2017-01-01

A modular architecture for transparent computation in recurrent neural networks

Carmantini, GS

10.1016/j.neunet.2016.09.001
Neural Networks

A modular architecture for transparent computation in Recurrent Neural Networks

Giovanni S. Carmantini[1], Peter beim Graben[2], Mathieu Desroches[3], and Serafim Rodrigues[1]

[1]School of Computing and Mathematics, Plymouth University,, Plymouth, United Kingdom
[2]Bernstein Center for Computational Neuroscience Berlin,, Humboldt-Universität zu Berlin, Berlin, Germany
[3]Inria Sophia-Antipolis Méditerranée, Valbonne, France

Abstract

Computation is classically studied in terms of automata, formal languages and algorithms; yet, the relation between neural dynamics and symbolic representations and operations is still unclear in traditional eliminative connectionism. Therefore, we suggest a unique perspective on this central issue, to which we would like to refer as to transparent connectionism, by proposing accounts of how symbolic computation can be implemented in neural substrates. In this study we first introduce a new model of dynamics on a symbolic space, the versatile shift, showing that it supports the real-time simulation of a range of automata. We then show that the Gödelization of versatile shifts defines nonlinear dynamical automata, dynamical systems evolving on a vectorial space. Finally, we present a mapping between nonlinear dynamical automata and recurrent artificial neural networks. The mapping defines an architecture characterized by its granular modularity, where data, symbolic operations and their control are not only distinguishable in activation space, but also spatially

localizable in the network itself, while maintaining a distributed encoding of symbolic representations. The resulting networks simulate automata in real-time and are programmed directly, in absence of network training. To discuss the unique characteristics of the architecture and their consequences, we present two examples: i) the design of a Central Pattern Generator from a finite-state locomotive controller, and ii) the creation of a network simulating a system of interactive automata that supports the parsing of garden-path sentences as investigated in psycholinguistics experiments.

1 Introduction

The relation between symbolic computation and neural dynamics is one of the most pertinent problems in computational neuroscience, artificial intelligence, and cognitive science. On the one hand, symbolic computation is generically codified in terms of production systems, formal languages, algorithms and automata [Hopcroft and Ullman, 1979]. On the other hand, neural dynamics in artificial neural networks (ANN) is described by nonlinear evolution laws [Hertz et al., 1991]. Approaches to connect these different realms of research go back to the seminal paper of McCulloch and Pitts [1943] on networks of idealized two-state neurons that behave as logic gates. Furthermore, fundamental work by Kleene [1956] and Minsky [1967] demonstrated the equivalence between such networks and finite-state automata, and thus digital computers (which are essentially large-scale networks of logic gates). Later examples for connectionist modeling of symbolic computation are the speech perception and production models *TRACE* by McClelland and Elman [1986] and *NETtalk* by Sejnowski and Rosenberg [1987]. A further important step was achieved by Elman when introducing simple recurrent networks (SRN) as prediction devices for letters in words [Elman, 1990] and syntactic categories in sentences [Elman, 1995]. SRN found a number of successful applications in linguistics and cognitive science [Christiansen and Chater, 1999, Farkas and Crocker, 2008, Lawrence et al., 2000, Tabor et al., 1997] where formal grammars have been employed for the generation of training sets. After training, grammatical relations emerged in the connectivity and activation patterns of the network's hidden layer which could be examined through clustering and principal component analysis (PCA).

A key problem of this and similar approaches based on *eliminative con-*

nectionism [Blutner, 2011], a theoretical stance aiming at the elimination of symbolic representations in connectionist models, is that the emerging representations, while comparable in a metric space through empirical methods such as clustering or PCA, do not allow inferences about the syntactic or structural relationships of the symbolic training data. This is even more the case with contemporary deep-learning [Bengio et al., 2013, Li, 2014], and reservoir computing approaches featuring large networks of randomly and recurrently connected nonlinear units [Dominey, 1995, Jaeger, 2001, Maass et al., 2002, Steil, 2004]. For that reason, another branch of research, which we may call *transparent connectionism*, has been developed in the framework of vector symbolic architectures (VSA) [beim Graben and Potthast, 2009, Gayler, 2006, Gayler et al., 2010, Mizraji, 1989, Smolensky, 1990, Smolensky and Legendre, 2006a,b]. Here, one explicitly starts with the symbolic data structures and processes, which are first decomposed into so-called filler-role bindings and then used to create vectorial images through tensor product representations [beim Graben and Potthast, 2009, Smolensky, 1990]. These serve as training patterns for subsequent connectionist modeling. In contrast to eliminative connectionism where representations that emerge during training are to a great extent opaque, representations in VSAs are completely transparent as they can be resolved in each step of the encoding procedure. Depending on the structure of the chosen vector space one arrives at different kinds of integrated connectionist/symbolic architectures (ICS) [Smolensky, 1990, Smolensky and Legendre, 2006a,b]: Gödelizations for one-dimensional representations in the field of real numbers, proper vectorial representations for finite-dimensional vector spaces, and functional representations for infinite-dimensional vector spaces [beim Graben and Potthast, 2009]. Importantly, Siegelmann and Sontag [1991, 1995] used a combination of Gödelization and localist finite-dimensional representation to prove that Recursive ANNs (R-ANN) with rational weights and ramp activation functions can simulate any n-tape ($n \geq 2$) stack machine – or, equivalently, any Turing machine (TM) and any partial recursive function – when endowed with a specific localist architecture. Moreover, Siegelmann and Sontag showed that a R-ANN consisting of 886 units can simulate a universal Turing machine (UTM). Recent work by Cabessa [Cabessa and Siegelmann, 2012, Cabessa and Villa, 2012, 2013] extends these results on R-ANNs to the realm of interactive computation [Wegner, 1998], a framework studying systems that can interact with the environment throughout their computation (as opposed to the framework of classical computation, where the interaction

is limited to the input-output exchange), proving that R-ANNs are equivalent in power to interactive TMs.

Very-large-scale and reservoir-like neural network approaches can also rely on VSA as a key ingredient, as in the *neural engineering* framework [Eliasmith et al., 2012, Stewart et al., 2014], which employs semantic pointers for addressing symbolic representations in activation space, and recent work at the interface between *reservoir computing* and connectionist/symbolic approaches [Hinaut and Dominey, 2013, Hinaut et al., 2014]

In contrast, the present work focuses on parsimonious VSA implementations, building upon the seminal results from Siegelmann and Sontag [1991, 1995], and work from Moore [1990, 1991] who has shown that nonlinear dynamical automata (NDA), piecewise-affine linear dynamical systems on the unit square, can simulate the dynamics of any TM in real-time[1] when the machine is represented as a generalized shift (GS) on dotted sequences. In this work we first extend Moore's results by showing that NDA can support the real-time simulation of a range of models of computation, including but not limited to Turing Machines (of course, TMs can simulate any other model of computation of lesser or equal power, but not necessarily in real-time; see Section 2.1.1 for a discussion). We achieve this by relaxing the definition of GS, which leads to a novel and more expressive shift map, the versatile shift (VS) which enables the parsimonious and real-time emulation of symbolic computation in a range of models. We then show that VS dynamics can be mapped to NDA dynamics on the unit square through Gödelization. Finally, we present a mapping between VS and R-ANNs through NDA (extending preliminary results shown in Carmantini et al., 2015).

Symbolic models of computation distinguish between data, operations on data and the control of these operations. For example, automata implement a set of symbolic operations and its control through a look-up table (the transition function), and the data as a string encoding the so-called *configuration* of the automaton. In grammars and term rewriting systems, operations are instead defined as a set of substitution/rewriting rules on some symbolic string, where the application of these rules is controlled by a set of conditions. NDA can perform symbolic computation on a vectorial space while preserving, in their formulation, the division between data, operations on data, and their control. Basing our construction on NDA, we derive an ar-

[1] In a real-time simulation, a single computation step in the original model is mapped to a single computation step in the model simulating it.

chitecture that also preserves this division, thus obtaining networks that are transparent, modular and parsimonious. Importantly, the operations embedded within the architecture we propose herein are not only distinguishable in activation space, but are also spatially localized, while still relying on a distributed representation of the symbolic data. The granular modularity of the architecture brought about by its relation with NDA differentiates our approach from previous work, and has important consequences for the constructive mapping of interactive automata networks (IANs) to R-ANNs, and for the possibility of correlational studies with electrophysiological data, which we will discuss in subsequent Sections.

We illustrate our approach by means of two examples. As a first example, we construct a central pattern generator (CPG) from a finite-state automaton for gait patterns of quadruped animals [Collins and Richmond, 1994, Golubitsky et al., 1999, Grillner and Zangger, 1975]. The neuronal sequential activations by CPGs are usually modeled through networks of coupled nonlinear oscillators that undergo symmetry-breaking bifurcations under changes in their driving input [Collins and Richmond, 1994, Golubitsky et al., 1998, 1999, Schöner et al., 1990]. We show that our construction, although symbolically inspired, allows the investigation of similar bifurcation scenarios. Additionally, the results of these example are relevant to the design of CPGs for the control of robotic locomotion [Ijspeert, 2008]. As a second example, we show how our approach is ideally suited to tackle the mapping of interactive machines to neural networks, because of the separation in the network architecture of data, transformations and their control. This makes it straightforward to construct R-ANNs simulating networks of automata that e.g. share states, are organized in complex hierarchies, or are bound by interactions of conditions in the application of symbolic transformations. We demonstrate this by constructing an interactive automata network (IAN) that implements a diagnosis and repair parser for syntactic language processing [Lewis, 1998] and by subsequently mapping it to a R-ANN performing the same computation. We are then able to derive vectorial observables from the network; specifically, we compute synthetic event-related brain potentials (synth-ERPs, Barrès et al., 2013) and discuss their relation with event-related potentials as measured in experiments involving garden-path sentences [Frisch et al., 2004].

Abbreviation	Extended name
ANN	Artificial neural network
BSL	Branch selection layer
CFG	Context-free grammar
CL	Configuration layer
CPG	Central pattern generator
EEG	Electroencephalography
ERP	Event-related brain potentials
FSM	Finite-state machine
GS	Generalized shift
LFP	Local field potentials
LTL	Linear transformation layer
MCL	Machine configuration layer
NDA	Nonlinear dynamical automaton
PCA	Principal component analysis
PDA	Push-down automaton
R-ANN	Recurrent artificial neural network
SRN	Simple recurrent network
synth-ERP	Synthetic event-related brain potential
TDR	Top-down recognizer
TM	Turing machine
UTM	Universal Turing machine
VS	Versatile shift
VSA	Vector symbolic architecture

Table 1: **List of abbreviations used in this paper.**

2 Methods

The present Section outlines our general method which allows the mapping of a range of models of computation to R-ANNs. In Figure 1 we summarize the complete mapping procedure to accompany its exposition. Our construction is a two-step process. We first define a Versatile shift (a generalization of the shift map introduced in Moore, 1990) that emulates some model of computation, and we subsequently encode its dynamics on the unit square via Gödelization, obtaining a two-dimensional piecewise affine-linear map on the unit square, i.e. a NDA. As a second step, the NDA is mapped onto a first-order R-ANN, which is endowed with an architecture that captures the NDA's three key components: i) a state, encoding the symbolic data of the model of computation; ii) a set of affine-linear transformations, encoding its operations on data; iii) a switching rule that selects the relevant affine-linear transformation to apply given the state, thus implementing the control of the symbolic operations.

Next, the theoretical methods employed are discussed in detail. In the presentation of various objects from Formal Language Theory and Automata Theory, we essentially follow the well-established definitions in Hopcroft and Ullman [1979], and in Sipser [2006].

2.1 Elements of Symbolic Computation

A symbol is meant to be a distinguished element from a finite set $\mathbf{A}$, which we call an *alphabet*. Symbols can be concatenated, i.e. for $a, b \in \mathbf{A}$, $ab \equiv (a, b) \in \mathbf{A}^2$. A sequence of symbols $w \in \mathbf{A}^n$ is called a word of length n, denoted $n = |w|$. The set of words of all possible lengths w of finite length $|w| \geq 0$ is denoted $\mathbf{A}^*$ (for $|w| = 0$, $w = \epsilon$ denotes the "empty word").

2.1.1 From Generalized to Versatile Shifts

The theory of symbolic dynamics [Lind and Marcus, 1995] is a tool to study dynamical systems based on the discretization of time and space in order to interpret trajectories in a vectorial space as discrete sequences of infinite strings of symbols. Importantly, its theoretical apparatus can also be used to do the opposite, mapping sequences of strings to a vectorial space. We start by redefining a representation for strings of symbols, the dotted sequence.

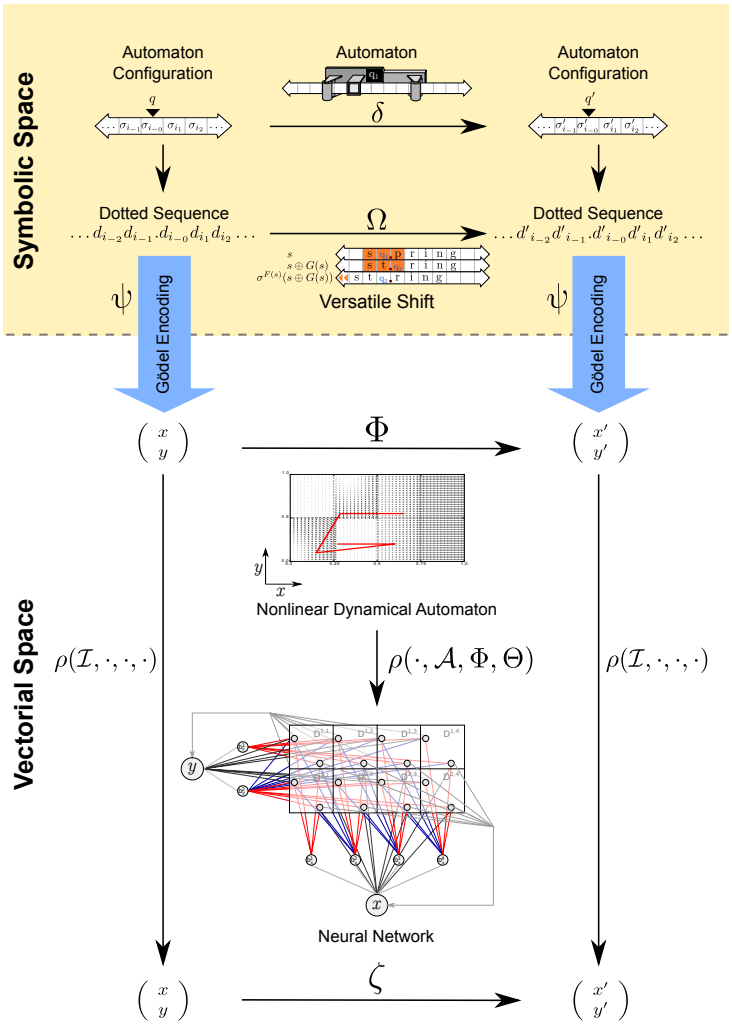

Figure 1: **An automaton is mapped to a recurrent artificial neural network (R-ANN).** The representation of machine configurations as dotted sequences allows for the mapping of the machine transition function to the action of a Ω versatile shift (VS) map upon said sequences, simulating the computation performed by the automaton. A Gödel encoding ψ acts as a bridge between the Symbolic and the Vectorial representation of the automaton's dynamics, and enables the representation of Ω as an affine-linear map Φ by a nonlinear dynamical automaton (NDA). Finally, a map ρ generates a R-ANN, with a specific network architecture and internal dynamics ζ that operates on the same Vectorial space as Φ, where the NDA states are identically mapped through $\rho(I,\cdot,\cdot,\cdot)$ to the activation of a specialized layer in the R-ANN.

According to Moore [1990, 1991], a *dotted sequence* $s \in \mathbf{A}^{\mathbb{Z}}$ on an alphabet $\mathbf{A}$ is a two-sided infinite sequence of symbols "$s = \ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ d_2\ \ldots$" where $d_i \in \mathbf{A}$, for all indices $i \in \mathbb{Z}$. Here, the dot "." is simply used as a mnemonic sign, indicating that the index 0 is to its right. A shift space $M_S = (\mathbf{A}^{\mathbb{Z}}, \sigma)$ is then given by a shift map $\sigma : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ [Lind and Marcus, 1995], such that $\sigma(s)_i = (s)_{i+1}$, i.e. σ shifts all symbols in s one place to the left (or, equivalently, shifts the dot one place to the right). Similarly, it is possible to define an inverse to the shift map, σ^{-1}, shifting all symbols in s one place to the right (or, equivalently, the dot one place to the left).

Notice how shifting the dot in a dotted sequence to the left or the right resembles the movement of the read-write head of a Turing machine on its tape (see section 2.1.2 for more details on Turing machines). In order to fully attain the power of Turing machines, Moore [1990, 1991] endows the shift space M_S with three additional maps

$$\begin{aligned} F &: \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \\ \oplus &: \mathbf{A}^{\mathbb{Z}} \times (\mathbf{A} \cup \{\phi\})^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}} \\ G &: \mathbf{A}^{\mathbb{Z}} \to (\mathbf{A} \cup \{\phi\})^{\mathbb{Z}}, \end{aligned} \tag{1}$$

such that their composition $\Omega(s) = \sigma^{F(s)}(s \oplus G(s))$ can fully simulate any Turing machine. The augmented shift space $M_{GS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$ is called *generalized shift* (GS) if there is an open interval of indices around the dot, called *Domain of Dependence* DoD $= (k_l, k_r)$ $(k_l \leq 0 \leq k_r)$, such that $F(s)$ and $G(s)$ only depend on the content of s within the DoD, $F(s)$ determines a number of left shifts $(F(s) > 0)$, right shifts $(F(s) < 0)$, or no shift at all $(F(s) = 0)$ and $G(s)$ maps the symbols s_i within the DoD onto other symbols g_i, while all symbols outside the DoD are mapped onto an auxiliary symbol ϕ. Finally, the composition operator overwrites all symbols s_i within the DoD through their images g_i under G while not changing s outside the DoD, i.e. $(s \oplus g)_i = s_i$ if $g_i = \phi$, but $(s \oplus g)_i = g_i$ if $g_i \neq \phi$.[2]

According to Moore's proof [Moore, 1990, 1991], any Turing machine can be realized as a GS M_{GS}. Since Turing machines can be programmed to simulate the computation carried out by any model of lower or equal compu-

[2]In his 1991 paper, Moore actually defines the DoD of a GS as a finite set of integers which need not be consecutive, and introduces a second finite set of integers, the *Domain of Effect* (DoE) to indicate the cells to be rewritten (as a function of the cells in the DoD). Nevertheless, it is always possible, given any GS with arbitrary DoD and DoE, to construct an equivalent GS as defined here; we thus decided to propose a simplified definition.

tational power, such as finite-state automata or push-down automata, this implies that these can also be described in terms of equivalent GSs. In practice, however, simulating other automata via Turing machines will lead to rather complicated machine tables even for the simplest symbolic algorithms, and thus to unnecessarily complicated shift spaces. In fact, different automata implement different atomic operations, so that a Turing machine can require multiple computation steps to simulate a single computation step of another automaton, even when the automaton is computationally less powerful. Therefore, we introduce a novel shift space to which we shall henceforth refer as versatile shift (VS), which will allow us to represent automata configuration dynamics on dotted sequences in a more straightforward and parsimonious fashion, simulating it in real-time. Our construction essentially relies on a redefinition of the concept of dotted sequence. Above, the dot was only used as a mnemonic symbol without any functional implication. Now, we introduce the dot as a meta-symbol which can be concatenated with two words $v_1, v_2 \in \mathbf{A}^*$ through $v = v_1.v_2$. Let $\hat{\mathbf{A}}^*$ denote the set of these dotted words. Moreover, let $\mathbb{Z}^- = \{i \mid i < 0,\ i \in \mathbb{Z}\}$ and $\mathbb{Z}^+ = \{i \mid i \geq 0,\ i \in \mathbb{Z}\}$ the sets of negative and non-negative indices. We can then reintroduce the notion of a dotted sequence as follows. Let $s \in \mathbf{A}^{\mathbb{Z}}$ be a bi-infinite sequence of symbols such that $s = w_\alpha v w_\beta$ with $v \in \hat{\mathbf{A}}^*$ as a dotted word $v = v_1.v_2$ and $w_\alpha v_1 \in \mathbf{A}^{\mathbb{Z}^-}$ and $v_2 w_\beta \in \mathbf{A}^{\mathbb{Z}^+}$. Through this definition, the indices of s are inherited from the dotted word v and are thus not explicitly prescribed. Whereas GSs can only rewrite each symbol in their DoD with a new one, VSs are endowed with a more general rewriting operation, substituting dotted words in their DoD with other dotted words of equal or different lengths (as already hinted, yet not implemented, by Moore, 1990). This adds expressiveness to VSs, allowing for the parsimonious real-time simulation of a range of automata (see Figure 2 for a pictorial representation of the difference in substitution operations between GSs and VSs).

More formally, we define a VS as a pair $M_{VS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences, and $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}(s \oplus G(s)) \tag{2}$$

with

$$\begin{aligned} &F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \\ &\oplus : \mathbf{A}^{\mathbb{Z}} \times \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}} \\ &G : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}, \end{aligned} \tag{3}$$

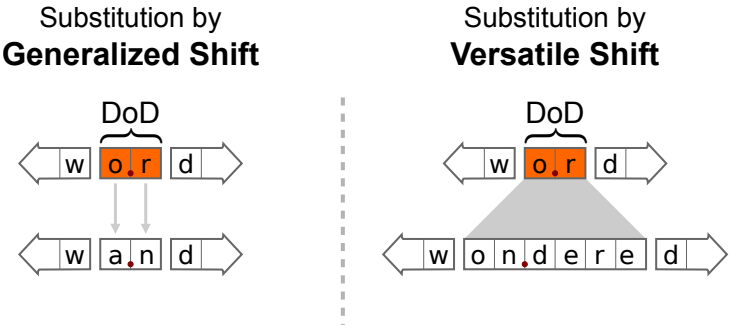

Figure 2: **Difference between substitution operation in generalized and versatile shifts.** In this Figure, we show two example substitutions by respectively a generalized shift and a versatile shift. While a generalized shift can only rewrite each symbol of the dotted word in its Domain of Dependence (DoD) with a new one, a versatile shift can substitute the dotted word in its DoD with any other arbitrary dotted word.

where the operator "$\oplus$" substitutes the dotted word $v_1.v_2 \in \hat{\mathbf{A}}^*$ in s with a new dotted word $\hat{v_1}.\hat{v_2} \in \hat{\mathbf{A}}^*$ specified by G, while $F(s) = F|_{\hat{\mathbf{A}}^*}(v_1.v_2)$ determines the number of shift steps as for the GS above. The action of F, G and $\oplus$ in the VS depends on a finite dotted sub-sequence $v_1.v_2$ inside the original dotted sequence $s = w_\alpha v w_\beta$, as determined by the DoD of the VS, again defined as a set of consecutive integers denoting cell positions on the original dotted sequence. The DoD of a GS can be specified by an open interval (k_l, k_r) on the integers, with $k_l \leq 0$ and $k_r \geq 0$. Additionally, for a DoD $= (k_l, k_r)$, it is useful to define $\text{DoD}_\alpha = (k_l, 0)$ and $\text{DoD}_\beta = (-1, k_r)$ to denote the left and right part of the complete DoD on dotted sequences $\alpha.\beta$, with $\text{DoD} = \text{DoD}_\alpha \cup \text{DoD}_\beta$. The set V of dotted words that can appear in the DoD of a VS is a subset of $\hat{\mathbf{A}}^*$, and can be defined as $V = \{v \mid v = v_1.v_2 \in \hat{\mathbf{A}}^*,\ |v_1| = |\text{DoD}_\alpha|,\ |v_2| = |\text{DoD}_\beta|\}$.

To illustrate how VSs act on dotted sequences, consider for example the

dotted sequence "wo.rd", and define a VS $\Omega_{\text{ex}}$with

$$\text{DoD} = (-2, 1) = \{-1, 0\},$$

$$G : \begin{cases} \texttt{o.r} \mapsto \texttt{a.n} \\ \texttt{a.n} \mapsto \texttt{on.dere}, \end{cases}$$

$$F : \begin{cases} \texttt{o.r} \mapsto 0 \\ \texttt{a.n} \mapsto 1, \end{cases}$$

then, applying Ω_{ex} to "wo.rd" once yields

$$\begin{aligned} \Omega_{\text{ex}}(\texttt{wo.rd}) &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wo.rd} \oplus G(\texttt{wo.rd})\big) \\ &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wo.rd} \oplus \texttt{a.n}\big) \\ &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wa.nd}\big) \\ &= \sigma^{0}\big(\texttt{wa.nd}\big) \\ &= \texttt{wa.nd} \end{aligned}$$

and applying it again to the resulting "wa.nd" dotted sequence yields

$$\begin{aligned} \Omega_{\text{ex}}(\texttt{wa.nd}) &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{wa.nd} \oplus G(\texttt{wa.nd})\big) \\ &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{wa.nd} \oplus \texttt{on.dere}\big) \\ &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{won.dered}\big) \\ &= \sigma^{1}\big(\texttt{won.dered}\big) \\ &= \texttt{wo.ndered} \end{aligned}$$

where the DoD of the input string has been highlighted for clarity (again, contrast this with the pictorial representation given in Figure 2). Note that a VS reduces to a GS in the special case when G always substitutes a dotted sequence with one of the same (finite) length in both the left and the right sub-sequences, as in the previous example where $\texttt{wo.rd} \oplus G(\texttt{wo.rd}) = \texttt{wo.rd} \oplus \texttt{a.n} = \texttt{wa.nd}$.

A point worth noting is that endowing VS with the rewriting capability extends the GS in the direction of semi-Thue systems (also known as string rewriting systems), a universal model of computation introduced by Axel Thue in 1914 (see chapter 7 of Davis et al., 1994). These rewriting systems play an important role, for example, in algebraic specifications of abstract data structures, equational programming, program transformation and automated theorem proving, where the conditional and successive application of a finite set of rewrite rules transforms a given symbolic structure.

2.1.2 Simulation of Various Automata by Versatile Shifts

We will now discuss how a range of automata can be simulated in real-time by VSs by choosing appropriate dotted sequence representations of machine configurations, and by constructing F and G to reproduce the machine's operations and their conditional application.

Finite-state machines The finite-state machine (FSM) model of computation has been introduced by McCulloch and Pitts in 1943 and is widely used to describe systems in many application fields, ranging from computer science to engineering and biology, to name a few. At every step of a computation a FSM is in one of a finite set of states, and it can change its state as a result of an incoming input signal. More formally, a FSM can be defined as a 5-tuple $M_{\text{FSM}} = (Q, \mathbf{T}, q_0, F, \delta)$, where Q is a finite set of control states, $\mathbf{T}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subseteq Q$ is a set of accept states, and $\delta : Q \times \mathbf{T} \to Q$ is a transition function defined as follows:

$$\delta : (q_t, d_{0_t}) \mapsto q_{t+1}, \tag{4}$$

where $q_t, q_{t+1} \in Q$ are states, and $d_{0_t} \in \mathbf{T}$ is an input symbol. At each computation step, a FSM reads its current state q_t, consumes (i.e. reads and discards) its current input symbol d_t, and transitions to a new state $q_{t+1} = \delta(q_t, d_t)$ as prescribed by its transition function. It is possible to encode FSM configurations on dotted sequences as

$$q_t \,.\, d_{0_t}\ d_{1_t}\ \ldots\ d_{n_t} \tag{5}$$

where q_t, d_{0_t} and $d_{1_t} \ldots d_{n_t}$ are respectively the state, input symbol, and the rest of the unconsumed input of the FSM at time t. A VS simulating a FSM in real-time can be constructed by defining the Domain of Dependence to be $\text{DoD} = (-2, 1) = \{-1, 0\}$, F to always map to 0, and G so that for all $q_t \in Q, d_t \in \mathbf{T}$:

$$G : q_t.d_{0_t} \mapsto q_{t+1}.\epsilon \tag{6}$$

where $q_{t+1} = \delta(q_t, d_{0_t})$.

Push-down automata and Context-Free Grammars A push-down automaton (PDA) is a computing machine that has sequential access to its input and can manipulate a stack memory by popping and pushing symbols

on top of it. More formally, a PDA can be defined as a 6-tuple $M_{\text{PDA}} = (Q, \mathbf{N}, \mathbf{T}, q_0, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is the stack alphabet, $\mathbf{T}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subseteq Q$ is a set of accept states, and δ is a transition function. If $F = \varnothing$, the PDA accepts its input when both the input tape and the stack are empty, and it is thus said to accept by *empty stack*.

A Deterministic PDA is a PDA in which any configuration of the machine defines at most one transition. As the mapping of non-deterministic automata computation to Neural Networks is outside the scope of this work, in what follows we will only discuss Deterministic PDAs. Determinism will thus be implied from this point on. The transition function of a PDA is defined as follows:

$$\delta : Q \times \mathbf{T} \cup \{\epsilon\} \times \mathbf{N} \to Q \times (\mathbf{N} \cup \{\epsilon\}). \tag{7}$$

At each computation step, a PDA consumes an input symbol, pushes or pops a symbol on the top of its stack, and changes state as prescribed by its transition function applied to the current state q_t, currently read input symbol d_{0_t}, and the current top-of-stack symbol s_{0_t}. In particular, if $s_{0_t} \dots s_{m_t}$ is the current content of the stack, transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, \epsilon)$$

apply a pop operation, such that the new stack content becomes equal to $s_{1_t} \dots s_{m_t}$. Push operations are instead applied by transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, s_{0_{t+1}}),$$

so that the updated stack contains the symbols $s_{0_{t+1}} s_{0_t} \dots s_{m_t}$. Finally, for transitions of the form

$$\delta : (q_t, \epsilon, s_{0_t}) \mapsto (q_{t+1}, \chi),$$

the PDA does not consume any input symbol (i.e. it does not access its input at all), but either pops its top-of-stack, if $\chi = \epsilon$, or pushes symbol χ, if $\chi \in \mathbf{N}$.

PDA configurations can be encoded on dotted sequences as follows:

$$\underbrace{s_{m_t} \ \dots \ s_{0_t}}_{s_t} \ q_t \ . \ \underbrace{d_{0_t} \ \dots \ d_{n_t}}_{d_t} \tag{8}$$

where q_t, d_t and s_t are respectively the state, the unconsumed input and the content of the stack of the automaton in reversed order at time t.

A VS simulating a PDA in real-time can be constructed from the PDA's transition function by defining the Domain of Dependence to be DoD $= (-3, 1) = \{-2, -1, 0\}$, F to always map to 0, and G so that, given $\delta : (q_t, \kappa, s_{0_t}) \mapsto (q_{t+1}, \chi)$,

$$G : \begin{cases} s_{0_t}\ q_t\ .\ \kappa & \mapsto & \epsilon\ q_{t+1}\ .\ \epsilon & \text{if } \chi = \epsilon \\ s_{0_t}\ q_t\ .\ \kappa & \mapsto & s_{0_t}\ \chi\ q_{t+1}\ .\ \epsilon & \text{otherwise.} \end{cases} \tag{9}$$

PDA recognize the class of languages generated by context-free grammars (CFG). PDA and CGFs are thus equivalent in power. A CFG specifies a language, i.e. a set of strings on some alphabet, by defining how its words can be constructed, moving from a distinguished starting symbol and applying substitution rules until a string of unsubstitutable symbols (terminals) is reached.

A CFG can be formally defined as a 4-tuple $G_{CF} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$, where $\mathbf{N}$ is a set of non-terminal symbols, $\mathbf{T}$ is a set of terminal symbols, $R \subset \mathbf{N} \times (\mathbf{N} \cup \mathbf{T})^*$ a set of substitution rules and $\mathtt{S}$ a distinguished start symbol. In particular, each rule in R can be written as $X \to w$, with $X \in \mathbf{N}$ and $w \in (\mathbf{N} \cup \mathbf{T})^*$.

For example, let us define a CFG G_{ex} with $\mathbf{N} = \{\mathtt{S}\}$, $\mathbf{T} = \{\mathtt{(}, \mathtt{[}, \mathtt{)}, \mathtt{]}\}$, and R containing the rules

$$\begin{aligned} \mathtt{S} &\to \mathtt{(S)} \\ \mathtt{S} &\to \mathtt{[S]} \\ \mathtt{S} &\to \epsilon. \end{aligned}$$

Then G_{ex} generates the language $\mathcal{L}_{\text{ex}}$ of balanced round and square brackets. By applying the substitution rules we can in fact derive any string in that language. For illustration purposes, an example derivation would be: $\mathtt{S} \to \mathtt{[S]} \to \mathtt{[(S)]} \to \mathtt{[()]} \in \mathcal{L}_{\text{ex}}$. It is always possible to construct, given any CFG, a PDA recognizing its language, and viceversa.

Top-down recognizers In one of the examples presented later in the text, we will make use of top-down recognizers (TDRs, see Aho and Ullman, 1972) that can process locally unambiguous non-left-recursive CFGs[3]. TDRs are

[3]A recursive CFG is a CFG including rules $A \to uAv$ that expand a non-terminal symbol A into a string containing the same non-terminal. A CFG is called left-recursive

a subclass of PDA that can simulate rule expansion to accept languages generated by non-left-recursive CFGs. Given any CFG G_{CF} that is not left-recursive, it is possible to construct a TDR that can parse strings belonging to the context-free language generated by that grammar. If the input string of a TDR constructed from G_{CF} is in the language generated by that grammar (and thus it can be derived by the grammar), then the TDR will end its computation with an empty stack and input, and is said to accept the string by empty stack. We are specifically interested in TDRs that process locally unambiguous CFGs, which have the additional property of needing only one state to perform their computation. To construct such a TDR from a locally unambiguous non-left-recursive CFG $G_{CF} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$ it is sufficient to define its δ function in the following way:

$$\delta : \begin{cases} (q_0,\, a,\, a) & \mapsto & (q_0,\, \epsilon) & \text{for all } a \in \mathbf{T} \\ (q_0,\, \epsilon,\, X) & \mapsto & (q_0,\, w) & \text{for all } (X \to w) \in R \end{cases} \tag{10}$$

where $X \in \mathbf{N}$ is a non-terminal, $w \in (\mathbf{N} \cup \mathbf{T})^*$ is a string of terminals and non-terminals, and q_0 is the TDR's only state. Note that in the definition above we endow TDRs with the additional capability of pushing strings w on the stack rather than single symbols.

As our TDRs only have one state q_0, we can describe their machine configuration without referring to the current state. It is thus possible to encode TDR configurations on dotted sequences as follows:

$$\underbrace{s_{m_t} \ \ldots \ s_{0_t}}_{s_t} \cdot \underbrace{d_{0_t} \ \ldots \ d_{n_t}}_{d_t} \tag{11}$$

where d_t and s_t are respectively the unconsumed input and the content of the stack of the automaton in reverse order at time t. Similarly, simpler VSs than those needed to simulate PDAs can be constructed from a TDR's transition function, by defining the Domain of Dependence to be $\text{DoD} = (-2, 1) = \{-1, 0\}$, F to always map to 0 and G to mirror Equation 10 so that

$$G : \begin{cases} a\,.a & \mapsto & \epsilon\,.\epsilon \\ X.a & \mapsto & w.\epsilon \end{cases} \tag{12}$$

for all $a \in \mathbf{T}$, $(X \to w) \in \mathbf{R}$.

if such rules appear in the form $A \to Aw$. A CFG is locally unambiguous if there are no two rules expanding the same nonterminal.

Turing machines A Turing machine (TM) is an automaton with read-write random access to a two-sided infinite tape [Sipser, 2006, Turing, 1937]. TMs are central to the Theory of Computation, and they are thought to be powerful enough to model any physically realizable computation (with assumptions of unbounded resources). A TM has an in-built tape (doubly-infinite one dimensional memory with one symbol capacity at each memory location) and a finite-state controller endowed with a read-write head that follows the instructions encoded by the transition function. At each step of the computation, given the current state and the current symbol read by the read-write head, the controller determines via a δ transition function the writing of a symbol on the current memory location, a shift of the read-write head to the memory location to the left ($\mathcal{L}$) or to the right ($\mathcal{R}$) of the current one, and the transition to a new state for the next computation step. Formally, a TM [Turing, 1937] can be defined as a 7-tuple $M_{\text{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is a finite set of tape symbols also containing the blank symbol $\sqcup$, $\mathbf{T} \subset \mathbf{N} \setminus \{\sqcup\}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subset Q$ is a set of 'halting' states reached at the end of the computation and $\delta : Q \times \mathbf{T} \to Q \times \mathbf{T} \times \{\mathcal{L}, \mathcal{R}\}$ is a partial transition function, the so-called machine table, that determines the dynamics of the machine. In particular, δ is defined as follows:

$$\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, d_{0_{t+1}}, m) \tag{13}$$

where $q_t, q_{t+1} \in Q$ are the state of the machine before and after the transition, $d_{0_t}, d_{0_{t+1}} \in \mathbf{N}$ are respectively the read and rewritten symbol, and $m \in \{\mathcal{L}, \mathcal{R}\}$ denotes the shift of the read-write head to the left or to the right.

At a given computation step, the content of the tape together with the position of the read-write head and the current controller state define a machine configuration. It is possible to encode TM configurations on dotted sequences as follows:

$$s = \underbrace{\ldots\ d_{-2_t}\ d_{-1_t}}_{l_t}\ q_t\ .\ \underbrace{d_{0_t}\ d_{1_t}\ d_{2_t}\ \ldots}_{r_t}, \tag{14}$$

where l_t describes the part of the tape to the left of the read-write head, r_t describes the part to its right, q_t describes the current state of the machine controller, and the central dot denotes the current position of the read-write head, i.e. d_{0_t}, the symbol to its right.

A VS simulating a TM in real-time can be constructed from the TM's transition function by defining the Domain of Dependence to be $\text{DoD} = (-3, 1) = \{-2, -1, 0\}$, and G and F so that, given $\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, \hat{d}_{0_t}, m)$,

$$\begin{aligned} G: & \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & d_{-1_t}\ \hat{d}_{0_t}\ .\ q_{t+1} & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & q_{t+1}\ d_{-1_t}\ .\ \hat{d}_{0_t} & \text{if } m = \mathcal{L} \end{cases} \\ F: & \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & -1 & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & +1 & \text{if } m = \mathcal{L} \end{cases} \end{aligned} \tag{15}$$

for all $d_{-1_t} \in \mathbf{N}$.

The following example will clarify how the VS defined as above (Equation 15) can simulate a TM. Consider, for instance, the dotted sequence "$\mathtt{w}q_0\mathtt{.ord}$", and define a TM such that $\delta : (q_0, \mathtt{o}) \mapsto (q_1, \mathtt{a}, \mathcal{R})$ and $\delta : (q_1, \mathtt{r}) \mapsto (q_1, \mathtt{n}, \mathcal{L})$. Then a computation step of the TM starting from the "$\mathtt{w}q_0\mathtt{.ord}$" configuration would yield a new configuration "$\mathtt{wa}q_1\mathtt{.rd}$"; by running the TM again, this time starting from "$\mathtt{wa}q_1\mathtt{.rd}$", a computation step would yield "$\mathtt{w}q_1\mathtt{.and}$", as prescribed by the transition function we defined. Constructing a VS Ω_{ex} as specified by Equation 15 and applying it to "$\mathtt{w}q_0\mathtt{.ord}$":

$$\begin{aligned} \Omega_{\text{ex}}(\mathtt{w}q_0\mathtt{.ord}) &= \sigma^{F(\mathtt{w}q_0\mathtt{.ord})}\big(\mathtt{w}q_0\mathtt{.ord} \oplus G(\mathtt{w}q_0\mathtt{.ord})\big) \\ &= \sigma^{-1}\big(\mathtt{w}q_0\mathtt{.ord} \oplus \mathtt{wa.}q_1\big) \\ &= \sigma^{-1}\big(\mathtt{wa.}q_1\mathtt{rd}\big) \\ &= \mathtt{wa}q_1\mathtt{.rd} \end{aligned} \tag{16}$$

and by applying it again to the resulting "$\mathtt{wa}q_1\mathtt{.rd}$" dotted sequence we obtain

$$\begin{aligned} \Omega_{\text{ex}}(\mathtt{wa}q_1\mathtt{.rd}) &= \sigma^{F(\mathtt{wa}q_1\mathtt{.rd})}\big(\mathtt{wa}q_1\mathtt{.rd} \oplus G\big(\mathtt{wa}q_1\mathtt{.rd}\big) \\ &= \sigma^{+1}\big(\mathtt{wa}q_1\mathtt{.rd} \oplus q_0\mathtt{a.n}\big) \\ &= \sigma^{+1}\big(\mathtt{w}q_0\mathtt{a.nd}\big) \\ &= \mathtt{w}q_0\mathtt{.and} \end{aligned} \tag{17}$$

where the DoD of the input string to the VS has been highlighted for clarity. Note that the dotted representation of the machine configuration requires index -1 to always contain the machine state. For this reason, it is not enough to only rewrite the symbols in $\{-1, 0\}$ (i.e. the machine state and the current symbol under the read-write head) to simulate a TM, as intuition would instead suggest. In fact, a VS first applies a rewriting of its DoD, and

then shifts the resulting dotted sequence to the left (when $F(s) = -1$) or the right (when $F(s) = +1$). In particular, the shift is needed to simulate the movement of the read-write head on the machine tape. In order to make sure that at the end of the substitution and shift the machine state is correctly placed at its reserved index -1, the substitution must leave it displaced one place to the right if a left shift is to be applied (as in Equation 16), or one to the left in case of a right shift (as in Equation 17). This last case requires the additional dependence of the VS on index -2. Furthermore, note that our construction is equivalent to that from Moore [1990, 1991]: the VS defined in Equation 15 is nothing more than the GS introduced by Moore to prove the equivalence between GSs and TMs.

2.2 Introducing Nonlinear Dynamical Automata

We will now discuss how VSs, and thus the models of symbolic computation they can simulate, can be mapped to piecewise affine-linear systems on a vectorial space, obtaining nonlinear dynamical automata.

2.2.1 Gödel Encodings and the Symbol Plane

A Gödel encoding (or Gödelization, see Gödel, 1931) allows one to uniquely assign a real number to a sequence such that the space of one-sided infinite sequences can be mapped to the real interval $[0, 1]$.[4] For completeness, Gödelization is subsequently discussed alongside its graphical representation, provided in Figure 3.

Let $\mathbf{A}^{\mathbb{N}}$ be the space of one-sided infinite sequences over an alphabet $\mathbf{A}$ containing $|\mathbf{A}| = g$ symbols, and $s = d_1 d_2 \ldots$ a sequence in this space, with d_k being the k-th symbol in s. Additionally, let $\gamma : \mathbf{A} \to \mathbb{N}$ be a one-to-one function associating each symbol in the alphabet $\mathbf{A}$ with a natural number. Then a Gödelization is a mapping from $\mathbf{A}^{\mathbb{N}}$ to $[0, 1] \subset \mathbb{R}$ defined as follows:

$$\psi(s) := \sum_{k=1}^{\infty} \gamma(d_k) g^{-k}. \tag{18}$$

[4] A Gödel encoding maps sequences on some alphabet $\mathbf{A}$ to real numbers through the use of a base-b expansion, with $b = |\mathbf{A}|$. It can be proven that any base-b expansion represents a real number, and that any real number has a unique base-b representation under a weak condition. The uniqueness of the Gödel encoding (and decoding) of any sequence follows from the same proof.

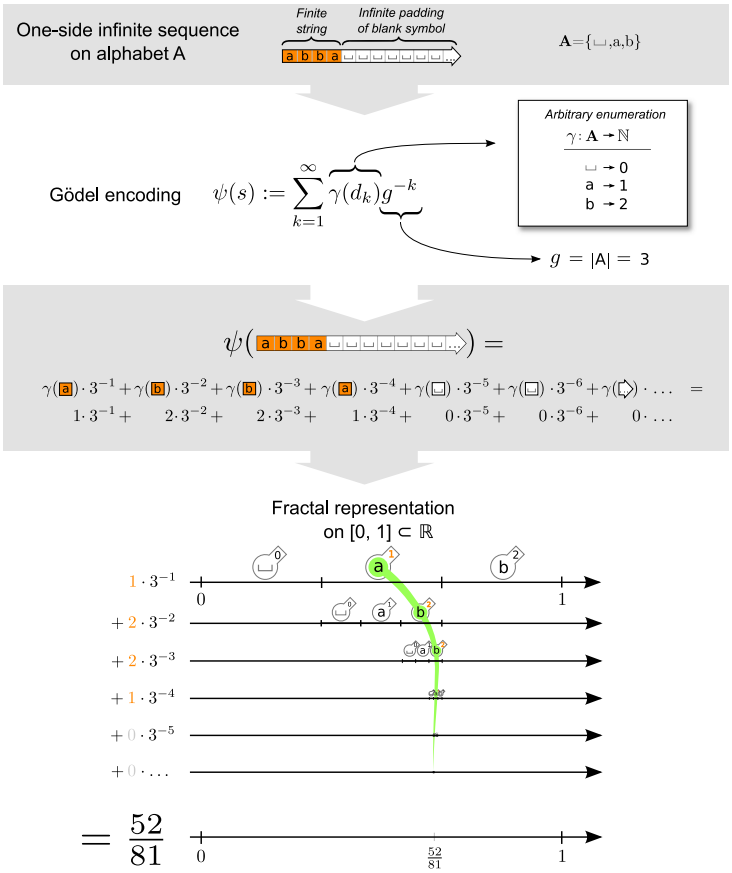

Figure 3: **Three representations of the Gödel Encoding of a sequence.** The first one is just the definition of the Gödel encoding, with details on the specific choice of the enumerating γ function and the induction of the g constant, given the alphabet $\mathbf{A}$ from which the sequence takes its symbols. The second one is an expansion of the series in the definition. The third one visually conveys the fractal and convergent nature of the series, highlighting the relation between numbers and symbols by the use of the color orange. At each level of this representation, from top to bottom, the encoding of the sequence "`abba` ␣ ␣ ␣ ..." is sequentially constructed, highlighting the contribution of each encoded symbol to the real number resulting from the complete Gödelization.

Conveniently, Gödelization can also be employed on a dotted sequence $\alpha.\beta \in \mathbf{A}^{\mathbb{Z}}$ — herein representing a machine configuration — by splitting it into its two one-sided constituents α' (the reversed α) and β. Defining two Gödel encodings ψ_x and ψ_y for α' and β respectively, induces a two-dimensional representation for $\alpha.\beta$, i.e. $\big(\psi_x(\alpha'), \psi_y(\beta)\big)$, known as symbol plane or symbologram, which is contained in the unit square $[0,1]^2 \subset \mathbb{R}^2$. In encoding dotted sequences $\alpha.\beta$ representing configurations of the machines we consider in this paper, α often only ever contains states as first symbols, and tape symbols in the rest of the sequence. In this case we can define a more refined Gödelization that covers all of the representational space $[0,1] \in \mathbb{R}$:

$$\psi_x(s) := \gamma_q(d_1)n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(d_{k+1})n_s^{-k}n_q^{-1}, \tag{19}$$

where γ_q and γ_s respectively enumerate the set of states Q and the tape alphabet $\mathbf{A}$, and where $n_q = |Q|$, $n_s = |\mathbf{A}|$.

2.2.2 Versatile Shifts as Affine-Linear Transformations

Push and pop operators can be defined on one-sided infinite sequences $\mathbf{A}^{\mathbb{N}}$ on some alphabet $\mathbf{A}$. The push operator $\odot$ is defined so that $s \odot b$ adds the contents of a word $b \in \mathbf{A}^*$ to the beginning of $s \in \mathbf{A}^{\mathbb{N}}$, whereas the pop operator $\ominus$ is defined so that $\ominus^p s$ removes the first p symbols in s. We will now show that Gödelizing a sequence resulting from the application of pop and push operations is equivalent to applying an affine-linear transformation on the original Gödelized sequence. We will then show that VSs on a dotted sequence $\alpha.\beta$ can be mapped to push and pop operations on its one-sided constituents α' and β. Let $s = d_1d_2d_3\ldots$ be a one-sided infinite sequence on an alphabet A. Applying a pop operation $\ominus^p$ to s yields $\ominus^p s = d_{p+1}d_{p+2}d_{p+3}\ldots$, while pushing a word $b = b_1\ldots b_r$ to the beginning of s yields $s \odot b = b_1\ldots b_r d_1 d_2\ldots$. In this case

$$\psi\left(s\right) = \gamma(d_1)g^{-1} + \gamma(d_2)g^{-2} + \gamma(d_3)g^{-3} + \ldots,$$

so that

$$\begin{aligned}\psi(\ominus^p s) &= \gamma(d_{p+1})g^{-1} + \gamma(d_{p+2})g^{-2} + \gamma(d_{p+3})g^{-3} + \ldots \\ &= \psi(s)\cdot g^p - \sum_{i=1}^{p}\gamma(d_i)g^{p-i},\end{aligned}$$

and

$$\begin{aligned}\psi(s \odot b) &= \gamma(b_1)g^{-1} + \ldots + \gamma(b_r)g^{-r} + \\ &\quad \gamma(d_1)g^{-(r+1)} + \gamma(d_2)g^{-(r+2)} + \ldots \\ &= \psi(s) \cdot g^{-r} + \sum_{i=1}^{r} \gamma(b_i)g^{-i},\end{aligned}$$

proving that the resulting Gödelized sequences can be obtained by applying affine-linear transformations to the original Gödelized sequences. For both pop and push operations, the parameters of the affine-linear transformations only depend on the number and identities on the symbols that are respectively removed from or added to the beginning of the original sequence. This is of particular importance in the framework of interactive computation [Wegner, 1998], where the newly added symbol stems from the network's interaction with its environment. Accordingly, the symbol b becomes represented by a linear operator acting on the system's state space, analogous to quantum operators acting on Hilbert spaces [beim Graben et al., 2008].

As previously discussed, a VS defines two operations on dotted sequences, a substitution operation $s \oplus G(s)$ which replaces the dotted sub-sequence in the DoD of the shift with a new dotted sequence $G(s)$, and a shift operation $\sigma^{F(s)}$ shifting the symbols in s to the left or to the right by $F(s)$ positions. Let $s \oplus G(s) = w_\alpha u.v w_\beta \oplus \hat{u}.\hat{v}$ be a substitution replacing the dotted sub-sequence $u.v$ in s with the dotted word $\hat{u}.\hat{v}$, then $s \oplus G(s)$ can be straightforwardly mapped to pop and push operations on $u'{w_\alpha}'$ and $v w_\beta$, the one-sided constituents of the original dotted sequence s, as follows:

$$\begin{aligned}w_\alpha u.v w_\beta \oplus \hat{u}.\hat{v} &= \left((\ominus^{|u'|} u' {w_\alpha}') \odot \hat{u}'\right)'.\left((\ominus^{|v|} v w_\beta) \odot \hat{v}\right) \\ &= \left({w_\alpha}' \odot \hat{u}'\right)' . \left(w_\beta \odot \hat{v}\right) \\ &= w_\alpha \hat{u}.\hat{v} w_\beta\end{aligned}$$

showing that substitutions on dotted sequences can be mapped to pop and push operations on its one-sided constituents. A left shift σ^{-1} and a right shift σ^1 on a dotted sequence $\alpha.\beta = \ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots$ can be mapped to push and pop operations on its one-sided constituents as follows:

$$\begin{aligned}\sigma^{-1}(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (\alpha' \odot d_0)'.(\ominus^1 \beta) \\ &= \ldots\ d_{-1}\ d_0\ .\ d_1\ d_2\ \ldots\ ,\end{aligned}$$

and

$$\begin{aligned}\sigma^1(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (\ominus^1\alpha')'.(\beta \odot d_{-1}) \\ &= \ldots d_{-3}\ d_{-2}\ .\ d_{-1}\ d_0\ \ldots\ ,\end{aligned}$$

showing that shifts on dotted sequences can be mapped to pop and push operations on its one-sided constituents. Any arbitrary shift σ^k with $k \in \mathbb{Z}$ can be obtained by composition of left and right shifts; as the composition of affine-linear transformations is an affine-linear transformation, the Gödelization of a sequence resulting from the composition of shift operations is equivalent to an affine-linear transformation on the original Gödelized sequence. We have thus shown that VSs on dotted sequences can be mapped to pop and push operations on one-sided infinite sequences, and that the Gödelization of these operations can be mapped to affine-linear transformations on the original sequences. On the symbologram, each substitution and shift operation on a Gödelized dotted sequence $\alpha.\beta$ by a VS involves two affine-linear transformations, one acting on the Gödelized α' (the reversed α) and one on the Gödelized β. The parameters of the affine-linear transformations only depend on the symbols of the dotted sequence in the DoD of the VS. All dotted sequences which share the same DoD symbols are thus associated to the same pair of affine-linear transformations. For this reason, the symbologram representation of VSs leads to piecewise affine-linear maps on rectangular partitions of the unit square, referred to as a nonlinear dynamical automata [beim Graben et al., 2008, 2004, Tabor, 2000, Tabor et al., 2013].

2.2.3 Nonlinear Dynamical Automata

A nonlinear dynamical automaton (NDA) is a triple $M_{NDA} = (X, P, \Phi)$, where P is a rectangular partition of the unit square $X = [0,1]^2 \subset \mathbb{R}^2$, that is

$$P = \{D^{i,j} \subset X |\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ m, n \in \mathbb{N}\}, \tag{20}$$

so that each cell is defined as $D^{i,j} = I_i \times J_j$, with $I_i, J_j \subset [0,1]$ being real intervals for each bi-index (i,j), with $D^{i,j} \cap D^{k,l} = \varnothing$ if $(i,j) \neq (k,l)$, and $\bigcup_{i,j} D^{i,j} = X$. The couple (X, Φ) is a time-discrete dynamical system with phase space X and the flow $\Phi : X \to X$ is a piecewise affine-linear map such that $\Phi_{|D^{i,j}} := \Phi^{i,j}$, with $\Phi^{i,j}$ having the following form:

$$\Phi^{i,j}(\mathbf{x}) = \begin{pmatrix} a_x^{i,j} \\ a_y^{i,j} \end{pmatrix} + \begin{pmatrix} \lambda_x^{i,j} & 0 \\ 0 & \lambda_y^{i,j} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{21}$$

Note that the NDA, as any piecewise affine-linear system, also requires a switching rule $\Theta(x, y) \in \{(i, j)|\ 1 \leq i \leq m, 1 \leq j \leq n\}$, which selects the appropriate branch, and thus dynamics (i.e. $\Phi(x, y) = \Phi^{i,j}(x, y) \iff \Theta(x, y) = (i, j)$). A mapping between a VS and a NDA can be defined following the methods outlined in Section 2.2.1 and Section 2.2.2, therefore enabling the derivation of the parameters of the NDA. That is, first each cell $D^{i,j} = I_i \times J_j$ can be seen as containing all the Gödelized dotted sequences $\alpha.\beta$ which agree (i.e. have the same symbols) in the Domain of Dependence. In particular, the I_i interval contains all the DoD-agreeing Gödelized α' (the reversed α) sub-sequences, whereas the J_j interval contains all the DoD-agreeing Gödelized β sub-sequences. This leads to a partition of the unit square with a number i of I intervals equal to the number of possible one-sided sub-sequences that can appear in the left DoD of the VS, and a number j of J intervals equal to the number of possible one-sided sub-sequences that can appear in the right DoD. For example, for a VS simulating a FSM, the left Domain of Dependence $\text{DoD}_\alpha = \{-1\}$ of the dotted sequences representing machine configurations only ever contains states, and the right Domain of Dependence $\text{DoD}_\beta = \{0\}$ only ever contains input symbols. In this case the number of I_i intervals becomes equal to the number of states $n_q = |Q|$ in the FSM, and the number of J_j intervals equal to the number of input symbols $n_s = |\mathbf{T}|$, where Q and $\mathbf{T}$ are respectively the set of states and that of input symbols in the FSM. For a VS simulating a TM, instead, the left Domain of Dependence $\text{DoD}_\alpha = \{-2, -1\}$ only ever contains states at index -1, and tape symbols at index -2, and the right Domain of Dependence $\text{DoD}_\beta = \{0\}$ always contains tape symbols. This leads to a partition of the unit square with a number of I_i intervals equal to $m = n_q n_s$, and one of J_j intervals equal to $n = n_s$, leading to a total of $n_q n_s^2$ cells, where n_s is the number of symbols in the tape alphabet $\mathbf{N}$ and n_q is the number of states in Q.

Following Section 2.2.2, substitutions and shifts on a sequence can be mapped to affine-linear transformations on its Gödelization. For this reason, each cell in the partition P of the unit square is associated with a different affine-linear transformation with parameters $(a_x^{i,j}, a_y^{i,j})$ and $(\lambda_x^{i,j}, \lambda_y^{i,j})$, which can be derived using the methods outlined in Section 2.2.2. Therefore a model of computation can be represented as a NDA by means of its Gödelized VS representation.

2.3 Solution Map between NDA and R-ANNs

The design of the map between the NDA and a first order R-ANN follows a conceptually natural and simple solution, which attempts to mimic the affine-linear dynamics (given by Equation 21) of the NDA on the partitioned unit square (see Carmantini et al., 2015 for preliminary work in this direction).

Let $\rho(\cdot)$ denote the proposed map. The objective is to map the orbits of the NDA (i.e. $\Phi^{i,j}(x,y)$) to orbits of the R-ANN, denoted as $\zeta^{i,j}(x,y)$. The role of ρ is to encode both the affine-linear dynamics within each partition cell ($D^{i,j}$) and to emulate the transitions from cell to cell by suitably activating certain neural units within the R-ANN. To achieve this, we propose a network architecture with three layers, namely a machine configuration layer (MCL), a branch selection layer (BSL) and a linear transformation layer (LTL), as depicted in Figure 4. Therefore, we generically define the proposed map as follows:

$$\zeta = \rho(\mathcal{I}, \mathcal{A}, \Phi, \Theta), \tag{22}$$

where $\mathcal{I}_{2\times 2}$ is the identity matrix that maps (identically) the initial conditions of the NDA to the R-ANN and $\mathcal{A}$ is the synaptic weight matrix that defines the network architecture, which will be discussed in subsequent Sections. In addition, ρ generates different neural dynamics for each type of the neural units, i.e. $\zeta = (\zeta_1, \zeta_2, \zeta_3)$, corresponding to MCL, BSL and LTL, respectively. The details of the R-ANN architecture and its dynamics will now be presented.

2.3.1 Network Architecture and Neural Dynamics

The simulation of a NDA orbit within the R-ANNs is distributed among MCL, BSL and LTL. Since $\Phi^{i,j}(x)$ is a two-dimensional de-coupled discrete map it suggests only two neural units in a read-out layer, which is a role taken by the MCL. We refer to the two MCL units as c_x and c_y. At each computation step the MCL stores the encoding of the current machine configuration, which is then passed on to the BSL and LTL units. Subsequently, two sets of BSL units (b_x and b_y) functionally act as a switching system that determines to which cell $D^{i,j}$ the current machine configuration belongs, triggering the appropriate units within two sets of LTL units (t_x and t_y), effectively emulating the application of an affine-linear transformation $\Phi^{i,j}$ on an encoded machine configuration. This action corresponds to the application of a symbolic operation by the original machine, leading to a

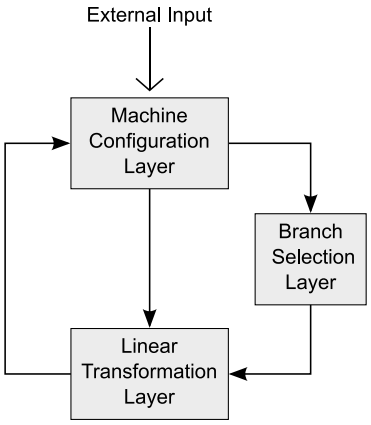

Figure 4: **Connectivity between neural layers within the network.** The machine configuration layer (MCL) receives external input (in this case the encoded initial machine configuration), and synaptically couples to the branch selection layer (BSL) and linear transformation layer (LTL). The BSL feed-forwards to the LTL and finally the LTL recurrently feedbacks to the MCL, where the output is read-out.

configuration update. The result of the transformation is then fed back to the MCL, representing the configuration (i.e. the machine's symbolic data) for the next computation step. These successive transformations effectively emulate the action of a NDA, where for every computational step an affine-linear transformation is applied to the values encoding the representation of the machine configuration. The neural units in the various layers make use of either the Heaviside (H) or the Ramp (R) activation functions defined as follows (see also Figure 5):

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{23}$$

2.3.2 Machine Configuration Layer

The MCL encodes the state of the simulated NDA, and thus the data of the simulated automaton, while acting as a read-out neural layer. At the same time it mediates at each computation step the transmission of the current

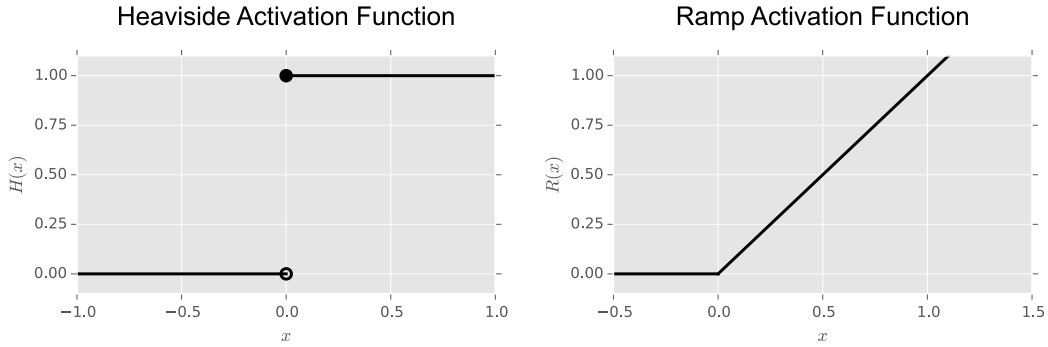

Figure 5: **Activation functions employed in the network.** In particular, the Heaviside function $H(x)$ is employed by units in the branch selection layer and the Ramp function $R(x)$ is used in both machine configuration layer and linear transformation layer.

Gödel encoding of the emulated machine's configuration to the BSL and LTL units. Since the Gödel encoding of a dotted sequence representing a machine configuration consists of two values (see Section 2.2.1), this implies that the MCL solely requires two neural units (c_x and c_y) to code for the current configuration. As a consequence, the initialization of the R-ANNs is performed in this layer, where the initial conditions $(\psi_x(\alpha'), \psi_x(\beta))$ are identically transformed (via $\mathcal{I}$) by the map $\rho(\cdot)$ as follows:

$$(c_x, c_y) = (\psi_x(\alpha'), \psi_x(\beta)) \equiv \zeta_1 = \rho(\mathcal{I}, \cdot, \cdot, \cdot)_{|(\psi_x(\alpha'), \psi_x(\beta))} \tag{24}$$

Following every computation step, these neural units receive inputs from the LTL units and are subsequently activated via the ramp activation function (Equation 23); in other words $\zeta_1 \equiv (c_x, c_y) = (R(\sum_i t_x^i), R(\sum_j t_y^j))$. Finally, these synaptically project onto the BSL and LTL neural units (refer to Figure 6 for details of the connectivity).

2.3.3 Branch Selection Layer

The BSL acts as a control unit that enables the sequential mapping of the orbits of the NDA, $\Phi^{i,j}(x, y)$, to orbits of the R-ANNs, $\zeta^{i,j}(x, y)$. Specifically, the BSL functionally embodies the switching rule $\Theta(x, y)$ and coordinates the dynamic switching between LTL units. Sequentially, under the action of BSL units, only a single pair of LTL units $(t_x^{i,j}, t_y^{i,j})$ dedicated to emulate $\Phi^{i,j}$ become active, which then operate on an encoded Machine configuration.

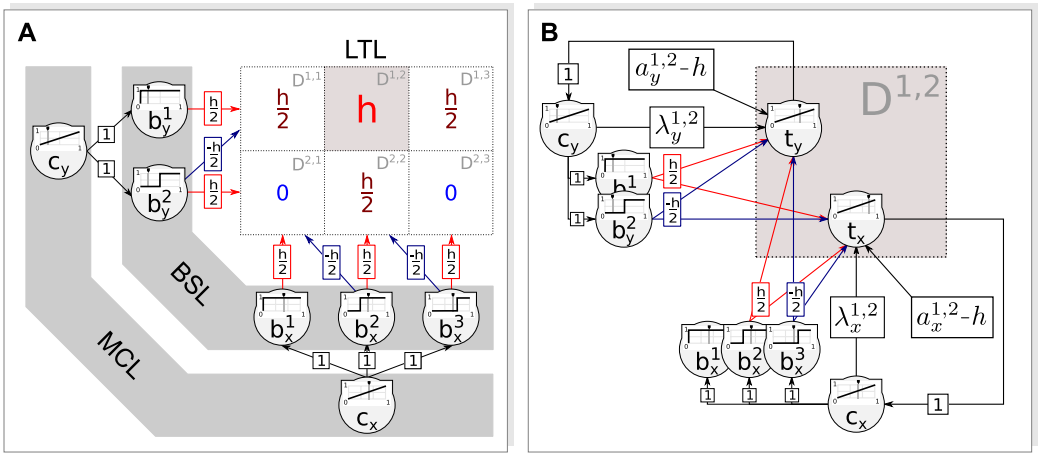

Figure 6: **Detailed feedforward connectivity and weights for neural network simulating a nonlinear dynamical automaton with only 6 branches.** (A) The machine configuration layer (MCL) units (c_x, c_y) feedforward connect to all the branch selection layer (BSL) units with weight of 1. Every BSL unit excites with weights $\frac{h}{2}$ (in red) and also inhibits with weights $\frac{h}{2}$ (in blue) the relevant linear transformation layer (LTL) units contained within each cell (as indicated by the red and blue arrows respectively). Each cell D^{ij} indicates the overall summed input value received by each LTL unit (for visualization purpose/convenience not shown) from the BSL. In this case only the LTL units in cell $D^{1,2}$ are activated with overall BSL input value of h (red). (B) A zoom-in of panel (A), shows in detail how each pair of LTL units contained within each cell (in this case $D^{1,2}$) receives inputs from the MCL and BSL units as shown. In addition, the LTL units may have internal dynamics described by parameter a (equivalently, this can be seen as input from an always-active unit). To actually produce output, the overall input to an LTL unit must overcome its internal h inhibition. Upon activation, the LTL unit's output is fed back to the paired MCL unit with weight of 1.

In particular, the BSL units make sure that $(t_x^{i,j}, t_y^{i,j})$ become active only if $(c_x, c_y) \in D^{i,j} = I_i \times J_j$, with $I_i = [\xi_i, \xi_{i+1})$ being the i-th interval on the x-axis and $J_j = [\eta_j, \eta_{j+1})$ being the j-th interval on the y-axis. The switching rule is mapped by $\rho(\cdot)$ as follows:

$$\zeta_2(x, y) = \rho(\cdot, \cdot, \cdot, \Theta(x, y) = \{i, j\}) \tag{25}$$

The implementation of $\zeta_2(x, y)$ is mediated by two sets of neural units, i) the b_x set with m units (the number of I intervals on the x-axis) and ii) the b_y set with n units (the number of J intervals on the y axis), which are activated via a Heaviside activation function (Equation 23) after receiving excitatory inputs with synaptic weight 1 from the MCL layer (i.e. c_x and c_y units) in the following way:

$$\begin{aligned} b_x^i &= H(c_x - \xi^i) \qquad \text{with} \qquad \xi^i = \min(I_i), \\ b_y^j &= H(c_y - \eta^j) \qquad \text{with} \qquad \eta^j = \min(J_j). \end{aligned} \tag{26}$$

That is, the activation of the BSL units depends on a threshold, implemented here as a synaptic projection from an always-active bias unit, that is defined as the minimum of the intervals I_i and J_j respectively for the b_x^i and b_y^i units. This has the effect of centering the threshold towards the left boundary of each interval (i.e. a bias of $-\xi^i$ for b_x^i unit and $-\eta^j$ for b_y^j). Therefore, if the read-out (i.e. encoded machine configuration) of the c_x and c_y units in the MCL corresponded to a point on the unit square belonging to cell $D^{i,j}$, then the b_x^i unit would be triggered active as well as all units b_x^k with $k < i$. The same would occur for neurons b_y^j and all neural units b_y^k with $k < j$.[5] Upon excitation, these BSL units then synaptically project to the relevant LTL units, $(t_x^{i,j}, t_y^{i,j})$ that are naturally inactive due to a strong inhibitory bias with magnitude h (the role and value of h will be clarified in the subsequent Section). Specifically, each neural unit b_x^i establishes synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units within the cells $D^{k,i}$ (i.e. $(t_x^{k,i}, t_y^{k,i})$) and also project with synaptic inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units within the cells $D^{k,i-1}$ (i.e. $(t_x^{k,i-1}, t_y^{k,i-1})$), where $k = 1, \dots m$; for a graphical depiction see Figure 6. Similarly, each neural unit b_y^j projects with synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units within the cells $D^{j,k}$ (i.e. $(t_x^{j,k}, t_y^{j,k})$) and also projects with

[5]Note that the action of the BSL could be equivalently implemented by interval indicator functions represented as linear combinations of Heaviside functions.

synaptic inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units within the cells $D^{j-1,k}$ (i.e. $(t_x^{j-1,k}, t_y^{j-1,k})$), where $k = 1, \ldots n$; see Figure 6. The combined effect of the b_x^i units and b_y^j is therefore to counterbalance through their synaptic weights the natural inhibition (of bias h) of the LTL units in cell $D^{i,j}$. In other words each couple of LTL units $(t_x^{i,j}, t_y^{i,j})$ receives an input $B_x^i + B_y^j$, defined as follows:

$$\begin{aligned} B_x^i &= b_x^i \frac{h}{2} + b_x^{i+1} \frac{-h}{2} \\ B_y^j &= b_y^j \frac{h}{2} + b_y^{j+1} \frac{-h}{2}, \end{aligned} \tag{27}$$

where the input sum

$$B_x^i + B_y^j = \begin{cases} h & \text{if } (c_x, c_y) \in D_{i,j} \\ \frac{h}{2} & \text{if } c_x \in I_i, c_y \notin J_j \quad \text{or} \quad c_x \notin I_i, c_y \in J_j \\ 0 & \text{if } (c_x, c_y) \notin D_{i,j} \end{cases} \tag{28}$$

only triggers the relevant LTL unit if it reaches the value h. That is, if the pair $(t_x^{i,i}, t_y^{i,j})$, is selected by the BSL units (and thus $(c_x, c_y) \in D_{i,j}$), then $B_x^i + B_y^j = h$. Otherwise $B_x^i + B_y^j$ is either equal to $\frac{h}{2}$ or 0. An example of this mechanism is shown in Figure 6, where the LTL units in cell $D^{1,2}$ are activated via mediation of $b_x = \{b_x^1, b_x^2, b_x^3\}$ and $b_y = \{b_y^1, b_y^2\}$. Here, both b_x^3 and b_y^2 are not excited since respectively c_x and c_y are not activated enough to drive them towards their threshold. However, b_x^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,2}$ and $D^{1,2}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cell $D^{2,1}$ and $D^{1,1}$. Equally, b_y^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,1}$, $D^{2,2}$ and $D^{2,3}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cells $D^{1,1}$, $D^{1,2}$ and $D^{1,3}$. The b_x^1 and b_y^1 units excite respectively cells $\{D^{2,1}, D^{1,1}\}$ and $\{D^{1,1}, D^{1,2}, D^{1,3}\}$, but these do not inhibit any cells (due to boundary conditions).

2.3.4 Linear Transformation Layer

The LTL embodies the set of affine-linear transformations of the NDA from which the network is constructed, and thus the set of symbolic operations defined by the transition table of the simulated automaton. This endows the LTL with the functional ability of generating an updated encoded machine configuration from the current one. That is, the affine-linear transformation

of a NDA, $\Phi^{i,j}(x,y) = (\lambda_x^{i,j}x + a_x^{i,j}, \lambda_y^{i,j}y + a_y^{i,j})$ within a cell $D^{i,j}$ is simulated by the LTL unit $(t_x^{i,j}, t_y^{i,j})$. This induces the following mapping:

$$(t_x^{i,j}, t_y^{i,j}) \quad = \quad \zeta_3^{i,j}(x,y) = \rho(\cdot,\cdot,\Phi^{i,j}(x,y),\cdot). \tag{29}$$

This affine-linear transformation is implemented in the form of synaptic computation, which is only triggered when the BSL units provide enough excitation enabling the two neural units $(t_x^{i,j}, t_y^{i,j})$ to cross their threshold value and execute the operation. The read-out of this process is as follows:

$$\begin{array}{rcl} t_x^{i,j} & = & R(\lambda_x^{i,j}c_x + a_x^{i,j} - h + B_x^i + B_y^j) \\ t_y^{i,j} & = & R(\lambda_y^{i,j}c_y + a_y^{i,j} - h + B_x^i + B_y^j), \end{array} \tag{30}$$

that is, initially the LTL units are rendered inactive with a strong inhibition bias h implemented as a synaptic projection from a bias unit, which is defined as follows:

$$-\frac{h}{2} \leq -\max_{i,j,k}(a_k^{i,j} + \lambda_k^{i,j}) \quad \text{with} \quad k = \{x,y\}. \tag{31}$$

This results from the fact that each BSL inputs B_x^i and B_y^i contribute respectively to half of the necessary excitation $(\frac{h}{2})$, that sum up and counterbalance the LTL's natural inhibition (refer to Equation 27 and Equation 28). The LTL units also receive inputs from the MCL units (c_x, c_y), which are respectively modulated by the synaptic weights $(\lambda_x^{i,j}, \lambda_y^{i,j})$ and once the LTL units cross their threshold (mediated by the ramp activation function) then the intrinsic constant LTL neural dynamics $(a_x^{i,j}, a_y^{i,j})$ completes the desired affine-linear transformation. The read-out is an updated encoded machine configuration, which is then synaptically projected back to the MCL units (c_x, c_y), initiating the next computation step (related to the original machine).

2.4 Neuronal Observation Models

In order to compare connectionist simulation results with experimental evidence from neurophysiology or psychology, one needs a mapping from the high-dimensional neural activation space $\Gamma \subset \mathbb{R}^n$ into a much lower-dimensional *observation space* that is spanned by $p \in \mathbb{N}$ observables $\varphi_k : \Gamma \to \mathbb{R}$ $(1 \leq k \leq p)$. A standard method for such a projection is PCA [Elman, 1991]. If PCA is restricted to the first principal axis, the resulting scalar

variable could be conceived as a measure of the overall activity in the neural network (as in beim Graben et al., 2008). Other important scalar observables that have been discussed in the literature are Smolensky's harmony [Smolensky, 1986]

$$H = \sum_{ij} u_i w_{ij} u_j$$

with $\boldsymbol{u} = (u_i)$ as the network's activation vector and $\boldsymbol{W} = (w_{ij})$ its synaptic weight matrix, or Amari's mean network activity [Amari, 1974]

$$A = \frac{1}{n} \sum_i u_i \, . \tag{32}$$

The development of biophysically inspired observation models is an important research field in computational neuroscience [beim Graben and Rodrigues, 2013] as it could eventually lead to "synthetic" local field potentials (LFPs), electroencephalogram (EEG), or event-related brain potentials (ERPs) [Barrès et al., 2013]. We shall use Amari's measure (32) to derive such synthetic ERPs in what follows.

3 Results

The implementation of the R-ANN discussed in the previous Sections simulates a NDA in real-time and thus simulates its associated machine in real-time. More formally, it can be shown that under the map $\rho(\cdot)$ the commutativity property, $\zeta \circ \rho = \rho \circ \Phi$ (see commutative diagram of Figure 1) is satisfied. The NDA simulation (and thus the machine simulation) by the R-ANN is achieved by a combination of synaptic and neural computation among three neural types (MCL, BSL, and LTL) and with a total of neural units equal to

$$n_{\text{units}} = 2 + n_\alpha + n_\beta + 2 n_\alpha n_\beta + 1 \tag{33}$$

where n_α and n_β are the number of sub-sequences that can appear respectively in the left and right Domain of Dependence of the VS from which the NDA and the R-ANN are constructed. That is, a total of 2 MCL units, $(n_\alpha + n_\beta)$ BSL units, $2 n_\alpha n_\beta$ LTL units and a bias unit, that establish synaptic connections according to a synaptic weight matrix $\mathcal{A}$ of size $(n_{\text{units}} \times n_{\text{units}})$ following the connectivity pattern described in Figure 4. Specifically, the synaptic weights in $\mathcal{A}$ are entries from the set $\{0, 1, \frac{h}{2}, \frac{-h}{2}\} \cup \{a_k^{i,j} - h \mid i =$

$1, \ldots, n_\alpha n_\beta \ , \ j = 1, \ldots, n_\beta \ , \ k = x, y\}$, with the second set being the set of biases. A point worth mentioning is that the original formulation of the NDA relied on a simple Gödel encoding of the machine configurations, but subsequent work highlighted the advantages of using a more flexible representation by employing Cylinder sets, in order to preserve important structural relationships of the symbolic descriptions and to facilitate modeling [beim Graben et al., 2008, 2004, beim Graben and Potthast, 2009]. Our R-ANN can be extended to incorporate a Cylinder set encoding of machine configurations by simply doubling the MCL and LTL layer.

An important modeling issue to consider is that of the halting conditions for the ANN, i.e. when to consider the computation as terminated. VSs, on which NDA and consequently our ANN model depend, do not define explicit halting conditions. However, two equally reasonable choices of halting conditions could be employed as follows. The first one is that of using a *homunculus* [beim Graben et al., 2004], an external observer which decides to intervene on the computation once some condition is met (for example, halting the computation when the input is in a certain region of the unit square). The second one is that of using a fixed point condition: implementing a machine halting state as an Identity branch on the NDA. This way a halting configuration will result in a fixed point on the NDA, and thus on the R-ANN. In other words, the network's computation halts if and only if

$$\zeta_1(x', y') = (x', y'). \tag{34}$$

A halting by *homunculus* could be more appropriate in the context of interactive computation [beim Graben et al., 2008, Wegner, 1998] where constant and non-terminating interaction with the environment is assumed, or in cognitive modeling, where different kinds of fixed points, either desired or unwanted ones, are required in order to describe sequential decision problems [Rabinovich et al., 2008], such as linguistic garden paths [beim Graben et al., 2008, 2004].

We will now present two examples to demonstrate the strength of our developed methodology in mapping automata computation to R-ANN computation in real-time (an additional example on Turing Machines is available in the supplementary materials). The source code for all the examples is freely accessible via Carmantini [2015].

3.1 Example 1: Finite-State Locomotive Pattern Generator

FSMs are at the basis of many state-of-the-art approaches to the construction of locomotion controllers for articulated robots (see for example Alvarez-Alvarez et al., 2012, Collins and Ruina, 2005). They are easy to design, implement, and debug, and their relation with animal gait is well characterized [McGhee, 1968]. On the other hand, recent research in robot locomotion control shows an increasing interest towards alternative approaches based on CPGs, neural networks capable of producing rhythmic patterns of activation in absence of rhythmic input sources. In his 2008 paper, Ijspeert presented the benefits and drawbacks of CPGs with respect to other approaches for robot locomotion control. We briefly summarize the benefits identified by the author: i) the rhythmic behavior supported by CPGs is robust to the transient perturbation of state variables; ii) CPGs are well-suited for distributed implementations (such as in modular robots); iii) CPGs reduce the dimensionality of the control problem by introducing few high-level control parameters allowing for the modulation of the locomotion; iv) CPGs are ideally suited for the integration of sensory feedback through coupling terms in the differential equations of the controller; v) CPGs often work well with learning and optimization algorithms. On the other hand, as specified by the author, CPG-based approaches are still lacking of a sound design methodology and theoretical grounding for their description. In the example presented in this Section, we will show how our mapping could aid the design of CPGs producing arbitrary patterns for locomotion in robots, starting from a FSM description of the desired rhythmic pattern. By combining the two approaches, the design of these controllers benefits from the solid theoretical grounding of FSM-based locomotion and from its ease of design and implementation. To contextualize our derived CPG in terms of familiar animal locomotion, we qualitatively model the results of a well-known experiment on cat gait.

In their seminal work, Shik et al. [1966] applied different levels of electrical stimulation to the midbrain of a decerebrated cat. The authors observed transitions in the gait of the animal as an increasing level of stimulation was applied, eliciting first a *walk*, then a *trot* and finally a *gallop* gait. Our theoretical framework can qualitatively reproduce these experimental observations, by deriving a R-ANN which generates the relevant gait patterns, and reproduces the transition between them as a function of the applied

Symbols	States			
	q_1	q_2	q_3	q_4
<lo>	q_3	q_4	q_2	q_1
<hi>	q_2	q_3	q_4	q_1

Table 2: **State transition table for the simulated Central Pattern Generator finite-state automaton.** It is possible to observe how different input leads to different produced patterns, implemented as sequences of states.

stimulus strength. To keep the exposition simple, we will only consider the *walk* and *gallop* gaits, and the transition between the two. In the study of the mammalian quadruped gait, the four legs are numbered so that each gait can be associated with a certain sequence, given by the order in which the legs touch the ground over one gait cycle. The left and right hind legs are associated respectively with the numbers 1 and 2, and the left and right fore legs are associated respectively with the numbers 3 and 4. The gait cycle is assumed to start when the left hind leg touches the ground. A *walk* gait is thus defined by the sequence $(1, 3, 2, 4)$, and a *gallop* gait is defined by the sequence $(1, 2, 3, 4)$. At a very high level, the computation carried out by the CPG in charge of producing the gait patterns in the quadruped mammalian can be informally stated as: if stimulation from midbrain is low, sequentially activate legs following pattern $(1, 3, 2, 4)$. If it is high, sequentially activate legs from pattern $(1, 2, 3, 4)$. We can implement the low level and high level of stimulation as the two input symbols of a FSM, and construct the δ transition function to sequentially reproduce the two patterns by switching between states. The FSM can thus be defined as in Table 2.

This FSM can now be mapped (via our proposed approach) into a R-ANN, consisting in this case of 22 neural units (according to Equation 33). The chosen gamma functions for the Gödel encoding of this FSM are defined as

follows:

$$\gamma_s(\sigma) := \begin{cases} 0 & \text{if } \sigma = \texttt{<lo>} \\ 1 & \text{if } \sigma = \texttt{<hi>} \end{cases} \qquad \gamma_q(q) := \begin{cases} 0 & \text{if } q = q_1 \\ 1 & \text{if } q = q_2 \\ 2 & \text{if } q = q_3 \\ 3 & \text{if } q = q_4 \end{cases}$$

The step-by-step dynamics of the derived R-ANN can be observed in Figure 7. Here we use the machine's input as the substrate for the external stimulus, which is ultimately encoded by the neural unit c_y within our R-ANN as shown in the bottom plot of Figure 7. Note how we manipulate the activation of c_y to gradually increase from a low to a high level of stimulation. That is, we introduce a continuous control parameter into an originally pure symbolic model, enabling us to carry out a bifurcation study in analogy with traditional coupled oscillator models [Collins and Richmond, 1994, Golubitsky et al., 1998, 1999, Schöner et al., 1990]. Under this stimulation, the R-ANN defined by the mapping qualitatively reproduces the key features of the CPG involved in the locomotion and transitions described in Shik et al. [1966]. In particular, it is possible to observe how low levels of stimulation elicit the production of the *walk* gait cycle, whereas an increase in the level of stimulation induces a sudden transition to the *gallop* gait cycle.

This key relation between the stimulation level (i.e a real control parameter) and the computation carried out by the network, which can be related to the underlying symbolic space thanks to the mapping, depends upon an informed decision in the gamma numbering of the states for the Gödel encoding. In fact, the chosen gamma numbering ensures that the unit square encoding of machine configurations where `<lo>` is the current input symbol corresponds to all points (x, y) such that $x < \psi_y(\texttt{<hi>})$ where ψ_y is defined as in Equation 18, and specifically $\psi_y(\texttt{<hi>}) = \gamma_s(\texttt{<hi>})g_s^{-1} = \frac{1}{2}$. In terms of the underlying NDA representation, increasing the activation of c_y until its value reaches and exceeds $\frac{1}{2}$ corresponds to forcing the encoded machine state to cross the boundary between cells associated to a `<lo>` input symbol to those associated to a `<hi>` input symbol, thus causing a transition between a *walk* and a *gallop* gait. Note that in this example, we do not model halting conditions for the derived network, as it is not clear what halting means in the context of the computation performed by CPGs.

To summarize, we derived a CPG from a FSM description of a locomotion controller, inspired by results on the generation of gait patterns in the cat

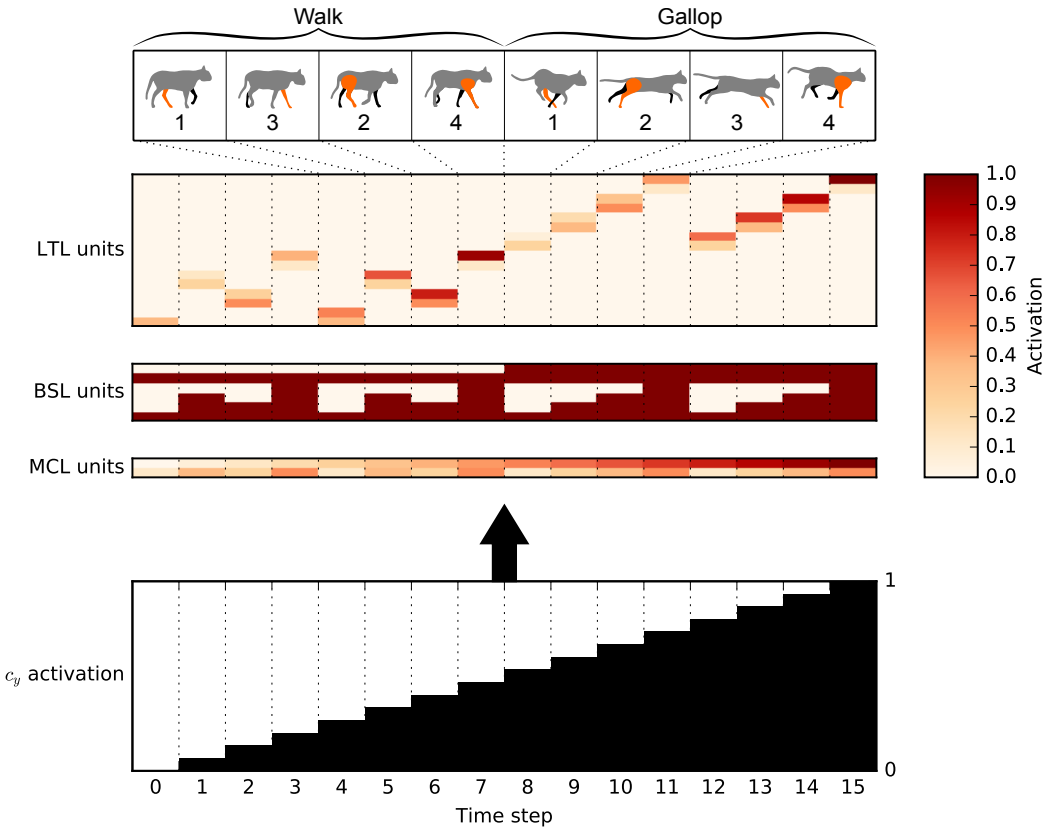

Figure 7: **Recurrent artificial neural network functioning as a Central Pattern Generator.** The network reproduces the qualitative behavior of the locomotive Central Pattern Generator described in Shik et al. [1966]. In the bottom plot, the level of stimulation applied to the network through neuron c_y is shown. In the top three plots, the levels of activation of each neural unit in the three layers is shown for each time step. Note how two different patterns, *walk* and *gallop*, are generated depending on the level of stimulation. This results from the way the original finite-state machine was programmed.

midbrain. By doing so, we outlined a new design methodology for CPG-based locomotion control in robots which does not suffer from some of the drawbacks of other CPG approaches, by grounding the description and design of the CPG on the theoretical grounding of FSM-based approaches. Some problematic aspects of the methodology we outlined are due to the discrete-time nature of our mapping. In fact, fully realizing the benefits of CPG-based approaches summarized at the beginning of this Section requires continuous time models. This notwithstanding, we believe that the proof of concept we provide here already shows encouraging results for future developments.

As an additional remark, the methods we describe in this paper are ideally suited for the deriving of neural networks implementing paradigms of interactive computation, as we will demonstrate shortly. This is especially relevant for the design of CPGs. In fact, recent research has unveiled a surprising degree of hierarchical organization in mammalian respiratory CPGs, which allows for a highly robust and flexible pattern production that can adapt to a variety of conditions (see for example work by Smith et al., 2007, 2013). Our methodology easily accommodates the mapping of hierarchies of automata to hierarchically organized neural networks, as we demonstrate in the next example through the modeling of garden-path parsing, a concept employed in language processing [beim Graben et al., 2004]. Importantly, networks of automata could be used to design complex pattern generation in modular robots (see Spröwitz et al., 2014 for a recent example of modular robots using a distributed CPG for locomotion).

3.2 Example 2: Interactive Automata Networks

Interactive computation [Wegner, 1998] is a recent theoretical development that seeks to formalize the complexity of interactions that we observe in real-world computing. In classical Automata Theory, the interaction between an automaton and the external world is restricted to an input-output relation. That is, the external world provides an input, the automaton performs its computation on that input, and then returns an output to the external world. Within the framework of interactive computation, instead, automata can interact with the external world (and with other automata) at every step of their computation. External forces can act on the configuration of the automaton, and the configuration can itself affect the external world. Clearly, this framework provides a much richer language to describe models of computation, and is especially useful to express notions of compositionality and

concurrency. These constructs are essential not only in the study of modern computing systems, but also in the context of cognitive modeling. In this example, we will build a model of the human processing of locally ambiguous sentences by constructing a network of interactive automata. Through this proof-of-concept, we want to demonstrate the flexibility of our approach by showing how it can be seamlessly used to construct neural networks implementing interactive systems. In order to do so, we choose a system that i) is simple enough to allow for clear exposition, but complex enough to carry out a meaningful computation; ii) is composed by a range of different automata; iii) incorporates different forms of interaction between its automata components.

Garden-path sentences are locally ambiguous sentences that induce the temporary production of an erroneous parse by the reader, which is then forced to reconsider their interpretation of the previously presented material in order to finally reach a correct parse. Consider for example the sentence "I convinced her children are noisy". In reading the sentence, the reader first constructs an intermediate parse where "her children" is the object of the phrase "I convinced". After reading the rest of the sentence, the reader realizes that the intermediate parse was incorrect: "her" is the object of "I convinced", and "children are noisy" is a subordinate clause. The reader thus reanalyzes the sentence to produce a correct parse. Osterhout et al. [1994] have shown that the reanalysis of a sentence due to a garden-path is associated in the brain of the reader with a positive deflection 600 milliseconds (P600) after the onset of a garden-path – the word "are" in the example above – in sequentially presented sentences, as measured by a trial averaged electroencephalogram (thus obtaining event-related brain potentials).

Many proposals have been advanced to account for the mechanisms underlying the reanalysis of incorrectly parsed sentences due to garden-path effects. In our model, we implement the reanalysis through a diagnosis and repair mechanism, described in Lewis [1998]. By this account, the parser tries to incrementally build a parse as the sentence material is presented. If a dead-end is reached (i.e. the parser becomes stuck in a garden-path), the parser diagnoses the need for reanalysis, and the search space of possible continuations of the parse is modified by some repair operator that "bridges" the dead-end to another point in the search space, allowing the parser to correctly complete the processing of the sentence. The parser model we create implements this mechanism to process garden-path sentences where the local ambiguity is given by the incorrect assignment of the subject and object

grammatical constituents.

In many languages, native speakers have been shown to prefer to interpret an ambiguous nominal constituent as a subject rather than an object. Consider for example the following two sentences, extracted from the ERP study on ambiguous pronouns by Frisch et al. [2004] on German speakers. Both sentences start with

Nachdem	*die Kommissarin*	*den Detektiv*	*getroffen hatte*	...
After	the cop	the detective	had met	...

"After the cop had met the detective, ..."

One of the sentences then continues with a clause in subject-object order (s-o sentence), i.e. the preferred order in the parsing of ambiguous constituents:

[(1)]

(s-o sentence)

...	*sah*	*sie*$_{\mathbf{s}}$	*den Schmuggler*$_{\mathbf{o}}$
...	saw	she	the smuggler
...	"she saw the smuggler"		

In this case, the reader correctly interprets "sie" to be the subject of the second clause, and "den Schmuggler" as the object (as "den Schmuggler" is in the accusative case, thus specifying a direct object to the verb "sah").
The second sentence is instead in the dispreferred object-subject order (o-s sentence):

[(1)]

(o-s sentence)

...	*sah*	*sie*$_{\mathbf{o}}$	*der Schmuggler*$_{\mathbf{s}}$
...	saw	she	the smuggler
...	"the smuggler saw her"		

The psycholinguistic study by Frisch et al. [2004] has shown that the reader first tries to apply the preferred subject-object parsing strategy to this clause (and sentences with similar subject/object pronoun ambiguity). The reader thus initially interprets "sie" as the subject of the clause in nominative case, expecting it to be followed by the object in accusative. Upon further reading, however, they realize that "der Schmuggler" is in the nominative case instead, and thus has to be the subject. This leads the reader to reconsider the

previous material to correctly parse "sie" as a pronoun in accusative case, the direct object of the verb "sah". This reanalysis was observed as a P600 effect in the ERP.
At a high level of abstraction [beim Graben et al., 2004], we can capture the structure of these sentences through a CFG G with production rules:

$$\mathtt{S} \rightarrow \mathtt{s}\ \mathtt{o} \qquad (s\text{-}o)$$
$$\mathtt{S} \rightarrow \mathtt{o}\ \mathtt{s}, \qquad (o\text{-}s)$$

where S is a distinguished starting non-terminal, and where the s and o terminals stand respectively for "subject" and "object" phrase.

In our model we thus split the G grammar into two grammars $G_{s\text{-}o}$ and $G_{o\text{-}s}$, comprising respectively of the *s-o* and *o-s* production rules [beim Graben et al., 2004], and reflecting the existence of two strategies in the parsing of sentences with subject/object pronoun ambiguity. To recognize the two different sentence structures, our model is endowed with two specialized TDRs, constructed from the $G_{s\text{-}o}$ and $G_{o\text{-}s}$ grammars as shown in section 2.1.2. Initially, the *s-o* TDR is tried on the input, to model the subject-object interpretation preference. In case it fails because of a garden path, the model acts as prescribed by a diagnosis and repair account. That is, it first diagnoses that a problem has arisen in parsing, repairs the parse, and finally switches strategy to correctly parse the input. In order to implement the diagnosis step, our model needs a way to monitor the state of the parse and extract the relevant diagnostic information. We implement this through a *Diagnosis* PDA (see Table 3), which compares the current parse with that from the previous time step; if the parse didn't change, that means that the parser is stuck and can't process the input further. In that case the *Diagnosis* PDA changes its state to an "error" state, thus implementing the diagnosis step. The repair step is realized by introducing a *Repair* VS , that can be described by the following rewriting rule:

$$\mathtt{s}\ \mathtt{o}\ .\ w \rightarrow \mathtt{o}\ \mathtt{s}\ .\ w, \qquad (35)$$

corresponding to a reanalysis of the ambiguous sentence in terms of the dispreferred object-subject sentence structure. Once the sentence has been reanalyzed and thus the parse repaired, the second parser can proceed to process the input until it has been completely consumed and the stack is emptied. In order to switch strategies, our model needs a higher-level controller that has access to diagnostic information about the current parse, and decides which

Symbols	States		
	${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{pda}}q_{\text{error}}$
$(\sqcup, \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$
(x, y)	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$
$(x, x);\ x \neq \sqcup$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$

Table 3: **State transition table for the *Diagnosis* push-down automaton (PDA).** The input to this machine is the parse produced by the top-down recognizers (TDRs). For any state, input symbol and stack symbol, the machine pushes its input to the stack, in order to be able to compare its current input with the one from the previous time step. In particular, if the input symbol and top-of-stack are blank symbols, the machine transitions to an "idle" state, signaling that nothing is happening; if the current input and the one from the previous time step are different, the machine transitions to a "parsing" state, signaling that the TDRs are successfully parsing their input; if the current input and the one from the previous time step are the same (but not both blanks), then the TDR parsing the input is stuck, and the machine transitions to an "error" state.

Symbols	States		
	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{repair}}$
${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$
${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$
${}^{\text{pda}}q_{\text{error}}$	${}^{\text{fsm}}q_{\text{repair}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$

Table 4: **State transition table for the *Strategy* finite-state machine (FSM).** The input to this machine is the diagnostic information produced by the *Diagnosis* push-down automaton (PDA), i.e. its state. The FSM starts in state ${}^{\text{fsm}}q_{\text{s-o}}$. In fact, the preferred parsing strategy is that implemented by the *s-o* top-down recognizer (TDR), corresponding to the parsing of subject-object sentences, so that it is tried first. If the *s-o* TDR fails, the *Diagnosis* PDA signals an error; the input sentence is not in subject-object order, and a switch of parsing strategy is needed. The *Strategy* FSM first changes state to ${}^{\text{fsm}}q_{\text{repair}}$, activating the *Repair* versatile shift (VS) so that the switch can take place. Repairing the parse leads the *Diagnosis* PDA to signal that the parsing started again, so that the new input for the *Strategy* FSM becomes again ${}^{\text{pda}}q_{\text{parsing}}$. Given ${}^{\text{pda}}q_{\text{parsing}}$ in input and ${}^{\text{fsm}}q_{\text{repair}}$ as a current state, the FSM moves to the ${}^{\text{fsm}}q_{\text{o-s}}$ state, leading to the activation of the *o-s* TDR, until the input has been parsed.

parsing strategy to apply. In particular, this controller should first activate the preferred *s-o* TDR. If the parser failed (as signaled by the *Diagnosis* PDA) then the higher-level controller should first activate the *Repair* VS to allow for the reanalysis of the ambiguous sentence, and subsequently activate the *o-s* TDR. We implement the high level controller through a *Strategy* FSM (see Table 4), endowed with the capability of selectively activating the *s-o* and *o-s* TDRs, as well as the *Repair* VS, by switching its internal state. This machine receives the diagnostic information provided by the *Diagnosis* PDA as input. The FSM has three states, namely an "s-o" state, a "repair" state, and an "o-s" state. By switching between these states, the FSM can activate the respective automata. Note that this form of interaction is not defined for the VS introduced in Section 2.1.1. That is, we do not define a way for a VS to "call" other shifts. Extending VSs to incorporate notions of compositionality and concurrency will allow the refining of the mapping

presented in this paper to reflect these new capabilities. For the moment, we just want to demonstrate the possibilities opened by the present work; for this reason, we will implement the "subroutine" capability in our neural network through a familiar mechanism already encountered in the previous Sections, ignoring momentarily the missing theoretical details and leaving their definition for future work.

To avoid race conditions, at most one automaton in the interactive network can re-write symbols in a sub-sequence at any given computation step . The "parse" sub-sequence can only be read, but not re-written, by the *Diagnosis* PDA. Similarly, the "diagnosis" sub-sequence can only be read, but not re-written, by the *Strategy* FSM. Furthermore, the selective activation of the *s-o* TDR, the *o-s* TDR, and the *Repair* VS operated by the *Strategy* FSM ensures that at any given computation step only one between these automata can perform symbolic re-writing on the "input" and "parse" sub-sequences.

To map the system of interactive automata to a R-ANN, we first convert each of its component in the familiar way, as described in the previous Sections. That is, the *s-o* and *o-s* TDRs, the *Repair* VS, the *Diagnosis* PDA, and the *Strategy* FSM are first converted to VSs acting on dotted sequences, then mapped to their NDA representation and finally to R-ANNs. The Gödelizations of the "input", "parse" and "strategy" sub-sequences are defined as in Equation 18, with each of the gamma enumerating functions defined as follows:

$$\begin{aligned} \gamma_{\text{input}} &:= \{(\sqcup, 0), (\mathtt{S}, 1), (\mathtt{o}, 2), (\mathtt{s}, 3)\} \\ \gamma_{\text{parse}} &:= \{(\sqcup, 0), (\mathtt{o}, 1), (\mathtt{s}, 2)\} \\ \gamma_{\text{diagnosis}} &:= \{({}^{\text{pda}}q_{\text{idle}}, 0), ({}^{\text{pda}}q_{\text{parsing}}, 1), ({}^{\text{pda}}q_{\text{error}}, 2)\} \end{aligned} \tag{36}$$

where each function is represented as a set of (σ, k) pairs, with σ being a symbol and $k \in \mathbb{N}$ its enumeration. The Gödelization of the "diagnosis" sub-sequence is instead defined as in Equation 19, with

$$\gamma_{\text{strategy}} := \{({}^{\text{fsm}}q_{\text{s-o}}, 0), ({}^{\text{fsm}}q_{\text{o-s}}, 1), ({}^{\text{fsm}}q_{\text{repair}}, 2)\}$$

enumerating the states of the *Diagnosis* PDA, and γ_{parse} (already defined in Equation 36) enumerating its stack symbols. Having mapped each of the machines to a R-ANN, we can use the derived networks as components of the overall system architecture (see Figure 9 for the full architecture). In order to simplify the exposition, we construct the overall network to feature only one set of recurrent connections. To do so, we endow our architecture with 4

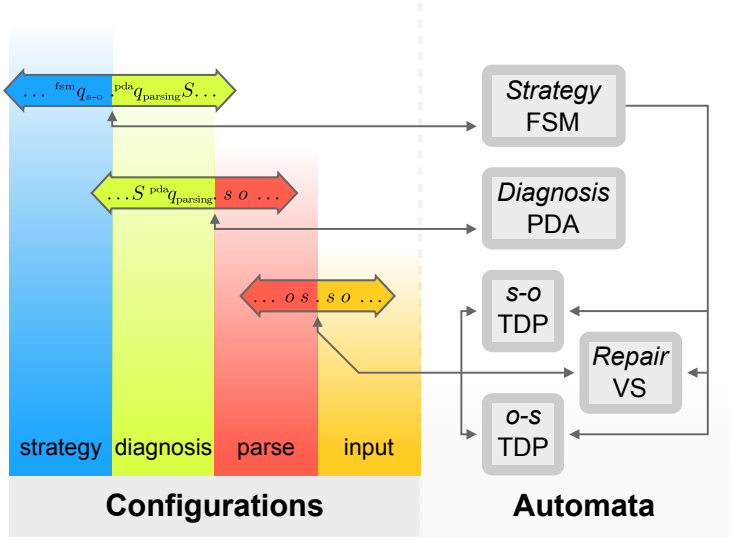

Figure 8: **Interactive automata network for parsing of garden path sentences.** The figure shows the complete system described in Section 3.2. For simplicity, we show the various automata components as acting on their configurations represented as dotted sequences. Dotted sub-sequences of the same color are coupled, i.e. they are for all intents and purposes the same sub-sequence. For example, the "parse" dotted sub-sequence that contains the current stack of the top-down recognizers (TDRs) and of the *Repair* versatile shift (VS), is at the same time the input tape of the *Diagnosis* push-down automaton (PDA). Similarly, the "diagnosis" sub-sequence that stores the current state and stack of the *Diagnosis* PDA, is at the same time the input tape of the *Strategy* finite-state machine (FSM). Note that a second form of interaction, other than that allowed through the sharing of dotted sequences, is present in the automaton. In particular, the *s-o* and *o-s* TDRs and the *Repair* VS are activated based on the state of the *Strategy* FSM.

Configuration Layers (CLs), containing the "strategy", "diagnosis", "parse", and "input" sub-sequences. Between each CL and the next, the network components derived from the automata are connected to perform their part of the processing on the relevant subsequences. In particular, if the VS representation of an automaton acts on some $\alpha.\beta$ dotted sequence, the input of its associated network component is connected to the units encoding the α and β subsequences in the i-th CL, whereas its output (which is a recurrent connection to the MCL layer in the original mapping) is connected to the units encoding α and β in the $(i+1)$-th CL. The final CL is connected with a $\mathcal{I}_{4\times4}$ synaptic weight matrix to the first CL layer (i.e. each unit encoding a subsequence of the last CL is connected with a weight of 1 to the same unit in the first CL). Finally, to implement the subroutine call capabilities of the strategy FSM, we add a *Meta* branch selection layer that takes the "strategy" subsequence as input, and is connected with the lateral inhibition connection pattern specified in Section 2.3.3 to the *s-o* and *o-s* TDRs, and to the *Repair* VS. Note how this creates a nested structure, with the *s-o* TDR, the *o-s* TDR, and the *Repair* VS functioning as higher-level symbolic operations of a *Parser* machine. This is reflected in the nested structure of the *Parser* R-ANN sub-network, where the lower-level machines function as cells in a LTL, controlled by the *Meta* BSL (see Figure 9).

In Figure 10 we show the network activation when two different sentence structures are presented in input. In particular, note the serial activation of the *s-o* TDR, *Repair* VS and *o-s* TDR sub-networks when a object-subject sentence is presented. By mapping the parser from a machine evolving in a symbolic space to a neural network evolving in a vectorial space, we are now able to compute synthetic event-related potentials, or "synth-ERPs", [Barrès et al., 2013, beim Graben et al., 2008] as trial-averages of the mean network activation, as discussed in Figure 11. This is achieved by calculating the mean global network activation according to Amari [1974] (Equation 32) for a simulation over 100 trials for each input stimulus, where random initial conditions compatible with the symbologram representation of the input are prepared according to beim Graben et al. [2008]. In brief, symbologram-compatible random initial conditions are generated through the Gödelization of sequences of the form $w_\alpha u.vw_\beta$, where $u.v$ is the dotted sequence describing the input to the system, and $w_\alpha, w_\beta \in \mathbf{A}^*$ are random sequences of symbols in $\mathbf{A}$.

As Figure 11 reveals, the network shows a P600-like effect in the processing of garden-path sentences, with a peak of increased and sustained

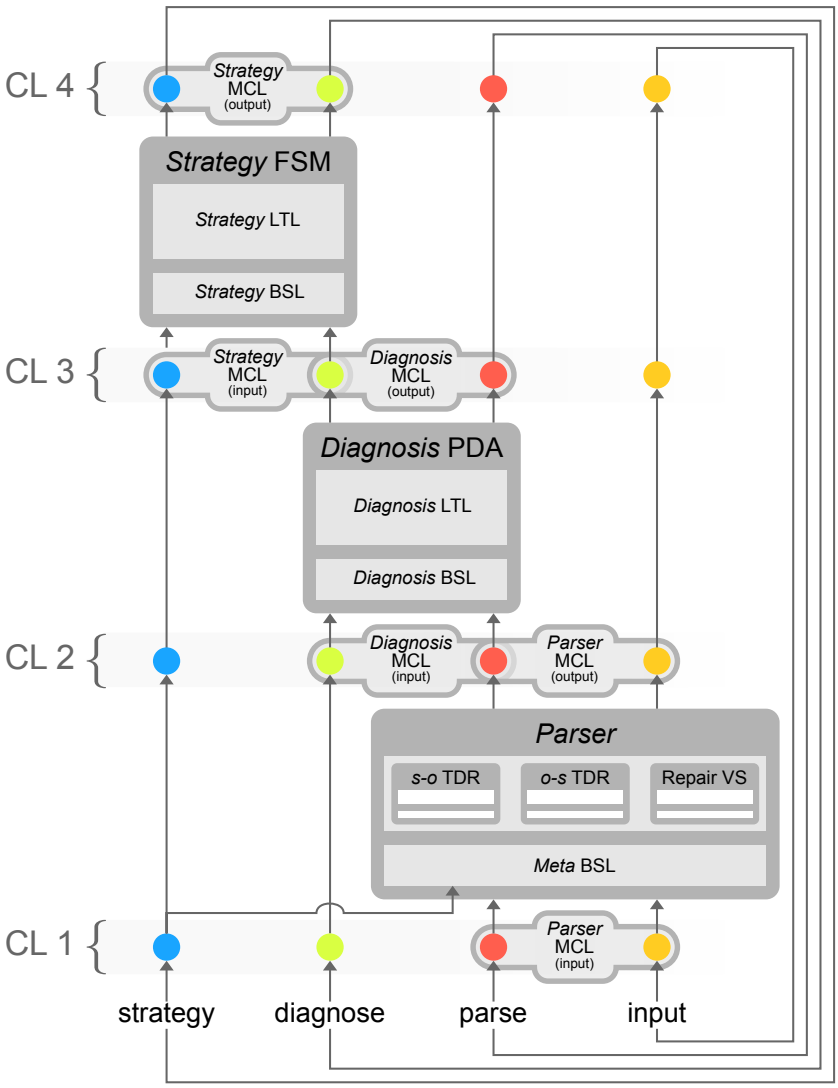

Figure 9: **Garden-path parsing network architecture.** In order to simplify the exposition, we construct our network such that the only recurrent connection is that from the last to the first layer of the network (i.e. CL 4 to CL 1, where CL stands for Configuration Layer). Note that the *Parser* sub-network is itself composed of the *s-o* top-down recognizer (TDR), the *o-s* TDR, and the *Repair* versatile shift (VS) sub-networks. These are arranged as cells of a linear transformation layer (LTL), and selectively activated by a *Meta* branch selection layer (BSL) controlled by a "strategy" neural unit.

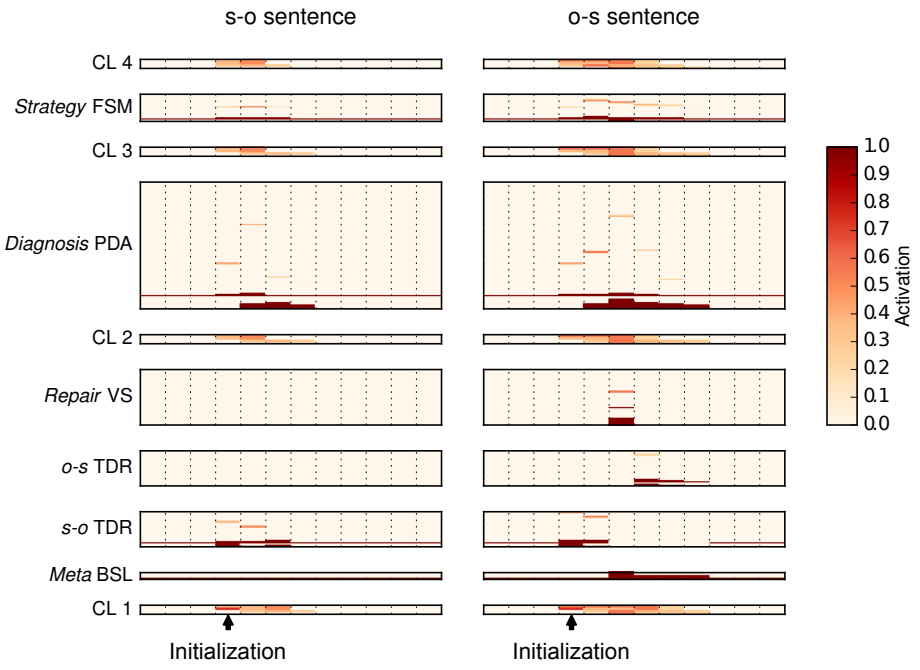

Figure 10: **Network activation for subject-object and object-subject sentence presentation.** Notice the serial activation of the *s-o*, *Repair* and *o-s* sub-networks in case of an object-subject sentence presentation, and the longer "tail" of activation, reflecting the additional computation needed to process the dispreferred input.

activation with respect to the control condition. The simplified model of garden-path processing we presented here does not yet allow for a direct quantitative comparison with experiments such as in Frisch et al. [2004] (in fact, a carefully crafted model would require a level of detail and attention which goes beyond the scope of this paper). Yet, these simulations could be the starting point for more detailed statistical correlation analyses [beim Graben and Drenhaus, 2012, Frank et al., 2015] in future work, relating these computations to electrophysiological measurements.

4 Discussion and Outlook

In this study we have developed a constructive, transparent, modular and parsimonious mapping from symbolic algorithms to neural networks. We first introduced a novel shift map, the versatile shift, that extends the generalized shift and allows for the real-time simulation of a range of symbolic models of computation. We then showed how VSs can be represented on a vectorial space through Gödelization, obtaining piecewise affine-linear systems on the unit square known as nonlinear dynamical automata [beim Graben et al., 2008, 2004, Tabor, 2000, Tabor et al., 2013]. Finally, we presented a modular R-ANN architecture that simulates the dynamics of NDA. The proposed architecture consists of three layers: a machine configuration layer representing the NDA state, and thus the symbolic data in the simulated automaton; a branch selection layer implementing the NDA switching rule, thus characterizing the automaton's decision space, or control; and the linear transformation layer implementing the set of piecewise affine-linear functions in the NDA, i.e. the vectorial representation of the symbolic operations defined in the transition table of the simulated automaton. Additionally, the linear transformation layer is itself modular, in that each operation specified by the δ transition function of the simulated automaton is applied by a specific pair of units in the layer.

The mapping can be used to simulate any Turing machine through R-ANNs, thus making the architecture universal (an example of the mapping on Turing Machines is reported in the supplementary materials). In particular, it is possible to simulate the 7-states 4-symbols UTM by Minsky [1962] in real-time with a R-ANN consisting of 259 units [6] (see Equation 33), and the

[6] This implies a reduction factor of 1/3 when compared to the solution by Siegelmann and Sontag [1991, 1995], which simulates Minsky's UTM with a network of 886 units.

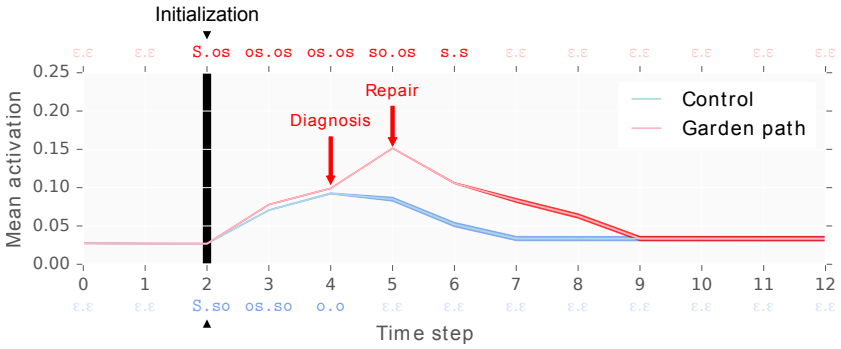

Figure 11: **Synthetic P600 event-related brain potential as mean Network activation for random cloud of initial conditions.** In this figure we show the mean global network activation calculated through Equation 32 for each time step of two simulations, averaged over 100 trials. For each of the two simulations, we run the network presenting at time $t = 2$ one of 100 random inputs generated compatibly to the symbologram representation of one of two sequences. In other words, noise is added to each input such that, if the input was generated by Gödelizing a sequence of length n, decoding the input would yield the original sequence in the first n symbols, with the rest being a random symbolic continuation. If stronger noise was added instead, that would have prevented the network to correctly perform its computation, as we would have destroyed essential input information. In blue, we show the averaged mean activation (light blue) and its standard deviation (dark blue) for a presented input encoding the sequence `S.so`, representing an input sequence in subject-object order, i.e. the network's preferred order as explained in Section 3.2. Note that the parsing is completed at $t = 5$. In red, the averaged mean activation (light red) and its standard deviation (dark red) for an input encoding the sequence `S.os`, representing an input sequence in object-subject order, leading to a garden path in the parsing of the input. The time at which the diagnosis ($t = 4$) and repair ($t = 5$) steps are carried out in the symbolic interactive system (and thus in its recurrent artificial neural network mapping) is indicated by arrows. We also report, at the top and bottom of the plot, the configuration of the parser networks as a dotted sequence for each time step, for respectively the garden path and the control condition. Note how the garden-path processing is associated with a strong divergence in activation starting from time $t = 5$, and followed by a longer tail than that of the network in the control (preferred) condition. This reflects the additional computation needed by the network to successfully resolve the garden path in parsing, and qualitatively corresponds to the P600 event-related brain potential measured in psycholinguistics experiments (see Section 3.2). Furthermore, note that in both conditions the network starts and returns to a "resting state", waiting for input to process from the external world, implementing a notion of continuous computation which is the hallmark of interactive systems.

6-states 4-symbols UTM by Neary and Woods [2009] with one consisting of 223 units.

It is important to analyze some of the modeling choices that have been made in the R-ANN architecture we described. A choice worth discussing is that of implementing biases as synaptic projections from an always-active unit as opposed to implementing them as parameters intrinsic to the individual units. We decided for simplicity to add a bias unit. Nonetheless, a parameterized bias would have been equally reasonable. While it does not have strong bearings on the model here discussed, it is interesting to note that the specific choice of implementation does more or less put the emphasis on a predominantly synaptic computation versus a computation which is more distributed between the synaptic and the neuron level, reflecting similar issues to be considered in the biological domain. A second consideration concerns the cell's boundaries in the NDA. In fact, the distance between the right bound of a cell and the left bound of the next one is zero. This poses some challenges, as even extremely small noise on the state vector at a boundary can lead to an erroneous application of the switching rule on the real state, and thus to a disruption of the computation. This is of course reflected in the dynamics of the associated R-ANN as well. Siegelmann and Sontag [1995] solve this issue by using a Cantor encoding as opposed to a simple Gödelization, ensuring a greater than zero distance between two encoded configurations with different leading symbols. The same methods can be applied here. Interestingly, by switching to a Cantor encoding, the Heaviside units in the BSL layer can be substituted with functionally equivalent Ramp units, so that the R-ANN would only make use of linear units à la Siegelmann and Sontag.

We will now first discuss the advantages of our approach over those based on eliminative connectionism, and then the advances that the present work brings to transparent connectionism.

Compared to eliminative approaches, our work allows the direct interpretation of the representations and the dynamics in the derived network in terms of symbolic computation. This has many important consequences. First, while conventional neural networks have to be trained on large data sets (usually using backpropagation or related algorithms, see Werbos, 1990) our method does not require any training, as the synaptic weight matrix is explicitly designed from the machine table of the encoded automaton. Emergent representations and operations are not opaquely encoded in several hidden layers but transparently realized through Gödelization of symbolic configura-

tions. Second, even when considering learning applications – which we plan to explore in future developments – the derived approach could bring about the exciting possibility of a symbolic read-out of a learned algorithm from the network weights; Note that in this architecture all weights are necessarily fixed, with the exception of the connections encoding the symbolic operations in the simulated automaton, i.e. those between the MCL and the LTL layer. Third, anchoring the computation of the network to well-understood computation models is worthwhile when tackling problems that can benefit from the integration of the two perspectives. In the first example, we constructed a R-ANN (24 units) performing a FSM machine computation abstracting a CPG for animal locomotion. FSMs are widely used in locomotion controllers in robotics, because of their simplicity and strong theoretical grounding in relation to animal locomotion. On the other hand, neural implementations of CPG have many desirable characteristics (as discussed in Ijspeert, 2008) that are not present in FSM-based implementations, but they are difficult to engineer. We showed that by integrating the two approaches we can tackle the problem of pattern generation in robotic locomotion more effectively. Of course, a satisfactory solution would entail the use of continuous-time models in the mapping; nevertheless, our preliminary results already present distinct benefits in the integration of the two approaches as compared with their use in isolation. Fourth, having a complete understanding of the network's inner workings allows for the intelligent manipulation of its parameters. In the discussed CPG example, understanding the computation carried out by the derived network allowed us to introduce a continuous control parameter eliciting a bifurcation in the dynamics of the network, as present in systems of coupled nonlinear oscillator models [Collins and Richmond, 1994, Golubitsky et al., 1998, 1999, Schöner et al., 1990], widely studied in the CPG literature.

In regards to previous work on transparent connectionism, our work advances the field in several ways. As a first advancement, by introducing VSs we are now able to use NDA to simulate a broad range of symbolic computation models in real-time, extending the original work by Moore [1990, 1991]. Interestingly, it would be straightforward to define n-sided infinite dotted sequences (where the dot splits a sequence in its n one-sided infinite components), and extended VSs on these. By Gödelization, we would obtain NDA on the n-dimensional hypercube, which could be simulated by R-ANNs through a straightforward extension of the architecture presented in this work. This would further extend the range of real-time simulable computational models to automata with multiple tapes or stacks [Aho, 1969,

Weir, 1994]. Secondly, by basing our construction on NDA, we obtain an architecture characterized by a fully distributed representation coupled with a granular modularity, differentiating our approach from previous work and granting a series of advantages. The mapping is transparent not only with regards to the representations (the data), but also with regards to the symbolic operations defined in the simulated computational model and their control, all clearly localizable in the architecture. We regard this as an advance in itself (in line with the goals of transparent connectionism), but it also allows, for example, for the straightforward mapping of interactive automata networks to R-ANNs. This is of fundamental importance, as the framework of interactive computation provides a rich language for the description of many complex systems, for example in cognitive modeling. In the second example we constructed a network of interacting automata as a diagnosis and repair model [beim Graben et al., 2008, 2004, Lewis, 1998] for the reanalysis of linguistic garden path sentences. The network consisted of three PDA (two of them as TDRs), a VS, and one FSM as a master control program, with each component carrying out a specific and intelligible task in the overall computation. We then mapped this network to a R-ANN (266 units), thus obtaining a symbolic/connectionist implementation of a cognitive model. Interestingly, due to the multiple levels of hierarchical organization that can be present in the automata network (which comprises nesting, as in the diagnosis and repair network) and, thus, in the derived R-ANN, one could even speculate about thermodynamic limit networks when the number of modules approaches infinity, presenting emergent scale-free or small world properties [Albert and Barabási, 2002]. The granular modularity of our approach is also a key advancement when considering the possibility of correlational studies with neurophysiological measurements. In previous work we showed how to devise large-scale biophysical observation models in order to correlate top-down modeling approaches with neurophysiological data obtained from bottom-up measurements [Amari, 1974, beim Graben and Rodrigues, 2013]. The process involves associating neural units of our model with neuronal masses [Jansen and Rit, 1995, Lopes da Silva et al., 1974] or Hebbian cell assemblies [Hebb, 1949, Huyck, 2009, Wennekers and Palm, 2009] in large-scale brain models, as investigated, e.g., in neural field theory. With this setup we then show that our observational models lead to improved interpretation, e.g of "synthetic event-related brain potentials" (as discussed in Section 2.4, see Barrès et al., 2013, beim Graben et al., 2008) as used in computational neurolinguistics studies [Barrès et al., 2013, beim Graben and Drenhaus, 2012,

Gigley, 1985], where mental/cognitive states can be associated to metastable states of a dynamical system. In the second example presented here, we computed Amari's mean activation [Amari, 1974] as an observation model for the diagnose and repair R-ANN, in order to obtain synthetic ERPs [Barrès et al., 2013, beim Graben et al., 2008]. Qualitatively, the computed signal exhibited a similar divergence between conditions as measured in the experiment presented in Frisch et al. [2004]. While preliminary, these are already encouraging results for the development of our approach in this direction. In future work, we envisage that it will be possible to selectively correlate electrophysiological measurements with specific components in a derived R-ANN, as informed by a suitable symbolic model for the computation underlying the measured quantities. As a third point of interest, the architecture presents a clear 2D spatial organization in its layout, particularly at the level of LTL (as highlighted in Figure 6). In a NDA, different transformations are applied based on the position of the Gödelized automaton data on the unit square. In the R-ANN architecture, this is implemented through the BSL, which performs a form of spatial pattern matching, activating a specific pair of units in the LTL through a lateral inhibition mechanism. When considering extensions to models of higher complexity, the functionality of BSL and LTL could be implemented through the use of a grid of units with receptive fields, as defined for example in self-organizing maps (SOMs, see Kohonen, 1982, Kohonen and Somervuo, 1998).

In future work, we plan to overcome fundamental issues with the current model which have bearing both in relation to learning applications and to the extension of the model to continuous dynamics. For what concerns the learning of algorithms from data, the current model suffers from a missing end-to-end differentiability, due to the use of Gödel encodings. This is a serious limitation, as it prevents the use of gradient descent methods for the training of the network's weights. Future work will have to address this limitation, possibly relying on methods of data access and manipulation akin to modern R-ANN approaches such as in Graves et al. [2014], Grefenstette et al. [2015], Joulin and Mikolov [2015], Sukhbaatar et al. [2015], Weston et al. [2014]. Encouraging work on the learning of exponential state growth languages by Fractal Learning Neural Networks [Tabor, 2003, 2011] could also inform a revised trainable architecture.

With regards to the extension of the model to continuous dynamics, there are many ways in which this could be achieved in future work. Importantly, we are mostly interested in extensions to continuous-time models that are

excitable. In such systems, trajectories can be perturbed away from a stable equilibrium (or rest state) and come back to it only after a large excursion (or spike) in the phase space, upon sufficiently strong input; biophysical examples of excitable models were initiated in Hodgkin and Huxley, 1952. One possibility would be to first extend the mapping to discrete-time excitable models (as in map-based neuronal models, see Girardi-Schappo et al., 2013, Ibarz et al., 2011), and then move to continuous time via so-called *suspension* procedures. There are some potential issues in this endeavor. First of all it would be crucial to first explore and understand the possible relationships between excitable regimes in neural models and symbolic dynamics in a computation. That is, to answer the question: how does the excitability property translate in the realm of symbolic computation? We think there could be meaningful answers to this question when tackled through the framework of interactive computation. Another potential issue is that the suspension process is non-unique and non-trivial in the general case; moreover, it does not guarantee that the excitability property will be preserved.

Excitability is a crucial matter when dealing with neural tissue of lower brain structures, such as the Brain stem, where it is possible to neurophysiologically identify clear and small neuronal networks. However, neural networks models are not the most appropriate level of description for higher cortical structures, due to the presence of large and highly interconnected neuronal masses. Models of these structures express slow but large scale processes as measured by LFP/EEG. In this context, an alternative approach to achieve continuous-time dynamics, which we have already explored to some extent in previous work, is by the framework of *heteroclinic dynamics*, where Turing machine configurations can be interpreted as metastable states with attracting and repelling directions [beim Graben and Potthast, 2009, Krupa, 1997, Rabinovich et al., 2008, Tsuda, 2001], or by the framework of multiple-time scale dynamical systems [Desroches et al., 2013, Fernández-García et al., 2015].

Acknowledgements

This research has been supported by a Heisenberg fellowship (GR 3711/1-2) of the German Research Foundation (DFG) awarded to PbG.

References

Aho, A. V. (1969). Nested stack automata. *Journal of the Association for Computing Machinery*, 16(3):383 – 406.

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling*. Prentice-Hall.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47 – 97.

Alvarez-Alvarez, A., Trivino, G., and Cordón, O. (2012). Human gait modeling using a genetic fuzzy finite state machine. *IEEE Transactions on Fuzzy Systems*, 20(2):205–223.

Amari, S.-I. (1974). A method of statistical neurodynamics. *Kybernetik*, 14:201 – 215.

Barrès, V., III, A. S., and Arbib, M. (2013). Synthetic event-related potentials: A computational bridge between neurolinguistic models and experiments. *Neural Networks*, 37:66 – 92.

beim Graben, P. and Drenhaus, H. (2012). Computationelle Neurolinguistik. *Zeitschrift für Germanistische Linguistik*, 40(1):97 – 125.

beim Graben, P., Gerth, S., and Vasishth, S. (2008). Towards dynamical system models of language-related brain potentials. *Cognitive Neurodynamics*, 2(3):229–255.

beim Graben, P., Jurish, B., Saddy, D., and Frisch, S. (2004). Language processing by dynamical systems. *International Journal of Bifurcation and Chaos*, 14(02):599–621.

beim Graben, P. and Potthast, R. (2009). Inverse problems in dynamic cognitive modeling. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 19(1):015103.

beim Graben, P. and Rodrigues, S. (2013). A biophysical observation model for field potentials of networks of leaky integrate-and-fire neurons. *Frontiers in Computational Neuroscience*, 6(100).

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798 – 1828.

Blutner, R. (2011). Taking a broader view: Abstraction and idealization. *Theoretical Linguistics*, 37(1-2):27 – 35.

Cabessa, J. and Siegelmann, H. T. (2012). The computational power of interactive recurrent neural networks. *Neural Computation*, 24(4):996–1019.

Cabessa, J. and Villa, A. E. (2012). The expressive power of analog recurrent neural networks on infinite input streams. *Theoretical Computer Science*, 436:23–34.

Cabessa, J. and Villa, A. E. (2013). The super-turing computational power of interactive evolving recurrent neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 58–65. Springer.

Carmantini, G. S. (2015). Turing neural networks. *GitHub repository*. `https://github.com/TuringMachinegun/Turing_Neural_Networks`.

Carmantini, G. S., beim Graben, P., Desroches, M., and Rodrigues, S. (2015). Turing computation with recurrent artificial neural networks. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, pages 5 – 13. arXiv:1511.01427 [cs.NE].

Christiansen, M. H. and Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23(4):157 – 205.

Collins, J. J. and Richmond, S. A. (1994). Hard-wired central pattern generators for quadrupedal locomotion. *Biological Cybernetics*, 71(5):375 – 385.

Collins, S. H. and Ruina, A. (2005). A bipedal walking robot with efficient and human-like gait. In *ICRA 2005. Proceedings of the 2005 IEEE International Conference on Robotics and Automation.*, pages 1983–1988. IEEE.

Davis, M. D., Sigal, R., and Weyuker, E. J., editors (1994). *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, Harcourt, Brace and Company.

Desroches, M., Krupa, M., and Rodrigues, S. (2013). Inflection, canards and excitability threshold in neuronal models. *Journal of Mathematical Biology*, 67(4):989–1017.

Dominey, P. F. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202 – 1205.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179 – 211.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195 – 225.

Elman, J. L. (1995). Language as a dynamical system. In Port, R. F. and van Gelder, T., editors, *Mind as Motion: Explorations in the Dynamics of Cognition*, pages 195 – 223. MIT Press, Cambridge (MA).

Farkas, I. and Crocker, M. W. (2008). Syntactic systematicity in sentence processing with a recurrent self-organizing network. *Neurocomputing*, 71:1172 – 1179.

Fernández-García, S., Desroches, M., Krupa, M., and Clément, F. (2015). A multiple time scale coupling of piecewise linear oscillators. application to a neuroendocrine system. *SIAM Journal on Applied Dynamical Systems*, 14(2):643–673.

Frank, S. L., Otten, L. J., Galli, G., and Vigliocco, G. (2015). The ERP response to the amount of information conveyed by words in sentences. *Brain and Language*, 140:1 – 11.

Frisch, S., beim Graben, P., and Schlesewsky, M. (2004). Parallelizing grammatical functions: P600 and P345 reflect different cost of reanalysis. *International Journal of Bifurcation and Chaos*, 14(2):531 – 549.

Gayler, R. W. (2006). Vector symbolic architectures are a viable alternative for Jackendoff's challenges. *Behavioral and Brain Sciences*, 29:78 – 79.

Gayler, R. W., Levy, S. D., and Bod, R. (2010). Explanatory aspirations and the scandal of cognitive neuroscience. In Samsonovich, A. V., Johannsdottir, K. R., Chella, A., and Goertzel, B., editors, *Proceedings of the 2010 conference on Biologically Inspired Cognitive Architectures 2010: Proceedings of the First Annual Meeting of the BICA Society*, pages 42 – 51, Amsterdam. IOS Press.

Gigley, H. M. (1985). Computational neurolinguistics: What is it all about? In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, volume 1 of *IJCAI'85*, pages 260 – 266, San Francisco (CA).

Girardi-Schappo, M., Tragtenberg, M., and Kinouchi, O. (2013). A brief history of excitable map-based neurons and neural networks. *Journal of neuroscience methods*, 220(2):116–130.

Gödel, K. (1931). Über formal unentscheidbare Sätze der *Principia mathematica* und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173 – 198.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1998). A modular network for legged locomotion. *Physica D*, 115(1-2):56 – 72.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1999). Symmetry in locomotor central pattern generators and animal gaits. *Nature*, 401(6754):693 – 695.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *preprint*. arXiv:1511.01427 [cs.NE].

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827.

Grillner, S. and Zangger, P. (1975). How detailed is the central pattern generation for locomotion? *Brain Research*, 88(2):367 – 371.

Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York (NY). Partly reprinted in J. A. Anderson and E. Rosenfeld (1988), pp. 45ff.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Lecture Notes of the Santa Fe Institute Studies in the Science of Complexity. Perseus Books, Cambridge (MA).

Hinaut, X. and Dominey, P. F. (2013). Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing. *PLoS ONE*, 8(2):e52946.

Hinaut, X., Petit, M., Pointeau, G., and Dominey, P. F. (2014). Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks. *Frontiers in Neurorobotics*, 8(16).

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison–Wesley, Menlo Park, California.

Huyck, C. R. (2009). A psycholinguistic model of natural language parsing implemented in simulated neurons. *Cognitive Neurodynamics*, 3(4):317 – 330.

Ibarz, B., Casado, J. M., and Sanjuán, M. A. (2011). Map-based models in neuronal dynamics. *Physics Reports*, 501(1):1–74.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.

Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

Jansen, B. H. and Rit, V. G. (1995). Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns. *Biological Cybernetics*, 73:357 – 366.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198.

Kleene, S. (1956). Neural nets and automata. *Automata Studies*, pages 3–43.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Kohonen, T. and Somervuo, P. (1998). Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30.

Krupa, M. (1997). Robust heteroclinic cycles. *Journal of Nonlinear Science*, 7(2):129–176.

Lawrence, S., Giles, C. L., and Fong, S. (2000). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126 – 140.

Lewis, R. L. (1998). Reanalysis and limited repair parsing: Leaping off the garden path. In Fodor, J. D. and Ferreira, F., editors, *Reanalysis in Sentence Processing*, pages 247 – 285. Kluwer, Dordrecht.

Li, D. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3.

Lind, D. and Marcus, B. (1995). *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge (UK). Reprint 1999.

Lopes da Silva, F. H., Hoecks, A., Smits, H., and Zetterberg, L. H. (1974). Model of brain rhythmic activity: The Alpha-rhythm of the thalamus. *Kybernetik*, 15:27 – 37.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531 – 2560.

McClelland, J. L. and Elman, J. L. (1986). The TRACE model of speech perception. *Cognitive Psychology*, 18(1):1 – 86.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133.

McGhee, R. B. (1968). Some finite state aspects of legged locomotion. *Mathematical Biosciences*, 2(1-2):67–84.

Minsky, M. (1962). Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238.

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc.

Mizraji, E. (1989). Context-dependent associations in linear distributed memories. *Bulletin of Mathematical Biology*, 51(2):195 – 205.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354 – 2357.

Moore, C. (1991). Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4:199 – 230.

Neary, T. and Woods, D. (2009). Four small universal Turing machines. *Fundamenta Informaticae*, 91(1):123–144.

Osterhout, L., Holcomb, P. J., and Swinney, D. A. (1994). Brain potentials elicited by garden-path sentences: Evidence of the application of verb information during parsing. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(4):786 – 803.

Rabinovich, M. I., Huerta, R., Varona, P., and Afraimovich, V. S. (2008). Transient cognitive dynamics, metastability, and decision making. *PLoS Computational Biology*, 4(5):e1000072.

Schöner, G., Jiang, W. Y., and Kelso, J. A. S. (1990). A synergetic theory of quadrupedal gaits and gait transitions. *Journal of Theoretical Biology*, 142(3):359 – 391.

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145 – 168.

Shik, M. L., Severin, F. V., and Orlovsky, G. N. (1966). Control of walking and running by means of electrical stimulation of mid-brain. *BIOPHYSICS-USSR*, 11(4):756.

Siegelmann, H. T. and Sontag, E. D. (1991). Turing computability with neural nets. *Appl. Math. Lett*, 4(6):77–80.

Siegelmann, H. T. and Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology Boston.

Smith, J. C., Abdala, A., Koizumi, H., Rybak, I. A., and Paton, J. F. (2007). Spatial and functional architecture of the mammalian brain stem respiratory network: a hierarchy of three oscillatory mechanisms. *Journal of neurophysiology*, 98(6):3370–3387.

Smith, J. C., Abdala, A. P., Borgmann, A., Rybak, I. A., and Paton, J. F. (2013). Brainstem respiratory networks: building blocks and microcircuits. *Trends in neurosciences*, 36(3):152–162.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, chapter 6, pages 194 – 281. MIT Press, Cambridge (MA).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159 – 216.

Smolensky, P. and Legendre, G. (2006a). *The Harmonic Mind. From Neural Computation to Optimality-Theoretic Grammar*, volume 1: Cognitive Architecture. MIT Press, Cambridge (MA).

Smolensky, P. and Legendre, G. (2006b). *The Harmonic Mind. From Neural Computation to Optimality-Theoretic Grammar*, volume 2: Linguistic and Philosophic Implications. MIT Press, Cambridge (MA).

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S., and Ijspeert, A. J. (2014). Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033.

Steil, J. J. (2004). Backpropagation-decorrelation: online recurrent learning with O(N) complexity. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 843–848. IEEE.

Stewart, T. C., Choo, X., and Eliasmith, C. (2014). Sentence processing in spiking neurons: A biologically plausible left-corner parser. In *Proceedings of the Cognitive Science Conference*.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439.

Tabor, W. (2000). Fractal encoding of context-free grammars in connectionist networks. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, 17(1):41 – 56.

Tabor, W. (2003). Learning Exponential State-Growth Languages by Hill Climbing. *IEEE Transactions on Neural Networks*, 14(2): 444–446.

Tabor, W. (2011). Recursion and Recursion-Like Structure in Ensembles of Neural Elements. *Proceedings of the VIII International Conference on Complex Systems*, 1494–1508.

Tabor, W., Cho, P. W., and Szkudlarek, E. (2013). Fractal analyis illuminates the form of connectionist structural gradualness. *Topics in Cognitive Science*, 5:634 – 667.

Tabor, W., Juliano, C., and Tanenhaus, M. K. (1997). Parsing in a dynamical system: An attractor-based account of the interaction of lexical and structural constraints in sentence processing. *Language and Cognitive Processes*, 12(2/3):211 – 271.

Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behavioral and Brain Sciences*, 24:793 – 810.

Turing, A. M. (1937). On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London Mathematical Society*, 42.

Wegner, P. (1998). Interactive foundations of computing. *Theoretical Computer Science*, 192:315 – 351.

Weir, D. J. (1994). Linear iterated pushdowns. *Computational Intelligence*, 10(4):431 – 439.

Wennekers, T. and Palm, G. (2009). Syntactic sequencing in Hebbian cell assemblies. *Cognitive Neurodynamics*, 3(4):429 – 441.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550 – 1560.

Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *preprint*. arXiv:1410.3916 [cs:AI].