computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

LEARNING AND UNDERSTANDING BASED ON NEURAL NETWORK TREES

Qiangfu Zhao

The University of Aizu
Tsuruga, Ikkimachi, Aizuwakamatsu, Japan 965-8580
qf-zhao@u-aizu.ac.jp, http://www.u-aizu.ac.jp/~qf-zhao

Abstract: *Models for machine learning can be categorized roughly into two groups: symbolic and non-symbolic Generally speaking, symbolic model based learning can provide understandable results, but cannot adapt to changing environments efficiently. On the other hand, non-symbolic model based learning can adapt to changing environments, but the results are usually "black-boxes". In our study, we introduced a hybrid model called neural network tree (NNTree). An NNTree is a decision tree (DT) with each non-terminal node containing an expert neural network (ENN). Results obtained so far show that an NNTree can be re-trained incrementally using new data. In addition, an NNTree can be interpreted easily if we restrict the number of inputs for each ENN. Thus, it is possible to perform recognition, learning and understand using the NNTree model alone.*

Keywords: – *Neural networks, decision trees, neural network trees, pattern recognition, machine learning and understanding, neural network interpretation, incremental learning*

1. INTRODUCTION

Models proposed for machine learning can be categorized roughly into two groups: symbolic and non-symbolic (or sub-symbolic). Symbolic models include decision trees (DTs), decision rules (DRs), finite state automata (FSA), and so on. Typical non-symbolic models are fuzzy logic (FL) based systems and different kinds of neural networks (NNs). Generally speaking, symbolic approaches can provide nderstandable results, but they are usually not efficient for changing environments. On the other hand, non-symbolic approaches can provide good results even if the data are not given all at once. The problem is that results provided by non-symbolic approaches are often "black-boxes" that cannot be understood easily. The point is that no single symbolic or non-symbolic model can perform both learning and understanding simultaneously. To have the advantage of both symbolic and non-symbolic learning, it is necessary to combine them together. For this purpose, many methods have been proposed in the literature [1]-[3]. Here, we just consider how to combine DTs and NNs. Briefly, we have the following approaches:

1) **Transformational approaches:** We can design a DT first, and then transform it into an NN. This approach is useful for quick design of NNs because from the DT we can get a good initial condition and determine the NN structure easily [4] [5]. For understanding, however, we are more interested in the inverse transformation. That is, we should transform a trained NN into a DT [6] [9]. This inverse transformation is often not easy.

2) **Embed the DTs into a NN:** This is a kind of modular neural network with each module being a small DT [13]. We can use this model when we know some basic concepts, but do not know how to make the global decisions. The problem in using this model is that although we know a great detail about each part, we cannot understand the whole system.

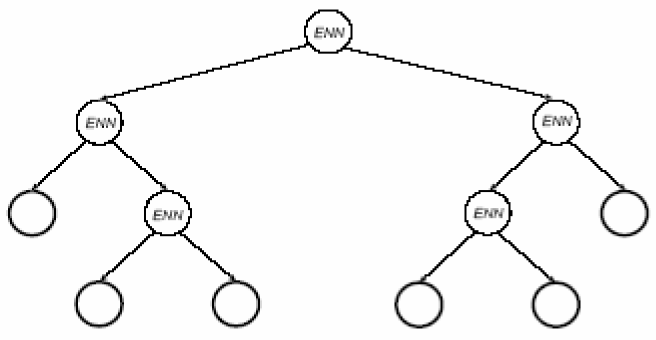

Fig. 1 An example of neural network trees

3) **Embed the NNs into a DT:** This is the neural network tree (NNTree) considered in our study. An NNTree is DT with each non-terminal node containing an expert neural network (ENN). Fig. 1 is an example of the NNTrees. This model was first used for improving the performance of the DTs [14]. In our study, we are trying to use it both for learning and for understanding [22]-[26]. Using this model, we can understand the final decision roughly if we use each ENN to extract a symbol or concept. If we are also interested in the detail, we can extract rules from the ENNs. Depends on the types of the ENNs we use, rule extraction can be very easy. In addition, the "symbols" (i.e., the ENNs) can be improved using newly observed data. Therefore, NNTree is a possible model for unifying learning and understanding.

4) **Miscellaneous:** There are many hybrid systems that can also be considered as combination of DTs and NNs, in a broad sense. Examples include: fuzzy decision trees [15] [16] and hierarchical mixtures of experts [19] [20]. Some authors also represent an NN in a tree structured genotype for genetic programming based learning [17] [18]. All these models are not understandable because they still make decisions in a black-box manner.

This paper is a summary of the results we obtained so far in our study on NNTrees. In section 2, we introduce the basic idea for evolving an NNTree off-line. In Sections 3 and 4, an incremental learning algorithm is given and its effectiveness is verified using experimental results. Section 5 and 6 prove experimentally that NNTrees can be interpreted easily if we restrict the number of inputs for each ENN. Section 7 is the conclusion.

2. OFF-LINE EVOLUTIONARY LEARNING OF NNTREES

To construct a DT, it is often assumed that a training set consisting of feature vectors and their corresponding class labels are available. The DT is then constructed by partitioning the feature space in such a way as to recursively generate the tree. This procedure involves three steps: splitting nodes, determining which nodes are terminal nodes, and assigning class labels to terminal nodes. Among them, the most important and most time consuming step is splitting the nodes. There are many criteria for splitting nodes. One of the most popular criteria is the information gain ratio which is used in C4.5 [21]. For off-line learning, the overall learning process of NNTree is the same as that of C4.5. The point here is to find an ENN in each non-terminal node to maximize the information gain ratio. To find the ENNs, we can use genetic algorithm (GA). The reason for using GA here is that we do not know in advance how to partition the examples.

Three basic operations are used in GA: selection, crossover and mutation. In our study, for simplicity, we adopted the truncation selection, one point crossover and bit-by-bit mutation. The genotype of an individual ENN is a binary string consisting of all connection weights, with each weight represented in binary number. The ENNs used in our study are multilayer perceptrons (MLPs), although other models can also be adopted. The fitness is defined as the information gain ratio. Based on these definitions, a good ENN can be generated evolutionally for each non-terminal node [23].

2. INCREMENTAL LEARNING OF NNTREES

The purpose of incremental learning is to improve an existing NNTree using new data. Incremental learning is important for a learner to adapt to environment changes. The first algorithm we proposed is *learning with fixed structure* (LWFS). In this method, an initial tree is first designed through off-line learning with currently available data. The ENNs of the NNTree are then re-trained using newly observed data.

Note that LWFS is actually supervised learning. For any new training example ***x***, the teacher signal for node ***N*** (starting from the root) can be determined as follows. Suppose that for ***x***, the *i*-th output of the corresponding ENN is the maximum (i □ [0,1] in this study), if the *i*-th child of ***N*** was assigned some examples of the class *c*=*label*(***x***) during off-line learning, the teacher signal is defined by

$$d_k = \begin{cases} 1 & k = i \\ 0 & otherwise \end{cases} \qquad (1)$$

On the other hand, if the *i*-th child of ***N*** was not assigned any example of the class *c* during off-line learning, but the *j*-th ($i \neq j$) child of ***N*** was, the teacher signal is defined by

$$d_k = \begin{cases} 1 & k = j \\ 0 & otherwise \end{cases} \qquad (2)$$

Once the teacher signals are determined, we can re-train the ENN using, say BP (back-propagation) algorithm.

Currently, we have applied LWFS to produce smaller NNTrees [25]. The basic idea is to design an NNTree using partial training data (say, 1/10 of the data) first, and then re-train the tree using all data (also off-line learning). Since the tree size is approximately proportional to the number of training data, the tree obtained by this two-stage approach is usually much smaller than that designed directly from all data. If the redundancy of the training set is high, the generalization ability of the tree will not be decreased significantly.

For incremental learning, however, we cannot expect that the data obtained at the beginning contain enough (not to say redundant) domain information for obtaining a good tree structure. We should allow the structure of the tree changeable to integrate information incrementally. For this purpose, we proposed *learning with variable structure* (LWVS) [26]. The learning process is described as follows. Suppose that the new example is $\boldsymbol{x}$, and its label is c=*label*($\boldsymbol{x}$). Start from the root node

Step 1: See if the current node is a terminal node. If not, go to Step 2; if yes, see if the class label of the node is *c*. If yes (i.e., this example is recognizable), receive the next training example, and reset the current node as the root; if not, split the node into two, with one of them containing examples of the old node, and another containing the current example. The current node now becomes a parent node. A new ENN is then designed for this node. In designing the ENN, all training examples assigned to this node so far can be used.

Step 2: Re-train the current node. In this step, we update the weights of the ENN only once using BP algorithm with the current example. The teacher signal is defined by (1) or (2).

Step 3: See if x can be classified to the correct branch. If yes, go to Step 5; otherwise, continue.

Step 4: Re-train the node again. Now, we re-train the ENN using all examples assigned to this node up to now. This is actually a review process. This will result in an ENN that can classify both new and old examples better.

Step 5: Re-train the *j*-th child recursively, where we suppose that the *j*-th output of the ENN is the maximum for the input $\boldsymbol{x}$.

Remark-1: Note that in the above algorithm, for any input example, each node on the classification-path (a similar concept as search-path) is re-trained in two steps: minor revision (Step 2) and major revision (Step 4). The basic idea is to revise the ENN slightly using BP. If the node is already good enough to recognize the current example after minor revision, input another example. Otherwise, we revise the ENN with all currently available data. These two steps can be considered as ``learning and reviewing'', which seems to be a simplified process of human learning. Of course, reviewing with all data assigned to this node so far is actually not practical for on-line incremental learning. In the future, we would like to introduce some forgetting mechanism in the reviewing process.

Remark-2: In the growing algorithm, a new terminal node is split into two whenever an example is misclassified. As the result, the tree may grow too fast. One method for preventing the tree from growing too large is to split the nodes only when some splitting condition is satisfied. The following condition is used in our study:

$$n_{total} > T \wedge n_{wrong} > s \times n_{total} \qquad (3)$$

where n_{total} is the total number of examples assigned to the node by the tree, n_{wrong} is the number of misclassified examples, *T* is a threshold and *s* is the splitting-rate. In general, *T* and *s* depend on the training set size and the number of classes. In our study, we just set *T*=30 and *s*=0.1 for simplicity. Fine-tuning is not performed. With these values, the above condition can be read as ``a new node will not be created if n_{total} is less than 30, or if the percentage of misclassified examples is less than 10%''.

Remark-3: It is interesting to note that using the above algorithm, even if we start from an empty tree, the tree can grow automatically through incremental learning. When a new ENN is created, one child contains examples of the old node, and another contains the misclassified examples. Therefore, the teacher signals can be defined straightforwardly. For minor revision and major revision, we can define the teacher signals using (1) and (2). The only difference is that we use examples assigned to the nodes *so far* instead of those assigned in *off-line learning*. Therefore, BP algorithm can be used throughout the learning process. This is important because the computational cost of BP is usually much less than that of GA. As will be shown later, the NNTrees so obtained are smaller, and have better generalization ability, as compared with those obtained by using GA based learning.

4. EXPERIMENTAL RESULTS FOR INCREMENTAL LEARNING

To verify the effectiveness of LWVS, we conducted several experiments using five databases taken from the machine learning repository of the University of California at Irvine. Parameters related with these databases are given in Table 1, where # means ``the number of''. To make the results more reliable, we adopted n-fold cross validation for all databases. For example, for ``Housevotes84'', n=5, and thus a 5-fold cross validation is used. That is, 4/5 of the data are used for learning, and 1/5 of the data are used for testing. The number n is chosen so that there is enough number of examples in the test set. This is important for reliable evaluation of the results. To increase the reliability further, 10 or 20 runs are conducted for each case; altogether 100 experiments are conducted for each database. In each run, the training data are shuffled before learning, so that the examples are provided in different order. For on-line learning, the examples in the training set are provided one-by-one until all examples are visited once. Three algorithms are compared:

1) Off-line learning: The off-line learning algorithm given in Section 2. Since all data are supposed to be available all at once, results obtained by this algorithm can be considered as the upper-limit for incremental learning.

2) GA-LWVS: Incremental learning using GA for creating new nodes and for major revision.

3) BP-LWVS: Incremental learning using BP only.

Table 1: Parameters of the databases

	#Example	#Cross valida-tion	#Run	#Feature	#Class
Derma-toloty	366	5	20	34	5
Iono-sphere	351	5	20	34	2
Tic-tac-toe	958	10	10	9	2
House-votes84	435	5	20	16	2
Car	1728	10	10	6	4

Parameters related to the ENNs include 1) the number of inputs equals to the number of features; 2) the number of hidden neurons is fixed to 5; and 3) the number of outputs is 2. Parameters related to GA for off-line learning and for creating a new node are 1) the number of generations is 1,000; 2) the population size is 200; 3) the number of bits per weight is 16; 4) the selection rate is 0.2 (That is, 20% of individuals with low fitness values are selected against in each generation); 5) the mutation rate is 0.01; and 6) the crossover rate is 0.7. For major revision in GA-LWVS, the number of generations is 100. Parameters related to BP for minor revision are 1) the learning rate is 0.5; 2) the momentum is 0; and 3) the number of epochs is 1. For major revision and for creating new nodes in BP-LWVF, the umber of epochs is 1,000.

Table 2 shows the results obtained by off-line learning. Only the recognition rates (averaged over 100 runs) for the test sets are given here. Those for the training set are always 1 (or 100%). Tables 3-5 show the sizes of the NNTrees obtained using different algorithms. For off-line learning, we have results obtained using all data, 1/2 of the data, 1/5 of the data, and 1/10 of the data. Clearly, if we reduce the number of data used in off-line learning, the tree size can be reduced. This is the basic idea for designing small NNTrees [25].

Table 2 Performance of NNTrees designed by off-line learning

Database	Tree size	Recognition rate for test set
Dermatology	12.72	0.92469
Ionosphere	8.08	0.89949
Tic-tac-toe	20.32	0.92040
Housevotes84	8.8	0.91450
Car	41.06	0.94763

Table 3 Sizes of the NNTrees obtained by off-line learning with part of the training data

	All	1/2	1/5	1/10
Dermatology	12.72	11.5	10.94	10.66
Ionosphere	8.08	5.42	3.62	3.02
Tic-tac-toe	20.32	15.96	11.02	7.66
Housevotes84	8.8	5.16	3.34	3.04
Car	41.06	31.38	20.64	14.32

Table 4 Sizes of the NNTrees obtained by GA-LWVS

Database	1/2	1/5	1/10	Empty
Dermatology	13.52	13.38	14.06	28.96
Ionosphere	10.3	10	10.88	10.04
Tic-tac-toe	62.32	63.5	64.48	60.48
Housevotes84	9.14	7.64	7.38	8.14
Car	78.04	66.52	60.84	160

Table 5 Sizes of the NNTrees obtained by BP-LWVS

Database	1/2	1/5	1/10	Empty
Dermatology	12.02	11.88	11.9	11.58
Ionosphere	8.4	7.36	6.36	5.24
Tic-tac-toe	22.36	17.62	14.74	8.36
Housevotes84	6.8	5.54	5.14	3.8
Car	42.6	31.16	26.36	18.46

For incremental learning, we have results obtained with and without initial trees. We can see that the NNTrees obtained by BP-LWVS are smaller than those obtained by GA-LWVS. They are even comparable with the results of off-line learning.

Tables 6-10 show the performance of the NNTrees obtained using different methods, for different databases. These tables show that the recognition rate for the test set. The recognition rate for the training set is given in the parenthese. From these results we can see that, in most cases, the trees after incremental learning are better than the initial trees (obtained by off-line learning). In addition, the generalization ability (recognition rate for the test set) of the NNTrees generated by the BP-LWVS is much better than those generated by GA-LWVS. For some cases the results obtained by the former are even better than those obtained by off-line learning (although not very significant).

Table 6 Results for Dermatology

	1/2	1/5	1/10	Empty
Initial	0.90038 (0.92836)	0.81074 (0.84946)	0.67050 (0.70365)	
GA-LWVS	0.90545 (0.95962)	0.89858 (0.95892)	0.89245 (0.94918)	0.84521 (0.91410)
BP-LWVS	0.92280 (0.96693)	0.92021 (0.96731)	0.92207 (0.95817)	0.92989 (0.97257)

Table 7 Results for Ionosphere

	1/2	1/5	1/10	Empty
Initial	0.87057 (0.93361)	0.82275 (0.8536)	0.77777 (0.79419)	
GA-LWVS	0.87441 (0.92999)	0.87052 (0.92974)	0.85181 (0.91984)	0.86696 (0.92336)
BP-LWVS	0.89994 (0.96897)	0.89933 (0.9709)	0.8986 (0.96688)	0.90333 (0.97667)

Table 8 Results for Tic-tac-toe

	1/2	1/5	1/10	Empty
Initial	0.8637 (0.93155)	0.7824 (0.82331)	0.7266 (0.75004)	
GA-LWVS	0.80579 (0.83522)	0.78547 (0.8255)	0.78168 (0.82183)	0.7887 (0.82464)
BP-LWVS	0.97818 (0.99)	0.97829 (0.98944)	0.97588 (0.97588)	0.97695 (0.98969)

Table 9 Results for Housevotes84

	1/2	1/5	1/10	Empty
Initial	0.9005 (0.95353)	0.8877 (0.90842)	0.8566 (0.87003)	
GA-LWVS	0.90034 (0.94754)	0.88517 (0.93721)	0.89356 (0.94552)	0.89655 (0.94448)
BP-LWVS	0.91437 (0.97822)	0.91494 (0.9754)	0.91621 (0.97914)	0.92241 (0.98348)

Table 10 Results for Car

	1/2	1/5	1/10	Empty
Initial	0.91539 (0.95596)	0.95098 (0.88048)	0.80216 (0.81778)	
GA-LWVS	0.86742 (0.87993)	0.86517 (0.87947)	0.85999 (0.87603)	0.80607 (0.81828)
BP-LWVS	0.95422 (0.96638)	0.96098 (0.97057)	0.95173 (0.96457)	0.9588 (0.96798)

5. DESIGNING INTERPRETABLE NNTREES

In the previous sections, we have shown that NNTree is a model suitable for incremental learning. Domain knowledge contained in the new data can be integrated by increasing the number of nodes dynamically. If a proper strategy for splitting node is used, the tree will not become very large. In fact, as shown by the experimental results, in most cases the tree sizes are comparable with the results obtained by off-line learning.

Now let us consider interpretation of NNTrees. In the recent years, many algorithms for interpreting a trained NN have been proposed in the literature [6]-[12]. The algorithms can be roughly divided into two categories: decompositional and pedagogical. Decompositional algorithms extract rules from each neuron (unit) in the NN and then aggregate them, while pedagogical algorithms generate training examples from the NN, and then induce rules from the examples directly. The advantage of pedagogical algorithms is that they can extract rules independent of the learning rules and the network structure. However, from the point of view of understandability and accuracy, decompositional algorithms are usually considered better because they can extract exact rules contained in the given NN.

In fact, in the worst case the computational complexity (the memory space and the computation time needed) for interpreting a single neuron is exponentially proportional to the number of inputs. For example, if the number of inputs for a neuron is 128, the computational complexity for extracting a Boolean function from this neuron is proportional to 2^{128}. This means that decompositional algorithms are in general NP- complete.

To reduce the computational cost for interpreting an NN, a direct way is to reduce the number of inputs of the neurons. For conventional NNs, we can reduce the number of inputs using some feature selection techniques. However, the number of features cannot be reduced if the features are already well selected. If we use NNTrees, it is possible to reduce the number of inputs of each ENN greatly because an ENN can make a local decision using only a few features. Thus, if we restrict the number of inputs of each ENN, the NNTree can be interpreted easily. Assume again that the number of features is 128. The computational complexity will be proportional to 2^{128} if we interpret the NNTree directly. If we restrict the number of inputs for each ENN to 8, the computational complexity will be proportional to $2^{8} \times Size$, where *Size* is the number of non-terminal nodes in the NNTree. If the inputs are properly chosen for each ENN, *Size* is usually much smaller than the number of non-terminal nodes in the conventional DT obtained by C4.5, and thus *Size* can be considered as a *constant.*

One question is that whether the tree size will be greatly increased if we restrict the number of inputs for each ENN. Theoretically, the answer is no. In fact, if we select the features used by each ENN properly, the partitioning ability of the ENN might be increased because unrelated features are sometimes just noise for making correct decisions. In this sense, restricting the number of inputs for each ENN may actually reduce the size of the NNTrees.

The point is how can we select useful features for each ENN? In general, feature selection is a difficult problem. This is more difficult for incremental learning because we cannot select good features unless we get enough information. If we just consider the case of off-line learning, we can use GA to select the features and to determine the weights of each ENN simultaneously. For this purpose, information related to feature selection can be embedded in the genotype of the ENN. Three methods were proposed in our study:

Method-I: Assign a fixed number of consecutive features to each ENN. The number of features used is pre-defined, and it is usually much smaller than the number of all features. The position of the first feature is encoded in the genotype along with the connection weights.

Method-II: As in Method-I, a number of consecutive features are assigned to an ENN. However, this time, the number of features for different ENNs can be different. The maximum number of features to be used is pre-defined and fixed. The position of the first feature and the number of features to be used are

encoded in the genotype along with the weights of the ENN.

Method-III: In this method, a number of separate features are selected from different positions. The number of used features and the positions of the features are all encoded to the genotype along with the weights. As in Method-II, the upper limit of the number of features is pre-defined. This method might be more flexible than the previous two Methods, but the search space will be much larger.

6. EXPERIMENTAL RESULTS FOR DESIGNING INTERPRETABLE NNTREES

To verify the effectiveness of the methods introduced above, and to examine the performance of the NNTrees so obtained, we conducted several experiments using four databases taken from the machine learning repository of the UCI. The databases are Dermatology, Ionosphere, Mushroom and Optdigits. To show the effect of feature selection, databases with many features were used. For detailed descriptions of the databases, see related web pages.

The experiment parameters are 1) the number of output neurons of each ENN is 2; 2) the number of hidden layers is 1; 3) the number of hidden neurons is 4; 4) the number of inputs is |F|/n, where |F| is the dimension of the feature space, and n=1, 2, 4 or 8 (For Method-II and Method-III, |F|/n is the upper limit of the number of inputs); 5) the number of runs is 10; 6) the number of generations is 1,000; 7) the population size is 200; 8) the selection rate is 0.2; 9) the crossover rate is 0.7; 10) the mutation rate is 0.01; 11) the number of bits per weight is 16; 12) each weight takes value from [-16,16]; and 13) the number of bits for encoding the position of a selected feature is 8.

We first conducted a group of experiments using Method-I only. In the experiments, the number of features for each ENN is the total number |F| divided by n=1, 2, 4, or 8. The results for n=1 are traditional NNTrees. The experimental results are shown in Tables 11-14. The results of C4.5 after pruning (with default value for the pruning parameter) are also provided for comparison. In the tables, #Inputs is the number of inputs for each ENN, Error Rate is the percentage of misclassification for the test set, and Tree Size is the number of all nodes. The Error Rate and Tree Size are averaged over 10 runs except for C4.5. The error rates for the training set are always zero because perfect training was performed.

As stated earlier, if the features are properly selected, the partitioning ability of the ENNs will not be decreased, and the tree size will not be increased. From the experimental results, however, we can see that the size of the NNTree has somewhat been increased when we restrict the number of inputs for each ENN. There are many reasons for this. For instance, the number of generations might be too small to select the best set of features for each ENN; the GA used in the experiments might be too simple; or the restriction might be to severe for the problems considered. Anyway, we should do more experiments to verify these points. What we can say right now is that in all cases the increase in tree size is *slower* than the decrease in the dimension. That is, the number of nodes will be increased less than n times when the number of features are reduced n times. This is a less-than-linear increase, but the computational complexity can be exponentially reduced.

To compare the effectiveness of different methods, we conducted another set of experiments. In these experiments, the maximum number of inputs assigned to each ENN is set to 8 or 16. Certainly, the number of inputs of Method-I is fixed. Tables 15-18 show the experimental results. In the tables, Max_#Input denotes the maximum number of inputs. In all experiments, the average sizes of NNTrees designed by using Method-II are the largest if the number of inputs for the ENN is the same. Probably, this is due to the fact that the search space of Method-II is larger than that of Method-I. From this point of view, we may think that the NNTrees designed by Method-III would be larger because the search space is larger. However, it was not the case. Actually, for the database Optdigits, the results of Method-III are much better than those obtained by using other two methods. The reason might be that Method-III can, in general, select better features, although the search space is larger.

Table 11: Results for Dermatology, using Method-I

#Inputs	Error Rate(%)	Tree Size
|F|=34	10.7	12.8
|F|/2	9.1	11.6
|F|/4	7.8	14.6
|F|/8	3.9	18.6
C4.5	5.7	29

Table 12: Results for Ionosphere, using Method-I

#Inputs	Error Rate(%)	Tree Size
|F|=34	6.2	7.6
|F|/2	7.7	9.6
|F|/4	8.5	10.0
|F|/8	8.2	12.8
C4.5	10.3	27

Table 13: Results for Mushroom, using Method-I

#Inputs	Error Rate(%)	Tree Size
|F|=117	9.0	4.2
|F|/2	9.4	4.8
|F|/4	9.7	6.6
|F|/8	10.3	7.0
C4.5	12.4	27

Table 14: Results for Optdigits, using Method-I

#Inputs	Error Rate(%)	Tree Size
|F|=64	8.6	84.8
|F|/2	9.5	114.6
|F|/4	11.0	144.4
|F|/8	13.1	235.0
C4.5	14.2	319

Table 15 Results for Dermatology

Max_#Input	Method	Error Rate (%)	Tree Size
8	I	7.8	14.6
8	II	5.6	15.4
8	III	4.8	14.4
16	I	9.1	11.6
16	II	7.0	12.0
16	III	6.0	12.8

Table 16 Results for Ionoshpere

Max_#Input	Method	Error Rate (%)	Tree Size
8	I	8.5	10.0
8	II	9.2	13.0
8	III	10.3	10.4
16	I	7.7	9.6
16	II	9.0	10.8
16	III	10.4	11.8

Table 17. - Results for Mushroom

Max_#Input	Method	Error Rate (%)	Tree Size
8	I	8.7	9.4
8	II	8.6	11.0
8	III	6.8	9.8
16	I	9.1	6.6
16	II	8.5	8.4
16	III	8.0	7.0

Table 18 Results for Optdigits

Max_#Input	Method	Error Rate (%)	Tree Size
8	I	13.1	235.0
8	II	13.7	260.4
8	III	10.6	162.6
16	I	11.0	144.4
16	II	12.3	186.6
16	III	9.7	135.2

7. CONCLUSION AND REMARKS

In this study, we have introduced a hybrid learning model called neural network tree (NNTree). Results obtained so far have confirmed that the NNTrees are suitable both for incremental learning and for understanding. However, so far the two aspects (learning and understanding) of NNTree have been investigated separatedly. In fact, it is difficult to select a proper set of features for each ENN incrementally. This is because that features useful for decision making must be selected based on enough domain knowledge.

One possible solution for this is to design a normal NNTree incrementally first, and then select using say, GA, the best features for each ENN before interpreting the NNTree. In this method, feature selection is performed after learning, and thus only information for feature selection are to be evolved. The question is that can we always reduce the number of features greatly in each ENN *after* learning?

If we have enough information at the very beginning (e.g., for data mining), we can design an NNTree with restricted number of inputs for each ENN, and then interpret. The NNTree can be re-trained incrementally using new data. Here, we need a mechanism for *dynamic replacement* of features during incremental learning. That is, if some features are found more useful than existing ones, we should use them instead of the old ones.

8. ACKNOWLEDGMENT

This research is supported in part by the Grant-in-Aid for scientific research of Japan Society for the Promotion of Science (No. 14580426). The author would like to thank Mr. Takeda and Mr. Mizuno for conducting a great number of experiments.

9. REFERENCES

[1] M. Hilario, “An overview of strategies for neurosymbolic integration,” Proc. International Joint Conference on Artificial Intelligence, 1995.

[2] S. Wermter and R. Sun, “An overview of hybrid neural systems,” in Hybrid Neural Systems, S. Wermter and R. Sun editors, Springer-Verlga, Berlin Heidelderg, 2000.

[3] L. R. Medsker and D. L. Bailey, “Models and guidelines for integrating expert systems and neural networks,” in Intelligent Hybrid Systems, A. Kandel and G. Langholz editors, CRC Press, 1992.

[4] R. P. Brent, "Fast training algorithms for multilayer neural nets," IEEE Trans. on Neural Networks, Vol. 2, No. 3, pp. 346-354, 1991.

[5] I. K. Sethi, “Entropy nets: from decision trees to neural networks,” Proc. IEEE, Vol. 78, No. 10, pp. 1605-1613, 1990.

[6] M. W. Craven and J. W. Shavlik, “Extracting tree-structured representations of trained networks,” In D. Touretzky, M. Mozer and M. Hasselmo editors, *Advances in Neural Information Processing Systems (Vol. 8)*, MIT Press, 1996.

[7] J. L. Castro, C. J. Mantas and J. M. Benitez, "Interpretation of artificial neural networks by means of fuzzy rules," IEEE Trans. Neural Networks, Vol. 13, No. 1, pp. 101-116, 2002.

[8] L. M. Fu, "Rule generation from neural networks," IEEE Trans. System, Man, and Cybernetics, Vol. 24, No. 8, pp. 114-124, 1994.

[9] G. P. J. Schmitz, C. Aldrich and F. S. Gouws, "ANN-DT: an algorithm for extraction of decision trees from artificial neural networks," IEEE Trans. Neural Networks, Vol. 10, No. 6, pp. 1392-1401, 1999.

[10] T. Mitchell and S. Thrun, “Explanation-based neuralnetwork learning: a lifelong learning approach,” Kluwer Academic Publishers, Boston, 1996.

[11] H. Tsukimoto, "Extracting rules from trained neural networks," IEEE Trans. Neural Networks, Vol. 11, No. 2, pp. 377-389, 2000.

[12] A. B. Tickle, R. Andrews, M. Golea and J. Diederich, “The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks,” IEEE Trans. on Neural Networks, Vol. 9, No. 6, pp. 1057-1068, 1998.

[13] http://www.salford-systems.com/index.html

[14] H. Guo and S. B. Gelfand,``Classification trees with neural network feature extraction," IEEE Trans. on Neural Networks, Vol. 3, No. 6, pp. 923-933, Nov. 1992.

[15] A. Suarez and J. Lutsko, “Globally optimal fuzzy decision trees for classification and regression,” IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 21, No. 12, pp. 1297-1311, 1999.

[16] C. Z. Janickow, “Fuzzy decision trees: issues and methods,” IEEE Trans. on Systems, Man and Cybernetics B, Vol. 28, No. 1, pp. 1-14, 1998.

[17] J. R. Koza, *Genetic Programming – I*, Fourth Printing, The MIT Press, 1994.

[18] B. T. Zhang and H. Muhlenbein, “Evolving optimal neural networks using genetic algorithms with Occam’s razor,” Complex Systems, Vol. 7, No. 3, pp. 199-220, 1993.

[19] R. A. Jacobs and M. I. Jordan, “Adaptive mixtures of local experts,” Neural Computation, Vol. 3, pp. 79-87, 1991.

[20] M. I. Jordan and R. A. Jacobs, “Hierarchical mixtures of experts and EM algorithm,” Neural Computation, Vol. 6, pp. 181-214, 1994.

[21] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

[22] S. Mizuno and Q. F. Zhao, "Neural Network Trees with Nodes of Limited Inputs are Good for Learning and Understanding," Proc. 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL2002), pp. 573-576, Singapore, 2002.

[23] Q. F. Zhao, "Evolutionary design of neural network tree - integration of decision tree, neural network and GA," Proc. IEEE Congress on Evolutionary Computation, pp. 240-244, Seoul, 2001.

[24] Q. F. Zhao, "Training and re-training of neural network trees," Proc. INNS-IEEE International Joint Conference on Neural Networks, pp. 726-731, 2001.

[25] T. Takeda and Q. F. Zhao, "Size reduction of neural network trees through re-training," Technical Report of IEICE, PRMU2002-105 (2002-10).

[26] T. Takeda, Q. F. Zhao and Y. Liu, "A Study on On-line Learning of NNTrees" Proc. INNS-IEEE International Joint Conference on Neural Networks, 2003.

Dr. Zhao *received the Ph. D degree from Tohoku University of Japan in 1988. He joined the Department of Electronic Engineering of Beijing Institute of Technology of China in 1988, first as a post doctoral fellow and then associate professor. He was associate professor from Oct. 1993 at the Department of Electronic Engineering of Tohoku University of Japan. He joined the University of Aizu of Japan from April 1995 as an associate professor, and became tenure full professor in April 1999.*
His research interests include image processing, pattern recognition and understanding, computational intelligence, neurocomputing and evolutionary computation.