REVERSIBLE EVOLVABLE NETWORKS

A Reversible Evolvable Boolean Network Architecture and Methodology to Overcome the Heat Generation Problem in Molecular Scale Brain Building

Hugo de GARIS, Jonathan DINERSTEIN, Ravichandra SRIRAM

Brain Builder Group, Computer Science Dept,
Utah State University, Logan, UT, USA.
tel +1 435 512 1826, degaris@cs.usu.edu, www.cs.usu.edu/~degaris

Abstract

Today's irreversible computing style, in which bits of information are routinely wiped out (e.g. a NAND gate has 2 input bits, and only 1 output bit), cannot continue. If Moore's Law remains valid until 2020, as many commentators think, then the heat generated in molecular scale circuits that Moore's Law will provide, would be so intense that they will explode [Hall 1992]. To avoid such heat generation problems, it has been known since the early 1970s [Bennet 1973] that the secret to 'heatless computation' is to compute reversibly, i.e. not to destroy bits, by sending in the input bit-string through a computer built from reversible logic gates (e.g. Fredkin gates [Fredkin et al 1982], to record the output answer and then send the output bit-string backwards through the computer to obtain the original input bit-string. This reversible style of computing takes twice as long, but does not destroy bits, hence does not generate heat. (Landauer's principle states that "the heat generated from irreversible computing is derived from the destruction of bits of information" [Landauer 1961]). The first author intends to build artificial brains over the remaining 20 years of his active research career, by evolving (neural) network modules directly in electronics (at electronic speeds) in their 100,000s and assembling them into artificial brains. In the next 10-20 years, electronic circuitry will reach molecular scales; hence a conceptual problem needs to be faced. How to make evolvable (neural) networks that are reversible? This paper proposes a reversible evolvable Boolean network architecture and methodology which, it is hoped, will stimulate the evolvable hardware and evolvable neural network research communities to devote more effort towards solving this problem, which can only accentuate as Moore's Law continues to bite.

1. INTRODUCTION

As the above abstract states, it is only a question of time (probably less than 10 years from now) before the issue of heat dissipation in increasingly smaller electronic circuit components will become a major problem if computers continue to function in the same irreversible way, which routinely wipes out bits of information during the computation process. The only known solution to this problem is to use reversible computing, which stores all bits and does not wipe out information. Hence the first author, if he wishes to continue building artificial brains by evolving neural networks directly in electronics at electronic speeds, will need to find a solution to evolving neural networks that are REVERSIBLE. This paper introduces an architecture and methodology for the creation of Reversible Evolvable (Boolean) Networks (RENs), which hopefully makes a contribution towards the solution of this problem, and it is further hoped that it will motivate other researchers in the evolvable hardware, evolvable neural networks, brain building and related fields to work in this new research area.

The contents of the remainder of this paper are as follows. Section 2 presents the basic ideas behind the architecture and methodology. Section 3 goes into more detail. Section 4 presents results of some early simulation experiments that use the architecture. Section 5 discusses and considers ideas for future research.

2. A REVERSIBLE EVOLVABLE (BOOLEAN) NETWORK (REN) ARCHITECTURE

2.1 BASIC IDEAS

The basic idea employed in this reversible evolvable network architecture is to use a Boolean network, where the nodes are Boolean function truth tables, and the links are connections of single bit streams taken from one of the output
columns in a node's truth table to one of the input columns of (usually) another node's truth table. Some constraints need to be applied to these Boolean truth tables to ensure reversibility (in so far as total reversibility is achievable). It should be obvious that the number of bits input to, and output from, a node must be equal. If this were not the case, then inevitably there will be a many-to-one, or one-to-many mapping between the N input bits and the M (not equal to N) output bits (i.e. some of the rows in the truth table will be the same in either the input or the output side). Also, the $2^N$ rows in the output side must all be different. If these conditions are satisfied, then we have reversibility, meaning that for any one of the $2^N$ different inputs, one can consult the truth table to find the corresponding output string of N bits. Similarly, if the signaling were reversed (for example, by winding the movie backwards) then the incoming "output bits" would generate the corresponding outgoing "input bits". Such a phenomenon is needed to occur if the circuit is said to be reversible. Since no bits are lost (i.e. not wiped out) then no (or very little) heat should be generated.

A Boolean network of this type can have any number M of nodes. There will thus be a total of $N^M$ input lines and $N^M$ output lines between all the nodes, where N is both the number of input bits and the number of output bits in a node’s truth table (for all nodes in the network). An output line from one node becomes another node’s input line (although an output line can loop back to the same node). These one bit input and output lines are connected randomly (under genetic control) between the nodes. Every node thus emits N output lines, and receives N input lines (from (mostly) other nodes’ output lines). The N incoming bits for a node are fed into the input column of its truth table and the corresponding N output bits at the same row position of the truth table are fed into the node’s N output lines (which are then distributed to (usually) other nodes’ input lines).

2.2 METHODOLOGY

To start signaling, all nodes are given an initial input bit-string signal of $N$ 1s. Arbitrarily choose a single output node T. Arbitrarily choose one of its output bits B. Assume the signaling is clocked (synchronous). The (time varying) bit-string output of this node's bit B is used as the output of the network, from which the fitness (in Genetic Algorithm terms) is measured.

In earlier work [Korkin et al 1998] this bit string can be converted into a time dependent (digitized) analog signal by convoluting the bit string with a digitized convolution window, so that this converted analog signal can then be compared with an analog target signal for evolution. The fitness of the network can then be defined by how closely the (converted) output signal matches (i.e. follows) the target output signal curve over time.

For a Genetic Algorithm to be applied to a population of networks, a single network's structure needs to be recorded in a data structure, usually called its "chromosome". This chromosome will contain two sets of information -

a) the truth tables of the M nodes in the network (if there are $N$ in/output bits in (all) the tables, then these truth tables will have $2^N$ input and output rows).

b) a connection matrix $CM(Npq, Nrs)$ where $Npq$ is the $q$th output line of node $p$, is connected (or not) to the $st$ input line of the $r$th node. ($p$ can be equal to $r$). This matrix can be represented as an $(M*N)*(M*N)$ integer array. A 1 in this matrix $CM(i,j)$ indicates a connection between the appropriate output line $i$ and input line $j$.

2.3 GENETIC OPERATORS

Given the above chromosome data structure, the following genetic operators suggest themselves. (Note, some of these operators were changed in subsequent experiments, as shown later in this paper)

i) Truth Table Mutations

a) Choose randomly a node in the network.

b) Choose randomly two different truth table output rows for that chosen node.

c) Swap the two output rows. (One cannot simply flip a random bit in the output row because this would then generate a row equal to some other row in the output side of the truth table, so swapping is necessitated).

ii) Connection Mutations

Swapping Input Lines on the Same Node

a) Choose randomly a node, in the network.

b) Choose randomly two of its input lines (using the connection matrix).

c) Swap these two input lines (e.g. imagine neuron 5 has 2 input lines, which come from output column 2 of neuron 7 and column 1 of neuron 3, then using the above connection matrix notation $CM(Nij,Npq)$, the two input links to neuron 5, $[CM(7-2,5-1), CM(3-1,5-2)]$ becomes $[CM(7-2,5-2), CM(3-1,5-1)]$

d) Update the connection matrix.

Swapping Output Lines on the Same Neuron

a) Choose randomly a node in the network.

b) Choose randomly two of its output lines (using the connection matrix).

c) Swap these two output lines (e.g. imagine node 9 has 2 output lines, the first which comes from output column 1, to column 1 of node 4, and the second which comes from output column 2, to column 1 of node 2, then using the
above connection matrix notation the two output links from node 9, \([\text{CM}(9-1,2-1), \text{CM}(9-2,2-1)]\) become \([\text{CM}(9-2,2-1), \text{CM}(9-1,2-1)]\).

d) Update the connection matrix.

**Swapping Input Lines on Different Neurons**

a) Choose randomly two different nodes, in the network.
b) Choose randomly one input line for each of the two nodes.
c) Swap these two input lines.
d) Update the connection matrix.

Actually, the above swapping mutations can be absorbed into a single approach, by simply using the connection matrix to swap rows or columns. Since there can only be a single 1 in any row or column, then the above operations can be absorbed into the following scheme.

i) Randomly choose any two columns in the connection matrix. Observe the two row positions in which the 1s occur. Swap the two 1s in those two row positions.

ii) Randomly choose any two rows in the connection matrix. Observe the two column positions in which the 1s occur. Swap the two 1s in those two column positions.

2.4 FITNESS MEASUREMENT

Various target output curves were chosen in a series of experiments, starting with trivial targets to test minimal performance. The first target curve was a simple flat line of a constant value. The next was a sloping straight-line curve. The third target curve was a time varying “Fourier like” curve with formula

\[
S(t) = 300 + 100\sin(\frac{2\pi}{100}t) + 50\sin(\frac{2\pi}{80}t) + 30\sin(\frac{2\pi}{70}t) - 20\sin(\frac{2\pi}{60}t)
\]

Assume that the equation of the target curve to be followed by the evolved output signal is \(S(t) = 300 + 100\sin(\frac{2\pi}{100}t) + 50\sin(\frac{2\pi}{80}t) + 30\sin(\frac{2\pi}{70}t) - 20\sin(\frac{2\pi}{60}t)\). The integer values of the digitized convolution function window (of width 20 clock ticks) is \(\{8, 16, 26, 35, 44, 52, 59, 64, 65, 64, 61, 57, 52, 45, 37, 29, 21, 13, 7, 4\}\).

The fitness is defined to be the inverse of the sum of the squares of the errors over 100 clock ticks between the target values at time \(t\) and the actual output values at time \(t\). Fig 2 shows the target curve and the best result obtained with (give GA parameters here).

3. EXPERIMENTAL RESULTS

**Expt. 1**

The first experiment undertaken was to see whether the model would evolve at all. It used a constant target curve = 8 (as shown in Fig. 1). Since the time dependent output from the Boolean network was a bit string, a digitized convolution function was applied to the output converting it to a dynamic analog signal. The digits used in the convolution window were \(\{2, 2, 2, 2\}\). At each clock tick, these 4 digits were lined up against 4 contiguous output bits. If an output bit was 1, the corresponding convolution bit was added to the running total. For example - 

\[
10101001 \ldots
\]

\[
2222 \quad \text{would give}
\]

\[
1*2 + 0*2 + 1*2 + 0*2 = 4
\]

At the next clock tick if the output bit is 0, the window moves one bit position to the right. The next convoluted output would be \(0*2 + 1*2 + 0*2 + 1*2 = 4\). The convolution function was devised rather arbitrarily, just to see if any evolution at all would emerge. It did, and well, as shown in Fig. 1. The Boolean net used 18 nodes with 6 bits in the input and output tables. We proceeded to more demanding evolutionary tasks.

**Expt. 2**

In this experiment, the target curve was a function of time, but still fairly simple to test whether this reversible Boolean network could evolve a simple time dependent output. The target curve is shown in Fig. 2 and takes the form of a simple straight line slope. The target curve equation was 

\[
S(t) = 0.5t
\]

for 32 clocks. The convolution window used was \(\{0.5, 2, 1, 2, 1, 2, 1, 2, 0.5\}\). The fitness function taken was the reciprocal of the sum of the squares of the differences of the target and the convoluted output values. Fig. 2 shows that it evolved quite well. Again the number of nodes used was 18, and 6 bits for the input and output tables.

**Expt. 3**

In this experiment a much more complex dynamic “Fourier like” target curve was employed as shown in Fig. 3, which had the formula - 

\[
S(t) = 300 + 100\sin(\frac{2\pi}{100}t) + 50\sin(\frac{2\pi}{80}t) + 30\sin(\frac{2\pi}{70}t) - 20\sin(\frac{2\pi}{60}t)
\]

The convolution window chosen for this experiment is \(\{8, 16, 26, 35, 44, 52, 59, 64, 65, 64, 61, 57, 52, 45, 37, 29, 21, 13, 7, 4\}\). With the same number of nodes and bits widths, Fig. 3 shows that the output curve evolved moderately well. (One does not expect great accuracy from the convolution window approach). The output curve ran for 150 ticks.
Expt. 4

In order to gain some idea of the limits to the evolvability of the network (with 18 nodes and 6 bitwidths), the target curve was extended to 250 ticks. Fig. 4 shows that the network is starting to show an averaging effect, which we interpret as displaying the limit of the network’s “evolvable capacity”.

Expt. 5(to know pattern recognition capability)

The pattern detection ability of the Boolean network is also tested. We have taken encoding of 10 decimal digits as follow 0-1110111, 1-0010010, 2-1011101, 3-1011011, 4-0101010, 5-1101011, 6-1101111, 7-1010011, 8-1111111 and 9-1111011 and were able to evolve the network to detect one positive (i.e., to give high output for +ve case, when for example 8 (i.e., 1111111) is given and to give low input when all other 9 negatives are given to the same net. Multitestning is done by giving positive case input (i.e. for 8 detector 8 is positive case input and all others are negatives) to the net only at the zeroth clock tick and then at every succeeding 8th clock tick each negative input is given. The number of nodes used was 18, 7 bits for input output tables (each digit has exactly 7 digits in its encoded form) The net evolved quite well and proved that this type of Boolean neural net model can be used for both curve following and also pattern detection. The result is shown in Fig 5.

Expt. 6

This experiment used a rather different architecture (RENb) for the Boolean network, and was undertaken by the second author, whereas the above experiments were performed by the 3rd author. In this experiment, every node has an even number of inputs >= 3. The first input is a control bit (swap/don’t swap), and all other inputs the data inputs. When a swap command is received, the first half and second half of the input data signals are swapped. If a swap command isn’t received, the data inputs pass straight through to the output links (somewhat like a reversible “Fredkin gate” [Fredkin et al 1982]). 40 nodes worked well, with 9 inputs/outputs per gate, as can be seen if Fig. 6. Each output goes to one and only one input (so that the net will be reversible). Therefore, there were 40*9 total links. Any output can go to any input in the neural net, but each input can only receive one output (so that the net will be reversible). The truth table had 9 bit inputs and outputs, with 1 link leaving each output and 1 link going to each input. Any output can be linked to any input. The only genetic operator is to swap two input links (i.e. swap where two links are connected to supply input to two gates).

The output was obtained from the net by sampling one output bit of one node for each clock tick. A convolution function 11 digits wide was used. Each output 1 bit was multiplied by 0.1, and the result summed together. Input was supplied to the network by initializing all links to a value, 0 or 1. The initialization values were a series of bits 010101... In other words, every other link was initialized with 0, and the other links with 1. Once the neural net started running, no more external input is supplied. Fig. 6 shows the evolved curve for 150 ticks, and Fig. 6 for 400 ticks. The curve in Fig. 7 shows that the evolvable capacity of the network has probably been reached.

4. DISCUSSION AND FUTURE RESEARCH

The figures below show clearly that the model is evolvable, hence reversible Boolean networks can be evolved, which is encouraging. However, this preliminary paper leaves many questions and issues unanswered and perhaps obscure. Despite this, the authors hope that this paper will stimulate other researchers to undertake further work into reversible evolvable networks. The topic is of critical future importance for the new field of brain building in particular and for computer science in general. The whole of computer science will sooner or later have to pass through the bottlenecks of reversible computing. Moore’s Law is doomming the traditional style of computing, which is irreversible, as explained in the abstract. The first author wants to be building artificial brains for the next 20 years, by evolving large numbers of neural networks directly in hardware at hardware speeds. These evolved networks are then assembled into artificial brain architectures [de Garis et al 2002]. Within 20 years, electronic circuits will be of molecular scale and hence will have to be reversible. So to the first author, the question “Can evolvable neural networks be made reversible?” is important. The authors feel that this question has now been answered by this paper, and its experimental results. Of course more evolvable models can always be found. Our group’s irreversible TiPo model (Fig. 8) which evolves virtually perfect curve following abilities is a standard to be equaled in the future by reversible models.

But some questions need further work, for example -

a) How many of the steps mentioned above are truly reversible?

b) How to insert external inputs and yet keep the circuit as reversible as possible? Future evolvable networks will need to have external inputs if they are to be useful, but how to process such input reversibly?

c) How is the above architecture to be implemented in evolvable hardware?

d) Are there better ways, better architectures to build reversible evolvable neural networks, or even reversible evolvable hardware?
REFERENCES


Fig. 1  REN’s Flat Line Target Evolution
Fig. 2  REN’s Sloped Line Target Evolution

Fig. 3  REN’s Curve Following, 150 Ticks

Fig. 4  REN’s Curve Following, 250 Ticks
Fig. 5. Digit Recognition

digit recognition. one +ve and 9 -ves, +ve aimed at target 6 and -ve aimed at target 0, only the evolved curve shown (exact evolved output points shown with small triangles)

Fig. 6. RENb’s Curve Following, 150 Ticks
Fig. 7  RENb’s Curve Following, 400 Ticks

Fig. 8  TiPo Model’s Virtually Perfect Curve Following
(The Target and Evolved Curves Overlap So Closely, the 2 Curves are Offset)