Evolving Space-Filling Curves to Distribute Radial Basis Functions Over an Input Space

Bruce A. Whitehead and Timothy D. Choate

Abstract—**An evolutionary neural network training algorithm is proposed for radial basis function (RBF) networks. The locations of basis function centers are not directly encoded in a genetic string, but are governed by space-filling curves whose parameters evolve genetically. This encoding causes each group of co-determined basis functions to evolve to fit a region of the input space. A network produced from this encoding is evaluated by training its output connections only. Networks produced by this evolutionary algorithm appear to have better generalization performance on the Mackey-Glass time series than corresponding networks whose centers are determined by k-means clustering.**

I. Introduction

A NEURAL NETWORK model typically describes a class of architectures from which a specific architecture may be instantiated by choosing a network structure and various associated parameters. Training a neural network typically requires a human to make many training runs with different choices of structures and parameter settings. In this trial-and-error process, two nested optimizations are performed. One optimization (typically the outer loop) consists of finding the best specific structure and parameter settings for the network. The parameters to be selected may include both constants, such as the rate of gradient descent, and constraints, such as ranges of initial distributions of basis function centers or weights. Given a choice of structure and parameters, the second optimization (typically the inner loop) is then the training of the neural network. Training attempts to optimize the connection weights to yield the best performance of the neural network over the training data.

Genetic algorithms (GA's) have been applied to both types of neural network optimization problems, individually and in various combinations. Since the extensive literature on these efforts has been thoroughly reviewed by Yao [1], such a detailed review will not be repeated here. Yao identifies three basic approaches that are used in these efforts.

In one type of approach, the entire set of connection weights is encoded into either a binary [2]–[4] or real-valued [5]–[8] string. A population of these strings is randomly generated and then evolves by repeated application of selection, mutation, and sometimes recombination. Since each such string encodes a neural network with a completely specified set of weights, its evaluation is a relatively simple computation. The fitness of the string is evaluated by translating it into the encoded neural network and making one pass through the training set. The performance of the neural network is evaluated over the training set with all connection weights fixed at the values specified in the genetic string. While the specific encoding of every connection weight makes it easy to evaluate fitness, it has the disadvantage that the space to be searched is very large. In an architecture with N connection weights, the space of all possible combinations of weights is an N-dimensional space. Unfortunately, the number of connection weights N grows more rapidly than the number of nodes in the network, so the dimensionality of the space to be searched is very large for the large neural networks typically used to solve difficult problems.

The other two approaches identified by Yao are the evolution of neural network architectures [9]–[12], and the evolution of learning rules and their associated parameters [13], [14] and fitness functions [15]. In these approaches, the genetic string does not encode the specific connection weights of the neural network. Rather, the string encodes the connection pattern and/or learning rule of an untrained neural network. The fitness of this neural network must then be evaluated by training it on a data set using a conventional neural network training algorithm. The space to be searched is relatively small in comparison with the first approach above, particularly if the string encodes parameters governing the connection pattern, rather than the detailed connection pattern itself [13]–[17]. The disadvantage in comparison with the first approach is that an entire neural network training procedure must be carried out for each string in the population for each generation of the evolutionary search. The evolution of both neural network architectures and connection weights can also be accomplished by evolutionary programming (EP) algorithms [18] that operate on neural network structures directly rather than on specifications encoded in strings. EP algorithms have been employed to simultaneously evolve both the architecture and weights of feedforward neural networks, either by doing crossover directly on the neural structures [19] or by employing mutation only [18], [20], [21].

II. Approach

In this paper, we propose a new approach that lies between the one extreme of a genetic encoding that specifies all the weights, requiring no training to be evaluated, and the other extreme of a genetic encoding that specifies none of the weights and that therefore requires lengthy neural network training to be evaluated. Consider that the weights in a typical two-layer feedforward network determine two func-

Manuscript received January 3, 1993; revised July 26, 1993.

B. A. Whitehead is with the Computer Science Program, University of Tennessee Space Institute, Tullahoma, TN 37388.

T. D. Choate is with ERC, Inc. Tullahoma, TN 37388.

IEEE Log Number 9213548.

tions whose composition is computed by the neural network: 1) the function from the input to the first-layer activation, and 2) the function from the first-layer activation to the output.

Our approach is to genetically encode the parameters of a space-filling curve that determines the function from the input to the first layer. Each genetic string thus encodes a neural network in which only the weights from the first layer to the output remain unspecified. The training required to evaluate such a network is computationally very light compared with the training required to modify both sets of connection weights simultaneously (as in backpropagation).

The neural network architecture adapted by our approach is the radial-basis-function (RBF) architecture [22]–[25]. Coordinates of basis function centers take the place of weights in specifying the function from the input to the first-layer activations. The goal of the GA is to optimally position these RBF centers. If the genetic string were to encode every center coordinate, then our approach would not differ greatly from approaches that encode all the weights in a network—the GA would still have a very large search space.

The local geometry of radial basis functions, however, suggests a more parsimonious approach. Each RBF is local in the sense that it responds only to the portion of the input space in the neighborhood of its center. The set of RBF centers should therefore be distributed so as to cover the region of the input space to which the neural network must respond. Because RBF's are local, however, they are subject to the "curse of dimensionality." Even for a relatively small number of dimensions, it would be computationally prohibitive to fill a volume of the input space with a regularly-spaced grid of RBF centers.

What is desired, therefore, is a way to generate sets of RBF's that are spread out over regions of the input space to the extent possible given that there are too few RBF's to fill the regions in grid-like manner. We therefore desire a genetic encoding that produces not all possible sets of RBF centers but only sets that are spread out over relevant portions of the input space. Furthermore, we would like to generate such combinations of RBF's from an encoding with many fewer degrees of freedom than that of all possible combinations of RBF centers. Only by reducing the degrees of freedom in the genetic encoding [16] can we expect to reduce the size of the search space significantly in comparison with GA's that encode every weight.

We therefore adapt a genetic string that encodes the parameters of an adjustable space-filling curve. Such a curve maps one dimension (distance along the curve) into the number of dimensions of the input space. The basis function centers are then generated at equal intervals along this curve. In the case where the number of basis functions happens to be an integer raised to the power of the number of dimensions, this algorithm produces a regular grid filling the specified region of the input space. In the typical case where there are far too few RBF's to produce a regular grid, the result still yields RBF's that are reasonably spread out over the space. These concepts are illustrated in only two dimensions in Fig. 1, but will work in the same way for any number of dimensions.

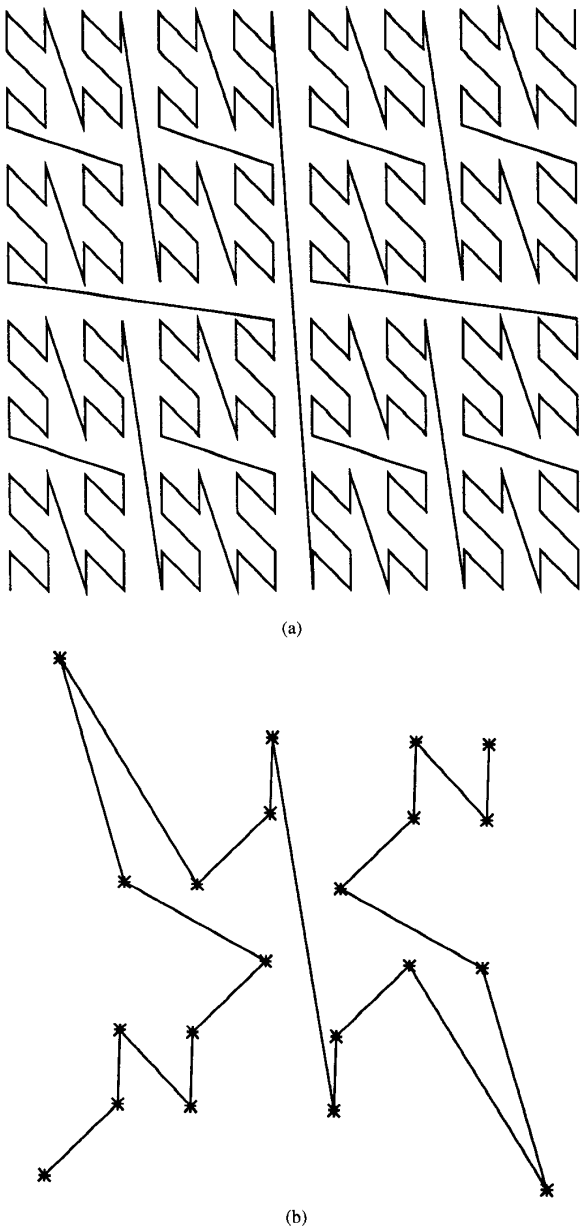


Fig. 1. (a) Simple two-dimensional example of a space-filling curve generated by the dovetail algorithm with $m = 4, n = 2$. The curve fills the unit square. The curve is generated by the sequence of 256 binary fractions $0/256, 1/256, \ldots, 255/256$. Each such binary fraction is encoded in 8 bits, which are translated into two coordinates for each point plotted: the odd (first, third, fifth, and seventh) bits are concatenated to form a binary fraction encoding of the x-axis coordinate; the even (second, fourth, sixth, and eighth) bits form the y-axis coordinate. (b) 21 points placed at equal intervals along a higher-precision ($m = 8, n = 2$) version of the two-dimensional space-filling curve in (a). Connecting lines merely indicate the sequence of the points traversing the curve.

This type of genetic encoding will clearly generate a much smaller search space than one that explicitly encodes every connection weight or every RBF center. This gain comes at a price, however. The weights from the first layer to the output

layer (which form linear combinations of the RBF's) are left unspecified by this encoding, and must therefore be trained by a conventional technique that minimizes either the mean-squared error over the training set, or some more general regularization criterion [26]–[28] that takes into account the smoothness of the function computed by the linear combination of basis functions. Since only the connections from the first layer to the output node are modified, and since the weights linearly combine the RBF activations into the neural network output, the total squared error over the training set is a quadratic function of the weights. This function can be minimized relatively easily either by direct algebraic methods such as singular value decomposition or by the commonly-used LMS learning rule [29]. (These minimization methods can also be used with a regularization criterion, if the criterion can be formulated as a quadratic function of the weights [28].) We can therefore expect the evaluation of a genetic string encoding an RBF to be computationally much lighter than a multi-layer training algorithm such as backpropagation. Furthermore, the local response of each RBF can be truncated to zero beyond a genetically selected radius, so that only a small fraction of the weights are modified for each training example. All of these factors can lessen the burden of the "loop within a loop" training of a neural network within a GA.

The LMS gradient descent algorithm with an appropriate (genetically selected) learning rate can be expected to yield a large error reduction on the first few passes through the training data. Even if the number of training passes is severely limited, therefore, it is conceivable that the LMS algorithm will yield a rough indication of the optimal mean-squared error to which it will ultimately converge.

From a practical standpoint, our purpose is to suggest an algorithm which has the potential of putting neural network tools on the same basis as other commonly used "black box" numerical procedures (such as singular value decomposition, or the transcendental functions provided by commercial program libraries) that are expected to produce good results over a wide range of inputs with little or no intervention on the part of the user.

The organization of the remainder of this paper is essentially bottom-up. Section III presents the RBF network formulation, training procedure, and fitness function. Section IV then shows how such an RBF network is created from a space-filling curve, which in turn is created from a list of genetically-encoded parameters. These genetically-encoded parameters evolve by means of the genetic operators discussed in Section V. Section VI gives the procedure that was used to test the approach on a standard benchmark problem. The results are discussed in Section VII.

III. RBF Network Fitness Evaluation

The radial basis function network [22]–[25] produced from each genetic string consists of two layers—one layer with as many nodes as there are basis functions, and a second layer with a single node that produces a scalar output. (More generally, RBF networks can be formulated with more than one output node, in which case each output node has its own weight vector that is trained independently of the others.) The m nodes of the first layer are indexed by $j = 1, \cdots, m$. The node indexed by j implements an RBF unit for which a width σ_j and a center μ_j must be specified. Each vector μ_j, with coordinates $\mu_{jk}, k = 1, \cdots, n$ represents a point in the n-dimensional input space, this point being the center of the RBF. A given input vector $\mathbf{x} = x_1, x_2, \cdots, x_n$ produces an activation y_j for each node of the first layer. This activation represents the value of the Gaussian radial basis function

$$y_j = \exp\left[-\sum_{k=1}^{n} \frac{(x_k - \mu_{jk})^2}{2\sigma_j^2}\right] \tag{1}$$

for the given input. This basis function is radially symmetric with a single maximum at μ_j. The function drops off rapidly to zero at large radii. The result is a node in the first layer that responds to a local portion of the input space determined by the center μ_j and width σ_j. The vector of RBF activations $\mathbf{y} = (y_1, \cdots, y_m)$ is then multiplied by the weight vector $\mathbf{w}$ to obtain the scalar output z.

In an RBF neural network, the centers μ_j and widths σ_j are normally fixed, and only the components of the weight vector $\mathbf{w}$ change during training. An RBF network therefore requires some strategy for selecting appropriate centers and widths. The most common method of selecting the centers is to apply a clustering algorithm such as k-means clustering to the data points in the training set [25]. Once the RBF centers have been selected, appropriate widths may be determined using various k-nearest-neighbor heuristics [25]. More elaborate clustering and width-determination schemes that take into account the class membership of each training example have also been proposed [30].

In our evolutionary approach, by contrast, the centers and widths are not fixed algorithmically, but are specified by a genetic encoding that evolves in a population over the generations of a genetic algorithm. The RBF centers μ_j are determined from the space-filling curve described in the next section, which in turn is determined from the genetic string. The values of the widths σ_j are parameters directly encoded in the genetic string. Each genetic string in the population thus indirectly determines a different choice of centers and widths. Each such choice must be evaluated within the genetic algorithm.

To evaluate the fitness of a given genetic string, the RBF centers and widths are determined from the parameters encoded in the string. The weight vector $\mathbf{w}$ is then adjusted to minimize the mean-squared error over the training set, using the LMS gradient-descent learning rule [29] $\Delta w_j = E \cdot \alpha \cdot y_j$ where Δw_j is the change in the weight w_j from one time step to the next, α is the learning rate determined by the GA, and E is the prediction error for the training example presented at the current time step.

The training procedure employed within the GA indirectly rewarded the use of the minimal number of RBF's to achieve a given level of network performance. This was done by placing a fixed ceiling (300 in our experiments) on the product of the number of RBF's used and the number of passes allowed

through the training data. Thus, a network using fewer RBF's would have more training passes in which to reduce the error and thereby increase fitness. In most cases, fewer than the maximum number of units were actually allocated by the most fit genetic strings evolved by the GA. The LMS rule with the number of passes through the data constrained in this manner was used to quickly compute a good approximation to the error that would result from an optimal fit. LMS learning was used within the evolutionary process only.

A fitness value is assigned to each genetic string by the GA based on the normalized error over the training set when training is complete. (Normalized error is defined as the root-mean-squared error in approximating z over the training set divided by the standard deviation of the set of correct outputs.) The expected number of copies of each genetic string produced by selection will be proportional to this fitness value. For the GA to work well, it is desirable for the expected number of copies of the better strings in the population to be significantly higher than the expected number of copies of strings of average performance [31]. We observed that if fitness is defined simply as the reciprocal of normalized error, this desirable property is lost as the population becomes more uniform in fitness. There are two main approaches to solving this well-known fitness scaling problem [32]: either defining fitness in terms of a mathematical function that is rescaled at each generation, or rank-ordering the population and defining fitness from these rank orders. To preserve the metric information provided by the normalized error measure, we chose the former approach. Since our purpose was to investigate the encoding of a neural network specification rather than the mechanics of the GA itself, we chose an admittedly heuristic fitness scaling that was observed to give the better strings a significant reproductive advantage over the life of the GA. The normalized error E computed for each string in each generation was linearly rescaled to

$$E_{\text{rescaled}} = \cdot\frac{E - E_{\min}}{(1 + E_{\text{avg}})/2 - E_{\min}} \tag{2}$$

where $E_{\min}$ is the minimum normalized error over the strings in the current generation and E_{avg} is the average normalized error. This linearly rescales the error to a scale where a value of zero is assigned to the best string and a value of one is assigned to a hypothetical mediocre string whose error is midway between that of an average string and that of a constant mean prediction. (Always predicting the mean yields a normalized error of 1.) Equation (2) represents a compromise between scaling to an absolute error (in which the denominator of equation (2) would be $1 - E_{\min}$ and the hypothetical mediocre string would always predict the mean) and a relative scale (in which the denominator of equation (2) would be $E_{avg} - E_{min}$ and the hypothetical mediocre string would be a string of average fitness). The fitness assigned to each string is then $(s+1)/(E_{\text{rescaled}} \cdot s + 1)$ where s is a constant (set to 20 in our experiments). The parameter s represents selection pressure in the sense that the best string in the current generation is expected to receive $(s + 1)$ times as much representation in the next generation as the hypothetical mediocre string defined above.

IV. Use of the Space-Filling Curve

To reduce the number of degrees of freedom of the genetic encoding, the centers of the basis functions are not encoded directly. Instead, the genetic encoding of each RBF network specifies a small set of space-filling curves. Given a limited number of available RBF's, space-filling curves appear to yield better coverage of the relevant portions of the input space than simply generating the RBF centers randomly from some specified or genetically-encoded probability distribution. For reasons of computational efficiency, the form of space-filling curve used was the dovetail algorithm [33], which was originally proposed to perform numerical integration.

The curve can be expressed as a mapping from one dimension to n dimensions, where in our case n is the number of dimensions of the input space of the neural network. Computationally, the resolution of the mapping is determined by the number of bits used to encode the one-dimensional domain of the function, and it is convenient to make this number of bits an integral multiple m of the number of dimensions n, so that the domain of the mapping is the set of all mn-bit binary fractions in [0,1). This binary fraction represents the percentage of distance travelled around the curve. The space-filling mapping converts this binary fraction into the n coordinates of the point this far around the curve. The binary fraction in [0,1) representing coordinate i is obtained by selecting and concatenating bit positions i modulo n from the original binary fraction. Fig. 1a shows an example of a space-filling curve in two dimensions, with 4 bits of precision in each coordinate. Such a curve uniformly fills the unit hypercube of dimension n. The coordinate system of the range of the mapping can easily be transformed to convert this uniform density into a nonuniform density, as indicated below, and to map [0,1) into the actual range of each input variable of the training data.

The genetic encoding of each RBF network specifies a small set of space-filling curves. Each curve in the set occupies a region in the input space specified by a genetically encoded region center and region radius. Once the mapping that defines the curve is decoded, the centers of individual RBF's are spaced along this curve at equal intervals. This placement begins at a genetically encoded initial offset along the curve. A two-dimensional illustration of this is shown in Fig. 1b.

For example, in the GA simulations reported below, n is 4 because there are 4 input variables, and m is set to 8 so that mn is 32, the word size of the workstation on which the software is implemented. Then to generate a cluster of, say, 21 centers distributed within the unit volume in 4 dimensions, we begin by choosing 21 unsigned 32-bit integers $a_1, \cdots, a_{21}$ which are equally spaced beginning at a genetically-encoded initial offset a_1 and spaced so that $a_{j+1} = a_j + 2^{32}/21$, modulo 2^{32}, where the quotient $2^{32}/21$ is rounded to the nearest integer. Since n is 4, let us express each integer a_j in hexadecimal notation, where each hex digit corresponds to 4 adjacent bits in the binary representation. Each a_j will then consist of 8 hex digits. Each hex integer a_j is then translated into a 4-dimensional point each of whose coordinates is an 8-bit unsigned integer. The 8 bits of the first coordinate are the highest-order bits

of the 8 hex digits in a_j, the 8 bits of the second coordinate are the second bits of each hex digit, the 8 bits of the third coordinate are the third bits of each hex digit, and the 8 bits of the fourth coordinate are the low-order fourth bits of each hex digit in a_j.

The resulting points in the unit hypercube are then transformed to convert the uniform distribution of the space-filling curve into a Cauchy distribution centered at $(\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2})$. Each coordinate x_j of each point generated on the space-filling curve is transformed by applying the inverse of the cumulative Cauchy distribution function $\tan[\pi(x_j - \frac{1}{2})/(1 + \beta)]$ to it, where β in (0,1) is a genetically encoded shape parameter determining the sharpness or flatness of this Cauchy distribution. The net effect therefore is to spread the set of RBF's over the genetically-encoded region with a probability density greater near the center of the region, in accordance with the Cauchy distribution. The Cauchy distribution was chosen because its density has this property and because the inverse of its cumulative distribution is straightforward to compute. The detailed layout of the RBF's is thus determined by the following list of genetically encoded parameters for each curve in the set:

- Center of the region to be filled by the curve (n coordinates)
- Radius of the region
- Fraction of total available RBF's allocated to this region
- Shape parameter β of Cauchy distribution used to distribute the RBF's
- Offset a_1 from the beginning of the curve to the first RBF center
- Width σ_j, applicable to each RBF in the set

This list of parameters is repeated for the maximum number of space-filling curves among which the available RBF's are to be apportioned. In the simulations reported below, up to 12 such curves could be encoded on each genetic string. Thus, each genetic string could allocate the available RBF's in a network among up to 12 regions of the input space, yielding the opportunity to allocate RBF's more densely in regions where the function to be approximated is less smooth.

RBF's were allocated in this way within a constraint fixed by the experimenter on the maximum number of RBF's allowed in a network. Two additional parameters applicable to the entire set of RBF's were encoded on the genetic string:

- Degree of truncation of the tails of the Gaussian basis function
- Learning rate α used by the LMS gradient descent algorithm

The degree of truncation of the Gaussian tails was specified as an activation threshold below which the response of an RBF would be set to zero. The same threshold was applied to all RBF's in the network encoded by a given string. The effect of this threshold is to truncate the tails of the Gaussian function of equation (1) above, so that the response of each RBF is zero beyond a certain radius from its center. Including this genetically-determined truncation was observed to dramatically speed up training of the RBF networks generated by the GA. This was because, with truncated Gaussians, each example in the training set elicits a non-zero response in only a small fraction of the RBF units, and only the connections leading from these active RBF's need to have their weights adjusted. This truncation was a computation-saving heuristic employed during the evolutionary process only, where training occurs in an inner loop. After the completion of the evolutionary process, the single most fit network produced by the GA was subject to a more computationally intensive evaluation in which the full Gaussian distribution was used for each RBF.

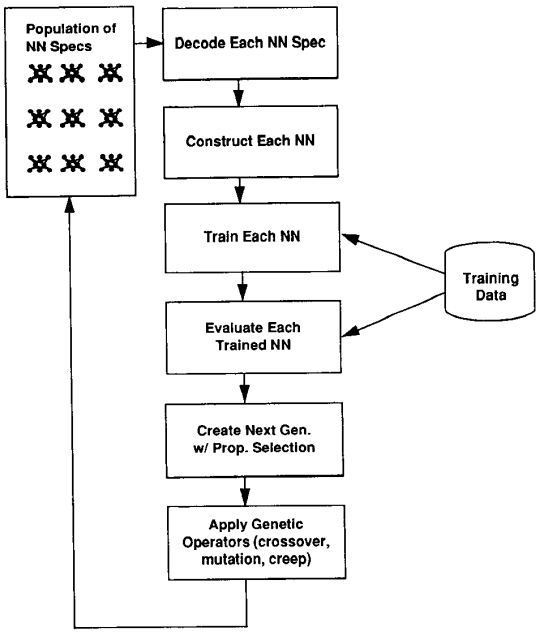


Fig. 2. One generation of the GA. Each string in the genetic population is evaluated by decoding it into a radial basis function (RBF) network with genetically-determined RBF centers and widths. The weights from the RBF nodes to the output node are then trained and the resulting normalized error translated into the fitness evaluation of the given string. These evaluated strings are then selected in proportion to their fitness to construct the next generation population. The new population is then modified using the genetic operators of crossover, mutation, and creep.

V. Genetic Encoding and Algorithm

A binary genetic encoding was used, with 8 bits of precision allocated to each of the encoded parameters listed in the previous section. During each generation of the GA, genetic selection was carried out by translating each string into the specified distribution of RBF's and then training the RBF network. Fitness was determined from the mean-squared error over the training set at the completion of training, as described in Section III above. These steps are summarized in Fig. 2.

Roulette-wheel proportional selection was based on the fitness measure described in Section III above. The entire population was replaced in each generation. Two-point crossover was used, with crossover points selected according to a compromise between the binary-encoding philosophy and the real-valued encoding philosophy [34]. Given a crossover to be performed, there was a 50% probability that the point at which crossover begins would be constrained to fall on the boundary between different 8-bit encodings of parameters. The other

50% of the time, the beginning crossover point was a randomly determined bit position, and thus likely to fall within the 8-bit genetic encoding of some parameter. The ending crossover point was never constrained to a parameter boundary. Our motivation for giving slight preference to exact parameter boundaries was to promote the efficient recombination of 8-bit units representing distinct parameters of the RBF. Bit-level recombination was still allowed, however, to prevent the possible blocking effects of a purely real-valued genetic alphabet [34].

The unit of application of the mutation operator was the 8-bit unit that represents each genetically encoded parameter. Each such parameter encoding produced by selection was altered with probability equal to the specified mutation probability. The alteration was to a new 8-bit encoding drawn from a uniform random distribution. A finer-grained phenotypic creep operator was also implemented. Its unit of application was also the 8-bit parameter encoding, with probability equal to the specified creep probability. When the creep operator is applied, the 8-bit encoding is decoded as an unsigned integer; this integer is either increased by one or decreased by one, then the resulting integer is encoded back into the affected 8-bit unit. Our motivation for using this operator was to allow fine-tuning of the most fit strings as the GA approaches convergence.

The overall algorithm, which is summarized in Fig. 2, has the following structure:

```
Randomly initialize the population of
  genetic strings.
For each generation of the GA:
   For each string in the population
    (which specifies an NN):
   -- Decode the NN
        specified by the string:
          Decode learning rate and
            truncation of Gaussian tails.
          For each space-filling curve
            encoded within the string:
              Decode the parameters of the curve.
              Generate equally-spaced RBF
                centers along the curve.
              Decode and apply the width
              parameter to each RBF.
   -- Construct the resulting RBF network.
   -- Train the RBF network:
          For each pass through
            the training data:
              For each example in
                the training set:
              Compute the activation
                of each RBF.
              Compute the output
                from each output node.
              Determine the error of
                the network's output.
              For each RBF with
                nonzero activation:
                 Modify the weight from
                   that RBF to the output.
   -- Evaluate the RBF network:
          Compute the normalized error
            of the trained network.
          Translate this into the
            fitness of the genetic string.
Create next generation using proportional selection.
Apply crossover, mutation, and creep genetic
operators.
```

The innermost loop of this algorithm is the calculation of each weight change within the LMS training procedure. Even with the heuristics to speed up training by truncating the RBF's and constraining the number of passes through the training data, in the simulations reported below, over 98% of the computation of the GA was consumed by the evaluation function.

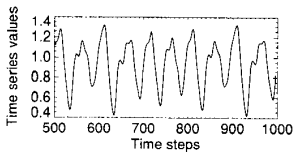


Fig. 3. Sample of the Mackey-Glass time series.

VI. Procedure

The time series generated by the Mackey-Glass [35] differential equation

$$\frac{dx(t)}{dt} = -bx(t) + a \cdot \frac{x(t-T)}{1 + x(t-T)^{10}} \tag{3}$$

is a well-recognized benchmark for evaluating neural network performance. Following previous research [25], [36]–[40], we generated this time series using the parameters $a = 0.2$, $b = 0.1$, and $T = 17$. As in most previous work, the training data consisted of 500 randomly selected points in the time series, and the test data consisted of 500 points in sequence following the training data. The 500 training examples were randomly chosen from points 500 to 4000 of the time series, and the test set began at point 4000. A sample of the time series is shown in Fig. 3.

As in the work cited above (and as discussed in [36]), 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 GA was executed on the above task with the maximum number of RBF units set to 50, 100, and 150, and the most fit RBF network evolved by each GA run was saved. For each choice of maximum network size, the GA was run three times using a different seed for the random number generator, and the results of these three runs were averaged. In each case, the GA was free to allocate fewer than the maximum number of RBF's; the actual numbers of RBF's in the evolved networks averaged 39 when the constraint was 50, 69 when the constraint was 100, and 110 when the constraint was 150.

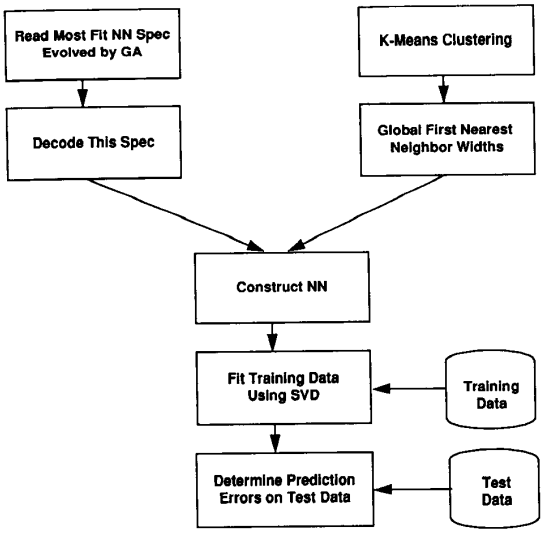


Fig. 4. Final evaluation of the best RBF network evolved by the GA in comparison to an RBF network of equal size produced by k-means clustering. In both cases, singular value decomposition is used to linearly fit the training data to the specified basis functions. Generalization performance is then evaluated by computing the normalized prediction error over a test data set different from the training set.

For all simulations of the genetic algorithm, the following parameters were used:

Population size	100
Probability of mutation (per 8-bit unit)	.01
Probability of creep (per 8-bit unit)	.05
Probability of crossover (per mating)	.50
Maximum number of space-filling curves encoded in each genetic string	12

Each GA run resulted in a single RBF network (the one with the highest fitness), consisting of a specification of RBF centers and widths. We wished to compare the quality of the networks produced by the GA with the quality of networks of equal size produced by k-means clustering [25], the method most commonly used to determine RBF centers and widths. Accordingly, for each run of the GA, a network with the same number of basis function units was produced using k-means clustering. We employed the k-means clustering algorithm used by [25] for the same task of applying Gaussian RBF's to the Mackey-Glass time series prediction task. RBF unit widths were then determined using the global first nearest neighbors algorithm used by [25] for this task.

Like the GA, k-means clustering with this width-determination algorithm results in a specification of RBF centers and widths. The network specifications produced by the GA and those produced by the k-means method were each evaluated using singular value decomposition to obtain an optimal least-squares linear fit of the specified set of basis functions to the Mackey-Glass training data. Each optimally-trained RBF network was then presented with the test set (beginning at point 4000 of the time series) and the normalized prediction error over the test set was determined. This assessment of generalization performance for RBF's produced by the GA and by k-means clustering is shown in Fig. 4.

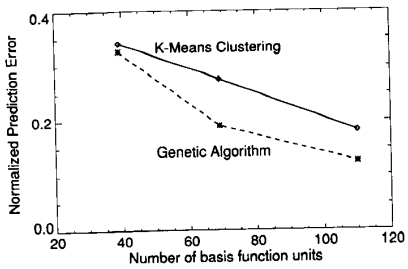


Fig. 5. Normalized prediction error achieved by radial basis function networks on a sample of the Mackey-Glass time series later than the training set. Asterisks connected by dashed lines indicate results for networks evolved by the genetic algorithm; diamonds connected by solid lines indicate results for networks of corresponding size produced by k-means clustering. Each point plotted represents the average of three different networks.

VII. Results and Discussion

Figure 5 compares the prediction accuracy on the test set for the two algorithms with identical numbers of first-layer units. Each point plotted for the GA method represents the average of 3 GA runs, and each point plotted for the k-means method represents the average of 3 k-means runs with numbers of basis function units corresponding to the GA runs. The results shown for the k-means method are consistent with the relationship between network size and normalized error plotted by [25].

At least for the two larger network sizes studied, RBF networks produced by evolving space-filling curves appear to have better generalization performance on this task than corresponding networks produced by k-means clustering. While the difference between methods appears greater for the two larger network sizes, further study is needed before any conclusions can be drawn about the scaling properties of the evolutionary method.

In our prototype implementation, each GA run (which was limited to one billion executions of the single weight change in the innermost loop of the algorithm) took approximately 35–40 times as much CPU time as the corresponding k-means algorithm. We believe that the GA can be made substantially more efficient, but there is little doubt that it will remain slower in any serial implementation. While the evolutionary process itself is slow, the RBF network it produces is evolved to be computationally efficient in the sense of producing the best approximation using the fewest units. The evolutionary approach might therefore be useful for those practical applications where the main concern is run-time performance of the trained network.

Further work is needed to compare this evolutionary approach with deterministic algorithms for placing basis functions, such as orthogonal least-squares techniques and clustering algorithms. The orthogonal least-squares technique [41] starts with a set of basis-function building blocks consisting of one RBF for each training example, centered on that training example. The higher-order basis functions used in the network are fixed linear combinations of these building-block basis

functions. They are generated one at a time using the Gram-Schmidt orthogonalization procedure. This guarantees that the higher-order basis functions are mutually orthogonal (in the sense that they generate mutually uncorrelated activations over the training set), and that each is selected to maximally reduce the residual error in the function approximation performed by the network. The resulting function approximation is likely to be more costly to compute than the RBF approximation produced by the GA, however. While the function approximation produced by the GA evolves to require as few RBF's as possible, the set of higher-order basis functions constructed by the Gram-Schmidt process could in general depend on the entire set of building-block RBF's. Function approximation could therefore be costly, given that there are as many building-block RBF's as training examples.

Clustering algorithms more elaborate than the k-means algorithm [25] used herein have been proposed that take into account the class membership of each training example [30], [42]. Class membership information may also be used to select RBF widths to provide smooth coverage within a given class while minimizing overlap between the basis functions representing different classes [30]. This type of clustering is better suited to classification tasks than to continuous function approximation, however.

One drawback of clustering techniques for locating RBF centers is that, even for local basis functions like the Gaussian RBF's we use, there is no guarantee that the optimal set of RBF centers lies within the convex hull of the training data [27]. This is one possible explanation for the observed superiority of the RBF networks produced by the GA over those produced by k-means clustering in the results above. A large proportion of the RBF centers produced by the GA were observed to lie outside the convex hull of the training data, while the centers produced by k-means clustering stayed within the hull as expected.

The GA's ability to evolve RBF centers outside the convex hull of the training data may become particularly important if we consider non-local basis functions such as the multiquadric $(r^2 + c^2)^{1/2}$, or the thin-plate spline $r^2 \log r$ where r is the radius from a given RBF center and c is a constant. Evolutionary algorithms employing these non-local RBF's in place of the Gaussian RBF merit further study because of the superior theoretical interpolation properties of these non-local RBF's. For several choices of basis functions, including the multiquadric and thin-plate spline, but *not* the Gaussian, interpolation properties of regularly-spaced grids of RBF's actually improve as the number of dimensions increases [24]. The multiquadric and thin-plate spline have also been used effectively with additional techniques such as blended projections from spherical to planar domains of the function to be approximated [43] and moving least squares methods [44] to produce local approximations. All of these results suggest that non-local RBF's might prove superior to Gaussian RBF's in the evolution of sets of RBF's, and that an evolutionary approach might be better suited than clustering techniques to the placement of these non-local RBF's.

Evolving a set of space-filling curves to distribute radial basis functions differs in a fundamental way both from conventional neural network training and from the evolution of specific connection weights. The difference is that the parameters of the space-filling curve are adapting a *group* of RBF's to a *region* of the input space. By contrast, both weight-adapting GA's and most neural networks allow each first-layer node its own independent degrees of freedom, so that each can change independently of the others.

By restricting these degrees of freedom, our approach forces a group of first-layer nodes, taken as a whole, to adapt to the properties of a region of the input space, taken as a whole. The validity and generalizability of the resultant trained networks depend upon the hypothesis that the parameters of a space-filling curve can capture relevant properties of a region of the input space. The results reported herein appear to offer preliminary support for this hypothesis.

ACKNOWLEDGMENT

We thank the reviewers for their valuable suggestions.

REFERENCES

[1] X. Yao, "A review of evolutionary artificial neural networks," *International Journal of Intelligent Systems*, vol. 8, no. 4, pp. 539–567, 1993.

[2] 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. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 391–396.

[3] 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. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 370–374.

[4] 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. San Mateo, CA: Morgan Kaufmann, June 1990, pp. 132–139.

[5] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, August 1989, pp. 762–767.

[6] H. Kitano, "Empirical studies on the speed of converengence of neural network training using genetic algorithms," in *Proceedings of the Eighth National Conference on AI (AAAI–90)*, MIT Press, July–August 1990, pp. 789–795.

[7] 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*, IEEE, July 1991, pp. II-71–II-76.

[8] 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*, IEEE, July 1991, pp. II-397–II-404.

[9] 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. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 379–384.

[10] 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. Cambridge, MA: MIT Press, 1990, pp. 244–248.

[11] 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. Cambridge, MA: The MIT Press, 1990, pp. 249–256.

[12] 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.

[13] S. A. Harp, T. Samad, and A. Guha, "Designing application-specific neural networks using the genetic algorithms," in *Advances in neural*

Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 447–454.

[14] S. A. Harp and T. Samad, "Genetic optimization of self-organizing feature maps," in *IJCNN-91-Seattle: International Joint Conference on Neural Networks*, IEEE, July 1991, pp. I-341–I-346.

[15] S. A. Harp, T. Samad and A. Guha, "Towards the genetic synthesis of neural networks," in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 360–369.

[16] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, pp. 461–476, 1990.

[17] J. W. L. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Networks*, vol. 4, pp. 53–60, 1991.

[18] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybermetics*, vol. 63, pp. 487–493, 1990.

[19] H. Barun and J. Weisbrod, "Evolving neural networks for application oriented problems," in *Proceedings of 2nd Annual Conference on Evolutionary Programming*, D. B. Fogel and W. Atmar, Eds. La Jolla, CA: Evolutionary Programming Society, 1993, pp. 62–71.

[20] D. B. Fogel, "Using evolutionary programming to create neural networks that are capable of playing tic–tac–toe," in *International Conference on Neural Networks*, 1993, pp. 875–880.

[21] J. R. McDonnell and D. Waagen, "Neural network structure design by evolutionary programming," in *Proceedings of 2nd Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar, Eds. La Jolla, CA: Evolutionary Programming Society, 1993, pp. 79–89.

[22] 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. Royal Military College of Science: Oxford University Press, 1987, pp. 143–167.

[23] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

[24] M. J. D. Powell, "The theory of radial basis function approximation in 1990, volume II," in *Advances in Numerical Analysis*, W. Light, Ed. Oxford: Clarendon Press, 1992, pp. 105–210.

[25] J. Moody and C. J. Darken, "Fast learning in networks of locally–tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.

[26] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, 1990, pp. 1481–1497.

[27] 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.

[28] C. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, vol. 3, no. 4, pp. 579–588, 1991.

[29] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 WESCON Convention*, 1960, pp. 96–104.

[30] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M . Hummels, "On the training of radial basis function classifiers," *Neural Networks*, vol. 5, pp. 595–603, 1992.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press, 1975.

[32] D. Whitely, "The genitor algorithm and selection pressure: Why rank–based allocation of reproductive trials is best," in *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 116–121.

[33] E. A. Patrick, D. R. Anderson, and F. K. Bechtel, "Mapping multidimensional space to one dimension for computer output display," *IEEE Transactions on Computers*, vol. C–17, no. 10, pp. 949–953, 1968.

[34] D. E. Goldberg, "Real-coded genetic algorithms, virtual alphabets, and blocking," *Complex Systems*, vol. 5, pp. 139–167, 1991.

[35] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.

[36] A. Lapedes and R. Farber, "How neural nets work," in *Neural Information Processing Systems*. New York: American Institute of Physics, 1988, pp. 442–456.

[37] J. D. Farmer and J. H. Sidorowich, "Predicting chaotic time series," *Physical Review Letters*, vol. 59, no. 8, pp. 845–848, 1987.

[38] M. F. Tenorio, "Self-organizing network for optimum supervised learning," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 100–110, 1990.

[39] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, pp. 213–225, 1991.

[40] E. Hartman and J. D. Keeler, "Predicting the future: Advantages of semilocal units," *Neural Computation*, vol. 3, pp. 566–578, 1991.

[41] 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.

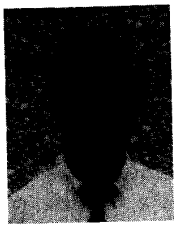
[42] T. Kohonen, "An introduction to neural computing," *Neural Networks*, vol. 1, no. 1, pp. 3–16, 1988.

[43] T. A. Foley, "The map and blend scattered data interpolant on a sphere," *Computers Math. Applic.*, vol. 24, no. 12, pp. 49–60, 1992.

[44] K. Salkauskas, "Moving least squares interpolation with thin-plate splines and radial basis functions," *Computers Math. Applic.*, vol. 24, no. 12, pp. 177–185, 1992.

Bruce A. Whitehead received the B.S. degree in psychology and the M.S. and Ph.D. degrees in Computer and Communication Sciences from the University of Michigan in 1977 for a neural network model of human pattern recognition.

He is currently an Associate Professor of Computer Science at the University of Tennessee Space Institute in Tullahoma, TN. He has held positions at the University of Louisville, AT&T Bell Laboratories, and Drexel University. His current research interests include neural networks, applications of neural networks to propulsion system testing, and genetic evolution of neural networks.



Timothy D. Choate was born in Linden, Tennessee on May 28, 1964. He received the B.S. degree in computer science from Middle Tennessee State University in 1986 and the M.S. degree in computer science from the University of Tennessee Space Institute in 1991.

From 1986 to 1992 he worked for OAO Corporation at Arnold Engineering Development Center in Tullahoma, TN, where he was involved in diverse software development projects, including mainframe applications development, CASE tool development, neural network design, virtual reality environments, operations research, and software development process management.

From 1992 through the present he has been employed at ERC Incorporated, where his primary mission is the design of neural network architectures for function approximation applications. His research interests include complex systems, intelligent systems, and simulated environments.