Neurocomputing 72 (2009) 2134–2145

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Using promoters and functional introns in genetic algorithms for neuroevolutionary learning in non-stationary problems

F. Bellas *, J.A. Becerra, R.J. Duro

Integrated Group for Engineering Research, Universidade da Coruña, Spain

ARTICLE INFO

Available online 16 December 2008

Keywords:
Genetic algorithms
Artificial neural networks
Neuroevolution
Non-stationary functions
Robotic learning

ABSTRACT

This paper addresses the problem of adaptive learning in non-stationary problems through neuroevolution. It is a general problem that is very relevant in many tasks, for example, in the context of robot model learning from interaction with the world. Traditional learning algorithms fail in this task as they have mostly been designed for learning a single model in a static setting. Neuroevolutionary techniques have obtained promising results in this non-stationary context but are still lacking in certain types of problems, especially those dealing with information streams where different portions correspond to different models. An extension through the introduction of the concept of introns and promoter genes enables neuroevolutionary algorithms to improve their performance on this type of problems. Following this approach, an implementation of these concepts on a genetic algorithm for neuroevolution is presented here. This algorithm is called promoter based genetic algorithm (PBGA) and it uses a genotypic representation with a set of features that allows for an intrinsic memory in the population that is self-regulated, in the sense that functional parts of the individuals are preserved through generations without an explicit knowledge about the number of different tasks or models that have to arise from the data stream. Some illustrative tests of the potential of these techniques based on the continuous switch between completely different objective functions that must be learnt are presented and the results are analyzed and compared to other neuroevolutionary algorithms.

1. Introduction

In real-world robot learning, there are usually no direct targets that permit choosing the correct action for every situation, leading to processes whereby optimal behavior must be learnt by exploring different actions and observing their results, thus, in a certain sense, by obtaining models of the interaction with the world. The domains in which these processes take place are usually continuous, partially observable, non-stationary and, in general, episodic. These four characteristics are what make it so difficult to provide good enough learning strategies for them.

Continuity implies that well-established approaches like Reinforcement Learning are not scalable in large state spaces because of the infinite number of possible states that must be handled [1]. On the other hand, the fact that the domains are non-stationary means that, the robot state, the environment and the objective may change in time, thus, there is a lack of a fixed mapping between solution encoding and solution fitness, and this problem is usually compounded with partial observability. It is in this context where neuroevolution, that is, to evolve artificial neural networks (ANN) using some type of evolutionary algorithm, becomes a reference tool due to its robustness and adaptability to dynamic environments [2] and in some types of non-stationary tasks [3].

However, it is also necessary to consider the episodic nature of the problem. The robot perceives episodes of sensorial information to be modeled as one model intermingled with episodes of sensorial information corresponding to other models. This implies that whatever perceptual streams the robot receives could contain information corresponding to different learning processes or models that are intermingled (periodically or not), that is, learning samples need not arise in an orderly and appropriate manner. Some of these sequences of samples are related to different sensorial or perceptual modalities and might not overlap in their information content; others correspond to the same modalities but should be assigned to different models.

The problem that arises is how to learn all of these different models, the samples of which are perceived as partial sequences that appear randomly intermingled with those of the others. Most traditional learning algorithms fail in this task as they have mostly been designed for learning a single model from sets of data that correspond to that model and, at most, some noisy samples or outliers within the training set. This problem becomes even more interesting if we consider that it would be nice to be able to reuse

* Corresponding author at: Escuela Politécnica Superior, Universidade da Coruña, Mendizábal s/n, 15403 Ferrol (A Coruña), Spain.
Tel.: +34 981 337400x3886; fax: +34 981 337410.
E-mail addresses: fran@udc.es (F. Bellas), ronin@udc.es (J.A. Becerra), richard@udc.es (R.J. Duro).

0925-2312/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.10.022

some of the models, or at least parts of them, that have been successful in previous tasks in order to produce models for more complex tasks in an easier and more straightforward manner.

In this work, we propose considering the concept of promoters and introns in neuroevolutionary algorithms. The concept of promoter is not new, but until now it has been applied very few times and only in terms of the ability to turn on or off any gene in a genotypic representation, no matter the meaning and the context of that gene. An example of this is the sGA [4–6] by Dasgupta or the promoters introduced in NEAT [7,8]. In this paper we go one step further and pair the concept of promoters with that of functional introns in terms of turning on or off functional units and not just any gene. In this work, introns are understood as functional units or gene sequences whose phenotypic expression is regulated by the promoters. This concept has been implemented in an algorithm called promoter based genetic algorithm (PBGA) that uses a genotypic representation with a set of features that allows for an intrinsic memory in the sense that functional parts of the ANNs are preserved in the individuals through the generations. With this representation a self-regulated evolutionary mechanism is obtained that stores previously learned information without explicit knowledge about the number or type of target functions or models.

The rest of the paper is structured as follows: Section 2 formally presents the domain of the problem. Section 3 introduces the concepts of intron and promoter and how they are implemented in the PBGA. Section 4 is devoted to the application results and to comparing this approach, as implemented in the PBGA, with other neurovolutionary algorithms in regards to this particular type of problem. Finally, Section 5 presents some conclusions.

2. Background

Neuroevolution is the artificial evolution of neural networks through evolutionary algorithms and has shown to be a very powerful technique for learning in non-stationary problems [2,9]. Evolution has been applied to ANNs at three different levels: connection weights, architectures and learning rules. Typical approaches, where the architecture is fixed and evolution searches the space of connection weights, have been successfully applied in the last decade for solving very complex problems [10] but, recently, several articles [7,11] have argued that this approach limits the functionality of the ANN. As a consequence, several researchers have proposed different algorithms that evolve both the connection weights and the architecture of the ANN [2,7,11].

Some of the most relevant neuroevolution methods presented in last few years are SANE [12], a cooperative coevolutionary algorithm that evolves a population of neurons instead of complete networks; ESP [10], similar to SANE but allocating a separate population for each of the units in the network, where each neuron can only be recombined with members of its own subpopulation; and NEAT [7], nowadays probably the most widely used neuroevolutionary algorithm, which can evolve networks of unbounded complexity from a minimal starting point and that is based on three fundamental principles: employing a principled method of crossover of different topologies, protecting structural innovation through speciation, and incrementally growing networks from a minimal structure [8].

Initially, most of the work on testing and benchmarking evolutionary algorithms found in the literature was carried out on stationary problems through either linear or non-linear control benchmarks like the pole balancing problem. The last decade, however, has seen an increase in the application of evolutionary algorithms to non-stationary problems. Most of this work was focused on optimization problems and using typical benchmarks for non-stationary optimization such as Osmera's dynamic problems [13] or the dynamic Knapsack problem [14].

The application field of the work presented in this paper is not directly optimization but learning. As established by Yao in [2], "learning is different from optimization because we want the learned system to have best generalization, which is different from minimizing an error function on a training dataset". In particular, the objective is to consider learning in non-stationary problems.

Thus, to formalize and frame the application domain of this work we will resort to the formalism in Trojanowsky's work [3] for delimiting the scope of non-stationary optimization problems and extend it to learning problems in terms of the types of changes that may take place in the objective function. This way, real-world learning problems can be, in general, modeled by:

$$M(P) = (D, F, C)$$

Meaning that a model M of a problem P can be expressed by defining the variables of the problem and their domains (D), the objective function to be learned (F) and a set of constraints that must be satisfied (C).

In a general non-stationary problem these three elements D, F and C, can change over time. The constraints of the problem C may vary in time because solutions that were acceptable in a given instant of time, become unacceptable. For a detailed reference in dynamic constraint optimization problems see [15]. On the other hand, D could change if the domains of the variables are modified or if the number of dimensions of the search space changes. Finally, changes in the objective function to be learnt may occur with time either because the function is intrinsically time dependent or, in the case of robotic systems, because the robot moves around or changes tasks and this implies a different F.

In this paper, which initially considers the problem of robot learning in non-stationary problems, we will leave aside constraints and we will consider that the robot has an unchanging set of sensors and actuators (D does not change). Thus, we are trying to learn in real environments through models of robot-environment interaction and all that is clear is what sensor and what actuators the system has, but it is not clear for each task or learning process which of these are necessary or what is the real target to be learnt in each case. This implies dealing with changes in the objective function F, the most typical case in real-world learning in robotics, referenced in general as non-stationary problems.

In this sense, there are different types of changes of F that may occur in time [3]:

1. *Random changes*: where the next change in F does not depend on the previous one. This usually leads to the learning of different problems. It is the case where two streams of data corresponding to different models are intermingled and occurs when, for example, a robot is exploring an unknown dynamic environment.
2. *Non-random non predictable changes*: the changes in F are not random but they are too complex to predict. This is another typical situation in robotics where the types of different environments and/or tasks are limited but the robot cannot predict the next one it will be faced with.
3. *Predictable changes*: these are changes that may be predicted and they come in two flavours: cyclical and non cyclical. This situation is possible in real-world robotics, and would make life much easier, but it is not typical.

Furthermore, the changes in the objective function can be continuous (adiabatic) or discrete. For the former case, several