Evolutionary Development of ANNs for Data Mining

Daniel Rivero
University of A Coruña, Spain

Juan R. Rabuñal
University of A Coruña, Spain

Julián Dorado
University of A Coruña, Spain

Alejandro Pazos
University of A Coruña, Spain

INTRODUCTION

Artificial Neural Networks (ANNs) are learning systems from the Artificial Intelligence (AI) world that have been used for solving complex problems related to different aspects as classification, clustering, or regression (Haykin, 1999), although they have been specially used in Data Mining. These systems are, due to their interesting characteristics, powerful techniques used by the researchers in different environments (Rabuñal, 2005).

Nevertheless, the use of ANNs implies certain problems, mainly related to their development processes. The development of ANNs can be divided into two parts: architecture development and training and validation. The architecture development determines not only the number of neurons of the ANN, but also the type of the connections among those neurons. The training will determine the connection weights for such architecture.

Traditionally, and given that the architecture of the network depends on the problem to be solved, the architecture design process is usually performed by the use of a manual process, meaning that the expert has to test different architectures to find the one able to achieve the best results. Therefore, the expert must perform various tests for training different architectures in order to determine which one of these architectures is the best one. This is a slow process due to the fact that architecture determination is a manual process, although techniques for relatively automatic creation of ANNs have been recently developed.

This work presents various techniques for the development of ANNs, so that there would be needed much less human participation for such development.

BACKGROUND

The development of ANNs has been widely treated with very different techniques in AI. The world of evolutionary algorithms is not an exception, and proof of it is the large amount of works that have been published is this aspect using several techniques (Nolfi, 2002; Cantú-Paz, 2005). These techniques follow the general strategy of an evolutionary algorithm: an initial population with different types of genotypes encoding also different parameters – commonly, the connection weights and/or the architecture of the network and/or the learning rules – is randomly created. Such population is evaluated for determining the fitness of every individual. Subsequently, the population is repeatedly induced to evolve by means of different genetic operators (replication, crossover, mutation) until a certain termination parameter has been fulfilled (for instance, the achievement of an individual good enough or the accomplishment of a predetermined number of generations).

As a general rule, the field of ANNs generation using evolutionary algorithms is divided into three

main groups: evolution of weights, architectures and learning rules.

The evolution of weight starts from an ANN with an already determined topology. In this case, the problem to be solved is the training of the connection weights, attempting to minimise the network failure. Most of training algorithms, as backpropagation (BP) algorithm (Rumelhart, 1986), are based on gradient minimisation, which presents several inconveniences (Sutton, 1986). The main of these disadvantages is that, quite frequently, the algorithm gets stuck into a local minimum of the fitness function and it is unable to reach a global minimum. One of the options for overcoming this situation is the use of an evolutionary algorithm, so the training process is done by means of the evolution of the connection weights within the environment defined by both, the network architecture, and the task to be solved. In such cases, the weights can be represented either as the concatenation of binary values or of real numbers on a genetic algorithm (GA) (Greenwood, 1997). The main disadvantage of this type of encoding is the permutation problem. This problem means that the order in which weights are taken at the vector might cause that equivalent networks might correspond to completely different chromosomes, making the crossover operator inefficient.

The evolution of architectures includes the generation of the topological structure. This means establishing the connectivity and the transfer function of each neuron. The network architecture is highly important for the successful application of the ANN, since the architecture has a very significant impact on the processing ability of the network. Therefore, the network design, traditionally performed by a human expert using trial and error techniques on a group of different architectures, is crucial. In order to develop ANN architectures by means of an evolutionary algorithm it is needed to choose how to encode the genotype of a given network for it to be used by the genetic operators.

At the first option, direct encoding, there is a one-to-one correspondence between every one of the genes and their subsequent phenotypes (Miller, 1989). The most typical encoding method consists of a matrix that represents an architecture where every element reveals the presence or absence of connection between two nodes (Alba, 1993). These types of encoding are generally quite simple and easy to implement. However, they also have a large amount of inconveniences as scalability (Kitano, 1990), the incapability of encoding repeated structures, or permutation (Yao, 1998).

In comparison with direct encoding, there are some indirect encoding methods. In these methods, only some characteristics of the architecture are encoded in the chromosome and. These methods have several types of representation.

Firstly, the parametric representations represent the network as a group of parameters such as number of hidden layers, number of nodes for each layer, number of connections between two layers, etc (Harp, 1989). Although the parametric representation can reduce the length of the chromosome, the evolutionary algorithm performs the search within a restricted area in the search space representing all the possible architectures. Another non direct representation type is based on a representation system that uses the grammatical rules (Kitano, 1990). In this system, the network is represented by a group of rules, shaped as production rules that make a matrix that represents the network, which has several restrictions.

The growing methods represent another type of encoding. In this case, the genotype does not encode a network directly, but it contains a group of instructions for building up the phenotype. The genotype decoding will consist on the execution of those instructions (Nolfi, 2002), which can include neuronal migrations, neuronal duplication or transformation, and neuronal differentiation.

Another important non-direct codification is based on the use of fractal subsets of a map. According to Merrill (1991), the fractal representation of the architectures is biologically more plausible than a representation with the shape of rules. Three parameters are used which take real values to specify each node in an architecture: a border code, an entry coefficient and an exit code.

One important characteristic to bear in mind is that all those methods evolve architectures, either alone (most commonly) or together with the weights. The transfer function for every node of the architecture is supposed to have been previously fixed by a human expert and is the same for all the nodes of the network –or at least, all the nodes of the same layer–, despite of the fact that such function has proved to have a significant impact on network performance (DasGupta, 1992). Only few methods that also induce the evolution of the transfer function have been developed (Hwang, 1997).

With regards to the evolution of the learning rule, there are several approaches (Turney, 1996), although