ESTIMATION OF DISTRIBUTION ALGORITHMS

ESTIMATION OF DISTRIBUTION ALGORITHMS
A New Tool for Evolutionary Computation

Edited by
P. LARRAÑAGA

J.A. LOZANO
University of the Basque Country

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

List of Figures

List of Tables

Chapter 1

Adjusting Weights in Artificial Neural Networks using Evolutionary Algorithms

Carlos Cotta, Enrique Alba
Department of Computer Science
University of Málaga
{ccottap, eat}@lcc.uma.es

R. Sagarna, P. Larrañaga
Department of Computer Science and Artificial Intelligence
University of the Basque Country
{ccbsaalr, ccplamup}@si.ehu.es

Abstract Training artificial neural networks is a complex task of great practical importance. Besides classical *ad-hoc* algorithms such as backpropagation, this task can be approached by using evolutionary computation, a highly configurable and effective optimization paradigm. This chapter provides a brief overview of these techniques, and shows how they can be readily applied to the resolution of this problem. Three popular variants of evolutionary algorithms –Genetic Algorithms, Evolution Strategies and Estimation of Distribution Algorithms– are described and compared. This comparison is done on the basis of a benchmark comprising several standard classification problems of interest for neural networks. The experimental results confirm the general appropriateness of evolutionary computation for this problem. Furthermore, Evolution Strategies seem particularly proficient techniques in this optimization domain, being Estimation of Distribution Algorithms a competitive approach as well.

Keywords: Evolutionary Algorithms, Artificial Neural Networks, Supervised Training, Hybridization

1. Introduction

Artificial Neural Networks (ANNs) are computational models based on parallel processing (McClelland and Rumelhart, 1986). Essentially, an ANN can be defined as a pool of simple processing units which communicate among themselves by means of sending analog signals. These signals travel through weighted connections between units. Each of these processing units accumulates the inputs it receives, producing an output according to an internal activation function. This output can serve as an input for other units, or can be a part of the network output. The interest of ANNs resides in the very appealing properties they exhibit, such as adaptivity, learning capability, and ability to generalize. Nowadays, ANNs have a wide spectrum of applications ranging from classification to robot control or vision (Alander, 1994).

The rough description of ANNs given in the previous paragraph provides some clues on the design tasks involved in the application of ANNs to a particular problem. As a first step, the architecture of the network has to be decided. Basicly, two major option can be considered: *feed-forward* networks and *recurrent* networks. The former model comprises networks in which the connections are strictly feed-forward, i.e., no unit receives input from a unit to which the former sends its output. The latter model comprises networks in which feedback connections are allowed, thus making the dynamical properties of the network turning to be important. In this work we will concentrate on the first and simpler model, feed-forward networks. To be precise, we will consider the so-called *multilayer perceptron* (Rosenblatt, 1959), in which units are structured into ordered layers, being connections allowed only between adjacent layers.

Once the architecture of the ANN is restricted to that of a multilayer perceptron, some parameters such as the number of layers, and the number of units per layer must be defined. After having done this, the last step is adjusting the weights of the network, so as to make it produce the desired output when confronted with a particular input. This process is known as *training* the ANN or *learning* the network weights[1]. We will focus on the learning situation known as *supervised* training, in which a set of current-input/desired-output patterns is available. Thus, the ANN has to be trained to produce the desired output according to these examples.

The most classical approach to supervised training is a domain-dependent technique known as Backpropagation (BP) (Rumelhart et al., 1986). This algorithm is based on measuring the total error in the input/output behaviour of the network, calculating the gradient of this error, and adjusting the weights in the descending gradient direction. Hence, BP is a gradient-descent local search procedure. This implies that BP is subject to some well-known problems such as the existence of local-minima in the error surface, or the non-differentiability

of the weight space. Different solutions have been proposed to this problem, resulting in several algorithmic variants, e.g., see (Silva and Almeida, 1990). A completely different alternative is the use of *evolutionary algorithms* for this training task.

Evolutionary algorithms (EAs) are heuristic search techniques loosely based on the principles of natural evolution, namely adaptation and survival of the fittest. These techniques have been shown to be very effective in solving hard optimization tasks with similar properties to the training of ANNs, i.e., problems in which gradient-descent techniques get trapped into local minima, or are fooled by the complexity and/or non-differentiability of the search space. This work will provide a gentle introduction to the use of these techniques for the supervised training of ANNs. To be precise, this task will be tackled by means of three different EA models, namely Genetic Algorithms (GAs), Evolution Strategies (ESs), and Estimation of Distribution Algorithms (EDAs).

The remainder of the chapter is organized as follows. Section 2. addresses the application of these techniques to the training of an ANN. This section gives a brief overview on the classical BP algorithm, in order to clarify the difference and distinctiveness of the EA approach, subsequently described. Some basic differences and similarities in the application of the several variants of EAs mentioned to the problem at hand are illustrated in this section too. Next, an experimental comparison of these techniques is provided in Section 3. Finally, some conclusions and directions for further developments are outlined in Section 4.

2. An Evolutionary Approach to ANN Training

As mentioned in Section 1, this section is intended to provide an overview of an evolutionary approach to weight adjusting in ANNs. This is done in Subsections 2.2 and 2.3. Previously, a classical technique for this task, the BP algorithm, is described in Subsection 2.1. This description is important for the purposes of a further combination of both –evolutionary and classical– approaches.

2.1 The BP algorithm

It has been already mentioned that the BP algorithm is based on determining the descending gradient direction of the error function of the network, adjusting the weights accordingly. It is thus necessary to define the error function in the first place. This function is the *summed squared error* E defined as follows:

$$E = \frac{1}{2} \sum_{1 \leq p \leq m} E^p = \frac{1}{2} \sum_{1 \leq p \leq m} \sum_{1 \leq o \leq n_o} (d_o^p - y_o^p)^2, \tag{1.1}$$

where m is the number of patterns, n_o the number of outputs of the network, d_o^p is the desired value of the o-th output in the p-th pattern, and y_o^p is the actual value of this output. This actual value is computed as a function of the total input s_o^p received by the unit, i.e.,

$$y_o^p = F(s_o^p) = F(\sum_{u_r \to u_o} w_{ro} y_r^p), \tag{1.2}$$

where F is the activation function on the corresponding unit, and r ranges across the units from which unit o receives input.

The gradient of this error function E with respect to individual weights is

$$\frac{\partial E}{\partial w_{ij}} = \sum_{1 \le p \le m} \frac{\partial E^p}{\partial w_{ij}} = \sum_{1 \le p \le m} \frac{\partial E^p}{\partial s_j^p} \frac{\partial s_j^p}{\partial w_{ij}} = \sum_{1 \le p \le m} \frac{\partial E^p}{\partial s_j^p} y_i^p \ . \tag{1.3}$$

By defining $\delta_j^p = -\frac{\partial E^p}{\partial s_j^p}$, the weight change is

$$\Delta w_{ij} = \sum_{1 \le p \le m} \Delta_p w_{ij} = \sum_{1 \le p \le m} \gamma \delta_j^p y_i^p, \tag{1.4}$$

where γ is a parameter called *learning rate*.

In order to calculate the δ_j^p terms, two situations must be considered: the j-th unit being an output unit or an internal unit. In the former case,

$$\delta_j^p = (d_j^p - y_j^p) F'(s_j^p) \tag{1.5}$$

In the latter case, the error is *backpropagated* as follows:

$$\delta_j^p = -\frac{\partial E^p}{\partial s_j^p} = -\frac{\partial E^p}{\partial y_j^p} \frac{\partial y_j^p}{\partial s_j^p} = -\frac{\partial E^p}{\partial y_j^p} F'(s_j^p). \tag{1.6}$$

The term $\frac{\partial E^p}{\partial y_j^p}$ can be developed as

$$\frac{\partial E^p}{\partial y_j^p} = \sum_{u_j \to u_r} \frac{\partial E^p}{\partial s_r^p} \frac{\partial s_r^p}{\partial y_j^p} = \sum_{u_j \to u_r} \frac{\partial E^p}{\partial s_r^p} w_{jr} = -\sum_{u_j \to u_r} \delta_r^p w_{jr}, \tag{1.7}$$

where r ranges across the units receiving input from unit j. Thus,

$$\delta_j^p = F'(s_j^p) \sum_{u_j \to u_r} \delta_r^p w_{jr}. \tag{1.8}$$

One of the problems of following this update rule is the fact that some oscillation can take place were γ large. For this reason, a *momentum* term α is added, so

$$\Delta w_{ij}(t+1) = \sum_{1 \le p \le m} \gamma \delta_j^p y_i^p + \alpha \Delta w_{ij}(t). \tag{1.9}$$

This modification notwithstanding, the BP algorithm is still sensitive to the ruggedness of the error surface, being often trapped into local optima. Hence, the necessity of alternative search techniques.

2.2 The Basic Evolutionary Approach

EAs can be used for adjusting the weights of an ANN. This approach is relatively popular, dating back to late 80s – e.g., see (Caudell and Dolan, 1989; Montana and Davis, 1989; Whitley and Hanson, 1989; Fogel et al., 1990; Whitley et al., 1990)– and constituting nowadays a state-of-the-art tool for supervised learning. The underlying idea is making individuals represent the weights of the ANN, using the network error function as a cost function to be minimized (alternatively, an accuracy function such as the number of correctly classified patterns could be used as a fitness function to be maximized; this approach is rarely used though). Some general considerations must be taken into account when using an evolutionary approach to ANN training. These are commented below.

The first topic that has to be addressed is the representation of solutions. In this case, it is clear that the phenotype space $\mathcal{F}$ is $\mathcal{R}^k$, where $\mathcal{R} \subset \mathbb{R}$ is a closed interval [min,max], and k is the number of weights of the ANN being trained, i.e., solutions are k-dimensional vectors of real numbers in the range [min,max]. This phenotype space must be appropriately translated to a genotype space $\mathcal{G}$ which will depend on the particulars of the EA used. In this work we will consider the linear encoding of these weights. Thus, $\mathcal{G} \equiv \mathcal{G}_w^k$, i.e, each weight is conveniently encoded in an algorithm-dependent way; subsequently, the genotype is constructed by concatenating the encoding of each weight into a linear string.

This linear encoding of weights raises a second consideration, the distribution of weights within the string. This distribution is important in connection with the particular recombination operator used. If this operator breaks the strings into large blocks using them as units for exchange (e.g., one-point crossover), this distribution might be relevant. On the contrary, using a recombination operator that breaks the string into very small blocks (e.g., uniform crossover) makes the distribution be irrelevant. A good piece of advice is grouping together the input weights for each unit. This way, the probability of transmitting them as a block is increased, in case an operator such as one-point crossover were used. Obviously, recombination is not used in some EAs, e.g., in EDAs, so this consideration should be rendered mute in such a situation.

2.3 Specific EA Details

The basic idea outlined in the previous subsection can be implemented in a variety of ways depending upon the particular EA used. We will now discuss

Figure 1.1 The weights of an ANN are encoded into a linear binary string in GAs, or into a $2k$-dimensional real vector in ESs (k weights plus k stepsizes). The EDA encoding is similar to that of the ES, excluding the stepsizes, i.e., a k-dimensional real vector.

these implementation details for the EA models mentioned in the previous section, namely, GAs, ESs and EDAs.

2.3.1 Genetic Algorithms. GAs are popular members of the evolutionary-computing paradigm. Initially conceived by Holland (Holland, 1975), these techniques constitute nowadays the most widespread flavor of EAs. In the context of traditional GAs, the encoding of solutions is approached via binary strings. More precisely, m bits are used to represent each single weight; subsequently, the k m-bit segments are concatenated into a ℓ-bit binary string, $\ell = k \cdot m$. This process is illustrated in Fig. 1.1.

This encoding of the network weights raises a number of issues. The first one is the choice of m (the length of each segment encoding a weight). It is intuitively clear that a low value of m would induce a very coarse discretization of the allowed range for weights, thus introducing oscillations and slowing down convergence during the learning process. On the contrary, too large a value for m would result in very long strings, whose evolution is known to be very slow. Hence, intermediate values for m seem to be appropriate. Unfortunately, such intermediate values seem to be problem dependent, sometimes requiring a costly trial-and-error process. Alternatively, advanced encoding techniques such as *delta coding* (Whitley et al., 1991) could be used, although it has to be taken into account that this introduces an additional level of complexity in the algorithm.

A related issue is the encoding mechanism for individual weights, i.e., pure binary, Gray-coded numbers, magnitude-sign, etc. Some authors have advocated for the use of Gray-coded numbers (Whitley, 1999) on the basis of theoretical studies regarding the preservation of some topological properties in the resulting fitness landscape (Jones, 1995). However, the suitability of such analysis to this problem is barely understood. Furthermore, the disruption caused by classical recombination operators, as well as the effects of multiple mutations per segment being performed (a usual scenario) will dilute with high probability the advantages (if any) of this particular encoding scheme. Hence, no preferred encoding technique can be distinguished in principle.

2.3.2 Evolution Strategies. The ES (Rechenberg, 1973; Schwefel, 1977) approach is somewhat different from the GA approach presented in the previous subsection. As a matter of fact, the relative intricacy of deciding the

representation of the ANN weights in a genetic algorithm contrasts with the simplicity of the ES approach. In this case, each solution is represented as it is, a k-dimensional vector of real numbers in the interval [min,max] (see Fig. 1.1)[2].

Associated to each weight w_i, a stepsize parameter σ_i for performing Gaussian mutation on each single weight is included[3]. These stepsizes undergo evolution together with the parameters that constitute the solution, thus allowing the algorithm to self-adapt the way the search is performed.

Notice also that the use of recombination operators (let alone positional recombination operators) is often neglected in ESs, thus making irrelevant the distribution of weights inside the vector.

Some works using ESs in the context of ANN training can be found in (Wienholt, 1993; Berlanga et al., 1999a; Berlanga et al., 1999b).

2.3.3 Estimation of Distribution Algorithms. EDAs, introduced by (Mühlenbein and Paaß, 1996), constitute a new tool for evolutionary computation, in which the usual crossover and mutation operators have been replaced by the estimation of the joint density function of the individuals selected at each generation, and the posterior simulation of this probability distribution, in order to obtain a new population of individuals. For details about different approaches the reader can consult Chapter 3 in this book.

When facing the weighting learning problem in the field of ANNs, discrete as well as continuous EDAs may constitute effective approaches to solve it since this problem can be viewed as an optimization problem.

If discrete EDAs are used to tackle the problem, then the representation of the individuals would be similar to the one previously explained for GAs. On the other hand, if continuous EDAs are considered, then the representation would be analogous to one used by ESs. In the last case the representation is even simpler than for evolutionary strategies as no mutation parameter is required.

Works where EDA approaches have been applied to evolve weights in artificial neural networks can be consulted in (Baluja, 1995; Galić and Höhfeld, 1996; Maxwell and Anderson, 1999; Gallagher, 2000; Zhang and Cho, 2000).

2.3.4 Memetic Algorithms. Besides the standard operators used in each of the EA models discussed above, it is possible to consider additional operators adapted for the particular problem at hand. It is well-known –and supported both by theoretical (Wolpert and Macready, 1997) and empirical (Davis, 1991) results– that the appropriate utilization of problem-dependent knowledge within the EA redounds to highly effective algorithms. In this case, this addition of problem-dependent knowledge can be done by means of a local search procedure specifically designed for ANN training: the BP algorithm.

The resulting combination of an EA and BP can be described as a *hybrid* or *memetic* (Moscato, 1999) algorithm.

The BP algorithm can be used in combination with an EA in a variety of ways. For example, an EA has been utilized in (Gruau and Whitley, 1993) to find the initial weights used in the BP algorithm for further training. Another approach is using BP as a mutation operator, that is, as a procedure for modifying a solution (Davis, 1991). Due to the fact that BP is a gradient-descent algorithm, this mutation is ensured to be monotonic in the sense that the mutated solution will be no worse that the original solution. However, care has to be taken with respect to the amount of computation left to the BP operator. Despite BP can produce better solutions when executed for a longer time, it can fall within a local optimum, being the subsequent computational effort useless; moreover, even when BP steadily progressed, the amount of improvement could be negligible with respect to additional overhead introduced. For these reasons, it is preferable to keep the BP utilization at a low level (the exact meaning of "low level" is again a matter of the specific problem being tackled, so no general guideline can be given).

3. Experimental Results

This section provides an empirical comparison of different evolutionary approaches for training ANNs. The details of these approaches, as well as a description of the benchmark used are portrayed in Subsection 3.2. Next, the results of the experimental evaluation of these techniques are presented and analyzed in Subsection 3.3.

3.1 ANNs and Databases

The algorithms described in the previous section have been confronted with the supervised training of three different ANNs. Each of these ANNs has a different architecture, and is fed with different databases. These are the following:

- KILN: This database corresponds to the fault detection and diagnosis of an industrial lime kiln (Ribeiro et al., 1995). There are 70 patterns in this database, each one comprising 8 descriptive attributes, and its ascription to one out of 8 eight different classes. The ANN architecture used in this case is 8-4-8.

- ECOLI: This database corresponds to the prediction of protein localization sites in eukaryotic cells (Nakai and Kanehisa, 1992). There are 336 patterns in this database, each one comprising 8 descriptive attributes, and its ascription to one out of 8 eight different classes. The ANN architecture used in this case is 8-4-2-8.

- BC: This database corresponds to the diagnosis of breast cancer (Mangasarian and Wolberg, 1990). There are 683 patterns in this database, each one comprising 9 descriptive attributes, and one Boolean predictive attribute ('malignant' or 'benign'). The ANN architecture used in this case is 9-4-3-1.

The weight range for each of the ANNs trained is [-10,10]. The sigmoid function $F(x) = (1 + e^{-x})^{-1}$ has been utilized as the activation function of all units.

3.2 The Algorithms

The parameterization of the GA for these problems is as follows: *populationSize* = 100, σ = Roulette-Wheel, ψ = Steady-state, *crossoverOp*= Uniform-Crossover (p_c = 1.0, 80% bias to the best parent), *mutationOp* = Bit-Flip ($p_m = 1/\ell$), m = 16 bits per weight.

As to the ES, the parameterization is even simpler: a standard (1,10)-ES without recombination, and using non-correlated mutations has been utilized. The stepsizes are mutated following the guidelines shown in (Bäck, 1996), i.e., a global learning rate $\tau = 1/\sqrt{2n}$, and a local learning rate $\tau' = 1/\sqrt{\sqrt{2n}}$.

Two instances of the EDA paradigm have been used to carry out the experiments. The difference between them corresponds to the way in which the factorization of the joint density function of selected individuals has been done. In the case that the factorization is done as a product of univariate marginal densities, we obtain the UMDA_c. If the joint density is factorized as a chain that considers statistics of order two, we refer to the algorithm as MIMIC_c. For more information about these algorithms see (Larrañaga, 2001). In the EDAs used in the experiments the number of simulated individuals at each generation was 250. The best half of the population was selected to perform the learning of the joint probability density function.

For any of the three basic algorithms (GAs, ESs, and EDAs), a maximum number of 50.000 RMSE (*rooted mean square error*[4]) evaluations across the whole training set is allowed. These algorithms have been hybridized with the backpropagation algorithm as well. This is done by training each network during 10 epochs, using the parameters γ = .1, and α = .5.

3.3 Analysis of Results

The experiments have been carried out in two different scenarios. In the first one, all patterns within each database have been utilized for training purposes. The RMSE has been used as the performance measure in this case. In the second scenario, 5–fold cross–validation has been performed. The performance measures in this case are the average RMSE for test patterns, and the percentage of correctly classified test patterns. To determine whether a pattern

Table 1.1 Results obtained with the BC database.

Algorithm	*error–training*	*error–test–5CV*	*per–test–5CV*
BP	0.4550±0.0324	0.2244±0.0074	63.2650±2.9311
GA	0.1879±0.0117	0.1125±0.0062	90.8676±1.1248
ES	0.1104±0.0017	0.0776±0.0039	95.8565±0.4529
UMDA_c	0.1184±0.0081	0.0746±0.0035	95.2353±0.4609
MIMIC_c	0.1181±0.0091	0.0753±0.0042	95.0735±0.5892
GA + BP	0.3648±0.0246	0.1817±0.0059	71.3824±3.0779
ES + BP	0.1777±0.0266	0.0952±0.0098	93.7189±1.2528
UMDA_c + BP	0.3081±0.0259	0.2747±0.0100	51.3529±3.4916
MIMIC_c + BP	0.3106±0.0018	0.2659±0.0206	54.2206±7.2556

has been correctly classified, the Boolean nature of desired output is exploited. To be precise, the actual activation values for each output unit are saturated to the closest Boolean value, and then compared to the desired output. If all saturated actual outputs match the desired output, the pattern is considered correctly classified.

Tables 1.1, 1.2 and 1.3 summarize the experimental results obtained. First of all, a general inspection at the column showing the percentage of correctly classified test patterns reveals an evident hardness-ranking: the easiest database is BC, and the hardest one is KILN. This particular ranking can be due to several factors. On one hand, it is clear that the saturation criterion used to determine whether a pattern has been correctly classified might be advantageous for BC, since just one output-per-pattern exists. On the other hand, the network architecture is more complex (and hence the ANN is more adaptable) in BC and ECOLI than in KILN. Finally, KILN has lowest number of patterns, a drawback *a priori* for learning to generalize. Actually, this hardness-ranking coincides with the ordering of databases according to their size (smallest is hardest).

Focusing in the *error-training* column, it can be seen that both ESs and EDAs offer the best results in quality and stability, the former being slightly better. It is not surprising that these two models are precisely the ones using real-coded representation of weights. Unlike the binary representation, this representation is less prone to abrupt changes in weight values[5]. This allows a better exploitation of any gradient information that might be present. Notice that the population-based search performed by these techniques makes it much

Table 1.2 Results obtained with the ECOLI database.

Algorithm	*error–training*	*error–test–5CV*	*per–test–5CV*
BP	0.2584±0.0051	0.1289±0.0017	8.3333±6.3909
GA	0.1968±0.0165	0.1001±0.0038	47.8308±7.8949
ES	0.1667±0.0085	0.0891±0.0027	65.8929±2.5301
$UMDA_c$	0.1830±0.0067	0.0808±0.0022	58.5970±5.9286
$MIMIC_c$	0.1778±0.0134	0.0802±0.0018	58.5075±4.8153
GA + BP	0.3004±0.0126	0.1522±0.0040	8.0398±5.9927
ES + BP	0.1925±0.0202	0.0939±0.0019	53.5417±4.2519
$UMDA_c$ + BP	0.2569±0.0069	0.1593±0.0011	9.8209±6.9430
$MIMIC_c$ + BP	0.2587±0.0064	0.1585±0.0010	10.4179±7.3287

Table 1.3 Results obtained with the KILN database.

Algorithm	*error–training*	*error–test–5CV*	*per–test–5CV*
BP	0.3334±0.0011	0.1664±0.0003	0±0
GA	0.2379±0.0112	0.1229±0.0040	10.7619±5.3680
ES	0.2361±0.0043	0.1243±0.0023	19.7143±5.2511
$UMDA_c$	0.2398±0.0025	0.1132±0.0002	6.1429±3.1623
$MIMIC_c$	0.2378±0.0077	0.1132±0.0002	8.4286±3.6546
GA + BP	0.3202±0.0392	0.1686±0.0076	4.5714±4.0301
ES + BP	0.2367±0.0074	0.1241±0.0039	8.2857±5.1199
$UMDA_c$ + BP	0.2760±0.0059	0.1437±0.0058	2.2857±2.4467
$MIMIC_c$ + BP	0.2751±0.0086	0.1420±0.0044	3.1429±3.3537

more unlikely getting trapped into local optima (this is specifically true in the non-elitist ES model used), and allows a better diversification of the search.

Moving to the 5CV columns, the results are fairly similar: again ESs and EDAs yield resembling results, generally better than GAs. An interesting fact that it is worth mentioning is the superiority of EDAs over ESs in test error, and the superiority of the latter in the percentage of correctly classified patterns. This might indicate a difference in the underlying search progress. Nevertheless, more extensive results would be required in order to extract convincing conclusions.

Notice also that the hybrid models of EAs and BP perform worse than non-hybridized EAs. This can be due to several reasons. First of all, it was mentioned before that the balance of computation between BP and EAs is a very important factor. The parameterization chosen in this work might have been inadequate in this sense. Also, it can not be ruled out that different results would be obtained, were the BP parameters γ and α given different values.

A deeper analysis of the results was done by testing the null hypothesis that stated that the results achieved by some groups of algorithms followed the same density distribution. For this task the non–parametric Kruskal–Wallis and Mann–Whitney tests were used. This analysis was carried out with the statistical package S. P. S. S. release 10.0.6. The results were as follows:

- Between no memetic algorithms. Using the Kruskal–Wallis test, we tested the null hypothesis that the results obtained by GA, ES, UMDA_c and MIMIC_c follow the same density distribution obtaining that, for the three databases and the three parameters (error–training, error–test–5CV and per–test-5CV), the differences were statistically significant ($p < 0.05$) except for the error–training parameter in the KILN database ($p = 0.5743$).

- Between memetic algorithms. Comparing the density distributions followed by the results of GA+BP, ES+BP, UMDA_c+BP and MIMIC_c+BP by the Kruskal–Wallis test, we obtained that there were statistically significant differences ($p < 0.05$) in the three databases and for the three parameters.

- Between one no memetic algorithm and its corresponding memetic algorithm. We also compared the differences in the behavior of the no memetic algorithms and their corresponding memetic ones, that is, GA vs GA+BP, ES vs ES+BP, UMDA_c vs UMDA_c+BP and MIMIC_c vs MIMIC_c+BP. Using the Mann–Whitney test we obtained that for the comparisons between GA vs GA+BP, UMDA_c vs UMDA_c+BP and MIMIC_c vs MIMIC_c+BP the differences were statistically significant ($p < 0.05$). When comparing ES vs ES+BP we obtain that the differences were not statistically significant for the error–training ($p = 0.6305$) and error–test–5CV ($p = 0.9118$)

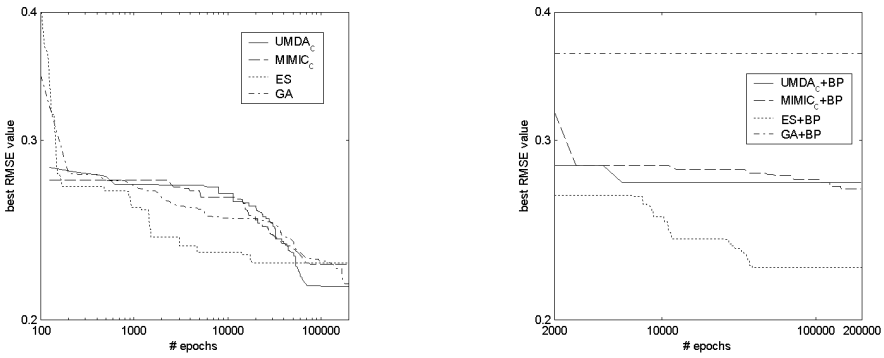

Figure 1.2 Convergence plot of different EAs on the KILN database.

parameters in the KILN database, maintaining the significativity in the differences ($p < 0.05$) in the rest of the databases and parameters.

In the line of the above remarks about parameterization, it is also interesting to consider the situation in which a larger number of RMSE calculations are allowed. To be precise, the convergence properties of any of these algorithms is concern arousing. A final experiment has been done to shed some light on this: the convergence in a long ($2 \cdot 10^5$ RMSE calculations) run of the different algorithms considered has been compared. The results are shown in Fig. 1.2. Focusing first in the leftmost plot (corresponding to *pure* evolutionary approaches) it is evident the superiority of ESs in the short term ($\leq 10^4$ RMSE calculations). In the medium term ($\sim 5 \cdot 10^4$ RMSE calculations), UMDA_c emerges as a competitive approach. In the long term ($\sim 10^5$ RMSE calculations), UMDA_c yields the best results, being the remaining techniques fairly similar in performance. From that point on, there is not much progress, except in the GA case, in which an abrupt advance takes place around $1.5 \cdot 10^5$ RMSE calculations. Due to this abruptness, it would be necessary to carry on additional tests to determine the likelihood of such an event.

The scenario is different in the case of the hybridized algorithms. These techniques seem to suffer from premature convergence to same extent (in a high degree in the case of the GA, somewhat lower in the case of the EDAs, and not so severely in the case of the ES). As a consequence, only ES and MIMIC_c can advance beyond the 10^4-RMSE-calculation point. In any case, and as mentioned before, more tests are necessary in order to obtain conclusive results.

4. Conclusions

This work has surveyed the utilization of EAs for supervised training in ANNs. It is a remarkable fact that EAs remain a competitive technique for this problem, despite their apparent simplicity. There obviously exist very specialized algorithms for training ANNs that can outperform these evolutionary approaches but, in the same line of reasoning, it is foreseeable that more sophisticated versions of these techniques could again constitute highly competitive approaches. As a matter of fact, the study of specialized models of EAs for this domain is a hot topic, continuously yielding new encouraging results, e.g., see (Castillo et al., 1999; Yang et al., 1999).

Future research can precisely be directed to the study of such sophisticated models. There are a number of questions that remain open. For example, the real usefulness of recombination within this application domain is still under debate. Furthermore, and granting this usefulness, the design of appropriate recombination operators for this problem is an area in which a lot of work remains to be done. Finally, the lack of theoretical support of some of these approaches (a situation that could alternatively be formulated as their excessive experimental bias) is a problem to whose solution many efforts have to be directed.

Acknowledgments

Carlos Cotta and Enrique Alba are partially supported by the Spanish *Comisión Interministerial de Ciencia y Tecnología* (CICYT) under grant TIC99-0754-C03-03.

Notes

1. Network weights comprise both the previously mentioned connection weights, as well as *bias* terms for each unit. The latter can be viewed as the weight of a constant saturated input the corresponding unit always receives.

2. Although it is possible to use real-number encodings in GAs, such models still lack the strong theoretical corpus available for ESs (Beyer, 1993; Beyer, 1995; Beyer, 1996). Furthermore, crossover is the main reproductive operator in GAs, so it is necessary to define sophisticated crossover operators for this representation (Herrera et al., 1996). Again, ESs offer a much simpler approach.

3. Some advanced ES models also include covariance values θ_{ij} to make all perturbations be correlated. We have not considered this possibility in this work since we intended to keep the ES approach simple. On the other hand, notice that the number of these covariance values is $O(n^2)$, where n is the number of variables being optimized. Thus, very long vectors would have been required in the context of ANN training.

4. $RMSE = \sqrt{\frac{2E}{mn_o}}$.

5. Of course, this also depends on the particular operators used in the algorithm. Recombination is a potentially disturbing operator in this sense. No recombination has been considered in these two models though.

References

Alander, J. (1994). Indexed bibliography of genetic algorithms and neural networks. Technical Report 94-1-NN, University of Vaasa, Department of Information Technology and Production Economics.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York.

Baluja, S. (1995). An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University.

Berlanga, A., Isasi, P., Sanchís, A., and Molina, J. (1999a). Neural networks robot controller trained with evolution strategies. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 413–419, Washington D.C. IEEE Press.

Berlanga, A., Molina, J., Sanchís, A., and Isasi, P. (1999b). Applying evolution strategies to neural networks robot controllers. In Mira, J. and Sánchez-Andrés, J., editors, *Engineering Applications of Bio-Inspired Artificial Neural Networks*, volume 1607 of *Lecture Notes in Computer Science*, pages 516–525. Springer-Verlag, Berlin.

Beyer, H.-G. (1993). Toward a theory of evolution strategies: Some asymptotical results from the $(1^+_,\lambda)$-theory. *Evolutionary Computation*, 1(2):165–188.

Beyer, H.-G. (1995). Toward a theory of evolution strategies: The (μ, λ)-theory. *Evolutionary Computation*, 3(1):81–111.

Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self adaptation. *Evolutionary Computation*, 3(3):311–347.

Castillo, P. A., González, J., Merelo, J., Prieto, A., Rivas, V., and Romero, G. (1999). GA-Prop-II: Global optimization of multilayer perceptrons using GAs. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 2022–2027, Washington D.C. IEEE Press.

Caudell, T. and Dolan, C. (1989). Parametric connectivity: training of constrained networks using genetic algoritms. In Schaffer, J., editor, *Proceed-*

ings of the Third International Conference on Genetic Algorithms, pages 370–374, San Mateo, CA. Morgan Kaufmann.

Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York.

Fogel, D., Fogel, L., and Porto, V. (1990). Evolving neural networks. *Biological Cybernetics*, 63:487–493.

Galić, E. and Höhfeld, M. (1996). Improving the generalization performance of multi-layer-perceptrons with population-based incremental learning. In *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 740–750. Springer-Verlag, Berlin.

Gallagher, M. R. (2000). *Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling*. PhD thesis, Department of Computer Science and Electrical Engineering, University of Queensland.

Gruau, F. and Whitley, D. (1993). Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1:213–233.

Herrera, F., Lozano, M., and Verdegay, J. (1996). Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convengence of real coded genetic algorithms. *Journal of Intelligent Systems*, 11:1013–1041.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Harbor.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico.

Larrañaga, P. (2001). A review on Estimation of Distribution Algorithms. In Larrañaga, P. and Lozano, J. A., editors, *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation*. Kluwer Academic Publishers.

Mangasarian, O. and Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18.

Maxwell, B. and Anderson, S. (1999). Training hidden Markov models using population-based learning. In Banzhaf, W. et al., editors, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, page 944, Orlando FL. Morgan Kaufmann.

McClelland, J. and Rumelhart, D. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. The MIT Press.

Montana, D. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767, San Mateo, CA. Morgan Kaufmann.

Moscato, P. (1999). Memetic algorithms: A short introduction. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill.

Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In H. M. Voigt, e. a., editor, *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer-Verlag, Berlin.

Nakai, K. and Kanehisa, M. (1992). A knowledge base for predicting protein localization sites in eukaryotic cells. *Genomics*, 14:897–911.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart.

Ribeiro, B., Costa, E., and Dourado, A. (1995). Lime kiln fault detection and diagnosis by neural networks. In Pearson, D., Steele, N., and Albrecht, R., editors, *Artificial Neural Nets and Genetic Algorithms 2*, pages 112–115, Wien New York. Springer-Verlag.

Rosenblatt, F. (1959). *Principles of Neurodynamics*. Spartan Books, New York.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by backpropagating errors. *Nature*, 323:533–536.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basel.

Silva, F. and Almeida, L. (1990). Speeding up backpropagation. In Eckmiller, R., editor, *Advanced Neural Computers*. North Holland.

Whitley, D. (1999). A free lunch proof for gray versus binary encoding. In Banzhaf, W. et al., editors, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 726–733, Orlando FL. Morgan Kaufmann.

Whitley, D. and Hanson, T. (1989). Optimizing neural networks using faster, more accurate genetic search. In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396, San Mateo, CA. Morgan Kaufmann.

Whitley, D., Mathias, K., and Fitzhorn, P. (1991). Delta coding: An iterative search strategy for genetic algorithms. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84, San Mateo CA. Morgan Kaufmann.

Whitley, D., Starkweather, T., and Bogart, B. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361.

Wienholt, W. (1993). Minimizing the system error in feedforward neural networks with evolution strategy. In Gielen, S. and Kappen, B., editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 490–493, London. Springer-Verlag.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Yang, J.-M., Horng, J.-T., and Kao, C.-Y. (1999). Incorporation family competition into Gaussian and Cauchy mutations to training neural networks using an evolutionary algorithm. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1994–2001, Washington D.C. IEEE Press.

Zhang, B.-T. and Cho, D.-Y. (2000). Evolving neural trees for time series prediction using Bayesian evolutionary algorithms. In *Proceedings of the First IEEE Workshop on Combinations of Evolutionary Computation and Neural Networks (ECNN-2000).*