A developmental artificial neural network model for solving multiple problems

Julian F. Miller
University of York
Heslington
York, UK YO10 5DD
julian.miller@york.ac.uk

Dennis G. Wilson
University of Toulouse, IRIT - CNRS - UMR5505
21 allee de Brienne
Toulouse, France 31015
dennis.wilson@irit.fr

ABSTRACT

A developmental model of an artificial neuron is presented. In this model, a pair of neural developmental programs develop an entire artificial neural network of arbitrary size. The pair of neural chromosomes are evolved using Cartesian Genetic Programming. During development, neurons and their connections can move, change, die or be created. We show that this two-chromosome genotype can be evolved to develop into a single neural network from which multiple conventional artificial neural networks can be extracted. The extracted conventional ANNs share some neurons across tasks. We have evaluated the performance of this method on three standard classification problems. The evolved pair of neuron programs can generate artificial neural networks that perform reasonably well on all three benchmark problems simultaneously. It appears to be the first attempt to solve multiple standard classification problems using a developmental approach.

CCS CONCEPTS

•Computing methodologies → Neural networks; Generative and developmental approaches; *Genetic programming;*

KEYWORDS

Cartesian Genetic Programming, Artificial Neural Networks, Classification

ACM Reference format:
Julian F. Miller and Dennis G. Wilson. 2017. A developmental artificial neural network model for solving multiple problems. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3075976

1 INTRODUCTION

Despite artificial neural networks (ANNs) being first proposed over sixty years ago, there remains a large gap in the cognitive ability between ANNs and their biologic counterpart. We suggest that one weakness of many ANNs models is that they encode learned knowledge solely in the form of connection strengths (i.e. weights) [4] This leads to "catastrophic forgetting" (CF) in which ANNs trained to perform well on one problem, forget how to solve the original problem when they are re-trained on a new problem [2]. In principle, developmental neural approaches could alleviate CF by growing numerous connections between pairs of neurons. While ANNs that develop based on an evolved rule set have been explored, there remains a need for a greater effort to explore developmental neural networks which modify their topology online, as seen in biologic neural growth. We propose a new conceptually simple neural model and suggest that at least two neural programs are required to construct neural networks: one to represent the neuron soma and the other the dendrite. The role of these programs is to allow neurons to move, change, die or replicate. For the dendrite, the program needs to be able to grow and change dendrites, cause them to die and also to replicate. Since developmental programs build networks that change over time it is necessary to define new problem classes that are suitable to evaluate such approaches. We argue that trying to solve multiple computational problems (potentially even of different types) is an appropriate type of problem. We show that the pair of evolved programs can build a network from which multiple conventional ANNs can be extracted, each of which can solve a different classification problem.

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3075976

2 THE NEURON MODEL

Our aim is to construct a *minimal* developmental model. Minimal means that if we take a snapshot of the neural network at a particular time we would see a conventional graph of neurons, weighted connections and sigmoidal activation functions. However, to make a *developmental* neural network we require a mechanism whereby the ANN can change over time, even during training. In addition, we take a cellular view of development, in which an entire network is developed from a few cells. The network itself grows from the interaction of neurons acting in parallel (but sequentially simulated). In the model, all the neuron and dendrite parameters (weights, health and position) are defined by numbers in the range $[-1, -1]$. The places where external inputs are provided is predetermined uniformly within the region between -1 and I_u. The parameter I_u defines the upper bound of their position. Similarly, output neurons are initially uniformly distributed between the parameter O_l and 1. However, output neurons, as with other neurons, can move according to the neuron program. All neurons are marked as to whether they provide an external output or not. In the initial network there are N_{init} non-output neurons and N_o output neurons, where N_o denotes the number of outputs required by the computational problem being solved.

To construct such a developmental model of an artificial neural network we need neural programs that not only apply a weighted

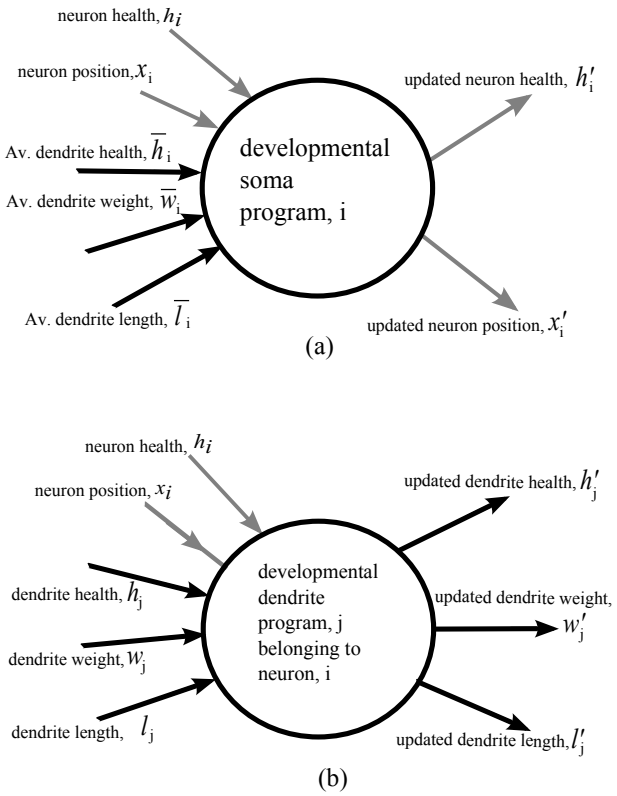
sum of inputs to an activation function to determine the output from the neuron, but a program that can adjust weights, create or delete connections, and create or delete neurons. Following [1] we have used the concept of health to make this possible. Neuron and connection healths are real numbers between $[-1, 1]$. However, if the health of a neuron falls below (exceeds) a user-defined threshold, HN_l (HN_u) the neuron will be deleted (replicated). Likewise, dendrites are subject to user defined health thresholds, HD_l (HD_u) which determine whether the dendrite will be deleted or a new one will be created. The number of dendrites each neuron can have is bounded by user-defined lower (upper) bounds denoted by $ND_l(ND_u)$. Also the total number of neurons allowed in the network is also bounded between a user-defined lower (upper) bound $N_l(N_u)$. These parameters ensure that the number of neurons and connections per neuron remain in well-defined bounds, so that a network can not eliminate itself or grow too large. The model is illustrated in in Fig. 1. All weights are assumed to lie in the range $[-1, 1]$ and a sigmoid transfer function is used. The two neural programs are represented and evolved using a form of Genetic Programming (GP) known as Cartesian Genetic Programming (CGP). CGP [3, 5] is a form of GP in which computational structures are represented as directed, often acyclic graphs indexed by their Cartesian coordinates. The programs are actually sets of mathematical equations that read variables associated with neurons and dendrites to output updates of those variables. This approach was inspired by some aspects of a developmental method for evolving graphs and circuits proposed by Miller and Thomson [6] and was influenced by some of the ideas described in [1]. In the proposed model, weights are determined from a program that is a function of neuron position, together with the health, weight and length of dendrites.

The inputs to the soma program are as follows: the health and position of the neuron and the average health, length and weight of all dendrites connected to the neuron. The soma program updates its own health and position based on these inputs. These are indicated by primed symbols in Fig. 1. Every dendrite belonging to each neuron is controlled by an evolved dendrite program. The inputs to this program are the health, weight and length of the dendrite and also the health and position of the parent neuron. The evolved dendrite program decides how the health, weight and length of the dendrite are to be updated.

3 EXPERIMENTS AND CONCLUSIONS

We have evolved neural programs that build ANNs for solving three standard classification problems (cancer, diabetes and glass). The definitions of these problems are available in the UCI repository of machine learning problems[1]. The main research questions for this experimental investigation are: *Are multiple learning epochs more effective than a single epoch? Are shared inputs to the ANN more effective than non-shared? Should output neurons be allowed to move? Should evolved program outputs update neural variables directly or should they determine user-defined increments in those variables (linear or non-linear)?* Experimental tests showed that the best performance occurs when problem inputs are shared, output

Figure 1: The model of a developmental neuron



neuron can move and adjustment of neuron variables is incremental. Comparison of results against other published developmental approaches is intended for future work, as there is little precedent for using developmental ANNs to simultaneously solve multiple classification problems.

REFERENCES

[1] G. M. Khan, J. F. Miller, and D. M. Halliday. Evolution of Cartesian Genetic Programs for Development of Learning Neural Architecture. *Evol. Computation*, 19(3):469–523, 2011.
[2] M. McCloskey and N. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation*, 24:109–165, 1989.
[3] J. F. Miller, editor. *Cartesian Genetic Programming*. Springer, 2011.
[4] J. F. Miller and G. M. Khan. Where is the Brain inside the Brain? *Memetic Computing*, 3(3):217–228, 2011.
[5] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proc. European Conf. on Genetic Programming*, volume 10802 of *LNCS*, pages 121–132, 2000.
[6] J. F. Miller and P. Thomson. A Developmental Method for Growing Graphs and Circuits. In *Proc. Int. Conf. on Evolvable Systems*, volume 2606 of *LNCS*, pages 93–104, 2003.

[1] https://archive.ics.uci.edu/ml/datasets.html