ATR's Artificial Brain (CAM-Brain) Project: A Sample of What Individual CoDi-1Bit Model Evolved Neural Net Modules Can Do

Hugo de Garis
Dept. 6, ATR-HIP
Kyoto, Japan
www.hip.atr.co.jp/∼degaris

Felix Gers
IDSIA
Lugano, Switzerland
www.idsia.ch/∼felix

Michael Korkin
Genobyte, Inc.
Boulder, CO, USA
www.genobyte.com

Michael Hough
CS Dept., Stanford University
Stanford, CA, USA
www.stanford.edu/∼mhough

Abstract- This paper presents a sample of what evolved neural net circuit modules using the socalled "CoDi-1Bit" neural net model can do. This work is part of an 8 year research project at ATR which aims to build an artificial brain containing a billion neurons by the year 2001, that will be used to control the behaviors of a kitten robot "Robokoneko". It looks as though the figure is more likely to be 40 million, but the numbers are not of great concern. What is more important is the issue of evolvability of the cellular automata (CA) based neural net circuits which grow and evolve in special FPGA (Field Programmable Gate Array) hardware, at hardware speeds (e.g. updating 150 billion CA cells per second, and performing a complete run of a genetic algorithm, i.e. tens of thousands of circuit growths and fitness evaluations, to evolve the elite neural net circuit in about 1 second). The specialized hardware which performs this evolution is labeled the CAM-Brain Machine (CBM). It implements the CoDi-1Bit model, and will be delivered to ATR probably in January 1999. The CBM should make practical the assemblage of 10,000s of evolved neural net modules into humanly defined artificial brains. For the past few months, the latest hardware version of the CBM has been simulated in software to see just how evolvable and functional individual evolved modules can be. This paper reports on some of the results of these simulations.

1 Introduction

ATR's CAM-Brain Project aims to build an artificial brain containing up to a billion artificial neurons by the year 2001. The essential ingredient in this project is a special piece of hardware, based on Xilinx company's FPGA XC6264 chips which grow and evolve cellular automata based neural network circuits (modules) at electonic speeds. This machine, called a "CBM" (CAM-Brain Machine), can update the cellular automata (CA) cells which form the basis of the neural network at a rate of 150 Billion a second, and can complete a full run of a genetic algorithm with tens of thousands of circuit grows and fitness evaluations of those grown circuits in about one second. Hence the CBM will make "brain building" practical. Tens of thousands (and higher magnitudes) of evolved neural net modules can be evolved and assembled into humanly defined artificial brain architectures. The cellular automata based neural net model used in the CBM had to be simple enough to be implementable in state-of-the-art programmable logic (the Xilinx XC6264 chips). The constraints imposed by the electronics were rather severe, so we could not afford to give many bits to the states of the neural signals which traverse the grown neural nets. In fact, the model we use is called "CoDi" (Collect and Distribute) and uses only single bit signaling [1]. Thus the inputs and outputs of each "CoDi module" are spiketrains.

We were then faced with the problem of interpreting the meaning of a spiketrain input or output (i.e. choosing a representation for the spiketrains). After some initial experimentation with various spiketrain representations, we eventually settled on one we called "SIIC" (Spike Interval Information Coding" [3], which convolves the spiketrain output with a digitized analog convolution function (see section 3 for an explanation, and Fig. 1 for the convolution function). The result of this convolving (convolution) is an analog waveform (usually time varying) output which can then be compared to some user supplied analog target waveform. The fitness of the CoDi module (the CA based neural net circuit) grown and signaled by the CBM is then a function of the sum of the absolute differences between the target and the actual analog waveform values at each clocktick. Experiments were performed using SIIC to generate random "Fourier" curves of the form $\sin(t) + 0.3\cos(2t) + 0.5\sin(3t)$ etc. Constantly firing binary inputs were supplied to a CoDi module, which was evolved to output a spike train which when convolved with the SIIC gave the above "Fourier" curve quite accurately. See Fig. 2. Thus the SIIC enabled users ("evolutionary engineers" (EEs)) to convert the abstract spiketrains into a visually comprehensible analog wave form, whose fitness can be easily measured when compared to a target waveform.

However, despite the success of the SIIC, we felt that this was only half the story. We needed some process which would perform the opposite task, namely converting an analog waveform into a spike train. This is done with the "Hough Spiker Algorithm (HSA)" which is explained in section 4.

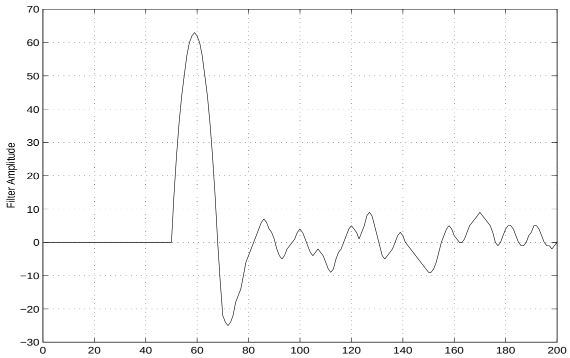

Figure 1: Decoding filter for the spike trains.

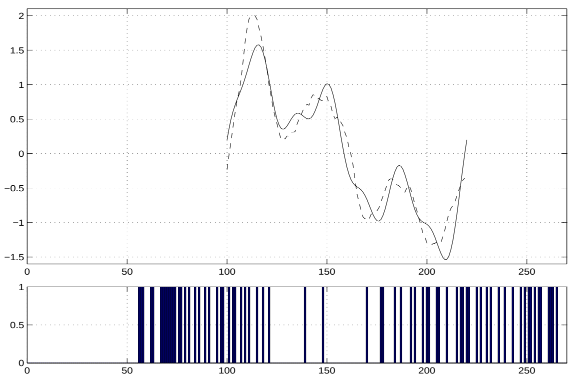

Figure 2: Sum of sines and cosines generated by the CoDi model and SIIC method.

The SIIC and the HSA combined will be very useful to evolutionary engineers (EEs), because they will be able to think entirely in terms of analog waveform inputs and outputs when evolving CoDi modules.

An EE will be able to use the HSA (the ”spiker”) in the context of CoDi module evolution. The EE specifies (for example) two analog inputs I1 and I2. Each of these inputs are passed through the HSA, resulting in two spiketrains ST1 and ST2, which are input to each CoDi module in the genetic algorithm population. Each CoDi module outputs (usually) a single spiketrain STout which passes through the SIIC convolver resulting in an output analog waveform WFout. This output waveform can then be compared to a target waveform WFtarg to measure the CoDi module's fitness. Note, that the inputs to a CoDi module need not begin as analog waveforms. They can take the form of raw bit streams and be input directly to the CoDi module without a spiking conversion. For example, if a CoDi module is to be used to detect some ”visual” pattern of 1 bit pixels, then those 1 bit pixels can be supplied directly to the CoDi module's input points. However, with the HSA approach, it may be possible to take analog input, e.g. in the form of speech sounds or phonemes etc, and convert them to spiketrains beforehand. Using the HSA and the SIIC as two transforming procedures (i.e. from analog to spiketrain and back), the EE is freed from having to think in terms of spiketrains, which may be difficult to interpret. These two transforms will allow the evolution of CoDi modules to be automated to some extent. For example, one can imagine EEs drawing or computer generating an initial input analog waveform that represents some pattern to be detected, i.e. a CoDi module is to be evolved that gives a high output when that particular waveform is presented to it, and a low output for any other waveform. Since the CBM can evolve such a module in about a second, it will be possible to evolve many such modules rather quickly and then assemble them into humanly defined artificial brain architectures.

Actually, evolving a pattern detector as suggested above may involve several sequential and partial fitness measurements. For example, to evolve a particular waveform WFa detector module, would involve presenting the waveform WFa to it (the ”positive” example) and then sequentially, ”negative” examples which differ from it. The target output waveform for the positive example would be a high output (strong activation) for the positive example fitness score F+, and a low output (weak activation) for the negative examples fitness scores F(i)-, for each of the negative examples ”i”. The total fitness of the module would be the sum of the partial fitnesses (F+ + F(i)- for all ”i”). To change the weighting of the partial fitnesses, the number of clocks ticks over which the partial fitness measurements occur can be changed, e.g. the time for the positive example measurement could be increased to equal the total number of clocks ticks for the negative examples. The CBM itself clears out the internal 1 bit signals for each partial fitness measurement, and then sums the partial fitness values, while using the same circuit.

The above gives an overview of the context in which the Hough Spiker Algorithm (HSA) operates, otherwise simply supplying the algorithm itself would be rather meaningless. The remainder of this paper is structured as follows. Section 2 gives a brief description of the ”CoDi-1Bit” model [1], which is implemented by the CAM-Brain Machine (CBM) whose electronic restrictions impose a 1 bit neural signaling model. This 1 bit signaling then requires interpretation, and hence the need for transformations such as the SIIC and the HSA. Section 3 gives a brief description of the SIIC (Spike Interval Information Coding) representation which converts a spiketrain into an analog signal. Section 4 presents the Hough Spiker Algorithm (HSA) itself which does the opposite, i.e. converts an analog signal into a spiketrain, and shows some results of simulation experiments. Section 5 presents a sample of evolved CoDi modules, showing their functionalities and evolvabilities. Section 6 summarizes.

2 The ”CoDi-1Bit” Neural Network Model & CAM-Brain Machine (CBM)

The CBM implements a so called ”CoDi” (i.e. Collect and Distribute) [1] neural model. It is a simplified cellular automata based neural network model developed at ATR HIP

(Kyoto, Japan) in the summer of 1996 with two goals in mind. One was to make neural network functioning much simpler and more compact compared to the original ATR HIP model, to achieve considerably faster evolution runs on the CAM-8 (Cellular Automata Machine), a dedicated hardware tool developed at Massachusetts Institute of Technology in 1989.

In order to evolve one neural module, a population of 30-100 modules is run through a genetic algorithm for 200-600 generations, resulting in up to 60,000 different module evaluations. Each module evaluation consists of - firstly, growing a new set of axonic and dendritic trees, guided by the module's chromosome. These trees interconnect several hundred neurons in the 3D cellular automata space of 13,824 cells (24*24*24). Evaluation is continued by sending spiketrains to the module through its efferent axons (external connections) to evaluate its performance (fitness) by looking at the outgoing spiketrains. This typically requires up to 1000 update cycles for all the cells in the module.

On the MIT CAM-8 machine, it takes up to 69 minutes to go through 829 billion cell updates needed to evolve a single neural module, as described above. A simple "insect- like" artificial brain has hundreds of thousands of neurons arranged into ten thousand modules. It would take 500 days (running 24 hours a day) to finish the computations.

Another limitation was apparent in the full brain simulation mode, involving thousands of modules interconnected together. For a 10,000-module brain, the CAM-8 is capable of updating every module at the rate of one update cycle 1.4 times a second. However, for real time control of a robotic device, an update rate of 50-100 cycles per module, 10-20 times a second is needed. So, the second goal was to have a model which would be portable into electronic hardware to eventually design a machine capable of accelerating both brain evolution and brain simulation by a factor of 500 compared to CAM-8. Now that these two transforms exist, it will be a lot more practical now for EEs to evolve CoDi modules quickly and easily, provided of course that the evolvability of the modules is adequate.

The CoDi model operates as a 3D cellular automata (CA). Each cell is a cube which has six neighbor cells, one for each of its faces. By loading a different phenotype code into a cell, it can be reconfigured to function as a neuron, an axon, or a dendrite. Neurons are configurable on a coarser grid, namely one per block of 2*2*3 CA cells. Cells are interconnected with bidirectional 1-bit buses and assembled into 3D modules of 13,824 cells (24*24*24).

Modules are further interconnected with 92 1-bit connections to function together as an artificial brain. Each module can receive signals from up to 92 other modules and send its output signals to up to 32,768 modules. These intermodular connections are virtual and implemented as a cross-reference list in a module interconnection memory (see below).

In a neuron cell, five (of its six) connections are dendritic inputs, and one is an axonic output. A 4-bit accumulator sums incoming signals and fires an output signal when a threshold is exceeded. Each of the inputs can perform an inhibitory or an excitatory function (depending on the neuron's chromosome) and either adds to or subtracts from the accumulator. The neuron cell's output can be oriented in 6 different ways in the 3D space. A dendrite cell also has five inputs and one output, to collect signals from other cells. The incoming signals are passed to the output with an 5-bit XOR function. An axon cell is the opposite of a dendrite. It has 1 input and 5 outputs, and distributes signals to its neighbors. The "Collect and Distribute" mechanism of this neural model is reflected in its name "CoDi". Blank cells perform no function in an evolved neural network. They are used to grow new sets of dendritic and axonic trees during the evolution mode.

Before the growth begins, the module space consists of blank cells. Each cell is seeded with a 6-bit chromosome. The chromosome will guide the local direction of the dendritic and axonic tree growth. Six bits serve as a mask to encode different growth instructions, such as grow straight, turn left, split into three branches, block growth, T- split up and down etc. Before the growth phase starts, some cells are seeded as neurons at random locations. As the growth starts, each neuron continuously sends growth signals to the surrounding blank cells, alternating between "grow dendrite" (sent in the direction of future dendritic inputs) and "grow axon" (sent towards the future axonic output). A blank cell which receives a growth signal becomes a dendrite cell, or an axon cell, and further propagates the growth signal, being continuously sent by the root neuron, to other blank cells. The direction of the propagation is guided by the 6-bit growth instruction, described above. This mechanism grows a complex 3D system of branching dendritic and axonic trees, with each tree having one neuron cell associated with it. The trees can conduct signals between the neurons to perform complex spatio-temporal functions. The end-product of the growth phase is a phenotype bitstring which encodes the type and spatial orientation of each cell.

3 The Spike Interval Information Coding Representation, "SIIC"

3.1 Choosing a Representation for the CoDi-1Bit Signaling

The constraints imposed by state-of-the-art programmable (evolvable) FPGAs in 1998 are such that the CA based model (the CoDi model) had to be very simple in order to be implementable within those constraints. Consequently, the 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 0's and 1's 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 the robot kitten 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 [4]. 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 many papers on the results of this convolution representation work [3]. From Fig. 2 and other similar experiments, we thought the results were good enough to settle on this representation. The CBM will implement this representation in the FPGAs when measuring fitness values at electronic speeds.

3.1.1 Simplified Example

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

```
time-shifted spike train :  1 0 0 1 0 1 1 ---> (moves left to right)

convolution filter :   1  4  9  5 -2

  1  0  0  1  0  1  1
                       0  0  0  0  0  : 0     t = -1

     1  0  0  1  0  1  1
                       1  0  0  0  0  : 1     t = 0

        1  0  0  1  0  1  1
                       1  4  0  0  0  : 5     t = 1

           1  0  0  1  0  1  1
                       0  4  9  0  0  : 13    t = 2

              1  0  0  1  0  1  1
                       1  0  9  5  0  : 15    t = 3

                 1  0  0  1  0  1  1
                       0  4  0  5 -2  : 7     t = 4

                    1  0  0  1  0  1
                       0  0  9  0 -2  : 7     t = 5

                       1  0  0  1  0
                       1  0  0  5  0  : 6     t = 6

                          1  0  0  1
                       0  4  0  0 -2  : 2     t = 7

                             1  0  0
                       0  0  9  0  0  : 9     t = 8

                                1  0
                       0  0  0  5  0  : 5     t = 9

                                   1
                       0  0  0  0 -2  : -2    t = 1
```

Hence, the time-dependent output of the convolution filter takes the values $(0, 1, 5, 13, 15, 7, 7, 6, 2, 9, 5, -2)$. This is a time varying analog signal, which is the desired result.

4 The ”Hough Spiker Algorithm” (HSA) for Deconvolution

Section 3 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 (digitized, i.e. binary 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 spike trains. 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 spike train from a time varying analog signal is now presented. It is called the ”Hough Spiker Algorithm” (HSA) and can be viewed as the inverse of the convolution algorithm described above in section 3.

To give an intuitive feel for this deconvolution algorithm, consider a spike train consisting of a single pulse (i.e. all 0's 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: $(100000 \ldots t = +\infty)$ 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 spike train is a series of spikes with time delays between them, so too the convolved spike train will 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:

```
spike train        1 1 0 1 0 0 1
convolution filter 1 4 9 5 -2

    t -> 0  1  2  3  4  5  6  7  8  9 10
out:
 1       1  4  9  5 -2
 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
         -------------------------------
         1  5 13 15  7  7  6  2  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 value of the convolution function as in the simple example of the previous section:

```
         1  5 13 15  7  7  6  2  9  5 -2
compare: 1  4  9  5 -2                       convolution function values
```

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.

Below is the C code for the algorithm. "`DUR`" is the duration of the spike train (the number of bits), "`error`" is the sum of the differences of the convolution value and waveform value at each clock tick, "`N_CONV`" is the number of elements (integers) in the convolution function, "`wave`" stores the value of the waveform at each clock tick, "`bits`" stores the spiking history (0 or 1) for each clocktick, "`thresh`" is the threshold value to decide when error is small enough for a spike, "`conv_fn`" contains the values of the convolution function.

```
void deconv(int wave[ ], char bits[ ], int thresh)
{
  int i,j,ni;
  int error;

  for(i=0;i< DUR;i++) {
     error=0;
     for(j=0;j< N_CONV;j++) {
        if(i+j>DUR) break;
        if(wave[i+j]< conv_fn[j]) error+=conv_fn[j
     }

     /* if error is okay, SPIKE! */
     if(error< thresh) {
        bits[i]=1;
        for(j=0;j< N_CONV;j++) {
           if(i+j>DUR) break;
           wave[i+j]-=conv_fn[j];
        }
     }
     else bits[i]=0;
  }
}
```

Fig. 3 shows an examples of the HSA in action. The original input analog signal O_i is shown as a stippled line. The spike train resulting from the analog input is sent into the SIIC convolutor (shown in Fig. 1). The resulting analog output should be very close to the original O_i and is shown as a solid line. The third line near the bottom is the absolute difference (error) between the two analog signals. The HSA seems to work well when the values of the waveforms are large and do not take values close to zero, and do not change too quickly relative to the time width of the convolution filter window. An example of applying too stringent an analog wave form is shown in Fig. 4. It may be possible to simply add a constant value to incoming analog signals before spiking them and to ensure that the analog signal does not change too rapidly.

Note however, that the HSA deconvolution algorithm was only discovered very recently, so the neural net module evolution that is discussed in section 5, does not use it. The inputs to these modules as specified by the EE (evolutionary engineer) were binary, not analog.

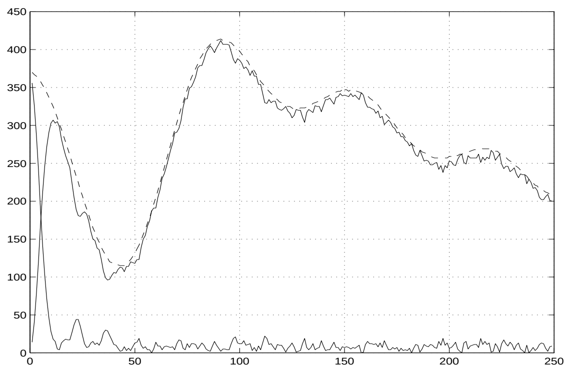

Figure 3: A good result of the Spiker Module.

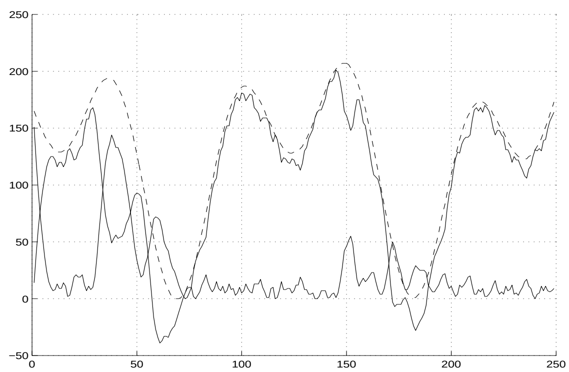

Figure 4: A not-so-good result of the Spiker Module.

5 A Sampler of CoDi-1Bit Evolved Neural Net Modules

Since the whole point of using the CBM is to attain a high evolution speed, it is useful if the representation chosen to interpret the 1 bit signals which enter and leave the CoDi modules can be unique, otherwise several representations would need to be implemented in the electronics. (For the CBM to be efficient, i.e. to evolve CoDi modules in about 1 second, fitness measurements need to be performed at electronic speeds, which implies that the representation chosen for the signals be implemented directly in the hardware). We chose the SIIC to be our unique representation. However, as mentioned at the bottom of section 4, most of the evolutionary experiments presented here were already undertaken before the SIIC representation was chosen. Since the results of these earlier experiments are interesting in their own right, we report on them here. They show to what extent that CoDi modules are evolvable and the power of their functionality. The evolution of SIIC-representation-based and HSA-based modules will be the subject of work in the very near future, given that both algorithms are now ready. So is the CBM multi-module simulation code, so progress should be rather rapid in the coming months prior to the delivery of the CBM itself. Once the CBM is delivered, multi-module systems should be built as fast as we can dream them up. The bottleneck in building large scale multi-module systems will become human creativity lag, not module evolution lag (as was the case with software evolution speeds in the ”pre-CBM era”.) We now provide a sample of evolved CoDi neural net modules, their specified functionalities, and their actual performances, to give a feel for what they can do.

5.1 XOR Module

If a CoDi module could not be evolved to perform something as simple as an exclusive OR, then the whole CAM-Brain approach would be cast in doubt, so one of the first things we tried was to evolve an XOR module. The module size was a cube of 24 cells in each one of the axes (X, Y, Z) of the 3D CA space (the standard CBM module size, as implemented in the hardware). Two binary signals, A and B, on the $Z = 0$ face of the cube, on axon cells at coordinates of $(1, 1, 0)$ and $(7, 1, 0)$ were input over 64 clock ticks. There were 4 test cases (using a multi-test fitness measurement:
Case 1:
input A was a steady stream of 1's for 64 clocks, and
input B was a steady stream of 1's for 64 clocks.
Case 2:
input A was a steady stream of 1's for 64 clocks, and
input B was a steady stream of 0's for 64 clocks.
Case 3:
input A was a steady stream of 0's for 64 clocks, and
input B was a steady stream of 1's for 64 clocks.
Case 4:
input A was a steady stream of 0's for 64 clocks, and
input B was a steady stream of 0's for 64 clocks.

At the output point $(4, 4, 0)$, the target output values were a stream of 0's, 1's, 1's and 0's respectively, as a boolean XOR logic function. The total fitness value was the sum of the partial fitness values of the 4 test cases, i.e. the total number of correct output values, giving a theoretical maximum fitness value of $4 * 64 = 256$. However, since there are inevitable time delays, as the signals progress through the CA cells of the module, this perfect score is not possible.

The elite module, (population of 15, mutation rate of 0.005 per bit per generation, one point crossover rate of 0.6 per chromosome per generation), gave after about 20 generations the following outputs for the 4 cases, with partial fitness scores of 63, 57, 56, 64 respectively. Hence, total fitness was 240.
Case 1:
00000001000000000000000000000000000000000000000
00000000000000000
Case 2:
00000001111111111111111111111111111111111111111
11111111111111111
Case 3:
00000000111111111111111111111111111111111111111
11111111111111111
Case 4:

00000000000000000000000000000000000000000000000
00000000000000000

So the XOR module evolved fairly quickly and easily. Note that the XOR case is an example of a *multi-test fitness measurement*. For each test, a partial fitness score is obtained, which is later summed with other partial fitess scores, resulting in the overall fitness score of the CoDi module. Between each partial test, the signal states of the module are cleared, and a new test run is performed using the same circuit, but with different input. The same occurs in the CBM hardware.

5.2 Timer Module

One of the first experiments performed, when the CoDi-1Bit model was proposed, was to see if a CoDi module (consisting of 4K 3D CA cells, with about 150 artificial neurons in the 4K space) could evolve a ”timer”, i.e. where constantly firing binary inputs generate at a single cell output (placed elsewhere in the CA space) a string of 0's during the first 30 clock ticks, then a string of 1's during the next 20 clock ticks, and finally a string of 0's in the last 20 clocks, as shown below. This was quite a demanding evolutionary task, which was a useful test vehicle during the early evolutionary trials.
Target
00000000000000000000000000000011111111111111111
11100000000000000000000

pretty simple. (Note, that the evolution of this module did not use the SIIC or HSA approaches. Inputs and outputs were specified directly in binary signals). If a 0 appeared in the first (0) block, score 1 point. If a 1 appeared in the second (1) block, score 3 points. If a 0 appeared in the third block (0), score 2 points. Hence a perfect score would be $30*1+20*3+20*2 = 130$. Population size was 24, with no crossover.

The CoDi-1Bit model software simulation evolved this with a fitness of 100% in about 150 generations. A few days later, by strongly increasing the neuron density in the CA space to about 90% of the maximum value, we got the same result in about 50 generations!

5.3 Multiple Timer Module

Since a 100% fitness score does not test the limits of evolvability of a module, a more demanding output function was tried. The target output (similar to the above pattern) and the actual evolved output (placed immediately under the target pattern for comparison) were as follows:
Target
00000000000000000000000000000011111111111111111
Evolved
00000000000000000000000000000000011111111111111
Target ctd.
11100000000000000000000000011111111111111110000
0000000000000000
Evolved ctd.
11110000000000000000000000001111111111111111000
0000000000000000

The fitness definition was similar to the above. If a 0 appeared in the first (0) block, score 12 points, if a 1 appeared in the second (1) block, score 7 points, if a 0 appeared in the third block (0), score 3 points, if 1 appeared in the fourth block (1), score 2 points, if 0 appeared in the fifth block (0), score 1 point. Hence a perfect score would be $30*12+20*7+24*3+16*2+20*1 = 624$. These weightings were chosen so as to encourage the earlier outputs to be correct before the later outputs. Population size was 30, no crossover. This result converged after about 100 generations with a fitness value of 0.957.

It is interesting to note that these good results are evolving in about 100 generations or so, and yet the chromosome length is very large. The standard CBM chromosome length is of the order of 90K bits. One might think that such a long chromosome would be very slow in evolving, but this was not the case. One possible explanation for this is that there may be so many possible solutions that a suitable one is quickly found.

5.4 Switchable Dual Function Module

Our thoughts then turned to the idea of trying to evolve a module whose behaviors could be placed under switchable control, i.e. a module with dual functionality, which could be switched from one behavior to the other, depending on whether a control input was activated or not.

More specifically, two fixed position input points `IN` and `SWITCH` were placed at positions $(8, 8, 0)$ and $(16, 16, 0)$ for a rectanguloid of $24*24*18$ 3D CA cells, with a fixed output point at position $(11, 12, 9)$. If the output point was not an axon, fitness was defined to be zero.

Two experiments were run on the same module (the same CoDi module, with signal flushing between experiments). In both experiments, the `IN` input fired at every clock tick. In the first experiment, the `SWITCH` input was off for every clock tick. In the second experiment the `SWITCH` input fired for every clock tick. The module was evolved to give a very active output (lots of 1's) if the `SWITCH` was off, but a low output (few 1's) if the `SWITCH` was on. That is, the `SWITCH` acted as an inhibitor.

The bitstrings below show the outputs for the two cases, firstly with `SWITCH` off, then on. Over 90 clock ticks, the first output had 42 more 1's than the second output.
`SWITCH` off
0000000000000000000000000000000000011000111011
11111111111111111111111111111111111111111111
`SWITCH` on
0000000000000000000000000000000000001000001000
00100000100000100000100000100000100000100000

The number of 1's in the two outputs were labeled as S_1 and S_2, respectively. The fitness function (F) was defined as:

$$
\begin{aligned}
&IF(S_1 > S_2)\\
&\qquad F = 10000 * (S_1 - S_2) + 0.001 * (S_1 + S_2)\\
&IF(S_1 < S_2)
\end{aligned}
$$

$$F = 100 * (S_2 - S_1) + 0.001 * (S_1 + S_2)$$

The term $0.001 * (S_1 + S_2)$ was used to encourage circuits to give nonzero output at the output point. The terms $100 * (S_2 - S_1)$ and $10000 * (S_1 - S_2)$ encouraged differences in the two outputs, with a strong preference for the first case to give more 1's in the output.

This result was very encouraging because it shows that controllable multifunction modules, at least like this switchable function, are evolvable with the CoDi model. Such modules will be very useful when the time comes to evolve modules to be placed in "artificial brain" architectures.

5.5 Pattern Detector Module

Slightly modifying the previous set up, a pattern detector module was evolved, which was capable of distinguishing between two square wave inputs, of 111000111000... and 11111000001111100000... In this case, no switch input was used. Two experiments were run. In the first, the input was the 6-clocktick cycle square wave input, applied at the fixed input point $(8, 8, 0)$. In the second experiment, the circuit was regrown with the same chromosome and the 10-clocktick cycle square wave input was applied to the same fixed input point. The fitness definition was the same as above. Over 90 clockticks, the first output had 48 more 1's than the second output.

Square wave input 111000111000...
Output
00000000000000000000000010011011101111111111111
1111111111111111111111111111111111111111111
Square wave input 11111000001111100000...
Output
00000000000000000000000000001000100010001000100
0100010001000100010000000000010001000100010

Since the CoDi modules seem capable of evolving such detectors, it may be possible to evolve modules which are capable of detecting a specific phoneme analog input, e.g. the spike train (bitstring) that represents the time dependent analog signal. In a manner similar to the above, one could input the signal in the first experiment, and a random signal in the second, in a multi-test experiment, and evolve the phoneme detector. Maybe one could evolve a set of detectors, one for each phoneme. By using the SIIC and HSA digital/analog conversions, this kind of thing may become quite practical.

5.6 Hubel-Wiesel Line Motion Detector Module

The results of the following experiment were significant for the CAM-Brain Project as a whole, we felt. It involved the evolution of a Hubel Wiesel type line motion detector. Hubel and Wiesel won a Nobel prize for discovering that certain neural cells in the visual region of the cat's brain fired strongly when lines of light at particular orientations and speeds were shone onto a screen that the cats were observing. These cells (neurons) were detecting the motion of lines at a particular orientation. The evolution of this "Hubel-Wiesel" module used the same fitness definition and a similar methodology as in the above case. In the first experiment, a square 12*12 neuron input grid was used. At the first clock tick, the top horizontal 12 neurons were made to fire; at the second clock tick, the second horizontal row of 12 neurons was made to fire, etc, for 12 clock ticks, then the cycle was repeated. This input firing pattern simulated the motion of a line of light moving horizontally down the visual field on the retina of a cat. In the second experiment, 12 randomly positioned input neurons were fired at each clock tick. These 12 positions were randomly generated for each clock tick. This second input firing pattern simulated input noise, to be contrasted with the line motion input. Output results are shown below:

Line Motion Input Case
Output
00000000000000000000000010011010011011111111111
1111111111111111111111111111111111111111111
Random Input Case
Output
00000000000000000000000000001000000000000000010
0010101010101010101010101010101010101010101

There were 35 more 1 bit outputs in the first case than the second. Since the inputs to the second case are positioned randomly, the same neural net module will generate a different fitness value depending on the input. Nevertheless the evolution still improved over time, developing a fairly robust net giving fitness values corresponding to over 30 1-bit differences (between the two experiments) in most cases (e.g. the top 5 fitness chromosomes were saved for each generation and not crossed over or mutated. The fitness values (1-bit difference count) of these top 5 were 31, 34, 35, 30, 29 after several hundred generations). However, we have no idea how the circuit does what it does. Evolved circuits can achieve performance levels beyond what human engineers can achieve with traditional top-down design techniques, i.e. attain superior engineering performance levels, but the price is that one loses scientific understanding, due to the overwhelming structural and dynamical complexity of these CoDi circuits.

6 Summary and Conclusions

This paper presented some software simulation results of cellular automata based neural network circuit modules which will be grown and evolved at electronic speeds in special FPGA based hardware (the CAM-Brain Machine (CBM) [2], which should be delivered to ATR by January 1999). The neural net model implemented by the CBM is called "CoDi-1Bit" [1]. The constraints imposed by the electronics necessitated a very simple model, namely one whose signal states contain only 1 bit. This restriction implied that the spike trains (bit string of 1's and 0's) which are input and output to and from modules need to be interpreted. We can convert the spike train into an analog waveform using a convolution technique that we call "SIIC" (Spike Interval Information Coding [3]). The reverse process, i.e. converting from an analog waveform to a spiketrain was performed using the "HSA"

(Hough Spiker Algorithm). These two transformations allow users (”EEs” or ”evolutionary engineers”) to think entirely in terms of analog waveforms, both at input and when specifying target (desired) outputs. Thinking in analog terms is much easier than thinking in terms of spike intervals (i.e. the number of 0s between spikes (i.e. the 1s), which visually is rather meaningless. However, since the HSA was only invented very recently, the experiments reported on in this paper have the user specifying input in raw binary terms, not in analog waveforms which would be converted into spiketrains by the HSA. Nevertheless the results of the raw binary input cases are still very interesting. The results shown in section 5 make it clear that the evolvability of he CoDi-1Bit model modules is powerful and interesting.To fully test the capabilities of the CBM, we will need a CBM, but prior to its delivery, we have been simulating its performance and evolvability.

The issue of evolvability is always an open question, because not all modules evolve the way one wants. Evolutionary engineering is still a ”black art”. Criteria for good evolvability are not well understood. However, in practice, if is it found that a particular module does not evolve well, then alternative modules with different functional specifications can often be found to solve the same problem and that these alternative modules do evolve well.

The next step in the CAM-Brain Project is to design multi module systems, and to scale up the number of modules used. The CAM-Brain Machine (CBM) can update roughly 32000 modules at sufficient speed (150 Billion cellular automata (CA) cells a second) to enable real time control of a kitten robot. So with 32000 modules allowable by the hardware, BAs (brain architects) and EEs (evolutionary engineers) can afford to be ambitious. The great challenge now is how to design artificial brains. Hopefully within a few years, a new research field will be established, called simply ”Brain Building”.

Bibliography

[1] Felix Gers, Hugo de Garis, and Michael Korkin. CoDi-1 Bit: A simplified cellular automata based neuron model. In *Proceedings of AE97, Artificial Evolution Conference*, October 1997.

[2] Michael Korkin, Hugo de Garis, Felix Gers, and Hitoshi Hemmi. CBM (CAM-Brain Machine): A hardware tool which evolves a neural net module in a fraction of a second and runs a million neuron artificial brain in real time. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, July 1997.

[3] Michael Korkin, Norberto Eiji Nawa, and Hugo de Garis. A 'spike interval information coding' representation for ATR's CAM-brain machine (CBM). In *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES'98)*. Springer-Verlag, September 1998.

[4] Fred Rieke, David Warland, Rob de Ruyter van Steveninck, and William Bialek. *Spikes: exploring the neural code*. MIT Press/Bradford Books, Cambridge, MA, 1997.