DOI 10.1007/s00165-014-0294-y
BCS © 2014
Formal Aspects of Computing (2014) 26: 1169–1204

Formal Aspects of Computing

Analysing neurobiological models using communicating automata

Li Su[1,2], Rodolfo Gomez[3] and Howard Bowman[3,4]

[1] Department of Psychiatry, University of Cambridge, Level E4, Box 189, Addenbrooke's Hospital, Hills Road, Cambridge, CB2 0QQ, UK
[2] Department of Psychology, University of Cambridge, Cambridge, UK
[3] Centre for Cognitive Neuroscience and Cognitive Systems, School of Computing, University of Kent, Canterbury, UK
[4] School of Psychology, University of Birmingham, Birmingham, UK

Abstract. Two important issues in computational modelling in cognitive neuroscience are: first, how to formally describe neuronal networks (i.e. biologically plausible models of the central nervous system), and second, how to analyse complex models, in particular, their dynamics and capacity to learn. We make progress towards these goals by presenting a communicating automata perspective on neuronal networks. Specifically, we describe neuronal networks and their biological mechanisms using Data-rich Communicating Automata, which extend classic automata theory with rich data types and communication. We use two case studies to illustrate our approach. In the first case study, we model a number of learning frameworks, which vary in respect of their biological detail, for instance the Backpropagation (BP) and the Generalized Recirculation (GeneRec) learning algorithms. We then used the SPIN model checker to investigate a number of behavioral properties of the neural learning algorithms. SPIN is a well-known model checker for reactive distributed systems, which has been successfully applied to many non-trivial problems. The verification results show that the biologically plausible GeneRec learning is less stable than BP learning. In the second case study, we presented a large scale (cognitive-level) neuronal network, which models an attentional spotlight mechanism in the visual system. A set of properties of this model was verified using Uppaal, a popular real-time model checker. The results show that the asynchronous processing supported by concurrency theory is not only a more biologically plausible way to model neural systems, but also provides a better performance in cognitive modelling of the brain than conventional artificial neural networks that use synchronous updates. Finally, we compared our approach with several other related theories that apply formal methods to cognitive modelling. In addition, the practical implications of the approach are discussed in the context of neuronal network based controllers.

Keywords: Neuronal networks, Communicating automata, Model checking, Backpropagation, Generalized recirculation algorithm, Plasticity–stability dilemma, Visual attention, Cognitive neuroscience, AI, SPIN, PROMELA, Uppaal

Correspondence and offprint requests to: L. Su, E-mail: ls514@cam.ac.uk

1. Introduction

There has recently been much interest in neuronal network modelling specifically targeted at simulating the brain in cognitive neuroscience [AIS08]. One of the main objectives for such biologically plausible models is to define and simulate cognitive processes at the cellular, circuit or subsystem-level (e.g. a single neuron or assembly of neurons). Combining principles from systems biology, computational and systems neuroscience is proving a promising approach to understanding how the brain gives rise to humans' cognitive capacities. This paper applies concurrency theory in modelling and verification of neuronal networks and their adaptive behaviours. Specifically, we introduce a formalism called Data-rich Communicating Automata (DRCA), which is used as a general framework for describing neuronal networks and their biological mechanisms. In two case studies, we explore the capacity of such theoretical models to study how neurons interact to perform learning tasks and how visual systems allocate attention in the visual field. The aim of this paper is to specify neural processes in neuronal network models and reason about their properties in a rigorous manner. In particular, information processing within neuronal networks is systematically specified in an automata notation. The interaction between neurons and the environment, in which they behave, is also made explicit. That is, the observable behaviour of biological systems is specified explicitly using logical formulae. In this respect, cognitive phenomena or mental processes could be seen as emergent behaviours of a collection of complex neural activities in space and time.

It is important to note that the class of neuronal network model that we specify and analyse is more sophisticated than those arising from the classic (connectionist) neural network framework. In particular, we employ and build upon what O'Reilly and Munakata called *the point neuron model* [ORE00], which is more neurophysiologically plausible than the classic connectionist model. In particular, activation update equations we specify can be linked to Hodgkin–Huxley equations [HOD52], and thus directly reflect ion channels crossing the cell membrane, fluctuation in membrane potential and the shunting properties known to underlie biological neuron dynamics. In this way, the neuronal networks we specify fit with the current emphasis on linking cognition to neuroscience (i.e. the cognitive neuroscience revolution).

We have, though, not gone as far as to explicitly model neural spiking. Thus, our models remain rate coded, with the key variable modelled being the rate of neural spiking. In particular, it is important to note that there is no absolute criteria for judging biophysiological plausibility—indeed, spiking models are themselves abstractions. Thus, a claim of neurophysiological plausibility is a relative statement, i.e. relative to some existing norm, and ours is relative to the classic/conventional (rate coded) artificial neural network framework, as found in what traditionally would be called connectionist models. Our claim is to have added neurophysiological plausibility to those models.

In the context of this paper, one of the motivations for applying formal methods in cognitive modelling is to specify and justify behaviour of complex cognitive architectures in an abstract and accurate way. Firstly, numerous cognitive theories assume that mental modules, or their neural substrates, are processing information at the same time. Hence, any realistic cognitive architecture must be concurrent at some level. Secondly, the control of concurrent processing can be distributed. So, cognition should be viewed as the behaviour that emerges from interaction between independently evolving modules. Finally, hierarchical decomposition is needed in order to reflect the characteristics of the mind and model complicated mental processes [FOD88]. Here, we apply the DRCA notation to specify neuronal network models. This approach has the advantage of allowing distributed and concurrent processing. It also potentially enables neuronal networks to be built hierarchically and compositionally, i.e. complex systems can be composed by joining subsystems together. However, while concurrency, distribution and hierarchical decomposition are key objectives of our work, the necessity to progress to verifiable models has superseded the full realisation of some of these aims, in particular, full provision of hierarchical decomposition in our neuronal networks awaits further work (but see Sect. 6.1).

Moreover, complex biological systems, such as neuronal networks, often cannot be explained at a single level [LIL94]. In response to similar challenges, computer science, and particularly software engineering, boasts a plethora of multiple perspective approaches [BOW01, BOW02, DER01, KIC97, ROS98]. Probably the longest standing and most extensively investigated question is how to relate descriptions at different levels of abstraction, e.g. the requirements level, i.e. the abstract specification of "what the system must do", and the implementation level, i.e. the structurally detailed realisation of "how the system does it". Furthermore, there has been much work on how to relate the requirements level to the implementation level, which, in logical metaphors, amounts to demonstrating that the implementation satisfies the requirements or, in other words, that the implementation is a model of the requirements. In this respect, computer scientists have developed a large number of theories and tools to verify that a model satisfies its properties. One such technique is model checking [MCM93, HEN94,

BEN95, CLA00, HOL03], which has a number of advantages over other techniques, such as simulation. For instance, it allows exhaustive formal verification.

Analysing the properties of neuronal networks also has implications beyond cognitive science and cognitive neuroscience. It has been widely recognised that Artificial Intelligence (AI), including various neural learning algorithms, is attractive for improving applications that are difficult to implement using other techniques [MEN05]. However, developers must gain confidence in the correctness of the systems, in particular, when considering mission or life critical situations. Menzies and Pecheur have discussed several readily available methods[1] in terms of their applicability to the verification and validation (V&V) of AI systems, and found that traditional techniques are limited in their capacity to analyse AI systems [MEN05]. They have also pointed out that a limitation of model checking AI systems is that the transformation of the AI systems to formal languages may be difficult and expensive. However, we argue that the basic components of neuronal networks are homogeneous, hence once specified, more complex models could be constructed from such components. (This is not always possible for other types of AI systems.) In addition, as will be discussed in later sections, mathematical analysis (and theorem proving) may be labour-intensive, and require expert skills and effort.

The main contribution, then, of this paper is to provide a proof of concept that formal methods, and particularly model checking can be used to analyse the dynamics of neural networks, and especially the dynamics of a certain class of neurophysiologically more detailed neural nets. Most significantly, we will show that certain stability properties can be verified over neural networks using model checking. Specially, our first case study, see Sect. 3, assesses the effectiveness of our communicating automata—model checking approach to assessing the stability of learning; that is, to evaluating whether neural network leaning algorithms will always reach a state where it has learnt the given task and it will never lose the capacity to perform that task with further training. To test our method in this respect, we focused on learning the XOR problem [HAM98, SPR96]. XOR was chosen because it is a "canonically difficult" learning problem. This is because it is characteristic of linearly inseparable learning. In particular, even though XOR is a small problem in terms of network size and number of patterns, it is very hard to learn.

We, then, address the issue of our method's robustness to network size in our second case study, which considers a large neural network model of visual attention. This illustrates the breadth of applicability of the method, i.e. to a domain beyond learning. This second case study also enables us to illustrate how the asynchronous parallelism offered by the communicating automata paradigm (and, indeed, concurrency theory methods in general) bring execution efficiency benefits.

In this paper, Sect. 2 introduces the DRCA notation. Then, Sect. 3 introduces the first case study. We specified neuronal networks and their environment, as well as, two different learning algorithms. Then, the communicating automata models are translated to the SPIN model checker [HOL03] to verify fundamental stability properties in neural learning. In Sect. 4, the DRCA models are extended with asynchronous updates and real time, so Uppaal model checker [BEN95] was used to verify a number of behavioural properties regarding visual attention. In Sect. 5, we draw concluding remarks. Finally, Sect. 6 discusses and compares our models with other related theories.

2. Data-rich communicating automata

For readers not familiar with the basic principles of model checking and communicating automata, this section provides brief background. However, readers are encouraged to follow the references to obtain a full perspective.

Data-rich Communicating Automata extend classic automata theory with rich data types and communication. This formalism allows a natural representation of biological systems, in particular neuronal networks, which are inherently parallel and distributed. In this notation, real values can be used as variables and constants. We define our DRCA model using a graphical notation, which makes it easier for non-experts to use. Moreover, its formal semantics potentially allow for automatic verification of correctness properties. However, there exist by now a number of mature tools that perform automatic analysis of models specified in communicating automata [BOW06] (SPIN and Uppaal being two of the most prominent examples [HOL03, BEN95]), which can be used to simulate and verify DRCA models. Here, we introduce an untimed DRCA approach; a timed extension is presented in Sect. 4.

[1] Theses methods are: scenario based testing, run-time monitoring, static analysis, model checking and theorem proving.

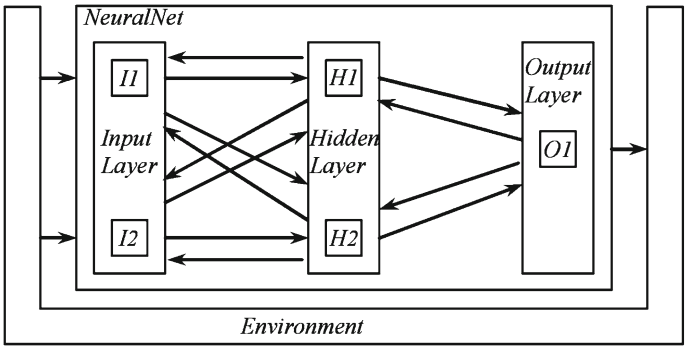

Fig. 1. A network view of a three layer recurrent neuronal network. The *smallest boxes* are neuron automata and *bigger boxes* around them are layers and networks. *Arrows* are message passing channels that transmit activations

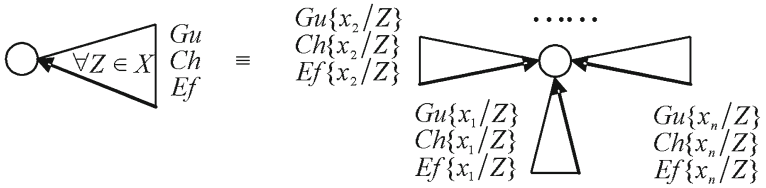

Fig. 2. A parameterised loop (*left*) is expanded out to the structure shown on the *right*. *Gu*, *Ch* and *Ef* are expressions which refer to Z

Our neuronal network models are networks of automata, each of which represents a neuron (called neuron automaton in this paper). The graphical notation contains a *network view* and *behavioural specifications*. At the top level, a network view defines how neurons are connected, i.e. neuron automata are represented by boxes and connected by message passing channels represented as arrows. It is through these channels that activation is transmitted between neurons. As shown in Fig. 1, neuronal networks are specified as a hierarchy of components. That is, at the top-level, it has two modules: *Environment* and *NeuralNet*. Each of which can be composed from a set of modules. For example, *NeuralNet* is composed of *InputLayer*, *HiddenLayer* and *OutputLayer*, each of which is composed of a set of neuron automata. For standard backpropagation (BP) learning, there are only feedforward connections, but BP learning with recurrent neuronal networks (BPRec) and the Generalized Recirculation (GeneRec) algorithm use both feedforward and feedback dynamics, i.e. the neurons are fully connected in both directions between adjacent layers [ORE00].

The behavioural specification of an automaton contains locations denoted by circles (with an initial location denoted by a double circle, as found, for example, in the Uppaal model checker [BEN95]) and transitions denoted by arrows, which are often associated with communication and evaluation of data expressions. Transitions may be associated with a condition under which it will take place. We call such a condition a *guard*, denoted by *Gu*. Neural activations are distributed to successor neurons via a set of communication channels, labeled by *Ch*. We can also specify an effect set, denoted by *Ef*, which contains a number of functions to update variables. The notation used to describe the neuronal networks also contains a number of (presentation simplifying) conventions. In particular, we use a parameterised transition notation, as shown in Fig. 2, where $X = \{x_1, x_2, \ldots, x_n\}$, and $t\{x/y\}$ denotes a substitution, which replaces all occurrences of y in the expression t by x. This notation denotes that there are n self transitions; each involving instantiation of the variable Z with an element from the set X. In addition, the scope of Z is restricted to this self transition and expressions associated with this self transition, such as *Gu*, *Ch* and *Ef*.

Semantically, communicating automata are interpreted over transition systems [MOL90], which are triples: $(S, s_0, \mapsto)$, where S denotes a set of states (pairs comprising a location vector and valuation), s_0 denotes an initial state, and $\mapsto$ denotes a transition relation. In a transition, when the guard holds, automata move from one location to another and update data variables. Please see "Appendix A" for the full semantics of DRCA.

It is important to note that, while none of the constituents of our DRCA notation is in itself new, the particular combination of constructs that make up the notation has not, to our knowledge, been presented. For example, our notation uses data passing synchronous communication (inspired by the communication mechanism in LOTOS [BOL88, BOW06]) and this notation is combined with real valued data types. Thus, we believe the specific combination of constructs is particularly tailored to the needs of neuronal network specification. Perhaps our most significant contribution, though, is not the DRCA notation per se, but the resulting framework for neuronal network specification, and indeed analysis. We present a format by which neuronal networks can be formally specified and then verified.

3. Case study 1: biological learning

Comparing the properties of different learning algorithms has significant scientific value, especially when biological plausibility is considered. In this case study, we investigate classes of neuronal networks that differ with respect to how they employ biological principles, and whether they can learn in a stable fashion. The requirement of stability arises in the context of the plasticity–stability dilemma [GRO76], which observes that animals have to continuously learn new aspects of a changing environment, and maintain their obtained knowledge at the same time. The brain has evolved to solve this problem in order to allow continuous learning of new knowledge in a stable fashion without catastrophic interference with previously learned knowledge. However, the precise biological mechanism and computational explanation for such stable learning remains unclear. So, we argue that it is important to explore the stability of neural learning models in order to gain insight into its computational properties.

A number of mathematical analyses have provided insights into the stability of learning in neuronal networks. For example, the error surface of a simple network that computes the XOR problem has been explored and it has been shown that the structure of the error surface is extremely lumpy, containing a number of regions with local minima and saddle points between regions [HAM98]. Although their analysis has demonstrated that it is theoretically possible to investigate the stability property of neuronal networks mathematically by studying the error surface, it is only practical for relatively simple networks, but not generalisable to more complex models, e.g. with recurrent connections between layers. Recently, mathematical analysis has also been applied to the stability of recurrent neural networks, resulting in several criteria under which a network is well-behaved [ARI03]. However, the stability of more general recurrent networks is not guaranteed. Studying the static structure of the neuronal network, i.e. the error surface, is not sufficient to determine the dynamic training trajectory of a specific learning algorithm. Moverover, analytical approaches cannot be automated and require expert mathematical skills. So, we argue that model checking may be a potentially useful tool to verify properties automatically, especially in the context of the more complex neuronal networks used in cognitive neuroscience. Demostrating the applicability of our neuroautomata—model checking approach to verifying the stability of such learning algorithms is a central objective of this paper.

In contrast, traditional techniques, such as simulations, are limited in their capacity to investigate stability properties, which may be characterised by a complex and dynamically changing pattern of activation over time. In particular, simulations cannot test the entire state space, hence they are limited in their capacity to guarantee that the learning is going to be stable and catastrophic instability cannot arise. Finally, most simulations stop training after a certain number of epochs or when a certain number of correct outputs have been obtained. Such stopping criteria do not ensure that the network is stable indefinitely. In addition, turning the learning algorithm off and testing the behaviour of the network with learning off is not a realistic general assumption when considering animal cognition. As previously discussed, a much more viable assumption is that learning is continuously active, thus realistic models of learning should be stable in the sense being considered here.

As a starting point, to explore the capability of model checking neuronal networks we concentrate on the XOR task, which is a linearly inseparable problem, and has great historical and theoretical significance [ORE00]. In addition, XOR can be viewed as a canonically difficult learning task. That is, although it is very simple in respect of number of patterns and number of neurons, its linear inseparability makes it characteristically difficult [SUT89]. In particular, this case study is not intended to illustrate how our approach responds to the problem

of scale, at least not in respect of number of neurons or number of synapses. Our second case study, c.f. Sect. 4, performs that role. Here we demostrate the approach's capacity to cope with the most difficult class of pattern classification problem, in respect of error surface complexity. Classification problems involving far more neurons and synapses, but simple error surfaces, are unlikely to provide more stringent tests of our model.

3.1. Specification of neuronal networks

3.1.1. Point neuron model

While we claim neurophysiological plausibility for the neuron model we used here, we have not gone as far as to employ a spiking model. Thus, we have remained within the rate coding domain, i.e. our neural units model the average rate at which a population of biological neurons spike. There are a number of reasons for this choice. Firstly, there are some that still argue that the rate coded level is a major, and some would say the major, "information bearing" level, e.g. [ORE00]. As a reflection of this, many of the central figures in computational cognitive neuroscience focus on rate coded frameworks, e.g. [FRI00a, FRI00b, FRI00c, BLY11, GRO12, NIE05, ROL02, ORE00]. Secondly, from a practical perspective, building a spiking model from the ground-up for the kind of applications we are exploring would be a major undertaking, and significantly more demanding than construction of our rate coded models.

The neurophysiologically more detailed neuron model used in this paper has the format shown in Fig. 3. Broadly speaking, the direction of computation is from bottom to top. Thus, ge (strictly the excitatory input conductance) is calculated first as a normalised sum of weighted inputs combined with a neuron specific bias (β_k) as shown in Eq. (1). Thus, ge is an aggregate measure of the excitatory input to the neuron. gl is a permanently active leak (strictly the conductance of the leak channel). The leak is always trying to pull the neuron down to resting levels ($Vm_k = 0.15$). gi is the inhibitory input (strictly the inhibitory input conductance). Thus, neurons that project in an inhibitory fashion to this neuron do so by adjusting gi, which also attempts to pull the neuron back to resting levels. The key quantity (which does not arise in classical artificial neural networks) is the membrane potential (Vm_k) as shown in Eq. (2). This reflects the extent to which the neuron is excited, having a range from 0 to 1. It is a purely internal measure, i.e. not directly observable externally. ge, gl and gi all act upon Vm_k. ge acts to push Vm_k up towards the excitatory reversal potential (Ee), which is set to 1. In contrast, gl and gi seek to push Vm_k down towards resting levels. This effect is obtained by setting the leak reversal potential (El) and the inhibitory reversal potential (Ei) to resting levels. With constant inputs, the membrane potential (Vm_k) settles at a level between the resting level and 1, at which the inhibitory and leak forces (pushing Vm_k down) and the excitatory force (pushing Vm_k up) are equal and opposite. When these forces are at balance, we can derive the equilibrium membrane potential (Eq. 3) from Eq. (2). Finally, the output activation (a_k) is directly calculated from Vm_k; in fact, it is a monotonically increasing continuous nonlinear function of Vm_k, as shown in Eq. (4). This nonlinearity reflects the thresholded (i.e. very little, if any, output at low levels of Vm_k) and the saturating (i.e. asymptotic progression to a maximum output at high levels of Vm_k) character of biological neurons. Finally, this is a rate coded model, thus a_k reflects the rate of neuron spiking, rather than individual spikes themselves. We will explain these equations throughout this section.

$$ge := f_1(predIds, a, w, k) = \frac{1}{|predIds|}\left[\sum_{pred \in predIds}(a_{pred} \times w_{pred,k}) + \beta_k\right] \tag{1}$$

$$\begin{aligned} Vm_k(t+1) &:= f_2(ge, Ee, gi, Ei, gl, El, Vm_k(t)) \\ &= Vm_k(t) + dt_{vm}[ge(Ee - Vm_k(t)) + gi(Ei - Vm_k(t)) + gl(El - Vm_k(t))] \end{aligned} \tag{2}$$

$$Vm_k := f_3(ge, Ee, gi, Ei, gl, El) = \frac{ge \times Ee + gi \times Ei + gl \times El}{ge + gi + gl} \tag{3}$$

$$a_k := f_4(\theta, Vm_k, \gamma) = \frac{1}{1 + e^{\gamma(\theta - Vm_k)}} \tag{4}$$

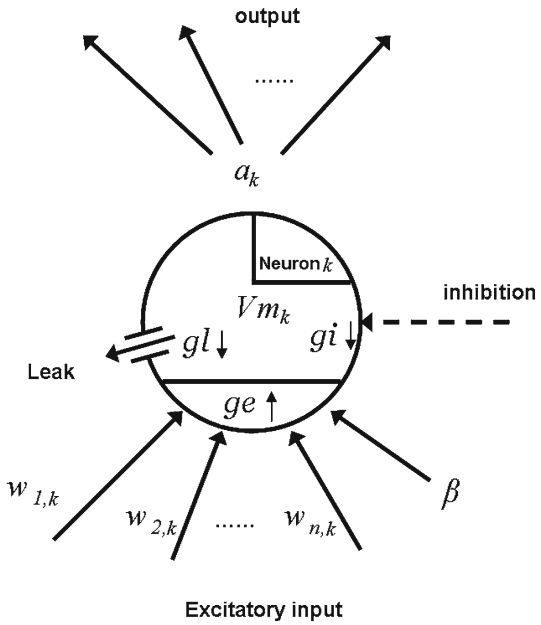

Fig. 3. A schematic diagram of a point neuron model

3.1.2. Neuron signature

As previously discussed, the DRCA model of neuronal networks contains a collection of neuron automata and an environment automaton, and signatures specify the data structures used in the model. A top-level signature is called the global declaration, which defines a class of constants for *all* neurons in the model, i.e. $G = \langle \theta, \gamma, \varepsilon, Ee, Ei, El, gl \rangle$, where $\theta : \mathbb{R}_0^1$ sets the firing threshold ($\mathbb{R}_0^1$ are real values in the range from 0 to 1), $\gamma : \mathbb{R}_0^1$ is the steepness parameter for the output activation function, and $\varepsilon : \mathbb{R}_0^1$ denotes the learning rate [ORE00]. $Ee, Ei, El : \mathbb{R}_0^1$ are reversal potentials of excitatory, inhibitory and leak channels respectively. (Note, these channels are ion channels in the cell membrane, and should not be confused with communication channels in the automata models.) $gl : \mathbb{R}_0^1$ is the leak current, which is also a constant in our model.

The neuron signature is defined as $NEURON[C, V, E]$, where $C = \langle predIds, succIds \rangle$ is a tuple of constants that defines the connectivity of this neuron with other neurons. With respect to the current neuron, $predIds : \mathbb{P}(Id)$ is the set of predecessor neuron identities, $succIds : \mathbb{P}(Id)$ is the set of successor neuron identities, and $\mathbb{P}(Id)$ denotes the power set of Id. In general, values in these sets ($predIds$ and $succIds$) vary among neurons in different layers, but the connectivity does not change during learning. $V = \langle k, a, w, ge, gi \rangle$ is a set of variables of the neuron. $k : Id$ is the identity of this neuron automaton, and it is assumed to be unique. $a : Id \to \mathbb{R}_0^1$ is a function, and its domain is $predIds \cup \{k\}$. For example, a_{pred} would denote the predecessor neuron activation with respect to a predecessor neuron identity $pred \in predIds$, and a_k is the activation of the current neuron. $w : Id \times Id \to \mathbb{R}_0^1$ is also a function, i.e. $w_{pred,k}$ denotes the weight from a predecessor neuron with identity $pred$ to the current neuron (identified by k). $ge, gi : \mathbb{R}_0^1$ denote the excitatory and inhibitory inputs respectively (also called input conductances, as previously discussed) [ORE00]. The other variables, which are defined to be in E do not relate to neurobiology. They are specific to our particular implementation, i.e. $E = \langle tempIds, pred, succ \rangle$, where $tempIds : \mathbb{P}(Id)$ is used to temporarily record unprocessed predecessor or successor neuron identities. $pred, succ : Id$ index the identities of specific predecessor and successor neuron identities respectively.

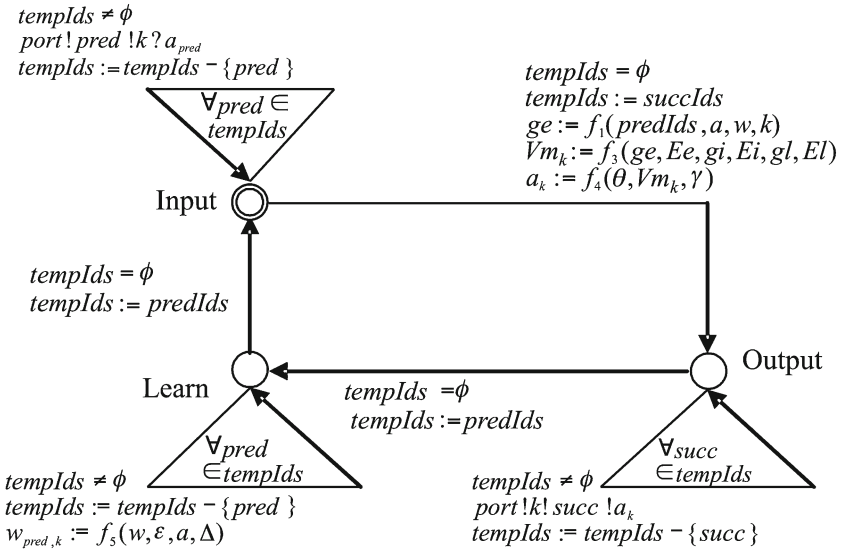

Fig. 4. A communicating automata model of a single neuron. The initial location is *input*, where predecessor neural activations are received. The net input and current neural activation is evaluated at the transition between *input* and *output*, where the activation is sent to all successor neurons. In the final location, *learn*, weights are updated

3.1.3. Behavioural specification

The behavioural specification defines the internal structure of neurons. In each location, neuron automata sit waiting to perform transitions, which specify how neural information is processed and distributed. For example, dynamic update of neural activations and weights is specified in a set of equations in the transition relationship between locations; see Fig. 4.

We consider learning in this case study, so each neuron automaton has three locations: *Input*, *Output* and *Learn*. In the initial location, *Input*, the neuron retrieves all inputs from its input channels, each of which has the form: $port!pred!k?a_{pred}$, where *port* is the label of a communication channel. The expression $!pred!k?a_{pred}$ is a data passing declaration, which identifies the nature of the communication to be made [BOW06]. (This is the same multiway communication as found in LOTOS [BOL88] and CSP [ROS98].) That is, working from left to right, the first declaration denotes the predecessor neuron identity with respect to this communication, the second denotes the current neuron identity and the third receives the activation value transmitted over the link. In this expression, $!pred$ and $!k$ are value declarations, which enforce a value upon communication, and $?a_{pred}$ is a variable declaration, which receives a value upon communication [BOW06]. To be more specific, this notation offers a very general notation for performing multiway synchronisations, which can take place if all communication parties (which could be more than two) impose compatible constraints at data passing declarations. Specifically, a value declaration (e.g. $!pred$) enforces a value upon communication at that position, while a variable declaration (e.g. $?a_{pred}$) does not constrain the value communicated at that position (although types have to match). For example, three action offers of the form, $aaa!5!x$, $aaa!y?z$ and $aaa!w!2$ could synchronize to perform an action aaa_5_2, as long as, $w = y = 5$ and $x = 2$, when that action is performed, and as a result of performing the action, z will became equal to 2.

The effect associated with the *port* action at the *Input* location is an assignment: $tempIds := tempIds - \{pred\}$, which indicates the completion of retrieving the input from the predecessor neuron with identity $pred$; thus this identity can be removed from the set *tempIds*. Once the neuron has received all its inputs, the guard $tempIds = \phi$ will hold. Then, the neuron evaluates the excitatory input *ge*, the membrane potential Vm_k and the activation a_k using Eqs. 3 and 4 respectively. In the context of this case study, no inhibition is applied to the neuronal network, so *gi* is set to zero.

In the excitatory input function f_1, *predIds* denotes the set of predecessor neurons. The *ge* is calculated as a weighed sum of all its input activations plus a bias weight β_k normalised by the total number of inputs [ORE00]. For any set S, $\mid S \mid$ denotes the cardinality of S. The function f_3 defines the equilibrium membrane potential, this is the value that the membrane potential will stablise at with constant inputs [ORE00]. (Note, the nonstationary membrane potential rule, Eq. (2), will be demonstrated in the second case study, see Sect. 4.) The membrane

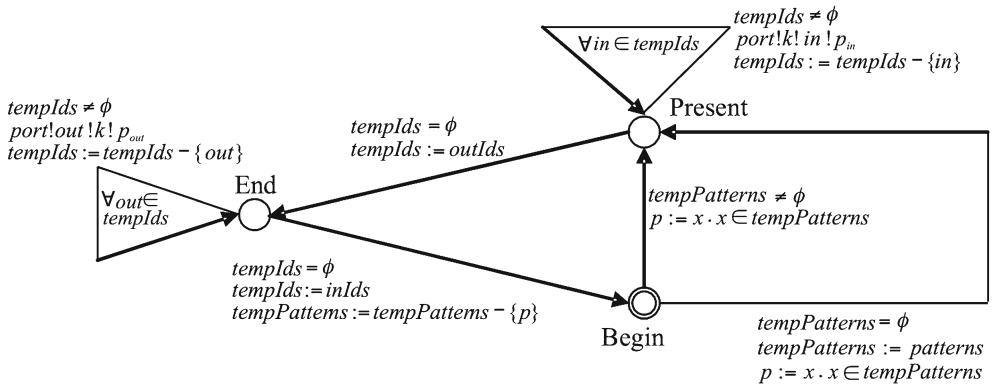

Fig. 5. Model of the environment. The outgoing transitions at the initial location, *begin*, determine when an epoch finishes. The next location, *present*, sends activations to all input neurons. Then, output activations of the neuronal networks are received at the final location, *end*

potential is the critical *internal* measure of the extent to which the neuron is excited; neurophysiologically it corresponds to the charge disparity across the neuron's membrane. The neural activation is computed using function f_4, which is a sigmoid nonlinear activation function. This function is biologically realistic in the sense that it captures the nonlinearity of biological neurons, i.e. a graded threshold at low membrane potential values and progressive saturation to a maximum firing rate at high membrane potentials. Moreover, this function is differentiable, allowing the implementation of BP [ORE00].

In the location, *Output*, the neuron sends its activation by communicating on $port!k!succ!a_k$. To tie this with inputs, the first two value declarations are used to achieve value synchronisation, i.e. communication is only allowed between $port!p!q!a$ in the sending neuron and $port!p!q?v$ in the receiving neuron, where p, q and a are the values of corresponding variables. The result is the sending neuron's activation a being bound to v. In the last location, *Learn*, weights are changed according to learning algorithms defined as function f_5, which will be defined in Sects. 3.2.1 and 3.2.2.

3.1.4. Environment

In general, the environment presents patterns of input activation to the neuronal network, and receives output activation from the network. The signature of environment is $ENVIRONMENT[C, V, E]$, where $C = \langle inIds, outIds\rangle$, $V = \langle k, p, q\rangle$ and $E = \langle in, out, tempIds, patterns, tempPatterns\rangle$. In the set C, vectors $inIds$, $outIds : \mathbb{P}(Id)$ are sets of input and output neurons relative to the entire neuronal network. In V, $k : Id$ denotes the identity of the environment automaton, $p : Pattern$ denotes the current pattern being presented to the neuronal network, and $q : Pattern$ is a dummy variable that holds the output of neuronal networks. In E, in, $out : Id$ index the identities of specific input or output neuron identities respectively. $patterns : \mathbb{P}(Pattern)$ specify the training data, and $Pattern : Id \to \mathbb{R}_0^1$ is a function from neuron identities (e.g. input or output neuron identities) to their activations. For supervised learning algorithms, e.g. BP learning, patterns of activation for both input and output neurons are specified in this function. For unsupervised learning, only the patterns of activation for input neurons would be specified. in, out, $tempIds$ and $tempPatterns$ are variables for temporally storing the neuron identities or patterns.

As shown in Fig. 5, when learning starts, the environment is at the *Begin* location. Initially, *tempPatterns* is assigned to all training patterns, and *tempIds* is set to all input neuron identities. If *tempPatterns* is not empty, a randomly chosen element in *tempPatterns* is assigned to p, which denotes the current pattern, i.e. $p := x \cdot x \in tempPatterns$. Otherwise, *tempPatterns* is reset before the current pattern is assigned. Next, the environment automaton moves to the *Present* location, where it sends activations to all input neurons. Then, it moves to the *End* location, where it receives outputs from all output neurons. Finally, it returns to the *Begin* location after removing the current pattern from the set *tempPatterns*. This ensures that the same pattern is not presented twice in each epoch. As previously discussed, at the end of each epoch, *tempPatterns* is empty, so is reset before the next epoch starts. Please note that the environment will depend on the application and learning algorithms used, so, additional modifications may be required. For example, in the case of supervised learning, instead of receiving

output activations, the environment may send expected activations to the output neurons. Also, when there is recurrent connections in the network, it may specify criteria for when activations are settled in order to begin the subsequent epoch, see Sect. 3.2.1.

3.2. Modelling neuronal learning algorithms

In this section, we model two well-known supervised neuronal learning algorithms: the (BP) and the (GeneRec) algorithms. Using a similar modelling framework, unsupervised learning algorithms can also be implemented using communicating automata models. However, they are outside the scope of this paper. The structure of the neural networks in this section is based on the model shown in Fig. 1 for all our learning algorithms, however, the equations for the learning rules are different among different models.

3.2.1. Backpropagation (BP) learning

The BP learning algorithm contains the following stages: first, activations of input layer neurons are determined by input patterns defined in the environment automaton. Then, activations of each layer are propagated forward to the next layer until the output layer. Second, the error terms Δ_k in the output layer are computed, and then these errors are propagated back to previous layers. Additional channels are defined between hidden and output layers in order to pass error signals backwards (not depicted). The error associated with neuron k is evaluated using gradient descent [ORE00], and weights are adjusted in the neuron automata using the following equation:

$$w_{pred,k} := f_5(w, \varepsilon, a, \Delta) = w_{pred,k} + \varepsilon \times a_{pred} \times \Delta_k \tag{5}$$

where

$$\Delta_k = \begin{cases} p_k - a_k & \text{if neuron } k \text{ is an output neuron} \\ a_k(1 - a_k) \sum_{succ} w_{k,succ} \times \Delta_{succ} & \text{if neuron } k \text{ is a hidden neuron} \end{cases}$$

In Eq. (5), $a_k(1 - a_k)$ is the derivative of the sigmoid function [ORE00], $succ$ ranges over neurons in the successor layer and $pred$ is the relevant predecessor neuron as specified in f_5's usage in the *Learn* location in Figure 4. When implemented in recurrent networks, the basic equations of BP learning do not change. Note, the neural activations in recurrent networks normally take several iterations to converge, so the BP learning algorithm (and GeneRec learning in the next section) are only applied after the activation in the entire network has completely settled. We say the activation of the network has settled in the current state whenever the difference between the output activations in the current and previous iterations are less than a given small value (e.g. <0.00001). In the rest of the paper, we use BPRec to denote the BP learning algorithm in recurrent networks.

3.2.2. The generalized recirculation algorithm

The Generalized Recirculation algorithm was introduced by O'Reilly, as an alternative to BP learning [ORE96]. This type of learning is promising due to its biological plausibility [ORE96]. Although BP learning is a powerful and widely used supervised learning algorithm, it suffers from biological implausibility. The essential problem is that BP learning requires the BP of error signals from the dendrite of the post-synaptic neuron to the axon of the pre-synaptic neuron via synapses. However, there is no biological mechanism that performs such a transmission. O'Reilly has though argued that GeneRec learning can approximate BP learning using only local activations and avoiding the problematic error signals, hence, it is a more biologically plausible learning mechanism [ORE96]. The GeneRec algorithm requires bi-directional propagation of neural activations (i.e. a recurrent neuronal network). In addition, the learning algorithm includes two activation phases: the *minus* and *plus* phases [ORE96]. In the minus phase, the input (layer) activations drive the network's output, i.e. the actual activations at the output layer. This phase is the same as the forward propagation of activation in the recurrent version of BP learning. Activations in this phase are denoted as a_k^-. In the plus phase, the outputs of the network are externally clamped

to the expected outcomes. So, both the input and output activations drive the network [ORE00]. Activations in this phase are denoted as a_k^+.

It has been argued that the difference between the two stages of activation, $a_k^+ - a_k^-$, is a justifiable approximation/alternative to the error signals used in BP learning [ORE96, ORE00]. So, GeneRec learning is defined similarly to BP learning: $w_{pred,k} := f_5(w, \varepsilon, a, \Delta)$, except for the computation of the error term, which is set according to $\Delta_k = a_k^+ - a_k^-$, where k can be either a hidden or output neuron. It can be seen that although the weight update equation for GeneRec learning is the same as for BP learning, it is only based on local variables, i.e. neural activation on the plus and the minus phases.

3.3. Formal specification and verification with SPIN

3.3.1. SPIN model checker

In the context of model checking, there are two levels of specification: the requirements specification and the system specification, which describe a system at two different levels of abstraction. A requirements specification states what the system is intended to achieve and the attributes or features the system must possess. The requirements language used in this case study is Linear Time Temporal Logic (LTL) [PNU77]. The language consists of state formulae, which are built upon references to locations and variables in the model, combined with the usual Boolean and relational operators. In addition, temporal formulae express temporal properties, which cross states. The property $\Box\varphi$ requires that all reachable states satisfy the formula φ and $\Diamond\varphi$ requires that at least one reachable state satisfies φ (on all paths from the current state). In this paper, the system specification defines the internal structure of the biological system and the interaction between different components within the system using DRCA.

A model checker automatically assesses the relationship between the model (i.e. the system specification) and the requirements, and then outputs whether the model satisfies the set of requirements. In addition, the model checker can generate traces of witnesses/counter-examples. A witness example shows a trace where the property holds, and the counter-example shows a trace where the property does not hold. We translated the DRCA models of neuronal networks to PROMELA, the language of SPIN, in order to benefit from SPIN's modelling facilities and efficient verification algorithms. Spin is a well-known model checker for reactive distributed systems, which has been successfully applied to many non-trivial problems [HOL03]. In this paper, we use SPIN [HOL03] to automatically verify a number of behavioral properties of the neural learning algorithms.

The SPIN system language, PROMELA, is a process oriented language that allows distributed systems to be modelled as a collection of parallel processes communicating via synchronization channels and shared variables. Although our communicating automata models of learning algorithms can be straightforwardly expressed in PROMELA, we have instead modelled each algorithm as a single PROMELA process. This "serialization" of the naturally concurrent behavior of the neuronal networks is common in the majority of cognitive modelling and is possible due to the independence of activations among neurons of the same layer and the fixed order of activation update among layers, which make the networks fully deterministic. More formally, one can prove that the behavior of the neuronal networks that model each learning algorithm (BP, GeneRec and BPRec) is independent of the order in which the neurons in the same layer update their activation values (furthermore, the behavior is also equivalent to simultaneous updates of two or more neurons in the same layer), provided the activations of the layers occur in the order specified by each algorithm (e.g., in BP learning, the activations of the input layer are propagated to the hidden layer and then to the output layer, and then the errors are propagated backwards from the output layer through the hidden layer to the input layer). However, in the second case study, we will present an asynchronous (nonserialised) model in Uppaal.

3.3.2. LTL properties

The SPIN requirements language is LTL [PNU77], and we verified the following LTL properties for each learning algorithm. (a) "Stability", meaning that the system eventually reaches a state where the output activations are sufficiently close to the expected outputs and remain thereafter in states where activations are sufficiently close to expected values. This was expressed by the LTL formula, $\Diamond\Box$ `SUCCESS`, where `SUCCESS` is a global variable in the PROMELA model (of each algorithm) that holds when the difference between the output neuron activations and

the output pattern is less than a given error (typically 0.5). (b) "Recurrence", meaning that from every reachable state, the algorithm eventually reaches a SUCCESS-state (i.e., a state where SUCCESS holds). This was expressed by the formula $\Box\Diamond$ SUCCESS; thus, even if permanent (i.e. henceforth) SUCCESS is not reached, it is always the case that a SUCCESS state will eventually occur. (c) "Eventuality", meaning that, from the start state, the algorithm eventually reaches a SUCCESS-state although it may not remain in such a state. This was expressed by the formula $\Diamond$ SUCCESS. We first verified stability, then recurrence and, finally, eventuality; recurrence was verified only in the case that stability did not hold, and eventuality was verified only in the case that neither stability nor recurrence held. This strategy follows since $\mathcal{M}(stability) \subset \mathcal{M}(recurrence) \subset \mathcal{M}(eventuality)$, where $\mathcal{M}$ yields the models of an LTL formula.

We were mainly interested in the stability property as it expresses successful learning; the remaining properties refined our knowledge of the algorithm's behavior under a given set of initial weights, when it failed to learn in a stable fashion. In cognitive modelling, little attention has been paid to stability properties of learning. In particular, stability in its simplest form, i.e. maintaining obtained solutions, while continuing to learn the same task, has not been explored. We argue that it can serve as a baseline for stability in more complex situations, i.e. learning different tasks without catastrophic forgetting, which has been more extensively explored. This is because, if a learning algorithm fails to be stable in this simple sense (i.e. self stable), then it certainly will not be stable in a more general sense. The other two LTL properties are only tested if the stability property fails. The eventuality property checks if the algorithm can learn the task at all, so it is the minimum requirement for any model. The recurrence property sits somewhere in the middle between the stability and the eventuality properties, because it does not require the system to converge to a permanently stable state. Instead, it allows the system to unlearn the task as long as it can *always* re-learn. We argue that this situation is not stable in an absolute sense, but it could be acceptable for specific contexts or applications.

The PROMELA process repeatedly updates the current state an epoch at a time (there is no need to observe intermediate states, since the properties to verify depend on the state variable SUCCESS, which is calculated as a function of the activations of the output layer for each pattern presented in the current epoch). The process is implemented in embedded C code, which allowed us to represent the real-valued weights as floating point variables and to implement activation functions by C library (math) functions. State-explosion problems were avoided by rounding the weights to five decimal digits. As a result of using C code, verification must be interpreted modulo precision and termination (and thus, exhaustive exploration of the state-space cannot be guaranteed, in general). However, we remark that five decimal digits sufficed to perform a fine grain analysis of the learning algorithms, and that SPIN terminated in all cases.

The process sets initial weights, patterns, and parameters of the neuron activation functions to a given set of values. A Java script was used to batch run the SPIN model checker and verify the properties on different random initial weights.

3.3.3. An example of the PROMELA model

By way of example, we give below the pseudo-code of the GeneRec learning algorithm. We consider a neuronal network with a single output neuron. For each $p \in Patterns$ and $k \in Ids$, p_k denotes the expected activation value associated with neuron k. For $n \in \mathbb{R}$, $\mid n \mid$ denotes the absolute value; $\mid n \mid^{+}=$ if $n \geq 0$ then n else 0; and $\mid n \mid^{-}=$ if $n < 0$ then n else 0. To be consistent with the DRCA model, we use $\mathbb{R}$ to denote the reals and $\mathbb{B}$ to denote the Booleans.

The algorithm is implemented as a non-terminating loop, where each iteration corresponds to an epoch. In turn, each epoch proceeds by presenting all patterns to the network. For each pattern, the algorithm first copies the input values as initial activations of input neurons (line 4), and then enters the minus phase in which only input layer neuron activations are set. The loop terminates when activations are settled (line 7). Hidden neuron activations in the minus phase are stored for later use. Then, the process enters the plus phase as previously discussed, and neural activations in both input and output layers are set externally. Finally, GeneRec learning is applied to all layers (lines 18 and 19). The Boolean variable SUCCESS tells whether the difference between the actual and expected activation values of the output neurons in the current epoch are below a certain threshold τ (line 15). In the PROMELA model, each epoch (i.e., the sequence of actions inside the non-terminating outermost loop) is implemented as an atomic operation, and the state is defined by the weights (w), β biases and the SUCCESS variable. All other variables are reset at the start of each epoch and therefore need not be tracked by the state descriptor.

All variable names in the pseudo-code are matched to the DRCA model as closely as possible. Function names used in the pseudo-code are different from the previous sections, however, they can be simply mapped to the arithmetic expressions used previously. We also applied a form of *soft weight bounding* [ORE00] during learning, see lines 1 and 2 of learnNeuron. This ensures that the weight changes reduce exponentially as the bounds (0 or 1) are approached. Moreover, we assume weight symmetry in both BPRec and GeneRec learning as suggested by [ORE00]. The soft weight bounding and symmetry assumptions are justified by both computational requirements and neurobiological evidence [ORE00].

parameters: w is the set of weights,
a is the set of activation values,
β is the set of biases,
Patterns is the set of patterns,
Ids is the set of neuron identities,
$InputIds \subseteq Ids$ is the set of identities of input neurons,
$HiddenIds \subseteq Ids$ is the set of identities of hidden neurons,
$PredIds_k \subseteq Ids$ is the set of predecessor neurons of neuron k, for each $k \in Ids$,
$SuccIds_k \subseteq Ids$ is the set of successor neurons of neuron k, for each $k \in Ids$,
o is the identity of the output neuron,
$\tau, \gamma, \theta, \varepsilon, Ee, Ei, El, gi, gl$ are various parameters
local data : $d_h(h \in HiddenIds)$, d_o, $q \in \mathbb{R}$; settled, SUCCESS $\in \mathbb{B}$

```
1  while true do
2      SUCCESS := true;
3      foreach p ∈ Patterns do
4          foreach i ∈ InputIds do a_i := p_i;
5          settled := false;
6          q := 0.0;
7          while ! settled do
8              foreach h ∈ HiddenIds do fireHiddenNeuron(h,w,a,β,PredIds_h,SuccIds_h,q);
9              fireOutputNeuron(o,w,a,β,PredIds_o);
10             settled := q = a_o;
11             q := a_o;
12         end
13         foreach h ∈ HiddenIds do d_h := a_h;
14         d_o := a_o;
15         SUCCESS := SUCCESS and | a_o − p_o |< τ;
16         a_o := p_o;
17         foreach h ∈ HiddenIds do fireHiddenNeuron(h,w,a,β,PredIds_h,SuccIds_h,a_o);
18         learnNeuron(o,w,a,β, PredIds_o,d_o);
19         foreach h ∈ HiddenIds do
20             learnNeuron(h,w,a,β,PredIds_h,d_h);
21             w_{o,h} := w_{h,o};
22         end
23     end
24 end
```

Algorithm 1: The GeneRec algorithm

3.3.4. Results of SPIN model checking

We model checked three different neuronal networks: standard BP learning, BP learning in recurrent networks (BPRec) and GeneRec learning, which increasingly employ more biological detail. In this respect, BP learning is the least biologically plausible model. BPRec learning is more biologically plausible than standard BP learning, because it captures the fact that bidirectional connections arise widely in the cortex. In addition, it has a number of attractive properties that support human cognition [ORE00]. GeneRec learning is the most biologically plausible model examined here. In particular, it is proposed to provide the computational power of standard supervised learning, while only using realistic local information.

parameters: $k \in Ids$ is this neuron's id,
$I \subseteq Ids$ is the set of this neuron's predecessor neurons,
$J \subseteq Ids$ is the set of this neuron's successor neurons,
w is the set of weights,
a is the set of activation values,
β is the set of biases.
local data : ge, $Vm \in \mathbb{R}$, $q \in \mathbb{R}$ is the activation of the output neuron

1 $ge := \frac{1}{|I|+|J|}(\Sigma_{i \in I}(a_i \times w_{i,k}) + \Sigma_{j \in J}(q \times w_{j,k}) + \beta_k)$;
2 $Vm := \frac{geEe+giEi+glEl}{ge+gi+gl}$;
3 $a_k := \frac{1}{1+e^{\gamma(\theta - Vm)}}$;

Procedure fireHiddenNeuron(k,w,a,β,I,J,q)

parameters: $k \in Ids$ is this neuron's id,
w is the set of weights,
a is the set of activation values,
β is the set of biases,
$I \subseteq Ids$ is the set of this neuron's predecessor neurons
local data : ge, $Vm \in \mathbb{R}$

1 $ge := \frac{1}{|I|}(\Sigma_{i \in I}(a_i \times w_{i,k}) + \beta_k)$;
2 $Vm := \frac{geEe+giEi+glEl}{ge+gi+gl}$;
3 $a_k := \frac{1}{1+e^{\gamma(\theta - Vm)}}$;

Procedure fireOutputNeuron(k,w,a,β,I)

Model checking was performed 500 times for each learning algorithm from different initial weights randomly sampled from five uniform distributions (0–0.2, 0.2–0.4, 0.4–0.6, 0.6–0.8 and 0.8–1). The parameters used in these models are shown in Table 1, and they are the same across all three learning algorithms. These parameters are inherited from [ORE00]. This is because our objective is to assess the GeneRec approach presented in [ORE00]. The results of model checking are shown in Table 2. It can be seen that standard (feedforward) BP learning learnt the XOR task given small to medium random initial weights, i.e. between 0 and 0.6. Moreover, feedforward BP learning was always stable in terms of the LTL property $\Diamond\Box$ `SUCCESS`. However, it did not learn the XOR task when the initial weights were large, e.g. above 0.6. This suggests that standard BP learning is sensitive to the initial weights. Using Monte Carlo techniques, other researchers [KOL91] have discovered that BP learning converges within 50,000 epochs given relatively small initial weights ($\leq$ 0.5). They called this property *t-convergency*, i.e. learning converges within *t* epochs. They argued that failing to converge when the initial weights are large is caused by BP learning's chaotic error surface and nonlinearity. Our model checking not only replicated their findings, i.e. the sensitivity of the initial weights, but also explored the stability properties, which cannot be guaranteed by testing t-convergency since they require the exploration of the entire state space. As standard BP learning is an extensively studied algorithm, in the rest of this section, we concentrate on the relatively under-explored relationship between BPRec and GeneRec learning methods.

parameters: $k \in Ids$ is this neuron's id,
w is the set of weights,
a is the set of activation values,
β is the set of biases,
$I \subseteq Ids$ is the set of this neuron's predecessor neurons,
$d_k \in \mathbb{R}$ is this neuron's activation value in a previous phase

1 **foreach** $i \in I$ **do** $w_{i,k} := w_{i,k} + \epsilon(|\ a_k - d_k\ |^+ (1 - w_{i,k}) + |\ a_k - d_k\ |^- (w_{i,k}))a_i$;
2 $\beta_k := \beta_k + \epsilon(|\ a_k - d_k\ |^+ (1 - \beta_k) + |\ a_k - d_k\ |^- (1 + \beta_k))$;

Procedure learnNeuron(k,w,a,β,I,d_k)

Table 1. Parameters used in all three learning models are similar to those specified in O'Reilly and Munakata [ORE00]

	Model parameters
Learning rate	$\varepsilon = 0.01$
Firing threshold	$\theta = 0.32$
Steepness of the sigmoid	$\gamma = 50$
Excitatory reversal potentials	$Ee = 1$
Inhibitory reversal potentials	$Ei = 0.15$
Leak reversal potentials	$El = 0.15$
Inhibitory input currents	$gi = 0$ (note, in this model we do not model inhibitory inputs)
Leak current	$gl = 2.8$

Table 2. Comparison between different learning algorithms using model checking

$\gamma = 50$	Initial weights	Stability (%)	Recurrence (%)	Eventuality (%)
BP	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	0	0	0
	0.8–1	0	0	0
BPRec	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100
GeneRec	0–0.2	17	100	100
	0.2–0.4	0	100	100
	0.4–0.6	0	100	100
	0.6–0.8	0	100	100
	0.8–1	0	100	100

The percentages represent the proportion of initial weight settings satisfying each property

As shown in Table 2, BP learning with both feedforward and feedback connections (i.e. BPRec) performs well with all the initial weights tested. It not only successfully solved the XOR problem, but also learnt in a stable fashion. So, we argue that feedback activation alone is not the factor causing unstable learning. Unlike its feedforward counterpart, BPRec was able to solve the XOR task even when the initial weights were set to very large values.

GeneRec learning showed a different pattern to the other two learning algorithms. When the initial weights were set to small values between 0 and 0.2, GeneRec was only able to stably learn the XOR problem in 17 % of runs. For larger initial weights ($\geq$0.2), although GeneRec can learn the task, model checking has shown that it is not stable for these initial weights. In all cases, GeneRec satisfies the recurrence property, but not the permanent stability property, i.e. the network always unlearns the XOR problem, and then re-learns it afterwards. Bakker et al. [BAK00] have discovered a very similar behaviour. They argued that recurrent neuronal networks might show an arbitrarily unstable jumpy character during learning, and it is likely to be since recurrent networks have the character of chaotic attractor systems and are thus very sensitive to noise [BAK00]. However, our results suggest that unstable characteristics could be found without introducing noise. Moreover, the instability of learning is unlikely to be caused by recurrent connections alone. This is shown by the contrast between the GeneRec and BPRec learning algorithms, which both have bi-directional connections. In the next section, we will discuss several potential causes of the unstable character of GeneRec learning.

3.3.5. Potential causes of instability

One potential cause of instability is the characteristics of learning tasks. In order to test this, we have model checked these three learning algorithms with a simple OR task. The results show that all learning methods are stable for all initial weights ranging from 0 to 1. Hence, the previously observed unstable behaviour during learning may be caused by different patterns within the XOR task imposing conflicting requirements on the weight space (which

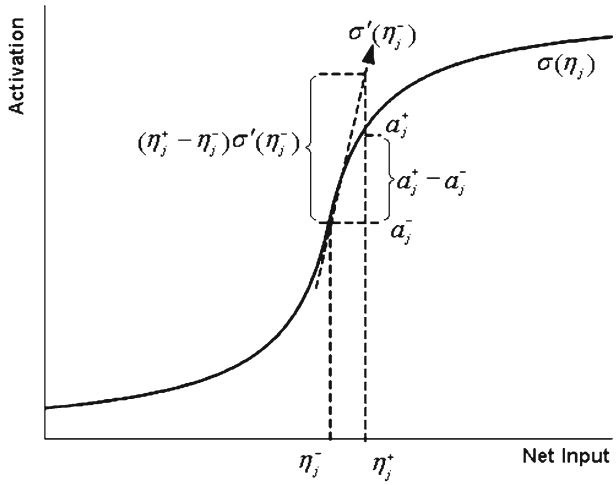

Fig. 6. The difference in activation approximates the difference in net input times the derivative of the sigmoid activation function. Adapted from O'Reilly and Munakata [ORE00]

is not the case in the OR task). This is a particular issue in linearly inseparable problems, i.e. a pattern is likely to change the weights in an opposite direction to the previous one. Hence, the error surface tends to be convoluted and lumpy. Moreover, both BP and GeneRec learning algorithms use a gradient descent technique, in which the weight changes are strongly related to the nonlinearity of the activation function [ORE00]. Importantly, the same activation function is used for both BP and GeneRec learning, despite them showing significantly different stability properties. In order to analysis the unstable weight changes in GeneRec learning, we have investigated how the activation function could cause the difference between BP and GeneRec learning.

As argued by [ORE96, ORE00], GeneRec is an approximation of BP learning. This is because the difference of the two stage activations, $a^+ - a^-$, approximates the error signal $(\eta^+ - \eta^-)\sigma'(\eta^-)$, which is equivalent to the error term used in BP learning, where $\eta = Vm - \theta$, corresponds to the net input used in classic artificial neural networks, σ' is the derivative of the activation function (i.e. $da/d\eta = a \times (1 - a)$) and $\eta^+ - \eta^-$ equals the term $\sum_{succ} w_{k,succ} \times \Delta_{succ}$ used in BP learning, given weight symmetry. The validity of such an approximation is claimed by O'Reilly et al. [ORE00], with respect to a depiction such as is shown in Fig. 6, "where it should be clear that differences in the Y axis are approximately equal to differences in the X axis times the slope of the function that maps X to Y". This approximation is perfect when the activation is linear and may fail under the following situations.

Firstly, the approximation worsens as the activation function becomes more nonlinear. To demonstrate this, we repeated the above model checking using different steepness parameters ($\gamma = 10$ and 25) of the activation function. The activation functions for these three different steepness parameter values are plotted in Fig. 7. The results are shown in Table 3. Comparing the results shown in Tables 2 and 3, we can see that the steepness of the activation function affects the stability of GeneRec learning. (Note, BP learning also improves when the activation function becomes more linear.) However, the increase in stability is more dramatic in GeneRec (where in Table 3 all weight settings yield 100 % stability) than in BP. This is, though, not surprising, since the central role played by recurrence and attractor dynamics in general is likely to make the algorithm acutely sensitive to parameter setting, particularly those that reduce the "lumpiness" of the error surface.

Table 3. Comparison between different learning algorithms using different activation functions

	Initial weights	Stability (%)	Recurrence (%)	Eventuality (%)
$\gamma = 25$				
BP	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	21	21	26
	0.8–1	0	0	0
BPRec	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100
GeneRec	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100
$\gamma = 10$				
BP	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100
BPRec	0–0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100
GeneRec	0-0.2	100	100	100
	0.2–0.4	100	100	100
	0.4–0.6	100	100	100
	0.6–0.8	100	100	100
	0.8–1	100	100	100

The percentages represent the proportion of initial weight settings satisfying each property

Secondly, the approximation worsens as $\mid \eta^+ - \eta^- \mid$ increases, since $\sigma'(\eta^-)$ generally becomes less reflective of $\sigma'(\eta^+)$. And finally, the approximation worsens as η^+ and η^- straddle inflection points. These situations can be characterised by the rate at which the rate of weight change is changing, i.e. the second derivative of the activation function. To illustrate this, the first and second derivatives ($\sigma'(\eta)$ and $\sigma''(\eta)$ respectively) of the sigmoid function are plotted in Fig. 8. From these plots, we can divide the sigmoid activation function into (1) a centre, near-linear region, (2) two highly nonlinear regions in which the absolute value of $\sigma''(\eta)$ is large, and (3) two asymptotic regions where the activation tends toward 0 or 1. The approximation is poor when either the absolute value of $\sigma''(\eta^+)$ or $\sigma''(\eta^-)$ is large and η^+ and η^- are, at least somewhat, different, in particular when η^+ and η^- are on different sides of the peak (or trough) of the $\sigma''(\eta)$ curve. In other words, the approximation is poor when the third derivative of plus and minus phases has different signs, i.e. $sign(\sigma'''(\eta^+)) \neq sign(\sigma'''(\eta^-))$.

On the one hand, if learning starts with activation values near 0.5, and generally pushes activations away from 0.5 (which is the case for most learning algorithms), GeneRec underestimates the magnitude of error and, thus, the size of weight change, during the highly nonlinear regions of the activation function. Thus, it is less likely to commit weights into asymptotic regions. On the other hand, GeneRec will overestimate the error when returning to the centre region of the activation function, thus making exaggerated moves. Both situations lead to a tendency not to commit weights and stabilise to asymptotic regions of the activation function.[2] As shown in Table 3, given more linear activation functions, i.e. small values of the steepness parameter, GeneRec learning is as stable as the recurrent version of BP learning (i.e. BPRec).

[2] Generally speaking, once weights are extreme enough, net inputs become extreme and the absolute value of the derivative term in BP learning becomes small, weight changes become small and weights stabilise.

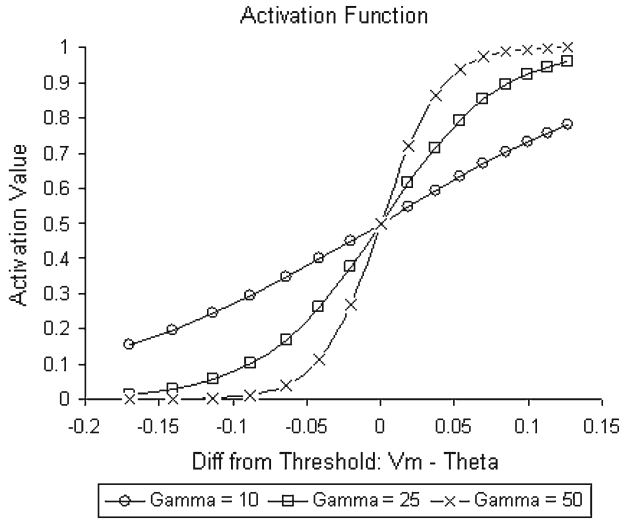

Fig. 7. The different sigmoid activation functions using $\gamma = 10, 25$ and 50

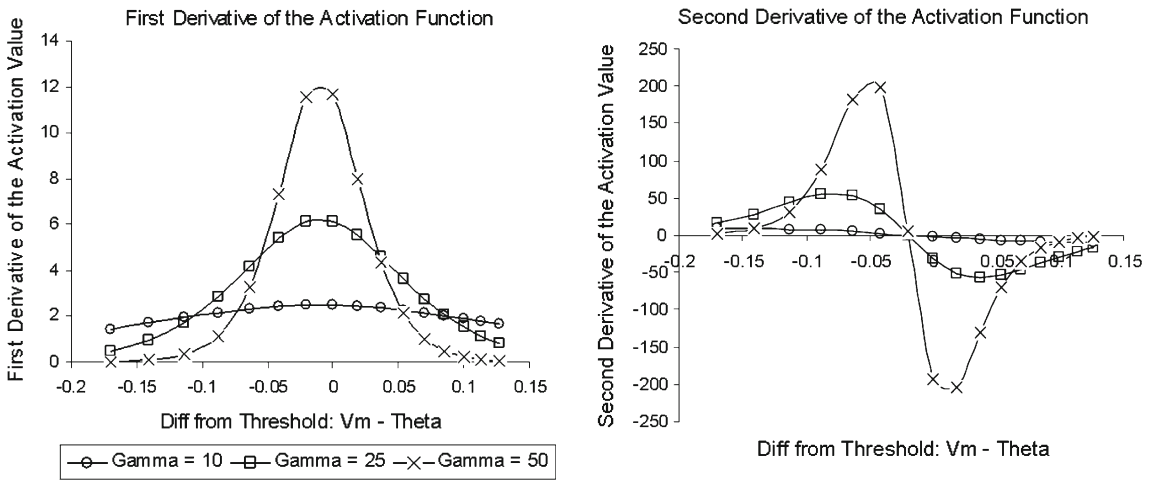

Fig. 8. The first and second derivatives of different sigmoid activation functions using $\gamma = 10, 25$ and 50

4. Case study 2: visual attention

This case study illustrates a number of further aspects of the proposed approach. Specifically, it allows us to demonstrate the following.

1. **Fine grained update**. Case study 1 used the equilibrium membrane potential equation (see Eq. 3), which is a special case of the full neuron update equation. This special case obtains when the derivative of the membrane update is zero, i.e. update has stabilised. However, the neural dynamics on the path to stability or, indeed, those arising in non-stationary contexts is of great importance. Thus, we explore verification in the context of fine grain neural update, using the full (Hodgkin–Huxley inspired) membrane potential update, see Eq. (2).

2. **Temporal responsiveness**. Clearly, the speed with which the brain responds to, say, stimulation is a critical facet of its effectiveness. Thus, being able to verify that a neural model attains a particular state within a particular period of time is of great potential importance. We thus illustrate model checking, using Uppaal, of timing constraints in neuron automata specifications.
3. **Asynchronous update**. Neural modelling in cognitive neuroscience typically employs a synchronous sequential update cycle. On each cycle, units calculate their net input from units presynaptic to them and then calculate their new activation level. Since this update is performed solely with respect to unit activations from the previous cycle, it does not in fact matter what order the units are updated in and so, a conceptually simple order is typically used, e.g. sequentially from lowest index to highest. This is effectively the approach we adopted in our first case study, with communications between neurons performed in a regular serial order, which was synchronous in the sense that all communications were performed once (and no more) on each cycle; see the discussion in Sect. 3.3.1. Clearly, concurrency theory is not constrained to such regular serial communication schemes. In addition, the possibility for asynchronous processing with variation in sequencing of interactions across updates is interesting to consider in a neural modelling context. In particular, putting aside speculations on the putative role of the thalamus as a global synchroniser [DIB10], control does seem to be distributed in the brain, with regions executing autonomously of one another. This suggests asynchronous processing, as is characteristic of concurrency theory paradigms. In addition, fixed synchronous update can incur a considerable computational cost. For example, under such a scheme, units that are quiescent, i.e. are not receiving external stimulation and are active below the output threshold, still receive and, even, initiate output signals, even though all the resulting communications are transmitting null values, i.e. zero activations. Clearly, the burden of this (redundant) communication can be large in neural systems with sparse islands of activation, and it may even be prohibitive, to the point that simulation and/or verification become infeasible. Taking the visual system as a case in point, it is clear that quiescence is likely to be pervasive. In particular, modules in visual cortex, leading into the ventral stream of the inferior temporal lobe, typically code for all possible relevant features at all possible retinal locations. Since there will normally only be one feature present in the visual field at each location, and indeed only a small region of the entire field attended to, the vast majority of the neurons in these modules will be quiescent at any particular time. This also fits with the now widely accepted view that the brain employs sparse distributed codes [OLS97]. The implication of this observation is that a realistic model of the visual pathway (and, in fact, the majority of brain regions) could incur a massive computational cost if implemented using fixed synchronous update. Accordingly, this case study explores the feasibility of constructing neuron automata models more reflective of asynchronous concurrency, in which inter automata communication is not fixed and regular and indeed, communication demands can be reduced through components becoming quiescent.
4. **Large scale**. Finally, this case study explores the feasibility of building larger neuroautomata specifications, with hundreds, rather than tens (or less) of units, and applying model checking to such larger scale models.

4.1. The spotlit visual field (SVF) model

To realise these goals, we present a stylised model of visual processing, loosely based upon early visual areas in the primate brain, e.g. those found in the ventral stream from visual area one (V1) to Inferotemporal Cortex (IT). A particular aspect of the model will be the incorporation of a spotlight of visual attention, for which we seek to verify a number of correctness properties, concerning the accuracy with which attention is deployed. In addition, we will verify stability properties, which in this case study focused on freedom from oscillatory dynamics.

We call the model, the Spotlit Visual Field (SVF) model. We begin by presenting the model in general terms and then we focus down onto a reduced SVF model, which contains less units than the full model. We verify the key properties we are interested in over the reduced model. We then scale the specification up to a full SVF model. Here, the explosion in possible interleavings renders a full state space search intractable. The model, though, enables us to illustrate how a neuroautomata model could allow quiescence, with a corresponding tractability benefit.

The basic structure of the model is depicted in Fig. 9. Activation patterns are externally set at the input layer, which contains 100 units, divided into four five-by-five grids. Each grid schematically represents a portion of the visual field. External patterns impose letter stimuli on these grids and the model is set-up to detect the letters

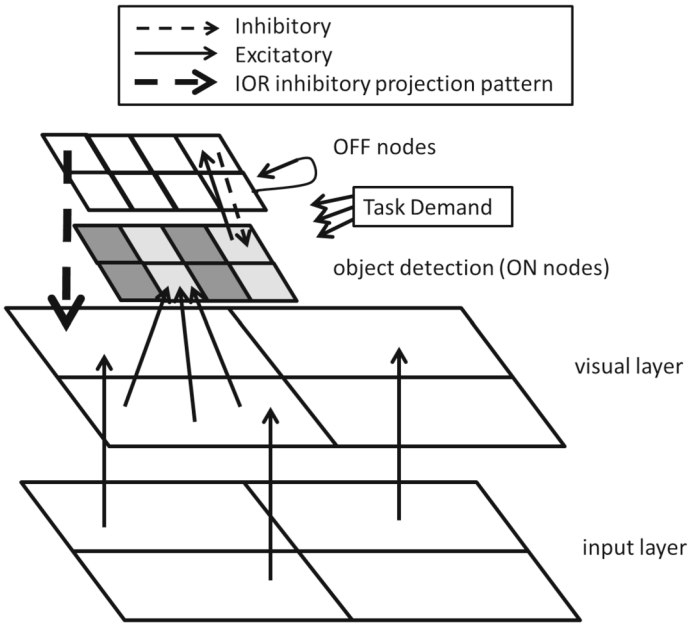

Fig. 9. Configuration of the Full SVF model's basic connectivity. *Grey fill* in the object detection ON unit layer reflects the distinction between the two target letters—*dark grey*, say, for A and light for V. The input to visual layer, ON to OFF node and OFF node to OFF node connectivity is all one-to-one. The visual to object detection ON nodes are many-to-one, implementing object recognition, e.g. detecting an A; the Task Demand to object detection ON units are one-to-many, selecting one or other letter to foreground; and the Inhibition of Return (IOR) projection is one-to-many, suppressing all visual units in one of the four regions of space

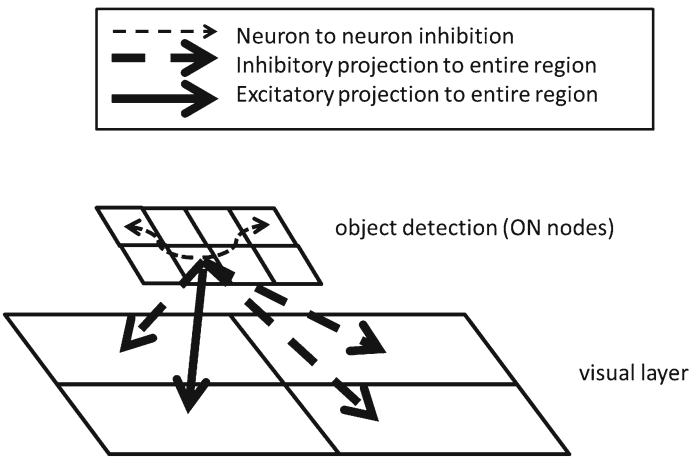

Fig. 10. Attentional spotlight and sharpening projections of the full SVF model, only part of which (i.e. from one object detection ON unit) is shown

A and V, when either is presented to it, in a sequence of letter stimuli appearing randomly in one of the four spatial positions. The input layer projects one-to-one to the visual layer, which mirrors the input layer's layout: four five-by-five grids. It is the first stage at which neural activation dynamics obtain, through the membrane potential update Eq. (2).

The object detection layer is also divided into four spatial positions (one for each region of the visual field), each of which comprises two units—one representing presence of an A in that region and the other V. Thus, the model reports presence of a target letter in a particular position. Visual to object detection layer connectivity performs this recognition; it is hardwired, although it could easily be learnt.

A task demand system directs the model to detect either A's or V's by foregrounding the relevant object detection layer units, one in each grid position. This then gives these units a competitive advantage (the lateral inhibition at the object detection layer will be discussed shortly), biasing the model to detecting that stimulus. This, then, is an approach consistent with Desimone and Duncan's [DES95] biased competition theory of top–down attention.

The model is set up to perform a sequence of stimulus detections. To do this, each detection episode needs to be terminated to enable the next to start. This termination involves the layer of OFF nodes (see Fig. 10), with one for each of the object detection layer units (which are conceptually ON units). Each OFF unit accumulates activation driven by the corresponding ON unit, which is conceptually similar to measuring a (time discounted) area under the ON units activation trace curve. An OFF unit crossing its output threshold marks the point at which a stimulus is perceived. Accordingly, the corresponding ON unit gets suppressed, freeing up the object detection layer to commence a next perceptual episode. OFF nodes are self excitatory, ensuring that once they are active they stay so, providing a primitive working memory, as per the trace-gate pairs in [BOW07]. A spatial Inhibition of Return (IOR) [TIP91] is also incorporated, whereby, the entire region at the visual layer of a perceived stimulus is suppressed. This ensures that the spotlight of attention does not immediately return to the position of a perceived stimulus. Both these mechanisms, OFF to ON feedback and IOR, drive the system to explore new regions of space, i.e. for attention to move once a perceptual episode completes.

The implementation of the attentional spotlight is shown in Fig. 10. Specifically, each object detection unit excites the whole of the corresponding visual layer region, and inhibits all other regions. In addition, lateral inhibition at the object detection layer enforces a winner-takes-all dynamic. Thus, the most strongly active object detection unit, over time, suppresses all others, thereby disinhibiting itself. These mechanisms ensure that when a salient item is detected, i.e. a target, at the object detection layer, the location of that item at the visual layer is enhanced, i.e. the attentional spotlight moves there. In addition, the lateral inhibition at the object detection layer, sharpens the representation, enhancing the winning unit, driving it to further focus the spotlight back down onto the relevant visual layer region.

4.2. Automata details

Units in the model have a regular structure, in the sense that each is modelled with two automata, the first realising a, so called, *input compartment* and the second an *output compartment*. The first of these accumulates inputs to the unit and calculates the membrane potential, while the second calculates the output activation and relays it on. The two compartments are synchronised through the action $update$.

Importantly, this two-compartment approach enables the input compartment to be permanently responsive; that is, all input actions are always offered. This is required to enable asynchronous update. In other words, if there is no fixed order to the pattern of communication throughout the model, and indeed, the possibility that some communications will not occur on some cycles (due to quiescence), then a unit cannot know when it has completed all the necessary communications that will occur on a cycle. In particular, it cannot know when it has reached communication completion from the inputs it performs. As a result, it cannot schedule updates by when all inputs have completed (which was the mechanism employed in the first case study). Indeed, purely on the basis of inputs performed, a unit cannot know when it is ready to update. Accordingly, the update decision is decoupled from the completion of inputs. It is then natural to decouple input and output functions altogether, and in doing this, to incorporate a fully responsive input compartment.

4.2.1. Uppaal specific notations

Clocks are the way to handle time in Uppaal. Time is continuous and clocks measure time passing. Time progresses globally at the same pace for the whole system, and the value of a clock can be tested or reset [BEN95]. Hence, guards in Uppaal models can have conditions on clock variables as well as other variables. Time progress conditions can also be associated with locations. These conditions are called *invariants* in Uppaal, and an automaton can only stay in a location when the invariant holds [BEN95]. It is important to understand the difference between the role of guards and invariants. In this respect we can distinguish between *may* and *must* timing. Guards express *may* behaviour, i.e. they state that a transition is possible or in other words may be taken. However, guards cannot "force" transitions to be taken. In contrast, invariants define *must* behaviour. This *must* aspect corresponds to urgency, since an alternative expression is that when an invariant expires, outgoing transitions must be taken straightaway. Finally, a location can be urgent permanently (marked by U), i.e. time is not allowed to pass when an automaton is in an urgent location [BEN95].

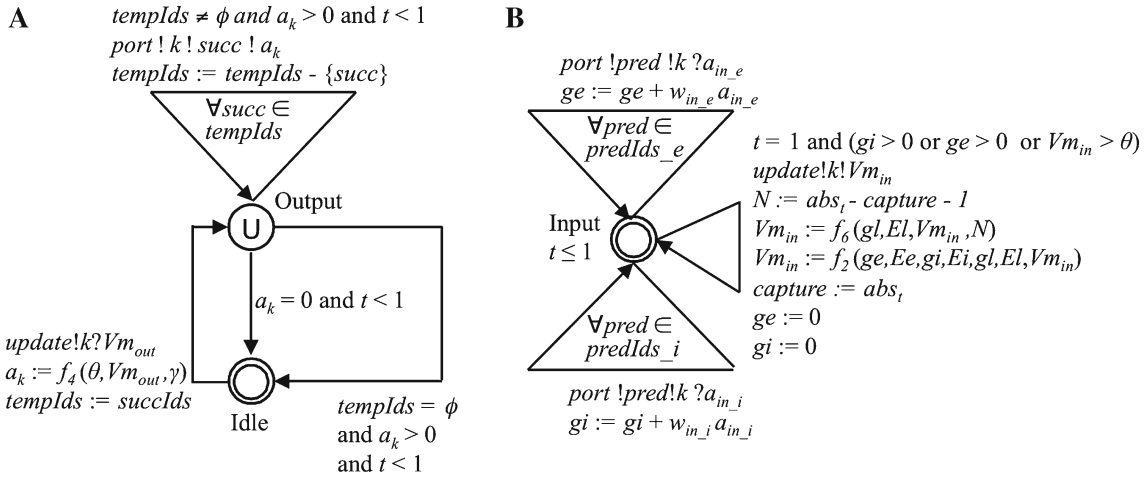

Fig. 11. Structure of **a** the output compartment and **b** the input compartment

4.2.2. Output compartment

Figure 11a depicts the basic output compartment. This is common to all model units, which are distinguished across different layers solely by the form of the input compartment that is placed in parallel with the (common) output compartment.

The DRCA approach is extended here to support description of timing constraints. These are based directly upon the syntax of Uppaal [BEN95]. Thus, t here is a global clock variable, which is updated in automaton *Environment*, see Fig. 12.

The overall behaviour of the output compartment is as follows. Firstly, it waits for the input compartment to signal readiness to update the units activation (which happens when t equals one). This involves synchronising on the action $update$ and calculation of unit activation (a_k) in terms of the current membrane potential (Vm_{out}). The Output compartment then branches dependent upon whether a_k is zero or not; if it is, then the membrane potential is below threshold and the unit becomes quiescent. Thus, output action $port$ is only performed if there is a non zero activation to transmit.

This capacity for units to become dormant and thus not incur any communication cost brings a tractability benefit, and is the primary methodological advantage of the approach presented in this second case study.

4.2.3. Input compartments

Visual layer Figure 11b presents the generic form for the input compartment employed in all units. The compartment is permanently input responsive, i.e. all input actions it can perform are offered at all times. The first set of such actions are the excitatory inputs, the identifiers of which are ranged over by $predIds\_e$. These include (one-to-one) excitatory projections from the input layer, see Fig. 9, and (one-to-an entire region) excitatory projections from the object detection ON units, see Fig. 10. The latter of these implement the excitatory part of the attentional spotlight. Specifically, evidence (even precursory and weak) for presence of a $target$, which would be observed at the object detection layer (ON nodes), causes feedback amplification of the corresponding region in the visual layer. This enhances processing at that region, in turn accelerating the representational build-up for that item at the object detection layer, bringing an advantage for attended items, i.e. those that are targets.

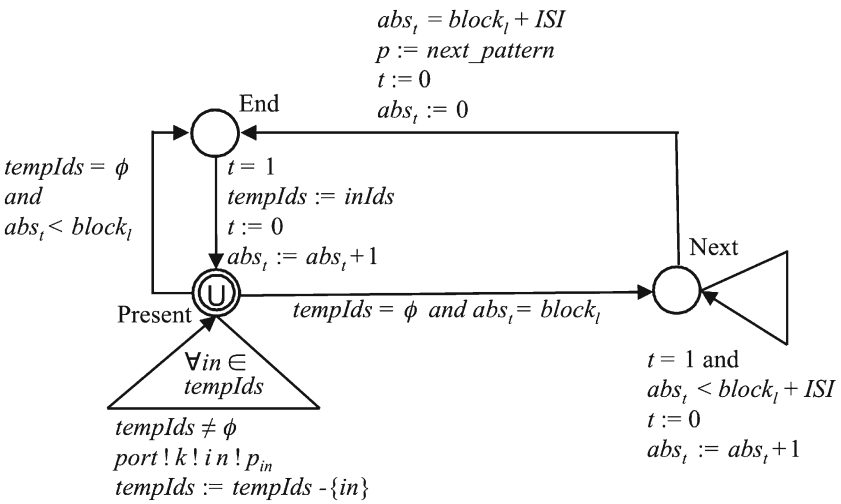

Fig. 12. Model of the automaton *Environment*, where p is initially set to the first pattern, and $tempIds$ is initialised to $inIds$

The second set of actions are inhibitory, with identifiers ranged over by $predIds\_i$. These include (one-to-all other regions) inhibitory projections from object detection ON units, see Fig. 10, and (one-to-an entire region) inhibitory projections from object detection OFF units, see Fig. 9. The former of these (from object detection ON units) further focuses the attentional spotlight, by actively suppressing nonattended regions, i.e. those in which the target being enhanced did not appear. In contrast, the latter (from object detection OFF units) implements IOR [TIP91]. Specifically, activation of an OFF unit is taken as a marker that the corresponding item has been sufficiently activated at the item's ON unit (i.e. sufficient evidence has accumulated) for it to have been perceived and encoded into short-term memory.

Clearly, a well functioning attentional system can make a sequence of fixations. To do this, a mechanism to enable attention to move once an item is fully perceived at a particular position is required. This is the role of inhibition of return. Specifically, once an item has been perceived, the portion of space in which it occurred is suppressed, effectively releasing the spotlight. This is because the object detection layer representation of the stimulus at that spatial position will subside (since the visual layer is no longer driving it) and it is that representation that enforces spotlight deployment. The spotlight can then move to the next most salient region of visual space. This ensures that attention scans visual scenes through a sequence of discrete attentional amplifications.

A further transition performs the interaction between input and output compartment. This is the $update$ transition, which signals that the membrane potential has been updated and the output compartment can proceed to update the output activation of the unit. The guard of the transition ensures that at the allotted time point (when $t = 1$), the unit becomes quiescent, i.e. does not update, specifically, when ge and gi (the excitatory and inhibitory inputs to the unit) are zero and the membrane potential is below threshold θ. In this situation, the unit cannot have any affect on the units it projects to and thus, can refrain from communicating. When a unit goes into quiescence, it records the clock value (i.e. $capture := abs_t$), which can be compared to the current time when it is excited again. We know a unit is activated from a previous quiescent state if there is a non zero difference between the current time and the recorded clock value (i.e. the period of quiescence $N > 0$).

A consequence of this though, is that when a non zero ge or gi first arises (i.e. the unit starts being excited) after a period of quiescence, a more complex (compensatory) membrane potential update needs to be performed, as shown in Fig. 11b. This is incorporated in Eq. (6), which is only computed if $N > 0$. The membrane potential reached during a period of quiescence (N time points) can be determined from this aggregated update, since the only current active during that period is the leak. Thus, rather than the neuron leaking step-by-step, with the membrane potential reducing on each clock tick, the effect of a sequence of updates is enforced in a single step. (Please see "Appendix B" for the proof of this equation.)

$$Vm_{in}(t+N) := f_6(gl, El, Vm_{in}(t), N) = Vm_{in}(t) + (El - Vm_{in}(t))[(-1)^{N+1}(dt_{vm}\ gl - 1)^N + 1] \quad (6)$$

Object detection layer The Object Detection (OD) layer is itself composed of two sublayers; see Fig. 9. The first of these contains OD layer ON units, each of which represents a particular object, A or V, at a particular position. The activation level of each such unit reflects the amount of current evidence for that object having been present in the visual field at that position.

Each ON unit has a corresponding OFF unit, which is a form of inhibitory interneuron. The OFF unit is excited by the ON unit, which it in turn inhibits, see Fig. 9. Such ON–OFF circuits are commonly found in neural models of sequential behaviour, see for example, [BOW07]. So, here, the OFF unit's membrane potential builds up as the corresponding ON unit does, but with the OFF unit's activation delayed relative to the ON units. Functionally, this implements a form of OFF switch; that is, once sufficient evidence has accumulated (at the appropriate ON unit) to judge a particular object as seen at a specific spatial position, then the corresponding OFF node crosses threshold and suppresses the ON unit. This then terminates that perception-encoding episode, disinhibiting the other object detection layer ON units (which were suppressed due to lateral inhibition, see Fig.10), and, thus, enabling a new perception-encoding episode to commence.

The input compartment of OD layer ON units is as shown in Fig. 11b. The self transition (at the $Input$ location) of excitatory inputs carries the object and spatial position-specific input from the visual layer. (Note, we talk about self transitions when discussing automata specifications and self loops when describing self links in neural nets.) Thus, the weights w_{in_e} on these links enable a particular object, A or V, to be detected at a particular position in space. In contrast, the transition self loop (at the $Input$ automaton location) of inhibitory inputs carries firstly, lateral inhibition from the rest of the OD layer ON units. This sharpens object detection representations, inhibiting losing units and, in turn, disinhibiting the winning unit. Secondly, the inhibitory self transition carries the feedback inhibition from the corresponding OFF node. Finally, an individual self transition updates the membrane potential, again compensating for periods of quiescence, and another enables task demand to act upon the OD layer. Specifically, task demand can either foreground all A units or all V units across all spatial positions.

OFF unit input compartments have a simple structure since they do not have inhibitory inputs; they are excited by their corresponding ON unit and they excite themselves, and update in the usual fashion. The self excitatory loop ensures that, once an OFF unit has crossed threshold, it will stay active. This then acts as an actively maintained representation that the particular object-location conjunction occurred. This is similar to the gate-trace structures found in the Simultaneous Type/Serial Token model, see [BOW07, BOW08].

4.2.4. Task demand

The input compartment of a task demand unit has no input from other units, so it does not have any self loop with input action $port$. Instead, it simply maintains its current membrane potential at a constant super-threshold value. The associated output compartment projects to all OD layer ON units that code for the object, A or V, the particular task demand unit selects for. In addition, task demand units are space nonspecific. That is, they project to all units coding the relevant objects, whatever the spatial position.

4.2.5. Stimulus presentation

Extended from the model in the first case study, the environment automaton controls the presentation of stimuli to the model, see Fig. 12. For the verifications presented here, stimulus presentation comprises two episodes (though the model allows an arbitrary number of stimuli to be presented to the visual field): a first in which objects A and V are simultaneously presented in two different spatial positions for the block length of $block_l$ units of time and then a second episode, in which A and V are again simultaneously presented, but, here, in the remaining two positions. Between these two blocks, no stimulus was presented to the visual field for *ISI* units of time. This is referred to the *Inter Stimulus Interval*. The model dynamics sought are that in each episode, both objects are similarly represented (i.e. activated) initially; however, the selectivity of task demand causes attentional enhancement of the position of one of A or V, whichever is the target. This in turn, gives the target a competitive advantage, initiating winner-takes-all dynamics, sharpening the representation, until only the target object (in its presented position) is strongly active at the OD layer.

Initially, the p_{in}'s are set to the first pattern, which are presented to the network at the $Present$ location of the *Environment* automaton. At the End location, the *Environment* automaton waits until units has finished updating their activations, i.e. when $t = 1$, before presenting the pattern again at the $Present$ location. When the first block ends, the automaton moves to the $Next$ location, where it does not present any pattern to the network for ISI units of time. And then, p_{in}'s are set to the $next\_pattern$ when the second block starts.

4.2.6. Clock update

A clock variable t is reset every one time unit. This is performed in the $Environment$ automaton. In addition, another clock variable abs_t is set every one time unit, c.f. Fig. 12, and reset to zero at the beginning of every block of stimulus presentation in the $Environment$. In this model, as previously discussed the input and output processes are decoupled and inter unit communications are asynchronous. In addition, we restrict the order of updates between the neuron automata and the environment automaton by assigning a lower priority for the environment. Uppaal ensures that the clock resetting in the environment automaton is only performed after all inter/intra neuron communications have taken place. So, the clock variable t is in effect a "global synchroniser" that ensures all units update no later than $t = 1$ if they are not quiescent.

4.3. Full and reduced models

As previously discussed, we actually investigate two model variants: full and reduced. The full model is as described in the previous sections, but with large layers. In particular, it has 100 input layer and 100 visual layer units, with one-to-one unidirectional links between them. Each of these layers comprises four five-by-five grids. Stimuli are letters (and what we have called objects are letters), which can each be presented on one such grid, each of which represents a spatial position. Thus, when nonactive, each grid represents a position-stimulus combination. Finally, there are eight OFF units, each paired with one ON unit.

The reduced model maintains the same functional structure, but compresses layers down in size. Thus, the input and visual layers contain just four units each. These reflect two spatial locations, with two stimulus representation units at each location. These stimulus units provide a localist representation of each stimulus/object, i.e. A or V. Similarly, the reduced OD layer collapses down to just four ON–OFF pairs, one for each location-stimulus combination.

The reason for including two model versions is to illustrate two different aspects of verification that can be performed on this case study. Specifically, with the reduced model, we are able to perform a complete verification of all the properties we are interested in, including a complete state space search, in which, verification needs to explore all possible interleavings of updates, causing an exponential state space explosion. It is only possible to perform this complete verification with the reduced model. We consider the full model in order to investigate verification in the context of a nontrivially sized model, where the possibility for units to attain quiescence becomes especially important. We first discuss verification of the reduced model and then scale up to the full model.

4.4. Reduced model verification

We test two basic reachability properties expressed in the Uppaal query language—a simplified version of CTL [BEH04]. Both use the following formula.

$$R(targ\_OFFx) = \exists \Diamond (a_{targ\_OFFx} > 0)$$

which is satisfied if there exists a reachable state in which the target OD layer OFF unit x is active for judging that detection has completed (i.e. a_{targ_OFFx} is greater than zero). The two formulae, $R(targ\_OFF1)$ and $R(targ\_OFF2)$, are shown to hold, where $targ\_OFF1$ is the first target stimulus and $targ\_OFF2$ the second.

We also verify that attention deployment is well behaved. This uses a test automaton, see Fig. 13a, where abs_t is a clock variable that is never reset within a block and, thus, represents the absolute time lag from the onset of the stimuli at any moment. In addition, $block_l$ denotes the length of time a stimulus is presented to the model for, which we call the block length. We wish to verify that attention is always deployed to a target stimulus before its presentation at the input layer terminates.

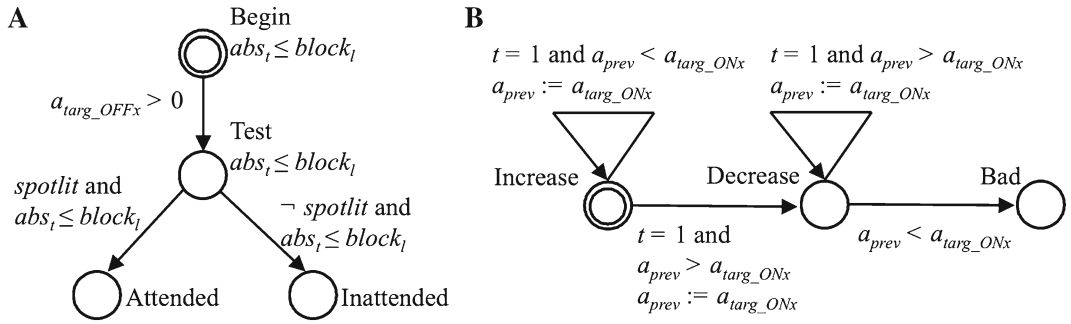

Fig. 13. Test automata used to **a** verify deployment of the attentional spotlight, **b** verify freedom from oscillations

We test the reachability of locations $Attended$ and $Inattended$, which can only be reached if the proposition *spotlit* holds, respectively, does not hold. *spotlit* holds when the sum of the visual layer activation in the spatial position containing the target is larger than this sum at any other spatial position. This is judged to reflect successful deployment of attention. Thus, we test the following formulae.

$$\exists \Diamond(Attended)$$
$$\exists \Diamond(Inattended)$$

As hoped, the first of these holds, but the second does not, indicating that attention is always successfully deployed. Similarly, we were able to show that the second target is always attended before its presentation terminates.

In the first case study, we explored the stability of learning, which focused on the trajectory of the networks error during training. A more basic form of stability is relevant to nontrivial recurrent neural networks, such as the Spotlit Visual Field model presented here. This concerns whether the model can exhibit oscillations and the character of those oscillations if they can arise. Again this is the type of property where model checking may be especially useful, since one would like to know that a particular configuration can never exhibit oscillatory instability. Simulation does not confer such certainty, since, for nontrivial systems, the set of simulation runs performed will only ever have explored a subset of the full state space.

If one is explicitly modelling the brains oscillatory dynamics, one would like to verify that oscillations are damped, i.e. reducing in size through time. However, for the type of rate coded models we are considering here, the issue is more to verify complete freedom from oscillations, and that is what we investigate. The ON–OFF circuits we employ in the OD layers are particularly susceptible to generating oscillations, since they can exhibit execution trajectories in which activation of ON and OFF units are out of phase with one another.

To assess whether such oscillations can manifest, we present another test automaton as shown in Fig. 13b. We test reachability of the location Bad, which determines whether the activation at a_{targ_ONx} can oscillate, i.e. increase again at some point after it started to decay. This is a general approach, since it considers how the direction of activation changes. That is, once activation has gone up (location $Increase$) and then down again (location $Decrease$), if it again increases the tester passes to the Bad location. If Bad is reachable we say an oscillatory dynamic is possible.

Uppaal shows that if the self loops on OD layer OFF units are not present, location Bad can be reached. However, inclusion of the OFF self loop stabilises the system and Uppaal shows that Bad becomes unreachable. This is because once the OFF unit crosses its activation threshold, the strong excitatory self loop drives the OFF unit up to saturation, where it will sit from then onwards, self supported, whatever the level of the corresponding ON node. In particular, suppression of the ON node down to resting by the strongly active OFF node reduces the feedforward excitatory drive to the OFF node, but that reduction of excitation is compensated for by the OFF nodes self excitation. This maintains the OFF node at saturation and its maximum inhibition of the ON node from then onwards; the ON and OFF units then stay permanently at those levels.

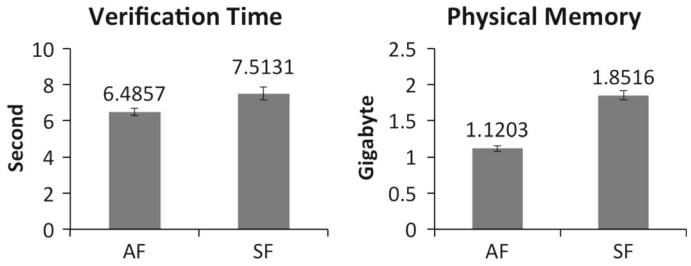

Fig. 14. Performance of the model checking. *AF* asynchronous model allowing units to stay quiescent, *SF* synchronous model in which all units update even when not activated. AF was significantly faster to verify ($p = 0.0203$) and used significantly less memory ($p < 0.000003$). *Error bars* show the standard error of the mean

4.5. Full model verification

As previously stated, we also scale up the Spotlit Visual Field model to explore the advantage gained by incorporating quiescence. Accordingly, the full model is structurally identical to the reduced model, but with layers (particularly Input and Visual) dramatically increased in size. Consequently, at any one instant, only a small subset of the model units will be active above threshold. In this context, not updating below threshold units, with the corresponding reduction in communication exchange, could bring a considerable efficiency benefit, making some otherwise infeasible verifications feasible.

With this full model, with quiescence, we verify a number of properties, which mirror those considered for the reduced model. Firstly, we tested $R(targ\_OFFx)$ for both first and second targets, as we did for the reduced model, with the number of OD layer units increased from four to eight. As expected, these properties are shown to hold.

Secondly, we tested reachability of the $Attended$ and $Inattended$ locations for both targets using test automata similar to those considered for the reduced models. For both targets, $Attended$ was reachable, but the verification of $Inattended$ was inconclusive, since the verification ran out of memory. The model checking was performed on a Macbook Air laptop computer with Mac OS X version 10.7.5, 1.7 GHz Intel Core i5 CPU, 4 GB 1,333 MHz DDR3 RAM and 120 GB of disk space. We used the Academic and non-commercial version of the Uppaal 4.0.13 (http://www.uppaal.com).

To demonstrate the value of the model with quiescence, we tested the reachability of the $Attended$ location in the full model with quiescence (asynchronous updates) or without quiescence (synchronous updates). To model without quiescence, we modified the input compartment of all units in the full model. Specially, on the self loop with action $update$ in Fig. 11b, we changed the guard to $t = 1$ and removed the constraint on inputs and membrane potential. So, after this modification, the unit will update every time even it is not activated. The verification was repeated ten times in order to account for any random noise introduced during the evaluations of CPU and memory performance. The inference was drawn by performing a *Student's T test* on the two distributions of time (or memory) for both models, i.e. with or without quiescence. The results are shown in Fig. 14. It can be seen that the asynchronous model (AF) was verified significantly faster ($p = 0.0203$) and used significantly less memory ($p < 0.000003$) than the synchronous model (SF). So, the model with asynchronous update is not only more biologically plausible, but also results in a large improvement in performance.

5. Conclusions

In this paper, we presented a communicating automata approach to specifying neuronal networks and their learning mechanisms. Our models can be regarded as symbolic descriptions of neuronal behaviour, which allows model checking to be applied. At one (low) level, our models describe biological mechanisms underlying the central nervous system. At another (high) level, the requirements expressed in logical formulae represent high level

(often externally observable) behaviour and properties of the cognitive system without concern for the description of internal structure. Verification, in particular, model checking, may provide theoretically well-founded ways to relate descriptions of the same system at different levels, e.g. it could also potentially determine whether certain high-level cognitive properties might emerge from neurobiological mechanisms. Our verification has demonstrated that analysing stability related properties is important when comparing and contrasting different learning mechanisms, and testing properties of attention mechanisms in visual systems.

In the first case study, our results suggest that although proposed as a biologically plausible alternative to BP learning, GeneRec has distinct characteristics regarding stability. This difference has significant impact on the quality of the learning, and whether or not GeneRec is a suitable model of the mind/brain. (When biological plausibility is not required, such as in AI applications, users can opt for BP learning.) We also observed that the unstable characteristics of GeneRec learning is related to the difficulty of the task, e.g. the linear inseparable XOR problem, and the nonlinearity of the activation function. In particular, the approximation assumption made by GeneRec worsens when the neural activation exhibits nonlinear changes. Similar findings have been observed by a number of researchers, e.g. [BAK00], but none has exclusively focussed on GeneRec. Our results suggest that the choice of activation function to use with GeneRec is an important one, constraining the generality of the point neuron model [ORE00].

It is also important to note that the steepness issue with regard to activation functions is not just a theoretical matter. In particular, shallower activation functions (i.e. close to linear) are fundamentally more sensitive; that is, their output differentiates more fully amongst the range of possible inputs (i.e. net inputs). One boundary condition here is a step function (which is approached as γ increases). Such a function only distinguishes high and low values, but is insensitive to gradations within either of those ranges. Importantly, one would assume that as steepness reduces and sensitivity increases, the underlying biological system becomes, necessarily, more complex. Again thinking of boundary conditions, implementing a neuron or cell assembly that exhibits a step transition in output would be biophysiological trivial, e.g. it only requires a threshold on output. In particular, shunting membrane potential terms and leak stabilization current can be ignored. Thus, the instability of GeneRec with steep activation functions imposes a real constraint on biological implementations. If the brain does in fact implement a learning algorithm similar to GeneRec, then somewhat more complex, and presumably, more resource hungry neural implementations have to be present.

In the second case study, we indeed extended the model to allow asynchronous processing and proposed a non-serialized Spotlit Visual Field model for the visual system. We modelled the deployment of an attentional spotlight to different spatial positions in the visual field in recognizing different letter stimuli. We then verified a number of correctness properties using Uppaal. The results show that the models are well behaved in terms of allocating attention in space and time. And we also showed using Uppaal that the model is stable (free from oscillatory dynamics) when the OFF units have excitatory self loops. The full model tested in case study 2 contains several hundreds of neurons and represents a non-trivial cognitive phenomenon. The stability properties tested in both case studies are only verifiable using model checking, and cannot be guaranteed using simulations demonstrating the value of applying concurrency theory to the modelling of neural systems.

Case study 2 also shows the benefit of modelling quiescence by allowing units to communicate asynchronously. It is argued that parallel and distributed control is a fundamental property of the brain and our cognitive systems. However, the state-of-the-art modelling techniques still often implement cognitive models by assuming synchronous updates. Our analysis has shown that such synchronous updates are not only biologically implausible, but also increase the cost to simulate and verify. As the field of cognitive neuroscience and computational neuroscience evolves, imaging the whole brain and modelling brain-wide dynamics with not just hundreds but billions of units promises to reveal the emergent properties of the human mind and its true complexity. For instance, the ambitious Human Connectome Project (http://www.humanconnectomeproject.org) aims to map out the entire human brain structure and functional networks with fine grained neuronal connections and interactions. Another equally ambitious project is the Human Brain Project (https://www.humanbrainproject.eu/) with the goal to simulate the whole human brain with detailed biophysiological processes. One would predict an extremely high demand for computing power in modelling whole brain dynamics, and supercomputing technology will eventually reach its energy limit. So, we argue that the modelling framework proposed in case study 2 may bring performance advantages to such large scale virtual brain projects.

We know that the brain is very efficient in terms of its energy consumption, since the cortex widely adopts distributed codes and sparse representations. It has been estimated that less than 15 % of neurons in the brain

are likely to be active simultaneously, and it is the action potentials and postsynaptic effects of glutamate that consume most of the energy [DAV01]. When neurons are at rest, they only consume a small amount (13 %) of energy to maintain their resting potential [DAV01]. So, modelling the brain with most of the units staying in quiescence provides a nature way to emulate neurobiology and its energy strategy.

It is also worth reflecting on how complexity issues are impacted by the proposed communicating automata approach to analysing neural networks. There seem then to be two complexity questions: (1) does the implementation in communicating automata add complexity in respect of simply "running" a neural net; and (2) how do the complexity bounds of model checking impact the kind of verifications (of neural nets) we are proposing.

On the first of these questions, the update cycle of the neural network really governs how costs of executing the neural network increase with network size, i.e. number of units or synapses/links. The activation update scheme employed, e.g. serial or asynchronous update, is a decision that one has to make for any neural network framework, e.g. whether implemented using formal method or not, and in that sense, is an orthogonal question to that of the framework chosen. In other words, any update scheme could, it would seem, be implemented in our communicating automata framework and that implementation would inherit the complexity inherent to that scheme. Thus, it is the update scheme that determines this aspect of complexity, not the implementation framework.

On the second of these questions, the complexity of model checkers has been well characterised, and our approach inherits that. The basic result is that space, rather than time, is the key bound on complexity, which is logarithmic in the size of the model and polynomial in the size of the formula being tested.

We also realise that there are a number of limitations of our approach. Firstly, the size of the neuronal networks used in this paper remain small (i.e. a highly abstracted model of the visual system) compared to the size of the human brain, although it is theoretically possible to build larger and likely more complicated models by hierarchically composing such simple networks (see Sect. 6.1). Secondly, we should explore probabilistic and stochastic process algebra in the future. They may provide a more powerful means to specify biological processes and human cognitive behaviour, which are intrinsically probabilistic. However, such future work introduces challenges for more powerful and effective model checkers.

6. Comparing with other approaches applying formal methods to cognitive systems

6.1. Smith et al.: a framework for neural net specification

Although conventional connectionist networks are commonly regarded as concurrent and distributed, they are typically limited to only one level of concurrently evolving modules, i.e. the primitive elements of neuronal networks are neurons but not (sub) neuronal networks. It is hard to construct and understand large architectures without hierarchical structuring. In certain respects, modelling based on classic neuronal networks is low-level in character, i.e. it is hard to relate neuronal networks to primitive constructs, data structures and computational components found in high-level notations preferred by the symbolic modelling community.

A similar framework to our approach was introduced by Smith et al. [SMI92] in a general mathematical setting. Their work has addressed the issue of constructing neuronal networks hierarchically. However, it does not consider automatic verification, although it may be that their formalisation and our notation could be translated from one to the other. Despite many similarities between Smith et al's method and our DRCA and SPIN model checker approach, these two methods have major differences in their objectives. That is, Smith et al. only proposed a formal framework to systematically and hierarchically construct sophisticated neuronal networks. Our approach, at least to some extent, inherits such abilities, but with an important extension to model checking. We argue that this is a major advance over Smith et al's method, because it allows formal comparison of alternative cognitive models. It also has implications for closing the gap between observable behaviour and underlying biological mechanisms. A rigorous methodology for analysing the behaviour of neuronal networks is as important as specifying their structure. It is relevant to both cognitive modelling and practical applications in machine learning. The former requires explanation of psychological phenomena and formulation of predictions about the properties of cognitive systems. The latter requires analysis of the reliability and performance of AI systems.

6.2. Gabbay et al.: neural-symbolic learning systems

Symbolic systems are good for manipulating, explaining and reasoning about complex data structures, but neural networks are good at dealing with complex and highly nonlinear systems, especially in handling catastrophic changes or gradual degradations [NAP99]. To integrate the strengths of both symbolic or rule-based systems and sub-symbolic (mainly neural) systems, Gabbay et al. [GAR02] has introduced a hybrid approach that provides a superior learning capacity than any of the single systems alone in solving many difficult learning tasks. The neural-symbolic learning system allows users to input prior knowledge in the form of logical formulae or rules. Then, the rules are coded as neural networks, and a second stage of neural network learning is applied, i.e. statistical learning with examples. After the neural networks have been refined with additional training data, rules can be extracted as logical formulae from the trained networks.

Gabbay et al.'s approach differs from ours, as for them, logic is regarded as a means for helping to improve conventional neural networks, and to explain the inference process underlying them. Instead, LTL in this paper is used to specify the observable behaviour of the system. Such a requirements language can be used in model checking and investigation of the properties of neuronal networks. However, the approaches share some commonalities. For instance, we argue that many fundamental properties of system dynamics are not explicit in a (neural) system specification and, therefore, verification of neuronal networks is required. For Gabbay et al., rules extracted from the trained neuronal networks also provide some degree of assurance to the network's behaviour, because the extracted logical formulae are often more readable to human users and suitable for mathematical analysis.

6.3. Blandford et al.: verification-guided modelling of salience and cognitive load

Blandford et al. have developed several models for high-level cognition in the context of salience and cognitive load effects in human–computer interaction. These models can be applied to evaluate the designs for various computer interfaces, such as ATM terminals and a fire engine dispatch interface [RUK09]. In these models, cognitive entities and processes are modelled as logical formulae. The usability of the modelled interfaces are evaluated using model checking. This approach is very similar to our DRCA and model checking methods, except that their models are specified at a much higher level of abstraction. This is justifiable when the models need to capture high-level cognitive human behaviour in the context of interactive systems, where a large amount of biological detail of the user can be abstracted away.

In comparison, our approach is much more detailed about the internal organization of cognitive systems and biological mechanisms. Such an approach is applicable to the variety of connectionist modelling that has gained much attention in cognitive neuroscience. It also provides a formal and mathematically rigorous connection between the high-level (observable) properties of cognitive systems, such as the quality of learning, and low-level neurobiology, via model checking.

6.4. Barnard et al.: process algebraic modelling of interacting cognitive subsystems

Barnard et al. have argued that concurrency theory is a useful computational formalism for modelling complex mental architectures. For example, Barnard and Bowman [BAB04] have proposed a process algebraic model of Interacting Cognitive Subsystems (ICS) [BAR85]. Their model sits at the psychological level and uses the process algebra LOTOS [BOL88]. Barnard and Bowman have also discussed the relationship between such formalisms and neuronal network approaches, particularly the pitfalls of using neuronal networks to construct large scale mental architectures, such as ICS. The ICS model contains a set of top-level modules that are connected by communication channels. Modules interact by exchanging data items along channels. Also, control is distributed and each module evolves independently.

The ICS model has been used to simulate human attention in the context of Barnard's key-distractor Attentional Blink task [BAB04, SBB07, SBB11]. This model captures core aspects of the allocation of attention over time and is applicable across a range of practical settings where human attentional limitations come into play. In addition, this model makes predictions about human electrophysiological data, which can be compared to real electrophysiological data recorded from human participants [SBB09].

The major difference between Barnard et al.'s methods and our communicating automata model of neuronal networks, is that they work at different levels of abstraction. That is, the ICS model is defined at a higher psychological level, while our approach is at a lower (biological) level. Another difference between these two approaches is that we have applied formal verification here, but so far, the ICS model has only been tested using simulations (although, see [BOW99]). (Based on the work from Blandford et al., it is reasonable to believe that model checking is also feasible for models expressed at higher abstraction levels.)

6.5. NASA: formal verification and validation of artificial intelligence systems

NASA has widely employed AI systems in controlling their space/air crafts. However, testing the reliability of such control systems is difficult and expensive. One reason is that the inputs to the AI systems are frequently huge and discontinuous. In addition, small changes in inputs often lead to dramatic changes in performance, because these control systems are highly interactive and nonlinear. Currently, testing can only be applied to a subset of the input space, and, effectively, it is assumed that other inputs would behave similarly. So, the effectiveness of the testing largely depends on selecting a *good* test set. For example, several case studies (e.g. [ART03, BAR04, GIA02]) have applied run-time monitoring and model checking of temporal logic properties to the K9 planetary rover. In addition, some parts of NASA's Remote Agent Experiment Executive have been model checked using the SPIN model checker, and five concurrency errors were uncovered and fixed [HAV00].

However in these previous applications, controllers were mostly pre-trained networks, so verification does not consider the adaptability of the system, i.e. the online learning capacity was not assessed using model checking. In fact, the correctness of such systems was only guaranteed offline by model checking [ROD01]. However, there are, surely, other applications, in particular those using online trained neuronal networks, in which model checking has not yet been applied. So, our approach could provide some degree of online quality assurance for such neuronal network based controllers.

A. Semantics of data rich communicating automata

The following two sections present inference rules for generating labelled transition systems from networks of automata defined in the DRCA notation. The first section gives definitions for the basic untimed DRCAs used in case study 1, presented in Sect. 3, and the second section gives definitions for the timed DRCAs used in case study 2, presented in Sect. 4.

A.1. Untimed semantics

A network of DRCA automata is mapped to a labelled transition system, where labels are internal or observable actions; the former being indicated by the label τ. We define states as mappings from variable names to values; Guards as mappings from data states to truth values, and Effects as mappings from data states to data states, i.e.

$$\begin{aligned} State &= Names \rightarrow Values \\ Guards &= State \rightarrow Boolean \\ Effects &= State \rightarrow State \end{aligned}$$

Now, DRCA transitions are all annotated with a guard, and action offer (which could be τ) and an effect. In other terminology, often used in formal methods, guards are preconditions for performing the action and effects are imperative postconditions.

Then, an automaton in the network can make an internal transition via the following internal transition rule.

$$\frac{(i,l) \xrightarrow{g,\tau,E} (i,l') \quad g(\sigma) \quad \bar{V}_n[i] = l}{\bar{V}_{n,\sigma} \xrightarrow{\tau} \bar{V}_{n,E(\sigma)}^{i \rightarrow l'}} \quad (1 \leq i \leq n)$$

where the following notation is used, (i, l) indicates automaton i in location l, where $l \in Loc$, the set of all locations. g is a guard and E an effect. τ is the internal action. $\bar{V}_{n,\sigma}$ denotes a vector of n locations, applied to a global data state, $\sigma \in State$. So, $V_{n,\sigma}$ defines the current state of the DRCA network, both in respect of locations component automata are in and data state. Thus, $\bar{V}$ is of the form $\langle l_1, \ldots, l_n \rangle$. $\bar{V}_n[i] = l$ states that the ith automaton is in location l. $\bar{V}_n^{i \rightarrow l'}$ is defined as $\langle l_1, \ldots, l_i, \ldots, l_n \rangle^{i \rightarrow l'} = \langle l_1, \ldots, l_i', \ldots, l_n \rangle$.

Semantics for synchronisation We now move to define the rules by which automata synchronise in performing observable actions. As a part of this synchronisation, data values can be passed through data passing actions. Specifically, $e\bar{d}_m$ denotes an event name and a vector of data passing declarations. Thus, $\bar{d}_m = \langle d_1, \ldots, d_m \rangle$.

We impose a number of constraints on data passing actions to simplify the semantic model.

1. In a vector of data passing declarations, such as, $\bar{d}_m = \langle d_1, ..., d_m \rangle$, if two variable declarations occur, i.e. for $i, j (i \neq j)$, $d_i = ?z$ and $d_j = ?y$, then these declarations refer to different variables, i.e. $y \neq z$. Without this constraint, the order in which such declarations are resolved would affect evaluation, and indeed such situations would not be intuitive.
2. All variable declarations in vectors of data passing declarations, refer to a variable local to the automaton that declaration occurs in. This constraint prevents effective nondeterminism arising due to the order in which updates are made to a global variable in different automata.
3. No two transitions in different automata that may synchronize can refer in their effects to the same variable, where two transitions have the possibility to synchronise when they have the same event name, the same length data passing vector and do not have data passing declarations in the same position in the vector (this last constraint is clarified in the next point). Again, this avoids the possibility of effective nondeterminism due to the order in which effects are performed when two automata transitions synchronise.
4. We limit the complexity of multiway synchronisation to simplify semantic definition. Specifically, we only consider two-party synchronisations, and we do not allow synchronisations in which two data passing declarations match, e.g. $?x$ and $?y$. Thus, in LOTOS terms, we do not allow value generation [BOW06]. That is $a?$ can match $a!$ and two $!'s$ can match, assuming consistent types, but not ? and ?. This constraint is sufficient for this paper, since none of our case studies uses more than two-party synchronisation or value generation. The semantics could though easily be extended to fully general multiway synchronisation.

The synchronisation rule is as follows.

$$\frac{(i,l) \overset{g, e\bar{d}_m, E}{\longrightarrow} (i,l') \quad (j,r) \overset{g', e\bar{d}'_m, E'}{\longrightarrow} (j,r') \quad i \neq j \quad \bar{V}_n[i] = l \quad \bar{V}_n[j] = r \quad (g \wedge g')(\sigma) \quad \bar{u}_m = e\bar{val}(\bar{d}_m, \sigma) \quad \bar{u}'_m = e\bar{val}(\bar{d}'_m, \sigma) \quad \bar{u}_m \doteq \bar{u}'_m}{\bar{V}_{n,\sigma} \overset{e.\#(\bar{u}_m, \bar{u}'_m)}{\longrightarrow} \bar{V}^{i \to l', j \to r'}_{E(E'(\alpha(\bar{u}_m, \bar{u}'_m, \sigma)))}}$$

where we use the following definitions,

$$e\bar{val} : Decl\_tuple \times State \to Partial\_value\_tuple$$
$$e\bar{val}(\langle d_1, \ldots, d_m \rangle, \sigma) = \langle eval(d_1, \sigma), \ldots, eval(d_m, \sigma) \rangle$$

where,

$$eval(!f, \sigma) = f(\sigma)$$
$$eval(?x, \sigma) = ?x$$

So, $e\bar{val}$ evaluates the elements of a vector of data passing declarations (where $Decl\_tuple$ is the set of all such vectors) according to a current data state. It generates a vector of type $Partial\_value\_tuple$, in which all elements are values or variables to which the synchronisation will assign a value (indicated as $?x$). Next, $\doteq$ ensures corresponding elements of two vectors of data passing declarations match in order that synchronisation can occur.

$$\doteq : Partial\_value\_tuple \times Partial\_value\_tuple \to Boolean$$

$$\langle u_1, \ldots, u_m \rangle \doteq \langle u'_1, \ldots, u'_m \rangle \text{ iff } \forall j (1 \leq j \leq m) \cdot (\neg(u_j = ?x \wedge u'_j = ?y) \wedge ((u_j \neq ?x \wedge u'_j \neq ?y) \Rightarrow u_j = u'_j))$$

Next, # generates the vector of values that is the data result of the synchronisation, thereby annotating the transition generated by the communication.

$$\# : Partial\_value\_tuple \times Partial\_value\_tuple \to Value\_tuple$$
$$\#(\langle u_1, \ldots, u_m \rangle, \langle u'_1, \ldots, u'_m \rangle) = \langle get\_val(u_1, u'_1), \ldots, get\_val(u_m, u'_m) \rangle$$
$$\text{where, } get\_val(v, u) = v, \; get\_val(?x, v) = v$$

Finally, the function α updates a data state according to the data passing effects of a synchronisation, i.e. the assignments of variables to values resulting from $?x$ terms.

$$\alpha : \mathit{Partial\_value\_tuple} \times \mathit{Partial\_value\_tuple} \times \mathit{State} \to \mathit{State}$$
$$\forall\, x \in \mathit{Name} \cdot \alpha(\langle u_1, \ldots, u_m\rangle, \langle u'_1, \ldots, u'_m\rangle, \sigma)(x)$$
$$= \begin{cases} v & \text{if} \quad \exists j (1 \le j \le m) \cdot ((u_j = ?x \wedge u'_j = v) \vee (u_j = v \wedge u'_j = ?x)) \\ \sigma(x) & \text{otherwise} \end{cases}$$

A.2. Semantics with time

The semantics for timed DRCA are built directly upon those for untimed DRCA. The following are new definitions that are required.

$$\mathit{tStates} = \mathit{Clocks} \to \mathbb{R}_0^+$$

where $\mathit{tStates}$ is the set of all mappings from clocks to clock values, i.e. the set of all timing states. Then, resets have the following type, where there is one type of reset function for each reset set, i.e. each $C \subseteq \mathit{Clocks}$.

$$\mathit{Resets}_{(C)} = \mathit{tStates} \to \mathit{tStates}$$

where, $r_c \in \mathit{Resets}_{(C)}$ is such that,

$$r_C(\sigma_t) = \sigma'_t \;\; s.t. \;\; \sigma'_t(x) = 0 \;\; \mathit{if} \;\; x \in C \;\; \mathit{and} \;\; \sigma'_t(x) = \sigma_t(x) \;\; \mathit{otherwise}$$

where, $C \subseteq \mathit{Clocks}$, $\sigma_t, \sigma'_t \in \mathit{tStates}$. Thus, all clocks in C are reset to zero. In addition, timing guards are typed as follows,

$$\mathit{tGuards} = \mathit{tStates} \to \mathit{Boolean}$$

and invariants as follows,

$$\mathit{Invar} = \mathit{Loc} \times \mathit{tState} \to \mathit{Boolean}$$

expressing that an automaton can only stay in a location when the invariant holds at the current timing state. An urgency constraint can be associated with any location; these have the following type,

$$\mathit{Urg} = \mathit{Loc} \to \mathit{Boolean}$$

Thus, $\mathit{Urg}(l)$ holds for all urgent locations, and is false for all others. When entered, an urgent location must be immediately left. Finally, to express that time passes in a timing state, we define the following.

$$\forall\, \sigma_t \in \mathit{tState},\, d \in \mathbb{R}_0^+,\, x \in \mathit{Clocks},\, (\sigma_t + d)(x) = \sigma_t(x) + d$$

Inference rules The following rule supercedes the first inference rule from the untimed semantics.

$$\frac{(i, l) \xrightarrow{g_t, g, \tau, E, r_c} (i, l') \quad g_t(\sigma_t) \quad g(\sigma) \quad \bar{V}_n[i] = l}{\bar{V}_{n,\sigma,\sigma_t} \xrightarrow{\tau} \bar{V}^{i \to l'}_{n, E(\sigma), r_C(\sigma_t)}} \; (1 \le i \le n)$$

where notation is as per the corresponding untimed rule, apart from the following additions, $g_t \in \mathit{tGuards}$ is a constraint on timing states; $\sigma_t \in \mathit{tStates}$ is a timing state; $r_C \in \mathit{Resets}_{(C)}$ is a reset set on clocks in C.

The following new rule is added, which defines time progress.

$$\frac{\forall i (1 \le i \le n) \cdot \bar{V}_n[i] = l \quad \Rightarrow \quad \neg \mathit{Urg}(l) \quad \wedge \quad \forall\, d' \le d \cdot \mathit{Invar}(l)(\sigma_t + d')}{\bar{V}_{n,\sigma,\sigma_t} \xrightarrow{d} \bar{V}_{n,\sigma,(\sigma_t + d)}} \; (d \in \mathbb{R}^+)$$

So, the network of automata can pass time if no locations are urgent, and if in passing that time, no location invariant becomes false. Then, the following rule supercedes the untimed synchronisation rule.

$$\frac{(i,l) \xrightarrow{g_t, g, e\bar{d}_m, E, r_C} (i,l') \quad (j,k) \xrightarrow{g'_t, g', e\bar{d}'_m, E', r'_C} (j,k') \quad i \neq j \quad \bar{V}_n[i] = l \quad \bar{V}_n[j] = k \quad (g_t \wedge g'_t)(\sigma_t) \quad (g \wedge g')(\sigma) \quad \bar{u}_m = e\bar{v}al(\bar{d}_m, \sigma) \quad \bar{u}'_m = e\bar{v}al(\bar{d}'_m, \sigma) \quad \bar{u}_m \doteq \bar{u}'_m}{\bar{V}_{n,\sigma,\sigma_t} \xrightarrow{e.\#(\bar{u}_m, \bar{u}'_m)} \bar{V}^{i\to l', j\to k'}_{n, E(E'(\alpha(\bar{u}_m, \bar{u}'_m, \sigma))), r_{(C\cup C')}(\sigma_t)}}$$

This rule extends the corresponding untimed rule by requiring that time guards are satisfied and all relevant clocks are reset when the networks of automata takes its transition.

B. Proof of equation 6

When $N = 0$ this equation is trivial. And, we prove Eq. (6) by induction for cases where $N > 0$. Note, we can assume that $ge = 0$ and $gi = 0$, since the equation is only applied when this is the case.

Base case : show that Eq. (6) holds for $N = 1$.

$$\begin{aligned} & Vm_k(t+1) \\ &= Vm_k(t) + (El - Vm_k(t))[(-1)^2(dt_{Vm}\, gl - 1) + 1] \qquad \text{(due to Eq. 6)} \\ &= Vm_k(t) + dt_{Vm}\, gl(El - Vm_k(t)) \end{aligned}$$

The above holds when $ge = 0$ and $gi = 0$ in Eq. (2), which is as required.

Inductive step we need to show that if Equation 6 holds for N then it also holds for $N + 1$. So, assume Equation 6 holds for N; we have the following.

$$\begin{aligned} & Vm_k(t+N+1) \\ &= Vm_k(t+N) + dt_{Vm} gl[El - Vm_k(t+N)] \qquad \text{(due to Eq. 2, and since } ge = 0 \text{ and } gi = 0) \\ &= Vm_k(t) + (El - Vm_k(t))[(-1)^{N+1}(dt_{Vm} gl - 1)^N + 1] \\ &\quad + dt_{Vm} gl[El - [\,Vm_k(t) + (El - Vm_k(t))[(-1)^{N+1}(dt_{Vm} gl - 1)^N + 1]]] \\ &\quad \text{(due to Eq. 6, under the inductive hypothesis)} \\ &= Vm_k(t) + (El - Vm_k(t))[(-1)^{N+1}(dt_{Vm} gl - 1)^N + 1] \\ &\quad + dt_{Vm} gl(El - Vm_k(t)) - dt_{Vm} gl(El - Vm_k(t))[(-1)^{N+1}(dt_{Vm} gl - 1)^N + 1] \\ &= Vm_k(t) + dt_{Vm} gl(El - Vm_k(t)) \\ &\quad + (1 - dt_{Vm} gl)(El - Vm_k(t))[(-1)^{N+1}(dt_{Vm} gl - 1)^N + 1] \\ &= Vm_k(t) + dt_{Vm} gl(El - Vm_k(t)) \\ &\quad + (dt_{Vm} gl - 1)(El - Vm_k(t))[(-1)^{N+2}(dt_{Vm} gl - 1)^N - 1] \\ &= Vm_k(t) + dt_{Vm} gl(El - Vm_k(t)) \\ &\quad + (El - Vm_k(t))[(-1)^{N+2}(dt_{Vm} gl - 1)^{N+1} - (dt_{Vm} gl - 1)] \\ &= Vm_k(t) + dt_{Vm} gl(El - Vm_k(t)) \\ &\quad + (El - Vm_k(t))[(-1)^{N+2}(dt_{Vm} gl - 1)^{N+1} + 1] - dt_{Vm} gl(El - Vm_k(t)) \\ &= Vm_k(t) + (El - Vm_k(t))[(-1)^{N+2}(dt_{Vm} gl - 1)^{N+1} + 1] \end{aligned}$$

Thereby showing that indeed Eq. (6) holds for $N + 1$. Since both the base case and the inductive step hold, by mathematical induction, Equation (6) holds for all natural numbers N.

Acknowledgements

LS is partly funded by School of Computing at University of Kent and the NIHR Biomedical Research Centre and Biomedical Research Unit in Dementia based at Cambridge University Hospitals NHS Foundation Trust and the University of Cambridge.

References

[AIS08] Aisa B, Mingus B, O'Reilly R (2008) The emergent neural modeling system. Neural Netw 21:1146–1152

[ARI03] Arik S (2003) Global robust stability of delayed neural networks. IEEE Trans Circuits Syst I Fundam Theory Appl 50:156–160

[ART03] Artho C, Drusinsky D, Goldberg A, Havelund K, Lowry M, Pasareanu C, Rosu G, Visser W (2003) Experiments with test case generation and runtime analysis. In: Proceedings of the abstract state machines (ASM'03). Lecture notes in computer science, vol 2589, pp 87–107

[BAK00] Bakker R, Schouten JC, Giles CL, Takens F, van den Bleek CM (2000) Learning chaotic attractors by neural networks. Neural Comput 12:2355–2383

[BAB04] Barringer H, Goldberg A, Havelund K, Sen K (2004) Rule-based runtime verification. In: Proceedings of the 5th international conference on verification, model checking and abstract interpretation. Lecture Notes in Computer Science, vol 2937, pp 44–57

[BAR85] Barnard P (1985) Interacting cognitive subsystems: a psycholinguistic approach to short-term memory. In: Ellis A (ed) Progress in the psychology of language, vol 2, pp 197–258. Lawrence Erlbaum Associates Ltd., Hove

[BAR04] Barringer H, Goldberg A, Havelund K, Sen K (2004) Rule-based runtime verification. In: Proceedings of the 5th international conference on verification, model checking and abstract interpretation. Lecture Notes in Computer Science, vol 2937, pp 44–57

[BEH04] Behrmann G, David A, Larsen KG (2004) A tutorial on Uppaal. In: Proceedings of the 4th international school on formal methods for the design of computer, communication, and software systems (SFM-RT'04). Lecture notes in computer science, vol 3185, pp 200–236

[BEN95] Bengtsson J, Larsen KG, Larsson F, Pettersson P, Yi W (1995) UPPAAL—a tool suite for automatic verification of real-time systems. Hybrid Syst 232–243

[BLY11] Boly M, Garrido MI, Gosseries O, Bruno MA, Boveroux P, Schnakers C, Massimini M, Litvak V, Laureys S, Friston K (2011) Preserved feedforward but impaired top–down processes in the vegetative state. Science 13:858–862

[BOL88] Bolognesi T, Brinksma E (1988) Introduction to the ISO specification language LOTOS. Comput Netw ISDN Syst 14910:25–29

[BOW99] Bowman H, Faconti G (1999) Analysing cognitive behaviour using LOTOS and Mexitl. Form Asp Comput 11(2):132–159

[BOW01] Bowman H, Derrick J (eds) (2001) Formal methods for distributed processing, a survey of object oriented approaches. Cambridge University Press, Cambridge

[BOW02] Bowman H, Steen MWA, Boiten EA, Derrick J (2002) A formal framework for viewpoint consistency. Form Methods Syst Des 21(2):111–166

[BOW06] Bowman H, Gomez RS (2006) Concurrency theory, calculi and automata for modelling untimed and timed concurrent systems. Springer, New York

[BOW07] Bowman H, Wyble B (2007) The simultaneous type, serial token model of temporal attention and working memory. Psychol Rev 114(1):38–70

[BOW08] Bowman H, Wyble B, Chennu S, Craston P (2008) A reciprocal relationship between bottom-up trace strength and the attentional blink bottleneck: relating the LC-NE and ST2 models. Brain Res 1202:25–42

[CLA00] Clarke EM, Grumberg O, Peled DA (2000) Model Checking. The MIT Press, Cambridge

[DAV01] Attwell D, Laughlin SB (2001) An energy budget for signaling in the grey matter of the brain. J Cereb Blood Flow Metab 21:1133–1145

[DER01] Derrick J, Boiten E (2001) Refinement in Z and object-Z: foundations and advanced applications. Springer, Berlin

[DES95] Desimone R, Duncan J (1995) Neural mechanism of selective visual attention. Annu Rev Neurosc 18:193–222

[DIB10] Dibner C, Schibler U, Albrecht U (2010) The mammalian circadian timing system: organization and coordination of central and peripheral clocks. Annu Rev Physiol 72:517–549

[FOD88] Fodor JA, Pylyshyn ZW (1988) Connectionism and cognitive architecture: a critical analysis. Cognition 28:3–71

[FRI00a] Friston KJ (2000) The labile brain. I. Neuronal transients and nonlinear coupling. Philos Trans R Soc B Biol Sci 355(1394):215–236

[FRI00b] Friston KJ (2000) The labile brain. II. Transients, complexity and selection. Philos Trans R Soc B Biol Sci 355(1394):237–252

[FRI00c] Friston KJ (2000) The labile brain. III. Transients and spatio-temporal receptive fields. Philos Trans R Soc B Biol Sci 355(1394):253–265

[GAR02] Garcez AS, Broda KB, Gabbay DM (2002) Neural-symbolic learning systems, foundations and applications. Springer, New York

[GIA02] Giannakopoulou D, Pasareanu C, Barringer H (2002) Assumption generation for software component verification. In: Proceedings of the 17th IEEE international conference on automated software engineering, pp 3–12

[GRO76] Grossberg S (1976) Adaptive pattern classification and universal recoding, I: parallel development and coding of neural feature detectors. Biol Cybern 23:121–134

[GRO12] Grossberg S (2012) Adaptive resonance theory: how a brain learns to consciously attend, learn, and recognize a changing world. Neural Netw 37:1–47

[HAM98] Hamey LGC (1998) XOR has no local minima: a case study in neural network error surface analysis. Neural Netw 11:669–681

[HAV00] Havelund K, Lowry M, Park S, Pecheur C, Penix J, Visser W, White JL (2000) Formal analysis of the remote agent before and after flight. In: Proceedings of 5th NASA Langley formal methods workshop, pp 13–15

[HEN94] Henzinger TA, Nicollin X, Sifakis J, Yovine S (1994) Symbolic model checking for real-time systems. Inf Comput 111(2):193–244

[HOD52] Hodgkin A, Huxley A (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117:500–544

[HOL03] Holzmann G (2003) The SPIN model checker: primer and reference manual. Addison-Wesley, Boston

[KIC97] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes CV, Loingtier JM, Irwin J (1997) Aspect-oriented programming. In: Lecture notes in computer science, vol 1241. Springer-Verlag, New York

[KOL91] Kolen JF, Pollack JB (1991) Back propagation is sensitive to initial conditions. In Lippmann RP, Moody JE, Touretzky DS (eds) Advances in neural information processing systems, vol 3, pp 860–867
[LIL94] Liljenström H (1994) Cognition, neurodynamics, and computer models. Connectionism in a broad perspective. Ellis Horwood, Chichester
[MCM93] McMillan KL (1993) Symbolic model checking. Springer, New York
[MEN05] Menzies T, Pecheur C (2005) Verification and validation and artificial intelligence. Adv Comput 65:154–203
[MOL90] Moller F, Tofts CMN (1990) A temporal calculus of communicating systems. In: CONCUR'90, pp 401–415
[NAP99] Napolitano MR, Molinaro G, Innocenti M, Seanor B, Martinelli D (1999) A complete hardware package for a fault-tolerant flight control system using on-line learning neural networks. In: Proceedings of the American Control Conference, pp 2615–2619
[NIE05] Nieuwenhuis S, Gilzenrat MS, Holmes BD, Cohen JD (2005) The role of the locus coeruleus in mediating the attentional blink: a neurocomputational theory. J Exp Psychol Gen 134:291–307
[OLS97] Olshausen BA, Field DJ (1997) Sparse coding with an overcomplete basis set: a strategy employed by V1? Vis Res 37(23):3311–3325
[ORE96] O'Reilly RC (1996) Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. Neural Comput 8:895–938
[ORE00] O'Reilly RC, Munakata Y (2000) Computational explorations in cognitive neuroscience: understanding the mind by simulating the brain. MIT Press, Cambridge
[PNU77] Pnueli A (1977) The temporal logic of programs. In: Proceedings of the 18th IEEE symposium foundations of computer science, FOCS 1977, pp 46–57
[ROD01] Rodrigues P, Costa JF, Siegelmann HT (2001) Verifying properties of neural networks. Artificial and natural neural networks. In: Lecture notes in computer science, vol 2084, pp 158–165, Springer, New York
[ROL02] Rolls ET, Deco G (2002) Computational neuroscience of vision. Oxford University Press, Oxford
[ROS98] Roscoe AW (1998) The theory and practice of concurrency. Prentice-Hall, London
[RUK09] Rukšenas R, Back RJ, Curzon P, Blandford A (2009) Verification-guided modelling of salience and cognitive load. Form Asp Comput 21:541–569
[SMI92] Smith LS (1992) A framework for neural net specification. IEEE Trans Softw Eng 18(7):601–612
[SPR96] Sprinkhuizen-Kuyper IG, Boers EJW (1996) The error surface of the simplest XOR network has only global minima. Neural Comput 8:1301–1320
[SBB07] Su L, Bowman H, Barnard PJ (2007) Attentional capture by meaning, a multi-level modelling study. In: Proceedings of 29th annual meeting of the Congitive Science Society, Nashville, pp 1521–1526
[SBB09] Su L, Bowman H, Barnard PJ, Wyble B (2009) Process algebraic modelling of attentional capture and human electrophysiology in interactive systems. Form Asp Comput 21(6):513–539
[SBB11] Su L, Bowman H, Barnard P (2011) Glancing and then looking: on the role of body, affect, and meaning in cognitive control. Front Psychol 2:348 doi:10.3389/fpsyg.2011.00348
[SUT89] Sutherland RJ, Rudy JW (1989) Configural association theory: the role of the hippocampal formation in learning, memory, and amnesia. Psychobiology 17(2):129–144
[TIP91] Tipper SP, Driver J, Weaver B (1991) Object-centred inhibition of return of visual attention. Q J Exp Psychol A 43:289–298

Received 3 June 2011
Revised 23 September 2013
Accepted 18 December 2013 by David Duce
Published online 23 April 2014