A Lamarckian Approach for Neural Network Training

Paulo Cortez, Miguel Rocha, and José Neves
Departamento de Informática, Campus de Gualtar,
Universidade do Minho, 4710-057 Braga, PORTUGAL
pcortez@di.uminho.pt mrocha@di.uminho.pt jneves@di.uminho.pt

Abstract. In Nature, living beings improve their adaptation to surrounding environments by means of two main orthogonal processes: *evolution* and *lifetime learning.* Within the *Artificial Intelligence* arena, both mechanisms inspired the development of non-orthodox problem solving tools, namely: *Genetic and Evolutionary Algorithms (GEAs)* and *Artificial Neural Networks (ANNs).* In the past, several gradient-based methods have been developed for *ANN* training, with considerable success. However, in some situations, these may lead to local minima in the error surface. Under this scenario, the combination of *evolution* and *learning* techniques may induce better results, desirably reaching global optima. Comparative tests that were carried out with *classification* and *regression* tasks, attest this claim.

Keywords: feedforward neural networks, genetic and evolutionary algorithms, hybrid systems, lamarckian optimization, learning algorithms

1. Introduction

The remarkable adaptation of some living creatures to their environments comes as a result of the interaction of two processes, working at different time scales: *evolution* and *lifetime learning. Evolution* is a slow stochastic process that takes place at the population level and determines the basic structures of an organism. *Lifetime learning* works by tuning up the structures of an individual, by a process of a gradual improvement of the individual's capability of adaptation to its surrounding environment.

Several breakthroughs in the *Artificial Intelligence* arena have occurred, namely in terms of new optimization procedures, based on analogies with the evolution of natural living systems. Problem solving techniques, such as the *Artificial Neural Networks (ANNs)* and *Genetic and Evolutionary Algorithms (GEAs)*, have already on their shoulders interesting results on a broad set of scientific and engineering tasks, such as *Combinatorial* and *Numerical Optimization*, *Pattern Recognition*, *Computer Vision* or *Robotics.* In terms of a computational procedure, *evolution* seems suitable for global search, while *learning* should be used to perform local search.

Feedforward Neural Networks (FNNs) are one of the most popular *ANN* 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. The interest in supervised learning to problem solving and *FNNs* was stimulated by the advent of the *Backpropagation* algorithm [20]; since then several variants have been proposed, such as the *Quickprop* and the *RPROP* [17]. However, these procedures are not free from escaping to local minima when the error surface is rugged. Moreover, most of these algorithms strongly depend on problem specific parameter settings.

On the other hand, *GEAs* are suited for combinatorial tasks, where the exhaustion of all possible solutions requires a huge computation. *GEAs* perform a global multi-point search, quickly locating high quality solutions, being able to escape from undesired local minima. *GEAs* and *ANNs* have been combined in three major ways: to set the weights in fixed architectures, to learn neural network topologies, and to select training data for *ANNs* [24].

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. In the past, *GEAs* have presented difficulties in fine tuning, being overtaken by other techniques, like the gradient descent methods (e.g., *Quickprop*) [22]. Nevertheless, *GEAs* have also shown advantages in training, under domains where gradient information is difficult to obtain (e.g., *Recurrent Networks* [3]).

Evolution and *learning* can be combined in two major ways, following the *Baldwin* [8] and *Lamarckian evolution* [1] approaches. Both processes use *lifetime learning* to accelerate *evolution*; their main difference is that the latter allows the inheritance of the acquired information.

The aim of this work is to study the benefits of the combination of *evolution* and *lifetime learning*, when applied to *Machine Learning* (e.g., *classification* and *regression* tasks). The combination of *evolution* and *learning* will be materialized via a *GEA*, where each individual codes for the weights of a *FNN*. The individuals are allowed to improve their fitness during lifetime, by a gradient descent process. The *Lamarckian* point of view will be adopted, since previous work favored this strategy under static environments [19].

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

2. Genetic and Evolutionary Algorithms

The term *Genetic and Evolutionary Algorithm (GEA)* is used to name a family of computational procedures where a number of potential solutions to a problem makes the way to an evolving population. Each individual codes a solution in a string (*chromosome*) of symbols (*genes*), being assigned a numerical value (*fitness*), that stands for a solution's quality measure. New solutions are created through the application of genetic operators (typically *crossover* or *mutation*). The whole process evolves via a process of stochastic selection biased to favor individuals with higher fitnesses.

The first *GEAs* [9], and most of the ones developed so far, make use of a binary representation; i.e., the solutions to a given problem are coded into a $\{0, 1\}$ alphabet. However, for larger problems, binary encodings result in very large strings which can slow down the evolutionary process. Some authors have argued that the best strategy is to represent weights directly into the chromosome, thus using a *Real-Valued Representation (RVR)*, which opens the way for richer genetic operators [25].

In this work, two genetic operators were adopted. Its picture is given below:

Two-point crossover - It works by selecting two random cutting points on a chromosome. The offspring are created by taking intermediate segments of one of the ancestors and inserting them into the other one [9].

Gaussian Perturbation - This is a *mutation* operator that adds, to a given gene, in a selected chromosome, a value taken from a Gaussian distribution, with a zero mean (i.e., small perturbations will be preferred over larger ones) [23].

3. Learning Models

Four different models will be defined to approach each learning task, that is to say:

The *Connectionist Model (CM)* - The learning is achieved by a single *FNN*, with a fixed problem dependent topology. The training is attained by the *RPROP* algorithm, chosen due to its faster convergence and stability in terms of problem's parameter adjustment.

The *Darwinian Model (DM)* - The learning process is accomplished by a *GEA* as described in Section 2, where a population of n real-valued chromosomes is evolving (in this case n was set to 20), 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 genetic operators (*crossover* or *mutation*). The *crossover* operation is responsible for breeding 75% of the offspring and the *mutation* operation is accountable for the remaining ones. The *selection* procedure is done by converting the fitness value of each chromosome into its ranking in the population, and then applying a roulette wheel scheme.

The *Lamarckian Model (LM)* - It combines both *lifetime learning* and *evolution*, making use of *GEAs*, as the main engine, and gradient-based *FNN* training methods, for local search (in this case, 20 epochs of the *RPROP* algorithm). The *ANN's* improved weights are encoded back into the chromosome (Figure 1).

The *Population of Connectionist Models (PM)* - This approach is added with the purpose of achieving a fair comparison among the different models so far presented, in particular to measure the weight of the genetic operators in the learning process. A population of 20 *ANN's* will improve only via the learning algorithm (*RPROP*); i.e., no genetic or selection procedure is applied.

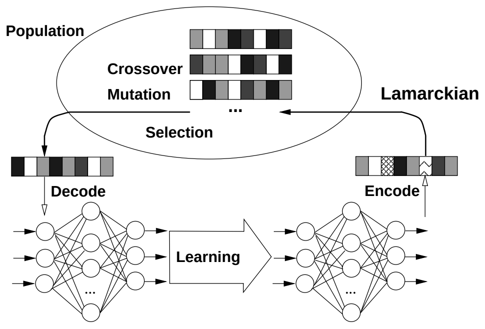

Figure 1. An illustration of the Lamarckian strategy of inheritance.

For all models, the initial weights are 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:

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

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 *DM* and *LM* models. For the population based models (*DM*, *LM* and *PM*), the overall accuracy is given by the *RMSE* of the best individual.

4. Machine Learning Tasks

The experiments that will be considered in this work endorse two main types of problems, which encompass the majority of *ANNs'* applications: the *classification* and the *regression* ones. The former requires a correct association between input patterns and output classes (e.g., classifying cells for cancer diagnosis). The latter deals with a functional approximation between n-dimensional input vectors and m-dimensional output ones (e.g., bank creditworthiness prediction). The main difference will be set in terms of the output representation. The outputs in *classification* tasks will be given by one boolean value for each possible class, while in *regression* problems, a real-valued output will code each of the m variables.

The learning problems will also be classified in terms of their provenience: *artificial*, whose data is generated by a computer; *realistic*, although artificially generated, it is modeled with attributes found in real data (e.g., with added noise); and *real*, where the data available is taken from a physical phenomena. Prechelt [15] argued that a minimum of two realistic or real problems from different domains should be adopted when comparing learning algorithms, since *realistic* tasks give a better representation of real world applications, while *real* phenomena may contain surprising or unknown features.

In terms of the *classification* problems, two synthetic and two real cases were selected:

The *N Bit Parity (NBP)* - This is a famous benchmark [17], being defined by 2^N patterns of N inputs (N was set to 6 in the experiments) 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.

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 2) [19]. 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 2). 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).

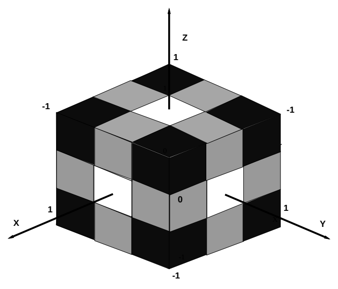

Figure 2. The *Three Color Cube* problem.

The *Sonar: Mines vs Rocks (SMR)* - The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced of a roughly cylindrical rock [7]. The data has 104 training cases with 60 real inputs and one boolean output.

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 [18].

Four *regression* tasks, ranging from different fields, were also contemplated:

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 3).

The *Pumadyn Robo Arm (PRA)* - A realistic simulation of the dynamics of a Puma 560 robot arm, where the angular acceleration of one robot arm is predicted from eight inputs (e.g., angular positions and velocities) [4]. The data consists in 128 training cases, taken from a nonlinear simulation with moderate noise.

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 [16].

The *Prognostic Breast Cancer (PBC)* - The aim is to predict the cancer to recur or disease-free time (in months), using 32 features extracted from a breast mass (e.g., texture or tumor size). The data is defined by 198 training samples taken from a Wisconsin database [12].

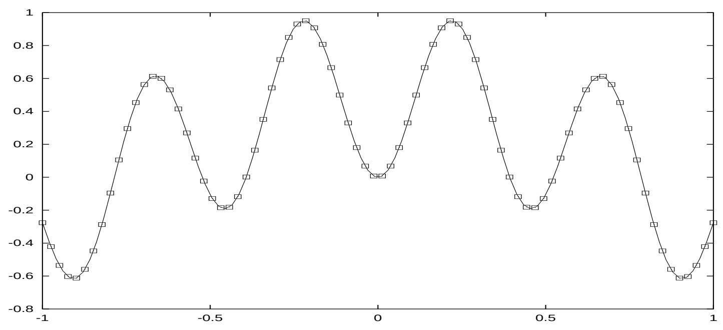

Figure 3. The *Sin Times Sin* problem.

5. Experiments and Results

A set of experiments were conducted, in order to evaluate the performance of each of the given models in the learning tasks. The results obtained are compared in terms of two orthogonal parameters: the overall learning process *accuracy* and the process *efficiency*, measured by the time elapsed. The termination criteria for the experiments was defined by a bound on CPU time, adjusted to the complexity of each task.

All experiments reported in this work were conducted using programming environments developed in *C++* [14], under the *Linux* operating system running on a *PC* with a *Pentium II 350 MHz* processor. For all case studies, at each time slot, it were considered the average of the results obtained in thirty independent runs.

Table I. Neural network architectures (number of nodes per layer).

Task	**Input**	**Hidden**	**Output**
6BP	6	6	1
TCC	3	6	3
SMR	60	6	1
DPI	7	7	1
STS	1	8	1
PRA	8	8	1
RTS	4	4	1
PBC	32	4	1

Each problem was modeled with fully connected *FNNs*, with one hidden layer and *bias* connections, being the number of neurons in each layer given in Table I. The topologies were chosen in order to make the learning of the task possible with a minimum complexity, following the Occam's Razor principle.

The standard logistic activation function was used in all *classification* tasks. In the case of the *regression* problems, the logistic function, adapted to a $[-1, 1]$ codomain, was used on the *STS* task. For the other ones, 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 give rise to loose information (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 Figures 4 and 5, where the evolution of the *RMSE* is plotted, in terms of *CPU* time (in seconds). More accurate information on the final results is presented in Table II. An analysis of the results shows that the *LM* outperforms the other models in both *classification* and *regression* tasks, being the most accurate model in the long term. The *CM* shows a faster convergence, presenting itself as the best choice when few computational time is available. However, after the initial stages, it looks as being trapped in local minima.

The results obtained by the *PM* make clear the importance of the genetic recombination. In fact, it is not enough to introduce diversity into the population to escape from local minima. The behavior of the *PM* was even worse than that of the *CM* one, due to the sharing of computational resources between the several *ANNs*. Therefore, the combination of *lifetime learning* and *evolution* in *ANN's* training may exceed the contribution of the sum of its parts.

Table II. Best learning results.

Task	**CM**	**DM**	**LM**	**PM**
6BP	0.236	0.315	**0.109**	0.247
TCC	0.280	0.339	**0.226**	0.248
SMR	0.051	0.378	**0.014**	0.314
DPI	0.171	0.340	**0.142**	0.192
STS	0.299	0.375	**0.277**	0.319
PRA	0.946	2.23	**0.784**	1.08
RTS	0.359	0.582	**0.261**	0.418
PBC	22.2	29.4	**17.9**	23.2

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) [21]. 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.

6. Conclusions and Future Work

Some work in this arena has already been put forward, where similar models have been compared and good results have been presented for *Lamarckian* approaches. Belew et al. [2] reported success with the *Lamarckian evolution*, although only one problem was considered for the experiments. Furthermore, the evolution process was based on a *GEA* with a binary encoding and the learning was performed by the standard *Backpropagation* algorithm. Other *ANN* architectures have also been considered. Ku et al. [11] have developed similar work in the context of *Recurrent Neural Networks*, where it is believed that the training is more prone to local minima. *Radial Basis-Function Networks* have also been focused in [6].

However, some of these studies consider only the benefits of *lifetime learning*, and the tradeoff between benefits and costs is rarely considered [13]. In the present work, the comparisons between the models is made by considering the CPU time, so that they can be fair.

On the other hand, Kitano's experiments [10] suggested that hybrid *GEA/ANN* approaches do not provide advantages over randomly initialized multiple applications of a fast gradient descent algorithm (e.g., *Quickprop*). However, experiments carried out in this study have shown that a better accomplishment can be achieved by a *Lamarckian* approach (the *RPROP* algorithm is considered to have an equal or faster convergence than *Quickprop* [17]).

The results do support the idea that the *Lamarckian* evolution of learning entities makes itself a very interesting method for *Machine Learning*. It is also hard not to fall into the temptation of comparisons with natural systems, where Lamarckian evolution is not the prevailing rule. Yet, it may be referred that, due to the complexity of the natural embryogenetic processes, a recoding of the genetic information after *lifetime learning* had occurred would be a difficult and costly task.

In the future one intends to enlarge the experiments domain, by looking at some real-world applications, such as those of *system's control*, *time-series forecasting* or *medical diagnosis*. Some of these problems are typically embedded in dynamic environments, where the learning tasks evolve over time. To cope with such features, some degree of adaptability is necessary, and it is believed that the *GEAs* can assume a decisive role in this process, namely when considering *Lamarckian* and *Baldwinian* learning models [19].

Acknowledgments

The work of Paulo Cortez was supported by the portuguese *Foundation for Science & Technology* through the grant PRAXIS XXI/BD/13793/97.

References

1. D. H. Ackley and M. L. Littman. *A case for Lamarckian evolution*, pages 3–10. Addison-Wesley, Reading, MA, 1994.
2. R. Belew, J. McInerney, and N. Schraudolph. Evolving Networks: Using the Genetic Algorithms with Connectionist Learning. CSE Technical Report CS90-174, Computer Science, UCSD, 1990.
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. P. Corke. A Robotics Toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, 1996.
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. C. Giraud-Carrier. Unifying Learning with Evolution Through Baldwinian Evolution and Lamarckism: A Case Study. In *Proceedings of the Symposium on Computational Intelligence and Learning (CoIL-2000)*, pages 36–41, 2000.
7. R. P. Gorman and T. J. Sejnowski. Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks*, 1:75–89, 1986.
8. G. Hinton and S. Nolan. How Learning Can Guide Evolution. *Complex Systems*, (1):495-502, 1987.
9. John Holland. *Adaptation in Natural and Artificial Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.
10. H. Kitano. Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms. In *Proceedings of the International Joint Conference on Neural Networks*, pages 397–404, IEEE, 1990.
11. Kim Ku, Man Mak, and Wan-Chi Siu. A study of the lamarckian evolution of recurrent neural networks. *IEEE Transactions on Evolutionary Computation*, 4(1):31–42, April 2000.
12. O. Mangasarian, W. Street, and W. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.
13. Gilles Mayley. The evolutionary cost of learning. In P. Maes, M. Mataric, J. A. Meyer, J. Pollack, and S. Wilson, editors, *From Animals to Animats 4: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior.* MIT Press, 1996.
14. J. Neves, M. Rocha, H. Rodrigues, M. Biscaia, and J. Alves. Adaptive Strategies and the Design of Evolutionary Applications. In W.Banzhaf, J.Daida, A.Eiben, M.Garzon, V.Honavar, M.Jakiela, and R.Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99)*, pages 473–479, Orlando, Florida, USA, July 1999. Morgan Kaufmann.

15. L. Prechelt. A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice. *Neural Networks*, 9, 1995.
16. J. Quinlan. Combining instance-based and model-based learning. In P. E. Utgoff, editor, *Machine Learning - ML'93*. San Mateo: Morgan Kaufmann, 1993.
17. M. Riedmiller. Supervised Learning in Multilayer Perceptrons - from Backpropagation to Adaptive Learning Techniques . *Computer Standards and Interfaces*, 16, 1994.
18. B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.
19. 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.
20. 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.
21. 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.
22. 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.
23. Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* Wiley, 1981.
24. D. Whitley. Genetic Algorithms and Neural Networks. In J. Perioux, G. Winter, M.Galan, and P.Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science.* John Willey & Sons Ltd., 1995.
25. B. Yooni, D. Holmes, G. Langholz, and A. Kandel. Efficient genetic algorithms for training layered feedforward neural networks. *Information Sciences*, 76:67–85, 1994.

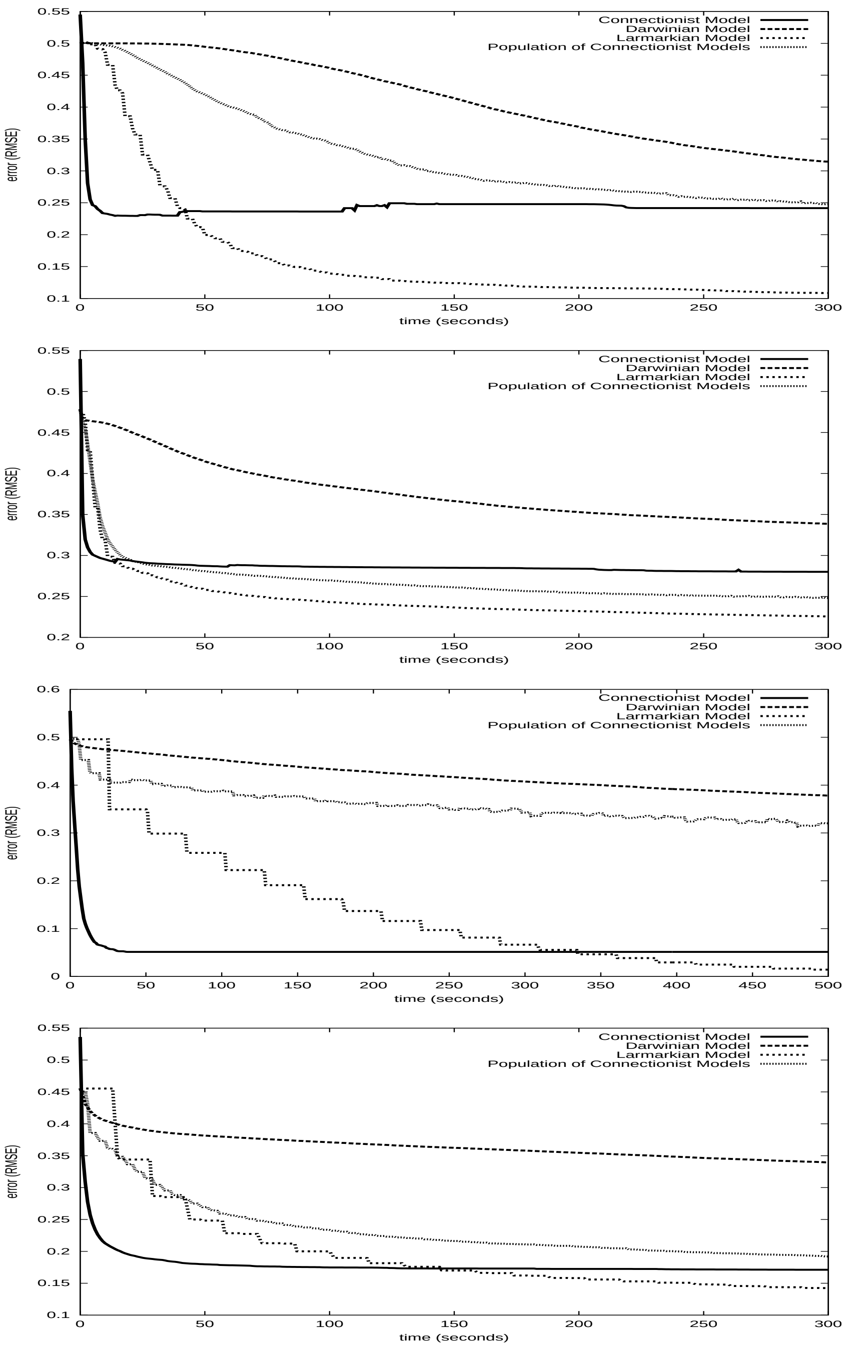

Figure 4. Results for the *classification* tasks (*6BP*, *TCC*, *SMR* and *DPI*).

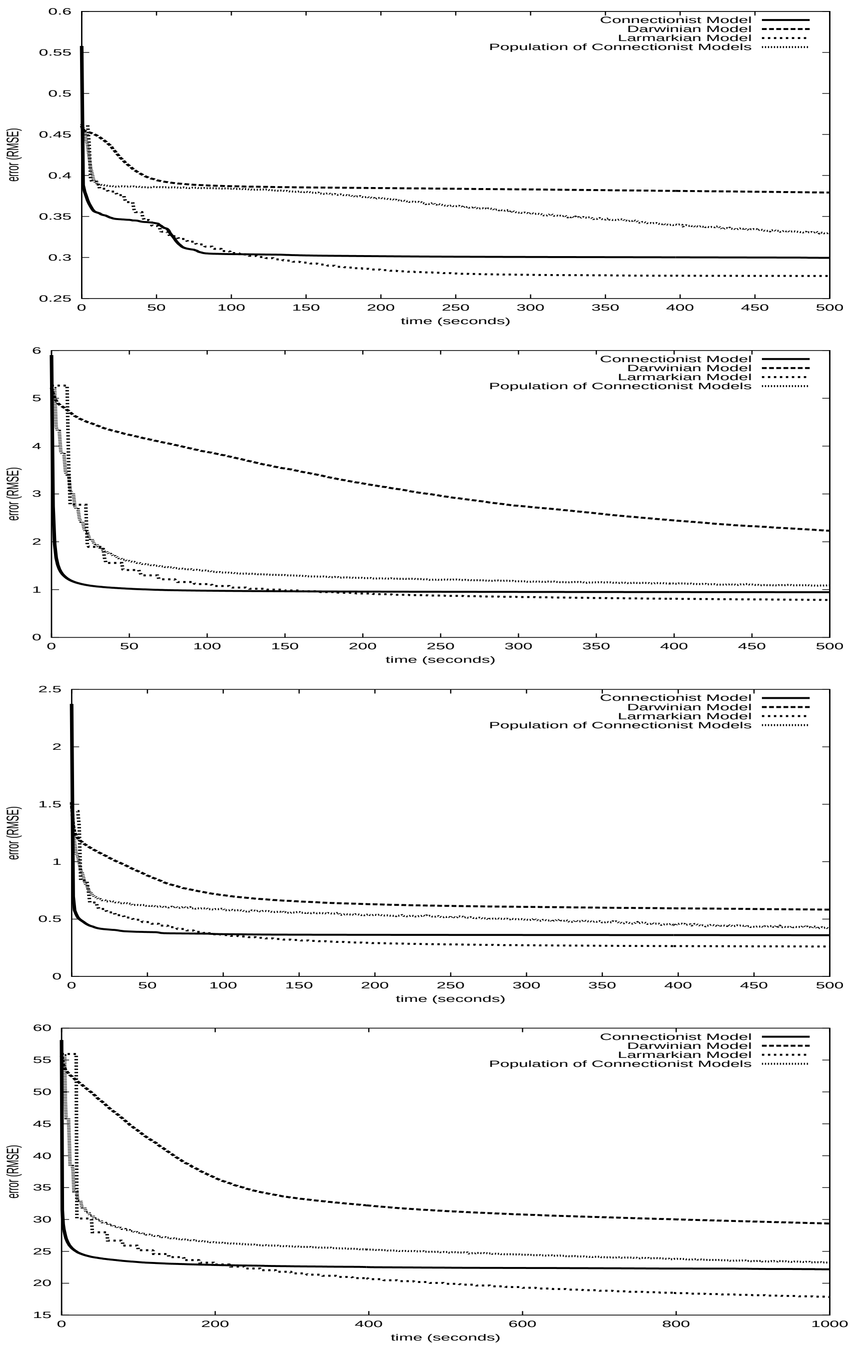

Figure 5. Results for the regression tasks (*STS*, *PRA*, *RTS* and *PBC*).