Concurrent Evolution of Neural Networks and Their Data Sets

Je-Gun Joung

School of Computer Sci. & Eng.
Seoul National University
Seoul 151-742, Korea
jgjoung@bi.snu.ac.kr

Byoung-Tak Zhang

School of Computer Sci. & Eng.
Seoul National University
Seoul 151-742, Korea
btzhang@bi.snu.ac.kr

Abstract

The ultimate goal of designing and training a neural network is optimizing the ability to minimize the expectation of the generalization error. Because active learning techniques can be used to find optimal complexity of network, active learning has emerged as an efficient alternative to improve the generalization performance of neural networks. In this paper, we propose an evolutionary approach that can design networks automatically through active data selection, where networks and data sets are evolved at the same time. Empirical results on regression and classification show improved generalization accuracy of the proposed approach for two real-world problems.

1 Introduction

The multi-layer perceptron (MLP) known to be a universal approximator has been applied quite successfully to many real world problems. Generalization can be defined as the capability of a network to perform reasonably well on unknown examples from the same probability distribution as the training examples with which the network was trained. The generalization capability is determined by the trade-off between bias and variance [1]. To achieve the best generalization we need to optimize the complexity of the networks. For a network having few hidden nodes, we can not expect to accurately solve problems because of large bias, also a network with many hidden nodes shows low accuracy because of the large variance.

The network complexity is related to the training sample size, too. If training size is too small, overfitting may occur. On the other hand, if too many data points are used, a great amount of computing time can be wasted. For any given size of data set, there is an optimal balance between bias and variance for the smallest average generalization error. For example, White [2] has shown how the complexity of a two-layer network must grow in relation to the size of the data set in order to be consistent.

One way to achieve this problem is by active learning that can improve generalization performance through data selection. In conventional neural network learning, all the available data are presented for the network training. However active learning refers to the selection of a subset of the available training data. Active learning selects more informative examples from available data for training the network. The network then plays active role in data selection, rather than being a passive learner. The neural network learner utilizes information based on its current state to gather useful examples for further training. Zhang proposed selective incremental learning, where a small number of training examples is randomly selected from a candidate set, then new examples are added from candidate set based on their criticality [3].

The propose of this paper is to present a very general evolutionary approach that evolves both population of MLPs and training data sets over generation. In order to improve the performance of the networks we deal with complexity of networks as well as the method to select data points. In evolving MLPs we use neural trees (NTs) that represent higher order neural networks (HONNs) as a generalization of the multilayer perceptrons [4].

The paper is organized as follows. Section 2 describes the evolutionary neural trees for designing and training MLPs. Section 3 describes active data selection in general. Sections 4 describes active data inheritance in more detail. Section 5 presents the experimental setup and results on benchmark problems from the Santa Fe time-series competition database and the UCI machine learning database. Section 6 concludes this paper.

2 Evolutionary Neural Trees for Neural Net Learning

A neural tree consists of nonterminal nodes and terminal nodes [4] (see Fig. 1). 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 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 and is associated with a value w_{ij}, called the synaptic

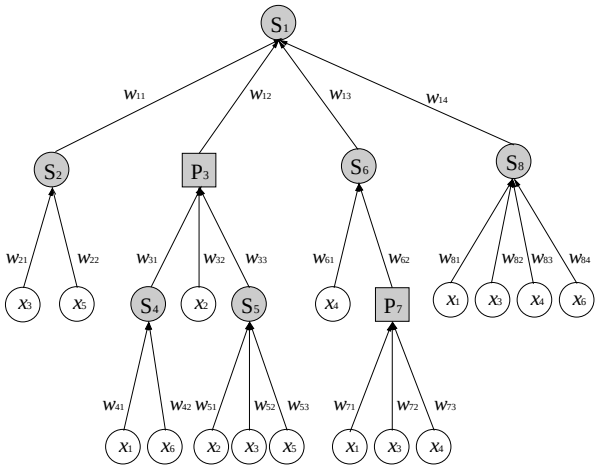

Figure 1: The structure of a neural tree.

weight.

The root node is also called the output unit and the terminal nodes are called input units. Nodes that are neither input nor output units are hidden units. The layer of a node is defined as the longest path length to any terminal node of its subtrees.

Different neuron types are distinguished in the way of computing net inputs. Sigma units compute the sum of weighted inputs from the lower layer and Pi units compute the product of weighted inputs from the lower layer:

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

where y_j are the inputs to the ith neuron. The output of a neuron is computed either by the threshold response function

$$y_i = \sigma(net_i) = \begin{cases} 1 & : \quad net_i \geq 0 \\ -1 & : \quad net_i < 0 \end{cases} \tag{2}$$

or 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 equations (1).

A higher-order network with m output units can be represented by m sigma-pi neural trees. That is, the genotype A_i of ith individual in our evolutionary framework consists of m neural trees.

The neural tree representation does not restrict the functionality since any feedforward network can be represented with a forest of neural trees:

$$A_i = (A_{i,1}, A_{i,2}, ..., A_{i,m})$$

where $A_{i,k} \in \mathcal{NT}(d, b)\ \forall k \in \{1, ..., m\}$.

The connections between input units to arbitrary units in the network is also possible since input units can appear more than once in the neural tree representation. The output of one unit can be used as input to more than one unit. The duplication does not necessarily mean more space requirements in trees than network representations since frequently-used fit submodules can be stored and multiply reused. This leads to the construction of modular structures and reduces memory requirements for representing the population [4].

Neural trees do not require decoding for their fitness evaluation. Training and evaluation of fitness can be performed directly on the genotype since both the genotype and phenotype are equivalent. Since subtree crossover used in genetic programming may be applied without modification to this representation, we can use genetic programming as the main evolutionary engine.

For the construction of neural models, we maintain a population $\mathcal{A}$ consisting of M individuals of variable size:

$$\mathcal{A}(g) = \{A_1, A_2, ..., A_M\}. \tag{4}$$

Each individual A_i is a neural network represented as neural trees. The initial population $\mathcal{A}(0)$ is created at random. In each generation g, the fitness values $F_i(g)$ of networks are evaluated and the upper $\tau\%$ are selected to be in the mating pool $\mathcal{B}(g)$. The next generation $\mathcal{A}(g+1)$ of M individuals are then created by exchanging subtrees call by crossover operator. This operator has function to reproduce neural trees with various topology, size, depth. Mutation changes the node type and the index of incoming units. For example, a sigma unit is replaced by a pi unit, and terminal node, x_5 is replaced by x_3. The best individual is always retained in the next generation so that the population performance does not decrease as generation goes on (elitist strategy).

Between generations the network weights are adapted by a stochastic hill-climbing search. This search method is based on exponential mutation [4], in which the new weight value w'_{ij} of w_{ij} is computed as follows:

$$w'_{ij} = w_{ij} \pm R \cdot \delta, \tag{5}$$

$$\delta = 2^{-K \cdot \eta} \text{ with } \eta \in [0, 1], \tag{6}$$

where R and K are constants specifying the range and slope of the exponential curve. In the experiments, the values were $R = 2$ and $K = 3$. This method proved very robust for a wide range of parameter-optimization problems. The random number η is chosen from a uniform distribution over $[0, 1]$.

The fitness F_i of the individuals A_i is defined as

$$F_i(g) = F(D|A_i^g) = E(D|A_i^g) + \alpha(g) C(A_i^g), \tag{7}$$

where $\alpha(g)$ is the Occam factor [4] that adaptively balances the error $E(D|A_i^g)$ and complexity $C(A_i^g)$

of the neural trees. This evaluation measure prefers simple networks to complex ones and turned out to be important for achieving good generalization.

3 Active Learning

The main goal of active learning algorithms is to train on the most informative examples in a candidate training set. Previous research showed improved generalization performance as well as reduced training set sizes through each active learning algorithm [3][6][7].

In most neural network applications, training examples are selected by human intuition, but active learning algorithms differ from these selection methods, where some criteria is used to trigger the selection of new examples for selecting the informative examples.

Two main approaches for active learning have been proposed [5]. One is incremental learning, where examples are selected and removed from a candidate training set. The selected patterns are added to the actual training set as training progresses, while the candidate training set is pruned. The other is selective learning, where a subset of the training patterns that satisfies a selection criterion is selected and used for training. The candidate training set is not pruned. At each pattern selection interval, all the candidate examples have a chance to be selected.

Our method combines the two approaches described above. It is the same as selective incremental learning of SEL [3]. However, the selection scheme in our method is different from the one in general active learning. Selection criterion is applied not explicitly but implicitly. The network does not select directly training examples regarded as useful for further training, but selects stochastically by applying an evolutionary selection scheme, which is the basic idea behind the incremental data inheritance (IDI) approach [8].

4 Active Data Inheritance

At each generation g, the fitness value $F_i(g)$ of all neural trees $A_i(g)$ are evaluated using the associated data sets $D_i(g)$. Fitter neural trees are chosen into the mating pool $B(g)$ and then the mating process is repeated until M offspring neural trees are produced.

The mating process is divided into two phases: neural tree inheritance and data inheritance. The neural tree inheritance phase evolves child neural trees, $A_i(g+1)$ and $A_j(g+1)$, from parent neural trees, $A_i(g)$ and $A_j(g)$, using crossover and mutation. The data inheritance phase is similar to the neural tree inheritance phase except it evolves data sets rather than neural trees.

Several data inheritance mechanisms are possible. We propose a variant of uniform crossover that we call uniform data crossover. A simplified example for illustrating this process is given in Figure 2.

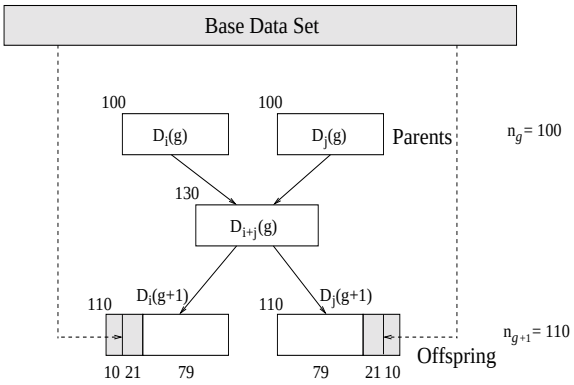

Figure 2: Uniform data crossover.

Two parent data sets, $D_i(g)$ and $D_j(g)$, are crossed to inherit their subsets to two offspring data sets, $D_i(g+1)$ and $D_j(g+1)$. In the uniform data crossover, the data of parents' are mixed into a union set

$$D_{i+j}(g) \quad = \quad D_i(g) \cup D_j(g), \tag{8}$$

which are then redistributed to two offspring $D_i(g+1)$ and $D_j(g+1)$, where the size of offspring data sets are equal to $n_{g+1} = n_g + \lambda$, where $\lambda \geq 1$ is the data increment size. Thus, the size of data sets monotonically increases as generation goes on.

To ensure performance improvement, it is important to maintain the diversity of the training data during evolution. The diversity of data set $D_i(g)$ is measured by the ratio of distinctive examples:

$$d_i = \frac{|D_{i+j}(g)|}{|D_i(g)|} - 1, \qquad 0 \leq d_i \leq 1 \tag{9}$$

where $d_i = 0$ if the parents have the same data and $d_i = 1$ if parents have no common training examples. To maintain the diversity, a portion ρ of the diversity factor d_i is used to import examples from the basis data set.

$$r_i = \rho \cdot (1 - d_i), \qquad 0 \leq \rho \leq 1. \tag{10}$$

This injection can be regarded as a data mutation.

5 Experiments

5.1 Experimental Setup

The problem used for experiment were far-infrared NH_3 laser [9] and the UCI machine learning dataset [10]. First experiment is time series prediction problem that was generated from far-infrared NH_3 laser in a physics laboratory. This problem was used as a benchmark in the 1992 Santa Fe

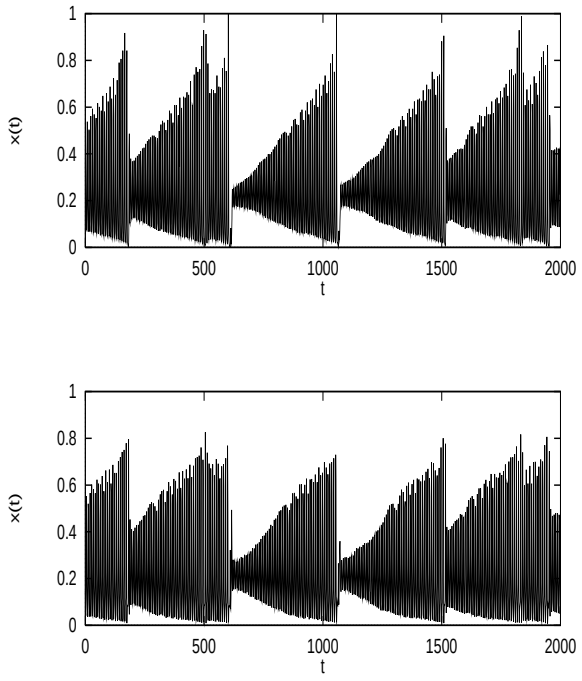

Figure 3: Performance on the laser intensity time series:(top) measured values $x(t)$ of laser intensity at the time point of $t = 1, ...2000$, (bottom) values predicted by a neural tree model evolved.

time series competition. We used the first 1000 data points for evolving the neural trees and the rest 1000 data points for testing the generalization performance.

The training examples were constructed from the time series data $x(1), ..., x(1000)$ as follows: for each t $(t = 1, ...1000)$, three contiguous values $x(t-3)$, $x(t-2)$, $x(t-1)$ were used as input for the tth training pattern, and the immediate next point $x(t)$ was used as the target value to be predicted. The test set was constructed in a similar way from the time series $x(1001), ..., x(2000)$. The $x(t)$-values were linearly scaled into the interval $[0, 1]$.

In experiments, each run consists of 50 generations with a population size of 200. The initial subset size was 40 and the increment size was $\lambda = 8$ The parameter value for import strength was $\rho = 0.3$.

Second experiment is the benchmark problem from the UCI machine learning database. They have different characteristics in many aspects, including the number of attributes, the number of examples, the distribution of positive and negative examples, and mixed numeric/discrete attributes.

The first is the breast cancer databases obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The data set contains 9 attributes, and each instance has one of 2 possible classes: benign or malignant. For 699

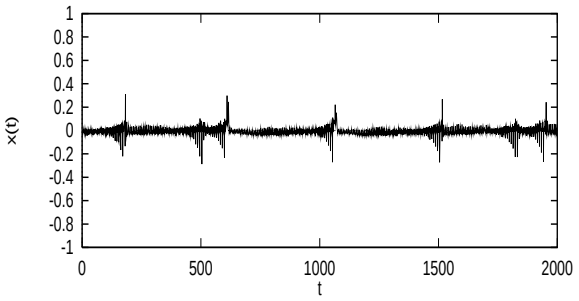

Figure 4: Performance on the laser intensity time series: prediction errors.

examples, 458 examples are benign and 241 are malignant. The second is the diabetes data set. This set is binary classes problem that tested positive or negative for diabetes. All 8 attributes are numeric-valued. The original set has 768, 500 examples belong to class '1' and 268 to class '2'. The third is heart disease data set. The purpose is to predict the presence or absence of heart disease. This database contains 75 attributes, but all published experiments refer to using a subset of 13 of them. Data set contains 297 examples. The fourth is the Australian credit card data set. There are 690 examples. This is two class problem. The 14 attributes include 6 numeric values and 8 discrete ones.

In experiments, each data set was divided into a training set (50%), a validation set (25%), and a test set (25%). The input attribute values of all data were linearly scaled into a value between 0.1 and 0.9. The output attribute has a value of 0 or 1. The same parameter values were used in all experiments. The initial subset size was 20 and the increment size was $\lambda = 4$, except the heart disease data set for which the increment size was 2. The parameter λ for the heart disease was set to a smaller value since this problem contained much smaller training data than others. In general, we found λ not very critical to the performance if it was set so that a large portion of the training data are selected by g_{max} generations. The parameter value for import strength was $\rho = 0.3$. The max generation was 50 and the population size was 100 for all runs.

5.2 Results

Fig. 3 shows a series of 2000 measurements of chaotic intensity fluctuations. In figure, the first half $(t = 1, ..., 1000)$ shows the data used for evolving the models and the rest $(t = 1001, ..., 2000)$ shows the data for testing the prediction performance of the models. Fig. 4 shows prediction error that is difference between measured value and predicted value.

Fig. 5 shows the evolution of fitness values as a

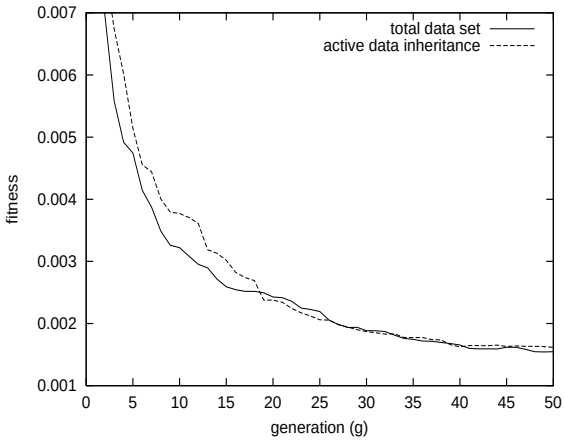

Figure 5: Comparison of fitness vs. generation for each generation with and without active data inheritance. The fitness values are averaged over 10 runs for each method.

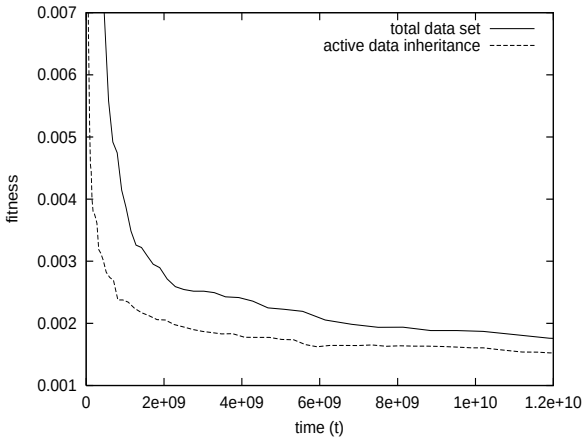

Figure 6: Comparison of cumulative computational time by generation g with and without active data inheritance. The fitness values are averaged over 10 runs for each method.

function of generation. Here, fitness given by using active data inheritance method outperform one given by training fixed data set totally. More important is the results shown in Fig. 6, where the fitness is plotted as a function of pseudo-cpu time, i.e. the number of nodes in each tree *times* the population size *times* the data size for each generation. The fitness values are averaged over 10 runs. Since we used 50 hill-climbing iterations for fitness evaluation of a model and data inheritance occurs once at each generation, the time for data selection can be ignored.

Table 1 compare evolutionary neural trees (ENT) with active data inheritance and multilayer perceptrons. We show the mean values for 20 runs. In our experiment, predictive accuracy is better than multilayer perceptrons except the breast cancer set

Table 1: Performance comparison of evolutionary neural trees (ENT) with active data inheritance and multilayer perceptrons (MLP). This is result for 30 run.

Algorithm	Problem			
	Cancer	Diabetes	Heart	Credit
ENT	3.27	22.83	19.49	16.09
MLP	3.06	23.23	19.56	16.66

6 Conclusions

We have presented a method for designing neural nets by incrementally evolving subsets of given data set. Through active learning, our method can improve the generalization performance of neural networks. In contrast to conventional approaches where training use the total data set, the adaptive data inheritance method has the advantage that informative training data set is constructed automatically during evolution.

Experimental results show that generalization performance can be enhanced by genetically selecting training data sets. A byproduct in our method is that it can reduce computational effort compared to learning with fixed data set. Our work show that evolving concurrently both the neural trees and its data can be another alternative plan for finding an optimal combination for minimizing bias and variance.

Acknowledgments

This research was supported by the Brain Science and Engineering Research Program sponsored by Korea Ministry of Science and Technology.

References

[1] C. M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[2] H. White. Connectionist nonparametric regression: multilayer feed-forward networks can learn arbitrary mappings. *Neural Networks*, Vol. 3, No. 5, pp. 535–549, 1990.

[3] B.-T. Zhang. Accelerated learning by active example selection. *International Journal of Neural Systems*, Vol. 5, No. 1, pp. 67–75, 1994.

[4] B.-T. Zhang, P. Ohm, and H. Mühlenbein. Evolutionary induction of sparse neural trees. *Evolutionary Computation*, Vol. 5, No. 1, pp. 213–236, 1997.

[5] A. Adejumo and AP. Engelbrecht. A Comparative study of neural network active learning algorithms. *Proceedings of the International*

Conference on Artificial Intelligence, pp. 32–35, 1999.

[6] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, Vol. 4, pp. 129–145, 1996.

[7] H. A. Mayer and R. Schwaiger. Towards the evolution of training data sets for artificial neural networks. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 663–666, 1997.

[8] B.-T. Zhang and J.-G. Joung. Genetic programming with incremental data inheritance. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pp. 1217–1224, 1999.

[9] H. Hübner, C. O. Weiss, N. B. Abraham, and D. Tang. Lorenz-like chaos in the NH3-FIR laser. In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A. Weigend and N. Gershenfeld, Eds. Addison-Wesley, pp. 73–104, 1993.

[10] P. M. Murphy and D. W. Aha. *UCI Repository of Machine Learning Datasets*, `http:// www.ics.uci.edu/ mlearn/ MLRepository.html`, Department of Information and Computer Science, University of California, Irvine, 1995.