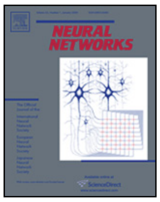
Neural Networks 22 (2009) 614–622

Contents lists available at ScienceDirect

Neural Networks

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



2009 Special Issue

Neural networks with multiple general neuron models: A hybrid computational intelligence approach using Genetic Programming

Alan J. Barton[*], Julio J. Valdés, Robert Orchard

Knowledge Discovery Group, Institute for Information Technology, National Research Council Canada, Ottawa, Ontario, Canada

ARTICLE INFO

Article history:
Received 5 May 2009
Received in revised form 11 June 2009
Accepted 25 June 2009

Keywords:
General neuron model
Evolutionary Computation
Genetic Programming
Hybrid algorithm
Machine learning
Parameter space
Visualization

ABSTRACT

Classical neural networks are composed of neurons whose nature is determined by a certain function (the neuron model), usually pre-specified. In this paper, a type of neural network (NN-GP) is presented in which: (i) each neuron may have its own neuron model in the form of a general function, (ii) any layout (i.e network interconnection) is possible, and (iii) no bias nodes or weights are associated to the connections, neurons or layers. The general functions associated to a neuron are learned by searching a function space. They are not provided a priori, but are rather built as part of an Evolutionary Computation process based on Genetic Programming. The resulting network solutions are evaluated based on a fitness measure, which may, for example, be based on classification or regression errors. Two real-world examples are presented to illustrate the promising behaviour on classification problems via construction of a low-dimensional representation of a high-dimensional parameter space associated to the set of all network solutions.

1. Introduction

Many different neuron models, neural network architectures and learning procedures have been proposed, addressing two main types of problems: regression and classification. Both can be seen as derived from the general problem of function approximation. The feed forward neural network (multilayer perceptron) (Bishop, 2004; Ripley, 1996) is probably the most popular and it is composed of a layout of similar neurons arranged into layers, trained with the backpropagation algorithm or one of its many variants. In networks of these types, the goal of the learning procedure is to find the vector of weights for each neuron in the network such that a given function is optimized (a classification error, a least squared error, an entropy measure, etc.).

From a more general perspective, the networks may be composed of neurons in which the neuron model is not fixed and in which the architecture is not necessarily a layered one. Fig. 1 shows a general network architecture with general multiple neuron models and Fig. 2 shows the same neuron models for a layered network. In particular, the neurons do not consist of the composition of aggregation and activation functions endowed with a vector of weights (possibly with a bias). Rather, they are given by a general analytical (deterministic) function, which can be connected in any way specified by a directed graph in order to define a network. In this case, the learning procedure is oriented to find the collection of functions (and their parameters) such that the network output optimizes a given performance measure, as previously indicated. The use of general analytic functions as neuron models is appealing because they are: (i) easy to understand by humans, (ii) the preferred building blocks of modeling, and (iii) a classical and highly condensed form of knowledge. There are many possible approaches for Evolutionary Computation based learning of neural networks (Koehn, 1996; Stanley, 2004; Yao, 1993; Zhang & Mühlenbein, 1993). For example, Montana and Davis (1989) encode the weights of a fixed architecture neural network within one chromosome and use a Genetic Algorithm (GA) to perform global optimization in order to find potential weight solutions for a sonar image classification problem.

This paper uses multiple chromosomes (Valdés, Orchard, & Barton, 2007) and learns the complete function associated to a neuron, not only the weights (Barton & Valdés, 2008; Barton, 2009). The purpose of this paper is to extend previous results (Barton, Valdés, & Orchard, 2009) and to report the promising empirical behaviour of NN-GP when used for two real-world classification problems; underground cave detection and hydrochemical research in the Arctic. In particular, a low-dimensional visual representation of the algorithm's parameter spaces are used to demonstrate the network solution qualities w.r.t. their parameters.

2. Evolutionary Computation

A general Evolutionary Computation (EC) algorithm (Algorithm 1) consists of a problem, $\mathbb{P}$, as input and a set of solutions, $\mathbb{S}$, as

[*] Corresponding author. Tel.: +1 613 991 5486; fax: +1 613 952 0215.
E-mail addresses: alan.barton@nrc-cnrc.gc.ca (A.J. Barton), julio.valdes@nrc-cnrc.gc.ca (J.J. Valdés), orchardr@rogers.com (R. Orchard).

0893-6080/$ – see front matter © 2009 Published by Elsevier Ltd
doi:10.1016/j.neunet.2009.06.043

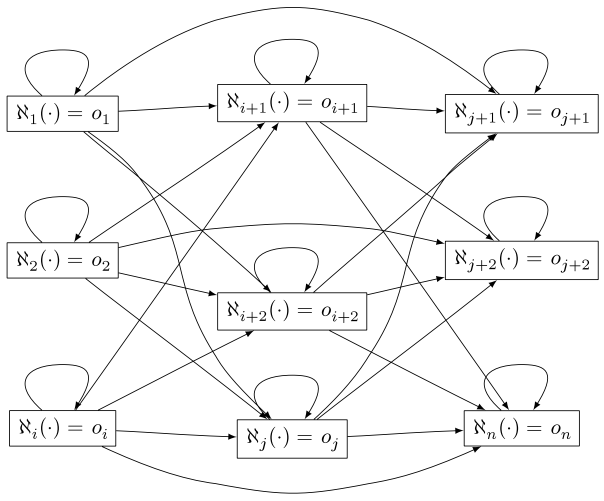

$\aleph_1(\cdot) = o_1$
$\aleph_{i+1}(\cdot) = o_{i+1}$
$\aleph_{j+1}(\cdot) = o_{j+1}$
$\aleph_2(\cdot) = o_2$
$\aleph_{j+2}(\cdot) = o_{j+2}$
$\aleph_{i+2}(\cdot) = o_{i+2}$
$\aleph_i(\cdot) = o_i$
$\aleph_j(\cdot) = o_j$
$\aleph_n(\cdot) = o_n$

Fig. 1. General Neural Network containing (n) neurons where all activities occur (e.g. activation, aggregation). Weights are learned within the neuron. $\aleph_i(\cdot)$ is the function associated to the ith neuron; with output o_i. The network may be organized into layers. Not all possible connections are shown. No bias neurons are used.

output. Algorithm 1 consists of three main stages: (i) initialization Line 1–4, (ii) processing generations Line 5–11, and (iii) post-processing Line 12.

```
Algorithm 1: General EC Specification
   Input  : A problem, $\mathbb{P}$. (i.e. a question)
   Output: A set of solutions, $\mathbb{S}$. (i.e. possible answer(s) to the
           question)
 1 Let $\mathcal{A} \Leftarrow$ archivePopulations($\emptyset, \emptyset$) ;
 2 Let $\mathcal{F_P} \Leftarrow$ constructPopulations($\emptyset$) ;
 3 computeFitness($\mathcal{F_P}$) ;
 4 $\mathcal{A} \Leftarrow$ archivePopulations($\mathcal{A}, \mathcal{F_P}$) ;
 5 while not terminationCriteriaSatisfied($\mathcal{F_P}$) do
 6     Let $\mathcal{F_{P'}} \Leftarrow$ constructPopulations($\mathcal{F_P}$) ;
 7     computeFitness($\mathcal{F_{P'}}$) ;
 8     $\mathcal{F_P} \Leftarrow$ combinePopulations($\mathcal{F_P}, \mathcal{F_{P'}}$) ;
 9     $\mathcal{A} \Leftarrow$ archivePopulations($\mathcal{A}, \mathcal{F_P}$) ;
10     $\mathcal{F_P} \Leftarrow$ selectIndividuals($\mathcal{F_P}, \mathcal{A}$) ;
11 end
12 $\mathbb{S} \Leftarrow$ selectIndividuals($\emptyset, \mathcal{A}$) ;
```

2.1. Genetic Programming (GP)

Analytic functions are among the preferred building blocks for modeling and a highly condensed form of knowledge, but direct discovery of general analytic functions poses enormous challenges because of the size of the search space. This problem can be approached from a computational intelligence perspective via Evolutionary Computation. In particular, Genetic Programming techniques aim at evolving computer programs, which ultimately are functions.

Genetic Programming was introduced by Koza (1989, 1992, 1994) and Koza, Bennett, Andre, and Keane (1999) as an extension of genetic algorithms that evolves a population of computer programs. In Algorithm 1, the family of populations of individuals $\mathcal{F_P}$ (Line 2,8,9,10) or $\mathcal{F_{P'}}$ (Line 6,8) is usually a family composed of one set. In other words, $\mathcal{F_P}$ and $\mathcal{F_{P'}}$ are each a simple set of individuals $\mathcal{I}$. GP programs may be symbolic expression trees, which, essentially, are functions. Algorithm 1 uses Koza's computer programs on lines (Line 2,6,8), and names them *internal representations*. More specifically, one of Koza's computer programs is called an S-expression (S stands for symbolic) and is used, for example, within the List Processing Language (LISP). An example of a LISP S-expression that represents a neural network is given in Koza and Rice (1991).

2.2. Gene expression programming (GEP)

Gene expression programming (GEP) (Ferreira, 2001, 2006) is a variant of Genetic Programming where individuals are expression trees encoded as simple strings of fixed length (chromosomes) representing entities of different sizes and shapes, generally nonlinear expressions. For the interplay of the GEP chromosomes and the expression trees (ET), GEP uses an unambiguous translation system to transfer the language of chromosomes into the language of expression trees and vise versa (Ferreira, 2006). The set of genetic operators applied to GEP chromosomes always produces valid ETs.

The chromosomes in GEP itself are composed of genes structurally organized in a head and a tail (Ferreira, 2001). The head contains symbols that represent both functions (elements from a function set F) and terminals (elements from a terminal set T), whereas the tail contains only terminals. Therefore, two different alphabets occur at different regions within a gene. For each problem, the length of the head h is chosen, whereas the length of the tail t is a function of h, and the number of arguments of the function with the largest arity ($n_{\max}$). The length of the tail is evaluated by $t = h(n_{\max} - 1) + 1$.

As an example, consider a gene composed of the function set $F = \{Q, +, -, *, /\}$, where Q represents the square root function, and the terminal set $T = \{a, b\}$. In this case $n_{\max} = 2$. For instance $h = 10$ and $t = 11$, the length of the gene is $10 + 11 = 21$. Such a gene looks like (the tail is shown in **bold**): $\star$ Q-b++a/-b**baabaaabaab**, and corresponds to the mathematical equation $f(a, b) = \sqrt{b} \cdot \left(\left(a + \frac{b}{a}\right) - ((a - b) + b)\right)$ simplified as $f(a, b) = \frac{b \cdot \sqrt{b}}{a}$.

GEP chromosomes are usually composed of more than one gene of equal length. For each problem the number of genes as well as the length of the head has to be chosen. Each gene encodes a sub-ET

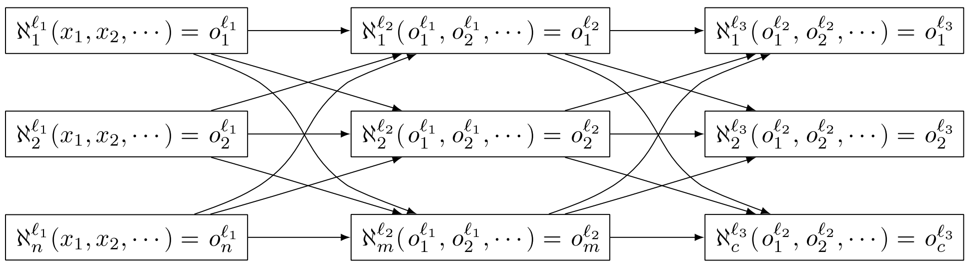

$\aleph_1^{\ell_1}(x_1, x_2, \cdots) = o_1^{\ell_1}$
$\aleph_1^{\ell_2}(o_1^{\ell_1}, o_2^{\ell_1}, \cdots) = o_1^{\ell_2}$
$\aleph_1^{\ell_3}(o_1^{\ell_2}, o_2^{\ell_2}, \cdots) = o_1^{\ell_3}$
$\aleph_2^{\ell_1}(x_1, x_2, \cdots) = o_2^{\ell_1}$
$\aleph_2^{\ell_2}(o_1^{\ell_1}, o_2^{\ell_1}, \cdots) = o_2^{\ell_2}$
$\aleph_2^{\ell_3}(o_1^{\ell_2}, o_2^{\ell_2}, \cdots) = o_2^{\ell_3}$
$\aleph_n^{\ell_1}(x_1, x_2, \cdots) = o_n^{\ell_1}$
$\aleph_m^{\ell_2}(o_1^{\ell_1}, o_2^{\ell_1}, \cdots) = o_m^{\ell_2}$
$\aleph_c^{\ell_3}(o_1^{\ell_2}, o_2^{\ell_2}, \cdots) = o_c^{\ell_3}$

Fig. 2. Special case of the general neural network: a feed forward neural network for one specific architecture ($NN : n - m - c$). There are 3 layers and ($n + m + c$) neurons where neurons in layer i are connected to neurons in layer $i + 1$. Neurons may not use all inputs; implying a connectivity upper bound. Possibilities exist providing more reuse (e.g. neurons in layer i may connect to neurons in *all* layers greater than i).