Neural Networks 24 (2011) 199–207

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

A neurodynamical model for working memory

Razvan Pascanu [a,*], Herbert Jaeger [b]

[a] DIRO, Université de Montréal, H3T 1J4 Quebec, Canada

[b] Jacobs University Bremen, School of Engineering and Science, 28759 Bremen, Germany

ARTICLE INFO

Article history:
Received 21 March 2010
Received in revised form 28 September 2010
Accepted 8 October 2010

Keywords:
Recurrent neural networks
Echo state networks
Working memory
Attractor

ABSTRACT

Neurodynamical models of working memory (WM) should provide mechanisms for storing, maintaining, retrieving, and deleting information. Many models address only a subset of these aspects. Here we present a rather simple WM model in which all of these performance modes are trained into a recurrent neural network (RNN) of the echo state network (ESN) type. The model is demonstrated on a bracket level parsing task with a stream of rich and noisy graphical script input. In terms of nonlinear dynamics, memory states correspond, intuitively, to attractors in an input-driven system. As a supplementary contribution, the article proposes a rigorous formal framework to describe such attractors, generalizing from the standard definition of attractors in autonomous (input-free) dynamical systems.

1. Introduction

Working memory (WM), citing a version of the standard definition from Durstewitz, Seamans, and Sejnowski (2000), is *"the ability to transiently hold and manipulate goal-related information to guide forthcoming actions"*. We would like to add that the storing should, in principle, be stable (or stabilizable on demand, e.g. by rehearsal) for unbounded time spans. This commitment is implicit in all models of WM that we are aware of.

Most of the recurrent neural network (RNN)-oriented literature on WM is concerned with models to explain behavioral and neural observations from cognitive processing experiments (see the reviews by Durstewitz et al. (2000) and Howard (2009)). This is, however, not the only possible motivation for neural network investigations into WM. One may also investigate WM mechanisms with the aim of improving the engineering of complex signal processing applications. This view is adopted in this contribution. The background is a European research endeavour which aims at establishing neurodynamical models as a viable alternative to the standard hidden Markov model (HMM)-based speech and handwriting recognition technology.[1] Thus, our examples will be taken from that domain.

If the aim is engineering applications, not explaining nature, an immediate question is why one should investigate RNN-based WM models at all. After all, standard digital signal processing systems offer a number of convenient basic types of transient memories (e.g., multistable switching elements, FILO and FIFO buffers, RAMs), and task-specific memory management routines are readily programmed. Thus, what could be a benefit of "going neural"? Our lead in this regard is that

- a potential advantage of complex RNN-based signal processing is the *richness and dynamical adaptivity* of internal representations — in the sense that they are high dimensional, may have interesting and relevant self-organizing properties, are dynamically evolving, and are adaptable through learning;
- items stored in WM should be enabled to *modulate* the ongoing dynamical processing—in the sense of providing context;
- thus, these items should *directly interact* with the RNN that uses them;
- this, in turn, is most naturally warranted if the WM itself is framed as an RNN — be it as a separate one connected with feedback loops to the RNN that does the ongoing processing, or (our approach) as a subdynamics of the latter.

A widely adopted idea is that stable short-term memory is realized in RNNs by way of *attractors*. A diverse zoo of attractors in RNNs has been explored in a variety of contexts. The biologically oriented WM literature mainly appears to be considering point attractors (cell assemblies and bistable neurons) and traveling waves (synfire chains); see the survey of Durstewitz et al. (2000). In theoretical physics, pattern (= spatiotemporal attractor) formation in excitable media is an area of its own standing, with connections into computational neuroscience and robotics via neural field theories of cortical representation (Freeman, 2007a, 2007b; Schöner, Dose, & Engels, 1995). Similarly, the investigation of coupled oscillator networks is a significant research arena in theoretical physics

* Corresponding author. Tel.: +1 514 568 8106.
E-mail addresses: pascanur@iro.umontreal.ca, r.pascanu@gmail.com (R. Pascanu), h.jaeger@jacobs-university.de (H. Jaeger).

[1] www.reservoir-computing.org/organic.

0893-6080/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2010.10.003

with obvious connections to attractors in (mostly spiking) RNNs; see, for instance, Radicchi and Meyer-Ortmanns (2006). Chaotic attractors have been investigated as information representing neural mechanisms at least since Yao and Freeman (1990) and Babloyantz and Lourenço (1994); they are intriguing because of their richness of structure and the option to stabilize and address sublobes as representational subunits (Stollenwerk & Pasemann, 1996; Tsuda, 2001). In machine learning, the task of training an RNN to become a stable pattern generator, i.e., a periodic attractor, has been addressed in many ways (see, e.g., Zegers and Sundareshan (2003)).

Attractors, by definition, keep the system trajectory confined in their support. Since clearly cognitive dynamics does not get ultimately trapped in attractors, it is a long-standing modeling challenge to account for "attractors" that can be left again. Many answers have been proposed. If one adheres to the standard concept of an attractor, neural noise is a plausible agent to "kick" a trajectory out of an attractor. A difficulty with this simple solution is that the effects of noise are unspecific and not easily reconciled with systematic information processing. Departing from the classical notion of attractors, a number of alternative "attractor-like" metastability phenomena have been considered that may arise in high-dimensional nonlinear dynamics: saddle point dynamics (Rabinovich, Huerta, Varona, & Afraimovich, 2008); *attractor relics* (or *attractor ruins*), where classical attractors in a fast-timescale subsystem are destroyed by a slow-timescale saturation dynamics (Gros, 2009); *transient attractors*, defined by transient volume contractions of a flow (Jaeger, 1995); *unstable attractors*, a mathematically surprising kind of classical attractors, which however arise generically in certain spiking neural networks and can be left under the impact of arbitrarily small noise because they are surrounded arbitrarily closely by basins of other attractors (Timme, Wolf, & Geisel, 2002); *high-dimensional attractors* (initially named *partial attractors*), which govern only some dimensions of a high-dimensional phase space (Maass, Joshi, & Sontag, 2007); *attractor landscapes* shaped by control parameter (input) dynamics which lead to the appearance and disappearance of attractors due to incessant bifurcations (Negrello & Pasemann, 2008).

Certainly yet more proposals exist of which we are not aware—they may not be easy to spot because of the diversity of fields where they are described. Biological brains may exploit several of these phenomena simultaneously; evolution does not care about uniform and simple mathematical models. In our opinion, all of this is likely only the beginning of explorations into attractor-like phenomena in high-dimensional dynamics. We suspect that current mathematics is still blind to important phenomena in high-dimensional dynamics which may be important for understanding information processing in RNNs in general and WM in particular, and regard research on the latter, including our own, as provisional.

This article makes two contributions which are quite different in nature. First, we describe a concrete WM mechanism based on echo state networks (ESNs, Jaeger (2001)). This mechanism is obtained through a supervised training procedure, in which not only is the "raw" memory shaped, but additional functionality is trained. In our examples, this additional functionality concerns image processing (for detecting specific bracket symbols in a visual input stream) and counting (to keep track of opening levels of detected brackets). The training scheme is generic and could similarly be employed for other input processing tasks and other operations on memorized elements, provided they can be defined through training examples. The memory items are represented by a dynamical phenomenon related to point attractors, in that each stored item is reflected by locking one of the RNN neurons in a specific "on" state. Much of the reservoir dynamics, however, is left largely unconstrained and remains utilizable for other processing tasks (here, ongoing character recognition and classification of text statistics). Attempting to give a precise mathematical account of this phenomenon, we were guided to the formal model of input-induced attractors. A rigorous definition of these attractors (which we call γ*-attractors*) is the second contribution of this paper.

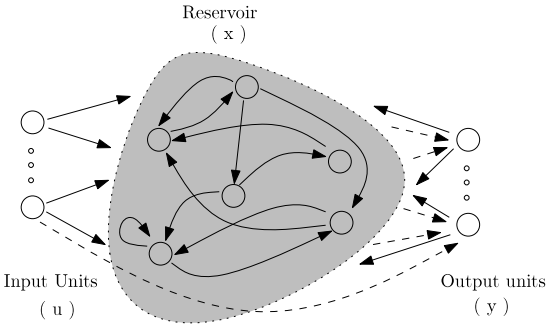

Fig. 1. General structure of an echo state network. Only the connections indicated by dashed lines are trainable (collected in $\mathbf{W}^{out}$).

ESNs are an instance of the *reservoir computing* (RC) principle of designing and training RNNs. According to this principle, a fixed, large, random *reservoir* RNN is excited by input signals, and the desired output is combined from the excited reservoir signals by a trainable readout mechanism (often a simple linear regression). The RC principle has been independently discovered in cognitive neuroscience (*temporal recurrent networks*, Dominey, Arbib, and Joseph (1995)), in computational neuroscience (*liquid state machines*, Maass, Natschläger, and Markram (2002)), and in machine learning (*echo state networks*, Jaeger (2001)). RC has demonstrated its usefulness for a variety of time series processing tasks (Jaeger & Haas, 2004; Lukosevicius, Jaeger, Popovici, & Siewert, 2007; Verstraeten, Schrauwen, & Stroobandt, 2006), is becoming a standard tool in recurrent neural network modeling (Jaeger, Maass, & Principe, 2007), and has spurred an active area of research (for a survey, see Lukosevicius and Jaeger (2009)).

Attractor-based memory phenomena in RC systems were first described in Jaeger (2002), where a multistable switching system was trained. The switching states were associated with specific output neurons, which could be in "on" or "off" states. A similarly switchable system was recently obtained (as one among many other examples) in ESNs by a novel RC training algorithm which may operate simultaneously on the reservoir and the readout, and which has unprecedented stability properties; see (Sussillo & Abbott, 2009). In this system, the switching states were not created and maintained by feedback from specific output neurons, but as complex (and freely combinable) submodes of the reservoir dynamics. The present contribution goes beyond these investigations in that it combines the raw WM functionality with ongoing input data processing and trained interactions between WM states, and thus proposes an advancement on the road to practically usable, RNN-based WM mechanisms.

The article is organized as follows. Section 2 gives a brief introduction to ESNs. Section 3 introduces a model of working memory based on an ESN. This concrete model is followed by our proposal of input-induced attractors in Section 4, together with a brief survey of related mathematical approaches. Section 5 presents a discussion on the results obtained throughout this paper.

2. Echo state networks

Echo state networks (ESNs) are a family of recurrent neural networks composed of a reservoir, a recurrent internal layer of size N, to which K input and L output units are added; for a diagram of this architecture see Fig. 1. If $\mathbf{u}$, $\mathbf{x}$, and $\mathbf{y}$ are the activations of the input, the internal, and the output units, respectively, and $\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{out}$, and $\mathbf{W}^{b}$ are the weights matrices for the input connections, internal connections, output connections, and

feedback connections from the output units back to the reservoir, the system update equations are the following:

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{b}\mathbf{y}(n)) \quad (1)$$

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))). \quad (2)$$

Here, $\mathbf{f}$ subsumes the activation functions of the internal units (usually the hyperbolic tangent) and $\mathbf{f}^{out}$ the activation functions of the output units. $\mathbf{W}^{in}$, $\mathbf{W}$, and $\mathbf{W}^{b}$ are randomly initialized, and only $\mathbf{W}^{out}$ changes during training. For the standard supervised offline learning approach, a training sequence is fed to the network through the input units. The target of the output units is also needed to compute the reservoir activations (see Eq. (1)). We use the target at time step $t-1$ in order to compute the reservoir activation at time step t. The reservoir state vectors, together with the activations of the input units, are stored row-wise in data collection matrix $\mathbf{G}$, and, if $\mathbf{Y}_{target}$ is the target signal, then the output weights are computed using linear regression, as shown in Eq. (3), where † stands for pseudo-inverse.

$$\mathbf{W}^{out} = (\mathbf{G}^{\dagger} \cdot \mathbf{f}^{out^{-1}}(\mathbf{Y}_{target}))^{T}. \quad (3)$$

An important condition to make the learning of output weights by regression a well-defined procedure is the *echo state property* (ESP). This is a property of the reservoir and the admitted input. Roughly stated, a reservoir has the ESP w.r.t. a given admitted input range if for any infinite input sequence the network states $\mathbf{x}(n)$ asymptotically "forget" the (arbitrary) initial state $\mathbf{x}(0)$ used at start-up time. Formal definitions of the ESP are given in Jaeger (2001), and refined algebraic conditions are given in Buehner and Young (2006). In practice, the ESP is usually ensured when the spectral radius of the reservoir weight matrix $\mathbf{W}$ is set to a value below unity, but we emphasize that this is neither a necessary nor a sufficient criterion (Jaeger, 2007), in spite of a folklore belief in the field that it is both.

According to the task that needs to be solved, there are a few global parameters that need to be tuned for optimal learning, namely, global scalings of input weights, reservoir weights, and output feedback weights. In the RC field, the global scaling of the reservoir weights $\mathbf{W}$ is typically specified through the spectral radius of this matrix. All these tunable parameters are explained in more detail in Jaeger (2001).

3. Working memory model

In this section, we present a working memory model based on an ESN which uses a dynamical mechanism related to point attractors for storing information. It is able to simultaneously store information, use the stored information to modulate further storing, detect patterns in the input that trigger memory content switches, and perform input classification.

3.1. Model

Our model is obtained by adding a set of special output units to an otherwise standard ESN. We called these units *WM-units*, and they differ from normal output units by having trainable connection from one to another or to themselves, as in Fig. 2. In our set-up we only allow feedback connections from the WM-units to the reservoir but not from the other, regular output units. Another difference between WM-units and output units is that they are binary-state neurons which can store memory bits. To achieve this behavior we use a sharp threshold function f^{m} as the activation function of these units:

$$f^{m}(x) = \begin{cases} -0.5 & x \leq 0. \\ +0.5 & x > 0. \end{cases} \quad (4)$$

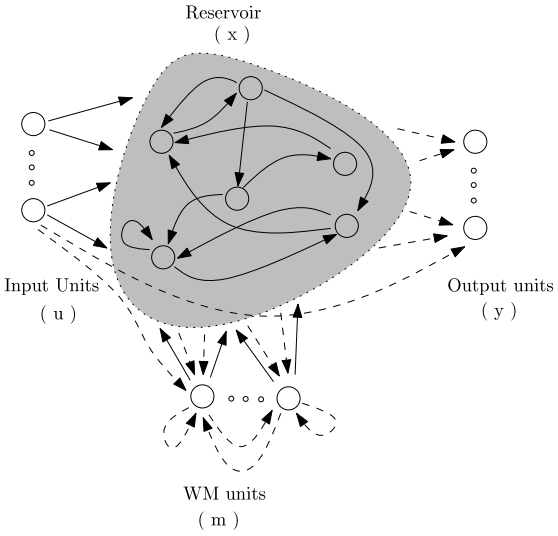

Fig. 2. Diagram of the WM model. Dashed connections are trained; the others are left untouched. Note that the main differences to Fig. 1 are the memory units. They differentiate themselves from output units by having trainable connections among them. Also in our set-up the output units do not have feedback connections.

The network is described by Eqs. (5)–(7), where we use the same conventions as for the ESN (Section 2). Eq. (5) is similar to Eq. (1), with the difference that $\mathbf{W}^{b}$ is the matrix collecting the feedback weights from the WM-units (not output units) to the reservoir, and $\mathbf{m}$ denotes the activations of the WM-units. Eq. (6) is identical to Eq. (2). The activations of the WM-units are computed through (7), where $\mathbf{W}^{mem}$ collects the weights from the input units, the reservoir, and the WM-units to the WM-units.

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{b}\mathbf{m}(n)) \quad (5)$$

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))) \quad (6)$$

$$\mathbf{m}(n+1) = \mathbf{f}^{m}(\mathbf{W}^{mem}(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{m}(n))). \quad (7)$$

To train the model one needs to use Eq. (5) to compute the reservoir response, where $\mathbf{m}$ is replaced by the target of the WM-units (*teacher forcing*). The activations of the reservoir together with the input are collected in the matrix $\mathbf{G}$, while the same activations, the input, and the target of the WM-units are collected in $\mathbf{H}$. Given $\mathbf{Y}_{target}$ the target for the output units and $\mathbf{M}_{target}$ the target of the WM-units, $\mathbf{W}^{mem}$ and $\mathbf{W}^{out}$ are computed using linear regression, as shown in Eqs. (9) and (8). As before, † stands for pseudo-inverse. In (9), we did not pass the target through $\mathbf{f}^{mem^{-1}}$ because there is no inverse for $\mathbf{f}^{mem}$ (as defined in Eq. (4)).

$$\mathbf{W}^{out} = (\mathbf{G}^{\dagger} \cdot \mathbf{f}^{out^{-1}}(\mathbf{Y}_{target}))^{T} \quad (8)$$

$$\mathbf{W}^{mem} = (\mathbf{H}^{\dagger} \cdot \mathbf{M}_{target})^{T}. \quad (9)$$

At first sight, the strong couplings between WM-units through the trained $\mathbf{W}^{mem}$ might appear problematic for a "clean" storing of memory items, because in technical storage devices one does not usually desire dynamical interaction between stored items. However, we will demonstrate that such interactions can be harnessed for realizing desirable processing functionalities which go beyond pure storage and retrieval.

3.2. Experiments

Task. The task is to keep track of the number of opened curly brackets as the system reads a rich graphic script input, one vertical pixel line per time step. An input sample is shown in Fig. 3. The system is required to maintain a counter. Any time an opened curly bracket appears at the input, the counter has to increment

Fig. 3. An fragment example of the rich graphic script used as input. The image was scaled for better visualization.

by 1. When a closed curly bracket appears, the counter needs to decrement by 1. The architecture is required to be able to count up to 6. The input data are generated such that no overflow or underflow occurs. This task requires a persistent memory, as the network must remember the number of brackets seen for unbounded periods of time, but it also requires the ability to do the basic arithmetic operations of adding and subtracting 1.

In addition to this, as a "computational payload", the network also has to predict the next character. This functionality is trained into the "normal" output units. We use the same number of output units as the number of possible characters (excepting the curly bracket characters). Each output unit predicts how probable is that the corresponding character will follow in the input stream. In doing this, the network will benefit from taking into account the current bracket level (the number of unclosed brackets), since the conditional distribution of the next character given the previous differs across the bracketing levels. This additional task is meant to demonstrate that the network is able to use the information stored in the WM-units.

Data. We train the network in two stages. During the first stage, only $\mathbf{W}^{mem}$ is computed. In this stage, training sequences of only 10,000 symbols are used. In the second stage, the weights $\mathbf{W}^{out}$ to the output units are computed. For the second stage, we use sequences of 49,000 characters. In both cases, the input sequences are generated in exactly the same way.

For generating the sequences, characters are chosen randomly with a probability of 70% from a set of 65 different ASCII symbols (letters in lower case, numbers and other symbols typically used in text including other types of brackets and the blank space character), with a probability of 15% an open curly bracket and with 15% a closed curly bracket. The opening and closing curly brackets are inserted such that they form matching pairs with a nesting level of up to 6. According to the nesting level i (which can be between 0 and 6), a different Markov chain is used to sample from the other 65 characters. The Markov chain is defined such that if the current character has the index j (a number between 1 and 65) then the next character will be $j+i+1$ modulo 65 with probability 80% and with equal probability (0.3125%) any of the other 64 characters.

The testing sequence is generated similarly. It has 35,000 characters, picked now with a probability of 94% from the same set of 65 ASCII characters, while curly brackets are picked with a probability of only 6%. The same seven Markov chains are used to sample characters in the periods corresponding to the seven bracket levels.

The symbols are transformed afterwards to images by printing them with a randomly selected font from four different font sets (FreeMono, FreeMono Bold, FreeMono Italic, and FreeMono Bold Italic of Gimp 2.3.6). All fonts have a width of 7 pixels and height of 12 pixels, where each pixel is a grayscale value between 0 and 1. Before printing, the character images are stretched randomly to a width of 6, 7, or 8 pixels. Salt-and-pepper noise with an amplitude of 0.1 is added. The final image-per-symbol has a fixed height of 12 pixels and a varying width. The images are concatenated one after the other (no extra blank space is introduced); they appear in the script as the result of choosing the blank space character. The resulting "script video sequence" is fed to the network one vertical line at a time step through 12 input units. The testing data are more challenging than the training data in the sense that switches between curly bracket levels occur more rarely, which means that the WM must maintain the current bracket level for longer periods. More precisely, in our training sequence this period

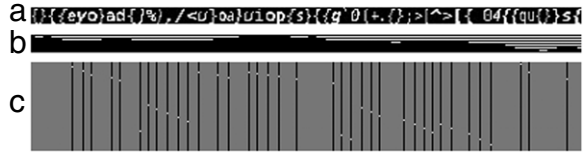

Fig. 4. Fragment of data used during training. (a) shows the input data. (b) represents WM-units' target. Each unit is shown on a different line by using black for −0.5 (off state) and white for 0.5 (on state). A black line is inserted between any two lines corresponding to WM-units' targets. (c) shows the target for the output units. The target is defined only when switching between characters (the undefined regions are plotted as gray). At the points where the target is defined, only the unit representing the next character is at 1 (plotted as white), while the others are at 0.

ranged between 0 to 248 cycle updates, with a mean of 17.7, while in testing data the period ranges between 0 to 691 cycles with a mean of 96.8. Note that while the average amount of time spent in a state during training is well within the reach of the innate, "fading" short-term memory of the reservoir (Jaeger, 2002), this is not the case for the testing data.

Architecture detail. The model used has 13 input units, with 12 representing a line of the data, while the last unit feeds a constant bias of −0.5. The input to reservoir connections are sparse, 80% of them being 0. The rest are randomly chosen to be either −0.5 or 0.5 with equal probability. The reservoir has 1200 units, with only 12,000 non-zero connections. The non-zero weights are either −0.1540 or +0.1540 with equal probability. The reservoir weight matrix has a spectral radius of 0.5. The activation function of the reservoir's units is tanh. The number of units as well as the spectral radius was chosen so as to maximize the performance of the model. We remark that these values are a compromise between the requirements of the different subprocesses that go on simultaneously in the network. The task that the network is asked to solve requires the network to recognize characters, to memorize the number of unclosed curly brackets, and to be able to do basic arithmetic operations.

Six WM-units are connected to the reservoir. The feedback weight matrix from the WM-units units to the reservoir is dense, all weights being randomly picked to be either −.4 or +.4. The value k of the counter is represented in the WM-units by having the first k units at +0.5, while the rest are at −0.5. The value 0 means that all WM-units are at −0.5. The network is able to represent the numbers 0, 1, 2, 3, 4, 5, and 6. The WM-units have the activation function described in Eq. (4). In addition to this, 65 ordinary output units are connected to the reservoir, with no feedback to the reservoir. The activation function for the output units is the identity function.

Training.

A teacher signal (target) for the six WM-units is generated for the training sequence, which represents the prescribed −0.5/0.5 switching as curly brackets pass by in the input. The target switches occur in the middle of the presentation of a bracket character (see Fig. 4). The training is done using the standard supervised offline learning that computes the WM-units' weights such that the distance between the WM-units and the target is minimized in the mean square error sense, as shown by (9).

Computing the weight matrix for the output units is done in a similar manner. We first generate the target signal for the 65 output units. At any given time step t, either the target is undefined if t occurs during a character presentation or it is 0 everywhere except for the unit corresponding to the next character where the value is 1. This is represented in Fig. 4 by using gray where the target is undefined. This allows us to use longer training sequences than those for the WM-units. To obtain the output weights, we collect only the states of the reservoir and the input at those steps when the target is defined. We then use Eq. (8) to compute

Table 1
Number of erroneous WM states obtained by the ESN, averaged over 30 runs.

Type of error	Number of errors	Percentage of curly brackets (%)	Percentage of characters (%)	Percentage of time steps (%)
False negatives	7.2 ± 6.5	0.34 ± 0.30	0.02 ± 0.018	0.003 ± 0.002
False positives	59.8 ± 21.6	2.84 ± 1.02	0.17 ± 0.061	0.024 ± 0.008
Total	67.0 ± 22.9	3.18 ± 1.09	0.19 ± 0.065	0.027 ± 0.009

Table 2
Trigger characters for false positives, averaged over 30 runs.

Character (number of characters in the testing sequences)	Number of times the counter increased	Number of times the counter decreased
"(" (499.5 ± 22.3)	21.5 ± 10.1	0 ± 0
")" (502.4 ± 18.6)	0 ± 0	0.5 ± 0.2
"[" (496.2 ± 22.8)	5.8 ± 5.1	0 ± 0
"]" (501.3 ± 15.1)	0.05 ± 0.03	6.0 ± 5.4
"@" (492.7 ± 21.3)	25.1 ± 14.1	0.2 ± 0.1
Other	0.05 ± 0.04	0.6 ± 0.5

the output weights that would minimize the mean square error between the outputs and the target.

Results. We run the experiment 30 times, each time with different randomly generated training and testing sequences, and freshly randomly generated reservoir, input, and feedback weights.

We start by inspecting the performance of the WM-units. An appropriate measure of the performance of the memory performance is the number of mistakes made. As a mistake, we consider events where the WM-unit state is different from that of the target. We do not check for mistakes during the presentation of a curly bracket (i.e., we do not evaluate transient effects within the time span of a curly bracket). Once a mistake is present, we count it and then correct the state of the network. To do so, we only correct the WM-units' state by externally forcing them for one time step to the desired configuration; the feedback connections will then also correct the reservoir's state. We sorted the errors into false positives (when the network detects a bracket character even though none is present in the input) and false negatives (when the network fails to detect a bracket). Table 1 lists the error counts.

On closer inspection of the errors produced in the 30 runs, we found that the network never changed the WM-units to an invalid (non-coding) state or by increasing or decreasing the counter by more than 1. This suggests that any error is actually the result of misclassification of a character, and not by the other subprocesses of WM-unit state management (adding/subtracting 1 or keeping a certain value stable).

Following up on this observation, we further differentiated the number of false positives according to which character triggered the error. Table 2 shows these results, which coincide with our intuition of when the recognition subtask might fail.

Another question one might raise is if correcting only the WM-units' state is sufficient for correcting the state of the network. If this were not the case, we would expect several errors to occur in rapid succession in a short time span (during the same character presentation or over two consecutive characters). Such errors would also suggest instability of WM-unit locking. But such scenarios never happen in any of the 30 runs.

We also measured the average absolute value of the computed $\mathbf{W}^{mem}$ weights, i.e., the weights leading to the WM-units (Table 3). Their modest size is indicative of a robust generalization, which indeed was observed, since the testing data were more challenging than the training data.

Table 3
Average learned output weights of the ESN (over 30 runs).

Considered weights	Average absolute value
Input to WM-units' weights	0.2327 ± 0.1813
Reservoir to WM-units' weights	0.0667 ± 0.0591
WM-units to WM-units' weights	0.5825 ± 0.5627

The "payload" task our architecture had to solve was to predict the next input character. In order to measure the performance of the network on this task, we considered as the predicted next character the most probable one (the one that corresponded to the output unit with the maximal score). The performance is quantified by simply counting the number of erroneous predictions. Note that the output units do not feed back to the reservoir and therefore a bad prediction will not affect any of the following predictions.

At any character switching step in between curly brackets, the next character k is selected according to a distribution that puts 80% weight on a single character. What we ask the network to do is to learn 65 such peaked conditional next-character distributions for each bracket level, in total 455 different distributions. Assuming that the network learns them perfectly, due to the deterministic approach of selecting the prediction of the next character, the network will always pick the character that has 80% weight, yielding an error rate of 20%, which is the best performance that we can expect to achieve on this task.

The error rate that we found on average over the 30 runs was $24.83 \pm 0.27\%$.

To demonstrate the importance of the WM-units in achieving this performance level, we ran the same experiments with an ESN that had no WM-units but was otherwise set up similarly. What we expect to happen is that the network will not able to distinguish between bracket levels anymore. Assuming that the current character is j, the network would learn in this case a distribution that gives a larger, almost equal, probability to $j + 1$ modulo 65, $j + 2$ modulo 65, $\ldots$, $j + 7$ modulo 65 (the most probable characters for the different bracket levels), and much smaller equal probability to all other characters. This implies that on average across the different bracketing levels (which cannot now be memorized for longer time spans) the network is likely to give wrong predictions in $20 + (6/7) * 80 \approx 80.5\%$ of the cases. We found an error rate of $83.75 \pm 0.11\%$ in this condition, close to what we expected.

4. Input-induced attractors

Now we proceed to characterize in a more rigorous way the dynamical phenomena that we just witnessed. Abstractly and intuitively speaking, the system from the previous section has the following characteristics.

1. It is a discrete-time dynamical system with stochastic input.
2. It could "lock" into distinctly different "dynamical configurations", each one associated with one of the possible WM states.
3. Each of these dynamical configurations is resistant to a large class of variation in the input (provided no further curly bracket comes along).

These "dynamical configurations" – a term that is just a working name which will soon be replaced by a rigorously defined concept – have an intuitive similarity with attractors, as familiar from dynamical systems theory. But the standard definitions of attractors refer to *autonomous* systems, i.e., systems without input. We will propose a generalization of the familiar notion of attractors in autonomous systems to attractors in systems with input.

A cautionary remark is in place. There are several subtly non-equivalent definitions of attractors used in dynamical systems theory. It is neither straightforward nor easy to arrive at a convincing definition of attractors even in the simpler case of autonomous

systems. We may expect a multiplication of conceptual subtleties when we turn to systems with input. Therefore we consider our proposal as preliminary.

Adapting Milnor (2006), we depart from the following classical definition of an attractor in an autonomous system.

Definition 1. Let $X \subset \mathbb{R}^m$ be a compact *phase space*, and $f : X \to X$ a map. A subset $A \subset X$ is *f-invariant* if $f(A) = A$. A *neighborhood* of A is a subset of X which contains A in its interior. We say that a neighborhood N of A is a *forward isolating neighborhood* of A if A is equal to the intersection of the forward images $f^n(N)$ of N. Then, A is an *attractor* iff

1. A is compact,
2. A is f-invariant,
3. A has a forward isolating neighborhood, and
4. no proper subset of A has the previous three properties.

Now let us consider a discrete-time dynamical system with input. Let again $X \subset \mathbb{R}^m$ be a compact phase space, $U \subset \mathbb{R}^k$ an *input space*, and $f : X \times U \to X$ a map. In the curly bracket counting system from the previous section, $X = [-1, 1]^N \times \{-0.5, 0.5\}^6$ is the set of possible reservoir-and-WM-unit states and $U = [0, 1]^{13}$. It is clear that a "dynamical configuration" switch of this system (i.e., a change in the WM-units) at some time step $n \to n + 1$ depends on both the reservoir state $\mathbf{x}(n)$ and the input $\mathbf{u}(n)$. In order to capture and analyze the interacting effects of current state and current input on "dynamical configuration" switches, we need a way to connect these two relevant items. The most straightforward method is to specify, for each state, a set of admissible inputs.

Definition 2. Let $X \subset \mathbb{R}^m$ be a compact phase space, $U \subset \mathbb{R}^k$ (where $k \geq 0$) an input space, and $f : X \times U \to X$ a map. Let $\gamma : X \to 2^U$ (where 2^U is the power set of U) be a map that assigns to every state a set of *associated* inputs. Then we call γ a *state-input-association* (SIA), and (X, U, f, γ) a γ-system.

Once an SIA is fixed, we can define the set of input-driven trajectories relative to the SIA.

Definition 3. Let X, U, f, and γ be as in the previous definition.

1. Let $f_\gamma : X \to 2^X$ be defined by

   $f_\gamma : \mathbf{x} \mapsto \{\mathbf{y} \in X \mid \exists \mathbf{u} \in \gamma(\mathbf{x}) \text{ such that } \mathbf{y} = f(\mathbf{x}, \mathbf{u})\}$.

   We call f_γ the *forward map induced by the SIA* γ, or simply the *γ-map*.
2. A *forward γ-orbit* is a right-infinite sequence $(\mathbf{x}_n)_{n=0,1,2,\ldots}$ of states, such that $\forall n : \mathbf{x}_{n+1} \in f_\gamma(\mathbf{x}_n)$.

Before we proceed to define attractors in γ-systems, it will be helpful to inspect some examples of such systems.

Example 1. *Autonomous systems as γ-systems.* If (X, g) is an ordinary autonomous system, i.e., $g : X \to X$, then using a "dummy" singleton input set $U = \{c\}$ and putting $f : X \times U \to X, ((\mathbf{x}), c) \mapsto g(\mathbf{x})$ and $\gamma(\mathbf{x}) = \{c\}$ for all $\mathbf{x} \in X$ yields a representation of the original autonomous system as a γ-system.

Example 2. *Systems driven by i.i.d. input.* If one wants to model a situation where the input has no temporal structure in the sense that, at every time step, any value $\mathbf{u} \in U$ is possible regardless of history, the natural choice for γ is $\gamma(\mathbf{x}) = U$ for all $\mathbf{x} \in X$.

Example 3. *A simple RNN driven by deterministic chaotic input.* This example can be regarded as a baby version of an RNN working memory, simplified as much as possible in order to obtain instructive visualizations. Consider an ESN whose reservoir consists of just two tanh sigmoid neurons, which receive a scalar input signal, and which has a single output neuron with feedback to the reservoir (the analog of a memory unit). Formally, this system is governed by

$$\mathbf{x}(n+1) = \tanh(\mathbf{W}\,\mathbf{x}(n) + \mathbf{W}^{in}u(n+1) + \mathbf{W}^{fb}y(n)),$$
$$y(n+1) = \tanh(\mathbf{W}^{out}\mathbf{x}(n+1)),$$

where $\mathbf{x} \in (-1, 1) \times (-1, 1), u \in (0, 1) = U, y \in (-1, 1)$ are the reservoir state, the input, and the output unit states. Note that, from a formal perspective, this is a three-dimensional input-driven system with system states $(x_1(n), x_2(n), y(n)) \in (-1, 1)^3 = X$, and a map $f : X \times U \to X$. To make this example concrete, we put

$$\mathbf{W} = \begin{pmatrix} 1/2 & -1 \\ 1 & 0 \end{pmatrix}, \qquad \mathbf{W}^{in} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \qquad \mathbf{W}^{fb} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$
$$\mathbf{W}^{out} = (-1, 1).$$

We generate scalar input sequences from the iterated logistic map in its chaotic regime by

$$u(n+1) = 4 \cdot u(n) \cdot (1 - u(n)) =: g(u(n)).$$

This system can lock in two "memory states" depending on the value of the output unit, which, when negative, will remain negative regardless of the input, and when positive, will remain positive. Fig. 5(e) shows the state/input tuples $(x_1(n), x_2(n), y(n), u(n+1))$ obtained from two long runs of this system, where one run has the output unit taking only positive, and the other run only negative values (some initial washout points have been discarded from this plot). The y-coordinate of system states is not plotted directly; only its sign is reflected in the coloring of the $x_1(n), x_2(n)$ components plotted at the bottom plane of each diagram. It can be seen (rather, guessed) from this diagram that (after washing out initial transients) the set of occurring $(x_1(n), x_2(n), y(n))$ system states is a fractal, and that every state (x_1, x_2, y) from such an asymptotic state set is paired with exactly one next input u.

In order to start a mathematical analysis of the state–input pairings shown in Fig. 5(e), and the input-driven attractor dynamics which, intuitively speaking, is visible here, one has to define an SIA that is adapted to this situation. More generally speaking, defining an SIA is the first step in a formal analysis, and it depends on the goals of the analysis how the SIA is set up. Here we want to analyze how trajectories of our baby system are pulled to one of the two asymptotic behaviors visible in Fig. 5(e), starting from any randomly chosen initial state $(x_1(0), x_2(0), y(0))$ and input $u(1)$. The natural way to proceed, for these purposes, is to define γ inductively, as follows. Define a descending sequence $X_0 \supset X_1 \supset X_2 \cdots$ of state sets, together with a sequence of input associations $\gamma_0, \gamma_1, \gamma_2, \ldots$, where $\gamma_i : X_i \to 2^U$, by putting

$$X_0 = X; \quad \forall \mathbf{x} \in X_0 : \gamma_0(\mathbf{x}) = U; \tag{10}$$

$$X_{i+1} = \{\mathbf{z} \in X \mid \exists \mathbf{x} \in X_i\ \exists u \in \gamma_i(\mathbf{x}) : \mathbf{z} = f(\mathbf{x}, u)\}; \tag{11}$$

$$\gamma_{i+1}(\mathbf{z}) = \{v \in U \mid \exists \mathbf{x} \in X_i\ \exists u \in \gamma_i(\mathbf{x}) : \mathbf{z} = f(\mathbf{x}, u) \wedge v = g(u)\}. \tag{12}$$

Then we set, for any $\mathbf{x} \in X$,

$$\gamma(\mathbf{x}) = \bigcap_{\{i \in \mathbb{N} \mid \mathbf{x} \in X_i\}} \gamma_i(\mathbf{x}).$$

Panels (a)–(d) in Fig. 5 show X_i and γ_i for $i = 0$, 1, 2, and 4. It should be noted that the SIA constructed here contains exactly those state–input pairs $(\mathbf{x}, u)$ which can ever arise when our baby system is started from random initial conditions. It can be seen that $(X_i)_{i=1,2,\ldots}$ converges to the (topological closure of the) carrier of the two asymptotic trajectories that we obtain from two infinite runs of the system, as depicted in panel (e). It is also clear that, with this construction of γ, all forward γ-orbits will converge to

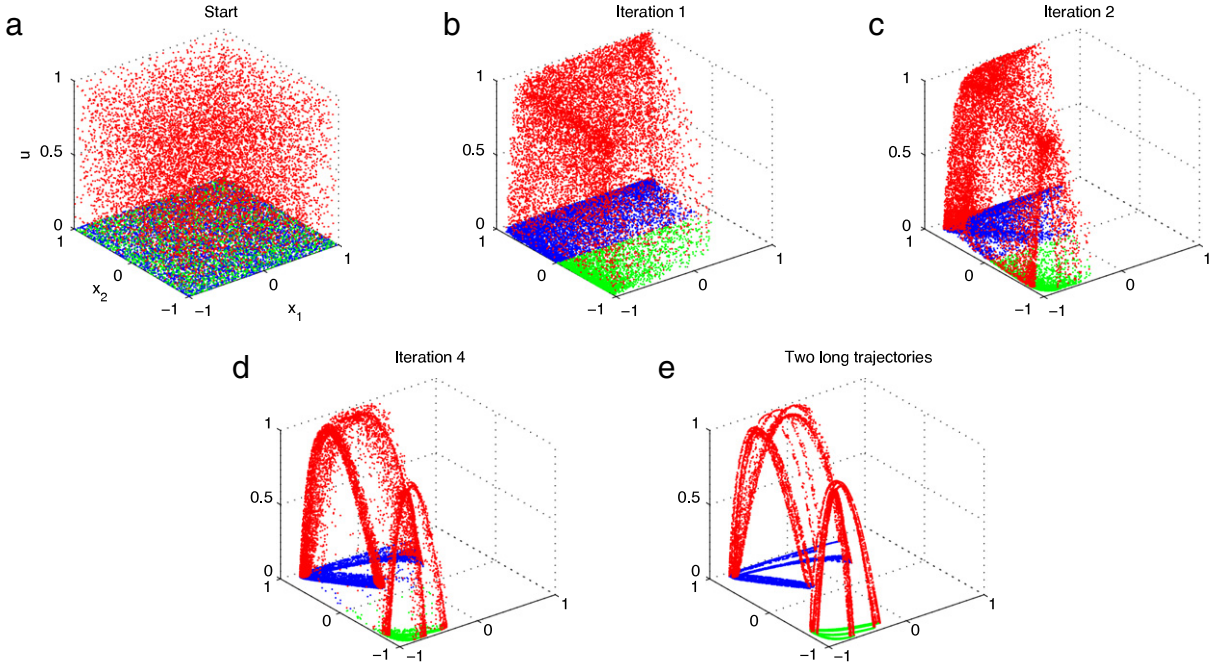

Fig. 5. Demonstration of SIAs in the "baby" ESN model. (a)–(d) show the first iterations of constructing an SIA γ, while (e) shows two separate trajectories. On the (x_1, x_2)-plane, the reservoir states are shown in blue or green, indicating whether the corresponding state of the output unit y is positive (blue) or negative (green). The red dots show $(x_1(n), x_2(n), u(n+1))$ triples; that is, $\gamma(x_1(n), x_2(n), y(n))$ contains the u values appearing above the $(x_1(n), x_2(n))$ plot points. 40,000 points are plotted per diagram. For details, see the text. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

one of the asymptotic sets shown in panel (e). (We state all these observations informally and without proof, as aids for intuition. A rigorous analytical workout of the material indicated in this section is planned for a separate publication.)

Equipped with SIAs, we can now make our intuitions about attractor-like "dynamical configurations" precise, by extending Definition 1 to systems with input.

Definition 4. Let (X, U, f, γ) be a γ-system. We define the following.

1. Let $A \subset X$. The *f_γ-forward image* of A is the set
$$f_\gamma(A) = \bigcup_{a \in A} f_\gamma(a).$$
2. A subset $A \subset X$ is *f_γ-invariant* iff $A = f_\gamma(A)$.
3. A neighborhood N of $A \subset X$ is a *forward f_γ-isolating neighborhood* of A if A is equal to the intersection of the forward images $f_\gamma^n(N)$ of N.
4. $A \subset X$ is a *γ-attractor* if
   (a) A is compact,
   (b) A is f_γ-invariant,
   (c) A has a forward f_γ-isolating neighborhood, and
   (d) there exists no proper subset of A having the previous three properties.

Note that, if $\gamma \equiv \{\mathbf{u}_0\}$ (constant input), this definition reduces to Definition 1.

The concept of γ-attractors enables us to formally describe some aspects of the "dynamical configurations" associated with the locking states of the WM-units, in terms of attractors. A somewhat simplified analysis might run as follows. As phase space X we take the combined reservoir-and-WM-unit state space. Notice that $X = [-1, 1]^N \times \{-0.5, 0.5\}^6$, equipped with the product topology of the Euclidean metric topology of $[-1, 1]^N$ with the discrete topology of $\{-0.5, 0.5\}^6$. The input space U is the 13-dimensional pixel vector space $[0, 1]^{13}$. f is the input-dependent update function for reservoir-plus-WM-units states, assembled from Eqs. (1), (2) and (4).

In order to obtain an SIA analog to the one from Example 3 above, one would ideally proceed as follows:

- For each of the seven WM-unit configurations that code bracketing levels, we compute a separate preliminary SIA γ^k ($k = 1, \ldots, 7$), as follows.
  - Clamp the WM units in the corresponding kth coding configuration;
  - consider a right-infinite input signal $\mathbf{u}(n)$ corresponding in its Markov chain properties to the current bracketing level k;
  - call the closure of the set of all inputs in this sequence U^k;
  - as an analog to (10), put $X_0^k = X^k$ (where $X^k = [-1, 1]^N \times (y_1, \ldots, y_6)$, with $(y_1, \ldots, y_6)$ fixed to the current WM configuration), and put $\gamma_0^k(\mathbf{x}) = \{\mathbf{u} \in U^k \mid f(\mathbf{x}, \mathbf{u}) \in X^k\}$;
  - as an analog to (11), put $X_{i+1}^k = \{\mathbf{z} \in X^k \mid \exists \mathbf{x} \in X_i^k \; \exists \mathbf{u}(n) \in \gamma_i^k(\mathbf{x}) : \mathbf{z} = f(\mathbf{x}, \mathbf{u}(n))\}$;
  - as an analog to (12), put $\gamma_{i+1}^k = \{\mathbf{v} \in U^k \mid \exists \mathbf{x} \in X_i^k \; \exists \mathbf{u}(n) \in \gamma_i^k(\mathbf{x}) : \mathbf{z} = f(\mathbf{x}, \mathbf{u}(n)) \wedge \mathbf{z} \in X^k \wedge \mathbf{v} = \mathbf{u}(n+1)\}$;
  - finally, for any $\mathbf{x} \in X^k$, put $\gamma^k(\mathbf{x}) = \bigcap_{\{i \in \mathbb{N} \mid \mathbf{x} \in X_i^k\}} \gamma_i^k(\mathbf{x})$.
- Notice that γ^k is defined on X^k, which is a proper subset of X. Thus, w.r.t. X, the γ^k are partially defined functions. Assuming that in runs of the WM system no other than the seven intended coding configurations of WM-units occur, it holds that $X = \bigcup X^k$. In order to obtain the desired totally defined SIA $\gamma : X \to 2^U$, join the γ^k.

For practical investigations, one has to resort to some feasible sampling strategy which approximates this theoretical construction.

The SIA γ thus obtained is a basis for analyzing γ-attractors that arise under the assumptions that

- no other than the bracket-level coding configurations for the WM-units do occur, and

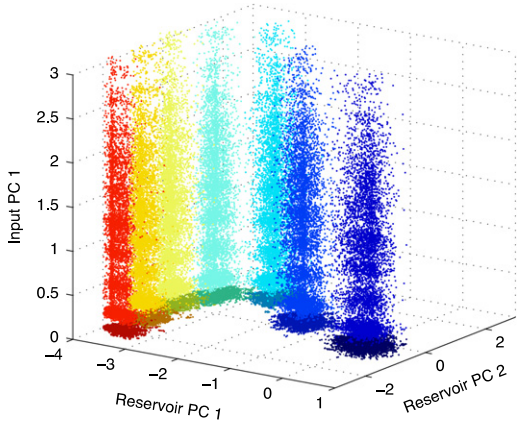

Fig. 6. An analog of Fig. 5(e) for the WM model system. The first PC of input signals is plotted against the first two PCs of reservoir states, for the seven WM unit configurations described in the previous section. Different colors correspond to different WM configurations. Projections of the reservoir state PCs are shown in darker shading on the ground plane. 6000 points are plotted per attractor. Notice that the value ranges no longer correspond to the $(-1, 1)$ range of tanh reservoirs because we display projections on the PCs. For details, see the text.

- when the system is in a WM-unit configuration coding for the kth bracketing level, and input is given which contains no curly brackets and corresponds in its Markov chain properties to that level, that WM-unit configuration is preserved by the system state updates.

Given these premises, one can then study on the basis of our γ how the reservoir state converges to different γ-attractors, when the system is started with the WM-units set to one of the seven admissible configurations and otherwise arbitrary reservoir states, and receiving input that preserves the WM-unit configuration.

For illustration, we give an analog of Fig. 5(e). It was obtained as follows.

- We ran the WM model from the previous section seven times for approximately 45,000 network updates, each time with the memory units clamped in one of the seven settings of the coding for one bracketing level; the driving input was in each case generated from an input character sequence whose Markov chain properties were the same as used in the previous section, not containing curly brackets;
- the seven sets of reservoir states and input vectors obtained were concatenated and the first principal components (PCs) of the reservoir states and inputs were computed;
- separately for each of the seven datasets, the first PC of the inputs was plotted against the first two PCs of the reservoir states.

Fig. 6 shows the plot obtained. One sees that even in only the first two reservoir PCs, the reservoir state sets corresponding to the different memory configurations become very well separated.

We have only started to work out the mathematical theory of γ-attractors. Therefore, this sketch can serve only as a demonstration of the conceptual usefulness of these concepts. Our formal research aims at rigorous characterizations of the interplay between input, critical state sets in which particular inputs have attractor-switching effects, sizes of basins, types of attractors (point, cyclic, strange), and more.

To our knowledge, attractors in systems with input have not previously been investigated, at least not in the spirit of generalizing the classical concept of attractors as subsets of phase space in autonomous systems. There are however some related strands of research that we want to point out. To see the connections, we first remark that, when the SIA γ is fixed, one can drop the reference to inputs from the model and consider maps $f : X \to 2^X$ per se. Such maps, and attractors arising from them, have been analyzed in several contexts.

Random dynamical systems (RDSs). This is an important subfield of ergodic theory. A characteristic of RDSs is that the possible transitions from some state x to successor states are governed by a probability distribution. In the framework of RDSs, these distributions are specified in terms of a family of maps $(\varphi(\omega))_{\omega\in\Omega}$, where Ω is the domain of an underlying probability space $(\Omega, \mathcal{F}, P)$, and, for each ω, $\varphi(\omega) : X \to X$ is a map on X (here we consider only iterated function system RDSs for reasons of expository simplicity; the theory of RDSs is usually expressed more generally for continuous-time systems). The distribution on the successors y of x is then given by $P_\varphi(Y) = P(\{\omega \mid \varphi(\omega)x \in Y\})$. This setting leads to a concept of attractors as *random sets* A, i.e., they are objects of the type $A : \Omega \to 2^X$. Attractors in such systems can be defined in several ways; see Crauel and Flandoli (1994) for a particular instantiation and pointers to alternatives. In Kwiecinska and Slomczynski (2000), a generalization of the concept is presented in which the family of maps $(\varphi(\omega))_{\omega\in\Omega}$ is governed by mixture distributions which vary with x. This brings the RDS approach closer to our Definition 4, where likewise the family of next-state functions changes with x. A fundamental difference between RDSs and our proposal is the probabilistic nature of an RDS versus the non-deterministic (but not probabilistic) character of input-induced attractors.

Relational state transition dynamics. (Manca, Franco, & Scollo, 2005; Scollo, Franco, & Manca, 2007) is an approach developed in a context of biomolecular dynamics which, like our suggested framework, considers non-deterministic iterated maps $f : X \to 2^X$, and attractors in such systems which are framed as subsets $A \subset X$, like in our proposal. The intended use of the theory is in symbolic dynamics and formal languages, and thus the phase space X is assumed to be a countable set (of words).

Non-deterministic systems generated by open maps and their attractors have been studied in detail in Duarte and Torres (2006). In this approach, iterated non-deterministic maps $f : X \to 2^X$ are required to have, for every $x \in X$, an image $f(x)$ which is open and connected. From this topological starting point, a theory is developed which allows to give conditions on f that ensure the existence of (only) finitely many attractors, and whose basins of attraction almost cover X. This work is performed in the tradition of "pure" mathematical dynamical systems theory and focusses on questions of structural stability.

While probabilistic RDSs have been widely investigated, the study of non-deterministic iterated maps appears to be a quite recent phenomenon, with a sparse "point support" of a few approaches in specialized communities, each relying on different restrictions on the underlying phase spaces and maps. It is likely that we did not detect all such approaches. We see our proposal as one further member in this small and scattered family, with the distinctive characteristic that we obtain our $f : X \to 2^X$ from a single underlying dynamics with input, via SIAs. We see this as an appropriate starting point for an analysis of attractors in input-driven systems, which – finally we close the circle of our argument – is the basic abstract mechanism of working memory.

5. Discussion

This paper showed how ESN-based architectures can be set up and trained to react to specific cues contained in a noisy and highly variable input signal, transforming them into specifically organized, stable memory traces. The network reliably exhibits

such behavior, and greatly exploits information stored in the memory states for ongoing "payload" processing (here, prediction) of the input stream. We see this architecture as a step toward practically useful RNN-based speech and handwriting recognition systems, which in our view hold principled but as yet unfathomed promises with respect to robustness and speed of recognition.

The presented model implements memory items by a dynamical phenomenon that is intuitively akin to attractors. However, due to the presence of input, this dynamical phenomenon cannot be understood as a classical (Milnor) attractor. We presented a coarse survey of known, generalized attractor concepts. None of these completely fits the situation at hand. We proposed yet another formal definition of attractors, γ-attractors, tailored to systems driven by essentially arbitrary and unmodelled (i.e., non-deterministic) input. Ongoing research in the group of the second author is concerned with a rigorous formal working-out of this emerging mathematical picture.

Each γ-attractor in our WM model represents one memorizable concept/category — one of seven bracketing levels. While we are in venerable company here (the classical Hopfield networks likewise identify a given, finite set of memorizable categories with point attractors, though as a long-term memory model), we think it would be more elegant, intellectually satisfactory, useful, and cognitively adequate if we had an WM model in which essentially *arbitrary* content items can be stored. In the cognitive sciences, an archetype of such models is Baddeley's WM model (see the review by Baddeley (2003)), which in its *phonological loop* component can store arbitrary phonological sequences, subject only to length constraints. In Jaeger and Eck (2008), we presented an ESN-based model with the functionality of such a phonological loop. More specifically, the *cue-addressable periodic attractor system* (CAPAS) presented there lets an ESN lock in to any binary-coded periodic signal (up to a maximum duration) after two presentations (the minimum needed for identification) and stably repeat it. The system can store only a single such sequence at a time, and it is not an input-induced, but classical, global, autonomous (Milnor) attractor, which blocks the CAPAS system for simultaneous input processing. As a next step on our way toward RNN-based speech recognition, we are investigating methods to combine the CAPAS mechanism of storing arbitrary sequential information *in* the attractor with the mechanisms suggested in this contribution, using input-induced attractors to store several arbitrary items simultaneously, with ongoing other processing.

Acknowledgements

The work reported in this paper was partially supported through the EU FP7 project ORGANIC (www.reservoir-computing.org).

References

Babloyantz, A., & Lourenço, C. (1994). Computation with chaos: a paradigm for cortical activity. *Proceedings of the National Academy of Sciences of the USA*, *91*, 9027–9031.

Baddeley, A. (2003). Working memory: looking back and looking forward. *Nature Reviews: Neuroscience*, *4*(10), 829–839.

Buehner, M., & Young, P. (2006). A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, *17*(3), 820–824.

Crauel, H., & Flandoli, F. (1994). Attractors for random dynamical systems. *Probability Theory and Related Fields*, *100*, 365–393.

Dominey, P. F., Arbib, M. A., & Joseph, J. P. (1995). A model of cortico-striatal plasticity for learning oculomotor associations and sequences. *Cognitive Neuroscience Journal*, *7*(3), 311–336.

Duarte, P., & Torres, M. J. (2006). Combinatorial stability of non-deterministic systems. *Ergodic Theory and Dynamical Systems*, *26*, 93–128.

Durstewitz, D., Seamans, J. K., & Sejnowski, T. J. (2000). Neurocomputational models of working memory. *Nature Neuroscience*, *3*, 1184–1191.

Freeman, W. J. (2007a). Definitions of state variables and state space for brain-computer interface. part 1: multiple hierarchical levels of brain function. *Cognitive Neurodynamics*, *1*(1), 3–14.

Freeman, W. J. (2007b). Definitions of state variables and state space for brain-computer interface. part 2. extraction and classification of feature vectors. *Cognitive Neurodynamics*, *1*(2), 85–96.

Gros, C. (2009). Cognitive computation with autonomously active neural networks: an emerging field. *Cognitive Computation*, *1*, 77–90.

Howard, M. (2009). Memory: computational models. In Larry R. Squire (Ed.), *Encyclopedia of neuroscience* (pp. 771–777). Oxford: Academic Press, ISBN: 978-0-08-045046-9, doi:10.1016/B978-008045046-9.00754-3.

Jaeger, H. (1995). Identification of behaviors in an agent's phase space. Arbeitspapiere der GMD 951, GMD, St. Augustin.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. *GMD report 148*. GMD — German national research institute for computer science. http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf.

Jaeger, H. (2002). Short term memory in echo state networks. *GMD-report 152*. GMD — German national research institute for computer science. http://www.faculty.jacobs-university.de/hjaeger/pubs/STMEchoStatesTechRep.pdf.

Jaeger, H. (2007). Echo state network. In *Scholarpedia Vol. 2* (p. 2330). http://www.scholarpedia.org/article/Echo_State_Network.

Jaeger, H., & Eck, D. (2008). Can't get you out of my head: a connectionist model of cyclic rehearsal. In I. Wachsmuth, & G. Knoblich (Eds.), *Lecture notes in artificial intelligence*: *Vol. 4930. Modeling communication for robots and virtual humans* (pp. 310–335). Springer Verlag.

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80.

Jaeger, H., Maass, W., & Principe, J. (2007). Special issue on echo state networks and liquid state machines. *Neural Networks*, *20*(3), 287–289. doi:10.1016/j.neunet.2007.04.001.

Kwiecinska, A. A., & Slomczynski, W. (2000). Random dynamical systems arising from iterated function systems with place-dependent probabilities. *Statistics & Probability Letters*, *50*, 401–407.

Lukosevicius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *4*(3), 127–149.

Lukosevicius, M., Jaeger, H., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state network with leaky integrator neurons. *Neural Networks*, *20*(3), 335–352.

Maass, W., Joshi, P., & Sontag, E. (2007). Computational aspects of feedback in neural circuits. *PLOS Computational Biology*, *3*(1), 1–20.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560. http://www.cis.tugraz.at/igi/maass/psfiles/LSM-v106.pdf.

Manca, V., Franco, G., & Scollo, G. (2005). State transition dynamics: basic concepts and molecular computing perspectives. In M. Gheorghe (Ed.), *Molecular computational models: unconventional approaches* (pp. 32–55). Idea Group Inc., Ch.2.

Milnor, J. W. (2006). Attractor. *Scholarpedia*, *1*(11), 1815.

Negrello, M., & Pasemann, F. (2008). Attractor landscapes and active tracking: the neurodynamics of embodied action. *Adaptive Behaviour*, *16*, 196–216.

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

Radicchi, F., & Meyer-Ortmanns, H. (2006). Entrainment of coupled oscillators on regular networks by pacemakers. *Physical Review E*, *73*, 036218 [7 pages]. http://arxiv.org/abs/cond-mat/0603346v1.

Schöner, G., Dose, M., & Engels, C. (1995). Dynamics of behavior: theory and applications for autonomous robot architectures. *Robotics and Autonomous Systems*, *16*(2), 213–246.

Scollo, G., Franco, G., & Manca, V. (2007). Relational state transition dynamics. *The Journal of Logic and Algebraic Programming*, *76*, 130–144.

Stollenwerk, N., & Pasemann, F. (1996). Control strategies for chaotic neuromodules. *International Journal of Bifurcation and Chaos*, *6*(4), 693–703.

Sussillo, D., & Abbott, L. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, *63*, 544–557.

Timme, M., Wolf, F., & Geisel, T. (2002). Prevalence of unstable attractors in networks of pulse-coupled oscillators. *Physical Review Letters*, *89*(15), 154105.

Tsuda, I. (2001). Towards an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behavioural and Brain Sciences*, *24*(5), 793–847.

Verstraeten, D., Schrauwen, B., & Stroobandt, D. (2006). Reservoir-based techniques for speech recognition. In *Proceedings of the world conference on computational intelligence* (pp. 1050–1053).

Yao, Y., & Freeman, W. (1990). A model of biological pattern recognition with spatially chaotic dynamics. *Neural Networks*, *3*(2), 153–170.

Zegers, P., & Sundareshan, M. K. (2003). Trajectory generation and modulation using dynamic neural networks. *IEEE Transactions on Neural Networks*, *14*(3), 520–533.