The Relationship between Learning and Evolution in Static and Dynamic Environments

Miguel Rocha, Paulo Cortez and José Neves

Departamento de Informática - Universidade do Minho
Largo do Paço, 4709 Braga Codex, PORTUGAL
{mrocha, pcortez, jneves}@di.uminho.pt

Abstract

Evolution and lifetime learning have been adopted by living creatures to get the best of the adaptation processes to natural environments. Within the *Machine Learning (ML)* arena such methods have been treated, particularly in the fields of *Genetic and Evolutionary Computation* and *Artificial Neural Networks*. Why not to combine both techniques, giving rise to several *ML* models, namely those based on *Lamarckian* or *Baldwinian* approaches? The results so far obtained point to better performances with the former ones under static settings, but reward the latter under dynamic environments, where the learning tasks change over time.

Keywords: Genetic and Evolutionary Algorithms, Artificial Neural Networks, Hybrid Systems, Lamarckian Optimization, Baldwin Effect.

1 Introduction

Since the early ages of the *Artificial Intelligence (AI)* field, 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 *Biology*, most scientists agree that the remarkable adaptation of some complex organisms to their environments comes as a result of the interaction of two processes, working at different time scales: evolution and lifetime learning. Evolution takes place at the population level and determines the basic structures of an organism. It is a slow process that works by stochastically selecting the better individuals to survive and to reproduce, and therefore works in order to propagate good features to the next generations. The lifetime learning is responsible for some degree of adaptation at the individual's level. It works by tuning up the structures, built in accordance with the genetic information, by a process of gradual improvement of the adaptation to the surrounding environment. In terms of a computational procedure, evolution seems suitable for global search, while learning should be used to perform local search.

In this work, the evolution process is represented by a *Genetic and Evolutionary Algorithm (GEA)*, while the lifetime learning one is approached by a gradient based training procedure to *Artificial Neural Networks (ANNs)*. The work to combine both strategies has been going on for more than a decade, and has been developed in several directions. In the past, combinations have been both *supportive* (i.e., the methods have been used sequentially), and *collaborative* (i.e., they have been used simultaneously). The former ones work by preparing data for consumption by the other, namely using a *GEA* to select features to be used by *ANNs'* classifiers. *Collaborative* combinations, on the other hand, use *GEAs* to evolve the *ANN's* connection strengths, the *ANNs* topology, or both [10].

One aims at studying the relationship between evolution and lifetime learning, when applied to a simple *ML* task. The purpose is to analyze potential synergetic effects when one combines both approaches, and to compare those models with simple learning or evolution procedures. The combination of *GEAs* and *ANNs* is materialized via an evolution of populations of individuals, each one coding an *ANN*. Each individual is allowed to improve its fitness during lifetime, by a gradient descent process. Two dif-

ferent frameworks are defined, named *Lamarckian* and *Baldwinian*, depending on the fact that the acquired information is recoded into the chromosome or not. These different models are compared both in static and dynamic environments; i.e., the *ML* task is changed at regular intervals in time.

The paper is organized as follows: firstly, one defines the basic concepts and the programming environments for the *ANNs* and *GEAs*; then, one defines the *ML* task used as benchmark; one follows with a description of the different models implemented; finally, the experiments are described and the results obtained are presented and discussed.

2 Artificial Neural Networks

2.1 Basic Concepts

Artificial Neural Networks (ANNs) are connectionist models that mimic the central nervous system of living species, acquiring knowledge from the environment through a learning process, in order to react to new situations, even when noise or incomplete data are present. An *ANN* is made up by simple processing units, called *neurons*, and interneuron synaptic strengths, known as *connection weights*, where the acquired knowledge is stored [2].

One can find a kaleidoscope of different *ANNs*, that diverge on several features, such as the training paradigm or the internal architecture. Each of these networks has some merits and demerits, and can be used to tackle different tasks. An important class of *ANNs* is the *MultiLayer FeedForward (MLFF)* one. In a *MLFF*, neurons are grouped in layers and only forward connections exist; i.e., one is not allowed to have cycles in the *ANN*. This kind of network is characterized to have one input and one output layer, and also one or more hidden layers, that provide high connectivity. This feature, in conjunction with nonlinear procedures (e.g. *Backpropagation*) provides a powerful tool, with successful applications ranging from computer vision, data analysis or expert systems, just to name a few.

2.2 The Artificial Neural Network Programming Environment

The *Artificial Neural Network Programming Environment (ANNPE)* was built with the purpose to increase productivity when developing applications with *ANNs*. It takes advantage on the features of the object-oriented paradigm, being developed in *C++*, allowing easy use and incremental development. One developed an hierarchy of classes just turn in to common sense, where the main one (also named *ANN*), is made of behaviors, interfaces (that must be defined in subclasses), and data structures, which are shared by all kinds of *ANNs*. *MLFF* networks are defined in a subclass, with several training algorithms being available, such as the standard *Backpropagation*, or the more sophisticated ones, such as *Quickprop* or *Rprop* [7].

3 Genetic and Evolutionary Algorithms

3.1 Basic Concepts

In this work, the term *Genetic and Evolutionary Algorithm (GEA)* is used to name a family of computational procedures that share a set of common features:

- there are a number of potential solutions (individuals) to a problem, evolving simultaneously (a population);
- each individual represents a solution to a problem, which is coded by a string (chromosome) of symbols (genes), taken from a well defined alphabet;
- the individuals are evaluated; i.e., to each of them is assigned a numeric value (fitness), that stands for a solution's quality or appropriateness metric;
- the solutions to the problem can be recombined and/or changed in some way, by using genetic operators (eg. crossover, mutation), in order to create new solutions (reproduction);
- the process is evolutionary; i.e., it is based on the *Darwinian* process of natural selection, where the fittest individuals have greater chances of surviving;

- its major structure is the one outlined in the pseudo-code of Figure 1.

BEGIN
  Initialize time ($t = 0$).
  Generate and evaluate the individuals in the initial population (P_0).
  WHILE NOT (termination criteria) DO
    Select from P_t, a number of individuals for reproduction.
    Apply to those individuals the genetic operators to breed the offspring.
    Evaluate the offspring.
    Select the offspring to insert into the next population (P_{t+1}).
    Select the survivors from P_t to be reinserted into P_{t+1}.
    Increase current time ($t = t + 1$).
END

Figure 1: Structure of a *GEA*

3.2 Real-valued Representations

The first *GEAs* [4], and most of the ones developed so far, make use of a binary representation; i.e., the solutions to a given problem are coded in a 0/1 alphabet. In the last few years some authors have argued that when one is faced with problems where the parameters are given by real values, the best strategy is to represent them directly in the chromosome [5], thus using a *Real-Valued Representation (RVR)*. These kind of representations allows for the definition of different operators. In this work, one developed three genetic operators, two of which are crossover ones; i.e., they take two individuals as inputs, and recombine their genetic information, by generating two new individuals (offspring), and the other one is a mutation; i.e., it takes one individual, and creates a new one by a small change in the genome. They can be described as follows:

- *One-point Crossover*: it is similar to its binary counterpart, so it works by randomly selecting a cutting point, collecting the genes in the first ancestor until the cutting point is reached, proceeding with the genes in the second parent from that point on, and in this way building a new individual. The inverse operation is used to breed the second offspring;
- *One-Gene Sum Crossover*: it randomly selects a gene in the chromosome and, for that position, the first offspring receives the sum of the values for that position in the ancestors, and the second receives the difference. The other genes are kept unchanged from each of the ancestors.
- *Gaussian Perturbation*: it is a mutation operator that adds, to a given gene, a value taken from a Gaussian distribution, with zero mean.

3.3 The Genetic and Evolutionary Programming Environment

The *Genetic and Evolutionary Programming Environment (GEPE)* [6] was built within the same spirit of its *ANNPE* counterpart; i.e., it aims to make easier the task of developing applications with *GEAs*. The framework developed is made of four main blocks (Figure 2), namely the *individuals*, the *populations*, the *GEAs* and the *evaluation module*. Each of these modules is materialized by an hierarchy of classes, that are built in such a way that the common attributes and behaviors are defined in the root classes, and a process of specialization is followed when one walks toward the leafs, by redefining or adding new attributes and/or behaviors.

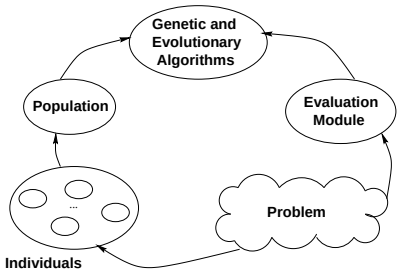

Figure 2: The GEPE's Archetype

At the *individual's* level, the root is an abstract class with a template field that contains its genotype; i.e., its genetic information. In this way, one sets the doings for any kind of representational scheme, simply by assignment of the template with the necessary data type (Figure 3). In order to implement the *RVR* under this setting one has the *RVRIndiv* class, where the operators described above are implemented. At the *population* and *GEA* levels, similar strategies are followed, allowing for the easy definition of default behaviors, but also for the possibility of redefining some parameters, such as the selection or the reinsertion processes, or the structure of the overall

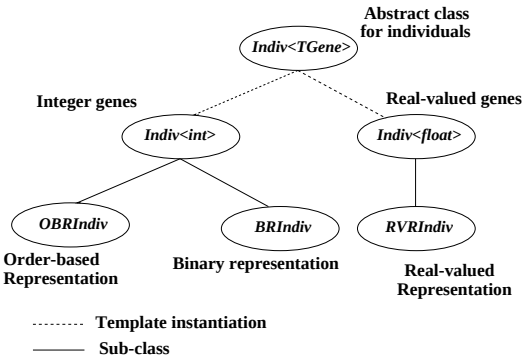

Figure 3: The Individuals hierarchy's class

algorithm. The last of the modules in the system is the *evaluation* one, where the programmer defines the decoding procedure; i.e., how to reach a solution to a given problem from the chromosome, and how to assign it a fitness value. A subclass to this module was created, in order to define the way one decodes a string of real values into an *ANN*, and the way each chromosome is evaluated in terms of the *ML* task.

4 Problem Formulation

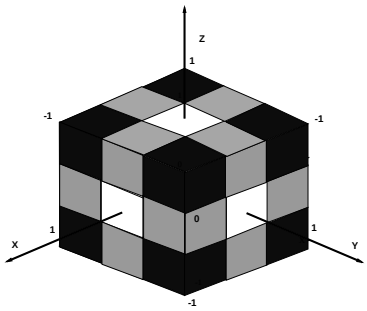

Figure 4: The *Color Cube Problem* in its static variant

The *Color Cube Problem (CCP)* is a simple *ML* task, that can be easily adapted to make dynamic environments. The task consists on learning how to paint a large 3D cube made up by a 3x3 grid of blocks (27 smaller cubes) (Figure 4). 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.

In the static variant, the corners are black (8 cubes), the cubes in the center are white (7 cubes), being the others grey (Figure 4). Two different approaches were followed when building the dynamic environments. In the former one (*E1*), the same discrimination rules prevail, although all the cubes are periodically repainted, following a predefined order (black follows grey, grey follows white and white follows black). In the latter case (*E2*), each cube is initially assigned a random color (black, grey or white), and then, periodically, the color of K cubes randomly chosen is changed.

In terms of the *ANN* training cases, 27 patterns are created, one for each cube, consisting of 3 inputs and 3 outputs. Each input stands for the node's coordinates, while each output represents a different color (100 for black, 010 for grey and 001 for white).

5 Learning Models

It were defined four different models to approach the *CCP*, namely:

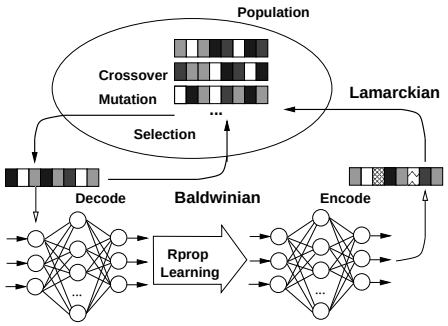

Figure 5: An illustration of the Baldwinian and Lamarckian strategies of inheritance

- The *Connectionist Model (CM)*. Under these circumstances the learning is achieved by a single *ANN*. The adopted *ANN* has a fixed topology, being a *MLFF* one with an input, an hidden and an output layer, with 3, 10 and 3 neurons, respectively. The activation function used was the logistic one, and the training is achieved by the *Rprop* algorithm, a more efficient variant of the well known *back-propagation* one [7]. The initial weights were randomly assigned within the range $[-1; 1]$.
- The *Darwinian Model (DM)*. In this approach, the overall learning process is accomplished by a *GEA*, where a population of 20 real-valued chromosomes is evolving, each coding the weights for

an *ANN* similar to the one described above. In each iteration, 40% of the individuals are kept from the previous generation, and 60% are generated by the application of the genetic operators described in Section 3.2. The fitness of each chromosome is calculated by an error metric, the *Root Mean Squared Error (RMSE)*, that ranges over all the 27 training patterns.

- The *Lamarckian Model (LM)*. The *LM* combines both lifetime learning and evolutionary approaches, making use of *GEAs* and *ANNs*. Similarly to the *DM*, the *GEA* is still the engine of the process, but in this case each individual is allowed to learn during its lifetime, in this case by running the *Rprop* algorithm for 20 epochs. Then, the improved weights are encoded back into the chromosome (Figure 5).

- The *Baldwinian Model (BM)*. In natural environments the process defined in the *LM* does not occur, or occurs with a negligible frequency. According to J.M. Baldwin [1], there is a synergetic effect when there is an evolving population of learning individuals; i.e., lifetime learning can accelerate evolution. This was known as the *Baldwin effect*, and is exploited in this model. The approach followed is close to the *LM* one, except that the lifetime learning is only used to improve the fitness of the individuals, and the new weights are not encoded back into the genome (Figure 5). This means that, in the process of reproduction, the offspring do not inherit the acquired genetic information from their ancestors.

6 Results

A set of experiments were conducted, in order to evaluate the performance of each of the given models in the *CCP*. This comparison took place under two different settings, the former considering a static environment, and the latter a dynamic one; i.e., the *ML* task evolves over time. The results obtained are compared in terms of two orthogonal parameters, the overall learning's *accuracy*, measured by the *RMSE* obtained for the set of patterns, and the process' *efficiency*, measured by the time elapsed (in seconds). When one considers a dynamic setting, it is also considered the robustness of the method; i.e., how the error metric evolves when the conditions suffer a sudden change. For all models, at each time/generation slot, one considers the average of the results obtained in ten independent runs. When applicable, one considered the best individual in the population.

Some work in this arena has already been put forward, where similar models have been compared. However, most of these studies consider only a subset of the given strategies, typically looking only at the *Baldwin effect* synergy [3], comparing the *LM* and the *BM* approaches [8], or simply showing the benefits of lifetime learning. Furthermore, some researchers have focussed primarily on these benefits, and the tradeoff between benefits and costs is rarely considered [9]. In the present framework, the comparisons between the models is made by considering the CPU time, so that they can be fair. Finally, most of the studies consider only static environments in their experiments, although some researchers have already focused on the dynamic case [8].

6.1 Experiments with a Static Environment

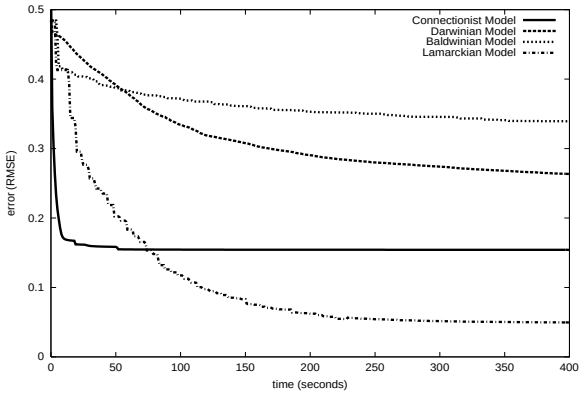

Figure 6: Results for the experiment in a static environment (time elapsed)

Initially, the four models were compared, in terms of their aptitude to learn the *CCP*, in the original static version. The evolution of the *RMSE*, for each model, was measured both in terms of the CPU time (Figure 6) and of the generations elapsed (Figure 7). An analysis of the results shows that the *LM* behaves in a better way, although the *CM* achieves a faster convergence in the initial stages, but looks as being trapped in a local minima. Therefore, the combination of lifetime learning and evolution may exceed the sum of its parts. The *DM* and the *BM* are much slower in their convergence under an acceptable margin of error. In Figure 7 one can observe the *Baldwin effect*; i.e., there are some evolutionary benefits taken

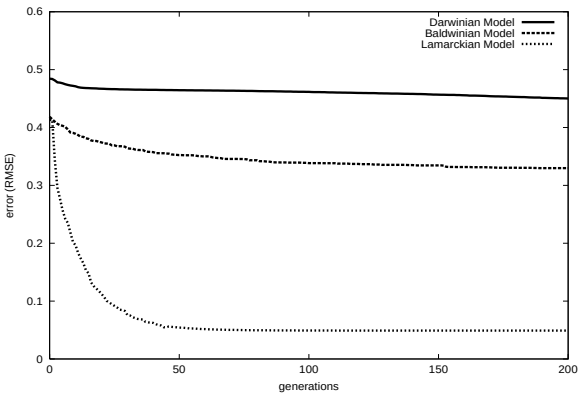

Figure 7: Results for the experiment in a static environment (by generations)

from the lifetime learning process. However, when the computational costs are taken into account (Figure 6), the *DM* still makes better use of the resources available.

6.2 Experiments with a Dynamic Environment

The previous results can be intuitively explained; the *LM* presents itself as more effective due to a cumulative effect of the evolution and lifetime learning processes. It even exceeded the performance of the *CM*, due to the diversity induced by the *GEA*. It is not to expect the same behavior from the *BM*, since individuals need to restart the learning at each new generation. Yet, *Nature* evolves in dynamic and not static environments, where adaptation strategies may need to change or adapt over time.

When the problem to be learned changes with time, in accordance with the environments *E1* and *E2* referred to above, the results obtained are significantly different, as it is shown in Figures 8 and 9. For *E1*, the cubes are repainted every 50 seconds, while for *E2*, 2 cubes are changed every 10 seconds. Firstly, one can observe the inability of the *CM* to cope with changes in the environment, denoting a complete lack of adaptability, being incapable of escaping the initial learning bias. The *DM* shows a stable pattern, but seems unable to improve, in both cases. The *LM* seems the best model for some time in the beginning of the experiments, following a pattern of improvement, almost to a level of perfection. From a given point onwards, however, it shows a trend of decreasing performance. Finally, the *BM* shows almost the inverse behavior, starting with poor performances, but steadily evolving in a nearly continuous way, towards a stable solution, showing robustness when facing the changes in the environments. This results can be explained by concluding that the *BM* is evolving not a solution to a given task, but instead structures with a good learning capability.

7 Conclusions and Future Work

When faced with the given results, it is hard not to fall into the temptation of comparisons with natural systems. In fact, these were the inspiration of the work developed, and the results are a clear confirmation of the models that lead to the diversity and complexity of the living creatures. However, one developed a very simplified model, and to extrapolate with such narrow basis would be dangerous. Namely, when it comes to the comparison of the *Lamarckian* and the *Baldwinian* strategies, a lot has been said, and the natural systems are often used to support the latter. One can refer the fact that, due to the complexity of natural embryogenetic processes, a recoding of the genetic information after lifetime learning had occurred, would be a difficult and costly task. This is an important factor, that should not be forgotten when comparing both models.

Nevertheless, the results do support the idea that the evolution of learning entities makes itself has a very interesting method for *ML* tasks, when facing dynamic environments, that constitute most of the real-world applications. Referring to the *LM* vs *BM* debate, this work supports the belief that the former is preferable in static settings, while the latter reveals a greater robustness in dynamic ones.

In the future one intends to enlarge the experiments by tackling a greater domain of tasks, namely some real-world applications, such as *medical diagnosis*, *time-series forecasting* or *production scheduling*.

References

[1] J.M. Baldwin. A New Factor in Evolution. *American Naturalist*, (30):441–451, 1896.

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

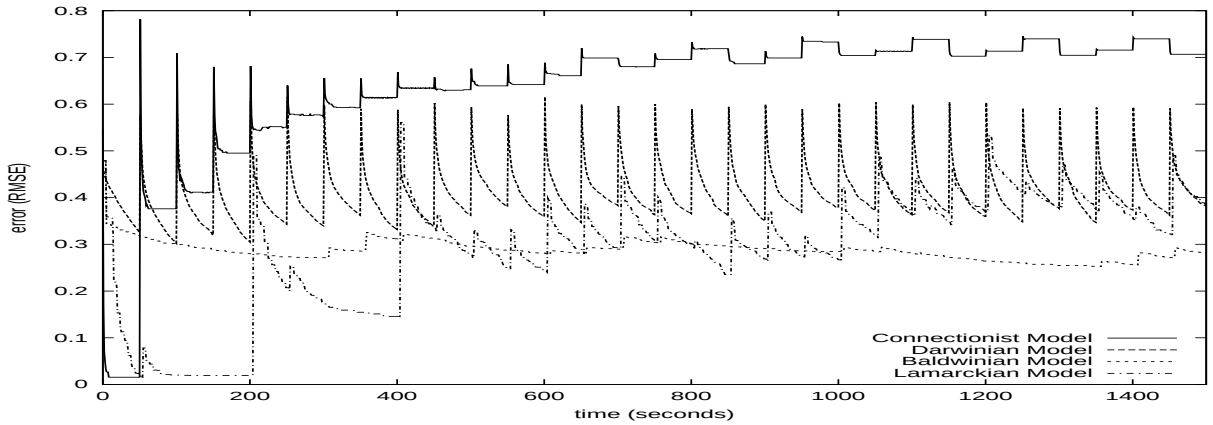

Figure 8: Results for the experiment in the dynamic environment *E1*

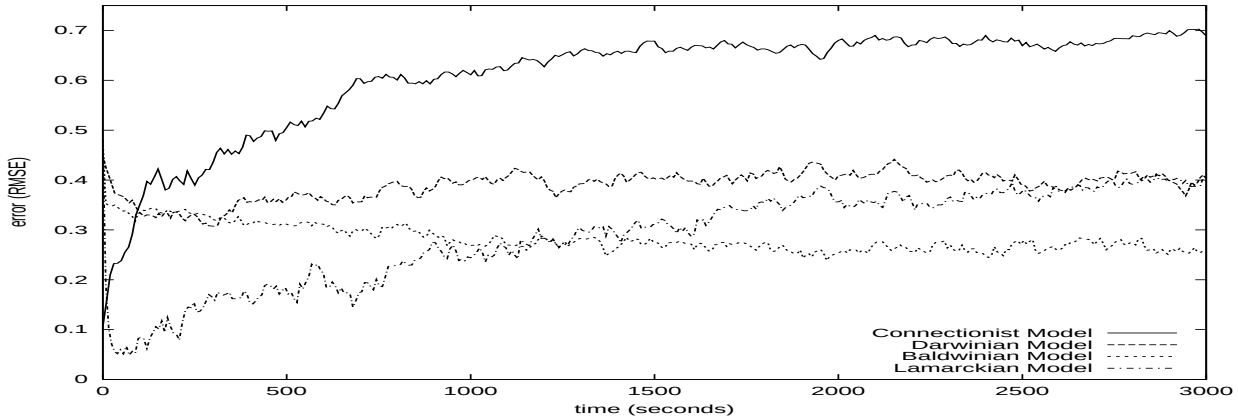

Figure 9: Results for the experiment in the dynamic environment *E2*

[3] G. Hinton and S. Nolan. How Learning Can Guide Evolution. *Complex Systems*, (1):495-502, 1987.

[4] John Holland. *Adaptation in Natural and Artificial Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[5] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, USA, third edition, 1996.

[6] J. Neves, M. Rocha, H. Rodrigues, M. Biscaia, and J. Alves. Adaptive Strategies and the Design of Evolutionary Applications. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99)*, Orlando, Florida, USA, 1999.

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

[8] T. Saski and M. Tokoro. Adaptation toward Changing Environments: Why Darwinian in Nature? In *Proceedings of the Fourth European Conference on Artificial Life*, 1997.

[9] P. Turney. Myths and Legends of the Baldwin Effect. 13th International Conference on Machine Learning (ICML96), Workshop on Evolutionary Computation and Machine Learning, Bari, Italy, July, 135-142, 1996.

[10] 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.