Decision Trees Can Initialize Radial-Basis Function Networks

Miroslav Kubat

Abstract— **Successful implementations of radial-basis function (RBF) networks for classification tasks must deal with architectural issues, the burden of irrelevant attributes, scaling, and some other problems. This paper addresses these issues by initializing RBF networks with decision trees that define relatively pure regions in the instance space; each of these regions then determines one basis function. The resulting network is compact, easy to induce, and has favorable classification accuracy.**

Keywords— **Pattern recognition, neural networks, radial-basis functions, decision trees**

I. Introduction

A system that learns to recognize concepts accepts pairs $[\mathbf{x}, c(\mathbf{x})]$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is a vector describing an *example*, and $c(\mathbf{x})$ is a concept label. The variables x_i are referred to as *attributes*. The space occupied by all possible examples that can be described by the given set of attributes is called the *instance space*. A *concept* is a binary function, $c : R^n \rightarrow \{-1, 1\}$, where '1' is the positive label and '-1' is the negative label. If the training set contains examples of more than two concepts, then $c(\mathbf{x})$ is defined as $c : R^n \rightarrow L$ where L is a set of concept labels. The task of the learning system is to find another function, a *classifier*, $h : R^n \rightarrow \{-1, 1\}$, that approximates the concept.

A popular class of classifiers have the form of *basis-function* networks such as the one depicted in Figure 1. The example to be classified is passed to a set of basis functions, each returning one scalar value, φ_j, $j = 1, 2, \ldots, m$. Although theoretical and practical considerations in [31] indicate that the concrete choice of the basis function is not critical, common implementations prefer *radial* basis functions (the gaussian "bell" functions). Hence the name: *radial-basis-function* networks (RBF networks). The rationale is to transform the instance space in a manner that increases the separability of concepts. The idea was first cast into the neural-network setting by [5].

Suppose that the examples have been normalized and that the standard deviation along each attribute is σ. The following equation describes an n-dimensional gaussian surface, centered at $\boldsymbol{\mu_j} = [\mu_{j1}, \ldots \mu_{jn}]$:

$$\varphi_j(\mathbf{x}) = \exp\{-\frac{\| \mathbf{x} - \boldsymbol{\mu_j} \|^2}{2\sigma^2}\} \quad (1)$$

$\varphi_j(\mathbf{x})$ measures similarity: the larger the distance between $\mathbf{x}$ and $\boldsymbol{\mu_j}$, the smaller the value of $\varphi_j(\mathbf{x})$. If an example is to be classified, the network first redescribes it

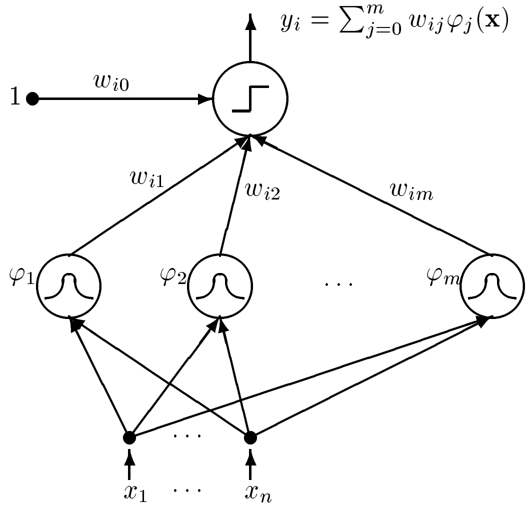

Fig. 1. A radial-basis function network

as $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \ldots, \varphi_m(\mathbf{x})]$. The activation of the i-th output neuron is then calculated as $y_i = \sum_{j=0}^{m} w_{ij}\varphi_j(\mathbf{x})$, where w_{ij} is the weight of the link from the j-th hidden neuron to the i-th output neuron (the weights w_{i0} are connected to a fixed $\varphi_0 = 1$). The example is assigned the i-th concept label if $y_i = \max_k(y_k)$.

It is relatively simple to find the output-layer *weights* w_{ij} using the delta rule [38] or some traditional statistical approach such as the pseudoinverse matrix. Literature usually addresses more difficult problems:

1. [30] identifies each gaussian *center* with the coordinates of one example. However, if the examples are abundant, the network becomes unmanageably large and prone to overfit the data. Therefore, [23] and [22] pick for gaussian centers only a random subset of the training set. This is unsatifactory if the examples are noisy. More sophisticated approaches pre-cluster the examples [24], [26] or use a search technique with operators 'add a neuron' and 'delete a neuron' [8], [7]. Methods for tuning the centers were developed by [13], [14] and by [37]. However, most of these techniques are computationally expensive.
2. For proper setting of the *standard deviation*, σ, heuristics are necessary. [23] suggests to use a value proportional to the maximum distance between centers. In [33] each hidden neuron has a different σ, calculated as a function of the distance between its center and the center of its nearest neighbor. Adaptation formulae for learning σ from examples can be found in [17].

M. Kubat is with the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504-4330, and with the Department of Computer Science, Southern University–Baton Rouge, Baton Rouge, LA 70813-0400. E-mail: mkubat@cacs.usl.edu

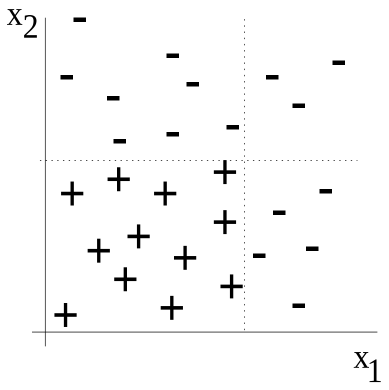

Fig. 2. The relevance of attributes varies in different parts of the instance space.

The ideal σ is often different for each dimension, especially when each attribute has a different *scale*. The search for these values can, again, be expensive.

3. Radial-basis functions are closely related to nearest-neighbor classifiers [9], [10], sharing their sensitivity to *irrelevant attributes* that can distort many similarity metrics, including the one from Equation 1.
   Importantly, the relevance of individual attributes can vary in different parts of the instance space as illustrated by Figure 2: for $x_1 < 5$, both attributes are needed; in the subspace $x_1 > 5$, the information about x_2 is irrelevant; and for $x_2 > 4$, the information about x_1 is irrelevant. This peculiarity is rarely considered [1].
4. Finally, a limiting factor for *hardware implementations* is the number of links. If a RBF network has 100 hidden neurons, each connected to 100 inputs, then the number of links is 10^4. This constrains the capacity of a VLSI chip where the values are digitized and each link is represented by 3–5 wires.

To summarize, the design of a successful RBF network involves several nontrivial problems, and so far there does not seem to exist any simple and general algorithm, or heuristic, addressing them all at the same time. To rectify the situation, the paper suggests a simple mechanism that designs the RBF network from a "blueprint" drawn by a decision tree (the system will be referred to as TB-RBF: a tree-based radial-basis functions). The next section will show how each of the above issues is addressed in a fairly straightforward manner.

Decision trees have been thoroughly investigated during the last decades [3], and advanced software packages for their induction from data in a computationally efficient manner are now ubiquitous [32]. Transforming a decision tree to a RBF network is quite easy, and experiments indicate synergy: the RBF paradigm benefits from simple initialization, whereas the classification accuracy of TB-RBF outstrips the original decision tree in domains characterized by nontrivial decision surfaces and noise.

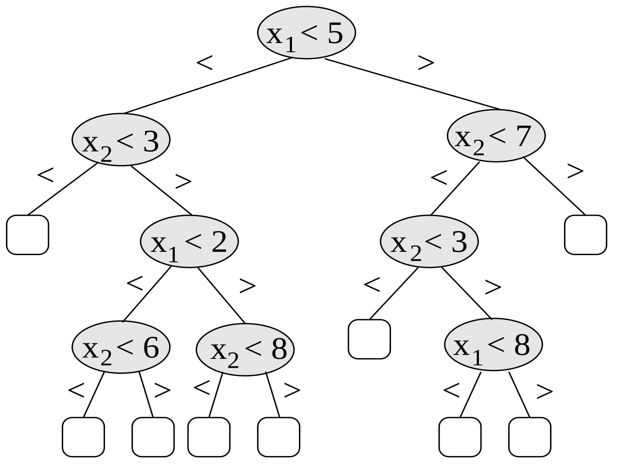

Fig. 3. An example decision tree

II. Decision-Tree-Based Initialization of RBF Nets

Most programs for induction of decision trees decompose the instance space into pairwise disjoint regions. Ideally, each region will contain examples of a single concept. Examples with different concept labels decrease the region's *purity*. The tree-induction process is guided by heuristics to ensure that the tree is small, and still has relatively high purity. Figures 3 and 4 show a decision tree that partitions R^2 (two-dimensional space with continuous attributes) into axis-parallel rectangles. The concept labels associated with these regions have been omited because they are not explicitly used by TB-RBF.

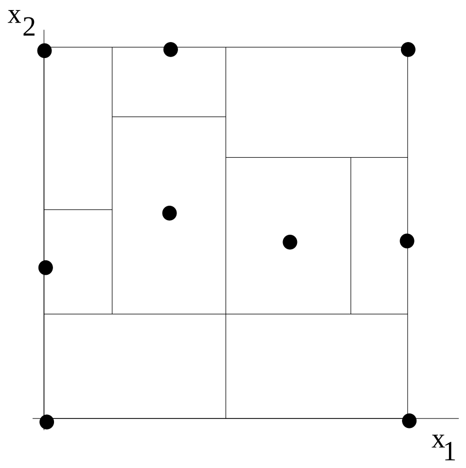

Fig. 4. The decomposed space in R^2

The idea of partitioning pertains to the essence of RBF networks where the function φ_j returns a high value if the example falls into an area close to $\boldsymbol{\mu_j}$. For instance, φ_j can be interpreted as evidence in favor of the label prevailing in the vicinity of $\boldsymbol{\mu_j}$; this evidence is then weighted. The next step in this line of thought is to associate each φ_j with one of the regions defined by the decision tree. Figure 4 shows where the centers $\boldsymbol{\mu_j}$ will be placed by TB-RBF. Note that the instance space is supposed to be bounded, the boundaries being given by the maximum values of the attributes as observed in the training set. Three options are possible:

1. If one and only one side of the region lies on the border of the instance space, then $\boldsymbol{\mu_j}$ is placed at the center of this side;
2. If two adjacent sides of the region lie on the borders of the instance space, then $\boldsymbol{\mu_j}$ is placed into the corner defined by these sides.
3. If the region borders on all sides with other regions, then $\boldsymbol{\mu_j}$ is placed in the geometric center of the region;

The principle of TB-RBF is germane to methods with clustering [26] because each hyperrectangle is, in a way, interpreted as a "cluster" of examples. However, TB-RBF does *not* indentify $\boldsymbol{\mu_j}$ with the centers of gravity of the clusters.

In the following, μ_{jk} will denote the k-th coordinate of $\boldsymbol{\mu_j}$. Let the standard deviation of the j-th gaussian along the k-th dimension be σ_{jk}^2. Equation 1 can be re-written using $\| \mathbf{x} - \boldsymbol{\mu_j} \|^2 = \sum_k (x_k - \mu_{jk})^2$ and the fact that $e^{\Sigma x_i} = \prod e^{x_i}$:

$$\varphi_j(\mathbf{x}) = \prod_{k=1}^{n} \exp\{-\frac{(x_k - \mu_{jk})^2}{2\sigma_{jk}^2}\} \tag{2}$$

The unknowns σ_{jk}^2 determine how steeply $\varphi_j(\mathbf{x})$ decreases with the growing distance from $\boldsymbol{\mu_j}$ along each dimension. An intuitive requirement is to have these parameters depend on the sizes of the hyperrectangles so that the value of φ_j at the hyperrectangle's border is always the same. If I_{jk} is the size of the k-th dimension of the j-th hyperrectangle, a new parameter, α, can be introduced such that $\alpha^2 = I_{jk}^2 / \sigma_{jk}^2$. Note that α is the same for all dimensions and for all j, keeping the ratio between I_{jk} and σ_{jk} constant. Then:

$$\varphi_j(\mathbf{x}) = \prod_{k=1}^{n} \exp\{-\frac{\alpha^2 (x_k - \mu_{jk})^2}{2I_{jk}^2}\} \tag{3}$$

A single branch of a decision tree will often test only a subset of attributes. Since the other attributes are deemed irrelevant in the given subspace, it is enough to calculate the product in Equation 3 only for those values of k that correspond to relevant attributes (for irrelevant attributes the exponential function is set to 1).

What remains to be done is to determine the output-layer weights, w_{ij}. Suppose the transformed training examples have been arranged in a matrix, $\mathbf{X}$, such that each row represents one example and the j-th column contains the value of φ_j for this example. One additional attribute whose value is always 1 will stand for φ_0. $\mathbf{C}$ will denote the *classification matrix* where each column represents one concept label: if the r-th example is labeled with the i-th concept, then the i-th element in the r-th row of $\mathbf{C}$ is set to 1 and all other elements in this row are set to -1. Given $\mathbf{X}$ and $\mathbf{C}$, the objective is to find the weight matrix, $\mathbf{W}$, minimizing the mean square error calculated as $\mathbf{X} \cdot \mathbf{W} - \mathbf{C}$. This is accomplished by means of the *pseudoinverse-matrix*, $\mathbf{X^P}$, using the well-known fact [12] that the mean square error is minimized when:

$$\mathbf{W} = (\mathbf{X^T X})^{-1} \mathbf{X^T C} = \mathbf{X^P C} \tag{4}$$

This technique was chosen for implementational convenience: the system was implemented in MATLAB where the calculation of pseudoinverse matrix is one of the standard functions. The author believes that the method of calculating $\mathbf{W}$ is not decisive for the system's performance.

With this, all necessary steps to define a RBF network using a decision tree have been described. The algorithm is summarized in Table I. As a final remark, note that a decision tree is sometimes pruned: some of its subtrees are cut off or replaced with smaller subtrees [32].

TABLE I
The algorithm of TB-RBF

1. Induce a decision tree that partitions the instance space into hyperrectangular regions.
2. Turn the tree into a RBF net such that each region is represented by one neuron.
3. Determine the output-layer weights using a pseudoinverse matrix.

The transition from hyperrectangles to gaussian functions is illustrated by Figure 5. Here, the decision tree defined three intervals along the given attribute. Over each of the intervals, a gaussian function is placed such that its maximum is 1 and the value at the boundaries is z—this value is determined by α. The classifier's sensitivity to α will be moderate because the values of the adjacent φ_j will be equal at the borders. If there were no weights (or if all weights were the same), then the resulting network with gaussian distributions characterized by the functions φ_j would assign to the examples the same labels as the original decision tree. In TB-RBF, though, the gaussians are multiplied by the weights, w_{ij}, and it is the weighted sum $y_i = \sum_j w_{ij} \varphi_j$ that decides about the actual label.

What does the initialization with decision trees offer to the RBF paradigm? It is instructive to observe how TB-RBF addresses the weaknesses listed in the previous section.

1. The gaussian centers are quite naturally determined by the regions that are relatively pure thanks to the heuristics driving the tree induction. For m training examples and n attributes, the computational costs of the tree induction are upper bounded by $O(nm \log m)$ [32] which is clearly less than the costs of common clustering techniques.
2. The sides of the hyperrectangles define the gaussian spreads separately for each dimension. The param-

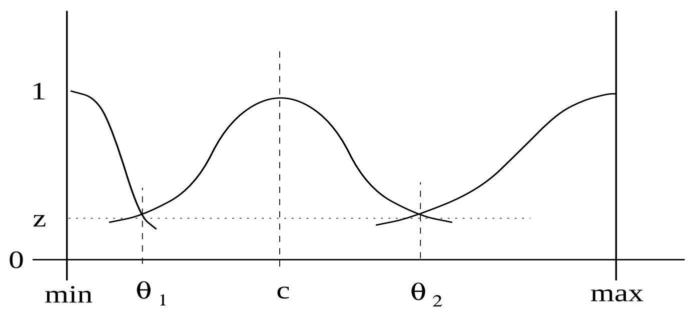

Fig. 5. Gaussian bell functions are placed over intervals determined by C4.5.

eters σ_{jk} are replaced with the single parameter α. Also the scaling problem is solved.

3. Sensitivity to irrelevant attributes is reduced because irrelevant attributes usually do not appear in pruned decision trees. The fact that different attributes can be relevant in different parts of the instance space is reflected: each tree-branch contains tests on attributes relevant in the corresponding subspace.
4. The fact that hidden neurons are connected only to relevant attributes reduces the number of links in the network.

These four issues represent the primary benefits of TB-RBF. Note that the number of user-set parameters is greatly reduced compared to most other implementations of RBF nets. When the decision tree is ready, the only parameter to be set is α (optionally also the extent of tree pruning). On the other hand, decision trees, too, should profit from being transformed to RBF nets. At least, improved classification accuracy in difficult domains is a natural expectation. To examine these merits experimentally is the task for the next section.

III. Experiments

The experimentation was carried out in three stages. First, some intuition about the system's behavior was developed by experiments with simple geometrical concepts. Then, the system was tested on more realistic problems, and the impacts of some parameter values were observed. Finally, the possibility of downsizing the resulting network was investigated.

A. Initial Experiments with Synthetic Data

To investigate TB-RBF's behavior, three artificial domains were synthesized. The training set and the testing set always contained 600 examples each, and the positive and negative examples were equally distributed in the training sets as well as in the testing sets. The domains were named Gauss, Geom1, and Geom2:

Gauss (Gss): Positive examples are randomly generated according to the gaussian distribution with $\boldsymbol{\mu}_+ = (0,0)$ and $\sigma_+^2 = 1$. Negative examples are generated according to the gaussian distribution with $\boldsymbol{\mu}_- = (2,0)$ and $\sigma_-^2 = 4$.

Geom1 (Gm1): Generate, according to the uniform distribution, pairs of random numbers, (x_1, x_2), each number from the interval $(0, 10)$. Label as positive those examples that lie in the shaded area of Figure 6.

Geom2 (Gm2): Generate, according to the uniform distribution, pairs of random numbers, (x_1, x_2), each number from the interval $(0, 10)$. Label as positive those examples that lie in the shaded area of Figure 7.

TABLE II
Performance in synthetic domains. Parameter `-c` determines the degree of pruning. For each set, the first row gives the number of leaves of the tree and the other two rows give the accuracies of TB-RBF (acc RBF) and C4.5 (acc C4.5).

`-c`		1	5	10	25	50	100
Gss	# units	6.1	9.2	10.7	35.0	101.9	109.8
	acc RBF	79.4	81.2	80.2	79.3	79.2	78.2
	acc C4.5	80.1	80.0	80.5	79.5	77.2	76.0
Gm1	# units	14.3	16.0	18.8	53.6	54.0	71.2
	acc RBF	85.5	84.2	83.5	83.2	82.1	80.8
	acc C4.5	81.8	82.0	79.8	80.0	81.0	80.0
Gm2	# units	8.2	11.8	18.2	22.0	82.3	118.3
	acc RBF	80.8	77.2	77.7	78.8	80.5	77.8
	acc C4.5	81.0	78.2	79.8	79.0	78.0	76.8

The decision trees were generated by Quinlan's C4.5. Note that Gauss and Geom1 are not ideally suited for C4.5. The former because of the highly nonlinear decision surface, and the latter for its oblique boundaries. In contrast, Geom2 can easily be represented by a small decision tree. To make the task more difficult, *attribute noise* was introduced in the sets Geom1 and Geom2 by adding a randomly generated number (normal distribution, mean 0 and standard deviation 1) to each attribute in each example so that the real attribute value is $a' = a + 0.2ra$ where r is the random number. Attribute noise creates overlaps between the positive and negative regions.

These domains were not selected arbitrarily. Noisefree tasks with axis-parallel boundaries would be easy for C4.5, and no improvement by RBF could reasonably be expected. Whereas the presence of noise is easily overcome by C4.5 (especially with pruning), oblique boundaries and nonlinearities pose problems to be rectified by the RBF part.

The results are summarized in Table II, where the first column contains the information about the parameter that controls pruning (`-c`). The value `-c 25` represents default pruning in C4.5, smaller values lead to small trees, whereas large values imply insignificant pruning and, therefore, large decision trees. For each domain, the table contains information about the number of leaves in the decision tree (units of the RBF network) and about the accuracy achieved by TB-RBF and C4.5 on an independent testing set.

Expectedly, substantial pruning (small values of `-c`) was needed to overcome the negative impact of concept overlap and noise on decision trees. Large trees overfitted the

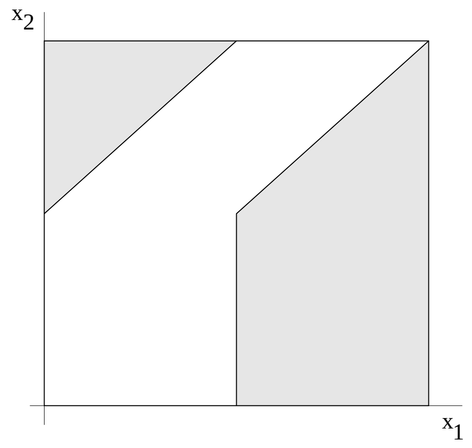

Fig. 6. Geom1 data file

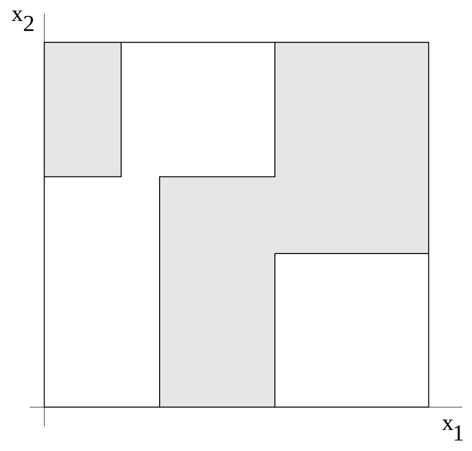

Fig. 7. Geom2 data file

training data (hence the poorer results on testing sets) but this overfitting was compensated with TB-RBF, perhaps thanks to its tendency to defined smooth decision surfaces and to "outvote" erroneous evidence. This is seen in the rows representing higher values of `-c` (`-c 50` and `-c 100`). In Geom1, TB-RBF easily compensated for the obliqueness of the decision surface and clearly outperformed the original decision tree even in the case of heavily pruned trees. The margin somewhat dissipated with growing trees.

In summary, the accuracies achieved in the synthetic domains do encourage further experimentation. However, these results must be complemented by experiments with more realistic domains.

B. Experiments with Benchmark Data

The benchmark data have to be carefully chosen. The primary claim to be verified is that TB-RBF has a performance edge in domains where the decision surfaces are non-linear or oblique, and the data are noisy. The focus is on domains with continuous attributes. Implementational

TABLE III
Benchmark data files

	#attr.	# concepts	# ex.	majority
bupa	6	2	345	58.0
diab	8	2	768	65.1
glass	9	6	214	35.5
vowels	10	11	990	9.1
kr2	15	7	931	32.4
vehicles	18	4	846	28.3

convenience (the use of pseudoinverse matrix) limited the experiments to domains with no more than, say, a thousand examples.

With this in mind, the following public-domain benchmark data were selected from the UCI repository [25]: liver disorders (bupa), diabetes (diab), glass, vowels, and vehicles. The file kr2 comes from the author's previous research [21]. Basic information about the data is summarized in Table III: the number of attributes, the number of examples in the entire data set, and the number of different concept labels. The rightmost column gives the percentage with which the majority concept is represented. This number can be understood as the performance that would be achieved if the system consistently labeled all testing examples with the label of the most frequently represented concept.

In cases where the two different techniques (TB-RBF and C4.5) were compared, the statistical confidence of the results was assessed by the t-test with confidence level 95%. Dietterich, after a detailed analysis of the utility of t-tests for the needs of concept learning suggests as a good methodology a technique called the "5x2cv" test [11]. The data file is randomly split into two equally sized parts, S_1 and S_2. The learner takes S_1 for training and S_2 for testing, and then S_2 for training and S_1 for testing. This is repeated for 5 different splits so that, in effect, the approach implements 5 replications of 2-fold cross-validation. For details see [11].

The performance of TB-RBF is demonstrated by Table IV which compares the accuracies of TB-RBF and C4.5. The *default pruning* in C4.5 (`-c 25`) was used to initialize the RBF network. TB-RBF outperforms C4.5 in all domains, sometimes quite persuasively (by 10% in *vowels* and *vehicles*, and by 5% in *kr2*). With the exception of the *glass* domain, the results are statistically significant.

The same table also gives the size of the resulting RBF network expressed in terms of the number of hidden neurons, and in terms of the number of links from the input layer to the hidden layer. The number of hidden neurons is equal to the number of branches of the decision tree. The number of links leading to a hidden neuron is given by the number of attributes tested in the corresponding tree-branch.

In all these experiments, the values of `-c` and α were

TABLE IV
TB-RBF HAS AN EDGE OVER DECISION TREES; 'ACCP' STANDS FOR ACCURACY OF THE DECISION TREE AFTER DEFAULT PRUNING.

file	TB-RBF			C4.5	
	# units	# links	acc	acc	accp
bupa	24.0	104.8	68.8	65.7	65.3
diab	42.6	215.0	74.8	70.7	71.8
glass	16.4	72.9	62.8	60.9	61.7
vowels	70.5	358.6	80.1	70.7	71.0
kr2	65.6	321.0	65.1	59.1	60.2
vehicles	57.2	395.0	74.6	64.5	64.9

fixed, and one can rightly ask whether changed values of these parameters will significantly affect the system's classification performance. Auxiliary experimentation (not reported here in detail) showed that in most benchmark domains the classification performance was better when the pruning was only moderate if any. In five out of six domains, this difference was statistically significant. The exception was the domain *diab* where no difference was observed and the domain *glass* where the improvement (with large values of `-c`) was not significant. At the other extreme, in the domains *bupa* and *vehicles* the difference in classification accuracy on testing data exceeded 5%. The experiments suggest that whereas pruning has a well-known merit in the induction of decision trees, TB-RBF seems to be able to compensate for the performance loss caused by overfitting. However, beware of the obvious trade-off: unpruned trees are large, and as such lead to large RBF networks.

The parameter α controls the spread of the gaussian functions. Detailed experimentation with different values of α (ranging from 0.5 through 2.5) showed that TB-RBF was relatively insensitive to the setting of this parameter. With one exception (*glass*) the accuracy was maximal by $\alpha = 1.0$.

The reader is reminded of TB-RBF's limitations: in simple tasks that are easy for decision trees, no increase in accuracy can reasonably be expected from mapping the tree to a RBF network. For illustration, this would be the case of the synthetic domain Geom2 (see the previous section) if there were no noise.

C. Trimming the RBF Network

The replacement of the hyperrectangles with gaussian functions obviously enhances the flexibility of the classifier. Considering the still somewhat great size of the decision trees generated by C4.5, one might ask whether a much smaller RBF network would yield similar classification performance. Indeed, are those dozens of hidden neurons necessary? What if some of them are removed?

To investigate the dependence of the classification performance on the tree size in more detail, a simple mechanism was implemented. The principle is depicted in Figure 8. First, the whole decision tree is taken, turned into a RBF network, and tested on independent data. Then, the longest sequence of tests is taken (ties are broken arbirarily or "from the left to the right") and the bottom test is replaced with a leaf. The reduced tree is, again, turned into a RBF network and tested on independent data. This is repeated until a rudimentary decision tree consisting of no more than a few tests is reached. The transition from a larger tree to a smaller tree actually results in the removal of one of the hyperplanes that bisect the corresponding parts of the instance space; occasionally, the gaussian centers have to be shifted as indicated in the figure.

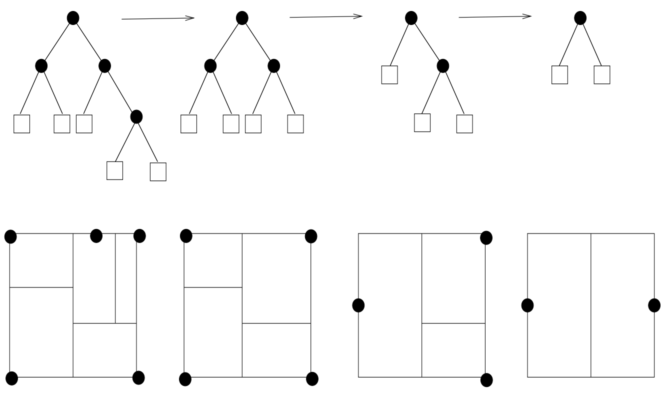
Fig. 8. Reducing the number of regions

Figures 9 and 10 show, for two domains, how the performance of the RBF network depends on the number of hidden neurons (the plots for the remaining four domains would be similar). The solid line plots the accuracy on the testing set, and the dotted line plots the accuracy on the training set. The graphs were obtained as averages of three runs where randomly selected 70% of the given data file were used for training, and the remaining data were used for testing (note that the accuracies are now slightly higher than in the previous section because more data have been used for learning).

In all graphs, the performance tends to grow slightly with shrinking RBF networks, until they reach a maximum. Only for a very small number of units (typically less then 5-10) does the classification performance degrade. Again, these results are somewhat at variance with the resuls from the previous section because of the larger training set (70% as opposed to the previous 50%). However, the main lesson is that it is not necessary to use the entire decision tree generated by C4.5. It is also not necessary to prune it. Rather, the mapping mechanism can simply use some vaguely defined "upper part" of the tree, say one third of it.

Thus a relatively small RBF network will often significantly outperform C4.5. Thanks to the computational effectivity of C4.5, the network is fast to build (the computational costs of the tree-to-network mapping are low). The number of links is moderate thanks to the nature of decision trees. Irrelevant attributes are dismissed.

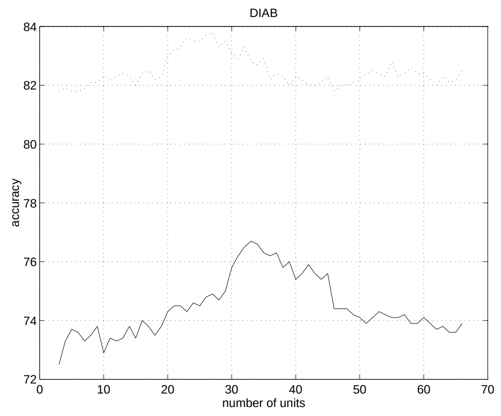

Fig. 9. Reducing the classifier in the domain diab. Solid: testing set acc; dotted: training set acc.

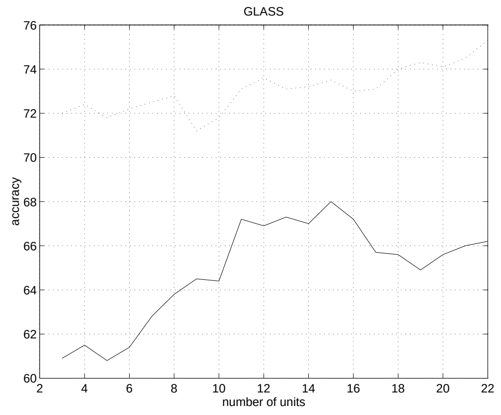

Fig. 10. Reducing the classifier in the domain glass. Solid: testing set acc; dotted: training set acc.

IV. Related Research

The possibility to map decision trees on RBF networks was first suggested in a short conference communication [20]. Their system differed from the current implementation in several details such as the specific basis function; the way gaussian centers were determined; and the method to determine the output-layer weights.

The idea of initializing neural networks with symbolic techniques is not new. [35] was the first to map decision trees to feedforward neural networks with sigmoidal activation functions. Variants of this approach were later proposed by [4], [15], [18] and [2]. Unfortunatelly, feedforward networks are expensive to train, and the abundance of their parameters can render the training procedure inefficient if the training set is small. These considerations led to the implementation of TB-RBF.

Other workers investigated methods initializing neural networks (feedforward with sigmoids) using production rules. The idea was to implement domain knowledge, specified by a human expert, in a network. This was done, for instance, by [36] and by [16]. [27], [28] describe techniques that further improve the rule-based architecture of the neural network. Generally speaking, all these approaches can use the scenario from this paper where the knowledge is induced from the data by machine learning rather than elicited from an expert. The specificity of decision trees is that they usually totally decompose the instance space.

V. Conclusion

Why would a person want to initialize RBF networks with decision trees? First of all, the regularities of the instance space (with respect to the given concepts) are "discovered" by a decision tree generator that provides the first approximation of the learned concept. The decomposition of the instance space then directly answers many questions faced by RBF-network designers: how many gaussian centers are needed, where to place them, how to adjust the spreads, how to detect and remove irrelevant attributes, what to do with scaling. Also the number of links is reduced. Moreover, one of the caveats frequently raised by critics of connectionism is the large number of parameters whose values have to be adjusted by the user. The only parameters in TB-RBF are the tree-pruning extent and the constant α that controls the slope of the gaussians. Experiments show that the performance is fairly insensitive to the actual setting of either of them.

Thanks to the effectivity of the decision-tree generator (C4.5) the method poses only moderate computational requirements. The tree-to-network mapping is straightforward and does not represent any significant computational burden. The weights in the resulting network are determined by a pseudoinverse matrix, which can be expensive when the training set is large. In that case, some less demanding technique, such as the delta rule, is recommended.

Amalgamation of connectionist and symbolic approaches definitely shows merit. Whereas symbolic representations tend to be too coarse for many concepts in numeric domains, connectionist algorithms entail high computational costs and the need of large training sets. Both of these shortcomings can be partly eliminated by appropriate cooperation of the two paradigms.

Although the paper focused on the classification task (concept recognition), the author believes that the basic idea can be used also for function approximation—the only change would be the necessity to work with regression trees instead of classification trees. This is a topic for further research.

Acknowledgement

Part of this research was done during the author's stay at the University of Ottawa. Thanks to Irena Koprinska (Ivanova) for her continuing interest in decision-tree-based networks. Without our countless discussions, this paper would probably never have been written. Thanks also to the anonymous reviewers for constructive criticism and useful suggestions.

References

[1] Andrew, C., Kubat, M., and Pfurtscheller, G. (1995). Trimming the Inputs of RBF Networks. *European Symposium on Artificial Neural Networks*, Brussels, Belgium

[2] Bioch, J.C., Carsouw, R., and Potharst, R. (1995). On the Mapping of Linear Classification Trees onto One-Hidden-Layer Neural Nets. *Proceeddings of the IEEE International Conference on Neural Neworks (ICNN95)*, Vol.4, 1739–1743

[3] Breiman, L., Friedman, J., Olshen, R., & Stone, C.J. (1984). *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA

[4] Brent, R.P. (1991). Fast Training Algorithms for Multilayer Neural Nets. *IEEE Transactions on Neural Networks*, 2, 346–354

[5] Broomhead, D.S. and Lowe, D. (1988). Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2, 321–355.

[6] Carter, C. and Catlett, J. (1987). Assessing Credit Card Applications Using Machine Learning, *IEEE Expert*, Fall issue, 71–79

[7] Chen, S., Cowan, C.F.N., and Grant, P.M. (1991). Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 2, 302–309.

[8] Cheng, Y.-H. and Lin, C.-S. (1994). A Learning Algorithm for Radial Basis Function Networks: with the Capability of Adding and Pruning Neurons. *Proceedings of the IEEE*, 797–801

[9] Cover, T.M. and Hart, P.E. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, IT-13, 21–27

[10] Dasarathy, B.V. (1991). *Nearest-Neighbor Classification Techniques.* IEEE Computer Society Press, Los Alomitos, CA

[11] Dietterich, T.G. (1996). Statistical Tests for Comparing Supervised Classification Learning Algorithms. Technical Report, Department of Computer Science, Oregon State University

[12] Duda R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York

[13] Fritzke, B. (1993). Supervised Learning with Growing Cell Structures. In *Advances in Neural Information Processing Systems 6*, J. Cowan, G. Tesauro, and J. Alspector (eds.), San Mateo, CA: Morgan Kaufmann, 255–262

[14] Fritzke, B. (1994). Fast Learning with Incremental RBF Networks. *Neural Processing Letters*, 1, 2–5

[15] Fu, L.M. (1994). *Neural Networks in Computer Intelligence.* Mc-Graw Hill, New York

[16] Goodman, R.M., Higgins, C.M., Miller, J.W. and Smyth,P. (1992). Rule-Based Neural Networks for Classification and Probability Estimation. *Neural Computation* 4:781–804

[17] Haykin, S. (1994). *Neural Networks, A Comprehensive Foundation.* Maxmillan College Publishing Company, New York

[18] Ivanova, I. and Kubat, M. (1995). Initialization of Neural Networks by Means of Decision Trees. *Knowledge-Based Systems*, 8, 333–344

[19] Kubat, M. (1996). Second Tier for Decision Trees. *Machine Learning: Proceedings of the 13th International Conference* (pp. 293–301), Morgan Kaufmann Publishers, San Francisco, CA

[20] Kubat, M. and Ivanova, I. (1995). Initialization of RBF Networks with Decision Trees. *Proceedings of the 5th Belgian-Dutch Conference on Machine Learning, BENELEARN'95* (pp. 61–70), Leuven, Belgium

[21] Kubat, M., Pfurtscheller, G., and Flotzinger D. (1994). AI-Based Approach to Automatic Sleep Classification. *Biological Cybernetics*, 79, 443–448

[22] Lee, Y. (1991). Handwritten Digit Recognition Using K-Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks. *Neural Computation*, 3, 440–449

[23] Lowe, D. (1989). Adaptive Radial Basis Function Nonlinearities and the Problem of Generalization. *1st International Conference on Artificial Neural Networks* (pp.171–175), London, UK

[24] Moody, J.E. and Darken, C.J. (1989). Fast Algorithms in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1, 282–294

[25] Murphy, P., & Aha, D. (1994). UCI Repository of Machine Learning Databases [machine-readable data repository]. Technical Report, University of California, Irvine

[26] Musavi, M.T., Ahmed, W., Chan, K.H., Faris, K.B., and Hummels, D.M. (1992). On the Training of Radial Basis Function Classifiers. *Neural Networks*, 5, 595–603.

[27] Opitz, D.W. and Shavlik, J.W. (1993). Heuristically Expanding Knowledge-Based Neural Networks. *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (pp. 1360–1360), Chamberey, France, Morgan Kaufmann

[28] Opitz, D.W. and Shavlik, J.W. (1994). Using Genetic Search to Refine Knowledge-Based Neural Networks. *Proceedings of the 11th International Conference on Machine Learning*, New Brunswick, NJ, Morgan Kaufmann

[29] Park, Y. (1990). A Mapping from Linear Tree Classifiers to Neural Net Classifiers. *Proceedings of IEEE International Conference on Neural Networks* (pp. 94–100), Orlando, Florida, Vol.I

[30] Poggio, T. and Girosi, F. (1990). Regularization Algorithms for Learning that are Equivalent to Multilayer Networks. *Science*, 247, 987–982

[31] Powell, M.D.J. (1988). Radial Basis Function Approximations to Polynomials. *Numerical Analysis 1987 Proceedings* (pp.223–241), Dendee, UK

[32] Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo

[33] Saha, A. and Keeler, J.D. (1990). Algorithms for Better Representation and Faster Learning in Radial Basis Function Networks. In *Advances in Neural Information Processing Systems 2* (D.S. Touretzky), San Mateo, CA, Morgan Kaufmann, 482–489

[34] Sanger, T.D. (1991). A Tree-Structured Adaptive Network for Function Approximation in High-Dimensional Spaces. *IEEE Transactions on Neural Networks*, 2:285–293

[35] Sethi, I.K. (1990). Entropy Nets: From Decision Trees to Neural Networks. *Proceedings of the IEEE*, 78, 1605–1613

[36] Towell, G.G., Shavlik, J.W., and Noordewier, M.O. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. *Proceedings of the 8th National Conference on Artificial Intelligence, AAAI'90*, 861–866

[37] Wettschereck D. and Dietterich T.G. (1992). Improving the Performance of Radial Basis Function Networks by Learning Center Locations. *Advances in Neural Information Processing Systems 4*, (J.E. Moody, S.J. Hanson, and R.P. Lippmann, eds.), San Mateo, CA: Morgan Kaufmann, 1133–1140

[38] Widrow, B. and Hoff, M.E. (1960). Adaptive Switching Circuits. *IRE WESCON Convention Record*, 96–104.