Evolving an Optimal De/Convolution Function for the Neural Net Modules of ATR’s Artificial Brain Project

Hugo de Garis(1), Norberto Eiji Nawa(1), Michael Hough(2), Michael Korkin(3)
(1) Brain Builder Group, Evolutionary Systems Department, ATR
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto-fu, 619-0288, Japan
tel: (+81) (0)774 95 1079, fax: (+81) (0)774 95 1008
degaris@hip.atr.co.jp, http://www.hip.atr.co.jp/~degaris
(2) Computer Science Department
Stanford University, Stanford, 94305, CA, USA
mhough@cs.stanford.edu, http://www.stanford.edu/~mhough
(3) Genobyte Inc.,
1503 Spruce Street, Suite 3, Boulder, Colorado 80302, USA
tel: (+1) 303 545 6790, fax: (+1) 303 415 1109
korkin@genobyte.com, http://www.genobyte.com

Abstract

This paper reports on efforts to evolve an optimum de/convolution function to be used to convert analog to binary signals (spike trains) and vice versa for the binary input/output signals of the neural net circuit modules evolved at electronic speeds by the so-called "CAM-Brain Machine (CBM)" of ATR’s Artificial Brain Project [1, 2, 3]. The CBM is an FPGA based piece of hardware which will be used to evolve tens of thousands of cellular automata based neural network circuits or modules at electronic speeds in about a second each, which are then downloaded into humanly architected artificial brains in a large RAM space [2, 3]. Since state-of-the-art programmable FPGAs constrained us to use 1 bit binary signaling in our neural model (the "CoDi-1Bit" model [4]), an efficient de/convolution technique is needed to convert digital signals to analog and vice versa, so that "evolutionary engineers" (EEs) who evolve the many modules, can think in terms of analog signals when they need to, rather than in terms of the abstract incomprehensible spike trains (where the information in the signal is contained in the spacing between spikes). An earlier convolution function, digitized from a text book figure [7] (shown in Fig. 1), gave only moderate accuracy between actual and desired output results. By applying a genetic algorithm to the evolution of the de/convolution function we were able to make it nearly twice as accurate. Accuracy is important so as to reduce cumulative errors when the output of one neural net module becomes the input of another, in long sequential chains of modules in artificial brain architectures consisting of 10,000s of modules. The CBM can handle up to 32000 modules (of maximum about 1000 artificial neurons each) and will be delivered to ATR by the spring of 1999.

1 Introduction

This paper reports on our efforts to evolve an optimum de/convolution function which is used to convert analog to binary signals (spike trains) and vice versa for the neural net modules evolved by ATR’s CAM-Brain Machine (CBM) [5]. The CBM is an FPGA based piece of specialist hardware which grows and evolves cellular automata (CA) based neural network circuits (modules) at electronic speeds. It updates the CA at a rate of 150 Billion a second and can evolve a neural net module (i.e. a complete run of a genetic algorithm (GA), with tens of thousands of circuit growths and fitness evaluations) in about a second. However, the constraints imposed by the FPGA chips used in the CBM are such that only 1 bit neural signaling is possible. Hence an interpretation problem arose. What do the 1 bit signals entering and exiting the modules mean? We tried various interpretations (representations) [6] and eventually chose a convolution approach, where the spike train (a binary string) is convoluted with a "convolution function" that we digitized from a figure in a text book [7]. (See Fig. 1, where the horizontal axis shows the number of clock cycles, and the vertical axis shows the values of the convolution function). The re-
sulting convoluted analog waveforms were only moderately accurate (i.e. the actual waveform convoluted from a spike train evolved with a simulation of the CBM differed from the target waveform by roughly 10%, which was thought not to be particularly good. Hence we were motivated to see if a more accurate de/convolution function could be evolved.

The reason it is important to get greater accuracy, is because if each module has a margin of error of about 10%, then when the time comes to assemble tens of thousands of modules into artificial brains, in hierarchical modular structures, the errors will accumulate as they pass sequentially from one module to another, to the point of rendering the artificial brain inoperative. We therefore set out to see by how much we could improve the accuracy between the actual outputs and the target outputs. The success of the CAM-Brain Project (ATR's Artificial Brain Project) may depend upon attaining a greater accuracy.

Note, that there are two sources of error when evolving CA based neural nets using the CBM. One is due to the limitations of the de/convolution functions when users supply analog values as inputs and analog target outputs to the CBM. (The CBM itself uses purely digital inputs and outputs, so if the user wants to think in terms of analog waveforms, which is much easier to do than abstract spike-trains, then deconvolution and convolution transformations will be necessary). The other source of error is due to the quality of the evolved neural net module itself. In this paper we do not address ourselves to this second source of error. There have already been several papers published on the quality of evolved modules [2, 3].

In an attempt to reduce the first kind of error, we took the following approach. An initial arbitrary sinusoidal analog wave form (curve) of the form shown in Fig. 2 (the smooth curve) was input into a deconvolution function (as explained in section 3 below) which resulted in a spike train (bit string) output (Fig. 3). The deconvolution (spiking) algorithm was based on a convolution function which did the reverse, namely converted the spike train back to an analog output (as explained in section 2). In theory, the result of the final convolution output should be the same as the original analog input waveform. This two step approach is shown in Fig. 4. A genetic algorithm was used to evolve the digitized values of the convolution function (an analog waveform) which gave the best matching to the target waveform (i.e. the original input analog waveform).

The remainder of this paper consists of the following sections. Section 2 explains the convolution approach we use (called "SIIC" = "Spike Interval Information Coding"), which converts a spiketrain (a bitstring) into an analog waveform. Section 3 explains the reverse process, i.e. deconvolution (or spiking) which converts an analog waveform into a spiketrain. Our deconvolution algorithm (called "HSA" = "Hough Spiker Algorithm" [3]) uses the digitized values of the SIIC convolution function. Section 4 explains how we used these two operations to optimize the digitized values plus the "window width" of the convolution function, which when incorporated into the deconvolution function, gave the closest fit between the original analog input waveform and the final output analog waveform. Section 4 also reports on the experimental results.

2 Convolution with the "SIIC" (Spike Interval Information Coding) Representation

The constraints imposed by state-of-the-art programmable (evolvable) FPGAs in 1998 were such that the CA based model (the CoDi model) had to be very simple in order to be implementable within those constraints. Consequently, the neural signaling states in the model were made to contain only 1 bit of information (as happens in nature’s "binary" spike trains). The problem then arose as to interpretation. How were we to assign meaning to the binary pulse streams (i.e. the clocked sequences of 0s and 1s which are a neural net module’s inputs and outputs? We tried various ideas such as a frequency based interpretation, i.e. count the number of pulses (i.e. 1s) in a given time window (of N clock cycles). But this was thought to be too slow. In an artificial brain with tens of thousands of modules which may be vertically nested to a depth of 20 or more (i.e. the outputs of a module in layer "n" get fed into a module in layer "n+1"), where "n" may be as large as 20 or 30) then the cumulative delays may end up in a total response time of a robot kitten (that the CAM-Brain is to control) being too slow (e.g. if you wave your finger in front of its eye, it might...
react many seconds later). We wanted a representation that would deliver an integer or real valued number at each clock tick, i.e. the ultimate in speed. The first such representation we looked at we called "unary" i.e. if \( N \) neurons on an output surface are firing at a given clock tick, then the firing pattern represented the integer \( N \), independently of where the outputs were coming from. We found this representation to be too stochastic, too jerky. Ultimately we chose a representation which convolves the binary pulse string with the convolution function shown in Fig. 1. We call this representation "SIIC" (Spike Interval Information Coding) which was inspired by [7]. This representation delivers a real valued output at each clock tick, thus converting a binary pulse string into an analog time dependent signal. Our team has already published several papers on the results of this convolution representation work [6]. The CBM will implement this representation in the FPGAs when measuring fitness values at electronic speeds.

2.1 Details

The convolution algorithm we use takes the output spike train (a bit string of 0s and 1s) from a CBM module, and runs the pulses (i.e. the 1s) by the convolution function shown in the simplified example below. The output at any given time \( t \) is defined as the sum of those samples of the convolution filter that have a 1 in the corresponding spiketrain positions. The example below should clarify what is meant by this.

2.1.1 Simplified Example

Convolve the spiketrain 1101001 (where the left most bit is the earliest, the right most bit, the latest) using the convolution filter values \( \{ 1 4 9 5 \cdot 2 \} \). The spiketrain in this diagram moves from left to right across the convolution filter. Alternatively, one can view the convolution filter (window) moving across the spiketrain. The number to the right of the colon (:) shows the value of the convolution sum at each time \( t \).

\[
\begin{align*}
\text{time-shifted spike train} &: 1 0 0 1 0 1 1 \quad \text{---> (moves left to right)} \\
\text{convolution filter} &: 1 4 9 5 \cdot 2 \\
1 0 0 1 0 1 1 &: 0 0 0 0 0 : 0 \quad t = -1 \\
1 0 0 1 0 1 1 &: 1 0 0 0 0 : 1 \quad t = 0 \\
1 0 0 1 0 1 1 &: 1 4 0 0 0 : 5 \quad t = 1 \\
1 0 0 1 0 1 1 &: 0 4 9 0 0 : 13 \quad t = 2 \\
1 0 0 1 0 1 1 &: 1 0 9 5 0 : 15 \quad t = 3 \\
1 0 0 1 0 1 1 &: 0 4 0 5 \cdot 2 : 7 \quad t = 4 \\
1 0 0 1 0 1 1 &: 0 0 9 0 \cdot 2 : 7 \quad t = 5 \\
1 0 0 1 0 1 1 &: 1 0 0 5 0 : 6 \quad t = 6 \\
1 0 0 1 0 1 1 &: 0 4 0 0 \cdot 2 : 2 \quad t = 7
\end{align*}
\]
Hence, the time-dependent output of the convolution filter takes the values (0, 1, 5, 13, 15, 7, 6, 2, 9, 5, -2). This is a time varying analog signal, which is the desired result.

3 Deconvolution with the "HSA" (Hough Spiker Algorithm)

Section 2 above explained the use of the SIIC (Spike Interval Information Coding) representation which provides an efficient transformation of a spike train (i.e. string of bits) into a (clocked) time varying "analog" signal. We need this interpretation in order to interpret the spike train output from the CoDi modules to evaluate their fitness values (e.g. by comparing the actual converted analog output waveforms with user specified target waveforms). However, we also need the inverse process, i.e. an algorithm which takes as input, a clocked (digitised, i.e. integer numbered) time varying "analog" signal, and outputs a spike train. This conversion is needed as an interface between the motors/sensors of the robot bodies (e.g. a kitten robot) that the artificial brain controls, and the brain’s CoDi modules. However, it is also very useful to users, i.e. EEs (evolutionary engineers) to be able to think entirely in terms of analog signals (at both the inputs and outputs) rather than in abstract, visually unintelligible spiketrains. This will make their task of evolving many CoDi modules much easier. We therefore present next an algorithm which is the opposite of the SIIC, namely one which takes as input, a time varying analog signal, and outputs a spike train, which if later is convoluted with the SIIC convolution filter, should result in the original analog signal.

A brief description of the algorithm used to generate a spiketrain from a time varying analog signal is now presented. It is called the "Hough Spiker Algorithm" (HSA) [3] and can be viewed as the inverse of the convolution algorithm described above in section 2.

To give an intuitive feel for this deconvolution algorithm, consider a spiketrain consisting of a single pulse (i.e. all 0s with one 1). When this pulse passes through the convolution function window, it adds each value of the convolution function to the output in turn.

A single pulse: (100000... → t = +∞) will be convoluted with the convolution function expressed as a function of time. At t = 0 its value will be the first value of the convolution filter, at t = 1 its value will be the second value of the convolution filter, etc. Just as a particular spiketrain is a series of spikes with time delays between them, so too will the convolved spiketrain be the sum of the convolution filters, with (possibly) time delays between them. At each clock tick when there is a spike, add the convolution filter to the output. If there is no spike, just shift the time offset and repeat.

The same example.

\[
\text{spike train: } 1 1 0 1 0 0 1 \\
\text{convolution filter: } 1 4 9 5 -2
\]

\[
t \rightarrow 0 1 2 3 4 5 6 7 8 9 10 \\
\text{out: } 1 1 4 9 5 -2 \\
0 0 0 0 0 0 \\
1 1 4 9 5 -2 \\
0 0 0 0 0 0 \\
0 0 0 0 0 0 \\
1 1 4 9 5 -2 \\
\]

In the HSA deconvolution algorithm, we take advantage of this summation, and in effect do the reverse, i.e. a kind of progressive subtraction of the convolution function. If at a given clock tick, the values of the convolution function are less than the analog values at the corresponding positions, then subtract the convolution function values from the analog values. The justification for this is that for the analog values to be greater than the convolution values, implies that to generate the analog signal values at that clock tick, the CoDi module must have fired at that moment, and this firing contributed the set of convolution values to the analog output. Once one has determined that at that clock tick, there should be a spike, one subtracts the convolution function’s values, so that a similar process can be undertaken at the next clock tick. For example, to deconvolve the convolved output (using the same values of the convolution function as in the simple example of the previous section.

\[
1 1 0 1 0 0 1 \\
\text{comp: } 1 4 9 5 -2 \\
\text{conv.vals.LT.analog signal values, so spike 1} \\
0 1 4 10 9 7 6 2 9 5 -2 \\
\text{subtract} \\
1 1 4 9 5 -2 \\
\text{less, so spike 11} \\
0 0 0 1 4 9 6 2 9 5 -2 \\
\text{subtract} \\
1 1 4 9 5 -2 \\
\text{notless nospike 110} \\
0 0 0 0 1 4 9 6 2 9 5 -2 \\
\text{(time++)} \\
\text{compare} \\
1 1 4 9 5 -2 \\
\text{less, spike 1101} \\
0 0 0 0 1 4 9 5 -2 \\
\text{subtract} \\
1 1 4 9 5 -2 \\
\text{notless 11010} \\
0 0 0 0 0 1 4 9 5 -2 \\
\text{(time++)} \\
\text{compare} \\
1 1 4 9 5 -2 \\
\text{notless 110100}
\]
It is assumed that spiking will irreversibly raise the value of the convolved output. If the convolution filter value at a given clock tick is less than that of the target waveform, spiking will bring the two values closer together. If the waveform value is still too low after a spike has occurred, a near future spike will bring the two closer together.

Figs 2 and 3 show an example of the HSA in action. The original input analog signal is the smooth curve in Fig. 2. The spiketrain output generated by the HSA deconvolutor (the HSA "spiker") is shown in Fig. 3. The spiketrain resulting from each analog input is sent into the SIIC convolutor (whose convolution function is shown in Fig. 1). The resulting analog output should be very close to the original analog input curve. The smooth curve in Fig. 2 is the resulting output curve. The HSA works well when the values of the waveforms are large (e.g. greater than 200, and less than 1000) and do not change too quickly relative to the time width of the convolution filter window. One can simply add a constant to incoming analog signals before spiking them, and ensure that the analog signals do not change too rapidly (i.e. one can filter out the high frequency components of signals with a low frequency band pass filter).

4 Evolving the Optimal De/Convolution Function, Experimental Results

How to evolve a more accurate de/convolution function? The initial approach we tried was to evolve the integer values of the digitized convolution function of the type shown in Fig. 1 for fixed window widths (e.g. 48 in Fig. 1).

We initialised a population of integer strings (actually, each integer was a constant of 70) which were the integer values of the digitized convolution function, and used the average percentage error of the actual analog output and the target analog output as the genetic algorithm fitness function for each member of the GA population. These fitness scores served as the basis for comparison to see if we could improve upon these scores by evolving the convolution function for progressively smaller sizes of its window width. To do so, we arbitrarily constructed two sinusoidal type wave forms namely:

\[
F_1(t) = 300 + 100 \sin(\frac{2\pi t}{100}) + 50 \sin(\frac{2\pi t}{80}) + 30 \sin(\frac{2\pi t}{70}) - 20 \sin(\frac{2\pi t}{60})
\]

\[
F_2(t) = 300 - 100 \cos(\frac{2\pi t}{80}) + 50 \sin(\frac{2\pi t}{70}) + 30 \sin(\frac{2\pi t}{65}) + 20 \sin(\frac{2\pi t}{40})
\]

and evolved the values of the convolution function so that the average error (between actual and target wave form values) over these two wave forms was least. Since the computational load per window width case was fairly light, we were able to run through 3000 generations of the GA in less than a minute on an IBM Thinkpad 133 MHz machine, so this seemed a practical idea.

The GA chromosome representation was a list of positive integers (the HSA deconvolution algorithm does not work with negative values, hence we simply added a large constant to keep everything positive), one per digitized value of the convolution function. We chose the following GA parameter values and genetic operators: - population size of 17, no crossover, the mutation operators were integer incrementors and decrementors of the integer convolution values. A random position along the convolution values was generated and with a 50/50 probability, the convolution value was incremented or decremented by 1. The integer convolution values list was used in the convolution function and the deconvolution function to perform the experiment. The fitness definition was the inverse of the sum of the absolute differences between the actual output values and the target values (the values of the original wave function). Table 4 shows the percentage errors for various window lengths averaged over the two curves. The third column shows a truncated percent error, meaning that the first dozen or so output values of the 136 clock cycles used were ignored. The output curve, generated by the convolution function takes time to climb to the target curve generating excessive errors. See the jerk curve of Fig. 2 for an example of this climbing phenomenon.

We chose the convolution function with window width of 20. Its truncated percentage error was only about 1.5%. Its convolution function took the values (8, 16, 26, 35, 44, 52, 59, 64, 65, 64, 61, 57, 52, 45, 37, 29, 21, 13, 7, 4) depicted graphically in Fig. 5.

Now armed with this evolved convolution function we then tested it against the original 48 width convolution function on a pair of new arbitrary test wave forms as shown below.

\[
F_3(t) = 300 + 100 \sin(\frac{2\pi t}{100}) - 45 \sin(\frac{2\pi t}{80}) + 35 \sin(\frac{2\pi t}{70}) - 20 \cos(\frac{2\pi t}{60})
\]

\[
F_4(t) = 300 + 100 \cos(\frac{2\pi t}{80}) + 50 \sin(\frac{2\pi t}{70}) + 30 \sin(\frac{2\pi t}{65}) + 20 \sin(\frac{2\pi t}{40})
\]
### Table 1. Percentage Errors for Various Window Widths

<table>
<thead>
<tr>
<th>Window Width</th>
<th>% error</th>
<th>% error truncated</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>5.61</td>
<td>2.25</td>
</tr>
<tr>
<td>44</td>
<td>5.32</td>
<td>2.03</td>
</tr>
<tr>
<td>40</td>
<td>5.30</td>
<td>2.12</td>
</tr>
<tr>
<td>36</td>
<td>5.12</td>
<td>2.07</td>
</tr>
<tr>
<td>32</td>
<td>5.08</td>
<td>1.66</td>
</tr>
<tr>
<td>28</td>
<td>5.13</td>
<td>1.78</td>
</tr>
<tr>
<td>24</td>
<td>5.52</td>
<td>2.36</td>
</tr>
<tr>
<td>20</td>
<td>4.95</td>
<td>1.46</td>
</tr>
<tr>
<td>16</td>
<td>5.63</td>
<td>2.82</td>
</tr>
<tr>
<td>12</td>
<td>5.66</td>
<td>3.35</td>
</tr>
<tr>
<td>8</td>
<td>9.04</td>
<td>7.37</td>
</tr>
<tr>
<td>4</td>
<td>14.44</td>
<td>13.86</td>
</tr>
</tbody>
</table>

### Table 2. Percentage Error Comparison between Textbook and Evolved Cases

<table>
<thead>
<tr>
<th>Window Width</th>
<th>% error</th>
<th>% error trunc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>48 (Text)</td>
<td>8.42</td>
<td>4.36</td>
</tr>
<tr>
<td>20 (Evolved)</td>
<td>6.25</td>
<td>2.27</td>
</tr>
</tbody>
</table>

Table 2 shows the results of the comparison between the two cases, the original 48 width convolution function taken from the text book, and the evolved 20 width convolution function. Accuracy improved with the evolved convolution function by a factor of nearly two, thus justifying the effort. From now on, we will be using the evolved convolution function.

Fig. 2 above shows the target and actual output for the waveform $F_3$, using the evolved convolution function.

### References


