# Evolving a Strongly Recurrent Neural Network to Simulate Biological Neurons


Terence Soule
Computer Science
University of Idaho
Moscow, ID 83844
*tsoule@cs.uidaho.edu*

YingYin Chen
Computer Science
University of Idaho
Moscow, ID 83844
*chen1536@cs.uidaho.edu*

Richard B. Wells, Senior Member
Electrical Engineering
University of Idaho
Moscow, ID 83844
*rwells@mrc.uidaho.edu*



*Abstract*—
**In this research we use evolutionary techniques to evolve recurrent neural networks that produce a pulsed output when triggered by a constant valued input. Networks of several different sizes and configurations are successfully evolved demonstrating that this is a robust technique. The resultant networks can be used as approximations of certain types of biological neurons or of central pattern generators.**



*Keywords*— **Evolutionary computation, genetic algorithms, recurrent neural networks, Biomimetic Neurons**


## I. Introduction

There is considerable interest in the tendency of biological, neuronal systems to use periodic signals. Individual neurons produce pulse-coded outputs, with information encoded in the pulse frequencies. Small collections of neurons, called central pattern generators, produce periodic signals that are used to control repetitive motions such as walking, swimming, heartbeats, etc. in invertebrates.

Recurrent neural networks that produce periodic signals give researchers models of biological neurons that use pulse coded information and of central pattern generators. Combining multiple small networks with periodic outputs would produce a larger network that uses pulse-coded information. Alternatively, these small networks could be used in applications requiring periodic outputs, such as walking robots.

There are several significant features of this research. There is no periodicity in the neurons or the network's inputs. Thus, the periodic output of the networks is produced only by the recurrent nature of the networks themselves. The periodic output is turned on and off by the presence (or absence) of a constant valued input. Thus, for example, if these networks were used to control a walking robot, motion could be turned on and off by the simple application of a constant input signal to the network.

Finally, the size and complexity of the networks (up to 3 hidden layers of 7 nodes apiece)and the relatively long period of the output (20 time steps) would make it almost impossible to train these networks with more traditional methods such as backpropogation. Thus, the success of evolutionary techniques in evolving these networks is an important result.


This work is supported by NSF EPSCoR EPS-0132626.


## II. Background

Inspiration for this research comes from two types of biological, neuronal systems. The first are neurons that seem to use pulsed spike potential to convey information. Primary examples of such neurons are the burst and tonic neurons found in the human eye [1]. Burst neurons produce a relatively high frequency 'burst' of activations when they are triggered. Tonic neurons produce higher frequency pulsed signals for the duration of a triggering signal.

The second biological inspiration is central pattern generators. Central pattern generators are relatively small collections of neurons that interact to produce a periodic signal [1], [2], [3]. Examples of central pattern generators include heart rate control in some species of leach [2] and swimming motion in some species of sea slug [2], [3].

Evolutionary algorithms are search techniques based on natural selection. They apply analogs of selection, mutation and recombination to a population of individuals. These individuals represent potential solutions to a particular problem. Two common types of evolutionary computation are genetic algorithms (GAs) [4] in which the solutions are typically encoded as binary or integer strings and genetic programming (GP) [5], [6] in which the solutions are programs.

Considerable research has been done in applying evolutionary computation techniques to neural networks. Of particular interest are evolutionary approaches for evolving recurrent neural networks and finding network topologies, both of which are difficult problems for more traditional methods. A survey of papers on evolving neural nets is well beyond the scope of this work. (An indexed bibliography composed by Jarmo Alander included over 1000 works in the year 2000 [7].) What follows are the approaches that most directly contributed to this research.

In 1994 Angeline, Saunders and Pollack developed a successful approach for evolving both the topologies and weights of recurrent neural networks [8], [9]. Their algorithm, titled GNARL, is an evolutionary program rather than a GA in that it does not use crossover. Some of it's components have been adopted by other researchers [10], [11]. The flexibility of their neural network encoding makes this a very powerful approach.

The NEAT algorithm, developed by Stanley and Miikkulainen, is another approach to evolving the weights and topologies for neural networks [12], [13], [14]. Unlike GNARL, the NEAT algorithm uses both mutation and crossover, making it a variation of the standard GA. The NEAT algorithm begins with neural networks that contain no hidden nodes and adds them during the evolutionary process as necessary. New genes, which may represent either new nodes or new connections, are numbered allowing crossover to mix genes based on their historical origin.

Recently Kim and Moon used a GA with a two-dimensional chromosome to evolve the weights of a strongly neural network with a fixed topology [15]. The two-dimensional chromosome allows more complex crossover operations than a standard one-dimensional chromosome. These crossover operations appears to help preserve the relationship among the genes and improved the performance of the GA on a hand writing recognition problem.

## III. Experiment

Our goal is evolve a strongly connected recurrent neural network that has properties similar to those of biological neurons and central pattern generators. In this paper we use a GA to evolve a network that reproduces the simple pulsed output of a neuron that has been activated by the application of an external voltage.

Of particular interest is the size of the neural network required to mimic this fairly simple behavior. Thus, we experimented with networks with a variety of hidden layers and nodes per hidden layer. For each network we ran 50 separate trials of the GA starting with a random population of networks with random weights.

### *A. The Test Problem*

For this research we want to evolve a network that produces a simple pulsed output when an activation 'voltage' is applied to the networks input. The test lasts for 100 time steps. The activation 'voltage' is applied to the input node at 10 time steps and is turned off at 90 time steps. I.e. the input node is set to one from time step 10 to time step 90 and is set to zero for all other times.

While the input is high the goal is to produce a pulsed output with a period of 20 time steps (see Figure 1). The error is the sum of the absolute value of the difference between the desired output and the actual output at each time step.

### *B. The Neural Network*

Our neural network is a strongly connected recurrent network. It has a single input node and a single output node. The value at the output node represents the action potential of our simulated neuron. The number of hidden layers and nodes per hidden layer is fixed for each experiment, but different configurations are tested in seperate experiments. Eventually a modificantion of the GNARL or NEAT systems will be used to allow the network topology to evolve. Figure 2 shows a sample network with two hidden layers and two nodes per hidden layer. The nodes use a symmetric sigmoid activation function.

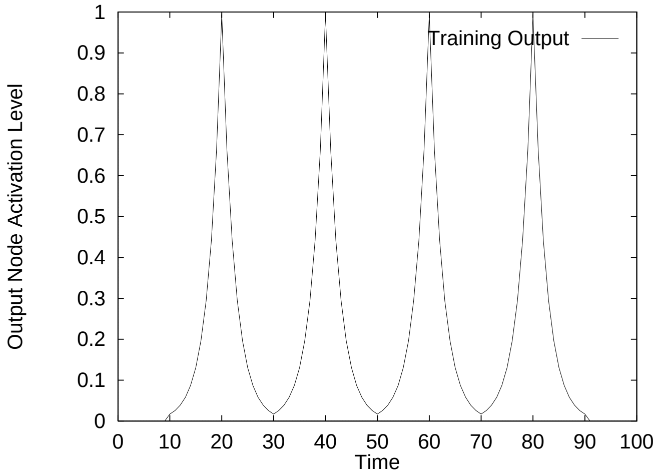


Fig. 1. Target output signal for the evolved network.

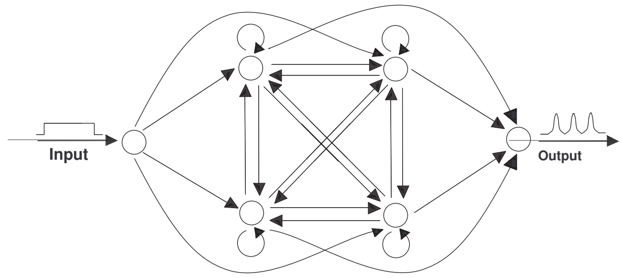


Fig. 2. A sample strongly connected recurrent network with two hidden layers and two nodes per hidden layer.

The algorithm for activating the neural network is as follows:

```
Set all node values to zero.
For each time step (1 to 100) do{
    Set the input node value.
    For each hidden layer do{
        For each hidden node in the layer do
            Calculate the activation level.
        For each hidden node in the layer do
            Update the activation level.
    }
    Calculate the output node's activation
level and return in.
}
```

At each time step the input node's activation level (0 or 1) is set. Then for each hidden layer the activation level of the nodes in that layer are calculated. The activation levels for a layer are updated to the new values only after all of the activation levels for that layer have been calculated. Then the process repeats for the next hidden layer.

Effectively the activation levels of the neurons in the same hidden layer are updated simultaneously. The activation levels of the neurons in different layers are updated sequentially.

It is worth noting that there is no source of periodicity within the network, nodes, or input. Thus, the only way to produce the desired output pattern is to use the recurrent connections within the network. Further, the desired output period is fairly long (20 time steps). This means that typical deterministic training algorithms, such as backpropogation, would find this problem almost impossible to solve.

### C. The Genetic Algorithm

A simple GA is used in these experiments. The GA uses a chromosome consisting of real values. Each real value corresponds to the weight between one pair of nodes (or between a node and itself). The chromosome size varies depending on the size of the network. For a network with L hidden layers and N nodes per hidden layers there are a total of $(LN)^2+2NL$ weights (each of the $LN$ hidden nodes is connected to every other hidden node and the input and output nodes are each connected to $NL$ nodes). Thus, the chromosome consists of $(LN)^2+2NL$ real values.

The GA is generational. It is run for 250 generations on a population of 500 individuals. The two best individuals are copied into the next generation (elitism). Tournament selection is used, with a tournament of size 3. The initial weights were randomly chosen in the range (-1.0,1.0). The mutation rate is $1/(LN)^2$. Mutation changes a weight by up to 25% of the weight's original value.

When crossover is applied to two individuals the same random (non-input) node is chosen in both individuals. The weights of all connections leading *into* the chosen node are exchanged between the two individuals. Other crossover operations are possible, such as exchanging all output weights or all of the weights between two layers, but they were not tested in these experiments. The crossover rate is 0.8.

## IV. Results

Figure 3 shows a 'successful' and an 'unsuccessful' trial using a network with 2 hidden layers and 5 nodes per hidden layer. The successful trial evolves a network that produces periodic output with the desired frequency, whereas the unsuccessful trial fails to produce periodic behavior. However, even in the unsuccessful trial the network correctly marks the beginning and end of the input signal and it evolved a constant output value that tends to minimize the total error. This type of output

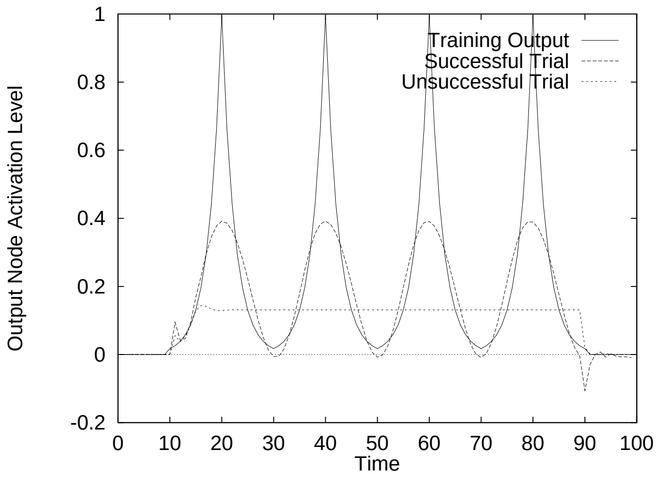


Fig. 3. Network output for two sample trials, showing a successful and an unsuccessful trial. The network used had 2 hidden layers and 5 nodes per layer.

TABLE I

Number of successful trials (out of fifty) for each network size. Layers refers to the number of hidden node layers.

| Layers | 1 | 2 | 3 |
|---|---|---|---|
| 1 Node/Layer | 0 | 0 | 0 |
| 3 Nodes/Layer | 0 | 3 | 10 |
| 5 Nodes/Layer | 0 | 5 | 27 |
| 7 Nodes/Layer | 0 | 19 | 29 |

signal (a constant value that turns on and off with the activation signal) was seen in every trial that failed to evolve periodic behavior. Thus, it is clear that this 'unsuccessful' behavior represents a very large, easy to find, local optimum for this problem.

The successful trial in Figure 3 clearly shows that a small recurrent network can be evolved to produce a neuron-like pulsed output that turns on and off based on a constant valued input.

Table I shows the number of successful trials (out of fifty total trials) for each of the network sizes. There is a clear correlation between both the number of hidden layers and the number of nodes per hidden layer and the probability of finding a successful solution.

We also tried evolving weights for networks with one hidden layer of nine nodes. For that network two of the fifty trials produced output with the correct period. Finally, we tried networks of nine and fifteen layers with one node per hidden layer. The networks with nine hidden layers were unsuccessful in every trial. The networks with fifteen hidden layers were successful in eight of the fifty trials. These results suggest that success is primarily dependent on the number of nodes. However, the networks with more uniformly distributed nodes (e.g. 3x3) seemed to work best.

In addition to the successful and unsuccessful trials shown in Figure 3, several other interesting behaviors were observed. In many of the trials a network evolved that successfully matched the first cycle, but failed to match the other 3. Figure 4 shows output from one such network, which had 2 hidden layers and 5 nodes per hidden layer.

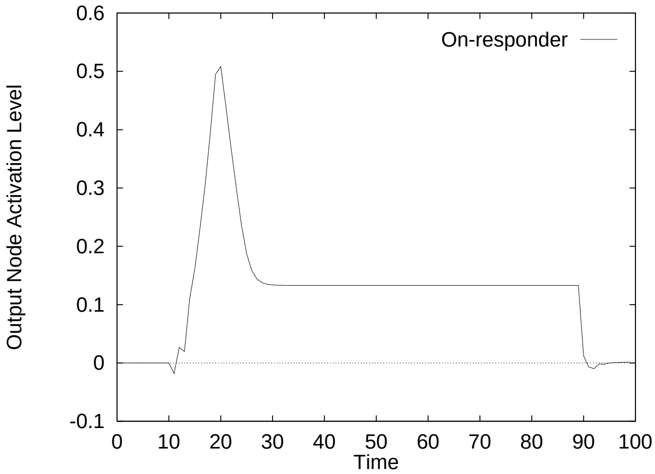


Fig. 4. Output from a network showing behavior similar to a biological on-responder.

This behavior is significant because it simulates what are known as on-responders in biological neurons; neurons that pulse only once, or a few times, when an activation signal is received. Although we were not attempting to evolve on-responders in this work it is promising to see that recurrent neural networks can simulate their behavior. In future studies we will attempt to explicitly evolve on-responders.

Some biological neurons are also off-responders, producing a pulse, or pulses, when an activation signal is turned off. We did not observe that behavior in any of the evolved networks. However, unlike the on-responders an off-responder would have a lower than average fitness because it would become active after the input was turned off. Thus, our fitness function is actively selecting against such behavior and it is not surprising that it was not observed.

We also observed some networks that produced output signals with a lower period than the target signal. Figure 5 shows the output from two such networks with periods approximately one half and one third of the target frequency. Again this is not a behavior we were actively trying to evolve. However, it does show the flexibility of the evolving networks and suggests that had we chosen a target signal with a different period the networks still would have evolved successfully.

## V. Future Work

There are several avenues of work to pursue based on these, admittedly preliminary, results.

We would like to explore, and hopefully improve, the evolutionary algorithm. For these experiments we chose

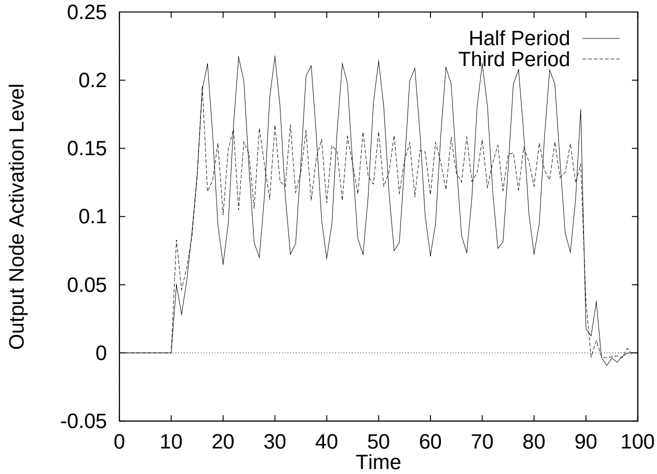


Fig. 5. Output from networks producing signals with a shorter period.

a very simple form of crossover that only exchanges the weights of connections leading into a chosen node. Alternative forms of crossover could have a significant effect on the evolutionary process.

The constant output solution is clearly a very significant local optimum. It seems likely that a fitness function that more strongly favors non-uniform output could keep evolution from getting stuck on this sub-optimum solution.

We are also interested in allowing the topology of the networks to evolve. A combination of evolvable topologies, improved crossover and fitness functions, and parsimony pressure on the network sizes may make it possible to evolve minimal networks that consistently produce the desired output.

In addition to exploring the GA, we would like to evolve more complex behaviors from the neural networks. We plan to purposefully evolve on-responders, off-responders, and pulsed outputs with different periods. The results shown here indicate that these behaviors should not be too difficult to produce. Eventually, we plan to attempt to evolve a single network that can produce a variety of behaviors depending on the input signals. Such a network would closely resemble biological neurons that produce a variety of behaviors when activated depending on which ion channels are active.

Finally, we would like to look more closely at how the evolved networks produce their output. Are the evolved weights similar to the type of solutions that a deterministic training algorithm would find? Or does the evolutionary approach tend to find very different types of solutions?

## VI. Conclusions

In this paper we demonstrated that evolutionary techniques can be used to evolve strongly recurrent networks to produce long period, pulsed output signals based on constant valued inputs. These results are significant for several reasons. Recurrent networks are notoriously difficult to train using deterministic techniques. The relatively long period outputs produced in this paper would be particularly challenging to train. Thus, the success of evolutionary techniques is very significant.

It is also worth noting that the behaviors we have evolved and observed can be quite useful in practical applications. Our evolved networks are functionally very similar to the central pattern generators the invertebrates use for a wide variety of tasks, such as walking, swimming and heartbeat regulation.

Our 'accidental' results show that these recurrent networks can produce a fairly wide range of outputs, including on-responders, and signals with a variety of periods. This demonstrates the power and flexibility of recurrent networks.

## References


[1] Eric R. Kandel and James H. Schwartz, *Principles of Neuroscience, 2nd Edition*, Elsevier, New York, NY, 1985.

[2] John E. Dowling, *Neurons and Networks: An Introduction to Neuroscience*, The Belknap Press of Harvard University Press, Cambridge, MA, 1992.

[3] Gordon M. Shepherd, *Neurobiology*, Oxford University Press, New York, NY, 1994.

[4] David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[5] John R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge, MA: The MIT Press, 1994.

[6] W. Banzhaf, P. Nordin, R. Keller, and F.Francone, *Genetic Programming – An Introduction*, San Francisco, CA: Morgan Kaufmann, 1998.

[7] Jamo T. Alander, "Indexed bibliography of genetic algotithms and neural networks," Report 94-1-NN, Univeristy of Naasa, Department of Information Technology and Production Economics, 2000.

[8] Gregory M. Saunders, Peter J. Angeline, and Jordan B. Pollack, "Structural and behavioral evolution of recurrent networks," in *Advances in Neural Information Processing Systems*, Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, Eds. 1994, vol. 6, pp. 88–95, Morgan Kaufmann Publishers, Inc.

[9] Peter J. Angeline, Gregory M. Saunders, and Jordan P. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, January 1994.

[10] Alessandro Fadda and Marc Schoenauer, "Evolutionary chromatographic law identification by recurrent neural nets," in *Evolutionary Programming*, 1995, pp. 219–235.

[11] Byoung-Tak Zhang, Peter Ohm, and Heinz Mühlenbein, "Evolutionary induction of sparse neural trees," *Evolutionary Computation*, vol. 5, no. 2, pp. 213–236, 1997.

[12] Kenneth O. Stanley and Risto Miikkulainen, "Continual coevolution through complexification," in *GECCO 2002:Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull adn M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., 2002, pp. 113–120.

[13] Kenneth O. Stanley and Risto Miikkulainen, "Efficient evolution of neural network topologies," in *Proceeedings of the 2002 Congresson Evolutionary Computation*, IEEE, Ed., 2002.

[14] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," .

[15] Jung-Hwan Kin and Byung-Ro Moon, "Neuron reording for better neuro-genetic hybrids," in *GECCO 2002:Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull adn M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., 2002, pp. 407–414.