
An Evolutionary Algorithm for Neural Network Learning using Direct Encoding

Paul Batchis

PBATCHIS@CS.RUTGERS.EDU

Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019 USA

Abstract

This paper examines the use of evolutionary algorithms to evolve architectures and connection weights for artificial neural networks (ANNs). This approach to ANN learning is compared against the more traditional approach of using a fixed architecture, and learning the connection weights by back propagation. By demonstrating that the evolutionary approach to ANN learning can be as accurate and feasible as other approaches, this paper suggests evolutionary methods should be part of mainstream ANN research. This leads to the promising prospect that evolutionary approaches to ANN learning could be scaled up to more complex learning schemes that traditional ANNs cannot handle effectively. Possibilities include large ANNs, multi-layer ANNs, ANNs with novel architectures, ANNs made up of sub-ANNs, ANNs that use evolutionary techniques for their internal operation, and ANNs containing programs, human-coded or evolved. Biology suggests these types of ideas are important for representing complex concepts in brains. This paper takes a step in that direction by introducing an ANN encoding scheme for an evolutionary algorithm that make these ideas feasible.

1. Introduction

Artificial Neural Networks (ANNs) provide a practical way of representing a function such as a classifier when the input data is complex or noisy. Typically the ANN representation is learned by creating a fixed network architecture with random connection weights, then introducing training data to a back propagation algorithm that adjusts the connection weights until the desired accuracy is achieved.

This fixed-architecture back propagation learning method works well for simple functions of simple data. However, if ANNs are to be used to represent elaborate functions of very complex data, the network structure will need to be

more complex, in which case back propagation has several potential weaknesses:

- Since the architecture is constrained, there is no opportunity for the learning algorithm to find a better architecture for the problem. It might be desirable to have a different connection scheme between the nodes or a larger number of nodes, yet this cannot be learned.
- Back propagation becomes impractical if there are too many nodes or too many layers.
- Back propagation tends to converge to local maxima that may not be the global maximum.

An alternate approach to ANN learning that may not have these disadvantages is an evolutionary algorithm. Evolution is well suited for searching a large complex space, such as possible network architectures. Evolution is also less likely to get stuck in local maxima that are not the global maximum because of its strong tendency toward exploration.

In this paper, an evolutionary algorithm for ANN learning of binary classifiers is tested and compared to back-propagation. The algorithm creates a population of ANNs each with a random architecture and connection weights. Each member of the population is tested for fitness on a set of data for a classification problem. The best performing ANNs are selected to be copied and mutated for the next generation. This process is repeated for as many generations as necessary to find an ANN with the desired accuracy.

2. Background

There are many possible approaches to using evolution for neural network learning. Some that have been tried are listed below.

2.1 Evolution of Weights

A fixed architecture can be used, with the weights determined by a genetic algorithm. This approach has been investigated with success (Sasaki & Tokoro, 1999). It has advantages over back propagation: The global

space is more thoroughly searched. Also back propagation can be sensitive to the initial condition of the network weights.

2.2 Evolution of Architecture

Here, the goal is to use evolution to find an ideal architecture for a neural network. Once this is found, a technique such as back propagation can be used to find the correct weights. This approach has been shown to be effective (Harp & Samad, 1991). The idea is to use a fitness function that tests how well an architecture learns from the data. One drawback to this approach is that the fitness function can be costly to compute.

2.3 Simultaneous Evolution of Architecture and Weights

By evolving both the architecture and weights of the network, a fully working network with an ideal architecture can be learned without human interaction. This approach attempts to more fully take advantage of the power of evolution to handle many parameters. Although such complexity might seem too much for evolution to handle, this approach has been used successfully (Yao, 1999). This is the approach used for the experiments of the current paper.

3. The Learning Algorithm

The algorithm we implemented is a typical evolutionary (genetic) algorithm, except that instead of bit-strings (genetic codes), the population consists of independently functional ANNs. At each generation the ANNs are tested for fitness, and the best ANNs move into the next generation and make mutated copies of themselves. After many generations this process should tend to produce ANNs with increasing fitness.

The algorithm is described here in two parts: The ANN Encoding, and The Evolutionary Process.

3.1 The ANN Encoding

There are two general types of encoding strategies for the population members in an evolutionary neural network algorithm. They are known as *direct encoding* and *indirect encoding* (Curran & O’Riordan 2002). A direct encoding fully describes a functional ANN in all aspects including the connection weights. An indirect encoding only describes a method of assembling an ANN. The algorithm described here was designed to give the evolutionary process plenty of freedom to explore the space of possible ANNs without too many built in rules. Hence a direct encoding strategy was used.

This encoding is a fully functional ANN (one that can act as a binary classifier for some given problem), but the format of the ANN must lend itself to being randomly mutated, and mutated in a way such that a fully functional

ANN is always produced. This means that if a mutation changes the architecture (by adding or removing nodes or connections), the ANN must still function and produce output.

For operational simplicity, the ANN was designed to be a feed-forward network of threshold units. The number of nodes can vary, but the input nodes and output node always remain the same for a given problem.

Mutations can occur in one of three forms: connection weight changes, removal of connections, and addition of new nodes and connections. The form of a mutation is randomly chosen among these possibilities, each with equal probability.

The encoding strategy used here introduces a novel mechanism for allowing mutations to efficiently add new nodes in a very arbitrary way, while still preserving the feed-forward nature of the network. The way it works is every node is created with an immutable real-valued “level”: the input nodes are all level -0.1, the output node is level 1.1, and the other nodes have random values between 0 and 1. A node’s level is never changed, even during mutation. The rule here is that the direction of information flow between two nodes is always from the lower level node to the higher level node. Whenever a mutation (randomly) chooses two nodes to connect, it simply chooses the direction of the connection such that this rule is enforced. In case the two nodes have exactly equal level, the connection is simply not added (this should be rare except for the case of two input nodes.)

The idea of using real-valued levels was motivated by the desire to allow fully arbitrary feed-forward architectures to evolve. A standard multi-layer ANN could have been used, with each new node being created on a random layer. However the issue of connecting nodes on the same layer would require either a complex algorithm to prevent looping, or a ban on such connections that restricts the possible architectures. The approach used here handles this problem simply and elegantly.

3.2 The Evolutionary Process

For the evolutionary learning process, a population of random ANNs is created. The population size is set to 1000. Each member of the population is tested for fitness by how many of the training examples it classifies correctly. The less fit half of the population is destroyed and the next generation consists of the remaining members plus one mutated copy of each. This is a simple evolutionary process that guarantees fitness will never decrease.

Crossover was not used in this algorithm because it has been observed that crossover doesn’t work well for evolving ANNs due to what is known as the *Competing Conventions Problem* (Radcliff 1990). It may be worthwhile in the future to investigate how to work around this problem, but it is not dealt with here.

This process runs for 500 generations or until one of the ANNs achieves perfect classification. The most fit ANN of the final generation is the output of the learning algorithm.

4. Testing

4.1 Testing Methodology

To evaluate the viability of the algorithm, it was tested and compared to fixed-architecture ANNs with their weights learned by back propagation. The Weka Knowledge Explorer software package was used to test back propagation ANNs on three classification problems. The data for these problems is included in the Weka software. The problems chosen were those involving binary classification.

The back propagation algorithm was run using the default parameters of the software: number of hidden layers = attributes + classes; learning rate = 0.3; momentum = 0.2; training time = 500.

For each problem, testing for both methods was done by using 50% of the examples as training data, and 50% of the examples as test data. Validation to prevent overfitting was not used for these tests in order to avoid possible inconsistencies between our validation method and that of Weka.

4.2 Test Results

The test results for each method for each problem is expressed in terms of the percent of test examples correctly classified. On all three problems used for testing, the evolutionary ANNs equaled or outperformed the back propagation ANNs (see Figure 1).

It is possible that some overfitting took place. If so, the performance could be improved further by using validation techniques to prevent this.

In all of the test problems the number of nodes in the evolved ANNs was two to four times the number chosen by the algorithm used with back propagation.

Neural Network Test Results

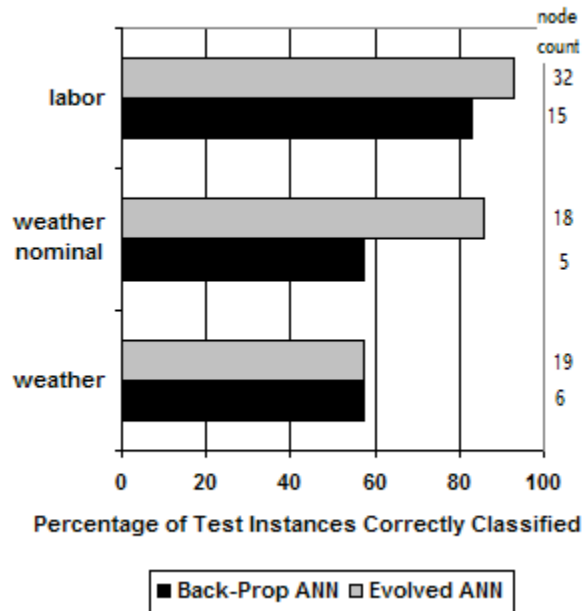


Figure 1. Chart showing performance of two ANN learning methods on each of three classification problems in terms of percentage of test instances correctly classified. The number of nodes in each network is listed at the right. Evolution consistently performed as well or better than Back Propagation.

5. Conclusion

The use of evolution for ANN learning is an appealing concept. It offers several key advantages over back-propagation. Evolution offers a straightforward general way of learning ideal network architectures along with their connection weights. Evolution tends to search the space of possible solutions globally, thus reducing the chance of getting stuck in local maxima. This paper demonstrates that evolution is a viable technique because it gets results comparable to or better than back propagation.

Perhaps the most promising aspect of using evolution in this way is that it may scale well to more complex representations. Some areas for future investigation are:

- Finding an effective way to take advantage of crossover.
- Evolving components and sub-components inside neural networks.
- Evolving rules for network mutation.
- Evolving more complex network components than simple neurons, such as human-coded or evolved algorithms.

The evolutionary paradigm is simple and flexible to use for ANN learning. The apparent success of evolution in

biology is indicative of its potential. The successful results of the experiments shown in this paper suggest this is an area worthy of further investigation.

References

- Curran, D., & O'Riordan, C. (2002). *Applying Evolutionary Computation to Designing Neural Networks: A Study of the State of the Art*. National University of Ireland, Galway.
- Gruau, F. (1995). *Automatic Definition of Modular Neural Networks*. *Adaptive Behavior*, 3(2):151-183.
- Harp, S., & Samad, T. (1991). *Genetic Synthesis of Neural Network Architecture*, In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 202-221. Van Nostrand Reinhold.
- Kohn, P. (1996). *Genetic Encoding Strategies for Neural Networks*. University of Tennessee.
- Radcliffe, N. (1990). *Genetic neural networks on MIMD computers*. Doctoral Dissertation, University of Edinburgh, Edinburgh, Scotland.
- Sasaki, T., & Tokoro, M. (1999). *Evolving Learnable Neural Networks under Changing Environments with Various Rates of Inheritance of Acquired Characters: Comparison between Darwinian and Lamarckian Evolution*, *Artificial Life*, 5(3):203-223.
- Whitley, D. (1995) *Genetic Algorithms and Neural Networks*. *Genetic Algorithms in Engineering and Computer Science*.
- Yao, X. (1999). *Evolving Artificial Neural Networks*. *Proceedings of the IEEE*, pages 1423-1447.
- Yao, X., & Liu, Y. (1998). *Making use of Population Information in Evolutionary Artificial Neural Networks*, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 28(3):417-425, June 1998.
- Yao, X., & Liu, Y. (1997). *A New Evolutionary System for Evolving Artificial Neural Networks*, *IEEE Transactions on Neural Networks*, 8(3):694-713, May 1997.