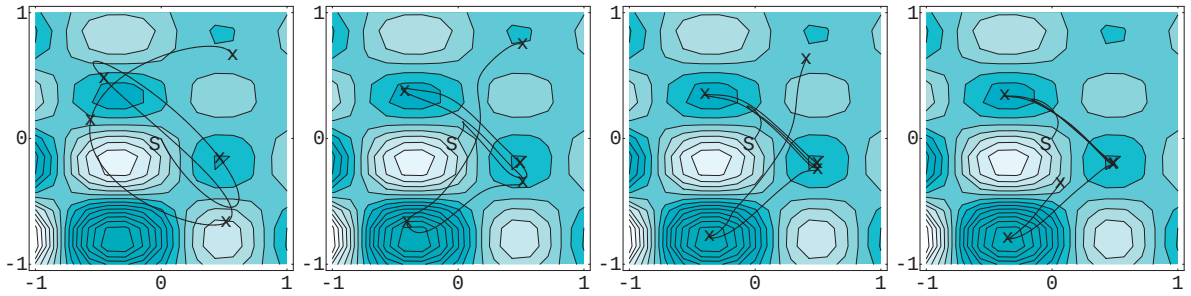

Figure 4: 2-D contour plots of Levy's No. 3 function with superimposed search trajectories: trace function starting from S (first graph), trajectory after Stage 1 (second graph), trajectory after Stage 2 (third graph), and trajectory after Stage 3 (fourth graph). A darker color represents a smaller function value. The trajectories are solved by *LSODE*, a differential-equation solver.

problem [5], which involves finding the global minimum of a function of two variables x_1 and x_2:

$$f_{l_3}(x) = \sum_{i=1}^{5} i\, cos[(i-1)x_1 + i] \sum_{j=1}^{5} j\, cos[(j+1)x_2 + j] \tag{2}$$

Figure 4 shows the 2-D contour plots of this function and the search trajectories of *NOVEL*. In the range shown, the function has three local minima, one of which is the global minimum. Using a search range of [-1, 1] in each dimension, we start *NOVEL* from (0, 0) and run it until logical time $t = 5$. Although the trace function visits all three basins, it only touches the basin with the global minimum. The trajectories are pulled closer to the local basins after Stages 1, 2 and 3, respectively. By following the trajectories, three basins with local minima are identified, and a set of minimal points in each trajectory can be used as initial points in the local-search phase.

3.2 Major components in the global-search phase

Assume $f(X)$ with gradient $\nabla_X f(X)$ is to be minimized, where $X = (x_1, x_2, \cdots, x_n)$ are variables. There may be simple bounds like $x_i \in [a_i, b_i]$, where a_i, b_i, $i = 1, \cdots, n$, are real numbers.

Each stage in the global-search phase of *NOVEL* defines a *trajectory* $X(t) = (x_1(t), \cdots, x_n(t))$ that is governed by the following ordinary differential equation:

$$\dot{X}(t) = P(\nabla_X f(X(t))) + Q(T(t), X(t)) \tag{3}$$

where t is the autonomous variable; T, the trace function, is a function of t; and P and Q are general nonlinear functions. This equation specifies a trajectory through variable space X. It has two components, $P(\nabla_X f(X))$ that enables the gradient to attract the trajectory to a local minimum, and $Q(T, X)$ that allows the trace function to lead the trajectory out of the local minimum.

P and Q can have various forms. A simple form we have used in our experiments is a constant function.

$$\dot{X}(t) = -\mu_g \nabla_X f(X(t)) - \mu_t (X(t) - T(t)) \tag{4}$$

where μ_g and μ_t are constant coefficients.

To find the global minima efficiently without any knowledge on the terrain, we should design a trace function that traverses the search space uniformly. There are two alternatives in traversing the space: (a) divide the space into subspaces and search one subspace extensively before another; and (b) search the space from coarse to fine. We have chosen the second approach because the number of dimensions is usually too large for the first approach to be practical. Using the second approach and after substantial experimentation,

Benchmark problems studied in our experiments *(obtained from ftp.cs.cmu.edu)*

Two-spiral problem – Discriminate between two sets of training points that lie on two distinct spirals in the x-y plane. Each spiral has 94 input-output pairs in both the training and test sets.
Sonar problem — Discriminate between sonar signals bounced off a metallic cylinder and those bounced off a roughly cylindrical rock. We used the training and test samples in "aspect angle-dependent" experiments.
Vowel recognition problem — Train a network to have speaker-independent recognition of the eleven steady-state vowels of British English. Vowels are classified correctly when the distance of the correct output to the actual output is the smallest among the distances from the actual output to all possible target outputs.
10-parity problem — Train a network that computes the modulo-two sum of ten binary digits. There are 1,024 training patterns and no test patterns.
NetTalk problem — Train a network to produce proper phonemes, given a string of letters as input. NetTalk data set contains 20,008 English words. We have used the same network settings and unary encoding as in Sejnowski and Rosenberg's experiments [7], 1,000 most common English words as the training set, the entire data set as the test set, and the "best-guess" criterion.

we have designed a non-periodic, analytical trace function as follows:

$$T_i(t) = \rho sin\left[2\pi\left(\frac{t}{2}\right)^{1-(0.05+0.45(i-1)/n)} + \frac{2\pi(i-1)}{n}\right] \tag{5}$$

where i represents the i'th dimension, ρ is a coefficient specifying the range, and n is the number of dimensions.

Given (4), various numerical approaches can be applied to evaluate the ordinary differential equation. We have used both a differential-equation solver and a difference-equation solver.

A differential-equation solver solves (4) as an ordinary differential equation. The software package we have used is the Livermore Solver for Ordinary Differential Equations [6] (*LSODE*) that solves (4) to within a prescribed degree of accuracy. However, it is usually computationally expensive, especially when the number of weights is large. Further, it requires the true gradient, meaning that neural-network learning can only be done in an epoch-wise mode, not in a pattern-wise mode.

The second approach is to discretize (4) and use a finite-difference equation solver. The difference equation derived from (4) is as follows.

$$X(t+\delta t) = X(t) + \delta t[-\mu_g \nabla_X f(X(t)) - \mu_t(X(t) - T(t))] \tag{6}$$

where δt is the step size. A large δt causes a large stride of variable modification, possibly resulting in oscillations. On the other hand, a small δt means a longer computation time for traversing the same distance. This approach is fast, and allows learning in both pattern-wise and epoch-wise mode. However solutions may be slightly worse as compared to those found by LSODE.

In the next two sections, we present experimental results in applying *NOVEL* to solve some neural-network benchmark problems. In general, *NOVEL* is able to find better results as compared to other global minimization algorithms in the same amount of time.

4 Two-Spiral Problem

In this section, we compare the performance of *NOVEL* with that of other good methods for global minimization. We then describe how to speed up *NOVEL* using a difference-equation solver, and show trade-offs between solution quality and computation speed.

The two-spiral problem is a difficult classification problem. Published results include training feed-forward networks using *BP*, *CASCOR* [8], and projection pursuit learning [9]. The smallest network is believed to have nine hidden units with 75 weights trained by *CASCOR*.

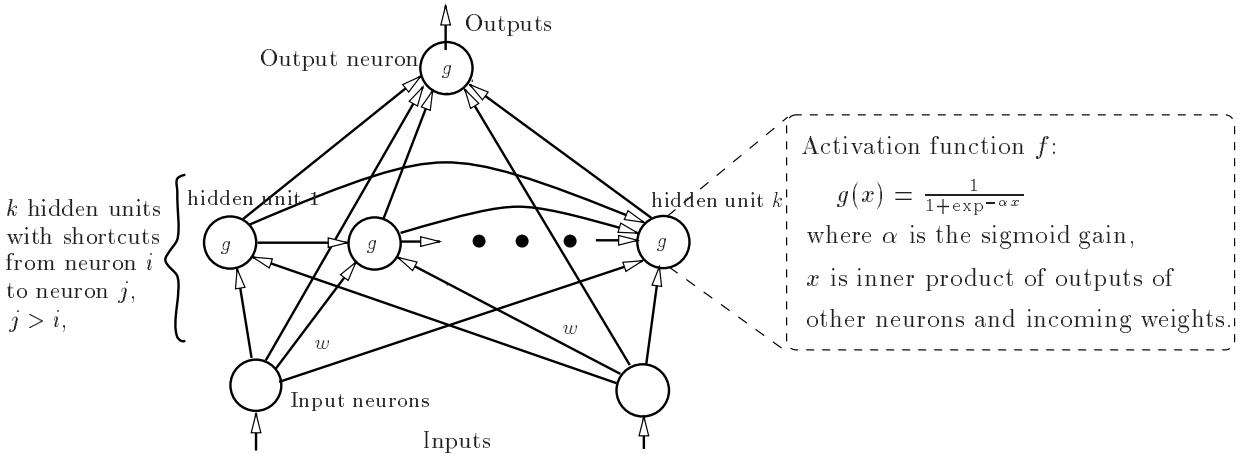

Figure 5: Neural-network structure for the two-spiral problem.

In our experiments, we have used feed-forward networks with shortcuts (see Figure 5.) Each hidden unit is ordered and labeled by an index, and has incoming connections from all input nodes and from all hidden units with smaller indexes. The activation function is an asymmetric sigmoidal function $f(x) = 1/(1+e^{-\alpha x})$, where α is the sigmoid gain. We have fixed the search range as [-1,1] in each dimension, and have varied α from 1 to 150. The error function $E(w)$ defined in (1) is the total sum of squared error (TSSE). All our experiments were carried out on Sun SparcStation 20 model 71 (75 MHz) workstations.

In applying *NOVEL* with the differential-equation solver LSODE, we always started our trace from the origin of the weight space. This eliminates any bias in choosing "good" starting points in the search. *NOVEL* generates trajectories that are function of the autonomous variable t, which we call *logical time*, and one *time unit* represents a change from $t = \tau$ to $t = \tau + 1$. In our experiments, we executed all three stages in the global-search phase in each time unit.

After trying various combinations of α, μ_g and μ_t for 4 and 5 hidden-unit networks, we found that the combination of $\mu_g = 1$, $\mu_t = 20$ and $\alpha = 100$ works well. This set of parameters were used in our experiments.

NOVEL successfully trained five hidden-unit networks in less than 100 time units. Training four hidden-unit networks is more difficult. After running *NOVEL* for 800 time units, which was 77.48 hours of CPU time on a SparcStation 20/71, we found a solution with TSSE of 2.1 and 99% correct. Using this solution as a new starting point, we executed *NOVEL* for another 89.44 hours and found in a solution that is 100% correct. The second figure in the first row of Figure 6 shows how the best four hidden-unit network found classifies the 2-D space.

Next, we compare the performance of *NOVEL* with that of simulated annealing, evolutionary algorithms, cascade correlation with multi-starts (*CASCOR-MS*), gradient descent with multi-starts (*GRAD-MS*), and truncated Newton's method with multi-starts (*TN-MS*). (See side-bar for explanation.) To allow a fair comparison, we ran all these methods for the same amount of time using the same network structure.

The simulated annealing program used in our experiments is *SIMANN* from netlib [10]. We experimented with various temperature scheduling factors RT, function evaluation factors NT, and search ranges. The best results were achieved when $RT = 0.99$, $NT = 5n$, and the search range is [-2.0, 2.0].

We have also studied two evolutionary algorithms (EAs): GENOCOP (GEnetic algorithm for Numerical Optimization for COnstrained Problems) by Michalewicz [11] and LICE (LInear Cellular Evolution) by Sprave [12]. GENOCOP aims at finding a global minimum of an objective function under linear constraints. We have tried various search ranges and population sizes. Search range [-0.5, 0.5] and population size $100n$ give the best results. LICE is a parameter optimization program based on evolutionary strategies. In applying LICE, we have tried various initial search ranges and population sizes. Range [-0.1, 0.1] and population size $100n$ give the best results.

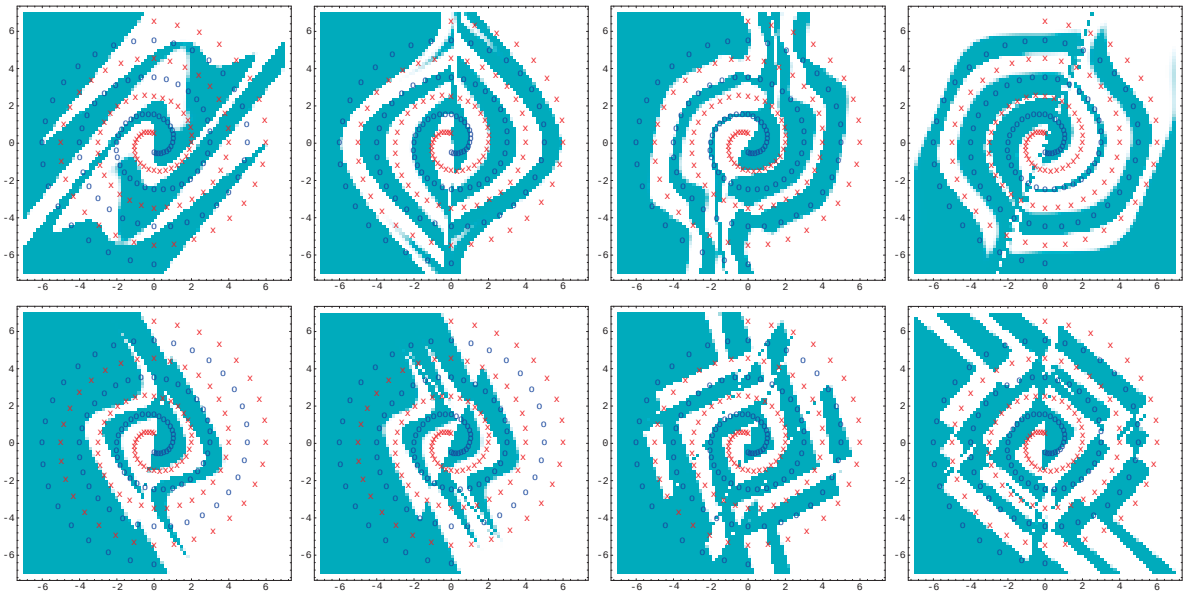

Figure 6: 2-D classification graphs for the two-spiral problem by 3 (first column), 4 (second column), 5 (third column) and 6 (fourth column) hidden-unit neural networks trained by *NOVEL* (upper row) and *SIMANN* (lower row). Parameters for *NOVEL* are $\mu_g = 1, \mu_t = 20$, and $\alpha = 100$. Parameters for *SIMANN* are $RT = 0.99, NT = 5n$, and the search range is [-2.0, 2.0]. The crosses and circles represent the training patterns.

In applying *CASCOR-MS*, we ran Fahlman's *CASCOR* program [8] from random initial weights. We started from a new starting point when the current run did not result in a converged network for a maximum of 3, 4, 5 and 6 hidden units, respectively.

In *GRAD-MS*, we generated multiple random initial points in the range [-0.2, 0.2]. Gradient descents were done using *LSODE*.

Finally, we have used truncated Newton's method obtained from netlib with multi-starts (*TN-MS*). We generated random initial points in the range [-1, 1], and set the sigmoid gain to 1. Since one run of *TN-MS* is very fast, a large number of runs were done within the time limit.

The best performance of these algorithms is shown in Figure 7, each showing the progress of one run of a learning algorithm. Figure 8 summarizes the training and test results of the best solutions found by each of these algorithms when run under 20 hours of CPU time on a Sun SparcStation 20/71. The graphs show that *NOVEL* has the best training and test results for the neural networks found, followed by *SIMANN*, *TN-MS*, *CASCOR-MS*, and the two evolutionary algorithms. Figure 6 shows the best solutions obtained by *NOVEL* and *SIMANN*.

The experimental results show that a learning algorithm's performance depends on the complexity of the error function. When the error function is complex and good solutions are few, *NOVEL* performs much better than other algorithms.

The differential-equation solver is computationally expensive. To improve the computational speed, we have used a difference-equation solver instead of *LSODE*. In using a difference-equation solver, we have tried four pairs of coefficients: $\mu_g = 0.001$ and $\mu_t = 0.01$; $\mu_g = 0.001$ and $\mu_t = 0.1$; $\mu_g = 0.01$ and $\mu_t = 0.1$; and $\mu_g = 0.01$ and $\mu_t = 0.1$. Further, we have tried the following sigmoid gains α: 1, 10, 30, 50 and 100. Table 1 presents the combination of parameters leading to the best results of *NOVEL* with epoch-wise training. The results show that the difference-equation solver is about ten times faster than *LSODE*. however, the solution quality is slightly worse.

Good global minimization methods for neural network learning

Simulated annealing (*SA*) — *SA* is a stochastic global minimization method. Starting from an initial point, the algorithm takes a step and evaluates the error function once. When minimizing a function, any down-hill movement is accepted, and the process repeats from this new starting point. An uphill movement may be accepted, and by doing so, the search can escape from local minima. This uphill decision is made by the Metropolis criteria. As the minimization process proceeds, the length of steps decreases and the probability of accepting uphill movements decreases as well. The search converges to a local (sometimes global) minimum at the end.

Evolutionary algorithm (*EA*) — *EA* is based on the computational model of evolution. A variety of EAs have been proposed in the past, among which include genetic algorithms, evolutionary programming, and evolutionary strategies. EAs maintain a population of individual points in the search space, and the performance of the population evolves to be better through selection, recombination, mutation and reproduction. The fittest individual has the largest probability of survival. EAs have been applied to complex, multi-modal minimization problems with both discrete and continuous variables.

Cascade correlation with multi-starts (*CASCOR-MS*) — Cascade correlation learning algorithm is a constructive method that starts from a small network, and gradually builds a larger network to solve the problem. This algorithm was originally proposed by Fahlman and Lebiere [8] and has been applied successfully to some neural-network learning problems. In *CASCOR-MS*, multiple runs of *CASCOR* were executed from randomly selected initial points.

Gradient descent with multi-starts (*GRAD-MS*) — Gradient-descent algorithms are simple and popular, and their variants have been applied in many engineering applications. An example is the back-propagation learning algorithm. For solving the two-spiral problem using *NOVEL*, gradient descents were performed by solving an ordinary differential equation using *LSODE*.

Truncated Newton's method with multi-starts (*TN-MS*) — Truncated Newton's method uses second-order information that may help convergence. They are usually much faster than *LSODE*.

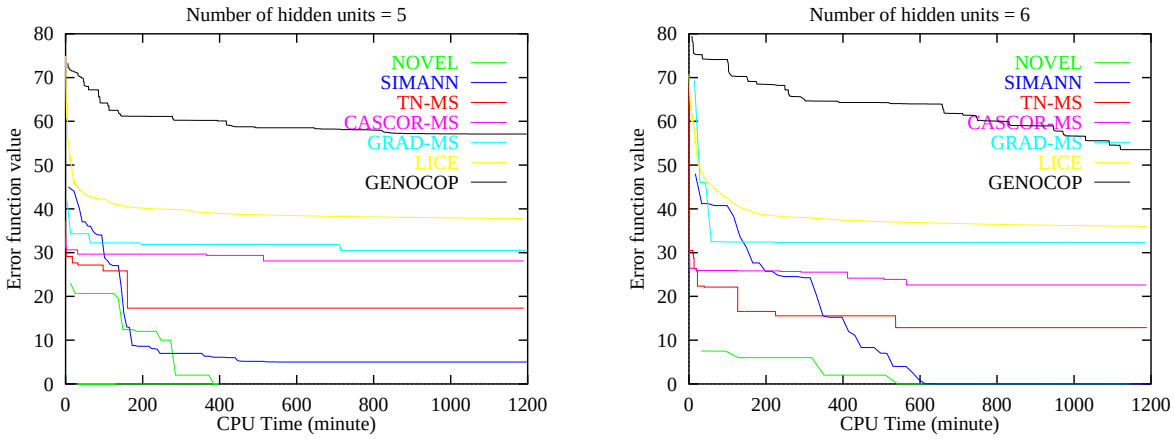

Figure 7: The best performance of one run of various global minimization algorithms for learning the weights of neural networks with 5 and 6 hidden units for solving the two-spiral problem. (Sigmoid gain $\alpha = 100$ for all algorithms except *CASCOR-MS* and *TN-MS*, which has $\alpha = 1$. CPU time allowed for each experiment is 20 hours on Sun 20/71.)

5 Experimental Results on Other Benchmarks

In this section, we show our results in applying *NOVEL* on four benchmark problems described in the last section (sonar, vowel-recognition, 10-parity, and NetTalk problems). All these benchmarks were obtained from ftp.cs.cmu.edu in directory /afs/cs/project/connect/bench.

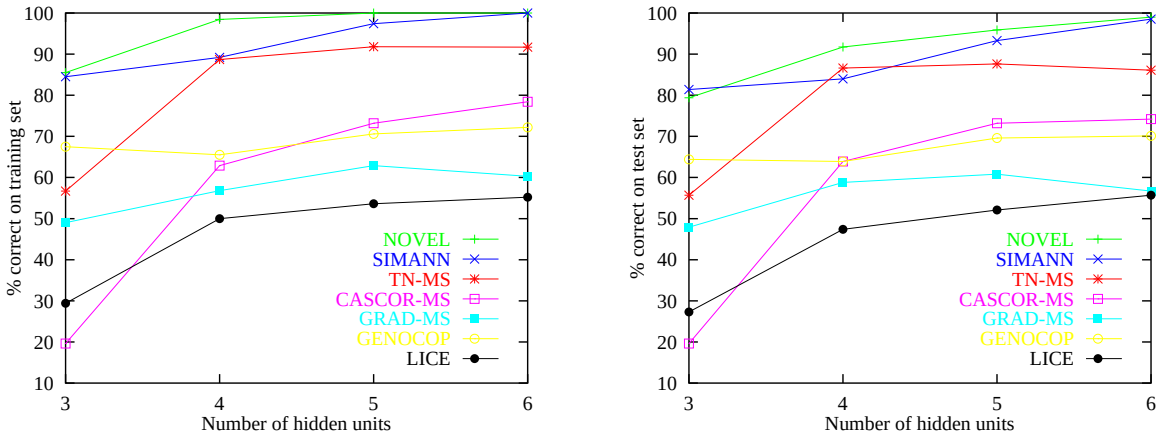

Figure 8: Training and test errors of the best designs obtained by various algorithms for solving the two-spiral problem. There are 18, 25, 33, and 42 weights (including biases in neurons) in the neural network for networks with, respectively, 3, 4, 5, and 6 hidden units.

Table 1: Summary results of *NOVEL* with a finite difference-equation solver for solving the two-spiral problem. The total number of time units in each run is 400.

Number of hidden units	Number of weights	Sigmoid gain α	Coefficients		Best Solution			CPU time per time unit (minutes)
					training		testing	
			μ_g	μ_t	TSSE	Correct %	Correct %	
4	25	50	0.01	0.1	14.0	92.8	85.6	0.20
5	33	50	0.001	0.01	6.0	96.9	94.8	0.36
6	42	10	0.01	0.1	0.0	100	95.4	0.55

The network topologies used in these experiments are layered feed-forward networks without shortcuts (to be consistent with what others have used), and the goal is to minimize the total sum of squared errors. Other setups are similar to those described for the two-spiral problem.

For the sonar problem, we have applied *NOVEL* with a difference-equation solver, *TN-MS*, *SIMANN*, and *BP*. As found by Dixon [4], *TN* runs much faster than epoch-wise *BP* and achieves comparable solutions. *SIMANN* is one order of magnitude slower than *TN-MS* and *NOVEL* with a difference-equation solver, and the results are not better. For these reasons, we describe only the results for *TN-MS* and *NOVEL* using a difference-equation solver, where *TN* is used in the local-search phase of *NOVEL*.

Table 2 [2] shows the best solutions of both algorithms that achieve the highest percentage of being correct on test patterns of the sonar problem. Our results show that *NOVEL* has found solutions with 1%-4% better test accuracy.

The reason why *NOVEL* can find better local minima is attributed to its global-search stage. Since the function searched is very rugged, it is important to identify good basins before committing expensive local descents into them. In contrast, the number of restarts that a multi-start algorithm can have is relatively small, given the limited time that the algorithm can run. However, multi-start algorithms may provide good starting points for *NOVEL*, although this is not the case for the sonar problem.

[2] All the results in Table 2 were run under similar conditions and time limits. In particular, *NOVEL* always started from the origin and searched in the range $[-1, 1]$ for each variable, using some combinations of sigmoid gains from the set {1, 10, 30, 50, 100, 300} and (μ_g, μ_t) from the set {(10, 1), (1, 1), (1, 0.1), (0.1, 0.1), (0.1, 1), (0.1, 0.01), (0.01, 0.1)}. *TN-MS* was run using different combinations of random initial points in the same search ranges and the same sigmoid gains as in *NOVEL*. In *TN-MS+NOVEL*, *NOVEL* always started from the best result of *TN-MS* using the same sigmoid gain when *TN-MS* was run. In solving the NetTalk problem, *NOVEL* used a learning rate of 2, a momentum of 0.1, and a search range of [-1,1]. *BP* generated its initial point in the range [-0.5,0.5].

Table 2: Comparison of the best results obtained by *NOVEL* and truncated Newton's algorithm with multi-starts (*TN-MS*) for solving four benchmark problems, where the parameters in one method that obtains the best result may be different from those of another method. Results in bold font are better than or equal to results obtained by *TN-MS*.

Problems	# of H.U.	# of Wts.	*TN-MS* Correct % training	*TN-MS* Correct % test	*TN-MS* # of restarts	*NOVEL* Correct % training	*NOVEL* Correct % test	*NOVEL* # time units	*TN-MS + NOVEL* Correct % training	*TN-MS + NOVEL* Correct % test	*TN-MS + NOVEL* # time units	CPU time limits
Sonar	2	125	98.1	90.4	454	**98.1**	**94.2**	191	**98.1**	**92.3**	226	1000 sec
	3	187	100	91.3	485	**100**	**92.3**	291	**100**	**92.3**	315	2000 sec
Vowel	2	55	72.2	50.9	298	**72.5**	49.1	131	**73.5**	50.6	203	2 hours
	4	99	80.7	56.5	152	**82.6**	**57.8**	41	**81.2**	**57.1**	168	2 hours
10-parity	5	61	97.2	—	148	**98.9**	—	51	**97.2**	—	49	2000 sec
	6	73	97.6	—	108	**99.8**	—	62	**97.6**	—	44	3000 sec
NetTalk			*Pattern-wise BP*			*NOVEL*			*Pattern-wise BP + NOVEL*			
	15	3,476	86.3	70.5	13	**87.4**	**72.7**	11	**89.0**	70.4	11	3 hours
	30	6,926	92.9	73.1	9	**93.2**	72.5	4	**94.7**	72.3	7	4 hours

On the vowel-recognition problem, Table 2 shows that *NOVEL* improves in training as compared to *TN-MS*, but performs slightly worse in testing when there are two hidden units. *TN-MS+NOVEL* obtained designs that are slightly worse than those by *NOVEL* but better than those by *TN-MS*.

The next application is based on the 10-parity problem. Using a similar setup as described for the sonar problem, *NOVEL* was able to improve the learning results as compared to *TN-MS*.

In the last application, we have studied the NetTalk problem. Since the number of weights and training patterns are very large, we have used pattern-wise learning when applying *BP* (as in the original experiments by Sejnowski and Rosenberg [7]). By experimenting with different parameter settings of *BP*, we have found the best learning result by *BP* is 86.3% for a 15 hidden-unit network.

In applying *NOVEL* to solve the NetTalk problem, the number of weights is too large for us to use any method other than pattern-wise mode in the global-search phase and pattern-wise *BP* in the local-search phase. Even so, very few (logical) time units could be simulated, and our designs perform slightly better in learning but worse in testing when the number of hidden units is 15. To find better designs, we took the best designs obtained by pattern-wise *BP* and applied *NOVEL* for 11 and 7 time units, respectively. Table 2 shows slightly improved learning results but worse testing results. The worse testing results are probably due to the small number of time units that *NOVEL* was run.

In short, our experimental results by *NOVEL* in learning are always better than or equal to those by *TN-MS* but occasionally may be slightly worse in testing. This is attributed to the reasons that we did not have enough time to run *NOVEL*, and that the solutions found by existing methods are already very good. In general, improvements of solutions that are close to the global minima are very hard, often requiring exponential amount of time unless a better search method is used.

6 Conclusions

In this paper, we have applied various global minimization methods to supervised learning of feed-forward neural networks. The learning of weights in such networks can be treated as a nonlinear continuous minimization problem with rugged terrains. Our goal is to find neural networks with small number of weights using the same amount of time as in other algorithms, while avoiding the overfitting of weights in large networks. Our reasoning is that there are many good local minima in the error space of large networks, hence increasing the chance to find a good local minimum that does not generalize well to test patterns.

We have identified two crucial features of suitable algorithms for solving these problems. (a) Use gradient information to descend into local minima. Many algorithms have difficulties when the surface is flat, or when gradients can vary in a large range, or when the terrain is rugged. (b) Escape from a local minimum once the search gets there. Such mechanisms can be classified into probabilistic and deterministic. The suitability of a specific strategy is usually problem dependent.

Different algorithms performs different trade-offs between local search and global search. Algorithms that focus on either extreme do not work well. These include gradient descent with multi-starts (such as back-propagation and cascade correlation that focus too much on local search) and covering methods (that focus too much on global search). A good algorithm generally combines global and local searches, switching from one to another dynamically depending on run-time information obtained. These include algorithms based on simulated annealing, evolution, and clustering.

We have examined the pros and cons of various algorithms and have proposed *NOVEL*, a novel global minimization method for general nonlinear optimization. *NOVEL* has a global-search phase that relies on two counteracting forces: local gradient information that drives the search to a local minimum, and a deterministic trace that leads the search out of a local minimum once it gets there. The result is an efficient method that identifies good basins without spending a lot of time in them. Good starting points identified in the global-search phase are used in the local-search phase in which pure gradient descents are applied.

We have implemented *NOVEL* using a differential-equation solver as well as a difference-equation solver, and have shown improved performance for five neural-network benchmark problems. For the two-spiral problem, we have shown a design with near-perfect classification using four hidden units and 25 weights. (The best design known today requires nine hidden units and 75 weights.)

Although we have demonstrated the power of *NOVEL* in solving some neural-network benchmarks, we still need further study on the applicability of *NOVEL* to other benchmarks as well as to general non-linear optimization problems. In particular, we need to study new trace functions that cover the search space from coarse to fine, their search range, the combination of *NOVEL* with other local/global search methods, the relative weights between local descent and affinity to the traveling trace, parallel processing of *NOVEL*, and applying *NOVEL* to solve other application problems.

In short, *NOVEL* represents a significant advance in the state-of-the-art in supervised learning of feed-forward neural networks and optimization of general high-dimensional nonlinear continuous functions.

References

[1] A. Torn and A. Zilinskas, *Global Optimization*. Springer-Verlag, 1987.

[2] D. G. Luenberger, *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1984.

[3] R. Battiti, "First- and second-order methods for learning: Between steepest descent and Newton's method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.

[4] L. C. W. Dixon, "Neural networks and unconstrained optimization," in *Algorithms for Continuous Optimization: The State of the Art* (E. Spedicato, ed.), pp. 513–530, Netherlands: Kluwer Academic Publishers, 1994.

[5] A. V. Levy, A. Montalvo, S. Gomez, and A. Calderon, *Topics in Global Optimization, Lecture Notes in Mathematics No. 909*. New York: Springer-Verlag, 1981.

[6] A. C. Hindmarsh, "ODEPACK, a systematized collection of ODE solvers," in *Scientific Computing* (R. S. Stepleman, ed.), pp. 55–64, Amsterdam: North Holland, 1983.

[7] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, vol. 1, pp. 145–168, Feb. 1987.

[8] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 524–532, San Mateo, CA: Morgan Kaufmann, 1990.

[9] J. Hwang, S. Lay, M. Maechler, R. D. Martin, and J. Schimert, "Regression modeling in back-propagation and projection pursuit learning," *IEEE Transactions on Neural Networks*, vol. 5, pp. 1–24, May 1994.

[10] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multi-modal functions of continuous variables with the simulated annealing algorithm," *ACM Transactions on Mathematical Software*, vol. 13, no. 3, pp. 262–280, 1987.

[11] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, 1994.

[12] J. Sprave, "Linear neighborhood evolution stategies," in *Proc. of the third Annual Conf. on Evolutionary Programming*, (San Diego, CA), World Scientific, 1994.