Supervised Learning for Self-Generating Neural Networks

Wilson X. Wen
w.wen@trl.oz.au
AI Systems
Telecom Research Laboratories
Clayton, Victoria 3168
Australia

Huan Liu
liuh@iscs.nus.sg
Dept. of ISCS
National University of Singapore
Kent Ridge
Singapore 0511

ABSTRACT

In this paper, **supervised** *learning for Self-Generating Neural Networks (SGNN) method, which was originally developed for the purpose of* **unsupervised** *learning, is discussed. An information analytical method is proposed to assign weights to attributes in the training examples if class information is available. This significantly improves the learning speed and the accuracy of the SGNN classifier. The performance of the supervised version of SGNN is analyzed and compared with those of other well-known supervised learning methods.*

1 INTRODUCTION

The SGNN (Self-Generating Neural Network) method proposed in [11] is an unsupervised learning method. It has been applied to different application areas such as image coding, diagnostic expert system and document/information retrieval system [10]. Although the SGNN method performs quite satisfactorily comparing with other unsupervised learning methods, it cannot compete with other supervised learning methods in some cases since the class information has not been used.

In this paper, an information analytical method is proposed to assign weights to attributes in the training examples using class information. This significantly improves learning time and the predictive accuracy of the SGNN classifiers. The method is different from the conventional supervised learning methods, such as ID3 [5, 7], CN2 [1] and Backpropagation (BP) [8, 4], and can find its analogy with self-learning or learning from textbook. Class information is only used in assigning weights and not used in training for reinforcement. That is, there is no teacher supervising the learning procedure all the time and the system learns from a preprocessed/weighted training set which plays a similar role of a well-edited textbook. The performance of the supervised version of SGNN is analyzed and compared with those of well-known supervised learning methods. Below, we will use a simple version of SGNN - SGNT (Self-Generating Neural Tree) to elaborate how supervised learning (i.e., using class information) is carried out.

In the following section, we describe the basic unsupervised SGNT algorithm and show that it is comparable with some other known unsupervised methods but not as good as some supervised methods. In Section 3, we propose some information measures that are used in Section 4 to assign weights to attributes and select features. Stopping criteria are also given for the weight assignment procedure. In Section 5, we show in details how the information measures are used to assign weights and select features through three problems. In Section 6, we give a discussion on the features of supervised/unsupervised SGNT and the method of extending SGNT from the unsupervised to the supervised and its limitations.

2 SGNT ALGORITHM

Neural networks are usually designed by human experts. It is fairly tricky to choose the right structure of the neural network suitable for a particular application at hand. In this section, we briefly discuss the SGNT method proposed in [11] to generate neural trees [2] automatically from the training examples without human intervention.

For this kind of networks, not only the weights of the network connections but also the structure of the whole network are learned from the training examples directly. These include:

1. the number of the neurons in the network,
2. the interconnections among the neurons, and
3. the weights on the connections.

A neural tree generated in this way is called a self-generating neural tree.

Before we describe the algorithm, some related definitions are given below:

An *instance* e_i is a vector of attributes: $e_i = < a_{i1}, ..., a_{in} >$. A *neuron* n_j is an ordered pair $< W_j, C_j >$, where W_j is the weight vector of the neuron: $W_j = < a_{j1}, ..., a_{jn} >$, and C_j is the child neuron set of n_j.

An SGNT is a tree $< \{n_j\}, \{l_k\} >$ of neurons generated automatically from a set of training instances by the algorithm given below, where $\{n_j\}$ is the neuron/node set and $\{l_k\}$ is the link set of the tree.

A neuron n_k in a neuron set $\{n_j\}$ is called a winner for an instance e_i if $\forall j, d(n_k, e_i) \leq d(n_j, e_i)$ where $d(n, e)$ is the distance between neuron n and instance e. Any distance measure could be used, but we use a modified Euclidean distance measure as an example in this paper:

$$d(n_j, e_i) = \sqrt{\frac{\sum_{k=1}^{n} \rho_k \cdot (a_{jk} - a_{ik})^2}{n}} \quad (1)$$

where ρ_k is the weight for the k-th attribute. Note that, these weights, ρ, are different from those weights, w, defined later, for the links of the SGNT. The former are determined by a method proposed in the next section or a human expert prior to the learning phase while the latter are learned from the given training data set. If the learning system makes use of the class information available to select a set of suitable weights for the attributes, the performance of the classifier generated by SGNN/SGNT will be improved significantly. Especially when there are many attributes irrelevant to the underlying classification problem and other attributes make contributions which are of different importance. The main purpose of this paper is to propose an information analytical method to analyse the training set to assign different weights to different attributes so that we can improve both the classification and generalization significantly. We first introduce our SGNT algorithm:

Algorithm (SGNT Generation)

Input:

1. A set of training instances $E = \{e_i\}$, $i = 0, 1, ..., N$.
2. A threshold $\xi > 0$.
3. A distance measure for each attribute/weight in instances/neurons.

Output: An SGNT generated from E.

Method:

```
copy(root,e_0);
for each item in the training data
   minDistance = test(e_i,winner);
   if(minDistance > ξ)
      if winner is a leaf do
           copy(n_1,winner);
           connect(n_1,winner);
      copy(n_2,e_i);
      connect(n_2,winner);
   update(winner,e_i);
```

A neuron in an SGNT is called a leaf neuron if it has no child neuron. The routines/functions above are defined as follows:

1. `copy(n,e)`: generate a neuron n and copy the attribute values in the instance e to the weights in n.
2. `test(e,winner)`: find a winner in the current SGNT for instance e and return the distance between the winner and e.
3. `connect(`n_0, n_1`)`: connect neuron n_0 to n_1 making n_0 as a child neuron of n_1.
4. `update(n,e)`: update the weight vector of neuron n by the attribute vector of e based on the updating rule below:

$$w_{ij,k+1} = w_{ij,k} + \frac{1}{k+1} \cdot (a_{k+1,j} - w_{ij,k}).$$

where $w_{ij,k}$ is the j-th weight of n_i after having seen the first k examples covered by n_i.

As the network is trained repeatedly by the training set, the dead branches which stop growing are pruned away in order to improve the quality of the network and decrease its size. An SGNT branch is called dead if the number of examples covered by its root does not increase during the repeated training. Finally, it is possible to simplify an SGNT by a method similar to Quinlan's method of decision tree simplification [6]. For more detailed information about the SGNT algorithm, optimization, pruning, and simplification, please refer to [11].

The performance of the SGNT method described early is quite satisfactory in both accuracy and speed aspects comparing with some well-known unsupervised learning methods such as CLASSWEB and ECOBWEB[1] (see Table 1, where the unit of time is second).

Since the SGNN/SGNT classifier does not make use of the class information even when this information is

[1]These two are the variations of Fisher's COBWEB [3].

available, it cannot compete with the classifiers developed by some well known supervised methods (see Table 2 starting from the second row). Therefore, it seems necessary to find a way for the SGNN/SGNT method to make use of the available class information. Note that except the first rows in Tables 1 and 2 are reported by us, the rest can be found in [9]. The MONK's problems are to be explained in Section 5.

3 INFORMATION MEASURES

In this section, we propose some information measures to assign weights to attributes in the training examples so that a more informative attribute for classification is assigned with a larger weight. In order to determine whether an attribute is informative or not, we give the following discussion:

Suppose a set $\mathcal{S}$ of objects is divided into $\mathcal{N}_c$ subsets, called classes. The expected information (or entropy) for classification is

$$I(\mathcal{S}) = -\sum_{c=1}^{\mathcal{N}_c} \frac{n_c}{n} \cdot \log_2 \frac{n_c}{n},$$

where n_c is the number of objects belonging to class $\mathcal{C}_c$, and n is the total number of objects in $\mathcal{S}$. Similarly suppose $\mathcal{S}_{ik}$ is a subset of $\mathcal{S}$ in which the i-th attribute A_i of all objects has its k-th value. The expected information in $\mathcal{S}_{ik}$ for classification is

$$I(\mathcal{S}_{ik}) = -\sum_{c=1}^{\mathcal{N}_c} \frac{n_{ikc}}{n_{ik}} \cdot \log_2 \frac{n_{ikc}}{n_{ik}},$$

where n_{ik} is the number of objects for which A_i takes its k-th value, and among these n_{ik} objects, n_{ikc} belong to class $\mathcal{C}_c$. Therefore, for attribute A_i the expected information is

$$\mathcal{E}_i = \sum_{k=1}^{\mathcal{N}_i} \frac{n_{ik}}{n} \cdot I(\mathcal{S}_{ik}),$$

where $\mathcal{N}_i$ is the number of values A_i can take. Thus, the *first order information gain* contributed by A_i is

$$\mathcal{G}_i = I(\mathcal{S}) - \mathcal{E}_i.$$

and is a measure of A_i's contribution to classification. Because the gain $\mathcal{G}_i$ defined above is related to the number of values $\mathcal{N}_i$ that an attribute A_i can take, i.e., A_i tends to have a large $\mathcal{G}_i$ if $\mathcal{N}_i$ is large, we also use normalized gains to measure the contribution of each attribute. The normalized gain is defined as

$$\mathcal{G}_i' = \frac{\mathcal{G}_i}{I_i}, \quad I_i = -\sum_{k=1}^{\mathcal{N}_i} \frac{n_{ik}}{n} \cdot \log \frac{n_{ik}}{n} \ \textit{and} \ I_i \neq 0$$

In [5], $\mathcal{G}_i$ and $\mathcal{G}_i'$ are maximized to choose attribute to branch in decision trees for ID3.

In our case, if $\mathcal{G}_i$ is greater than an appropriate threshold, we may use it as the attribute weight of A_i in (1). If $\mathcal{G}_i$ is too small, it depends if $\mathcal{G}_i$ should be used as the weight of A_i, because the information gain may be not only contributed by A_i itself but also by some noise or a deviation of the sample set from the true distribution of $\mathcal{S}$. In this case, second or higher order information gains may be considered. The definitions of second or higher order information gains can be given in a way similar to that of first order information gains. For example, the *second order information gain* of a pair of attributes A_i and A_j can be defined as follows:

$$\mathcal{G}_{ij} = I(\mathcal{S}) - \mathcal{E}_{ij} \tag{2}$$

where

$$\mathcal{E}_{ij} = \sum_{k=1}^{\mathcal{N}_i} \sum_{l=1}^{\mathcal{N}_j} \frac{n_{ijkl}}{n} \cdot I(\mathcal{S}_{ijkl});$$

n_{ijkl} is the number of objects in S whose i-th and j-th attributes take their k-th and l-th values, respectively;

$$I(\mathcal{S}_{ijkl}) = -\sum_{c=1}^{\mathcal{N}_c} \frac{n_{ijklc}}{n_{ijkl}} \cdot \log_2 \frac{n_{ijklc}}{n_{ijkl}};$$

and n_{ijklc} is the number of objects belonging to class $\mathcal{C}_c$ whose i-th and j-th attributes take their k-th and l-th

Learning method tested		Test results M_1	M_2	M_3	**Average**	Tester(s)
SGNT (unsupervised)	Accuracy	82.6%	78.2%	84.5%	**81.8%**	Wen et. al.
	time	1.47	2.37	2.69		
CLASSWEB 0.10	Accuracy	71.8%	64.8%	80.8%	**72.5%**	Kreuziger et. al.
	time	1406.47	2013.78	1311.25		
CLASSWEB 0.15	Accuracy	65.7%	61.6%	85.4%	**70.9%**	Kreuziger et. al.
	time	867.47	977.04	822.09		
CLASSWEB 0.20	Accuracy	63.0%	57.2%	75.2%	**65.1%**	Kreuziger et. al.
	time	499.94	646.06	521.21		
ECOBWEB leaf prediction	Accuracy	71.8%	67.4%	69.1%	**72.5%**	Reich & Fisher
	time	N/A				
ECOBWEB l.p & inform.util.	Accuracy	82.7%	71.3%	68.0%	**74.0%**	Reich & Fisher
	time	N/A				

Table 1: Comparisons for MONK's problems among unsupervised learning methods

Learning method tested	Test result		
	M_1	M_2	M_3
SGNT (supervised)	100%	87%	100%
AQ17-DCI	100%	100%	94.2%
AQ17-HCI	100%	93.1%	100%
AQ15-GA	100%	86.8%	100%
Assistant Professional	100%	81.3%	100%
mFOIL	100%	69.2%	100%
ID5R	81.7%	61.8%	N/A
IDL	97.2%	66.2%	N/A
ID5R-hat	90.3%	65.7%	N/A
TDIDT	75.7%	66.7%	N/A
ID3	98.6%	67.9%	94.4%
ID3 (no windowing)	83.2%	69.1%	95.6%
ID5R	79.7%	69.2%	95.2%
AQR	95.9%	79.7%	87.0%
CN2	100%	69.0%	89.1%
PRISM	86.3%	72.7%	90.3%
Backpropagation	100%	100%	93.1%
Backprop. with weight decay	100%	100%	97.2%
Cascade Correlation	100%	100%	97.2%

Table 2: Comparisons for MONK's problems among supervised methods

values, respectively. The normalized second order gain can be defined accordingly.

$$\mathcal{G}'_{ij} = \frac{\mathcal{G}_{ij}}{I_{ij}}, \quad I_{ij} = -\sum_{k=1}^{\mathcal{N}_i}\sum_{l=1}^{\mathcal{N}_j} \frac{n_{ijkl}}{n} \cdot \log_2 \frac{n_{ijkl}}{n}. \qquad (3)$$

Sometimes, we need consider not only the absolute higher order gains but also the higher order gains related to the corresponding lower order gains. For example, the second order gain of attribute pair A_i and A_j related to the first order gain $\mathcal{G}_i$ is $\mathcal{G}_{ij,i} = \mathcal{G}_{ij} - \mathcal{G}_i$ and the corresponding normalized relative gain $\mathcal{G}'_{ij,i} = \mathcal{G}'_{ij} - \mathcal{G}'_i$. The relative gains are useful when both $\mathcal{G}_i$ and $\mathcal{G}_{ij}$ are greater than the corresponding thresholds, and we would like to know whether the contribution of A_j is significant to the classification.

Although the impact of higher order information gains are not as obvious as that of first order gains, it is sometimes important for the performance of the classifiers.

4 ATTRIBUTE SELECTION AND WEIGHT ASSIGNMENT

In this section, we discuss heuristic rules to select attributes related to the underlying classification, when there are irrelevant attributes in the training examples, and to assign weights to the selected attributes.

An attribute A_i is called *first order selectable*, if its gain $\mathcal{G}'_i > \mathcal{G}'^{(1)}$, where $\mathcal{G}'^{(1)}$ is the average of all normalized first order gains. If an attribute is first order selectable, its contribution to the classification is above average and thus should not be ignored. If an attribute is not first order selectable, it can still be selected because of the joint contribution of the attribute and other attribute(s). An attribute A_i is called *second order selectable* if one of the following conditions is satisfied:

1. There is another attribute A_j, A_i and A_j are not first order selectable but the normalized second order gain $\mathcal{G}'_{ij} > \mathcal{G}'^{(2)}$, where $\mathcal{G}'^{(2)}$ is the average of all normalized second order gains except those second order gains corresponding to attributes which are both first order selectable.

2. A_j is first order selectable, but A_i is not, and $\mathcal{G}_{ij,j} > \mathcal{G}^{(2,1)} \wedge \mathcal{G}'_{ij,j} > \mathcal{G}'^{(2,1)}$, where $\mathcal{G}'^{(2,1)}$ and $\mathcal{G}^{(2,1)}$ are the averages of relative normalized/unnormalized second order gains of all pairs of the attributes except those that correspond to two first order selectable attributes.

Note that the order of the above logic expression is significant. That is, if $\mathcal{G}_{ij,j} > \mathcal{G}^{(2,1)}$ is not true, we can simply reject A_i as selectable, and it is not necessary to calculate and test $\mathcal{G}'_{ij,j} > \mathcal{G}'^{(2,1)}$. In these cases, the average of the gains is only over those gains which have already been calculated.

Higher order selectibility can be also defined similarly. If expert knowledge is available in the learning, the averages can be replaced by any thresholds chosen according to the knowledge. In this paper, we use the averages as the thresholds assuming no expert knowledge is available.

If an attribute is neither first nor second order selectable, we call it *second order rejectable*. In this paper, we reckon that an attribute is irrelevant to a classification problem if it is at least second order rejectable and thus is assigned a weight 0. For all other attributes, if their first order gains are greater than a threshold, say $\frac{1}{\mathcal{N}^2}$, where $\mathcal{N}$ is the number of attributes in a training vector, set their weights equal to the gains. Otherwise, set their weights to the reciprocal of the corresponding gains since we would like to reduce the deviation of the training samples from the true distribution. It should be noted that although normalized gains are used for attribute selection, unnormalized gains should be used for weight assignment since attributes with larger numbers of values need bigger driving forces for classification.

Before we show, in examples, how to use information gains and the above heuristic rules to assign weights to the attributes, we discuss when the higher order gains should be calculated, or when the calculation is not necessary.

4.1 Stopping Criteria

At the beginning, two sets are defined: SELECT and START. SELECT is null and START contains all the attributes. Data is treated in a conventional manner, divided into two sets: TRAIN and TEST. P_i is a performance measure, e.g. predictive accuracy.

```
1. i = 1, β = 0.02; # β is a tunable parameter
2. compute ith order gains to select attributes
```

```
      from START and add into SELECT;
    induce a concept hierarchy, C, using
      the attributes in SELECT;
    obtain P1 of C on TEST;
3. i = i +1;
    compute ith order gains to select attributes
      from START and add into SELECT;
    induce a concept hierarchy, C, using
      the attributes in SELECT;
    obtain P2 of C on TEST;
4. if (START = SELECT) or
       (P2 < P1) or
       (P2 < P1 + β) or
       (P2 ≥ 1 - β)
    then STOP
    else P1 = P2;
         go to 3.
```

Here we provide four stopping criteria. First one says if all attributes are selected, the selection algorithm should stop. Second one points out a theoretical possibility that the more attributes selected, the worse the performance. The algorithm should stop if this occurs. Third one amounts to if there is no significant improvement, stop. And fourth one states that if the performance is sufficiently good, stop.

5 EXAMPLES

The MONK's problems proposed in [9] become a *de facto* standard set of benchmarks because

1. MONK's tests have been performed for many (at least 20) well-known supervised/unsupervised learning methods at the 2nd European Summer School on Machine Learning during summer 1991. This makes a comparison between our method and those well-known methods much easier and less biased.

2. The training and test data and the test results for other well-known methods are easily accessed.

3. The MONK's problems are not too complicated, time-consuming to test but are also reasonably difficult to be solved in different aspects such noise, joint concepts.

The MONK's problems [9] rely on an artificial robot domain, in which robots are described by six different attributes:

A_1:	head_shape	$\in$	round, square, octagon;
A_2:	body_shape	$\in$	round, square, octagon;
A_3:	is_smiling	$\in$	yes, no;
A_4:	holding	$\in$	sword, balloon, flag;
A_5:	jacket_color	$\in$	red, yellow, green, blue;
A_6	has_tie	$\in$	yes, no.

The learning tasks of the three MONK's problems are binary classification tasks, each of them is given by the following logical description of a class.

- Problem M_1: (head_shape = body_shape) or (jacket_color = red). From 432 possible examples, 124 were randomly selected for the training set. No noise is present.

- Problem M_2: Exactly two of the six attributes have their first value. From 432 examples, 169 were selected randomly. No noise is present.

- Problem M_3: (Jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is no octagon). From 432 examples, 122 were selected randomly. and among them there were 5% misclassifications, i.e., noise in the training set.

The graphical representations of the training and testing sets for all these three problems are given in the Appendix of the report. It is easy to see that M_3 should be the easiest one to learn if there were no noise present because there is only a Disjunctive Normal Form (DNF) of first order relations to be learned in the description. M_1 is a bit harder because there is a second order relation, head_shape = body_shape, to learn. However, the noise in M_3 changes the order of difficulty between M_1 and M_3. The most difficult one is M_2, one has to examine all the six attributes of any example to tell whether it belongs to the right class.

Methods like SGNT/SGNN or COBWEB [3], will not perform well for these examples. The reason is twofold: firstly, these methods are unsupervised; secondly, these methods treat all attributes equally important, but in some (M_1 and M_3) of these problems some attributes may be much more important than the others. In effect, A_3, A_4, and A_6 are completely irrelevant in M_1, and A_1,A_3, and A_6 in M_3 to the classification problems. Methods such ID3 [5] and the like should perform better than the above mentioned methods not only because they are supervised, but also because they discriminate some minor attributes from the others in one way or another. However, even ID3 cannot achieve the best results for MONK's problems because the problems are obviously involved in higher order relations but ID3 deals only with first order information. An information analysis conducted below also confirms the above intuitive conjectures.

5.1 MONK'S PROBLEM 1

The information analysis for the training set can be performed starting from calculating the first order information gains. For M_1, the result of first order information analysis is shown in Table 3.

The expected information for classification in M_1 is

$$I(M_1) = -\frac{62}{124} \cdot \log_2 \frac{62}{124} - \frac{62}{124} \cdot \log_2 \frac{62}{124} = 1.0$$

For $A_1 = 0$ in M_1, the information is

$$I(A_1) = -\frac{14}{45} \cdot \log_2 \frac{14}{45} - \frac{31}{45} \cdot \log_2 \frac{31}{45} = 0.8946.$$

Attr	Value		class		Info.	N'lized
	val	#	$\mathcal{C}_0$	$\mathcal{C}_1$	gain $\mathcal{G}$	gain $\mathcal{G}'$
A_1	0	45	14	31	.0753	.0476
	1	42	22	20		
	2	37	26	11		
A_2	0	35	15	20	.0058	.0037
	1	42	22	20		
	2	47	25	22		
A_3	0	65	35	30	.0047	.0047
	1	59	27	32		
A_4	0	42	26	16	.0263	.0166
	1	43	21	22		
	2	39	15	24		
A_5	0	30	11	19	.2870	.1437
	1	31	11	20		
	2	34	11	23		
	3	29	29	0		
A_6	0	56	29	27	.0008	.0008
	1	68	33	35		

Table 3: Information analysis for M_1

Gains	Norm Gain	Rela Gain	Norm Rela
$\mathcal{G}_{12}$: .5147	$\mathcal{G}'_{12}$: .2403	$\mathcal{G}_{12,1}$: .4394	$\mathcal{G}'_{12,1}$: .1926
$\mathcal{G}_{13}$: .0855	$\mathcal{G}'_{13}$: .0487		
$\mathcal{G}_{14}$: .0968	$\mathcal{G}'_{14}$: .0447		
$\mathcal{G}_{16}$: .0876	$\mathcal{G}'_{16}$: .0498		
$\mathcal{G}_{23}$: .0125	$\mathcal{G}'_{23}$: .0075		
$\mathcal{G}_{24}$: .0335	$\mathcal{G}'_{24}$: .0167		
$\mathcal{G}_{25}$: .3006	$\mathcal{G}'_{25}$: .1322	$\mathcal{G}_{25,5}$: .0136	
$\mathcal{G}_{26}$: .0305	$\mathcal{G}'_{26}$: .0183		
$\mathcal{G}_{34}$: .0353	$\mathcal{G}'_{34}$: .0268		
$\mathcal{G}_{35}$: .3078	$\mathcal{G}'_{35}$: .2004	$\mathcal{G}_{35,5}$: .0208	
$\mathcal{G}_{36}$: .0195	$\mathcal{G}'_{36}$: .0193		
$\mathcal{G}_{45}$: .3247	$\mathcal{G}'_{45}$: .1344	$\mathcal{G}_{45,5}$: .0377	
$\mathcal{G}_{46}$: .0509	$\mathcal{G}'_{46}$: .0293		
$\mathcal{G}_{56}$: .3053	$\mathcal{G}'_{56}$: .1351	$\mathcal{G}_{56,5}$: .0183	
Average:	.0552	.1060	.1926

Table 4: Second gains for M_1

Similarly, the information for $A_1 = 1$, and 2 is $0.9985, 0.8780$, respectively. Thus, the expected information and the first order information gain for A_1 are

$$\mathcal{E}_1 = \frac{45}{124} \cdot 0.8946 + \frac{42}{124} \cdot 0.9985 + \frac{37}{124} \cdot 0.8780 = 0.9246,$$

and

$$\mathcal{G}_1 = 1 - 0.9246 = 0.0753.$$

Other first order information gains and the normalized gains can be calculated similarly, and are shown in the last two columns of Table 3. The average of this the information gains 0.0666 can be chosen as the threshold of the information gains. Therefore, only two first order selectable of six attributes, A_1 and A_5 are selected with weights 0.0753 and 0.2870. The second order gains can be calculated according to (2) and (3), and are given in Table 4.

It is easy to see from Table 4 that only A_2 is second order selectable (see the first row in Table 4, in fact, it is not necessary to calculate $\mathcal{G}'_{12,1}$ because it is the only candidate). Thus, A_2 is assigned a weight of 0.0058 and all the other attributes are second order rejectable and assigned weights 0. Using the attribute weight (ρ) set $\{0.0753, 0.0058, 0.0, 0.0, 0.2870, 0.0\}$ for M_1, we obtain a network with a classification accuracy of 100% for both training and testing sets (Table 2), thus the selection algorithm stops.

$\mathcal{G}_1$: .0071	$\mathcal{G}_2$: .2937	$\mathcal{G}_3$: .0008
$\mathcal{G}_4$: .0029	$\mathcal{G}_5$: .2559	$\mathcal{G}_6$: .0071
$\mathcal{G}'_1$: .0045	$\mathcal{G}'_2$: .1854	$\mathcal{G}'_3$: .0008
$\mathcal{G}'_4$: .0018	$\mathcal{G}'_5$: .1281	$\mathcal{G}'_6$: .0071

Table 5: First order information gains for M_3

5.2 MONK's PROBLEM 3

As mentioned above, there were 5% misclassifications or noise in the training set of M_3. This makes M_3 more difficult than M_1. If there were no noise in the training sets, M1 is more difficult than M_3. Hence, the selection of the gain threshold will be more sensitive to the classification performance for M_3. The first order information gains in the training set is given in Table 5.

Among the first order gains only $\mathcal{G}'_2$ and $\mathcal{G}'_5$ are greater than the average 0.0468 of the 6 first order normalized gains. Therefore, A_2 and A_5 are assigned weights 0.2937 and 0.2559, respectively. There are 8 attribute pairs whose second order normalized gains are greater than the average 0.0682. Among them only three $\mathcal{G}_{15,5}$, $\mathcal{G}_{24,2}$, $\mathcal{G}_{45,5}$, have relative gains above the average. Because the normalized relative gain $\mathcal{G}'_{15,5}$ is below average 0.0090, only A_4 is second order selectable and is assigned a weight 0.0029. Using attribute set $\{0.0, 0.2937, 0.0, 0.0029, 0.2559, 0.0\}$, we generate a network for M_3, again, with 100% classification accuracy for testing set and thus the selection algorithm stops. It is interesting to note that the network constantly corrects the 6 misclassified examples when testing the training set.

Our intuition has been verified by our information analysis that M_3 would be harder than M_1 because of the noise added in the training set. For M_1, no normalized relative gain needs to be calculated. However, for M_3, if we did not calculate the normalized relative gains and compare them with the average, we might well have selected A_1 in addition to A_4. The result network generated by attribute weight (ρ) set $\{0.0071, 0.2937, 0.0, 0.0029, 0.2559, 0.0\}$ always misclassifies 6-12 testing examples.

5.3 MONK's PROBLEM 2

As we mentioned above, M_2 should be the hardest one among the three MONK's problems because the classification does not directly related to the values of the attributes. This intuition will be again verified by our information analysis. The first order information gains are given in Table 6.

$\mathcal{G}_1$: 0.0038	$\mathcal{G}_2$: 0.0025	$\mathcal{G}_3$: 0.0011
$\mathcal{G}_4$: .0157	$\mathcal{G}_5$: .0173	$\mathcal{G}_6$: .0062
$\mathcal{G}'_1$: .0024	$\mathcal{G}'_2$: .0016	$\mathcal{G}'_3$: .0011
$\mathcal{G}'_4$: .0099	$\mathcal{G}'_5$: .0087	$\mathcal{G}'_6$: .0062

Table 6: First order information gains for M_2

All first order information gains and normalized information gains are very small (the average of the normalized gains is less than $\frac{1}{\mathcal{N}^3} = 0.0046)^2$. In this case, it is hard to tell whether these gains are contributed by the attributes themselves or the deviation of the sample distribution of the training set from the real distribution. The second order analysis shows that the second order gains are much larger than the first ones although they are still fairly small (the normalized gain average is less than $\frac{1}{\mathcal{N}^2} = 0.0277$). Using the same analysis procedure as that in previous two subsections, we find that A_4, A_5, and A_6 are first order selectable, and A_1, A_2 (they are not first order selectable but $\mathcal{G}'_{12}$ is above the average 0.0164), and A_3 are second order selectable. This means that although each attribute alone contributes little but the higher order combinations of the attributes may contribute a lot to the classification. Higher order information analysis is not needed since all six attributes are selected. We could either simply use the second order gain average over each attribute as the weights of the attribute or use weights proportional to the reciprocal of the corresponding first information gains to reduce the deviation of the training sample from the real distribution. For example, using the reciprocal method, the network generated has a classification accuracy of 87.3%.

6 DISCUSSION

Although the SGNT/SGNN method was mainly developed for the purpose of unsupervised learning and the class information is only used to assign weights to the attributes (never used in training process), Table 2 shows that the performance of a supervised version of SGNT is encouraging. It can be seen that the supervised SGNT reaches perfect classifications for the testing sets of both M_1 and M_3. This means that SGNT performs very well for learning even with noise presented. The information analysis and weight assignment method improves the performance nearly 20% over the corresponding unsupervised SGNT. For M_2, the improvement is not so impressive because no direct correlation is found between the classification and the values of the attributes. We conjecture that the method proposed here does not help much in cases such as parity problems where the classification is not directly correlated with the values of the attributes. The AQ like methods reach a better performance because they generated new attributes to reflect the number of attributes belonging to different classes. The BP and the like seem very good for this kind of problems (M_2, parity, and XOR) but require efforts from human experts to design the network structures (including number of hidden units, weight decay, etc.) manually.

The speed comparisons in Table 7 show again that the SGNT method is much faster than any other supervised learning methods tested. The time spent by calculating the first and second order information gains for all three MONK's problems are the same, 0.3 second, and is not included in the training time given in Table 9. In general, the computation amount for the second order information analysis is $\mathcal{O}(\mathcal{N}^2 \cdot \mathcal{N}_{max}^2 \cdot n)$, where $\mathcal{N}$, $\mathcal{N}_{max}$, and n are the number of attributes, the maximum number of values of the attributes, and the number of training examples. In the supervised version of SGNT, there is no training set splitting and gain recalculation in ID3 and the like, nor the global network updating in BP type neural networks. The information gains are only calculated once before the training phase, the class information is never used again during training for reinforcement and there is no information loss caused by splitting the training set. The network is only locally updated each time and the irrelevant attributes to the classification problems are ruled out prior to the training.

It turns out that the supervised SGNT method has quite a few attractive features comparing with the conventional neural networks and the other unsupervised learning techniques in conventional AI:

1. The structure is automatically generated for the applications at hand. The performance is quite satisfactory. Whenever it is difficult to determine the right network structure, this method may well be considered as an alternative to conventional neural network methods such as backpropagation for which the network structures have to be skillfully determined by human experts. Some recent work shows that it is possible to automatically structure a neural network.

2. There is not much waste of neurons and links to solve the current task. Therefore, some large scale applications such as mushroom classification, image coding, and electronic yellow pages, etc. can be implemented on mini/personal computers without difficulty. The optimization, pruning, and simplification techniques make our method more practical for large scale applications.

3. There is no delay caused by the redundant neurons in conventional neural nets. Only a small part of the SGNT hierarchical structure needs to be searched to find the right place to update the network, and no reinforcement is necessary during the training phase. This provides the main speed advantage over most other learning methods, and makes it possible to apply our method to real time applications.

[2] Using the reciprocal of the number of attributes is a good rule of thumb we employ to make approximate judgment. Normally, second order information gains are larger than first order ones. We use $\frac{1}{\mathcal{N}^3}$ for first order information gains and $\frac{1}{\mathcal{N}^2}$ for second order ones

4. The method can be used for supervised, unsupervised or even hybrid learning tasks depending on whether the class information is available or not.

Acknowledgment

The permission of the director of Telecom Australia Research Laboratories to publish this paper is acknowledged.

References

[1] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[2] L. Fang, A.Jennings, W.Wen, K.Li, and T.Li. Unsupervised learning in a self-organizing tree. In *Proc. IJCNN'91 (International Joint Conf. on Neural Networks)*, Singapore, Nov. 1991.

[3] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[4] J. McClelland and D. Rumelhart. *Parallel Distributed Processing*, volume 2. The MIT Press, 1986.

[5] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[6] J. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221 – 234, 1987.

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

[8] D. Rumelhart, J. McClelland, and the PDP Rearch Group. *Parallel Distributed Processing*, volume 1. Cambridge, Mass. The MIT Press, 1986.

[9] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. Michalski, T. Mitchell, P. Pachowicz, R. Y., H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, December 1991.

[10] W. Wen, A. Jennings, and H. Liu. Self-generating neural networks and their applications to telecommunications. In *Proc. of ICCT'92 (International Conference on Communication Technology*, Sept. 1992.

[11] W. Wen, H. Liu, and A. Jennings. Self-generating neural networks. In *Proc. of IJCNN'92 Baltimore*, June 1992.

Learning method tested	Training time (in seconds)			Platform used for testing	Tester(s)
	M_1	M_2	M_3		
SGNT (supervised)	1.00	2.10	0.50	SUN SPARC 1	Wen et. al.
ID3	35.51	154.02	23.04	SUN SPARC 1^+	Kreuziger et.al.
ID3 no windowing	4.98	7.61	3.74	SUN SPARC 1^+	Kreuziger et.al.
ID5R	99.20	407.61	78.91	SUN SPARC 1^+	Kreuziger et.al.
AQR	4.17	9.45	4.00	SUN SPARC 1^+	Kreuziger et.al.
CN2	4.48	74.04	10.25	SUN SPARC 1^+	Kreuziger et.al.
Cascade correlation	5.11	7.75	12.27	DECSTATION 3100	Fahlman
Backprop.	10 to 30			SUN SPARC	Thrun

Table 7: Speed comparisons for MONK's problems among supervised learning methods