Combining Symbolic and Numeric Methods for Learning to Predict Temporal Series

M. Botta and A. Giordana
Dipartimento di Informatica, Università di Torino,
C.so Svizzera 185, 10149 Torino, Italy
e-mail:{botta,attilio}di.unito.it

Abstract

Radial Basis Function Networks are universal function approximators which can be easely constructed from rule sets learned by a symbolic learner. This paper proposes the use of a more expressive concept description language, based on first order logics, and of a learning system (FLASH) working in such an environment, in order to incorporate the feature selection phase into the learning process. The main advantage of the first order description language used by FLASH, is that it allows to define and manipulate a large number of features while keeping simple the description of instances. The method is tested on two non trivial problems of functional regression represented by the Mackey-Glass temporal series and by the control function for a manipulation robot. The obtained results demonstrate the high accuracy of the proposed method.

Introduction

After an initial period, in which the symbolic and the connectionist approaches to machine learning have been seen in counterposition, several researchers begun to work in the direction of an integrated approach that would take advantage of the specific benefits that each one of them offers. On one hand, the connectionist approach is easier to be implemented, is more suitable to deal with continuous domains and more robust with respect to noisy data. On the other hand, in the original proposal (Rumelhart & McClelland 1986), a neural network is basically a black box that makes it difficult to extract the knowledge it encodes. In this sense, the symbolic approach seems to be better because the learned knowledge is immediately readable and easy to integrate with possibly existing one.

However, in the last years it has been shown that, after all, a neural network is not so opaque as it was assumed to be and, then, it is possible to encode a priori knowledge into a neural net and, viceversa, to extract symbolic knowledge from it. For instance, Shawlik and Towell (Towell, Shavlik, & Noordwier 1990; Towell & Shavlik 1994) provided an algorithm, called KBANN, for transforming a propositional theory into a multilayer perceptron which can be trained by backpropagation. Again Shavlik and Towell (Towell & Shavlik 1993) and, later on, Craven and Shavlik (Craven & Shavlik 1994) presented algorithms for mapping back a multilayer perceptron into a propositional theory. Similar results have been obtained by Tresp (Tresp, Hollatz, & Ahmad 1993) and by Baroglio and others (Baroglio, Giordana, & Piola 1994; Baroglio *et al.* 1996) using another kind of neural network, called Radial Basis Function Networks (RBFNs), introduced by Moody and Darken (Moody & Darken 1988) and mathematically formalized by Poggio and Girosi (Poggio & Girosi 1990). In particular, RBFNs have a structure which immediately matches a flat propositional theory, so that algorithms for mapping symbolic knowledge into a network and viceversa become extremely simple. However, it is worth noting that RBFNs are members of a broader family of networks, including Fuzzy Controllers (Mamdani & Assilian 1975), Probabilistic Neural Networks (Specht 1988) and others, which also benefit from an immediate symbolic interpretation.

The experience with RBFNs, shows that their underlying architecture is excellent for combining symbolic and connectionist learning methods while maintaining the same accuracy available with other kinds of networks, such as multilayer perceptrons. In particular, symbolic algorithms can be advantageously exploited in order to construct a network layout, whereas numeric algorithms, such as the popular Δ-rule, can be used to finely tune numeric coefficients in the network.

This paper continues the research line of Baroglio and others (Baroglio, Giordana, & Piola 1994; Baroglio *et al.* 1996), where decision and regression trees were proposed for constructing RBFN's layouts, and explores a way of using symbolic algorithms designed for learning in first order logics, such as (Michalski 1983; Pazzani & Kibler 1992; Botta & Giordana 1993). The aim is to incorporate in the learning process large part of the feature selection phase, which, when propositional algorithms are used, takes place in a preprocessing step. As described in the following, this new approach has been applied using a modified version of SMART+ system (Botta & Giordana 1993), called FLASH, and obtained better results than any other method, in learning to predict temporal sequences such as the Mackey-Glass chaotic time series and the control function in a robot control problem. In general, the approach seems feasible and appropriate to regression tasks involving large databases as it frequently happens in data mining applications.

The paper is organized as follows: firstly, we introduce the RBFNs and discuss how they can be constructed using a symbolic algorithm and then refined by performing the error gradient descent. Then, the benefits of using a First Order logic learner are analyzed. Afterwards, we provide some details about the experimental algorithm FLASH and presents the results obtained on the two case studies mentioned above. Finally, some conclusions are drawn.

Learning RBFNs

Given a function $f(\vec{x})$ defined in a vectorial space **X**, a Radial Basis Function approximates $f(\vec{x})$ as a weighted sum:

$$\hat{f}(\vec{x}) = \frac{\sum_j w_j r_j(\vec{x})}{\sum_j r_j(\vec{x})} \tag{1}$$

where $r_j(\vec{x})$ is a n-dimensional radial function defined in $X^{(n)}$ and w_j is a real coefficient (weight)[1].

By requiring in (1) that the functions r_j be of the form:

$$r_j(\vec{x}) = \prod_i \mu_{ij}(x_i) \tag{2}$$

being μ_{ij} unidimensional bell-shaped functions, we obtain a Factorizable Radial Basis Function Network (F-RBFN) as described in (Poggio & Girosi 1990). Through the paper we will make use of RBFNs of this type because better suited to encode a set of propositional rules, as it will be explained in the following.

F-RBFNs can be represented as a 4-layer network (see Figure 1) with two hidden layers.

The neurons in the first hidden layer are feature detectors, each associated to a single unidimensional radial function $\mu_{ij}(x_i)$, and are connected to a single input only. Using Gauss's functions, as it is done through

[1]The canonical form for RBFNs is $\hat{f}(\vec{x}) = \sum_j w_j r_j(\vec{x})$. Nevertheless in this paper we will prefer form (1), also used by some authors, because produces more accurate approximations.

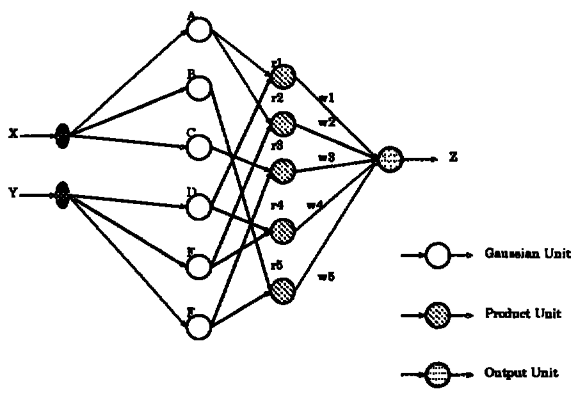


Figure 1: Reference F-RBFN architecture. The first hidden layer units have a unidimensional Gaussian activation function. The second hidden layer units compose the input values computing a product. An output unit performs the normalized weighted sum of the activation values received from product units.

this paper, each neuron μ_{ij} computes the output:

$$\mu_{ij}(x_i) = e^{-\left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right)^2} \tag{3}$$

where c_{ij} is the center of the Gauss's function along the x_i axis and the σ_{ij} is the width. The neurons in the second hidden layer simply compute a product and construct a multi-dimensional radial function according to expression (2). Finally, the output neuron combines the contributes of the composite functions from the second hidden layer computing expression (1).

The usual methods for learning from examples Radial Basis Function networks (Poggio & Girosi 1990) are based on a two step learning procedure. First a statistical clustering algorithm, such as *k-Means* is used to determine the centers and the amplitude of the activation functions. Then, the weights on the links to the output neuron are determined by computing the coefficients of the pseudo-inverse matrix (Poggio & Girosi 1990) or, alternatively, by performing the error gradient descent by means of the Δ-rule.

Using continuous radial functions that are derivable in the whole parameter space, it is immediate to apply the classical error gradient descent technique in order to finely tune not only the weights w_j but also the parameters c_{ij} and σ_{ij} in the first hidden layer units. Let E be the quadratic error evaluated on the learning set and let, moreover, λ_k indicate a generic parameter in the network, such as, a weight w on a link, the width σ or the center c of a Gauss activation function; all the necessary derivatives can be easely computed, and the learning rule takes the form:

$$\Delta\lambda_k = -\eta \frac{\partial E}{\partial \lambda_k} \quad (4)$$

The method based on the pseudo-inverse matrix is usually faster than the gradient descent. On the other hand, this last is sometime preferable, because simpler to implement and suitable for on-line learning.

An alternative method for constructing a RBFN is to use a symbolic learning algorithm (Baroglio *et al.* 1996; Tresp, Hollatz, & Ahmad 1993). This becomes particularly simple in the case of F-RBFNs. Approximating each unidimensional Gauss's function μ_{ij} by means of a segment A_{ij}, of length $2\sigma_{ij}$, and interpreting the product in the second layer hidden unit as a logical *AND*, a F-RBFN can be approximated by a set of propositional clauses of the type:

$$R_j = member(x_1, A_{1j}) \wedge member(x_2, A_{2j}) \wedge \ldots \wedge member(x_n, A_{nj}) \rightarrow w_j \quad (5)$$

Rules of this type can be easely learned using an algorithm for inducing decision trees, such as ID3 (Quinlan 1979; Sammut *et al.* 1992) or, better an algorithm for inducing regression trees, such as CART (Breiman *et al.* 1984). Owing to the symbolic approximation property, factorizable radial functions as well as the group of corresponding hidden units will be simply referred to as *rules*, in the following.

Approaching Functional Regression in First Order Logics

A regression problem can be easely mapped into a classification problem, by approximating the co-domain of the target function $f(\vec{x})$ with a set W of discrete values. Then, each value $w \in W$ can be considered as a class for which a discriminant definition has to be learned (Baroglio *et al.* 1996). As the learning problem is set in the framework of the propositional calculus, a learner of the same order is sufficient. Nevertheless, we will now propose a new approach based on a First Order Logics learner.

Usually, the input space $\mathbf{X}$ to a RBFN, is not immediately available from the data, but is obtained by applying a transformation $T(D)$, said *feature construction*, on a data space D. When a RBFN is learned using one of the methods mentioned in the previous section, the choice of a proper transformation $T(D)$ is made a priori either on the basis of a domain expert's knowledge, or using feature selection techniques. Nevertheless, this choice should take into account the way the learning algorithm uses the features. Since this is not usually feasible, a $T(D)$ chosen a priori is not optimal, in general. An alternative approach (the compiled approach), which can be followed when the network construction is made using an algorithm for learning decision or regression trees, is to construct a redundant set of features $\mathbf{X_R}$ in order to be sure that the good ones are included. Then, the learning algorithm will select the ones it finds more suitable and so $\mathbf{X}$ will implicitly be chosen out of $\mathbf{X_R}$. Nevertheless, this approach becomes impractical for learning from large learning sets.

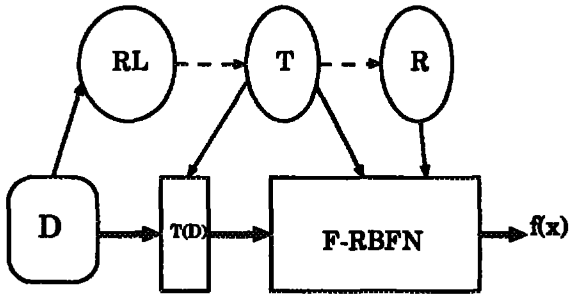


Figure 2: Sequence of steps for learning a F-RBFN. D = Learning set; RL = Rule Learning; T = Translation from rules to F-RBFN; R = Refinement by means of the Δ-Rule.

Let us consider a typical regression problem, frequently found in data mining, consisting in learning to predict a value $f(d_{t+\tau})$ associated to an event $d_{t+\tau}$ on the basis of the N events occurred in a time window going from event d_t back to d_{t-N+1}. Let, moreover, $\mathcal{F}$ be a set of functions assigned in order to construct the transformations $T(D)$ on the data space D. Every function $f_i(y_0, y_1, \ldots, y_k) \in \mathcal{F}$ can potentially define a new feature for each one of the N^k possible bindings between the arguments $y_0, y_1, \ldots, y_k$ and the N events in the temporal window. Then, the construction of all possible features, as required for an unrestricted compiled approach, would lead to an extremely complex learning set description (if $\mathcal{F}$ contains many operators and the temporal window ranges on many events).

The alternative approach, we propose here, is to introduce a mechanism that allows for the construction of features on demand of the learning algorithm. This can be realized by adopting a learning algorithm for learning relations extended with a mechanism similar to the one used in databases for supporting *computable constraints*. In particular, we will introduce a concept description language L in which predicate semantics is defined by means of a boolean function built on top of a feature constructor in the set $\mathcal{F}$. For instance, the predicate $older(x_0, x_1)$ stating that the event bound to x_0 occurred before the one bound to x_1, can be defined as:

$$older(x_0, x_1) :: \delta(x_0, x_1) > 0 \quad (6)$$

where $\delta(x_0,x_1) \in \mathcal{F}$ and computes the time interval between x_0 and x_1. When the learning algorithm uses predicate $older(x_0,x_1)$, its truth will be evaluated on the items bound to its variables. Therefore, computable predicates do not change the basic learning strategy of the algorithm but only delay the construction of the corresponding feature until it is used.

A first benefit is that memory requirements for the learning set is strongly reduced. However, this is balanced by a potential increase in the computational complexity, because a feature needs to be constructed again every time the learning algorithm tries to use a predicate depending on it. Fortunately, this does not happen in general, because only a small subset of the possible bindings is explored by a learning algorithm so that the replicated feature constructions are widely compensated by the strong reduction of the instantiations actually explored. It is worth noting that computable predicates have already been used in ML-smart (Bergadano, Giordana, & Saitta 1988) and in Smart+ (Botta & Giordana 1993).

Then, a rule set, learned in this way, has to be transformed into a feature constructor set $T(D)$ and a F-RBFN, which will be refined by performing the error gradient descent (see Figure 2). The set $T(D)$ can be easely obtained by collecting all the instances of the functions defining computable predicates occurring in the rule set. However, this translation can only be done if, for each learned rule $\varphi \rightarrow w$, the variables occurring in φ are bound to events occurring in the same position in every temporal window where φ is verified. As it will be described in the next section this condition can be enforced by adopting a specific control strategy in the learning algorithm.

The Symbolic Learning Algorithm

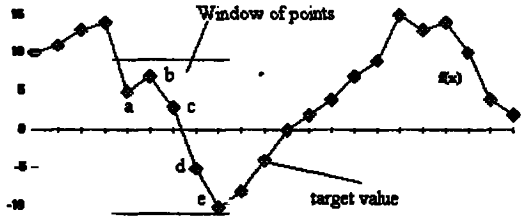
The FLASH (Fuzzy Logic Approximation SHaper) system, used for learning the layout of a RBFN, is a direct descendent of Smart+ (Botta & Giordana 1993) from which it inherits the general architecture. Nevertheless, several fundamental novelties have been introduced in order to handle the specific task we are addressing. FLASH shares with Smart+, as well as other learning relations algorithms, a general-to-specific search strategy for inductive learning, but it has a richer set of specializing operators, and it uses search strategies specific for regression tasks. FLASH uses two main formalisms for representing its knowledge: a relational database is used to store the training examples and partial hypothesis extensions; a first order logic language is used to represent the background and the acquired knowledge. A training example is represented as a set of distinct elements, called ***objects***, each described by a set of properties, called ***basic attributes*** (Botta, Giordana, & Saitta 1992). Each object is typed and can be described by its own set of basic attributes. The main advantage of this representation scheme is its flexibility: the set of basic attributes can be easily extended; as well, the number of component objects can change from instance to instance and it naturally reduces to attribute-valued representation when all instances are made of only one object. Furthermore, this representation scheme is suited to a more flexible definition of the concept description language. In a regression problem, this representation can be exploited in the following way: let us consider the function plotted in Figure 3. An instance presented to FLASH is generated by taking into account a window of N temporal events and by defining an object for each one of them, described by two basic attributes: the relative (to the window) position of the event and the value of the function for that event. The instance is then labeled by the value that the function takes on in the event we have to predict. FLASH concept description language L is a first order Horn clause language, in which a clause body is built up by a conjunction of predicates in a set P. For each predicate $p \in P$, a semantic function S_p, is defined to instruct the system how to evaluate the truth of p on the learning events, by correlating terms in the predicate p to objects and attribute values in the instances. In general, S_p evaluates the truth μ_p of a numerical expression f_p as in the following example:

$$S_p(x_1,\ldots,x_n) :: member(f_p(x_1,\ldots,x_n),I_p) \qquad (7)$$

being I_p an interval (segment) defining the range on the feature f_p where the predicate p is considered true. The numerical expression f_p can be any C language expression combining attribute values, C library mathematical functions and user-defined functions. This intensional definition of semantic functions usually results in a more concise representation, leads to a computationally faster implementation and allows ***predicate schema***, that depend on run-time numerical constants, to be defined. By instantiating these constants with values in a specified range, FLASH can select the best instantiations in the context of the current hypothesis. For example, the semantic function S_δ, for a predicate schema $\delta(x_1,x_2,k_1,k_2)$ can be defined as follows:

$$\begin{array}{ll} \multicolumn{2}{c}{S_\delta(x_1,x_2,k_1,k_2) :: member(f_\delta, I_\delta)} \\ f_\delta :: |h(x_2)-h(x_1)| & I_\delta = (k_1-k_2, k_1+k_2) \\ k_1 = [0.0 \div 1.0, 0.05] & k_2 = [0.1 \div 0.2, 0.05] \end{array} \qquad (8)$$

where I_δ is a segment with center k_1, that can assume the values 0.0, 0.05, 0.1, ..., 1.0, and width k_2, that



Id	Position	Val
a	0	4.8
b	1	6.3
c	2	3.2
d	3	-5.1
e	4	-9.7

Figure 3: Example of function to be approximated.

can assume the values 0.1, 0.15, 0.2, and $h(x)$ is a function that returns the value of attribute h of objects x. Each numerical expression f_p can be thought of as a descriptive feature of the instances.

FLASH learning duty also comprises the selection of the suitable features for the task at hand, and, as a result, the learned knowledge will specify not only a set of implication rules, but also the set of features that are to be used to represent the instances. We will return to this point later on. FLASH can use background information in the form of necessity constraints on the applicability of predicates and on the acceptability of classification rules. The former are statements of the type:

$$p(x_1, x_2, \ldots, x_n) \textit{ needs } \varphi(y_1, y_2, \ldots, y_m) \qquad (9)$$

where $\varphi(y_1, y_2, \ldots, y_m)$ is a formula belonging to the concept description language. Expression (9) states that $p(x_1, x_2, \ldots, x_n)$ can be true only if $\varphi(y_1, y_2, \ldots, y_m)$ has been proven true. The induction space can be arbitrarily constrained by properly defining a set of necessity constraints. For instance, the following constraint states that predicate $\delta(x_1, x_2)$ can be added to a formula φ only if predicate $height(x_1)$ has already been applied (is part of the formula):

$$CR1(\varphi) :: \delta(x_1, x_2) \textit{ needs } height(x_1) \qquad (10)$$

These necessity constraints are used by FLASH to collect applicable predicates at each step of the search process. The second type of necessity constraints affects the structure of the classification rules the system should generate:

$$\psi(x_1, x_2, \ldots, x_n) \textit{ needs } \varphi(y_1, y_2, \ldots, y_m) \qquad (11)$$

where ψ and φ are formulas of the concept description language L. Expression (11) means that formula ψ can be accepted as a classification rule only if it contains φ. For instance, according to control rule CR2, a formula ψ can be accepted as a classification rule only if it contains predicates $height(x_1)$ and $\delta(x_1, x_2)$:

$$CR2(\psi) :: \psi \textit{ needs } height(x_1) \wedge \delta(x_1, x_2) \qquad (12)$$

By specifying which predicates must be present in a formula, it is possible to achieve a complete characterization of the input region associated to the value represented by the target class. Control rules of this type are applied at each step of the learning process to check if a formula needs further specialization (independent of its classification accuracy) or can be considered as a classification rule (evaluated for classification accuracy). One specific necessary constraint has been defined for the approximation task: as FLASH learns first order rules that are to be translated into a propositional network, classification rules must have a unique model on the data, in such a way that a direct mapping is feasible. This is assured by the following constraint:

$$\begin{aligned} CR(\psi) :: \psi(x_1, \ldots, x_n) \textit{ needs } isobj(x_1, k_1) \\ \wedge \ldots \wedge isobj(x_n, k_n) \end{aligned} \qquad (13)$$

where $isobj(x, k)$ is a predicate scheme which binds a variable x to a specific object in a sequence:

$$S_{isobj} :: position(x) = k \qquad (14)$$

FLASH is configured to learn a concept at a time by using a best first search coupled with a covering strategy: at each step, the most promising formula is selected for specialization and applicable predicate schema are collected according to the necessary constraints; then, for every predicate schema, numerical parameters are instantiated, the predicate is tested on the data and scored according to an information gain function. The best $n \geq 1$ predicate schema so obtained are used to expand a search tree. The found formulas are tested for classification accuracy and against necessary constraints; those that are accepted as classification rules are saved and instances are declared covered, so that FLASH can focus its attention on the remaining training examples.

1. Start with an "or" tree T containing the formula "True", or previously initialized with a set of formulas predicted by a theory (or by an expert).
2. Insert in the list $OPEN$ all the leaves currently existing in T.

3. Select from $OPEN$ the formula ϕ having the best ranking $\nu(\phi)$.
4. Determine the set P_A of applicable predicates to ϕ according to the predicate constraints.
5. For each predicate $p(\vec{x}, \vec{K}) \in P_A$ find the assignment $\vec{k}$ for the parameters $\vec{K}$ in p, which maximizes $\nu(\phi \wedge p(\vec{x}, \vec{k}))$, and if $\nu > 0$ put $\phi \wedge p(\vec{x}, \vec{k})$ in the list $TODO$ sorted according to decreasing values of ν.
6. Then, expand T with the $n \geq 1$ first formulas in $TODO$ and put them in $OPEN$.
7. If now $OPEN$ contains some classification rule ψ (according to an assigned criterion), go to step 1; otherwise go to step 3.

This cycle is repeated until all instances have been covered, or there is no possibility to find other classification rules (e.g., because allocated computational resources have been exhausted). The function ν is a measure of the quality of an inductive hypothesis ϕ, and has been chosen to be the function

$$if(v \geq v_0)\ then\ \nu = v \cdot u\ else\ \nu = 0 \qquad (15)$$

being $0 \leq v \leq 1$ and $0 \leq u \leq 1$ the proportion of positive examples covered by ϕ and the correctness of ϕ, respectively. The condition $v \leq v_0$ prevents the generation of too specific classification rules, in order to control the overfitting.

Inductive hypotheses are accepted as classification rules also if they are slightly inconsistent. In particular, a classification rule ϕ is allowed to cover some examples of the two classes corresponding to the two values in W adjacents to the target class, provided that its correctness doesn't drop below an assigned threshold u_0. The tuning of u_0 and v_0 is left to the user.

FLASH learns rules of the following type:

$$p_1(\vec{x}, k_{11}, k_{12}) \wedge \ldots \wedge p_n(\vec{x}, k_{n1}, k_{n2}) \rightarrow w \qquad (16)$$

in which all parameters k_{ij}s have been instantiated, there are as many *isobj* predicates as variables in the rule, and w is one of the values of the function to be approximated belonging to the set W. Each rule is translated into a second hidden layer neuron connected to the output neuron with a link having weight equal to w. A predicate $p_i(x, k_{i1}, k_{i2})$ in the antecedent of a rule can be translated into a first hidden layer neuron with gaussian activation function, derived by the segment defined by k_{i1} and k_{i2} according to the transformation described in the previous sections, and linked to the corresponding second hidden layer neuron and to an input neuron.

Evaluation of the Method

The described methodology has been tested on two approximation problems, bearing quite different characteristics: the first testbed is a classical approximation problem of a continuous function generated by a well known chaotic series (Mackey-Glass), which has been addressed using other techniques and allows comparisons to be made, whereas the second is concerned with the approximation of a control function in a robotics environment.

The Makey-Glass chaotic series is generated by the following differential equation:

$$\dot{x} = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1x(t) \qquad (17)$$

with $x(t < 0) = 0$, $x(0) = 1.2$ and $\tau = 17$. The classical learning problem is defined as follows: given the series values at instants t, t-6, t-12, t-18, forecast the series value at t+84. Table 1 shows some of the best results presented in the literature, according to the NDEI (Non-Dimensional Error Index) error index, that is the ratio between the root average square error and the standard deviation. MLP refers to the result obtained by a multi-layer perceptron; ANFIS (Jang 1993) is a fuzzy inference system implemented as an adaptive neural network, whose structure has been manually defined. FC stands for a fuzzy controller built from symbolic knowledge learned by CART (Breiman *et al.* 1984) and Smart+ (on a propositional definition of the problem) and refined for 5000 epochs. $RBFN_{Smart+}$ is the result obtained by using a RBFN on the same symbolic knowledge used by FC, refined for 5000 epochs.

Table 1: Comparison between the best results on the Mackey-Glass learning problem.

Method	Learning Instances	NDEI
MLP	500	0.05
$ANFIS$	500	0.036
$FC_{CART} + R$	1000	0.037
$FC_{Smart+} + R$	1000	0.032
$RBFN_{Smart+} + R$	1000	0.032

As it can be noted, the Radial Basis Function Networks obtained a result that compares favorably to the others, being only better at the 6th decimal digit than the one obtained with FC. In order to test FLASH ability to automatically select which input features are relevant to the problem, a different formulation of the above problem has been used in two new experiments: instead of using only the four above mentioned values as input features, all the values of the series corresponding to instants from t-18 to t have been considered. In particular, in the first experiment, we used a propositional definition of the problem, by defining the learning problem in terms of 37 numerical attributes, 19 of which refers to the values at instants t-18, ...t, and the remaining 18 refers to the differences between the values of two consecutive instants. The concept description language consists of 37 predicate schemas, whose semantics functions compute the membership of an attribute value in an interval. In the second experiment, we used a first order representation of the instances: each instance consists of 19 objects described by two attributes, the position inside the window and the corresponding series value at that point. In this second case, we defined only two predicate schemas, which refer to the series value and the difference between the values at instants t-x, t-y, $x \neq y \in [0,18]$, respectively. In all experiments, we used the same learning set, consisting of 1000 instances, and test set, of 500 instances, as the ones used in the reported experiments, labeled according to 12 classes of values. For the sake of comparison, CART has been run on the propositional definition of the problem and the learned knowledge translated in RBFN and refined for 5000 epochs. The results on the test set are reported in Table 2. As can be noted, the approximator constructed with the described methodology obtained better performances than any other method analyzed. What is significant in this experiment is not only the smallest error index obtained, but the fact that the first order representation language used allowed FLASH to analyze a large number of possible features (190) and to select the most relevant ones (46), which resulted in a very good network layout.

Table 2: Results obtained on the new problem definition.

Method	Learning Instances	NDEI
$RBFN_{CART} + R$	1000	0.049
$RBFN_{FLASHprop} + R$	1000	0.046
$RBFN_{FLASH} + R$	1000	0.024

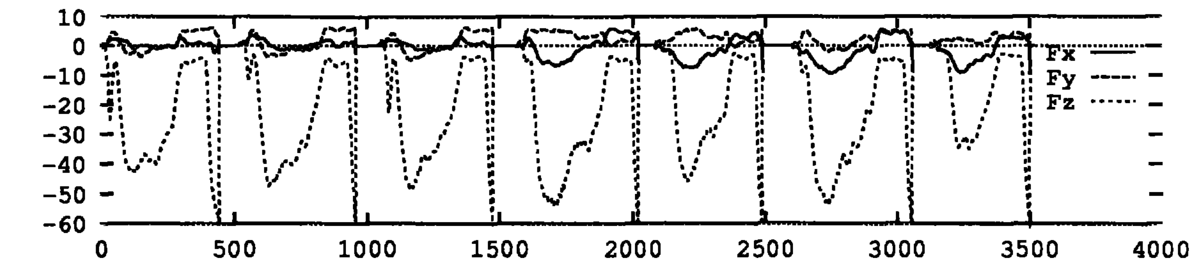
In the second testbed, we considered an approximation problem in a robotics environment: the goal was to learn a control function of a robot arm performing the peg-into-hole task, in such a way that it can be reproduced at a higher speed. Both the peg and the hole are of circular shape. Input values are forces and torques along the three directions, axial, radial and vertical, measured by sensors placed on the robot arm (Figure 6 reports plots of these values), whereas output values are the corresponding velocity in the three directions the robot arm should move at (Figure 7 reports a plot of the output values).

For every output velocity, a learning problem has been defined according to the methodology previously described: in particular, each instance presented to FLASH consists of 8 objects, each described by 6 attributes (three force values and three torque values) and corresponding to sensory readings at successive time points t-7, t-6, ..., t, and it is labeled according to the velocity value at time point t+1 (14 classes of values). Furthermore, we defined three sets of predicate schemas (each consisting of 6 predicate schemas) in order to account for the fact that controllers used in this task are usually Integral-Differential controllers: one predicate schema refers to the values of an attribute at a given time point; a second predicate schema refers to the difference between attribute values at time points t and t-x, $\forall x \in [1,7]$, that should capture properties of the derivative of the signal; a third one refers to the sum of attribute values between any time point t-x, $x \in [1,7]$, and t, that should capture properties of the integral of the signals.

Table 3: Statistical comparative results for V_z, the most critical output control signal for the peg-into-hole task; all numbers are percentages (%).

Method	Epochs	V_z $Err_{avg} \pm Std.dev.$
CART (alone)	0	1.38 ± 5.01
FC_{CART}	0	11.84 ± 20.92
$FC_{CART} + R$	5000	0.45 ± 2.75
FC_{SMART+}	0	2.12 ± 4.94
$FC_{SMART+} + R$	5000	1.04 ± 4.18
$RBFN$	10000	3.3 ± 3.83
$TDNN$	10000	2.7 ± 2.30
MLP	10000	4.5 ± 3.84
$FLASH$ (alone)	0	3.35 ± 5.26
$RBFN_{FLASH}$	0	10.21 ± 22.15
$RBFN_{FLASH} + R$	3000	0.43 ± 2.74

As shown in Table 3, several learning algorithm have been compared, by using the same learning (four insertion sequences) and test sets (three insertion sequences). In particular, we reported results obtained by using CART (Breiman *et al.* 1984) and Smart+ (Botta & Giordana 1993) to initialize a fuzzy controller (FC), implemented according to the schema reported in Figure 1, on a propositional problem definition: each instance is described by 24 attributes, corresponding to force and torque values measured in the last four time points. The number of neurons used in the fuzzy controller varies from 84 to 168 in the first hidden layer with unidimensional gaussian activation function, and from 46 to 85 in the second hidden layer. RBFN has been initialized by using a clustering algo-



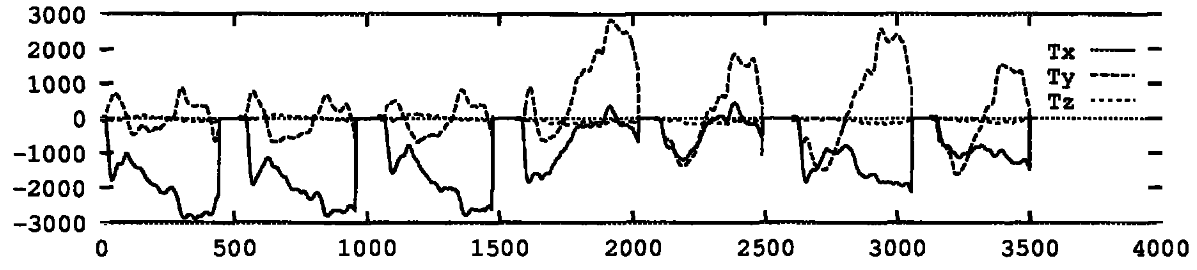


Figure 4: Input signals for seven insertion sequences: Fx, Fy e Fz correspond to the force sensors along the three coordinates while Tx, Ty e Tz correspond to the torque sensors. The values have been sampled at 10ms.

rithm (Musavi *et al.* 1992) that built a network with a number of neurons in the hidden layer that varies from 30 to 60, but with a 6-dimensional gaussian activation function. Moreover, we reported results obtained by other researchers (Kaiser 1994) with two other kind of neural networks, TDNN (time-delay neural network) and MLP (multi-layer perceptron). The layout of these networks has been manually chosen by performing several experiments with different configuration, by varying the number of neurons in the hidden layer and the number of delay units (for TDNN). The reported values correspond to the configuration that obtained the best results. Finally, the last three rows contain results obtained by the described methodology at each step: by using FLASH classifier, by using RBFN without refinement and after refinement. The number of neurons in the first hidden layer varies from 31 to 88 whereas, in the second hidden layer, it varies from 23 to 36. The first point to deserve attention is that the presented methodology builds up networks that need less training to reach equivalent or even better performances. This is mainly due to the compactness of the network produced, that, if compared to the fuzzy controller case, needs about 50% neurons, so having much less parameters to be tuned.

Figure 6 compares the obtained approximation with the original curve of velocity Vz (the most critical for a good insertion operation) in one of the experiments performed. After refinement (Figure 6c), an almost complete identity in the resulting behavior can be observed.

Conclusion

A new method based on an algorithm for learning relations in First Order Logics (FOL) has been presented in order to learn RBFN layouts from examples. The experimental results presented above show that the method can construct very accurate approximators in regression problems.

However, beyond the specific algorithms, the novelty presented in this paper is in the way FOL has been used: not as a target, but as an intermediate step in the construction of a knowledge base which still belongs to the propositional logics. In particular, the ability at exploring many alternative bindings offered by a FOL learner has been exploited in order to search the feature space usable for constructing a RBFN.

In our opinion, this is a new way of looking at FOL which up to now has not been considered and that deserves attention.

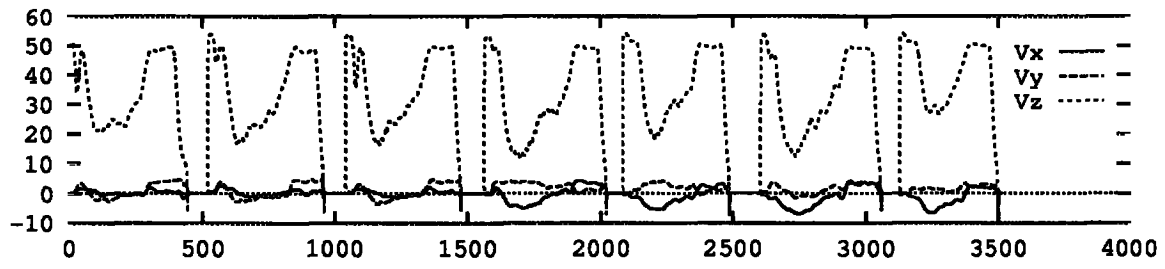


Figure 5: Velocity signals computed by PID-controller from the signals reported in the previous pictures.

References

Baroglio, C.; Giordana, A.; Kaiser, M.; Nuttin, M.; and Piola, R. 1996. Learning controllers for industrial robots. *Machine Learning.*

Baroglio, C.; Giordana, A.; and Piola, R. 1994. Learning control functions for industrial robots. In *Proceedings of the "Workshop on Robotics", Machine Learning Conference.*

Bergadano, F.; Giordana, A.; and Saitta, L. 1988. Learning concepts in noisy environment. *IEEE Transaction on Pattern Analysis ans Machine Intelligence* 555–578.

Botta, M., and Giordana, A. 1993. SMART+: A multi-strategy learning tool. In *IJCAI-93, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 2.

Botta, M.; Giordana, A.; and Saitta, L. 1992. Learning in uncertain environments. In Yager, R., and Zadeh, L., eds., *An Introduction to Fuzzy Logic Applications in Intelligent Systems.* Kluver Academic Publishers.

Breiman, L.; Friedman, J.; Ohlsen, R.; and Stone, C. 1984. *Classification And Regression Trees.* Pacific Grove, CA: Wadsworth & Brooks.

Craven, M., and Shavlik, J. 1994. Using sampling and queries to extract rules from trained neural networks. In *Proceedings of the Eleventh International Conference on Machine Learning*, 37–45.

Jang, J. 1993. ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Transactions on Systems, Men and Cybernetics* SMC-23(3):665–687.

Kaiser, M. 1994. Time-delay neural networks for control. In *Proceedings of the Symposium on Robot Control '94 (SYROCO '94).*

Mamdani, E., and Assilian, S. 1975. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies* 7(1):1–13.

Michalski, R. 1983. A theory and methodology of inductive learning. In Michalski, R.; Carbonell, J.; and Mitchell, T., eds., *Machine Learning: An Artificial Intelligence Approach*, 83–134. Los Altos, CA: Morgan Kaufmann.

Moody, J., and Darken, C. 1988. Learning with localized receptive fields. In Sejnowski, T.; Touretzky, D.; and Hinton, G., eds., *Connectionist Models Summer School.*

Musavi, M.; Ahmed, W.; Chan, K.; Faris, K.; and Hummels, D. 1992. On the training of radial basis function classifiers. *Neural Networks* 5:595–603.

Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine Learning* 9:57–94.

Poggio, T., and Girosi, F. 1990. Networks for approximation and learning. *Proceedings of the IEEE* 78(9):1481–1497.

Quinlan, J. 1979. Induction over large data bases. Technical Report HPP-79-14, Heuristic Programming Project, Stanford University.

Rumelhart, D. E., and McClelland, J. L. 1986. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Parts I & II.* Cambridge, Massachusetts: MIT Press.

Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In Sleeman, D., and Edwards, P., eds., *Machine Learning - Proceedings of the Ninth International Workshop (ML92)*, 385–393. Morgan Kaufmann.

Specht, D. 1988. Probabilistic neural networks for classification mapping, or associative memory. In

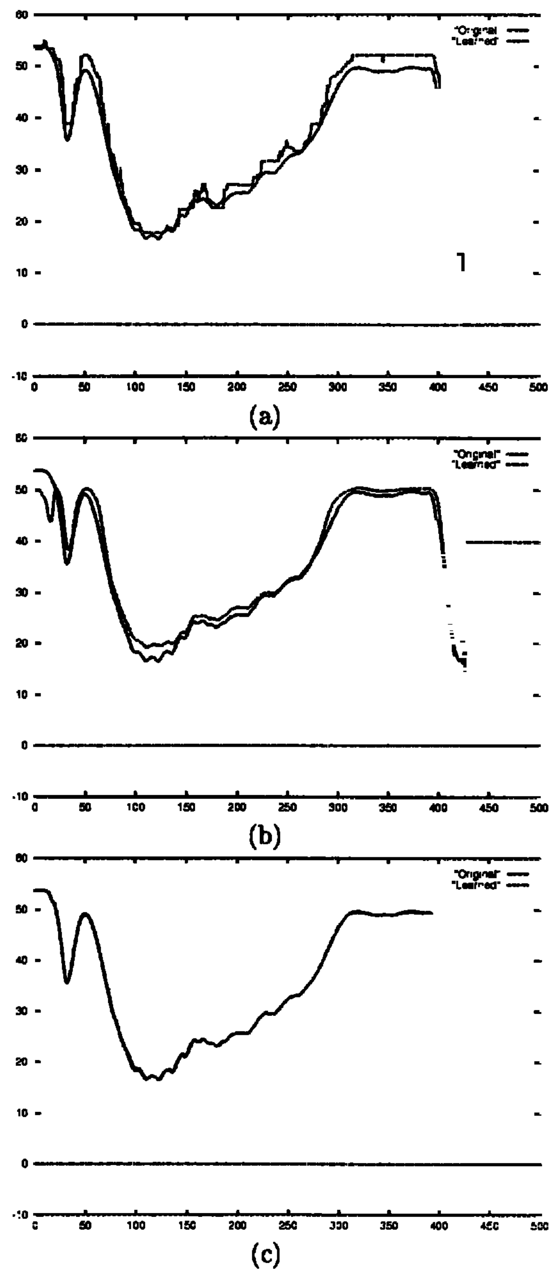


Figure 6: Approximation of the control function V_z obtained using: (a) FLASH only; (b) RBFN before refinement; (c) RBFN after refinement.

IEEE International Conference on Neural Networks, volume 1, 525–532.

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

Towell, G., and Shavlik, J. 1994. Knowledge based artificial neural networks. *Artificial Intelligence* 70(4):119–166.

Towell, G.; Shavlik, J.; and Noordwier, M. 1990. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the 8th National Conference on Artificial Intelligence AAAI'90*, 861–866.

Tresp, V.; Hollatz, J.; and Ahmad, S. 1993. Network structuring and training using rule-based knowledge. In *Advances in Neural Information Processing Systems 5 (NIPS-5)*.