Knowledge-Based Systems, 2002, vol.15, no.8, pp.515-528.   @Elsevier

Hybrid Decision Tree

Zhi-Hua Zhou[*], Zhao-Qian Chen

National Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, P.R.China

Abstract

In this paper, a hybrid learning approach named HDT is proposed. HDT simulates human reasoning by using symbolic learning to do qualitative analysis and using neural learning to do subsequent quantitative analysis. It generates the trunk of a binary hybrid decision tree according to the binary information gain ratio criterion in an instance space defined by only original unordered attributes. If unordered attributes cannot further distinguish training examples falling into a leaf node whose diversity is beyond the diversity-threshold, then the node is marked as a dummy node. After all those dummy nodes are marked, a specific feedforward neural network named FANNC that is trained in an instance space defined by only original ordered attributes is exploited to accomplish the learning task. Moreover, this paper distinguishes three kinds of incremental learning tasks. Two incremental learning procedures designed for example-incremental learning with different storage requirements are provided, which enables HDT to deal gracefully with data sets where new data are frequently appended. Also a hypothesis-driven constructive induction mechanism is provided, which enables HDT to generate compact concept descriptions.

Keywords: Machine learning; Knowledge acquisition; Decision tree; Neural networks; Hybrid learning; Incremental learning; Constructive induction

1. Introduction

Decision trees and neural networks are alternative methodologies for pattern classification. Atlas et al. (1990) presented a performance comparison of those two methodologies used in load forecasting, power security and vowel recognition, and found that both methodologies have their own advantages. Some researchers (Minsky, 1991; Hendler, 1991; Sun et al., 1992) indicated that hybrid approaches can take advantage of both symbolic and connectionist models to attack tough problems. Therefore much research has addressed the issue of combining decision trees with neural networks.

Some approaches proposed in this area were motivated by the lack of a reliable procedure for determining the appropriate size of feedforward neural networks in practical applications. These approaches used decision trees to help to determine the topology of neural networks in order to facilitate learning and/or improve generalization by controlling the number of nodes and connections. Examples are as follows. Deffuant (1990) proposed an algorithm named NEURAL to generate a tree-structured network whose size and architecture were not specified before the learning process because the nodes were appended during the growth of the tree. Sethi (1990) proposed a procedure for mapping a decision tree into a multilayer feedforward neural network, which was used

[*] Corresponding author. Tel.: +86-25-359-3163; fax: +86-25-330-0710.
E-mail addresses: zhouzh@nju.edu.cn (Z.-H. Zhou).

to translate the knowledge represented by the decision tree into the architecture of a neural network whose connections could be retrained by a backpropagation algorithm. Sanger (1991) proposed a tree-structured adaptive network for function approximation in high-dimension spaces by employing a gradient-based learning procedure to grow a neural tree whose structure depended on both the input data and the function to be approximated. Li et al. (1992) proposed a competitive neural network named adaptive neural tree whose nodes were organized in a tree topology for classification and vector quantization by combining competitive learning principles with structural adaptation during learning. Kubat (1998) initialized a RBF network with decision trees that defined relatively pure regions in the instance space, each of which then determined one basis function.

Some approaches were motivated by the lack of a powerful procedure for determining the appropriate splits or tests of decision trees. These approaches used neural networks to refine the splits or even directly embeded neural networks functioning as splits in decision trees in order to improve generalization. Examples are as follows. Utgoff (1988) proposed perceptron tree in which the leaf nodes of a binary decision tree were realized with linear threshold units. Later, Utgoff and Brodley (1991) developed linear machine decision tree in which the internal nodes were linear machine (Nilsson, 1990) trained by a thermal training method (Frean, 1990). Sankar and Mammone (1991) presented a neural tree network that connected single-layered neural networks in a tree architecture where the networks were used to recursively partition the feature space into subregions. Guo and Gelfand (1992) used multilayer neural networks at the decision nodes of a binary classification tree to extract nonlinear features by employing a gradient-based learning algorithm in conjunction with a class-aggregation algorithm to train the networks and grow the tree. Jordan and Jacobs (1994) used a neural learning algorithm to determine the parameters of the classifier in their tree-structured hierarchical mixture of experts. Murthy et al. (1994) proposed an oblique decision tree algorithm OC1 that combined deterministic hill-climing with two forms of randomization to find a good oblique split at each internal node of a decision tree. Behnke and Karayiannis (1996) proposed a competitive neural tree that performed hierarchical clustering of the feature vectors and employed competitive learning at the node level. Sethi and Yoo (1997) exploited Sethi (1990)'s decision-tree-to-neural-networks mapping to generate a decision tree whose hierarchy of splits was determined in a global fashion by a neural learning algorithm. Apolloni et al. (1998) proposed a recurrent neural network to suggest the next move during the descent along the branches of a decision tree through providing the degree of membership of each possible move to the fuzzy set "good move". Suarez and Lutsko (1999) proposed a fuzzy decision tree by superimposing fuzzy structure over the skeleton of CART (Breiman et al., 1984), which was then trained by a procedure similar to backpropagation to globally optimize the fuzzy splits.

Also there were some approaches motivated by the lack of an intelligible procedure for interpreting the knowledge learned by neural networks. These approaches used decision trees to approximate the function of the trained neural networks in order to improve comprehensibility. Examples are as follows. Craven and Shavlik (1996) proposed an algorithm named TREPAN to induce an *M*-of-*N* decision tree that approximated the concept represented by a trained neural network. Krishnan et al. (1999) presented a methodology for extracting decision trees from input data generated from trained neural networks where genetic algorithm was used to query the trained network and a prototype selection mechanism was then used to extract the decision tree. Setiono and Liu (1999) constructed and pruned a multilayer feedforward neural network, built a decision tree from the hidden node activation values of the pruned network, then obtained an oblique decision tree by replacing each split at the nodes of the tree by a linear combination of original input attributes. Schmitz et al. (1999) proposed an algorithm named ANN-DT which used neural network to generate outputs for examples interpolated from the training data set and then extracted a univariate binary decision tree from the network.

In this paper, a novel machine learning approach named HDT (Hybrid Decision Tree) that virtually embeds feedforward neural network in some leaves of a binary decision tree is proposed, which is motivated by recognizing that dealing with unordered/ordered attributes is similar to performing qualitative/quantitative analysis. HDT employs unique techniques of tree growing, neural processing, incremental learning and constructive induction, which enables it to generate accurate and compact hybrid decision trees and deal gracefully with new appended data.

The rest of this paper is organized as follows. In Section 2, we present HDT approach, including the techniques of tree growing, neural processing, incremental learning and constructive induction. In Section 3, we give out some experimental results on comparing HDT with several other classifiers, and some experimental results on comparing different neural settings, incremental learning settings, and constructive induction settings of HDT. Finally in Section 4, we conclude.

2. HDT approach

2.1 Tree growing

At present, attribute-value pair representation is the most often used knowledge representation scheme due to its straightness and availability. A fundamental distinction among attributes is whether an attribute's values are ordered or not (Utgoff & Brodley, 1990). Accordingly, an *ordered attribute* is one whose possible values are totally ordered. This class includes continuous and integer-valued attributes. An *unordered attribute* is one whose values are not totally ordered. From cognitive perspective, processing unordered attributes could be viewed as performing qualitative analysis while processing ordered attributes could be viewed as performing quantitative analysis. In most cases, when we human beings solve problems, we perform qualitative analysis at first and resort to quantitative analysis only when the problem cannot be tackled by sole qualitative analysis. For example, when a production quality manager of a fabricator is required to manually examine the quality of iron cans, he/she often checks at first whether the iron cans are over large-grained, cracky, painted in error colors, etc., which acts as qualitative analysis. In cases those observations are enough for labeling the cans as defective works, no more check is needed. Only in cases those observations are not enough to draw a "defective" conclusion, digital measurements such as radius, height, weight, etc. are exploited, which acts as quantitative analysis.

In general, decision trees are good at dealing with unordered attributes. When they are used in dealing with ordered attributes, although some multivariate trees utilize linear combination tests (Breiman et al., 1984) or oblique splits (Murthy et al., 1994), traditional univariate trees exploit an explicit or implicit discretization procedure to cluster the numeric area to a set of subintervals, i.e. to transform the ordered attributes to unordered ones. Such discretization may lead to ill results due to following reasons. Firstly, most discretization procedures suffer from the user's bias, e.g. they may require the user provide values of some parameters such as distance-threshold or number of clusters, which is used in generating subintervals (Dougherty et al., 1995; Kohavi & Sahami, 1996). Secondly, since discretization is performed on a finite training set, it deserves doubts about whether the clustered subintervals encapsulate the real distribution. Moreover, since no information-lossless discretization procedure is available, some helpful information may lose in the transformation from infinite numeric area to finite subintervals. On the other hand, neural learning approaches such as feedforward neural networks are good at dealing with ordered attributes in most cases due to their powerful nonlinear processing ability. Therefore we believe that a strong learning paradigm can be attained if symbolic and neural learning are combined in such a way that the former is used to process unordered attributes and the latter is used subsequently to process ordered attributes when necessary.

Our paradigm, i.e. HDT, is implemented by virtually embedding feedforward neural networks in some leaves of a binary decision tree that is grown in a well known way described as follows. In each step of the learning process, training examples falling into current node are partitioned into two subsets according to a selected test, i.e. a split. Examples satisfying the split enter the left branch while the rest ones enter the right branch of current node. Splits are selected according to the binary information gain ratio that is similar to C4.5's information gain ratio (Quinlan, 1993) except in two aspects.

Firstly, splits of HDT are selected based upon only unordered attributes. In other words, ordered attributes play no role in split selection. Suppose training set $\mathcal{S}$ is originally described in an instance space $\mathcal{L}$ = <$attr_1$, ⋯, $attr_i$, $attr_{i+1}$, ⋯, $attr_n$> where $attr_1$ to $attr_i$ are unordered attributes and $attr_{i+1}$ to $attr_n$ are ordered attributes. In split selection, $\mathcal{L}$ is implicitly transformed to $\mathcal{L}_0$ = <$attr_1$, ⋯, $attr_i$> by ignoring all original ordered attributes and the splits are generated in $\mathcal{L}_0$.

Secondly, splits of HDT are selected based upon attribute-value pair instead of only attribute so that a binary tree instead of a multi-nary tree is generated. Suppose attribute $attr_k$ has m possible values, i.e. <$attrval_{k1}$, ⋯, $attrval_{km}$>. In some multi-nary tree algorithms, <$attr_k$> may constitute a split so that m branches respectively corresponding to the m possible values of $attr_k$ are generated. But in HDT, <$attr_k = attrval_{kj}$> instead of <$attr_k$> may constitute an equality split so that only two branches respectively corresponding to the answer "yes" or "no" to the question "Does the example take value $attrval_{kl}$ at attribute $attr_k$?" are generated.

A leaf node is generated when either of the following situations occurs. The first situation is that all training examples falling into current node belong to a same class. In other words, a node is split as long as there are examples that belong to different classes. Since in most cases this claim is so strong to aggravate overfitting, a prepruning strategy that relaxes the claim is provided. In detail, when current node is to split, its diversity is measured and compared with a pre-set diversity-threshold. If its diversity is larger than the diversity-threshold, then current node is split. Otherwise the learning process terminates and future examples falling into current node are classified to the most probable class of current node, i.e. the class that has the most number of training examples in current node. Here diversity is measured as the proportion of the training examples that does not belong to the most probable class of the node. Suppose N training examples fall into current node, in which N_i belong to class i, and A is the most probable class of current node, i.e.

$$N_A = \underset{i}{MAX}\, N_i \qquad \text{where} \qquad N = \underset{i}{\Sigma}\, N_i \ ,$$

then the diversity of current node is measured as $(1 - N_A / N)$. Note that it is also possible to grow the tree to the maximum depth then employ postpruning strategy to reduce the tree. In most cases the result of postpruning is more accurate than that of prepruning but there are also cases where the former is easier to suffer overfitting than the latter (Mingers, 1989). Since postpruning is hard to be used in an incremental setting, here HDT employs prepruning.

The second situation for the generation of a leaf node is that although a large amount of training examples fall into current node and the diversity of current node is larger than the diversity-threshold, current node cannot be split because those training examples cannot be further distinguished in the instance space $\mathcal{L}_0$. An extreme of this situation is that all the training examples take same values at each unordered attribute, that is, there is only one unordered attribute value vector <$attrval_1$, ⋯, $attrval_i$> corresponding to the unordered attribute vector <$attr_1$, ⋯, $attr_i$> appears in current node. This is very likely to occur when the number of unordered attributes is too small relative to the complexity of the problem so that all unordered attributes have already been positively used in the branch leading to current node. Here we say an unordered attribute is *positively used* if a split involving this attribute has already appeared in the branch and the answer to the corresponding question is "yes".

For example, suppose there are only two unordered attributes, i.e. $attr_1$ and $attr_2$, can be used to distinguish a quite large training set, and during the learning process a tree is generated, which is depicted in Fig. 1. Now, when considering node C, we will find that no unordered attribute can be exploited to constitute a valid split because all the unordered attributes have already been positively used in the branch A-B-C. In other words, all the unordered attributes are helpless in further splitting node C. However, since there are a large amount of training examples belong to different classes, node C must be further processed. In HDT, when such situation appears, a virtual neural node is generated and later a specific feedforward neural network is used to accomplish the learning task in an instance space $\mathcal{L}_1$ that is defined by only ordered attributes, which is detailedly described in Section 2.2.

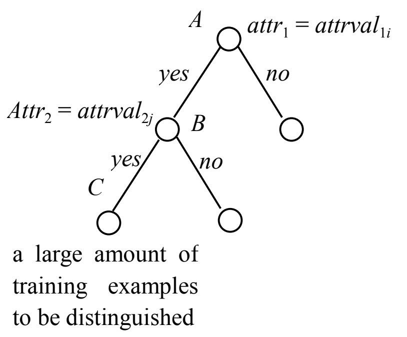


Fig. 1 A virtual neural node is generated at C

The tree growing process of HDT is sketched in Table 1, which involves an iterative procedure **TreeGrow ($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)**.

Table 1 Tree growing of HDT

```
TreeGrow ($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)

Input:    training set $\mathcal{S}$
          original instance space $\mathcal{L}$
          diversity-threshold $\mathcal{D}$

Output:   hybrid decision tree $\mathcal{T}$

Procedure:
Initialize($\mathcal{T}$, $\mathcal{S}$);              /* $\mathcal{S}$ is the training set of current node */
$\delta$= Diversity($\mathcal{S}$);             /* measure the diversity of current node */
if ($\delta < \mathcal{D}$ )
    LeafNode ($\mathcal{S}$);                /* a leaf node is generated */
    Return($\mathcal{T}$);
else
    $\mathcal{L}_0$ = UnorderedOnly($\mathcal{L}$);      /* $\mathcal{L}_0$ is the instance space described by only original */
                                      /* unordered attributes */
    if CannotDistinguish($\mathcal{S}$, $\mathcal{L}_0$)   /* if training examples falling into current node cannot be */
                                      /* further distinguished in instance space $\mathcal{L}_0$ */
        MarkNeuralNode($\mathcal{S}$);       /* a virtual neural node is marked */
        return($\mathcal{T}$);
    else
        $\mathcal{E}$ = SelectSplit ($\mathcal{S}$, $\mathcal{L}_0$);   /* a split is selected according to binary information gain ratio */
        Split ($\mathcal{E}$, $\mathcal{S}$, $\mathcal{S}_L$, $\mathcal{S}_R$);      /* two subnodes are generated whose training set is respective */
                                      /* $\mathcal{S}_L$ and $\mathcal{S}_R$ */
        LeftSon($\mathcal{T}$) = TreeGrow ($\mathcal{S}_L$, $\mathcal{L}$, $\mathcal{D}$);     /* process left-son of current node */
        RightSon($\mathcal{T}$) = TreeGrow ($\mathcal{S}_R$, $\mathcal{L}$, $\mathcal{D}$);    /* process right-son of current node */
```

2.2 Neural processing

As stated in Section 1, currently there are many approaches that embed neural nodes in a decision tree, in which two embedding ways, i.e. either all the nodes of the tree or all the leaf nodes of the tree are neural nodes, are often used. However, HDT adopts neither of those two ways. Instead, it virtually embeds feedforward neural network in leaf nodes where and only where neural processing is necessary. In other words, only a part of leaf nodes in the tree are marked as virtual neural nodes. When an example falls into such a node, it is re-represented in an instance space $\mathcal{L}_1$ that is defined by only ordered attributes. In the supposal used in Section 2.1, the examples are originally described in instance space $\mathcal{L} = <attr_1, \cdots, attr_i, attr_{i+1}, \cdots, attr_n>$ where $attr_1$ to $attr_i$ are unordered attributes and $attr_{i+1}$ to $attr_n$ are ordered attributes. In this case $\mathcal{L}_1 = <attr_{i+1}, \cdots, attr_n>$ is got by ignoring all original unordered attributes. Moreover, all attributes in $\mathcal{L}_1$ are normalized to facilitate neural learning. Suppose the possible value range of ordered attribute $attr_k$ is ($attrval_{k,min}$, $attrval_{k,max}$), and the real value that example e takes at $attr_k$ is $attrval_{ke}$, then the normalized value of $attrval_{ke}$ is

$$Normalize(attrval_{ke}) = \frac{attrval_{ke} - attrval_{k,\min}}{attrval_{k,\max} - attrval_{k,\min}}$$

Above is a quite simple linear normalization scheme. Other normalization schemes such as standard normal normalization can also be used here. Moreover, we believe that if a priori knowledge on the value distribution of the attributes is available, then a knowledge-based normalization can be more helpful.

Two troublesome obstacles appear when we consider the marked neural nodes. Firstly, in most cases only a small number of training examples falling into those nodes. This is because the original training set has been partitioned to many subsets each belongs to a leaf node of the tree. If some prevailing neural algorithms such as BP (Rumelhart et al., 1986) are used to train those nodes one by one, then few networks can generalize well due to the lack of enough training examples. Secondly, the speed advantage of decision tree will be blotted out due to the iterative training process of most neural algorithms. The solution of HDT is to utilize a specific feedforward neural network named FANNC (Zhou et al., 2000) to train the marked neural nodes.

FANNC is a fast adaptive neural classifier that exploits the advantages of both Adaptive Resonance Theory (Grossberg, 1976) and Field Theory (Wasserman, 1993). It needs only one-pass learning and achieves not only high predictive accuracy but also fast learning speed. Besides, FANNC has incremental learning ability. When new training examples are fed, it does not retrain the entire training set. Instead, it can learn the knowledge encoded in those new examples through modifying some parameters associated with existing hidden units, or slightly adjusting the network topology by adaptively appending one or two hidden units and relevant connections to existing network. Since the network architecture is adaptively set up, the disadvantage of manually determining the number of hidden units of most feedforward neural networks is overcome. Benchmark tests show that FANNC works well even when only a few training examples are provided (Zhou et al., 2000). Therefore FANNC is strong and fast enough to cope with the two obstacles that we stated in the beginning of this paragraph. Moreover, the incremental learning ability of FANNC is the key of that of HDT, which is introduced in the next section. Note that there are many other neural classifiers having the ability of dynamically adjusting network topology (Ash, 1989; Fahlman, 1990). However, this is not the prerequisite for determining that whether a neural classifier fits for hybrid decision tree or not. Instead, the prerequisite is that the neural classifier having both strong generalization ability and fast learning speed. Also it is better for the neural classifier to have incremental learning ability.

Another technique may be helpful to alleviate the lack of enough training examples, that is, collecting all the training examples falling into marked neural nodes and training a sole collective neural network instead of training multiple individual networks each corresponding to a marked neural node. During the growth of the hybrid decision tree, the marked neural nodes are regarded as dummy nodes. After all the dummy nodes are marked, training examples falling into them are collected together to train a neural network. Any future examples falling into dummy nodes are classified by the trained collective network. This is just the reason that why we call the dummy nodes as virtual neural nodes and say that HDT virtually embeds neural network in decision tree.

Such collective setting is similar to decision graph (Oliver, 1993) to some extent, where duplicated subtrees are merged with a *Join* operator to solve the *replication* (Pagallo & Haussler, 1990) and *fragmentation* (Oliver, 1993) problems of decision trees. Note that although it is very unusual in real-world applications, there does exist possibility that all the leaf nodes of the hybrid decision tree are marked neural nodes. In such a case, the splits in the trunk of the tree are in fact useless in distinguishing the training examples. Thus the entire tree degenerates to a collective neural network so that any future examples are directly fed to the network without matching with the trunk of the tree.

2.3 Incremental learning

Incremental learning ability is very important to machine learning approaches designed for solving real-world problems due to two reasons. Firstly, it is almost impossible to collect all helpful training examples before the trained system being in use. Therefore when new examples are fed, the learning approach should have the ability of doing some revisions on the trained system so that unlearned knowledge encoded in those new examples can be incorporated. Secondly, modifying a trained system may be cheaper in time cost than building a new system from scratch, which is useful especially in real-time applications.

In the last two decades much research has addressed the issue of endowing decision trees with incremental learning ability. Examples are as follows. Schlimmer and Fisher (1986) proposed an algorithm named ID4 which incrementally generated a decision tree by updating the splits that were not best when considering new examples along with previous examples, discarding and re-generating the subtrees expanded from those splits. Utgoff (1989) presented ID5R which incrementally generated a tree identical to that could be generated by ID3 (Quinlan, 1986) through using a pull-up process to move expected splits to the roots of subtrees when new examples were provided. Crawford (1989) designed an incremental version of CART, which generated a new subtree from scratch when a new example was provided and the old split was to be updated. Utgoff and Brodley (1990) proposed an incremental perceptron tree algorithm named PT2 which adjusted the linear threshold units embedded in the tree and then searched for a linear threshold unit based on a reduced set of original input attributes. Kalles and Morris (1996) presented TDIDT algorithm which improved ID5R by evaluating the quality of attributes selected at the nodes of the decision tree and estimating a minimum number of steps for which those attributes were guaranteed such a selection. Lovell and Bradley (1996) developed an algorithm named MSC that incrementally refined a decision tree with limited backtracking, whose performance was sensitive to the order in which the training examples are provided. Utgoff et al. (1997) designed an incremental tree induction algorithm ITI based upon a tree revision mechanism that updated the splits in the branch where the new example passed down. Hamzei and Mulvaney (Hamzei & Mulvaney, 1999) proposed an incremental fuzzy decision tree that combined decision tree ITI-2.8 and fuzzy logic for concept learning and suitable reasoning in the presence of noisy and imprecise input data. McSherry (1999) presented an incremental decision tree induction algorithm whose attribute selection was based upon the evidence-gathering strategies used by doctors in sequential diagnosis. Fern and Givan (2000) extended ID4 through adopting three strategies, i.e. advanced warm-up, postpruning by subtree monitoring, and feature-switch suppression by subtree monitoring, and then used the extension algorithm in investigating online ensemble learning.

Most approaches perform incremental learning with the help of saved previous training examples. They usually examine new training examples along with saved previous ones to see whether a split should be updated and to determine which is the best split at present. However, in our opinion, exploiting saved previous training examples is not a good solution. The reason is that in most cases where incremental learning is necessary, saving all the used training examples is too luxury to realize. For example, suppose we have an electronic lock that distinguishes family members from strangers by a learning system performing face recognition. Also suppose the learning system has incremental learning ability so that the more time the lock is used, the more reliable it is. Now, consider the problem that can the lock system store all the used training examples, i.e. face images, to facilitate its incremental learning? The answer is definitely negative because the storage of the system may be flooded by emerging examples soon. So, we believe that incremental learning should be performed in the situation where none or only a few previous training examples are available.

On the other hand, most works on incremental learning by far is to address problems caused by new training examples, which is called E-IL (Example-Incremental Learning) task here. We believe that due to the progressing of machine learning, now it is the time to consider other kinds of incremental learning tasks. Here we distinguish three tasks as:

E-IL (Example-Incremental Learning): New training examples are provided after a learning system being trained. The trained system should be modified so that unlearned knowledge encoded in those new examples can be incorporated without sacrificing learned knowledge too much. None or only a few previous training examples are available. For example, the electronic lock problem described above is an E-IL task.

C-IL (Class-Incremental Learning): New output classes are provided after a learning system being trained. The trained system should be modified so that knowledge on new classes can be incorporated without sacrificing learned knowledge too much. None or only a few previous training examples are available. For example, suppose a new lodger moves into the house, the electronic lock system should have the ability of distinguishing him along with old lodgers from strangers without re-configuring and/or retraining the entire system.

A-IL (Attribute-Incremental Learning): New input attributes are provided after a learning system being trained. The trained system should be modified so that knowledge on new attributes can be incorporated without sacrificing learned knowledge too much. None or only a few previous training examples are available. For example, suppose a new face feature is provided to the electronic lock system by a new auxiliary equipment that was bought yesterday, the electronic lock should have the ability of utilizing the new feature along with old ones without re-configuring and/or retraining the entire system.

We believe that the most difficult incremental learning task is the one where E-IL, C-IL and A-IL should be simultaneously performed. Although C-IL and A-IL have not received much attention by far, we believe that they will attract many researchers in the near future due to their importance.

Only a few works (Utgoff & Brodley, 1990) addressed the issue of endowing hybrid algorithms that combine decision tree with neural networks with incremental learning ability. Here in HDT, two incremental learning procedures, i.e. **Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$)** and **Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, $\{\mathcal{S}_i\}$)**, designed for E-IL are provided, which enables HDT to deal gracefully with data sets where new data are frequently appended.

Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$) is designed for the situation where no previous training examples are available. In this procedure, the new example passes down the tree until a leaf node is arrived. If the leaf node is not a virtual neural node, then nothing happens even when the new example belongs to a different class to the class of the node because it is hard to find an appropriate split according to only the new example without the help of previous training examples. If the leaf node is a virtual neural node, then the new example is re-represented in instance space $\mathcal{L}_1$ and fed to the trained FANNC network so that unlearned knowledge encoded in the new example is incorporated into the tree through modifying some parameters of the FANNC network or appending one or two hidden units to the FANNC network.

Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, $\{\mathcal{S}_i\}$) is designed for the situation where only a few previous training examples are available. In this procedure, previous training examples that fall into non-neural leaf nodes whose diversity is beyond zero are saved. The new example passes down the tree until a leaf node is arrived. If the node is a virtual neural node, then the new example is re-represented in instance space $\mathcal{L}_1$ and fed to the trained FANNC network

Table 2 Incremental learning of HDT

```
Incre_noexp ($\mathcal{T}_0$, $e$, $\mathcal{L}$)

Input:   hybrid decision tree $\mathcal{T}_0$        /* $\mathcal{T}_0$ is a primitive tree generated before incremental learning */
         new training example $e$
         original instance space $\mathcal{L}$

Output:  hybrid decision tree $\mathcal{T}$

Procedure:
$\mathcal{Q}$ = MatchToLeaf($e$, $\mathcal{T}_0$);            /* match $e$ with $\mathcal{T}_0$ until a leaf node $\mathcal{Q}$ is arrived */
if BeNeuralNode($\mathcal{Q}$)                   /* if the leaf node $\mathcal{Q}$ is a virtual neural node */
    $\mathcal{L}_1$ = OrderedOnly($\mathcal{L}$);            /* $\mathcal{L}_1$ is the instance space described by only original */
                                         /* ordered attributes */
    $\mathcal{T}$ = FANNC_Incre($e$, $\mathcal{L}_1$);         /* utilize the incremental learning ability of FANNC to incorporate */
                                         /* unlearned knowledge into the tree */
    return($\mathcal{T}$);
else
    return($\mathcal{T}_0$);


Incre_fewexp ($\mathcal{T}_0$, $e$, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})

Input:   hybrid decision tree $\mathcal{T}_0$
         new training example $e$
         original instance space $\mathcal{L}$
         diversity-threshold $\mathcal{D}$
         set of training subsets {$\mathcal{S}_i$}   /* $\mathcal{S}_i$ keeps previous training examples belonging to the $i$-th */
                                         /* non-neural leaf node whose diversity is beyond zero */
Output:  hybrid decision tree $\mathcal{T}$

Procedure:
$\mathcal{Q}$ = MatchToLeaf($e$, $\mathcal{T}_0$);
if BeNeuralNode($\mathcal{Q}$)
    $\mathcal{L}_1$ = OrderedOnly($\mathcal{L}$);
    $\mathcal{T}$ = FANNC_Incre($e$, $\mathcal{L}_1$);
    return($\mathcal{T}$);
else
    $\mathcal{S}'$ = FindSavedSet($\mathcal{Q}$, {$\mathcal{S}_i$})      /* retrieve previous training examples of $\mathcal{Q}$ from saved subsets */
    if ($\mathcal{S}'$ = Ø )                      /* no previous examples of $\mathcal{Q}$ are saved because its diversity is zero */
        return($\mathcal{T}_0$);                  /* nothing happens */
    else
        $\mathcal{S}''$ = $\mathcal{S}'$ ∪ {$e$};               /* add $e$ to $\mathcal{S}'$ */
        $\mathcal{T}$ = TreeGrow($\mathcal{S}''$, $\mathcal{L}$, $\mathcal{D}$);     /* split leaf node $\mathcal{Q}$ further */
        TrimSavedExp({$\mathcal{S}_i$});          /* trim {$\mathcal{S}_i$}, including delete $\mathcal{S}'$ and add training sets corresponding */
                                         /* to new generated non-neural leaf nodes whose diversity */
                                         /* is beyond zero */
        return($\mathcal{T}$);
```

so that unlearned knowledge encoded in the new example is incorporated into the tree through modifying some parameters of the FANNC network or appending one or two hidden units to the FANNC network. If the node is a non-neural node whose diversity is zero, then nothing happens because it is hard to find an appropriate split according to only the new example without the help of previous training examples. Otherwise the new example is utilized along with the saved previous training examples to split current node.

Since both **Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$)** and **Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** operate on only leaf nodes, HDT has noise-tolerant ability to a certain extent, which is explained as follows. For most incremental tree induction algorithms, when a new example is fed, some splits appearing in the branch where the new example passes down are updated. If the new example is noisy data, then the error caused by it is introduced to the trunk of the tree. But in HDT the error is only introduced to the leaves. Since a tree in use is matched from root to trunk to leaves, error in trunk is obviously more serious than that in leaves.

Note that both **Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$)** and **Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** are sensitive to the order in which the training examples are provided. This is because there are virtual neural nodes in the tree. Since neither procedure saves previous training examples falling into virtual neural nodes, it is hard to split a virtual neural node to child nodes. An extreme example is that if the first two training examples cannot be distinguished by unordered attributes, then the tree degenerates to a virtual neural node. So, in order to avoid crazy tree topologies, hybrid decision tree cannot be generated entirely in a pure incremental way, i.e. each time learning only one example. In fact, HDT employs a unique two-phase incremental learning strategy. In the first phase, a training set is exploited to generate a primitive tree. Then in the second phase, pure incremental learning is performed on the primitive tree. This strategy is practical because almost in all learning tasks we can collect quite a number of training examples before the pure incremental learning begins. For example, suppose we want to build a fault detection system with incremental learning ability for supervising a real-time production line. It is usual that before the final system being used, i.e. before the pure incremental learning being performed, a lot of training examples can be collected from history records of the production line or even from the debugging of the prototype of the system. It is obvious that we should not ignore the existence of those helpful training examples. Then, why not utilize those examples to establish a good basis for incremental learning?

Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$) and **Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** are sketched in Table 2.

2.4 Constructive induction

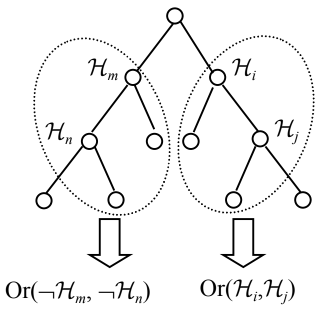
Traditional inductive algorithms are selective, whose performance are poor when provided attributes are inappropriate for the target concept. An improvement is to have the learning approaches construct new attributes automatically, which is called constructive induction (Michalski, 1983). This technique focuses on improving the representation by adding additional task-relevant attributes and/or deleting irrelevant ones from the instance space, so that the learning tasks become easier to be performed or the learning results are improved. In general, there are three kinds of constructive induction schemes that are classified according to the information used in searching for the best representation space (Bloedorn & Michalski, 1998), that is, data-driven constructive induction that uses input examples, hypothesis-driven constructive induction that uses intermediate hypotheses, and knowledge-driven constructive induction that uses domain knowledge provided by an expert. Also there are multi-strategy constructive induction methods such as AQ17-MCI (Bloedorn, 1993) that use two or more those schemes.

In the past decade, much research has addressed the issue of endowing decision trees with constructive induction ability. Examples are as follows. Pagallo and Haussler (1990) identified the *replication* problem, which is a representational shortcoming of decision trees, and then developed methods that adaptively constructed new attributes from original attributes so that relatively small decision trees could be generated in learning Boolean functions with a short DNF representation. Matheus and Rendell (1989) presented a general constructive induction frame and applied the frame to a decision tree algorithm named CITRE, which repeatedly constructed new attributes and generated new trees until no new attribute could be constructed. Gama (1997) proposed an oblique linear tree algorithm named Ltree, which inserted new attributes that were projections of the examples falling into the node over the hyperplane defined by a linear discriminant function at each decision node. Zheng

(1998) designed a path-based method that dynamically determined conditions for constructing new conjunction attributes by carrying out a systematic search with pruning over each path of the tree.

However, few works has addressed the issue of endowing hybrid algorithms that combine decision trees with neural networks with constructive induction ability. In HDT, a simple hypothesis-driven constructive induction mechanism is provided. For the convenience of explanation, here we introduce a notion *homo-leaves* defined as leaf nodes that are both virtual neural nodes or both non-neural nodes that belong to a same class. For example, if both leaf nodes *A* and *B* are virtual neural nodes, then *A* and *B* are homo-leaves; if both leaf nodes *C* and *D* are non-neural nodes that are labeled to a same class during learning, then *C* and *D* are also homo-leaves.

The construction of new attributes is illustrated in Fig. 2. A primitive tree is traversed and adjacent homo-leaves are detected. For a non-leaf node, if its right-son and its left-son's right-son are homo-leaves, then a new attribute is constructed via disjuncting the negation of the split condition of the node and that of its left-son; if its left-son and its right-son's left-son are homo-leaves, then a new attribute is constructed via disjuncting the split condition of the node and that of its right-son.



$\mathcal{H}_m$
$\mathcal{H}_i$
$\mathcal{H}_n$
$\mathcal{H}_j$
Or($\neg\mathcal{H}_m$, $\neg\mathcal{H}_n$)
Or($\mathcal{H}_i$,$\mathcal{H}_j$)

Fig. 2 Constructing new attributes

It is obvious that the constructed new attributes are all binary unordered attributes in the form of "<$attr_i$ =/≠ $attrval_{iu}$> OR <$attr_j$ =/≠ $attrval_{jv}$>" whose possible attribute values are only "yes" or "no". After the new attributes are all constructed from the primitive tree, the original instance space $\mathcal{L}$ is transformed to a new instance space $\mathcal{L}'$ by appending new attributes to the attribute set. Then a new tree is generated in $\mathcal{L}'$ and the training set accuracy of the new tree is measured. If the accuracy of the new tree is worse than that of the primitive tree, then the construction process terminates and the primitive tree is regarded as the learning result. Otherwise the new tree is viewed as primitive tree and the construction process repeats. Here we use training set accuracy as the termination criterion because HDT is a universal learning approach where no specific domain knowledge is available. Since in some cases such a criterion will aggravate overfitting, when enough training examples are provided, a better alternative is to keep a validation set that does not be used in training and then compare trees according to the validation set accuracy instead of the training set accuracy. The constructive induction procedure of HDT is sketched in Table 3.

Note that since **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** repeatedly generates new attributes and new trees, all training examples must be saved during constructive induction. So, **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** can only be performed in the first phase of the incremental learning process of HDT because neither **Incre_noexp($\mathcal{T}_0$, *e*, $\mathcal{L}$)** nor **Incre_fewexp($\mathcal{T}_0$, *e*, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** saves all previous training examples. In other words, there is no cooperation between constructive induction and pure incremental learning in HDT. Another issue to be noted is that although **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** is useful in generating compact trees so that the test time cost of HDT is reduced, the training time cost is increased especially when the construction process repeats for many times. Fortunately, if the user prefers training time cost to test time cost, then he/she can easily modify **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** by using a parameter to control the maximum number of cycles that can be repeated in constructive induction so that the training time cost is limited.

3. Experiments and comparisons

3.1 Summary

In this section, we perform four series of experiments on HDT. Firstly, we compare HDT with some other

Table 3 Constructive induction of HDT

```
Construct ($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)

Input:   training set $\mathcal{S}$
         original instance space $\mathcal{L}$
         diversity-threshold $\mathcal{D}$

Output:  hybrid decision tree $\mathcal{T}$

Procedure:
$\mathcal{T}_0$ = TreeGrow($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$);     /* generate a primitive tree $\mathcal{T}_0$ */
$\alpha_0$ = Accuracy($\mathcal{T}_0$, $\mathcal{S}$);          /* measure the accuracy of $\mathcal{T}_0$ */
while true                       /* repeatedly perform constructive induction till termination */
do
    NewAttrSet = $\varnothing$;
    while(node = Traverse($\mathcal{T}_0$))    /* check each node of $\mathcal{T}_0$ via traversing the tree */
    do
        if (HomoLeaves(RightSon(node), RightSon(LeftSon(node))))
            attr = Or(Not(node), Not(LeftSon(node)));          /* construct new attribute */
            NewAttrSet = NewAttrSet $\cup$ {attr};
        else
            if(HomoLeaves(LeftSon(node), LeftSon(RightSon(node))))
                attr = Or(node, RightSon(node));               /* construct new attribute */
                NewAttrSet = NewAttrSet $\cup$ {attr};
    if(NewAttrSet = $\varnothing$)             /* if no new attribute can be constructed */
        return($\mathcal{T}_0$);
    else
        $\mathcal{L}'$ = Append($\mathcal{L}$, NewAttrSet);
        $\mathcal{T}$ = TreeGrow($\mathcal{S}$, $\mathcal{L}'$, $\mathcal{D}$);   /* generate a new tree with the help of new attributes */
        $\alpha$ = Accuracy($\mathcal{T}$, $\mathcal{S}$);
        if($\alpha \geq \alpha_0$)
            $\mathcal{T}_0$ = $\mathcal{T}$;
            $\alpha_0$ = $\alpha$;
        else           /* if the accuracy of the new tree is worse than that of the primitive tree */
            return($\mathcal{T}_0$);
```

classifiers including C4.5 decision tree, BP neural network and FANNC neural network. Secondly, we compare the performance of different neural settings of HDT, including collective/individual FANNC/BP. Thirdly, we compare the performance of different incremental learning settings of HDT. Finally, we compare the performance of HDTs with or without constructive induction.

All the data sets are from UCI machine learning repository (Blake et al., 1998) and are designed for classification tasks. Examples that have missing values are removed. Information of the data sets is tabulated in Table 4, where the abbreviations of the names of data sets are provided for future use. The experimental machine is a PC of Pentium 350MHz, 128MB RAM. The C4.5 software is available from http://www.rulequest.com. The BP, FANNC and HDT are coded with Visual C++ 6.0 on Windows NT.

The BP networks used in the experiments are with single hidden layer and trained with SuperSAB (Tollenaere, 1990) algorithm that is one of the fastest variations of BP. Tollenaere (1990) reported that SuperSAB is 10 – 100 times faster than standard BP (Rumelhart et al., 1986). In our experiments, the parameters of SuperSAB are set to the values suggested by Wasserman (1993), that is, the maximum weight step η_{ij} is always set to 10, the weight increase factor η_{up} and weight decrease factor η_{down} are respectively set to 1.05 and 0.2. In order to avoid overfitting, we use a validation set constructed via bootstrap sampling from training set. The training process is

Table 4 Data sets used in experiments

data set		size	class	attributes		
abr.	original name			total	ordered	unordered
aut	Automobile	159	7	25	15	10
bal	Balance scale	625	3	4	4	0
con	Congressional voting records	232	2	16	0	16
gla	Glass identification	214	6	9	9	0
hep	Hepatitis	80	2	19	6	13
ion	Ionosphere	351	2	34	34	0
iri	Iris plant	150	3	4	4	0
liv	Liver disorders	345	2	6	6	0
pim	Pima indians diabetes	768	2	8	8	0
sau	Statlog Australian credit approval	690	2	15	6	9
sge	Statlog German credit	1,000	2	20	7	13
she	Stalog heart disease	270	2	13	7	6
sve	Statlog vehicle silhouettes	846	4	18	18	0
win	Wine recognition	178	3	13	13	0
wis	Wisconsin breast cancer	683	2	9	9	0

terminated when there is no validation set accuracy improvement in consecutive 5 epochs.

The parameters of FANNC are set to default values (Zhou et al., 2000), that is, the default responsive characteristic width α_{ij} of *Gaussian* weight is set to 0.25, the bias of the hidden layer unit is set to 0.3, the responsive center adjustment step δ is set to 0.1, the leakage competition threshold is set to 0.8, and the maximum allowable error is set to 0.11.

3.2 HDT vs. other classifiers

In this section, we compare the performance of HDT with C4.5, BP and FANNC. Here HDT is with its default setting, i.e. with a sole collective FANNC network and with constructive induction. The diversity-threshold of HDT is set to zero. In each experiment, a validation set is used to control the constructive induction of HDT,

Table 5 Comparisons of HDT vs. other classifiers

data	HDT		C4.5		BP		FANNC	
set	time	accuracy	time	accuracy	time	accuracy	time	accuracy
aut	1.00	0.8323	0.07	0.8326	74.49	0.7624	4.33	0.8012
bal	1.00	0.8416	0.01	0.7780	3.16	0.9120	0.99	0.8416
con	1.00	0.9612	1.00	0.9612	723.00	0.9054	186.67	0.8592
gla	1.00	0.6415	0.02	0.6734	13.60	0.6114	0.98	0.6415
hep	1.00	0.8875	0.44	0.8222	72.35	0.8252	28.03	0.8628
ion	1.00	0.9057	0.03	0.8716	4.93	0.8732	0.99	0.9057
iri	1.00	0.9398	0.02	0.9468	1.56	0.9362	0.98	0.9398
liv	1.00	0.8812	0.02	0.6866	1.49	0.6924	0.99	0.8812
pim	1.00	0.7076	0.02	0.7426	3.65	0.7106	0.99	0.7076
sau	1.00	0.8768	0.02	0.8434	50.02	0.8464	5.21	0.8410
sge	1.00	0.7050	0.17	0.7060	535.04	0.7100	69.91	0.7070
she	1.00	0.7963	0.03	0.7982	10.33	0.7680	2.66	0.7680
sve	1.00	0.6526	0.01	0.7276	13.26	0.6528	0.99	0.6526
win	1.00	0.9656	0.03	0.9254	1.84	0.9690	0.98	0.9656
wis	1.00	0.9560	0.01	0.9300	1.35	0.9606	0.99	0.9560

which is generated via bootstrap sampling from training set. Experimental results are tabulated in Table 5, where *time* denotes relative training time cost measuring against the training time of HDT that is defined to be 1.00, and *accuracy* is attained by 10-fold cross validation. Since the number of hidden units of BP is difficult to determine, in each experiment three networks that respectively have 5, 10 and 15 hidden units are trained. The data of BP cited in Table 5 belongs to the network with the best accuracy.

The rank of classifiers according to accuracy and training time cost are respectively shown in Table 6 and Table 7. Items without significant difference are labeled to a same rank, e.g. accuracy of all the four classifiers on *iri* data set are not significantly different so that they are all ranked as first.

Table 6 Rank of classifiers according to accuracy

data set	HDT	C4.5	BP	FANNC
aut	1	1	4	3
bal	2	4	1	2
con	1	1	3	4
gla	2	1	4	2
hep	1	3	3	2
ion	1	3	3	1
iri	1	1	1	1
liv	1	3	3	1
pim	2	1	2	2
sau	1	2	2	2
sge	1	1	1	1
she	1	1	3	3
sve	2	1	2	2
win	1	4	1	1
wis	1	4	1	1
ave.	1.27	2.07	2.27	1.87

Table 7 Rank of classifiers according to training time cost

data set	HDT	C4.5	BP	FANNC
aut	2	1	4	3
bal	2	1	4	2
con	1	1	4	3
gla	2	1	4	2
hep	2	1	4	3
ion	2	1	4	2
iri	2	1	4	2
liv	2	1	4	2
pim	2	1	4	2
sau	2	1	4	3
sge	2	1	4	3
she	2	1	4	3
sve	2	1	4	2
win	2	1	4	2
wis	2	1	4	2
ave.	1.93	1	4	2.4

Table 5 to Table 7 show that HDT plays quite well. According to accuracy, it ranks first for 11 times, which is superior to C4.5 (8 times), BP (5 times), and FANNC (6 times). Moreover, it neither ranks last, which is also superior to C4.5 (3 times), BP (2 times), and FANNC (1 time). This means that the generalization ability of HDT is good in most cases. According to training time cost, C4.5 is definitely the winner because it always ranks first. However, we believe that the speed of HDT is acceptable because its time cost is in the order of seconds, e.g. the training time cost is 3.8 seconds on *pim* and 5.7 seconds on *sge*. Moreover, considering that the code for HDT has not been optimized, the gap between the time cost of HDT and that of C4.5 may be less than that is observed in Table 5.

3.3 Collective vs. individual and FANNC vs. BP

In this section, we compare the performance of HDT with different kinds of neural settings, including collective/ individual FANNC/BP. The diversity-threshold of HDT is set to zero. In each experiment, a validation set is used to control the constructive induction of HDT, which is generated via bootstrap sampling from training set. Table 8 shows the percentage of training examples falling into virtual neural nodes (*neural per*) and the accuracy of HDT that is attained by 10-fold cross validation. Here we do not record the training time cost due to two reasons. Firstly, Table 5 has already revealed that FANNC is far more faster than BP. Secondly, since the learning of FANNC is performed in an incremental style where each example is fed in only one pass and the training set of the collective FANNC is just the union of the training sets of those individual FANNCs, the time cost of training a sole collective FANNC and that of training multiple individual FANNCs are not significantly different.

Note that on data sets where none or all training examples falling into virtual neural nodes, the accuracy of HDT recorded in Table 8 does not really reflect the difference of the neural settings. On *con*, HDT has no virtual neural node. On *bal*, *gla*, *ion*, *iri*, *liv*, *pim*, *sve*, *win*, and *wis*, the difference of the performance of HDTs are in fact caused by adopting FANNC or BP. Adopting FANNC is significantly better on *gla*, *ion*, and *liv*. Adopting BP is significantly better on only *bal*. On *iri*, *pim*, *sve*, *win*, and *wis*, adopting FANNC or BP causes no significant difference to the performance of HDT.

Table 8 Comparisons of collective vs. individual and FANNC vs. BP

data	neural	collective		individual	
set	per (%)	FANNC	BP	FANNC	BP
aut	22.05	0.8323	0.7633	0.8090	0.7438
bal	100	0.8416	0.9120	0.8416	0.9120
con	0.00	0.9612	0.9612	0.9612	0.9612
gla	100	0.6415	0.6114	0.6415	0.6114
hep	3.13	0.8875	0.8325	0.8650	0.8125
ion	100	0.9057	0.8732	0.9057	0.8732
iri	100	0.9398	0.9362	0.9398	0.9362
liv	100	0.8812	0.6924	0.8812	0.6924
pim	100	0.7076	0.7106	0.7076	0.7106
sau	19.2	0.8768	0.8296	0.8333	0.6819
sge	0.50	0.7050	0.7050	0.7038	0.7038
she	34.72	0.7963	0.7185	0.7496	0.6481
sve	100	0.6526	0.6528	0.6526	0.6528
win	100	0.9656	0.9690	0.9656	0.9690
wis	100	0.9560	0.9606	0.9560	0.9606

The difference of the performance of HDTs on the rest data sets does reveal the difference of those four kinds of neural settings. The rank of neural settings according to the accuracy of HDTs are shown in Table 9, where items without significant difference are labeled to a same rank.

Table 9 Rank of neural settings according to the accuracy of HDT

data	collective		individual	
set	FANNC	BP	FANNC	BP
aut	1	3	2	4
hep	1	3	2	4
sau	1	2	2	4
sge	1	1	1	1
she	1	3	2	4
ave.	1	2.4	1.8	3.4

Table 9 shows that collective FANNC always ranks first. This is just the reason that why the default setting of HDT is with a sole collective FANNC network. Table 9 also reveals that collective neural setting is always better than individual neural setting no matter what kind of neural algorithm is adopted by HDT. Moreover, Table 9 reveals that FANNC is more fit for hybrid decision tree than BP because either collective or individual setting is adopted, HDT adopting FANNC is always better than that adopting BP.

3.4 Incremental vs. non-incremental

In this section, we compare the performance of different kinds of incremental learning settings of HDT, including **Incre_noexp($\mathcal{T}_0$, e, $\mathcal{L}$)**, **Incre_fewexp($\mathcal{T}_0$, e, $\mathcal{L}$, $\mathcal{D}$, $\{\mathcal{S}_i\}$)**, and non-incremental procedure where new examples are batch processed along with all the previous training examples. HDT is with a collective FANNC network. The diversity-threshold of HDT is set to 0.2. No constructive induction is performed. For HDT adopting any incremental learning procedure, the training examples are not provided in one pass since HDT employs a two-phase incremental learning strategy described in Section 2.3. Instead, we split the training set into three subsets with similar size. We use one subset to train a primitive tree, use the second subset for pure incremental learning, and use the third subset to attain the accuracy of HDT. The records tabulated in Table 10

Table 10 Comparisons of incremental vs. non-incremental

data set	primitive accuracy	*Incre_noexp*		*Incre_fewexp*		*Non-Incremental*	
		storage (%)	accuracy	storage (%)	accuracy	storage (%)	accuracy
aut	0.6903	0.00	0.7438	27.18	0.8060	100	0.8278
bal	0.6800	0.00	0.8020	0.00	0.8020	100	0.8380
con	0.8040	0.00	0.8723	64.32	0.9374	100	0.9571
gla	0.5460	0.00	0.6176	0.00	0.6176	100	0.6176
hep	0.7500	0.00	0.8125	17.20	0.8750	100	0.8875
ion	0.8140	0.00	0.8814	0.00	0.8814	100	0.8971
iri	0.8670	0.00	0.9333	0.00	0.9333	100	0.9398
liv	0.6238	0.00	0.8520	0.00	0.8520	100	0.8687
pim	0.5439	0.00	0.6799	0.00	0.6799	100	0.6979
sau	0.7536	0.00	0.8551	44.20	0.8333	100	0.8643
sge	0.6200	0.00	0.6650	11.30	0.6950	100	0.7150
she	0.6111	0.00	0.7407	24.00	0.7738	100	0.7923
sve	0.5529	0.00	0.5941	0.00	0.5941	100	0.6526
win	0.8786	0.00	0.9333	0.00	0.9333	100	0.9611
wis	0.8197	0.00	0.9338	0.00	0.9338	100	0.9424

are attained by averaging the results of six experiments where every subset is used in training primitive tree, in pure incremental learning, and in testing the accuracy, which can be viewed as a variation of 3-fold cross validation. In Table 10, *primitive accuracy* denotes the accuracy of the primitive trees, *storage* denotes the percentage of previous training examples that are saved, and *accuracy* denotes the accuracy of the final trees.

Table 10 shows that both incremental procedures can significantly improve the generalization ability of the primitive trees. As anticipated, HDT without incremental learning procedure always achieves the best accuracy because it processes new examples with the help of all the previous training examples. HDT with **Incre_fewexp($\mathcal{T}_0$, *e*, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** is more accurate than or equally accurate to that with **Incre_noexp($\mathcal{T}_0$, *e*, $\mathcal{L}$)** in almost all cases except on data set *sau*. Therefore we believe that **Incre_fewexp($\mathcal{T}_0$, *e*, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** tradeoffs the storage requirement and the improvement of generalization ability.

Here we do not record the incremental time cost because we believe that it is useless to compare the time cost difference between incremental and non-incremental procedures. The reason is just as Utgoff *et al.* (1997) indicated, although the average incremental cost of updating the tree is far lower than that of re-building a new tree from scratch, the sum of the incremental costs may not be lower. As for those two incremental procedures, we believe that **Incre_noexp($\mathcal{T}_0$, *e*, $\mathcal{L}$)** is faster than **Incre_fewexp($\mathcal{T}_0$, *e*, $\mathcal{L}$, $\mathcal{D}$, {$\mathcal{S}_i$})** because the latter does more modification on primitive trees than the former does.

3.5 Constructive vs. non-constructive

In this section, we compare the performance of HDTs with or without constructive induction. Here HDT is with a sole collective FANNC network. The diversity-threshold of HDT is set to zero. In each experiment, a validation set is used to control the constructive induction of HDT, which is generated via bootstrap sampling from training set. Among those 15 data sets, constructive induction is observed on *aut*, *con*, *hep*, *sau*, *sge*, and *she*. Experimental results are tabulated in Table 11,where *time per* denotes the percentage of training time of HDT spent on constructive induction, *attribute* denotes the number of new attributes being constructed, *accuracy* denotes the change of the accuracy of the tree, *height* denotes the change of the height of the tree, *node*, *leaf* and *neural* respectively denotes the change of the number of nodes, leaf nodes, and virtual neural nodes in the tree

Table 11 Comparisons of constructive vs. non-constructive

data set	time per (%)	attribute	accuracy	height	node	leaf	neural
aut	31.8	+ 10.6	+ 2.50	- 2.2	- 12.8	- 6.4	± 0.0
con	27.4	+ 2.6	+ 4.18	- 4.4	- 17.6	- 8.8	± 0.0
hep	12.3	+ 1.6	+ 2.50	- 2.6	- 5.2	- 2.6	± 0.0
sau	46.9	+ 47.8	- 0.14	+ 1.2	+ 10.8	+ 8.4	+ 4.0
sge	54.8	+ 57.0	- 3.90	- 4.0	- 8.4	- 4.2	± 0.0
she	63.6	+ 21.0	+ 3.71	+ 3.6	- 4.0	- 3.2	+ 1.6

due to constructive induction. All the items in Table 11 are attained by 10-fold cross validation.

Table 11 shows that in most cases the constructive induction procedure **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** significantly simplifies the tree by reducing the height of the tree, the number of nodes, and the number of leaf nodes. On *she*, although **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** increases the height of the tree and the number of virtual neural nodes, it does reduce the number of nodes and the number of leaf nodes. Only on *sau*, **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** significantly complicates the tree. Therefore we believe that in most cases **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** can drive the hybrid decision tree become more compact.

On the other hand, the relationship between the accuracy of the final tree and the constructive induction procedure is quite complicated. There are cases (*aut*, *con*, *hep*, and *she*) where **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** improves the accuracy, but there are also cases (*sau* and *sge*) where **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** worsens the accuracy. Note that **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** utilizes a validation set to control the termination of constructive induction. If the validation set accurately captures the distribution of the instance space, then **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** never worsens the performance of the tree because any trends of decline of accuracy will terminate the procedure. However, such a requirement on validation set is seldom satisfied especially in real-world applications. Therefore the improvement of the accuracy of hybrid decision trees caused by **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** is unsteady.

Table 11 also reveals that the time used in constructive induction occupies a large proportion of the entire training time cost of HDT. Note that although **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** increases the training time cost of HDT, it does reduce the test time cost via generating compact trees. Since test speed is more important than training speed in most cases, we believe that **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** worth such expense. However, if the user prefers training time cost to test time cost, then he/she can easily modify **Construct($\mathcal{S}$, $\mathcal{L}$, $\mathcal{D}$)** by using a parameter to control the maximum number of cycles that can be repeated in constructive induction so that the training time cost is limited.

4. Conclusions

In this paper, a novel machine learning approach named HDT is proposed. By virtually embedding a specific feedforward neural network in some leaves of a binary decision tree, HDT simulates human reasoning in a way that symbolic learning is used to do qualitative analysis and neural learning is used to do subsequent quantitative analysis. HDT employs unique techniques of tree growing, neural processing, incremental learning and constructive induction, which enables it to generate accurate and compact hybrid decision trees and deal gracefully with data sets where new data are frequently appended.

Along with the explosive increasing of data and information, incremental learning ability has become more and more important for machine learning approaches. This paper distinguishes three kinds of incremental learning tasks and provides HDT with two kinds of E-IL incremental learning procedures with different storage requirements. However, the other two kinds of incremental learning tasks, i.e. C-IL and A-IL, have not been investigated in this paper. In the near future, we want to do some work on C-IL and A-IL. Moreover, we want to apply HDT to some real-world applications, which may facilitate the refinement of our current work.

Acknowledgements

The National Natural Science Foundation of P.R.China and the Natural Science Foundation of Jiangsu Province, P.R.China, supported this research.

References

Apolloni, B., Zamponi, G. & Zanaboni, A.M. (1998). Learning fuzzy decision trees. *Neural Networks*, 11(5):885-895.

Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365-375.

Atlas, L., Cole, R., Muthusamy, Y., Lippman, A., Connor, J., Park, D., El-Sharkawi, M. & Marks, R.J. (1990). A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE*, 78(10):1614-1619.

Behnke, S. & Karayiannis, N.B. (1996). CNeT: Competitive neural trees for pattern classification. *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, volume 3, pages 1439-1444, 1996.

Bloedorn, E. & Michalski, R.S. (1998). Data-driven constructive induction. *IEEE Intelligent Systems*, 13(2):30-37.

Bloedorn, E., Wnek, J. & Michalski, R.S. (1993). Multistrategy constructive induction. *Proceedings of the 2nd International Workshop on Machine Learning*, San Francisco, CA: Morgan Kaufmann, pages 188-203, 1993.

Blake, C., Keogh, E. & Merz, C. (1998). UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Craven, M.W. & Shavlik, J.W. (1996). Extracting tree-structured representations of trained neural networks. Touretzky, D., Mozer, M. & Hasselmo, M. eds. *Advances in Neural Information Processing Systems 8*, Cambridge, MA: MIT Press, pages 24-30, 1996.

Crawford, S.L. (1989). Extensions to the CART algorithm. *International Journal of Man-Machine Studies*, 31(2):197-217.

Deffuant, G. (1990). Neural units recruitment algorithm for generation of decision trees. *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA, volume 1, pages 637-642, 1990.

Dougherty, J., Kohavi, R. & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proceedings of the 12th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann, pages 194-202, 1995.

Fahlman, S. The cascade-correlation learning architecture. Touretzky, D. ed. *Advances in Neural Information Processing Systems 2*, Cambridge, MA: MIT Press, pages 524-532, 1990.

Fern, A. & Givan, R. (2000). Online ensemble learning: An empirical study. *Proceedings of the 17th International Conference on Machine Learning*, Stanford, CA: Morgan Kaufmann, pages 279-286, 2000.

Frean, M. (1990). *Small nets and short paths: Optimising neural computation*. Ph.d thesis, Centre of Cognitive Science, University of Edinburgh.

Gama, J. (1997). Oblique linear tree. *Lecture Notes in Computer Science 1280*, pages 187-198, 1997.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23(2):121-134.

Guo, H. & Gelfand, S.B. (1992). Classification trees with neural network feature extraction. *IEEE Transactions on Neural Networks*, 3(6):923-933.

Hamzei, G.H.S. & Mulvaney, D.J. (1999). On-line learning of fuzzy decision trees for global path planning. *Engineering Applications of Artificial Intelligence*, 12(1): 93-109.

Hendler, J. (1991). Developing hybrid symbolic/connectionist models. Barnden, J. ed. *Advances in Connectionist and Neural Computation Theory*, Norwood, NJ: Ablex, pages 165-179, 1991.

Jordan, M.I. & Jacobs, R.A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(1):181-214.

Kalles, D. & Morris, T. (1996). Efficient incremental induction of decision trees. *Machine Learning*, 24(3):231-242.

Kohavi, R. & Sahami, M. (1996). Error-based and entropy-based discretization of continuous features. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR: AAAI Press, pages 114-119, 1996.

Krishnan, R., Sivakumar, G. & Bhattacharya, P. Extracting decision trees from trained neural networks. *Pattern Recognition*, 1999, 32(12):1999-2009.

Kubat, M. (1998). Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5):813-821.

Li, T., Fang, L. & Jennings, A. (1992). Structurally adaptive self-organizing neural trees. *Proceedings of the International Joint Conference on Neural Network*s, Baltimore, MD, volume 3, pages 329-334, 1992.

Lovell, B.C. & Bradley, A.P. (1996). The multiscale classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):124-137.

Matheus, C.J. & Rendell, L.A. (1989). Constructive induction on decision trees. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann, pages 645-650, 1989.

McSherry, D. (1999). Strategic induction of decision trees. *Knowledge-Based Systems*, 12(5-6):269-275.

Michalski, R.S. (1983). A theory and methodology of inductive learning. Michalski, R., Carbonell, J. & Mitchell, T. eds. *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, CA: Tioga, pages 83-134.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227-243.

Minsky, M. (1991). Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):35-51.

Murthy, S.K., Kasif, S. & Salzberg S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1-32.

Nilsson, N. (1990). *Learning machines*. San Mateo, CA: Morgan Kaufmann.

Oliver, J.J. (1993). Decision graphs – An extension of decision trees. *Proceedings of the 4th International Conference on Artificial Intelligence and Statistics*, Miami, FL, pages 343-350, 1993.

Pagallo G. & Haussler D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71-100.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 62(1):81-106.

Quinlan, J.R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rumelhart, D., Hinton, G. & Williams, R. (1986). Learning representations by backpropagating errors. *Nature*, 323(9):318-362.

Sanger, T.D. (1991). A tree-structured adaptive network for function approximation in high-dimensional spaces.

IEEE Transactions on Neural Networks, 2(2):285-293.

Sankar, A. & Mammone, R.J. (1991). Optimal pruning of neural tree networks for improved generalization. *Proceedings of the International Joint Conference on Neural Networks*, Seattle, WA, volume 2, pages 219-224, 1991.

Schlimmer, J.C. & Fisher, D. (1986). A case study of incremental concept induction. *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelpha, PA: Morgan Kaufmann, pages 496-501, 1986.

Schmitz, G.P.J., Aldrich, C. & Gouws, F.S. (1999). ANN-DT: An algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6): 1392-1401.

Sethi, I.K. (1990). Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605-1613.

Sethi, I.K. & Yoo, J.H. (1997). Structure-driven induction of decision tree classifiers through neural learning. *Pattern Recognition*, 30(11):1893-1904.

Setiono, R. & Liu, H. (1999). A connectionist approach to generating oblique decision trees. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 29(3):440-444.

Suarez, A. & Lutsko, J.F. (1999). Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297-1311.

Sun, R., Bookman, L.A. & Shekhar, S. (1992). eds. Working Notes of the AAAI Workshop on Integrating Neural and Symbolic Processes, Menlo Park, CA: AAAI, 1992.

Tollenaere, T. (1990). SuperSAB: Fast adaptive backpropagation with good scaling properties. *Neural Networks*, 3(5):561-573.

Utgoff, P.E. (1988). Perceptron trees: A case study in hybrid concept representations. *Proceedings of the 7th National Conference on Artificial Intelligence*, Saint Paul, MN: Morgan Kaufmann, pages 601-606, 1988.

Utgoff, P.E. (1989). Incremental induction of decision trees. *Machine Learning*, 4(2):161-186.

Utgoff, P.E., Berkman, N.C. & Clouse, J.A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5-44.

Utgoff, P.E. & Brodley, C.E. (1990). An incremental method for finding multivariate splits for decision trees. *Proceedings of the 7th International Conference on Machine Learning*, Austin, TX: Morgan Kaufmann, pages 58-65, 1990.

Utgoff, P.E. & Brodley, C.E. (1991). Linear machine decision trees. Technical Report 10, University of Massachusetts at Amherst, 1991.

Wasserman, D. (1993). *Advanced methods in neural computing*. NY: Van Nostrand Reinhold.

Zheng, Z.J. (1998). Constructing conjunctions using systematic search on decision trees. *Knowledge-Based Systems*, 10(7):421-430.

Zhou, Z.-H., Chen, S.-F. & Chen, Z.-Q. (2000). FANNC: A fast adaptive neural network classifier. *Knowledge and Information Systems*, 2(1):115-129.