Evolving predictors for chaotic time series

Peter Angeline

Peter J. Angeline, "Evolving predictors for chaotic time series," Proc. SPIE 3390, Applications and Science of Computational Intelligence, (25 March 1998); doi: 10.1117/12.304803

Event: Aerospace/Defense Sensing and Controls, 1998, Orlando, FL, United States

Evolving Predictors for Chaotic Time Series

Peter J. Angeline
Natural Selection, Inc., 509 Colgate St., Vestal, NY 13850

ABSTRACT

Neural networks are a popular representation for inducing single-step predictors for chaotic times series. For complex time series it is often the case that a large number of hidden units must be used to reliably acquire appropriate predictors. This paper describes an evolutionary method that evolves a class of dynamic systems with a form similar to neural networks but requiring fewer computational units. Results for experiments on two popular chaotic times series are described and the current method's performance is shown to compare favorably with using larger neural networks.

Keywords: evolutionary computation, evolutionary programming, genetic programming, neural networks, chaotic time series prediction

1. INTRODUCTION

What once were thought to be random, unpredictable sequences in science, technology, and nature are now newly identified as complex but yet deterministic and consequently predictable. Chaos, the science of non-linear systems, has provided new tools and understanding that permits modeling important processes that were previously thought to be unmodelable. Chaotic time series prediction studies the application of these techniques to the induction of models for pseudo-random sequences such as the stock market, complex interference patterns, control systems and many others.[1, 2]

Often, neural networks are used to represent the induced prediction model. Neural networks provide both a general representation capable of modeling any function[3] and an automatic method for creating a suitable model in that representation.[4,5] The representation uses a set of units or "nodes" that each perform a computation that determines its activation. A node's activation function has the form:

$$\alpha_i = \sigma\left(\sum_j \omega_{ji}\alpha_j + \beta_i\right) \tag{1}$$

where α_i is the activation of node i, β_i is the bias for node i, ω_{ji} is the weight between node j and node i, and σ is a non-linear function which must be non-decreasing, semi-linear and differentiable everywhere for the training algorithm to applicable.[5] A common function used for σ is the sigmoid function given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

A subset of the nodes in a network are designated as the output nodes, the remainder being identified as intermediate or hidden units. Occasionally, the sigmoid function is not applied to the summation performed at the output nodes permitting the output units to be linear functions of their inputs.

The training algorithm dictates the properties of the activation function for all nodes to ensure the network's weights can be modified using gradient descent. Iterations of the training algorithm eventually approximate the desired function within some tolerance. When the activation function used in a neural network does not supply sufficient non-linearity to succinctly model a desired function, neural network models must use large numbers of nodes often placed into multiple layers or, alternatively, add additional input information to reduce the need for complex intermediate computations. Such networks are commonplace when modeling chaotic time series.[1] Ideally, a training method that creates activation functions appropriate for the

Additional author information:
Email: angeline@natural-selection.com
WWW: http://www.natural-selection.com

task at hand should minimize the number of nodes, hidden and input, required to adequately model a given function including chaotic times series.

Evolutionary computations[6], search and optimization algorithms based on analogies to biological evolution, can be used to remove architectural restrictions from neural networks including the limitations on the activation function.[7,8] The work described in this paper investigates an evolutionary training method, called Multiple Interacting Programs (MIPs)[9] for a class of dynamic systems that are similar to neural networks. The ability of MIPs to induce appropriate dynamical systems with relatively few computational units is demonstrated on two well investigated chaotic times series prediction problems.

2. METHOD

Multiple Interacting Programs (MIPs) combine the training flexibility of evolutionary computations with the organization of a neural network.[9] In MIPs, a set of symbolic expressions are co-evolved with each referring to the other's outputs. Formally, a MIPs net, M, is defined by a 4-tuple $M = (\Sigma, \iota, \theta, \kappa)$ where the symbols denote the following sets:

- Σ is the set of functions associated with the problem. This corresponds to the function set used to create the expressions.
- ι is the set of input variables given by the problem definition. This corresponds to the set of input nodes for a neural network. It may include special terminals like $\Re$ which, when selected, returns a randomly generated value from a user specified range.
- θ is the set of output variables associated with the problem. This is equivalent to the set of output nodes in a neural network. Note that there can be several output variables associated with a problem all of which are computed by separate parse trees.
- κ is the set of intermediate variables to be used to construct solutions for the problem. These variables are equivalent to the set of hidden nodes in a neural network.

In a MIPs net, distinct symbolic expressions, stored as trees, are associated with each defined intermediate and output variable. This expression is called the variable's *update expression*. A variable's update expression, expressed in terms of Σ, ι, θ, and κ, stipulates the value its variable takes on when next evaluated.

Without loss of generality, assume that the output variables are identified as $T_0 \ldots T_n$ where $n+1$ is the number of output variables and the intermediate variables are identified as $T_{n+1} \ldots T_m$ where $m-n$ is the number of intermediate variables in the MIPs net. A MIPs net is executed by evaluating the update expression associated with each variable in turn, starting with T_m and ending with T_0. After a variable's expression is executed its value is stored and any future references to that variable in other update expressions will take on the newly stored value.

Define a *feed-forward MIPs net* to be a MIPs net such that the following holds for each variable:

$$\Gamma(T_i) = \Sigma \cup \iota \cup \{T_{i+1} \ldots T_m\} \tag{3}$$

where $\Gamma(T_i)$ is the language used to construct the update expression associated with variable T_i. Equation 3 stipulates that a feed-forward MIPs net limits a given variable's expression to reference only variables with a higher ordinal number. This is consistent with the order of execution of the update expressions so that each variable's value is restricted to a function of only the variables that have been previously updated. This prevents any circular variable references and is consistent with the definition of a feed-forward neural network.

A *recurrent MIPs net* is a MIPs net where the parse tree associated with any variable can refer to any other variable. Formally:

$$\Gamma(T_i) = \Sigma \cup \iota \cup \theta \cup \kappa \tag{4}$$

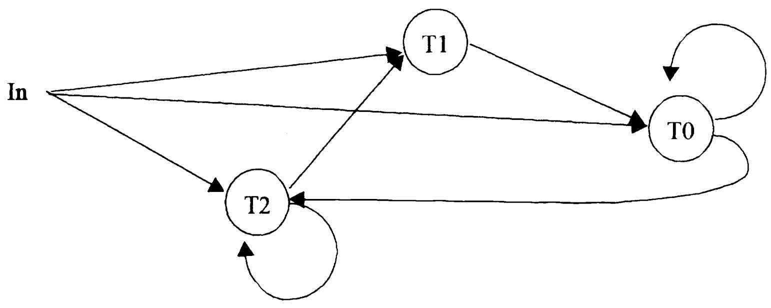


Figure 1: Connectivity of the MIPs network described by Equation 5. Each node corresponds to an expression in Equation 5 which determines that node's activation. Nodes are evaluated left to right.

which states that the language used to express how a variable is updated is the union of all defined language elements. This permits a variable's expression to reference the value of variables that have yet to be evaluated thus allowing circular references. Again, this definition is consistent with the definition of a recurrent neural network.

A MIPs net is most clearly displayed as a system of equations. For instance, a recurrent MIPs net of the form $\Sigma = \{+, /, -, *, \Re\}$, $\iota = \{\text{In}\}$, $\theta = \{T_0\}$, and $\kappa = \{T_1, T_2\}$ could be:

$$\begin{aligned} T_2 &= T_2 + 0.3745\text{In}(T_0 + T_1) \\ T_1 &= \frac{\text{In}}{T_2 + 0.97831}(T_0 - 0.11345) \\ T_0 &= (\text{In} + T_1)\text{T}_0 \end{aligned} \tag{5}$$

Equation 5 shows the update expressions for each of the intermediate and output variables in the network. As already described, this MIPs net would be evaluated starting with T_2 and ending with T_0 which is the designated output of this MIPs net. The references in T_2 to the other variables denote recurrent links in the MIPs net. Figure 1 shows the connectivity of the network described by Equation 5. When viewed in this manner, it is evident that the individual expressions in a MIPs net can be described as the activation functions of the various nodes, each node having a unique activation function. In the experiments below only feedforward MIPs nets are used.

When suitable to the problem, functions are applied to the outputs of the expressions before the variable's value is updated. Typically these take the same form as used in neural networks. For the experiments below, the sigmoid function defined in Equation 2 is applied to the result of a hidden node's update expression before being stored in the associated variable. This "squashing function" ensures that the variables will have a specific range, namely (0,1) but how those values are related to the inputs is dependent on the variable's update expression. For the experiments below, the squashing function is not applied to the output units however the output expressions may still be non-linear functions of their inputs by virtue of the functions in the function set.

Training MIPs Nets

Given that symbolic form are being manipulated, the prospect of using a standard gradient descent technique as is typically used when training neural networks must be discarded. A MIPs net is trained using an evolutionary program[6, 10] adapted from a previous evolutionary program that evolved recurrent neural networks.[7] The difference here is that rather than manipulating weights and arcs between nodes, the mutation operations are tailored to manipulate symbolic expressions in a system of equations.

Evolutionary programming is a form of evolutionary computation. Evolutionary computations can be succinctly described using the following equation:

$$x' = m(s(x, f(x))) \tag{6}$$

where x is a vector of candidate solutions called the *population*, f is a evaluation function, termed the *fitness functions*, that returns a vector of values corresponding to the fitness of each element of x as a solution to the task at hand, s is a *selection function* which takes the current population and their rankings and returns a set of copies of the pervious population, and m is the mutation function which randomly modifies the selected individuals. Evolutionary programming differs from most other evolutionary computations in that the representation selected to be most appropriate for the task, and the mutation operations used are representation specific and typically asexual, meaning they manipulate an individual in the population without borrowing content from another population member.

In order to induce the symbolic equations associated with a MIPs net, a set of mutations are required that can manipulate symbolic expressions. Seven mutations are used in the training algorithm for the experiments below:

- *insert element* - select an element in an expression in the MIPs net and insert a new language element above it making the current element one of its children. Select terminals for all other newly added arguments.
- *delete element* - select a language element that is not a terminal and remove it from the expression promoting its child to its position. If the element has more than one child then select one child at random and delete the rest.
- *cycle element* - select an element and change it to another element with the same number of arguments.
- *swap elements* - select a language element in an expression and exchange it with one of its siblings.
- *replace subtree* - select a subtree in an expression at random and replace it by a randomly generated subtree.
- *delete subtree* - select a subtree in an expression at random and replace it by a random terminal.
- *numeric terminal mutation* - Apply Gaussian noise of a user defined variance to a randomly selected numeric terminal.

When creating an offspring from a parent, the number of mutations to be applied is given by a Poisson random variable with a rate of two. Given that k mutations are to be applied, a copy of the parent is made and k mutation operations are selected uniformly from the above set with replacement and applied in turn to create the child. A limitation on the number of symbols allowed in the definition of an individual is set. In MIPs, a maximum number of symbols is defined for the entire network only, the network is free to distribute the symbols over the various expressions as needed. If a created child violates the maximum symbol limit then the child is discarded and the reproduction is nullified.

Selection is performed using probabilistic selection[6] as follows. For each individual in the population, k other population members are selected uniformly and a point is awarded for each that has worse fitness. The population is then sorted by this comparison score and the top half of the population is reserved as parents for the next generation. Each parent is allowed to create a single offspring, once again filling the population. For the experiments described below, $k = 7$.

The fitness of an individual in the population for the chaotic time series prediction tasks investigated here is the Normalized Mean Squared Error (NMSE) given by:

$$\text{NMSE}(x_i, T) = \frac{1}{\sigma_T^2 N_T} \sum_{k=1}^{N_T} (v(x_i, T, k) - \hat{T})^2 \tag{7}$$

where T is the time series, T_k is the kth element of that time series, $\hat{T}$ is the sample mean of the series, σ_T^2 is the variance of the series, N_T is the length of the series, $v(x_i, T, k)$ is the output of individual x_i when it needs to predict the value for T_k. This function is essentially the mean squared prediction error normalized by the series variance and has the property of returning 1.0 when the mean of the time series is consistently predicted.

3. EXPERIMENTS

Two experiments using two different chaotic time series were performed to demonstrate the ability of the described method to induce single-step predictors using comparatively few units. The first experiment modeled the familiar sunspot data while the

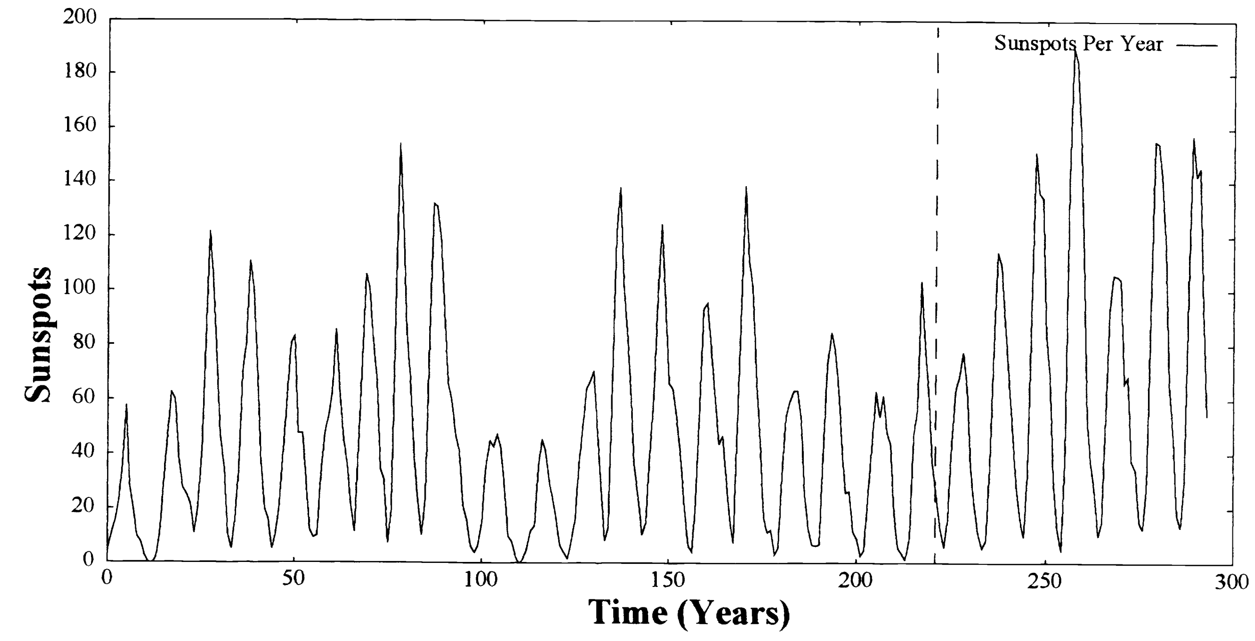


Figure 2: Average number of sunspots observed each year starting in the year 1700, when observations began, and ending in 1993.The dashed line marks the division between the training and test sets.

second investigated a chaotic times series previously used in a time series prediction competition. These time series were selected over computationally generated series (e.g. the Mackey-Glass equation) to ensure no knowledge of the underlying generation system could influence the results.

The Wolfe sunspot data* is a time series created from the number of sunspots observed each calendar year from the year 1700 to the present. Figure 1graphs the values from 1700 to 1993 used in the experiment. This data has been used in several other neural network prediction studies.[11, 12, 13] In these studies, the period from 1700 to 1920 was used as the training set while the period from 1921 to 1950 was used as the test set to determine the generalization ability of the induced model. The present study also investigates the long term ability of the induced model through an additional test period spanning the years from 1921 to 1993. The previous studies have used 12 input values and three hidden units to predict this time series. The 12 input values were the 12 values immediately preceding the value to be predicted. Given that the pseudo-periodic nature of the time series has an approximate cycle of just more than 11 years, this is a judicious choice. In the experiment here, only two hidden units and four input values are used. The four input values are d_1, d_2, d_4 and d_8 which are the values one year, two years, four years and eight years prior to the predicted value. Data values were normalized using a value of 200 which is slightly larger than the maximum of the time series. The combination of fewer input units and less cycle data make this a more difficult problem. The MIPs nets evolved for this experiment had the following form:

$$\begin{aligned} \Sigma &= \{+, /, -, *, \Re\} \\ \iota &= \{d_1, d_2, d_4, d_8\} \\ \kappa &= \{T_1, T_2, T_3\} \\ \theta &= \{T_0\} \end{aligned} \tag{8}$$

with the addition that the values of the hidden units were squashed using the sigmoid function as described above. No squashing function was applied to the output unit's activation.

The time series for the second experiment was originally offered as a test problem in a time series prediction competition.[1] The difficulty associated with this time series are the relatively rare "collapse" events occurring just twice in the training

* This data can be found on the WWW at http://www.ngdc.noaa.gov/stp/SOLAR/SSN/ssn.html

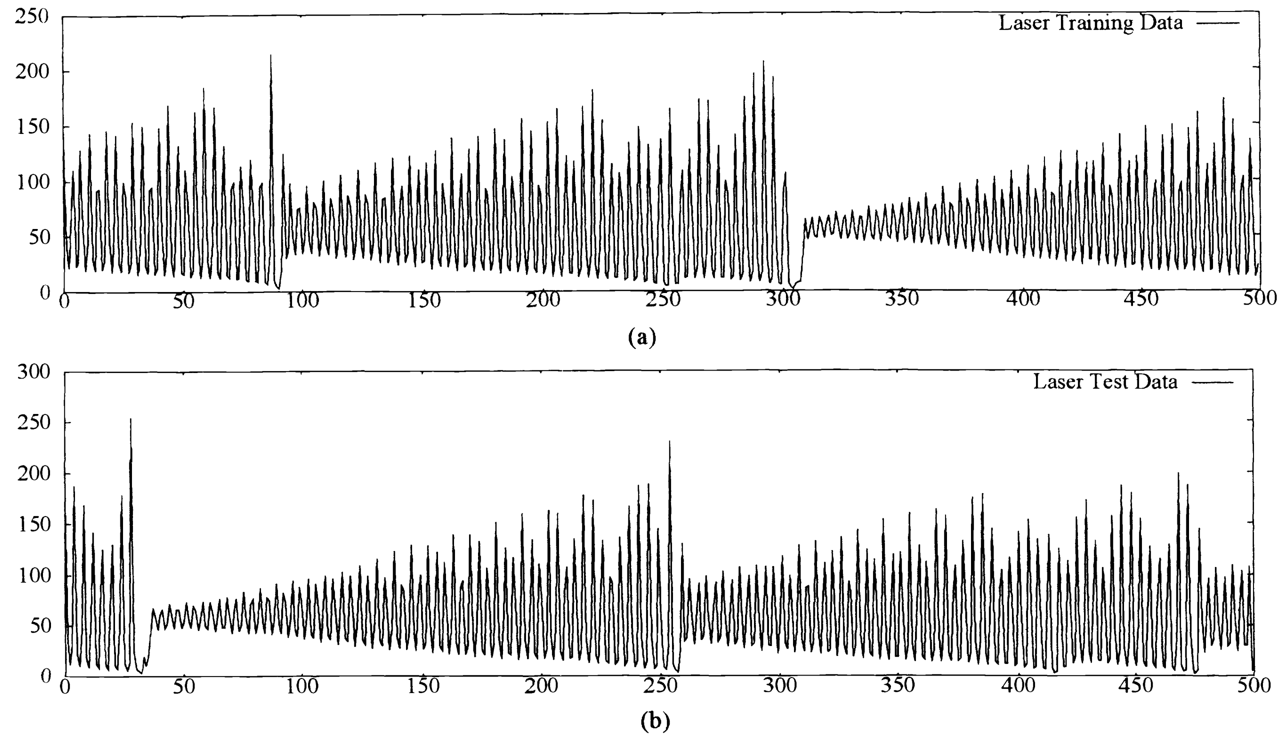


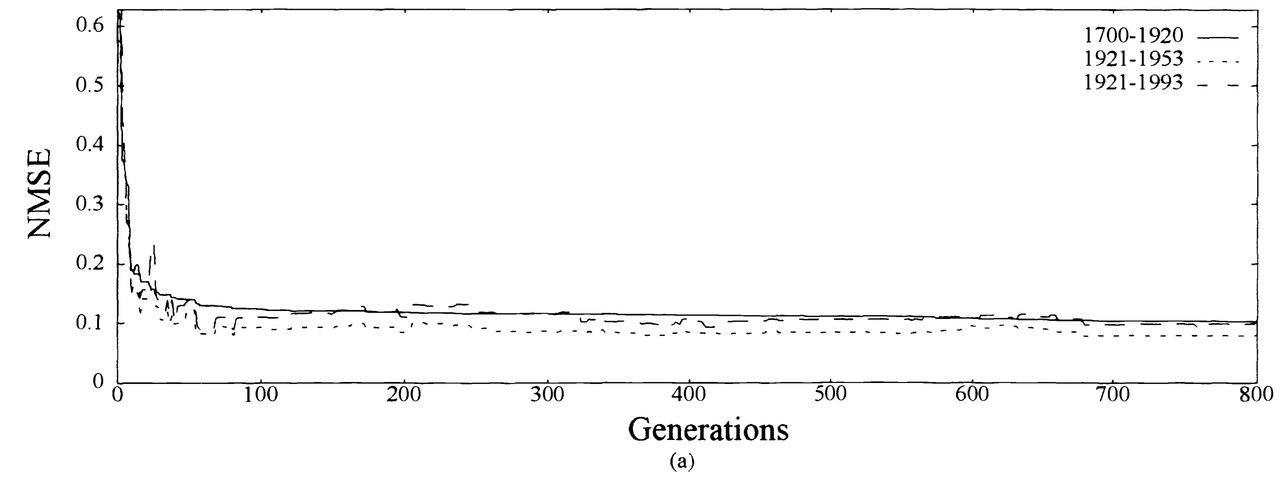
Figure 3: Data used for the second experiment generated by measuring the intensity of an NH_3 FIR laser. (a) Training set (b) Test set which is the continuation of the training set. Both data sets are the even positioned points from the original published data sets.

data and three times in the test data. Following Zhang et al.[14], the even points from the original time series were selected out and divided in half into a training set and a test set giving the data sets shown in Figure 3. The winning submission for this time series used a recurrent network that contained two different layers of hidden units each with 12 units giving a total of 24.[1] Zhang et al.[14], using an evolutionary method, created a network containing 19 units. Here, a maximum of four hidden units was permitted. Again following Zhang et al.[14], the three series values immediately preceding the value to be predicted were used as inputs to the network. The form of the MIPs networks evolved for this problem were similar to those of the previous experiment:

$$\begin{aligned} \Sigma &= \{+, /, -, *, \Re\} \\ \iota &= \{d_1, d_2, d_3\} \\ \kappa &= \{T_1, T_2, T_3, T_4\} \\ \theta &= \{T_0\} \end{aligned} \tag{9}$$

Note this form includes four hidden units and the three time series values just prior to the predicted value.

Other experimental parameters were as follows. For both experiments, the size of the population was 250 with 100 parents reserved between generations. This allowed the best 50 parents to create two children in the next generation while the worst 50 parents created only one. The sunspot experiment was allowed to run for 800 generations while the laser experiment ran for 900 generations. In order to observe the evolution of the generalization performance the NMSE for the test sets was monitored for the best individual for each generation. This information was not used in the algorithm it was simply recorded during the run to be presented below. Random numeric terminals were limited to the range (-1.0, 1.0). The size of all expressions in an initial population member was selected uniformly between five and 25 symbols and any randomly generated subtree thereafter was restricted to have ten symbols or fewer. All expressions were limited to a maximum of 40 symbols. No restrictions on the organization of the symbols within an expression, other than syntactic correctness, was imposed.



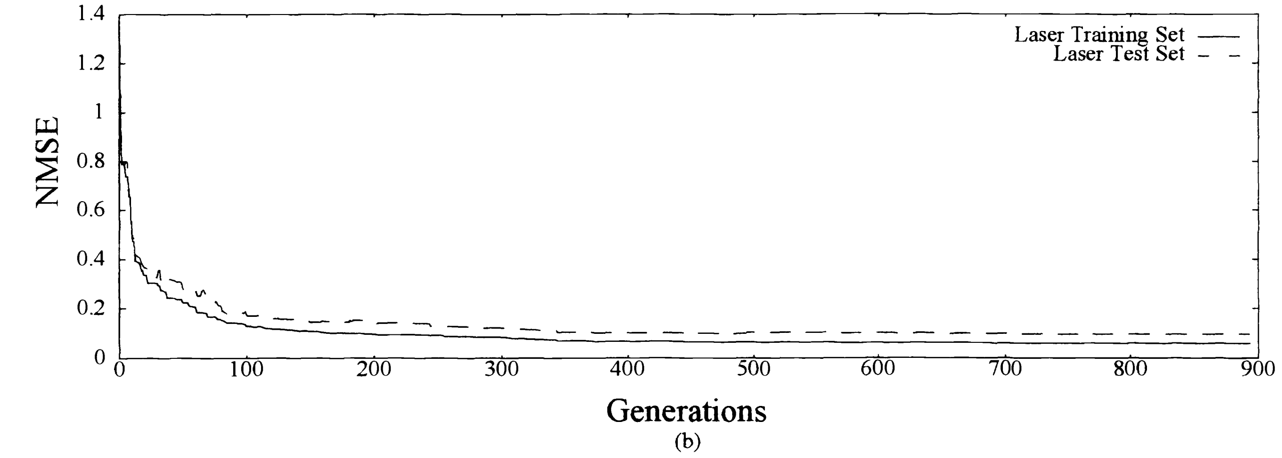


Figure 4: Results for the two experiments. (a) Results for the sunspot time series. The NMSE for the best of each generation on the training set (1700-1921) and two overlapping periods from the test set are shown. (b) Results for the laser times series. The NMSE for the training set and test set are shown.

4. RESULTS

Figure 4a shows the results of the first experiment for evolving a MIPs net for the sunspot data. The solid line in the graph represents the NMSE on the sunspot training set for the best found individual in each generation. Note that the graph drops rapidly for the first 100 generations and then levels out to a value of 0.103. The two dashed lines indicate the performance of the best individual at each generation for the two test sets. The short dashed line shows the performance on the test set from 1921 to 1953 while the long dashed line shows performance on the test set from 1921 to 1993. The jumping of the NMSE for the test sets is a result of the competition of various architectures evolving concurrently in the population, some of which generalize better than others. Table 1 shows a comparison of the result of this study and pervious studies on this training and test sets.

Table 1: Comparison of Results for Sunspot Data Experiment

Study	*Total Number of Nodes*	*Training Set NMSE*	*1921-1993 Test Set NMSE*
Weigend et al.	16 (12-3-1)	0.082	0.086
Svarer et al.	16 (12-3-1)	0.090	0.082
Aerrabotu et al.	16 (12-3-1)	0.100	0.106
MIPs	8 (4-1-1-1-1)	0.103	0.079

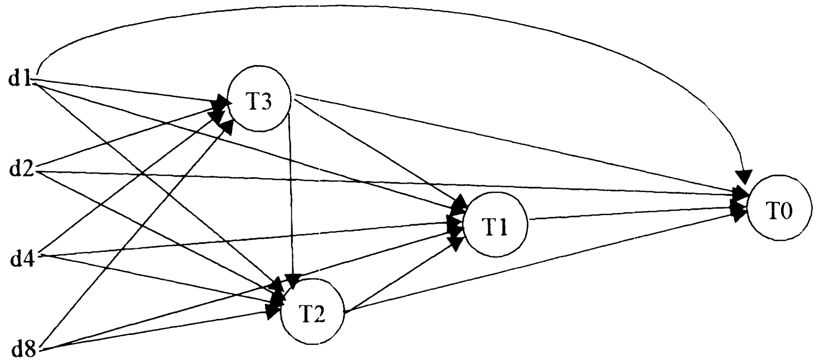


Figure 5: Neural network connectivity induced by the described evolutionary program for predicting values in the sunspot times series. Circles denote nodes that apply the corresponding activation function given in Equation 10. A connection between two nodes indicates that one node uses the value computed by the other in its activation function.

The table shows that the MIPs net compares favorably with the results of previous studies with a slightly higher training set NMSE but a better NMSE for the test set while using only half of the input nodes of the other studies. It is possible that the activation functions found using MIPs were more conducive to generalization of this data than the standard activation functions used in the previous studies resulting in better generalization on the test set with a slightly worse NMSE value on the training set. Additional experiments are planned to determine if this is indeed the case.

The update equations evolved for the best individual of generation 800 were as follows:

$$\begin{aligned}
T_3 &= \sigma(-0.8637 - (d_2 - 0.1021 + d_4 - d_4 d_4 + (d_2 + d_1 + d_4 + d_4 + d_4)d_8)) \\
T_2 &= \sigma\left(\frac{\dfrac{d_2}{T_3}}{d_1 + d_4 + T_3 d_1} - d_4 + \frac{d_8 + d_2}{-0.9370} + \frac{d_4 d_4}{0.1120} \frac{d_4}{T_3} - \frac{0.3027 - (0.0657 + d_2)}{d_2 + d_1}\right) \\
T_1 &= \sigma\left(\frac{\dfrac{T_2 d_1}{d_4 + d_8} + (d_4 - 0.4183)\left(d_8 + \dfrac{d_8}{T_2 + 0.4601 + T_3 - (T_3 + 0.9195)} - T_2\right) + d_4 + \dfrac{T_2}{d_1} - d_8}{d_1} - d_8\right) \\
T_0 &= \frac{d_1 - T_1 T_3 d_2}{T_3 + T_2}
\end{aligned} \tag{10}$$

(The σ function was added to the equations to reflect its application to each of the hidden units.) Note that equation T_0, the output of the network, is a function that scales and translates the value for d_1, the most recent value of the time series. Given that the time series is fairly smooth, and that the hidden units are limited to an output between 0 and 1 from the sigmoid function, predicting the next value as a scaled and translated value from the most recent value is an appropriate method.

Figure 5 shows the connectivity of the equations of the evolved MIPs net in the form of a neural network. The organization of the net is a single hidden unit in each of three layers. Such organizations are common in MIPs nets as they tend to provide the most computational power for the output units. Figure 6 shows the generalization performance of the dynamic system of Equation 10 evolved by MIPs. The fit is fairly tight everywhere except for the peaks in the test set data. This is especially true towards the end of the test set. Such results are to be expected for chaotic time series models.

Figure 4b shows the results of the second experiment evolving a MIPs net for the prediction of the laser data set. The solid line in the graph represents the NMSE on the laser training set for the best found individual in each generation. As for experiment 1 the graph drops rapidly for the first 100 generations and then levels out to a value below 0.06 NMSE. The dashed line indicates the performance of the best individual at each generation for the laser test set which achieved a NMSE of 0.097 on the 500 points in the test set.

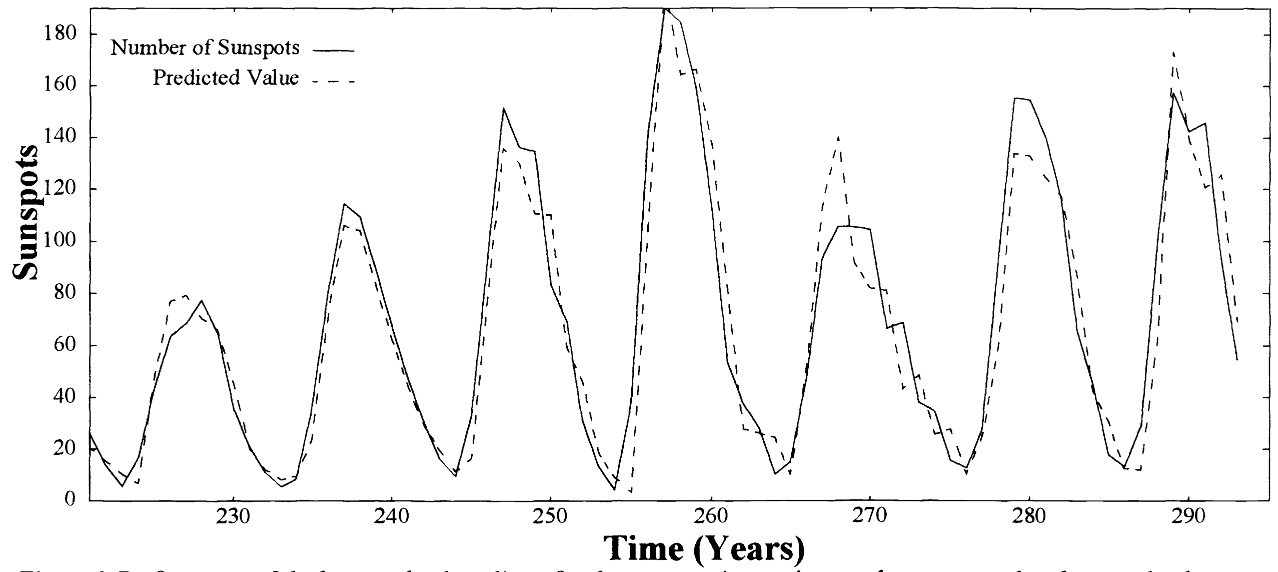


Figure 6: Performance of the best evolved predictor for the sunspot time series test data compared to the actual values.

Equation 11 shows the network for the best individual found in generation 900:

$$
\begin{aligned}
T_4 &= \sigma\left(\left(-0.3459-\frac{(-0.7805)d_2(-0.5848)}{0.3706}\right)\left(\frac{0.2547}{0.7687}+d_2\right)-\frac{d_2}{-0.3377}+\left(d_1-\left(\frac{-0.5519}{d_3}-d_3+(d_3+d_1)(-0.5784)\right)\right)d_1+d_2-0.4127\right)\\
T_3 &= \sigma\left(\frac{d_1}{-0.7106}\left(\frac{0.4242}{(d_1+d_2)/\left(\frac{0.5749}{d_2}-\left(\frac{0.8409}{d_1+d_1}-(d_3+d_3+d_3)\left(\frac{d_2}{d_3}-(T_4+d_1)\right)\right)\right)}+T_4+d_2-d_3\right)\right)\\
T_2 &= \sigma\left(\frac{T_4}{d_1}(-0.4868-d_2+d_2)T_4+d_3\right) \qquad (11)\\
T_1 &= \sigma\left(\frac{\frac{0.6390}{T_4}+0.4527}{0.9432}-d_2-\left(d_3 0.9797+\frac{d_2}{T_2+T_4 d_2}-(d_1-(T_3+T_2+T_3)(T_4+T_4-d_2))\right)\right)\\
T_0 &= T_2-d_1+T_1
\end{aligned}
$$

(The σ function was again added to the equations to reflect its application to each of the hidden units.) Note that as in the first experiment, the output equation is a function modifying the value of the latest known value of the time series. Again, given the application of the sigmoid function to the hidden values, this is an appropriate construction. The connectivity of this set of equations in terms of a neural network is shown in Figure 7. MIPs has constructed a network with four hidden layers each having a single node only two of which feed into the output node.

Figure 8 shows the output of the best evolved MIPs net for the laser test set and the difference between the predicted value and the actual value for each point in the test set. The evolved network has some difficulty predicting values around the catastrophe points in the test data, which is to be expected given only two examples in the training data. Overall, the evolved network predicts the data reasonably well especially when the number of nodes is taken into consideration. In comparison, Zhang et al.[14] report evolving a 19 node sigma-pi[5] network for this data using a considerable amount of local search to optimize the weights. Their network performed slightly better than the one evolved by MIPs, also displaying similar difficulties predicting values around the catastrophe points in the test set. A direct comparison to their results is not possible since they did not report their errors in terms of NMSE but instead in terms of a fitness function based on minimum description length computations.

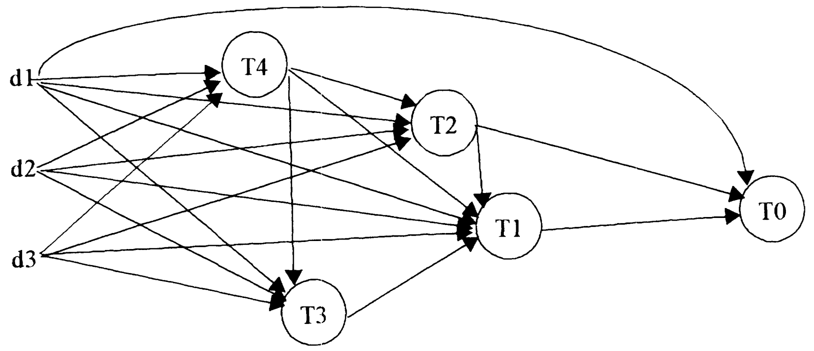


Figure 7: Neural network connectivity induced by the evolutionary program for predicting values in the laser times series. Circles denote nodes that apply the corresponding activation function given in Equation 11. A connection between two nodes indicates that one node uses the value computed by the other in its activation function.

5. CONCLUSIONS

Parameterized representations that permit automatic adjustment in order to minimize the mean squared error between the model and a desired function, such as neural networks, have proven to be considerably useful tools for scientific and industrial applications. When the representation is parameterized solely along numeric coefficients, the organization of the representation can severely limit the potential accuracy of the created model. In the event that an inappropriate basic architecture is selected, such as too few or too many hidden units, the resulting models will not achieve the desired level of accuracy. Searching over a class of symbolic organizations offers the prospect for constructing more accurate models that are more compact.

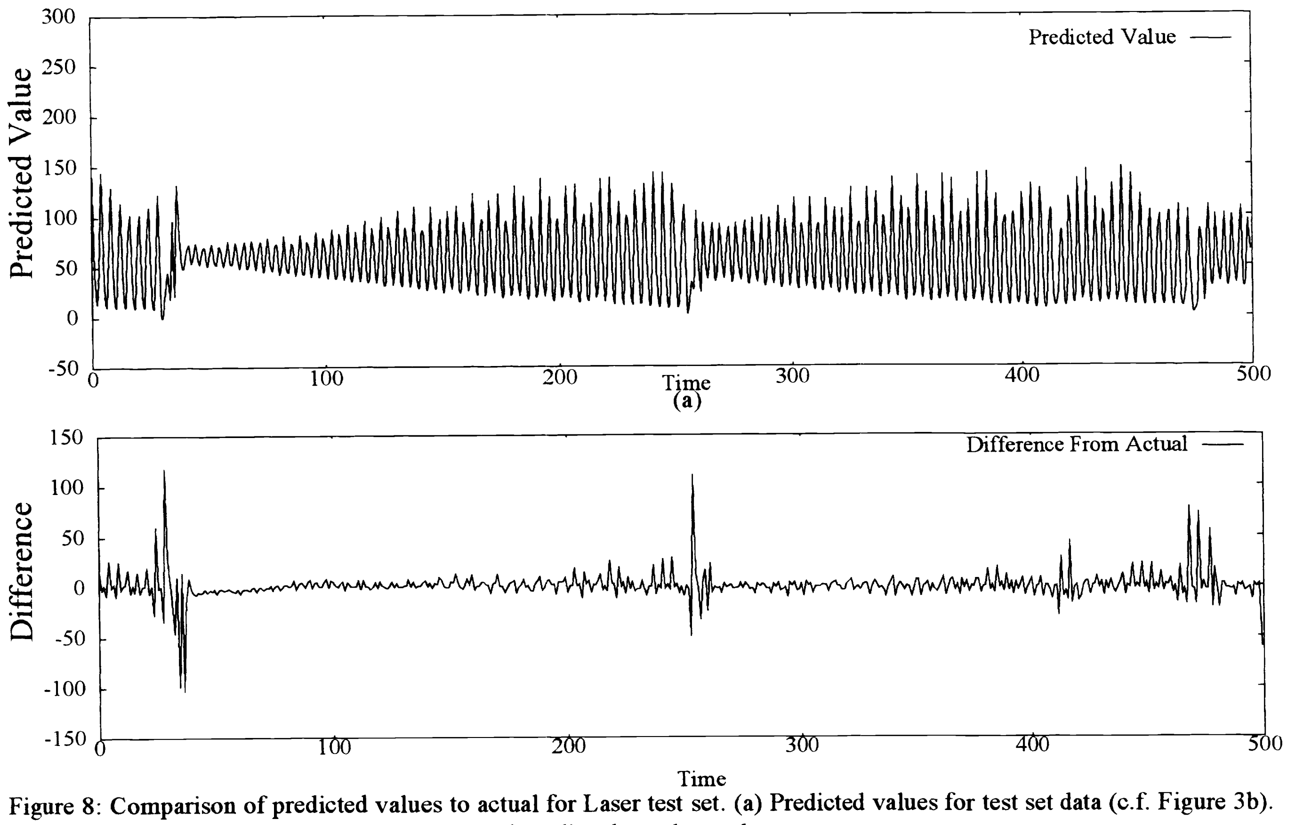


Figure 8: Comparison of predicted values to actual for Laser test set. (a) Predicted values for test set data (c.f. Figure 3b). (b) Difference between actual test data value and predicted test data value.

This improvement results from the discovery of basic symbolic organizations that more adequately reflect the necessary computational requirements for modeling the desired function. This paper takes a first step in this direction.

This paper has shown that MIPs can be used to evolve task specific symbolic activation functions for neural network-like systems of equations performing time series prediction and that these specialized functions allow fewer inputs or hidden units to be used to achieve performance comparable to other published approaches. An evolutionary program was offered as the method of manipulation for the systems of symbolic expressions. Such a method is necessary since symbolic spaces are discrete in nature and not conducive to the more conventional method of gradient search. The MIPs method works independently of the semantic interpretation of the symbols it manipulates permitting any collection of functions to be used as components for the constructed models.

An important goal for this work is the identification of symbol sets that provide sufficient flexibility. In this paper, the basic constructs of neural networks (simple arithmetic operations and a semi-linear activation function) were used to construct models using MIPs. This symbol set has proven effective for a variety of tasks. However, it may be the case that a different symbol set, say based on generalized splines or wavelets, may provide a more powerful and generally more useful evolved constructs. Ultimately, a theory relating problem properties to appropriate model components is desired so that the most effective symbol set can be provided to the algorithm to model the process. Until then, general symbol sets can be used to construct sufficient models with the process offered here.

REFERENCES

1. A. Weigend and N. Gershenfeld (eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past*, New York:Addison Wesley, 1994.
2. E. Peters, *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*, New York: John Wiley and Sons, Inc., 1994.
3. K. Hornick, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, **2** (5), pp. 359-366, 1989.
4. D. Rumelhart, D., G. H. Hinton and R. Williams, "Learning Internal Representations Through Error Propagation," in [5].
5. Rumelhart, and J. McClelland (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge MA: MIT Press. 1986.
6. D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, Piscataway, NJ: IEEE Press, 1995.
7. P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, **5** (1), pp. 54-65, 1994.
8. P. J. Angeline, "Genetic Programming and Emergent Intelligence". *In Advances in Genetic Programming*, K. Kinnear (ed.), Cambridge, MA: MIT Press, pp. 75-96, 1994.
9. P. J. Angeline, "Multiple interacting programs: A hybrid representation for evolving complex behavior", submitted to IEEE Trans. on Evolutionary Computations, 1997.
10. L.J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, New York: John Wiley, 1966.
11. N. Aerrabotu, G. Tagliarini, and E. Page, "Ensemble encoding for time series forecasting with MLP networks", *Applications and Science of Artificial Neural Networks: Proceedings of the SPIE Volume 3077*, S. Rogers (ed.), SPIE, Bellingham, Washington, USA, pp. 84-89, 1997.
12. C. Svarer, L. K. Hansen, and J. Larsen, "On design and evaluation of tapped-delay neural architectures", *IEEE International Conf. on Neural Networks*, pp. 46-51, 1992.
13. A.S. Weigend, D. E. Rummelhart, and B. A. Huberman, "Back-propagation, weight elimination and time series prediction," *Proc. of the 1990 Connectionist Models Summer School*, pp. 105-116, 1990.
14. B.-T. Zhang, P. Ohm, and H. Muhlenbein, "Evolutionary Induction of Sparse Neural Trees," *Evolutionary Computation*, **5** (2), pp. 213-237, 1997.