Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 78–93
Neural Networks
www.elsevier.com/locate/neunet

Neural network explanation using inversion

Emad W. Saad[a,*], Donald C. Wunsch II[b]

[a] *Phantom Works, The Boeing Company, Seattle, WA 98124, United States*
[b] *Department of Electrical and Computer Engineering, University of Missouri-Rolla, Rolla, MO 65409, United States*

Received 9 February 2005; received in revised form 7 July 2006; accepted 7 July 2006

Abstract

An important drawback of many artificial neural networks (ANN) is their lack of explanation capability [Andrews, R., Diederich, J., & Tickle, A. B. (1996). A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8, 373–389]. This paper starts with a survey of algorithms which attempt to explain the ANN output. We then present HYPINV,[1] a new explanation algorithm which relies on network inversion; i.e. calculating the ANN input which produces a desired output. HYPINV is a pedagogical algorithm, that extracts rules, in the form of hyperplanes. It is able to generate rules with arbitrarily desired fidelity, maintaining a fidelity–complexity tradeoff. To our knowledge, HYPINV is the only pedagogical rule extraction method, which extracts hyperplane rules from continuous or binary attribute neural networks. Different network inversion techniques, involving gradient descent as well as an evolutionary algorithm, are presented. An information theoretic treatment of rule extraction is presented. HYPINV is applied to example synthetic problems, to a real aerospace problem, and compared with similar algorithms using benchmark problems.

Keywords: Rule extraction; Neural network explanation; Explanation capability of neural networks; Inversion; Hyperplanes; Evolutionary algorithm; Pedagogical

1. Introduction

Artificial neural networks (ANN) have been used during the last few decades in a wide variety of applications. It is often useful to have a symbolic representation of the rule base or the function calculated by the network. When the application is decision support, like in classification or clustering problems, it is often desirable to understand how the ANN determines its decision. Such understanding is often desired in areas such as data mining and financial engineering. For instance, a financial institution rejecting a loan application based on neural network screening might be required by law to provide the grounds for refusal. The explanation capability of ANNs not only serves for justification, but also can be useful in different ways, including debugging and data theory induction. For example, it improves generalization (Andrews, Diederich, & Tickle, 1996) by analyzing rules representations of the input space, and adding data to cover weakly represented areas by using active learning techniques, such as Query-Based Learning (Saad, Choi, Vian, & Wunsch, 2003).

The function learned by the neural network, and used to generate the output, is often referred to as a hypothesis. Many approaches have been suggested in order to generate an explanation of the neural network hypothesis. Most of them try to create a symbolic representation of the information stored in the weights. Rich surveys exist (Andrews et al., 1996; Tickle, Andrews, Golea, & Diederich, 1998; Tickle, Maire, Bologna, Andrews, & Diederich, 2000) which treat rule extraction from ANNs used in decision problems, where the output is discrete.

This paper starts with a survey of algorithms related to the new rule extraction method HYPINV in Section 2, and then proceeds to presenting the HYPINV algorithm in Section 3. Section 4 provides analysis and evaluation criteria of rule extraction methods and shows the experimental results of testing HYPINV. It has been tested on two synthetic problems: XOR and circular distribution; then on a real-world aerospace problem pertaining to ejection seat safety in military airplanes; and experimentally compared against similar algorithms using benchmark problems. Finally, Section 5 has the conclusion and suggestions for future work.

* Corresponding address: The Boeing Company, PO Box 3707, MC 45-85, Seattle, WA 98124-2207, United States. Tel.: +1 206 655 7128.
E-mail address: Emad.w.saad@boeing.com (E.W. Saad).

[1] HYPINV stands for an algorithm which extracts HYPerplanes using INVersion.

0893-6080/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2006.07.005

List of Symbols

A	Rule antecedent
b	Scaling factor
B_i	Bottom value for input variable x_i
C^-	Negative class
C^+	Positive class
CI'_{kj}	Normalized causality index for output neuron k and input neuron i
CI_{kj}	Causality index for output neuron k and input neuron i
C_i	Class i
$\mathbf{D}$	Actual class to which the vector $\mathbf{X}$ belongs
E	Squared error
f	Rule fidelity
$f(.)$	Activation function
$\mathbf{g}$	Gradient vector
$H(\mathbf{X})$	Entropy of $\mathbf{X}$
$I(\mathbf{X}, \mathbf{Y})$	Mutual information of $\mathbf{X}$ and $\mathbf{Y}$
M	Number of classes
M_w	Number of points deleted in the evolutionary algorithm
N	Dimension of input space
$\mathbf{N}$	Neural network output for vector $\mathbf{X}$
$\mathbf{n}$	Vector normal to the decision boundary at $\mathbf{x}_0$
$\hat{\mathbf{n}}$	Unit gradient vector
N_0	Number of neurons in the input layer
N_L	Number of neurons in the output layer
N_{l-1}	Number of neurons in the previous layer
N_p	Number of points generated by the evolutionary algorithm
N_r	Rule counter
N_t	Number of points in a test set
$p(n_j)$	Probability that the neural network output is class j
$p(r_i)$	Probability that the rule base output is class i
$p(r_i, n_j)$	Joint probability that the rule base and neural network outputs are classes i and j respectively
$\mathbf{R}$	Rule base output for vector $\mathbf{X}$
R^+	Area where the rule base classifies the input patterns as belonging to C^+
R^-	Area where the rule base classifies the input patterns as belonging to C^-
R^i	Area where the rule base classifies the input patterns as belonging to class i
S	Input space
t	Iteration number
T_i	Top value for input variable x_i
tr	Target output
$\mathbf{u}$	Vector tangent to the decision boundary
u^l_j	Activation potential of neuron j in layer l
$\mathbf{v}$	Vector between $\mathbf{x}^0$ and $\mathbf{x}_1$
w^l_{ji}	Weight between neuron i in layer l and neuron j in layer $l-1$
w^l_{i0}	Threshold
x	Output of a neuron
$\mathbf{X}$	Random vector
$\mathbf{x}^0$	Network input vector
$\mathbf{x}_1$	Point in the input space
$\mathbf{x}_{\text{int}}$	Point on the decision boundary not necessarily closest to $\mathbf{x}_1$
x^l_j	Output of neuron j in layer l
y	Actual neural network output
z_i	Number of points classified by the rule base as belonging to class i
z_j	Number of points classified by the neural network as belonging to class j
z_{ij}	Number of points classified by the rule base as belonging to class i, and by the neural network as belonging to class j
α	Step size in the iterative sliding along the boundary
η	Learning rate
δ^l_j	Derivative of the squared error with respect to the output of neuron j in layer l
μ	Weighting factor

2. Related work

Andrews et al. (1996) classifies rule extraction algorithms into three approaches based on the nature of the resulting rules, and the approach followed by the algorithm: Boolean using decompositional approaches, Boolean using pedagogical approaches, and Fuzzy rules. In contrast with decompositional approaches, which analyze the activations and weights of the hidden layers of a neural network, the pedagogical approaches treat the ANN as a black box (Andrews et al., 1996; Tickle, Andrews, Golea, & Diederich, 1997) and extract rules by only looking at the input and output activations. Pedagogical approaches aim at extracting symbolic rules which map the input–output relationship as closely as possible to the way the ANN understands the relationship. The number of these rules and their form do not directly correspond to the number of weights or the architecture of the ANN. However, they help explain the ANN behavior by expressing the mapping function performed by the ANN in symbolic rules which are easier to understand.

2.1. Boolean rules extracted by decompositional approaches

Boolean rules can be extracted from networks with binary outputs. Multiclass problems can be coded as multiple binary outputs. Two algorithms which generate rules in the form of conjunction and disjunction of hyperplanes are NeuroLinear of Setiono and Liu (1997a, 1997b) and the algorithm of Kim and Lee (2000). Since HYPINV extracts rules of this form, we will put some focus on both of these algorithms. A more detailed analysis will be presented in Section 4.6. Hyperplane rules have the advantage of showing the trend and correlation between the attributes as opposed to most of the other algorithms above, which generate rules in the form of hypercubes whose axes are parallel to components of the input vector. Using hyperplanes

in problems where the inputs are correlated also leads to the reduction of the number of rules as compared with hypercubes.

2.1.1. NeuroLinear

This algorithm extracts rules from multiplayer–perceptron (MLP) networks used in classification having a single hidden layer. The basic idea is to discretize the hidden unit activation values and to generate rules describing the network output in terms of the discretized values. The relation between the discretized values and the inputs found in the first layer's weights is then represented as a hyperplane rule. The above two sets of rules are finally merged. The algorithm also uses pruning during training, to reduce the number of weights, and hence simplify the rules.

2.1.2. Kim and Lee

This algorithm applies to classification MLP networks with two hidden layers. The algorithm is based on a two-phase process: feature extraction and feature combination. In the first phase, the first layer's weights are collected in form of hyperplanes. In the second phase, these hyperplanes are combined using an induction tree.

Both of the above algorithms can be applied to problems with either discrete or continuous attributes, like HYPINV. They can be applied to multi-class classification problems. A detailed comparison of both algorithms with HYPINV is presented in Section 4.6.

2.2. Boolean rules extracted by pedagogical approaches

These algorithms look only at the input and output of the ANN regardless of the architecture in order to extract the rules, therefore their application is generally more flexible in comparison with the decompositional approaches. The ANN output is binary. HYPINV falls in this category. Andrews et al. (1996) and Tickle et al. (1998) summarize several of these algorithms. The rules extracted by these algorithms can have a variety of formats. For example the Validity Interval Analysis (VIA) by Thrun (1994) extracts rules of either of the following two forms:

IF(input $\in$ hypercube $I = [a_i, b_i]^m$) THEN class is C_j
IF(l of X_1, AND m of $X_2, \ldots$ AND n of X_n) THEN class is C_j

where X_i is a subset of input units.

The Interval Analysis (IA) of Filer, Sethi, and Austin (1996) extracts rules of the form:

IF $\forall 1 \leq i \leq n : x_i \in [a_i^{\min}, a_i^{\max}]$
THEN concept represented by the unit is true.

Thus, VIA and IA extract rules of the form of hypercubes or M-of-N. TREPAN of Craven and Shavlik (1994) extracts rules in the form of a decision tree. RULENEG of Pop, Hayward, and Diederich (1994) extract rules in the form of disjunction of conjunctions. It works with binary inputs. This algorithm iterates through the patterns of the training set and in the worst case the number of rules is equal to the number of training patterns. The BRAINNE system of Sestito and Dillon (1991, 1992) also extracts propositional *if... then... else...* rules. This algorithm deals with continuous input data by first segmenting it into discrete ranges. The usefulness of the explanation provided by the extracted rules depends on their complexity, comprehensibility, and expressive form. In this paper, we define rule complexity as the number of rules in a rule base, and rule comprehensibility as the number of attributes in each rule. The expressive form is rule format (e.g. fuzzy, Boolean, hyperplanes) (see Section 4). The complexity and comprehensibility of the different rule formats depend on their application. The above propositional rules are more useful with binary inputs. They deal with continuous inputs by segmenting them into hypercubes. Such segmentation results in an enormous number of rules. For example, IA extracted 68 rules for 55% accuracy and 947 rules for 88% accuracy with the USER dataset. M-of-N rules are good for binary inputs and they are more compact than propositional rules. For continuous input problems, Hyperplane rules have the lowest complexity as will be seen in Section 4. Thus, the complexity depends mainly on the type of inputs (continuous, discrete, or binary).

3. HYPINV

3.1. HYPINV overview

Here we present the new explanation algorithm HYPINV for neural networks used in classification problems. This paper focuses on two-class problems, but HYPINV can be extended to multiple-class problems as outlined in Section 3.3. HYPINV generates rules in the form of conjunction and disjunction of hyperplanes. The fidelity/complexity tradeoff can be controlled for the rules. HYPINV can be applied to any neural classifier or even any general classifier with graded output function. Though HYPINV is more suitable for differentiable networks, it can be applied to non-differentiable networks by using an evolutionary search algorithm (Reed & Marks, 1995) instead of the gradient descent method in order to invert the network. The neural network inputs can be either binary or continuous. The neural network output should be binary. (A future research topic may be to relax this restriction, in, for example, regression problems, as we briefly discuss in the conclusion.) HYPINV does not depend on the method used to train the neural network. According to the classification scheme of Andrews et al. (1996), HYPINV is of the pedagogical type, since it does not analyze the neural network structure. It actually treats the neural network as a black box, and the extracted rules are not directly related to the neural network architecture or the weight values. HYPINV finds the network decision boundary in the form of a conjunction and disjunction of hyperplanes, i.e. it approximates the ANN decision boundary in a piecewise form. The piecewise form is not generated directly from the data because our aim is to explain the ANN behavior. Rules directly generated from the data may be correct but will not reflect the ANN mapping of the problem. To our knowledge, HYPINV is the only pedagogical rule extraction method, which extracts hyperplane rules from continuous or binary attribute ANN classifiers.

3.2. Algorithm description

A neural network can be expressed as a nonlinear function $y = N(\mathbf{x})$, where the input pattern $\mathbf{x} \in S$, S being the input space of dimension N. In a two-class decision problem, the neural network decision boundary (not the real class boundary) divides the input space into the two half-spaces S^+ and S^-. The input patterns lying in S^+ satisfy the neural network hypothesis h. If $y > \theta$, (θ being a threshold), the neural network says that $\mathbf{x}$ belongs to the positive class C^+, otherwise it says that $\mathbf{x}$ belongs to the negative class C^-.

HYPINV approximates the neural network decision boundary in a piecewise linear form. It proceeds by finding hyperplanes tangent to the neural network decision hypersurface. Each hyperplane divides the input space into a positive and a negative half-space where the data points are classified by this hyperplane as belonging to the positive and negative class, respectively. The network decision boundary is then expressed in the form of a rule base, made of conjunctions and disjunctions of different rule antecedents[2] A_i. The rule antecedents are linear inequalities defining the positive half-spaces delimited on one side by the hyperplanes. Let us denote by R^+ the volume, formed by conjunction and disjunction of those half spaces, within which the rule base classifies the input patterns as belonging to C^+. We then define $R^- = \neg R^+$, and $R^- \vee R^+ = S$. R^+ is an approximation of S^+. Example:
If $A_1 \wedge A_2 \wedge A_3 \vee A_4 \wedge A_5 \vee A_6$ then $\mathbf{x} \in C^+$
where

$$A_1 : 2x_1 - 0.5x_2 + \cdots + 5x_N - 6 > 0$$
$$A_2 : 4x_1 + x_2 + \cdots + 3.6x_N - 3 < 0$$
$$\cdots$$
$$A_6 : 20x_1 - 4.5x_2 + \cdots + 3x_N + 7 > 0.$$

The algorithm can be summarized as follows:

1. Initialize the rule counter $N_r = 0$, and initialize the vector $\mathbf{x}_1$ as either the centroid of S or as a random point in S. The centroid is found by calculating the mean value of the input range (e.g. [0.0, 0.0, 0.0] in the case of a 3 inputs network with bipolar sigmoid activation function). Note that the training data is normalized to fit in this input range.
2. Find the closest hyperplane to $\mathbf{x}_1$ which is tangent to the neural network decision boundary as follows:
   a. Use the neural network inversion technique of Section 3.5 to find the projection of $\mathbf{x}_1$ on the network decision boundary (closest point on the boundary to $\mathbf{x}_1$), and call it $\mathbf{x}_0$.
   b. Calculate the vector $\mathbf{n} = \mathbf{x}_1 - \mathbf{x}_0$, normal to the decision boundary at $\mathbf{x}_0$.
   c. Calculate the equation of the hyperplane tangent to the decision boundary at $\mathbf{x}_0 : \mathbf{n} \cdot (\mathbf{x} - \mathbf{x}_0) = 0$.
3. If $\mathbf{x}_1 \in S^+$, form an antecedent "If $\mathbf{n} \cdot (\mathbf{x} - \mathbf{x}_0) > 0$ then $\mathbf{x} \in C^+$".
   If $\mathbf{x}_1 \in S^-$, form an antecedent "If $\mathbf{n} \cdot (\mathbf{x} - \mathbf{x}_0) < 0$ then $\mathbf{x} \in C^+$".
   Increment the rule counter: $N_r = N_r + 1$.
4. Starting from the second antecedent ($N_r > 1$), add the antecedent as a new rule to the rule base as follows:
   If $\mathbf{x}_1 \in S^+$, then R^+ needs to be enlarged to include $\mathbf{x}_1$. Add the antecedent as a new rule with a disjunction to the rule base.
   If $\mathbf{x}_1 \in S^-$, then R^+ needs to be reduced to exclude $\mathbf{x}_1$. Form a conjunction between the antecedent and those pre-existing antecedents which were satisfied by $\mathbf{x}_1$.
5. Using a testing data set, test the rule base and calculate the fidelity of the rule base f.

$$f = \frac{\#\text{instances where NN and rule base agree}}{\text{size of test set}}. \quad (1)$$

6. If $f \geq F_r$, (F_r is the desired rules fidelity), stop. Otherwise, select a new point $\mathbf{x}_1$, and proceed back to step 2. The point $\mathbf{x}_1$ can be selected using either of these strategies:
   (a) Among the points where the ANN and the rule base disagree, choose the farthest point from the boundary. A good choice is the point with largest absolute value of the ANN output (in the case of bipolar sigmoid activation function) or the largest absolute value of [ANN output – 0.5] (in the case of binary sigmoid).
   (b) Choose a random point. If the ANN and the rule base disagree in classifying it, select it as $\mathbf{x}_1$, otherwise discard it and repeat the random selection.

While method (b) is computationally less expensive, method (a) is more intelligent and generally minimizes the number of rules for the same fidelity level. The idea behind choosing $\mathbf{x}_1$ from among the points where the ANN and the rule base disagree is to generate a new rule which would maximize the gain in the fidelity. Choosing the farthest point from the boundary has two benefits. First, it indicates that there is a large region surrounding the point, where the rule base still disagrees with the neural network; the new rule will thus produce a large increase in the fidelity. Second, it increases the precision in calculating the direction of the vector $\mathbf{n}$. In all our experiments we used method (a). This method results in the extraction of the most important rules first as showed in Fig. 6. It should be noted that occasionally the new rule may not increase the fidelity due to the complexity of the decision boundary. Therefore if the new rule decreases the fidelity, it should not be used and another point should be selected instead. If method (a) is used, the next farthest point from the boundary is selected. Fig. 1 shows an example of the application of the above algorithm.

3.3. Multiple-class problems

Instead of positive and negative classes, we will have class $1, 2, \ldots, M$, where M is the number of classes. The rule base will have the form:

[2] The terms "rule", "rule antecedent" and "antecedent" are used interchangeably throughout this paper.

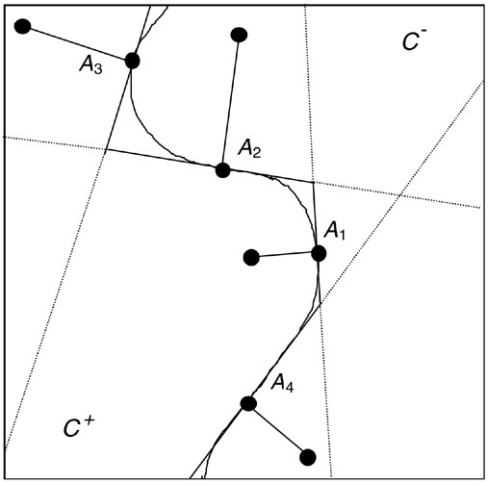

Fig. 1. Example of the approximation of the neural network decision boundary using hyperplanes, in the case of 2D input space. The rule base $A_1 \wedge A_2 \wedge A_4 \vee A_3$. For every rule, the figure shows the generating point, its projection, and the hyperplane normal to the vector $\mathbf{n}$ connecting them. The index of the rule indicates the order in which HYPINV would generate it. Notice that, except for the first rule, the length of $\mathbf{n}$ decreases with the rule index, due to choosing $\mathbf{x}_1$ with method (a).

If R^1 then $\mathbf{x} \in C^1$,
else if R^2 then $\mathbf{x} \in C^2$,
...
else if R^{M-1} then $\mathbf{x} \in C^{M-1}$,
else $\mathbf{x} \in C^M$,

where R^j is made of conjunctions and disjunctions of rule antecedents A_i.

In steps 3 and 4 of the algorithm, if before adding the new rule, $\mathbf{x}_1 \in R^i$ according to the rule base but $\mathbf{x}_1 \in S^j$ according to the neural network, a new rule antecedent $\mathbf{n} \cdot (\mathbf{x} - \mathbf{x}_0) > 0$ is added with conjunction to R^j and with disjunction to R^i.

3.4. Rule simplification

3.4.1. Reducing rule application range

In order to make the rules more comprehensible, it is possible to simplify the rules in the following way. For each input variable x_i, we try to find a top value T_i and a bottom value B_i, such that if $x_{i<}T_i$ the rule base outcome is always either $\mathbf{x} \in C^+$ or $\mathbf{x} \in C^-$, and if $x_{i>}B_i$ the rule base outcome is always either $\mathbf{x} \in C^+$ or $\mathbf{x} \in C^-$. Thus, we obtain areas in the input space where we don't need to calculate the individual rules. First ranges corresponding to the positive class are found then those corresponding to the negative class. This can be done by rewriting the rules such that all parameters except one are on the RHS of the inequality, and then substituting their extreme values in order to find the top and/or bottom values corresponding to the parameter on the LHS. Ranges corresponding to individual rules are then combined using the logical operators relating the rules. Obviously, during this operation some top or bottom values will be invalid since they may lie outside the input domain. This method was tested in Sections 4.4 and 4.5.

3.4.2. Reducing number of rules

The selection method of $\mathbf{x}_1$ described in step 6-(a) of Section 3.2 results in extracting the most important rules first. Thus, the gain in fidelity with each new rule is large at the beginning then tapers off as is seen in Fig. 6. The number of rules can be reduced while only slightly affecting the subspace R^+ and the rule fidelity by removing the most recently generated rules first.

3.4.3. Reducing number of attributes

A simple heuristic for reducing the number of attributes in each rule is to remove attributes with coefficients smaller than a preset threshold. These attributes have minimal contribution to the rule outcome and their removal will only slightly affect the rule accuracy and fidelity. This method was tested in Section 4.6.6.

3.5. Neural network inversion to project a point on the boundary

3.5.1. Inversion of MLP

The inversion of MLP neural networks has been previously shown to be useful in finding a set of input patterns which produce a target output pattern (Linden & Kindermann, 1989). The technique seeks to minimize a least-square cost function by using a gradient descent algorithm.

In the forward path through the network, the output of a neuron unit j in layer l is given as

$$x_j^l = f(u_j^l), \tag{2}$$

and

$$u_i^l = \sum_{j=0}^{N_{l-1}} w_{ij}^l x_j^{l-1}, \tag{3}$$

where N_{l-1} is the number of neurons in the previous layer, w_{i0}^l is the threshold, $f(.)$ is an activation function, and x represents the output of a neural unit. The most commonly used activation functions are the sigmoid

$$f(x) = \frac{1}{1 + \mathrm{e}^{-bx}} \tag{4}$$

and the hyperbolic tangent.

$$f(x) = \tanh(bx) = \frac{\mathrm{e}^{bx} - \mathrm{e}^{-bx}}{\mathrm{e}^{bx} + \mathrm{e}^{-bx}} = \frac{1 - \mathrm{e}^{-2bx}}{1 + \mathrm{e}^{-2bx}}, \tag{5}$$

where b is a scaling factor, which can often be set to one. The latter is antisymmetric, and has the advantage of accelerating the learning process (Haykin, 1998).

In a single output network, the forward mapping from input to output is achieved by finding a set of weights which iteratively minimizes the squared error

$$E = 1/2(\mathrm{tr} - y)^2, \tag{6}$$

where tr is the target output and y is the actual neural network output corresponding to the sample input presented at the

current iteration. This training scheme is know as "pattern update". Pattern update is known to statistically minimize the sum of the squared errors of all training samples. Another training method known as "batch update" averages the weight adjustments needed over all training samples before it updates the weights.

The reverse activity of producing an input vector corresponding to a pre-determined output value in a feed forward network is referred to as network inversion (Linden & Kindermann, 1989). The idea is similar to the backpropagation (BP) algorithm, where the error signals are propagated back to tell the weights the manner in which to change in order to decrease the output error. The inversion algorithm backpropagates the error signals to the input layer to update the activation values of input units so that the output error is minimized. Of course, in the network inversion process, the network weights are frozen.

The iterative update rule for the activation potential u_j^0 in the input layer at the t^{th} iteration becomes

$$\begin{aligned} u_j^0(t+1) &= u_j^0(t) - \eta \frac{\partial E}{\partial u_j^0(t)} \\ &= u_j^0(t) - \eta \frac{\partial E}{\partial x_j^0(t)} \frac{\partial x_j^0(t)}{\partial u_j^0(t)} \\ &= u_j^0(t) - \eta \delta_j^0(t) \frac{\partial x_j^0(t)}{\partial u_j^0(t)}. \end{aligned} \tag{7}$$

The derivative $\delta_j^l(t)$ for the neurons in layer l is obtained in an iterative way using the backpropagation algorithm (Werbos, 1990) as follows:

$$\begin{aligned} \delta_j^l &= \frac{\partial E}{\partial x_j^l} \\ &= \sum_{i=1}^{N_{l+1}} \frac{\partial E}{\partial x_i^{l+1}} \frac{\partial x_i^{l+1}}{\partial x_j^l} \\ &= \sum_{i=1}^{N_{l+1}} \delta_i^{l+1} w_{ij}^{l+1} \frac{\partial x_i^{l+1}}{\partial u_i^{l+1}}. \end{aligned} \tag{8}$$

For one neuron in the output layer, using the cost function in (6),

$$\begin{aligned} \delta_1^L &= -(\text{tr} - x_1^L) \\ &= -(\text{tr} - y). \end{aligned} \tag{9}$$

The final input activation is obtained by $x_j^0 = f(u_j^0)$, which guarantees that the network input will be in the range of the activation function.

Inversion of other network architectures such as Time-Delay Neural Network (TDNN) or Probabilistic Neural Network (PNN) is similarly possible, since these networks have differentiable activation functions. In the case of TDNN, the delays in the network have to be unfolded.

3.5.2. Finding the closest point on the boundary to x_1

The boundary point resulting from the above inversion technique may not always be the closest point to $\mathbf{x}_1$. This depends on its basin of attraction. In order to find the closest point on the neural network decision boundary to a given point $\mathbf{x}_1$, we suggest the following three techniques:

3.5.2.1. Sliding along boundary. At every cycle of the rule extraction process, starting with the point $\mathbf{x}_1$, use the gradient descent to find a point $\mathbf{x}_{\text{int}}$ on the decision boundary, but not necessarily closest to $\mathbf{x}_1$, in this intermediate step. Slide $\mathbf{x}_{\text{int}}$ along the boundary until it reaches the closest position to $\mathbf{x}_1$. The idea is based on calculating the vector tangent to the boundary (Reed & Marks, 1995).

1. Calculate the vector
$$\mathbf{v} = \mathbf{x}_1 - \mathbf{x}^0, \tag{10}$$
where $\mathbf{x}^0$ is the network input vector, or the current position on the boundary.
2. Calculate the gradient vector
$$\mathbf{g} = \frac{\partial y}{\partial \mathbf{x}^0}, \tag{11}$$
and the unit gradient vector
$$\hat{\mathbf{n}} = \frac{\mathbf{g}}{\|\mathbf{g}\|}. \tag{12}$$
3. Calculate $\mathbf{u}$, the projection of $\mathbf{v}$ onto the subspace orthogonal to $\mathbf{g}$, which will be tangent to the boundary:
$$\mathbf{u} = \mathbf{v} - \mathbf{v}^T \hat{\mathbf{n}}. \tag{13}$$
4. Move $\mathbf{x}^0$ with a small step α in the direction of $\mathbf{u}$, such that
$$\mathbf{x}^0(t+1) = \mathbf{x}^0(t) + \alpha \mathbf{u}(t). \tag{14}$$
5. Repeat the above steps until the length of $\mathbf{u}$ is below a small tolerance value.

Remarks:

- The value of α depends on the boundary curvature and can be equal to 1 in the case of a hyperplane boundary. In most cases $\alpha = 0.01$ was appropriate.
- At the end of the sliding process, it is desirable to measure the network output at the reached point, and if found deviating from the boundary, inversion starting with the reached point can be used to pull it back to the boundary.

3.5.2.2. Using an evolutionary algorithm. The gradient descent-based technique described above has the disadvantage that the boundary points obtained may not be evenly distributed because some attractors may have larger basins. An evolutionary algorithm suggested by Reed and Marks (1995) can generate evenly distributed points on the decision boundary. The algorithm can be summarized as follows:

1. Generate N_p points randomly covering the input space.
2. Sort the points by their corresponding network output error.
3. Delete M_w points with worst errors.
4. Generate a replacement for each deleted point as follows. Sort the remaining points in order of their average distance to their nearest m neighbors. Select a parent point from the least crowded points (e.g. by random selection from the top $N_p/5$

points). Generate the new point by adding a small normal random perturbation to the parent.
5. Optionally, repel the most crowded $N_p/5$ points by moving them along the direction tangent to the decision boundary, away from their nearest neighbors. This step needs gradient information to calculate the tangent direction.
6. Repeat steps 2–5 until the maximum error falls under the desired tolerance.

Using the above algorithm, a large number of points can be generated only once at the beginning of the rule extraction procedure. At every step, the closest point to $\mathbf{x}_1$ is chosen to form the new hyperplane. The need to invert the network only once makes this method very efficient for low dimensionality problems, but it is not practical in high dimensionality problems, where a large number of points need to be generated on the decision boundary.

3.5.2.3. Modifying the cost function. The cost function is modified to include the distance between the initial point $\mathbf{x}_1$ and the input pattern. A weighting factor μ multiplies this distance term and serves to control its importance compared to the original network output term

$$E = 1/2(\mathrm{tr} - y)^2 + \mu \sum_{j=1}^{N_0} (x_j^0 - x_{1j})^2. \tag{15}$$

For neurons in the input layer, the derivative of the cost function with respect to their activation becomes:

$$\delta_j^0 = \sum_{i=1}^{N_1} \delta_i^1 w_{ij}^1 \frac{\partial x_i^1}{\partial u_i^1} + \mu(x_j^0 - x_{1j}). \tag{16}$$

Though this technique may work in some simple cases when the point $\mathbf{x}_1$ is very close to the boundary (relative to the sigmoid scaling factor), in general it is not guaranteed to lead to the closest point on the boundary to $\mathbf{x}_1$.

Among the three methods above, the evolutionary algorithm—doing the inversion only once—is less computationally expensive for low dimensionality problems than the sliding along the boundary method. Compared to gradient descent methods, evolutionary algorithms are more likely to find a global optimum; however they are not as good in zooming exactly into the optimum. This inversion method has been tested on the XOR problem in Section 4.3.1. The sliding along the boundary method, being gradient descent-based, has the potential of falling into a local minimum. Therefore it is appropriate for relatively smooth boundary surfaces or when $\mathbf{x}_{\mathrm{int}}$ is relatively close to $\mathbf{x}_0$. In all our experiments we did not encounter any local minimum problem. This method is capable of solving high dimensionality problems at the expense of more computations per iteration. Its scaling with input dimension is $O(N)$, while the evolutionary algorithm scaling is $O(\mathrm{e}^{2N})$ (assuming e^N points generated in order to maintain a given resolution in the evolutionary algorithm). We used this method with data with up to 60 dimensions without any problem (see Sonar data in Section 4.6). This method has been used in all our tests. The modified cost function method is the simplest in implementation and the least computationally expensive but is practically limited by the condition of closeness of $\mathbf{x}_1$ to the boundary. This method was tested on the XOR problem, and was found to be impractical due to the above limitation.

4. Analysis and experimental results

4.1. Evaluation criteria of rule extraction algorithms

The quality of the rule extraction approach depends on several features. The accuracy of the rules measures how close the outputs of the rule base and the neural network are for the same input. Their comprehensibility depends on their expressive form. Different forms include Boolean, fuzzy if-then-else rules, and deterministic finite-state automata extracted from recurrent neural networks. The portability of the algorithm is the possibility of the application of the algorithm to different kinds of neural networks, or neural networks trained with different methods. The translucency (Andrews et al., 1996) of the rules is how much they give insight about the ANN architecture. Decompositional approaches are translucent, in contrast with the pedagogical approaches. Two other criteria are the complexity of the algorithm which is measured by the number of steps needed to extract the rule base, and the complexity of the extracted rules which depends on the number of rules extracted.

4.2. Information theory analysis of rule extraction

Neural networks have been analyzed from an information theory point of view. (e.g. Haykin (1998), Jones, Barnes, Lee, and Mead (1991), Linsker (1988, 1989, 1990a, 1990b) and Yu and Krile (1994)). The principle of maximum information preservation (Linsker, 1988) states that the transformation performed by a neural network between an input vector and an output vector should be chosen such that it maximizes the mutual information between the input and output vectors. In Quinlan (1983, 1986) and Mingers (1989) an information measure is used to build decision trees. The same idea was used by Pratt (1993) and Ramachandran and Pratt (1991) to evaluate the utility of a hyperplane corresponding to a neuron when transferring knowledge between neural networks. Here we suggest a similar approach to evaluate the rules extracted from a neural network. The following analysis can be applied to any rule extraction method and to classification problems with any number of classes.

Let's assume that a random vector $\mathbf{X}$ is presented at the input of the neural network, as well as to the rule base extracted from the network. We will designate the actual class to which the vector belongs as $\mathbf{D}$, the neural network output as $\mathbf{N}$, and the rule base output as $\mathbf{R}$. Note that $\mathbf{D}$, $\mathbf{N}$, and $\mathbf{R}$ are random vectors too. When we extract rules from a network there is an information transfer between the network and the rules, which we seek to maximize. Consider the three mutual information quantities $I(\mathbf{R}, \mathbf{N})$, $I(\mathbf{R}, \mathbf{D})$, and $I(\mathbf{N}, \mathbf{D})$. Ideally, at maximum information transfer, these three quantities are equal to the

entropy of the data, i.e.:

$$I(\mathbf{R},\mathbf{N}) = I(\mathbf{R},\mathbf{D}) = I(\mathbf{N},\mathbf{D}) = H(\mathbf{D}) = H(\mathbf{N}) = H(\mathbf{R}). \tag{17}$$

Practically,

$$\begin{aligned} &I(\mathbf{R},\mathbf{N}) \le \min[H(\mathbf{R}), H(\mathbf{N})],\\ &I(\mathbf{N},\mathbf{D}) \le \min[H(\mathbf{N}), H(\mathbf{D})], \quad \text{and}\\ &I(\mathbf{R},\mathbf{D}) \le \min[H(\mathbf{R}), H(\mathbf{D})]. \end{aligned} \tag{18}$$

Let's consider the mutual information between the rules and the network,

$$\begin{aligned} I(\mathbf{R},\mathbf{N}) &= H(\mathbf{R}) - H(\mathbf{R}\mid\mathbf{N})\\ &= H(\mathbf{R}) + H(\mathbf{N}) - H(\mathbf{R},\mathbf{N})\\ &= -\sum_{i=1}^{M} p(r_i)\log(p(r_i)) - \sum_{j=1}^{M} p(n_j)\log(p(n_j))\\ &\quad + \sum_{j=1}^{M}\sum_{i=1}^{M} p(r_i,n_j)\log(p(r_i,n_j)). \end{aligned} \tag{19}$$

M is the number of classes, $p(r_i)$ and $p(n_j)$ are the probabilities that the rule base and neural network outputs are classes i and j respectively, and $p(r_i,n_j)$ is the corresponding joint probability. Using an appropriate test set, the above mutual information can be approximated by replacing the different probabilities by the corresponding number of data points z in the different classes:

$$\begin{aligned} I(\mathbf{R},\mathbf{N}) &= -\sum_{i=1}^{M}\frac{z_i}{N_t}\log\left(\frac{z_i}{N_t}\right) - \sum_{j=1}^{M}\frac{z_j}{N_t}\log\left(\frac{z_j}{N_t}\right)\\ &\quad + \sum_{j=1}^{M}\sum_{i=1}^{M}\frac{z_{ij}}{N_t}\log\left(\frac{z_{ij}}{N_t}\right)\\ &= 1/N_t\left[\sum_{j=1}^{M}\sum_{i=1}^{M} z_{ij}\log(z_{ij}) - \sum_{i=1}^{M} z_i\log(z_i)\right.\\ &\quad - \sum_{j=1}^{M} z_j\log(z_j)\\ &\quad - \sum_{j=1}^{M}\sum_{i=1}^{M} z_{ij}\log(N_t) + \sum_{i=1}^{M} z_i\log(N_t)\\ &\quad \left. + \sum_{j=1}^{M} z_j\log(N_t)\right]\\ I(\mathbf{R},\mathbf{N}) &= 1/N_t\left[\sum_{j=1}^{M}\sum_{i=1}^{M} z_{ij}\log(z_{ij}) - \sum_{i=1}^{M} z_i\log(z_i)\right.\\ &\quad \left. - \sum_{j=1}^{M} z_j\log(z_j) + N_t\log(N_t)\right], \end{aligned} \tag{20}$$

where N_t is the total number of points in the test set, z_i is the number of points classified by the rule base as belonging to class i, and z_j is the number of points classified by the neural network as belonging to class j. Similarly we can calculate the other two information transfer quantities:

$$\begin{aligned} I(\mathbf{R},\mathbf{D}) &= 1/N_t\left[\sum_{k=1}^{M}\sum_{i=1}^{M} z_{ik}\log(z_{ik}) - \sum_{i=1}^{M} z_i\log(z_i)\right.\\ &\quad \left. - \sum_{k=1}^{M} z_k\log(z_k) + N_t\log(N_t)\right], \end{aligned} \tag{21}$$

and

$$\begin{aligned} I(\mathbf{N},\mathbf{D}) &= 1/N_t\left[\sum_{k=1}^{M}\sum_{j=1}^{M} z_{jk}\log(z_{jk}) - \sum_{j=1}^{M} z_j\log(z_j)\right.\\ &\quad \left. - \sum_{k=1}^{M} z_k\log(z_k) + N_t\log(N_t)\right]. \end{aligned} \tag{22}$$

Thus, using a test set and Eqs. (20)–(22), we can measure the information transfer from the data to the neural network and to the rules, as well as between the neural network and the rules. One advantage of rule extraction is that the rules may be able to generalize better than the underlying neural network. In this case, the rules accuracy would exceed the network accuracy.

Results of calculating the above information measures are shown in Section 4.5.2. The mutual information calculation is compared with the fidelity and accuracy of the rules and neural network. The results indicate that the mutual information between the rule base and the network could not be used as a measure of fidelity. However, the mutual information if calculated for individual rules (hyperplanes) in a fashion similar to that of Pratt (1993) could serve in evaluating rules while pruning the rule base by eliminating those rules of little value.

4.3. Experimental testing of HYPINV on XOR problem

A one hidden layer MLP with 2 hidden neurons has been trained on an XOR problem, resulting in the set of weights: (−3.179332, −4.820779, 4.715858), (−1.665238, 3.051616, −3.410621), and (−2.145716, 4.830636, 4.300482) for the hidden and output neurons respectively, where the first weight of every neuron is the threshold. HYPINV has been used to extract rules. The gradient descent inversion method followed by sliding along the boundary was used to find the point $\mathbf{x}_0$ for every hyperplane. The algorithm resulted in hyperplanes almost coinciding with the actual network decision boundary with fidelity 98.9%. The tiny difference is due to the error tolerance used as stopping criteria during inversion and sliding. Table 1 shows the extracted rules. Fig. 2 shows the extracted hyperplanes and the actual ones. It also shows the path taken during inversion in the case of the second hyperplane A_1. In the case of the first one, the point $\mathbf{x}_0$ lies outside the input space, but still the algorithm produced a valid hyperplane.

Table 1
Rules extracted for the XOR problem and the corresponding points $\mathbf{x}_1$ and $\mathbf{x}_0$

$\mathbf{x}_1$	$\mathbf{x}_0$	Rule antecedent
(0.983, 0758)	(1.16, 0.56)	$A_1 : -0.174x_1 + 0.198x_2 + 0.09 < 0$
(0.0118, 0.997)	(0.197, 0.819)	$A_2 : -0.185x_1 + 0.178x_2 - 0.11 > 0$
Rule Base:	If $A_1 \vee A_2$ then $\mathbf{x} \in C^+$	

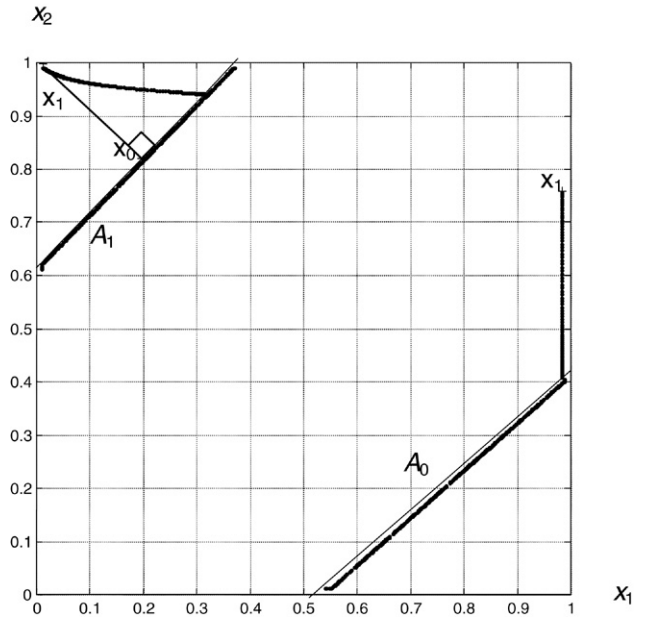

Fig. 2. Hyperplanes extracted for XOR problem (solid line), actual network decision boundary shown by a group of points resulting from network inversion, points $\mathbf{x}_1$ and $\mathbf{x}_0$ for every hyperplane, and path followed during inversion from point $\mathbf{x}_1$ to the boundary in the case of A_1.

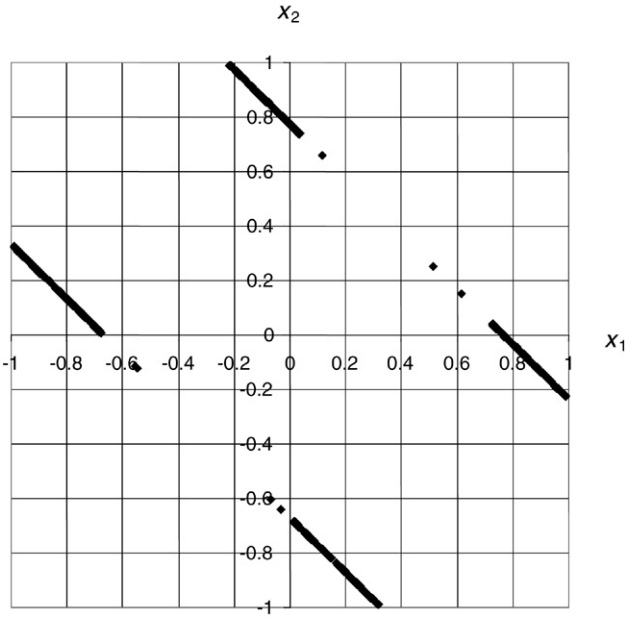

Fig. 3. Gradient descent inversion. Distribution of points depend on basin of attraction.

4.3.1. Comparing gradient descent against evolutionary algorithm inversion

Another MLP with the same architecture as above, except for a hypertangent activation function replacing the sigmoid, was trained again on an XOR problem. In this case the '0' input or output was replaced by the value '−1' for symmetry. Although the gradient descent inversion produced evenly distributed points in the previous case of binary XOR, in the case of the bipolar XOR, the points did not cover the entire decision boundary. The evolutionary algorithm produced evenly distributed points at the expense of more computation time. The results of generating 1000 points by both methods are compared in Figs. 3 and 4.

4.4. Experimental testing of HYPINV on a circular distribution

This is another synthetic problem where we can test HYPINV in the case where the decision boundary is a curve. A one hidden layer MLP with 8 hidden neurons has been trained on the classification problem shown in Fig. 5. Points inside the circle belong to the positive class, those outside belong to the negative class. The training and test sets were of equal size and had 1000 points each. HYPINV was able to approximate the network decision boundary by the conjunction of 7 rules, with 97.7% fidelity, as shown below:

Rule Base:

If $A_1 \wedge A_2 \wedge A_3 \wedge A_4 \wedge A_5 \wedge A_6 \wedge A_7$ then $\mathbf{x} \in C^+$ where

$$A_1 : -0.233x_1 - 0.095x_2 - 0.178 < 0$$
$$A_2 : -0.336x_1 + 0.276x_2 - 0.305 < 0$$
$$A_3 : +0.255x_1 + 0.183x_2 - 0.221 < 0$$

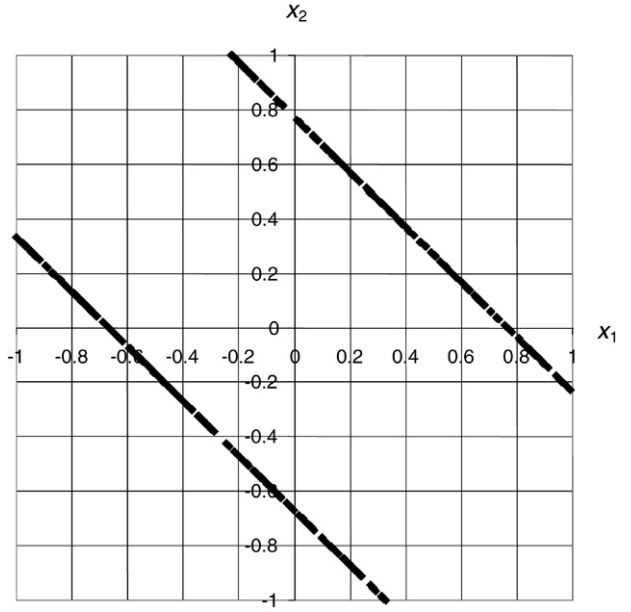

Fig. 4. Evolutionary algorithm inversion. Points are evenly distributed on the decision boundary.

$$A_4 : +0.038x_1 - 0.121x_2 - 0.091 < 0$$
$$A_5 : -0.057x_1 - 0.173x_2 - 0.128 < 0$$
$$A_6 : 0.268x_1 - 0.108x_2 - 0.199 < 0$$
$$A_7 : 0.031x_1 + 0.266x_2 - 0.191 < 0.$$

Fig. 5 shows the decision boundary and the hyperplanes. The ANN decision boundary is shown by 500 inversion points. Fig. 6 shows that HYPINV extracts the more important rules at the beginning, therefore the fidelity improvement starts at a large value and decreases as more rules are extracted.

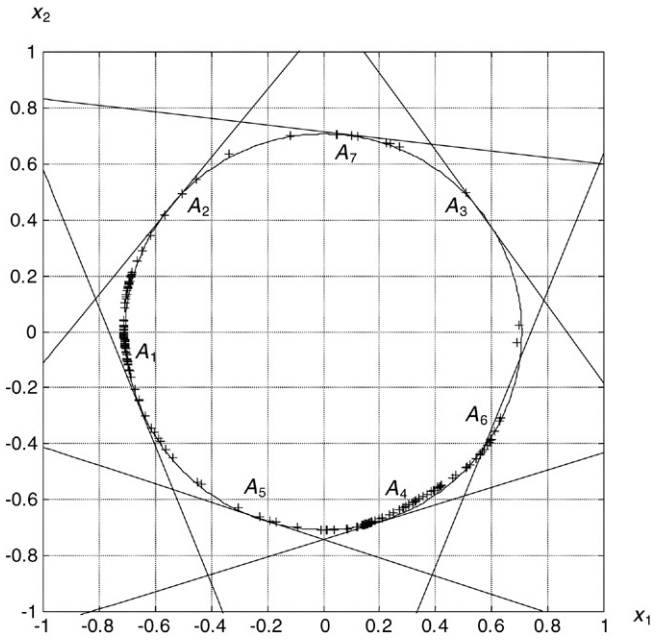

Fig. 5. Approximating network decision boundary using HYPINV. Network decision boundary generated by inversion (shown by '+' signs) coincides with true decision boundary (solid line). Seven linear rules approximate the network decision boundary with fidelity 97.7%.

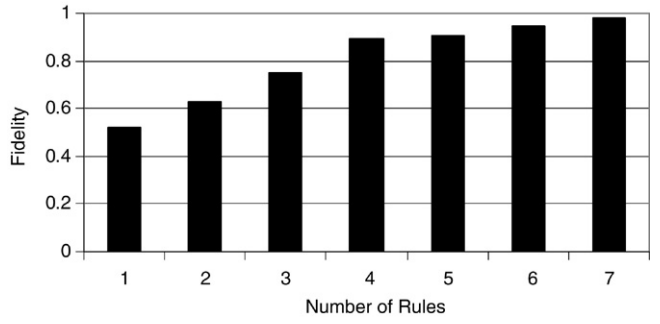

Fig. 6. Relation between the fidelity and number of rules extracted by HYPINV. HYPINV extracts the most important rules first.

Attempting to simplify the extracted rules, would lead to:

If $x2 > 0.83206$ then negative class.
Otherwise calculate rules.

Assuming a uniform distribution of the parameters, this would eliminate 8.4% of the cases where the complete rule base needs to be calculated. While this is only a small saving, the next aerospace problem will show a much larger saving achieved from rule simplification, which shows that the saving is problem dependent.

4.5. Application of network explanation to an aerospace application

4.5.1. Problem statement

We are trying to determine the safety envelope for ejection seats in military airplanes, as illustrated in Fig. 7. It is a two-class decision problem. The safe escape envelope is to be determined as a function of airplane velocity, attitude

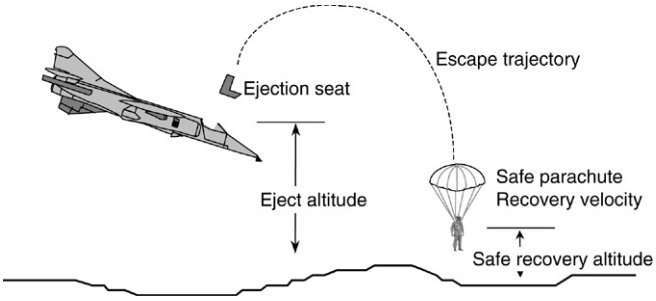

Fig. 7. Safe escape scenario.

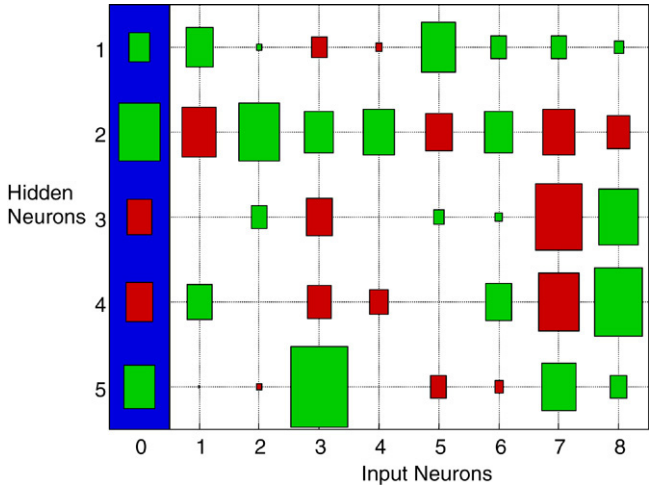

Fig. 8. Hinton diagram for the weights between the input and hidden layers in the network used in the safe escape problem. The leftmost represents the bias.

and other parameters. The system should inform the pilot whether the ejection would be safe beforehand. Ejection safety can be determined via high fidelity ejection seat simulation, using EASY5® software (*EASY5 User's Guide*, 1997). However, such simulations are too slow for in-flight real-time use. Alternately, data can be generated using the EASY5® simulation, and a network can be trained to predict the ejection safety based on the flight parameters. The network has the advantage of much higher speed than the simulator.

The safe escape criteria is that during the escape trajectory, when the total recovery velocity is 50 ft/s, the recovery altitude should be greater than or equal to 50 ft. There are 8 flight parameters, which affect the ejection safety. These parameters are the airplane attitude: pitch and roll angle, the flight path angle (FPA), the angular rates p, q, and r, the ejection altitude, and airplane speed.

Due to the high dimensionality of the input space, sampling with a reasonable resolution as shown in Table 2 would give 302 330 880 points. Generating this amount of data would take 35 years. Query-Based Learning (Hwang, Choi, Oh, & Marks, 1991; Saad, Choi, Vian, & Wunsch, 1999; Saad et al., 2003) has been used in order to efficiently generate and use training data.

4.5.2. Training and explanation results

A multilayer perceptron with 5 hidden neurons was trained using the node decoupled extended Kalman filter (NDEKF) method and query-based learning. The resulting network had

Table 2
Flight parameters determining the escape safety

Parameter	Lower limit	Upper limit	Measurement resolution	Number of measurements
Pitch angle	−90°	+90°	30°	6
Roll angle	−180°	180°	30°	12
Flight path angle	−90°	90°	20°	9
p	−180°/s	180°/s	30°/s	12
q	−180°/s	180°/s	30°/s	12
r	−180°/s	180°/s	30°/s	12
Altitude	0	1500 ft	50 ft	30
Velocity	0	450 keas	50 keas	9

97.46% correct classifications when tested on 2048 uniformly spaced data points. HYPINV has been run several times with different random seeds. The extracted rules in all cases had a fidelity of 80%–85%. Though theoretically the algorithm is able to attain arbitrarily high fidelity, continuing beyond this level usually had problems in convergence during inversion. One remedy under investigation is to use a line-search and adaptive step size during inversion. The input parameters are x_1 = pitch angle, x_2 = roll angle, x_3 = FPA, $x_4 = p$, $x_5 = q$, $x_6 = r$, and x_7 = altitude, x_8 = velocity. HYPINV was able to approximate the network decision boundary by 3 rules, with a 83.35% fidelity, as shown below:

Rule Base:
If $A_1 \vee A_2 \wedge A_3$ then $\mathbf{x} \in C^+$
where

$$A_1 : 6.493pitch - 2.1422roll + 94.551FPA - 2.0456p + 1.1002q - 1.8886r + 318.29alt + 20.431vel - 1.133 \times 10^5 > 0$$

$$A_2 : -3.2914pitch + 11.603roll + 58.099FPA + 9.9544p - 8.5739q + 5.1669r + 293.43alt - 40.824vel + 4572.2 > 0$$

$$A_3 : -0.90337pitch - 10.133roll - 48.159FPA - 5.5235p - 1.2003q - 5.4438r - 135.53alt - 4.6534vel + 27927 < 0$$

Simplifying the rule base would lead to:

If $alt > 162.3995$ then safe.
If $alt < 94.84994$ then safe.
Otherwise calculate rules.

Assuming a uniform distribution of the parameters, this would eliminate 95.5% of the cases where the complete rule base needs to be calculated. This is a much larger saving than that achieved in the circular distribution problem.

Table 3 shows the mutual information between the rules and each of the neural network and the data set, as well as the rules entropy, fidelity and accuracy after the addition of each rule. The fidelity is defined as the ratio of number of points in the test set where both rules and neural network agree to the total number of points. The rules accuracy is defined as the ratio of number of points where the rules give correct classification to the total number of points. The neural network accuracy was 97.42%, its entropy was 0.66779, the data entropy was 0.67223, and the mutual information between the network and the data was 0.50757. We can see that though the rules fidelity and accuracy consistently increase, the mutual information $I(\mathbf{R}, \mathbf{N})$ and $I(\mathbf{R}, \mathbf{D})$ decrease after the addition of the second rule, then increase again after the addition of the third one. This is because these mutual information measures depend on the rules entropy. We also notice that with only the first rule, the rule base fidelity is higher than its accuracy. After the addition of the second rule, its accuracy approaches the fidelity, and after adding the third rule, the accuracy exceeds the fidelity.

Table 3
Fidelity, rule accuracy and information measures for the rule extracted in the aerospace problem

Number of rules	Fidelity (%)	Rule accuracy (%)	$I(\mathbf{R}, \mathbf{N})$	$I(\mathbf{R}, \mathbf{D})$	$H(\mathbf{R})$
1	80.57	81.66	0.0968	0.115	0.765
2	82.6	82.41	0.0044	0.0043	0.0771
3	83.35	84.74	0.0759	0.1005	0.5635

The angle between each pair of hyperplanes was calculated and found to be 13.24°, 166.9°, and 173.3° between the pair of hyperplanes (1,2), (2,3) and (3,1) respectively. This angle relatively close to 0° or 180° indicates that the decision boundary is close to linear.

4.5.3. The Causality Index

The Causality Index (CI) (Baba, Enbutu, & Yoda, 1992; Boger, 1997; Mak & Blanning, 1998) measures the dependence of a neural network output on each of its inputs. It measures the average sensitivity of the network output to each of its inputs. The CI value for a feed forward network with a single hidden layer having N_1 hidden units is computed by

$$CI_{ki} = \sum_{j=1}^{N_1} w_{kj}^2 \cdot w_{ji}^1, \tag{23}$$

where w_{kj}^2 is the weight value between the kth output unit and the jth unit in the hidden layer, and w_{ji}^1 is the weight value between the jth hidden unit and the ith input unit.

In order to compare CI values of different ANN architectures, a normalized causality index is defined as,

$$CI'_{ki} = \frac{\sum_{j=1}^{N_1} w_{kj}^2 \cdot w_{ji}^1}{\sqrt{\frac{1}{N_2 N_0} \sum_{m=1}^{N_2} \sum_{l=1}^{N_0} \left(\sum_{j=1}^{N_1} w_{mj}^2 \cdot w_{jl}^1 \right)^2}}, \tag{24}$$

Table 4
Normalized causality index of several networks trained on the safe escape date

Case no.	No. hidden nodes	No. training epochs	No. training patterns	Sampling methods	Causality index of each of the 8 inputs							
					Pitch	Roll	FPA	p	q	r	Altitude	Velocity
1	**5**	**61**	**2869**	**Uniform**	**−0.091**	**0.1497**	**2.1202**	**0.1717**	**−0.1**	**0.0529**	**1.6256**	**−0.888**
2	5	415	461	QBL	0.0986	0.2389	2.501	−0.074	−0.058	−0.143	1.1481	−0.575
3	5	40	100	Random	−0.087	0.033	2.3734	0.1089	−0.044	0.2883	1.3854	−0.585
4	5	415	1000	Random	0.016	0.2192	2.1199	−0.008	−0.021	0.0307	1.7302	−0.68

The index is consistent for the flight path angle, altitude and velocity inputs.

where N_2 and N_0 are the number of output and input neurons respectively.

The causality index has been calculated for different neural networks trained on the safe escape data. The sample results shown in Table 4 are for different neural network architectures and different training sets. The main remark we can make is that the FPA and the altitude have a large positive causality on the output, and the velocity has a large negative causality on the output. The rest of the inputs have smaller causality indices with inconsistent signs. This can be due to either their small effect on the output and/or their non-monotonic relationship with the output.

The first case in the above table corresponds to the network from which the previous rules were extracted. We notice that the rules coefficients are correlated with the causality index. In the rules, the largest three coefficients are those of the altitude, velocity and FPA, which confirms the importance of these three parameters.

4.5.4. Visualization by Hinton diagram

In attempting to explain a neural network, visualization would be of great aid in increasing human comprehensibility. A Hinton diagram displays the magnitude of weight matrix between neurons in two layers of a neural network. The weight matrix is displayed as a square grid. The area of a square is proportional to the weight magnitude, and its color indicates its sign (light = positive, dark = negative), as shown in Fig. 8. When plotting the weights for the network used above, we could notice that the weights corresponding to the FPA, altitude and velocity inputs are much larger than the rest, again confirming the results of the causality index.

4.6. Experimental comparison with other methods

We have experimentally compared the performance of HYPINV against two similar algorithms: NeuroLinear of Setiono and Liu (1997a) and the algorithm of Kim and Lee (2000) which were briefly described in Section 2.1. In this comparison we used data sets publicly available at the machine learning data repository at the University of California, Irvine (UCI) (Blake & Merz, 1998), for which test results has been previously reported by Kim and Lee (2000) and Setiono and Liu (1997a). We used these reported results without simulating the two algorithms. From among the data sets used in these two papers, we tested our algorithms on all the two-class sets. These data sets are presented in Table 5.

Table 5
UCI datasets used in the experiments

#	Dataset	Size	Attributes		
			Total	Discrete	Continuous
1	Wisconsin breast cancer	683	9	0	9
2	Australian credit approval	653	14	8	6
3	Sonar target	208	60	0	60
4	Ionosphere	351	34	0	34
5	Pima Indians diabetes	768	8	0	8
6	BUPA liver-disorders	345	6	0	6

Table 6
Neural network accuracy

Dataset	Saad et al.	Setiono et al.	Kim et al.
Wisconsin breast cancer	94.74%	94.57%	95.1%
Australian credit approval	83.44%	83.84%	–
Sonar target	87.5%	88.63%	–
Ionosphere	92.05%	–	91.4%
Pima Indians diabetes	75.52%	–	75.3%
BUPA liver-disorders	67.44%	–	67.4%

The goal is to have a fair comparison of the rule extraction algorithms independently of the goodness of the underlying neural networks. Therefore, in training our neural networks, we aimed at achieving error rates as close as possible to the ones reported for the above two algorithms. We used cross validation for minimizing the generalization error. We then fine tuned the error by either early stopping or changing the random seed of the weight initialization. The resulting neural network accuracy on the test set was within 1% of the other two algorithms on all data sets except for Sonar where the difference was 1.13%.

For all data sets we used an MLP with one hidden layer of 4 neurons guided in our choice by the architecture used by Setiono et al. All neurons had a sigmoid activation function and all weights were randomly initialized in the range $[-1, 1]$. Each data set was split in the proportion of about 50% training, 25% validation and 25% test data, except for the sonar set which had 176 training data and the rest equally split between validation and test data. Table 6 compares the neural network accuracy of the different algorithms. The accuracy was measured as the ratio of the number of correctly classified patterns in the test set to the size of the test set.

After training the neural networks for all data sets, we used HYPINV to generate rules for them. HYPINV was run 10 times for each data set using a different random seed every time. The average and standard deviation of the results of the 10 runs was

Table 7
Rules accuracy (%)—average and standard deviation

Dataset	Saad et al.	Setiono et al.	Kim et al.
Wisconsin breast cancer	95.26 (0.8)	95.73 (3.75)	95.6 (0.3)
Australian credit approval	82.52 (1.01)	83.64 (5.74)	–
Sonar target	76.25 (4.93)	85.39 (12.77)	–
Ionosphere	90.11 (2.28)	–	91.2 (0.9)
Pima Indians diabetes	78.59 (0.46)	–	73.4 (3.9)
BUPA liver-disorders	65.93 (3.72)	–	67.6 (2.9)

Table 8
Number of rules—average and standard deviation

Dataset	Saad et al.	Setiono et al.	Kim et al.
Wisconsin breast cancer	1.1 (0.32)	2.89 (2.52)	7.8 (0.8)
Australian credit approval	2.0 (0.67)	6.6 (4.40)	–
Sonar target	1.0 (0.00)	7.03 (3.73)	–
Ionosphere	1.3 (0.48)	–	6.3 (0.5)
Pima Indians diabetes	1.0 (0.00)	–	23.6 (2.9)
BUPA liver-disorders	2.9 (1.29)	–	15.3 (1.5)

Table 9
Rules fidelity (%)—average and standard deviation

Dataset	Saad et al.
Wisconsin breast cancer	99.12 (0.31)
Australian credit approval	97.48 (1.67)
Sonar target	88.75 (4.93)
Ionosphere	96.02 (3.09)
Pima Indians diabetes	98.70 (0.86)
BUPA liver-disorders	91.05 (4.52)

calculated for each set. The three algorithms are compared in Table 7 in terms of the accuracy of the extracted rules and in Table 8, in terms of the number of extracted rules. In the case of Kim et al., they reported results by two variations of their algorithm. Here we compared against their best result for each data set. We can see that the accuracy of HYPINV is in the same range as the other two algorithms. However, HYPINV gives fewer rules. The fidelity of our rules is shown in Table 9 for the test sets of all cases. Fidelity was not reported for the compared algorithms. Let's discuss the comparison:

Both of the compared algorithms can be applied to problems with either discrete or continuous attributes, like HYPINV. They can be applied to multi-class classification problems. HYPINV is extendable to multi-class problems. By comparison with the above two algorithms, HYPINV is more flexible and has wider applicability. The above two algorithms can only be applied to MLPs with a specific architecture. HYPINV is pedagogical, and hence it is applicable to a variety of architectures. It can generally be applied to any neural classifier.

4.6.1. Accuracy

The accuracy of HYPINV is comparable with the compared ones for all cases. In some cases the difference is larger than others. In two cases (Breast cancer and Pima), the rule accuracy is higher than the ANN accuracy. In fact, if the ANN is overfit, the rules may generalize better by simplifying the decision surface.

4.6.2. Fidelity versus complexity of the extracted rules

One main advantage of HYPINV is the ability of generating rules with arbitrary fidelity while maintaining a fidelity–complexity tradeoff. This is due to its ability of extracting the most important rules first. The more we add rules, we increase the fidelity, but we also increase the complexity of the rules. The other compared algorithms extract a set of rules with fixed fidelity. Their complexity depends on the number of weights in the network. The testing of HYPINV showed a great reduction in the complexity of the rules compared with the other two algorithms. The reduction in the number of rules ranges from 62% for the breast cancer data to 96% for the Pima data. Although we could not compare the fidelity, our accuracy was comparable to the other algorithms. Other algorithms which addressed a tradeoff between interpretability and accuracy include that of Plate (1999).

4.6.3. Complexity of the algorithm

HYPINV consists of these main operations for each rule: network inversion, one forward propagation to calculate the class of $\mathbf{x}_1$, and multiple forward propagations with the test set to calculate the fidelity. Network inversion can use either backpropagation or an evolutionary algorithm, which involves forward propagation to calculate the network output error. Thus, the algorithm complexity and execution time depend on the network size. The other two algorithms analyze the network weights and their complexity depends on the network size as well.

4.6.4. Portability

HYPINV is pedagogical, and therefore is less dependent on the network structure. It can be applied to any neural classifier without limit to certain architecture. The other two algorithms are decompositional and analyze the weight structure of the network. NeuroLinear is restricted to MLPs with one hidden layer, while Kim's algorithm is restricted to MLPs with two hidden layers using steep sigmoid activation function.

HYPINV is also independent of the training method. Kim's algorithm requires a special training method involving increasing the sigmoid slope gradually during training. When using NeuroLinear, the network can be trained by any method, but the weights need to be pruned before the rules are extracted.

4.6.5. Translucency

HYPINV is pedagogical, it does not give insight about the ANN architecture. We cannot tell for example how many layers are in the network just from the rules. The other compared algorithms are decompositional and therefore more translucent. It is our assertion that generating explanations is more important than translucency. That is, one is usually not interested in inferring facts about the network architecture, since it is often available for direct examination anyway.

4.6.6. Comprehensibility

The three algorithms extract rules of the same form (hyperplanes). Their relative comprehensibility will be measured by the number of attributes in each rule. HYPINV produces rules with all the original attributes in each of them. The rules of

Table 10
Comparison of algorithms extracting rules in the form of hyperplanes

Property	Saad et al.	Setiono et al.	Kim et al.
Input	Continuous/Discrete	Continuous/Discrete	Continuous/Discrete
Output	2-class, extendable to multi class	Multi class	Multi class
Algorithm class	Pedagogical	Decompositional	Decompositional
Network type	Any network	MLP	MLP with steep sigmoid
Number of hidden layers	Any number	One	Two

Kim et al. use additional attributes derived from the original ones, the number of which is of the same order of magnitude as the original ones. Setiono et al. do not report the number of attributes in their rules for all data sets. However, in their detailed analysis of two of the sets (Setiono & Liu, 1997b), their rules generally have fewer attributes due to neural network pruning. We can see that our rules have comprehensibility similar to that of Kim et al. but lower than that of Setiono et al. We can improve the comprehensibility of our rules by pruning the extracted rules. One simple way to do this is by removing those attributes with coefficients smaller than a preset threshold. This method was tested on the sonar rules which had the largest number of attributes among the tested sets. By using a threshold of 0.018, the number of attributes was reduced from 60 in each rule to 39 in the first rule and 33 in the second rule without sacrificing any accuracy.

Table 10 compares the main characteristics of the three algorithms.

5. Conclusion

We presented HYPINV, a new explanation algorithm based on network inversion. It can approximate the network decision surface in terms of hyperplanes for continuous or binary attribute neural networks. The majority of the algorithms extracting rules from neural networks with continuous attributes transform them to a binary form (Bologna, 2000), which corresponds to approximating the decision region using hypercubes. Only a few algorithms—of the decompositional type—generate hyperplane rules from neural networks with continuous attributes. The general principle of HYPINV is to extract hyperplane rules in a pedagogical manner. To our knowledge, HYPINV is the only one of the pedagogical type, which extracts hyperplane rules from continuous or binary attribute neural networks. Being pedagogical, HYPINV is portable to any neural classifier without restriction on the architecture or the training method, unlike decompositional approaches. Extraction of rules in the form of hyperplanes results in fewer rules (less complex) compared with rules in the form of hypercubes extracted by decision trees or other algorithms designed for binary attributes which discretize continuous attributes. Hyperplanes also show the relation between the attributes. In addition, HYPINV allows flexibility in choosing the rules fidelity and complexity while maintaining a fidelity–complexity tradeoff. This is allowed by the fact that the algorithm extracts the most important rules first. The increased value of HYPINV is obtained while keeping it cost-effective. Experiments showed that its run time is typically a few seconds.

In HYPINV, inversion, followed by sliding along the boundary, has been used to project a point in the input space on the network decision boundary. An information theoretic analysis of rule extraction has been given and measures of the rules fidelity and accuracy have been derived. HYPINV has been applied to synthetic problems for validation as well to a real aerospace problem. It has also been compared against similar algorithms using benchmark problems. The extracted rules were further simplified for better human understandability. Visualization has also been added to aid the explanation.

Future work can investigate the extension of HYPINV to multiple class problems as well, such as to regression problems. One approach which can be taken regarding regression problems is to discretize the output. This approach may be more suitable for single-output networks, but may face the curse of dimensionality in multiple-output networks. Note that there is a large class of single-output, nonlinear, multivariate regression problems which would benefit greatly from an explanation capability.

Acknowledgment

The authors would like to thank Dr. John Vian with Phantom Works, The Boeing Co., for consultation and guidance relating to the aerospace application.

References

Andrews, R., Diederich, J., & Tickle, A. B. (1996). A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, *8*, 373–389.

Baba, K., Enbutu, I., & Yoda, M. (1992). Explicit representation of knowledge acquired from plant historical data using neural network. In *Proceedings of the international joint conference on neural networks* (pp. 579–583).

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

Boger, Z. (1997). Knowledge extraction from artificial neural networks models. In *Proceedings of the IEEE international conference on systems, Man & Cybernetics* (pp. 3030–3035).

Bologna, G. (2000). A study on rule extraction from neural network applied to medical databases. In *Proceedings of the 4th european conference on principles and practice of knowledge discovery (PKDD2000)*.

Craven, M. W., & Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In *Proceedings of the eleventh international conference on machine learning*.

EASY5 User's Guide. (1997). Seattle, WA: The Boeing Company.

Filer, R., Sethi, I., & Austin, J. (1996). A comparison between two rule extraction methods for continuous input data. In *Proceedings of the neural information processing systems (NIPS'97), rule extraction from trained*

artificial neural networks workshop (pp. 38–45). Queensland University of Technology.

Haykin, S. (1998). *Neural networks: A comprehensive foundation*. Englewood Cliffs, NJ: Prentice-Hall.

Hwang, J., Choi, J., Oh, S., & Marks, R., II (1991). Query-based learning applied to partially trained multilayer perceptrons. *IEEE Transactions on Neural Networks*, *2*, 131–136.

Jones, R., Barnes, C., Lee, Y., & Mead, W. (1991). Information theoretic derivation of network architecture and learning algorithms. In *Proceedings of the international joint conference on neural networks* (pp. 473–478).

Kim, D., & Lee, J. (2000). Handling continuous-valued attributes in decision tree with neural network modeling. In R. López de Mántaras, & E. Plaza (Eds.), *Lecture notes in computer science*: *Vol. 1810*. *Proceedings of the 11th european conference on machine learning* (pp. 211–219). Berlin, Heidelberg: Springer-Verlag.

Linden, A., & Kindermann, J. (1989). Inversion of multilayer nets. In *Proceedings of the international joint conference on neural networks* (pp. 425–430).

Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, *21*, 105–117.

Linsker, R. (1989). How to generate ordered maps by maximizing the mutual information between input and output signals. *Neural Computation*, *1*, 402–411.

Linsker, R. (1990a). Perceptual neural organization: Some approaches based on network models and information theory. *Annual Review of Neuroscience*, *13*, 257–281.

Linsker, R. (1990b). Self-organization in a perceptual system: How network models and information theory shed light on neural organization. In S. J. Hanson, & C. R. Olson (Eds.), *Connectionist modeling and brain function: The developing interface* (pp. 351–392). Cambridge, MA: MIT Press.

Mak, B., & Blanning, R. (1998). An empirical measure of element contribution in neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, *28*, 561–564.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, *3*, 319–342.

Plate, T. A. (1999). Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using gaussian process models. *Behaviormetrika*, *26*, 29–50.

Pop, E., Hayward, R., & Diederich, J. (1994). RULENEG: Extracting rules from a trained ANN by stepwise negation. QUT NRC.

Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. *Advances in Neural Information Processing Systems*, *5*, 204–211.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 463–482). Palo Alto, CA: Tioga Pub.

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

Ramachandran, S., & Pratt, L. Y. (1991). Information measure based skeletonisation. *Advances in Neural Information Processing*, *4*, 1080–1087.

Reed, R. D., & Marks, R. (1995). An evolutionary algorithm for function inversion and boundary marking. In *Proceedings of the IEEE international conference on evolutionary computation* (pp. 794–797).

Saad, E. W., Choi, J. J., Vian, J. L., & Wunsch, D. C. II. (1999). Efficient training techniques for classification with vast input space. In *Proceedings of the international joint conference on neural networks*, no. 288.

Saad, E. W., Choi, J. J., Vian, J. L., & Wunsch, D. C., II (2003). Query-based learning for aerospace applications. *IEEE Transactions on Neural Networks*, *14*, 1437–1448.

Sestito, S., & Dillon, T. (1991). The use of sub-symbolic methods for the automation of knowledge acquisition for expert systems. In *Proceedings of the 11th international conference on expert systems and their applications* (pp. 317–328).

Sestito, S., & Dillon, T. (1992). Automated knowledge acquisition of rules with continuously valued attributes. In *Proceedings of the 12th international conference on expert systems and their applications* (pp. 645–656).

Setiono, R., & Liu, H. (1997a). NeuroLinear: A system for extracting oblique decision rules from neural networks. In M. van Someren, & G. Widmer (Eds.), *Lecture notes in artificial intelligence*: *Vol. 1224*. *Proceedings of the 9th european conference on machine learning* (pp. 221–233). Prague, Czech Republic: Springer.

Setiono, R., & Liu, H. (1997b). NeuroLinear: From neural networks to oblique decision rules. *Neurocomputing*, *17*, 1–24.

Thrun, S. (1994). Extracting provably correct rules from artificial neural networks. Technical report IAI-TR-93-5. Institut für Informatik III Universität Bonn.

Tickle, A. B., Andrews, R., Golea, M., & Diederich, J. (1997). Rule extraction from trained artificial neural networks. In A. Browne (Ed.), *Neural networks analysis, architectures and applications* (pp. 61–99). UK: Institute of Physics Pub.

Tickle, A. B., Andrews, R., Golea, M., & Diederich, J. (1998). The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, *9*, 1057–1068.

Tickle, A. B., Maire, F., Bologna, G., Andrews, R., & Diederich, J. (2000). Lessons from past, current issues and future research directions in extracting the knowledge embedded in artificial neural networks. In S. Wermter, & R. Sun (Eds.), *Hybrid neural systems* (pp. 227–240). Heidelberg: Springer.

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *7*, 1550–1560.

Yu, Q., & Krile, T. F. (1994). Information theory analysis of hebbian-type associative memories (HAMs). In *Proceedings of the world congress on neural networks* (pp. 64–69).

Further reading

Andrews, R., & Geva, S. (1995). RULEX & CEBP networks as the basis for a rule refinement system. In J. Hallam (Ed.), *Hybrid problem, hybrid solutions* (pp. 1–12). Amsterdam: IOS Press.

Benitez, J. M., Castro, J. L., & Requena, I. (1997). Are artificial neural networks black boxes. *IEEE Transactions on Neural Networks*, *8*, 1156–1164.

Berenji, H. R. (1991). Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proceedings of the eighth international machine learning workshop* (pp. 475–479).

Browne, A., Hudson, B., Whitley, D., & Picton, P. (2003). Knowledge extraction from neural networks. In *Proceedings of the 29th annual conference of the IEEE industrial electronics society* (pp. 1909–1913).

Callan, R. E., & Palmer-Brown, D. (1997). (S)RAAM: An analytical technique for fast and reliable derivation of connectionist symbol structure representations. *Connection Science*, *9*, 139–160.

Craven, M. W. (1996). Extracting comprehensible models from trained neural networks. Ph.D. Thesis. Department of Computer Sciences, University of Wisconsin-Madison.

d'Avila Garcez, A. S., Broda, K., & Gabbay, D. M. (2001). Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, *125*, 155–207.

Dietterich, T. G., & Flann, N. S. (1997). Explanation-based learning and reinforcement learning: A unified view. *Machine Learning*, *28*, 169–214.

Fu, L. (1994). Rule generation from neural networks. *IEEE Transactions on Neural Networks*, *24*, 1114–1124.

Géczy, P., & Usui, S. (1999). Rule extraction from trained artificial neural networks. *Behaviormetrika*, *26*, 89–106.

Giles, C. L., Lawrence, S., & Tsoi, A. C. (1997). Rule inference for financial prediction using recurrent neural networks. In *Proceedings of the IEEE/IAFE conference on computational intelligence for financial engineering (CIFEr)* (pp. 253–259).

Halgamuge, S. K., & Glesner, M. (1994). Neural networks in designing fuzzy systems for real world applications. *Fuzzy Sets and Systems*, *65*, 1–12.

Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules and its application to medical diagnosis. *Advances in Neural Information Processing Systems*, *3*, 578–584.

Hayashi, Y., Setiono, R., & Yoshida, K. (2000). A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders. *Artificial Intelligence in Medicine*, *20*, 205–216.

Healy, M. J., & Caudell, T. P. (1997). Acquiring rule sets as a product of learning in a logical neural architecture. *IEEE Transactions on Neural Networks*, *8*, 461–474.

Horikawa, S., Furuhashi, T., & Uchikawa, Y. (1992). On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Transactions on Neural Networks*, *3*, 801–806.

Ishikawa, M. (2000). Rule extraction by successive regularization. *Neural Networks*, *13*, 1171–1183.

Kindermann, J., & Linden, A. (1989). Detection of minimal microfeatures by internal feedback. In *Proceedings of the fifth Austrian artificial intelligence meeting* (pp. 230–239).

Masuoka, R., Watanabe, N., Kawamura, A., Owada, Y., & Asakawa, K. (1990). Neurofuzzy systems: Fuzzy inference using a structured neural network. In *Proceedings of the international conference on fuzzy logic and neural networks* (pp. 173–177).

McGarry, K., Wermter, S., & MacIntyre, J. (2001). The extraction and comparison of knowledge from local function networks. *International Journal of Computational Intelligence and Applications*, *1*, 369–382.

Mitra, S. (1994). Fuzzy MLP based expert system for medical diagnosis. *Fuzzy Sets and Systems*, *65*, 285–296.

Okada, H., Masuoka, R., & Kawamura, A. (1993). Knowledge based neural network using fuzzy logic to initialize a multilayered neural network and interpret postlearning results. *Fujitsu Scientific and Technical Journal FAL*, *29*, 217–226.

Omlin, C. W., & Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, *9*, 41.

Rabuñal, J. R., Dorado, J., Pazos, A., Pereira, J., & Rivero, D. (2004). A new approach to the extraction of ANN rules and to their generalization capacity through GP. *Neural Computation*, *16*, 1483–1523.

Saito, K., & Nakano, R. (1988). MEDICAL diagnostic expert system based on PDP model. *Proceedings of the IEEE international conference on neural networks*, *1*, 255–262.

Setiono, R. (2000). Extracting M-of-N rules from trained neural networks. *IEEE Transactions on Neural Networks*, *11*, 512–519.

Setiono, R., Leow, W. K., & Zurada, J. M. (2002). Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, *13*, 564–577.

Sun, R. (2000). Beyond simple rule extraction: The extraction of planning knowledge from reinforcement learners. In *Proceedings of the international joint conference on neural networks*. Piscataway, NJ: IEEE Press.

Takane, Y., Oshima-Takane, Y., & Shultz, T. R. (1999). Analysis of knowledge representations in cascade correlation networks. *Behaviormetrika*, *26*, 5–28.

Thrun, S. (1996). *Explanation-based neural network learning: A Lifelong learning approach*. Boston: Kluwer.

Tickle, A. B., Orlowski, M., & Diederich, J. (1994). DEDEC: Decision detection by rule extraction from neural networks. QUT NRC.

Towell, G. G. (1992). Symbolic knowledge and neural networks: Insertion, refinement, and extraction. Doctoral dissertation. Madison, WI: University of Wisconsin, Computer Sciences Department.

Towell, G. G., & Shavlik, J. W. (1993). The extraction of refined rules from knowledge based neural networks. *Machine Learning*, *131*, 71–101.

Tsukimoto, H. (1997). Extracting propositions from trained neural networks. In *Proceedings of the international conference on artificial intelligence* (pp. 1098–1105).

Tsukimoto, H. (2000). Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks*, *11*, 377–389.

Weijters, A. J. M. M., & van den Bosch, A. P. J. (1999). Interpreting knowledge representations in BP-SOM. *Behaviormetrika*, *26*, 107–129.

Wermter, S., & Sun, R. (Eds.) (2000). *Hybrid neural systems*. Heidelberg: Springer.

Zhou, Z. H., Jiang, Y., & Chen, S. F. (2003). Extracting symbolic rules from trained neural network ensembles. *AI Communications*, *16*, 3–15.