Evolving Graphs for ANN Development and Simplification

Daniel Rivero
University of A Coruña, Spain

David Periscal
University of A Coruña, Spain

INTRODUCTION

One of the most successful tools in the Artificial Intelligence (AI) world is Artificial Neural Networks (ANNs). This technique is a powerful tool used in many different environments, with many different purposes, like classification, clustering, signal modelization, or regression (Haykin, 1999). Although they are very easy to use, their creation is not a simple task, because the expert has to do much effort and spend much time on it.

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 determines the connection weights for such architecture.

The architecture design task is usually performed by means of a manual process, meaning that the expert has to test different architectures to find the one able to achieve the best results. Each architecture trial means training and validating it, which can be a process that needs many computational resources, depending on the complexity of the problem. Therefore, the expert has much participation in the whole ANN development, although techniques for relatively automatic creation of ANNs have been recently developed.

BACKGROUND

ANN development is a research topic that has attracted many researchers from the world of evolutionary algorithms (Nolfi & Parisi D., 2002) (Cantú-Paz & Kamath, 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 and repeatedly induced to evolve.

The most direct application of EC tools in the ANN world is to perform the evolution of the weights of the connections. This process 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, Hinton & Williams, 1986), are based on gradient minimisation, which presents several inconveniences. 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 evolution of architectures consists on the generation of the topological structure, i.e., establishing the connectivity and the transfer function of each neuron. To achieve this goal with 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.

The most typical approach is called direct encoding. In this technique there is a one-to-one correspondence between each of the genes and a determined part of the network. A binary matrix represents an architecture where every element reveals the presence or absence

of connection between two nodes (Alba, Aldana & Troya, 1993).

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. 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, Samad & Guha, 1989). Another non direct representation type is based on a representation system that uses grammatical rules (Kitano, 1990), shaped as production rules that make a matrix that represents the network.

Another type of encoding is the growing methods. In this case, the genotype contains a group of instructions for building up the network (Nolfi & Parisi, 2002).

All of these 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. Only few methods that also induce the evolution of the transfer function have been developed (Hwang, Choi & Park, 1997).

ANN DEVELOPMENT WITH GENETIC PROGRAMMING

This section very briefly shows an example of how to develop ANNs using an AI tool, Genetic Programming (GP), which performs an evolutionary algorithm, and how it can be applied to Data Mining tasks.

Genetic Programming

GP (Koza, 92) is based on the evolution of a given population. Its working is similar to a GA. In this population, every individual represents a solution for a problem that is intended to be solved. The evolution is achieved by means of the selection of the best individuals – although the worst ones have also a little chance of being selected – and their mutual combination for creating new solutions. After several generations, the population is expected to contain some good solutions for the problem.

The GP encoding for the solutions is tree-shaped, so the user must specify which are the terminals (leaves of the tree) and the functions (nodes capable of having descendants) for being used by the evolutionary algorithm in order to build complex expressions.

The wide application of GP to various environments and its consequent success are due to its capability for being adapted to numerous different problems. Although the main and more direct application is the generation of mathematical expressions (Rivero, Rabuñal, Dorado & Pazos, 2005), GP has been also used in other fields such as filter design (Rabuñal, Dorado, Puertas, Pazos, Santos & Rivero D., 2003), knowledge extraction, image processing (Rivero, Rabuñal, Dorado & Pazos, 2004), etc.

Model Overview

This work will use a graph-based codification to represent ANNs in the genotype. These graphs will not contain any cycles. Due to this type of codification the genetic operators had to be changed in order to be able to use the GP algorithm. The operators were changed in this way:

- The creation algorithm must allow the creation of graphs. This means that, at the moment of the creation of a node's child, this algorithm must allow not only the creation of this node, but also a link to an existing one in the same graph, without making cycles inside the graph.
- The crossover algorithm must allow the crossing of graphs. This algorithm works very similar to the existing one for trees, i.e. a node is chosen on each individual to change the whole subgraph it represents to the other individual. Special care has to be taken with graphs, because before the crossover there may be links from outside this subgraph to any nodes on it. In this case, after the crossover these links are updated and changed to point to random nodes in the new subgraph.
- The mutation algorithm has been changed too, and also works very similar to the GP tree-based mutation algorithm. A node is chosen from the individual and its subgraph is deleted and replaced with a new one. Before the mutation occurs, there may be nodes in the individual pointing to other nodes in the subgraph. These links are updated