

Cooperative - Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction

Bruce A. Whitehead, *Member, IEEE*, and Timothy D. Choate

Abstract— In a radial basis function (RBF) network, the RBF centers and widths can be evolved by a cooperative-competitive genetic algorithm. The set of genetic strings in one generation of the algorithm represents one RBF network, not a population of competing networks. This leads to moderate computation times for the algorithm as a whole. Selection operates on individual RBFs rather than on whole networks. Selection therefore requires a genetic fitness function that promotes competition among RBFs which are doing nearly the same job, while at the same time promoting cooperation among RBFs which cover different parts of the domain of the function to be approximated. Niche creation resulting from a fitness function of the form $|w_i|^\beta / E(|w_i|^\beta)$, $1 < \beta < 2$ can facilitate the desired cooperative-competitive behavior. The feasibility of the resulting algorithm to evolve networks of Gaussian, inverse multiquadric, and thin-plate spline RBFs is demonstrated by predicting the Mackey-Glass time series. For each type of RBF, and for networks of 25, 50, 75, 100, 125, and 150 RBF units, prediction errors for the evolved Gaussian RBF networks are 50-70% lower than RBF networks obtained by k -means clustering.

Keywords— Cooperative-competitive, Genetic algorithms, Radial basis functions, Evolutionary computation, Nicheing, Time-series prediction

I. INTRODUCTION

A radial basis function (RBF) network [1], [2], [3] used to approximate an unknown function $f : \mathfrak{R}^n \rightarrow \mathfrak{R}$ can be described by the affine mapping

$$f(\mathbf{x}) \approx w_0 + \sum_{i=1}^m w_i \phi_i(\mathbf{x}) \quad (1)$$

in which the m radially-symmetric basis functions ϕ_i are often taken to be translated dilations of a prototype radial basis function $\phi : \mathfrak{R} \rightarrow \mathfrak{R}$, i.e. $\phi_i(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}_i\|/d_i)$, where $\mathbf{c}_i \in \mathfrak{R}^n$ is the center of basis function ϕ_i , $d_i \in \mathfrak{R}$ is a dilation or scaling factor for the radius $\|\mathbf{x} - \mathbf{c}_i\|$, and $\|\cdot\|$ is typically the Euclidean norm on \mathfrak{R}^n . Choices of ϕ considered in theoretical investigations [1], [3], [4], [5], [6], [7] and practical applications [5], [7], [8], [9], [10] include $\phi(r) = r$ (linear), $\phi(r) = r^3$ (cubic), $\phi(r) = r^2 \log r$ (thin plate spline), $\phi(r) = e^{-r^2/2}$ (Gaussian), $\phi(r) = \sqrt{r^2 + 1}$ (multiquadric), and $\phi(r) = 1/\sqrt{r^2 + 1}$ (inverse multiquadric), where in all cases r is the scaled radius $\|\mathbf{x} - \mathbf{c}_i\|/d_i$.

Radial basis functions were originally proposed as an interpolation method, and their properties as interpolants

have been extensively studied [3]. In this context, if the value of f is known at p data points $\mathbf{x}_1, \dots, \mathbf{x}_p$ in \mathfrak{R}^n , then each basis function ϕ_i is centered on one of these data points, so that there are as many centers \mathbf{c}_i as data points: $m = p$. In the context of neural networks, on the other hand, it is commonly assumed that there are significantly fewer basis functions than data points. The central problem then becomes the placement of the centers \mathbf{c}_i and determination of the radial dilation factors (usually called *widths*) d_i to achieve the best prediction and generalization performance [11]. This problem is most often approached by clustering the data points. The p data points are clustered into m clusters, and the centers of these clusters are then used as the RBF centers \mathbf{c}_i [12]. Clustering is typically performed by a vector quantization algorithm [13], which iteratively minimizes some measure of distortion such as the mean squared distance from each data point to the center of the cluster to which it belongs.

Once the centers and widths of the basis functions are determined, each weight w_i used in the approximation of equation (1) may be determined either by direct numerical least-squares methods such as singular value decomposition, or by iterative methods such as the LMS algorithm [14].

In this paper, we consider a novel genetic approach to optimizing the center locations \mathbf{c}_i and widths d_i when $m < p$. This genetic approach operates on a population of competing basis functions ϕ_i . The entire population corresponds to a single RBF network. This means that individual RBFs in the population, although genetically competing with each other, must evolve to *cooperatively* model the function f over the entire domain of interest. While cooperative-competitive algorithms are well known in neural networks [15], previous genetic approaches to neural network optimization have emphasized purely competitive genetic algorithms (GAs) [16-29].

The remainder of this paper is organized as follows: Section II reviews various approaches to genetic optimization of neural networks, fitting our approach into this framework. Section III considers how a genetic fitness function can be chosen to facilitate cooperative-competitive evolution. Section IV provides further details of our model. Section V then explains the benchmark problems used to show the model's feasibility, and Section VI compares the results with those produced by k -means clustering.

Bruce A. Whitehead is with the University of Tennessee Space Institute, Tullahoma, TN 37388, USA. E-mail: bwhitehe@utsi.edu

Timothy D. Choate is with Edge Internet Services, 530 Third Avenue South, Suite 3, Nashville, TN 37210, USA. E-mail: tchoate@edge.net

II. EVOLUTIONARY OPTIMIZATION OF NEURAL NETWORKS

Since genetic and other evolutionary algorithms for optimizing neural networks have been thoroughly reviewed elsewhere [30], [31], only the major trends in this research will be summarized here.

A. Competing Whole Networks

Let us first consider the main body of work in which the competing individual is a whole neural network. In this framework, each individual in the population specifies a separate neural network. Competition then occurs among these individual networks, based on the performance of each network. Within this framework, different lines of research can be categorized according to which parameters of the neural network are specified by the evolutionary algorithm.

One approach is to evolve only the structural specification of an untrained feedforward neural network, such as the number of layers and the distribution of connections between layers [16-29]. To evaluate such an evolved structural specification, it must be instantiated and the resulting network must then be trained and tested by conventional non-evolutionary neural algorithms. A contrasting approach is to evolve both the structure and weights of a network [32-46]. Here the evaluation of each evolved network is much simpler. Since the weights of each evolved network have already been adapted by the evolutionary algorithm, its evaluation does not require additional training by a non-evolutionary algorithm. The space of possible networks to be searched by the evolutionary algorithm becomes much larger, however, if every weight must be adapted by the evolutionary algorithm. A compromise approach is to evolve a specification which leaves only the final layer of weights unspecified. In this case the evaluation of each network requires training of one layer of weights only, which is typically a much lighter computation than a multilayer training algorithm. This compromise approach has been studied on feedforward networks using both sigmoidal [42] and radial [47] basis functions.

Within this framework of competition among separate neural networks, other differentiating factors include the basic alphabet of binary or real-valued representations which encode the specification of a neural network for genetic [48] or non-genetic [35] evolution.

B. Evolving One Hidden Unit at a Time

Cascade-correlation learning [49] (and similar methods such as projection pursuit learning [50]) employ non-traditional neural network architectures in which the hidden units are added sequentially. The cascade correlation architecture does not attempt to simultaneously optimize all hidden units. Instead, the placement of one hidden unit is optimized and frozen before beginning to optimize the placement of another hidden unit. (This architecture typically employs hidden units with sigmoidal activation functions rather than RBF units. The “placement” of such

a sigmoidal unit is determined by the vector of weights leading into it and can be visualized as a hyperplane perpendicular to this weight vector.)

Since cascade correlation optimizes one unit at a time, the gradient descent procedure typically used to optimize the unit’s placement can be replaced by a competitive genetic [51], [52] or evolutionary programming (EP) [42], [53] algorithm. Successive runs of the GA or EP optimize successive hidden units. The competing units are thus individual units rather than whole networks. During each successive run, candidate hidden units compete to evolve the placement of the single unit to be optimized by that run. In this framework of evolving one unit at a time, the space to be searched by each run of the GA or EP is much smaller. There is no guarantee, however, that optimizing one unit at a time will lead to a globally optimal placement of the entire set of hidden units.

III. EVOLVING COOPERATING AND COMPETING UNITS

In the present paper, we consider a framework different from the two major frameworks discussed in Sections II-A and II-B above. Like the first framework of evolving whole networks, we wish to simultaneously optimize the entire set of centers and widths which specify an RBF network. Instead of a population of competing networks, however, the entire population of our GA encodes only one network, as in the second framework. The desired framework is similar to that of [54] in which the hidden units reproduce or die based on their performance, in addition to being modified by back propagation learning. In our framework, however, an evolutionary algorithm is the sole means of modifying unit placement.

In the desired framework, each individual in the population specifies a different RBF belonging to the same network. The desired outcome is therefore not a single optimal RBF, but rather a population of coadapted RBFs which work together to yield a well-performing RBF network. To achieve this outcome using a GA, two problems must be solved. First is the well-known credit apportionment problem [48], [55], [56]: the quantity to be optimized is a performance measure of the whole population, but the GA requires a performance measure for each separate individual in the population. The second problem is known in the GA literature as a “niching problem” [57], [58]: different RBFs must evolve to do different parts of the overall job of approximating f .

Intuitively, we approach the credit apportionment problem by assigning credit to each RBF based on the contribution of that RBF to the overall prediction of the network. As in [56], credit is apportioned among the units within a network. Unlike [56], where a population of competing networks is necessary, our apportionment scheme depends only on competition within a single network. We approach the niching problem by varying the intensity of competition among RBFs based on the degree of overlap in the jobs they perform. Niche creation is *implicit* [59], [60] in the sense that there is no explicit calculation of a fitness sharing function. The remainder of this section further ex-

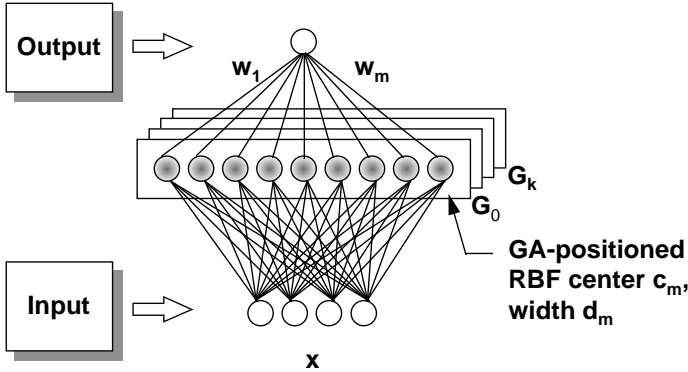


Fig. 1. Architecture of the cooperative-competitive genetic model. Each node in the middle layer represents a radial basis function (RBF) whose center \mathbf{c}_i and width d_i are encoded in generation G_k by a genetic string ϕ_i . The activation computed by each RBF depends on the distance between the input vector \mathbf{x} and the center of that RBF. Each generation G_k of the genetic algorithm produces a population of RBFs all belonging to the same neural network. The neural network produced in each generation is trained to optimize the set of weights w_i which compute the output of the network as a linear combination of RBF activations.

plains our approach to these two problems.

A. Cooperative - Competitive Genetic Selection

Figure 1 summarizes the architecture of the proposed cooperative-competitive approach. As in most GAs, the outer loop of the algorithm is the construction and evaluation of successive generations $G_0, G_1, \dots, G_k, \dots$. The state of the algorithm in each generation consists of a population of bit strings ϕ_i , $i = 1, \dots, m$, where m is the size of the population, and each bit string ϕ_i encodes the center \mathbf{c}_i and width d_i of one RBF ϕ_i of equation (1). If the scalar d_i and the coordinates of the vector $\mathbf{c}_i \in \mathbb{R}^n$ are each encoded with ℓ bits of precision, then one RBF ϕ_i can be encoded in a bit string ϕ_i containing $(n+1)\ell$ bits, i.e. $\phi_i = b_1 b_2 b_3 \dots b_n \dots b_L$, $b_n \in \{0, 1\}$, $L = (n+1)\ell$. To avoid proliferating notation, the symbol ϕ_i will represent both the bit string and the RBF unit it encodes, provided context makes the referent clear. The details of the binary encoding are deferred to section IV, since these details are relevant to the genetic operators of recombination, mutation, and creep (perturbation of a genetically-encoded scalar value), discussed in that section, but not relevant to the present discussion of RBF performance evaluation.

A fitness-based selection procedure is used to generate the population G_{k+1} from the population G_k . The algorithm employs proportionate selection in a fixed-sized population, meaning that each slot to be filled in G_{k+1} is drawn by sampling from G_k with replacement, and with each encoded RBF ϕ_i in G_k having a probability of selection proportional to its performance. Thus, the individual RBF performance measure (or RBF *fitness*, in GA terminology) will specify the expected number of copies of ϕ_i to occur in the next generation G_{k+1} before these copies are

altered by genetic operators. After filling the slots in G_{k+1} with these selected copies, genetic operators of recombination, creep, and mutation are applied. The performances of the RBFs in G_{k+1} are then evaluated in turn, yielding the sampling probabilities for forming G_{k+2} , and so on. Before discussing the details of selection, recombination, creep, and mutation, we consider the central problem of how to evaluate the performance of each individual RBF within a population G_k .

Let us therefore assume for the present that we have generated the population of encoded RBFs ϕ_i , $i = 1, \dots, m$ belonging to generation G_k of the algorithm, and discuss how this population is to be evaluated. This evaluation must support a competitive process, in which RBF units which are better predictors displace those which are worse predictors. The evaluation must at the same time support a cooperative process, in which different RBFs in the population evolve to cooperatively predict the value of the function f over the domain of f represented by the given set of training examples.

To consider the role of the fitness measure in the desired cooperative-competitive process, it will be convenient to normalize each RBF unit ϕ_i to have activations with a squared sum of 1 over the set of training examples presented to the RBF network in each generation G_k . The training set in generation G_k will consist of a set of pairs, $(\mathbf{x}_j, f(\mathbf{x}_j))$, $j = 1, \dots, p$, in which each $\mathbf{x}_j \in \mathbb{R}^n$ is a point in the domain of the function f to be approximated, and $f(\mathbf{x}_j)$ is the known value of the function at that point. Successive generations G_k might always operate on the same set of training examples, or they might operate on successive samples of some ongoing stochastic process. In either case, within a given generation, the normalized version $\hat{\phi}_i$ of each basis function ϕ_i can be defined as $\hat{\phi}_i(\mathbf{x}) = \phi_i(\mathbf{x}) / \sqrt{\sum_{j=1}^p \phi_i^2(\mathbf{x}_j)}$, and these normalized RBFs constitute the network to be trained in that generation.

If the weights w_i that form the affine combination $w_0 + \sum_{i=1}^m w_i \hat{\phi}_i(\mathbf{x})$ are adjusted to minimize the mean-squared error in approximating f , then the weight w_i assigned to each normalized RBF $\hat{\phi}_i$ would intuitively seem to offer some indication of the relative contribution of this RBF to the overall prediction of f . Two obvious candidates for measuring the performance (fitness) of each RBF are $|w_i|$ and w_i^2 . In our pilot experiments, however, GAs employing these fitness measures (whether the RBFs were normalized or not) did not evolve RBF networks comparable to those produced by conventional k -means clustering in terms of generalization performance.

As a guide in determining a more appropriate RBF fitness measure, consider the tradeoff between cooperation and competition in the population G_k in terms of the inner product $\hat{\phi}_i \cdot \hat{\phi}_i$, where the *normalized activation sequence* $\hat{\phi}_i$ is the vector in \mathbb{R}^p whose components are $(\hat{\phi}_i(\mathbf{x}_1), \hat{\phi}_i(\mathbf{x}_2), \dots, \hat{\phi}_i(\mathbf{x}_p))$ [7], [49]. Since each $\hat{\phi}_i$ is normalized, $\|\hat{\phi}_i\| = 1$. (In general, any symbol with a hat will denote a vector in \mathbb{R}^p which has one component per training example.) Figure 2 depicts the activation sequences of

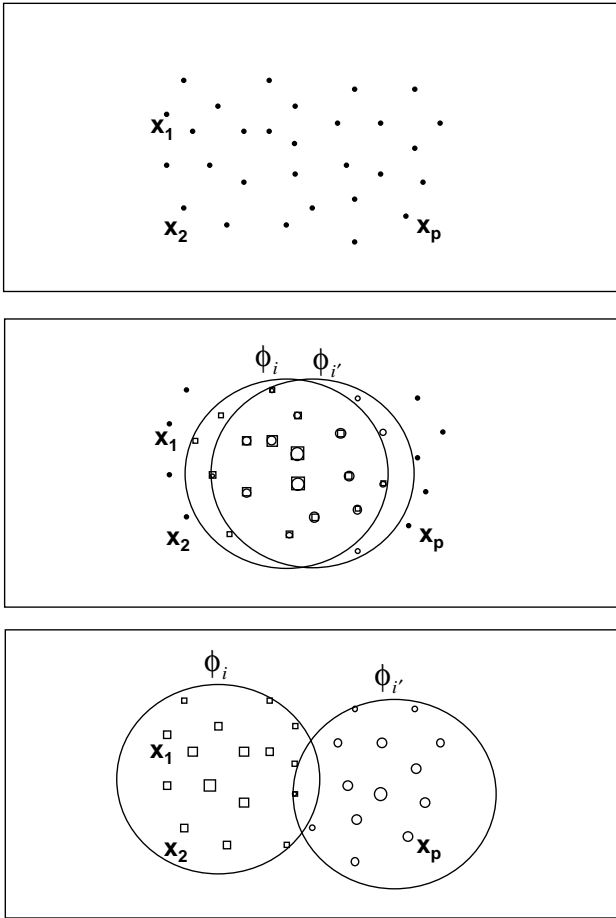


Fig. 2. Overlap in functionality of two RBFs ϕ_i and $\phi_{i'}$ as measured by the inner product of their normalized activation sequences. (a) The set of p training examples used to generate the activation sequences. Each point represents an input vector \mathbf{x}_j for one training example. (b, c) The tiny squares and tiny circles of parts b and c overlay this same set of data points. For each data point, the size of the tiny square around that data point represents the normalized activation of an RBF ϕ_i , while the size of the tiny circle around that data point represents the normalized activation of another RBF $\phi_{i'}$. (b) An example in which the inner product of the normalized activation sequences is high. The high inner product indicates that these two RBFs are attempting to do nearly the same job, and should therefore compete. (c) An example in which the inner product of the normalized activation sequences is low. The low inner product indicates that these two RBFs are performing different tasks and should therefore cooperate rather than compete.

two such RBFs, ϕ_i and $\phi_{i'}$. Figure 2a shows the set of data points \mathbf{x}_j , $j = 1, \dots, p$ in the domain of the function to be approximated. Parts b and c of the figure visualize the normalized RBF activations for each data point. If vectors $\hat{\phi}_i$ and $\hat{\phi}_{i'}$ are defined with these normalized activations as their components, then the inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ can be visualized as the unnormalized correlation of sizes between the set of tiny squares and the corresponding set of tiny circles.

If the inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is near 1 (as in Figure 2b), then ϕ_i and $\phi_{i'}$ are both attempting to do nearly the same job. In this case, they should directly compete in the sense

that selecting one of these RBFs to be in the next generation should tend to exclude selection of the other. This exclusivity can be modeled by forcing the ϕ_i and $\phi_{i'}$ to share fitness; for example, by forcing the fitness assigned to these two RBFs to sum to a fixed amount. In GA terminology, ϕ_i and $\phi_{i'}$ would then be sharing a *niche* [61].

By contrast, if $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is near zero (as in Figure 2c), then the RBFs ϕ_i and $\phi_{i'}$ are making relatively independent contributions to the overall prediction performed by the RBF network. In this case, the relationship between the RBFs should be mainly cooperative rather than competitive. It would not make sense for ϕ_i and $\phi_{i'}$ to compete, as they are performing two different functions, and both must be performed to successfully approximate f .

It is therefore desirable that RBFs be forced to share fitness to the extent that the inner product of their normalized activation sequences differs from zero. Before looking at specific fitness functions in this light, it will be useful to make some observations about the two extreme cases.

At one extreme, RBFs with mutually orthogonal activation sequences should occupy independent niches, so that increasing the fitness of any one RBF does not directly reduce the fitness of any other RBF. The set of RBF activation sequences $\{\hat{\phi}_i\}$, $i = 1, \dots, m$ would then form an orthonormal basis for the linear combination of RBFs used to approximate f . In this case, training by any least-squares method would converge toward an optimal set of weights in which each weight w_i is $\hat{\phi}_i \cdot \hat{f}$, where \hat{f} is the vector whose components are $(f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_p))$.

At the other extreme, when the inner product of the activation sequences $\hat{\phi}_i$ and $\hat{\phi}_{i'}$ is 1, then the underlying RBFs ϕ_i and $\phi_{i'}$ will be identical and the weights assigned to them will follow nearly identical trajectories under any learning rule which treats the RBFs approximately symmetrically. (An example of such a rule would be the LMS rule with weights initially zero.) When the inner product is -1, without loss of generality the sign of the function computed by one of the RBFs may be inverted and considered as a case where the inner product is 1.

With these considerations in mind, we need to understand how a given fitness measure will behave in the extreme cases where the inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is either 0 or 1, in comparison to the desired behavior. The desired behavior is that the fitness measure provide independent niches (yielding pure cooperation) when the inner product is 0, but niche sharing (yielding pure competition) when the inner product is 1.

B. Fitness measure $w_i^2/E(w_{i'}^2)$

First, consider $w_i^2/E(w_{i'}^2)$ as the fitness measure for each RBF ϕ_i , where $E()$ denotes mean value over all RBFs $\hat{\phi}_{i'}$, $i' = 1, \dots, m$ in the population. (The denominator $E(w_{i'}^2)$ normalizes the fitnesses to sum to m over a population of size m . The fitness therefore gives the expected number of copies of ϕ_i in the next generation.) At one extreme, as long as the RBF activations remain mutually orthogonal, any RBF center could be moved to increase its inner product with \hat{f} , with no effect on the inner products

with \hat{f} of other RBFs, and therefore with no effect on the weights assigned to other RBFs. The only remaining way such orthogonal RBFs could compete for fitness would be by affecting the fitness normalizing factor $E(w_i^2)$, but this normalizing factor will be relatively stable from one generation to the next, as $E(w_i^2) = \|\hat{f}\|^2/m - \epsilon^2$ where ϵ is the minimum rms error achieved by least-squares training of this population of RBFs. Thus, a fitness measure proportional to w_i^2 approximates the desired independence of niches for mutually orthogonal RBF activations.

Unfortunately, this fitness measure overshoots the desired behavior on the other extreme where the inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is 1. Consider a simple example in which the activations of ϕ_i are orthogonal to the activations of all other RBFs in G_k , and in which the fitness of ϕ_i is several (say, s) times that of the average fitness of all RBFs in G_k . Let us also assume that the weights w_i are determined by a learning rule in which RBFs are treated symmetrically. The fitness of the average RBF in G_k will accordingly be $w_i^2/sE(w_i^2)$. If there is one copy of ϕ_i in G_k , then by this definition of fitness there will be s identical copies expected in G_{k+1} . If only selection is considered, ignoring recombination, creep, and mutation, then these s copies will still be there when G_{k+1} is trained, and their weights will follow nearly identical trajectories during training. The linear combination of these s copies will be constrained to produce the same total weighted sum as that produced by $w_i\hat{\phi}_i$, so each will receive a weight of approximately w_i/s , and a fitness of approximately $w_i^2/s^2E(w_i^2)$, which is only $1/s$ of the average RBF fitness, producing a total of only one expected copy in G_{k+2} . The *total* fitness to be shared among these copies thus *decreases* as more RBFs share a niche, rather than remaining constant as desired. The level of competition is too high to allow stable niche sharing, and yields oscillation (see equation 3 below) instead.

C. Fitness measure $|w_i|/E(|w_{i'}|)$

A fitness measure of $|w_i|/E(|w_{i'}|)$ would solve this problem of excessive competition. In the case of s identical copies discussed above, each copy would now receive a fitness of $|w_i|/sE(|w_{i'}|)$ which is the average RBF fitness in G_k . The s copies would sum to the original fitness of ϕ_i , and so would be exactly sharing the niche originally occupied by ϕ_i alone. This level of competition yields a stable niche with s expected copies in each succeeding generation. But this fitness measure has trouble on the other extreme. Since the average population fitness $E(|w_{i'}|)$ is not a Euclidean norm in \mathfrak{R}^p , the Pythagorean theorem no longer forces the sum of the average population fitness and the mean squared training error to be constant (i.e., $\|\hat{f}\|^2/m$) as it was in Section III-B. Thus the mechanism of Section III-B no longer exists to prevent competition among RBFs with orthogonal activation sequences. In other words, there is no longer a mechanism to provide the independent niches needed evolve cooperative modeling of f .

Fitness measures proportional to both w_i^2 and $|w_i|$ are consistent with the basic argument that as $|\hat{\phi}_i \cdot \hat{\phi}_{i'}|$ increases, the degree of fitness sharing should increase, mov-

ing the relationship between these RBFs from cooperation to competition. On one hand, a fitness measure of $w_i^2/E(w_i^2)$ is too sensitive to the inner product, starting with the correct purely cooperative behavior at an inner product of zero but moving to excessive competition at an inner product of 1. On the other hand, $|w_i|/E(|w_{i'}|)$ is not sensitive enough to the inner product, having the correct purely competitive behavior when the inner product is 1, but failing to reduce the competition to pure cooperation when the inner product is zero. Neither fitness measure would be expected to yield the desired cooperative-competitive evolutionary behavior, and neither did in pilot studies.

D. Fitness measure $|w_i|^\beta/E(|w_{i'}|^\beta)$

Once the problem is stated in this way, it makes sense to consider fitness measures of the form $|w_i|^\beta/E(|w_{i'}|^\beta)$, with the idea that a compromise obtained by setting β between 1 and 2 might yield better results. Such values will still force too much competition among the copies of an RBF ϕ_i whose fitness is s times the population average, but now the resulting oscillation will be damped and will converge toward a fixed point of $s^{1/\beta}$ expected copies in each generation, considering selection alone. This can be seen most easily by defining y_k to be the expected number of copies of ϕ_i after k successive generations, in the absence of other correlated RBFs. For convenience, let $z_k = y_k/s^{1/\beta}$. Suppose w_i is the weight that would be assigned by LMS learning to ϕ_i if there were no other copies or significant correlates of ϕ_i in the population. Then in a generation k which has y_k copies of ϕ_i , the weight assigned to each copy will be w_i/y_k , and the fitness assigned to each copy will be $|w_i/y_k|^\beta/E(|w_{i'}|^\beta) = s/y_k^\beta$. This fitness per copy, multiplied by the number of copies in generation k , gives the expected number of copies in generation $k+1$:

$$y_{k+1} = \frac{s}{y_k^\beta} \cdot y_k. \quad (2)$$

Rewriting this in terms of z_{k+1} yields $z_{k+1} = z_k^{1-\beta}$. To see the resulting behavior clearly, let $u_k = \log(z_k)$, which gives $u_{k+1} = (1-\beta)u_k$, or

$$u_k = (1-\beta)^k u_0. \quad (3)$$

Equation 3 makes it clear why values of β between 1 and 2 might be desirable. Setting $\beta > 2$ produces a diverging oscillation in the number of copies of ϕ_i ; $\beta = 2$ produces a flip-flop; and $1 < \beta < 2$ produces a damped oscillation with $u_k \rightarrow 0$, and y_k converging to $s^{1/\beta}$ expected copies of ϕ_i . (Setting $\beta = 1$ produces a steady state of s copies, as noted in Section III-C, which argued that such values for β are insufficient to prevent competition among RBFs filling different niches. This situation can produce highly fit RBFs, but it cannot force them to distribute themselves to model all the examples of f , a deficiency which appeared in visualizations of pilot studies.)

In light of these arguments, a reasonable hypothesis might be that values of β between 1 and 2 are close enough

to the Euclidean norm to facilitate the formation of independent cooperating niches by the mechanism of Section III-B, while still providing stable (in the sense of convergence of equation 3) competition within each niche. The choice of $\beta = \frac{3}{2}$ was accordingly used in the experiments reported in Sections V and VI. The expected result is that fitness sharing, based on the inner product of activation sequences, should force RBFs away from competing RBFs with overlapping function, into unfilled niches where there is less competition [48].

Note that niche sharing based on the inner product of normalized activation sequences is accomplished *without* actually calculating a niche sharing function as in more explicit niche sharing approaches [58], [62], [63]. The inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is never actually calculated. Rather, fitness sharing based on this inner product is an automatic consequence of ordinary LMS learning when normalized RBFs are used. If $\|\hat{f}\|^2$ is considered to be a “resource” to be covered, in the sense of [59], then the LMS learning rule automatically shares this resource among overlapping RBFs, producing niches and cooperative coverage of the training examples in a manner analogous to that of recent genetic classifier systems [59], [64], [65].

IV. GENETIC ENCODING AND OPERATORS

As discussed in Section III, each RBF in the population G_k must be specified by a bit string ϕ_i which encodes the center \mathbf{c}_i and width d_i of the RBF. Before carrying out this encoding, the training data \mathbf{x}_j , $j = 1, \dots, p$ in each generation are scaled to occupy the unit hypercube $[0, 1]^n$. The centers of all RBFs are constrained to fall within this same unit hypercube.

A. Encoding

The first ℓ bits of the string ϕ_i encode the width d_i of the RBF as a binary fraction in $[0, 1]$. During evaluation of the genetic string, this binary fraction is rescaled to the range $(0, \text{MAX_UNIT_WIDTH}]$, where MAX_UNIT_WIDTH is a parameter of the algorithm specifying the maximum possible RBF radius in units of the $[0, 1]^n$ hypercube containing the scaled training data.

The vector $\mathbf{c}_i \in [0, 1]^n$ is encoded in the remaining $n\ell$ bits, where ℓ is the desired precision in bits with which each coordinate will be represented. A 2^n -tree encoding is used. Within these $n\ell$ bits, the first n bits select one of the 2^n smaller hypercubes obtained by bisecting each dimension of the unit hypercube. The selected hypercube is in turn subdivided into 2^n smaller hypercubes by the next n bits, and so on until ℓ subdivisions have been made. (In pilot studies, this encoding appeared to lead to better GA convergence than a conventional encoding of n successive coordinates, each encoded as an ℓ -bit binary fraction, although the difference in convergence rate between these two techniques did not appear to be of major significance.)

To summarize, the genetic population at generation k consists of m bit strings, each of length $L = (n + 1)\ell$, each representing the scalar radius of an RBF followed by a 2^n -tree encoding of the vector center of that RBF.

B. Genetic Operators

Genetic operators of recombination, creep, and mutation are employed. The recombination and mutation operators operate on bits in the genetic string without regard for the interpretation of these bits as RBF parameters, and without regard for the boundaries between the encodings of different parameters. The creep operator decodes the genetic bit string into real-valued RBF parameters, the parameter values are perturbed, and the perturbed values are encoded back into the bit string. After the selection process described in Section III generates the population G_k , these operators are applied to the members of that population.

The recombination operator is 2-point crossover. To recombine two strings in G_k , starting and ending crossover bit positions are selected randomly, and the bits from the starting position to the ending position (with wraparound) are exchanged between the two strings. Following typical GA mating restriction schemes where niching is desired [58], the probability of crossover decreases with a power function of the Euclidean distance between the two strings in $[0, 1]^{n+1}$. (Mating restriction was based on Euclidean distance rather than the inner product of normalized activation sequences $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ because the latter would have been computationally prohibitive. Unlike the fitness sharing described in Section III, the mating restriction function must be explicitly calculated.) Probabilities decreasing with various powers of this distance were tried in pilot studies, and the best results were obtained with a squared distance relationship. Two strings ϕ_i and ϕ_j are recombined only if they are within $\text{MAX_CROSSOVER_RADIUS}$ of each other, in which case the probability of recombination is $\min[1, \text{CROSS_RATE_FACTOR} \cdot (1 - \mathbf{d}_{ij}^2)]$ where

$$\mathbf{d}_{ij}^2 = \frac{\|\mathbf{c}_i - \mathbf{c}_j\|^2 + |d_i - d_j|^2}{(\text{MAX_CROSSOVER_RADIUS})^2}. \quad (4)$$

This $\text{MAX_CROSSOVER_RADIUS}$ is set to 1 in the first generation, and decays exponentially as in an annealing schedule, as discussed by [63]. This radius decays by a factor of $(1 - \text{CROSSOVER_RADIUS_DECAY})$ in each generation.

Generation G_{k+1} is initially formed from G_k by evaluating the fitness of the strings in G_k as described in Section III. The least fit $\text{GEN_PERCENT_REPLACE}$ percentage of the strings are deleted and replaced by proportional selection from the surviving strings using the stochastic remainder algorithm [66]. The strings in G_{k+1} are then placed in random order, and each successive pair of strings is considered for recombination with probability governed by equation (4).

A creep operator is applied to both strings of any pair not recombined by crossover. The creep operator is complementary to the crossover operator in this sense in order to perturb the exact replicas in G_{k+1} of a highly fit string G_k , and hence minimize the reproductive oscillations (discussed in Section III-D) caused by these exact replicas. The creep operator causes the width d_i and each component of the RBF center \mathbf{c}_i to be perturbed by adding a random scalar value drawn uniformly from the range $[-\text{CREEP_RANGE}/2, +\text{CREEP_RANGE}/2]$.

TABLE I
GA PARAMETERS FOR TIME SERIES PREDICTION

POPULATION_SIZE (different runs)	25, 50, 75, 100, 125, 150
GEN_PERCENT_REPLACE	0.25
MUTATION_RATE	1/POPULATION_SIZE
CREEP_RANGE	0.3
CROSS_RATE_FACTOR	2.0
MAX_CROSSOVER_RADIUS (initial value)	1.0
CROSSOVER_RADIUS_DECAY	0.001
LEARNING_RATE	0.06/ σ
TRAINING_QUOTA	3000
TRAINING_QUOTA_INCREMENT	10
MAX_UNIT_WIDTH	0.25

Mutation operates on individual bits of the strings in G_{k+1} , independently of the recombination and creep operators. Mutations (random bit flips) are scheduled with a probability of $1/m$ per bit, so that the total expected number of mutations in G_{k+1} will be $L = (n + 1)\ell$, i.e. one expected mutation for each bit position in the genetic encoding scheme.

C. RBF Network Training

After applying these genetic operators, the strings in G_{k+1} are decoded into the centers and widths of a set of RBFs, all of which will belong to the same neural network. This RBF network is trained to approximate the unknown function f by a training rule which minimizes the rms error $\|\hat{f} - w_0 - \sum_{i=1}^m w_i \hat{\phi}_i\|$. The fitness $|w_i|^\beta / E(|w_i|^\beta)$ assigned to each genetic string ϕ_i is accordingly based on the weight w_i given to the normalized activation sequence of the RBF encoded by that string.

To keep the total computation reasonable in comparison with non-genetic methods, the RBF network at each generation G_k is trained using the LMS algorithm [14] for only a few passes through the training data, yielding weights which are only an approximation of the optimal weights. The LMS algorithm employs a constant **LEARNING_RATE** to update the weights after each training example. The amount of training allowed at each generation is expressed as a constraint, **TRAINING_QUOTA**, on the product of the number of the RBFs and the number of passes through the training set. This is done so that different population sizes can be compared while holding the total amount of LMS training computation constant. The **TRAINING_QUOTA** is increased each generation by a constant **TRAINING_QUOTA_INCREMENT** to make the fitness evaluation process more accurate as the GA converges to networks capable of better approximations. Finally, SVD training is used once at the termination of the GA to more precisely evaluate the performance of the best network evolved by that GA. Objective comparisons can then be made with other RBF placement methods by applying the same SVD training to networks of RBFs placed by these methods.

V. FEASIBILITY TESTS

A. Time Series Prediction

A time-series prediction problem based on the Mackey-Glass [67] differential equation

$$\frac{dx(t)}{dt} = -bx(t) + a \cdot \frac{x(t-T)}{1 + x(t-T)^{10}} \quad (5)$$

is recognized as a benchmark for comparing the learning and generalization ability of different neural architectures. Following previous studies [12], [68], [69], [70], [71], [72], we generated this time series using the parameters $a = 0.2$, $b = 0.1$, and $T = 17$.

As in the studies cited above, the task for the neural network is to predict the value of the time series at point $x[t + I]$ from the earlier points ($x[t]$, $x[t - D]$, $x[t - 2D]$, $x[t - 3D]$) where the points used to make the prediction are spaced $D = 6$ time steps apart, and the point to be predicted is $I = 85$ time steps in the future. The justification for using this task with these parameters is discussed in [68]. (This is in contrast to tasks with much smaller values of D and T studied by [73].)

As in most previous work, 500 randomly selected points in the time series constituted the training data. These training examples were randomly chosen from points 500 to 4000 of the time series. These training data were used to train the RBF network arising in each generation of the GA, using the LMS [14] training rule with a training quota as described above.

Networks of Gaussian, inverse multiquadric, and thin-plate spline RBFs were evolved by the cooperative-competitive GA to approximate this training set. For each type of RBF, networks were evolved using GA population sizes of $m = 25, 50, 75, 100, 125$, and 150 genetic strings. As the cooperative-competitive algorithm encodes one RBF per genetic string, the number of RBF units in the evolved network is always the same as the population size of the GA. For each type of RBF, and for each population size given above, eight runs of the GA were made differing only in the seed initializing the random number

TABLE II
GA PARAMETERS FOR PATTERN CLASSIFICATION

POPULATION_SIZE (different runs)	2, 4, 6, 8, ..., 20
GEN_PERCENT_REPLACE	2/POPULATION_SIZE
MUTATION_RATE	1/POPULATION_SIZE
CREEP_RANGE	0.3
CROSS_RATE_FACTOR	2.0
MAX_CROSSOVER_RADIUS (initial value)	1.0
CROSSOVER_RADIUS_DECAY	0.001
LEARNING_RATE	$0.02/\sqrt{\text{POPULATION_SIZE}}$
TRAINING_QUOTA	3000
TRAINING_QUOTA_INCREMENT	10
MAX_UNIT_WIDTH	0.5

generator, and the results of these eight runs were averaged to yield the prediction errors reported in Section VI.

The genetic encoding for this problem used $\ell = 8$ bits per scalar value, a value chosen mainly for computational efficiency. Since the input space has dimension $n = 4$, each RBF was encoded in a 40-bit genetic string, 32 bits for the RBF center and 8 bits for the RBF width. The parameter values used in the GA are shown in Table I (in which σ is the standard deviation of the activation leading into the connection being trained).

The quality of the RBF networks produced by the GA was compared with RBF networks of the same size produced by k -means clustering [12], the method most commonly used to determine RBF centers. The k -means clustering algorithm was that used by [12] for the same task of predicting the Mackey-Glass time series using Gaussian RBFs. The RBF unit widths were then determined using the global first nearest neighbors algorithm used by [12] for this task. The results reported in Section VI for this clustering algorithm were consistent with those reported by [12].

Given a set of m RBF centers and widths evolved by the GA, the appropriate comparison is with a set of m RBFs produced by the k -means clustering algorithm above. To make this comparison objectively, each such set of RBFs was fit to the training data using singular value decomposition (SVD) to determine the weights of the optimal least-squares fit using the normalized RBF activations. The SVD was done in single precision (~ 7 significant digits), and so any singular value less than 10^{-7} of the largest singular value was zeroed. Each RBF network trained in this manner was then tested on a test data set consisting of 500 points in sequence following the end of the training data in the time series. The results in Section VI report the normalized error on this test data, where normalized error is the rms prediction error divided by the standard deviation of the correct prediction.

B. Pattern Classification

In addition to being used for approximating continuous functions, RBFs can be used for discrete pattern classification. The purpose of the RBFs in this case is to construct a high-dimensional basis in which the given classes are separable, or more nearly so than in their original low-dimensional basis [8]. The standard IRIS benchmark [74] contains data for three classifications of iris plants, each represented by 50 specimens identified by four measurements (sepal length, sepal width, petal length, petal width).

As in the time series prediction task, RBFs evolved by the cooperative-competitive GA were compared with those produced by k -means clustering. Both methods were applied to the iris classification problem using even-numbered population sizes ranging from 2 to 20 RBFs. For each population size, eight runs of the GA (differing only in the seed initializing the random number generator), and eight comparable runs of the k -means clustering algorithm (differing only in which data points were used to initialize the cluster centers) were made.

For this classification task, the LMS and SVD algorithms were extended to three outputs to encode the three types of iris as (1,0,0), (0,1,0), and (0,0,1). As before, LMS training was used within the evolutionary algorithm. The fitness function was extended to three outputs by taking the fitness of an RBF ϕ_i to be $\max\{|w_{hi}|^\beta / E(|w_{hi'}|^\beta)\}$, $h = 1, 2, 3$ where w_{hi} denotes the weight to output node h from RBF ϕ_i and where $\beta = \frac{3}{2}$ as discussed previously.

As before, SVD was used to determine the weights of the optimal least-squares fit obtainable using RBFs resulting from the GA, and using RBFs resulting from k -means clustering. Finally, to evaluate the classification performance of the resulting sets of RBFs, real-valued activations of three output nodes were converted into discrete classifications. Of the three output nodes, the one with the highest activation was defined to be the resulting discrete classification.

For the very small sets of RBFs evolved for this task, replacement of 25% of the population at each generation did

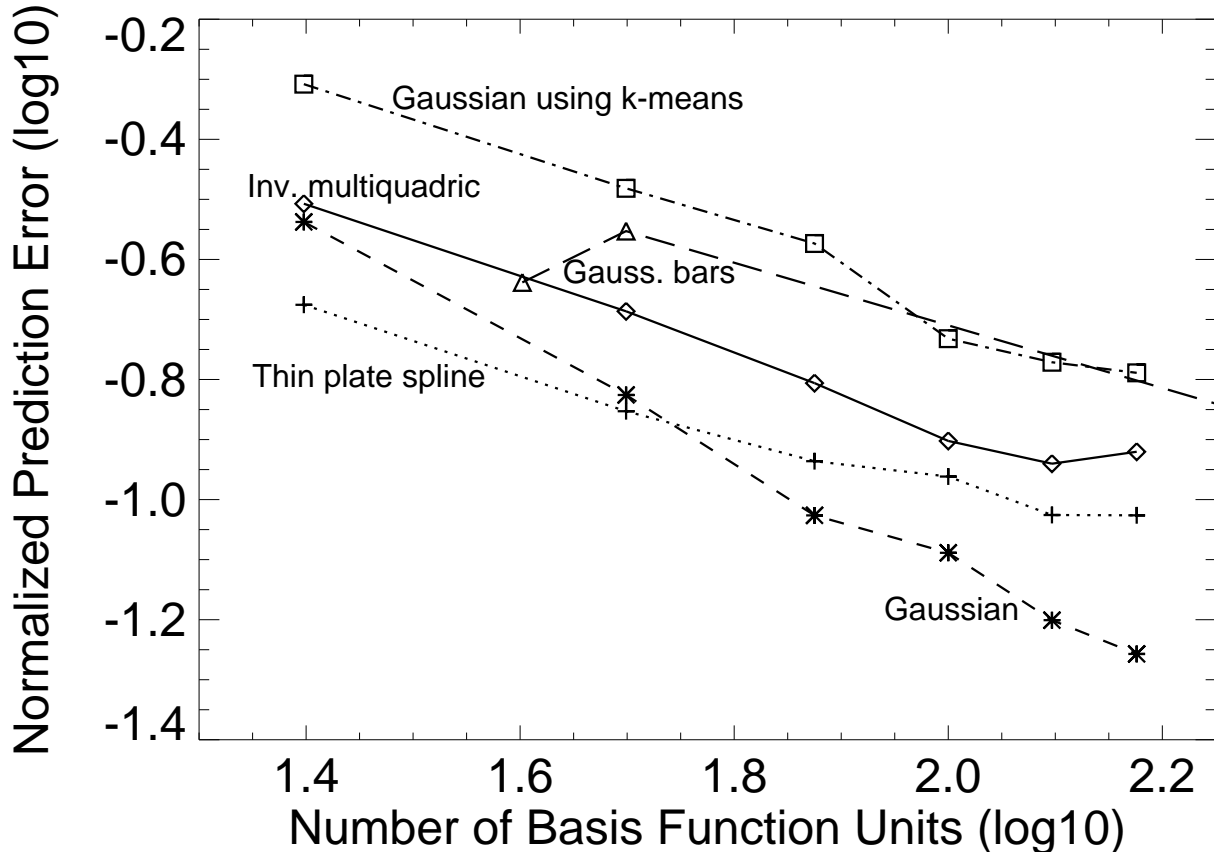


Fig. 3. Generalization performance of the cooperative-competitive genetic algorithm evolving Gaussian (—*—), inverse multiquadric (—◇—), and thin-plate-spline (···+···) RBFs, in comparison with Gaussian RBFs placed by k -means clustering (—□—). The performance of each method, using 25, 50, 75, 100, 125, and 150 RBFs, is measured as the normalized prediction error on test data later than the training data in the Mackey-Glass time series. For comparison, results of [72] using Gaussian bars (—△—) are also shown.

not yield a stable evolutionary process. To ameliorate this, a “steady-state” GA was used which replaced (and with specified probability recombined) only two RBFs during each generation. Also due to the small numbers of RBFs employed, the maximum RBF width was increased from $\frac{1}{4}$ to $\frac{1}{2}$ of the unit hypercube containing the data. Because of the discrete $\{0,1\}$ -valued target outputs, the RBFs were not normalized, and accordingly the LMS learning rate was based on population size rather than on the standard deviation of normalized RBF activations. In all other respects, the GA algorithm and its parameter settings (shown in Table II) were the same in this discrete classification task as in the continuous time series prediction task.

VI. RESULTS AND DISCUSSION

A. Time Series Prediction

Figure 3 compares the normalized prediction error produced by the cooperative-competitive GA (evolving Gaussian, inverse multiquadric, and thin-plate spline RBFs) with that produced by k -means clustering.

Recall that in all cases the same SVD algorithm is used to obtain the best least-squares fit to the training data. The resulting least-squares fit is then used to predict the

test data. What varies in Figure 3 is that a different set of RBFs is used to make the fit in each case. The quality of sets of RBFs evolved by the cooperative-competitive algorithm is therefore compared with the quality of sets of RBFs produced by k -means clustering. The results of [72] using non-radially-symmetric Gaussian bars are also shown for comparison.

The main result is that the cooperative-competitive genetic algorithm appears to evolve sets of Gaussian RBFs which are superior to those produced by the k -means method for this time-series prediction task. The gap between the two on the log plots translates into a prediction error for the evolved Gaussians that is roughly $\frac{1}{3}$ to $\frac{1}{2}$ the prediction error of the corresponding set of Gaussians produced by clustering.

An additional result is that the cooperative-competitive algorithm is feasible for all three types of RBFs investigated, although for inverse multiquadric and thin-plate spline RBFs, adding more than 100 units does not appear to improve prediction performance as much as it does for Gaussians. In particular, performance worsens when the number of inverse multiquadric RBFs increases from 125 to 150. This appears to be due to overfitting the data, be-

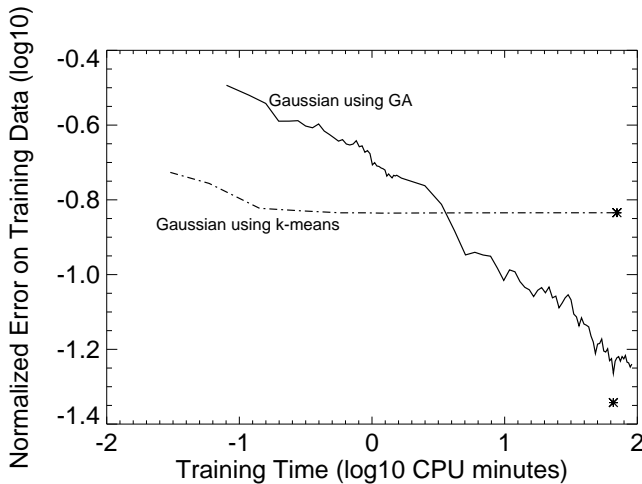


Fig. 4. Training error as a function of Sparc10 CPU time for the cooperative-competitive genetic algorithm (—) in comparison with k -means clustering (---), using 100 Gaussian RBFs in both cases. SVD training of the best network produced by each algorithm is shown by a *.

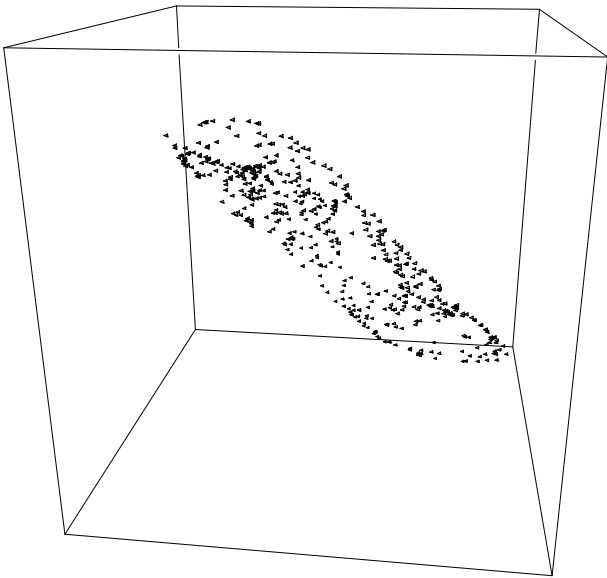


Fig. 5. Locations of the training examples \mathbf{x}_j , $j = 1, \dots, 500$ in the Mackey-Glass time-series prediction problem. The cube represents a 3-dimensional projection of the 4-dimensional input space. The location of each training example within this cube is shown as a tiny tetrahedron, in order to be distinguishable from the spheres of various sizes which represent RBFs in Figure 6.

cause the error on the training data (not shown in the figure) decreases smoothly and monotonically as more RBFs are added.

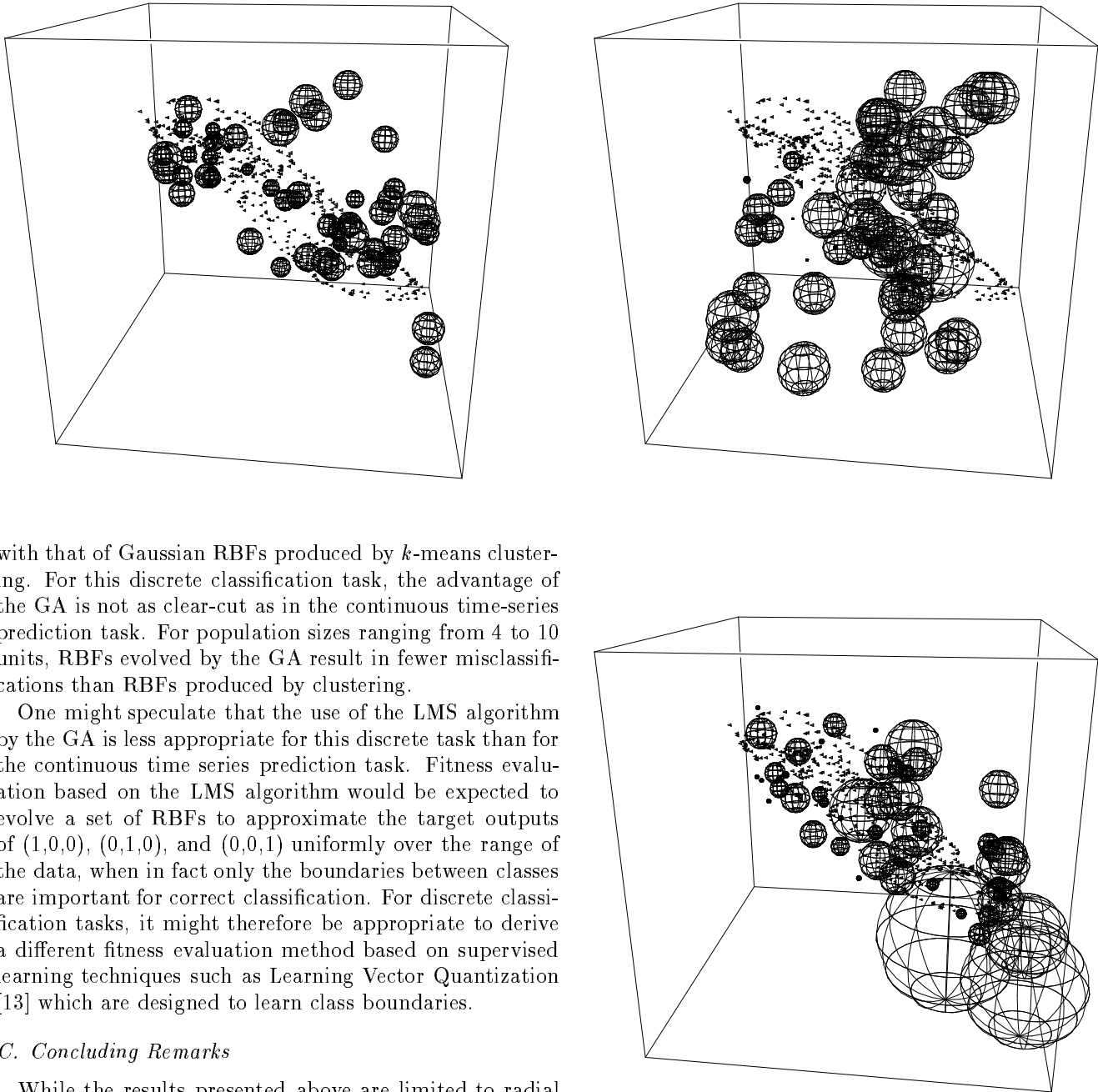
Figure 4 compares the computational effort required by the genetic algorithm with that required by k -means clustering. The k -means algorithm typically employs a predetermined schedule of decreasing step size, and it is not fair to judge its performance when this schedule is only partly completed. Accordingly, the k -means training errors were obtained by varying the rate of decrease in the training schedule, allowing each schedule to run to completion. In accordance with [13], the step size (learning rate) began at 0.1 and decreased linearly to 0 in all cases. All values plotted for the k -means method are the result of SVD training. The figure shows LMS training within each generation of the GA, as well as SVD training of the best generation.

Figure 4 shows that the genetic algorithm's training error falls below that of k -means clustering after about 4 minutes of CPU time, and continues to decrease thereafter. 72 RBF networks were evaluated in these 4 minutes. This initial training performance depends on the fact that the cooperative-competitive algorithm evaluates only one network per generation, rather than a whole population of competing networks. It may also result from competition for fitness based implicitly on the inner product of activation sequences. As Section III-D argues, this competition should push RBFs away from each other (in terms of their activation sequences) so as to efficiently fill the niches needed to cover $\|\hat{f}\|^2$. Note that in normalized units, any network's coverage of $\|\hat{f}\|^2$ is simply $1 - \epsilon^2$, where ϵ is the normalized error shown in Figures 3 and 4. Figure 4, as well as visualizations of RBF placement after each generation, support the expectation that this coverage is achieved rapidly, even when the initial random distribution of centers is much more dispersed than that of k -means clustering. After all the relevant niches are populated by the cooperative-competitive algorithm, the subsequent computation might be speeded up by switching to gradient descent for "fine tuning" the RBF locations within each niche [37], [54].

The evolutionary algorithm may achieve better prediction performance than k -means clustering partly because the evolutionary algorithm is free to select RBF locations outside the convex hull of the training data. K -means clustering, by contrast, places RBF centers within this convex hull, which may not be optimal [5]. Figures 5 and 6 show that the evolutionary algorithm does indeed select many RBF centers outside the convex hull of the training data. For all three types of RBFs, especially the Gaussians (Fig. 6a) and thin-plate-splines (Fig. 6b), a large portion of the evolved RBF centers lie outside the training data. This observation supports the suggestion [5] that the globally optimal set of RBF centers does not necessarily lie within the convex hull of the training data.

B. Pattern Classification

Figure 7 compares the classification performance of Gaussian RBFs evolved by the cooperative-competitive GA



with that of Gaussian RBFs produced by k -means clustering. For this discrete classification task, the advantage of the GA is not as clear-cut as in the continuous time-series prediction task. For population sizes ranging from 4 to 10 units, RBFs evolved by the GA result in fewer misclassifications than RBFs produced by clustering.

One might speculate that the use of the LMS algorithm by the GA is less appropriate for this discrete task than for the continuous time series prediction task. Fitness evaluation based on the LMS algorithm would be expected to evolve a set of RBFs to approximate the target outputs of $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ uniformly over the range of the data, when in fact only the boundaries between classes are important for correct classification. For discrete classification tasks, it might therefore be appropriate to derive a different fitness evaluation method based on supervised learning techniques such as Learning Vector Quantization [13] which are designed to learn class boundaries.

C. Concluding Remarks

While the results presented above are limited to radial basis functions, the same methods might be applied to non-radially-symmetric basis functions. Scalar basis functions, for example, are potentially more efficient than RBFs in covering input spaces of high dimensionality [72]. Neural algorithms which successively add scalar units along dimensions where the error reduction would be greatest [75] might be extendible into a niche-based fitness evaluation mechanism.

Approximation methods employing basis functions [76], [53] have obtained good results by including linear terms in addition to the given non-linear basis functions. Use of such a technique in a cooperative-competitive GA would require a fitness evaluation of basis functions which takes

Fig. 6. Centers and widths of RBFs evolved by the cooperative-competitive algorithm. The three plots shown in parts a, b, and c each represent the best network evolved by the algorithm using (a) Gaussian RBFs, (b) thin-plate-spline RBFs, and (c) inverse multiquadric RBFs, for a population size of 50 RBFs in each case. The projection of the input space and locations of the training data are the same as shown in Figure 5. The location of each RBF ϕ_i is represented by a sphere whose center is the center c_i of the RBF, and whose radius is (a) $\frac{1}{4}d_i$, (b) $\frac{1}{8}d_i$, or (c) d_i (the fractions of d_i being chosen to avoid clutter in each display). Note that for the Gaussian and thin-plate spline RBFs shown in parts a and b, a large portion of the RBF centers lie outside the convex hull of the training data.

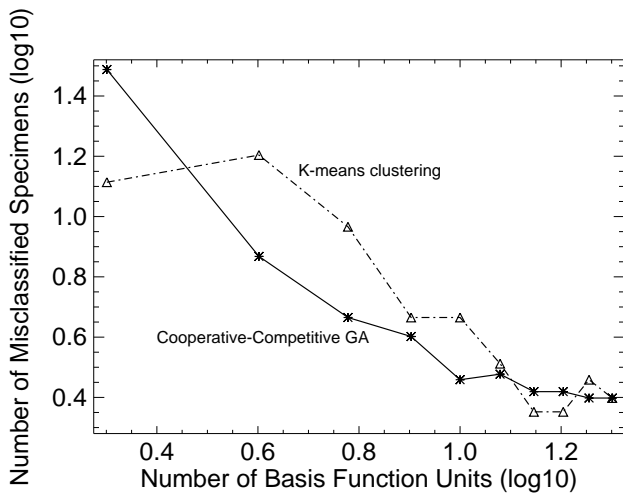


Fig. 7. Classification performance of sets of Gaussian RBFs (—*) evolved by the cooperative-competitive genetic algorithm, in comparison with those placed by k -means clustering (--- Δ ---). The performance of each method, using 2, 4, 6, 8, ..., 20 RBFs, is measured as the number of misclassified specimens, where discrete classifications are derived from continuously-valued outputs of affine combinations of RBFs as described in Section V-B.

into account the contribution of the linear terms. An approach to basis function normalization derived from Taylor expansions around RBF centers [76] might be useful in deriving such a fitness evaluation.

In conclusion, it is feasible to evolve an RBF network by genetically selecting individual RBFs, based on their individual contribution to the performance of the network as a whole. To simultaneously evolve a complete set of RBFs within a single genetic population, a fitness function must be devised which promotes competition or cooperation among RBFs depending on the degree of overlap in the contribution they make to the overall job of approximating the function represented by the training examples. The appropriate blend of cooperation and competition can be provided by niche sharing. A fitness function of the form $|w_i|^\beta / E(|w_i|^\beta)$ where $1 < \beta < 2$ approximates the desired form of niche sharing without the need to explicitly calculate a niche sharing function for every pair of RBFs in every generation. The feasibility of evolving an RBF network in this manner has been demonstrated on a standard time-series prediction benchmark and a standard pattern classification task. The results appear promising enough to merit further theoretical study of the niche sharing mechanism, and further experimental study of the performance of the algorithm on other types of problems.

VII. ACKNOWLEDGMENTS

Visualizations of RBF centers and widths were based on software written by Kenneth McDonald employing the VOGL graphics library developed by Eric Echidna. We are grateful to these authors for making their source code freely available. We thank the anonymous reviewers for many constructive suggestions.

REFERENCES

- [1] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.
- [2] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," in *IMA Conference on Algorithms for the Approximation of Functions and Data* (J. C. Mason and M. G. Cox, eds.), pp. 143–167, Oxford University Press, 1987.
- [3] M. J. D. Powell, "The theory of radial basis function approximation in 1990, volume II," in *Advances in Numerical Analysis* (W. Light, ed.), pp. 105–210, Oxford: Clarendon Press, 1992.
- [4] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.
- [5] F. Girosi, "Some extensions of radial basis functions and their applications in artificial intelligence," *Computers Math. Applic.*, vol. 24, no. 12, pp. 61–80, 1992.
- [6] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummel, "On the training of radial basis function classifiers," *Neural Networks*, vol. 5, pp. 595–603, 1992.
- [7] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.
- [8] S. Renals and R. Rohwer, "Phoneme classification experiments using radial basis functions," in *IJCNN: International Joint Conference on Neural Networks*, pp. I-461–I-467, Piscataway, N.J.: IEEE, June 1989.
- [9] K. J. Cios, R. E. Tjia, N. Liu, and R. A. Langenderfer, "Study of continuous ID3 and radial basis function algorithms for the recognition of glass defects," in *IJCNN-91-Seattle: International Joint Conference on Neural Networks*, pp. I-49–I-54, Piscataway, N.J.: IEEE, July 1991.
- [10] T. A. Foley, "The map and blend scattered data interpolant on a sphere," *Computers Math. Applic.*, vol. 24, no. 12, pp. 49–60, 1992.
- [11] C. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, vol. 3, no. 4, pp. 579–588, 1991.
- [12] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.
- [13] T. Kohonen, "An introduction to neural computing," *Neural Networks*, vol. 1, no. 1, pp. 3–16, 1988.
- [14] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 WESCON Convention*, pp. 96–104, New York: Institute of Radio Engineers, 1960.
- [15] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [16] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), pp. 379–384, San Francisco: Morgan Kaufmann, June 1989.
- [17] S. A. Harp, T. Samad, and A. Guha, "Designing application-specific neural networks using the genetic algorithm," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 447–454, San Francisco: Morgan Kaufmann, 1990.
- [18] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, pp. 461–476, 1990.
- [19] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," in *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks* (S. Forrest, ed.), pp. 244–248, Cambridge, MA: MIT Press, 1990.
- [20] S. W. Wilson, "Perceptron redux: Emergence of structure," in *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks* (S. Forrest, ed.), pp. 249–256, Cambridge, MA: The MIT Press, 1990.
- [21] S. A. Harp and T. Samad, "Genetic optimization of self-organizing feature maps," in *IJCNN-91-Seattle: International Joint Conference on Neural Networks*, pp. I-341–I-346, Piscataway, N.J.: IEEE, July 1991.
- [22] J. W. L. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Networks*, vol. 4, pp. 53–60, 1991.

- [23] K. U. Hoffgen, H. P. Siemon, and A. Ultsch, "Genetic improvements of feedforward nets for approximating functions," in *Lecture Notes in Computer Science 496: Parallel Problem Solving from Nature* (G. Goos and J. Hartmanis, eds.), pp. 302–306, Berlin: Springer-Verlag, 1991.
- [24] W. Schiffmann, M. Joost, and R. Werner, "Performance evaluation of evolutionarily created neural network topologies," in *Lecture Notes in Computer Science 496: Parallel Problem Solving from Nature*, pp. 274–283, Berlin: Springer-Verlag, 1991.
- [25] S. Bornholdt and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms," *Neural Networks*, vol. 5, no. 1, pp. 327–334, 1992.
- [26] F. Z. Brill, D. E. Brown, and W. N. Martin, "Fast genetic selection of features for neural network classifiers," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 324–328, 1992.
- [27] J. G. Elias, "Genetic generation of connection patterns for a dynamic artificial neural network," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 38–54, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [28] P. J. Hancock, "Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 108–122, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [29] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 39–53, 1994.
- [30] X. Yao, "A review of evolutionary artificial neural networks," *International Journal of Intelligent Systems*, vol. 8, no. 4, pp. 539–567, 1993.
- [31] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 1–37, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [32] T. P. Caudell and C. P. Dolan, "Parametric connectivity: Training of constrained networks using genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), pp. 370–374, San Francisco: Morgan Kaufmann, June 1989.
- [33] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 762–767, San Francisco: Morgan Kaufmann, August 1989.
- [34] D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), pp. 391–396, San Francisco: Morgan Kaufmann, June 1989.
- [35] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybernetics*, vol. 63, pp. 487–493, 1990.
- [36] H. de Garis, "Genetic programming: Building artificial nervous systems using genetically programmed neural network modules," in *Machine Learning: Proceedings of the Seventh International Conference* (B. Porter and R. Mooney, eds.), pp. 132–139, San Francisco: Morgan Kaufmann, June 1990.
- [37] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in *Proceedings of the Eighth National Conference on AI (AAAI-90)*, pp. 789–795, Cambridge, MA: MIT Press, July–August 1990.
- [38] J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for a neural network," in *IJCNN-91-Seattle: International Joint Conference on Neural Networks*, pp. II-397–II-404, Piscataway, N.J.: IEEE, July 1991.
- [39] S. Dominic, R. Das, D. Whitley, and C. Anderson, "Genetic reinforcement learning for neural networks," in *IJCNN-91-Seattle: International Joint Conference on Neural Networks*, pp. II-71–II-76, Piscataway, N.J.: IEEE, July 1991.
- [40] H. Braun and J. Weisbrod, "Evolving neural networks for application oriented problems," in *Proceeding of 2nd Annual Conference on Evolutionary Programming* (D. B. Fogel and W. Atmar, eds.), pp. 62–71, La Jolla, CA: Evolutionary Programming Society, 1993.
- [41] D. B. Fogel, "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe," in *International Conference on Neural Networks*, pp. 875–880, Piscataway, N.J.: IEEE, 1993.
- [42] J. R. McDonnell and D. Waagen, "Neural network structure design by evolutionary programming," in *Proceeding of 2nd Annual Conference on Evolutionary Programming* (D. B. Fogel and W. Atmar, eds.), pp. 79–89, La Jolla, CA: Evolutionary Programming Society, 1993.
- [43] S. Oliner and M. Furst, "A distributed genetic algorithm for neural network design and training," *Complex Systems*, vol. 6, no. 5, pp. 459–477, 1992.
- [44] D. Dasgupta and D. R. McGregor, "Designing application-specific neural networks using the structured genetic algorithm," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 87–107, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [45] J. McDonnell and D. Waagen, "Evolving recurrent perceptrons for time-series modeling," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 24–38, 1994.
- [46] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.
- [47] B. A. Whitehead and T. D. Choate, "Evolving space-filling curves to distribute radial basis functions over an input space," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 15–23, 1994.
- [48] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press, 1975.
- [49] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2* (R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds.), pp. 524–532, San Francisco: Morgan Kaufmann, 1991.
- [50] J.-N. Hwang, S.-R. 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, no. 3, pp. 342–353, 1994.
- [51] N. Karunanithi, R. Das, and D. Whitley, "Genetic cascade learning for neural networks," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 134–145, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [52] M. A. Potter, "A genetic cascade-correlation learning algorithm," in *Combinations of Genetic Algorithms and Neural Networks* (L. D. Whitley and J. D. Schaffer, eds.), pp. 123–133, Los Alamitos, CA: IEEE Computer Society Press, June 1992.
- [53] J. McDonnell and D. Waagen, "Evolving cascade-correlation networks for time-series forecasting," *IEEE Transactions on Systems, Man, and Cybernetics*, 1995. in press.
- [54] S. V. Odri, D. P. Petrovacki, and G. A. Krstonosic, "Evolutional development of a multilevel neural network," *Neural Networks*, vol. 6, no. 4, pp. 583–595, 1993.
- [55] G. F. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Technical Report 1-37, Carnegie-Mellon University, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.
- [56] R. Smalz and M. Conrad, "Combining evolution with credit apportionment: A new learning algorithm for neural nets," *Neural Networks*, vol. 7, no. 2, pp. 341–351, 1994.
- [57] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive algorithms," doctoral dissertation, University of Michigan, 1975.
- [58] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 42–50, San Francisco: Morgan Kaufmann, June 1989.
- [59] J. Horn, D. E. Goldberg, and K. Deb, "Implicit niching in a learning classifier system: Nature's way," *Evolutionary Computation*, vol. 2, no. 1, pp. 37–66, 1994.
- [60] R. E. Smith, S. Forrest, and A. S. Perelson, "Searching for diverse, cooperative populations with genetic algorithms," *Evolutionary Computation*, vol. 1, no. 2, pp. 127–149, 1993.
- [61] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.
- [62] D. E. Goldberg, K. Deb, and J. Horn, "Massive multimodality, deception, and genetic algorithms," in *Parallel Problem Solving*

- from *Nature*, 2 (R. Manner and B. Manderick, eds.), pp. 37–50, Elsevier Science, 1992.
- [63] M. E. Palmer and S. J. Smith, “Improved evolutionary optimization of difficult landscapes: Control of premature convergence through scheduled sharing,” *Complex Systems*, vol. 5, no. 5, pp. 443–458, 1991.
 - [64] R. E. Smith and H. B. Cribbs, III, “Is a learning classifier a type of neural network?,” *Evolutionary Computation*, vol. 2, no. 1, pp. 19–36, 1994.
 - [65] D. P. Greene and S. F. Smith, “Using coverage as a model building constraint in learning classifier systems,” *Evolutionary Computation*, vol. 2, no. 1, pp. 67–91, 1994.
 - [66] J. E. Baker, “Reducing bias and inefficiency in the selection algorithm,” in *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 14–21, San Francisco: Morgan Kaufmann, 1987.
 - [67] M. C. Mackey and L. Glass, “Oscillation and chaos in physiological control systems,” *Science*, vol. 197, pp. 287–289, 1977.
 - [68] A. Lapedes and R. Farber, “How neural nets work,” in *Neural Information Processing Systems*, pp. 442–456, New York, N.Y.: American Institute of Physics, 1988.
 - [69] J. D. Farmer and J. J. Sidorowich, “Predicting chaotic time series,” *Physical Review Letters*, vol. 59, no. 8, pp. 845–848, 1987.
 - [70] M. F. Tenorio, “Self-organizing network for optimum supervised learning,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 100–110, 1990.
 - [71] J. Platt, “A resource-allocating network for function interpolation,” *Neural Computation*, vol. 3, pp. 213–225, 1991.
 - [72] E. Hartman and J. D. Keeler, “Predicting the future: Advantages of semilocal units,” *Neural Computation*, vol. 3, no. 4, pp. 566–578, 1991.
 - [73] A. M. Logar, E. M. Corwin, and W. J. Oldham, “A comparison of recurrent neural network learning algorithms,” in *1993 IEEE International Conference on Neural Networks*, pp. 1129–1134, Piscataway, N.J.: IEEE, 1993.
 - [74] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annual Eugenics*, vol. 7, Part II, pp. 179–188, 1936. (Data set available by anonymous ftp from `ftp.ics.uci.edu:/pub/machine-learning-databases/iris`).
 - [75] T. D. Sanger, “A tree-structured adaptive network for function approximation in high-dimensional spaces,” *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 285–293, 1991.
 - [76] W. C. Mead, R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, L. A. Lee, and M. K. O’Rourke, “Using CNLS-net to predict the Mackey-Glass chaotic time series,” in *IJCNN: International Joint Conference on Neural Networks*, pp. II-485–II-490, Piscataway, N.J.: IEEE, July 1991.