To appear in the *International Journal of Neural Systems* special issue on Noisy Time Series.

Understanding Time-Series Networks: A Case Study in Rule Extraction

Mark W. Craven* Jude W. Shavlik
Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton St.
Madison, WI 53706
mark.craven@cs.cmu.edu shavlik@cs.wisc.edu

Abstract

A significant limitation of neural networks is that the representations they learn are usually incomprehensible to humans. We have developed an algorithm, called TREPAN, for extracting comprehensible, symbolic representations from trained neural networks. Given a trained network, TREPAN produces a decision tree that approximates the concept represented by the network. In this article, we discuss the application of TREPAN to a neural network trained on a noisy time series task: predicting the Dollar-Mark exchange rate. We present experiments that show that TREPAN is able to extract a decision tree from this network that equals the network in terms of predictive accuracy, yet provides a comprehensible concept representation. Moreover, our experiments indicate that decision trees induced directly from the training data using conventional algorithms do not match the accuracy nor the comprehensibility of the tree extracted by TREPAN.

1 Introduction

Neural networks are perhaps the most widely used and successful method for learning in difficult, time-series problem domains. Although neural networks often provide a high level of predictive accuracy, rarely do they provide insight into the dynamics of the underlying problem domain. This situation is due to the fact that neural networks represent their learned solutions in a manner that does not facilitate human inspection or understanding. In contrast to neural networks, the solutions formed by "symbolic" learning systems, such as those that induce decision trees (Breiman et al., 1984; Quinlan, 1993), are usually much more amenable to human comprehension. We have developed an algorithm, called TREPAN (Craven & Shavlik, 1996; Craven, 1996), which takes a trained neural network as input, and

*Current address is: School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891.

produces as output an approximate, symbolic representation of the concept represented by the trained network. Specifically, Trepan uses queries to the network to induce a decision tree that approximates the network's concept description. Recently we have used Trepan to extract descriptions of a network trained on the problem of predicting the Dollar-Mark exchange rate (Weigend et al., 1996). In this article we review the Trepan algorithm and describe our experiments in applying it to this time-series network.

The comprehensibility of learned solutions is an important consideration in many problem domains. In some cases, it is important that the users of a learning system understand how it arrives at its decisions in order to be confident in its performance. In other cases, there is the possibility that the learning system may discover interesting relationships in the data whose importance was not previously recognized. Because comprehensibility is often an important issue, numerous research groups have investigated methods for extracting symbolic representations from trained neural networks. This work is usually presented under the rubric of "rule extraction."

One of the principal strengths of Trepan in comparison to many related approaches to rule extraction (Hayashi, 1991; McMillan et al., 1992; Craven & Shavlik, 1993; Sethi et al., 1993; Tan, 1994; Towell & Shavlik, 1993; Tchoumatchenko & Ganascia, 1994; Alexander & Mozer, 1995; Setiono & Liu, 1995) is that Trepan does not require a special network architecture or training method. Instead Trepan takes an arbitrary network and simply treats it as a black box, using it to answer queries during the extraction process. In our experiments, we apply Trepan to a network that was trained independently by Weigend et al. (1996) without any expectation that it would later be analyzed by our method.

In applying Trepan to Weigend et al.'s time-series network, we evaluate our extracted tree along the dimensions of complexity, predictive accuracy, and fidelity to the network. Our experiments indicate that Trepan is able to produce a concise decision tree that provides a fairly high level of fidelity to the network. The predictive accuracy of the tree nearly matches the network, and exceeds the accuracy of trees induced directly from the training data by two conventional algorithms. Moreover, the Trepan tree is more concise than those induced by these conventional methods.

2 The Exchange-Rate Prediction Domain

The time-series domain that we consider in our experiments is that of predicting the daily foreign exchange rate between the U.S. Dollar and the German Mark (Weigend et al., 1996). The data for this task consists of daily values from the period of January 15, 1985 through January 27, 1994. The last 216 days were set aside as a test set before the neural network was trained. From the remaining data, every fourth day was held aside to form a validation set (535 days), and the rest of the data (1607 days) was used as a training set.

The neural network we use was trained by Weigend et al. It has 69 input units, a single layer of 15 hyperbolic-tangent units, and three output units. Twelve of the 69 inputs represent information derived from the time series itself (relative strength index, skewness, point and figure chart indicators, etc.), and the other 57 inputs represent fundamental information beyond the series itself (indicators dependent on exchange rates between different countries, interest rates, stock indices, currency futures, etc.). The first output unit predicts a normalized version of the *return*: the logarithm of the ratio of tomorrow's price to today's price is divided by the standard deviation computed over the last 10 trading days. The second output unit predicts the number of days to the next *turning point*: the point at which the daily change in the exchange rate will reverse direction. The third output unit predicts the return between today and the next turning point.

The network was trained using the technique of *clearning* (Weigend et al., 1996). The *clearning* method involves simultaneously *cleaning* the data and *learning* the underlying structure of the data. Specifically, *clearning* uses a cost function that consists of two terms:

$$C = \frac{1}{2}\eta\left(y - y^d\right)^2 + \frac{1}{2}\kappa\left(x - x^d\right)^2$$

The first term, which corresponds to the *learning* aspect of *clearning*, is the squared error between the network's output y and the target value y^d. The second term, which corresponds to the *cleaning* aspect, is the squared deviation between the cleaned input x and the actual input x^d. The idea behind cleaning is that the learned model can be used to adjust training instances so that their values are more likely, given that the model represents the true structure of the problem. The parameters η and κ are the learning rate and the cleaning rate respectively.

In addition to *clearning*, a pruning process is also applied to the network during training. *Clearning* is run until the network starts over-fitting the training data (as indicated by accuracy on the validation set), and then some of the weights from the input-to-hidden layer are pruned. This process is iterated until overpruning is detected. At this point, the last set of pruned weights is restored and the *clearning* step is run a final time. Out of the original 69 features, the *clearned* network used in these experiments retained connections to 15 of the real-valued ones and five of the discrete-valued features.

Although the network is trained to predict three separate quantities (return, number of days to next turning point, and total return between today and the next turning point), we investigate the task of extracting a description of only the *return* output. Moreover, although the *return* output of the network is a continuous variable, we frame the extraction task as one of predicting whether the current day's price is going *up* or *down*. In other words, the *return* output is trained to perform a regression task, but we set up the extraction process as a classification task: describing the qualitative behavior of the *return* output. Addressing the task in this way is necessary because the current version of TREPAN is able to extract classification trees, but not regression trees. We argue, however, that describing the network in this abstract manner results in trees that are easier to comprehend.

3 Extracting Decision Trees

Our approach views the task of extracting a comprehensible concept description from a trained network as an inductive learning problem. In this learning task, the target concept is the function represented by the network, and the concept description produced by our learning algorithm is a decision tree that approximates the network. However, unlike most inductive learning problems, we have available an *oracle* that is able to answer queries during the learning process. Since the target function is simply the concept represented by the

Table 1: **The** TREPAN **algorithm (see text for additional details).**

TREPAN
Input: ORACLE(), training set S, feature set F, *min_sample*, *beam_width*

initialize the root of the tree, R, as a leaf node

`/* get a sample of instances */`
use S to construct a model M_R of the distribution of instances covered by node R
$q := \max(0, \mathit{min\_sample} - \mid S \mid)$
$\mathit{query\_instances}_R$:= a set of q instances generated using model M_R

`/* use the network to label all instances */`
for each example $x \in (S \cup \mathit{query\_instances}_R)$
  class label for x := ORACLE(x)

`/* do a best-first expansion of the tree */`
initialize *Queue* with tuple $\langle R, S, \mathit{query\_instances}_R, \{\} \rangle$
while *Queue* is not empty and global stopping criteria not satisfied

  `/* make node at head of` *Queue* `into an internal node */`
  remove $\langle$ node N, S_N, $\mathit{query\_instances}_N$, $\mathit{constraints}_N \rangle$ from head of *Queue*
  use F, S_N, $\mathit{query\_instances}_N$, and *beam_width* to construct a splitting test T at node N

  `/* make children nodes */`
  for each outcome, t, of test T
    make C, a new child node of N
    $\mathit{constraints}_C := \mathit{constraints}_N \cup \{T = t\}$

    `/* get a sample of instances for the node` C `*/`
    S_C := members of S_N with outcome t on test T
    construct a model M_C of the distribution of instances covered by node C
    $q := \max(0, \mathit{min\_sample} - \mid S_C \mid)$
    $\mathit{query\_instances}_C$:= a set of q instances generated using model M_C and $\mathit{constraints}_C$
    for each example $x \in \mathit{query\_instances}_C$
      class label for x := ORACLE(x)

    `/* make node` C `a leaf for now */`
    use S_C and $\mathit{query\_instances}_C$ to determine class label for C

    `/* determine if node` C `should be expanded */`
    if local stopping criteria not satisfied then
      put $\langle C, S_C, \mathit{query\_instances}_C, \mathit{constraints}_C \rangle$ into *Queue*

Return: tree with root R

---

network, the network itself serves as the oracle. The advantage of learning with queries, as opposed to ordinary training examples, is that they can be used to garner information precisely where it is needed during the learning process.

Our algorithm, as shown in Table 1, is similar to conventional decision-tree algorithms, such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993), which learn directly from a training set. These algorithms build decision trees by recursively partitioning the input space. Each internal node in such a tree represents a splitting criterion that partitions some part of the input space, and each leaf represents a predicted class. Unlike these algorithms, however, TREPAN constructs a decision tree in a best-first manner (the notion of "best" is described shortly). It maintains a queue of leaves which are expanded into subtrees as they are removed from the queue. With each node in the queue, TREPAN stores (i) a subset of the training examples, (ii) another set of instances which we shall refer to as *query instances*, and (iii) a set of constraints. The stored subset of training examples consists simply of those examples that reach the node. The query instances are used, along with the training examples, to select the splitting test if the node is an internal node or to determine the class label if it is a leaf. The constraint set describes the conditions that instances must satisfy in order to reach the node; this information is used when drawing a set of query instances for a newly created node.

Although TREPAN has many similarities to conventional decision-tree algorithms, it is substantially different in a number of respects, which we detail below.

Membership Queries and the Oracle. The generality of TREPAN derives from the fact that its interaction with the network consists solely of *membership queries* (Angluin, 1988). A membership query is a question to an oracle that consists of an instance from the learner's instance space. Given a membership query, the oracle returns the class label for the instance. Recall that, in this context, the target concept we are trying to learn is the function represented by the network. Hence, the network serves as the oracle, and answering a membership query simply involves using the network to classify an instance.

Membership queries are used in two different ways in TREPAN. Initially, they are used to get class labels for the network's training examples. Note that these class labels are not necessarily the "true" class labels, but instead they are determined by the network's classification of the instances. Since we are interested in inducing a description of the trained network, we treat the network's classifications as ground truth. However, TREPAN is not limited to using only the network's training data; it makes membership queries for other instances[1] as well (the query instances). Specifically, TREPAN ensures that it has at least min_sample instances at a node before giving a class label to the node or choosing a splitting test for it. Thus, if m training examples reach a node and $m < min\_sample$, then TREPAN will draw and make membership queries for another $(min\_sample - m)$ instances before making any decisions at the node.

Drawing Query Instances. Usually when TREPAN is selecting query instances, it is subject to a set of constraints that are determined by the location of the node in the tree. Specifically, the constraints state that instances must have outcomes for the tests at nodes higher in the tree that cause the instance to follow the path from the root of the tree to the given node. A key issue then is how to convert a set of constraints into a set of instances that can be used for membership queries. The approach taken by TREPAN is to construct models of the underlying distribution of data, and to use these models in a generative manner to draw instances. Given a model and a set of constraints for each feature, TREPAN generates a value for the feature by sampling from the distribution that is defined by the model conditioned on the constraints.

Although TREPAN could employ sophisticated domain-specific models for this purpose,

[1] We use the term *instance* to refer to any point in the instance space, and the term *example* to refer to an actual observed instance, such as a member of the training set.

by default it uses a fairly simple approach based on modeling the marginal distributions of individual features. TREPAN uses *empirical distributions* to model discrete-valued features, and *kernel density estimates* (Silverman, 1986) to model continuous features. The empirical distribution of a feature is simply the distribution of values that occurs in a given sample of the feature. Thus, the empirical distribution for a discrete-valued feature is represented by a parameter for each possible value of the feature indicating the frequency of that value in the training set. The kernel density estimation method used by TREPAN models the probability density function for real-valued feature x as:

$$f(x) = \frac{1}{m} \sum_{j}^{m} \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{x-\mu_j}{2\sigma}\right)^2} \right]$$

where m is the number of training examples used in the estimate, μ_j is the value of the feature for the jth example, and σ is the width of the Gaussian kernel. TREPAN sets the value of σ to $1/\sqrt{m}$.

This method of modeling the underlying data distribution suffers from one significant limitation: since it estimates only marginal distributions, it does not take into account dependencies among features. TREPAN partially overcomes this limitation by estimating the marginal distributions *locally* as it grows a decision tree. That is, instead of just estimating the marginal distributions once using all of the training data, TREPAN constructs these estimates specific to certain nodes in the tree using only the training examples that reach those nodes. The advantage of constructing these estimates locally is that some of the conditional dependencies are captured, and thus a more accurate model of the true distribution is formed.

Although local models may provide more accurate estimates than a global model by taking into account feature dependencies, in some cases they may instead provide worse estimates because they are based on less data. To handle this trade-off, TREPAN uses a statistical test to decide whether or not to use the local model for a node. In this test, TREPAN compares the distribution of training examples at the node of interest to the distributions at the next highest ancestor in the tree at which a local model was used. If the distributions are significantly different, then TREPAN uses the newly computed distributions as a local model, otherwise it uses the ancestor's model.

To decide if the distributions are significantly different, TREPAN compares the marginal distributions for each unconstrained feature separately using a χ^2 *test* for discrete-valued features and the *Kolmogorov-Smirnov test* (Sachs, 1984) for real-valued features. An unconstrained feature is one whose value is not constrained by tests at internal nodes that are ancestors in the tree. TREPAN rejects the null hypothesis (that the distributions at the two nodes are the same) if the marginal distributions for any feature are significantly different. Since each feature presents an opportunity to spuriously reject the null hypothesis, TREPAN uses the *Bonferroni correction* (Rice, 1995) to adjust the significance level of the overall test downward for the individual tests. TREPAN follows common practice (Loh, personal communication) and uses the relatively high significance value of 0.10 for this test since the Bonferroni correction is conservative in nature (it makes a worst-case assumption about the outcomes of its constituent tests). Note that if a node has very little data on which to base its model, then it is unlikely that the null hypothesis will be rejected. Similarly, if two distributions are statistically indistinguishable, then TREPAN will not reject the null hypothesis, and instead will prefer the distribution that is based on more data.

Tree Expansion. Unlike most decision-tree algorithms, which grow trees in a depth-first manner, TREPAN grows trees using a best-first expansion. The notion of the best node, in this case, is the one at which there is the greatest potential to increase the fidelity of the extracted tree to the network. The function used to evaluate node n is:

$$f(N) = reach(N) \times (1 - fidelity(N))$$

where $reach(N)$ is the estimated fraction of instances that reach N when passed through the tree, and $fidelity(N)$ is the estimated fidelity of the tree to the network for those instances. The motivation for expanding an extracted tree in a best-first manner is that it gives the user a fine degree of control over the size of the tree to be returned: a tree of arbitrary size (in terms of the number of internal nodes) can be selected to describe a given network.

Splitting Tests. Selecting a test for an internal node in a decision tree involves deciding how to partition the part of the instance space covered by the internal node. Whereas C4.5 and CART use single-feature tests for their splitting criteria, TREPAN uses m-of-n expressions for its tests. An m-of-n expression is a Boolean expression that is specified by an integer threshold, m, and a set of n Boolean literals. An m-of-n expression is satisfied when at least m of its n literals are satisfied. For example, suppose we have three Boolean features, x_1, x_2, and x_3; the m-of-n expression *2-of-*$\{x_1, \neg x_2, x_3\}$ is logically equivalent to $(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$. The advantage of using m-of-n tests is that they often result in more concise trees. Murphy and Pazzani (1991) introduced the idea of using m-of-n expressions as splitting criteria in decision trees, and TREPAN's method for constructing such tests is patterned after their ID2-of-3 algorithm.

Like the ID2-of-3 algorithm, TREPAN uses a heuristic search process to construct its m-of-n splitting tests. In the experiments presented here, a beam search is used. The search process begins by first selecting the best binary test at the current node; TREPAN uses the *information gain* criterion (Quinlan, 1993) to evaluate candidate splitting tests. For two-valued features, a binary test separates examples according to their values for the feature. For discrete features with more than two values, we consider binary tests based on each allowable value of the feature (e.g., *color=red?, color=blue?, ...*). For continuous features we consider binary tests on thresholds, as is done by C4.5. The selected binary test serves as a seed for the m-of-n search process. This search uses the information-gain measure as its heuristic evaluation function, and uses the following two operators (Murphy & Pazzani, 1991):

- *m–of–n+1* : Add a new literal to the set, and hold the threshold constant.
  For example, *2-of-*$\{x_1, x_2\} \Longrightarrow$ *2-of-*$\{x_1, x_2, x_3\}$.
- *m+1–of–n+1*: Add a new literal to the set, and increment the threshold.
  For example, *2-of-*$\{x_1, x_2, x_3\} \Longrightarrow$ *3-of-*$\{x_1, x_2, x_3, x_4\}$.

TREPAN constrains m-of-n splitting tests so that the same feature is not used in two or more disjunctive tests which lie on the same path between the root and a leaf of the tree. Without this restriction, TREPAN might have to solve difficult satisfiability problems in order create instances for nodes on such a path. More detail about the algorithm used by TREPAN to ensure that m-of-n constraints are satisfied can be found elsewhere (Craven, 1996).

Using a heuristic search to find m-of-n tests as we do, it can be easy to overfit a given set of instances when there are many possible operator applications. To avoid this pitfall, we place an additional restriction on operator applications. Namely, to decide whether or not a given operator application is admissible, we use a χ^2 test (with a significance level of 0.05) to determine if the proposed change to the m-of-n split results in a significantly different partitioning of the instances than the partition induced by the split before the proposed change. If not, then the particular operator application is not allowed.

Stopping Criteria. To decide when to stop growing an extracted tree, TREPAN uses one *local* stopping criterion, and two *global* criteria. Whereas a local criterion considers the state of only a single node to decide whether or not it should be made a leaf, a global criterion considers the state of the entire tree to decide if the tree-growing process should stop.

The local criterion used by TREPAN is based on the "purity" of the set of examples covered by a node. The given node becomes a leaf in the tree if, with high probability, the node covers only instances of a single class. To make this decision, TREPAN estimates the proportion of examples, $prop_c$, that fall into the most common class at a given node, and then calculates a confidence interval around this estimated proportion (Hogg & Tanis, 1983). TREPAN makes the node a leaf if $Prob(prop_c < 1 - \epsilon) < \alpha$. Here, α is the significance level for the test, and ϵ specifies how tight the confidence interval around the estimated value of $prop_c$ must be. As defaults, TREPAN sets both values to 0.01. A stringent significance level is used for this test because it does not hurt to be conservative in making this decision since TREPAN uses global stopping criteria to more directly control the size of the returned tree.

TREPAN employs two global stopping criteria. The first is simply a limit on the size of the tree that TREPAN can return. This parameter, which is specified in terms of internal nodes, gives the user some control over the comprehensibility of the tree produced by enabling a user to specify the largest tree that would be acceptable. The second global stopping criterion involves using a validation set, in conjunction with the size-limit parameter, to decide on the tree to be returned. Since TREPAN grows trees in a best-first manner, it can be thought of as producing a nested sequence of trees in which each tree in the sequence differs from its predecessor only by the subtree that corresponds to the node expanded at the last step. When given a validation set, TREPAN uses it to measure the fidelity of each tree in this sequence, and then returns the tree that has the highest level of fidelity to the target network.

4 Experimental Methodology

In our experiments, we are interested in evaluating the trees extracted by our algorithm according to three criteria: (i) their predictive accuracy; (ii) their comprehensibility; (iii) and their fidelity to the networks from which they were extracted.

We measure accuracy and fidelity using the examples in the test set. Whereas accuracy is defined as the percentage of test-set examples that are correctly classified, *fidelity* is defined as the percentage of test-set examples on which the classification made by a tree agrees with its neural-network counterpart. Since the comprehensibility of a decision tree is problematic to measure, we measure the syntactic complexity of trees and take this as being representative of their comprehensibility. Specifically, we measure the complexity of each tree in two ways:

(i) the number of internal (i.e., non-leaf) nodes in the tree, and (ii) the number of *feature references* used in the splitting tests of the tree. We count an ordinary, single-feature test as one feature reference. We count an m-of-n test as n feature references, since such a test lists n feature values.

We make six runs of TREPAN varying two parameters: the *beam_width* parameter for the m-of-n search, and the *min_sample* parameter which specifies how many instances TREPAN considers at a node before giving a class label to the node or choosing a splitting test for it. For the *beam_width* parameter, values of 2, 5, and 10 are tried. For the *min_sample* parameter, values of 1,000, 5,000, and 10,000 are tried. For each configuration of parameters, TREPAN grows a tree until it has 31 internal nodes, which is the size of a complete binary tree of depth five. From the sequence of nested trees for a given run (that results from the best-first expansion), TREPAN returns the one that has the highest validation-set fidelity to the network.

As a baseline for evaluating the accuracy and comprehensibility of the trees extracted by TREPAN, we also run two conventional decision-tree algorithms: C4.5 (Quinlan, 1993) and our enhanced version of ID2-of-3 (Murphy & Pazzani, 1991). C4.5 (the successor to ID3) is one of the most widely used inductive learning algorithms, and is perhaps the most popular inductive algorithm for learning "symbolic" hypotheses. ID2-of-3 is a similar algorithm that differs from C4.5 primarily in the nature of the splitting tests it uses. Whereas C4.5 uses single-feature splitting tests, ID2-of-3 uses m-of-n expressions for tests at its internal nodes. The induction problem for both of these algorithms is to learn the classification task of predicting whether the exchange rate will go up or down on the next day. Both algorithms induce their trees directly from the data used to train the neural network.

Our version of ID2-of-3, which we will refer to as ID2-of-3+, includes a number of enhancements on the original algorithm:

1. M-of-n tests can include literals on real-valued features.

2. The heuristic search process that constructs m-of-n tests uses a significance test to decide when to stop the search.

3. The m-of-n search process uses a literal-pruning routine to simplify tests after they are constructed.

4. The heuristic search process uses a beam search (with a beam width of two) instead of a hill-climbing search.

5. After trees are grown, they are pruned using the same pruning process as C4.5.

The first four of these modifications are common to TREPAN also.

We run C4.5 and ID2-of-3+ using two different feature sets. In the first run, both algorithms are given the entire set of 69 features. In the second run, they are given only the 20 features that were incorporated into the hypothesis of the *clearned* network. The latter configuration allows us to test the hypothesis that any performance differences between the trees extracted by TREPAN and the trees induced by C4.5 and ID2-of-3+ are explained by the fact that TREPAN is effectively working with a reduced feature set (the one selected by the *clearned* network).

Table 2: **Parameter settings for algorithms used in the experiments.**

method	parameter settings
Trepan	$beam\_width = 2$, $min\_sample = 5000$
C4.5	pruning level = 15%
C4.5 (reduced feature set)	pruning level = 5%
ID2-of-3+	pruning level = 5%
ID2-of-3+ (reduced feature set)	pruning level = 25%

The pruning method used by C4.5 and ID2-of-3+ is parameterized by a *confidence level* that specifies how liberally the trees should be pruned. We evaluate unpruned trees and trees pruned with confidence levels ranging from 5% (liberal) to 95% (conservative). Accuracy measurements on the validation set are used to determine which tree to return for each algorithm. Like Trepan, ID2-of-3+ uses a χ^2 test to decide when to stop the search process that constructs m-of-n tests. The significance level for this test is set to 0.05, as it is for Trepan.

In addition to C4.5 and ID2-of-3+, we also consider a naive prediction algorithm as a baseline. This algorithm simply predicts that the exchange rate will do the same thing it did yesterday (i.e., if it went up yesterday then predict that it will go up today).

5 Experimental Results

From the six Trepan runs, we select the tree returned by the run that produced the best validation-set fidelity. Similarly, using validation-set accuracy, we select the best C4.5 and ID2-of-3+ trees for the various pruning levels tried. Table 2 lists the values of these parameters for the selected trees.

Table 3 shows test-set accuracy results for the neural network, the Trepan tree extracted from the network, the decision trees induced by the two conventional algorithms, and the naive prediction strategy. It can be seen that the *clearned* network provides the most accurate predictions, and the naive rule (predicting same as previous day) and C4.5 produce the worst predictions. Although the ID2-of-3+ trees are more accurate than the C4.5 trees, they are not as accurate as the tree extracted by Trepan, which makes predictions that are nearly as accurate as the *clearned* network. These results suggest that, by exploiting the concept representation learned by the neural network, Trepan is able to find a better decision tree than either C4.5 and ID2-of-3+. We test the statistical significance of these accuracy differences using the sign test known as the McNemar χ^2 test (Sachs, 1984). Possibly because the test set for this problem is small, none of the differences are significant at $p \leq 0.05$. We note, however, that Trepan is more accurate than the naive algorithm at $p = 0.10$, than C4.5 at $p = 0.09$, and than C4.5 with the reduced feature set at $p = 0.11$. The *clearned* network is more accurate than the naive algorithm at $p = 0.07$, than C4.5 at $p = 0.07$, and than C4.5 with the reduced feature set at $p = 0.09$.

Figure 1 shows the test-set fidelity and accuracy of the intermediate trees produced by Trepan as it add nodes to the tree being extracted. It can be seen in this figure that the

Table 3: **Test-set accuracy.**

method	accuracy (%)
naive rule	52.8
C4.5	52.8
C4.5 (reduced feature set)	54.6
ID2-of-3+	59.3
ID2-of-3+ (reduced feature set)	57.4
TREPAN	60.6
clearned network	61.6

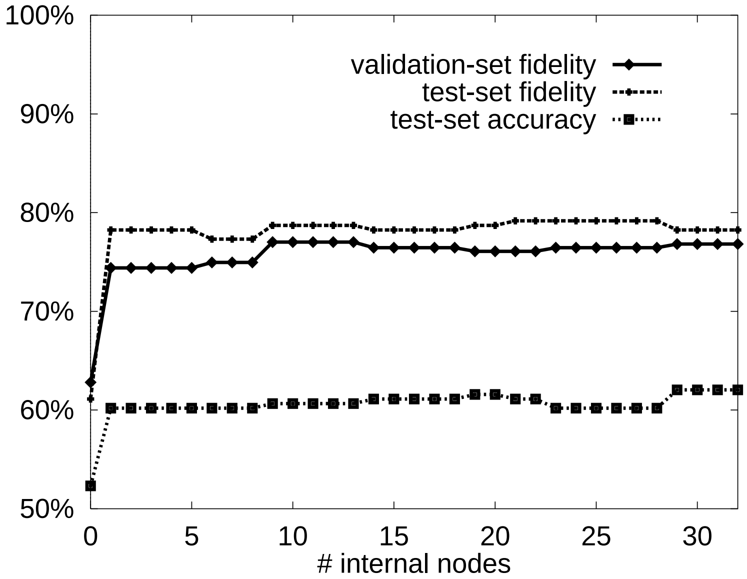

Figure 1: TREPAN **fidelity and accuracy.** The x-axis indicates the number of internal nodes in the extracted tree, and the y-axis indicates test-set fidelity and accuracy.

accuracy of the intermediate trees is fairly constant after the first node is added to the tree. A disappointing result is that the fidelity of the tree is nearly constant as well. The first node in the tree quickly attains fidelity near 80%, but successive nodes added to the tree do not significantly improve this value. This result seems to suggest that TREPAN is failing to find a good decision-tree representation of the network. However, it is interesting to consider the profit-and-loss curves produced by the *clearned* network and the TREPAN tree extracted from it.

Figure 2 shows the profit-and-loss curves that result from trading based on the predictions made by the network and the TREPAN tree extracted from it.[2] Also shown as a reference point is the profit-and-loss curve for trading based on the naive strategy. This figure shows

[2]We calculate these curves using the same method as Weigend et al. (1996). The *full position* is taken each day, meaning that the trader's daily gain/loss is equal to the percentage change in the exchange rate. Additionally, a small transaction cost (0.001) is charged whenever the trader changes its position, meaning that the direction of today's prediction is the opposite of yesterday's prediction.

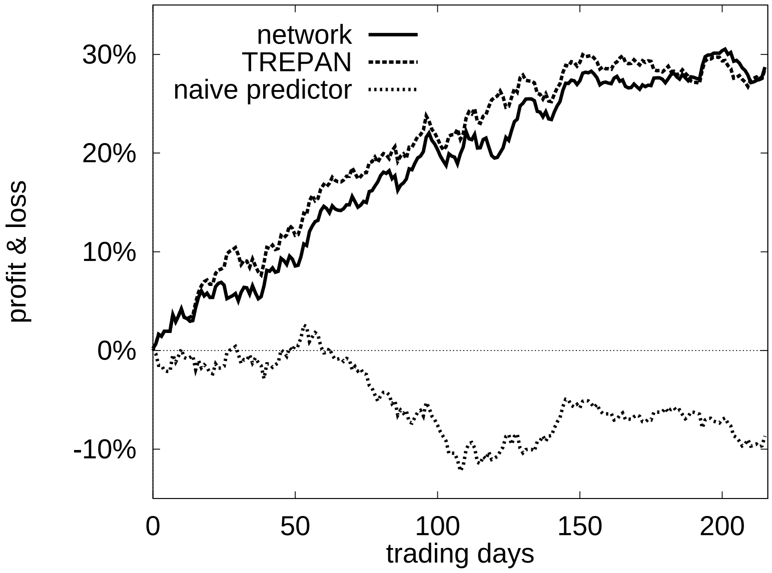

Figure 2: **Profit-and-loss curves.** The x-axis indicates the trading days in the test set, and the y-axis indicates the return on investment that would result from trading based on the predictions made by each method.

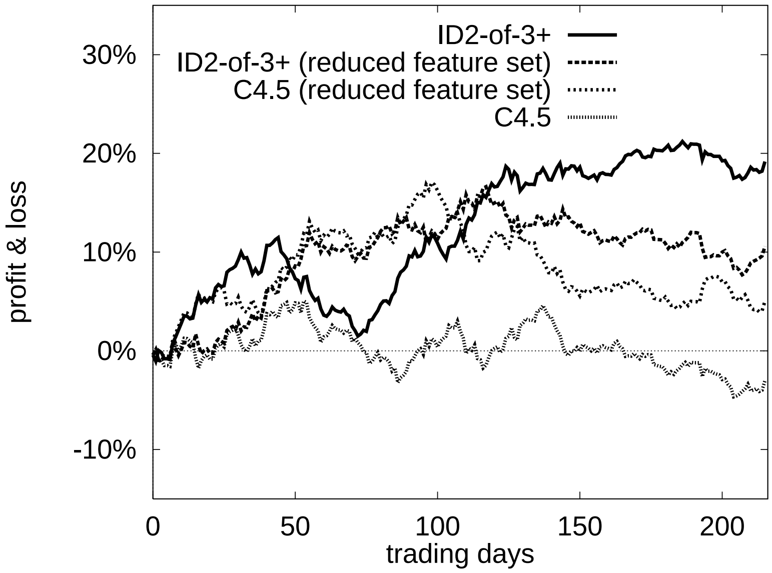

Figure 3: **Profit-and-loss curves.** The axes are the same as in Figure 2. For clarity, not all curves are shown in the same figure.

Table 4: **Tree complexity.** The middle column indicates the number of internal nodes in the tree, and the rightmost column indicates the total number of feature references in the splitting tests used in the tree.

method	# internal nodes	# feature references
C4.5	103	103
C4.5 (selected)	53	53
ID2-of-3+	78	303
ID2-of-3+ (selected)	103	358
TREPAN	5	14

that, although the fidelity of the TREPAN tree (when measured as the percentage of discrete predictions in agreement) is not very high, the tree has captured much of the underlying structure of the solution represented by the network. The profit-and-loss curve for TREPAN closely mirrors that of the neural network. An explanation for the seeming dissonance between this result and the result shown in Figure 1 is that, because the predictions made by the network are clustered around zero, the regression surface represented by the network has many crossings of the plane that divides *up* predictions from *down* predictions. Although TREPAN has a hard time fitting a decision tree to all of these fluctuations in the decision surface, it does a good job of fitting the overall structure of the surface. That is, it more accurately models the regions of the surface that correspond to *way up* or *way down* than it does those regions that represent small values of *up* and *down*.

Figure 3 shows the profit-and-loss curves that result from trading based on the C4.5 and ID2-of-3+ trees. None of these trees performs as well as the network or TREPAN. The best tree in this group is the one induced by ID2-of-3+ using the full feature set. Its total profit is about ten percentage points less than the profit earned by the neural network and the TREPAN tree extracted from it. Moreover, it obviously does not model the gross behavior of the network nearly as closely as the TREPAN tree.

Table 4 shows tree-complexity measurements for the C4.5, ID2-of-3+, and TREPAN trees. For both complexity measures, the TREPAN trees are considerably simpler than the C4.5 and ID2-of-3+ trees, even though the latter trees were pruned. Based on these results, we argue that the tree extracted by TREPAN is more comprehensible than the trees learned by the conventional decision-tree algorithms.

Figure 4 shows the tree extracted by TREPAN. This tree has one m-of-n test, and four ordinary splitting tests. Table 5 provides brief descriptions of the features that are incorporated into the depicted tree. There are sets of features in this domain that involve multiple functions computed on the same underlying indicator; these features are indicated by subscripts in the table and the tree. For example, `Dem_USD_ex[3]` and `Dem_USD_ex[8]` represent different functions of the Mark-Dollar exchange rate.

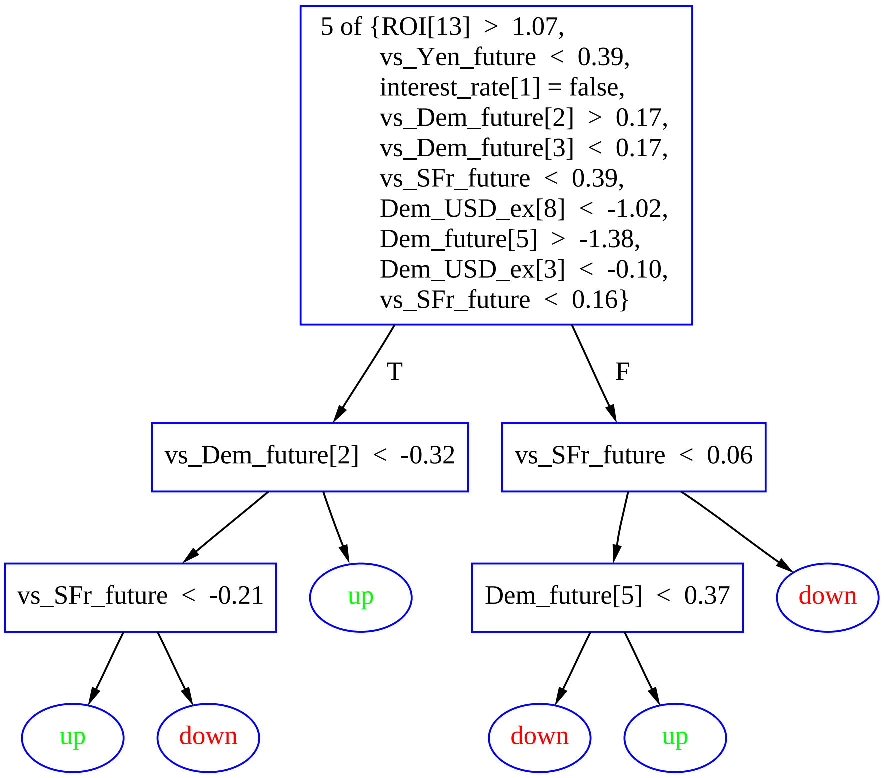

Figure 4: **The** Trepan **tree.** This tree was extracted from the *clearned* network by Trepan. The boxes illustrate internal nodes and the ovals indicate leaves. Each internal node has an associated splitting criterion, and each leaf predicts a class – whether the exchange rate will go up or down.

Table 5: **The features incorporated into the** Trepan **tree.**

abbreviation	description
`Dem_future[5]`	A function of Deutsche Mark futures.
`Dem_USD_ex[3]` `Dem_USD_ex[8]`	Functions of the Mark-Dollar exchange rate.
`interest_rate[1]`	A comparison of the Mark-Dollar exchange rate to the German interest rate.
`ROI[13]`	A comparison of the return on investment (ROI) of investing Marks in the U.S. stock market versus investing Marks in the French stock market.
`vs_Dem_future[2]` `vs_Dem_future[3]`	Measures of the Mark-Dollar exchange rate versus Mark futures.
`vs_SFr_future`	A measure of the Swiss Franc-Dollar exchange rate versus Swiss Franc futures.
`vs_Yen_future`	A measure of the Yen-Dollar exchange rate versus Yen futures.

6 Conclusions

We have illustrated, using the exchange-rate prediction domain, that it is possible to extract a concise, symbolic concept description from a trained neural network. The tree extracted by TREPAN in this domain nearly matches the accuracy of the network itself, and is less complex than trees produced by conventional decision-tree induction algorithms running directly on the training data.

One of the principal strengths of TREPAN is its generality. In contrast to most rule-extraction algorithms, it makes few assumptions about the architecture of a given network, and it does not require a special training method for the network. Moreover, TREPAN is able to handle tasks that involve both discrete-valued and continuous-valued features. The case study presented herein represents one of the rare cases in which a rule-extraction method has been applied to a neural network that was independently trained without any intention of later extracting a symbolic representation from it. In a similar situation, TREPAN has also been applied to a reinforcement-learning network that performs elevator control (Crites & Barto, 1996; Craven, 1996).

In future work, we plan to address the primary limitations of TREPAN, which are that it cannot describe the real-valued predictions of networks trained to perform regression tasks, and it sometimes fails to find trees that are concise yet exhibit a high level of fidelity to their target networks. To address the first limitation, we plan to extend TREPAN so that it can extract *regression trees* in addition to classification trees. A regression tree is a tree-structured model like a decision tree, except that its leaves are characterized by real-valued functions as opposed to class labels. To address the second limitation, we plan to extend the general approach underlying TREPAN to extraction algorithms that use languages other than decision trees to represent their extracted models. We also plan to develop methods for simplifying extracted decision trees by applying truth-preserving transformations to them.

In summary, a significant limitation of neural networks is that their concept representations are usually not amenable to human understanding. We have presented a case study in which we applied our TREPAN algorithm to the task of extracting a comprehensible description from a network trained in a noisy time-series domain. TREPAN produced a concise and accurate tree that provides a good approximation to the concept represented by the network. We believe that our algorithm represents a promising advance towards the goal of general methods for understanding the solutions encoded by trained networks.

Acknowledgements

This research was partially supported by ONR grant N00014-93-1-0998, and by NSF grant IRI-9502990. Andreas Weigend suggested the idea of applying TREPAN to his exchange-rate network, and graciously supplied the network and his expertise.

References

Alexander, J. A. & Mozer, M. C. (1995). Template-based algorithms for connectionist rule extraction. In Tesauro, G., Touretzky, D., & Leen, T., editors, *Advances in Neural Information Processing Systems (volume 7)*. MIT Press, Cambridge, MA.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Craven, M. & Shavlik, J. (1993). Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 73–80), Amherst, MA. Morgan Kaufmann.

Craven, M. W. (1996). *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, WI. Available as CS Technical Report 1326. Available by WWW as ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/craven.thesis.ps.Z.

Craven, M. W. & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. In Touretzky, D., Mozer, M., & Hasselmo, M., editors, *Advances in Neural Information Processing Systems (volume 8)*. MIT Press, Cambridge, MA.

Crites, R. H. & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D., Mozer, M., & Hasselmo, M., editors, *Advances in Neural Information Processing Systems (volume 8)*. MIT Press, Cambridge, MA.

Hayashi, Y. (1991). A neural expert system with automated extraction of fuzzy if-then rules. In Lippmann, R., Moody, J., & Touretzky, D., editors, *Advances in Neural Information Processing Systems (volume 3)*. Morgan Kaufmann, San Mateo, CA.

Hogg, R. V. & Tanis, E. A. (1983). *Probability and Statistical Inference*. MacMillan Publishing, New York, NY.

McMillan, C., Mozer, M. C., & Smolensky, P. (1992). Rule induction through integrated symbolic and subsymbolic processing. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 4)*. Morgan Kaufmann, San Mateo, CA.

Murphy, P. M. & Pazzani, M. J. (1991). ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 183–187), Evanston, IL. Morgan Kaufmann.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Rice, J. A. (1995). *Mathematical Statistics and Data Analysis*. Wadsworth, Belmont, CA.

Sachs, L. (1984). *Applied Statistics: A Handbook of Techniques*. Springer-Verlag, New York, NY, 2nd edition.

Sethi, I. K., Yoo, J. H., & Brickman, C. M. (1993). Extraction of diagnostic rules using neural networks. In *Proceedings of the Sixth IEEE Symposium on Computer-Based Medical Systems*, (pp. 217–222), Ann Arbor, MI. IEEE Press.

Setiono, R. & Liu, H. (1995). Understanding neural networks via rule extraction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, (pp. 480–485), Montreal, Quebec. Morgan Kaufmann.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York, NY.

Tan, A.-H. (1994). Rule learning and extraction with self-organizing neural networks. In *Proceedings of the 1993 Connectionist Models Summer School*, (pp. 192–199), Hillsdale, NJ. Lawrence Erlbaum Associates.

Tchoumatchenko, I. & Ganascia, J.-G. (1994). A Bayesian framework to integrate symbolic and neural learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, (pp. 302–308), New Brunswick, NJ. Morgan Kaufmann.

Towell, G. & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

Weigend, A. S., Zimmermann, H. G., & Neuneier, R. (1996). Clearning. In Refenes, P., Abu-Mostafa, Y., Moody, J., & Weigend, A. S., editors, *Neural Networks in Financial Engineering*. World Scientific, Singapore.