Universal Approximation by Genetic Programming

Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
Email: xin@cs.bham.ac.uk
URL: http://www.cs.bham.ac.uk/~xin

Abstract

Genetic programming (GP) has been applied successfully to many difficult problems. However, little theory is currently available to explain why GP works or does not work for a particular problem. We investigate the power of GP in terms of its approximation capability to arbitrary functions. The relationship between artificial neural networks (ANNs) and GP is discussed. Such relationship enables us to apply the existing theoretical results in ANNs to GP and show that GP can be a universal approximator. This result shows at least partially why GP is capable of solving some very difficult problems. It also sheds some light on choosing the function set for GP applications.

1 Introduction

Many researchers have reported successful genetic programming (GP) applications to a wide range of problems. However, little theory is currently available to explain such success and other related issues. Although it is often claimed that "Genetic programming may be more powerful than neural networks and other machine learning techniques," no results have been established to either prove or disprove such a statement. This paper attempts to address this fundamental issue in GP and presents some theoretical results.

2 Similarities and Differences Between GP and Feedforward ANNs

The essence of most artificial intelligence (AI) problems can be summarised as representation and search. We need a good representation and the corresponding search algorithm in order to have an efficient AI system.

2.1 Feedforward Artificial Neural Networks (FANNs)

FANNs can be regarded as a weighted, directed and acyclic graph. They have an input layer which takes the input data for a problem, an output layer which produce the output for the problem, and zero or more hidden layers in between. In FANNs, each node in a hidden layer or the output layer only takes inputs from nodes in previous layers. Each node implements a function, usually a nonlinear continuous one although this is not necessary. The training algorithm used by FANNs is the search engine to find a near optimal FANN.

2.2 Similarities Between GP and FANNs

Tree is a special case of graph. A GP tree can easily be mapped to an FANN. The input layer of the FANN consists of all the leaves of the tree. The output layer consists of the root. The hidden nodes in the FANNs consist of all the internal nodes of the tree. Both GP trees and FANNs are feedforward structures.

2.3 Differences Between GP and FANNs

In spite of similarities, there are some important differences between GP and FANNs. The most important one is that GP adapts *functions*, while FANNs mainly adapts connection weights and biases (which can be treated as a special case of weights). In other words, the node transfer function in FANNs is usually fixed (e.g., a sigmoid function). The adaptation of a FANN is normally achieved through weight adjustment. Learning in FANNs means weight adaptation. In a GP tree, the function implemented at each node is not fixed. It can be any one in the primitive function set or an ADF (automatically defined functions). It is determined by the evolutionary process. No weight is attached to a link in a GP tree. Learning in GP implies function (internal node) adaptation.

However, the above difference is not absolute because there are FANNs which use different node transfer functions and GP trees which use only one function. Nevertheless, there are not many such cases.

Other differences between GP and FANNs may be their learning algorithms and domains of applications. GP usually uses GAs, while FANNs have used back-propagation (BP), GAs, EP, etc. GP can deal with both symbolic and numerical tasks, while FANNs are mainly used in the numerical domain.

3 Universal Approximation by GP

The task faced by GP is the same as that faced by FANNs and other machine learning techniques, that is, given a set of input-output examples, find or approximate the true function underlying these examples. A natural question arises is that whether GP is capable of approximating such a function? The answer to this question is important because it establishes the power of GP in learning arbitrary functions theoretically.

Based on some existing results in FANNs [1, 2, 3], we can show that GP is capable of approximating arbitrary functions, provided its primitive function set or ADFs contain a Tauber-Wiener function (which may or may not be continuous) and a linear function. Common functions such as $\sin(\cdot)$ and $\cos(\cdot)$ are examples of a Tauber-Wiener function.

The above result shows that GP can approximate arbitrary functions under some very mild assumptions, since $\sin(\cdot)$ or $\cos(\cdot)$ (or some other sigmoid functions) have often been used in GP. It does not tell us, however, how to find such approximation. We still need to use GAs to search for it.

4 Conclusion

This paper shows universal approximation by GP. It answers theoretically how powerful GP is in learning (approximating) an arbitrary function given a set of training examples. The paper also points out the similarities and differences between GP and FANNs. A mapping between the two is given.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[2] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, pp. 551–560, 1990.

[3] H. White, "Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings," *Neural Networks*, vol. 3, pp. 535–549, 1990.