From Radial to Rectangular Basis Functions: A new Approach for Rule Learning from Large Datasets

Michael R. Berthold and Klaus–Peter Huber
Institut für Rechnerentwurf und Fehlertoleranz (Prof. D. Schmid)
Universität Karlsruhe

***Abstract*— Automatic extraction of rules from datasets has gained considerable interest during the last few years. Several approaches have been proposed, mainly based on Machine Learning algorithms, the most prominent example being Quinlan's C4.5.**

In this paper we propose a new method to find rules in large databases, that make use of so–called *Rectangular Basis Functions* (or RecBF). Each RecBF directly represents one rule, formulating a condition on all or a subset of all attributes. Because not all attributes have to be used in each rule, rules tend to be less restrictive and result in a more generalizing rule set.

The rule finding mechanism makes use of a slightly modified constructive algorithm already known from Radial Basis Functions. This algorithm allows to generate the "network of rules" on–line. It starts off with large, general rules and specializes them individually, based on conflicts.

In this paper we present the algorithm to construct the rule base, discuss its properties using a few data sets and outline some extensions.

I. Introduction

In recent years the analysis of huge datasets has gained much interest because modern technical systems are capable to automatically generate large amounts of data from the underlying process. To improve the quality of this process the recorded data has to be analyzed efficiently. Existing statistical methods like a correlation analysis are hard to apply because there are too many parameters and too much data to explore all possible combinations of parameters. This raises the need for methods that are capable to analyze huge amounts of numerical data efficiently and extract information about the underlying process. Questions of interest could range from simple rule extraction (i.e. assigning simple regions in input space to specific classes) to the extraction of the sensibility of parameters, e.g. which range of parameters leads to a satisfying quality.

Today many supervised algorithms that can build classifiers from data examples (input–class pairs) are known. Some of them directly generate rules, others build good classifiers and allow the extraction of their knowledge about the relation between inputs and classes. Usually one concentrates on methods that generate if–then–rules as knowledge representation, because these types of rules are easy to understand by humans and can be used for further machine–based post–processing (rule minimization, sorting etc.)

For most applications in process supervision or control it is desirable to find a small set of "good" rules efficiently. In this context "good" means that the condition part of the rules is small (i.e. only depends on a few important attributes) and that each rule covers a large area of the feature space, meaning it is as general as possible and is highly relevant. In addition the set of rules should describe the data used to find the rules as complete as possible. A 100% coverage is sometimes undesirable, especially if the data set is noisy, in that case the resulting rule–set should describe the important properties of the data set. A good generalization is desired as well, resulting in a rule base that also covers unknown patterns in a meaningful way.

Several approaches to extract rules from data were proposed, the three main directions are based on:

- Knowledge extraction from trained Neural Networks,
- Decision tree or rule learning, and
- Methods that learn hyper–rectangles (or rules) directly.

While *Neural Networks* have been proven to be successful in building good classifiers in many ap-

Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz, Zirkel 2, 76128 Karlsruhe, Germany;
eMail: berthold@ira.uka.de, kphuber@ira.uka.de
Published by the University of Karlsruhe as internal report 15–95

plications, nearly all refuse to tell what they have learned, due to their distributed knowledge representation. Nevertheless, there exist some methods to extract rules of trained neural networks ([16], [17]). But these approaches have to be used carefully for data analysis, because the learning and as well the complex extraction process can be very time consuming. In addition, the rules are extracted from a Neural Network that has been trained on the data. This can lead to an over simplification if the network overgeneralizes and adds another influencing factor that has to be carefully adjusted.

Many *decision rule algorithms* that learn from raw data have already been proposed, two well–known ones are AQ15 and CN2 (see [7] and [2]). Both algorithms take all training examples to construct the rule base. Starting with most general rules they try to find the best matching rules by specializing step by step. The quality of a rule is evaluated by an information measure. Additionally, the maximum number of conditions in each rule is controlled by an user parameter to reduce the search space. Since these algorithms are batch oriented (they have to know all examples before training starts) and are using beam search, rule generation can be a very time consuming process. Another group of algorithms builds decision trees, for example Quinlan's wellknown ID3 and C4.5 (see [11] and [12])). The underlying idea is to build a classifier by recursively dividing the parameter space of numeric (or nominal) attributes into regions that can be assigned one class. This results in a classifier in form of a tree which can be easily transformed into rules by traversing every path from the root to one leaf and generating the corresponding rule (In C4.5 a more sophisticated strategy is implemented, that analyzes the tree as well as the data). Although C4.5 is a very efficient tool, in the context of data analysis two problems arise. First, the rules generated by C4.5 are ordered, which makes the interpretation of isolated rules more difficult. And secondly, the used statistical significance tests have the consequence that the rules do not classify all examples correctly. This may be helpful for noisy data, but has to be carefully controlled to avoid over–generalization.

The third kind of algorithm is based on the idea to build a classifier directly by constructing n–dimensional rectangles (assuming n attributes). Each rectangle specifies a region that belongs to one class (see [15], [18]) and can be easily transformed into one if–then–rule. These approaches are primarily designed to build good classifiers, which makes it hard to minimize the number of rectangles or number of attributes that are used to describe one rectangle. As a consequence the resulting rule base can be very large with many highly specialized rules.

In the following we will present an algorithm that builds a so–called *Rectangular Basis Functions Network* (RecBFN). RecBFNs also consist of a set of hyper–rectangles, but the used training algorithm is derived from Radial Basis Function Networks. By extending an efficient, constructive algorithm the final rules are as large as possible and use only some required attributes. This leads to well generalizing rules with small condition–parts. Another feature offered by this method is the extraction of two types of rectangles (or rules). One rectangle describes the region with high confidence (*required rules*), the smallest rectangle that just covers a subset of the training points belonging to one class. The other rectangle describes the largest possible region, i.e. a rule with lower confidence (*sufficient rules*), that just avoids examples from conflicting classes. In addition, the underlying algorithm enables controlled overlap between conflicting rules, which can be helpful to find few, general rules in noisy data.

This paper is organized as follows: section IIintroduces the basic unit of RecBFN's, the Rectangular Basis Function. The next section shows how single units build up a network. Section IV.presents the algorithm to construct these networks (as well as the rulebase) dynamically. In section Vsome experiments are discussed and finally we conclude with a discussion of RecBFNs, some extensions to the algorithm and ideas for future work.

II. Rules and Rectangles

The approach presented here originated from the idea that rules are usually local in nature, a rule operates on a part of the feature space in opposite to a function. This led to the idea of using Neural Networks with local nature, the most popular example being Radial Basis Function Networks (see [8], [1]). Here each unit can be interpreted independently of all other units. Unfortunately the radial nature of these functions only describes the knowledge with a center and a standard deviation. But in most cases, rules of the form:

$$\text{if } (x_1 \in [x_1^a, x_1^b]) \wedge \cdots \wedge (x_n \in [x_n^a, x_n^b]) \text{ then class } c$$

are desired (with x_i^a being the lower and x_i^b the upper bound of the interval corresponding to the i–th attribute).

The idea of *Rectangular Basis Functions* (RecBF) presented in this paper[1] allow the one–to–one correspondence between Basis Function and rule.

[1] Local Basis Functions with rectangular basis were already used in [10] but with different focus. Here, the rectangular nature was the result of an approach to get rid of the multiplication to better suite hardware implementations — an approach long before used by Intel Corp. for their Ni1000 Radial Basis Function Coprocessor (see [3]), but never explicitly published.

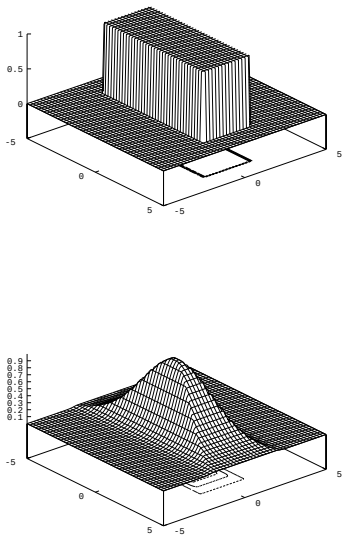

Figure 1: Rectangular Basis Functions in 2D–space, using the Signum function (top) or Gaussian (bottom) as an activation function along each axes.

The well–known activation functions $R(\cdot)$ used for Radial Basis Functions (RBF, see [8]) are modified slightly to the following form:

$$R(\vec{x}) = \min_{1 \le i \le n} A(x_i, r_i, \sigma_i) \qquad (1)$$

were the function $A(\cdot)$ computes the projection on dimension i of the chosen activation function. $\vec{x}$ denotes the input vector and $\vec{r}$ the reference vector of the RecBF. Instead of using only one radius (which results in the radial–symmetric nature) n individual radii σ_i are used. Now, the σ_i represent a set of individual radii along each dimension. Two examples for two–dimensional RecBFs are shown in figure 1. For these examples, $A(\cdot)$ would have been for the Signum function:

$$A(x_i, r_i, \sigma_i) = \begin{cases} 1 & : \quad |x_i - r_i| \le \sigma_i \\ 0 & : \quad \text{else} \end{cases} \qquad (2)$$

and if the Gaussian is used:

$$A(x_i, r_i, \sigma_i) = e^{\frac{(x_i - r_i)^2}{\sigma_i^2}} \qquad (3)$$

The Signum function allows an easier — or at least more obvious — rule extraction, while Gaussians would result in a smoother function of a classifier.

Each RecBF has an additional super–script c, that indicates the class the RecBF belongs to. Now each

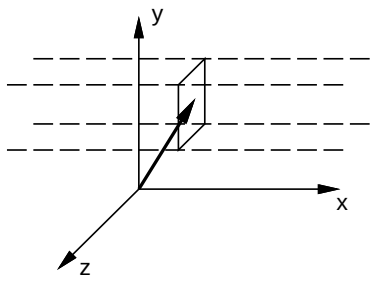

Figure 2: A rectangle in 3D that has finite radii parallel to the y– and z–axes but is not limited along the x–axis. This rectangle represents a rule depending only on the attributes y and z.

RecBF can be straightforward (without any additional processing) interpreted as a rule of the form:

$$\text{if } \forall 1 \le i \le n : x_i \in [r_i - \sigma_i, r_i + \sigma_i] \text{ then class } c \quad (4)$$

One of our primary goals was to extract rules, which operate only on some significant attributes. So far, every rule has a finite width parallel to each axe, which is equivalent to a rule depending on each and every attribute. One solution to solve this dilemma are radii that are allowed to be infinite. This results in rectangles that can have infinite dimensions parallel to a subset of coordinate–axes, representing rules that do not depend on the attributes associated with these axes. Figure 2 shows an example of a rectangle that has finite dimensions parallel to the y and z axes but is not limited along the x–axis. This would represent a rule not depending on the attribute x:

$$\text{if } (y_{\min} \le y \le y_{\max}) \wedge (z_{\min} \le z \le z_{\max}) \text{ then class } c$$

III. Rectangular Basis Function Networks

Rectangular Basis Function Networks (RecBFN) are equivalent in structure to Radial Basis Function Networks, also consisting of an input, one hidden, and an output layer. The only major differences are the activation functions of the hidden units and the propagation rule. RecBFNs use rectangular basis functions in contrast to the radial ones used for RBFNs. Furthermore RecBF Networks can be seen as using a Manhattan distance that replaces the Euclidean distance used by RBFNs.

An example of a full RecBF Network is shown in figure 3. For classification the trained network receives vector $\vec{x}$ as an input and R_i indicates the activation of one RecBF–unit with reference vector $\vec{r}_i$. In this illustration the weight vector that connects all input units to one hidden unit represents the center of the Rectangle. Each RecBF computes

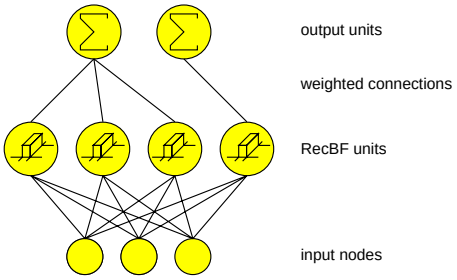

Figure 3: The structure of a Rectangular Basis Function Network (RecBFN).

a value that indicates the membership of vector $\vec{x}$ to the class this RecBF belongs to. This will either be a value equal to 1 if the input vector lies within the boundaries of the rectangle or 0 if not. In case of more complex activation functions (i.e. Gaussians or even membership functions as used in Fuzzy rules) the activation of the RecBF indicates the degree of membership. The output layer only computes a weighted sum of the activations of the hidden units; often this sum is than fed through a squashing function like a sigmoid. The weights are simply set to the number of training patterns that were covered by the specific RecBF.

Additionally extraction of rules is easy (since each RecBF represents one rule) and all rules can be combined into a disjunctive form:

$$\text{if } R_1^c \vee \cdots \vee R_{m_c}^c \text{ then class } c \tag{5}$$

where m_c indicates the number of rules for class c.

In section IV.an example is shown how a RecBF Network is being constructed in two–dimensional feature space.

IV. Training of RecBFNs

Since RecBFs are very similar to Radial Basis Functions and their structure is actually the same, using already existing algorithms for RBF Networks seems a reasonable approach. To avoid problems with a–priori fixed number of units in the hidden layer, which is common for most known algorithms (see for example [8], [19]) a constructive algorithm is more appropriate.

A. The Dynamic Decay Adjustment Algorithm

We use the Dynamic Decay Adjustment Algorithm (DDA) that was introduced recently in [1] which extends the RCE–algorithm (see [6], [13]). In short the main properties of the DDA–algorithm are:

- **constructive training**: new RBF–nodes are added whenever necessary. The network is built from scratch, the number of required hidden units is determined during training. Individual radii are adjusted dynamically during training.
- **fast training**: usually about five training cycles (epochs) are needed to complete training, due to the constructive nature of the algorithm.
- **guaranteed convergence**: the algorithm can be proven to terminate.

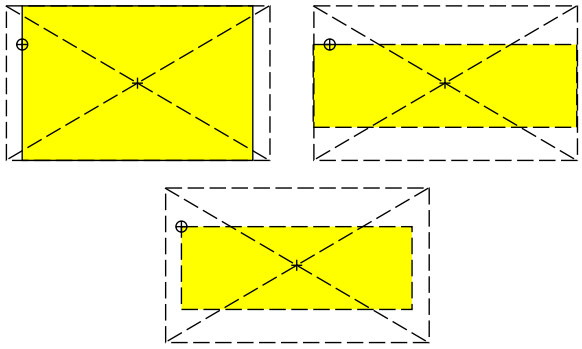

Figure 4: Three ways to shrink a rectangle to avoid a conflicting point

The introduction of this algorithm made the application of RBFNs easy since neither network architecture (i.e. number of hidden units) nor critical parameters have to be determined manually. The algorithm is based on two steps:

- **commit**: if a new pattern is not covered by a prototype of the correct class a new hidden unit will be introduced. Its reference vector will be the same as the new training instance and the width will be as large as possible, without running into conflict with already existing prototypes.
- **shrink**: if a new pattern is incorrectly classified by an already existing prototype of conflicting class, this prototype's width will be reduced (e.g. shrunk) so that the conflict is solved. In the case of RBFs this shrinking only affects the standard deviation of the radial activation function. In the case of rectangles shrinking is more complicated as will be shown later.

B. Extending the Algorithm to Rectangular Basis Functions

As illustrated in figure 4 there is no unique way of shrinking a rectangle so that a specific point will lie outside of the modified rectangle. Instead only one of the dimensions or even a combination of them can be shrunk. All these approaches would result in a smaller rectangle where the conflicting point would lie on the outside. Obviously there are two goals in shrinking a RecBF:

1. the conflicting point has to lie outside of the shrunken rectangle.
2. the rectangle should shrink as little as possible, to avoid RecBFs that are too small (i.e. highly specialized rules).

It is easy to show that it is not necessary to shrink more than one dimension:

- A point $\vec{x} = (x_1, \cdots, x_n)$ lies inside of a rectangle i if and only if:

$$\forall 1 \leq j \leq n : x_j \in [r_{i,j} - \sigma_{i,j}, r_{i,j} + \sigma_{i,j}] \quad (6)$$

- It is sufficient to decrease one σ_i to guarantee that $\vec{x}$ lies outside of the rectangle because negation of equation 6 leads to:

$$\exists 1 \leq j \leq n : x_j \notin [r_{i,j} - \sigma_{i,j}, r_{i,j} + \sigma_{i,j}] \quad (7)$$

- It is critical to select the dimension to shrink carefully to maximize the volume of the remaining rectangle. The loss in volume, if dimension j is shrunk, can be expressed as follows:

$$(\sigma_{i,j} - |r_{i,j} - x_j|) \prod_{1 \leq k \leq n, k \neq j} \sigma_{i,k} \quad (8)$$

$$= (\sigma_{i,j} - |r_{i,j} - x_j|) \frac{\prod_{1 \leq k \leq n} \sigma_{i,k}}{\sigma_{i,j}} \quad (9)$$

  The constant term $\prod_{\cdots} \sigma_{i,k}$ can be omitted and minimization of the following expression is sufficient to find the optimum dimension along which to shrink:

$$\frac{(\sigma_{i,j} - |r_{i,j} - x_j|)}{\sigma_{i,j}} \quad (10)$$

RecBFs with infinite dimensions add some difficulties. Restricting a previously infinite dimension to a finite interval would always mean loosing an "infinite volume". Therefore shrinking of already finite dimensions would be preferred and the rectangle would degenerate to a thin line stretched out along $n-1$ dimensions. To avoid this kind of dilemma, one restriction is enforced on each dimension σ_i, namely a minimum value $\sigma_{i,\min}$.

The patches to the DDA algorithm required to train Rectangular Basis Functions are now easy to apply. The only difference is the procedure "shrink" that has to deal with a set of radii instead of one standard deviation. According to the criteria shown above it selects the dimension where the minimum loss in volume is to be expected and shrinks the rectangle along this axis. The complete algorithm will be shown in the next paragraph.

One additional patch includes asymmetric rectangles. In the case of a signum activation function, symmetry of the rules towards their reference vector is not necessary. Therefore a second set of radii is introduced and the (hyper–) rectangle can have different radii in the positive and negative direction of each coordiante axe. In addition this feature makes the shrinking of rectangles almost independent on the order of training patterns. Figure 5 shows an example where the normal algorithm introduces 3 RecBFs, while only one asymmetric RecBF is needed. This example demonstrates nicely how starting with pattern 3 would have resulted in only one rectangle. Using asymmetric rules this effect can be supressed.

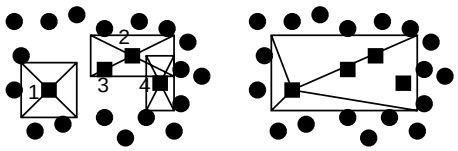

Figure 5: Depending on the order of training patterns, three symmetric rectangles are needed in the worst case (left). For asymmetric RecBFs the order of patterns looses its influence and only one RecBF is needed (right).

C. The modified DDA–algorithm for RecBFNs

Before presenting the modified DDA–algorithm for RecBFNs, a few definitions are required: each RecBF or *prototype* p_i^c of class c with index i ($1 \leq i \leq m_c$, m_c being the number of prototypes of this class) consists of:

- an activation $R_i^c(\cdot)$
- a reference vector (*center*): $\vec{r}_i^c = (r_{i,1}^c, \cdots, r_{i,n}^c)$
- an amplitude (or *weight*): A_i^c
- two sets of "radii" (or standard deviations in the case of Gaussians):
  - set of axes along which the rectangle is spread out towards infinity: K^∞
  - set of axes K^e along which the rectangle is restricted with a radius of σ_j. Only here the σ_j are meaningful.

  Now the following equations hold:
  - $K^\infty \cap K^e = \emptyset$
  - $K^\infty \cup K^e = \{\sigma_1, \cdots, \sigma_n\}$

In addition a set of "minimal radii" is defined:

$$(\sigma_{1,\min}, \cdots, \sigma_{n,\min})$$

(possibly defined via the variance of the single components in the training data).

This leads to a new shrink–procedure for the algorithm as shown in table 1. Three cases are considered. If an existing finite dimension can be shrunk

PROCEDURE shrink

// compute new σ's for three cases:
// 1) the largest new σ for a previously
// infinite one:
$\sigma_{\max,i} = \max\{|r_i - x_i| : \sigma_i \in K^\infty\}$
// 2) the σ with the smallest loss in volume
// for a finite one:
$\sigma_{\min,j} = \min\{|r_j - x_j| : \forall 1 \leq l \leq n, j \neq l :$
$\frac{\sigma_j - |r_j - x_j|}{\sigma_j} \leq \frac{\sigma_l - |r_l - x_l|}{\sigma_l}\}$
// 3) same as 2), but with $\sigma \geq \sigma_{\min}$:
$\sigma_{best,k} = \min\{|r_k - x_k| : \forall 1 \leq l \leq n, k \neq l :$
$(\frac{\sigma_k - |r_k - x_k|}{\sigma_k} \leq \frac{\sigma_l - |r_l - x_l|}{\sigma_l})$
$\wedge(\sigma_i \geq \sigma_{i,\min})\}$
IF σ_{best} does exist **THEN**
$\sigma_k = \sigma_{best,k}$
ELSEIF $\sigma_{\max,i} \geq \sigma_{best,j}$
$\sigma_i = \sigma_{\max,i}$
ELSE
$\sigma_j = \sigma_{\min,j}$
ENDIF
END PROCEDURE

Table 1: The new SHRINK–procedure for RecBFNs.

ALGO DDA–RecBFN
// reset weights:
FORALL prototypes p_i^x **DO**
$A_i^x = 0.0$
ENDFOR
// train one complete epoch
FORALL training pattern $(\vec{x}, c)$ **DO**:
IF $\exists p_i^c : R_i^c(\vec{x}) = 1$ **THEN**
$A_i^c += 1.0$
ELSE
// "commit": introduce new prototype
add new prototype $p_{m_c+1}^c$ with:
$\vec{r}_{m_c+1}^c = \vec{x}$
FORALL $1 \leq i \leq n$ **DO**
$\sigma_i = \infty$
FORALL $k \neq c, 1 \leq j \leq m_k$ **DO**
$p_{m_c+1}^c$.SHRINK(p_j^k)
$A_{m_c+1}^c = 1.0$
$m_c += 1$
ENDIF
// "shrink": adjust conflicting prototypes
FORALL $k \neq c, 1 \leq j \leq m_k$ **DO**
p_j^k.SHRINK$(\vec{x})$
ENDFOR
END ALGO

Table 2: The DDA–algorithm. p_i^c indicates prototype i of class c, A_i^c the corresponding weight, $\vec{r}_i^c$ the center vector and σ_i^c the standard deviation. See text for further explanation.

without shrinking below $\sigma_{\min}$ the one with the smallest loss in volume will be chosen. If this is not the case either one of the remaining infinite dimensions will be shrunk, or if this would result in a new radius smaller than $\sigma_{\min}$ one of the existing finite dimensions will be shrunk below $\sigma_{\min}$.

The complete code using this procedure to perform training for one epoch is shown in table 2.

First, all weights are set to zero because otherwise they would accumulate duplicate information about training patterns. Next all training patterns are presented to the network. If the new pattern is classified correctly, the weight of the closest prototype is increased; otherwise a new prototype is introduced, having the new pattern as its center. The last step must include shrinking all prototypes of conflicting classes if their activations are too high for this specific pattern.

D. How does the Algorithm work?

Figure 6 demonstrates how the modified DDA–algorithm works for a small experimental data set in two–dimensional feature space. The dataset consists of four patterns, one of class A and three belonging to the other class B. After presenting the first pattern (class A) one RecBF with infinite axes is created — so far the network was empty. The second pattern, belonging to class B, results in a

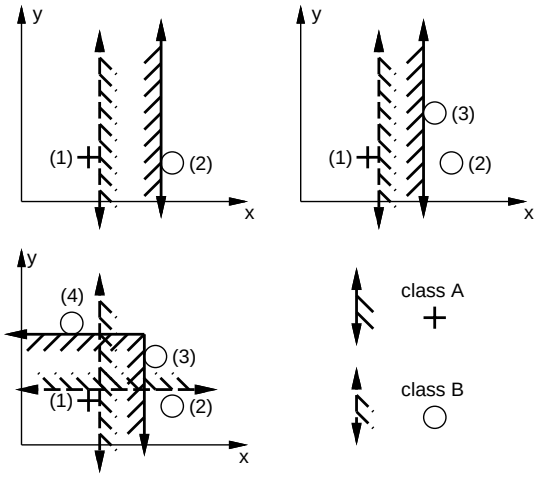

Figure 6: A small example in two–dimensional feature space to illustrate how the RecBF DDA algorithm works. See text for further explanation.

conflict with the existing RecBF. Therefore the one dimension (x^+) which results in the smaller "loss of volume" is shrunk. For class B a first RecBF is be-

ing committed, with an infinite dimension parallel to y and x^+ and a finite width along x^-, to avoid a conflict with the already existing prototype of class A. Pattern 3 is covered by the RecBF of class B and does again result in a conflict with A — it is shrunk along x^+. Pattern 4 finally lies outside of the B–RecBF, thus resulting in a second RecBF of class B and shrinks y^+ of the existing prototype of class A.

The structure of the resulting network will not change during subsequent presentations of the four training patterns. With real world data it could happen that shrinking of prototypes leads to uncovered patterns which would result in a few new RecBFs during the next epochs. Typically training takes only about 3–5 epochs.

V. Experiments

The proposed algorithm for Rectangular Basis Functions was applied to several small data sets to illustrate how the algorithm performs. Results from C4.5 ([12], own experiments) and Thrun's VI–analysis (from [16]) were added to compare the rules found by the RecBFN to other approaches.

A. The MONKs data set

All MONKs problems are discrete classification problems defined in an artificial "robot" domain. The three data sets were used in [4] to compare the performance of different learning algorithms. Robots are described by six different attributes describing their appearance. Three attributes have 3 values, one has 4 and two attributes are binary. In our experiments with RecBFNs all attributes were coded as numerical values. For a larger range of values for one attribute it would be beneficial to use a 1–of–n coding to better suite the RecBFN approach.

A.1. The first MONKs problem

is defined by the following target:

if (HEAD–SHAPE = BODY–SHAPE)
or (JACKET–COLOR = RED)
then $\mathbf{M}_1$

Running the RecBF DDA–algorithm on this problem produces 4 rules for class M_1:

- $(x_1, \cdots, x_6) \in$ (ROUND, ROUND, –, –, –, –)
- $(x_1, \cdots, x_6) \in$ (SQUARE, SQUARE, –, –, –, –)
- $(x_1, \cdots, x_6) \in$ (OCTAGON, OCTAGON, –, –, –, –)
- $(x_1, \cdots, x_6) \in$ (–, –, –, –, RED, –)

Considering that it is not (yet?) possible to have rules of the form $x_i = x_j$ represented by RecBFs this set of four rules is describing class M_1 in the best possible way. The first three rules describe the fact that HEAD–SHAPE = BODY–SHAPE and the fourth rule represents the other possibility JACKET–COLOR = RED.

Experiments with C4.5 resulted in the same four rules as produced by the RecBF–DDA. In [16] nine rules are found that need further manual simplification to find the final rule.

A.2. The second MONKs problem

has a more complex target (note that it is not easily expressed in a simple disjunctive form):

if exactly two of the six attributes
have their *first* value
then $\mathbf{M}_2$

It is clear that there is no simple compact solution for this problem (see also [16]) because M_2 is a parity problem. Therefore the RecBF–DDA finds a set of small, very specialized rules that describe class M_2. The RecBF network contains 43 rules for class M_2 which describe the problem in a disjunctive form.

Although C4.5 produces only 3 special and 1 default rule to describe M_2, these rules do not represent the underlying target concept in an understandable way — resulting in an error rate of more than 15% on the training data. In [16] an automatic rule extraction is not reported, only the verification of more complex rules that were applied at hand is described.

A.3. The third MONKs problem

is defined by:

if ((JACKET–COLOR = GREEN)
and (HOLDING = SWORD))
or ((JACKET–COLOR = **not** BLUE)
and (BODY–SHAPE = **not** OCTAGON))
then $\mathbf{M}_3$

Note that in this problem the training set contains noisy instances.

This problem is being classified by the RecBF Network using 14 rules for class M_3. Some of them were only committed to cover noisy patterns, but sorting the rules according to their weight results in an ordered list of rules, starting with:

- $(x_1, \cdots, x_6) \in$ (–, **not** OCTAGON, –, –, RED, –)
- $(x_1, \cdots, x_6) \in$ (–, ROUND, –, –, **not** BLUE, –)
- $(x_1, \cdots, x_6) \in$ (–, SQUARE, –, FLAG, GREEN, –)
- ...

Note that the first two rules do overlap and almost describe the second part in M_3. The missing case (square body and green jacket) is covered by several smaller rules.

Using C4.5 in standard configuration results in 6 rules to describe M_3, also difficult to interpret. But if the grouping option of C4.5 is used, the underlying concept is represented by the two M_3–rules. In [16]

only 6 rules are generated for this dataset, which is due to the fact that these rules are extracted from a Neural Network trained on the data, rather than from the data itself. In this scenario the Neural Network already took care of the noisy patterns.

B. The IRIS–data

This very famous dataset from Fisher ([5]) contains three classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other. A good rule extraction system should at least be able to make use of this fact and find the one linear rule that separates the first class from the other two.

RecBFN finds eight rules after three training cycles. One rule for class 1, three for class 2 and four for class 4. The single rule for class 1 is exactly the one to divide class 1 linearly from the other two:

$$\texttt{if } x_3 \in (-\infty, 3.0) \texttt{ then class } 1$$

C4.5 does not find the one rule that separates class 1 from the other two. Class 1 is represented through the default–class and one specialized rule. This makes it hard to find the one underlying global rule. Thrun does not report results on this dataset.

C. Further Results

This approach was also applied to the breast–cancer dataset (see [9]), originating from a real world application. This dataset contains about 700 patterns, each pattern described by 9 real–valued attributes. Interestingly the RecBF network trained on the data performed also very well as a classifier. Only a 5% error rate on the testdata (using 10–fold cross–validation) was achieved, which compares well to existing classifiers (C4.5: 4%, MLP (using RPROP, see [14]): 5%).

In all cases the RecBF network was trained almost instantaneous and needed only about 2–4 cycles through the training data.

VI. Discussion

One obvious problem with all rule finding systems that are based on simple geometric structures (i.e. hyper–rectangles or lines) is the fact that they are not able to find complex rules, xor–relationships for example. The second MONKs problem shows that if the underlying rules can not be represented in a disjunctive form, the RecBFN–approach will find lots of small rules that describe the data. In the future we will include a post–processing which is logic based and will extract rules not only in disjunctive form.

Obviously noise makes it difficult for the underlying DDA–algorithm to construct large rectangles.

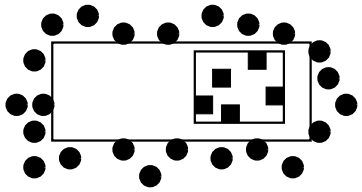

Figure 7: This figure illustrates how the larger, *sufficient* rectangle covers an area that is not fully occupied by patterns of its class. The smaller, *required* rectangle barely covers the patterns of its class, resulting in a rule with a higher confidence.

This is demonstrated by the third MONKs problem, where too many small rules are generated, that also try to classify the noise. This could be solved by a "hit"–counter for conflicting patterns. If there is only one conflict in a specific area it is assumed that this pattern is an outlier and it does not lead to a shrinking of the rule. But this rather heuristic method would add another parameter to the algorithm. Another idea would be to use a non–signum activation function, maybe of triangular or Gaussian shape. Using the two thresholds introduced in [1], one can allow a certain overlap between rectangles of conflicting classes. For the third MONKs problem preliminary experiments show that this approach results in the extraction of only few rules. This rule base does not classify 100% of the training data, but generalizes much better than the "exact" solution.

In addition, we noted that the underlying DDA–algorithm tends to build the rectangles purely based on conflicts. No information about the correct patterns is used to determine the shape of the rule. This leads to very general rules (or large rectangles). We call this type of rectangle *sufficient rule*. Without any complex computation, the DDA can additionally be used to find the smallest rectangle (called *required rule*) inside each sufficient rule. This rectangle will be completely inside the larger sufficient rectangle and will just barely cover all patterns that are covered by the surrounding sufficient rule. Figure 7 shows an example for the two types of rectangles. This feature can be used to extract regions of higher vs. regions of lower confidence. In addition, fuzzy–like soft rules can be defined, using the two intervals as boundaries for the trapezoid.

VII. Conclusions

In this paper we have proposed an extension to Radial Basis Functions that allows an easy extraction of rules. The resulting network has Rectangular Basis Functions which have a clear one–to–one resemblance to rules in their disjunctive form. The Network can be constructed dynamically without any

a–priori knowledge about number of rules (i.e. number of hidden units). The resulting network has one hidden unit for each rule and allows rules that depend only on a few attributes. This leads to a rule–set that contains a small number of general rules. RecBF Networks are constructed automatically and during training no user–interaction is needed.

We have used a few small datasets to show that RecBFNs are actually able to extract meaningful rules from datasets and that the time needed to construct these networks is low. If there is not much noise the number of rules extracted is close to the optimum number of rules needed to represent the underlying model in a disjunctive form. In the case of noisy data, a modification of the underlying training algorithm can be used to reduce the number of rules found.

We are well aware that this is work in progress. A deeper evaluation of the algorithm, scalability with larger datasets and behaviour with real world data are issues of current work.

VIII. Acknowledgement

We would like to thank Prof. D. Schmid for his support and the opportunity to work on this interesting project.

References

[1] Michael R. Berthold and Jay Diamond. Boosting the performance of RBF networks with Dynamic Decay Adjustment. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, 7, Cambridge MA, 1995. MIT Press.

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

[3] Mark Holler et al. A 40,000 pattern/s classification coprocessor with learning capability. In *Hot Chips*, 1993.

[4] S. B. Thrun et al. The MONK's problems – a performance comparison of different learning algorithms. Technical report, Carnegie Mellon Univeristy, Pittsburgh, PA, December 1991.

[5] R. A. Fisher. The use of multiple measurements in taxonomic problems. In *Annual Eugenics, II*, volume 7, pages 179–188. John Wiley, NY, 1950.

[6] Michael H. Hudak. RCE classifiers: Theory and practice. In *Cybernetics and Systems*, volume 23, pages 483–515. Hemisphere Publishing Corporation, 1992.

[7] R. S. Michalski, I. Mzetic, J. Hong, and N. Lavrac. The multipurpose incremental learning system AQ15. In *Proceedings of the National Conference on AI, AAAI*, 5, pages 1041–1045, 1986.

[8] John Moody and Christian J. Darken. Fast learning in networks of locally–tuned processing units. In *Neural Computation*, 1, pages 281–294. MIT, 1989.

[9] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Machine-readable data repository.

[10] W. Poechmueller, S. K. Halamuge, M. Glesner, P. Schweikert, and A. Pfeffermann. RBF and CBF neural network learning procedures. In *International Conference on Neural Networks*, volume 1, pages 407–412. IEEE, july 1994.

[11] J. Ross Quinlan. Induction of decision trees. In *Machine Learning*, 1, pages 81–106, 1986.

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

[13] Douglas L. Reilly, Leon N. Cooper, and Charles Elbaum. A neural model for category learning. In *Biological Cybernetics*, 45, pages 35–41, 1982.

[14] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In *International Conference on Neural Networks*, volume 1, pages 586–591. IEEE, march 1993.

[15] S. Salzberg. A nearest hyperrectangle learning method. In *Machine Learning*, 6, pages 251–276, 1991.

[16] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. In Gerald Tesauro, David Touretzky, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, 7, California, 1995. Morgan Kaufmann.

[17] G. Towell and J. W. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, pages 977–984. Morgan Kaufmann, 1992.

[18] D. Wettschereck. A hybrid nearest-neighbour and nearest-hyperrectangle learning algorithm. In *Proceedings of the European Conference in Machine Learning*, pages 323–335, 1994.

[19] Dietrich Wettschereck and Thomas Dietterich. Improving the performance of Radial Basis Function Networks by learning center locations. In *Advances in Neural Information Processing Systems*, 4, pages 1133–1140, California, 1992. Morgan Kaufmann.