INCREMENTAL EVOLUTION OF TRAINABLE NEURAL NETWORKS THAT ARE BACKWARDS COMPATIBLE

Chris Christenson, MS
Department of Computer Science
Southwest Texas Junior College
Uvalde, Texas 78802
cpchristenson@yahoo.com

Khosrow Kaikhah, Ph.D.
Department of Computer Science
Texas State University
San Marcos, Texas 78666
kk02@TxState.edu

ABSTRACT: Supervised learning has long been used to modify the artificial neural network in order to perform classification tasks. However, the standard fully-connected layered design is often inadequate when performing such tasks. We demonstrate that evolution can be used to design an artificial neural network that learns faster and more accurately. By evolving artificial neural networks within a dynamic environment, the artificial neural network is forced to use learning. This strategy combined with incremental evolution produces an artificial neural network that outperforms the standard fully-connected layered design. The resulting artificial neural network can learn to perform an entire domain of tasks, including those of reduced complexity. Evolution alone can be used to create a network that performs a single task. However, real world environments are dynamic and thus require the ability to adapt to changes.

KEY WORDS: incremental evolution, neural networks, training, backwards compatible

1 Introduction

Genetic Algorithms have been applied to the design of Artificial Neural Networks (ANNs) in several ways. Evolution of ANNs was initially used to optimize the set of weights. With a pre-established architecture, mutation and crossover were performed on the connection weights of the ANN. Evolution has also been applied to the search for optimal architecture in which mutation includes either neuron addition from a small initial network or neuron deletion from a large initial network. Finally, evolution has been utilized to search for optimal learning parameters. Mutation and crossover were performed on the learning parameters of each connection in the ANN within the pre-established architecture. [1]

Most of the research on the evolution of ANNs has focused on the search for optimal weights. Researchers have avoided evolving structure due to the difficulty of performing crossover operations on complex ANNs. In order to perform evolution, the crossover operator must be able to combine two highly performing networks in a meaningful way. Extensive analysis of the neurons and their connection weights has to be performed in order to determine which weights contribute to the desired outputs. Thus, until recently, evolution of ANNs was limited to weights and learning parameters.

Caruana et.al. demonstrated that the architecture affected the speed and accuracy of learning.[2] Furthermore, evolving the structure removes the trial and error approach widely used to determine the number of hidden nodes for any given problem. Finally, evolution of structure and weights was shown to create networks with high performance and minimal structure.

We used evolution to create ANNs that can adapt to perform any task within the environment it was evolved in. This functionality is imperative for the future of artificial life since organisms do not live in isolation. The world is continuously changing and the ability to adapt to change will provide for more robust artificial life. We will demonstrate that evolution can be applied to design an artificial neural network that has the ability to adapt to drastic changes in its environment in an incremental manner.

2 Combining Evolution and Learning

Learning's fundamental purpose is to facilitate adaptation to a changing environment. The Baldwin Effect describes learning as smoothing the fitness curve so that evolution can climb it with less difficulty.[3] This difficulty arises when the environment changes. Thus, without a dynamic environment, learning has little purpose. This is shown in the second aspect of the Baldwin Effect in which the genome acquires the traits as instinct that previously had to be learned.[4, 5] So, in order to find the optimum design for an ANN to learn, evolution must take place in a dynamic environment.

Forcing an ANN to perform many tasks drives evolution to optimize the ANN for learning. A network's fitness in a changing environment is based upon the network's ability to learn. After many generations, the fittest network will be able to adapt to any problem in its environment. The final result of this process is an evolved network that has the potential to perform tasks it has never seen before. Furthermore, we hypothesize that

evolution can produce an optimized network that can be trained to perform a new task faster and more accurately than a traditional, fully-connected layered network.

2.1 Process of Evolving Trainable Networks

We chose the problem of graphing high-degree polynomials to determine whether a network could be evolved to learn many problems. By simply changing the degree of the polynomial it was possible to create varying degrees of complexity. Thus, we have named our evolving ANNs Polygraphers.

In order to create an environment that forces evolution toward learning rather than specialization, we set each Polygrapher to graph five significantly different, complex 3rd degree polynomials. Each Polygrapher was given a certain number of iterations to learn to graph each polynomial. After the Polygrapher had a chance to graph a polynomial for the preset number of iterations, the Polygrapher's network was reset to the initial weights stored in its genome. This is important to the evolution of a trainable network because it has been shown that Back-propagation is sensitive to the initial weights.[6] With the weights reset after each polynomial, the Polygrapher was evaluated on its ability to graph each polynomial from the same starting point. Furthermore, the initial weights were stored in the Polygrapher's genome and therefore passed down to the following generation. When fitness was evaluated based on the network's ability to graph from a defined starting point, evolution was able to optimize a network to learn to graph any polynomial. Without a defined starting point, training would be hampered by the modified weights since the weights from the previous polynomial were specialized to that polynomial. The Baldwin Effect further emphasizes the need to reset the weights to an initial starting point. The Baldwin Effect states that evolution does not pass on learned behavior, rather it passes on the ability to learn.[7] After the population of Polygraphers had a chance to be trained to graph each polynomial, the Polygrapher's fitness was calculated.

3 Results & Analysis

By implementing the Evolving Trainable Neural Networks for graphing polynomials, we demonstrated that evolution can be used to design more efficient ANNs to graph polynomials than the standard fully-connected ANNs. We determined that stepwise evolution can be used to design ANNs to graph very complex polynomials. Finally, we analyzed the ANN's ability to learn to graph less complex polynomials than those it was exposed to during evolution.

3.1 Evolution of a 3rd Degree Polygrapher

We evolved a Polygrapher for graphing 3rd degree polynomials. The fitness of Polygraphers reached a saturation level of 80% after 775 generations, Figure 1. The resulting network has 6 hidden nodes and 13 connections, Figure 2. Each connection has an evolved initial weight and learning rate.

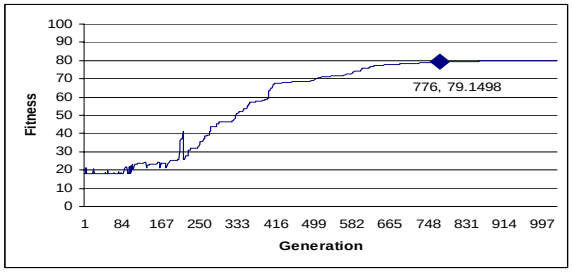

Figure 1: Fitness graph of the evolution of a Polygrapher neural network on 3rd degree polynomials.

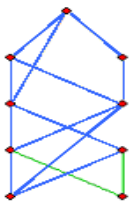

Figure 2: ANN Evolved for 3rd degree polynomials

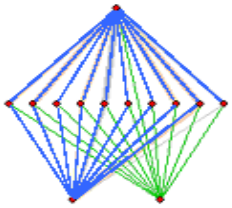

Figure 3: Fully-Connected ANN

The 3rd degree Polygrapher was able to graph each of the five polynomials in its lifetime within a total error of 3 units. However, success was not only dependant upon the ability to graph the third degree polynomials. In order to meet all the success criteria, the Polygrapher was required to accurately graph the five polynomials quickly. Since speed was relative to the complexity of the problem, we compared the speed of the Polygrapher to graph the 3rd degree polynomial with the traditionally designed fully-connected ANN, Figure 3. The evolved network was able to achieve an acceptable error (less than 3 units) 400 iterations faster than the fully connected network, Figure 4.

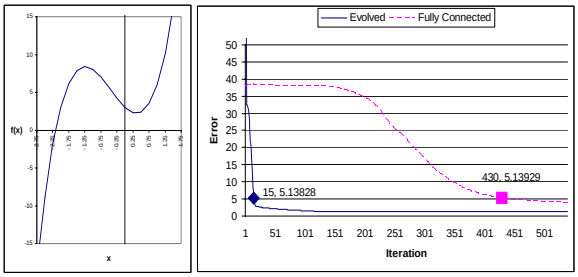

Figure 4: Back-Propagation error reduction graph on the 3rd degree polynomial, $f(x) = 3x^3 + 4x^2 - 4x + 3$. The evolved network was evolved to learn to graph 3rd degree polynomials. The evolved network was introduced to this polynomial during evolution.

Evolution was able to determine a near optimal design including hidden nodes, connections, initial weights, and learning rates. Clearly, the evolved design was more accurate and faster at graphing polynomials than the traditional design. However, was it more versatile than the traditional design? The previous results were based

on a polynomial that was included in evolution. Therefore, it was expected that the evolved network would be good at graphing a polynomial that it was evolved to graph. Thus, an additional test of the evolved network included a polynomial that it did not encounter during evolution.

When the evolved network was compared to the traditional network on a previously unseen 3rd degree polynomial, the evolved network outperformed the traditional network, Figure 5. For the previously unseen polynomial, the evolved network was able to achieve an acceptable error (less than 3 units) after 14 iterations, 60 iterations faster than the fully-connected network. Furthermore, it achieved a more accurate classification with only half the total error of the fully-connected network even after 800 iterations. Thus, we concluded that evolution can be used to design a faster, more accurate and more versatile network than traditional design techniques.

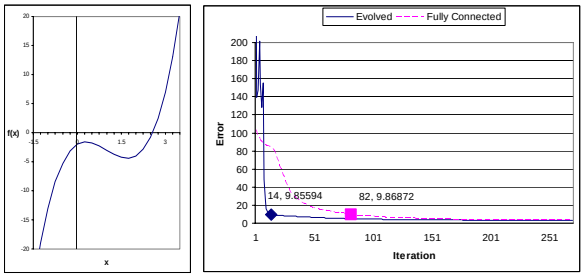

Figure 5: Back-Propagation error reduction graph on the 3rd degree polynomial $f(x) = 2x^3 - 6x^2 + 3x - 2$. The evolved network was evolved to learn to graph 3rd degree polynomials. The evolved network never encountered this polynomial during evolution.

3.2 Incremental Evolution

We attempted to evolve a Polygrapher to graph a more complex polynomial, 4th degree, starting with the minimal structure. The evolution process required many more generations to achieve marginal fitness, Figure 6. After 1200 generations, the fitness remained at 60%.

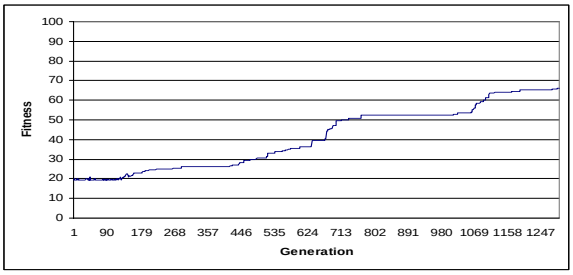

Figure 6: Evolution of a 4th degree Polygrapher starting from minimal structure

Evolution works well under conditions in which it can improve its fitness incrementally rather than abruptly. We hypothesized that it would take fewer generations to evolve networks to solve more complex problems if we started with a set of networks which had evolved to solve less complex problems rather than a set of minimal structures. Therefore, rather than evolving the 4th degree Polygrapher from a minimal structure, we began evolution from a population of previously evolved 3rd degree Polygraphers, Figure 7.

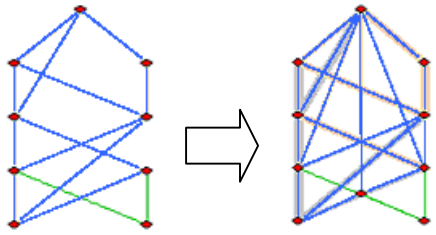

Figure 7: Structure of a 4th degree Polygrapher incrementally evolved from 3rd degree Polygraphers

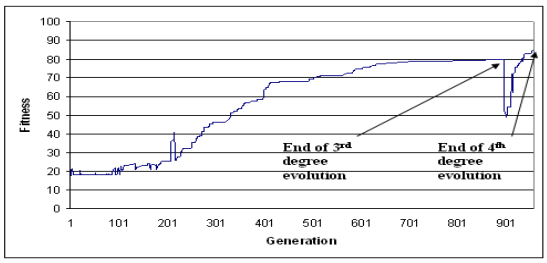

Figure 8: Incremental evolution of a 4th degree Polygrapher starting from a population of 3rd degree Polygraphers

The fitness level of 80% for 3rd degree Polygraphers dropped to 50% when the network started to graph 4th degree polynomials, Figure 8. This was expected because the Polygraphers had evolved the structure necessary to graph less complex, 3rd degree polynomials. However, after only 60 generations, the fitness level of the Polygraphers rose to above 80%. This was a dramatic reduction in the number of generations compared to Polygraphers starting from a minimal structure. Incremental evolution achieved a fitness level of over 80% in 1000 generations, while evolution from minimal structure only achieved a fitness level of 66% in 1200 generations.

We continued our study of the effects of incremental evolution by evolving 4th degree Polygraphers from a population of 2nd degree Polygraphers. Once again, the fitness dropped from 85% to 25% when presented with 4th degree polynomials, Figure 9. This drop was more dramatic than the drop resulting from a 3rd degree Polygrapher being introduced with 4th degree polynomials. This was a direct result of the 2nd degree Polygraphers' structure. Since 2nd degree polynomials are less complex than 3rd degree, the 2nd degree Polygraphers were evolved to contain less structure. However, the number of generations required to achieve high performing 4th degree Polygraphers was comparable for 2nd and 3rd degree incremental evolution. Whether evolution began from 2nd or 3rd degree, the number of generations required to evolve 4th degree Polygraphers

was approximately 1000 generations. This was due to the fact that a 2nd degree Polygrapher takes less generations (approximately 500) than the 3rd degree Polygrapher (approximately 900), and therefore had more generations to evolve to the 4th degree.

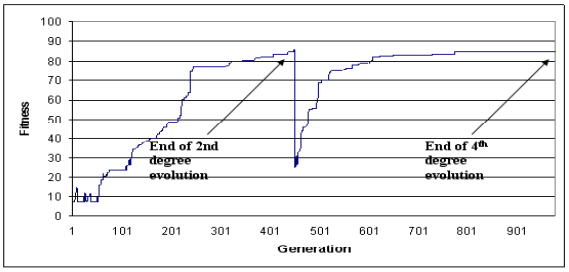

Figure 9: Incremental evolution of 4th degree Polygraphers starting from a population of 2nd degree Polygraphers

To complete our analysis of incremental evolution, we evolved 4th degree Polygraphers incrementally starting with 2nd to 3rd degree, Figure 10. As expected, there was a drop of fitness from the 2nd degree to the 3rd degree and again from the 3rd degree to the 4th degree, Figure 11. The result of incremental evolution through 2nd and 3rd degree was high performing 4th degree Polygraphers in approximately 200 less generations than the incremental evolution starting from either 2nd or 3rd degree respectively. Incremental evolution achieved high performance in fewer generations because it was able to evolve Polygraphers for a less complex polynomial first and then build upon that structure.

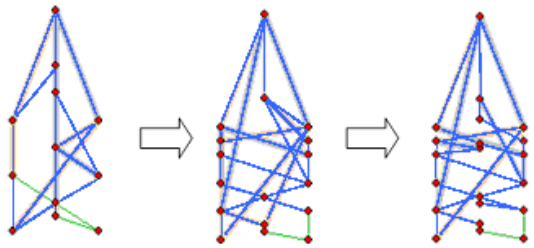

Figure 10: Structure of a 4th degree Polygrapher incrementally evolved from 2nd and 3rd degree Polygraphers

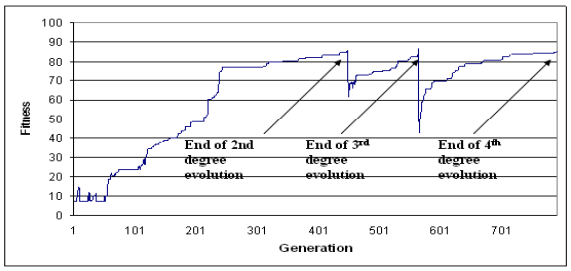

Figure 11: Incremental Evolution of a 4th degree Polygrapher starting from 2nd to 3rd degree Polygraphers

Incremental evolution is clearly a major improvement in terms of the number of generations required to achieve high fitness. It allowed for the evolution of more complex classification tasks by first evolving for less difficult tasks. The evolved 4th degree Polygraphers were able to graph 4th degree polynomials 1000 iterations faster than the fully connected network and did so with less error, Figure 12. We concluded that incremental evolution can be used to design a faster, more accurate and more versatile network for very complex tasks.

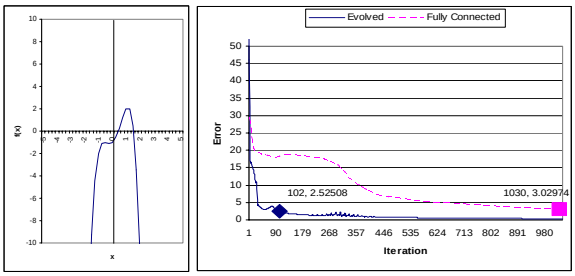

Figure 12: Back-Propagation error reduction graph of on the 4th degree polynomial, $f(x) = -2x^4 + x^3 + 3x^2 + x - 1$. The evolved neural network was evolved to learn to graph 4th degree polynomials. The evolved network never encountered this polynomial during evolution.

3.3 Backwards Compatibility

We were interested in determining how well networks which were evolved to graph more complex polynomial were performing when exposed to less complex polynomials. We call this feature backwards compatibility. As expected, the 3rd degree Polygrapher was able to graph 2nd degree polynomials with very little error. Furthermore, it was able to graph the 2nd degree polynomial faster and more accurately than a fully connected network, Figure 13. However, the 3rd degree Polygrapher was not able to graph a 4th degree polynomial as accurately as the fully-connected network without incremental evolution, Figure 14.

We expected an even better backwards compatibility when analyzing the incrementally evolved 4th degree Polygrapher since it had experienced 3rd degree polynomials previously. The incrementally evolved 4th degree Polygrapher was indeed able to graph the 2nd and 3rd degree polynomials faster and more accurately than the fully-connected network, Figures 15 and 16.

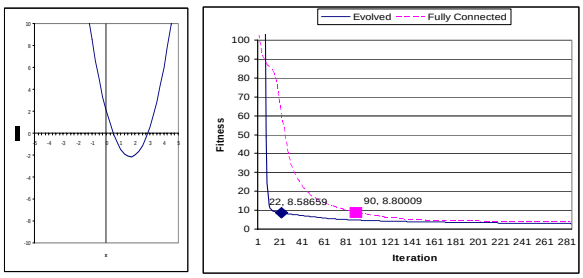

Figure 13: Back-Propagation error reduction graph on the 2nd degree polynomial, $f(x) = 1.5x^2 - 5x + 2$. The evolved neural network was evolved to learn to graph 3rd degree polynomials. The evolved network never encountered this polynomial during evolution.

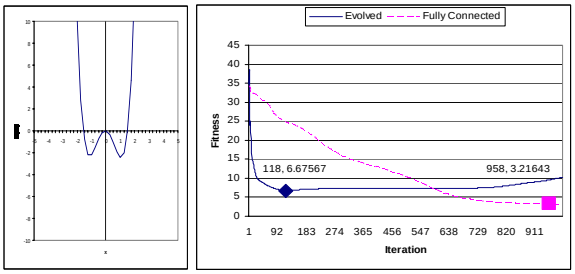

Figure 14: Back-Propagation error reduction graph on the 4th degree polynomial, $f(x) = 1.7x^4 + .3x^3 - 4x^2 - .4x$. The evolved neural network was evolved to learn to graph 3rd degree polynomials.

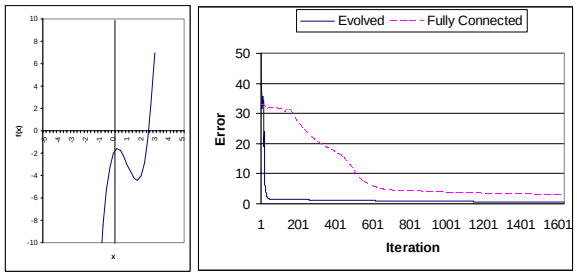

Figure 15: Back-Propagation error reduction graph on the 3rd degree polynomial, $f(x) = 2x^3 - 6x^2 + 3x - 2$. The evolved neural network was evolved to learn to graph 4th degree polynomials. The evolved network never encountered this polynomial during evolution.

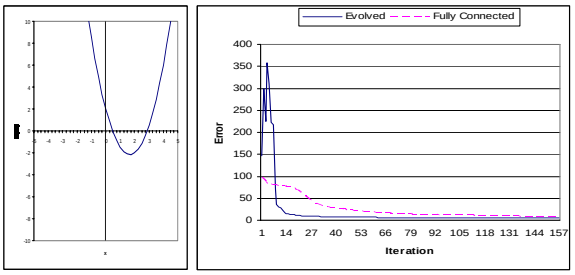

Figure 16: Back-Propagation error reduction graph on the 2nd degree polynomial, $f(x) = 1.5x^2 - 5x + 2$. The evolved neural network was evolved to learn to graph 4th degree polynomials. The evolved network never encountered this polynomial during evolution.

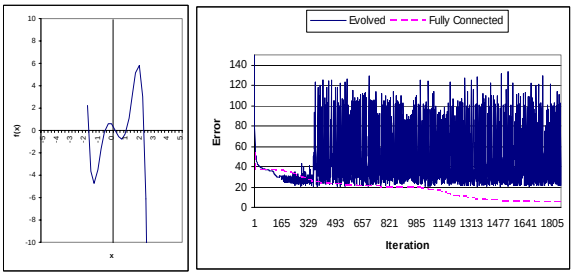

Figure 17: Back-Propagation error reduction graph on the 5th degree polynomial, $f(x) = -1.1x^5 + 1.7x^4 + 4.1x^3 – 4.2x^2 - 1.4x + .6$. The evolved neural network was evolved to learn to graph 4th degree polynomials.

However, the 4th degree Polygrapher was not able to accurately graph a 5th degree polynomial, Figure 17. Again, this is due to evolution's selection of only the minimal necessary structure.

Further examination of backwards compatibility revealed an interesting result. While incremental evolution enables networks to quickly learn problems of lesser complexities, as they get more complex they lose some of their ability to learn simpler problems. As depicted in Figure 18, when the neural network evolved to graph 4th degree polynomials learns to graph a 2nd degree polynomial, its error does not reduce as quickly as the neural network evolved to graph 3rd degree polynomials. This suggests that backwards compatibility may be a function of the difference in complexity of the task.

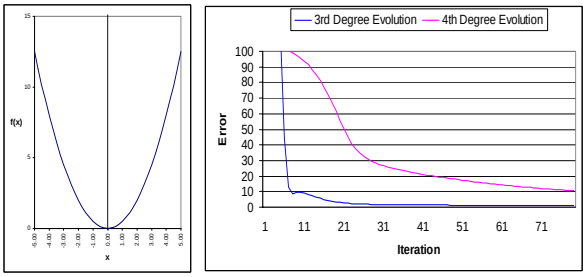

Figure 18: Back-Propagation error reduction graph on the 2nd degree polynomial, $f(x) = .5x^2$. The graph compares the performance of a network evolved to learn to graph 3rd degree polynomials and a network evolved to learn to graph 4th degree polynomials.

4 Conclusion

There are many difficulties associated with designing artificial neural networks to use Back-propagation. Determining the number of hidden nodes and even hidden layers has lead to many ad-hoc algorithms that deal with complexity analysis or simply trial and error. Our goal was to find a method for designing a network that could be used on many different learning tasks. This method would have to be able to determine the number of hidden nodes, initial weights, and the learning rates. Furthermore, the desired method would create a network that could be applied to any problem within the desired domain.

The Evolving Trainable Networks approach applies evolution to ANNs that use Back-propagation in a dynamic environment. By setting the fitness function based on the networks ability to solve an array of problems from the desired domain, the networks were forced to use learning. The ability to learn was then optimized by setting the fitness function to increase as the learning time decreased.

When this method was applied with Incremental Evolution, networks evolved with better fitness and in fewer generations. In addition, the evolved networks outperformed the traditional networks in speed and accuracy. Furthermore, a network evolved for a complex task outperformed the standard design for less complex tasks. This shows that the evolved networks learn to solve complex tasks rather than specializing on a specific task. Thus, the best method for evolving networks is to start with a less complex task and incrementally evolve the network for more complex tasks. The ability to learn complex tasks while retaining the ability to learn less complex tasks should improve artificial neural network's contribution to the field of artificial life.

References

[1] K. O. Stanley and R. Miikkulainen, “Efficient Reinforcement Learning through Evolving Neural Network Topologies”, Evolutionary Computation, Vol. 10, pp. 99-127, 2002

[2] R. Caruana, S. Lawrence and C.L. Giles. "Overfitting in neural networks: backpropagation, conjugate gradient, and early Stopping". Advance in Neural Information Processing Systems, Vol. 13, pp. 402-408, 2001

[3] Mark J Baldwin, “A new Factor in Evolution”, American Naturalist, Vol. 30, pp. 441-451, 1996

[4] Peter Turney, “Myths and Legends of the Baldwin Effect”, Proceedings Workshop on Evolutionary Computation and Machine Learning at the 13th International Conference on Machine Learning, pp. 135-142, 1996

[5] Kim Sterelny, “The Baldwin Effect and Its Significance: A Review of Bruce Weber and David Depew (eds) Evolution and Learning: The Baldwin Effect Reconsidered” Evolution and Development, pp. 341-351, 2004

[6] Anil K. Enumulapally, Ligguo Bu, and Khosrow Kaikhah “Backpropagation: In Search of Performance Parameters”, WSEAS Transactions on Systems, Issue 2, Vol. 3, pp. 950-956, 2004

[7] E. J. W. Boers and I.G. Sprinkhuized-Kuyper, “Evolving Artificial Neural Networks using the Baldwin Effect”, Artificial Neural Nets and Genetic Algorithms, pp. 333-336, 1995