ANN Development with EC Tools: An Overview

Daniel Rivero
University of A Coruña, Spain

Juan Rabuñal
University of A Coruña, Spain

INTRODUCTION

Among all of the Artificial Intelligence techniques, Artificial Neural Networks (ANNs) have shown to be a very powerful tool (McCulloch & Pitts, 1943) (Haykin, 1999). This technique is very versatile and therefore has been succesfully applied to many different disciplines (classification, clustering, regression, modellization, etc.) (Rabuñal & Dorado, 2005).

However, one of the greatest problems when using ANNs is the great manual effort that has to be done in their development. A big myth of ANNs is that they are easy to work with and their development is almost automatically done. This development process can be divided into two parts: architecture development and training and validation. As the network architecture is problem-dependant, the design process of this architecture used to be manually performed, meaning that the expert had to test different architectures and train them until finding the one that achieved best results after the training process. The manual nature of the described process determines its slow performance although the training part is completely automated due to the existence of several algorithms that perform this part.

With the creation of Evolutionary Computation (EC) tools, researchers have worked on the application of these techniques to the development of algorithms for automatically creating and training ANNs so the whole process (or, at least, a great part of it) can be automatically performed by computers and therefore few human efforts has to be done in this process.

BACKGROUND

EC is a set of tools based on the imitation of the natural behaviour of the living beings for solving optimization problems. One of the most typical subset of tools inside EC is called Evolutionary Algorithms (EAs), which are based on natural evolution and its implementation on computers. All of these tools work with the same basis: a population of solutions to that particular problem is randomly created and an evolutionary process is applied to it. From this initial random population, the evolution is done by means of selection and combination of the best individuals (although the worst ones also have a small probability of being chosen) to create new solutions. This process is carried out by selection, crossover, and mutation operators. These operators are typically used in biology in its evolution for adaptation and survival. After several generations, it is hoped that the population contains a good solution to the problem.

The first EA to appear was Genetic Algorithms (GAs), in 1975 (Holland, 1975). With the working explained above, GAs use a binary codification (i.e., each solution is codified into a string of bits). Later, in the early 90s a new technique appeared, called Genetic Programming (GP). This one is based ob the evolution of trees, i.e., each individual is codified as a tree instead of a binary string. This allows its application to a wider set of environments.

Although GAs and GP are the two most used techniques in EAs, more tools can be classified as part of this world, such as Evolutionary Programming or Evolution Strategies, all of them with the same basis: the evolution of a population following the natural evolution rules.

DEVELOPMENT OF ANNS WITH EC TOOLS

The development of ANNs is a topic that has been extensively dealt with very diverse techniques. The world of evolutionary algorithms is not an exception, and proof of that is the great amount of works that have

been published about different techniques in this area (Cantú-Paz & Kamath, 2005). These techniques follow the general strategy of an evolutionary algorithm: an initial population consisting of different genotypes, each one of them codifying different parameters (typically, the weight of the connections and / or the architecture of the network and / or the learning rules), and is randomly created. This population is evaluated in order to determine the fitness of each individual. Afterwards, this population is repeatedly made to evolve by means of different genetic operators (replication, crossover, mutation, etc.) until a determined termination criteria is fulfilled (for example, a sufficiently good individual is obtained, or a predetermined maximum number of generations is achieved).

Essentially, the ANN generation process by means of evolutionary algorithms is divided into three main groups: evolution of the weights, architectures, and learning rules.

Evolution of Weights

The evolution of the weights begins with a network with a predetermined topology. In this case, the problem is to establish, by means of training, the values of the network connection weights. This is generally conceived as a problem of minimization of the network error, taken, for example, as the result of the Mean Square Error of the network between the desired outputs and the ones achieved by the network. Most the training algorithms, such as the backpropagation algorithm (BP) (Rumelhart, Hinton & Williams, 1986), are based on gradient minimization. This has several drawbacks (Whitley, Starkweather & Bogart, 1990), the most important is that quite frequently the algorithm becomes stuck in a local minimum of the error function and is unable of finding the global minimum, especially if the error function is multimodal and / or non-differentiable. One way of overcoming these problems is to carry out the training by means of an Evolutionary Algorithm (Whitley, Starkweather & Bogart, 1990); i.e., formulate the training process as the evolution of the weights in an environment defined by the network architecture and the task to be done (the problem to be solved). In these cases, the weights can be represented in the individuals' genetic material as a string of binary values (Whitley, Starkweather & Bogart, 1990) or a string of real numbers (Greenwood, 1997). Traditional genetic algorithms (Holland, 1975) use a genotypic codification method with the shape of binary strings. In this way, much work has emerged that codifies the values of the weights by means of a concatenation of the binary values which represent them (Whitley, Starkweather & Bogart, 1990). The big advantage of these approximations is their generality and that they are very simple to apply, i.e., it is very easy and quick to apply the operators of uniform crossover and mutation on a binary string. The disadvantage of using this type of codification is the problem of permutation. This problem was raised upon considering that the order in which the weights are taken in the string causes equivalent networks to possibly correspond with totally different individuals. This leads the crossing operator to become very inefficient. Logically, the weight value codification has also emerged in the form of real number concatenation, each one of them associated with a determined weight (Greenwood 1997). By means of genetic operators designed to work with this type of codification, and given that the existing ones for bit string cannot be used here, several studies (Montana & Davis, 1989) showed that this type of codification produces better results and with more efficiency and scalability than the BP algorithm.

Evolution of the Architectures

The evolution of the architectures includes the generation of the topological structure; i.e., the topology and connectivity of the neurons, and the transfer function of each neuron of the network. The architecture of a network has a great importance in order to successfully apply the ANNs, as the architecture has a very significant impact on the process capacity of the network. In this way, on one hand, a network with few connections and a lineal transfer function may not be able to resolve a problem that another network having other characteristics (distinct number of neurons, connections or types of functions) would be able to resolve. On the other hand, a network having a high number of non-lineal connections and nodes could be overfitted and learn the noise which is present in the training as an inherent part of it, without being able to discriminate between them, and in the end, not have a good generalization capacity. Therefore, the design of a network is crucial, and this task is classically carried out by human experts using their own experience, based on "trial and error", experimenting with a different set of architectures. The evolution of architectures has