Particle Swarms for Feedforward Neural Network Training

Rui Mendes
Dep. de Informática
Universidade do Minho
Braga - PORTUGAL
azuki@di.uminho.pt

Paulo Cortez
Dep. de Sist. de Informação
Universidade do Minho
Guimarães - PORTUGAL
pcortez@dsi.uminho.pt

Miguel Rocha
Dep. de Informática
Universidade do Minho
Braga - PORTUGAL
mrocha@di.uminho.pt

José Neves
Dep. de Informática
Universidade do Minho
Braga - PORTUGAL
jneves@di.uminho.pt

Abstract - *Particle Swarm* is a novel optimization paradigm for real-valued functions, based on the social dynamics of group interaction. In this work, it is proposed its application to the training of *Neural Networks*. Comparative tests were carried out, for *classification* and *regression* tasks, being the results compared with other approaches.

Keywords: Evolutionary Programming, Feedforward Neural Networks, Gradient-Based Algorithms, Particle Swarm Optimization.

I. INTRODUCTION

Since the early ages of *Artificial Intelligence (AI)*, in general, and the *Machine Learning (ML)* arena in particular, one has observed a trend to look at *Nature* for inspiration, when building problem solving models. In particular, the study of phenomena of *natural selection*, the *central nervous system* or *social behavior* has lead to the arise of the *Evolutionary*, *Neural* and *Swarm* computation fields.

Feedforward Neural Networks (FNNs) are one of the most popular *Artificial Neural Network* architectures, where neurons are grouped in layers and only forward connections exist. This provides a powerful connectionist model that can learn any kind of continuous nonlinear mapping, with successful applications such as *Time Series Forecasting*, *Medical Diagnostics* or *Handwritten Recognition*, just to name a few [8].

The interest in supervised learning to problem solving and *FNNs* was stimulated by the advent of the *BackPropagation* algorithm [17]; since then, several variants have been proposed, such as the *QuickProp* and the *RProp* [14]. However, these procedures are not free of escaping from local minima when the error surface is rugged. Moreover, most of these algorithms strongly depend on problem specific parameter settings.

The use of evolutionary search as an *ANN* learning method may overcome gradient-based handicaps, but convergence is in general much slower, since these are general purpose methods [2].

More recently, Kennedy and Eberhart [10] proposed a new technique for nonlinear optimization, called *Particle Swarm Optimization (PSO)*, which presents the advantages of being a very simple concept, requiring few computational costs. The authors argued that *PSO* could train *FNNs* with a performance similar to the *BackPropagation* method, for the *Xor* and *Iris* benchmarks. Since then, several researchers have adopted *PSO* for *FNN* learning [6], [1].

However, the comparison between *PSO* and other approaches seems scarce. Moreover, recent research in *PSO* [9], [11] suggests that the uniqueness of the algorithm lies in the dynamic interactions among the particles. Under this scenario, it seems natural to explore the use of different *PSO* topologies for *FNN* training.

The aim of this work is to study the benefits of *PSO* for the training of *FNNs*, when applied to *classification* and *regression* tasks. Furthermore, the results will be compared with the ones obtained by gradient-based algorithms (e.g., *BackPropagation*, *QuickProp* and *RProp*) and evolutionary approaches (e.g., *Evolutionary Programming*).

The paper is organized as follows: firstly, the basic concepts of the *PSO* paradigm and of the remaining learning models are defined; then, the learning benchmarks are presented; finally, the experiments and results are shown and discussed.

II. LEARNING MODELS

Three different models to approach *FNN* training will be considered, that is to say:

A. Particle Swarm Model

A *Particle Swarm Optimization (PSO)* algorithm is defined by the evolution of a population of particles, represented as vectors in a n-dimensional space. The particles' trajectories oscillate around a region, which is influenced by the individual's previous performance and by the success of his group of acquaintances.

Several methods have been proposed to describe each individual's group, (i.e., the *topology* or *social network*), being the *Gbest* and *Lbest*, the most common ones. In *Gbest*, all particles know each other, while in *Lbest*, each individual has only two neighbors, in a ring lattice (Figure 1).

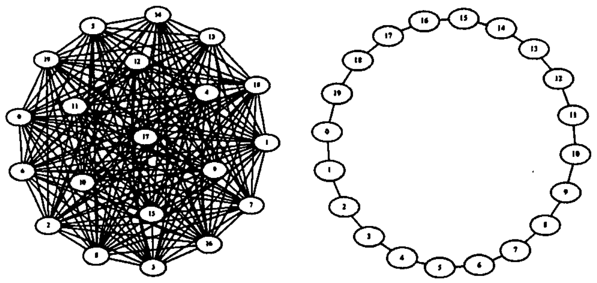

Fig. 1. *Gbest* and *Lbest* topologies.

In this work, three new hand-tuned topologies are proposed (Figure 2):

- *Square* : a graph where all nodes have a degree of four;
- *Pyramid* : a three-dimensional wire-frame triangle; and
- *4Clusters* : four *cliques*[1] of cardinality five, loosely connected, with two edges between neighboring clusters and one edge between opposite clusters.

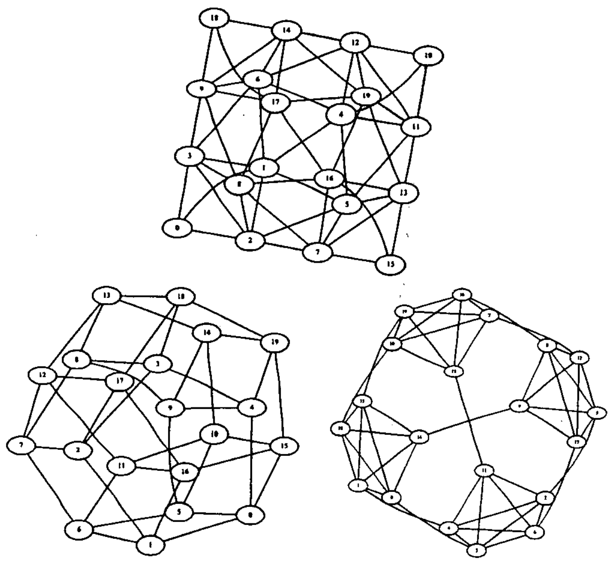

Fig. 2. The *Pyramid*, *Square* and *4Clusters* topologies.

Particle Swarms with 20 individuals and Type 1" constriction (recommended by [4]) were used:

$$\begin{aligned}\vec{v}_{t+1}^{(i)} &= \chi\,(\vec{v}_t^{(i)} + \varphi\,(\vec{U}[0,1]\,(\vec{p}_t^{(i)} - \vec{x}_t^{(i)}) \\ &\quad + \vec{U}[0,1]\,(\vec{p}_t^{(g)} - \vec{x}_t^{(i)}))) \\ \vec{x}_t^{(i)} &= \vec{v}_{t+1}^{(i)}\end{aligned} \qquad (1)$$

where $\varphi = 2.01$, $\chi = 0.729844$, $\vec{v}_t^{(i)}$ is the i particle's velocity at instant t, $\vec{p}_t^{(i)}$ is the position, $\vec{x}_t^{(i)}$ is the particle's current position corresponding to the particle's best experience, $\vec{p}_t^{(g)}$ is the group's best experience and $\vec{U}[0,1]$ is a vector, whose coordinates are random uniform variables within the range $[0,1]$.

When applied to *FNN* training, each particle represents a possible *FNN* configuration, i.e., its weights. Therefore, each vector has a dimension equal to the number of weights in the *FNN*, whose topology is kept fixed during the evolution.

B. Gradient-Based Models

Under this model, the learning is achieved by a single *FNN*, with a fixed problem dependent topology. The training is attained by a gradient-based algorithm. In this work, three different algorithms were considered, namely the standard *BackPropagation (BP)*, and two variants, the *QuickProp (QP)* and the *RProp (RP)*, being their parameters fixed to standard values ($\epsilon = 0.1$ for *BP*; $\epsilon = 0.1, \nu = 1.75$ for *QP*; and $\Delta_0 = 0.1, \Delta_{max} = 50.0$ for *RP*) [14].

C. Evolutionary Programming Model

In the past, evolutionary approaches to *FNN* training have been attempted. However, these have presented difficulties in fine tuning, being overtaken by other techniques, like the gradient descent methods (e.g., *Quickprop* [19]). Nevertheless, they have also shown some advantages, under domains where gradient information is difficult to obtain (e.g., *Recurrent Networks* [3]).

In this work, an evolutionary process is considered, being accomplished by *Evolutionary Programming (EP)* [7], where a population of *ps* real-valued chromosomes is evolving (in this case *ps* was set to 30), each coding the weights of an *FNN*.

In each iteration, 50% of the individuals are kept from the previous generation, being the remaining bred through the application of a *mutation* operator. The operator works by adding, to a given gene in a selected chromosome, a value taken from a narrow Gaussian distribution, with a zero mean (i.e., small perturbations will be preferred over larger ones).

The *selection* procedure is done by converting the fitness value of each chromosome into its ranking in the

[1]A *clique* is a subgraph where there is an edge between any two vertices.

population, and then applying a *roulette wheel* scheme. Other evolutionary approaches to *FNN* training have been attempted, considering the use of *crossover* operators. However, there is no clear advantage in using any of the proposed approaches, which although being more complex than the proposed one, do not appear to improve significantly on the results [2].

III. MACHINE LEARNING TASKS

The experiments that will be considered in this work endorse two types of problems, which encompass the majority of *ANNs'* applications: the *classification* and the *regression* ones.

The learning problems will also be classified in terms of their provenience: *artificial*, whose data is generated by a computer; and *real*, where the data available is taken from a physical phenomena [12].

In terms of the *Machine Learning* problems, three classification and two regression tasks were selected:

The N Bit Parity (NBP) - This is a famous nonlinear benchmark [14], being defined by 2^N patterns of N inputs and one output, which is set to the value 1, if the total number of input bits set to 1 is odd, and 0 otherwise. In the experiments, N will be set to 2 (also know as the *Xor* problem) and 4.

The Three Color Cube (TCC) - This is a simple artificial task that consists in learning how to paint a large 3D cube, made up by a 3x3 grid of blocks (27 smaller cubes) (Figure 3) [16]. Each smaller cube is represented by its coordinates on the X, Y and Z axis, that can take values from the set $\{-1, 0, 1\}$, and can be painted with three different colors: black, grey and white. The corners are black, the cubes in the center are white, being the others grey (Figure 3). In terms of the *ANN* training cases, 27 patterns are created, one for each cube, consisting of 3 inputs and 3 outputs (one for each color).

The Diabetes in Pima Indians (DPI) - This task consists in diabetes diagnosis (a boolean output) from seven input real variables (e.g., *number of pregnancies*). The data is defined by 200 samples, taken from a population of women of a Pima Indian heritage [15].

The Sin Times Sin (STS) - A nonlinear artificial function approximation, where 80 points were generated from two cyclic curves: $y = sin(8x) \times sin(6x)$ (Figure 4).

The Rise Time Servomechanism (RTS) - An extremely nonlinear phenomenon, where the goal is to predict the rise time of a servomechanism, based in 167 training instances, using two gain settings (integers) and two discrete choices of mechanical linkages [13].

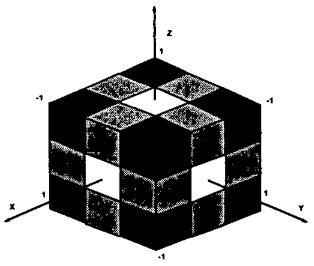

Fig. 3. The *Three Color Cube* problem.

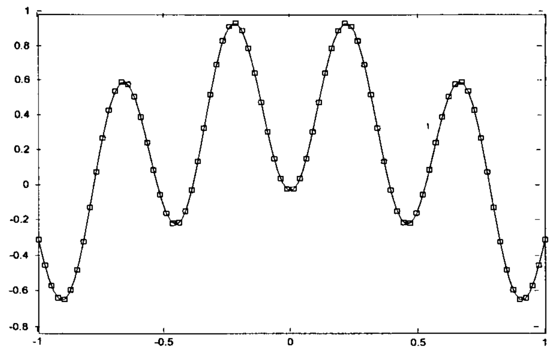

Fig. 4. The *Sin Times Sin* problem.

IV. EXPERIMENTS AND RESULTS

All experiments reported in this work were conducted using programming environments developed in *C/C++*, under the *Linux* operating system.

For all models, the initial weights were randomly assigned within the range $[-1; 1]$. The training accuracy of each *FNN* is measured in terms of the *Root Mean Squared Error (RMSE)*, according to the expression:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{p} \sum_{j=1}^{m} (T_{i,j} - F_{i,j})^2}{pm}} \quad (2)$$

where p denotes the number of the training patterns, m the number of the *FNN* outputs, $T_{i,j}$ the target and $F_{i,j}$ the actual value, both for the output j and the i input pattern.

This metric is used as the fitness value for each individual in the population based models (*EP* and *PSO*), being the overall accuracy given by the *RMSE* of the best individual.

Each problem was modeled with fully connected *FNNs*, with one hidden layer and *bias*, being the number of neurons in each layer given in Table I. The topologies were chosen in order to make the learning task difficult, by

TABLE I
NEURAL NETWORK ARCHITECTURES.

Task	Input	Hidden	Output
2BP	2	2	1
4BP	4	4	1
TCC	3	8	3
DPI	7	7	1
STS	1	8	1
RTS	4	4	1

adopting a minimum complexity, following an Occam's Razor principle.

The standard logistic activation function was used in all *classification* tasks. In the case of the *regression* problems, a different strategy was adopted, since outputs were unbounded. In this case, the *logistic* activation function was applied on the hidden nodes, while the output nodes used shortcut connections and *linear* functions, to scale the range of the outputs. This solution avoids the need of filtering procedures, which may lead to information loss (e.g., *rescaling*). On the other hand, it has been applied successfully in other situations, like *Time Series Forecasting* [5].

The results obtained are shown in Tables II and III, in terms of the mean and the respective 95% confidence intervals of thirty independent runs. All methods were stopped after 2000 iterations. The first five rows stand for *PSO* approaches, while the last four show the results for the remaining methods.

Concerning the comparison among the different *PSO* topologies, the *Lbest* presents the worst behavior, which is probably due to the low degree of connectivity, implying a slower convergence. The *Gbest* performs well on some benchmarks, although it finds difficulties in problems with several local minima (e.g., *NBP*). In such situations, the *Pyramid* seems to be a good alternative.

The relative ranks of each learning method are shown in Table IV, where the *PSO* results are taken from the best overall topology (*Pyramid*).

A closer look at the table reveals that the *QuickProp* yields good results in the classification tasks, but surprisingly shows poor performances on the regression ones. On the other hand, the *RProp* algorithm presents the best overall performance, although it fails in the *NBP* benchmarks, which can be taken as a sign of some weakness in problems with local minima. This fact is indeed quite normal since it is a gradient-based procedure. It is on these problems that the *PSO* shows its strength.

Furthermore, when compared to *EP*, *PSO* seems to be in a slight advantage, the same being true for a similar comparison with the standard *BackPropagation* method.

TABLE II
RESULTS FOR THE CLASSIFICATION TASKS.

	2BP	4BP	TCC	DPI
Gbest	0.04±0.00	0.14±0.01	0.26±0.02	0.25±0.02
Lbest	**0.00**±0.00	0.18±0.01	0.34±0.02	0.31±0.02
Pyramid	0.01±0.00	**0.11**±0.01	0.29±0.02	0.27±0.02
Square	**0.00**±0.00	0.13±0.01	0.29±0.02	0.28±0.02
4Clusters	**0.00**±0.00	0.14±0.01	0.30±0.02	0.28±0.02
EP	0.17±0.01	0.20±0.01	0.29±0.02	0.26±0.02
BP	0.48±0.03	0.50±0.03	0.32±0.02	0.18±0.01
QP	0.18±0.01	0.17±0.01	**0.19**±0.01	**0.16**±0.01
RP	0.06±0.00	0.22±0.01	0.21±0.01	0.18±0.01

TABLE III
RESULTS FOR THE REGRESSION TASKS.

	STS	RTS
Gbest	0.29±0.02	0.50±0.03
Lbest	0.32±0.02	0.54±0.04
Pyramid	0.31±0.02	0.48±0.03
Square	0.30±0.02	0.51±0.03
4Clusters	0.30±0.02	0.51±0.03
EP	0.32±0.02	0.54±0.04
BP	0.34±0.02	0.57±0.04
QP	0.34±0.02	1.73±0.11
RP	**0.23**±0.02	**0.45**±0.03

The performances were reported only over training sets, apparently disregarding overfitting. The two main approaches to tackle this issue are regularization (e.g., *early stopping*) and model selection (e.g., *BIC* criterion) [18]. This last alternative, avoids the need for validation sets, and has shown better results in previous work [5]. In this case, choosing the best topology and training are independent tasks, favoring algorithms that provide lower errors.

V. CONCLUSIONS AND FUTURE WORK

The surge of new optimization methods that handle real-valued functions is commonly followed by its application to *Artificial Neural Network* training. In such studies, it is usual to present comparisons of the newcomer only with the standard *BackPropagation*, although it is common knowledge in the field, that better algorithms

TABLE IV
TRAINING RANKS.

	2BP	4BP	TCC	DPI	STS	RTS
Pyramid	1	1	4	5	2	2
EP	3	3	3	4	3	3
BP	5	5	5	2	5	4
QP	4	2	1	1	4	5
RP	2	4	2	3	1	1

exist [14]. This work confirms this assumption, being clear that the *Rprop* is more robust.

It is obvious that using a more demanding comparison, makes it difficult to present outstanding results. Nevertheless, in some kinds of problems, the *PSO* showed to be valuable, namely in cases where a high number of local minima is known to exist.

This work is only meant as a starting point in this field and the results are encouraging, considering that *PSO* is a general purpose method, where no problem specific knowledge is used. Indeed, when compared to a similar method, the *EP*, its performance is promising.

In the future, a number of paths is to be followed:

- to enlarge the experiments domain, by looking at more real-world applications, such as those of *system's control*, *time-series forecasting* or *medical diagnosis*;
- to explore different *ANN* topologies, namely those where the gradient information is difficult to obtain (e.g. *Recurrent Neural Networks*); and
- to adopt *PSO* in the training of *ANNs* for reinforcement learning tasks, where the gradient-based algorithms are not applicable.

ACKNOWLEDGEMENTS

This work was supported by the PRAXIS project PRAXIS/P/EEI/13096/98.

REFERENCES

[1] F. Bergh and A. Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *SACJ/SART*, (26):84–90, 2000.

[2] J. Branke. Genetic algorithms for neural network design and training. In *Proceedings of the First Nordic Workshop on Genetic Algorithms*, pages 145–163, University of Vaasa, Finland, January 1995.

[3] T. Chin and D. Mital. An Evolutionary Approach to Training Feed-Forward and Recurrent Neural Networks. In L. C. Jain and R. K. Jain, editors, *Proceedings of the Second International Conference on Knowledge-Based Intelligent Electronic Systems*, pages 596–602, Adelaide, Australia, 1998.

[4] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergence in a multi-dimensional complex space. To appear in *IEEE Transactions on Evolutionary Computation*, 2002.

[5] P. Cortez, M. Rocha, and J. Neves. Evolving Time Series Forecasting Neural Network Models. In *Proceedings of International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models (ISAS 2001)*, Havana, Cuba, March 2001.

[6] R. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence PC Tools.* Academic Press, Boston, USA, 1996.

[7] L. J. Fogel. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming.* John Wiley, New York, 1999.

[8] S. Haykin. *Neural Networks - A Compreensive Foundation.* Prentice-Hall, New Jersey, 2nd edition, 1999.

[9] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Conference on Evolutionary Computation*, pages 1931–1938. IEEE Computer Society, 1999.

[10] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Western Australia, November 1995.

[11] J. Kennedy and R. Mendes. Topological Structure and Particle Swarm Performance. In *Proceedings of the Conference on Evolutionary Computatiton - CEC2002.* IEEE Computer Society, 2002.

[12] L. Prechelt. A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice. *Neural Networks*, 9, 1995.

[13] J. Quinlan. Combining instance-based and model-based learning. In P. E. Utgoff, editor, *Machine Learning - ML'93.* San Mateo: Morgan Kaufmann, 1993.

[14] M. Riedmiller. Supervised Learning in Multilayer Perceptrons - from Backpropagation to Adaptive Learning Techniques . *Computer Standards and Interfaces*, 16, 1994.

[15] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[16] M. Rocha, P. Cortez, and J. Neves. The Relationship between Learning and Evolution in Static and in Dynamic Environments. In C. Fyfe, editor, *Proceedings of the 2nd ICSC Symposium on Engineering of Intelligent Systems (EIS'2000)*, pages 377–383. ICSC Academic Press, July 2000.

[17] D. Rumelhart, G. Hinton, and R. Williams. Learning Internal Representations by Error Propagation. In D. Rulmelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume 1, pages 318–362, MIT Press, Cambridge MA, 1986.

[18] W. Sarle. Stopped Training and Other Remedies for Overfitting. In *Proceedings of the 27th Symposium on the Interface of Computer Science and Statistics*, pages 352–360, 1995.

[19] J. Schaffer, D. Whitley, and L. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In Whitley and Schaffer, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, June 1992.