Chapter 6
Associative Memory in Reaction-Diffusion Chemistry

James Stovold and Simon O'Keefe

Abstract Unconventional computing paradigms are typically very difficult to program. By implementing efficient parallel control architectures such as artificial neural networks, we show that it is possible to program unconventional paradigms with relative ease. The work presented implements correlation matrix memories (a form of artificial neural network based on associative memory) in reaction-diffusion chemistry, and shows that implementations of such artificial neural networks can be trained and act in a similar way to conventional implementations.

6.1 Introduction

Whilst, in theory at least, unconventional computing paradigms offer significant advantages [33] over the traditional Turing/von Neumann approach, there remain a number of concerns [32] regarding larger-scale applicability, including the appropriate method for programming and controlling such unconventional approaches to computation. The massive parallelism obtainable through many of these paradigms is both the basis for, and a source of problems for, much of this power. Given how quickly these approaches could overtake conventional computing methods, large-scale implementations would be incredibly exciting, and would allow for a larger proportion of possible computations to be computable [12]. The power of such parallelism can be seen by the recent advent of general-purpose graphic processing unit (GPGPU) technology, where huge speed-ups are gained by parallelising repetitive tasks. However, translating between the traditional, algorithmic approach to problem solving that is ubiquitous in computer science, and the complex,

J. Stovold · S. O'Keefe (✉)
Department of Computer Science, University of York,
York, North Yorkshire YO10 5GE, England
e-mail: simon.okeefe@york.ac.uk

J. Stovold
e-mail: jhs503@york.ac.uk

© Springer International Publishing Switzerland 2017

A. Adamatzky (ed.), *Advances in Unconventional Computing*,
Emergence, Complexity and Computation 23,
DOI 10.1007/978-3-319-33921-4_6

dynamical, nonlinear environment that most unconventional paradigms exploit, can be a highly non-trivial task.

Conrad [11] proposed the principle of a tradeoff between programmability, efficiency and evolvability of a computational system. This principle states that the price paid for having such a highly-programmable system is in terms of efficiency, evolvability or both. A corollary to this is that, in order to increase the efficiency of a computational construct, either the programmability or the evolvability must be sacrificed.

This principle underlies the field of unconventional computing. By considering alternative, non-standard approaches to computation, there is the possibility that the efficiency of the system can be increased with a minimal loss of programmability.

There are many examples of unconventional approaches to computation, each with their potential advantages, all with a notable decrease in programmability. For example, Adleman [5] showed experimentally that DNA could be used as a computational substrate by computing a solution to the Hamiltonian Path Problem; Nakagaki [28] showed that the plasmodial slime mould *Physarum polycephalum* could compute the shortest path through a maze; and Laplante [24] showed how a mass-coupled chemical reaction could implement a Hopfield network. While these are all clearly much harder to program than traditional computational substrates, the potential for much more efficient computation is present in all of them through their intrinsic parallelism. By exploring many alternate paths of computation simultaneously, certain unconventional substrates have the effect of being able to implement a non-deterministic UTM, allowing them to potentially solve NP problems in polynomial time.

The problem faced by researchers in the unconventional computing field is that in order to get access to this huge potential power, the system must be programmed, and in order to use the substrate as a general-purpose computer, the system must be reliable. These considerations are necessarily non-trivial. The reliability problem is inherent in the non-determinism, as there is no guarantee that the substrate will perform the computation intended. The programmability problem is that of finding 'algorithms' that can be used to perform meaningful computation on the substrate.

Conrad [12] quantified the idea that general-purpose machines are not particularly general-purpose, and that only certain problems can be solved using conventional computational methods. From this, it becomes clear that different approaches to the same problem—whilst being Turing equivalent—may be different from other perspectives. For example, consider *Turing tarpit* languages such as 'OISC' [15] and 'brainfuck' [*sic*] [27]. Although these languages are Turing complete in the same sense as languages such as C and Java, they have particularly obfuscated programs. These systems serve to elucidate the difference between being Turing computable and being *usably* computable. While an extreme example, 'brainfuck' shows the importance of programmability in general-purpose computers. If this idea is extended to computability in general, just because a system is labelled as Turing complete does not mean that it should be explicitly used as a Turing machine.

Recent approaches have attempted to utilise parallelism to increase the number of computable functions. In conventional computing paradigms, this consists of many UTMs (Universal Turing Machines) working in parallel on the same function. The use of GPGPU processing has been the primary result of this, but the problem with using multiple forms of the same computational construct is that each computational unit has the same limitations. Field-programmable gate arrays (FPGAs) help with this, in that processing units can be implemented alongside bespoke hardware, to speed up regularly-used operations, but problems arise in terms of the time taken to reprogram the devices. Because of this, the reprogrammability—and hence the programmability—of FPGAs as a computational construct is reduced in response to the efficiency gained.

If unconventional computing paradigms are considered as a complement to UTMs, the number of computable functions is likely to be increased, as the limitations of each system may offset those of the other. This is because there will inevitably be certain functions that UTMs are better suited to, and other functions that unconventional approaches are better suited to.

In this paper, we present an implementation of an artificial neural network (a correlation matrix memory) in an unconventional paradigm (reaction-diffusion chemistry). This implementation will serve as a method of simplifying the process of programming the unconventional paradigm (we already know how to program correlation matrix memories, and it's simpler than programming reaction-diffusion reactors).

The paper is organised as follows: in Sect. 6.2 we introduce diffusive computation and reaction-diffusion chemistry, and show how this can be used for computation; in Sect. 6.3 we describe associative memory and correlation matrix memories; Sect. 6.4 discusses the methods used to get our results; Sects. 6.5, 6.6, 6.7, and 6.8 detail the process of designing and testing the correlation matrix memories in reaction-diffusion chemistry. Finally, Sect. 6.9 presents our conclusions.

6.2 Diffusive Computation and Reaction-Diffusion Chemistry

We define *diffusive computation* as an unconventional approach to computing that harnesses the diffusion of particles as a representation of data. The power of diffusive computing systems come from their highly parallel architectures, with the emergent behaviour of interactions between diffusing particles dictating the operation of the system. A simple example of a diffusive computing system is a cellular automaton. While nothing physical diffuses in a cellular automaton, the emergent behaviour of structures such as gliders gives rise to the diffusion of information. A simple rule for the cellular automaton can be used to give complex behaviour that is shown to be similar to that of reaction-diffusion systems [4].

One of the primary benefits of considering this unconventional computing paradigm is the potential it offers compared to existing systems. The properties of diffusive computation compared with general-purpose machines make it a much better candidate for producing systems that have similar complexity to that of the brain [29].

There are many different forms of diffusive computation, such as silicon-based diffusion processors [3], slime mould [2] and cellular automata [4]. In this paper, we consider a fundamental form of diffusive computation: reaction-diffusion chemistry.

Reaction-diffusion chemistry (RD chemistry) is a form of chemical reaction that changes state in such a way that wavefronts of reagent appear to flow (or diffuse) across the solution. The most widely-used models are based on the Belousov–Zhabotinsky reaction [9, 40], which may display single waves of reaction or an oscillation between two (observable) states, depending on the setup used.

6.2.1 Belousov–Zhabotinsky Reaction

Belousov [9] discovered a visibly periodic chemical reaction based on Bromate and citric acid. The reaction cycled through a series of colour changes on a timescale of many seconds—making it very easy for humans to observe the changes in state. Zhabotinsky [40] then enhanced the effect of the colour change by replacing the citric acid used by Belousov with malonic acid. This reaction was soon termed the *Belousov–Zhabotinsky* (BZ) reaction.

The chemical reaction can be distributed across a wide area, such as in a Petri dish, with the depth of the liquid reduced to less than 2 mm. This can also be achieved by binding the catalyst to a chemical gel, with a thin layer of chemicals over the top. This 'thin-layer' reactor allows the system to be considered as a pseudo-two-dimensional system. As the reaction oscillates, each point in the reactor will react only with the chemicals available locally to it, meaning that a small disturbance in the reaction—such as an inhomogeneity (gas bubble, speck of dust etc.)—will cause a wave of colour change to propagate across the reactor [41], and can result in some very interesting effects (such as those in Fig. 6.1).

There are many potential catalysts for BZ reactions, each with different properties. Ruthenium bipyridyl complex ($Ru(bipy)_3^{2+}$) is particularly interesting as a catalyst in BZ reactions as it is photosensitive. This in an incredibly helpful property, as the rate with which ruthenium catalyses the reaction is inversely proportional to the level of illumination. This can be harnessed as a method of controlling the reaction

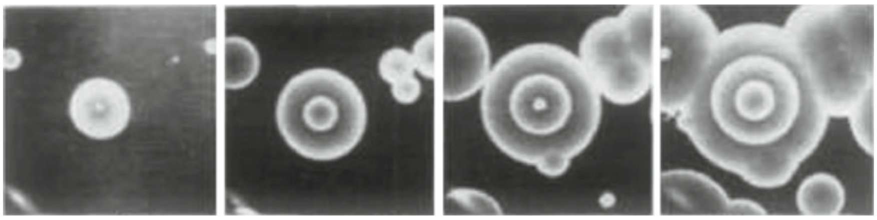


Fig. 6.1 A 'leading centre' pattern, observed in a thin-layer Belousov–Zhabotinsky reaction (from [39])

for computation [23]. By disabling different regions of the reactor, either through illumination or by constraining the distribution of catalyst in a chemical gel, circuits can be constructed that allow logic gates and other computation to be performed [6].

6.2.2 Applications

The different applications of diffusive chemical reactions have been studied in great depth since it was proposed as a basis for morphogenesis by [37]. Between 1986 and 1989, a series of seminal papers proposed a method of using RD chemistry as a computational medium, showing that it can be used for a number of image processing operations: in particular, contrast modification and contour discernment [21–23]. These papers sparked much research interest, particularly given the CPU-intensive nature of sequential image processing algorithms and how intrinsically parallelised the process becomes through the use of RD chemistry.

Since then, a large amount of work has been invested examining the computational properties of RD chemistry. RD chemistry was shown to have the intrinsic ability to find the shortest path through a maze, with a time complexity that is independent of the complexity of the maze, only dependent on the distance between start and finish [31]. This is because the propagating waves explore every reachable path simultaneously, but at each junction, only the first wave to arrive can continue (because of the refractory period of the waves, which will inhibit later waves).

There have been many other applications of RD chemistry, for example, construction of a Voronoi diagram [35]; a chemical diode [6]; and the involute for a static object [25].

The computational universality of RD chemistry (and other nonlinear media) has been discussed in depth [1], and many authors have suggested different approaches to constructing logic gates. The use of capillary tubes to constrain the reaction has been considered, using the diameter of the tube to control the behaviour of the wave, which can be used to implement logic gates (including a universal set) [36]. Various logic gates, a memory cell and a coincidence detector have been constructed using simulations of the diffusion dynamics when constrained to channels [26].

Gorecki et al. [16] take this idea a step further, constructing the coincidence detector, which implements an AND gate, both in simulation and in experiments. The coincidence detector (shown in Fig. 6.2) allows a single wave to pass through in either direction, but the presence of two waves travelling towards each other raises the activator concentration between the colliding waves higher than either of the single waves would alone. This higher concentration then diffuses farther into the gap in catalytic material compared to the concentration from the single waves. The lower part of the gate, which acts as the output, can then be stimulated by this diffusion. The same paper then presents an extension of this idea by constructing a cascadable binary counter.

If a series of waves arrive at a gap in chemical medium within a certain time period, the concentration on the far side of the gap will rise higher than with a single

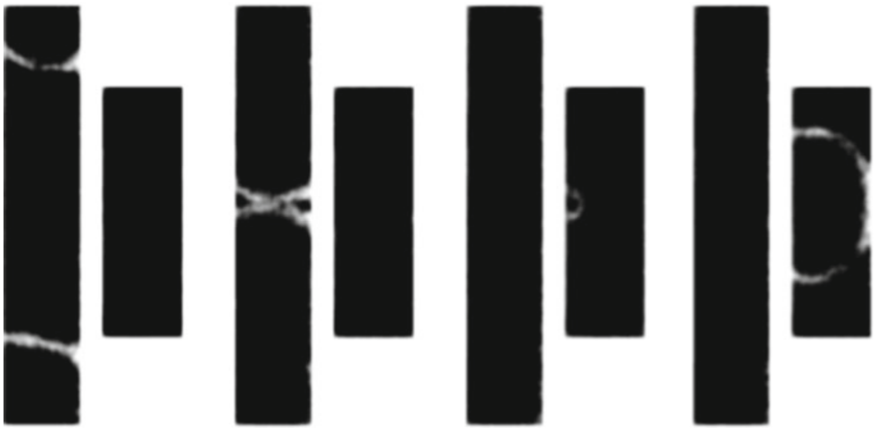


Fig. 6.2 Four photographs of a chemical coincidence detector. Each photo has two sections of active substrate, the *left-most* acting as inputs, the *right-most* as output. The waves are inputted (from the *top* and *bottom*), and as they collide, the activator concentration between them rises higher than if only one wave was present. This results in the activator diffusing farther into the gap between input and output section, producing a wave on the output section (from [16])

wave, because of the diffusion of chemicals across the gap [17]. If this stimulation is sufficient, it can cause an excitatory response on the far side of the gap [13], which is directly analogous to the behaviour of spiking neurons [34].

6.3 Associative Memory

Associative memory is a form of memory that associates stimuli with responses. There are two forms of associative memory, *autoassociative* memory and *heteroassociative* memory. In autoassociative memory, the associated, complete pattern can be retrieved from a small cue: for example, given the cue 'lived happily ever…' most people would immediately think of '…after'. However, with heteroassociative memory, other patterns that are associated with the cue are retrieved, but not necessarily the completed pattern from the cue. The brain works in a similar way to a combination of these forms of memory, as when presented with the same cue as above, most people would still complete the associated pattern, but subsequently think of a fairytale or children's story.

From a computational perspective, this could be implemented as an artificial neural network [18] that behaves in a similar fashion, where a noisy pattern or subset of the pattern could be used as the stimulus to the network in order to retrieve the complete, clean pattern.

The basic idea of an artificial associative memory is to train a fully-connected, two layer artificial neural network (see Fig. 6.3) in such a way that, when presented

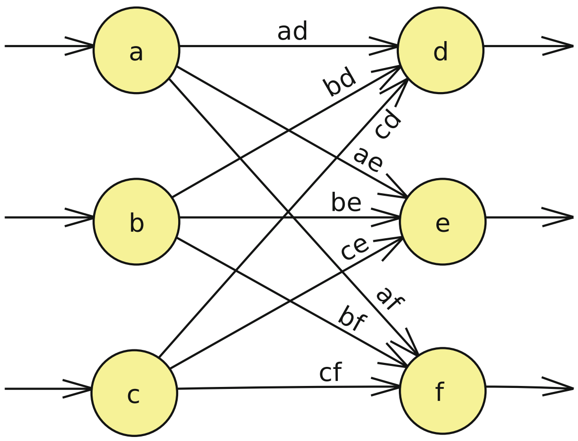


Fig. 6.3 Two-layer artificial neural network showing the relationship between input–output pairs and the correlation matrix $\mathcal{M}$, as given in matrix (6.1), below

with the stimulus vector x, the connections between the input and output layers will produce the associated output vector y [10]. In training such a network, the weights on these connections are crucial, as they are what define the association. The correlation weight matrix, w, stores associations representing p patterns, and is produced via:

$$w_{ij} = \sum_{1}^{p} x_i^{(p)} \cdot y_j^{(p)}$$

Because of the importance of the correlation weight matrix, one form of artificial associative memory uses it as the entire basis for the computational construct. This is the *Correlation Matrix Memory*.

6.3.1 Correlation Matrix Memories

Willshaw [38] proposed a potential method for the brain to store memories inspired by 'correlograms,' produced by shining light through two pieces of card with pinholes on. The resulting spots of light were captured on a third piece of card, and pertained to the correlation between the patterns on the first two cards. By capturing the correlation present on the third card, an inverted pattern could be used to retrieve the original pattern. Willshaw [38] then proposed a discretised form of correlogram, called an 'associative net' that captured this idea, but without the imperfections of an experimental setup. By presenting a series of input patterns and output patterns to the associative net simultaneously, the associations can be built up within the net and the patterns later retrieved by presenting one of the original patterns.

The idea of an associative net later developed into the binary Correlation Matrix Memory (CMM) [20]. The CMM is a matrix-based representation of a fully-connected two-layer neural network (one input layer, one output layer), where binary values in the matrix represent the binary weights on the connections between the two layers. As such, the neural network in Fig. 6.3 would be represented by the CMM, $\mathcal{M}$, with k input–output pairs $\mathcal{I}$ and $\mathcal{O}$:

$$\begin{array}{c|c} \begin{pmatrix}\mathcal{I}_k\end{pmatrix}\begin{pmatrix} \quad \mathcal{M} \quad \end{pmatrix} & \begin{pmatrix}a\\b\\c\end{pmatrix}\begin{pmatrix}ad&ae&af\\bd&be&bf\\cd&ce&cf\end{pmatrix} \\ & \\ \begin{pmatrix}\quad \mathcal{O}_k \quad\end{pmatrix} & \begin{pmatrix}d&e&f\end{pmatrix}\end{array} \tag{6.1}$$

Before training, the initial matrix $\mathcal{M}$ would be filled with zeros (as there are no associations stored in the network). As the k binary-valued input–output pairs are presented to the network, the associations are built up in the matrix.

Upon recall, we get [18]:

$$\mathcal{O} = \mathcal{M}\mathcal{I}_r$$

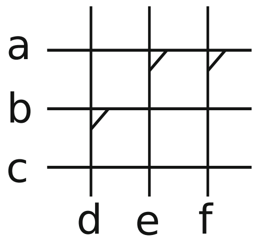
where $\mathcal{I}_r$ is the input pattern, and $\mathcal{O}$ is the output from the network trained with associations stored in $\mathcal{M}$. The desired output pattern, $\mathcal{O}_r$ is currently combined with noise from the other patterns stored in the network, e_r, hence:

$$\begin{aligned} \mathcal{O} &= \mathcal{O}_r + e_r \\ e_r &= \sum_{\substack{k=1 \\ k \neq r}}^{N} (\mathcal{I}_k^T \mathcal{I}_r)\, \mathcal{O}_k \end{aligned}$$

The output vector $\mathcal{O}$ can be thresholded to a appropriate level, which (depending on how saturated the network is) should leave the desired output vector $\mathcal{O}_r$. This process can be used as a method of generalisation, allowing the network to retrieve complete patterns from a noisy or distorted cue.

Because the association matrix, $\mathcal{M}$, is a binary matrix (only allowing 0 or 1), the network can also be represented by a grid of wires, with connections between horizontal and vertical wires representing the 1 s in $\mathcal{M}$, and hence the associations stored in the network (see Fig. 6.4).

The binary nature of the CMM lends itself to be efficiently implemented using RD chemistry. The propagating waves in RD chemistry can be used to implement the binary signals (0/1 encoded as absence/presence of a wave) that are required to perform training and recall in the CMM. Interactions between the input waves and correlation matrix can allow recall to occur without external influence. Section 6.5 discusses how the correlation matrix, $\mathcal{M}$, may be stored in an RD memory structure.



$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Fig. 6.4 A CMM can be viewed as an electrical circuit, with associations between input–output pairs represented by the connections between input and output wires (*left*). The matrix representation of this diagram is given as the correlation matrix on the *right*

6.4 Methods

Given the speed of the chemical reaction, we have simulated the dynamics of the BZ reaction instead of using a wet lab setup. A large number of authors have taken this approach previously, including some of the early papers by Rovinsky [30] who give a mathematical model of the dynamics alongside the chemical definition.

6.4.1 Simulation

We simulate a diffusive chemical reaction using the Oregonator model [14]:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{1}{\varepsilon}\left[u - u^2 - (fv + \phi)\cdot\frac{u-q}{u+q}\right] + D_u\nabla^2 u \\ \frac{\partial v}{\partial t} &= u - v \end{aligned} \tag{6.2}$$

with parameter values as given in Table 6.1. The parameters (ε, f, ϕ, q, D_u) are described in detail in [19], but (briefly): ε is a scaling factor, f is the stoichiometric coefficient (a conservation of mass parameter), q is a propagation scaling factor, ϕ is representative of the illumination, and D_u is the diffusion coefficient for the solution. These are fixed over the course of the reaction and reactor, other than ϕ, which varies spatially across the reactor in order to construct 'circuits.'

The diffusion term ($D_x\nabla^2 x$ for chemical x) is provided for the activator u, but not for inhibitor v. This is because the simulation assumes that v is bound to an immobile substrate, such as a chemical gel. Equation 6.2 are integrated using an explicit forward Euler integration, with timestep $\delta t = 0.001$ and five-node discrete Laplace operator with grid spacing $\delta x = 0.25$. Waves are started through the simultaneous stimulation of 15 adjacent pixels by setting their activator concentration (u) to 1.0.

Table 6.1 Parameter values for excitable Belousov–Zhabotinsky simulation

Parameter	Value
ε	0.0243
f	1.4
ϕ_{active}	0.054
$\phi_{passive}$	0.0975
q	0.002
D_u	0.45
δt	0.001
δx	0.25

6.4.2 Extracting Results

Two different methods are used to extract results from the simulation. The first is to use time-lapse images produced by the simulation to visually check the presence or absence of a wave in a given channel; the second is through the use of 'pixel traces'.

Time-lapse images were produced by thresholding the activator concentration at $u = 0.04$ every 1000 simulated time-steps. All points above $u = 0.04$ were marked as white, while the rest left as they were (see Fig. 6.5).

The idea behind pixel traces, on the other hand, is to record the concentration values of a small set of pixels throughout the entire run of the simulation. This record of chemical concentrations can then be used to produce a graph, showing how the concentrations vary over time. Alternatively, if many runs are being performed, and the arrival of a single wave is all that matters, the activator concentration can be thresholded to an appropriate value, and used to determine the approximate arrival time of the wave. The graph presented in Fig. 6.6 shows an example pixel trace extracted from the simulator, and the threshold point. All time values with concentration above this threshold are then extracted, and the centre of the 'bump' is calculated, as an approximation of the waves arrival time at the specified pixel. For the work presented, this threshold value is held constant at $u = 0.54$.

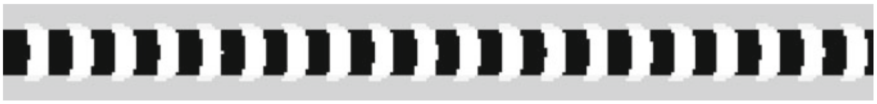


Fig. 6.5 Example time-lapse output from the simulator. The activator concentration is thresholded at $u = 0.04$ every 1000 simulated time-steps, and marked as *white*. The illuminated (passive) regions of reactor (high ϕ) are signified by *grey* background, and non-illuminated (active) regions of reactor (low ϕ) are *black*

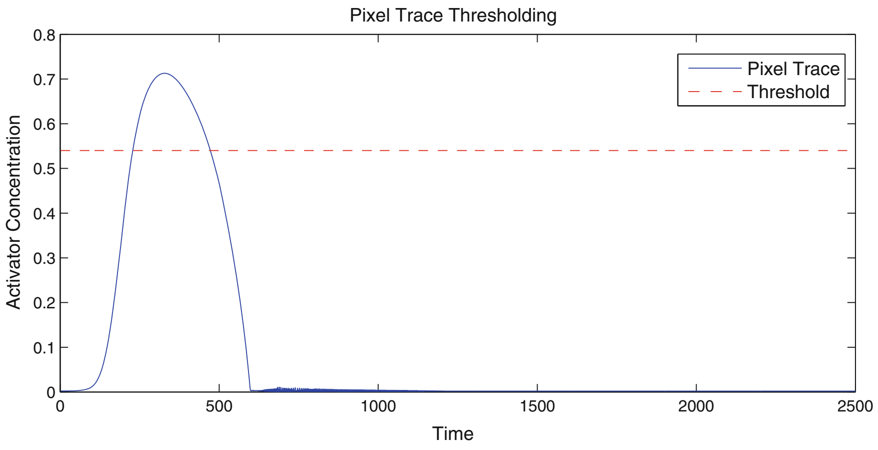


Fig. 6.6 Graph showing the pixel trace values over time (*solid line*) and threshold point (*dashed line*) for approximating wave arrival time

6.5 Isolated CMM Neuron

While the basic structure of the CMM is a matrix of binary neurons, we first consider the construction of an individual binary neuron and its connections with other neurons in the matrix. There already exist a number of physical implementations, including the use of a simple RAM device [8]. This use of a memory device to store the neural network was the starting point for the construction of the CMM presented here.

Motoike [26] showed how memory cells can be constructed in RD chemistry using a unidirectional ring of catalytic channels. By arranging the catalyst such that an external signal can excite the channels, or annihilate an existing excitatory wavefront, the memory cell can perform set/reset operations. Gorecki [17] proposed a method of separating the reset and read operations that were previously linked. An early design for this is given in Fig. 6.7, where the ring round the edge contains the state of the memory cell and the 'S' shape in the centre can be used as a reset signal.

A side-effect of this particular design of memory cell is that by taking an output from one point in the ring, it can be used to produce a periodic signal (as the wave travels round the ring, it will pass the output point at a rate proportional to the internal perimeter of the ring).

By storing the value of the CMM neuron in a memory cell in this way (i.e. if there is a connection in the matrix at that point, the memory cell is set to 1), then training a CMM is just a case of setting the corresponding memory cells.

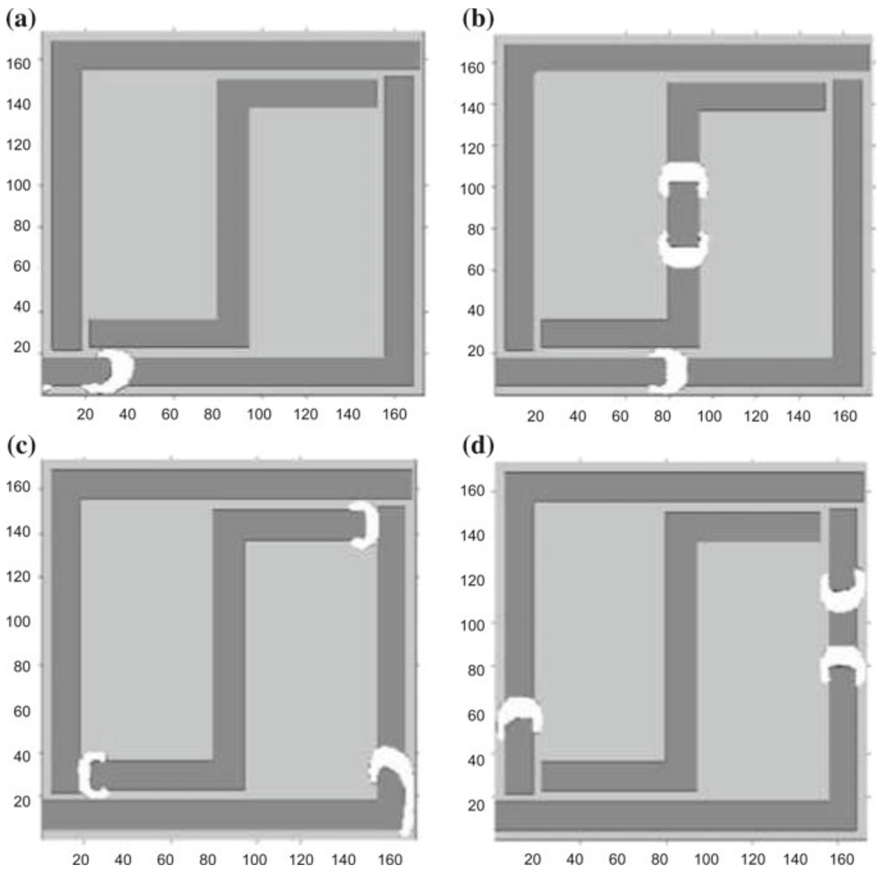


Fig. 6.7 Basic (simulated) memory cell. The wave propagates round the outer ring, in one direction only, but can be cleared by stimulating the S-shaped structure in the centre. From [17]

6.5.1 Requirements

The following requirements were identified as the basis for the CMM design:

Req. 1: The system must implement functionality of a CMM neuron.
Req. 1a: The neuron must maintain an internal state, $s_i \in \{0, 1\}$.
Req. 1b: The neuron must output $s_i \wedge x$ for some binary input $x \in \{0, 1\}$.
Req. 1c: The neuron must allow the state, s_i to be set to 1 (for training).
Req. 2: The neuron must allow the input (output) of one neuron to be passed to the next as input horizontally (output vertically).

6.5.2 Design

Figure 6.8 shows the initial logical design of the circuit. The circuit consists of a single memory cell, used to store the internal state (s_i) of the neuron. This fulfills Req. 1a. It is also used to provide a periodic pulse to coincidence detector (a), when trained to value 1. The second input to coincidence detector (a) is the input signal x, which corresponds to an individual binary element of the input vector $\mathcal{I}$. The output of detector (a) is then $s_i \wedge x$, and so fulfills Req. 1b. Finally, on coincidence of input signal x and training signal z (which corresponds to an individual binary element of the output vector $\mathcal{O}$) the second coincidence detector (b) sets the memory cell to 1, satisfying Req. 1c. From the fulfilling of all three sub-requirements, the system can then be said to fulfill Req. 1.

The cascading of inputs horizontally is achieved by splitting the x input value so that it feeds into both the coincidence detectors in the neuron and to the next neuron in the row. The z input in the design is representative of the vertical cascade functionality. This is used to pass the training signals through to each neuron and also to pass through the outputs from those neurons above this in the column to the bottom of the matrix. These points fulfill Req. 2.

In order to implement the vertical (train/output) cascading without interfering with the logic of the neuron, it has been moved from the centre to the side, leaving space for the memory cell to be compactly placed in the centre of the neuron (see Fig. 6.9).

During training, only an input–output pair of (1, 1) will load the memory cell, because coincidence detector (b), which is used for training, implements a logical-AND operation.

During recall, given an input of $x = 1$, if the memory cell is loaded (i.e. if there exists a link at this point in the matrix), then this cell will produce an output wave. This is then fed down to the output channel, marked z. This output channel feeds into the top of the cell below it in the matrix, as a means of propagating all the values from that column down to the thresholding logic that will be present at the bottom of the matrix. Because all the waves in any particular column will be produced

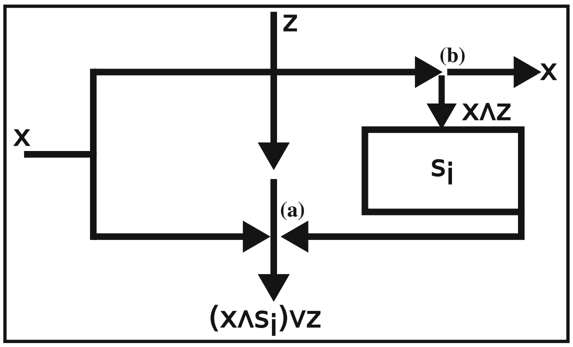


Fig. 6.8 Diagram showing the logical design of a single CMM neuron. The loop on the right implements a memory cell (with state $s_i \in \{0, 1\}$), the input (output) path is represented by x (z), and is used for both training and recall, and the original value of x (z) is passed through to the next neuron in the row (column)

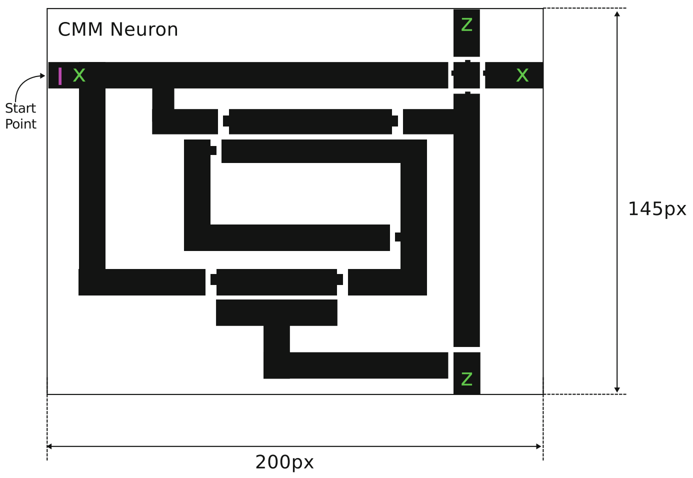


Fig. 6.9 Chemical layout for single CMM neuron. There are two coincidence detectors (AND-gates) in this design, the first is above the memory cell and is used during training ($x \wedge z$, detector (b) in Fig. 6.8). The second is below the memory cell and is used during recall ($x \wedge s_i$, detector (a) in Fig. 6.8)

simultaneously, the waves in the z-channel will not interfere with each other in any way. They also cannot interfere with the x-channels, as by the time an output wave is generated, the x-value will already have propagated to the next column.

Because of the way the neuron has been designed, we need to include a further requirement to ensure the design is scalable:

Req. 2a: The periodicity of the memory cell output must match the time to cascade horizontally.

This requirement means that the cascaded inputs will match with the periodic output of the memory cell across the entire row of neurons (i.e. the inputs to each neuron will arrive at the same point in the memory cell's periodic cycle).

6.5.3 Testing

Three tests were identified to ensure the proposed design was sufficient for use as a CMM neuron:

Test 1: Test periodicity of memory cell output is equal to time for cascading input horizontally.
Test 2: Test recall logic works for two neurons in each direction.
Test 3: Test training logic works for two neurons in each direction.

Test 1 requires precise measurements of the periodicity of the memory cell output and of the cascading input. To implement this test, two neurons were wired up horizontally, and a periodic boundary implemented on the far-side of the second neuron. This means that when the input propagates through the second neuron, it will reappear at the input of the first neuron. In doing so, the input becomes a periodic signal that can be compared to the signals from the memory cell output.

Test 2 can be implemented alongside Test 1, as a side effect of having a memory cell loaded and providing an input signal. Provided the first neuron produces output at every input signal, and the second produces no output, then the logic is correct.

Tests 1 and 2 were measured through pixel traces at locations $(98, 44)$ and $(98, 121)$ and time-lapse images.

Test 3 required training signals to be provided to the neurons from both the top and left. As this cannot be implemented alongside the other tests, it was set up separately. The test was looking for whether the coincidence of training signals will load the memory cell, and also tested that the output of said memory cell will coincide with the input from a recall signal. On top of this, the test needed to check for whether another training signal will 'overload' the memory cell, i.e. loading the cell with two waves instead of one.

Test 3 was measured using time-lapse images.

6.5.4 Results

Figure 6.10 shows the pixel traces for both memory cell and input cascade, for Test 1. The times that the waves pass the measurement points line up exactly for every other memory cell output—the semi-periodic nature of the x signal is due to there being two neurons wired up, so the signal takes twice as long to reach the measurement point.

The recall logic of the CMM neuron worked as expected, as can be seen from the time-lapse image produced towards the start of the run, in Fig. 6.11.

Finally, the training logic of the CMM also worked as expected, as can be seen from the time-lapse images in Fig. 6.12.

At this point, all three tests have been shown to be a success and the CMM neuron can now be considered as fulfilling its requirements.

6.6 CMM Thresholding

The next stage in the development of a complete CMM network is the ability to threshold the output. One of the main strengths of associative memory is the ability to generalise from a noisy or incomplete input pattern and returning the complete pattern. As described in Sect. 6.3.1, the output pattern, $\mathcal{O}_r$ should contribute the most

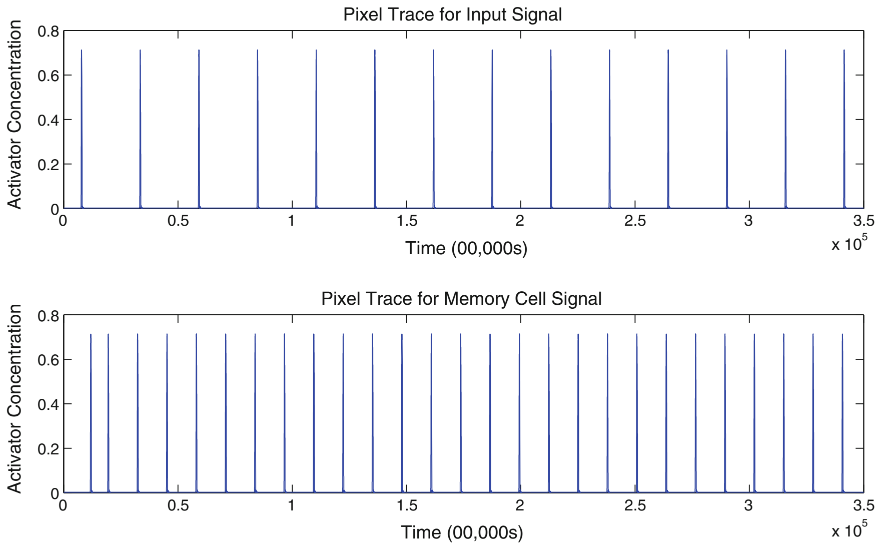


Fig. 6.10 *Top* graph shows semi-periodic input of x signal, *bottom* graph shows periodic output of memory cell. The spikes at the start of the *bottom* graph result from the initial stimulations

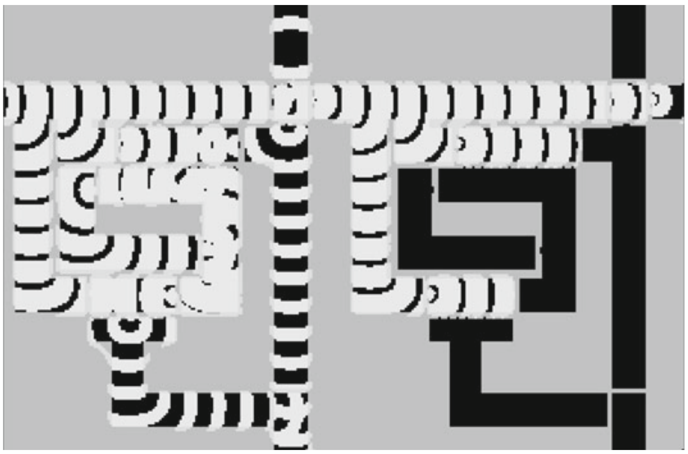


Fig. 6.11 Time-lapse image showing correctly implemented recall logic in CMM neuron. *Left* Neuron is trained to value 1, and upon recall outputs value 1. *Right* Neuron is trained to value 0, and upon recall outputs value 0

to the output of the network, with just a comparatively small noise term that needs to be cancelled out.

One method for achieving this is to input a series of waves (corresponding to the integer value of the threshold) up the output channels of the matrix. This will have the effect of annihilating that number of waves in each channel. Any waves remaining will propagate to the output, and should represent $\mathcal{O}_r$.

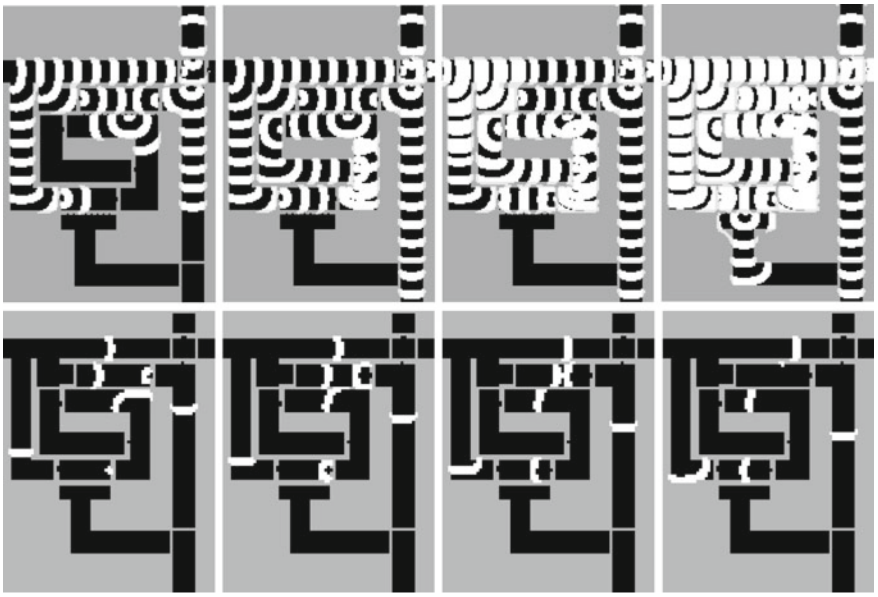


Fig. 6.12 Time-lapse images (*top*) showing CMM neuron training, and failure to 'overload' the neuron (where more than one value is stored in the memory cell at the same time (images are multiples of 10,000 simulated time steps). Images on the *bottom* show snapshots (at 500 simulated time step intervals) of the same behaviour, showing how the refractory period of the wave in the cell prevents a further wave from starting

6.6.1 Requirements

The following were identified as requirements for an appropriate thresholding circuit:

Req. 1: The system must generate the specified number (θ) of waves as output.
Req. 1a: The system must maintain an internal state representing the number of waves outputted.
Req. 2: The system must allow extraneous (i.e. $>\theta$) output waves from the CMM to pass through uninhibited.

6.6.2 Design

Figure 6.13 shows the basic logical design proposed to fill these requirements. The memory cell is used to produce the periodic output required from the circuit, while the binary counter keeps track of how many waves have been produced, fulfilling Req. 1a. Once the binary counter reaches threshold, it inhibits the production of further waves by clearing the memory cell, fulfilling Req. 1.

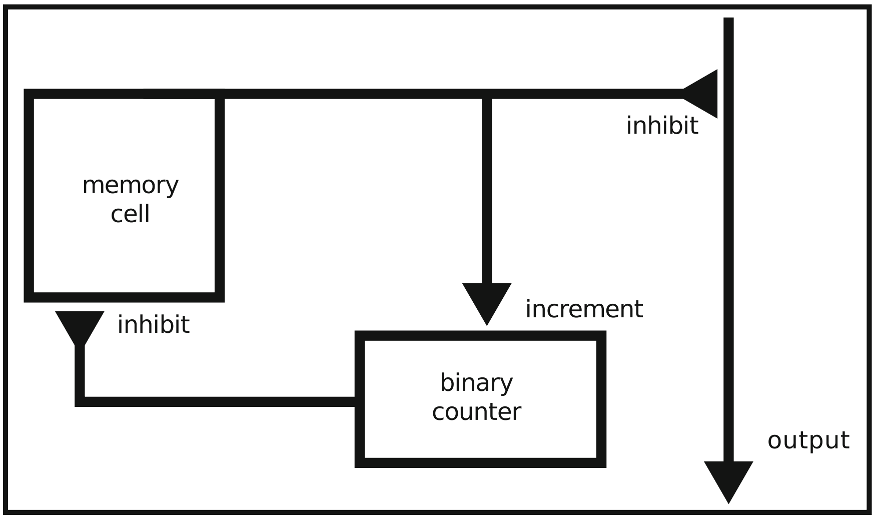


Fig. 6.13 Design for thresholding logic with threshold value θ. The circuit should produce θ waves, which are used to inhibit the output line. Once the counter reaches θ, it produces a single wave that inhibits the memory cell, stopping the output of waves

Figure 6.14 shows the chemical design (for threshold $\theta = 4$). By inserting a diode junction [6] on the output channel, the circuit fulfills Req. 2, as the thresholding waves will propagate up the output channel, until the appropriate threshold has been reached, at which point the output waves will propagate past the thresholding logic to the output.

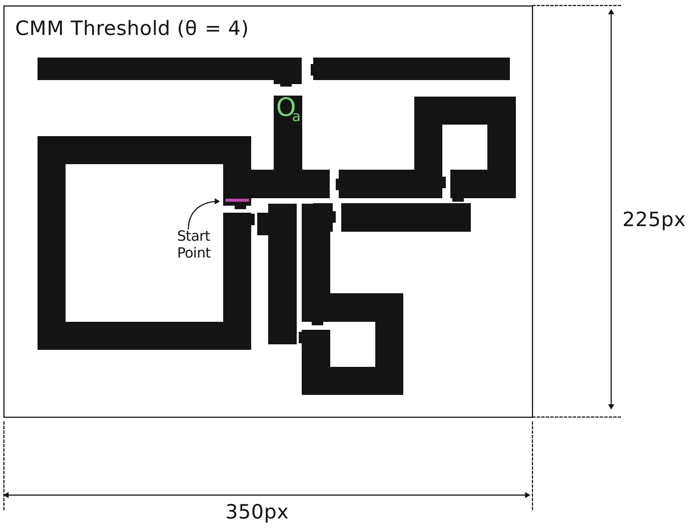


Fig. 6.14 Reaction layout for thresholding logic with threshold value $\theta = 4$. The two memory cells on the *right* and *bottom* are the binary counter, and the large memory cell on the *left* produces the output waves to inhibit the output line (across the *top*)

6.6.3 *Testing*

Test 1: Test number of waves outputted from thresholding logic matches threshold value, θ.
Test 2: Test number of waves outputted from CMM after thresholding matches number produced less threshold value.

Both tests can be performed simultaneously, by hooking up a series of five CMM neurons to a threshold circuit with $\theta = 4$. If both tests are successful, the circuit should output a single wave after thresholding. These tests can be measured using pixel traces, counting the number of waves produced on the output of the threshold logic, and then visually counting the number of waves produced on the post-threshold output channel.

Figure 6.15 shows the pixel trace for the threshold circuit output, clearly showing four waves at regular intervals, as expected. Figure 6.16 shows a time-lapse image with a single output wave on the output channel. At this stage, each test has been passed successfully, and the thresholding logic can now be considered as fulfilling its requirements.

6.7 CMM Training

The final checks to perform on a column of CMM neurons is to ensure that the training mechanism functions correctly. While Req. 1c in Sect. 6.5 showed that an individual neuron can be set to 1, this did not show how multiple neurons could be trained to a specific binary pattern.

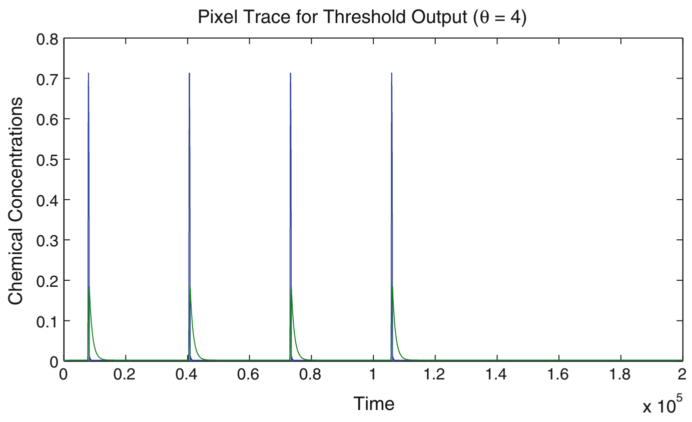


Fig. 6.15 Graph showing correct output from threshold circuit, with four periodic waves produced

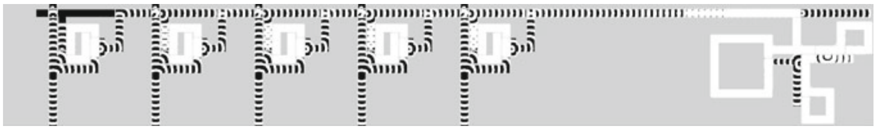


Fig. 6.16 Time-lapse image showing single output wave on output channel, after thresholding. *Note* The image has been rotated from the original definition of a CMM, so this represents a single column of neurons

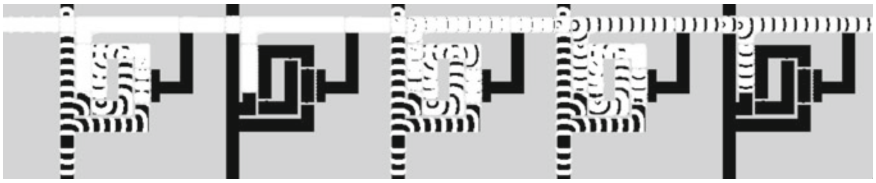


Fig. 6.17 A single column (rotated) of a correlation matrix, successfully trained to the binary value of 21 (10110)

The first test will be to check that a column of neurons is able to be trained. In order to achieve this, a column of five CMM neurons have been wired up, as in Fig. 6.16, but without the threshold circuit (for simplicity). These will be trained to the binary pattern that encodes 21 (10110).

Five waves are sent down the z-channel of the column, with a periodicity such that the waves arrive at the neurons at the same time as each other (one wave every 15,750 simulated timesteps). When all five had been sent, the x-channels of the column were stimulated with the binary pattern to be stored (at simulated time $t = 77{,}000$). For the training to be successful, the memory cells pertaining to the 1 s in the stored pattern will be set, and no others.

As can be seen from the time-lapse image in Fig. 6.17, the first, third and fourth memory cells (from the top of the column) have been set to 1, and no others. This shows that the training stage for a single pattern in a single column has been successful.

The next step in testing the single column is to store a second pattern in the pre-trained column of CMM neurons. At present, three of the neurons (1, 3, 4) have been set. By subsequently storing the binary pattern encoding 5 (00101), the column should only set the final (fifth) neuron in the column, so that the final neurons set are (1, 3, 4, 5). This pertains to the logical OR of the first and second patterns (10111). If any of the pre-set neurons are affected, then this test will have failed.

As can be seen from the time-lapse image in Fig. 6.18, the training has been successful and the appropriate neurons are set, as anticipated.

The next stage in the training testing is to ensure that multiple columns can be trained with different patterns. This will be achieved by connecting up two columns of CMM neurons, and training the first (left-most) column with the binary pattern encoding 25 (11001) and the second column with the binary pattern encoding 21 (10110, as before).

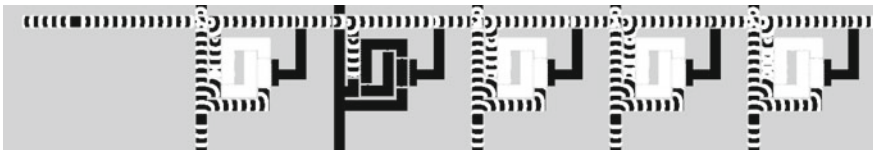


Fig. 6.18 A single column (rotated) with binary value of 21 (10110) already stored, is successfully trained with the binary value of 5 (00101), resulting in the superposed value 10111 stored in the column

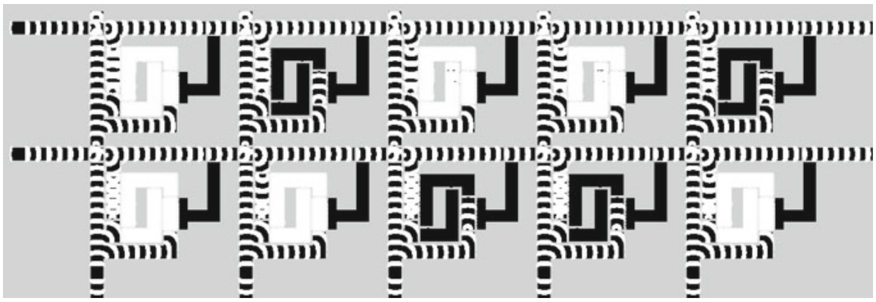


Fig. 6.19 Two columns (rotated) are successfully trained with distinct patterns, without interference or crosstalk between the patterns. First pattern (*bottom*) is the binary value of 25 (11001); second pattern (*top*) is the binary value of 21 (10110)

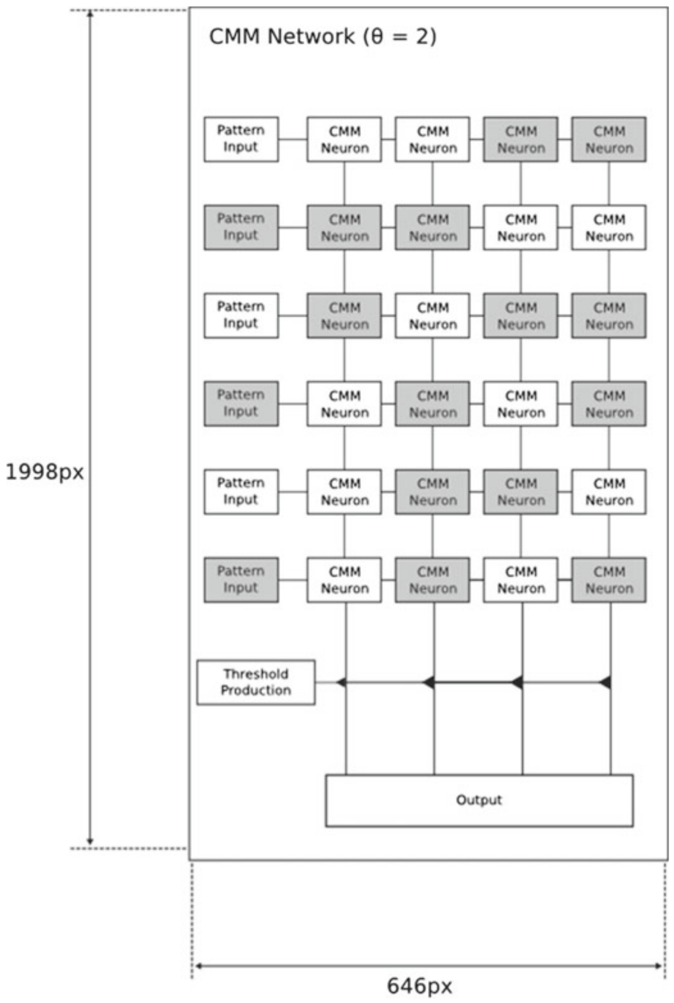
As can be seen from Fig. 6.19, the multi-column training was successful, with each column successfully trained without interference from the other column.

At this point, it is evident that the mechanisms provided for training a matrix of CMMs are sufficient. The next stage is to check the recall from a trained network of CMMs.

6.8 Full CMM Networks

Before we can be sure we have constructed associative memory, we need to ensure a network of CMM neurons will perform the computation we expect it to. The design decisions made when considering the individual neuron, and the testing that has already been performed on the thresholding and training aspects of the network, ensure that this is straightforward.

The testing in this section will consist of all the major stages required to construct a fully-functional CMM network. The network will be trained on a number of patterns. It will then be presented with a noisy version of one of the patterns, and the response thresholded to retrieve the answer. These tests will be performed on a 6×4 matrix of CMM neurons, with threshold $\theta = 2$. The patterns stored in the matrix will be



$$\begin{pmatrix}0\\1\\0\\1\\0\\1\end{pmatrix} \begin{pmatrix}0&0&1&1\\1&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\\0&1&0&1\end{pmatrix}$$

$$\frac{\begin{pmatrix}1&3&0&2\end{pmatrix}}{\begin{pmatrix}0&1&0&0\end{pmatrix}}$$

Fig. 6.20 *Left* Logical construction of a trained 6 × 4 CMM, with four patterns (011000, 010111, 101010 and 101101) stored, and input pattern 010101). *Right* Matrix representation of the same CMM, including expected results after thresholding at $\theta = 2$

the binary representations of 24, 23, 42 and 45. Upon presentation of the binary representation of 21, the network should respond with 23.

Figure 6.20 shows the logical (trained) construction of this network, where the grey and white boxes represent 1 and 0 respectively.

As is evident from Fig. 6.21, the training phase successfully stored the four patterns in the network.

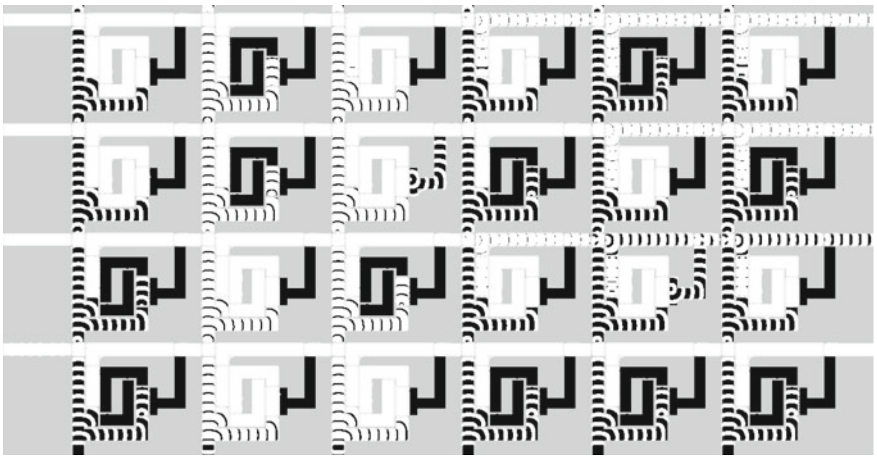


Fig. 6.21 Time-lapse image of the 6 × 4 CMM network (rotated) after training with the four patterns (011000, 010111, 101010 and 101101)

The final phase is to check the recall and generalisation of the network. The threshold circuit is started, and the input pattern (010101) presented to the network. As is evident from Fig. 6.22, the network responded with the second column, which corresponds to the number 23. From this, it can be seen that the circuit successfully implements a CMM-based associative memory.

6.9 Conclusions and Future Work

Reaction-diffusion chemistry and diffusive computation could offer a viable alternative to traditional approaches to computer science, but are generally very difficult to program. By providing a method of implementing different forms of neural network (spiking neurons previously [34]) and associative memory herein, we offer different methods of encoding problems in RD chemistry using paradigms that are more well-known.

The work presented implements a binary correlation matrix memory, and shows that the memory exhibits that same behaviour in RD chemistry as it would in traditional substrates. One of the main benefits of using binary CMMs is that training the network doesn't require altering the circuit in any way, just requires the setting of a series of memory cells. This is much simpler than training the form of spiking neuron we proposed previously [34], although a simpler spiking neuron implementation has been proposed [29] but requires a more implicit representation to that proposed by the current authors.

We suggest that diffusive computation, such as reaction-diffusion chemistry, has many further applications, and through the use of silicon-based diffusive proces-

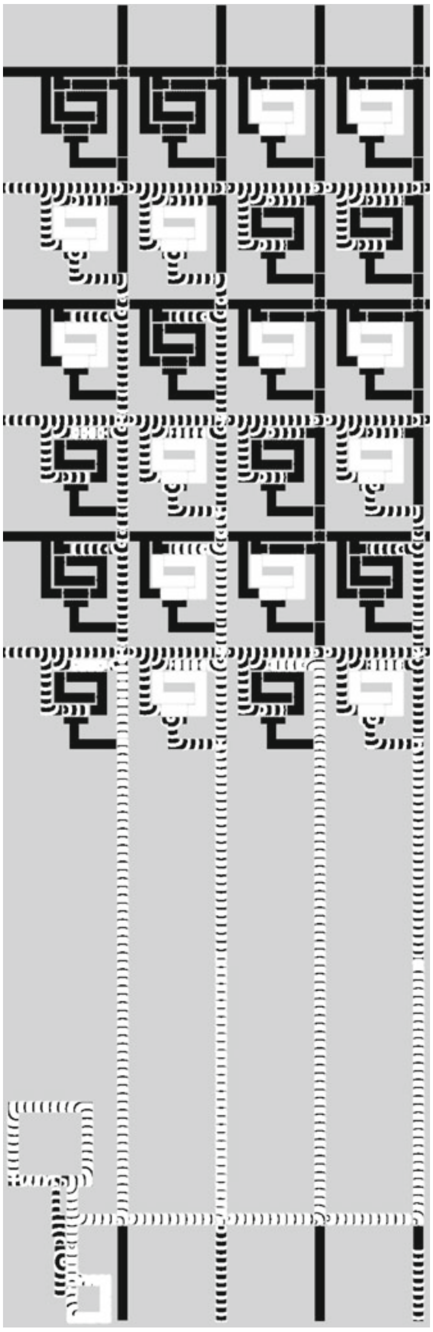


Fig. 6.22 Final time-lapse of the 6 × 4 CMM network. The time-lapse was cleared at simulated time 575000, to make the results clearer. The input pattern (010101) gives a single output after thresholding (second column) which pertains to the value 010111

sors [7] these systems could potentially be used alongside traditional computing systems, such that those problems that are amenable to solution by diffusive computation can be offloaded to the diffusive processor for efficient processing.

References

1. Adamatzky, A.: Computing in Nonlinear Media and Automata Collectives. IoP Publishing, Bristol, UK (2001)
2. Adamatzky, A.: If BZ medium did spanning trees these would be the same trees as Physarum built. Phys. Lett. A **373**(10), 952–956 (2009)
3. Adamatzky, A., De Lacy Costello, B., Asai, T.: Reaction-Diffusion Computers. Elsevier Science, Amsterdam (2005)
4. Adamatzky, A., Wuensche, A., De Lacy Costello, B.: Glider-based computing in reaction-diffusion hexagonal cellular automata. Chaos, Solitons Fractals **27**(2), 287–295 (2006)
5. Adleman, L.: Molecular computation of solutions to combinatorial problems. Science **266**(5187), 1021–1024 (1994)
6. Agladze, K., Aliev, R.R., Yamaguchi, T., Yoshikawa, K.: Chemical diode. J. Phys. Chem. **100**, 13,895–13,897 (1996)
7. Asai, T., De Lacy Costello, B., Adamatzky, A.: Silicon implementation of a chemical reaction-diffusion processor for computation of Voronoi diagram. Int. J. Bifurc. Chaos **15**(10), 3307–3320 (2005)
8. Austin, J., Stonham, T.J.: Distributed associative memory for use in scene analysis. Image Vis. Comput. **5**(4), 251–260 (1987)
9. Belousov, B.P.: A Periodic Reaction and Its Mechanism. Med Publ, Moscow (1959)
10. Carpenter, G.A.: Neural network models for pattern recognition and associative memory. Neural Netw. **2**(4), 243–257 (1989)
11. Conrad, M.: The brain-machine disanalogy. Biosystems **22**(3), 197–213 (1989)
12. Conrad, M., Zauner, K.: Conformation-based computing: A rationale and a recipe. In: Sienko, T., Adamatzky, A., Rambidi, N., Conrad, M. (eds.) Molecular Computing, chap 1. MIT Press, Massachusetts, pp. 1–31 (2003)
13. Dolnik, M., Marek, M., Epstein, I.R.: Resonances in periodically forced excitable systems. J. Phys. Chem. **96**(8), 3218–3224 (1992)
14. Field, R.J., Janz, R.D., Vanecek, D.J.: Composite double oscillation in a modified version of the Oregonator model of the Belousov-Zhabotinsky reaction. J. Chem. Phys. **73**(7), 3132–3138 (1980)
15. Gilreath, W.F., Laplante, P.A.: Historical review of OISC. In: Computer Architecture: A Minimalist Perspective. The Kluwer International Series in Engineering and Computer Science, vol. 730, pp. 51–54. Springer, US (2003)
16. Gorecki, J., Yoshikawa, K., Igarashi, Y.: On chemical reactors that can count. J. Phys. Chem. A **107**(10), 1664–1669 (2003)
17. Gorecki, J., Gorecka, J., Igarashi, Y.: Information processing with structured excitable medium. Nat. Comput. **8**, 473–492 (2009)
18. Haykin, S.: Neural Networks: A Comprehensive Foundation, 1st edn. Prentice Hall PTR, Upper Saddle River (1994)
19. Holley, J., Adamatzky, A., Bull, L., De Lacy Costello, B., Jahan, I.: Computational modalities of Belousov–Zhabotinsky encapsulated vesicles. ArXiv e-prints (2010)
20. Kohonen, T.: Correlation matrix memories. IEEE Trans. Comput. **C-21**(4), 353–359 (1972)
21. Kuhnert, L.: A new optical photochemical memory device in a light-sensitive chemical active medium. Nature **319**(6052), 393–394 (1986)
22. Kuhnert, L.: Photochemische manipulation von chemischen wellen (in German). Naturwissenschaften **73**, 96–97 (1986)

23. Kuhnert, L., Agladze, K.I., Krinsky, V.I.: Image processing using light-sensitive chemical waves. Nature **337**(6204), 244–247 (1989)
24. Laplante, J., Pemberton, M., Hjelmfelt, A., Ross, J.: Experiments on pattern recognition by chemical kinetics. J. Phys. Chem. **99**(25), 10,063–10,065 (1995)
25. Lázár, A., Noszticzius, Z., Farkas, H., Försterling, H.D.: Involutes: the geometry of chemical waves rotating in annular membranes. Chaos **5**(2), 443–447 (1995)
26. Motoike, I., Yoshikawa, K.: Information operations with an excitable field. Phys. Rev. E **59**, 5354–5360 (1999)
27. Müller, U.: Brainfuck—an eight-instruction Turing-complete programming language. http://www.muppetlabs.com/~breadbox/bf/ (1993)
28. Nakagaki, T., Yamada, H., Tóth, Á.: Path finding by tube morphogenesis in an amoeboid organism. Biophys. Chem. **92**(1–2), 47–52 (2001)
29. Rambidi, N.: Chemical-based computing and problems of high computational complexity: the reaction-diffusion paradigm. In: Sienko, T., Adamatzky, A., Rambidi, N., Conrad, M. (eds.) Molecular Computing, chap 4, pp. 91–152. MIT Press, Massachusetts (2003)
30. Rovinsky, A.B., Zhabotinsky, A.M.: Mechanism and mathematical model of the oscillating bromate-ferroin-bromomalonic acid reaction. J. Phys. Chem. **88**(25), 6081–6084 (1984)
31. Steinbock, O., Tóth, Á., Showalter, K.: Navigating complex labyrinths: optimal paths from chemical waves. Science **267**(5199), 868–871 (1995)
32. Stepney, S.: Unconventional computer programming. In: Symposium on Natural/Unconventional Computing and Its Philosophical Significance (2012)
33. Stepney, S., Abramsky, S., Adamatzky, A., Johnson, C., Timmis, J.: Grand challenge 7: journeys in non-classical computation. In: Visions of Computer Science, London, UK, September 2008, pp. 407–421 (2008)
34. Stovold, J., O'Keefe, S.: Simulating neurons in reaction-diffusion chemistry. In: Lones, M., Smith, S., Teichmann, S., Naef, F., Walker, J., Trefzer, M. (eds.) Information Processing in Cells and Tissues. Lecture Notes in Computer Science, vol. 7223, pp. 143–149. Springer, Berlin (2012)
35. Tolmachiev, D., Adamatzky, A.: Chemical processor for computation of Voronoi diagram. Adv. Mater. Opt. Electron. **6**(4), 191–196 (1996)
36. Tóth, Á., Showalter, K.: Logic gates in excitable media. J. Chem. Phys. **103**(6), 2058–2066 (1995)
37. Turing, A.M.: The chemical basis of morphogenesis. Philos. Trans. R. Soc. Lond. Ser. B, Biol. Sci. **237**(641), 37–72 (1952)
38. Willshaw, D.J., Buneman, O.P., Longuet-Higgins, H.C.: Non-holographic associative memory. Nature **222**(5197), 960–962 (1969)
39. Zhabotinsky, A., Zaikin, A.: Autowave processes in a distributed chemical system. J. Theor. Biol. **40**(1), 45–61 (1973)
40. Zhabotinsky, A.M.: Periodic course of the oxidation of malonic acid in a solution (studies on the kinetics of Belousov's reaction). Biofizika **9** (1964)
41. Zhabotinsky, A.M.: A history of chemical oscillations and waves. Chaos: Interdiscip. J. Nonlinear Sci. **1**(4), 379–386 (1991)