THE USE OF GENETIC ALGORITHMS AND NEURAL NETWORKS TO INVESTIGATE THE BALDWIN EFFECT

Michael Jones
Trinity University
San Antonio, Texas
mjones@trinity.edu

Aaron Konstam
Trinity University
San Antonio, Texas
akonstam@trinity.edu

KEYWORDS: genetic algorithms, neural networks, learning, Baldwin effect

Abstract

Standard evolutionary theory states that learned information will not be transferred into an underlying genotype. There is, however, a hypothesis that is consistent with the belief that learned behavior somehow influences the course of evolution. This hypothesis is called the Baldwin effect and it has been shown to occur in experiments with artificial life by Hinton and Nowlan and Ackley and Littman. A analysis was done of the effects of mutation and crossover rates on a computational model of the Baldwin effect which showed that this effect is most pronounced in asexual populations with low mutation rates. It was also noticed that the learning that occurred through the Baldwin effect exhibited the punctuated equilibrium behavior that is believed to be a part of all evolution.

1 INTRODUCTION

There is no physical mechanism for translating knowledge acquired during a single lifetime to the genetic code so that it will be available for genetic propagation via selection, crossover, and mutation. Indeed, the conception of a genotype composed of encoded learned behavior (a conception referred to as Lamarckian evolution) has been discredited in population genetics for nearly 100 years. There is, however, a hypothesis that is consistent with both the denial of Lamarckian evolution and the belief that learned behavior somehow influences the course of evolution. This hypothesis is called the Baldwin effect, after the nineteenth century geneticist Baldwin [2], and it remains a controversial topic even today. The Baldwin effect states that there are phenotypic tendencies rewarded in organisms that learn a certain skill, and that these rewards serve to change the criteria for fitness. Therefore, even though learning cannot directly affect the underlying genotype of an organism, the genetic makeup of the organisms that did the learning will in effect be rewarded. Individuals with these genes will, therefore, be favored for further evolution.

The Baldwin effect has been shown to occur in experiments with artificial life by Hinton and Nowlan [5] and Ackley and Littman [1]. However, showing that it occurs is only the first step in understanding how it operates. Mitchell [8] identifies the study of the Baldwin effect as an important future direction for research in evolutionary programming [8, page 183]. Likewise, Levy [7], Whitley [12], and Fogel [3] indicate that understanding the mechanics of the Baldwin effect should shed light on the importance (if any) of learning in evolution — artificial or otherwise.

SAC '99, San Antonio, Texas

2 IMPLEMENTATION

A simulation of the Baldwin effect consists of two major components: one, a model of the selective forces of evolution, and two, an analogous and biologically justifiable model of learning. The two components that have been used in this simulation are, broadly, the genetic algorithm and the artificial neural network learning to classify a set of input vectors. More specifically, we have used a GA with floating point allele values to model evolution, and a Kohonen Self-Organizing Feature Map [6, 4] to model learning.

Throughout the simulation we used a class hierarchy developed by Sutton and Santamaria at the University of Oklahoma [10]. This provided a convenient and clear conceptual paradigm in which the functionality of the simulation was separated into logical components.

The simulation was divided into three parts: the Agent (the collection of neural nets and genetic algorithms), the Environment (the input space of n-dimensional vectors) and a simulation manager that takes care of the communication between the two. Figure 1 [10] gives a graphical depiction of how the simulation was structured. The algorithm used for

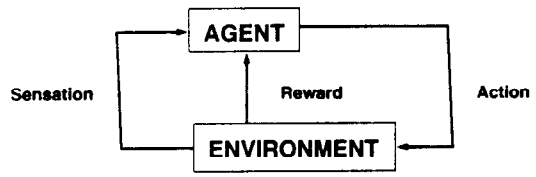

Figure 1: Agent-Environment Model

the simulation is described below ignoring most of the low level details of the implementation.

1. Create a population of m Kohonen Self-Organizing Feature Maps with parameters specified by the user.
2. Access the input space of n-dimensional vectors.
3. Create a manager responsible for handling communication between agents and the input vectors.
4. Run the simulation for a specified number of iterations while the nodes learn to classify the input vectors.
5. Choose agents that took the fewest number of learning iterations (i.e., the fittest) and create a new population according to the operations of selection, mutation, and crossover. These operations are applied to the initial weights of the neural network (as opposed to the weights as optimized through the Kohonen algorithm).
6. Record statistics on the old population (average fitness, best fitness, representative chromosome schemas, standard deviation in fitness, etc.).
7. Repeat steps 3-6 for a specified number of generations.

The hope was that, in the later trials of the simulation, the neural networks would require fewer learning iterations to learn the task, resulting in an increased average fitness among the population. This is what occurred.

3 KOHONEN SELF-ORGANIZING MAP

The Kohonen Self-Organizing Map (SOM) designed by Tuevo Kohonen [6] is a variation of the traditional Artificial Neural Network. It is a third generation neural network, meaning that many of its functional characteristics are thought to mirror those found in biological fact.

An SOM consists of a collection of nodes of neurons that are each connected to every other node and each node has associated with it a set of input weights w. The SOM also has associated with it a metric for determining which nodes are in the neighborhood N of a given node.

When the network is presented with a vector x_i at its input, it computes the neural response s_j of the node j using the formula:

$$s_j = w_j \cdot x_i \quad (1)$$

Normalize both w_j and x_i before computing the dot product, s_j, and refer to the node that produces the largest value of s as node k. Since the dot product of the normalized w_k and x_i vectors is the cosine of the angle between them, we can conclude that the winning node is the one with the weight vector closest to the input vector in its spatial orientation. We can then say that node k giving the largest s is closest to recognizing the input vector. We allow the nodes to learn by applying a $\triangle w$ to their weights using the formula:

$$\triangle w_k = \alpha\ (x_i - w_k) \qquad (2)$$

where α is a constant in the range [0,1] called the learning constant. The learning process is applied to the maximum response neuron and neurons in its defined neighborhood.

This training process can be described by the following algorithm:

1. A cycle: for every input vector x_i
   (a) Apply vector input to the network and evaluate the dot products of the normalized weights on each node and a normalized input vector. Call these dot products s.
   (b) Find the node k with the maximal response s_k.
   (c) Train node k, and all the nodes in some neighborhood of k, according to the learning equation above.
   (d) Calculate a running average of the angular distance between the values of w_k and their associated input vectors.
   (e) Decrease the learning rate, α.
2. After every M cycles, called the period, decrease the size of the neighborhood N.
3. Repeat steps 1-2 for some finite period of time or until the average angular distance calculated above is below a certain tolerance.

The effect of this process is to train the SOM to classify the input vectors into groups that will be characterized by particular values of w_k. In our simulations we used 10 input vectors and M, the number of cycles in a period, was also 10.

4 HARDWARE AND SOFTWARE PACKAGES USED

This simulation makes extensive use of GALib, a library of genetic algorithms and statistical tools [11]. GALib is a thoroughly documented, production quality C++ class library. In addition to making several very basic functions (crossover, mutation, selection) much easier to implement, the library also provides basic statistical analysis of the results. The implementation of the Kohonen Self Organizing Feature Map was based on the presentation found in Rao[9].

5 RESULTS

The first experiment was designed to gauge the effects of different mutation rates upon the emergence of the Baldwin effect. 3-dimensional input vectors were used to train the SOM. Data on the effects of each mutation rate was collected from 20 trials of 1,000 generations each. Mutation rates (p_m) of 0.001 to 0.01 in increments of 0.001 were used. Then the average learning iterations per generation (ALI) was plotted against mutation rate. The value of ALI plotted was the average of the 20 trials for each value of p_m. In all the runs the crossover probability (p_c) was kept constant at 0.2. The results are plotted in Figure 2. We observe in this figure that average fitness

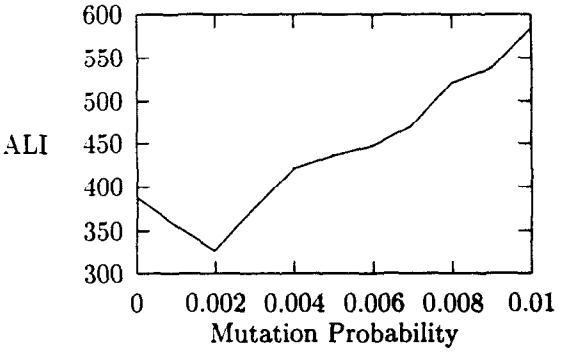

Figure 2: ALI vs. Mutation Probability

decreased as the mutation rate increased in a roughly linear fashion. Fitting the results to a linear equation

using the least square method yielded the following equation:

$$ALI = 267788p_m + 301 \quad (3)$$

A subsequent run of the simulation showed that the equation's predictions were correct to within 8% of the actual data points.

Other simulations using input vectors of higher dimensionality produced similar linear equations whose predicted values are consistently within 10% of the actual data. The equations produced are given below:

4-dimensional:

$$ALI = 27512p_m + 277 \quad (4)$$

5-dimensional:

$$ALI = 27280p_m + 298 \quad (5)$$

6-dimensional:

$$ALI = 29100p_m + 306 \quad (6)$$

The rates of deterioration of fitness (the slope terms) were consistently within 2% of each other — even when the search space was expanded from 3 to 6 dimensions. In each case, fitness was found to be a linearly decreasing function of mutation rate.

The effect of higher crossover rates on the Baldwin effect is similar to that of increasing the mutation rates. Again using the least squares approximation algorithm, we arrive at the following linear fit for the 3-dimensional vector case ($p_m = 0.001$):

$$y = 526p_c + 262 \quad (7)$$

for $0 \leq p_c \leq 1$. As in the case of mutation, increasing the probability of crossover decreases the ability of the network to learn. In fact, the individuals with the least number of learning iterations were part of an asexual population ($p_c = 0$). For 400 trials of 1000 generations each, by far the highest performing population operated with $p_c = 0$ and $p_m = 0.001$. In other words, most of the optimization performance of the algorithm seemed to be coming from straight selection with low mutation rates. Doing similar simulations with higher dimensional input vectors led to similar results. The linear equations produced by least square analysis are as follows:

4-dimensional:

$$ALI = 531p_c + 273 \quad (8)$$

5-dimensional:

$$ALI = 554c + 293 \quad (9)$$

6-dimensional:

$$ALI = 592c + 301 \quad (10)$$

As before, the behavior of the learning process was very similar in all the cases without regard to the dimensionality of the input vectors.

We performed one further simulation of this system with a $p_c = 0.2$ and a $p_m = 0.0$. The ALI obtained was 388. This value clearly does not fall on the line pictured in Figure 2. This value of ALI lies between those obtained using a p_m of 0.003 and 0.004. It is also 50 points higher than the ALI obtained at $p_c = 0.2$ and a $p_m = 0.001$. From this we believe it can be concluded that although the Baldwin effect can operate effectively in an asexual environment its efficiency is decreased by the absence of a small quantity of mutative pressure.

6 THE BALDWIN EFFECT AND EVOLUTION

Hillis [7, pages 204-208] points out that biologists currently believe that evolution does not proceed as a steady hill climbing process. "Evolution moves by leaps and bounds, alternating with periods of stasis. Species remain in virtual equilibrium where fitness is suited to the environment. Then a sudden change in the environment, or an empowering mutation, causes an abrupt jump in fitness, as new and effective physical characteristics express themselves in the phenotype of the species."

Hillis was able to produce and study this behavior in his experiment with artificial organisms. What he found was that while the population seemed to be resting as far as its phenotype were concerned,

the underlying genetic makeup was actively evolving. Eventually, enough genetic change occurred to produce the phenotype change through epistasis. This latter change was the abrupt change seen in the fitness of the species.

In our studies of the Baldwin effect we have observed similar evolutionary behavior. Figure 3 shows the behavior of one learning run with $p_m = 0.002$. As can be seen in this figure, the improvement in the

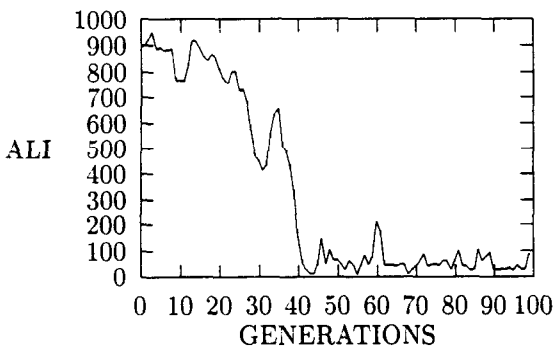

Figure 3: ALI vs. generations ($p_m = 0.002$)

learning rate is not a smooth change. The value of ALI oscillates moving slowly downward. But suddenly (i.e., in relatively few generations) it decreases sharply. Finally, it returns to a state of relative equilibrium.

7 CONCLUSIONS

This research has indicated one way in which evolutionary factors modify the Baldwin effect. It was shown that the effect is most apparent in asexual evolution with low mutation rates. It was also noticed that the learning that occurred through the Baldwin effect exhibited the punctuated equilibrium behavior that is believed to be a part of all evolution.

References

[1] ACKLEY, D., AND LITMAN, M. Interactions between learning and evolution. In *Proceedings of the Second Conference on Artificial Life* (Reading, MA, 1991), Addison-Wesley, pp. 487–509.

[2] BALDWIN, J. *Development and Evolution: Including Psychological Evolution, Evolution by Orthoplasy and Theory of Genetics.* AMS Press, New York, NY, 1990.

[3] FOGEL, D. *Evolutionary Computation.* IEEE Press, New York, NY, 1995.

[4] HAGEN, M., DEMUTH, H., AND BEALE, M. *Neural Network Design.* PWS, Boston, MA, 1995.

[5] HINTON, G. E., AND NOLWAN, S. J. How learning can guide evolution. *Complex Systems 1* (1987), 495–502.

[6] KOHONEN, T. *Self-Organizing Maps.* Springer-Verlag, Berlin, Germany, 1995.

[7] LEVY, S. *Artificial Life: A Quest for New Creation.* Pantheon Books, New York, NY, 1992.

[8] MITCHELL, M. *Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.

[9] RAO, V., AND RAO, H. *C++ Neural Networks and Fuzzy Logic.* IDG Books, Foster City, CA, 1995.

[10] SANTAMARIA, J., AND SUTTON, R. A standard interface for reinforcement learning, 1996. [http://envy.cs.umass.edu/~rich/RLinterface/RLinterface.html].

[11] WALL, M. Galib, a collection of genetic programming components, 1998. [http://lancet.mit.edu].

[12] WHITLEY, D., GORDON, V. S., AND MATHIAS, K. Lamarckian evolution, the baldwin effect and function optimization. In *Parallel Problem Solving from Nature-PPSN III.* (1994), Y. Davidor, H. Schwefel, and R. Manner, Eds., no. 866 in Lecture Notes in Computer Science, pp. 6–15.