The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'

Article · January 2001

CITATIONS
189

READS
991

1 author:

Herbert Jaeger
Jacobs University
86 PUBLICATIONS **4,697** CITATIONS

SEE PROFILE

The "echo state" approach to analysing and training recurrent neural networks – with an Erratum note[1]

Herbert Jaeger
Fraunhofer Institute for Autonomous Intelligent Systems

January 26, 2010

[1]This is a corrected version of the technical report *H. Jaeger(2001): The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001.* In this report, one part of Definition 3 was flawed, with repercussions in Proposition 1. This error was pointed out to me by Tobias Strauss. It his corrected in this version. For historical veracity, the original techreport wasn't altered except that margin notes point to the flaws. The corrections are given in a new Erratum section which is added as an Appendix.

Abstract. The report introduces a constructive learning algorithm for recurrent neural networks, which modifies only the weights to output units in order to achieve the learning task.
key words: recurrent neural networks, supervised learning

Zusammenfassung. Der Report führt ein konstruktives Lernverfahren für rekurrente neuronale Netze ein, welches zum Erreichen des Lernzieles lediglich die Gewichte der zu den Ausgabeneuronen führenden Verbindungen modifiziert.
Stichwörter: rekurrente neuronale Netze, überwachtes Lernen

1 Introduction

Under certain conditions, the activation state $\mathbf{x}(n)$ of a recurrent neural network (RNN) is a function of the input history $\mathbf{u}(n), \mathbf{u}(n-1), \ldots$ presented to the network, i.e., there exists a function $\mathbf{E}$ such that $\mathbf{x}(n) = \mathbf{E}(\mathbf{u}(n), \mathbf{u}(n-1), \ldots)$. Metaphorically, the state $\mathbf{x}(n)$ can be understood as an "echo" of the input history.

This article investigates what can be gained when RNN states are understood as echo states. Specifically, the article discusses under which conditions echo states arise and describes how RNNs can be trained, exploiting echo states.

The perspective taken is mainly that of mathematics and engineering, where a recurrent network is seen as a computational device for realizing a dynamical system. The RNNs considered here are mostly discrete-time, sigmoid-unit networks. The basic idea of echo states has been independently investigated in [11] in a complementary fashion. In that work, the emphasis is on biological modeling and continuous-time (spiking) networks.

The article is organized as follows.

The second section defines echo states and provides several equivalent characterizations.

The third section explains how echo state networks can be trained in a supervised way. The natural approach here is to adapt only the weights of network-to-output connections. Essentially, this trains readout functions which transform the echo state into the desired output signal. Technically, this amounts to a linear regression task.

The fourth section gives two complementary, basic examples. Echo state networks are trained to generate periodic sequences (mathematically: an m-point attractor), and to function as multiple switches (mathematically: m point attractors).

The fifth section describes echo state networks with leaky integrator neurons.

The sixth section demonstrates how leaky integrator echo state networks master the quasi-benchmark task of learning the Mackey-Glass chaotic attractor.

Section 7 concludes with a comparative discussion and points out open problems.

2 Echo states

First we fix our terminology. We consider discrete-time neural networks with K input units, N internal network units and L output units. Activations of input units at time step n are $\mathbf{u}(n) = (u_1(n), \ldots, u_K(n))$, of internal units are $\mathbf{x}(n) = (x_1(n), \ldots, x_N(n))$, and of output units $\mathbf{y}(n) = (y_1(n), \ldots, y_L(n))$. Real-valued connection weights are collected in a $N \times K$ weight matrix $\mathbf{W}^{\text{in}} = (w^{\text{in}}_{ij})$ for the input weights, in an $N \times N$ matrix $\mathbf{W} = (w_{ij})$ for the internal connections, in an $L \times (K + N + L)$ matrix $\mathbf{W}^{\text{out}} = (w^{\text{out}}_{ij})$ for the connections to the output units, and in a $N \times L$ matrix $\mathbf{W}^{\text{back}} = (w^{\text{back}}_{ij})$ for the connections that project back from the output to the internal units. Note that connections directly from the input to the output units and connections between output units are allowed. No further conditions on the network topology induced by the internal weights $\mathbf{W}$ are imposed (e.g., no layer structure). We will also not formally require, but generally intend that the internal connections $\mathbf{W}$ induce recurrent pathways between internal units. Without further mention, we will always assume real-valued inputs, weights, and activations. Figure 1 shows the basic network architecture considered here.

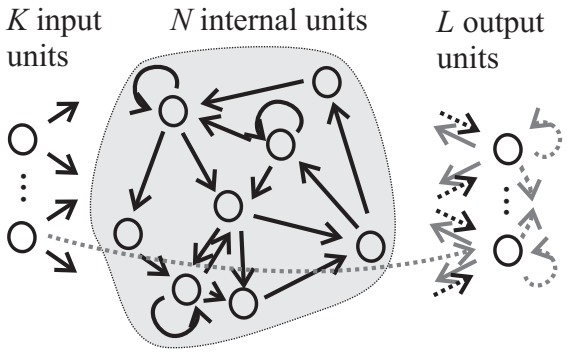

Figure 1: The basic network architecture assumed in this article. Dashed arrows indicate connections that are possible but not required.

The activation of internal units is updated according to

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{back}}\mathbf{y}(n)), \tag{1}$$

where $\mathbf{f} = (f_1, \ldots, f_N)$ are the internal unit's output functions (typically sigmoid functions). The output is computed according to

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

where $\mathbf{f}^{\text{out}} = (f_1^{\text{out}}, \ldots, f_L^{\text{out}})$ are the output unit's output functions and $(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n))$ is the concatenation of the input, internal, and previous output activation vectors.

In this paper we consider input sequences $(\mathbf{u}(n))_{n \in J} \in U^J$. We require that U be compact. We use shorthand $\bar{\mathbf{u}}^{\pm\infty}, \bar{\mathbf{u}}^{+\infty}, \bar{\mathbf{u}}^{-\infty}, \bar{\mathbf{u}}^h$ to denote input sequences which are left-right-infinite ($J = \mathbb{Z}$), right-infinite ($J = k, k+1, \ldots$ for some $k \in \mathbb{Z}$), left-infinite, and finite of length h, respectively. Similar shorthands are used for state and output sequences. Other than compactness we impose no restrictions on input sequences, i.e., every $(\mathbf{u}(n))_{n \in J} \in U^J$ is admissible. This implies shift-invariance of admissible inputs. We introduce a network state update operator T and write $\mathbf{x}(n+h) = T(\mathbf{x}(n), \mathbf{y}(n), \bar{\mathbf{u}}^h)$ to denote the network state that results from an iterated application of Eq. (1) if the input sequence $\mathbf{u}(n+1), \ldots, \mathbf{u}(n+h)$ is fed into the network which at time n is in state $\mathbf{x}(n)$ and had output $\mathbf{y}(n)$. In networks without output feedback, this simplifies to $\mathbf{x}(n+h) = T(\mathbf{x}(n), \bar{\mathbf{u}}^h)$.

Recurrent networks (as all nonlinear dynamical systems) may exhibit disjoint regions A, B of their state space in which network states stay contained regardless of input. We therefore discuss network dynamics always relative to a compact set $A \subset \mathbb{R}^N$ of *admissible* states. Among other things, this implies that if the network is started at some time n, the initial state $\mathbf{x}(n)$ must be in A. We require that A is closed under network update, i.e. for every $\mathbf{u} \in U, \mathbf{x} \in A$, it holds that $T(\mathbf{x}, \mathbf{u}) \in A$.

Thus, our analysis will always rely on the following generic setup: (i) input is drawn from a compact input space U; (ii) network states lie in a compact set A. We will call these conditions *standard compactness conditions*.

We now define echo state networks for the case of networks with no output feedback.

Definition 1 *Assume standard compactness conditions. Assume that the network has no output feedback connections. Then, the network has* echo states*, if the network state $\mathbf{x}(n)$ is uniquely determined by any left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$. More precisely, this means that for every input sequence $\ldots, \mathbf{u}(n-1), \mathbf{u}(n) \in U^{-\mathbb{N}}$, for all state sequences $\ldots, \mathbf{x}(n-1), \mathbf{x}(n)$ and $\mathbf{x}'(n-1), \mathbf{x}'(n) \in A^{-\mathbb{N}}$, where $\mathbf{x}(i) = T(\mathbf{x}(i-1), \mathbf{u}(i))$ and $\mathbf{x}'(i) = T(\mathbf{x}'(i-1), \mathbf{u}(i))$, it holds that $\mathbf{x}(n) = \mathbf{x}'(n)$.*

An equivalent way of stating the echo state property is to say that there exist *input echo functions* $\mathbf{E} = (e_1, \ldots, e_N)$, where $e_i : U^{-\mathbb{N}} \to \mathbb{R}$, such that for all left-infinite input histories $\ldots, \mathbf{u}(n-1), \mathbf{u}(n) \in U^{-\mathbb{N}}$ the current network state is

$$\mathbf{x}(n) = \mathbf{E}(\ldots, \mathbf{u}(n-1), \mathbf{u}(n)). \tag{3}$$

We now provide a number of equivalent characterizations of echo states. Definition 2 provides requisite terminology, definition 4 states the alternative characterizations.

Definition 2 (a) *A state sequence* $\bar{\mathbf{x}}^{-\infty} = \ldots, \mathbf{x}(n-1), \mathbf{x}(n) \in A^{-\mathbb{N}}$ *is called* compatible *with an input sequence* $\bar{\mathbf{u}}^{-\infty} = \ldots, \mathbf{u}(n-1), \mathbf{u}(n)$*, if* $\forall i < n : T(\mathbf{x}(i), \mathbf{u}(i+1)) = \mathbf{x}(i+1)$. (b) *Similarly, a left-right-infinite state sequence* $\bar{\mathbf{x}}^{\infty}$ *is called compatible with an input sequence* $\bar{\mathbf{u}}^{\infty}$*, if* $\forall i : T(\mathbf{x}(i), \mathbf{u}(i+1)) = \mathbf{x}(i+1)$. (c) *A network state* $\mathbf{x} \in A$ *is called* end-compatible *with an input sequence* $\bar{\mathbf{u}}^{-\infty}$ *if there exists a state sequence* $\ldots, \mathbf{x}(n-1), \mathbf{x}(n)$ *such that* $T(\mathbf{x}(i), \mathbf{u}(i+1)) = \mathbf{x}(i+1)$*, and* $\mathbf{x} = \mathbf{x}(n)$. (d) *A network state* $\mathbf{x} \in A$ *is called* end-compatible *with a finite input sequence* $\bar{\mathbf{u}}^{h}$ *if there exists a state sequence* $\mathbf{x}(n-h), \ldots, \mathbf{x}(n) \in A^{h+1}$ *such that* $T(\mathbf{x}(i), \mathbf{u}(i+1)) = \mathbf{x}(i+1)$*, and* $\mathbf{x} = \mathbf{x}(n)$.

Definition 3 *Assume standard compactness conditions and a network without output feedback.*

1. *The network is called* state contracting *if for all right-infinite input sequences* $\bar{\mathbf{u}}^{+\infty}$ *there exists a null sequence* $(\delta_h)_{h \geq 0}$ *such that for all states* $\mathbf{x}, \mathbf{x}' \in A$*, for all* $h \geq 0$*, for all input sequence prefixes* $\bar{\mathbf{u}}_h = \mathbf{u}(n), \ldots, \mathbf{u}(n+h)$ *it holds that* $d(T(\mathbf{x}, \bar{\mathbf{u}}_h), T(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$*, where* d *is the Euclidean distance on* $\mathbb{R}^N$*.*[1] EdNote(1)

2. *The network is called* state forgetting *if for all left-infinite input sequences* $\ldots, \mathbf{u}(n-1), \mathbf{u}(n) \in U^{-\mathbb{N}}$ *there exists a null sequence* $(\delta_h)_{h \geq 0}$ *such that for all states* $\mathbf{x}, \mathbf{x}' \in A$*, for all* $h \geq 0$*, for all input sequence suffixes* $\bar{\mathbf{u}}_h = \mathbf{u}(n-h), \ldots, \mathbf{u}(n)$ *it holds that* $d(T(\mathbf{x}, \bar{\mathbf{u}}_h), T(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$.

3. *The network is called* input forgetting *if for all left-infinite input sequences* $\bar{\mathbf{u}}^{-\infty}$ *there exists a null sequence* $(\delta_h)_{h \geq 0}$ *such that for all* $h \geq 0$*, for all input sequence suffixes* $\bar{\mathbf{u}}_h = \mathbf{u}(n-h), \ldots, \mathbf{u}(n)$*, for all left-infinite input sequences of the form* $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h, \bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$*, for all states* $\mathbf{x}$ *end-compatible with* $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ *and states* $\mathbf{x}'$ *end-compatible with* $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ *it holds that* $d(\mathbf{x}, \mathbf{x}') < \delta_h$.

Proposition 1 *Assume standard compactness conditions and a network without output feedback. Assume that* T *is continuous in state and input. Then the properties of being state contracting, state forgetting, and input forgetting are all equivalent to the network having echo states.*[2] EdNote(2)

[1] EdNote: The state contracting property is flawed – see Erratum note in Appendix E

[2] EdNote: See Erratum note in Appendix E.

The proof is given in the Appendix. Echo state networks have a desirable continuity property. Intuitively, the following proposition states that in order to know the current network output with a given precision, we only have to know the most recent inputs with a similar precision:

Proposition 2 *Assume standard compactness conditions, no output feedback, continuity of T in state and input, and echo states. Then it holds that for all left-infinite input sequences $\bar{\mathbf{u}}^{-\infty}$, for all $\varepsilon > 0$ there exist $\delta > 0$ and $h > 0$ such that $d(\mathbf{E}(\bar{\mathbf{u}}^{-\infty}), \mathbf{E}(\bar{\mathbf{u}}'^{-\infty})) < \varepsilon$ for all input sequences $\bar{\mathbf{u}}'^{-\infty}$ which satisfy $d(\mathbf{u}(k), \mathbf{u}'(k)) < \delta$ for all $-h \leq k \leq 0$.*

The proof is given in the Appendix. This continuity property was called "fading memory" in [11].

The conditions of Def. 4 are hard to check in practice. Unfortunately, I cannot offer conditions for echo states that are both easy to check and equivalent to echo states. The next proposition gives (a) a sufficient condition for echo states, which depends on a Lipschitz property of the weight matrix, and (b) a sufficient condition for the non-existence of echo states.

Proposition 3 *Assume a sigmoid network with unit output functions $f_i = \tanh$. (a) Let the weight matrix $\mathbf{W}$ satisfy $\sigma_{max} = \Lambda < 1$, where σ_{max} is its largest singular value. Then $d(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u})) < \Lambda\ d(\mathbf{x}, \mathbf{x}')$ for all inputs $\mathbf{u}$, for all states $\mathbf{x}, \mathbf{x}' \in [-1, 1]^N$. This implies echo states for all inputs $\mathbf{u}$, for all states $\mathbf{x}, \mathbf{x}' \in [-1, 1]^N$. (b) Let the weight matrix have a spectral radius $|\lambda_{max}| > 1$, where λ_{max} is an eigenvalue of $\mathbf{W}$ with the largest absolute value. Then the network has an asymptotically unstable null state. This implies that it has no echo states for any input set U containing $\mathbf{0}$ and admissible state set $A = [-1, 1]^N$.*

The proof is given in the Appendix. Both conditions are easy to check and mark the boundaries of an interesting scaling range for weight matrices, as follows. In practice, a convenient strategy to obtain useful echo state networks is to start with some weight matrix $\tilde{\mathbf{W}}$ and try out global scalings $\alpha\tilde{\mathbf{W}}$ until one is satisfied with the properties of some finally fixed weight matrix $\mathbf{W} = \alpha_{\text{opt}}\tilde{\mathbf{W}}$. Let $\sigma_{\max}(\mathbf{W})$ and $|\lambda_{\max}|\ (\mathbf{W})$ denote the largest singular value and the spectral radius of a matrix $\mathbf{W}$. Observe that the maximal singular value and the spectral radius of $\tilde{\mathbf{W}}$ scale with α, i.e. $\sigma_{\max}(\alpha\tilde{\mathbf{W}}) = \alpha\sigma_{\max}(\tilde{\mathbf{W}})$ and $|\lambda_{\max}|\ (\alpha\tilde{\mathbf{W}}) = \alpha\ |\lambda_{\max}|\ (\tilde{\mathbf{W}})$. Observe further that for every square matrix $\mathbf{W}$, $|\lambda_{\max}|\ (\mathbf{W}) \leq \sigma_{\max}(\mathbf{W})$. Thus, if one puts $\alpha_{\min} = 1/\sigma_{\max}(\tilde{\mathbf{W}})$ and $\alpha_{\max} = 1/\ |\lambda_{\max}|\ (\tilde{\mathbf{W}})$, one obtains a scaling range $\alpha_{\min} \leq \alpha \leq \alpha_{\max}$, where below the smallest scaling factor $\alpha_{\min}$ one would certainly have echo

states, and above $\alpha_{\max}$, certainly not. My (by now extensive) experience with this scaling game indicates that one obtains echo states even when α is only marginally smaller than $\alpha_{\max}$: the sufficient condition from Prop. 3(a) apparently is very restrictive.

Note also that if a network has a weight matrix with $\mid \lambda_{\max} \mid > 1$, the echo state property can hold nonetheless if input comes from a set U not containing the null input. For a demonstration of this fact, consider a single-unit network ($N = 1$) with a weight matrix $\mathbf{W} = (2)$. For null input, the network has an instable null state and two stable states $s, -s$, where s is defined by $\tanh(2s) = s$, so the network is clearly not echo state for an input set U containing $\mathbf{0}$ and admissible state set $A = [-1, 1]^N$. But if one restricts the input to sufficiently high values, e.g. to $U = [3, 4]$, a Lipschitz condition is met (exercise: use that $\tanh(x) < 1/2$ for $x \geq 1$).

More generally stated, non-echo-state networks sometimes can be turned into echo-state networks by suitable "tonic" components in the input. This might be an observation of interest for the analysis of biological neural networks, a matter that is outside the scope of this article.

3 How to compute with echo state networks, and how to train them

In this section we describe the intuitions about computation in echo state networks.

Generally speaking, RNNs are potentially powerful approximators of dynamics. There are many ways to make this statement more precise. For instance, RNNs can be casted as representations of the vector field of a differential system [9], or they can be trained to embody some desired attractor dynamics [5]. In those approaches, systems without input are modeled. Since we wish to deal with input-driven systems, we adopt a standard perspective of systems theory and view a (deterministic, discrete-time) dynamical system as a function $\mathbf{G}$ which yields the next system output, given the input and output history:

$$\mathbf{y}(n+1) = \mathbf{G}(\ldots, \mathbf{u}(n), \mathbf{u}(n+1); \ldots, \mathbf{y}(n-1), \mathbf{y}(n)). \tag{4}$$

In control engineering and signal processing, one typically uses rather restricted versions of Eq. 4; specifically, one employs finite-history approximations of Eq. 4; furthermore, one often assumes that $\mathbf{G}$ is linear in the $\mathbf{u}$ and/or $\mathbf{y}$ arguments. The most common approach to model such restricted systems is to approximate $\mathbf{G}$ by a suitably powerful feedforward network and

iteratively feed it with $\mathbf{u}(n-h), \ldots, \mathbf{u}(n); \mathbf{y}(n-h'), \ldots, \mathbf{y}(n-1)$ through a sliding window (e.g., [14]).

Echo state networks afford a direct way of modeling systems represented by the fully general Equation 4, without the necessity to convert a time series into static input patterns by sliding windows. For expository reasons, we start by considering the restricted case where the output depends only on the input history, i.e. systems without output feedback:

$$\mathbf{y}(n+1) = \mathbf{G}(\ldots, \mathbf{u}(n), \mathbf{u}(n+1)). \tag{5}$$

The remainder of this section is organized as follows. First we state the intuitions in informal terms, using a toy example for illustration (Subsection 3.1); then give a formal account of the restricted case (Subsection 3.2); and conclude with a formal description of the general case in the last Subsection.

3.1 The basic idea: informal statement and toy example

Assume that we have some echo state network with many units, and assume that the network's internal connections are rather inhomogeneous. Assume that a long input sequence is presented to the network. Due to the input forgetting property of echo state networks, after some initial transient the internal unit's activation can be be written as follows (with some liberty of notation)

$$x_i(n) \;\approx\; e_i(\ldots, \mathbf{u}(n), \mathbf{u}(n+1)), \tag{6}$$

where e_i is the echo function of the i-th unit. If the network is suitably inhomogeneous, the various echo functions will differ from each other. For an illustration of this fact, consider a single-channel sinusoid input $u(n) = \sin(2\pi n/P)$ with period length P. The echo state property implies that the activations $x_i(n)$ will also be periodic signals with the same period length; but the network's inhomogeneity will induce conspicuous deviations from the input sinusoid form. Figure 2 gives an example of a 100-unit echo state network. A period length of $P = 10\pi$ was used, i.e. the input signal was $u(n) = \sin(n/5)$. The first six traces show the signals $x_i(n)$ of some randomly selected units, the seventh trace shows the sinusoid input signal.

Now assume we want to train the network to produce a single-channel output, which is an interesting transformation of the input, for instance $y(n)_{\text{teach}} = 1/2 \sin^7(2\pi n/P)$. In terms of Eq. 5, this means we want to have $y_{\text{teach}}(n+1) = \mathbf{G}_{\text{teach}}(\ldots, u(n), u(n+1)) = \mathbf{G}_{\text{teach}}(n+1) = 1/2 u(n+1)^7$.

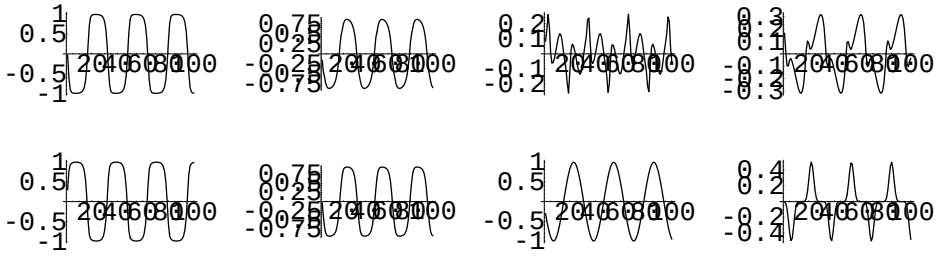

Figure 2: Traces of some selected units of a 100-unit echo state network driven by sinusoid input. The input signal is shown in the last but one trace, the output signal is shown in the last trace.

The idea is to combine $\mathbf{G}_{\text{teach}}$ from the echo functions e_i in a mean square error way. To this end, note that when there are no output feedbacks, the network output is given by the following single-channel version of Eq. 2:

$$y(n) = f^{\text{out}}(\sum_{i=1,\ldots,N} w_i^{\text{out}} x_i(n)), \tag{7}$$

where w_i^{out} is the weight of the i-th output connection. We use $f^{\text{out}} = \tanh$, which is invertible, therefore (7) is equivalent to

$$(f^{\text{out}})^{-1} y(n) = \sum_{i=1,\ldots,N} w_i^{\text{out}} x_i(n). \tag{8}$$

Inserting the echo functions e_i yields

$$(f^{\text{out}})^{-1} y(n) \;\; = \;\; \sum_{i=1,\ldots,N} w_i^{\text{out}} e_i(\ldots, u(n-1), u(n)). \tag{9}$$

Now we determine the weights w_i^{out} such that the error

$$\begin{aligned} \epsilon_{\text{train}}(n) &= (f^{\text{out}})^{-1} y_{\text{teach}}(n) - (f^{\text{out}})^{-1} y(n) \\ &= (f^{\text{out}})^{-1} y_{\text{teach}}(n) - \sum_{i=1,\ldots,N} w_i^{\text{out}} e_i(\ldots, u(n-1), u(n)) \end{aligned} \tag{10}$$

is minimized in the mean square error (MSE) sense, i.e. such that

$$\text{mse}_{\text{train}} = 1/(n_{\max} - n_{\min}) \sum_{i=n_{\min},\ldots,n_{\max}} \epsilon_{\text{train}}^2(n) \tag{11}$$

becomes minimal, where $n_{\min}$ refers to some data point of the training sequence after dismissal of an initial transient, and $n_{\max}$ is the last training point. We will refer to $\text{mse}_{\text{train}}$ as *training error*. Inspection of (10) reveals that minimizing (11) is a simple task of computing a linear regression.

Concretely, (i) we let the network run for $n = 0$ to $n_{\max} = 300$, starting from a zero network state, (ii) dismiss an initial transient of 100 steps after which the effects of the initial state have died out (state forgetting property), (iii) collect the network states $\mathbf{x}(n)$ for $n = n_{\min} = 101, \ldots, n_{\max} = 300$, and (iv) compute the weights w_i offline from these collected states, such that the error (10) becomes minimal.

There is another statement of this task which is somewhat imprecise but more intuitive. Rename $(f^{\text{out}})^{-1}\mathbf{G}_{\text{teach}}$ to $\mathbf{G}'$. Then, compute the weights such that

$$\mathbf{G}' \approx \sum w_i e_i \tag{12}$$

becomes a MSE approximation of $\mathbf{G}'$ by a weighted combination of the echo functions e_i.

Actually, instead of minimizing the error (10) one would like to minimize directly $\tilde{\epsilon}_{\text{train}} = \mathbf{G}(n) - y(n)$ in the MSE sense. However, our recipe minimizes ϵ_{train} instead. But since $\tilde{\epsilon}_{\text{train}} = \mathbf{G}(n) - y(n)$ is strictly related to ϵ_{train} by f^{out}, minimizing ϵ_{train} will also yield a good minimization of $\tilde{\epsilon}_{\text{train}} = \mathbf{G}(n) - y(n)$. (If we would use linear output functions $f^{\text{out}} = \text{id}$, we would have $\epsilon_{\text{train}} = \tilde{\epsilon}_{\text{train}} = \mathbf{G}(n) - y(n)$.)

In our sin-input, $\sin^7$-output example, with $f^{\text{out}} = \tanh$ we found a training error $\text{mse}_{\text{train}} \approx 3.3 \times 10^{-15}$. When the trained network was tested, a test error $\text{mse}_{\text{test}} = \langle (y_{\text{test}} - y)^2 \rangle \approx 3.7 \times 10^{-15}$ was obtained ($\langle \cdot, \cdot \rangle$ denotes expectation).

Two important points become apparent in this simple example:

1. The proposed learning procedure computes *only* the weights of connections leading to the output units; all other connections remain unchanged. This makes it possible to employ any of the many available fast, constructive linear regression algorithms for the training. No special, iterative gradient-descent procedure is needed.

2. In order to achive a good approximation $\mathbf{G}' \approx \sum w_i e_i$, the echo functions should provide a "rich" set of dynamics to combine from. The network should be prepared in a suitably "inhomogeneous" way to meet this demand. Metaphorically speaking, the echo state network should provide a rich "reservoir" of dynamics which is "tapped" by the output weights.

One simple method to prepare such a rich "reservoir" echo state network is to supply a network which is sparsely and randomly connected. Sparse connectivity provides for a relative decoupling of subnetworks, which encourages the development of individual dynamics. The 100-unit network used in the example was randomly connected; weights were set to values of 0, +0.4 and −0.4 with probabilities 0.95, 0.025, 0.025 respectively. This means a sparse connectivity of 5 %. The value of 0.4 for non-null weights resulted from a global scaling such that $|\lambda_{\max}| \approx 0.88 < 1$ was obtained.

The input weights were set in an offhand decision (without any optimization) to values of +1, -1 with equal probability.

Calculations were done with the Mathematica software package; the linear regression was done by calling Mathematica's `Fit` procedure.

3.2 Formal statement of restricted case

We now present a more rigorous formulation of the training procedure for the case with no output feedback.

Task. Given: a teacher I/O time series $(\mathbf{u}_{\text{teach}}(n), \mathbf{y}_{\text{teach}}(n))_{n=0,\ldots,n_{\max}}$, where the inputs come from a compact set U. Wanted: a RNN whose output $\mathbf{y}(n)$ approximates $\mathbf{y}_{\text{teach}}(n)$.

Procure an echo-state network. Construct a RNN that has echo states in an admissible state set A with input from U. A convenient heuristic to obtain such a network is by way of Proposition 3(b): in all experiments carried out so far, every standard sigmoid network whose weight matrix $\mathbf{W}$ satisfies $|\lambda_{\max}| < 1$ had echo states.

Choose input connection weights. Attach input units to the network. If the original network satisfied the Lipschitz condition of Prop. 3(a), input connections $\mathbf{W}^{\text{in}}$ can be freely chosen without harming the echo state property. Moreover, the experience accumulated so far indicates that the echo state property remains intact with arbitrarily chosen input connection weights, even if only the weaker condition $|\lambda_{\max}| < 1$ was ascertained in the previous step.

Run network with teacher input, dismiss initial transient. Start with an arbitrary network state $\mathbf{x}(0)$ and update the network with the training input for $n = 0, \ldots, n_{\max}$:

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{\text{in}}(\mathbf{u}_{\text{teach}}(n+1)) + \mathbf{W}\mathbf{x}(n)). \tag{13}$$

In agreement with the input forgetting property, choose an initial transient such that after the transient time $n_{\min}$ the internal network state is determined by the preceding input history up to a negligible error.

Compute output weights which minimize the training error. Let $\mathbf{y}_{\text{teach}}(n) = (y_{\text{teach},1}(n), \ldots, y_{\text{teach},L}(n))$ and put $g'_j(n) = (f_j^{\text{out}})^{-1} y_{\text{teach},j}(n)$. For each $j = 1, \ldots, L$ compute output weights w_{ji}^{out} such that the MSE

$$\text{mse}_{\text{train},j} = 1/(n_{\max} - n_{\min}) \sum_{n=n_{\min}}^{n_{\max}} \left(g'_j(n) - \sum_{i=1}^{N} w_{ji}^{\text{out}} x_i(n) \right)^2 \qquad (14)$$

is minimized. Use your favorite linear regression algorithm for this. With these output weights, the network is ready for use.

3.3 Formal statement of general case

In the terminology of signal processing, RNNs trained like above basically are a "nonlinear moving average" (NMA) kind of models. However, the output of many interesting dynamical systems depends not only on the input history, but also of the output history. The most general form of such "nonlinear autoregressive moving average" (NARMA) models is stated in Eq. 4. We now describe how the restricted training procedure from the previous subsection can be generalized accordingly.

The key idea is that from the "perspective" of internal units, fed-back output signals are just another kind of input. With this idea in mind, the general training procedure becomes completely analogous to the restricted case when the teacher output $\mathbf{y}_{\text{teach}}(n)$ is written into the output units during the training period (teacher forcing). We repeat the instructions from the previous subsection with the appropriate modifications.

Task. Given: a teacher I/O time series $(\mathbf{u}_{\text{teach}}(n), \mathbf{y}_{\text{teach}}(n))_{n=0,\ldots,n_{\max}}$, where the inputs come from a compact set U_{in} and the desired outputs $\mathbf{y}_{\text{teach}}(n)$ from a compact set U_{out}. Wanted: a RNN whose output $\mathbf{y}(n)$ approximates $\mathbf{y}_{\text{teach}}(n)$.

Procure an echo-state network. Using the theoretical results from Section 2, construct a RNN that has echo states in an admissible state set A with respect to input $\mathbf{u}_{\text{teach}}$ from U_{in} and "pseudo"-input $\mathbf{y}_{\text{teach}}(n)$ from U_{out}. In formal terms, (i) re-interpret the $N \times L$ matrix $\mathbf{W}^{\text{back}}$ as another input weight matrix and join it with $N \times K$-matrix $\mathbf{W}^{\text{in}}$ into a $N \times (K + L)$-matrix $\mathbf{W}^{\text{in \& back}}$, (ii) join the input

$\mathbf{u}_{\text{teach}}(n+1)$ with the output $\mathbf{y}_{\text{teach}}(n)$ into a compound pseudo-input $\tilde{\mathbf{u}}_{\text{teach}}(n+1) = (\mathbf{u}_{\text{teach}}(n+1), \mathbf{y}_{\text{teach}}(n))$, and (iii) make sure that the resulting network has the echo state property in an admissible state set A with respect to input $\tilde{\mathbf{u}}_{\text{teach}}(n+1)$ from the compact set $\tilde{U} = U_{\text{in}} \times U_{\text{out}}$.

A convenient heuristic to obtain such a network is again provided by Proposition 3(b): any standard sigmoid network whose weight matrix $\mathbf{W}$ satisfies $|\lambda_{\max}| < 1$ will (– according to experience –) do. Experience also suggests that the echo state property is independent of the input connections $\mathbf{W}^{\text{in}}$, which can be freely chosen; furthermore, it appears also to be independent of the output back projection connection weights $\mathbf{W}^{\text{back}}$, which can also be set arbitrarily.

Run network with teacher input and with teacher output forcing, dismiss initial transient. Start with an arbitrary network state $\mathbf{x}(0)$ and update the network with the training input and teacher-forced output for $n = 0, \ldots, n_{\max}$:

$$\mathbf{x}(n+1) = \mathbf{f}\left(\mathbf{W}^{\text{in}}(\mathbf{u}_{\text{teach}}(n+1)) + \mathbf{W}^{\text{back}}\mathbf{y}_{\text{teach}}(n) + \mathbf{W}\mathbf{x}(n)\right). \quad (15)$$

Dismiss from this run a suitably long initial transient up to $n_{\min}$.

Compute output weights that minimize the output MSE. The rest of the training is identical to the restricted case.

3.4 Example with output feedback

To illustrate that output feedback works similarly like ordinary input feed-in, we reconsider the $\sin^7$ example. This time, we give the network no input; it has to generate the desired waveform autonomously.

The same network as above was used, but without an input unit. The output feedback weights were set to values of +1, -1 with equal probability. The network was trained from a run of 300 steps, of which the first 100 were discarded. Fig. 3 shows the results. The training error was $\text{mse}_{\text{train}} \approx 1.3 \times 10^{-9}$. In order to determine a test error, the network was run for 100 steps with teacher-forcing of the teacher signal in the output unit, then was unlocked from teacher forcing. During the first free-running 100 steps, a $\text{mse}_{\text{test}} \approx 2.6 \times 10^{-8}$ was found. After more free-running steps, the teacher and and network signal diverge more strongly. This is owed to the fact that the frequency of the network signal is not completely the same as the teacher's, so a phase lag starts to build up. After 1000 steps, we observe $\text{mse}_{\text{test}} \approx 1.0 \times 10^{-5}$; in plots the two signals are still undistinguishable.

Figure 3: Traces of some selected units of a 100-unit echo state network trained to generate a $1/2 \sin^7(n/5)$-signal. The first seven traces show arbitrarily selected internal units during a test run. The output signal is shown in the last trace.

4 Two basic experiments

We now demonstrate how the echo state approach copes with two learning tasks that are mathematically fundamental: periodic sequence learning and switchable point attractor learning. The first is a primordeal form of temporal memory, the second, of static pattern memory.

4.1 Discrete-periodic sequence learning

Here we investigate how our network learns a sequence which is periodic with an integer period length. (Note that the $\sin^7$ examples are periodic with a period that stands in no rational ratio with the update interval). Mathematically speaking, we train the network to cycle through a periodic attractor; in everyday language, it learns to play a melody.

4.1.1 House of the Rising Sun

In the first task, the target signal was prepared from "The House of the Rising Sun". The notes of this melody were assigned numerical values ranging from -1 (g#) to 14 (a'), with halftone intervals corresponding to unit increments. Fig. 4 shows this melody. These values were divided by 28 to make the melody range fit into the network output unit's range of $[-1, 1]$ (an output function $f^{\text{out}} = \tanh$ was used). This resulted in a periodic sequence of period length 48. The target sequence $(y(n))_{n \geq 0}$ is a concatenation of identical copies of this melody.

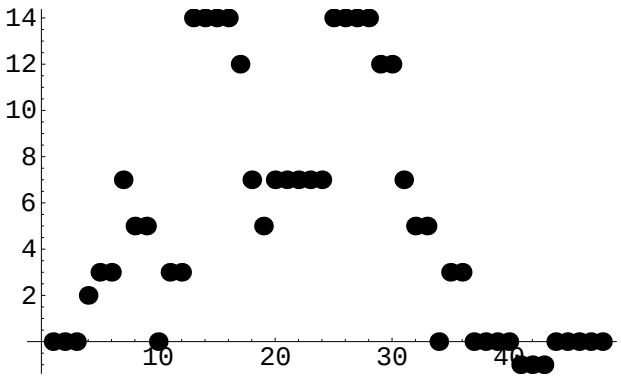

Figure 4: The melody of "House of the Rising Sun" before squashing into the network output interval $[-1, 1]$.

4.1.2 Network preparation

A 400 unit sigmoid network was used. Internal connections were randomly assigned values of $0, 0.4, -0.4$ with probabilities $0.9875, 0.00625, 0.00625$. This resulted in a weight matrix $\mathbf{W}$ with a sparse connectivity of 1.25%. The maximal eigenvalue of $\mathbf{W}$ was $|\lambda_{\max}| \approx 0.908$. The fact that $|\lambda_{\max}|$ is close to 1 means that the network exhibits a long-lasting response to a unit impulse input. Fig. 5 shows how the network (without output units, before training) responds to a unit impulse input.

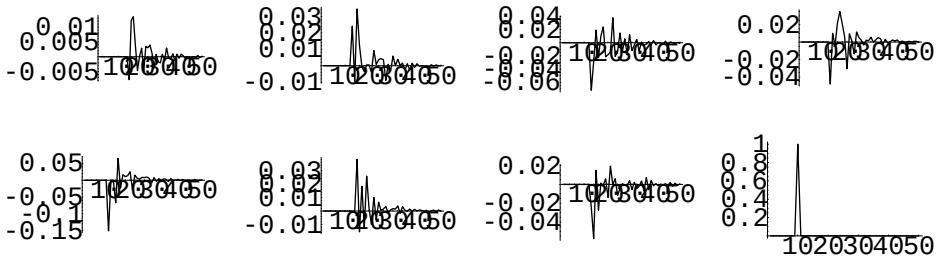

Figure 5: Response of 7 arbitrarily selected units of the network to an impulse input. The last trace shows the input.

Comment. Generally, the closer $|\lambda_{\max}|$ is to unity, the slower is the decay of the network's response to an impulse input. A relatively long-lasting "echoing" of inputs in the internal network dynamics is a requisite for a sizable short-term memory performance of the network. A substantial

short-term memory is required for our present learning task, because the target signal contains a subsequence of 8 consecutive 0's (namely, the last 5 notes concatenated with the first 3 notes of the subsequent instance of the melody). This implies that the network must realize a memory span of at least 9 update steps in order to correctly produce the first non-0 output after this sequence.

Since at time n the output $y(n)$ depends on the previous outputs, output feedback is required in this task. The output feedback weights were sampled randomly from the uniform distribution in $[-2, 2]$.

Figure 6 shows the setup of the network prepared for this learning task.

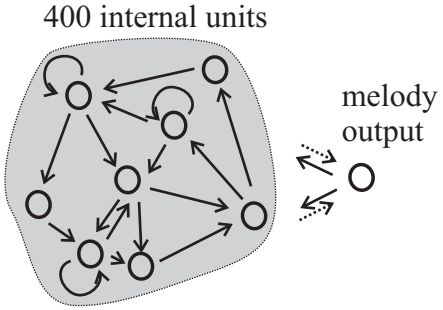

Figure 6: Network setup of the "melody" periodic sequence learning task. Solid arrows indicate weights that were fixed and remained so during learning, dashed arrows mark weights to be learnt.

4.1.3 Training and testing

The network was trained first in a "naive" fashion, which fails in an interesting way. We describe training and failure and show how the latter can be remedied.

The network was run for 1500 steps with teacher-forced melody. The inital 500 steps were discarded, and the output connections computed from the network states that were collected during the remaining period.

The training error was $\text{mse}_{\text{train}} \approx 2.2 \times 10^{-31}$, i.e. zero up to machine precision. This zero training error can be explained as follows. After the initial 500 steps, the network states $\mathbf{x}(n)$ had converged to a periodic sequence, i.e. for $n \geq 500, k = 1, 2, \ldots$, $\mathbf{x}(n) = \mathbf{x}(48\,k + n)$ up to machine precision. According to Eq. (14), the 400 output weights $\mathbf{W}^{\text{out}} = (w_{1i})_{i=1,\ldots,400}$ were computed such that

$$\mathrm{mse}_{\mathrm{train}} = 1/1000 \sum_{n=501}^{1500} \left((f^{\mathrm{out}})^{-1} y_{\mathrm{teach}}(n) - \mathbf{W}^{\mathrm{out}} \mathbf{x}(n) \right)^2 . \qquad (16)$$

was minimized. Since the network states $\mathbf{x}(n)$ are 48-periodic, (16) effectively reduces to

$$\mathrm{mse}_{\mathrm{train}} = 1/48 \sum_{n=501}^{548} \left((f^{\mathrm{out}})^{-1} y_{\mathrm{teach}}(n) - \mathbf{W}^{\mathrm{out}} \mathbf{x}(n) \right)^2 , \qquad (17)$$

which renders the computation of the weights $\mathbf{W}^{\mathrm{out}}$ underdetermined, because the linear mapping $\mathbf{W}^{\mathrm{out}} : \mathbb{R}^{400} \to \mathbb{R}$ is determined by (17) only through 48 linearly independent arguments. Therefore, there exist many perfect solutions. The `Fit` procedure of Mathematica selects one of them.

In order to check whether the network had indeed learnt its task, it was tested whether it could stably continue to generate the desired ouputs after a starting period with teacher forcing. The network was started from the null state $\mathbf{x}(0) = \mathbf{0}$, and the correct melody was written into the output units for 500 initial steps. After this initial period, the network state had developed into the same periodic state set as during training. Then the network was left running freely for further 300 steps. Figure 7 shows what happens: the network continues to produce the desired periodic output sequence for some time but then degenerates into a novel dynamics which has nothing in common with the desired target.

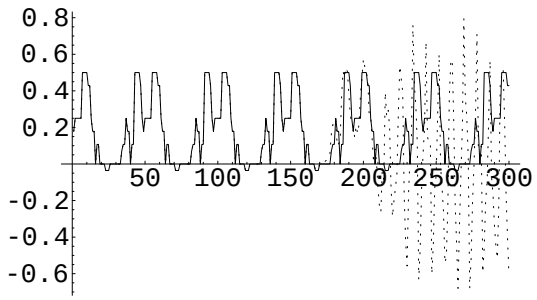

Figure 7: Degeneration of network performance trained with the naive algorithm. Solid line: target, dashed line: network output.

Obviously, the solution found in the "naive" training is unstable. This can be stated more precisely as follows. Consider the state $\mathbf{x}(501)$ and the 48-step update operator T^{48}. Then $\mathbf{x}(501)$ should be a fixed point of T^{48}.

Up to some very high precision this is true, as becomes evident from the first precise repetitions of the melody in Fig. 7. Unfortunately, $\mathbf{x}(501)$ is not a stable fixed point.

Likewise, the states $\mathbf{x}(502), \ldots, \mathbf{x}(548)$ should be stable fixed points of T^{48}. (Actually, stability of $\mathbf{x}(501)$ implies stability of these others).

How can this fixed point be stabilized? Recall that the computation of weights in (17) was underdetermined. Therefore, we have still the possibility to find alternative solutions of (17) which likewise render $\mathbf{x}(501)$ a fixed point, but a stable one. One trick to achieve this goal is to insert noise into the training procedure.

Instead of updating the network according to Eq. (1), during training in steps 1 to 1500 we use

$$\mathbf{x}(n+1) = \mathbf{f}\left(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{back}}(\mathbf{y}(n) + \nu(n))\right), \tag{18}$$

to update the network state, where $\nu(n)$ is a noise term sampled from a uniform distribution over $[-0.001, 0.001]$. This noise makes the network states "wobble" around the perfect periodic state sequence used in the naive approach. Now the computation of weights minimizes

$$\text{mse}_{\text{wobbletrain}} = 1/1000 \sum_{n=501}^{1500} \left((f^{\text{out}})^{-1} y_{\text{teach}}(n) - \mathbf{W}^{\text{out}} \mathbf{x}_{\text{wobble}}(n)\right)^2. \tag{19}$$

Since now we have truly 1000 arguments $\mathbf{x}_{\text{wobble}}(n)$ (instead of effectively only 48), the linear regression is overdetermined and a nonzero training error of $\text{mse}_{\text{wobbletrain}} \approx 1.3 \times 10^{-9}$ is obtained.

The solution found with this wobbling trick is stable. Fig. 8 shows what happens when the trained network is started from the null state with an initial teacher-forced startup of 12 steps. This short initialization brings the network state close, but not very close to the proper state. Fig. 8(a) shows the first 100 free-running steps after this startup. The initial state discrepancy leads to a visible initial discrepancy between target and network output. After more free-running steps, the network eventually settles into the desired attractor up to a residual $\text{mse}_{\text{test}} \approx 10^{-7}$.

This "stabilization through wobbling" can be stably reproduced: it works every time the training procedure is repeated. However, I cannot offer a rigorous explanation. Intuitively, what happens is that the weights are computed such that small deviations from the target state/output sequence are counteracted as good as possible, exploiting the remaining $400 - 48 = 352$ degrees of freedom in the weight determination.

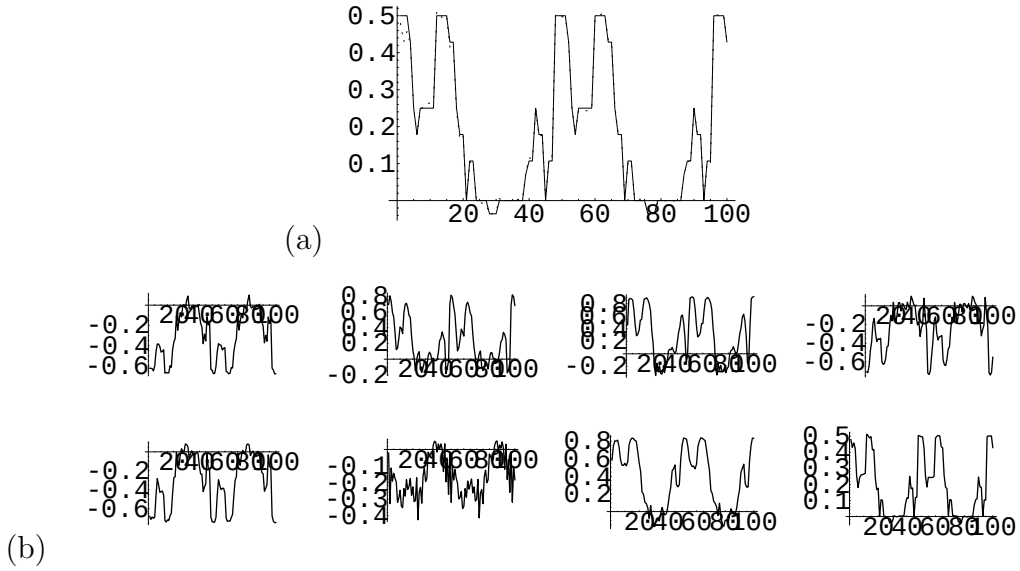

Figure 8: Network performance after "wobbling" stabilization. (a) First 100 steps after 30 step teacher forced initialization: a slight discrepancy between target and network output in the beginning is visible, but then the network gets caught in the target attractor. (b) Traces of some internal states. The last trace is from the output unit.

4.1.4 Comments

We conclude this treatment of melody learning with miscellaneous comments.

- The choice of network parameters $\mathbf{W}$ is robust. No delicate parameter tuning was necessary besides respecting that $|\,\lambda_{\max}\,|$ be close to unity in order to assure a sufficiently long decay time of output feedback effects in the network state, which is required for the network to bridge consecutive identical outputs. Using the same network, but with internal connection weights globally scaled such that $|\,\lambda_{\max}\,| = 0.987$ or $|\,\lambda_{\max}\,| = 0.79$ did not affect the success of stable learning. If, however, the network was scaled to the smaller value $|\,\lambda_{\max}\,| = 0.69$, solutions could not be stabilized through the wobbling trick any more. The maximal singular value of the weight matrix in this last (useless) case was $\sigma_{\max} \approx 1.49$, which is considerably greater than 1. Further downscaling of the weight matrix would be necessary to achieve $\sigma_{\max} < 1$, which would lead even further into the "non-stabilizable" region. This is an indication that the sufficient condition for echo states of Prop. 3(a)

is overly restrictive, at least in cases where "long short-term memory" properties are requested from the network.

- The sizing of output feedback weights $\mathbf{W}^{\text{back}}$ is likewise robust. The training went as well when the feedback weights were chosen from $[-0.5, 0.5]$ or from $[-4, 4]$.

- The magnitude of noise is also uncritical. The learning works with no obvious difference whether noise is taken from $[-0.01, 0.01]$ or from $[-0.000001, 0.000001]$.

- The same network was trained to generate periodic random sequences of period length 96, which succeeded, and to periodic sequences of period length 104, which didn't. In the latter case, the training error was still of the same (zero) size as in our length 48 example, but "stabilization through wobbling" wouldn't succeed any more. It appears that somewhere between 96 and 104 there is a maximal sequence length which this echo state network could *stably* reproduce. Stated somewhat imprecisely, the network needs to have enough "residual" degrees of freedom to stabilize the fixed point. This matter deserves a more thorough investigation.

- *Stable* generation of a periodic sequence requires a *nonlinear* network. It can be easily seen that if the network had linear units (i.e., it $\mathbf{f}$ and $\mathbf{f}^{\text{out}}$ were linear), one could only realize a linear (autoregressive, AR) type of system. Such systems are inherently incapable of settling into a stable attractor, because if $y(n)$ is a periodic signal generated by an AR system, then also $Cy(n)$ would be a possible output signal for every $C \in \mathbb{R}$.

- There is no obvious upper limit to the period length achievable with this approach: longer periodic sequences should be learnable by accordingly larger echo state networks.

4.2 Multiple attractor learning

We now describe an example which can be seen as dual to the previous one. While there we trained a multipoint attractor, we will now train multiple point attractors. Suitable input signal can switch between them.

We consider the following task. The network has m input units and m output units. The input signal are rare spikes, which arrive randomly in the input units. With each input unit we associate a point attractor of the

network. If a spike arrives at input unit i, the network is switched into its ith point attractor, where it stays until another spike arrives at another input unit. The output units indicate in which attractor the system is currently in. The ith output unit has activation 0.5 when the system is in the ith attractor; the remaining output units have then an activation of -0.5. Figure 9 provides a sketch of the setup.

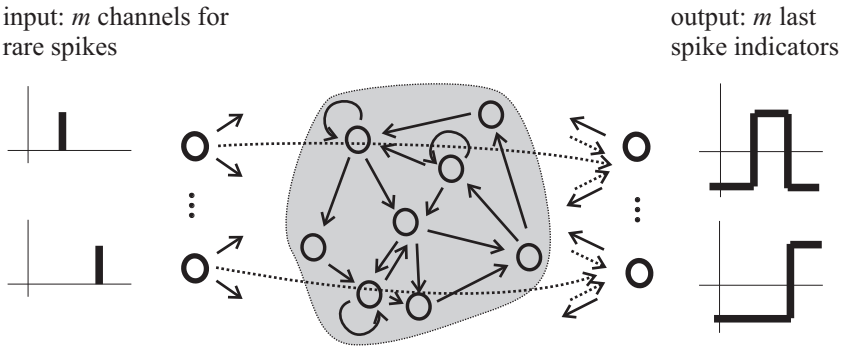

Figure 9: Setup of the switchable attractor network. Solid lines mark connections whose weights are unchanged by training, dashed connections are trained.

4.2.1 Task specification

The input signal is a vector $\mathbf{u}(n) = (u_1(n), \ldots, u_m(n))$, where $u_i(n) \in \{0, 0.5\}$. At any time n, at most one of the input channels "spikes", i.e. takes a value of 0.5. Spikes are rare: the probability that at time n one of the channels spikes is 0.02.

The output is a vector $\mathbf{y}(n) = (y_1(n), \ldots, y_m(n))$, where $y_i(n) \in \{-0.5, 0.5\}$. The desired output is $y_i(n) = 0.5$ if the ith input spiked at some time $n - k$ (where $k \geq 0$) and no other input channel spiked during $n, n-1, \ldots, n-k+1$.

We report here on a task where the number of channels/attractors $m = 20$.

4.2.2 Network preparation

A 100 unit standard sigmoid network was randomly connected (the same network as in the $\sin^7$-example was used). Weights were scaled such that $\mid \lambda_{\max} \mid \approx 0.44$ was obtained. This value is much smaller than unity and guarantees a fast input/state forgetting. The weight matrix had a maximal

singular value of $\sigma_{\max} \approx 0.89$. Thus, in this example the sufficient condition for echo states given in Prop. 3(a) was satisfied.

20 input units and 20 output units were attached. For each input channel, the input connection weights were randomly chosen to be 5 or -5 with equal probability. The network had output feedback connections, which were randomly set to either of the three values of $0, 0.1, -0.1$ with probabilities $0.8, 0.1, 0.1$.

4.2.3 Training and testing

The training data consisted of a 4050-step input/output sequence. Starting with a spike at input channel $i = 1$ at time $n = 1$, every 200 steps one spike was issued, cycling once through the input channels such that at time 4001 the first input channel spiked again. The outputs were set to -0.5 or 0.5 as specified above.

The network was run with these training data, where the output signals were teacher-forced into the output units. For updating, Eq. (1) was used (no extra noise insertion).

The first 50 steps were discarded and the output weights were computed from the network states collected from $n = 51$ through $n = 4050$. The twenty training errors varied between a smallest value of $\mathrm{mse}_{\mathrm{train},i} \approx 8 \times 10^{-7}$ and a largest value of $\approx 8 \times 10^{-6}$.

The trained network was tested by feeding it with long input sequences prepared according to the task specification. The 20 test errors varied across the output channels between values of 6×10^{-6} and 5×10^{-5}. Fig. 10 gives a demonstration of the trained network's performance.

It is not difficult to understand how and why the trained network functions. There are two aspects to account for: (1), why the network locks into stable states in the absence of input spikes, (2) why the network switches states according to incoming spikes.

We first discuss the first point. During training, for extended periods of time the network's input and teacher-forced output feedback signals remain fixed, namely, during the intervals after some spike arrives and before the next spike arrives. Because of the network's fast input/state forgetting characteristics, the network quickly converges to stationary states during these training subintervals. A consequence of the learning algorithm (and again, an underdetermination of the linear regression) is that these states become fixed points of the network's dynamics (while there is no input spike). We can observe in the testing phase that these fixed points are stable. Like in the periodic sequence example, I cannot offer a rigorous explanation of this fact. However, hypothetically a similar "wobbling" mechanism as above sta-

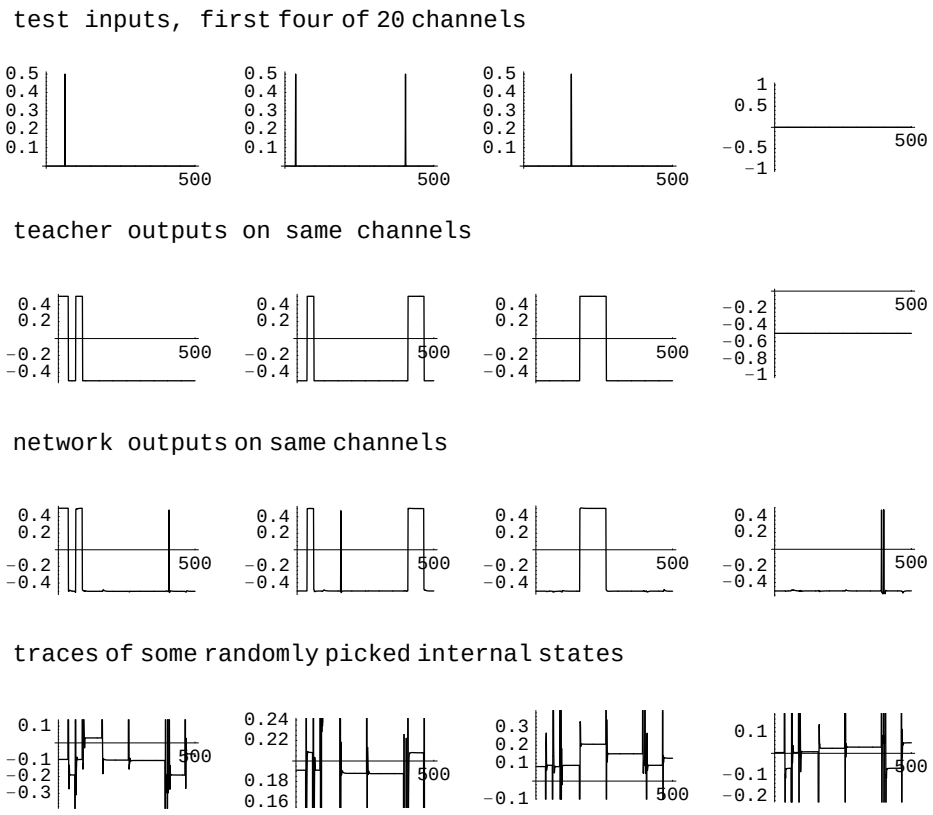

Figure 10: A 500-step section of a test run of the trained multi-attractor network. The first four (of twenty) inputs and outputs are shown. Note that in the shown interval, spikes arrived at other input channels than the four shown ones. For further comments see text.

bilizes these fixed points. The role of noise from the previous example is filled here by two other sources of deviations. The first deviation from the fixed point states in the training material comes from the short transient dynamics after input spikes (compare last sequence in Fig. 10). The second source arises from the fact that an output value of -0.5 at output node i is trained from 19 different fixed points, namely, the fixed network states obtained in the 19 interspike intervals. Both effects together and the fact that there are $100 - 20 = 80$ freely disposable parameters in each output weight vector is apparently sufficient to make these fixed points stable.

The switching between different stable states may be understood as follows. Due to the large absolute values (5.0) of the input connections, an input spike that arrives at time n at channel i essentially erases all preceding

state information from the network and sets the states to $x_j(n) = 0.5\, f_j w^{\text{in}}_{ji}$. Let's call this the "signature state" of the ith channel. The linear regression maximises the immediate (same time n) output response of the ith output unit to this signature state, and minimizes the response of all others. Therefore, in the trained network the immediate response of the ith output unit will be greater than the response of all other output units. After time n, the network evolves according to the trained fixed point dynamics. For the short transient after time n, the trained response also favors 0.5-valued responses in the ith unit and -0.5-valued responses in the others. The resulting transient dynamics in the test phase is apparently close enough to the corresponding transient dynamics in the (teacher-forced) training phase to ensure that the network settles into the desired state.

A closer inspection of the traces in Fig. 10 (third row) reveals that some output units i "spike" ephemerically (without locking into their attractor) also when input spikes arrive at unassociated channals i'. This must happen when the signature states of the two channels have a high similarity in the sense that $\langle \mathbf{w}^{\text{in}}_i, \mathbf{w}^{\text{in}}_{i'} \rangle \gg 0$ (here, $\langle \cdot, \cdot \rangle$ denotes inner product). This situation has to be expected for some pairs i, i' due to the random choice of the input weights. If the input weight vectors would have been chosen orthogonally, we may hypothesize that these erroneous ephemeral output spikes would not occur. However, this has not been tested.

5 Echo state networks with leaky integrator neurons

The units in standard sigmoid networks have no memory; their values at time $n+1$ depend only fractionally and indirectly on their previous value. Thus, these networks are best suited for modeling intrinsically discrete-time systems with a "computational", "jumpy" flavor. It is difficult, for instance, to learn slow dynamics like very slow sine waves. For learning slowly and continuously changing systems, it is more adequate to use networks with a continuous dynamics. We take a hybrid approach here and use discrete-time networks whose dynamics is a coarse approximation to continuous networks with leaky integrator units. The evolution of a continuous-time leaky integrator network is

$$\dot{\mathbf{x}} = C\left(-a\mathbf{x} + \mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x} + \mathbf{W}^{\text{back}}\mathbf{y})\right), \tag{20}$$

where C is a time constant and a the leaking decay rate. When this differential equation is turned into an approximate difference equation with stepsize

δ, one obtains

$$\mathbf{x}(n+1) = (1-\delta Ca)\mathbf{x}(n) + \delta C\left(\mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{back}}\mathbf{y}(n))\right). \tag{21}$$

If all internal unit output functions are standard sigmoids $f_i = \tanh$, we can spell out an analog to Prop. 3 for a system of the kind (21).

Proposition 4 *Let a network be updated according to (21), with teacher-forced output. Let $f_i = \tanh$ for all $i = 1, \ldots, N$. (a) If $\mid 1-\delta C(a\ -\sigma_{\max}) \mid = L < 1$ (where $\sigma_{\max}$ is the maximal singular value of the weight matrix $\mathbf{W}$), the network has echo states, for every input and/or output feedback. (b) If the matrix $\tilde{\mathbf{W}} = \delta C\mathbf{W} + (1-\delta Ca)\mathbf{id}$ (where $\mathbf{id}$ is the identity matrix) has a spectral radius greater than 1, the network has no echo states for any input set containing $\mathbf{0}$ and admissible state set $[-1,1]^N$.*

The proof is sketched in the Appendix.

By choosing sufficiently small time constants C, one can employ echo state networks of type (21) to model slow systems. The training is done in a completely analogous way as described in Section 3, with one optional modification: If a small stepsize δ is used, one can downsample the network states collected in the training period before the linear regression computation is performed. This reduces the computational load of the regression computation and does not markedly impair the model quality, because network states between two sampled states will be almost linear interpolates of the latter and thus do not contribute relevant new information to the regression computation.

6 The Mackey-Glass system

A popular test for learning dynamical systems from data is chaotic attractor learning. A particularly often used system is defined by the Mackey-Glass delay differential equation

$$\dot{y}(t) = \alpha\, y(t-\tau)/(1+y(t-\tau)^{\beta}) - \gamma y(t), \tag{22}$$

where invariably in the chaotic systems modeling community the parameters are set to $\alpha = 0.2, \beta = 10, \gamma = 0.1$. We will use the same parametrization. The system has a chaotic attractor if $\tau > 16.8$. In the majority of studies, $\tau = 17$ is used, which yields a mildly chaotic attractor. A more rarely used value is $\tau = 30$, which leads to a wilder chaotic behavior. We will tackle both the mild and the wild system.

6.1 Task specification

Several discrete-time training sequences were prepared by first approximating (22) through

$$y(n+1) = y(n) + \delta \left(\frac{0.2\, y(n-\tau/\delta)}{(1+y(n-\tau/\delta)^{10}} - 0.1\, y(n) \right) \tag{23}$$

with stepsize $\delta = 1/10$ and then subsampling by 10. One step from n to $n+1$ in the resulting sequences corresponds to a unit time interval $[t, t+1]$ of the original continuous system.

In this manner, four training sequences were generated, two of length 3000 and two of length 21000. For each length, one sequence was generated with a delay of $\tau = 17$ and the other with $\tau = 30$. These sequences were then shifted and squashed into a range of $[-1, 1]$ by a transformation $y \mapsto \tanh(y-1)$.

The task is to train four echo-state networks from these training sequences, which after training should re-generate the corresponding chaotic attractors.

Fig. 11 shows 500-step subsequences of these training sequences for $\tau = 17$ and $\tau = 30$.

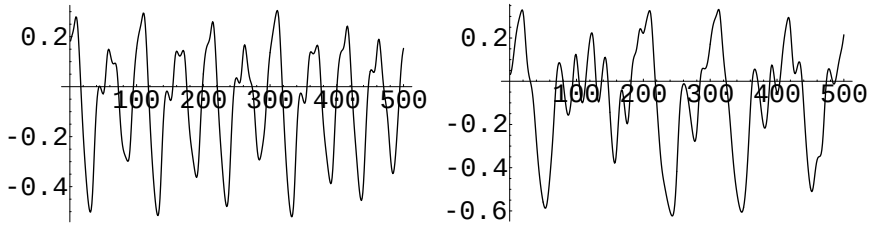

Figure 11: 500-step sections of the training sequences for delays $\tau = 17$ (left) and $\tau = 30$ (right).

6.2 Network preparation

Essentially the same 400-unit network as in the periodic attractor learning task was used. Its weight matrix $\mathbf{W}$ was globally rescaled such that $|\lambda_{\max}| \approx 0.79$ resulted.

The network update was done according to Eq. (21) with stepsize $\delta = 1$, global time constant $C = 0.44$ and decay rate $a = 0.9$. (This yields an effective spectral radius of $|\lambda_{\max}|\,(\tilde{\mathbf{W}}) \approx 0.95$ according to Prop. 4(b)).

One input unit was attached which served to feed a constant bias signal $u(n) = 0.2$ into the network. The input connections were randomly chosen to be $0, 0.14, -0.14$ with probabilities $0.5, 0.25, 0.25$. The purpose of this bias is to increase the variability of the individual units' dynamics.

One output unit was attached with output feedback connections sampled randomly from the uniform distribution over $[-0.56, 0.56]$.

6.3 Training and testing for $\tau = 30$

6.3.1 The length 3000 training sequence

The network was run from a zero starting state in teacher-forced mode with the $\tau = 30$, length 3000 training sequence. During this run, noise was inserted to the network using the following variant of (21):

$$\mathbf{x}(n+1) = \qquad (24)$$
$$(1 - \delta Ca)\mathbf{x}(n) + \delta C \left(\mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{back}}\mathbf{y}(n) + \nu(n))\right),$$

where the noise vector $\nu(n)$ was sampled from $[-0.00001, 0.00001]^N$. The first 1000 steps were discarded and the output weights were computed by a linear regression from the remaining 2000 network states.

The reason for discarding such a substantial number of initial states is that $\mid \lambda_{\max} \mid (\tilde{\mathbf{W}})$ is close to unity, which implies a slow forgetting of the starting state.

The network's performance was first visually judged by comparing the network's output with the original system. To this end, the trained network was run for 4000 steps. The first 1000 steps were teacher-forced with a newly generated sequence from the original system. The network output of the remaining free-running 3000 steps was re-transformed to the original coordinates by $y \mapsto \operatorname{arctanh}(y) + 1$. The resulting sequence $y(n)$ was rendered as a two-dimensional plot by plotting points $(y(n), y(n+20))$ and connecting subsequent points. A similar rendering of the original 3000 step training sequence was generated for comparison. Figure 12 shows the original attractor plot (left) and the plot obtained from the trained network's output (right). It appears that the learnt model has captured the essential structure of the original attractor. Not much more can be gleaned from these plots, besides maybe the fact that the network stably remained in the desired attractor state for at least 3000 steps.

In the literature one often finds plots where the original system's evolution is plotted together with the evolution predicted from a learnt attractor model. With chaotic systems, such plots are not very informative, because

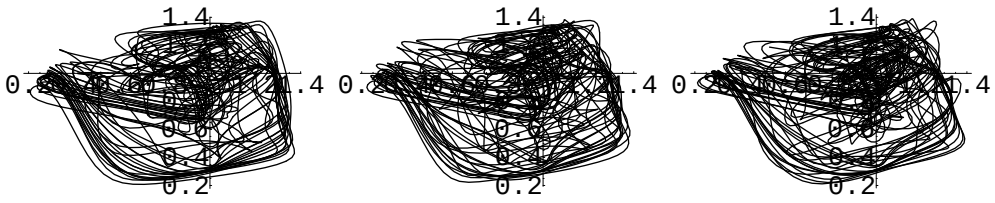

Figure 12: Case $\tau = 30$. Attractors of the original system (left), the network trained from the 21000 step sequence (center) and the network trained from the 3000 step sequence (right). For details see text.

the evolution of chaotic attractors sometimes goes through periods where prediction is easy (in the sense that small deviations do not diverge quickly), but at other times goes through very instable periods where small deviations explode quickly. Figure 13 demonstrates this fact. The plots in this Figure were obtained by teacher-forcing the network with the original attractor signal for 1500 steps and then letting it run freely for another 500 steps. The free-running steps are plotted. The left diagram shows a case where prediction is easy. The right shows another case where the original (and the learnt) system go through a highly divergent period around $n = 200$.

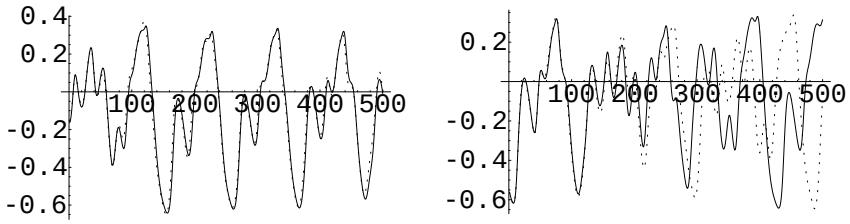

Figure 13: Two test runs for $\tau = 30$, model learnt from the 3000 step training sequence. The free running network's output (dashed) is plotted against the original attractor (solid line).

A more informative measure of model quality, which is standardly given, is the normalized root mean square error (NRMSE) of model predictions for a given prediction horizon. A much-used prediction horizon is to compare the model's prediction 84 steps ahead, $\hat{y}(n + 84)$ (after retransformation to the original coordinates), with the original value $y(n+84)$ (after retransformation to the original coordinates). To obtain this NRMSE_{84}, we ran the network 50 times. Each run consisted of a 1000 step initial period with teacher forcing

and a subsequent 84 step free run. The original evolutions were taken from 50 subsequent 1084-step evolutions of a 50×1084 run of the original attractor. At each of the 50 runs, the values $y_i(n+84)$ and $\hat{y}_i(n+84)$ were collected $(i = 1, \ldots, 50)$ and the NRMSE_{84} was computed by

$$\mathrm{NRMSE}_{84} \approx \left(\frac{\sum_i (y_i(n+84) - \hat{y}_i(n+84))^2}{50\sigma^2} \right)^{1/2}, \tag{25}$$

where $\sigma^2 \approx 0.067$ is the variance of the original attractor signal. For the model learnt from the 3000 step training sequence, a $\mathrm{NRMSE}_{84} \approx 0.11$ was found.

The work reported in [5] contains the best modeling of the $\tau = 30$ case I am aware of. The authors use a multilayer perceptron with output feedback, which is trained from 1880-step training sequences with a refined version of backpropagation through time, "extended Kalman filter multi-stream training". In [5] another error criterium is used, namely the root mean square error of 120-step predictions:

$$\mathrm{RMSE}_{120} \approx (\langle (y(n+84) - \hat{y}_i(n+84))^2 \rangle)^{1/2}, \tag{26}$$

where $\langle \cdot, \cdot \rangle$ denotes expectation. We estimated this value from 50 runs for the 3000-step-trained model and found $\mathrm{RMSE}_{120} \approx 0.048$. The error given in [5] is $\mathrm{RMSE}_{120} \approx 0.040$, a value similar in magnitude to our result.

6.3.2 The nuisance of initial state determination

In the echo state approach described here, a substantial amount of training data is wasted for the discarded initial period of the training run. Likewise, when the trained network is used to predict a sequence, a long initial run of the sequence must be fed into the network before the actual prediction can start. This is a nuisance because such long initial transients are in principle unneccessary. By Takens' theorem [16], very few (actually, 4) successive data points $y(n), y(n+k), y(n+2k), y(n+3k)$ suffice to fully determine the attractor's state at time $n + 3k$. This is exploited in most approaches to attractor modeling: typical prediction models found in the literature predict a future data point $y(n+3k+l)$ from $y(n), y(n+k), y(n+2k), y(n+3k)$ (mostly $k = 6$ is used with the Mackey-Glass system). Such models allow the user to start predictions from prior histories with a length of only $3k+1$.

By contrast, a recurrent neural network such as our echo state network, but also such as the networks used in [5] need long initial runs to "tune in" to the to-be-predicted sequence. [5] tackle this problem by training a second, auxiliary "initiator" network. This is a feedforward network which

computes an appropriate starting state of the recurrent model network from a $3k+1$-long initial sequence.

This idea could be adapted to our echo state network approach in the following way.

1. Run the network in teacher-forced mode on the training data, discard initial transient as above. Collect the network states $\mathbf{x}(n) = (x_i(n))_{i=1,\ldots,N}$ over the remaining steps $n_{\min}, \ldots, n_{\max}$.

2. For each internal unit i of the echo state network, compute a separate weight vector $\mathbf{W}_{\text{stateInit},i}$ of length N, in the following way.

   (a) For $m = n_{\min} + 3k$ to $m = n_{\max}$, run the network in from zero starting state for $3k+1$ steps, teacher-forcing it with a subsequence of the original training subsequence taken from steps $m - 3k$ to m. For each m, collect the network step $\mathbf{x}_m$ obtained at the end of this short run.

   (b) By a linear regression, compute $\mathbf{W}_{\text{stateInit},i}$ such that the MSE

   $$1/(n_{\max} - (n_{\min} + 3k)) \sum_{m=n_{\min}+3k,\ldots,n_{\max}} (x_i(m) - \mathbf{W}_{\text{stateInit},i}\mathbf{x}_m) \quad (27)$$

   is minimized.

3. To compute an appropriate network starting state for the prediction task from a prior history of short length $3k+1$, run the network from zero starting state for $3k+1$ steps, teacher-forcing it with the short initial history. The resulting final network state is $\tilde{\mathbf{x}}$. Compute a starting state $\mathbf{x}(3k+1)$ for the prediction task by putting $\mathbf{x}(3k+1) = (\mathbf{W}_{\text{stateInit},1}\tilde{\mathbf{x}}, \ldots, \mathbf{W}_{\text{stateInit},N}\tilde{\mathbf{x}})$.

The logic of this procedure is intuitively clear but remains to be implemented and tested.

6.3.3 The length 21000 training sequence

The same experiment was done with the 21000-step training sequence. Due to memory restrictions on my PC (the Mathematica `Fit` procedure is memory intensive) the task had to be broken up into four tasks of length 6000 each (splitting the 21000 training sequence into four with 1000 step overlaps). From each of the four length 6000 training sequences, output weight vectors

were computed as above. These four vectors were then averaged to obtain a final weight vector. This averaging results in a poorer estimate than if a full length 21000-step training could have been performed, but it still improves over the single 6000 step weight vectors.

The noise inserted here was smaller than in the 3000-step trial; it was sampled from $[-0.00000001, 0.00000001]^N$.

Figure 12 (center) shows the resulting learnt attractor. The 84-step prediction error (again averaged from 50 prediction runs) was $\text{NRMSE}_{84} \approx 0.032$. To my knowledge, this is the best prediction model for the $\tau = 30$ Mackey-Glass attractor achieved with any method and any training data size so far.

This experiment was carried out mainly to probe the representational limits of the 400-unit echo state network. This trial demonstrates that echo state networks are well-suited devices for chaotic attractor modeling, in the sense that by learning a relatively small number of parameters (400 here) in a generic model structure very close fits to complex dynamics can be achieved.

6.3.4 Stability and noise insertion

Noise insertion was found to be crucial for stability in the $\tau = 30$ task. Trials with smaller amounts of noise than reported above resulted in unstable trained networks. Their dynamics left the desired attractor typically after about 100 steps and turned into fixed points or oscillations. Also, it was found that when noise was added to the output feedback as in the melody learning task (instead to the network states) the resulting networks needed much larger amounts of noise to become stable. The larger amount of noise degraded the precision of the learnt model. It remains to be investigated why noise insertion to the states was superior to noise insertion to output feedback in this task.

While a rigorous theoretical understanding of the stabilizing function of noise is lacking, it is worth mentioning that there appears to exist a stability-precision tradeoff. Larger noise was generally found to result in more stable models (in the sense that they converged to the desired attractor from a larger subspace of starting states) but at the same time also meant less precise prediction.

6.3.5 A note on parameter selection

The general structure of the network was arbitrarily generated, including connectivity and setting of the input and output feedback weights. In fact, I

have been re-using the very same basic 400-unit network for a large variety of tasks since I started investigating echo states networks.

However, there are a handful of parameters which were optimized by hand for this particular task: the global scaling of the raw echo state network (which determines $\mid \lambda_{\max} \mid$), a global scaling factor for the input weights, another global scaling factor for the output feedback weights, the global time constant C and the leaking decay rate a. The optimiziation was done in a crude fashion, one parameter after the other, testing a few settings of each parameter on small test problems and choosing the best-performing one. No iteration of the procedure was done after all parameters had been optimized once. The overall time spent on this was about one hour of manual experimentation, excluding PC running time.

A more systematic cross-optimization of the parameters would probably improve the training results, but much of the charm of the echo state approach lies in its simplicity and robustness. The basic experience of working with echo state networks is that with a small effort good results can be achieved. However, it should be emphasized that a rough optimization of these few parameters is crucial and task-specific.

6.4 Training and testing for $\tau = 17$

The same experiments as reported above were repeated with the $\tau = 17$ attractor. The network used in the $\tau = 30$ trials was re-used without new optimization of the few parameters mentioned in the previous subsection.

It turned out that the $\tau = 17$ system is much simpler to cope with than the $\tau = 30$ one. It was found that no noise insertion was required to obtain stable networks, and the achieved model precision was by far greater.

Figure 14 is the dual of Fig. 12. The attractor plots were generated with plot points $(y(n), y(n+15))$ here. The three plots are visually indistinguishable.

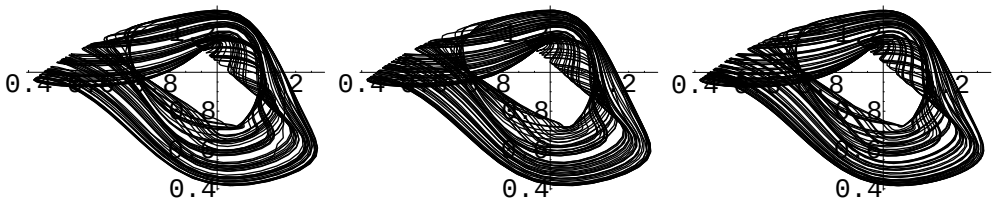

Figure 14: Case $\tau = 17$. Attractors of the original system (left), the network trained from the 21000 step sequence (right) and the network trained from the 3000 step sequence (bottom).

The $\tau = 17$ attractor seems not to have so differently divergent periods as the $\tau = 30$ attractor. In plots, predictions start to deviate perceptibly from the original not earlier than after about 1200 steps for models learnt from the 21000 step training sequence (Fig. 15).

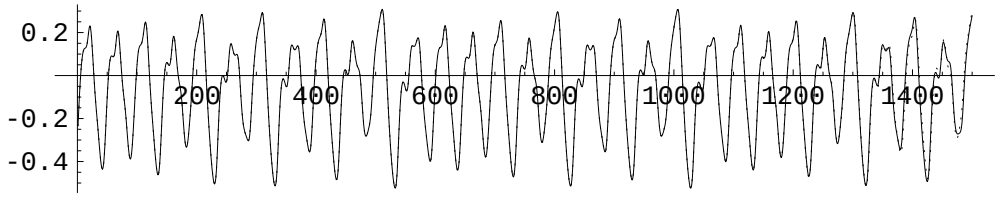

Figure 15: A 1500 step prediction with the model learnt from the 21000 step training sequence.

The prediction errors were found to be $\text{NRMSE}_{84} \approx 0.00028$ for the model learnt from the 3000 step training sequence and $\text{NRMSE}_{84} \approx 0.00012$ for the model obtained from 21000 training data points. Since the variance of prediction errors is much smaller here than in the $\tau = 30$ case, only 20 trials were averaged to estimate these errors.

For comparison we consider first the work of [18], which contains the best results from a survey given in [6]. In that approach, a local modelling technique was used (see [13] for an introduction to local modelling) in combination with a self-organizing feature map (SOM) which was used to generate prototype vectors for choosing the best local model. [18] report on numerous experiments. The best model has $\text{NRMSE}_{84} \approx 0.0088$. It was trained from about 50,000 (or 10,000, this is not clear from the paper) training data points. The SOM part of the model had 10,000 parameters (local models have no clear number of parameters because in a certain sense, the training data themselves constitute the model). The best model obtained from 6000 (possibly 1200, which is also the number of SOM parameters) data points (comparable to our model obtained from 3000 points) had $\text{NRMSE}_{84} \approx 0.032$ and the best model learnt from 28000 (or maybe 5600) points had $\text{NRMSE}_{84} \approx 0.016$.

Another good model, not contained in the survey of [6], was achieved in [12]. Unfortunately, NRMSE's are only graphically rendered – and by rather fat lines, so one can only guess an $\text{NRMSE}_{84} \approx 0.015 - 0.02$ for a model learnt from 2000 data points. Again, the methods relied on local models.

All in all, it seems fair to say that the echo state model performs about two orders of magnitude better than previous attempts to model the $\tau = 17$ Mackey-Glass system, regardless of the method. Furthermore, echo state networks are far simpler and faster to compute than other kinds of models.

7 Discussion

Recurrent networks have usually been trained by gradient descent on an error function. Several basic algorithms are known and have been refined in various directions (overviews in [3], [15], [2]). All of these methods are not completely satisfactory, because of (i) local optima, (ii) slow convergence, (iii) disruption and slowdown of the learning process when the network is driven through bifurcations during learning, and (iv) difficulties to learn long-range temporal dependencies due to vanishing or exploding gradient estimates. Furthermore, these methods are relatively complex, an obstacle to their wide-spread use.

In practice, when a dynamical system has to be realized by a neural network, one takes resort to a number of other solutions:

1. Train a feedforward network to predict the system output from network input drawn from a few delayed instances of the system input and/or output. This approach has its foundation in Takens embedding theorem and is probably the most widely used solution.

2. Use restricted recurrent network architectures for which specialized learning algorithms are known, like e.g. the well-known Elman networks (overview in [10]).

3. Use custom-designed network architectures to overcome some of the listed problems. A conspicuous example is "Long Short Term Memory" networks [7], which use specialized linear memory units with learnable read and write gates to achieve very long memory spans.

One tantalizing motive to go beyond these working solutions is biological neural networks. Biological RNNs are a proof of existence that efficient and versatile learning is possible with recurrent neural networks. While much is known about temporal learning phenomena at the synapse level (overview: [1]), not much is clear concerning the learning dynamics at the network level. In fact, the basic idea of this article (viz., that a large network can serve as a "reservoir of dynamics" which is "tapped" by trained output units) has been independently discovered in a context of biologically motivated, continuous-time, spiking neural networks [11]. That work assumes a larger variety of readout functions (while here only linear readouts with an optional subsequent transformation $\mathbf{f}^{\text{out}}$ are considered), shows that under those assumptions the class of considered networks is computationally universal in a certain sense, and demonstrates a number of biologically motivated training tasks.

I have been investigating echo state networks for about 1.5 years now and many more results than the ones reported here have been obtained. They will be reported in subsequent publications. Here is a preview:

- Echo state networks have been trained with similar ease and precision as in the above examples on a large variety of other tasks. They include tuneable sinewave generators, waveform recognizers, word recognizers, dynamical pattern recognizers, frequency measuring devices, controllers for nonlinear plants, excitable media, other chaotic attractors, and spectral analysis devices. In all cases (except excitable media modeling), the very same two 100- and 400-unit networks also employed in this article were used.

- A basic phenomenon in echo state networks is their short-term memory capability. A quantitative measure of memory capacity has been defined. Several facts about this memory capacity have been proved, the most important one being that for i.i.d. input, the memory capacity of an echo state network is bounded by the number of network units.

- The main computation for training echo state networks is a linear regression, or in other words, a mean square error minimization. This was done here offline. In the area of adaptive signal processing, numerous online methods for the same purpose have been developed. They can be used with echo state networks, too. Specifically, the LMS and the RLS algorithm (see [4] for definitions) have been tested. It turns out that LMS converges too slowly, and the reason has been elucidated. RLS works very well.

One might object that echo state networks are a waste of units. It is quite likely true that any task solved by a large, say 400-unit, echo state network can also be solved by a much smaller, say 20-unit RNN (in both, one has about 400 parameters to train). One problem, of course, is to *find* this Occamish small network – this is what previous learning techniques have attempted. But the proper reply to the waste-of-units objection is that the very same instance of a big echo state network can be re-used for a variety of tasks. For each task, only the requisite number of input and output units has to be added, and a roughly appropriate setting of the crucial global scaling parameters has to be effected for each task. This sounds natural from a biological perspective, where from all we know the same pieces of brain matter participate in many tasks, and maybe sometimes are globally modulated in task-specific ways.

Training echo state networks is both versatile (one basic network can be trained on many different tasks) and precise (training and test errors in the order of 10^{-4} for difficult tasks [like frequency measuring] to 10^{-15} for simple tasks [like periodic signal generation] were found in the tasks addressed so far). The pairing of versatility with precision springs from a single, fundamental fact. Consider Equation 12: $\mathbf{G}' \approx \sum w_i e_i$. The e_i on the rhs, written out, spell $e_i = e_i(\ldots, \mathbf{u}(n-1), \mathbf{u}(n); \ldots, \mathbf{y}(n-2), \mathbf{y}(n-1))$. These functions depend on the desired output signal $\mathbf{y}$ itself. By virtue of the echo state property, in each particular task the signals e_i will share certain properties with the desired output and/or given input, but at the same time introduce variations on the input/output theme. For instance, when the desired output is a sine with period P, all e_i will be oscillations of various forms but with same period P. Fig. 8(b) and the last row of Fig. 10 provide other striking demonstrations of this fact. So what happens is that the desired output and given input shape the "basis" systems after themselves. **The task automatically shapes the tool used for its solution.** Thus it is no really amazing that the output can be so precisely recombined from these e_i. The true miracle (which remains to be analyzed) is not the precision of the learning results, but their stability.

Mathematical analysis of echo state networks has just started. Here is a list of questions, for which presently no rigorous answers are known:

1. Under which conditions are trained networks stable? How, exactly, does noise insertion during training improve stability?

2. How can the sufficient condition from Prop. 3 (a), which is too restrictive, be relaxed to obtain a more useful sufficient condition for echo states?

3. Which task characteristics lead to which optimal values for the global tuning parameters?

4. How is the "raw" echo state best prepared in order to obtain a "rich" variety of internal dynamics?

5. What are the representational/computational limits of echo state networks? In [11] it is shown that they are computationally universal in a certain sense, but there more general readout functions than here are admitted.

Due to the basic simplicity of the approach (best expressed in the equation $\mathbf{G}' \approx \sum w_i e_i$), swift progress can reasonably be expected.

Acknowledgments I am greatly indebted to Thomas Christaller for his confidence and unfaltering support. Wolfgang Maass contributed inspiring discussions and valuable references, and pointed out an error in Prop. 3(a) in the first printed version of this report. An international patent application for the network architectures and training methods described in this paper was filed on October 13, 2000 (PCT/EP01/11490).

A Proof of proposition 1

Let

$$\begin{aligned} D \quad = \quad & \{(\mathbf{x}, \mathbf{x}') \in A^2 \,|\, \exists\, \bar{\mathbf{u}}^\infty \in\ U^{\mathbb{Z}}, \exists\, \bar{\mathbf{x}}^\infty, \bar{\mathbf{x}}'^\infty \in\ A^{\mathbb{Z}},\ \exists n \in \mathbb{Z}: \\ & \bar{\mathbf{x}}^\infty, \bar{\mathbf{x}}'^\infty \text{ compatible with } \bar{\mathbf{u}}^\infty \text{ and } \mathbf{x} = \bar{\mathbf{x}}(n) \text{ and } \mathbf{x}' = \bar{\mathbf{x}}'(n)\} \end{aligned}$$

denote the set of all state pairs that are compatible with some input sequence. It is easy to see that the echo state property is equivalent to the condition that D contain only identical pairs of the form $(\mathbf{x}, \mathbf{x})$.

We first derive an alternative characterization of D. Consider the set

$$\begin{aligned} P^+ \quad = \quad & \{(\mathbf{x}, \mathbf{x}', 1/h) \in A \times A \times [0,1] \ \mid \\ & h \in \mathbb{N},\ \exists \bar{\mathbf{u}}^h \in U^h,\ \mathbf{x} \text{ and } \mathbf{x}' \text{ are end-compatible with } \bar{\mathbf{u}}^h\}. \end{aligned} \tag{28}$$

Let D^+ be the set of all points $(\mathbf{x}, \mathbf{x}')$ such that $(\mathbf{x}, \mathbf{x}', 0)$ is an accumulation point of P^+ in the product topology of $A \times A \times [0,1]$. Note that this topology is compact and has a countable basis. We show that $D^+ = D$.

$D \subseteq D^+$: If $(\mathbf{x}, \mathbf{x}') \in D$, then $\forall h:\ (\mathbf{x}, \mathbf{x}', 1/h) \in P^+$ due to input shift invariance, hence $(\mathbf{x}, \mathbf{x}', 0)$ is an accumulation point of P^+.

$D^+ \subseteq D$: (a) From continuity of T and compactness of A, a straightforward argument shows that D^+ is closed under network update T, i.e., if $(\mathbf{x}, \mathbf{x}') \in D^+$, $\mathbf{u} \in U$, then $(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u})) \in D^+$. (b) Furthermore, it holds that for every $(\mathbf{x}, \mathbf{x}') \in D^+$, there exist $\mathbf{u} \in U$, $(\mathbf{z}, \mathbf{z}') \in D^+$ such that $(T(\mathbf{z}, \mathbf{u}), T(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$. To see this, let $\lim_{i\to\infty}(\mathbf{x}_i, \mathbf{x}'_i, 1/h_i) = (\mathbf{x}, \mathbf{x}', 0)$. For each of the $(\mathbf{x}_i, \mathbf{x}'_i)$ there exist $\mathbf{u}_i$, $(\mathbf{z}_i, \mathbf{z}'_i) \in A \times A$ such that $(T(\mathbf{z}_i, \mathbf{u}_i), T(\mathbf{z}'_i, \mathbf{u}_i)) = (\mathbf{x}_i, \mathbf{x}'_i)$. Select from the sequence $(\mathbf{z}_i, \mathbf{z}'_i, \mathbf{u}_i)$ a convergent subsequence $(\mathbf{z}_j, \mathbf{z}'_j, \mathbf{u}_j)$ (such a convergent subsequence must exist because $A \times A \times U$ is compact and has a countable topological base). Let $(\mathbf{z}, \mathbf{z}', \mathbf{u})$ be the limit of this subsequence. It holds that $(\mathbf{z}, \mathbf{z}') \in D^+$ (compactness argument) and that $(T(\mathbf{z}, \mathbf{u}), T(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$ (continuity argument about T). (c) Use (a) and (b) to conclude that for every $(\mathbf{x}, \mathbf{x}') \in D^+$

there exists an input sequence $\bar{\mathbf{u}}^{\infty}$, state sequences $\bar{\mathbf{x}}(n)^{\infty}$, $\bar{\mathbf{x}}'(n)^{\infty}$ compatible with $\bar{\mathbf{u}}^{\infty}$, and $n \in \mathbf{Z}$ such that $\mathbf{x} = \mathbf{x}(n)$ and $\mathbf{x}' = \mathbf{x}'(n)$.

With this preparation we proceed to the proof of the statements of the proposition.

"state contracting $\Rightarrow$ echo state": Assume the network has no echo states, i.e., $\exists(\mathbf{x}, \mathbf{x}') \in D^+$, $d(\mathbf{x}, \mathbf{x}') > 2\varepsilon > 0$. This implies that there exists a strictly growing sequence $(h_i)_{i\geq 0} \in \mathbb{N}^{\mathbb{N}}$, finite input sequences $(\bar{\mathbf{u}}_i^{h_i})_{i\geq 0}$, states $\mathbf{x}_i, \mathbf{x}'_i$, such that $d(T(\mathbf{x}_i, \bar{\mathbf{u}}_i^{h_i}), \mathbf{x}) < \varepsilon$ and $d(T(\mathbf{x}'_i, \bar{\mathbf{u}}_i^{h_i}), \mathbf{x}') < \varepsilon$. Complete every $\mathbf{u}_i^{h_i}$ on the right with an arbitrary right-infinite input sequence $\bar{\mathbf{v}}^{+\infty}$ to obtain a sequence $(\bar{\mathbf{u}}_i^{h_i}\bar{\mathbf{v}}^{+\infty})_{i\geq 0} \in (U^{\mathbf{N}})^{\mathbf{N}}$. By the theorem of Tychonov known in topology, $U^{\mathbf{N}}$ equipped with the product topology is compact. Furthermore, this topology of $U^{\mathbf{N}}$ has a countable basis. This implies that every sequence in $U^{\mathbf{N}}$ has a convergent subsequence. Use this to establish that there exists a subsequence $(h_{i_j})_{j\geq 0}$ of $(h_i)_{i\geq 0}$ such that $(\bar{\mathbf{u}}_{i_j}^{h_{i_j}}\bar{\mathbf{v}}^{+\infty})_{j\geq 0}$ converges to an input sequence $\bar{\mathbf{u}}^{+\infty}$. For yet a further suitable subsequence $(h_{i_{j_k}})_{k\geq 0}$ of $(h_{i_j})_{j\geq 0}$, there exist states $\mathbf{x}_k, \mathbf{x}'_k$, such that $d(T(\mathbf{x}_k, \bar{\mathbf{u}}^{+\infty}[h_{i_{j_k}}]), \mathbf{x}) < \varepsilon$ and $d(T(\mathbf{x}'_k, \bar{\mathbf{u}}^{+\infty}[h_{i_{j_k}}]), \mathbf{x}') < \varepsilon$, where $\bar{\mathbf{u}}^{+\infty}[h_{i_{j_k}}]$ is the prefix of length $h_{i_{j_k}}$ of $\bar{\mathbf{u}}^{+\infty}$. Since $d(\mathbf{x}, \mathbf{x}') > 2\varepsilon$, this contradicts the state contracting property.

"echo state $\Rightarrow$ state contracting": Assume that the network is not state contracting. This implies that there exist an input sequence $\bar{\mathbf{u}}^{+\infty}$, a strictly growing index sequence $(h_i)_{i\geq 0}$, states $\mathbf{x}_i, \mathbf{x}'_i$, and some $\varepsilon > 0$, such that $\forall i : d(T(\mathbf{x}_i, \bar{\mathbf{u}}^{+\infty}[h_i]), T(\mathbf{x}'_i, \bar{\mathbf{u}}^{+\infty}[h_i])) > \varepsilon$. By compactness of $A \times A \times [0, 1]$, this implies that there exists some $(\mathbf{x}, \mathbf{x}') \in D^+$ with $d(\mathbf{x}, \mathbf{x}') > \varepsilon$. Since $D^+ \subseteq D$, this implies that the network has not the echo state property.

" state contracting $\Rightarrow$ state forgetting": Assume the network is not state forgetting. This implies that there exists a left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$, a strictly growing index sequence $(h_i)_{i\geq 0}$, states $\mathbf{x}_i, \mathbf{x}'_i$, and some $\varepsilon > 0$, such that $\forall i : d(T(\mathbf{x}_i, \bar{\mathbf{u}}^{+\infty}[-h_i]), T(\mathbf{x}'_i, \bar{\mathbf{u}}^{+\infty}[-h_i])) > \varepsilon$, where $\bar{\mathbf{u}}^{+\infty}[-h_i]$ denotes the suffix of lenght h_i of $\bar{\mathbf{u}}^{+\infty}$. Complete every $\bar{\mathbf{u}}^{+\infty}[-h_i]$ on the right with an arbitrary right-infinite input sequence and repeat the argument from the case "state contracting $\Rightarrow$ echo state" to derive a contradiction to the state forgetting property.

"state contracting $\Rightarrow$ input forgetting": trivial.

"input forgetting $\Rightarrow$ echo state": Assume that the network does not have the echo state property. Then there exists an input sequence $\bar{\mathbf{u}}^{-\infty}$, states $\mathbf{x}, \mathbf{x}'$ end-compatible with $\bar{\mathbf{u}}^{-\infty}$, $d(\mathbf{x}, \mathbf{x}') > 0$. This leads straightforwardly to a contradiction to input forgetting.

B Proof of proposition 2

Let $\bar{\mathbf{u}}^{-\infty}$ be an input sequence, let $\varepsilon > 0$. Using the echo state property, we have to provide δ and h according to the proposition. We use the "input forgetting" characterization of echo states. Let $(\delta_h)_{h\geq 0}$ be the null sequence connected with $\bar{\mathbf{u}}^{-\infty}$ according to Def. 4(*3.*). Then there exists some h such that for all $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h, \bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$, for all $\mathbf{x}, \mathbf{x}'$ end-compatible with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$, $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$, it holds that $d(\mathbf{x}, \mathbf{x}') < \varepsilon/2$. Call $\bar{\mathbf{u}}_h$ *δ-close* to $\bar{\mathbf{u}}'_h$ if $d(\mathbf{u}(k), \mathbf{u}'(k)) < \delta$ for all $-h \leq k \leq 0$. By continuity of T, there exists some δ such that for all $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h, \bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}'_h$, where $\bar{\mathbf{u}}_h$ is δ-close to $\bar{\mathbf{u}}'_h$, for all $\mathbf{x}, \mathbf{x}'$ end-compatible with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$, $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}'_h$, it holds that $d(\mathbf{x}, \mathbf{x}') < \varepsilon$.

C Proof of proposition 3

"(a)":

$$\begin{aligned} & d(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u}) = \\ &= \; d(\mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x}), \mathbf{f}(\mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x}')) \\ &\leq \; d(\mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x}, \mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x}') \\ &= \; d(\mathbf{W}\mathbf{x}, \mathbf{W}\mathbf{x}') \\ &= \; \|\mathbf{W}(\mathbf{x} - \mathbf{x})\| \\ &\leq \; \Lambda d(\mathbf{x}, \mathbf{x}'), \end{aligned}$$

i.e., the distance between two states $\mathbf{x}, \mathbf{x}'$ shrinks by a factor $\Lambda < 1$ at every step, regardless of the input. This Lipschitz condition obviously results in echo states.

"(b)": Consider the left-infinite null input sequence $\bar{\mathbf{0}}^{-\infty} \in U^{-\mathbb{N}}$. The null state sequence $\bar{\mathbf{0}}^{-\infty} \in A^{-\mathbb{N}}$ is compatible with the null input sequence. But if $|\lambda_{\max}| > 1$, the null state $\mathbf{0} \in \mathbf{R}^N$ is not asymptotically stable (see e.g. [8], Exercise 3.52), which implies the existence of another state sequence $\bar{\mathbf{x}}^{-\infty} \neq \bar{\mathbf{0}}^{-\infty}$ compatible with the null input sequence. This violates the echo state property.

D Proof of proposition 4

"(a)": Observe the general fact that if F_i are functions with global Lipschitz constants L_i, then $\sum_i F_i$ is globally Lipschitz with Lipschitz constant $\sum_i L_i$. Observing that the rhs of (21) is a sum of two functions. The first has a global Lipschitz constant of $L_1 = 1 - \delta Ca$. Argue like in the proof of Prop. 3(a)

that the second has a global Lipschitz constant $L_2 = \delta C \sigma_{\max}$, where $\sigma_{\max}$ is the largest eigenvalue of $\mathbf{W}$. Observe that $L_1 + L_2 = 1 - \delta C(a - \sigma_{\max})$.

"(b)": With null input $\mathbf{u}(n) = \mathbf{0}$, the null state (and output) sequence is a solution of (21). To check its asymptotic stability, consider the system (21) linearized in the origin. It is governed by $\mathbf{x}(n+1) = (\delta C \mathbf{W} + (1 - \delta C a)\mathbf{id})\,\mathbf{x}(n)$. If the matrix $\delta C \mathbf{W} + (1 - \delta C a)\mathbf{id}$ has spectral radius greater than 1, the linear system is not asymptotically stable. This implies the existence of another solution of (21) besides the null solution, which in turn means that the network has no echo states for any input set containing $\mathbf{0}$ and admissible state set $[-1, 1]^N$.

E Erratum

E.1 Corrected version of Definition 3

The version of Definition 3 in the original techreport provided three properties that were claimed in Proposition 1 to be all equivalent with the echo state property. However, the first property was too weak. Here a corrected version of this definition is given. The only change is in the statement of property 1. It was called the *state contracting* property in the original techreport. Tobias Strauss in [17] calls the corrected version the *uniformly state contracting* property, a terminology that I would want to adopt (and dismiss the old name altogether with its defunct definition).

Definition 4 *Assume standard compactness conditions and a network without output feedback.*

1. [Corrected] *The network is called* uniformly state contracting *if there exists a null sequence $(\delta_h)_{h \geq 0}$ such that for all right-infinite input sequences $\bar{\mathbf{u}}^{+\infty}$, and for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$, for all input sequence prefixes $\bar{\mathbf{u}}_h = \mathbf{u}(n), \ldots, \mathbf{u}(n+h)$ it holds that $d(T(\mathbf{x}, \bar{\mathbf{u}}_h), T(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$, where d is the Euclidean distance on $\mathbb{R}^N$.*

2. *The network is called* state forgetting *if for all left-infinite input sequences $\ldots, \mathbf{u}(n-1), \mathbf{u}(n) \in U^{-\mathbb{N}}$ there exists a null sequence $(\delta_h)_{h \geq 0}$ such that for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$, for all input sequence suffixes $\bar{\mathbf{u}}_h = \mathbf{u}(n-h), \ldots, \mathbf{u}(n)$ it holds that $d(T(\mathbf{x}, \bar{\mathbf{u}}_h), T(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$.*

3. *The network is called* input forgetting *if for all left-infinite input sequences $\bar{\mathbf{u}}^{-\infty}$ there exists a null sequence $(\delta_h)_{h \geq 0}$ such that for all $h \geq 0$,*

for all input sequence suffixes $\bar{\mathbf{u}}_h = \mathbf{u}(n-h), \ldots, \mathbf{u}(n)$*, for all left-infinite input sequences of the form* $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h, \bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$*, for all states* $\mathbf{x}$ *end-compatible with* $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ *and states* $\mathbf{x}'$ *end-compatible with* $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ *it holds that* $d(\mathbf{x}, \mathbf{x}') < \delta_h$*.*

The following identically re-states Proposition 1 from the original version of the techreport, except that *state contracting* has been changed to *uniformly state contracting.*

Proposition 5 *Assume standard compactness conditions and a network without output feedback. Assume that* T *is continuous in state and input. Then the properties of being uniformly state contracting, state forgetting, and input forgetting are all equivalent to the network having echo states.*

The following proof of Prop. 5 by and large replicates the original proof from the techreport, except a re-arrangement, some completions and adding a proof for the implication *uniformly state contracting* $\Rightarrow$ *echo states.*

Proof.

Part 1: *echo states* $\Rightarrow$ *uniformly state contracting.*

Let

$$\begin{aligned} D \quad = \quad & \{(\mathbf{x}, \mathbf{x}') \in A^2 \,|\, \exists\, \bar{\mathbf{u}}^{\infty} \in\ U^{\mathbb{Z}}, \exists\, \bar{\mathbf{x}}^{\infty}, \bar{\mathbf{x}}'^{\infty} \in\ A^{\mathbb{Z}},\ \exists n \in \mathbb{Z} : \\ & \bar{\mathbf{x}}^{\infty}, \bar{\mathbf{x}}'^{\infty} \text{ compatible with } \bar{\mathbf{u}}^{\infty} \text{ and } \mathbf{x} = \bar{\mathbf{x}}(n) \text{ and } \mathbf{x}' = \bar{\mathbf{x}}'(n)\} \end{aligned}$$

denote the set of all state pairs that are compatible with some input sequence. It is easy to see that the echo state property is equivalent to the condition that D contain only identical pairs of the form $(\mathbf{x}, \mathbf{x})$.

Like in the original techreport, we first derive an alternative characterization of D. Consider the set

$$\begin{aligned} P^{+} \quad = \quad & \{(\mathbf{x}, \mathbf{x}', 1/h) \in A \times A \times [0,1] \ \mid \\ & h \in \mathbb{N},\ \exists \bar{\mathbf{u}}^h \in U^h,\ \mathbf{x} \text{ and } \mathbf{x}' \text{ are end-compatible with } \bar{\mathbf{u}}^h\}. \end{aligned}$$

Let D^{+} be the set of all points $(\mathbf{x}, \mathbf{x}')$ such that $(\mathbf{x}, \mathbf{x}', 0)$ is an accumulation point of P^{+} in the product topology of $A \times A \times [0,1]$. Note that this topology is compact and has a countable basis. We show that $D^{+} = D$.

$D \subseteq D^{+}$: If $(\mathbf{x}, \mathbf{x}') \in D$, then $\forall h :\ (\mathbf{x}, \mathbf{x}', 1/h) \in P^{+}$ due to input shift invariance, hence $(\mathbf{x}, \mathbf{x}', 0)$ is an accumulation point of P^{+} .

$D^+ \subseteq D$: (a) From continuity of T and compactness of A, a straightforward argument shows that D^+ is closed under network update T, i.e., if $(\mathbf{x}, \mathbf{x}') \in D^+$, $\mathbf{u} \in U$, then $(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u})) \in D^+$. (b) Furthermore, it holds that for every $(\mathbf{x}, \mathbf{x}') \in D^+$, there exist $\mathbf{u} \in U$, $(\mathbf{z}, \mathbf{z}') \in D^+$ such that $(T(\mathbf{z}, \mathbf{u}), T(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$. To see this, let $\lim_{i\to\infty}(\mathbf{x}_i, \mathbf{x}'_i, 1/h_i) = (\mathbf{x}, \mathbf{x}', 0)$. For each of the $(\mathbf{x}_i, \mathbf{x}'_i)$ there exist $\mathbf{u}_i$, $(\mathbf{z}_i, \mathbf{z}'_i) \in A \times A$ such that $(T(\mathbf{z}_i, \mathbf{u}_i), T(\mathbf{z}'_i, \mathbf{u}_i)) = (\mathbf{x}_i, \mathbf{x}'_i)$. Select from the sequence $(\mathbf{z}_i, \mathbf{z}'_i, \mathbf{u}_i)$ a convergent subsequence $(\mathbf{z}_j, \mathbf{z}'_j, \mathbf{u}_j)$ (such a convergent subsequence must exist because $A \times A \times U$ is compact and has a countable topological base). Let $(\mathbf{z}, \mathbf{z}', \mathbf{u})$ be the limit of this subsequence. It holds that $(\mathbf{z}, \mathbf{z}') \in D^+$ (compactness argument) and that $(T(\mathbf{z}, \mathbf{u}), T(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$ (continuity argument about T). (c) Use (a) and (b) to conclude that for every $(\mathbf{x}, \mathbf{x}') \in D^+$ there exists an input sequence $\bar{\mathbf{u}}^\infty$, state sequences $\bar{\mathbf{x}}(n)^\infty$, $\bar{\mathbf{x}}'(n)^\infty$ compatible with $\bar{\mathbf{u}}^\infty$, and $n \in \mathbb{Z}$ such that $\mathbf{x} = \mathbf{x}(n)$ and $\mathbf{x}' = \mathbf{x}'(n)$.

With this preparation we proceed to the proof of *echo states* $\Rightarrow$ *uniformly state contracting*, repeating (and translating to English) the argument given by Tobias Strauss.

Assume the network is not uniformly state contracting. This implies that for every null sequence $(\delta_i)_{i\geq 0}$ there exists a $h \geq 0$, an input sequence $\bar{\mathbf{u}}_h$ of length h, and states $\mathbf{x}, \mathbf{x}' \in A$, such that

$$d(T(\mathbf{x}, \bar{\mathbf{u}}_h), T(\mathbf{x}', \bar{\mathbf{u}}_h)) \geq \delta_h.$$

Since A is compact, it is bounded. Therefore, the sequence $(\mu_i)_{i\geq 0}$ defined by

$$\mu_i := \sup\{d(T(\mathbf{x}, \bar{\mathbf{u}}_i), T(\mathbf{x}', \bar{\mathbf{u}}_i)) \mid \mathbf{x}, \mathbf{x}' \in A, \bar{\mathbf{u}}_i \in U^i\}$$

is bounded, say by M. Because we assumed that the network is not uniformly state contracting, $(\mu_i)_{i\geq 0}$ is not a null sequence. Therefore there exists a subsequence $(\mu_{i_j})_{j\geq 0}$ of $(\mu_i)_{i\geq 0}$, which converges to some $\varepsilon > 0$. Since for all i, the space $U^i \times A$ is compact and $T : U^i \times A \to A$ is continuous, the supremum μ_i is realized by suitable $\mathbf{x}, \mathbf{x}' \in A$. Let $(\mathbf{x}_{i_j}, \mathbf{x}'_{i_j}) \in A^2$ be such that

$$\begin{aligned}(\mathbf{x}_{i_j}, \mathbf{x}'_{i_j}) \in \{&(T(\mathbf{x}, \bar{\mathbf{u}}_{i_j}), T(\mathbf{x}', \bar{\mathbf{u}}_{i_j})) \mid \\ &\bar{\mathbf{u}}_{i_j} \in U^{i_j}, \mathbf{x}, \mathbf{x}' \in A, d(T(\mathbf{x}, \bar{\mathbf{u}}_{i_j}), T(\mathbf{x}', \bar{\mathbf{u}}_{i_j})) = \mu_{i_j}\}.\end{aligned}$$

Since A^2 is compact, there exist a subsequence $(\mathbf{x}_{i_{j_k}}, \mathbf{x}'_{i_{j_k}})_{k\geq 0}$ of $(\mathbf{x}_{i_j}, \mathbf{x}'_{i_j})_{j\geq 0}$ which converges to some $(\mathbf{y}, \mathbf{y}') \in A^2$. Obviously it holds that $(\mathbf{x}_{i_j}, \mathbf{x}'_{i_j}, \frac{1}{i_j}) \in$

P^+. Thus $(\mathbf{y}, \mathbf{y}', 0)$ is an accumulation point of P^+, i.e., $(\mathbf{y}, \mathbf{y}') \in D^+$. On the other hand,

$$0 < \varepsilon = \lim_{k\to\infty} \mu_{i_{j_k}} = \lim_{k\to\infty} d(\mathbf{x}_{i_j} - \mathbf{x}'_{i_j}) = d(\mathbf{y}, \mathbf{y}').$$

This contradicts the echo state property, because D^+ does not contain pairs $(\mathbf{y}, \mathbf{y}')$ with $\mathbf{y} \neq \mathbf{y}'$.

Part 2: *uniformly state contracting* $\Rightarrow$ *state forgetting*.

Assume the network is not state forgetting. This implies that there exists a left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$, a strictly growing index sequence $(h_i)_{i\geq 0}$, states $\mathbf{x}_i, \mathbf{x}'_i$, and some $\varepsilon > 0$, such that

$$\forall i : d(T(\mathbf{x}_i, \bar{\mathbf{u}}^{-\infty}[h_i]), T(\mathbf{x}'_i, \bar{\mathbf{u}}^{-\infty}[h_i])) > \varepsilon,$$

where $\bar{\mathbf{u}}^{-\infty}[h_i]$ denotes the suffix of lenght h_i of $\bar{\mathbf{u}}^{-\infty}$. Complete every $\bar{\mathbf{u}}^{-\infty}[h_i]$ on the right with an arbitrary right-infinite input sequence, to get a series of right-infinite input sequences $(\bar{\mathbf{v}}_i)_{i=1,2,\ldots}$. For the i-th series $\bar{\mathbf{v}}_i$ it holds that $d(T(\mathbf{x}_i, \bar{\mathbf{v}}_i[h_i], T(\mathbf{x}'_i, \bar{\mathbf{v}}_i[h_i])) > \varepsilon$, where $\bar{\mathbf{v}}_i[h_i]$ is the prefix of length h_i of $\bar{\mathbf{v}}_i$, which contradicts the uniform state contraction property.

Part 3: *state forgetting* $\Rightarrow$ *input forgetting*.

Let $\bar{\mathbf{u}}^{-\infty}$ be a left-infinite input sequence, and $(\delta_h)_{h\geq 0}$ be an associated null sequence according to the state forgetting property. For the suffix $\bar{\mathbf{u}}_h$ of length h of $\bar{\mathbf{u}}^{-\infty}$, consider any pair $\mathbf{y}, \mathbf{y}'$ of states from A. By the state forgetting property it holds that $d(T(\mathbf{y}, \bar{\mathbf{u}}_h), T(\mathbf{y}', \bar{\mathbf{u}}_h)) < \delta_h$. Now consider any left-infinite $\bar{\mathbf{w}}^{-\infty}$ and $\bar{\mathbf{v}}^{-\infty}$. If, specifically, $\mathbf{y}, \mathbf{y}'$ are end-compatible with $\bar{\mathbf{w}}^{-\infty}$ and $\bar{\mathbf{v}}^{-\infty}$, respectively, it still holds that $d(T(\mathbf{y}, \bar{\mathbf{u}}_h), T(\mathbf{y}', \bar{\mathbf{u}}_h)) < \delta_h$. This implies that for all states $\mathbf{x}$ and $\mathbf{x}'$ which are end-compatible with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ and $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$, respectively, it holds that $d(\mathbf{x}, \mathbf{x}') < \delta_h$.

Part 4: *input forgetting* $\Rightarrow$ *echo states*.

Assume that the network does not have the echo state property. Then there exists a left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$, states $\mathbf{x}, \mathbf{x}'$ end-compatible with $\bar{\mathbf{u}}^{-\infty}$, $d(\mathbf{x}$, such that $\mathbf{x}') > 0$. This leads immediately to a contradiction to input forgetting, by setting $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h = \bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h = \bar{\mathbf{u}}^{-\infty}$.

References

[1] L.F. Abbott and S.B. Nelson. Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3(supplement):1178–1183, 2000.

[2] A.F. Atiya and A.G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Trans. Neural Networks*, 11(3):697–709, 2000.

[3] K. Doya. Recurrent neural networks: Supervised learning. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 796–800. MIT Press / Bradford Books, 1995.

[4] B. Farhang-Boroujeny. *Adaptive Filters: Theory and Applications*. Wiley, 1998.

[5] L.A. Feldkamp, D.V. Prokhorov, C.F. Eagen, and F. Yuan. Enhanced multi-stream Kalman filter training for recurrent neural networks. In J.A.K. Suykens and J. Vandewalle, editors, *Nonlinear Modeling: Advanced Black-Box Techniques*, pages 29–54. Kluwer, 1998.

[6] F. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. Technical report IDSIA-IDSIA-22-00, IDSI/USI-SUPSI, Instituto Dalle Molle di studi sull' intelligenza artificiale, Manno, Switzerland, 2000. http://www.idsia.ch/∼felix/Publications.html.

[7] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[8] H. K. Khalil. *Nonlinear Systems (second edition)*. Prentice Hall, 1996.

[9] M. Kimura and R. Nakano. Learning dynamical systems by recurrent neural networks from orbits. *Neural Networks*, 11(9):1589–1600, 1998.

[10] S.C. Kremer. Spatiotemporal connectionist neural networks: a taxonomy and review. *Neural Computation*, 13:249–306, 2001.

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

[12] J McNames. *Innovations in local modeling for time series prediction*. Phd thesis, Dept. of Electrical Engineering, Stanford University, 1999. www.ee.pdx.edu/∼mcnames/Publications/Dissertation.pdf.

[13] J. McNames. Local modeling optimization for time series prediction. In *Proc. 8th European Symposium on Artificial Neural Networks*, 2000. http://www.ee.pdx.edu/∼mcnames/Publications/ESANN2000.pdf.

[14] K.S. Narendra and S. Mukhopadhyay. Adaptive control using neural networks and approximate models. *IEEE Transactions on Neural Networks*, 8(3):475–485, 1997.

[15] B.A. Pearlmutter. Gradient calculation for dynamic recurrent neural networks: a survey. *IEEE Trans. on Neural Networks*, 6(5):1212–1228, 1995. http://www.bcl.hamilton.ie/∼bap/papers/ieee-dynnn-draft.ps.gz.

[16] J. Stark, D.S. Broomhead, M.E. Davies, and J. Huke. Takens embedding theorems for forced and stochastic systems. *Nonlinear Analysis, Theory, Methods & Applications*, 30(8):5303–5314, 1997.

[17] T. Strauss. *Alternative Konvergenzmaße für die Beschreibung des Verhaltens von Echo-State-Netzen*. Diplomarbeit, Math.-Naturwissenschaftliche Fakultät, Institut für Mathematik, Universität Rostock, 2009.

[18] J. Vesanto. Using the SOM and local models in time-series prediction. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, 1997. http://www.cis.hut.fi/projects/monitor/publications/papers/wsom97.ps.zip.

Original techreport received and put to printing Dec. 6, 2001. Erratum note added Jan 25, 2010.