Evolution of Functional Link Networks
A. Sierra, J. A. Macías, and F. Corbacho

Abstract—This paper addresses the genetic design of functional link networks (FLNs). FLNs are high-order perceptrons (HOPs) without hidden units. Despite their linear nature, FLNs can capture nonlinear input–output relationships, provided that they are fed with an adequate set of polynomial inputs, which are constructed out of the original input attributes. Given this set, it turns out to be very simple to train the network, as compared with a multilayer perceptron (MLP). However, finding the optimal subset of units is a difficult problem because of its nongradient nature and the large number of available units, especially for high degrees. Some constructive growing methods have been proposed to address this issue. Here, we rely on the global search capabilities of a genetic algorithm to scan the space of subsets of polynomial units, which is plagued by a host of local minima. By contrast, the quadratic error function of each individual FLN has only one minimum, which makes fitness evaluation practically noiseless. We find that surprisingly simple FLNs compare favorably with other more complex architectures derived by means of constructive and evolutionary algorithms on some UCI benchmark data sets. Moreover, our models are especially amenable to interpretation, due to an incremental approach that penalizes complex architectures and starts with a pool of single-attribute FLNs.

Index Terms—Evolutionary neural networks, feature subset selection, functional link networks, polynomial regression.

I. INTRODUCTION

Neural networks have become one of the most popular pattern recognition systems [1]. This popularity stems from a number of reasons, including: their capability to capture nonlinear relationships between their input and output patterns; their biological plausibility, as compared to conventional statistical models; their potential for parallel implementations; their celebrated robustness and graceful degradation, etc. In fact, a multilayer perceptron with a suitable architecture is capable of approximating virtually any function of interest [2]. This does not mean that finding such a network is easy. On the contrary, problems, such as local minima trapping, saturation, weight interference, initial weight dependence, and overfitting, make neural network training difficult. Moreover, most neural learning methods, being based on gradient descent, cannot search the nondifferentiable landscape of multilayer architectures. This is a key point, since it can be proved that if a network is allowed to adapt its architecture, it can solve any learnable problem [3] in polynomial time.

An easy way to avoid these architectural problems consists in removing the hidden layers. This may sound a little harsh at first, since it is due to them that nonlinear input–output relationships can be captured. Encouragingly enough, the removing procedure can be executed without giving up nonlinearity, provided that the input layer is endowed with additional higher order units [4], also known as sigma-pi units [5]. This is the idea behind functional link networks [6] and higher order networks (HONs) [7]. FLNs are HONs without hidden units. The extra units are constructed as functions of the original attributes, such as polynomials, up to a certain degree. Even in the absence of hidden units, the resulting networks are capable of approximating nonlinear mappings. Of course, synthetic problems can be designed, for which high-degree added features would have to be considered, in order for a simple linear network to be able to solve the problem. For example, an FLN would need a 5-dimensional polynomial term, \( x_1^2x_2^3x_3^4x_5^5 \), to solve the XOR problem in five dimensions. This is easy to check if a bipolar XOR coding is used. In such a case, the above polynomial term evaluates to \(-1\) whenever it receives an odd number of \(-1\) features and it evaluates to 1 otherwise, exactly matching what the XOR mapping requires. However, the main difficulty in practice is not usually an intricate sort of nonlinearity but the limitation of the available attributes to discriminate between classes, together with possible noise. Most of the time, a simple linear model can perform well, as long as good attributes are selected, possibly rejecting some other ones. For example, the classical iris problem can be solved (with only 4 misclassified flowers), due to a single attribute constructed by multiplying petal length by petal width, i.e., petal area [8].

An FLN is much more modest than a MLP, since it does not guarantee universal approximation (a variation known as random vector FLN does [9]). The very optimal solution, promised by an adequate MLP architecture (most times, difficult to be discovered), may be completely unavailable for a specific FLN. Notwithstanding, due to the absence of local minima, it happens to be very easy to reach the global minimum associated with each specific FLN architecture. Therefore, solving a classification problem is reduced to finding the FLN with a minimum as close to the optimal one as possible, much in the same way that hidden units of an MLP do while searching for the correct hidden projection before the final classification [10]. For FLNs, the computational cost moves from the hidden layer to the selection of a suitable input layer.

The lack of universal approximation has deterred interest in FLNs. The discrete nature and often vast dimension of the polynomial search space have not helped either. Only a few applications can be found in the literature: the structure of HONs allows translation of image invariances into weight constraints, which greatly reduces the number of these weights and makes the problem tractable [11]. In general, for a \(d\)-dimensional classification problem there are \((d+\#d)!/(\#d)!\) possible polynomials up to degree \(r\). For most real-life problems, this is too big a
number, even for degree 2, which obviously precludes exhaustive search for subsets of polynomial input terms. However, we can still resort to constructive and pruning algorithms [12] in order to address this problem. In fact, some specific learning machines can be found for the construction of HOPs [13]–[15].

In this paper, rather than following a purely constructive approach, we address the search for architectures from an evolutionary point of view [16]–[18]. Following the success of evolutionary MLPs [19] we introduce EFLN, a new algorithm for the evolution of functional link networks. EFLN makes use of a standard genetic algorithm (GA) [16] to evolve near-minimal linear architectures, by means of a complexity penalizing fitness function and an initial pool of single-attribute networks. The GAs task consists in figuring out whether a connection should be considered at all. The FLN training algorithm later fine-tunes the value of this connection. The mapping from genotypes to networks is here one-to-one, due to the absence of local minima in the error landscape of each FLN. This fact, together with the reduced training times, turn EFLN into a viable alternative to the evolution of MLPs.

The structure of the paper is as follows: Section II reviews previous work on combinations of neural networks with evolutionary algorithms, placing ours in context. Section III describes EFLNs implementation details and Section IV discusses the experiments conducted on a set of UCI databases. The discussion and conclusions can be found in Sections V and VI, respectively.

II. EFLN IN CONTEXT

Practically every kind of neural network has been approached from an evolutionary perspective: MLPs [20], modular neural networks [21], recurrent neural networks [22], learning vector quantization networks [23], etc. In general, an arbitrary neural network is determined by its architecture, its weights, its activation functions, and the learning algorithm used for training. Work on evolutionary neural networks has been classified, according to which of the above mentioned elements actually are codified in the chromosomes evolved by the evolutionary algorithm [19].

One of the simplest approaches consists in the evolution of chromosomes coding the weights of a fixed multilayer architecture. Some successful applications have been developed following this approach: a GA with binary coded weights is shown to outperform significantly backpropagation in the binary classification of underwater sonic lofargrams [24]. This approach, known as the direct coding scheme, scales badly for complex architectures since the number of weights augments very rapidly with the number of hidden units. Alternative nonbinary coding schemes, such as real [25] and grammatical encoding [26], can alleviate this problem. However, scalability is not the only drawback of directly coded fixed architectures. The main advantage of evolved neural networks, as compared with traditional ones, is their ability to learn in spaces where no gradient information is available. This is the case for the landscape of architectures. Nowadays, any system not addressing this point falls short of providing a solid alternative to backpropagation. Some degree of architectural encoding will not only scale better than direct codifications, but will address the search for suitable architectures.

This second means of evolution is called the indirect coding scheme, which, in its extreme form, could be defined as the evolutionary approach where only architectural parameters and not weights are coded into chromosomes. For example, connection matrices coding feedforward single hidden layer networks trained by backpropagation have been used successfully [27]. It is important to notice that here backpropagation is not substituted by another algorithm, but combined with it. Many other such combinations have been tried. For example, GAs have also been used for weight initialization. Backpropagation is used later for fine-tuning the initial weights.

The indirect coding scheme gets in trouble when assigning fitness to chromosomes since network performance is highly dependent on the initial weight configuration. Much care has to be taken to avoid this noise. The answer may be in schemes in between the two extremes (direct and indirect) described above. It has been claimed that evolving behaviors may be the key to good evolutionary designs [20]. It might even be beneficial not to converge to a best individual, as happens in some control problems where maintaining adaptability is crucial. For example, SANE [28] coevolves a neuron-based population that cooperates to form a neural network.

Another important issue is the evolution of learning rules. The evolution of backpropagation parameters [29] can be thought of as a first attempt to evolve learning algorithms. A more ambitious experiment [30] evolves successful learning rules for single-layer networks applied to a variety of linearly separable tasks. Econs constitute another effort oriented to the understanding of the evolution of learning [31].

As reviewed above, most approaches to the genetic design of neural architectures search the space of one-hidden-layer networks. Evolutionary techniques have proved to search efficiently this kind of huge, nondifferentiable, noisy, deceptive, and full of local minima spaces [27], that defeat more conventional search techniques [32]. Our system, on the contrary, gives up hidden units from the very beginning. The computational burden moves from the hidden layer to the input layer. The GA task consists in finding the right subset of polynomial input attributes, that leads to an effective separation of the given pattern classes. Let us discuss this point further.

The universal approximation capabilities of multilayer perceptrons can be understood as follows [10]. Take a perceptron with $d$ input units and $h$ sigmoidal hidden units as an example. The output function calculated by this neural network is

$$y_k = \sum_{j=1}^{h} w_{jk}^{(2)} \phi_j(x)$$  

(1)

where $w_{jk}^{(2)}$ is the weight connecting hidden unit $j$ with output unit $k$. The hidden layer units calculate a projection of the original input space into an intermediate one

$$(x_1, \cdots, x_i, \cdots, x_d) \rightarrow (\phi_1(x), \cdots, \phi_j(x), \cdots, \phi_h(x))$$  

(2)
by means of the first-layer weights $u^{(1)}$

$$
\phi_j(x) = \sigma \left( \sum_{i=1}^{d} u_{ij}^{(1)} x_i \right),
$$

(3)

In this hidden space, linear discrimination, to be carried out by the output weights $u^{(2)}$, becomes easier than in the original input space. By contrast, the linear networks used in EFLN take the hidden units to the input layer and work with a single layer of weights $u$. The new output function is

$$
y_k = \sum_{i=1}^{d'} w_{ik} \phi_i(x)
$$

(4)

where $d'$ is the dimension of the new input space

$$
(\phi_1(x), \cdots, \phi_{d'}(x)).
$$

(5)

The new units $\phi_i(x)$, instead of being learnable arbitrary functions of the original attributes as in (3), are now fixed polynomial terms constructed out of the original attributes. For example, a possible input layer for a two-dimensional (2-D) problem could be $(x_1, x_2, x_1 x_2)$. The corresponding FLN architecture is shown in Fig. 1. Despite giving up hidden units, this simple architecture is capable of capturing nonlinear relationships between its original input and output patterns. A linear mapping in this polynomial space is, in fact, nonlinear in the original input space. Learning will not involve full backpropagation because no hidden units have been added. Thus, nonlinear modeling can be accomplished, by means of a simple linear learning rule, such as the delta rule [33].

Notwithstanding, the orthogonality of the set of polynomial inputs makes the choice of these terms crucial for the success of a FLN. In these networks, one polynomial unit cannot be reproduced exactly as a linear combination of the others. Thus, the FLN designer first is tempted to retain all of them. However, this is too costly from a computational point of view, especially for high degrees, and surely would lead to overfitting. Several constructive and growing algorithms have been proposed to address this issue, since they are capable of searching the nondifferentiable landscape of subsets of polynomial features. Some add units by multiplying extant ones with new inputs [13], and many others focus on the construction of multilayer FLNs [14]. A recent constructive method [15] performs a Boolean approximation of the data to select the relevant polynomials, followed by a final pruning phase. Its main limitation is the very Boolean or thresholding approximation, which can be proved to skip relevant information for some problems.

Most constructive and pruning algorithms are likely to be caught in local architectural minima. In order to avoid this problem, we have followed an evolutionary approach and counted on the global search abilities of a GA, which is described in next section. The evolutionary design of FLNs can even render some benefits over the evolution of MLPs. First, an indirect coding scheme does not come with noisy fitness evaluations. This is due to the linear nature of the phenotypes and, thus, the absence of local minima in their error landscapes. Therefore, there is a one-to-one mapping between genotypes and phenotypes. Second, the absence of backpropagation, together with our specific initial population, leads to reduced training times. We can afford large populations, and still get results fast enough. Third, due also to the absence of hidden layers, it becomes easy to penalize complex architectures. The penalizing term of our fitness function yields surprisingly simple architectures as shown below.

### III. EFLN IMPLEMENTATION

The main idea behind our system can be expressed very easily: EFLN evolves a population of FLNs applied to the classification of a dataset. Each individual in the population specifies a set of input attributes, which completely determines each FLN architecture due to the absence of hidden neurons. The genetic operators search for effective individuals capable of performing a linear discrimination between a set of pattern classes. Next, let us describe some implementation details.

#### A. Coding Scheme

Each network is encoded by means of a binary chromosome of length equal to the number of available polynomial terms. A bit 1 specifies that the corresponding polynomial is fed into the network. For a $d$-dimensional classification problem with attributes $(x_1, x_2, \cdots, x_d)$ there are

$$
\frac{(d+r)!}{d!r!}
$$

(6)

possible polynomials up to degree $r$. Decoding requires establishing an order for the polynomials. The following convention was found to be convenient: polynomials are ordered by increasing degree and, within each degree, polynomials are ordered according to increasing sums of their factors’ subindexes. The following loop generates degree $r$ polynomials in this order:

$$
\sum_{i_1=1}^{d} \sum_{i_2=i_1}^{d} \cdots \sum_{i_r=i_{r-1}}^{d} x_{i_1} x_{i_2} \cdots x_{i_r}.
$$

(7)

For example, two attributes yield six polynomials up to degree 2

$$
(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2).
$$

The independent coefficient will be dropped since our networks always make use of a bias unit. Thus, a chromosome such as $(1, 0, 0, 1, 0)$ codes a linear network with two weights (apart from the bias term) connecting units $x_1$ and $x_1 x_2$ with the output units.
TABLE I
TEST ERROR RATES OF SINGLE-FEATURE FLNs FOR THE MUSHROOM1 DATABASE

<table>
<thead>
<tr>
<th>Feature</th>
<th>Error %</th>
<th>Feature</th>
<th>Error %</th>
<th>Feature</th>
<th>Error %</th>
<th>Feature</th>
<th>Error %</th>
<th>Feature</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_{29}</td>
<td>10.73</td>
<td>x_{40}</td>
<td>39.29</td>
<td>x_{126}</td>
<td>46.13</td>
<td>x_{32}</td>
<td>50.42</td>
<td>x_{25}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{27}</td>
<td>22.30</td>
<td>x_{45}</td>
<td>40.67</td>
<td>x_{39}</td>
<td>46.58</td>
<td>x_{24}</td>
<td>50.42</td>
<td>x_{29}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{63}</td>
<td>22.70</td>
<td>x_{52}</td>
<td>41.75</td>
<td>x_{65}</td>
<td>46.58</td>
<td>x_{84}</td>
<td>50.76</td>
<td>x_{31}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{97}</td>
<td>22.94</td>
<td>x_{7}</td>
<td>42.49</td>
<td>x_{19}</td>
<td>46.68</td>
<td>x_{66}</td>
<td>50.81</td>
<td>x_{33}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{102}</td>
<td>22.99</td>
<td>x_{70}</td>
<td>42.84</td>
<td>x_{56}</td>
<td>46.68</td>
<td>x_{50}</td>
<td>50.96</td>
<td>x_{35}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{39}</td>
<td>23.04</td>
<td>x_{110}</td>
<td>43.03</td>
<td>x_{94}</td>
<td>47.07</td>
<td>x_{70}</td>
<td>51.01</td>
<td>x_{38}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{40}</td>
<td>23.04</td>
<td>x_{79}</td>
<td>43.18</td>
<td>x_{1}</td>
<td>47.17</td>
<td>x_{25}</td>
<td>51.06</td>
<td>x_{46}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{21}</td>
<td>24.57</td>
<td>x_{90}</td>
<td>43.23</td>
<td>x_{24}</td>
<td>47.22</td>
<td>x_{57}</td>
<td>51.16</td>
<td>x_{48}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{22}</td>
<td>24.57</td>
<td>x_{120}</td>
<td>43.53</td>
<td>x_{113}</td>
<td>47.27</td>
<td>x_{65}</td>
<td>51.16</td>
<td>x_{57}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{45}</td>
<td>25.70</td>
<td>x_{75}</td>
<td>43.62</td>
<td>x_{65}</td>
<td>47.51</td>
<td>x_{47}</td>
<td>51.26</td>
<td>x_{59}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{64}</td>
<td>26.83</td>
<td>x_{78}</td>
<td>43.77</td>
<td>x_{115}</td>
<td>47.56</td>
<td>x_{104}</td>
<td>51.30</td>
<td>x_{77}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{117}</td>
<td>27.67</td>
<td>x_{51}</td>
<td>43.87</td>
<td>x_{23}</td>
<td>47.61</td>
<td>x_{85}</td>
<td>51.47</td>
<td>x_{77}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{98}</td>
<td>28.71</td>
<td>x_{53}</td>
<td>43.92</td>
<td>x_{4}</td>
<td>48.06</td>
<td>x_{109}</td>
<td>51.45</td>
<td>x_{80}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{107}</td>
<td>30.43</td>
<td>x_{54}</td>
<td>43.92</td>
<td>x_{61}</td>
<td>48.06</td>
<td>x_{106}</td>
<td>51.50</td>
<td>x_{86}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{111}</td>
<td>31.46</td>
<td>x_{58}</td>
<td>43.97</td>
<td>x_{41}</td>
<td>48.65</td>
<td>x_{112}</td>
<td>51.55</td>
<td>x_{87}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{99}</td>
<td>32.89</td>
<td>x_{125}</td>
<td>44.41</td>
<td>x_{3}</td>
<td>48.79</td>
<td>x_{92}</td>
<td>51.70</td>
<td>x_{88}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{102}</td>
<td>33.14</td>
<td>x_{45}</td>
<td>44.46</td>
<td>x_{14}</td>
<td>49.14</td>
<td>x_{6}</td>
<td>51.73</td>
<td>x_{92}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{100}</td>
<td>34.42</td>
<td>x_{44}</td>
<td>44.66</td>
<td>x_{121}</td>
<td>49.28</td>
<td>x_{13}</td>
<td>51.80</td>
<td>x_{93}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{122}</td>
<td>36.48</td>
<td>x_{10}</td>
<td>44.66</td>
<td>x_{114}</td>
<td>49.48</td>
<td>x_{15}</td>
<td>51.85</td>
<td>x_{96}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{127}</td>
<td>37.96</td>
<td>x_{72}</td>
<td>44.76</td>
<td>x_{124}</td>
<td>49.68</td>
<td>x_{17}</td>
<td>51.90</td>
<td>x_{100}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{83}</td>
<td>38.45</td>
<td>x_{50}</td>
<td>45.10</td>
<td>x_{71}</td>
<td>49.83</td>
<td>x_{102}</td>
<td>51.90</td>
<td>x_{102}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{36}</td>
<td>38.65</td>
<td>x_{81}</td>
<td>45.10</td>
<td>x_{60}</td>
<td>50.02</td>
<td>x_{2}</td>
<td>52.04</td>
<td>x_{103}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{37}</td>
<td>38.65</td>
<td>x_{20}</td>
<td>45.15</td>
<td>x_{73}</td>
<td>50.12</td>
<td>x_{9}</td>
<td>52.04</td>
<td>x_{108}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{85}</td>
<td>38.70</td>
<td>x_{18}</td>
<td>45.20</td>
<td>x_{82}</td>
<td>50.12</td>
<td>x_{12}</td>
<td>52.04</td>
<td>x_{123}</td>
<td>52.04</td>
</tr>
<tr>
<td>x_{76}</td>
<td>38.90</td>
<td>x_{116}</td>
<td>45.40</td>
<td>x_{91}</td>
<td>50.27</td>
<td>x_{14}</td>
<td>52.04</td>
<td>x_{9}</td>
<td>54.11</td>
</tr>
</tbody>
</table>

Output codification is independent of the chromosome and makes use of a bipolar 1-of-N coding scheme. For example, in a three-class problem, classes would be coded as $(0, 0.9, -0.9), (-0.9, 0.9, -0.9)$, and $(-0.9, -0.9, 0.9)$, respectively. One output unit is enough in two-class problems. The value 0.9 is used to minimize the saturation of the delta learning rule.

B. Seeding Scheme

Our algorithm depends critically on the initial population from which the evolution begins. Instead of starting from a random population, we have found it much more efficient to start from a pool of single-feature networks. These latter networks are FLNs constructed out of a single polynomial feature plus the bias term. Using these individuals as the start point makes sense since they are the simplest available models. In this respect, our coding scheme makes things easy because it associates the simplest models to the simplest strings, i.e., strings with just one bit set to one. For instance, a classification problem in two dimensions would be allocated the following initial pool up to degree 2 (5 polynomial terms):

$$(1, 0, 0, 0, 0) \quad (0, 1, 0, 0, 0) \quad (0, 0, 1, 0, 0) \quad (0, 0, 0, 1, 0)$$

Had we used a more direct coding approach, it would have been much more involved to determine actually which strings are associated with minimal models.

The seeding scheme described above has very nice properties. It forces the GA to select initially based on the discriminant capabilities of individual polynomial terms, since each term is put to the test individually. This fact reduces the probability of hitchhiking. The chance that irrelevant bits take part of the final solution is much lower than when random initial pools are used. Besides, as is usual with constructive algorithms, computation time benefits much from trying simpler models first.

C. Genetic Operators and Fitness Function

Universal stochastic selection and uniform crossover are used. This kind of crossover is not sensitive to bit ordering as happens with one-point crossover, thus avoiding the permutation problem [34]. Elitism is imposed, to make sure that the best individual survives its generation. Thus, one copy of the best chromosome is placed automatically into the next population. The rest of the population is integrated by the best individuals found among the old population and the offspring. Although mutation has proved to be a key ingredient in EFLN, performance does not seem to depend critically on the probability used. In our experiments, 15% of chromosomes get a bit mutation.

The output calculated by the FLN architecture with genotype $c$ for a binary classification problem in $d$ dimensions up to degree $r$ is

$$y = \sum_{i=1}^{N} c_i \phi_i(x)$$

where $\phi_i(x)$ is the polynomial term associated with bit $c_i$ and $N$ is the number of terms up to degree $r$. Quadratic error minimization leads to the following weight updating rule

$$\Delta w_i = \mu c_i (t - y) \phi_i(x)$$

where $\mu = 0.1$ is the learning rate, $t$ is the desired output (0.9 for one class and -0.9 for the other) and $y$ is the output (8) proposed by the net for pattern $x$. Networks are trained for 100
epochs on the training set, and the final weights are those corresponding to the epoch with lowest classification error for the validation set. This is a variety of early stopping traditionally used to avoid overfitting in neural systems [1]. The fitness function for chromosome $c$ is determined as follows

$$f(c) = E_{\text{min}}(c) + \eta(c) \text{div}(d + r)!/d!r!$$

(10)

where $\eta(c)$ is the number of 1s in chromosome $c$, $E_{\text{min}}$ is the sum of training and validation error rates reached by the network in the selected epoch, and parameter $\alpha$ can be tuned in order to find a compromise between the complexity of the architecture, penalized by the second component of (10) and its corresponding performance. Any value $\alpha \ll 1$ will guarantee that performance will never be sacrificed for architectural simplicity.

### D. Polynomial Degree

The choice of polynomial degree is obviously a key question for our learning algorithm. It is the only remaining architectural parameter. Given the ease with which parameters can be coded into chromosomes, one could be tempted to evolve the degree itself. However, this would increase the complexity of the model significantly, and it has been a main concern in this paper to keep things as simple as possible. Alternatively, the complexity term of the fitness function could be changed in order to penalize features in proportion to their degree. This sounds promising, but would mean introducing extra parameters in our system, those necessary to weigh each degree.

In order to avoid the introduction of extra parameters, we will be following a stepwise procedure to address the selection of the polynomial degree. We will always start evolution just making use of the original attributes. This makes sense since some problems can be solved linearly, perhaps after rejecting some noisy or irrelevant attributes. This possibility should be always tried first before attempting any further nonlinear modeling. Afterwards, if the error reached by the best linear individual on the validation set is not satisfactory, a second-order evolution will be carried out. This time, the best solution found during the first-order (degree 1) evolution will be added to the initial pool. In this way, the search is focused around it and the GA tries to improve what we had already reached by using the original features. We have checked this as a simple way to accelerate convergence in high-degree polynomial spaces and control overfitting, since linear models tend to generalize quite well in most problems. This process will be repeated until a good solution to the classification problem is found, i.e., a satisfactory validation error rate is reached. For higher than second-order evolutions, we run a high risk of overfitting because the number of terms grows very quickly with the degree. In practice, this means that third-order and higher solutions should be kept only when they yield very significant validation error rate reductions.

Our algorithm can then be summarized as follows.

1) Set the number $p$ of evolutions.
2) Run \( p \) evolutions from a pool of first-order single-feature FLNs, i.e., FLNs fed with one original input attribute plus bias term.
3) Save the fittest individual found during these first-order evolutions.
4) Repeat the following steps until attaining a satisfactory validation error rate:
   a) increase the polynomial degree;
   b) run \( p \) evolutions from a pool of single-feature FLNs plus the best individual saved in point 3.
5) Return the fittest individual or, alternatively, a series of top performers.

IV. EXPERIMENTS

The availability of results, with previous evolutionary and constructive algorithms, has guided the selection of the following varied datasets taken from the UCI repository of machine learning databases, some of them especially prepared by Prechelt [35] for neural network learning.

- **Cancer1**: Breast cancer tumors described by nine attributes gathered by microscopic examination and classified as benign (65.5%) and malignant.
- **Diabetes1**: Female Pima Indian patients characterized by two pieces of personal data (age and number of pregnancies) and six results of medical examinations, classified as diabetes positive and negative (65.1%).
- **Ionosphere**: Radar returns (34 attributes) collected by a system in Goose Bay, Labrador, and classified as good or bad, depending on their showing evidence of some type of structure in the ionosphere.
- **Mushroom1**: Gillied mushrooms in the Agaricus and Lepiota family described by shape, color, odor, and habitat (125 binary attributes) and classified as edible (52%) or poisonous.

A. An Illustrative Example: Mushrooms

One of the advantages of the evolutionary system described here is the ease of interpreting the models obtained as compared with the evolution of multilayer architectures. Let us examine this point with a specific evolution on the mushroom1 dataset. We hope this example will make clear the EFLN implementation details described in Section III.

The purpose here is to discern edible from poisonous mushrooms by making use of attributes such as shape, color, odor, and habitat. The multicategorical nature of these features leads to a codification by means of 125 binary attributes. Given that the two classes can be separated completely by means of a linear model fed with the whole set of features [35], the main challenge consists in determining which of these features are really useful. Therefore, this set will serve to check EFLN’s ability to identify and recombine relevant features, while removing noisy ones. This simple, but powerful, characteristic is rarely found among evolutionary neural networks and yields surprisingly compact solutions such as the one described next.

Although universal stochastic selection and uniform crossover are used in all of the experiments conducted in Section IV-C, standard roulette wheel selection and one-point crossover are applied to this and the following illustrative example, showing that the success of EFLN does not depend critically on the type of GA operators actually used, but rather on the initial population. As discussed in Section III-B, evolution is started from a pool of as many single-feature FLNs as there are input attributes in the database. Thus, the initial population is integrated by 125 chromosomes of length 125

\[
\mathbf{e}_i = (\delta_{ij}) \quad \forall \ i, j = 1, \cdots, 125
\]

where \( \delta \) is Kronecker’s delta. The search space has dimension \( 2^{125} \), which precludes any kind of exhaustive search. Each of the initial individuals codes single-attribute networks, whose probabilities of selection are thus proportional to the discriminant capabilities of individual features. This kind of initial population discourages noisy and irrelevant features from hitchhiking with true relevant ones, avoiding unnecessarily complex architectures. The diversity of the initial population becomes obvious in Table I, where mushroom1’ features have been ordered by their corresponding FLN performance on the test set. Attribute 29, absence of odor, is distinguished clearly from the rest. The corresponding FLN proves to be the fittest (4.57) with an 11% test classification error. In fact, absence of odor seems to be characteristic of poisonous mushrooms: only 30 out of 974 test poisonous mushrooms smell. The test classification matrix for this FLN is

\[
\begin{bmatrix}
E & P \\
988 & 69 \\
30 & 944
\end{bmatrix}
\]

where E stands for edible and P for poisonous. The rows correspond to the true class and the columns to the system’s response. Thus, 30 poisonous mushrooms are classified as edible, and 188 edible mushrooms are labeled as poisonous. Odor is definitely a key feature, since later evolution proves that more precise odor descriptions (almond, anise) help improve this classification matrix. The data mining ability of the GA can find solutions from individually noninformative features. Thus, although neither feature 23 (almond odor with 48% test error rate) nor 121 (meadow habitat with 49% test error rate) discriminate by themselves, when combined with feature 29, they yield quite a good solution (5% test error). The classification matrix for this new network (23, 29, 121) becomes

\[
\begin{bmatrix}
E & P \\
988 & 69 \\
30 & 944
\end{bmatrix}
\]

showing that the added features help with some previously misclassified edible mushrooms. This is the kind of knowledge that our system returns while constructing the neural model.

Table II shows the best solutions successively found by EFLN up to convergence. Column 1 displays the attributes used (a bias unit is always present). Column 2 indicates the generation in which the individual was found. Column 3 shows the fitness of our system returns while constructing the neural model.
Added features are only retained when they yield an improvement in performance because of our fitness function’s penalizing term. This behavior can be observed in conventional constructive algorithms, such as cascade correlation or constructive backpropagation [12], [36]. On the contrary, feature dropping is more difficult to implement in this kind of algorithms. For example, solution \(23 + 29 + 44 + 121\) is later surpassed by \(23 + 29 + 121\), where fewer units yield similar generalization. The algorithm realizes that feature 44 is not adding any useful information. This kind of change is difficult to achieve for gradient descent algorithms, but is quite common in global search methods dealing with multiple solutions simultaneously. Not only are features dropped, but several can be added simultaneously. It long has been acknowledged that adding units one at a time may not lead to optimal architectures. However, it is not obvious how to do this in standard constructive algorithms. Here, genetic operators serve this purpose quite naturally. For example, network \(2 + 23 + 29 + 60 + 77 + 108 + 115 + 121\) yields \(2 + 23 + 24 + 29 + 60 + 108 + 115 + 121\), where feature 24 is added and 77 is removed, reducing the misclassification rate from 1.92% down to 0.59%. This behavior of EFLN stems from the global search abilities of the GA, especially in the first generations, as can be checked by examining column 2 of Table II. The two good individuals, found in generation 4, are located distant from each other, showing evidence of the exploratory component of the GA. Later in the evolution, the improvements are usually one bit apart from each other, being the result of the exploitation component of the GA.

The final network is composed of eight input units, with the following meanings: bruises (21); almond odor, anise odor, and absence of odor (23, 24, and 29, respectively); gill size (39); scaly surface before ring (66); green spore print color (108); and meadow habitat (121). EFLN has been able to find a zero-error solution, making use of only 7% of the original attributes. The evolution of the best individual, as well as the mean one, are shown in Fig. 2. It takes almost 70 generations to complete convergence, but the best individual is already found in generation 58.

### B. Another Example: Breast Cancer and Degree 2 Polynomial Terms

The cancer1 database contains nine attribute tumors to be classified into benign and malignant. The nine input attributes describing each tumor make a total of 54 polynomials up to degree 2 (apart from the independent coefficient). This example illustrates the ability of EFLN to extract good polynomial features. An intermediate individual of an evolution will allow us to visualize this complex database in a plane.

Table III is the equivalent for the cancer1 dataset of Table II, discussed in the previous section. It can be read from this table that the FLN fed with attribute \(x_6x_8\) (bare nuclei times normal nucleoli) is the fittest (1.68) of all the single-attribute networks. Only 4% of the test tumors are misclassified with this simple network. The fittest network is found in the tenth generation, with the following input layer: \(x_1, x_2, x_8, x_6\), and fitness 0.93.

Let us discuss the best FLN, with two input attributes discovered in this run: \(x_8, x_1x_6\), i.e., (normal nucleoli, clump thickness times bare nuclei). The corresponding weights are

\[
\begin{align*}
\text{bias} &= 1.06 \\
 w_1 &= 1.20 \\
 w_2 &= 5.10.
\end{align*}
\]

In this example, we are making use of a bipolar sigmoidal function in the output units of the networks. These activations are dropped in Section IV-C, in order to make fitness evaluation really noiseless. Here the output returned by the network is

\[
f(x) = \frac{2}{1 + \exp(-1.06 - 1.2x_8 - 5.1x_1x_6)} - 1.
\]

This function makes use of only three out of the nine original attributes of each tumor, and yields the following classification frontier between tumors

\[
1.06 = -1.2x_8 - 5.1x_1x_6
\]
number of generations used till convergence.

rows corresponding to degree 2. Column 9 shows the average
rows associated with degree 1, and the nonlinear MLPs in the
MLPs [35], called HMLPs. The linear MLPs are shown in the
by Prechelt for his hand-designed and backpropagation trained
validation error rates. Column 8 shows the best models obtained
of the fittest solution, i.e., the one with lowest training plus

which is represented in Fig. 3 for the test dataset. A simple line
is capable of discriminating effectively between benign and ma-
lignant tumors in this polynomial space.

C. Analysis of Experimental Results

We report in this section about the results of the experiments
conducted on the cancer, diabetes, and ionosphere databases and
compare them with other evolutionary, constructive, and ma-
chine learning algorithms, when available.

1) Experimental Setup: All of the data sets used in this paper
have been partitioned into three different sets: training, valida-
tion, and test sets. Our FLN networks are trained for 100 epochs
on the training set. The validation set is used to select the op-
timal epoch, in order to avoid overfitting. The test set is not used
during the learning phase whatsoever. It is used only to assess
the generalization capabilities of our algorithm. For the cancer
and diabetes datasets, we have used the partitions cancer1 and
diabetes1, respectively, prepared in PROBEN1 [35]. Cancer1
contains 350 training, 175 validation and 174 test patterns. Di-
betes1 contains 384 training, 192 validation and 192 test patterns.
The ionosphere dataset has been taken directly from the
UCI repository of machine learning datasets, in order to make
an specific comparison of EFLN with another constructive al-
gorithm of HOPs [15]. This set has been partitioned randomly
into three sets: training (200), validation (76), and test (75). We
have run our algorithm as described in Section III-D. Let us start
discussing the medical diagnosis problems.

2) The Cancer and Diabetes Datasets: The average and
best EFLN results for the cancer1 and diabetes1 datasets can be
found in Table IV. Columns 3, 4, and 5 show the mean training,
validation, and test error rates, respectively, over $p = 100$ runs.
Column 6 shows the mean number of weights used in the final
individuals of each run. Column 7 contains the test error rate of
the fittest solution, i.e., the one with lowest training plus
validation error rates. Column 8 shows the best models obtained
by Prechelt for his hand-designed and backpropagation trained
MLPs [35], called HMLPs. The linear MLPs are shown in the
rows associated with degree 1, and the nonlinear MLPs in the
rows corresponding to degree 2. Column 9 shows the average
number of generations used till convergence.

A simple observation can be extracted from the first-order
evolutions summarized in Table IV: rejecting some original fea-
tures leads to improved linear models. The best linear percep-
tron for the cancer1 database (2.93% error rate) clearly is out-
performed by the best FLN (1.72% error rate). The reason stems
from its not using features 1, 3, and 4. Likewise, the fittest FLN
for the diabetes1 database (20.83% error rate) discards features
3 and 7, and outperforms the best linear perceptron (25.83%
error rate). It is well known how other learning machines, such
as decision trees, benefit from removing irrelevant attributes [8].
This ability to reject information is one of the main advantages
of an evolutionary approach that adapts the input layer. Let us
next comment on higher order evolutions.

As expected, average fitness (essentially training plus valida-
tion error rates) is decreased for the two datasets by increasing
the polynomial degree. After adding second-order polynomial
terms, the average validation error rate gets reduced by 21% and
9% for the cancer1 and diabetes1 datasets, respectively. This
yields better generalization performance only for the cancer1
dataset. Notwithstanding, the degree-2 results are preferable be-
cause of the reduced validation error rates they yield. The hand-
designed perceptrons are clearly outperformed by our models
for the diabetes1 dataset. It is also noticeable that the mean
number of weights used gets doubled for both problems by in-
creasing the polynomial degree.

Higher order evolutions should be taken very cautiously be-
cause, as the average test error rates in Table IV suggest, we run
a big risk of overfitting. The reason is that the number of avail-
able terms grows very quickly with the degree. For instance,
for the cancer1 dataset, we move from 54 to 219 polynomial
terms by increasing the degree from two to three. Degree 3
and higher solutions should be kept only when they yield ex-
tremely good validation error reductions. This is not the case
here since the reductions are 17% and 11% for the cancer1
and diabetes1 datasets, respectively. The overfitting is especially
high for the diabetes1 dataset, whose generalization error rate
increases from 21.92% to 23.29%.

Table V compares the results obtained by EFLN with those
reported by hand-designed MLPs [35], FNNCA [37], and
EPNet [20] for the cancer1 database, in terms of architectural
complexity (average, minimum and maximum number of
connections), test error rates and computational cost. FNNCA
is a neural network constructive algorithm based on a variant of
the quasi-Newton method known as SR1/BFGS. It constructs
single-hidden-layer perceptrons starting with two hidden units
and adding one at a time till reaching a zero training error rate
or stopping when no performance improvement is observed.
EPNet is a recent evolutionary programming neural network
learning algorithm that not only evolves architectures, but
also connection weights or behaviors. Five different mutation
operators reflect this emphasis on evolving behaviors.

The performance is quite similar for all of the algorithms. By
contrast, the architectures vary considerably. The most popu-
lated networks are those obtained manually for MLPs as ex-
pected. FNNCA is reported to yield an average number of 2.86
hidden units over 50 runs. This brings us to 38 connections on
average if two output units and fully connected adjacent layers
are considered. EPNets capacity to evolve sparsely connected
networks yields a minimum number of connections, lower than FNNCAs: 15 weights against 26. However, the average number of connections is slightly superior. The simplest architectures are those found with our algorithm, which yields 13 connections on average, 50% below the best of all other algorithms.

Table VI compares the results found by EFLN, with those reported by other researchers for the diabetes dataset [35], [8], [20]. The algorithms considered are as follows:

- EPNet: Evolutionary programming neural learning algorithm;
- Classical Discriminant algorithms: Discr (linear), Quadisc (quadratic), and Logdisc (logistic);
- Decision Tree Algorithms: AC2, NewID, Cal5, C4.5, CART, and IndCART;
- Rule based algorithms: CN2, ITrule, and BayTree;
- NaiveBay: Bayes density estimation with attribute independence;
- ALLOC80: Multiplicative kernel method with smoothing parameters;
- DIPOL92: Piecewise linear classifier;
- k-NN: k nearest neighbors;
- CASTLE: Bayesian algorithm that returns a polytree providing a graphical explanation of the probabilistic relationships between attributes and classes;
- Neural learning algorithms: Kohonen, Backprop, RBF, and LVQ.

The comparison shown in Table VI should be taken cautiously, since most of these results were found by 12-fold cross validation, except EPNet and ours. It is clear that no algorithm performs exceptionally well, due to the large degree of noise present in this dataset. The class label of this dataset is a binarized version of a continuous attribute indicative of certain types of diabetes. However, it seems not to be very correlated with the medical condition of being diabetic. ALLOC80 and k-NN are the worst performers. EFLN yields the best classification performance, followed by Logdisc and EPNet.

The simplicity of the networks found by EFLN, as compared to most other algorithms, is most noticeable. The diabetes1 problem is solved with a mean number of connections equal to 12. This comes as no surprise, since our fitness function explicitly penalizes complex architectures and the evolution begins with the simplest available networks. HMLPs pivot architectures have a much bigger number of connections [35]: 370 for diabetes1. The reason is that the author was not especially concerned about finding minimal architectures, while hand designing his MLPs. EPNet yields much more compact architectures: on average, it makes use of 52 connections for the diabetes dataset. This figure is still significantly above our mean number of weights. Cal5 makes use of only eight nodes in its decision tree, a surprisingly compact solution that even improves our mean number of connections. However, its generalization performance is quite poor for this database (25%). Finally, NewId, AC2, and C4.5 have 119, 116, and 32 nodes, respectively.

3) The Ionosphere Database: This set has been chosen, in order to make an specific comparison of our learning algorithm with another constructive algorithm of HOPs [15], that will be called CHOP from now on. CHOP starts with a Boolean approximation of continuous data. This Boolean dataset is used to construct Boolean expressions that are translated into HOPs. Later, these HOPs are trained with the original data and finally pruned.

Table VII gathers the results of the EFLN experiments conducted on the ionosphere dataset. This dataset has 34 original attributes, what makes a total of 629 polynomial terms up to degree 2. Only first-order and second-order evolutions have been tried because the validation error reached by the latter is almost zero. Columns 2, 3, and 4 show the average training, validation, and test error rates, respectively, over 10 runs. Column 5 contains the test error rate of the fittest solution, the one with lowest training plus validation error rates. Column 6 shows the average number of weights used in the final individuals of each run. Column 7 contains the mean number of generations used until convergence.

It becomes clear from this table that the second-order solutions outperform the first-order ones, yielding almost zero mean validation error. In fact, the first-order evolutions yield quite poor solutions for this dataset, with an average validation error rate close to 7%. However, second-order evolutions reduce...
this average figure by 88%. The fittest second-order individual reaches a surprisingly good performance: 1.33% test error rate.

Table VIII compares these results with those reported previously [38] and CHOPs results [15]. CHOP is reported to make use of a third-order network, that learns the training set to 99% recognition. This network has 141 connections, and goes through a number of 20 pruning simulations. The smallest of these pruned networks has 34 connections, a number only slightly superior to the average number of weights of our networks (30). However, our best FLN performance is 75% better, with a 1.33% test error rate. The second-best performer is IB3, followed by C4.5 and CHOP.

4) Computational Analysis: From a computational point of view, our evolutionary approach compares well with other evolutionary algorithms dealing with MLPs. For instance, EPNet makes use on average of 132 generations for the diabetes database and 137 for the cancer database. The corresponding figures for EFLN are: 93 generations for degree 1 and 152 for degree 2 and the diabetes1 dataset. For the cancer1 dataset EFLN makes use of 94 (degree 1) and 146 (degree 2) generations. The higher the degree, the more generations are required for convergence, as expected. Encouragingly enough, this number does not grow as quickly as the search space dimension growing rate would have suggested. Anyway, training each FLN is obviously much simpler than training MLPs, which sensibly reduces the CPU time used to handle each of our generations as compared to those evolved by EPNet, for instance.

HON constructive algorithms such as CHOP [15] or neural network constructive algorithms such as FNNCA [37] tend to train much faster than evolutionary learning machines. For instance, FNNCA makes use of 471 epochs on average to solve the cancer problem (see Table V). However, these algorithms are more prone to be caught in local architectural minima than algorithms such as EFLN or EPNet. Likewise, hand-designed MLPs, like those reported by Prechelt [35], can be trained quickly. For example, cancer1 is reported to train for 152 epochs, with a number of relevant epochs equal to 133. Needless to say, the search for an optimal architecture is not addressed in a systematic way. Any algorithm searching for optimal architectures will be likely to spend much time probing the architectural landscape. Our approach becomes specially affordable, due to the simplicity of each individual network and our bottom-up procedure.

With respect to memory requirements, it must be noticed that our polynomial terms are saved in memory and not recalculated before training each FLN. This is not very demanding for degree 2 and most real-life problems and accelerates evolution considerably. A key point is that search is focused on compact networks from the very beginning, due to both the complexity penalizing term and the initial population. Moreover, EFLN not only encourages networks that perform well, but architectures that can be trained quickly enough, given the small and fixed number of training epochs used.

V. DISCUSSION

Although there is not much work on HONs in the neurocomputing literature, the use of polynomials in pattern classification dates back to the beginning of this discipline [39]. The statistician making use of polynomial regression (PR) focuses on exploiting linear dependencies between polynomials, in order to avoid solution manifolds. As a consequence, the number of terms used gets reduced significantly. For example, the minimum residual variance strategy [40] is a pivot selection rule of polynomial terms designed to optimize a quadratic measure of discriminative power. This is an interesting, but greedy, algorithm that does not guarantee optimal solutions. By contrast, the global search capabilities of GAs make EFLN less prone to be caught in local minima than these constructive polynomial regression models. On the other hand, the possibility of adding sigmoidal activations to the FLN output units makes them superior to classical PR. There are, of course, more powerful nonlinear regression techniques than PR such as projection pursuit [41] that even guarantees universal approximation. However,
they share with backpropagation the local minima problems we are trying to avoid here. Ultimately, this is the main reason why we limit ourselves to polynomial classification, even at the expense of losing universal approximation.

The evolution of MLPs does not usually pay much attention to the input layer. This is generally kept fixed and evolution is focused on finding optimal hidden units. Notwithstanding, there are some multilayer evolutionary algorithms that also evolve the input layer. For instance, a GA has been used to select the features fed into a neural network trained by DistAI, a new constructive neural network learning algorithm [42]. This learning algorithm adds hidden neurons one at a time by means of a greedy strategy. The authors use a constructive algorithm, in order to eliminate the need of a priori choices of network architectures. Also concerned about this point, we have preferred to keep neural learning to a minimum and rely as much as possible on the global search abilities of the GA, all of this intended to avoid local minima as much as possible.

EFLN constructive characteristics have been compared with those of other constructive algorithms, such as CHOP [15], a constructive and pruning algorithm for HOPs. The main limitation of this algorithm is the need for an initial Boolean approximation of continuous data, which can lead to suboptimal HOPs. Our GA-driven algorithm is less likely than conventional constructive algorithms to be caught in local architectural minima. EFLN has been shown to outperform CHOP for the ionsphere database. It yields extremely good networks (1.33% test error rate) for this problem with quite compact architectures (29 weights).

The attention paid by our algorithm to the input layer locates it close to some work on feature extraction and selection. Feature subset selection algorithms can be divided into two main groups, wrapper and filter methods, depending on whether or not the selection involves the learning algorithm that will later use the selected features. In general, wrapper methods demand more computational effort than filter methods. However, they are obviously much more sensitive to the representational biases of the learning system within which the selected features are to be used. Sometimes, features selected with an alternative criteria may not be optimal for the considered algorithm. Our approach is clearly a wrapper method, and, of course, not the first one to use a GA to select features. For example, a recent paper [43] analyzes features by means of a fitness function defined to be the performance of a nearest prototype classifier. The substitution of traditional crossover for a new self-crossover operation allows the authors to keep the cardinality of the evolved subset of features fixed. This work is oriented to the ranking of a given set of features, while ours is designed primarily to extract new polynomial features out of the original ones. Thus, the ability to adapt the evolved number of features becomes crucial in EFLN.

VI. Conclusion

Most classification problems become trivial once good attributes are at hand. This is what the hidden layers of MLPs do marvelously, provided a suitable architecture has been somehow found in advance. On the other hand, FLN architectures, having no hidden layers, have to be fed with optimal features from the very beginning. This is a purely nongradient oriented problem that calls for a constructive or evolutionary kind of search such as EFLN, the algorithm proposed in this paper.

We have used an indirect binary coding scheme that, given the absence of local minima in the error landscape associated with each FLN, does not give rise to noisy fitness evaluations. Training each FLN is very fast, since no hidden layers are present whatsoever. EFLN is thus an easy means to address the computational burden of evolutionary neural algorithms, one of the key concerns about these learning machines.

The main parameters of our system are the degree of the polynomial terms used and the order in which these terms are located into the chromosomes. The optimal degree is found by means of a stepwise procedure. We start with degree 1, and increase this figure until reaching satisfactory validation error rates. The fittest linear individual is always added to the higher order initial pools. For instance, the mushroom database has been solved to zero validation error using the original attributes exclusively, actually only 7% of the 125 original attributes.

The problem of the ordering of polynomials is equivalent to the permutation problem [34] or competing conventions problem [44] that appears in multilayer evolution. Any genotypic permutation of the hidden nodes of an MLP produces an equivalent network from the behavioral point of view. However, crossover will not treat all of these networks on an equal basis. Likewise, the ordering of polynomial terms in EFLN also will interfere with crossover operators such as one-point crossover. In order to avoid this problem, we have used uniform crossover, which is not sensitive to the ordering of polynomial terms.

EFLN has been shown to yield state-of-the-art recognition error rates for a variety of real-life classification problems. Moreover, the architectures found are significantly more compact than those found by other evolutionary and constructive neural network algorithms. The extra calculations generated by the higher order units can be alleviated, provided that these polynomial terms are saved in memory instead of being recalculated each time an FLN is trained.

Another important aspect of our system is the ease of interpretability. Our seeding scheme, together with the complexity penalizing term, leads to a bottom-up strategy quite amenable to interpretation. This property is difficult to find either in conventional neural networks or in other evolutionary approaches. In this sense, our system gets closer to some attempts of feature extraction based on GAs. However, our main concern is not the extraction of features, but the construction of classifiers. The fact that both extraction and classification are performed simultaneously gives EFLN very interesting properties. Whatever the dimension of the original feature space, EFLN focuses from the very beginning on small subsets of polynomial features. Therefore, noisy features are not likely to blur its functioning as would happen with conventional neural networks, for which high dimension has proved to be a true curse. In this respect, data mining applications surely will benefit from the ideas put forward in this paper.
REFERENCES