# Evolutionary Neural Network Learning

Miguel Rocha[1], Paulo Cortez[2], and José Neves[1]

[1] Departamento de Informática - Universidade do Minho
Campus de Gualtar, 4710-057 Braga - PORTUGAL
[2] Departamento de Sistemas de Informação - Universidade do Minho
Campus de Azurém, 4800-058 Guimarães - PORTUGAL
mrocha@di.uminho.pt pcortez@dsi.uminho.pt jneves@di.uminho.pt

**Abstract.** Several gradient-based methods have been developed for *Artificial Neural Network (ANN)* training. Still, in some situations, such procedures may lead to local minima, making *Evolutionary Algorithms (EAs)* a promising alternative. In this work, *EAs* using direct representations are applied to several *classification* and *regression ANN* learning tasks. Furthermore, *EAs* are also combined with local optimization, under the *Lamarckian* framework. Both strategies are compared with conventional training methods. The results reveal an enhanced performance by a macro-mutation based *Lamarckian* approach.

## 1 Introduction

In *MultiLayer Perceptrons (MLPs)*, one of the most popular *Artificial Neural Network (ANN)* architectures, *neurons* are grouped in *layers* and only *forward connections* exist [2]. The interest in *MLPs* was stimulated by the advent of the *Backpropagation* algorithm and since then several variants have been proposed, such as the *RPROP* [7]. Yet, these gradient-based procedures are not free from getting trapped into local minima when the error surface is rugged, being also sensitive to their parameter settings and to the network initial weights.

An alternative approach comes from the use of *Evolutionary Algorithms (EAs)*, where a number of potential solutions to a problem makes an evolving population [5, 4]. *EAs* are appealing for *ANN* training since [8]: a global multi-point search is provided; no gradient information is required; and they are general purpose methods (the same *EA* may be used in different types of *ANNs*).

Following this trend, this work aims at exploring the use of *EAs* for *MLP* training, when applied to *classification* and *regression* tasks.

## 2 Experimental Setup

A set of ten benchmarks was considered in this work (Table 1), endorsing two main types (column **T**) of problems: *Classification (C)* and *Regression (R)* tasks. Six real-world problems were chosen from the *UCI* machine learning repository [3]. The *PRA* is based on a realistic simulation of the dynamics of a robot arm.

The artificial tasks include the famous *N Bit Parity* [7], the *TCC* which consists on assigning one of three colors to each block of a 3x3x3 grid cube and the *STS*, a regression task where the output is given by: $y = sin(8x) \times sin(6x)$.

**Table 1.** The *MLP* learning benchmarks.

| Task | T | Description | C | I | H | O | W |
|---|---|---|---|---|---|---|---|
| *6BP* | C | Six Bit Parity | 64 | 6 | 6 | 1 | 49 |
| *TCC* | C | Three Color Cube | 27 | 3 | 8 | 3 | 59 |
| *SMR* | C | Sonar: Rocks vs Mines | 104 | 60 | 6 | 1 | 373 |
| *PID* | C | Pima Indians Diabetes | 200 | 7 | 7 | 1 | 64 |
| *IPD* | C | Iris Plant Database | 150 | 4 | 3 | 3 | 27 |
| *WBC* | C | Wisconsin Breast Cancer | 499 | 9 | 3 | 1 | 34 |
| *STS* | R | Sin Times Sin | 80 | 1 | 8 | 1 | 25 |
| *PRA* | R | Pumadyn Robot Arm | 128 | 8 | 8 | 1 | 81 |
| *RTS* | R | Rise Time Servomechanism | 167 | 4 | 4 | 1 | 25 |
| *PBC* | R | Prognostic Breast Cancer | 198 | 32 | 4 | 1 | 137 |

Each problem will be modeled by a fully connected *MLP*, with one hidden layer and *bias* connections, being the topology given in Table 1, where columns **I**, **H** and **O** denote the number of input, hidden and output nodes, while column **W** shows the number of connections. Classification tasks make use of a single binary output (if two classes are present) or one boolean value per each class. In *regression* problems one real-valued output encodes the *dependent* variable. The standard logistic activation function ($\frac{1}{1+e^{-x}}$) was used in all *classification* tasks. A different strategy was adopted for the *regression* problems, since outputs may lie out of the co-domain ($[0, 1]$). Thus, the *logistic* function was adopted on the hidden nodes, while the output ones used shortcut connections and *linear* functions, to scale the range of the outputs. For all training methods, the initial weights are randomly assigned within the range $[-1; 1]$, being the accuracy of each *MLP* measured in terms of the *Root Mean Squared Error (RMSE)*.

## 3 Experiments with Evolutionary Algorithms

In this study, *direct* encoding is embraced (one gene per connection weight), an alternative closer to the phenotype, allowing the definition of richer genetic operators [5]. Two mutation operators were used, namely:

- *Random Mutation*, which replaces one weight by a new randomly generated value, within the range $[-1, 1]$; and
- *Gaussian Mutation*, which adds to a given gene a value taken from a gaussian distribution, with a zero mean and 0.25 standard deviation [4].

In both cases, a random number of genes is changed, between 1% to 20% of the number of *ANN* weights. The following crossover operators were also tested:

- *Two-Point*, *Uniform*, *Arithmetical* and *Sum*, standard *EA* operators [5];
- *Input* and *Output* connections, similar to a *one-point* crossover except that the set of input (output) connections to a node can not be disrupted [6]; and
- *Hidden* nodes, that combines the previous two operators; i.e., all connections to/from a hidden node can not be separated.

The *EAs* population size was set to 30, being the *selection* done by converting the fitness value (*RMSE*) into its ranking, and then applying a roulette wheel scheme, being used a substitution rate of 50% and the *elitism* value set to one. All tests were conducted using the *Java* language, running on a *Pentium III 933 MHz PC*. The termination criteria was set by CPU time (100 seconds).

The results are compiled in Table 2, which shows the quality $Q_m$, measured by how far (in percentage) its error ($RMSE_{m,t}$, the mean of thirty runs for the model $m$ and task $t$) is from the best result ($B_t$), given by: $Q_m = 100 \times (\sum_{t \in T} \frac{RMSE_{m,t}}{B_t} - 1)$ where $B_t = min_{k \in M}(RMSE_{k,t})$, and $T$ and $M$ denote the set of learning *tasks* and *models*. In the first row, only the mutation operator is applied. For the others, each operator breeds 50% of the offspring. The best performance is achieved by *gaussian* mutation, being no gain in using a crossover operator, thus favoring *Evolutionary Programming* [4]. This may be due to the *permutation* problem; i.e., several genomes may encode the same *ANN* [8].

**Table 2.** The overall *EA's* results for each model $m$ ($Q_m$ values).

| Crossover | Gaussian Mutation | Random Mutation |
|---|---|---|
| *None* | **2.1%** | 148.2% |
| *Two-Point* | 9.8% | 143.6% |
| *Uniform* | 10.3% | 143.4% |
| *Arithmetical* | 24.3% | 146.0% |
| *Sum* | 74.4% | 78.3% |
| *Input* | 9.3% | 143.7% |
| *Output* | 7.8% | 143.5% |
| *Hidden* | 9.5% | 143.5% |

## 4 Experiments with Lamarckian Optimization

The *EAs* performance can be improved by the use of the *Lamarckian* point of view [1]: in this work and in every generation, each individual is subject to 50 epochs of the *RPROP* algorithm [7], being the new weights encoded back into the chromosome (Figure 1). Two distinct *Lamarckian EAs (LEAs)* were tested (Table 3), with 20 individuals and one mutation operator, *gaussian* (column **GL**) or *random* (column **RL**), since the crossovers revealed poor performances. Here the comparison favors the latter *macro-mutation*, which may allow individuals to

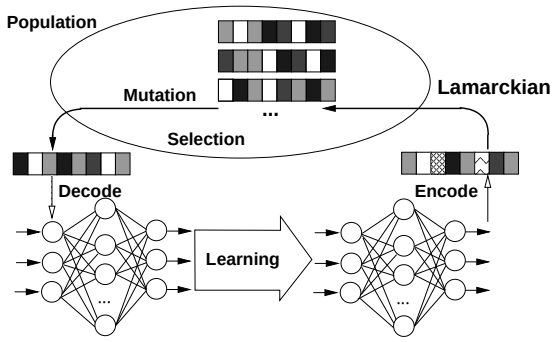


**Fig. 1.** An illustration of the Lamarckian strategy of inheritance.

jump between local minima, while the *gaussian* mutation effect may be reversed by the *RPROP*.

Table 3 also compares the best *EAs* with gradient-based methods (values are presented in terms of the mean of thirty runs). The *Neural Population (NP)* model was added, where 20 *MLP's* will be trained via the *RPROP* algorithm, in order to achieve a fair comparison among *population* and *non-population* approaches. The *BackPropagation (BP)* is outperformed by the *gaussian* mutation *EA* in four benchmarks, while the *RPROP (RP)* always surpasses the *EA*. The *NP* behaves better, although the *RL* excels all methods, stressing the importance of the random mutation and selection operators.

A temporal perspective is given in Figure 2 for the *TCC* task, reflecting each methods' traits: the *EA* and *BP* show slow learning rates; the *RP* gets the fastest convergence, but it quickly stagnates; both the random mutation *LEA* and *NP* reveal better long term performances, albeit the former method gains an advantage.

**Table 3.** Comparison between different training approaches ($RMSE$ values).

| Task | EA | GL | RL | NP | RP | BP |
|---|---|---|---|---|---|---|
| *6BP* | 0.148 | 0.078 | **0.036** | 0.070 | 0.243 | 0.364 |
| *TCC* | 0.216 | 0.113 | **0.069** | 0.101 | 0.194 | 0.201 |
| *SMR* | 0.153 | **0.000** | **0.000** | **0.000** | 0.067 | 0.045 |
| *PID* | 0.262 | 0.144 | **0.143** | 0.151 | 0.175 | 0.164 |
| *IPD* | 0.081 | 0.045 | **0.030** | 0.040 | 0.064 | 0.088 |
| *WBC* | 0.131 | **0.094** | **0.094** | 0.099 | 0.107 | 0.104 |
| *STS* | 0.329 | 0.095 | **0.078** | 0.109 | 0.095 | 0.299 |
| *PRA* | 1.190 | 0.420 | **0.390** | 0.420 | 0.440 | 1.780 |
| *RTS* | 0.571 | 0.266 | **0.242** | 0.254 | 0.381 | 0.523 |
| *PBC* | 26.1 | 19.8 | **19.0** | 21.5 | 21.9 | 38.6 |

## 5 Conclusions

Results obtained by pure *EAs* stress the importance of the gaussian mutation and the difficulty in the design of crossover operators. Although other methods are more effective in supervised tasks, this approach can be quite useful for *recurrent neural networks* or *reinforcement learning*. For *classification* and *regression*, the experiments carried out have shown that the *RPROP* algorithm is the best choice when few computational resources are available. However, a better performance is achieved by the use of a *Lamarckian* approach, being shown that incorporating a macro-mutation is essential to obtain improved performances.

## References


1. R. Belew, J. McInerney, and N. Schraudolph. Evolving Networks: Using the Genetic Algorithms with Connectionist Learning. CSE TR CS90-174, UCSD, 1990.
2. C. Bishop. *Neural Networks for Pattern Recognition*. Oxford Univ. Press, 1995.
3. C. Blake and C. Merz. UCI Repository of Machine Learning Databases, 1998.
4. L. J. Fogel. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. John Wiley, New York, 1999.
5. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, USA, third edition, 1996.
6. D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proc. 11th IJCAI*, pages 762–767. Morgan Kaufmann, 1989.
7. M. Riedmiller. Supervised Learning in Multilayer Perceptrons - from Backpropagation to Adaptive Learning Techniques . *Comp. Stand. and Interfaces*, 16, 1994.
8. X. Yao. Evolving Artificial Neural Networks. *Proc. IEEE*, 87(9):1423-1447, 1999.


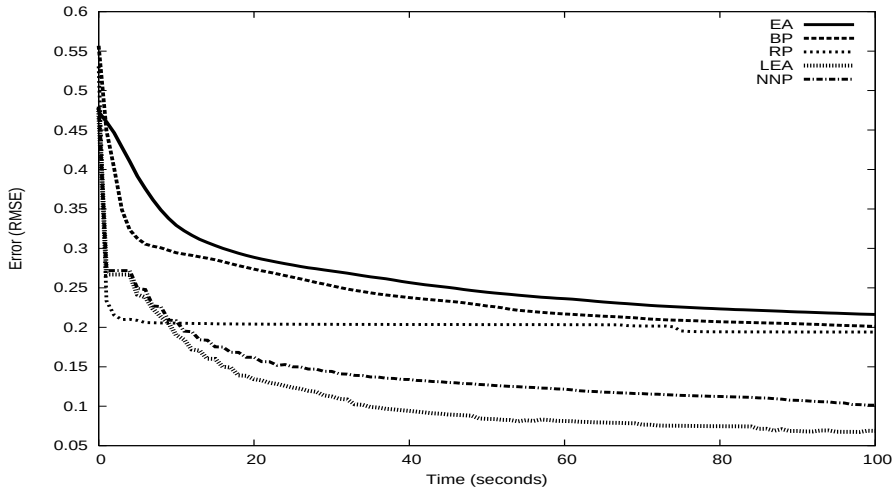


**Fig. 2.** The error evolution for the *TCC* task.