Evolving Neural Trees for Time Series Prediction Using Bayesian Evolutionary Algorithms

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
btzhang@scai.snu.ac.kr

Dong-Yeon Cho
Artificial Intelligence Lab (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
dycho@scai.snu.ac.kr

Abstract- Bayesian evolutionary algorithms (BEAs) are a probabilistic model for evolutionary computation. Instead of simply generating new populations as in conventional evolutionary algorithms, the BEAs attempt to explicitly estimate the posterior distribution of the individuals from their prior probability and likelihood, and then sample offspring from the distribution. In this paper we apply the Bayesian evolutionary algorithms to evolving neural trees, i.e. tree-structured neural networks. Explicit formulae for specifying the distributions on the model space are provided in the context of neural trees. The effectiveness and robustness of the method is demonstrated on the time series prediction problem. We also study the effect of the population size and the amount of information exchanged by subtree crossover and subtree mutations. Experimental results show that small-step mutation-oriented variations are most effective when the population size is small, while large-step recombinative variations are more effective for large population sizes.

1 Introduction

Evolutionary algorithms have been successfully applied to the design and learning of neural networks. Schaffer et al. [15] and Yao [17] provide early reviews of combining evolutionary algorithms and neural networks. Mühlenbein and Kindermann [10] suggest general schemes for evolving neural networks. Montana and Davis [9] and Fogel et al. [2] present evolutionary methods to train the connection weights of neural networks. They report some encouraging results which are comparable with conventional learning algorithms. Evolutionary algorithms have also been used to optimize the topology of neural networks that best fits to the specified task according to some explicit design criteria [16]. All these methods have been applied to conventional neural network architectures, such as multilayer perceptrons, and based on simple evolutionary methods.

In this paper we present an evolutionary method for evolving non-conventional neural networks. The network architectures we evolve are non-conventional in the sense that they may have heterogeneous neuron types in a single network and the connectivity of the neurons are irregular and sparse. The crux of our method is the neural tree representation scheme which is general and flexible enough to represent a broad class of network architectures and to manipulate the architectures using fairly standard variation operators.

For an efficient evolution of the neural trees, we apply the Bayesian evolutionary algorithms (BEAs). BEAs are a probabilistic model of evolutionary computation for learning and optimization. Starting from a population of individuals drawn from the prior distribution, a Bayesian evolutionary algorithm iteratively generates a new population by estimating the posterior fitness distribution of parent individuals and then sampling from the distribution offspring individuals via variation and selection operators.

In this paper, explicit formulae for the distributions of the model space are given in the context of neural trees. The effectiveness and robustness of the method is demonstrated on the time series prediction problem. We also study the effect of the population size and the amount of information exchanged by subtree crossover and subtree mutations. Experimental results show that the small-step mutation-oriented variations are most effective when the population size is small, while the large-step recombinative variations are more effective for large population sizes. This work demonstrates that evolutionary algorithms can be successfully applied to evolve neural networks. In particular, it shows the effectiveness and robustness of the Bayesian evolutionary algorithms applied to neural trees for solving time-series prediction problems.

The paper is organized as follows. In Section 2 we describe the structure of neural trees. Section 3 presents the Bayesian evolutionary algorithms for evolving neural tree models. Section 4 reports experimental results for the laser data and analyzes the effect of the prior probabilities. Section 5 summarizes our findings from this study.

2 Neural Tree Models

A neural tree is composed of terminal nodes, nonterminal nodes, and weights of connection links between two nodes [19]. The nonterminal nodes represent neural units and the neuron type is an element of the basis function set $\mathcal{F} = \{\text{neuron types}\}$. Each terminal node is labeled with an element from the terminal set $\mathcal{T} = \{x_1, x_2, ..., x_n\}$, where

0-7803-6572-0/00/$10.00 ©2000 IEEE.

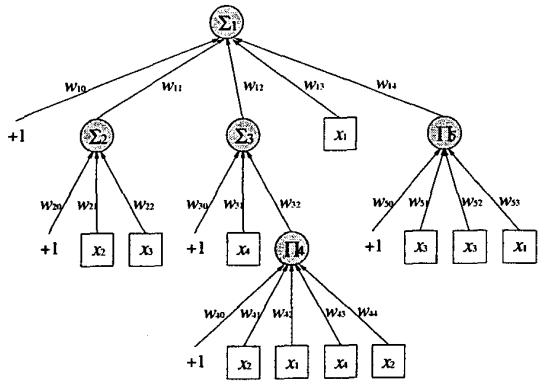

Figure 1: The structure of a neural tree.

x_i is the ith component of the external input $\mathbf{x}$. Each link (j,i) represents a directed connection from node j to node i, where node i is parent of node j and node j is child of node i. There is also a value w_{ij} which is associated with each link. In this neural tree, the root node is a output unit and the terminal nodes are input units. The depth of a neural tree d_{max} is defined as the longest path length from the root node to any terminal node of the tree.

Each nonterminal node gets input signals from maximum b_{max} child nodes and produces a single output. Different neuron types are distinguished in the way that the net inputs are computed. One of the most popular neuron types is the sigma unit, which computes the sum of weighted inputs from the lower layer by

$$net_i = \sum_j w_{ij} y_j, \tag{1}$$

where y_j are the inputs to the ith neuron. Another useful neuron type is the pi unit, which calculates the product of weighted inputs from the lower layer as

$$net_i = \prod_j w_{ij} y_j, \tag{2}$$

where y_j are the inputs to i. The output of a neuron is computed by the sigmoid transfer function

$$y_i = f(net_i) = \frac{1}{1 + e^{-net_i}}, \tag{3}$$

where net_i is the net input to the unit computed by Equation (1) or (2).

An instance of the neural tree is shown in Figure 1, where $\mathcal{F} = \{\Sigma, \Pi\}$, $\mathcal{T} = \{x_1, x_2, x_3, x_4\}$, and $b_{max} = 4$. Neural networks can represent a broad class of feedforward networks that have irregular connectivity and not-strictly layered structures [19]. The tree structure allows for easy exchange of substructures by standard subtree variation operators without destroying building blocks.

3 Bayesian Evolutionary Algorithms for Evolving Neural Trees

3.1 Bayesian Inference

Bayes theorem provides a direct method for computing the posterior probability $P(A|D)$ of each model A given the observed training data D. By Bayes theorem, we have

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)}, \tag{4}$$

where $P(A)$ is the prior probability for the model, $P(D|A)$ is the likelihood of the model for the data, and $P(D)$ is a normalizing constant and computed as

$$P(D) = \int P(D|A)P(A)dA. \tag{5}$$

With this posterior probability, we can compute the expected value of output for the unknown data $\mathbf{x}$ as follows

$$E[f_A(\mathbf{x})] = \int f_A(\mathbf{x})P(A|D)dA, \tag{6}$$

where f_A is the function implemented by a model A. In most applications, however, we cannot easily evaluate the integration in Equation (6) and numerical calculation is impossible especially for the high dimensional function f_A. Alternatively, we can find the model A^* that maximizes the posterior probability

$$A^* = \arg\max_A P(A|D), \tag{7}$$

and then use it to predict the output y for a given input x:

$$y = f_{A^*}(\mathbf{x}) \tag{8}$$

These values can be approximated by the Bayesian evolutionary algorithms [20].

3.2 Defining the Probability Distributions of Neural Trees

To find the fittest model by the BEAs, we first define the probability distributions of neural tree models for data. The posterior probability of a neural tree A is written as

$$\begin{aligned} P(A|D) &\propto P(D|A)P(A) = P(D|\mathbf{w},k)P(\mathbf{w},k) \\ &= P(D|\mathbf{w},k)P(\mathbf{w}|k)P(k), \end{aligned} \tag{9}$$

where k is the number of nodes in the neural tree (including bias terms) and $\mathbf{w}$ is the weight vector in A. Given the training data

$$D = \{(\mathbf{x}_c, y_c)\}_{c=1}^N, \tag{10}$$

the model A can represent the following input-output mapping

$$y_c = f_A(\mathbf{x}_c) + \epsilon. \tag{11}$$

Here, the noise ϵ is assumed to be zero-mean Gaussian with the standard deviation σ.

If we additionally assume that data items are independent of each other, then the likelihood of the neural tree can be expressed as follows

$$P(D|\mathbf{w},k) = \prod_{c=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_c - f_{(\mathbf{w},k)}(\mathbf{x}_c))^2}{2\sigma^2}\Big)$$

$$= \Big(\frac{1}{\sqrt{2\pi}\sigma}\Big)^N \exp\Big(-\frac{\sum_{c=1}^{N}(y_c - f_{(\mathbf{w},k)}(\mathbf{x}_c))^2}{2\sigma^2}\Big). \quad (12)$$

We define the following prior probability for weights of the neural tree

$$P(\mathbf{w}|k) = \prod_{j=1}^{k-1} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{w_j^2}{2}\Big)$$

$$= \Big(\frac{1}{\sqrt{2\pi}}\Big)^{k-1} \exp\Big(-\frac{\sum_{j=1}^{k-1} w_j^2}{2}\Big), \quad (13)$$

where the components of the weight vector are assumed to be independent of each other and distributed according to zero-mean Gaussian with the standard deviation 1. We also assume that the number of nodes in the neural tree is distributed according to following Poisson distribution

$$P(k-3) = \frac{\lambda^{k-3}\exp(-\lambda)}{(k-3)!}, \quad (14)$$

where $k = 3, 4, ...$ since the neural tree which consists of one terminal node was not considered. Substituting Equations (12), (13), and (14) into Equation (9), we obtain the following posterior probability for the neural tree

$$P(A|D) \propto P(D|\mathbf{w},k)P(\mathbf{w}|k)P(k)$$

$$= \Big(\frac{1}{\sqrt{2\pi}\sigma}\Big)^N \exp\Big(-\frac{\sum_{c=1}^{N}(y_c - f_{(\mathbf{w},k)}(\mathbf{x}_c))^2}{2\sigma^2}\Big)$$

$$\times \Big(\frac{1}{\sqrt{2\pi}}\Big)^{k-1} \exp\Big(-\frac{\sum_{j=1}^{k-1} w_j^2}{2}\Big) \frac{\lambda^{k-3}\exp(-\lambda)}{(k-3)!}. \quad (15)$$

3.3 Bayesian Evolution of Neural Trees

To search the structure and parameters of neural trees, we maintain a population $\mathcal{A}$ of individuals A_i at gth generation

$$\mathcal{A}(g) = \{A_1, A_2, ..., A_M\}, \quad (16)$$

where M is the population size. The initial population $\mathcal{A}(0)$ is created according to the prior probability of models. This is performed by sampling first a value k_i for the number of nodes from the Poisson distribution (14) and then $\mathbf{w}_i$ from the Gaussian distribution (13) using the k_i-value. In each generation g, the error $E_i(g)$ of neural trees are evaluated as follows

$$E_i(g) = \sum_{c=1}^{N}(y_c - f_{A_i}(\mathbf{x}_c)). \quad (17)$$

Using this value, we can calculate the likelihood of the neural tree in the population. Finally, the posterior probability of each model is computed by Equation (15).

To construct the next generation $\mathcal{A}(g+1)$, a candidate model A_i' is first created from the parent model A_i in the current population. The candidate model is then accepted with the following probability

$$\alpha(A_i, A_i') = \min\Big\{1, \frac{P(A_i'|D)}{P(A_i|D)}\Big\} \quad (18)$$

which is called the acceptance probability. The candidate model is always accepted when the posterior probability of the candidate model is higher than that of the parent; otherwise, it is accepted according to the ratio of two probabilities. If the candidate model is accepted, A_i' is copied into the next generation. If candidate is rejected, then A_i is copied into the next generation.

Two major variation operators are applied to the parent models for generating candidate models. First, crossover operators swap two subtrees chosen at random from the parent tree A_i and another tree $A_j, (i \neq j)$ which is selected randomly from the current population to create the candidate model A_i'. Second, mutation operators change the type of nonterminal nodes or the index of incoming units in the subtree which is also chosen randomly from the parent tree. Other kinds of mutations can also be defined (see below). The probabilities for applying these operators are p_c and p_m, respectively. These mating steps are performed iteratively until L individuals are produced.

Weights of a neural tree are adjusted through a stochastic hill-climbing. All components of the weight vector $\mathbf{w}$ are changed just once in a random order by a Gaussian mutation expression

$$w_j' = w_j + \mathrm{N}(0,1) \quad j = 1, 2, ..., k_i - 1, \quad (19)$$

where k_i is the number of nodes in tree A_i and $\mathrm{N}(0,1)$ is a normal distribution with mean 0 and variance 1. Each change of the weight is also accepted by Equation (18).

The offspring population $\mathcal{A}'(g)$ is obtained through the above procedure and we finally generate the parent population $\mathcal{A}(g+1)$ of the next generation by selecting the best M individuals from $\mathcal{A}'(g)$.

3.4 Various Implementations of the BEAs

BEAs can be divided into several classes depending on their characteristics on population structure and variation operators.

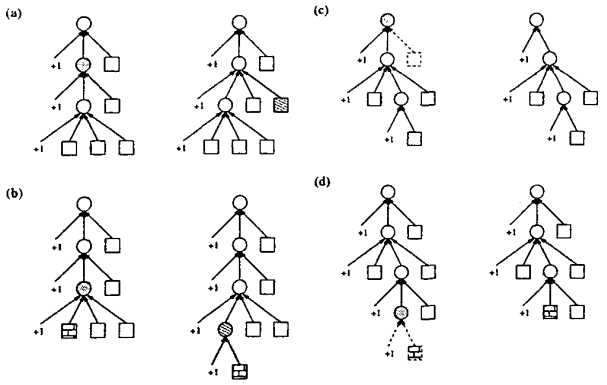

Figure 2: Insert and delete operation ($b_{max} = 3$).

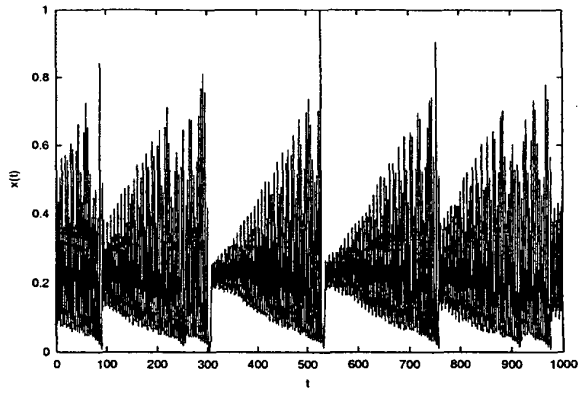

Figure 3: Laser data.

3.4.1 Individual-Based Exploitative BEAs

In this class of BEAs, there is just one model ($M = 1$) in the population. Therefore, crossover operator is not applicable. However, candidate models can be created by the insert and delete (variants of mutation) operators. These operators are limited to modify the structure of neural trees by one node at a time. This is an exploitative search in the sense the candidate model is located near the current model in the search space. The insert operator adds a random terminal node to a randomly chosen nonterminal node (Figure 2a). If the nonterminal node has b_{max} branches, one terminal node of the children nodes is changed into a nonterminal node and the terminal node becomes the child node of the new nonterminal node (Figure 2b). The delete operator removes a random terminal node from the child nodes of randomly selected nonterminal node (Figure 2c). If the nonterminal node has only one child node, the nonterminal is removed and the child node is linked to the parent node of the nonterminal node (Figure 2d). The mutation operator is applied to not a random subtree but a random node. For example, Π node can be changed into Σ node or x_1 can be changed into x_3.

3.4.2 Individual-Based Explorative BEAs

In this class of BEAs, only one individual is maintained as in the individual-based exploitative BEA, but different operators are used. A randomly chosen subtree whose depth does not exceed d_s is replaced by a randomly generated tree whose depth is d_s or less ($d_s \geq 1$). Mutation can be applied to the random subtree whose depth is also d_s or less. This method can explore the search space more broadly from the current model than the exploitative BEAs.

3.4.3 Parallelized Individual-Based BEAs

This class of BEAs maintains a large number of individuals in the population. However, crossover is not used and thus the information is not exchanged between the individuals. That is, each individual is evolved independently and in parallel.

3.4.4 Population-Based BEAs

A population-based explorative BEA is the most general implementation of BEAs, where the crossover and mutation operators are applied to any subtree of a neural tree. This is in contrast to the population-based exploitative BEA where variation operators are limited to be applied to a subtree of depth d_s or lower.

4 Experimental Results

4.1 Laser Data

The method is applied to a time-series prediction problem. The data set was generated from far-infrared NH_3 laser data in a physics laboratory [7]. It was used as a benchmark problem in the 1992 Santa Fe time series competition. The even points from the original data set were used in our experiments. The input attributes of all data set were linearly rescaled into the interval [0,1], as shown in Figure 3.

We used the first 500 data points for evolving the neural tree models and the remaining 500 data points for testing the predictive accuracy. Other experimental setup is as follows: maximum branches of a nonterminal node is $b_{max} = 3$, which is the same as the input size, the standard deviation of the noise is $\sigma = 0.05$, the average number of nodes in a tree is $\lambda = 30$, and the maximum number of evaluations is $E_{max} = 10^6$. The candidate population size is identical to the parent population size ($M = L$) in all experiments. The sum of the insert and delete probabilities is the same as the crossover probability ($p_c = p_i + p_d = 2/3$).

Figures 4 and 5 show the change in mean squared errors (MSE) of the best model for individual-based BEA (iBEA) and population-based BEA (pBEA), respectively. Tables 1 and 2 summarize the normalized MSE (NMSE) value of the best neural tree for the test data set. All results are averaged over ten runs for each method. The final results are competitive to existing neural network methods.

Regarding to the effect of population size on the evolution performance, both iBEA and pBEA have a better per-

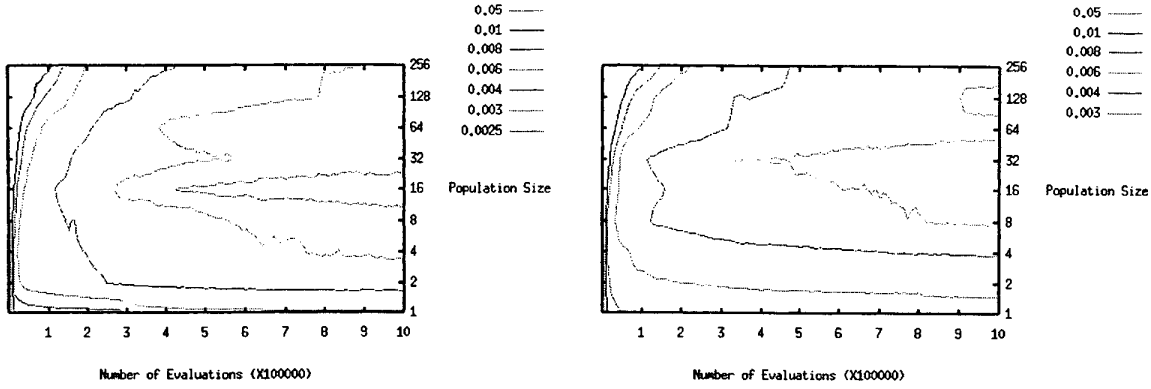

Figure 4: Evolution of MSE values of the exploitative (left) and explorative (right) iBEAs for the laser data.

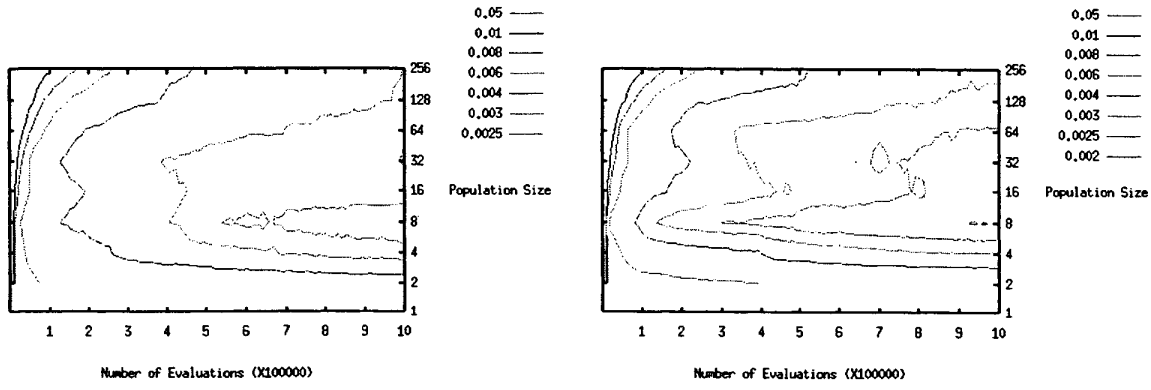

Figure 5: Evolution of MSE values of the exploitative (left) and explorative (right) pBEAs for the laser data.

Table 1: NMSE of the individual-based BEAs for the laser data.

	exploitative			explorative		
popsize	Mean ± Stdev	Min	Max	Mean ± Stdev	Min	Max
1	0.24647 ± 0.07857	0.11482	0.43451	0.27592 ± 0.10973	0.10044	0.42620
2	0.15672 ± 0.04153	0.10917	0.22679	0.20650 ± 0.07256	0.11753	0.30024
4	0.13409 ± 0.03942	0.10703	0.24168	0.16807 ± 0.05415	0.11706	0.25733
8	0.12962 ± 0.02663	0.01473	0.19283	0.13516 ± 0.02163	0.10900	0.17012
16	0.11929 ± 0.00700	0.10350	0.13050	0.13454 ± 0.01414	0.11242	0.16075
32	0.13449 ± 0.02030	0.11634	0.17976	0.12831 ± 0.01735	0.11293	0.17258
64	0.12505 ± 0.00882	0.11571	0.14449	0.14739 ± 0.01865	0.11937	0.18928
128	0.12890 ± 0.00833	0.11394	0.14143	0.12965 ± 0.01142	0.11580	0.15130
256	0.13487 ± 0.01546	0.11746	0.16799	0.14221 ± 0.01401	0.12420	0.17256

Table 2: NMSE of the population-based BEAs for the laser data.

	exploitative			explorative		
popsize	Mean ± Stdev	Min	Max	Mean ± Stdev	Min	Max
2	0.18987 ± 0.06524	0.11207	0.29105	0.22444 ± 0.05837	0.10982	0.33205
4	0.13425 ± 0.02700	0.10731	0.19897	0.14834 ± 0.05560	0.10784	0.26261
8	0.12222 ± 0.01154	0.10410	0.14444	0.11850 ± 0.00692	0.11074	0.13449
16	0.12705 ± 0.01736	0.10698	0.16315	0.12806 ± 0.01701	0.11175	0.16837
32	0.12834 ± 0.01122	0.11050	0.15154	0.12358 ± 0.00997	0.11005	0.14293
64	0.12751 ± 0.01434	0.10964	0.16282	0.12533 ± 0.00883	0.10929	0.13947
128	0.13843 ± 0.01687	0.11787	0.17168	0.13218 ± 0.00811	0.12347	0.14827
256	0.13905 ± 0.01025	0.12357	0.15887	0.13969 ± 0.01719	0.11829	0.18385

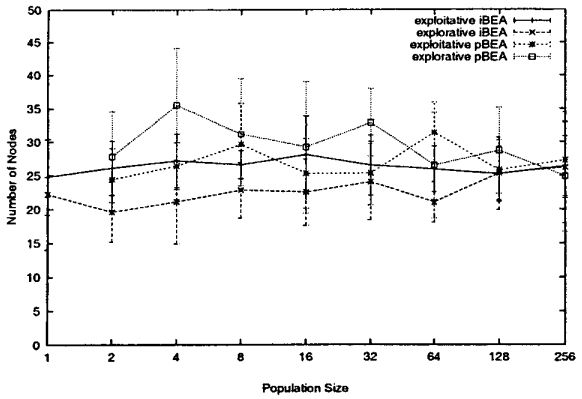

Figure 6: Effects on the number of nodes by the prior for the laser data.

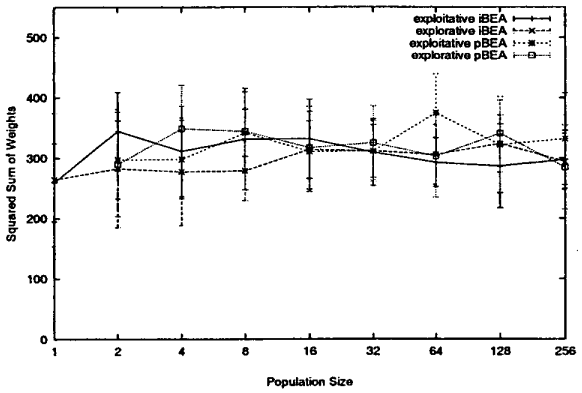

Figure 7: Effects on the weights by the prior for the laser data.

formance as the population size gets larger, but their performance degrades for very large populations. This is because the weight searching time per individual is reduced as the population size grows for the fixed evaluation time. The exploitative methods which use the insert and delete operations are generally better than the explorative methods in iBEAs while the explorative methods with unlimited crossover are better in pBEAs. Compared with iBEAs, pBEAs find better solution more efficiently, although there is minor differences in small populations. This implies that the subtrees obtained from other neural trees by crossover are more useful than the randomly generated subtrees.

To summarize, the experimental results show that iBEAs with small-step mutations are generally more effective in evolving neural trees when small populations are used, while pBEAs with large-step recombinations are effective when large populations are used.

4.2 Analysis of the Effects by the Prior

In conventional evolutionary algorithms, it is usually difficult to incorporate the prior knowledge about the model explicitly to influence the evolutionary process. However, in BEAs, it is easy to specify the prior knowledge through the prior probability. This allows for effective local search. Of course, the standard evolutionary algorithms are also able to search locally by using hill-climbing on the weight space and keeping the best individual in the next generation (elitism). However, the straightforward hill-climbing tends to increase the complexity of models as the iteration repeats and thus, eventually, the models are overfitted to the training data. This ill-behavior can be avoided by using the prior probability to control the model complexity (for more formal description, see Section 3 in [20]). Figures 6 and 7 show the complexities of the best model in terms of the number of nodes and the squared sum of weight values. It is observed that most neural trees have similar complexities which are smaller than the prior complexity λ.

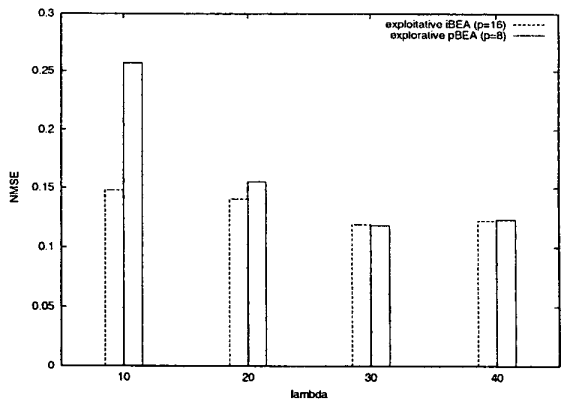

Figure 8: Effects on the NMSE values by the prior for the laser data.

In general, the optimal model size for a given problem is not known at the outset. In this case, BEAs are especially useful since they are relatively robust with respect to prior probability. To demonstrate this, we selected different values for λ. Figure 8 shows the NMSE values obtained for the test data. The whole results are similar to the case of previous λ, except that the pBEAs started with very small trees. Due to the lack of the diversity in the population, in this case, the model space was not well-explored by crossover.

5 Conclusions

In this paper, we presented Bayesian evolutionary algorithms that evolve neural tree models of time series data. The probability distributions for the prior and likelihood of the neural trees were defined and Bayes theorem was used to estimate the posterior probabilities of individuals. The experimental results on the laser data show that the method is effective in accuracy, compared with existing evolutionary algorithms.

To see the robustness of the method, we analyzed the performance of BEAs by varying the population size and the type of variation operators. The experimental results show

that using multiple individuals (i.e. pBEAs or parallelized iBEAs), contrary to the typical Bayesian inference using single chain Monte Carlo methods, is of benefit to searching the solution. In addition, the population-based BEAs with large-size crossover generally find good neural trees more efficiently when the population size is large. However, when the population size is small, mutation-oriented variations with small steps are also effective.

BEAs allow the background knowledge about the given data to be incorporated in the evolutionary procedure, and thus local search can be performed more effectively without overfitting of neural trees to the training data. Our experimental results also show that BEAs are relatively robust with respect to the prior specification in evolving neural trees.

Acknowledgements

This research was supported in part by the Korea Ministry of Science and Technology through KISTEP under grant BR-2-1-G-06.

Bibliography

[1] Andrieu, C., de Freitas, N., and Doucet, A., (2000) "Robust full Bayesian methods for neural networks," *Advances in Neural Information Processing Systems*, vol. 12, MIT Press.

[2] Fogel, D.B., Fogel, L.J., and Porto, V.W., (1990) "Evolving neural networks," *Biological Cybernetics*, vol. 63, pp. 487-493.

[3] Gilks, W.R., Richardson, S., and Spiegelhalter, D.J., (1996) *Markov chain Monte Carlo in Practice*, Chapman & Hall.

[4] Green, P.J., (1995) "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination," *Biometrika*, vol. 82, no. 4, pp. 711-732.

[5] Goldberg, D.E., (1994) "Genetic and evolutionary algorithms come of age," *Communications of the ACM*, vol. 37, no. 3, pp. 113-119.

[6] Harp, S.A., Samad, T., and Guha, A., (1989) "Towards the genetic synthesis of neural networks," *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 360-369, Morgan Kaufmann.

[7] Hübner, H., Weiss, C.O., Abraham, N.B., and Tang, D., (1993) "Lorenz-like chaos in NH_3-FIR laser," *Time series prediction: Forecasting the future and understanding the past*, pp. 73-104, Addison-Wesley.

[8] Kreutz, M., Reimetz, A.M., Sendhoff, B., Weihs, C., and von Geelen, W., (1998) "Optimisation of density estimation models with evolutionary algorithms," *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 1498, pp. 998-1007, Springer.

[9] Montana, D. and Davis, L., (1989) "Training feedforward neural networks using genetic algorithms," In *Proc. Int. Joint Conf. Artificial Intelligence.*

[10] Mühlenbein, H. and Kindermann, J., (1989) "The dynamics of evolution and learning - Towards genetic neural networks," *Connectionism in Perspective*, pp. 173-197, .

[11] Neal, R.M., (1996) *Bayesian Learning for Neural Networks*, Lecture Notes in Statistics 118, Springer.

[12] Press, S.J., (1989) *Bayesian Statistics: Principles, Models, and Applications*, Whiley.

[13] Richardson, S. and Green, P.J., (1997) "On Bayesian analysis of mixtures with an unknown number of components (with discussion)," *Journal of Royal Statistics Society B*, vol. 59, no. 4, pp. 731-792.

[14] Rios Insua, D. and Müller, P., (1998) "Feedforward neural networks for nonparametric regression," *Practical Nonparametric and Semiparametric Bayesian Statistics*, Lecture Notes in Statistics 133, pp. 181-194, Springer.

[15] Schaffer, J.D., Whitley, D., and Eshelman, L.J., (1992) "Combinations of genetic algorithms and neural networks: A survey of the state of the art," *Proceedings the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 1-37.

[16] Whitley, D., Starkweather, T., and Bogart, C., (1990) "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, **14**:347–361.

[17] Yao, X., (1993) "Evolutionary artificial neural networks," *International Journal of Neural Systems*, vol. 4, no. 3, pp. 203-222.

[18] Zhang, B.-T. and Mühlenbein, H., (1993) "Evolving optimal neural networks using genetic algorithms with Occam's Razor," *Complex Systems*, vol. 7, pp. 199-220.

[19] Zhang, B.-T., Ohm, P., and Mühlenbein, H., (1997) "Evolutionary Induction of Sparse Neural Trees," *Evolutionary Computation*, vol. 5, no. 2, pp. 213-236.

[20] Zhang, B.-T., (1999) "A Bayesian Framework for Evolutionary Computation," *Proceedings of the 1999 Congress on Evolutionary Computation*, vol. 1, pp. 722-728.