A Continuous Time Dynamical Turing Machine

Claire M. Postlethwaite, Peter Ashwin, Matthew Egbert

***Abstract*—Continuous time recurrent neural networks (CTRNN) are systems of coupled ordinary differential equations inspired by the structure of neural networks in the brain. CTRNN are known to be universal dynamical approximators: given a large enough system, the parameters of a CTRNN can be tuned to produce output that is arbitrarily close to that of any other dynamical system. However, in practise, both designing systems of CTRNN to have a certain output, and the reverse—understanding the dynamics of a given system of CTRNN—can be non-trivial. In this paper, we describe a method for embedding any specified Turing machine in its entirety into a CTRNN. As such, we describe in detail a continuous-time dynamical system that performs arbitrary discrete-state computations. We suggest that in acting as both a continuous-time dynamical system and as a computer, the study of such systems can help refine and advance the debate concerning the Computational Hypothesis that cognition is a form of computation and the Dynamical Hypothesis that cognitive systems are dynamical systems.**

***Index Terms*—CTRNN, Turing machine, network attractor**

I. Introduction

THE DYNAMICAL HYPOTHESIS in Cognitive Science proposes that "cognitive agents are dynamical systems" and as such, cognition is best understood in dynamical systems terms [1]. These claims developed as a rejection of the Computational Hypothesis that cognition is a form of computation [2], [3], i. e. a kind of symbolic manipulation whereby internal representations of the agent's world are transformed or processed by algorithms.

So which is it? Is cognition a kind of computation or is it a dynamical system? Or is this a false dichotomy? Connectionist neural networks present what some see as a middle ground. In these systems, cognition is a kind of computational processing of internal representations but the representations (symbols) are not pre-determined. They instead emerge in the dynamics of 'sub-symbolic' neural-network architectures [4]. Some see this as a fully-fledged alternative to classic computationalism, while others see the connectionist neural networks as just an implementation of computationalist ideas that do not give sufficient emphasis to the roles played by the body and environment, and the rich, dynamic interaction between these and the brain in the generation of intelligent behaviour (see e. g. [5], [6]). The debate is ongoing. Perhaps one reason for this is that despite efforts to be clear on what the real differences are between the dynamical and computational hypotheses (including those in the original presentation of the dynamical hypothesis [1]), central terms are interpreted in various ways. What counts as computation and what doesn't? What counts as a dynamical system and what doesn't?

Computation is often defined as either 'information processing' or the transformation of 'inputs' into 'outputs,' but just about anything can be construed as such. Similarly, a dynamical system is one that changes over time according to a fixed rule. What *doesn't* fit into this category? In these broad interpretations, a real-world computer *is* a dynamical system, and just about any dynamical system can be analyzed as a system that 'processes information.'

Despite these points and despite the fact that definitions of both computation and dynamical systems can seem all encompassing, the debate is not vapid, as the different metaphors suggest different ways for investigating cognition. If one thinks about cognition as a form of computation (e.g., [7], [8]) applied to solving problems, we find ourselves studying problems that are readily presented to a computational device, like chess and video games [9]. If, on the other hand, we consider cognition to be the result of dynamical interplay between coupled brain, body and world, then we find ourselves considering cognitive tasks that can be readily formalized as fitness functions, e. g. investigating categorical perception in a minimal embodied robot that can use whisker-like sensors to distinguish between circle and diamond shaped objects [10].

The different metaphors also suggest different methods for analyzing putatively cognitive systems. As a case in point, Beer et al. [11] demonstrate how dynamical and informational analysis can both be applied to the same artificial cognitive architecture. One would be forgiven for assuming that the information theoretic analysis is associated with the Computational Hypothesis, and that the dynamical systems analysis is similarly associated with the Dynamical Hypothesis, but as Beer et al. argue, the different analysis methods are, in a sense hypothesis-agnostic. Whether one embraces the computational or dynamical hypotheses (or neither), each of these analytic methods reveals different aspects of how the system that they are studying operates. Each of these aspects are informative and can be integrated into a more complete understanding of that system.

The benefits of the debate just described are epistemological—i. e. they relate to the ways that we study, describe and come to understand cognition. Is there more that the debate adds beyond these benefits? Are the definitions of both computational and dynamical systems so encompassing that they lose most of their meaning? Or are there real differences between A, the set of systems that are 'computational' and B, the set of systems that are 'dynamical'? If there are such differences, what precisely are they? To approach these questions, this paper presents a system designed to qualify as both a computational system and as dynamical system. The system we present is a Turing

Department of Mathematics, University of Auckland, Auckland, 1142, New Zealand. Email: c.postlethwaite@auckland.ac.nz

Department of Mathematics and Statistics, University of Exeter, Exeter EX4 4QF, UK. Email: P.Ashwin@exeter.ac.uk

Department of Computer Science, University of Auckland, Auckland, 1142, New Zealand. Email: m.egbert@auckland.ac.nz

machine [12] realized in its entirety in the dynamics of a continuous time recurrent neural network (CTRNN) [13] (using a system similar to that investigated by many others since Hopfield [14]). The system we present is also clearly a dynamical system as it is defined by a state space and a set of ordinary differential equations (ODEs) that describe how the state of the system changes over time. The only parameters in the system are a weight matrix of interconnections and a common threshold nonlinearity.

In qualifying as both a computational system and as a dynamical system, we propose that the system described in this paper can be useful as an examplar for highlighting ways that computational systems are 'unusual' or different from most other dynamical systems, hopefully clarifying positions around the computationalist/anti-computationalist debate. After giving an overview of the necessary background on Turing machines and network attractors in section II, we present the model in detail in sections III and IV. Then, in Section V, we show how our model can be used to consider the relationship between computation and dynamics by highlighting three salient features of the designed computational dynamical system: topological regularity; dynamical sparcity, and temporal regularity. We briefly discuss how these salient properties may be the result of design requirements (i. e. that the system must act as a computational device) *or* the result of by-human engineering and the requirements implicit in that process.

II. Background

The dynamical system we design in this paper has the form of a *network attractor*, designed within a continuous time recurrent neural network (CTRNN). In this background section, we give a brief overview of previous work related to ours, then a review on how Turing machines function, and finally we give some background on network attractors in CTRNNs. We chose CTRNN as a dynamical medium as they have been widely used in investigations into how cognitive processes (e.g., associative learning [15], categorical perception [10], and learning [16]) can be understood not as computation, but as the interaction between brain, body and world dynamics [6], [17].

A. Related work

Turing machines are widely accepted as the archetype of discrete computation, and so it is fair to suggest that at the base of the computational hypothesis is the view that something essential is shared by cognitive systems and finite state automata such as Turing machines. It has been known since [18], [19] that simple networks are capable of realizing finite automata and hence Turing-type computation in certain senses. This line has been investigated by various authors, for example [20], [21], [22], [23]. Universal computation can be found in universal memcomputing machines [24], Neural P-systems [25] and reservoir computers [26] to name a few approaches. Recently, [27] present an algorithm to embed an arbitrary Turing maching by means of a modular structure of synfire chains. We note that some investigations use the analogue nature of RNNs to perform computation that goes beyond Turing in the sense of encoding a continuum of states [28], [29]; however we restrict ourselves to implementation of finite state computation.

Individual neurons are diverse. Their dynamics are complex and varied, leading to a variety of proposals as to how neurons could might accomplish different types of computation with different dynamic media. Maass [30] compares the computational capabilities of spiking neurons to threshold-gate neurons (McCulloch–Pitts) and sigmoid-gate neurons. Spiking neurons are more complicated and more computationally powerful than minimalistic McCulloch–Pitts neurons, and Maass investigates how different computations can be accomplished with the 'temporal patterns' inherent in spiking neural dynamics. A general point here is that some dynamic media (i.e. some basic neural units) are more naturally suited to different kinds of tasks than others. For example, spiking neural networks are well suited to detecting event synchrony [30].

At a higher topological scale, a 'cell assembly' (CA) is a group of neurons that tend to become excited in concert, either in response to a particular environmental stimulus or due to influences of other cell assemblies. Excitatory connections within a CA can maintain its activity after the stimulus has ended. Neuroscientific evidence suggests that CA exist in various organisms [31] and links between CA and fundamentals of cognition (e.g., categorisation, short-term memory and long-term memory) have been proposed [21]. One of the most well known CA models is the Hopfield network [14]. A CA built from fatiguing leaky integrate-and-fire neurons (fLIF) can be used to implement any finite state automaton (FSA) [32]. Similar constructions of FSA have been achieved in networks of Hebbian cell assemblies (see, e.g., [23] and references therein). This presents an algorithm for transforming any finite state automaton into a corresponding Hodgkin—Huxley neural network of synfire rings, proposing that synfire rings could act as "a theoretical ground for the realization of biological neural computers" [23] and proposes an alternative 'Assembly Calculus', i.e., a set of biologically plausible operations on CA that could realize a complete computational system.

B. Turing machines

Our construction is adapted and simplified from a previous paper [33], and follows [34]. A finite-state single-tape Turing computing machine consists of

- a finite set of *internal states* Q,
- a finite set Γ of *tape symbols*,
- a *starting state* $q_1 \in Q$,
- at least one, and possibly several, *halting states* $F \subset Q$,
- a *transition function* $\rho : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

Let $n_Q := |Q|$, the number of internal states, and $n_\Gamma := |\Gamma|$, the number of possible symbols. We number the states and symbols $q_i \in Q$, $i = 1, \ldots, n_Q$ and $s_j \in \Gamma$, $j = 1, \ldots, n_\Gamma$, so we can represent the action of the transition function ρ as

$$\rho(q_i, s_j) =: (\tilde{q}_{ij}, \tilde{s}_{ij}, \tilde{\sigma}_{ij})$$

where $\tilde{q}_{ij} \in Q$, $\tilde{s}_{ij} \in \Gamma$ and $\tilde{\sigma}_{ij} \in \{L, R\}$.

The *tape* consists of a discrete string of symbols $\gamma_j \in \Gamma$ and Turing machines usually consider an infinite tape $j \in \mathbb{Z}$. We

say there is a *finite loop tape* if γ_j is defined for j modulo n_t for some $n_t \in \mathbb{N}$ corresponding to the tape length. We note that when using a finite, rather than infinite tape, a Turing machine is not technically Turing complete, but is only equivalent to a finite state automata. However, in systems where we can make the tape arbitrarily long, this is often referred to as Turing complete, even though proper simulation would require an infinite tape. In this paper, we consider Turing machines with only two symbols, which we label 0 (or blank) and 1. We consider the 0 to be a 'blank', so in fact we are only considering a single letter alphabet $A = \{1\}$. However, it would also be straightforward to adapt our design technique to allow for a larger number of symbols, and we describe how to do this in section III-B.

Starting at tape position $j = 0$ and internal state $q(0) = q_1$ suppose that the machine arrives after n steps at internal state $q(n) \in Q$ and tape position $\beta(n)$. If $q(n) \in F$ then the computation has finished, while otherwise it reads the transition function

$$(\tilde{q}, \tilde{\gamma}, \tilde{\sigma}) = \rho(q(n), \gamma_{\beta(n)})$$

where $q \in Q$, $\tilde{\gamma} \in \Gamma$, $\tilde{\sigma} \in \{L, R\}$. The machine then updates the internal state to $q(n+1) = \tilde{q}$, the symbol at site $\beta(n)$ is changed to $\gamma_{\beta(n)} = \tilde{\gamma}$ and then the machine moves along the tape according to

$$\beta(n+1) = \beta(n) + 1 \quad \text{if} \quad \tilde{\sigma} = R$$
$$\beta(n+1) = \beta(n) - 1 \quad \text{if} \quad \tilde{\sigma} = L.$$

The machine repeats this either forever, or until it reaches a halting state and computation has finished. Clearly there will be restrictions placed by a finite loop tape that will depend on choice of n_t, but for any given computation that comes to completion, only finitely many tape positions will be used by the time the halting state is reached.

C. Excitable network attractors

A previous paper [33] demonstrated that it is possible to realise a Turing machine in the dynamics of excitable or heteroclinic networks, but with a heterogeneous of network of nonlinear units that were designed to have 'robust connections'. In [35], we show how to embed an 'excitable network attractor' of arbitrary structure within the dynamics of a CTRNN. Note that 'heteroclinic attractors' have been used to construct computational systems (see for example, [35], [36]), but excitable network attractors are in some sense more robust in that they link attracting states. In this section, we give an overview of excitable network attractors, and a brief description of how the embedding works.

In the most general terms, an excitable network attractor is an invariant set within the phase space of a dynamical system, consisting of a number of attractors (states) that may be selectively sensitive to perturbations in certain directions. Such perturbations allow the trajectory to move between the states, but only along pre-specified connections: in this way the network attractor realizes a directed graph, and the trajectory in the phase space represents the directed path taken around this directed graph.

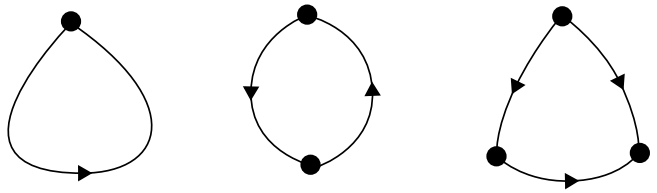

Fig. 1. The figure shows, from left to right, a one-loop, a two-loop, and a Δ-clique in a directed graph; sub-networks which are not allowed in the topology of the directed graph to be embedded in a CTRNN.

CTRNNs are sets of differential equations describing the dynamics of a set of *nodes*; each dependent variable in the differential equation gives the excitation level of a single node. Nodes are thought of as *excited* if the value of a nonlinear activation function $\phi(.)$ of that node is order 1. With the choice of weight matrix we describe in [35], excitation of nodes is mutually exclusive: only a single node can have $\phi(.)$ order 1 at any given time. This is in contrast to the dynamics of a typical CTRNN (e.g. one with either a weight matrix that is either a randomly assigned, or optimised for some purpose using, for instance, a genetic algorithm), which would not have such a restriction.

In [35] we showed that it is possible to choose the weight matrix of an n-dimensional CTRNN so that, in addition to the above restriction, the attracting dynamics can be described in terms of a path around a directed graph with almost any desired topology (restricted only so that the graph does not include any one-loops, two-loops or Δ-cliques; see figure 1). Each node in the CTRNN is associated with a vertex in the directed graph: when node y_j is excited, we say that a path around the directed graph is *at* vertex j.

More specifically, for our construction, there exist stable equilibria in the phase space, at each of which a single node of the CTRNN is excited. When a trajectory is close to such an equilibria, we refer to the excited node as *active*, and to the remaining nodes as either *leading*, *disconnected* and *trailing*, depending on the connectivity to other nodes within the directed graph. Leading nodes have arrow heads pointing from the active node to them, i.e. by following the arrows in the directed graph, it is possible for them to become the next active node. Trailing nodes have arrow heads pointed from themselves to the active node. Disconnected nodes have no arrows between them and the active node. The top panel of figure 2 shows a schematic of part of a directed graph. When the red node is active, the nodes labelled LA and LD are leading, the node labelled TD is trailing, and the nodes labelled DL and DT are disconnected.

In [35], it is shown that a parameter in the weight matrix can be varied through a bifurcation such that the connections between the different nodes can be in one of two different dynamical regimes, which we call *excitable* and *automatic*. When connections are *excitable*, perturbations, either added at specified times, or from additive noise, are required to push trajectories from one state to another. Parameters can be chosen to adjust how large these perturbations need to be to affect a transition. Alternatively, it is also possible to choose

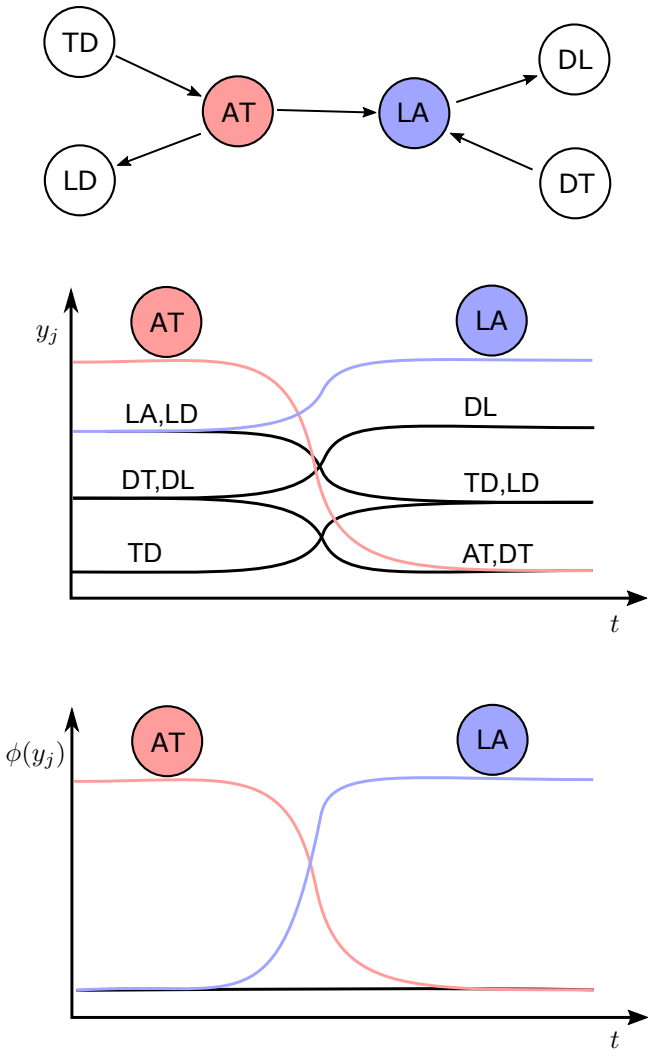

Fig. 2. The figure shows (top) a schematic of part of a directed graph, and below, schematic showing timeseries of the values of y_j and $\phi(y_j)$ for a trajectory as it transitions from a state where the red node is active to one where the blue node is active. The nodes of the directed graph are labelled with two letters corresponding to whether they are (A)ctive, (L)eading, (T)railing or (D)isconnected, before and after the transition.

the perturbations so that the connections between the states is what we call *automatic*. In this case the transition between states happens without any external inputs. Here, the states are no longer stable equilibria, but are regions of phase space close to the *ghost* of a saddle-node bifurcation [37] (also known as a 'funnel region'). The trajectory remains in the region of the ghost for a relatively long period of time, so it appears to be at equilibrium, before rapidly switching to the next state.

In [35] we show that parameters can be chosen so that connections can be a mixture of excitable and automatic, as desired by the designer of the network. The weight matrix depends only on the topology of the directed graph, and four parameters, as described in section III. We refer to the entire network in phase space as a *CTRNN excitable network*.

III. Realising a Dynamical Turing Machine

The full system comprises multiple CTRNN excitable networks, coupled together with small perturbations (or inputs). The perturbations are designed in such a way that in the Turing machine realisation is an input-free CTRNN that we call a *Dynamical Turing Machine*.

The sub-networks of this are:

1) A *state network* which describes the internal state of the Turing machine.
2) A *tape network* which describes the position of the head of the tape, and the symbol at each position of the tape. This network actually consists of n_t disjoint sub-networks, where n_t is the length of the tape, and there is one sub-network for each position on the tape.

Each of the state network and the tape sub-networks is, in the absence of external perturbations, a CTRNN excitable network, and so only a single node within each of these sub-networks can be active at any one time. Which of the nodes is active in the state network describes the internal state of the Turing machine, and which of the nodes is active in the tape sub-networks describes both the symbols on the tape, and also the position of the tapehead.

The governing equations for an infinite tape are of the form

$$\dot{\mathbf{y}} = f_y(\mathbf{y}) + \zeta_y g_y(\phi(\mathbf{x})), \tag{1a}$$

$$\dot{\mathbf{x}}_j = f_x(\mathbf{x}) + \zeta_x g_x(\phi(\mathbf{y})) + \zeta_\beta(g_R(\phi(\mathbf{x}_{j-1})) + g_L(\phi(\mathbf{x}_{j+1}))), \quad j \in \mathbb{Z} \tag{1b}$$

where the vector $\mathbf{y}$ describes the variables in the state network, and $\mathbf{x} = (\dots, \mathbf{x}_{-1}, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots)$ describes the variables in the tape network, with each vector $\mathbf{x}_j$ representing the variables in the sub-network on the jth position of the tape. For a finite (periodic) tape, the subscript on each $\mathbf{x}_j$ is taken modulo n_t, and so $\mathbf{x}$ is finite. The parameters ζ_x, ζ_y and ζ_β are small constants and control the size of the perturbations coupling the sub-networks. Note that each of the tape sub-networks only receives inputs from the state network and its immediate neighbours in the tape. Note also that these perturbations are small enough that the structure of the dynamics within each sub-network remains that of a CTRNN excitable network (as described in section II): the perturbations act *only* as the perturbations which move the trajectory along the excitable connections.

The functions f_x and f_y have the form of a CTRNN, so, for example,

$$f_y(\mathbf{y}) = -\mathbf{y} + W_y\phi(\mathbf{y})$$

where the function ϕ, given by

$$\phi(x) = \frac{1}{1 + \mathrm{e}^{-K(x-\theta)}}$$

for parameters K and θ, is applied component-wise to the vector $\mathbf{y}$, and W_y is a matrix, the entries of which depend on the topology of the state or tape network, as will be described in the forthcoming sections. Each of the g_α functions ($\alpha = y, x, R, L$) are linear in their arguments, and provide inputs, or perturbations between the state network and each of the tape

Inputs		Outputs		
q_i	s_j	$\tilde{q}_{ij}$	$\tilde{s}_{ij}$	σ_{ij}
q_1	0	H	N	0
q_1	1	q_2	0	R
q_2	0	q_3	0	R
q_2	1	q_2	1	R
q_3	0	q_4	1	L
q_3	1	q_3	1	R
q_4	0	q_5	0	L
q_4	1	q_4	1	L
q_5	0	q_1	1	R
q_5	1	q_5	1	L

TABLE I
TRANSITION FUNCTION ρ FOR THE COPY ROUTINE.

sub-networks. Thus, the entire system (the Dynamical Turing Machine) has governing equations of the following form:

$$\dot{\mathbf{Y}} = -\mathbf{Y} + W\phi(\mathbf{Y}) \tag{2}$$

where we refer to W as the *weight matrix* and $\phi(\mathbf{V})$ is applied component-wise.

In the following sections, we first describe the construction of the functions f_x and f_y, which give the internal dynamics of the state network and tape sub-networks. These are chosen so each is a CTRNN excitable network with the topology of a specified directed graph. Each has some automatic transitions, which mean the trajectory transitions between the nodes in the network without input, and some excitable transitions. When the trajectory is near a node which only has excitable transitions leaving it, an input is required to prevent the system remaining at equilibrium; these inputs come from the other networks. In section III-C we describe the functions g_α which provide these small inputs from one network to another.

In summary, a trajectory with appropriate initial conditions will visit a sequence of regions of phase space which are near equilibria and can be associated with different internal states of the Turing machine and different configurations of the tape. This will continue until the Turing machine reaches a halting state.

A. State network

The first step in the construction is to express the internal states Q and transition function ρ of the Turing machine as a directed graph. Throughout this section, we use the 'copy' routine [38] as an example, but the same process can be used for other Turing machines. The transition function for the copy routine is shown in table I, and the associated directed graph in figure 3. The vertices in the graph represent the internal states $Q = \{1, 2, 3, 4, 5, H\}$ (where $H \in Q$ is the halting state), and the connecting arrows are labelled with a 0/1 if they occur when the tapehead reads a 0 or 1 respectively. The red letters R and L indicate if the transition between states moves the tapehead right or left, and the red 0 or 1 on the arrows indicate if the tapehead writes a 0 or 1 to the tape.

The *state network* is constructed from this directed graph. In order to generate the required perturbations to the tape network when transitions are made in the state network, we add an additional two vertices along each edge in the original graph. Recall that each vertex in the directed graph corresponds to a

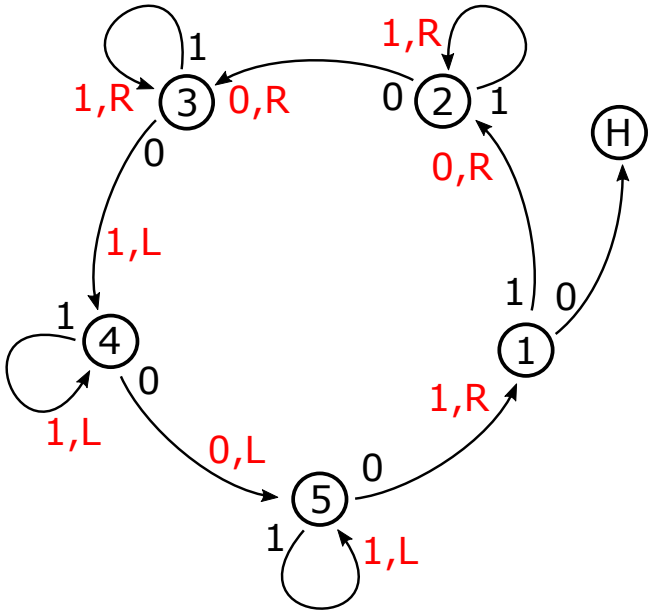

Fig. 3. The transition function ρ (table I) for the copy routine represented as a directed graph. The states are represented by the circled numbers. The black 0s and 1s on the arrows indicate the current state on the tape for which that arrow should be followed, and the red 1s, 0s, Rs and Ls indicate the symbol which should be written on the tape and the direction the tape head should move when that transition occurs.

node (or variable) in the CTRNN, and so from hereon we use the word 'node' to also describe the vertices in this directed graph. Of each pair of additional nodes, the first one (coloured blue in figure 4) provides an input into the tape network. In what follows, we refer to these nodes as the *input nodes*. The second additional node (coloured gray) does nothing, except allow for a time delay before the state network reaches the next state. We refer to these as the *delay nodes*. We refer to the original nodes (in this case, those labelled 1-6 and outlined in black) as the *internal nodes*. The addition of these nodes also has the additional effect of ensuring there are no one-loops, two-loops or Δ-cliques in the state network.

In figure 4 we show this extended graph for the copy routine. The internal dynamics of the state network are then determined using the method described in [35] (and in section II) to embed this directed graph as a CTRNN excitable network. The connections leaving both the input and delay nodes are automatic (and are coloured green in figure 4). The connections leaving the internal nodes in the state network (coloured black and dotted in figure 4) are excitable. In the absence of any inputs or perturbations, trajectories would remain close to these nodes, but perturbations from the tape network will push the trajectory away from these. Specifically, inputs from the tape network, when the tape reads a zero or a one, as indicated by the labels on the arrows, push the trajectory from an internal node to the appropriate input node. The automatic transitions then allow the trajectory to move to the next internal node without further input. While the trajectory is close to the input node, the state network in turn provides inputs to the tape network so that the symbols and position of the tape head are adjusted appropriately.

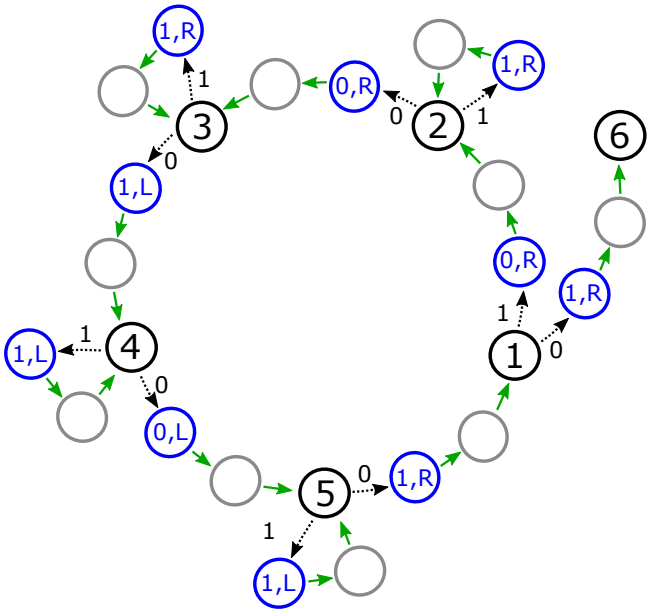

Fig. 4. The extended directed graph for the copy routine. The *input nodes* are coloured blue, and the *delay nodes* are coloured gray. Further details are in the text. Automatic transitions are indicated by green arrows, and excitable connections by black dotted arrows. Note that we have added the instruction $(1, R)$ between state 1 and state 6 (the relabelled halting state) for consistency; this does not change the outcome of the Turing machine.

We write the adjacency matrix A of this extended graph in two parts, one which enumerates the excitable connections, and one which enumerates the automatic connections:

$$A = A_a + A_e$$

The matrix W_y has elements $(W_y)_{ij}$, which are given by

$$\begin{aligned}(W_y)_{ij} = w_t + (w_s - w_t)\delta_{ij} + (w_a - w_t)(A_a)_{ij} + \\ (w_e - w_t)((A_e)_{ij}) + (w_m - w_t)((A_a)_{ij} + (A_e)_{ij})\end{aligned} \tag{3}$$

where δ_{ij} is the Kronecker δ. This choice ensures that $(W_y)_{ii} = w_m$, $(W_y)_{ij} = w_a$ if there is an automatic connection from node j to node i, $(W_y)_{ij} = w_e$ if there is an excitable connection from node j to node i, and $(W_y)_{ij} = w_t$ otherwise. The parameters w_t, w_e, w_a and w_m are chosen as described in [35], to ensure the existence of an excitable network. The numerical values we use are given in section IV.

In order to generate the appropriate inputs between the state network and the tape network, we write down a number of vectors which essentially enumerate the different types of nodes in the networks. The required vectors for the state network are as follows.

- Four vectors v_{0R}, v_{0L}, v_{1R} and v_{1L}, which enumerate which nodes correspond to transitions that move the tape right/left and write 0/1 on the tape. These vectors are zeros except in the positions of the *input nodes* that, for v_{1R} say, write a one and move the tape right. For the copy routine, for instance, the vector v_{1R} would have ones in the positions corresponding to the blue nodes labelled $1, R$ in figure 4.
- Two vectors v_0 and v_1 which give the positions of the nodes which should be perturbed when the tape head reads a zero or a one. Each are zeros except for the positions of the *input nodes* along the transitions which occur when there is a 0 or 1 on the tape. For instance, v_0 for the copy routine would have ones in the positions corresponding to nodes at the heads of black arrows labelled with a 1.

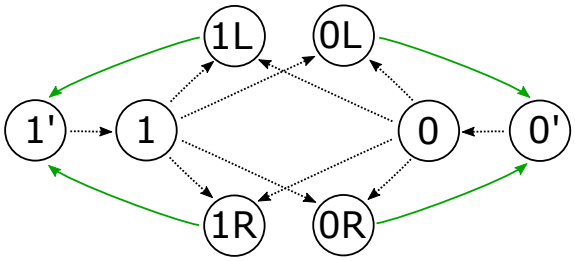

Fig. 5. Schematic of one sub-network of the tape and symbol network. Green arrows correspond to automatic connections, black dotted arrows correspond to excitable connections.

B. Tape network

The symbols on the tape and the position of the tape head are enumerated and recorded using a set of sub-networks, one of which is shown in figure 5. We assume a finite loop tape with tape length n_t, so that the tape network consists of n_t copies of the network shown in figure 5. Again, the internal dynamics of each tape sub-network is such that there exists a CTRNN excitable network with this topology.

When a 0 or 1 node is active in the sub-network in the kth position of the tape ('sub-network k') this corresponds to the symbol at the kth position on the tape being a 0 or a 1, respectively, and that the tapehead is at position k. When a $0'$ or $1'$ node is active in sub-network k, this corresponds to the symbol at the kth position on the tape being a 0 or a 1, respectively, and that the tapehead is *not* at position k. As will be seen in what follows, so long as initial conditions are chosen appropriately, only one sub-network can have an 'unprimed' 0 or 1 node active at any given time: these nodes are only excited when a neighbouring tape sub-network changes from an 'unprimed' 0 or 1 to a 'primed' 0 or 1 node.

As can be seen in figure 5, each sub-network of the tape and symbol network has two parts consisting of four nodes, one with '0' labels and one with '1' labels. To extend the system to include more symbols, we simply add additional four-node units to each tape sub-network, one for each additional symbol. Additional excitable connections are added from each symbols 'unprimed' node to every other symbols 'L' and 'R' nodes.

The adjacency matrix of each tape sub-network is again divided into automatic and excitable parts. The connections between the nodes are shown in figure 5, where green arrows correspond to automatic connections and black dotted arrows correspond to excitable connections. Let the adjacency matrix of the tape sub-network be the matrix B, then we write

$$B = B_a + B_e.$$

The matrix W_x which generates the internal dynamics of each tape sub-network has elements $(W_x)_{ij}$, with

$$(W_x)_{ij} = w_t + (w_s - w_t)\delta_{ij} + (w_a - w_t)(B_a)_{ij} + (w_e - w_t)((B_e)_{ij}) + (w_m - w_t)((B_a)_{ij} + (B_e)_{ij}) \quad (4)$$

Again, in order to generate the appropriate perturbations between the sub-networks, we write down a number of vectors which enumerate certain nodes in the networks. Those vectors for the tape sub-networks are as follows. All these vectors are 8-dimensional, consisting only of zeros or ones.

- Vectors w_0 and w_1 which indicate when a tape sub-network has either of the nodes 0 or 1 active, that is, these vectors are zeros everywhere except in the positions corresponding to the 0 nodes (for w_0) and 1 node (for w_1) in the tape sub-network. These vectors have the affect of 'reading the tape at the position of the tapehead'.
- Four vectors w_{0R}, w_{1R}, w_{0L} and w_{1L}, which are zeros except in the position corresponding to the 0R, 1R, 0L and 1L nodes in each tape sub-network. These are used so that perturbations from the state network push the tape network to the correct node.
- $w_L = w_{0L} + w_{1L}$ and $w_R = w_{0R} + w_{1R}$, used to indicate whether the tape should be moved right or left, and send perturbations from one tape sub-network to the appropriate neighbour.
- $w_{01} = w_0 + w_1$, which indicates the nodes in tape sub-network j which should be perturbed as the tapehead moves to position j.

C. Coupling the networks

There are three classes of coupling between the networks:

1) Inputs from a tape sub-network to the state network, when that tape sub-network has either of the nodes 0 or 1 active. This corresponds to the tapehead reading a 0 or 1, and the input pushes the trajectory in the state network along the appropriate connection.
2) Inputs from the state network to the tape sub-networks when the state network has an active *input node*. This input both (over)writes symbols on the tape and moves the tapehead.
3) Inputs from a tape sub-network to its immediately neighbouring tape sub-networks, to affect the movement of the tapehead.

All three of these coupling types are shown by dashed arrows in figure 6. More details and specific equations for each of these input types are as follows:

1) When a tape sub-network has one of the nodes 0 or 1 active, this corresponds to the tapehead being at that position on the tape. The tape network provides an input into all of the input nodes in the state network which are at the end of an excitable connection labelled with a 0 or 1, respectively. This input to the state network is given by the expression

$$g_y(\mathbf{x}) = \left(\sum_{j=1}^{n_t} \phi(\mathbf{x}_j) \circ w_0\right) v_0 + \left(\sum_{j=1}^{n_t} \phi(\mathbf{x}_j) \circ w_1\right) v_1$$

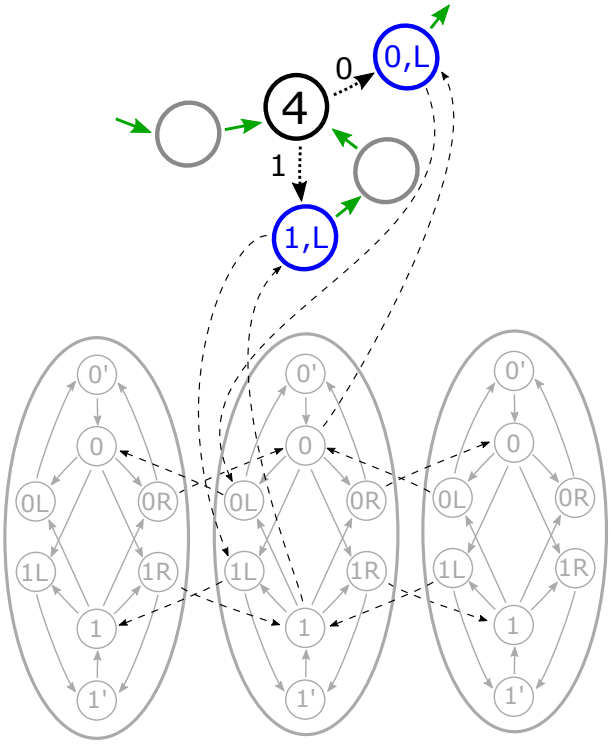

Fig. 6. The figure shows a portion of the state network from figure 4 (top), and three sub-networks of the tape network (bottom). The black dashed lines represent a subset of the perturbations which occur between the sub-networks. Each arrow indicates that when the node at the tail of the arrow is active, a perturbation is applied to the node at the head of the arrow. If that node is a leading node, then it will become active. All such arrows which are either incoming or leaving the middle of the three tape sub-networks are shown. Note that these dashed arrows are *not* indicative of a directed graph forming a CTRNN excitable network.

where $\circ$ is the Hadamard (entry-wise) product. The first term corresponds to the case where a tape sub-network is at 0, the second term to when it is at 1. Since $\phi(x)$ is close to zero unless the node corresponding to variable x is active, and w_0 has the effect of selecting the coordinating corresponding to the 0 node, the first term in brackets is only $O(1)$ when one of the tape sub-networks has a 0 node active. The vector v_0 provides the required coordinates for the input into the state network. The second term is equivalent for the 1 nodes.

2) When the state network has an *input node* active, this corresponds to a transition occuring between the original states of the Turing machine. The input needs to both write a symbol on the tape and moves the tapehead. For instance, if the active node in the state network is an input node labelled $0, R$, an input is given to all the $0, R$ nodes in each of the tape sub-networks. For all of the tape sub-networks which have active $0'$ or $1'$ nodes, nothing will happen, but this perturbation will affect a transition for the tape sub-network which has an active 0 or 1 node (that is, the tape position corresponding to the position of the tapehead). Following this, in that tape sub-network, there will be an automatic transition to node $0'$. This part of the dynamics simulates the writing of the symbol 0 on the tape.

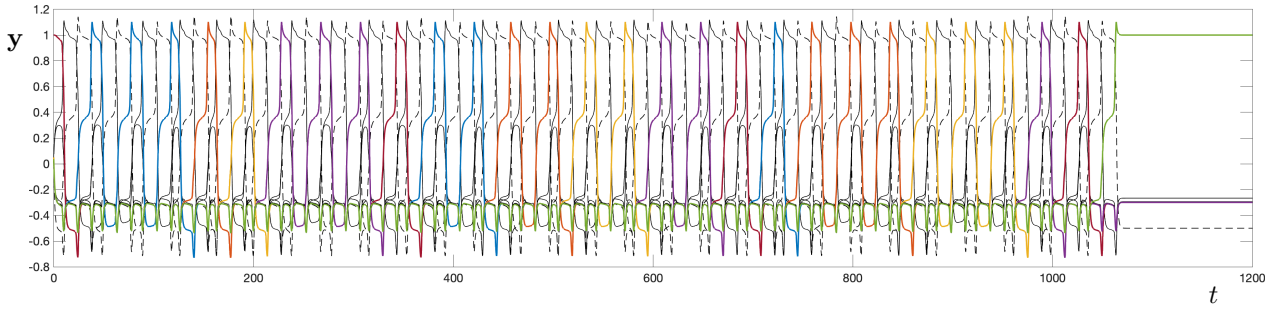

Fig. 7. A time series plot of all the states in the state network. The coordinates for the state nodes are coloured, in numerical order, red, blue, orange, yellow, purple and green. The input nodes are all black solid lines, and the delay nodes are black dashed lines. The halting state is reached at $t \approx 1080$. A zoom of the first 100 time units is shown in figure 8(a).

This input from the state network to the tape sub-networks is given by:

$$g_x(\mathbf{y}) = (v_{0R} \circ \phi(\mathbf{y}))w_{0R} + (v_{1R} \circ \phi(\mathbf{y}))w_{1R} + (v_{0L} \circ \phi(\mathbf{y}))w_{0L} + (v_{1L} \circ \phi(\mathbf{y}))w_{1L} \quad (5)$$

where each of the v_{ij} vector selects the appropriate dimensions from the state network, and each of the w_{ij} vectors provides the inputs into the correct coordinates in the tape sub-networks.

3) The movement of the tapehead along the tape is implemented by inputs from the tape sub-networks to their right or left neighbours. Specifically, when any of the nodes $1L$, $0L$, $1R$ and $0R$ in the tape sub-network are active, an input is given in the direction of nodes 0 and 1 in the tape sub-network either to the right or left, depending on the label. This has the affect of activating the 'unprimed' state 0 or 1 node in the sub-network to the right or left, indicating the movement of the tapehead. The input each sub-network receives from its neighbours are given by the functions

$$g_R(\mathbf{x}_j) = (\phi(\mathbf{x}_j) \circ w_R)w_{01}$$

and

$$g_L(\mathbf{x}_j) = (\phi(\mathbf{x}_j) \circ w_L)w_{01}$$

IV. A NUMERICAL EXAMPLE: THE COPY ROUTINE

In this section we demonstrate that the above equations do indeed provide an embedding of a Turing machine, again using the copy routine [38] as an example. We use a tape of length 10, and set initial conditions so that the tape has initial configuration 0011100000, and the tape head starts at tape position 3. This means that tape sub-network 3 has an active 1' node, tape sub-networks 4 and 5 have an active 1 node, and the remainder of the tape networks have the 0 node active. The state network starts with node 1 active. We expect the final configuration of the tape network to be 0011101110.

We use the following parameters in our integrations:

$$\begin{aligned} w_s = 1, \quad & w_m = -0.5, \quad w_t = -0.3, \\ w_a = 0.31, \quad & w_e = 0.29, \\ \zeta_x = 0.03, \quad & \zeta_y = 0.05, \quad \zeta_\beta = 0.02, \\ K = 20, \quad & \theta = 0.5. \end{aligned}$$

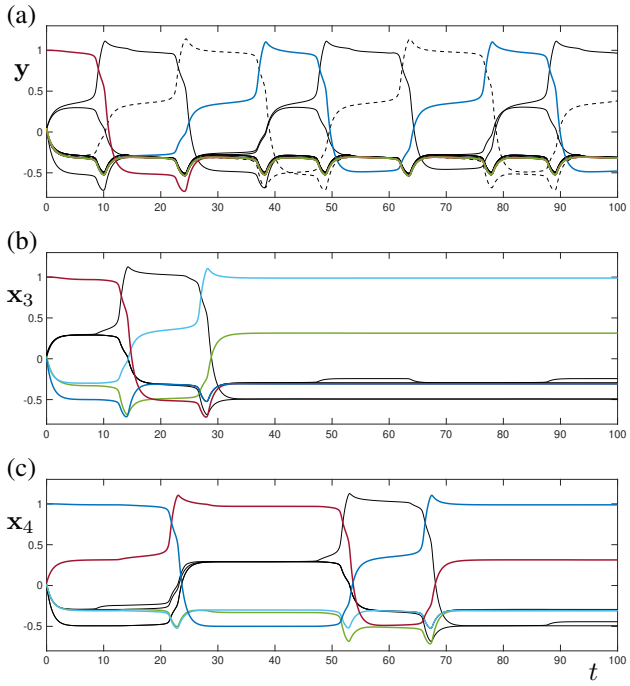

Fig. 8. The figure shows a zoom of a portion of the time series in figure 7. Panel (a) shows the state network, with line styles as in figure 7. Panels (b) and (c) show tape sub-networks 3 and 4, respectively. In these panels, the coordinates are coloured as: 0: green; 1: red; $0'$: light blue; $1'$: blue. The remaining coordinates are coloured black. Notice in panel (c), that when the $1'$ is active, the 1 coordinate has an intermediate value - this is because it is a *leading* node.

Figures 7, 8 and 9 show the results of numerical integration of equations 1 with the above parameters and initial conditions. Figure 7 shows a time series of the values of the coordinates in the state network. The coordinates corresponding to the state nodes are coloured (as detailed in the caption); the input nodes are black solid lines, and the delay nodes are black dashed lines. Note that the value of the coordinates can be close to four different values, depending on whether the nodes are *active* ($y \approx 1$), *leading* ($y \approx 0.3$), *disconnected* ($y \approx -0.3$) or *trailing* ($y \approx -0.5$).

Figure 8 shows a time series of only the first 100 time units of the integration, but includes the variables from the third and

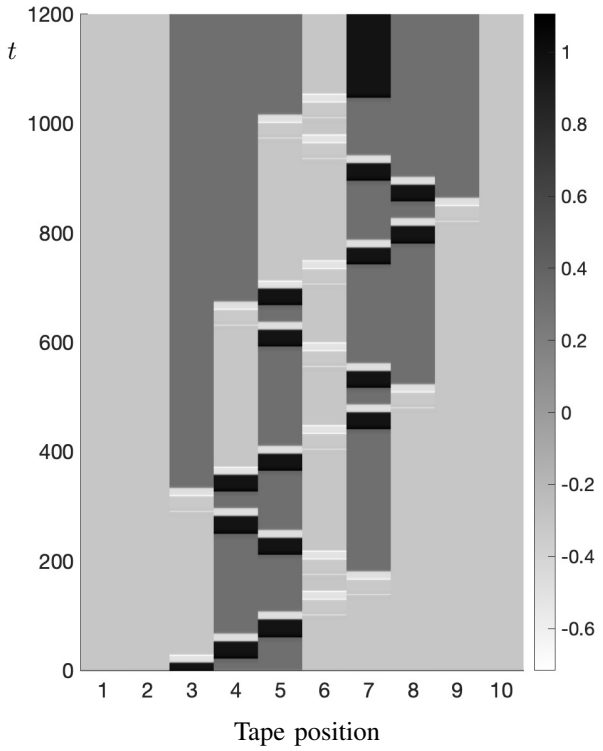

Fig. 9. The figure shows the value of the 1 node in each of the tape sub-networks over time. The black and dark gray regions together indicate which positions in the tape currently have the symbol 1; the light gray is the symbol 0. See text for more details.

fourth tape sub-networks as well as the state network. Here we can see the details of the transitions between the nodes and the inputs between the networks. Initially, the active node in the third tape sub-network is node 1, and this provides an input into the state network which activates the input node labelled $0, R$ (see figure 4). This transition is completed at around $t = 10$, at which point an input into the tape networks cause the third sub-network to transition from node 1 to node $0, R$. The input can also be seen as a perturbation in the fourth sub-network, but there is no effect here. The transition in the third tape sub-network to having node $0, R$ active is completed around $t = 13$, and the corresponding input from this node into the fourth sub-network affects a transition there from node 1' active to node 1 active. There are subsequently automatic transitions in the state network from the input node $0, R$ to the delay node, and then to the state node 2, and also in the third tape sub-network from node $0, R$ to node $0'$. Node 2 in the state network becomes active at around $t = 39$, with the third tape sub-network having node $1'$ active and the fourth tape sub-network having node 1 active, and the process can begin again.

In figure 9 we show the value of the activation level of the 1 node (i.e. the value of the variable corresponding to that node) for each of the tape sub-networks, in grayscale. When the 1 node is active, the value is close to 1, and shows as black. When the 1 node is *leading*, the value is close to 0.3 and shows as dark grey. Because of the topology of the network (see figure 5), if the 1 node is leading, the $1'$ node must be active, so together the black and dark grey regions indicate when the symbol on the tape at that position is a 1. The white regions indicate when the 1 node is trailing, which means that one of the $(0, R)$, $(0, L)$, $(1, R)$ or $(1, L)$ nodes must be active, which means that the tape head is (recently) at that position in the tape. Thus, this figure gives us all the information about the symbols on the tape, and the position of the tape head. Note that the copy routine has worked successfully: at the end of the integration, the 1 symbol appears in positions 3, 4, 5, 7, 8 and 9 on the tape.

A. Behaviour in presence of noise

We consider the effect of additive noise to equations (1), so that, for example, equation (1a) becomes

$$d\mathbf{y} = [f_y(\mathbf{y}) + \zeta_y g_y(\phi(\mathbf{x}))]\, dt + \sigma d\mathbf{W}(t)$$

where the parameter σ controls the noise amplitude and $\mathbf{W}(t)$ represents i.i.d. Gaussian noise (a Wiener process) with mean 0 and variance 1 per unit time in each component. The effects of additive noise to heteroclinic networks, which are dynamical object closely related to the network attractors described here, are complex; see for some examples [39], [40], [41]. In [35], we considered the effect of noise on the CTRNN network attractors we describe here.

The first effect, for small noise, is that the transition times between nodes is no longer a constant but a random variable. As noise is increased, the mean transition times typically decrease, and the variance increases. When noise becomes sufficiently large, the noise can act as a perturbation which can cause the trajectory to follow an excitable connection that it otherwise would not have followed. Thus the trajectory follows the 'incorrect' route around the network and this results in an error in the computation performed by the Turing machine. We note that errors that give transitions of the state network that are incorrect may be "read errors" (where the error is equivalent to an incorrect reading of the symbol on the tape) or "wild errors" (where the state network jumps to a state which could not be correctly arrived at through any symbol on the tape) — both of these error are also described in [33]. In addition, there may be "tape errors" where noise causes the tape to malfunction by, for instance, the tape head moving in an incorrect direction, or not moving at all.

The relative size of the noise required to cause such errors, and the frequency and type of the errors, will depend on the choice of parameters. A detailed investigation of this relationship is beyond the scope of this paper; see [33] for a detailed study of the effect of noise on a differently designed Turing machine, and [42] for a study of the interaction between the noise amplitude and parameters in a related system, including an explanation of some non-intuitive results.

For the parameter set we use for the integrations above, we find that for $\sigma \lesssim 0.005$ the Turing machine typically completes the computation correctly, with the variation in transition times becoming more obvious for larger values of the noise. When $\sigma \gtrsim 0.01$, the machine rarely completes the task successfully.

V. Network and dynamical properties

Having presented the system, we can now return our attention to the questions raised in the introduction. The system was

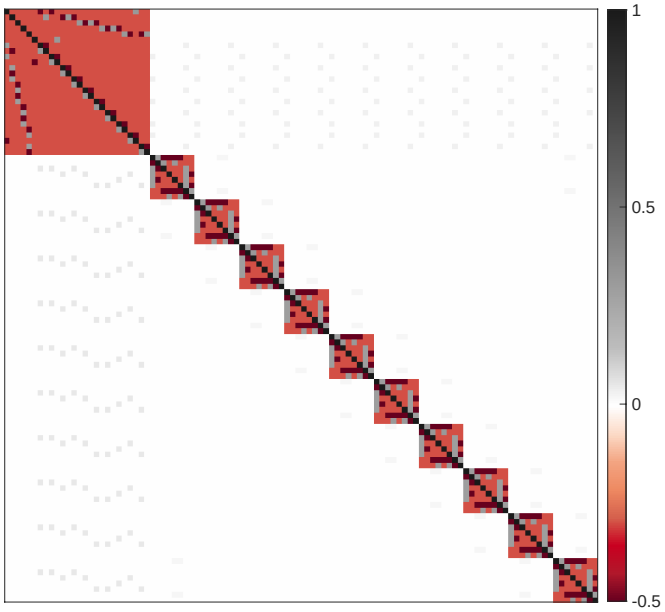

Fig. 10. The figure shows the relative size (as indicated by the colourbar) of each of the elements in the weight matrix. Note that there are non-zero elements outside of the block diagonal which are of very small magnitude (very light grey). The matrix entries are unlabelled, but range from $(1, 1)$ in the top left, to $(106, 106)$ in the bottom right, as would be expected.

designed to act as a computational device, a Turing machine. What is the dynamical system that emerged in this design process and what, if anything, stands out about its *structure*, i. e. the ways that the state variables relate to each other? We have identified three such salient properties which we now outline.

A. Properties of the weight matrix

Recall, firstly, that if an entry (i, j) of the weight matrix is zero, then node j does not directly affect node i. We can thus consider the directed graph of connections made only by considering the non-zero entries of the weight matrix. Then, by 'topology' we mean the structure of the connections of this graph.

There are three noteworthy features of the weight matrix. It is highly sparse; the entries in the weight matrix can only take one of eight different values; and it contains a high degree of topological regularity.

All three of these aspects can be seen in figure 10, which shows the values of the entries in the weight matrix of equation (2), for the copy routine with a tape of length 10. The first 26 entries correspond to the state network, and the remaining 80 are divided into ten groups of eight, one for each of the tape sub-networks.

The largest non-zero entries of the weight matrix can be seen in a block diagonal form, which indicate the couplings between the nodes in the state network and between the nodes in each of the tape sub-networks. Within each of these blocks, each matrix entry can take one of five different values: $w_s = 1$ (black), $w_m = -0.5$ (dark red), $w_t = -0.3$ (light red), $w_a = 0.31$ and $w_e = 0.31$ (both grey). Within the state network, there are further regularities which can be observed in the couplings for nodes 7-26: these are the delay and input nodes and each pair has couplings that are chosen in a repeated and patterned fashion. There also exists a translational symmetry between the sub-networks of the tape network: each is coupled to the state network in the same way, and in the same way to the sub-networks to their left and right.

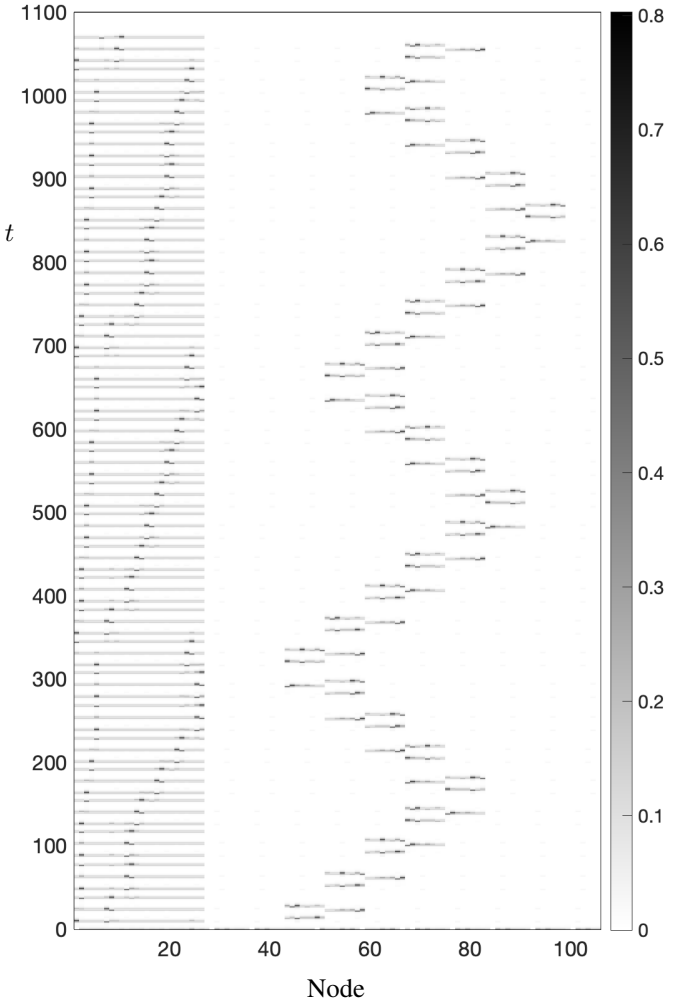

Fig. 11. The figure shows the speed (absolute value of velocity), indicated by the grayscale, of each of variables describing the activity of the nodes in the trajectory shown in figure 7.

Outside of the block diagonal entries, the only non-zero entries correspond to the couplings between the sub-networks, and these are both sparse and small in magnitude. These can take one of three different values (γ_x, γ_y and γ_β), all of which appear light grey in the figure.

B. Dynamical sparsity

In additional to the sparseness of the weight matrix of connections, the *dynamics* of the system are also sparse, in the sense that the majority of the variables (i.e. the excitation levels of each node) in the system are close to being at rest for the majority of the time. In particular, for much of the time, the system is very close to equilibrium, and even when the

system is not close to equilibrium, the number of variables that are changing is much smaller than the total dimension of the system. This can be seen in figure 11, which plots the absolute value of the velocity of each of the variables as a function of time, for the same trajectory as shown in figure 7. As in figure 10, the first 26 nodes correspond to the state network, and the remaining 80 to the tape network. It can clearly be seen that the maximum speed of ~ 0.8 is attained by isolated nodes only for short periods of time. Lower speeds are attained by 'nearby' nodes in a structural sense (compare with the weight matrix in figure 10). Nodes which are not directly connected to the fastest moving node (e.g., those in distant tape sub-networks) have speeds which are essentially zero. One can criticize our realization as being sparse and inefficient relative to activity seen in natural neural systems and it is a very interesting and relavent question to find more efficient and dense encoding of states that are still explictly understandable. Nonetheless, we note that our realization is much less sparse than for example [27] who use a synfire chain to encode a single bit of information.

C. Temporal regularity

In addition to the structural regularities noted above, the resulting observed dynamics also have what we refer to as *temporal regularity*. This is also related to the dynamical sparcity previously discussed. Specifically, not only are most of the varibales at rest for much of the time, but in addition, the timing of when variables are *not* at rest is highly structured: many varibles move in some form of synchronicity, and there is a regular waiting time of inactivity between periods of activity.

In terms of the underlying nonlinear dynamics, the Turing machine without the tape consists of a number of excitable network attractors (one for the state network, and one for each of the tape sub-networks) where intrinsic transitions will not occur without external inputs or noise. When these networks are coupled, the dynamics will slowly move through "bottlenecks" in phase space that determine the timing of the transitions between the states.

This temporal regularity can be seen in figure 7, as the regular appearance of spikes in the timeseries, and also in figure 11, as the regular appearance of horizontal stripes indicating that some of the nodes are changing. During each stripe, the first set of nodes to move are in the state network, and these affect transitions in the tape networks in a domino-like fashion: first to write a new symbol on the tape, and then to move the tapehead in the correct direction. After this period of activity, the nodes relax to near equilibrium, eventually transitioning through the bottlenecks in phase space that trigger the next transition.

Note that compared to implementations that need an external clock [28], or such as [27] that dedicate a separate part of the system as a clock, this regularity of timing, or 'clock' is an emergent property of the dynamics. Furthermore, the regularity is holistic - it cannot necessarily be seen by only observing a subset of the variables.

D. Why are these salient features present?

Having identified these features, we can consider why they are present. We did not consciously choose to add them, so why are they there? Why is the weight matrix of the designed system sparse? Why do only a few of the state variables change significantly at any given time? And why do we observe temporal regularity—a kind of periodic pulsing of change in the system?

One possibility is these features appeared by chance—that if someone else were to independently implement a Turing machine by specifying a set of ODEs, their implementation might not include one or more of the properties just described. This seems unlikely to us, though possible. The alternative possibility is that the specification of what we wanted to build (a Turing machine) and the design process that produced this system both impose constraints that increase the likelihood of these features appearing. We now elaborate upon these 'specification constraints' and 'design-process constraints.'

For the designed system to act as a Turing machine, it must satisfy a number of criteria. For example, it must be capable of undergoing a variety of trajectories that correspond with the execution of different programs and/or different inputs. This particular *specification-constraint* excludes many possible designs. As a trivial example, we can make the sweeping generalization that any design that consists of only a single ordinary differential equation of the form $\frac{\mathrm{d}x}{\mathrm{d}t} = kx$ is not a viable design as we know it fails this part of the specification, as it is not capable of sufficiently diverse trajectories. This and other specification-constraints are determined by how "Turing-machineness" is defined, and so it is possible that the salient features described above are present because to be a Turing machine, a system *must* have these features.

However, the Turing-machine specification constraints do not completely determine the design of the final system on their own. Other, *design-process constraints* also influence the set of possible final designs. In particular, some constraints are present simply because the system is designed by people. As argued by Thompson et al. [43], when people use classic engineering methods, they must understand what they are engineering and this limits the forms of things that people can engineer. To make it possible to understand what they are building, engineers include in their design features that limit system complexity. They do this to be able to reason about, predict, summarise and even prove how the system will behave in its normal operating conditions. It is thus also possible that the salient features described above are present not because they are a necessary part of a dynamical system acting as a Turing machine, but rather because they were helpful (or even necessary) for us to understand the system well enough to develop it.

As part of their argument, Thompson et al. [43] emphasize the hierarchichal nature of typical by-human engineering: how engineers take a big problem that is impossible to solve directly and decompose this problem into sub-problems which are in turn further decomposed into sub-problems, until the problems are simple enough to be understood, reasoned about and solved. It seems likely that the salient features described

above are (at least partially) the result of this kind of decomposition. We see repeated, identical units in the tape and the state networks. We also see limited communication between these units (avoidance of 'cross-talk'). This kind of modularization and compartmentalization (also used extensively for example in [27]) helps understandability of the construction, and the three features, topological and dynamical sparsity and regularity could be a direct result of these "so-we-could-understand" features.

A question now presents itself: which of these features are necessary for a system to be considered computational? All of them? None of them? By design, the system is both a dynamical system and a computational system. Is it possible to have a dynamical system that is also a computational system, but that does not have the salient dynamical properties that we have highlighted above? This is an interesting question. Unfortunately, at this stage, the answer is not clear, though there are a variety of further experiments that could be done to explore this possibility. For example, we could try to design a dynamical Turing machines that explicitly does not have one (or more) of the salient properties just described. These efforts might include trying to use different other types of differential equations, such as other forms of ODEs, partial differential equations or delay-differential equations.

An alternative approach that might reduce the effects of by-human design, we could use an optimization algorithm (e. g. an evolutionary algorithm) to change the weights of the system described above. The multi-objective optimisation process might, for example, force the output of the system to stay the same while also (i) minimizing the number of nodes that are ever active (the intent here is to reduce the dimensionality of the system) and/or (ii) optimizing of the time spent performing the computation (without simply changing the timescale).

Success at implementing a dynamical Turing machine without one of these salient features, whether in by-human engineering or via an optimization algorithm, would show that that feature is not an essential property of a Turing machine. Such work might also help us to understand more broadly what kinds of systems we can consider to be Turing machines, or more broadly as computing devices.

VI. Conclusion

We have presented a dynamical continuous time recurrent neural network architecture that allows logical manipulations of a symbolic nature and indeed universal computations in the sense of Turing. The construction highlights that the Dynamic Hypothesis and the Computational Hypothesis (DH/CH) are not necessarily inherently contradictory, and we hope that the system provides an approach that can be used productively in the debate between dynamical and computational perspectives of cognition, going forward.

In doing so, we highlight some limits on what can be represented; the tape capacity is finite and computational errors will arise when this capacity is reached. As discussed in [33], errors during computation in a real, noisy, system may be of various types. Transitions between states corresponding to false interpretation of tape symbol may give rise to a "read error" on the tape while "wild errors" can occur that correspond to transitions not allowed by any read. In addition to these, "tape errors" may occur in tape transport or writing symbols to the virtual tape.

The excitable network framework for discrete state, discrete time neural computation is also flexible in the way the timing of transitions between states are made. As we describe in section V, the system is free-running and self-contained, in the sense that the timescales are determined by the network itself. However, as discussed in [33], there is a framework under which the transitions could be externally clocked by external inputs. The same approach should work when modelling systems working in parallel asynchronously [44] and only interfacing when they arrive at some event where synchrony is required. In addition to the timing of valid functioning of a computational network, the framework can be helpful explaining characterizing possible errors that may arise through stochastic perturbations in the system.

The realisation of the Turing machine we present here is idealised in that it is an input-free (autonomous) system where not only the machine but also the tape is embedded in the CTRNN. This realization is comparatively compact in that each state is directly represented as on element in the network rather than needing synfire chains as in [27]. We suggest a generally programmable computational system with external inputs could be designed in a similar way, by including input-dependence (e.g. making the "tape" external) and/or making weights adaptive to a changing environment.

Code availability

The Matlab code used to implement the system described in this paper is available at https://github.com/mathclaire/ctrnn_turingmachine.

Acknowledgments

CMP thanks the Royal Society Te Apārangi (Marsden Fund Council, NZ Government), for funding via 21-UOA-048. PA thanks the UK EPSRC for support via grant numbers EP/T018178/1 and EP/T017856/1. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

References

[1] T. van Gelder, "The dynamical hypothesis in cognitive science," *Behavioral and Brain Sciences*, vol. 21, no. 05, pp. 615–628, Oct. 1998.

[2] J. A. Fodor, *The Language of Thought*. Harvard university press, 1975, vol. 5.

[3] A. Newell and H. A. Simon, "Computer science as empirical inquiry: Symbols and search," *Communications of the ACM*, vol. 19, no. 3, pp. 113–126, 1976.

[4] C. Buckner and J. Garson, "Connectionism," in *The Stanford Encyclopedia of Philosophy*, fall 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019.

[5] R. A. Brooks, "Intelligence without reason," *Artificial intelligence: critical concepts*, vol. 3, 1991.

[6] H. J. Chiel and R. D. Beer, "The brain has a body: Adaptive behavior emerges from interactions of nervous system, body and environment," *Trends in Neurosciences*, vol. 20, no. 12, pp. 553–557, Dec. 1997.

[7] F. Lieder and T. L. Griffiths, "Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources," *Behavioral and Brain Sciences*, vol. 43, p. e1, Jan. 2020.

[8] D. L. Barack and J. W. Krakauer, "Two views on the cognitive brain," *Nature Reviews Neuroscience*, vol. 22, no. 6, pp. 359–371, Jun. 2021.

[9] S. Risi and M. Preuss, "From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI," *KI - Künstliche Intelligenz*, vol. 34, no. 1, pp. 7–17, Mar. 2020.

[10] R. D. Beer, "The Dynamics of Active Categorical Perception in an Evolved Model Agent," *Adaptive Behavior*, vol. 11, no. 4, pp. 209–243, Jan. 2003.

[11] R. D. Beer and P. L. Williams, "Information Processing and Dynamics in Minimally Cognitive Agents," *Cognitive Science*, pp. n/a–n/a, Jul. 2014.

[12] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[13] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, 1995.

[14] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[15] E. Izquierdo, I. Harvey, and R. D. Beer, "Associative Learning on a Continuum in Evolved Dynamical Neural Networks," *Adaptive Behavior*, vol. 16, no. 6, pp. 361–384, Dec. 2008.

[16] I. Harvey, E. D. Paolo, R. Wood, M. Quinn, and E. Tuci, "Evolutionary Robotics: A New Scientific Tool for Studying Cognition," *Artificial Life*, vol. 11, no. 1-2, pp. 79–98, Jan. 2005.

[17] R. D. Beer, "Beyond Control: The Dynamics of Brain-Body-Environment Interaction in Motor Systems," in *Progress in Motor Control*, D. Sternad, Ed. Boston, MA: Springer US, 2009, vol. 629, pp. 7–24.

[18] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, p. 115–133., 1943.

[19] M. L. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ,: Prentice-Hall, Inc., 1967.

[20] J. Šíma and P. Orponen, "Continuous-Time Symmetric Hopfield Nets Are Computationally Universal," *Neural Computation*, vol. 15, no. 3, pp. 693–733, 03 2003. [Online]. Available: https://doi.org/10.1162/089976603321192130

[21] E. Byrne and C. Huyck, "Processing with cell assemblies," *Neurocomputing*, vol. 74, no. 1, pp. 76–83, 2010, artificial Brains.

[22] C. H. Papadimitriou, S. S. Vempala, D. Mitropolsky, M. Collins, and W. Maass, "Brain computation by assemblies of neurons," *Proceedings of the National Academy of Sciences*, vol. 117, no. 25, pp. 14 464–14 472, 2020.

[23] J. Cabessa and A. Tchaptchet, "Automata complete computation with hodgkin–huxley neural networks composed of synfire rings," *Neural Networks*, vol. 126, pp. 312–334, 2020.

[24] F. L. Traversa and M. Di Ventra, "Universal memcomputing machines," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2702–2715, 2015.

[25] T. Wu, L. Pan, Q. Yu, and K. C. Tan, "Numerical spiking neural p systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2443–2457, 2021.

[26] B. Paaßen, A. Schulz, T. C. Stewart, and B. Hammer, "Reservoir memory machines as neural computers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 6, pp. 2575–2585, 2022.

[27] J. Cabessa, "Turing computation with neural networks composed of synfire rings," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[28] H. T. Siegelmann, "Computation beyond the turing limit," *Science*, vol. 268, no. 5210, pp. 545–548, 1995.

[29] J. Cabessa and H. Siegelmann, "The super-turing computational power of plastic recurrent neural networks," *Int J Neural Syst.*, vol. 24, 2014.

[30] W. Maass and C. M. Bishop, *Pulsed neural networks*, 2001.

[31] C. R. Huyck and P. J. Passmore, "A review of cell assemblies," *Biological cybernetics*, vol. 107, pp. 263–288, 2013.

[32] Y. Fan and C. Huyck, "Implementation of finite state automata using flif neurons," in *2008 7th IEEE International Conference on Cybernetic Intelligent Systems*. IEEE, 2008, pp. 1–5.

[33] P. Ashwin and C. Postlethwaite, "Sensitive finite-state computations using a distributed network with a noisy network attractor," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5847–5858, 2018.

[34] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation," *Acm Sigact News*, vol. 32, no. 1, pp. 60–65, 2001.

[35] P. Ashwin and C. Postlethwaite, "Excitable networks for finite state computation with continuous time recurrent neural networks," *Biological cybernetics*, vol. 115, no. 5, pp. 519–538, 2021.

[36] G. S. Carmantini, P. beim Graben, M. Desroches, and S. Rodrigues, "A modular architecture for transparent computation in recurrent neural networks," *Neural Networks*, vol. 85, pp. 85–105, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608016301198

[37] S. H. Strogatz, *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*. Addison-Wesley, 1994.

[38] "Turing machine examples," https://en.wikipedia.org/wiki/Turing_machine_examples#A_copy_subroutine, accessed: 2023-02-22.

[39] P. Ashwin and C. M. Postlethwaite, "Quantifying Noisy Attractors: From Heteroclinic to Excitable Networks," *SIAM J. Appl. Dynam. Syst.*, vol. 15, no. 4, pp. 1989–2016, 2016.

[40] Y. Bakhtin, "Noisy heteroclinic networks," *Probability theory and related fields*, vol. 150, no. 1, pp. 1–42, 2011.

[41] E. Stone and P. Holmes, "Random perturbations of heteroclinic attractors," *SIAM Journal on Applied Mathematics*, vol. 50, no. 3, pp. 726–743, 1990.

[42] V. Jeong and C. Postlethwaite, "Effect of noise on residence times of a heteroclinic cycle," *Dynamical Systems*, pp. 1–23, 2022.

[43] A. Thompson, P. Layzell, and R. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 167–196, 1999.

[44] C. Bick and M. Field, "Asynchronous networks and event driven dynamics," *Nonlinearity*, vol. 30, no. 2, p. 558, jan 2017. [Online]. Available: https://dx.doi.org/10.1088/1361-6544/aa4f62

Claire Postlethwaite is an Associate Professor of Mathematics at the University of Auckland. Her research interests include applications of dynamical systems and mathematical ecology. Within the area of applied dynamical systems, one of her main interests is understanding the dynamics near heteroclinic cycles and networks, and using them as models for physical systems.

Peter Ashwin is Professor of Mathematics at the University of Exeter. His research interests are especially nonlinear dynamical systems and applications, especially for systems that are coupled and/or symmetric: in these settings it is often possible to find attractors that have network structure. He is also interested in ergodic properties of dynamical systems, nonautonomous dynamics and computational modelling for a variety of applications.

Matthew Egbert is a Lecturer in the Department of Computer Science at the University of Auckland. His current research involves interdisciplinary collaborative investigation of mind and life that emphasizes the situated, embodied, dynamical and self-maintaining nature of these systems.